MySQL 8.0 Reference Manual

Including MySQL NDB Cluster 8.0

Abstract

This is the MySQL™ Reference Manual. It documents MySQL 8.0 through 8.0.19, as well as NDB Cluster releases based on version 8.0 of NDB through 8.0.19-ndb-8.0.19, respectively. It may include documentation of features of MySQL versions that have not yet been released. For information about which versions have been released, see the MySQL 8.0 Release Notes.

MySQL 8.0 features. This manual describes features that are not included in every edition of MySQL 8.0; such features may not be included in the edition of MySQL 8.0 licensed to you. If you have any questions about the features included in your edition of MySQL 8.0, refer to your MySQL 8.0 license agreement or contact your Oracle sales representative.

For notes detailing the changes in each release, see the MySQL 8.0 Release Notes.

For legal information, including licensing information, see the Preface and Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2019-09-22 (revision: 63550)

Table of Contents

Preface and Legal Notices xxix
1 General Information 1
1.1 About This Manual 2
1.2 Typographical and Syntax Conventions 3
1.3 Overview of the MySQL Database Management System 4
1.3.1 What is MySQL? 4
1.3.2 The Main Features of MySQL 6
1.3.3 History of MySQL 9
1.4 What Is New in MySQL 8.0 9
1.5 Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 8.0 36
1.6 MySQL Information Sources 51
1.6.1 MySQL Websites 51
1.6.2 MySQL Community Support at the MySQL Forums 51
1.6.3 MySQL Enterprise 52
1.7 How to Report Bugs or Problems 52
1.8 MySQL Standards Compliance 56
1.8.1 MySQL Extensions to Standard SQL 58
1.8.2 MySQL Differences from Standard SQL 60
1.8.3 How MySQL Deals with Constraints 62
1.9 Credits 66
1.9.1 Contributors to MySQL 66
1.9.2 Documenters and translators 70
1.9.3 Packages that support MySQL 72
1.9.4 Tools that were used to create MySQL 72
1.9.5 Supporters of MySQL 73
2 Installing and Upgrading MySQL 75
2.1 General Installation Guidance 78
2.1.1 Which MySQL Version and Distribution to Install 78
2.1.2 How to Get MySQL 79
2.1.3 Verifying Package Integrity Using MD5 Checksums or GnuPG 79
2.1.4 Installation Layouts 93
2.1.5 Compiler-Specific Build Characteristics 93
2.2 Installing MySQL on Unix/Linux Using Generic Binaries 93
2.3 Installing MySQL on Microsoft Windows 96
2.3.1 MySQL Installation Layout on Microsoft Windows 99
2.3.2 Choosing an Installation Package 100
2.3.3 MySQL Installer for Windows 101
2.3.4 Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive 127
2.3.5 Troubleshooting a Microsoft Windows MySQL Server Installation 136
2.3.6 Windows Postinstallation Procedures 137
2.3.7 Windows Platform Restrictions 140
2.4 Installing MySQL on macOS 141
2.4.1 General Notes on Installing MySQL on macOS 142
2.4.2 Installing MySQL on macOS Using Native Packages 143
2.4.3 Installing and Using the MySQL Launch Daemon 147
2.4.4 Installing and Using the MySQL Preference Pane 150
2.5 Installing MySQL on Linux 154
2.5.1 Installing MySQL on Linux Using the MySQL Yum Repository 155
2.5.2 Installing MySQL on Linux Using the MySQL APT Repository 160
2.5.3 Installing MySQL on Linux Using the MySQL SLES Repository 160
2.5.4 Installing MySQL on Linux Using RPM Packages from Oracle 160

2.5.5 Installing MySQL on Linux Using Debian Packages from Oracle 165
2.5.6 Deploying MySQL on Linux with Docker 167
2.5.7 Installing MySQL on Linux from the Native Software Repositories 177
2.5.8 Installing MySQL on Linux with Juju 181
2.5.9 Managing MySQL Server with systemd 181
2.6 Installing MySQL Using Unbreakable Linux Network (ULN) 186
2.7 Installing MySQL on Solaris 187
2.7.1 Installing MySQL on Solaris Using a Solaris PKG 188
2.8 Installing MySQL on FreeBSD 189
2.9 Installing MySQL from Source 190
2.9.1 Source Installation Methods 190
2.9.2 Source Installation Prerequisites 191
2.9.3 MySQL Layout for Source Installation 192
2.9.4 Installing MySQL Using a Standard Source Distribution 193
2.9.5 Installing MySQL Using a Development Source Tree 197
2.9.6 Configuring SSL Library Support 199
2.9.7 MySQL Source-Configuration Options 200
2.9.8 Dealing with Problems Compiling MySQL 225
2.9.9 MySQL Configuration and Third-Party Tools 227
2.9.10 Generating MySQL Doxygen Documentation Content 227
2.10 Postinstallation Setup and Testing 228
2.10.1 Initializing the Data Directory 229
2.10.2 Starting the Server 235
2.10.3 Testing the Server 238
2.10.4 Securing the Initial MySQL Account 240
2.10.5 Starting and Stopping MySQL Automatically 242
2.11 Upgrading MySQL 243
2.11.1 Before You Begin 243
2.11.2 Upgrade Paths 244
2.11.3 What the MySQL Upgrade Process Upgrades 245
2.11.4 Changes in MySQL 8.0 248
2.11.5 Preparing Your Installation for Upgrade 259
2.11.6 Upgrading MySQL Binary or Package-based Installations on Unix/Linux 262
2.11.7 Upgrading MySQL with the MySQL Yum Repository 267
2.11.8 Upgrading MySQL with the MySQL APT Repository 268
2.11.9 Upgrading MySQL with the MySQL SLES Repository 269
2.11.10 Upgrading MySQL on Windows 269
2.11.11 Upgrading a Docker Installation of MySQL 270
2.11.12 Upgrade Troubleshooting 271
2.11.13 Rebuilding or Repairing Tables or Indexes 271
2.11.14 Copying MySQL Databases to Another Machine 272
2.12 Downgrading MySQL 274
2.13 Perl Installation Notes 274
2.13.1 Installing Perl on Unix 274
2.13.2 Installing ActiveState Perl on Windows 275
2.13.3 Problems Using the Perl DBI/DBD Interface 276
3 Tutorial 277
3.1 Connecting to and Disconnecting from the Server 277
3.2 Entering Queries 278
3.3 Creating and Using a Database 282
3.3.1 Creating and Selecting a Database 283
3.3.2 Creating a Table 284
3.3.3 Loading Data into a Table 285
3.3.4 Retrieving Information from a Table 286

3.4 Getting Information About Databases and Tables 300
3.5 Using mysql in Batch Mode 301
3.6 Examples of Common Queries 303
3.6.1 The Maximum Value for a Column 304
3.6.2 The Row Holding the Maximum of a Certain Column 304
3.6.3 Maximum of Column per Group 304
3.6.4 The Rows Holding the Group-wise Maximum of a Certain Column 305
3.6.5 Using User-Defined Variables 306
3.6.6 Using Foreign Keys 306
3.6.7 Searching on Two Keys 308
3.6.8 Calculating Visits Per Day 308
3.6.9 Using AUTO_INCREMENT 309
3.7 Using MySQL with Apache 311
4 MySQL Programs 313
4.1 Overview of MySQL Programs 314
4.2 Using MySQL Programs 317
4.2.1 Invoking MySQL Programs 317
4.2.2 Specifying Program Options 318
4.2.3 Command Options for Connecting to the Server 332
4.2.4 Connecting to the MySQL Server Using Command Options 340
4.2.5 Connecting to the Server Using URI-Like Strings or Key-Value Pairs 343
4.2.6 Connection Compression Control 349
4.2.7 Setting Environment Variables 353
4.3 MySQL Server and Server-Startup Programs 354
4.3.1 `mysqld` — The MySQL Server 354
4.3.2 `mysqld_safe` — MySQL Server Startup Script 355
4.3.3 `mysql.server` — MySQL Server Startup Script 361
4.3.4 `mysqld_multi` — Manage Multiple MySQL Servers 363
4.4 MySQL Installation-Related Programs 367
4.4.1 `comp_err` — Compile MySQL Error Message File 367
4.4.2 `mysql_secure_installation` — Improve MySQL Installation Security 368
4.4.3 `mysql_ssl_rsa_setup` — Create SSL/RSA Files 372
4.4.4 `mysql_tzinfo_to_sql` — Load the Time Zone Tables 375
4.4.5 `mysql_upgrade` — Check and Upgrade MySQL Tables 376
4.5 MySQL Client Programs 385
4.5.1 `mysql` — The MySQL Command-Line Client 385
4.5.2 `mysqladmin` — Client for Administering a MySQL Server 415
4.5.3 `mysqlcheck` — A Table Maintenance Program 426
4.5.4 `mysqldump` — A Database Backup Program 436
4.5.5 `mysqlimport` — A Data Import Program 461
4.5.6 `mysqlpump` — A Database Backup Program 470
4.5.7 `mysqlshow` — Display Database, Table, and Column Information 487
4.5.8 `mysqlslap` — Load Emulation Client 495
4.6 MySQL Administrative and Utility Programs 505
4.6.1 `ibd2sdi` — InnoDB Tablespace SDI Extraction Utility 505
4.6.2 `innochecksum` — Offline InnoDB File Checksum Utility 509
4.6.3 `myisam_ftdump` — Display Full-Text Index information 514
4.6.4 `myisamchk` — MyISAM Table-Maintenance Utility 516
4.6.5 `myisamlog` — Display MyISAM Log File Contents 533
4.6.6 `myisampack` — Generate Compressed, Read-Only MyISAM Tables 534
4.6.7 `mysql_config_editor` — MySQL Configuration Utility 540
4.6.8 `mysqlbinlog` — Utility for Processing Binary Log Files 547
4.6.9 `mysqldumpslow` — Summarize Slow Query Log Files 571
4.7 MySQL Program Development Utilities 573

4.7.1 mysql_config — Display Options for Compiling Clients 573
4.7.2 my_print_defaults — Display Options from Option Files 575
4.8 Miscellaneous Programs 576
4.8.1 lz4_decompress — Decompress mysqlpump LZ4-Compressed Output 576
4.8.2 perror — Display MySQL Error Message Information 577
4.8.3 zlib_decompress — Decompress mysqlpump ZLIB-Compressed Output 578
4.9 MySQL Environment Variables 578
5 MySQL Server Administration 581
5.1 The MySQL Server 582
5.1.1 Configuring the Server 582
5.1.2 Server Configuration Defaults 584
5.1.3 Server Configuration Validation 584
5.1.4 Server Option, System Variable, and Status Variable Reference 585
5.1.5 Server System Variable Reference 632
5.1.6 Server Status Variable Reference 657
5.1.7 Server Command Options 673
5.1.8 Server System Variables 704
5.1.9 Using System Variables 861
5.1.10 Server Status Variables 892
5.1.11 Server SQL Modes 914
5.1.12 IPv6 Support 926
5.1.13 MySQL Server Time Zone Support 930
5.1.14 Server-Side Help Support 936
5.1.15 Server Tracking of Client Session State Changes 936
5.1.16 Server Response to Signals 940
5.1.17 The Server Shutdown Process 940
5.2 The MySQL Data Directory 942
5.3 The mysql System Schema 943
5.4 MySQL Server Logs 948
5.4.1 Selecting General Query Log and Slow Query Log Output Destinations 949
5.4.2 The Error Log 951
5.4.3 The General Query Log 969
5.4.4 The Binary Log 971
5.4.5 The Slow Query Log 984
5.4.6 The DDL Log 987
5.4.7 Server Log Maintenance 988
5.5 MySQL Server Components 989
5.5.1 Installing and Uninstalling Components 990
5.5.2 Obtaining Server Component Information 991
5.5.3 Error Log Components 991
5.6 MySQL Server Plugins 993
5.6.1 Installing and Uninstalling Plugins 994
5.6.2 Obtaining Server Plugin Information 998
5.6.3 MySQL Enterprise Thread Pool 999
5.6.4 The Rewriter Query Rewrite Plugin 1006
5.6.5 The ddl_rewriter Plugin 1016
5.6.6 Version Tokens 1018
5.6.7 The Clone Plugin 1030
5.7 MySQL Server User-Defined Functions 1054
5.7.1 Installing and Uninstalling User-Defined Functions 1054
5.7.2 Obtaining User-Defined Function Information 1055
5.8 Running Multiple MySQL Instances on One Machine 1055
5.8.1 Setting Up Multiple Data Directories 1057
5.8.2 Running Multiple MySQL Instances on Windows 1058

5.8.3 Running Multiple MySQL Instances on Unix 1061
5.8.4 Using Client Programs in a Multiple-Server Environment 1062
6 Security 1065
6.1 General Security Issues 1066
6.1.1 Security Guidelines 1066
6.1.2 Keeping Passwords Secure 1068
6.1.3 Making MySQL Secure Against Attackers 1071
6.1.4 Security-Related mysqld Options and Variables 1072
6.1.5 How to Run MySQL as a Normal User 1073
6.1.6 Security Issues with LOAD DATA LOCAL 1074
6.1.7 Client Programming Security Guidelines 1075
6.2 Access Control and Account Management 1077
6.2.1 Account User Names and Passwords 1078
6.2.2 Privileges Provided by MySQL 1079
6.2.3 Grant Tables 1095
6.2.4 Specifying Account Names 1104
6.2.5 Specifying Role Names 1107
6.2.6 Access Control, Stage 1: Connection Verification 1107
6.2.7 Access Control, Stage 2: Request Verification 1110
6.2.8 Adding Accounts, Assigning Privileges, and Dropping Accounts 1112
6.2.9 Reserved Accounts 1115
6.2.10 Using Roles 1116
6.2.11 Account Categories 1123
6.2.12 Privilege Restriction Using Partial Revokes 1127
6.2.13 When Privilege Changes Take Effect 1134
6.2.14 Assigning Account Passwords 1135
6.2.15 Password Management 1136
6.2.16 Server Handling of Expired Passwords 1146
6.2.17 Pluggable Authentication 1148
6.2.18 Proxy Users 1153
6.2.19 Account Locking 1160
6.2.20 Setting Account Resource Limits 1161
6.2.21 Troubleshooting Problems Connecting to MySQL 1163
6.2.22 SQL-Based Account Activity Auditing 1168
6.3 Using Encrypted Connections 1169
6.3.1 Configuring MySQL to Use Encrypted Connections 1170
6.3.2 Encrypted Connection TLS Protocols and Ciphers 1177
6.3.3 Creating SSL and RSA Certificates and Keys 1183
6.3.4 Connecting to MySQL Remotely from Windows with SSH 1192
6.4 Security Components and Plugins 1193
6.4.1 Authentication Plugins 1194
6.4.2 The Connection-Control Plugins 1258
6.4.3 The Password Validation Component 1265
6.4.4 The MySQL Keyring 1277
6.4.5 MySQL Enterprise Audit 1312
6.4.6 The Audit Message Component 1383
6.4.7 MySQL Enterprise Firewall 1386
6.4.8 MySQL Enterprise Data Masking and De-Identification 1398
6.5 FIPS Support 1416
7 Backup and Recovery 1419
7.1 Backup and Recovery Types 1420
7.2 Database Backup Methods 1423
7.3 Example Backup and Recovery Strategy 1425
7.3.1 Establishing a Backup Policy 1426

7.3.2 Using Backups for Recovery 1428
7.3.3 Backup Strategy Summary 1428
7.4 Using mysqldump for Backups 1429
7.4.1 Dumping Data in SQL Format with mysqldump 1429
7.4.2 Reloading SQL-Format Backups 1430
7.4.3 Dumping Data in Delimited-Text Format with mysqldump 1431
7.4.4 Reloading Delimited-Text Format Backups 1432
7.4.5 mysqldump Tips 1433
7.5 Point-in-Time (Incremental) Recovery Using the Binary Log 1435
7.5.1 Point-in-Time Recovery Using Event Times 1437
7.5.2 Point-in-Time Recovery Using Event Positions 1437
7.6 MyISAM Table Maintenance and Crash Recovery 1438
7.6.1 Using myisamchk for Crash Recovery 1439
7.6.2 How to Check MyISAM Tables for Errors 1439
7.6.3 How to Repair MyISAM Tables 1440
7.6.4 MyISAM Table Optimization 1442
7.6.5 Setting Up a MyISAM Table Maintenance Schedule 1443
8 Optimization 1445
8.1 Optimization Overview 1447
8.2 Optimizing SQL Statements 1448
8.2.1 Optimizing SELECT Statements 1448
8.2.2 Optimizing Subqueries, Derived Tables, View References, and Common Table Expressions 1502
8.2.3 Optimizing INFORMATION_SCHEMA Queries 1516
8.2.4 Optimizing Performance Schema Queries 1519
8.2.5 Optimizing Data Change Statements 1520
8.2.6 Optimizing Database Privileges 1522
8.2.7 Other Optimization Tips 1522
8.3 Optimization and Indexes 1522
8.3.1 How MySQL Uses Indexes 1523
8.3.2 Primary Key Optimization 1524
8.3.3 SPATIAL Index Optimization 1524
8.3.4 Foreign Key Optimization 1525
8.3.5 Column Indexes 1525
8.3.6 Multiple-Column Indexes 1527
8.3.7 Verifying Index Usage 1528
8.3.8 InnoDB and MyISAM Index Statistics Collection 1528
8.3.9 Comparison of B-Tree and Hash Indexes 1530
8.3.10 Use of Index Extensions 1531
8.3.11 Optimizer Use of Generated Column Indexes 1534
8.3.12 Invisible Indexes 1536
8.3.13 Descending Indexes 1537
8.3.14 Indexed Lookups from TIMESTAMP Columns 1538
8.4 Optimizing Database Structure 1540
8.4.1 Optimizing Data Size 1540
8.4.2 Optimizing MySQL Data Types 1542
8.4.3 Optimizing for Many Tables 1544
8.4.4 Internal Temporary Table Use in MySQL 1545
8.4.5 Limits on Number of Databases and Tables 1549
8.4.6 Limits on Table Size 1549
8.4.7 Limits on Table Column Count and Row Size 1550
8.5 Optimizing for InnoDB Tables 1553
8.5.1 Optimizing Storage Layout for InnoDB Tables 1553
8.5.2 Optimizing InnoDB Transaction Management 1554

8.5.3 Optimizing InnoDB Read-Only Transactions ... 1555
8.5.4 Optimizing InnoDB Redo Logging ... 1556
8.5.5 Bulk Data Loading for InnoDB Tables ... 1557
8.5.6 Optimizing InnoDB Queries ... 1558
8.5.7 Optimizing InnoDB DDL Operations ... 1559
8.5.8 Optimizing InnoDB Disk I/O ... 1559
8.5.9 Optimizing InnoDB Configuration Variables ... 1562
8.5.10 Optimizing InnoDB for Systems with Many Tables ... 1564
8.6 Optimizing for MyISAM Tables ... 1564
8.6.1 Optimizing MyISAM Queries ... 1564
8.6.2 Bulk Data Loading for MyISAM Tables ... 1565
8.6.3 Optimizing REPAIR TABLE Statements ... 1566
8.7 Optimizing for MEMORY Tables ... 1568
8.8 Understanding the Query Execution Plan ... 1568
8.8.1 Optimizing Queries with EXPLAIN ... 1568
8.8.2 EXPLAIN Output Format ... 1569
8.8.3 Extended EXPLAIN Output Format ... 1583
8.8.4 Obtaining Execution Plan Information for a Named Connection ... 1585
8.8.5 Estimating Query Performance ... 1586
8.9 Controlling the Query Optimizer ... 1586
8.9.1 Controlling Query Plan Evaluation ... 1586
8.9.2 Switchable Optimizations ... 1587
8.9.3 Optimizer Hints ... 1592
8.9.4 Index Hints ... 1606
8.9.5 The Optimizer Cost Model ... 1608
8.9.6 Optimizer Statistics ... 1612
8.10 Buffering and Caching ... 1615
8.10.1 InnoDB Buffer Pool Optimization ... 1615
8.10.2 The MyISAM Key Cache ... 1615
8.10.3 Caching of Prepared Statements and Stored Programs ... 1620
8.11 Optimizing Locking Operations ... 1621
8.11.1 Internal Locking Methods ... 1622
8.11.2 Table Locking Issues ... 1624
8.11.3 Concurrent Inserts ... 1626
8.11.4 Metadata Locking ... 1626
8.11.5 External Locking ... 1630
8.12 Optimizing the MySQL Server ... 1631
8.12.1 Optimizing Disk I/O ... 1631
8.12.2 Using Symbolic Links ... 1632
8.12.3 Optimizing Memory Use ... 1635
8.12.4 Optimizing Network Use ... 1642
8.12.5 Resource Groups ... 1646
8.13 Measuring Performance (Benchmarking) ... 1651
8.13.1 Measuring the Speed of Expressions and Functions ... 1652
8.13.2 Using Your Own Benchmarks ... 1652
8.13.3 Measuring Performance with performance_schema ... 1652
8.14 Examining Thread Information ... 1653
8.14.1 Thread Command Values ... 1654
8.14.2 General Thread States ... 1656
8.14.3 Replication Master Thread States ... 1662
8.14.4 Replication Slave I/O Thread States ... 1663
8.14.5 Replication Slave SQL Thread States ... 1664
8.14.6 Replication Slave Connection Thread States ... 1665
8.14.7 NDB Cluster Thread States ... 1665

8.14.8 Event Scheduler Thread States 1666
9 Language Structure 1669
9.1 Literal Values 1669
9.1.1 String Literals 1669
9.1.2 Numeric Literals 1672
9.1.3 Date and Time Literals 1672
9.1.4 Hexadecimal Literals 1675
9.1.5 Bit-Value Literals 1677
9.1.6 Boolean Literals 1679
9.1.7 NULL Values 1679
9.2 Schema Object Names 1679
9.2.1 Identifier Length Limits 1681
9.2.2 Identifier Qualifiers 1682
9.2.3 Identifier Case Sensitivity 1684
9.2.4 Mapping of Identifiers to File Names 1686
9.2.5 Function Name Parsing and Resolution 1687
9.3 Keywords and Reserved Words 1691
9.4 User-Defined Variables 1720
9.5 Expressions 1723
9.6 Comment Syntax 1727
10 Character Sets, Collations, Unicode 1731
10.1 Character Sets and Collations in General 1732
10.2 Character Sets and Collations in MySQL 1733
10.2.1 Character Set Repertoire 1735
10.2.2 UTF-8 for Metadata 1737
10.3 Specifying Character Sets and Collations 1739
10.3.1 Collation Naming Conventions 1739
10.3.2 Server Character Set and Collation 1740
10.3.3 Database Character Set and Collation 1741
10.3.4 Table Character Set and Collation 1742
10.3.5 Column Character Set and Collation 1743
10.3.6 Character String Literal Character Set and Collation 1744
10.3.7 The National Character Set 1746
10.3.8 Character Set Introducers 1747
10.3.9 Examples of Character Set and Collation Assignment 1749
10.3.10 Compatibility with Other DBMSs 1750
10.4 Connection Character Sets and Collations 1750
10.5 Configuring Application Character Set and Collation 1757
10.6 Error Message Character Set 1758
10.7 Column Character Set Conversion 1759
10.8 Collation Issues 1760
10.8.1 Using COLLATE in SQL Statements 1761
10.8.2 COLLATE Clause Precedence 1761
10.8.3 Character Set and Collation Compatibility 1762
10.8.4 Collation Coercibility in Expressions 1762
10.8.5 The binary Collation Compared to _bin Collations 1763
10.8.6 Examples of the Effect of Collation 1766
10.8.7 Using Collation in INFORMATION_SCHEMA Searches 1767
10.9 Unicode Support 1769
10.9.1 The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding) 1771
10.9.2 The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding) 1772
10.9.3 The utf8 Character Set (Alias for utf8mb3) 1773
10.9.4 The ucs2 Character Set (UCS-2 Unicode Encoding) 1773
10.9.5 The utf16 Character Set (UTF-16 Unicode Encoding) 1773

10.9.6 The utf16le Character Set (UTF-16LE Unicode Encoding) 1774
10.9.7 The utf32 Character Set (UTF-32 Unicode Encoding) 1774
10.9.8 Converting Between 3-Byte and 4-Byte Unicode Character Sets 1774
10.10 Supported Character Sets and Collations 1777
10.10.1 Unicode Character Sets 1778
10.10.2 West European Character Sets 1785
10.10.3 Central European Character Sets 1787
10.10.4 South European and Middle East Character Sets 1788
10.10.5 Baltic Character Sets 1788
10.10.6 Cyrillic Character Sets 1789
10.10.7 Asian Character Sets 1789
10.10.8 The Binary Character Set 1794
10.11 Restrictions on Character Sets 1795
10.12 Setting the Error Message Language 1795
10.13 Adding a Character Set 1796
10.13.1 Character Definition Arrays 1798
10.13.2 String Collating Support for Complex Character Sets 1799
10.13.3 Multi-Byte Character Support for Complex Character Sets 1799
10.14 Adding a Collation to a Character Set 1799
10.14.1 Collation Implementation Types 1801
10.14.2 Choosing a Collation ID 1803
10.14.3 Adding a Simple Collation to an 8-Bit Character Set 1804
10.14.4 Adding a UCA Collation to a Unicode Character Set 1805
10.15 Character Set Configuration 1813
10.16 MySQL Server Locale Support 1814
11 Data Types 1817
11.1 Data Type Overview 1818
11.1.1 Numeric Type Overview 1818
11.1.2 Date and Time Type Overview 1822
11.1.3 String Type Overview 1824
11.2 Numeric Types 1828
11.2.1 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT 1828
11.2.2 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC 1829
11.2.3 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE 1829
11.2.4 Bit-Value Type - BIT 1830
11.2.5 Numeric Type Attributes 1830
11.2.6 Out-of-Range and Overflow Handling 1831
11.3 Date and Time Types 1833
11.3.1 The DATE, DATETIME, and TIMESTAMP Types 1834
11.3.2 The TIME Type 1836
11.3.3 The YEAR Type 1836
11.3.4 Automatic Initialization and Updating for TIMESTAMP and DATETIME 1837
11.3.5 Fractional Seconds in Time Values 1840
11.3.6 Conversion Between Date and Time Types 1841
11.3.7 Two-Digit Years in Dates 1842
11.4 String Types 1843
11.4.1 The CHAR and VARCHAR Types 1843
11.4.2 The BINARY and VARBINARY Types 1845
11.4.3 The BLOB and TEXT Types 1846
11.4.4 The ENUM Type 1848
11.4.5 The SET Type 1851
11.5 Spatial Data Types 1853
11.5.1 Spatial Data Types 1855

11.5.2 The OpenGIS Geometry Model 1856
11.5.3 Supported Spatial Data Formats 1862
11.5.4 Geometry Well-Formedness and Validity 1865
11.5.5 Spatial Reference System Support 1866
11.5.6 Creating Spatial Columns 1867
11.5.7 Populating Spatial Columns 1867
11.5.8 Fetching Spatial Data 1869
11.5.9 Optimizing Spatial Analysis 1869
11.5.10 Creating Spatial Indexes 1869
11.5.11 Using Spatial Indexes 1871
11.6 The JSON Data Type 1872
11.7 Data Type Default Values 1890
11.8 Data Type Storage Requirements 1893
11.9 Choosing the Right Type for a Column 1897
11.10 Using Data Types from Other Database Engines 1898
12 Functions and Operators 1899
12.1 Function and Operator Reference 1901
12.2 Type Conversion in Expression Evaluation 1914
12.3 Operators 1917
12.3.1 Operator Precedence 1918
12.3.2 Comparison Functions and Operators 1919
12.3.3 Logical Operators 1926
12.3.4 Assignment Operators 1928
12.4 Control Flow Functions 1929
12.5 String Functions and Operators 1932
12.5.1 String Comparison Functions and Operators 1948
12.5.2 Regular Expressions 1951
12.5.3 Character Set and Collation of Function Results 1962
12.6 Numeric Functions and Operators 1963
12.6.1 Arithmetic Operators 1964
12.6.2 Mathematical Functions 1966
12.7 Date and Time Functions 1975
12.8 What Calendar Is Used By MySQL? 1997
12.9 Full-Text Search Functions 1997
12.9.1 Natural Language Full-Text Searches 1999
12.9.2 Boolean Full-Text Searches 2002
12.9.3 Full-Text Searches with Query Expansion 2008
12.9.4 Full-Text Stopwords 2009
12.9.5 Full-Text Restrictions 2013
12.9.6 Fine-Tuning MySQL Full-Text Search 2014
12.9.7 Adding a Collation for Full-Text Indexing 2017
12.9.8 ngram Full-Text Parser 2019
12.9.9 MeCab Full-Text Parser Plugin 2021
12.10 Cast Functions and Operators 2026
12.11 XML Functions 2032
12.12 Bit Functions and Operators 2043
12.13 Encryption and Compression Functions 2055
12.14 Locking Functions 2062
12.15 Information Functions 2065
12.16 Spatial Analysis Functions 2076
12.16.1 Spatial Function Reference 2076
12.16.2 Argument Handling by Spatial Functions 2079
12.16.3 Functions That Create Geometry Values from WKT Values 2080
12.16.4 Functions That Create Geometry Values from WKB Values 2082

12.16.5 MySQL-Specific Functions That Create Geometry Values 2084
12.16.6 Geometry Format Conversion Functions 2085
12.16.7 Geometry Property Functions 2087
12.16.8 Spatial Operator Functions 2100
12.16.9 Functions That Test Spatial Relations Between Geometry Objects 2104
12.16.10 Spatial Geohash Functions 2110
12.16.11 Spatial GeoJSON Functions 2112
12.16.12 Spatial Convenience Functions 2114
12.17 JSON Functions 2118
12.17.1 JSON Function Reference 2118
12.17.2 Functions That Create JSON Values 2120
12.17.3 Functions That Search JSON Values 2121
12.17.4 Functions That Modify JSON Values 2134
12.17.5 Functions That Return JSON Value Attributes 2143
12.17.6 JSON Table Functions 2146
12.17.7 JSON Schema Validation Functions 2150
12.17.8 JSON Utility Functions 2155
12.18 Functions Used with Global Transaction Identifiers (GTIDs) 2161
12.19 MySQL Enterprise Encryption Functions 2163
12.19.1 MySQL Enterprise Encryption Installation 2163
12.19.2 MySQL Enterprise Encryption Usage and Examples 2164
12.19.3 MySQL Enterprise Encryption Function Reference 2166
12.19.4 MySQL Enterprise Encryption Function Descriptions 2167
12.20 Aggregate (GROUP BY) Functions 2171
12.20.1 Aggregate (GROUP BY) Function Descriptions 2171
12.20.2 GROUP BY Modifiers 2181
12.20.3 MySQL Handling of GROUP BY 2187
12.20.4 Detection of Functional Dependence 2190
12.21 Window Functions 2194
12.21.1 Window Function Descriptions 2194
12.21.2 Window Function Concepts and Syntax 2200
12.21.3 Window Function Frame Specification 2203
12.21.4 Named Windows 2207
12.21.5 Window Function Restrictions 2209
12.22 Performance Schema Functions 2209
12.23 Internal Functions 2212
12.24 Miscellaneous Functions 2213
12.25 Precision Math 2228
12.25.1 Types of Numeric Values 2229
12.25.2 DECIMAL Data Type Characteristics 2229
12.25.3 Expression Handling 2230
12.25.4 Rounding Behavior 2232
12.25.5 Precision Math Examples 2233
13 SQL Statement Syntax 2237
13.1 Data Definition Statements 2238
13.1.1 Atomic Data Definition Statement Support 2238
13.1.2 ALTER DATABASE Syntax 2244
13.1.3 ALTER EVENT Syntax 2245
13.1.4 ALTER FUNCTION Syntax 2247
13.1.5 ALTER INSTANCE Syntax 2247
13.1.6 ALTER LOGFILE GROUP Syntax 2248
13.1.7 ALTER PROCEDURE Syntax 2249
13.1.8 ALTER SERVER Syntax 2250
13.1.9 ALTER TABLE Syntax 2250

13.1.10 ALTER TABLESPACE Syntax ... 2274
13.1.11 ALTER VIEW Syntax ... 2276
13.1.12 CREATE DATABASE Syntax ... 2276
13.1.13 CREATE EVENT Syntax ... 2277
13.1.14 CREATE FUNCTION Syntax ... 2282
13.1.15 CREATE INDEX Syntax ... 2282
13.1.16 CREATE LOGFILE GROUP Syntax ... 2297
13.1.17 CREATE PROCEDURE and CREATE FUNCTION Syntax ... 2299
13.1.18 CREATE SERVER Syntax ... 2304
13.1.19 CREATE SPATIAL REFERENCE SYSTEM Syntax ... 2305
13.1.20 CREATE TABLE Syntax ... 2310
13.1.21 CREATE TABLESPACE Syntax ... 2358
13.1.22 CREATE TRIGGER Syntax ... 2365
13.1.23 CREATE VIEW Syntax ... 2367
13.1.24 DROP DATABASE Syntax ... 2371
13.1.25 DROP EVENT Syntax ... 2372
13.1.26 DROP FUNCTION Syntax ... 2372
13.1.27 DROP INDEX Syntax ... 2372
13.1.28 DROP LOGFILE GROUP Syntax ... 2373
13.1.29 DROP PROCEDURE and DROP FUNCTION Syntax ... 2373
13.1.30 DROP SERVER Syntax ... 2373
13.1.31 DROP SPATIAL REFERENCE SYSTEM Syntax ... 2374
13.1.32 DROP TABLE Syntax ... 2374
13.1.33 DROP TABLESPACE Syntax ... 2375
13.1.34 DROP TRIGGER Syntax ... 2377
13.1.35 DROP VIEW Syntax ... 2377
13.1.36 RENAME TABLE Syntax ... 2377
13.1.37 TRUNCATE TABLE Syntax ... 2379
13.2 Data Manipulation Statements ... 2381
13.2.1 CALL Syntax ... 2381
13.2.2 DELETE Syntax ... 2382
13.2.3 DO Syntax ... 2386
13.2.4 HANDLER Syntax ... 2387
13.2.5 IMPORT TABLE Syntax ... 2389
13.2.6 INSERT Syntax ... 2391
13.2.7 LOAD DATA Syntax ... 2399
13.2.8 LOAD XML Syntax ... 2410
13.2.9 REPLACE Syntax ... 2418
13.2.10 SELECT Syntax ... 2421
13.2.11 Subquery Syntax ... 2438
13.2.12 UPDATE Syntax ... 2455
13.2.13 WITH Syntax (Common Table Expressions) ... 2458
13.3 Transactional and Locking Statements ... 2470
13.3.1 START TRANSACTION, COMMIT, and ROLLBACK Syntax ... 2470
13.3.2 Statements That Cannot Be Rolled Back ... 2473
13.3.3 Statements That Cause an Implicit Commit ... 2474
13.3.4 SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax ... 2475
13.3.5 LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Syntax ... 2475
13.3.6 LOCK TABLES and UNLOCK TABLES Syntax ... 2476
13.3.7 SET TRANSACTION Syntax ... 2482
13.3.8 XA Transactions ... 2485
13.4 Replication Statements ... 2490
13.4.1 SQL Statements for Controlling Master Servers ... 2490
13.4.2 SQL Statements for Controlling Slave Servers ... 2493

13.4.3 SQL Statements for Controlling Group Replication ... 2511
13.5 Prepared SQL Statement Syntax ... 2515
13.5.1 PREPARE Syntax ... 2519
13.5.2 EXECUTE Syntax ... 2519
13.5.3 DEALLOCATE PREPARE Syntax ... 2519
13.6 Compound-Statement Syntax ... 2520
13.6.1 BEGIN ... END Compound-Statement Syntax ... 2520
13.6.2 Statement Label Syntax ... 2520
13.6.3 DECLARE Syntax ... 2521
13.6.4 Variables in Stored Programs ... 2522
13.6.5 Flow Control Statements ... 2523
13.6.6 Cursors ... 2528
13.6.7 Condition Handling ... 2530
13.6.8 Restrictions on Condition Handling ... 2559
13.7 Database Administration Statements ... 2560
13.7.1 Account Management Statements ... 2560
13.7.2 Resource Group Management Statements ... 2604
13.7.3 Table Maintenance Statements ... 2608
13.7.4 Component, Plugin, and User-Defined Function Statements ... 2620
13.7.5 CLONE Syntax ... 2624
13.7.6 SET Syntax ... 2625
13.7.7 SHOW Syntax ... 2631
13.7.8 Other Administrative Statements ... 2689
13.8 Utility Statements ... 2701
13.8.1 DESCRIBE Syntax ... 2701
13.8.2 EXPLAIN Syntax ... 2701
13.8.3 HELP Syntax ... 2704
13.8.4 USE Syntax ... 2706
14 MySQL Data Dictionary ... 2709
14.1 Data Dictionary Schema ... 2709
14.2 Removal of File-based Metadata Storage ... 2711
14.3 Transactional Storage of Dictionary Data ... 2711
14.4 Dictionary Object Cache ... 2711
14.5 INFORMATION_SCHEMA and Data Dictionary Integration ... 2712
14.6 Serialized Dictionary Information (SDI) ... 2714
14.7 Data Dictionary Usage Differences ... 2715
14.8 Data Dictionary Limitations ... 2716
15 The InnoDB Storage Engine ... 2717
15.1 Introduction to InnoDB ... 2718
15.1.1 Benefits of Using InnoDB Tables ... 2720
15.1.2 Best Practices for InnoDB Tables ... 2721
15.1.3 Verifying that InnoDB is the Default Storage Engine ... 2722
15.1.4 Testing and Benchmarking with InnoDB ... 2722
15.2 InnoDB and the ACID Model ... 2722
15.3 InnoDB Multi-Versioning ... 2724
15.4 InnoDB Architecture ... 2725
15.5 InnoDB In-Memory Structures ... 2726
15.5.1 Buffer Pool ... 2726
15.5.2 Change Buffer ... 2731
15.5.3 Adaptive Hash Index ... 2735
15.5.4 Log Buffer ... 2735
15.6 InnoDB On-Disk Structures ... 2736
15.6.1 Tables ... 2736
15.6.2 Indexes ... 2757

15.6.3 Tablespaces 2765
15.6.4 Doublewrite Buffer 2804
15.6.5 Redo Log 2805
15.6.6 Undo Logs 2808
15.7 InnoDB Locking and Transaction Model 2810
15.7.1 InnoDB Locking 2810
15.7.2 InnoDB Transaction Model 2815
15.7.3 Locks Set by Different SQL Statements in InnoDB 2824
15.7.4 Phantom Rows 2827
15.7.5 Deadlocks in InnoDB 2828
15.8 InnoDB Configuration 2832
15.8.1 InnoDB Startup Configuration 2832
15.8.2 Configuring InnoDB for Read-Only Operation 2838
15.8.3 InnoDB Buffer Pool Configuration 2839
15.8.4 Configuring Thread Concurrency for InnoDB 2854
15.8.5 Configuring the Number of Background InnoDB I/O Threads 2855
15.8.6 Using Asynchronous I/O on Linux 2855
15.8.7 Configuring InnoDB I/O Capacity 2856
15.8.8 Configuring Spin Lock Polling 2856
15.8.9 Purge Configuration 2857
15.8.10 Configuring Optimizer Statistics for InnoDB 2859
15.8.11 Configuring the Merge Threshold for Index Pages 2870
15.8.12 Enabling Automatic Configuration for a Dedicated MySQL Server 2873
15.9 InnoDB Table and Page Compression 2875
15.9.1 InnoDB Table Compression 2875
15.9.2 InnoDB Page Compression 2889
15.10 InnoDB Row Formats 2893
15.11 InnoDB Disk I/O and File Space Management 2899
15.11.1 InnoDB Disk I/O 2900
15.11.2 File Space Management 2900
15.11.3 InnoDB Checkpoints 2902
15.11.4 Defragmenting a Table 2902
15.11.5 Reclaiming Disk Space with TRUNCATE TABLE 2903
15.12 InnoDB and Online DDL 2903
15.12.1 Online DDL Operations 2904
15.12.2 Online DDL Performance and Concurrency 2918
15.12.3 Online DDL Space Requirements 2922
15.12.4 Simplifying DDL Statements with Online DDL 2923
15.12.5 Online DDL Failure Conditions 2923
15.12.6 Online DDL Limitations 2924
15.13 InnoDB Startup Options and System Variables 2925
15.14 InnoDB INFORMATION_SCHEMA Tables 3019
15.14.1 InnoDB INFORMATION_SCHEMA Tables about Compression 3019
15.14.2 InnoDB INFORMATION_SCHEMA Transaction and Locking Information 3021
15.14.3 InnoDB INFORMATION_SCHEMA Schema Object Tables 3028
15.14.4 InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables 3033
15.14.5 InnoDB INFORMATION_SCHEMA Buffer Pool Tables 3037
15.14.6 InnoDB INFORMATION_SCHEMA Metrics Table 3041
15.14.7 InnoDB INFORMATION_SCHEMA Temporary Table Info Table 3050
15.14.8 Retrieving InnoDB Tablespace Metadata from INFORMATION_SCHEMA.FILES 3051
15.15 InnoDB Integration with MySQL Performance Schema 3053
15.15.1 Monitoring ALTER TABLE Progress for InnoDB Tables Using Performance Schema 3055
15.15.2 Monitoring InnoDB Mutex Waits Using Performance Schema 3057

15.16 InnoDB Monitors 3061
15.16.1 InnoDB Monitor Types 3061
15.16.2 Enabling InnoDB Monitors 3061
15.16.3 InnoDB Standard Monitor and Lock Monitor Output 3063
15.17 InnoDB Backup and Recovery 3067
15.17.1 InnoDB Backup 3068
15.17.2 InnoDB Recovery 3068
15.18 InnoDB and MySQL Replication 3071
15.19 InnoDB memcached Plugin 3073
15.19.1 Benefits of the InnoDB memcached Plugin 3073
15.19.2 InnoDB memcached Architecture 3075
15.19.3 Setting Up the InnoDB memcached Plugin 3078
15.19.4 InnoDB memcached Multiple get and Range Query Support 3084
15.19.5 Security Considerations for the InnoDB memcached Plugin 3087
15.19.6 Writing Applications for the InnoDB memcached Plugin 3088
15.19.7 The InnoDB memcached Plugin and Replication 3101
15.19.8 InnoDB memcached Plugin Internals 3105
15.19.9 Troubleshooting the InnoDB memcached Plugin 3110
15.20 InnoDB Troubleshooting 3112
15.20.1 Troubleshooting InnoDB I/O Problems 3112
15.20.2 Forcing InnoDB Recovery 3113
15.20.3 Troubleshooting InnoDB Data Dictionary Operations 3114
15.20.4 InnoDB Error Handling 3116
16 Alternative Storage Engines 3119
16.1 Setting the Storage Engine 3123
16.2 The MyISAM Storage Engine 3124
16.2.1 MyISAM Startup Options 3126
16.2.2 Space Needed for Keys 3128
16.2.3 MyISAM Table Storage Formats 3128
16.2.4 MyISAM Table Problems 3131
16.3 The MEMORY Storage Engine 3132
16.4 The CSV Storage Engine 3137
16.4.1 Repairing and Checking CSV Tables 3138
16.4.2 CSV Limitations 3139
16.5 The ARCHIVE Storage Engine 3139
16.6 The BLACKHOLE Storage Engine 3141
16.7 The MERGE Storage Engine 3143
16.7.1 MERGE Table Advantages and Disadvantages 3146
16.7.2 MERGE Table Problems 3147
16.8 The FEDERATED Storage Engine 3149
16.8.1 FEDERATED Storage Engine Overview 3149
16.8.2 How to Create FEDERATED Tables 3150
16.8.3 FEDERATED Storage Engine Notes and Tips 3153
16.8.4 FEDERATED Storage Engine Resources 3154
16.9 The EXAMPLE Storage Engine 3154
16.10 Other Storage Engines 3155
16.11 Overview of MySQL Storage Engine Architecture 3155
16.11.1 Pluggable Storage Engine Architecture 3155
16.11.2 The Common Database Server Layer 3156
17 Replication 3159
17.1 Configuring Replication 3161
17.1.1 Binary Log File Position Based Replication Configuration Overview 3161
17.1.2 Setting Up Binary Log File Position Based Replication 3162
17.1.3 Replication with Global Transaction Identifiers 3172

17.1.4 MySQL Multi-Source Replication 3194
17.1.5 Changing Replication Modes on Online Servers 3198
17.1.6 Replication and Binary Logging Options and Variables 3204
17.1.7 Common Replication Administration Tasks 3300
17.2 Replication Implementation 3304
17.2.1 Replication Formats 3304
17.2.2 Replication Implementation Details 3312
17.2.3 Replication Channels 3314
17.2.4 Replication Relay and Status Logs 3317
17.2.5 How Servers Evaluate Replication Filtering Rules 3323
17.3 Replication Security 3331
17.3.1 Setting Up Replication to Use Encrypted Connections 3331
17.3.2 Encrypting Binary Log Files and Relay Log Files 3333
17.3.3 Replication Privilege Checks 3337
17.4 Replication Solutions 3342
17.4.1 Using Replication for Backups 3342
17.4.2 Handling an Unexpected Halt of a Replication Slave 3346
17.4.3 Monitoring Row-based Replication 3348
17.4.4 Using Replication with Different Master and Slave Storage Engines 3349
17.4.5 Using Replication for Scale-Out 3350
17.4.6 Replicating Different Databases to Different Slaves 3351
17.4.7 Improving Replication Performance 3352
17.4.8 Switching Masters During Failover 3354
17.4.9 Semisynchronous Replication 3356
17.4.10 Delayed Replication 3361
17.5 Replication Notes and Tips 3364
17.5.1 Replication Features and Issues 3364
17.5.2 Replication Compatibility Between MySQL Versions 3390
17.5.3 Upgrading a Replication Setup 3391
17.5.4 Troubleshooting Replication 3393
17.5.5 How to Report Replication Bugs or Problems 3394
18 Group Replication 3397
18.1 Group Replication Background 3398
18.1.1 Replication Technologies 3399
18.1.2 Group Replication Use Cases 3401
18.1.3 Multi-Primary and Single-Primary Modes 3402
18.1.4 Group Replication Services 3406
18.1.5 Group Replication Plugin Architecture 3409
18.2 Getting Started 3410
18.2.1 Deploying Group Replication in Single-Primary Mode 3410
18.2.2 Deploying Group Replication Locally 3422
18.3 Monitoring Group Replication 3424
18.3.1 Group Replication Server States 3424
18.3.2 The replication_group_members Table 3425
18.3.3 The replication_group_member_stats Table 3426
18.4 Group Replication Operations 3426
18.4.1 Configuring an Online Group 3426
18.4.2 Transaction Consistency Guarantees 3431
18.4.3 Distributed Recovery 3437
18.4.4 Network Partitioning 3449
18.4.5 Support For IPv6 And For Mixed IPv6 And IPv4 Groups 3454
18.4.6 Using MySQL Enterprise Backup with Group Replication 3456
18.5 Group Replication Security 3462
18.5.1 Group Replication IP Address Whitelisting 3462

18.5.2 Group Replication Secure Socket Layer (SSL) Support ... 3464
18.6 Group Replication Performance ... 3468
18.6.1 Fine Tuning the Group Communication Thread ... 3468
18.6.2 Flow Control ... 3468
18.6.3 Message Compression ... 3470
18.6.4 Message Fragmentation ... 3472
18.6.5 XCom Cache Management ... 3472
18.6.6 Responses to Failure Detection and Network Partitioning ... 3473
18.7 Upgrading Group Replication ... 3478
18.7.1 Combining Different Member Versions in a Group ... 3479
18.7.2 Group Replication Offline Upgrade ... 3481
18.7.3 Group Replication Online Upgrade ... 3482
18.8 Group Replication System Variables ... 3485
18.9 Requirements and Limitations ... 3518
18.9.1 Group Replication Requirements ... 3518
18.9.2 Group Replication Limitations ... 3520
18.10 Frequently Asked Questions ... 3522
19 MySQL Shell ... 3527
20 Using MySQL as a Document Store ... 3529
20.1 Interfaces to a MySQL Document Store ... 3530
20.2 Document Store Concepts ... 3530
20.3 JavaScript Quick-Start Guide: MySQL Shell for Document Store ... 3531
20.3.1 MySQL Shell ... 3532
20.3.2 Download and Import world_x Database ... 3533
20.3.3 Documents and Collections ... 3534
20.3.4 Relational Tables ... 3545
20.3.5 Documents in Tables ... 3551
20.4 Python Quick-Start Guide: MySQL Shell for Document Store ... 3552
20.4.1 MySQL Shell ... 3553
20.4.2 Download and Import world_x Database ... 3554
20.4.3 Documents and Collections ... 3555
20.4.4 Relational Tables ... 3567
20.4.5 Documents in Tables ... 3573
20.5 X Plugin ... 3574
20.5.1 Checking X Plugin Installation ... 3574
20.5.2 Disabling X Plugin ... 3574
20.5.3 Using Secure Connections with X Plugin ... 3574
20.5.4 Using X Plugin with the Caching SHA-2 Authentication Plugin ... 3575
20.5.5 X Plugin Options and Variables ... 3575
20.5.6 Monitoring X Plugin ... 3587
21 InnoDB Cluster ... 3593
21.1 Introducing InnoDB Cluster ... 3593
21.2 Creating an InnoDB Cluster ... 3596
21.2.1 Deployment Scenarios ... 3596
21.2.2 InnoDB Cluster Requirements ... 3596
21.2.3 Methods of Installing ... 3597
21.2.4 Production Deployment of InnoDB Cluster ... 3597
21.2.5 Using MySQL Clone with InnoDB cluster ... 3608
21.2.6 Sandbox Deployment of InnoDB Cluster ... 3613
21.2.7 Adopting a Group Replication Deployment ... 3616
21.3 Using MySQL Router with InnoDB Cluster ... 3616
21.4 Working with InnoDB Cluster ... 3620
21.5 Known Limitations ... 3647
22 MySQL NDB Cluster 8.0 ... 3649

22.1 NDB Cluster Overview 3653
22.1.1 NDB Cluster Core Concepts 3654
22.1.2 NDB Cluster Nodes, Node Groups, Replicas, and Partitions 3657
22.1.3 NDB Cluster Hardware, Software, and Networking Requirements 3660
22.1.4 What is New in NDB Cluster 3661
22.1.5 Options, Variables, and Parameters Added, Deprecated or Removed in NDB 8.0 ... 3670
22.1.6 MySQL Server Using InnoDB Compared with NDB Cluster 3671
22.1.7 Known Limitations of NDB Cluster 3674
22.2 NDB Cluster Installation 3685
22.2.1 The NDB Cluster Auto-Installer 3688
22.2.2 Installation of NDB Cluster on Linux 3710
22.2.3 Installing NDB Cluster on Windows 3719
22.2.4 Initial Configuration of NDB Cluster 3728
22.2.5 Initial Startup of NDB Cluster 3730
22.2.6 NDB Cluster Example with Tables and Data 3731
22.2.7 Safe Shutdown and Restart of NDB Cluster 3735
22.2.8 Upgrading and Downgrading NDB Cluster 3736
22.3 Configuration of NDB Cluster 3737
22.3.1 Quick Test Setup of NDB Cluster 3737
22.3.2 Overview of NDB Cluster Configuration Parameters, Options, and Variables 3740
22.3.3 NDB Cluster Configuration Files 3759
22.3.4 Using High-Speed Interconnects with NDB Cluster 3946
22.4 NDB Cluster Programs 3946
22.4.1 ndbd — The NDB Cluster Data Node Daemon 3947
22.4.2 ndbinfo_select_all — Select From ndbinfo Tables 3955
22.4.3 ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded) 3956
22.4.4 ndb_mgmd — The NDB Cluster Management Server Daemon 3958
22.4.5 ndb_mgm — The NDB Cluster Management Client 3966
22.4.6 ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables 3968
22.4.7 ndb_config — Extract NDB Cluster Configuration Information 3971
22.4.8 ndb_delete_all — Delete All Rows from an NDB Table 3980
22.4.9 ndb_desc — Describe NDB Tables 3981
22.4.10 ndb_drop_index — Drop Index from an NDB Table 3987
22.4.11 ndb_drop_table — Drop an NDB Table 3988
22.4.12 ndb_error_reporter — NDB Error-Reporting Utility 3989
22.4.13 ndb_import — Import CSV Data Into NDB 3990
22.4.14 ndb_index_stat — NDB Index Statistics Utility 4004
22.4.15 ndb_move_data — NDB Data Copy Utility 4010
22.4.16 ndb_perror — Obtain NDB Error Message Information 4013
22.4.17 ndb_print_backup_file — Print NDB Backup File Contents 4015
22.4.18 ndb_print_file — Print NDB Disk Data File Contents 4015
22.4.19 ndb_print_frag_file — Print NDB Fragment List File Contents 4016
22.4.20 ndb_print_schema_file — Print NDB Schema File Contents 4017
22.4.21 ndb_print_sys_file — Print NDB System File Contents 4017
22.4.22 ndb_redo_log_reader — Check and Print Content of Cluster Redo Log 4018
22.4.23 ndb_restore — Restore an NDB Cluster Backup 4021
22.4.24 ndb_select_all — Print Rows from an NDB Table 4044
22.4.25 ndb_select_count — Print Row Counts for NDB Tables 4047
22.4.26 ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster 4048
22.4.27 ndb_show_tables — Display List of NDB Tables 4052
22.4.28 ndb_size.pl — NDBCLUSTER Size Requirement Estimator 4053
22.4.29 ndb_top — View CPU usage information for NDB threads 4056
22.4.30 ndb_waiter — Wait for NDB Cluster to Reach a Given Status 4062

22.4.31 Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs 4065
22.5 Management of NDB Cluster 4070
22.5.1 Summary of NDB Cluster Start Phases 4071
22.5.2 Commands in the NDB Cluster Management Client 4073
22.5.3 Online Backup of NDB Cluster 4078
22.5.4 MySQL Server Usage for NDB Cluster 4083
22.5.5 Performing a Rolling Restart of an NDB Cluster 4085
22.5.6 Event Reports Generated in NDB Cluster 4087
22.5.7 NDB Cluster Log Messages 4098
22.5.8 NDB Cluster Single User Mode 4114
22.5.9 Quick Reference: NDB Cluster SQL Statements 4114
22.5.10 ndbinfo: The NDB Cluster Information Database 4117
22.5.11 INFORMATION_SCHEMA Tables for NDB Cluster 4168
22.5.12 NDB Cluster Security Issues 4169
22.5.13 NDB Cluster Disk Data Tables 4176
22.5.14 Online Operations with ALTER TABLE in NDB Cluster 4184
22.5.15 Adding NDB Cluster Data Nodes Online 4187
22.5.16 Distributed MySQL Privileges with NDB_STORED_USER 4199
22.5.17 NDB API Statistics Counters and Variables 4200
22.6 NDB Cluster Replication 4211
22.6.1 NDB Cluster Replication: Abbreviations and Symbols 4212
22.6.2 General Requirements for NDB Cluster Replication 4213
22.6.3 Known Issues in NDB Cluster Replication 4214
22.6.4 NDB Cluster Replication Schema and Tables 4221
22.6.5 Preparing the NDB Cluster for Replication 4224
22.6.6 Starting NDB Cluster Replication (Single Replication Channel) 4226
22.6.7 Using Two Replication Channels for NDB Cluster Replication 4228
22.6.8 Implementing Failover with NDB Cluster Replication 4229
22.6.9 NDB Cluster Backups With NDB Cluster Replication 4230
22.6.10 NDB Cluster Replication: Multi-Master and Circular Replication 4237
22.6.11 NDB Cluster Replication Conflict Resolution 4241
22.7 NDB Cluster Release Notes 4255
23 Partitioning 4257
23.1 Overview of Partitioning in MySQL 4258
23.2 Partitioning Types 4261
23.2.1 RANGE Partitioning 4263
23.2.2 LIST Partitioning 4267
23.2.3 COLUMNS Partitioning 4269
23.2.4 HASH Partitioning 4278
23.2.5 KEY Partitioning 4281
23.2.6 Subpartitioning 4283
23.2.7 How MySQL Partitioning Handles NULL 4284
23.3 Partition Management 4289
23.3.1 Management of RANGE and LIST Partitions 4290
23.3.2 Management of HASH and KEY Partitions 4297
23.3.3 Exchanging Partitions and Subpartitions with Tables 4298
23.3.4 Maintenance of Partitions 4306
23.3.5 Obtaining Information About Partitions 4307
23.4 Partition Pruning 4309
23.5 Partition Selection 4313
23.6 Restrictions and Limitations on Partitioning 4319
23.6.1 Partitioning Keys, Primary Keys, and Unique Keys 4324
23.6.2 Partitioning Limitations Relating to Storage Engines 4328

23.6.3 Partitioning Limitations Relating to Functions 4329
24 Stored Objects 4331
24.1 Defining Stored Programs 4332
24.2 Using Stored Routines 4333
24.2.1 Stored Routine Syntax 4334
24.2.2 Stored Routines and MySQL Privileges 4335
24.2.3 Stored Routine Metadata 4335
24.2.4 Stored Procedures, Functions, Triggers, and LAST_INSERT_ID() 4335
24.3 Using Triggers 4335
24.3.1 Trigger Syntax and Examples 4336
24.3.2 Trigger Metadata 4340
24.4 Using the Event Scheduler 4340
24.4.1 Event Scheduler Overview 4341
24.4.2 Event Scheduler Configuration 4342
24.4.3 Event Syntax 4344
24.4.4 Event Metadata 4344
24.4.5 Event Scheduler Status 4345
24.4.6 The Event Scheduler and MySQL Privileges 4346
24.5 Using Views 4349
24.5.1 View Syntax 4349
24.5.2 View Processing Algorithms 4349
24.5.3 Updatable and Insertable Views 4351
24.5.4 The View WITH CHECK OPTION Clause 4354
24.5.5 View Metadata 4354
24.6 Stored Object Access Control 4354
24.7 Stored Program Binary Logging 4357
24.8 Restrictions on Stored Programs 4363
24.9 Restrictions on Views 4367
25 INFORMATION_SCHEMA Tables 4369
25.1 Introduction 4370
25.2 The INFORMATION_SCHEMA CHARACTER_SETS Table 4373
25.3 The INFORMATION_SCHEMA CHECK_CONSTRAINTS Table 4374
25.4 The INFORMATION_SCHEMA COLLATIONS Table 4374
25.5 The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table 4375
25.6 The INFORMATION_SCHEMA COLUMNS Table 4375
25.7 The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table 4378
25.8 The INFORMATION_SCHEMA COLUMN_STATISTICS Table 4379
25.9 The INFORMATION_SCHEMA ENGINES Table 4379
25.10 The INFORMATION_SCHEMA EVENTS Table 4381
25.11 The INFORMATION_SCHEMA FILES Table 4384
25.12 The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table 4392
25.13 The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table 4394
25.14 The INFORMATION_SCHEMA KEYWORDS Table 4395
25.15 The INFORMATION_SCHEMA OPTIMIZER_TRACE Table 4396
25.16 The INFORMATION_SCHEMA PARAMETERS Table 4396
25.17 The INFORMATION_SCHEMA PARTITIONS Table 4398
25.18 The INFORMATION_SCHEMA PLUGINS Table 4401
25.19 The INFORMATION_SCHEMA PROCESSLIST Table 4402
25.20 The INFORMATION_SCHEMA PROFILING Table 4404
25.21 The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table 4405
25.22 The INFORMATION_SCHEMA RESOURCE_GROUPS Table 4406
25.23 The INFORMATION_SCHEMA ROUTINES Table 4406
25.24 The INFORMATION_SCHEMA SCHEMATA Table 4409
25.25 The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table 4410

25.26 The INFORMATION_SCHEMA STATISTICS Table 4411
25.27 The INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS Table 4413
25.28 The INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS Table 4414
25.29 The INFORMATION_SCHEMA ST_UNITS_OF_MEASURE Table 4415
25.30 The INFORMATION_SCHEMA TABLES Table 4416
25.31 The INFORMATION_SCHEMA TABLESPACES Table 4420
25.32 The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table 4421
25.33 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table 4421
25.34 The INFORMATION_SCHEMA TRIGGERS Table 4422
25.35 The INFORMATION_SCHEMA USER_PRIVILEGES Table 4424
25.36 The INFORMATION_SCHEMA VIEWS Table 4425
25.37 The INFORMATION_SCHEMA VIEW_ROUTINE_USAGE Table 4426
25.38 The INFORMATION_SCHEMA VIEW_TABLE_USAGE Table 4427
25.39 INFORMATION_SCHEMA InnoDB Tables 4428
25.39.1 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table 4428
25.39.2 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table 4431
25.39.3 The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table 4435
25.39.4 The INFORMATION_SCHEMA INNODB_CACHED_INDEXES Table 4438
25.39.5 The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables 4439
25.39.6 The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables 4441
25.39.7 The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables 4442
25.39.8 The INFORMATION_SCHEMA INNODB_COLUMNS Table 4444
25.39.9 The INFORMATION_SCHEMA INNODB_DATAFILES Table 4445
25.39.10 The INFORMATION_SCHEMA INNODB_FIELDS Table 4446
25.39.11 The INFORMATION_SCHEMA INNODB_FOREIGN Table 4447
25.39.12 The INFORMATION_SCHEMA INNODB_FOREIGN_COLS Table 4447
25.39.13 The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table 4448
25.39.14 The INFORMATION_SCHEMA INNODB_FT_CONFIG Table 4449
25.39.15 The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table 4450
25.39.16 The INFORMATION_SCHEMA INNODB_FT_DELETED Table 4451
25.39.17 The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table 4452
25.39.18 The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table 4453
25.39.19 The INFORMATION_SCHEMA INNODB_INDEXES Table 4455
25.39.20 The INFORMATION_SCHEMA INNODB_LOCKS Table 4456
25.39.21 The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table 4457
25.39.22 The INFORMATION_SCHEMA INNODB_METRICS Table 4458
25.39.23 The INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES Table 4460
25.39.24 The INFORMATION_SCHEMA INNODB_TABLES Table 4461
25.39.25 The INFORMATION_SCHEMA INNODB_TABLESPACES Table 4462
25.39.26 The INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF Table 4464
25.39.27 The INFORMATION_SCHEMA INNODB_TABLESTATS View 4465
25.39.28 The INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO Table 4467
25.39.29 The INFORMATION_SCHEMA INNODB_TRX Table 4468
25.39.30 The INFORMATION_SCHEMA INNODB_VIRTUAL Table 4471
25.40 INFORMATION_SCHEMA Thread Pool Tables 4472
25.40.1 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table 4473
25.40.2 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table 4473
25.40.3 The INFORMATION_SCHEMA TP_THREAD_STATE Table 4473
25.41 INFORMATION_SCHEMA Connection-Control Tables 4474

25.41.1 The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table 4474
25.42 Extensions to SHOW Statements 4475
26 MySQL Performance Schema 4477
26.1 Performance Schema Quick Start 4479
26.2 Performance Schema Build Configuration 4485
26.3 Performance Schema Startup Configuration 4486
26.4 Performance Schema Runtime Configuration 4488
26.4.1 Performance Schema Event Timing 4489
26.4.2 Performance Schema Event Filtering 4491
26.4.3 Event Pre-Filtering 4493
26.4.4 Pre-Filtering by Instrument 4494
26.4.5 Pre-Filtering by Object 4495
26.4.6 Pre-Filtering by Thread 4497
26.4.7 Pre-Filtering by Consumer 4499
26.4.8 Example Consumer Configurations 4502
26.4.9 Naming Instruments or Consumers for Filtering Operations 4507
26.4.10 Determining What Is Instrumented 4508
26.5 Performance Schema Queries 4509
26.6 Performance Schema Instrument Naming Conventions 4509
26.7 Performance Schema Status Monitoring 4513
26.8 Performance Schema Atom and Molecule Events 4517
26.9 Performance Schema Tables for Current and Historical Events 4517
26.10 Performance Schema Statement Digests and Sampling 4518
26.11 Performance Schema General Table Characteristics 4523
26.12 Performance Schema Table Descriptions 4524
26.12.1 Performance Schema Table Index 4525
26.12.2 Performance Schema Setup Tables 4528
26.12.3 Performance Schema Instance Tables 4536
26.12.4 Performance Schema Wait Event Tables 4542
26.12.5 Performance Schema Stage Event Tables 4547
26.12.6 Performance Schema Statement Event Tables 4553
26.12.7 Performance Schema Transaction Tables 4564
26.12.8 Performance Schema Connection Tables 4572
26.12.9 Performance Schema Connection Attribute Tables 4575
26.12.10 Performance Schema User-Defined Variable Tables 4579
26.12.11 Performance Schema Replication Tables 4580
26.12.12 Performance Schema Lock Tables 4598
26.12.13 Performance Schema System Variable Tables 4607
26.12.14 Performance Schema Status Variable Tables 4612
26.12.15 Performance Schema Thread Pool Tables 4613
26.12.16 Performance Schema Clone Tables 4619
26.12.17 Performance Schema Summary Tables 4621
26.12.18 Performance Schema Miscellaneous Tables 4650
26.13 Performance Schema Option and Variable Reference 4662
26.14 Performance Schema Command Options 4665
26.15 Performance Schema System Variables 4666
26.16 Performance Schema Status Variables 4684
26.17 The Performance Schema Memory-Allocation Model 4687
26.18 Performance Schema and Plugins 4688
26.19 Using the Performance Schema to Diagnose Problems 4688
26.19.1 Query Profiling Using Performance Schema 4690
26.19.2 Obtaining Parent Event Information 4692
26.20 Restrictions on Performance Schema 4693

27 MySQL sys Schema 4695
27.1 Prerequisites for Using the sys Schema 4695
27.2 Using the sys Schema 4696
27.3 sys Schema Progress Reporting 4697
27.4 sys Schema Object Reference 4698
27.4.1 sys Schema Object Index 4698
27.4.2 sys Schema Tables and Triggers 4702
27.4.3 sys Schema Views 4705
27.4.4 sys Schema Stored Procedures 4747
27.4.5 sys Schema Stored Functions 4767
28 Connectors and APIs 4781
28.1 MySQL Connector/C 4784
28.2 MySQL Connector/C++ 4784
28.3 MySQL Connector/J 4784
28.4 MySQL Connector/NET 4784
28.5 MySQL Connector/ODBC 4785
28.6 MySQL Connector/Python 4785
28.7 MySQL C API 4785
28.7.1 MySQL C API Implementations 4786
28.7.2 Simultaneous MySQL Server and Connector/C Installations 4786
28.7.3 Example C API Client Programs 4787
28.7.4 Building and Running C API Client Programs 4788
28.7.5 C API Data Structures 4793
28.7.6 C API Function Overview 4799
28.7.7 C API Function Descriptions 4804
28.7.8 C API Prepared Statements 4874
28.7.9 C API Prepared Statement Data Structures 4874
28.7.10 C API Prepared Statement Function Overview 4881
28.7.11 C API Prepared Statement Function Descriptions 4883
28.7.12 C API Asynchronous Interface 4908
28.7.13 C API Asynchronous Interface Data Structures 4913
28.7.14 C API Asynchronous Function Overview 4913
28.7.15 C API Asynchronous Function Descriptions 4914
28.7.16 C API Threaded Function Descriptions 4918
28.7.17 C API Client Plugin Functions 4919
28.7.18 C API Binary Log Interface 4922
28.7.19 C API Binary Log Data Structures 4922
28.7.20 C API Binary Log Function Overview 4924
28.7.21 C API Binary Log Function Descriptions 4924
28.7.22 C API Encrypted Connection Support 4927
28.7.23 C API Multiple Statement Execution Support 4928
28.7.24 C API Prepared Statement Handling of Date and Time Values 4931
28.7.25 C API Prepared CALL Statement Support 4932
28.7.26 C API Prepared Statement Problems 4936
28.7.27 C API Optional Result Set Metadata 4936
28.7.28 C API Automatic Reconnection Control 4937
28.7.29 C API Common Issues 4938
28.8 MySQL PHP API 4940
28.9 MySQL Perl API 4940
28.10 MySQL Python API 4940
28.11 MySQL Ruby APIs 4941
28.11.1 The MySQL/Ruby API 4941
28.11.2 The Ruby/MySQL API 4941
28.12 MySQL Tcl API 4941

28.13 MySQL Eiffel Wrapper 4941
29 Extending MySQL 4943
29.1 MySQL Internals 4943
29.1.1 MySQL Threads 4943
29.1.2 The MySQL Test Suite 4944
29.2 The MySQL Plugin API 4945
29.2.1 Types of Plugins 4945
29.2.2 Plugin API Characteristics 4950
29.2.3 Plugin API Components 4951
29.2.4 Writing Plugins 4952
29.3 MySQL Services for Plugins 5010
29.3.1 The Locking Service 5012
29.3.2 The Keyring Service 5018
29.4 Adding New Functions to MySQL 5020
29.4.1 Features of the User-Defined Function Interface 5021
29.4.2 Adding a New User-Defined Function 5021
29.4.3 Adding a New Native Function 5031
29.5 Debugging and Porting MySQL 5033
29.5.1 Debugging a MySQL Server 5033
29.5.2 Debugging a MySQL Client 5039
29.5.3 The LOCK_ORDER Tool 5040
29.5.4 The DBUG Package 5046
30 MySQL Enterprise Edition 5051
30.1 MySQL Enterprise Monitor Overview 5051
30.2 MySQL Enterprise Backup Overview 5052
30.3 MySQL Enterprise Security Overview 5053
30.4 MySQL Enterprise Encryption Overview 5053
30.5 MySQL Enterprise Audit Overview 5053
30.6 MySQL Enterprise Firewall Overview 5054
30.7 MySQL Enterprise Thread Pool Overview 5054
30.8 MySQL Enterprise Data Masking and De-Identification Overview 5054
31 MySQL Workbench 5055
A MySQL 8.0 Frequently Asked Questions 5057
A.1 MySQL 8.0 FAQ: General 5057
A.2 MySQL 8.0 FAQ: Storage Engines 5058
A.3 MySQL 8.0 FAQ: Server SQL Mode 5059
A.4 MySQL 8.0 FAQ: Stored Procedures and Functions 5060
A.5 MySQL 8.0 FAQ: Triggers 5064
A.6 MySQL 8.0 FAQ: Views 5067
A.7 MySQL 8.0 FAQ: INFORMATION_SCHEMA 5067
A.8 MySQL 8.0 FAQ: Migration 5068
A.9 MySQL 8.0 FAQ: Security 5069
A.10 MySQL 8.0 FAQ: NDB Cluster 5070
A.11 MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets 5071
A.12 MySQL 8.0 FAQ: Connectors & APIs 5082
A.13 MySQL 8.0 FAQ: C API, Connector/C, libmysql 5082
A.14 MySQL 8.0 FAQ: Replication 5083
A.15 MySQL 8.0 FAQ: MySQL Enterprise Thread Pool 5087
A.16 MySQL 8.0 FAQ: InnoDB Change Buffer 5088
A.17 MySQL 8.0 FAQ: InnoDB Data-at-Rest Encryption 5090
A.18 MySQL 8.0 FAQ: Virtualization Support 5092
B Errors, Error Codes, and Common Problems 5093
B.1 Error Message Sources and Components 5093
B.2 Error Information Interfaces 5096

B.3 Error Message Reference 5098
B.3.1 Server Error Message Reference 5098
B.3.2 Client Error Message Reference 5604
B.3.3 Global Error Message Reference 5609
B.4 Problems and Common Errors 5617
B.4.1 How to Determine What Is Causing a Problem 5617
B.4.2 Common Errors When Using MySQL Programs 5618
B.4.3 Administration-Related Issues 5630
B.4.4 Query-Related Issues 5639
B.4.5 Optimizer-Related Issues 5646
B.4.6 Table Definition-Related Issues 5647
B.4.7 Known Issues in MySQL 5648
C Indexes 5651
MySQL Glossary 6575

Preface and Legal Notices

This is the Reference Manual for the MySQL Database System, version 8.0, through release 8.0.19. Differences between minor versions of MySQL 8.0 are noted in the present text with reference to release numbers (8.0.*x*). For license information, see the Legal Notices.

This manual is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 8.0 and previous versions. If you are using an earlier release of the MySQL software, please refer to the appropriate manual. For example, *MySQL 5.7 Reference Manual* covers the 5.7 series of MySQL software releases.

Licensing information—MySQL 8.0. This product may include third-party software, used under license. If you are using a *Commercial* release of MySQL 8.0, see the MySQL 8.0 Commercial Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of MySQL 8.0, see the MySQL 8.0 Community Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Licensing information—MySQL NDB Cluster 8.0. If you are using a *Community* release of MySQL NDB Cluster 8.0, see the MySQL NDB Cluster 8.0 Community Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Legal Notices

Copyright © 1997, 2019, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Chapter 1 General Information

Table of Contents

1.1 About This Manual 2
1.2 Typographical and Syntax Conventions 3
1.3 Overview of the MySQL Database Management System 4
    1.3.1 What is MySQL? 4
    1.3.2 The Main Features of MySQL 6
    1.3.3 History of MySQL 9
1.4 What Is New in MySQL 8.0 9
1.5 Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 8.0 36
1.6 MySQL Information Sources 51
    1.6.1 MySQL Websites 51
    1.6.2 MySQL Community Support at the MySQL Forums 51
    1.6.3 MySQL Enterprise 52
1.7 How to Report Bugs or Problems 52
1.8 MySQL Standards Compliance 56
    1.8.1 MySQL Extensions to Standard SQL 58
    1.8.2 MySQL Differences from Standard SQL 60
    1.8.3 How MySQL Deals with Constraints 62
1.9 Credits 66
    1.9.1 Contributors to MySQL 66
    1.9.2 Documenters and translators 70
    1.9.3 Packages that support MySQL 72
    1.9.4 Tools that were used to create MySQL 72
    1.9.5 Supporters of MySQL 73

The MySQL™ software delivers a very fast, multithreaded, multi-user, and robust SQL (Structured Query Language) database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. Oracle is a registered trademark of Oracle Corporation and/or its affiliates. MySQL is a trademark of Oracle Corporation and/or its affiliates, and shall not be used by Customer without Oracle's express written authorization. Other names may be trademarks of their respective owners.

The MySQL software is Dual Licensed. Users can choose to use the MySQL software as an Open Source product under the terms of the GNU General Public License (http://www.fsf.org/licenses/) or can purchase a standard commercial license from Oracle. See http://www.mysql.com/company/legal/licensing/ for more information on our licensing policies.

The following list describes some sections of particular interest in this manual:

- For a discussion of MySQL Database Server capabilities, see Section 1.3.2, “The Main Features of MySQL”.
- For an overview of new MySQL features, see Section 1.4, “What Is New in MySQL 8.0”. For information about the changes in each version, see the Release Notes.
- For installation instructions, see Chapter 2, *Installing and Upgrading MySQL*. For information about upgrading MySQL, see Section 2.11, “Upgrading MySQL”.
- For a tutorial introduction to the MySQL Database Server, see Chapter 3, *Tutorial*.

- For information about configuring and administering MySQL Server, see Chapter 5, *MySQL Server Administration*.
- For information about security in MySQL, see Chapter 6, *Security*.
- For information about setting up replication servers, see Chapter 17, *Replication*.
- For information about MySQL Enterprise, the commercial MySQL release with advanced features and management tools, see Chapter 30, *MySQL Enterprise Edition*.
- For answers to a number of questions that are often asked concerning the MySQL Database Server and its capabilities, see Appendix A, *MySQL 8.0 Frequently Asked Questions*.
- For a history of new features and bug fixes, see the Release Notes.

Important

To report problems or bugs, please use the instructions at Section 1.7, "How to Report Bugs or Problems". If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to `<secalert_us@oracle.com>`. Exception: Support customers should report all problems, including security bugs, to Oracle Support.

1.1 About This Manual

This is the Reference Manual for the MySQL Database System, version 8.0, through release 8.0.19. Differences between minor versions of MySQL 8.0 are noted in the present text with reference to release numbers (8.0.*x*). For license information, see the Legal Notices.

This manual is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 8.0 and previous versions. If you are using an earlier release of the MySQL software, please refer to the appropriate manual. For example, *MySQL 5.7 Reference Manual* covers the 5.7 series of MySQL software releases.

Because this manual serves as a reference, it does not provide general instruction on SQL or relational database concepts. It also does not teach you how to use your operating system or command-line interpreter.

The MySQL Database Software is under constant development, and the Reference Manual is updated frequently as well. The most recent version of the manual is available online in searchable form at https://dev.mysql.com/doc/. Other formats also are available there, including HTML, PDF, and EPUB versions.

The Reference Manual source files are written in DocBook XML format. The HTML version and other formats are produced automatically, primarily using the DocBook XSL stylesheets. For information about DocBook, see http://docbook.org/

The source code for MySQL itself contains internal documentation written using Doxygen. The generated Doxygen content is available https://dev.mysql.com/doc/index-other.html. It is also possible to generate this content locally from a MySQL source distribution using the instructions at Section 2.9.10, "Generating MySQL Doxygen Documentation Content".

If you have questions about using MySQL, join the MySQL Community Slack, or ask in our forums; see Section 1.6.2, "MySQL Community Support at the MySQL Forums". If you have suggestions concerning additions or corrections to the manual itself, please send them to the http://www.mysql.com/company/contact/.

This manual was originally written by David Axmark and Michael “Monty” Widenius. It is maintained by the MySQL Documentation Team, consisting of Chris Cole, Paul DuBois, Margaret Fisher, Edward Gilmore, Stefan Hinz, David Moss, Philip Olson, Daniel Price, Daniel So, and Jon Stephens.

1.2 Typographical and Syntax Conventions

This manual uses certain typographical conventions:

- `Text in this style` is used for SQL statements; database, table, and column names; program listings and source code; and environment variables. Example: “To reload the grant tables, use the `FLUSH PRIVILEGES` statement.”
- **`Text in this style`** indicates input that you type in examples.
- `Text in this style` indicates the names of executable programs and scripts, examples being `mysql` (the MySQL command-line client program) and `mysqld` (the MySQL server executable).
- *`Text in this style`* is used for variable input for which you should substitute a value of your own choosing.
- *Text in this style* is used for emphasis.
- **Text in this style** is used in table headings and to convey especially strong emphasis.
- `Text in this style` is used to indicate a program option that affects how the program is executed, or that supplies information that is needed for the program to function in a certain way. *Example*: “The `--host` option (short form `-h`) tells the `mysql` client program the hostname or IP address of the MySQL server that it should connect to”.
- File names and directory names are written like this: “The global `my.cnf` file is located in the `/etc` directory.”
- Character sequences are written like this: “To specify a wildcard, use the ‘`%`’ character.”

When commands are shown that are meant to be executed from within a particular program, the prompt shown preceding the command indicates which command to use. For example, `shell>` indicates a command that you execute from your login shell, `root-shell>` is similar but should be executed as `root`, and `mysql>` indicates a statement that you execute from the `mysql` client program:

```
shell> type a shell command here
root-shell> type a shell command as root here
mysql> type a mysql statement here
```

In some areas different systems may be distinguished from each other to show that commands should be executed in two different environments. For example, while working with replication the commands might be prefixed with `master` and `slave`:

```
master> type a mysql command on the replication master here
slave> type a mysql command on the replication slave here
```

The “shell” is your command interpreter. On Unix, this is typically a program such as `sh`, `csh`, or `bash`. On Windows, the equivalent program is `command.com` or `cmd.exe`, typically run in a console window.

When you enter a command or statement shown in an example, do not type the prompt shown in the example.

Database, table, and column names must often be substituted into statements. To indicate that such substitution is necessary, this manual uses *`db_name`*, *`tbl_name`*, and *`col_name`*. For example, you might see a statement like this:

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

This means that if you were to enter a similar statement, you would supply your own database, table, and column names, perhaps like this:

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL keywords are not case-sensitive and may be written in any lettercase. This manual uses uppercase.

In syntax descriptions, square brackets (“`[`” and “`]`”) indicate optional words or clauses. For example, in the following statement, `IF EXISTS` is optional:

```
DROP TABLE [IF EXISTS] tbl_name
```

When a syntax element consists of a number of alternatives, the alternatives are separated by vertical bars (“`|`”). When one member from a set of choices *may* be chosen, the alternatives are listed within square brackets (“`[`” and “`]`”):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

When one member from a set of choices *must* be chosen, the alternatives are listed within braces (“`{`” and “`}`”):

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

An ellipsis (`...`) indicates the omission of a section of a statement, typically to provide a shorter version of more complex syntax. For example, `SELECT ... INTO OUTFILE` is shorthand for the form of `SELECT` statement that has an `INTO OUTFILE` clause following other parts of the statement.

An ellipsis can also indicate that the preceding syntax element of a statement may be repeated. In the following example, multiple *`reset_option`* values may be given, with each of those after the first preceded by commas:

```
RESET reset_option [,reset_option] ...
```

Commands for setting shell variables are shown using Bourne shell syntax. For example, the sequence to set the `CC` environment variable and run the `configure` command looks like this in Bourne shell syntax:

```
shell> CC=gcc ./configure
```

If you are using `csh` or `tcsh`, you must issue commands somewhat differently:

```
shell> setenv CC gcc
shell> ./configure
```

1.3 Overview of the MySQL Database Management System

1.3.1 What is MySQL?

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by Oracle Corporation.

The MySQL website (http://www.mysql.com/) provides the latest information about MySQL software.

- **MySQL is a database management system.**

  A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities, or as parts of other applications.

- **MySQL databases are relational.**

  A relational database stores data in separate tables rather than putting all the data in one big storeroom. The database structures are organized into physical files optimized for speed. The logical model, with objects such as databases, tables, views, rows, and columns, offers a flexible programming environment. You set up rules governing the relationships between different data fields, such as one-to-one, one-to-many, unique, required or optional, and “pointers” between different tables. The database enforces these rules, so that with a well-designed database, your application never sees inconsistent, duplicate, orphan, out-of-date, or missing data.

  The SQL part of “MySQL” stands for “Structured Query Language”. SQL is the most common standardized language used to access databases. Depending on your programming environment, you might enter SQL directly (for example, to generate reports), embed SQL statements into code written in another language, or use a language-specific API that hides the SQL syntax.

  SQL is defined by the ANSI/ISO SQL Standard. The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992, “SQL:1999” refers to the standard released in 1999, and “SQL:2003” refers to the current version of the standard. We use the phrase “the SQL standard” to mean the current version of the SQL Standard at any time.

- **MySQL software is Open Source.**

  Open Source means that it is possible for anyone to use and modify the software. Anybody can download the MySQL software from the Internet and use it without paying anything. If you wish, you may study the source code and change it to suit your needs. The MySQL software uses the GPL (GNU General Public License), http://www.fsf.org/licenses/, to define what you may and may not do with the software in different situations. If you feel uncomfortable with the GPL or need to embed MySQL code into a commercial application, you can buy a commercially licensed version from us. See the MySQL Licensing Overview for more information (http://www.mysql.com/company/legal/licensing/).

- **The MySQL Database Server is very fast, reliable, scalable, and easy to use.**

  If that is what you are looking for, you should give it a try. MySQL Server can run comfortably on a desktop or laptop, alongside your other applications, web servers, and so on, requiring little or no attention. If you dedicate an entire machine to MySQL, you can adjust the settings to take advantage of all the memory, CPU power, and I/O capacity available. MySQL can also scale up to clusters of machines, networked together.

  MySQL Server was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. Although under constant development, MySQL Server today offers a rich and useful set of functions. Its connectivity, speed, and security make MySQL Server highly suited for accessing databases on the Internet.

- **MySQL Server works in client/server or embedded systems.**

  The MySQL Database Software is a client/server system that consists of a multithreaded SQL server that supports different back ends, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs).

  We also provide MySQL Server as an embedded multithreaded library that you can link into your application to get a smaller, faster, easier-to-manage standalone product.

- **A large amount of contributed MySQL software is available.**

  MySQL Server has a practical set of features developed in close cooperation with our users. It is very likely that your favorite application or language supports the MySQL Database Server.

The official way to pronounce “MySQL” is “My Ess Que Ell” (not “my sequel”), but we do not mind if you pronounce it as “my sequel” or in some other localized way.

1.3.2 The Main Features of MySQL

This section describes some of the important characteristics of the MySQL Database Software. In most respects, the roadmap applies to all versions of MySQL. For information about features as they are introduced into MySQL on a series-specific basis, see the “In a Nutshell” section of the appropriate Manual:

- MySQL 8.0: Section 1.4, “What Is New in MySQL 8.0”
- MySQL 5.7: What Is New in MySQL 5.7
- MySQL 5.6: What Is New in MySQL 5.6
- MySQL 5.5: What Is New in MySQL 5.5

Internals and Portability

- Written in C and C++.
- Tested with a broad range of different compilers.
- Works on many different platforms. See https://www.mysql.com/support/supportedplatforms/database.html.
- For portability, uses `CMake` in MySQL 5.5 and up. Previous series use GNU Automake, Autoconf, and Libtool.
- Tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool (http://developer.kde.org/~sewardj/).
- Uses multi-layered server design with independent modules.
- Designed to be fully multithreaded using kernel threads, to easily use multiple CPUs if they are available.
- Provides transactional and nontransactional storage engines.
- Uses very fast B-tree disk tables (`MyISAM`) with index compression.
- Designed to make it relatively easy to add other storage engines. This is useful if you want to provide an SQL interface for an in-house database.
- Uses a very fast thread-based memory allocation system.
- Executes very fast joins using an optimized nested-loop join.

- Implements in-memory hash tables, which are used as temporary tables.
- Implements SQL functions using a highly optimized class library that should be as fast as possible. Usually there is no memory allocation at all after query initialization.
- Provides the server as a separate program for use in a client/server networked environment, and as a library that can be embedded (linked) into standalone applications. Such applications can be used in isolation or in environments where no network is available.

Data Types

- Many data types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long, `FLOAT`, `DOUBLE`, `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `TEXT`, `BLOB`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, `YEAR`, `SET`, `ENUM`, and OpenGIS spatial types. See Chapter 11, *Data Types*.
- Fixed-length and variable-length string types.

Statements and Functions

- Full operator and function support in the `SELECT` list and `WHERE` clause of queries. For example:

```
mysql> SELECT CONCAT(first_name, ' ', last_name)
    -> FROM citizen
    -> WHERE income/dependents > 10000 AND age > 30;
```

- Full support for SQL `GROUP BY` and `ORDER BY` clauses. Support for group functions (`COUNT()`, `AVG()`, `STD()`, `SUM()`, `MAX()`, `MIN()`, and `GROUP_CONCAT()`).
- Support for `LEFT OUTER JOIN` and `RIGHT OUTER JOIN` with both standard SQL and ODBC syntax.
- Support for aliases on tables and columns as required by standard SQL.
- Support for `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` to return the number of rows that were changed (affected), or to return the number of rows matched instead by setting a flag when connecting to the server.
- Support for MySQL-specific `SHOW` statements that retrieve information about databases, storage engines, tables, and indexes. Support for the `INFORMATION_SCHEMA` database, implemented according to standard SQL.
- An `EXPLAIN` statement to show how the optimizer resolves a query.
- Independence of function names from table or column names. For example, `ABS` is a valid column name. The only restriction is that for a function call, no spaces are permitted between the function name and the "`(`" that follows it. See Section 9.3, "Keywords and Reserved Words".
- You can refer to tables from different databases in the same statement.

Security

- A privilege and password system that is very flexible and secure, and that enables host-based verification.
- Password security by encryption of all password traffic when you connect to a server.

Scalability and Limits

- Support for large databases. We use MySQL Server with databases that contain 50 million records. We also know of users who use MySQL Server with 200,000 tables and about 5,000,000,000 rows.

- Support for up to 64 indexes per table. Each index may consist of 1 to 16 columns or parts of columns. The maximum index width for `InnoDB` tables is either 767 bytes or 3072 bytes. See Section 15.6.1.6, “Limits on InnoDB Tables”. The maximum index width for `MyISAM` tables is 1000 bytes. See Section 16.2, “The MyISAM Storage Engine”. An index may use a prefix of a column for `CHAR`, `VARCHAR`, `BLOB`, or `TEXT` column types.

Connectivity

- Clients can connect to MySQL Server using several protocols:
  - Clients can connect using TCP/IP sockets on any platform.
  - On Windows systems, clients can connect using named pipes if the server is started with the `named_pipe` system variable enabled. Windows servers also support shared-memory connections if started with the `shared_memory` system variable enabled. Clients can connect through shared memory by using the `--protocol=memory` option.
  - On Unix systems, clients can connect using Unix domain socket files.
- MySQL client programs can be written in many languages. A client library written in C is available for clients written in C or C++, or for any language that provides C bindings.
- APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available, enabling MySQL clients to be written in many languages. See Chapter 28, *Connectors and APIs*.
- The Connector/ODBC (MyODBC) interface provides MySQL support for client programs that use ODBC (Open Database Connectivity) connections. For example, you can use MS Access to connect to your MySQL server. Clients can be run on Windows or Unix. Connector/ODBC source is available. All ODBC 2.5 functions are supported, as are many others. See MySQL Connector/ODBC Developer Guide.
- The Connector/J interface provides MySQL support for Java client programs that use JDBC connections. Clients can be run on Windows or Unix. Connector/J source is available. See MySQL Connector/J 5.1 Developer Guide.
- MySQL Connector/NET enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and integrates into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. MySQL Connector/NET is a fully managed ADO.NET driver written in 100% pure C#. See MySQL Connector/NET Developer Guide.

Localization

- The server can provide error messages to clients in many languages. See Section 10.12, “Setting the Error Message Language”.
- Full support for several different character sets, including `latin1` (cp1252), `german`, `big5`, `ujis`, several Unicode character sets, and more. For example, the Scandinavian characters “`å`”, “`ä`” and “`ö`” are permitted in table and column names.
- All data is saved in the chosen character set.
- Sorting and comparisons are done according to the default character set and collation. is possible to change this when the MySQL server is started (see Section 10.3.2, “Server Character Set and Collation”). To see an example of very advanced sorting, look at the Czech sorting code. MySQL Server supports many different character sets that can be specified at compile time and runtime.
- The server time zone can be changed dynamically, and individual clients can specify their own time zone. See Section 5.1.13, “MySQL Server Time Zone Support”.

Clients and Tools

- MySQL includes several client and utility programs. These include both command-line programs such as `mysqldump` and `mysqladmin`, and graphical programs such as MySQL Workbench.
- MySQL Server has built-in support for SQL statements to check, optimize, and repair tables. These statements are available from the command line through the `mysqlcheck` client. MySQL also includes `myisamchk`, a very fast command-line utility for performing these operations on `MyISAM` tables. See Chapter 4, *MySQL Programs*.
- MySQL programs can be invoked with the `--help` or `-?` option to obtain online assistance.

1.3.3 History of MySQL

We started out with the intention of using the `mSQL` database system to connect to our tables using our own fast low-level (ISAM) routines. However, after some testing, we came to the conclusion that `mSQL` was not fast enough or flexible enough for our needs. This resulted in a new SQL interface to our database but with almost the same API interface as `mSQL`. This API was designed to enable third-party code that was written for use with `mSQL` to be ported easily for use with MySQL.

MySQL is named after co-founder Monty Widenius's daughter, My.

The name of the MySQL Dolphin (our logo) is “Sakila,” which was chosen from a huge list of names suggested by users in our “Name the Dolphin” contest. The winning name was submitted by Ambrose Twebaze, an Open Source software developer from Swaziland, Africa. According to Ambrose, the feminine name Sakila has its roots in SiSwati, the local language of Swaziland. Sakila is also the name of a town in Arusha, Tanzania, near Ambrose's country of origin, Uganda.

1.4 What Is New in MySQL 8.0

This section summarizes what has been added to, deprecated in, and removed from MySQL 8.0. A companion section lists MySQL server options and variables that have been added, deprecated, or removed in MySQL 8.0. See Section 1.5, “Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 8.0”.

- Features Added in MySQL 8.0
- Features Deprecated in MySQL 8.0
- Features Removed in MySQL 8.0

Features Added in MySQL 8.0

The following features have been added to MySQL 8.0:

- **Data dictionary.** MySQL now incorporates a transactional data dictionary that stores information about database objects. In previous MySQL releases, dictionary data was stored in metadata files and nontransactional tables. For more information, see Chapter 14, *MySQL Data Dictionary*.
- **Atomic data definition statements (Atomic DDL).** An atomic DDL statement combines the data dictionary updates, storage engine operations, and binary log writes associated with a DDL operation into a single, atomic transaction. For more information, see Section 13.1.1, “Atomic Data Definition Statement Support”.
- **Upgrade procedure.** Previously, after installation of a new version of MySQL, the MySQL server automatically upgrades the data dictionary tables at the next startup, after which the DBA is expected to invoke `mysql_upgrade` manually to upgrade the system tables in the `mysql` schema, as well as objects in other schemas such as the `sys` schema and user schemas.

As of MySQL 8.0.16, the server performs the tasks previously handled by `mysql_upgrade`. After installation of a new MySQL version, the server now automatically performs all necessary upgrade tasks at the next startup and is not dependent on the DBA invoking `mysql_upgrade`. In addition, the server updates the contents of the help tables (something `mysql_upgrade` did not do). A new `--upgrade` server option provides control over how the server performs automatic data dictionary and server upgrade operations. For more information, see Section 2.11.3, "What the MySQL Upgrade Process Upgrades".

- **Security and account management.** These enhancements were added to improve security and enable greater DBA flexibility in account management:

  - The grant tables in the `mysql` system database are now `InnoDB` (transactional) tables. Previously, these were `MyISAM` (nontransactional) tables. The change of grant table storage engine underlies an accompanying change to the behavior of account-management statements. Previously, an account-management statement (such as `CREATE USER` or `DROP USER`) that named multiple users could succeed for some users and fail for others. Now, each statement is transactional and either succeeds for all named users or rolls back and has no effect if any error occurs. The statement is written to the binary log if it succeeds, but not if it fails; in that case, rollback occurs and no changes are made. For more information, see Section 13.1.1, "Atomic Data Definition Statement Support".

  - A new `caching_sha2_password` authentication plugin is available. Like the `sha256_password` plugin, `caching_sha2_password` implements SHA-256 password hashing, but uses caching to address latency issues at connect time. It also supports more connection protocols and does not require linking against OpenSSL for RSA key pair-based password-exchange capabilities. See Section 6.4.1.2, "Caching SHA-2 Pluggable Authentication".

    The `caching_sha2_password` and `sha256_password` authentication plugins provide more secure password encryption than the `mysql_native_password` plugin, and `caching_sha2_password` provides better performance than `sha256_password`. Due to these superior security and performance characteristics of `caching_sha2_password`, it is now the preferred authentication plugin, and is also the default authentication plugin rather than `mysql_native_password`. For information about the implications of this change of default plugin for server operation and compatibility of the server with clients and connectors, see caching_sha2_password as the Preferred Authentication Plugin.

  - MySQL now supports roles, which are named collections of privileges. Roles can be created and dropped. Roles can have privileges granted to and revoked from them. Roles can be granted to and revoked from user accounts. The active applicable roles for an account can be selected from among those granted to the account, and can be changed during sessions for that account. For more information, see Section 6.2.10, "Using Roles".

  - MySQL now incorporates the concept of user account categories, with system and regular users distinguished according to whether they have the `SYSTEM_USER` privilege. See Section 6.2.11, "Account Categories".

  - Previously, it was not possible to grant privileges that apply globally except for certain schemas. This is now possible if the `partial_revokes` system variable is enabled. See Section 6.2.12, "Privilege Restriction Using Partial Revokes".

  - The `GRANT` statement has an `AS *user* [WITH ROLE]` clause that specifies additional information about the privilege context to use for statement execution. This syntax is visible at the SQL level, although its primary purpose is to enable uniform replication across all nodes of grantor privilege restrictions imposed by partial revokes, by causing those restrictions to appear in the binary log. See Section 13.7.1.6, "GRANT Syntax".

- MySQL now maintains information about password history, enabling restrictions on reuse of previous passwords. DBAs can require that new passwords not be selected from previous passwords for some number of password changes or period of time. It is possible to establish password-reuse policy globally as well as on a per-account basis.

  It is now possible to require that attempts to change account passwords be verified by specifying the current password to be replaced. This enables DBAs to prevent users from changing password without proving that they know the current password. It is possible to establish password-verification policy globally as well as on a per-account basis.

  Accounts are now permitted to have dual passwords, which enables phased password changes to be performed seamlessly in complex multiple-server systems, without downtime.

  These new capabilities provide DBAs more complete control over password management. For more information, see Section 6.2.15, “Password Management”.

- MySQL now supports FIPS mode, if compiled using OpenSSL, and an OpenSSL library and FIPS Object Module are available at runtime. FIPS mode imposes conditions on cryptographic operations such as restrictions on acceptable encryption algorithms or requirements for longer key lengths. See Section 6.5, “FIPS Support”.

- The SSL context the server uses for new connections now is reconfigurable at runtime. This capability may be useful, for example, to avoid restarting a MySQL server that has been running so long that its SSL certificate has expired. See Server-Side Runtime Configuration for Encrypted Connections.

- OpenSSL 1.1.1 supports the TLS v1.3 protocol for encrypted connections, and MySQL 8.0.16 and higher supports TLS v1.3 as well, if both the server and client are compiled using OpenSSL 1.1.1 or higher. See Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- MySQL now sets the access control granted to clients on the named pipe to the minimum necessary for successful communication on Windows. Newer MySQL client software can open named pipe connections without any additional configuration. If older client software cannot be upgraded immediately, the new `named_pipe_full_access_group` system variable can be used to give a Windows group the necessary permissions to open a named pipe connection. Membership in the full-access group should be restricted and temporary.

- **Resource management.** MySQL now supports creation and management of resource groups, and permits assigning threads running within the server to particular groups so that threads execute according to the resources available to the group. Group attributes enable control over its resources, to enable or restrict resource consumption by threads in the group. DBAs can modify these attributes as appropriate for different workloads. Currently, CPU time is a manageable resource, represented by the concept of “virtual CPU” as a term that includes CPU cores, hyperthreads, hardware threads, and so forth. The server determines at startup how many virtual CPUs are available, and database administrators with appropriate privileges can associate these CPUs with resource groups and assign threads to groups. For more information, see Section 8.12.5, “Resource Groups”.

- **Table encryption management.** Table encryption can now be managed globally by defining and enforcing encryption defaults. The `default_table_encryption` variable defines an encryption default for newly created schemas and general tablespace. The encryption default for a schema can also be defined using the `DEFAULT ENCRYPTION` clause when creating a schema. By default, a table inherits the encryption of the schema or general tablespace it is created in. Encryption defaults are enforced by enabling the `table_encryption_privilege_check` variable. The privilege check occurs when creating or altering a schema or general tablespace with an encryption setting that differs from the `default_table_encryption` setting, or when creating or altering a table with an encryption setting that differs from the default schema encryption. The `TABLE_ENCRYPTION_ADMIN`

privilege permits overriding default encryption settings when `table_encryption_privilege_check` is enabled. For more information, see Defining an Encryption Default for Schemas and General Tablespaces.

- **InnoDB enhancements.** These `InnoDB` enhancements were added:

  - The current maximum auto-increment counter value is written to the redo log each time the value changes, and saved to an engine-private system table on each checkpoint. These changes make the current maximum auto-increment counter value persistent across server restarts. Additionally:

    - A server restart no longer cancels the effect of the `AUTO_INCREMENT = N` table option. If you initialize the auto-increment counter to a specific value, or if you alter the auto-increment counter value to a larger value, the new value is persisted across server restarts.

    - A server restart immediately following a `ROLLBACK` operation no longer results in the reuse of auto-increment values that were allocated to the rolled-back transaction.

    - If you modify an `AUTO_INCREMENT` column value to a value larger than the current maximum auto-increment value (in an `UPDATE` operation, for example), the new value is persisted, and subsequent `INSERT` operations allocate auto-increment values starting from the new, larger value.

    For more information, see Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”, and InnoDB AUTO_INCREMENT Counter Initialization.

  - When encountering index tree corruption, `InnoDB` writes a corruption flag to the redo log, which makes the corruption flag crash safe. `InnoDB` also writes in-memory corruption flag data to an engine-private system table on each checkpoint. During recovery, `InnoDB` reads corruption flags from both locations and merges results before marking in-memory table and index objects as corrupt.

  - The `InnoDB` `memcached` plugin supports multiple `get` operations (fetching multiple key-value pairs in a single `memcached` query) and range queries. See Section 15.19.4, “InnoDB memcached Multiple get and Range Query Support”.

  - A new dynamic variables, `innodb_deadlock_detect`, may be used to disable deadlock detection. On high concurrency systems, deadlock detection can cause a slowdown when numerous threads wait for the same lock. At times, it may be more efficient to disable deadlock detection and rely on the `innodb_lock_wait_timeout` setting for transaction rollback when a deadlock occurs.

  - The new `INFORMATION_SCHEMA.INNODB_CACHED_INDEXES` table reports the number of index pages cached in the `InnoDB` buffer pool for each index.

  - `InnoDB` temporary tables are now created in the shared temporary tablespace, `ibtmp1`.

  - The `InnoDB` tablespace encryption feature supports encryption of redo log and undo log data. See Redo Log Encryption, and Undo Log Encryption.

  - `InnoDB` supports `NOWAIT` and `SKIP LOCKED` options with `SELECT ... FOR SHARE` and `SELECT ... FOR UPDATE` locking read statements. `NOWAIT` causes the statement to return immediately if a requested row is locked by another transaction. `SKIP LOCKED` removes locked rows from the result set. See Locking Read Concurrency with NOWAIT and SKIP LOCKED.

    `SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE`, but `LOCK IN SHARE MODE` remains available for backward compatibility. The statements are equivalent. However, `FOR UPDATE` and `FOR SHARE` support `NOWAIT`, `SKIP LOCKED`, and `OF tbl_name` options. See Section 13.2.10, “SELECT Syntax”.

`OF tbl_name` applies locking queries to named tables.

- `ADD PARTITION`, `DROP PARTITION`, `COALESCE PARTITION`, `REORGANIZE PARTITION`, and `REBUILD PARTITION` `ALTER TABLE` options are supported by native partitioning in-place APIs and may be used with `ALGORITHM={COPY|INPLACE}` and `LOCK` clauses.

  `DROP PARTITION` with `ALGORITHM=INPLACE` deletes data stored in the partition and drops the partition. However, `DROP PARTITION` with `ALGORITHM=COPY` or `old_alter_table=ON` rebuilds the partitioned table and attempts to move data from the dropped partition to another partition with a compatible `PARTITION ... VALUES` definition. Data that cannot be moved to another partition is deleted.

- The `InnoDB` storage engine now uses the MySQL data dictionary rather than its own storage engine-specific data dictionary. For information about the data dictionary, see Chapter 14, *MySQL Data Dictionary*.

- `mysql` system tables and data dictionary tables are now created in a single `InnoDB` tablespace file named `mysql.ibd` in the MySQL data directory. Previously, these tables were created in individual `InnoDB` tablespace files in the `mysql` database directory.

- The following undo tablespace changes are introduced in MySQL 8.0:

  - By default, undo logs now reside in two undo tablespaces that are created when the MySQL instance is initialized. Undo logs are no longer created in the system tablespace.

  - As of MySQL 8.0.14, additional undo tablespaces can be created in a chosen location at runtime using `CREATE UNDO TABLESPACE` syntax.

    ```
    CREATE UNDO TABLESPACE tablespace_name ADD DATAFILE 'file_name.ibu';
    ```

    Undo tablespaces created using `CREATE UNDO TABLESPACE` syntax can be dropped at runtime using `DROP UNDO TABLESPACE` syntax.

    ```
    DROP UNDO TABLESPACE tablespace_name;
    ```

    `ALTER UNDO TABLESPACE` syntax can be used to mark an undo tablespace as active or inactive.

    ```
    ALTER UNDO TABLESPACE tablespace_name SET {ACTIVE|INACTIVE};
    ```

    A `STATE` column that shows the state of a tablespace was added to the `INFORMATION_SCHEMA.INNODB_TABLESPACES` table. An undo tablespace must be in an `empty` state before it can be dropped.

  - The `innodb_undo_log_truncate` variable is enabled by default.

  - The `innodb_rollback_segments` variable defines the number of rollback segments per undo tablespace. Previously, `innodb_rollback_segments` specified the total number of rollback segments for the MySQL instance. This change increases the number of rollback segments available for concurrent transactions. More rollback segments increases the likelihood that concurrent transactions use separate rollback segments for undo logs, resulting in less resource contention.

- Default values for variables that affect buffer pool preflushing and flushing behavior were modified:

  - The `innodb_max_dirty_pages_pct_lwm` default value is now 10. The previous default value of 0 disables buffer pool preflushing. A value of 10 enables preflushing when the percentage of dirty pages in the buffer pool exceeds 10%. Enabling preflushing improves performance consistency.

  - The `innodb_max_dirty_pages_pct` default value was increased from 75 to 90. `InnoDB` attempts to flush data from the buffer pool so that the percentage of dirty pages does not exceed this value. The increased default value permits a greater percentage of dirty pages in the buffer pool.

- The default `innodb_autoinc_lock_mode` setting is now 2 (interleaved). Interleaved lock mode permits the execution of multi-row inserts in parallel, which improves concurrency and scalability. The new `innodb_autoinc_lock_mode` default setting reflects the change from statement-based replication to row based replication as the default replication type in MySQL 5.7. Statement-based replication requires the consecutive auto-increment lock mode (the previous default) to ensure that auto-increment values are assigned in a predictable and repeatable order for a given sequence of SQL statements, whereas row-based replication is not sensitive to the execution order of SQL statements. For more information, see InnoDB AUTO_INCREMENT Lock Modes.

  For systems that use statement-based replication, the new `innodb_autoinc_lock_mode` default setting may break applications that depend on sequential auto-increment values. To restore the previous default, set `innodb_autoinc_lock_mode` to 1.

- Renaming a general tablespace is supported by `ALTER TABLESPACE ... RENAME TO` syntax.

- The new `innodb_dedicated_server` variable, which is disabled by default, can be used to have `InnoDB` automatically configure the following options according to the amount of memory detected on the server:

  - `innodb_buffer_pool_size`

  - `innodb_log_file_size`

  - `innodb_flush_method`

  This option is intended for MySQL server instances that run on a dedicated server. For more information, see Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

- The new `INFORMATION_SCHEMA.INNODB_TABLESPACES_BRIEF` view provides space, name, path, flag, and space type data for `InnoDB` tablespaces.

- The zlib library version bundled with MySQL was raised from version 1.2.3 to version 1.2.11. MySQL implements compression with the help of the zlib library.

  If you use `InnoDB` compressed tables, see Section 2.11.4, “Changes in MySQL 8.0” for related upgrade implications.

- Serialized dictionary information (SDI) is present in all `InnoDB` tablespace files except for global temporary tablespace and undo tablespace files. SDI is serialized metadata for table and tablespace objects. The presence of SDI data provides metadata redundancy. For example, dictionary object metadata may be extracted from tablespace files if the data dictionary becomes unavailable. SDI extraction is performed using the `ibd2sdi` tool. SDI data is stored in `JSON` format.

  The inclusion of SDI data in tablespace files increases tablespace file size. An SDI record requires a single index page, which is 16KB in size by default. However, SDI data is compressed when it is stored to reduce the storage footprint.

- The `InnoDB` storage engine now supports atomic DDL, which ensures that DDL operations are either fully committed or rolled back, even if the server halts during the operation. For more information, see Section 13.1.1, “Atomic Data Definition Statement Support”.
- Tablespace files can be moved or restored to a new location while the server is offline using the `innodb_directories` option. For more information, see Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”.
- The following redo logging optimizations were implemented:
  - User threads can now write concurrently to the log buffer without synchronizing writes.
  - User threads can now add dirty pages to the flush list in a relaxed order.
  - A dedicated log thread is now responsible for writing the log buffer to the system buffers, flushing system buffers to disk, notifying user threads about written and flushed redo, maintaining the lag required for the relaxed flush list order, and write checkpoints.
  - System variables were added for configuring the use of spin delay by user threads waiting for flushed redo:
    - `innodb_log_wait_for_flush_spin_hwm`: Defines the maximum average log flush time beyond which user threads no longer spin while waiting for flushed redo.
    - `innodb_log_spin_cpu_abs_lwm`: Defines the minimum amount of CPU usage below which user threads no longer spin while waiting for flushed redo.
    - `innodb_log_spin_cpu_pct_hwm`: Defines the maximum amount of CPU usage above which user threads no longer spin while waiting for flushed redo.
  - The `innodb_log_buffer_size` variable is now dynamic, which permits resizing of the log buffer while the server is running.

  For more information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.
- As of MySQL 8.0.12, undo logging is supported for small updates to large object (LOB) data, which improves performance of LOB updates that are 100 bytes in size or less. Previously, LOB updates were a minimum of one LOB page in size, which is less than optimal for updates that might only modify a few bytes. This enhancement builds upon support added in MySQL 8.0.4 for partial update of LOB data.
- As of MySQL 8.0.12, `ALGORITHM=INSTANT` is supported for the following `ALTER TABLE` operations:
  - Adding a column. This feature is also referred to as “Instant `ADD COLUMN`”. Limitations apply. See Section 15.12.1, “Online DDL Operations”.
  - Adding or dropping a virtual column.
  - Adding or dropping a column default value.
  - Modifying the definition of an `ENUM` or `SET` column.
  - Changing the index type.
  - Renaming a table.

Operations that support `ALGORITHM=INSTANT` only modify metadata in the data dictionary. No metadata locks are taken on the table, and table data is unaffected, making the operations instantaneous. If not specified explicitly, `ALGORITHM=INSTANT` is used by default by operations that support it. If `ALGORITHM=INSTANT` is specified but not supported, the operation fails immediately with an error.

For more information about operations that support `ALGORITHM=INSTANT`, see Section 15.12.1, "Online DDL Operations".

- As of MySQL 8.0.13, the `TempTable` storage engine supports storage of binary large object (BLOB) type columns. This enhancement improves performance for queries that use temporary tables containing BLOB data. Previously, temporary tables that contained BLOB data were stored in the on-disk storage engine defined by `internal_tmp_disk_storage_engine`. For more information, see Section 8.4.4, "Internal Temporary Table Use in MySQL".

- As of MySQL 8.0.13, the `InnoDB` data-at-rest encryption feature supports general tablespaces. Previously, only file-per-table tablespaces could be encrypted. To support encryption of general tablespaces, `CREATE TABLESPACE` and `ALTER TABLESPACE` syntax was extended to include an `ENCRYPTION` clause.

  The `INFORMATION_SCHEMA.INNODB_TABLESPACES` table now includes an `ENCRYPTION` column that indicates whether or not a tablespace is encrypted.

  The `stage/innodb/alter tablespace (encryption)` Performance Schema stage instrument was added to permit monitoring of general tablespace encryption operations.

- Disabling the `innodb_buffer_pool_in_core_file` variable reduces the size of core files by excluding `InnoDB` buffer pool pages. To use this variable, the `core_file` variable must be enabled and the operating system must support the `MADV_DONTDUMP` non-POSIX extension to `madvise()`, which is supported in Linux 3.4 and later. For more information, see Section 15.8.3.7, "Excluding Buffer Pool Pages from Core Files".

- As of MySQL 8.0.13, user-created temporary tables and internal temporary tables created by the optimizer are stored in session temporary tablespaces that are allocated to a session from a pool of temporary tablespaces. When a session disconnects, its temporary tablespaces are truncated and released back to the pool. In previous releases, temporary tables were created in the global temporary tablespace (`ibtmp1`), which did not return disk space to the operating system after temporary tables were dropped.

  The `innodb_temp_tablespaces_dir` variable defines the location where session temporary tablespaces are created. The default location is the `#innodb_temp` directory in the data directory.

  The `INNODB_SESSION_TEMP_TABLESPACES` table provides metadata about session temporary tablespaces.

  The global temporary tablespace (`ibtmp1`) now stores rollback segments for changes made to user-created temporary tables.

- As of MySQL 8.0.14, `InnoDB` supports parallel clustered index reads, which can improve `CHECK TABLE` performance. This feature does not apply to secondary index scans. The `innodb_parallel_read_threads` session variable must be set to a value greater than 1 for parallel clustered index reads to occur. The default value is 4. The actual number of threads used to perform a parallel clustered index read is determined by the `innodb_parallel_read_threads` setting or the number of index subtrees to scan, whichever is smaller.

- As of 8.0.14, when the `innodb_dedicated_server` variable is enabled, the size and number of log files are configured according to the automatically configured buffer pool size. Previously, log file size was configured according to the amount of memory detected on the server, and the number of log files was not configured automatically.

- As of 8.0.14, the `ADD DATAFILE` clause of the `CREATE TABLESPACE` statement is optional, which permits users without the `FILE` privilege to create tablespaces. A `CREATE TABLESPACE` statement executed without an `ADD DATAFILE` clause implicitly creates a tablespace data file with a unique file name.

- By default, when the amount of memory occupied by the TempTable storage engine exceeds the memory limit defined by the `temptable_max_ram` variable, the TempTable storage engine begins allocating memory-mapped temporary files from disk. As of MySQL 8.0.16, this behavior is controlled by the `temptable_use_mmap` variable. Disabling `temptable_use_mmap` causes the TempTable storage engine to use `InnoDB` on-disk internal temporary tables instead of memory-mapped files as its overflow mechanism. For more information, see Internal Temporary Table Storage Engine.

- As of MySQL 8.0.16, the `InnoDB` data-at-rest encryption feature supports encryption of the `mysql` system tablespace. The `mysql` system tablespace contains the `mysql` system database and the MySQL data dictionary tables. For more information, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”.

- The `innodb_spin_wait_pause_multiplier` variable, introduced in MySQL 8.0.16, provides greater control over the duration of spin-lock polling delays that occur when a thread waits to acquire a mutex or rw-lock. Delays can be tuned more finely to account for differences in PAUSE instruction duration on different processor architectures. For more information, see Section 15.8.8, “Configuring Spin Lock Polling”.

- `InnoDB` parallel read thread performance for large data sets was improved in MySQL 8.0.17 through better utilization of read threads, through a reduction in read thread I/O for prefetch activity that occurs during parallel scans, and through support for parallel scanning of partitions.

  The parallel read thread feature is controlled by the `innodb_parallel_read_threads` variable. The maximum setting is now 256, which is the total number of threads for all client connections. If the thread limit is reached, connections fall back to using a single thread.

- The `innodb_idle_flush_pct` variable, introduced in MySQL 8.0.18, permits placing a limit on page flushing during idle periods, which can help extend the life of solid state storage devices. See Limiting Buffer Flushing During Idle Periods.

- **Character set support.** The default character set has changed from `latin1` to `utf8mb4`. The `utf8mb4` character set has several new collations, including `utf8mb4_ja_0900_as_cs`, the first Japanese language-specific collation available for Unicode in MySQL. For more information, see Section 10.10.1, “Unicode Character Sets”.

- **JSON enhancements.** The following enhancements or additions were made to MySQL's JSON functionality:

  - Added the `->>` (inline path) operator, which is equivalent to calling `JSON_UNQUOTE()` on the result of `JSON_EXTRACT()`.

    This is a refinement of the column path operator `->` introduced in MySQL 5.7; `col->>"$.path"` is equivalent to `JSON_UNQUOTE(col->"$.path")`. The inline path operator can be used wherever you can use `JSON_UNQUOTE(JSON_EXTRACT())`, such `SELECT` column lists, `WHERE` and `HAVING`

clauses, and `ORDER BY` and `GROUP BY` clauses. For more information, see the description of the operator, as well as JSON Path Syntax.

- Added two JSON aggregation functions `JSON_ARRAYAGG()` and `JSON_OBJECTAGG()`. `JSON_ARRAYAGG()` takes a column or expression as its argument, and aggregates the result as a single `JSON` array. The expression can evaluate to any MySQL data type; this does not have to be a `JSON` value. `JSON_OBJECTAGG()` takes two columns or expressions which it interprets as a key and a value; it returns the result as a single `JSON` object. For more information and examples, see Section 12.20, “Aggregate (GROUP BY) Functions”.

- Added the JSON utility function `JSON_PRETTY()`, which outputs an existing `JSON` value in an easy-to-read format; each JSON object member or array value is printed on a separate line, and a child object or array is intended 2 spaces with respect to its parent.

  This function also works with a string that can be parsed as a JSON value.

  For more detailed information and examples, see Section 12.17.8, “JSON Utility Functions”.

- When sorting `JSON` values in a query using `ORDER BY`, each value is now represented by a variable-length part of the sort key, rather than a part of a fixed 1K in size. In many cases this can reduce excessive usage. For example, a scalar `INT` or even `BIGINT` value actually requires very few bytes, so that the remainder of this space (up to 90% or more) was taken up by padding. This change has the following benefits for performance:

  - Sort buffer space is now used more effectively, so that filesorts need not flush to disk as early or often as with fixed-length sort keys. This means that more data can be sorted in memory, avoiding unnecessary disk access.

  - Shorter keys can be compared more quickly than longer ones, providing a noticeable improvement in performance. This is true for sorts performed entirely in memory as well as for sorts that require writing to and reading from disk.

- Added support in MySQL 8.0.2 for partial, in-place updates of `JSON` column values, which is more efficient than completely removing an existing JSON value and writing a new one in its place, as was done previously when updating any `JSON` column. For this optimization to be applied, the update must be applied using `JSON_SET()`, `JSON_REPLACE()`, or `JSON_REMOVE()`. New elements cannot be added to the JSON document being updated; values within the document cannot take more space than they did before the update. See Partial Updates of JSON Values, for a detailed discussion of the requirements.

  Partial updates of JSON documents can be written to the binary log, taking up less space than logging complete JSON documents. Partial updates are always logged as such when statement-based replication is in use. For this to work with row-based replication, you must first set `binlog_row_value_options=PARTIAL_JSON`; see this variable's description for more information.

- Added the JSON utility functions `JSON_STORAGE_SIZE()` and `JSON_STORAGE_FREE()`. `JSON_STORAGE_SIZE()` returns the storage space in bytes used for the binary representation of a JSON document prior to any partial update (see previous item). `JSON_STORAGE_FREE()` shows the amount of space remaining in a table column of type `JSON` after it has been partially updated using `JSON_SET()` or `JSON_REPLACE()`; this is greater than zero if the binary representation of the new value is less than that of the previous value.

  Each of these functions also accepts a valid string representation of a JSON document. For such a value, `JSON_STORAGE_SIZE()` returns the space used by its binary representation following its conversion to a JSON document. For a variable containing the string representation of a JSON

document, `JSON_STORAGE_FREE()` returns zero. Either function produces an error if its (non-null) argument cannot be parsed as a valid JSON document, and `NULL` if the argument is `NULL`.

For more information and examples, see Section 12.17.8, “JSON Utility Functions”.

`JSON_STORAGE_SIZE()` and `JSON_STORAGE_FREE()` were implemented in MySQL 8.0.2.

- Added support in MySQL 8.0.2 for ranges such as `$[1 to 5]` in XPath expressions. Also added support in this version for the `last` keyword and relative addressing, such that `$[last]` always selects the last (highest-numbered) element in the array and `$[last-1]` the next to last element. `last` and expressions using it can also be included in range definitions. For example, `$[last-2 to last-1]` returns the last two elements but one from an array. See Searching and Modifying JSON Values, for additional information and examples.

- Added a JSON merge function intended to conform to RFC 7396. `JSON_MERGE_PATCH()`, when used on 2 JSON objects, merges them into a single JSON object that has as members a union of the following sets:

  - Each member of the first object for which there is no member with the same key in the second object.

  - Each member of the second object for which there is no member having the same key in the first object, and whose value is not the JSON `null` literal.

  - Each member having a key that exists in both objects, and whose value in the second object is not the JSON `null` literal.

  As part of this work, the `JSON_MERGE()` function has been renamed `JSON_MERGE_PRESERVE()`. `JSON_MERGE()` continues to be recognized as an alias for `JSON_MERGE_PRESERVE()` in MySQL 8.0, but is now deprecated and is subject to removal in a future version of MySQL.

  For more information and examples, see Section 12.17.4, “Functions That Modify JSON Values”.

- Implemented “last duplicate key wins” normalization of duplicate keys, consistent with RFC 7159 and most JavaScript parsers. An example of this behavior is shown here, where only the rightmost member having the key `x` is preserved:

```
mysql> SELECT JSON_OBJECT('x', '32', 'y', '[true, false]',
    >                     'x', '"abc"', 'x', '100') AS Result;
+-----------------------------------+
| Result                            |
+-----------------------------------+
| {"x": "100", "y": "[true, false]"} |
+-----------------------------------+
1 row in set (0.00 sec)
```

Values inserted into MySQL `JSON` columns are also normalized in this way, as shown in this example:

```
mysql> CREATE TABLE t1 (c1 JSON);

mysql> INSERT INTO t1 VALUES ('{"x": 17, "x": "red", "x": [3, 5, 7]}');

mysql> SELECT c1 FROM t1;
+------------------+
| c1               |
+------------------+
| {"x": [3, 5, 7]} |
+------------------+
```

This is an incompatible change from previous versions of MySQL, where a “first duplicate key wins” algorithm was used in such cases.

See Normalization, Merging, and Autowrapping of JSON Values, for more information and examples.

- Added the `JSON_TABLE()` function in MySQL 8.0.4. This function accepts JSON data and returns it as a relational table having the specified columns.

  This function has the syntax `JSON_TABLE(expr, path COLUMNS column_list) [AS] alias)`, where *`expr`* is an expression that returns JSON data, *`path`* is a JSON path applied to the source, and *`column_list`* is a list of column definitions. An example is shown here:

```
mysql> SELECT *
    -> FROM
    ->   JSON_TABLE(
    ->     '[{"a":3,"b":"0"},{"a":"3","b":"1"},{"a":2,"b":1},{"a":0},{"b":[1,2]}]',
    ->     "$[*]" COLUMNS(
    ->       rowid FOR ORDINALITY,
    ->
    ->       xa INT EXISTS PATH "$.a",
    ->       xb INT EXISTS PATH "$.b",
    ->
    ->       sa VARCHAR(100) PATH "$.a",
    ->       sb VARCHAR(100) PATH "$.b",
    ->
    ->       ja JSON PATH "$.a",
    ->       jb JSON PATH "$.b"
    ->     )
    ->   ) AS  jt1;
+-------+------+------+------+------+------+--------+
| rowid | xa   | xb   | sa   | sb   | ja   | jb     |
+-------+------+------+------+------+------+--------+
|     1 |    1 |    1 | 3    | 0    | 3    | "0"    |
|     2 |    1 |    1 | 3    | 1    | "3"  | "1"    |
|     3 |    1 |    1 | 2    | 1    | 2    | 1      |
|     4 |    1 |    0 | 0    | NULL | 0    | NULL   |
|     5 |    0 |    1 | NULL | NULL | NULL | [1, 2] |
+-------+------+------+------+------+------+--------+
```

  The JSON source expression can be any expression that yields a valid JSON document, including a JSON literal, a table column, or a function call that returns JSON such as `JSON_EXTRACT(t1, data, '$.post.comments')`. For more information, see Section 12.17.6, “JSON Table Functions”.

- **Data type support.** MySQL now supports use of expressions as default values in data type specifications. This includes the use of expressions as default values for the `BLOB`, `TEXT`, `GEOMETRY`, and `JSON` data types, which previously could not be assigned default values at all. For details, see Section 11.7, “Data Type Default Values”.

- **Optimizer.** These optimizer enhancements were added:

  - MySQL now supports invisible indexes. An invisible index is not used by the optimizer at all, but is otherwise maintained normally. Indexes are visible by default. Invisible indexes make it possible to test the effect of removing an index on query performance, without making a destructive change that must be undone should the index turn out to be required. See Section 8.3.12, “Invisible Indexes”.

  - MySQL now supports descending indexes: `DESC` in an index definition is no longer ignored but causes storage of key values in descending order. Previously, indexes could be scanned in reverse order but at a performance penalty. A descending index can be scanned in forward order, which is more efficient. Descending indexes also make it possible for the optimizer to use multiple-column indexes

when the most efficient scan order mixes ascending order for some columns and descending order for others. See Section 8.3.13, “Descending Indexes”.

- MySQL now supports creation of functional index key parts that index expression values rather than column values. Functional key parts enable indexing of values that cannot be indexed otherwise, such as `JSON` values. For details, see Section 13.1.15, “CREATE INDEX Syntax”.

- In MySQL 8.0.14 and later, trivial `WHERE` conditions arising from constant literal expressions are removed during preparation, rather than later on during optimization. Removal of the condition earlier in the process makes it possible to simplify joins for queries with outer joins having trivial conditions, such as this one:

  ```
  SELECT * FROM t1 LEFT JOIN t2 ON condition_1 WHERE condition_2 OR 0 = 1
  ```

  The optimizer now sees during preparation that 0 = 1 is always false, making `OR 0 = 1` redundant, and removes it, leaving this:

  ```
  SELECT * FROM t1 LEFT JOIN t2 ON condition_1 where condition_2
  ```

  Now the optimizer can rewrite the query as an inner join, like this:

  ```
  SELECT * FROM t1 LEFT JOIN t2 WHERE condition_1 AND condition_2
  ```

  For more information, see Section 8.2.1.9, “Outer Join Optimization”.

- In MySQL 8.0.16 and later, MySQL can use constant folding at optimization time to handle comparisons between a column and a constant value where the constant is out of range or on a range boundary with respect to the type of the column, rather than doing so for each row at execution time. For example, given a table `t` with a `TINYINT UNSIGNED` column `c`, the optimizer can rewrite a condition such as `WHERE c < 256` to `WHERE 1` (and optimize the condition away altogether), or `WHERE c >= 255` to `WHERE c = 255`.

  See Section 8.2.1.14, “Constant-Folding Optimization”, for more information.

- Beginning with MySQL 8.0.16, the semijoin optimizations used with `IN` subqueries can now be applied to `EXISTS` subqueries as well. In addition, the optimizer now decorrelates trivially-correlated equality predicates in the `WHERE` condition attached to the subquery, so that they can be treated similarly to expressions in `IN` subqueries; this applies to both `EXISTS` and `IN` subqueries.

  For more information, see Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”.

- In MySQL 8.0.17 and later a `WHERE` condition having `NOT IN (subquery)` or `NOT EXISTS (subquery)` is transformed internally into an antijoin. (An antijoin returns all rows from the table for which there is no row in the table to which it is joined matching the join condition.) This removes the subquery which can result in faster query execution since the subquery's tables are now handled on the top level.

  This is similar to, and reuses, the existing `IS NULL` (`Not exists`) optimization for outer joins; see EXPLAIN Extra Information.

- **Common table expressions.** MySQL now supports common table expressions, both nonrecursive and recursive. Common table expressions enable use of named temporary result sets, implemented

by permitting a `WITH` clause preceding `SELECT` statements and certain other statements. For more information, see Section 13.2.13, “WITH Syntax (Common Table Expressions)”.

- **Window functions.** MySQL now supports window functions that, for each row from a query, perform a calculation using rows related to that row. These include functions such as `RANK()`, `LAG()`, and `NTILE()`. In addition, several existing aggregate functions now can be used as window functions (for example, `SUM()` and `AVG()`). For more information, see Section 12.21, “Window Functions”.

- **Lateral derived tables.** A derived table now may be preceded by the `LATERAL` keyword to specify that it is permitted to refer to (depend on) columns of preceding tables in the same `FROM` clause. Lateral derived tables make possible certain SQL operations that cannot be done with nonlateral derived tables or that require less-efficient workarounds. See Section 13.2.11.9, “Lateral Derived Tables”.

- **Aliases in single-table DELETE statements.** In MySQL 8.0.16 and later, single-table `DELETE` statements support the use of table aliases.

- **Regular expression support.** Previously, MySQL used the Henry Spencer regular expression library to support regular expression operators (`REGEXP`, `RLIKE`). Regular expression support has been reimplemented using International Components for Unicode (ICU), which provides full Unicode support and is multibyte safe. The `REGEXP_LIKE()` function performs regular expression matching in the manner of the `REGEXP` and `RLIKE` operators, which now are synonyms for that function. In addition, the `REGEXP_INSTR()`, `REGEXP_REPLACE()`, and `REGEXP_SUBSTR()` functions are available to find match positions and perform substring substitution and extraction, respectively. The `regexp_stack_limit` and `regexp_time_limit` system variables provide control over resource consumption by the match engine. For more information, see Section 12.5.2, “Regular Expressions”. For information about ways in which applications that use regular expressions may be affected by the implementation change, see Regular Expression Compatibility Considerations.

- **Internal temporary tables.** The `TempTable` storage engine replaces the `MEMORY` storage engine as the default engine for in-memory internal temporary tables. The `TempTable` storage engine provides efficient storage for `VARCHAR` and `VARBINARY` columns. The `internal_tmp_mem_storage_engine` session variable defines the storage engine for in-memory internal temporary tables. Permitted values are `TempTable` (the default) and `MEMORY`. The `temptable_max_ram` variable defines the maximum amount of memory that the `TempTable` storage engine can use before data is stored to disk.

- **Logging.** Error logging was rewritten to use the MySQL component architecture. Traditional error logging is implemented using built-in components, and logging using the system log is implemented as a loadable component. In addition, a loadable JSON log writer is available. To control which log components to enable, use the `log_error_services` system variable. For more information, see Section 5.4.2, “The Error Log”.

- **Backup lock.** A new type of backup lock permits DML during an online backup while preventing operations that could result in an inconsistent snapshot. The new backup lock is supported by `LOCK INSTANCE FOR BACKUP` and `UNLOCK INSTANCE` syntax. The `BACKUP_ADMIN` privilege is required to use these statements.

- **Replication.** The following enhancements have been made to MySQL Replication:

  - MySQL Replication now supports binary logging of partial updates to JSON documents using a compact binary format, saving space in the log over logging complete JSON documents. Such compact logging is done automatically when statement-based logging is in use, and can be enabled by setting the new `binlog_row_value_options` system variable to `PARTIAL_JSON`. For more information, see Partial Updates of JSON Values, as well as the description of `binlog_row_value_options`.

- **Connection management.** MySQL Server now permits a TCP/IP port to be configured specifically for administrative connections. This provides an alternative to the single administrative connection that

is permitted on the network interfaces used for ordinary connections even when `max_connections` connections are already established. See Section 8.12.4.1, “How MySQL Handles Client Connections”.

MySQL now provides more control over the use of compression to minimize the number of bytes sent over connections to the server. Previously, a given connection was either uncompressed or used the `zlib` compression algorithm. Now, it is also possible to use the `zstd` algorithm, and to select a compression level for `zstd` connections. The permitted compression algorithms can be configured on the server side, as well as on the connection-origination side for connections by client programs and by servers participating in master/slave replication or Group Replication. For more information, see Section 4.2.6, “Connection Compression Control”.

- **Configuration.** The maximum permitted length of host names throughout MySQL has been raised to 255 ASCII characters, up from the previous limit of 60 characters. This applies to, for example, host name-related columns in the data dictionary, `mysql` system schema, Performance Schema, `INFORMATION_SCHEMA`, and `sys` schema; the `MASTER_HOST` value for the `CHANGE MASTER TO` statement; the `Host` column in `SHOW PROCESSLIST` statement output; host names in account names (such as used in account-management statements and in `DEFINER` attributes); and host name-related command options and system variables.

  Caveats:

  - The increase in permitted host name length can affect tables with indexes on host name columns. For example, tables in the `mysql` system schema that index host names now have an explicit `ROW_FORMAT` attribute of `DYNAMIC` to accommodate longer index values.
  - Some file name-valued configuration settings might be constructed based on the server host name. The permitted values are constrained by the underlying operating system, which may not permit file names long enough to include 255-character host names. This affects the `general_log_file`, `log_error`, `pid_file`, `relay_log`, and `slow_query_log_file` system variables and corresponding options. If host name-based values are too long for the OS, explicit shorter values must be provided.
  - Although the server now supports 255-character host names, connections to the server established using the `--ssl-mode=VERIFY_IDENTITY` option are constrained by maximum host name length supported by OpenSSL. Host name matches pertain to two fields of SSL certificates, which have maximum lengths as follows: Common Name: maximum length 64; Subject Alternative Name: maximum length as per RFC#1034.

- **Plugins.** Previously, MySQL plugins could be written in C or C++. MySQL header files used by plugins now contain C++ code, which means that plugins must be written in C++, not C.

- **C API.** The MySQL C API now supports asynchronous functions for nonblocking communication with the MySQL server. Each function is the asynchronous counterpart to an existing synchronous function. The synchronous functions block if reads from or writes to the server connection must wait. The asynchronous functions enable an application to check whether work on the server connection is ready to proceed. If not, the application can perform other work before checking again later. See Section 28.7.12, “C API Asynchronous Interface”.

- **Additional target types for casts.** The functions `CAST()` and `CONVERT()` now support conversions to types `DOUBLE`, `FLOAT`, and `REAL`. Added in MySQL 8.0.17. See Section 12.10, “Cast Functions and Operators”.

- **JSON Schema validation.** MySQL 8.0.17 adds two functions `JSON_SCHEMA_VALID()` and `JSON_SCHEMA_VALIDATION_REPORT()` for validating JSON documents again JSON schemas. `JSON_SCHEMA_VALID()` returns TRUE (1) if the document validates against the schema and FALSE (0) if it does not. `JSON_SCHEMA_VALIDATION_REPORT()` returns a JSON document containing

detailed information about the results of the validation. The following statements apply to both of these functions:

- The schema must conform to Draft 4 of the JSON Schema specification.
- `required` attributes are supported.
- External resources and the `$ref` keyword are not supported.
- Regular expression patterns are supported; invalid patterns are silently ignored.

See Section 12.17.7, "JSON Schema Validation Functions", for more information and examples.

- **Multi-valued indexes.** Beginning with MySQL 8.0.17, `InnoDB` supports the creation of a multi-valued index, which is a secondary index defined on a `JSON` column that stores an array of values and which can have multiple index records for a single data record. Such an index uses a key part definition such as `CAST(data->'$.zipcode' AS UNSIGNED ARRAY)`. A multi-valued index is used automatically by the MySQL optimizer for suitable queries, as can be viewed in the output of `EXPLAIN`.

  As part of this work, MySQL adds a new function `JSON_OVERLAPS()` and a new `MEMBER OF()` operator for working with `JSON` documents, additionally extending the `CAST()` function with a new `ARRAY` keyword, as described in the following list:

  - `JSON_OVERLAPS()` compares two `JSON` documents. If they contain any key-value pairs or array elements in common, the function returns TRUE (1); otherwise it returns FALSE (0). If both values are scalars, the function performs a simple test for equality. If one argument is a JSON array and the other is a scalar, the scalar is treated as an array element. Thus, `JSON_OVERLAPS()` acts as a complement to `JSON_CONTAINS()`.
  - `MEMBER OF()` tests whether the first operand (a scalar or JSON document) is a member of the JSON array passed as the second operand, returning TRUE (1) if it is, and FALSE (0) if it is not. No type conversion of the operand is performed.
  - `CAST(path AS type ARRAY)` permits creation of a functional index by casting the JSON array found in a JSON document at *`json_path`* to an SQL array. Type specifiers are limited to those already supported by `CAST()`, with the exception of `BINARY` (not supported). This usage of `CAST()` (and the `ARRAY` keyword) is supported only by `InnoDB`, and only for the creation of a multi-valued index.

  For detailed information about multi-valued indexes, including examples, see Multi-Valued Indexes. Section 12.17.3, "Functions That Search JSON Values", provides information about `JSON_OVERLAPS()` and `MEMBER OF()`, along with examples of use.

- **Hintable time_zone.** As of MySQL 8.0.17, the `time_zone` session variable is hintable using `SET_VAR`.

- **Red Log Archiving.** As of MySQL 8.0.17, `InnoDB` supports redo log archiving. Backup utilities that copy redo log records may sometimes fail to keep pace with redo log generation while a backup operation is in progress, resulting in lost redo log records due to those records being overwritten. The redo log archiving feature addresses this issue by sequentially writing redo log records to an archive file. Backup utilities can copy redo log records from the archive file as necessary, thereby avoiding the potential loss of data. For more information, see Redo Log Archiving.

- **The Clone Plugin.** As of MySQL 8.0.17, MySQL provides a clone plugin that permits cloning `InnoDB` data locally or from a remote MySQL server instance. A local cloning operation stores cloned data on the same server or node where the MySQL instance runs. A remote cloning operation transfers cloned data

over the network from a donor MySQL server instance to the recipient server or node where the cloning operation was initiated.

The clone plugin supports replication. In addition to cloning data, a cloning operation extracts and transfers replication coordinates from the donor and applies them on the recipient, which enables using the clone plugin for provisioning Group Replication members and replication slaves. Using the clone plugin for provisioning is considerably faster and more efficient than replicating a large number of transactions. Group Replication members can also be configured to use the clone plugin as an alternative method of recovery, so that members automatically choose the most efficient way to retrieve group data from seed members.

For more information, see Section 5.6.7, “The Clone Plugin”, and Section 18.4.3.1, “Cloning for Distributed Recovery”.

- **Hash Join Optimization.** Beginning with MySQL 8.0.18, a hash join is used whenever each pair of tables in a join includes at least one equi-join condition. A hash join does not require indexes, and is more efficient in most cases than the block-nested loop algorithm. Joins such as those shown here can be optimized in this manner:

```
SELECT *
    FROM t1
    JOIN t2
        ON t1.c1=t2.c1;

SELECT *
    FROM t1
    JOIN t2
        ON (t1.c1 = t2.c1 AND t1.c2 < t2.c2)
    JOIN t3
        ON (t2.c1 = t3.c1)
```

  Hash joins can also be used for Cartesian products—that is, when no join condition is specified.

  You can see when the hash join optimization is being used for a particular query using `EXPLAIN FORMAT=TREE` or `EXPLAIN ANALYZE`.

  Use of hash joins can be controlled using the `optimizer_switch` system variable' `hash_join` flag (`on` by default) as well as the `HASH_JOIN` and `NO_HASH_JOIN` optimizer hints.

  The amount of memory available to a hash join is limited by the value of `join_buffer_size`. A hash join that requires more than this much memory is executed on disk; the number of disk files that can be used by an on-disk hash join is limited by `open_files_limit`.

  For more information and examples, see Section 8.2.1.4, “Hash Join Optimization”.

- **EXPLAIN ANALYZE Statement.** A new form of the `EXPLAIN` statement, `EXPLAIN ANALYZE`, is implemented in MySQL 8.0.18, providing expanded information about the execution of `SELECT` statements in `TREE` format for each iterator used in processing the query, and making it possible to compare estimated cost with the actual cost of the query. This information includes startup cost, total cost, number of rows returned by this iterator, and the number of loops executed.

  See Obtaining Information with EXPLAIN ANALYZE, for more information.

- **Query cast injection.** In version 8.0.18 and later, MySQL now injects cast operations into the query item tree inside expressions and conditions in which the data type of the argument and the expected data type do not match. This has no effect on query results or speed of execution, but makes the query

as executed equivalent to one which is compliant with the SQL standard while maintaining backwards compatibility with previous releases of MySQL.

Such implicit casts are now performed between temporal types (`DATE`, `DATETIME`, `TIMESTAMP`, `TIME`) and numeric types (`SMALLINT`, `TINYINT`, `MEDIUMINT`, `INT`/`INTEGER`, `BIGINT`; `DECIMAL`/`NUMERIC`; `FLOAT`, `DOUBLE`, `REAL`; `BIT`) whenever they are compared using any of the standard numeric comparison operators (`=`, `>=`, `>`, `<`, `<=`, `<>`/`!=`, or `<=>`). In this case, any value that is not already a `DOUBLE` is cast as one. Cast injection is also now performed for comparisons between `DATE` or `TIME` values and `DATETIME` values, where the arguments are cast whenever necessary as `DATETIME`.

It is possible to see when casts are injected into a given query by viewing the output of `EXPLAIN ANALYZE`, `EXPLAIN FORMAT=JSON`, or, as shown here, `EXPLAIN FORMAT=TREE`:

```
mysql> CREATE TABLE d (dt DATETIME, d DATE, t TIME);
Query OK, 0 rows affected (0.62 sec)

mysql> CREATE TABLE n (i INT, d DECIMAL, f FLOAT, dc DECIMAL);
Query OK, 0 rows affected (0.51 sec)

mysql> EXPLAIN FORMAT=TREE SELECT * from d JOIN n ON d.dt = n.i\G
*************************** 1. row ***************************
EXPLAIN: -> Inner hash join (cast(d.dt as double) = cast(n.i as double))
(cost=0.70 rows=1)
    -> Table scan on n  (cost=0.35 rows=1)
    -> Hash
        -> Table scan on d  (cost=0.35 rows=1)
```

Such casts can also be seen by executing `EXPLAIN [FORMAT=TRADITIONAL]`, in which case it is also necessary to issue `SHOW WARNINGS` afterwards.

Features Deprecated in MySQL 8.0

The following features are deprecated in MySQL 8.0 and may be or will be removed in a future series. Where alternatives are shown, applications should be updated to use them.

For applications that use features deprecated in MySQL 8.0 that have been removed in a higher MySQL series, statements may fail when replicated from a MySQL 8.0 master to a higher-series slave, or may have different effects on master and slave. To avoid such problems, applications that use features deprecated in 8.0 should be revised to avoid them and use alternatives when possible.

- The `utf8mb3` character set is deprecated. Please use `utf8mb4` instead.
- Because `caching_sha2_password` is the default authentication plugin in MySQL 8.0 and provides a superset of the capabilities of the `sha256_password` authentication plugin, `sha256_password` is deprecated and will be removed in a future MySQL version. MySQL accounts that authenticate using `sha256_password` should be migrated to use `caching_sha2_password` instead.
- The `validate_password` plugin has been reimplemented to use the server component infrastructure. The plugin form of `validate_password` is still available but is deprecated and will be removed in a future version of MySQL. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.4.3.3, “Transitioning to the Password Validation Component”.
- The `ALTER TABLESPACE` and `DROP TABLESPACE ENGINE` clause is deprecated.
- The `PAD_CHAR_TO_FULL_LENGTH` SQL mode is deprecated.
- `AUTO_INCREMENT` support is deprecated for columns of type `FLOAT` and `DOUBLE` (and any synonyms). Consider removing the `AUTO_INCREMENT` attribute from such columns, or convert them to an integer type.

- The `UNSIGNED` attribute is deprecated for columns of type `FLOAT`, `DOUBLE`, and `DECIMAL` (and any synonyms). Consider using a simple `CHECK` constraint instead for such columns.
- `FLOAT(M,D)` and `DOUBLE(M,D)` syntax to specify the number of digits for columns of type `FLOAT` and `DOUBLE` (and any synonyms) is a nonstandard MySQL extension. This syntax is deprecated.
- The `ZEROFILL` attribute is deprecated for numeric data types, as is the display width attribute for integer data types. Consider using an alternative means of producing the effect of these attributes. For example, applications could use the `LPAD()` function to zero-pad numbers up to the desired width, or they could store the formatted numbers in `CHAR` columns.
- For string data types, the `BINARY` attribute is a nonstandard MySQL extension that is shorthand for specifying the binary (`_bin`) collation of the column character set (or of the table default character set if no column character set is specified). In MySQL 8.0, this nonstandard use of `BINARY` is ambiguous because the `utf8mb4` character set has multiple `_bin` collations, so the `BINARY` attribute is deprecated and support for it will be removed in a future MySQL version. Applications should be adjusted to use an explicit `_bin` collation instead.

  The use of `BINARY` to specify a data type or character set remains unchanged.
- The nonstandard C-style `&&`, `||`, and `!` operators that are synonyms for the standard SQL `AND`, `OR`, and `NOT` operators, respectively, are deprecated. Applications that use the nonstandard operators should be adjusted to use the standard operators.

  Note

  Use of `||` is deprecated unless the `PIPES_AS_CONCAT` SQL mode is enabled. In that case, `||` signifies the SQL-standard string concatenation operator).
- The `JSON_MERGE()` function is deprecated. Use `JSON_MERGE_PRESERVE()` instead.
- The `SQL_CALC_FOUND_ROWS` query modifier and accompanying `FOUND_ROWS()` function are deprecated. See the `FOUND_ROWS()` description for information about an alternative strategy.
- Support for `TABLESPACE = innodb_file_per_table` and `TABLESPACE = innodb_temporary` clauses with `CREATE TEMPORARY TABLE` is deprecated as of MySQL 8.0.13.
- The `mysql_upgrade` client is deprecated because its capabilities for upgrading the system tables in the `mysql` system schema and objects in other schemas have been moved into the MySQL server. See Section 2.11.3, “What the MySQL Upgrade Process Upgrades”.
- The `--no-dd-upgrade` server option is deprecated. It is superseded by the `--upgrade` option, which provides finer control over data dictionary and server upgrade behavior.
- The `mysql_upgrade_info` file, which is created data directory and used to store the MySQL version number, is deprecated and will be removed in a future MySQL version.
- The `relay_log_info_file` system variable and `--master-info-file` option are deprecated. Previously, these were used to specify the name of the relay log info log and master info log when `--relay-log-info-repository=FILE` and `--master-info-repository=FILE` were set, but those settings have been deprecated. The use of files for the relay log info log and master info log has been superseded by crash-safe slave tables, which are the default in MySQL 8.0.
- These legacy parameters for compression of connections to the server are deprecated: The `--compress` client command-line option; the `MYSQL_OPT_COMPRESS` option for the `mysql_options()` C API function; the `slave_compressed_protocol` system variable. For information about parameters to use instead, see Section 4.2.6, “Connection Compression Control”.

- Use of the MYSQL_PWD environment variable to specify a MySQL password is deprecated.

Features Removed in MySQL 8.0

The following items are obsolete and have been removed in MySQL 8.0. Where alternatives are shown, applications should be updated to use them.

For MySQL 5.7 applications that use features removed in MySQL 8.0, statements may fail when replicated from a MySQL 5.7 master to a MySQL 8.0 slave, or may have different effects on master and slave. To avoid such problems, applications that use features removed in MySQL 8.0 should be revised to avoid them and use alternatives when possible.

- The innodb_locks_unsafe_for_binlog system variable was removed. The READ COMMITTED isolation level provides similar functionality.
- The information_schema_stats variable, introduced in MySQL 8.0.0, was removed and replaced by information_schema_stats_expiry in MySQL 8.0.3.

  information_schema_stats_expiry defines an expiration setting for cached INFORMATION_SCHEMA table statistics. For more information, see Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”.
- Code related to obsoleted InnoDB system tables was removed in MySQL 8.0.3. INFORMATION_SCHEMA views based on InnoDB system tables were replaced by internal system views on data dictionary tables. Affected InnoDB INFORMATION_SCHEMA views were renamed:

  Table 1.1 Renamed InnoDB Information Schema Views

Old Name	New Name
INNODB_SYS_COLUMNS	INNODB_COLUMNS
INNODB_SYS_DATAFILES	INNODB_DATAFILES
INNODB_SYS_FIELDS	INNODB_FIELDS
INNODB_SYS_FOREIGN	INNODB_FOREIGN
INNODB_SYS_FOREIGN_COLS	INNODB_FOREIGN_COLS
INNODB_SYS_INDEXES	INNODB_INDEXES
INNODB_SYS_TABLES	INNODB_TABLES
INNODB_SYS_TABLESPACES	INNODB_TABLESPACES
INNODB_SYS_TABLESTATS	INNODB_TABLESTATS
INNODB_SYS_VIRTUAL	INNODB_VIRTUAL

  After upgrading to MySQL 8.0.3 or later, update any scripts that reference previous InnoDB INFORMATION_SCHEMA view names.
- The following features related to account management are removed:
  - Using GRANT to create users. Instead, use CREATE USER. Following this practice makes the NO_AUTO_CREATE_USER SQL mode immaterial for GRANT statements, so it too is removed, and an error now is written to the server log when the presence of this value for the sql_mode option in the options file prevents mysqld from starting.
  - Using GRANT to modify account properties other than privilege assignments. This includes authentication, SSL, and resource-limit properties. Instead, establish such properties at account-creation time with CREATE USER or modify them afterward with ALTER USER.

  - `IDENTIFIED BY PASSWORD 'auth_string'` syntax for `CREATE USER` and `GRANT`. Instead, use `IDENTIFIED WITH auth_plugin AS 'auth_string'` for `CREATE USER` and `ALTER USER`, where the `'auth_string'` value is in a format compatible with the named plugin.

    Additionally, because `IDENTIFIED BY PASSWORD` syntax was removed, the `log_builtin_as_identified_by_password` system variable is superfluous and was removed.

  - The `PASSWORD()` function. Additionally, `PASSWORD()` removal means that `SET PASSWORD ... = PASSWORD('auth_string')` syntax is no longer available.

  - The `old_passwords` system variable.

- The query cache was removed. Removal includes these items:

  - The `FLUSH QUERY CACHE` and `RESET QUERY CACHE` statements.

  - These system variables: `query_cache_limit`, `query_cache_min_res_unit`, `query_cache_size`, `query_cache_type`, `query_cache_wlock_invalidate`.

  - These status variables: `Qcache_free_blocks`, `Qcache_free_memory`, `Qcache_hits`, `Qcache_inserts`, `Qcache_lowmem_prunes`, `Qcache_not_cached`, `Qcache_queries_in_cache`, `Qcache_total_blocks`.

  - These thread states: `checking privileges on cached query`, `checking query cache for query`, `invalidating query cache entries`, `sending cached result to client`, `storing result in query cache`, `Waiting for query cache lock`.

  - The `SQL_CACHE SELECT` modifier.

  These deprecated query cache items remain deprecated, but have no effect, and will be removed in a future MySQL release:

  - `SQL_NO_CACHE SELECT` modifier.

  - The `ndb_cache_check_time` system variable.

  The `have_query_cache` system variable remains deprecated, always has a value of `NO`, and will be removed in a future MySQL release.

- The data dictionary provides information about database objects, so the server no longer checks directory names in the data directory to find databases. Consequently, the `--ignore-db-dir` option and `ignore_db_dirs` system variables are extraneous and are removed.

- The `tx_isolation` and `tx_read_only` system variables have been removed. Use `transaction_isolation` and `transaction_read_only` instead.

- The `sync_frm` system variable has been removed because `.frm` files have become obsolete.

- The `secure_auth` system variable and `--secure-auth` client option have been removed. The `MYSQL_SECURE_AUTH` option for the `mysql_options()` C API function was removed.

- The `multi_range_count` system variable is removed.

- The `log_warnings` system variable and `--log-warnings` server option have been removed. Use the `log_error_verbosity` system variable instead.

- The global scope for the `sql_log_bin` system variable was removed. `sql_log_bin` has session scope only, and applications that rely on accessing `@@GLOBAL.sql_log_bin` should be adjusted.
- The `metadata_locks_cache_size` and `metadata_locks_hash_instances` system variables are removed.
- The unused `date_format`, `datetime_format`, `time_format`, and `max_tmp_tables` system variables are removed.
- These deprecated compatibility SQL modes are removed: `DB2`, `MAXDB`, `MSSQL`, `MYSQL323`, `MYSQL40`, `ORACLE`, `POSTGRESQL`, `NO_FIELD_OPTIONS`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`. They can no longer be assigned to the `sql_mode` system variable or used as permitted values for the `mysqldump --compatible` option.

  Removal of `MAXDB` means that the `TIMESTAMP` data type for `CREATE TABLE` or `ALTER TABLE` is treated as `TIMESTAMP`, and is no longer treated as `DATETIME`.
- The deprecated `ASC` or `DESC` qualifiers for `GROUP BY` clauses are removed. Queries that previously relied on `GROUP BY` sorting may produce results that differ from previous MySQL versions. To produce a given sort order, provide an `ORDER BY` clause.
- The `EXTENDED` and `PARTITIONS` keywords for the `EXPLAIN` statement have been removed. These keywords are unnecessary because their effect is always enabled.
- These encryption-related items are removed:
  - The `ENCODE()` and `DECODE()` functions.
  - The `ENCRYPT()` function.
  - The `DES_ENCRYPT()`, and `DES_DECRYPT()` functions, the `--des-key-file` option, the `have_crypt` system variable, the `DES_KEY_FILE` option for the `FLUSH` statement, and the `HAVE_CRYPT` `CMake` option.

  In place of the removed encryption functions: For `ENCRYPT()`, consider using `SHA2()` instead for one-way hashing. For the others, consider using `AES_ENCRYPT()` and `AES_DECRYPT()` instead.
- In MySQL 5.7, several spatial functions available under multiple names were deprecated to move in the direction of making the spatial function namespace more consistent, the goal being that each spatial function name begin with `ST_` if it performs an exact operation, or with `MBR` if it performs an operation based on minimum bounding rectangles. In MySQL 8.0, the deprecated functions are removed to leave only the corresponding `ST_` and `MBR` functions:
  - These functions are removed in favor of the `MBR` names: `Contains()`, `Disjoint()`, `Equals()`, `Intersects()`, `Overlaps()`, `Within()`.
  - These functions are removed in favor of the `ST_` names: `Area()`, `AsBinary()`, `AsText()`, `AsWKB()`, `AsWKT()`, `Buffer()`, `Centroid()`, `ConvexHull()`, `Crosses()`, `Dimension()`, `Distance()`, `EndPoint()`, `Envelope()`, `ExteriorRing()`, `GeomCollFromText()`, `GeomCollFromWKB()`, `GeomFromText()`, `GeomFromWKB()`, `GeometryCollectionFromText()`, `GeometryCollectionFromWKB()`, `GeometryFromText()`, `GeometryFromWKB()`, `GeometryN()`, `GeometryType()`, `InteriorRingN()`, `IsClosed()`, `IsEmpty()`, `IsSimple()`, `LineFromText()`, `LineFromWKB()`, `LineStringFromText()`, `LineStringFromWKB()`, `MLineFromText()`, `MLineFromWKB()`, `MPointFromText()`, `MPointFromWKB()`, `MPolyFromText()`, `MPolyFromWKB()`, `MultiLineStringFromText()`, `MultiLineStringFromWKB()`, `MultiPointFromText()`, `MultiPointFromWKB()`, `MultiPolygonFromText()`, `MultiPolygonFromWKB()`, `NumGeometries()`,

`NumInteriorRings()`, `NumPoints()`, `PointFromText()`, `PointFromWKB()`, `PointN()`, `PolyFromText()`, `PolyFromWKB()`, `PolygonFromText()`, `PolygonFromWKB()`, `SRID()`, `StartPoint()`, `Touches()`, `X()`, `Y()`.

- `GLength()` is removed in favor of `ST_Length()`.

- The functions described in Section 12.16.4, “Functions That Create Geometry Values from WKB Values” previously accepted either WKB strings or geometry arguments. Geometry arguments are no longer permitted and produce an error. See that section for guidelines for migrating queries away from using geometry arguments.

- The parser no longer treats `\N` as a synonym for `NULL` in SQL statements. Use `NULL` instead.

  This change does not affect text file import or export operations performed with `LOAD DATA` or `SELECT ... INTO OUTFILE`, for which `NULL` continues to be represented by `\N`. See Section 13.2.7, “LOAD DATA Syntax”.

- `PROCEDURE ANALYSE()` syntax is removed.

- The client-side `--ssl` and `--ssl-verify-server-cert` options have been removed. Use `--ssl-mode=REQUIRED` instead of `--ssl=1` or `--enable-ssl`. Use `--ssl-mode=DISABLED` instead of `--ssl=0`, `--skip-ssl`, or `--disable-ssl`. Use `--ssl-mode=VERIFY_IDENTITY` instead of `--ssl-verify-server-cert` options. (The server-side `--ssl` option remains unchanged.)

  For the C API, `MYSQL_OPT_SSL_ENFORCE` and `MYSQL_OPT_SSL_VERIFY_SERVER_CERT` options for `mysql_options()` correspond to the client-side `--ssl` and `--ssl-verify-server-cert` options and are removed. Use `MYSQL_OPT_SSL_MODE` with an option value of `SSL_MODE_REQUIRED` or `SSL_MODE_VERIFY_IDENTITY` instead.

- The `--temp-pool` server option was removed.

- The `ignore_builtin_innodb` system variable is removed.

- The server no longer performs conversion of pre-MySQL 5.1 database names containing special characters to 5.1 format with the addition of a `#mysql50#` prefix. Because these conversions are no longer performed, the `--fix-db-names` and `--fix-table-names` options for `mysqlcheck`, the `UPGRADE DATA DIRECTORY NAME` clause for the `ALTER DATABASE` statement, and the `Com_alter_db_upgrade` status variable are removed.

  Upgrades are supported only from one major version to another (for example, 5.0 to 5.1, or 5.1 to 5.5), so there should be little remaining need for conversion of older 5.0 database names to current versions of MySQL. As a workaround, upgrade a MySQL 5.0 installation to MySQL 5.1 before upgrading to a more recent release.

- The `mysql_install_db` program has been removed from MySQL distributions. Data directory initialization should be performed by invoking `mysqld` with the `--initialize` or `--initialize-insecure` option instead. In addition, the `--bootstrap` option for `mysqld` that was used by `mysql_install_db` was removed, and the `INSTALL_SCRIPTDIR` `CMake` option that controlled the installation location for `mysql_install_db` was removed.

- The generic partitioning handler was removed from the MySQL server. In order to support partitioning of a given table, the storage engine used for the table must now provide its own (“native”) partitioning handler. The `--partition` and `--skip-partition` options are removed from the MySQL Server, and partitioning-related entries are no longer shown in the output of `SHOW PLUGINS` or in the `INFORMATION_SCHEMA.PLUGINS` table.

Two MySQL storage engines currently provide native partitioning support—`InnoDB` and `NDB`; of these, only `InnoDB` is supported in MySQL 8.0. Any attempt to create partitioned tables in MySQL 8.0 using any other storage engine fails.

Ramifications for upgrades. The direct upgrade of a partitioned table using a storage engine other than `InnoDB` (such as `MyISAM`) from MySQL 5.7 (or earlier) to MySQL 8.0 is not supported. There are two options for handling such a table:

- Remove the table's partitioning, using `ALTER TABLE ... REMOVE PARTITIONING`.
- Change the storage engine used for the table to `InnoDB`, with `ALTER TABLE ... ENGINE=INNODB`.

At least one of the two operations just listed must be performed for each partitioned non-`InnoDB` table prior to upgrading the server to MySQL 8.0. Otherwise, such a table cannot be used following the upgrade.

Due to the fact that table creation statements that would result in a partitioned table using a storage engine without partitioning support now fail with an error (`ER_CHECK_NOT_IMPLEMENTED`), you must make sure that any statements in a dump file (such as that written by `mysqldump`) from an older version of MySQL that you wish to import into a MySQL 8.0 server that create partitioned tables do not also specify a storage engine such as `MyISAM` that has no native partitioning handler. You can do this by performing either of the following:

- Remove any references to partitioning from `CREATE TABLE` statements that use a value for the `STORAGE ENGINE` option other than `InnoDB`.
- Specifying the storage engine as `InnoDB`, or allow `InnoDB` to be used as the table's storage engine by default.

For more information, see Section 23.6.2, “Partitioning Limitations Relating to Storage Engines”.

- System and status variable information is no longer maintained in the `INFORMATION_SCHEMA`. These tables are removed: `GLOBAL_VARIABLES`, `SESSION_VARIABLES`, `GLOBAL_STATUS`, `SESSION_STATUS`. Use the corresponding Performance Schema tables instead. See Section 26.12.13, “Performance Schema System Variable Tables”, and Section 26.12.14, “Performance Schema Status Variable Tables”. In addition, the `show_compatibility_56` system variable was removed. It was used in the transition period during which system and status variable information in `INFORMATION_SCHEMA` tables was moved to Performance Schema tables, and is no longer needed. These status variables are removed: `Slave_heartbeat_period`, `Slave_last_heartbeat`, `Slave_received_heartbeats`, `Slave_retried_transactions`, `Slave_running`. The information they provided is available in Performance Schema tables; see Migrating to Performance Schema System and Status Variable Tables.
- The Performance Schema `setup_timers` table was removed, as was the `TICK` row in the `performance_timers` table.
- The `libmysqld` embedded server library is removed, along with:
  - The `mysql_options()` `MYSQL_OPT_GUESS_CONNECTION`, `MYSQL_OPT_USE_EMBEDDED_CONNECTION`, `MYSQL_OPT_USE_REMOTE_CONNECTION`, and `MYSQL_SET_CLIENT_IP` options
  - The `mysql_config` `--libmysqld-libs`, `--embedded-libs`, and `--embedded` options
  - The `CMake` `WITH_EMBEDDED_SERVER`, `WITH_EMBEDDED_SHARED_LIBRARY`, and `INSTALL_SECURE_FILE_PRIV_EMBEDDEDDIR` options

  - The (undocumented) `mysql --server-arg` option
  - The `mysqltest --embedded-server`, `--server-arg`, and `--server-file` options
  - The `mysqltest_embedded` and `mysql_client_test_embedded` test programs
- The `mysql_plugin` utility was removed. Alternatives include loading plugins at server startup using the `--plugin-load` or `--plugin-load-add` option, or at runtime using the `INSTALL PLUGIN` statement.
- The `resolveip` utility is removed. `nslookup`, `host`, or `dig` can be used instead.
- The `resolve_stack_dump` utility is removed. Stack traces from official MySQL builds are always symbolized, so there is no need to use `resolve_stack_dump`.
- The following server error codes are not used and have been removed. Applications that test specifically for any of these errors should be updated.

```
ER_BINLOG_READ_EVENT_CHECKSUM_FAILURE
ER_BINLOG_ROW_RBR_TO_SBR
ER_BINLOG_ROW_WRONG_TABLE_DEF
ER_CANT_ACTIVATE_LOG
ER_CANT_CHANGE_GTID_NEXT_IN_TRANSACTION
ER_CANT_CREATE_FEDERATED_TABLE
ER_CANT_CREATE_SROUTINE
ER_CANT_DELETE_FILE
ER_CANT_GET_WD
ER_CANT_SET_GTID_PURGED_WHEN_GTID_MODE_IS_OFF
ER_CANT_SET_WD
ER_CANT_WRITE_LOCK_LOG_TABLE
ER_CREATE_DB_WITH_READ_LOCK
ER_CYCLIC_REFERENCE
ER_DB_DROP_DELETE
ER_DELAYED_NOT_SUPPORTED
ER_DIFF_GROUPS_PROC
ER_DISK_FULL
ER_DROP_DB_WITH_READ_LOCK
ER_DROP_USER
ER_DUMP_NOT_IMPLEMENTED
ER_ERROR_DURING_CHECKPOINT
ER_ERROR_ON_CLOSE
ER_EVENTS_DB_ERROR
ER_EVENT_CANNOT_DELETE
ER_EVENT_CANT_ALTER
ER_EVENT_COMPILE_ERROR
ER_EVENT_DATA_TOO_LONG
ER_EVENT_DROP_FAILED
ER_EVENT_MODIFY_QUEUE_ERROR
ER_EVENT_NEITHER_M_EXPR_NOR_M_AT
ER_EVENT_OPEN_TABLE_FAILED
ER_EVENT_STORE_FAILED
ER_EXEC_STMT_WITH_OPEN_CURSOR
ER_FAILED_ROUTINE_BREAK_BINLOG
ER_FLUSH_MASTER_BINLOG_CLOSED
ER_FORM_NOT_FOUND
ER_FOUND_GTID_EVENT_WHEN_GTID_MODE_IS_OFF__UNUSED
ER_FRM_UNKNOWN_TYPE
ER_GOT_SIGNAL
ER_GRANT_PLUGIN_USER_EXISTS
ER_GTID_MODE_REQUIRES_BINLOG
ER_GTID_NEXT_IS_NOT_IN_GTID_NEXT_LIST
ER_HASHCHK
ER_INDEX_REBUILD
```

```
ER_INNODB_NO_FT_USES_PARSER
ER_LIST_OF_FIELDS_ONLY_IN_HASH_ERROR
ER_LOAD_DATA_INVALID_COLUMN_UNUSED
ER_LOGGING_PROHIBIT_CHANGING_OF
ER_MALFORMED_DEFINER
ER_MASTER_KEY_ROTATION_ERROR_BY_SE
ER_NDB_CANT_SWITCH_BINLOG_FORMAT
ER_NEVER_USED
ER_NISAMCHK
ER_NO_CONST_EXPR_IN_RANGE_OR_LIST_ERROR
ER_NO_FILE_MAPPING
ER_NO_GROUP_FOR_PROC
ER_NO_RAID_COMPILED
ER_NO_SUCH_KEY_VALUE
ER_NO_SUCH_PARTITION__UNUSED
ER_OBSOLETE_CANNOT_LOAD_FROM_TABLE
ER_OBSOLETE_COL_COUNT_DOESNT_MATCH_CORRUPTED
ER_ORDER_WITH_PROC
ER_PARTITION_SUBPARTITION_ERROR
ER_PARTITION_SUBPART_MIX_ERROR
ER_PART_STATE_ERROR
ER_PASSWD_LENGTH
ER_QUERY_ON_MASTER
ER_RBR_NOT_AVAILABLE
ER_SKIPPING_LOGGED_TRANSACTION
ER_SLAVE_CHANNEL_DELETE
ER_SLAVE_MULTIPLE_CHANNELS_HOST_PORT
ER_SLAVE_MUST_STOP
ER_SLAVE_WAS_NOT_RUNNING
ER_SLAVE_WAS_RUNNING
ER_SP_GOTO_IN_HNDLR
ER_SP_PROC_TABLE_CORRUPT
ER_SQL_MODE_NO_EFFECT
ER_SR_INVALID_CREATION_CTX
ER_TABLE_NEEDS_UPG_PART
ER_TOO_MUCH_AUTO_TIMESTAMP_COLS
ER_UNEXPECTED_EOF
ER_UNION_TABLES_IN_DIFFERENT_DIR
ER_UNSUPPORTED_BY_REPLICATION_THREAD
ER_UNUSED1
ER_UNUSED2
ER_UNUSED3
ER_UNUSED4
ER_UNUSED5
ER_UNUSED6
ER_VIEW_SELECT_DERIVED_UNUSED
ER_WRONG_MAGIC
ER_WSAS_FAILED
```

- The deprecated `INFORMATION_SCHEMA` `INNODB_LOCKS` and `INNODB_LOCK_WAITS` tables are removed. Use the Performance Schema `data_locks` and `data_lock_waits` tables instead.

  Note

  In MySQL 5.7, the `LOCK_TABLE` column in the `INNODB_LOCKS` table and the `locked_table` column in the `sys` schema `innodb_lock_waits` and `x$innodb_lock_waits` views contain combined schema/table name values. In MySQL 8.0, the `data_locks` table and the `sys` schema views contain separate schema name and table name columns. See Section 27.4.3.9, “The innodb_lock_waits and x$innodb_lock_waits Views”.

- `InnoDB` no longer supports compressed temporary tables. When `innodb_strict_mode` is enabled (the default), `CREATE TEMPORARY TABLE` returns an error if `ROW_FORMAT=COMPRESSED` or

KEY_BLOCK_SIZE is specified. If innodb_strict_mode is disabled, warnings are issued and the temporary table is created using a non-compressed row format.

- InnoDB no longer creates .isl files (InnoDB Symbolic Link files) when creating tablespace data files outside of the MySQL data directory. The innodb_directories option now supports locating tablespace files created outside of the data directory.

  With this change, moving a remote tablespace while the server is offline by manually modifying an .isl file is no longer supported. Moving remote tablespace files is now supported by the innodb_directories option. See Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”.

- The following InnoDB file format variables were removed:

  - innodb_file_format
  - innodb_file_format_check
  - innodb_file_format_max
  - innodb_large_prefix

  File format variables were necessary for creating tables compatible with earlier versions of InnoDB in MySQL 5.1. Now that MySQL 5.1 has reached the end of its product lifecycle, these options are no longer required.

  The FILE_FORMAT column was removed from the INNODB_TABLES and INNODB_TABLESPACES Information Schema tables.

- The innodb_support_xa system variable, which enables support for two-phase commit in XA transactions, was removed. InnoDB support for two-phase commit in XA transactions is always enabled.

- Support for DTrace was removed.

- The JSON_APPEND() function was removed. Use JSON_ARRAY_APPEND() instead.

- Support for placing table partitions in shared InnoDB tablespaces was removed in MySQL 8.0.13. Shared tablespaces include the InnoDB system tablespace and general tablespaces. For information about identifying partitions in shared tablespaces and moving them to file-per-table tablespaces, see Section 2.11.5, “Preparing Your Installation for Upgrade”.

- Support for setting user variables in statements other than SET was deprecated in MySQL 8.0.13. This functionality is subject to removal in MySQL 9.0.

- The --ndb perror option was removed. Use the ndb_perror utility instead.

- The innodb_undo_logs variable was removed. The innodb_rollback_segments variables performs the same function and should be used instead.

- The Innodb_available_undo_logs status variable was removed. The number of available rollback segments per tablespace may be retrieved using SHOW VARIABLES LIKE 'innodb_rollback_segments';

- As of MySQL 8.0.14, the previously deprecated innodb_undo_tablespaces variable is no longer configurable. For more information, see Section 15.6.3.4, “Undo Tablespaces”.

- Support for the ALTER TABLE ... UPGRADE PARTITIONING statement has been removed.

- As of MySQL 8.0.16, support for the `internal_tmp_disk_storage_engine` system variable has been removed; internal temporary tables on disk now always use the `InnoDB` storage engine. See Storage Engine for On-Disk Internal Temporary Tables,for more information.
- The `DISABLE_SHARED CMake` option was unused and has been removed.

1.5 Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 8.0

This section lists server variables, status variables, and options that were added for the first time, have been deprecated, or have been removed in MySQL 8.0.

- Options and Variables Introduced in MySQL 8.0
- Options and Variables Deprecated in MySQL 8.0
- Options and Variables Removed in MySQL 8.0

Options and Variables Introduced in MySQL 8.0

The following system variables, status variables, and options are new in MySQL 8.0, and have not been included in any previous release series.

- `Acl_cache_items_count`: Number of cached privilege objects. Added in MySQL 8.0.0.
- `Audit_log_current_size`: Audit log file current size. Added in MySQL 8.0.11.
- `Audit_log_event_max_drop_size`: Size of largest dropped audited event. Added in MySQL 8.0.11.
- `Audit_log_events`: Number of handled audited events. Added in MySQL 8.0.11.
- `Audit_log_events_filtered`: Number of filtered audited events. Added in MySQL 8.0.11.
- `Audit_log_events_lost`: Number of dropped audited events. Added in MySQL 8.0.11.
- `Audit_log_events_written`: Number of written audited events. Added in MySQL 8.0.11.
- `Audit_log_total_size`: Combined size of written audited events. Added in MySQL 8.0.11.
- `Audit_log_write_waits`: Number of write-delayed audited events. Added in MySQL 8.0.11.
- `Caching_sha2_password_rsa_public_key`: caching_sha2_password authentication plugin RSA public key value. Added in MySQL 8.0.4.
- `Com_alter_resource_group`: Count of ALTER RESOURCE GROUP statements. Added in MySQL 8.0.3.
- `Com_alter_user_default_role`: Count of ALTER USER ... DEFAULT ROLE statements. Added in MySQL 8.0.0.
- `Com_clone`: Count of CLONE statements. Added in MySQL 8.0.2.
- `Com_create_resource_group`: Count of CREATE RESOURCE GROUP statements. Added in MySQL 8.0.3.
- `Com_create_role`: Count of CREATE ROLE statements. Added in MySQL 8.0.0.

- `Com_drop_resource_group`: Count of DROP RESOURCE GROUP statements. Added in MySQL 8.0.3.
- `Com_drop_role`: Count of DROP ROLE statements. Added in MySQL 8.0.0.
- `Com_grant_roles`: Count of GRANT ROLE statements. Added in MySQL 8.0.0.
- `Com_install_component`: Count of INSTALL COMPONENT statements. Added in MySQL 8.0.0.
- `Com_revoke_roles`: Count of REVOKE ROLES statements. Added in MySQL 8.0.0.
- `Com_set_resource_group`: Count of SET RESOURCE GROUP statements. Added in MySQL 8.0.3.
- `Com_set_role`: Count of SET ROLE statements. Added in MySQL 8.0.0.
- `Com_uninstall_component`: Count of UINSTALL COMPONENT statements. Added in MySQL 8.0.0.
- `Compression_algorithm`: Compression algorithm for current connection. Added in MySQL 8.0.18.
- `Compression_level`: Compression level for current connection. Added in MySQL 8.0.18.
- `Connection_control_delay_generated`: How many times the server delayed a connection request. Added in MySQL 8.0.1.
- `Current_tls_ca`: Current value of ssl_ca system variable. Added in MySQL 8.0.16.
- `Current_tls_capath`: Current value of ssl_capath system variable. Added in MySQL 8.0.16.
- `Current_tls_cert`: Current value of ssl_cert system variable. Added in MySQL 8.0.16.
- `Current_tls_cipher`: Current value of ssl_cipher system variable. Added in MySQL 8.0.16.
- `Current_tls_ciphersuites`: Current value of tsl_ciphersuites system variable. Added in MySQL 8.0.16.
- `Current_tls_crl`: Current value of ssl_crl system variable. Added in MySQL 8.0.16.
- `Current_tls_crlpath`: Current value of ssl_crlpath system variable. Added in MySQL 8.0.16.
- `Current_tls_key`: Current value of ssl_key system variable. Added in MySQL 8.0.16.
- `Current_tls_version`: Current value of tls_version system variable. Added in MySQL 8.0.16.
- `Firewall_access_denied`: Number of statements rejected by MySQL Enterprise Firewall. Added in MySQL 8.0.11.
- `Firewall_access_granted`: Number of statements accepted by MySQL Enterprise Firewall. Added in MySQL 8.0.11.
- `Firewall_cached_entries`: Number of statements recorded by MySQL Enterprise Firewall. Added in MySQL 8.0.11.
- `Innodb_system_rows_deleted`: Number of rows deleted from system schema tables. Added in MySQL 8.0.19.
- `Innodb_system_rows_inserted`: Number of rows inserted into system schema tables. Added in MySQL 8.0.19.
- `Innodb_system_rows_read`: Number of rows read from system schema tables. Added in MySQL 8.0.19.

- `Secondary_engine_execution_count`: For future use. Added in MySQL 8.0.13.
- `activate_all_roles_on_login`: Whether to activate all user roles at connect time. Added in MySQL 8.0.2.
- `admin_address`: IP address to bind to for connections on administrative interface. Added in MySQL 8.0.14.
- `admin_port`: TCP/IP number to use for connections on administrative interface. Added in MySQL 8.0.14.
- `audit-log`: Whether to activate the audit log plugin. Added in MySQL 8.0.11.
- `audit_log_buffer_size`: The size of the audit log buffer. Added in MySQL 8.0.11.
- `audit_log_compression`: Audit log file compression method. Added in MySQL 8.0.11.
- `audit_log_connection_policy`: Audit logging policy for connection-related events. Added in MySQL 8.0.11.
- `audit_log_current_session`: Whether to audit current session. Added in MySQL 8.0.11.
- `audit_log_encryption`: Audit log file encryption method. Added in MySQL 8.0.11.
- `audit_log_exclude_accounts`: Accounts not to audit. Added in MySQL 8.0.11.
- `audit_log_file`: The name of the audit log file. Added in MySQL 8.0.11.
- `audit_log_filter_id`: ID of current audit log filter. Added in MySQL 8.0.11.
- `audit_log_flush`: Close and reopen the audit log file. Added in MySQL 8.0.11.
- `audit_log_format`: The audit log file format. Added in MySQL 8.0.11.
- `audit_log_include_accounts`: Accounts to audit. Added in MySQL 8.0.11.
- `audit_log_password_history_keep_days`: Number of days to keep archived audit log encryption passwords. Added in MySQL 8.0.17.
- `audit_log_policy`: Audit logging policy. Added in MySQL 8.0.11.
- `audit_log_read_buffer_size`: Audit log file read buffer size. Added in MySQL 8.0.11.
- `audit_log_rotate_on_size`: Close and reopen the audit log file at a certain size. Added in MySQL 8.0.11.
- `audit_log_statement_policy`: Audit logging policy for statement-related events. Added in MySQL 8.0.11.
- `audit_log_strategy`: The audit logging strategy. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_auth_method_name`: Authentication method name. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_bind_base_dn`: LDAP server base distinguished name. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_bind_root_dn`: LDAP server root distinguished name. Added in MySQL 8.0.11.

- `authentication_ldap_sasl_bind_root_pwd`: LDAP server root bind password. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_ca_path`: LDAP server certificate authority file name. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_group_search_attr`: LDAP server group search attribute. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_group_search_filter`: LDAP custom group search filter. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_init_pool_size`: LDAP server initial connection pool size. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_log_status`: LDAP server log level. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_max_pool_size`: LDAP server maximum connection pool size. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_server_host`: LDAP server host name or IP address. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_server_port`: LDAP server port number. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_tls`: Whether to use encrypted connections to LDAP server. Added in MySQL 8.0.11.
- `authentication_ldap_sasl_user_search_attr`: LDAP server user search attribute. Added in MySQL 8.0.11.
- `authentication_ldap_simple_auth_method_name`: Authentication method name. Added in MySQL 8.0.11.
- `authentication_ldap_simple_bind_base_dn`: LDAP server base distinguished name. Added in MySQL 8.0.11.
- `authentication_ldap_simple_bind_root_dn`: LDAP server root distinguished name. Added in MySQL 8.0.11.
- `authentication_ldap_simple_bind_root_pwd`: LDAP server root bind password. Added in MySQL 8.0.11.
- `authentication_ldap_simple_ca_path`: LDAP server certificate authority file name. Added in MySQL 8.0.11.
- `authentication_ldap_simple_group_search_attr`: LDAP server group search attribute. Added in MySQL 8.0.11.
- `authentication_ldap_simple_group_search_filter`: LDAP custom group search filter. Added in MySQL 8.0.11.
- `authentication_ldap_simple_init_pool_size`: LDAP server initial connection pool size. Added in MySQL 8.0.11.
- `authentication_ldap_simple_log_status`: LDAP server log level. Added in MySQL 8.0.11.
- `authentication_ldap_simple_max_pool_size`: LDAP server maximum connection pool size. Added in MySQL 8.0.11.

- `authentication_ldap_simple_server_host`: LDAP server host name or IP address. Added in MySQL 8.0.11.
- `authentication_ldap_simple_server_port`: LDAP server port number. Added in MySQL 8.0.11.
- `authentication_ldap_simple_tls`: Whether to use encrypted connections to LDAP server. Added in MySQL 8.0.11.
- `authentication_ldap_simple_user_search_attr`: LDAP server user search attribute. Added in MySQL 8.0.11.
- `authentication_windows_log_level`: Windows authentication plugin logging level. Added in MySQL 8.0.11.
- `authentication_windows_use_principal_name`: Whether to use Windows authentication plugin principal name. Added in MySQL 8.0.11.
- `binlog_encryption`: Enable encryption for binary log files and relay log files on this server. Added in MySQL 8.0.14.
- `binlog_expire_logs_seconds`: Purge binary logs after this many seconds. Added in MySQL 8.0.1.
- `binlog_rotate_encryption_master_key_at_startup`: Rotate the binary log master key at server startup. Added in MySQL 8.0.14.
- `binlog_row_metadata`: Configures the amount of table related metadata binary logged when using row-based logging.. Added in MySQL 8.0.1.
- `binlog_row_value_options`: Enables binary logging of partial JSON updates for row-based replication.. Added in MySQL 8.0.3.
- `binlog_transaction_dependency_history_size`: Number of row hashes kept for looking up transaction that last updated some row.. Added in MySQL 8.0.1.
- `binlog_transaction_dependency_tracking`: Source of dependency information (commit timestamps or transaction write sets) from which to assess which transactions can be executed in parallel by slave's multithreaded applier.. Added in MySQL 8.0.1.
- `caching_sha2_password_auto_generate_rsa_keys`: Whether to autogenerate RSA key-pair files. Added in MySQL 8.0.4.
- `caching_sha2_password_private_key_path`: SHA2 authentication plugin private key path name. Added in MySQL 8.0.3.
- `caching_sha2_password_public_key_path`: SHA2 authentication plugin public key path name. Added in MySQL 8.0.3.
- `clone_autotune_concurrency`: Enables dynamic spawning of threads for remote cloning operations. Added in MySQL 8.0.17.
- `clone_buffer_size`: Defines the size of the intermediate buffer on the donor MySQL server instance. Added in MySQL 8.0.17.
- `clone_ddl_timeout`: The number of seconds that a cloning operation waits for backup lock. Added in MySQL 8.0.17.
- `clone_enable_compression`: Enables compression of data at the network layer during cloning. Added in MySQL 8.0.17.

- `clone_max_concurrency`: The maximum number of concurrent threads used to perform cloning operation. Added in MySQL 8.0.17.
- `clone_max_data_bandwidth`: The maximum data transfer rate in MiB per second for a remote cloning operation. Added in MySQL 8.0.17.
- `clone_max_network_bandwidth`: The maximum network transfer rate in MiB per second for a remote cloning operation. Added in MySQL 8.0.17.
- `clone_ssl_ca`: Specifies the path to the certificate authority (CA) file. Added in MySQL 8.0.14.
- `clone_ssl_cert`: Specifies the path to the public key certificate file. Added in MySQL 8.0.14.
- `clone_ssl_key`: Specifies the path to the private key file. Added in MySQL 8.0.14.
- `clone_valid_donor_list`: Defines donor host addresses for remote cloning operations. Added in MySQL 8.0.17.
- `connection_control_failed_connections_threshold`: Consecutive failed connection attempts before delays occur. Added in MySQL 8.0.1.
- `connection_control_max_connection_delay`: Maximum delay (milliseconds) for server response to failed connection attempts. Added in MySQL 8.0.1.
- `connection_control_min_connection_delay`: Minimum delay (milliseconds) for server response to failed connection attempts. Added in MySQL 8.0.1.
- `create_admin_listener_thread`: Whether to use dedicated listening thread for connections on administrative interface. Added in MySQL 8.0.14.
- `cte_max_recursion_depth`: Common table expression maximum recursion depth. Added in MySQL 8.0.3.
- `ddl-rewriter`: Whether to activate the ddl_rewriter plugin. Added in MySQL 8.0.16.
- `default_collation_for_utf8mb4`: Default collation for utf8mb4 character set. Added in MySQL 8.0.11.
- `default_table_encryption`: The default schema and tablespace encryption setting. Added in MySQL 8.0.16.
- `dragnet.Status`: Result of most recent assignment to dragnet.log_error_filter_rules. Added in MySQL 8.0.12.
- `dragnet.log_error_filter_rules`: Filter rules for error logging. Added in MySQL 8.0.4.
- `early-plugin-load`: Specify plugins to load before loading mandatory built-in plugins and before storage engine initialization. Added in MySQL 8.0.0.
- `group_replication_autorejoin_tries`: Number of tries that a member makes to automatically rejoin the group. Added in MySQL 8.0.16.
- `group_replication_clone_threshold`: The transaction number gap between the donor and recipient above which a remote cloning operation is used for state transfer. Added in MySQL 8.0.17.
- `group_replication_communication_debug_options`: The level of debugging messages for Group Replication components.. Added in MySQL 8.0.3.
- `group_replication_communication_max_message_size`: Maximum message size for Group Replication communications, larger messages are fragmented. Added in MySQL 8.0.16.

- `group_replication_consistency`: The type of transaction consistency guarantee which the group provides. Added in MySQL 8.0.14.
- `group_replication_exit_state_action`: How the instance behaves when it leaves the group involuntarily. Added in MySQL 8.0.12.
- `group_replication_flow_control_hold_percent`: Defines what percentage of the group quota remains unused. Added in MySQL 8.0.2.
- `group_replication_flow_control_max_commit_quota`: Defines the maximum flow control quota of the group. Added in MySQL 8.0.2.
- `group_replication_flow_control_member_quota_percent`: Defines the percentage of the quota that a member should assume is available for itself when calculating the quotas. Added in MySQL 8.0.2.
- `group_replication_flow_control_min_quota`: Controls the lowest flow control quota that can be assigned to a member. Added in MySQL 8.0.2.
- `group_replication_flow_control_min_recovery_quota`: Controls the lowest quota that can be assigned to a member because of another recovering member in the group. Added in MySQL 8.0.2.
- `group_replication_flow_control_period`: Defines how many seconds to wait between flow control iterations. Added in MySQL 8.0.2.
- `group_replication_flow_control_release_percent`: Defines how the group quota should be released when flow control no longer needs to throttle the writer members. Added in MySQL 8.0.2.
- `group_replication_member_expel_timeout`: The time between a suspected failure of a group member and it being expelled from the group, causing a group membership reconfiguration.. Added in MySQL 8.0.13.
- `group_replication_member_weight`: Chance of this member being elected as primary. Added in MySQL 8.0.2.
- `group_replication_message_cache_size`: Maximum memory for the message cache in the group communication engine (XCom). Added in MySQL 8.0.16.
- `group_replication_recovery_compression_algorithm`: Permitted compression algorithms for outgoing recovery connections. Added in MySQL 8.0.18.
- `group_replication_recovery_get_public_key`: Whether to accept preference about fetching public key from master. Added in MySQL 8.0.4.
- `group_replication_recovery_public_key_path`: To accept public key information. Added in MySQL 8.0.4.
- `group_replication_recovery_zstd_compression_level`: Compression level for recovery connections that use zstd compression. Added in MySQL 8.0.18.
- `group_replication_unreachable_majority_timeout`: How long to wait for network partitions that result in a minority to leave the group. Added in MySQL 8.0.2.
- `histogram_generation_max_mem_size`: Maximum memory for creating histogram statistics. Added in MySQL 8.0.2.
- `immediate_server_version`: The MySQL Server release number of the server that is the immediate master in a replication topology. Added in MySQL 8.0.14.

- `information_schema_stats_expiry`: Expiration setting for cached table statistics. Added in MySQL 8.0.3.
- `innodb_buffer_pool_debug`: Permits multiple buffer pool instances when the buffer pool is less than 1GB in size. Added in MySQL 8.0.0.
- `innodb_buffer_pool_in_core_file`: Controls writing of buffer pool pages to core files. Added in MySQL 8.0.14.
- `innodb_checkpoint_disabled`: Disables checkpoints so that a deliberate server exit always initiates recovery. Added in MySQL 8.0.2.
- `innodb_ddl_log_crash_reset_debug`: A debug option that resets DDL log crash injection counters. Added in MySQL 8.0.3.
- `innodb_deadlock_detect`: Enables or disables deadlock detection. Added in MySQL 8.0.0.
- `innodb_dedicated_server`: Enables automatic configuration of buffer pool size, log file size, and flush method. Added in MySQL 8.0.3.
- `innodb_directories`: Defines directories to scan at startup for tablespace data files. Added in MySQL 8.0.4.
- `innodb_fsync_threshold`: Controls how often InnoDB calls fsync when creating a new file. Added in MySQL 8.0.13.
- `innodb_idle_flush_pct`: Limits I/0 operations when InnoDB is idle. Added in MySQL 8.0.18.
- `innodb_log_checkpoint_fuzzy_now`: A debug option that forces InnoDB to write a fuzzy checkpoint. Added in MySQL 8.0.13.
- `innodb_log_spin_cpu_abs_lwm`: Minimum amount of CPU usage below which user threads no longer spin while waiting for flushed redo. Added in MySQL 8.0.11.
- `innodb_log_spin_cpu_pct_hwm`: Maximum amount of CPU usage above which user threads no longer spin while waiting for flushed redo. Added in MySQL 8.0.11.
- `innodb_log_wait_for_flush_spin_hwm`: The maximum average log flush time beyond which user threads no longer spin while waiting for flushed redo. Added in MySQL 8.0.11.
- `innodb_parallel_read_threads`: Defines the number of threads for parallel index reads. Added in MySQL 8.0.14.
- `innodb_print_ddl_logs`: Whether or not to print DDL logs to the error log. Added in MySQL 8.0.3.
- `innodb_redo_log_archive_dirs`: Labeled redo log archive directories. Added in MySQL 8.0.17.
- `innodb_redo_log_encrypt`: Controls encryption of redo log data for encrypted tablespaces. Added in MySQL 8.0.1.
- `innodb_scan_directories`: Defines directories to scan for tablespace files during InnoDB recovery. Added in MySQL 8.0.2.
- `innodb_spin_wait_pause_multiplier`: Defines a multiplier value used to determine the number of PAUSE instructions in spin-wait loops.. Added in MySQL 8.0.16.
- `innodb_stats_include_delete_marked`: Include delete-marked records when calculating persistent InnoDB statistics. Added in MySQL 8.0.1.

- innodb_temp_tablespaces_dir: Session temporary tablespaces path. Added in MySQL 8.0.13.
- innodb_tmpdir: The directory location for the temporary table files created during online ALTER TABLE operations. Added in MySQL 8.0.0.
- innodb_undo_log_encrypt: Controls encryption of undo log data for encrypted tablespaces. Added in MySQL 8.0.1.
- internal_tmp_mem_storage_engine: Defines the storage to use for internal in-memory temporary tables. Added in MySQL 8.0.2.
- keyring-migration-destination: Key migration destination keyring plugin. Added in MySQL 8.0.4.
- keyring-migration-host: Host name for connecting to running server for key migration. Added in MySQL 8.0.4.
- keyring-migration-password: Password for connecting to running server for key migration. Added in MySQL 8.0.4.
- keyring-migration-port: TCP/IP port number for connecting to running server for key migration. Added in MySQL 8.0.4.
- keyring-migration-socket: Unix socket file or Windows named pipe for connecting to running server for key migration. Added in MySQL 8.0.4.
- keyring-migration-source: Key migration source keyring plugin. Added in MySQL 8.0.4.
- keyring-migration-user: User name for connecting to running server for key migration. Added in MySQL 8.0.4.
- keyring_aws_cmk_id: AWS keyring plugin customer master key ID value. Added in MySQL 8.0.11.
- keyring_aws_conf_file: AWS keyring plugin configuration file location. Added in MySQL 8.0.11.
- keyring_aws_data_file: AWS keyring plugin storage file location. Added in MySQL 8.0.11.
- keyring_aws_region: AWS keyring plugin region. Added in MySQL 8.0.11.
- keyring_encrypted_file_data: keyring_encrypted_file plugin data file. Added in MySQL 8.0.11.
- keyring_encrypted_file_password: keyring_encrypted_file plugin password. Added in MySQL 8.0.11.
- keyring_okv_conf_dir: Oracle Key Vault keyring plugin configuration directory. Added in MySQL 8.0.11.
- keyring_operations: Whether keyring operations are enabled. Added in MySQL 8.0.4.
- lock_order: Whether to enable LOCK_ORDER tool at runtime. Added in MySQL 8.0.17.
- lock_order_debug_loop: Whether to cause debug assert when LOCK_ORDER tool encounters dependency flagged as a loop. Added in MySQL 8.0.17.
- lock_order_debug_missing_arc: Whether to cause debug assert when LOCK_ORDER tool encounters undeclared dependency. Added in MySQL 8.0.17.
- lock_order_debug_missing_key: Whether to cause debug assert when LOCK_ORDER tool encounters object not properly instrumented with the Performance Schema. Added in MySQL 8.0.17.

- `lock_order_debug_missing_unlock`: Whether to cause debug assert when LOCK_ORDER tool encounters lock that is destroyed while still held. Added in MySQL 8.0.17.
- `lock_order_dependencies`: Path to the lock_order_dependencies.txt file. Added in MySQL 8.0.17.
- `lock_order_extra_dependencies`: Path to a second dependency file. Added in MySQL 8.0.17.
- `lock_order_output_directory`: Directory where LOCK_ORDER tool writes logs. Added in MySQL 8.0.17.
- `lock_order_print_txt`: Whether to perform lock-order graph analysis and print textual report. Added in MySQL 8.0.17.
- `lock_order_trace_loop`: Whether to print log file trace when LOCK_ORDER tool encounters dependency flagged as a loop. Added in MySQL 8.0.17.
- `lock_order_trace_missing_arc`: Whether to print log file trace when LOCK_ORDER tool encounters undeclared dependency. Added in MySQL 8.0.17.
- `lock_order_trace_missing_key`: Whether to print log file trace when LOCK_ORDER tool encounters object not properly instrumented with the Performance Schema. Added in MySQL 8.0.17.
- `lock_order_trace_missing_unlock`: Whether to print log file trace when LOCK_ORDER tool encounters lock that is destroyed while still held. Added in MySQL 8.0.17.
- `log_error_filter_rules`: Filter rules for error logging. Added in MySQL 8.0.2.
- `log_error_services`: Components to use for error logging. Added in MySQL 8.0.2.
- `log_error_suppression_list`: Warning/information error log messages to suppress. Added in MySQL 8.0.13.
- `log_slow_extra`: Whether to write extra information to the slow query log file. Added in MySQL 8.0.14.
- `mandatory_roles`: Automatically granted roles for all users. Added in MySQL 8.0.2.
- `mysql_firewall_mode`: Whether MySQL Enterprise Firewall is operational. Added in MySQL 8.0.11.
- `mysql_firewall_trace`: Whether to enable firewall trace. Added in MySQL 8.0.11.
- `mysqlx`: Whether X Plugin is initialized. Added in MySQL 8.0.11.
- `mysqlx_interactive_timeout`: Number of seconds to wait for interactive clients to timeout. Added in MySQL 8.0.4.
- `mysqlx_read_timeout`: Number of seconds to wait for blocking read operations to complete. Added in MySQL 8.0.4.
- `mysqlx_wait_timeout`: Number of seconds to wait for activity on a connection. Added in MySQL 8.0.4.
- `mysqlx_write_timeout`: Number of seconds to wait for blocking write operations to complete. Added in MySQL 8.0.4.
- `named_pipe_full_access_group`: Name of Windows group granted full access to the named pipe. Added in MySQL 8.0.14.
- `no-dd-upgrade`: Prevent automatic upgrade of data dictionary tables at startup. Added in MySQL 8.0.4.

- `no-monitor`: Do not fork monitor process required for RESTART. Added in MySQL 8.0.12.
- `original_commit_timestamp`: The time when a transaction was committed on the original master. Added in MySQL 8.0.1.
- `original_server_version`: The MySQL Server release number of the server where a transaction was originally committed. Added in MySQL 8.0.14.
- `partial_revokes`: Whether partial revocation is enabled. Added in MySQL 8.0.16.
- `password_history`: Number of password changes required before password reuse. Added in MySQL 8.0.3.
- `password_require_current`: Whether password changes require current password verification. Added in MySQL 8.0.13.
- `password_reuse_interval`: Number of days elapsed required before password reuse. Added in MySQL 8.0.3.
- `performance_schema_max_digest_sample_age`: The query resample age in seconds. Added in MySQL 8.0.3.
- `persist_only_admin_x509_subject`: SSL certificate X.509 Subject that enables persisting persist-restricted system variables. Added in MySQL 8.0.14.
- `persisted_globals_load`: Whether to load persisted configuration settings. Added in MySQL 8.0.0.
- `print_identified_with_as_hex`: For SHOW CREATE USER, print hash values containing unprintable characters in hex. Added in MySQL 8.0.17.
- `protocol_compression_algorithms`: Permitted compression algorithms for incoming connections. Added in MySQL 8.0.18.
- `regexp_stack_limit`: Regular expression match stack size limit. Added in MySQL 8.0.4.
- `regexp_time_limit`: Regular expression match timeout. Added in MySQL 8.0.4.
- `resultset_metadata`: Whether the server returns result set metadata. Added in MySQL 8.0.3.
- `rpl_read_size`: Set the minimum amount of data in bytes that is read from the binary log files and relay log files. Added in MySQL 8.0.11.
- `secondary_engine_cost_threshold`: For future use. Added in MySQL 8.0.16.
- `show_create_table_skip_secondary_engine`: Whether to exclude the SECONDARY ENGINE clause from SHOW CREATE TABLE output. Added in MySQL 8.0.18.
- `show_create_table_verbosity`: Whether to display ROW_FORMAT in SHOW CREATE TABLE even if it has the default value. Added in MySQL 8.0.11.
- `sql_require_primary_key`: Whether tables must have a primary key. Added in MySQL 8.0.13.
- `ssl_fips_mode`: Whether to enable FIPS mode on server side. Added in MySQL 8.0.11.
- `syseventlog.facility`: Facility for syslog messages. Added in MySQL 8.0.13.
- `syseventlog.include_pid`: Whether to include server PID in syslog messages. Added in MySQL 8.0.13.

- `syseventlog.tag`: Tag for server identifier in syslog messages. Added in MySQL 8.0.13.
- `table_encryption_privilege_check`: Enables the TABLE_ENCRYPTION_ADMIN privilege check. Added in MySQL 8.0.16.
- `temptable_max_ram`: Defines the maximum amount of memory that can occupied by the TempTable storage engine before data is stored on disk. Added in MySQL 8.0.2.
- `temptable_use_mmap`: Defines whether the TempTable storage engine allocates memory-mapped files when the temptable_max_ram threshold is reached.. Added in MySQL 8.0.16.
- `thread_pool_algorithm`: The thread pool algorithm. Added in MySQL 8.0.11.
- `thread_pool_high_priority_connection`: Whether the current session is high priority. Added in MySQL 8.0.11.
- `thread_pool_max_active_query_threads`: Maximum permissible number of active query threads per group. Added in MySQL 8.0.19.
- `thread_pool_max_unused_threads`: Maximum permissible number of unused threads. Added in MySQL 8.0.11.
- `thread_pool_prio_kickup_timer`: How long before a statement is moved to high-priority execution. Added in MySQL 8.0.11.
- `thread_pool_size`: Number of thread groups in the thread pool. Added in MySQL 8.0.11.
- `thread_pool_stall_limit`: How long before a statement is defined as stalled. Added in MySQL 8.0.11.
- `tls_ciphersuites`: Permissible TLSv1.3 ciphersuites for encrypted connections. Added in MySQL 8.0.16.
- `upgrade`: Control automatic upgrade at startup. Added in MySQL 8.0.16.
- `use_secondary_engine`: For future use. Added in MySQL 8.0.13.
- `validate-config`: Validate server configuration. Added in MySQL 8.0.16.
- `validate_password.check_user_name`: Whether to check passwords against user name. Added in MySQL 8.0.4.
- `validate_password.dictionary_file`: validate_password dictionary file. Added in MySQL 8.0.4.
- `validate_password.dictionary_file_last_parsed`: When the dictionary file was last parsed. Added in MySQL 8.0.4.
- `validate_password.dictionary_file_words_count`: Number of words in dictionary file. Added in MySQL 8.0.4.
- `validate_password.length`: validate_password required password length. Added in MySQL 8.0.4.
- `validate_password.mixed_case_count`: validate_password required number of uppercase/ lowercase characters. Added in MySQL 8.0.4.
- `validate_password.number_count`: validate_password required number of digit characters. Added in MySQL 8.0.4.
- `validate_password.policy`: validate_password password policy. Added in MySQL 8.0.4.

- `validate_password.special_char_count`: validate_password required number of special characters. Added in MySQL 8.0.4.
- `version_compile_zlib`: Version of compiled-in zlib library. Added in MySQL 8.0.11.
- `windowing_use_high_precision`: Whether to compute window functions to high precision. Added in MySQL 8.0.2.

Options and Variables Deprecated in MySQL 8.0

The following system variables, status variables, and options have been deprecated in MySQL 8.0.

- `Compression`: Whether the client connection uses compression in the client/server protocol. Deprecated as of MySQL 8.0.18.
- `expire_logs_days`: Purge binary logs after this many days. Deprecated as of MySQL 8.0.3.
- `innodb_undo_tablespaces`: Number of tablespace files that rollback segments are divided between. Deprecated as of MySQL 8.0.4.
- `log_bin_use_v1_row_events`: Whether server is using version 1 binary log row events. Deprecated as of MySQL 8.0.18.
- `log_syslog`: Whether to write error log to syslog. Deprecated as of MySQL 8.0.2.
- `master-info-file`: The location and name of the file that remembers the master and where the I/O replication thread is in the master's binary logs. Deprecated as of MySQL 8.0.18.
- `no-dd-upgrade`: Prevent automatic upgrade of data dictionary tables at startup. Deprecated as of MySQL 8.0.16.
- `relay_log_info_file`: File in which the slave records information about the relay logs. Deprecated as of MySQL 8.0.18.
- `slave-rows-search-algorithms`: Determines search algorithms used for slave update batching. Any 2 or 3 from the list INDEX_SEARCH, TABLE_SCAN, HASH_SCAN. Deprecated as of MySQL 8.0.18.
- `slave_compressed_protocol`: Use compression of master/slave protocol. Deprecated as of MySQL 8.0.18.
- `slave_rows_search_algorithms`: Determines search algorithms used for slave update batching. Any 2 or 3 from the list INDEX_SEARCH, TABLE_SCAN, HASH_SCAN.. Deprecated as of MySQL 8.0.18.
- `symbolic-links`: Permit symbolic links for MyISAM tables. Deprecated as of MySQL 8.0.2.

Options and Variables Removed in MySQL 8.0

The following system variables, status variables, and options have been removed in MySQL 8.0.

- `Com_alter_db_upgrade`: Count of ALTER DATABASE ... UPGRADE DATA DIRECTORY NAME statements. Removed in MySQL 8.0.0.
- `Innodb_available_undo_logs`: Display the total number of InnoDB rollback segments; different from innodb_rollback_segments, which displays the number of active rollback segments. Removed in MySQL 8.0.2.

- `Qcache_free_blocks`: Number of free memory blocks in the query cache. Removed in MySQL 8.0.3.
- `Qcache_free_memory`: The amount of free memory for the query cache. Removed in MySQL 8.0.3.
- `Qcache_hits`: Number of query cache hits. Removed in MySQL 8.0.3.
- `Qcache_inserts`: Number of query cache inserts. Removed in MySQL 8.0.3.
- `Qcache_lowmem_prunes`: Number of queries that were deleted from the query cache due to lack of free memory in the cache. Removed in MySQL 8.0.3.
- `Qcache_not_cached`: Number of noncached queries (not cacheable, or not cached due to the query_cache_type setting). Removed in MySQL 8.0.3.
- `Qcache_queries_in_cache`: Number of queries registered in the query cache. Removed in MySQL 8.0.3.
- `Qcache_total_blocks`: The total number of blocks in the query cache. Removed in MySQL 8.0.3.
- `Slave_heartbeat_period`: The slave's replication heartbeat interval, in seconds. Removed in MySQL 8.0.1.
- `Slave_last_heartbeat`: Shows when the latest heartbeat signal was received, in TIMESTAMP format. Removed in MySQL 8.0.1.
- `Slave_received_heartbeats`: Number of heartbeats received by a replication slave since previous reset. Removed in MySQL 8.0.1.
- `Slave_retried_transactions`: The total number of times since startup that the replication slave SQL thread has retried transactions. Removed in MySQL 8.0.1.
- `Slave_running`: The state of this server as a replication slave (slave I/O thread status). Removed in MySQL 8.0.1.
- `bootstrap`: Used by mysql installation scripts. Removed in MySQL 8.0.0.
- `date_format`: The DATE format (unused). Removed in MySQL 8.0.3.
- `datetime_format`: The DATETIME/TIMESTAMP format (unused). Removed in MySQL 8.0.3.
- `des-key-file`: Load keys for des_encrypt() and des_encrypt from given file. Removed in MySQL 8.0.3.
- `group_replication_allow_local_disjoint_gtids_join`: Allow the current server to join the group even if it has transactions not present in the group. Removed in MySQL 8.0.4.
- `have_crypt`: Availability of the crypt() system call. Removed in MySQL 8.0.3.
- `ignore-db-dir`: Treat directory as nondatabase directory. Removed in MySQL 8.0.0.
- `ignore_builtin_innodb`: Ignore the built-in InnoDB. Removed in MySQL 8.0.3.
- `ignore_db_dirs`: Directories treated as nondatabase directories. Removed in MySQL 8.0.0.
- `innodb_checksums`: Enable InnoDB checksums validation. Removed in MySQL 8.0.0.
- `innodb_disable_resize_buffer_pool_debug`: Disables resizing of the InnoDB buffer pool. Removed in MySQL 8.0.0.

- `innodb_file_format`: The format for new InnoDB tables. Removed in MySQL 8.0.0.
- `innodb_file_format_check`: Whether InnoDB performs file format compatibility checking. Removed in MySQL 8.0.0.
- `innodb_file_format_max`: The file format tag in the shared tablespace. Removed in MySQL 8.0.0.
- `innodb_large_prefix`: Enables longer keys for column prefix indexes. Removed in MySQL 8.0.0.
- `innodb_locks_unsafe_for_binlog`: Force InnoDB not to use next-key locking. Instead use only row-level locking. Removed in MySQL 8.0.0.
- `innodb_scan_directories`: Defines directories to scan for tablespace files during InnoDB recovery. Removed in MySQL 8.0.4.
- `innodb_stats_sample_pages`: Number of index pages to sample for index distribution statistics. Removed in MySQL 8.0.0.
- `innodb_support_xa`: Enable InnoDB support for the XA two-phase commit. Removed in MySQL 8.0.0.
- `innodb_undo_logs`: Defines the number of undo logs (rollback segments) used by InnoDB; an alias for innodb_rollback_segments. Removed in MySQL 8.0.2.
- `internal_tmp_disk_storage_engine`: Storage engine for internal temporary tables. Removed in MySQL 8.0.16.
- `log-warnings`: Log some noncritical warnings to the log file. Removed in MySQL 8.0.3.
- `log_builtin_as_identified_by_password`: Whether to log CREATE/ALTER USER, GRANT in backward-compatible fashion. Removed in MySQL 8.0.11.
- `log_error_filter_rules`: Filter rules for error logging. Removed in MySQL 8.0.4.
- `log_syslog`: Whether to write error log to syslog. Removed in MySQL 8.0.13.
- `log_syslog_facility`: Facility for syslog messages. Removed in MySQL 8.0.13.
- `log_syslog_include_pid`: Whether to include server PID in syslog messages. Removed in MySQL 8.0.13.
- `log_syslog_tag`: Tag for server identifier in syslog messages. Removed in MySQL 8.0.13.
- `max_tmp_tables`: Unused. Removed in MySQL 8.0.3.
- `metadata_locks_cache_size`: Size of the metadata locks cache. Removed in MySQL 8.0.13.
- `metadata_locks_hash_instances`: Number of metadata lock hashes. Removed in MySQL 8.0.13.
- `multi_range_count`: The maximum number of ranges to send to a table handler at once during range selects. Removed in MySQL 8.0.3.
- `old_passwords`: Selects password hashing method for PASSWORD(). Removed in MySQL 8.0.11.
- `partition`: Enable (or disable) partitioning support. Removed in MySQL 8.0.0.
- `query_cache_limit`: Do not cache results that are bigger than this. Removed in MySQL 8.0.3.
- `query_cache_min_res_unit`: Minimal size of unit in which space for results is allocated (last unit will be trimmed after writing all result data). Removed in MySQL 8.0.3.

- `query_cache_size`: The memory allocated to store results from old queries. Removed in MySQL 8.0.3.
- `query_cache_type`: Query cache type. Removed in MySQL 8.0.3.
- `query_cache_wlock_invalidate`: Invalidate queries in query cache on LOCK for write. Removed in MySQL 8.0.3.
- `secure-auth`: Disallow authentication for accounts that have old (pre-4.1) passwords. Removed in MySQL 8.0.3.
- `show_compatibility_56`: Compatibility for SHOW STATUS/VARIABLES. Removed in MySQL 8.0.1.
- `skip-partition`: Do not enable user-defined partitioning. Removed in MySQL 8.0.0.
- `sync_frm`: Sync .frm to disk on create. Enabled by default. Removed in MySQL 8.0.0.
- `temp-pool`: Using this option will cause most temporary files created to use a small set of names, rather than a unique name for each new file. Removed in MySQL 8.0.1.
- `time_format`: The TIME format (unused). Removed in MySQL 8.0.3.
- `tx_isolation`: The default transaction isolation level. Removed in MySQL 8.0.3.
- `tx_read_only`: Default transaction access mode. Removed in MySQL 8.0.3.

1.6 MySQL Information Sources

This section lists sources of additional information that you may find helpful, such as MySQL websites, mailing lists, user forums, and Internet Relay Chat.

1.6.1 MySQL Websites

The primary website for MySQL documentation is https://dev.mysql.com/doc/. Online and downloadable documentation formats are available for the MySQL Reference Manual, MySQL Connectors, and more.

The MySQL developers provide information about new and upcoming features as the MySQL Server Blog.

1.6.2 MySQL Community Support at the MySQL Forums

The forums at http://forums.mysql.com are an important community resource. Many forums are available, grouped into these general categories:

- Migration
- MySQL Usage
- MySQL Connectors
- Programming Languages
- Tools
- 3rd-Party Applications
- Storage Engines
- MySQL Technology

- SQL Standards
- Business

1.6.3 MySQL Enterprise

Oracle offers technical support in the form of MySQL Enterprise. For organizations that rely on the MySQL DBMS for business-critical production applications, MySQL Enterprise is a commercial subscription offering which includes:

- MySQL Enterprise Server
- MySQL Enterprise Monitor
- Monthly Rapid Updates and Quarterly Service Packs
- MySQL Knowledge Base
- 24x7 Technical and Consultative Support

MySQL Enterprise is available in multiple tiers, giving you the flexibility to choose the level of service that best matches your needs. For more information, see MySQL Enterprise.

1.7 How to Report Bugs or Problems

Before posting a bug report about a problem, please try to verify that it is a bug and that it has not been reported already:

- Start by searching the MySQL online manual at https://dev.mysql.com/doc/. We try to keep the manual up to date by updating it frequently with solutions to newly found problems. In addition, the release notes accompanying the manual can be particularly useful since it is quite possible that a newer version contains a solution to your problem. The release notes are available at the location just given for the manual.
- If you get a parse error for an SQL statement, please check your syntax closely. If you cannot find something wrong with it, it is extremely likely that your current version of MySQL Server doesn't support the syntax you are using. If you are using the current version and the manual doesn't cover the syntax that you are using, MySQL Server doesn't support your statement.

  If the manual covers the syntax you are using, but you have an older version of MySQL Server, you should check the MySQL change history to see when the syntax was implemented. In this case, you have the option of upgrading to a newer version of MySQL Server.
- For solutions to some common problems, see Section B.4, “Problems and Common Errors”.
- Search the bugs database at http://bugs.mysql.com/ to see whether the bug has been reported and fixed.
- You can also use http://www.mysql.com/search/ to search all the Web pages (including the manual) that are located at the MySQL website.

If you cannot find an answer in the manual, the bugs database, or the mailing list archives, check with your local MySQL expert. If you still cannot find an answer to your question, please use the following guidelines for reporting the bug.

The normal way to report bugs is to visit http://bugs.mysql.com/, which is the address for our bugs database. This database is public and can be browsed and searched by anyone. If you log in to the system, you can enter new reports.

Bugs posted in the bugs database at http://bugs.mysql.com/ that are corrected for a given release are noted in the release notes.

If you find a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to `<secalert_us@oracle.com>`. Exception: Support customers should report all problems, including security bugs, to Oracle Support at http://support.oracle.com/.

To discuss problems with other users, you can use the MySQL Community Slack.

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release. This section helps you write your report correctly so that you do not waste your time doing things that may not help us much or at all. Please read this section carefully and make sure that all the information described here is included in your report.

Preferably, you should test the problem using the latest production or development version of MySQL Server before posting. Anyone should be able to repeat the bug by just using `mysql test < script_file` on your test case or by running the shell or Perl script that you include in the bug report. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

It is most helpful when a good description of the problem is included in the bug report. That is, give a good example of everything you did that led to the problem and describe, in exact detail, the problem itself. The best reports are those that include a full example showing how to reproduce the bug or problem. See Section 29.5, “Debugging and Porting MySQL”.

Remember that it is possible for us to respond to a report containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details do not matter. A good principle to follow is that if you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of the MySQL distribution that you use, and (b) not fully describing the platform on which the MySQL server is installed (including the platform type and version number). These are highly relevant pieces of information, and in 99 cases out of 100, the bug report is useless without them. Very often we get questions like, “Why doesn't this work for me?” Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has been fixed in newer MySQL versions. Errors often are platform-dependent. In such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If you compiled MySQL from source, remember also to provide information about your compiler if it is related to the problem. Often people find bugs in compilers and think the problem is MySQL-related. Most compilers are under development all the time and become better version by version. To determine whether your problem depends on your compiler, we need to know what compiler you used. Note that every compiling problem should be regarded as a bug and reported accordingly.

If a program produces an error message, it is very important to include the message in your report. If we try to search for something from the archives, it is better that the error message reported exactly matches the one that the program produces. (Even the lettercase should be observed.) It is best to copy and paste the entire error message into your report. You should never try to reproduce the message from memory.

If you have a problem with Connector/ODBC (MyODBC), please try to generate a trace file and send it with your report. See How to Report Connector/ODBC Problems or Bugs.

If your report includes long query output lines from test cases that you run with the `mysql` command-line tool, you can make the output more readable by using the `--vertical` option or the `\G` statement terminator. The `EXPLAIN SELECT` example later in this section demonstrates the use of `\G`.

Please include the following information in your report:

- The version number of the MySQL distribution you are using (for example, MySQL 5.7.10). You can find out which version you are running by executing `mysqladmin version`. The `mysqladmin` program can be found in the `bin` directory under your MySQL installation directory.
- The manufacturer and model of the machine on which you experience the problem.
- The operating system name and version. If you work with Windows, you can usually get the name and version number by double-clicking your My Computer icon and pulling down the “Help/About Windows” menu. For most Unix-like operating systems, you can get this information by executing the command `uname -a`.
- Sometimes the amount of memory (real and virtual) is relevant. If in doubt, include these values.
- The contents of the `docs/INFO_BIN` file from your MySQL installation. This file contains information about how MySQL was configured and compiled.
- If you are using a source distribution of the MySQL software, include the name and version number of the compiler that you used. If you have a binary distribution, include the distribution name.
- If the problem occurs during compilation, include the exact error messages and also a few lines of context around the offending code in the file where the error occurs.
- If `mysqld` died, you should also report the statement that crashed `mysqld`. You can usually get this information by running `mysqld` with query logging enabled, and then looking in the log after `mysqld` crashes. See Section 29.5, “Debugging and Porting MySQL”.
- If a database table is related to the problem, include the output from the `SHOW CREATE TABLE db_name.tbl_name` statement in the bug report. This is a very easy way to get the definition of any table in a database. The information helps us create a situation matching the one that you have experienced.
- The SQL mode in effect when the problem occurred can be significant, so please report the value of the `sql_mode` system variable. For stored procedure, stored function, and trigger objects, the relevant `sql_mode` value is the one in effect when the object was created. For a stored procedure or function, the `SHOW CREATE PROCEDURE` or `SHOW CREATE FUNCTION` statement shows the relevant SQL mode, or you can query `INFORMATION_SCHEMA` for the information:

  ```
  SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE
  FROM INFORMATION_SCHEMA.ROUTINES;
  ```

  For triggers, you can use this statement:

  ```
  SELECT EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, TRIGGER_NAME, SQL_MODE
  FROM INFORMATION_SCHEMA.TRIGGERS;
  ```

- For performance-related bugs or problems with `SELECT` statements, you should always include the output of `EXPLAIN SELECT ...`, and at least the number of rows that the `SELECT` statement produces. You should also include the output from `SHOW CREATE TABLE tbl_name` for each table that is involved. The more information you provide about your situation, the more likely it is that someone can help you.

  The following is an example of a very good bug report. The statements are run using the `mysql` command-line tool. Note the use of the `\G` statement terminator for statements that would otherwise provide very long output lines that are difficult to read.

```
mysql> SHOW VARIABLES;
mysql> SHOW COLUMNS FROM ...\G
       <output from SHOW COLUMNS>
mysql> EXPLAIN SELECT ...\G
       <output from EXPLAIN>
mysql> FLUSH STATUS;
mysql> SELECT ...;
       <A short version of the output from SELECT,
       including the time taken to run the query>
mysql> SHOW STATUS;
       <output from SHOW STATUS>
```

- If a bug or problem occurs while running `mysqld`, try to provide an input script that reproduces the anomaly. This script should include any necessary source files. The more closely the script can reproduce your situation, the better. If you can make a reproducible test case, you should upload it to be attached to the bug report.

  If you cannot provide a script, you should at least include the output from `mysqladmin variables extended-status processlist` in your report to provide some information on how your system is performing.

- If you cannot produce a test case with only a few rows, or if the test table is too big to be included in the bug report (more than 10 rows), you should dump your tables using `mysqldump` and create a `README` file that describes your problem. Create a compressed archive of your files using `tar` and `gzip` or `zip`. After you initiate a bug report for our bugs database at http://bugs.mysql.com/, click the Files tab in the bug report for instructions on uploading the archive to the bugs database.

- If you believe that the MySQL server produces a strange result from a statement, include not only the result, but also your opinion of what the result should be, and an explanation describing the basis for your opinion.

- When you provide an example of the problem, it is better to use the table names, variable names, and so forth that exist in your actual situation than to come up with new names. The problem could be related to the name of a table or variable. These cases are rare, perhaps, but it is better to be safe than sorry. After all, it should be easier for you to provide an example that uses your actual situation, and it is by all means better for us. If you have data that you do not want to be visible to others in the bug report, you can upload it using the Files tab as previously described. If the information is really top secret and you do not want to show it even to us, go ahead and provide an example using other names, but please regard this as the last choice.

- Include all the options given to the relevant programs, if possible. For example, indicate the options that you use when you start the `mysqld` server, as well as the options that you use to run any MySQL client programs. The options to programs such as `mysqld` and `mysql`, and to the `configure` script, are often key to resolving problems and are very relevant. It is never a bad idea to include them. If your problem involves a program written in a language such as Perl or PHP, please include the language processor's version number, as well as the version for any modules that the program uses. For example, if you have a Perl script that uses the `DBI` and `DBD::mysql` modules, include the version numbers for Perl, `DBI`, and `DBD::mysql`.

- If your question is related to the privilege system, please include the output of `mysqladmin reload`, and all the error messages you get when trying to connect. When you test your privileges, you should execute `mysqladmin reload version` and try to connect with the program that gives you trouble.

- If you have a patch for a bug, do include it. But do not assume that the patch is all we need, or that we can use it, if you do not provide some necessary information such as test cases showing the bug that your patch fixes. We might find problems with your patch or we might not understand it at all. If so, we cannot use it.

If we cannot verify the exact purpose of the patch, we will not use it. Test cases help us here. Show that the patch handles all the situations that may occur. If we find a borderline case (even a rare one) where the patch will not work, it may be useless.

- Guesses about what the bug is, why it occurs, or what it depends on are usually wrong. Even the MySQL team cannot guess such things without first using a debugger to determine the real cause of a bug.
- Indicate in your bug report that you have checked the reference manual and mail archive so that others know you have tried to solve the problem yourself.
- If your data appears corrupt or you get errors when you access a particular table, first check your tables with `CHECK TABLE`. If that statement reports any errors:
  - The `InnoDB` crash recovery mechanism handles cleanup when the server is restarted after being killed, so in typical operation there is no need to “repair” tables. If you encounter an error with `InnoDB` tables, restart the server and see whether the problem persists, or whether the error affected only cached data in memory. If data is corrupted on disk, consider restarting with the `innodb_force_recovery` option enabled so that you can dump the affected tables.
  - For non-transactional tables, try to repair them with `REPAIR TABLE` or with `myisamchk`. See Chapter 5, *MySQL Server Administration*.

  If you are running Windows, please verify the value of `lower_case_table_names` using the `SHOW VARIABLES LIKE 'lower_case_table_names'` statement. This variable affects how the server handles lettercase of database and table names. Its effect for a given value should be as described in Section 9.2.3, “Identifier Case Sensitivity”.
- If you often get corrupted tables, you should try to find out when and why this happens. In this case, the error log in the MySQL data directory may contain some information about what happened. (This is the file with the `.err` suffix in the name.) See Section 5.4.2, “The Error Log”. Please include any relevant information from this file in your bug report. Normally `mysqld` should *never* crash a table if nothing killed it in the middle of an update. If you can find the cause of `mysqld` dying, it is much easier for us to provide you with a fix for the problem. See Section B.4.1, “How to Determine What Is Causing a Problem”.
- If possible, download and install the most recent version of MySQL Server and check whether it solves your problem. All versions of the MySQL software are thoroughly tested and should work without problems. We believe in making everything as backward-compatible as possible, and you should be able to switch MySQL versions without difficulty. See Section 2.1.1, “Which MySQL Version and Distribution to Install”.

1.8 MySQL Standards Compliance

This section describes how MySQL relates to the ANSI/ISO SQL standards. MySQL Server has many extensions to the SQL standard, and here you can find out what they are and how to use them. You can also find information about functionality missing from MySQL Server, and how to work around some of the differences.

The SQL standard has been evolving since 1986 and several versions exist. In this manual, “SQL-92” refers to the standard released in 1992. “SQL:1999”, “SQL:2003”, “SQL:2008”, and “SQL:2011” refer to the versions of the standard released in the corresponding years, with the last being the most recent version. We use the phrase “the SQL standard” or “standard SQL” to mean the current version of the SQL Standard at any time.

One of our main goals with the product is to continue to work toward compliance with the SQL standard, but without sacrificing speed or reliability. We are not afraid to add extensions to SQL or support for non-SQL features if this greatly increases the usability of MySQL Server for a large segment of our user base. The `HANDLER` interface is an example of this strategy. See Section 13.2.4, “HANDLER Syntax”.

We continue to support transactional and nontransactional databases to satisfy both mission-critical 24/7 usage and heavy Web or logging usage.

MySQL Server was originally designed to work with medium-sized databases (10-100 million rows, or about 100MB per table) on small computer systems. Today MySQL Server handles terabyte-sized databases.

We are not targeting real-time support, although MySQL replication capabilities offer significant functionality.

MySQL supports ODBC levels 0 to 3.51.

MySQL supports high-availability database clustering using the `NDBCLUSTER` storage engine. See Chapter 22, *MySQL NDB Cluster 8.0*.

We implement XML functionality which supports most of the W3C XPath standard. See Section 12.11, “XML Functions”.

MySQL supports a native JSON data type as defined by RFC 7159, and based on the ECMAScript standard (ECMA-262). See Section 11.6, “The JSON Data Type”. MySQL also implements a subset of the SQL/JSON functions specified by a pre-publication draft of the SQL:2016 standard; see Section 12.17, “JSON Functions”, for more information.

Selecting SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients, depending on the value of the `sql_mode` system variable. DBAs can set the global SQL mode to match site server operating requirements, and each application can set its session SQL mode to its own requirements.

Modes affect the SQL syntax MySQL supports and the data validation checks it performs. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

For more information on setting the SQL mode, see Section 5.1.11, “Server SQL Modes”.

Running MySQL in ANSI Mode

To run MySQL Server in ANSI mode, start `mysqld` with the `--ansi` option. Running the server in ANSI mode is the same as starting it with the following options:

```
--transaction-isolation=SERIALIZABLE --sql-mode=ANSI
```

To achieve the same effect at runtime, execute these two statements:

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET GLOBAL sql_mode = 'ANSI';
```

You can see that setting the `sql_mode` system variable to `'ANSI'` enables all SQL mode options that are relevant for ANSI mode as follows:

```
mysql> SET GLOBAL sql_mode='ANSI';
```

```
mysql> SELECT @@GLOBAL.sql_mode;
        -> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,IGNORE_SPACE,ANSI'
```

Running the server in ANSI mode with `--ansi` is not quite the same as setting the SQL mode to `'ANSI'` because the `--ansi` option also sets the transaction isolation level.

See Section 5.1.7, “Server Command Options”.

1.8.1 MySQL Extensions to Standard SQL

MySQL Server supports some extensions that you probably will not find in other SQL DBMSs. Be warned that if you use them, your code will not be portable to other SQL servers. In some cases, you can write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the `!` character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `KEY_BLOCK_SIZE` clause in the following comment is executed only by servers from MySQL 5.1.10 or higher:

```
CREATE TABLE t1(a INT, KEY (a)) /*!50110 KEY_BLOCK_SIZE=1024 */;
```

The following descriptions list MySQL extensions, organized by category.

- Organization of data on disk

  MySQL Server maps each database to a directory under the MySQL data directory, and maps tables within a database to file names in the database directory. Consequently, database and table names are case-sensitive in MySQL Server on operating systems that have case-sensitive file names (such as most Unix systems). See Section 9.2.3, “Identifier Case Sensitivity”.

- General language syntax

  - By default, strings can be enclosed by `"` as well as `'`. If the `ANSI_QUOTES` SQL mode is enabled, strings can be enclosed only by `'` and the server interprets strings enclosed by `"` as identifiers.
  - `\` is the escape character in strings.
  - In SQL statements, you can access tables from different databases with the *`db_name.tbl_name`* syntax. Some SQL servers provide the same functionality but call this `User space`. MySQL Server doesn't support tablespaces such as used in statements like this: `CREATE TABLE ralph.my_table ... IN my_tablespace`.

- SQL statement syntax

  - The `ANALYZE TABLE`, `CHECK TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.
  - The `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE` statements. See Section 13.1.12, “CREATE DATABASE Syntax”, Section 13.1.24, “DROP DATABASE Syntax”, and Section 13.1.2, “ALTER DATABASE Syntax”.

- The `DO` statement.
- `EXPLAIN SELECT` to obtain a description of how tables are processed by the query optimizer.
- The `FLUSH` and `RESET` statements.
- The `SET` statement. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.
- The `SHOW` statement. See Section 13.7.7, “SHOW Syntax”. The information produced by many of the MySQL-specific `SHOW` statements can be obtained in more standard fashion by using `SELECT` to query `INFORMATION_SCHEMA`. See Chapter 25, *INFORMATION_SCHEMA Tables*.
- Use of `LOAD DATA`. In many cases, this syntax is compatible with Oracle `LOAD DATA`. See Section 13.2.7, “LOAD DATA Syntax”.
- Use of `RENAME TABLE`. See Section 13.1.36, “RENAME TABLE Syntax”.
- Use of `REPLACE` instead of `DELETE` plus `INSERT`. See Section 13.2.9, “REPLACE Syntax”.
- Use of `CHANGE col_name`, `DROP col_name`, or `DROP INDEX`, `IGNORE` or `RENAME` in `ALTER TABLE` statements. Use of multiple `ADD`, `ALTER`, `DROP`, or `CHANGE` clauses in an `ALTER TABLE` statement. See Section 13.1.9, “ALTER TABLE Syntax”.
- Use of index names, indexes on a prefix of a column, and use of `INDEX` or `KEY` in `CREATE TABLE` statements. See Section 13.1.20, “CREATE TABLE Syntax”.
- Use of `TEMPORARY` or `IF NOT EXISTS` with `CREATE TABLE`.
- Use of `IF EXISTS` with `DROP TABLE` and `DROP DATABASE`.
- The capability of dropping multiple tables with a single `DROP TABLE` statement.
- The `ORDER BY` and `LIMIT` clauses of the `UPDATE` and `DELETE` statements.
- `INSERT INTO tbl_name SET col_name = ...` syntax.
- The `DELAYED` clause of the `INSERT` and `REPLACE` statements.
- The `LOW_PRIORITY` clause of the `INSERT`, `REPLACE`, `DELETE`, and `UPDATE` statements.
- Use of `INTO OUTFILE` or `INTO DUMPFILE` in `SELECT` statements. See Section 13.2.10, “SELECT Syntax”.
- Options such as `STRAIGHT_JOIN` or `SQL_SMALL_RESULT` in `SELECT` statements.
- You don't need to name all selected columns in the `GROUP BY` clause. This gives better performance for some very specific, but quite normal queries. See Section 12.20, “Aggregate (GROUP BY) Functions”.
- You can specify `ASC` and `DESC` with `GROUP BY`, not just with `ORDER BY`.
- The ability to set variables in a statement with the `:=` assignment operator. See Section 9.4, “User-Defined Variables”.

- Data types
  - The `MEDIUMINT`, `SET`, and `ENUM` data types, and the various `BLOB` and `TEXT` data types.

  - The `AUTO_INCREMENT`, `BINARY`, `NULL`, `UNSIGNED`, and `ZEROFILL` data type attributes.

- Functions and operators

  - To make it easier for users who migrate from other SQL environments, MySQL Server supports aliases for many functions. For example, all string functions support both standard SQL syntax and ODBC syntax.

  - MySQL Server understands the `||` and `&&` operators to mean logical OR and AND, as in the C programming language. In MySQL Server, `||` and `OR` are synonyms, as are `&&` and `AND`. Because of this nice syntax, MySQL Server doesn't support the standard SQL `||` operator for string concatenation; use `CONCAT()` instead. Because `CONCAT()` takes any number of arguments, it is easy to convert use of the `||` operator to MySQL Server.

  - Use of `COUNT(DISTINCT value_list)` where *`value_list`* has more than one element.

  - String comparisons are case insensitive by default, with sort ordering determined by the collation of the current character set, which is `utf8mb4` by default. To perform case-sensitive comparisons instead, you should declare your columns with the `BINARY` attribute or use the `BINARY` cast, which causes comparisons to be done using the underlying character code values rather than a lexical ordering.

  - The `%` operator is a synonym for `MOD()`. That is, *`N % M`* is equivalent to `MOD(N,M)`. `%` is supported for C programmers and for compatibility with PostgreSQL.

  - The `=`, `<>`, `<=`, `<`, `>=`, `>`, `<<`, `>>`, `<=>`, `AND`, `OR`, or `LIKE` operators may be used in expressions in the output column list (to the left of the `FROM`) in `SELECT` statements. For example:

    ```
    mysql> SELECT col1=1 AND col2=2 FROM my_table;
    ```

  - The `LAST_INSERT_ID()` function returns the most recent `AUTO_INCREMENT` value. See Section 12.15, “Information Functions”.

  - `LIKE` is permitted on numeric values.

  - The `REGEXP` and `NOT REGEXP` extended regular expression operators.

  - `CONCAT()` or `CHAR()` with one argument or more than two arguments. (In MySQL Server, these functions can take a variable number of arguments.)

  - The `BIT_COUNT()`, `CASE`, `ELT()`, `FROM_DAYS()`, `FORMAT()`, `IF()`, `MD5()`, `PERIOD_ADD()`, `PERIOD_DIFF()`, `TO_DAYS()`, and `WEEKDAY()` functions.

  - Use of `TRIM()` to trim substrings. Standard SQL supports removal of single characters only.

  - The `GROUP BY` functions `STD()`, `BIT_OR()`, `BIT_AND()`, `BIT_XOR()`, and `GROUP_CONCAT()`. See Section 12.20, “Aggregate (GROUP BY) Functions”.

1.8.2 MySQL Differences from Standard SQL

We try to make MySQL Server follow the ANSI SQL standard and the ODBC SQL standard, but MySQL Server performs operations differently in some cases:

- There are several differences between the MySQL and standard SQL privilege systems. For example, in MySQL, privileges for a table are not automatically revoked when you delete a table. You must explicitly

issue a REVOKE statement to revoke privileges for a table. For more information, see Section 13.7.1.8, “REVOKE Syntax”.

- The CAST() function does not support cast to REAL or BIGINT. See Section 12.10, “Cast Functions and Operators”.

1.8.2.1 SELECT INTO TABLE Differences

MySQL Server doesn't support the SELECT ... INTO TABLE Sybase SQL extension. Instead, MySQL Server supports the INSERT INTO ... SELECT standard SQL syntax, which is basically the same thing. See Section 13.2.6.1, “INSERT ... SELECT Syntax”. For example:

```
INSERT INTO tbl_temp2 (fld_id)
    SELECT tbl_temp1.fld_order_id
    FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

Alternatively, you can use SELECT ... INTO OUTFILE or CREATE TABLE ... SELECT.

You can use SELECT ... INTO with user-defined variables. The same syntax can also be used inside stored routines using cursors and local variables. See Section 13.2.10.1, “SELECT ... INTO Syntax”.

1.8.2.2 UPDATE Differences

If you access a column from the table to be updated in an expression, UPDATE uses the current value of the column. The second assignment in the following statement sets col2 to the current (updated) col1 value, not the original col1 value. The result is that col1 and col2 have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

1.8.2.3 Foreign Key Differences

The MySQL implementation of foreign keys differs from the SQL standard in the following key respects:

- If there are several rows in the parent table that have the same referenced key value, InnoDB acts in foreign key checks as if the other parent rows with the same key value do not exist. For example, if you have defined a RESTRICT type constraint, and there is a child row with several parent rows, InnoDB does not permit the deletion of any of those parent rows.

  InnoDB performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.

- A FOREIGN KEY constraint that references a non-UNIQUE key is not standard SQL but rather an InnoDB extension.

- If ON UPDATE CASCADE or ON UPDATE SET NULL recurses to update the *same table* it has previously updated during the same cascade, it acts like RESTRICT. This means that you cannot use self-referential ON UPDATE CASCADE or ON UPDATE SET NULL operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential ON DELETE SET NULL, on the other hand, is possible, as is a self-referential ON DELETE CASCADE. Cascading operations may not be nested more than 15 levels deep.

- In an SQL statement that inserts, deletes, or updates many rows, foreign key constraints (like unique constraints) are checked row-by-row. When performing foreign key checks, InnoDB sets shared row-level locks on child or parent records that it must examine. MySQL checks foreign key constraints immediately; the check is not deferred to transaction commit. According to the SQL standard, the

default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. This means that it is not possible to delete a row that refers to itself using a foreign key.

For information about how the `InnoDB` storage engine handles foreign keys, see Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”.

1.8.2.4 '--' as the Start of a Comment

Standard SQL uses the C syntax `/* this is a comment */` for comments, and MySQL Server supports this syntax as well. MySQL also support extensions to this syntax that enable MySQL-specific SQL to be embedded in the comment, as described in Section 9.6, “Comment Syntax”.

Standard SQL uses “`--`” as a start-comment sequence. MySQL Server uses `#` as the start comment character. MySQL Server also supports a variant of the `--` comment style. That is, the `--` start-comment sequence must be followed by a space (or by a control character such as a newline). The space is required to prevent problems with automatically generated SQL queries that use constructs such as the following, where we automatically insert the value of the payment for `payment`:

```
UPDATE account SET credit=credit-payment
```

Consider about what happens if `payment` has a negative value such as `-1`:

```
UPDATE account SET credit=credit--1
```

`credit--1` is a valid expression in SQL, but `--` is interpreted as the start of a comment, part of the expression is discarded. The result is a statement that has a completely different meaning than intended:

```
UPDATE account SET credit=credit
```

The statement produces no change in value at all. This illustrates that permitting comments to start with `--` can have serious consequences.

Using our implementation requires a space following the `--` for it to be recognized as a start-comment sequence in MySQL Server. Therefore, `credit--1` is safe to use.

Another safe feature is that the `mysql` command-line client ignores lines that start with `--`.

1.8.3 How MySQL Deals with Constraints

MySQL enables you to work both with transactional tables that permit rollback and with nontransactional tables that do not. Because of this, constraint handling is a bit different in MySQL than in other DBMSs. We must handle the case when you have inserted or updated a lot of rows in a nontransactional table for which changes cannot be rolled back when an error occurs.

The basic philosophy is that MySQL Server tries to produce an error for anything that it can detect while parsing a statement to be executed, and tries to recover from any errors that occur while executing the statement. We do this in most cases, but not yet for all.

The options MySQL has when an error occurs are to stop the statement in the middle or to recover as well as possible from the problem and continue. By default, the server follows the latter course. This means, for example, that the server may coerce invalid values to the closest valid values.

Several SQL mode options are available to provide greater control over handling of bad data values and whether to continue statement execution or abort when errors occur. Using these options, you can

configure MySQL Server to act in a more traditional fashion that is like other DBMSs that reject improper input. The SQL mode can be set globally at server startup to affect all clients. Individual clients can set the SQL mode at runtime, which enables each client to select the behavior most appropriate for its requirements. See Section 5.1.11, “Server SQL Modes”.

The following sections describe how MySQL Server handles different types of constraints.

1.8.3.1 PRIMARY KEY and UNIQUE Index Constraints

Normally, errors occur for data-change statements (such as `INSERT` or `UPDATE`) that would violate primary-key, unique-key, or foreign-key constraints. If you are using a transactional storage engine such as `InnoDB`, MySQL automatically rolls back the statement. If you are using a nontransactional storage engine, MySQL stops processing the statement at the row for which the error occurred and leaves any remaining rows unprocessed.

MySQL supports an `IGNORE` keyword for `INSERT`, `UPDATE`, and so forth. If you use it, MySQL ignores primary-key or unique-key violations and continues processing with the next row. See the section for the statement that you are using (Section 13.2.6, “INSERT Syntax”, Section 13.2.12, “UPDATE Syntax”, and so forth).

You can get information about the number of rows actually inserted or updated with the `mysql_info()` C API function. You can also use the `SHOW WARNINGS` statement. See Section 28.7.7.36, “mysql_info()”, and Section 13.7.7.40, “SHOW WARNINGS Syntax”.

`InnoDB` and `NDB` tables support foreign keys. See Section 1.8.3.2, “FOREIGN KEY Constraints”.

1.8.3.2 FOREIGN KEY Constraints

Foreign keys let you cross-reference related data across tables, and foreign key constraints help keep this spread-out data consistent.

MySQL supports `ON UPDATE` and `ON DELETE` foreign key references in `CREATE TABLE` and `ALTER TABLE` statements. The available referential actions are `RESTRICT`, `CASCADE`, `SET NULL`, and `NO ACTION` (the default).

`SET DEFAULT` is also supported by the MySQL Server but is currently rejected as invalid by `InnoDB`. Since MySQL does not support deferred constraint checking, `NO ACTION` is treated as `RESTRICT`. For the exact syntax supported by MySQL for foreign keys, see Section 13.1.20.6, “Using FOREIGN KEY Constraints”.

`MATCH FULL`, `MATCH PARTIAL`, and `MATCH SIMPLE` are allowed, but their use should be avoided, as they cause the MySQL Server to ignore any `ON DELETE` or `ON UPDATE` clause used in the same statement. `MATCH` options do not have any other effect in MySQL, which in effect enforces `MATCH SIMPLE` semantics full-time.

MySQL requires that foreign key columns be indexed; if you create a table with a foreign key constraint but no index on a given column, an index is created.

You can obtain information about foreign keys from the `INFORMATION_SCHEMA.KEY_COLUMN_USAGE` table. An example of a query against this table is shown here:

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME
     > FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
     > WHERE REFERENCED_TABLE_SCHEMA IS NOT NULL;
+-------------+---------------+-------------+-----------------+
| TABLE_SCHEMA | TABLE_NAME    | COLUMN_NAME | CONSTRAINT_NAME |
+-------------+---------------+-------------+-----------------+
```

```
| fk1          | myuser        | myuser_id   | f               |
| fk1          | product_order | customer_id | f2              |
| fk1          | product_order | product_id  | f1              |
+--------------+---------------+-------------+-----------------+
3 rows in set (0.01 sec)
```

Information about foreign keys on `InnoDB` tables can also be found in the `INNODB_FOREIGN` and `INNODB_FOREIGN_COLS` tables, in the `INFORMATION_SCHEMA` database.

`InnoDB` and `NDB` tables support foreign keys. See Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”, for information specific to foreign key support in `InnoDB`.

1.8.3.3 Constraints on Invalid Data

By default, MySQL is forgiving of invalid or improper data values and coerces them to valid values for data entry. However, you can enable strict SQL mode to select more traditional treatment of bad values such that the server rejects them and aborts the statement in which they occur. See Section 5.1.11, “Server SQL Modes”.

This section describes the default (forgiving) behavior of MySQL, as well as the strict SQL mode and how it differs.

If you are not using strict mode, then whenever you insert an “incorrect” value into a column, such as a `NULL` into a `NOT NULL` column or a too-large numeric value into a numeric column, MySQL sets the column to the “best possible value” instead of producing an error: The following rules describe in more detail how this works:

- If you try to store an out of range value into a numeric column, MySQL Server instead stores zero, the smallest possible value, or the largest possible value, whichever is closest to the invalid value.
- For strings, MySQL stores either the empty string or as much of the string as can be stored in the column.
- If you try to store a string that does not start with a number into a numeric column, MySQL Server stores 0.
- Invalid values for `ENUM` and `SET` columns are handled as described in Section 1.8.3.4, “ENUM and SET Constraints”.
- MySQL permits you to store certain incorrect date values into `DATE` and `DATETIME` columns (such as `'2000-02-31'` or `'2000-02-00'`). In this case, when an application has not enabled strict SQL mode, it up to the application to validate the dates before storing them. If MySQL can store a date value and retrieve exactly the same value, MySQL stores it as given. If the date is totally wrong (outside the server's ability to store it), the special “zero” date value `'0000-00-00'` is stored in the column instead.
- If you try to store `NULL` into a column that doesn't take `NULL` values, an error occurs for single-row `INSERT` statements. For multiple-row `INSERT` statements or for `INSERT INTO ... SELECT` statements, MySQL Server stores the implicit default value for the column data type. In general, this is `0` for numeric types, the empty string (`''`) for string types, and the “zero” value for date and time types. Implicit default values are discussed in Section 11.7, “Data Type Default Values”.
- If an `INSERT` statement specifies no value for a column, MySQL inserts its default value if the column definition includes an explicit `DEFAULT` clause. If the definition has no such `DEFAULT` clause, MySQL inserts the implicit default value for the column data type.

The reason for using the preceding rules in nonstrict mode is that we can't check these conditions until the statement has begun executing. We can't just roll back if we encounter a problem after updating a few

rows, because the storage engine may not support rollback. The option of terminating the statement is not that good; in this case, the update would be “half done,” which is probably the worst possible scenario. In this case, it is better to “do the best you can” and then continue as if nothing happened.

You can select stricter treatment of input values by using the `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` SQL modes:

```
SET sql_mode = 'STRICT_TRANS_TABLES';
SET sql_mode = 'STRICT_ALL_TABLES';
```

`STRICT_TRANS_TABLES` enables strict mode for transactional storage engines, and also to some extent for nontransactional engines. It works like this:

- For transactional storage engines, bad data values occurring anywhere in a statement cause the statement to abort and roll back.
- For nontransactional storage engines, a statement aborts if the error occurs in the first row to be inserted or updated. (When the error occurs in the first row, the statement can be aborted to leave the table unchanged, just as for a transactional table.) Errors in rows after the first do not abort the statement, because the table has already been changed by the first row. Instead, bad data values are adjusted and result in warnings rather than errors. In other words, with `STRICT_TRANS_TABLES`, a wrong value causes MySQL to roll back all updates done so far, if that can be done without changing the table. But once the table has been changed, further errors result in adjustments and warnings.

For even stricter checking, enable `STRICT_ALL_TABLES`. This is the same as `STRICT_TRANS_TABLES` except that for nontransactional storage engines, errors abort the statement even for bad data in rows following the first row. This means that if an error occurs partway through a multiple-row insert or update for a nontransactional table, a partial update results. Earlier rows are inserted or updated, but those from the point of the error on are not. To avoid this for nontransactional tables, either use single-row statements or else use `STRICT_TRANS_TABLES` if conversion warnings rather than errors are acceptable. To avoid problems in the first place, do not use MySQL to check column content. It is safest (and often faster) to let the application ensure that it passes only valid values to the database.

With either of the strict mode options, you can cause errors to be treated as warnings by using `INSERT IGNORE` or `UPDATE IGNORE` rather than `INSERT` or `UPDATE` without `IGNORE`.

1.8.3.4 ENUM and SET Constraints

`ENUM` and `SET` columns provide an efficient way to define columns that can contain only a given set of values. See Section 11.4.4, “The ENUM Type”, and Section 11.4.5, “The SET Type”.

With strict mode enabled (see Section 5.1.11, “Server SQL Modes”), the definition of a `ENUM` or `SET` column acts as a constraint on values entered into the column. An error occurs for values that do not satisfy these conditions:

- An `ENUM` value must be one of those listed in the column definition, or the internal numeric equivalent thereof. The value cannot be the error value (that is, 0 or the empty string). For a column defined as `ENUM('a','b','c')`, values such as `''`, `'d'`, or `'ax'` are invalid and are rejected.
- A `SET` value must be the empty string or a value consisting only of the values listed in the column definition separated by commas. For a column defined as `SET('a','b','c')`, values such as `'d'` or `'a,b,c,d'` are invalid and are rejected.

Errors for invalid values can be suppressed in strict mode if you use `INSERT IGNORE` or `UPDATE IGNORE`. In this case, a warning is generated rather than an error. For `ENUM`, the value is inserted as the

error member (`0`). For `SET`, the value is inserted as given except that any invalid substrings are deleted. For example, `'a,x,b,y'` results in a value of `'a,b'`.

1.9 Credits

The following sections list developers, contributors, and supporters that have helped to make MySQL what it is today.

1.9.1 Contributors to MySQL

Although Oracle Corporation and/or its affiliates own all copyrights in the `MySQL server` and the `MySQL manual`, we wish to recognize those who have made contributions of one kind or another to the `MySQL distribution`. Contributors are listed here, in somewhat random order:

- Gianmassimo Vigazzola `<qwerg@mbox.vol.it>` or `<qwerg@tin.it>`

  The initial port to Win32/NT.

- Per Eric Olsson

  For constructive criticism and real testing of the dynamic record format.

- Irena Pancirov `<irena@mail.yacc.it>`

  Win32 port with Borland compiler. `mysqlshutdown.exe` and `mysqlwatch.exe`.

- David J. Hughes

  For the effort to make a shareware SQL database. At TcX, the predecessor of MySQL AB, we started with `mSQL`, but found that it couldn't satisfy our purposes so instead we wrote an SQL interface to our application builder Unireg. `mysqladmin` and `mysql` client are programs that were largely influenced by their `mSQL` counterparts. We have put a lot of effort into making the MySQL syntax a superset of `mSQL`. Many of the API's ideas are borrowed from `mSQL` to make it easy to port free `mSQL` programs to the MySQL API. The MySQL software doesn't contain any code from `mSQL`. Two files in the distribution (`client/insert_test.c` and `client/select_test.c`) are based on the corresponding (noncopyrighted) files in the `mSQL` distribution, but are modified as examples showing the changes necessary to convert code from `mSQL` to MySQL Server. (`mSQL` is copyrighted David J. Hughes.)

- Patrick Lynch

  For helping us acquire http://www.mysql.com/.

- Fred Lindberg

  For setting up qmail to handle the MySQL mailing list and for the incredible help we got in managing the MySQL mailing lists.

- Igor Romanenko `<igor@frog.kiev.ua>`

  `mysqldump` (previously `msqldump`, but ported and enhanced by Monty).

- Yuri Dario

  For keeping up and extending the MySQL OS/2 port.

- Tim Bunce

Author of `mysqlhotcopy`.

- Zarko Mocnik `<zarko.mocnik@dem.si>`

  Sorting for Slovenian language.

- "TAMITO" `<tommy@valley.ne.jp>`

  The `_MB` character set macros and the ujis and sjis character sets.

- Joshua Chamas `<joshua@chamas.com>`

  Base for concurrent insert, extended date syntax, debugging on NT, and answering on the MySQL mailing list.

- Yves Carlier `<Yves.Carlier@rug.ac.be>`

  `mysqlaccess`, a program to show the access rights for a user.

- Rhys Jones `<rhys@wales.com>` (And GWE Technologies Limited)

  For one of the early JDBC drivers.

- Dr Xiaokun Kelvin ZHU `<X.Zhu@brad.ac.uk>`

  Further development of one of the early JDBC drivers and other MySQL-related Java tools.

- James Cooper `<pixel@organic.com>`

  For setting up a searchable mailing list archive at his site.

- Rick Mehalick `<Rick_Mehalick@i-o.com>`

  For `xmysql`, a graphical X client for MySQL Server.

- Doug Sisk `<sisk@wix.com>`

  For providing RPM packages of MySQL for Red Hat Linux.

- Diemand Alexander V. `<axeld@vial.ethz.ch>`

  For providing RPM packages of MySQL for Red Hat Linux-Alpha.

- Antoni Pamies Olive `<toni@readysoft.es>`

  For providing RPM versions of a lot of MySQL clients for Intel and SPARC.

- Jay Bloodworth `<jay@pathways.sde.state.sc.us>`

  For providing RPM versions for MySQL 3.21.

- David Sacerdote `<davids@secnet.com>`

  Ideas for secure checking of DNS host names.

- Wei-Jou Chen `<jou@nematic.ieo.nctu.edu.tw>`

  Some support for Chinese(BIG5) characters.

- Wei He <hewei@mail.ied.ac.cn>

  A lot of functionality for the Chinese(GBK) character set.

- Jan Pazdziora <adelton@fi.muni.cz>

  Czech sorting order.

- Zeev Suraski <bourbon@netvision.net.il>

  `FROM_UNIXTIME()` time formatting, `ENCRYPT()` functions, and `bison` advisor. Active mailing list member.

- Luuk de Boer <luuk@wxs.nl>

  Ported (and extended) the benchmark suite to `DBI`/`DBD`. Have been of great help with `crash-me` and running benchmarks. Some new date functions. The `mysql_setpermission` script.

- Alexis Mikhailov <root@medinf.chuvashia.su>

  User-defined functions (UDFs); `CREATE FUNCTION` and `DROP FUNCTION`.

- Andreas F. Bobak <bobak@relog.ch>

  The `AGGREGATE` extension to user-defined functions.

- Ross Wakelin <R.Wakelin@march.co.uk>

  Help to set up InstallShield for MySQL-Win32.

- Jethro Wright III <jetman@li.net>

  The `libmysql.dll` library.

- James Pereria <jpereira@iafrica.com>

  Mysqlmanager, a Win32 GUI tool for administering MySQL Servers.

- Curt Sampson <cjs@portal.ca>

  Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.

- Martin Ramsch <m.ramsch@computer.org>

  Examples in the MySQL Tutorial.

- Steve Harvey

  For making `mysqlaccess` more secure.

- Konark IA-64 Centre of Persistent Systems Private Limited

  Help with the Win64 port of the MySQL server.

- Albert Chin-A-Young.

  Configure updates for Tru64, large file support and better TCP wrappers support.

- John Birrell

Emulation of `pthread_mutex()` for OS/2.

- Benjamin Pflugmann

  Extended `MERGE` tables to handle `INSERTS`. Active member on the MySQL mailing lists.

- Jocelyn Fournier

  Excellent spotting and reporting innumerable bugs (especially in the MySQL 4.1 subquery code).

- Marc Liyanage

  Maintaining the OS X packages and providing invaluable feedback on how to create OS X packages.

- Robert Rutherford

  Providing invaluable information and feedback about the QNX port.

- Previous developers of NDB Cluster

  Lots of people were involved in various ways summer students, master thesis students, employees. In total more than 100 people so too many to mention here. Notable name is Ataullah Dabaghi who up until 1999 contributed around a third of the code base. A special thanks also to developers of the AXE system which provided much of the architectural foundations for NDB Cluster with blocks, signals and crash tracing functionality. Also credit should be given to those who believed in the ideas enough to allocate of their budgets for its development from 1992 to present time.

- Google Inc.

  We wish to recognize Google Inc. for contributions to the MySQL distribution: Mark Callaghan's SMP Performance patches and other patches.

Other contributors, bugfinders, and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, `<jehamby@lightside>`, `<psmith@BayNetworks.com>`, `<duane@connect.com.au>`, Ted Deppner `<ted@psyber.com>`, Mike Simons, Jaakko Hyvatti.

And lots of bug report/patches from the folks on the mailing list.

A big tribute goes to those that help us answer questions on the MySQL mailing lists:

- Daniel Koch `<dkoch@amcity.com>`

  Irix setup.

- Luuk de Boer `<luuk@wxs.nl>`

  Benchmark questions.

- Tim Sailer `<tps@users.buoy.com>`

  `DBD::mysql` questions.

- Boyd Lynn Gerber `<gerberb@zenez.com>`

  SCO-related questions.

- Richard Mehalick `<RM186061@shellus.com>`

`xmysql`-related questions and basic installation questions.

- Zeev Suraski `<bourbon@netvision.net.il>`

  Apache module configuration questions (log & auth), PHP-related questions, SQL syntax-related questions and other general questions.

- Francesc Guasch `<frankie@citel.upc.es>`

  General questions.

- Jonathan J Smith `<jsmith@wtp.net>`

  Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might need some work.

- David Sklar `<sklar@student.net>`

  Using MySQL from PHP and Perl.

- Alistair MacDonald `<A.MacDonald@uel.ac.uk>`

  Is flexible and can handle Linux and perhaps HP-UX.

- John Lyon `<jlyon@imag.net>`

  Questions about installing MySQL on Linux systems, using either `.rpm` files or compiling from source.

- Lorvid Ltd. `<lorvid@WOLFENET.com>`

  Simple billing/license/support/copyright issues.

- Patrick Sherrill `<patrick@coconet.com>`

  ODBC and VisualC++ interface questions.

- Randy Harmon `<rjharmon@uptimecomputers.com>`

  `DBD`, Linux, some SQL syntax questions.

1.9.2 Documenters and translators

The following people have helped us with writing the MySQL documentation and translating the documentation or error messages in MySQL.

- Paul DuBois

  Ongoing help with making this manual correct and understandable. That includes rewriting Monty's and David's attempts at English into English as other people know it.

- Kim Aldale

  Helped to rewrite Monty's and David's early attempts at English into English.

- Michael J. Miller Jr. `<mke@terrapin.turbolift.com>`

  For the first MySQL manual. And a lot of spelling/language fixes for the FAQ (that turned into the MySQL manual a long time ago).

- Yan Cailin

  First translator of the MySQL Reference Manual into simplified Chinese in early 2000 on which the Big5 and HK coded versions were based.

- Jay Flaherty <fty@mediapulse.com>

  Big parts of the Perl DBI/DBD section in the manual.

- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>

  Proof-reading of the Reference Manual.

- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaire.fr>

  French error messages.

- Petr Snajdr, <snajdr@pvt.net>

  Czech error messages.

- Jaroslaw Lewandowski <jotel@itnet.com.pl>

  Polish error messages.

- Miguel Angel Fernandez Roiz

  Spanish error messages.

- Roy-Magne Mo <rmo@www.hivolda.no>

  Norwegian error messages and testing of MySQL 3.21.xx.

- Timur I. Bakeyev <root@timur.tatarstan.ru>

  Russian error messages.

- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>

  Italian error messages.

- Dirk Munzinger <dirk@trinity.saar.de>

  German error messages.

- Billik Stefan <billik@sun.uniag.sk>

  Slovak error messages.

- Stefan Saroiu <tzoompy@cs.washington.edu>

  Romanian error messages.

- Peter Feher

  Hungarian error messages.

- Roberto M. Serqueira

  Portuguese error messages.

- Carsten H. Pedersen

  Danish error messages.

- Arjen Lentz

  Dutch error messages, completing earlier partial translation (also work on consistency and spelling).

1.9.3 Packages that support MySQL

The following is a list of creators/maintainers of some of the most important API/packages/applications that a lot of people use with MySQL.

We cannot list every possible package here because the list would then be way to hard to maintain. For other packages, please refer to the software portal at http://solutions.mysql.com/software/.

- Tim Bunce, Alligator Descartes

  For the `DBD` (Perl) interface.

- Andreas Koenig `<a.koenig@mind.de>`

  For the Perl interface for MySQL Server.

- Jochen Wiedmann `<wiedmann@neckar-alb.de>`

  For maintaining the Perl `DBD::mysql` module.

- Eugene Chan `<eugene@acenet.com.sg>`

  For porting PHP for MySQL Server.

- Georg Richter

  MySQL 4.1 testing and bug hunting. New PHP 5.0 `mysqli` extension (API) for use with MySQL 4.1 and up.

- Giovanni Maruzzelli `<maruzz@matrice.it>`

  For porting iODBC (Unix ODBC).

- Xavier Leroy `<Xavier.Leroy@inria.fr>`

  The author of LinuxThreads (used by the MySQL Server on Linux).

1.9.4 Tools that were used to create MySQL

The following is a list of some of the tools we have used to create MySQL. We use this to express our thanks to those that has created them as without these we could not have made MySQL what it is today.

- Free Software Foundation

  From whom we got an excellent compiler (`gcc`), an excellent debugger (`gdb` and the `libc` library (from which we have borrowed `strto.c` to get some code working in Linux).

- Free Software Foundation & The XEmacs development team

  For a really great editor/environment.

- Julian Seward

  Author of `valgrind`, an excellent memory checker tool that has helped us find a lot of otherwise hard to find bugs in MySQL.

- Dorothea Lütkehaus and Andreas Zeller

  For `DDD` (The Data Display Debugger) which is an excellent graphical front end to `gdb`).

1.9.5 Supporters of MySQL

Although Oracle Corporation and/or its affiliates own all copyrights in the `MySQL server` and the `MySQL manual`, we wish to recognize the following companies, which helped us finance the development of the `MySQL server`, such as by paying us for developing a new feature or giving us hardware for development of the `MySQL server`.

- VA Linux / Andover.net

  Funded replication.

- NuSphere

  Editing of the MySQL manual.

- Stork Design studio

  The MySQL website in use between 1998-2000.

- Intel

  Contributed to development on Windows and Linux platforms.

- Compaq

  Contributed to Development on Linux/Alpha.

- SWSoft

  Development on the embedded `mysqld` version.

- FutureQuest

  The `--skip-show-database` option.

Chapter 2 Installing and Upgrading MySQL

Table of Contents

2.1 General Installation Guidance 78
2.1.1 Which MySQL Version and Distribution to Install 78
2.1.2 How to Get MySQL 79
2.1.3 Verifying Package Integrity Using MD5 Checksums or GnuPG 79
2.1.4 Installation Layouts 93
2.1.5 Compiler-Specific Build Characteristics 93
2.2 Installing MySQL on Unix/Linux Using Generic Binaries 93
2.3 Installing MySQL on Microsoft Windows 96
2.3.1 MySQL Installation Layout on Microsoft Windows 99
2.3.2 Choosing an Installation Package 100
2.3.3 MySQL Installer for Windows 101
2.3.4 Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive 127
2.3.5 Troubleshooting a Microsoft Windows MySQL Server Installation 136
2.3.6 Windows Postinstallation Procedures 137
2.3.7 Windows Platform Restrictions 140
2.4 Installing MySQL on macOS 141
2.4.1 General Notes on Installing MySQL on macOS 142
2.4.2 Installing MySQL on macOS Using Native Packages 143
2.4.3 Installing and Using the MySQL Launch Daemon 147
2.4.4 Installing and Using the MySQL Preference Pane 150
2.5 Installing MySQL on Linux 154
2.5.1 Installing MySQL on Linux Using the MySQL Yum Repository 155
2.5.2 Installing MySQL on Linux Using the MySQL APT Repository 160
2.5.3 Installing MySQL on Linux Using the MySQL SLES Repository 160
2.5.4 Installing MySQL on Linux Using RPM Packages from Oracle 160
2.5.5 Installing MySQL on Linux Using Debian Packages from Oracle 165
2.5.6 Deploying MySQL on Linux with Docker 167
2.5.7 Installing MySQL on Linux from the Native Software Repositories 177
2.5.8 Installing MySQL on Linux with Juju 181
2.5.9 Managing MySQL Server with systemd 181
2.6 Installing MySQL Using Unbreakable Linux Network (ULN) 186
2.7 Installing MySQL on Solaris 187
2.7.1 Installing MySQL on Solaris Using a Solaris PKG 188
2.8 Installing MySQL on FreeBSD 189
2.9 Installing MySQL from Source 190
2.9.1 Source Installation Methods 190
2.9.2 Source Installation Prerequisites 191
2.9.3 MySQL Layout for Source Installation 192
2.9.4 Installing MySQL Using a Standard Source Distribution 193
2.9.5 Installing MySQL Using a Development Source Tree 197
2.9.6 Configuring SSL Library Support 199
2.9.7 MySQL Source-Configuration Options 200
2.9.8 Dealing with Problems Compiling MySQL 225
2.9.9 MySQL Configuration and Third-Party Tools 227
2.9.10 Generating MySQL Doxygen Documentation Content 227
2.10 Postinstallation Setup and Testing 228
2.10.1 Initializing the Data Directory 229

2.10.2 Starting the Server 235
2.10.3 Testing the Server 238
2.10.4 Securing the Initial MySQL Account 240
2.10.5 Starting and Stopping MySQL Automatically 242
2.11 Upgrading MySQL 243
2.11.1 Before You Begin 243
2.11.2 Upgrade Paths 244
2.11.3 What the MySQL Upgrade Process Upgrades 245
2.11.4 Changes in MySQL 8.0 248
2.11.5 Preparing Your Installation for Upgrade 259
2.11.6 Upgrading MySQL Binary or Package-based Installations on Unix/Linux 262
2.11.7 Upgrading MySQL with the MySQL Yum Repository 267
2.11.8 Upgrading MySQL with the MySQL APT Repository 268
2.11.9 Upgrading MySQL with the MySQL SLES Repository 269
2.11.10 Upgrading MySQL on Windows 269
2.11.11 Upgrading a Docker Installation of MySQL 270
2.11.12 Upgrade Troubleshooting 271
2.11.13 Rebuilding or Repairing Tables or Indexes 271
2.11.14 Copying MySQL Databases to Another Machine 272
2.12 Downgrading MySQL 274
2.13 Perl Installation Notes 274
2.13.1 Installing Perl on Unix 274
2.13.2 Installing ActiveState Perl on Windows 275
2.13.3 Problems Using the Perl DBI/DBD Interface 276

This chapter describes how to obtain and install MySQL. A summary of the procedure follows and later sections provide the details. If you plan to upgrade an existing version of MySQL to a newer version rather than install MySQL for the first time, see Section 2.11, “Upgrading MySQL”, for information about upgrade procedures and about issues that you should consider before upgrading.

If you are interested in migrating to MySQL from another database system, see Section A.8, “MySQL 8.0 FAQ: Migration”, which contains answers to some common questions concerning migration issues.

Installation of MySQL generally follows the steps outlined here:

1. **Determine whether MySQL runs and is supported on your platform.**

   Please note that not all platforms are equally suitable for running MySQL, and that not all platforms on which MySQL is known to run are officially supported by Oracle Corporation. For information about those platforms that are officially supported, see https://www.mysql.com/support/supportedplatforms/database.html on the MySQL website.

2. **Choose which distribution to install.**

   Several versions of MySQL are available, and most are available in several distribution formats. You can choose from pre-packaged distributions containing binary (precompiled) programs or source code. When in doubt, use a binary distribution. Oracle also provides access to the MySQL source code for those who want to see recent developments and test new code. To determine which version and type of distribution you should use, see Section 2.1.1, “Which MySQL Version and Distribution to Install”.

3. **Download the distribution that you want to install.**

   For instructions, see Section 2.1.2, “How to Get MySQL”. To verify the integrity of the distribution, use the instructions in Section 2.1.3, “Verifying Package Integrity Using MD5 Checksums or GnuPG”.

4. **Install the distribution.**

   To install MySQL from a binary distribution, use the instructions in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

   To install MySQL from a source distribution or from the current development source tree, use the instructions in Section 2.9, “Installing MySQL from Source”.

5. **Perform any necessary postinstallation setup.**

   After installing MySQL, see Section 2.10, “Postinstallation Setup and Testing” for information about making sure the MySQL server is working properly. Also refer to the information provided in Section 2.10.4, “Securing the Initial MySQL Account”. This section describes how to secure the initial MySQL `root` user account, *which has no password* until you assign one. The section applies whether you install MySQL using a binary or source distribution.

6. If you want to run the MySQL benchmark scripts, Perl support for MySQL must be available. See Section 2.13, “Perl Installation Notes”.

Instructions for installing MySQL on different platforms and environments is available on a platform by platform basis:

- **Unix, Linux, FreeBSD**

  For instructions on installing MySQL on most Linux and Unix platforms using a generic binary (for example, a `.tar.gz` package), see Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

  For information on building MySQL entirely from the source code distributions or the source code repositories, see Section 2.9, “Installing MySQL from Source”

  For specific platform help on installation, configuration, and building from source see the corresponding platform section:

  - Linux, including notes on distribution specific methods, see Section 2.5, “Installing MySQL on Linux”.
  - IBM AIX, see Section 2.7, “Installing MySQL on Solaris”.
  - FreeBSD, see Section 2.8, “Installing MySQL on FreeBSD”.

- **Microsoft Windows**

  For instructions on installing MySQL on Microsoft Windows, using either the MySQL Installer or Zipped binary, see Section 2.3, “Installing MySQL on Microsoft Windows”.

  For information about managing MySQL instances, see MySQL Notifier Overview.

  For details and instructions on building MySQL from source code using Microsoft Visual Studio, see Section 2.9, “Installing MySQL from Source”.

- **OS X**

  For installation on OS X, including using both the binary package and native PKG formats, see Section 2.4, “Installing MySQL on macOS”.

  For information on making use of an OS X Launch Daemon to automatically start and stop MySQL, see Section 2.4.3, “Installing and Using the MySQL Launch Daemon”.

For information on the MySQL Preference Pane, see Section 2.4.4, “Installing and Using the MySQL Preference Pane”.

2.1 General Installation Guidance

The immediately following sections contain the information necessary to choose, download, and verify your distribution. The instructions in later sections of the chapter describe how to install the distribution that you choose. For binary distributions, see the instructions at Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries” or the corresponding section for your platform if available. To build MySQL from source, use the instructions in Section 2.9, “Installing MySQL from Source”.

2.1.1 Which MySQL Version and Distribution to Install

MySQL is available on a number of operating systems and platforms. For information about those platforms that are officially supported, see https://www.mysql.com/support/supportedplatforms/database.html on the MySQL website.

When preparing to install MySQL, decide which version and distribution format (binary or source) to use.

First, decide whether to install a development release or a General Availability (GA) release. Development releases have the newest features, but are not recommended for production use. GA releases, also called production or stable releases, are meant for production use. We recommend using the most recent GA release.

The naming scheme in MySQL 8.0 uses release names that consist of three numbers and an optional suffix (for example, **mysql-8.0.1-dmr**). The numbers within the release name are interpreted as follows:

- The first number (**8**) is the major version number.
- The second number (**0**) is the minor version number. Taken together, the major and minor numbers constitute the release series number. The series number describes the stable feature set.
- The third number (**1**) is the version number within the release series. This is incremented for each new bugfix release. In most cases, the most recent version within a series is the best choice.

Release names can also include a suffix to indicate the stability level of the release. Releases within a series progress through a set of suffixes to indicate how the stability level improves. The possible suffixes are:

- **dmr** indicates a development milestone release (DMR). MySQL development uses a milestone model, in which each milestone introduces a small subset of thoroughly tested features. From one milestone to the next, feature interfaces may change or features may even be removed, based on feedback provided by community members who try these earily releases. Features within milestone releases may be considered to be of pre-production quality.
- **rc** indicates a Release Candidate (RC). Release candidates are believed to be stable, having passed all of MySQL's internal testing. New features may still be introduced in RC releases, but the focus shifts to fixing bugs to stabilize features introduced earlier within the series.
- Absence of a suffix indicates a General Availability (GA) or Production release. GA releases are stable, having successfully passed through the earlier release stages, and are believed to be reliable, free of serious bugs, and suitable for use in production systems.

Development within a series begins with DMR releases, followed by RC releases, and finally reaches GA status releases.

After choosing which MySQL version to install, decide which distribution format to install for your operating system. For most use cases, a binary distribution is the right choice. Binary distributions are available in native format for many platforms, such as RPM packages for Linux or DMG packages for OS X. Distributions are also available in more generic formats such as Zip archives or compressed `tar` files. On Windows, you can use the MySQL Installer to install a binary distribution.

Under some circumstances, it may be preferable to install MySQL from a source distribution:

- You want to install MySQL at some explicit location. The standard binary distributions are ready to run at any installation location, but you might require even more flexibility to place MySQL components where you want.
- You want to configure `mysqld` with features that might not be included in the standard binary distributions. Here is a list of the most common extra options used to ensure feature availability:
  - `-DWITH_LIBWRAP=1` for TCP wrappers support.
  - `-DWITH_ZLIB={system|bundled}` for features that depend on compression
  - `-DWITH_DEBUG=1` for debugging support

  For additional information, see Section 2.9.7, “MySQL Source-Configuration Options”.
- You want to configure `mysqld` without some features that are included in the standard binary distributions.
- You want to read or modify the C and C++ code that makes up MySQL. For this purpose, obtain a source distribution.
- Source distributions contain more tests and examples than binary distributions.

2.1.2 How to Get MySQL

Check our downloads page at https://dev.mysql.com/downloads/ for information about the current version of MySQL and for downloading instructions. For a complete up-to-date list of MySQL download mirror sites, see https://dev.mysql.com/downloads/mirrors.html. You can also find information there about becoming a MySQL mirror site and how to report a bad or out-of-date mirror.

For RPM-based Linux platforms that use Yum as their package management system, MySQL can be installed using the MySQL Yum Repository. See Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details.

For Debian-based Linux platforms, MySQL can be installed using the MySQL APT Repository. See Section 2.5.2, “Installing MySQL on Linux Using the MySQL APT Repository” for details.

For SUSE Linux Enterprise Server (SLES) platforms, MySQL can be installed using the MySQL SLES Repository. See Section 2.5.3, “Installing MySQL on Linux Using the MySQL SLES Repository” for details.

To obtain the latest development source, see Section 2.9.5, “Installing MySQL Using a Development Source Tree”.

2.1.3 Verifying Package Integrity Using MD5 Checksums or GnuPG

After downloading the MySQL package that suits your needs and before attempting to install it, make sure that it is intact and has not been tampered with. There are three means of integrity checking:

- MD5 checksums
- Cryptographic signatures using GnuPG, the GNU Privacy Guard
- For RPM packages, the built-in RPM integrity verification mechanism

The following sections describe how to use these methods.

If you notice that the MD5 checksum or GPG signatures do not match, first try to download the respective package one more time, perhaps from another mirror site.

2.1.3.1 Verifying the MD5 Checksum

After you have downloaded a MySQL package, you should make sure that its MD5 checksum matches the one provided on the MySQL download pages. Each package has an individual checksum that you can verify against the package that you downloaded. The correct MD5 checksum is listed on the downloads page for each MySQL product, and you will compare it against the MD5 checksum of the file (product) that you download.

Each operating system and setup offers its own version of tools for checking the MD5 checksum. Typically the command is named `md5sum`, or it may be named `md5`, and some operating systems do not ship it at all. On Linux, it is part of the **GNU Text Utilities** package, which is available for a wide range of platforms. You can also download the source code from http://www.gnu.org/software/textutils/. If you have OpenSSL installed, you can use the command `openssl md5 package_name` instead. A Windows implementation of the `md5` command line utility is available from http://www.fourmilab.ch/md5/. `winMd5Sum` is a graphical MD5 checking tool that can be obtained from http://www.nullriver.com/index/products/winmd5sum. Our Microsoft Windows examples will assume the name `md5.exe`.

Linux and Microsoft Windows examples:

```
shell> md5sum mysql-standard-8.0.19-linux-i686.tar.gz
aaab65abbec64d5e907dcd41b8699945  mysql-standard-8.0.19-linux-i686.tar.gz
```

```
shell> md5.exe mysql-installer-community-8.0.19.msi
aaab65abbec64d5e907dcd41b8699945  mysql-installer-community-8.0.19.msi
```

You should verify that the resulting checksum (the string of hexadecimal digits) matches the one displayed on the download page immediately below the respective package.

Note

Make sure to verify the checksum of the *archive file* (for example, the `.zip`, `.tar.gz`, or `.msi` file) and not of the files that are contained inside of the archive. In other words, verify the file before extracting its contents.

2.1.3.2 Signature Checking Using GnuPG

Another method of verifying the integrity and authenticity of a package is to use cryptographic signatures. This is more reliable than using MD5 checksums, but requires more work.

We sign MySQL downloadable packages with GnuPG (GNU Privacy Guard). GnuPG is an Open Source alternative to the well-known Pretty Good Privacy (PGP) by Phil Zimmermann. Most Linux distributions ship with GnuPG installed by default. Otherwise, see http://www.gnupg.org/ for more information about GnuPG and how to obtain and install it.

To verify the signature for a specific package, you first need to obtain a copy of our public GPG build key, which you can download from http://pgp.mit.edu/. The key that you want to obtain is named `mysql-build@oss.oracle.com`. Alternatively, you can copy and paste the key directly from the following text:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQGiBD4+owwRBAC14GIfUfCyEDSIePvEW3SAFUdJBtoQHH/nJKZyQT7h9bPlUWC3
RODjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPkBdCk96+OmSLN9brZ
fw2vOUgCmYv2hW0hyDHuvYlQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqOxRznNCRCRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxwR9pRWVArNYJdDRT+rf2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQc7fJYLV
K2HtkrPbP72vwsEKMYhhr0eKCbtLGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTkamEnPCia2ZoOHODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNlYsafwAPEOMDKpMqAK6IyisNtPvaLd8lH0bPAnWqcyefep
rv0sxxqUEMcM3o7wwgfN83POkDasDbs3pjwPhxvhz6//62zQJ7Q2TXlTUUwgUmVs
ZWFzZSBFbmdpbmVlcmluZyA8bXlzcWwtYnVpbGRAb3NzLm9yYWNsZS5jb20+iGwE
ExECACwCGyMCHgECF4ACGQEGCwkIBwMCBhUKCQgCAwUWAgMBAAUCXEBY+wUJI87e
5AAKCRCMcY07UHLh9RZPAJ9uvm0zlzfCN+DHxHVaoFLFjdVYTQCfborsC9tmEZYa
whhogjeBkZkorbyIaQQTEQIAKQIbIwYLCQgHAwIEFQIIAwQWAgMBAh4BAheAAhkB
BQJTAdRmBQkaZsvLAAoJEIxxjTtQcuH1X4MAoKNLWAbCBUj96637kv6Xa/fJuX5m
AJwPtmgDfjUe2iuhXdTrFEPT19SB6ohmBBMRAgAmAhsjBgsJCAcDAgQVAggCDBBYC
AwECHgECF4AFAk53PioFCRP7AhUACgkQjHGNO1By4fUmzACeJdfqgc9gWTUhgmcM
AOmG4RjwuxcAoKfM+U8yMOGELi+TRif7MtKEms6piGkEExECACkCGyMGCwkIBwMC
BBUCCAMEFgIDAQIeAQIXgAIZAQUCUZSROgUJFTchqgAKCRCMcY07UHLh9YtAAJ9X
rA/ymlmozPZn+A9ls8/uwMcTsQCfaQMNq1dNkhH2kyByc3Rx9/W2xfqJARwEEAEC
AAYFAlAS6+UACgkQ8aIC+GoXHivrWwf/dtLk/x+NC2VMDlg+vOeM0qqG1IlhXZfi
NsEisvvGaz4m8fSFRGe+1bvvfDoKRhxiGXU48RusjixzvBb6KTMuY6JpOVfz9Dj3
H9spYriHa+i6rYySXZIpOhfLiMnTy7NH2OvYCyNzSS/ciIUACIfH/2NH8zNT5CNF
1uPNRs7HsHzzz7pOlTjtTWiF4cq/Ij6Z6CNrmdj+SiMvjYN9u6sdEKGtoNtpycgD
5HGKR+I7Nd/7v56yhaUe4FpuvsNXig86K9tI6MUFS8CUyy7Hj3kVBZOUWVBM053k
nGdALSygQr50DA3jMGKVl4ZnHje2RVWRmFTr5YWoRTMxUSQPMLpBNIkBHAQQAQIA
BgUCU1B+vQAKCRAohbcD0zcc8dWwCACWXXWDXIcAWRUw+j3ph8dr9u3SItljn3wB
c7clpclKWPuLvTz7lGgzlVB0s8hH4xgkSA+zLzl6u56mpUzskFl7f1I3Ac9GGpM4
0M5vmmR9hwlD1HdZtGfbD+wkjlqgitNLoRcGdRf/+U7x09GhSS7Bf339sunIX6sM
gXSC4L32D3zDjF5icGdb0kj+3lCrRmp853dGyA3ff9yUiBkxcKNawpi7Vz3D2ddU
pOF3BP+8NKPg4P2+srKgkFbd4HidcISQCt3rY4vaTkEkLKg0nNA6U4r0YgOa7wIT
SsxFlntMMzaRg53QtK0+YkH0KuZR3GY8B7pi+tlgycyVR7mIFo7riQEcBBABCAAG
BQJWgVd0AAoJEEZu4b/gk4UKk9MH/Rnt7EccPjSJC5CrB2AU5LY2Dsr+PePI2ubP
WsEdG82qSjjGpbhIH8LSg/PzQoGHiFWMmmZWJktRT+dcgLbs3b2VwCNAwCE8jOHd
UkQhEowgomdNvHiBHKHjP4/lF68KOPiO/2mxYYkmpM7BWf3kB57DJ5CTi3/JLoN7
zF40qIs/p09ePvnwStpglbbtUn7XPO+1/Ee8VHzimABom52PkQIuxNiVUzLVn3bS
Wqrd5ecuqLk6yzjPXd2XhDHWC9Twpl68GePru6EzQtusi0m6S/sHgEXqh/IxrFZV
JlljF75JvosZq5zeulr0i6kOij+Y1p6MFffihITZ1gTmk+CLvK2JASIEEAECAAwF
Ak53QS4FAwASdQAACgkQlxC4m8pXrXwJ8Qf/be/UO9mqfoc2sMyhwMpN4/fdBWwf
LkA12FXQDOQMvwH9HsmEjnfUgYKXschZRi+DuHXe1P7l8G2aQLubhBsQf9ejKvRF
TzuWMQkdIq+6Koulxv6ofkCcv3d1xtO2W7nb5yxcpVBPrRfGFGebJvZa58DymCNg
yGtAU6AOz4veavNmI2+GIDQsY66+tYDvZ+CxwzdYu+HDV9HmrJfc6deM0mnBn7SR
jqzxJPgoTQhihTav6q/R5/2p5NvQ/H84OgS6GjosfGc2duUDzCP/kheMRKfzuyKC
OHQPtJuIj8++gfpHtEU7IDUX1So3c9n0PdpeBvclsDbpRnCNxQWU4mBot4kBIgQQ
AQIADAUCToi2GQUDABJ1AAAKCRCXELibyletfLZAB/9oRqx+NC98UQD/wlxCRytz
vi/MuPnbgQUPLHEap10tvEi33S/H/xDR/tcGofY4cjAvo5skZXXeWq93Av7PACUb
zkg0X0eSr2oL6wy66xfov72AwSuX+iUK68qtKaLqRLitM02y8aNRV/ggKvt7UMvG
mOvs5yLaYlobyvGaFC2ClfkNOt2MlVnQZCmnYBCwOktPGkExiu2yZMifcYGxQcpH
KVFG59KeF2cM2d4xYM8HJqkSGGW306LFVSyeRwG+wbttgLpD5bM/T2b3fF/J35ra
CSMLZearRTq8aygPl+XM7MM2eR946aw6jmOsgNBErbvvIdQj6LudAZj+8imcXV2K
iQEiBBABAgAMBQJOmdnRBQMAEnUAAAoJEJcQuJvKV618AvIIAIEF1ZJ+Ry7WOdKF
5oeQ/ynaYUigzN92fW/9zB8yuQlngkFJGidYMbci1tR1siziIVJFusR3ZonqAPGK
/SUta9Y6KWLhmc7c5UnEHklq/NfdMZ2WVSIykXlctqw0sbb+z1ecEd4G8u9j5ill
MO1B36rQayYAPoeXLX8dY4VyFLVGaQ00rWQBYFZrpw16ATWbWGJP332NSfCk4zZq
6kXEW07q0st3YBgAAGdNQyEeZCa4d4pBRSX6189Kjg6GDnIcaiOF6HO6PLr9fRlL
r5ObCgU+G9gEhfiVwDEV9E+7/Bq2pYZ9whhkBqWQzdpXTNTM24uaEhE01EPO5zeC
O214q6mJASIEEAECAAwFAk6rpgEFAwASdQAACgkQlxC4m8pXrXzAhwf/f9O99z16
3Y5FZVIxexyqXQ/Mct9uKHuXEVnRFYbA49dQLD4S73N+zN7gn9jFeQcBo4w8qVUV
94U/ta/VbLkdtNREyplPM4XY8YE5Wfd9bfyg3q1PbEiVjk995sBF+2+To99YYKst
```

```
gXPqjlH0jUfEyDmexOj+hsp8Rc63kvkIx36VBa4ONRYFefGAhKDMigL2YAhc1UkG
tkGTuLmlCGwIV6lviDZD3RJf5375VFnaHv7eXfwQxCwE+BxG3CURrjfxjaxMTmMP
yAG2rhDp5oTUEvqDYNbko5UxYOmrSjvF4FzXwqerElXJUkUzSh0pp7RxHB/1lCxD
s7D1F1hlgFQuNIkBIgQQAQIADAUCTrzZHAUDABJ1AAAKCRCXELibyletfMUpB/4s
07dREULIBnA1D6qr3fHsQJNZqbAuyDlvgGGLWzoyEDs+1JMFFlaa+EeLIo1386GU
2DammDC23p3IB79uQhJeD2Z1TcVg4cA64SfF/CHca5coeRSrdAiudzU/cgLGtXIP
/OaFamXgdMxAhloLFbSHPCZkyb00phVa8+xeIVDrK1HByZsNIXy/SSK8U26S2PVZ
2o14fWvKbJ1Aga8N6DuWY/D8P2mi3RAbiuZgfzkmKL5idH/wSKfnFKdTgJzssdCc
1jZEGVk5rFYcWOrJARHeP/tsnb/UxKBEsNtO7e3N2e/rLVnEykVIO066hz7xZK/V
NBSpx3k3qj4XPK41IHy2iQEiBBABAgAMBQJOzqO8BQMAEnUAAAoJEJcQuJvKV618
2twH/0IzjXLxN45nvIfEjC75a+i9ZSLlqR8lsHL4GpEScFKI0a0lT4IVAIY2RKG+
MAs2eHm0UfKuwGs5jluRZ9RqKrc61sY0XQV9/7znY9Db16ghX04JjknOKs/fPi87
rvKkB/QxJWS8qbb/erRmW+cPNjbRxTFPS5JIwFWHA16ieFEpvdAgKV6nfvJVTq1r
jPDcnIA9CJN2SmUFx9Qx3SRc6ITbam1hjFnY6sCh6AUhxLI2f1mq1xH9PqEy42Um
68prRqTyJ7Iox1g/UDDkeeUcAg7T1viTz7uXpS3Wrq4zzo4yOpaJfLDR3pI5g2Zk
SNGTMo6aySE4OABt8i1Pc1Pm6AmJASIEEAECAAwFAk7yPFYFAwASdQAACgkQlxC4
m8pXrXzXiAf9FrXe0lgcPM+tYOWMLhv5gXJi2VUBaLxpyRXm/kJcmxInKq1GCd3y
D4/FLHNu3ZcCz/uklPAbZXWI0O6ewq0LWsRtklmJjWiedH+hGyaTv95VklojRIBd
8nBaJ6M98rljMBHTFwWvjQFVf4FLRJQZqHlvjcCkq2Dd9BWJpGXvr/gpKkmMJYNK
/ftfZRcChb35NI19WRpOhj9u808OPcqKVvZBcPwFGV5cEBzmAC94J7JcD8+S8Ik8
iUJMQGGL3QcmZOBozovh86hj7KTSEBHlLXl832z89H1hLeuLbnXoGLv3zeUFSxkv
1h35LhZLqIMDQRXLuUzxGHMBpLhPyGWRJ4kBIgQQAQIADAUCTwQJFwUDABJ1AAAK
CRCXELibyletfABvB/9Cy69cjOqLGywITs3Cpg//40jmdhSAVxilJivP6J5bubFH
DJlVTx541Dv5h4hTG2BQuueQ4q1VCpSGW+rHcdhPyvmZGRz1rxdQQGh1Dv0Bod2c
3PJVSYPSrRSwCZJkJHOtVRBdjK4mkZb5aFTza+Tor9kxzj4FcXVd4KAS+hHQHYHc
Ar8tt2eOLzqdEFTULeGiSoNn+PVzvzdfhndphK+8F2jfQ2UKuc01O7k0Yn9xZVx0
OG6fE1gStzLv7C5amWLRd8+xh+MN0G8MgNglpBoExsEMMlPBYSUHa6lxpdMNMuib
rIyVncE9X8QOhImt8K0sNn/EdbuldJNGYbDLt7O4iQEiBBABAgAMBQJPFdTcBQMA
EnUAAAoJEJcQuJvKV6184owH+wZ/uLpezXnSxigeH1sig72QEXMrNd5DVHCJdig3
bo+K5YmmN710/m5z+63XKUEWpd6/knajObgckThzWftNeK1SSFQGPmoYZP9EZnSU
7L+/dSUpExbj842G5LYagrCyMGtlxRywWEmbi72TKS/JOK0jLiOdvVy+PHrZSu0D
TVQ7cJh1BmPsbz7zzxjmcI5l+7B7K7RHZHq45nDLoIabwDacj7BXvBK0Ajqz4QyJ
GQUjXC7q+88I+ptPvOXlE5nI/NbiCJOMI6d/bWN1KwYrC80fZuFaznfQFcPyUaDw
yRaun+K3kEji2wXecq+yMmLUEp01TKsUeOL50HD6hHH07W+JASIEEAECAAwFAk85
bQsFAwASdQAACgkQlxC4m8pXrXwKPQgAlkbUsTr7nkq+haOk0jKpaHWEbRMEGMrB
I3F7E+RDO6V/8y4Jtn04EYDc8GgZMBah+mOgeINq3y8jRMYV5jVtZXv2MWYFUcjM
kVBKeqhi/pGEjmUdmdt3DlPv3Z+fMTMRmAocI981iY/go8PVPg/+nrR6cFK2xxnO
R8TacikJBFeSfkkORg1tDzjjYv1B5ZIEkpplepl5ahJBBq7cpYhTdY6Yk0Sz0J8w
EdffLSaNxrRuWLrRhWzZU7p9bFzfb/7OHc21dJnB7wKv5VvtgE+jiQw9tOKaf5hc
SgRYuF6heu+B25gc5Uu88lo409mZ7oxQ6hDCn7JHvzh0rhmSN+Kid4kBIgQQAQIA
DAUCT0qQrQUDABJ1AAAKCRCXELibyletfC9UB/4o2ggJYM0CLxEpP0GU8UKOh3+/
zm1DN7Qe4kY2iCtF1plKHQaTgt5FlgRCFaiXcVv7WzGz/FnmxonR1leLl+kfRlwy
PPnoI/AWPCy/NO4Cl5KnjsSmsdDUpObwZ4KYsdilZR7ViJu2swdAIgnXBUwrlRJR
7CK4TAKrTeonRgVSrVx8Vt//8/cYj73CLq8oY/KK0iHiQrSwo44uyhdiFIAssjyX
n6/2E+w0zgvPexNSNNROHQ8pjbq+NTY6GwKIGsaej3UTRwQ7psvKXz8y7xdzmOAr
/khGvxB5gjkx02pimjeia8v66aH6rbnojJMAovNUS4EHdHnulv4rovC8Kf9iiQEi
BBABAgAMBQJPVdsaBQMAEnUAAAoJEJcQuJvKV618vVEIALFXPBzcAO1SnQarBLzy
YMVZZumPvSXKnUHAO+6kjApXPJ+qFRdUaSNshZxVKY9Zryblu4ol/fLUTt0CliSD
IxD6L4GXEm4VYYCl4lPO3bVsJnGITLFwQGHM27EmjVoTiD8Ch7kPq2EXr3dMRgzj
pdz+6aHGSUfOdLTPXufDvW83bEWGaRVuTJKw+wIrcuRqQ+ucWJgJGwcE4zeHjZad
Jx1XUm1X+BbI73uiQussyjhhQVVNU7QEdrjyuscaZ/H38wjUwNbylxDPB4I8quC1
knQ0wSHr7gKpM+E9nhiS14poRqU18u78/sJ2MUPXnQA6533IC238/LP8JgqB+BiQ
BTSJASIEEAECAAwFAk9ng3cFAwASdQAACgkQlxC4m8pXrXxQRAf/UZlkkpFJj1om
9hIRz7gS+l7YvTaKSzpo+TBcx3C7aqKJpir6TlMK9cb9HGTHo2Xp1N3FtQL72NvO
6CcJpBURbvSyb4i0hrm/YcbUC4Y3eajWhkRS3iVfGNFbc/rHthViz0r6Y5lhXX16
aVkDv5CIFWaF3BiUK0FnHrZiy4FPacUXCwEjv3uf8MpxV5oEmo8Vs1h4TL3obyUz
qrImFrEMYE/12lkE8iR5KWCaF8eFyl56HL3PPl90JMQBXzhwsFoWCPuwjfM5w6sW
Ll//zynwxtlJ9CRz9c2vK6aJ8DRu3OfBKN1iiEcNEynksDnNXErn5xXKz3p5pYdq
e9BLzUQCDYkBIgQQAQIADAUCT3inRgUDABJ1AAAKCRCXELibyletfGMKCADJ97qk
geBntQ+tZtKSFyXznAugYQmbzJld8U6eGSQnQkM40Vd62UZLdA8MjlWKS8y4A4L2
0cI14zs5tKG9Q72BxQOw5xkxlLASw1/8WeYEbw7ZA+sPG//q9v3kIkru3sv64mMA
enZtxsykexRGyCumxLjzlAcL1drWJGUYE2Kl6uzQS7jb+3PNBloQvz6nb3YRZ+Cg
Ly9D41SIK+fpnV8r4iqhu7r4LmAQ7Q1DF9aoGaYvn2+xLGyWHxJAUet4xkMNOLp6
k9RF1nbNe4I/sqeCB25CZhCTEvHdjSGTD2yJR5jfoWkwO9w8DZG1Q9WrWqki4hSB
l0cmcvO34pC1SJYziQEiBBABAgAMBQJPinQFBQMAEnUAAAoJEJcQuJvKV618CFEI
AJp5BbcV7+JBMRSvkoUcAWDoJSP2ug9zGw5FB8J90PDefKWCKs5Tjayf2TvM5ntq
5DE9SGaXbloIwa74FoZlgqlhMZ4AtY9Br+oyPJ5S844wpAmWMFc6NnEPFaHQkQ+b
```

```
dJYpRVNd9lzagJP261P3S+S9T2UeHVdOJBgWIq9Mbs4lnZzWsnZfQ4Lsz0aPqe48
tkU8hw+nflby994qIwNOlk/u+I/lJbNz5zDY91oscXTRl2jV1qBgKYwwCXxyB3j9
fyVpRl+7QnqbTWcCICVFL+uuYpP0HjdoKNqhzEguAUQQLOB9msPTXfa2hG+32ZYg
5pzI5V7GCHq0KO6u5Ctj3TGJASIEEAECAAwFAk+cQEEFAwASdQAACgkQlxC4m8pX
rXzi7AgAx8wJzNdD7UlgdKmrAK//YqH7arSssb33Xf45sVHDpUVA454DXeBrZpi+
zEuo03o5BhAuf38cwfbkV6jN1mC2N0FZfpy4v7RxHKLYr7tr6r+DRn1L1giX5ybx
CgY0fLAxkwscWUKGKABWxkz9b/beEXaO2rMt+7DBUdpAOP5FNRQ8WLRWBcMGQiaT
S4YcNDAiNkrSP8CMLQP+04hQjahxwCgBnksylciqz3Y5/MreybNnTOrdjVDsF0Oe
t0uLOiWXUZV1FfaGIdb/oBQLg+e1B74p5+q3aF8YI97qAZpPa1qiQzWIDX8LX9QX
EFyZ3mvqzGrxkFoocXleNPgWT8fRuokBIgQQAQIADAUCT64N/QUDABJ1AAAKCRCX
ELibyletfDOGCACKfcjQlSxrWlEUrYYZpoBP7DE+YdlIGumt5l6vBmxmt/5OEhqr
+dWwuoiyC5tm9CvJbuZup8anWfFzTTJmPRPsmE4z7Ek+3CNMVM2wIynsLOt1pRFK
4/5RNjRLbwI6EtoCQfpLcZJ//SB56sK4DoFKH28Ok4cplESPnoMqA3QafdSEA/FL
qvZV/iPgtTz7vjQkMgrXAIUM4fvKe3iXkAExGXtmgdXHVFoKmHrxJ2DTSvM7/19z
jGJeu2MhIKHyqEmCk6hLjxyCE5pAH59KlbAQOP1bS28xlRskBApm2wN+LOZWzC62
HhEReQ50inCGuuubK0PqUQnyYc+lUFxrFpcliQEiBBABAgAMBQJPv9lVBQMAEnUA
AAoJEJcQuJvKV618AzgH/iRFFCi4qjvoqji1fi7yNPZVOMMO2H13Ks+AfcjRtHuV
aa30u50ND7TH+XQe6yerTapLh3aAm/sNP99aTxIuwRSlyKEoDs93+XVSgRqPBgbF
/vxv0ykok3p6L9DxFO/w5cL8JrBhMZoJrEkIBFkwN8tWlcXPRFQvcdBYv3M3DTZU
qY+UHnOxHvSzsl+LJ0S9Xcd9C5bvYfabmYJvG5eRS3pj1L/y3a6yw6hvY+JtnQAk
t05TdeHMIgQH/zb8V9wxDzmE0un8LyoC2Jx5TpikQsJSejwK6b3coxVBlngku6+C
qDAimObZLw6H9xYYIK0FoJs7j5bQZEwUO7OLBgjcMOqJASIEEAECAAwFAk/Rpc8F
AwASdQAACgkQlxC4m8pXrXw49Qf/TdNbun2htQ+cRWarszOx8BLEiW/x6PVyUQpZ
nV/0qvhKzlJUjM9hQPcA0AsOjhqtCN6Cy8KXbK/TvPm9D/Nk6HWwD1PomzrJVFk2
ywGFIuTR+lluKSp7mzm5ym0wJs5cPq731Im31RUQU8ndjLrq9YOf5FVL8NqmcOAU
4E8d68BbmVCQC5MMr0901FKwKznShfpy7VYN25/BASj8dhnynBYQErqToOJB6Cnd
JhdTlbfR4SirqAYZZg3XeqGhByytEHE1x7FMWWFYhdNtsnAVhYBbWqAzBs8lF9Jd
Mhaf0VQU/4z10gVrRtXLR/ixrCi+P4cM/fOQkqd6pwqWkaXt6okBIgQQAQIADAUC
T+NxIAUDABJ1AAAKCRCXELibyletfFBBCAC6+0TUJDcNaqOxOG1KViY6KYg9NCL8
pwNK+RKNK/N1V+WGJQH7qDMwRoOn3yogrHax4xIeOWiILrvHK0O6drS1DjsymIhR
Sm2XbE/8pYmEbuJ9vHh3b/FTChmSAO7dDjSKdWD3dvaY8lSsuDDqPdTX8FzOfrXC
M22C/YPg7oUG2A5svE1b+yismP4KmVNWAepEuPZcnEMPFgop3haHg9X2+mj/btDB
Yr6p9kAgIY17nigtNTNjtI0dMLu43aIzedCYHqOlNHiB049jkJs54fMGBjF9qPtc
m0k44xyKd1/JXWMdNUmtwKsChAXJS3YOciMgIx6tqYUTndrP4I6q1rfriQEiBBAB
AgAMBQJP9T1VBQMAEnUAAAoJEJcQuJvKV618J9wIAI1lId9SMbEHF6PKXRe154lE
pap5imMU/lGTj+9ZcXmlf8o2PoMMmb3/E1k+EZUaeSBoOmjS8C2gwd5XFwRrlwAD
RlK/pG5XsL4h5wmN2fj1ororrJXvqH427PLRQK9yzdwG4+9HTBOxjoS8qZT9plyK
AJZzAydAMqyseRHgNo0vMwlgrs4ojo+GcFGQHrF3IaUjvVfUPOmIj7afopFdIZmI
GaSF0TXBzqcZ1chFv/eTBcIuIKRvlaDee5FgV7+nLH2nKOARCLvV/+8uDi2zbr83
Ip5x2tD3XuUZ0ZWxD0AQWcrLdmGb4lkxbGxvCtsaJHaLXWQ2m760RjIUcwVMEBKJ
ASIEEAECAAwFAlAGYWsFAwASdQAACgkQlxC4m8pXrXwyVAgAvuvEl6yuGkniWOlv
uHEusUv/+2GCBg6qV+IEpVtbTCCgiFjYR5GasSp1gpZ5r4BocOlbGdjdJGHTpyK8
xD1i+6qZWUYhNRg2POXUVzcNEl2hhouwPLOifcmTwAKU76TEv3L5STviL3hWgUR2
yEUZ3Ut0IGVV6uPER9jpR3qd6O3PeuFkwf+NaGTye4jioLAy3aYwtZCUXzvYmNLP
90K4y+5yauZteLmNeq26miKC/NQu4snNFClPbGRjHD1ex9KDiAMttOgN4WEq7srT
rYgtT531WY4deHpNgoPlHPuAfC0H+S6YWuMbgfcb6dV+Rrd8Ij6zM3B/PcjmsYUf
OPdPtIkBIgQQAQIADAUCUBgtfQUDABJ1AAAKCRCXELibyletfAm3CACQlw21Lfeg
d8RmIITsfnFG/sfM3MvZcjVfEAtsY3fTK9NiyU0B3yX0PU3ei37qEW+50BzqiStf
5VhNvLfbZR+yPou7o2MAP31mq3Uc6grpTV64BRIkCmRWg40WMjNI1hv7AN/0atgj
ATYQXgnEw7mfFb0XZtMTD6cmrz/A9nTPVgZDxzopOMgCCC1ZK4Vpq9FKdCYUaHpX
3sqnDf+gpVIHkTCMgWLYQOeX5Nl+fgnq6JppaQ3ySZRUDr+uFUs0uvDRvI/cn+ur
ri92wdDnczjFumKvz/cLJAg5TG2Jv1Jx3wecALsVqQ3gL7f7vr1OMaqhI5FEBqdN
29L9cZe/ZmkriQEiBBIBCgAMBQJVoNxyBYMHhh+AAAoJEEoz7NUmyPxLD1EH/2eh
7a4+8A1lPLy2L9xcNt2bifLfFP2pEjcG6ulBoMKpHvuTCgtX6ZPdHpM7uUOje/F1
CCN0IPB533U1NIoWIKndwNUJjughtoRM+caMUdYyc4kQm29Se6hMPDfyswXE5Bwe
PmoOm4xWPVOH/cVN04zyLuxdlQZNQF/nJg6PMsz4w5z+K6NGGm24NEPcc72iv+6R
Uc/ry/7v5cVu4hO5+r104mmNV5yLecQF13cHy2JlngIHXPSlxTZbeJX7qqxE7TQh
5nviSPgdk89oB5jFSx4g1efXiwtLlP7lbDlxHduomyQuH9yqmPZMbkJt9uZDc8Zz
MYsDDwlc7BIe5bGKfjqJAhwEEAECAAYFAlSanFIACgkQdzHqU52lcqLdvg//cAEP
qdN5VTKWEoDFjDS4I6t8+0KzdDWDacVFwKJ8RAo1M2SklDxnIvnzysZd2VHp5Pq7
i4LYCZo5lDkertQ6LwaQxc4X6myKY4LTA652ObFqsSfgh9kW+aJBBAyeahPQ8CDD
+Yl23+MY5wTsj4qt7KffNzy78vLbYnVnvRQ3/CboVix0SRzg0I3Oi7n3B0lihvXy
5goy9ikjzZevejMEfjfeRCgoryy9j5RvHH9PF3fJVtUtHCS4f+kxLmbQJ1XqNDVD
hlFzjz8oUzz/8YXy3im5MY7Zuq4P4wWiI7rkIFMjTYSpz/evxkVlkR74qOngT2pY
VHLyJkqwh56i0aXcjMZiuu2cymUt2LB9IsaMyWBNJjXr2doRGMAfjuR5ZaittmML
yZwix9mWVk7tkwlIxmT/IW6Np0qMhDZcWYqPRpf7+MqY3ZYMK4552b8aDMjhXrnO
OwLsz+UI4bZa1r9dguIWIt2C2b5C1RQ9AsQBPwg7h5P+HhRuFAuDKK+vgV8FRuzR
```

```
JeKkFqwB4y0Nv7BzKbFKmP+V+/krRv+/Dyz9Bz/jyAQgw02u1tPupH9BGhlRyluN
yCJFTSNj7G+OLU0/l4XNph5OOC7sy+AMZcsL/gsT/TXCizRcCuApNTPDaenACpbv
g8OoIzmNWhh4LXbAUHCKmY//hEw9PvTZA1xKHgyJAhwEEgECAAYFAlJYsKQACgkQ
oirk60MpxUV2XQ//b2/uvThkkbeOegusDC4AZfjnL/V3mgk4iYy4AC9hum0R9oNl
XDR51P1TEw9mC1btHj+7m7Iq1a5ke5wIC7ENZiilr0yPqeWgL5+LC98dz/L85hqA
wIoGeOfMhrlaVbAZEj4yQTAJDA35vZHVsQmp87il0m+fZX04OBLXBzw86EoAAZ7Q
EoH4qFcT9k1T363tvNnIm3mEvkQ5WjE1R9uchJa1g7hdlNQlVkjFmPZrJK9fl4z5
6Dto89Po4Sge48jDH0pias4HATYHsxW819nz5jZzGcxLnFRRR5iITVZi9qzsHP7N
bUh3qxuWCHS9xziXpOcSZY848xXw63Y5jDJfpzupzu/KHj6CzXYJUEEqp9MluoGb
/BCCEPzdZ0ovyxFutM/BRcc6DvE6sTDF/UES21ROqfuwtJ6qJYWX+lBIgyCJvj4o
RdbzxUleePuzqCzmwrIXtoOKW0Rlj4SCeF9yCwUMBTGW5/nCLmN4dwf1KW2RP2Eg
4ERbuUy7QnwRP5UCl+0ISZJyYUISfg8fmPIdQsetUK9Cj+Q5jpB2GXwELXWnIK6h
K/6jXp+EGEXSqdIE53vAFe7LwfHiP/D5M71D2h62sdIOmUm3lm7xMOnM5tKlBiV+
4jJSUmriCT62zo710+6iLGqmUUYlEll6Ppvo8yuanXkYRCFJpSSP7VP0bBqIZgQT
EQIAJgUCTnc9dgIbIwUJEPPzpwYLCQgHAwIEFQIIAwQWAgMBAh4BAheAAAoJEIxx
jTtQcuH1Ut4AoIKjhdf70899d+7JFq3LD7zeeyI0AJ9Z+YyE1HZSnzYi73brScil
bIV6sbQ7TXlTUUwgUGFja2FnZSBzaWduaW5nIGtleSAod3d3Lm15c3FsLmNvbSkg
PGJ1aWxkQG15c3FsLmNvbT6IbwQwEQIALwUCTnc9rSgdIGJ1aWxkQG15c3FsLmNv
bSB3aWxsIHN0b3Agd29ya2luZyBzb29uAAoJEIxxjTtQcuH1tT0An3EMrSjEkUv2
9OX05JkLiVfQr0DPAJwKtL1ycnLPv15pGMvSzav8JyWN3IhlBBMRAgAdBQJHrJS0
BQkNMFioBQsHCgMEAxUDAgMWAgECF4AAEgkQjHGNO1By4fUHZUdQRwABAa6SAJ9/
PgZQSPNeQ6LvVVzCALEBJOBt7QCffgs+vWP18JutdZc7XiawgAN9vmmITAQTEQIA
DAUCPj6j0QWDCWYAuwAKCRBJUOEqsnKR8iThAJ9ZsR4o37dNGyl77nEqP6RAlJqa
YgCeNTPTEVY+VXHR/yjfyo0bVurRxT2ITAQTEQIADAUCPkKCAwWDCWIiiQAKCRC2
9c1NxrokP5aRAKCIaaegaMyiPKenmmm8xeTJSR+fKQCgrv0TqHyvCRINmi6LPucx
GKwfy7KIRgQQEQIABgUCP6zjrwAKCRCvxSNIeIN0D/aWAKDbUiEgwwAFNh2n8gGJ
Sw/8lAuISgCdHMzLAS26NDP8T2iejsfUOR5sNriIRgQQEQIABgUCP7RDdwAKCRCF
lq+rMHNOZsbDAJ0WoPV+tWILtZG3wYqg5LuHM03faQCeKuVvCmdPtro06xDzeeTX
VrZ14+GIRgQQEQIABgUCQ1uz6gAKCRCL2C5vMLlLXH90AJ0QsqhdAqTAk3SBnO2w
zuSOwiDIUwCdFExsdDtXf1cL3Q4ilo+OTdrTW2CIRgQTEQIABgUCRPEzJgAKCRD2
ScT0YJNTDApxAKCJtqT9LCHFYfWKNGGBgKjka0zi9wCcCG3MvnvBzDUqDVebudUZ
61Sont+ITAQQEQIADAUCQYHLAQWDBiLZiwAKCRAYWdAfZ3uh7EKNAJwPywk0Nz+Z
Lybw4YNQ7H1UxZycaQCePVhY4P5CHGjeYj9SX2gQCE2SNx+ITAQQEQIADAUCQYHL
NAWDBiLZWAAKCRCBwvfr4hO2kiIjAJ0VU1VQHzF7yYVeg+bh31nng9OOkwCeJI8D
9mx8neg4wspqvgXRA8+t2saITAQQEQIADAUCQYHLYgWDBiLZKgAKCRBrcOzZXcP0
cwmqAJsFjOvkY9c5eA/zyMrOZ1uPB6pd4QCdGyzgbYb/eoPu6FMvVI9PVIeNZReI
TAQQEQIADAUCQdCTJAWDBdQRaAAKCRB9JcoKwSmnwmJVAKCG9a+Q+qjCzDzDtZKx
5NzDW1+W+QCeL68seX8OoiXLQuRlifmPMrV2m9+ITAQQEQIADAUCQitbugWDBXlI
0gAKCRDmG6SJFeu5q/MTAKCTMvlCQtLKlzD0sYdwVLHXJrRUvgCffmdeS6aDpwIn
U0/yvYjg1xlYiuqITAQSEQIADAUCQCpZOgWDB3pLUgAKCRA8oR80lPr4YSZcAJwP
4DncDk4YzvDvnRbXW6SriJn1yQCdEy+d0CqfdhM7HGUs+PZQ9mJKBKqITAQSEQIA
DAUCQD36ugWDB2ap0gAKCRDy11xj45xlnLLfAKC0NzCVqrbTDRw25cUss14RRoUV
PACeLpEc3zSahJUB0NNGTNlpwlTczlCITAQSEQIADAUCQQ4KhAWDBpaaCAAKCRA5
yiv0PWqKX/zdAJ4hNn3AijtcAyMLrLhlZQvib551mwCgw6FEhGLjZ+as0W681luc
wZ6PzW+ITAQSEQIADAUCQoClNAWDBSP/WAAKCRAEDcCFfIOfqOMkAJwPUDhS1eTz
gnXclDKgf353LbjvXgCeLCWyyj/2d0gIk6SqzaPl2UcWrqiITAQTEQIADAUCPk1N
hAWDCVdXCAAKCRAtu3a/rdTJMwUMAKCVPkbk1Up/kyPrlsVKU/Nv3bOTZACfW5za
HX38jDCuxsjIr/084n4kw/uITAQTEQIADAUCQdeAdgWDBc0kFgAKCRBm79vIzYL9
Pj+8AJ9d7rvGJIcHzTCSYVnaStv6jP+AEACeNHa5yltqieRBCCcLcacGqYK81omI
TAQTEQIADAUCQhiBDgWDBYwjfgAKCRB2wQMcojFuoaDuAJ9CLYdysef7IsW42UfW
hI6HjxkzSgCfeEpXS4hEmmGicdpRiJQ/W21aB0GIZQQTEQIAHQULBwoDBAMVAwID
FgIBAheABQJLcC/KBQkQ8/OnABIHZUdQRwABAQkQjHGNO1By4fWw2wCeJilgEarL
8eEyfDdYTyRdqE45HkoAnjFSZY8Zg/iXeErHI0r04BRukNVgiHsEMBECADsFAkJ3
NfU0HQBPb3BzLi4uIHNob3VsZCBoYXZlIGJlZW4gbG9jYWwhIEknbSAqc28qIHN0
dXBpZC4uLgAKCRA5yiv0PWqKX+9HAJ0WjTx/rqgouK4QCrOV/2IOU+jMQQCfYSC8
JgsIIeN8aiyuStTdYrk0VWCIjwQwEQIATwUCRW8Av0gdAFNob3VsZCBoYXZlIGJl
ZW4gYSBsb2NhbCBzaWduYXR1cmUsIG9yIHNvbWV0aGluZyAtIFdURiB3YXMgSSB0
aGlua2luZz8ACgkQOcor9D1qil+g+wCfcFWoo5qUl4XTE9K8tH3Q+xGWeYYAnjii
KxjtOXc0ls+BlqXxbfZ9uqBsiQIiBBABAgAMBQJBgcuFBYMGItkHAAoJEKrj5s5m
oURoqC8QAIISudocbJRhrTAROOPoMsReyp46Jdp3iL1oFDGcPfkZSBwWh8L+cJjh
dycIwwSeZ1D2h9S5Tc4EnoE0khsS6wBpuAuih5s//coRqIIiLKEdhTmNqulkCH5m
imCzc5zXWZDW0hpLr2InGsZMuh2QCwAkB4RTBM+r18cUXMLV4YHKyjIVaDhsiPP/
MKUj6rJNsUDmDq1GiJdOjySjtCFjYADlQYSD7zcd1vpqQLThnZBESvEoCqumEfOP
xemNU6xAB0CL+pUpB40pE6Un6Krr5h6yZxYZ/N5vzt0Y3B5UUMkgYDSpjbulNvaU
TFiOxEU3gJvXc1+h0BsxM7FwBZnuMA8LEA+UdQb76YcyuFBcROhmcEUTiducLu84
E2BZ2NSBdymRQKSinhvXsEWlH6Txm1gtJLynYsvPi4B4JxKbb+awnFPusL8W+gfz
jbygeKdyqzYgKj3M79R3geaY7Q75Kxl1UogiOKcbI5VZvg47OQCWeeERnejqEAdx
```

```
EQiwGA/ARhVOP/1l0LQA7jg2P1xTtrBqqC2ufDB+v+jhXaCXxstKSW1lTbv/b0d6
454UaOUV7RisN39pE2zFvJvY7bwfiwbUJVmYLm4rWJAEOJLIDtDRtt2h8JahDObm
3CWkpadjw57S5v1c/mn+xV9yTgVx5YUfC/788L1HNKXfeVDq8zbAiQIiBBMBAgAM
BQJCnwocBYMFBZpwAAoJENjCCglaJFfPIT4P/25zvPp8ixqV85igs3rRqMBtBsj+
5EoEW6DJnlGhoi26yf1nasC2frVasWG7i4JIm0U3WfLZERGDjR/nqlOCEqsP5gS3
43N7r4UpDkBsYh0WxH/ZtST5llFK3zd7XgtxvqKL98l/OSgijH2W2SJ9DGpjtO+T
iegq7igtJzw7Vax9z/LQH2xhRQKZR9yernwMSYaJ72i9SyWbK3k0+e95fGnlR5pF
zlGq320rYHgD7v9yoQ2t1klsAxK6e3b7Z+RiJG6cAU8o8F0kGxjWzF4v8D1op7S+
IoRdB0Bap01ko0KLyt3+g4/33/2UxsW50BtfqcvYNJvU4bZns1YSqAgDOOanBhg8
Ip5XPlDxH6J/3997n5JNj/nk5ojfd8nYfe/5TjflWNiput6tZ7frEki1wl6pTNbv
V9C1eLUJMSXfDZyHtUXmiP9DKNpsucCUeBKWRKLqnsHLkLYydsIeUJ8+ciKc+EWh
FxEY+Ml72cXAaz5BuW9L8KHNzZZfez/ZJabiARQpFfjOwAnmhzJ9r++TEKRLEr96
taUI9/8nVPvT6LnBpcM38Td6dJ639YvuH3ilAqmPPw50YvglIEe4BUYD5r52Seqc
8XQowouGOuBX4vs7zgWFuYA/s9ebfGaIw+uJd/56Xl9ll6q5CghqB/yt1EceFEnF
CAjQc2SeRo6qzx22iEYEEBECAAYFAkSAbycACgkQCywYeUxD5vWDcACfQsVk/XGi
ITFyFVQ3IR/3Wt7zqBMAoNhso/cX8VUfs2BzxPvvGS3y+5Q9iEYEEBECAAYFAkUw
ntcACgkQOI4l6LNBlYkyFgCbBcw5gIii0RTDJsdNiuJDcu/NPqEAniSq9iTaLjgF
HZbaizUU8arsVCB5iEYEEBECAAYFAkWho2sACgkQu9u2hBuwKr6bjwCfa7ZK6O+X
mT08Sysg4DEoZnK4L9UAoLWgHuYg35wbZYx+ZUTh98diGU/miF0EExECAB0FCwcKAwQD
AxUDAgMWAgECF4AFAkequSEFCQ0ufRUACgkQjHGNO1By4fUdtwCfRNcueXikBMy7
tE2BbfwEyTLBTFAAnifQGbkmcARVS7nqauGhe1ED/vdgiF0EExECAB0FCwcKAwQD
FQMCAxYCAQIXgAUCS3AuZQUJEPPyWQAKCRCMcY07UHLh9aA+AKCHDkOBKBrGb8tO
g9BIub3LFhMvHQCeIOOot1hHHUlsTIXAUrD8+ubIeZaJARwEEgECAAYFAkvCIgMA
CgkQ3PTrHsNvDi8eQgf/dSx0R9Klozz8iK79w00NOsdoJY0Na0NTFmTbqHg30XJo
G62cXYgc3+TJnd+pYhYi5gyBixF/L8k/kPVPzX9W0YfwChZDsfTw0iDVmGxOswiN
jzSo0lhWq86/nEL30Khl9AhCC1XFNRw8WZYq9Z1qUXHHJ2rDARaedvpKHOjzRY0N
dx6R2zNyHDx2mlfCQ9wDchWEuJdAv0uHrQ0HV9+xq7lW/Q3L/V5AuU0tiowyAbBL
PPYrB6x9vt2ZcXS7BOy8SfQ1i8W2QDQ/Toork4YwBiv6WCW/ociy7paAoPOWV/Nf
2S6hDispeecbk7wqpbUj5klDmwrlgB/jmoAXWEnbsYkBIgQQAQIADAUCSSpooAUD
ABJ1AAAKCRCXELibyletfFOMCACpP+OVZ7lH/cNY+373c4FnSI0/S5PXS0ABgdd4
BFWRFWKrWBeXBGc8sZfHOzVEwkzV96iyHbpddeAOAkEA4OVPW1MMFCmlHxi2s9/N
JrSrTPVfQOH5fR9hn7Hbpq/ETw0IoX1FKo7vndMnHZnFEnI+PDXLcdMYQgljYzhT
xER4vYY0UKu8ekSshUy4zOX7XSJxwqPUvps8qs/TvojIF+vDJvgFYHVkgvS+shp8
Oh/exg9vKETBlgU87Jgsqn/SN2LrR/Jhl0aLd0G0iQ+/wHmVYdQUMFaCZwk/BKNa
XPzmGZEUZ3RNbYa19Mo7hcE3js76nh5YMxFvxbTggVu4kdFkiQEiBBABAgAMBQJK
M06IBQMAEnUAAAoJEJcQuJvKV618F4gH/innejIHffGMk8jYix4ZZT7pW6ApyoI+
N9Iy85H4L+8rVQrtcTHyq0VkcN3wPSwtfZszUF/0qP6P8sLJNJ1BtrHxLORYjJPm
gveeyHPzA2oJl6imqWUTiW822fyjY/azwhvZFzxmvbFJ+r5N/Z57+Ia4t9LTSqTN
HzMUYaXKDaAqzZeK7P0E6XUaaeygbjWjBLQ1O0ezozAy+Kk/gXApmDCGFuHSFe7Z
mgtFcbXLM2XFQpMUooETD2R8MUsd+xnQsff/k6pQOLxi+jUEsWSr/iqmvlk6gZ4D
pemBjuhcXYlxJYjUaX9Zmn5s+ofF4GFxRqXoY7l9Z+tCM9AX37lm6S+JASIEEAEC
AAwFAkpEcgoFAwASdQAACgkQlxC4m8pXrXz2mgf/RQkpmMM+5r8znx2TpRAGHi5w
ktvdFxlvPaOBWE28NDwTrpcoMqo9kzAiuvEQjVNihbP21wR3kvnQ84rTAH0mlC2I
uyybggpqwzOUl+Wi0o+vk8ZA0A0dStWRN8uqneCsd1XnqDe1rvqC4/9yY223tLmA
kPvz54ka2vX9GdJ3kxMWewhrVQSLCktQpygU0dujGTDqJtnk0WcBhVF9T87lv3W2
eGdPielzHU5trXezmGFj21d56G5ZFK8co7RrTt4qdznt80glh1BTGmhLlzjMPLTe
dcMusm3D1QB9ITogcG94ghSf9tEKmmRJ6OnnWM5Kn9KcL63E5oj2/lY9H54wSYkB
IgQQAQIADAUCSlY+RwUDABJ1AAAKCRCXELibyletfOOQB/0dyJBiBjgf+8d3yNID
pDktLhZYw8crIjPBVdOgX12xaUYBTGcQITRVHSggzffDA5BQXeUuWhpL4QB0uz1c
EPPwSMiWiXlBtwF5q6RVf3PZGJ9fmFuTkPRO7SruZeVDo9WP8HjbQtOLukYf566e
grzAYR9p74UgWftpDtmrqrRTobiuvsFBxosbeRCvEQCrN0n+p5D9hCVB88tUPHnO
WA4mlduAFZDxQWTApKQ92frHiBqy+M1JFezz2OM3fYN+Dqo/Cb7ZwOAA/2dbwS7o
y4sXEHbfWonjskgPQwFYB23tsFUuM4uZwVEbJg+bveglDsDStbDlfgArXSL/0+ak
lFcHiQEiBBABAgAMBQJKaAqEBQMAEnUAAAoJEJcQuJvKV618rH0H/iCciD4U6YZN
JBj0GN7/Xt851t9FWocmcaC+qtuXnkFhplXkxZVOCU4VBMs4GBoqfIvagbBTyfV4
Di+W8Uxr+/1jiu3l/HvoFxwdwNkGG6zNBhWSjdwQpGwPvh5ryV1OfLX/mgQgdDmx
vqz5+kFDUj4m7uLaeuU2j1T0lR4zU0yAsbt7J3hwfqJCXHOc9bm5nvJwMrSm+sdC
TP5HjUlwHr9mTe8xuZvj6sO/w0P4AqIMxjC9W7pT9q0ofG2KSTwt7wFbh05sbG4U
QYOJe4+Soh3+KjAa1c0cvmIh4cKX9qfCWwhhdeNfh1A9VTHhnl5zTv/UjvnQtjhl
H/Fq1eBSKcSJASIEEAECAAwFAkp5LgoFAwASdQAACgkQlxC4m8pXrXwY6wgAg3f8
76L3qDZTYlFAWs3pXBl8GsUr1DEkTlEDZMZKDM3wPmhaWBR1hMA3y6p3aaCUyJIJ
BEneXzgyU9uqCxXpC78d5qc3xs/Jd/SswzNYuvuzLYOw5wN5L31SLmQTQ8KqE0uo
RynBmtDCQ4M2UKifSnv+0+3mPh85LVAS481GNpL+VVfCYtKesWNu40+98Yg6L9NG
```

```
WwRTfsQbcdokZo44Jz7Y7f81ObC4r/X1DgPj2+d4AU/plzDcdrbINOyprs+7340e
cnaGO4Lsgd19b1CvcgJgltRquu3kRvd+Ero2RYpDv6GVK8Ea0Lto4+b/Ae8cLXAh
QnaWQCEWmw+AU4Jbz4kBIgQQAQIADAUCSo5fvQUDABJ1AAAKCRCXELibyletfA08
B/9w8yJdc8K+k07U30wR/RUg3Yb2lBDygmy091mVsyB0RGixBDXEPOXBqGKAXiV1
QSMAXM2VKRsuKahY2HFkPbyhZtjbdTa7Pr/bSnPvRhAh9GNWvvRg2Kp3qXDdjv9x
ywEghKVxcEIVXtNRvpbqRoKmHzIExvUQck5DM1VwfREeYIoxgs4035WADhVMdngQ
S2Gt8P2WaU/p8EZhFGg6X8KtOlD68zGboaJe0hj2VDc+Jc+KdjRfE3fW5ITOid/o
DkUaIW6tB3WkXb0g6D/2hrEJbX3headChHKSB8eQdOR9bcCJDhhU8csd501qmrhC
ctmvlpeWQZdIQdk6sABPWeeCiQEiBBABAgAMBQJKoBJHBQMAEnUAAAoJEJcQuJvK
V618Ml8H/1D88/g/p9fSVor4Wu5WlMbg8zEAik3BIxQruEFWda6nART6M9E7e+P1
++UHZsWYs6l9ROpWxRLG1Yy9jLec2Y3nUtb20m65p+IVeKR2a9PHW35WZDV9dOYP
GZabKkO1clLeWLVgp9LRjZ+AeRG+ljHqsULXro1dwewLTB/gg9I2vgNv6dKxyKak
nM/GrqZLATAq2KoaE/u/6lzRFZIzZnLtjZh8X7+nS+V8v9IiY4ntrpkrbvFk30U6
WJp79oBIWwnW/84RbxutRoEwSar/TLwVRkcZyRXeJTapbnLGnQ/lDO1o1d7+Vbjd
q/Sg/cKHHf7NthCwkQNsCnHL0f51gZCJASIEEAECAAwFAkqoEAAFAwASdQAACgkQ
lxC4m8pXrXwE/Af/XD4R/A5R6Ir/nCvKwCTKJmalajssuAcLEa2pMnFZYO/8rzLO
+Gp8p0qFH9C4LFwA0NvR5q6X/swuROf4zxljSvNcdlQVaAfJ2ZDEgJ5GXzsPplrv
SAI9jS3LL7fSWDZgKuUe0a4qx7A0NgyGMUYGhP+QlRFa8vWEBI9fANd/0mMqAeBV
qQyOH0X1FiW1Ca2Jn4NKfuMy9GEvRddVIbB1LvoNVtXPNzeeKMyNb9Jdx1MFWssy
COBP2DayJKTmjvqPEc/YOjOowoN5sJ/jn4mVSTvvlTooLiReSs6GSCAjMVxN7eYS
/Oyq6Iu1JDcJvmB8N2WixAZtAVgF8OA7CWXKVYkBIgQQAQIADAUCSrnHiQUDABJ1
AAAKCRCXELibyletfPChB/9uECti1dZeNuFsd0/RuGyRUVlrrhJE6WCcOrLO9par
rPbewbKBmjSzB0MygJXGvcC06mPNuquJ7/WpxKsFmfg4vJBPlADFKtgRUy9BLzjC
eotWchPHFBVW9ftPbaQViSUu7d89NLjDDM5xrh80puDIApxoQLDoIrh3T1kpZx56
jSWv0gelFUMbXAzmqkJSyL4Xdh1aqzgUbREd7Xf2ICzuh0sV6V7c/AwWtjWEGEsA
HZaiQDywZwbC18GwrMLiAzGWb/AScFDQRCZKJDjL+Ql8YT6z+ZMVr8gb7CIU5PKY
dhiIf2UVTQwLAoW7lNRCQQAqcGjK3IMIz7SO/yk4HmVUiQEiBBABAgAMBQJK3gjG
BQMAEnUAAAoJEJcQuJvKV618jkEH+wb0Zv9z7xQgpLMowVuBFQVu8/z7P5ASumyB
PUO3+0JVxSHBhlCKQK7n11m1fhuGt2fCxXhSU6LzXj36rsKRY53lGZ9QhvqFUtQH
3Xb2IQLIJC4UKjG2jSSCdcuA/x98bwp2v7O03rn7ndCS16CwXnRV3geQoNipRKMS
DajKPpZv1RiZm8pMKqEb8WSw352xWoOcxuffjlsOEwvJ85SEGCAZ9tmIlkZOc7Ai
QONDvii9b8AYhQ60RIQC0HP2ASSmK0V92VeFPxHmAygdDQgZNVtbVxgnnt7oTNEu
VRXNY+z4OfBArp7R+cTsvijDRZY4kML1n22hUybwoxUEvjqZV2+JASIEEAECAAwF
AkrvOlQFAwASdQAACgkQlxC4m8pXrXxrPAgArXiNgZirNuBhfNCXlkzkCHLx5wnV
e4SmTpbWzTwWw7+qk7d4l9hlWtdImISORINzo7f4ShSUzJX2GciNaXhaHRo7+y5O
Zbu82jQb09aQQj/nibKYuqxqUrobTEm+DuYz3JUQZm2PsPcHLS8mX9cxvrJUncPG
nXEV0DRaq71SGWDprtkvBbp6i38aY3sIhYgz8wM5m1szKDtjywmBYcFehIdozt9z
hm7wZshzRWQX1+Rf/pIsnk+OzBIa34crSemTnacbV/B7278z2XAyziPNFuqz0xu+
iltOmYmayfNWAmumuw9NcuwWMlth6Mc2HLrpo0ZBheJ6iuDMPsHnwqdB/4kBIgQQ
AQIADAUCSwBd2gUDABJ1AAAKCRCXELibyletfP6tB/4m1w0BtlkJgtS6E+B/ns14
z4A4PGors+n+MYm05qzvi+EnDF/sytCmVcKeimrtvDcfoDtKAFFvJjcYXfnJdGWm
Pu0SJMRL5KKCirAKwZmU/saxOgoB5QLNw+DHPteJ3w9GmWlGxIqG1r15WC5duzBC
y3FsnjJYG3jaLnHOO9yXXb5h0kUTORfUKdvAr1gxF2KoatZWqGoaPPnHoqb88rjt
zk8I7gDqoXnzh8wLxa0ZYvfTC/McxdWTrwXLft+krmMQ18iIZEne2hvVLNJVuluU
oiWLeHA8iNCQ4W4WTdLc1mCnCjGTMX/MN41uLH0C9Ka4R6wEaqj4lPDk1B/1TV+Q
iQEiBBABAgAMBQJLEYGrBQMAEnUAAAoJEJcQuJvKV618naIH/2t9aH5mBTKBN6fU
qhrf79vIsjtI/QNS5qisBISZMX3/1/0Gu6WnxkPSfdCUJMWCjMcnVj7KU2wxTHHG
VpAStd9r2afUNxRyqZwzwyytktuZok0XngAEDYDDBS3ssu2R4uWLCsC2ysXEqO/5
tI5YrTWJZrfeIphTaYP5hxrMujvqy3kEwKKbiMz91cDeiLS+YCBcalj5n/1dMYf7
8U8C6ieurxAg/L8h6x25VM4Ilx4MmG2T8QGtkkUXd+Fd/KYWmf0LE5LLPknf0Hhw
oVslPXeinp4FsHK/5wzviv4YZpzuTqs9NlKcMsa4IuuPOB0FDf0pn+OFQbEg9QwY
2gCozK+JASIEEAECAAwFAksjTdQFAwASdQAACgkQlxC4m8pXrXwlogf/XBGbXRVX
LMaRN4SczOjwT3/tUCriTkb3v+zKjRG90zFhYAccjn7w+7jKQicjq6quQG1EH2X4
/Su6ps1lDLqGHHhiJW3ZhxQScLZmhdAYsh2qG4GP/UW3QjXG7c61t+H3olvWg2cr
wqCxxFZAgkAAkr9xcHWFZJEQeXoob6cCZObaUnHSANdmC6s5lUxXYa2bmL7Q3UB4
4KCzDvAfbPZKJOw9k0qb3lc11zx+vGdyZFbm4R0+3LPp/vT0b3GlSbbF9lU1GOXh
VaphrgFFa76dmjfHCkPplXAkK1VSIU/aPGAefduTFMdlSZpdMtJ5AULjGcszBDlR
pLlPxvqVa0ZpgIkBIgQQAQIADAUCSycmkgUDABJ1AAAKCRCXELibyletfHlNCACp
1YespiHfQt2alcscE5zgfETEHHic8Ai6pNkU9HT4TeWcFHHEDe5QqfYcpjLrQvBXS
kSvxEittbyRdv+e+j5Z+HyHjiG8nAQBL6qy9eHqQE4+d7gYs6DTk7sG9ZMYphREb
ltzD+F4hVCQdLT8LNr0eVFN7ehqECScDaCG8/Qyti+l/0M902/Yn+mz0ilOiUdWJ
9x6LPaIINtb1gsYDEylLjwGIZmI0r5Kh9wYoV4vnNezFbxO1uRiW0B7iaPjIEsbt
OOKp7wx2aX+DM3N9F3BtaIY8XnzcnomNm83SNsgmgrZljpQltUnNqIhNM8DupQ+I
WOV5gtl6pTC7CgeVTVyRiQEiBBABAgAMBQJLOGXuBQMAEnUAAAoJEJcQuJvKV618
ll4IAKJ9mm4jb0c8fe9+uDI8eCJRbzNbVXm8zWzpA8GUtQAakwxoKv332QP1Wa1P
odni/e3EMhsSREOZJJv79YqGxGRBTE9Kb/VjM34nas4XSnXKW28XWhKyIw+XwQAi
nY2swFHh+83Htr/mwTdJfS2aEYl2zboBvd/JZCdhOGU2GH737S/3uEczoKkfVQ/w
```

```
OTM8X1xWwlYWqx23k/DsGcuDs9lA2g7Mx7DSqBtVjaTkn9h0zATzXLDkmP4SAUVj
cZ83WDpFre5WnizZjdXlBMM5OCexp5WpmzyHLTnaBFK4jEmnsk5C2Rnoyp8Ivz6g
Ecg1tRbEXijRw++d2TFYlJwLKtiJASIEEAECAAwFAktKMicFAwASdQAACgkQlxC4
m8pXrXxqHQgAuYY5scKrh0m/GS9EYnyC9494lOlO6iytU0CpE6oBC31M3hfX/Dbj
UbcS5szZNU+2CPYo4ujQLZ7suN7+tTjG6pZFfMevajT9+jsL+NPMF8RLdLOVYmbl
TmSQGNO+XGEYaKYH5oZIeIW5AKCgi2ozkdFlBBLAx7Kqo/FyybhkURFEcvEyVmgf
3KLV7IIiX/fYLfoCMCJ/Lcm9/llSFB1n8Nvg66Xd533DKoHjueD3jyaNAVlo2mq/
sIAv++kntvOiB3GDK5pfwHZ78WWiCpsWZpE5gzAnzJ1Y0WEigRo0PVLu3cLO0jLG
23d+H/CbfZ8rkajHJeCDQF7YVmP0t0nYpYkBIgQQAQIADAUCS1v+ZgUDABJ1AAAK
CRCXELibyletfNS/CACqt2TkB86mjqM+cJ74+dWBvJ2aFuURuxzm95i9Q/W/hU08
2iMbC3+0k2oD8CrTOe61P+3oRyLjv/UEDUNzLncNe2YsA9JeV+4hvPwH5Vp3Om13
089fCKZUbqslXNKkHiWYU+zAaZJXEuGRmRz0HbQIeAMOWF4oa226uo1e4ws1Jhc+
F3E/ApCRyFBqBUdL05hapQLditYpsBjIdiBGpjzidMLE2wX2W4ZpAdN0U6BIyIqR
mTPjbSkvzS9kSWFmfhQgnBDKEYJpVZgE1sN52rYC1sDeGeiuKxlzjVov9MMhYMWa
Zo3R5o3F2iIM/BK6FbC252lf/Mhu3ICuXujNBZNYiQEiBBABAgAMBQJLbSH4BQMA
EnUAAAoJEJcQuJvKV618kd0IAJLLwDH6gvgAlBFklQJXqQxUdcSOOVMAWtlHgWOy
ozjgomZZBkRL8dtCDr9YBMcj5czcQ3qpmLJdppXhKB+kJV2iUXfDMSFXwJ4wLfIs
8FNnXw8H5U01oBkGH/Ku6ngL9Vwt+MjYHtCWkw9QueUKZnDudX9qIzLAIt+mwSTu
A6+fY4VWIg40AA0v3exaQM55YR/UhlKunpGG9o8Qkq77dMEbTMpOmBoLbOMRB3Dd
MAvVU6G2l6Pcb7KobVCuOBnb6batXARV/G8sw+nzfJ16fr/KobZT2A6m+Jrqk4dl
F14ljLbz16O5JGUPAryN2G2ddBdSAy7dtFSVhWWiWC9n88q5Ag0EPj6jHRAIAO/h
iX8WzHWOMLJT54x/axeDdqn1rBDf5cWmaCWHN2ujNNlgpx5emoU9v7QStsNUCOGB
bXkeO4Ar7YG+jtSR33zqNh3y5kQ0YkY3dQ0wh6nsl+wh4XIIY/3TUZVtmdJeUBRH
JlfVNFYad2hX1guFI37Ny1PoZAFsxO82g+XB/Se8r/+sbmVcONdcdIeFKrE3FjLt
IjNQcxC6l9Q2Oy8KDxG/zvUZG3+H5i3tdRMyGgmuD6gEV0GXOHYUopzLeit1+Aa0
bCk36Mwbu+BeOw/CJW3+b0mB27hOaf9aCA855IP6fJFvtxcblq8nHIqhU3Dc9tec
sl9/S1xZ5S8ylG/xeRsAAwUH/i8KqmvAhq0X7DgCcYputwh37cuZlHOa1Ep07JRm
BCDgkdQXkGrsj2Wzw7Aw/TGdWWkmn2pxb8BRui5cfcZFO7c6vryi6FpJuLucX975
+eVY50ndWkPXkJ1HF4i+HJwRqE2zliN/RHMs4LJcwXQvvjD43EE3AO6eiVFbD+qA
AdxUFoOeLblKNBHPG7DPG9xL+Ni5rkE+TXShxsB7F0z7ZdJJZOG0JODmox7IstQT
GoaU9u41oyZTIiXPiFidJoIZCh7fdurP8pn3X+R5HUNXMr7M+ba8lSNxce/F3kmH
0L7rsKqdh9d/aVxhJINJ+inVDnrXWVoXu9GBjT8Nco1iU9SIVAQYEQIADAUCTnc9
7QUJE/sBuAASB2VHUEcAAQEJEIxxjTtQcuH1FJsAmwWK9vmwRJ/y9gTnJ8PWf0BV
roUTAKClYAhZuX2nUNwH4vlEJQHDqYa5yQ==
=ghXk
-----END PGP PUBLIC KEY BLOCK-----
```

To import the build key into your personal public GPG keyring, use `gpg --import`. For example, if you have saved the key in a file named `mysql_pubkey.asc`, the import command looks like this:

```
shell> gpg --import mysql_pubkey.asc
gpg: key 5072E1F5: public key "MySQL Release Engineering
<mysql-build@oss.oracle.com>" imported
gpg: Total number processed: 1
gpg:               imported: 1
gpg: no ultimately trusted keys found
```

You can also download the key from the public keyserver using the public key id, `5072E1F5`:

```
shell> gpg --recv-keys 5072E1F5
gpg: requesting key 5072E1F5 from hkp server keys.gnupg.net
gpg: key 5072E1F5: "MySQL Release Engineering <mysql-build@oss.oracle.com>"
1 new user ID
gpg: key 5072E1F5: "MySQL Release Engineering <mysql-build@oss.oracle.com>"
53 new signatures
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:           new user IDs: 1
gpg:         new signatures: 53
```

If you want to import the key into your RPM configuration to validate RPM install packages, you should be able to import the key directly:

```
shell> rpm --import mysql_pubkey.asc
```

If you experience problems or require RPM specific information, see Section 2.1.3.4, “Signature Checking Using RPM”.

After you have downloaded and imported the public build key, download your desired MySQL package and the corresponding signature, which also is available from the download page. The signature file has the same name as the distribution file with an `.asc` extension, as shown by the examples in the following table.

Table 2.1 MySQL Package and Signature Files for Source files

File Type	File Name
Distribution file	`mysql-standard-8.0.19-linux-i686.tar.gz`
Signature file	`mysql-standard-8.0.19-linux-i686.tar.gz.asc`

Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file:

```
shell> gpg --verify package_name.asc
```

If the downloaded package is valid, you will see a "Good signature" similar to:

```
shell> gpg --verify mysql-standard-8.0.19-linux-i686.tar.gz.asc
gpg: Signature made Tue 01 Feb 2011 02:38:30 AM CST using DSA key ID 5072E1F5
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
```

The `Good signature` message indicates that the file signature is valid, when compared to the signature listed on our site. But you might also see warnings, like so:

```
shell> gpg --verify mysql-standard-8.0.19-linux-i686.tar.gz.asc
gpg: Signature made Wed 23 Jan 2013 02:25:45 AM PST using DSA key ID 5072E1F5
gpg: checking the trustdb
gpg: no ultimately trusted keys found
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: A4A9 4068 76FC BD3C 4567  70C8 8C71 8D3B 5072 E1F5
```

That is normal, as they depend on your setup and configuration. Here are explanations for these warnings:

- *gpg: no ultimately trusted keys found*: This means that the specific key is not "ultimately trusted" by you or your web of trust, which is okay for the purposes of verifying file signatures.
- *WARNING: This key is not certified with a trusted signature! There is no indication that the signature belongs to the owner.*: This refers to your level of trust in your belief that you possess our real public key. This is a personal decision. Ideally, a MySQL developer would hand you the key in person, but more commonly, you downloaded it. Was the download tampered with? Probably not, but this decision is up to you. Setting up a web of trust is one method for trusting them.

See the GPG documentation for more information on how to work with public keys.

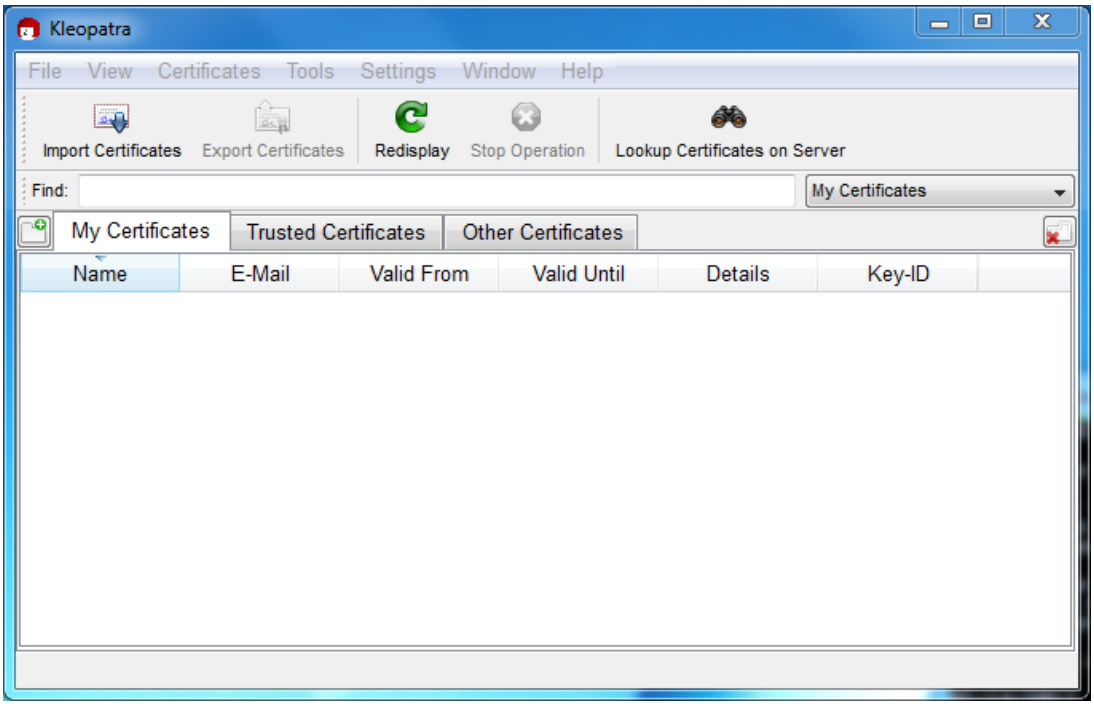
2.1.3.3 Signature Checking Using Gpg4win for Windows

The Section 2.1.3.2, “Signature Checking Using GnuPG” section describes how to verify MySQL downloads using GPG. That guide also applies to Microsoft Windows, but another option is to use a GUI

tool like Gpg4win. You may use a different tool but our examples are based on Gpg4win, and utilize its bundled `Kleopatra` GUI.

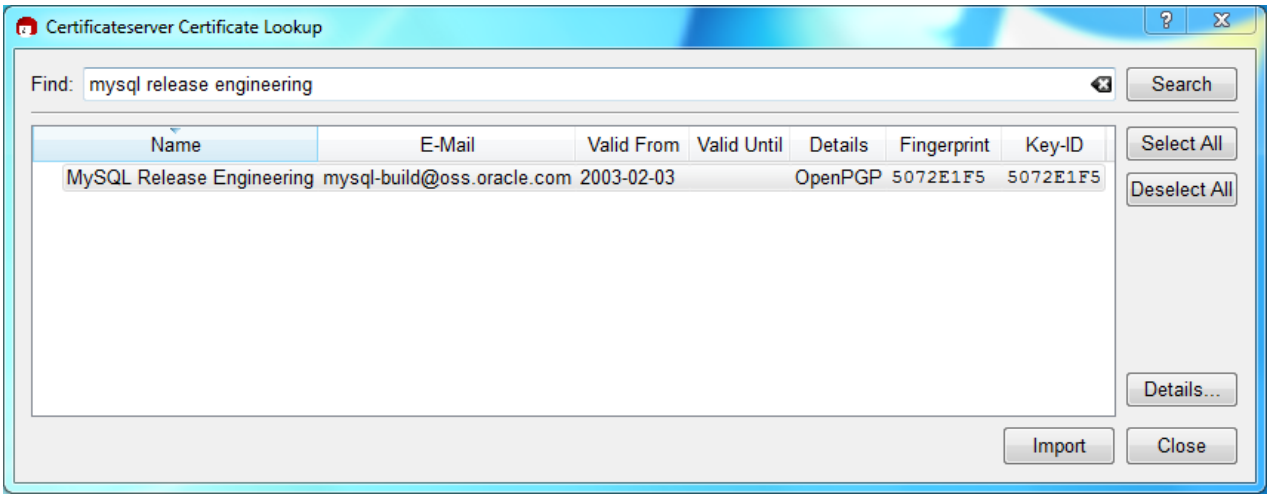
Download and install Gpg4win, and then load Kleopatra. The dialog should look similar to:

Figure 2.1 Kleopatra: Initial Screen



Next, add the MySQL Release Engineering certificate. Do this by clicking **File**, **Lookup Certificates on Server**. Type "Mysql Release Engineering" into the search box and press **Search**.

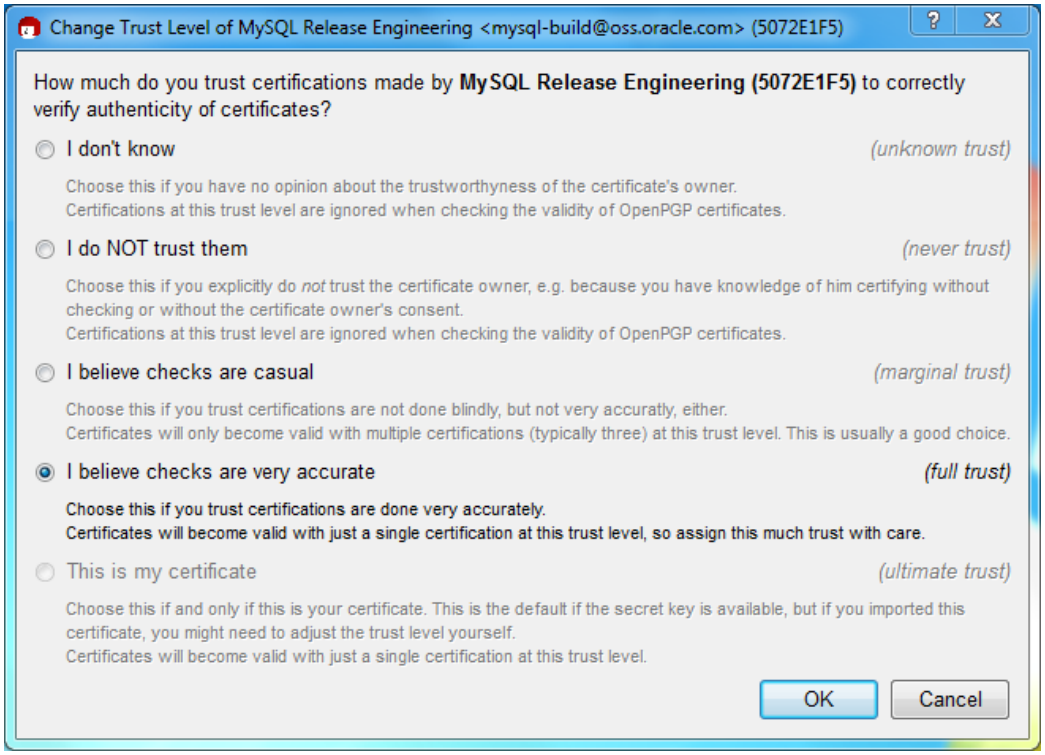
Figure 2.2 Kleopatra: Lookup Certificates on Server Wizard: Finding a Certificate



Select the "MySQL Release Engineering" certificate. The Fingerprint and Key-ID must be "5072E1F5", or choose **Details...** to confirm the certificate is valid. Now, import it by clicking **Import**. An import dialog will be displayed, choose **Okay**, and this certificate will now be listed under the **Imported Certificates** tab.

Next, configure the trust level for our certificate. Select our certificate, then from the main menu select **Certificates**, **Change Owner Trust...**. We suggest choosing **I believe checks are very accurate** for our certificate, as otherwise you might not be able to verify our signature. Select **I believe checks are very accurate** to enable "full trust" and then press **OK**.

Figure 2.3 Kleopatra: Change Trust level for MySQL Release Engineering



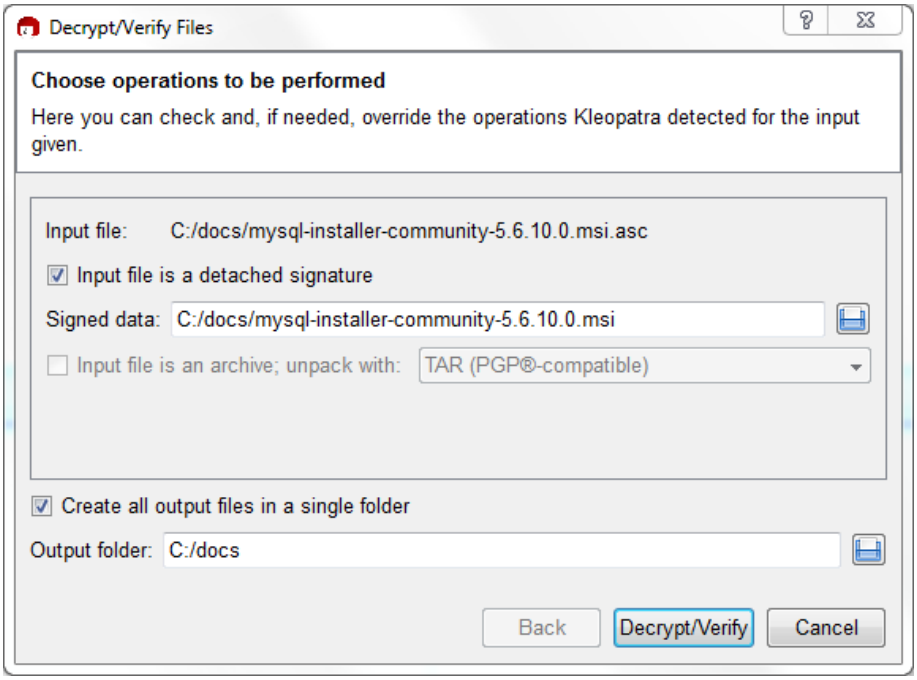
Next, verify the downloaded MySQL package file. This requires files for both the packaged file, and the signature. The signature file must have the same name as the packaged file but with an appended `.asc` extension, as shown by the example in the following table. The signature is linked to on the downloads page for each MySQL product. You must create the `.asc` file with this signature.

Table 2.2 MySQL Package and Signature Files for MySQL Installer for Microsoft Windows

File Type	File Name
Distribution file	`mysql-installer-community-8.0.19.msi`
Signature file	`mysql-installer-community-8.0.19.msi.asc`

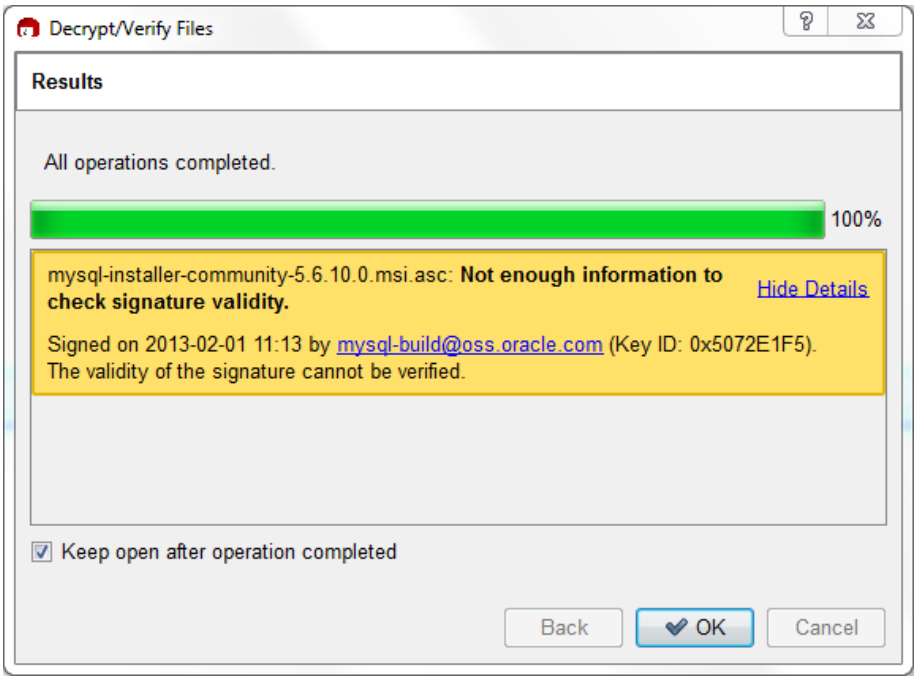
Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file. Either drag and drop the signature (`.asc`) file into Kleopatra, or load the dialog from **File**, **Decrypt/Verify Files...**, and then choose either the `.msi` or `.asc` file.

Figure 2.4 Kleopatra: The Decrypt and Verify Files Dialog



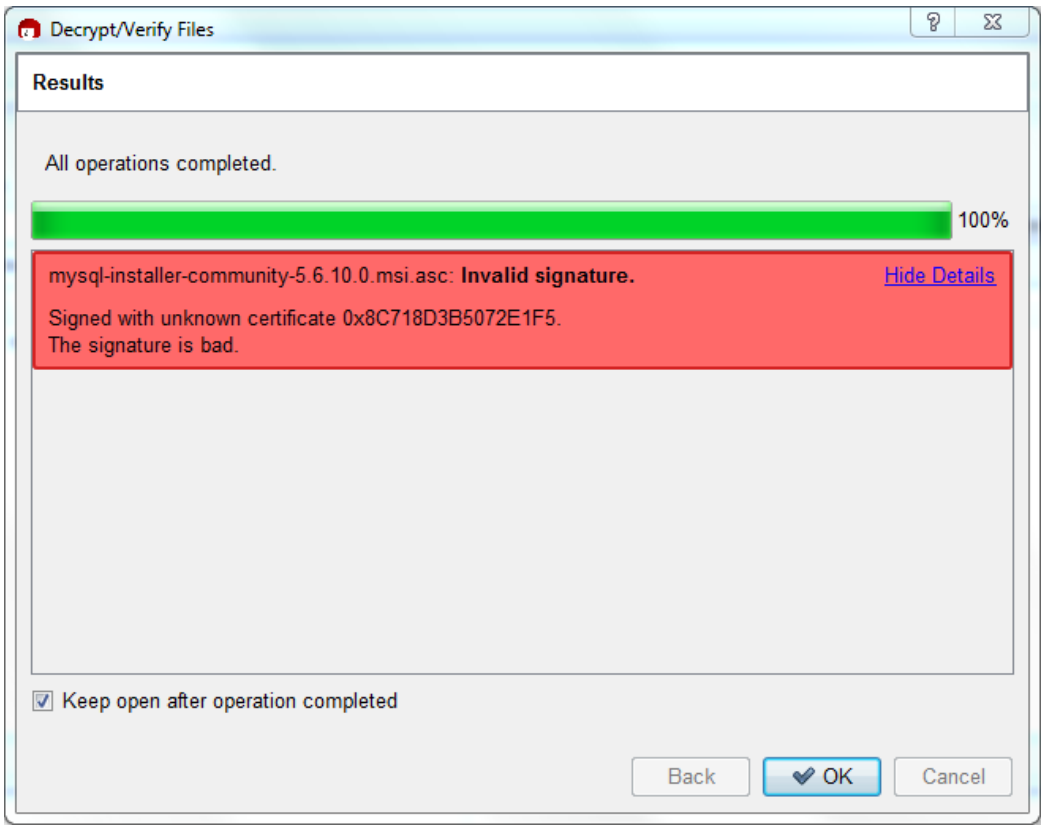
Click **Decrypt/Verify** to check the file. The two most common results will look like the following, and although the yellow warning looks problematic, the following means that the file check passed with success. You may now run this installer.

Figure 2.5 Kleopatra: the Decrypt and Verify Results Dialog: All operations completed



Seeing a red "The signature is bad" error means the file is invalid. Do not execute the MSI file if you see this error.

Figure 2.6 Kleopatra: the Decrypt and Verify Results Dialog: Bad



The Section 2.1.3.2, “Signature Checking Using GnuPG” section explains why you probably don't see a green `Good signature` result.

2.1.3.4 Signature Checking Using RPM

For RPM packages, there is no separate signature. RPM packages have a built-in GPG signature and MD5 checksum. You can verify a package by running the following command:

```
shell> rpm --checksig package_name.rpm
```

Example:

```
shell> rpm --checksig MySQL-server-8.0.19-0.linux_glibc2.5.i386.rpm
MySQL-server-8.0.19-0.linux_glibc2.5.i386.rpm: md5 gpg OK
```

Note

If you are using RPM 4.1 and it complains about `(GPG) NOT OK (MISSING KEYS: GPG#5072e1f5)`, even though you have imported the MySQL public build key into your own GPG keyring, you need to import the key into the RPM keyring first. RPM 4.1 no longer uses your personal GPG keyring (or GPG itself). Rather, RPM maintains a separate keyring because it is a system-wide application and a user's GPG public keyring is a user-specific file. To import the MySQL public key into the RPM keyring, first obtain the key, then use `rpm --import` to import the key. For example:

```
shell> gpg --export -a 5072e1f5 > 5072e1f5.asc
shell> rpm --import 5072e1f5.asc
```

Alternatively, `rpm` also supports loading the key directly from a URL, and you can use this manual page:

```
shell> rpm --import https://dev.mysql.com/doc/refman/8.0/en/checking-gpg-signature.html
```

If you need to obtain the MySQL public key, see Section 2.1.3.2, “Signature Checking Using GnuPG”.

2.1.4 Installation Layouts

The installation layout differs for different installation types (for example, native packages, binary tarballs, and source tarballs), which can lead to confusion when managing different systems or using different installation sources. The individual layouts are given in the corresponding installation type or platform chapter, as described following. Note that the layout of installations from vendors other than Oracle may differ from these layouts.

- Section 2.3.1, “MySQL Installation Layout on Microsoft Windows”
- Section 2.9.3, “MySQL Layout for Source Installation”
- Table 2.3, “MySQL Installation Layout for Generic Unix/Linux Binary Package”
- Table 2.11, “MySQL Installation Layout for Linux RPM Packages from the MySQL Developer Zone”
- Table 2.6, “MySQL Installation Layout on macOS”

2.1.5 Compiler-Specific Build Characteristics

In some cases, the compiler used to build MySQL affects the features available for use. The notes in this section apply for binary distributions provided by Oracle Corporation or that you compile yourself from source.

`icc` (Intel C++ Compiler) Builds

A server built with `icc` has these characteristics:

- SSL support is not included.

2.2 Installing MySQL on Unix/Linux Using Generic Binaries

Oracle provides a set of binary distributions of MySQL. These include generic binary distributions in the form of compressed `tar` files (files with a `.tar.xz` extension) for a number of platforms, and binaries in platform-specific package formats for selected platforms.

This section covers the installation of MySQL from a compressed `tar` file binary distribution on Unix/Linux platforms. For other platform-specific binary package formats, see the other platform-specific sections in this manual. For example, for Windows distributions, see Section 2.3, “Installing MySQL on Microsoft Windows”. See Section 2.1.2, “How to Get MySQL” on how to obtain MySQL in different distribution formats.

MySQL compressed `tar` file binary distributions have names of the form `mysql-`*`VERSION`*`-`*`OS`*`.tar.xz`, where *`VERSION`* is a number (for example, `8.0.19`), and *`OS`* indicates the type of operating system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

There is also a “minimal install” version of the MySQL compressed `tar` file for the Linux generic binary distribution, which has a name of the form `mysql-`*`VERSION`*`-`*`OS`*`-minimal.tar.xz`. The minimal install

distribution excludes debug binaries and is stripped of debug symbols, making it significantly smaller than the regular binary distribution. If you choose to install the minimal install distribution, remember to adjust for the difference in file name format in the instructions that follow.

Warnings

- If you have previously installed MySQL using your operating system native package management system, such as Yum or APT, you may experience problems installing using a native binary. Make sure your previous MySQL installation has been removed entirely (using your package management system), and that any additional files, such as old versions of your data files, have also been removed. You should also check for configuration files such as `/etc/my.cnf` or the `/etc/mysql` directory and delete them.

  For information about replacing third-party packages with official MySQL packages, see the related APT guide or Yum guide.

- MySQL has a dependency on the `libaio` library. Data directory initialization and subsequent server startup steps will fail if this library is not installed locally. If necessary, install it using the appropriate package manager. For example, on Yum-based systems:

  ```
  shell> yum search libaio  # search for info
  shell> yum install libaio # install library
  ```

  Or, on APT-based systems:

  ```
  shell> apt-cache search libaio # search for info
  shell> apt-get install libaio1 # install library
  ```

- **Oracle Linux 8 / Red Hat 8** (EL8): These platforms by default do not install the file `/lib64/libtinfo.so.5`, which is required by the MySQL client `bin/mysql` for packages `mysql-VERSION-el7-x86_64.tar.gz` and `mysql-VERSION-linux-glibc2.12-x86_64.tar.xz`. To work around this issue, install the `ncurses-compat-libs` package:

  ```
  shell> yum install ncurses-compat-libs
  ```

To install a compressed `tar` file binary distribution, unpack it at the installation location you choose (typically `/usr/local/mysql`). This creates the directories shown in the following table.

Table 2.3 MySQL Installation Layout for Generic Unix/Linux Binary Package

Directory	Contents of Directory
`bin`	`mysqld` server, client and utility programs
`docs`	MySQL manual in Info format
`man`	Unix manual pages
`include`	Include (header) files
`lib`	Libraries
`share`	Error messages, dictionary, and SQL for database installation
`support-files`	Miscellaneous support files

Debug versions of the `mysqld` binary are available as `mysqld-debug`. To compile your own debug version of MySQL from a source distribution, use the appropriate configuration options to enable debugging support. See Section 2.9, “Installing MySQL from Source”.

To install and use a MySQL binary distribution, the command sequence looks like this:

```
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
shell> cd /usr/local
shell> tar xvf /path/to/mysql-VERSION-OS.tar.xz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> mkdir mysql-files
shell> chown mysql:mysql mysql-files
shell> chmod 750 mysql-files
shell> bin/mysqld --initialize --user=mysql
shell> bin/mysql_ssl_rsa_setup
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

Note

This procedure assumes that you have `root` (administrator) access to your system. Alternatively, you can prefix each command using the `sudo` (Linux) or `pfexec` (Solaris) command.

The `mysql-files` directory provides a convenient location to use as the value for the `secure_file_priv` system variable, which limits import and export operations to a specific directory. See Section 5.1.8, “Server System Variables”.

A more detailed version of the preceding description for installing a binary distribution follows.

Create a mysql User and Group

If your system does not already have a user and group to use for running `mysqld`, you may need to create them. The following commands add the `mysql` group and the `mysql` user. You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following instructions. The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix/Linux, or they may have different names such as `adduser` and `addgroup`.

```
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
```

Note

Because the user is required only for ownership purposes, not login purposes, the `useradd` command uses the `-r` and `-s /bin/false` options to create a user that does not have login permissions to your server host. Omit these options if your `useradd` does not support them.

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it. The example here unpacks the distribution under `/usr/local`. The instructions, therefore, assume that you have permission to create files and directories in `/usr/local`. If that directory is protected, you must perform the installation as `root`.

```
shell> cd /usr/local
```

Obtain a distribution file using the instructions in Section 2.1.2, “How to Get MySQL”. For a given release, binary distributions for all platforms are built from the same MySQL source distribution.

Unpack the distribution, which creates the installation directory. `tar` can uncompress and unpack the distribution if it has `z` option support:

```
shell> tar xvf /path/to/mysql-VERSION-OS.tar.xz
```

The `tar` command creates a directory named `mysql-VERSION-OS`.

To install MySQL from a compressed `tar` file binary distribution, your system must have GNU `XZ Utils` to uncompress the distribution and a reasonable `tar` to unpack it.

Note

The compression algorithm changed from Gzip to XZ in MySQL Server 8.0.12; and the generic binary's file extension changed from .tar.gz to .tar.xz.

GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.

If your `tar` does not support the `xz` format then use the `xz` command to unpack the distribution and `tar` to unpack it. Replace the preceding `tar` command with the following alternative command to uncompress and extract the distribution:

```
shell> xz -dc /path/to/mysql-VERSION-OS.tar.xz | tar x
```

Next, create a symbolic link to the installation directory created by `tar`:

```
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

The `ln` command makes a symbolic link to the installation directory. This enables you to refer more easily to it as `/usr/local/mysql`. To avoid having to type the path name of client programs always when you are working with MySQL, you can add the `/usr/local/mysql/bin` directory to your `PATH` variable:

```
shell> export PATH=$PATH:/usr/local/mysql/bin
```

Perform Postinstallation Setup

The remainder of the installation process involves setting distribution ownership and access permissions, initializing the data directory, starting the MySQL server, and setting up the configuration file. For instructions, see Section 2.10, “Postinstallation Setup and Testing”.

2.3 Installing MySQL on Microsoft Windows

Important

MySQL 8.0 Server requires the Microsoft Visual C++ 2015 Redistributable Package to run on Windows platforms. Users should make sure the package has been installed on the system before installing the server. The package is available at

the Microsoft Download Center. Additionally, MySQL debug binaries require Visual Studio 2015 to be installed.

MySQL is available for Microsoft Windows 64-bit operating systems only. For supported Windows platform information, see https://www.mysql.com/support/supportedplatforms/database.html.

There are different methods to install MySQL on Microsoft Windows.

MySQL Installer Method

The simplest and recommended method is to download MySQL Installer (for Windows) and let it install and configure a specific version of MySQL Server as follows:

1. Download MySQL Installer from https://dev.mysql.com/downloads/installer/ and execute it.

   Note

   Unlike the standard MySQL Installer, the smaller "web-community" version does not bundle any MySQL applications but it will download the MySQL products you choose to install.

2. Determine the setup type to use for the initial installation of MySQL products. For example:

   - **Developer Default**: Provides a setup type that includes the selected version of MySQL Server and other MySQL tools related to MySQL development, such as MySQL Workbench.
   - **Server Only**: Provides a setup for the selected version of MySQL Server without other products.
   - **Custom**: Enables you to select any version of MySQL Server and other MySQL products.

3. Install the server instance (and products) and then begin the server configuration by first selecting one of the following levels of availability for the server instance:

   - **Standalone MySQL Server / Classic MySQL Replication (default)**

     Configures a server instance to run without high availability.

   - **InnoDB cluster**

     Provides two configuration options based on MySQL Group Replication to:

     - Configure multiple server instances in a sandbox InnoDB cluster on the local host (for testing only).
     - Create a new InnoDB cluster and configure one seed instance or add a new server instance to an existing InnoDB cluster.

4. Complete the configuration process by following the onscreen instructions. For more information about each individual step, see MySQL Server Configuration with MySQL Installer.

MySQL is now installed. If you configured MySQL as a service, then Windows will automatically start MySQL server every time you restart your system. Also, this process installs the MySQL Installer application on the local host, which you can use later to upgrade or reconfigure MySQL server.

Note

If you installed MySQL Workbench on your system, consider using it to check your new MySQL server connection. By default, the program automatically start after installing MySQL.

Additional Installation Information

It is possible to run MySQL as a standard application or as a Windows service. By using a service, you can monitor and control the operation of the server through the standard Windows service management tools. For more information, see Section 2.3.4.8, “Starting MySQL as a Windows Service”.

To accommodate the `RESTART` statement, the MySQL server forks when run as a service or standalone, to enable a monitor process to supervise the server process. In this case, you will observe two `mysqld` processes. If `RESTART` capability is not required, the server can be started with the `--no-monitor` option. See Section 13.7.8.8, “RESTART Syntax”.

Generally, you should install MySQL on Windows using an account that has administrator rights. Otherwise, you may encounter problems with certain operations such as editing the `PATH` environment variable or accessing the `Service Control Manager`. When installed, MySQL does not need to be executed using a user with Administrator privileges.

For a list of limitations on the use of MySQL on the Windows platform, see Section 2.3.7, “Windows Platform Restrictions”.

In addition to the MySQL Server package, you may need or want additional components to use MySQL with your application or development environment. These include, but are not limited to:

- To connect to the MySQL server using ODBC, you must have a Connector/ODBC driver. For more information, including installation and configuration instructions, see MySQL Connector/ODBC Developer Guide.

  Note

  MySQL Installer will install and configure Connector/ODBC for you.

- To use MySQL server with .NET applications, you must have the Connector/NET driver. For more information, including installation and configuration instructions, see MySQL Connector/NET Developer Guide.

  Note

  MySQL Installer will install and configure MySQL Connector/NET for you.

MySQL distributions for Windows can be downloaded from https://dev.mysql.com/downloads/. See Section 2.1.2, “How to Get MySQL”.

MySQL for Windows is available in several distribution formats, detailed here. Generally speaking, you should use MySQL Installer. It contains more features and MySQL products than the older MSI, is simpler to use than the compressed file, and you need no additional tools to get MySQL up and running. MySQL Installer automatically installs MySQL Server and additional MySQL products, creates an options file, starts the server, and enables you to create default user accounts. For more information on choosing a package, see Section 2.3.2, “Choosing an Installation Package”.

- A MySQL Installer distribution includes MySQL Server and additional MySQL products including MySQL Workbench, MySQL for Visual Studio, and MySQL for Excel. MySQL Installer can also be used to upgrade these products in the future (see https://dev.mysql.com/doc/mysql-compat-matrix/en/).

  For instructions on installing MySQL using MySQL Installer, see Section 2.3.3, “MySQL Installer for Windows”.

- The standard binary distribution (packaged as a compressed file) contains all of the necessary files that you unpack into your chosen location. This package contains all of the files in the full Windows MSI Installer package, but does not include an installation program.

  For instructions on installing MySQL using the compressed file, see Section 2.3.4, “Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive”.

- The source distribution format contains all the code and support files for building the executables using the Visual Studio compiler system.

  For instructions on building MySQL from source on Windows, see Section 2.9, “Installing MySQL from Source”.

MySQL on Windows Considerations

- **Large Table Support**

  If you need tables with a size larger than 4GB, install MySQL on an NTFS or newer file system. Do not forget to use `MAX_ROWS` and `AVG_ROW_LENGTH` when you create tables. See Section 13.1.20, “CREATE TABLE Syntax”.

- **MySQL and Virus Checking Software**

  Virus-scanning software such as Norton/Symantec Anti-Virus on directories containing MySQL data and temporary tables can cause issues, both in terms of the performance of MySQL and the virus-scanning software misidentifying the contents of the files as containing spam. This is due to the fingerprinting mechanism used by the virus-scanning software, and the way in which MySQL rapidly updates different files, which may be identified as a potential security risk.

  After installing MySQL Server, it is recommended that you disable virus scanning on the main directory (`datadir`) used to store your MySQL table data. There is usually a system built into the virus-scanning software to enable specific directories to be ignored.

  In addition, by default, MySQL creates temporary files in the standard Windows temporary directory. To prevent the temporary files also being scanned, configure a separate temporary directory for MySQL temporary files and add this directory to the virus scanning exclusion list. To do this, add a configuration option for the `tmpdir` parameter to your `my.ini` configuration file. For more information, see Section 2.3.4.2, “Creating an Option File”.

2.3.1 MySQL Installation Layout on Microsoft Windows

For MySQL 8.0 on Windows, the default installation directory is `C:\Program Files\MySQL\MySQL Server 8.0` for installations performed with MySQL Installer. If you use the ZIP archive method to install MySQL, you may prefer to install in `C:\mysql`. However, the layout of the subdirectories remains the same.

All of the files are located within this parent directory, using the structure shown in the following table.

Table 2.4 Default MySQL Installation Layout for Microsoft Windows

Directory	Contents of Directory	Notes
`bin`	`mysqld` server, client and utility programs	
`%PROGRAMDATA%\MySQL\MySQL Server 8.0\`	Log files, databases	The Windows system variable `%PROGRAMDATA%` defaults to `C:\ProgramData`.

Directory	Contents of Directory	Notes
`docs`	Release documentation	With MySQL Installer, use the `Modify` operation to select this optional folder.
`include`	Include (header) files	
`lib`	Libraries	
`share`	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation	

2.3.2 Choosing an Installation Package

For MySQL 8.0, there are multiple installation package formats to choose from when installing MySQL on Windows. The package formats described in this section are:

- MySQL Installer
- MySQL noinstall ZIP Archives
- MySQL Docker Images

Program Database (PDB) files (with file name extension `pdb`) provide information for debugging your MySQL installation in the event of a problem. These files are included in ZIP Archive distributions (but not MSI distributions) of MySQL.

MySQL Installer

This package has a file name similar to `mysql-installer-community-8.0.19.0.msi` or `mysql-installer-commercial-8.0.19.0.msi`, and utilizes MSIs to automatically install MySQL server and other products. MySQL Installer will download and apply updates to itself, and for each of the installed products. It also configures the installed MySQL server (including a sandbox InnoDB cluster test setup) and MySQL Router. MySQL Installer is recommended for most users.

MySQL Installer can install and manage (add, modify, upgrade, and remove) many other MySQL products, including:

- Applications – MySQL Workbench, MySQL for Visual Studio, MySQL for Excel, MySQL Shell, and MySQL Router (see https://dev.mysql.com/doc/mysql-compat-matrix/en/)
- Connectors – MySQL Connector/C, MySQL Connector/C++, MySQL Connector/NET, Connector/ODBC, MySQL Connector/Python, MySQL Connector/J, MySQL Connector/Node.js
- Documentation – MySQL Manual (PDF format), samples and examples

MySQL Installer operates on all MySQL supported versions of Windows (see https://www.mysql.com/support/supportedplatforms/database.html).

Note

Because MySQL Installer is not a native component of Microsoft Windows and depends on .NET, it will not work on minimal installation options like the Server Core version of Windows Server.

For instructions on how to install MySQL using MySQL Installer, see Section 2.3.3, “MySQL Installer for Windows”.

MySQL noinstall ZIP Archives

These packages contain the files found in the complete MySQL Server installation package, with the exception of the GUI. This format does not include an automated installer, and must be manually installed and configured.

The `noinstall` ZIP archives are split into two separate compressed files. The main package is named `mysql-VERSION-winx64.zip`. This contains the components needed to use MySQL on your system. The optional MySQL test suite, MySQL benchmark suite, and debugging binaries/information components (including PDB files) are in a separate compressed file named `mysql-VERSION-winx64-debug-test.zip`.

If you choose to install a `noinstall` ZIP archive, see Section 2.3.4, “Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive”.

MySQL Docker Images

For information on using the MySQL Docker images provided by Oracle on Windows platform, see Section 2.5.6.3, “Deploying MySQL on Windows and Other Non-Linux Platforms with Docker”.

Warning

The MySQL Docker images provided by Oracle are built specifically for Linux platforms. Other platforms are not supported, and users running the MySQL Docker images from Oracle on them are doing so at their own risk.

2.3.3 MySQL Installer for Windows

MySQL Installer is a standalone application designed to ease the complexity of installing and configuring MySQL products that run on Microsoft Windows. It supports the following MySQL products:

- MySQL Servers

  MySQL Installer can install and manage multiple, separate MySQL server instances on the same host at the same time. For example, MySQL Installer can install, configure, and upgrade a separate instance of MySQL 5.6, MySQL 5.7, and MySQL 8.0 on the same host. MySQL Installer does not permit server upgrades between major and minor version numbers, but does permit upgrades within a release series (such as 5.7.18 to 5.7.19).

  Note

  MySQL Installer cannot install both *Community* and *Commercial* (Standard Edition and Enterprise Edition) releases of MySQL server on the same host. If you require both releases on the same host, consider using the ZIP archive distribution to install one of the releases.

- MySQL Applications

  MySQL Workbench, MySQL Shell, MySQL Router, MySQL for Visual Studio, MySQL for Excel, MySQL Notifier, and MySQL Utilities.

- MySQL Connectors

  MySQL Connector/NET, MySQL Connector/Python, MySQL Connector/ODBC, MySQL Connector/J, MySQL Connector/C, and MySQL Connector/C++.

Note

To install MySQL Connector/Node.js, see https://dev.mysql.com/downloads/connector/nodejs/. Connector/Node.js does not provide an `.msi` file for use with MySQL Installer.

- Documentation and Samples

  MySQL Reference Manuals (by version) in PDF format and MySQL database samples (by version).

Installation Requirements

MySQL Installer requires Microsoft .NET Framework 4.5.2 or later. If this version is not installed on the host computer, you can download it by visiting the Microsoft website.

MySQL Installer Community Release

Download software from https://dev.mysql.com/downloads/installer/ to install the Community release of all MySQL products for Windows. Select one of the following MySQL Installer package options:

- *Web*: Contains MySQL Installer and configuration files only. The web package downloads only the MySQL products you select to install, but it requires an internet connection for each download. The size of this file is approximately 2 MB; the name of the file has the form `mysql-installer-community-web-VERSION.N.msi` where `VERSION` is the MySQL server version number such as 8.0 and `N` is the package number, which begins at 0.

- *Full or Current Bundle*: Bundles all of the MySQL products for Windows (including the MySQL server). The file size is over 300 MB, and the name has the form `mysql-installer-community-VERSION.N.msi` where `VERSION` is the MySQL Server version number such as 8.0 and `N` is the package number, which begins at 0.

MySQL Installer Commercial Release

Download software from https://edelivery.oracle.com/ to install the Commercial (Standard Edition or Enterprise Edition) release of MySQL products for Windows. The Commercial release includes all of the current and previous GA versions in the Community release (excludes development-milestone versions) and also includes the following products:

- Workbench SE/EE
- MySQL Enterprise Backup
- MySQL Enterprise Firewall

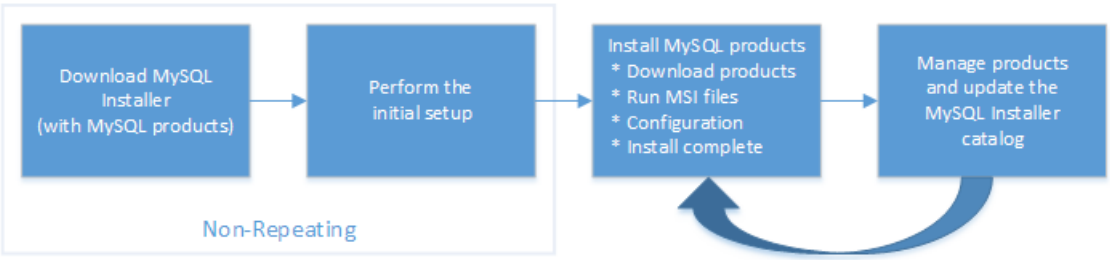
The Commercial release integrates with your My Oracle Support (MOS) account. For knowledge-base content and patches, see My Oracle Support.

2.3.3.1 MySQL Installer Initial Setup

- Choosing a Setup Type
- Path Conflicts
- Check Requirements
- MySQL Installer Configuration Files

When you download MySQL Installer for the first time, a setup wizard guides you through the initial installation of MySQL products. As the following figure shows, the initial setup is a one-time activity in the overall process. MySQL Installer detects existing MySQL products installed on the host during its initial setup and adds them to the list of products to be managed.

Figure 2.7 MySQL Installer Process Overview



MySQL Installer extracts configuration files (described later) to the hard drive of the host during the initial setup. Although MySQL Installer is a 32-bit application, it can install both 32-bit and 64-bit binaries.

The initial setup adds a link to the Start menu under the **MySQL** group. Click **Start**, **All Programs**, **MySQL**, **MySQL Installer** to open MySQL Installer.

Choosing a Setup Type

During the initial setup, you are prompted to select the MySQL products to be installed on the host. One alternative is to use a predetermined setup type that matches your setup requirements. By default, both GA and pre-release products are included in the download and installation with the **Developer Default**, **Client only**, and **Full** setup types. Select the **Only install GA products** option to restrict the product set to include GA products only when using these setup types.

Choosing one of the following setup types determines the initial installation only and does not limit your ability to install or update MySQL products for Windows later:

- **Developer Default**: Install the following products that compliment application development with MySQL:
  - MySQL Server (Installs the version that you selected when you downloaded MySQL Installer.)
  - MySQL Shell
  - MySQL Router
  - MySQL Workbench
  - MySQL for Visual Studio
  - MySQL for Excel
  - MySQL Notifier
  - MySQL Connectors (.NET / Python / ODBC / Java / C / C++)
  - MySQL Utilities
  - MySQL Documentation
  - MySQL Samples and Examples

- **Server only**: Only install the MySQL server. This setup type installs the general availability (GA) or development release server that you selected when you downloaded MySQL Installer. It uses the default installation and data paths.
- **Client only**: Only install the most recent MySQL applications and MySQL connectors. This setup type is similar to the `Developer Default` type, except that it does not include MySQL server or the client programs typically bundled with the server, such as `mysql` or `mysqladmin`.
- **Full**: Install all available MySQL products.
- **Custom** The custom setup type enables you to filter and select individual MySQL products from the MySQL Installer catalog.

  Use the `Custom` setup type to install:

  - A product or product version that is not available from the usual download locations. The catalog contains all product releases, including the other releases between pre-release (or development) and GA.
  - An instance of MySQL server using an alternative installation path, data path, or both. For instructions on how to adjust the paths, see Section 2.3.3.2, “Setting Alternative Server Paths with MySQL Installer”.
  - Two or more MySQL server versions on the same host at the same time (for example, 5.6, 5.7, and 8.0).
  - A specific combination of products and features not offered as a predetermine setup type. For example, you can install a single product, such as MySQL Workbench, instead of installing all client applications for Windows.

Path Conflicts

When the default installation or data folder (required by MySQL server) for a product to be installed already exists on the host, the wizard displays the **Path Conflict** step to identify each conflict and enable you to take action to avoid having files in the existing folder overwritten by the new installation. You see this step in the initial setup only when MySQL Installer detects a conflict.

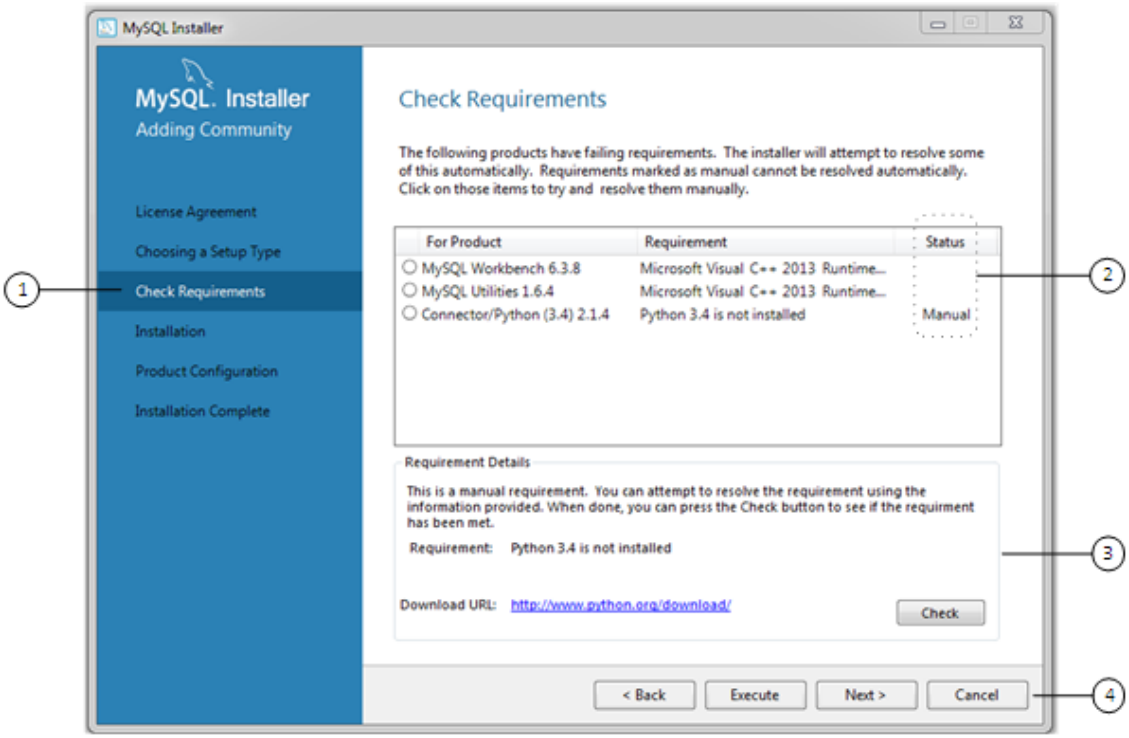
To resolve the path conflict, do one of the following:

- Select a product from the list to display the conflict options. A warning symbol indicates which path is in conflict. Use the browse button to choose a new path and then click **Next**.
- Click **Back** to choose a different setup type or product version, if applicable. The `Custom` setup type enables you to select individual product versions.
- Click **Next** to ignore the conflict and overwrite files in the existing folder.
- Delete the existing product. Click **Cancel** to stop the initial setup and close MySQL Installer. Open MySQL Installer again from the Start menu and delete the installed product from the host using the Delete operation from the dashboard.

Check Requirements

MySQL Installer uses entries in the `package-rules.xml` file to determine whether the prerequisite software for each product is installed on the host. When the requirements check fails, MySQL Installer displays the **Check Requirements** step to help you update the host. The following figure identifies and describes the key areas of this step.

Figure 2.8 Check Requirements



Description of Check Requirements Elements

1. Shows the current step in the initial setup. Steps in this list may change slightly depending on the products already installed on the host, the availability of prerequisite software, and the products to be installed on the host.
2. Lists all pending installation requirements by product and indicates the status as follows:
   - A blank space in the **Status** column means that MySQL Installer can attempt to download and install the required software for you.
   - The word *Manual* in the **Status** column means that you must satisfy the requirement manually. Select each product in the list to see its requirement details.
3. Describes the requirement in detail to assist you with each manual resolution. When possible, a download URL is provided. After you download and install the required software, click **Check** to verify that the requirement has been met.
4. Provides the following set operations to proceed:
   - **Back** – Return to the previous step. This action enables you to select a different the setup type.
   - **Execute** – Have MySQL Installer attempt to download and install the required software for all items without a manual status. Manual requirements are resolved by you and verified by clicking **Check**.
   - **Next** – Do not execute the request to apply the requirements automatically and proceed to the installation without including the products that fail the check requirements step.
   - **Cancel** – Stop the installation of MySQL products. Because MySQL Installer is already installed, the initial setup begins again when you open MySQL Installer from the Start menu and click **Add** from the dashboard. For a description of the available management operations, see Product Catalog.

MySQL Installer Configuration Files

All MySQL Installer files are located within the `C:\Program Files (x86)` and `C:\ProgramData` folders. The following table describes the files and folders that define MySQL Installer as a standalone application.

Note

Installed MySQL products are neither altered nor removed when you update or uninstall MySQL Installer.

Table 2.5 MySQL Installer Configuration Files

File or Folder	Description	Folder Hierarchy
`MySQL Installer for Windows`	This folder contains all of the files needed to run MySQL Installer and MySQLInstallerConsole.exe, a command-line program with similar functionality.	`C:\Program Files (x86)`
`Templates`	The `Templates` folder has one file for each version of MySQL server. Template files contain keys and formulas to calculate some values dynamically.	`C:\ProgramData\MySQL\MySQL Installer for Windows\Manifest`
`package-rules.xml`	This file contains the prerequisites for every product to be installed.	`C:\ProgramData\MySQL\MySQL Installer for Windows\Manifest`
`produts.xml`	The `products` file (or product catalog) contains a list of all products available for download.	`C:\ProgramData\MySQL\MySQL Installer for Windows\Manifest`
`Product Cache`	The `Product Cache` folder contains all standalone `.msi` files bundled with the full package or downloaded afterward.	`C:\ProgramData\MySQL\MySQL Installer for Windows`

2.3.3.2 Setting Alternative Server Paths with MySQL Installer

You can change the default installation path, the data path, or both when you install MySQL server. After you have installed the server, the paths cannot be altered without removing and reinstalling the server instance.

To change paths for MySQL server

1. Identify the MySQL server to change and display the **Advanced Options** link.

   a. Navigate to the **Select Products and Features** step by doing one of the following:

      i. If this is an initial setup of MySQL Installer, select the `Custom` setup type and click **Next**.

      ii. If MySQL Installer is installed already, launch it from the Start menu and then click **Add** from the dashboard.

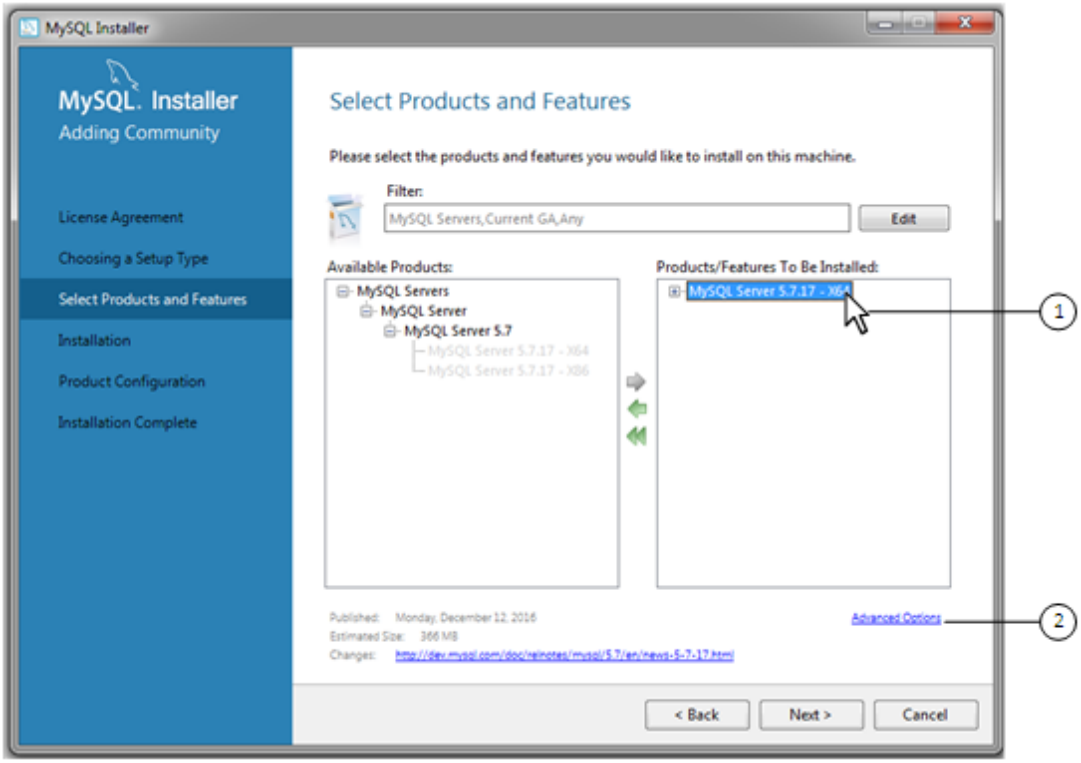
   b. Click **Edit** to apply a filter on the product list shown in **Available Products** (see Locating Products to Install.

   c. With the server instance selected, use the arrow to move the selected server to the **Products/Features To Be Installed** list.

d. Click the server to select it. When you select the server, the **Advanced Options** link appears. For details, see the figure that follows.

2. Click **Advanced Options** to open a dialog box where you can enter alternative path names. After the path names are validated, click **Next** to continue with the configuration steps.

Figure 2.9 Change MySQL Server Path



2.3.3.3 Installation Workflow with MySQL Installer

MySQL Installer provides a wizard-like tool to install and configure new MySQL products for Windows. Unlike the initial setup, which runs only once, MySQL Installer invokes the wizard each time you download or install a new product. For first-time installations, the steps of the initial setup proceed directly into the steps of the installation. For assistance with product selection, see Locating Products to Install.

Note

Full permissions are granted to the user executing MySQL Installer to all generated files, such as `my.ini`. This does not apply to files and directories for specific products, such as the MySQL server data directory in `%ProgramData%` that is owned by `SYSTEM`.

Products installed and configured on a host follow a general pattern that might require your input during the various steps. If you attempt to install a product that is incompatible with the existing MySQL server version (or a version selected for upgrade), you are alerted about the possible mismatch.

MySQL Installer loads all selected products together using the following workflow:

- **Product download.** If you installed the full (not web) MySQL Installer package, all `.msi` files were loaded to the `Product Cache` folder during the initial setup and are not downloaded again. Otherwise,

click **Execute** to begin the download. The status of each product changes from `Downloading` to `Downloaded`.

- **Product installation.** The status of each product in the list changes from `Ready to Install`, to `Installing`, and lastly to `Complete`. During the process, click **Show Details** to view the installation actions.

  If you cancel the installation at this point, the products are installed, but the server (if installed) is not yet configured. To restart the server configuration, open MySQL Installer from the Start menu and click the **Reconfigure** link next to the appropriate server in the dashboard.

- **Product configuration.** This step applies to MySQL Server, MySQL Router, and samples only. The status for each item in the list should indicate `Ready to Configure`.

  Click **Next** to start the configuration wizard for all items in the list. The configuration options presented during this step are specific to the version of database or router that you selected to install.

  Click **Execute** to begin applying the configuration options or click **Back** (repeatedly) to return to each configuration page. Click **Finish** to open the MySQL Installer dashboard.

- **Installation complete.** This step finalizes the installation for products that do not require configuration. It enables you to copy the log to a clipboard and to start certain applications, such as MySQL Workbench and MySQL Shell. Click **Finish** to open the MySQL Installer dashboard.

MySQL Server Configuration with MySQL Installer

MySQL Installer performs the initial configuration of the MySQL server. For example:

- For the MySQL 8.0 release series, a server can be configured to run as a standalone database, as a sandbox InnoDB cluster on a single host, or to create a production InnoDB cluster inside a local network (see High Availability).
- It creates the configuration file (`my.ini`) that is used to configure the MySQL server. The values written to this file are influenced by choices you make during the installation process. Some definitions are host dependent. For example, query_cache is enabled if the host has fewer than three cores.

  Note

  Query cache was deprecated in MySQL 5.7 and removed in MySQL 8.0 (and later).

- By default, a Windows service for the MySQL server is added.
- Provides default installation and data paths for MySQL server. For instructions on how to change the default paths, see Section 2.3.3.2, “Setting Alternative Server Paths with MySQL Installer”.
- It can optionally create MySQL server user accounts with configurable permissions based on general roles, such as DB Administrator, DB Designer, and Backup Admin. It optionally creates a Windows user named `MysqlSys` with limited privileges, which would then run the MySQL Server.

  User accounts may also be added and configured in MySQL Workbench.

- Checking **Show Advanced Options** enables additional **Logging Options** to be set. This includes defining custom file paths for the error log, general log, slow query log (including the configuration of seconds it requires to execute a query), and the binary log.

During the configuration process, click **Next** to proceed to the next step or **Back** to return to the previous step. Click **Execute** at the final step to apply the server configuration.

The sections that follow describe the server configuration options that apply to MySQL server on Windows. The server version you installed will determine which steps and options you can configure. Configuring MySQL server may include some or all of the steps.

High Availability

MySQL Installer enables you to install, configure, and deploy MySQL Server as a standalone instance or as a member of a highly available cluster using MySQL Group Replication. In either case, MySQL Installer restricts the installation and configuration of the server (or servers) to the local Windows host computer.

- Standalone MySQL Server / Classic MySQL Replication (default)

  Select this option to configure one MySQL instance to run as a standalone database server. This option is ideal if you intend to set up classic replication later and then to include this server in your custom solution. The remaining configuration steps are described in the sections that follow, starting with Type and Networking.

- InnoDB cluster

  Select this option to create or extend an InnoDB cluster solution that is based on MySQL Group Replication (see Section 21.1, “Introducing InnoDB Cluster”). You can configure (or reconfigure) a minimum of three server instances to perform a basic setup as a test-only sandbox cluster on a single computer or to create a production cluster inside a local network.

  InnoDB Cluster Log Verbosity Level. This configuration step includes an adjustable log that captures information during the configuration of each server instance in the production or sandbox cluster. The values are: `MINIMAL`, `MEDIUM` (default), and `DEBUG`. If the cluster configuration fails, use the **Reconfigure** action from the MySQL Installer dashboard to restart the configuration and then set the verbosity level to `DEBUG` to gather additional information during your next attempt.

  MySQL Installer provides the following configuration variations to deploy an InnoDB cluster:

  - Set Up a Local Server Cluster for Testing Only

    Select **Create a Sandbox InnoDB cluster for Testing** to enable this option. When prompted, define the number of server sandbox instances in the cluster, set a password for the `root` user, and adjust the InnoDB cluster log verbosity level as needed. For a more detailed description of the configuration, see Deploying a Sandbox InnoDB Cluster with MySQL Installer. This setup requires MySQL 5.7.17 or higher.

  - Create or Join an InnoDB cluster

    To set up a highly available InnoDB cluster using MySQL Installer, you must have a minimum of three computers on a local network. If you require a more advanced setup, use MySQL Shell to configure some or all of the server instances in the cluster. For details about how to perform a local-network cluster setup, see Setting up an InnoDB cluster with MySQL Installer. This setup requires MySQL 8.0.0 or higher.

  InnoDB cluster was designed to operate with MySQL Shell, which enables you to perform advanced cluster administration, and MySQL Router to automate the connections made between client applications and server instances. Neither MySQL Shell nor MySQL Router are required to deploy a cluster on Windows using MySQL Installer.

Deploying a Sandbox InnoDB Cluster with MySQL Installer

A sandbox deployment includes multiple server sandbox instances that run together on the same computer. Because all server instances reside on the same computer, a sandbox cluster does not meet the

requirements of a highly available solution. Instead, this deployment option simulates an environment from which you can explore the techniques associated with InnoDB cluster administration.

When you select **Create a Sandbox InnoDB cluster for Testing**, a follow-on step prompts you to select a cluster consisting of three, five, seven, or nine MySQL server instances. Unlike the other server setups provided by MySQL Installer, the sandbox deployment skips the usual server configuration steps (except Authentication Method). The resulting cluster, named `sandboxCluster`, is available on selected ports that are configured for you.

Note

MySQL Installer deletes ports 3310 to 3390 during the configuration, if those ports were set for the sandbox InnoDB cluster manually using MySQL Shell.

Each sandbox instance is configured to run as a process (not a Windows service). You must start each instance in the sandbox cluster manually after restarting the computer.

After you create the test cluster, click the **Summary** tab to view the specific ports that apply to your cluster. To modify the number of server instances within the existing cluster or to adjust the logging level, use the **Reconfigure** quick action from the MySQL Installer dashboard.

Warning

MySQL Installer deletes all existing sandbox cluster data when the cluster is reconfigured or when the server instances within the sandbox cluster are upgraded.

MySQL Installer stores all sandbox InnoDB cluster configuration entries in the `installer_config.xml` file. By default, MySQL Installer creates the sandbox instances in `%userprofile%\MySQL\mysql-sandboxes` on the local host.

Setting up InnoDB Cluster with MySQL Installer

To create a single InnoDB cluster, select **InnoDB Cluster** as the High Availability option and then select **Create a New InnoDB Cluster**. Adjust the log verbosity level (as needed), and click **Next** to configure the first server instance. This setup process involves installing and running MySQL Installer on multiple computers.

Define the first server instance (or seed) by providing the following configuration information:

- **InnoDB Cluster Name:**

  The default cluster name is `myCluster`. If you intend to configure multiple clusters, replace the default name with one that is meaningful within your solution. Alphanumeric characters, spaces, and underscore (_) characters are valid for this field. The limit is 40 characters.

- **Cluster Admin User Name:**

  The default cluster administrator name is `ic`. You can reuse the same MySQL administrative account across multiple clusters. You will be prompted for this account name (and password) later when you configure other server instances to join the cluster. The limit is 32 characters.

- **Cluster Admin Password:**

  Enter a password for the cluster administrator account (minimum length is four characters). MySQL Installer will evaluate the strength of the MySQL password as you type. Use the **Repeat Password** field to confirm the password.

- **Host Address:**

Select the host name or IP address of the local host from the list. When joining additional server instances to the cluster, you will be prompted to identify the seed instance by the host name or IP address.

- **Server ID:**

  The default value is `1`. This identifier is required to record the events of a server instance in the binary log. The ID of each server instance within a cluster must be unique; however, you can reuse the same number in a different cluster. The server ID you specify in this field also appears later in Advanced Options step. If you change the value in Advanced Option, the number is changed for the InnoDB cluster Setup too.

Click **Next** and then complete the remaining configuration steps, which are described in the sections that follow, starting with Type and Networking. After the seed instance is added and the cluster is created, it requires more instances for full tolerance. At this point, the status is `OK_NO_TOLERANCE`.

To add the second and third server instances to the cluster, you must use a separate computer inside the local network for each. Some of the configuration details of the seed instance are required to complete the join operation.

After you start MySQL Installer and install the server instance on the next computer, begin the configuration by selecting **InnoDB Cluster** as the High Availability option and then select **Add Local MySQL Server Instance to an InnoDB Cluster**. Adjust the **InnoDB Cluster Log Verbosity Level** (as needed) and then click **Next**.

Define the joining server instance by providing the following configuration information:

- **Seed Instance Address:**

  Enter the host name or IP address of the computer that hosts the seed instance.

- **Seed Instance Port:**

  The default value is `3306`, which is the port for classic MySQL. Use the same TCP port that you configured for the seed instance.

- **Cluster Admin User Name:**

  The default cluster administrator name is `ic`. If you assigned a different name when you configured the seed instance, enter the alternative cluster administrator name.

- **Cluster Admin Password:**

  Enter the password assigned to the cluster administrator account.

- **Host Address:**

  Select the host name or IP address of the local host from the list.

- **Server ID:**

  The default value is `1`. This identifier is required to record the events of a server instance in the binary log. The ID of each server instance within a cluster must be unique; however, you can reuse the same number in a different cluster. The server ID you specify in this field also appears later in Advanced Options step. If you change the value in Advanced Option, the number is changed for the InnoDB cluster Setup too.

- **Test Connection**

Use this button to verify the connection between the local server instance and the seed instance defined for the cluster. A valid connection is required to proceed.

Click **Next** and then complete the remaining configuration steps, which are described in the sections that follow, starting with Type and Networking.

With one seed instance and a second server instance in the cluster, the status is `OK_NO_TOLERANCE`. After you add the third server instance, the status is `OK`, which indicates that the cluster now is tolerant to the failure of one instance.

Type and Networking

- Server Configuration Type

  Choose the MySQL server configuration type that describes your setup. This setting defines the amount of system resources (memory) that will be assigned to your MySQL server instance.

  - **Development**: A machine that will host many other applications, and typically this is your personal workstation. This option configures MySQL to use the least amount of memory.
  - **Server**: Several other applications will be running on this machine, such as a web server. This option configures MySQL to use a medium amount of memory.
  - **Dedicated**: A machine that is dedicated to running the MySQL server. Because no other major applications will run on this server, such as a web server, this option configures MySQL to use the majority of available memory.

- Connectivity

  Connectivity options control how the connection to MySQL is made. Options include:

  - **TCP/IP**: You may enable TCP/IP Networking here as otherwise only local host connections are permitted. Also define the **Port** (for the classic MySQL protocol), **X Protocol Port** (for MySQL as a document store), and whether to open the firewall port for network access. If the port number is in use already, you will see the information icon ( ) next to the default value and **Next** is disabled until you provide a new port number.
  - **Named Pipe**: Enable and define the pipe name, similar to setting the `named_pipe` system variable. The default name is `MySQL`.
  - **Shared Memory**: Enable and then define the memory name, similar to setting the `shared_memory` system variable. The default name is `MySQL`.

- Advanced Configuration

  Check **Show Advanced and Logging Options** to set custom logging and advanced options in later steps. The Logging Options step enables you to define custom file paths for the error log, general log, slow query log (including the configuration of seconds it requires to execute a query), and the binary log. The Advanced Options step enables you to set the unique server ID required when binary logging is enabled in a replication topology.

- MySQL Enterprise Firewall (Enterprise Edition only)

  The **Enable Enterprise Firewall** check box is selected by default. For post-installation instructions, see Section 6.4.7, “MySQL Enterprise Firewall”.

Authentication Method

The **Authentication Method** step is visible only during the installation or upgrade of MySQL 8.0.4 or higher. It introduces a choice between two server-side authentication options. The MySQL user accounts that you create in the next step will use the authentication method that you select in this step.

MySQL 8.0 connectors and community drivers that use `libmysqlclient` 8.0 now support the `mysql_native_password` default authentication plugin. However, if you are unable to update your clients and applications to support this new authentication method, you can configure the MySQL server to use `mysql_native_password` for legacy authentication. For more information about the implications of this change, see caching_sha2_password as the Preferred Authentication Plugin.

If you are installing or upgrading to MySQL 8.0.4 or higher, select one of the following authentication methods:

- Use Strong Password Encryption for Authentication (RECOMMENDED)

  MySQL 8.0 supports a new authentication based on improved, stronger SHA256-based password methods. It is recommended that all new MySQL server installations use this method going forward.

  Important

  The `caching_sha2_password` authentication plugin on the server requires new versions of connectors and clients, which add support for the new MySQL 8.0 default authentication.

- Use Legacy Authentication Method (Retain MySQL 5.x Compatibility)

  Using the old MySQL 5.x legacy authentication method should be considered only in the following cases:

  - Applications cannot be updated to use MySQL 8.0 connectors and drivers.
  - Recompilation of an existing application is not feasible.
  - An updated, language-specific connector or driver is not available yet.

Accounts and Roles

- Root Account Password

  Assigning a root password is required and you will be asked for it when performing other MySQL Installer operations. Password strength is evaluated when you repeat the password in the box provided. For descriptive information regarding password requirements or status, move your mouse pointer over the information icon ( ) when it appears.

- MySQL User Accounts (Optional)

  Click **Add User** or **Edit User** to create or modify MySQL user accounts with predefined roles. Next, enter the required account credentials:

  - **User Name:** MySQL user names can be up to 32 characters long.
  - **Host:** Select `localhost` for local connections only or `<All Hosts (%)>` when remote connections to the server are required.

- **Role:** Each predefined role, such as `DB Admin`, is configured with its own set of privileges. For example, the `DB Admin` role has more privileges than the `DB Designer` role. The **Role** drop-down list contains a description of each role.
- **Password:** Password strength assessment is performed while you type the password. Passwords must be confirmed. MySQL permits a blank or empty password (considered to be insecure).

MySQL Installer Commercial Release Only: MySQL Enterprise Edition for Windows, a commercial product, also supports an authentication method that performs external authentication on Windows. Accounts authenticated by the Windows operating system can access the MySQL server without providing an additional password.

To create a new MySQL account that uses Windows authentication, enter the user name and then select a value for **Host** and **Role**. Click **Windows** authentication to enable the `authentication_windows` plugin. In the Windows Security Tokens area, enter a token for each Windows user (or group) who can authenticate with the MySQL user name. MySQL accounts can include security tokens for both local Windows users and Windows users that belong to a domain. Multiple security tokens are separated by the semicolon character (`;`) and use the following format for local and domain accounts:

- Local account

  Enter the simple Windows user name as the security token for each local user or group; for example, **`finley;jeffrey;admin`**.

- Domain account

  Use standard Windows syntax (*`domain\domainuser`*) or MySQL syntax (*`domain\\domainuser`*) to enter Windows domain users and groups.

  For domain accounts, you may need to use the credentials of an administrator within the domain if the account running MySQL Installer lacks the permissions to query the Active Directory. If this is the case, select **Validate Active Directory users with** to activate the domain administrator credentials.

Windows authentication permits you to test all of the security tokens each time you add or modify a token. Click **Test Security Tokens** to validate (or revalidate) each token. Invalid tokens generate a descriptive error message along with a red `X` icon and red token text. When all tokens resolve as valid (green text without an `X` icon), you can click **OK** to save the changes.

Windows Service

On the Windows platform, MySQL server can run as a named service managed by the operating system and be configured to start up automatically when Windows starts. Alternatively, you can configure MySQL server to run as an executable program that requires manual configuration.

- **Configure MySQL server as a Windows service** (Selected by default.)

  When the default configuration option is selected, you can also select the following:

  - **Start the MySQL Server at System Startup**

    When selected (default), the service startup type is set to Automatic; otherwise, the startup type is set to Manual.

  - **Run Windows Service as**

    When **Standard System Account** is selected (default), the service logs on as Network Service.

The **Custom User** option must have privileges to log on to Microsoft Windows as a service. The **Next** button will be disabled until this user is configured with the required privileges.

A custom user account is configured in Windows by searching for "local security policy" in the Start menu. In the Local Security Policy window, select **Local Policies**, **User Rights Assignment**, and then **Log On As A Service** to open the property dialog. Click **Add User or Group** to add the custom user and then click **OK** in each dialog to save the changes.

- Deselect the Windows Service option

Logging Options

This step is available if the **Show Advanced Configuration** check box was selected during the **Type and Networking** step. To enable this step now, click **Back** to return to the **Type and Networking** step and select the check box.

Advanced configuration options are related to the following MySQL log files:

- Error Log
- General Log
- Slow Query Log
- Bin Log

Note

The binary log is enabled by default for MySQL 5.7 and higher.

Advanced Options

This step is available if the **Show Advanced Configuration** check box was selected during the **Type and Networking** step. To enable this step now, click **Back** to return to the **Type and Networking** step and select the check box.

The advanced-configuration options include:

- **Server ID**

  Set the unique identifier used in a replication topology. If binary logging is enabled, you must specify a server ID. The default ID value depends on the server version. For more information, see the description of the `--server-id` option.

Tip

If you specified an ID for a server instance of an InnoDB cluster, then MySQL Installer adjusts the ID (shown on this page) to match the previous identifier.

- **Table Names Case**

  You can set the following options during the initial and subsequent configuration the server. For the MySQL 8.0 release series, these options apply only to the initial configuration of the server.

  - Lower Case

Sets the `lower_case_table_names` option value to 1 (default), in which table names are stored in lowercase on disk and comparisons are not case sensitive.

- Preserve Given Case

  Sets the `lower_case_table_names` option value to 2, in which table names are stored as given but compared in lowercase.

Apply Server Configuration

All configuration settings are applied to the MySQL server when you click **Execute**. Use the **Configuration Steps** tab to follow the progress of each action; the icon for each toggles from white to green (with a check mark) on success. Otherwise, the process stops and displays an error message if an individual action times out. Click the **Log** tab to view the log.

When the installation completes successfully and you click **Finish**, MySQL Installer and the installed MySQL products are added to the Microsoft Windows Start menu under the `MySQL` group. Opening MySQL Installer loads the dashboard where installed MySQL products are listed and other MySQL Installer operations are available.

MySQL Router Configuration with MySQL Installer

MySQL Installer downloads and installs a suite of tools for developing and managing business-critical applications on Windows. The suite consist of applications, connectors, documentation, and samples.

During the initial setup, choose any predetermined setup type, except `Server only`, to install the latest GA version of the tools. Use the `Custom` setup type to install an individual tool or specific version. If MySQL Installer is installed on the host already, use the **Add** operation to select and install tools from the MySQL Installer dashboard.

MySQL Router Configuration

MySQL Installer provides a configuration wizard that can bootstrap an installed instance of MySQL Router 8.0 or later to route traffic between MySQL applications and an InnoDB cluster. When configured, MySQL Router runs as a local Windows service. For detailed information about using MySQL Router with an InnoDB cluster, see Routing for MySQL InnoDB cluster.

Note

You are prompted to configure MySQL Router after the initial installation and when you reconfigure an installed router explicitly. In contrast, the upgrade operation does not require or prompt you to configure the upgraded product.

To configure MySQL Router, do the following:

1. Set up InnoDB cluster. For instructions on how to configure a sandbox InnoDB cluster on the local host using MySQL Installer, see High Availability.

   For general InnoDB cluster information, see Chapter 21, *InnoDB Cluster*.

2. Using MySQL Installer, download and install the MySQL Router application. After the installation finishes, the configuration wizard prompts you for information. Select the **Configure MySQL Router for InnoDB cluster** check box to begin the configuration and provide the following configuration values:

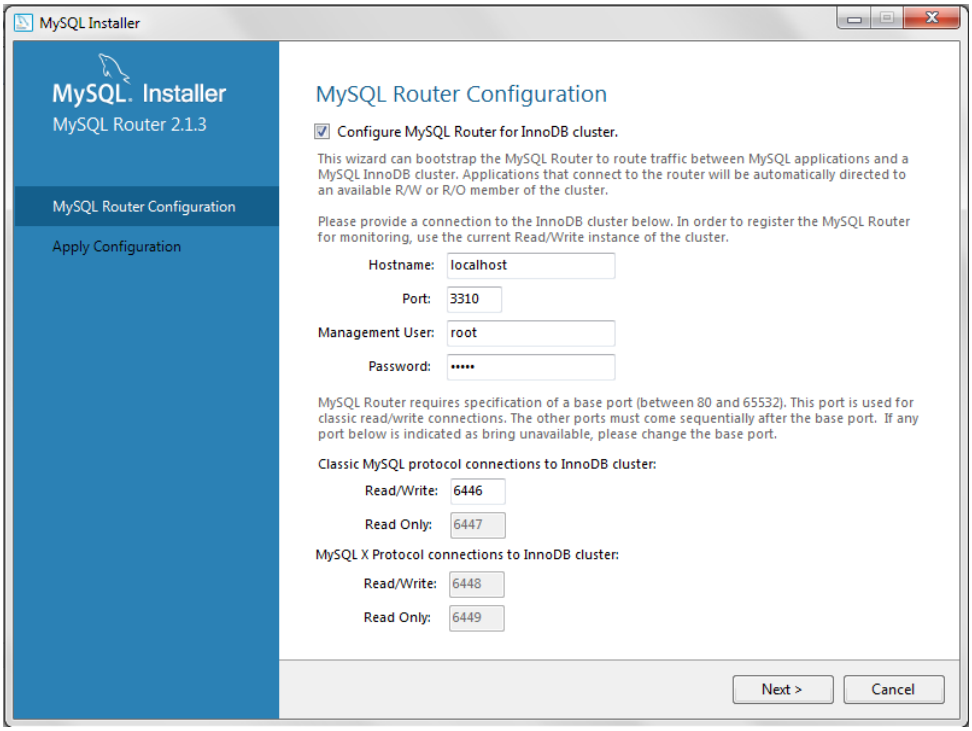
   - **Hostname:** Host name of the primary (seed) server in the InnoDB cluster (`localhost` by default).

- **Port:** The port number of the primary (seed) server in the InnoDB cluster (`3310` by default).
- **Management User:** An administrative user with root-level privileges.
- **Password:** The password for the management user.
- **Classic MySQL protocol connections to InnoDB cluster**

  Read/Write: Set the first base port number to one that is unused (between 80 and 65532) and the wizard will select the remaining ports for you.

  The figure that follows shows an example of the MySQL Router configuration page, with the first base port number specified as 6446 and the remaining ports set by the wizard as 6447, 6448, and 6449.

Figure 2.10 MySQL Router Configuration



3. Click **Next** and then **Execute** to apply the configuration. Click **Finish** to close MySQL Installer or return to the MySQL Installer dashboard.

After installing a production cluster with MySQL Router, the root account only exists in the user table as `root@localhost` (local), instead of `root@%` (remote). Regardless of where the router or client are located, even if both are located on the same host as the seed server, any connection that passes through the router is viewed by server as being remote, not local. As a result, a connection made to the server using the local host (see the example that follows), does not authenticate.

```
shell> \c root@localhost:6446
```

2.3.3.4 MySQL Installer Product Catalog and Dashboard

- Product Catalog

- MySQL Installer Dashboard
- Locating Products to Install
- Upgrading MySQL Installer

This section describes the MySQL Installer product catalog and the dashboard.

Product Catalog

The product catalog stores the complete list of released MySQL products for Microsoft Windows that are available to download from MySQL Downloads. By default, and when an Internet connection is present, MySQL Installer updates the catalog daily. You can also update the catalog manually from the dashboard (described later).

An up-to-date catalog performs the following actions:

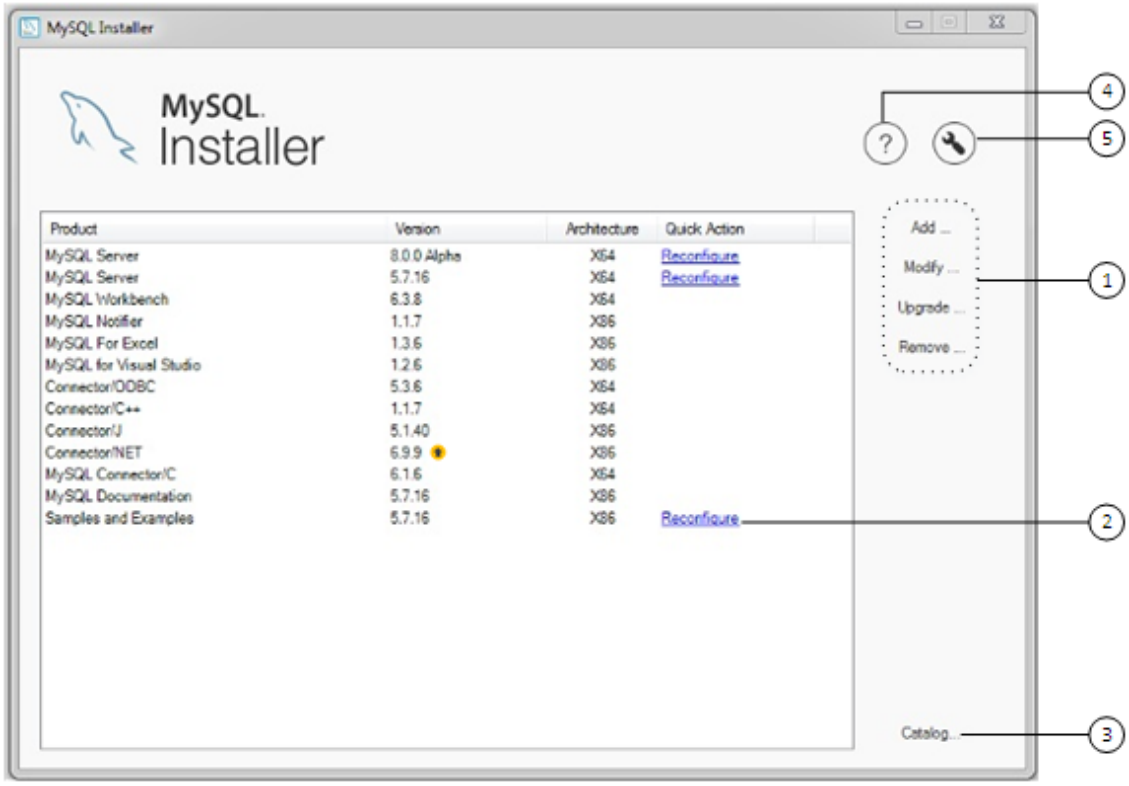
- Populates the **Available Products** pane of the Select Products and Features step. This step appears when you select:
  - The `Custom` setup type during the initial setup.
  - The **Add** operation from the dashboard.
- Identifies when product updates are available for the installed products listed in the dashboard.

The catalog includes all development releases (Pre-Release), general releases (Current GA), and minor releases (Other Releases). Products in the catalog will vary somewhat, depending on the MySQL Installer release that you download.

MySQL Installer Dashboard

The MySQL Installer dashboard is the default view that you see when you start MySQL Installer after the initial setup finishes. If you closed MySQL Installer before the setup was finished, MySQL Installer resumes the initial setup before it displays the dashboard.

Figure 2.11 MySQL Installer Dashboard Elements



Description of MySQL Installer Dashboard Elements

1. MySQL Installer dashboard operations provide a variety of actions that apply to installed products or products listed in the catalog. To initiate the following operations, first click the operation link and then select the product or products to manage:

   - **Add**: This operation opens the Select Products and Features page. From there, you can filter the product in the product catalog, select one or more products to download (as needed), and begin the installation. For hints about using the filter, see Locating Products to Install.

   - **Modify**: Use this operation to add or remove the features associated with installed products. Features that you can modify vary in complexity by product. When the **Program Shortcut** check box is selected, the product appears in the Start menu under the `MySQL` group.

   - **Upgrade**: This operation loads the Select Products to Upgrade page and populates it with all the upgrade candidates. An installed product can have more than one upgrade version and requires a current product catalog.

     Important server upgrade conditions:

     - MySQL Installer does not permit server upgrades between major release versions or minor release versions, but does permit upgrades within a release series, such as an upgrade from 5.7.18 to 5.7.19.

     - Upgrades between milestone releases (or from a milestone release to a GA release) are not supported. Significant development changes take place in milestone releases and you may encounter compatibility issues or problems starting the server.

- For upgrades to MySQL 8.0.16 server and higher, a check box enables you to skip the upgrade check and process for system tables, while checking and processing data dictionary tables normally. MySQL Installer does not prompt you with the check box when the previous server upgrade was skipped or when the server was configured as a sandbox InnoDB cluster. This behavior represents a change in how MySQL Server performs an upgrade (see Section 2.11.3, "What the MySQL Upgrade Process Upgrades") and it alters the sequence of steps that MySQL Installer applies to the configuration process.

  If you select **Skip system tables upgrade check and process. (Not recommended)**, MySQL Installer starts the upgraded server with the `--upgrade=MINIMAL` server option, which upgrades the data dictionary only. If you stop and then restart the server without the `--upgrade=MINIMAL` option, the server upgrades the system tables automatically, if needed.

  The following information appears in the **Log** tab and log file after the upgrade configuration (with system tables skipped) is complete:

```
WARNING: The system tables upgrade was skipped after upgrading MySQL Server. The
server will be started now with the --upgrade=MINIMAL option, but then each
time the server is started it will attempt to upgrade the system tables, unless
you modify the Windows service (command line) to add --upgrade=MINIMAL to bypass
the upgrade.

FOR THE BEST RESULTS: Run mysqld.exe --upgrade=FORCE on the command line to upgrade
the system tables manually.
```

To choose a new product version:

a. Click **Upgrade**. Confirm that the check box next to product name in the **Upgradeable Products** pane has a check mark. Deselect the products that you do not intend to upgrade at this time.

Note

For server milestone releases in the same release series, MySQL Installer deselects the server upgrade and displays a warning to indicate that the upgrade is not supported, identifies the risks of continuing, and provides a summary of the steps to perform a logical upgrade manually. You can reselect server upgrade at your own risk. For instructions on how to perform a logical upgrade with a milestone release, see Logical Upgrade.

b. Click a product in the list to highlight it. This action populates the **Upgradeable Versions** pane with the details of each available version for the selected product: version number, published date, and a `Changes` link to open the release notes for that version.

MySQL Installer upgrades all of the selected products in one action. Click **Show Details** to view the actions performed by MySQL Installer.

- **Remove**: This operation opens the Remove Products page and populates it with the MySQL products installed on the host. Select the MySQL products you want to remove (uninstall) and then click **Execute** to begin the removal process. During the operation, an indicator shows the number of steps that are executed as a percentage of all steps.

  To select products to remove, do one of the following:

  - Select the check box for one or more products.
  - Select the **Product** check box to select all products.

To remove a local MySQL server:

a. Determine whether the local data directory should be removed. If you retain the data directory, another server installation can reuse the data. This option is enabled by default (removes the data directory).

b. If the local server is a member of an InnoDB cluster, reconfigure the cluster as follows:

   i. Type the administrator password for the cluster affected when the local server is removed and then click **Test Connection** to verify the credentials. MySQL Installer can perform the following actions, depending on the configuration of the existing cluster:

      - If the local server is a seed instance and the number of instances in the cluster is one, dissolve the cluster when you remove the local server.
      - If the local server is a seed instance and the number of instances in the cluster is greater than one, remove the instance from the cluster or dissolve the cluster when you remove the local server.
      - If the local server is a slave instance within the cluster and the number of instances in the cluster is greater than two, remove the local instance from the cluster. (A single slave instance within a cluster reverts to a seed instance automatically.)
      - If the local server is configured as a sandbox InnoDB cluster, remove all instances created for the sandbox server installation.

   ii. When prompted, do one of the following:

      - Select an action to apply to the cluster and click **Next**.
      - Click **Next** without selecting an action. In most cases, MySQL Group Replication can manage the cluster when the local server becomes unavailable. A warning message reminds you that skipping the step may result in an inconsistent InnoDB cluster configuration.

c. Click **Execute** to begin uninstalling the local server. Note that all products that you selected to remove are also uninstalled at this time.

d. (Optional) Click the **Log** tab to display the current actions performed by MySQL Installer.

2. The **Reconfigure** link in the Quick Action column next to each installed server loads the current configuration values for the server and then cycles through all configuration steps enabling you to change the options and values. You must provide credentials with root privileges to reconfigure these items. Click the **Log** tab to show the output of each configuration step performed by MySQL Installer.

   On completion, MySQL Installer stops the server, applies the configuration changes, and restarts the server for you. For a description of each configuration option, see MySQL Server Configuration with MySQL Installer. Installed `Samples and Examples` associated with a specific MySQL server version can be also be reconfigured to apply new feature settings, if any.

3. The **Catalog** link enables you to download the latest catalog of MySQL products manually and then to integrate those product changes with MySQL Installer. The catalog-download action does not perform an upgrade of the products already installed on the host. Instead, it returns to the dashboard and displays an arrow icon in the Version column for each installed product that has a newer version. Use the **Upgrade** operation to install the newer product version.

You can also use the **Catalog** link to display the current change history of each product without downloading the new catalog. Select the **Do not update at this time** check box to view the change history only.

4. The MySQL Installer About icon ( ) shows the current version of MySQL Installer and general information about MySQL. The version number is located above the **Back** button.

   Tip

   Always include this version number when reporting a problem with MySQL Installer.

   In addition to the About MySQL information ( ), you can also select the following icons from the side panel:

   - License icon ( ) for MySQL Installer.

     This product may include third-party software, used under license. If you are using a Commercial release of MySQL Installer, the icon opens the MySQL Installer Commercial License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a Community release of MySQL Installer, the icon opens the MySQL Installer Community License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

   - Resource links icon ( ) to the latest MySQL product documentation, blogs, webinars, and more.

5. The MySQL Installer Options icon ( ) includes the following tabs:

   - **Product Catalog**: Manages the daily automatic catalog updates. By default, catalog updates are scheduled at a fixed hour. When new products or product versions are available, MySQL Installer adds them to the catalog and then displays an arrow icon ( ) next to the version number of installed products listed in the dashboard.

     Use this option to enable or disable automatic catalog updates and to reset the time of day when the MySQL Installer updates the catalog automatically. For specific settings, see the task named `ManifestUpdate` in the Windows Task Scheduler.

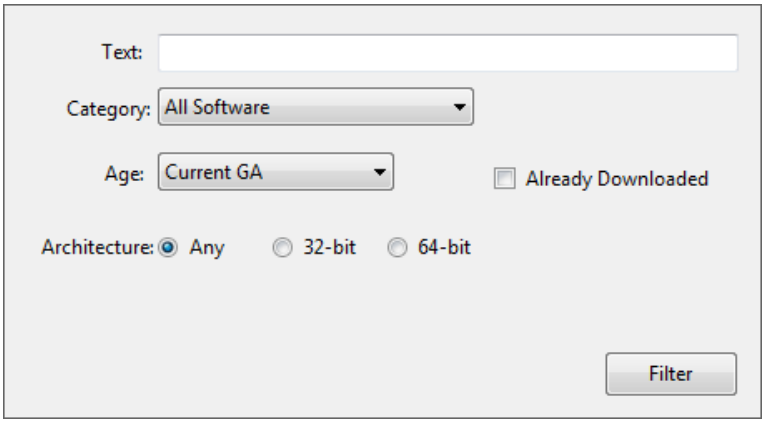
   - **Connectivity Settings**: Several operations performed by MySQL Installer require internet access. This option enables you to use a default value to validate the connection or to use a different URL, one selected from a list or added by you manually. With the **Manual** option selected, new URLs can be added and all URLs in the list can be moved or deleted. When the **Automatic** option is selected, MySQL Installer attempts to connect to each default URL in the list (in order) until a connection is made. If no connection can be made, it raises an error.

Locating Products to Install

MySQL products in the catalog are listed by category: MySQL Servers, Applications, MySQL Connectors, and Documentation. Only the latest GA versions appear in the **Available Products** pane by default. If you are looking for a pre-release or older version of a product, it may not be visible in the default list.

To change the default product list, click **Add** on the dashboard to open the Select Products and Features page, and then click **Edit** to open the filter dialog box (see the figure that follows). Modify the product values and then click **Filter**.

Figure 2.12 Filter Available Products



Reset one or more of the following values to filter the list of available products:

- Text: Filter by text.
- Category: All Software (default), MySQL Servers, Applications, MySQL Connectors, or Documentation (for samples and documentation).
- Maturity: Current Bundle (appears initially with the full package only), Pre-Release, Current GA, or Other Releases.

Note

The Commercial release of MySQL Installer does not display any MySQL products when you select the Pre-Release age filter. Products in development are available from the Community release of MySQL Installer only.

- Already Downloaded (the check box is deselected by default).
- Architecture: Any (default), 32-bit, or 64-bit.

Upgrading MySQL Installer

MySQL Installer remains installed on your computer, and like other software, MySQL Installer can be upgraded from the previous version. In some cases, other MySQL software may require that you upgrade MySQL Installer for compatibility. This section describes how to identify the current version of MySQL Installer and how to upgrade MySQL Installer manually.

To locate the installed version of MySQL Installer:

1. Start MySQL Installer from the search menu. The MySQL Installer dashboard opens.
2. Click the MySQL Installer About icon ( ). The version number is located above the **Back** button.

To initiate an on-demand upgrade of MySQL Installer:

1. Connect the computer with MySQL Installer installed to the internet.

2. Start MySQL Installer from the search menu. The MySQL Installer dashboard opens.
3. Click **Catalog** on the bottom of the dashboard to open the Update Catalog window.
4. Click **Execute** to begin the process. If the installed version of MySQL Installer can be upgraded, you will be prompted to start the upgrade.
5. Click **Next** to review all changes to the catalog and then click **Finish** to return to the dashboard.
6. Verify the (new) installed version of MySQL Installer (see the previous procedure).

2.3.3.5 MySQLInstallerConsole Reference

`MySQLInstallerConsole.exe` provides command-line functionality that is similar to MySQL Installer. It is installed when MySQL Installer is initially executed and then available within the `MySQL Installer` directory. Typically, that is in `C:\Program Files (x86)\MySQL\MySQL Installer\`, and the console must be executed with administrative privileges.

To use, invoke the command prompt with administrative privileges by choosing **Start**, **Accessories**, then right-click on **Command Prompt** and choose `Run as administrator`. And from the command line, optionally change the directory to where `MySQLInstallerConsole.exe` is located:

```
C:\> cd Program Files (x86)\MySQL\MySQL Installer for Windows
C:\Program Files (x86)\MySQL\MySQL Installer for Windows> MySQLInstallerConsole.exe help
=================== Start Initialization ===================
MySQL Installer is running in Community mode

Attempting to update manifest.
Initializing product requirements
Loading product catalog
Checking for product catalog snippets
Checking for product packages in the bundle
Categorizing product catalog
Finding all installed packages.
Your product catalog was last updated at 11/1/2016 4:10:38 PM
=================== End Initialization ===================

The following commands are available:

Configure - Configures one or more of your installed programs.
Help      - Provides list of available commands.
Install   - Install and configure one or more available MySQL programs.
List      - Provides an interactive way to list all products available.
Modify    - Modifies the features of installed products.
Remove    - Removes one or more products from your system.
Status    - Shows the status of all installed products.
Update    - Update the current product catalog.
Upgrade   - Upgrades one or more of your installed programs.
```

`MySQLInstallerConsole.exe` supports the following commands:

Note

Configuration block values that contain a colon (":") must be wrapped in double quotes. For example, installdir="C:\MySQL\MySQL Server 8.0".

- `configure` *`[product1]:[setting]=[value]; [product2]:[setting]=[value]; [...]`*

  Configure one or more MySQL products on your system. Multiple setting=value pairs can be configured for each product.

Switches include:

- `-showsettings` : Displays the available options for the selected product, by passing in the product name after `-showsettings`.
- `-silent` : Disable confirmation prompts.

```
C:\> MySQLInstallerConsole configure -showsettings server
C:\> MySQLInstallerConsole configure server:port=3307
```

- `help` *`[command]`*

Displays a help message with usage examples, and then exits. Pass in an additional command to receive help specific to that command.

```
C:\> MySQLInstallerConsole help
C:\> MySQLInstallerConsole help install
```

- `install` *`[product]:[features]:[config block]:[config block]:[config block];[...]`*

Install one or more MySQL products on your system. If pre-release products are available, both GA and pre-release products are installed when the value of the `-type` switch is `Developer`, `Client`, or `Full`. Use the `-only_ga_products` switch to restrict the product set to GA products only when using these setup types.

Switches and syntax options include:

- `-only_ga_products` : Restricts the product set to include GA products only.
- `-type=[SetupType]` : Installs a predefined set of software. The "SetupType" can be one of the following:

Note

Non-custom setup types can only be chosen if no other MySQL products are installed.

  - **Developer**: Installs a complete development environment.
  - **Server**: Installs a single MySQL server
  - **Client**: Installs client programs and libraries
  - **Full**: Installs everything
  - **Custom**: Installs user selected products. This is the default option.
- `-showsettings` : Displays the available options for the selected product, by passing in the product name after `-showsettings`.
- `-silent` : Disable confirmation prompts.
- *`[config block]`*: One or more configuration blocks can be specified. Each configuration block is a semicolon separated list of key value pairs. A block can include either a "config" or "user" type key, where "config" is the default type if one is not defined.

Configuration block values that contain a colon character (`:`) must be wrapped in double quotes. For example, `installdir="C:\MySQL\MySQL Server 8.0"`.

Only one "config" type block can be defined per product. A "user" block should be defined for each user that should be created during the product's installation.

Note

Adding users is not supported when a product is being reconfigured.

- *`[feature]`*: The feature block is a semicolon separated list of features, or an asterisk character (`*`) to select all features.

```
C:\> MySQLInstallerConsole install server;5.6.25:*:port=3307;serverid=2:type=user;username=foo;password=bar;
C:\> MySQLInstallerConsole install server;5.6.25;x64 -silent
```

An example that passes in additional configuration blocks, separated by `^` to fit:

```
C:\> MySQLInstallerConsole install server;5.6.25;x64:*:type=config;openfirewall=true; ^
         generallog=true;binlog=true;serverid=3306;enable_tcpip=true;port=3306;rootpasswd=pass; ^
         installdir="C:\MySQL\MySQL Server 5.6":type=user;datadir="C:\MySQL\data";username=foo;password=bar
```

- `list`

Lists an interactive console where all of the available MySQL products can be searched. Execute `MySQLInstallerConsole list` to launch the console, and enter in a substring to search.

```
C:\> MySQLInstallerConsole list
```

- `modify` *`[product1:-removelist|+addlist] [product2:-removelist|+addlist] [...]`*

Modifies or displays features of a previously installed MySQL product.

- `-silent` : Disable confirmation prompts.

```
C:\> MySQLInstallerConsole modify server
C:\> MySQLInstallerConsole modify server:+documentation
C:\> MySQLInstallerConsole modify server:-debug
```

- `remove` *`[product1] [product2] [...]`*

Removes one ore more products from your system.

- `*` : Pass in `*` to remove all of the MySQL products.
- `-continue` : Continue the operation even if an error occurs.
- `-silent` : Disable confirmation prompts.

```
C:\> MySQLInstallerConsole remove *
C:\> MySQLInstallerConsole remove server
```

- `status`

Provides a quick overview of the MySQL products that are installed on the system. Information includes product name and version, architecture, date installed, and install location.

```
C:\> MySQLInstallerConsole status
```

- `update`

  Downloads the latest MySQL product catalog to your system. On success, the download catalog will be applied the next time either MySQLInstaller or MySQLInstallerConsole is executed.

  ```
  C:\> MySQLInstallerConsole update
  ```

  Note

  The **Automatic Catalog Update** GUI option executes this command from the Windows Task Scheduler.

- `upgrade [product1:version] [product2:version] [...]`

  Upgrades one or more products on your system. Syntax options include:

  - `*` : Pass in `*` to upgrade all products to the latest version, or pass in specific products.
  - `!` : Pass in `!` as a version number to upgrade the MySQL product to its latest version.
  - `-silent` : Disable confirmation prompts.

  ```
  C:\> MySQLInstallerConsole upgrade *
  C:\> MySQLInstallerConsole upgrade workbench:6.3.5
  C:\> MySQLInstallerConsole upgrade workbench:!
  C:\> MySQLInstallerConsole upgrade workbench:6.3.5 excel:1.3.2
  ```

2.3.4 Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive

Users who are installing from the `noinstall` package can use the instructions in this section to manually install MySQL. The process for installing MySQL from a ZIP Archive package is as follows:

1. Extract the main archive to the desired install directory

   Optional: also extract the debug-test archive if you plan to execute the MySQL benchmark and test suite

2. Create an option file
3. Choose a MySQL server type
4. Initialize MySQL
5. Start the MySQL server
6. Secure the default user accounts

This process is described in the sections that follow.

2.3.4.1 Extracting the Install Archive

To install MySQL manually, do the following:

1. If you are upgrading from a previous version please refer to Section 2.11.10, “Upgrading MySQL on Windows”, before beginning the upgrade process.
2. Make sure that you are logged in as a user with administrator privileges.
3. Choose an installation location. Traditionally, the MySQL server is installed in C:\mysql. If you do not install MySQL at C:\mysql, you must specify the path to the install directory during startup or in an option file. See Section 2.3.4.2, “Creating an Option File”.

Note

The MySQL Installer installs MySQL under C:\Program Files\MySQL.

4. Extract the install archive to the chosen installation location using your preferred file-compression tool. Some tools may extract the archive to a folder within your chosen installation location. If this occurs, you can move the contents of the subfolder into the chosen installation location.

2.3.4.2 Creating an Option File

If you need to specify startup options when you run the server, you can indicate them on the command line or place them in an option file. For options that are used every time the server starts, you may find it most convenient to use an option file to specify your MySQL configuration. This is particularly true under the following circumstances:

- The installation or data directory locations are different from the default locations (C:\Program Files\MySQL\MySQL Server 8.0 and C:\Program Files\MySQL\MySQL Server 8.0\data).
- You need to tune the server settings, such as memory, cache, or InnoDB configuration information.

When the MySQL server starts on Windows, it looks for option files in several locations, such as the Windows directory, C:\, and the MySQL installation directory (for the full list of locations, see Section 4.2.2.2, “Using Option Files”). The Windows directory typically is named something like C:\WINDOWS. You can determine its exact location from the value of the WINDIR environment variable using the following command:

```
C:\> echo %WINDIR%
```

MySQL looks for options in each location first in the my.ini file, and then in the my.cnf file. However, to avoid confusion, it is best if you use only one file. If your PC uses a boot loader where C: is not the boot drive, your only option is to use the my.ini file. Whichever option file you use, it must be a plain text file.

Note

When using the MySQL Installer to install MySQL Server, it will create the my.ini at the default location, and the user executing MySQL Installer is granted full permissions to this new my.ini file.

In other words, be sure that the MySQL Server user has permission to read the my.ini file.

You can also make use of the example option files included with your MySQL distribution; see Section 5.1.2, “Server Configuration Defaults”.

An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in E:\mysql and the data directory is in E:\mydata\data, you can create an option file containing a [mysqld] section to specify values for the basedir and datadir options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=E:/mydata/data
```

Microsoft Windows path names are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
# set basedir to your installation path
basedir=E:\\mysql
# set datadir to the location of your data directory
datadir=E:\\mydata\\data
```

The rules for use of backslash in option file values are given in Section 4.2.2.2, “Using Option Files”.

The ZIP archive does not include a `data` directory. To initialize a MySQL installation by creating the data directory and populating the tables in the mysql system database, initialize MySQL using either `--initialize` or `--initialize-insecure`. For additional information, see Section 2.10.1, “Initializing the Data Directory”.

If you would like to use a data directory in a different location, you should copy the entire contents of the `data` directory to the new location. For example, if you want to use `E:\mydata` as the data directory instead, you must do two things:

1. Move the entire `data` directory and all of its contents from the default location (for example `C:\Program Files\MySQL\MySQL Server 8.0\data`) to `E:\mydata`.
2. Use a `--datadir` option to specify the new data directory location each time you start the server.

2.3.4.3 Selecting a MySQL Server Type

The following table shows the available servers for Windows in MySQL 8.0.

Binary	Description
`mysqld`	Optimized binary with named-pipe support
`mysqld-debug`	Like `mysqld`, but compiled with full debugging and automatic memory allocation checking

All of the preceding binaries are optimized for modern Intel processors, but should work on any Intel i386-class or higher processor.

Each of the servers in a distribution support the same set of storage engines. The `SHOW ENGINES` statement displays which engines a given server supports.

All Windows MySQL 8.0 servers have support for symbolic linking of database directories.

MySQL supports TCP/IP on all Windows platforms. MySQL servers on Windows also support named pipes, if you start the server with the `named_pipe` system variable enabled. It is necessary to enable this variable explicitly because some users have experienced problems with shutting down the MySQL server when named pipes were used. The default is to use TCP/IP regardless of platform because named pipes are slower than TCP/IP in many Windows configurations.

2.3.4.4 Initializing the Data Directory

If you installed MySQL using the `noinstall` package, no data directory is included. To initialize the data directory, use the instructions at Section 2.10.1, “Initializing the Data Directory”.

2.3.4.5 Starting the Server for the First Time

This section gives a general overview of starting the MySQL server. The following sections provide more specific information for starting the MySQL server from the command line or as a Windows service.

The information here applies primarily if you installed MySQL using the noinstall version, or if you wish to configure and test MySQL manually rather than with the MySQL Installer.

The examples in these sections assume that MySQL is installed under the default location of C:\Program Files\MySQL\MySQL Server 8.0. Adjust the path names shown in the examples if you have MySQL installed in a different location.

Clients have two options. They can use TCP/IP, or they can use a named pipe if the server supports named-pipe connections.

MySQL for Windows also supports shared-memory connections if the server is started with the shared_memory system variable enabled. Clients can connect through shared memory by using the --protocol=MEMORY option.

For information about which server binary to run, see Section 2.3.4.3, “Selecting a MySQL Server Type”.

Testing is best done from a command prompt in a console window (or “DOS window”). In this way you can have the server display status messages in the window where they are easy to see. If something is wrong with your configuration, these messages make it easier for you to identify and fix any problems.

Note

The database must be initialized before MySQL can be started. For additional information about the initialization process, see Section 2.10.1, “Initializing the Data Directory”.

To start the server, enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --console
```

For a server that includes InnoDB support, you should see the messages similar to those following as it starts (the path names and sizes may differ):

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

When the server finishes its startup sequence, you should see something like this, which indicates that the server is ready to service client connections:

```
mysqld: ready for connections
```

```
Version: '8.0.19'  socket: ''  port: 3306
```

The server continues to write to the console any further diagnostic output it produces. You can open a new console window in which to run client programs.

If you omit the `--console` option, the server writes diagnostic output to the error log in the data directory (`C:\Program Files\MySQL\MySQL Server 8.0\data` by default). The error log is the file with the `.err` extension, and may be set using the `--log-error` option.

Note

The initial `root` account in the MySQL grant tables has no password. After starting the server, you should set up a password for it using the instructions in Section 2.10.4, “Securing the Initial MySQL Account”.

2.3.4.6 Starting MySQL from the Windows Command Line

The MySQL server can be started manually from the command line. This can be done on any version of Windows.

To start the `mysqld` server from the command line, you should start a console window (or “DOS window”) and enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld"
```

The path to `mysqld` may vary depending on the install location of MySQL on your system.

You can stop the MySQL server by executing this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqladmin" -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

This command invokes the MySQL administrative utility `mysqladmin` to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system.

Note

Users in the MySQL grant system are wholly independent from any operating system users under Microsoft Windows.

If `mysqld` doesn't start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. By default, the error log is located in the `C:\Program Files\MySQL\MySQL Server 8.0\data` directory. It is the file with a suffix of `.err`, or may be specified by passing in the `--log-error` option. Alternatively, you can try to start the server with the `--console` option; in this case, the server may display some useful information on the screen that will help solve the problem.

The last option is to start `mysqld` with the `--standalone` and `--debug` options. In this case, `mysqld` writes a log file `C:\mysqld.trace` that should contain the reason why `mysqld` doesn't start. See Section 29.5.4, “The DBUG Package”.

Use `mysqld --verbose --help` to display all the options that `mysqld` supports.

2.3.4.7 Customizing the PATH for MySQL Tools

Warning

You must exercise great care when editing your system PATH by hand; accidental deletion or modification of any portion of the existing PATH value can leave you with a malfunctioning or even unusable system.

To make it easier to invoke MySQL programs, you can add the path name of the MySQL bin directory to your Windows system PATH environment variable:

- On the Windows desktop, right-click the **My Computer** icon, and select **Properties**.
- Next select the **Advanced** tab from the **System Properties** menu that appears, and click the **Environment Variables** button.
- Under **System Variables**, select **Path**, and then click the **Edit** button. The **Edit System Variable** dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **Variable Value**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete path name of your MySQL bin directory (for example, C:\Program Files\MySQL\MySQL Server 8.0\bin)

  Note

  There must be a semicolon separating this path from any values present in this field.

  Dismiss this dialogue, and each dialogue in turn, by clicking **OK** until all of the dialogues that were opened have been dismissed. The new PATH value should now be available to any new command shell you open, allowing you to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the mysql client, and all MySQL command-line utilities such as mysqladmin and mysqldump.

You should not add the MySQL bin directory to your Windows PATH if you are running multiple MySQL servers on the same machine.

2.3.4.8 Starting MySQL as a Windows Service

On Windows, the recommended way to run MySQL is to install it as a Windows service, so that MySQL starts and stops automatically when Windows starts and stops. A MySQL server installed as a service can also be controlled from the command line using NET commands, or with the graphical Services utility. Generally, to install MySQL as a Windows service you should be logged in using an account that has administrator rights.

The Services utility (the Windows Service Control Manager) can be found in the Windows Control Panel. To avoid conflicts, it is advisable to close the Services utility while performing server installation or removal operations from the command line.

Installing the service

Before installing MySQL as a Windows service, you should first stop the current server if it is running by using the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqladmin"
```

```
          -u root shutdown
```

Note

If the MySQL root user account has a password, you need to invoke mysqladmin with the -p option and supply the password when prompted.

This command invokes the MySQL administrative utility mysqladmin to connect to the server and tell it to shut down. The command connects as the MySQL root user, which is the default administrative account in the MySQL grant system.

Note

Users in the MySQL grant system are wholly independent from any operating system users under Windows.

Install the server as a service using this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --install
```

The service-installation command does not start the server. Instructions for that are given later in this section.

To make it easier to invoke MySQL programs, you can add the path name of the MySQL bin directory to your Windows system PATH environment variable:

- On the Windows desktop, right-click the **My Computer** icon, and select **Properties**.
- Next select the **Advanced** tab from the **System Properties** menu that appears, and click the **Environment Variables** button.
- Under **System Variables**, select **Path**, and then click the **Edit** button. The **Edit System Variable** dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **Variable Value**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete path name of your MySQL bin directory (for example, C:\Program Files\MySQL\MySQL Server 8.0\bin), and there should be a semicolon separating this path from any values present in this field. Dismiss this dialogue, and each dialogue in turn, by clicking **OK** until all of the dialogues that were opened have been dismissed. You should now be able to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the mysql client, and all MySQL command-line utilities such as mysqladmin and mysqldump.

  You should not add the MySQL bin directory to your Windows PATH if you are running multiple MySQL servers on the same machine.

Warning

You must exercise great care when editing your system PATH by hand; accidental deletion or modification of any portion of the existing PATH value can leave you with a malfunctioning or even unusable system.

The following additional arguments can be used when installing the service:

- You can specify a service name immediately following the --install option. The default service name is MySQL.

- If a service name is given, it can be followed by a single option. By convention, this should be `--defaults-file=file_name` to specify the name of an option file from which the server should read options when it starts.

  The use of a single option other than `--defaults-file` is possible but discouraged. `--defaults-file` is more flexible because it enables you to specify multiple startup options for the server by placing them in the named option file.

- You can also specify a `--local-service` option following the service name. This causes the server to run using the `LocalService` Windows account that has limited system privileges. If both `--defaults-file` and `--local-service` are given following the service name, they can be in any order.

For a MySQL server that is installed as a Windows service, the following rules determine the service name and option files that the server uses:

- If the service-installation command specifies no service name or the default service name (`MySQL`) following the `--install` option, the server uses the service name of `MySQL` and reads options from the `[mysqld]` group in the standard option files.
- If the service-installation command specifies a service name other than `MySQL` following the `--install` option, the server uses that service name. It reads options from the `[mysqld]` group and the group that has the same name as the service in the standard option files. This enables you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group with the service name for use by the server installed with that service name.
- If the service-installation command specifies a `--defaults-file` option after the service name, the server reads options the same way as described in the previous item, except that it reads options only from the named file and ignores the standard option files.

As a more complex example, consider the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld"
          --install MySQL --defaults-file=C:\my-opts.cnf
```

Here, the default service name (`MySQL`) is given after the `--install` option. If no `--defaults-file` option had been given, this command would have the effect of causing the server to read the `[mysqld]` group from the standard option files. However, because the `--defaults-file` option is present, the server reads options from the `[mysqld]` option group, and only from the named file.

Note

On Windows, if the server is started with the `--defaults-file` and `--install` options, `--install` must be first. Otherwise, `mysqld.exe` will attempt to start the MySQL server.

You can also specify options as Start parameters in the Windows `Services` utility before you start the MySQL service.

Finally, before trying to start the MySQL service, make sure the user variables `%TEMP%` and `%TMP%` (and also `%TMPDIR%`, if it has ever been set) for the operating system user who is to run the service are pointing to a folder to which the user has write access. The default user for running the MySQL service is `LocalSystem`, and the default value for its `%TEMP%` and `%TMP%` is `C:\Windows\Temp`, a directory `LocalSystem` has write access to by default. However, if there are any changes to that default setup (for example, changes to the user who runs the service or to the mentioned user variables, or the `--tmpdir` option has been used to put the temporary directory somewhere else), the MySQL service might fail to run because write access to the temporary directory has not been granted to the proper user.

Starting the service

After a MySQL server instance has been installed as a service, Windows starts the service automatically whenever Windows starts. The service also can be started immediately from the Services utility, or by using an sc start *mysqld_service_name* or NET START *mysqld_service_name* command. SC and NET commands are not case-sensitive.

When run as a service, mysqld has no access to a console window, so no messages can be seen there. If mysqld does not start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the MySQL data directory (for example, C:\Program Files\MySQL\MySQL Server 8.0\data). It is the file with a suffix of .err.

When a MySQL server has been installed as a service, and the service is running, Windows stops the service automatically when Windows shuts down. The server also can be stopped manually using the Services utility, the sc stop *mysqld_service_name* command, the NET START *mysqld_service_name* command, or the mysqladmin shutdown command.

You also have the choice of installing the server as a manual service if you do not wish for the service to be started automatically during the boot process. To do this, use the --install-manual option rather than the --install option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --install-manual
```

Removing the service

To remove a server that is installed as a service, first stop it if it is running by executing SC STOP *mysqld_service_name* or NET STOP *mysqld_service_name*. Then use SC DELETE *mysqld_service_name* to remove it:

```
C:\> SC DELETE mysql
```

Alternatively, use the mysqld --remove option to remove the service.

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --remove
```

If mysqld is not running as a service, you can start it from the command line. For instructions, see Section 2.3.4.6, “Starting MySQL from the Windows Command Line”.

If you encounter difficulties during installation, see Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”.

For more information about stopping or removing a Windows service, see Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”.

2.3.4.9 Testing The MySQL Installation

You can test whether the MySQL server is working by executing any of the following commands:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqlshow"
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqlshow" -u root mysql
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqladmin" version status proc
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql" test
```

If mysqld is slow to respond to TCP/IP connections from client programs, there is probably a problem with your DNS. In this case, start mysqld with the --skip-name-resolve option and use only localhost

and IP addresses in the `Host` column of the MySQL grant tables. (Be sure that an account exists that specifies an IP address or you may not be able to connect.)

You can force a MySQL client to use a named-pipe connection rather than TCP/IP by specifying the `--pipe` or `--protocol=PIPE` option, or by specifying `.` (period) as the host name. Use the `--socket` option to specify the name of the pipe if you do not want to use the default pipe name.

If you have set a password for the `root` account, deleted the anonymous account, or created a new user account, then to connect to the MySQL server you must use the appropriate `-u` and `-p` options with the commands shown previously. See Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

For more information about `mysqlshow`, see Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”.

2.3.5 Troubleshooting a Microsoft Windows MySQL Server Installation

When installing and running MySQL for the first time, you may encounter certain errors that prevent the MySQL server from starting. This section helps you diagnose and correct some of these errors.

Your first resource when troubleshooting server issues is the error log. The MySQL server uses the error log to record information relevant to the error that prevents the server from starting. The error log is located in the data directory specified in your `my.ini` file. The default data directory location is `C:\Program Files\MySQL\MySQL Server 8.0\data`, or `C:\ProgramData\Mysql` on Windows 7 and Windows Server 2008. The `C:\ProgramData` directory is hidden by default. You need to change your folder options to see the directory and contents. For more information on the error log and understanding the content, see Section 5.4.2, “The Error Log”.

For information regarding possible errors, also consult the console messages displayed when the MySQL service is starting. Use the `SC START` *`mysqld_service_name`* or `NET START` *`mysqld_service_name`* command from the command line after installing `mysqld` as a service to see any error messages regarding the starting of the MySQL server as a service. See Section 2.3.4.8, “Starting MySQL as a Windows Service”.

The following examples show other common error messages you might encounter when installing MySQL and starting the server for the first time:

- If the MySQL server cannot find the `mysql` privileges database or other critical files, it displays these messages:

```
System error 1067 has occurred.
Fatal error: Can't open and lock privilege tables:
Table 'mysql.user' doesn't exist
```

  These messages often occur when the MySQL base or data directories are installed in different locations than the default locations (`C:\Program Files\MySQL\MySQL Server 8.0` and `C:\Program Files\MySQL\MySQL Server 8.0\data`, respectively).

  This situation can occur when MySQL is upgraded and installed to a new location, but the configuration file is not updated to reflect the new location. In addition, old and new configuration files might conflict. Be sure to delete or rename any old configuration files when upgrading MySQL.

  If you have installed MySQL to a directory other than `C:\Program Files\MySQL\MySQL Server 8.0`, ensure that the MySQL server is aware of this through the use of a configuration (`my.ini`) file. Put the `my.ini` file in your Windows directory, typically `C:\WINDOWS`. To determine its exact location from the value of the `WINDIR` environment variable, issue the following command from the command prompt:

```
C:\> echo %WINDIR%
```

You can create or modify an option file with any text editor, such as Notepad. For example, if MySQL is installed in `E:\mysql` and the data directory is `D:\MySQLdata`, you can create the option file and set up a `[mysqld]` section to specify values for the `basedir` and `datadir` options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=D:/MySQLdata
```

Microsoft Windows path names are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
# set basedir to your installation path
basedir=C:\\Program Files\\MySQL\\MySQL Server 8.0
# set datadir to the location of your data directory
datadir=D:\\MySQLdata
```

The rules for use of backslash in option file values are given in Section 4.2.2.2, “Using Option Files”.

If you change the `datadir` value in your MySQL configuration file, you must move the contents of the existing MySQL data directory before restarting the MySQL server.

See Section 2.3.4.2, “Creating an Option File”.

- If you reinstall or upgrade MySQL without first stopping and removing the existing MySQL service and install MySQL using the MySQL Installer, you might see this error:

```
Error: Cannot create Windows service for MySql. Error: 0
```

This occurs when the Configuration Wizard tries to install the service and finds an existing service with the same name.

One solution to this problem is to choose a service name other than `mysql` when using the configuration wizard. This enables the new service to be installed correctly, but leaves the outdated service in place. Although this is harmless, it is best to remove old services that are no longer in use.

To permanently remove the old `mysql` service, execute the following command as a user with administrative privileges, on the command line:

```
C:\> SC DELETE mysql
[SC] DeleteService SUCCESS
```

If the `SC` utility is not available for your version of Windows, download the `delsrv` utility from http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp and use the `delsrv mysql` syntax.

2.3.6 Windows Postinstallation Procedures

GUI tools exist that perform most of the tasks described in this section, including:

- MySQL Installer: Used to install and upgrade MySQL products.

- MySQL Workbench: Manages the MySQL server and edits SQL statements.

If necessary, initialize the data directory and create the MySQL grant tables. Windows installation operations performed by MySQL Installer initialize the data directory automatically. For installation from a ZIP Archive package, initialize the data directory as described at Section 2.10.1, “Initializing the Data Directory”.

Regarding passwords, if you installed MySQL using the MySQL Installer, you may have already assigned a password to the initial `root` account. (See Section 2.3.3, “MySQL Installer for Windows”.) Otherwise, use the password-assignment procedure given in Section 2.10.4, “Securing the Initial MySQL Account”.

Before assigning a password, you might want to try running some client programs to make sure that you can connect to the server and that it is operating properly. Make sure that the server is running (see Section 2.3.4.5, “Starting the Server for the First Time”). You can also set up a MySQL service that runs automatically when Windows starts (see Section 2.3.4.8, “Starting MySQL as a Windows Service”).

These instructions assume that your current location is the MySQL installation directory and that it has a `bin` subdirectory containing the MySQL programs used here. If that is not true, adjust the command path names accordingly.

If you installed MySQL using MySQL Installer (see Section 2.3.3, “MySQL Installer for Windows”), the default installation directory is `C:\Program Files\MySQL\MySQL Server 8.0`:

```
C:\> cd "C:\Program Files\MySQL\MySQL Server 8.0"
```

A common installation location for installation from a ZIP archive is `C:\mysql`:

```
C:\> cd C:\mysql
```

Alternatively, add the `bin` directory to your `PATH` environment variable setting. That enables your command interpreter to find MySQL programs properly, so that you can run a program by typing only its name, not its path name. See Section 2.3.4.7, “Customizing the PATH for MySQL Tools”.

With the server running, issue the following commands to verify that you can retrieve information from the server. The output should be similar to that shown here.

Use `mysqlshow` to see what databases exist:

```
C:\> bin\mysqlshow
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

The list of installed databases may vary, but always includes at least `mysql` and `information_schema`.

The preceding command (and commands for other MySQL programs such as `mysql`) may not work if the correct MySQL account does not exist. For example, the program may fail with an error, or you may not be able to view all databases. If you install MySQL using MySQL Installer, the `root` user is created automatically with the password you supplied. In this case, you should use the `-u root` and `-p` options.

(You must use those options if you have already secured the initial MySQL accounts.) With `-p`, the client program prompts for the `root` password. For example:

```
C:\> bin\mysqlshow -u root -p
Enter password: (enter root password here)
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

If you specify a database name, `mysqlshow` displays a list of the tables within the database:

```
C:\> bin\mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
| columns_priv              |
| component                 |
| db                        |
| default_roles             |
| engine_cost               |
| func                      |
| general_log               |
| global_grants             |
| gtid_executed             |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| innodb_index_stats        |
| innodb_table_stats        |
| ndb_binlog_index          |
| password_history          |
| plugin                    |
| procs_priv                |
| proxies_priv              |
| role_edges                |
| server_cost               |
| servers                   |
| slave_master_info         |
| slave_relay_log_info      |
| slave_worker_info         |
| slow_log                  |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

Use the `mysql` program to select information from a table in the `mysql` database:

```
C:\> bin\mysql -e "SELECT User, Host, plugin FROM mysql.user" mysql
+------+-----------+-----------------------+
| User | Host      | plugin                |
+------+-----------+-----------------------+
```

```
| root | localhost | caching_sha2_password |
+------+-----------+-----------------------+
```

For more information about `mysql` and `mysqlshow`, see Section 4.5.1, “`mysql` — The MySQL Command-Line Client”, and Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”.

2.3.7 Windows Platform Restrictions

The following restrictions apply to use of MySQL on the Windows platform:

- **Process memory**

  On Windows 32-bit platforms, it is not possible by default to use more than 2GB of RAM within a single process, including MySQL. This is because the physical address limit on Windows 32-bit is 4GB and the default setting within Windows is to split the virtual address space between kernel (2GB) and user/applications (2GB).

  Some versions of Windows have a boot time setting to enable larger applications by reducing the kernel application. Alternatively, to use more than 2GB, use a 64-bit version of Windows.

- **File system aliases**

  When using `MyISAM` tables, you cannot use aliases within Windows link to the data files on another volume and then link back to the main MySQL `datadir` location.

  This facility is often used to move the data and index files to a RAID or other fast solution.

- **Limited number of ports**

  Windows systems have about 4,000 ports available for client connections, and after a connection on a port closes, it takes two to four minutes before the port can be reused. In situations where clients connect to and disconnect from the server at a high rate, it is possible for all available ports to be used up before closed ports become available again. If this happens, the MySQL server appears to be unresponsive even though it is running. Ports may be used by other applications running on the machine as well, in which case the number of ports available to MySQL is lower.

  For more information about this problem, see http://support.microsoft.com/default.aspx?scid=kb;en-us;196271.

- **`DATA DIRECTORY` and `INDEX DIRECTORY`**

  The `DATA DIRECTORY` option for `CREATE TABLE` is supported on Windows only for `InnoDB` tables, as described in Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”. For `MyISAM` and other storage engines, the `DATA DIRECTORY` and `INDEX DIRECTORY` options for `CREATE TABLE` are ignored on Windows and any other platforms with a nonfunctional `realpath()` call.

- **`DROP DATABASE`**

  You cannot drop a database that is in use by another session.

- **Case-insensitive names**

  File names are not case-sensitive on Windows, so MySQL database and table names are also not case-sensitive on Windows. The only restriction is that database and table names must be specified using the same case throughout a given statement. See Section 9.2.3, “Identifier Case Sensitivity”.

- **Directory and file names**

On Windows, MySQL Server supports only directory and file names that are compatible with the current ANSI code pages. For example, the following Japanese directory name will not work in the Western locale (code page 1252):

```
datadir="C:/私たちのプロジェクトのデータ"
```

The same limitation applies to directory and file names referred to in SQL statements, such as the data file path name in `LOAD DATA`.

- **The `\` path name separator character**

  Path name components in Windows are separated by the `\` character, which is also the escape character in MySQL. If you are using `LOAD DATA` or `SELECT ... INTO OUTFILE`, use Unix-style file names with `/` characters:

  ```
  mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
  ```

  Alternatively, you must double the `\` character:

  ```
  mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
  ```

- **Problems with pipes**

  Pipes do not work reliably from the Windows command-line prompt. If the pipe includes the character `^Z` / `CHAR(24)`, Windows thinks that it has encountered end-of-file and aborts the program.

  This is mainly a problem when you try to apply a binary log as follows:

  ```
  C:\> mysqlbinlog binary_log_file | mysql --user=root
  ```

  If you have a problem applying the log and suspect that it is because of a `^Z` / `CHAR(24)` character, you can use the following workaround:

  ```
  C:\> mysqlbinlog binary_log_file --result-file=/tmp/bin.sql
  C:\> mysql --user=root --execute "source /tmp/bin.sql"
  ```

  The latter command also can be used to reliably read any SQL file that may contain binary data.

2.4 Installing MySQL on macOS

For a list of macOS versions that the MySQL server supports, see https://www.mysql.com/support/supportedplatforms/database.html.

MySQL for macOS is available in a number of different forms:

- Native Package Installer, which uses the native macOS installer (DMG) to walk you through the installation of MySQL. For more information, see Section 2.4.2, “Installing MySQL on macOS Using Native Packages”. You can use the package installer with macOS. The user you use to perform the installation must have administrator privileges.
- Compressed TAR archive, which uses a file packaged using the Unix `tar` and `gzip` commands. To use this method, you will need to open a `Terminal` window. You do not need administrator privileges using

this method, as you can install the MySQL server anywhere using this method. For more information on using this method, you can use the generic instructions for using a tarball, Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

In addition to the core installation, the Package Installer also includes Section 2.4.3, “Installing and Using the MySQL Launch Daemon” and Section 2.4.4, “Installing and Using the MySQL Preference Pane” to simplify the management of your installation.

For additional information on using MySQL on macOS, see Section 2.4.1, “General Notes on Installing MySQL on macOS”.

2.4.1 General Notes on Installing MySQL on macOS

You should keep the following issues and notes in mind:

- **Other MySQL installations**: The installation procedure does not recognize MySQL installations by package managers such as Homebrew. The installation and upgrade process is for MySQL packages provided by us. If other installations are present, then consider stopping them before executing this installer to avoid port conflicts.

  Homebrew: For example, if you installed MySQL Server using Homebrew to its default location then the MySQL installer installs to a different location and won't upgrade the version from Homebrew. In this scenario you would end up with multiple MySQL installations that, by default, attempt to use the same ports. Stop the other MySQL Server instances before running this installer, such as executing *brew services stop mysql* to stop the Homebrew's MySQL service.

- **Launchd**: A launchd daemon is installed that alters MySQL configuration options. Consider editing it if needed, see the documentation below for additional information. Also, macOS 10.10 removed startup item support in favor of launchd daemons. The optional MySQL preference pane under macOS **System Preferences** uses the launchd daemon.

- **Users**: You may need (or want) to create a specific `mysql` user to own the MySQL directory and data. You can do this through the `Directory Utility`, and the `mysql` user should already exist. For use in single user mode, an entry for `_mysql` (note the underscore prefix) should already exist within the system `/etc/passwd` file.

- **Data**: Because the MySQL package installer installs the MySQL contents into a version and platform specific directory, you can use this to upgrade and migrate your database between versions. You will need to either copy the `data` directory from the old version to the new version, or alternatively specify an alternative `datadir` value to set location of the data directory. By default, the MySQL directories are installed under `/usr/local/`.

- **Aliases**: You might want to add aliases to your shell's resource file to make it easier to access commonly used programs such as `mysql` and `mysqladmin` from the command line. The syntax for `bash` is:

  ```
  alias mysql=/usr/local/mysql/bin/mysql
  alias mysqladmin=/usr/local/mysql/bin/mysqladmin
  ```

  For `tcsh`, use:

  ```
  alias mysql /usr/local/mysql/bin/mysql
  alias mysqladmin /usr/local/mysql/bin/mysqladmin
  ```

Even better, add `/usr/local/mysql/bin` to your `PATH` environment variable. You can do this by modifying the appropriate startup file for your shell. For more information, see Section 4.2.1, “Invoking MySQL Programs”.

- **Removing**: After you have copied over the MySQL database files from the previous installation and have successfully started the new server, you should consider removing the old installation files to save disk space. Additionally, you should also remove older versions of the Package Receipt directories located in `/Library/Receipts/mysql-VERSION.pkg`.
- **Legacy**: Prior to OS X 10.7, MySQL server was bundled with OS X Server.

2.4.2 Installing MySQL on macOS Using Native Packages

The package is located inside a disk image (`.dmg`) file that you first need to mount by double-clicking its icon in the Finder. It should then mount the image and display its contents.

Note

Before proceeding with the installation, be sure to stop all running MySQL server instances by using either the MySQL Manager Application (on macOS Server), the preference pane, or `mysqladmin shutdown` on the command line.

To install MySQL using the package installer:

1. Download the disk image (`.dmg`) file (the community version is available here) that contains the MySQL package installer. Double-click the file to mount the disk image and see its contents.

   Double-click the MySQL installer package from the disk. It is named according to the version of MySQL you have downloaded. For example, for MySQL server 8.0.19 it might be named `mysql-8.0.19-osx-10.13-x86_64.pkg`.

2. The initial wizard introduction screen references the MySQL server version to install. Click **Continue** to begin the installation.

   The MySQL community edition shows a copy of the relevant GNU General Public License. Click **Continue** and then **Agree** to continue.

3. From the **Installation Type** page you can either click **Install** to execute the installation wizard using all defaults, click **Customize** to alter which components to install (MySQL server, MySQL Test, Preference Pane, Launchd Support -- all but MySQL Test are enabled by default).

Note

Although the **Change Install Location** option is visible, the installation location cannot be changed.

Figure 2.13 MySQL Package Installer Wizard: Installation Type

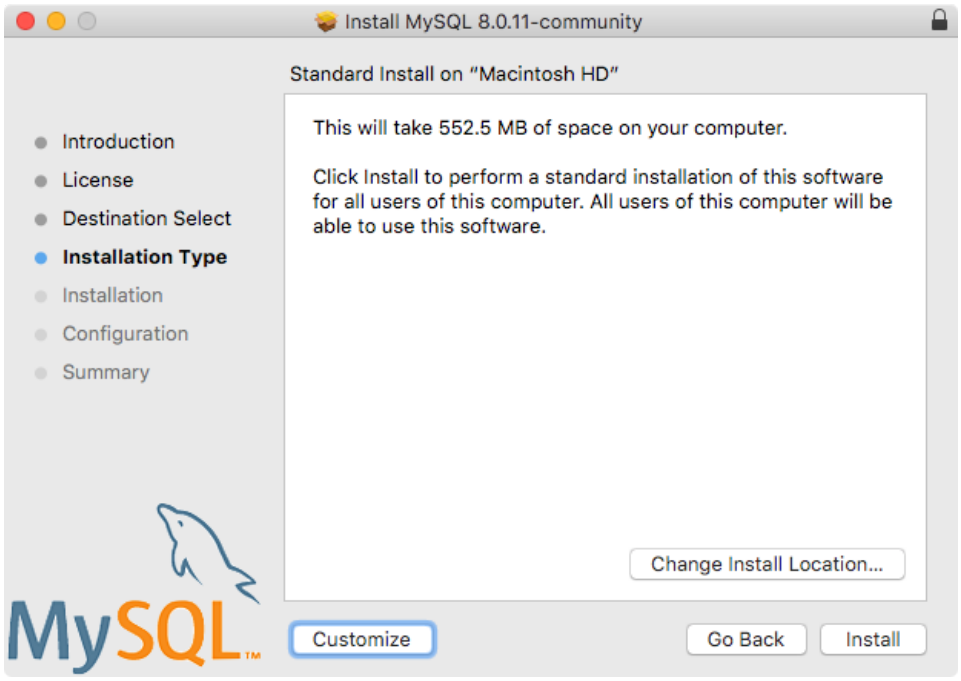
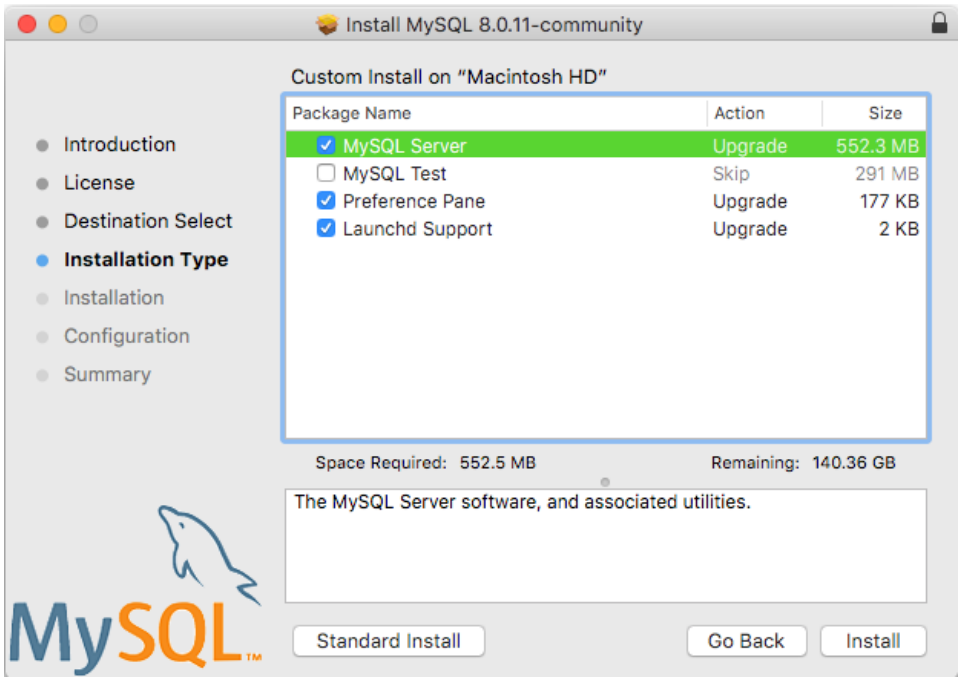


Figure 2.14 MySQL Package Installer Wizard: Customize



4. Click **Install** to install MySQL Server. The installation process ends here if upgrading a current MySQL Server installation, otherwise follow the wizard's additional configuration steps for your new MySQL Server installation.

5. After a successful new MySQL Server installation, complete the configuration steps by choosing the default encryption type for passwords, define the root password, and also enable (or disable) MySQL server at startup.

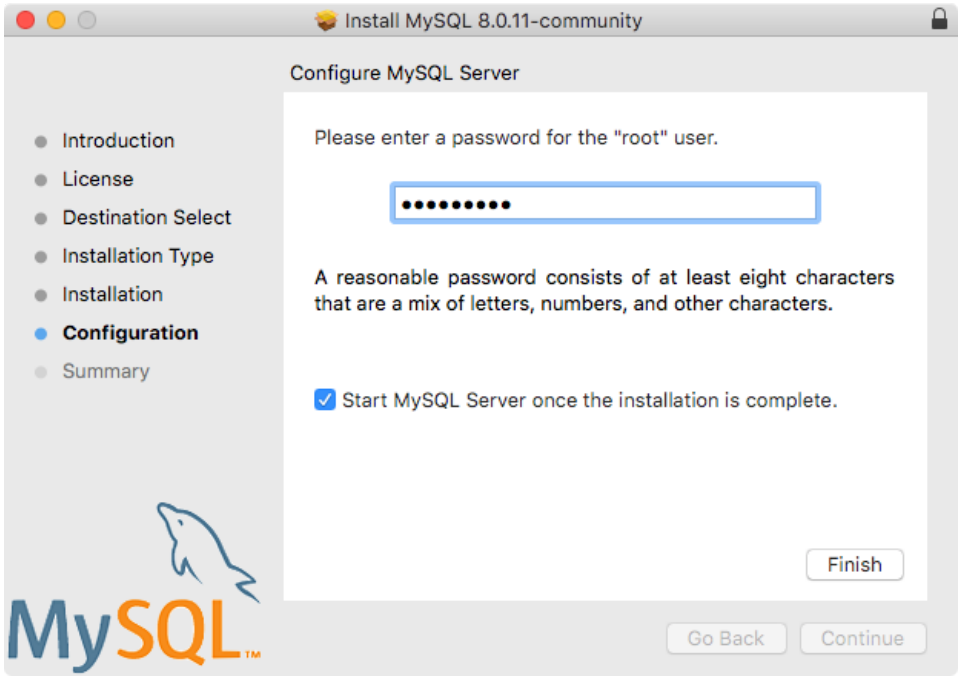
6. The default MySQL 8.0 password mechanism is `caching_sha2_password` (Strong), and this step allows you to change it to `mysql_native_password` (Legacy).

Figure 2.15 MySQL Package Installer Wizard: Choose a Password Encryption Type

Choosing the legacy password mechanism alters the generated launchd file to set `--default_authentication_plugin=mysql_native_password` under `ProgramArguments`. Choosing strong password encryption does not set `--default_authentication_plugin` because the default MySQL Server value is used, which is `caching_sha2_password`.

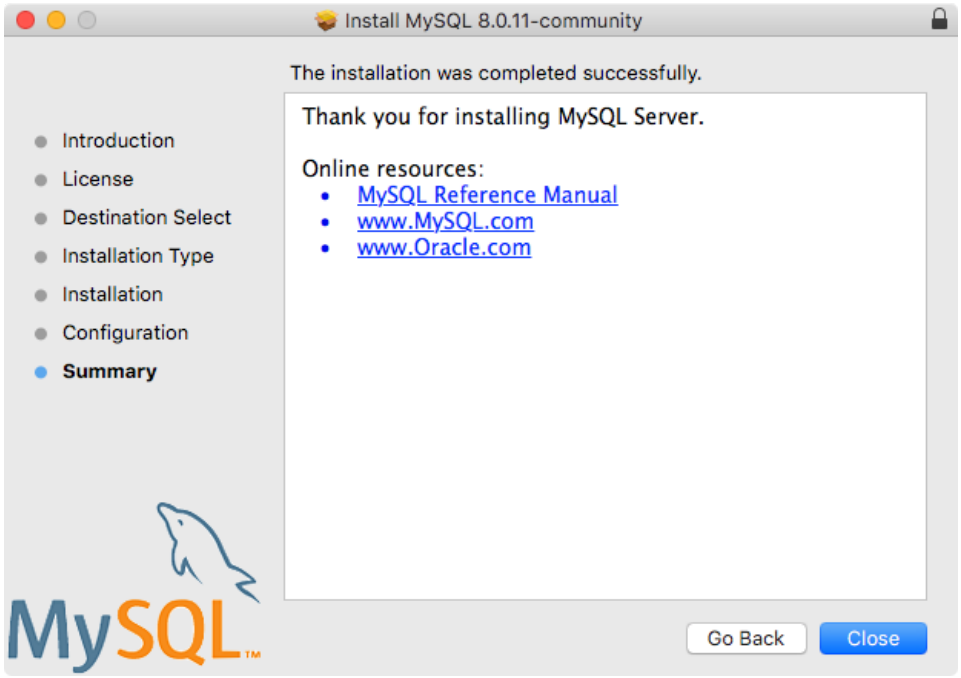
7. Define a password for the root user, and also toggle whether MySQL Server should start after the configuration step is complete.

Figure 2.16 MySQL Package Installer Wizard: Define Root Password



8. **Summary** is the final step and references a successful and complete MySQL Server installation. **Close** the wizard.

Figure 2.17 MySQL Package Installer Wizard: Summary



MySQL server is now installed. If you chose to not start MySQL, then use either launchctl from the command line or start MySQL by clicking "Start" using the MySQL preference pane. For additional information, see Section 2.4.3, “Installing and Using the MySQL Launch Daemon”, and Section 2.4.4,

"Installing and Using the MySQL Preference Pane". Use the MySQL Preference Pane or launchd to configure MySQL to automatically start at bootup.

When installing using the package installer, the files are installed into a directory within `/usr/local` matching the name of the installation version and platform. For example, the installer file `mysql-8.0.19-osx10.13-x86_64.dmg` installs MySQL into `/usr/local/mysql-8.0.19-osx10.13-x86_64/` with a symlink to `/usr/local/mysql`. The following table shows the layout of this MySQL installation directory.

Table 2.6 MySQL Installation Layout on macOS

Directory	Contents of Directory
`bin`	`mysqld` server, client and utility programs
`data`	Log files, databases, where `/usr/local/mysql/data/mysqld.local.err` is the default error log
`docs`	Helper documents, like the Release Notes and build information
`include`	Include (header) files
`lib`	Libraries
`man`	Unix manual pages
`mysql-test`	MySQL test suite ('MySQL Test' is disabled by default during the installation process when using the installer package (DMG))
`share`	Miscellaneous support files, including error messages, sample configuration files, SQL for database installation
`support-files`	Scripts and sample configuration files
`/tmp/mysql.sock`	Location of the MySQL Unix socket

2.4.3 Installing and Using the MySQL Launch Daemon

macOS uses launch daemons to automatically start, stop, and manage processes and applications such as MySQL.

By default, the installation package (DMG) on macOS installs a launchd file named `/Library/LaunchDaemons/com.oracle.oss.mysql.mysqld.plist` that contains a plist definition similar to:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.
<plist version="1.0">
<dict>
    <key>Label</key>            <string>com.oracle.oss.mysql.mysqld</string>
    <key>ProcessType</key>      <string>Interactive</string>
    <key>Disabled</key>         <false/>
    <key>RunAtLoad</key>        <true/>
    <key>KeepAlive</key>        <true/>
    <key>SessionCreate</key>    <true/>
    <key>LaunchOnlyOnce</key>   <false/>
    <key>UserName</key>         <string>_mysql</string>
    <key>GroupName</key>        <string>_mysql</string>
    <key>ExitTimeOut</key>      <integer>600</integer>
    <key>Program</key>          <string>/usr/local/mysql/bin/mysqld</string>
    <key>ProgramArguments</key>
        <array>
            <string>/usr/local/mysql/bin/mysqld</string>
```

```
            <string>--user=_mysql</string>
            <string>--basedir=/usr/local/mysql</string>
            <string>--datadir=/usr/local/mysql/data</string>
            <string>--plugin-dir=/usr/local/mysql/lib/plugin</string>
            <string>--log-error=/usr/local/mysql/data/mysqld.local.err</string>
            <string>--pid-file=/usr/local/mysql/data/mysqld.local.pid</string>
            <string>--keyring-file-data=/usr/local/mysql/keyring/keyring</string>
            <string>--early-plugin-load=keyring_file=keyring_file.so</string>
        </array>
    <key>WorkingDirectory</key>  <string>/usr/local/mysql</string>
</dict>
</plist>
```

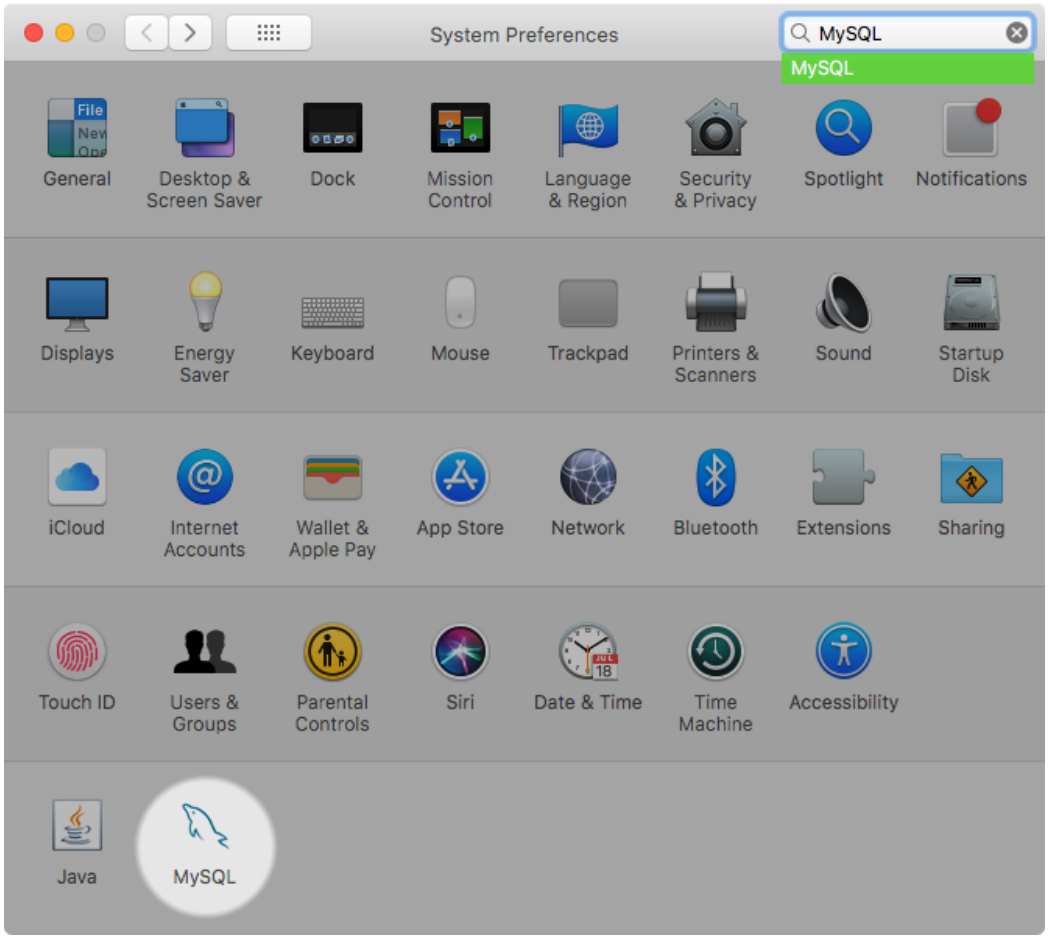
Note

Some users report that adding a plist DOCTYPE declaration causes the launchd operation to fail, despite it passing the lint check. We suspect it's a copy-n-paste error. The md5 checksum of a file containing the above snippet is *d925f05f6d1b6ee5ce5451b596d6baed*.

To enable the launchd service, you can either:

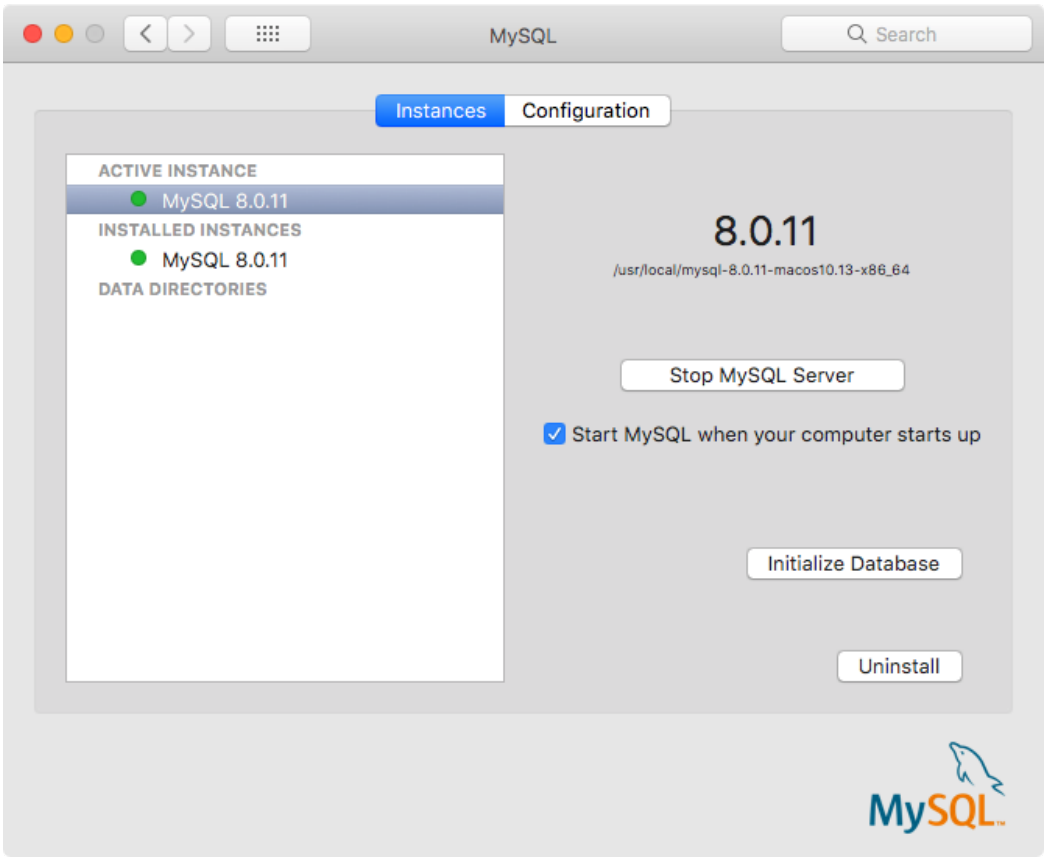
- Open macOS system preferences and select the MySQL preference panel, and then execute **Start MySQL Server**.

  Figure 2.18 MySQL Preference Pane: Location



The **Instances** page includes an option to start or stop MySQL, and **Initialize Database** recreates the `data/` directory. **Uninstall** uninstalls MySQL Server and optionally the MySQL preference panel and launchd information.

Figure 2.19 MySQL Preference Pane: Instances



- Or, manually load the launchd file.

```
shell> cd /Library/LaunchDaemons
shell> sudo launchctl load -F com.oracle.oss.mysql.mysqld.plist
```

- To configure MySQL to automatically start at bootup, you can:

```
shell> sudo launchctl load -w com.oracle.oss.mysql.mysqld.plist
```

Note

When upgrading MySQL server, the launchd installation process will remove the old startup items that were installed with MySQL server 5.7.7 and below.

Also, upgrading will replace your existing launchd file named `com.oracle.oss.mysql.mysqld.plist`.

Additional launchd related information:

- The plist entries override `my.cnf` entries, because they are passed in as command line arguments. For additional information about passing in program options, see Section 4.2.2, “Specifying Program Options”.
- The **ProgramArguments** section defines the command line options that are passed into the program, which is the `mysqld` binary in this case.
- The default plist definition is written with less sophisticated use cases in mind. For more complicated setups, you may want to remove some of the arguments and instead rely on a MySQL configuration file, such as `my.cnf`.
- If you edit the plist file, then uncheck the installer option when reinstalling or upgrading MySQL. Otherwise, your edited plist file will be overwritten, and all edits will be lost.

Because the default plist definition defines several **ProgramArguments**, you might remove most of these arguments and instead rely upon your `my.cnf` MySQL configuration file to define them. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd"
<plist version="1.0">
<dict>
    <key>Label</key>             <string>com.oracle.oss.mysql.mysqld</string>
    <key>ProcessType</key>       <string>Interactive</string>
    <key>Disabled</key>          <false/>
    <key>RunAtLoad</key>         <true/>
    <key>KeepAlive</key>         <true/>
    <key>SessionCreate</key>     <true/>
    <key>LaunchOnlyOnce</key>    <false/>
    <key>UserName</key>          <string>_mysql</string>
    <key>GroupName</key>         <string>_mysql</string>
    <key>ExitTimeOut</key>       <integer>600</integer>
    <key>Program</key>           <string>/usr/local/mysql/bin/mysqld</string>
    <key>ProgramArguments</key>
        <array>
            <string>/usr/local/mysql/bin/mysqld</string>
            <string>--user=_mysql</string>
            <string>--basedir=/usr/local/mysql</string>
            <string>--datadir=/usr/local/mysql/data</string>
            <string>--plugin-dir=/usr/local/mysql/lib/plugin</string>
            <string>--log-error=/usr/local/mysql/data/mysqld.local.err</string>
            <string>--pid-file=/usr/local/mysql/data/mysqld.local.pid</string>
            <string>--keyring-file-data=/usr/local/mysql/keyring/keyring</string>
            <string>--early-plugin-load=keyring_file=keyring_file.so</string>
        </array>
    <key>WorkingDirectory</key>  <string>/usr/local/mysql</string>
</dict>
</plist>
```

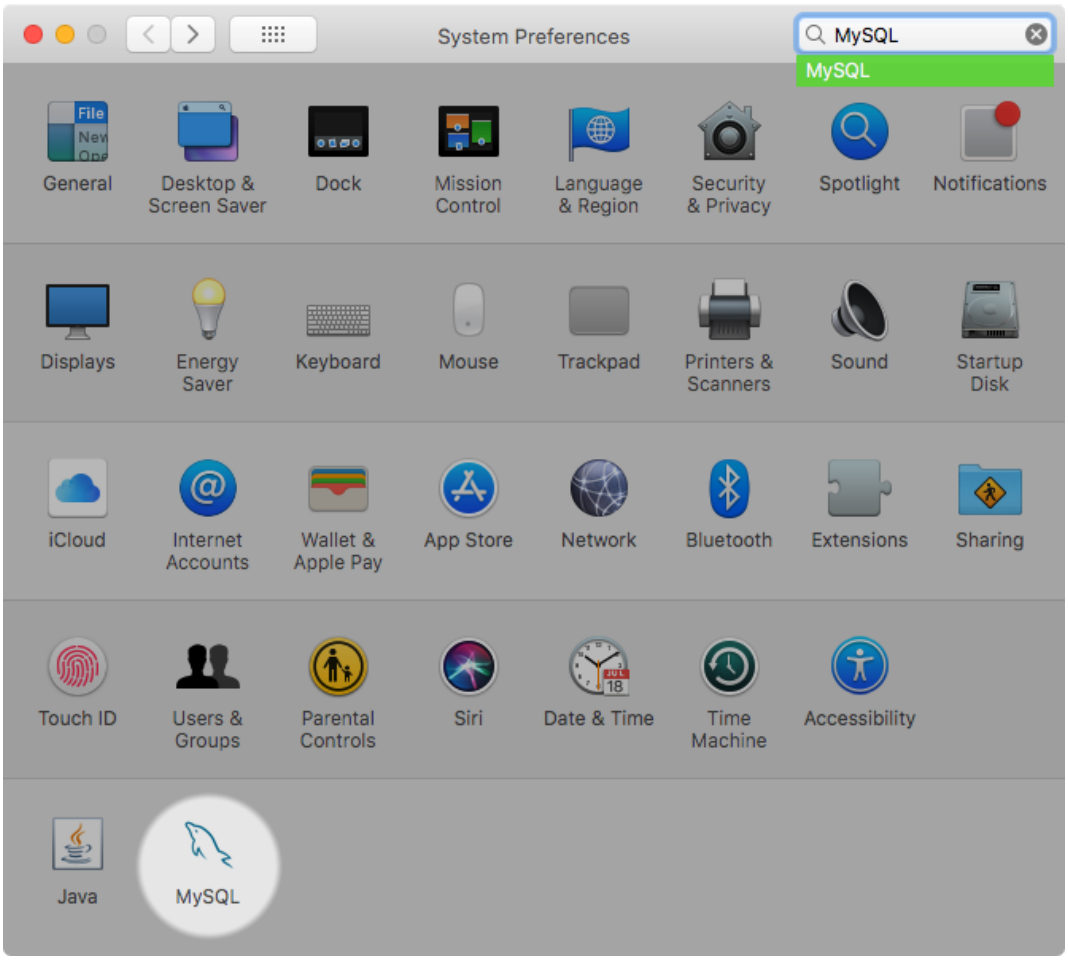
In this case, the `basedir`, `datadir`, `plugin_dir`, `log_error`, `pid_file`, `keyring_file_data`, and `--early-plugin-load` options were removed from the default plist *ProgramArguments* definition, which you might have defined in `my.cnf` instead.

2.4.4 Installing and Using the MySQL Preference Pane

The MySQL Installation Package includes a MySQL preference pane that enables you to start, stop, and control automated startup during boot of your MySQL installation.

This preference pane is installed by default, and is listed under your system's *System Preferences* window.

Figure 2.20 MySQL Preference Pane: Location



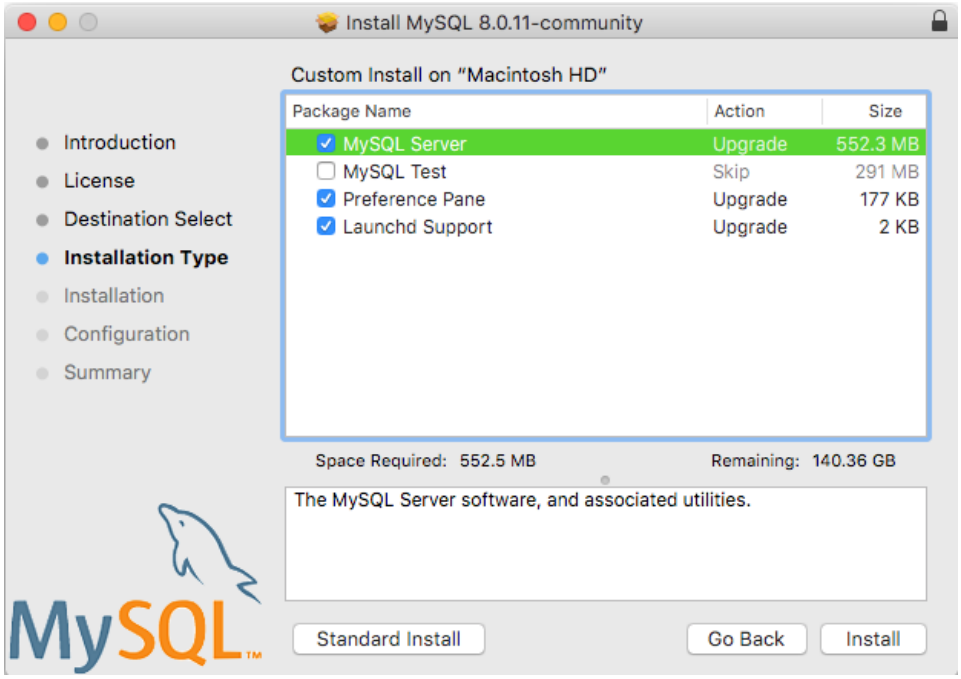
The MySQL preference pane is installed with the same DMG file that installs MySQL Server. Typically it is installed with MySQL Server but it can be installed by itself too.

To install the MySQL preference pane:

1. Go through the process of installing the MySQL server, as described in the documentation at Section 2.4.2, “Installing MySQL on macOS Using Native Packages”.

2. Click **Customize** at the **Installation Type** step. The "Preference Pane" option is listed there and enabled by default; make sure it is not deselected. The other options, such as MySQL Server, can be selected or deslected.

Figure 2.21 MySQL Package Installer Wizard: Customize



3. Complete the installation process.

Note

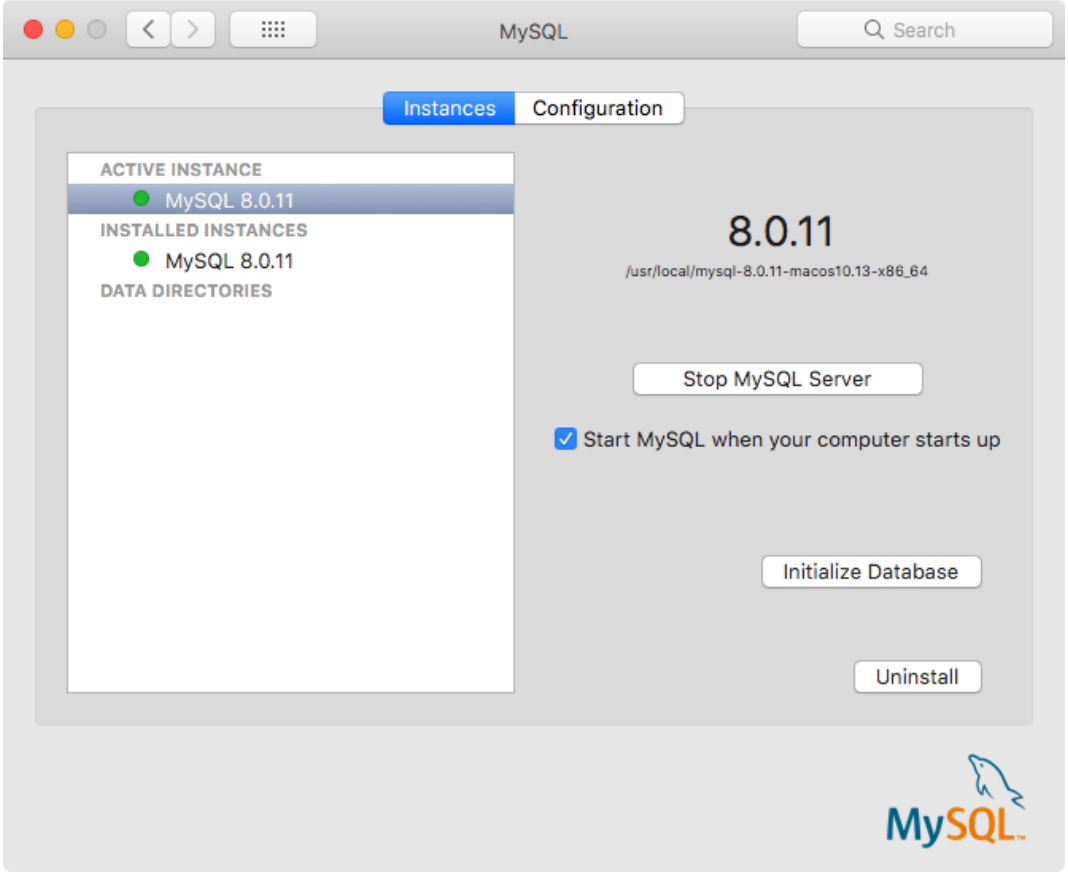
The MySQL preference pane only starts and stops MySQL installation installed from the MySQL package installation that have been installed in the default location.

Once the MySQL preference pane has been installed, you can control your MySQL server instance using this preference pane.

The **Instances** page includes an option to start and stop MySQL, and **Initialize Database** recreates the `data/` directory. **Uninstall** uninstalls MySQL Server and optionally the pain and launchd information.

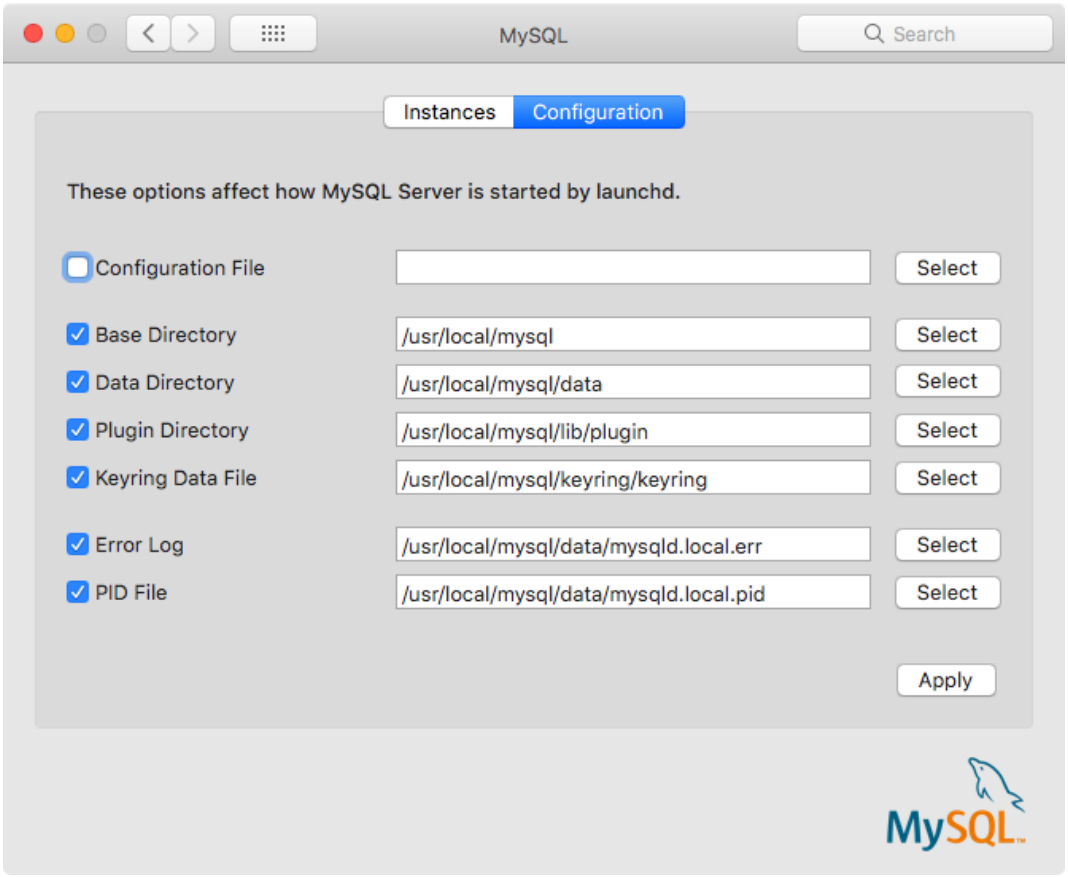
The **Instances** page includes an option to start or stop MySQL, and **Initialize Database** recreates the `data/` directory. **Uninstall** uninstalls MySQL Server and optionally the MySQL preference panel and launchd information.

Figure 2.22 MySQL Preference Pane: Instances



The **Configuration** page shows MySQL Server options including the path to the MySQL configuration file.

Figure 2.23 MySQL Preference Pane: Configuration



The MySQL Preference Pane shows the current status of the MySQL server, showing **stopped** (in red) if the server is not running and **running** (in green) if the server has already been started. The preference pane also shows the current setting for whether the MySQL server has been set to start automatically.

2.5 Installing MySQL on Linux

Linux supports a number of different solutions for installing MySQL. We recommend that you use one of the distributions from Oracle, for which several methods for installation are available:

Table 2.7 Linux Installation Methods and Information

Type	Setup Method	Additional Information
Apt	Enable the MySQL Apt repository	Documentation
Yum	Enable the MySQL Yum repository	Documentation
Zypper	Enable the MySQL SLES repository	Documentation
RPM	Download a specific package	Documentation
DEB	Download a specific package	Documentation
Generic	Download a generic package	Documentation
Source	Compile from source	Documentation

Type	Setup Method	Additional Information
Docker	Use Docker Hub for MySQL Community Edition; download Docker image for MySQL Enterprise Edition from My Oracle Support	Documentation
Oracle Unbreakable Linux Network	Use ULN channels	Documentation

As an alternative, you can use the package manager on your system to automatically download and install MySQL with packages from the native software repositories of your Linux distribution. These native packages are often several versions behind the currently available release. You will also normally be unable to install development milestone releases (DMRs), as these are not usually made available in the native repositories. For more information on using the native package installers, see Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”.

Note

For many Linux installations, you will want to set up MySQL to be started automatically when your machine starts. Many of the native package installations perform this operation for you, but for source, binary and RPM solutions you may need to set this up separately. The required script, `mysql.server`, can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.

2.5.1 Installing MySQL on Linux Using the MySQL Yum Repository

The MySQL Yum repository for Oracle Linux, Red Hat Enterprise Linux, CentOS, and Fedora provides RPM packages for installing the MySQL server, client, MySQL Workbench, MySQL Utilities, MySQL Router, MySQL Shell, Connector/ODBC, Connector/Python and so on (not all packages are available for all the distributions; see Installing Additional MySQL Products and Components with Yum for details).

Before You Start

As a popular, open-source software, MySQL, in its original or re-packaged form, is widely installed on many systems from various sources, including different software download sites, software repositories, and so on. The following instructions assume that MySQL is not already installed on your system using a third-party-distributed RPM package; if that is not the case, follow the instructions given in Section 2.11.7, “Upgrading MySQL with the MySQL Yum Repository” or Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository.

Steps for a Fresh Installation of MySQL

Follow the steps below to install the latest GA version of MySQL with the MySQL Yum repository:

Adding the MySQL Yum Repository

First, add the MySQL Yum repository to your system's repository list. This is a one-time operation, which can be performed by installing an RPM provided by MySQL. Follow these steps:

a. Go to the Download MySQL Yum Repository page (https://dev.mysql.com/downloads/repo/yum/) in the MySQL Developer Zone.

b. Select and download the release package for your platform.

c. Install the downloaded release package with the following command, replacing *platform-and-version-specific-package-name* with the name of the downloaded RPM package:

```
shell> sudo yum localinstall platform-and-version-specific-package-name.rpm
```

For an EL6-based system, the command is in the form of:

```
shell> sudo yum localinstall mysql80-community-release-el6-{version-number}.noarch.rpm
```

For an EL7-based system:

```
shell> sudo yum localinstall mysql80-community-release-el7-{version-number}.noarch.rpm
```

For an EL8-based system:

```
shell> sudo yum localinstall mysql80-community-release-el8-{version-number}.noarch.rpm
```

For Fedora 30:

```
shell> sudo dnf localinstall mysql80-community-release-fc30-{version-number}.noarch.rpm
```

For Fedora 29:

```
shell> sudo dnf localinstall mysql80-community-release-fc29-{version-number}.noarch.rpm
```

The installation command adds the MySQL Yum repository to your system's repository list and downloads the GnuPG key to check the integrity of the software packages. See Section 2.1.3.2, “Signature Checking Using GnuPG” for details on GnuPG key checking.

You can check that the MySQL Yum repository has been successfully added by the following command (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
shell> yum repolist enabled | grep "mysql.*-community.*"
```

Note

Once the MySQL Yum repository is enabled on your system, any system-wide update by the `yum update` command (or `dnf upgrade` for dnf-enabled systems) will upgrade MySQL packages on your system and also replace any native third-party packages, if Yum finds replacements for them in the MySQL Yum repository; see Section 2.11.7, “Upgrading MySQL with the MySQL Yum Repository” and, for a discussion on some possible effects of that on your system, see Upgrading the Shared Client Libraries.

Selecting a Release Series

When using the MySQL Yum repository, the latest GA series (currently MySQL 8.0) is selected for installation by default. If this is what you want, you can skip to the next step, Installing MySQL.

Within the MySQL Yum repository, different release series of the MySQL Community Server are hosted in different subrepositories. The subrepository for the latest GA series (currently MySQL 8.0) is enabled by default, and the subrepositories for all other series (for example, the MySQL 8.0 series) are disabled by default. Use this command to see all the subrepositories in the MySQL Yum repository, and see which of them are enabled or disabled (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
shell> yum repolist all | grep mysql
```

To install the latest release from the latest GA series, no configuration is needed. To install the latest release from a specific series other than the latest GA series, disable the subrepository for the latest GA series and enable the subrepository for the specific series before running the installation command. If your platform supports `yum-config-manager`, you can do that by issuing these commands, which disable the subrepository for the 5.7 series and enable the one for the 8.0 series:

```
shell> sudo yum-config-manager --disable mysql57-community
shell> sudo yum-config-manager --enable mysql80-community
```

For dnf-enabled platforms:

```
shell> sudo dnf config-manager --disable mysql57-community
shell> sudo dnf config-manager --enable mysql80-community
```

Besides using `yum-config-manager` or the `dnf config-manager` command, you can also select a release series by editing manually the `/etc/yum.repos.d/mysql-community.repo` file. This is a typical entry for a release series' subrepository in the file:

```
[mysql57-community]
name=MySQL 5.7 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.7-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

Find the entry for the subrepository you want to configure, and edit the `enabled` option. Specify `enabled=0` to disable a subrepository, or `enabled=1` to enable a subrepository. For example, to install MySQL 8.0, make sure you have `enabled=0` for the above subrepository entry for MySQL 5.7, and have `enabled=1` for the entry for the 8.0 series:

```
# Enable to use MySQL 8.0
[mysql80-community]
name=MySQL 8.0 Community Server
baseurl=http://repo.mysql.com/yum/mysql-8.0-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

You should only enable subrepository for one release series at any time. When subrepositories for more than one release series are enabled, the latest series will be used by Yum.

Verify that the correct subrepositories have been enabled and disabled by running the following command and checking its output (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
shell> yum repolist enabled | grep mysql
```

Installing MySQL

3.

Install MySQL by the following command (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
shell> sudo yum install mysql-community-server
```

This installs the package for MySQL server (`mysql-community-server`) and also packages for the components required to run the server, including packages for the client (`mysql-community-client`), the common error messages and character sets for client and server (`mysql-community-common`), and the shared client libraries (`mysql-community-libs`).

Starting the MySQL Server

4.

Start the MySQL server with the following command:

```
shell> sudo service mysqld start
Starting mysqld:[ OK ]
```

You can check the status of the MySQL server with the following command:

```
shell> sudo service mysqld status
mysqld (pid 3066) is running.
```

At the initial start up of the server, the following happens, given that the data directory of the server is empty:

- The server is initialized.
- SSL certificate and key files are generated in the data directory.
- `validate_password` is installed and enabled.
- A superuser account `'root'@'localhost'` is created. A password for the superuser is set and stored in the error log file. To reveal it, use the following command:

```
shell> sudo grep 'temporary password' /var/log/mysqld.log
```

Change the root password as soon as possible by logging in with the generated, temporary password and set a custom password for the superuser account:

```
shell> mysql -uroot -p
```

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';
```

Note

`validate_password` is installed by default. The default password policy implemented by `validate_password` requires that passwords contain at least one upper case letter, one lower case letter, one digit, and one special character, and that the total password length is at least 8 characters.

For more information on the postinstallation procedures, see Section 2.10, “Postinstallation Setup and Testing”.

Note

Compatibility Information for EL7-based platforms: The following RPM packages from the native software repositories of the platforms are incompatible with the package from the MySQL Yum repository that installs the MySQL server. Once you have installed MySQL using the MySQL Yum repository, you will not be able to install these packages (and vice versa).

- akonadi-mysql

Installing Additional MySQL Products and Components with Yum

You can use Yum to install and manage individual components of MySQL. Some of these components are hosted in sub-repositories of the MySQL Yum repository: for example, the MySQL Connectors are to be found in the MySQL Connectors Community sub-repository, and the MySQL Workbench in MySQL Tools Community. You can use the following command to list the packages for all the MySQL components available for your platform from the MySQL Yum repository (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
shell> sudo yum --disablerepo=\* --enablerepo='mysql*-community*' list available
```

Install any packages of your choice with the following command, replacing *`package-name`* with name of the package (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
shell> sudo yum install package-name
```

For example, to install MySQL Workbench on Fedora:

```
shell> sudo dnf install mysql-workbench-community
```

To install the shared client libraries (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
shell> sudo yum install mysql-community-libs
```

Platform Specific Notes

ARM Support

ARM 64-bit (aarch64) is supported on Oracle Linux 7 and requires the Oracle Linux 7 Software Collections Repository (ol7_software_collections). For example, to install the server:

```
shell> yum-config-manager --enable ol7_software_collections
shell> yum install mysql-community-server
```

Note

ARM 64-bit (aarch64) is supported on Oracle Linux 7 as of MySQL 8.0.12.

Known Limitation

The 8.0.12 release requires you to adjust the *libstdc++7* path by executing `ln -s /opt/oracle/oracle-armtoolset-1/root/usr/lib64 /usr/lib64/gcc7` after executing the `yum install` step.

Updating MySQL with Yum

Besides installation, you can also perform updates for MySQL products and components using the MySQL Yum repository. See Section 2.11.7, “Upgrading MySQL with the MySQL Yum Repository” for details.

2.5.2 Installing MySQL on Linux Using the MySQL APT Repository

The MySQL APT repository provides `deb` packages for installing and managing the MySQL server, client, and other components on the following Linux platforms:

- Debian 9
- Ubuntu 16.04, 17.10, and 18.04

Instructions for using the MySQL APT Repository are available in A Quick Guide to Using the MySQL APT Repository.

2.5.3 Installing MySQL on Linux Using the MySQL SLES Repository

The MySQL SLES repository provides RPM packages for installing and managing the MySQL server, client, and other components on SUSE Enterprise Linux Server.

Instructions for using the MySQL SLES repository are available in A Quick Guide to Using the MySQL SLES Repository.

2.5.4 Installing MySQL on Linux Using RPM Packages from Oracle

The recommended way to install MySQL on RPM-based Linux distributions is by using the RPM packages provided by Oracle. There are two sources for obtaining them, for the Community Edition of MySQL:

- From the MySQL software repositories:
  - The MySQL Yum repository (see Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details).
  - The MySQL SLES repository (see Section 2.5.3, “Installing MySQL on Linux Using the MySQL SLES Repository” for details).
- From the Download MySQL Community Server page in the MySQL Developer Zone.

Note

RPM distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by Oracle in features, capabilities, and conventions (including communication setup), and that the installation instructions in this manual do not necessarily apply to them. The vendor's instructions should be consulted instead.

MySQL RPM Packages

Table 2.8 RPM Packages for MySQL Community Edition

Package Name	Summary
`mysql-community-client`	MySQL client applications and tools
`mysql-community-common`	Common files for server and client libraries

Package Name	Summary
mysql-community-devel	Development header files and libraries for MySQL database client applications
mysql-community-embedded-compat	MySQL server as an embedded library with compatibility for applications using version 18 of the library
mysql-community-libs	Shared libraries for MySQL database client applications
mysql-community-libs-compat	Shared compatibility libraries for previous MySQL installations
mysql-community-server	Database server and related tools
mysql-community-server-debug	Debug server and plugin binaries
mysql-community-test	Test suite for the MySQL server
mysql-community	The source code RPM looks similar to mysql-community-8.0.19-1.el7.src.rpm, depending on selected OS

Table 2.9 RPM Packages for the MySQL Enterprise Edition

Package Name	Summary
mysql-commercial-backup	MySQL Enterprise Backup (added in 8.0.11)
mysql-commercial-client	MySQL client applications and tools
mysql-commercial-common	Common files for server and client libraries
mysql-commercial-devel	Development header files and libraries for MySQL database client applications
mysql-commercial-embedded-compat	MySQL server as an embedded library with compatibility for applications using version 18 of the library
mysql-commercial-libs	Shared libraries for MySQL database client applications

Package Name	Summary
mysql-commercial-libs-compat	Shared compatibility libraries for previous MySQL installations; the version of the libraries matches the version of the libraries installed by default by the distribution you are using
mysql-commercial-server	Database server and related tools
mysql-commercial-test	Test suite for the MySQL server

The full names for the RPMs have the following syntax:

```
packagename-version-distribution-arch.rpm
```

The *distribution* and *arch* values indicate the Linux distribution and the processor type for which the package was built. See the table below for lists of the distribution identifiers:

Table 2.10 MySQL Linux RPM Package Distribution Identifiers

Distribution Value	Intended Use
el6, el7, and el8	Red Hat Enterprise Linux/Oracle Linux/CentOS 6, 7, and 8
fc29 fc30	Fedora 29 and 30
sles12	SUSE Linux Enterprise Server 12

To see all files in an RPM package (for example, mysql-community-server), use the following command:

```
shell> rpm -qpl mysql-community-server-version-distribution-arch.rpm
```

The discussion in the rest of this section applies only to an installation process using the RPM packages directly downloaded from Oracle, instead of through a MySQL repository.

Dependency relationships exist among some of the packages. If you plan to install many of the packages, you may wish to download the RPM bundle tar file instead, which contains all the RPM packages listed above, so that you need not download them separately.

In most cases, you need to install the mysql-community-server, mysql-community-client, mysql-community-libs, mysql-community-common, and mysql-community-libs-compat packages to get a functional, standard MySQL installation. To perform such a standard, basic installation, go to the folder that contains all those packages (and, preferably, no other RPM packages with similar names), and issue the following command:

```
shell> sudo yum install mysql-community-{server,client,common,libs}-*
```

Replace yum with zypper for SLES, and with dnf for Fedora.

While it is much preferable to use a high-level package management tool like yum to install the packages, users who prefer direct rpm commands can replace the yum install command with the rpm -Uvh

command; however, using `rpm -Uvh` instead makes the installation process more prone to failure, due to potential dependency issues the installation process might run into.

To install only the client programs, you can skip `mysql-community-server` in your list of packages to install; issue the following command:

```
shell> sudo yum install mysql-community-{client,common,libs}-*
```

Replace `yum` with `zypper` for SLES, and with `dnf` for Fedora.

A standard installation of MySQL using the RPM packages result in files and resources created under the system directories, shown in the following table.

Table 2.11 MySQL Installation Layout for Linux RPM Packages from the MySQL Developer Zone

Files or Resources	Location
Client programs and scripts	`/usr/bin`
`mysqld` server	`/usr/sbin`
Configuration file	`/etc/my.cnf`
Data directory	`/var/lib/mysql`
Error log file	For RHEL, Oracle Linux, CentOS or Fedora platforms: `/var/log/mysqld.log` For SLES: `/var/log/mysql/mysqld.log`
Value of `secure_file_priv`	`/var/lib/mysql-files`
System V init script	For RHEL, Oracle Linux, CentOS or Fedora platforms: `/etc/init.d/mysqld` For SLES: `/etc/init.d/mysql`
Systemd service	For RHEL, Oracle Linux, CentOS or Fedora platforms: `mysqld` For SLES: `mysql`
Pid file	`/var/run/mysql/mysqld.pid`
Socket	`/var/lib/mysql/mysql.sock`
Keyring directory	`/var/lib/mysql-keyring`
Unix manual pages	`/usr/share/man`
Include (header) files	`/usr/include/mysql`
Libraries	`/usr/lib/mysql`
Miscellaneous support files (for example, error messages, and character set files)	`/usr/share/mysql`

The installation also creates a user named `mysql` and a group named `mysql` on the system.

Note

Installation of previous versions of MySQL using older packages might have created a configuration file named `/usr/my.cnf`. It is highly recommended that you examine the contents of the file and migrate the desired settings inside to the file `/etc/my.cnf` file, then remove `/usr/my.cnf`.

MySQL is NOT automatically started at the end of the installation process. For Red Hat Enterprise Linux, Oracle Linux, CentOS, and Fedora systems, use the following command to start MySQL:

```
shell> systemctl start mysqld
```

For SLES systems, the command is the same, but the service name is different:

```
shell> systemctl start mysql
```

If the operating system is systemd enabled, standard `systemctl` (or alternatively, `service` with the arguments reversed) commands such as `stop`, `start`, `status`, and `restart` should be used to manage the MySQL server service. The `mysqld` service is enabled by default, and it starts at system reboot. Notice that certain things might work differently on systemd platforms: for example, changing the location of the data directory might cause issues. See Section 2.5.9, “Managing MySQL Server with systemd” for additional information.

During an upgrade installation using RPM packages, if the MySQL server is running when the upgrade occurs then the MySQL server is stopped, the upgrade occurs, and the MySQL server is restarted. One exception: if the edition also changes during an upgrade (such as community to commercial, or vice-versa), then MySQL server is not restarted.

At the initial start up of the server, the following happens, given that the data directory of the server is empty:

- The server is initialized.
- An SSL certificate and key files are generated in the data directory.
- `validate_password` is installed and enabled.
- A superuser account `'root'@'localhost'` is created. A password for the superuser is set and stored in the error log file. To reveal it, use the following command for RHEL, Oracle Linux, CentOS, and Fedora systems:

  ```
  shell> sudo grep 'temporary password' /var/log/mysqld.log
  ```

  Use the following command for SLES systems:

  ```
  shell> sudo grep 'temporary password' /var/log/mysql/mysqld.log
  ```

  The next step is to log in with the generated, temporary password and set a custom password for the superuser account:

```
shell> mysql -uroot -p
```

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass4!';
```

Note

`validate_password` is installed by default. The default password policy implemented by `validate_password` requires that passwords contain at least one upper case letter, one lower case letter, one digit, and one special character, and that the total password length is at least 8 characters.

If something goes wrong during installation, you might find debug information in the error log file `/var/log/mysqld.log`.

For some Linux distributions, it might be necessary to increase the limit on number of file descriptors available to `mysqld`. See Section B.4.2.17, “File Not Found and Similar Errors”

Installing Client Libraries from Multiple MySQL Versions. It is possible to install multiple client library versions, such as for the case that you want to maintain compatibility with older applications linked against previous libraries. To install an older client library, use the `--oldpackage` option with `rpm`. For example, to install `mysql-community-libs-5.5` on an EL6 system that has `libmysqlclient.21` from MySQL 8.0, use a command like this:

```
shell> rpm --oldpackage -ivh mysql-community-libs-5.5.50-2.el6.x86_64.rpm
```

Debug Package. A special variant of MySQL Server compiled with the debug package has been included in the server RPM packages. It performs debugging and memory allocation checks and produces a trace file when the server is running. To use that debug version, start MySQL with `/usr/sbin/mysqld-debug`, instead of starting it as a service or with `/usr/sbin/mysqld`. See Section 29.5.4, “The DBUG Package” for the debug options you can use.

Note

The default plugin directory for debug builds changed from `/usr/lib64/mysql/plugin` to `/usr/lib64/mysql/plugin/debug` in MySQL 8.0.4. Previously, it was necessary to change `plugin_dir` to `/usr/lib64/mysql/plugin/debug` for debug builds.

Rebuilding RPMs from source SRPMs. Source code SRPM packages for MySQL are available for download. They can be used as-is to rebuild the MySQL RPMs with the standard `rpmbuild` tool chain.

2.5.5 Installing MySQL on Linux Using Debian Packages from Oracle

Oracle provides Debian packages for installing MySQL on Debian or Debian-like Linux systems. The packages are available through two different channels:

- The MySQL APT Repository. This is the preferred method for installing MySQL on Debian-like systems, as it provides a simple and convenient way to install and update MySQL products. For details, see Section 2.5.2, “Installing MySQL on Linux Using the MySQL APT Repository”.
- The MySQL Developer Zone's Download Area. For details, see Section 2.1.2, “How to Get MySQL”. The following are some information on the Debian packages available there and the instructions for installing them:
  - Various Debian packages are provided in the MySQL Developer Zone for installing different components of MySQL on different Debian or Ubuntu platforms (currently, Debian 9, and Ubuntu 16, 17, and 18 are supported). The preferred method is to use the tarball bundle, which contains the packages needed for a basic setup of MySQL. The tarball bundles have names in the format of `mysql-server_`*`MVER`*`-`*`DVER`*`_CPU.deb-bundle.tar`. *`MVER`* is the MySQL version and *`DVER`* is the Linux distribution version. The *`CPU`* value indicates the processor type or family for which the package is built, as shown in the following table:

    Table 2.12 MySQL Debian and Ubuntu Installation Packages CPU Identifiers

`CPU` Value	Intended Processor Type or Family
`i386`	Pentium processor or better, 32 bit

CPU Value	Intended Processor Type or Family
amd64	64-bit x86 processor

- After downloading the tarball, unpack it with the following command:

```
shell> tar -xvf mysql-server_MVER-DVER_CPU.deb-bundle.tar
```

- You may need to install the `libaio` library if it is not already present on your system:

```
shell> sudo apt-get install libaio1
```

- Preconfigure the MySQL server package with the following command:

```
shell> sudo dpkg-preconfigure mysql-community-server_*.deb
```

  You will be asked to provide a password for the root user for your MySQL installation. You might also be asked other questions regarding the installation.

Important

Make sure you remember the root password you set. Users who want to set a password later can leave the **password** field blank in the dialogue box and just press **OK**; in that case, root access to the server is authenticated using the MySQL Socket Peer-Credential Authentication Plugin for connections using a Unix socket file. You can set the root password later using `mysql_secure_installation`.

- For a basic installation of the MySQL server, install the database common files package, the client package, the client metapackage, the server package, and the server metapackage (in that order); you can do that with a single command:

```
shell> sudo dpkg -i mysql-{common,community-client,client,community-server,server}_*.deb
```

  There are also packages with `server-core` and `client-core` in the package names. These contain binaries only and are installed automatically by the standard packages. Installing them by themselves will not result in a functioning MySQL setup.

  If you are being warned of unmet dependencies by `dpkg`, you can fix them using `apt-get`:

```
sudo apt-get -f install
```

  Here are where the files are installed on the system:

  - All configuration files (like `my.cnf`) are under `/etc/mysql`
  - All binaries, libraries, headers, etc., are under `/usr/bin` and `/usr/sbin`
  - The data directory is under `/var/lib/mysql`

Note

Debian distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by Oracle in features, capabilities, and conventions (including communication setup), and that the instructions in this manual do not

necessarily apply to installing them. The vendor's instructions should be consulted instead.

2.5.6 Deploying MySQL on Linux with Docker

The Docker deployment framework supports easy installation and configuration of MySQL Server. This section explains how to use a MySQL Server Docker image.

You need to have Docker installed on your system before you can use a MySQL Server Docker image. See Install Docker for instructions.

Important

You need to either run `docker` commands with `sudo`, or create a `docker` usergroup, and then add to it any users who want to run `docker` commands. See details here. Because Docker containers are always run with root privileges, you should understand the Docker daemon attack surface and properly mitigate the related risks.

2.5.6.1 Basic Steps for MySQL Server Deployment with Docker

Warning

The MySQL Docker images maintained by the MySQL team are built specifically for Linux platforms. Other platforms are not supported, and users using these MySQL Docker images on them are doing so at their own risk. See the discussion here for some known limitations for running these containers on non-Linux operating systems.

- Downloading a MySQL Server Docker Image
- Starting a MySQL Server Instance
- Connecting to MySQL Server from within the Container
- Container Shell Access
- Stopping and Deleting a MySQL Container
- Upgrading a MySQL Server Container
- More Topics on Deploying MySQL Server with Docker

Downloading a MySQL Server Docker Image

Downloading the server image in a separate step is not strictly necessary; however, performing this step before you create your Docker container ensures your local image is up to date. To download the MySQL Community Edition image, run this command:

```
docker pull mysql/mysql-server:tag
```

The *`tag`* is the label for the image version you want to pull (for example, `5.5`, `5.6`, `5.7`, `8.0`, or `latest`). If **`:tag`** is omitted, the `latest` label is used, and the image for the latest GA version of MySQL Community Server is downloaded. Refer to the list of tags for available versions on the mysql/mysql-server page in the Docker Hub.

You can list downloaded Docker images with this command:

```
```

```
shell> docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
mysql/mysql-server   latest              3157d7f55f8d        4 weeks ago         241MB
```

To download the MySQL Enterprise Edition image from the My Oracle Support website, sign in to your Oracle account, download from **Patches and Updates** the `tar.zip` file for the Docker image (`mysql-commercial-`*`version`*`_linux_x86_64_docker_tar.zip`), unzip it to obtain the tarball inside (`mysql-enterprise-server-`*`version`*`.tar`), and then load the image by running this command:

```
docker load -i mysql-enterprise-server-version.tar
```

Starting a MySQL Server Instance

To start a new Docker container for a MySQL Server, use the following command:

```
docker run --name=container_name -d image_name:tag
```

The image name can be obtained using the `docker images` command, as explained in Downloading a MySQL Server Docker Image. The `--name` option, for supplying a custom name for your server container, is optional; if no container name is supplied, a random one is generated.

For example, to start a new Docker container for the MySQL Community Server, use this command:

```
docker run --name=mysql1 -d mysql/mysql-server:8.0
```

To start a new Docker container for the MySQL Enterprise Server with a Docker image downloaded from My Oracle Support, use this command:

```
docker run --name=mysql1 -d mysql/enterprise-server:8.0
```

If the Docker image of the specified name and tag has not been downloaded by an earlier `docker pull` or `docker run` command, the image is now downloaded. Initialization for the container begins, and the container appears in the list of running containers when you run the `docker ps` command. For example:

```
shell> docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED             STATUS
a24888f0d6f4   mysql/mysql-server   "/entrypoint.sh my..."   14 seconds ago      Up 13 seconds (health: starti
```

The container initialization might take some time. When the server is ready for use, the `STATUS` of the container in the output of the `docker ps` command changes from `(health: starting)` to `(healthy)`.

The `-d` option used in the `docker run` command above makes the container run in the background. Use this command to monitor the output from the container:

```
docker logs mysql1
```

Once initialization is finished, the command's output is going to contain the random password generated for the root user; check the password with, for example, this command:

```
shell> docker logs mysql1 2>&1 | grep GENERATED
GENERATED ROOT PASSWORD: Axegh3kAJyDLaRuBemecis&EShOs
```

Connecting to MySQL Server from within the Container

Once the server is ready, you can run the `mysql` client within the MySQL Server container you just started, and connect it to the MySQL Server. Use the `docker exec -it` command to start a `mysql` client inside the Docker container you have started, like the following:

```
docker exec -it mysql1 mysql -uroot -p
```

When asked, enter the generated root password (see the last step in Starting a MySQL Server Instance above on how to find the password). Because the `MYSQL_ONETIME_PASSWORD` option is true by default, after you have connected a `mysql` client to the server, you must reset the server root password by issuing this statement:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'password';
```

Substitute *`password`* with the password of your choice. Once the password is reset, the server is ready for use.

Container Shell Access

To have shell access to your MySQL Server container, use the `docker exec -it` command to start a bash shell inside the container:

```
shell> docker exec -it mysql1 bash
bash-4.2#
```

You can then run Linux commands inside the container. For example, to view contents in the server's data directory inside the container, use this command:

```
bash-4.2# ls /var/lib/mysql
auto.cnf    ca.pem      client-key.pem  ib_logfile0  ibdata1  mysql       mysql.sock.lock    private_key.pe
ca-key.pem  client-cert.pem  ib_buffer_pool  ib_logfile1  ibtmp1   mysql.sock  performance_schema  public_
```

Stopping and Deleting a MySQL Container

To stop the MySQL Server container we have created, use this command:

```
docker stop mysql1
```

`docker stop` sends a SIGTERM signal to the `mysqld` process, so that the server is shut down gracefully.

Also notice that when the main process of a container (`mysqld` in the case of a MySQL Server container) is stopped, the Docker container stops automatically.

To start the MySQL Server container again:

```
docker start mysql1
```

To stop and start again the MySQL Server container with a single command:

```
docker restart mysql1
```

To delete the MySQL container, stop it first, and then use the `docker rm` command:

```
docker stop mysql1
```

```
docker rm mysql1
```

If you want the Docker volume for the server's data directory to be deleted at the same time, add the `-v` option to the `docker rm` command.

Upgrading a MySQL Server Container

Important

- Before performing any upgrade to MySQL, follow carefully the instructions in Section 2.11, “Upgrading MySQL”. Among other instructions discussed there, it is especially important to back up your database before the upgrade.

- The instructions in this section require that the server's data and configuration have been persisted on the host. See Persisting Data and Configuration Changes for details.

Follow these steps to upgrade a Docker installation of MySQL 5.7 to 8.0:

- Stop the MySQL 5.7 server (container name is `mysql57` in this example):

```
docker stop mysql57
```

- Download the MySQL 8.0 Server Docker image. See instructions in Downloading a MySQL Server Docker Image; make sure you use the right tag for MySQL 8.0.

- Start a new MySQL 8.0 Docker container (named `mysql80` in this example) with the old server data and configuration (with proper modifications if needed—see Section 2.11, "Upgrading MySQL") that have been persisted on the host (by bind-mounting in this example). For the MySQL Community Server, run this command:

```
docker run --name=mysql80 \
   --mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
   --mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
   -d mysql/mysql-server:8.0
```

  If needed, adjust `mysql/mysql-server` to the correct repository name—for example, replace it with `mysql/enterprise-server` for MySQL Enterprise Edition images downloaded from My Oracle Support.

- Wait for the server to finish startup. You can check the status of the server using the `docker ps` command (see Starting a MySQL Server Instance for how to do that).

- *For MySQL 8.0.15 and earlier:* Run the mysql_upgrade utility in the MySQL 8.0 Server container (not required for MySQL 8.0.16 and later):

```
docker exec -it mysql80 mysql_upgrade -uroot -p
```

  When prompted, enter the root password for your old MySQL 5.7 Server.

- Finish the upgrade by restarting the MySQL 8.0 Server container:

```
docker restart mysql80
```

More Topics on Deploying MySQL Server with Docker

For more topics on deploying MySQL Server with Docker like server configuration, persisting data and configuration, server error log, and container environment variables, see Section 2.5.6.2, "More Topics on Deploying MySQL Server with Docker".

2.5.6.2 More Topics on Deploying MySQL Server with Docker

Note

Most of the sample commands below have `mysql/mysql-server` as the Docker image repository when that has to be specified (like with the `docker pull` and `docker run` commands); change that if your image is from another repository—for example, replace it with `mysql/enterprise-server` for MySQL Enterprise Edition images downloaded from My Oracle Support.

- The Optimized MySQL Installation for Docker
- Configuring the MySQL Server
- Persisting Data and Configuration Changes
- Running Additional Initialization Scripts
- Connect to MySQL from an Application in Another Docker Container
- Server Error Log
- Using MySQL Enterprise Backup with Docker
- Docker Environment Variables

The Optimized MySQL Installation for Docker

Docker images for MySQL are optimized for code size, which means they only include crucial components that are expected to be relevant for the majority of users who run MySQL instances in Docker containers. A MySQL Docker installation is different from a common, non-Docker installation in the following aspects:

- Included binaries are limited to:
  - `/usr/bin/my_print_defaults`
  - `/usr/bin/mysql`
  - `/usr/bin/mysql_config`
  - `/usr/bin/mysql_install_db`
  - `/usr/bin/mysql_tzinfo_to_sql`
  - `/usr/bin/mysql_upgrade`
  - `/usr/bin/mysqladmin`
  - `/usr/bin/mysqlcheck`
  - `/usr/bin/mysqldump`
  - `/usr/bin/mysqlpump`
  - `/usr/bin/mysqlbackup` (for MySQL Enterprise Edition 8.0 only)
  - `/usr/sbin/mysqld`
- All binaries are stripped; they contain no debug information.

Configuring the MySQL Server

When you start the MySQL Docker container, you can pass configuration options to the server through the `docker run` command. For example:

```
docker run --name mysql1 -d mysql/mysql-server:tag --character-set-server=utf8mb4 --collation-server=utf8m
```

The command starts your MySQL Server with `utf8mb4` as the default character set and `utf8mb4_col` as the default collation for your databases.

Another way to configure the MySQL Server is to prepare a configuration file and mount it at the location of the server configuration file inside the container. See Persisting Data and Configuration Changes for details.

Persisting Data and Configuration Changes

Docker containers are in principle ephemeral, and any data or configuration are expected to be lost if the container is deleted or corrupted (see discussions here). Docker volumes, however, provides a mechanism to persist data created inside a Docker container. At its initialization, the MySQL Server container creates a Docker volume for the server data directory. The JSON output for running the `docker inspect` command on the container has a `Mount` key, whose value provides information on the data directory volume:

```
shell> docker inspect mysql1
...
 "Mounts": [
            {
                "Type": "volume",
                "Name": "4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89f7e845d27652",
                "Source": "/var/lib/docker/volumes/4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89f7e845d
                "Destination": "/var/lib/mysql",
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
...
```

The output shows that the source folder `/var/lib/docker/volumes/4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89f7e845d27652/_data`, in which data is persisted on the host, has been mounted at `/var/lib/mysql`, the server data directory inside the container.

Another way to preserve data is to bind-mount a host directory using the `--mount` option when creating the container. The same technique can be used to persist the configuration of the server. The following command creates a MySQL Server container and bind-mounts both the data directory and the server configuration file:

```
docker run --name=mysql1 \
--mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
-d mysql/mysql-server:tag
```

The command mounts *`path-on-host-machine/my.cnf`* at `/etc/my.cnf` (the server configuration file inside the container), and *`path-on-host-machine/datadir`* at `/var/lib/mysql` (the data directory inside the container). The following conditions must be met for the bind-mounting to work:

- The configuration file *`path-on-host-machine/my.cnf`* must already exist, and it must contain the specification for starting the server using the user `mysql`:

  ```
  [mysqld]
  user=mysql
  ```

  You can also include other server configuration options in the file.

- The data directory *`path-on-host-machine/datadir`* must already exist. For server initialization to happen, the directory must be empty. You can also mount a directory prepopulated with data and start the server with it; however, you must make sure you start the Docker container with the same configuration as the server that created the data, and any host files or directories required are mounted when starting the container.

Running Additional Initialization Scripts

If there are any `.sh` or `.sql` scripts you want to run on the database immediately after it has been created, you can put them into a host directory and then mount the directory at `/docker-entrypoint-initdb.d/` inside the container. For example:

```
docker run --name=mysql1 \
--mount type=bind,src=/path-on-host-machine/scripts/,dst=/docker-entrypoint-initdb.d/ \
-d mysql/mysql-server:tag
```

Connect to MySQL from an Application in Another Docker Container

By setting up a Docker network, you can allow multiple Docker containers to communicate with each other, so that a client application in another Docker container can access the MySQL Server in the server container. First, create a Docker network:

```
docker network create my-custom-net
```

Then, when you are creating and starting the server and the client containers, use the `--network` option to put them on network you created. For example:

```
docker run --name=mysql1 --network=my-custom-net -d mysql/mysql-server
```

```
docker run --name=myapp1 --network=my-custom-net -d myapp
```

The `myapp1` container can then connect to the `mysql1` container with the `mysql1` hostname and vice versa, as Docker automatically sets up a DNS for the given container names. In the following example, we run the `mysql` client from inside the `myapp1` container to connect to host `mysql1` in its own container:

```
docker exec -it myapp1 mysql --host=mysql1 --user=myuser --password
```

For other networking techniques for containers, see the Docker container networking section in the Docker Documentation.

Server Error Log

When the MySQL Server is first started with your server container, a server error log is NOT generated if either of the following conditions is true:

- A server configuration file from the host has been mounted, but the file does not contain the system variable `log_error` (see Persisting Data and Configuration Changes on bind-mounting a server configuration file).
- A server configuration file from the host has not been mounted, but the Docker environment variable `MYSQL_LOG_CONSOLE` is `true` (which is the variable's default state for MySQL 8.0 server containers). The MySQL Server's error log is then redirected to `stderr`, so that the error log goes into the Docker container's log and is viewable using the `docker logs mysqld-container` command.

To make MySQL Server generate an error log when either of the two conditions is true, use the `--log-error` option to configure the server to generate the error log at a specific location inside the container. To persist the error log, mount a host file at the location of the error log inside the container as explained in Persisting Data and Configuration Changes. However, you must make sure your MySQL Server inside its container has write access to the mounted host file.

Using MySQL Enterprise Backup with Docker

MySQL Enterprise Backup is a commercially-licensed backup utility for MySQL Server, available with MySQL Enterprise Edition. MySQL Enterprise Backup is included in the Docker installation of MySQL Enterprise Edition.

In the following example, we assume that you already have a MySQL Server running in a Docker container (see Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker” on how to start a MySQL Server instance with Docker). For MySQL Enterprise Backup to back up the MySQL Server, it must have access to the server's data directory. This can be achieved by, for example, bind-mounting a host directory on the data directory of the MySQL Server when you start the server:

```
docker run --name=mysqlserver \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
-d mysql/enterprise-server:8.0
```

With this command, the MySQL Server is started with a Docker image of the MySQL Enterprise Edition, and the host directory */path-on-host-machine/datadir/* has been mounted onto the server's data directory (`/var/lib/mysql`) inside the server container. We also assume that, after the server has been started, the required privileges have also been set up for MySQL Enterprise Backup to access the server (see Grant MySQL Privileges to Backup Administrator for details). Use the following steps then to backup and restore a MySQL Server instance.

To backup a MySQL Server instance running in a Docker container using MySQL Enterprise Backup with Docker:

1. On the same host where the MySQL Server container is running, start another container with an image of MySQL Enterprise Edition to perform a back up with the MySQL Enterprise Backup command `backup-to-image`. Provide access to the server's data directory using the bind mount we created in the last step. Also, mount a host directory (*/path-on-host-machine/backups/* in this example) onto the storage folder for backups in the container (`/data/backups` in the example) to persist the backups we are creating. Here is a sample command for this step, in which MySQL Enterprise Backup is started with a Docker image downloaded from My Oracle Support):

   ```
   shell> docker run \
   --mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
   --mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
   --rm mysql/enterprise-server:8.0 \
   mysqlbackup -umysqlbackup -ppassword --backup-dir=/tmp/backup-tmp --with-timestamp \
   --backup-image=/data/backups/db.mbi backup-to-image

   [Entrypoint] MySQL Docker Image 8.0.11-1.1.5
   MySQL Enterprise Backup version 8.0.11 Linux-4.1.12-61.1.16.el7uek.x86_64-x86_64 [2018-04-08  07:06:45]
   Copyright (c) 2003, 2018, Oracle and/or its affiliates. All Rights Reserved.

   180921 17:27:25 MAIN    INFO: A thread created with Id '140594390935680'
   180921 17:27:25 MAIN    INFO: Starting with following command line ...
   ...

   ------------------------------------------------------------
      Parameters Summary
   ------------------------------------------------------------
      Start LSN                  : 29615616
      End LSN                    : 29651854
   ------------------------------------------------------------

   mysqlbackup completed OK!
   ```

   It is important to check the end of the output by `mysqlbackup` to make sure the backup has been completed successfully.

2. The container exits once the backup job is finished and, with the `--rm` option used to start it, it is removed after it exits. An image backup has been created, and can be found in the host directory mounted in the last step for storing backups:

   ```
   shell> ls /tmp/backups
   db.mbi
   ```

To restore a MySQL Server instance in a Docker container using MySQL Enterprise Backup with Docker:

1. Stop the MySQL Server container, which also stops the MySQL Server running inside:

```
docker stop mysqlserver
```

2. On the host, delete all contents in the bind mount for the MySQL Server data directory:

```
rm -rf /path-on-host-machine/datadir/*
```

3. Start a container with an image of MySQL Enterprise Edition to perform the restore with the MySQL Enterprise Backup command `copy-back-and-apply-log`. Bind-mount the server's data directory and the storage folder for the backups, like what we did when we backed up the server:

```
shell> docker run \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
--mount type=bind,src=/path-on-host-machine/backups/,dst=/data/backups \
--rm mysql/enterprise-server:8.0 \
mysqlbackup --backup-dir=/tmp/backup-tmp --with-timestamp \
--datadir=/var/lib/mysql --backup-image=/data/backups/db.mbi copy-back-and-apply-log

[Entrypoint] MySQL Docker Image 8.0.11-1.1.5
MySQL Enterprise Backup version 8.0.11 Linux-4.1.12-61.1.16.el7uek.x86_64-x86_64 [2018-04-08  07:06:45]
Copyright (c) 2003, 2018, Oracle and/or its affiliates. All Rights Reserved.

180921 22:06:52 MAIN    INFO: A thread created with Id '139768047519872'
180921 22:06:52 MAIN    INFO: Starting with following command line ...
...
180921 22:06:52 PCR1    INFO: We were able to parse ibbackup_logfile up to
          lsn 29680612.
180921 22:06:52 PCR1    INFO: Last MySQL binlog file position 0 155, file name binlog.000003
180921 22:06:52 PCR1    INFO: The first data file is '/var/lib/mysql/ibdata1'
                              and the new created log files are at '/var/lib/mysql'
180921 22:06:52 MAIN    INFO: No Keyring file to process.
180921 22:06:52 MAIN    INFO: Apply-log operation completed successfully.
180921 22:06:52 MAIN    INFO: Full Backup has been restored successfully.

mysqlbackup completed OK! with 3 warnings
```

   The container exits once the backup job is finished and, with the `--rm` option used when starting it, it is removed after it exits.

4. Restart the server container, which also restarts the restored server:

```
docker restart mysqlserver
```

   Or, start a new MySQL Server on the restored data directory:

```
docker run --name=mysqlserver2 \
--mount type=bind,src=/path-on-host-machine/datadir/,dst=/var/lib/mysql \
-d mysql/enterprise-server:8.0
```

   Log on to the server to check that the server is running with the restored data.

Docker Environment Variables

When you create a MySQL Server container, you can configure the MySQL instance by using the `--env` option (`-e` in short) and specifying one or more of the following environment variables.

Notes

- None of the variables below has any effect if the data directory you mount is not empty, as no server initialization is going to be attempted then (see Persisting Data and Configuration Changes for more details). Any pre-existing contents in the folder, including any old server settings, are not modified during the container startup.
- The boolean variables including `MYSQL_RANDOM_ROOT_PASSWORD`, `MYSQL_ONETIME_PASSWORD`, `MYSQL_ALLOW_EMPTY_PASSWORD`, and `MYSQL_LOG_CONSOLE` are made true by setting them with any strings of nonzero lengths. Therefore, setting them to, for example, “0”, “false”, or “no” does not make them false, but actually makes them true. This is a known issue of the MySQL Server containers.

- `MYSQL_RANDOM_ROOT_PASSWORD`: When this variable is true (which is its default state, unless `MYSQL_ROOT_PASSWORD` is set or `MYSQL_ALLOW_EMPTY_PASSWORD` is set to true), a random password for the server's root user is generated when the Docker container is started. The password is printed to `stdout` of the container and can be found by looking at the container’s log (see Starting a MySQL Server Instance).
- `MYSQL_ONETIME_PASSWORD`: When the variable is true (which is its default state, unless `MYSQL_ROOT_PASSWORD` is set or `MYSQL_ALLOW_EMPTY_PASSWORD` is set to true), the root user's password is set as expired and must be changed before MySQL can be used normally.
- `MYSQL_DATABASE`: This variable allows you to specify the name of a database to be created on image startup. If a user name and a password are supplied with `MYSQL_USER` and `MYSQL_PASSWORD`, the user is created and granted superuser access to this database (corresponding to `GRANT ALL`). The specified database is created by a CREATE DATABASE IF NOT EXIST statement, so that the variable has no effect if the database already exists.
- `MYSQL_USER`, `MYSQL_PASSWORD`: These variables are used in conjunction to create a user and set that user's password, and the user is granted superuser permissions for the database specified by the `MYSQL_DATABASE` variable. Both `MYSQL_USER` and `MYSQL_PASSWORD` are required for a user to be created—if any of the two variables is not set, the other is ignored. If both variables are set but `MYSQL_DATABASE` is not, the user is created without any privileges.

Note

There is no need to use this mechanism to create the root superuser, which is created by default with the password set by either one of the mechanisms discussed in the descriptions for `MYSQL_ROOT_PASSWORD` and `MYSQL_RANDOM_ROOT_PASSWORD`, unless `MYSQL_ALLOW_EMPTY_PASSWORD` is true.

- `MYSQL_ROOT_HOST`: By default, MySQL creates the `'root'@'localhost'` account. This account can only be connected to from inside the container as described in Connecting to MySQL Server from within the Container. To allow root connections from other hosts, set this environment variable. For example, the value `172.17.0.1`, which is the default Docker gateway IP, allows connections from the host machine that runs the container. The option accepts only one entry, but wildcards are allowed (for example, `MYSQL_ROOT_HOST=172.*.*.*` or `MYSQL_ROOT_HOST=%`).
- `MYSQL_LOG_CONSOLE`: When the variable is true (which is its default state for MySQL 8.0 server containers), the MySQL Server's error log is redirected to `stderr`, so that the error log goes into the Docker container's log and is viewable using the `docker logs` *`mysqld-container`* command.

Note

The variable has no effect if a server configuration file from the host has been mounted (see Persisting Data and Configuration Changes on bind-mounting a configuration file).

- `MYSQL_ROOT_PASSWORD`: This variable specifies a password that is set for the MySQL root account.

Warning

Setting the MySQL root user password on the command line is insecure. As an alternative to specifying the password explicitly, you can set the variable with a container file path for a password file, and then mount a file from your host that contains the password at the container file path. This is still not very secure, as the location of the password file is still exposed. It is preferable to use the default settings of `MYSQL_RANDOM_ROOT_PASSWORD` and `MYSQL_ONETIME_PASSWORD` both being true.

- `MYSQL_ALLOW_EMPTY_PASSWORD`. Set it to true to allow the container to be started with a blank password for the root user.

Warning

Setting this variable to true is insecure, because it is going to leave your MySQL instance completely unprotected, allowing anyone to gain complete superuser access. It is preferable to use the default settings of `MYSQL_RANDOM_ROOT_PASSWORD` and `MYSQL_ONETIME_PASSWORD` both being true.

2.5.6.3 Deploying MySQL on Windows and Other Non-Linux Platforms with Docker

Warning

The MySQL Docker images provided by Oracle are built specifically for Linux platforms. Other platforms are not supported, and users running the MySQL Docker images from Oracle on them are doing so at their own risk. This section discusses some known issues for the images when used on non-Linux platforms.

Known Issues for using the MySQL Server Docker images from Oracle on Windows include:

- If you are bind-mounting on the container's MySQL data directory (see Persisting Data and Configuration Changes for details), you have to set the location of the server socket file with the `--socket` option to somewhere outside of the MySQL data directory; otherwise, the server will fail to start. This is because the way Docker for Windows handles file mounting does not allow a host file from being bind-mounted on the socket file.

2.5.7 Installing MySQL on Linux from the Native Software Repositories

Many Linux distributions include a version of the MySQL server, client tools, and development components in their native software repositories and can be installed with the platforms' standard package management systems. This section provides basic instructions for installing MySQL using those package management systems.

Important

Native packages are often several versions behind the currently available release. You will also normally be unable to install development milestone releases (DMRs), as these are not usually made available in the native repositories. Before proceeding, we recommend that you check out the other installation options described in Section 2.5, “Installing MySQL on Linux”.

Distribution specific instructions are shown below:

- **Red Hat Linux, Fedora, CentOS**

Note

For a number of Linux distributions, you can install MySQL using the MySQL Yum repository instead of the platform's native software repository. See Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details.

For Red Hat and similar distributions, the MySQL distribution is divided into a number of separate packages, `mysql` for the client tools, `mysql-server` for the server and associated tools, and `mysql-libs` for the libraries. The libraries are required if you want to provide connectivity from different languages and environments such as Perl, Python and others.

To install, use the `yum` command to specify the packages that you want to install. For example:

```
root-shell> yum install mysql mysql-server mysql-libs mysql-server
Loaded plugins: presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package mysql.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-libs.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-server.x86_64 0:5.1.48-2.fc13 set to be updated
--> Processing Dependency: perl-DBD-MySQL for package: mysql-server-5.1.48-2.fc13.x86_64
--> Running transaction check
---> Package perl-DBD-MySQL.x86_64 0:4.017-1.fc13 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package               Arch          Version               Repository      Size
================================================================================
Installing:
 mysql                 x86_64        5.1.48-2.fc13         updates        889 k
 mysql-libs            x86_64        5.1.48-2.fc13         updates        1.2 M
 mysql-server          x86_64        5.1.48-2.fc13         updates        8.1 M
Installing for dependencies:
 perl-DBD-MySQL        x86_64        4.017-1.fc13          updates        136 k

Transaction Summary
================================================================================
Install       4 Package(s)
Upgrade       0 Package(s)

Total download size: 10 M
Installed size: 30 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
Processing delta metadata
```

```
Package(s) data still to download: 10 M
(1/4): mysql-5.1.48-2.fc13.x86_64.rpm                     | 889 kB     00:04
(2/4): mysql-libs-5.1.48-2.fc13.x86_64.rpm                | 1.2 MB     00:06
(3/4): mysql-server-5.1.48-2.fc13.x86_64.rpm              | 8.1 MB     00:40
(4/4): perl-DBD-MySQL-4.017-1.fc13.x86_64.rpm             | 136 kB     00:00
--------------------------------------------------------------------------------
Total                                          201 kB/s |  10 MB     00:52
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : mysql-libs-5.1.48-2.fc13.x86_64                         1/4
  Installing     : mysql-5.1.48-2.fc13.x86_64                              2/4
  Installing     : perl-DBD-MySQL-4.017-1.fc13.x86_64                      3/4
  Installing     : mysql-server-5.1.48-2.fc13.x86_64                       4/4

Installed:
  mysql.x86_64 0:5.1.48-2.fc13            mysql-libs.x86_64 0:5.1.48-2.fc13
  mysql-server.x86_64 0:5.1.48-2.fc13

Dependency Installed:
  perl-DBD-MySQL.x86_64 0:4.017-1.fc13

Complete!
```

MySQL and the MySQL server should now be installed. A sample configuration file is installed into `/etc/my.cnf`. An init script, to start and stop the server, will have been installed into `/etc/init.d/mysqld`. To start the MySQL server use `service`:

```
root-shell> service mysqld start
```

To enable the server to be started and stopped automatically during boot, use `chkconfig`:

```
root-shell> chkconfig --levels 235 mysqld on
```

Which enables the MySQL server to be started (and stopped) automatically at the specified the run levels.

The database tables will have been automatically created for you, if they do not already exist. You should, however, run `mysql_secure_installation` to set the root passwords on your server.

- **Debian, Ubuntu, Kubuntu**

Note

For supported Debian and Ubuntu versions, MySQL can be installed using the MySQL APT Repository instead of the platform's native software repository. See Section 2.5.2, “Installing MySQL on Linux Using the MySQL APT Repository” for details.

On Debian and related distributions, there are two packages for MySQL in their software repositories, `mysql-client` and `mysql-server`, for the client and server components respectively. You should specify an explicit version, for example `mysql-client-5.1`, to ensure that you install the version of MySQL that you want.

To download and install, including any dependencies, use the `apt-get` command, specifying the packages that you want to install.

Note

Before installing, make sure that you update your `apt-get` index files to ensure you are downloading the latest available version.

A sample installation of the MySQL packages might look like this (some sections trimmed for clarity):

```
root-shell> apt-get install mysql-client-5.1 mysql-server-5.1
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-2.6.28-11 linux-headers-2.6.28-11-generic
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
  bsd-mailx libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient15off libmysqlclient16 libnet-daemon-perl libplrpc-perl mailx
  mysql-common postfix
Suggested packages:
  dbishell libipc-sharedcache-perl tinyca procmail postfix-mysql postfix-pgsql
  postfix-ldap postfix-pcre sasl2-bin resolvconf postfix-cdb
The following NEW packages will be installed
  bsd-mailx libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient15off libmysqlclient16 libnet-daemon-perl libplrpc-perl mailx
  mysql-client-5.1 mysql-common mysql-server-5.1 postfix
0 upgraded, 13 newly installed, 0 to remove and 182 not upgraded.
Need to get 1907kB/25.3MB of archives.
After this operation, 59.5MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get: 1 http://gb.archive.ubuntu.com jaunty-updates/main mysql-common 5.1.30really5.0.75-0ubuntu10.5 [63.6kB]
Get: 2 http://gb.archive.ubuntu.com jaunty-updates/main libmysqlclient15off 5.1.30really5.0.75-0ubuntu10.5 [
Fetched 1907kB in 9s (205kB/s)
Preconfiguring packages ...
Selecting previously deselected package mysql-common.
(Reading database ... 121260 files and directories currently installed.)
...
Processing 1 added doc-base file(s)...
Registering documents with scrollkeeper...
Setting up libnet-daemon-perl (0.43-1) ...
Setting up libplrpc-perl (0.2020-1) ...
Setting up libdbi-perl (1.607-1) ...
Setting up libmysqlclient15off (5.1.30really5.0.75-0ubuntu10.5) ...

Setting up libdbd-mysql-perl (4.008-1) ...
Setting up libmysqlclient16 (5.1.31-1ubuntu2) ...

Setting up mysql-client-5.1 (5.1.31-1ubuntu2) ...

Setting up mysql-server-5.1 (5.1.31-1ubuntu2) ...
 * Stopping MySQL database server mysqld
   ...done.
2013-09-24T13:03:09.048353Z 0 [Note] InnoDB: 8.0.19 started; log sequence number 1566036
2013-09-24T13:03:10.057269Z 0 [Note] InnoDB: Starting shutdown...
2013-09-24T13:03:10.857032Z 0 [Note] InnoDB: Shutdown completed; log sequence number 1566036
 * Starting MySQL database server mysqld
   ...done.
 * Checking for corrupt, not cleanly closed and upgrade needing tables.
...
Processing triggers for libc6 ...
ldconfig deferred processing now taking place
```

Note

The `apt-get` command will install a number of packages, including the MySQL server, in order to provide the typical tools and application environment. This can mean that you install a large number of packages in addition to the main MySQL package.

During installation, the initial database will be created, and you will be prompted for the MySQL root password (and confirmation). A configuration file will have been created in `/etc/mysql/my.cnf`. An init script will have been created in `/etc/init.d/mysql`.

The server will already be started. You can manually start and stop the server using:

```
root-shell> service mysql [start|stop]
```

The service will automatically be added to the 2, 3 and 4 run levels, with stop scripts in the single, shutdown and restart levels.

2.5.8 Installing MySQL on Linux with Juju

The Juju deployment framework supports easy installation and configuration of MySQL servers. For instructions, see https://jujucharms.com/mysql/.

2.5.9 Managing MySQL Server with systemd

If you install MySQL using an RPM or Debian package on the following Linux platforms, server startup and shutdown is managed by systemd:

- RPM package platforms:
  - Red Hat Enterprise Linux 7; Oracle Linux 7; CentOS 7
  - SUSE Linux Enterprise Server 12
  - Fedora 29 and 30
- Debian package platforms:
  - Debian 8 or higher
  - Ubuntu 16 or higher

If you install MySQL from a generic binary distribution on a platform that uses systemd, you can manually configure systemd support for MySQL following the instructions provided in the post-installation setup section of the MySQL 8.0 Secure Deployment Guide.

If you install MySQL from a source distribution on a platform that uses systemd, obtain systemd support for MySQL by configuring the distribution using the `-DWITH_SYSTEMD=1` `CMake` option. See Section 2.9.7, “MySQL Source-Configuration Options”.

The following discussion covers these topics:

- Overview of systemd
- Configuring systemd for MySQL

- Configuring Multiple MySQL Instances Using systemd
- Migrating from mysqld_safe to systemd

Note

On platforms for which systemd support for MySQL is installed, scripts such as `mysqld_safe` and the System V initialization script are unnecessary and are not installed. For example, `mysqld_safe` can handle server restarts, but systemd provides the same capability, and does so in a manner consistent with management of other services rather than by using an application-specific program.

Because systemd has the capability of managing multiple MySQL instances on platforms for which systemd support for MySQL is installed, `mysqld_multi` and `mysqld_multi.server` are unnecessary and are not installed.

Overview of systemd

systemd provides automatic MySQL server startup and shutdown. It also enables manual server management using the `systemctl` command. For example:

```
systemctl {start|stop|restart|status} mysqld
```

Alternatively, use the `service` command (with the arguments reversed), which is compatible with System V systems:

```
service mysqld {start|stop|restart|status}
```

Note

For the `systemctl` or `service` commands, if the MySQL service name is not `mysqld`, use the appropriate name. For example, use `mysql` rather than `mysqld` on Debian-based and SLES systems.

Support for systemd includes these files:

- `mysqld.service` (RPM platforms), `mysql.service` (Debian platforms): systemd service unit configuration file, with details about the MySQL service.
- `mysqld@.service` (RPM platforms), `mysql@.service` (Debian platforms): Like `mysqld.service` or `mysql.service`, but used for managing multiple MySQL instances.
- `mysqld.tmpfiles.d`: File containing information to support the `tmpfiles` feature. This file is installed under the name `mysql.conf`.
- `mysqld_pre_systemd` (RPM platforms), `mysql-system-start` (Debian platforms): Support script for the unit file. This script assists in creating the error log file only if the log location matches a pattern (`/var/log/mysql*.log` for RPM platforms, `/var/log/mysql/*.log` for Debian platforms). In other cases, the error log directory must be writable or the error log must be present and writable for the user running the `mysqld` process.

Configuring systemd for MySQL

To add or change systemd options for MySQL, these methods are available:

- Use a localized systemd configuration file.

- Arrange for systemd to set environment variables for the MySQL server process.
- Set the `MYSQLD_OPTS` systemd variable.

To use a localized systemd configuration file, create the `/etc/systemd/system/mysqld.service.d` directory if it does not exist. In that directory, create a file that contains a `[Service]` section listing the desired settings. For example:

```
[Service]
LimitNOFILE=max_open_files
PIDFile=/path/to/pid/file
Nice=nice_level
LimitCore=core_file_limit
Environment="LD_PRELOAD=/path/to/malloc/library"
Environment="TZ=time_zone_setting"
```

The discussion here uses `override.conf` as the name of this file. Newer versions of systemd support the following command, which opens an editor and permits you to edit the file:

```
systemctl edit mysqld  # RPM platforms
systemctl edit mysql   # Debian platforms
```

Whenever you create or change `override.conf`, reload the systemd configuration, then tell systemd to restart the MySQL service:

```
systemctl daemon-reload
systemctl restart mysqld  # RPM platforms
systemctl restart mysql   # Debian platforms
```

With systemd, the `override.conf` configuration method must be used for certain parameters, rather than settings in a `[mysqld]` or `[mysqld_safe]` group in a MySQL option file:

- For some parameters, `override.conf` must be used because systemd itself must know their values and it cannot read MySQL option files to get them.
- Parameters that specify values otherwise settable only using options known to `mysqld_safe` must be specified using systemd because there is no corresponding `mysqld` parameter.

For additional information about using systemd rather than `mysqld_safe`, see Migrating from mysqld_safe to systemd.

You can set the following parameters in `override.conf`:

- To specify the process ID file, use `override.conf` and change both `PIDFile` and `ExecStart` to name the PID file path name. Any setting of the process ID file in MySQL option files is ignored. To modify `ExecStart`, it must first be cleared. For example:

  ```
  [Service]
  PIDFile=/var/run/mysqld/mysqld-custom.pid
  ExecStart=
  ExecStart=/usr/sbin/mysqld --pid-file=/var/run/mysqld/mysqld-custom.pid $MYSQLD_OPTS
  ```

- To set the number of file descriptors available to the MySQL server, use `LimitNOFILE` in `override.conf` rather than the `open_files_limit` system variable for `mysqld` or `--open-files-limit` option for `mysqld_safe`.
- To set the maximum core file size, use `LimitCore` in `override.conf` rather than the `--core-file-size` option for `mysqld_safe`.

- To set the scheduling priority for the MySQL server, use `Nice` in `override.conf` rather than the `--nice` option for `mysqld_safe`.

Some MySQL parameters are configured using environment variables:

- `LD_PRELOAD`: Set this variable if the MySQL server should use a specific memory-allocation library.
- `NOTIFY_SOCKET`: This environment variable specifies the socket that `mysqld` uses to communicate notification of startup completion and service status change with systemd. It is set by systemd when the `mysqld` service is started. The `mysqld` service reads the variable setting and writes to the defined location. If the variable is not set, `mysqld` writes nothing.
- `TZ`: Set this variable to specify the default time zone for the server.

There are multiple ways to specify environment variable values for use by the MySQL server process managed by systemd:

- Use `Environment` lines in the `override.conf` file. For the syntax, see the example in the preceding discussion that describes how to use this file.
- Specify the values in the `/etc/sysconfig/mysql` file (create the file if it does not exist). Assign values using the following syntax:

```
LD_PRELOAD=/path/to/malloc/library
TZ=time_zone_setting
```

  After modifying `/etc/sysconfig/mysql`, restart the server to make the changes effective:

```
systemctl restart mysqld  # RPM platforms
systemctl restart mysql   # Debian platforms
```

To specify options for `mysqld` without modifying systemd configuration files directly, set or unset the `MYSQLD_OPTS` systemd variable. For example:

```
systemctl set-environment MYSQLD_OPTS="--general_log=1"
systemctl unset-environment MYSQLD_OPTS
```

`MYSQLD_OPTS` can also be set in the `/etc/sysconfig/mysql` file.

After modifying the systemd environment, restart the server to make the changes effective:

```
systemctl restart mysqld  # RPM platforms
systemctl restart mysql   # Debian platforms
```

For platforms that use systemd, the data directory is initialized if empty at server startup. This might be a problem if the data directory is a remote mount that has temporarily disappeared: The mount point would appear to be an empty data directory, which then would be initialized as a new data directory. To suppress this automatic initialization behavior, specify the following line in the `/etc/sysconfig/mysql` file (create the file if it does not exist):

```
NO_INIT=true
```

Configuring Multiple MySQL Instances Using systemd

This section describes how to configure systemd for multiple instances of MySQL.

Note

Because systemd has the capability of managing multiple MySQL instances on platforms for which systemd support is installed, `mysqld_multi` and `mysqld_multi.server` are unnecessary and are not installed.

To use multiple-instance capability, modify the `my.cnf` option file to include configuration of key options for each instance. These file locations are typical:

- `/etc/my.cnf` or `/etc/mysql/my.cnf` (RPM platforms)
- `/etc/mysql/mysql.conf.d/mysqld.cnf` (Debian platforms)

For example, to manage two instances named `replica01` and `replica02`, add something like this to the option file:

RPM platforms:

```
[mysqld@replica01]
datadir=/var/lib/mysql-replica01
socket=/var/lib/mysql-replica01/mysql.sock
port=3307
log-error=/var/log/mysqld-replica01.log

[mysqld@replica02]
datadir=/var/lib/mysql-replica02
socket=/var/lib/mysql-replica02/mysql.sock
port=3308
log-error=/var/log/mysqld-replica02.log
```

Debian platforms:

```
[mysqld@replica01]
datadir=/var/lib/mysql-replica01
socket=/var/lib/mysql-replica01/mysql.sock
port=3307
log-error=/var/log/mysql/replica01.log

[mysqld@replica02]
datadir=/var/lib/mysql-replica02
socket=/var/lib/mysql-replica02/mysql.sock
port=3308
log-error=/var/log/mysql/replica02.log
```

The replica names shown here use `@` as the delimiter because that is the only delimiter supported by systemd.

Instances then are managed by normal systemd commands, such as:

```
systemctl start mysqld@replica01
systemctl start mysqld@replica02
```

To enable instances to run at boot time, do this:

```
systemctl enable mysqld@replica01
systemctl enable mysqld@replica02
```

Use of wildcards is also supported. For example, this command displays the status of all replica instances:

```
systemctl status 'mysqld@replica*'
```

For management of multiple MySQL instances on the same machine, systemd automatically uses a different unit file:

- `mysqld@.service` rather than `mysqld.service` (RPM platforms)
- `mysql@.service` rather than `mysql.service` (Debian platforms)

In the unit file, `%I` and `%i` reference the parameter passed in after the `@` marker and are used to manage the specific instance. For a command such as this:

```
systemctl start mysqld@replica01
```

systemd starts the server using a command such as this:

```
mysqld --defaults-group-suffix=@%I ...
```

The result is that the `[server]`, `[mysqld]`, and `[mysqld@replica01]` option groups are read and used for that instance of the service.

Note

On Debian platforms, AppArmor prevents the server from reading or writing `/var/lib/mysql-replica*`, or anything other than the default locations. To address this, you must customize or disable the profile in `/etc/apparmor.d/usr.sbin.mysqld`.

Note

On Debian platforms, the packaging scripts for MySQL uninstallation cannot currently handle `mysqld@` instances. Before removing or upgrading the package, you must stop any extra instances manually first.

Migrating from mysqld_safe to systemd

Because `mysqld_safe` is not installed on platforms that use systemd to manage MySQL, options previously specified for that program (for example, in an `[mysqld_safe]` option group) must be specified another way:

- Some `mysqld_safe` options are also understood by `mysqld` and can be moved from the `[mysqld_safe]` option group to the `[mysqld]` group. This does *not* include `--pid-file`, `--open-files-limit`, or `--nice`. To specify those options, use the `override.conf` systemd file, described previously.
- For some `mysqld_safe` options, there are alternative `mysqld` procedures. For example, the `mysqld_safe` option for enabling `syslog` logging is `--syslog`, which is deprecated. To write error log output to the system log, use the instructions at Section 5.4.2.7, “Error Logging to the System Log”.
- `mysqld_safe` options not understood by `mysqld` can be specified in `override.conf` or environment variables. For example, with `mysqld_safe`, if the server should use a specific memory allocation library, this is specified using the `--malloc-lib` option. For installations that manage the server with systemd, arrange to set the `LD_PRELOAD` environment variable instead, as described previously.

2.6 Installing MySQL Using Unbreakable Linux Network (ULN)

Linux supports a number of different solutions for installing MySQL, covered in Section 2.5, “Installing MySQL on Linux”. One of the methods, covered in this section, is installing from Oracle's Unbreakable Linux Network (ULN). You can find information about Oracle Linux and ULN under http://linux.oracle.com/.

To use ULN, you need to obtain a ULN login and register the machine used for installation with ULN. This is described in detail in the ULN FAQ. The page also describes how to install and update packages. The MySQL packages are in the “MySQL for Oracle Linux 6” and “MySQL for Oracle Linux 7” channels for your system architecture on ULN.

Note

At the time of this writing, ULN provides MySQL 8.0 for Oracle Linux 6 and Oracle Linux 7.

Once MySQL has been installed using ULN, you can find information on starting and stopping the server, and more, in this section, particularly under Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”.

If you're updating an existing MySQL installation to an installation using ULN, the recommended procedure is to export your data using `mysqldump`, remove the existing installation, install MySQL from ULN, and load the exported data into your freshly installed MySQL.

If the existing MySQL installation you're upgrading from is from a previous release series (prior to MySQL 8.0), make sure to read the section on upgrading MySQL, Section 2.11, “Upgrading MySQL”.

2.7 Installing MySQL on Solaris

Note

MySQL 8.0 supports Solaris 11.4 and higher

MySQL on Solaris is available in a number of different formats.

- For information on installing using the native Solaris PKG format, see Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”.
- To use a standard `tar` binary installation, use the notes provided in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”. Check the notes and hints at the end of this section for Solaris specific notes that you may need before or after installation.

Important

The installation packages have a dependency on the Oracle Developer Studio 12.6 Runtime Libraries, which must be installed before you run the MySQL installation package. See the download options for Oracle Developer Studio here. The installation package enables you to install the runtime libraries only instead of the full Oracle Developer Studio; see instructions in Installing Only the Runtime Libraries on Oracle Solaris 11.

To obtain a binary MySQL distribution for Solaris in tarball or PKG format, https://dev.mysql.com/downloads/mysql/8.0.html.

Additional notes to be aware of when installing and using MySQL on Solaris:

- If you want to use MySQL with the `mysql` user and group, use the `groupadd` and `useradd` commands:

```
groupadd mysql
useradd -g mysql -s /bin/false mysql
```

- If you install MySQL using a binary tarball distribution on Solaris, because the Solaris `tar` cannot handle long file names, use GNU `tar` (`gtar`) to unpack the distribution. If you do not have GNU `tar` on your system, install it with the following command:

```
pkg install archiver/gnu-tar
```

- You should mount any file systems on which you intend to store `InnoDB` files with the `forcedirectio` option. (By default mounting is done without this option.) Failing to do so will cause a significant drop in performance when using the `InnoDB` storage engine on this platform.

- If you would like MySQL to start automatically, you can copy `support-files/mysql.server` to `/etc/init.d` and create a symbolic link to it named `/etc/rc3.d/S99mysql.server`.

- If too many processes try to connect very rapidly to `mysqld`, you should see this error in the MySQL log:

```
Error in accept: Protocol error
```

  You might try starting the server with the `--back_log=50` option as a workaround for this.

- To configure the generation of core files on Solaris you should use the `coreadm` command. Because of the security implications of generating a core on a `setuid()` application, by default, Solaris does not support core files on `setuid()` programs. However, you can modify this behavior using `coreadm`. If you enable `setuid()` core files for the current user, they will be generated using the mode 600 and owned by the superuser.

2.7.1 Installing MySQL on Solaris Using a Solaris PKG

You can install MySQL on Solaris using a binary package of the native Solaris PKG format instead of the binary tarball distribution.

Important

The installation package has a dependency on the Oracle Developer Studio 12.6 Runtime Libraries, which must be installed before you run the MySQL installation package. See the download options for Oracle Developer Studio here. The installation package enables you to install the runtime libraries only instead of the full Oracle Developer Studio; see instructions in Installing Only the Runtime Libraries on Oracle Solaris 11.

To use this package, download the corresponding `mysql-VERSION-solaris11-PLATFORM.pkg.gz` file, then uncompress it. For example:

```
shell> gunzip mysql-8.0.19-solaris11-x86_64.pkg.gz
```

To install a new package, use `pkgadd` and follow the onscreen prompts. You must have root privileges to perform this operation:

```
shell> pkgadd -d mysql-8.0.19-solaris11-x86_64.pkg

The following packages are available:
  1  mysql     MySQL Community Server (GPL)
               (i86pc) 8.0.19

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]:
```

The PKG installer installs all of the files and tools needed, and then initializes your database if one does not exist. To complete the installation, you should set the root password for MySQL as provided in the instructions at the end of the installation. Alternatively, you can run the `mysql_secure_installation` script that comes with the installation.

By default, the PKG package installs MySQL under the root path `/opt/mysql`. You can change only the installation root path when using `pkgadd`, which can be used to install MySQL in a different Solaris zone. If you need to install in a specific directory, use a binary `tar` file distribution.

The `pkg` installer copies a suitable startup script for MySQL into `/etc/init.d/mysql`. To enable MySQL to startup and shutdown automatically, you should create a link between this file and the init script directories. For example, to ensure safe startup and shutdown of MySQL you could use the following commands to add the right links:

```
shell> ln /etc/init.d/mysql /etc/rc3.d/S91mysql
shell> ln /etc/init.d/mysql /etc/rc0.d/K02mysql
```

To remove MySQL, the installed package name is `mysql`. You can use this in combination with the `pkgrm` command to remove the installation.

To upgrade when using the Solaris package file format, you must remove the existing installation before installing the updated package. Removal of the package does not delete the existing database information, only the server, binaries and support files. The typical upgrade sequence is therefore:

```
shell> mysqladmin shutdown
shell> pkgrm mysql
shell> pkgadd -d mysql-8.0.19-solaris11-x86_64.pkg
shell> mysqld_safe &
shell> mysql_upgrade   # prior to MySQL 8.0.16 only
```

You should check the notes in Section 2.11, “Upgrading MySQL” before performing any upgrade.

2.8 Installing MySQL on FreeBSD

This section provides information about installing MySQL on variants of FreeBSD Unix.

You can install MySQL on FreeBSD by using the binary distribution provided by Oracle. For more information, see Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

The easiest (and preferred) way to install MySQL is to use the `mysql-server` and `mysql-client` ports available at http://www.freebsd.org/. Using these ports gives you the following benefits:

- A working MySQL with all optimizations enabled that are known to work on your version of FreeBSD.
- Automatic configuration and build.
- Startup scripts installed in `/usr/local/etc/rc.d`.
- The ability to use `pkg_info -L` to see which files are installed.
- The ability to use `pkg_delete` to remove MySQL if you no longer want it on your machine.

The MySQL build process requires GNU make (`gmake`) to work. If GNU `make` is not available, you must install it first before compiling MySQL.

Note

Prerequisite libraries as per `ldd mysqld`: libthr, libcrypt, libm, librt, libexecinfo, libunwind, and libssl.

To install using the ports system:

```
# cd /usr/ports/databases/mysql80-server
# make
...
# cd /usr/ports/databases/mysql80-client
# make
...
```

The standard port installation places the server into `/usr/local/libexec/mysqld`, with the startup script for the MySQL server placed in `/usr/local/etc/rc.d/mysql-server`.

Some additional notes on the BSD implementation:

- To remove MySQL after installation using the ports system:

```
# cd /usr/ports/databases/mysql80-server
# make deinstall
...
# cd /usr/ports/databases/mysql80-client
# make deinstall
...
```

- If you get problems with the current date in MySQL, setting the `TZ` variable should help. See Section 4.9, “MySQL Environment Variables”.

2.9 Installing MySQL from Source

Building MySQL from the source code enables you to customize build parameters, compiler optimizations, and installation location. For a list of systems on which MySQL is known to run, see https://www.mysql.com/support/supportedplatforms/database.html.

Before you proceed with an installation from source, check whether Oracle produces a precompiled binary distribution for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options for optimal performance. Instructions for installing binary distributions are available in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

Warning

Building MySQL with nonstandard options may lead to reduced functionality, performance, or security.

The MySQL source code contains internal documentation written using Doxygen. The generated Doxygen content is available at https://dev.mysql.com/doc/index-other.html. It is also possible to generate this content locally from a MySQL source distribution using the instructions at Section 2.9.10, “Generating MySQL Doxygen Documentation Content”.

2.9.1 Source Installation Methods

There are two methods for installing MySQL from source:

- Use a standard MySQL source distribution. To obtain a standard distribution, see Section 2.1.2, “How to Get MySQL”. For instructions on building from a standard distribution, see Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”.

  Standard distributions are available as compressed `tar` files, Zip archives, or RPM packages. Distribution files have names of the form `mysql-`*`VERSION`*`.tar.gz`, `mysql-`*`VERSION`*`.zip`, or

`mysql-VERSION.rpm`, where *`VERSION`* is a number like `8.0.19`. File names for source distributions can be distinguished from those for precompiled binary distributions in that source distribution names are generic and include no platform name, whereas binary distribution names include a platform name indicating the type of system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

- Use a MySQL development tree. For information on building from one of the development trees, see Section 2.9.5, “Installing MySQL Using a Development Source Tree”.

2.9.2 Source Installation Prerequisites

Installation of MySQL from source requires several development tools. Some of these tools are needed no matter whether you use a standard source distribution or a development source tree. Other tool requirements depend on which installation method you use.

To install MySQL from source, the following system requirements must be satisfied, regardless of installation method:

- `CMake`, which is used as the build framework on all platforms. `CMake` can be downloaded from http://www.cmake.org.
- A good `make` program. Although some platforms come with their own `make` implementations, it is highly recommended that you use GNU `make` 3.75 or higher. It may already be available on your system as `gmake`. GNU `make` is available from http://www.gnu.org/software/make/.
- MySQL 8.0 source code permits use of C++14 features. To enable a good level of C++14 support across all supported platforms, the following minimum compiler versions apply:
  - GCC 5.3 (Linux)
  - Clang 4.0 (FreeBSD)
  - XCode 9 (MacOS)
  - Developer Studio 12.6 (Solaris)
  - Visual Studio 2017 (Windows)
- The MySQL C API requires a C++ or C99 compiler to compile.
- An SSL library is required for support of encrypted connections, entropy for random number generation, and other encryption-related operations. By default, the build uses the OpenSSL library installed on the host system. To specify the library explicitly, use the `WITH_SSL` option when you invoke `CMake`. For additional information, see Section 2.9.6, “Configuring SSL Library Support”.
- The Boost C++ libraries are required to build MySQL (but not to use it). MySQL compilation requires a particular Boost version. Typically, that is the current Boost version, but if a specific MySQL source distribution requires a different version, the configuration process will stop with a message indicating the Boost version that it requires. To obtain Boost and its installation instructions, visit the official site. After Boost is installed, tell the build system where the Boost files are located by defining the `WITH_BOOST` option when you invoke `CMake`. For example:

  ```
  cmake . -DWITH_BOOST=/usr/local/boost_version_number
  ```

  Adjust the path as necessary to match your installation.
- The ncurses library.

- Sufficient free memory. If you encounter problems such as “internal compiler error” when compiling large source files, it may be that you have too little memory. If compiling on a virtual machine, try increasing the memory allocation.
- Perl is needed if you intend to run test scripts. Most Unix-like systems include Perl. On Windows, you can use a version such as ActiveState Perl.

To install MySQL from a standard source distribution, one of the following tools is required to unpack the distribution file:

- For a `.tar.gz` compressed `tar` file: GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it. If your `tar` program supports the `z` option, it can both uncompress and unpack the file.

  GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.
- For a `.zip` Zip archive: `WinZip` or another tool that can read `.zip` files.
- For an `.rpm` RPM package: The `rpmbuild` program used to build the distribution unpacks it.

To install MySQL from a development source tree, the following additional tools are required:

- The Git revision control system is required to obtain the development source code. The GitHub Help provides instructions for downloading and installing Git on different platforms. MySQL officially joined GitHub in September, 2014. For more information about MySQL's move to GitHub, refer to the announcement on the MySQL Release Engineering blog: MySQL on GitHub
- `bison` 2.1 or higher, available from http://www.gnu.org/software/bison/. (Version 1 is no longer supported.) Use the latest version of `bison` where possible; if you experience problems, upgrade to a later version, rather than revert to an earlier one.

  `bison` is available from http://www.gnu.org/software/bison/. `bison` for Windows can be downloaded from http://gnuwin32.sourceforge.net/packages/bison.htm. Download the package labeled “Complete package, excluding sources”. On Windows, the default location for `bison` is the `C:\Program Files\GnuWin32` directory. Some utilities may fail to find `bison` because of the space in the directory name. Also, Visual Studio may simply hang if there are spaces in the path. You can resolve these problems by installing into a directory that does not contain a space (for example `C:\GnuWin32`).
- On Solaris Express, `m4` must be installed in addition to `bison`. `m4` is available from http://www.gnu.org/software/m4/.

Note

If you have to install any programs, modify your `PATH` environment variable to include any directories in which the programs are located. See Section 4.2.7, “Setting Environment Variables”.

If you run into problems and need to file a bug report, please use the instructions in Section 1.7, “How to Report Bugs or Problems”.

2.9.3 MySQL Layout for Source Installation

By default, when you install MySQL after compiling it from source, the installation step installs files under `/usr/local/mysql`. The component locations under the installation directory are the same as for binary distributions. See Table 2.3, “MySQL Installation Layout for Generic Unix/Linux Binary Package”, and

Section 2.3.1, “MySQL Installation Layout on Microsoft Windows”. To configure installation locations different from the defaults, use the options described at Section 2.9.7, “MySQL Source-Configuration Options”.

2.9.4 Installing MySQL Using a Standard Source Distribution

To install MySQL from a standard source distribution:

1. Verify that your system satisfies the tool requirements listed at Section 2.9.2, “Source Installation Prerequisites”.
2. Obtain a distribution file using the instructions in Section 2.1.2, “How to Get MySQL”.
3. Configure, build, and install the distribution using the instructions in this section.
4. Perform postinstallation procedures using the instructions in Section 2.10, “Postinstallation Setup and Testing”.

MySQL uses `CMake` as the build framework on all platforms. The instructions given here should enable you to produce a working installation. For additional information on using `CMake` to build MySQL, see How to Build MySQL Server with CMake.

If you start from a source RPM, use the following command to make a binary RPM that you can install. If you do not have `rpmbuild`, use `rpm` instead.

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

The result is one or more binary RPM packages that you install as indicated in Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”.

The sequence for installation from a compressed `tar` file or Zip archive source distribution is similar to the process for installing from a generic binary distribution (see Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”), except that it is used on all platforms and includes steps to configure and compile the distribution. For example, with a compressed `tar` file source distribution on Unix, the basic installation command sequence looks like this:

```
# Preconfiguration setup
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
# Beginning of source-build specific instructions
shell> tar zxvf mysql-VERSION.tar.gz
shell> cd mysql-VERSION
shell> mkdir bld
shell> cd bld
shell> cmake ..
shell> make
shell> make install
# End of source-build specific instructions
# Postinstallation setup
shell> cd /usr/local/mysql
shell> mkdir mysql-files
shell> chown mysql:mysql mysql-files
shell> chmod 750 mysql-files
shell> bin/mysqld --initialize --user=mysql
shell> bin/mysql_ssl_rsa_setup
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

A more detailed version of the source-build specific instructions is shown following.

Note

The procedure shown here does not set up any passwords for MySQL accounts. After following the procedure, proceed to Section 2.10, “Postinstallation Setup and Testing”, for postinstallation setup and testing.

- Perform Preconfiguration Setup
- Obtain and Unpack the Distribution
- Configure the Distribution
- Build the Distribution
- Install the Distribution
- Perform Postinstallation Setup

Perform Preconfiguration Setup

On Unix, set up the `mysql` user and group that will be used to run and execute the MySQL server and own the database directory. For details, see Create a mysql User and Group. Then perform the following steps as the `mysql` user, except as noted.

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it.

Obtain a distribution file using the instructions in Section 2.1.2, “How to Get MySQL”.

Unpack the distribution into the current directory:

- To unpack a compressed `tar` file, `tar` can uncompress and unpack the distribution if it has `z` option support:

  ```
  shell> tar zxvf mysql-VERSION.tar.gz
  ```

  If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it:

  ```
  shell> gunzip < mysql-VERSION.tar.gz | tar xvf -
  ```

  Alternatively, `CMake` can uncompress and unpack the distribution:

  ```
  shell> cmake -E tar zxvf mysql-VERSION.tar.gz
  ```

- To unpack a Zip archive, use `WinZip` or another tool that can read `.zip` files.

Unpacking the distribution file creates a directory named `mysql-VERSION`.

Configure the Distribution

Change location into the top-level directory of the unpacked distribution:

```
shell> cd mysql-VERSION
```

Build outside of the source tree to keep the tree clean. If the top-level source directory is named `mysql-src` under your current working directory, you can build in a directory named `bld` at the same level. Create the directory and go there:

```
shell> mkdir bld
shell> cd bld
```

Configure the build directory. The minimum configuration command includes no options to override configuration defaults:

```
shell> cmake ../mysql-src
```

The build directory needs not be outside the source tree. For example, you can build in a directory named `bld` under the top-level source tree. To do this, starting with `mysql-src` as your current working directory, create the directory `bld` and then go there:

```
shell> mkdir bld
shell> cd bld
```

Configure the build directory. The minimum configuration command includes no options to override configuration defaults:

```
shell> cmake ..
```

If you have multiple source trees at the same level (for example, to build multiple versions of MySQL), the second strategy can be advantageous. The first strategy places all build directories at the same level, which requires that you choose a unique name for each. With the second strategy, you can use the same name for the build directory within each source tree. The following instructions assume this second strategy.

On Windows, specify the development environment. For example, the following commands configure MySQL for 32-bit or 64-bit builds, respectively:

```
shell> cmake .. -G "Visual Studio 12 2013"
shell> cmake .. -G "Visual Studio 12 2013 Win64"
```

On macOS, to use the Xcode IDE:

```
shell> cmake .. -G Xcode
```

When you run `cmake`, you might want to add options to the command line. Here are some examples:

- `-DBUILD_CONFIG=mysql_release`: Configure the source with the same build options used by Oracle to produce binary distributions for official MySQL releases.
- `-DCMAKE_INSTALL_PREFIX=`*`dir_name`*: Configure the distribution for installation under a particular location.
- `-DCPACK_MONOLITHIC_INSTALL=1`: Cause `make package` to generate a single installation file rather than multiple files.
- `-DWITH_DEBUG=1`: Build the distribution with debugging support.

For a more extensive list of options, see Section 2.9.7, “MySQL Source-Configuration Options”.

To list the configuration options, use one of the following commands:

```
shell> cmake .. -L   # overview
shell> cmake .. -LH  # overview with help text
shell> cmake .. -LAH # all params with help text
```

```
shell> ccmake ..     # interactive display
```

If CMake fails, you might need to reconfigure by running it again with different options. If you do reconfigure, take note of the following:

- If CMake is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in CMakeCache.txt. When CMake starts, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run CMake, you must run make again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old object files or configuration information from being used, run these commands in the build direcotry on Unix before re-running CMake:

```
shell> make clean
shell> rm CMakeCache.txt
```

Or, on Windows:

```
shell> devenv MySQL.sln /clean
shell> del CMakeCache.txt
```

Before asking on the MySQL Community Slack, check the files in the CMakeFiles directory for useful information about the failure. To file a bug report, please use the instructions in Section 1.7, “How to Report Bugs or Problems”.

Build the Distribution

On Unix:

```
shell> make
shell> make VERBOSE=1
```

The second command sets VERBOSE to show the commands for each compiled source.

Use gmake instead on systems where you are using GNU make and it has been installed as gmake.

On Windows:

```
shell> devenv MySQL.sln /build RelWithDebInfo
```

If you have gotten to the compilation stage, but the distribution does not build, see Section 2.9.8, “Dealing with Problems Compiling MySQL”, for help. If that does not solve the problem, please enter it into our bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”. If you have installed the latest versions of the required tools, and they crash trying to process our configuration files, please report that also. However, if you get a command not found error or a similar problem for required tools, do not report it. Instead, make sure that all the required tools are installed and that your PATH variable is set correctly so that your shell can find them.

Install the Distribution

On Unix:

```
shell> make install
```

This installs the files under the configured installation directory (by default, `/usr/local/mysql`). You might need to run the command as `root`.

To install in a specific directory, add a `DESTDIR` parameter to the command line:

```
shell> make install DESTDIR="/opt/mysql"
```

Alternatively, generate installation package files that you can install where you like:

```
shell> make package
```

This operation produces one or more `.tar.gz` files that can be installed like generic binary distribution packages. See Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”. If you run `CMake` with `-DCPACK_MONOLITHIC_INSTALL=1`, the operation produces a single file. Otherwise, it produces multiple files.

On Windows, generate the data directory, then create a `.zip` archive installation package:

```
shell> devenv MySQL.sln /build RelWithDebInfo /project initial_database
shell> devenv MySQL.sln /build RelWithDebInfo /project package
```

You can install the resulting `.zip` archive where you like. See Section 2.3.4, “Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive”.

Perform Postinstallation Setup

The remainder of the installation process involves setting up the configuration file, creating the core databases, and starting the MySQL server. For instructions, see Section 2.10, “Postinstallation Setup and Testing”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.10, “Postinstallation Setup and Testing”.

2.9.5 Installing MySQL Using a Development Source Tree

This section describes how to install MySQL from the latest development source code, which is hosted on GitHub. To obtain the MySQL Server source code from this repository hosting service, you can set up a local MySQL Git repository.

On GitHub, MySQL Server and other MySQL projects are found on the MySQL page. The MySQL Server project is a single repository that contains branches for several MySQL series.

MySQL officially joined GitHub in September, 2014. For more information about MySQL's move to GitHub, refer to the announcement on the MySQL Release Engineering blog: MySQL on GitHub

- Prerequisites for Installing from Development Source
- Setting Up a MySQL Git Repository

Prerequisites for Installing from Development Source

To install MySQL from a development source tree, your system must satisfy the tool requirements listed at Section 2.9.2, “Source Installation Prerequisites”.

Setting Up a MySQL Git Repository

To set up a MySQL Git repository on your machine, use this procedure:

1. Clone the MySQL Git repository to your machine. The following command clones the MySQL Git repository to a directory named `mysql-server`. The initial download will take some time to complete, depending on the speed of your connection.

```
~$ git clone https://github.com/mysql/mysql-server.git
Cloning into 'mysql-server'...
remote: Counting objects: 1198513, done.
remote: Total 1198513 (delta 0), reused 0 (delta 0), pack-reused 1198513
Receiving objects: 100% (1198513/1198513), 1.01 GiB | 7.44 MiB/s, done.
Resolving deltas: 100% (993200/993200), done.
Checking connectivity... done.
Checking out files: 100% (25510/25510), done.
```

2. When the clone operation completes, the contents of your local MySQL Git repository appear similar to the following:

```
~$ cd mysql-server
~/mysql-server$ ls
BUILD                cmd-line-utils       libservices          sql
CMakeLists.txt       config.h.cmake       man                  sql-common
COPYING              configure.cmake      mysql-test           storage
Docs                 dbug                 mysys                strings
Doxyfile-perfschema  extra                mysys_ssl            support-files
INSTALL              include              packaging            testclients
README               libbinlogevents      plugin               unittest
VERSION              libbinlogstandalone  rapid                vio
client               libevent             regex                win
cmake                libmysql             scripts              zlib
```

3. Use the `git branch -r` command to view the remote tracking branches for the MySQL repository.

```
~/mysql-server$ git branch -r
  origin/5.5
  origin/5.6
  origin/5.7
  origin/8.0
  origin/HEAD -> origin/5.7
  origin/cluster-7.2
  origin/cluster-7.3
  origin/cluster-7.4
  origin/cluster-7.5
```

4. To view the branches that are checked out in your local repository, issue the `git branch` command. When you cloned the MySQL Git repository, the MySQL 5.7 branch was checked out automatically. The asterisk identifies the 5.7 branch as the active branch.

```
~/mysql-server$ git branch
* 5.7
```

5. To check out a different MySQL branch, run the `git checkout` command, specifying the branch name. For example, to check out the MySQL 8.0 branch:

```
~/mysql-server$ git checkout 8.0
Checking out files: 100% (9600/9600), done.
Branch 8.0 set up to track remote branch 8.0 from origin.
Switched to a new branch '8.0'
```

6. Run `git branch` to verify that the MySQL 8.0 branch is present. MySQL 8.0, which is the last branch you checked out, is marked by an asterisk indicating that it is the active branch.

```
~/mysql-server$ git branch
  5.7
* 8.0
```

7. Use the `git checkout` command to switch between branches. For example:

```
~/mysql-server$ git checkout 5.7
```

8. To obtain changes made after your initial setup of the MySQL Git repository, switch to the branch you want to update and issue the `git pull` command:

```
~/mysql-server$ git checkout 8.0
~/mysql-server$ git pull
```

   To examine the commit history, use the `git log` option:

```
~/mysql-server$ git log
```

   You can also browse commit history and source code on the GitHub MySQL site.

   If you see changes or code that you have a question about, ask on the MySQL Community Slack. For information about contributing a patch, see Contributing to MySQL Server.

9. After you have cloned the MySQL Git repository and have checked out the branch you want to build, you can build MySQL Server from the source code. Instructions are provided in Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”, except that you skip the part about obtaining and unpacking the distribution.

   Be careful about installing a build from a distribution source tree on a production machine. The installation command may overwrite your live release installation. If you already have MySQL installed and do not want to overwrite it, run `CMake` with values for the `CMAKE_INSTALL_PREFIX`, `MYSQL_TCP_PORT`, and `MYSQL_UNIX_ADDR` options different from those used by your production server. For additional information about preventing multiple servers from interfering with each other, see Section 5.8, “Running Multiple MySQL Instances on One Machine”.

   Play hard with your new installation. For example, try to make new features crash. Start by running `make test`. See Section 29.1.2, “The MySQL Test Suite”.

2.9.6 Configuring SSL Library Support

An SSL library is required for support of encrypted connections, entropy for random number generation, and other encryption-related operations.

If you compile MySQL from a source distribution, `CMake` configures the distribution to use the installed OpenSSL library by default.

To compile using OpenSSL, use this procedure:

1. Ensure that OpenSSL 1.0.1 or higher is installed on your system. If the installed OpenSSL version is lower than 1.0.1, `CMake` produces an error at MySQL configuration time. If it is necessary to obtain OpenSSL, visit http://www.openssl.org.

2. The `WITH_SSL` `CMake` option determines which SSL library to use for compiling MySQL (see Section 2.9.7, “MySQL Source-Configuration Options”). The default is `-DWITH_SSL=system`, which uses OpenSSL. To make this explicit, specify that option on the `CMake` command line. For example:

```
cmake . -DWITH_SSL=system
```

That command configures the distribution to use the installed OpenSSL library. Alternatively, to explicitly specify the path name to the OpenSSL installation, use the following syntax. This can be useful if you have multiple versions of OpenSSL installed, to prevent `CMake` from choosing the wrong one:

```
cmake . -DWITH_SSL=path_name
```

3. Compile and install the distribution.

To check whether a `mysqld` server supports encrypted connections, examine the value of the `have_ssl` system variable:

```
mysql> SHOW VARIABLES LIKE 'have_ssl';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_ssl      | YES   |
+---------------+-------+
```

If the value is `YES`, the server supports encrypted connections. If the value is `DISABLED`, the server is capable of supporting encrypted connections but was not started with the appropriate `--ssl-xxx` options to enable encrypted connections to be used; see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

2.9.7 MySQL Source-Configuration Options

The `CMake` program provides a great deal of control over how you configure a MySQL source distribution. Typically, you do this using options on the `CMake` command line. For information about options supported by `CMake`, run either of these commands in the top-level source directory:

```
cmake . -LH
ccmake .
```

You can also affect `CMake` using certain environment variables. See Section 4.9, “MySQL Environment Variables”.

For boolean options, the value may be specified as 1 or `ON` to enable the option, or as 0 or `OFF` to disable the option.

Many options configure compile-time defaults that can be overridden at server startup. For example, the `CMAKE_INSTALL_PREFIX`, `MYSQL_TCP_PORT`, and `MYSQL_UNIX_ADDR` options that configure the default installation base directory location, TCP/IP port number, and Unix socket file can be changed at server startup with the `--basedir`, `--port`, and `--socket` options for `mysqld`. Where applicable, configuration option descriptions indicate the corresponding `mysqld` startup option.

The following sections provide more information about `CMake` options.

- CMake Option Reference
- General Options
- Installation Layout Options
- Storage Engine Options

- Feature Options
- Compiler Flags
- CMake Options for Compiling NDB Cluster

CMake Option Reference

The following table shows the available `CMake` options. In the `Default` column, `PREFIX` stands for the value of the `CMAKE_INSTALL_PREFIX` option, which specifies the installation base directory. This value is used as the parent location for several of the installation subdirectories.

Table 2.13 MySQL Source-Configuration Option Reference (CMake)

Formats	Description	Default	Introduced	Removed
`ADD_GDB_INDEX`	Whether to enable generation of .gdb_index section in binaries		8.0.18	
`BUILD_CONFIG`	Use same build options as official releases			
`BUNDLE_RUNTIME_LIBRARIES`	Bundle runtime libraries with server MSI and Zip packages for Windows	`OFF`	8.0.11	
`CMAKE_BUILD_TYPE`	Type of build to produce	`RelWithDebInfo`		
`CMAKE_CXX_FLAGS`	Flags for C++ Compiler			
`CMAKE_C_FLAGS`	Flags for C Compiler			
`CMAKE_INSTALL_PREFIX`	Installation base directory	`/usr/local/mysql`		
`COMPILATION_COMMENT`	Comment about compilation environment			
`COMPILATION_COMMENT_SERVER`	Comment about compilation environment for use by mysqld		8.0.14	
`CPACK_MONOLITHIC_INSTALL`	Whether package build produces single file	`OFF`		
`DEFAULT_CHARSET`	The default server character set	`utf8mb4`		
`DEFAULT_COLLATION`	The default server collation	`utf8mb4_0900_ai_ci`		
`DISABLE_DATA_LOCK`	Exclude the performance schema data lock instrumentation	`OFF`		
`DISABLE_PSI_COND`	Exclude Performance Schema condition instrumentation	`OFF`		
`DISABLE_PSI_ERROR`	Exclude the performance schema server error instrumentation	`OFF`		
`DISABLE_PSI_FILE`	Exclude Performance Schema file instrumentation	`OFF`		
`DISABLE_PSI_IDLE`	Exclude Performance Schema idle instrumentation	`OFF`		

Formats	Description	Default	Introduced	Removed
DISABLE_PSI_MEMORY	Exclude Performance Schema memory instrumentation	OFF		
DISABLE_PSI_METADATA	Exclude Performance Schema metadata instrumentation	OFF		
DISABLE_PSI_MUTEX	Exclude Performance Schema mutex instrumentation	OFF		
DISABLE_PSI_PS	Exclude the performance schema prepared statements	OFF		
DISABLE_PSI_RWLOCK	Exclude Performance Schema rwlock instrumentation	OFF		
DISABLE_PSI_SOCKET	Exclude Performance Schema socket instrumentation	OFF		
DISABLE_PSI_SP	Exclude Performance Schema stored program instrumentation	OFF		
DISABLE_PSI_STAGE	Exclude Performance Schema stage instrumentation	OFF		
DISABLE_PSI_STATEMENT	Exclude Performance Schema statement instrumentation	OFF		
DISABLE_PSI_STATEMENT_DIGEST	Exclude Performance Schema statements_digest instrumentation	OFF		
DISABLE_PSI_TABLE	Exclude Performance Schema table instrumentation	OFF		
DISABLE_PSI_THREAD	Exclude the performance schema thread instrumentation	OFF		
DISABLE_PSI_TRANSACTION	Exclude the performance schema transaction instrumentation	OFF		
DISABLE_SHARED	Do not build shared libraries, compile position-dependent code	OFF		8.0.18
DOWNLOAD_BOOST	Whether to download the Boost library	OFF		
DOWNLOAD_BOOST_TIMEOUT	Timeout in seconds for downloading the Boost library	600		
ENABLED_LOCAL_INFILE	Whether to enable LOCAL for LOAD DATA	OFF		
ENABLED_PROFILING	Whether to enable query profiling code	ON		
ENABLE_DEBUG_SYNC	Whether to enable Debug Sync support	ON		8.0.1
ENABLE_DOWNLOADS	Whether to download optional files	OFF		

Formats	Description	Default	Introduced	Removed
ENABLE_DTRACE	Whether to include DTrace support			8.0.1
ENABLE_EXPERIMENTAL_SYSVARS	Whether to enabled experimental InnoDB system variables	OFF	8.0.11	
ENABLE_GCOV	Whether to include gcov support			
ENABLE_GPROF	Enable gprof (optimized Linux builds only)	OFF		
FORCE_INSOURCE_BUILD	Whether to force an in-source build	OFF	8.0.14	
FORCE_UNSUPPORTED_COMPILER	Whether to permit unsupported compiler	OFF		
FPROFILE_GENERATE	Whether to generate profile guided optimization data	OFF	8.0.19	
FPROFILE_USE	Whether to use profile guided optimization data	OFF	8.0.19	
IGNORE_AIO_CHECK	With -DBUILD_CONFIG=mysql_release, ignore libaio check	OFF		
INSTALL_BINDIR	User executables directory	PREFIX/bin		
INSTALL_DOCDIR	Documentation directory	PREFIX/docs		
INSTALL_DOCREADMEDIR	README file directory	PREFIX		
INSTALL_INCLUDEDIR	Header file directory	PREFIX/include		
INSTALL_INFODIR	Info file directory	PREFIX/docs		
INSTALL_LAYOUT	Select predefined installation layout	STANDALONE		
INSTALL_LIBDIR	Library file directory	PREFIX/lib		
INSTALL_MANDIR	Manual page directory	PREFIX/man		
INSTALL_MYSQLKEYRINGDIR	Directory for keyring_file plugin data file	platform specific		
INSTALL_MYSQLSHAREDIR	Shared data directory	PREFIX/share		
INSTALL_MYSQLTESTDIR	mysql-test directory	PREFIX/mysql-test		
INSTALL_PKGCONFIGDIR	Directory for mysqlclient.pc pkg-config file	INSTALL_LIBDIR/pkgconfig		
INSTALL_PLUGINDIR	Plugin directory	PREFIX/lib/plugin		
INSTALL_SBINDIR	Server executable directory	PREFIX/bin		
INSTALL_SECURE_FILE_PRIVDIR	secure_file_priv default value	platform specific		
INSTALL_SHAREDIR	aclocal/mysql.m4 installation directory	PREFIX/share		

Formats	Description	Default	Introduced	Removed
INSTALL_STATIC_LIBRARIES	Whether to install static libraries	ON		
INSTALL_SUPPORTFILESDIR	Extra support files directory	PREFIX/support-files		
LINK_RANDOMIZE	Whether to randomize order of symbols in mysqld binary	OFF	8.0.1	
LINK_RANDOMIZE_SEED	Seed value for LINK_RANDOMIZE option	mysql	8.0.1	
MAX_INDEXES	Maximum indexes per table	64		
MUTEX_TYPE	InnoDB mutex type	event		
MYSQLX_TCP_PORT	TCP/IP port number used by X Plugin	33060		
MYSQLX_UNIX_ADDR	Unix socket file used by X Plugin	/tmp/mysqlx.sock		
MYSQL_DATADIR	Data directory			
MYSQL_MAINTAINER_MODE	Whether to enable MySQL maintainer-specific development environment	OFF		
MYSQL_PROJECT_NAME	Windows/OS X project name	MySQL		
MYSQL_TCP_PORT	TCP/IP port number	3306		
MYSQL_UNIX_ADDR	Unix socket file	/tmp/mysql.sock		
ODBC_INCLUDES	ODBC includes directory			
ODBC_LIB_DIR	ODBC library directory			
OPTIMIZER_TRACE	Whether to support optimizer tracing			
REPRODUCIBLE_BUILD	Take extra care to create a build result independent of build location and time		8.0.11	
SYSCONFDIR	Option file directory			
SYSTEMD_PID_DIR	Directory for PID file under systemd	/var/run/mysqld		
SYSTEMD_SERVICE_NAME	Name of MySQL service under systemd	mysqld		
TMPDIR	tmpdir default value			
USE_LD_GOLD	Whether to use GNU gold linker	ON	8.0.0	
USE_LD_LLD	Whether to use llvm lld linker	ON	8.0.16	
WIN_DEBUG_NO_INLINE	Whether to disable function inlining	OFF		
WITHOUT_xxx_STORAGE_ENGINE	Exclude storage engine xxx from build			

Formats	Description	Default	Introduced	Removed
WITH_ANT	Path to Ant for building GCS Java wrapper		8.0.11	
WITH_ASAN	Enable AddressSanitizer	OFF		
WITH_ASAN_SCOPE	Enable AddressSanitizer -fsanitize-address-use-after-scope Clang flag	OFF	8.0.4	
WITH_AUTHENTICATION_LDAP	Whether to report error if LDAP authentication plugins cannot be built	OFF	8.0.2	
WITH_AUTHENTICATION_PAM	Build PAM authentication plugin	OFF		
WITH_AWS_SDK	Location of Amazon Web Services software development kit		8.0.2	
WITH_BOOST	The location of the Boost library sources			
WITH_CLIENT_PROTOCOL_TRACING	Build client-side protocol tracing framework	ON		
WITH_CURL	Location of curl library		8.0.2	
WITH_DEBUG	Whether to include debugging support	OFF		
WITH_DEFAULT_COMPILER_OPTIONS	Whether to use default compiler options	ON		
WITH_DEFAULT_FEATURE_SET	Whether to use default feature set	ON		
WITH_EDITLINE	Which libedit/editline library to use	bundled		
WITH_GMOCK	Path to googlemock distribution			
WITH_ICU	Type of ICU support	bundled	8.0.4	
WITH_INNODB_EXTRA_DEBUG	Whether to include extra debugging support for InnoDB.	OFF		
WITH_INNODB_MEMCACHED	Whether to generate memcached shared libraries.	OFF		
WITH_JEMALLOC	Whether to link with -ljemalloc	OFF	8.0.16	
WITH_KEYRING_TEST	Build the keyring test program	OFF		
WITH_LIBEVENT	Which libevent library to use	bundled		
WITH_LIBWRAP	Whether to include libwrap (TCP wrappers) support	OFF		
WITH_LOCK_ORDER	Whether to enable LOCK_ORDER tooling	OFF	8.0.17	
WITH_LSAN	Whether to run LeakSanitizer, without AddressSanitizer	OFF	8.0.16	

Formats	Description	Default	Introduced	Removed
WITH_LTO	Enable link-time optimizer	OFF	8.0.13	
WITH_LZ4	Type of LZ4 library support	bundled		
WITH_LZMA	Type of LZMA library support	bundled	8.0.4	8.0.16
WITH_MECAB	Compiles MeCab			
WITH_MSAN	Enable MemorySanitizer	OFF		
WITH_MSCRT_DEBUG	Enable Visual Studio CRT memory leak tracing	OFF		
WITH_MYSQLX	Whether to disable X Protocol	ON	8.0.11	
WITH_NUMA	Set NUMA memory allocation policy			
WITH_PROTOBUF	Which Protocol Buffers package to use	bundled		
WITH_RAPID	Whether to build rapid development cycle plugins	ON		
WITH_RAPIDJSON	Type of RapidJSON support	bundled	8.0.13	
WITH_RE2	Type of RE2 library support	bundled	8.0.4	8.0.18
WITH_ROUTER	Whether to build MySQL Router	ON	8.0.16	
WITH_SSL	Type of SSL support	system		
WITH_SYSTEMD	Enable installation of systemd support files	OFF		
WITH_SYSTEM_LIBS	Set system value of library options not set explicitly	OFF	8.0.11	
WITH_TEST_TRACE_PLUGIN	Build test protocol trace plugin	OFF		
WITH_TSAN	Enable ThreadSanitizer	OFF		
WITH_UBSAN	Enable Undefined Behavior Sanitizer	OFF		
WITH_UNIT_TESTS	Compile MySQL with unit tests	ON		
WITH_UNIXODBC	Enable unixODBC support	OFF		
WITH_VALGRIND	Whether to compile in Valgrind header files	OFF		
WITH_ZLIB	Type of zlib support	bundled		
WITH_ZSTD	Type of zstd support	bundled	8.0.18	
WITH_xxx_STORAGE_ENGINE	Compile storage engine xxx statically into server			

General Options

- -DBUILD_CONFIG=mysql_release

  This option configures a source distribution with the same build options used by Oracle to produce binary distributions for official MySQL releases.

- -DBUNDLE_RUNTIME_LIBRARIES=*bool*

Whether to bundle runtime libraries with server MSI and Zip packages for Windows.

- `-DCMAKE_BUILD_TYPE=type`

  The type of build to produce:

  - `RelWithDebInfo`: Enable optimizations and generate debugging information. This is the default MySQL build type.
  - `Release`: Enable optimizations but omit debugging information to reduce the build size. This build type was added in MySQL 8.0.13.
  - `Debug`: Disable optimizations and generate debugging information. This build type is also used if the `WITH_DEBUG` option is enabled. That is, `-DWITH_DEBUG=1` has the same effect as `-DCMAKE_BUILD_TYPE=Debug`.

- `-DCPACK_MONOLITHIC_INSTALL=bool`

  This option affects whether the `make package` operation produces multiple installation package files or a single file. If disabled, the operation produces multiple installation package files, which may be useful if you want to install only a subset of a full MySQL installation. If enabled, it produces a single file for installing everything.

- `-DFORCE_INSOURCE_BUILD=bool`

  Defines whether to force an in-source build. Out-of-source builds are recommended, as they permit multiple builds from the same source, and cleanup can be performed quickly by removing the build directory. To force an in-source build, invoke `CMake` with `-DFORCE_INSOURCE_BUILD=ON`.

Installation Layout Options

The `CMAKE_INSTALL_PREFIX` option indicates the base installation directory. Other options with names of the form `INSTALL_xxx` that indicate component locations are interpreted relative to the prefix and their values are relative pathnames. Their values should not include the prefix.

- `-DCMAKE_INSTALL_PREFIX=dir_name`

  The installation base directory.

  This value can be set at server startup with the `--basedir` option.

- `-DINSTALL_BINDIR=dir_name`

  Where to install user programs.

- `-DINSTALL_DOCDIR=dir_name`

  Where to install documentation.

- `-DINSTALL_DOCREADMEDIR=dir_name`

  Where to install `README` files.

- `-DINSTALL_INCLUDEDIR=dir_name`

  Where to install header files.

- `-DINSTALL_INFODIR=dir_name`

Where to install Info files.

- `-DINSTALL_LAYOUT=name`

  Select a predefined installation layout:

  - `STANDALONE`: Same layout as used for `.tar.gz` and `.zip` packages. This is the default.
  - `RPM`: Layout similar to RPM packages.
  - `SVR4`: Solaris package layout.
  - `DEB`: DEB package layout (experimental).

  You can select a predefined layout but modify individual component installation locations by specifying other options. For example:

  ```
  cmake . -DINSTALL_LAYOUT=SVR4 -DMYSQL_DATADIR=/var/mysql/data
  ```

  The `INSTALL_LAYOUT` value determines the default value of the `secure_file_priv`, `keyring_encrypted_file_data`, and `keyring_file_data` system variables. See the descriptions of those variables in Section 5.1.8, “Server System Variables”, and Section 6.4.4.11, “Keyring System Variables”.

- `-DINSTALL_LIBDIR=dir_name`

  Where to install library files.

- `-DINSTALL_MANDIR=dir_name`

  Where to install manual pages.

- `-DINSTALL_MYSQLKEYRINGDIR=dir_path`

  The default directory to use as the location of the `keyring_file` plugin data file. The default value is platform specific and depends on the value of the `INSTALL_LAYOUT` `CMake` option; see the description of the `keyring_file_data` system variable in Section 5.1.8, “Server System Variables”.

- `-DINSTALL_MYSQLSHAREDIR=dir_name`

  Where to install shared data files.

- `-DINSTALL_MYSQLTESTDIR=dir_name`

  Where to install the `mysql-test` directory. To suppress installation of this directory, explicitly set the option to the empty value (`-DINSTALL_MYSQLTESTDIR=`).

- `-DINSTALL_PKGCONFIGDIR=dir_name`

  The directory in which to install the `mysqlclient.pc` file for use by `pkg-config`. The default value is `INSTALL_LIBDIR/pkgconfig`, unless `INSTALL_LIBDIR` ends with `/mysql`, in which case that is removed first.

- `-DINSTALL_PLUGINDIR=dir_name`

  The location of the plugin directory.

This value can be set at server startup with the `--plugin_dir` option.

- `-DINSTALL_SBINDIR=dir_name`

  Where to install the `mysqld` server.

- `-DINSTALL_SECURE_FILE_PRIVDIR=dir_name`

  The default value for the `secure_file_priv` system variable. The default value is platform specific and depends on the value of the `INSTALL_LAYOUT` `CMake` option; see the description of the `secure_file_priv` system variable in Section 5.1.8, “Server System Variables”.

- `-DINSTALL_SHAREDIR=dir_name`

  Where to install `aclocal/mysql.m4`.

- `-DINSTALL_STATIC_LIBRARIES=bool`

  Whether to install static libraries. The default is `ON`. If set to `OFF`, these libraries are not installed: `libmysqlclient.a`, `libmysqlservices.a`.

- `-DINSTALL_SUPPORTFILESDIR=dir_name`

  Where to install extra support files.

- `-DLINK_RANDOMIZE=bool`

  Whether to randomize the order of symbols in the `mysqld` binary. The default is `OFF`. This option should be enabled only for debugging purposes.

- `-DLINK_RANDOMIZE_SEED=val`

  Seed value for the `LINK_RANDOMIZE` option. The value is a string. The default is `mysql`, an arbitrary choice.

- `-DMYSQL_DATADIR=dir_name`

  The location of the MySQL data directory.

  This value can be set at server startup with the `--datadir` option.

- `-DODBC_INCLUDES=dir_name`

  The location of the ODBC includes directory, and may be used while configuring Connector/ODBC.

- `-DODBC_LIB_DIR=dir_name`

  The location of the ODBC library directory, and may be used while configuring Connector/ODBC.

- `-DSYSCONFDIR=dir_name`

  The default `my.cnf` option file directory.

  This location cannot be set at server startup, but you can start the server with a given option file using the `--defaults-file=file_name` option, where *`file_name`* is the full path name to the file.

- `-DSYSTEMD_PID_DIR=dir_name`

The name of the directory in which to create the PID file when MySQL is managed by systemd. The default is `/var/run/mysqld`; this might be changed implicitly according to the `INSTALL_LAYOUT` value.

This option is ignored unless `WITH_SYSTEMD` is enabled.

- `-DSYSTEMD_SERVICE_NAME=name`

  The name of the MySQL service to use when MySQL is managed by systemd. The default is `mysqld`; this might be changed implicitly according to the `INSTALL_LAYOUT` value.

  This option is ignored unless `WITH_SYSTEMD` is enabled.

- `-DTMPDIR=dir_name`

  The default location to use for the `tmpdir` system variable. If unspecified, the value defaults to `P_tmpdir` in `<stdio.h>`.

Storage Engine Options

Storage engines are built as plugins. You can build a plugin as a static module (compiled into the server) or a dynamic module (built as a dynamic library that must be installed into the server using the `INSTALL PLUGIN` statement or the `--plugin-load` option before it can be used). Some plugins might not support static or dynamic building.

The `InnoDB`, `MyISAM`, `MERGE`, `MEMORY`, and `CSV` engines are mandatory (always compiled into the server) and need not be installed explicitly.

To compile a storage engine statically into the server, use `-DWITH_engine_STORAGE_ENGINE=1`. Some permissible *`engine`* values are `ARCHIVE`, `BLACKHOLE`, `EXAMPLE`, `FEDERATED`, and `NDB` or `NDBCLUSTER` (`NDB` support). Examples:

```
-DWITH_ARCHIVE_STORAGE_ENGINE=1
-DWITH_BLACKHOLE_STORAGE_ENGINE=1
```

Note

It is not possible to compile without Performance Schema support. If it is desired to compile without particular types of instrumentation, that can be done with the following `CMake` options:

```
DISABLE_PSI_COND
DISABLE_PSI_DATA_LOCK
DISABLE_PSI_ERROR
DISABLE_PSI_FILE
DISABLE_PSI_IDLE
DISABLE_PSI_MEMORY
DISABLE_PSI_METADATA
DISABLE_PSI_MUTEX
DISABLE_PSI_PS
DISABLE_PSI_RWLOCK
DISABLE_PSI_SOCKET
DISABLE_PSI_SP
DISABLE_PSI_STAGE
DISABLE_PSI_STATEMENT
DISABLE_PSI_STATEMENT_DIGEST
DISABLE_PSI_TABLE
```

```
DISABLE_PSI_THREAD
DISABLE_PSI_TRANSACTION
```

For example, to compile without mutex instrumentation, configure MySQL using the `-DDISABLE_PSI_MUTEX=1` option.

To exclude a storage engine from the build, use `-DWITH_engine_STORAGE_ENGINE=0`. Examples:

```
-DWITH_ARCHIVE_STORAGE_ENGINE=0
-DWITH_EXAMPLE_STORAGE_ENGINE=0
-DWITH_FEDERATED_STORAGE_ENGINE=0
```

It is also possible to exclude a storage engine from the build using `-DWITHOUT_engine_STORAGE_ENGINE=1` (but `-DWITH_engine_STORAGE_ENGINE=0` is preferred). Examples:

```
-DWITHOUT_ARCHIVE_STORAGE_ENGINE=1
-DWITHOUT_EXAMPLE_STORAGE_ENGINE=1
-DWITHOUT_FEDERATED_STORAGE_ENGINE=1
```

If neither `-DWITH_engine_STORAGE_ENGINE` nor `-DWITHOUT_engine_STORAGE_ENGINE` are specified for a given storage engine, the engine is built as a shared module, or excluded if it cannot be built as a shared module.

Feature Options

- `-DADD_GDB_INDEX=bool`

  This option determines whether to enable generation of a `.gdb_index` section in binaries, which makes loading them in a debugger faster. The option is disabled by default. `lld` linker is used, and is disabled by It has no effect if a linker other than `lld` or GNU `gold` is used.

  This option was added in MySQL 8.0.18.

- `-DCOMPILATION_COMMENT=string`

  A descriptive comment about the compilation environment. As of MySQL 8.0.14, `mysqld` uses `COMPILATION_COMMENT_SERVER`. Other programs continue to use `COMPILATION_COMMENT`.

- `-DCOMPILATION_COMMENT_SERVER=string`

  A descriptive comment about the compilation environment for use by `mysqld` (for example, to set the `version_comment` system variable). This option was added in MySQL 8.0.14. Prior to 8.0.14, the server uses `COMPILATION_COMMENT`.

- `-DDEFAULT_CHARSET=charset_name`

  The server character set. By default, MySQL uses the `utf8mb4` character set.

  `charset_name` may be one of `binary`, `armscii8`, `ascii`, `big5`, `cp1250`, `cp1251`, `cp1256`, `cp1257`, `cp850`, `cp852`, `cp866`, `cp932`, `dec8`, `eucjpms`, `euckr`, `gb2312`, `gbk`, `geostd8`, `greek`, `hebrew`, `hp8`, `keybcs2`, `koi8r`, `koi8u`, `latin1`, `latin2`, `latin5`, `latin7`, `macce`, `macroman`, `sjis`, `swe7`, `tis620`, `ucs2`, `ujis`, `utf8`, `utf8mb4`, `utf16`, `utf16le`, `utf32`. The permissible character sets are listed in the `cmake/character_sets.cmake` file as the value of `CHARSETS_AVAILABLE`.

  This value can be set at server startup with the `--character_set_server` option.

- `-DDEFAULT_COLLATION=collation_name`

  The server collation. By default, MySQL uses `utf8mb4_0900_ai_ci`. Use the `SHOW COLLATION` statement to determine which collations are available for each character set.

  This value can be set at server startup with the `--collation_server` option.

- `-DDISABLE_PSI_COND=bool`

  Whether to exclude the Performance Schema condition instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_FILE=bool`

  Whether to exclude the Performance Schema file instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_IDLE=bool`

  Whether to exclude the Performance Schema idle instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_MEMORY=bool`

  Whether to exclude the Performance Schema memory instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_METADATA=bool`

  Whether to exclude the Performance Schema metadata instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_MUTEX=bool`

  Whether to exclude the Performance Schema mutex instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_RWLOCK=bool`

  Whether to exclude the Performance Schema rwlock instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_SOCKET=bool`

  Whether to exclude the Performance Schema socket instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_SP=bool`

  Whether to exclude the Performance Schema stored program instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_STAGE=bool`

  Whether to exclude the Performance Schema stage instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_STATEMENT=bool`

  Whether to exclude the Performance Schema statement instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_STATEMENT_DIGEST=bool`

  Whether to exclude the Performance Schema statement_digest instrumentation. The default is `OFF` (include).

- `-DDISABLE_PSI_TABLE=bool`

  Whether to exclude the Performance Schema table instrumentation. The default is `OFF` (include).

- -DDISABLE_SHARED=*bool*

  Whether to disable building build shared libraries and compile position-dependent code. The default is OFF (compile position-independent code).

  This option is unused and was removed in MySQL 8.0.18.

- -DDISABLE_PSI_PS=*bool*

  Exclude the performance schema prepared statements instances instrumentation. The default is OFF (include).

- -DDISABLE_PSI_THREAD=*bool*

  Exclude the performance schema thread instrumentation. The default is OFF (include).

  Only disable threads when building without any instrumentation, because other instrumentations have a dependency on threads.

- -DDISABLE_PSI_TRANSACTION=*bool*

  Exclude the performance schema transaction instrumentation. The default is OFF (include).

- -DDISABLE_PSI_DATA_LOCK=*bool*

  Exclude the performance schema data lock instrumentation. The default is OFF (include).

- -DDISABLE_PSI_ERROR=*bool*

  Exclude the performance schema server error instrumentation. The default is OFF (include).

- -DDOWNLOAD_BOOST=*bool*

  Whether to download the Boost library. The default is OFF.

  See the WITH_BOOST option for additional discussion about using Boost.

- -DDOWNLOAD_BOOST_TIMEOUT=*seconds*

  The timeout in seconds for downloading the Boost library. The default is 600 seconds.

  See the WITH_BOOST option for additional discussion about using Boost.

- -DENABLE_DEBUG_SYNC=*bool*

  As of MySQL 8.0.1, ENABLE_DEBUG_SYNC is removed and enabling WITH_DEBUG enables Debug Sync.

- -DENABLE_DOWNLOADS=*bool*

  Whether to download optional files. For example, with this option enabled, CMake downloads the Google Test distribution that is used by the test suite to run unit tests, or Ant and JUnit required for building GCS Java wrapper.

- -DENABLE_DTRACE=*bool*

  Whether to include support for DTrace probes.

  This option was removed in MySQL 8.0.1.

- -DENABLE_EXPERIMENTAL_SYSVARS=*bool*

Whether to enable experimental InnoDB system variables. Experimental system variables are intended for those engaged in MySQL development, should only be used in a development or test environment, and may be removed without notice in a future MySQL release. For information about experimental system variables, refer to /storage/innobase/handler/ha_innodb.cc in the MySQL source tree. Experimental system variables can be identified by searching for “PLUGIN_VAR_EXPERIMENTAL”.

- -DENABLE_GCOV=*bool*

  Whether to include gcov support (Linux only).

- -DENABLE_GPROF=*bool*

  Whether to enable gprof (optimized Linux builds only).

- -DENABLED_LOCAL_INFILE=*bool*

  This option controls the compiled-in default LOCAL capability for the MySQL client library. Clients that make no explicit arrangements therefore have LOCAL capability disabled or enabled according to the ENABLED_LOCAL_INFILE setting specified at MySQL build time.

  By default, the client library in MySQL binary distributions is compiled with ENABLED_LOCAL_INFILE disabled. If you compile MySQL from source, configure it with ENABLED_LOCAL_INFILE disabled or enabled based on whether clients that make no explicit arrangements should have LOCAL capability disabled or enabled, respectively.

  ENABLED_LOCAL_INFILE controls the default for client-side LOCAL capability. For the server, the local_infile system variable controls server-side LOCAL capability. To explicitly cause the server to refuse or permit LOAD DATA LOCAL statements (regardless of how client programs and libraries are configured at build time or runtime), start mysqld with local_infile disabled or enabled, respectively. local_infile can also be set at runtime. See Section 6.1.6, “Security Issues with LOAD DATA LOCAL”.

- -DENABLED_PROFILING=*bool*

  Whether to enable query profiling code (for the SHOW PROFILE and SHOW PROFILES statements).

- -DFORCE_UNSUPPORTED_COMPILER=*bool*

  By default, CMake checks for minimum versions of supported compilers: Visual Studio 2015 (Windows); GCC 4.8 or Clang 3.4 (Linux); Developer Studio 12.5 (Solaris server); Developer Studio 12.4 or GCC 4.8 (Solaris client library); Clang 3.6 (macOS), Clang 3.4 (FreeBSD). To disable this check, use -DFORCE_UNSUPPORTED_COMPILER=ON.

- -DFPROFILE_GENERATE=*bool*

  Whether to generate profile guided optimization (PGO) data. This option is available for experimenting with PGO with GCC. See the cmake/fprofile.cmake file in a MySQL source distribution for information about using FPROFILE_GENERATE and FPROFILE_USE. These options have been tested with GCC 8 and 9.

  This option was added in MySQL 8.0.19.

- -DFPROFILE_USE=*bool*

  Whether to use profile guided optimization (PGO) data. This option is available for experimenting with PGO with GCC. See the cmake/fprofile.cmake file in a MySQL source distribution for information

about using FPROFILE_GENERATE and FPROFILE_USE. These options have been tested with GCC 8 and 9.

Enabling FPROFILE_USE also enables WITH_LTO.

This option was added in MySQL 8.0.19.

- -DIGNORE_AIO_CHECK=*bool*

  If the -DBUILD_CONFIG=mysql_release option is given on Linux, the libaio library must be linked in by default. If you do not have libaio or do not want to install it, you can suppress the check for it by specifying -DIGNORE_AIO_CHECK=1.

- -DMAX_INDEXES=*num*

  The maximum number of indexes per table. The default is 64. The maximum is 255. Values smaller than 64 are ignored and the default of 64 is used.

- -DMYSQL_MAINTAINER_MODE=*bool*

  Whether to enable a MySQL maintainer-specific development environment. If enabled, this option causes compiler warnings to become errors.

- -DMUTEX_TYPE=*type*

  The mutex type used by InnoDB. Options include:

  - event: Use event mutexes. This is the default value and the original InnoDB mutex implementation.
  - sys: Use POSIX mutexes on UNIX systems. Use CRITICAL_SECTION onjects on Windows, if available.
  - futex: Use Linux futexes instead of condition variables to schedule waiting threads.

- -DMYSQLX_TCP_PORT=*port_num*

  The port number on which X Plugin listens for TCP/IP connections. The default is 33060.

  This value can be set at server startup with the mysqlx_port system variable.

- -DMYSQLX_UNIX_ADDR=*file_name*

  The Unix socket file path on which the server listens for X Plugin socket connections. This must be an absolute path name. The default is /tmp/mysqlx.sock.

  This value can be set at server startup with the mysqlx_port system variable.

- -DMYSQL_PROJECT_NAME=*name*

  For Windows or macOS, the project name to incorporate into the project file name.

- -DMYSQL_TCP_PORT=*port_num*

  The port number on which the server listens for TCP/IP connections. The default is 3306.

  This value can be set at server startup with the --port option.

- -DMYSQL_UNIX_ADDR=*file_name*

The Unix socket file path on which the server listens for socket connections. This must be an absolute path name. The default is `/tmp/mysql.sock`.

This value can be set at server startup with the `--socket` option.

- `-DOPTIMIZER_TRACE=bool`

  Whether to support optimizer tracing. See MySQL Internals: Tracing the Optimizer.

- `-DREPRODUCIBLE_BUILD=bool`

  For builds on Linux systems, this option controls whether to take extra care to create a build result independent of build location and time.

  This option was added in MySQL 8.0.11. As of MySQL 8.0.12, it defaults to `ON` for `RelWithDebInfo` builds.

- `-DUSE_LD_GOLD=bool`

  `CMake` causes the build process to link with the GNU `gold` linker if it is available and not explicitly disabled. To disable use of this linker, specify the `-DUSE_LD_GOLD=OFF` option.

- `-DUSE_LD_LLD=bool`

  `CMake` causes the build process to link with the `llvm lld` linker for Clang if it is available and not explicitly disabled. To disable use of this linker, specify the `-DUSE_LD_LLD=OFF` option.

  This option was added in MySQL 8.0.16.

- `-DWIN_DEBUG_NO_INLINE=bool`

  Whether to disable function inlining on Windows. The default is off (inlining enabled).

- `-DWITH_ANT=path_name`

  Set the path to Ant, required when building GCS Java wrapper. Works in a similar way to the existing `WITH_BOOST` CMake option. Set `WITH_ANT` to the path of a directory where the Ant tarball, or an already unpacked archive, is saved. When `WITH_ANT` is not set, or is set with the special value `system`, the build assumes a binary `ant` exists in `$PATH`.

- `-DWITH_ASAN=bool`

  Whether to enable the AddressSanitizer, for compilers that support it. The default is off.

- `-DWITH_ASAN_SCOPE=bool`

  Whether to enable the AddressSanitizer `-fsanitize-address-use-after-scope` Clang flag for use-after-scope detection. The default is off. To use this option, `-DWITH_ASAN` must also be enabled.

- `-DWITH_AUTHENTICATION_LDAP=bool`

  Whether to report an error if the LDAP authentication plugins cannot be built:

  - If this option is disabled (the default), the LDAP plugins are built if the required header files and libraries are found. If they are not, `CMake` displays a note about it.
  - If this option is enabled, a failure to find the required header file andlibraries causes `CMake` to produce an error, preventing the server from being built.

- `-DWITH_AUTHENTICATION_PAM=bool`

  Whether to build the PAM authentication plugin, for source trees that include this plugin. (See Section 6.4.1.5, “PAM Pluggable Authentication”.) If this option is specified and the plugin cannot be compiled, the build fails.

- `-DWITH_AWS_SDK=path_name`

  The location of the Amazon Web Services software development kit.

- `-DWITH_BOOST=path_name`

  The Boost library is required to build MySQL. These `CMake` options enable control over the library source location, and whether to download it automatically:

  - `-DWITH_BOOST=path_name` specifies the Boost library directory location. It is also possible to specify the Boost location by setting the `BOOST_ROOT` or `WITH_BOOST` environment variable.

    `-DWITH_BOOST=system` is also permitted and indicates that the correct version of Boost is installed on the compilation host in the standard location. In this case, the installed version of Boost is used rather than any version included with a MySQL source distribution.

  - `-DDOWNLOAD_BOOST=bool` specifies whether to download the Boost source if it is not present in the specified location. The default is `OFF`.

  - `-DDOWNLOAD_BOOST_TIMEOUT=seconds` the timeout in seconds for downloading the Boost library. The default is 600 seconds.

  For example, if you normally build MySQL placing the object output in the `bld` subdirectory of your MySQL source tree, you can build with Boost like this:

  ```
  mkdir bld
  cd bld
  cmake .. -DDOWNLOAD_BOOST=ON -DWITH_BOOST=$HOME/my_boost
  ```

  This causes Boost to be downloaded into the `my_boost` directory under your home directory. If the required Boost version is already there, no download is done. If the required Boost version changes, the newer version is downloaded.

  If Boost is already installed locally and your compiler finds the Boost header files on its own, it may not be necessary to specify the preceding `CMake` options. However, if the version of Boost required by MySQL changes and the locally installed version has not been upgraded, you may have build problems. Using the `CMake` options should give you a successful build.

  With the above settings that allow Boost download into a specified location, when the required Boost version changes, you need to remove the `bld` folder, recreate it, and perform the `cmake` step again. Otherwise, the new Boost version might not get downloaded, and compilation might fail.

- `-DWITH_CLIENT_PROTOCOL_TRACING=bool`

  Whether to build the client-side protocol tracing framework into the client library. By default, this option is enabled.

  For information about writing protocol trace client plugins, see Section 29.2.4.11, “Writing Protocol Trace Plugins”.

  See also the `WITH_TEST_TRACE_PLUGIN` option.

- `-DWITH_CURL=curl_type`

  The location of the `curl` library. *`curl_type`* can be `system` (use the system `curl` library) or a path name to the `curl` library.

- `-DWITH_DEBUG=bool`

  Whether to include debugging support.

  Configuring MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

  Sync debug checking for the `InnoDB` storage engine is defined under `UNIV_DEBUG` and is available when debugging support is compiled in using the `WITH_DEBUG` option. When debugging support is compiled in, the `innodb_sync_debug` configuration option can be used to enable or disable `InnoDB` sync debug checking.

  Enabling `WITH_DEBUG` also enables Debug Sync. This facility is used for testing and debugging. When compiled in, Debug Sync is disabled by default at runtime. To enable it, start `mysqld` with the `--debug-sync-timeout=N` option, where *`N`* is a timeout value greater than 0. (The default value is 0, which disables Debug Sync.) *`N`* becomes the default timeout for individual synchronization points.

  Sync debug checking for the `InnoDB` storage engine is available when debugging support is compiled in using the `WITH_DEBUG` option.

  For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `-DWITH_DEFAULT_FEATURE_SET=bool`

  Whether to use the flags from `cmake/build_configurations/feature_set.cmake`.

- `-DWITH_EDITLINE=value`

  Which `libedit`/`editline` library to use. The permitted values are `bundled` (the default) and `system`.

- `-DWITH_ICU={icu_type|path_name}`

  MySQL uses International Components for Unicode (ICU) to support regular expression operations. The `WITH_ICU` option indicates the type of ICU support to include or the path name to the ICU installation to use.

  - *`icu_type`* can be one of the following values:

    - `bundled`: Use the ICU library bundled with the distribution. This is the default, and is the only supported option for Windows.

    - `system`: Use the system ICU library.

  - *`path_name`* is the path name to the ICU installation to use. This can be preferable to using the *`icu_type`* value of `system` because it can prevent CMake from detecting and using an older or incorrect ICU version installed on the system. (Another permitted way to do the same thing is to set `WITH_ICU` to `system` and set the `CMAKE_PREFIX_PATH` option to *`path_name`*.)

- `-DWITH_INNODB_EXTRA_DEBUG=bool`

Whether to include extra InnoDB debugging support.

Enabling `WITH_INNODB_EXTRA_DEBUG` turns on extra InnoDB debug checks. This option can only be enabled when `WITH_DEBUG` is enabled.

- `-DWITH_GMOCK=path_name`

  The path to the googlemock distribution, for use with Google Test-based unit tests. The option value is the path to the distribution Zip file. Alternatively, set the `WITH_GMOCK` environment variable to the path name. It is also possible to use `-DENABLE_DOWNLOADS=1` and `CMake` will download the distribution from GitHub.

  If you build MySQL without the Google Test-based unit tests (by configuring wihout `WITH_GMOCK`), `CMake` displays a message indicating how to download it.

- `-DWITH_INNODB_MEMCACHED=bool`

  Whether to generate memcached shared libraries (`libmemcached.so` and `innodb_engine.so`).

- `-DWITH_JEMALLOC=bool`

  Whether to link with `-ljemalloc`. If enabled, built-in `malloc()`, `calloc()`, `realloc()`, and `free()` routines are disabled. The default is `OFF`.

  This option was added in MySQL 8.0.16.

- `-DWITH_KEYRING_TEST=bool`

  Whether to build the test program that accompanies the `keyring_file` plugin. The default is `OFF`. Test file source code is located in the `plugin/keyring/keyring-test` directory.

- `-DWITH_LIBEVENT=string`

  Which `libevent` library to use. Permitted values are `bundled` (default), `system`, and `yes`. If you specify `system` or `yes`, the system `libevent` library is used if present. If the system library is not found, the bundled `libevent` library is used. The `libevent` library is required by `InnoDB` memcached.

- `-DWITH_LIBWRAP=bool`

  Whether to include `libwrap` (TCP wrappers) support.

- `-DWITH_LOCK_ORDER=bool`

  Whether to enable LOCK_ORDER tooling. By default, this option is disabled and server builds contain no tooling. If tooling is enabled, the LOCK_ORDER tool is available and can be used as described in Section 29.5.3, “The LOCK_ORDER Tool”.

  Note

  With the `WITH_LOCK_ORDER` option enabled, MySQL builds require the `flex` program.

  This option was added in MySQL 8.0.17.

- `-DWITH_LSAN=bool`

  Whether to run LeakSanitizer, without AddressSanitizer. The default is `OFF`.

This option was added in MySQL 8.0.16.

- `-DWITH_LTO=bool`

  Whether to enable the link-time optimizer, if the compiler supports it. The default is `OFF` unless `FPROFILE_USE` is enabled.

  This option was added in MySQL 8.0.13.

- `-DWITH_LZ4=lz4_type`

  The `WITH_LZ4` indicates the source of `zlib` support:

  - `bundled`: Use the `lz4` library bundled with the distribution. This is the default.
  - `system`: Use the system `lz4` library. If `WITH_LZ4` is set to this value, the `lz4_decompress` utility is not built. In this case, the system `lz4` command can be used instead.

- `-DWITH_MSAN=bool`

  Whether to enable MemorySanitizer, for compilers that support it. The default is off.

  For this option to have an effect if enabled, all libraries linked to MySQL must also have been compiled with the option enabled.

- `-DWITH_MECAB={disabled|system|path_name}`

  Use this option to compile the MeCab parser. If you have installed MeCab to its default installation directory, set `-DWITH_MECAB=system`. The `system` option applies to MeCab installations performed from source or from binaries using a native package management utility. If you installed MeCab to a custom installation directory, specify the path to the MeCab installation. For example, `-DWITH_MECAB=/opt/mecab`. If the `system` option does not work, specifying the MeCab installation path should work in all cases.

  For related information, see Section 12.9.9, “MeCab Full-Text Parser Plugin”.

- `-DWITH_MSCRT_DEBUG=bool`

  Whether to enable Visual Studio CRT memory leak tracing. The default is `OFF`.

- `-DWITH_MYSQLX=bool`

  Whether to build with support for X Plugin. Default `ON`. See Chapter 20, *Using MySQL as a Document Store*.

- `-DWITH_NUMA=bool`

  Explicitly set the NUMA memory allocation policy. `CMake` sets the default `WITH_NUMA` value based on whether the current platform has `NUMA` support. For platforms without NUMA support, `CMake` behaves as follows:

  - With no NUMA option (the normal case), `CMake` continues normally, producing only this warning: NUMA library missing or required version not available
  - With `-DWITH_NUMA=ON`, `CMake` aborts with this error: NUMA library missing or required version not available

- `-DWITH_PROTOBUF=protobuf_type`

Which Protocol Buffers package to use. *protobuf_type* can be one of the following values:

- bundled: Use the package bundled with the distribution. This is the default.
- system: Use the package installed on the system.

Other values are ignored, with a fallback to bundled.

- -DWITH_RAPID=*bool*

  Whether to build the rapid development cycle plugins. When enabled, a rapid directory is created in the build tree containing these plugins. When disabled, no rapid directory is created in the build tree. The default is ON, unless the rapid directory is removed from the source tree, in which case the default becomes OFF.

- -DWITH_RAPIDJSON=*rapidjson_type*

  The type of RapidJSON library support to include. *rapidjson_type* can be one of the following values:

  - bundled: Use the RapidJSON library bundled with the distribution. This is the default.
  - system: Use the system RapidJSON library. Version 1.1.0 or higher is required.

  This option was added in MySQL 8.0.13.

- -DWITH_LZMA=*lzma_type*

  The type of LZMA library support to include. *lzma_type* can be one of the following values:

  - bundled: Use the LZMA library bundled with the distribution. This is the default.
  - system: Use the system LZMA library.

  This option was removed in MySQL 8.0.16.

- -DWITH_RE2=*re2_type*

  The type of RE2 library support to include. *re2_type* can be one of the following values:

  - bundled: Use the RE2 library bundled with the distribution. This is the default.
  - system: Use the system RE2 library.

  As of MySQL 8.0.18, MySQL no longer uses the RE2 library and this option was removed.

- -DWITH_ROUTER=*bool*

  Whether to build MySQL Router. The default is ON.

  This option was added in MySQL 8.0.16.

- -DWITH_SSL={*ssl_type*|*path_name*}

  For support of encrypted connections, entropy for random number generation, and other encryption-related operations, MySQL must be built using an SSL library. This option specifies which SSL library to use.

  - *ssl_type* can be one of the following values:

- `system`: Use the system OpenSSL library. This is the default.

  On macOS and Windows, using `system` configures MySQL to build as if CMake was invoked with *`path_name`* points to a manually installed OpenSSL library. This is because they do not have system SSL libraries. On macOS, *brew install openssl* installs to `/usr/local/opt/openssl` and `system` will find it. On Windows, it checks *%ProgramFiles%/OpenSSL*, *%ProgramFiles%/OpenSSL-Win32*, *%ProgramFiles%/OpenSSL-Win64*, *C:/OpenSSL*, *C:/OpenSSL-Win32*, and *C:/OpenSSL-Win64*.

- `yes`: This is a synonym for `system`.

- *`path_name`* is the path name to the OpenSSL installation to use. This can be preferable to using the *`ssl_type`* value of `system` because it can prevent CMake from detecting and using an older or incorrect OpenSSL version installed on the system. (Another permitted way to do the same thing is to set `WITH_SSL` to `system` and set the `CMAKE_PREFIX_PATH` option to *`path_name`*.)

For additional information about configuring the SSL library, see Section 2.9.6, “Configuring SSL Library Support”.

- `-DWITH_SYSTEMD=`*`bool`*

  Whether to enable installation of systemd support files. By default, this option is disabled. When enabled, systemd support files are installed, and scripts such as `mysqld_safe` and the System V initialization script are not installed. On platforms where systemd is not available, enabling `WITH_SYSTEMD` results in an error from `CMake`.

  For more information about using systemd, see Section 2.5.9, “Managing MySQL Server with systemd”. That section also includes information about specifying options previously specified in `[mysqld_safe]` option groups. Because `mysqld_safe` is not installed when systemd is used, such options must be specified another way.

- `-DWITH_SYSTEM_LIBS=`*`bool`*

  This option serves as an “umbrella” option to set the `system` value of any of the following `CMake` options that are not set explicitly: `WITH_CURL`, `WITH_EDITLINE`, `WITH_ICU`, `WITH_LIBEVENT`, `WITH_LZ4`, `WITH_LZMA`, `WITH_PROTOBUF`, `WITH_RE2`, `WITH_SSL`, `WITH_ZLIB`, `WITH_ZSTD`.

- `-DWITH_TEST_TRACE_PLUGIN=`*`bool`*

  Whether to build the test protocol trace client plugin (see Using the Test Protocol Trace Plugin). By default, this option is disabled. Enabling this option has no effect unless the `WITH_CLIENT_PROTOCOL_TRACING` option is enabled. If MySQL is configured with both options enabled, the `libmysqlclient` client library is built with the test protocol trace plugin built in, and all the standard MySQL clients load the plugin. However, even when the test plugin is enabled, it has no effect by default. Control over the plugin is afforded using environment variables; see Using the Test Protocol Trace Plugin.

  Note

  Do *not* enable the `WITH_TEST_TRACE_PLUGIN` option if you want to use your own protocol trace plugins because only one such plugin can be loaded at a time and an error occurs for attempts to load a second one. If you have already built MySQL with the test protocol trace plugin enabled to see how it works, you must rebuild MySQL without it before you can use your own plugins.

  For information about writing trace plugins, see Section 29.2.4.11, “Writing Protocol Trace Plugins”.

- `-DWITH_TSAN=bool`

  Whether to enable the ThreadSanitizer, for compilers that support it. The default is off.

- `-DWITH_UBSAN=bool`

  Whether to enable the Undefined Behavior Sanitizer, for compilers that support it. The default is off.

- `-DWITH_UNIT_TESTS={ON|OFF}`

  If enabled, compile MySQL with unit tests. The default is ON unless the server is not being compiled.

- `-DWITH_UNIXODBC=1`

  Enables unixODBC support, for Connector/ODBC.

- `-DWITH_VALGRIND=bool`

  Whether to compile in the Valgrind header files, which exposes the Valgrind API to MySQL code. The default is `OFF`.

  To generate a Valgrind-aware debug build, `-DWITH_VALGRIND=1` normally is combined with `-DWITH_DEBUG=1`. See Building Debug Configurations.

- `-DWITH_ZLIB=zlib_type`

  Some features require that the server be built with compression library support, such as the `COMPRESS()` and `UNCOMPRESS()` functions, and compression of the client/server protocol. The `WITH_ZLIB` indicates the source of `zlib` support:

  - `bundled`: Use the `zlib` library bundled with the distribution. This is the default.
  - `system`: Use the system `zlib` library. If `WITH_ZLIB` is set to this value, the `zlib_decompress` utility is not built. In this case, the system `openssl zlib` command can be used instead.

- `-DWITH_ZSTD=zstd_type`

  Connection compression using the `zstd` algorithm (see Section 4.2.6, “Connection Compression Control”) requires that the server be built with `zstd` library support. The `WITH_ZSTD` indicates the source of `zstd` support:

  - `bundled`: Use the `zstd` library bundled with the distribution. This is the default.
  - `system`: Use the system `zstd` library.

  This option was added in MySQL 8.0.18.

Compiler Flags

- `-DCMAKE_C_FLAGS="flags"`

  Flags for the C Compiler.

- `-DCMAKE_CXX_FLAGS="flags"`

  Flags for the C++ Compiler.

- `-DWITH_DEFAULT_COMPILER_OPTIONS=bool`

Whether to use the flags from `cmake/build_configurations/compiler_options.cmake`.

Note

All optimization flags were carefully chosen and tested by the MySQL build team. Overriding them can lead to unexpected results and is done at your own risk.

To specify your own C and C++ compiler flags, for flags that do not affect optimization, use the `CMAKE_C_FLAGS` and `CMAKE_CXX_FLAGS` CMake options.

When providing your own compiler flags, you might want to specify `CMAKE_BUILD_TYPE` as well.

For example, to create a 32-bit release build on a 64-bit Linux machine, do this:

```
mkdir bld
cd bld
cmake .. -DCMAKE_C_FLAGS=-m32 \
  -DCMAKE_CXX_FLAGS=-m32 \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

If you set flags that affect optimization (`-O`*`number`*), you must set the `CMAKE_C_FLAGS_`*`build_type`* and/or `CMAKE_CXX_FLAGS_`*`build_type`* options, where *`build_type`* corresponds to the `CMAKE_BUILD_TYPE` value. To specify a different optimization for the default build type (`RelWithDebInfo`) set the `CMAKE_C_FLAGS_RELWITHDEBINFO` and `CMAKE_CXX_FLAGS_RELWITHDEBINFO` options. For example, to compile on Linux with `-O3` and with debug symbols, do this:

```
cmake .. -DCMAKE_C_FLAGS_RELWITHDEBINFO="-O3 -g" \
  -DCMAKE_CXX_FLAGS_RELWITHDEBINFO="-O3 -g"
```

CMake Options for Compiling NDB Cluster

The following options are for use when building MySQL 8.0 sources with NDB Cluster support.

- `-DMEMCACHED_HOME=`*`dir_name`*

  Perform the build using the memcached (version 1.6 or later) installed in the system directory indicated by *`dir_name`*. Files from this installation that are used in the build include the memcached binary, header files, and libraries, as well as the `memcached_utilities` library and the header file `engine_testapp.h`.

  You must leave this option unset when building `ndbmemcache` using the bundled memcached sources (`WITH_BUNDLED_MEMCACHED` option); in other words, the bundled sources are used by default).

  While additional `CMake` options—such as for SASL authorization and for providing `dtrace` support—are available for use when compiling `memcached` from external sources, these options are currently not enabled for the `memcached` sources bundled with NDB Cluster.

- `-DWITH_BUNDLED_LIBEVENT={ON|OFF}`

  Use the `libevent` included in the NDB Cluster sources when building NDB Cluster with `ndbmemcached` support. Enabled by default. `OFF` causes the system's `libevent` to be used instead.

- `-DWITH_BUNDLED_MEMCACHED={ON|OFF}`

  Build the memcached sources included in the NDB Cluster source tree, then use the resulting memcached server when building the `ndbmemcache` engine. In this case, `make install` places the

`memcached` binary in the installation `bin` directory, and the `ndbmemcache` engine shared library file `ndb_engine.so` in the installation `lib` directory.

This option is ON by default.

- `-DWITH_CLASSPATH=path`

  Sets the classpath for building NDB Cluster Connector for Java. The default is empty. This option is ignored if `-DWITH_NDB_JAVA=OFF` is used.

- `-DWITH_ERROR_INSERT={ON|OFF}`

  Enables error injection in the `NDB` kernel. For testing only; not intended for use in building production binaries. The default is `OFF`.

- `-DWITH_NDBCLUSTER_STORAGE_ENGINE={ON|OFF}`

  This is an alias for `WITH_NDBCLUSTER`.

- `-DWITH_NDBCLUSTER={ON|OFF}`

  Build and link in support for the `NDB` (`NDBCLUSTER`) storage engine in `mysqld`. The default is `ON`.

- `-DWITH_NDBMTD={ON|OFF}`

  Build the multithreaded data node executable `ndbmtd`. The default is `ON`.

- `-DWITH_NDB_BINLOG={ON|OFF}`

  Enable binary logging by default in the `mysqld` built using this option. ON by default.

- `-DWITH_NDB_DEBUG={ON|OFF}`

  Enable building the debug versions of the NDB Cluster binaries. OFF by default.

- `-DWITH_NDB_JAVA={ON|OFF}`

  Enable building NDB Cluster with Java support, including `ClusterJ`.

  This option is ON by default. If you do not wish to compile NDB Cluster with Java support, you must disable it explicitly by specifying `-DWITH_NDB_JAVA=OFF` when running `CMake`. Otherwise, if Java cannot be found, configuration of the build fails.

- `-DWITH_NDB_PORT=port`

  Causes the NDB Cluster management server (`ndb_mgmd`) that is built to use this *`port`* by default. If this option is unset, the resulting management server tries to use port 1186 by default.

- `-DWITH_NDB_TEST={ON|OFF}`

  If enabled, include a set of NDB API test programs. The default is OFF.

- `-DWITH_PLUGIN_NDBCLUSTER={ON|OFF}`

  Alias for `WITH_NDBCLUSTER`.

2.9.8 Dealing with Problems Compiling MySQL

The solution to many problems involves reconfiguring. If you do reconfigure, take note of the following:

- If `CMake` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `CMakeCache.txt`. When `CMake` starts, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `CMake`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old object files or configuration information from being used, run the following commands before re-running `CMake`:

On Unix:

```
shell> make clean
shell> rm CMakeCache.txt
```

On Windows:

```
shell> devenv MySQL.sln /clean
shell> del CMakeCache.txt
```

If you build outside of the source tree, remove and recreate your build directory before re-running `CMake`. For instructions on building outside of the source tree, see How to Build MySQL Server with CMake.

On some systems, warnings may occur due to differences in system include files. The following list describes other problems that have been found to occur most often when compiling MySQL:

- To define which C and C++ compilers to use, you can define the `CC` and `CXX` environment variables. For example:

  ```
  shell> CC=gcc
  shell> CXX=g++
  shell> export CC CXX
  ```

  To specify your own C and C++ compiler flags, use the `CMAKE_C_FLAGS` and `CMAKE_CXX_FLAGS` CMake options. See Compiler Flags.

  To see what flags you might need to specify, invoke `mysql_config` with the `--cflags` and `--cxxflags` options.

- To see what commands are executed during the compile stage, after using `CMake` to configure MySQL, run `make VERBOSE=1` rather than just `make`.
- If compilation fails, check whether the `MYSQL_MAINTAINER_MODE` option is enabled. This mode causes compiler warnings to become errors, so disabling it may enable compilation to proceed.
- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

  ```
  make: Fatal error in reader: Makefile, line 18:
  Badly formed macro assignment
  ```

  Or:

  ```
  make: file `Makefile' line 18: Must be a separator (:
  ```

Or:

```
pthread.h: No such file or directory
```

Solaris and FreeBSD are known to have troublesome `make` programs.

GNU `make` 3.75 is known to work.

- The `sql_yacc.cc` file is generated from `sql_yacc.yy`. Normally, the build process does not need to create `sql_yacc.cc` because MySQL comes with a pregenerated copy. However, if you do need to re-create it, you might encounter this error:

  ```
  "sql_yacc.yy", line xxx fatal: default action causes potential...
  ```

  This is a sign that your version of `yacc` is deficient. You probably need to install a recent version of `bison` (the GNU version of `yacc`) and use that instead.

  Versions of `bison` older than 1.75 may report this error:

  ```
  sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
  ```

  The maximum table size is not actually exceeded; the error is caused by bugs in older versions of `bison`.

For information about acquiring or updating tools, see the system requirements in Section 2.9, “Installing MySQL from Source”.

2.9.9 MySQL Configuration and Third-Party Tools

Third-party tools that need to determine the MySQL version from the MySQL source can read the `VERSION` file in the top-level source directory. The file lists the pieces of the version separately. For example, if the version is MySQL 8.0.4-rc, the file looks like this:

```
MYSQL_VERSION_MAJOR=8
MYSQL_VERSION_MINOR=0
MYSQL_VERSION_PATCH=4
MYSQL_VERSION_EXTRA=-rc
```

If the source is not for a General Availablility (GA) release, the `MYSQL_VERSION_EXTRA` value will be nonempty. For the example, the value corresponds to Release Candidate.

To construct a five-digit number from the version components, use this formula:

```
MYSQL_VERSION_MAJOR*10000 + MYSQL_VERSION_MINOR*100 + MYSQL_VERSION_PATCH
```

2.9.10 Generating MySQL Doxygen Documentation Content

The MySQL source code contains internal documentation written using Doxygen. The generated Doxygen content is available at https://dev.mysql.com/doc/index-other.html. It is also possible to generate this content locally from a MySQL source distribution using the following procedure:

1. Install `doxygen` 1.8.11 or higher. Distributions are available here at http://www.doxygen.nl/.

   After installing `doxygen`, verify the version number:

```
shell> doxygen --version
1.8.13
```

2. Install PlantUML.

   When you install PlantUML on Windows (tested on Windows 10), you must run it at least once as administrator so it creates the registry keys. Open an administrator console and run this command:

   ```
   shell> java -jar path-to-plantuml.jar
   ```

   The command should open a GUI window and return no errors on the console.

3. Set the `PLANTUML_JAR_PATH` environment to the location where you installed PlantUML. For example:

   ```
   shell> export PLANTUML_JAR_PATH=path-to-plantuml.jar
   ```

4. Install the Graphviz `dot` command.

   After installing Graphviz, verify `dot` availability. For example:

   ```
   shell> which dot
   /usr/bin/dot

   shell> dot -V
   dot - graphviz version 2.28.0 (20130928.0220)
   ```

5. Change location to the top-level directory of your MySQL source distribution and do the following:

   First, execute `cmake`:

   ```
   shell> cd your-mysql-source-directory
   shell> mkdir bld
   shell> cd bld
   shell> cmake ..
   ```

   Next, generate the `doxygen` documentation:

   ```
   shell> make doxygen
   ```

   Inspect the error log. It is available in the `doxyerror.log` file in the top-level directory. Assuming that the build executed successfully, view the generated output using a browser. For example:

   ```
   shell> firefox doxygen/html/index.html
   ```

2.10 Postinstallation Setup and Testing

This section discusses tasks that you should perform after installing MySQL:

- If necessary, initialize the data directory and create the MySQL grant tables. For some MySQL installation methods, data directory initialization may be done for you automatically:
  - Windows installation operations performed by MySQL Installer.

- Installation on Linux using a server RPM or Debian distribution from Oracle.
- Installation using the native packaging system on many platforms, including Debian Linux, Ubuntu Linux, Gentoo Linux, and others.
- Installation on macOS using a DMG distribution.

For other platforms and installation types, you must initialize the data directory manually. These include installation from generic binary and source distributions on Unix and Unix-like system, and installation from a ZIP Archive package on Windows. For instructions, see Section 2.10.1, “Initializing the Data Directory”.

- Start the server and make sure that it can be accessed. For instructions, see Section 2.10.2, “Starting the Server”, and Section 2.10.3, “Testing the Server”.
- Assign passwords to the initial `root` account in the grant tables, if that was not already done during data directory initialization. Passwords prevent unauthorized access to the MySQL server. For instructions, see Section 2.10.4, “Securing the Initial MySQL Account”.
- Optionally, arrange for the server to start and stop automatically when your system starts and stops. For instructions, see Section 2.10.5, “Starting and Stopping MySQL Automatically”.
- Optionally, populate time zone tables to enable recognition of named time zones. For instructions, see Section 5.1.13, “MySQL Server Time Zone Support”.

When you are ready to create additional user accounts, you can find information on the MySQL access control system and account management in Section 6.2, “Access Control and Account Management”.

2.10.1 Initializing the Data Directory

After MySQL is installed, the data directory must be initialized, including the tables in the `mysql` system schema:

- For some MySQL installation methods, data directory initialization is automatic, as described in Section 2.10, “Postinstallation Setup and Testing”.
- For other installation methods, you must initialize the data directory manually. These include installation from generic binary and source distributions on Unix and Unix-like systems, and installation from a ZIP Archive package on Windows.

This section describes how to initialize the data directory manually for MySQL installation methods for which data directory initialization is not automatic. For some suggested commands that enable testing whether the server is accessible and working properly, see Section 2.10.3, “Testing the Server”.

Note

In MySQL 8.0, the default authentication plugin has changed from `mysql_native_password` to `caching_sha2_password`, and the `'root'@'localhost'` administrative account uses `caching_sha2_password` by default. If you prefer that the `root` account use the previous default authentication plugin (`mysql_native_password`), see caching_sha2_password and the root Administrative Account.

- Data Directory Initialization Overview
- Data Directory Initialization Procedure

- Server Actions During Data Directory Initialization
- Post-Initialization root Password Assignment

Data Directory Initialization Overview

In the examples shown here, the server is intended to run under the user ID of the `mysql` login account. Either create the account if it does not exist (see Create a mysql User and Group), or substitute the name of a different existing login account that you plan to use for running the server.

1. Change location to the top-level directory of your MySQL installation, which is typically `/usr/local/mysql` (adjust the path name for your system as necessary):

   ```
   cd /usr/local/mysql
   ```

   Within the directory you will find several files and subdirectories, including the `bin` subdirectory that contains the server as well as client and utility programs.

2. The `secure_file_priv` system variable limits import and export operations to a specific directory. Create a directory whose location can be specified as the value of that variable:

   ```
   mkdir mysql-files
   ```

   Grant directory user and group ownership to the `mysql` user and `mysql` group, and set the directory permissions appropriately:

   ```
   chown mysql:mysql mysql-files
   chmod 750 mysql-files
   ```

3. Use the server to initialize the data directory, including the `mysql` schema containing the initial MySQL grant tables that determine how users are permitted to connect to the server. For example:

   ```
   bin/mysqld --initialize --user=mysql
   ```

   For important information about the command, especially regarding command options you might use, see Data Directory Initialization Procedure. For details about how the server performs initialization, see Server Actions During Data Directory Initialization.

   Typically, data directory initialization need be done only after you first install MySQL. (For upgrades to an existing installation, perform the upgrade procedure instead; see Section 2.11, “Upgrading MySQL”.) However, the command that initializes the data directory does not overwrite any existing `mysql` schema tables, so it is safe to run in any circumstances.

4. If you want to deploy the server with automatic support for secure connections, use the `mysql_ssl_rsa_setup` utility to create default SSL and RSA files:

   ```
   bin/mysql_ssl_rsa_setup
   ```

   For more information, see Section 4.4.3, “`mysql_ssl_rsa_setup` — Create SSL/RSA Files”.

5. In the absence of any option files, the server starts with its default settings. (See Section 5.1.2, “Server Configuration Defaults”.) To explicitly specify options that the MySQL server should use at startup, put them in an option file such as `/etc/my.cnf` or `/etc/mysql/my.cnf`. (See Section 4.2.2.2, “Using Option Files”.) For example, you can use an option file to set the `secure_file_priv` system variable.

6. To arrange for MySQL to start without manual intervention at system boot time, see Section 2.10.5, “Starting and Stopping MySQL Automatically”.

7. Data directory initialization creates time zone tables in the `mysql` schema but does not populate them. To do so, use the instructions in Section 5.1.13, “MySQL Server Time Zone Support”.

Data Directory Initialization Procedure

Change location to the top-level directory of your MySQL installation, which is typically `/usr/local/mysql` (adjust the path name for your system as necessary):

```
cd /usr/local/mysql
```

To initialize the data directory, invoke `mysqld` with the `--initialize` or `--initialize-insecure` option, depending on whether you want the server to generate a random initial password for the `'root'@'localhost'` account, or to create that account with no password:

- Use `--initialize` for “secure by default” installation (that is, including generation of a random initial `root` password). In this case, the password is marked as expired and you will need to choose a new one.
- With `--initialize-insecure`, no `root` password is generated. This is insecure; it is assumed that you will assign a password to the account in timely fashion before putting the server into production use.

For instructions on assigning a new `'root'@'localhost'` password, see Post-Initialization root Password Assignment.

Note

The server writes any messages (including any initial password) to its standard error output. This may be redirected to the error log, so look there if you do not see the messages on your screen. For information about the error log, including where it is located, see Section 5.4.2, “The Error Log”.

On Windows, use the `--console` option to direct messages to the console.

On Unix and Unix-like systems, it is important for the database directories and files to be owned by the `mysql` login account so that the server has read and write access to them when you run it later. To ensure this, start `mysqld` from the system `root` account and include the `--user` option as shown here:

```
bin/mysqld --initialize --user=mysql
bin/mysqld --initialize-insecure --user=mysql
```

Alternatively, execute `mysqld` while logged in as `mysql`, in which case you can omit the `--user` option from the command.

On Windows, use one of these commands:

```
bin\mysqld --initialize --console
bin\mysqld --initialize-insecure --console
```

Note

Data directory initialization might fail if required system libraries are missing. For example, you might see an error like this:

```
bin/mysqld: error while loading shared libraries:
libnuma.so.1: cannot open shared object file:
No such file or directory
```

If this happens, you must install the missing libraries manually or with your system's package manager. Then retry the data directory initialization command.

It might be necessary to specify other options such as `--basedir` or `--datadir` if `mysqld` cannot identify the correct locations for the installation directory or data directory. For example (enter the command on a single line):

```
bin/mysqld --initialize --user=mysql
  --basedir=/opt/mysql/mysql
  --datadir=/opt/mysql/mysql/data
```

Alternatively, put the relevant option settings in an option file and pass the name of that file to `mysqld`. For Unix and Unix-like systems, suppose that the option file name is `/opt/mysql/mysql/etc/my.cnf`. Put these lines in the file:

```
[mysqld]
basedir=/opt/mysql/mysql
datadir=/opt/mysql/mysql/data
```

Then invoke `mysqld` as follows (enter the command on a single line with the `--defaults-file` option first):

```
bin/mysqld --defaults-file=/opt/mysql/mysql/etc/my.cnf
  --initialize --user=mysql
```

On Windows, suppose that `C:\my.ini` contains these lines:

```
[mysqld]
basedir=C:\\Program Files\\MySQL\\MySQL Server 8.0
datadir=D:\\MySQLdata
```

Then invoke `mysqld` as follows (enter the command on a single line with the `--defaults-file` option first):

```
bin\mysqld --defaults-file=C:\my.ini
   --initialize --console
```

Server Actions During Data Directory Initialization

Note

The data directory initialization sequence performed by the server does not substitute for the actions performed by `mysql_secure_installation` and `mysql_ssl_rsa_setup`. See Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”, and Section 4.4.3, “`mysql_ssl_rsa_setup` — Create SSL/RSA Files”.

When invoked with the `--initialize` or `--initialize-insecure` option, `mysqld` performs the following actions during the data directory initialization sequence:

1. The server checks for the existence of the data directory as follows:

   - If no data directory exists, the server creates it.
   - If the data directory exists but is not empty (that is, it contains files or subdirectories), the server exits after producing an error message:

     ```
     [ERROR] --initialize specified but the data directory exists. Aborting.
     ```

     In this case, remove or rename the data directory and try again.

     An existing data directory is permitted to be nonempty if every entry has a name that begins with a period (`.`).

2. Within the data directory, the server creates the `mysql` system schema and its tables, including the data dictionary tables, grant tables, time zone tables, and server-side help tables. See Section 5.3, "The mysql System Schema".

3. The server initializes the system tablespace and related data structures needed to manage `InnoDB` tables.

   Note

   After `mysqld` sets up the `InnoDB` system tablespace, certain changes to tablespace characteristics require setting up a whole new instance. Qualifying changes include the file name of the first file in the system tablespace and the number of undo logs. If you do not want to use the default values, make sure that the settings for the `innodb_data_file_path` and `innodb_log_file_size` configuration parameters are in place in the MySQL configuration file *before* running `mysqld`. Also make sure to specify as necessary other parameters that affect the creation and location of `InnoDB` files, such as `innodb_data_home_dir` and `innodb_log_group_home_dir`.

   If those options are in your configuration file but that file is not in a location that MySQL reads by default, specify the file location using the `--defaults-extra-file` option when you run `mysqld`.

4. The server creates a `'root'@'localhost'` superuser account and other reserved accounts (see Section 6.2.9, "Reserved Accounts"). Some reserved accounts are locked and cannot be used by clients, but `'root'@'localhost'` is intended for administrative use and you should assign it a password.

   Server actions with respect to a password for the `'root'@'localhost'` account depend on how you invoke it:

   - With `--initialize` but not `--initialize-insecure`, the server generates a random password, marks it as expired, and writes a message displaying the password:

     ```
     [Warning] A temporary password is generated for root@localhost:
     iTag*AfrH5ej
     ```

   - With `--initialize-insecure`, (either with or without `--initialize` because `--initialize-insecure` implies `--initialize`), the server does not generate a password or mark it expired, and writes a warning message:

     ```
     ```

```
[Warning] root@localhost is created with an empty password ! Please
consider switching off the --initialize-insecure option.
```

For instructions on assigning a new `'root'@'localhost'` password, see Post-Initialization root Password Assignment.

5. The server populates the server-side help tables used for the `HELP` statement (see Section 13.8.3, “HELP Syntax”). The server does not populate the time zone tables. To do so manually, see Section 5.1.13, “MySQL Server Time Zone Support”.

6. If the `init_file` system variable was given to name a file of SQL statements, the server executes the statements in the file. This option enables you to perform custom bootstrapping sequences.

   When the server operates in bootstrap mode, some functionality is unavailable that limits the statements permitted in the file. These include statements that relate to account management (such as `CREATE USER` or `GRANT`), replication, and global transaction identifiers.

7. The server exits.

Post-Initialization root Password Assignment

After you initialize the data directory by starting the server with `--initialize` or `--initialize-insecure`, start the server normally (that is, without either of those options) and assign the `'root'@'localhost'` account a new password:

1. Start the server. For instructions, see Section 2.10.2, “Starting the Server”.

2. Connect to the server:

   - If you used `--initialize` but not `--initialize-insecure` to initialize the data directory, connect to the server as `root`:

     ```
     mysql -u root -p
     ```

     Then, at the password prompt, enter the random password that the server generated during the initialization sequence:

     ```
     Enter password: (enter the random root password here)
     ```

     Look in the server error log if you do not know this password.

   - If you used `--initialize-insecure` to initialize the data directory, connect to the server as `root` without a password:

     ```
     mysql -u root --skip-password
     ```

3. After connecting, use an `ALTER USER` statement to assign a new `root` password:

   ```
   ALTER USER 'root'@'localhost' IDENTIFIED BY 'root-password';
   ```

See also Section 2.10.4, “Securing the Initial MySQL Account”.

Note

Attempts to connect to the host `127.0.0.1` normally resolve to the `localhost` account. However, this fails if the server is run with the `--skip-name-`

`resolve` option. If you plan to do that, make sure that an account exists that can accept a connection. For example, to be able to connect as `root` using `--host=127.0.0.1` or `--host=::1`, create these accounts:

```
CREATE USER 'root'@'127.0.0.1' IDENTIFIED BY 'root-password';
CREATE USER 'root'@'::1' IDENTIFIED BY 'root-password';
```

It is possible to put those statements in a file to be executed using the `init_file` system variable, as discussed in Server Actions During Data Directory Initialization.

2.10.2 Starting the Server

This section describes how start the server on Unix and Unix-like systems. (For Windows, see Section 2.3.4.5, “Starting the Server for the First Time”.) For some suggested commands that you can use to test whether the server is accessible and working properly, see Section 2.10.3, “Testing the Server”.

Start the MySQL server like this if your installation includes `mysqld_safe`:

```
shell> bin/mysqld_safe --user=mysql &
```

Note

For Linux systems on which MySQL is installed using RPM packages, server startup and shutdown is managed using systemd rather than `mysqld_safe`, and `mysqld_safe` is not installed. See Section 2.5.9, “Managing MySQL Server with systemd”.

Start the server like this if your installation includes systemd support:

```
shell> systemctl start mysqld
```

Substitute the appropriate service name if it differs from `mysqld` (for example, `mysql` on SLES systems).

It is important that the MySQL server be run using an unprivileged (non-`root`) login account. To ensure this, run `mysqld_safe` as `root` and include the `--user` option as shown. Otherwise, you should execute the program while logged in as `mysql`, in which case you can omit the `--user` option from the command.

For further instructions for running MySQL as an unprivileged user, see Section 6.1.5, “How to Run MySQL as a Normal User”.

If the command fails immediately and prints `mysqld ended`, look for information in the error log (which by default is the *`host_name`*`.err` file in the data directory).

If the server is unable to access the data directory it starts or read the grant tables in the `mysql` schema, it writes a message to its error log. Such problems can occur if you neglected to create the grant tables by initializing the data directory before proceeding to this step, or if you ran the command that initializes the data directory without the `--user` option. Remove the `data` directory and run the command with the `--user` option.

If you have other problems starting the server, see Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”. For more information about `mysqld_safe`, see Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”. For more information about systemd support, see Section 2.5.9, “Managing MySQL Server with systemd”.

2.10.2.1 Troubleshooting Problems Starting the MySQL Server

This section provides troubleshooting suggestions for problems starting the server. For additional suggestions for Windows systems, see Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”.

If you have problems starting the server, here are some things to try:

- Check the error log to see why the server does not start. Log files are located in the data directory (typically `C:\Program Files\MySQL\MySQL Server 8.0\data` on Windows, `/usr/local/mysql/data` for a Unix/Linux binary distribution, and `/usr/local/var` for a Unix/Linux source distribution). Look in the data directory for files with names of the form *`host_name`*`.err` and *`host_name`*`.log`, where *`host_name`* is the name of your server host. Then examine the last few lines of these files. Use `tail` to display them:

  ```
  shell> tail host_name.err
  shell> tail host_name.log
  ```

- Specify any special options needed by the storage engines you are using. You can create a `my.cnf` file and specify startup options for the engines that you plan to use. If you are going to use storage engines that support transactional tables (`InnoDB`, `NDB`), be sure that you have them configured the way you want before starting the server. If you are using `InnoDB` tables, see Section 15.8, “InnoDB Configuration” for guidelines and Section 15.13, “InnoDB Startup Options and System Variables” for option syntax.

  Although storage engines use default values for options that you omit, Oracle recommends that you review the available options and specify explicit values for any options whose defaults are not appropriate for your installation.

- Make sure that the server knows where to find the data directory. The `mysqld` server uses this directory as its current directory. This is where it expects to find databases and where it expects to write log files. The server also writes the pid (process ID) file in the data directory.

  The default data directory location is hardcoded when the server is compiled. To determine what the default path settings are, invoke `mysqld` with the `--verbose` and `--help` options. If the data directory is located somewhere else on your system, specify that location with the `--datadir` option to `mysqld` or `mysqld_safe`, on the command line or in an option file. Otherwise, the server will not work properly. As an alternative to the `--datadir` option, you can specify `mysqld` the location of the base directory under which MySQL is installed with the `--basedir`, and `mysqld` looks for the `data` directory there.

  To check the effect of specifying path options, invoke `mysqld` with those options followed by the `--verbose` and `--help` options. For example, if you change location to the directory where `mysqld` is installed and then run the following command, it shows the effect of starting the server with a base directory of `/usr/local`:

  ```
  shell> ./mysqld --basedir=/usr/local --verbose --help
  ```

  You can specify other options such as `--datadir` as well, but `--verbose` and `--help` must be the last options.

  Once you determine the path settings you want, start the server without `--verbose` and `--help`.

  If `mysqld` is currently running, you can find out what path settings it is using by executing this command:

  ```
  shell> mysqladmin variables
  ```

Or:

```
shell> mysqladmin -h host_name variables
```

host_name is the name of the MySQL server host.

- Make sure that the server can access the data directory. The ownership and permissions of the data directory and its contents must allow the server to read and modify them.

  If you get `Errcode 13` (which means `Permission denied`) when starting `mysqld`, this means that the privileges of the data directory or its contents do not permit server access. In this case, you change the permissions for the involved files and directories so that the server has the right to use them. You can also start the server as `root`, but this raises security issues and should be avoided.

  Change location to the data directory and check the ownership of the data directory and its contents to make sure the server has access. For example, if the data directory is `/usr/local/mysql/var`, use this command:

  ```
  shell> ls -la /usr/local/mysql/var
  ```

  If the data directory or its files or subdirectories are not owned by the login account that you use for running the server, change their ownership to that account. If the account is named `mysql`, use these commands:

  ```
  shell> chown -R mysql /usr/local/mysql/var
  shell> chgrp -R mysql /usr/local/mysql/var
  ```

  Even with correct ownership, MySQL might fail to start up if there is other security software running on your system that manages application access to various parts of the file system. In this case, reconfigure that software to enable `mysqld` to access the directories it uses during normal operation.

- Verify that the network interfaces the server wants to use are available.

  If either of the following errors occur, it means that some other program (perhaps another `mysqld` server) is using the TCP/IP port or Unix socket file that `mysqld` is trying to use:

  ```
  Can't start server: Bind on TCP/IP port: Address already in use
  Can't start server: Bind on unix socket...
  ```

  Use `ps` to determine whether you have another `mysqld` server running. If so, shut down the server before starting `mysqld` again. (If another server is running, and you really want to run multiple servers, you can find information about how to do so in Section 5.8, “Running Multiple MySQL Instances on One Machine”.)

  If no other server is running, execute the command `telnet your_host_name tcp_ip_port_number`. (The default MySQL port number is 3306.) Then press Enter a couple of times. If you do not get an error message like `telnet: Unable to connect to remote host: Connection refused`, some other program is using the TCP/IP port that `mysqld` is trying to use. Track down what program this is and disable it, or tell `mysqld` to listen to a different port with the `--port` option. In this case, specify the same non-default port number for client programs when connecting to the server using TCP/IP.

  Another reason the port might be inaccessible is that you have a firewall running that blocks connections to it. If so, modify the firewall settings to permit access to the port.

If the server starts but you cannot connect to it, make sure that you have an entry in `/etc/hosts` that looks like this:

```
127.0.0.1       localhost
```

- If you cannot get `mysqld` to start, try to make a trace file to find the problem by using the `--debug` option. See Section 29.5.4, “The DBUG Package”.

2.10.3 Testing the Server

After the data directory is initialized and you have started the server, perform some simple tests to make sure that it works satisfactorily. This section assumes that your current location is the MySQL installation directory and that it has a `bin` subdirectory containing the MySQL programs used here. If that is not true, adjust the command path names accordingly.

Alternatively, add the `bin` directory to your `PATH` environment variable setting. That enables your shell (command interpreter) to find MySQL programs properly, so that you can run a program by typing only its name, not its path name. See Section 4.2.7, “Setting Environment Variables”.

Use `mysqladmin` to verify that the server is running. The following commands provide simple tests to check whether the server is up and responding to connections:

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

If you cannot connect to the server, specify a `-u root` option to connect as `root`. If you have assigned a password for the `root` account already, you'll also need to specify `-p` on the command line and enter the password when prompted. For example:

```
shell> bin/mysqladmin -u root -p version
Enter password: (enter root password here)
```

The output from `mysqladmin version` varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

```
shell> bin/mysqladmin version
mysqladmin  Ver 14.12 Distrib 8.0.19, for pc-linux-gnu on i686
...

Server version          8.0.19
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 14 days 5 hours 5 min 21 sec

Threads: 1  Questions: 366  Slow queries: 0
Opens: 0  Flush tables: 1  Open tables: 19
Queries per second avg: 0.000
```

To see what else you can do with `mysqladmin`, invoke it with the `--help` option.

Verify that you can shut down the server (include a `-p` option if the `root` account has a password already):

```
shell> bin/mysqladmin -u root shutdown
```

Verify that you can start the server again. Do this by using `mysqld_safe` or by invoking `mysqld` directly. For example:

```
shell> bin/mysqld_safe --user=mysql &
```

If `mysqld_safe` fails, see Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”.

Run some simple tests to verify that you can retrieve information from the server. The output should be similar to that shown here.

Use `mysqlshow` to see what databases exist:

```
shell> bin/mysqlshow
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
```

The list of installed databases may vary, but always includes at least `mysql` and `information_schema`.

If you specify a database name, `mysqlshow` displays a list of the tables within the database:

```
shell> bin/mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
| columns_priv              |
| component                 |
| db                        |
| default_roles             |
| engine_cost               |
| func                      |
| general_log               |
| global_grants             |
| gtid_executed             |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| innodb_index_stats        |
| innodb_table_stats        |
| ndb_binlog_index          |
| password_history          |
| plugin                    |
| procs_priv                |
| proxies_priv              |
| role_edges                |
| server_cost               |
| servers                   |
| slave_master_info         |
| slave_relay_log_info      |
| slave_worker_info         |
| slow_log                  |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
```

```
| time_zone_transition_type |
| user                      |
+---------------------------+
```

Use the `mysql` program to select information from a table in the `mysql` schema:

```
shell> bin/mysql -e "SELECT User, Host, plugin FROM mysql.user" mysql
+------+-----------+-----------------------+
| User | Host      | plugin                |
+------+-----------+-----------------------+
| root | localhost | caching_sha2_password |
+------+-----------+-----------------------+
```

At this point, your server is running and you can access it. To tighten security if you have not yet assigned a password to the initial account, follow the instructions in Section 2.10.4, “Securing the Initial MySQL Account”.

For more information about `mysql`, `mysqladmin`, and `mysqlshow`, see Section 4.5.1, “`mysql` — The MySQL Command-Line Client”, Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”, and Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”.

2.10.4 Securing the Initial MySQL Account

The MySQL installation process involves initializing the data directory, including the grant tables in the `mysql` system schema that define MySQL accounts. For details, see Section 2.10.1, “Initializing the Data Directory”.

This section describes how to assign a password to the initial `root` account created during the MySQL installation procedure, if you have not already done so.

Note

Alternative means for performing the process described in this section:

- On Windows, you can perform the process during installation with MySQL Installer (see Section 2.3.3, “MySQL Installer for Windows”).
- On all platforms, the MySQL distribution includes `mysql_secure_installation`, a command-line utility that automates much of the process of securing a MySQL installation.
- On all platforms, MySQL Workbench is available and offers the ability to manage user accounts (see Chapter 31, *MySQL Workbench*).

A password may already be assigned to the initial account under these circumstances:

- On Windows, installations performed using MySQL Installer give you the option of assigning a password.
- Installation using the macOS installer generates an initial random password, which the installer displays to the user in a dialog box.
- Installation using RPM packages generates an initial random password, which is written to the server error log.
- Installations using Debian packages give you the option of assigning a password.
- For data directory initialization performed manually using `mysqld --initialize`, `mysqld` generates an initial random password, marks it expired, and writes it to the server error log. See Section 2.10.1, “Initializing the Data Directory”.

The `mysql.user` grant table defines the initial MySQL user account and its access privileges. Installation of MySQL creates only a `'root'@'localhost'` superuser account that has all privileges and can do anything. If the `root` account has an empty password, your MySQL installation is unprotected: Anyone can connect to the MySQL server as `root` *without a password* and be granted all privileges.

The `'root'@'localhost'` account also has a row in the `mysql.proxies_priv` table that enables granting the `PROXY` privilege for `''@''`, that is, for all users and all hosts. This enables `root` to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. See Section 6.2.18, “Proxy Users”.

To assign a password for the initial MySQL `root` account, use the following procedure. Replace *`root-password`* in the examples with the password that you want to use.

Start the server if it is not running. For instructions, see Section 2.10.2, “Starting the Server”.

The initial `root` account may or may not have a password. Choose whichever of the following procedures applies:

- If the `root` account exists with an initial random password that has been expired, connect to the server as `root` using that password, then choose a new password. This is the case if the data directory was initialized using `mysqld --initialize`, either manually or using an installer that does not give you the option of specifying a password during the install operation. Because the password exists, you must use it to connect to the server. But because the password is expired, you cannot use the account for any purpose other than to choose a new password, until you do choose one.

  1. If you do not know the initial random password, look in the server error log.
  2. Connect to the server as `root` using the password:

     ```
     shell> mysql -u root -p
     Enter password: (enter the random root password here)
     ```

  3. Choose a new password to replace the random password:

     ```
     mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'root-password';
     ```

- If the `root` account exists but has no password, connect to the server as `root` using no password, then assign a password. This is the case if you initialized the data directory using `mysqld --initialize-insecure`.

  1. Connect to the server as `root` using no password:

     ```
     shell> mysql -u root --skip-password
     ```

  2. Assign a password:

     ```
     mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'root-password';
     ```

After assigning the `root` account a password, you must supply that password whenever you connect to the server using the account. For example, to connect to the server using the `mysql` client, use this command:

```
shell> mysql -u root -p
Enter password: (enter root password here)
```

To shut down the server with `mysqladmin`, use this command:

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

Note

For additional information about setting passwords, see Section 6.2.14, “Assigning Account Passwords”. If you forget your `root` password after setting it, see Section B.4.3.2, “How to Reset the Root Password”.

To set up additional accounts, see Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”.

2.10.5 Starting and Stopping MySQL Automatically

This section discusses methods for starting and stopping the MySQL server.

Generally, you start the `mysqld` server in one of these ways:

- Invoke `mysqld` directly. This works on any platform.
- On Windows, you can set up a MySQL service that runs automatically when Windows starts. See Section 2.3.4.8, “Starting MySQL as a Windows Service”.
- On Unix and Unix-like systems, you can invoke `mysqld_safe`, which tries to determine the proper options for `mysqld` and then runs it with those options. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.
- On Linux systems that support systemd, you can use it to control the server. See Section 2.5.9, “Managing MySQL Server with systemd”.
- On systems that use System V-style run directories (that is, `/etc/init.d` and run-level specific directories), invoke `mysql.server`. This script is used primarily at system startup and shutdown. It usually is installed under the name `mysql`. The `mysql.server` script starts the server by invoking `mysqld_safe`. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.
- On macOS, install a launchd daemon to enable automatic MySQL startup at system startup. The daemon starts the server by invoking `mysqld_safe`. For details, see Section 2.4.3, “Installing and Using the MySQL Launch Daemon”. A MySQL Preference Pane also provides control for starting and stopping MySQL through the System Preferences. See Section 2.4.4, “Installing and Using the MySQL Preference Pane”.
- On Solaris, use the service management framework (SMF) system to initiate and control MySQL startup.

systemd, the `mysqld_safe` and `mysql.server` scripts, Solaris SMF, and the macOS Startup Item (or MySQL Preference Pane) can be used to start the server manually, or automatically at system startup time. systemd, `mysql.server`, and the Startup Item also can be used to stop the server.

The following table shows which option groups the server and startup scripts read from option files.

Table 2.14 MySQL Startup Scripts and Supported Server Option Groups

Script	Option Groups
`mysqld`	`[mysqld]`, `[server]`, `[mysqld-major_version]`
`mysqld_safe`	`[mysqld]`, `[server]`, `[mysqld_safe]`

Script	Option Groups
`mysql.server`	`[mysqld]`, `[mysql.server]`, `[server]`

`[mysqld-major_version]` means that groups with names like `[mysqld-5.7]` and `[mysqld-8.0]` are read by servers having versions 5.7.x, 8.0.x, and so forth. This feature can be used to specify options that can be read only by servers within a given release series.

For backward compatibility, `mysql.server` also reads the `[mysql_server]` group and `mysqld_safe` also reads the `[safe_mysqld]` group. To be current, you should update your option files to use the `[mysql.server]` and `[mysqld_safe]` groups instead.

For more information on MySQL configuration files and their structure and contents, see Section 4.2.2.2, “Using Option Files”.

2.11 Upgrading MySQL

This section describes the steps to upgrade a MySQL installation.

Upgrading is a common procedure, as you pick up bug fixes within the same MySQL release series or significant features between major MySQL releases. You perform this procedure first on some test systems to make sure everything works smoothly, and then on the production systems.

Note

In the following discussion, MySQL commands that must be run using a MySQL account with administrative privileges include `-u root` on the command line to specify the MySQL `root` user. Commands that require a password for `root` also include a `-p` option. Because `-p` is followed by no option value, such commands prompt for the password. Type the password when prompted and press Enter.

SQL statements can be executed using the `mysql` command-line client (connect as `root` to ensure that you have the necessary privileges).

2.11.1 Before You Begin

Review the information in this section before upgrading. Perform any recommended actions.

- Understand what may occur during an upgrade. See Section 2.11.3, “What the MySQL Upgrade Process Upgrades”.
- Protect your data by creating a backup. The backup should include the `mysql` system database, which contains the MySQL data dictionary tables and system tables. See Section 7.2, “Database Backup Methods”.

Important

Downgrade from MySQL 8.0 to MySQL 5.7, or from a MySQL 8.0 release to a previous MySQL 8.0 release, is not supported. The only supported alternative is to restore a backup taken *before* upgrading. It is therefore imperative that you backup your data before starting the upgrade process.

- Review Section 2.11.2, “Upgrade Paths” to ensure that your intended upgrade path is supported.
- Review Section 2.11.4, “Changes in MySQL 8.0” for changes that you should be aware of before upgrading. Some changes may require action.

- Review Section 1.4, “What Is New in MySQL 8.0” for deprecated and removed features. An upgrade may require changes with respect to those features if you use any of them.
- Review Section 1.5, “Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 8.0”. If you use deprecated or removed variables, an upgrade may require configuration changes.
- Review the Release Notes for information about fixes, changes, and new features.
- If you use replication, review Section 17.5.3, “Upgrading a Replication Setup”.
- Upgrade procedures vary by platform and how the initial installation was performed. Use the procedure that applies to your current MySQL installation:
  - For binary and package-based installations on non-Windows platforms, refer to Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”.

    Note

    For supported Linux distributions, the preferred method for upgrading package-based installations is to use the MySQL software repositories (MySQL Yum Repository, MySQL APT Repository, and MySQL SLES Repository).
  - For installations on an Enterprise Linux platform or Fedora using the MySQL Yum Repository, refer to Section 2.11.7, “Upgrading MySQL with the MySQL Yum Repository”.
  - For installations on Ubuntu using the MySQL APT repository, refer to Section 2.11.8, “Upgrading MySQL with the MySQL APT Repository”.
  - For installations on SLES using the MySQL SLES repository, refer to Section 2.11.9, “Upgrading MySQL with the MySQL SLES Repository”.
  - For installations performed using Docker, refer to Section 2.11.11, “Upgrading a Docker Installation of MySQL”.
  - For installations on Windows, refer to Section 2.11.10, “Upgrading MySQL on Windows”.
- If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, it may be useful to create a test instance for assessing the conversions that are required and the work involved to perform them. To create a test instance, make a copy of your MySQL instance that contains the `mysql` database and other databases without the data. Run the upgrade procedure on the test instance to assess the work involved to perform the actual data conversion.
- Rebuilding and reinstalling MySQL language interfaces is recommended when you install or upgrade to a new release of MySQL. This applies to MySQL interfaces such as PHP `mysql` extensions and the Perl `DBD::mysql` module.

2.11.2 Upgrade Paths

- Upgrade from MySQL 5.7 to 8.0 is supported. However, upgrade is only supported between General Availability (GA) releases. For MySQL 8.0, it is required that you upgrade from a MySQL 5.7 GA release (5.7.9 or higher). Upgrades from non-GA releases of MySQL 5.7 are not supported.
- Upgrading to the latest release is recommended before upgrading to the next version. For example, upgrade to the latest MySQL 5.7 release before upgrading to MySQL 8.0.
- Upgrade that skips versions is not supported. For example, upgrading directly from MySQL 5.6 to 8.0 is not supported.

- Once a release series reaches General Availability (GA) status, upgrade within the release series (from one GA version to another GA version) is supported. For example, upgrading from MySQL 8.0.*x* to 8.0.*y* is supported. (Upgrade involving development-status non-GA releases is not supported.) Skipping a release is also supported. For example, upgrading from MySQL 8.0.*x* to 8.0.*z* is supported. MySQL 8.0.11 is the first GA status release within the MySQL 8.0 release series.

2.11.3 What the MySQL Upgrade Process Upgrades

Installing a new version of MySQL may require upgrading these parts of the existing installation:

- The `mysql` system schema, which contains tables that store information required by the MySQL server as it runs (see Section 5.3, “The mysql System Schema”). `mysql` schema tables fall into two broad categories:
  - Data dictionary tables, which store database object metadata.
  - System tables (that is, the remaining non-data dictionary tables), which are used for other operational purposes.
- Other schemas, some of which are built in and may be considered “owned” by the server, and others which are not:
  - The Performance Schema, `INFORMATION_SCHEMA`, and `sys` schema.
  - User schemas.

Two distinct version numbers are associated with parts of the installation that may require upgrading:

- The data dictionary version. This applies to the data dictionary tables.
- The server version, also known as the MySQL version. This applies to the system tables and objects in other schemas.

In both cases, the actual version applicable to the existing MySQL installation is stored in the data dictionary, and the current expected version is compiled into the new version of MySQL. When an actual version is lower than the current expected version, those parts of the installation associated with that version must be upgraded to the current version. If both versions indicate an upgrade is needed, the data dictionary upgrade must occur first.

As a reflection of the two distinct versions just mentioned, the upgrade occurs in two steps:

- Step 1: Data dictionary upgrade.

  This step upgrades:

  - The data dictionary tables in the `mysql` schema. If the actual data dictionary version is lower than the current expected version, the server creates data dictionary tables with updated definitions, copies persisted metadata to the new tables, atomically replaces the old tables with the new ones, and reinitializes the data dictionary.
  - The Performance Schema and `INFORMATION_SCHEMA`.
- Step 2: Server upgrade.

  This step comprises all other upgrade tasks. If the server version of the existing MySQL installation is lower than that of the new installed MySQL version, everything else must be upgraded:

  - The system tables in the `mysql` schema (the remaining non-data dictionary tables).

- The `sys` schema.
- User schemas.

The data dictionary upgrade (step 1) is the responsibility of the server, which performs this task as necessary at startup unless invoked with an option that prevents it from doing so. The option is `--upgrade=NONE` as of MySQL 8.0.16, `--no-dd-upgrade` prior to MySQL 8.0.16.

If the data dictionary is out of date but the server is prevented from upgrading it, the server will not run and exits with an error. For example:

```
[ERROR] [MY-013381] [Server] Server shutting down because upgrade is
required, yet prohibited by the command line option '--upgrade=NONE'.
[ERROR] [MY-010334] [Server] Failed to initialize DD Storage Engine
[ERROR] [MY-010020] [Server] Data Dictionary initialization failed.
```

Some changes to the responsibility for step 2 occurred in MySQL 8.0.16:

- Prior to MySQL 8.0.16, `mysql_upgrade` upgrades the Performance Schema, the `INFORMATION_SCHEMA`, and the objects described in step 2. The DBA is expected to invoke `mysql_upgrade` manually after starting the server.
- As of MySQL 8.0.16, the server performs all tasks previously handled by `mysql_upgrade`. Although upgrading remains a two-step operation, the server performs them both, resulting in a simpler process.

Depending on the version of MySQL to which you are upgrading, the instructions in In-Place Upgrade and Logical Upgrade indicate whether the server performs all upgrade tasks or whether you must also invoke `mysql_upgrade` after server startup.

Note

Because the server upgrades the Performance Schema, `INFORMATION_SCHEMA`, and the objects described in step 2 as of MySQL 8.0.16, `mysql_upgrade` is unneeded and is deprecated as of that version, and will be removed in a future MySQL version.

Most aspects of what occurs during step 2 are the same prior to and as of MySQL 8.0.16, although different command options may be needed to achieve a particular effect.

As of MySQL 8.0.16, the `--upgrade` server option controls whether and how the server performs an automatic upgrade at startup:

- With no option or with `--upgrade=AUTO`, the server upgrades anything it determines to be out of date (steps 1 and 2).
- With `--upgrade=NONE`, the server upgrades nothing (skips steps 1 and 2), but also exits with an error if the data dictionary must be upgraded. It is not possible to run the server with an out-of-date data dictionary; the server insists on either upgrading it or exiting.
- With `--upgrade=MINIMAL`, the server upgrades the data dictionary, the Performance Schema, and the `INFORMATION_SCHEMA`, if necessary (step 1). Note that following an upgrade with this option, Group Replication cannot be started, because system tables on which the replication internals depend are not updated, and reduced functionality might also be apparent in other areas.
- With `--upgrade=FORCE`, the server upgrades the data dictionary, the Performance Schema, and the `INFORMATION_SCHEMA`, if necessary (step 1), and forces an upgrade of everything else (step 2). Expect server startup to take longer with this option because the server checks all objects in all schemas.

`FORCE` is useful to force step 2 actions to be performed if the server thinks they are not necessary. One way that `FORCE` differs from `AUTO` is that with `FORCE`, the server re-creates system tables such as help tables or time zone tables if they are missing.

The following list shows upgrade commands prior to MySQL 8.0.16 and the equivalent commands for MySQL 8.0.16 and higher:

- Perform a normal upgrade (steps 1 and 2 as necessary):
  - Prior to MySQL 8.0.16: `mysqld` followed by `mysql_upgrade`
  - As of MySQL 8.0.16: `mysqld`
- Perform only step 1 as necessary:
  - Prior to MySQL 8.0.16: It is not possible to perform all upgrade tasks described in step 1 while excluding those described in step 2. However, you can avoid upgrading user schemas and the `sys` schema using `mysqld` followed by `mysql_upgrade` with the `--upgrade-system-tables` and `--skip-sys-schema` options.
  - As of MySQL 8.0.16: `mysqld --upgrade=MINIMAL`
- Perform step 1 as necessary, and force step 2:
  - Prior to MySQL 8.0.16: `mysqld` followed by `mysql_upgrade --force`
  - As of MySQL 8.0.16: `mysqld --upgrade=FORCE`

Prior to MySQL 8.0.16, certain `mysql_upgrade` options affect the actions it performs. The following table shows which server `--upgrade` option values to use as of MySQL 8.0.16 to achieve similar effects. (These are not necessarily exact equivalents because a given `--upgrade` option value may have additional effects.)

mysql_upgrade Option	Server Option
`--skip-sys-schema`	`--upgrade=NONE` or `--upgrade=MINIMAL`
`--upgrade-system-tables`	`--upgrade=NONE` or `--upgrade=MINIMAL`
`--force`	`--upgrade=FORCE`

Additional notes about what occurs during upgrade step 2:

- Step 2 installs the `sys` schema if it is not installed, and upgrades it to the current version otherwise. An error occurs if a `sys` schema exists but has no `version` view, on the assumption that its absence indicates a user-created schema:

  ```
  A sys schema exists with no sys.version view. If
  you have a user created sys schema, this must be renamed for the
  upgrade to succeed.
  ```

  To upgrade in this case, remove or rename the existing `sys` schema first. Then perform the upgrade procedure again. (It may be necessary to force step 2.)

  To prevent the `sys` schema check:

  - As of MySQL 8.0.16: Start the server with the `--upgrade=NONE` or `--upgrade=MINIMAL` option.
  - Prior to MySQL 8.0.16: Invoke `mysql_upgrade` with the `--skip-sys-schema` option.

- Step 2 processes all tables in all user schemas as necessary. Table checking might take a long time to complete. Each table is locked and therefore unavailable to other sessions while it is being processed. Check and repair operations can be time-consuming, particularly for large tables. Table checking uses the `FOR UPGRADE` option of the `CHECK TABLE` statement. For details about what this option entails, see Section 13.7.3.2, “CHECK TABLE Syntax”.

  To prevent table checking:

  - As of MySQL 8.0.16: Start the server with the `--upgrade=NONE` or `--upgrade=MINIMAL` option.
  - Prior to MySQL 8.0.16: Invoke `mysql_upgrade` with the `--upgrade-system-tables` option.

  To force table checking:

  - As of MySQL 8.0.16: Start the server with the `--upgrade=FORCE` option.
  - Prior to MySQL 8.0.16: Invoke `mysql_upgrade` with the `--force` option.

- Step 2 saves the MySQL version number in a file named `mysql_upgrade_info` in the data directory.

  To ignore the `mysql_upgrade_info` file and perform the check regardless:

  - As of MySQL 8.0.16: Start the server with the `--upgrade=FORCE` option.
  - Prior to MySQL 8.0.16: Invoke `mysql_upgrade` with the `--force` option.

  Note

  The `mysql_upgrade_info` file is deprecated and will be removed in a future MySQL version.

- Step 2 marks all checked and repaired tables with the current MySQL version number. This ensures that the next time upgrade checking occurs with the same version of the server, it can be determined whether there is any need to check or repair a given table again.
- Step 2 upgrades the system tables to ensure that they have the current structure. This is true whether the server or `mysql_upgrade` performs the step. With respect to the content of the help tables and time zone tables, `mysql_upgrade` does not load either type of table, whereas the server loads the help tables, but not the time zone tables. (That is, prior to MySQL 8.0.16, the server loads the help tables only at data directory initialization time. As of MySQL 8.0.16, it loads the help tables at initialization and upgrade time.) The procedure for loading time zone tables is platform dependent and requires decision making by the DBA, so it cannot be done automatically.

2.11.4 Changes in MySQL 8.0

Before upgrading to MySQL 8.0, review the changes described in this section to identify those that apply to your current MySQL installation and applications. Perform any recommended actions.

Changes marked as **Incompatible change** are incompatibilities with earlier versions of MySQL, and may require your attention *before upgrading*. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases. If an upgrade issue applicable to your installation involves an incompatibility, follow the instructions given in the description.

- Data Dictionary Changes
- caching_sha2_password as the Preferred Authentication Plugin

- Configuration Changes
- Server Changes
- InnoDB Changes
- SQL Changes

Data Dictionary Changes

MySQL Server 8.0 incorporates a global data dictionary containing information about database objects in transactional tables. In previous MySQL series, dictionary data was stored in metadata files and nontransactional system tables. As a result, the upgrade procedure requires that you verify the upgrade readiness of your installation by checking specific prerequisites. For more information, see Section 2.11.5, “Preparing Your Installation for Upgrade”. A data dictionary-enabled server entails some general operational differences; see Section 14.7, “Data Dictionary Usage Differences”.

caching_sha2_password as the Preferred Authentication Plugin

The `caching_sha2_password` and `sha256_password` authentication plugins provide more secure password encryption than the `mysql_native_password` plugin, and `caching_sha2_password` provides better performance than `sha256_password`. Due to these superior security and performance characteristics of `caching_sha2_password`, it is as of MySQL 8.0 the preferred authentication plugin, and is also the default authentication plugin rather than `mysql_native_password`. This change affects both the server and the `libmysqlclient` client library:

- For the server, the default value of the `default_authentication_plugin` system variable changes from `mysql_native_password` to `caching_sha2_password`.

  This change applies only to new accounts created after installing or upgrading to MySQL 8.0 or higher. For accounts already existing in an upgraded installation, their authentication plugin remains unchanged. Existing users who wish to switch to `caching_sha2_password` can do so using the `ALTER USER` statement:

  ```
  ALTER USER user
    IDENTIFIED WITH caching_sha2_password
    BY 'password';
  ```

- The `libmysqlclient` library treats `caching_sha2_password` as the default authentication plugin rather than `mysql_native_password`.

The following sections discuss the implications of the more prominent role of `caching_sha2_password`:

- caching_sha2_password Compatibility Issues and Solutions
- caching_sha2_password-Compatible Clients and Connectors
- caching_sha2_password and the root Administrative Account
- caching_sha2_password and Replication

caching_sha2_password Compatibility Issues and Solutions

Important

If your MySQL installation must serve pre-8.0 clients and you encounter compatibility issues after upgrading to MySQL 8.0 or higher, the simplest way to

address those issues and restore pre-8.0 compatibility is to reconfigure the server to revert to the previous default authentication plugin (`mysql_native_password`). For example, use these lines in the server option file:

```
[mysqld]
default_authentication_plugin=mysql_native_password
```

That setting enables pre-8.0 clients to connect to 8.0 servers until such time as the clients and connectors in use at your installation are upgraded to know about `caching_sha2_password`. However, the setting should be viewed as temporary, not as a long term or permanent solution, because it causes new accounts created with the setting in effect to forego the improved authentication security provided by `caching_sha2_password`.

The use of `caching_sha2_password` offers more secure password hashing than `mysql_native_password` (and consequent improved client connection authentication). However, it also has compatibility implications that may affect existing MySQL installations:

- Clients and connectors that have not been updated to know about `caching_sha2_password` may have trouble connecting to a MySQL 8.0 server configured with `caching_sha2_password` as the default authentication plugin, even to use accounts that do not authenticate with `caching_sha2_password`. This issue occurs because the server specifies the name of its default authentication plugin to clients. If a client or connector is based on a client/server protocol implementation that does not gracefully handle an unrecognized default authentication plugin, it may fail with an error such as one of these:

  ```
  Authentication plugin 'caching_sha2_password' is not supported
  ```

  ```
  Authentication plugin 'caching_sha2_password' cannot be loaded:
  dlopen(/usr/local/mysql/lib/plugin/caching_sha2_password.so, 2):
  image not found
  ```

  ```
  Warning: mysqli_connect(): The server requested authentication
  method unknown to the client [caching_sha2_password]
  ```

  For information about writing connectors to gracefully handle requests from the server for unknown default authentication plugins, see Authentication Plugin Connector-Writing Considerations.

- Clients that use an account that authenticates with `caching_sha2_password` must use either a secure connection (made using TCP using TLS/SSL credentials, a Unix socket file, or shared memory), or an unencrypted connection that supports password exchange using an RSA key pair. This security requirement does not apply to `mysql_native_passsword`, so the switch to `caching_sha2_password` may require additional configuration (see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”). However, client connections in MySQL 8.0 prefer use of TLS/SSL by default, so clients that already conform to that preference may need no additional configuration.

- Clients and connectors that have not been updated to know about `caching_sha2_password` *cannot* connect to accounts that authenticate with `caching_sha2_password` because they do not recognize this plugin as valid. (This is a particular instance of how client/server authentication plugin compatibility requirements apply, as discussed at Authentication Plugin Client/Server Compatibility.) To work around this issue, relink clients against `libmysqlclient` from MySQL 8.0 or higher, or obtain an updated connector that recognizes `caching_sha2_password`.

- Because `caching_sha2_password` is also now the default authentication plugin in the `libmysqlclient` client library, authentication requires an extra round trip in the client/server

protocol for connections from MySQL 8.0 clients to accounts that use `mysql_native_password` (the previous default authentication plugin), unless the client program is invoked with a `--default-auth=mysql_native_password` option.

The `libmysqlclient` client library for pre-8.0 MySQL versions is able to connect to MySQL 8.0 servers (except for accounts that authenticate with `caching_sha2_password`). That means pre-8.0 clients based on `libmysqlclient` should also be able to connect. Examples:

- Standard MySQL clients such as `mysql` and `mysqladmin` are `libmysqlclient`-based.
- The DBD::mysql driver for Perl DBI is `libmysqlclient`-based.
- MySQL Connector/Python has a C Extension module that is `libmysqlclient`-based. To use it, include the `use_pure=False` option at connect time.

When an existing MySQL 8.0 installation is upgraded to MySQL 8.0.4 or higher, some older `libmysqlclient`-based clients may “automatically” upgrade if they are dynamically linked, because they use the new client library installed by the upgrade. For example, if the DBD::mysql driver for Perl DBI uses dynamic linking, it can use the `libmysqlclient` in place after an upgrade to MySQL 8.0.4 or higher, with this result:

- Prior to the upgrade, DBI scripts that use DBD::mysql can connect to a MySQL 8.0 server, except for accounts that authenticate with `caching_sha2_password`.
- After the upgrade, the same scripts become able to use `caching_sha2_password` accounts as well.

However, the preceding results occur because `libmysqlclient` instances from MySQL 8.0 installations prior to 8.0.4 are binary compatible: They both use a shared library major version number of 21. For clients linked to `libmysqlclient` from MySQL 5.7 or older, they link to a shared library with a different version number that is not binary compatible. In this case, the client must be recompiled against `libmysqlclient` from 8.0.4 or higher for full compatibility with MySQL 8.0 servers and `caching_sha2_password` accounts.

MySQL Connector/J 5.1 through 8.0.8 is able to connect to MySQL 8.0 servers, except for accounts that authenticate with `caching_sha2_password`. (Connector/J 8.0.9 or higher is required to connect to `caching_sha2_password` accounts.)

Clients that use an implementation of the client/server protocol other than `libmysqlclient` may need to be upgraded to a newer version that understands the new authentication plugin. For example, in PHP, MySQL connectivity usually is based on `mysqlnd`, which currently does not know about `caching_sha2_password`. Until an updated version of `mysqlnd` is available, the way to enable PHP clients to connect to MySQL 8.0 is to reconfigure the server to revert to `mysql_native_password` as the default authentication plugin, as previously discussed.

If a client or connector supports an option to explicitly specify a default authentication plugin, use it to name a plugin other than `caching_sha2_password`. Examples:

- Some MySQL clients support a `--default-auth` option. (Standard MySQL clients such as `mysql` and `mysqladmin` support this option but can successfully connect to 8.0 servers without it. However, other clients may support a similar option. If so, it is worth trying it.)
- Programs that use the `libmysqlclient` C API can call the `mysql_options()` function with the `MYSQL_DEFAULT_AUTH` option.
- MySQL Connector/Python scripts that use the native Python implementation of the client/server protocol can specify the `auth_plugin` connection option. (Alternatively, use the Connector/Python C Extension, which is able to connect to MySQL 8.0 servers without the need for `auth_plugin`.)

caching_sha2_password-Compatible Clients and Connectors

If a client or connector is available that has been updated to know about `caching_sha2_password`, using it is the best way to ensure compatibility when connecting to a MySQL 8.0 server configured with `caching_sha2_password` as the default authentication plugin.

These clients and connectors have been upgraded to support `caching_sha2_password`:

- The `libmysqlclient` client library in MySQL 8.0 (8.0.4 or higher). Standard MySQL clients such as `mysql` and `mysqladmin` are `libmysqlclient`-based, so they are compatible as well.
- The `libmysqlclient` client library in MySQL 5.7 (5.7.23 or higher). Standard MySQL clients such as `mysql` and `mysqladmin` are `libmysqlclient`-based, so they are compatible as well.
- MySQL Connector/C++ 1.1.11 or higher or 8.0.7 or higher.
- MySQL Connector/J 8.0.9 or higher.
- MySQL Connector/NET 8.0.10 or higher (through the classic MySQL protocol).
- MySQL Connector/Node.js 8.0.9 or higher.
- PHP: the X DevAPI PHP extension (mysql_xdevapi) supports `caching_sha2_password`.

  PHP: the PDO_MySQL and ext/mysqli extensions do not support `caching_sha2_password`. In addition, when used with PHP versions before 7.1.16 and PHP 7.2 before 7.2.4, they fail to connect with `default_authentication_plugin=caching_sha2_password` even if `caching_sha2_password` is not used.

caching_sha2_password and the root Administrative Account

For upgrades to MySQL 8.0, the authentication plugin existing accounts remains unchanged, including the plugin for the `'root'@'localhost'` administrative account.

For new MySQL 8.0 installations, when you initialize the data directory (using the instructions at Section 2.10.1, “Initializing the Data Directory”), the `'root'@'localhost'` account is created, and that account uses `caching_sha2_password` by default. To connect to the server following data directory initialization, you must therefore use a client or connector that supports `caching_sha2_password`. If you can do this but prefer that the `root` account use `mysql_native_password` after installation, install MySQL and initialize the data directory as you normally would. Then connect to the server as `root` and use `ALTER USER` as follows to change the account authentication plugin and password:

```
ALTER USER 'root'@'localhost'
  IDENTIFIED WITH mysql_native_password
  BY 'password';
```

If the client or connector that you use does not yet support `caching_sha2_password`, you can use a modified data directory-initialization procedure that associates the `root` account with `mysql_native_password` as soon as the account is created. To do so, use either of these techniques:

- Supply a `--default-authentication-plugin=mysql_native_password` option along with `--initialize` or `--initialize-insecure`.
- Set `default_authentication_plugin` to `mysql_native_password` in an option file, and name that option file using a `--defaults-file` option along with `--initialize` or `--initialize-insecure`. (In this case, if you continue to use that option file for subsequent server startups, new

accounts will be created with `mysql_native_password` rather than `caching_sha2_password` unless you remove the `default_authentication_plugin` setting from the option file.)

caching_sha2_password and Replication

In replication scenarios for which all servers have been upgraded to MySQL 8.0.4 or higher, slave/replica connections to master/primary servers can use accounts that authenticate with `caching_sha2_password`. For such connections, the same requirement applies as for other clients that use accounts that authenticate with `caching_sha2_password`: Use a secure connection or RSA-based password exchange.

To connect to a `caching_sha2_password` account for master/slave replication:

- Use any of the following `CHANGE MASTER TO` options:

```
MASTER_SSL = 1
GET_MASTER_PUBLIC_KEY = 1
MASTER_PUBLIC_KEY_PATH='path to RSA public key file'
```

- Alternatively, you can use the RSA public key-related options if the required keys are supplied at server startup.

To connect to a `caching_sha2_password` account for Group Replication:

- For MySQL built using OpenSSL, set any of the following system variables:

```
SET GLOBAL group_replication_recovery_use_ssl = ON;
SET GLOBAL group_replication_recovery_get_public_key = 1;
SET GLOBAL group_replication_recovery_public_key_path = 'path to RSA public key file';
```

- Alternatively, you can use the RSA public key-related options if the required keys are supplied at server startup.

Configuration Changes

- **Incompatible change**: A MySQL storage engine is now responsible for providing its own partitioning handler, and the MySQL server no longer provides generic partitioning support. `InnoDB` and `NDB` are the only storage engines that provide a native partitioning handler that is supported in MySQL 8.0. A partitioned table using any other storage engine must be altered—either to convert it to `InnoDB` or `NDB`, or to remove its partitioning—*before* upgrading the server, else it cannot be used afterwards.

  For information about converting `MyISAM` tables to `InnoDB`, see Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”.

  A table creation statement that would result in a partitioned table using a storage engine without such support fails with an error (`ER_CHECK_NOT_IMPLEMENTED`) in MySQL 8.0. If you import databases from a dump file created in MySQL 5.7 (or earlier) using `mysqldump` into a MySQL 8.0 server, you must make sure that any statements creating partitioned tables do not also specify an unsupported storage engine, either by removing any references to partitioning, or by specifying the storage engine as `InnoDB` or allowing it to be set as `InnoDB` by default.

  Note

  The procedure given at Section 2.11.5, “Preparing Your Installation for Upgrade”, describes how to identify partitioned tables that must be altered before upgrading to MySQL 8.0.

See Section 23.6.2, “Partitioning Limitations Relating to Storage Engines”, for further information.

- **Incompatible change**: Several server error codes are not used and have been removed (for a list, see Features Removed in MySQL 8.0). Applications that test specifically for any of them should be updated.
- **Important change**: The default character set has changed from `latin1` to `utf8mb4`. These system variables are affected:
  - The default value of the `character_set_server` and `character_set_database` system variables has changed from `latin1` to `utf8mb4`.
  - The default value of the `collation_server` and `collation_database` system variables has changed from `latin1_swedish_ci` to `utf8mb4_0900_ai_ci`.

  As a result, the default character set and collation for new objects differ from previously unless an explicit character set and collation are specified. This includes databases and objects within them, such as tables, views, and stored programs. Assuming that the previous defaults were used, one way to preserve them is to start the server with these lines in the `my.cnf` file:

  ```
  [mysqld]
  character_set_server=latin1
  collation_server=latin1_swedish_ci
  ```

  In a replicated setting, when upgrading from MySQL 5.7 to 8.0, it is advisable to change the default character set back to the character set used in MySQL 5.7 before upgrading. After the upgrade is completed, the default character set can be changed to `utf8mb4`.
- **Incompatible change**: As of MySQL 8.0.11, it is prohibited to start the server with a `lower_case_table_names` setting that is different from the setting used when the server was initialized. The restriction is necessary because collations used by various data dictionary table fields are based on the `lower_case_table_names` setting that was defined when the server was initialized, and restarting the server with a different setting would introduce inconsistencies with respect to how identifiers are ordered and compared.

Server Changes

- In MySQL 8.0.11, several deprecated features related to account management have been removed, such as use of the `GRANT` statement to modify nonprivilege characteristics of user accounts, the `NO_AUTO_CREATE_USER` SQL mode, the `PASSWORD()` function, and the `old_passwords` system variable.

  Replication from MySQL 5.7 to 8.0 of statements that refer to these removed features can cause replication failure. Applications that use any of the removed features should be revised to avoid them and use alternatives when possible, as described in Features Removed in MySQL 8.0.

  To avoid a startup failure on MySQL 8.0, remove any instance of `NO_AUTO_CREATE_USER` from `sql_mode` system variable settings in MySQL option files.

  Loading a dump file that includes the `NO_AUTO_CREATE_USER` SQL mode in stored program definitions into a MySQL 8.0 server causes a failure. As of MySQL 5.7.24 and MySQL 8.0.13, `mysqldump` removes `NO_AUTO_CREATE_USER` from stored program definitions. Dump files created with an earlier version of `mysqldump` must be modified manually to remove instances of `NO_AUTO_CREATE_USER`.
- In MySQL 8.0.11, these deprecated compatibility SQL modes were removed: `DB2`, `MAXDB`, `MSSQL`, `MYSQL323`, `MYSQL40`, `ORACLE`, `POSTGRESQL`, `NO_FIELD_OPTIONS`, `NO_KEY_OPTIONS`,

NO_TABLE_OPTIONS. They can no longer be assigned to the sql_mode system variable or used as permitted values for the mysqldump --compatible option.

Removal of MAXDB means that the TIMESTAMP data type for CREATE TABLE or ALTER TABLE is no longer treated as DATETIME.

Replication from MySQL 5.7 to 8.0 of statements that refer to the removed SQL modes can cause replication failure. This includes replication of CREATE statements for stored programs (stored procedures and functions, triggers, and events) that are executed while the current sql_mode value includes any of the removed modes. Applications that use any of the removed modes should be revised to avoid them.

- As of MySQL 8.0.3, spatial data types permit an SRID attribute, to explicitly indicate the spatial reference system (SRS) for values stored in the column. See Section 11.5.1, "Spatial Data Types".

  A spatial column with an explicit SRID attribute is SRID-restricted: The column takes only values with that ID, and SPATIAL indexes on the column become subject to use by the optimizer. The optimizer ignores SPATIAL indexes on spatial columns with no SRID attribute. See Section 8.3.3, "SPATIAL Index Optimization". If you want the optimizer to consider SPATIAL indexes on spatial columns that are not SRID-restricted, each such column should be modified:

  - Verify that all values within the column have the same SRID. To determine the SRIDs contained in a geometry column *col_name*, use the following query:

    ```
    SELECT DISTINCT ST_SRID(col_name) FROM tbl_name;
    ```

    If the query returns more than one row, the column contains a mix of SRIDs. In that case, modify its contents so all values have the same SRID.

  - Redefine the column to have an explicit SRID attribute.

  - Recreate the SPATIAL index.

- Several spatial functions were removed in MySQL 8.0.0 due to a spatial function namespace change that implemented an ST_ prefix for functions that perform an exact operation, or an MBR prefix for functions that perform an operation based on minimum bounding rectangles. The use of removed spatial functions in generated column definitions could cause an upgrade failure. Before upgrading, run mysqlcheck --check-upgrade for removed spatial functions and replace any that you find with their ST_ or MBR named replacements. For a list of removed spatial functions, refer to Features Removed in MySQL 8.0.

- The BACKUP_ADMIN privilege is automatically granted to users with the RELOAD privilege when performing an in-place upgrade to MySQL 8.0.3 or higher.

- From MySQL 8.0.13, because of differences between row-based or mixed replication mode and statement-based replication mode in the way that temporary tables are handled, there are new restrictions on switching the binary logging format at runtime.

  - SET @@SESSION.binlog_format cannot be used if the session has any open temporary tables.

  - SET @@global.binlog_format and SET @@persist.binlog_format cannot be used if any replication channel has any open temporary tables. SET @@persist_only.binlog_format is allowed if replication channels have open temporary tables, because unlike PERSIST, PERSIST_ONLY does not modify the runtime global system variable value.

- `SET @@global.binlog_format` and `SET @@persist.binlog_format` cannot be used if any replication channel applier is running. This is because the change only takes effect on a replication channel when its applier is restarted, at which time the replication channel might have open temporary tables. This behavior is more restrictive than before. `SET @@persist_only.binlog_format` is allowed if any replication channel applier is running.

InnoDB Changes

- `INFORMATION_SCHEMA` views based on `InnoDB` system tables were replaced by internal system views on data dictionary tables. Affected `InnoDB` `INFORMATION_SCHEMA` views were renamed:

  Table 2.15 Renamed InnoDB Information Schema Views

Old Name	New Name
`INNODB_SYS_COLUMNS`	`INNODB_COLUMNS`
`INNODB_SYS_DATAFILES`	`INNODB_DATAFILES`
`INNODB_SYS_FIELDS`	`INNODB_FIELDS`
`INNODB_SYS_FOREIGN`	`INNODB_FOREIGN`
`INNODB_SYS_FOREIGN_COLS`	`INNODB_FOREIGN_COLS`
`INNODB_SYS_INDEXES`	`INNODB_INDEXES`
`INNODB_SYS_TABLES`	`INNODB_TABLES`
`INNODB_SYS_TABLESPACES`	`INNODB_TABLESPACES`
`INNODB_SYS_TABLESTATS`	`INNODB_TABLESTATS`
`INNODB_SYS_VIRTUAL`	`INNODB_VIRTUAL`

  After upgrading to MySQL 8.0.3 or higher, update any scripts that reference previous `InnoDB` `INFORMATION_SCHEMA` view names.

- The zlib library version bundled with MySQL was raised from version 1.2.3 to version 1.2.11.

  The zlib `compressBound()` function in zlib 1.2.11 returns a slightly higher estimate of the buffer size required to compress a given length of bytes than it did in zlib version 1.2.3. The `compressBound()` function is called by `InnoDB` functions that determine the maximum row size permitted when creating compressed `InnoDB` tables or inserting rows into compressed `InnoDB` tables. As a result, `CREATE TABLE ... ROW_FORMAT=COMPRESSED` or `INSERT` operations with row sizes very close to the maximum row size that were successful in earlier releases could now fail.

  If you have compressed `InnoDB` tables with large rows, it is recommended that you test compressed table `CREATE TABLE` statements on a MySQL 8.0 test instance prior to upgrading.

- With the introduction of the `--innodb-directories` feature, the location of file-per-table and general tablespace files created with an absolute path or in a location outside of the data directory should be added to the `innodb_directories` argument value. Otherwise, `InnoDB` is not able to locate these files during recovery. To view tablespace file locations, query the `INFORMATION_SCHEMA.FILES` table:

  ```
  SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES \G
  ```

- Undo logs can no longer reside in the system tablespace. In MySQL 8.0, undo logs reside in two undo tablespaces by default. For more information, see Section 15.6.3.4, “Undo Tablespaces”.

When upgrading from MySQL 5.7 to MySQL 8.0, any undo tablespaces that exist in the MySQL 5.7 instance are removed and replaced by two new default undo tablespaces. Default undo tablespaces are created in the location defined by the `innodb_undo_directory` variable. If the `innodb_undo_directory` variable is undefined, undo tablespaces are created in the data directory. Upgrade from MySQL 5.7 to MySQL 8.0 requires a slow shutdown which ensures that undo tablespaces in the MySQL 5.7 instance are empty, permitting them to be removed safely.

When upgrading to MySQL 8.0.14 or later from an earlier MySQL 8.0 release, undo tablespaces that exist in the pre-upgrade instance as a result of an `innodb_undo_tablespaces` setting greater than 2 are treated as user-defined undo tablespaces, which can be deactivated and dropped using `ALTER UNDO TABLESPACE` and `DROP UNDO TABLESPACE` syntax, respectively, after upgrading. Upgrade within the MySQL 8.0 release series may not always require a slow shutdown which means that existing undo tablespaces could contain undo logs. Therefore, existing undo tablespaces are not removed by the upgrade process.

- **Incompatible change**: As of MySQL 8.0.17, the `CREATE TABLESPACE ... ADD DATAFILE` clause does not permit circular directory references. For example, the circular directory reference (`/../`) in the following statement is not permitted:

  ```
  CREATE TABLESPACE ts1 ADD DATAFILE ts1.ibd 'any_directory/../ts1.ibd';
  ```

  An exception to the restriction exists on Linux, where a circular directory reference is permitted if the preceding directory is a symbolic link. For example, the data file path in the example above is permitted if *`any_directory`* is a symbolic link. (It is still permitted for data file paths to begin with '`../`'.)

  To avoid upgrade issues, remove any circular directory references from tablespace data file paths before upgrading to MySQL 8.0.17 or higher. To inspect tablespace paths, query the `INFORMATION_SCHEMA.INNODB_DATAFILES` table.

- Due to a regression introduced in MySQL 8.0.14, in-place upgrade on a case sensitive file system from MySQL 5.7 or a MySQL 8.0 release prior to MySQL 8.0.14 to MySQL 8.0.16 failed for instances with partitioned tables and `lower_case_table_names=1`. The failure was caused by a case mismatch issue related to partitioned table file names. The fix that introduced the regression was reverted, which permits upgrades to MySQL 8.0.17 from MySQL 5.7 or MySQL 8.0 releases prior to MySQL 8.0.14 to function as normal. However, the regression is still present in the MySQL 8.0.14, 8.0.15, and 8.0.16 releases.

  In-place upgrade on a case sensitive file system from MySQL 8.0.14, 8.0.15, or 8.0.16 to MySQL 8.0.17 fails with the following error when starting the server after upgrading binaries or packages to MySQL 8.0.17 if partitioned tables are present and `lower_case_table_names=1`:

  ```
  Upgrading from server version version_number with
  partitioned tables and lower_case_table_names == 1 on a case sensitive file
  system may cause issues, and is therefore prohibited. To upgrade anyway, restart
  the new server version with the command line option 'upgrade=FORCE'. When
  upgrade is completed, please execute 'RENAME TABLE part_table_name
  TO new_table_name; RENAME TABLE new_table_name
  TO part_table_name;' for each of the partitioned tables.
  Please see the documentation for further information.
  ```

  If you encounter this error when upgrading to MySQL 8.0.17, perform the following workaround:

  1. Restart the server with `--upgrade=force` to force the upgrade operation to proceed.

  2. Identify partitioned table file names with lowercase suffixes:

```
mysql> SELECT FILE_NAME FROM INFORMATION_SCHEMA.FILES WHERE FILE_NAME LIKE '%#p#%' OR FILE_NAME LIKE '%#s
```

3. For each file identified, rename the associated table using a temporary name, then rename the table back to its original name.

```
mysql> RENAME TABLE table_name TO temporary_table_name;
mysql> RENAME TABLE temporary_table_name TO table_name;
```

4. Verify that there are no partitioned table file names with lowercase suffixes (an empty result set should be returned).

```
mysql> SELECT FILE_NAME FROM INFORMATION_SCHEMA.FILES WHERE FILE_NAME LIKE '%#p#%' OR FILE_NAME LIKE '%#s
Empty set (0.00 sec)
```

5. Run `ANALYZE TABLE` on each renamed table to update the optimizer statistics in the `mysql.innodb_index_stats` and `mysql.innodb_table_stats` tables.

Because of the regression still present in the MySQL 8.0.14, 8.0.15, and 8.0.16 releases, importing partitioned tables from MySQL 8.0.14, 8.0.15, or 8.0.16 to MySQL 8.0.17 is not supported on case sensitive file systems where `lower_case_table_names=1`. Attempting to do so results in a "Tablespace is missing for table" error.

SQL Changes

- **Incompatible change**: As of MySQL 8.0.13, the deprecated `ASC` or `DESC` qualifiers for `GROUP BY` clauses have been removed. Queries that previously relied on `GROUP BY` sorting may produce results that differ from previous MySQL versions. To produce a given sort order, provide an `ORDER BY` clause.

  Queries and stored program definitions from MySQL 8.0.12 or lower that use `ASC` or `DESC` qualifiers for `GROUP BY` clauses should be amended. Otherwise, upgrading to MySQL 8.0.13 or higher may fail, as may replicating to MySQL 8.0.13 or higher slave servers.

- Some keywords may be reserved in MySQL 8.0 that were not reserved in MySQL 5.7. See Section 9.3, "Keywords and Reserved Words". This can cause words previously used as identifiers to become illegal. To fix affected statements, use identifier quoting. See Section 9.2, "Schema Object Names".

- After upgrading, it is recommended that you test optimizer hints specified in application code to ensure that the hints are still required to achieve the desired optimization strategy. Optimizer enhancements can sometimes render certain optimizer hints unnecessary. In some cases, an unnecessary optimizer hint may even be counterproductive.

- **Incompatible change**: In MySQL 5.7, specifying a `FOREIGN KEY` definition for an `InnoDB` table without a `CONSTRAINT symbol` clause, or specifying the `CONSTRAINT` keyword without a `symbol`, causes `InnoDB` to use a generated constraint name. That behavior changed in MySQL 8.0, with `InnoDB` using the `FOREIGN KEY index_name` value instead of a generated name. Because constraint names must be unique per schema (database), the change caused errors due to foreign key index names that were not unique per schema. To avoid such errors, the new constraint naming behavior has been reverted in MySQL 8.0.16, and `InnoDB` once again uses a generated constraint name.

  For consistency with `InnoDB`, `NDB` releases based on MySQL 8.0.16 or higher use a generated constraint name if the `CONSTRAINT symbol` clause is not specified, or the `CONSTRAINT` keyword is specified without a `symbol`. `NDB` releases based on MySQL 5.7 and earlier MySQL 8.0 releases used the `FOREIGN KEY index_name` value.

The changes described above may introduce incompatibilities for applications that depend on the previous foreign key constraint naming behavior.

2.11.5 Preparing Your Installation for Upgrade

Before upgrading to the latest MySQL 8.0 release, ensure the upgrade readiness of your current MySQL 5.7 or MySQL 8.0 server instance by performing the preliminary checks described below. The upgrade process may fail otherwise.

The same checks and others can be performed using the MySQL Shell upgrade checker utility. For more information, see Upgrade Checker Utility.

Preliminary checks:

1. The following issues must not be present:

   - There must be no tables that use obsolete data types or functions.

     In-place upgrade to MySQL 8.0 is not supported if tables contain old temporal columns in pre-5.6.4 format (`TIME`, `DATETIME`, and `TIMESTAMP` columns without support for fractional seconds precision). If your tables still use the old temporal column format, upgrade them before attempting an in-place upgrade to MySQL 8.0. For more information, see Server Changes.

   - There must be no orphan `.frm` files.

   - Triggers must not have a missing or empty definer or an invalid creation context (indicated by the `character_set_client`, `collation_connection`, `Database Collation` attributes displayed by `SHOW TRIGGERS` or the `INFORMATION_SCHEMA TRIGGERS` table).

   To check for these issues, execute this command:

   ```
   mysqlcheck -u root -p --all-databases --check-upgrade
   ```

   If `mysqlcheck` reports any errors, correct the issues.

2. There must be no partitioned tables that use a storage engine that does not have native partitioning support. To identify such tables, execute this query:

   ```
   SELECT TABLE_SCHEMA, TABLE_NAME
   FROM INFORMATION_SCHEMA.TABLES
   WHERE ENGINE NOT IN ('innodb', 'ndbcluster')
   AND CREATE_OPTIONS LIKE '%partitioned%';
   ```

   Any table reported by the query must be altered to use `InnoDB` or be made nonpartitioned. To change a table storage engine to `InnoDB`, execute this statement:

   ```
   ALTER TABLE table_name ENGINE = INNODB;
   ```

   For information about converting `MyISAM` tables to `InnoDB`, see Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”.

   To make a partitioned table nonpartitioned, execute this statement:

   ```
   ALTER TABLE table_name REMOVE PARTITIONING;
   ```

3. Some keywords may be reserved in MySQL 8.0 that were not reserved previously. See Section 9.3, “Keywords and Reserved Words”. This can cause words previously used as identifiers to become illegal. To fix affected statements, use identifier quoting. See Section 9.2, “Schema Object Names”.

4. There must be no tables in the MySQL 5.7 `mysql` system database that have the same name as a table used by the MySQL 8.0 data dictionary. To identify tables with those names, execute this query:

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE LOWER(TABLE_SCHEMA) = 'mysql'
and LOWER(TABLE_NAME) IN
(
'catalogs',
'character_sets',
'check_constraints',
'collations',
'column_statistics',
'column_type_elements',
'columns',
'dd_properties',
'events',
'foreign_key_column_usage',
'foreign_keys',
'index_column_usage',
'index_partitions',
'index_stats',
'indexes',
'parameter_type_elements',
'parameters',
'resource_groups',
'routines',
'schemata',
'st_spatial_reference_systems',
'table_partition_values',
'table_partitions',
'table_stats',
'tables',
'tablespace_files',
'tablespaces',
'triggers',
'view_routine_usage',
'view_table_usage'
);
```

   Any tables reported by the query must be renamed (use `RENAME TABLE`). This may also entail changes to applications that use the affected tables.

5. There must be no tables that have foreign key constraint names longer than 64 characters. To identify tables with too-long constraint names, execute this query:

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME IN
  (SELECT LEFT(SUBSTR(ID,INSTR(ID,'/')+1),
              INSTR(SUBSTR(ID,INSTR(ID,'/')+1),'_ibfk_')-1)
   FROM INFORMATION_SCHEMA.INNODB_SYS_FOREIGN
   WHERE LENGTH(SUBSTR(ID,INSTR(ID,'/')+1))>64);
```

   Any tables reported by the query must be altered to have constraint names no longer than 64 characters (use `ALTER TABLE`).

6. The must be no obsolete SQL modes defined in your `sql_mode` system variable setting. Attempting to use an obsolete SQL mode will cause a startup failure on MySQL 8.0. Applications that use obsolete SQL modes should also be revised to avoid them. For information about SQL modes removed in MySQL 8.0, see Server Changes.

7. There must be no tables or stored procedures with individual `ENUM` or `SET` column elements that exceed 255 characters or 1020 bytes in length. Prior to MySQL 8.0, the maximum combined length of `ENUM` or `SET` column elements was 64K. In MySQL 8.0, the maximum character length of an individual `ENUM` or `SET` column element is 255 characters, and the maximum byte length is 1020 bytes. (The 1020 byte limit supports multitibyte character sets). Before upgrading to MySQL 8.0, modify any `ENUM` or `SET` column elements that exceed the new limits. Failing to do so causes the upgrade to fail with an error.

8. Before upgrading to MySQL 8.0.13 or higher, there must be no table partitions that reside in shared `InnoDB` tablespaces, which include the system tablespace and general tablespaces. Identify table partitions in shared tablespaces by querying `INFORMATION_SCHEMA`:

   If upgrading from MySQL 5.7, run this query:

   ```
   SELECT DISTINCT NAME, SPACE, SPACE_TYPE FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES
     WHERE NAME LIKE '%#P#%' AND SPACE_TYPE NOT LIKE 'Single';
   ```

   If upgrading from an earlier MySQL 8.0 release, run this query:

   ```
   SELECT DISTINCT NAME, SPACE, SPACE_TYPE FROM INFORMATION_SCHEMA.INNODB_TABLES
     WHERE NAME LIKE '%#P#%' AND SPACE_TYPE NOT LIKE 'Single';
   ```

   Move table partitions from shared tablespaces to file-per-table tablespaces using `ALTER TABLE ... REORGANIZE PARTITION`:

   ```
   ALTER TABLE table_name REORGANIZE PARTITION partition_name
     INTO (partition_definition TABLESPACE=innodb_file_per_table);
   ```

9. There must be no queries and stored program definitions from MySQL 8.0.12 or lower that use `ASC` or `DESC` qualifiers for `GROUP BY` clauses. Otherwise, upgrading to MySQL 8.0.13 or higher may fail, as may replicating to MySQL 8.0.13 or higher slave servers. For additional details, see SQL Changes.

10. Your MySQL 5.7 installation must not use features that are not supported by MySQL 8.0. Any changes here are necessarily installation specific, but the following example illustrates the kind of thing to look for:

    Some server startup options and system variables have been removed in MySQL 8.0. See Features Removed in MySQL 8.0, and Section 1.5, “Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 8.0”. If you use any of these, an upgrade requires configuration changes.

    Example: Because the data dictionary provides information about database objects, the server no longer checks directory names in the data directory to find databases. Consequently, the `--ignore-db-dir` option is extraneous and has been removed. To handle this, remove any instances of `--ignore-db-dir` from your startup configuration. In addition, remove or move the named data directory subdirectories before upgrading to MySQL 8.0. (Alternatively, let the 8.0 server add those directories to the data dictionary as databases, then remove each of those databases using `DROP DATABASE`.)

2.11.6 Upgrading MySQL Binary or Package-based Installations on Unix/Linux

This section describes how to upgrade MySQL binary and package-based installations on Unix/Linux. In-place and logical upgrade methods are described.

- In-Place Upgrade
- Logical Upgrade
- MySQL Cluster Upgrade

In-Place Upgrade

An in-place upgrade involves shutting down the old MySQL server, replacing the old MySQL binaries or packages with the new ones, restarting MySQL on the existing data directory, and upgrading any remaining parts of the existing installation that require upgrading. For details about what may need upgrading, see Section 2.11.3, “What the MySQL Upgrade Process Upgrades”.

Note

If you are upgrading an installation originally produced by installing multiple RPM packages, upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.

For some Linux platforms, MySQL installation from RPM or Debian packages includes systemd support for managing MySQL server startup and shutdown. On these platforms, `mysqld_safe` is not installed. In such cases, use systemd for server startup and shutdown instead of the methods used in the following instructions. See Section 2.5.9, “Managing MySQL Server with systemd”.

For upgrades to MySQL Cluster installations, see also MySQL Cluster Upgrade.

To perform an in-place upgrade:

1. Review the information in Section 2.11.1, “Before You Begin”.
2. Ensure the upgrade readiness of your installation by completing the preliminary checks in Section 2.11.5, “Preparing Your Installation for Upgrade”.
3. If you use XA transactions with `InnoDB`, run `XA RECOVER` before upgrading to check for uncommitted XA transactions. If results are returned, either commit or rollback the XA transactions by issuing an `XA COMMIT` or `XA ROLLBACK` statement.
4. If there are encrypted `InnoDB` tablespaces, rotate the keyring master key by executing this statement:

   ```
   ALTER INSTANCE ROTATE INNODB MASTER KEY;
   ```

5. If you normally run your MySQL server configured with `innodb_fast_shutdown` set to `2` (cold shutdown), configure it to perform a fast or slow shutdown by executing either of these statements:

   ```
   SET GLOBAL innodb_fast_shutdown = 1; -- fast shutdown
   SET GLOBAL innodb_fast_shutdown = 0; -- slow shutdown
   ```

   With a fast or slow shutdown, `InnoDB` leaves its undo logs and data files in a state that can be dealt with in case of file format differences between releases.

6. Shut down the old MySQL server. For example:

```
mysqladmin -u root -p shutdown
```

7. Upgrade the MySQL binaries or packages. If upgrading a binary installation, unpack the new MySQL binary distribution package. See Obtain and Unpack the Distribution. For package-based installations, install the new packages.

8. Start the MySQL 8.0 server, using the existing data directory. For example:

   ```
   mysqld_safe --user=mysql --datadir=/path/to/existing-datadir &
   ```

   If there are encrypted `InnoDB` tablespaces, use the `--early-plugin-load` option to load the keyring plugin.

   When you start the MySQL 8.0 server, it automatically detects whether data dictionary tables are present. If not, the server creates them in the data directory, populates them with metadata, and then proceeds with its normal startup sequence. During this process, the server upgrades metadata for all database objects, including databases, tablespaces, system and user tables, views, and stored programs (stored procedures and functions, triggers, and Event Scheduler events). The server also removes files that previously were used for metadata storage. For example, after upgrading from MySQL 5.7 to MySQL 8.0, you will notice that tables no longer have `.frm` files.

   If this step fails, the server reverts all changes to the data directory. In this case, you should remove all redo log files, start your MySQL 5.7 server on the same data directory, and fix the cause of any errors. Then perform another slow shutdown of the 5.7 server and start the MySQL 8.0 server to try again.

9. In the previous step, the server upgrades the data dictionary as necessary. Now it is necessary to perform any remaining upgrade operations:

   - As of MySQL 8.0.16, the server will already have done so in the previous step: It makes any changes required in the `mysql` system schema between MySQL 5.7 and MySQL 8.0, so that you can take advantage of new privileges or capabilities. It also brings the Performance Schema, `INFORMATION_SCHEMA`, and `sys` schema up to date for MySQL 8.0, and examines all user schemas for incompatibilities with the current version of MySQL.

   - Prior to MySQL 8.0.16, the server upgrades only the data dictionary in the previous step. After the MySQL 8.0 server starts successfully, execute `mysql_upgrade` to perform the remaining upgrade tasks:

     ```
     mysql_upgrade -u root -p
     ```

     Then shut down and restart the MySQL server to ensure that any changes made to the system tables take effect. For example:

     ```
     mysqladmin -u root -p shutdown
     mysqld_safe --user=mysql --datadir=/path/to/existing-datadir &
     ```

     The first time you start the MySQL 8.0 server (in an earlier step), you may notice messages in the error log regarding nonupgraded tables. If `mysql_upgrade` has been run successfully, there should be no such messages the second time you start the server.

Note

The upgrade process does not upgrade the contents of the time zone tables. For upgrade instructions, see Section 5.1.13, “MySQL Server Time Zone Support”.

If the upgrade process uses `mysql_upgrade` (that is, prior to MySQL 8.0.16), the process does not upgrade the contents of the help tables, either. For upgrade instructions in that case, see Section 5.1.14, “Server-Side Help Support”.

Logical Upgrade

A logical upgrade involves exporting SQL from the old MySQL instance using a backup or export utility such as `mysqldump` or `mysqlpump`, installing the new MySQL server, and applying the SQL to your new MySQL instance. For details about what may need upgrading, see Section 2.11.3, “What the MySQL Upgrade Process Upgrades”.

Note

For some Linux platforms, MySQL installation from RPM or Debian packages includes systemd support for managing MySQL server startup and shutdown. On these platforms, `mysqld_safe` is not installed. In such cases, use systemd for server startup and shutdown instead of the methods used in the following instructions. See Section 2.5.9, “Managing MySQL Server with systemd”.

Warning

Applying SQL extracted from a previous MySQL release to a new MySQL release may result in errors due to incompatibilities introduced by new, changed, deprecated, or removed features and capabilities. Consequently, SQL extracted from a previous MySQL release may require modification to enable a logical upgrade.

To identify incompatibilities before upgrading to the latest MySQL 8.0 release, perform the steps described in Section 2.11.5, “Preparing Your Installation for Upgrade”.

To perform a logical upgrade:

1. Review the information in Section 2.11.1, “Before You Begin”.
2. Export your existing data from the previous MySQL installation:

```
mysqldump -u root -p
  --add-drop-table --routines --events
  --all-databases --force > data-for-upgrade.sql
```

Note

Use the `--routines` and `--events` options with `mysqldump` (as shown above) if your databases include stored programs. The `--all-databases` option includes all databases in the dump, including the `mysql` database that holds the system tables.

Important

If you have tables that contain generated columns, use the `mysqldump` utility provided with MySQL 5.7.9 or higher to create your dump files. The `mysqldump` utility provided in earlier releases uses incorrect syntax for generated column definitions (Bug #20769542). You can use the `INFORMATION_SCHEMA.COLUMNS` table to identify tables with generated columns.

3. Shut down the old MySQL server. For example:

```
mysqladmin -u root -p shutdown
```

4. Install MySQL 8.0. For installation instructions, see Chapter 2, *Installing and Upgrading MySQL*.

5. Initialize a new data directory, as described in Section 2.10.1, “Initializing the Data Directory”. For example:

```
mysqld --initialize --datadir=/path/to/8.0-datadir
```

   Copy the temporary `'root'@'localhost'` password displayed to your screen or written to your error log for later use.

6. Start the MySQL 8.0 server, using the new data directory. For example:

```
mysqld_safe --user=mysql --datadir=/path/to/8.0-datadir &
```

7. Reset the `root` password:

```
shell> mysql -u root -p
Enter password: ****  <- enter temporary root password
```

```
mysql> ALTER USER USER() IDENTIFIED BY 'your new password';
```

8. Load the previously created dump file into the new MySQL server. For example:

```
mysql -u root -p --force < data-for-upgrade.sql
```

Note

It is not recommended to load a dump file when GTIDs are enabled on the server (`gtid_mode=ON`), if your dump file includes system tables. `mysqldump` issues DML instructions for the system tables which use the non-transactional MyISAM storage engine, and this combination is not permitted when GTIDs are enabled. Also be aware that loading a dump file from a server with GTIDs enabled, into another server with GTIDs enabled, causes different transaction identifiers to be generated.

9. Perform any remaining upgrade operations:

   - In MySQL 8.0.16 and higher, shut down the server, then restart it with the `--upgrade=FORCE` option to perform the remaining upgrade tasks:

```
mysqladmin -u root -p shutdown
mysqld_safe --user=mysql --datadir=/path/to/8.0-datadir --upgrade=FORCE &
```

     Upon restart with `--upgrade=FORCE`, the server makes any changes required in the `mysql` system schema between MySQL 5.7 and MySQL 8.0, so that you can take advantage of new privileges or capabilities. It also brings the Performance Schema, `INFORMATION_SCHEMA`, and `sys` schema up to date for MySQL 8.0, and examines all user schemas for incompatibilities with the current version of MySQL.

   - Prior to MySQL 8.0.16, execute `mysql_upgrade` to perform the remaining upgrade tasks:

```
mysql_upgrade -u root -p
```

Then shut down and restart the MySQL server to ensure that any changes made to the system tables take effect. For example:

```
mysqladmin -u root -p shutdown
mysqld_safe --user=mysql --datadir=/path/to/8.0-datadir &
```

Note

The upgrade process does not upgrade the contents of the time zone tables. For upgrade instructions, see Section 5.1.13, “MySQL Server Time Zone Support”.

If the upgrade process uses `mysql_upgrade` (that is, prior to MySQL 8.0.16), the process does not upgrade the contents of the help tables, either. For upgrade instructions in that case, see Section 5.1.14, “Server-Side Help Support”.

Note

Loading a dump file that contains a MySQL 5.7 `mysql` schema re-creates two tables that are no longer used: `event` and `proc`. (The corresponding MySQL 8.0 tables are `events` and `routines`, both of which are data dictionary tables and are protected.) After you are satisfied that the upgrade was successful, you can remove the `event` and `proc` tables by executing these SQL statements:

```
DROP TABLE mysql.event;
DROP TABLE mysql.proc;
```

MySQL Cluster Upgrade

The information in this section is an adjunct to the in-place upgrade procedure described in In-Place Upgrade, for use if you are upgrading MySQL Cluster.

As of MySQL 8.0.16, a MySQL Cluster upgrade can be performed as a regular rolling upgrade, following the usual three ordered steps:

1. Upgrade MGM nodes.
2. Upgrade data nodes one at a time.
3. Upgrade API nodes one at a time (including MySQL servers).

The way to upgrade each of the nodes remains almost the same as prior to MySQL 8.0.16 because there is a separation between upgrading the data dictionary and upgrading the system tables. There are two steps to upgrading each individual `mysqld`:

1. Import the data dictionary.

   Start the new server with the `--upgrade=MINIMAL` option to upgrade the data dictionary but not the system tables. This is essentially the same as the pre-MySQL 8.0.16 action of starting the server and not invoking `mysql_upgrade`.

   The MySQL server must be connected to `NDB` for this phase to complete. If any `NDB` or `NDBINFO` tables exist, and the server cannot connect to the cluster, it exits with an error message:

```
Failed to Populate DD tables.
```

2. Upgrade the system tables.

   Prior to MySQL 8.0.16, the DBA invokes the `mysql_upgrade` client to upgrade the system tables. As of MySQL 8.0.16, the server performs this action: To upgrade the system tables, restart each individual `mysqld` without the `--upgrade=MINIMAL` option.

2.11.7 Upgrading MySQL with the MySQL Yum Repository

For supported Yum-based platforms (see Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”, for a list), you can perform an in-place upgrade for MySQL (that is, replacing the old version and then running the new version using the old data files) with the MySQL Yum repository.

Notes

- Before performing any update to MySQL, follow carefully the instructions in Section 2.11, “Upgrading MySQL”. Among other instructions discussed there, it is especially important to back up your database before the update.
- The following instructions assume you have installed MySQL with the MySQL Yum repository or with an RPM package directly downloaded from MySQL Developer Zone's MySQL Download page; if that is not the case, following the instructions in Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository.

Selecting a Target Series

1. By default, the MySQL Yum repository updates MySQL to the latest version in the release series you have chosen during installation (see Selecting a Release Series for details), which means, for example, a 5.7.x installation will *not* be updated to a 8.0.x release automatically. To update to another release series, you need to first disable the subrepository for the series that has been selected (by default, or by yourself) and enable the subrepository for your target series. To do that, see the general instructions given in Selecting a Release Series. For upgrading from MySQL 5.7 to 8.0, perform the *reverse* of the steps illustrated in Selecting a Release Series, disabling the subrepository for the MySQL 5.7 series and enabling that for the MySQL 8.0 series.

   As a general rule, to upgrade from one release series to another, go to the next series rather than skipping a series. For example, if you are currently running MySQL 5.6 and wish to upgrade to 8.0, upgrade to MySQL 5.7 first before upgrading to 8.0.

Important

For important information about upgrading from MySQL 5.7 to 8.0, see Upgrading from MySQL 5.7 to 8.0.

Upgrading MySQL

2. Upgrade MySQL and its components by the following command, for platforms that are not dnf-enabled:

```
sudo yum update mysql-server
```

For platforms that are dnf-enabled:

```
sudo dnf upgrade mysql-server
```

Alternatively, you can update MySQL by telling Yum to update everything on your system, which might take considerably more time. For platforms that are not dnf-enabled:

```
sudo yum update
```

For platforms that are dnf-enabled:

```
sudo dnf upgrade
```

Restarting MySQL

The MySQL server always restarts after an update by Yum. Prior to MySQL 8.0.16, run `mysql_upgrade` after the server restarts to check and possibly resolve any incompatibilities between the old data and the upgraded software. `mysql_upgrade` also performs other functions; for details, see Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”. As of MySQL 8.0.16, this step is not required, as the server performs all tasks previously handled by `mysql_upgrade`.

You can also update only a specific component. Use the following command to list all the installed packages for the MySQL components (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
sudo yum list installed | grep "^mysql"
```

After identifying the package name of the component of your choice, update the package with the following command, replacing *`package-name`* with the name of the package. For platforms that are not dnf-enabled:

```
sudo yum update package-name
```

For dnf-enabled platforms:

```
sudo dnf upgrade package-name
```

Upgrading the Shared Client Libraries

After updating MySQL using the Yum repository, applications compiled with older versions of the shared client libraries should continue to work.

If you recompile applications and dynamically link them with the updated libraries: As typical with new versions of shared libraries where there are differences or additions in symbol versioning between the newer and older libraries (for example, between the newer, standard 8.0 shared client libraries and some older—prior or variant—versions of the shared libraries shipped natively by the Linux distributions' software repositories, or from some other sources), any applications compiled using the updated, newer shared libraries will require those updated libraries on systems where the applications are deployed. And, as expected, if those libraries are not in place, the applications requiring the shared libraries will fail. So, be sure to deploy the packages for the shared libraries from MySQL on those systems. To do this, add the MySQL Yum repository to the systems (see Adding the MySQL Yum Repository) and install the latest shared libraries using the instructions given in Installing Additional MySQL Products and Components with Yum.

2.11.8 Upgrading MySQL with the MySQL APT Repository

On Debian and Ubuntu platforms, to perform an in-place upgrade of MySQL and its components, use the MySQL APT repository. See Upgrading MySQL with the MySQL APT Repository in A Quick Guide to Using the MySQL APT Repository.

2.11.9 Upgrading MySQL with the MySQL SLES Repository

On the SUSE Linux Enterprise Server (SLES) platform, to perform an in-place upgrade of MySQL and its components, use the MySQL SLES repository. See Upgrading MySQL with the MySQL SLES Repository in A Quick Guide to Using the MySQL SLES Repository.

2.11.10 Upgrading MySQL on Windows

There are two approaches for upgrading MySQL on Windows:

- Using MySQL Installer
- Using the Windows ZIP archive distribution

The approach you select depends on how the existing installation was performed. Before proceeding, review Section 2.11, “Upgrading MySQL” for additional information on upgrading MySQL that is not specific to Windows.

Note

Whichever approach you choose, always back up your current MySQL installation before performing an upgrade. See Section 7.2, “Database Backup Methods”.

Upgrades between non-GA releases (or from a non-GA release to a GA release) are not supported. Significant development changes take place in non-GA releases and you may encounter compatibility issues or problems starting the server.

Note

MySQL Installer does not support upgrades between *Community* releases and *Commercial* releases. If you require this type of upgrade, perform it using the ZIP archive approach.

Upgrading MySQL with MySQL Installer

Performing an upgrade with MySQL Installer is the best approach when the current server installation was performed with it and the upgrade is within the current release series. MySQL Installer does not support upgrades between release series, such as from 5.7 to 8.0, and it does not provide an upgrade indicator to prompt you to upgrade. For instructions on upgrading between release series, see Upgrading MySQL Using the Windows ZIP Distribution.

To perform an upgrade using MySQL Installer:

1. Start MySQL Installer.
2. From the dashboard, click **Catalog** to download the latest changes to the catalog. The installed server can be upgraded only if the dashboard displays an arrow next to the version number of the server.
3. Click **Upgrade**. All products that have a newer version now appear in a list.

Note

MySQL Installer deselects the server upgrade option for milestone releases (Pre-Release) in the same release series. In addition, it displays a warning to

indicate that the upgrade is not supported, identifies the risks of continuing, and provides a summary of the steps to perform an upgrade manually. You can reselect server upgrade and proceed at your own risk.

4. Deselect all but the MySQL server product, unless you intend to upgrade other products at this time, and click **Next**.

5. Click **Execute** to start the download. When the download finishes, click **Next** to begin the upgrade operation.

   Upgrades to MySQL 8.0.16 and higher may show an option to skip the upgrade check and process for system tables. For more information about this option, see Important server upgrade conditions.

6. Configure the server.

Upgrading MySQL Using the Windows ZIP Distribution

To perform an upgrade using the Windows ZIP archive distribution:

1. Download the latest Windows ZIP Archive distribution of MySQL from https://dev.mysql.com/downloads/.

2. If the server is running, stop it. If the server is installed as a service, stop the service with the following command from the command prompt:

```
C:\> SC STOP mysqld_service_name
```

   Alternatively, use `NET STOP mysqld_service_name`.

   If you are not running the MySQL server as a service, use `mysqladmin` to stop it. For example, before upgrading from MySQL 5.7 to 8.0, use `mysqladmin` from MySQL 5.7 as follows:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.7\bin\mysqladmin" -u root shutdown
```

Note

If the MySQL `root` user account has a password, invoke `mysqladmin` with the `-p` option and enter the password when prompted.

3. Extract the ZIP archive. You may either overwrite your existing MySQL installation (usually located at `C:\mysql`), or install it into a different directory, such as `C:\mysql8`. Overwriting the existing installation is recommended.

4. Restart the server. For example, use the `SC START mysqld_service_name` or `NET START mysqld_service_name` command if you run MySQL as a service, or invoke `mysqld` directly otherwise.

5. Prior to MySQL 8.0.16, run `mysql_upgrade` as Administrator to check your tables, attempt to repair them if necessary, and update your grant tables if they have changed so that you can take advantage of any new capabilities. See Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”. As of MySQL 8.0.16, this step is not required, as the server performs all tasks previously handled by `mysql_upgrade`.

6. If you encounter errors, see Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”.

2.11.11 Upgrading a Docker Installation of MySQL

To upgrade a Docker installation of MySQL, refer to Upgrading a MySQL Server Container.

2.11.12 Upgrade Troubleshooting

- A schema mismatch in a MySQL 5.7 instance between the `.frm` file of a table and the `InnoDB` data dictionary can cause an upgrade to MySQL 8.0 to fail. Such mismatches may be due to `.frm` file corruption. To address this issue, dump and restore affected tables before attempting the upgrade again.
- If problems occur, such as that the new `mysqld` server does not start, verify that you do not have an old `my.cnf` file from your previous installation. You can check this with the `--print-defaults` option (for example, `mysqld --print-defaults`). If this command displays anything other than the program name, you have an active `my.cnf` file that affects server or client operation.
- If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, you probably have used old header or library files when compiling your programs. In this case, check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example, from `libmysqlclient.so.20` to `libmysqlclient.so.21`).
- If you have created a user-defined function (UDF) with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name. The same is true if the new version of MySQL implements a built-in function with the same name as an existing stored function. See Section 9.2.5, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

2.11.13 Rebuilding or Repairing Tables or Indexes

This section describes how to rebuild or repair tables or indexes, which may be necessitated by:

- Changes to how MySQL handles data types or character sets. For example, an error in a collation might have been corrected, necessitating a table rebuild to update the indexes for character columns that use the collation.
- Required table repairs or upgrades reported by `CHECK TABLE`, `mysqlcheck`, or `mysql_upgrade`.

Methods for rebuilding a table include:

- Dump and Reload Method
- ALTER TABLE Method
- REPAIR TABLE Method

Dump and Reload Method

If you are rebuilding tables because a different version of MySQL will not handle them after a binary (in-place) upgrade or downgrade, you must use the dump-and-reload method. Dump the tables *before* upgrading or downgrading using your original version of MySQL. Then reload the tables *after* upgrading or downgrading.

If you use the dump-and-reload method of rebuilding tables only for the purpose of rebuilding indexes, you can perform the dump either before or after upgrading or downgrading. Reloading still must be done afterward.

If you need to rebuild an `InnoDB` table because a `CHECK TABLE` operation indicates that a table upgrade is required, use `mysqldump` to create a dump file and `mysql` to reload the file. If the `CHECK TABLE` operation indicates that there is a corruption or causes `InnoDB` to fail, refer to Section 15.20.2, “Forcing InnoDB Recovery” for information about using the `innodb_force_recovery` option to restart `InnoDB`. To understand the type of problem that `CHECK TABLE` may be encountering, refer to the `InnoDB` notes in Section 13.7.3.2, “CHECK TABLE Syntax”.

To rebuild a table by dumping and reloading it, use `mysqldump` to create a dump file and `mysql` to reload the file:

```
mysqldump db_name t1 > dump.sql
mysql db_name < dump.sql
```

To rebuild all the tables in a single database, specify the database name without any following table name:

```
mysqldump db_name > dump.sql
mysql db_name < dump.sql
```

To rebuild all tables in all databases, use the `--all-databases` option:

```
mysqldump --all-databases > dump.sql
mysql < dump.sql
```

ALTER TABLE Method

To rebuild a table with `ALTER TABLE`, use a “null” alteration; that is, an `ALTER TABLE` statement that “changes” the table to use the storage engine that it already has. For example, if `t1` is an `InnoDB` table, use this statement:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

If you are not sure which storage engine to specify in the `ALTER TABLE` statement, use `SHOW CREATE TABLE` to display the table definition.

REPAIR TABLE Method

The `REPAIR TABLE` method is only applicable to `MyISAM`, `ARCHIVE`, and `CSV` tables.

You can use `REPAIR TABLE` if the table checking operation indicates that there is a corruption or that an upgrade is required. For example, to repair a `MyISAM` table, use this statement:

```
REPAIR TABLE t1;
```

`mysqlcheck --repair` provides command-line access to the `REPAIR TABLE` statement. This can be a more convenient means of repairing tables because you can use the `--databases` or `--all-databases` option to repair all tables in specific databases or all databases, respectively:

```
mysqlcheck --repair --databases db_name ...
mysqlcheck --repair --all-databases
```

2.11.14 Copying MySQL Databases to Another Machine

In cases where you need to transfer databases between different architectures, you can use `mysqldump` to create a file containing SQL statements. You can then transfer the file to the other machine and feed it as input to the `mysql` client.

Use `mysqldump --help` to see what options are available.

Note

If GTIDs are in use on the server where you create the dump (`gtid_mode=ON`), by default, `mysqldump` includes the contents of the `gtid_executed` set in the dump to transfer these to the new machine. The results of this can vary depending on the MySQL Server versions involved. Check the description for `mysqldump's --set-gtid-purged` option to find what happens with the versions you are using, and how to change the behavior if the outcome of the default behavior is not suitable for your situation.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
mysqladmin -h 'other_hostname' create db_name
mysqldump db_name | mysql -h 'other_hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use these commands:

```
mysqladmin create db_name
mysqldump -h 'other_hostname' --compress db_name | mysql db_name
```

You can also store the dump in a file, transfer the file to the target machine, and then load the file into the database there. For example, you can dump a database to a compressed file on the source machine like this:

```
mysqldump --quick db_name | gzip > db_name.gz
```

Transfer the file containing the database contents to the target machine and run these commands there:

```
mysqladmin create db_name
gunzip < db_name.gz | mysql db_name
```

You can also use `mysqldump` and `mysqlimport` to transfer the database. For large tables, this is much faster than simply using `mysqldump`. In the following commands, *`DUMPDIR`* represents the full path name of the directory you use to store the output from `mysqldump`.

First, create the directory for the output files and dump the database:

```
mkdir DUMPDIR
mysqldump --tab=DUMPDIR db_name
```

Then transfer the files in the *`DUMPDIR`* directory to some corresponding directory on the target machine and load the files into MySQL there:

```
mysqladmin create db_name           # create database
cat DUMPDIR/*.sql | mysql db_name   # create tables in database
mysqlimport db_name DUMPDIR/*.txt   # load data into tables
```

Do not forget to copy the `mysql` database because that is where the grant tables are stored. You might have to run commands as the MySQL `root` user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute `mysqladmin flush-privileges` so that the server reloads the grant table information.

2.12 Downgrading MySQL

Downgrade from MySQL 8.0 to MySQL 5.7, or from a MySQL 8.0 release to a previous MySQL 8.0 release, is not supported. The only supported alternative is to restore a backup taken *before* upgrading. It is therefore imperative that you backup your data before starting the upgrade process.

2.13 Perl Installation Notes

The Perl `DBI` module provides a generic interface for database access. You can write a `DBI` script that works with many different database engines without change. To use `DBI`, you must install the `DBI` module, as well as a DataBase Driver (DBD) module for each type of database server you want to access. For MySQL, this driver is the `DBD::mysql` module.

Note

Perl support is not included with MySQL distributions. You can obtain the necessary modules from http://search.cpan.org for Unix, or by using the ActiveState `ppm` program on Windows. The following sections describe how to do this.

The `DBI`/`DBD` interface requires Perl 5.6.0, and 5.6.1 or later is preferred. DBI *does not work* if you have an older version of Perl. You should use `DBD::mysql` 4.009 or higher. Although earlier versions are available, they do not support the full functionality of MySQL 8.0.

2.13.1 Installing Perl on Unix

MySQL Perl support requires that you have installed MySQL client programming support (libraries and header files). Most installation methods install the necessary files. If you install MySQL from RPM files on Linux, be sure to install the developer RPM as well. The client programs are in the client RPM, but client programming support is in the developer RPM.

The files you need for Perl support can be obtained from the CPAN (Comprehensive Perl Archive Network) at http://search.cpan.org.

The easiest way to install Perl modules on Unix is to use the `CPAN` module. For example:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

The `DBD::mysql` installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default user name and password. (The default user name is your login name on Unix, and `ODBC` on Windows. The default password is “no password.”) If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use `force install DBD::mysql` to ignore the failed tests.

`DBI` requires the `Data::Dumper` module. It may be installed; if not, you should install it before installing `DBI`.

It is also possible to download the module distributions in the form of compressed `tar` archives and build the modules manually. For example, to unpack and build a DBI distribution, use a procedure such as this:

1. Unpack the distribution into the current directory:

```
shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
```

This command creates a directory named `DBI-VERSION`.

2. Change location into the top-level directory of the unpacked distribution:

```
shell> cd DBI-VERSION
```

3. Build the distribution and compile everything:

```
shell> perl Makefile.PL
shell> make
shell> make test
shell> make install
```

The `make test` command is important because it verifies that the module is working. Note that when you run that command during the `DBD::mysql` installation to exercise the interface code, the MySQL server must be running or the test fails.

It is a good idea to rebuild and reinstall the `DBD::mysql` distribution whenever you install a new release of MySQL. This ensures that the latest versions of the MySQL client libraries are installed correctly.

If you do not have access rights to install Perl modules in the system directory or if you want to install local Perl modules, the following reference may be useful: http://learn.perl.org/faq/perlfaq8.html#How-do-I-keep-my-own-module-library-directory-

2.13.2 Installing ActiveState Perl on Windows

On Windows, you should do the following to install the MySQL `DBD` module with ActiveState Perl:

1. Get ActiveState Perl from http://www.activestate.com/Products/ActivePerl/ and install it.
2. Open a console window.
3. If necessary, set the `HTTP_proxy` variable. For example, you might try a setting like this:

```
C:\> set HTTP_proxy=my.proxy.com:3128
```

4. Start the PPM program:

```
C:\> C:\perl\bin\ppm.pl
```

5. If you have not previously done so, install `DBI`:

```
ppm> install DBI
```

6. If this succeeds, run the following command:

```
ppm> install DBD-mysql
```

This procedure should work with ActiveState Perl 5.6 or higher.

If you cannot get the procedure to work, you should install the ODBC driver instead and connect to the MySQL server through ODBC:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.13.3 Problems Using the Perl DBI/DBD Interface

If Perl reports that it cannot find the `../mysql/mysql.so` module, the problem is probably that Perl cannot locate the `libmysqlclient.so` shared library. You should be able to fix this problem by one of the following methods:

- Copy `libmysqlclient.so` to the directory where your other shared libraries are located (probably `/usr/lib` or `/lib`).
- Modify the `-L` options used to compile `DBD::mysql` to reflect the actual location of `libmysqlclient.so`.
- On Linux, you can add the path name of the directory where `libmysqlclient.so` is located to the `/etc/ld.so.conf` file.
- Add the path name of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable. Some systems use `LD_LIBRARY_PATH` instead.

Note that you may also need to modify the `-L` options if there are other libraries that the linker fails to find. For example, if the linker cannot find `libc` because it is in `/lib` and the link command specifies `-L/usr/lib`, change the `-L` option to `-L/lib` or add `-L/lib` to the existing link command.

If you get the following errors from `DBD::mysql`, you are probably using `gcc` (or using an old binary compiled with `gcc`):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add `-L/usr/lib/gcc-lib/... -lgcc` to the link command when the `mysql.so` library gets built (check the output from `make` for `mysql.so` when you compile the Perl client). The `-L` option should specify the path name of the directory where `libgcc.a` is located on your system.

Another cause of this problem may be that Perl and MySQL are not both compiled with `gcc`. In this case, you can solve the mismatch by compiling both with `gcc`.

Chapter 3 Tutorial

Table of Contents

3.1 Connecting to and Disconnecting from the Server 277
3.2 Entering Queries 278
3.3 Creating and Using a Database 282
3.3.1 Creating and Selecting a Database 283
3.3.2 Creating a Table 284
3.3.3 Loading Data into a Table 285
3.3.4 Retrieving Information from a Table 286
3.4 Getting Information About Databases and Tables 300
3.5 Using mysql in Batch Mode 301
3.6 Examples of Common Queries 303
3.6.1 The Maximum Value for a Column 304
3.6.2 The Row Holding the Maximum of a Certain Column 304
3.6.3 Maximum of Column per Group 304
3.6.4 The Rows Holding the Group-wise Maximum of a Certain Column 305
3.6.5 Using User-Defined Variables 306
3.6.6 Using Foreign Keys 306
3.6.7 Searching on Two Keys 308
3.6.8 Calculating Visits Per Day 308
3.6.9 Using AUTO_INCREMENT 309
3.7 Using MySQL with Apache 311

This chapter provides a tutorial introduction to MySQL by showing how to use the `mysql` client program to create and use a simple database. `mysql` (sometimes referred to as the “terminal monitor” or just “monitor”) is an interactive program that enables you to connect to a MySQL server, run queries, and view the results. `mysql` may also be used in batch mode: you place your queries in a file beforehand, then tell `mysql` to execute the contents of the file. Both ways of using `mysql` are covered here.

To see a list of options provided by `mysql`, invoke it with the `--help` option:

```
shell> mysql --help
```

This chapter assumes that `mysql` is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If *you* are the administrator, you need to consult the relevant portions of this manual, such as Chapter 5, *MySQL Server Administration*.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an existing database, you may want to skip the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily omitted. Consult the relevant sections of the manual for more information on the topics covered here.

3.1 Connecting to and Disconnecting from the Server

To connect to the server, you will usually need to provide a MySQL user name when you invoke `mysql` and, most likely, a password. If the server runs on a machine other than the one where you log in, you will

also need to specify a host name. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: ********
```

`host` and *`user`* represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The `********` represents your password; enter it when `mysql` displays the `Enter password:` prompt.

If that works, you should see some introductory information followed by a `mysql>` prompt:

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 8.0.19-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The `mysql>` prompt tells you that `mysql` is ready for you to enter SQL statements.

If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
shell> mysql -u user -p
```

If, when you attempt to log in, you get an error message such as `ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)`, it means that the MySQL server daemon (Unix) or service (Windows) is not running. Consult the administrator or see the section of Chapter 2, *Installing and Upgrading MySQL* that is appropriate to your operating system.

For help with other problems often encountered when trying to log in, see Section B.4.2, “Common Errors When Using MySQL Programs”.

Some MySQL installations permit users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking `mysql` without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing `QUIT` (or `\q`) at the `mysql>` prompt:

```
mysql> QUIT
Bye
```

On Unix, you can also disconnect by pressing Control+D.

Most examples in the following sections assume that you are connected to the server. They indicate this by the `mysql>` prompt.

3.2 Entering Queries

Make sure that you are connected to the server, as discussed in the previous section. Doing so does not in itself select any database to work with, but that is okay. At this point, it is more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering queries, using several queries you can try out to familiarize yourself with how `mysql` works.

Here is a simple query that asks the server to tell you its version number and the current date. Type it in as shown here following the `mysql>` prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+-----------+--------------+
| VERSION() | CURRENT_DATE |
+-----------+--------------+
| 5.8.0-m17 | 2015-12-21   |
+-----------+--------------+
1 row in set (0.02 sec)
mysql>
```

This query illustrates several things about `mysql`:

- A query normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. `QUIT`, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a query, `mysql` sends it to the server for execution and displays the results, then prints another `mysql>` prompt to indicate that it is ready for another query.
- `mysql` displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), `mysql` labels the column using the expression itself.
- `mysql` shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the “rows in set” line is sometimes not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here is another query. It demonstrates that you can use `mysql` as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------------+---------+
| SIN(PI()/4)      | (4+1)*5 |
+------------------+---------+
| 0.70710678118655 |      25 |
+------------------+---------+
1 row in set (0.02 sec)
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
```

```
+-----------+
| VERSION() |
+-----------+
| 8.0.13    |
+-----------+
1 row in set (0.00 sec)

+---------------------+
| NOW()               |
+---------------------+
| 2018-08-24 00:56:40 |
+---------------------+
1 row in set (0.00 sec)
```

A query need not be given all on a single line, so lengthy queries that require several lines are not a problem. `mysql` determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, `mysql` accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here is a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2018-08-24   |
+---------------+--------------+
```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how `mysql` indicates that it has not yet seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you can always be aware of what `mysql` is waiting for.

If you decide you do not want to execute a query that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that `mysql` is ready for a new query.

The following table shows each of the prompts you may see and summarizes what they mean about the state that `mysql` is in.

Prompt	Meaning
`mysql>`	Ready for new query
`->`	Waiting for next line of multiple-line query
`'>`	Waiting for next line, waiting for completion of a string that began with a single quote (`'`)
`">`	Waiting for next line, waiting for completion of a string that began with a double quote (`"`)

Prompt	Meaning
`` `> ``	Waiting for next line, waiting for completion of an identifier that began with a backtick (`` ` ``)
`/*>`	Waiting for next line, waiting for completion of a comment that began with `/*`

Multiple-line statements commonly occur by accident when you intend to issue a query on a single line, but forget the terminating semicolon. In this case, `mysql` waits for more input:

```
mysql> SELECT USER()
    ->
```

If this happens to you (you think you've entered a statement but the only response is a `->` prompt), most likely `mysql` is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and `mysql` executes it:

```
mysql> SELECT USER()
    -> ;
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
```

The `'>` and `">` prompts occur during string collection (another way of saying that MySQL is waiting for completion of a string). In MySQL, you can write strings surrounded by either `'` or `"` characters (for example, `'hello'` or `"goodbye"`), and `mysql` lets you enter strings that span multiple lines. When you see a `'>` or `">` prompt, it means that you have entered a line containing a string that begins with a `'` or `"` quote character, but have not yet entered the matching quote that terminates the string. This often indicates that you have inadvertently left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '>
```

If you enter this `SELECT` statement, then press **Enter** and wait for the result, nothing happens. Instead of wondering why this query takes so long, notice the clue provided by the `'>` prompt. It tells you that `mysql` expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `'Smith` is missing the second single quotation mark.)

At this point, what do you do? The simplest thing is to cancel the query. However, you cannot just type `\c` in this case, because `mysql` interprets it as part of the string that it is collecting. Instead, enter the closing quote character (so `mysql` knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '> '\c
mysql>
```

The prompt changes back to `mysql>`, indicating that `mysql` is ready for a new query.

The `` `> `` prompt is similar to the `'>` and `">` prompts, but indicates that you have begun but not completed a backtick-quoted identifier.

It is important to know what the `'>`, `">`, and `` `> `` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type appear to be ignored by `mysql`—including a line containing

`QUIT`. This can be quite confusing, especially if you do not know that you need to supply the terminating quote before you can cancel the current query.

Note

Multiline statements from this point on are written without the secondary (`->` or other) prompts, to make it easier to copy and paste the statements to try for yourself.

3.3 Creating and Using a Database

Once you know how to enter SQL statements, you are ready to access a database.

Suppose that you have several pets in your home (your menagerie) and you would like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to perform the following operations:

- Create a database
- Create a table
- Load data into the table
- Retrieve data from the table in various ways
- Use multiple tables

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be obtained from the MySQL website. It is available in both compressed `tar` file and Zip formats at https://dev.mysql.com/doc/.

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

The `mysql` database describes user access privileges. The `test` database often is available as a workspace for users to try things out.

The list of databases displayed by the statement may be different on your machine; `SHOW DATABASES` does not show databases that you have no privileges for if you do not have the `SHOW DATABASES` privilege. See Section 13.7.7.14, “SHOW DATABASES Syntax”.

If the `test` database exists, try to access it:

```
mysql> USE test
Database changed
```

USE, like QUIT, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The USE statement is special in another way, too: it must be given on a single line.

You can use the test database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose that you want to call yours menagerie. The administrator needs to execute a statement like this:

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

where your_mysql_name is the MySQL user name assigned to you and your_client_host is the host from which you connect to the server.

3.3.1 Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case-sensitive (unlike SQL keywords), so you must always refer to your database as menagerie, not as Menagerie, MENAGERIE, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query. However, for a variety of reasons, the recommended best practice is always to use the same lettercase that was used when the database was created.)

Note

If you get an error such as `ERROR 1044 (42000): Access denied for user 'micah'@'localhost' to database 'menagerie'` when attempting to create a database, this means that your user account does not have the necessary privileges to do so. Discuss this with the administrator or see Section 6.2, “Access Control and Account Management”.

Creating a database does not select it for use; you must do that explicitly. To make menagerie the current database, use this statement:

```
mysql> USE menagerie
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a mysql session. You can do this by issuing a USE statement as shown in the example. Alternatively, you can select the database on the command line when you invoke mysql. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Important

menagerie in the command just shown is **not** your password. If you want to supply your password on the command line after the -p option, you must do so with no intervening space (for example, as -p*password*, not as -p *password*). However,

putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

Note

You can see at any time which database is currently selected using `SELECT DATABASE()`.

3.3.2 Creating a Table

Creating the database is the easy part, but at this point it is empty, as `SHOW TABLES` tells you:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the `pet` table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it is not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it is better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you need to send out birthday greetings in the current week or month, for that computer-assisted personal touch.)
- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the `pet` table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
       species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the `name`, `owner`, and `species` columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be `20`. You can normally pick any length from `1` to `65535`, whatever seems most reasonable to you. If you make a poor choice and it turns out later that you need a longer field, MySQL provides an `ALTER TABLE` statement.

Several types of values can be chosen to represent sex in animal records, such as `'m'` and `'f'`, or perhaps `'male'` and `'female'`. It is simplest to use the single characters `'m'` and `'f'`.

The use of the `DATE` data type for the `birth` and `death` columns is a fairly obvious choice.

Once you have created a table, `SHOW TABLES` should produce some output:

```
mysql> SHOW TABLES;
+--------------------+
| Tables in menagerie |
+--------------------+
| pet                |
+--------------------+
```

To verify that your table was created the way you expected, use a `DESCRIBE` statement:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

You can use `DESCRIBE` any time, for example, if you forget the names of the columns in your table or what types they have.

For more information about MySQL data types, see Chapter 11, *Data Types*.

3.3.3 Loading Data into a Table

After creating your table, you need to populate it. The `LOAD DATA` and `INSERT` statements are useful for this.

Suppose that your pet records can be described as shown here. (Observe that MySQL expects dates in `'YYYY-MM-DD'` format; this may be different from what you are used to.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt` containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use `NULL` values. To

represent these in your text file, use `\N` (backslash, capital-N). For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

```
Whistler        Gwen    bird    \N      1997-12-09      \N
```

To load the text file `pet.txt` into the `pet` table, use this statement:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

If you created the file on Windows with an editor that uses `\r\n` as a line terminator, you should use this statement instead:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
       LINES TERMINATED BY '\r\n';
```

(On an Apple machine running OS X, you would likely want to use `LINES TERMINATED BY '\r'`.)

You can specify the column value separator and end of line marker explicitly in the `LOAD DATA` statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file `pet.txt` properly.

If the statement fails, it is likely that your MySQL installation does not have local file capability enabled by default. See Section 6.1.6, “Security Issues with LOAD DATA LOCAL”, for information on how to change this.

When you want to add new records one at a time, the `INSERT` statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the `CREATE TABLE` statement. Suppose that Diane gets a new hamster named “Puffball.” You could add a new record using an `INSERT` statement like this:

```
mysql> INSERT INTO pet
       VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

String and date values are specified as quoted strings here. Also, with `INSERT`, you can insert `NULL` directly to represent a missing value. You do not use `\N` like you do with `LOAD DATA`.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several `INSERT` statements rather than a single `LOAD DATA` statement.

3.3.4 Retrieving Information from a Table

The `SELECT` statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

`what_to_select` indicates what you want to see. This can be a list of columns, or `*` to indicate “all columns.” *`which_table`* indicates the table from which you want to retrieve data. The `WHERE` clause is optional. If it is present, *`conditions_to_satisfy`* specifies one or more conditions that rows must satisfy to qualify for retrieval.

3.3.4.1 Selecting All Data

The simplest form of `SELECT` retrieves everything from a table:

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

This form of `SELECT` is useful if you want to review your entire table, for example, after you've just loaded it with your initial data set. For example, you may happen to think that the birth date for Bowser doesn't seem quite right. Consulting your original pedigree papers, you find that the correct birth year should be 1989, not 1979.

There are at least two ways to fix this:

- Edit the file `pet.txt` to correct the error, then empty the table and reload it using `DELETE` and `LOAD DATA`:

  ```
  mysql> DELETE FROM pet;
  mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
  ```

  However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an `UPDATE` statement:

  ```
  mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
  ```

  The `UPDATE` changes only the record in question and does not require you to reload the table.

3.3.4.2 Selecting Particular Rows

As shown in the preceding section, it is easy to retrieve an entire table. Just omit the `WHERE` clause from the `SELECT` statement. But typically you don't want to see the entire table, particularly when it becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

The output confirms that the year is correctly recorded as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as `'bowser'`, `'BOWSER'`, and so forth. The query result is the same.

You can specify conditions on any column, not just `name`. For example, if you want to know which animals were born during or after 1998, test the `birth` column:

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
```

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The preceding query uses the `AND` logical operator. There is also an `OR` operator:

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

`AND` and `OR` may be intermixed, although `AND` has higher precedence than `OR`. If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
       OR (species = 'dog' AND sex = 'f');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

3.3.4.3 Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas. For example, if you want to know when your animals were born, select the `name` and `birth` columns:

```
mysql> SELECT name, birth FROM pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
```

```
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

Notice that the query simply retrieves the `owner` column from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet
       WHERE species = 'dog' OR species = 'cat';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

3.3.4.4 Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. It is often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

On character type columns, sorting—like all other comparison operations—is normally performed in a case-insensitive fashion. This means that the order is undefined for columns that are identical except for their case. You can force a case-sensitive sort for a column by using `BINARY` like so: `ORDER BY BINARY col_name`.

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `DESC` keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

You can sort on multiple columns, and you can sort different columns in different directions. For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet
       ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Claws    | cat     | 1994-03-17 |
| Fluffy   | cat     | 1993-02-04 |
| Fang     | dog     | 1990-08-27 |
| Bowser   | dog     | 1989-08-31 |
| Buffy    | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+----------+---------+------------+
```

The `DESC` keyword applies only to the column name immediately preceding it (`birth`); it does not affect the `species` column sort order.

3.3.4.5 Date Calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, use the `TIMESTAMPDIFF()` function. Its arguments are the unit in which you want the result expressed, and the two dates for which to take the difference. The following query shows, for each pet, the birth date, the current date, and the age in years. An *alias* (`age`) is used to make the final output column label more meaningful.

```
mysql> SELECT name, birth, CURDATE(),
       TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
       FROM pet;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
+----------+------------+------------+------+
```

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an `ORDER BY name` clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
       TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
       FROM pet ORDER BY name;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
+----------+------------+------------+------+
```

To sort the output by `age` rather than `name`, just use a different `ORDER BY` clause:

```
mysql> SELECT name, birth, CURDATE(),
       TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
       FROM pet ORDER BY age;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
```

```
+----------+------------+------------+------+
```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the `death` value is `NULL`. Then, for those with non-`NULL` values, compute the difference between the `death` and `birth` values:

```
mysql> SELECT name, birth, death,
       TIMESTAMPDIFF(YEAR,birth,death) AS age
       FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 |    5 |
+--------+------------+------------+------+
```

The query uses `death IS NOT NULL` rather than `death <> NULL` because `NULL` is a special value that cannot be compared using the usual comparison operators. This is discussed later. See Section 3.3.4.6, “Working with NULL Values”.

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the `birth` column. MySQL provides several functions for extracting parts of dates, such as `YEAR()`, `MONTH()`, and `DAYOFMONTH()`. `MONTH()` is the appropriate function here. To see how it works, run a simple query that displays the value of both `birth` and `MONTH(birth)`:

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| Fluffy   | 1993-02-04 |            2 |
| Claws    | 1994-03-17 |            3 |
| Buffy    | 1989-05-13 |            5 |
| Fang     | 1990-08-27 |            8 |
| Bowser   | 1989-08-31 |            8 |
| Chirpy   | 1998-09-11 |            9 |
| Whistler | 1997-12-09 |           12 |
| Slim     | 1996-04-29 |            4 |
| Puffball | 1999-03-30 |            3 |
+----------+------------+--------------+
```

Finding animals with birthdays in the upcoming month is also simple. Suppose that the current month is April. Then the month value is `4` and you can look for animals born in May (month `5`) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

There is a small complication if the current month is December. You cannot merely add one to the month number (`12`) and look for animals born in month `13`, because there is no such month. Instead, you look for animals born in January (month `1`).

You can write the query so that it works no matter what the current month is, so that you do not have to use the number for a particular month. `DATE_ADD()` enables you to add a time interval to a given date. If you add a month to the value of `CURDATE()`, then extract the month part with `MONTH()`, the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
       WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

A different way to accomplish the same task is to add `1` to get the next month after the current one after using the modulo function (`MOD`) to wrap the month value to `0` if it is currently `12`:

```
mysql> SELECT name, birth FROM pet
       WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

`MONTH()` returns a number between `1` and `12`. And `MOD(something,12)` returns a number between `0` and `11`. So the addition has to be after the `MOD()`, otherwise we would go from November (`11`) to January (`1`).

If a calculation uses invalid dates, the calculation fails and produces warnings:

```
mysql> SELECT '2018-10-31' + INTERVAL 1 DAY;
+-----------------------------+
| '2018-10-31' + INTERVAL 1 DAY |
+-----------------------------+
| 2018-11-01                  |
+-----------------------------+
mysql> SELECT '2018-10-32' + INTERVAL 1 DAY;
+-----------------------------+
| '2018-10-32' + INTERVAL 1 DAY |
+-----------------------------+
| NULL                        |
+-----------------------------+
mysql> SHOW WARNINGS;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1292 | Incorrect datetime value: '2018-10-32' |
+---------+------+---------------------------------------+
```

3.3.4.6 Working with NULL Values

The `NULL` value can be surprising until you get used to it. Conceptually, `NULL` means “a missing unknown value” and it is treated somewhat differently from other values.

To test for `NULL`, use the `IS NULL` and `IS NOT NULL` operators, as shown here:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

You cannot use arithmetic comparison operators such as `=`, `<`, or `<>` to test for `NULL`. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

Because the result of any arithmetic comparison with `NULL` is also `NULL`, you cannot obtain any meaningful results from such comparisons.

In MySQL, `0` or `NULL` means false and anything else means true. The default truth value from a boolean operation is `1`.

This special treatment of `NULL` is why, in the previous section, it was necessary to determine which animals are no longer alive using `death IS NOT NULL` instead of `death <> NULL`.

Two `NULL` values are regarded as equal in a `GROUP BY`.

When doing an `ORDER BY`, `NULL` values are presented first if you do `ORDER BY ... ASC` and last if you do `ORDER BY ... DESC`.

A common error when working with `NULL` is to assume that it is not possible to insert a zero or an empty string into a column defined as `NOT NULL`, but this is not the case. These are in fact values, whereas `NULL` means "not having a value." You can test this easily enough by using `IS [NOT] NULL` as shown:

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----------+---------------+------------+----------------+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----------+---------------+------------+----------------+
|         0 |             1 |          0 |              1 |
+-----------+---------------+------------+----------------+
```

Thus it is entirely possible to insert a zero or empty string into a `NOT NULL` column, as these are in fact `NOT NULL`. See Section B.4.4.3, "Problems with NULL Values".

3.3.4.7 Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as `vi`, `grep`, and `sed`.

SQL pattern matching enables you to use `_` to match any single character and `%` to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. Do not use `=` or `<>` when you use SQL patterns. Use the `LIKE` or `NOT LIKE` comparison operators instead.

To find names beginning with `b`:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To find names ending with `fy`:

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a `w`:

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
```

```
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

To find names containing exactly five characters, use five instances of the `_` pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The other type of pattern matching provided by MySQL uses extended regular expressions. When you test for a match for this type of pattern, use the `REGEXP_LIKE()` function (or the `REGEXP` or `RLIKE` operators, which are synonyms for `REGEXP_LIKE()`).

The following list describes some characteristics of extended regular expressions:

- `.` matches any single character.
- A character class `[...]` matches any character within the brackets. For example, `[abc]` matches `a`, `b`, or `c`. To name a range of characters, use a dash. `[a-z]` matches any letter, whereas `[0-9]` matches any digit.
- `*` matches zero or more instances of the thing preceding it. For example, `x*` matches any number of `x` characters, `[0-9]*` matches any number of digits, and `.*` matches any number of anything.
- A regular expression pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a `LIKE` pattern match, which succeeds only if the pattern matches the entire value.)
- To anchor a pattern so that it must match the beginning or end of the value being tested, use `^` at the beginning or `$` at the end of the pattern.

To demonstrate how extended regular expressions work, the `LIKE` queries shown previously are rewritten here to use `REGEXP_LIKE()`.

To find names beginning with `b`, use `^` to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^b');
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To force a regular expression comparison to be case sensitive, use a case-sensitive collation, or use the `BINARY` keyword to make one of the strings a binary string, or specify the `c` match-control character. Each of these queries matches only lowercase `b` at the beginning of a name:

```
SELECT * FROM pet WHERE REGEXP_LIKE(name, '^b' COLLATE utf8mb4_0900_as_cs);
SELECT * FROM pet WHERE REGEXP_LIKE(name, BINARY '^b');
SELECT * FROM pet WHERE REGEXP_LIKE(name, '^b', 'c');
```

To find names ending with `fy`, use `$` to match the end of the name:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, 'fy$');
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a `w`, use this query:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, 'w');
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value as would be true with an SQL pattern.

To find names containing exactly five characters, use `^` and `$` to match the beginning and end of the name, and five instances of `.` in between:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^.....$');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

You could also write the previous query using the `{`*n*`}` (“repeat-*n*-times”) operator:

```
mysql> SELECT * FROM pet WHERE REGEXP_LIKE(name, '^.{5}$');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

For more information about the syntax for regular expressions, see Section 12.5.2, “Regular Expressions”.

3.3.4.8 Counting Rows

Databases are often used to answer the question, “How often does a certain type of data occur in a table?” For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

Counting the total number of animals you have is the same question as “How many rows are in the `pet` table?” because there is one record per pet. `COUNT(*)` counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
```

```
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

Earlier, you retrieved the names of the people who owned pets. You can use `COUNT()` if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

The preceding query uses `GROUP BY` to group all records for each `owner`. The use of `COUNT()` in conjunction with `GROUP BY` is useful for characterizing your data under various groupings. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

(In this output, `NULL` indicates that the sex is unknown.)

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
```

```
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

You need not retrieve an entire table when you use `COUNT()`. For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
       WHERE species = 'dog' OR species = 'cat'
       GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
+---------+------+----------+
```

Or, if you wanted the number of animals per sex only for animals whose sex is known:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
       WHERE sex IS NOT NULL
       GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

If you name columns to select in addition to the `COUNT()` value, a `GROUP BY` clause should be present that names those same columns. Otherwise, the following occurs:

- If the `ONLY_FULL_GROUP_BY` SQL mode is enabled, an error occurs:

```
mysql> SET sql_mode = 'ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression
#1 of SELECT list contains nonaggregated column 'menagerie.pet.owner';
this is incompatible with sql_mode=only_full_group_by
```

- If `ONLY_FULL_GROUP_BY` is not enabled, the query is processed by treating all rows as a single group, but the value selected for each named column is nondeterministic. The server is free to select the value from any row:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Harold |        8 |
```

```
+--------+----------+
1 row in set (0.00 sec)
```

See also Section 12.20.3, “MySQL Handling of GROUP BY”. See Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions” for information about `COUNT(expr)` behavior and related optimizations.

3.3.4.9 Using More Than one Table

The `pet` table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs to contain the following information:

- The pet name so that you know which animal each event pertains to.
- A date so that you know when the event occurred.
- A field to describe the event.
- An event type field, if you want to be able to categorize events.

Given these considerations, the `CREATE TABLE` statement for the `event` table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
       type VARCHAR(15), remark VARCHAR(255));
```

As with the `pet` table, it is easiest to load the initial records by creating a tab-delimited text file containing the following information.

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Based on what you have learned from the queries that you have run on the `pet` table, you should be able to perform retrievals on the records in the `event` table; the principles are the same. But when is the `event` table by itself insufficient to answer questions you might ask?

Suppose that you want to find out the ages at which each pet had its litters. We saw earlier how to calculate ages from two dates. The litter date of the mother is in the `event` table, but to calculate her age on that date you need her birth date, which is stored in the `pet` table. This means the query requires both tables:

```
mysql> SELECT pet.name,
       TIMESTAMPDIFF(YEAR,birth,date) AS age,
       remark
       FROM pet INNER JOIN event
         ON pet.name = event.name
       WHERE event.type = 'litter';
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy |    2 | 4 kittens, 3 female, 1 male |
| Buffy  |    4 | 5 puppies, 2 female, 3 male |
| Buffy  |    5 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

There are several things to note about this query:

- The `FROM` clause joins two tables because the query needs to pull information from both of them.
- When combining (joining) information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy because they both have a `name` column. The query uses an `ON` clause to match up records in the two tables based on the `name` values.

  The query uses an `INNER JOIN` to combine the tables. An `INNER JOIN` permits rows from either table to appear in the result if and only if both tables meet the conditions specified in the `ON` clause. In this example, the `ON` clause specifies that the `name` column in the `pet` table must match the `name` column in the `event` table. If a name appears in one table but not the other, the row will not appear in the result because the condition in the `ON` clause fails.
- Because the `name` column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the `pet` table with itself to produce candidate pairs of live males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
       FROM pet AS p1 INNER JOIN pet AS p2
         ON p1.species = p2.species
         AND p1.sex = 'f' AND p1.death IS NULL
         AND p2.sex = 'm' AND p2.death IS NULL;
+--------+------+-------+------+---------+
| name   | sex  | name  | sex  | species |
+--------+------+-------+------+---------+
| Fluffy | f    | Claws | m    | cat     |
| Buffy  | f    | Fang  | m    | dog     |
+--------+------+-------+------+---------+
```

In this query, we specify aliases for the table name to refer to the columns and keep straight which instance of the table each column reference is associated with.

3.4 Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

You have previously seen `SHOW DATABASES`, which lists the databases managed by the server. To find out which database is currently selected, use the `DATABASE()` function:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

If you have not yet selected any database, the result is `NULL`.

To find out what tables the default database contains (for example, when you are not sure about the name of a table), use this statement:

```
mysql> SHOW TABLES;
+---------------------+
| Tables_in_menagerie |
+---------------------+
| event               |
| pet                 |
+---------------------+
```

The name of the column in the output produced by this statement is always `Tables_in_`*`db_name`*, where *`db_name`* is the name of the database. See Section 13.7.7.37, “SHOW TABLES Syntax”, for more information.

If you want to find out about the structure of a table, the `DESCRIBE` statement is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

`Field` indicates the column name, `Type` is the data type for the column, `NULL` indicates whether the column can contain `NULL` values, `Key` indicates whether the column is indexed, and `Default` specifies the column's default value. `Extra` displays special information about columns: If a column was created with the `AUTO_INCREMENT` option, the value will be `auto_increment` rather than empty.

`DESC` is a short form of `DESCRIBE`. See Section 13.8.1, “DESCRIBE Syntax”, for more information.

You can obtain the `CREATE TABLE` statement necessary to create an existing table using the `SHOW CREATE TABLE` statement. See Section 13.7.7.10, “SHOW CREATE TABLE Syntax”.

If you have indexes on a table, `SHOW INDEX FROM` *`tbl_name`* produces information about them. See Section 13.7.7.22, “SHOW INDEX Syntax”, for more about this statement.

3.5 Using mysql in Batch Mode

In the previous sections, you used `mysql` interactively to enter statements and view the results. You can also run `mysql` in batch mode. To do this, put the statements you want to run in a file, then tell `mysql` to read its input from the file:

```
shell> mysql < batch-file
```

If you are running `mysql` under Windows and have some special characters in the file that cause problems, you can do this:

```
C:\> mysql -e "source batch-file"
```

If you need to specify connection parameters on the command line, the command might look like this:

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

When you use `mysql` this way, you are creating a script file, then executing the script.

If you want the script to continue even if some of the statements in it produce errors, you should use the `--force` command-line option.

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script enables you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line statements or multiple-statement sequences. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell `mysql` to execute it again.
- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

  ```
  shell> mysql < batch-file | more
  ```

- You can catch the output in a file for further processing:

  ```
  shell> mysql < batch-file > mysql.out
  ```

- You can distribute your script to other people so that they can also run the statements.
- Some situations do not allow for interactive use, for example, when you run a query from a `cron` job. In this case, you must use batch mode.

The default output format is different (more concise) when you run `mysql` in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when `mysql` is run interactively:

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

In batch mode, the output looks like this instead:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the statements that are executed, use `mysql -v`.

You can also use scripts from the `mysql` prompt by using the `source` command or `\.` command:

```
mysql> source filename;
mysql> \. filename
```

See Section 4.5.1.5, “Executing SQL Statements from a Text File”, for more information.

3.6 Examples of Common Queries

Here are examples of how to solve some common problems with MySQL.

Some of the examples use the table `shop` to hold the price of each article (item number) for certain traders (dealers). Supposing that each trader has a single fixed price per article, then (`article`, `dealer`) is a primary key for the records.

Start the command-line tool `mysql` and select a database:

```
shell> mysql your-database-name
```

To create and populate the example table, use these statements:

```
CREATE TABLE shop (
    article INT UNSIGNED  DEFAULT '0000' NOT NULL,
    dealer  CHAR(20)      DEFAULT ''     NOT NULL,
    price   DECIMAL(16,2) DEFAULT '0.00' NOT NULL,
    PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
    (1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
    (3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

After issuing the statements, the table should have the following contents:

```
SELECT * FROM shop ORDER BY article;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|       1 | A      |  3.45 |
|       1 | B      |  3.99 |
|       2 | A      | 10.99 |
|       3 | B      |  1.45 |
|       3 | C      |  1.69 |
|       3 | D      |  1.25 |
|       4 | D      | 19.95 |
+---------+--------+-------+
```

3.6.1 The Maximum Value for a Column

“What is the highest item number?”

```
SELECT MAX(article) AS article FROM shop;

+---------+
| article |
+---------+
|       4 |
+---------+
```

3.6.2 The Row Holding the Maximum of a Certain Column

Task: Find the number, dealer, and price of the most expensive article.

This is easily done with a subquery:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Other solutions are to use a `LEFT JOIN` or to sort all rows descending by price and get only the first row using the MySQL-specific `LIMIT` clause:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.price < s2.price
WHERE s2.article IS NULL;

SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

Note

If there were several most expensive articles, each with a price of 19.95, the `LIMIT` solution would show only one of them.

3.6.3 Maximum of Column per Group

Task: Find the highest price per article.

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article
ORDER BY article;

+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
```

```
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

3.6.4 The Rows Holding the Group-wise Maximum of a Certain Column

Task: For each article, find the dealer or dealers with the most expensive price.

This problem can be solved with a subquery like this one:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article)
ORDER BY article;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

The preceding example uses a correlated subquery, which can be inefficient (see Section 13.2.11.7, “Correlated Subqueries”). Other possibilities for solving the problem are to use an uncorrelated subquery in the `FROM` clause, a `LEFT JOIN`, or a common table expression with a window function.

Uncorrelated subquery:

```
SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
  ON s1.article = s2.article AND s1.price = s2.price
ORDER BY article;
```

`LEFT JOIN`:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
WHERE s2.article IS NULL
ORDER BY s1.article;
```

The `LEFT JOIN` works on the basis that when `s1.price` is at its maximum value, there is no `s2.price` with a greater value and thus the corresponding `s2.article` value is `NULL`. See Section 13.2.10.2, “JOIN Syntax”.

Common table expression with window function:

```
WITH s1 AS (
   SELECT article, dealer, price,
          RANK() OVER (PARTITION BY article
                       ORDER BY price DESC
```

```
                      ) AS `Rank`
    FROM shop
)
SELECT article, dealer, price
  FROM s1
  WHERE `Rank` = 1
ORDER BY article;
```

3.6.5 Using User-Defined Variables

You can employ MySQL user variables to remember results without having to store them in temporary variables in the client. (See Section 9.4, “User-Defined Variables”.)

For example, to find the articles with the highest and lowest price you can do this:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Note

It is also possible to store the name of a database object such as a table or a column in a user variable and then to use this variable in an SQL statement; however, this requires the use of a prepared statement. See Section 13.5, “Prepared SQL Statement Syntax”, for more information.

3.6.6 Using Foreign Keys

In MySQL, `InnoDB` tables support checking of foreign key constraints. See Chapter 15, *The InnoDB Storage Engine*, and Section 1.8.2.3, “Foreign Key Differences”.

A foreign key constraint is not required merely to join two tables. For storage engines other than `InnoDB`, it is possible when defining a column to use a `REFERENCES` *`tbl_name(col_name)`* clause, which has no actual effect, and *serves only as a memo or comment to you that the column which you are currently defining is intended to refer to a column in another table*. It is extremely important to realize when using this syntax that:

- MySQL does not perform any sort of check to make sure that *`col_name`* actually exists in *`tbl_name`* (or even that *`tbl_name`* itself exists).
- MySQL does not perform any sort of action on *`tbl_name`* such as deleting rows in response to actions taken on rows in the table which you are defining; in other words, this syntax induces no `ON DELETE` or `ON UPDATE` behavior whatsoever. (Although you can write an `ON DELETE` or `ON UPDATE` clause as part of the `REFERENCES` clause, it is also ignored.)
- This syntax creates a *column*; it does **not** create any sort of index or key.

You can use a column so created as a join column, as shown here:

```
CREATE TABLE person (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);
```

```
CREATE TABLE shirt (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
    PRIMARY KEY (id)
);

INSERT INTO person VALUES (NULL, 'Antonio Paz');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+----+--------------------+
| id | name               |
+----+--------------------+
|  1 | Antonio Paz        |
|  2 | Lilliana Angelovska |
+----+--------------------+

SELECT * FROM shirt;
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
|  3 | t-shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t-shirt | white  |     2 |
+----+---------+--------+-------+


SELECT s.* FROM person p INNER JOIN shirt s
   ON s.owner = p.id
 WHERE p.name LIKE 'Lilliana%'
   AND s.color <> 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

When used in this fashion, the `REFERENCES` clause is not displayed in the output of `SHOW CREATE TABLE` or `DESCRIBE`:

```

```

```
SHOW CREATE TABLE shirt\G
*************************** 1. row ***************************
Table: shirt
Create Table: CREATE TABLE `shirt` (
`id` smallint(5) unsigned NOT NULL auto_increment,
`style` enum('t-shirt','polo','dress') NOT NULL,
`color` enum('red','blue','orange','white','black') NOT NULL,
`owner` smallint(5) unsigned NOT NULL,
PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4
```

The use of `REFERENCES` in this way as a comment or “reminder” in a column definition works with `MyISAM` tables.

3.6.7 Searching on Two Keys

An `OR` using a single key is well optimized, as is the handling of `AND`.

The one tricky case is that of searching on two different keys combined with `OR`:

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR  field2_index = '1'
```

This case is optimized. See Section 8.2.1.3, “Index Merge Optimization”.

You can also solve the problem efficiently by using a `UNION` that combines the output of two separate `SELECT` statements. See Section 13.2.10.3, “UNION Syntax”.

Each `SELECT` searches only one key and can be optimized:

```
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
```

3.6.8 Calculating Visits Per Day

The following example shows how you can use the bit group functions to calculate the number of days per month a user has visited a Web page.

```
CREATE TABLE t1 (year YEAR(4), month INT UNSIGNED,
             day INT UNSIGNED);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
            (2000,2,23),(2000,2,23);
```

The example table contains year-month-day values representing visits by users to the page. To determine how many different days in each month these visits occur, use this query:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
       GROUP BY year,month;
```

Which returns:

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |     1 |    3 |
```

```
| 2000 |     2 |    2 |
+------+-------+------+
```

The query calculates how many different days appear in the table for each year/month combination, with automatic removal of duplicate entries.

3.6.9 Using AUTO_INCREMENT

The `AUTO_INCREMENT` attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (
     id MEDIUMINT NOT NULL AUTO_INCREMENT,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
);

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich');

SELECT * FROM animals;
```

Which returns:

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

No value was specified for the `AUTO_INCREMENT` column, so MySQL assigned sequence numbers automatically. You can also explicitly assign 0 to the column to generate sequence numbers, unless the `NO_AUTO_VALUE_ON_ZERO` SQL mode is enabled. For example:

```
INSERT INTO animals (id,name) VALUES(0,'groundhog');
```

If the column is declared `NOT NULL`, it is also possible to assign `NULL` to the column to generate sequence numbers. For example:

```
INSERT INTO animals (id,name) VALUES(NULL,'squirrel');
```

When you insert any other value into an `AUTO_INCREMENT` column, the column is set to that value and the sequence is reset so that the next automatically generated value follows sequentially from the largest column value. For example:

```
INSERT INTO animals (id,name) VALUES(100,'rabbit');
INSERT INTO animals (id,name) VALUES(NULL,'mouse');
SELECT * FROM animals;
+-----+-----------+
| id  | name      |
+-----+-----------+
|   1 | dog       |
|   2 | cat       |
```

```
|   3 | penguin   |
|   4 | lax       |
|   5 | whale     |
|   6 | ostrich   |
|   7 | groundhog |
|   8 | squirrel  |
| 100 | rabbit    |
| 101 | mouse     |
+-----+-----------+
```

Updating an existing `AUTO_INCREMENT` column value also resets the `AUTO_INCREMENT` sequence.

You can retrieve the most recent automatically generated `AUTO_INCREMENT` value with the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function. These functions are connection-specific, so their return values are not affected by another connection which is also performing inserts.

Use the smallest integer data type for the `AUTO_INCREMENT` column that is large enough to hold the maximum sequence value you will need. When the column reaches the upper limit of the data type, the next attempt to generate a sequence number fails. Use the `UNSIGNED` attribute if possible to allow a greater range. For example, if you use `TINYINT`, the maximum permissible sequence number is 127. For `TINYINT UNSIGNED`, the maximum is 255. See Section 11.2.1, “Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT” for the ranges of all the integer types.

Note

For a multiple-row insert, `LAST_INSERT_ID()` and `mysql_insert_id()` actually return the `AUTO_INCREMENT` key from the *first* of the inserted rows. This enables multiple-row inserts to be reproduced correctly on other servers in a replication setup.

To start with an `AUTO_INCREMENT` value other than 1, set that value with `CREATE TABLE` or `ALTER TABLE`, like this:

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

InnoDB Notes

For information about `AUTO_INCREMENT` usage specific to `InnoDB`, see Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”.

MyISAM Notes

- For `MyISAM` tables, you can specify `AUTO_INCREMENT` on a secondary column in a multiple-column index. In this case, the generated value for the `AUTO_INCREMENT` column is calculated as `MAX(`*`auto_increment_column`*`) + 1 WHERE prefix=`*`given-prefix`*. This is useful when you want to put data into ordered groups.

```
CREATE TABLE animals (
    grp ENUM('fish','mammal','bird') NOT NULL,
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (grp,id)
) ENGINE=MyISAM;

INSERT INTO animals (grp,name) VALUES
    ('mammal','dog'),('mammal','cat'),
    ('bird','penguin'),('fish','lax'),('mammal','whale'),
```

```
    ('bird','ostrich');

SELECT * FROM animals ORDER BY grp,id;
```

Which returns:

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

In this case (when the `AUTO_INCREMENT` column is part of a multiple-column index), `AUTO_INCREMENT` values are reused if you delete the row with the biggest `AUTO_INCREMENT` value in any group. This happens even for `MyISAM` tables, for which `AUTO_INCREMENT` values normally are not reused.

- If the `AUTO_INCREMENT` column is part of multiple indexes, MySQL generates sequence values using the index that begins with the `AUTO_INCREMENT` column, if there is one. For example, if the `animals` table contained indexes `PRIMARY KEY (grp, id)` and `INDEX (id)`, MySQL would ignore the `PRIMARY KEY` for generating sequence values. As a result, the table would contain a single sequence, not a sequence per `grp` value.

Further Reading

More information about `AUTO_INCREMENT` is available here:

- How to assign the `AUTO_INCREMENT` attribute to a column: Section 13.1.20, “CREATE TABLE Syntax”, and Section 13.1.9, “ALTER TABLE Syntax”.
- How `AUTO_INCREMENT` behaves depending on the `NO_AUTO_VALUE_ON_ZERO` SQL mode: Section 5.1.11, “Server SQL Modes”.
- How to use the `LAST_INSERT_ID()` function to find the row that contains the most recent `AUTO_INCREMENT` value: Section 12.15, “Information Functions”.
- Setting the `AUTO_INCREMENT` value to be used: Section 5.1.8, “Server System Variables”.
- Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
- `AUTO_INCREMENT` and replication: Section 17.5.1.1, “Replication and AUTO_INCREMENT”.
- Server-system variables related to `AUTO_INCREMENT` (`auto_increment_increment` and `auto_increment_offset`) that can be used for replication: Section 5.1.8, “Server System Variables”.

3.7 Using MySQL with Apache

There are programs that let you authenticate your users from a MySQL database and also let you write your log files into a MySQL table.

You can change the Apache logging format to be easily readable by MySQL by putting the following into the Apache configuration file:

```
```

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

To load a log file in that format into MySQL, you can use a statement something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

The named table should be created to have columns that correspond to those that the `LogFormat` line writes to the log file.

Chapter 4 MySQL Programs

Table of Contents

4.1 Overview of MySQL Programs 314
4.2 Using MySQL Programs 317
4.2.1 Invoking MySQL Programs 317
4.2.2 Specifying Program Options 318
4.2.3 Command Options for Connecting to the Server 332
4.2.4 Connecting to the MySQL Server Using Command Options 340
4.2.5 Connecting to the Server Using URI-Like Strings or Key-Value Pairs 343
4.2.6 Connection Compression Control 349
4.2.7 Setting Environment Variables 353
4.3 MySQL Server and Server-Startup Programs 354
4.3.1 `mysqld` — The MySQL Server 354
4.3.2 `mysqld_safe` — MySQL Server Startup Script 355
4.3.3 `mysql.server` — MySQL Server Startup Script 361
4.3.4 `mysqld_multi` — Manage Multiple MySQL Servers 363
4.4 MySQL Installation-Related Programs 367
4.4.1 `comp_err` — Compile MySQL Error Message File 367
4.4.2 `mysql_secure_installation` — Improve MySQL Installation Security 368
4.4.3 `mysql_ssl_rsa_setup` — Create SSL/RSA Files 372
4.4.4 `mysql_tzinfo_to_sql` — Load the Time Zone Tables 375
4.4.5 `mysql_upgrade` — Check and Upgrade MySQL Tables 376
4.5 MySQL Client Programs 385
4.5.1 `mysql` — The MySQL Command-Line Client 385
4.5.2 `mysqladmin` — Client for Administering a MySQL Server 415
4.5.3 `mysqlcheck` — A Table Maintenance Program 426
4.5.4 `mysqldump` — A Database Backup Program 436
4.5.5 `mysqlimport` — A Data Import Program 461
4.5.6 `mysqlpump` — A Database Backup Program 470
4.5.7 `mysqlshow` — Display Database, Table, and Column Information 487
4.5.8 `mysqlslap` — Load Emulation Client 495
4.6 MySQL Administrative and Utility Programs 505
4.6.1 `ibd2sdi` — InnoDB Tablespace SDI Extraction Utility 505
4.6.2 `innochecksum` — Offline InnoDB File Checksum Utility 509
4.6.3 `myisam_ftdump` — Display Full-Text Index information 514
4.6.4 `myisamchk` — MyISAM Table-Maintenance Utility 516
4.6.5 `myisamlog` — Display MyISAM Log File Contents 533
4.6.6 `myisampack` — Generate Compressed, Read-Only MyISAM Tables 534
4.6.7 `mysql_config_editor` — MySQL Configuration Utility 540
4.6.8 `mysqlbinlog` — Utility for Processing Binary Log Files 547
4.6.9 `mysqldumpslow` — Summarize Slow Query Log Files 571
4.7 MySQL Program Development Utilities 573
4.7.1 `mysql_config` — Display Options for Compiling Clients 573
4.7.2 `my_print_defaults` — Display Options from Option Files 575
4.8 Miscellaneous Programs 576
4.8.1 `lz4_decompress` — Decompress mysqlpump LZ4-Compressed Output 576
4.8.2 `perror` — Display MySQL Error Message Information 577
4.8.3 `zlib_decompress` — Decompress mysqlpump ZLIB-Compressed Output 578
4.9 MySQL Environment Variables 578

This chapter provides a brief overview of the MySQL command-line programs provided by Oracle Corporation. It also discusses the general syntax for specifying options when you run these programs. Most programs have options that are specific to their own operation, but the option syntax is similar for all of them. Finally, the chapter provides more detailed descriptions of individual programs, including which options they recognize.

4.1 Overview of MySQL Programs

There are many different programs in a MySQL installation. This section provides a brief overview of them. Later sections provide a more detailed description of each one, with the exception of NDB Cluster programs. Each program's description indicates its invocation syntax and the options that it supports. Section 22.4, “NDB Cluster Programs”, describes programs specific to NDB Cluster.

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see Chapter 2, *Installing and Upgrading MySQL*, for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install an additional package.

Each MySQL program takes many different options. Most programs provide a `--help` option that you can use to get a description of the program's different options. For example, try `mysql --help`.

You can override default option values for MySQL programs by specifying options on the command line or in an option file. See Section 4.2, “Using MySQL Programs”, for general information on invoking programs and specifying program options.

The MySQL server, `mysqld`, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that assist you in starting and stopping the server:

- `mysqld`

  The SQL daemon (that is, the MySQL server). To use client programs, `mysqld` must be running, because clients gain access to databases by connecting to the server. See Section 4.3.1, “`mysqld` — The MySQL Server”.

- `mysqld_safe`

  A server startup script. `mysqld_safe` attempts to start `mysqld`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

- `mysql.server`

  A server startup script. This script is used on systems that use System V-style run directories containing scripts that start system services for particular run levels. It invokes `mysqld_safe` to start the MySQL server. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.

- `mysqld_multi`

  A server startup script that can start or stop multiple servers installed on the system. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.

Several programs perform setup operations during MySQL installation or upgrading:

- `comp_err`

  This program is used during the MySQL build/installation process. It compiles error message files from the error source files. See Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”.

- `mysql_secure_installation`

  This program enables you to improve the security of your MySQL installation. See Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”.

- `mysql_ssl_rsa_setup`

  This program creates the SSL certificate and key files and RSA key-pair files required to support secure connections, if those files are missing. Files created by `mysql_ssl_rsa_setup` can be used for secure connections using SSL or RSA. See Section 4.4.3, “`mysql_ssl_rsa_setup` — Create SSL/RSA Files”.

- `mysql_tzinfo_to_sql`

  This program loads the time zone tables in the `mysql` database using the contents of the host system *zoneinfo* database (the set of files describing time zones). See Section 4.4.4, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”.

- `mysql_upgrade`

  This program is used after a MySQL upgrade operation. It updates the grant tables with any changes that have been made in newer versions of MySQL, and checks tables for incompatibilities and repairs them if necessary. See Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

MySQL client programs that connect to the MySQL server:

- `mysql`

  The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See Section 4.5.1, “`mysql` — The MySQL Command-Line Client”.

- `mysqladmin`

  A client that performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. `mysqladmin` can also be used to retrieve version, process, and status information from the server. See Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”.

- `mysqlcheck`

  A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”.

- `mysqldump`

  A client that dumps a MySQL database into a file as SQL, text, or XML. See Section 4.5.4, “`mysqldump` — A Database Backup Program”.

- `mysqlimport`

  A client that imports text files into their respective tables using `LOAD DATA`. See Section 4.5.5, “`mysqlimport` — A Data Import Program”.

- `mysqlpump`

  A client that dumps a MySQL database into a file as SQL. See Section 4.5.6, “`mysqlpump` — A Database Backup Program”.

- `mysqlsh`

MySQL Shell is an advanced client and code editor for MySQL Server. See MySQL Shell 8.0 (part of MySQL 8.0). In addition to the provided SQL functionality, similar to `mysql`, MySQL Shell provides scripting capabilities for JavaScript and Python and includes APIs for working with MySQL. X DevAPI enables you to work with both relational and document data, see Chapter 20, *Using MySQL as a Document Store*. AdminAPI enables you to work with InnoDB cluster, see Chapter 21, *InnoDB Cluster*.

- `mysqlshow`

  A client that displays information about databases, tables, columns, and indexes. See Section 4.5.7, "`mysqlshow` — Display Database, Table, and Column Information".

- `mysqlslap`

  A client that is designed to emulate client load for a MySQL server and report the timing of each stage. It works as if multiple clients are accessing the server. See Section 4.5.8, "`mysqlslap` — Load Emulation Client".

MySQL administrative and utility programs:

- `innochecksum`

  An offline `InnoDB` offline file checksum utility. See Section 4.6.2, "`innochecksum` — Offline InnoDB File Checksum Utility".

- `myisam_ftdump`

  A utility that displays information about full-text indexes in `MyISAM` tables. See Section 4.6.3, "`myisam_ftdump` — Display Full-Text Index information".

- `myisamchk`

  A utility to describe, check, optimize, and repair `MyISAM` tables. See Section 4.6.4, "`myisamchk` — MyISAM Table-Maintenance Utility".

- `myisamlog`

  A utility that processes the contents of a `MyISAM` log file. See Section 4.6.5, "`myisamlog` — Display MyISAM Log File Contents".

- `myisampack`

  A utility that compresses `MyISAM` tables to produce smaller read-only tables. See Section 4.6.6, "`myisampack` — Generate Compressed, Read-Only MyISAM Tables".

- `mysql_config_editor`

  A utility that enables you to store authentication credentials in a secure, encrypted login path file named `.mylogin.cnf`. See Section 4.6.7, "`mysql_config_editor` — MySQL Configuration Utility".

- `mysqlbinlog`

  A utility for reading statements from a binary log. The log of executed statements contained in the binary log files can be used to help recover from a crash. See Section 4.6.8, "`mysqlbinlog` — Utility for Processing Binary Log Files".

- `mysqldumpslow`

A utility to read and summarize the contents of a slow query log. See Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”.

MySQL program-development utilities:

- `mysql_config`

  A shell script that produces the option values needed when compiling MySQL programs. See Section 4.7.1, “mysql_config — Display Options for Compiling Clients”.

- `my_print_defaults`

  A utility that shows which options are present in option groups of option files. See Section 4.7.2, “my_print_defaults — Display Options from Option Files”.

Miscellaneous utilities:

- `lz4_decompress`

  A utility that decompresses `mysqlpump` output that was created using LZ4 compression. See Section 4.8.1, “lz4_decompress — Decompress mysqlpump LZ4-Compressed Output”.

- `perror`

  A utility that displays the meaning of system or MySQL error codes. See Section 4.8.2, “perror — Display MySQL Error Message Information”.

- `zlib_decompress`

  A utility that decompresses `mysqlpump` output that was created using ZLIB compression. See Section 4.8.3, “zlib_decompress — Decompress mysqlpump ZLIB-Compressed Output”.

Oracle Corporation also provides the MySQL Workbench GUI tool, which is used to administer MySQL servers and databases, to create, execute, and evaluate queries, and to migrate schemas and data from other relational database management systems for use with MySQL. Additional GUI tools include MySQL Notifier and MySQL for Excel.

MySQL client programs that communicate with the server using the MySQL client/server library use the following environment variables.

Environment Variable	Meaning
`MYSQL_UNIX_PORT`	The default Unix socket file; used for connections to `localhost`
`MYSQL_TCP_PORT`	The default port number; used for TCP/IP connections
`MYSQL_DEBUG`	Debug trace options when debugging
`TMPDIR`	The directory where temporary tables and files are created

For a full list of environment variables used by MySQL programs, see Section 4.9, “MySQL Environment Variables”.

4.2 Using MySQL Programs

4.2.1 Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you

want it to do. The following commands show some sample program invocations. `shell>` represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are `$` for `sh`, `ksh`, or `bash`, `%` for `csh` or `tcsh`, and `C:\>` for the Windows `command.com` or `cmd.exe` command interpreters.

```
shell> mysql --user=root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump -u root personnel
```

Arguments that begin with a single or double dash (`-`, `--`) specify program options. Options typically indicate the type of connection a program should make to the server or affect its operational mode. Option syntax is described in Section 4.2.2, “Specifying Program Options”.

Nonoption arguments (arguments with no leading dash) provide additional information to the program. For example, the `mysql` program interprets the first nonoption argument as a database name, so the command `mysql --user=root test` indicates that you want to use the `test` database.

Later sections that describe individual programs indicate which options a program supports and describe the meaning of any additional nonoption arguments.

Some options are common to a number of programs. The most frequently used of these are the `--host` (or `-h`), `--user` (or `-u`), and `--password` (or `-p`) options that specify connection parameters. They indicate the host where the MySQL server is running, and the user name and password of your MySQL account. All MySQL client programs understand these options; they enable you to specify which server to connect to and the account to use on that server. Other connection options are `--port` (or `-P`) to specify a TCP/IP port number and `--socket` (or `-S`) to specify a Unix socket file on Unix (or named-pipe name on Windows). For more information on options that specify connection options, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

You may find it necessary to invoke MySQL programs using the path name to the `bin` directory in which they are installed. This is likely to be the case if you get a “program not found” error whenever you attempt to run a MySQL program from any directory other than the `bin` directory. To make it more convenient to use MySQL, you can add the path name of the `bin` directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. For example, if `mysql` is installed in `/usr/local/mysql/bin`, you can run the program by invoking it as `mysql`, and it is not necessary to invoke it as `/usr/local/mysql/bin/mysql`.

Consult the documentation for your command interpreter for instructions on setting your `PATH` variable. The syntax for setting environment variables is interpreter-specific. (Some information is given in Section 4.2.7, “Setting Environment Variables”.) After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.2.2 Specifying Program Options

There are several ways to specify options for MySQL programs:

- List the options on the command line following the program name. This is common for options that apply to a specific invocation of the program.
- List the options in an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.
- List the options in environment variables (see Section 4.2.7, “Setting Environment Variables”). This method is useful for options that you want to apply each time the program runs. In practice, option files are used more commonly for this purpose, but Section 5.8.3, “Running Multiple MySQL Instances on

Unix", discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for the server and for client programs.

Options are processed in order, so if an option is specified multiple times, the last occurrence takes precedence. The following command causes `mysql` to connect to the server running on `localhost`:

```
mysql -h example.com -h localhost
```

There is one exception: For `mysqld`, the *first* instance of the `--user` option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

If conflicting or related options are given, later options take precedence over earlier options. The following command runs `mysql` in "no column names" mode:

```
mysql --column-names --skip-column-names
```

MySQL programs determine which options are given first by examining environment variables, then by processing option files, and then by checking the command line. Because later options take precedence over earlier ones, the processing order means that environment variables have the lowest precedence and command-line options the highest.

For the server, one exception applies: The `mysqld-auto.cnf` option file in the data directory is processed last, so it takes precedence even over command-line options.

You can take advantage of the way that MySQL programs process options by specifying default option values for a program in an option file. That enables you to avoid typing them each time you run the program while enabling you to override the defaults if necessary by using command-line options.

4.2.2.1 Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.
- An option argument begins with one dash or two dashes, depending on whether it is a short form or long form of the option name. Many options have both short and long forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display its help message.
- Option names are case-sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)
- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.
- For a long option that takes a value, separate the option name and the value by an `=` sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between: `-hlocalhost` and `-h localhost` are equivalent. An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `--password=pass_val` or as `--password`. In the latter case (with no password value given), the program interactively prompts you for the password. The password option also may be given in short form as `-ppass_val` or as `-p`. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*: If a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind of argument. Consequently, the following two commands have two completely different meanings:

```
mysql -ptest
mysql -p test
```

The first command instructs `mysql` to use a password value of `test`, but specifies no default database. The second instructs `mysql` to prompt for the password value and to use `test` as the default database.

- Within option names, dash (`-`) and underscore (_) may be used interchangeably. For example, `--skip-grant-tables` and `--skip_grant_tables` are equivalent. (However, the leading dashes cannot be given as underscores.)
- The MySQL server has certain command options that may be specified only at startup, and a set of system variables, some of which may be set at startup, at runtime, or both. System variable names use underscores rather than dashes, and when referenced at runtime (for example, using `SET` or `SELECT` statements), must be written using underscores:

  ```
  SET GLOBAL general_log = ON;
  SELECT @@GLOBAL.general_log;
  ```

  At server startup, the syntax for system variables is the same as for command options, so within variable names, dashes and underscores may be used interchangeably. For example, `--general_log=ON` and `--general-log=ON` are equivalent. (This is also true for system variables set within option files.)
- For options that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` to indicate a multiplier of 1024, 1024^2 or 1024^3. As of MySQL 8.0.14, a suffix can also be `T`, `P`, and `E` to indicate a multiplier of 1024^4, 1024^5 or 1024^6. Suffix letters can be uppercase or lowercase.

  For example, the following command tells `mysqladmin` to ping the server 1024 times, sleeping 10 seconds between each ping:

  ```
  mysqladmin --count=1K --sleep=10 ping
  ```
- When specifying file names as option values, avoid the use of the `~` shell metacharacter. It might not be interpreted as you expect.

Option values that contain spaces must be quoted when given on the command line. For example, the `--execute` (or `-e`) option can be used with `mysql` to pass one or more semicolon-separated SQL statements to the server. When this option is used, `mysql` executes the statements in the option value and exits. The statements must be enclosed by quotation marks. For example:

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW()"
Enter password: ******
+------------+
| VERSION()  |
+------------+
| 8.0.19     |
+------------+
+---------------------+
| NOW()               |
+---------------------+
| 2019-09-03 10:36:48 |
+---------------------+
shell>
```

Note

The long form (`--execute`) is followed by an equal sign (`=`).

To use quoted values within a statement, you must either escape the inner quotation marks, or use a different type of quotation marks within the statement from those used to quote the statement itself. The capabilities of your command processor dictate your choices for whether you can use single or double quotation marks and the syntax for escaping quote characters. For example, if your command processor supports quoting with single or double quotation marks, you can use double quotation marks around the statement, and single quotation marks for any quoted values within the statement.

4.2.2.2 Using Option Files

Most MySQL programs can read startup options from option files (sometimes called configuration files). Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program.

To determine whether a program reads option files, invoke it with the `--help` option. (For `mysqld`, use `--verbose` and `--help`.) If the program reads option files, the help message indicates which files it looks for and which option groups it recognizes.

Note

A MySQL program started with the `--no-defaults` option reads no option files other than `.mylogin.cnf`.

A server started with the `persisted_globals_load` system variable disabled does not read `mysqld-auto.cnf`.

Many option files are plain text files, created using any text editor. The exceptions are:

- The `.mylogin.cnf` file that contains login path options. This is an encrypted file created by the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”. A “login path” is an option group that permits only certain options: `host`, `user`, `password`, `port` and `socket`. Client programs specify which login path to read from `.mylogin.cnf` using the `--login-path` option.

  To specify an alternative login path file name, set the `MYSQL_TEST_LOGIN_FILE` environment variable. This variable is used by the `mysql-test-run.pl` testing utility, but also is recognized by `mysql_config_editor` and by MySQL clients such as `mysql`, `mysqladmin`, and so forth.

- The `mysqld-auto.cnf` file in the data directory. This JSON-format file contains persisted system variable settings. It is created by the server upon execution of `SET PERSIST` or `SET PERSIST_ONLY` statements. See Section 5.1.9.3, “Persisted System Variables”. Management of `mysqld-auto.cnf` should be left to the server and not performed manually.

- Option File Processing Order
- Option File Syntax
- Option File Inclusions

Option File Processing Order

MySQL looks for option files in the order described in the following discussion and reads any that exist. If an option file you want to use does not exist, create it using the appropriate method, as just discussed.

Note

For information about option files used with NDB Cluster programs, see Section 22.3, “Configuration of NDB Cluster”.

On Windows, MySQL programs read startup options from the files shown in the following table, in the specified order (files listed first are read first, files read later take precedence).

Table 4.1 Option Files Read on Windows Systems

File Name	Purpose
`%WINDIR%\my.ini`, `%WINDIR%\my.cnf`	Global options
`C:\my.ini`, `C:\my.cnf`	Global options
`BASEDIR``\my.ini`, *`BASEDIR`*`\my.cnf`	Global options
`defaults-extra-file`	The file specified with `--defaults-extra-file`, if any
`%APPDATA%\MySQL\.mylogin.cnf`	Login path options (clients only)
`DATADIR``\mysqld-auto.cnf`	System variables persisted with `SET PERSIST` or `SET PERSIST_ONLY` (server only)

In the preceding table, `%WINDIR%` represents the location of your Windows directory. This is commonly `C:\WINDOWS`. Use the following command to determine its exact location from the value of the `WINDIR` environment variable:

```
C:\> echo %WINDIR%
```

`%APPDATA%` represents the value of the Windows application data directory. Use the following command to determine its exact location from the value of the `APPDATA` environment variable:

```
C:\> echo %APPDATA%
```

`BASEDIR` represents the MySQL base installation directory. When MySQL 8.0 has been installed using MySQL Installer, this is typically `C:\`*`PROGRAMDIR`*`\MySQL\MySQL 8.0 Server` where *`PROGRAMDIR`* represents the programs directory (usually `Program Files` on English-language versions of Windows), See Section 2.3.3, “MySQL Installer for Windows”.

`DATADIR` represents the MySQL data directory. As used to find `mysqld-auto.cnf`, its default value is the data directory location built in when MySQL was compiled, but can be changed by `--datadir` specified as an option-file or command-line option processed before `mysqld-auto.cnf` is processed.

On Unix and Unix-like systems, MySQL programs read startup options from the files shown in the following table, in the specified order (files listed first are read first, files read later take precedence).

Note

On Unix platforms, MySQL ignores configuration files that are world-writable. This is intentional as a security measure.

Table 4.2 Option Files Read on Unix and Unix-Like Systems

File Name	Purpose
`/etc/my.cnf`	Global options
`/etc/mysql/my.cnf`	Global options
`SYSCONFDIR``/my.cnf`	Global options
`$MYSQL_HOME/my.cnf`	Server-specific options (server only)

File Name	Purpose
`defaults-extra-file`	The file specified with `--defaults-extra-file`, if any
`~/.my.cnf`	User-specific options
`~/.mylogin.cnf`	User-specific login path options (clients only)
`DATADIR``/mysqld-auto.cnf`	System variables persisted with `SET PERSIST` or `SE PERSIST_ONLY` (server only)

In the preceding table, `~` represents the current user's home directory (the value of `$HOME`).

`SYSCONFDIR` represents the directory specified with the `SYSCONFDIR` option to `CMake` when MySQL was built. By default, this is the `etc` directory located under the compiled-in installation directory.

`MYSQL_HOME` is an environment variable containing the path to the directory in which the server-specific `my.cnf` file resides. If `MYSQL_HOME` is not set and you start the server using the `mysqld_safe` program, `mysqld_safe` sets it to *`BASEDIR`*, the MySQL base installation directory.

`DATADIR` represents the MySQL data directory. As used to find `mysqld-auto.cnf`, its default value is the data directory location built in when MySQL was compiled, but can be changed by `--datadir` specified as an option-file or command-line option processed before `mysqld-auto.cnf` is processed.

If multiple instances of a given option are found, the last instance takes precedence, with one exception: For `mysqld`, the *first* instance of the `--user` option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

Option File Syntax

The following description of option file syntax applies to files that you edit manually. This excludes `.mylogin.cnf`, which is created using `mysql_config_editor` and is encrypted, and `mysqld-auto.cnf`, which the server creates in JSON format.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the `--help` option. (For `mysqld`, use `--verbose` and `--help`.)

The syntax for specifying options in an option file is similar to command-line syntax (see Section 4.2.2.1, “Using Options on the Command Line”). However, in an option file, you omit the leading two dashes from the option name and you specify only one option per line. For example, `--quick` and `--host=localhost` on the command line should be specified as `quick` and `host=localhost` on separate lines in an option file. To specify an option of the form `--loose-`*`opt_name`* in an option file, write it as `loose-`*`opt_name`*.

Empty lines in option files are ignored. Nonempty lines can take any of the following forms:

- *`#comment`*, *`;comment`*

  Comment lines start with `#` or `;`. A `#` comment can start in the middle of a line as well.

- `[`*`group`*`]`

  `group` is the name of the program or group for which you want to set options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given. Option group names are not case-sensitive.

- *`opt_name`*

  This is equivalent to `--`*`opt_name`* on the command line.

- *opt_name=value*

  This is equivalent to *--opt_name=value* on the command line. In an option file, you can have spaces around the = character, something that is not true on the command line. The value optionally can be enclosed within single quotation marks or double quotation marks, which is useful if the value contains a # comment character.

Leading and trailing spaces are automatically deleted from option names and values.

You can use the escape sequences \b, \t, \n, \r, \\, and \s in option values to represent the backspace, tab, newline, carriage return, backslash, and space characters. In option files, these escaping rules apply:

- A backslash followed by a valid escape sequence character is converted to the character represented by the sequence. For example, \s is converted to a space.
- A backslash not followed by a valid escape sequence character remains unchanged. For example, \S is retained as is.

The preceding rules mean that a literal backslash can be given as \\, or as \ if it is not followed by a valid escape sequence character.

The rules for escape sequences in option files differ slightly from the rules for escape sequences in string literals in SQL statements. In the latter context, if "*x*" is not a valid escape sequence character, *x* becomes "*x*" rather than *x*. See Section 9.1.1, "String Literals".

The escaping rules for option file values are especially pertinent for Windows path names, which use \ as a path name separator. A separator in a Windows path name must be written as \\ if it is followed by an escape sequence character. It can be written as \\ or \ if it is not. Alternatively, / may be used in Windows path names and will be treated as \. Suppose that you want to specify a base directory of C:\Program Files\MySQL\MySQL Server 8.0 in an option file. This can be done several ways. Some examples:

```
basedir="C:\Program Files\MySQL\MySQL Server 8.0"
basedir="C:\\Program Files\\MySQL\\MySQL Server 8.0"
basedir="C:/Program Files/MySQL/MySQL Server 8.0"
basedir=C:\\Program\sFiles\\MySQL\\MySQL\sServer\s8.0
```

If an option group name is the same as a program name, options in the group apply specifically to that program. For example, the [mysqld] and [mysql] groups apply to the mysqld server and the mysql client program, respectively.

The [client] option group is read by all client programs provided in MySQL distributions (but *not* by mysqld). To understand how third-party client programs that use the C API can use option files, see the C API documentation at Section 28.7.7.50, "mysql_options()".

The [client] group enables you to specify options that apply to all clients. For example, [client] is the appropriate group to use to specify the password for connecting to the server. (But make sure that the option file is accessible only by yourself, so that other people cannot discover your password.) Be sure not to put an option in the [client] group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

List more general option groups first and more specific groups later. For example, a [client] group is more general because it is read by all client programs, whereas a [mysqldump] group is read only by mysqldump. Options specified later override options specified earlier, so putting the option groups in the order [client], [mysqldump] enables mysqldump-specific options to override [client] options.

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=128M

[mysqldump]
quick
```

Here is a typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password="my password"

[mysql]
no-auto-rehash
connect_timeout=2
```

To create option groups to be read only by `mysqld` servers from specific MySQL release series, use groups with names of `[mysqld-5.7]`, `[mysqld-8.0]`, and so forth. The following group indicates that the `sql_mode` setting should be used only by MySQL servers with 8.0.x version numbers:

```
[mysqld-8.0]
sql_mode=TRADITIONAL
```

Option File Inclusions

It is possible to use `!include` directives in option files to include other option files and `!includedir` to search specific directories for option files. For example, to include the `/home/mydir/myopt.cnf` file, use the following directive:

```
!include /home/mydir/myopt.cnf
```

To search the `/home/mydir` directory and read option files found there, use this directive:

```
!includedir /home/mydir
```

MySQL makes no guarantee about the order in which option files in the directory are read.

Note

Any files to be found and included using the `!includedir` directive on Unix operating systems *must* have file names ending in `.cnf`. On Windows, this directive checks for files with the `.ini` or `.cnf` extension.

Write the contents of an included option file like any other option file. That is, it should contain groups of options, each preceded by a `[group]` line that indicates the program to which the options apply.

While an included file is being processed, only those options in groups that the current program is looking for are used. Other groups are ignored. Suppose that a `my.cnf` file contains this line:

```
!include /home/mydir/myopt.cnf
```

And suppose that `/home/mydir/myopt.cnf` looks like this:

```
[mysqladmin]
force

[mysqld]
key_buffer_size=16M
```

If `my.cnf` is processed by `mysqld`, only the `[mysqld]` group in `/home/mydir/myopt.cnf` is used. If the file is processed by `mysqladmin`, only the `[mysqladmin]` group is used. If the file is processed by any other program, no options in `/home/mydir/myopt.cnf` are used.

The `!includedir` directive is processed similarly except that all option files in the named directory are read.

If an option file contains `!include` or `!includedir` directives, files named by those directives are processed whenever the option file is processed, no matter where they appear in the file.

4.2.2.3 Command-Line Options that Affect Option-File Handling

Most MySQL programs that support option files handle the following options. Because these options affect option-file handling, they must be given on the command line and not in an option file. To work properly, each of these options must be given before other options, with these exceptions:

- `--print-defaults` may be used immediately after `--defaults-file`, `--defaults-extra-file`, or `--login-path`.
- On Windows, if the server is started with the `--defaults-file` and `--install` options, `--install` must be first. See Section 2.3.4.8, “Starting MySQL as a Windows Service”.

When specifying file names as option values, avoid the use of the `~` shell metacharacter because it might not be interpreted as you expect.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file and (on all platforms) before the login path file. (For information about the order in which option files are used, see Section 4.2.2.2, “Using Option Files”.) If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  See the introduction to this section regarding constraints on the position in which this option may be specified.

- `--defaults-file=`*`file_name`*

  Read only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exceptions: Even with `--defaults-file`, `mysqld` reads `mysqld-auto.cnf` and client programs read `.mylogin.cnf`.

  See the introduction to this section regarding constraints on the position in which this option may be specified.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, the `mysql` client normally reads the `[client]` and `[mysql]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql` also reads the `[client_other]` and `[mysql_other]` groups.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.

  A client program reads the option group corresponding to the named login path, in addition to option groups that the program reads by default. Consider this command:

  ```
  mysql --login-path=mypath
  ```

  By default, the `mysql` client reads the `[client]` and `[mysql]` option groups. So for the command shown, `mysql` reads `[client]` and `[mysql]` from other option files, and `[client]`, `[mysql]`, and `[mypath]` from the login path file.

  Client programs read the login path file even when the `--no-defaults` option is used.

  To specify an alternate login path file name, set the `MYSQL_TEST_LOGIN_FILE` environment variable.

  See the introduction to this section regarding constraints on the position in which this option may be specified.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that client programs read the `.mylogin.cnf` login path file, if it exists, even when `--no-defaults` is used. This permits passwords to be specified in a safer way than on the command line even if `--no-defaults` is present. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.)

- `--print-defaults`

  Print the program name and all options that it gets from option files. Password values are masked.

  See the introduction to this section regarding constraints on the position in which this option may be specified.

4.2.2.4 Program Option Modifiers

Some options are “boolean” and control behavior that can be turned on or off. For example, the `mysql` client supports a `--column-names` option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you may want to disable it in some instances, such as when sending the output of `mysql` into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The “enabled” form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

The values `ON`, `TRUE`, `OFF`, and `FALSE` are also recognized for boolean options (not case-sensitive).

If an option is prefixed by `--loose`, a program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--loose-no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine and list options in an option file. An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that recognize the option process it normally, and versions that do not recognize it issue a warning and ignore it.

The `--maximum` prefix is available for `mysqld` only and permits a limit to be placed on how large client programs can set session system variables. To do this, use a `--maximum` prefix with the variable name. For example, `--maximum-max_heap_table_size=32M` prevents any client from making the heap table size limit larger than 32M.

The `--maximum` prefix is intended for use with system variables that have a session value. If applied to a system variable that has only a global value, an error occurs. For example, with `--maximum-back_log=200`, the server produces this error:

```
Maximum value of 'back_log' cannot be set
```

4.2.2.5 Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime using the `SET` statement. See Section 13.7.6.1, “SET Syntax for Variable Assignment”, and Section 5.1.9, “Using System Variables”.

Most of these program variables also can be set at server startup by using the same syntax that applies to specifying program options. For example, `mysql` has a `max_allowed_packet` variable that controls the maximum size of its communication buffer. To set the `max_allowed_packet` variable for `mysql` to a value of 16MB, use either of the following commands:

```
mysql --max_allowed_packet=16777216
mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. For variables that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` to indicate a multiplier of 1024, 1024^2 or 1024^3. (For example, when used to set `max_allowed_packet`, the suffixes indicate units of kilobytes, megabytes, or gigabytes.) As of MySQL 8.0.14, a suffix can also be `T`, `P`, and `E` to indicate a multiplier of 1024^4, 1024^5 or 1024^6. Suffix letters can be uppercase or lowercase.

In an option file, variable settings are given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in a variable name can be specified as dashes. The following option groups are equivalent. Both set the size of the server's key buffer to 512MB:

```
[mysqld]
key_buffer_size=512M

[mysqld]
key-buffer-size=512M
```

A variable can be specified by writing it in full or as any unambiguous prefix. For example, the `max_allowed_packet` variable can be set for `mysql` as `--max_a`, but not as `--max` because the latter is ambiguous:

```
shell> mysql --max=1000000
mysql: ambiguous option '--max=1000000' (max_allowed_packet, max_join_size)
```

Be aware that the use of variable prefixes can cause problems in the event that new variables are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

Suffixes for specifying a value multiplier can be used when setting a variable at program invocation time, but not to set the value with `SET` at runtime. On the other hand, with `SET`, you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at program invocation time, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

4.2.2.6 Option Defaults, Options Expecting Values, and the = Sign

By convention, long forms of options that assign a value are written with an equals (`=`) sign, like this:

```
mysql --host=tonfisk --user=jon
```

For options that require a value (that is, not having a default value), the equal sign is not required, and so the following is also valid:

```
mysql --host tonfisk --user jon
```

In both cases, the `mysql` client attempts to connect to a MySQL server running on the host named “tonfisk” using an account with the user name “jon”.

Due to this behavior, problems can occasionally arise when no value is provided for an option that expects one. Consider the following example, where a user connects to a MySQL server running on host `tonfisk` as user `jon`:

```
shell> mysql --host 85.224.35.45 --user jon
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 8.0.19 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| jon@%          |
+----------------+
1 row in set (0.00 sec)
```

Omitting the required value for one of these option yields an error, such as the one shown here:

```
shell> mysql --host 85.224.35.45 --user
mysql: option '--user' requires an argument
```

In this case, `mysql` was unable to find a value following the `--user` option because nothing came after it on the command line. However, if you omit the value for an option that is *not* the last option to be used, you obtain a different error that you may not be expecting:

```
shell> mysql --host --user jon
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

Because `mysql` assumes that any string following `--host` on the command line is a host name, `--host --user` is interpreted as `--host=--user`, and the client attempts to connect to a MySQL server running on a host named “--user”.

Options having default values always require an equal sign when assigning a value; failing to do so causes an error. For example, the MySQL server `--log-error` option has the default value *`host_name`*`.err`, where *`host_name`* is the name of the host on which MySQL is running. Assume that you are running MySQL on a computer whose host name is “tonfisk”, and consider the following invocation of `mysqld_safe`:

```
shell> mysqld_safe &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

After shutting down the server, restart it as follows:

```
shell> mysqld_safe --log-error &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

The result is the same, since `--log-error` is not followed by anything else on the command line, and it supplies its own default value. (The `&` character tells the operating system to run MySQL in the background; it is ignored by MySQL itself.) Now suppose that you wish to log errors to a file named `my-`

`errors.err`. You might try starting the server with `--log-error my-errors`, but this does not have the intended effect, as shown here:

```
shell> mysqld_safe --log-error my-errors &
[1] 31357
shell> 080111 22:53:31 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080111 22:53:32 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended

[1]+  Done                    ./mysqld_safe --log-error my-errors
```

The server attempted to start using `/usr/local/mysql/var/tonfisk.err` as the error log, but then shut down. Examining the last few lines of this file shows the reason:

```
shell> tail /usr/local/mysql/var/tonfisk.err
2013-09-24T15:36:22.278034Z 0 [ERROR] Too many arguments (first extra is 'my-errors').
2013-09-24T15:36:22.278059Z 0 [Note] Use --verbose --help to get a list of available options!
2013-09-24T15:36:22.278076Z 0 [ERROR] Aborting
2013-09-24T15:36:22.279704Z 0 [Note] InnoDB: Starting shutdown...
2013-09-24T15:36:23.777471Z 0 [Note] InnoDB: Shutdown completed; log sequence number 2319086
2013-09-24T15:36:23.780134Z 0 [Note] mysqld: Shutdown complete
```

Because the `--log-error` option supplies a default value, you must use an equal sign to assign a different value to it, as shown here:

```
shell> mysqld_safe --log-error=my-errors &
[1] 31437
shell> 080111 22:54:15 mysqld_safe Logging to '/usr/local/mysql/var/my-errors.err'.
080111 22:54:15 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var

shell>
```

Now the server has been started successfully, and is logging errors to the file `/usr/local/mysql/var/my-errors.err`.

Similar issues can arise when specifying option values in option files. For example, consider a `my.cnf` file that contains the following:

```
[mysql]

host
user
```

When the `mysql` client reads this file, these entries are parsed as `--host --user` or `--host=--user`, with the result shown here:

```
shell> mysql
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

However, in option files, an equal sign is not assumed. Suppose the `my.cnf` file is as shown here:

```
[mysql]

user jon
```

Trying to start `mysql` in this case causes a different error:

```
shell> mysql
mysql: unknown option '--user jon'
```

A similar error would occur if you were to write `host tonfisk` in the option file rather than `host=tonfisk`. Instead, you must use the equal sign:

```
[mysql]

user=jon
```

Now the login attempt succeeds:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 8.0.19 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
1 row in set (0.00 sec)
```

This is not the same behavior as with the command line, where the equal sign is not required:

```
shell> mysql --user jon --host tonfisk
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 8.0.19 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@tonfisk   |
+---------------+
1 row in set (0.00 sec)
```

Specifying an option requiring a value without a value in an option file causes the server to abort with an error.

4.2.3 Command Options for Connecting to the Server

This section describes options supported by most MySQL client programs that control how client programs establish connections to the server, whether connections are encrypted, and whether connections are compressed. These options can be given on the command line or in an option file.

- Command Options for Connection Establishment
- Command Options for Encrypted Connections
- Command Options for Connection Compression

Command Options for Connection Establishment

This section describes options that control how client programs establish connections to the server. For additional information and examples showing how to use them, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

Table 4.3 Connection-Establishment Option Summary

Option Name	Description	Removed
--default-auth	Authentication plugin to use	
--host	Host on which MySQL server is located	
--password	Password to use when connecting to server	
--pipe	Connect to server using named pipe (Windows only)	
--plugin-dir	Directory where plugins are installed	
--port	TCP/IP port number for connection	
--protocol	Connection protocol to use	
--secure-auth	Do not send passwords to server in old (pre-4.1) format	8.0.3
--shared-memory-base-name	Name of shared memory to use for shared-memory connections	
--socket	Unix socket file or Windows named pipe to use	
--user	MySQL user name to use when connecting to server	

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--host=host_name`, `-h host_name`

  The host on which the MySQL server is running. The value can be a host name, IPv4 address, or IPv6 address. The default value is `localhost`.

- `--password[=pass_val]`, `-p[pass_val]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, the program prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

  To explicitly specify that there is no password and that the client program should not prompt for one, use the `--skip-password` option.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--plugin-dir=dir_name`

  The directory in which to look for plugins. Specify this option if the `--default-auth` option is used to specify an authentication plugin but the client program does not find it. See Section 6.2.17, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

For TCP/IP connections, the port number to use. The default port number is 3306.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  This option explicitly specifies which protocol to use for connecting to the server. It is useful when other connection parameters normally result in use of a protocol other than the one you want. For example, connections on Unix to `localhost` are made using a Unix socket file by default:

  ```
  mysql --host=localhost
  ```

  To force a TCP/IP connection to be used instead, specify a `--protocol` option:

  ```
  mysql --host=localhost --protocol=TCP
  ```

  The following table shows the permissible `--protocol` option values and indicates the platforms on which each value may be used. The values are not case-sensitive.

`--protocol` Value	**Connection Protocol**	**Permissible Operating Systems**
`TCP`	TCP/IP connection to local or remote server	All
`SOCKET`	Unix socket file connection to local server	Unix only
`PIPE`	Named-pipe connection to local or remote server	Windows only
`MEMORY`	Shared-memory connection to local server	Windows only

- `--secure-auth`

  This option was removed in MySQL 8.0.3.

- `--shared-memory-base-name=`*`name`*

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--socket=`*`path`*, `-S` *`path`*

  On Unix, the name of the Unix socket file to use for connections made using a named pipe to a local server. The default Unix socket file name is `/tmp/mysql.sock`.

  On Windows, the name of the named pipe to use for connections to a local server. The default Windows pipe name is `MySQL`. The pipe name is not case-sensitive.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The user name of the MySQL account to use for connecting to the server. The default user name is `ODBC` on Windows or your Unix login name on Unix.

Command Options for Encrypted Connections

This section describes options that specify whether to use encrypted connections, the names of certificate and key files, and other parameters related to encrypted-connection support. For examples of suggested use and how to check whether a connection is encrypted, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

For information about using encrypted connections from the MySQL C API, see Section 28.7.22, “C API Encrypted Connection Support”.

Table 4.4 Connection-Encryption Option Summary

Option Name	Description	Introduced
--get-server-public-key	Request RSA public key from server	8.0.3
--server-public-key-path	Path name to file containing RSA public key	
--skip-ssl	Disable connection encryption	
--ssl	Enable connection encryption	
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11
--ssl-key	File that contains X.509 key	
--ssl-mode	Desired security state of connection to server	
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16
--tls-version	Permissible TLS protocols for encrypted connections	

- `--get-server-public-key`

  Request from the server the public key required for RSA key pair-based password exchange. This option applies to clients that authenticate with the `caching_sha2_password` authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=`*`file_name`* is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  For information about the `caching_sha2_password` plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--server-public-key-path=`*`file_name`*

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-

based password exchange is not used, as is the case when the client connects to the server using a secure connection.

If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

This option is available only if MySQL was built using OpenSSL.

For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, "SHA-256 Pluggable Authentication", and Section 6.4.1.2, "Caching SHA-2 Pluggable Authentication".

- `--ssl`, `--skip-ssl`

  Note

  The client-side `--ssl` option is removed in MySQL 8.0. For client programs, use `--ssl-mode` instead.

  On the server side, the `--ssl` option specifies that the server permits but does not require encrypted connections. The option is enabled on the server side by default. `--ssl` is implied by other `--ssl-xxx` options, as indicated in the descriptions for those options.

  The `--ssl` option can be specified in negated form as `--skip-ssl` or a synonym (`--ssl=0`, `--disable-ssl`). In this case, the option specifies that the server does *not* permit encrypted connections and overrides other `--ssl-xxx` options.

  To specify additional parameters for encrypted connections, consider using at least `--ssl-cert` and `--ssl-key` on the server side and `--ssl-ca` on the client side. See Section 6.3.1, "Configuring MySQL to Use Encrypted Connections", which also describes server capabilities for certificate and key file autogeneration and autodiscovery.

- `--ssl-ca=file_name`

  The path name of the Certificate Authority (CA) certificate file in PEM format. On the server side, this option implies `--ssl`.

  To tell the client not to authenticate the server certificate when establishing an encrypted connection to the server, specify neither `--ssl-ca` nor `--ssl-capath`. The server still verifies the client according to any applicable requirements established for the client account, and it still uses any `--ssl-ca` or `--ssl-capath` option values specified on the server side.

- `--ssl-capath=dir_name`

  The path name of the directory that contains trusted SSL certificate authority (CA) certificate files in PEM format. On the server side, this option implies `--ssl`.

  To tell the client not to authenticate the server certificate when establishing an encrypted connection to the server, specify neither `--ssl-ca` nor `--ssl-capath`. The server still verifies the client according to any applicable requirements established for the client account, and it still uses any `--ssl-ca` or `--ssl-capath` option values specified on the server side.

- `--ssl-cert=file_name`

  The path name of the SSL public key certificate file in PEM format. On the client side, this is the client public key certificate. On the server side, this is the server public key certificate. On the server side, this option implies `--ssl`.

If the server is started with an `--ssl-cert` option specifying a certificate that uses any of the preceding restricted ciphers or cipher categories, the server starts with support for encrypted connections disabled.

- `--ssl-cipher=cipher_list`

  The list of permissible encryption ciphers for connections that use TLS protocols up through TLSv1.2. If no cipher in the list is supported, encrypted connections that use these TLS protocols will not work. On the server side, this option implies `--ssl`.

  For greatest portability, *`cipher_list`* should be a list of one or more cipher names, separated by colons. Examples:

  ```
  --ssl-cipher=AES128-SHA
  --ssl-cipher=DHE-RSA-AES128-GCM-SHA256:AES128-SHA
  ```

  OpenSSL supports the syntax for specifying ciphers described in the OpenSSL documentation at https://www.openssl.org/docs/manmaster/man1/ciphers.html.

  For information about which encryption ciphers MySQL supports, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `--ssl-crl=file_name`

  The path name of the file containing certificate revocation lists in PEM format. On the server side, this option implies `--ssl`.

  If neither `--ssl-crl` nor `--ssl-crlpath` is given, no CRL checks are performed, even if the CA path contains certificate revocation lists.

- `--ssl-crlpath=dir_name`

  The path name of the directory that contains certificate revocation-list files in PEM format. On the server side, this option implies `--ssl`.

  If neither `--ssl-crl` nor `--ssl-crlpath` is given, no CRL checks are performed, even if the CA path contains certificate revocation lists.

- `--ssl-fips-mode={OFF|ON|STRICT}`

  Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These `--ssl-fips-mode` values are permissible:

  - `OFF`: Disable FIPS mode.
  - `ON`: Enable FIPS mode.
  - `STRICT`: Enable “strict” FIPS mode.

  Note

  If the OpenSSL FIPS Object Module is not available, the only permissible value for `--ssl-fips-mode` is `OFF`. In this case, setting `--ssl-fips-mode` to `ON` or

> `STRICT` causes the client to produce a warning at startup and to operate in non-FIPS mode.

To specify the FIPS mode for the server, use the `ssl_fips_mode` system variable.

- `--ssl-key=file_name`

  The path name of the SSL private key file in PEM format. On the client side, this is the client private key. On the server side, this is the server private key. On the server side, this option implies `--ssl`.

  If the key file is protected by a passphrase, the program prompts the user for the passphrase. The password must be given interactively; it cannot be stored in a file. If the passphrase is incorrect, the program continues as if it could not read the key.

  For better security, use a certificate with an RSA key size of at least 2048 bits.

- `--ssl-mode=mode`

  This option is available only for client programs, not the server. It specifies the desired security state of the connection to the server. These mode values are permissible, in order of increasing strictness:

  - `DISABLED`: Establish an unencrypted connection.
  - `PREFERRED`: Establish an encrypted connection if the server supports encrypted connections, falling back to an unencrypted connection if an encrypted connection cannot be established. This is the default if `--ssl-mode` is not specified.

    Encrypted connections over Unix sockets are disabled by default, so `PREFERRED` does not establish an encrypted connection. To enforce encryption for Unix socket connections, use a mode of `REQUIRED` or stricter.
  - `REQUIRED`: Establish an encrypted connection if the server supports encrypted connections. The connection attempt fails if an encrypted connection cannot be established.
  - `VERIFY_CA`: Like `REQUIRED`, but additionally verify the server Certificate Authority (CA) certificate against the configured CA certificates. The connection attempt fails if no valid matching CA certificates are found.
  - `VERIFY_IDENTITY`: Like `VERIFY_CA`, but additionally perform host name identity verification by checking the host name the client uses for connecting to the server against the identity in the certificate that the server sends to the client:
    - As of MySQL 8.0.12, if the client uses OpenSSL 1.0.2 or higher, the client checks whether the host name that it uses for connecting matches either the Subject Alternative Name value or the Common Name value in the server certificate.
    - Otherwise, the client checks whether the host name that it uses for connecting matches the Common Name value in the server certificate.

    The connection fails if there is a mismatch. For encrypted connections, this option helps prevent man-in-the-middle attacks.

> **Note**
>
> Host name identity verification with `VERIFY_IDENTITY` does not work with self-signed certificates that are created automatically by the server or manually using `mysql_ssl_rsa_setup` (see Section 6.3.3.1, “Creating SSL and RSA

> Certificates and Keys using MySQL”). Such self-signed certificates do not contain the server name as the Common Name value.
>
> Host name identity verification also does not work with certificates that specify the Common Name using wildcards because that name is compared verbatim to the server name.

The `--ssl-mode` option interacts with CA certificate options as follows:

- If `--ssl-mode` is not explicitly set otherwise, use of `--ssl-ca` or `--ssl-capath` implies `--ssl-mode=VERIFY_CA`.
- For `--ssl-mode` values of `VERIFY_CA` or `VERIFY_IDENTITY`, `--ssl-ca` or `--ssl-capath` is also required, to supply a CA certificate that matches the one used by the server.
- An explicit `--ssl-mode` option with a value other than `VERIFY_CA` or `VERIFY_IDENTITY`, together with an explicit `--ssl-ca` or `--ssl-capath` option, produces a warning that no verification of the server certificate will be done, despite a CA certificate option being specified.

To require use of encrypted connections by a MySQL account, use `CREATE USER` to create the account with a `REQUIRE SSL` clause, or use `ALTER USER` for an existing account to add a `REQUIRE SSL` clause. This causes connection attempts by clients that use the account to be rejected unless MySQL supports encrypted connections and an encrypted connection can be established.

The `REQUIRE` clause permits other encryption-related options, which can be used to enforce security requirements stricter than `REQUIRE SSL`. For additional details about which command options may or must be specified by clients that connect using accounts configured using the various `REQUIRE` options, see CREATE USER SSL/TLS Options.

- `--tls-ciphersuites=`*`ciphersuite_list`*

  For client programs, which ciphersuites the client permits for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. For example:

  ```
  mysql --tls-ciphersuites="suite1:suite2:suite3"
  ```

  The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

  To specify which ciphersuites the server permits, use the `tls_ciphersuites` system variable.

- `--tls-version=`*`protocol_list`*

  For client programs, the TLS protocols the client permits for encrypted connections. The value is a list of one or more comma-separated protocol versions. For example:

  ```
  mysql --tls-version="TLSv1.1,TLSv1.2"
  ```

  The protocols that can be named for this option depend on the SSL library used to compile MySQL. Permitted protocols should be chosen such as not to leave “holes” in the list. For example, these values do not have holes:

  ```
  --tls-version="TLSv1,TLSv1.1,TLSv1.2,TLSv1.3"
  ```

```
--tls-version="TLSv1.1,TLSv1.2,TLSv1.3"
--tls-version="TLSv1.2,TLSv1.3"
--tls-version="TLSv1.3"
```

These values do have holes and should not be used:

```
--tls-version="TLSv1,TLSv1.2"
--tls-version="TLSv1.1,TLSv1.3"
```

For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

To specify which TLS protocols the server permits, use the `tls_version` system variable.

Command Options for Connection Compression

This section describes options that enable client programs to control use of compression for connections to the server. For additional information and examples showing how to use them, see Section 4.2.6, “Connection Compression Control”.

Table 4.5 Connection-Compression Option Summary

Option Name	Description	Introduced	Deprecated
--compress	Compress all information sent between client and server		8.0.18
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18	
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18	

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible.

  As of MySQL 8.0.18, this option is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `--compression-algorithms=`*`value`*

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. The default value is `uncompressed`.

  This option was added in MySQL 8.0.18.

- `--zstd-compression-level=`*`level`*

  The compression level to use for connections to the server that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. The compression level setting has no effect on connections that do not use `zstd` compression.

  This option was added in MySQL 8.0.18.

4.2.4 Connecting to the MySQL Server Using Command Options

This section describes use of command-line options to specify how to establish connections to the MySQL server, for clients such as `mysql` or `mysqldump`. For information on establishing connections using URI-

like connection strings or key-value pairs, for clients such as MySQL Shell, see Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”. For additional information if you are unable to connect, see Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”.

For a client program to connect to the MySQL server, it must use the proper connection parameters, such as the name of the host where the server is running and the user name and password of your MySQL account. Each connection parameter has a default value, but you can override default values as necessary using program options specified either on the command line or in an option file.

The examples here use the `mysql` client program, but the principles apply to other clients such as `mysqldump`, `mysqladmin`, or `mysqlshow`.

This command invokes `mysql` without specifying any explicit connection parameters:

```
mysql
```

Because there are no parameter options, the default values apply:

- The default host name is `localhost`. On Unix, this has a special meaning, as described later.
- The default user name is `ODBC` on Windows or your Unix login name on Unix.
- No password is sent because neither `--password` nor `-p` is given.
- For `mysql`, the first nonoption argument is taken as the name of the default database. Because there is no such argument, `mysql` selects no default database.

To specify the host name and user name explicitly, as well as a password, supply appropriate options on the command line. To select a default database, add a database-name argument. Examples:

```
mysql --host=localhost --user=myname --password=password mydb
mysql -h localhost -u myname -ppassword mydb
```

For password options, the password value is optional:

- If you use a `--password` or `-p` option and specify a password value, there must be *no space* between `--password=` or `-p` and the password following it.
- If you use `--password` or `-p` but do not specify a password value, the client program prompts you to enter the password. The password is not displayed as you enter it. This is more secure than giving the password on the command line, which might enable other users on your system to see the password line by executing a command such as `ps`. See Section 6.1.2.1, “End-User Guidelines for Password Security”.
- To explicitly specify that there is no password and that the client program should not prompt for one, use the `--skip-password` option.

As just mentioned, including the password value on the command line can be a security risk. To avoid this risk, specify the `--password` or `-p` option without any following password value:

```
mysql --host=localhost --user=myname --password mydb
mysql -h localhost -u myname -p mydb
```

When the `--password` or `-p` option is given with no password value, the client program prints a prompt and waits for you to enter the password. (In these examples, `mydb` is *not* interpreted as a password because it is separated from the preceding password option by a space.)

On some systems, the library routine that MySQL uses to prompt for a password automatically limits the password to eight characters. That limitation is a property of the system library, not MySQL. Internally,

MySQL does not have any limit for the length of the password. To work around the limitation on systems affected by it, specify your password in an option file (see Section 4.2.2.2, “Using Option Files”). Another workaround is to change your MySQL password to a value that has eight or fewer characters, but that may be unacceptable because shorter passwords tend to be less secure.

Client programs determine what type of connection to make as follows:

- If the host is not specified or is `localhost`, a connection to the local host occurs:
  - On Windows, the client connects using a shared-memory connection, if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.
  - On Unix, MySQL programs treat the host name `localhost` specially, in a way that is likely different from what you expect compared to other network-based programs: the client connects using a Unix socket file. The `--socket` option or the `MYSQL_UNIX_PORT` environment variable may be used to specify the socket name.
- On Windows, if `host` is `.` (period), or TCP/IP is not enabled and `--socket` is not specified or the host is empty, the client connects using a named pipe, if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. If named-pipe connections are not supported or if the user making the connection is not a member of the Windows group specified by the `named_pipe_full_access_group` system variable, an error occurs.
- Otherwise, the connection uses TCP/IP.

The `--protocol` option enables you to establish a particular type of connection even when other options normally result in use of a different protocol. That is, `--protocol` specifies the connection protocol explicitly and overrides the preceding rules, even for `localhost`.

Only connection options that are relevant to the selected protocol are used or checked. Other connection options are ignored. For example, with `--host=localhost` on Unix, the client attempts to connect to the local server using a Unix socket file. This occurs even if a `--port` or `-P` option is given to specify a port number.

To ensure that the client makes a TCP/IP connection to the local server, use `--host` or `-h` to specify a host name value of `127.0.0.1`, or the IP address or name of the local server. You can also specify the connection protocol explicitly, even for `localhost`, by using the `--protocol=TCP` option. Examples:

```
mysql --host=127.0.0.1
mysql --protocol=TCP
```

If the server is configured to accept IPv6 connections, clients can connect to the local server over IPv6 using `--host=::1`. See Section 5.1.12, “IPv6 Support”.

On Windows, you can force a MySQL client to use a named-pipe connection by specifying the `--pipe` or `--protocol=PIPE` option, or by specifying `.` (period) as the host name. If the server was not started with the `named_pipe` system variable enabled to support named-pipe connections or if the user making the connection is not a member of the Windows group specified by the `named_pipe_full_access_group` system variable, an error occurs. Use the `--socket` option to specify the name of the pipe if you do not want to use the default pipe name.

Connections to remote servers use TCP/IP. This command connects to the server running on `remote.example.com` using the default port number (3306):

```
mysql --host=remote.example.com
```

To specify a port number explicitly, use the `--port` or `-P` option:

```
mysql --host=remote.example.com --port=13306
```

You can specify a port number for connections to a local server, too. However, as indicated previously, connections to `localhost` on Unix use a socket file by default, so unless you force a TCP/IP connection as previously described, any option that specifies a port number is ignored.

For this command, the program uses a socket file on Unix and the `--port` option is ignored:

```
mysql --port=13306 --host=localhost
```

To cause the port number to be used, force a TCP/IP connection. For example, invoke the program in either of these ways:

```
mysql --port=13306 --host=127.0.0.1
mysql --port=13306 --protocol=TCP
```

For additional information about options that control how client programs establish connections to the server, see Section 4.2.3, “Command Options for Connecting to the Server”.

It is possible to specify connection parameters without entering them on the command line each time you invoke a client program:

- Specify the connection parameters in the `[client]` section of an option file. The relevant section of the file might look like this:

  ```
  [client]
  host=host_name
  user=user_name
  password=password
  ```

  For more information, see Section 4.2.2.2, “Using Option Files”.

- Some connection parameters can be specified using environment variables. Examples:

  - To specify the host for `mysql`, use `MYSQL_HOST`.
  - On Windows, to specify the MySQL user name, use `USER`.

  For a list of supported environment variables, see Section 4.9, “MySQL Environment Variables”.

4.2.5 Connecting to the Server Using URI-Like Strings or Key-Value Pairs

This section describes use of URI-like connection strings or key-value pairs to specify how to establish connections to the MySQL server, for clients such as MySQL Shell. For information on establishing connections using command-line options, for clients such as `mysql` or `mysqldump`, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”. For additional information if you are unable to connect, see Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”.

Note

The term “URI-like” signifies connection-string syntax that is similar to but not identical to the URI (uniform resource identifier) syntax defined by RFC 3986.

The following MySQL clients support connecting to a MySQL server using a URI-like connection string or key-value pairs:

- MySQL Shell
- MySQL Connectors which implement X DevAPI

This section documents all valid URI-like string and key-value pair connection parameters, many of which are similar to those specified with command-line options:

- Parameters specified with a URI-like string use a syntax such as `myuser@example.com:3306/main-schema`. For the full syntax, see Connecting Using URI-Like Connection Strings.
- Parameters specified with key-value pairs use a syntax such as `{user:'myuser', host:'example.com', port:3306, schema:'main-schema'}`. For the full syntax, see Connecting Using Key-Value Pairs.

Connection parameters are not case-sensitive. Each parameter, if specified, can be given only once. If a parameter is specified more than once, an error occurs.

This section covers the following topics:

- Base Connection Parameters
- Additional Connection parameters
- Connecting Using URI-Like Connection Strings
- Connecting Using Key-Value Pairs

Base Connection Parameters

The following discussion describes the parameters available when specifying a connection to MySQL. These parameters can be provided using either a string that conforms to the base URI-like syntax (see Connecting Using URI-Like Connection Strings), or as key-value pairs (see Connecting Using Key-Value Pairs).

- *`scheme`*: The connection protocol to use. Use `mysqlx` for X Protocol connections and `mysql` for classic MySQL protocol connections (see MySQL Shell Sessions Explained). If no protocol is specified, the server attempts to guess the protocol.
- *`user`*: The MySQL user account to use for the authentication process.
- *`password`*: The password to use for the authentication process.

Warning

Specifying an explicit password in the connection specification is insecure and not recommended. Later discussion shows how to cause an interactive prompt for the password to occur.

- *`host`*: The host on which the server instance is running. The value can be a host name, IPv4 address, or IPv6 address. If no host is specified, the default is `localhost`.
- *`port`*: The TCP/IP network port on which the target MySQL server is listening for connections. If no part is specified, the default is 33060 for X Protocol connections and 3306 for classic MySQL protocol connections.
- *`socket`*: The path to a Unix socket file or the name of a Windows named pipe. Values are local file paths. In URI-like strings, they must be encoded, using either percent encoding or by surrounding the path with parentheses. Parentheses eliminate the need to percent encode characters such as the `/`

directory separator character. For example, to connect as `root@localhost` using the Unix socket `/tmp/mysql.sock`, specify the path using percent encoding as `root@localhost?socket=%2Ftmp%2Fmysql.sock`, or using parentheses as `root@localhost?socket=(/tmp/mysql.sock)`.

- *`schema`*: The default database for the connection. If no database is specified, the connection has no default database.

Additional Connection parameters

You can specify options for the connection, either as attributes in a URI-like string by appending `?attribute=value`, or as key-value pairs. The following options are available:

- `ssl-mode`: The desired security state for the connection. The following modes are permissible:
  - `DISABLED`
  - `PREFERRED`
  - `REQUIRED`
  - `VERIFY_CA`
  - `VERIFY_IDENTITY`

  For information about these modes, see the `--ssl-mode` option description in Command Options for Encrypted Connections.
- `ssl-ca`: The path to the X.509 certificate authority file in PEM format.
- `ssl-capath`: The path to the directory that contains the X.509 certificates authority files in PEM format.
- `ssl-cert`: The path to the X.509 certificate file in PEM format.
- `ssl-cipher`: The encryption cipher to use for connections that use TLS protocols up through TLSv1.2.
- `ssl-crl`: The path to the file that contains certificate revocation lists in PEM format.
- `ssl-crlpath`: The path to the directory that contains certificate revocation-list files in PEM format.
- `ssl-key`: The path to the X.509 key file in PEM format.
- `tls-version`: The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol versions. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”. The following versions are permissible:
  - `TLSv1`
  - `TLSv1.1`
  - `TLSv1.2` (Supported only by commercial edition)
- `auth-method`: The authentication method to use for the connection. The default is `AUTO`, meaning that the server attempts to guess. The following methods are permissible:
  - `AUTO`
  - `MYSQL41`
  - `SHA256_MEMORY`

  - `FROM_CAPABILITIES`
  - `FALLBACK`
  - `PLAIN`

  For X Protocol connections, any configured `auth-method` is overridden to this sequence of authentication methods: `MYSQL41`, `SHA256_MEMORY`, `PLAIN`.

- `get-server-public-key`: Request from the server the public key required for RSA key pair-based password exchange. Use when connecting to MySQL 8.0 servers over classic MySQL protocol with SSL mode `DISABLED`. You must specify the protocol in this case. For example:

  ```
  mysql://user@localhost:3306?get-server-public-key=true
  ```

  This option applies to clients that authenticate with the `caching_sha2_password` authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `server-public-key-path=`*`file_name`* is given and specifies a valid public key file, it takes precedence over `get-server-public-key`.

  For information about the `caching_sha2_password` plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `server-public-key-path`: The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. Use when connecting to MySQL 8.0 servers over classic MySQL protocol with SSL mode `DISABLED`.

  The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `server-public-key-path=`*`file_name`* is given and specifies a valid public key file, it takes precedence over `get-server-public-key`.

  For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `connect-timeout`: An integer value used to configure the number of seconds that clients, such as MySQL Shell, wait until they stop trying to connect to an unresponsive MySQL server.

- `compression`: When set to `true` (or 1), this option enables compression of all information sent between the client and the server if possible. The default is no compression (`false` or 0). This option is available for MySQL Shell connections using classic MySQL protocol only. See Section 4.2.6, “Connection Compression Control”.

- `connection-attributes`: Controls the key-value pairs that application programs pass to the server at connect time. For general information about connection attributes, see Section 26.12.9, “Performance Schema Connection Attribute Tables”. Clients usually define a default set of attributes, which can be disabled or enabled. For example:

  ```
  ```

```
mysqlx://user@host?connection-attributes
mysqlx://user@host?connection-attributes=true
mysqlx://user@host?connection-attributes=false
```

The default behavior is to send the default attribute set. Applications can specify attributes to be passed in addition to the default attributes. You specify additional connection attributes as a `connection-attributes` parameter in a connection string. The `connection-attributes` parameter value must be empty (the same as specifying `true`), a `Boolean` value (`true` or `false` to enable or disable the default attribute set), or a list or zero or more `key=value` specifiers separated by commas (to be sent in addition to the default attribute set). Within a list, a missing key value evaluates as an empty string. Further examples:

```
mysqlx://user@host?connection-attributes=[attr1=val1,attr2,attr3=]
mysqlx://user@host?connection-attributes=[]
```

Application-defined attribute names cannot begin with `_` because such names are reserved for internal attributes.

Connecting Using URI-Like Connection Strings

You can specify a connection to MySQL Server using a URI-like string. Such strings can be used with the MySQL Shell with the `--uri` command option, the MySQL Shell `\connect` command, and MySQL Connectors which implement X DevAPI.

Note

The term “URI-like” signifies connection-string syntax that is similar to but not identical to the URI (uniform resource identifier) syntax defined by RFC 3986.

A URI-like connection string has the following syntax:

```
[scheme://][user[:[password]]@]host[:port][/schema][?attribute1=value1&attribute2=value2...
```

Important

Percent encoding must be used for reserved characters in the elements of the URI-like string. For example, if you specify a string that includes the `@` character, the character must be replaced by `%40`. If you include a zone ID in an IPv6 address, the `%` character used as the separator must be replaced with `%25`.

The parameters you can use in a URI-like connection string are described at Base Connection Parameters.

MySQL Shell's `shell.parseUri()` and `shell.unparseUri()` methods can be used to deconstruct and assemble a URI-like connection string. Given a URI-like connection string, `shell.parseUri()` returns a dictionary containing each element found in the string. `shell.unparseUri()` converts a dictionary of URI components and connection options into a valid URI-like connection string for connecting to MySQL, which can be used in MySQL Shell or by MySQL Connectors which implement X DevAPI.

If no password is specified in the URI-like string, which is recommended, interactive clients prompt for the password. The following examples show how to specify URI-like strings with the user name *`user_name`*. In each case, the password is prompted for.

- An X Protocol connection to a local server instance listening at port 33065.

```
mysqlx://user_name@localhost:33065
```

- A classic MySQL protocol connection to a local server instance listening at port 3333.

```
mysql://user_name@localhost:3333
```

- An X Protocol connection to a remote server instance, using a host name, an IPv4 address, and an IPv6 address.

```
mysqlx://user_name@server.example.com/
mysqlx://user_name@198.51.100.14:123
mysqlx://user_name@[2001:db8:85a3:8d3:1319:8a2e:370:7348]
```

- An X Protocol connection using a socket, with the path provided using either percent encoding or parentheses.

```
mysqlx://user_name@/path%2Fto%2Fsocket.sock
mysqlx://user_name@(/path/to/socket.sock)
```

- An optional path can be specified, which represents a database.

```
# use 'world' as the default database
mysqlx://user_name@198.51.100.1/world

# use 'world_x' as the default database, encoding _ as %5F
mysqlx://user_name@198.51.100.2:33060/world%5Fx
```

- An optional query can be specified, consisting of values each given as a *key=value* pair or as a single *key*. To specify multiple values, separate them by `,` characters. A mix of *key=value* and *key* values is permissible. Values can be of type list, with list values ordered by appearance. Strings must be either percent encoded or surrounded by parentheses. The following are equivalent.

```
ssluser@127.0.0.1?ssl-ca=%2Froot%2Fclientcert%2Fca-cert.pem\
&ssl-cert=%2Froot%2Fclientcert%2Fclient-cert.pem\
&ssl-key=%2Froot%2Fclientcert%2Fclient-key

ssluser@127.0.0.1?ssl-ca=(/root/clientcert/ca-cert.pem)\
&ssl-cert=(/root/clientcert/client-cert.pem)\
&ssl-key=(/root/clientcert/client-key)
```

The previous examples assume that connections require a password. With interactive clients, the specified user's password is requested at the login prompt. If the user account has no password (which is insecure and not recommended), or if socket peer-credential authentication is in use (for example, with Unix socket connections), you must explicitly specify in the connection string that no password is being provided and the password prompt is not required. To do this, place a `:` after the *user_name* in the string but do not specify a password after it. For example:

```
mysqlx://user_name:@localhost
```

Connecting Using Key-Value Pairs

In MySQL Shell and some MySQL Connectors which implement X DevAPI, you can specify a connection to MySQL Server using key-value pairs, supplied in language-natural constructs for the implementation. For example, you can supply connection parameters using key-value pairs as a JSON object in JavaScript, or as a dictionary in Python. Regardless of the way the key-value pairs are supplied, the concept remains the same: the keys as described in this section can be assigned values that are used to specify a connection. You can specify connections using key-value pairs in MySQL Shell's `shell.connect()` method or InnoDB cluster's `dba.createCluster()` method, and with some of the MySQL Connectors which implement X DevAPI.

Generally, key-value pairs are surrounded by `{` and `}` characters and the `,` character is used as a separator between key-value pairs. The `:` character is used between keys and values, and strings must be delimited (for example, using the `'` character). It is not necessary to percent encode strings, unlike URI-like connection strings.

A connection specified as key-value pairs has the following format:

```
{ key: value, key: value, ...}
```

The parameters you can use as keys for a connection are described at Base Connection Parameters.

If no password is specified in the key-value pairs, which is recommended, interactive clients prompt for the password. The following examples show how to specify connections using key-value pairs with the user name `'user_name'`. In each case, the password is prompted for.

- An X Protocol connection to a local server instance listening at port 33065.

  ```
  {user:'user_name', host:'localhost', port:33065}
  ```

- A classic MySQL protocol connection to a local server instance listening at port 3333.

  ```
  {user:'user_name', host:'localhost', port:3333}
  ```

- An X Protocol connection to a remote server instance, using a host name, an IPv4 address, and an IPv6 address.

  ```
  {user:'user_name', host:'server.example.com'}
  {user:'user_name', host:198.51.100.14:123}
  {user:'user_name', host:[2001:db8:85a3:8d3:1319:8a2e:370:7348]}
  ```

- An X Protocol connection using a socket.

  ```
  {user:'user_name', socket:'/path/to/socket/file'}
  ```

- An optional schema can be specified, which represents a database.

  ```
  {user:'user_name', host:'localhost', schema:'world'}
  ```

The previous examples assume that connections require a password. With interactive clients, the specified user's password is requested at the login prompt. If the user account has no password (which is insecure and not recommended), or if socket peer-credential authentication is in use (for example, with Unix socket connections), you must explicitly specify that no password is being provided and the password prompt is not required. To do this, provide an empty string using `''` after the `password` key. For example:

```
{user:'user_name', password:'', host:'localhost'}
```

4.2.6 Connection Compression Control

Connections to the server can use a compressed protocol that reduces the number of bytes sent over the connection. By default, connections are uncompressed, but can be compressed if the server and the client agree on the compression algorithm to use.

This section describes the available compression-control configuration parameters and the information sources available for monitoring use of compression.

Compression control applies to connections to the server by client programs and by servers participating in master/slave replication or Group Replication. (In the following discussion, “client connection” is shorthand for a connection to the server originating from any source for which compression is supported, unless context indicates a specific connection type.) Compression control does not apply to X Protocol connections or connections for `FEDERATED` tables.

Compressed connections originate on the client side but affect CPU load on both the client and server sides because both sides perform compression and decompression operations. Because enabling compression decreases performance, its benefits occur primarily when there is low network bandwidth, network transfer time dominates the cost of compression and decompression operations, and result sets are large.

- Connection Compression Configuration
- Legacy Connection Compression Configuration
- Connection Compression Monitoring

Connection Compression Configuration

As of MySQL 8.0.18, these configuration parameters are available for controlling connection compression:

- The `protocol_compression_algorithms` system variable configures which compression algorithms the server permits for incoming connections.
- The `--compression-algorithms` and `--zstd-compression-level` command-line options configure permitted compression algorithms and `zstd` compression level for these client programs: `mysql`, `mysqladmin`, `mysqlbinlog`, `mysqlcheck`, `mysqldump`, `mysqlimport`, `mysqlpump`, `mysqlshow`, `mysqlslap`, and `mysqltest`, and `mysql_upgrade`.
- The `MYSQL_OPT_COMPRESSION_ALGORITHMS` and `MYSQL_OPT_ZSTD_COMPRESSION_LEVEL` options for the `mysql_options()` function configure permitted compression algorithms and `zstd` compression level for client programs that use the MySQL C API.
- The `MASTER_COMPRESSION_ALGORITHMS` and `MASTER_ZSTD_COMPRESSION_LEVEL` options for the `CHANGE MASTER TO` statement configure permitted compression algorithms and `zstd` compression level for slave servers participating in master/slave replication.
- The `group_replication_recovery_compression_algorithm` and `group_replication_recovery_zstd_compression_level` system variables configure permitted compression algorithms and `zstd` compression level for Group Replication recovery connections when a new member joins a group and connects to a donor.

Configuration parameters that enable specifying compression algorithms are string-valued and take a list of one or more comma-separated compression algorithm names, in any order, chosen from the following items (not case-sensitive):

- `zlib`: Permit connections that use the `zlib` compression algorithm.
- `zstd`: Permit connections that use the `zstd` compression algorithm (zstd 1.3).
- `uncompressed`: Permit uncompressed connections.

Note

Because `uncompressed` is an algorithm name that may or may not be configured, it is possible to configure MySQL *not* to support uncompressed connections.

Examples:

- To configure which compression algorithms the server permits for incoming connections, set the `protocol_compression_algorithms` system variable. By default, the server permits all available algorithms. To configure that setting explicitly at startup, use these lines in the server `my.cnf` file:

  ```
  [mysqld]
  protocol_compression_algorithms=zlib,zstd,uncompressed
  ```

  To set and persist the `protocol_compression_algorithms` system variable to that value at runtime, use this statement:

  ```
  SET PERSIST protocol_compression_algorithms='zlib,zstd,uncompressed';
  ```

  `SET PERSIST` sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. To change the value for the running MySQL instance without saving it for subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

- To permit only incoming connections that use `zstd` compression, configure the server at startup like this:

  ```
  [mysqld]
  protocol_compression_algorithms=zstd
  ```

  Or, to make the change at runtime:

  ```
  SET PERSIST protocol_compression_algorithms='zstd';
  ```

- To permit the `mysql` client to initiate `zlib` or `uncompressed` connections, invoke it like this:

  ```
  mysql --compression-algorithms=zlib,uncompressed
  ```

- To configure replication slaves to connect to the master using `zlib` or `zstd` connections, with a compression level of 7 for `zstd` connections, use a `CHANGE MASTER TO` statement:

  ```
  CHANGE MASTER TO
    MASTER_COMPRESSION_ALGORITHMS = 'zlib,zstd',
    MASTER_ZSTD_COMPRESSION_LEVEL = 7;
  ```

  This assumes that the `slave_compressed_protocol` system variable is disabled, for reasons described in Legacy Connection Compression Configuration.

For successful connection setup, both sides of the connection must agree on which compression algorithm to use. The algorithm-negotiation process attempts to use `zlib`, then `zstd`, then `uncompressed`. If the two sides can find no common algorithm, the connection attempt fails.

Because both sides must agree on the compression algorithm, and because `uncompressed` is an algorithm value that need not be supported, fallback to an uncompressed connection does not necessarily occur. For example, if the server is configured to permit `zstd` and a client is configured to permit `zlib,uncompressed`, the client cannot connect at all. In this case, no algorithm is common to both sides, so connection attempts fail.

Configuration parameters that enable specifying the `zstd` compression level take an integer value from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. The compression level setting has no effect on connections that do not use `zstd` compression.

A configurable `zstd` compression level enables choosing between less network traffic and higher CPU load versus more network traffic and lower CPU load. Higher compression levels reduce network congestion but the additional CPU load may reduce server performance.

Legacy Connection Compression Configuration

Prior to MySQL 8.0.18, these configuration parameters are available for controlling connection compression:

- Client programs support a `--compress` command-line option to specify use of compression for the connection to the server.
- MySQL Shell supports compression for connections that use classic MySQL protocol. It provides a `--compress` command-line option, a `compression` parameter for URI-like string and key-value pair connection specifications, and a `defaultCompress` configuration option. (These are available as of MySQL Shell 8.0.14.)
- For programs that use the MySQL C API, enabling the `MYSQL_OPT_COMPRESS` option for the `mysql_options()` function specifies use of compression for the connection to the server.
- For master/server replication, enabling the `slave_compressed_protocol` system variable specifies use of compression for slave connections to the master.

In each case, when use of compression is specified, the connection uses the `zlib` compression algorithm if both sides support it, with fallback to an uncompressed connection otherwise.

As of MySQL 8.0.18, the compression parameters just described become legacy parameters, due to the additional compression parameters introduced for more control over connection compression that are described in Connection Compression Configuration. The legacy compression parameters interact with the newer parameters and their semantics change as follows:

- The meaning of the legacy `--compress` option depends on whether `--compression-algorithms` is specified:
  - When `--compression-algorithms` is not specified, `--compress` is equivalent to specifying a client-side algorithm set of `zlib,uncompressed`.
  - When `--compression-algorithms` is specified, `--compress` is equivalent to specifying an algorithm set of `zlib` and the full client-side algorithm set is the union of `zlib` plus the algorithms specified by `--compression-algorithms`. For example, with both `--compress` and `--compression-algorithms=zlib,zstd`, the permitted-algorithm set is `zlib` plus `zlib,zstd`; that is, `zlib,zstd`. With both `--compress` and `--compression-algorithms=zstd,uncompressed`, the permitted-algorithm set is `zlib` plus `zstd,uncompressed`; that is, `zlib,zstd,uncompressed`.
- The same type of interaction occurs between the legacy `MYSQL_OPT_COMPRESS` option and the `MYSQL_OPT_COMPRESSION_ALGORITHMS` option for the `mysql_options()` C API function.
- If the `slave_compressed_protocol` system variable is enabled, it takes precedence over `MASTER_COMPRESSION_ALGORITHMS` and connections to the master use `zlib` compression if both master and slave support that algorithm. If `slave_compressed_protocol` is disabled, the value of `MASTER_COMPRESSION_ALGORITHMS` applies.

Note

The legacy compression-control parameters are deprecated as of MySQL 8.0.18 and will be removed in a future MySQL version.

Connection Compression Monitoring

The `Compression` status variable is `ON` or `OFF` to indicate whether the current connection uses compression.

The `mysql` client `\status` command displays a line that says `Protocol: Compressed` if compression is enabled for the current connection. If that line is not present, the connection is uncompressed.

As of 8.0.14, the MySQL Shell `\status` command displays a `Compression:` line that says `Disabled` or `Enabled` to indicate whether the connection is compressed.

As of MySQL 8.0.18, these additional sources of information are available for monitoring connection compression:

- To monitor compression in use for client connections, use the `Compression_algorithm` and `Compression_level` status variables. For the current connection, their values indicate the compression algorithm and compression level, respectively.
- To determine which compression algorithms the server is configured to permit for incoming connections, check the `protocol_compression_algorithms` system variable.
- For master/slave replication connections, the configured compression algorithms and compression level are available from multiple sources:
  - The Performance Schema `replication_connection_configuration` table has `COMPRESSION_ALGORITHMS` and `ZSTD_COMPRESSION_LEVEL` columns.
  - The `mysql.slave_master_info` system table has `Master_compression_algorithms` and `Master_zstd_compression_level` columns. If the `master.info` file exists, it contains lines for those values as well.

4.2.7 Setting Environment Variables

Environment variables can be set at the command prompt to affect the current invocation of your command processor, or set permanently to affect future invocations. To set a variable permanently, you can set it in a startup file or by using the interface provided by your system for this purpose. Consult the documentation for your command interpreter for specific details. Section 4.9, “MySQL Environment Variables”, lists all environment variables that affect MySQL program operation.

To specify a value for an environment variable, use the syntax appropriate for your command processor. For example, on Windows, you can set the `USER` variable to specify your MySQL account name. To do so, use this syntax:

```
SET USER=your_name
```

The syntax on Unix depends on your shell. Suppose that you want to specify the TCP/IP port number using the `MYSQL_TCP_PORT` variable. Typical syntax (such as for `sh`, `ksh`, `bash`, `zsh`, and so on) is as follows:

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

The first command sets the variable, and the `export` command exports the variable to the shell environment so that its value becomes accessible to MySQL and other processes.

For `csh` and `tcsh`, use `setenv` to make the shell variable available to the environment:

```
```

```
setenv MYSQL_TCP_PORT 3306
```

The commands to set environment variables can be executed at your command prompt to take effect immediately, but the settings persist only until you log out. To have the settings take effect each time you log in, use the interface provided by your system or place the appropriate command or commands in a startup file that your command interpreter reads each time it starts.

On Windows, you can set environment variables using the System Control Panel (under Advanced).

On Unix, typical shell startup files are `.bashrc` or `.bash_profile` for `bash`, or `.tcshrc` for `tcsh`.

Suppose that your MySQL programs are installed in `/usr/local/mysql/bin` and that you want to make it easy to invoke these programs. To do this, set the value of the `PATH` environment variable to include that directory. For example, if your shell is `bash`, add the following line to your `.bashrc` file:

```
PATH=${PATH}:/usr/local/mysql/bin
```

`bash` uses different startup files for login and nonlogin shells, so you might want to add the setting to `.bashrc` for login shells and to `.bash_profile` for nonlogin shells to make sure that `PATH` is set regardless.

If your shell is `tcsh`, add the following line to your `.tcshrc` file:

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

If the appropriate startup file does not exist in your home directory, create it with a text editor.

After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.3 MySQL Server and Server-Startup Programs

This section describes `mysqld`, the MySQL server, and several programs that are used to start the server.

4.3.1 `mysqld` — The MySQL Server

`mysqld`, also known as MySQL Server, is the main program that does most of the work in a MySQL installation. MySQL Server manages access to the MySQL data directory that contains databases and tables. The data directory is also the default location for other information such as log files and status files.

Note

Some installation packages contain a debugging version of the server named `mysqld-debug`. Invoke this version instead of `mysqld` for debugging support, memory allocation checking, and trace file support (see Section 29.5.1.2, “Creating Trace Files”).

When MySQL server starts, it listens for network connections from client programs and manages access to databases on behalf of those clients.

The `mysqld` program has many options that can be specified at startup. For a complete list of options, run this command:

```
shell> mysqld --verbose --help
```

MySQL Server also has a set of system variables that affect its operation as it runs. System variables can be set at server startup, and many of them can be changed at runtime to effect dynamic server

reconfiguration. MySQL Server also has a set of status variables that provide information about its operation. You can monitor these status variables to access runtime performance characteristics.

For a full description of MySQL Server command options, system variables, and status variables, see Section 5.1, “The MySQL Server”. For information about installing MySQL and setting up the initial configuration, see Chapter 2, *Installing and Upgrading MySQL*.

4.3.2 mysqld_safe — MySQL Server Startup Script

`mysqld_safe` is the recommended way to start a `mysqld` server on Unix. `mysqld_safe` adds some safety features such as restarting the server when an error occurs and logging runtime information to an error log. A description of error logging is given later in this section.

Note

For some Linux platforms, MySQL installation from RPM or Debian packages includes systemd support for managing MySQL server startup and shutdown. On these platforms, `mysqld_safe` is not installed because it is unnecessary. For more information, see Section 2.5.9, “Managing MySQL Server with systemd”.

`mysqld_safe` tries to start an executable named `mysqld`. To override the default behavior and specify explicitly the name of the server you want to run, specify a `--mysqld` or `--mysqld-version` option to `mysqld_safe`. You can also use `--ledir` to indicate the directory where `mysqld_safe` should look for the server.

Many of the options to `mysqld_safe` are the same as the options to `mysqld`. See Section 5.1.7, “Server Command Options”.

Options unknown to `mysqld_safe` are passed to `mysqld` if they are specified on the command line, but ignored if they are specified in the `[mysqld_safe]` group of an option file. See Section 4.2.2.2, “Using Option Files”.

`mysqld_safe` reads all options from the `[mysqld]`, `[server]`, and `[mysqld_safe]` sections in option files. For example, if you specify a `[mysqld]` section like this, `mysqld_safe` will find and use the `--log-error` option:

```
[mysqld]
log-error=error.log
```

For backward compatibility, `mysqld_safe` also reads `[safe_mysqld]` sections, but to be current you should rename such sections to `[mysqld_safe]`.

`mysqld_safe` accepts options on the command line and in option files, as described in the following table. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.6 mysqld_safe Options

Option Name	Description
--basedir	Path to MySQL installation directory
--core-file-size	Size of core file that mysqld should be able to create
--datadir	Path to data directory
--defaults-extra-file	Read named option file in addition to usual option files
--defaults-file	Read only named option file
--help	Display help message and exit

Option Name	Description
--ledir	Path to directory where server is located
--log-error	Write error log to named file
--malloc-lib	Alternative malloc library to use for mysqld
--mysqld	Name of server program to start (in ledir directory)
--mysqld-safe-log-timestamps	Timestamp format for logging
--mysqld-version	Suffix for server program name
--nice	Use nice program to set server scheduling priority
--no-defaults	Read no option files
--open-files-limit	Number of files that mysqld should be able to open
--pid-file	Path name of server process ID file
--plugin-dir	Directory where plugins are installed
--port	Port number on which to listen for TCP/IP connections
--skip-kill-mysqld	Do not try to kill stray mysqld processes
--skip-syslog	Do not write error messages to syslog; use error log file
--socket	Socket file on which to listen for Unix socket connections
--syslog	Write error messages to syslog
--syslog-tag	Tag suffix for messages written to syslog
--timezone	Set TZ time zone environment variable to named value
--user	Run mysqld as user having name user_name or numeric user ID user_id

- `--help`

  Display a help message and exit.

- `--basedir=`*`dir_name`*

  The path to the MySQL installation directory.

- `--core-file-size=`*`size`*

  The size of the core file that `mysqld` should be able to create. The option value is passed to `ulimit -c`.

  Note

  The `innodb_buffer_pool_in_core_file` variable can be used to reduce the size of core files on operating systems that support it. For more information, see Section 15.8.3.7, “Excluding Buffer Pool Pages from Core Files”.

- `--datadir=`*`dir_name`*

  The path to the data directory.

- `--defaults-extra-file=`*`file_name`*

  Read this option file in addition to the usual option files. If the file does not exist or is otherwise inaccessible, the server will exit with an error. *`file_name`* is interpreted relative to the current directory if

given as a relative path name rather than a full path name. This must be the first option on the command line if it is used.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, the server will exit with an error. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name. This must be the first option on the command line if it is used.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--ledir=dir_name`

  If `mysqld_safe` cannot find the server, use this option to indicate the path name to the directory where the server is located.

  This option is accepted only on the command line, not in option files. On platforms that use systemd, the value can be specified in the value of `MYSQLD_OPTS`. See Section 2.5.9, “Managing MySQL Server with systemd”.

- `--log-error=file_name`

  Write the error log to the given file. See Section 5.4.2, “The Error Log”.

- `--mysqld-safe-log-timestamps`

  This option controls the format for timestamps in log output produced by `mysqld_safe`. The following list describes the permitted values. For any other value, `mysqld_safe` logs a warning and uses `UTC` format.

  - `UTC`, `utc`

    ISO 8601 UTC format (same as `--log_timestamps=UTC` for the server). This is the default.

  - `SYSTEM`, `system`

    ISO 8601 local time format (same as `--log_timestamps=SYSTEM` for the server).

  - `HYPHEN`, `hyphen`

    `YY-MM-DD h:mm:ss` format, as in `mysqld_safe` for MySQL 5.6.

  - `LEGACY`, `legacy`

    `YYMMDD hh:mm:ss` format, as in `mysqld_safe` prior to MySQL 5.6.

- `--malloc-lib=[lib_name]`

  The name of the library to use for memory allocation instead of the system `malloc()` library. The option value must be one of the directories `/usr/lib`, `/usr/lib64`, `/usr/lib/i386-linux-gnu`, or `/usr/lib/x86_64-linux-gnu`.

  The `--malloc-lib` option works by modifying the `LD_PRELOAD` environment value to affect dynamic linking to enable the loader to find the memory-allocation library when `mysqld` runs:

- If the option is not given, or is given without a value (--malloc-lib=), LD_PRELOAD is not modified and no attempt is made to use tcmalloc.
- If the option is given as --malloc-lib=tcmalloc, mysqld_safe looks for a tcmalloc library in /usr/lib. If tmalloc is found, its path name is added to the beginning of the LD_PRELOAD value for mysqld. If tcmalloc is not found, mysqld_safe aborts with an error.
- If the option is given as --malloc-lib=*/path/to/some/library*, that full path is added to the beginning of the LD_PRELOAD value. If the full path points to a nonexistent or unreadable file, mysqld_safe aborts with an error.
- For cases where mysqld_safe adds a path name to LD_PRELOAD, it adds the path to the beginning of any existing value the variable already has.

Note

On systems that manage the server using systemd, mysqld_safe is not available. Instead, specify the allocation library by setting LD_PRELOAD in /etc/sysconfig/mysql.

Linux users can use the libtcmalloc_minimal.so library on any platform for which a tcmalloc package is installed in /usr/lib by adding these lines to the my.cnf file:

```
[mysqld_safe]
malloc-lib=tcmalloc
```

To use a specific tcmalloc library, specify its full path name. Example:

```
[mysqld_safe]
malloc-lib=/opt/lib/libtcmalloc_minimal.so
```

- --mysqld=*prog_name*

  The name of the server program (in the ledir directory) that you want to start. This option is needed if you use the MySQL binary distribution but have the data directory outside of the binary distribution. If mysqld_safe cannot find the server, use the --ledir option to indicate the path name to the directory where the server is located.

  This option is accepted only on the command line, not in option files. On platforms that use systemd, the value can be specified in the value of MYSQLD_OPTS. See Section 2.5.9, “Managing MySQL Server with systemd”.

- --mysqld-version=*suffix*

  This option is similar to the --mysqld option, but you specify only the suffix for the server program name. The base name is assumed to be mysqld. For example, if you use --mysqld-version=debug, mysqld_safe starts the mysqld-debug program in the ledir directory. If the argument to --mysqld-version is empty, mysqld_safe uses mysqld in the ledir directory.

  This option is accepted only on the command line, not in option files. On platforms that use systemd, the value can be specified in the value of MYSQLD_OPTS. See Section 2.5.9, “Managing MySQL Server with systemd”.

- --nice=*priority*

Use the nice program to set the server's scheduling priority to the given value.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read. This must be the first option on the command line if it is used.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --open-files-limit=*count*

  The number of files that mysqld should be able to open. The option value is passed to ulimit -n.

  Note

  You must start mysqld_safe as root for this to function properly.

- --pid-file=*file_name*

  The path name that mysqld should use for its process ID file.

- --plugin-dir=*dir_name*

  The path name of the plugin directory.

- --port=*port_num*

  The port number that the server should use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the root operating system user.

- --skip-kill-mysqld

  Do not try to kill stray mysqld processes at startup. This option works only on Linux.

- --socket=*path*

  The Unix socket file that the server should use when listening for local connections.

- --syslog, --skip-syslog

  --syslog causes error messages to be sent to syslog on systems that support the logger program. --skip-syslog suppresses the use of syslog; messages are written to an error log file.

  When syslog is used for error logging, the daemon.err facility/severity is used for all log messages.

  Using these options to control mysqld logging is deprecated. To write error log output to the system log, use the instructions at Section 5.4.2.7, “Error Logging to the System Log”. To control the facility, use the server log_syslog_facility system variable.

- --syslog-tag=*tag*

  For logging to syslog, messages from mysqld_safe and mysqld are written with identifiers of mysqld_safe and mysqld, respectively. To specify a suffix for the identifiers, use --syslog-tag=*tag*, which modifies the identifiers to be mysqld_safe-*tag* and mysqld-*tag*.

Using this option to control mysqld logging is deprecated. Use the server log_syslog_tag system variable instead. See Section 5.4.2.7, “Error Logging to the System Log”.

- --timezone=*timezone*

  Set the TZ time zone environment variable to the given option value. Consult your operating system documentation for legal time zone specification formats.

- --user={*user_name*|*user_id*}

  Run the mysqld server as the user having the name *user_name* or the numeric user ID *user_id*. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

If you execute mysqld_safe with the --defaults-file or --defaults-extra-file option to name an option file, the option must be the first one given on the command line or the option file will not be used. For example, this command will not use the named option file:

```
mysql> mysqld_safe --port=port_num --defaults-file=file_name
```

Instead, use the following command:

```
mysql> mysqld_safe --defaults-file=file_name --port=port_num
```

The mysqld_safe script is written so that it normally can start a server that was installed from either a source or a binary distribution of MySQL, even though these types of distributions typically install the server in slightly different locations. (See Section 2.1.4, “Installation Layouts”.) mysqld_safe expects one of the following conditions to be true:

- The server and databases can be found relative to the working directory (the directory from which mysqld_safe is invoked). For binary distributions, mysqld_safe looks under its working directory for bin and data directories. For source distributions, it looks for libexec and var directories. This condition should be met if you execute mysqld_safe from your MySQL installation directory (for example, /usr/local/mysql for a binary distribution).
- If the server and databases cannot be found relative to the working directory, mysqld_safe attempts to locate them by absolute path names. Typical locations are /usr/local/libexec and /usr/local/var. The actual locations are determined from the values configured into the distribution at the time it was built. They should be correct if MySQL is installed in the location specified at configuration time.

Because mysqld_safe tries to find the server and databases relative to its own working directory, you can install a binary distribution of MySQL anywhere, as long as you run mysqld_safe from the MySQL installation directory:

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

If mysqld_safe fails, even when invoked from the MySQL installation directory, specify the --ledir and --datadir options to indicate the directories in which the server and databases are located on your system.

mysqld_safe tries to use the sleep and date system utilities to determine how many times per second it has attempted to start. If these utilities are present and the attempted starts per second is greater than 5, mysqld_safe waits 1 full second before starting again. This is intended to prevent excessive CPU usage in the event of repeated failures. (Bug #11761530, Bug #54035)

When you use mysqld_safe to start mysqld, mysqld_safe arranges for error (and notice) messages from itself and from mysqld to go to the same destination.

There are several mysqld_safe options for controlling the destination of these messages:

- --log-error=*file_name*: Write error messages to the named error file.
- --syslog: Write error messages to syslog on systems that support the logger program.
- --skip-syslog: Do not write error messages to syslog. Messages are written to the default error log file (*host_name*.err in the data directory), or to a named file if the --log-error option is given.

If none of these options is given, the default is --skip-syslog.

When mysqld_safe writes a message, notices go to the logging destination (syslog or the error log file) and stdout. Errors go to the logging destination and stderr.

Note

Controlling mysqld logging from mysqld_safe is deprecated. Use the server's native syslog support instead. For more information, see Section 5.4.2.7, “Error Logging to the System Log”.

4.3.3 mysql.server — MySQL Server Startup Script

MySQL distributions on Unix and Unix-like system include a script named mysql.server, which starts the MySQL server using mysqld_safe. It can be used on systems such as Linux and Solaris that use System V-style run directories to start and stop system services. It is also used by the macOS Startup Item for MySQL.

mysql.server is the script name as used within the MySQL source tree. The installed name might be different (for example, mysqld or mysql). In the following discussion, adjust the name mysql.server as appropriate for your system.

Note

For some Linux platforms, MySQL installation from RPM or Debian packages includes systemd support for managing MySQL server startup and shutdown. On these platforms, mysql.server and mysqld_safe are not installed because they are unnecessary. For more information, see Section 2.5.9, “Managing MySQL Server with systemd”.

To start or stop the server manually using the mysql.server script, invoke it from the command line with start or stop arguments:

```
shell> mysql.server start
shell> mysql.server stop
```

mysql.server changes location to the MySQL installation directory, then invokes mysqld_safe. To run the server as some specific user, add an appropriate user option to the [mysqld] group of the global /etc/my.cnf option file, as shown later in this section. (It is possible that you must edit mysql.server if you've installed a binary distribution of MySQL in a nonstandard location. Modify it to change location into the proper directory before it runs mysqld_safe. If you do this, your modified version of mysql.server may be overwritten if you upgrade MySQL in the future; make a copy of your edited version that you can reinstall.)

mysql.server stop stops the server by sending a signal to it. You can also stop the server manually by executing mysqladmin shutdown.

To start and stop MySQL automatically on your server, you must add start and stop commands to the appropriate places in your /etc/rc* files:

- If you use the Linux server RPM package (MySQL-server-*VERSION*.rpm), or a native Linux package installation, the mysql.server script may be installed in the /etc/init.d directory with the name mysqld or mysql. See Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”, for more information on the Linux RPM packages.
- If you install MySQL from a source distribution or using a binary distribution format that does not install mysql.server automatically, you can install the script manually. It can be found in the support-files directory under the MySQL installation directory or in a MySQL source tree. Copy the script to the /etc/init.d directory with the name mysql and make it executable:

  ```
  shell> cp mysql.server /etc/init.d/mysql
  shell> chmod +x /etc/init.d/mysql
  ```

  After installing the script, the commands needed to activate it to run at system startup depend on your operating system. On Linux, you can use chkconfig:

  ```
  shell> chkconfig --add mysql
  ```

  On some Linux systems, the following command also seems to be necessary to fully enable the mysql script:

  ```
  shell> chkconfig --level 345 mysql on
  ```

- On FreeBSD, startup scripts generally should go in /usr/local/etc/rc.d/. Install the mysql.server script as /usr/local/etc/rc.d/mysql.server.sh to enable automatic startup. The rc(8) manual page states that scripts in this directory are executed only if their base name matches the *.sh shell file name pattern. Any other files or directories present within the directory are silently ignored.
- As an alternative to the preceding setup, some operating systems also use /etc/rc.local or /etc/init.d/boot.local to start additional services on startup. To start up MySQL using this method, append a command like the one following to the appropriate startup file:

  ```
  /bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
  ```

- For other systems, consult your operating system documentation to see how to install startup scripts.

mysql.server reads options from the [mysql.server] and [mysqld] sections of option files. For backward compatibility, it also reads [mysql_server] sections, but to be current you should rename such sections to [mysql.server].

You can add options for mysql.server in a global /etc/my.cnf file. A typical my.cnf file might look like this:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql
```

```
[mysql.server]
basedir=/usr/local/mysql
```

The mysql.server script supports the options shown in the following table. If specified, they *must* be placed in an option file, not on the command line. mysql.server supports only start and stop as command-line arguments.

Table 4.7 mysql.server Option-File Options

Option Name	Description	Type
basedir	Path to MySQL installation directory	Directory name
datadir	Path to MySQL data directory	Directory name
pid-file	File in which server should write its process ID	File name
service-startup-timeout	How long to wait for server startup	Integer

- basedir=*dir_name*

  The path to the MySQL installation directory.

- datadir=*dir_name*

  The path to the MySQL data directory.

- pid-file=*file_name*

  The path name of the file in which the server should write its process ID. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

  If this option is not given, mysql.server uses a default value of *host_name*.pid. The PID file value passed to mysqld_safe overrides any value specified in the [mysqld_safe] option file group. Because mysql.server reads the [mysqld] option file group but not the [mysqld_safe] group, you can ensure that mysqld_safe gets the same value when invoked from mysql.server as when invoked manually by putting the same pid-file setting in both the [mysqld_safe] and [mysqld] groups.

- service-startup-timeout=*seconds*

  How long in seconds to wait for confirmation of server startup. If the server does not start within this time, mysql.server exits with an error. The default value is 900. A value of 0 means not to wait at all for startup. Negative values mean to wait forever (no timeout).

4.3.4 mysqld_multi — Manage Multiple MySQL Servers

mysqld_multi is designed to manage several mysqld processes that listen for connections on different Unix socket files and TCP/IP ports. It can start or stop servers, or report their current status.

Note

For some Linux platforms, MySQL installation from RPM or Debian packages includes systemd support for managing MySQL server startup and shutdown. On these platforms, mysqld_multi is not installed because it is unnecessary.

> For information about using systemd to handle multiple MySQL instances, see Section 2.5.9, “Managing MySQL Server with systemd”.

mysqld_multi searches for groups named [mysqldN] in my.cnf (or in the file named by the --defaults-file option). *N* can be any positive integer. This number is referred to in the following discussion as the option group number, or *GNR*. Group numbers distinguish option groups from one another and are used as arguments to mysqld_multi to specify which servers you want to start, stop, or obtain a status report for. Options listed in these groups are the same that you would use in the [mysqld] group used for starting mysqld. (See, for example, Section 2.10.5, “Starting and Stopping MySQL Automatically”.) However, when using multiple servers, it is necessary that each one use its own value for options such as the Unix socket file and TCP/IP port number. For more information on which options must be unique per server in a multiple-server environment, see Section 5.8, “Running Multiple MySQL Instances on One Machine”.

To invoke mysqld_multi, use the following syntax:

```
shell> mysqld_multi [options] {start|stop|reload|report} [GNR[,GNR] ...]
```

start, stop, reload (stop and restart), and report indicate which operation to perform. You can perform the designated operation for a single server or multiple servers, depending on the *GNR* list that follows the option name. If there is no list, mysqld_multi performs the operation for all servers in the option file.

Each *GNR* value represents an option group number or range of group numbers. The value should be the number at the end of the group name in the option file. For example, the *GNR* for a group named [mysqld17] is 17. To specify a range of numbers, separate the first and last numbers by a dash. The *GNR* value 10-13 represents groups [mysqld10] through [mysqld13]. Multiple groups or group ranges can be specified on the command line, separated by commas. There must be no whitespace characters (spaces or tabs) in the *GNR* list; anything after a whitespace character is ignored.

This command starts a single server using option group [mysqld17]:

```
shell> mysqld_multi start 17
```

This command stops several servers, using option groups [mysqld8] and [mysqld10] through [mysqld13]:

```
shell> mysqld_multi stop 8,10-13
```

For an example of how you might set up an option file, use this command:

```
shell> mysqld_multi --example
```

mysqld_multi searches for option files as follows:

- With --no-defaults, no option files are read.
- With --defaults-file=*file_name*, only the named file is read.
- Otherwise, option files in the standard list of locations are read, including any file named by the --defaults-extra-file=*file_name* option, if one is given. (If the option is given multiple times, the last value is used.)

For additional information about these and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

Option files read are searched for [mysqld_multi] and [mysqldN] option groups. The [mysqld_multi] group can be used for options to mysqld_multi itself. [mysqldN] groups can be used for options passed to specific mysqld instances.

The [mysqld] or [mysqld_safe] groups can be used for common options read by all instances of mysqld or mysqld_safe. You can specify a --defaults-file=*file_name* option to use a different configuration file for that instance, in which case the [mysqld] or [mysqld_safe] groups from that file will be used for that instance.

mysqld_multi supports the following options.

- --help

  Display a help message and exit.

- --example

  Display a sample option file.

- --log=*file_name*

  Specify the name of the log file. If the file exists, log output is appended to it.

- --mysqladmin=*prog_name*

  The mysqladmin binary to be used to stop servers.

- --mysqld=*prog_name*

  The mysqld binary to be used. Note that you can specify mysqld_safe as the value for this option also. If you use mysqld_safe to start the server, you can include the mysqld or ledir options in the corresponding [mysqldN] option group. These options indicate the name of the server that mysqld_safe should start and the path name of the directory where the server is located. (See the descriptions for these options in Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”.) Example:

  ```
  [mysqld38]
  mysqld = mysqld-debug
  ledir  = /opt/local/mysql/libexec
  ```

- --no-log

  Print log information to stdout rather than to the log file. By default, output goes to the log file.

- --password=*password*

  The password of the MySQL account to use when invoking mysqladmin. Note that the password value is not optional for this option, unlike for other MySQL programs.

- --silent

  Silent mode; disable warnings.

- --tcp-ip

  Connect to each MySQL server through the TCP/IP port instead of the Unix socket file. (If a socket file is missing, the server might still be running, but accessible only through the TCP/IP port.) By default, connections are made using the Unix socket file. This option affects stop and report operations.

- `--user=user_name`

  The user name of the MySQL account to use when invoking `mysqladmin`.

- `--verbose`

  Be more verbose.

- `--version`

  Display version information and exit.

Some notes about `mysqld_multi`:

- **Most important**: Before using `mysqld_multi` be sure that you understand the meanings of the options that are passed to the `mysqld` servers and *why* you would want to have separate `mysqld` processes. Beware of the dangers of using multiple `mysqld` servers with the same data directory. Use separate data directories, unless you *know* what you are doing. Starting multiple servers with the same data directory does *not* give you extra performance in a threaded system. See Section 5.8, “Running Multiple MySQL Instances on One Machine”.

  Important

  Make sure that the data directory for each server is fully accessible to the Unix account that the specific `mysqld` process is started as. *Do not* use the Unix `root` account for this, unless you *know* what you are doing. See Section 6.1.5, “How to Run MySQL as a Normal User”.

- Make sure that the MySQL account used for stopping the `mysqld` servers (with the `mysqladmin` program) has the same user name and password for each server. Also, make sure that the account has the `SHUTDOWN` privilege. If the servers that you want to manage have different user names or passwords for the administrative accounts, you might want to create an account on each server that has the same user name and password. For example, you might set up a common `multi_admin` account by executing the following commands for each server:

  ```
  shell> mysql -u root -S /tmp/mysql.sock -p
  Enter password:
  mysql> CREATE USER 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
  mysql> GRANT SHUTDOWN ON *.* TO 'multi_admin'@'localhost';
  ```

  See Section 6.2, “Access Control and Account Management”. You have to do this for each `mysqld` server. Change the connection parameters appropriately when connecting to each one. Note that the host name part of the account name must permit you to connect as `multi_admin` from the host where you want to run `mysqld_multi`.

- The Unix socket file and the TCP/IP port number must be different for every `mysqld`. (Alternatively, if the host has multiple network addresses, you can set the `bind_address` system variable to cause different servers to listen to different interfaces.)

- The `--pid-file` option is very important if you are using `mysqld_safe` to start `mysqld` (for example, `--mysqld=mysqld_safe`) Every `mysqld` should have its own process ID file. The advantage of using `mysqld_safe` instead of `mysqld` is that `mysqld_safe` monitors its `mysqld` process and restarts it if the process terminates due to a signal sent using `kill -9` or for other reasons, such as a segmentation fault.

- You might want to use the `--user` option for `mysqld`, but to do this you need to run the `mysqld_multi` script as the Unix superuser (`root`). Having the option in the option file doesn't matter;

you just get a warning if you are not the superuser and the `mysqld` processes are started under your own Unix account.

The following example shows how you might set up an option file for use with `mysqld_multi`. The order in which the `mysqld` programs are started or stopped depends on the order in which they appear in the option file. Group numbers need not form an unbroken sequence. The first and fifth `[mysqldN]` groups were intentionally omitted from the example to illustrate that you can have “gaps” in the option file. This gives you more flexibility.

```
# This is an example of a my.cnf file for mysqld_multi.
# Usually this file is located in home dir ~/.my.cnf or /etc/my.cnf

[mysqld_multi]
mysqld     = /usr/local/mysql/bin/mysqld_safe
mysqladmin = /usr/local/mysql/bin/mysqladmin
user       = multi_admin
password   = my_password

[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /usr/local/mysql/data2/hostname.pid2
datadir    = /usr/local/mysql/data2
language   = /usr/local/mysql/share/mysql/english
user       = unix_user1

[mysqld3]
mysqld     = /path/to/mysqld_safe
ledir      = /path/to/mysqld-binary/
mysqladmin = /path/to/mysqladmin
socket     = /tmp/mysql.sock3
port       = 3308
pid-file   = /usr/local/mysql/data3/hostname.pid3
datadir    = /usr/local/mysql/data3
language   = /usr/local/mysql/share/mysql/swedish
user       = unix_user2

[mysqld4]
socket     = /tmp/mysql.sock4
port       = 3309
pid-file   = /usr/local/mysql/data4/hostname.pid4
datadir    = /usr/local/mysql/data4
language   = /usr/local/mysql/share/mysql/estonia
user       = unix_user3

[mysqld6]
socket     = /tmp/mysql.sock6
port       = 3311
pid-file   = /usr/local/mysql/data6/hostname.pid6
datadir    = /usr/local/mysql/data6
language   = /usr/local/mysql/share/mysql/japanese
user       = unix_user4
```

See Section 4.2.2.2, “Using Option Files”.

4.4 MySQL Installation-Related Programs

The programs in this section are used when installing or upgrading MySQL.

4.4.1 `comp_err` — Compile MySQL Error Message File

`comp_err` creates the `errmsg.sys` file that is used by `mysqld` to determine the error messages to display for different error codes. `comp_err` normally is run automatically when MySQL is built. It compiles

the errmsg.sys file from the text file located at sql/share/errmsg-utf8.txt in MySQL source distributions.

comp_err also generates mysqld_error.h, mysqld_ername.h, and sql_state.h header files.

For more information about how error messages are defined, see the MySQL Internals Manual.

Invoke comp_err like this:

```
shell> comp_err [options]
```

comp_err supports the following options.

- --help, -?

  Display a help message and exit.

- --charset=*dir_name*, -C *dir_name*

  The character set directory. The default is ../sql/share/charsets.

- --debug=*debug_options*, -# *debug_options*

  Write a debugging log. A typical *debug_options* string is d:t:O,*file_name*. The default is d:t:O,/tmp/comp_err.trace.

- --debug-info, -T

  Print some debugging information when the program exits.

- --header_file=*file_name*, -H *file_name*

  The name of the error header file. The default is mysqld_error.h.

- --in_file=*file_name*, -F *file_name*

  The name of the input file. The default is ../sql/share/errmsg-utf8.txt.

- --name_file=*file_name*, -N *file_name*

  The name of the error name file. The default is mysqld_ername.h.

- --out_dir=*dir_name*, -D *dir_name*

  The name of the output base directory. The default is ../sql/share/.

- --out_file=*file_name*, -O *file_name*

  The name of the output file. The default is errmsg.sys.

- --statefile=*file_name*, -S *file_name*

  The name for the SQLSTATE header file. The default is sql_state.h.

- --version, -V

  Display version information and exit.

4.4.2 mysql_secure_installation — Improve MySQL Installation Security

This program enables you to improve the security of your MySQL installation in the following ways:

- You can set a password for root accounts.
- You can remove root accounts that are accessible from outside the local host.
- You can remove anonymous-user accounts.
- You can remove the test database (which by default can be accessed by all users, even anonymous users), and privileges that permit anyone to access databases with names that start with test_.

mysql_secure_installation helps you implement security recommendations similar to those described at Section 2.10.4, “Securing the Initial MySQL Account”.

Normal usage is to connect to the local MySQL server; invoke mysql_secure_installation without arguments:

```
shell> mysql_secure_installation
```

When executed, mysql_secure_installation prompts you to determine which actions to perform.

The validate_password component can be used for password strength checking. If the plugin is not installed, mysql_secure_installation prompts the user whether to install it. Any passwords entered later are checked using the plugin if it is enabled.

Most of the usual MySQL client options such as --host and --port can be used on the command line and in option files. For example, to connect to the local server over IPv6 using port 3307, use this command:

```
shell> mysql_secure_installation --host=::1 --port=3307
```

mysql_secure_installation supports the following options, which can be specified on the command line or in the [mysql_secure_installation] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.8 mysql_secure_installation Options

Option Name	Description	Introduced
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	
--help	Display help message and exit	
--host	Host on which MySQL server is located	
--no-defaults	Read no option files	
--password	Accepted but always ignored. Whenever mysql_secure_installation is invoked, the user is prompted for a password, regardless.	
--port	TCP/IP port number for connection	
--print-defaults	Print default options	
--protocol	Connection protocol to use	
--socket	Unix socket file or Windows named pipe to use	

Option Name	Description	Introduced
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11
--ssl-key	File that contains X.509 key	
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16
--tls-version	Permissible TLS protocols for encrypted connections	
--use-default	Execute with no user interactivity	
--user	MySQL user name to use when connecting to server	

- `--help`, `-?`

  Display a help message and exit.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=`*`file_name`*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=`*`str`*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysql_secure_installation` normally reads the `[client]` and `[mysql_secure_installation]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql_secure_installation` also reads the `[client_other]` and `[mysql_secure_installation_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--host=`*`host_name`*, `-h` *`host_name`*

  Connect to the MySQL server on the given host.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--password=`*`password`*, `-p` *`password`*

  This option is accepted but ignored. Whether or not this option is used, `mysql_secure_installation` always prompts the user for a password.

- `--port=`*`port_num`*, `-P` *`port_num`*

  For TCP/IP connections, the port number to use.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

- `--socket=`*`path`*, `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--ssl-fips-mode={OFF|ON|STRICT}`

  Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-`*`xxx`* options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These `--ssl-fips-mode` values are permitted:

  - `OFF`: Disable FIPS mode.

- `ON`: Enable FIPS mode.
- `STRICT`: Enable “strict” FIPS mode.

Note

If the OpenSSL FIPS Object Module is not available, the only permitted value for `--ssl-fips-mode` is `OFF`. In this case, setting `--ssl-fips-mode` to `ON` or `STRICT` causes the client to produce a warning at startup and to operate in non-FIPS mode.

- `--tls-ciphersuites=`*`ciphersuite_list`*

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- `--tls-version=`*`protocol_list`*

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `--use-default`

  Execute noninteractively. This option can be used for unattended installation operations.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The user name of the MySQL account to use for connecting to the server.

4.4.3 `mysql_ssl_rsa_setup` — Create SSL/RSA Files

This program creates the SSL certificate and key files and RSA key-pair files required to support secure connections using SSL and secure password exchange using RSA over unencrypted connections, if those files are missing. `mysql_ssl_rsa_setup` can also be used to create new SSL files if the existing ones have expired.

Note

`mysql_ssl_rsa_setup` uses the `openssl` command, so its use is contingent on having OpenSSL installed on your machine.

Another way to generate SSL and RSA files, for MySQL distributions compiled using OpenSSL, is to have the server generate them automatically. See Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”.

Important

`mysql_ssl_rsa_setup` helps lower the barrier to using SSL by making it easier to generate the required files. However, certificates generated by `mysql_ssl_rsa_setup` are self-signed, which is not very secure. After you gain

> experience using the files created by mysql_ssl_rsa_setup, consider obtaining a CA certificate from a registered certificate authority.

Invoke mysql_ssl_rsa_setup like this:

```
shell> mysql_ssl_rsa_setup [options]
```

Typical options are --datadir to specify where to create the files, and --verbose to see the openssl commands that mysql_ssl_rsa_setup executes.

mysql_ssl_rsa_setup attempts to create SSL and RSA files using a default set of file names. It works as follows:

1. mysql_ssl_rsa_setup checks for the openssl binary at the locations specified by the PATH environment variable. If openssl is not found, mysql_ssl_rsa_setup does nothing. If openssl is present, mysql_ssl_rsa_setup looks for default SSL and RSA files in the MySQL data directory specified by the --datadir option, or the compiled-in data directory if the --datadir option is not given.

2. mysql_ssl_rsa_setup checks the data directory for SSL files with the following names:

   ```
   ca.pem
   server-cert.pem
   server-key.pem
   ```

3. If any of those files are present, mysql_ssl_rsa_setup creates no SSL files. Otherwise, it invokes openssl to create them, plus some additional files:

   ```
   ca.pem              Self-signed CA certificate
   ca-key.pem          CA private key
   server-cert.pem     Server certificate
   server-key.pem      Server private key
   client-cert.pem     Client certificate
   client-key.pem      Client private key
   ```

   These files enable secure client connections using SSL; see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

4. mysql_ssl_rsa_setup checks the data directory for RSA files with the following names:

   ```
   private_key.pem     Private member of private/public key pair
   public_key.pem      Public member of private/public key pair
   ```

5. If any of these files are present, mysql_ssl_rsa_setup creates no RSA files. Otherwise, it invokes openssl to create them. These files enable secure password exchange using RSA over unencrypted connections for accounts authenticated by the sha256_password or caching_sha2_password plugin; see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

For information about the characteristics of files created by mysql_ssl_rsa_setup, see Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”.

At startup, the MySQL server automatically uses the SSL files created by mysql_ssl_rsa_setup to enable SSL if no explicit SSL options are given other than --ssl (possibly along with --ssl-cipher). If you prefer to designate the files explicitly, invoke clients with the --ssl-ca, --ssl-cert, and --ssl-key options at startup to name the ca.pem, server-cert.pem, and server-key.pem files, respectively.

The server also automatically uses the RSA files created by mysql_ssl_rsa_setup to enable RSA if no explicit RSA options are given.

If the server is SSL-enabled, clients use SSL by default for the connection. To specify certificate and key files explicitly, use the --ssl-ca, --ssl-cert, and --ssl-key options to name the ca.pem, client-cert.pem, and client-key.pem files, respectively. However, some additional client setup may be required first because mysql_ssl_rsa_setup by default creates those files in the data directory. The permissions for the data directory normally enable access only to the system account that runs the MySQL server, so client programs cannot use files located there. To make the files available, copy them to a directory that is readable (but *not* writable) by clients:

- For local clients, the MySQL installation directory can be used. For example, if the data directory is a subdirectory of the installation directory and your current location is the data directory, you can copy the files like this:

```
cp ca.pem client-cert.pem client-key.pem ..
```

- For remote clients, distribute the files using a secure channel to ensure they are not tampered with during transit.

If the SSL files used for a MySQL installation have expired, you can use mysql_ssl_rsa_setup to create new ones:

1. Stop the server.
2. Rename or remove the existing SSL files. You may wish to make a backup of them first. (The RSA files do not expire, so you need not remove them. mysql_ssl_rsa_setup will see that they exist and not overwrite them.)
3. Run mysql_ssl_rsa_setup with the --datadir option to specify where to create the new files.
4. Restart the server.

mysql_ssl_rsa_setup supports the following command-line options, which can be specified on the command line or in the [mysql_ssl_rsa_setup] and [mysqld] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.9 mysql_ssl_rsa_setup Options

Option Name	Description
--datadir	Path to data directory
--help	Display help message and exit
--suffix	Suffix for X.509 certificate Common Name attribute
--uid	Name of effective user to use for file permissions
--verbose	Verbose mode
--version	Display version information and exit

- --help, ?

  Display a help message and exit.

- --datadir=*dir_name*

  The path to the directory that mysql_ssl_rsa_setup should check for default SSL and RSA files and in which it should create files if they are missing. The default is the compiled-in data directory.

- `--suffix=str`

  The suffix for the Common Name attribute in X.509 certificates. The suffix value is limited to 17 characters. The default is based on the MySQL version number.

- `--uid=name`, `-v`

  The name of the user who should be the owner of any created files. The value is a user name, not a numeric user ID. In the absence of this option, files created by `mysql_ssl_rsa_setup` are owned by the user who executes it. This option is valid only if you execute the program as `root` on a system that supports the `chown()` system call.

- `--verbose`, `-v`

  Verbose mode. Produce more output about what the program does. For example, the program shows the `openssl` commands it runs, and produces output to indicate whether it skips SSL or RSA file creation because some default file already exists.

- `--version`, `-V`

  Display version information and exit.

4.4.4 `mysql_tzinfo_to_sql` — Load the Time Zone Tables

The `mysql_tzinfo_to_sql` program loads the time zone tables in the `mysql` database. It is used on systems that have a *zoneinfo* database (the set of files describing time zones). Examples of such systems are Linux, FreeBSD, Solaris, and OS X. One likely location for these files is the `/usr/share/zoneinfo` directory (`/usr/share/lib/zoneinfo` on Solaris). If your system does not have a zoneinfo database, you can use the downloadable package described in Section 5.1.13, “MySQL Server Time Zone Support”.

`mysql_tzinfo_to_sql` can be invoked several ways:

```
shell> mysql_tzinfo_to_sql tz_dir
shell> mysql_tzinfo_to_sql tz_file tz_name
shell> mysql_tzinfo_to_sql --leap tz_file
```

For the first invocation syntax, pass the zoneinfo directory path name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

The second syntax causes `mysql_tzinfo_to_sql` to load a single time zone file *`tz_file`* that corresponds to a time zone name *`tz_name`*:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

If your time zone needs to account for leap seconds, invoke `mysql_tzinfo_to_sql` using the third syntax, which initializes the leap second information. *`tz_file`* is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

After running `mysql_tzinfo_to_sql`, it is best to restart the server so that it does not continue to use any previously cached time zone data.

4.4.5 mysql_upgrade — Check and Upgrade MySQL Tables

Note

As of MySQL 8.0.16, the MySQL server performs the upgrade tasks previously handled by mysql_upgrade (for details, see Section 2.11.3, "What the MySQL Upgrade Process Upgrades"). Consequently, mysql_upgrade is unneeded and is deprecated as of that version, and will be removed in a future MySQL version. Because mysql_upgrade no longer performs upgrade tasks, it exits with status 0 unconditionally.

Each time you upgrade MySQL, you should execute mysql_upgrade, which looks for incompatibilities with the upgraded MySQL server:

- It upgrades the system tables in the mysql schema so that you can take advantage of new privileges or capabilities that might have been added.
- It upgrades the Performance Schema, INFORMATION_SCHEMA, and sys schema.
- It examines user schemas.

If mysql_upgrade finds that a table has a possible incompatibility, it performs a table check and, if problems are found, attempts a table repair. If the table cannot be repaired, see Section 2.11.13, "Rebuilding or Repairing Tables or Indexes" for manual table repair strategies.

mysql_upgrade communicates directly with the MySQL server, sending it the SQL statements required to perform an upgrade.

Caution

You should always back up your current MySQL installation *before* performing an upgrade. See Section 7.2, "Database Backup Methods".

Some upgrade incompatibilities may require special handling *before* upgrading your MySQL installation and running mysql_upgrade. See Section 2.11, "Upgrading MySQL", for instructions on determining whether any such incompatibilities apply to your installation and how to handle them.

Use mysql_upgrade like this:

1. Ensure that the server is running.
2. Invoke mysql_upgrade to upgrade the system tables in the mysql schema and check and repair tables in other schemas:

```
shell> mysql_upgrade [options]
```

3. Stop the server and restart it so that any system table changes take effect.

If you have multiple MySQL server instances to upgrade, invoke mysql_upgrade with connection parameters appropriate for connecting to each of the desired servers. For example, with servers running on the local host on parts 3306 through 3308, upgrade each of them by connecting to the appropriate port:

```
shell> mysql_upgrade --protocol=tcp -P 3306 [other_options]
shell> mysql_upgrade --protocol=tcp -P 3307 [other_options]
shell> mysql_upgrade --protocol=tcp -P 3308 [other_options]
```

For local host connections on Unix, the --protocol=tcp option forces a connection using TCP/IP rather than the Unix socket file.

By default, mysql_upgrade runs as the MySQL root user. If the root password is expired when you run mysql_upgrade, you will see a message that your password is expired and that mysql_upgrade failed as a result. To correct this, reset the root password to unexpire it and run mysql_upgrade again. First, connect to the server as root:

```
shell> mysql -u root -p
Enter password: ****  <- enter root password here
```

Reset the password using ALTER USER:

```
mysql> ALTER USER USER() IDENTIFIED BY 'root-password';
```

Then exit mysql and run mysql_upgrade again:

```
shell> mysql_upgrade [options]
```

Note

If you run the server with the disabled_storage_engines system variable set to disable certain storage engines (for example, MyISAM), mysql_upgrade might fail with an error like this:

```
mysql_upgrade: [ERROR] 3161: Storage engine MyISAM is disabled
(Table creation is disallowed).
```

To handle this, restart the server with disabled_storage_engines disabled. Then you should be able to run mysql_upgrade successfully. After that, restart the server with disabled_storage_engines set to its original value.

Unless invoked with the --upgrade-system-tables option, mysql_upgrade processes all tables in all user schemas as necessary. Table checking might take a long time to complete. Each table is locked and therefore unavailable to other sessions while it is being processed. Check and repair operations can be time-consuming, particularly for large tables. Table checking uses the FOR UPGRADE option of the CHECK TABLE statement. For details about what this option entails, see Section 13.7.3.2, “CHECK TABLE Syntax”.

mysql_upgrade marks all checked and repaired tables with the current MySQL version number. This ensures that the next time you run mysql_upgrade with the same version of the server, it can be determined whether there is any need to check or repair a given table again.

mysql_upgrade saves the MySQL version number in a file named mysql_upgrade_info in the data directory. This is used to quickly check whether all tables have been checked for this release so that table-checking can be skipped. To ignore this file and perform the check regardless, use the --force option.

Note

The mysql_upgrade_info file is deprecated and will be removed in a future MySQL version.

mysql_upgrade checks mysql.user system table rows and, for any row with an empty plugin column, sets that column to 'mysql_native_password' if the credentials use a hash format compatible with that plugin. Rows with a pre-4.1 password hash must be upgraded manually.

mysql_upgrade does not upgrade the contents of the time zone tables or help tables. For upgrade instructions, see Section 5.1.13, “MySQL Server Time Zone Support”, and Section 5.1.14, “Server-Side Help Support”.

Unless invoked with the --skip-sys-schema option, mysql_upgrade installs the sys schema if it is not installed, and upgrades it to the current version otherwise. An error occurs if a sys schema exists but has no version view, on the assumption that its absence indicates a user-created schema:

```
A sys schema exists with no sys.version view. If
you have a user created sys schema, this must be renamed for the
upgrade to succeed.
```

To upgrade in this case, remove or rename the existing sys schema first.

mysql_upgrade supports the following options, which can be specified on the command line or in the [mysql_upgrade] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.10 mysql_upgrade Options

Option Name	Description	Introduced	Deprecated
--bind-address	Use specified network interface to connect to MySQL Server		
--character-sets-dir	Directory where character sets are installed		
--compress	Compress all information sent between client and server		8.0.18
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18	
--debug	Write debugging log		
--debug-check	Print debugging information when program exits		
--debug-info	Print debugging information, memory, and CPU statistics when program exits		
--default-auth	Authentication plugin to use		
--default-character-set	Specify default character set		
--defaults-extra-file	Read named option file in addition to usual option files		
--defaults-file	Read only named option file		
--defaults-group-suffix	Option group suffix value		
--force	Force execution even if mysql_upgrade has already been executed for current MySQL version		
--get-server-public-key	Request RSA public key from server	8.0.3	
--help	Display help message and exit		
--host	Host on which MySQL server is located		
--login-path	Read login path options from .mylogin.cnf		
--max-allowed-packet	Maximum packet length to send to or receive from server		
--net-buffer-length	Buffer size for TCP/IP and socket communication		
--no-defaults	Read no option files		

Option Name	Description	Introduced	Deprecated
--password	Password to use when connecting to server		
--pipe	Connect to server using named pipe (Windows only)		
--plugin-dir	Directory where plugins are installed		
--port	TCP/IP port number for connection		
--print-defaults	Print default options		
--protocol	Connection protocol to use		
--server-public-key-path	Path name to file containing RSA public key	8.0.4	
--shared-memory-base-name	Name of shared memory to use for shared-memory connections		
--skip-sys-schema	Do not install or upgrade sys schema		
--socket	Unix socket file or Windows named pipe to use		
--ssl-ca	File that contains list of trusted SSL Certificate Authorities		
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files		
--ssl-cert	File that contains X.509 certificate		
--ssl-cipher	Permissible ciphers for connection encryption		
--ssl-crl	File that contains certificate revocation lists		
--ssl-crlpath	Directory that contains certificate revocation-list files		
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11	
--ssl-key	File that contains X.509 key		
--ssl-mode	Desired security state of connection to server		
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16	
--tls-version	Permissible TLS protocols for encrypted connections		
--upgrade-system-tables	Update only system tables, not user schemas		
--user	MySQL user name to use when connecting to server		
--verbose	Verbose mode		
--version-check	Check for proper server version		
--write-binlog	Write all statements to binary log		
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18	

- `--help`

  Display a short help message and exit.

- `--bind-address=`*`ip_address`*

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=dir_name`

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

  As of MySQL 8.0.18, this option is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `--compression-algorithms=value`

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. The default value is `uncompressed`.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:O,/tmp/mysql_upgrade.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`, `-T`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.15, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysql_upgrade` normally reads the `[client]` and `[mysql_upgrade]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql_upgrade` also reads the `[client_other]` and `[mysql_upgrade_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--force`

  Ignore the `mysql_upgrade_info` file and force execution even if `mysql_upgrade` has already been executed for the current version of MySQL.

- `--get-server-public-key`

  Request from the server the public key required for RSA key pair-based password exchange. This option applies to clients that authenticate with the `caching_sha2_password` authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=`*`file_name`* is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  For information about the `caching_sha2_password` plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--host=`*`host_name`*, `-h` *`host_name`*

  Connect to the MySQL server on the given host.

- `--login-path=`*`name`*

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--max-allowed-packet=`*`value`*

  The maximum size of the buffer for client/server communication. The default value is 24MB. The minimum and maximum values are 4KB and 2GB.

- `--net-buffer-length=`*`value`*

  The initial size of the buffer for client/server communication. The default value is 1MB − 1KB. The minimum and maximum values are 4KB and 16MB.

- `--no-defaults`

Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.7, "`mysql_config_editor` — MySQL Configuration Utility".)

For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--password[=password]`, `-p[password]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, `mysql_upgrade` prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, "End-User Guidelines for Password Security".

  To explicitly specify that there is no password and that `mysql_upgrade` should not prompt for one, use the `--skip-password` option.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--plugin-dir=dir_name`

  The directory in which to look for plugins. Specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysql_upgrade` does not find it. See Section 6.2.17, "Pluggable Authentication".

- `--port=port_num`, `-P port_num`

  For TCP/IP connections, the port number to use.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, "Connecting to the MySQL Server Using Command Options".

- `--server-public-key-path=file_name`

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This

option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

For `sha256_password`, this option applies only if MySQL was built using OpenSSL.

For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--skip-sys-schema`

  By default, `mysql_upgrade` installs the `sys` schema if it is not installed, and upgrades it to the current version otherwise. The `--skip-sys-schema` option suppresses this behavior.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--ssl-fips-mode={OFF|ON|STRICT}`

  Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These `--ssl-fips-mode` values are permitted:

  - `OFF`: Disable FIPS mode.
  - `ON`: Enable FIPS mode.
  - `STRICT`: Enable “strict” FIPS mode.

  Note

  If the OpenSSL FIPS Object Module is not available, the only permitted value for `--ssl-fips-mode` is `OFF`. In this case, setting `--ssl-fips-mode` to `ON` or

> STRICT causes the client to produce a warning at startup and to operate in non-FIPS mode.

- --tls-ciphersuites=*ciphersuite_list*

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- --tls-version=*protocol_list*

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- --upgrade-system-tables, -s

  Upgrade only the system tables in the mysql schema, do not upgrade user schemas.

- --user=*user_name*, -u *user_name*

  The user name of the MySQL account to use for connecting to the server. The default user name is root.

- --verbose

  Verbose mode. Print more information about what the program does.

- --version-check, -k

  Check the version of the server to which mysql_upgrade is connecting to verify that it is the same as the version for which mysql_upgrade was built. If not, mysql_upgrade exits. This option is enabled by default; to disable the check, use --skip-version-check.

- --write-binlog

  By default, binary logging by mysql_upgrade is disabled. Invoke the program with --write-binlog if you want its actions to be written to the binary log.

  When the server is running with global transaction identifiers (GTIDs) enabled (gtid_mode=ON), do not enable binary logging by mysql_upgrade.

- --zstd-compression-level=*level*

  The compression level to use for connections to the server that use the zstd compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default zstd compression level is 3. The compression level setting has no effect on connections that do not use zstd compression.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

4.5 MySQL Client Programs

This section describes client programs that connect to the MySQL server.

4.5.1 `mysql` — The MySQL Command-Line Client

`mysql` is a simple SQL shell with input line editing capabilities. It supports interactive and noninteractive use. When used interactively, query results are presented in an ASCII-table format. When used noninteractively (for example, as a filter), the result is presented in tab-separated format. The output format can be changed using command options.

If you have problems due to insufficient memory for large result sets, use the `--quick` option. This forces `mysql` to retrieve results from the server a row at a time rather than retrieving the entire result set and buffering it in memory before displaying it. This is done by returning the result set using the `mysql_use_result()` C API function in the client/server library rather than `mysql_store_result()`.

Note

Alternatively, MySQL Shell offers access to the X DevAPI. For details, see MySQL Shell 8.0 (part of MySQL 8.0).

Using `mysql` is very easy. Invoke it from the prompt of your command interpreter as follows:

```
shell> mysql db_name
```

Or:

```
shell> mysql --user=user_name --password db_name
Enter password: your_password
```

Then type an SQL statement, end it with `;`, `\g`, or `\G` and press Enter.

Typing **Control+C** interrupts the current statement if there is one, or cancels any partial input line otherwise.

You can execute SQL statements in a script file (batch file) like this:

```
shell> mysql db_name < script.sql > output.tab
```

On Unix, the `mysql` client logs statements executed interactively to a history file. See Section 4.5.1.3, “mysql Client Logging”.

4.5.1.1 mysql Client Options

`mysql` supports the following options, which can be specified on the command line or in the `[mysql]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.11 mysql Client Options

Option Name	Description	Introduced	Deprecated	Removed
--auto-rehash	Enable automatic rehashing			
--auto-vertical-output	Enable automatic vertical result set display			
--batch	Do not use history file			
--binary-as-hex	Display binary values in hexadecimal notation	8.0.2		

Option Name	Description	Introduced	Deprecated	Removed
--binary-mode	Disable \r\n - to - \n translation and treatment of \0 as end-of-query			
--bind-address	Use specified network interface to connect to MySQL Server			
--character-sets-dir	Directory where character sets are installed			
--column-names	Write column names in results			
--column-type-info	Display result set metadata			
--comments	Whether to retain or strip comments in statements sent to the server			
--compress	Compress all information sent between client and server		8.0.18	
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18		
--connect-expired-password	Indicate to server that client can handle expired-password sandbox mode.			
--connect-timeout	Number of seconds before connection timeout			
--database	The database to use			
--debug	Write debugging log; supported only if MySQL was built with debugging support			
--debug-check	Print debugging information when program exits			
--debug-info	Print debugging information, memory, and CPU statistics when program exits			
--default-auth	Authentication plugin to use			
--default-character-set	Specify default character set			
--defaults-extra-file	Read named option file in addition to usual option files			
--defaults-file	Read only named option file			
--defaults-group-suffix	Option group suffix value			
--delimiter	Set the statement delimiter			
--enable-cleartext-plugin	Enable cleartext authentication plugin			
--execute	Execute the statement and quit			
--force	Continue even if an SQL error occurs			
--get-server-public-key	Request RSA public key from server	8.0.3		
--help	Display help message and exit			
--histignore	Patterns specifying which statements to ignore for logging			
--host	Host on which MySQL server is located			
--html	Produce HTML output			
--ignore-spaces	Ignore spaces after function names			
--init-command	SQL statement to execute after connecting			

Option Name	Description	Introduced	Deprecated	Removed
--line-numbers	Write line numbers for errors			
--local-infile	Enable or disable for LOCAL capability for LOAD DATA			
--login-path	Read login path options from .mylogin.cnf			
--max-allowed-packet	Maximum packet length to send to or receive from server			
--max-join-size	The automatic limit for rows in a join when using --safe-updates			
--named-commands	Enable named mysql commands			
--net-buffer-length	Buffer size for TCP/IP and socket communication			
--no-auto-rehash	Disable automatic rehashing			
--no-beep	Do not beep when errors occur			
--no-defaults	Read no option files			
--one-database	Ignore statements except those for the default database named on the command line			
--pager	Use the given command for paging query output			
--password	Password to use when connecting to server			
--pipe	Connect to server using named pipe (Windows only)			
--plugin-dir	Directory where plugins are installed			
--port	TCP/IP port number for connection			
--print-defaults	Print default options			
--prompt	Set the prompt to the specified format			
--protocol	Connection protocol to use			
--quick	Do not cache each query result			
--raw	Write column values without escape conversion			
--reconnect	If the connection to the server is lost, automatically try to reconnect			
--i-am-a-dummy, --safe-updates	Allow only UPDATE and DELETE statements that specify key values			
--secure-auth	Do not send passwords to server in old (pre-4.1) format			8.0.3
--select-limit	The automatic limit for SELECT statements when using --safe-updates			
--server-public-key-path	Path name to file containing RSA public key			
--shared-memory-base-name	Name of shared memory to use for shared-memory connections			
--show-warnings	Show warnings after each statement if there are any			

Option Name	Description	Introduced	Deprecated	Removed
--sigint-ignore	Ignore SIGINT signals (typically the result of typing Control+C)			
--silent	Silent mode			
--skip-auto-rehash	Disable automatic rehashing			
--skip-column-names	Do not write column names in results			
--skip-line-numbers	Skip line numbers for errors			
--skip-named-commands	Disable named mysql commands			
--skip-pager	Disable paging			
--skip-reconnect	Disable reconnecting			
--socket	Unix socket file or Windows named pipe to use			
--ssl-ca	File that contains list of trusted SSL Certificate Authorities			
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files			
--ssl-cert	File that contains X.509 certificate			
--ssl-cipher	Permissible ciphers for connection encryption			
--ssl-crl	File that contains certificate revocation lists			
--ssl-crlpath	Directory that contains certificate revocation-list files			
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11		
--ssl-key	File that contains X.509 key			
--ssl-mode	Desired security state of connection to server			
--syslog	Log interactive statements to syslog			
--table	Display output in tabular format			
--tee	Append a copy of output to named file			
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16		
--tls-version	Permissible TLS protocols for encrypted connections			
--unbuffered	Flush the buffer after each query			
--user	MySQL user name to use when connecting to server			
--verbose	Verbose mode			
--version	Display version information and exit			
--vertical	Print query output rows vertically (one line per column value)			
--wait	If the connection cannot be established, wait and retry instead of aborting			
--xml	Produce XML output			
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18		

- `--help`, `-?`

  Display a help message and exit.

- `--auto-rehash`

  Enable automatic rehashing. This option is on by default, which enables database, table, and column name completion. Use `--disable-auto-rehash` to disable rehashing. That causes `mysql` to start faster, but you must issue the `rehash` command or its `\#` shortcut if you want to use name completion.

  To complete a name, enter the first part and press Tab. If the name is unambiguous, `mysql` completes it. Otherwise, you can press Tab again to see the possible names that begin with what you have typed so far. Completion does not occur if there is no default database.

  Note

  This feature requires a MySQL client that is compiled with the **readline** library. Typically, the **readline** library is not available on Windows.

- `--auto-vertical-output`

  Cause result sets to be displayed vertically if they are too wide for the current window, and using normal tabular format otherwise. (This applies to statements terminated by `;` or `\G`.)

- `--batch`, `-B`

  Print results using tab as the column separator, with each row on a new line. With this option, `mysql` does not use the history file.

  Batch mode results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` option.

- `--binary-as-hex`

  When this option is given, `mysql` displays binary data using hexadecimal notation (`0x`*`value`*). This occurs whether the overall output dislay format is tabular, vertical, HTML, or XML.

- `--binary-mode`

  This option helps when processing `mysqlbinlog` output that may contain `BLOB` values. By default, `mysql` translates `\r\n` in statement strings to `\n` and interprets `\0` as the statement terminator. `--binary-mode` disables both features. It also disables all `mysql` commands except `charset` and `delimiter` in noninteractive mode (for input piped to `mysql` or loaded using the `source` command).

- `--bind-address=`*`ip_address`*

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--column-names`

  Write column names in results.

- `--column-type-info`

Display result set metadata.

- `--comments`, `-c`

  Whether to strip or preserve comments in statements sent to the server. The default is `--skip-comments` (strip comments), enable with `--comments` (preserve comments).

  Note

  The `mysql` client always passes optimizer hints to the server, regardless of whether this option is given.

  Comment stripping is deprecated. This feature and the options to control it will be removed in a future MySQL release.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

  As of MySQL 8.0.18, this option is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `--compression-algorithms=`*`value`*

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. The default value is `uncompressed`.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

- `--connect-expired-password`

  Indicate to the server that the client can handle sandbox mode if the account used to connect has an expired password. This can be useful for noninteractive invocations of `mysql` because normally the server disconnects noninteractive clients that attempt to connect using an account with an expired password. (See Section 6.2.16, “Server Handling of Expired Passwords”.)

- `--connect-timeout=`*`value`*

  The number of seconds before connection timeout. (Default value is `0`.)

- `--database=`*`db_name`*, `-D` *`db_name`*

  The database to use. This is useful primarily in an option file.

- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:o,/tmp/mysql.trace`.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-check`

Print some debugging information when the program exits.

- `--debug-info`, `-T`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set for the client and connection.

  This option can be useful if the operating system uses one character set and the `mysql` client by default uses another. In this case, output may be formatted incorrectly. You can usually fix such issues by using this option to force the client to use the system character set instead.

  For more information, see Section 10.4, “Connection Character Sets and Collations”, and Section 10.15, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysql` normally reads the `[client]` and `[mysql]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql` also reads the `[client_other]` and `[mysql_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--delimiter=str`

  Set the statement delimiter. The default is the semicolon character (`;`).

- `--disable-named-commands`

Disable named commands. Use the `\*` form only, or use named commands only at the beginning of a line ending with a semicolon (`;`). `mysql` starts with this option *enabled* by default. However, even with this option, long-format commands still work from the first line. See Section 4.5.1.2, “mysql Client Commands”.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.)

- `--execute=`*`statement`*, `-e` *`statement`*

  Execute the statement and quit. The default output format is like that produced with `--batch`. See Section 4.2.2.1, “Using Options on the Command Line”, for some examples. With this option, `mysql` does not use the history file.

- `--force`, `-f`

  Continue even if an SQL error occurs.

- `--get-server-public-key`

  Request from the server the public key required for RSA key pair-based password exchange. This option applies to clients that authenticate with the `caching_sha2_password` authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=`*`file_name`* is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  For information about the `caching_sha2_password` plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--histignore`

  A list of one or more colon-separated patterns specifying statements to ignore for logging purposes. These patterns are added to the default pattern list (`"*IDENTIFIED*:*PASSWORD*"`). The value specified for this option affects logging of statements written to the history file, and to `syslog` if the `--syslog` option is given. For more information, see Section 4.5.1.3, “mysql Client Logging”.

- `--host=`*`host_name`*, `-h` *`host_name`*

  Connect to the MySQL server on the given host.

- `--html`, `-H`

  Produce HTML output.

- `--ignore-spaces`, `-i`

  Ignore spaces after function names. The effect of this is described in the discussion for the `IGNORE_SPACE` SQL mode (see Section 5.1.11, “Server SQL Modes”).

- `--init-command=str`

  SQL statement to execute after connecting to the server. If auto-reconnect is enabled, the statement is executed again after reconnection occurs.

- `--line-numbers`

  Write line numbers for errors. Disable this with `--skip-line-numbers`.

- `--local-infile[={0|1}]`

  Enable or disable `LOCAL` capability for `LOAD DATA`. For `mysql`, this capability is disabled by default. With no value, the option enables `LOCAL`. The option may be given as `--local-infile=0` or `--local-infile=1` to explicitly disable or enable `LOCAL`. Enabling local data loading also requires that the server permits it; see Section 6.1.6, “Security Issues with LOAD DATA LOCAL”

- `--login-path=`*name*

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--max-allowed-packet=`*value*

  The maximum size of the buffer for client/server communication. The default is 16MB, the maximum is 1GB.

- `--max-join-size=`*value*

  The automatic limit for rows in a join when using `--safe-updates`. (Default value is 1,000,000.)

- `--named-commands`, `-G`

  Enable named `mysql` commands. Long-format commands are permitted, not just short-format commands. For example, `quit` and `\q` both are recognized. Use `--skip-named-commands` to disable named commands. See Section 4.5.1.2, “mysql Client Commands”.

- `--net-buffer-length=`*value*

  The buffer size for TCP/IP and socket communication. (Default value is 16KB.)

- `--no-auto-rehash`, `-A`

  This has the same effect as `--skip-auto-rehash`. See the description for `--auto-rehash`.

- `--no-beep`, `-b`

  Do not beep when errors occur.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.)

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--one-database`, `-o`

  Ignore statements except those that occur while the default database is the one named on the command line. This option is rudimentary and should be used with care. Statement filtering is based only on `USE` statements.

  Initially, `mysql` executes statements in the input because specifying a database *`db_name`* on the command line is equivalent to inserting `USE` *`db_name`* at the beginning of the input. Then, for each `USE` statement encountered, `mysql` accepts or rejects following statements depending on whether the database named is the one on the command line. The content of the statements is immaterial.

  Suppose that `mysql` is invoked to process this set of statements:

  ```
  DELETE FROM db2.t2;
  USE db2;
  DROP TABLE db1.t1;
  CREATE TABLE db1.t1 (i INT);
  USE db1;
  INSERT INTO t1 (i) VALUES(1);
  CREATE TABLE db2.t1 (j INT);
  ```

  If the command line is `mysql --force --one-database db1`, `mysql` handles the input as follows:

  - The `DELETE` statement is executed because the default database is `db1`, even though the statement names a table in a different database.
  - The `DROP TABLE` and `CREATE TABLE` statements are not executed because the default database is not `db1`, even though the statements name a table in `db1`.
  - The `INSERT` and `CREATE TABLE` statements are executed because the default database is `db1`, even though the `CREATE TABLE` statement names a table in a different database.

- `--pager[=`*`command`*`]`

  Use the given command for paging query output. If the command is omitted, the default pager is the value of your `PAGER` environment variable. Valid pagers are `less`, `more`, `cat [> filename]`, and so forth. This option works only on Unix and only in interactive mode. To disable paging, use `--skip-pager`. Section 4.5.1.2, “mysql Client Commands”, discusses output paging further.

- `--password[=`*`password`*`]`, `-p[`*`password`*`]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, `mysql` prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

  To explicitly specify that there is no password and that `mysql` should not prompt for one, use the `--skip-password` option.

- `--pipe`, `-W`

On Windows, connect to the server using a named pipe. This option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysql does not find it. See Section 6.2.17, “Pluggable Authentication”.

- --port=*port_num*, -P *port_num*

  For TCP/IP connections, the port number to use.

- --print-defaults

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --prompt=*format_str*

  Set the prompt to the specified format. The default is mysql>. The special sequences that the prompt can contain are described in Section 4.5.1.2, “mysql Client Commands”.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

- --quick, -q

  Do not cache each query result, print each row as it is received. This may slow down the server if the output is suspended. With this option, mysql does not use the history file.

- --raw, -r

  For tabular output, the “boxing” around columns enables one column value to be distinguished from another. For nontabular output (such as is produced in batch mode or when the --batch or --silent option is given), special characters are escaped in the output so they can be identified easily. Newline, tab, NUL, and backslash are written as \n, \t, \0, and \\. The --raw option disables this character escaping.

  The following example demonstrates tabular versus nontabular output and the use of raw mode to disable escaping:

```
% mysql
mysql> SELECT CHAR(92);
+----------+
| CHAR(92) |
+----------+
| \        |
+----------+

% mysql -s
```

```
mysql> SELECT CHAR(92);
CHAR(92)
\\

% mysql -s -r
mysql> SELECT CHAR(92);
CHAR(92)
\
```

- `--reconnect`

  If the connection to the server is lost, automatically try to reconnect. A single reconnect attempt is made each time the connection is lost. To suppress reconnection behavior, use `--skip-reconnect`.

- `--safe-updates`, `--i-am-a-dummy`, `-U`

  If this option is enabled, `UPDATE` and `DELETE` statements that do not use a key in the `WHERE` clause or a `LIMIT` clause produce an error. In addition, restrictions are placed on `SELECT` statements that produce (or are estimated to produce) very large result sets. If you have set this option in an option file, you can use `--skip-safe-updates` on the command line to override it. For more information about this option, see Using Safe-Updates Mode (--safe-updates).

- `--secure-auth`

  This option was removed in MySQL 8.0.3.

- `--select-limit=`*`value`*

  The automatic limit for `SELECT` statements when using `--safe-updates`. (Default value is 1,000.)

- `--server-public-key-path=`*`file_name`*

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=`*`file_name`* is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  This option is available only if MySQL was built using OpenSSL.

  For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--shared-memory-base-name=`*`name`*

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--show-warnings`

Cause warnings to be shown after each statement if there are any. This option applies to interactive and batch mode.

- `--sigint-ignore`

  Ignore `SIGINT` signals (typically the result of typing **Control+C**).

- `--silent`, `-s`

  Silent mode. Produce less output. This option can be given multiple times to produce less and less output.

  This option results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` option.

- `--skip-column-names`, `-N`

  Do not write column names in results.

- `--skip-line-numbers`, `-L`

  Do not write line numbers for errors. Useful when you want to compare result files that include error messages.

- `--socket=`*`path`*, `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--ssl-fips-mode={OFF|ON|STRICT}`

  Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-`*`xxx`* options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These `--ssl-fips-mode` values are permitted:

  - `OFF`: Disable FIPS mode.
  - `ON`: Enable FIPS mode.
  - `STRICT`: Enable “strict” FIPS mode.

  Note

  If the OpenSSL FIPS Object Module is not available, the only permitted value for `--ssl-fips-mode` is `OFF`. In this case, setting `--ssl-fips-mode` to `ON` or `STRICT` causes the client to produce a warning at startup and to operate in non-FIPS mode.

- `--syslog`, `-j`

  This option causes `mysql` to send interactive statements to the system logging facility. On Unix, this is `syslog`; on Windows, it is the Windows Event Log. The destination where logged messages appear is system dependent. On Linux, the destination is often the `/var/log/messages` file.

  Here is a sample of output generated on Linux by using `--syslog`. This output is formatted for readability; each logged message actually takes a single line.

  ```
  Mar  7 12:39:25 myhost MysqlClient[20824]:
    SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,
    DB_SERVER:'127.0.0.1', DB:'--', QUERY:'USE test;'
  Mar  7 12:39:28 myhost MysqlClient[20824]:
    SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,
    DB_SERVER:'127.0.0.1', DB:'test', QUERY:'SHOW TABLES;'
  ```

  For more information, see Section 4.5.1.3, “mysql Client Logging”.

- `--table`, `-t`

  Display output in table format. This is the default for interactive use, but can be used to produce table output in batch mode.

- `--tee=`*`file_name`*

  Append a copy of output to the given file. This option works only in interactive mode. Section 4.5.1.2, “mysql Client Commands”, discusses tee files further.

- `--tls-ciphersuites=`*`ciphersuite_list`*

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- `--tls-version=`*`protocol_list`*

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `--unbuffered`, `-n`

  Flush the buffer after each query.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The user name of the MySQL account to use for connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Produce more output about what the program does. This option can be given multiple times to produce more and more output. (For example, `-v -v -v` produces table output format even in batch mode.)

- `--version`, `-V`

Display version information and exit.

- `--vertical`, `-E`

  Print query output rows vertically (one line per column value). Without this option, you can specify vertical output for individual statements by terminating them with `\G`.

- `--wait`, `-w`

  If the connection cannot be established, wait and retry instead of aborting.

- `--xml`, `-X`

  Produce XML output.

```
<field name="column_name">NULL</field>
```

  The output when `--xml` is used with `mysql` matches that of `mysqldump --xml`. See Section 4.5.4, “mysqldump — A Database Backup Program”, for details.

  The XML output also uses an XML namespace, as shown here:

```
shell> mysql --xml -uroot -e "SHOW VARIABLES LIKE 'version%'"
<?xml version="1.0"?>

<resultset statement="SHOW VARIABLES LIKE 'version%'" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
<row>
<field name="Variable_name">version</field>
<field name="Value">5.0.40-debug</field>
</row>

<row>
<field name="Variable_name">version_comment</field>
<field name="Value">Source distribution</field>
</row>

<row>
<field name="Variable_name">version_compile_machine</field>
<field name="Value">i686</field>
</row>

<row>
<field name="Variable_name">version_compile_os</field>
<field name="Value">suse-linux-gnu</field>
</row>
</resultset>
```

- `--zstd-compression-level=level`

  The compression level to use for connections to the server that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. The compression level setting has no effect on connections that do not use `zstd` compression.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

4.5.1.2 mysql Client Commands

mysql sends each SQL statement that you issue to the server to be executed. There is also a set of commands that mysql itself interprets. For a list of these commands, type help or \h at the mysql> prompt:

```
mysql> help

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear the current input statement.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter.
edit      (\e) Edit command with $EDITOR.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
system    (\!) Execute a system shell command.
tee       (\T) Set outfile [to_outfile]. Append everything into given
               outfile.
use       (\u) Use another database. Takes database name as argument.
charset   (\C) Switch to another charset. Might be needed for processing
               binlog with multi-byte charsets.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.
resetconnection(\x) Clean session context.

For server side help, type 'help contents'
```

If mysql is invoked with the --binary-mode option, all mysql commands are disabled except charset and delimiter in noninteractive mode (for input piped to mysql or loaded using the source command).

Each command has both a long and short form. The long form is not case-sensitive; the short form is. The long form can be followed by an optional semicolon terminator, but the short form should not.

The use of short-form commands within multiple-line /* ... */ comments is not supported. Short-form commands do work within single-line /*! ... */ version comments, as do /*+ ... */ optimizer-hint comments, which are stored in object definitions. If there is a concern that optimizer-hint comments may be stored in object definitions so that dump files when reloaded with mysql would result in execution of such commands, either invoke mysql with the --binary-mode option or use a reload client other than mysql.

- help [*arg*], \h [*arg*], \? [*arg*], ? [*arg*]

  Display a help message listing the available mysql commands.

  If you provide an argument to the help command, mysql uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. For more information, see Section 4.5.1.4, “mysql Client Server-Side Help”.

- charset *charset_name*, \C *charset_name*

Change the default character set and issue a SET NAMES statement. This enables the character set to remain synchronized on the client and server if mysql is run with auto-reconnect enabled (which is not recommended), because the specified character set is used for reconnects.

- clear, \c

  Clear the current input. Use this if you change your mind about executing the statement that you are entering.

- connect [*db_name host_name*], \r [*db_name host_name*]

  Reconnect to the server. The optional database name and host name arguments may be given to specify the default database or the host where the server is running. If omitted, the current values are used.

- delimiter *str*, \d *str*

  Change the string that mysql interprets as the separator between SQL statements. The default is the semicolon character (;).

  The delimiter string can be specified as an unquoted or quoted argument on the delimiter command line. Quoting can be done with either single quote ('), double quote ("), or backtick (`) characters. To include a quote within a quoted string, either quote the string with a different quote character or escape the quote with a backslash (\) character. Backslash should be avoided outside of quoted strings because it is the escape character for MySQL. For an unquoted argument, the delimiter is read up to the first space or end of line. For a quoted argument, the delimiter is read up to the matching quote on the line.

  mysql interprets instances of the delimiter string as a statement delimiter anywhere it occurs, except within quoted strings. Be careful about defining a delimiter that might occur within other words. For example, if you define the delimiter as X, you will be unable to use the word INDEX in statements. mysql interprets this as INDE followed by the delimiter X.

  When the delimiter recognized by mysql is set to something other than the default of ;, instances of that character are sent to the server without interpretation. However, the server itself still interprets ; as a statement delimiter and processes statements accordingly. This behavior on the server side comes into play for multiple-statement execution (see Section 28.7.23, “C API Multiple Statement Execution Support”), and for parsing the body of stored procedures and functions, triggers, and events (see Section 24.1, “Defining Stored Programs”).

- edit, \e

  Edit the current input statement. mysql checks the values of the EDITOR and VISUAL environment variables to determine which editor to use. The default editor is vi if neither variable is set.

  The edit command works only in Unix.

- ego, \G

  Send the current statement to the server to be executed and display the result using vertical format.

- exit, \q

  Exit mysql.

- go, \g

Send the current statement to the server to be executed.

- nopager, \n

  Disable output paging. See the description for pager.

  The nopager command works only in Unix.

- notee, \t

  Disable output copying to the tee file. See the description for tee.

- nowarning, \w

  Disable display of warnings after each statement.

- pager [*command*], \P [*command*]

  Enable output paging. By using the --pager option when you invoke mysql, it is possible to browse or search query results in interactive mode with Unix programs such as less, more, or any other similar program. If you specify no value for the option, mysql checks the value of the PAGER environment variable and sets the pager to that. Pager functionality works only in interactive mode.

  Output paging can be enabled interactively with the pager command and disabled with nopager. The command takes an optional argument; if given, the paging program is set to that. With no argument, the pager is set to the pager that was set on the command line, or stdout if no pager was specified.

  Output paging works only in Unix because it uses the popen() function, which does not exist on Windows. For Windows, the tee option can be used instead to save query output, although it is not as convenient as pager for browsing output in some situations.

- print, \p

  Print the current input statement without executing it.

- prompt [*str*], \R [*str*]

  Reconfigure the mysql prompt to the given string. The special character sequences that can be used in the prompt are described later in this section.

  If you specify the prompt command with no argument, mysql resets the prompt to the default of mysql>.

- quit, \q

  Exit mysql.

- rehash, \#

  Rebuild the completion hash that enables database, table, and column name completion while you are entering statements. (See the description for the --auto-rehash option.)

- resetconnection, \x

  Reset the connection to clear the session state.

Resetting a connection has effects similar to `mysql_change_user()` or an auto-reconnect except that the connection is not closed and reopened, and re-authentication is not done. See Section 28.7.7.3, “mysql_change_user()”, and Section 28.7.28, “C API Automatic Reconnection Control”.

This example shows how `resetconnection` clears a value maintained in the session state:

```
mysql> SELECT LAST_INSERT_ID(3);
+------------------+
| LAST_INSERT_ID(3) |
+------------------+
|                3 |
+------------------+

mysql> SELECT LAST_INSERT_ID();
+-----------------+
| LAST_INSERT_ID() |
+-----------------+
|               3 |
+-----------------+

mysql> resetconnection;

mysql> SELECT LAST_INSERT_ID();
+-----------------+
| LAST_INSERT_ID() |
+-----------------+
|               0 |
+-----------------+
```

- `source` *`file_name`*, `\.` *`file_name`*

  Read the named file and executes the statements contained therein. On Windows, specify path name separators as `/` or `\\`.

  Quote characters are taken as part of the file name itself. For best results, the name should not include space characters.

- `status`, `\s`

  Provide status information about the connection and the server you are using. If you are running with `--safe-updates` enabled, `status` also prints the values for the `mysql` variables that affect your queries.

- `system` *`command`*, `\!` *`command`*

  Execute the given command using your default command interpreter.

  Prior to MySQL 8.0.19, the `system` command works only in Unix. As of 8.0.19, it also works on Windows.

- `tee [`*`file_name`*`]`, `\T [`*`file_name`*`]`

  By using the `--tee` option when you invoke `mysql`, you can log statements and their output. All the data displayed on the screen is appended into a given file. This can be very useful for debugging purposes also. `mysql` flushes results to the file after each statement, just before it prints its next prompt. Tee functionality works only in interactive mode.

  You can enable this feature interactively with the `tee` command. Without a parameter, the previous file is used. The `tee` file can be disabled with the `notee` command. Executing `tee` again re-enables logging.

- `use db_name`, `\u db_name`

  Use *db_name* as the default database.

- `warnings`, `\W`

  Enable display of warnings after each statement (if there are any).

Here are a few tips about the `pager` command:

- You can use it to write to a file and the results go only to the file:

```
mysql> pager cat > /tmp/log.txt
```

  You can also pass any options for the program that you want to use as your pager:

```
mysql> pager less -n -i -S
```

- In the preceding example, note the `-S` option. You may find it very useful for browsing wide query results. Sometimes a very wide result set is difficult to read on the screen. The `-S` option to `less` can make the result set much more readable because you can scroll it horizontally using the left-arrow and right-arrow keys. You can also use `-S` interactively within `less` to switch the horizontal-browse mode on and off. For more information, read the `less` manual page:

```
shell> man less
```

- The `-F` and `-X` options may be used with `less` to cause it to exit if output fits on one screen, which is convenient when no scrolling is necessary:

```
mysql> pager less -n -i -S -F -X
```

- You can specify very complex pager commands for handling query output:

```
mysql> pager cat | tee /dr1/tmp/res.txt \
         | tee /dr2/tmp/res2.txt | less -n -i -S
```

  In this example, the command would send query results to two files in two different directories on two different file systems mounted on `/dr1` and `/dr2`, yet still display the results onscreen using `less`.

You can also combine the `tee` and `pager` functions. Have a `tee` file enabled and `pager` set to `less`, and you are able to browse the results using the `less` program and still have everything appended into a file the same time. The difference between the Unix `tee` used with the `pager` command and the `mysql` built-in `tee` command is that the built-in `tee` works even if you do not have the Unix `tee` available. The built-in `tee` also logs everything that is printed on the screen, whereas the Unix `tee` used with `pager` does not log quite that much. Additionally, `tee` file logging can be turned on and off interactively from within `mysql`. This is useful when you want to log some queries to a file, but not others.

The `prompt` command reconfigures the default `mysql>` prompt. The string for defining the prompt can contain the following special sequences.

Option	Description
`\C`	The current connection identifier
`\c`	A counter that increments for each statement you issue

Option	Description
\D	The full current date
\d	The default database
\h	The server host
\l	The current delimiter
\m	Minutes of the current time
\n	A newline character
\O	The current month in three-letter format (Jan, Feb, …)
\o	The current month in numeric format
\P	am/pm
\p	The current TCP/IP port or socket file
\R	The current time, in 24-hour military time (0–23)
\r	The current time, standard 12-hour time (1–12)
\S	Semicolon
\s	Seconds of the current time
\t	A tab character
\U	Your full *user_name@host_name* account name
\u	Your user name
\v	The server version
\w	The current day of the week in three-letter format (Mon, Tue, …)
\Y	The current year, four digits
\y	The current year, two digits
_	A space
\	A space (a space follows the backslash)
\'	Single quote
\"	Double quote
\\	A literal \ backslash character
x	*x*, for any "*x*" not listed above

You can set the prompt in several ways:

- *Use an environment variable.* You can set the MYSQL_PS1 environment variable to a prompt string. For example:

```
shell> export MYSQL_PS1="(\u@\h) [\d]> "
```

- *Use a command-line option.* You can set the --prompt option on the command line to mysql. For example:

```
shell> mysql --prompt="(\u@\h) [\d]> "
(user@host) [database]>
```

- *Use an option file.* You can set the prompt option in the [mysql] group of any MySQL option file, such as /etc/my.cnf or the .my.cnf file in your home directory. For example:

```
[mysql]
prompt=(\\u@\\h) [\\d]>\\_
```

In this example, note that the backslashes are doubled. If you set the prompt using the `prompt` option in an option file, it is advisable to double the backslashes when using the special prompt options. There is some overlap in the set of permissible prompt options and the set of special escape sequences that are recognized in option files. (The rules for escape sequences in option files are listed in Section 4.2.2.2, "Using Option Files".) The overlap may cause you problems if you use single backslashes. For example, `\s` is interpreted as a space rather than as the current seconds value. The following example shows how to define a prompt within an option file to include the current time in *`hh:mm:ss>`* format:

```
[mysql]
prompt="\\r:\\m:\\s> "
```

- *Set the prompt interactively.* You can change your prompt interactively by using the `prompt` (or `\R`) command. For example:

```
mysql> prompt (\u@\h) [\d]>\_
PROMPT set to '(\u@\h) [\d]>\_'
(user@host) [database]>
(user@host) [database]> prompt
Returning to default PROMPT of mysql>
mysql>
```

4.5.1.3 mysql Client Logging

The `mysql` client can do these types of logging for statements executed interactively:

- On Unix, `mysql` writes the statements to a history file. By default, this file is named `.mysql_history` in your home directory. To specify a different file, set the value of the `MYSQL_HISTFILE` environment variable.
- On all platforms, if the `--syslog` option is given, `mysql` writes the statements to the system logging facility. On Unix, this is `syslog`; on Windows, it is the Windows Event Log. The destination where logged messages appear is system dependent. On Linux, the destination is often the `/var/log/messages` file.

The following discussion describes characteristics that apply to all logging types and provides information specific to each logging type.

- How Logging Occurs
- Controlling the History File
- syslog Logging Characteristics

How Logging Occurs

For each enabled logging destination, statement logging occurs as follows:

- Statements are logged only when executed interactively. Statements are noninteractive, for example, when read from a file or a pipe. It is also possible to suppress statement logging by using the `--batch` or `--execute` option.
- Statements are ignored and not logged if they match any pattern in the "ignore" list. This list is described later.

- mysql logs each nonignored, nonempty statement line individually.
- If a nonignored statement spans multiple lines (not including the terminating delimiter), mysql concatenates the lines to form the complete statement, maps newlines to spaces, and logs the result, plus a delimiter.

Consequently, an input statement that spans multiple lines can be logged twice. Consider this input:

```
mysql> SELECT
    -> 'Today is'
    -> ,
    -> CURDATE()
    -> ;
```

In this case, mysql logs the “SELECT”, “'Today is'”, “,”, “CURDATE()”, and “;” lines as it reads them. It also logs the complete statement, after mapping SELECT\n'Today is'\n,\nCURDATE() to SELECT 'Today is' , CURDATE(), plus a delimiter. Thus, these lines appear in logged output:

```
SELECT
'Today is'
,
CURDATE()
;
SELECT 'Today is' , CURDATE();
```

mysql ignores for logging purposes statements that match any pattern in the “ignore” list. By default, the pattern list is "*IDENTIFIED*:*PASSWORD*", to ignore statements that refer to passwords. Pattern matching is not case-sensitive. Within patterns, two characters are special:

- ? matches any single character.
- * matches any sequence of zero or more characters.

To specify additional patterns, use the --histignore option or set the MYSQL_HISTIGNORE environment variable. (If both are specified, the option value takes precedence.) The value should be a list of one or more colon-separated patterns, which are appended to the default pattern list.

Patterns specified on the command line might need to be quoted or escaped to prevent your command interpreter from treating them specially. For example, to suppress logging for UPDATE and DELETE statements in addition to statements that refer to passwords, invoke mysql like this:

```
shell> mysql --histignore="*UPDATE*:*DELETE*"
```

Controlling the History File

The .mysql_history file should be protected with a restrictive access mode because sensitive information might be written to it, such as the text of SQL statements that contain passwords. See Section 6.1.2.1, “End-User Guidelines for Password Security”. Statements in the file are accessible from the mysql client when the **up-arrow** key is used to recall the history. See Disabling Interactive History.

If you do not want to maintain a history file, first remove .mysql_history if it exists. Then use either of the following techniques to prevent it from being created again:

- Set the MYSQL_HISTFILE environment variable to /dev/null. To cause this setting to take effect each time you log in, put it in one of your shell's startup files.
- Create .mysql_history as a symbolic link to /dev/null; this need be done only once:

```
shell> ln -s /dev/null $HOME/.mysql_history
```

syslog Logging Characteristics

If the `--syslog` option is given, `mysql` writes interactive statements to the system logging facility. Message logging has the following characteristics.

Logging occurs at the “information” level. This corresponds to the `LOG_INFO` priority for `syslog` on Unix/Linux `syslog` capability and to `EVENTLOG_INFORMATION_TYPE` for the Windows Event Log. Consult your system documentation for configuration of your logging capability.

Message size is limited to 1024 bytes.

Messages consist of the identifier `MysqlClient` followed by these values:

- `SYSTEM_USER`

  The operating system user name (login name) or `--` if the user is unknown.

- `MYSQL_USER`

  The MySQL user name (specified with the `--user` option) or `--` if the user is unknown.

- `CONNECTION_ID`:

  The client connection identifier. This is the same as the `CONNECTION_ID()` function value within the session.

- `DB_SERVER`

  The server host or `--` if the host is unknown.

- `DB`

  The default database or `--` if no database has been selected.

- `QUERY`

  The text of the logged statement.

Here is a sample of output generated on Linux by using `--syslog`. This output is formatted for readability; each logged message actually takes a single line.

```
Mar  7 12:39:25 myhost MysqlClient[20824]:
  SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,
  DB_SERVER:'127.0.0.1', DB:'--', QUERY:'USE test;'
Mar  7 12:39:28 myhost MysqlClient[20824]:
  SYSTEM_USER:'oscar', MYSQL_USER:'my_oscar', CONNECTION_ID:23,
  DB_SERVER:'127.0.0.1', DB:'test', QUERY:'SHOW TABLES;'
```

4.5.1.4 mysql Client Server-Side Help

```
mysql> help search_string
```

If you provide an argument to the `help` command, `mysql` uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. The proper operation of this command requires that the help tables in the `mysql` database be initialized with help topic information (see Section 5.1.14, “Server-Side Help Support”).

If there is no match for the search string, the search fails:

```
mysql> help me

Nothing found
Please try to run 'help contents' for a list of all accessible topics
```

Use `help contents` to see a list of the help categories:

```
mysql> help contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the
following categories:
   Account Management
   Administration
   Data Definition
   Data Manipulation
   Data Types
   Functions
   Functions and Modifiers for Use with GROUP BY
   Geographic Features
   Language Structure
   Plugins
   Storage Engines
   Stored Routines
   Table Maintenance
   Transactions
   Triggers
```

If the search string matches multiple items, `mysql` shows a list of matching topics:

```
mysql> help logs
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following topics:
   SHOW
   SHOW BINARY LOGS
   SHOW ENGINE
   SHOW LOGS
```

Use a topic as the search string to see the help entry for that topic:

```
mysql> help show binary logs
Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS

Lists the binary log files on the server. This statement is used as
part of the procedure described in [purge-binary-logs], that shows how
to determine which logs can be purged.

mysql> SHOW BINARY LOGS;
+---------------+-----------+-----------+
| Log_name      | File_size | Encrypted |
+---------------+-----------+-----------+
| binlog.000015 |    724935 | Yes       |
| binlog.000016 |    733481 | Yes       |
+---------------+-----------+-----------+
```

The search string can contain the wildcard characters `%` and `_`. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. For example, `HELP rep%` returns a list of topics that begin with `rep`:

```
mysql> HELP rep%
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following
topics:
   REPAIR TABLE
   REPEAT FUNCTION
   REPEAT LOOP
   REPLACE
   REPLACE FUNCTION
```

4.5.1.5 Executing SQL Statements from a Text File

The `mysql` client typically is used interactively, like this:

```
shell> mysql db_name
```

However, it is also possible to put your SQL statements in a file and then tell `mysql` to read its input from that file. To do so, create a text file *`text_file`* that contains the statements you wish to execute. Then invoke `mysql` as shown here:

```
shell> mysql db_name < text_file
```

If you place a `USE` *`db_name`* statement as the first statement in the file, it is unnecessary to specify the database name on the command line:

```
shell> mysql < text_file
```

If you are already running `mysql`, you can execute an SQL script file using the `source` command or `\.` command:

```
mysql> source file_name
mysql> \. file_name
```

Sometimes you may want your script to display progress information to the user. For this you can insert statements like this:

```
SELECT '<info_to_display>' AS ' ';
```

The statement shown outputs `<info_to_display>`.

You can also invoke `mysql` with the `--verbose` option, which causes each statement to be displayed before the result that it produces.

`mysql` ignores Unicode byte order mark (BOM) characters at the beginning of input files. Previously, it read them and sent them to the server, resulting in a syntax error. Presence of a BOM does not cause `mysql` to change its default character set. To do that, invoke `mysql` with an option such as `--default-character-set=utf8`.

For more information about batch mode, see Section 3.5, “Using mysql in Batch Mode”.

4.5.1.6 mysql Client Tips

This section provides information about techniques for more effective use of `mysql` and about `mysql` operational behavior.

- Input-Line Editing
- Disabling Interactive History
- Unicode Support on Windows
- Displaying Query Results Vertically
- Using Safe-Updates Mode (--safe-updates)
- Disabling mysql Auto-Reconnect
- mysql Client Parser Versus Server Parser

Input-Line Editing

`mysql` supports input-line editing, which enables you to modify the current input line in place or recall previous input lines. For example, the **left-arrow** and **right-arrow** keys move horizontally within the current input line, and the **up-arrow** and **down-arrow** keys move up and down through the set of previously entered lines. **Backspace** deletes the character before the cursor and typing new characters enters them at the cursor position. To enter the line, press **Enter**.

On Windows, the editing key sequences are the same as supported for command editing in console windows. On Unix, the key sequences depend on the input library used to build `mysql` (for example, the `libedit` or `readline` library).

Documentation for the `libedit` and `readline` libraries is available online. To change the set of key sequences permitted by a given input library, define key bindings in the library startup file. This is a file in your home directory: `.editrc` for `libedit` and `.inputrc` for `readline`.

For example, in `libedit`, **Control+W** deletes everything before the current cursor position and **Control+U** deletes the entire line. In `readline`, **Control+W** deletes the word before the cursor and **Control+U** deletes everything before the current cursor position. If `mysql` was built using `libedit`, a user who prefers the `readline` behavior for these two keys can put the following lines in the `.editrc` file (creating the file if necessary):

```
bind "^W" ed-delete-prev-word
bind "^U" vi-kill-line-prev
```

To see the current set of key bindings, temporarily put a line that says only `bind` at the end of `.editrc`. `mysql` will show the bindings when it starts.

Disabling Interactive History

The **up-arrow** key enables you to recall input lines from current and previous sessions. In cases where a console is shared, this behavior may be unsuitable. `mysql` supports disabling the interactive history partially or fully, depending on the host platform.

On Windows, the history is stored in memory. **Alt+F7** deletes all input lines stored in memory for the current history buffer. It also deletes the list of sequential numbers in front of the input lines displayed with **F7** and recalled (by number) with **F9**. New input lines entered after you press **Alt+F7** repopulate the current history buffer. Clearing the buffer does not prevent logging to the Windows Event Viewer, if the `--syslog` option was used to start `mysql`. Closing the console window also clears the current history buffer.

To disable interactive history on Unix, first delete the `.mysql_history` file, if it exists (previous entries are recalled otherwise). Then start `mysql` with the `--histignore="*"` option to ignore all new input lines. To re-enable the recall (and logging) behavior, restart `mysql` without the option.

If you prevent the .mysql_history file from being created (see Controlling the History File) and use --histignore="*" to start the mysql client, the interactive history recall facility is disabled fully. Alternatively, if you omit the --histignore option, you can recall the input lines entered during the current session.

Unicode Support on Windows

Windows provides APIs based on UTF-16LE for reading from and writing to the console; the mysql client for Windows is able to use these APIs. The Windows installer creates an item in the MySQL menu named MySQL command line client - Unicode. This item invokes the mysql client with properties set to communicate through the console to the MySQL server using Unicode.

To take advantage of this support manually, run mysql within a console that uses a compatible Unicode font and set the default character set to a Unicode character set that is supported for communication with the server:

1. Open a console window.
2. Go to the console window properties, select the font tab, and choose Lucida Console or some other compatible Unicode font. This is necessary because console windows start by default using a DOS raster font that is inadequate for Unicode.
3. Execute mysql.exe with the --default-character-set=utf8 (or utf8mb4) option. This option is necessary because utf16le is one of the character sets that cannot be used as the client character set. See Impermissible Client Character Sets.

With those changes, mysql will use the Windows APIs to communicate with the console using UTF-16LE, and communicate with the server using UTF-8. (The menu item mentioned previously sets the font and character set as just described.)

To avoid those steps each time you run mysql, you can create a shortcut that invokes mysql.exe. The shortcut should set the console font to Lucida Console or some other compatible Unicode font, and pass the --default-character-set=utf8 (or utf8mb4) option to mysql.exe.

Alternatively, create a shortcut that only sets the console font, and set the character set in the [mysql] group of your my.ini file:

```
[mysql]
default-character-set=utf8
```

Displaying Query Results Vertically

Some query results are much more readable when displayed vertically, instead of in the usual horizontal table format. Queries can be displayed vertically by terminating the query with \G instead of a semicolon. For example, longer text values that include newlines often are much easier to read with vertical output:

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
*************************** 1. row ***************************
  msg_nro: 3068
     date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Jones
    reply: jones@example.com
  mail_to: "John Smith" <smith@example.com>
      sbj: UTF-8
      txt: >>>>> "John" == John Smith writes:

John> Hi.  I think this is a good idea.  Is anyone familiar
```

```
John> with UTF-8 or Unicode? Otherwise, I'll put this on my
John> TODO list and see what happens.

Yes, please do that.

Regards,
Jones
     file: inbox-jani-1
     hash: 190402944
1 row in set (0.09 sec)
```

Using Safe-Updates Mode (--safe-updates)

For beginners, a useful startup option is `--safe-updates` (or `--i-am-a-dummy`, which has the same effect). Safe-updates mode is helpful for cases when you might have issued an `UPDATE` or `DELETE` statement but forgotten the `WHERE` clause indicating which rows to modify. Normally, such statements update or delete all rows in the table. With `--safe-updates`, you can modify rows only by specifying the key values that identify them, or a `LIMIT` clause, or both. This helps prevent accidents. Safe-updates mode also restricts `SELECT` statements that produce (or are estimated to produce) very large result sets.

The `--safe-updates` option causes `mysql` to execute the following statement when it connects to the MySQL server, to set the session values of the `sql_safe_updates`, `sql_select_limit`, and `max_join_size` system variables:

```
SET sql_safe_updates=1, sql_select_limit=1000, max_join_size=1000000;
```

The `SET` statement affects statement processing as follows:

- Enabling `sql_safe_updates` causes `UPDATE` and `DELETE` statements to produce an error if they do not specify a key constraint in the `WHERE` clause, or provide a `LIMIT` clause, or both. For example:

  ```
  UPDATE tbl_name SET not_key_column=val WHERE key_column=val;

  UPDATE tbl_name SET not_key_column=val LIMIT 1;
  ```

- Setting `sql_select_limit` to 1,000 causes the server to limit all `SELECT` result sets to 1,000 rows unless the statement includes a `LIMIT` clause.

- Setting `max_join_size` to 1,000,000 causes multiple-table `SELECT` statements to produce an error if the server estimates it must examine more than 1,000,000 row combinations.

To specify result set limits different from 1,000 and 1,000,000, you can override the defaults by using the `--select-limit` and `--max-join-size` options when you invoke `mysql`:

```
mysql --safe-updates --select-limit=500 --max-join-size=10000
```

It is possible for `UPDATE` and `DELETE` statements to produce an error in safe-updates mode even with a key specified in the `WHERE` clause, if the optimizer decides not to use the index on the key column:

- Range access on the index cannot be used if memory usage exceeds that permitted by the `range_optimizer_max_mem_size` system variable. The optimizer then falls back to a table scan. See Limiting Memory Use for Range Optimization.

- If key comparisons require type conversion, the index may not be used (see Section 8.3.1, “How MySQL Uses Indexes”). Suppose that an indexed string column `c1` is compared to a numeric value using `WHERE c1 = 2222`. For such comparisons, the string value is converted to a number and the operands are compared numerically (see Section 12.2, “Type Conversion in Expression Evaluation”), preventing use of the index. If safe-updates mode is enabled, an error occurs.

As of MySQL 8.0.13, safe-updates mode also includes these behaviors:

- EXPLAIN with UPDATE and DELETE statements does not produce safe-updates errors. This enables use of EXPLAIN plus SHOW WARNINGS to see why an index is not used, which can be helpful in cases such as when a range_optimizer_max_mem_size violation or type conversion occurs and the optimizer does not use an index even though a key column was specified in the WHERE clause.
- When a safe-updates error occurs, the error message includes the first diagnostic that was produced, to provide information about the reason for failure. For example, the message may indicate that the range_optimizer_max_mem_size value was exceeded or type conversion occurred, either of which can preclude use of an index.
- For multiple-table deletes and updates, an error is produced with safe updates enabled only if any target table uses a table scan.

Disabling mysql Auto-Reconnect

If the mysql client loses its connection to the server while sending a statement, it immediately and automatically tries to reconnect once to the server and send the statement again. However, even if mysql succeeds in reconnecting, your first connection has ended and all your previous session objects and settings are lost: temporary tables, the autocommit mode, and user-defined and session variables. Also, any current transaction rolls back. This behavior may be dangerous for you, as in the following example where the server was shut down and restarted between the first and second statements without you knowing it:

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
+------+
| a    |
+------+
| NULL |
+------+
1 row in set (0.05 sec)
```

The @a user variable has been lost with the connection, and after the reconnection it is undefined. If it is important to have mysql terminate with an error if the connection has been lost, you can start the mysql client with the --skip-reconnect option.

For more information about auto-reconnect and its effect on state information when a reconnection occurs, see Section 28.7.28, “C API Automatic Reconnection Control”.

mysql Client Parser Versus Server Parser

The mysql client uses a parser on the client side that is not a duplicate of the complete parser used by the mysqld server on the server side. This can lead to differences in treatment of certain constructs. Examples:

- The server parser treats strings delimited by " characters as identifiers rather than as plain strings if the ANSI_QUOTES SQL mode is enabled.

The mysql client parser does not take the ANSI_QUOTES SQL mode into account. It treats strings delimited by ", ', and ` characters the same, regardless of whether ANSI_QUOTES is enabled.

- Within /*! ... */ and /*+ ... */ comments, the mysql client parser interprets short-form mysql commands. The server parser does not interpret them because these commands have no meaning on the server side.

  If it is desirable for mysql not to interpret short-form commands within comments, a partial workaround is to use the --binary-mode option, which causes all mysql commands to be disabled except \C and \d in noninteractive mode (for input piped to mysql or loaded using the source command).

4.5.2 mysqladmin — Client for Administering a MySQL Server

mysqladmin is a client for performing administrative operations. You can use it to check the server's configuration and current status, to create and drop databases, and more.

Invoke mysqladmin like this:

```
shell> mysqladmin [options] command [command-arg] [command [command-arg]] ...
```

mysqladmin supports the following commands. Some of the commands take an argument following the command name.

- create *db_name*

  Create a new database named *db_name*.

- debug

  Tell the server to write debug information to the error log. The connected user must have the SUPER privilege. Format and content of this information is subject to change.

  This includes information about the Event Scheduler. See Section 24.4.5, “Event Scheduler Status”.

- drop *db_name*

  Delete the database named *db_name* and all its tables.

- extended-status

  Display the server status variables and their values.

- flush-hosts

  Flush all information in the host cache. See Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

- flush-logs [*log_type* ...]

  Flush all logs.

  The mysqladmin flush-logs command permits optional log types to be given, to specify which logs to flush. Following the flush-logs command, you can provide a space-separated list of one or more of the following log types: binary, engine, error, general, relay, slow. These correspond to the log types that can be specified for the FLUSH LOGS SQL statement.

- flush-privileges

Reload the grant tables (same as `reload`).

- `flush-status`

  Clear status variables.

- `flush-tables`

  Flush all tables.

- `flush-threads`

  Flush the thread cache.

- `kill id,id,...`

  Kill server threads. If multiple thread ID values are given, there must be no spaces in the list.

  To kill threads belonging to other users, the connected user must have the `CONNECTION_ADMIN` or `SUPER` privilege.

- `password new_password`

  Set a new password. This changes the password to *`new_password`* for the account that you use with `mysqladmin` for connecting to the server. Thus, the next time you invoke `mysqladmin` (or any other client program) using the same account, you will need to specify the new password.

Warning

Setting a password using `mysqladmin` should be considered *insecure*. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

If the *`new_password`* value contains spaces or other characters that are special to your command interpreter, you need to enclose it within quotation marks. On Windows, be sure to use double quotation marks rather than single quotation marks; single quotation marks are not stripped from the password, but rather are interpreted as part of the password. For example:

```
shell> mysqladmin password "my new password"
```

The new password can be omitted following the `password` command. In this case, `mysqladmin` prompts for the password value, which enables you to avoid specifying the password on the command line. Omitting the password value should be done only if `password` is the final command on the `mysqladmin` command line. Otherwise, the next argument is taken as the password.

Caution

Do not use this command used if the server was started with the `--skip-grant-tables` option. No password change will be applied. This is true even if you precede the `password` command with `flush-privileges` on the same command line to re-enable the grant tables because the flush operation occurs

> after you connect. However, you can use `mysqladmin flush-privileges` to re-enable the grant table and then use a separate `mysqladmin password` command to change the password.

- `ping`

  Check whether the server is available. The return status from `mysqladmin` is 0 if the server is running, 1 if it is not. This is 0 even in case of an error such as `Access denied`, because this means that the server is running but refused the connection, which is different from the server not running.

- `processlist`

  Show a list of active server threads. This is like the output of the `SHOW PROCESSLIST` statement. If the `--verbose` option is given, the output is like that of `SHOW FULL PROCESSLIST`. (See Section 13.7.7.29, “SHOW PROCESSLIST Syntax”.)

- `reload`

  Reload the grant tables.

- `refresh`

  Flush all tables and close and open log files.

- `shutdown`

  Stop the server.

- `start-slave`

  Start replication on a slave server.

- `status`

  Display a short server status message.

- `stop-slave`

  Stop replication on a slave server.

- `variables`

  Display the server system variables and their values.

- `version`

  Display version information from the server.

All commands can be shortened to any unique prefix. For example:

```
shell> mysqladmin proc stat
+----+-------+-----------+----+---------+------+-------+------------------+
| Id | User  | Host      | db | Command | Time | State | Info             |
+----+-------+-----------+----+---------+------+-------+------------------+
| 51 | jones | localhost |    | Query   | 0    |       | show processlist |
+----+-------+-----------+----+---------+------+-------+------------------+
Uptime: 1473624  Threads: 1  Questions: 39487
Slow queries: 0  Opens: 541  Flush tables: 1
Open tables: 19  Queries per second avg: 0.0268
```

The mysqladmin status command result displays the following values:

- Uptime

  The number of seconds the MySQL server has been running.

- Threads

  The number of active threads (clients).

- Questions

  The number of questions (queries) from clients since the server was started.

- Slow queries

  The number of queries that have taken more than long_query_time seconds. See Section 5.4.5, “The Slow Query Log”.

- Opens

  The number of tables the server has opened.

- Flush tables

  The number of flush-*, refresh, and reload commands the server has executed.

- Open tables

  The number of tables that currently are open.

If you execute mysqladmin shutdown when connecting to a local server using a Unix socket file, mysqladmin waits until the server's process ID file has been removed, to ensure that the server has stopped properly.

mysqladmin supports the following options, which can be specified on the command line or in the [mysqladmin] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.12 mysqladmin Options

Option Name	Description	Introduced	Deprecated	Removed
--bind-address	Use specified network interface to connect to MySQL Server			
--compress	Compress all information sent between client and server		8.0.18	
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18		
--connect_timeout	Number of seconds before connection timeout			
--count	Number of iterations to make for repeated command execution			
--debug	Write debugging log			
--debug-check	Print debugging information when program exits			
--debug-info	Print debugging information, memory, and CPU statistics when program exits			

Option Name	Description	Introduced	Deprecated	Removed
--default-auth	Authentication plugin to use			
--default-character-set	Specify default character set			
--defaults-extra-file	Read named option file in addition to usual option files			
--defaults-file	Read only named option file			
--defaults-group-suffix	Option group suffix value			
--enable-cleartext-plugin	Enable cleartext authentication plugin			
--force	Continue even if an SQL error occurs			
--get-server-public-key	Request RSA public key from server	8.0.3		
--help	Display help message and exit			
--host	Host on which MySQL server is located			
--login-path	Read login path options from .mylogin.cnf			
--no-beep	Do not beep when errors occur			
--no-defaults	Read no option files			
--password	Password to use when connecting to server			
--pipe	Connect to server using named pipe (Windows only)			
--plugin-dir	Directory where plugins are installed			
--port	TCP/IP port number for connection			
--print-defaults	Print default options			
--protocol	Connection protocol to use			
--relative	Show the difference between the current and previous values when used with the --sleep option			
--secure-auth	Do not send passwords to server in old (pre-4.1) format			8.0.3
--server-public-key-path	Path name to file containing RSA public key	8.0.4		
--shared-memory-base-name	Name of shared memory to use for shared-memory connections			
--show-warnings	Show warnings after statement execution			
--shutdown_timeout	The maximum number of seconds to wait for server shutdown			
--silent	Silent mode			
--sleep	Execute commands repeatedly, sleeping for delay seconds in between			
--socket	Unix socket file or Windows named pipe to use			
--ssl-ca	File that contains list of trusted SSL Certificate Authorities			
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files			
--ssl-cert	File that contains X.509 certificate			

Option Name	Description	Introduced	Deprecated	Removed
--ssl-cipher	Permissible ciphers for connection encryption			
--ssl-crl	File that contains certificate revocation lists			
--ssl-crlpath	Directory that contains certificate revocation-list files			
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11		
--ssl-key	File that contains X.509 key			
--ssl-mode	Desired security state of connection to server			
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16		
--tls-version	Permissible TLS protocols for encrypted connections			
--user	MySQL user name to use when connecting to server			
--verbose	Verbose mode			
--version	Display version information and exit			
--vertical	Print query output rows vertically (one line per column value)			
--wait	If the connection cannot be established, wait and retry instead of aborting			
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18		

- `--help`, `-?`

  Display a help message and exit.

- `--bind-address=`*`ip_address`*

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

  As of MySQL 8.0.18, this option is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `--compression-algorithms=`*`value`*

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. The default value is `uncompressed`.

  For more information, see Section 4.2.6, “Connection Compression Control”.

This option was added in MySQL 8.0.18.

- `--count=N`, `-c N`

  The number of iterations to make for repeated command execution if the `--sleep` option is given.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysqladmin.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.15, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqladmin` normally reads the `[client]` and `[mysqladmin]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqladmin` also reads the `[client_other]` and `[mysqladmin_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.)

- `--force`, `-f`

  Do not ask for confirmation for the `drop db_name` command. With multiple commands, continue even if an error occurs.

- `--get-server-public-key`

  Request from the server the public key required for RSA key pair-based password exchange. This option applies to clients that authenticate with the `caching_sha2_password` authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  For information about the `caching_sha2_password` plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-beep`, `-b`

  Suppress the warning beep that is emitted by default for errors such as a failure to connect to the server.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--password[=password]`, `-p[password]`

The password of the MySQL account used for connecting to the server. The password value is optional. If not given, `mysqladmin` prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

To explicitly specify that there is no password and that `mysqladmin` should not prompt for one, use the `--skip-password` option.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--plugin-dir=`*`dir_name`*

  The directory in which to look for plugins. Specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysqladmin` does not find it. See Section 6.2.17, “Pluggable Authentication”.

- `--port=`*`port_num`*, `-P` *`port_num`*

  For TCP/IP connections, the port number to use.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

- `--relative`, `-r`

  Show the difference between the current and previous values when used with the `--sleep` option. This option works only with the `extended-status` command.

- `--secure-auth`

  This option was removed in MySQL 8.0.3.

- `--server-public-key-path=`*`file_name`*

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-

based password exchange is not used, as is the case when the client connects to the server using a secure connection.

If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

For `sha256_password`, this option applies only if MySQL was built using OpenSSL.

For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--show-warnings`

  Show warnings resulting from execution of statements sent to the server.

- `--silent`, `-s`

  Exit silently if a connection to the server cannot be established.

- `--sleep=delay`, `-i delay`

  Execute commands repeatedly, sleeping for *`delay`* seconds in between. The `--count` option determines the number of iterations. If `--count` is not given, `mysqladmin` executes commands indefinitely until interrupted.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--ssl-fips-mode={OFF|ON|STRICT}`

  Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These `--ssl-fips-mode` values are permitted:

  - `OFF`: Disable FIPS mode.

- `ON`: Enable FIPS mode.
- `STRICT`: Enable “strict” FIPS mode.

Note

If the OpenSSL FIPS Object Module is not available, the only permitted value for `--ssl-fips-mode` is `OFF`. In this case, setting `--ssl-fips-mode` to `ON` or `STRICT` causes the client to produce a warning at startup and to operate in non-FIPS mode.

- `--tls-ciphersuites=`*`ciphersuite_list`*

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- `--tls-version=`*`protocol_list`*

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The user name of the MySQL account to use for connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

- `--version`, `-V`

  Display version information and exit.

- `--vertical`, `-E`

  Print output vertically. This is similar to `--relative`, but prints output vertically.

- `--wait[=`*`count`*`]`, `-w[`*`count`*`]`

  If the connection cannot be established, wait and retry instead of aborting. If a *`count`* value is given, it indicates the number of times to retry. The default is one time.

- `--zstd-compression-level=`*`level`*

  The compression level to use for connections to the server that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. The compression level setting has no effect on connections that do not use `zstd` compression.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

You can also set the following variables by using *--var_name=value*.

- connect_timeout

  The maximum number of seconds before connection timeout. The default value is 43200 (12 hours).

- shutdown_timeout

  The maximum number of seconds to wait for server shutdown. The default value is 3600 (1 hour).

4.5.3 mysqlcheck — A Table Maintenance Program

The mysqlcheck client performs table maintenance: It checks, repairs, optimizes, or analyzes tables.

Each table is locked and therefore unavailable to other sessions while it is being processed, although for check operations, the table is locked with a READ lock only (see Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”, for more information about READ and WRITE locks). Table maintenance operations can be time-consuming, particularly for large tables. If you use the --databases or --all-databases option to process all tables in one or more databases, an invocation of mysqlcheck might take a long time. (This is also true for the MySQL upgrade procedure if it determines that table checking is needed because it processes tables the same way.)

mysqlcheck must be used when the mysqld server is running, which means that you do not have to stop the server to perform table maintenance.

mysqlcheck uses the SQL statements CHECK TABLE, REPAIR TABLE, ANALYZE TABLE, and OPTIMIZE TABLE in a convenient way for the user. It determines which statements to use for the operation you want to perform, and then sends the statements to the server to be executed. For details about which storage engines each statement works with, see the descriptions for those statements in Section 13.7.3, “Table Maintenance Statements”.

All storage engines do not necessarily support all four maintenance operations. In such cases, an error message is displayed. For example, if test.t is an MEMORY table, an attempt to check it produces this result:

```
shell> mysqlcheck test t
test.t
note     : The storage engine for the table doesn't support check
```

If mysqlcheck is unable to repair a table, see Section 2.11.13, “Rebuilding or Repairing Tables or Indexes” for manual table repair strategies. This will be the case, for example, for InnoDB tables, which can be checked with CHECK TABLE, but not repaired with REPAIR TABLE.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

There are three general ways to invoke mysqlcheck:

```
shell> mysqlcheck [options] db_name [tbl_name ...]
shell> mysqlcheck [options] --databases db_name ...
shell> mysqlcheck [options] --all-databases
```

If you do not name any tables following *db_name* or if you use the --databases or --all-databases option, entire databases are checked.

mysqlcheck has a special feature compared to other client programs. The default behavior of checking tables (--check) can be changed by renaming the binary. If you want to have a tool that repairs tables by default, you should just make a copy of mysqlcheck named mysqlrepair, or make a symbolic link to mysqlcheck named mysqlrepair. If you invoke mysqlrepair, it repairs tables.

The names shown in the following table can be used to change mysqlcheck default behavior.

Command	Meaning
mysqlrepair	The default option is --repair
mysqlanalyze	The default option is --analyze
mysqloptimize	The default option is --optimize

mysqlcheck supports the following options, which can be specified on the command line or in the [mysqlcheck] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.13 mysqlcheck Options

Option Name	Description	Introduced	Deprecated	Removed
--all-databases	Check all tables in all databases			
--all-in-1	Execute a single statement for each database that names all the tables from that database			
--analyze	Analyze the tables			
--auto-repair	If a checked table is corrupted, automatically fix it			
--bind-address	Use specified network interface to connect to MySQL Server			
--character-sets-dir	Directory where character sets are installed			
--check	Check the tables for errors			
--check-only-changed	Check only tables that have changed since the last check			
--check-upgrade	Invoke CHECK TABLE with the FOR UPGRADE option			
--compress	Compress all information sent between client and server		8.0.18	
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18		
--databases	Interpret all arguments as database names			
--debug	Write debugging log			
--debug-check	Print debugging information when program exits			
--debug-info	Print debugging information, memory, and CPU statistics when program exits			
--default-auth	Authentication plugin to use			
--default-character-set	Specify default character set			
--defaults-extra-file	Read named option file in addition to usual option files			

Option Name	Description	Introduced	Deprecated	Removed
--defaults-file	Read only named option file			
--defaults-group-suffix	Option group suffix value			
--enable-cleartext-plugin	Enable cleartext authentication plugin			
--extended	Check and repair tables			
--fast	Check only tables that have not been closed properly			
--force	Continue even if an SQL error occurs			
--get-server-public-key	Request RSA public key from server	8.0.3		
--help	Display help message and exit			
--host	Host on which MySQL server is located			
--login-path	Read login path options from .mylogin.cnf			
--medium-check	Do a check that is faster than an --extended operation			
--no-defaults	Read no option files			
--optimize	Optimize the tables			
--password	Password to use when connecting to server			
--pipe	Connect to server using named pipe (Windows only)			
--plugin-dir	Directory where plugins are installed			
--port	TCP/IP port number for connection			
--print-defaults	Print default options			
--protocol	Connection protocol to use			
--quick	The fastest method of checking			
--repair	Perform a repair that can fix almost anything except unique keys that are not unique			
--secure-auth	Do not send passwords to server in old (pre-4.1) format			8.0.3
--server-public-key-path	Path name to file containing RSA public key	8.0.4		
--shared-memory-base-name	Name of shared memory to use for shared-memory connections			
--silent	Silent mode			
--skip-database	Omit this database from performed operations			
--socket	Unix socket file or Windows named pipe to use			
--ssl-ca	File that contains list of trusted SSL Certificate Authorities			
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files			
--ssl-cert	File that contains X.509 certificate			
--ssl-cipher	Permissible ciphers for connection encryption			
--ssl-crl	File that contains certificate revocation lists			

Option Name	Description	Introduced	Deprecated	Removed
--ssl-crlpath	Directory that contains certificate revocation-list files			
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11		
--ssl-key	File that contains X.509 key			
--ssl-mode	Desired security state of connection to server			
--tables	Overrides the --databases or -B option			
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16		
--tls-version	Permissible TLS protocols for encrypted connections			
--use-frm	For repair operations on MyISAM tables			
--user	MySQL user name to use when connecting to server			
--verbose	Verbose mode			
--version	Display version information and exit			
--write-binlog	Log ANALYZE, OPTIMIZE, REPAIR statements to binary log. --skip-write-binlog adds NO_WRITE_TO_BINLOG to these statements.			
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18		

- `--help`, `-?`

  Display a help message and exit.

- `--all-databases`, `-A`

  Check all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line, except that the `INFORMATION_SCHEMA` and `performance_schema` databases are not checked. They can be checked by explicitly naming them with the `--databases` option.

- `--all-in-1`, `-1`

  Instead of issuing a statement for each table, execute a single statement for each database that names all the tables from that database to be processed.

- `--analyze`, `-a`

  Analyze the tables.

- `--auto-repair`

  If a checked table is corrupted, automatically fix it. Any necessary repairs are done after all tables have been checked.

- `--bind-address=`*`ip_address`*

On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=dir_name`

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--check`, `-c`

  Check the tables for errors. This is the default operation.

- `--check-only-changed`, `-C`

  Check only tables that have changed since the last check or that have not been closed properly.

- `--check-upgrade`, `-g`

  Invoke `CHECK TABLE` with the `FOR UPGRADE` option to check tables for incompatibilities with the current version of the server.

- `--compress`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

  As of MySQL 8.0.18, this option is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `--compression-algorithms=value`

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. The default value is `uncompressed`.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

- `--databases`, `-B`

  Process all tables in the named databases. Normally, `mysqlcheck` treats the first name argument on the command line as a database name and any following names as table names. With this option, it treats all name arguments as database names.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.15, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqlcheck` normally reads the `[client]` and `[mysqlcheck]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlcheck` also reads the `[client_other]` and `[mysqlcheck_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--extended`, `-e`

  If you are using this option to check tables, it ensures that they are 100% consistent but takes a long time.

  If you are using this option to repair tables, it runs an extended repair that may not only take a long time to execute, but may produce a lot of garbage rows also!

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.)

- `--fast`, `-F`

  Check only tables that have not been closed properly.

- `--force`, `-f`

  Continue even if an SQL error occurs.

- `--get-server-public-key`

Request from the server the public key required for RSA key pair-based password exchange. This option applies to clients that authenticate with the caching_sha2_password authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

If --server-public-key-path=*file_name* is given and specifies a valid public key file, it takes precedence over --get-server-public-key.

For information about the caching_sha2_password plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- --host=*host_name*, -h *host_name*

  Connect to the MySQL server on the given host.

- --login-path=*name*

  Read options from the named login path in the .mylogin.cnf login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the mysql_config_editor utility. See Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --medium-check, -m

  Do a check that is faster than an --extended operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --optimize, -o

  Optimize the tables.

- --password[=*password*], -p[*password*]

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, mysqlcheck prompts for one. If given, there must be *no space* between --password= or -p and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

To explicitly specify that there is no password and that mysqlcheck should not prompt for one, use the --skip-password option.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysqlcheck does not find it. See Section 6.2.17, “Pluggable Authentication”.

- --port=*port_num*, -P *port_num*

  For TCP/IP connections, the port number to use.

- --print-defaults

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

- --quick, -q

  If you are using this option to check tables, it prevents the check from scanning the rows to check for incorrect links. This is the fastest check method.

  If you are using this option to repair tables, it tries to repair only the index tree. This is the fastest repair method.

- --repair, -r

  Perform a repair that can fix almost anything except unique keys that are not unique.

- --secure-auth

  This option was removed in MySQL 8.0.3.

- --server-public-key-path=*file_name*

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the sha256_password or caching_sha2_password authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

For `sha256_password`, this option applies only if MySQL was built using OpenSSL.

For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--silent`, `-s`

  Silent mode. Print only error messages.

- `--skip-database=db_name`

  Do not include the named database (case-sensitive) in the operations performed by `mysqlcheck`.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--ssl-fips-mode={OFF|ON|STRICT}`

  Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These `--ssl-fips-mode` values are permitted:

  - `OFF`: Disable FIPS mode.
  - `ON`: Enable FIPS mode.
  - `STRICT`: Enable “strict” FIPS mode.

  Note

  If the OpenSSL FIPS Object Module is not available, the only permitted value for `--ssl-fips-mode` is `OFF`. In this case, setting `--ssl-fips-mode` to `ON` or

> STRICT causes the client to produce a warning at startup and to operate in non-FIPS mode.

- --tables

  Override the --databases or -B option. All name arguments following the option are regarded as table names.

- --tls-ciphersuites=*ciphersuite_list*

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- --tls-version=*protocol_list*

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- --use-frm

  For repair operations on MyISAM tables, get the table structure from the data dictionary so that the table can be repaired even if the .MYI header is corrupted.

- --user=*user_name*, -u *user_name*

  The user name of the MySQL account to use for connecting to the server.

- --verbose, -v

  Verbose mode. Print information about the various stages of program operation.

- --version, -V

  Display version information and exit.

- --write-binlog

  This option is enabled by default, so that ANALYZE TABLE, OPTIMIZE TABLE, and REPAIR TABLE statements generated by mysqlcheck are written to the binary log. Use --skip-write-binlog to cause NO_WRITE_TO_BINLOG to be added to the statements so that they are not logged. Use the --skip-write-binlog when these statements should not be sent to replication slaves or run when using the binary logs for recovery from backup.

- --zstd-compression-level=*level*

  The compression level to use for connections to the server that use the zstd compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default zstd compression level is 3. The compression level setting has no effect on connections that do not use zstd compression.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

4.5.4 mysqldump — A Database Backup Program

The mysqldump client utility performs logical backups, producing a set of SQL statements that can be executed to reproduce the original database object definitions and table data. It dumps one or more MySQL databases for backup or transfer to another SQL server. The mysqldump command can also generate output in CSV, other delimited text, or XML format.

- Performance and Scalability Considerations
- Invocation Syntax
- Option Syntax - Alphabetical Summary
- Connection Options
- Option-File Options
- DDL Options
- Debug Options
- Help Options
- Internationalization Options
- Replication Options
- Format Options
- Filtering Options
- Performance Options
- Transactional Options
- Option Groups
- Examples
- Restrictions

mysqldump requires at least the SELECT privilege for dumped tables, SHOW VIEW for dumped views, TRIGGER for dumped triggers, and LOCK TABLES if the --single-transaction option is not used. Certain options might require other privileges as noted in the option descriptions.

To reload a dump file, you must have the privileges required to execute the statements that it contains, such as the appropriate CREATE privileges for objects created by those statements.

mysqldump output can include ALTER DATABASE statements that change the database collation. These may be used when dumping stored programs to preserve their character encodings. To reload a dump file containing such statements, the ALTER privilege for the affected database is required.

Note

A dump made using PowerShell on Windows with output redirection creates a file that has UTF-16 encoding:

```
shell> mysqldump [options] > dump.sql
```

> However, UTF-16 is not permitted as a connection character set (see Impermissible Client Character Sets), so the dump file will not load correctly. To work around this issue, use the `--result-file` option, which creates the output in ASCII format:
>
> ```
> shell> mysqldump [options] --result-file=dump.sql
> ```

Performance and Scalability Considerations

`mysqldump` advantages include the convenience and flexibility of viewing or even editing the output before restoring. You can clone databases for development and DBA work, or produce slight variations of an existing database for testing. It is not intended as a fast or scalable solution for backing up substantial amounts of data. With large data sizes, even if the backup step takes a reasonable time, restoring the data can be very slow because replaying the SQL statements involves disk I/O for insertion, index creation, and so on.

For large-scale backup and restore, a physical backup is more appropriate, to copy the data files in their original format that can be restored quickly:

- If your tables are primarily `InnoDB` tables, or if you have a mix of `InnoDB` and `MyISAM` tables, consider using the `mysqlbackup` command of the MySQL Enterprise Backup product. (Available as part of the Enterprise subscription.) It provides the best performance for `InnoDB` backups with minimal disruption; it can also back up tables from `MyISAM` and other storage engines; and it provides a number of convenient options to accommodate different backup scenarios. See Section 30.2, “MySQL Enterprise Backup Overview”.

`mysqldump` can retrieve and dump table contents row by row, or it can retrieve the entire content from a table and buffer it in memory before dumping it. Buffering in memory can be a problem if you are dumping large tables. To dump tables row by row, use the `--quick` option (or `--opt`, which enables `--quick`). The `--opt` option (and hence `--quick`) is enabled by default, so to enable memory buffering, use `--skip-quick`.

If you are using a recent version of `mysqldump` to generate a dump to be reloaded into a very old MySQL server, use the `--skip-opt` option instead of the `--opt` or `--extended-insert` option.

For additional information about `mysqldump`, see Section 7.4, “Using mysqldump for Backups”.

Invocation Syntax

There are in general three ways to use `mysqldump`—in order to dump a set of one or more tables, a set of one or more complete databases, or an entire MySQL server—as shown here:

```
shell> mysqldump [options] db_name [tbl_name ...]
shell> mysqldump [options] --databases db_name ...
shell> mysqldump [options] --all-databases
```

To dump entire databases, do not name any tables following *db_name*, or use the `--databases` or `--all-databases` option.

To see a list of the options your version of `mysqldump` supports, issue the command `mysqldump --help`.

Option Syntax - Alphabetical Summary

`mysqldump` supports the following options, which can be specified on the command line or in the `[mysqldump]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.14 mysqldump Options

Option Name	Description	Introduced	Deprecated	Removed
--add-drop-database	Add DROP DATABASE statement before each CREATE DATABASE statement			
--add-drop-table	Add DROP TABLE statement before each CREATE TABLE statement			
--add-drop-trigger	Add DROP TRIGGER statement before each CREATE TRIGGER statement			
--add-locks	Surround each table dump with LOCK TABLES and UNLOCK TABLES statements			
--all-databases	Dump all tables in all databases			
--allow-keywords	Allow creation of column names that are keywords			
--apply-slave-statements	Include STOP SLAVE prior to CHANGE MASTER statement and START SLAVE at end of output			
--bind-address	Use specified network interface to connect to MySQL Server			
--character-sets-dir	Directory where character sets are installed			
--column-statistics	Write ANALYZE TABLE statements to generate statistics histograms	8.0.2		
--comments	Add comments to dump file			
--compact	Produce more compact output			
--compatible	Produce output that is more compatible with other database systems or with older MySQL servers			
--complete-insert	Use complete INSERT statements that include column names			
--compress	Compress all information sent between client and server		8.0.18	
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18		
--create-options	Include all MySQL-specific table options in CREATE TABLE statements			
--databases	Interpret all name arguments as database names			
--debug	Write debugging log			
--debug-check	Print debugging information when program exits			
--debug-info	Print debugging information, memory, and CPU statistics when program exits			
--default-auth	Authentication plugin to use			
--default-character-set	Specify default character set			

Option Name	Description	Introduced	Deprecated	Removed
--defaults-extra-file	Read named option file in addition to usual option files			
--defaults-file	Read only named option file			
--defaults-group-suffix	Option group suffix value			
--delete-master-logs	On a master replication server, delete the binary logs after performing the dump operation			
--disable-keys	For each table, surround INSERT statements with statements to disable and enable keys			
--dump-date	Include dump date as "Dump completed on" comment if --comments is given			
--dump-slave	Include CHANGE MASTER statement that lists binary log coordinates of slave's master			
--enable-cleartext-plugin	Enable cleartext authentication plugin			
--events	Dump events from dumped databases			
--extended-insert	Use multiple-row INSERT syntax			
--fields-enclosed-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA			
--fields-escaped-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA			
--fields-optionally-enclosed-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA			
--fields-terminated-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA			
--flush-logs	Flush MySQL server log files before starting dump			
--flush-privileges	Emit a FLUSH PRIVILEGES statement after dumping mysql database			
--force	Continue even if an SQL error occurs during a table dump			
--get-server-public-key	Request RSA public key from server	8.0.3		
--help	Display help message and exit			
--hex-blob	Dump binary columns using hexadecimal notation			
--host	Host on which MySQL server is located			
--ignore-error	Ignore specified errors			
--ignore-table	Do not dump given table			

Option Name	Description	Introduced	Deprecated	Removed
--include-master-host-port	Include MASTER_HOST/MASTER_PORT options in CHANGE MASTER statement produced with --dump-slave			
--insert-ignore	Write INSERT IGNORE rather than INSERT statements			
--lines-terminated-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA			
--lock-all-tables	Lock all tables across all databases			
--lock-tables	Lock all tables before dumping them			
--log-error	Append warnings and errors to named file			
--login-path	Read login path options from .mylogin.cnf			
--master-data	Write the binary log file name and position to the output			
--max-allowed-packet	Maximum packet length to send to or receive from server			
--net-buffer-length	Buffer size for TCP/IP and socket communication			
--network-timeout	Increase network timeouts to permit larger table dumps	8.0.1		
--no-autocommit	Enclose the INSERT statements for each dumped table within SET autocommit = 0 and COMMIT statements			
--no-create-db	Do not write CREATE DATABASE statements			
--no-create-info	Do not write CREATE TABLE statements that re-create each dumped table			
--no-data	Do not dump table contents			
--no-defaults	Read no option files			
--no-set-names	Same as --skip-set-charset			
--no-tablespaces	Do not write any CREATE LOGFILE GROUP or CREATE TABLESPACE statements in output			
--opt	Shorthand for --add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset.			
--order-by-primary	Dump each table's rows sorted by its primary key, or by its first unique index			
--password	Password to use when connecting to server			
--pipe	Connect to server using named pipe (Windows only)			
--plugin-dir	Directory where plugins are installed			
--port	TCP/IP port number for connection			
--print-defaults	Print default options			

Option Name	Description	Introduced	Deprecated	Removed
--protocol	Connection protocol to use			
--quick	Retrieve rows for a table from the server a row at a time			
--quote-names	Quote identifiers within backtick characters			
--replace	Write REPLACE statements rather than INSERT statements			
--result-file	Direct output to a given file			
--routines	Dump stored routines (procedures and functions) from dumped databases			
--secure-auth	Do not send passwords to server in old (pre-4.1) format			8.0.3
--server-public-key-path	Path name to file containing RSA public key	8.0.4		
--set-charset	Add SET NAMES default_character_set to output			
--set-gtid-purged	Whether to add SET @@GLOBAL.GTID_PURGED to output			
--shared-memory-base-name	Name of shared memory to use for shared-memory connections			
--show-create-skip-secondary-engine	Exclude SECONDARY ENGINE clause from CREATE TABLE statements	8.0.18		
--single-transaction	Issue a BEGIN SQL statement before dumping data from server			
--skip-add-drop-table	Do not add a DROP TABLE statement before each CREATE TABLE statement			
--skip-add-locks	Do not add locks			
--skip-comments	Do not add comments to dump file			
--skip-compact	Do not produce more compact output			
--skip-disable-keys	Do not disable keys			
--skip-extended-insert	Turn off extended-insert			
--skip-opt	Turn off options set by --opt			
--skip-quick	Do not retrieve rows for a table from the server a row at a time			
--skip-quote-names	Do not quote identifiers			
--skip-set-charset	Do not write SET NAMES statement			
--skip-triggers	Do not dump triggers			
--skip-tz-utc	Turn off tz-utc			
--socket	Unix socket file or Windows named pipe to use			
--ssl-ca	File that contains list of trusted SSL Certificate Authorities			
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files			

Option Name	Description	Introduced	Deprecated	Removed
--ssl-cert	File that contains X.509 certificate			
--ssl-cipher	Permissible ciphers for connection encryption			
--ssl-crl	File that contains certificate revocation lists			
--ssl-crlpath	Directory that contains certificate revocation-list files			
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11		
--ssl-key	File that contains X.509 key			
--ssl-mode	Desired security state of connection to server			
--tab	Produce tab-separated data files			
--tables	Override --databases or -B option			
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16		
--tls-version	Permissible TLS protocols for encrypted connections			
--triggers	Dump triggers for each dumped table			
--tz-utc	Add SET TIME_ZONE='+00:00' to dump file			
--user	MySQL user name to use when connecting to server			
--verbose	Verbose mode			
--version	Display version information and exit			
--where	Dump only rows selected by given WHERE condition			
--xml	Produce XML output			
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18		

Connection Options

The mysqldump command logs into a MySQL server to extract information. The following options specify how to connect to the MySQL server, either on the same machine or a remote system.

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

  As of MySQL 8.0.18, this option is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `--compression-algorithms=value`

The permitted compression algorithms for connections to the server. The available algorithms are the same as for the protocol_compression_algorithms system variable. The default value is uncompressed.

For more information, see Section 4.2.6, “Connection Compression Control”.

This option was added in MySQL 8.0.18.

- --default-auth=*plugin*

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- --enable-cleartext-plugin

  Enable the mysql_clear_password cleartext authentication plugin. (See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.)

- --get-server-public-key

  Request from the server the public key required for RSA key pair-based password exchange. This option applies to clients that authenticate with the caching_sha2_password authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If --server-public-key-path=*file_name* is given and specifies a valid public key file, it takes precedence over --get-server-public-key.

  For information about the caching_sha2_password plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- --host=*host_name*, -h *host_name*

  Dump data from the MySQL server on the given host. The default host is localhost.

- --login-path=*name*

  Read options from the named login path in the .mylogin.cnf login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the mysql_config_editor utility. See Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --password[=*password*], -p[*password*]

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, mysqldump prompts for one. If given, there must be *no space* between --password= or -p and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

To explicitly specify that there is no password and that mysqldump should not prompt for one, use the --skip-password option.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysqldump does not find it. See Section 6.2.17, “Pluggable Authentication”.

- --port=*port_num*, -P *port_num*

  For TCP/IP connections, the port number to use.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

- --secure-auth

  This option was removed in MySQL 8.0.3.

- --server-public-key-path=*file_name*

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the sha256_password or caching_sha2_password authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If --server-public-key-path=*file_name* is given and specifies a valid public key file, it takes precedence over --get-server-public-key.

  For sha256_password, this option applies only if MySQL was built using OpenSSL.

  For information about the sha256_password and caching_sha2_password plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- --socket=*path*, -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--ssl-fips-mode={OFF|ON|STRICT}`

  Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These `--ssl-fips-mode` values are permitted:

  - `OFF`: Disable FIPS mode.
  - `ON`: Enable FIPS mode.
  - `STRICT`: Enable “strict” FIPS mode.

  Note

  If the OpenSSL FIPS Object Module is not available, the only permitted value for `--ssl-fips-mode` is `OFF`. In this case, setting `--ssl-fips-mode` to `ON` or `STRICT` causes the client to produce a warning at startup and to operate in non-FIPS mode.

- `--tls-ciphersuites=ciphersuite_list`

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- `--tls-version=protocol_list`

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `--user=user_name`, `-u user_name`

  The user name of the MySQL account to use for connecting to the server.

- `--zstd-compression-level=level`

  The compression level to use for connections to the server that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. The compression level setting has no effect on connections that do not use `zstd` compression.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

Option-File Options

These options are used to control which option files to read.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqldump` normally reads the `[client]` and `[mysqldump]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqldump` also reads the `[client_other]` and `[mysqldump_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

DDL Options

Usage scenarios for `mysqldump` include setting up an entire new MySQL instance (including database tables), and replacing data inside an existing instance with existing databases and tables. The following

options let you specify which things to tear down and set up when restoring a dump, by encoding various DDL statements within the dump file.

- `--add-drop-database`

  Write a `DROP DATABASE` statement before each `CREATE DATABASE` statement. This option is typically used in conjunction with the `--all-databases` or `--databases` option because no `CREATE DATABASE` statements are written unless one of those options is specified.

- `--add-drop-table`

  Write a `DROP TABLE` statement before each `CREATE TABLE` statement.

- `--add-drop-trigger`

  Write a `DROP TRIGGER` statement before each `CREATE TRIGGER` statement.

- `--all-tablespaces`, `-Y`

  Adds to a table dump all SQL statements needed to create any tablespaces used by an `NDB` table. This information is not otherwise included in the output from `mysqldump`. This option is currently relevant only to NDB Cluster tables.

- `--no-create-db`, `-n`

  Suppress the `CREATE DATABASE` statements that are otherwise included in the output if the `--databases` or `--all-databases` option is given.

- `--no-create-info`, `-t`

  Do not write `CREATE TABLE` statements that create each dumped table.

  Note

  This option does *not* exclude statements creating log file groups or tablespaces from `mysqldump` output; however, you can use the `--no-tablespaces` option for this purpose.

- `--no-tablespaces`, `-y`

  This option suppresses all `CREATE LOGFILE GROUP` and `CREATE TABLESPACE` statements in the output of `mysqldump`.

- `--replace`

  Write `REPLACE` statements rather than `INSERT` statements.

Debug Options

The following options print debugging information, encode debugging information in the dump file, or let the dump operation proceed regardless of potential problems.

- `--allow-keywords`

  Permit creation of column names that are keywords. This works by prefixing each column name with the table name.

- `--comments`, `-i`

Write additional information in the dump file such as program version, server version, and host. This option is enabled by default. To suppress this additional information, use `--skip-comments`.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default value is `d:t:o,/tmp/mysqldump.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--dump-date`

  If the `--comments` option is given, `mysqldump` produces a comment at the end of the dump of the following form:

  ```
  -- Dump completed on DATE
  ```

  However, the date causes dump files taken at different times to appear to be different, even if the data are otherwise identical. `--dump-date` and `--skip-dump-date` control whether the date is added to the comment. The default is `--dump-date` (include the date in the comment). `--skip-dump-date` suppresses date printing.

- `--force`, `-f`

  Ignore all errors; continue even if an SQL error occurs during a table dump.

  One use for this option is to cause `mysqldump` to continue executing even when it encounters a view that has become invalid because the definition refers to a table that has been dropped. Without `--force`, `mysqldump` exits with an error message. With `--force`, `mysqldump` prints the error message, but it also writes an SQL comment containing the view definition to the dump output and continues executing.

  If the `--ignore-error` option is also given to ignore specific errors, `--force` takes precedence.

- `--log-error=file_name`

  Log warnings and errors by appending them to the named file. The default is to do no logging.

- `--skip-comments`

  See the description for the `--comments` option.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

Help Options

The following options display information about the `mysqldump` command itself.

- `--help`, `-?`

Display a help message and exit.

- --version, -V

  Display version information and exit.

Internationalization Options

The following options change how the mysqldump command represents character data with national language settings.

- --character-sets-dir=*dir_name*

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- --default-character-set=*charset_name*

  Use *charset_name* as the default character set. See Section 10.15, “Character Set Configuration”. If no character set is specified, mysqldump uses utf8.

- --no-set-names, -N

  Turns off the --set-charset setting, the same as specifying --skip-set-charset.

- --set-charset

  Write SET NAMES *default_character_set* to the output. This option is enabled by default. To suppress the SET NAMES statement, use --skip-set-charset.

Replication Options

The mysqldump command is frequently used to create an empty instance, or an instance including data, on a slave server in a replication configuration. The following options apply to dumping and restoring data on replication master and slave servers.

- --apply-slave-statements

  For a slave dump produced with the --dump-slave option, add a STOP SLAVE statement before the CHANGE MASTER TO statement and a START SLAVE statement at the end of the output.

- --delete-master-logs

  On a master replication server, delete the binary logs by sending a PURGE BINARY LOGS statement to the server after performing the dump operation. This option automatically enables --master-data.

- --dump-slave[=*value*]

  This option is similar to --master-data except that it is used to dump a replication slave server to produce a dump file that can be used to set up another server as a slave that has the same master as the dumped server. It causes the dump output to include a CHANGE MASTER TO statement that indicates the binary log coordinates (file name and position) of the dumped slave's master. The CHANGE MASTER TO statement reads the values of Relay_Master_Log_File and Exec_Master_Log_Pos from the SHOW SLAVE STATUS output and uses them for MASTER_LOG_FILE and MASTER_LOG_POS respectively. These are the master server coordinates from which the slave should start replicating.

Note

Inconsistencies in the sequence of transactions from the relay log which have been executed can cause the wrong position to be used. See Section 17.5.1.33, "Replication and Transaction Inconsistencies" for more information.

`--dump-slave` causes the coordinates from the master to be used rather than those of the dumped server, as is done by the `--master-data` option. In addition, specfiying this option causes the `--master-data` option to be overridden, if used, and effectively ignored.

Warning

This option should not be used if the server where the dump is going to be applied uses `gtid_mode=ON` and `MASTER_AUTOPOSITION=1`.

The option value is handled the same way as for `--master-data` (setting no value or 1 causes a `CHANGE MASTER TO` statement to be written to the dump, setting 2 causes the statement to be written but encased in SQL comments) and has the same effect as `--master-data` in terms of enabling or disabling other options and in how locking is handled.

This option causes `mysqldump` to stop the slave SQL thread before the dump and restart it again after.

In conjunction with `--dump-slave`, the `--apply-slave-statements` and `--include-master-host-port` options can also be used.

- `--include-master-host-port`

  For the `CHANGE MASTER TO` statement in a slave dump produced with the `--dump-slave` option, add `MASTER_HOST` and `MASTER_PORT` options for the host name and TCP/IP port number of the slave's master.

- `--master-data[=value]`

  Use this option to dump a master replication server to produce a dump file that can be used to set up another server as a slave of the master. It causes the dump output to include a `CHANGE MASTER TO` statement that indicates the binary log coordinates (file name and position) of the dumped server. These are the master server coordinates from which the slave should start replicating after you load the dump file into the slave.

  If the option value is 2, the `CHANGE MASTER TO` statement is written as an SQL comment, and thus is informative only; it has no effect when the dump file is reloaded. If the option value is 1, the statement is not written as a comment and takes effect when the dump file is reloaded. If no option value is specified, the default value is 1.

  This option requires the `RELOAD` privilege and the binary log must be enabled.

  The `--master-data` option automatically turns off `--lock-tables`. It also turns on `--lock-all-tables`, unless `--single-transaction` also is specified, in which case, a global read lock is acquired only for a short time at the beginning of the dump (see the description for `--single-transaction`). In all cases, any action on logs happens at the exact moment of the dump.

  It is also possible to set up a slave by dumping an existing slave of the master, using the `--dump-slave` option, which overrides `--master-data` and causes it to be ignored if both options are used.

- `--set-gtid-purged=value`

This option is for servers that use GTID-based replication (gtid_mode=ON). It controls the inclusion of a SET @@GLOBAL.gtid_purged statement in the dump output, which updates the value of gtid_purged on a server where the dump file is reloaded, to add the GTID set from the source server's gtid_executed system variable. gtid_purged holds the GTIDs of all transactions that have been applied on the server, but do not exist on any binary log file on the server. mysqldump therefore adds the GTIDs for the transactions that were executed on the source server, so that the target server records these transactions as applied, although it does not have them in its binary logs. --set-gtid-purged also controls the inclusion of a SET @@SESSION.sql_log_bin=0 statement, which disables binary logging while the dump file is being reloaded. This statement prevents new GTIDs from being generated and assigned to the transactions in the dump file as they are executed, so that the original GTIDs for the transactions are used.

If you do not set the --set-gtid-purged option, the default is that a SET @@GLOBAL.gtid_purged statement is included in the dump output if GTIDs are enabled on the server you are backing up, and the set of GTIDs in the global value of the gtid_executed system variable is not empty. A SET @@SESSION.sql_log_bin=0 statement is also included if GTIDs are enabled on the server.

In MySQL 5.6 and 5.7, you can replace the value of gtid_purged with a specified GTID set, provided that gtid_executed and gtid_purged are empty. From MySQL 8.0, you can either replace the value of gtid_purged with a specified GTID set, or you can add a plus sign (+) to the statement to append a specified GTID set to the GTID set that is already held by gtid_purged. mysqldump's SET @@GLOBAL.gtid_purged statement includes a plus sign (+) in a version comment that takes effect when the dump file is replayed on releases from MySQL 8.0, meaning that for these releases, the GTID set from the dump file is added to the existing gtid_purged value. For MySQL 5.6 and 5.7, the value of gtid_purged is replaced with the GTID set from the dump file, which can only happen when gtid_executed is the empty set (so when replication has not been started previously, or when replication was not previously using GTIDs). For the exact details of how the SET @@GLOBAL.gtid_purged statement operates, see the gtid_purged description for the release where the dump file is to be replayed.

It is important to note that the value that is included by mysqldump for the SET @@GLOBAL.gtid_purged statement includes the GTIDs of all transactions in the gtid_executed set on the server, even those that changed suppressed parts of the database, or other databases on the server that were not included in a partial dump. This can mean that after the gtid_purged value has been updated on the server where the dump file is replayed, GTIDs are present that do not relate to any data on the target server. If you do not replay any further dump files on the target server, the extraneous GTIDs do not cause any problems with the future operation of the server, but they make it harder to compare or reconcile GTID sets on different servers in the replication topology. If you do replay a further dump file on the target server that contains the same GTIDs (for example, another partial dump from the same origin server), any SET @@GLOBAL.gtid_purged statement in the second dump file fails. In this case, either remove the statement manually before replaying the dump file, or output the dump file without the statement.

Note

For MySQL 5.6 and 5.7, it is not recommended to load a dump file when GTIDs are enabled on the server (gtid_mode=ON), if your dump file includes system tables. mysqldump issues DML instructions for the system tables which use the non-transactional MyISAM storage engine, and this combination is not permitted when GTIDs are enabled.

If the SET @@GLOBAL.gtid_purged statement would not have the desired result on your target server, you can exclude the statement from the output, or (from MySQL 8.0.17) include it but comment it out so

that it is not actioned automatically. You can also include the statement but manually edit it in the dump file to achieve the desired result.

The possible values for the `--set-gtid-purged` option are as follows:

`AUTO`	The default value. If GTIDs are enabled on the server you are backing up and `gtid_executed` is not empty, `SET @@GLOBAL.gtid_purged` is added to the output, containing the GTID set from `gtid_executed`. If GTIDs are enabled, `SET @@SESSION.sql_log_bin=0` is added to the output. If GTIDs are not enabled on the server, the statements are not added to the output.
`OFF`	`SET @@GLOBAL.gtid_purged` is not added to the output, and `SET @@SESSION.sql_log_bin=0` is not added to the output. For a server where GTIDs are not in use, use this option or `AUTO`. Only use this option for a server where GTIDs are in use if you are sure that the required GTID set is already present in `gtid_purged` on the target server and should not be changed, or if you plan to identify and add any missing GTIDs manually.
`ON`	If GTIDs are enabled on the server you are backing up, `SET @@GLOBAL.gtid_purged` is added to the output (unless `gtid_executed` is empty), and `SET @@SESSION.sql_log_bin=0` is added to the output. An error occurs if you set this option but GTIDs are not enabled on the server. For a server where GTIDs are in use, use this option or `AUTO`, unless you are sure that the GTIDs in `gtid_executed` are not needed on the target server.
`COMMENTED`	Available from MySQL 8.0.17. If GTIDs are enabled on the server you are backing up, `SET @@GLOBAL.gtid_purged` is added to the output (unless `gtid_executed` is empty), but it is commented out. This means that the value of `gtid_executed` is available in the output, but no action is taken automatically when the dump file is reloaded. `SET @@SESSION.sql_log_bin=0` is added to the output, and it is not commented out. With `COMMENTED`, you can control the use of the `gtid_executed` set manually or through automation. For example, you might prefer to do this if you are migrating data to another server that already has different active databases.

Format Options

The following options specify how to represent the entire dump file or certain kinds of data in the dump file. They also control whether certain optional information is written to the dump file.

- `--compact`

  Produce more compact output. This option enables the `--skip-add-drop-table`, `--skip-add-locks`, `--skip-comments`, `--skip-disable-keys`, and `--skip-set-charset` options.

- `--compatible=`*name*

  Produce output that is more compatible with other database systems or with older MySQL servers. The only permitted value for this option is `ansi`, which has the same meaning as the corresponding option for setting the server SQL mode. See Section 5.1.11, “Server SQL Modes”.

- `--complete-insert`, `-c`

  Use complete `INSERT` statements that include column names.

- `--create-options`

  Include all MySQL-specific table options in the `CREATE TABLE` statements.

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`

  These options are used with the `--tab` option and have the same meaning as the corresponding `FIELDS` clauses for `LOAD DATA`. See Section 13.2.7, “LOAD DATA Syntax”.

- `--hex-blob`

  Dump binary columns using hexadecimal notation (for example, `'abc'` becomes `0x616263`). The affected data types are `BINARY`, `VARBINARY`, `BLOB` types, `BIT`, all spatial data types, and other non-binary data types when used with the `binary` character set.

- `--lines-terminated-by=...`

  This option is used with the `--tab` option and has the same meaning as the corresponding `LINES` clause for `LOAD DATA`. See Section 13.2.7, “LOAD DATA Syntax”.

- `--quote-names`, `-Q`

  Quote identifiers (such as database, table, and column names) within `` ` `` characters. If the `ANSI_QUOTES` SQL mode is enabled, identifiers are quoted within `"` characters. This option is enabled by default. It can be disabled with `--skip-quote-names`, but this option should be given after any option such as `--compatible` that may enable `--quote-names`.

- `--result-file=`*`file_name`*, `-r` *`file_name`*

  Direct output to the named file. The result file is created and its previous contents overwritten, even if an error occurs while generating the dump.

  This option should be used on Windows to prevent newline `\n` characters from being converted to `\r\n` carriage return/newline sequences.

- `--show-create-skip-secondary-engine=`*`value`*

  Excludes the `SECONDARY ENGINE` clause from `CREATE TABLE` statements. It does so by enabling the `show_create_table_skip_secondary_engine` system variable for the duration of the dump operation. Alternatively, you can enable the `show_create_table_skip_secondary_engine` system variable prior to using `mysqldump`.

  This option was added in MySQL 8.0.18. Attempting a `mysqldump` operation with the `--show-create-skip-secondary-engine` option on a release prior to MySQL 8.0.18 that does not support the `show_create_table_skip_secondary_engine` variable causes an error.

- `--tab=`*`dir_name`*, `-T` *`dir_name`*

  Produce tab-separated text-format data files. For each dumped table, `mysqldump` creates a *`tbl_name`*`.sql` file that contains the `CREATE TABLE` statement that creates the table, and the server writes a *`tbl_name`*`.txt` file that contains its data. The option value is the directory in which to write the files.

Note

This option should be used only when `mysqldump` is run on the same machine as the `mysqld` server. Because the server creates `*.txt` files in the directory that you specify, the directory must be writable by the server and the MySQL account that you use must have the `FILE` privilege. Because `mysqldump` creates `*.sql` in the same directory, it must be writable by your system login account.

By default, the `.txt` data files are formatted using tab characters between column values and a newline at the end of each line. The format can be specified explicitly using the `--fields-xxx` and `--lines-terminated-by` options.

Column values are converted to the character set specified by the `--default-character-set` option.

- `--tz-utc`

  This option enables `TIMESTAMP` columns to be dumped and reloaded between servers in different time zones. `mysqldump` sets its connection time zone to UTC and adds `SET TIME_ZONE='+00:00'` to the dump file. Without this option, `TIMESTAMP` columns are dumped and reloaded in the time zones local to the source and destination servers, which can cause the values to change if the servers are in different time zones. `--tz-utc` also protects against changes due to daylight saving time. `--tz-utc` is enabled by default. To disable it, use `--skip-tz-utc`.

- `--xml`, `-X`

  Write dump output as well-formed XML.

  `NULL`, `'NULL'`, and Empty Values: For a column named *`column_name`*, the `NULL` value, an empty string, and the string value `'NULL'` are distinguished from one another in the output generated by this option as follows.

Value:	XML Representation:
`NULL` (*unknown value*)	`<field name="column_name" xsi:nil="true" />`
`''` (*empty string*)	`<field name="column_name"></field>`
`'NULL'` (*string value*)	`<field name="column_name">NULL</field>`

The output from the `mysql` client when run using the `--xml` option also follows the preceding rules. (See Section 4.5.1.1, “mysql Client Options”.)

XML output from `mysqldump` includes the XML namespace, as shown here:

```
shell> mysqldump --xml -u root world City
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="world">
<table_structure name="City">
<field Field="ID" Type="int(11)" Null="NO" Key="PRI" Extra="auto_increment" />
<field Field="Name" Type="char(35)" Null="NO" Key="" Default="" Extra="" />
<field Field="CountryCode" Type="char(3)" Null="NO" Key="" Default="" Extra="" />
<field Field="District" Type="char(20)" Null="NO" Key="" Default="" Extra="" />
<field Field="Population" Type="int(11)" Null="NO" Key="" Default="0" Extra="" />
<key Table="City" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="ID"
```

```
Collation="A" Cardinality="4079" Null="" Index_type="BTREE" Comment="" />
<options Name="City" Engine="MyISAM" Version="10" Row_format="Fixed" Rows="4079"
Avg_row_length="67" Data_length="273293" Max_data_length="18858823439613951"
Index_length="43008" Data_free="0" Auto_increment="4080"
Create_time="2007-03-31 01:47:01" Update_time="2007-03-31 01:47:02"
Collation="latin1_swedish_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="City">
<row>
<field name="ID">1</field>
<field name="Name">Kabul</field>
<field name="CountryCode">AFG</field>
<field name="District">Kabol</field>
<field name="Population">1780000</field>
</row>

...

<row>
<field name="ID">4079</field>
<field name="Name">Rafah</field>
<field name="CountryCode">PSE</field>
<field name="District">Rafah</field>
<field name="Population">92020</field>
</row>
</table_data>
</database>
</mysqldump>
```

Filtering Options

The following options control which kinds of schema objects are written to the dump file: by category, such as triggers or events; by name, for example, choosing which databases and tables to dump; or even filtering rows from the table data using a `WHERE` clause.

- `--all-databases`, `-A`

  Dump all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line.

  Prior to MySQL 8.0, the `--routines` and `--events` options for `mysqldump` and `mysqlpump` were not required to include stored routines and events when using the `--all-databases` option: The dump included the `mysql` system database, and therefore also the `mysql.proc` and `mysql.event` tables containing stored routine and event definitions. As of MySQL 8.0, the `mysql.event` and `mysql.proc` tables are not used. Definitions for the corresponding objects are stored in data dictionary tables, but those tables are not dumped. To include stored routines and events in a dump made using `--all-databases`, use the `--routines` and `--events` options explicitly.

- `--databases`, `-B`

  Dump several databases. Normally, `mysqldump` treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names. `CREATE DATABASE` and `USE` statements are included in the output before each new database.

  This option may be used to dump the `performance_schema` database, which normally is not dumped even with the `--all-databases` option. (Also use the `--skip-lock-tables` option.)

- `--events`, `-E`

Include Event Scheduler events for the dumped databases in the output. This option requires the EVENT privileges for those databases.

The output generated by using --events contains CREATE EVENT statements to create the events.

- --ignore-error=*error[,error]...*

  Ignore the specified errors. The option value is a list of comma-separated error numbers specifying the errors to ignore during mysqldump execution. If the --force option is also given to ignore all errors, --force takes precedence.

- --ignore-table=*db_name.tbl_name*

  Do not dump the given table, which must be specified using both the database and table names. To ignore multiple tables, use this option multiple times. This option also can be used to ignore views.

- --no-data, -d

  Do not write any table row information (that is, do not dump table contents). This is useful if you want to dump only the CREATE TABLE statement for the table (for example, to create an empty copy of the table by loading the dump file).

- --routines, -R

  Include stored routines (procedures and functions) for the dumped databases in the output. This option requires the global SELECT privilege.

  The output generated by using --routines contains CREATE PROCEDURE and CREATE FUNCTION statements to create the routines.

- --tables

  Override the --databases or -B option. mysqldump regards all name arguments following the option as table names.

- --triggers

  Include triggers for each dumped table in the output. This option is enabled by default; disable it with --skip-triggers.

  To be able to dump a table's triggers, you must have the TRIGGER privilege for the table.

  Multiple triggers are permitted. mysqldump dumps triggers in activation order so that when the dump file is reloaded, triggers are created in the same activation order. However, if a mysqldump dump file contains multiple triggers for a table that have the same trigger event and action time, an error occurs for attempts to load the dump file into an older server that does not support multiple triggers. (For a workaround, see Downgrade Notes; you can convert triggers to be compatible with older servers.)

- --where='*where_condition*', -w '*where_condition*'

  Dump only rows selected by the given WHERE condition. Quotes around the condition are mandatory if it contains spaces or other characters that are special to your command interpreter.

  Examples:

```
--where="user='jimf'"
```

```
-w"userid>1"
-w"userid<1"
```

Performance Options

The following options are the most relevant for the performance particularly of the restore operations. For large data sets, restore operation (processing the INSERT statements in the dump file) is the most time-consuming part. When it is urgent to restore data quickly, plan and test the performance of this stage in advance. For restore times measured in hours, you might prefer an alternative backup and restore solution, such as MySQL Enterprise Backup for InnoDB-only and mixed-use databases.

Performance is also affected by the transactional options, primarily for the dump operation.

- --column-statistics

  Add ANALYZE TABLE statements to the output to generate histogram statistics for dumped tables when the dump file is reloaded. This option is disabled by default because histogram generation for large tables can take a long time.

- --disable-keys, -K

  For each table, surround the INSERT statements with /*!40000 ALTER TABLE *tbl_name* DISABLE KEYS */; and /*!40000 ALTER TABLE *tbl_name* ENABLE KEYS */; statements. This makes loading the dump file faster because the indexes are created after all rows are inserted. This option is effective only for nonunique indexes of MyISAM tables.

- --extended-insert, -e

  Write INSERT statements using multiple-row syntax that includes several VALUES lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.

- --insert-ignore

  Write INSERT IGNORE statements rather than INSERT statements.

- --max-allowed-packet=*value*

  The maximum size of the buffer for client/server communication. The default is 24MB, the maximum is 1GB.

- --net-buffer-length=*value*

  The initial size of the buffer for client/server communication. When creating multiple-row INSERT statements (as with the --extended-insert or --opt option), mysqldump creates rows up to --net-buffer-length bytes long. If you increase this variable, ensure that the MySQL server net_buffer_length system variable has a value at least this large.

- --network-timeout, -M

  Enable large tables to be dumped by setting --max-allowed-packet to its maximum value and network read and write timeouts to a large value. This option is enabled by default. To disable it, use --skip-network-timeout.

- --opt

  This option, enabled by default, is shorthand for the combination of --add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-

charset. It gives a fast dump operation and produces a dump file that can be reloaded into a MySQL server quickly.

Because the --opt option is enabled by default, you only specify its converse, the --skip-opt to turn off several default settings. See the discussion of mysqldump option groups for information about selectively enabling or disabling a subset of the options affected by --opt.

- --quick, -q

  This option is useful for dumping large tables. It forces mysqldump to retrieve rows for a table from the server a row at a time rather than retrieving the entire row set and buffering it in memory before writing it out.

- --skip-opt

  See the description for the --opt option.

Transactional Options

The following options trade off the performance of the dump operation, against the reliability and consistency of the exported data.

- --add-locks

  Surround each table dump with LOCK TABLES and UNLOCK TABLES statements. This results in faster inserts when the dump file is reloaded. See Section 8.2.5.1, “Optimizing INSERT Statements”.

- --flush-logs, -F

  Flush the MySQL server log files before starting the dump. This option requires the RELOAD privilege. If you use this option in combination with the --all-databases option, the logs are flushed *for each database dumped*. The exception is when using --lock-all-tables, --master-data, or --single-transaction: In this case, the logs are flushed only once, corresponding to the moment that all tables are locked by FLUSH TABLES WITH READ LOCK. If you want your dump and the log flush to happen at exactly the same moment, you should use --flush-logs together with --lock-all-tables, --master-data, or --single-transaction.

- --flush-privileges

  Add a FLUSH PRIVILEGES statement to the dump output after dumping the mysql database. This option should be used any time the dump contains the mysql database and any other database that depends on the data in the mysql database for proper restoration.

  Note

  For upgrades to MySQL 5.7.2 or higher from older versions, do not use --flush-privileges. For upgrade instructions in this case, see Section 2.11.4, “Changes in MySQL 8.0”.

- --lock-all-tables, -x

  Lock all tables across all databases. This is achieved by acquiring a global read lock for the duration of the whole dump. This option automatically turns off --single-transaction and --lock-tables.

- --lock-tables, -l

  For each dumped database, lock all tables to be dumped before dumping them. The tables are locked with READ LOCAL to permit concurrent inserts in the case of MyISAM tables. For transactional tables

such as InnoDB, --single-transaction is a much better option than --lock-tables because it does not need to lock the tables at all.

Because --lock-tables locks tables for each database separately, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states.

Some options, such as --opt, automatically enable --lock-tables. If you want to override this, use --skip-lock-tables at the end of the option list.

- --no-autocommit

  Enclose the INSERT statements for each dumped table within SET autocommit = 0 and COMMIT statements.

- --order-by-primary

  Dump each table's rows sorted by its primary key, or by its first unique index, if such an index exists. This is useful when dumping a MyISAM table to be loaded into an InnoDB table, but makes the dump operation take considerably longer.

- --shared-memory-base-name=*name*

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is MYSQL. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the shared_memory system variable enabled to support shared-memory connections.

- --single-transaction

  This option sets the transaction isolation mode to REPEATABLE READ and sends a START TRANSACTION SQL statement to the server before dumping data. It is useful only with transactional tables such as InnoDB, because then it dumps the consistent state of the database at the time when START TRANSACTION was issued without blocking any applications.

  When using this option, you should keep in mind that only InnoDB tables are dumped in a consistent state. For example, any MyISAM or MEMORY tables dumped while using this option may still change state.

  While a --single-transaction dump is in process, to ensure a valid dump file (correct table contents and binary log coordinates), no other connection should use the following statements: ALTER TABLE, CREATE TABLE, DROP TABLE, RENAME TABLE, TRUNCATE TABLE. A consistent read is not isolated from those statements, so use of them on a table to be dumped can cause the SELECT that is performed by mysqldump to retrieve the table contents to obtain incorrect contents or fail.

  The --single-transaction option and the --lock-tables option are mutually exclusive because LOCK TABLES causes any pending transactions to be committed implicitly.

  To dump large tables, combine the --single-transaction option with the --quick option.

Option Groups

- The --opt option turns on several settings that work together to perform a fast dump operation. All of these settings are on by default, because --opt is on by default. Thus you rarely if ever specify --opt. Instead, you can turn these settings off as a group by specifying --skip-opt, the optionally re-enable certain settings by specifying the associated options later on the command line.

- The --compact option turns off several settings that control whether optional statements and comments appear in the output. Again, you can follow this option with other options that re-enable certain settings, or turn all the settings on by using the --skip-compact form.

When you selectively enable or disable the effect of a group option, order is important because options are processed first to last. For example, --disable-keys --lock-tables --skip-opt would not have the intended effect; it is the same as --skip-opt by itself.

Examples

To make a backup of an entire database:

```
shell> mysqldump db_name > backup-file.sql
```

To load the dump file back into the server:

```
shell> mysql db_name < backup-file.sql
```

Another way to reload the dump file:

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```

mysqldump is also very useful for populating databases by copying data from one MySQL server to another:

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

You can dump several databases with one command:

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.sql
```

To dump all databases, use the --all-databases option:

```
shell> mysqldump --all-databases > all_databases.sql
```

For InnoDB tables, mysqldump provides a way of making an online backup:

```
shell> mysqldump --all-databases --master-data --single-transaction > all_databases.sql
```

This backup acquires a global read lock on all tables (using FLUSH TABLES WITH READ LOCK) at the beginning of the dump. As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the FLUSH statement is issued, the MySQL server may get stalled until those statements finish. After that, the dump becomes lock free and does not disturb reads and writes on the tables. If the update statements that the MySQL server receives are short (in terms of execution time), the initial lock period should not be noticeable, even with many updates.

For point-in-time recovery (also known as “roll-forward,” when you need to restore an old backup and replay the changes that happened since that backup), it is often useful to rotate the binary log (see Section 5.4.4, “The Binary Log”) or at least know the binary log coordinates to which the dump corresponds:

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

Or:

```
shell> mysqldump --all-databases --flush-logs --master-data=2
            > all_databases.sql
```

The `--master-data` and `--single-transaction` options can be used simultaneously, which provides a convenient way to make an online backup suitable for use prior to point-in-time recovery if tables are stored using the `InnoDB` storage engine.

For more information on making backups, see Section 7.2, “Database Backup Methods”, and Section 7.3, “Example Backup and Recovery Strategy”.

- To select the effect of `--opt` except for some features, use the `--skip` option for each feature. To disable extended inserts and memory buffering, use `--opt --skip-extended-insert --skip-quick`. (Actually, `--skip-extended-insert --skip-quick` is sufficient because `--opt` is on by default.)
- To reverse `--opt` for all features except index disabling and table locking, use `--skip-opt --disable-keys --lock-tables`.

Restrictions

`mysqldump` does not dump the `performance_schema` or `sys` schema by default. To dump any of these, name them explicitly on the command line. You can also name them with the `--databases` option. For `performance_schema`, also use the `--skip-lock-tables` option.

`mysqldump` does not dump the `INFORMATION_SCHEMA` schema.

`mysqldump` does not dump `InnoDB` `CREATE TABLESPACE` statements.

`mysqldump` does not dump the NDB Cluster `ndbinfo` information database.

`mysqldump` includes statements to recreate the `general_log` and `slow_query_log` tables for dumps of the `mysql` database. Log table contents are not dumped.

If you encounter problems backing up views due to insufficient privileges, see Section 24.9, “Restrictions on Views” for a workaround.

4.5.5 `mysqlimport` — A Data Import Program

The `mysqlimport` client provides a command-line interface to the `LOAD DATA` SQL statement. Most options to `mysqlimport` correspond directly to clauses of `LOAD DATA` syntax. See Section 13.2.7, “LOAD DATA Syntax”.

Invoke `mysqlimport` like this:

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

For each text file named on the command line, `mysqlimport` strips any extension from the file name and uses the result to determine the name of the table into which to import the file's contents. For example, files named `patient.txt`, `patient.text`, and `patient` all would be imported into a table named `patient`.

`mysqlimport` supports the following options, which can be specified on the command line or in the `[mysqlimport]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.15 mysqlimport Options

Option Name	Description	Introduced	Deprecated	Removed
--bind-address	Use specified network interface to connect to MySQL Server			
--columns	This option takes a comma-separated list of column names as its value			
--compress	Compress all information sent between client and server		8.0.18	
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18		
--debug	Write debugging log			
--debug-check	Print debugging information when program exits			
--debug-info	Print debugging information, memory, and CPU statistics when program exits			
--default-auth	Authentication plugin to use			
--default-character-set	Specify default character set			
--defaults-extra-file	Read named option file in addition to usual option files			
--defaults-file	Read only named option file			
--defaults-group-suffix	Option group suffix value			
--delete	Empty the table before importing the text file			
--enable-cleartext-plugin	Enable cleartext authentication plugin			
--fields-enclosed-by	This option has the same meaning as the corresponding clause for LOAD DATA			
--fields-escaped-by	This option has the same meaning as the corresponding clause for LOAD DATA			
--fields-optionally-enclosed-by	This option has the same meaning as the corresponding clause for LOAD DATA			
--fields-terminated-by	This option has the same meaning as the corresponding clause for LOAD DATA			
--force	Continue even if an SQL error occurs			
--get-server-public-key	Request RSA public key from server	8.0.3		
--help	Display help message and exit			
--host	Host on which MySQL server is located			
--ignore	See the description for the --replace option			
--ignore-lines	Ignore the first N lines of the data file			
--lines-terminated-by	This option has the same meaning as the corresponding clause for LOAD DATA			
--local	Read input files locally from the client host			
--lock-tables	Lock all tables for writing before processing any text files			

Option Name	Description	Introduced	Deprecated	Removed
--login-path	Read login path options from .mylogin.cnf			
--low-priority	Use LOW_PRIORITY when loading the table.			
--no-defaults	Read no option files			
--password	Password to use when connecting to server			
--pipe	Connect to server using named pipe (Windows only)			
--plugin-dir	Directory where plugins are installed			
--port	TCP/IP port number for connection			
--print-defaults	Print default options			
--protocol	Connection protocol to use			
--replace	The --replace and --ignore options control handling of input rows that duplicate existing rows on unique key values			
--secure-auth	Do not send passwords to server in old (pre-4.1) format			8.0.3
--server-public-key-path	Path name to file containing RSA public key	8.0.4		
--shared-memory-base-name	Name of shared memory to use for shared-memory connections			
--silent	Produce output only when errors occur			
--socket	Unix socket file or Windows named pipe to use			
--ssl-ca	File that contains list of trusted SSL Certificate Authorities			
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files			
--ssl-cert	File that contains X.509 certificate			
--ssl-cipher	Permissible ciphers for connection encryption			
--ssl-crl	File that contains certificate revocation lists			
--ssl-crlpath	Directory that contains certificate revocation-list files			
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11		
--ssl-key	File that contains X.509 key			
--ssl-mode	Desired security state of connection to server			
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16		
--tls-version	Permissible TLS protocols for encrypted connections			
--use-threads	Number of threads for parallel file-loading			
--user	MySQL user name to use when connecting to server			
--verbose	Verbose mode			
--version	Display version information and exit			

Option Name	Description	Introduced	Deprecated	Removed
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18		

- `--help`, `-?`

  Display a help message and exit.

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=dir_name`

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--columns=column_list`, `-c column_list`

  This option takes a list of comma-separated column names as its value. The order of the column names indicates how to match data file columns with table columns.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

  As of MySQL 8.0.18, this option is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `--compression-algorithms=value`

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. The default value is `uncompressed`.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical `debug_options` string is `d:t:o,file_name`. The default is `d:t:o`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-character-set=charset_name`

  Use `charset_name` as the default character set. See Section 10.15, “Character Set Configuration”.

- `--default-auth=plugin`

A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqlimport` normally reads the `[client]` and `[mysqlimport]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlimport` also reads the `[client_other]` and `[mysqlimport_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--delete`, `-D`

  Empty the table before importing the text file.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.)

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`

  These options have the same meaning as the corresponding clauses for `LOAD DATA`. See Section 13.2.7, “LOAD DATA Syntax”.

- `--force`, `-f`

  Ignore errors. For example, if a table for a text file does not exist, continue processing any remaining files. Without `--force`, `mysqlimport` exits if a table does not exist.

- `--get-server-public-key`

  Request from the server the public key required for RSA key pair-based password exchange. This option applies to clients that authenticate with the `caching_sha2_password` authentication plugin. For that

plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

For information about the `caching_sha2_password` plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--host=host_name`, `-h host_name`

  Import data to the MySQL server on the given host. The default host is `localhost`.

- `--ignore`, `-i`

  See the description for the `--replace` option.

- `--ignore-lines=N`

  Ignore the first `N` lines of the data file.

- `--lines-terminated-by=...`

  This option has the same meaning as the corresponding clause for `LOAD DATA`. For example, to import Windows files that have lines terminated with carriage return/linefeed pairs, use `--lines-terminated-by="\r\n"`. (You might have to double the backslashes, depending on the escaping conventions of your command interpreter.) See Section 13.2.7, “LOAD DATA Syntax”.

- `--local`, `-L`

  By default, files are read by the server on the server host. With this option, `mysqlimport` reads input files locally on the client host. Enabling local data loading also requires that the server permits it; see Section 6.1.6, “Security Issues with LOAD DATA LOCAL”

- `--lock-tables`, `-l`

  Lock *all* tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--low-priority`

  Use `LOW_PRIORITY` when loading the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”.)

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --password[=*password*], -p[*password*]

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, mysqlimport prompts for one. If given, there must be *no space* between --password= or -p and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

  To explicitly specify that there is no password and that mysqlimport should not prompt for one, use the --skip-password option.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysqlimport does not find it. See Section 6.2.17, “Pluggable Authentication”.

- --port=*port_num*, -P *port_num*

  For TCP/IP connections, the port number to use.

- --print-defaults

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

- --replace, -r

  The --replace and --ignore options control handling of input rows that duplicate existing rows on unique key values. If you specify --replace, new rows replace existing rows that have the same unique key value. If you specify --ignore, input rows that duplicate an existing row on a unique key

value are skipped. If you do not specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

- `--secure-auth`

  This option was removed in MySQL 8.0.3.

- `--server-public-key-path=file_name`

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  For `sha256_password`, this option applies only if MySQL was built using OpenSSL.

  For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--silent`, `-s`

  Silent mode. Produce output only when errors occur.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--ssl-fips-mode={OFF|ON|STRICT}`

  Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These `--ssl-fips-mode` values are permitted:

- OFF: Disable FIPS mode.
- ON: Enable FIPS mode.
- STRICT: Enable “strict” FIPS mode.

Note

If the OpenSSL FIPS Object Module is not available, the only permitted value for --ssl-fips-mode is OFF. In this case, setting --ssl-fips-mode to ON or STRICT causes the client to produce a warning at startup and to operate in non-FIPS mode.

- --tls-ciphersuites=*ciphersuite_list*

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- --tls-version=*protocol_list*

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- --user=*user_name*, -u *user_name*

  The user name of the MySQL account to use for connecting to the server.

- --use-threads=*N*

  Load files in parallel using *N* threads.

- --verbose, -v

  Verbose mode. Print more information about what the program does.

- --version, -V

  Display version information and exit.

- --zstd-compression-level=*level*

  The compression level to use for connections to the server that use the zstd compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default zstd compression level is 3. The compression level setting has no effect on connections that do not use zstd compression.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

Here is a sample session that demonstrates use of mysqlimport:

```
shell> mysql -e 'CREATE TABLE imptest(id INT, n VARCHAR(30))' test
shell> ed
a
100     Max Sydow
101     Count Dracula
.
w imptest.txt
32
q
shell> od -c imptest.txt
0000000   1   0   0  \t   M   a   x       S   y   d   o   w  \n   1   0
0000020   1  \t   C   o   u   n   t       D   r   a   c   u   l   a  \n
0000040
shell> mysqlimport --local test imptest.txt
test.imptest: Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
shell> mysql -e 'SELECT * FROM imptest' test
+------+---------------+
| id   | n             |
+------+---------------+
|  100 | Max Sydow     |
|  101 | Count Dracula |
+------+---------------+
```

4.5.6 mysqlpump — A Database Backup Program

- mysqlpump Invocation Syntax
- mysqlpump Option Summary
- mysqlpump Option Descriptions
- mysqlpump Object Selection
- mysqlpump Parallel Processing
- mysqlpump Restrictions

The `mysqlpump` client utility performs logical backups, producing a set of SQL statements that can be executed to reproduce the original database object definitions and table data. It dumps one or more MySQL databases for backup or transfer to another SQL server.

`mysqlpump` features include:

- Parallel processing of databases, and of objects within databases, to speed up the dump process
- Better control over which databases and database objects (tables, stored programs, user accounts) to dump
- Dumping of user accounts as account-management statements (`CREATE USER`, `GRANT`) rather than as inserts into the `mysql` system database
- Capability of creating compressed output
- Progress indicator (the values are estimates)
- For dump file reloading, faster secondary index creation for `InnoDB` tables by adding indexes after rows are inserted

Note

`mysqlpump` uses MySQL features introduced in MySQL 5.7, and thus assumes use with MySQL 5.7 or higher.

mysqlpump requires at least the SELECT privilege for dumped tables, SHOW VIEW for dumped views, TRIGGER for dumped triggers, and LOCK TABLES if the --single-transaction option is not used. The SELECT privilege on the mysql system database is required to dump user definitions. Certain options might require other privileges as noted in the option descriptions.

To reload a dump file, you must have the privileges required to execute the statements that it contains, such as the appropriate CREATE privileges for objects created by those statements.

Note

A dump made using PowerShell on Windows with output redirection creates a file that has UTF-16 encoding:

```
shell> mysqlpump [options] > dump.sql
```

However, UTF-16 is not permitted as a connection character set (see Section 10.4, "Connection Character Sets and Collations"), so the dump file will not load correctly. To work around this issue, use the --result-file option, which creates the output in ASCII format:

```
shell> mysqlpump [options] --result-file=dump.sql
```

mysqlpump Invocation Syntax

By default, mysqlpump dumps all databases (with certain exceptions noted in mysqlpump Restrictions). To specify this behavior explicitly, use the --all-databases option:

```
shell> mysqlpump --all-databases
```

To dump a single database, or certain tables within that database, name the database on the command line, optionally followed by table names:

```
shell> mysqlpump db_name
shell> mysqlpump db_name tbl_name1 tbl_name2 ...
```

To treat all name arguments as database names, use the --databases option:

```
shell> mysqlpump --databases db_name1 db_name2 ...
```

By default, mysqlpump does not dump user account definitions, even if you dump the mysql system database that contains the grant tables. To dump grant table contents as logical definitions in the form of CREATE USER and GRANT statements, use the --users option and suppress all database dumping:

```
shell> mysqlpump --exclude-databases=% --users
```

In the preceding command, % is a wildcard that matches all database names for the --exclude-databases option.

mysqlpump supports several options for including or excluding databases, tables, stored programs, and user definitions. See mysqlpump Object Selection.

To reload a dump file, execute the statements that it contains. For example, use the mysql client:

```
shell> mysqlpump [options] > dump.sql
shell> mysql < dump.sql
```

The following discussion provides additional mysqlpump usage examples.

To see a list of the options mysqlpump supports, issue the command mysqlpump --help.

mysqlpump Option Summary

mysqlpump supports the following options, which can be specified on the command line or in the [mysqlpump] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.16 mysqlpump Options

Option Name	Description	Introduced	Deprecated	Removed
--add-drop-database	Add DROP DATABASE statement before each CREATE DATABASE statement			
--add-drop-table	Add DROP TABLE statement before each CREATE TABLE statement			
--add-drop-user	Add DROP USER statement before each CREATE USER statement			
--add-locks	Surround each table dump with LOCK TABLES and UNLOCK TABLES statements			
--all-databases	Dump all databases			
--bind-address	Use specified network interface to connect to MySQL Server			
--character-sets-dir	Directory where character sets are installed			
--column-statistics	Write ANALYZE TABLE statements to generate statistics histograms	8.0.2		
--complete-insert	Use complete INSERT statements that include column names			
--compress	Compress all information sent between client and server		8.0.18	
--compress-output	Output compression algorithm			
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18		
--databases	Interpret all name arguments as database names			
--debug	Write debugging log			
--debug-check	Print debugging information when program exits			
--debug-info	Print debugging information, memory, and CPU statistics when program exits			
--default-auth	Authentication plugin to use			
--default-character-set	Specify default character set			
--default-parallelism	Default number of threads for parallel processing			
--defaults-extra-file	Read named option file in addition to usual option files			

Option Name	Description	Introduced	Deprecated	Removed
--defaults-file	Read only named option file			
--defaults-group-suffix	Option group suffix value			
--defer-table-indexes	For reloading, defer index creation until after loading table rows			
--events	Dump events from dumped databases			
--exclude-databases	Databases to exclude from dump			
--exclude-events	Events to exclude from dump			
--exclude-routines	Routines to exclude from dump			
--exclude-tables	Tables to exclude from dump			
--exclude-triggers	Triggers to exclude from dump			
--exclude-users	Users to exclude from dump			
--extended-insert	Use multiple-row INSERT syntax			
--get-server-public-key	Request RSA public key from server	8.0.3		
--help	Display help message and exit			
--hex-blob	Dump binary columns using hexadecimal notation			
--host	Host on which MySQL server is located			
--include-databases	Databases to include in dump			
--include-events	Events to include in dump			
--include-routines	Routines to include in dump			
--include-tables	Tables to include in dump			
--include-triggers	Triggers to include in dump			
--include-users	Users to include in dump			
--insert-ignore	Write INSERT IGNORE rather than INSERT statements			
--log-error-file	Append warnings and errors to named file			
--login-path	Read login path options from .mylogin.cnf			
--max-allowed-packet	Maximum packet length to send to or receive from server			
--net-buffer-length	Buffer size for TCP/IP and socket communication			
--no-create-db	Do not write CREATE DATABASE statements			
--no-create-info	Do not write CREATE TABLE statements that re-create each dumped table			
--no-defaults	Read no option files			
--parallel-schemas	Specify schema-processing parallelism			
--password	Password to use when connecting to server			
--plugin-dir	Directory where plugins are installed			
--port	TCP/IP port number for connection			

Option Name	Description	Introduced	Deprecated	Removed
--print-defaults	Print default options			
--protocol	Connection protocol to use			
--replace	Write REPLACE statements rather than INSERT statements			
--result-file	Direct output to a given file			
--routines	Dump stored routines (procedures and functions) from dumped databases			
--secure-auth	Do not send passwords to server in old (pre-4.1) format			8.0.3
--server-public-key-path	Path name to file containing RSA public key	8.0.4		
--set-charset	Add SET NAMES default_character_set to output			
--set-gtid-purged	Whether to add SET @@GLOBAL.GTID_PURGED to output	8.0.1		
--single-transaction	Dump tables within single transaction			
--skip-definer	Omit DEFINER and SQL SECURITY clauses from view and stored program CREATE statements			
--skip-dump-rows	Do not dump table rows			
--socket	Unix socket file or Windows named pipe to use			
--ssl-ca	File that contains list of trusted SSL Certificate Authorities			
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files			
--ssl-cert	File that contains X.509 certificate			
--ssl-cipher	Permissible ciphers for connection encryption			
--ssl-crl	File that contains certificate revocation lists			
--ssl-crlpath	Directory that contains certificate revocation-list files			
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11		
--ssl-key	File that contains X.509 key			
--ssl-mode	Desired security state of connection to server			
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16		
--tls-version	Permissible TLS protocols for encrypted connections			
--triggers	Dump triggers for each dumped table			
--tz-utc	Add SET TIME_ZONE='+00:00' to dump file			
--user	MySQL user name to use when connecting to server			
--users	Dump user accounts			

Option Name	Description	Introduced	Deprecated	Removed
--version	Display version information and exit			
--watch-progress	Display progress indicator			
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18		

mysqlpump Option Descriptions

- `--help`, `-?`

  Display a help message and exit.

- `--add-drop-database`

  Write a `DROP DATABASE` statement before each `CREATE DATABASE` statement.

- `--add-drop-table`

  Write a `DROP TABLE` statement before each `CREATE TABLE` statement.

- `--add-drop-user`

  Write a `DROP USER` statement before each `CREATE USER` statement.

- `--add-locks`

  Surround each table dump with `LOCK TABLES` and `UNLOCK TABLES` statements. This results in faster inserts when the dump file is reloaded. See Section 8.2.5.1, “Optimizing INSERT Statements”.

  This option does not work with parallelism because `INSERT` statements from different tables can be interleaved and `UNLOCK TABLES` following the end of the inserts for one table could release locks on tables for which inserts remain.

  `--add-locks` and `--single-transaction` are mutually exclusive.

- `--all-databases`, `-A`

  Dump all databases (with certain exceptions noted in mysqlpump Restrictions). This is the default behavior if no other is specified explicitly.

  `--all-databases` and `--databases` are mutually exclusive.

  Prior to MySQL 8.0, the `--routines` and `--events` options for `mysqldump` and `mysqlpump` were not required to include stored routines and events when using the `--all-databases` option: The dump included the `mysql` system database, and therefore also the `mysql.proc` and `mysql.event` tables containing stored routine and event definitions. As of MySQL 8.0, the `mysql.event` and `mysql.proc` tables are not used. Definitions for the corresponding objects are stored in data dictionary tables, but those tables are not dumped. To include stored routines and events in a dump made using `--all-databases`, use the `--routines` and `--events` options explicitly.

- `--bind-address=`*`ip_address`*

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=`*`path`*

The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--column-statistics`

  Add `ANALYZE TABLE` statements to the output to generate histogram statistics for dumped tables when the dump file is reloaded. This option is disabled by default because histogram generation for large tables can take a long time.

- `--complete-insert`

  Write complete `INSERT` statements that include column names.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

  As of MySQL 8.0.18, this option is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `--compress-output=`*`algorithm`*

  By default, `mysqlpump` does not compress output. This option specifies output compression using the specified algorithm. Permitted algorithms are `LZ4` and `ZLIB`.

  To uncompress compressed output, you must have an appropriate utility. If the system commands `lz4` and `openssl zlib` are not available, MySQL distributions include `lz4_decompress` and `zlib_decompress` utilities that can be used to decompress `mysqlpump` output that was compressed using the `--compress-output=LZ4` and `--compress-output=ZLIB` options. For more information, see Section 4.8.1, “`lz4_decompress` — Decompress mysqlpump LZ4-Compressed Output”, and Section 4.8.3, “`zlib_decompress` — Decompress mysqlpump ZLIB-Compressed Output”.

- `--compression-algorithms=`*`value`*

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. The default value is `uncompressed`.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

- `--databases`, `-B`

  Normally, `mysqlpump` treats the first name argument on the command line as a database name and any following names as table names. With this option, it treats all name arguments as database names. `CREATE DATABASE` statements are included in the output before each new database.

  `--all-databases` and `--databases` are mutually exclusive.

- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:O,/tmp/mysqlpump.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`, `-T`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.15, “Character Set Configuration”. If no character set is specified, `mysqlpump` uses `utf8`.

- `--default-parallelism=N`

  The default number of threads for each parallel processing queue. The default is 2.

  The `--parallel-schemas` option also affects parallelism and can be used to override the default number of threads. For more information, see mysqlpump Parallel Processing.

  With `--default-parallelism=0` and no `--parallel-schemas` options, `mysqlpump` runs as a single-threaded process and creates no queues.

  With parallelism enabled, it is possible for output from different databases to be interleaved.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqlpump` normally reads the `[client]` and `[mysqlpump]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlpump` also reads the `[client_other]` and `[mysqlpump_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defer-table-indexes`

In the dump output, defer index creation for each table until after its rows have been loaded. This works for all storage engines, but for InnoDB applies only for secondary indexes.

This option is enabled by default; use --skip-defer-table-indexes to disable it.

- --events

  Include Event Scheduler events for the dumped databases in the output. Event dumping requires the EVENT privileges for those databases.

  The output generated by using --events contains CREATE EVENT statements to create the events.

  This option is enabled by default; use --skip-events to disable it.

- --exclude-databases=*db_list*

  Do not dump the databases in *db_list*, which is a list of one or more comma-separated database names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --exclude-events=*event_list*

  Do not dump the databases in *event_list*, which is a list of one or more comma-separated event names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --exclude-routines=*routine_list*

  Do not dump the events in *routine_list*, which is a list of one or more comma-separated routine (stored procedure or function) names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --exclude-tables=*table_list*

  Do not dump the tables in *table_list*, which is a list of one or more comma-separated table names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --exclude-triggers=*trigger_list*

  Do not dump the triggers in *trigger_list*, which is a list of one or more comma-separated trigger names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --exclude-users=*user_list*

  Do not dump the user accounts in *user_list*, which is a list of one or more comma-separated account names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --extended-insert=*N*

  Write INSERT statements using multiple-row syntax that includes several VALUES lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.

  The option value indicates the number of rows to include in each INSERT statement. The default is 250. A value of 1 produces one INSERT statement per table row.

- --get-server-public-key

Request from the server the public key required for RSA key pair-based password exchange. This option applies to clients that authenticate with the caching_sha2_password authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

If --server-public-key-path=*file_name* is given and specifies a valid public key file, it takes precedence over --get-server-public-key.

For information about the caching_sha2_password plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- --hex-blob

  Dump binary columns using hexadecimal notation (for example, 'abc' becomes 0x616263). The affected data types are BINARY, VARBINARY, BLOB types, BIT, all spatial data types, and other non-binary data types when used with the binary character set.

- --host=*host_name*, -h *host_name*

  Dump data from the MySQL server on the given host.

- --include-databases=*db_list*

  Dump the databases in *db_list*, which is a list of one or more comma-separated database names. The dump includes all objects in the named databases. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --include-events=*event_list*

  Dump the events in *event_list*, which is a list of one or more comma-separated event names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --include-routines=*routine_list*

  Dump the routines in *routine_list*, which is a list of one or more comma-separated routine (stored procedure or function) names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --include-tables=*table_list*

  Dump the tables in *table_list*, which is a list of one or more comma-separated table names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --include-triggers=*trigger_list*

  Dump the triggers in *trigger_list*, which is a list of one or more comma-separated trigger names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --include-users=*user_list*

  Dump the user accounts in *user_list*, which is a list of one or more comma-separated user names. Multiple instances of this option are additive. For more information, see mysqlpump Object Selection.

- --insert-ignore

  Write INSERT IGNORE statements rather than INSERT statements.

- `--log-error-file=file_name`

  Log warnings and errors by appending them to the named file. If this option is not given, `mysqlpump` writes warnings and errors to the standard error output.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A "login path" is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.7, "`mysql_config_editor` — MySQL Configuration Utility".

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--max-allowed-packet=N`

  The maximum size of the buffer for client/server communication. The default is 24MB, the maximum is 1GB.

- `--net-buffer-length=N`

  The initial size of the buffer for client/server communication. When creating multiple-row `INSERT` statements (as with the `--extended-insert` option), `mysqlpump` creates rows up to `N` bytes long. If you use this option to increase the value, ensure that the MySQL server `net_buffer_length` system variable has a value at least this large.

- `--no-create-db`

  Suppress any `CREATE DATABASE` statements that might otherwise be included in the output.

- `--no-create-info`, `-t`

  Do not write `CREATE TABLE` statements that create each dumped table.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.7, "`mysql_config_editor` — MySQL Configuration Utility".)

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--parallel-schemas=[N:]db_list`

  Create a queue for processing the databases in `db_list`, which is a list of one or more comma-separated database names. If `N` is given, the queue uses `N` threads. If `N` is not given, the `--default-parallelism` option determines the number of queue threads.

  Multiple instances of this option create multiple queues. `mysqlpump` also creates a default queue to use for databases not named in any `--parallel-schemas` option, and for dumping user definitions if command options select them. For more information, see mysqlpump Parallel Processing.

- `--password[=password]`, `-p[password]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, `mysqlpump` prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

  To explicitly specify that there is no password and that `mysqlpump` should not prompt for one, use the `--skip-password` option.

- `--plugin-dir=dir_name`

  The directory in which to look for plugins. Specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysqlpump` does not find it. See Section 6.2.17, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  For TCP/IP connections, the port number to use.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

- `--replace`

  Write `REPLACE` statements rather than `INSERT` statements.

- `--result-file=file_name`

  Direct output to the named file. The result file is created and its previous contents overwritten, even if an error occurs while generating the dump.

  This option should be used on Windows to prevent newline `\n` characters from being converted to `\r\n` carriage return/newline sequences.

- `--routines`

  Include stored routines (procedures and functions) for the dumped databases in the output. This option requires the global `SELECT` privilege.

  The output generated by using `--routines` contains `CREATE PROCEDURE` and `CREATE FUNCTION` statements to create the routines.

  This option is enabled by default; use `--skip-routines` to disable it.

- `--secure-auth`

This option was removed in MySQL 8.0.3.

- `--server-public-key-path=file_name`

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  For `sha256_password`, this option applies only if MySQL was built using OpenSSL.

  For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--set-charset`

  Write `SET NAMES default_character_set` to the output.

  This option is enabled by default. To disable it and suppress the `SET NAMES` statement, use `--skip-set-charset`.

- `--set-gtid-purged=value`

  This option enables control over global transaction ID (GTID) information written to the dump file, by indicating whether to add a `SET @@GLOBAL.gtid_purged` statement to the output. This option may also cause a statement to be written to the output that disables binary logging while the dump file is being reloaded.

  The following table shows the permitted option values. The default value is `AUTO`.

Value	Meaning
`OFF`	Add no `SET` statement to the output.
`ON`	Add a `SET` statement to the output. An error occurs if GTIDs are not enabled on the server.
`AUTO`	Add a `SET` statement to the output if GTIDs are enabled on the server.

  The `--set-gtid-purged` option has the following effect on binary logging when the dump file is reloaded:

  - `--set-gtid-purged=OFF`: `SET @@SESSION.SQL_LOG_BIN=0;` is not added to the output.
  - `--set-gtid-purged=ON`: `SET @@SESSION.SQL_LOG_BIN=0;` is added to the output.
  - `--set-gtid-purged=AUTO`: `SET @@SESSION.SQL_LOG_BIN=0;` is added to the output if GTIDs are enabled on the server you are backing up (that is, if `AUTO` evaluates to `ON`).

- `--single-transaction`

  This option sets the transaction isolation mode to `REPEATABLE READ` and sends a `START TRANSACTION` SQL statement to the server before dumping data. It is useful only with transactional

tables such as InnoDB, because then it dumps the consistent state of the database at the time when START TRANSACTION was issued without blocking any applications.

When using this option, you should keep in mind that only InnoDB tables are dumped in a consistent state. For example, any MyISAM or MEMORY tables dumped while using this option may still change state.

While a --single-transaction dump is in process, to ensure a valid dump file (correct table contents and binary log coordinates), no other connection should use the following statements: ALTER TABLE, CREATE TABLE, DROP TABLE, RENAME TABLE, TRUNCATE TABLE. A consistent read is not isolated from those statements, so use of them on a table to be dumped can cause the SELECT that is performed by mysqlpump to retrieve the table contents to obtain incorrect contents or fail.

--add-locks and --single-transaction are mutually exclusive.

- --skip-definer

  Omit DEFINER and SQL SECURITY clauses from the CREATE statements for views and stored programs. The dump file, when reloaded, creates objects that use the default DEFINER and SQL SECURITY values. See Section 24.6, “Stored Object Access Control”.

- --skip-dump-rows, -d

  Do not dump table rows.

- --socket=*path*, -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --ssl*

  Options that begin with --ssl specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- --ssl-fips-mode={OFF|ON|STRICT}

  Controls whether to enable FIPS mode on the client side. The --ssl-fips-mode option differs from other --ssl-*xxx* options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These --ssl-fips-mode values are permitted:

  - OFF: Disable FIPS mode.
  - ON: Enable FIPS mode.
  - STRICT: Enable “strict” FIPS mode.

  Note

  If the OpenSSL FIPS Object Module is not available, the only permitted value for --ssl-fips-mode is OFF. In this case, setting --ssl-fips-mode to ON or

> STRICT causes the client to produce a warning at startup and to operate in non-FIPS mode.

- --tls-ciphersuites=*ciphersuite_list*

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- --tls-version=*protocol_list*

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- --triggers

  Include triggers for each dumped table in the output.

  This option is enabled by default; use --skip-triggers to disable it.

- --tz-utc

  This option enables TIMESTAMP columns to be dumped and reloaded between servers in different time zones. mysqlpump sets its connection time zone to UTC and adds SET TIME_ZONE='+00:00' to the dump file. Without this option, TIMESTAMP columns are dumped and reloaded in the time zones local to the source and destination servers, which can cause the values to change if the servers are in different time zones. --tz-utc also protects against changes due to daylight saving time.

  This option is enabled by default; use --skip-tz-utc to disable it.

- --user=*user_name*, -u *user_name*

  The user name of the MySQL account to use for connecting to the server.

- --users

  Dump user accounts as logical definitions in the form of CREATE USER and GRANT statements.

  User definitions are stored in the grant tables in the mysql system database. By default, mysqlpump does not include the grant tables in mysql database dumps. To dump the contents of the grant tables as logical definitions, use the --users option and suppress all database dumping:

  ```
  shell> mysqlpump --exclude-databases=% --users
  ```

- --version, -V

  Display version information and exit.

- --watch-progress

  Periodically display a progress indicator that provides information about the completed and total number of tables, rows, and other objects.

This option is enabled by default; use `--skip-watch-progress` to disable it.

- `--zstd-compression-level=level`

  The compression level to use for connections to the server that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. The compression level setting has no effect on connections that do not use `zstd` compression.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

mysqlpump Object Selection

`mysqlpump` has a set of inclusion and exclusion options that enable filtering of several object types and provide flexible control over which objects to dump:

- `--include-databases` and `--exclude-databases` apply to databases and all objects within them.
- `--include-tables` and `--exclude-tables` apply to tables. These options also affect triggers associated with tables unless the trigger-specific options are given.
- `--include-triggers` and `--exclude-triggers` apply to triggers.
- `--include-routines` and `--exclude-routines` apply to stored procedures and functions. If a routine option matches a stored procedure name, it also matches a stored function of the same name.
- `--include-events` and `--exclude-events` apply to Event Scheduler events.
- `--include-users` and `--exclude-users` apply to user accounts.

Any inclusion or exclusion option may be given multiple times. The effect is additive. Order of these options does not matter.

The value of each inclusion and exclusion option is a list of comma-separated names of the appropriate object type. For example:

```
--exclude-databases=test,world
--include-tables=customer,invoice
```

Wildcard characters are permitted in the object names:

- `%` matches any sequence of zero or more characters.
- `_` matches any single character.

For example, `--include-tables=t%,__tmp` matches all table names that begin with `t` and all five-character table names that end with `tmp`.

For users, a name specified without a host part is interpreted with an implied host of `%`. For example, `u1` and `u1@%` are equivalent. This is the same equivalence that applies in MySQL generally (see Section 6.2.4, “Specifying Account Names”).

Inclusion and exclusion options interact as follows:

- By default, with no inclusion or exclusion options, `mysqlpump` dumps all databases (with certain exceptions noted in mysqlpump Restrictions).

- If inclusion options are given in the absence of exclusion options, only the objects named as included are dumped.
- If exclusion options are given in the absence of inclusion options, all objects are dumped except those named as excluded.
- If inclusion and exclusion options are given, all objects named as excluded and not named as included are not dumped. All other objects are dumped.

If multiple databases are being dumped, it is possible to name tables, triggers, and routines in a specific database by qualifying the object names with the database name. The following command dumps databases `db1` and `db2`, but excludes tables `db1.t1` and `db2.t2`:

```
shell> mysqlpump --include-databases=db1,db2 --exclude-tables=db1.t1,db2.t2
```

The following options provide alternative ways to specify which databases to dump:

- The `--all-databases` option dumps all databases (with certain exceptions noted in mysqlpump Restrictions). It is equivalent to specifying no object options at all (the default `mysqlpump` action is to dump everything).

  `--include-databases=%` is similar to `--all-databases`, but selects all databases for dumping, even those that are exceptions for `--all-databases`.

- The `--databases` option causes `mysqlpump` to treat all name arguments as names of databases to dump. It is equivalent to an `--include-databases` option that names the same databases.

mysqlpump Parallel Processing

`mysqlpump` can use parallelism to achieve concurrent processing. You can select concurrency between databases (to dump multiple databases simultaneously) and within databases (to dump multiple objects from a given database simultaneously).

By default, `mysqlpump` sets up one queue with two threads. You can create additional queues and control the number of threads assigned to each one, including the default queue:

- `--default-parallelism=`*`N`* specifies the default number of threads used for each queue. In the absence of this option, *`N`* is 2.

  The default queue always uses the default number of threads. Additional queues use the default number of threads unless you specify otherwise.

- `--parallel-schemas=[`*`N`*`:]`*`db_list`* sets up a processing queue for dumping the databases named in *`db_list`* and optionally specifies how many threads the queue uses. *`db_list`* is a list of comma-separated database names. If the option argument begins with *`N`*`:`, the queue uses *`N`* threads. Otherwise, the `--default-parallelism` option determines the number of queue threads.

  Multiple instances of the `--parallel-schemas` option create multiple queues.

  Names in the database list are permitted to contain the same `%` and `_` wildcard characters supported for filtering options (see mysqlpump Object Selection).

`mysqlpump` uses the default queue for processing any databases not named explicitly with a `--parallel-schemas` option, and for dumping user definitions if command options select them.

In general, with multiple queues, `mysqlpump` uses parallelism between the sets of databases processed by the queues, to dump multiple databases simultaneously. For a queue that uses multiple threads, `mysqlpump` uses parallelism within databases, to dump multiple objects from a given database

simultaneously. Exceptions can occur; for example, mysqlpump may block queues while it obtains from the server lists of objects in databases.

With parallelism enabled, it is possible for output from different databases to be interleaved. For example, INSERT statements from multiple tables dumped in parallel can be interleaved; the statements are not written in any particular order. This does not affect reloading because output statements qualify object names with database names or are preceded by USE statements as required.

The granularity for parallelism is a single database object. For example, a single table cannot be dumped in parallel using multiple threads.

Examples:

```
shell> mysqlpump --parallel-schemas=db1,db2 --parallel-schemas=db3
```

mysqlpump sets up a queue to process db1 and db2, another queue to process db3, and a default queue to process all other databases. All queues use two threads.

```
shell> mysqlpump --parallel-schemas=db1,db2 --parallel-schemas=db3
        --default-parallelism=4
```

This is the same as the previous example except that all queues use four threads.

```
shell> mysqlpump --parallel-schemas=5:db1,db2 --parallel-schemas=3:db3
```

The queue for db1 and db2 uses five threads, the queue for db3 uses three threads, and the default queue uses the default of two threads.

As a special case, with --default-parallelism=0 and no --parallel-schemas options, mysqlpump runs as a single-threaded process and creates no queues.

mysqlpump Restrictions

mysqlpump does not dump the performance_schema, ndbinfo, or sys schema by default. To dump any of these, name them explicitly on the command line. You can also name them with the --databases or --include-databases option.

mysqlpump does not dump the INFORMATION_SCHEMA schema.

mysqlpump does not dump InnoDB CREATE TABLESPACE statements.

mysqlpump dumps user accounts in logical form using CREATE USER and GRANT statements (for example, when you use the --include-users or --users option). For this reason, dumps of the mysql system database do not by default include the grant tables that contain user definitions: user, db, tables_priv, columns_priv, procs_priv, or proxies_priv. To dump any of the grant tables, name the mysql database followed by the table names:

```
shell> mysqlpump mysql user db ...
```

4.5.7 mysqlshow — Display Database, Table, and Column Information

The mysqlshow client can be used to quickly see which databases exist, their tables, or a table's columns or indexes.

mysqlshow provides a command-line interface to several SQL SHOW statements. See Section 13.7.7, “SHOW Syntax”. The same information can be obtained by using those statements directly. For example, you can issue them from the mysql client program.

Invoke mysqlshow like this:

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- If no database is given, a list of database names is shown.
- If no table is given, all matching tables in the database are shown.
- If no column is given, all matching columns and column types in the table are shown.

The output displays only the names of those databases, tables, or columns for which you have some privileges.

If the last argument contains shell or SQL wildcard characters (*, ?, %, or _), only those names that are matched by the wildcard are shown. If a database name contains any underscores, those should be escaped with a backslash (some Unix shells require two) to get a list of the proper tables or columns. * and ? characters are converted into SQL % and _ wildcard characters. This might cause some confusion when you try to display the columns for a table with a _ in the name, because in this case, mysqlshow shows you only the table names that match the pattern. This is easily fixed by adding an extra % last on the command line as a separate argument.

mysqlshow supports the following options, which can be specified on the command line or in the [mysqlshow] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.17 mysqlshow Options

Option Name	Description	Introduced	Deprecated	Removed
--bind-address	Use specified network interface to connect to MySQL Server			
--compress	Compress all information sent between client and server		8.0.18	
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18		
--count	Show the number of rows per table			
--debug	Write debugging log			
--debug-check	Print debugging information when program exits			
--debug-info	Print debugging information, memory, and CPU statistics when program exits			
--default-auth	Authentication plugin to use			
--default-character-set	Specify default character set			
--defaults-extra-file	Read named option file in addition to usual option files			
--defaults-file	Read only named option file			
--defaults-group-suffix	Option group suffix value			
--enable-cleartext-plugin	Enable cleartext authentication plugin			
--get-server-public-key	Request RSA public key from server	8.0.3		
--help	Display help message and exit			

Option Name	Description	Introduced	Deprecated	Removed
--host	Host on which MySQL server is located			
--keys	Show table indexes			
--login-path	Read login path options from .mylogin.cnf			
--no-defaults	Read no option files			
--password	Password to use when connecting to server			
--pipe	Connect to server using named pipe (Windows only)			
--plugin-dir	Directory where plugins are installed			
--port	TCP/IP port number for connection			
--print-defaults	Print default options			
--protocol	Connection protocol to use			
--secure-auth	Do not send passwords to server in old (pre-4.1) format			8.0.3
--server-public-key-path	Path name to file containing RSA public key	8.0.4		
--shared-memory-base-name	Name of shared memory to use for shared-memory connections			
--show-table-type	Show a column indicating the table type			
--socket	Unix socket file or Windows named pipe to use			
--ssl-ca	File that contains list of trusted SSL Certificate Authorities			
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files			
--ssl-cert	File that contains X.509 certificate			
--ssl-cipher	Permissible ciphers for connection encryption			
--ssl-crl	File that contains certificate revocation lists			
--ssl-crlpath	Directory that contains certificate revocation-list files			
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11		
--ssl-key	File that contains X.509 key			
--ssl-mode	Desired security state of connection to server			
--status	Display extra information about each table			
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16		
--tls-version	Permissible TLS protocols for encrypted connections			
--user	MySQL user name to use when connecting to server			
--verbose	Verbose mode			
--version	Display version information and exit			
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18		

- `--help`, `-?`

  Display a help message and exit.

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=dir_name`

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

  As of MySQL 8.0.18, this option is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `--compression-algorithms=value`

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. The default value is `uncompressed`.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

- `--count`

  Show the number of rows per table. This can be slow for non-`MyISAM` tables.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.15, “Character Set Configuration”.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--defaults-extra-file=file_name`

Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, `mysqlshow` normally reads the `[client]` and `[mysqlshow]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlshow` also reads the `[client_other]` and `[mysqlshow_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.4, "Client-Side Cleartext Pluggable Authentication".)

- `--get-server-public-key`

  Request from the server the RSA public key that it uses for key pair-based password exchange. This option applies to clients that connect to the server using an account that authenticates with the `caching_sha2_password` authentication plugin. For connections by such accounts, the server does not send the public key to the client unless requested. The option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not needed, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  For information about the `caching_sha2_password` plugin, see Section 6.4.1.2, "Caching SHA-2 Pluggable Authentication".

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--keys`, `-k`

  Show table indexes.

- `--login-path=name`

Read options from the named login path in the .mylogin.cnf login path file. A "login path" is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the mysql_config_editor utility. See Section 4.6.7, "mysql_config_editor — MySQL Configuration Utility".

For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.7, "mysql_config_editor — MySQL Configuration Utility".)

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --password[=*password*], -p[*password*]

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, mysqlshow prompts for one. If given, there must be *no space* between --password= or -p and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, "End-User Guidelines for Password Security".

  To explicitly specify that there is no password and that mysqlshow should not prompt for one, use the --skip-password option.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysqlshow does not find it. See Section 6.2.17, "Pluggable Authentication".

- --port=*port_num*, -P *port_num*

  For TCP/IP connections, the port number to use.

- --print-defaults

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

- `--secure-auth`

  This option was removed in MySQL 8.0.3.

- `--server-public-key-path=file_name`

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  For `sha256_password`, this option applies only if MySQL was built using OpenSSL.

  For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--show-table-type`, `-t`

  Show a column indicating the table type, as in `SHOW FULL TABLES`. The type is `BASE TABLE` or `VIEW`.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--ssl-fips-mode={OFF|ON|STRICT}`

Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

These `--ssl-fips-mode` values are permitted:

- `OFF`: Disable FIPS mode.
- `ON`: Enable FIPS mode.
- `STRICT`: Enable “strict” FIPS mode.

Note

If the OpenSSL FIPS Object Module is not available, the only permitted value for `--ssl-fips-mode` is `OFF`. In this case, setting `--ssl-fips-mode` to `ON` or `STRICT` causes the client to produce a warning at startup and to operate in non-FIPS mode.

- `--status`, `-i`

  Display extra information about each table.

- `--tls-ciphersuites=ciphersuite_list`

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- `--tls-version=protocol_list`

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `--user=user_name`, `-u user_name`

  The user name of the MySQL account to use for connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version`, `-V`

  Display version information and exit.

- `--zstd-compression-level=level`

  The compression level to use for connections to the server that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The

default zstd compression level is 3. The compression level setting has no effect on connections that do not use zstd compression.

For more information, see Section 4.2.6, “Connection Compression Control”.

This option was added in MySQL 8.0.18.

4.5.8 mysqlslap — Load Emulation Client

mysqlslap is a diagnostic program designed to emulate client load for a MySQL server and to report the timing of each stage. It works as if multiple clients are accessing the server.

Invoke mysqlslap like this:

```
shell> mysqlslap [options]
```

Some options such as --create or --query enable you to specify a string containing an SQL statement or a file containing statements. If you specify a file, by default it must contain one statement per line. (That is, the implicit statement delimiter is the newline character.) Use the --delimiter option to specify a different delimiter, which enables you to specify statements that span multiple lines or place multiple statements on a single line. You cannot include comments in a file; mysqlslap does not understand them.

mysqlslap runs in three stages:

1. Create schema, table, and optionally any stored programs or data to use for the test. This stage uses a single client connection.
2. Run the load test. This stage can use many client connections.
3. Clean up (disconnect, drop table if specified). This stage uses a single client connection.

Examples:

Supply your own create and query SQL statements, with 50 clients querying and 200 selects for each (enter the command on a single line):

```
mysqlslap --delimiter=";"
  --create="CREATE TABLE a (b int);INSERT INTO a VALUES (23)"
  --query="SELECT * FROM a" --concurrency=50 --iterations=200
```

Let mysqlslap build the query SQL statement with a table of two INT columns and three VARCHAR columns. Use five clients querying 20 times each. Do not create the table or insert the data (that is, use the previous test's schema and data):

```
mysqlslap --concurrency=5 --iterations=20
  --number-int-cols=2 --number-char-cols=3
  --auto-generate-sql
```

Tell the program to load the create, insert, and query SQL statements from the specified files, where the create.sql file has multiple table creation statements delimited by ';' and multiple insert statements delimited by ';'. The --query file will have multiple queries delimited by ';'. Run all the load statements, then run all the queries in the query file with five clients (five times each):

```
mysqlslap --concurrency=5
  --iterations=5 --query=query.sql --create=create.sql
  --delimiter=";"
```

mysqlslap supports the following options, which can be specified on the command line or in the [mysqlslap] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.18 mysqlslap Options

Option Name	Description	Introduced	Deprecated	Removed
--auto-generate-sql	Generate SQL statements automatically when they are not supplied in files or using command options			
--auto-generate-sql-add-autoincrement	Add AUTO_INCREMENT column to automatically generated tables			
--auto-generate-sql-execute-number	Specify how many queries to generate automatically			
--auto-generate-sql-guid-primary	Add a GUID-based primary key to automatically generated tables			
--auto-generate-sql-load-type	Specify the test load type			
--auto-generate-sql-secondary-indexes	Specify how many secondary indexes to add to automatically generated tables			
--auto-generate-sql-unique-query-number	How many different queries to generate for automatic tests.			
--auto-generate-sql-unique-write-number	How many different queries to generate for --auto-generate-sql-write-number			
--auto-generate-sql-write-number	How many row inserts to perform on each thread			
--commit	How many statements to execute before committing.			
--compress	Compress all information sent between client and server		8.0.18	
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18		
--concurrency	Number of clients to simulate when issuing the SELECT statement			
--create	File or string containing the statement to use for creating the table			
--create-schema	Schema in which to run the tests			
--csv	Generate output in comma-separated values format			
--debug	Write debugging log			
--debug-check	Print debugging information when program exits			
--debug-info	Print debugging information, memory, and CPU statistics when program exits			
--default-auth	Authentication plugin to use			
--defaults-extra-file	Read named option file in addition to usual option files			

Option Name	Description	Introduced	Deprecated	Removed
--defaults-file	Read only named option file			
--defaults-group-suffix	Option group suffix value			
--delimiter	Delimiter to use in SQL statements			
--detach	Detach (close and reopen) each connection after each N statements			
--enable-cleartext-plugin	Enable cleartext authentication plugin			
--engine	Storage engine to use for creating the table			
--get-server-public-key	Request RSA public key from server	8.0.3		
--help	Display help message and exit			
--host	Host on which MySQL server is located			
--iterations	Number of times to run the tests			
--login-path	Read login path options from .mylogin.cnf			
--no-defaults	Read no option files			
--no-drop	Do not drop any schema created during the test run			
--number-char-cols	Number of VARCHAR columns to use if --auto-generate-sql is specified			
--number-int-cols	Number of INT columns to use if --auto-generate-sql is specified			
--number-of-queries	Limit each client to approximately this number of queries			
--only-print	Do not connect to databases. mysqlslap only prints what it would have done			
--password	Password to use when connecting to server			
--pipe	Connect to server using named pipe (Windows only)			
--plugin-dir	Directory where plugins are installed			
--port	TCP/IP port number for connection			
--post-query	File or string containing the statement to execute after the tests have completed			
--post-system	String to execute using system() after the tests have completed			
--pre-query	File or string containing the statement to execute before running the tests			
--pre-system	String to execute using system() before running the tests			
--print-defaults	Print default options			
--protocol	Connection protocol to use			
--query	File or string containing the SELECT statement to use for retrieving data			

Option Name	Description	Introduced	Deprecated	Removed
--secure-auth	Do not send passwords to server in old (pre-4.1) format			8.0.3
--server-public-key-path	Path name to file containing RSA public key	8.0.4		
--shared-memory-base-name	Name of shared memory to use for shared-memory connections			
--silent	Silent mode			
--socket	Unix socket file or Windows named pipe to use			
--sql-mode	Set SQL mode for client session			
--ssl-ca	File that contains list of trusted SSL Certificate Authorities			
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files			
--ssl-cert	File that contains X.509 certificate			
--ssl-cipher	Permissible ciphers for connection encryption			
--ssl-crl	File that contains certificate revocation lists			
--ssl-crlpath	Directory that contains certificate revocation-list files			
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11		
--ssl-key	File that contains X.509 key			
--ssl-mode	Desired security state of connection to server			
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16		
--tls-version	Permissible TLS protocols for encrypted connections			
--user	MySQL user name to use when connecting to server			
--verbose	Verbose mode			
--version	Display version information and exit			
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18		

- `--help`, `-?`

  Display a help message and exit.

- `--auto-generate-sql`, `-a`

  Generate SQL statements automatically when they are not supplied in files or using command options.

- `--auto-generate-sql-add-autoincrement`

  Add an `AUTO_INCREMENT` column to automatically generated tables.

- `--auto-generate-sql-execute-number=N`

  Specify how many queries to generate automatically.

- `--auto-generate-sql-guid-primary`

  Add a GUID-based primary key to automatically generated tables.

- `--auto-generate-sql-load-type=type`

  Specify the test load type. The permissible values are `read` (scan tables), `write` (insert into tables), `key` (read primary keys), `update` (update primary keys), or `mixed` (half inserts, half scanning selects). The default is `mixed`.

- `--auto-generate-sql-secondary-indexes=N`

  Specify how many secondary indexes to add to automatically generated tables. By default, none are added.

- `--auto-generate-sql-unique-query-number=N`

  How many different queries to generate for automatic tests. For example, if you run a `key` test that performs 1000 selects, you can use this option with a value of 1000 to run 1000 unique queries, or with a value of 50 to perform 50 different selects. The default is 10.

- `--auto-generate-sql-unique-write-number=N`

  How many different queries to generate for `--auto-generate-sql-write-number`. The default is 10.

- `--auto-generate-sql-write-number=N`

  How many row inserts to perform. The default is 100.

- `--commit=N`

  How many statements to execute before committing. The default is 0 (no commits are done).

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

  As of MySQL 8.0.18, this option is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `--compression-algorithms=value`

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. The default value is `uncompressed`.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

- `--concurrency=N`, `-c N`

  The number of parallel clients to simulate.

- `--create=value`

  The file or string containing the statement to use for creating the table.

- `--create-schema=value`

  The schema in which to run the tests.

  Note

  If the `--auto-generate-sql` option is also given, `mysqlslap` drops the schema at the end of the test run. To avoid this, use the `--no-drop` option as well.

- `--csv[=file_name]`

  Generate output in comma-separated values format. The output goes to the named file, or to the standard output if no file is given.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysqlslap.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`, `-T`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqlslap` normally reads the `[client]` and `[mysqlslap]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlslap` also reads the `[client_other]` and `[mysqlslap_other]` groups.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--delimiter=str`, `-F str`

  The delimiter to use in SQL statements supplied in files or using command options.

- `--detach=N`

  Detach (close and reopen) each connection after each `N` statements. The default is 0 (connections are not detached).

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.)

- `--engine=engine_name`, `-e engine_name`

  The storage engine to use for creating tables.

- `--get-server-public-key`

  Request from the server the RSA public key that it uses for key pair-based password exchange. This option applies to clients that connect to the server using an account that authenticates with the `caching_sha2_password` authentication plugin. For connections by such accounts, the server does not send the public key to the client unless requested. The option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not needed, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  For information about the `caching_sha2_password` plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--iterations=N`, `-i N`

  The number of times to run the tests.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-drop`

  Prevent `mysqlslap` from dropping any schema it creates during the test run.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.7, "`mysql_config_editor` — MySQL Configuration Utility".)

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--number-char-cols=N`, `-x N`

  The number of `VARCHAR` columns to use if `--auto-generate-sql` is specified.

- `--number-int-cols=N`, `-y N`

  The number of `INT` columns to use if `--auto-generate-sql` is specified.

- `--number-of-queries=N`

  Limit each client to approximately this many queries. Query counting takes into account the statement delimiter. For example, if you invoke `mysqlslap` as follows, the `;` delimiter is recognized so that each instance of the query string counts as two queries. As a result, 5 rows (not 10) are inserted.

  ```
  shell> mysqlslap --delimiter=";" --number-of-queries=10
           --query="use test;insert into t values(null)"
  ```

- `--only-print`

  Do not connect to databases. `mysqlslap` only prints what it would have done.

- `--password[=password]`, `-p[password]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, `mysqlslap` prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, "End-User Guidelines for Password Security".

  To explicitly specify that there is no password and that `mysqlslap` should not prompt for one, use the `--skip-password` option.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--plugin-dir=dir_name`

The directory in which to look for plugins. Specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysqlslap` does not find it. See Section 6.2.17, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  For TCP/IP connections, the port number to use.

- `--post-query=value`

  The file or string containing the statement to execute after the tests have completed. This execution is not counted for timing purposes.

- `--post-system=str`

  The string to execute using `system()` after the tests have completed. This execution is not counted for timing purposes.

- `--pre-query=value`

  The file or string containing the statement to execute before running the tests. This execution is not counted for timing purposes.

- `--pre-system=str`

  The string to execute using `system()` before running the tests. This execution is not counted for timing purposes.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

- `--query=value`, `-q value`

  The file or string containing the `SELECT` statement to use for retrieving data.

- `--secure-auth`

  This option was removed in MySQL 8.0.3.

- `--server-public-key-path=file_name`

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

For `sha256_password`, this option applies only if MySQL was built using OpenSSL.

For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--silent`, `-s`

  Silent mode. No output.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--sql-mode=mode`

  Set the SQL mode for the client session.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--ssl-fips-mode={OFF|ON|STRICT}`

  Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These `--ssl-fips-mode` values are permitted:

  - `OFF`: Disable FIPS mode.
  - `ON`: Enable FIPS mode.
  - `STRICT`: Enable “strict” FIPS mode.

  Note

  If the OpenSSL FIPS Object Module is not available, the only permitted value for `--ssl-fips-mode` is `OFF`. In this case, setting `--ssl-fips-mode` to `ON` or

> `STRICT` causes the client to produce a warning at startup and to operate in non-FIPS mode.

- `--tls-ciphersuites=`*`ciphersuite_list`*

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- `--tls-version=`*`protocol_list`*

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The user name of the MySQL account to use for connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version`, `-V`

  Display version information and exit.

- `--zstd-compression-level=`*`level`*

  The compression level to use for connections to the server that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. The compression level setting has no effect on connections that do not use `zstd` compression.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

4.6 MySQL Administrative and Utility Programs

This section describes administrative programs and programs that perform miscellaneous utility operations.

4.6.1 `ibd2sdi` — InnoDB Tablespace SDI Extraction Utility

`ibd2sdi` is a utility for extracting serialized dictionary information (SDI) from `InnoDB` tablespace files. SDI data is present all persistent `InnoDB` tablespace files.

`ibd2sdi` can be run on file-per-table tablespace files (`*.ibd` files), general tablespace files (`*.ibd` files), system tablespace files (`ibdata*` files), and the data dictionary tablespace (`mysql.ibd`). It is not supported for use with temporary tablespaces or undo tablespaces.

ibd2sdi can be used at runtime or while the server is offline. During DDL operations, ROLLBACK operations, and undo log purge operations related to SDI, there may be a short interval of time when ibd2sdi fails to read SDI data stored in the tablespace.

ibd2sdi performs an uncommitted read of SDI from the specified tablespace. Redo logs and undo logs are not accessed.

Invoke the ibd2sdi utility like this:

```
shell> ibd2sdi [options] file_name1 [file_name2 file_name3 ...]
```

ibd2sdi supports multi-file tablespaces like the InnoDB system tablespace, but it cannot be run on more than one tablespace at a time. For multi-file tablespaces, specify each file:

```
shell> ibd2sdi ibdata1 ibdata2
```

The files of a multi-file tablespace must be specified in order of the ascending page number. If two successive files have the same space ID, the later file must start with the last page number of the previous file + 1.

ibd2sdi outputs SDI (containing id, type, and data fields) in JSON format.

ibd2sdi Options

ibd2sdi supports the following options:

- --help, -h

  Displays command-line help.

```
shell> ibd2sdi --help
Usage: ./ibd2sdi [-v] [-c <strict-check>] [-d <dump file name>] [-n] filename1 [filenames]
See http://dev.mysql.com/doc/refman/8.0/en/ibd2sdi.html for usage hints.
  -h, --help          Display this help and exit.
  -v, --version       Display version information and exit.
  -#, --debug[=name]  Output debug log. See
                      http://dev.mysql.com/doc/refman/8.0/en/dbug-package.html
  -d, --dump-file=name
                      Dump the tablespace SDI into the file passed by user.
                      Without the filename, it will default to stdout
  -s, --skip-data     Skip retrieving data from SDI records. Retrieve only id
                      and type.
  -i, --id=#          Retrieve the SDI record matching the id passed by user.
  -t, --type=#        Retrieve the SDI records matching the type passed by
                      user.
  -c, --strict-check=name
                      Specify the strict checksum algorithm by the user.
                      Allowed values are innodb, crc32, none.
  -n, --no-check      Ignore the checksum verification.
  -p, --pretty        Pretty format the SDI output.If false, SDI would be not
                      human readable but it will be of less size
                      (Defaults to on; use --skip-pretty to disable.)

Variables (--variable-name=value)
and boolean options {FALSE|TRUE}  Value (after reading options)
--------------------------------- ----------------------------------------
debug                             (No default value)
dump-file                         (No default value)
skip-data                         FALSE
id                                0
type                              0
```

```
strict-check                      crc32
no-check                          FALSE
pretty                            TRUE
```

- `--version`, `-v`

  Displays MySQL version information.

```
shell> ibd2sdi --version
ibd2sdi  Ver 8.0.3-dmr for Linux on x86_64 (Source distribution)
```

- `--debug[=debug_options]`, `-# [debug_options]`

  Prints a debug log. For debug options, refer to Section 29.5.4, “The DBUG Package”.

```
shell> ibd2sdi --debug=d:t /tmp/ibd2sdi.trace
```

- `--dump-file=`, `-d`

  Dumps serialized dictionary information (SDI) into the specified dump file. If a dump file is not specified, the tablespace SDI is dumped to `stdout`.

```
shell> ibd2sdi --dump-file=file_name ../data/test/t1.ibd
```

- `--skip-data`, `-s`

  Skips retrieval of `data` field values from the serialized dictionary information (SDI) and only retrieves the `id` and `type` field values, which are primary keys for SDI records.

```
shell> ibd2sdi --skip-data ../data/test/t1.ibd
["ibd2sdi"
,
{
 "type": 1,
 "id": 330
}
,
{
 "type": 2,
 "id": 7
}
]
```

- `--id=#`, `-i #`

  Retrieves serialized dictionary information (SDI) matching the specified table or tablespace object id. An object id is unique to the object type. Table and tablespace object IDs are also found in the `id` column of the `mysql.tables` and `mysql.tablespace` data dictionary tables. For information about data dictionary tables, see Section 14.1, “Data Dictionary Schema”.

```
shell> ibd2sdi --id=7 ../data/test/t1.ibd
["ibd2sdi"
,
{
 "type": 2,
 "id": 7,
 "object":
  {
    "mysqld_version_id": 80003,
```

```
    "dd_version": 80003,
    "sdi_version": 1,
    "dd_object_type": "Tablespace",
    "dd_object": {
        "name": "test/t1",
        "comment": "",
        "options": "",
        "se_private_data": "flags=16417;id=2;server_version=80003;space_version=1;",
        "engine": "InnoDB",
        "files": [
            {
                "ordinal_position": 1,
                "filename": "./test/t1.ibd",
                "se_private_data": "id=2;"
            }
        ]
    }
}
}
]
```

- `--type=#`, `-t #`

  Retrieves serialized dictionary information (SDI) matching the specified object type. SDI is provided for table (type=1) and tablespace (type=2) objects.

```
shell> ibd2sdi --type=2 ../data/test/t1.ibd
["ibd2sdi"
,
{
 "type": 2,
 "id": 7,
 "object":
  {
    "mysqld_version_id": 80003,
    "dd_version": 80003,
    "sdi_version": 1,
    "dd_object_type": "Tablespace",
    "dd_object": {
        "name": "test/t1",
        "comment": "",
        "options": "",
        "se_private_data": "flags=16417;id=2;server_version=80003;space_version=1;",
        "engine": "InnoDB",
        "files": [
            {
                "ordinal_position": 1,
                "filename": "./test/t1.ibd",
                "se_private_data": "id=2;"
            }
        ]
    }
}
}
]
```

- `--strict-check`, `-c`

  Specifies a strict checksum algorithm for validating the checksum of pages that are read. Options include `innodb`, `crc32`, and `none`.

  In this example, the strict version of the `innodb` checksum algorithm is specified:

```
```

```
shell> ibd2sdi --strict-check=innodb ../data/test/t1.ibd
```

In this example, the strict version of crc32 checksum algorithm is specified:

```
shell> ibd2sdi -c crc32 ../data/test/t1.ibd
```

If you do not specify the --strict-check option, validation is performed against non-strict innodb, crc32 and none checksums.

- --no-check, -n

  Skips checksum validation for pages that are read.

```
shell> ibd2sdi --no-check ../data/test/t1.ibd
```

- --pretty, -p

  Outputs SDI data in JSON pretty print format. Enabled by default. If disabled, SDI is not human readable but is smaller in size. Use --skip-pretty to disable.

```
shell> ibd2sdi --skip-pretty ../data/test/t1.ibd
```

4.6.2 innochecksum — Offline InnoDB File Checksum Utility

innochecksum prints checksums for InnoDB files. This tool reads an InnoDB tablespace file, calculates the checksum for each page, compares the calculated checksum to the stored checksum, and reports mismatches, which indicate damaged pages. It was originally developed to speed up verifying the integrity of tablespace files after power outages but can also be used after file copies. Because checksum mismatches cause InnoDB to deliberately shut down a running server, it may be preferable to use this tool rather than waiting for an in-production server to encounter the damaged pages.

innochecksum cannot be used on tablespace files that the server already has open. For such files, you should use CHECK TABLE to check tables within the tablespace. Attempting to run innochecksum on a tablespace that the server already has open will result in an "Unable to lock file" error.

If checksum mismatches are found, you would normally restore the tablespace from backup or start the server and attempt to use mysqldump to make a backup of the tables within the tablespace.

Invoke innochecksum like this:

```
shell> innochecksum [options] file_name
```

innochecksum Options

innochecksum supports the following options. For options that refer to page numbers, the numbers are zero-based.

- --help, -?

  Displays command line help. Example usage:

```
shell> innochecksum --help
```

- --info, -I

Synonym for `--help`. Displays command line help. Example usage:

```
shell> innochecksum --info
```

- `--version`, `-V`

  Displays version information. Example usage:

  ```
  shell> innochecksum --version
  ```

- `--verbose`, `-v`

  Verbose mode; prints a progress indicator to the log file every five seconds. In order for the progress indicator to be printed, the log file must be specified using the `--log option`. To turn on `verbose` mode, run:

  ```
  shell> innochecksum --verbose
  ```

  To turn off verbose mode, run:

  ```
  shell> innochecksum --verbose=FALSE
  ```

  The `--verbose` option and `--log` option can be specified at the same time. For example:

  ```
  shell> innochecksum --verbose --log=/var/lib/mysql/test/logtest.txt
  ```

  To locate the progress indicator information in the log file, you can perform the following search:

  ```
  shell> cat ./logtest.txt | grep -i "okay"
  ```

  The progress indicator information in the log file appears similar to the following:

  ```
  page 1663 okay: 2.863% done
  page 8447 okay: 14.537% done
  page 13695 okay: 23.568% done
  page 18815 okay: 32.379% done
  page 23039 okay: 39.648% done
  page 28351 okay: 48.789% done
  page 33023 okay: 56.828% done
  page 37951 okay: 65.308% done
  page 44095 okay: 75.881% done
  page 49407 okay: 85.022% done
  page 54463 okay: 93.722% done
  ...
  ```

- `--count`, `-c`

  Print a count of the number of pages in the file and exit. Example usage:

  ```
  shell> innochecksum --count ../data/test/tab1.ibd
  ```

- `--start-page=`*`num`*, `-s` *`num`*

  Start at this page number. Example usage:

```
shell> innochecksum --start-page=600 ../data/test/tab1.ibd
```

or:

```
shell> innochecksum -s 600 ../data/test/tab1.ibd
```

- `--end-page=num`, `-e num`

  End at this page number. Example usage:

  ```
  shell> innochecksum --end-page=700 ../data/test/tab1.ibd
  ```

  or:

  ```
  shell> innochecksum --p 700 ../data/test/tab1.ibd
  ```

- `--page=num`, `-p num`

  Check only this page number. Example usage:

  ```
  shell> innochecksum --page=701 ../data/test/tab1.ibd
  ```

- `--strict-check`, `-C`

  Specify a strict checksum algorithm. Options include `innodb`, `crc32`, and `none`.

  In this example, the `innodb` checksum algorithm is specified:

  ```
  shell> innochecksum --strict-check=innodb ../data/test/tab1.ibd
  ```

  In this example, the `crc32` checksum algorithm is specified:

  ```
  shell> innochecksum -C crc32 ../data/test/tab1.ibd
  ```

  The following conditions apply:

  - If you do not specify the `--strict-check` option, `innochecksum` validates against `innodb`, `crc32` and `none`.
  - If you specify the `none` option, only checksums generated by `none` are allowed.
  - If you specify the `innodb` option, only checksums generated by `innodb` are allowed.
  - If you specify the `crc32` option, only checksums generated by `crc32` are allowed.

- `--no-check`, `-n`

  Ignore the checksum verification when rewriting a checksum. This option may only be used with the `innochecksum --write` option. If the `--write` option is not specified, `innochecksum` will terminate.

  In this example, an `innodb` checksum is rewritten to replace an invalid checksum:

  ```
  shell> innochecksum --no-check --write innodb ../data/test/tab1.ibd
  ```

- `--allow-mismatches`, `-a`

  The maximum number of checksum mismatches allowed before `innochecksum` terminates. The default setting is 0. If `--allow-mismatches=N`, where `N>=0`, `N` mismatches are permitted and `innochecksum` terminates at `N+1`. When `--allow-mismatches` is set to 0, `innochecksum` terminates on the first checksum mismatch.

  In this example, an existing `innodb` checksum is rewritten to set `--allow-mismatches` to 1.

```
shell> innochecksum --allow-mismatches=1 --write innodb ../data/test/tab1.ibd
```

  With `--allow-mismatches` set to 1, if there is a mismatch at page 600 and another at page 700 on a file with 1000 pages, the checksum is updated for pages 0-599 and 601-699. Because `--allow-mismatches` is set to 1, the checksum tolerates the first mismatch and terminates on the second mismatch, leaving page 600 and pages 700-999 unchanged.

- `--write=name`, `-w num`

  Rewrite a checksum. When rewriting an invalid checksum, the `--no-check` option must be used together with the `--write` option. The `--no-check` option tells `innochecksum` to ignore verification of the invalid checksum. You do not have to specify the `--no-check` option if the current checksum is valid.

  An algorithm must be specified when using the `--write` option. Possible values for the `--write` option are:

  - `innodb`: A checksum calculated in software, using the original algorithm from `InnoDB`.
  - `crc32`: A checksum calculated using the `crc32` algorithm, possibly done with a hardware assist.
  - `none`: A constant number.

  The `--write` option rewrites entire pages to disk. If the new checksum is identical to the existing checksum, the new checksum is not written to disk in order to minimize I/O.

  `innochecksum` obtains an exclusive lock when the `--write` option is used.

  In this example, a `crc32` checksum is written for `tab1.ibd`:

```
shell> innochecksum -w crc32 ../data/test/tab1.ibd
```

  In this example, a `crc32` checksum is rewritten to replace an invalid `crc32` checksum:

```
shell> innochecksum --no-check --write crc32 ../data/test/tab1.ibd
```

- `--page-type-summary`, `-S`

  Display a count of each page type in a tablespace. Example usage:

```
shell> innochecksum --page-type-summary ../data/test/tab1.ibd
```

  Sample output for `--page-type-summary`:

```
File::../data/test/tab1.ibd
================PAGE TYPE SUMMARY==============
```

```
#PAGE_COUNT PAGE_TYPE
===============================================
       2        Index page
       0        Undo log page
       1        Inode page
       0        Insert buffer free list page
       2        Freshly allocated page
       1        Insert buffer bitmap
       0        System page
       0        Transaction system page
       1        File Space Header
       0        Extent descriptor page
       0        BLOB page
       0        Compressed BLOB page
       0        Other type of page
===============================================
Additional information:
Undo page type: 0 insert, 0 update, 0 other
Undo page state: 0 active, 0 cached, 0 to_free, 0 to_purge, 0 prepared, 0 other
```

- `--page-type-dump`, `-D`

  Dump the page type information for each page in a tablespace to `stderr` or `stdout`. Example usage:

```
shell> innochecksum --page-type-dump=/tmp/a.txt ../data/test/tab1.ibd
```

- `--log`, `-l`

  Log output for the `innochecksum` tool. A log file name must be provided. Log output contains checksum values for each tablespace page. For uncompressed tables, LSN values are also provided. The `--log` replaces the `--debug` option, which was available in earlier releases. Example usage:

```
shell> innochecksum --log=/tmp/log.txt ../data/test/tab1.ibd
```

  or:

```
shell> innochecksum -l /tmp/log.txt ../data/test/tab1.ibd
```

- `-` option.

  Specify the `-` option to read from standard input. If the `-` option is missing when “read from standard in” is expected, `innochecksum` will output `innochecksum` usage information indicating that the “-” option was omitted. Example usages:

```
shell> cat t1.ibd | innochecksum -
```

  In this example, `innochecksum` writes the `crc32` checksum algorithm to `a.ibd` without changing the original `t1.ibd` file.

```
shell> cat t1.ibd | innochecksum --write=crc32 - > a.ibd
```

Running innochecksum on Multiple User-defined Tablespace Files

The following examples demonstrate how to run `innochecksum` on multiple user-defined tablespace files (`.ibd` files).

Run `innochecksum` for all tablespace (`.ibd`) files in the “test” database:

```
shell> innochecksum ./data/test/*.ibd
```

Run innochecksum for all tablespace files (.ibd files) that have a file name starting with “t”:

```
shell> innochecksum ./data/test/t*.ibd
```

Run innochecksum for all tablespace files (.ibd files) in the data directory:

```
shell> innochecksum ./data/*/*.ibd
```

Note

Running innochecksum on multiple user-defined tablespace files is not supported on Windows operating systems, as Windows shells such as cmd.exe do not support glob pattern expansion. On Windows systems, innochecksum must be run separately for each user-defined tablespace file. For example:

```
cmd> innochecksum.exe t1.ibd
cmd> innochecksum.exe t2.ibd
cmd> innochecksum.exe t3.ibd
```

Running innochecksum on Multiple System Tablespace Files

By default, there is only one InnoDB system tablespace file (ibdata1) but multiple files for the system tablespace can be defined using the innodb_data_file_path option. In the following example, three files for the system tablespace are defined using the innodb_data_file_path option: ibdata1, ibdata2, and ibdata3.

```
shell> ./bin/mysqld --no-defaults --innodb-data-file-path="ibdata1:10M;ibdata2:10M;ibdata3:10M:autoextend"
```

The three files (ibdata1, ibdata2, and ibdata3) form one logical system tablespace. To run innochecksum on multiple files that form one logical system tablespace, innochecksum requires the - option to read tablespace files in from standard input, which is equivalent to concatenating multiple files to create one single file. For the example provided above, the following innochecksum command would be used:

```
shell> cat ibdata* | innochecksum -
```

Refer to the innochecksum options information for more information about the “-” option.

Note

Running innochecksum on multiple files in the same tablespace is not supported on Windows operating systems, as Windows shells such as cmd.exe do not support glob pattern expansion. On Windows systems, innochecksum must be run separately for each system tablespace file. For example:

```
cmd> innochecksum.exe ibdata1
cmd> innochecksum.exe ibdata2
cmd> innochecksum.exe ibdata3
```

4.6.3 myisam_ftdump — Display Full-Text Index information

myisam_ftdump displays information about FULLTEXT indexes in MyISAM tables. It reads the MyISAM index file directly, so it must be run on the server host where the table is located. Before using myisam_ftdump, be sure to issue a FLUSH TABLES statement first if the server is running.

myisam_ftdump scans and dumps the entire index, which is not particularly fast. On the other hand, the distribution of words changes infrequently, so it need not be run often.

Invoke myisam_ftdump like this:

```
shell> myisam_ftdump [options] tbl_name index_num
```

The *tbl_name* argument should be the name of a MyISAM table. You can also specify a table by naming its index file (the file with the .MYI suffix). If you do not invoke myisam_ftdump in the directory where the table files are located, the table or index file name must be preceded by the path name to the table's database directory. Index numbers begin with 0.

Example: Suppose that the test database contains a table named mytexttable that has the following definition:

```
CREATE TABLE mytexttable
(
  id   INT NOT NULL,
  txt  TEXT NOT NULL,
  PRIMARY KEY (id),
  FULLTEXT (txt)
) ENGINE=MyISAM;
```

The index on id is index 0 and the FULLTEXT index on txt is index 1. If your working directory is the test database directory, invoke myisam_ftdump as follows:

```
shell> myisam_ftdump mytexttable 1
```

If the path name to the test database directory is /usr/local/mysql/data/test, you can also specify the table name argument using that path name. This is useful if you do not invoke myisam_ftdump in the database directory:

```
shell> myisam_ftdump /usr/local/mysql/data/test/mytexttable 1
```

You can use myisam_ftdump to generate a list of index entries in order of frequency of occurrence like this on Unix-like systems:

```
shell> myisam_ftdump -c mytexttable 1 | sort -r
```

On Windows, use:

```
shell> myisam_ftdump -c mytexttable 1 | sort /R
```

myisam_ftdump supports the following options:

- --help, -h -?

  Display a help message and exit.

- --count, -c

  Calculate per-word statistics (counts and global weights).

- --dump, -d

Dump the index, including data offsets and word weights.

- `--length`, `-l`

  Report the length distribution.

- `--stats`, `-s`

  Report global index statistics. This is the default operation if no other operation is specified.

- `--verbose`, `-v`

  Verbose mode. Print more output about what the program does.

4.6.4 myisamchk — MyISAM Table-Maintenance Utility

The `myisamchk` utility gets information about your database tables or checks, repairs, or optimizes them. `myisamchk` works with `MyISAM` tables (tables that have `.MYD` and `.MYI` files for storing data and indexes).

You can also use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See Section 13.7.3.2, “CHECK TABLE Syntax”, and Section 13.7.3.5, “REPAIR TABLE Syntax”.

The use of `myisamchk` with partitioned tables is not supported.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

Invoke `myisamchk` like this:

```
shell> myisamchk [options] tbl_name ...
```

The *`options`* specify what you want `myisamchk` to do. They are described in the following sections. You can also get a list of options by invoking `myisamchk --help`.

With no options, `myisamchk` simply checks your table as the default operation. To get more information or to tell `myisamchk` to take corrective action, specify options as described in the following discussion.

`tbl_name` is the database table you want to check or repair. If you run `myisamchk` somewhere other than in the database directory, you must specify the path to the database directory, because `myisamchk` has no idea where the database is located. In fact, `myisamchk` does not actually care whether the files you are working on are located in a database directory. You can copy the files that correspond to a database table into some other location and perform recovery operations on them there.

You can name several tables on the `myisamchk` command line if you wish. You can also specify a table by naming its index file (the file with the `.MYI` suffix). This enables you to specify all tables in a directory by using the pattern `*.MYI`. For example, if you are in a database directory, you can check all the `MyISAM` tables in that directory like this:

```
shell> myisamchk *.MYI
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell> myisamchk /path/to/database_dir/*.MYI
```

You can even check all tables in all databases by specifying a wildcard with the path to the MySQL data directory:

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

The recommended way to quickly check all MyISAM tables is:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
```

If you want to check all MyISAM tables and repair any that are corrupted, you can use the following command:

```
shell> myisamchk --silent --force --fast --update-state \
          --key_buffer_size=64M --myisam_sort_buffer_size=64M \
          --read_buffer_size=1M --write_buffer_size=1M \
          /path/to/datadir/*/*.MYI
```

This command assumes that you have more than 64MB free. For more information about memory allocation with myisamchk, see Section 4.6.4.6, “myisamchk Memory Usage”.

For additional information about using myisamchk, see Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

Important

You must ensure that no other program is using the tables while you are running myisamchk. The most effective means of doing so is to shut down the MySQL server while running myisamchk, or to lock all tables that myisamchk is being used on.

Otherwise, when you run myisamchk, it may display the following error message:

```
warning: clients are using or haven't closed the table properly
```

This means that you are trying to check a table that has been updated by another program (such as the mysqld server) that hasn't yet closed the file or that has died without closing the file properly, which can sometimes lead to the corruption of one or more MyISAM tables.

If mysqld is running, you must force it to flush any table modifications that are still buffered in memory by using FLUSH TABLES. You should then ensure that no one is using the tables while you are running myisamchk

However, the easiest way to avoid this problem is to use CHECK TABLE instead of myisamchk to check tables. See Section 13.7.3.2, “CHECK TABLE Syntax”.

myisamchk supports the following options, which can be specified on the command line or in the [myisamchk] group of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.19 myisamchk Options

Option Name	Description
--analyze	Analyze the distribution of key values
--backup	Make a backup of the .MYD file as file_name-time.BAK
--block-search	Find the record that a block at the given offset belongs to
--check	Check the table for errors
--check-only-changed	Check only tables that have changed since the last check
--correct-checksum	Correct the checksum information for the table
--data-file-length	Maximum length of the data file (when re-creating data file when it is full)
--debug	Write debugging log
--decode_bits	Decode_bits
--defaults-extra-file	Read named option file in addition to usual option files
--defaults-file	Read only named option file
--defaults-group-suffix	Option group suffix value
--description	Print some descriptive information about the table
--extend-check	Do very thorough table check or repair that tries to recover every possible row from the data file
--fast	Check only tables that haven't been closed properly
--force	Do a repair operation automatically if myisamchk finds any errors in the table
--force	Overwrite old temporary files. For use with the -r or -o option
--ft_max_word_len	Maximum word length for FULLTEXT indexes
--ft_min_word_len	Minimum word length for FULLTEXT indexes
--ft_stopword_file	Use stopwords from this file instead of built-in list
--HELP	Display help message and exit
--help	Display help message and exit
--information	Print informational statistics about the table that is checked
--key_buffer_size	Size of buffer used for index blocks for MyISAM tables
--keys-used	A bit-value that indicates which indexes to update
--max-record-length	Skip rows larger than the given length if myisamchk cannot allocate memory to hold them
--medium-check	Do a check that is faster than an --extend-check operation
--myisam_block_size	Block size to be used for MyISAM index pages
--myisam_sort_buffer_size	The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE
--no-defaults	Read no option files
--parallel-recover	Uses the same technique as -r and -n, but creates all the keys in parallel, using different threads (beta)
--print-defaults	Print default options

Option Name	Description
--quick	Achieve a faster repair by not modifying the data file.
--read_buffer_size	Each thread that does a sequential scan allocates a buffer of this size for each table it scans
--read-only	Do not mark the table as checked
--recover	Do a repair that can fix almost any problem except unique keys that aren't unique
--safe-recover	Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found
--set-auto-increment	Force AUTO_INCREMENT numbering for new records to start at the given value
--set-collation	Specify the collation to use for sorting table indexes
--silent	Silent mode
--sort_buffer_size	The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE
--sort-index	Sort the index tree blocks in high-low order
--sort_key_blocks	sort_key_blocks
--sort-records	Sort records according to a particular index
--sort-recover	Force myisamchk to use sorting to resolve the keys even if the temporary files would be very large
--stats_method	Specifies how MyISAM index statistics collection code should treat NULLs
--tmpdir	Directory to be used for storing temporary files
--unpack	Unpack a table that was packed with myisampack
--update-state	Store information in the .MYI file to indicate when the table was checked and whether the table crashed
--verbose	Verbose mode
--version	Display version information and exit
--write_buffer_size	Write buffer size

4.6.4.1 myisamchk General Options

The options described in this section can be used for any type of table maintenance operation performed by `myisamchk`. The sections following this one describe options that pertain only to specific operations, such as table checking or repairing.

- `--help`, `-?`

  Display a help message and exit. Options are grouped by type of operation.

- `--HELP`, `-H`

  Display a help message and exit. Options are presented in a single list.

- `--debug=`*`debug_options`*, `-#` *`debug_options`*

Write a debugging log. A typical *debug_options* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/myisamchk.trace`.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, `myisamchk` normally reads the `[myisamchk]` group. If the `--defaults-group-suffix=_other` option is given, `myisamchk` also reads the `[myisamchk_other]` group.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--silent`, `-s`

  Silent mode. Write output only when errors occur. You can use `-s` twice (`-ss`) to make `myisamchk` very silent.

- `--verbose`, `-v`

Verbose mode. Print more information about what the program does. This can be used with `-d` and `-e`. Use `-v` multiple times (`-vv`, `-vvv`) for even more output.

- `--version`, `-V`

  Display version information and exit.

- `--wait`, `-w`

  Instead of terminating with an error if the table is locked, wait until the table is unlocked before continuing. If you are running `mysqld` with external locking disabled, the table can be locked only by another `myisamchk` command.

You can also set the following variables by using `--var_name=value` syntax:

Variable	Default Value
`decode_bits`	9
`ft_max_word_len`	version-dependent
`ft_min_word_len`	4
`ft_stopword_file`	built-in list
`key_buffer_size`	523264
`myisam_block_size`	1024
`myisam_sort_key_blocks`	16
`read_buffer_size`	262136
`sort_buffer_size`	2097144
`sort_key_blocks`	16
`stats_method`	nulls_unequal
`write_buffer_size`	262136

The possible `myisamchk` variables and their default values can be examined with `myisamchk --help`:

`myisam_sort_buffer_size` is used when the keys are repaired by sorting keys, which is the normal case when you use `--recover`. `sort_buffer_size` is a deprecated synonym for `myisam_sort_buffer_size`.

`key_buffer_size` is used when you are checking the table with `--extend-check` or when the keys are repaired by inserting keys row by row into the table (like when doing normal inserts). Repairing through the key buffer is used in the following cases:

- You use `--safe-recover`.
- The temporary files needed to sort the keys would be more than twice as big as when creating the key file directly. This is often the case when you have large key values for `CHAR`, `VARCHAR`, or `TEXT` columns, because the sort operation needs to store the complete key values as it proceeds. If you have lots of temporary space and you can force `myisamchk` to repair by sorting, you can use the `--sort-recover` option.

Repairing through the key buffer takes much less disk space than using sorting, but is also much slower.

If you want a faster repair, set the `key_buffer_size` and `myisam_sort_buffer_size` variables to about 25% of your available memory. You can set both variables to large values, because only one of them is used at a time.

myisam_block_size is the size used for index blocks.

stats_method influences how NULL values are treated for index statistics collection when the --analyze option is given. It acts like the myisam_stats_method system variable. For more information, see the description of myisam_stats_method in Section 5.1.8, “Server System Variables”, and Section 8.3.8, “InnoDB and MyISAM Index Statistics Collection”.

ft_min_word_len and ft_max_word_len indicate the minimum and maximum word length for FULLTEXT indexes on MyISAM tables. ft_stopword_file names the stopword file. These need to be set under the following circumstances.

If you use myisamchk to perform an operation that modifies table indexes (such as repair or analyze), the FULLTEXT indexes are rebuilt using the default full-text parameter values for minimum and maximum word length and the stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in MyISAM index files. To avoid the problem if you have modified the minimum or maximum word length or the stopword file in the server, specify the same ft_min_word_len, ft_max_word_len, and ft_stopword_file values to myisamchk that you use for mysqld. For example, if you have set the minimum word length to 3, you can repair a table with myisamchk like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that myisamchk and the server use the same values for full-text parameters, you can place each one in both the [mysqld] and [myisamchk] sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using myisamchk is to use the REPAIR TABLE, ANALYZE TABLE, OPTIMIZE TABLE, or ALTER TABLE. These statements are performed by the server, which knows the proper full-text parameter values to use.

4.6.4.2 myisamchk Check Options

myisamchk supports the following options for table checking operations:

- --check, -c

  Check the table for errors. This is the default operation if you specify no option that selects an operation type explicitly.

- --check-only-changed, -C

  Check only tables that have changed since the last check.

- --extend-check, -e

  Check the table very thoroughly. This is quite slow if the table has many indexes. This option should only be used in extreme cases. Normally, myisamchk or myisamchk --medium-check should be able to determine whether there are any errors in the table.

  If you are using --extend-check and have plenty of memory, setting the key_buffer_size variable to a large value helps the repair operation run faster.

See also the description of this option under table repair options.

For a description of the output format, see Section 4.6.4.5, “Obtaining Table Information with myisamchk”.

- `--fast`, `-F`

  Check only tables that haven't been closed properly.

- `--force`, `-f`

  Do a repair operation automatically if `myisamchk` finds any errors in the table. The repair type is the same as that specified with the `--recover` or `-r` option.

- `--information`, `-i`

  Print informational statistics about the table that is checked.

- `--medium-check`, `-m`

  Do a check that is faster than an `--extend-check` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--read-only`, `-T`

  Do not mark the table as checked. This is useful if you use `myisamchk` to check a table that is in use by some other application that does not use locking, such as `mysqld` when run with external locking disabled.

- `--update-state`, `-U`

  Store information in the `.MYI` file to indicate when the table was checked and whether the table crashed. This should be used to get full benefit of the `--check-only-changed` option, but you shouldn't use this option if the `mysqld` server is using the table and you are running it with external locking disabled.

4.6.4.3 myisamchk Repair Options

`myisamchk` supports the following options for table repair operations (operations performed when an option such as `--recover` or `--safe-recover` is given):

- `--backup`, `-B`

  Make a backup of the `.MYD` file as *`file_name-time`*`.BAK`

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--correct-checksum`

  Correct the checksum information for the table.

- `--data-file-length=`*`len`*, `-D` *`len`*

  The maximum length of the data file (when re-creating data file when it is “full”).

- `--extend-check`, `-e`

Do a repair that tries to recover every possible row from the data file. Normally, this also finds a lot of garbage rows. Do not use this option unless you are desperate.

See also the description of this option under table checking options.

For a description of the output format, see Section 4.6.4.5, “Obtaining Table Information with myisamchk”.

- `--force`, `-f`

  Overwrite old intermediate files (files with names like *`tbl_name`*`.TMD`) instead of aborting.

- `--keys-used=`*`val`*, `-k` *`val`*

  For `myisamchk`, the option value is a bit value that indicates which indexes to update. Each binary bit of the option value corresponds to a table index, where the first index is bit 0. An option value of 0 disables updates to all indexes, which can be used to get faster inserts. Deactivated indexes can be reactivated by using `myisamchk -r`.

- `--no-symlinks`, `-l`

  Do not follow symbolic links. Normally `myisamchk` repairs the table that a symlink points to. This option does not exist as of MySQL 4.0 because versions from 4.0 on do not remove symlinks during repair operations.

- `--max-record-length=`*`len`*

  Skip rows larger than the given length if `myisamchk` cannot allocate memory to hold them.

- `--parallel-recover`, `-p`

  Use the same technique as `-r` and `-n`, but create all the keys in parallel, using different threads. *This is beta-quality code. Use at your own risk!*

- `--quick`, `-q`

  Achieve a faster repair by modifying only the index file, not the data file. You can specify this option twice to force `myisamchk` to modify the original data file in case of duplicate keys.

- `--recover`, `-r`

  Do a repair that can fix almost any problem except unique keys that are not unique (which is an extremely unlikely error with `MyISAM` tables). If you want to recover a table, this is the option to try first. You should try `--safe-recover` only if `myisamchk` reports that the table cannot be recovered using `--recover`. (In the unlikely case that `--recover` fails, the data file remains intact.)

  If you have lots of memory, you should increase the value of `myisam_sort_buffer_size`.

- `--safe-recover`, `-o`

  Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found. This is an order of magnitude slower than `--recover`, but can handle a couple of very unlikely cases that `--recover` cannot. This recovery method also uses much less disk space than `--recover`. Normally, you should repair first using `--recover`, and then with `--safe-recover` only if `--recover` fails.

  If you have lots of memory, you should increase the value of `key_buffer_size`.

- `--set-collation=name`

  Specify the collation to use for sorting table indexes. The character set name is implied by the first part of the collation name.

- `--sort-recover`, `-n`

  Force `myisamchk` to use sorting to resolve the keys even if the temporary files would be very large.

- `--tmpdir=dir_name`, `-t dir_name`

  The path of the directory to be used for storing temporary files. If this is not set, `myisamchk` uses the value of the `TMPDIR` environment variable. `--tmpdir` can be set to a list of directory paths that are used successively in round-robin fashion for creating temporary files. The separator character between directory names is the colon (`:`) on Unix and the semicolon (`;`) on Windows.

- `--unpack`, `-u`

  Unpack a table that was packed with `myisampack`.

4.6.4.4 Other myisamchk Options

`myisamchk` supports the following options for actions other than table checks and repairs:

- `--analyze`, `-a`

  Analyze the distribution of key values. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use. To obtain information about the key distribution, use a `myisamchk --description --verbose tbl_name` command or the `SHOW INDEX FROM tbl_name` statement.

- `--block-search=offset`, `-b offset`

  Find the record that a block at the given offset belongs to.

- `--description`, `-d`

  Print some descriptive information about the table. Specifying the `--verbose` option once or twice produces additional information. See Section 4.6.4.5, “Obtaining Table Information with myisamchk”.

- `--set-auto-increment[=value]`, `-A[value]`

  Force `AUTO_INCREMENT` numbering for new records to start at the given value (or higher, if there are existing records with `AUTO_INCREMENT` values this large). If `value` is not specified, `AUTO_INCREMENT` numbers for new records begin with the largest value currently in the table, plus one.

- `--sort-index`, `-S`

  Sort the index tree blocks in high-low order. This optimizes seeks and makes table scans that use indexes faster.

- `--sort-records=N`, `-R N`

  Sort records according to a particular index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index. (The first time you use this option to sort a table, it may be very slow.) To determine a table's index numbers, use `SHOW INDEX`, which displays a table's indexes in the same order that `myisamchk` sees them. Indexes are numbered beginning with 1.

If keys are not packed (PACK_KEYS=0), they have the same length, so when myisamchk sorts and moves records, it just overwrites record offsets in the index. If keys are packed (PACK_KEYS=1), myisamchk must unpack key blocks first, then re-create indexes and pack the key blocks again. (In this case, re-creating indexes is faster than updating offsets for each index.)

4.6.4.5 Obtaining Table Information with myisamchk

To obtain a description of a MyISAM table or statistics about it, use the commands shown here. The output from these commands is explained later in this section.

- myisamchk -d *tbl_name*

  Runs myisamchk in "describe mode" to produce a description of your table. If you start the MySQL server with external locking disabled, myisamchk may report an error for a table that is updated while it runs. However, because myisamchk does not change the table in describe mode, there is no risk of destroying data.

- myisamchk -dv *tbl_name*

  Adding -v runs myisamchk in verbose mode so that it produces more information about the table. Adding -v a second time produces even more information.

- myisamchk -eis *tbl_name*

  Shows only the most important information from a table. This operation is slow because it must read the entire table.

- myisamchk -eiv *tbl_name*

  This is like -eis, but tells you what is being done.

The *tbl_name* argument can be either the name of a MyISAM table or the name of its index file, as described in Section 4.6.4, "myisamchk — MyISAM Table-Maintenance Utility". Multiple *tbl_name* arguments can be given.

Suppose that a table named person has the following structure. (The MAX_ROWS table option is included so that in the example output from myisamchk shown later, some values are smaller and fit the output format more easily.)

```
CREATE TABLE person
(
  id         INT NOT NULL AUTO_INCREMENT,
  last_name  VARCHAR(20) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  birth      DATE,
  death      DATE,
  PRIMARY KEY (id),
  INDEX (last_name, first_name),
  INDEX (birth)
) MAX_ROWS = 1000000 ENGINE=MYISAM;
```

Suppose also that the table has these data and index file sizes:

```
-rw-rw----  1 mysql  mysql  9347072 Aug 19 11:47 person.MYD
-rw-rw----  1 mysql  mysql  6066176 Aug 19 11:47 person.MYI
```

Example of myisamchk -dvv output:

```
MyISAM file:         person
Record format:       Packed
Character set:       utf8mb4_0900_ai_ci (255)
File-version:        1
Creation time:       2017-03-30 21:21:30
Status:              checked,analyzed,optimized keys,sorted index pages
Auto increment key:              1  Last value:                306688
Data records:               306688  Deleted blocks:                 0
Datafile parts:             306688  Deleted data:                   0
Datafile pointer (bytes):        4  Keyfile pointer (bytes):        3
Datafile length:           9347072  Keyfile length:           6066176
Max datafile length:    4294967294  Max keyfile length:   17179868159
Recordlength:                   54

table description:
Key Start Len Index   Type                     Rec/key         Root  Blocksize
1   2     4   unique  long                           1                    1024
2   6     80  multip. varchar prefix                 0                    1024
    87    80          varchar                        0
3   168   3   multip. uint24 NULL                    0                    1024

Field Start Length Nullpos Nullbit Type
1     1     1
2     2     4                     no zeros
3     6     81                    varchar
4     87    81                    varchar
5     168   3      1       1      no zeros
6     171   3      1       2      no zeros
```

Explanations for the types of information `myisamchk` produces are given here. “Keyfile” refers to the index file. “Record” and “row” are synonymous, as are “field” and “column.”

The initial part of the table description contains these values:

- `MyISAM file`

  Name of the `MyISAM` (index) file.

- `Record format`

  The format used to store table rows. The preceding examples use `Fixed length`. Other possible values are `Compressed` and `Packed`. (`Packed` corresponds to what `SHOW TABLE STATUS` reports as `Dynamic`.)

- `Chararacter set`

  The table default character set.

- `File-version`

  Version of `MyISAM` format. Always 1.

- `Creation time`

  When the data file was created.

- `Recover time`

  When the index/data file was last reconstructed.

- `Status`

Table status flags. Possible values are `crashed`, `open`, `changed`, `analyzed`, `optimized keys`, and `sorted index pages`.

- `Auto increment key`, `Last value`

  The key number associated the table's `AUTO_INCREMENT` column, and the most recently generated value for this column. These fields do not appear if there is no such column.

- `Data records`

  The number of rows in the table.

- `Deleted blocks`

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- `Datafile parts`

  For dynamic-row format, this indicates how many data blocks there are. For an optimized table without fragmented rows, this is the same as `Data records`.

- `Deleted data`

  How many bytes of unreclaimed deleted data there are. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- `Datafile pointer`

  The size of the data file pointer, in bytes. It is usually 2, 3, 4, or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from MySQL yet. For fixed tables, this is a row address. For dynamic tables, this is a byte address.

- `Keyfile pointer`

  The size of the index file pointer, in bytes. It is usually 1, 2, or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by MySQL. It is always a block address.

- `Max datafile length`

  How long the table data file can become, in bytes.

- `Max keyfile length`

  How long the table index file can become, in bytes.

- `Recordlength`

  How much space each row takes, in bytes.

The `table description` part of the output includes a list of all keys in the table. For each key, `myisamchk` displays some low-level information:

- `Key`

  This key's number. This value is shown only for the first column of the key. If this value is missing, the line corresponds to the second or later column of a multiple-column key. For the table shown in the example, there are two `table description` lines for the second index. This indicates that it is a multiple-part index with two parts.

- `Start`

  Where in the row this portion of the index starts.

- `Len`

  How long this portion of the index is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column. The total length of a multiple-part key is the sum of the `Len` values for all key parts.

- `Index`

  Whether a key value can exist multiple times in the index. Possible values are `unique` or `multip.` (multiple).

- `Type`

  What data type this portion of the index has. This is a `MyISAM` data type with the possible values `packed`, `stripped`, or `empty`.

- `Root`

  Address of the root index block.

- `Blocksize`

  The size of each index block. By default this is 1024, but the value may be changed at compile time when MySQL is built from source.

- `Rec/key`

  This is a statistical value used by the optimizer. It tells how many rows there are per value for this index. A unique index always has a value of 1. This may be updated after a table is loaded (or greatly changed) with `myisamchk -a`. If this is not updated at all, a default value of 30 is given.

The last part of the output provides information about each column:

- `Field`

  The column number.

- `Start`

  The byte position of the column within table rows.

- `Length`

  The length of the column in bytes.

- `Nullpos`, `Nullbit`

  For columns that can be `NULL`, `MyISAM` stores `NULL` values as a flag in a byte. Depending on how many nullable columns there are, there can be one or more bytes used for this purpose. The `Nullpos` and `Nullbit` values, if nonempty, indicate which byte and bit contains that flag indicating whether the column is `NULL`.

  The position and number of bytes used to store `NULL` flags is shown in the line for field 1. This is why there are six `Field` lines for the `person` table even though it has only five columns.

- `Type`

  The data type. The value may contain any of the following descriptors:

  - `constant`

    All rows have the same value.

  - `no endspace`

    Do not store endspace.

  - `no endspace, not_always`

    Do not store endspace and do not do endspace compression for all values.

  - `no endspace, no empty`

    Do not store endspace. Do not store empty values.

  - `table-lookup`

    The column was converted to an `ENUM`.

  - `zerofill(N)`

    The most significant *`N`* bytes in the value are always 0 and are not stored.

  - `no zeros`

    Do not store zeros.

  - `always zero`

    Zero values are stored using one bit.

- `Huff tree`

  The number of the Huffman tree associated with the column.

- `Bits`

  The number of bits used in the Huffman tree.

The `Huff tree` and `Bits` fields are displayed if the table has been compressed with `myisampack`. See Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”, for an example of this information.

Example of `myisamchk -eiv` output:

```
Checking MyISAM file: person
Data records:  306688   Deleted blocks:       0
- check file-size
- check record delete-chain
No recordlinks
- check key delete-chain
block_size 1024:
- check index reference
- check data record references index: 1
```

```
Key:  1:  Keyblocks used:  98%  Packed:    0%  Max levels:  3
- check data record references index: 2
Key:  2:  Keyblocks used:  99%  Packed:   97%  Max levels:  3
- check data record references index: 3
Key:  3:  Keyblocks used:  98%  Packed:  -14%  Max levels:  3
Total:    Keyblocks used:  98%  Packed:   89%

- check records and index references
*** LOTS OF ROW NUMBERS DELETED ***

Records:            306688  M.recordlength:       25  Packed:            83%
Recordspace used:       97% Empty space:           2% Blocks/Record:   1.00
Record blocks:      306688  Delete blocks:         0
Record data:       7934464  Deleted data:          0
Lost space:         256512  Linkdata:        1156096

User time 43.08, System time 1.68
Maximum resident set size 0, Integral resident set size 0
Non-physical pagefaults 0, Physical pagefaults 0, Swaps 0
Blocks in 0 out 7, Messages in 0 out 0, Signals 0
Voluntary context switches 0, Involuntary context switches 0
Maximum memory usage: 1046926 bytes (1023k)
```

`myisamchk -eiv` output includes the following information:

- `Data records`

  The number of rows in the table.

- `Deleted blocks`

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- `Key`

  The key number.

- `Keyblocks used`

  What percentage of the keyblocks are used. When a table has just been reorganized with `myisamchk`, the values are very high (very near theoretical maximum).

- `Packed`

  MySQL tries to pack key values that have a common suffix. This can only be used for indexes on `CHAR` and `VARCHAR` columns. For long indexed strings that have similar leftmost parts, this can significantly reduce the space used. In the preceding example, the second key is 40 bytes long and a 97% reduction in space is achieved.

- `Max levels`

  How deep the B-tree for this key is. Large tables with long key values get high values.

- `Records`

  How many rows are in the table.

- `M.recordlength`

  The average row length. This is the exact row length for tables with fixed-length rows, because all rows have the same length.

- `Packed`

  MySQL strips spaces from the end of strings. The `Packed` value indicates the percentage of savings achieved by doing this.

- `Recordspace used`

  What percentage of the data file is used.

- `Empty space`

  What percentage of the data file is unused.

- `Blocks/Record`

  Average number of blocks per row (that is, how many links a fragmented row is composed of). This is always 1.0 for fixed-format tables. This value should stay as close to 1.0 as possible. If it gets too large, you can reorganize the table. See Section 7.6.4, “MyISAM Table Optimization”.

- `Recordblocks`

  How many blocks (links) are used. For fixed-format tables, this is the same as the number of rows.

- `Deleteblocks`

  How many blocks (links) are deleted.

- `Recorddata`

  How many bytes in the data file are used.

- `Deleted data`

  How many bytes in the data file are deleted (unused).

- `Lost space`

  If a row is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

- `Linkdata`

  When the dynamic table format is used, row fragments are linked with pointers (4 to 7 bytes each). `Linkdata` is the sum of the amount of storage used by all such pointers.

4.6.4.6 myisamchk Memory Usage

Memory allocation is important when you run `myisamchk`. `myisamchk` uses no more memory than its memory-related variables are set to. If you are going to use `myisamchk` on very large tables, you should first decide how much memory you want it to use. The default is to use only about 3MB to perform repairs. By using larger values, you can get `myisamchk` to operate faster. For example, if you have more than 512MB RAM available, you could use options such as these (in addition to any other options you might specify):

```
shell> myisamchk --myisam_sort_buffer_size=256M \
           --key_buffer_size=512M \
           --read_buffer_size=64M \
           --write_buffer_size=64M ...
```

Using `--myisam_sort_buffer_size=16M` is probably enough for most cases.

Be aware that myisamchk uses temporary files in TMPDIR. If TMPDIR points to a memory file system, out of memory errors can easily occur. If this happens, run myisamchk with the --tmpdir=*dir_name* option to specify a directory located on a file system that has more space.

When performing repair operations, myisamchk also needs a lot of disk space:

- Twice the size of the data file (the original file and a copy). This space is not needed if you do a repair with --quick; in this case, only the index file is re-created. *This space must be available on the same file system as the original data file*, as the copy is created in the same directory as the original.
- Space for the new index file that replaces the old one. The old index file is truncated at the start of the repair operation, so you usually ignore this space. This space must be available on the same file system as the original data file.
- When using --recover or --sort-recover (but not when using --safe-recover), you need space on disk for sorting. This space is allocated in the temporary directory (specified by TMPDIR or --tmpdir=*dir_name*). The following formula yields the amount of space required:

  ```
  (largest_key + row_pointer_length) * number_of_rows * 2
  ```

  You can check the length of the keys and the *row_pointer_length* with myisamchk -dv *tbl_name* (see Section 4.6.4.5, “Obtaining Table Information with myisamchk”). The *row_pointer_length* and *number_of_rows* values are the Datafile pointer and Data records values in the table description. To determine the *largest_key* value, check the Key lines in the table description. The Len column indicates the number of bytes for each key part. For a multiple-column index, the key size is the sum of the Len values for all key parts.

If you have a problem with disk space during repair, you can try --safe-recover instead of --recover.

4.6.5 myisamlog — Display MyISAM Log File Contents

myisamlog processes the contents of a MyISAM log file. To create such a file, start the server with a --log-isam=log_file option.

Invoke myisamlog like this:

```
shell> myisamlog [options] [file_name [tbl_name] ...]
```

The default operation is update (-u). If a recovery is done (-r), all writes and possibly updates and deletes are done and errors are only counted. The default log file name is myisam.log if no *log_file* argument is given. If tables are named on the command line, only those tables are updated.

myisamlog supports the following options:

- -?, -I

  Display a help message and exit.

- -c *N*

  Execute only *N* commands.

- -f *N*

  Specify the maximum number of open files.

- -F *filepath/*

Specify the file path with a trailing slash.

- `-i`

  Display extra information before exiting.

- `-o offset`

  Specify the starting offset.

- `-p N`

  Remove `N` components from path.

- `-r`

  Perform a recovery operation.

- `-R record_pos_file record_pos`

  Specify record position file and record position.

- `-u`

  Perform an update operation.

- `-v`

  Verbose mode. Print more output about what the program does. This option can be given multiple times to produce more and more output.

- `-w write_file`

  Specify the write file.

- `-V`

  Display version information.

4.6.6 myisampack — Generate Compressed, Read-Only MyISAM Tables

The `myisampack` utility compresses `MyISAM` tables. `myisampack` works by compressing each column in the table separately. Usually, `myisampack` packs the data file 40% to 70%.

When the table is used later, the server reads into memory the information needed to decompress columns. This results in much better performance when accessing individual rows, because you only have to uncompress exactly one row.

MySQL uses `mmap()` when possible to perform memory mapping on compressed tables. If `mmap()` does not work, MySQL falls back to normal read/write file operations.

Please note the following:

- If the `mysqld` server was invoked with external locking disabled, it is not a good idea to invoke `myisampack` if the table might be updated by the server during the packing process. It is safest to compress tables with the server stopped.
- After packing a table, it becomes read only. This is generally intended (such as when accessing packed tables on a CD).

- myisampack does not support partitioned tables.

Invoke myisampack like this:

```
shell> myisampack [options] file_name ...
```

Each file name argument should be the name of an index (.MYI) file. If you are not in the database directory, you should specify the path name to the file. It is permissible to omit the .MYI extension.

After you compress a table with myisampack, use myisamchk -rq to rebuild its indexes. Section 4.6.4, “myisamchk — MyISAM Table-Maintenance Utility”.

myisampack supports the following options. It also reads option files and supports the options for processing them described at Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --help, -?

  Display a help message and exit.

- --backup, -b

  Make a backup of each table's data file using the name *tbl_name*.OLD.

- --character-sets-dir=*dir_name*

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- --debug[=*debug_options*], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default is d:t:o.

- --force, -f

  Produce a packed table even if it becomes larger than the original or if the intermediate file from an earlier invocation of myisampack exists. (myisampack creates an intermediate file named *tbl_name*.TMD in the database directory while it compresses the table. If you kill myisampack, the .TMD file might not be deleted.) Normally, myisampack exits with an error if it finds that *tbl_name*.TMD exists. With --force, myisampack packs the table anyway.

- --join=*big_tbl_name*, -j *big_tbl_name*

  Join all tables named on the command line into a single packed table *big_tbl_name*. All tables that are to be combined *must* have identical structure (same column names and types, same indexes, and so forth).

  big_tbl_name must not exist prior to the join operation. All source tables named on the command line to be merged into *big_tbl_name* must exist. The source tables are read for the join operation but not modified.

- --silent, -s

  Silent mode. Write output only when errors occur.

- --test, -t

  Do not actually pack the table, just test packing it.

- --tmpdir=*dir_name*, -T *dir_name*

Use the named directory as the location where myisampack creates temporary files.

- --verbose, -v

  Verbose mode. Write information about the progress of the packing operation and its result.

- --version, -V

  Display version information and exit.

- --wait, -w

  Wait and retry if the table is in use. If the mysqld server was invoked with external locking disabled, it is not a good idea to invoke myisampack if the table might be updated by the server during the packing process.

The following sequence of commands illustrates a typical table compression session:

```
shell> ls -l station.*
-rw-rw-r--   1 jones    my         994128 Apr 17 19:00 station.MYD
-rw-rw-r--   1 jones    my          53248 Apr 17 19:00 station.MYI

shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-02-02  3:06:43
Data records:              1192  Deleted blocks:              0
Datafile parts:            1192  Deleted data:                0
Datafile pointer (bytes):     2  Keyfile pointer (bytes):     2
Max datafile length:   54657023  Max keyfile length:   33554431
Recordlength:               834
Record format: Fixed length

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long        1024       1024          1
2   32    30  multip. text                10240       1024          1

Field Start Length Type
1     1     1
2     2     4
3     6     4
4     10    1
5     11    20
6     31    1
7     32    30
8     62    35
9     97    35
10    132   35
11    167   4
12    171   16
13    187   35
14    222   4
15    226   16
16    242   20
17    262   20
18    282   20
19    302   30
20    332   4
21    336   4
```

```
22    340   1
23    341   8
24    349   8
25    357   8
26    365   2
27    367   2
28    369   4
29    373   4
30    377   1
31    378   2
32    380   8
33    388   4
34    392   4
35    396   4
36    400   4
37    404   1
38    405   4
39    409   4
40    413   4
41    417   4
42    421   4
43    425   4
44    429   20
45    449   30
46    479   1
47    480   1
48    481   79
49    560   79
50    639   79
51    718   79
52    797   8
53    805   1
54    806   1
55    807   20
56    827   4
57    831   4

shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

normal:     20  empty-space:   16  empty-zero:     12  empty-fill:  11
pre-space:   0  end-space:     12  table-lookups:   5  zero:         7
Original trees:  57  After join: 17
- Compressing file
87.14%
Remember to run myisamchk -rq on compressed tables

shell> myisamchk -rq station
- check record delete-chain
- recovering (with sort) MyISAM-table 'station'
Data records: 1192
- Fixing index 1
- Fixing index 2

shell> mysqladmin -uroot flush-tables

shell> ls -l station.*
-rw-rw-r--   1 jones    my         127874 Apr 17 19:00 station.MYD
-rw-rw-r--   1 jones    my          55296 Apr 17 19:04 station.MYI

shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-04-17 19:04:26
```

```
Data records:               1192  Deleted blocks:              0
Datafile parts:             1192  Deleted data:                0
Datafile pointer (bytes):      3  Keyfile pointer (bytes):     1
Max datafile length:    16777215  Max keyfile length:     131071
Recordlength:                834
Record format: Compressed

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long       10240       1024          1
2   32    30  multip. text                54272       1024          1

Field Start Length Type                          Huff tree  Bits
1     1     1      constant                            1     0
2     2     4      zerofill(1)                         2     9
3     6     4      no zeros, zerofill(1)               2     9
4     10    1                                          3     9
5     11    20     table-lookup                        4     0
6     31    1                                          3     9
7     32    30     no endspace, not_always             5     9
8     62    35     no endspace, not_always, no empty   6     9
9     97    35     no empty                            7     9
10    132   35     no endspace, not_always, no empty   6     9
11    167   4      zerofill(1)                         2     9
12    171   16     no endspace, not_always, no empty   5     9
13    187   35     no endspace, not_always, no empty   6     9
14    222   4      zerofill(1)                         2     9
15    226   16     no endspace, not_always, no empty   5     9
16    242   20     no endspace, not_always             8     9
17    262   20     no endspace, no empty               8     9
18    282   20     no endspace, no empty               5     9
19    302   30     no endspace, no empty               6     9
20    332   4      always zero                         2     9
21    336   4      always zero                         2     9
22    340   1                                          3     9
23    341   8      table-lookup                        9     0
24    349   8      table-lookup                       10     0
25    357   8      always zero                         2     9
26    365   2                                          2     9
27    367   2      no zeros, zerofill(1)               2     9
28    369   4      no zeros, zerofill(1)               2     9
29    373   4      table-lookup                       11     0
30    377   1                                          3     9
31    378   2      no zeros, zerofill(1)               2     9
32    380   8      no zeros                            2     9
33    388   4      always zero                         2     9
34    392   4      table-lookup                       12     0
35    396   4      no zeros, zerofill(1)              13     9
36    400   4      no zeros, zerofill(1)               2     9
37    404   1                                          2     9
38    405   4      no zeros                            2     9
39    409   4      always zero                         2     9
40    413   4      no zeros                            2     9
41    417   4      always zero                         2     9
42    421   4      no zeros                            2     9
43    425   4      always zero                         2     9
44    429   20     no empty                            3     9
45    449   30     no empty                            3     9
46    479   1                                         14     4
47    480   1                                         14     4
48    481   79     no endspace, no empty              15     9
49    560   79     no empty                            2     9
50    639   79     no empty                            2     9
51    718   79     no endspace                        16     9
52    797   8      no empty                            2     9
53    805   1                                         17     1
54    806   1                                          3     9
```

```
55    807   20     no empty                              3     9
56    827   4      no zeros, zerofill(2)                 2     9
57    831   4      no zeros, zerofill(1)                 2     9
```

myisampack displays the following kinds of information:

- normal

  The number of columns for which no extra packing is used.

- empty-space

  The number of columns containing values that are only spaces. These occupy one bit.

- empty-zero

  The number of columns containing values that are only binary zeros. These occupy one bit.

- empty-fill

  The number of integer columns that do not occupy the full byte range of their type. These are changed to a smaller type. For example, a BIGINT column (eight bytes) can be stored as a TINYINT column (one byte) if all its values are in the range from -128 to 127.

- pre-space

  The number of decimal columns that are stored with leading spaces. In this case, each value contains a count for the number of leading spaces.

- end-space

  The number of columns that have a lot of trailing spaces. In this case, each value contains a count for the number of trailing spaces.

- table-lookup

  The column had only a small number of different values, which were converted to an ENUM before Huffman compression.

- zero

  The number of columns for which all values are zero.

- Original trees

  The initial number of Huffman trees.

- After join

  The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, the Field lines displayed by myisamchk -dvv include additional information about each column:

- Type

  The data type. The value may contain any of the following descriptors:

  - constant

All rows have the same value.

- `no endspace`

  Do not store endspace.

- `no endspace, not_always`

  Do not store endspace and do not do endspace compression for all values.

- `no endspace, no empty`

  Do not store endspace. Do not store empty values.

- `table-lookup`

  The column was converted to an `ENUM`.

- `zerofill(N)`

  The most significant *`N`* bytes in the value are always 0 and are not stored.

- `no zeros`

  Do not store zeros.

- `always zero`

  Zero values are stored using one bit.

- `Huff tree`

  The number of the Huffman tree associated with the column.

- `Bits`

  The number of bits used in the Huffman tree.

After you run `myisampack`, use `myisamchk` to re-create any indexes. At this time, you can also sort the index blocks and create statistics needed for the MySQL optimizer to work more efficiently:

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

After you have installed the packed table into the MySQL database directory, you should execute `mysqladmin flush-tables` to force `mysqld` to start using the new table.

To unpack a packed table, use the `--unpack` option to `myisamchk`.

4.6.7 `mysql_config_editor` — MySQL Configuration Utility

The `mysql_config_editor` utility enables you to store authentication credentials in an obfuscated login path file named `.mylogin.cnf`. The file location is the `%APPDATA%\MySQL` directory on Windows and the current user's home directory on non-Windows systems. The file can be read later by MySQL client programs to obtain authentication credentials for connecting to MySQL Server.

The unobfuscated format of the `.mylogin.cnf` login path file consists of option groups, similar to other option files. Each option group in `.mylogin.cnf` is called a “login path,” which is a group that permits only

certain options: host, user, password, port and socket. Think of a login path option group as a set of options that specify which MySQL server to connect to and which account to authenticate as. Here is an unobfuscated example:

```
[client]
user = mydefaultname
password = mydefaultpass
host = 127.0.0.1
[mypath]
user = myothername
password = myotherpass
host = localhost
```

When you invoke a client program to connect to the server, the client uses .mylogin.cnf in conjunction with other option files. Its precedence is higher than other option files, but less than options specified explicitly on the client command line. For information about the order in which option files are used, see Section 4.2.2.2, "Using Option Files".

To specify an alternate login path file name, set the MYSQL_TEST_LOGIN_FILE environment variable. This variable is recognized by mysql_config_editor, by standard MySQL clients (mysql, mysqladmin, and so forth), and by the mysql-test-run.pl testing utility.

Programs use groups in the login path file as follows:

- mysql_config_editor operates on the client login path by default if you specify no --login-path=*name* option to indicate explicitly which login path to use.
- Without a --login-path option, client programs read the same option groups from the login path file that they read from other option files. Consider this command:

  ```
  shell> mysql
  ```

  By default, the mysql client reads the [client] and [mysql] groups from other option files, so it reads them from the login path file as well.
- With a --login-path option, client programs additionally read the named login path from the login path file. The option groups read from other option files remain the same. Consider this command:

  ```
  shell> mysql --login-path=mypath
  ```

  The mysql client reads [client] and [mysql] from other option files, and [client], [mysql], and [mypath] from the login path file.
- Client programs read the login path file even when the --no-defaults option is used. This permits passwords to be specified in a safer way than on the command line even if --no-defaults is present.

mysql_config_editor obfuscates the .mylogin.cnf file so it cannot be read as cleartext, and its contents when unobfuscated by client programs are used only in memory. In this way, passwords can be stored in a file in non-cleartext format and used later without ever needing to be exposed on the command line or in an environment variable. mysql_config_editor provides a print command for displaying the login path file contents, but even in this case, password values are masked so as never to appear in a way that other users can see them.

The obfuscation used by mysql_config_editor prevents passwords from appearing in .mylogin.cnf as cleartext and provides a measure of security by preventing inadvertent password exposure. For example, if you display a regular unobfuscated my.cnf option file on the screen, any passwords it contains

are visible for anyone to see. With .mylogin.cnf, that is not true. But the obfuscation used will not deter a determined attacker and you should not consider it unbreakable. A user who can gain system administration privileges on your machine to access your files could unobfuscate the .mylogin.cnf file with some effort.

The login path file must be readable and writable to the current user, and inaccessible to other users. Otherwise, mysql_config_editor ignores it, and client programs do not use it, either.

Invoke mysql_config_editor like this:

```
shell> mysql_config_editor [program_options] command [command_options]
```

If the login path file does not exist, mysql_config_editor creates it.

Command arguments are given as follows:

- *program_options* consists of general mysql_config_editor options.
- command indicates what action to perform on the .mylogin.cnf login path file. For example, set writes a login path to the file, remove removes a login path, and print displays login path contents.
- *command_options* indicates any additional options specific to the command, such as the login path name and the values to use in the login path.

The position of the command name within the set of program arguments is significant. For example, these command lines have the same arguments, but produce different results:

```
shell> mysql_config_editor --help set
shell> mysql_config_editor set --help
```

The first command line displays a general mysql_config_editor help message, and ignores the set command. The second command line displays a help message specific to the set command.

Suppose that you want to establish a client login path that defines your default connection parameters, and an additional login path named remote for connecting to the MySQL server the host remote.example.com. You want to log in as follows:

- By default, to the local server with a user name and password of localuser and localpass
- To the remote server with a user name and password of remoteuser and remotepass

To set up the login paths in the .mylogin.cnf file, use the following set commands. Enter each command on a single line, and enter the appropriate passwords when prompted:

```
shell> mysql_config_editor set --login-path=client
         --host=localhost --user=localuser --password
Enter password: enter password "localpass" here
shell> mysql_config_editor set --login-path=remote
         --host=remote.example.com --user=remoteuser --password
Enter password: enter password "remotepass" here
```

mysql_config_editor uses the client login path by default, so the --login-path=client option can be omitted from the first command without changing its effect.

To see what mysql_config_editor writes to the .mylogin.cnf file, use the print command:

```
shell> mysql_config_editor print --all
```

```
[client]
user = localuser
password = *****
host = localhost
[remote]
user = remoteuser
password = *****
host = remote.example.com
```

The `print` command displays each login path as a set of lines beginning with a group header indicating the login path name in square brackets, followed by the option values for the login path. Password values are masked and do not appear as cleartext.

If you do not specify `--all` to display all login paths or `--login-path=`*`name`* to display a named login path, the `print` command displays the `client` login path by default, if there is one.

As shown by the preceding example, the login path file can contain multiple login paths. In this way, `mysql_config_editor` makes it easy to set up multiple “personalities” for connecting to different MySQL servers, or for connecting to a given server using different accounts. Any of these can be selected by name later using the `--login-path` option when you invoke a client program. For example, to connect to the remote server, use this command:

```
shell> mysql --login-path=remote
```

Here, `mysql` reads the `[client]` and `[mysql]` option groups from other option files, and the `[client]`, `[mysql]`, and `[remote]` groups from the login path file.

To connect to the local server, use this command:

```
shell> mysql --login-path=client
```

Because `mysql` reads the `client` and `mysql` login paths by default, the `--login-path` option does not add anything in this case. That command is equivalent to this one:

```
shell> mysql
```

Options read from the login path file take precedence over options read from other option files. Options read from login path groups appearing later in the login path file take precedence over options read from groups appearing earlier in the file.

`mysql_config_editor` adds login paths to the login path file in the order you create them, so you should create more general login paths first and more specific paths later. If you need to move a login path within the file, you can remove it, then recreate it to add it to the end. For example, a `client` login path is more general because it is read by all client programs, whereas a `mysqldump` login path is read only by `mysqldump`. Options specified later override options specified earlier, so putting the login paths in the order `client`, `mysqldump` enables `mysqldump`-specific options to override `client` options.

When you use the `set` command with `mysql_config_editor` to create a login path, you need not specify all possible option values (host name, user name, password, port, socket). Only those values given are written to the path. Any missing values required later can be specified when you invoke a client path to connect to the MySQL server, either in other option files or on the command line. Any options specified on the command line override those specified in the login path file or other option files. For example, if the credentials in the `remote` login path also apply for the host `remote2.example.com`, connect to the server on that host like this:

```
shell> mysql --login-path=remote --host=remote2.example.com
```

mysql_config_editor General Options

mysql_config_editor supports the following general options, which may be used preceding any command named on the command line. For descriptions of command-specific options, see mysql_config_editor Commands and Command-Specific Options.

Table 4.20 mysql_config_editor General Options

Option Name	Description
--debug	Write debugging log
--help	Display help message and exit
--verbose	Verbose mode
--version	Display version information and exit

- `--help`, `-?`

  Display a general help message and exit.

  To see a command-specific help message, invoke mysql_config_editor as follows, where *command* is a command other than help:

```
shell> mysql_config_editor command --help
```

- `--debug[=debug_options]`, `-# debug_options`

  Write a debugging log. A typical *debug_options* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysql_config_editor.trace`.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does. This option may be helpful in diagnosing problems if an operation does not have the effect you expect.

- `--version`, `-V`

  Display version information and exit.

mysql_config_editor Commands and Command-Specific Options

This section describes the permitted mysql_config_editor commands, and, for each one, the command-specific options permitted following the command name on the command line.

In addition, mysql_config_editor supports general options that can be used preceding any command. For descriptions of these options, see mysql_config_editor General Options.

mysql_config_editor supports these commands:

- help

  Display a general help message and exit. This command takes no following options.

  To see a command-specific help message, invoke mysql_config_editor as follows, where *command* is a command other than help:

```
shell> mysql_config_editor command --help
```

- `print [options]`

  Print the contents of the login path file in unobfuscated form, with the exception that passwords are displayed as `*****`.

  The default login path name is `client` if no login path is named. If both `--all` and `--login-path` are given, `--all` takes precedence.

  The `print` command permits these options following the command name:

  - `--help`, `-?`

    Display a help message for the `print` command and exit.

    To see a general help message, use `mysql_config_editor --help`.

  - `--all`

    Print the contents of all login paths in the login path file.

  - `--login-path=name`, `-G name`

    Print the contents of the named login path.

- `remove [options]`

  Remove a login path from the login path file, or modify a login path by removing options from it.

  This command removes from the login path only such options as are specified with the `--host`, `--password`, `--port`, `--socket`, and `--user` options. If none of those options are given, `remove` removes the entire login path. For example, this command removes only the `user` option from the `mypath` login path rather than the entire `mypath` login path:

```
shell> mysql_config_editor remove --login-path=mypath --user
```

  This command removes the entire `mypath` login path:

```
shell> mysql_config_editor remove --login-path=mypath
```

  The `remove` command permits these options following the command name:

  - `--help`, `-?`

    Display a help message for the `remove` command and exit.

    To see a general help message, use `mysql_config_editor --help`.

  - `--host`, `-h`

    Remove the host name from the login path.

  - `--login-path=name`, `-G name`

    The login path to remove or modify. The default login path name is `client` if this option is not given.

  - `--password`, `-p`

Remove the password from the login path.

- `--port`, `-P`

  Remove the TCP/IP port number from the login path.

- `--socket`, `-S`

  Remove the Unix socket file name from the login path.

- `--user`, `-u`

  Remove the user name from the login path.

- `--warn`, `-w`

  Warn and prompt the user for confirmation if the command attempts to remove the default login path (`client`) and `--login-path=client` was not specified. This option is enabled by default; use `--skip-warn` to disable it.

- `reset [options]`

  Empty the contents of the login path file.

  The `reset` command permits these options following the command name:

  - `--help`, `-?`

    Display a help message for the `reset` command and exit.

    To see a general help message, use `mysql_config_editor --help`.

- `set [options]`

  Write a login path to the login path file.

  This command writes to the login path only such options as are specified with the `--host`, `--password`, `--port`, `--socket`, and `--user` options. If none of those options are given, `mysql_config_editor` writes the login path as an empty group.

  The `set` command permits these options following the command name:

  - `--help`, `-?`

    Display a help message for the `set` command and exit.

    To see a general help message, use `mysql_config_editor --help`.

  - `--host=host_name`, `-h host_name`

    The host name to write to the login path.

  - `--login-path=name`, `-G name`

    The login path to create. The default login path name is `client` if this option is not given.

  - `--password`, `-p`

Prompt for a password to write to the login path. After mysql_config_editor displays the prompt, type the password and press Enter. To prevent other users from seeing the password, mysql_config_editor does not echo it.

To specify an empty password, press Enter at the password prompt. The resulting login path written to the login path file will include a line like this:

```
password =
```

- --port=*port_num*, -P *port_num*

  The TCP/IP port number to write to the login path.

- --socket=*file_name*, -S *file_name*

  The Unix socket file name to write to the login path.

- --user=*user_name*, -u *user_name*

  The user name to write to the login path.

- --warn, -w

  Warn and prompt the user for confirmation if the command attempts to overwrite an existing login path. This option is enabled by default; use --skip-warn to disable it.

4.6.8 mysqlbinlog — Utility for Processing Binary Log Files

The server's binary log consists of files containing “events” that describe modifications to database contents. The server writes these files in binary format. To display their contents in text format, use the mysqlbinlog utility. You can also use mysqlbinlog to display the contents of relay log files written by a slave server in a replication setup because relay logs have the same format as binary logs. The binary log and relay log are discussed further in Section 5.4.4, “The Binary Log”, and Section 17.2.4, “Replication Relay and Status Logs”.

Invoke mysqlbinlog like this:

```
shell> mysqlbinlog [options] log_file ...
```

For example, to display the contents of the binary log file named binlog.000003, use this command:

```
shell> mysqlbinlog binlog.0000003
```

The output includes events contained in binlog.000003. For statement-based logging, event information includes the SQL statement, the ID of the server on which it was executed, the timestamp when the statement was executed, how much time it took, and so forth. For row-based logging, the event indicates a row change rather than an SQL statement. See Section 17.2.1, “Replication Formats”, for information about logging modes.

Events are preceded by header comments that provide additional information. For example:

```
# at 141
#100309  9:28:36 server id 123  end_log_pos 245
  Query thread_id=3350  exec_time=11  error_code=0
```

In the first line, the number following at indicates the file offset, or starting position, of the event in the binary log file.

The second line starts with a date and time indicating when the statement started on the server where the event originated. For replication, this timestamp is propagated to slave servers. server id is the server_id value of the server where the event originated. end_log_pos indicates where the next event starts (that is, it is the end position of the current event + 1). thread_id indicates which thread executed the event. exec_time is the time spent executing the event, on a master server. On a slave, it is the difference of the end execution time on the slave minus the beginning execution time on the master. The difference serves as an indicator of how much replication lags behind the master. error_code indicates the result from executing the event. Zero means that no error occurred.

Note

When using event groups, the file offsets of events may be grouped together and the comments of events may be grouped together. Do not mistake these grouped events for blank file offsets.

The output from mysqlbinlog can be re-executed (for example, by using it as input to mysql) to redo the statements in the log. This is useful for recovery operations after a server crash. For other usage examples, see the discussion later in this section and in Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”. To execute the internal-use BINLOG statements used by mysqlbinlog, the user requires the BINLOG_ADMIN or SUPER privilege, or the REPLICATION_APPLIER privilege plus the appropriate privileges to execute each log event.

You can use mysqlbinlog to read binary log files directly and apply them to the local MySQL server. You can also read binary logs from a remote server by using the --read-from-remote-server option. To read remote binary logs, the connection parameter options can be given to indicate how to connect to the server. These options are --host, --password, --port, --protocol, --socket, and --user.

When binary log files have been encrypted, which can be done from MySQL 8.0.14 onwards, mysqlbinlog cannot read them directly, but can read them from the server using the --read-from-remote-server option. Binary log files are encrypted when the server's binlog_encryption system variable is set to ON. The SHOW BINARY LOGS statement shows whether a particular binary log file is encrypted or unencrypted. Encrypted and unencrypted binary log files can also be distinguished using the magic number at the start of the file header for encrypted log files (0xFD62696E), which differs from that used for unencrypted log files (0xFE62696E). Note that from MySQL 8.0.14, mysqlbinlog returns a suitable error if you attempt to read an encrypted binary log file directly, but older versions of mysqlbinlog do not recognise the file as a binary log file at all. For more information on binary log encryption, see Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”.

When running mysqlbinlog against a large binary log, be careful that the filesystem has enough space for the resulting files. To configure the directory that mysqlbinlog uses for temporary files, use the TMPDIR environment variable.

mysqlbinlog sets the value of pseudo_slave_mode to true before executing any SQL statements. This system variable affects the handling of XA transactions, the original_commit_timestamp replication delay timestamp and the original_server_version system variable, and unsupported SQL modes.

mysqlbinlog supports the following options, which can be specified on the command line or in the [mysqlbinlog] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.21 mysqlbinlog Options

Option Name	Description	Introduced	Deprecated	Removed
--base64-output	Print binary log entries using base-64 encoding			

Option Name	Description	Introduced	Deprecated	Removed
--bind-address	Use specified network interface to connect to MySQL Server			
--binlog-row-event-max-size	Binary log max event size			
--character-sets-dir	Directory where character sets are installed			
--compress	Compress all information sent between client and server	8.0.17	8.0.18	
--compression-algorithms	Permitted compression algorithms for connections to server	8.0.18		
--connection-server-id	Used for testing and debugging. See text for applicable default values and other particulars.			
--database	List entries for just this database			
--debug	Write debugging log			
--debug-check	Print debugging information when program exits			
--debug-info	Print debugging information, memory, and CPU statistics when program exits			
--default-auth	Authentication plugin to use			
--defaults-extra-file	Read named option file in addition to usual option files			
--defaults-file	Read only named option file			
--defaults-group-suffix	Option group suffix value			
--disable-log-bin	Disable binary logging			
--exclude-gtids	Do not show any of the groups in the GTID set provided			
--force-if-open	Read binary log files even if open or not closed properly			
--force-read	If mysqlbinlog reads a binary log event that it does not recognize, it prints a warning			
--get-server-public-key	Request RSA public key from server	8.0.3		
--help	Display help message and exit			
--hexdump	Display a hex dump of the log in comments			
--host	Host on which MySQL server is located			
--idempotent	Cause the server to use idempotent mode while processing binary log updates from this session only			
--include-gtids	Show only the groups in the GTID set provided			
--local-load	Prepare local temporary files for LOAD DATA in the specified directory			
--login-path	Read login path options from .mylogin.cnf			
--no-defaults	Read no option files			
--offset	Skip the first N entries in the log			

Option Name	Description	Introduced	Deprecated	Removed
--password	Password to use when connecting to server			
--plugin-dir	Directory where plugins are installed			
--port	TCP/IP port number for connection			
--print-defaults	Print default options			
--print-table-metadata	Print table metadata			
--protocol	Connection protocol to use			
--raw	Write events in raw (binary) format to output files			
--read-from-remote-master	Read the binary log from a MySQL master rather than reading a local log file			
--read-from-remote-server	Read binary log from MySQL server rather than local log file			
--result-file	Direct output to named file			
--rewrite-db	Create rewrite rules for databases when playing back from logs written in row-based format. Can be used multiple times.			
--secure-auth	Do not send passwords to server in old (pre-4.1) format			8.0.3
--server-id	Extract only those events created by the server having the given server ID			
--server-id-bits	Tell mysqlbinlog how to interpret server IDs in binary log when log was written by a mysqld having its server-id-bits set to less than the maximum; supported only by MySQL Cluster version of mysqlbinlog			
--server-public-key-path	Path name to file containing RSA public key	8.0.4		
--set-charset	Add a SET NAMES charset_name statement to the output			
--shared-memory-base-name	Name of shared memory to use for shared-memory connections			
--short-form	Display only the statements contained in the log			
--skip-gtids	Do not print any GTIDs; use this when writing a dump file from binary logs containing GTIDs.			
--socket	Unix socket file or Windows named pipe to use			
--ssl-ca	File that contains list of trusted SSL Certificate Authorities			
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files			
--ssl-cert	File that contains X.509 certificate			
--ssl-cipher	Permissible ciphers for connection encryption			
--ssl-crl	File that contains certificate revocation lists			

Option Name	Description	Introduced	Deprecated	Removed
--ssl-crlpath	Directory that contains certificate revocation-list files			
--ssl-fips-mode	Whether to enable FIPS mode on client side	8.0.11		
--ssl-key	File that contains X.509 key			
--ssl-mode	Desired security state of connection to server			
--start-datetime	Read binary log from first event with timestamp equal to or later than datetime argument			
--start-position	Read binary log from first event with position equal to or greater than argument			
--stop-datetime	Stop reading binary log at first event with timestamp equal to or greater than datetime argument			
--stop-never	Stay connected to server after reading last binary log file			
--stop-never-slave-server-id	Slave server ID to report when connecting to server			
--stop-position	Stop reading binary log at first event with position equal to or greater than argument			
--tls-ciphersuites	Permissible TLSv1.3 ciphersuites for encrypted connections	8.0.16		
--tls-version	Permissible TLS protocols for encrypted connections			
--to-last-log	Do not stop at the end of requested binary log from a MySQL server, but rather continue printing to end of last binary log			
--user	MySQL user name to use when connecting to server			
--verbose	Reconstruct row events as SQL statements			
--verify-binlog-checksum	Verify checksums in binary log			
--version	Display version information and exit			
--zstd-compression-level	Compression level for connections to server that use zstd compression	8.0.18		

- `--help`, `-?`

  Display a help message and exit.

- `--base64-output=value`

  This option determines when events should be displayed encoded as base-64 strings using `BINLOG` statements. The option has these permissible values (not case-sensitive):

  - `AUTO` ("automatic") or `UNSPEC` ("unspecified") displays `BINLOG` statements automatically when necessary (that is, for format description events and row events). If no `--base64-output` option is given, the effect is the same as `--base64-output=AUTO`.

Note

Automatic BINLOG display is the only safe behavior if you intend to use the output of mysqlbinlog to re-execute binary log file contents. The other option values are intended only for debugging or testing purposes because they may produce output that does not include all events in executable form.

- NEVER causes BINLOG statements not to be displayed. mysqlbinlog exits with an error if a row event is found that must be displayed using BINLOG.
- DECODE-ROWS specifies to mysqlbinlog that you intend for row events to be decoded and displayed as commented SQL statements by also specifying the --verbose option. Like NEVER, DECODE-ROWS suppresses display of BINLOG statements, but unlike NEVER, it does not exit with an error if a row event is found.

For examples that show the effect of --base64-output and --verbose on row event output, see Section 4.6.8.2, "mysqlbinlog Row Event Display".

- --bind-address=*ip_address*

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- --binlog-row-event-max-size=*N*

Property	Value
Command-Line Format	--binlog-row-event-max-size=#
Type	Numeric
Default Value	4294967040
Minimum Value	256
Maximum Value	18446744073709547520

  Specify the maximum size of a row-based binary log event, in bytes. Rows are grouped into events smaller than this size if possible. The value should be a multiple of 256. The default is 4GB.

- --character-sets-dir=*dir_name*

  The directory where character sets are installed. See Section 10.15, "Character Set Configuration".

- --compress

  Compress all information sent between the client and the server if possible. See Section 4.2.6, "Connection Compression Control".

  This option was added in MySQL 8.0.17. As of MySQL 8.0.18 it is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- --compression-algorithms=*value*

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the protocol_compression_algorithms system variable. The default value is uncompressed.

  For more information, see Section 4.2.6, "Connection Compression Control".

This option was added in MySQL 8.0.18.

- `--connection-server-id=server_id`

  `--connection-server-id` specifies the server ID that `mysqlbinlog` reports when it connects to the server. It can be used to avoid a conflict with the ID of a slave server or another `mysqlbinlog` process.

  If the `--read-from-remote-server` option is specified, `mysqlbinlog` reports a server ID of 0, which tells the server to disconnect after sending the last log file (nonblocking behavior). If the `--stop-never` option is also specified to maintain the connection to the server, `mysqlbinlog` reports a server ID of 1 by default instead of 0, and `--connection-server-id` can be used to replace that server ID if required. See Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”.

- `--database=db_name`, `-d db_name`

  This option causes `mysqlbinlog` to output entries from the binary log (local log only) that occur while *`db_name`* is been selected as the default database by `USE`.

  The `--database` option for `mysqlbinlog` is similar to the `--binlog-do-db` option for `mysqld`, but can be used to specify only one database. If `--database` is given multiple times, only the last instance is used.

  The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of `--binlog-do-db` depend on whether statement-based or row-based logging is in use.

  Statement-based logging. The `--database` option works as follows:

  - While *`db_name`* is the default database, statements are output whether they modify tables in *`db_name`* or a different database.
  - Unless *`db_name`* is selected as the default database, statements are not output, even if they modify tables in *`db_name`*.
  - There is an exception for `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE`. The database being *created, altered, or dropped* is considered to be the default database when determining whether to output the statement.

  Suppose that the binary log was created by executing these statements using statement-based-logging:

```
INSERT INTO test.t1 (i) VALUES(100);
INSERT INTO db2.t2 (j)  VALUES(200);
USE test;
INSERT INTO test.t1 (i) VALUES(101);
INSERT INTO t1 (i)      VALUES(102);
INSERT INTO db2.t2 (j)  VALUES(201);
USE db2;
INSERT INTO test.t1 (i) VALUES(103);
INSERT INTO db2.t2 (j)  VALUES(202);
INSERT INTO t2 (j)      VALUES(203);
```

  `mysqlbinlog --database=test` does not output the first two `INSERT` statements because there is no default database. It outputs the three `INSERT` statements following `USE test`, but not the three `INSERT` statements following `USE db2`.

`mysqlbinlog --database=db2` does not output the first two `INSERT` statements because there is no default database. It does not output the three `INSERT` statements following `USE test`, but does output the three `INSERT` statements following `USE db2`.

Row-based logging. `mysqlbinlog` outputs only entries that change tables belonging to *`db_name`*. The default database has no effect on this. Suppose that the binary log just described was created using row-based logging rather than statement-based logging. `mysqlbinlog --database=test` outputs only those entries that modify `t1` in the test database, regardless of whether `USE` was issued or what the default database is.

If a server is running with `binlog_format` set to `MIXED` and you want it to be possible to use `mysqlbinlog` with the `--database` option, you must ensure that tables that are modified are in the database selected by `USE`. (In particular, no cross-database updates should be used.)

When used together with the `--rewrite-db` option, the `--rewrite-db` option is applied first; then the `--database` option is applied, using the rewritten database name. The order in which the options are provided makes no difference in this regard.

- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:o,/tmp/mysqlbinlog.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=`*`plugin`*

  A hint about which client-side authentication plugin to use. See Section 6.2.17, “Pluggable Authentication”.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=`*`file_name`*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=`*`str`*

Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysqlbinlog normally reads the [client] and [mysqlbinlog] groups. If the --defaults-group-suffix=_other option is given, mysqlbinlog also reads the [client_other] and [mysqlbinlog_other] groups.

For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --disable-log-bin, -D

  Disable binary logging. This is useful for avoiding an endless loop if you use the --to-last-log option and are sending the output to the same MySQL server. This option also is useful when restoring after a crash to avoid duplication of the statements you have logged.

  This option causes mysqlbinlog to include a SET sql_log_bin = 0 statement in its output to disable binary logging of the remaining output. Manipulating the session value of the sql_log_bin system variable is a restricted operation, so this option requires that you have privileges sufficient to set restricted session variables. See Section 5.1.9.1, "System Variable Privileges".

- --exclude-gtids=*gtid_set*

  Do not display any of the groups listed in the *gtid_set*.

- --force-if-open, -F

  Read binary log files even if they are open or were not closed properly.

- --force-read, -f

  With this option, if mysqlbinlog reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this option, mysqlbinlog stops if it reads such an event.

- --get-server-public-key

  Request from the server the public key required for RSA key pair-based password exchange. This option applies to clients that authenticate with the caching_sha2_password authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If --server-public-key-path=*file_name* is given and specifies a valid public key file, it takes precedence over --get-server-public-key.

  For information about the caching_sha2_password plugin, see Section 6.4.1.2, "Caching SHA-2 Pluggable Authentication".

- --hexdump, -H

  Display a hex dump of the log in comments, as described in Section 4.6.8.1, "mysqlbinlog Hex Dump Format". The hex output can be helpful for replication debugging.

- --host=*host_name*, -h *host_name*

  Get the binary log from the MySQL server on the given host.

- --idempotent

Tell the MySQL Server to use idempotent mode while processing updates; this causes suppression of any duplicate-key or key-not-found errors that the server encounters in the current session while processing updates. This option may prove useful whenever it is desirable or necessary to replay one or more binary logs to a MySQL Server which may not contain all of the data to which the logs refer.

The scope of effect for this option includes the current `mysqlbinlog` client and session only.

- `--include-gtids=gtid_set`

  Display only the groups listed in the *`gtid_set`*.

- `--local-load=dir_name`, `-l dir_name`

  Prepare local temporary files for `LOAD DATA` in the specified directory.

  Important

  These temporary files are not automatically removed by `mysqlbinlog` or any other MySQL program.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A "login path" is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.7, "`mysql_config_editor` — MySQL Configuration Utility".

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.7, "`mysql_config_editor` — MySQL Configuration Utility".)

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--offset=N`, `-o N`

  Skip the first *`N`* entries in the log.

- `--open-files-limit=N`

  Specify the number of open file descriptors to reserve.

- `--password[=password]`, `-p[password]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, `mysqlbinlog` prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

To explicitly specify that there is no password and that mysqlbinlog should not prompt for one, use the --skip-password option.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysqlbinlog does not find it. See Section 6.2.17, “Pluggable Authentication”.

- --port=*port_num*, -P *port_num*

  The TCP/IP port number to use for connecting to a remote server.

- --print-defaults

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --print-table-metadata

  Print table related metadata from the binary log. Configure the amount of table related metadata binary logged using binlog-row-metadata.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The connection protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

- --raw

  By default, mysqlbinlog reads binary log files and writes events in text format. The --raw option tells mysqlbinlog to write them in their original binary format. Its use requires that --read-from-remote-server also be used because the files are requested from a server. mysqlbinlog writes one output file for each file read from the server. The --raw option can be used to make a backup of a server's binary log. With the --stop-never option, the backup is “live” because mysqlbinlog stays connected to the server. By default, output files are written in the current directory with the same names as the original log files. Output file names can be modified using the --result-file option. For more information, see Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”.

- --read-from-remote-master=*type*

  Read binary logs from a MySQL server with the COM_BINLOG_DUMP or COM_BINLOG_DUMP_GTID commands by setting the option value to either BINLOG-DUMP-NON-GTIDS or BINLOG-DUMP-GTIDS, respectively. If --read-from-remote-master=BINLOG-DUMP-GTIDS is combined with --exclude-gtids, transactions can be filtered out on the master, avoiding unnecessary network traffic.

  The connection parameter options are used with this option or the --read-from-remote-server option. These options are --host, --password, --port, --protocol, --socket, and --user. If neither of the remote options is specified, the connection parameter options are ignored.

The REPLICATION SLAVE privilege is required to use this option.

- --read-from-remote-server, -R

  Read the binary log from a MySQL server rather than reading a local log file. This option requires that the remote server be running. It works only for binary log files on the remote server, not relay log files.

  The connection parameter options are used with this option or the --read-from-remote-master option. These options are --host, --password, --port, --protocol, --socket, and --user. If neither of the remote options is specified, the connection parameter options are ignored.

  The REPLICATION SLAVE privilege is required to use this option.

  This option is like --read-from-remote-master=BINLOG-DUMP-NON-GTIDS.

- --result-file=*name*, -r *name*

  Without the --raw option, this option indicates the file to which mysqlbinlog writes text output. With --raw, mysqlbinlog writes one binary output file for each log file transferred from the server, writing them by default in the current directory using the same names as the original log file. In this case, the --result-file option value is treated as a prefix that modifies output file names.

- --rewrite-db='*from_name->to_name*'

  When reading from a row-based or statement-based log, rewrite all occurrences of *from_name* to *to_name*. Rewriting is done on the rows, for row-based logs, as well as on the USE clauses, for statement-based logs.

  Warning

  Statements in which table names are qualified with database names are not rewritten to use the new name when using this option.

  The rewrite rule employed as a value for this option is a string having the form '*from_name->to_name*', as shown previously, and for this reason must be enclosed by quotation marks.

  To employ multiple rewrite rules, specify the option multiple times, as shown here:

```
shell> mysqlbinlog --rewrite-db='dbcurrent->dbold' --rewrite-db='dbtest->dbcurrent' \
                   binlog.00001 > /tmp/statements.sql
```

  When used together with the --database option, the --rewrite-db option is applied first; then --database option is applied, using the rewritten database name. The order in which the options are provided makes no difference in this regard.

  This means that, for example, if mysqlbinlog is started with --rewrite-db='mydb->yourdb' --database=yourdb, then all updates to any tables in databases mydb and yourdb are included in the output. On the other hand, if it is started with --rewrite-db='mydb->yourdb' --database=mydb, then mysqlbinlog outputs no statements at all: since all updates to mydb are first rewritten as updates to yourdb before applying the --database option, there remain no updates that match --database=mydb.

- --secure-auth

- --server-id=*id*

Display only those events created by the server having the given server ID.

- `--server-id-bits=N`

  Use only the first `N` bits of the `server_id` to identify the server. If the binary log was written by a `mysqld` with server-id-bits set to less than 32 and user data stored in the most significant bit, running `mysqlbinlog` with `--server-id-bits` set to 32 enables this data to be seen.

  This option is supported only by the version of `mysqlbinlog` supplied with the NDB Cluster distribution, or built with NDB Cluster support.

- `--server-public-key-path=file_name`

  The path name to a file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `--server-public-key-path=file_name` is given and specifies a valid public key file, it takes precedence over `--get-server-public-key`.

  For `sha256_password`, this option applies only if MySQL was built using OpenSSL.

  For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `--set-charset=charset_name`

  Add a `SET NAMES charset_name` statement to the output to specify the character set to be used for processing log files.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--short-form`, `-s`

  Display only the statements contained in the log, without any extra information or row-based events. This is for testing only, and should not be used in production systems. It is deprecated, and will be removed in a future release.

- `--skip-gtids[=(true|false)]`

  Do not display any GTIDs in the output. This is needed when writing to a dump file from one or more binary logs containing GTIDs, as shown in this example:

```
shell> mysqlbinlog --skip-gtids binlog.000001 >  /tmp/dump.sql
shell> mysqlbinlog --skip-gtids binlog.000002 >> /tmp/dump.sql
shell> mysql -u root -p -e "source /tmp/dump.sql"
```

The use of this option is otherwise not normally recommended in production.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--ssl-fips-mode={OFF|ON|STRICT}`

  Controls whether to enable FIPS mode on the client side. The `--ssl-fips-mode` option differs from other `--ssl-xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

  These `--ssl-fips-mode` values are permitted:

  - `OFF`: Disable FIPS mode.
  - `ON`: Enable FIPS mode.
  - `STRICT`: Enable “strict” FIPS mode.

  Note

  If the OpenSSL FIPS Object Module is not available, the only permitted value for `--ssl-fips-mode` is `OFF`. In this case, setting `--ssl-fips-mode` to `ON` or `STRICT` causes the client to produce a warning at startup and to operate in non-FIPS mode.

- `--start-datetime=datetime`

  Start reading the binary log at the first event having a timestamp equal to or later than the *`datetime`* argument. The *`datetime`* value is relative to the local time zone on the machine where you run `mysqlbinlog`. The value should be in a format accepted for the `DATETIME` or `TIMESTAMP` data types. For example:

  ```
  shell> mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003
  ```

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--start-position=N`, `-j N`

  Start reading the binary log at the first event having a position equal to or greater than *`N`*. This option applies to the first log file named on the command line.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--stop-datetime=datetime`

  Stop reading the binary log at the first event having a timestamp equal to or later than the *`datetime`* argument. This option is useful for point-in-time recovery. See the description of the `--start-datetime` option for information about the *`datetime`* value.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--stop-never`

  This option is used with `--read-from-remote-server`. It tells `mysqlbinlog` to remain connected to the server. Otherwise `mysqlbinlog` exits when the last log file has been transferred from the server. `--stop-never` implies `--to-last-log`, so only the first log file to transfer need be named on the command line.

  `--stop-never` is commonly used with `--raw` to make a live binary log backup, but also can be used without `--raw` to maintain a continuous text display of log events as the server generates them.

  With `--stop-never`, by default, `mysqlbinlog` reports a server ID of 1 when it connects to the server. Use `--connection-server-id` to explicitly specify an alternative ID to report. It can be used to avoid a conflict with the ID of a slave server or another `mysqlbinlog` process. See Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”.

- `--stop-never-slave-server-id=id`

  This option is deprecated and will be removed in a future release. Use the `--connection-server-id` option instead to specify a server ID for mysqlbinlog to report.

- `--stop-position=N`

  Stop reading the binary log at the first event having a position equal to or greater than *`N`*. This option applies to the last log file named on the command line.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--tls-ciphersuites=ciphersuite_list`

  The permissible ciphersuites for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- `--tls-version=protocol_list`

  The permissible TLS protocols for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `--to-last-log`, `-t`

  Do not stop at the end of the requested binary log from a MySQL server, but rather continue printing until the end of the last binary log. If you send the output to the same MySQL server, this may lead to an endless loop. This option requires `--read-from-remote-server`.

- `--user=user_name`, `-u user_name`

  The user name of the MySQL account to use when connecting to a remote server.

- `--verbose`, `-v`

  Reconstruct row events and display them as commented SQL statements, with table partition information where applicable. If this option is given twice (by passing in either "-vv" or "--verbose --verbose"), the output includes comments to indicate column data types and some metadata, and informational log events such as row query log events if the `binlog_rows_query_log_events` system variable is set to `TRUE`.

  For examples that show the effect of `--base64-output` and `--verbose` on row event output, see Section 4.6.8.2, “mysqlbinlog Row Event Display”.

- `--verify-binlog-checksum`, `-c`

  Verify checksums in binary log files.

- `--version`, `-V`

  Display version information and exit.

  The `mysqlbinlog` version number shown when using this option is 3.4.

- `--zstd-compression-level=level`

  The compression level to use for connections to the server that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. The compression level setting has no effect on connections that do not use `zstd` compression.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

You can pipe the output of `mysqlbinlog` into the `mysql` client to execute the events contained in the binary log. This technique is used to recover from a crash when you have an old backup (see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”). For example:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p
```

Or:

```
shell> mysqlbinlog binlog.[0-9]* | mysql -u root -p
```

If the statements produced by `mysqlbinlog` may contain `BLOB` values, these may cause problems when `mysql` processes them. In this case, invoke `mysql` with the `--binary-mode` option.

You can also redirect the output of `mysqlbinlog` to a text file instead, if you need to modify the statement log first (for example, to remove statements that you do not want to execute for some reason). After editing the file, execute the statements that it contains by using it as input to the `mysql` program:

```
shell> mysqlbinlog binlog.000001 > tmpfile
shell> ... edit tmpfile ...
```

```
shell> mysql -u root -p < tmpfile
```

When mysqlbinlog is invoked with the --start-position option, it displays only those events with an offset in the binary log greater than or equal to a given position (the given position must match the start of one event). It also has options to stop and start when it sees an event with a given date and time. This enables you to perform point-in-time recovery using the --stop-datetime option (to be able to say, for example, “roll forward my databases to how they were today at 10:30 a.m.”).

Processing multiple files. If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using multiple connections to the server causes problems if the first log file contains a CREATE TEMPORARY TABLE statement and the second log contains a statement that uses the temporary table. When the first mysql process terminates, the server drops the temporary table. When the second mysql process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* mysql process to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

From MySQL 8.0.12, you can also supply multiple binary log files to mysqlbinlog as streamed input using a shell pipe. An archive of compressed binary log files can be decompressed and provided directly to mysqlbinlog. In this example, binlog-files_1.gz contains multiple binary log files for processing. The pipeline extracts the contents of binlog-files_1.gz, pipes the binary log files to mysqlbinlog as standard input, and pipes the output of mysqlbinlog into the mysql client for execution:

```
shell> gzip -cd binlog-files_1.gz | ./mysqlbinlog - | ./mysql -uroot  -p
```

You can specify more than one archive file, for example:

```
shell> gzip -cd binlog-files_1.gz binlog-files_2.gz | ./mysqlbinlog - | ./mysql -uroot  -p
```

For streamed input, do not use --stop-position, because mysqlbinlog cannot identify the last log file to apply this option.

LOAD DATA operations. mysqlbinlog can produce output that reproduces a LOAD DATA operation without the original data file. mysqlbinlog copies the data to a temporary file and writes a LOAD DATA LOCAL statement that refers to the file. The default location of the directory where these files are written is system-specific. To specify a directory explicitly, use the --local-load option.

Because mysqlbinlog converts LOAD DATA statements to LOAD DATA LOCAL statements (that is, it adds LOCAL), both the client and the server that you use to process the statements must be configured with the LOCAL capability enabled. See Section 6.1.6, “Security Issues with LOAD DATA LOCAL”.

Warning

The temporary files created for `LOAD DATA LOCAL` statements are *not* automatically deleted because they are needed until you actually execute those statements. You should delete the temporary files yourself after you no longer need the statement log. The files can be found in the temporary file directory and have names like *`original_file_name-#-#`*.

4.6.8.1 mysqlbinlog Hex Dump Format

The `--hexdump` option causes `mysqlbinlog` to produce a hex dump of the binary log contents:

```
shell> mysqlbinlog --hexdump master-bin.000001
```

The hex output consists of comment lines beginning with `#`, so the output might look like this for the preceding command:

```
/*!40019 SET @@SESSION.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
# at 4
#051024 17:24:13 server id 1  end_log_pos 98
# Position  Timestamp   Type   Master ID        Size      Master Pos    Flags
# 00000004 9d fc 5c 43   0f   01 00 00 00   5e 00 00 00   62 00 00 00   00 00
# 00000017 04 00 35 2e 30 2e 31 35  2d 64 65 62 75 67 2d 6c |..5.0.15.debug.l|
# 00000027 6f 67 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |og..............|
# 00000037 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
# 00000047 00 00 00 00 9d fc 5c 43  13 38 0d 00 08 00 12 00 |.......C.8......|
# 00000057 04 04 04 04 12 00 00 4b  00 04 1a               |.......K...|
#       Start: binlog v 4, server v 5.0.15-debug-log created 051024 17:24:13
#       at startup
ROLLBACK;
```

Hex dump output currently contains the elements in the following list. This format is subject to change. For more information about binary log format, see MySQL Internals: The Binary Log.

- `Position`: The byte position within the log file.
- `Timestamp`: The event timestamp. In the example shown, `'9d fc 5c 43'` is the representation of `'051024 17:24:13'` in hexadecimal.
- `Type`: The event type code.
- `Master ID`: The server ID of the master that created the event.
- `Size`: The size in bytes of the event.
- `Master Pos`: The position of the next event in the original master log file.
- `Flags`: Event flag values.

4.6.8.2 mysqlbinlog Row Event Display

The following examples illustrate how `mysqlbinlog` displays row events that specify data modifications. These correspond to events with the `WRITE_ROWS_EVENT`, `UPDATE_ROWS_EVENT`, and `DELETE_ROWS_EVENT` type codes. The `--base64-output=DECODE-ROWS` and `--verbose` options may be used to affect row event output.

Suppose that the server is using row-based binary logging and that you execute the following sequence of statements:

```
CREATE TABLE t
(
  id   INT NOT NULL,
  name VARCHAR(20) NOT NULL,
  date DATE NULL
) ENGINE = InnoDB;

START TRANSACTION;
INSERT INTO t VALUES(1, 'apple', NULL);
UPDATE t SET name = 'pear', date = '2009-01-01' WHERE id = 1;
DELETE FROM t WHERE id = 1;
COMMIT;
```

By default, `mysqlbinlog` displays row events encoded as base-64 strings using `BINLOG` statements. Omitting extraneous lines, the output for the row events produced by the preceding statement sequence looks like this:

```
shell> mysqlbinlog log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAAEAA//8AQAAAAVhcHBsZQ==
'/*!*/;
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAAEAA////AEAAAAFYXBwbGX4AQAAAARwZWFyIbIP
'/*!*/;
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAJABAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAAEAA//4AQAAAARwZWFyIbIP
'/*!*/;
```

To see the row events as comments in the form of “pseudo-SQL” statements, run `mysqlbinlog` with the `--verbose` or `-v` option. This output level also shows table partition information where applicable. The output will contain lines beginning with `###`:

```
shell> mysqlbinlog -v log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAAEAA//8AQAAAAVhcHBsZQ==
'/*!*/;
### INSERT INTO test.t
### SET
###   @1=1
###   @2='apple'
###   @3=NULL
...
# at 302
```

```
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAEAA////AEAAAAFYXBwbGX4AQAAAARwZWFyIbIP
'/*!*/;
### UPDATE test.t
### WHERE
###   @1=1
###   @2='apple'
###   @3=NULL
### SET
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAJABAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAEAA//4AQAAAARwZWFyIbIP
'/*!*/;
### DELETE FROM test.t
### WHERE
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
```

Specify `--verbose` or `-v` twice to also display data types and some metadata for each column, and informational log events such as row query log events if the `binlog_rows_query_log_events` system variable is set to `TRUE`. The output will contain an additional comment following each column change:

```
shell> mysqlbinlog -vv log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAEAA//8AQAAAAVhcHBsZQ==
'/*!*/;
### INSERT INTO test.t
### SET
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAEAA////AEAAAAFYXBwbGX4AQAAAARwZWFyIbIP
'/*!*/;
### UPDATE test.t
### WHERE
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
### SET
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
...
# at 400
```

```
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAJABAAAAABEAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAEAA//4AQAAAARwZWFyIbIP
'/*!*/;
### DELETE FROM test.t
### WHERE
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
```

You can tell mysqlbinlog to suppress the BINLOG statements for row events by using the --base64-output=DECODE-ROWS option. This is similar to --base64-output=NEVER but does not exit with an error if a row event is found. The combination of --base64-output=DECODE-ROWS and --verbose provides a convenient way to see row events only as SQL statements:

```
shell> mysqlbinlog -v --base64-output=DECODE-ROWS log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F
### INSERT INTO test.t
### SET
###   @1=1
###   @2='apple'
###   @3=NULL
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F
### UPDATE test.t
### WHERE
###   @1=1
###   @2='apple'
###   @3=NULL
### SET
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F
### DELETE FROM test.t
### WHERE
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
```

Note

You should not suppress BINLOG statements if you intend to re-execute mysqlbinlog output.

The SQL statements produced by --verbose for row events are much more readable than the corresponding BINLOG statements. However, they do not correspond exactly to the original SQL statements that generated the events. The following limitations apply:

- The original column names are lost and replaced by @*N*, where *N* is a column number.
- Character set information is not available in the binary log, which affects string column display:
  - There is no distinction made between corresponding binary and nonbinary string types (BINARY and CHAR, VARBINARY and VARCHAR, BLOB and TEXT). The output uses a data type of STRING for fixed-length strings and VARSTRING for variable-length strings.

- For multibyte character sets, the maximum number of bytes per character is not present in the binary log, so the length for string types is displayed in bytes rather than in characters. For example, `STRING(4)` will be used as the data type for values from either of these column types:

```
CHAR(4) CHARACTER SET latin1
CHAR(2) CHARACTER SET ucs2
```

- Due to the storage format for events of type `UPDATE_ROWS_EVENT`, `UPDATE` statements are displayed with the `WHERE` clause preceding the `SET` clause.

Proper interpretation of row events requires the information from the format description event at the beginning of the binary log. Because `mysqlbinlog` does not know in advance whether the rest of the log contains row events, by default it displays the format description event using a `BINLOG` statement in the initial part of the output.

If the binary log is known not to contain any events requiring a `BINLOG` statement (that is, no row events), the `--base64-output=NEVER` option can be used to prevent this header from being written.

4.6.8.3 Using mysqlbinlog to Back Up Binary Log Files

By default, `mysqlbinlog` reads binary log files and displays their contents in text format. This enables you to examine events within the files more easily and to re-execute them (for example, by using the output as input to `mysql`). `mysqlbinlog` can read log files directly from the local file system, or, with the `--read-from-remote-server` option, it can connect to a server and request binary log contents from that server. `mysqlbinlog` writes text output to its standard output, or to the file named as the value of the `--result-file=file_name` option if that option is given.

- mysqlbinlog Backup Capabilities
- mysqlbinlog Backup Options
- Static and Live Backups
- Output File Naming
- Example: mysqldump + mysqlbinlog for Backup and Restore
- mysqlbinlog Backup Restrictions

mysqlbinlog Backup Capabilities

`mysqlbinlog` can read binary log files and write new files containing the same content—that is, in binary format rather than text format. This capability enables you to easily back up a binary log in its original format. `mysqlbinlog` can make a static backup, backing up a set of log files and stopping when the end of the last file is reached. It can also make a continuous (“live”) backup, staying connected to the server when it reaches the end of the last log file and continuing to copy new events as they are generated. In continuous-backup operation, `mysqlbinlog` runs until the connection ends (for example, when the server exits) or `mysqlbinlog` is forcibly terminated. When the connection ends, `mysqlbinlog` does not wait and retry the connection, unlike a slave replication server. To continue a live backup after the server has been restarted, you must also restart `mysqlbinlog`.

Important

`mysqlbinlog` can back up both encrypted and unencrypted binary log files . However, copies of encrypted binary log files that are generated using `mysqlbinlog` are stored in an unencrypted format.

mysqlbinlog Backup Options

Binary log backup requires that you invoke mysqlbinlog with two options at minimum:

- The --read-from-remote-server (or -R) option tells mysqlbinlog to connect to a server and request its binary log. (This is similar to a slave replication server connecting to its master server.)
- The --raw option tells mysqlbinlog to write raw (binary) output, not text output.

Along with --read-from-remote-server, it is common to specify other options: --host indicates where the server is running, and you may also need to specify connection options such as --user and --password.

Several other options are useful in conjunction with --raw:

- --stop-never: Stay connected to the server after reaching the end of the last log file and continue to read new events.
- --connection-server-id=*id*: The server ID that mysqlbinlog reports when it connects to a server. When --stop-never is used, the default reported server ID is 1. If this causes a conflict with the ID of a slave server or another mysqlbinlog process, use --connection-server-id to specify an alternative server ID. See Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”.
- --result-file: A prefix for output file names, as described later.

Static and Live Backups

To back up a server's binary log files with mysqlbinlog, you must specify file names that actually exist on the server. If you do not know the names, connect to the server and use the SHOW BINARY LOGS statement to see the current names. Suppose that the statement produces this output:

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+-----------+
| Log_name      | File_size | Encrypted |
+---------------+-----------+-----------+
| binlog.000130 |     27459 | No        |
| binlog.000131 |     13719 | No        |
| binlog.000132 |     43268 | No        |
+---------------+-----------+-----------+
```

With that information, you can use mysqlbinlog to back up the binary log to the current directory as follows (enter each command on a single line):

- To make a static backup of binlog.000130 through binlog.000132, use either of these commands:

```
mysqlbinlog --read-from-remote-server --host=host_name --raw
  binlog.000130 binlog.000131 binlog.000132

mysqlbinlog --read-from-remote-server --host=host_name --raw
  --to-last-log binlog.000130
```

  The first command specifies every file name explicitly. The second names only the first file and uses --to-last-log to read through the last. A difference between these commands is that if the server happens to open binlog.000133 before mysqlbinlog reaches the end of binlog.000132, the first command will not read it, but the second command will.

- To make a live backup in which mysqlbinlog starts with binlog.000130 to copy existing log files, then stays connected to copy new events as the server generates them:

```
mysqlbinlog --read-from-remote-server --host=host_name --raw
  --stop-never binlog.000130
```

With --stop-never, it is not necessary to specify --to-last-log to read to the last log file because that option is implied.

Output File Naming

Without --raw, mysqlbinlog produces text output and the --result-file option, if given, specifies the name of the single file to which all output is written. With --raw, mysqlbinlog writes one binary output file for each log file transferred from the server. By default, mysqlbinlog writes the files in the current directory with the same names as the original log files. To modify the output file names, use the --result-file option. In conjunction with --raw, the --result-file option value is treated as a prefix that modifies the output file names.

Suppose that a server currently has binary log files named binlog.000999 and up. If you use mysqlbinlog --raw to back up the files, the --result-file option produces output file names as shown in the following table. You can write the files to a specific directory by beginning the --result-file value with the directory path. If the --result-file value consists only of a directory name, the value must end with the pathname separator character. Output files are overwritten if they exist.

--result-file Option	**Output File Names**
--result-file=x	xbinlog.000999 and up
--result-file=/tmp/	/tmp/binlog.000999 and up
--result-file=/tmp/x	/tmp/xbinlog.000999 and up

Example: mysqldump + mysqlbinlog for Backup and Restore

The following example describes a simple scenario that shows how to use mysqldump and mysqlbinlog together to back up a server's data and binary log, and how to use the backup to restore the server if data loss occurs. The example assumes that the server is running on host *host_name* and its first binary log file is named binlog.000999. Enter each command on a single line.

Use mysqlbinlog to make a continuous backup of the binary log:

```
mysqlbinlog --read-from-remote-server --host=host_name --raw
  --stop-never binlog.000999
```

Use mysqldump to create a dump file as a snapshot of the server's data. Use --all-databases, --events, and --routines to back up all data, and --master-data=2 to include the current binary log coordinates in the dump file.

```
mysqldump --host=host_name --all-databases --events --routines --master-data=2> dump_file
```

Execute the mysqldump command periodically to create newer snapshots as desired.

If data loss occurs (for example, if the server crashes), use the most recent dump file to restore the data:

```
mysql --host=host_name -u root -p < dump_file
```

Then use the binary log backup to re-execute events that were written after the coordinates listed in the dump file. Suppose that the coordinates in the file look like this:

```
-- CHANGE MASTER TO MASTER_LOG_FILE='binlog.001002', MASTER_LOG_POS=27284;
```

If the most recent backed-up log file is named binlog.001004, re-execute the log events like this:

```
mysqlbinlog --start-position=27284 binlog.001002 binlog.001003 binlog.001004
  | mysql --host=host_name -u root -p
```

You might find it easier to copy the backup files (dump file and binary log files) to the server host to make it easier to perform the restore operation, or if MySQL does not allow remote `root` access.

mysqlbinlog Backup Restrictions

Binary log backups with `mysqlbinlog` are subject to these restrictions:

- `mysqlbinlog` does not automatically reconnect to the MySQL server if the connection is lost (for example, if a server restart occurs or there is a network outage).
- The delay for a backup is similar to the delay for a replication slave.

4.6.8.4 Specifying the mysqlbinlog Server ID

When invoked with the `--read-from-remote-server` option, `mysqlbinlog` connects to a MySQL server, specifies a server ID to identify itself, and requests binary log files from the server. You can use `mysqlbinlog` to request log files from a server in several ways:

- Specify an explicitly named set of files: For each file, `mysqlbinlog` connects and issues a `Binlog dump` command. The server sends the file and disconnects. There is one connection per file.
- Specify the beginning file and `--to-last-log`: `mysqlbinlog` connects and issues a `Binlog dump` command for all files. The server sends all files and disconnects.
- Specify the beginning file and `--stop-never` (which implies `--to-last-log`): `mysqlbinlog` connects and issues a `Binlog dump` command for all files. The server sends all files, but does not disconnect after sending the last one.

With `--read-from-remote-server` only, `mysqlbinlog` connects using a server ID of 0, which tells the server to disconnect after sending the last requested log file.

With `--read-from-remote-server` and `--stop-never`, `mysqlbinlog` connects using a nonzero server ID, so the server does not disconnect after sending the last log file. The server ID is 1 by default, but this can be changed with `--connection-server-id`.

Thus, for the first two ways of requesting files, the server disconnects because `mysqlbinlog` specifies a server ID of 0. It does not disconnect if `--stop-never` is given because `mysqlbinlog` specifies a nonzero server ID.

4.6.9 `mysqldumpslow` — Summarize Slow Query Log Files

The MySQL slow query log contains information about queries that take a long time to execute (see Section 5.4.5, “The Slow Query Log”). `mysqldumpslow` parses MySQL slow query log files and summarizes their contents.

Normally, `mysqldumpslow` groups queries that are similar except for the particular values of number and string data values. It “abstracts” these values to `N` and `'S'` when displaying summary output. To modify value abstracting behavior, use the `-a` and `-n` options.

Invoke `mysqldumpslow` like this:

```
shell> mysqldumpslow [options] [log_file ...]
```

Example of usage:

```
shell> mysqldumpslow

Reading mysql slow query log from /usr/local/mysql/data/mysqld80-slow.log
Count: 1  Time=4.32s (4s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t2 select * from t1

Count: 3  Time=2.53s (7s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t2 select * from t1 limit N

Count: 3  Time=2.13s (6s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t1 select * from t1
```

`mysqldumpslow` supports the following options.

Table 4.22 mysqldumpslow Options

Option Name	Description
-a	Do not abstract all numbers to N and strings to 'S'
-n	Abstract numbers with at least the specified digits
--debug	Write debugging information
-g	Only consider statements that match the pattern
--help	Display help message and exit
-h	Host name of the server in the log file name
-i	Name of the server instance
-l	Do not subtract lock time from total time
-r	Reverse the sort order
-s	How to sort output
-t	Display only first num queries
--verbose	Verbose mode

- `--help`

  Display a help message and exit.

- `-a`

  Do not abstract all numbers to `N` and strings to `'S'`.

- `--debug`, `-d`

  Run in debug mode.

- `-g` *`pattern`*

  Consider only queries that match the (`grep`-style) pattern.

- `-h` *`host_name`*

  Host name of MySQL server for `*-slow.log` file name. The value can contain a wildcard. The default is `*` (match all).

- `-i` *`name`*

  Name of server instance (if using `mysql.server` startup script).

- `-l`

Do not subtract lock time from total time.

- `-n N`

  Abstract numbers with at least *`N`* digits within names.

- `-r`

  Reverse the sort order.

- `-s sort_type`

  How to sort the output. The value of *`sort_type`* should be chosen from the following list:

  - `t`, `at`: Sort by query time or average query time
  - `l`, `al`: Sort by lock time or average lock time
  - `r`, `ar`: Sort by rows sent or average rows sent
  - `c`: Sort by count

  By default, `mysqldumpslow` sorts by average query time (equivalent to `-s at`).

- `-t N`

  Display only the first *`N`* queries in the output.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

4.7 MySQL Program Development Utilities

This section describes some utilities that you may find useful when developing MySQL programs.

In shell scripts, you can use the `my_print_defaults` program to parse option files and see what options would be used by a given program. The following example shows the output that `my_print_defaults` might produce when asked to show the options found in the `[client]` and `[mysql]` groups:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Note for developers: Option file handling is implemented in the C client library simply by processing all options in the appropriate group or groups before any command-line arguments. This works well for programs that use the last instance of an option that is specified multiple times. If you have a C or C++ program that handles multiply specified options this way but that doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard MySQL clients to see how to do this.

Several other language interfaces to MySQL are based on the C client library, and some of them provide a way to access option file contents. These include Perl and Python. For details, see the documentation for your preferred interface.

4.7.1 `mysql_config` — Display Options for Compiling Clients

mysql_config provides you with useful information for compiling your MySQL client and connecting it to MySQL. It is a shell script, so it is available only on Unix and Unix-like systems.

Note

pkg-config can be used as an alternative to mysql_config for obtaining information such as compiler flags or link libraries required to compile MySQL applications. For more information, see Section 28.7.4.2, “Building C API Client Programs Using pkg-config”.

mysql_config supports the following options.

- --cflags

  C Compiler flags to find include files and critical compiler flags and defines used when compiling the libmysqlclient library. The options returned are tied to the specific compiler that was used when the library was created and might clash with the settings for your own compiler. Use --include for more portable options that contain only include paths.

- --cxxflags

  Like --cflags, but for C++ compiler flags.

- --include

  Compiler options to find MySQL include files.

- --libs

  Libraries and options required to link with the MySQL client library.

- --libs_r

  Libraries and options required to link with the thread-safe MySQL client library. In MySQL 8.0, all client libraries are thread-safe, so this option need not be used. The --libs option can be used in all cases.

- --plugindir

  The default plugin directory path name, defined when configuring MySQL.

- --port

  The default TCP/IP port number, defined when configuring MySQL.

- --socket

  The default Unix socket file, defined when configuring MySQL.

- --variable=*var_name*

  Display the value of the named configuration variable. Permitted *var_name* values are pkgincludedir (the header file directory), pkglibdir (the library directory), and plugindir (the plugin directory).

- --version

  Version number for the MySQL distribution.

If you invoke mysql_config with no options, it displays a list of all options that it supports, and their values:

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
  --cflags         [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
  --cxxflags       [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
  --include        [-I/usr/local/mysql/include/mysql]
  --libs           [-L/usr/local/mysql/lib/mysql -lmysqlclient
                    -lpthread -lm -lrt -lssl -lcrypto -ldl]
  --libs_r         [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
                    -lpthread -lm -lrt -lssl -lcrypto -ldl]
  --plugindir      [/usr/local/mysql/lib/plugin]
  --socket         [/tmp/mysql.sock]
  --port           [3306]
  --version        [5.8.0-m17]
  --variable=VAR   VAR is one of:
          pkgincludedir [/usr/local/mysql/include]
          pkglibdir     [/usr/local/mysql/lib]
          plugindir     [/usr/local/mysql/lib/plugin]
```

You can use `mysql_config` within a command line using backticks to include the output that it produces for particular options. For example, to compile and link a MySQL client program, use `mysql_config` as follows:

```
gcc -c `mysql_config --cflags` progname.c
gcc -o progname progname.o `mysql_config --libs`
```

4.7.2 `my_print_defaults` — Display Options from Option Files

`my_print_defaults` displays the options that are present in option groups of option files. The output indicates what options will be used by programs that read the specified option groups. For example, the `mysqlcheck` program reads the `[mysqlcheck]` and `[client]` option groups. To see what options are present in those groups in the standard option files, invoke `my_print_defaults` like this:

```
shell> my_print_defaults mysqlcheck client
--user=myusername
--password=password
--host=localhost
```

The output consists of options, one per line, in the form that they would be specified on the command line.

`my_print_defaults` supports the following options.

- `--help`, `-?`

  Display a help message and exit.

- `--config-file=file_name`, `--defaults-file=file_name`, `-c file_name`

  Read only the given option file.

- `--debug=debug_options`, `-# debug_options`

  Write a debugging log. A typical `debug_options` string is `d:t:o,file_name`. The default is `d:t:o,/tmp/my_print_defaults.trace`.

- `--defaults-extra-file=file_name`, `--extra-file=file_name`, `-e file_name`

  Read this option file after the global option file but (on Unix) before the user option file.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=suffix`, `-g suffix`

  In addition to the groups named on the command line, read groups that have the given suffix.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--login-path=name`, `-l name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-defaults`, `-n`

  Return an empty string.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--show`, `-s`

  `my_print_defaults` masks passwords by default. Use this option to display passwords as cleartext.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

- `--version`, `-V`

  Display version information and exit.

4.8 Miscellaneous Programs

4.8.1 `lz4_decompress` — Decompress mysqlpump LZ4-Compressed Output

The `lz4_decompress` utility decompresses `mysqlpump` output that was created using LZ4 compression.

Note

If MySQL was configured with the `-DWITH_LZ4=system` option, `lz4_decompress` is not built. In this case, the system `lz4` command can be used instead.

Invoke `lz4_decompress` like this:

```
shell> lz4_decompress input_file output_file
```

Example:

```
```

```
shell> mysqlpump --compress-output=LZ4 > dump.lz4
shell> lz4_decompress dump.lz4 dump.txt
```

To see a help message, invoke lz4_decompress with no arguments.

To decompress mysqlpump ZLIB-compressed output, use zlib_decompress. See Section 4.8.3, “zlib_decompress — Decompress mysqlpump ZLIB-Compressed Output”.

4.8.2 perror — Display MySQL Error Message Information

perror displays the error message for MySQL or operating system error codes. Invoke perror like this:

```
shell> perror [options] errorcode ...
```

perror attempts to be flexible in understanding its arguments. For example, for the ER_WRONG_VALUE_FOR_VAR error, perror understands any of these arguments: 1231, 001231, MY-1231, or MY-001231, or ER_WRONG_VALUE_FOR_VAR.

```
shell> perror 1231
MySQL error code MY-001231 (ER_WRONG_VALUE_FOR_VAR): Variable '%-.64s'
can't be set to the value of '%-.200s'
```

If an error number is in the range where MySQL and operating system errors overlap, perror displays both error messages:

```
shell> perror 1 13
OS error code   1:  Operation not permitted
MySQL error code MY-000001: Can't create/write to file '%s' (OS errno %d - %s)
OS error code  13:  Permission denied
MySQL error code MY-000013: Can't get stat of '%s' (OS errno %d - %s)
```

To obtain the error message for a MySQL Cluster error code, use the ndb_perror utility.

The meaning of system error messages may be dependent on your operating system. A given error code may mean different things on different operating systems.

perror supports the following options.

- --help, --info, -I, -?

  Display a help message and exit.

- --ndb

  Print the error message for a MySQL Cluster error code.

  This option was removed in MySQL 8.0.13. Use the ndb_perror utility instead.

- --silent, -s

  Silent mode. Print only the error message.

- --verbose, -v

  Verbose mode. Print error code and message. This is the default behavior.

- --version, -V

  Display version information and exit.

4.8.3 zlib_decompress — Decompress mysqlpump ZLIB-Compressed Output

The zlib_decompress utility decompresses mysqlpump output that was created using ZLIB compression.

Note

If MySQL was configured with the -DWITH_ZLIB=system option, zlib_decompress is not built. In this case, the system openssl zlib command can be used instead.

Invoke zlib_decompress like this:

```
shell> zlib_decompress input_file output_file
```

Example:

```
shell> mysqlpump --compress-output=ZLIB > dump.zlib
shell> zlib_decompress dump.zlib dump.txt
```

To see a help message, invoke zlib_decompress with no arguments.

To decompress mysqlpump LZ4-compressed output, use lz4_decompress. See Section 4.8.1, "lz4_decompress — Decompress mysqlpump LZ4-Compressed Output".

4.9 MySQL Environment Variables

This section lists environment variables that are used directly or indirectly by MySQL. Most of these can also be found in other places in this manual.

Options on the command line take precedence over values specified in option files and environment variables, and values in option files take precedence over values in environment variables. In many cases, it is preferable to use an option file instead of environment variables to modify the behavior of MySQL. See Section 4.2.2.2, "Using Option Files".

Variable	Description
AUTHENTICATION_PAM_LOG	PAM authentication plugin debug logging settings.
CC	The name of your C compiler (for running CMake).
CXX	The name of your C++ compiler (for running CMake).
CC	The name of your C compiler (for running CMake).
DBI_USER	The default user name for Perl DBI.
DBI_TRACE	Trace options for Perl DBI.
HOME	The default path for the mysql history file is $HOME/.mysql_history.
LD_RUN_PATH	Used to specify the location of libmysqlclient.so.
LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN	Enable mysql_clear_password authentication plugin; see Section 6.4.1.4, "Client-Side Cleartext Pluggable Authentication".
LIBMYSQL_PLUGIN_DIR	Directory in which to look for client plugins.
LIBMYSQL_PLUGINS	Client plugins to preload.

Variable	Description
MYSQL_DEBUG	Debug trace options when debugging.
MYSQL_GROUP_SUFFIX	Option group suffix value (like specifying --defaults-group-suffix).
MYSQL_HISTFILE	The path to the mysql history file. If this variable is set, its value overrides the default for $HOME/.mysql_history.
MYSQL_HISTIGNORE	Patterns specifying statements that mysql should not log to $HOME/.mysql_history, or syslog if --syslog is given.
MYSQL_HOME	The path to the directory in which the server-specific my.cnf file resides.
MYSQL_HOST	The default host name used by the mysql command-line client.
MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD	Maximum key length for CREATE_DH_PARAMETERS(). See Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”.
MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD	Maximum DSA key length for CREATE_ASYMMETRIC_PRIV_KEY(). See Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”.
MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD	Maximum RSA key length for CREATE_ASYMMETRIC_PRIV_KEY(). See Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”.
MYSQL_PS1	The command prompt to use in the mysql command-line client.
MYSQL_PWD	The default password when connecting to mysqld. Using this is insecure. See note following table.
MYSQL_TCP_PORT	The default TCP/IP port number.
MYSQL_TEST_LOGIN_FILE	The name of the .mylogin.cnf login path file.
MYSQL_TEST_TRACE_CRASH	Whether the test protocol trace plugin crashes clients. See note following table.
MYSQL_TEST_TRACE_DEBUG	Whether the test protocol trace plugin produces output. See note following table.
MYSQL_UNIX_PORT	The default Unix socket file name; used for connections to localhost.
MYSQLX_TCP_PORT	The X Plugin default TCP/IP port number.
MYSQLX_UNIX_PORT	The X Plugin default Unix socket file name; used for connections to localhost.
NOTIFY_SOCKET	Socket used by mysqld to communicate with systemd.
PATH	Used by the shell to find MySQL programs.
PKG_CONFIG_PATH	Location of mysqlclient.pc pkg-config file. See note following table.
TMPDIR	The directory in which temporary files are created.
TZ	This should be set to your local time zone. See Section B.4.3.7, “Time Zone Problems”.

Variable	Description
UMASK	The user-file creation mode when creating files. See note following table.
UMASK_DIR	The user-directory creation mode when creating directories. See note following table.
USER	The default user name on Windows when connecting to mysqld.

For information about the mysql history file, see Section 4.5.1.3, “mysql Client Logging”.

Use of MYSQL_PWD to specify a MySQL password must be considered *extremely insecure* and should not be used. Some versions of ps include an option to display the environment of running processes. On some systems, if you set MYSQL_PWD, your password is exposed to any other user who runs ps. Even on systems without such a version of ps, it is unwise to assume that there are no other methods by which users can examine process environments.

MYSQL_PWD is deprecated as of MySQL 8.0 and will be removed in a future MySQL version.

MYSQL_TEST_LOGIN_FILE is the path name of the login path file (the file created by mysql_config_editor). If not set, the default value is %APPDATA%\MySQL\.mylogin.cnf directory on Windows and $HOME/.mylogin.cnf on non-Windows systems. See Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”.

The MYSQL_TEST_TRACE_DEBUG and MYSQL_TEST_TRACE_CRASH variables control the test protocol trace client plugin, if MySQL is built with that plugin enabled. For more information, see Using the Test Protocol Trace Plugin.

The default UMASK and UMASK_DIR values are 0640 and 0750, respectively. MySQL assumes that the value for UMASK or UMASK_DIR is in octal if it starts with a zero. For example, setting UMASK=0600 is equivalent to UMASK=384 because 0600 octal is 384 decimal.

The UMASK and UMASK_DIR variables, despite their names, are used as modes, not masks:

- If UMASK is set, mysqld uses ($UMASK | 0600) as the mode for file creation, so that newly created files have a mode in the range from 0600 to 0666 (all values octal).
- If UMASK_DIR is set, mysqld uses ($UMASK_DIR | 0700) as the base mode for directory creation, which then is AND-ed with ~(~$UMASK & 0666), so that newly created directories have a mode in the range from 0700 to 0777 (all values octal). The AND operation may remove read and write permissions from the directory mode, but not execute permissions.

It may be necessary to set PKG_CONFIG_PATH if you use pkg-config for building MySQL programs. See Section 28.7.4.2, “Building C API Client Programs Using pkg-config”.

Chapter 5 MySQL Server Administration

Table of Contents

5.1 The MySQL Server 582
5.1.1 Configuring the Server 582
5.1.2 Server Configuration Defaults 584
5.1.3 Server Configuration Validation 584
5.1.4 Server Option, System Variable, and Status Variable Reference 585
5.1.5 Server System Variable Reference 632
5.1.6 Server Status Variable Reference 657
5.1.7 Server Command Options 673
5.1.8 Server System Variables 704
5.1.9 Using System Variables 861
5.1.10 Server Status Variables 892
5.1.11 Server SQL Modes 914
5.1.12 IPv6 Support 926
5.1.13 MySQL Server Time Zone Support 930
5.1.14 Server-Side Help Support 936
5.1.15 Server Tracking of Client Session State Changes 936
5.1.16 Server Response to Signals 940
5.1.17 The Server Shutdown Process 940
5.2 The MySQL Data Directory 942
5.3 The mysql System Schema 943
5.4 MySQL Server Logs 948
5.4.1 Selecting General Query Log and Slow Query Log Output Destinations 949
5.4.2 The Error Log 951
5.4.3 The General Query Log 969
5.4.4 The Binary Log 971
5.4.5 The Slow Query Log 984
5.4.6 The DDL Log 987
5.4.7 Server Log Maintenance 988
5.5 MySQL Server Components 989
5.5.1 Installing and Uninstalling Components 990
5.5.2 Obtaining Server Component Information 991
5.5.3 Error Log Components 991
5.6 MySQL Server Plugins 993
5.6.1 Installing and Uninstalling Plugins 994
5.6.2 Obtaining Server Plugin Information 998
5.6.3 MySQL Enterprise Thread Pool 999
5.6.4 The Rewriter Query Rewrite Plugin 1006
5.6.5 The ddl_rewriter Plugin 1016
5.6.6 Version Tokens 1018
5.6.7 The Clone Plugin 1030
5.7 MySQL Server User-Defined Functions 1054
5.7.1 Installing and Uninstalling User-Defined Functions 1054
5.7.2 Obtaining User-Defined Function Information 1055
5.8 Running Multiple MySQL Instances on One Machine 1055
5.8.1 Setting Up Multiple Data Directories 1057
5.8.2 Running Multiple MySQL Instances on Windows 1058
5.8.3 Running Multiple MySQL Instances on Unix 1061
5.8.4 Using Client Programs in a Multiple-Server Environment 1062

MySQL Server (`mysqld`) is the main program that does most of the work in a MySQL installation. This chapter provides an overview of MySQL Server and covers general server administration:

- Server configuration
- The data directory, particularly the `mysql` system schema
- The server log files
- Management of multiple servers on a single machine

For additional information on administrative topics, see also:

- Chapter 6, *Security*
- Chapter 7, *Backup and Recovery*
- Chapter 17, *Replication*

5.1 The MySQL Server

`mysqld` is the MySQL server. The following discussion covers these MySQL server configuration topics:

- Startup options that the server supports. You can specify these options on the command line, through configuration files, or both.
- Server system variables. These variables reflect the current state and values of the startup options, some of which can be modified while the server is running.
- Server status variables. These variables contain counters and statistics about runtime operation.
- How to set the server SQL mode. This setting modifies certain aspects of SQL syntax and semantics, for example for compatibility with code from other database systems, or to control the error handling for particular situations.
- Configuring and using IPv6 support.
- Configuring and using time zone support.
- Server-side help capabilities.
- The server shutdown process. There are performance and reliability considerations depending on the type of table (transactional or nontransactional) and whether you use replication.

For listings of MySQL server variables and options that have been added, deprecated, or removed in MySQL 8.0, see Section 1.5, “Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 8.0”.

Note

Not all storage engines are supported by all MySQL server binaries and configurations. To find out how to determine which storage engines your MySQL server installation supports, see Section 13.7.7.16, “SHOW ENGINES Syntax”.

5.1.1 Configuring the Server

The MySQL server, `mysqld`, has many command options and system variables that can be set at startup to configure its operation. To determine the default command option and system variable values used by the server, execute this command:

```
shell> mysqld --verbose --help
```

The command produces a list of all `mysqld` options and configurable system variables. Its output includes the default option and variable values and looks something like this:

```
abort-slave-event-count           0
allow-suspicious-udfs             FALSE
archive                           ON
auto-increment-increment          1
auto-increment-offset             1
autocommit                        TRUE
automatic-sp-privileges           TRUE
avoid-temporal-upgrade            FALSE
back-log                          80
basedir                           /home/jon/bin/mysql-8.0/
...
tmpdir                            /tmp
transaction-alloc-block-size      8192
transaction-isolation             REPEATABLE-READ
transaction-prealloc-size         4096
transaction-read-only             FALSE
transaction-write-set-extraction  OFF
updatable-views-with-limit        YES
validate-user-plugins             TRUE
verbose                           TRUE
wait-timeout                      28800
```

To see the current system variable values actually used by the server as it runs, connect to it and execute this statement:

```
mysql> SHOW VARIABLES;
```

To see some statistical and status indicators for a running server, execute this statement:

```
mysql> SHOW STATUS;
```

System variable and status information also is available using the `mysqladmin` command:

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

For a full description of all command options, system variables, and status variables, see these sections:

- Section 5.1.7, “Server Command Options”
- Section 5.1.8, “Server System Variables”
- Section 5.1.10, “Server Status Variables”

More detailed monitoring information is available from the Performance Schema; see Chapter 26, *MySQL Performance Schema*. In addition, the MySQL `sys` schema is a set of objects that provides convenient access to data collected by the Performance Schema; see Chapter 27, *MySQL sys Schema*.

If you specify an option on the command line for `mysqld` or `mysqld_safe`, it remains in effect only for that invocation of the server. To use the option every time the server runs, put it in an option file. See Section 4.2.2.2, “Using Option Files”.

5.1.2 Server Configuration Defaults

The MySQL server has many operating parameters, which you can change at server startup using command-line options or configuration files (option files). It is also possible to change many parameters at runtime. For general instructions on setting parameters at startup or runtime, see Section 5.1.7, “Server Command Options”, and Section 5.1.8, “Server System Variables”.

On Windows, MySQL Installer interacts with the user and creates a file named `my.ini` in the base installation directory as the default option file.

Note

On Windows, the `.ini` or `.cnf` option file extension might not be displayed.

After completing the installation process, you can edit the default option file at any time to modify the parameters used by the server. For example, to use a parameter setting in the file that is commented with a `#` character at the beginning of the line, remove the `#`, and modify the parameter value if necessary. To disable a setting, either add a `#` to the beginning of the line or remove it.

For non-Windows platforms, no default option file is created during either the server installation or the data directory initialization process. Create your option file by following the instructions given in Section 4.2.2.2, “Using Option Files”. Without an option file, the server just starts with its default settings—see Section 5.1.2, “Server Configuration Defaults” on how to check those settings.

For additional information about option file format and syntax, see Section 4.2.2.2, “Using Option Files”.

5.1.3 Server Configuration Validation

As of MySQL 8.0.16, MySQL Server supports a `--validate-config` option that enables the startup configuration to be checked for problems without running the server in normal operational mode:

```
mysqld --validate-config
```

If no errors are found, the server terminates with an exit code of 0. If an error is found, the server displays a diagnostic message and terminates with an exit code of 1. For example:

```
shell> mysqld --validate-config --no-such-option
2018-11-05T17:50:12.738919Z 0 [ERROR] [MY-000068] [Server] unknown
option '--no-such-option'.
2018-11-05T17:50:12.738962Z 0 [ERROR] [MY-010119] [Server] Aborting
```

The server terminates as soon as any error is found. For additional checks to occur, correct the initial problem and run the server with `--validate-config` again.

For the preceding example, where use of `--validate-config` results in display of an error message, the server exit code is 1. Warning and information messages may also be displayed, depending on the `log_error_verbosity` value, but do not produce immediate validation termination or an exit code of 1. For example, this command produces multiple warnings, both of which are displayed. But no error occurs, so the exit code is 0:

```
shell> mysqld --validate-config --log_error_verbosity=2
         --read-only=s --transaction_read_only=s
2018-11-05T15:43:18.445863Z 0 [Warning] [MY-000076] [Server] option
'read_only': boolean value 's' was not recognized. Set to OFF.
```

```
2018-11-05T15:43:18.445882Z 0 [Warning] [MY-000076] [Server] option
'transaction-read-only': boolean value 's' was not recognized. Set to OFF.
```

This command produces the same warnings, but also an error, so the error message is displayed along with the warnings and the exit code is 1:

```
shell> mysqld --validate-config --log_error_verbosity=2
         --no-such-option --read-only=s --transaction_read_only=s
2018-11-05T15:43:53.152886Z 0 [Warning] [MY-000076] [Server] option
'read_only': boolean value 's' was not recognized. Set to OFF.
2018-11-05T15:43:53.152913Z 0 [Warning] [MY-000076] [Server] option
'transaction-read-only': boolean value 's' was not recognized. Set to OFF.
2018-11-05T15:43:53.164889Z 0 [ERROR] [MY-000068] [Server] unknown
option '--no-such-option'.
2018-11-05T15:43:53.165053Z 0 [ERROR] [MY-010119] [Server] Aborting
```

The scope of the `--validate-config` option is limited to configuration checking that the server can perform without undergoing its normal startup process. As such, the configuration check does not initialize storage engines and other plugins, components, and so forth, and does not validate options associated with those uninitialized subsystems.

`--validate-config` can be used any time, but is particularly useful after an upgrade, to check whether any options previously used with the older server are considered by the upgraded server to be deprecated or obsolete. For example, the `tx_read_only` system variable was deprecated in MySQL 5.7 and removed in 8.0. Suppose that a MySQL 5.7 server was run using that system variable in its `my.cnf` file and then upgraded to MySQL 8.0. Running the upgraded server with `--validate-config` to check the configuration produces this result:

```
shell> mysqld --validate-config
2018-11-05T10:40:02.712141Z 0 [ERROR] [MY-000067] [Server] unknown variable
'tx_read_only=ON'.
2018-11-05T10:40:02.712178Z 0 [ERROR] [MY-010119] [Server] Aborting
```

`--validate-config` can be used with the `--defaults-file` option to validate only the options in a specific file:

```
shell> mysqld --defaults-file=./my.cnf-test --validate-config
2018-11-05T10:40:02.712141Z 0 [ERROR] [MY-000067] [Server] unknown variable
'tx_read_only=ON'.
2018-11-05T10:40:02.712178Z 0 [ERROR] [MY-010119] [Server] Aborting
```

Remember that `--defaults-file`, if specified, must be the first option on the command line. (Executing the preceding example with the option order reversed produces a message that `--defaults-file` itself is unknown.)

5.1.4 Server Option, System Variable, and Status Variable Reference

The following table lists all command-line options, system variables, and status variables applicable within `mysqld`.

The table lists command-line options (Cmd-line), options valid in configuration files (Option file), server system variables (System Var), and status variables (Status var) in one unified list, with an indication of where each option or variable is valid. If a server option set on the command line or in an option file differs from the name of the corresponding system variable, the variable name is noted immediately below the corresponding option. For system and status variables, the scope of the variable (Var Scope) is Global, Session, or both. Please see the corresponding item descriptions for details on setting and using the options and variables. Where appropriate, direct links to further information about the items are provided.

For a version of this table that is specific to NDB Cluster, see Section 22.3.2.5, “NDB Cluster mysqld Option and Variable Reference”.

Table 5.1 Command-Line Option, System Variable, and Status Variable Summary

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
abort-slave-event-count	Yes	Yes				
Aborted_clients				Yes	Global	No
Aborted_connects				Yes	Global	No
Acl_cache_items_count				Yes	Global	No
activate_all_roles_on_login	Yes	Yes	Yes		Global	Yes
admin_address	Yes	Yes	Yes		Global	No
admin_port	Yes	Yes	Yes		Global	No
allow-suspicious-udfs	Yes	Yes				
ansi	Yes	Yes				
audit-log	Yes	Yes				
audit_log_buffer_size	Yes	Yes	Yes		Global	No
audit_log_compression	Yes	Yes	Yes		Global	No
audit_log_connection_policy	Yes	Yes	Yes		Global	Yes
audit_log_current_session			Yes		Both	No
Audit_log_current_size				Yes	Global	No
audit_log_encryption	Yes	Yes	Yes		Global	No
Audit_log_event_max_drop_size				Yes	Global	No
Audit_log_events				Yes	Global	No
Audit_log_events_filtered				Yes	Global	No
Audit_log_events_lost				Yes	Global	No
Audit_log_events_written				Yes	Global	No
audit_log_exclude_accounts	Yes	Yes	Yes		Global	Yes
audit_log_file	Yes	Yes	Yes		Global	No
audit_log_filter_id			Yes		Both	No
audit_log_flush			Yes		Global	Yes
audit_log_format	Yes	Yes	Yes		Global	No
audit_log_include_accounts	Yes	Yes	Yes		Global	Yes
audit_log_password_history_keep_days	Yes	Yes	Yes		Global	Yes
audit_log_policy	Yes	Yes	Yes		Global	No
audit_log_read_buffer_size	Yes	Yes	Yes		Varies	Varies
audit_log_rotate_on_size	Yes	Yes	Yes		Global	Yes
audit_log_statement_policy	Yes	Yes	Yes		Global	Yes
audit_log_strategy	Yes	Yes	Yes		Global	No
Audit_log_total_size				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Audit_log_write_waits				Yes	Global	No
authentication_ldap_sasl_auth_method_name	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_bind_base_dn	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_bind_root_dn	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_bind_root_pwd	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_ca_path	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_group_search_attr	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_group_search_filter	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_init_pool_size	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_log_status	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_max_pool_size	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_server_host	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_server_port	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_tls	Yes	Yes	Yes		Global	Ye
authentication_ldap_sasl_user_search_attr	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_auth_method_name	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_bind_base_dn	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_bind_root_dn	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_bind_root_pwd	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_ca_path	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_group_search_attr	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_group_search_filter	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_init_pool_size	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_log_status	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_max_pool_size	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_server_host	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_server_port	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_tls	Yes	Yes	Yes		Global	Ye
authentication_ldap_simple_user_search_attr	Yes	Yes	Yes		Global	Ye
authentication_windows_log_level	Yes	Yes	Yes		Global	No
authentication_windows_use_principal_name	Yes	Yes	Yes		Global	No
auto_generate_certs	Yes	Yes	Yes		Global	No
auto_increment_increment	Yes	Yes	Yes		Both	Ye
auto_increment_offset	Yes	Yes	Yes		Both	Ye
autocommit	Yes	Yes	Yes		Both	Ye
automatic_sp_privileges	Yes	Yes	Yes		Global	Ye
avoid_temporal_upgrade	Yes	Yes	Yes		Global	Ye
back_log	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
basedir	Yes	Yes	Yes		Global	No
big_tables	Yes	Yes	Yes		Both	Yes
bind_address	Yes	Yes	Yes		Global	No
Binlog_cache_disk_use				Yes	Global	No
binlog_cache_size	Yes	Yes	Yes		Global	Yes
Binlog_cache_use				Yes	Global	No
binlog-checksum	Yes	Yes				
binlog_checksum	Yes	Yes	Yes		Global	Yes
binlog_direct_non_transactional_updates	Yes	Yes	Yes		Both	Yes
binlog-do-db	Yes	Yes				
binlog_encryption	Yes	Yes	Yes		Global	Yes
binlog_error_action	Yes	Yes	Yes		Global	Yes
binlog_expire_logs_seconds	Yes	Yes	Yes		Global	Yes
binlog_format	Yes	Yes	Yes		Both	Yes
binlog_group_commit_sync_delay	Yes	Yes	Yes		Global	Yes
binlog_group_commit_sync_no_delay_count	Yes	Yes	Yes		Global	Yes
binlog_gtid_simple_recovery	Yes	Yes	Yes		Global	No
binlog-ignore-db	Yes	Yes				
binlog_max_flush_queue_time	Yes	Yes	Yes		Global	Yes
binlog_order_commits	Yes	Yes	Yes		Global	Yes
binlog_rotate_encryption_master_key_at_startup	Yes	Yes	Yes		Global	No
binlog-row-event-max-size	Yes	Yes	Yes		Global	No
binlog_row_image	Yes	Yes	Yes		Both	Yes
binlog_row_metadata	Yes	Yes	Yes		Global	Yes
binlog_row_value_options	Yes	Yes	Yes		Both	Yes
binlog_rows_query_log_events	Yes	Yes	Yes		Both	Yes
Binlog_stmt_cache_disk_use				Yes	Global	No
binlog_stmt_cache_size	Yes	Yes	Yes		Global	Yes
Binlog_stmt_cache_use				Yes	Global	No
binlog_transaction_dependency_history_size	Yes	Yes	Yes		Global	Yes
binlog_transaction_dependency_tracking	Yes	Yes	Yes		Global	Yes
block_encryption_mode	Yes	Yes	Yes		Both	Yes
bulk_insert_buffer_size	Yes	Yes	Yes		Both	Yes
Bytes_received				Yes	Both	No
Bytes_sent				Yes	Both	No
caching_sha2_password_auto_generate_rsa_keys	Yes	Yes	Yes		Global	No
caching_sha2_password_private_key_path	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
caching_sha2_password_public_key_path	Yes	Yes	Yes		Global	No
Caching_sha2_password_rsa_public_key				Yes	Global	No
character_set_client			Yes		Both	Ye
character-set-client-handshake	Yes	Yes				
character_set_connection			Yes		Both	Ye
character_set_database (note 1)			Yes		Both	Ye
character_set_filesystem	Yes	Yes	Yes		Both	Ye
character_set_results			Yes		Both	Ye
character_set_server	Yes	Yes	Yes		Both	Ye
character_set_system			Yes		Global	No
character_sets_dir	Yes	Yes	Yes		Global	No
check_proxy_users	Yes	Yes	Yes		Global	Ye
chroot	Yes	Yes				
clone_autotune_concurrency	Yes	Yes	Yes		Global	Ye
clone_buffer_size	Yes	Yes	Yes		Global	Ye
clone_ddl_timeout	Yes	Yes	Yes		Global	Ye
clone_enable_compression	Yes	Yes	Yes		Global	Ye
clone_max_concurrency	Yes	Yes	Yes		Global	Ye
clone_max_data_bandwidth	Yes	Yes	Yes		Global	Ye
clone_max_network_bandwidth	Yes	Yes	Yes		Global	Ye
clone_ssl_ca	Yes	Yes	Yes		Global	Ye
clone_ssl_cert	Yes	Yes	Yes		Global	Ye
clone_ssl_key	Yes	Yes	Yes		Global	Ye
clone_valid_donor_list	Yes	Yes	Yes		Global	Ye
collation_connection			Yes		Both	Ye
collation_database (note 1)			Yes		Both	Ye
collation_server	Yes	Yes	Yes		Both	Ye
Com_admin_commands				Yes	Both	No
Com_alter_db				Yes	Both	No
Com_alter_event				Yes	Both	No
Com_alter_function				Yes	Both	No
Com_alter_procedure				Yes	Both	No
Com_alter_resource_group				Yes	Global	No
Com_alter_server				Yes	Both	No
Com_alter_table				Yes	Both	No
Com_alter_tablespace				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Com_alter_user				Yes	Both	No
Com_alter_user_default_role				Yes	Global	No
Com_analyze				Yes	Both	No
Com_assign_to_keycache				Yes	Both	No
Com_begin				Yes	Both	No
Com_binlog				Yes	Both	No
Com_call_procedure				Yes	Both	No
Com_change_db				Yes	Both	No
Com_change_master				Yes	Both	No
Com_change_repl_filter				Yes	Both	No
Com_check				Yes	Both	No
Com_checksum				Yes	Both	No
Com_clone				Yes	Global	No
Com_commit				Yes	Both	No
Com_create_db				Yes	Both	No
Com_create_event				Yes	Both	No
Com_create_function				Yes	Both	No
Com_create_index				Yes	Both	No
Com_create_procedure				Yes	Both	No
Com_create_resource_group				Yes	Global	No
Com_create_role				Yes	Global	No
Com_create_server				Yes	Both	No
Com_create_table				Yes	Both	No
Com_create_trigger				Yes	Both	No
Com_create_udf				Yes	Both	No
Com_create_user				Yes	Both	No
Com_create_view				Yes	Both	No
Com_dealloc_sql				Yes	Both	No
Com_delete				Yes	Both	No
Com_delete_multi				Yes	Both	No
Com_do				Yes	Both	No
Com_drop_db				Yes	Both	No
Com_drop_event				Yes	Both	No
Com_drop_function				Yes	Both	No
Com_drop_index				Yes	Both	No
Com_drop_procedure				Yes	Both	No
Com_drop_resource_group				Yes	Global	No
Com_drop_role				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Com_drop_server				Yes	Both	No
Com_drop_table				Yes	Both	No
Com_drop_trigger				Yes	Both	No
Com_drop_user				Yes	Both	No
Com_drop_view				Yes	Both	No
Com_empty_query				Yes	Both	No
Com_execute_sql				Yes	Both	No
Com_explain_other				Yes	Both	No
Com_flush				Yes	Both	No
Com_get_diagnostics				Yes	Both	No
Com_grant				Yes	Both	No
Com_grant_roles				Yes	Global	No
Com_group_replication_start				Yes	Global	No
Com_group_replication_stop				Yes	Global	No
Com_ha_close				Yes	Both	No
Com_ha_open				Yes	Both	No
Com_ha_read				Yes	Both	No
Com_help				Yes	Both	No
Com_insert				Yes	Both	No
Com_insert_select				Yes	Both	No
Com_install_component				Yes	Global	No
Com_install_plugin				Yes	Both	No
Com_kill				Yes	Both	No
Com_load				Yes	Both	No
Com_lock_tables				Yes	Both	No
Com_optimize				Yes	Both	No
Com_preload_keys				Yes	Both	No
Com_prepare_sql				Yes	Both	No
Com_purge				Yes	Both	No
Com_purge_before_date				Yes	Both	No
Com_release_savepoint				Yes	Both	No
Com_rename_table				Yes	Both	No
Com_rename_user				Yes	Both	No
Com_repair				Yes	Both	No
Com_replace				Yes	Both	No
Com_replace_select				Yes	Both	No
Com_reset				Yes	Both	No
Com_resignal				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Com_revoke				Yes	Both	No
Com_revoke_all				Yes	Both	No
Com_revoke_roles				Yes	Global	No
Com_rollback				Yes	Both	No
Com_rollback_to_savepoint				Yes	Both	No
Com_savepoint				Yes	Both	No
Com_select				Yes	Both	No
Com_set_option				Yes	Both	No
Com_set_resource_group				Yes	Global	No
Com_set_role				Yes	Global	No
Com_show_authors				Yes	Both	No
Com_show_binlog_events				Yes	Both	No
Com_show_binlogs				Yes	Both	No
Com_show_charsets				Yes	Both	No
Com_show_collations				Yes	Both	No
Com_show_contributors				Yes	Both	No
Com_show_create_db				Yes	Both	No
Com_show_create_event				Yes	Both	No
Com_show_create_func				Yes	Both	No
Com_show_create_proc				Yes	Both	No
Com_show_create_table				Yes	Both	No
Com_show_create_trigger				Yes	Both	No
Com_show_create_user				Yes	Both	No
Com_show_databases				Yes	Both	No
Com_show_engine_logs				Yes	Both	No
Com_show_engine_mutex				Yes	Both	No
Com_show_engine_status				Yes	Both	No
Com_show_errors				Yes	Both	No
Com_show_events				Yes	Both	No
Com_show_fields				Yes	Both	No
Com_show_function_code				Yes	Both	No
Com_show_function_status				Yes	Both	No
Com_show_grants				Yes	Both	No
Com_show_keys				Yes	Both	No
Com_show_master_status				Yes	Both	No
Com_show_ndb_status				Yes	Both	No
Com_show_open_tables				Yes	Both	No
Com_show_plugins				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Com_show_privileges				Yes	Both	No
Com_show_procedure_code				Yes	Both	No
Com_show_procedure_status				Yes	Both	No
Com_show_processlist				Yes	Both	No
Com_show_profile				Yes	Both	No
Com_show_profiles				Yes	Both	No
Com_show_relaylog_events				Yes	Both	No
Com_show_slave_hosts				Yes	Both	No
Com_show_slave_status				Yes	Both	No
Com_show_status				Yes	Both	No
Com_show_storage_engines				Yes	Both	No
Com_show_table_status				Yes	Both	No
Com_show_tables				Yes	Both	No
Com_show_triggers				Yes	Both	No
Com_show_variables				Yes	Both	No
Com_show_warnings				Yes	Both	No
Com_shutdown				Yes	Both	No
Com_signal				Yes	Both	No
Com_slave_start				Yes	Both	No
Com_slave_stop				Yes	Both	No
Com_stmt_close				Yes	Both	No
Com_stmt_execute				Yes	Both	No
Com_stmt_fetch				Yes	Both	No
Com_stmt_prepare				Yes	Both	No
Com_stmt_reprepare				Yes	Both	No
Com_stmt_reset				Yes	Both	No
Com_stmt_send_long_data				Yes	Both	No
Com_truncate				Yes	Both	No
Com_uninstall_component				Yes	Global	No
Com_uninstall_plugin				Yes	Both	No
Com_unlock_tables				Yes	Both	No
Com_update				Yes	Both	No
Com_update_multi				Yes	Both	No
Com_xa_commit				Yes	Both	No
Com_xa_end				Yes	Both	No
Com_xa_prepare				Yes	Both	No
Com_xa_recover				Yes	Both	No
Com_xa_rollback				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Com_xa_start				Yes	Both	No
completion_type	Yes	Yes	Yes		Both	Yes
Compression				Yes	Session	No
Compression_algorithm				Yes	Global	No
Compression_level				Yes	Global	No
concurrent_insert	Yes	Yes	Yes		Global	Yes
connect_timeout	Yes	Yes	Yes		Global	Yes
Connection_control_delay_generated				Yes	Global	No
connection_control_failed_connections_threshold	Yes	Yes	Yes		Global	Yes
connection_control_max_connection_delay	Yes	Yes	Yes		Global	Yes
connection_control_min_connection_delay	Yes	Yes	Yes		Global	Yes
Connection_errors_accept				Yes	Global	No
Connection_errors_internal				Yes	Global	No
Connection_errors_max_connections				Yes	Global	No
Connection_errors_peer_address				Yes	Global	No
Connection_errors_select				Yes	Global	No
Connection_errors_tcpwrap				Yes	Global	No
Connections				Yes	Global	No
console	Yes	Yes				
core-file	Yes	Yes				
core_file			Yes		Global	No
create_admin_listener_thread	Yes	Yes	Yes		Global	No
Created_tmp_disk_tables				Yes	Both	No
Created_tmp_files				Yes	Global	No
Created_tmp_tables				Yes	Both	No
cte_max_recursion_depth	Yes	Yes	Yes		Both	Yes
Current_tls_ca				Yes	Global	No
Current_tls_capath				Yes	Global	No
Current_tls_cert				Yes	Global	No
Current_tls_cipher				Yes	Global	No
Current_tls_ciphersuites				Yes	Global	No
Current_tls_crl				Yes	Global	No
Current_tls_crlpath				Yes	Global	No
Current_tls_key				Yes	Global	No
Current_tls_version				Yes	Global	No
daemon_memcached_enable_binlog	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_path	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
daemon_memcached_option	Yes	Yes	Yes		Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes		Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes		Global	No
daemonize	Yes	Yes				
datadir	Yes	Yes	Yes		Global	No
date_format			Yes		Global	No
datetime_format			Yes		Global	No
ddl-rewriter	Yes	Yes				
debug	Yes	Yes	Yes		Both	Ye
debug_sync			Yes		Session	Ye
debug-sync-timeout	Yes	Yes				
default_authentication_plugin	Yes	Yes	Yes		Global	No
default_collation_for_utf8mb4			Yes		Both	Ye
default_password_lifetime	Yes	Yes	Yes		Global	Ye
default_storage_engine	Yes	Yes	Yes		Both	Ye
default_table_encryption	Yes	Yes	Yes		Both	Ye
default-time-zone	Yes	Yes				
default_tmp_storage_engine	Yes	Yes	Yes		Both	Ye
default_week_format	Yes	Yes	Yes		Both	Ye
defaults-extra-file	Yes					
defaults-file	Yes					
defaults-group-suffix	Yes					
delay_key_write	Yes	Yes	Yes		Global	Ye
Delayed_errors				Yes	Global	No
delayed_insert_limit	Yes	Yes	Yes		Global	Ye
Delayed_insert_threads				Yes	Global	No
delayed_insert_timeout	Yes	Yes	Yes		Global	Ye
delayed_queue_size	Yes	Yes	Yes		Global	Ye
Delayed_writes				Yes	Global	No
des-key-file	Yes	Yes				
disabled_storage_engines	Yes	Yes	Yes		Global	No
disconnect_on_expired_password	Yes	Yes	Yes		Global	No
disconnect-slave-event-count	Yes	Yes				
div_precision_increment	Yes	Yes	Yes		Both	Ye
dragnet.log_error_filter_rules	Yes	Yes	Yes		Global	Ye
dragnet.Status				Yes	Global	No
early-plugin-load	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
end_markers_in_json	Yes	Yes	Yes		Both	Yes
enforce_gtid_consistency	Yes	Yes	Yes		Global	Yes
eq_range_index_dive_limit	Yes	Yes	Yes		Both	Yes
error_count			Yes		Session	No
event_scheduler	Yes	Yes	Yes		Global	Yes
exit-info	Yes	Yes				
expire_logs_days	Yes	Yes	Yes		Global	Yes
explicit_defaults_for_timestamp	Yes	Yes	Yes		Both	Yes
external-locking	Yes	Yes				
- *Variable*: skip_external_locking						
external_user			Yes		Session	No
federated	Yes	Yes				
Firewall_access_denied				Yes	Global	No
Firewall_access_granted				Yes	Global	No
Firewall_cached_entries				Yes	Global	No
flush	Yes	Yes	Yes		Global	Yes
Flush_commands				Yes	Global	No
flush_time	Yes	Yes	Yes		Global	Yes
foreign_key_checks			Yes		Both	Yes
ft_boolean_syntax	Yes	Yes	Yes		Global	Yes
ft_max_word_len	Yes	Yes	Yes		Global	No
ft_min_word_len	Yes	Yes	Yes		Global	No
ft_query_expansion_limit	Yes	Yes	Yes		Global	No
ft_stopword_file	Yes	Yes	Yes		Global	No
gdb	Yes	Yes				
general_log	Yes	Yes	Yes		Global	Yes
general_log_file	Yes	Yes	Yes		Global	Yes
generated_random_password_length	Yes	Yes	Yes		Both	Yes
group_concat_max_len	Yes	Yes	Yes		Both	Yes
group_replication_allow_local_disjoint_gtids_join	Yes	Yes	Yes		Global	Yes
group_replication_allow_local_lower_version_join	Yes	Yes	Yes		Global	Yes
group_replication_auto_increment_increment	Yes	Yes	Yes		Global	Yes
group_replication_autorejoin_tries	Yes	Yes	Yes		Global	Yes
group_replication_bootstrap_group	Yes	Yes	Yes		Global	Yes
group_replication_clone_threshold	Yes	Yes	Yes		Global	Yes
group_replication_communication_debug_options	Yes	Yes	Yes		Global	Yes
group_replication_communication_max_message_size	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
group_replication_components_stop_timeout	Yes	Yes	Yes		Global	Ye
group_replication_compression_threshold	Yes	Yes	Yes		Global	Ye
group_replication_consistency	Yes	Yes	Yes		Both	Ye
group_replication_enforce_update_everywhere_checks	Yes	Yes	Yes		Global	Ye
group_replication_exit_state_action	Yes	Yes	Yes		Global	Ye
group_replication_flow_control_applier_threshold	Yes	Yes	Yes		Global	Ye
group_replication_flow_control_certifier_threshold	Yes	Yes	Yes		Global	Ye
group_replication_flow_control_hold_percent	Yes	Yes	Yes		Global	Ye
group_replication_flow_control_max_commit_quota	Yes	Yes	Yes		Global	Ye
group_replication_flow_control_member_quota_percent	Yes	Yes	Yes		Global	Ye
group_replication_flow_control_min_quota	Yes	Yes	Yes		Global	Ye
group_replication_flow_control_min_recovery_quota	Yes	Yes	Yes		Global	Ye
group_replication_flow_control_mode	Yes	Yes	Yes		Global	Ye
group_replication_flow_control_period	Yes	Yes	Yes		Global	Ye
group_replication_flow_control_release_percent	Yes	Yes	Yes		Global	Ye
group_replication_force_members	Yes	Yes	Yes		Global	Ye
group_replication_group_name	Yes	Yes	Yes		Global	Ye
group_replication_group_seeds	Yes	Yes	Yes		Global	Ye
group_replication_gtid_assignment_block_size	Yes	Yes	Yes		Global	Ye
group_replication_ip_whitelist	Yes	Yes	Yes		Global	Ye
group_replication_local_address	Yes	Yes	Yes		Global	Ye
group_replication_member_expel_timeout	Yes	Yes	Yes		Global	Ye
group_replication_member_weight	Yes	Yes	Yes		Global	Ye
group_replication_message_cache_size	Yes	Yes	Yes		Global	Ye
group_replication_poll_spin_loops	Yes	Yes	Yes		Global	Ye
group_replication_primary_member				Yes	Global	No
group_replication_recovery_complete_at	Yes	Yes	Yes		Global	Ye
group_replication_recovery_compression_algorithms	Yes	Yes	Yes		Global	Ye
group_replication_recovery_get_public_key	Yes	Yes	Yes		Global	Ye
group_replication_recovery_public_key_path	Yes	Yes	Yes		Global	Ye
group_replication_recovery_reconnect_interval	Yes	Yes	Yes		Global	Ye
group_replication_recovery_retry_count	Yes	Yes	Yes		Global	Ye
group_replication_recovery_ssl_ca	Yes	Yes	Yes		Global	Ye
group_replication_recovery_ssl_capath	Yes	Yes	Yes		Global	Ye
group_replication_recovery_ssl_cert	Yes	Yes	Yes		Global	Ye
group_replication_recovery_ssl_cipher	Yes	Yes	Yes		Global	Ye
group_replication_recovery_ssl_crl	Yes	Yes	Yes		Global	Ye
group_replication_recovery_ssl_crlpath	Yes	Yes	Yes		Global	Ye

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
group_replication_recovery_ssl_key	Yes	Yes	Yes		Global	Yes
group_replication_recovery_ssl_verify_server_cert	Yes	Yes	Yes		Global	Yes
group_replication_recovery_use_ssl	Yes	Yes	Yes		Global	Yes
group_replication_recovery_zstd_compression_level	Yes	Yes	Yes		Global	Yes
group_replication_single_primary_mode	Yes	Yes	Yes		Global	Yes
group_replication_ssl_mode	Yes	Yes	Yes		Global	Yes
group_replication_start_on_boot	Yes	Yes	Yes		Global	Yes
group_replication_transaction_size_limit	Yes	Yes	Yes		Global	Yes
group_replication_unreachable_majority_timeout	Yes	Yes	Yes		Global	Yes
gtid_executed			Yes		Varies	No
gtid-executed-compression-period	Yes	Yes				
- *Variable*: gtid_executed_compression_period						
gtid_executed_compression_period			Yes		Global	Yes
gtid_mode	Yes	Yes	Yes		Global	Yes
gtid_next			Yes		Session	Yes
gtid_owned			Yes		Both	No
gtid_purged			Yes		Global	Yes
Handler_commit				Yes	Both	No
Handler_delete				Yes	Both	No
Handler_discover				Yes	Both	No
Handler_external_lock				Yes	Both	No
Handler_mrr_init				Yes	Both	No
Handler_prepare				Yes	Both	No
Handler_read_first				Yes	Both	No
Handler_read_key				Yes	Both	No
Handler_read_last				Yes	Both	No
Handler_read_next				Yes	Both	No
Handler_read_prev				Yes	Both	No
Handler_read_rnd				Yes	Both	No
Handler_read_rnd_next				Yes	Both	No
Handler_rollback				Yes	Both	No
Handler_savepoint				Yes	Both	No
Handler_savepoint_rollback				Yes	Both	No
Handler_update				Yes	Both	No
Handler_write				Yes	Both	No
have_compress			Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
have_crypt			Yes		Global	No
have_dynamic_loading			Yes		Global	No
have_geometry			Yes		Global	No
have_openssl			Yes		Global	No
have_profiling			Yes		Global	No
have_query_cache			Yes		Global	No
have_rtree_keys			Yes		Global	No
have_ssl			Yes		Global	No
have_statement_timeout			Yes		Global	No
have_symlink			Yes		Global	No
help	Yes	Yes				
histogram_generation_max_mem_size	Yes	Yes	Yes		Both	Ye
host_cache_size	Yes	Yes	Yes		Global	Ye
hostname			Yes		Global	No
identity			Yes		Session	Ye
ignore_builtin_innodb	Yes	Yes	Yes		Global	No
immediate_server_version			Yes		Session	Ye
information_schema_stats_expiry	Yes	Yes	Yes		Both	Ye
init_connect	Yes	Yes	Yes		Global	Ye
init_file	Yes	Yes	Yes		Global	No
init_slave	Yes	Yes	Yes		Global	Ye
initialize	Yes	Yes				
initialize-insecure	Yes	Yes				
innodb	Yes	Yes				
innodb_adaptive_flushing	Yes	Yes	Yes		Global	Ye
innodb_adaptive_flushing_lwm	Yes	Yes	Yes		Global	Ye
innodb_adaptive_hash_index	Yes	Yes	Yes		Global	Ye
innodb_adaptive_hash_index_parts	Yes	Yes	Yes		Global	No
innodb_adaptive_max_sleep_delay	Yes	Yes	Yes		Global	Ye
innodb_api_bk_commit_interval	Yes	Yes	Yes		Global	Ye
innodb_api_disable_rowlock	Yes	Yes	Yes		Global	No
innodb_api_enable_binlog	Yes	Yes	Yes		Global	No
innodb_api_enable_mdl	Yes	Yes	Yes		Global	No
innodb_api_trx_level	Yes	Yes	Yes		Global	Ye
innodb_autoextend_increment	Yes	Yes	Yes		Global	Ye
innodb_autoinc_lock_mode	Yes	Yes	Yes		Global	No
Innodb_available_undo_logs				Yes	Global	No
innodb_background_drop_list_empty	Yes	Yes	Yes		Global	Ye

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Innodb_buffer_pool_bytes_data				Yes	Global	No
Innodb_buffer_pool_bytes_dirty				Yes	Global	No
innodb_buffer_pool_chunk_size	Yes	Yes	Yes		Global	No
innodb_buffer_pool_debug	Yes	Yes	Yes		Global	No
innodb_buffer_pool_dump_at_shutdown	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_now	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_pct	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_dump_status				Yes	Global	No
innodb_buffer_pool_filename	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_in_core_file	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_instances	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_abort	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_load_at_startup	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_now	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_load_status				Yes	Global	No
Innodb_buffer_pool_pages_data				Yes	Global	No
Innodb_buffer_pool_pages_dirty				Yes	Global	No
Innodb_buffer_pool_pages_flushed				Yes	Global	No
Innodb_buffer_pool_pages_free				Yes	Global	No
Innodb_buffer_pool_pages_latched				Yes	Global	No
Innodb_buffer_pool_pages_misc				Yes	Global	No
Innodb_buffer_pool_pages_total				Yes	Global	No
Innodb_buffer_pool_read_ahead				Yes	Global	No
Innodb_buffer_pool_read_ahead_evicted				Yes	Global	No
Innodb_buffer_pool_read_ahead_rnd				Yes	Global	No
Innodb_buffer_pool_read_requests				Yes	Global	No
Innodb_buffer_pool_reads				Yes	Global	No
Innodb_buffer_pool_resize_status				Yes	Global	No
innodb_buffer_pool_size	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_wait_free				Yes	Global	No
Innodb_buffer_pool_write_requests				Yes	Global	No
innodb_change_buffer_max_size	Yes	Yes	Yes		Global	Yes
innodb_change_buffering	Yes	Yes	Yes		Global	Yes
innodb_change_buffering_debug	Yes	Yes	Yes		Global	Yes
innodb_checkpoint_disabled	Yes	Yes	Yes		Global	Yes
innodb_checksum_algorithm	Yes	Yes	Yes		Global	Yes
innodb_cmp_per_index_enabled	Yes	Yes	Yes		Global	Yes
innodb_commit_concurrency	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
innodb_compress_debug	Yes	Yes	Yes		Global	Ye
innodb_compression_failure_threshold_pct	Yes	Yes	Yes		Global	Ye
innodb_compression_level	Yes	Yes	Yes		Global	Ye
innodb_compression_pad_pct_max	Yes	Yes	Yes		Global	Ye
innodb_concurrency_tickets	Yes	Yes	Yes		Global	Ye
innodb_data_file_path	Yes	Yes	Yes		Global	No
Innodb_data_fsyncs				Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes		Global	No
Innodb_data_pending_fsyncs				Yes	Global	No
Innodb_data_pending_reads				Yes	Global	No
Innodb_data_pending_writes				Yes	Global	No
Innodb_data_read				Yes	Global	No
Innodb_data_reads				Yes	Global	No
Innodb_data_writes				Yes	Global	No
Innodb_data_written				Yes	Global	No
Innodb_dblwr_pages_written				Yes	Global	No
Innodb_dblwr_writes				Yes	Global	No
innodb_ddl_log_crash_reset_debug	Yes	Yes	Yes		Global	Ye
innodb_deadlock_detect	Yes	Yes	Yes		Global	Ye
innodb_dedicated_server	Yes	Yes	Yes		Global	No
innodb_default_row_format	Yes	Yes	Yes		Global	Ye
innodb_directories	Yes	Yes	Yes		Global	No
innodb_disable_sort_file_cache	Yes	Yes	Yes		Global	Ye
innodb_doublewrite	Yes	Yes	Yes		Global	No
innodb_fast_shutdown	Yes	Yes	Yes		Global	Ye
innodb_fil_make_page_dirty_debug	Yes	Yes	Yes		Global	Ye
innodb_file_per_table	Yes	Yes	Yes		Global	Ye
innodb_fill_factor	Yes	Yes	Yes		Global	Ye
innodb_flush_log_at_timeout	Yes	Yes	Yes		Global	Ye
innodb_flush_log_at_trx_commit	Yes	Yes	Yes		Global	Ye
innodb_flush_method	Yes	Yes	Yes		Global	No
innodb_flush_neighbors	Yes	Yes	Yes		Global	Ye
innodb_flush_sync	Yes	Yes	Yes		Global	Ye
innodb_flushing_avg_loops	Yes	Yes	Yes		Global	Ye
innodb_force_load_corrupted	Yes	Yes	Yes		Global	No
innodb_force_recovery	Yes	Yes	Yes		Global	No
innodb_fsync_threshold	Yes	Yes	Yes		Global	Ye
innodb_ft_aux_table			Yes		Global	Ye

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
innodb_ft_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_enable_diag_print	Yes	Yes	Yes		Global	Yes
innodb_ft_enable_stopword	Yes	Yes	Yes		Both	Yes
innodb_ft_max_token_size	Yes	Yes	Yes		Global	No
innodb_ft_min_token_size	Yes	Yes	Yes		Global	No
innodb_ft_num_word_optimize	Yes	Yes	Yes		Global	Yes
innodb_ft_result_cache_limit	Yes	Yes	Yes		Global	Yes
innodb_ft_server_stopword_table	Yes	Yes	Yes		Global	Yes
innodb_ft_sort_pll_degree	Yes	Yes	Yes		Global	No
innodb_ft_total_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_user_stopword_table	Yes	Yes	Yes		Both	Yes
Innodb_have_atomic_builtins				Yes	Global	No
innodb_idle_flush_pct	Yes	Yes	Yes		Global	Yes
innodb_io_capacity	Yes	Yes	Yes		Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes		Global	Yes
innodb_limit_optimistic_insert_debug	Yes	Yes	Yes		Global	Yes
innodb_lock_wait_timeout	Yes	Yes	Yes		Both	Yes
innodb_log_buffer_size	Yes	Yes	Yes		Global	Varies
innodb_log_checkpoint_fuzzy_now	Yes	Yes	Yes		Global	Yes
innodb_log_checkpoint_now	Yes	Yes	Yes		Global	Yes
innodb_log_checksums	Yes	Yes	Yes		Global	Yes
innodb_log_compressed_pages	Yes	Yes	Yes		Global	Yes
innodb_log_file_size	Yes	Yes	Yes		Global	No
innodb_log_files_in_group	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir	Yes	Yes	Yes		Global	No
innodb_log_spin_cpu_abs_lwm	Yes	Yes	Yes		Global	Yes
innodb_log_spin_cpu_pct_hwm	Yes	Yes	Yes		Global	Yes
innodb_log_wait_for_flush_spin_hwm	Yes	Yes	Yes		Global	Yes
Innodb_log_waits				Yes	Global	No
innodb_log_write_ahead_size	Yes	Yes	Yes		Global	Yes
Innodb_log_write_requests				Yes	Global	No
Innodb_log_writes				Yes	Global	No
innodb_lru_scan_depth	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct_lwm	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag_delay	Yes	Yes	Yes		Global	Yes
innodb_max_undo_log_size	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
innodb_merge_threshold	Yes	set_all_debug	Yes	Yes		Global	Ye
innodb_monitor_disable	Yes	Yes	Yes		Global	Ye
innodb_monitor_enable	Yes	Yes	Yes		Global	Ye
innodb_monitor_reset	Yes	Yes	Yes		Global	Ye
innodb_monitor_reset_all	Yes	Yes	Yes		Global	Ye
Innodb_num_open_files				Yes	Global	No
innodb_numa_interleave	Yes	Yes	Yes		Global	No
innodb_old_blocks_pct	Yes	Yes	Yes		Global	Ye
innodb_old_blocks_time	Yes	Yes	Yes		Global	Ye
innodb_online_alter_log_max_size	Yes	Yes	Yes		Global	Ye
innodb_open_files	Yes	Yes	Yes		Global	No
innodb_optimize_fulltext_only	Yes	Yes	Yes		Global	Ye
Innodb_os_log_fsyncs				Yes	Global	No
Innodb_os_log_pending_fsyncs				Yes	Global	No
Innodb_os_log_pending_writes				Yes	Global	No
Innodb_os_log_written				Yes	Global	No
innodb_page_cleaners	Yes	Yes	Yes		Global	No
Innodb_page_size				Yes	Global	No
innodb_page_size	Yes	Yes	Yes		Global	No
Innodb_pages_created				Yes	Global	No
Innodb_pages_read				Yes	Global	No
Innodb_pages_written				Yes	Global	No
innodb_parallel_read_threads	Yes	Yes	Yes		Session	Ye
innodb_print_all_deadlocks	Yes	Yes	Yes		Global	Ye
innodb_print_ddl_logs	Yes	Yes	Yes		Global	Ye
innodb_purge_batch_size	Yes	Yes	Yes		Global	Ye
innodb_purge_rseg_truncate_frequency	Yes	Yes	Yes		Global	Ye
innodb_purge_threads	Yes	Yes	Yes		Global	No
innodb_random_read_ahead	Yes	Yes	Yes		Global	Ye
innodb_read_ahead_threshold	Yes	Yes	Yes		Global	Ye
innodb_read_io_threads	Yes	Yes	Yes		Global	No
innodb_read_only	Yes	Yes	Yes		Global	No
innodb_redo_log_archive_dirs	Yes	Yes	Yes		Global	Ye
innodb_redo_log_encrypt	Yes	Yes	Yes		Global	Ye
innodb_replication_delay	Yes	Yes	Yes		Global	Ye
innodb_rollback_on_timeout	Yes	Yes	Yes		Global	No
innodb_rollback_segments	Yes	Yes	Yes		Global	Ye
Innodb_row_lock_current_waits				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Innodb_row_lock_time				Yes	Global	No
Innodb_row_lock_time_avg				Yes	Global	No
Innodb_row_lock_time_max				Yes	Global	No
Innodb_row_lock_waits				Yes	Global	No
Innodb_rows_deleted				Yes	Global	No
Innodb_rows_inserted				Yes	Global	No
Innodb_rows_read				Yes	Global	No
Innodb_rows_updated				Yes	Global	No
innodb_saved_page_number_debug	Yes	Yes	Yes		Global	Yes
innodb_scan_directories	Yes	Yes	Yes		Global	No
innodb_sort_buffer_size	Yes	Yes	Yes		Global	No
innodb_spin_wait_delay	Yes	Yes	Yes		Global	Yes
innodb_spin_wait_pause_multiplier	Yes	Yes	Yes		Global	Yes
innodb_stats_auto_recalc	Yes	Yes	Yes		Global	Yes
innodb_stats_include_delete_marked	Yes	Yes	Yes		Global	Yes
innodb_stats_method	Yes	Yes	Yes		Global	Yes
innodb_stats_on_metadata	Yes	Yes	Yes		Global	Yes
innodb_stats_persistent	Yes	Yes	Yes		Global	Yes
innodb_stats_persistent_sample_pages	Yes	Yes	Yes		Global	Yes
innodb_stats_transient_sample_pages	Yes	Yes	Yes		Global	Yes
innodb-status-file	Yes	Yes				
innodb_status_output	Yes	Yes	Yes		Global	Yes
innodb_status_output_locks	Yes	Yes	Yes		Global	Yes
innodb_strict_mode	Yes	Yes	Yes		Both	Yes
innodb_sync_array_size	Yes	Yes	Yes		Global	No
innodb_sync_debug	Yes	Yes	Yes		Global	No
innodb_sync_spin_loops	Yes	Yes	Yes		Global	Yes
Innodb_system_rows_deleted				Yes	Global	No
Innodb_system_rows_inserted				Yes	Global	No
Innodb_system_rows_read				Yes	Global	No
innodb_table_locks	Yes	Yes	Yes		Both	Yes
innodb_temp_data_file_path	Yes	Yes	Yes		Global	No
innodb_temp_tablespaces_dir	Yes	Yes	Yes		Global	No
innodb_thread_concurrency	Yes	Yes	Yes		Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_tmpdir	Yes	Yes	Yes		Both	Yes
Innodb_truncated_status_writes				Yes	Global	No
innodb_trx_purge_view_update_only_debug	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
innodb_trx_rseg_n_slots_debug	Yes	Yes	Yes		Global	Ye
innodb_undo_directory	Yes	Yes	Yes		Global	No
innodb_undo_log_encrypt	Yes	Yes	Yes		Global	Ye
innodb_undo_log_truncate	Yes	Yes	Yes		Global	Ye
innodb_undo_logs	Yes	Yes	Yes		Global	Ye
innodb_undo_tablespaces	Yes	Yes	Yes		Global	Va
innodb_use_native_aio	Yes	Yes	Yes		Global	No
innodb_version			Yes		Global	No
innodb_write_io_threads	Yes	Yes	Yes		Global	No
insert_id			Yes		Session	Ye
install	Yes					
install-manual	Yes					
interactive_timeout	Yes	Yes	Yes		Both	Ye
internal_tmp_disk_storage_engine	Yes	Yes	Yes		Global	Ye
internal_tmp_mem_storage_engine	Yes	Yes	Yes		Both	Ye
join_buffer_size	Yes	Yes	Yes		Both	Ye
keep_files_on_create	Yes	Yes	Yes		Both	Ye
Key_blocks_not_flushed				Yes	Global	No
Key_blocks_unused				Yes	Global	No
Key_blocks_used				Yes	Global	No
key_buffer_size	Yes	Yes	Yes		Global	Ye
key_cache_age_threshold	Yes	Yes	Yes		Global	Ye
key_cache_block_size	Yes	Yes	Yes		Global	Ye
key_cache_division_limit	Yes	Yes	Yes		Global	Ye
Key_read_requests				Yes	Global	No
Key_reads				Yes	Global	No
Key_write_requests				Yes	Global	No
Key_writes				Yes	Global	No
keyring_aws_cmk_id	Yes	Yes	Yes		Global	Ye
keyring_aws_conf_file	Yes	Yes	Yes		Global	No
keyring_aws_data_file	Yes	Yes	Yes		Global	No
keyring_aws_region	Yes	Yes	Yes		Global	Ye
keyring_encrypted_file_data	Yes	Yes	Yes		Global	Ye
keyring_encrypted_file_password	Yes	Yes	Yes		Global	Ye
keyring_file_data	Yes	Yes	Yes		Global	Ye
keyring-migration-destination	Yes	Yes				
keyring-migration-host	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
keyring-migration-password	Yes	Yes				
keyring-migration-port	Yes	Yes				
keyring-migration-socket	Yes	Yes				
keyring-migration-source	Yes	Yes				
keyring-migration-user	Yes	Yes				
keyring_okv_conf_dir	Yes	Yes	Yes		Global	Yes
keyring_operations			Yes		Global	Yes
language	Yes	Yes	Yes		Global	No
large_files_support			Yes		Global	No
large_page_size			Yes		Global	No
large-pages	Yes	Yes	Yes		Global	No
last_insert_id			Yes		Session	Yes
Last_query_cost				Yes	Session	No
Last_query_partial_plans				Yes	Session	No
lc-messages	Yes	Yes	Yes		Both	Yes
lc-messages-dir	Yes	Yes	Yes		Global	No
lc_time_names	Yes	Yes	Yes		Both	Yes
license			Yes		Global	No
local_infile	Yes	Yes	Yes		Global	Yes
local-service	Yes					
lock_order	Yes	Yes	Yes		Global	No
lock_order_debug_loop	Yes	Yes	Yes		Global	No
lock_order_debug_missing_arc	Yes	Yes	Yes		Global	No
lock_order_debug_missing_key	Yes	Yes	Yes		Global	No
lock_order_debug_missing_unlock	Yes	Yes	Yes		Global	No
lock_order_dependencies	Yes	Yes	Yes		Global	No
lock_order_extra_dependencies	Yes	Yes	Yes		Global	No
lock_order_output_directory	Yes	Yes	Yes		Global	No
lock_order_print_txt	Yes	Yes	Yes		Global	No
lock_order_trace_loop	Yes	Yes	Yes		Global	No
lock_order_trace_missing_arc	Yes	Yes	Yes		Global	No
lock_order_trace_missing_key	Yes	Yes	Yes		Global	No
lock_order_trace_missing_unlock	Yes	Yes	Yes		Global	No
lock_wait_timeout	Yes	Yes	Yes		Both	Yes
Locked_connects				Yes	Global	No
locked_in_memory			Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
log-bin	Yes	Yes				
log_bin			Yes		Global	No
log_bin_basename			Yes		Global	No
log-bin-index	Yes	Yes	Yes		Global	No
log_bin_trust_function_creators	Yes	Yes	Yes		Global	Ye
log_bin_use_v1_row_events	Yes	Yes	Yes		Global	No
log_builtin_as_identified_by_password	Yes	Yes	Yes		Global	Ye
log-error	Yes	Yes	Yes		Global	No
log_error_filter_rules	Yes	Yes	Yes		Global	Ye
log_error_services	Yes	Yes	Yes		Global	Ye
log_error_suppression_list	Yes	Yes	Yes		Global	Ye
log_error_verbosity	Yes	Yes	Yes		Global	Ye
log-isam	Yes	Yes				
log_output	Yes	Yes	Yes		Global	Ye
log_queries_not_using_indexes	Yes	Yes	Yes		Global	Ye
log-raw	Yes	Yes				
log-short-format	Yes	Yes				
log_slave_updates	Yes	Yes	Yes		Global	No
log_slow_admin_statements	Yes	Yes	Yes		Global	Ye
log_slow_extra	Yes	Yes	Yes		Global	Ye
log_slow_slave_statements	Yes	Yes	Yes		Global	Ye
log_statements_unsafe_for_binlog	Yes	Yes	Yes		Global	Ye
log_syslog	Yes	Yes	Yes		Global	Ye
log_syslog_facility	Yes	Yes	Yes		Global	Ye
log_syslog_include_pid	Yes	Yes	Yes		Global	Ye
log_syslog_tag	Yes	Yes	Yes		Global	Ye
log-tc	Yes	Yes				
log-tc-size	Yes	Yes				
log_throttle_queries_not_using_indexes	Yes	Yes	Yes		Global	Ye
log_timestamps	Yes	Yes	Yes		Global	Ye
log-warnings	Yes	Yes	Yes		Global	Ye
long_query_time	Yes	Yes	Yes		Both	Ye
low_priority_updates	Yes	Yes	Yes		Both	Ye
lower_case_file_system			Yes		Global	No
lower_case_table_names	Yes	Yes	Yes		Global	No
mandatory_roles	Yes	Yes	Yes		Global	Ye
master-info-file	Yes	Yes				
master-info-repository	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
- *Variable*: master_info_repository						
master_info_repository	Yes	Yes	Yes		Global	Yes
master-retry-count	Yes	Yes				
master-verify-checksum	Yes	Yes				
- *Variable*: master_verify_checksum						
master_verify_checksum	Yes	Yes	Yes		Global	Yes
max_allowed_packet	Yes	Yes	Yes		Both	Yes
max_binlog_cache_size	Yes	Yes	Yes		Global	Yes
max-binlog-dump-events	Yes	Yes				
max_binlog_size	Yes	Yes	Yes		Global	Yes
max_binlog_stmt_cache_size	Yes	Yes	Yes		Global	Yes
max_connect_errors	Yes	Yes	Yes		Global	Yes
max_connections	Yes	Yes	Yes		Global	Yes
max_delayed_threads	Yes	Yes	Yes		Both	Yes
max_digest_length	Yes	Yes	Yes		Global	No
max_error_count	Yes	Yes	Yes		Both	Yes
max_execution_time	Yes	Yes	Yes		Both	Yes
Max_execution_time_exceeded				Yes	Both	No
Max_execution_time_set				Yes	Both	No
Max_execution_time_set_failed				Yes	Both	No
max_heap_table_size	Yes	Yes	Yes		Both	Yes
max_insert_delayed_threads			Yes		Both	Yes
max_join_size	Yes	Yes	Yes		Both	Yes
max_length_for_sort_data	Yes	Yes	Yes		Both	Yes
max_points_in_geometry	Yes	Yes	Yes		Both	Yes
max_prepared_stmt_count	Yes	Yes	Yes		Global	Yes
max_relay_log_size	Yes	Yes	Yes		Global	Yes
max_seeks_for_key	Yes	Yes	Yes		Both	Yes
max_sort_length	Yes	Yes	Yes		Both	Yes
max_sp_recursion_depth	Yes	Yes	Yes		Both	Yes
max_tmp_tables			Yes		Both	Yes
Max_used_connections				Yes	Global	No
Max_used_connections_time				Yes	Global	No
max_user_connections	Yes	Yes	Yes		Both	Yes
max_write_lock_count	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
mecab_charset				Yes	Global	No
mecab_rc_file	Yes	Yes	Yes		Global	No
memlock	Yes	Yes				
- *Variable*: locked_in_memory						
metadata_locks_cache_size	Yes	Yes	Yes		Global	No
metadata_locks_hash_instances	Yes	Yes	Yes		Global	No
min_examined_row_limit	Yes	Yes	Yes		Both	Ye
multi_range_count	Yes	Yes	Yes		Both	Ye
myisam-block-size	Yes	Yes				
myisam_data_pointer_size	Yes	Yes	Yes		Global	Ye
myisam_max_sort_file_size	Yes	Yes	Yes		Global	Ye
myisam_mmap_size	Yes	Yes	Yes		Global	No
myisam_recover_options	Yes	Yes	Yes		Global	No
myisam_repair_threads	Yes	Yes	Yes		Both	Ye
myisam_sort_buffer_size	Yes	Yes	Yes		Both	Ye
myisam_stats_method	Yes	Yes	Yes		Both	Ye
myisam_use_mmap	Yes	Yes	Yes		Global	Ye
mysql_firewall_mode	Yes	Yes	Yes		Global	Ye
mysql_firewall_trace	Yes	Yes	Yes		Global	Ye
mysql_native_password_proxy_users	Yes	Yes	Yes		Global	Ye
mysqlx	Yes	Yes				
Mysqlx_aborted_clients				Yes	Global	No
Mysqlx_address				Yes	Global	No
mysqlx_bind_address	Yes	Yes	Yes		Global	No
Mysqlx_bytes_received				Yes	Both	No
Mysqlx_bytes_sent				Yes	Both	No
mysqlx_connect_timeout	Yes	Yes	Yes		Global	Ye
Mysqlx_connection_accept_errors				Yes	Both	No
Mysqlx_connection_errors				Yes	Both	No
Mysqlx_connections_accepted				Yes	Global	No
Mysqlx_connections_closed				Yes	Global	No
Mysqlx_connections_rejected				Yes	Global	No
Mysqlx_crud_create_view				Yes	Both	No
Mysqlx_crud_delete				Yes	Both	No
Mysqlx_crud_drop_view				Yes	Both	No
Mysqlx_crud_find				Yes	Both	No
Mysqlx_crud_insert				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Mysqlx_crud_modify_view				Yes	Both	No
Mysqlx_crud_update				Yes	Both	No
Mysqlx_cursor_close				Yes	Both	No
Mysqlx_cursor_fetch				Yes	Both	No
Mysqlx_cursor_open				Yes	Both	No
mysqlx_document_id_unique_prefix	Yes	Yes	Yes		Global	Yes
mysqlx_enable_hello_notice	Yes	Yes	Yes		Global	Yes
Mysqlx_errors_sent				Yes	Both	No
Mysqlx_errors_unknown_message_type				Yes	Both	No
Mysqlx_expect_close				Yes	Both	No
Mysqlx_expect_open				Yes	Both	No
mysqlx_idle_worker_thread_timeout	Yes	Yes	Yes		Global	Yes
Mysqlx_init_error				Yes	Both	No
mysqlx_interactive_timeout	Yes	Yes	Yes		Global	Yes
mysqlx_max_allowed_packet	Yes	Yes	Yes		Global	Yes
mysqlx_max_connections	Yes	Yes	Yes		Global	Yes
mysqlx_min_worker_threads	Yes	Yes	Yes		Global	Yes
Mysqlx_notice_global_sent				Yes	Both	No
Mysqlx_notice_other_sent				Yes	Both	No
Mysqlx_notice_warning_sent				Yes	Both	No
Mysqlx_notified_by_group_replication				Yes	Both	No
Mysqlx_port				Yes	Global	No
mysqlx_port	Yes	Yes	Yes		Global	No
mysqlx_port_open_timeout	Yes	Yes	Yes		Global	No
Mysqlx_prep_deallocate				Yes	Both	No
Mysqlx_prep_execute				Yes	Both	No
Mysqlx_prep_prepare				Yes	Both	No
mysqlx_read_timeout	Yes	Yes	Yes		Session	Yes
Mysqlx_rows_sent				Yes	Both	No
Mysqlx_sessions				Yes	Global	No
Mysqlx_sessions_accepted				Yes	Global	No
Mysqlx_sessions_closed				Yes	Global	No
Mysqlx_sessions_fatal_error				Yes	Global	No
Mysqlx_sessions_killed				Yes	Global	No
Mysqlx_sessions_rejected				Yes	Global	No
Mysqlx_socket				Yes	Global	No
mysqlx_socket	Yes	Yes	Yes		Global	No
Mysqlx_ssl_accept_renegotiates				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Mysqlx_ssl_accepts				Yes	Global	No
Mysqlx_ssl_active				Yes	Both	No
mysqlx_ssl_ca	Yes	Yes	Yes		Global	No
mysqlx_ssl_capath	Yes	Yes	Yes		Global	No
mysqlx_ssl_cert	Yes	Yes	Yes		Global	No
Mysqlx_ssl_cipher				Yes	Both	No
mysqlx_ssl_cipher	Yes	Yes	Yes		Global	No
Mysqlx_ssl_cipher_list				Yes	Both	No
mysqlx_ssl_crl	Yes	Yes	Yes		Global	No
mysqlx_ssl_crlpath	Yes	Yes	Yes		Global	No
Mysqlx_ssl_ctx_verify_depth				Yes	Both	No
Mysqlx_ssl_ctx_verify_mode				Yes	Both	No
Mysqlx_ssl_finished_accepts				Yes	Global	No
mysqlx_ssl_key	Yes	Yes	Yes		Global	No
Mysqlx_ssl_server_not_after				Yes	Global	No
Mysqlx_ssl_server_not_before				Yes	Global	No
Mysqlx_ssl_verify_depth				Yes	Global	No
Mysqlx_ssl_verify_mode				Yes	Global	No
Mysqlx_ssl_version				Yes	Both	No
Mysqlx_stmt_create_collection				Yes	Both	No
Mysqlx_stmt_create_collection_index				Yes	Both	No
Mysqlx_stmt_disable_notices				Yes	Both	No
Mysqlx_stmt_drop_collection				Yes	Both	No
Mysqlx_stmt_drop_collection_index				Yes	Both	No
Mysqlx_stmt_enable_notices				Yes	Both	No
Mysqlx_stmt_ensure_collection				Yes	Both	No
Mysqlx_stmt_execute_mysqlx				Yes	Both	No
Mysqlx_stmt_execute_sql				Yes	Both	No
Mysqlx_stmt_execute_xplugin				Yes	Both	No
Mysqlx_stmt_kill_client				Yes	Both	No
Mysqlx_stmt_list_clients				Yes	Both	No
Mysqlx_stmt_list_notices				Yes	Both	No
Mysqlx_stmt_list_objects				Yes	Both	No
Mysqlx_stmt_ping				Yes	Both	No
mysqlx_wait_timeout	Yes	Yes	Yes		Session	Ye
Mysqlx_worker_threads				Yes	Global	No
Mysqlx_worker_threads_active				Yes	Global	No
mysqlx_write_timeout	Yes	Yes	Yes		Session	Ye

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
named_pipe	Yes	Yes	Yes		Global	No
named_pipe_full_access_group	Yes	Yes	Yes		Global	No
ndb-allow-copying-alter-table	Yes	Yes	Yes		Both	Yes
Ndb_api_bytes_received_count				Yes	Global	No
Ndb_api_bytes_received_count_session				Yes	Session	No
Ndb_api_bytes_received_count_slave				Yes	Global	No
Ndb_api_bytes_sent_count				Yes	Global	No
Ndb_api_bytes_sent_count_session				Yes	Session	No
Ndb_api_bytes_sent_count_slave				Yes	Global	No
Ndb_api_event_bytes_count				Yes	Global	No
Ndb_api_event_bytes_count_injector				Yes	Global	No
Ndb_api_event_data_count				Yes	Global	No
Ndb_api_event_data_count_injector				Yes	Global	No
Ndb_api_event_nondata_count				Yes	Global	No
Ndb_api_event_nondata_count_injector				Yes	Global	No
Ndb_api_pk_op_count				Yes	Global	No
Ndb_api_pk_op_count_session				Yes	Session	No
Ndb_api_pk_op_count_slave				Yes	Global	No
Ndb_api_pruned_scan_count				Yes	Global	No
Ndb_api_pruned_scan_count_session				Yes	Session	No
Ndb_api_pruned_scan_count_slave				Yes	Global	No
Ndb_api_range_scan_count				Yes	Global	No
Ndb_api_range_scan_count_session				Yes	Session	No
Ndb_api_range_scan_count_slave				Yes	Global	No
Ndb_api_read_row_count				Yes	Global	No
Ndb_api_read_row_count_session				Yes	Session	No
Ndb_api_read_row_count_slave				Yes	Global	No
Ndb_api_scan_batch_count				Yes	Global	No
Ndb_api_scan_batch_count_session				Yes	Session	No
Ndb_api_scan_batch_count_slave				Yes	Global	No
Ndb_api_table_scan_count				Yes	Global	No
Ndb_api_table_scan_count_session				Yes	Session	No
Ndb_api_table_scan_count_slave				Yes	Global	No
Ndb_api_trans_abort_count				Yes	Global	No
Ndb_api_trans_abort_count_session				Yes	Session	No
Ndb_api_trans_abort_count_slave				Yes	Global	No
Ndb_api_trans_close_count				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Ndb_api_trans_close_count_session				Yes	Session	No
Ndb_api_trans_close_count_slave				Yes	Global	No
Ndb_api_trans_commit_count				Yes	Global	No
Ndb_api_trans_commit_count_session				Yes	Session	No
Ndb_api_trans_commit_count_slave				Yes	Global	No
Ndb_api_trans_local_read_row_count				Yes	Global	No
Ndb_api_trans_local_read_row_count_session				Yes	Session	No
Ndb_api_trans_local_read_row_count_slave				Yes	Global	No
Ndb_api_trans_start_count				Yes	Global	No
Ndb_api_trans_start_count_session				Yes	Session	No
Ndb_api_trans_start_count_slave				Yes	Global	No
Ndb_api_uk_op_count				Yes	Global	No
Ndb_api_uk_op_count_session				Yes	Session	No
Ndb_api_uk_op_count_slave				Yes	Global	No
Ndb_api_wait_exec_complete_count				Yes	Global	No
Ndb_api_wait_exec_complete_count_session				Yes	Session	No
Ndb_api_wait_exec_complete_count_slave				Yes	Global	No
Ndb_api_wait_meta_request_count				Yes	Global	No
Ndb_api_wait_meta_request_count_session				Yes	Session	No
Ndb_api_wait_meta_request_count_slave				Yes	Global	No
Ndb_api_wait_nanos_count				Yes	Global	No
Ndb_api_wait_nanos_count_session				Yes	Session	No
Ndb_api_wait_nanos_count_slave				Yes	Global	No
Ndb_api_wait_scan_result_count				Yes	Global	No
Ndb_api_wait_scan_result_count_session				Yes	Session	No
Ndb_api_wait_scan_result_count_slave				Yes	Global	No
ndb_autoincrement_prefetch_sz	Yes	Yes	Yes		Both	Ye
ndb-batch-size	Yes	Yes			Global	No
- *Variable*: ndb_batch_size			Yes		Global	No
ndb-blob-read-batch-bytes	Yes	Yes			Both	Ye
- *Variable*: ndb_blob_read_batch_bytes			Yes		Both	Ye
ndb-blob-write-batch-bytes	Yes	Yes			Both	Ye
- *Variable*: ndb_blob_write_batch_bytes			Yes		Both	Ye
ndb_cache_check_time	Yes	Yes	Yes		Global	Ye

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
ndb_clear_apply_status	Yes		Yes		Global	Yes
ndb-cluster-connection-pool	Yes	Yes	Yes		Global	No
ndb-cluster-connection-pool-nodeids	Yes	Yes	Yes		Global	No
Ndb_cluster_node_id				Yes	Global	No
Ndb_config_from_host				Yes	Both	No
Ndb_config_from_port				Yes	Both	No
Ndb_conflict_fn_epoch				Yes	Global	No
Ndb_conflict_fn_epoch_trans				Yes	Global	No
Ndb_conflict_fn_epoch2				Yes	Global	No
Ndb_conflict_fn_epoch2_trans				Yes	Global	No
Ndb_conflict_fn_max				Yes	Global	No
Ndb_conflict_fn_old				Yes	Global	No
Ndb_conflict_last_conflict_epoch			Yes		Global	No
Ndb_conflict_last_stable_epoch				Yes	Global	No
Ndb_conflict_reflected_op_discard_count				Yes	Global	No
Ndb_conflict_reflected_op_prepare_count				Yes	Global	No
Ndb_conflict_refresh_op_count				Yes	Global	No
Ndb_conflict_trans_conflict_commit_count				Yes	Global	No
Ndb_conflict_trans_detect_iter_count				Yes	Global	No
Ndb_conflict_trans_reject_count				Yes	Global	No
Ndb_conflict_trans_row_conflict_count				Yes	Global	No
Ndb_conflict_trans_row_reject_count				Yes	Global	No
ndb-connectstring	Yes	Yes				
ndb_data_node_neighbour	Yes	Yes	Yes		Global	Yes
ndb_dbg_check_shares	Yes	Yes	Yes		Both	Yes
ndb-default-column-format	Yes	Yes			Global	Yes
- *Variable*: ndb_default_column_format			Yes		Global	Yes
ndb_default_column_format	Yes	Yes	Yes		Global	Yes
ndb-deferred-constraints	Yes	Yes			Both	Yes
- *Variable*: ndb_deferred_constraints			Yes		Both	Yes
ndb_deferred_constraints	Yes	Yes	Yes		Both	Yes
ndb-distribution	Yes	Yes			Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
- *Variable*: ndb_distribution			Yes		Global	Ye
ndb_distribution	Yes	Yes	Yes		Global	Ye
Ndb_epoch_delete_delete_count				Yes	Global	No
ndb_eventbuffer_free_percent	Yes	Yes	Yes		Global	Ye
ndb_eventbuffer_max_alloc	Yes	Yes	Yes		Global	Ye
Ndb_execute_count				Yes	Global	No
ndb_extra_logging	Yes	Yes	Yes		Global	Ye
ndb_force_send	Yes	Yes	Yes		Both	Ye
ndb_fully_replicated	Yes	Yes	Yes		Both	Ye
ndb_index_stat_enable	Yes	Yes	Yes		Both	Ye
ndb_index_stat_option	Yes	Yes	Yes		Both	Ye
ndb_join_pushdown			Yes		Both	Ye
Ndb_last_commit_epoch_server				Yes	Global	No
Ndb_last_commit_epoch_session				Yes	Session	No
ndb-log-apply-status	Yes	Yes			Global	No
- *Variable*: ndb_log_apply_status			Yes		Global	No
ndb_log_apply_status	Yes	Yes	Yes		Global	No
ndb_log_bin	Yes		Yes		Both	Ye
ndb_log_binlog_index	Yes		Yes		Global	Ye
ndb-log-empty-epochs	Yes	Yes			Global	Ye
- *Variable*: ndb_log_empty_epochs			Yes		Global	Ye
ndb_log_empty_epochs	Yes	Yes	Yes		Global	Ye
ndb-log-empty-update	Yes	Yes			Global	Ye
- *Variable*: ndb_log_empty_update			Yes		Global	Ye
ndb_log_empty_update	Yes	Yes	Yes		Global	Ye
ndb-log-exclusive-reads	Yes	Yes			Both	Ye
- *Variable*: ndb_log_exclusive_reads			Yes		Both	Ye
ndb_log_exclusive_reads	Yes	Yes	Yes		Both	Ye
ndb-log-orig	Yes	Yes			Global	No
- *Variable*: ndb_log_orig			Yes		Global	No
ndb_log_orig	Yes	Yes	Yes		Global	No
ndb-log-transaction-id	Yes	Yes			Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
- *Variable*: ndb_log_transaction_id			Yes		Global	No
ndb_log_transaction_id			Yes		Global	No
ndb-log-update-as-write	Yes	Yes			Global	Yes
- *Variable*: ndb_log_update_as_write			Yes		Global	Yes
ndb-log-update-minimal	Yes	Yes			Global	Yes
- *Variable*: ndb_log_update_minimal			Yes		Global	Yes
ndb_log_updated_only	Yes	Yes	Yes		Global	Yes
Ndb_metadata_blacklist_size				Yes	Global	No
ndb_metadata_check	Yes	Yes	Yes		Global	Yes
ndb_metadata_check_interval	Yes	Yes	Yes		Global	Yes
Ndb_metadata_detected_count				Yes	Global	No
Ndb_metadata_synced_count				Yes	Global	No
ndb-mgmd-host	Yes	Yes				
ndb-nodeid	Yes	Yes		Yes	Global	No
Ndb_number_of_data_nodes				Yes	Global	No
ndb-optimization-delay	Yes	Yes			Global	Yes
- *Variable*: ndb_optimization_delay			Yes		Global	Yes
ndb_optimized_node_selection	Yes	Yes	Yes		Global	No
Ndb_pruned_scan_count				Yes	Global	No
Ndb_pushed_queries_defined				Yes	Global	No
Ndb_pushed_queries_dropped				Yes	Global	No
Ndb_pushed_queries_executed				Yes	Global	No
Ndb_pushed_reads				Yes	Global	No
ndb_read_backup	Yes	Yes	Yes		Global	Yes
ndb-recv-thread-activation-threshold	Yes	Yes				
- *Variable*: ndb_recv_thread_activation_threshold						
ndb_recv_thread_activation_threshold			Yes		Global	Yes
ndb-recv-thread-cpu-mask	Yes	Yes				
- *Variable*: ndb_recv_thread_cpu_mask						
ndb_recv_thread_cpu_mask			Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
ndb_report_thresh_binlog_epoch_slip	Yes	Yes	Yes		Global	Ye
ndb_report_thresh_binlog_mem_usage	Yes	Yes	Yes		Global	Ye
ndb_row_checksum			Yes		Both	Ye
Ndb_scan_count				Yes	Global	No
ndb_schema_dist_lock_wait_timeout	Yes	Yes	Yes		Global	Ye
ndb-schema-dist-timeout	Yes	Yes			Global	No
- *Variable*: ndb_schema_dist_timeout			Yes		Global	No
ndb_schema_dist_timeout	Yes	Yes	Yes		Global	No
ndb_schema_dist_upgrade_allowed	Yes	Yes	Yes		Global	No
ndb_show_foreign_key_mock_tables	Yes	Yes	Yes		Global	Ye
ndb_slave_conflict_role	Yes	Yes	Yes		Global	Ye
Ndb_slave_max_replicated_epoch			Yes		Global	No
Ndb_system_name			Yes		Global	No
ndb_table_no_logging			Yes		Session	Ye
ndb_table_temporary			Yes		Session	Ye
Ndb_trans_hint_count_session				Yes	Both	No
ndb-transid-mysql-connection-map	Yes					
ndb_use_copying_alter_table			Yes		Both	No
ndb_use_exact_count			Yes		Both	Ye
ndb_use_transactions	Yes	Yes	Yes		Both	Ye
ndb_version			Yes		Global	No
ndb_version_string			Yes		Global	No
ndb-wait-connected	Yes	Yes	Yes		Global	No
ndb-wait-setup	Yes	Yes	Yes		Global	No
ndbcluster	Yes	Yes				
ndbinfo	Yes					
ndbinfo_database			Yes		Global	No
ndbinfo_max_bytes	Yes		Yes		Both	Ye
ndbinfo_max_rows	Yes		Yes		Both	Ye
ndbinfo_offline			Yes		Global	Ye
ndbinfo_show_hidden	Yes		Yes		Both	Ye
ndbinfo_table_prefix	Yes		Yes		Both	Ye
ndbinfo_version			Yes		Global	No
net_buffer_length	Yes	Yes	Yes		Both	Ye
net_read_timeout	Yes	Yes	Yes		Both	Ye
net_retry_count	Yes	Yes	Yes		Both	Ye

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
net_write_timeout	Yes	Yes	Yes		Both	Yes
new	Yes	Yes	Yes		Both	Yes
ngram_token_size	Yes	Yes	Yes		Global	No
no-dd-upgrade	Yes	Yes				
no-defaults	Yes					
no-monitor	Yes	Yes				
Not_flushed_delayed_rows				Yes	Global	No
offline_mode	Yes	Yes	Yes		Global	Yes
old	Yes	Yes	Yes		Global	No
old_alter_table	Yes	Yes	Yes		Both	Yes
old_passwords	Yes	Yes	Yes		Both	Yes
old-style-user-limits	Yes	Yes				
Ongoing_anonymous_gtid_violating_transaction_count				Yes	Global	No
Ongoing_anonymous_transaction_count				Yes	Global	No
Ongoing_automatic_gtid_violating_transaction_count				Yes	Global	No
Open_files				Yes	Global	No
open_files_limit	Yes	Yes	Yes		Global	No
Open_streams				Yes	Global	No
Open_table_definitions				Yes	Global	No
Open_tables				Yes	Both	No
Opened_files				Yes	Global	No
Opened_table_definitions				Yes	Both	No
Opened_tables				Yes	Both	No
optimizer_prune_level	Yes	Yes	Yes		Both	Yes
optimizer_search_depth	Yes	Yes	Yes		Both	Yes
optimizer_switch	Yes	Yes	Yes		Both	Yes
optimizer_trace	Yes	Yes	Yes		Both	Yes
optimizer_trace_features	Yes	Yes	Yes		Both	Yes
optimizer_trace_limit	Yes	Yes	Yes		Both	Yes
optimizer_trace_max_mem_size	Yes	Yes	Yes		Both	Yes
optimizer_trace_offset	Yes	Yes	Yes		Both	Yes
original_commit_timestamp			Yes		Session	Yes
original_server_version			Yes		Session	Yes
parser_max_mem_size	Yes	Yes	Yes		Both	Yes
partial_revokes	Yes	Yes	Yes		Global	Yes
password_history	Yes	Yes	Yes		Global	Yes
password_require_current	Yes	Yes	Yes		Global	Yes
password_reuse_interval	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
performance_schema	Yes	Yes	Yes		Global	No
Performance_schema_accounts_lost				Yes	Global	No
performance_schema_accounts_size	Yes	Yes	Yes		Global	No
Performance_schema_cond_classes_lost				Yes	Global	No
Performance_schema_cond_instances_lost				Yes	Global	No
performance-schema-consumer-events-stages-current	Yes	Yes				
performance-schema-consumer-events-stages-history	Yes	Yes				
performance-schema-consumer-events-stages-history-long	Yes	Yes				
performance-schema-consumer-events-statements-current	Yes	Yes				
performance-schema-consumer-events-statements-history	Yes	Yes				
performance-schema-consumer-events-statements-history-long	Yes	Yes				
performance-schema-consumer-events-transactions-current	Yes	Yes				
performance-schema-consumer-events-transactions-history	Yes	Yes				
performance-schema-consumer-events-transactions-history-long	Yes	Yes				
performance-schema-consumer-events-waits-current	Yes	Yes				
performance-schema-consumer-events-waits-history	Yes	Yes				
performance-schema-consumer-events-waits-history-long	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
performance-schema-consumer-global-instrumentation	Yes	Yes				
performance-schema-consumer-statements-digest	Yes	Yes				
performance-schema-consumer-thread-instrumentation	Yes	Yes				
Performance_schema_digest_lost				Yes	Global	No
performance_schema_digests_size	Yes	Yes	Yes		Global	No
performance_schema_error_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_transactions_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_transactions_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_size	Yes	Yes	Yes		Global	No
Performance_schema_file_classes_lost				Yes	Global	No
Performance_schema_file_handles_lost				Yes	Global	No
Performance_schema_file_instances_lost				Yes	Global	No
Performance_schema_hosts_lost				Yes	Global	No
performance_schema_hosts_size	Yes	Yes	Yes		Global	No
Performance_schema_index_stat_lost				Yes	Global	No
performance-schema-instrument	Yes	Yes				
Performance_schema_locker_lost				Yes	Global	No
performance_schema_max_cond_classes	Yes	Yes	Yes		Global	No
performance_schema_max_cond_instances	Yes	Yes	Yes		Global	No
performance_schema_max_digest_length	Yes	Yes	Yes		Global	No
performance_schema_max_digest_sample_age	Yes	Yes	Yes		Global	Yes
performance_schema_max_file_classes	Yes	Yes	Yes		Global	No
performance_schema_max_file_handles	Yes	Yes	Yes		Global	No
performance_schema_max_file_instances	Yes	Yes	Yes		Global	No
performance_schema_max_index_stat	Yes	Yes	Yes		Global	No
performance_schema_max_memory_classes	Yes	Yes	Yes		Global	No
performance_schema_max_metadata_locks	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_classes	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
performance_schema_max_mutex_instances	Yes	Yes	Yes		Global	No
performance_schema_max_prepared_statements_instances	Yes	Yes	Yes		Global	No
performance_schema_max_program_instances	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_classes	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_instances	Yes	Yes	Yes		Global	No
performance_schema_max_socket_classes	Yes	Yes	Yes		Global	No
performance_schema_max_socket_instances	Yes	Yes	Yes		Global	No
performance_schema_max_sql_text_length	Yes	Yes	Yes		Global	No
performance_schema_max_stage_classes	Yes	Yes	Yes		Global	No
performance_schema_max_statement_classes	Yes	Yes	Yes		Global	No
performance_schema_max_statement_stack	Yes	Yes	Yes		Global	No
performance_schema_max_table_handles	Yes	Yes	Yes		Global	No
performance_schema_max_table_instances	Yes	Yes	Yes		Global	No
performance_schema_max_table_lock_stat	Yes	Yes	Yes		Global	No
performance_schema_max_thread_classes	Yes	Yes	Yes		Global	No
performance_schema_max_thread_instances	Yes	Yes	Yes		Global	No
Performance_schema_memory_classes_lost				Yes	Global	No
Performance_schema_metadata_lock_lost				Yes	Global	No
Performance_schema_mutex_classes_lost				Yes	Global	No
Performance_schema_mutex_instances_lost				Yes	Global	No
Performance_schema_nested_statement_lost				Yes	Global	No
Performance_schema_prepared_statements_lost				Yes	Global	No
Performance_schema_program_lost				Yes	Global	No
Performance_schema_rwlock_classes_lost				Yes	Global	No
Performance_schema_rwlock_instances_lost				Yes	Global	No
Performance_schema_session_connect_attrs_longest_seen				Yes	Global	No
Performance_schema_session_connect_attrs_lost				Yes	Global	No
performance_schema_session_connect_attrs_size	Yes	Yes	Yes		Global	No
performance_schema_setup_actors_size	Yes	Yes	Yes		Global	No
performance_schema_setup_objects_size	Yes	Yes	Yes		Global	No
Performance_schema_socket_classes_lost				Yes	Global	No
Performance_schema_socket_instances_lost				Yes	Global	No
Performance_schema_stage_classes_lost				Yes	Global	No
Performance_schema_statement_classes_lost				Yes	Global	No
Performance_schema_table_handles_lost				Yes	Global	No
Performance_schema_table_instances_lost				Yes	Global	No
Performance_schema_table_lock_stat_lost				Yes	Global	No
Performance_schema_thread_classes_lost				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Performance_schema_thread_instances_lost				Yes	Global	No
Performance_schema_users_lost				Yes	Global	No
performance_schema_users_size	Yes	Yes	Yes		Global	No
persist_only_admin_x509_subject	Yes	Yes	Yes		Global	No
persisted_globals_load	Yes	Yes	Yes		Global	No
pid_file	Yes	Yes	Yes		Global	No
plugin_dir	Yes	Yes	Yes		Global	No
plugin-load	Yes	Yes	Yes		Global	No
plugin-load-add	Yes	Yes	Yes		Global	No
plugin-xxx	Yes	Yes				
port	Yes	Yes	Yes		Global	No
port-open-timeout	Yes	Yes				
preload_buffer_size	Yes	Yes	Yes		Both	Yes
Prepared_stmt_count				Yes	Global	No
print-defaults	Yes					
print_identified_with_as_hex	Yes	Yes	Yes		Both	Yes
profiling			Yes		Both	Yes
profiling_history_size	Yes	Yes	Yes		Both	Yes
protocol_compression_algorithms	Yes	Yes	Yes		Global	Yes
protocol_version			Yes		Global	No
proxy_user			Yes		Session	No
pseudo_slave_mode			Yes		Session	Yes
pseudo_thread_id			Yes		Session	Yes
Qcache_free_blocks				Yes	Global	No
Qcache_free_memory				Yes	Global	No
Qcache_hits				Yes	Global	No
Qcache_inserts				Yes	Global	No
Qcache_lowmem_prunes				Yes	Global	No
Qcache_not_cached				Yes	Global	No
Qcache_queries_in_cache				Yes	Global	No
Qcache_total_blocks				Yes	Global	No
Queries				Yes	Both	No
query_alloc_block_size	Yes	Yes	Yes		Both	Yes
query_cache_limit	Yes	Yes	Yes		Global	Yes
query_cache_min_res_unit	Yes	Yes	Yes		Global	Yes
query_cache_size	Yes	Yes	Yes		Global	Yes
query_cache_type	Yes	Yes	Yes		Both	Yes
query_cache_wlock_invalidate	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
query_prealloc_size	Yes	Yes	Yes		Both	Ye
Questions				Yes	Both	No
rand_seed1			Yes		Session	Ye
rand_seed2			Yes		Session	Ye
range_alloc_block_size	Yes	Yes	Yes		Both	Ye
range_optimizer_max_mem_size	Yes	Yes	Yes		Both	Ye
rbr_exec_mode			Yes		Both	Ye
read_buffer_size	Yes	Yes	Yes		Both	Ye
read_only	Yes	Yes	Yes		Global	Ye
read_rnd_buffer_size	Yes	Yes	Yes		Both	Ye
regexp_stack_limit	Yes	Yes	Yes		Global	Ye
regexp_time_limit	Yes	Yes	Yes		Global	Ye
relay_log	Yes	Yes	Yes		Global	No
relay_log_basename			Yes		Global	No
relay_log_index	Yes	Yes	Yes		Global	No
relay_log_info_file	Yes	Yes	Yes		Global	No
relay-log-info-repository	Yes	Yes				
- *Variable*: relay_log_info_repository						
relay_log_info_repository	Yes	Yes	Yes		Global	Ye
relay_log_purge	Yes	Yes	Yes		Global	Ye
relay-log-recovery	Yes	Yes				
- *Variable*: relay_log_recovery						
relay_log_recovery	Yes	Yes	Yes		Global	No
relay_log_space_limit	Yes	Yes	Yes		Global	No
remove	Yes					
replicate-do-db	Yes	Yes				
replicate-do-table	Yes	Yes				
replicate-ignore-db	Yes	Yes				
replicate-ignore-table	Yes	Yes				
replicate-rewrite-db	Yes	Yes				
replicate-same-server-id	Yes	Yes				
replicate-wild-do-table	Yes	Yes				
replicate-wild-ignore-table	Yes	Yes				
report_host	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
report_password	Yes	Yes	Yes		Global	No
report_port	Yes	Yes	Yes		Global	No
report_user	Yes	Yes	Yes		Global	No
require_secure_transport	Yes	Yes	Yes		Global	Yes
resultset_metadata			Yes		Session	Yes
rewriter_enabled			Yes		Global	Yes
Rewriter_number_loaded_rules				Yes	Global	No
Rewriter_number_reloads				Yes	Global	No
Rewriter_number_rewritten_queries				Yes	Global	No
Rewriter_reload_error				Yes	Global	No
rewriter_verbose			Yes		Global	Yes
rpl_read_size	Yes	Yes	Yes		Global	Yes
Rpl_semi_sync_master_clients				Yes	Global	No
rpl_semi_sync_master_enabled	Yes	Yes	Yes		Global	Yes
Rpl_semi_sync_master_net_avg_wait_time				Yes	Global	No
Rpl_semi_sync_master_net_wait_time				Yes	Global	No
Rpl_semi_sync_master_net_waits				Yes	Global	No
Rpl_semi_sync_master_no_times				Yes	Global	No
Rpl_semi_sync_master_no_tx				Yes	Global	No
Rpl_semi_sync_master_status				Yes	Global	No
Rpl_semi_sync_master_timefunc_failures				Yes	Global	No
rpl_semi_sync_master_timeout	Yes	Yes	Yes		Global	Yes
rpl_semi_sync_master_trace_level	Yes	Yes	Yes		Global	Yes
Rpl_semi_sync_master_tx_avg_wait_time				Yes	Global	No
Rpl_semi_sync_master_tx_wait_time				Yes	Global	No
Rpl_semi_sync_master_tx_waits				Yes	Global	No
rpl_semi_sync_master_wait_for_slave_count	Yes	Yes	Yes		Global	Yes
rpl_semi_sync_master_wait_no_slave	Yes	Yes	Yes		Global	Yes
rpl_semi_sync_master_wait_point	Yes	Yes	Yes		Global	Yes
Rpl_semi_sync_master_wait_pos_backtraverse				Yes	Global	No
Rpl_semi_sync_master_wait_sessions				Yes	Global	No
Rpl_semi_sync_master_yes_tx				Yes	Global	No
rpl_semi_sync_slave_enabled	Yes	Yes	Yes		Global	Yes
Rpl_semi_sync_slave_status				Yes	Global	No
rpl_semi_sync_slave_trace_level	Yes	Yes	Yes		Global	Yes
rpl_stop_slave_timeout	Yes	Yes	Yes		Global	Yes
Rsa_public_key				Yes	Global	No
safe-user-create	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
schema_definition_cache	Yes	Yes	Yes		Global	Ye
secondary_engine_cost_threshold			Yes		Session	Ye
Secondary_engine_execution_count				Yes	Both	No
secure-auth	Yes	Yes			Global	Ye
- *Variable*: secure_auth			Yes		Global	Ye
secure_file_priv	Yes	Yes	Yes		Global	No
Select_full_join				Yes	Both	No
Select_full_range_join				Yes	Both	No
Select_range				Yes	Both	No
Select_range_check				Yes	Both	No
Select_scan				Yes	Both	No
server-id	Yes	Yes			Global	Ye
- *Variable*: server_id			Yes		Global	Ye
server-id-bits	Yes	Yes			Global	No
- *Variable*: server_id_bits			Yes		Global	No
server_uuid			Yes		Global	No
session_track_gtids	Yes	Yes	Yes		Both	Ye
session_track_schema	Yes	Yes	Yes		Both	Ye
session_track_state_change	Yes	Yes	Yes		Both	Ye
session_track_system_variables	Yes	Yes	Yes		Both	Ye
session_track_transaction_info	Yes	Yes	Yes		Both	Ye
sha256_password_auto_generate_rsa_keys	Yes	Yes	Yes		Global	No
sha256_password_private_key_path	Yes	Yes	Yes		Global	No
sha256_password_proxy_users	Yes	Yes	Yes		Global	Ye
sha256_password_public_key_path	Yes	Yes	Yes		Global	No
shared_memory	Yes	Yes	Yes		Global	No
shared_memory_base_name	Yes	Yes	Yes		Global	No
show_compatibility_56	Yes	Yes	Yes		Global	Ye
show_create_table_skip_secondary_engine	Yes	Yes	Yes		Session	Ye
show_create_table_verbosity	Yes	Yes	Yes		Both	Ye
show_old_temporals	Yes	Yes	Yes		Both	Ye
show-slave-auth-info	Yes	Yes				
skip-character-set-client-handshake	Yes	Yes				
skip-concurrent-insert	Yes	Yes				
- *Variable*: concurrent_insert						

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
skip_external_locking	Yes	Yes	Yes		Global	No
skip-grant-tables	Yes	Yes				
skip-host-cache	Yes	Yes				
skip-name-resolve	Yes	Yes			Global	No
- *Variable*: skip_name_resolve			Yes		Global	No
skip-ndbcluster	Yes	Yes				
skip-networking	Yes	Yes			Global	No
- *Variable*: skip_networking			Yes		Global	No
skip-new	Yes	Yes				
skip-show-database	Yes	Yes			Global	No
- *Variable*: skip_show_database			Yes		Global	No
skip-slave-start	Yes	Yes				
skip-ssl	Yes	Yes				
skip-stack-trace	Yes	Yes				
slave_allow_batching	Yes	Yes	Yes		Global	Yes
slave-checkpoint-group	Yes	Yes				
- *Variable*: slave_checkpoint_group						
slave_checkpoint_group	Yes	Yes	Yes		Global	Yes
slave-checkpoint-period	Yes	Yes				
- *Variable*: slave_checkpoint_period						
slave_checkpoint_period	Yes	Yes	Yes		Global	Yes
slave_compressed_protocol	Yes	Yes	Yes		Global	Yes
slave_exec_mode	Yes	Yes	Yes		Global	Yes
Slave_heartbeat_period				Yes	Global	No
Slave_last_heartbeat				Yes	Global	No
slave-load-tmpdir	Yes	Yes			Global	No
- *Variable*: slave_load_tmpdir			Yes		Global	No
slave-max-allowed-packet	Yes	Yes				
- *Variable*: slave_max_allowed_packet						
slave_max_allowed_packet	Yes	Yes	Yes		Global	Yes
slave-net-timeout	Yes	Yes			Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
- *Variable*: slave_net_timeout			Yes		Global	Ye
Slave_open_temp_tables				Yes	Global	No
slave-parallel-type	Yes	Yes				
- *Variable*: slave_parallel_type						
slave_parallel_type	Yes	Yes	Yes		Global	Ye
slave-parallel-workers	Yes	Yes				
- *Variable*: slave_parallel_workers						
slave_parallel_workers	Yes	Yes	Yes		Global	Ye
slave-pending-jobs-size-max	Yes					
- *Variable*: slave_pending_jobs_size_max						
slave_pending_jobs_size_max	Yes	Yes	Yes		Global	Ye
slave_preserve_commit_order	Yes	Yes	Yes		Global	Ye
Slave_received_heartbeats				Yes	Global	No
Slave_retried_transactions				Yes	Global	No
Slave_rows_last_search_algorithm_used				Yes	Global	No
slave-rows-search-algorithms	Yes	Yes				
- *Variable*: slave_rows_search_algorithms						
slave_rows_search_algorithms	Yes	Yes	Yes		Global	Ye
Slave_running				Yes	Global	No
slave-skip-errors	Yes	Yes	Yes		Global	No
slave-sql-verify-checksum	Yes	Yes				
slave_sql_verify_checksum	Yes	Yes	Yes		Global	Ye
slave_transaction_retries	Yes	Yes	Yes		Global	Ye
slave_type_conversions	Yes	Yes	Yes		Global	No
Slow_launch_threads				Yes	Both	No
slow_launch_time	Yes	Yes	Yes		Global	Ye
Slow_queries				Yes	Both	No
slow_query_log	Yes	Yes	Yes		Global	Ye
slow_query_log_file	Yes	Yes	Yes		Global	Ye
slow-start-timeout	Yes	Yes				
socket	Yes	Yes	Yes		Global	No
sort_buffer_size	Yes	Yes	Yes		Both	Ye

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Sort_merge_passes				Yes	Both	No
Sort_range				Yes	Both	No
Sort_rows				Yes	Both	No
Sort_scan				Yes	Both	No
sporadic-binlog-dump-fail	Yes	Yes				
sql_auto_is_null			Yes		Both	Yes
sql_big_selects			Yes		Both	Yes
sql_buffer_result			Yes		Both	Yes
sql_log_bin			Yes		Session	Yes
sql_log_off			Yes		Both	Yes
sql-mode	Yes	Yes	Yes		Both	Yes
sql_notes			Yes		Both	Yes
sql_quote_show_create			Yes		Both	Yes
sql_require_primary_key	Yes	Yes	Yes		Both	Yes
sql_safe_updates			Yes		Both	Yes
sql_select_limit			Yes		Both	Yes
sql_slave_skip_counter			Yes		Global	Yes
sql_warnings			Yes		Both	Yes
ssl	Yes	Yes				
Ssl_accept_renegotiates				Yes	Global	No
Ssl_accepts				Yes	Global	No
ssl-ca	Yes	Yes	Yes		Global	Varies
Ssl_callback_cache_hits				Yes	Global	No
ssl-capath	Yes	Yes	Yes		Global	Varies
ssl-cert	Yes	Yes	Yes		Global	Varies
Ssl_cipher				Yes	Both	No
ssl-cipher	Yes	Yes	Yes		Global	Varies
Ssl_cipher_list				Yes	Both	No
Ssl_client_connects				Yes	Global	No
Ssl_connect_renegotiates				Yes	Global	No
ssl-crl	Yes	Yes	Yes		Global	Varies
ssl-crlpath	Yes	Yes	Yes		Global	Varies
Ssl_ctx_verify_depth				Yes	Global	No
Ssl_ctx_verify_mode				Yes	Global	No
Ssl_default_timeout				Yes	Both	No
Ssl_finished_accepts				Yes	Global	No
Ssl_finished_connects				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
ssl_fips_mode	Yes	Yes	Yes		Global	Ye
ssl-key	Yes	Yes	Yes		Global	Va
Ssl_server_not_after				Yes	Both	No
Ssl_server_not_before				Yes	Both	No
Ssl_session_cache_hits				Yes	Global	No
Ssl_session_cache_misses				Yes	Global	No
Ssl_session_cache_mode				Yes	Global	No
Ssl_session_cache_overflows				Yes	Global	No
Ssl_session_cache_size				Yes	Global	No
Ssl_session_cache_timeouts				Yes	Global	No
Ssl_sessions_reused				Yes	Both	No
Ssl_used_session_cache_entries				Yes	Global	No
Ssl_verify_depth				Yes	Both	No
Ssl_verify_mode				Yes	Both	No
Ssl_version				Yes	Both	No
standalone	Yes	Yes				
stored_program_cache	Yes	Yes	Yes		Global	Ye
stored_program_definition_cache	Yes	Yes	Yes		Global	Ye
super-large-pages	Yes	Yes				
super_read_only	Yes	Yes	Yes		Global	Ye
symbolic-links	Yes	Yes				
sync_binlog	Yes	Yes	Yes		Global	Ye
sync_master_info	Yes	Yes	Yes		Global	Ye
sync_relay_log	Yes	Yes	Yes		Global	Ye
sync_relay_log_info	Yes	Yes	Yes		Global	Ye
sysdate-is-now	Yes	Yes				
syseventlog.facility	Yes	Yes	Yes		Global	Ye
syseventlog.include_pid	Yes	Yes	Yes		Global	Ye
syseventlog.tag	Yes	Yes	Yes		Global	Ye
system_time_zone			Yes		Global	No
table_definition_cache	Yes	Yes	Yes		Global	Ye
table_encryption_privilege_check	Yes	Yes	Yes		Global	Ye
Table_locks_immediate				Yes	Global	No
Table_locks_waited				Yes	Global	No
table_open_cache	Yes	Yes	Yes		Global	Ye
Table_open_cache_hits				Yes	Both	No
table_open_cache_instances	Yes	Yes	Yes		Global	No
Table_open_cache_misses				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Table_open_cache_overflows				Yes	Both	No
tablespace_definition_cache	Yes	Yes	Yes		Global	Yes
tc-heuristic-recover	Yes	Yes				
Tc_log_max_pages_used				Yes	Global	No
Tc_log_page_size				Yes	Global	No
Tc_log_page_waits				Yes	Global	No
temp-pool	Yes	Yes				
temptable_max_ram	Yes	Yes	Yes		Global	Yes
temptable_use_mmap	Yes	Yes	Yes		Global	Yes
thread_cache_size	Yes	Yes	Yes		Global	Yes
thread_handling	Yes	Yes	Yes		Global	No
thread_pool_algorithm	Yes	Yes	Yes		Global	No
thread_pool_high_priority_connection	Yes	Yes	Yes		Both	Yes
thread_pool_max_active_query_threads	Yes	Yes	Yes		Global	Yes
thread_pool_max_unused_threads	Yes	Yes	Yes		Global	Yes
thread_pool_prio_kickup_timer	Yes	Yes	Yes		Both	Yes
thread_pool_size	Yes	Yes	Yes		Global	No
thread_pool_stall_limit	Yes	Yes	Yes		Global	Yes
thread_stack	Yes	Yes	Yes		Global	No
Threads_cached				Yes	Global	No
Threads_connected				Yes	Global	No
Threads_created				Yes	Global	No
Threads_running				Yes	Global	No
time_format			Yes		Global	No
time_zone			Yes		Both	Yes
timestamp			Yes		Session	Yes
tls_ciphersuites	Yes	Yes	Yes		Global	Yes
tls_version	Yes	Yes	Yes		Global	Varies
tmp_table_size	Yes	Yes	Yes		Both	Yes
tmpdir	Yes	Yes	Yes		Global	No
transaction_alloc_block_size	Yes	Yes	Yes		Both	Yes
transaction_allow_batching			Yes		Session	Yes
transaction-isolation	Yes	Yes			Both	Yes
- *Variable*: transaction_isolation			Yes		Both	Yes
transaction_prealloc_size	Yes	Yes	Yes		Both	Yes
transaction-read-only	Yes	Yes			Both	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
- *Variable*: transaction_read_only			Yes		Both	Ye
transaction_write_set_extraction	Yes	Yes	Yes		Both	Ye
tx_isolation			Yes		Both	Ye
tx_read_only			Yes		Both	Ye
unique_checks			Yes		Both	Ye
updatable_views_with_limit	Yes	Yes	Yes		Both	Ye
upgrade	Yes	Yes				
Uptime				Yes	Global	No
Uptime_since_flush_status				Yes	Global	No
use_secondary_engine			Yes		Session	Ye
user	Yes	Yes				
validate-config	Yes	Yes				
validate-password	Yes	Yes				
validate_password_check_user_name	Yes	Yes	Yes		Global	Ye
validate_password_dictionary_file	Yes	Yes	Yes		Global	Ye
validate_password_dictionary_file_last_parsed				Yes	Global	No
validate_password_dictionary_file_words_count				Yes	Global	No
validate_password_length	Yes	Yes	Yes		Global	Ye
validate_password_mixed_case_count	Yes	Yes	Yes		Global	Ye
validate_password_number_count	Yes	Yes	Yes		Global	Ye
validate_password_policy	Yes	Yes	Yes		Global	Ye
validate_password_special_char_count	Yes	Yes	Yes		Global	Ye
validate_password.check_user_name	Yes	Yes	Yes		Global	Ye
validate_password.dictionary_file	Yes	Yes	Yes		Global	Ye
validate_password.dictionary_file_last_parsed				Yes	Global	No
validate_password.dictionary_file_words_count				Yes	Global	No
validate_password.length	Yes	Yes	Yes		Global	Ye
validate_password.mixed_case_count	Yes	Yes	Yes		Global	Ye
validate_password.number_count	Yes	Yes	Yes		Global	Ye
validate_password.policy	Yes	Yes	Yes		Global	Ye
validate_password.special_char_count	Yes	Yes	Yes		Global	Ye
validate_user_plugins	Yes	Yes	Yes		Global	No
verbose	Yes	Yes				
version			Yes		Global	No
version_comment			Yes		Global	No
version_compile_machine			Yes		Global	No
version_compile_os			Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
version_compile_zlib			Yes		Global	No
version_tokens_session	Yes	Yes	Yes		Both	Yes
version_tokens_session_number	Yes	Yes	Yes		Both	No
wait_timeout	Yes	Yes	Yes		Both	Yes
warning_count			Yes		Session	No
windowing_use_high_precision	Yes	Yes	Yes		Both	Yes

Notes:

1. This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

5.1.5 Server System Variable Reference

The following table lists all system variables applicable within `mysqld`.

The table lists command-line options (Cmd-line), options valid in configuration files (Option file), server system variables (System Var), and status variables (Status var) in one unified list, with an indication of where each option or variable is valid. If a server option set on the command line or in an option file differs from the name of the corresponding system variable, the variable name is noted immediately below the corresponding option. The scope of the variable (Var Scope) is Global, Session, or both. Please see the corresponding item descriptions for details on setting and using the variables. Where appropriate, direct links to further information about the items are provided.

Table 5.2 System Variable Summary

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
activate_all_roles_on_login	Yes	Yes	Yes	Global	Yes
admin_address	Yes	Yes	Yes	Global	No
admin_port	Yes	Yes	Yes	Global	No
audit_log_buffer_size	Yes	Yes	Yes	Global	No
audit_log_compression	Yes	Yes	Yes	Global	No
audit_log_connection_policy	Yes	Yes	Yes	Global	Yes
audit_log_current_session			Yes	Both	No
audit_log_encryption	Yes	Yes	Yes	Global	No
audit_log_exclude_accounts	Yes	Yes	Yes	Global	Yes
audit_log_file	Yes	Yes	Yes	Global	No
audit_log_filter_id			Yes	Both	No
audit_log_flush			Yes	Global	Yes
audit_log_format	Yes	Yes	Yes	Global	No
audit_log_include_accounts	Yes	Yes	Yes	Global	Yes
audit_log_password_history_keep_days	Yes	Yes	Yes	Global	Yes
audit_log_policy	Yes	Yes	Yes	Global	No
audit_log_read_buffer_size	Yes	Yes	Yes	Varies	Varies
audit_log_rotate_on_size	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
audit_log_statement_policy	Yes	Yes	Yes	Global	Yes
audit_log_strategy	Yes	Yes	Yes	Global	No
authentication_ldap_sasl_auth_method_name	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_bind_base_dn	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_bind_root_dn	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_bind_root_pwd	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_ca_path	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_group_search_attr	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_group_search_filter	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_init_pool_size	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_log_status	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_max_pool_size	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_server_host	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_server_port	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_tls	Yes	Yes	Yes	Global	Yes
authentication_ldap_sasl_user_search_attr	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_auth_method_name	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_bind_base_dn	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_bind_root_dn	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_bind_root_pwd	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_ca_path	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_group_search_attr	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_group_search_filter	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_init_pool_size	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_log_status	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_max_pool_size	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_server_host	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_server_port	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_tls	Yes	Yes	Yes	Global	Yes
authentication_ldap_simple_user_search_attr	Yes	Yes	Yes	Global	Yes
authentication_windows_log_level	Yes	Yes	Yes	Global	No
authentication_windows_use_principal_name	Yes	Yes	Yes	Global	No
auto_generate_certs	Yes	Yes	Yes	Global	No
auto_increment_increment	Yes	Yes	Yes	Both	Yes
auto_increment_offset	Yes	Yes	Yes	Both	Yes
autocommit	Yes	Yes	Yes	Both	Yes
automatic_sp_privileges	Yes	Yes	Yes	Global	Yes
avoid_temporal_upgrade	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
back_log	Yes	Yes	Yes	Global	No
basedir	Yes	Yes	Yes	Global	No
big_tables	Yes	Yes	Yes	Both	Yes
bind_address	Yes	Yes	Yes	Global	No
binlog_cache_size	Yes	Yes	Yes	Global	Yes
binlog_checksum	Yes	Yes	Yes	Global	Yes
binlog_direct_non_transactional_updates	Yes	Yes	Yes	Both	Yes
binlog_encryption	Yes	Yes	Yes	Global	Yes
binlog_error_action	Yes	Yes	Yes	Global	Yes
binlog_expire_logs_seconds	Yes	Yes	Yes	Global	Yes
binlog_format	Yes	Yes	Yes	Both	Yes
binlog_group_commit_sync_delay	Yes	Yes	Yes	Global	Yes
binlog_group_commit_sync_no_delay_count	Yes	Yes	Yes	Global	Yes
binlog_gtid_simple_recovery	Yes	Yes	Yes	Global	No
binlog_max_flush_queue_time	Yes	Yes	Yes	Global	Yes
binlog_order_commits	Yes	Yes	Yes	Global	Yes
binlog_rotate_encryption_master_key_at_startup	Yes	Yes	Yes	Global	No
binlog-row-event-max-size	Yes	Yes	Yes	Global	No
binlog_row_image	Yes	Yes	Yes	Both	Yes
binlog_row_metadata	Yes	Yes	Yes	Global	Yes
binlog_row_value_options	Yes	Yes	Yes	Both	Yes
binlog_rows_query_log_events	Yes	Yes	Yes	Both	Yes
binlog_stmt_cache_size	Yes	Yes	Yes	Global	Yes
binlog_transaction_dependency_history_size	Yes	Yes	Yes	Global	Yes
binlog_transaction_dependency_tracking	Yes	Yes	Yes	Global	Yes
block_encryption_mode	Yes	Yes	Yes	Both	Yes
bulk_insert_buffer_size	Yes	Yes	Yes	Both	Yes
caching_sha2_password_auto_generate_rsa_keys	Yes	Yes	Yes	Global	No
caching_sha2_password_private_key_path	Yes	Yes	Yes	Global	No
caching_sha2_password_public_key_path	Yes	Yes	Yes	Global	No
character_set_client			Yes	Both	Yes
character_set_connection			Yes	Both	Yes
character_set_database (note 1)			Yes	Both	Yes
character_set_filesystem	Yes	Yes	Yes	Both	Yes
character_set_results			Yes	Both	Yes
character_set_server	Yes	Yes	Yes	Both	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
character_set_system			Yes	Global	No
character_sets_dir	Yes	Yes	Yes	Global	No
check_proxy_users	Yes	Yes	Yes	Global	Yes
clone_autotune_concurrency	Yes	Yes	Yes	Global	Yes
clone_buffer_size	Yes	Yes	Yes	Global	Yes
clone_ddl_timeout	Yes	Yes	Yes	Global	Yes
clone_enable_compression	Yes	Yes	Yes	Global	Yes
clone_max_concurrency	Yes	Yes	Yes	Global	Yes
clone_max_data_bandwidth	Yes	Yes	Yes	Global	Yes
clone_max_network_bandwidth	Yes	Yes	Yes	Global	Yes
clone_ssl_ca	Yes	Yes	Yes	Global	Yes
clone_ssl_cert	Yes	Yes	Yes	Global	Yes
clone_ssl_key	Yes	Yes	Yes	Global	Yes
clone_valid_donor_list	Yes	Yes	Yes	Global	Yes
collation_connection			Yes	Both	Yes
collation_database (note 1)			Yes	Both	Yes
collation_server	Yes	Yes	Yes	Both	Yes
completion_type	Yes	Yes	Yes	Both	Yes
concurrent_insert	Yes	Yes	Yes	Global	Yes
connect_timeout	Yes	Yes	Yes	Global	Yes
connection_control_failed_connections_threshold	Yes	Yes	Yes	Global	Yes
connection_control_max_connection_delay	Yes	Yes	Yes	Global	Yes
connection_control_min_connection_delay	Yes	Yes	Yes	Global	Yes
core_file			Yes	Global	No
create_admin_listener_thread	Yes	Yes	Yes	Global	No
cte_max_recursion_depth	Yes	Yes	Yes	Both	Yes
daemon_memcached_enable_binlog	Yes	Yes	Yes	Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes	Global	No
daemon_memcached_engine_lib_path	Yes	Yes	Yes	Global	No
daemon_memcached_option	Yes	Yes	Yes	Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes	Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes	Global	No
datadir	Yes	Yes	Yes	Global	No
date_format			Yes	Global	No
datetime_format			Yes	Global	No
debug	Yes	Yes	Yes	Both	Yes
debug_sync			Yes	Session	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
default_authentication_plugin	Yes	Yes	Yes	Global	No
default_collation_for_utf8mb4			Yes	Both	Yes
default_password_lifetime	Yes	Yes	Yes	Global	Yes
default_storage_engine	Yes	Yes	Yes	Both	Yes
default_table_encryption	Yes	Yes	Yes	Both	Yes
default_tmp_storage_engine	Yes	Yes	Yes	Both	Yes
default_week_format	Yes	Yes	Yes	Both	Yes
delay_key_write	Yes	Yes	Yes	Global	Yes
delayed_insert_limit	Yes	Yes	Yes	Global	Yes
delayed_insert_timeout	Yes	Yes	Yes	Global	Yes
delayed_queue_size	Yes	Yes	Yes	Global	Yes
disabled_storage_engines	Yes	Yes	Yes	Global	No
disconnect_on_expired_password	Yes	Yes	Yes	Global	No
div_precision_increment	Yes	Yes	Yes	Both	Yes
dragnet.log_error_filter_rules	Yes	Yes	Yes	Global	Yes
end_markers_in_json	Yes	Yes	Yes	Both	Yes
enforce_gtid_consistency	Yes	Yes	Yes	Global	Yes
eq_range_index_dive_limit	Yes	Yes	Yes	Both	Yes
error_count			Yes	Session	No
event_scheduler	Yes	Yes	Yes	Global	Yes
expire_logs_days	Yes	Yes	Yes	Global	Yes
explicit_defaults_for_timestamp	Yes	Yes	Yes	Both	Yes
external_user			Yes	Session	No
flush	Yes	Yes	Yes	Global	Yes
flush_time	Yes	Yes	Yes	Global	Yes
foreign_key_checks			Yes	Both	Yes
ft_boolean_syntax	Yes	Yes	Yes	Global	Yes
ft_max_word_len	Yes	Yes	Yes	Global	No
ft_min_word_len	Yes	Yes	Yes	Global	No
ft_query_expansion_limit	Yes	Yes	Yes	Global	No
ft_stopword_file	Yes	Yes	Yes	Global	No
general_log	Yes	Yes	Yes	Global	Yes
general_log_file	Yes	Yes	Yes	Global	Yes
generated_random_password_length	Yes	Yes	Yes	Both	Yes
group_concat_max_len	Yes	Yes	Yes	Both	Yes
group_replication_allow_local_disjoint_gtids_join	Yes	Yes	Yes	Global	Yes
group_replication_allow_local_lower_version_join	Yes	Yes	Yes	Global	Yes
group_replication_auto_increment_increment	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
group_replication_autorejoin_tries	Yes	Yes	Yes	Global	Yes
group_replication_bootstrap_group	Yes	Yes	Yes	Global	Yes
group_replication_clone_threshold	Yes	Yes	Yes	Global	Yes
group_replication_communication_debug_options	Yes	Yes	Yes	Global	Yes
group_replication_communication_max_message_size	Yes	Yes	Yes	Global	Yes
group_replication_components_stop_timeout	Yes	Yes	Yes	Global	Yes
group_replication_compression_threshold	Yes	Yes	Yes	Global	Yes
group_replication_consistency	Yes	Yes	Yes	Both	Yes
group_replication_enforce_update_everywhere_checks	Yes	Yes	Yes	Global	Yes
group_replication_exit_state_action	Yes	Yes	Yes	Global	Yes
group_replication_flow_control_applier_threshold	Yes	Yes	Yes	Global	Yes
group_replication_flow_control_certifier_threshold	Yes	Yes	Yes	Global	Yes
group_replication_flow_control_hold_percent	Yes	Yes	Yes	Global	Yes
group_replication_flow_control_max_commit_quota	Yes	Yes	Yes	Global	Yes
group_replication_flow_control_member_quota_percent	Yes	Yes	Yes	Global	Yes
group_replication_flow_control_min_quota	Yes	Yes	Yes	Global	Yes
group_replication_flow_control_min_recovery_quota	Yes	Yes	Yes	Global	Yes
group_replication_flow_control_mode	Yes	Yes	Yes	Global	Yes
group_replication_flow_control_period	Yes	Yes	Yes	Global	Yes
group_replication_flow_control_release_percent	Yes	Yes	Yes	Global	Yes
group_replication_force_members	Yes	Yes	Yes	Global	Yes
group_replication_group_name	Yes	Yes	Yes	Global	Yes
group_replication_group_seeds	Yes	Yes	Yes	Global	Yes
group_replication_gtid_assignment_block_size	Yes	Yes	Yes	Global	Yes
group_replication_ip_whitelist	Yes	Yes	Yes	Global	Yes
group_replication_local_address	Yes	Yes	Yes	Global	Yes
group_replication_member_expel_timeout	Yes	Yes	Yes	Global	Yes
group_replication_member_weight	Yes	Yes	Yes	Global	Yes
group_replication_message_cache_size	Yes	Yes	Yes	Global	Yes
group_replication_poll_spin_loops	Yes	Yes	Yes	Global	Yes
group_replication_recovery_complete_at	Yes	Yes	Yes	Global	Yes
group_replication_recovery_compression_algorithms	Yes	Yes	Yes	Global	Yes
group_replication_recovery_get_public_key	Yes	Yes	Yes	Global	Yes
group_replication_recovery_public_key_path	Yes	Yes	Yes	Global	Yes
group_replication_recovery_reconnect_interval	Yes	Yes	Yes	Global	Yes
group_replication_recovery_retry_count	Yes	Yes	Yes	Global	Yes
group_replication_recovery_ssl_ca	Yes	Yes	Yes	Global	Yes
group_replication_recovery_ssl_capath	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
group_replication_recovery_ssl_cert	Yes	Yes	Yes	Global	Yes
group_replication_recovery_ssl_cipher	Yes	Yes	Yes	Global	Yes
group_replication_recovery_ssl_crl	Yes	Yes	Yes	Global	Yes
group_replication_recovery_ssl_crlpath	Yes	Yes	Yes	Global	Yes
group_replication_recovery_ssl_key	Yes	Yes	Yes	Global	Yes
group_replication_recovery_ssl_verify_server_cert	Yes	Yes	Yes	Global	Yes
group_replication_recovery_use_ssl	Yes	Yes	Yes	Global	Yes
group_replication_recovery_zstd_compression_level	Yes	Yes	Yes	Global	Yes
group_replication_single_primary_mode	Yes	Yes	Yes	Global	Yes
group_replication_ssl_mode	Yes	Yes	Yes	Global	Yes
group_replication_start_on_boot	Yes	Yes	Yes	Global	Yes
group_replication_transaction_size_limit	Yes	Yes	Yes	Global	Yes
group_replication_unreachable_majority_timeout	Yes	Yes	Yes	Global	Yes
gtid_executed			Yes	Varies	No
gtid_executed_compression_period			Yes	Global	Yes
gtid_mode	Yes	Yes	Yes	Global	Yes
gtid_next			Yes	Session	Yes
gtid_owned			Yes	Both	No
gtid_purged			Yes	Global	Yes
have_compress			Yes	Global	No
have_crypt			Yes	Global	No
have_dynamic_loading			Yes	Global	No
have_geometry			Yes	Global	No
have_openssl			Yes	Global	No
have_profiling			Yes	Global	No
have_query_cache			Yes	Global	No
have_rtree_keys			Yes	Global	No
have_ssl			Yes	Global	No
have_statement_timeout			Yes	Global	No
have_symlink			Yes	Global	No
histogram_generation_max_mem_size	Yes	Yes	Yes	Both	Yes
host_cache_size	Yes	Yes	Yes	Global	Yes
hostname			Yes	Global	No
identity			Yes	Session	Yes
ignore_builtin_innodb	Yes	Yes	Yes	Global	No
immediate_server_version			Yes	Session	Yes
information_schema_stats_expiry	Yes	Yes	Yes	Both	Yes
init_connect	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
init_file	Yes	Yes	Yes	Global	No
init_slave	Yes	Yes	Yes	Global	Yes
innodb_adaptive_flushing	Yes	Yes	Yes	Global	Yes
innodb_adaptive_flushing_lwm	Yes	Yes	Yes	Global	Yes
innodb_adaptive_hash_index	Yes	Yes	Yes	Global	Yes
innodb_adaptive_hash_index_parts	Yes	Yes	Yes	Global	No
innodb_adaptive_max_sleep_delay	Yes	Yes	Yes	Global	Yes
innodb_api_bk_commit_interval	Yes	Yes	Yes	Global	Yes
innodb_api_disable_rowlock	Yes	Yes	Yes	Global	No
innodb_api_enable_binlog	Yes	Yes	Yes	Global	No
innodb_api_enable_mdl	Yes	Yes	Yes	Global	No
innodb_api_trx_level	Yes	Yes	Yes	Global	Yes
innodb_autoextend_increment	Yes	Yes	Yes	Global	Yes
innodb_autoinc_lock_mode	Yes	Yes	Yes	Global	No
innodb_background_drop_list_empty	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_chunk_size	Yes	Yes	Yes	Global	No
innodb_buffer_pool_debug	Yes	Yes	Yes	Global	No
innodb_buffer_pool_dump_at_shutdown	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_dump_now	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_dump_pct	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_filename	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_in_core_file	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_instances	Yes	Yes	Yes	Global	No
innodb_buffer_pool_load_abort	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_load_at_startup	Yes	Yes	Yes	Global	No
innodb_buffer_pool_load_now	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_size	Yes	Yes	Yes	Global	Yes
innodb_change_buffer_max_size	Yes	Yes	Yes	Global	Yes
innodb_change_buffering	Yes	Yes	Yes	Global	Yes
innodb_change_buffering_debug	Yes	Yes	Yes	Global	Yes
innodb_checkpoint_disabled	Yes	Yes	Yes	Global	Yes
innodb_checksum_algorithm	Yes	Yes	Yes	Global	Yes
innodb_cmp_per_index_enabled	Yes	Yes	Yes	Global	Yes
innodb_commit_concurrency	Yes	Yes	Yes	Global	Yes
innodb_compress_debug	Yes	Yes	Yes	Global	Yes
innodb_compression_failure_threshold_pct	Yes	Yes	Yes	Global	Yes
innodb_compression_level	Yes	Yes	Yes	Global	Yes
innodb_compression_pad_pct_max	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
innodb_concurrency_tickets	Yes	Yes	Yes	Global	Yes
innodb_data_file_path	Yes	Yes	Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes	Global	No
innodb_ddl_log_crash_reset_debug	Yes	Yes	Yes	Global	Yes
innodb_deadlock_detect	Yes	Yes	Yes	Global	Yes
innodb_dedicated_server	Yes	Yes	Yes	Global	No
innodb_default_row_format	Yes	Yes	Yes	Global	Yes
innodb_directories	Yes	Yes	Yes	Global	No
innodb_disable_sort_file_cache	Yes	Yes	Yes	Global	Yes
innodb_doublewrite	Yes	Yes	Yes	Global	No
innodb_fast_shutdown	Yes	Yes	Yes	Global	Yes
innodb_fil_make_page_dirty_debug	Yes	Yes	Yes	Global	Yes
innodb_file_per_table	Yes	Yes	Yes	Global	Yes
innodb_fill_factor	Yes	Yes	Yes	Global	Yes
innodb_flush_log_at_timeout	Yes	Yes	Yes	Global	Yes
innodb_flush_log_at_trx_commit	Yes	Yes	Yes	Global	Yes
innodb_flush_method	Yes	Yes	Yes	Global	No
innodb_flush_neighbors	Yes	Yes	Yes	Global	Yes
innodb_flush_sync	Yes	Yes	Yes	Global	Yes
innodb_flushing_avg_loops	Yes	Yes	Yes	Global	Yes
innodb_force_load_corrupted	Yes	Yes	Yes	Global	No
innodb_force_recovery	Yes	Yes	Yes	Global	No
innodb_fsync_threshold	Yes	Yes	Yes	Global	Yes
innodb_ft_aux_table			Yes	Global	Yes
innodb_ft_cache_size	Yes	Yes	Yes	Global	No
innodb_ft_enable_diag_print	Yes	Yes	Yes	Global	Yes
innodb_ft_enable_stopword	Yes	Yes	Yes	Both	Yes
innodb_ft_max_token_size	Yes	Yes	Yes	Global	No
innodb_ft_min_token_size	Yes	Yes	Yes	Global	No
innodb_ft_num_word_optimize	Yes	Yes	Yes	Global	Yes
innodb_ft_result_cache_limit	Yes	Yes	Yes	Global	Yes
innodb_ft_server_stopword_table	Yes	Yes	Yes	Global	Yes
innodb_ft_sort_pll_degree	Yes	Yes	Yes	Global	No
innodb_ft_total_cache_size	Yes	Yes	Yes	Global	No
innodb_ft_user_stopword_table	Yes	Yes	Yes	Both	Yes
innodb_idle_flush_pct	Yes	Yes	Yes	Global	Yes
innodb_io_capacity	Yes	Yes	Yes	Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
innodb_limit_optimistic_insert_debug	Yes	Yes	Yes	Global	Yes
innodb_lock_wait_timeout	Yes	Yes	Yes	Both	Yes
innodb_log_buffer_size	Yes	Yes	Yes	Global	Varies
innodb_log_checkpoint_fuzzy_now	Yes	Yes	Yes	Global	Yes
innodb_log_checkpoint_now	Yes	Yes	Yes	Global	Yes
innodb_log_checksums	Yes	Yes	Yes	Global	Yes
innodb_log_compressed_pages	Yes	Yes	Yes	Global	Yes
innodb_log_file_size	Yes	Yes	Yes	Global	No
innodb_log_files_in_group	Yes	Yes	Yes	Global	No
innodb_log_group_home_dir	Yes	Yes	Yes	Global	No
innodb_log_spin_cpu_abs_lwm	Yes	Yes	Yes	Global	Yes
innodb_log_spin_cpu_pct_hwm	Yes	Yes	Yes	Global	Yes
innodb_log_wait_for_flush_spin_hwm	Yes	Yes	Yes	Global	Yes
innodb_log_write_ahead_size	Yes	Yes	Yes	Global	Yes
innodb_lru_scan_depth	Yes	Yes	Yes	Global	Yes
innodb_max_dirty_pages_pct	Yes	Yes	Yes	Global	Yes
innodb_max_dirty_pages_pct_lwm	Yes	Yes	Yes	Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes	Global	Yes
innodb_max_purge_lag_delay	Yes	Yes	Yes	Global	Yes
innodb_max_undo_log_size	Yes	Yes	Yes	Global	Yes
innodb_merge_threshold_set_all_debug	Yes	Yes	Yes	Global	Yes
innodb_monitor_disable	Yes	Yes	Yes	Global	Yes
innodb_monitor_enable	Yes	Yes	Yes	Global	Yes
innodb_monitor_reset	Yes	Yes	Yes	Global	Yes
innodb_monitor_reset_all	Yes	Yes	Yes	Global	Yes
innodb_numa_interleave	Yes	Yes	Yes	Global	No
innodb_old_blocks_pct	Yes	Yes	Yes	Global	Yes
innodb_old_blocks_time	Yes	Yes	Yes	Global	Yes
innodb_online_alter_log_max_size	Yes	Yes	Yes	Global	Yes
innodb_open_files	Yes	Yes	Yes	Global	No
innodb_optimize_fulltext_only	Yes	Yes	Yes	Global	Yes
innodb_page_cleaners	Yes	Yes	Yes	Global	No
innodb_page_size	Yes	Yes	Yes	Global	No
innodb_parallel_read_threads	Yes	Yes	Yes	Session	Yes
innodb_print_all_deadlocks	Yes	Yes	Yes	Global	Yes
innodb_print_ddl_logs	Yes	Yes	Yes	Global	Yes
innodb_purge_batch_size	Yes	Yes	Yes	Global	Yes
innodb_purge_rseg_truncate_frequency	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
innodb_purge_threads	Yes	Yes	Yes	Global	No
innodb_random_read_ahead	Yes	Yes	Yes	Global	Yes
innodb_read_ahead_threshold	Yes	Yes	Yes	Global	Yes
innodb_read_io_threads	Yes	Yes	Yes	Global	No
innodb_read_only	Yes	Yes	Yes	Global	No
innodb_redo_log_archive_dirs	Yes	Yes	Yes	Global	Yes
innodb_redo_log_encrypt	Yes	Yes	Yes	Global	Yes
innodb_replication_delay	Yes	Yes	Yes	Global	Yes
innodb_rollback_on_timeout	Yes	Yes	Yes	Global	No
innodb_rollback_segments	Yes	Yes	Yes	Global	Yes
innodb_saved_page_number_debug	Yes	Yes	Yes	Global	Yes
innodb_scan_directories	Yes	Yes	Yes	Global	No
innodb_sort_buffer_size	Yes	Yes	Yes	Global	No
innodb_spin_wait_delay	Yes	Yes	Yes	Global	Yes
innodb_spin_wait_pause_multiplier	Yes	Yes	Yes	Global	Yes
innodb_stats_auto_recalc	Yes	Yes	Yes	Global	Yes
innodb_stats_include_delete_marked	Yes	Yes	Yes	Global	Yes
innodb_stats_method	Yes	Yes	Yes	Global	Yes
innodb_stats_on_metadata	Yes	Yes	Yes	Global	Yes
innodb_stats_persistent	Yes	Yes	Yes	Global	Yes
innodb_stats_persistent_sample_pages	Yes	Yes	Yes	Global	Yes
innodb_stats_transient_sample_pages	Yes	Yes	Yes	Global	Yes
innodb_status_output	Yes	Yes	Yes	Global	Yes
innodb_status_output_locks	Yes	Yes	Yes	Global	Yes
innodb_strict_mode	Yes	Yes	Yes	Both	Yes
innodb_sync_array_size	Yes	Yes	Yes	Global	No
innodb_sync_debug	Yes	Yes	Yes	Global	No
innodb_sync_spin_loops	Yes	Yes	Yes	Global	Yes
innodb_table_locks	Yes	Yes	Yes	Both	Yes
innodb_temp_data_file_path	Yes	Yes	Yes	Global	No
innodb_temp_tablespaces_dir	Yes	Yes	Yes	Global	No
innodb_thread_concurrency	Yes	Yes	Yes	Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes	Global	Yes
innodb_tmpdir	Yes	Yes	Yes	Both	Yes
innodb_trx_purge_view_update_only_debug	Yes	Yes	Yes	Global	Yes
innodb_trx_rseg_n_slots_debug	Yes	Yes	Yes	Global	Yes
innodb_undo_directory	Yes	Yes	Yes	Global	No
innodb_undo_log_encrypt	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
innodb_undo_log_truncate	Yes	Yes	Yes	Global	Yes
innodb_undo_logs	Yes	Yes	Yes	Global	Yes
innodb_undo_tablespaces	Yes	Yes	Yes	Global	Varies
innodb_use_native_aio	Yes	Yes	Yes	Global	No
innodb_version			Yes	Global	No
innodb_write_io_threads	Yes	Yes	Yes	Global	No
insert_id			Yes	Session	Yes
interactive_timeout	Yes	Yes	Yes	Both	Yes
internal_tmp_disk_storage_engine	Yes	Yes	Yes	Global	Yes
internal_tmp_mem_storage_engine	Yes	Yes	Yes	Both	Yes
join_buffer_size	Yes	Yes	Yes	Both	Yes
keep_files_on_create	Yes	Yes	Yes	Both	Yes
key_buffer_size	Yes	Yes	Yes	Global	Yes
key_cache_age_threshold	Yes	Yes	Yes	Global	Yes
key_cache_block_size	Yes	Yes	Yes	Global	Yes
key_cache_division_limit	Yes	Yes	Yes	Global	Yes
keyring_aws_cmk_id	Yes	Yes	Yes	Global	Yes
keyring_aws_conf_file	Yes	Yes	Yes	Global	No
keyring_aws_data_file	Yes	Yes	Yes	Global	No
keyring_aws_region	Yes	Yes	Yes	Global	Yes
keyring_encrypted_file_data	Yes	Yes	Yes	Global	Yes
keyring_encrypted_file_password	Yes	Yes	Yes	Global	Yes
keyring_file_data	Yes	Yes	Yes	Global	Yes
keyring_okv_conf_dir	Yes	Yes	Yes	Global	Yes
keyring_operations			Yes	Global	Yes
language	Yes	Yes	Yes	Global	No
large_files_support			Yes	Global	No
large_page_size			Yes	Global	No
large-pages	Yes	Yes	Yes	Global	No
last_insert_id			Yes	Session	Yes
lc-messages	Yes	Yes	Yes	Both	Yes
lc-messages-dir	Yes	Yes	Yes	Global	No
lc_time_names	Yes	Yes	Yes	Both	Yes
license			Yes	Global	No
local_infile	Yes	Yes	Yes	Global	Yes
lock_order	Yes	Yes	Yes	Global	No
lock_order_debug_loops	Yes	Yes	Yes	Global	No
lock_order_debug_missing_arc	Yes	Yes	Yes	Global	No

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
lock_order_debug_missing_key	Yes	Yes	Yes	Global	No
lock_order_debug_missing_unlock	Yes	Yes	Yes	Global	No
lock_order_dependencies	Yes	Yes	Yes	Global	No
lock_order_extra_dependencies	Yes	Yes	Yes	Global	No
lock_order_output_directory	Yes	Yes	Yes	Global	No
lock_order_print_txt	Yes	Yes	Yes	Global	No
lock_order_trace_loop	Yes	Yes	Yes	Global	No
lock_order_trace_missing_arc	Yes	Yes	Yes	Global	No
lock_order_trace_missing_key	Yes	Yes	Yes	Global	No
lock_order_trace_missing_unlock	Yes	Yes	Yes	Global	No
lock_wait_timeout	Yes	Yes	Yes	Both	Yes
locked_in_memory			Yes	Global	No
log_bin			Yes	Global	No
log_bin_basename			Yes	Global	No
log-bin-index	Yes	Yes	Yes	Global	No
log_bin_trust_function_creators	Yes	Yes	Yes	Global	Yes
log_bin_use_v1_row_events	Yes	Yes	Yes	Global	No
log_builtin_as_identified_by_password	Yes	Yes	Yes	Global	Yes
log-error	Yes	Yes	Yes	Global	No
log_error_filter_rules	Yes	Yes	Yes	Global	Yes
log_error_services	Yes	Yes	Yes	Global	Yes
log_error_suppression_list	Yes	Yes	Yes	Global	Yes
log_error_verbosity	Yes	Yes	Yes	Global	Yes
log_output	Yes	Yes	Yes	Global	Yes
log_queries_not_using_indexes	Yes	Yes	Yes	Global	Yes
log_slave_updates	Yes	Yes	Yes	Global	No
log_slow_admin_statements	Yes	Yes	Yes	Global	Yes
log_slow_extra	Yes	Yes	Yes	Global	Yes
log_slow_slave_statements	Yes	Yes	Yes	Global	Yes
log_statements_unsafe_for_binlog	Yes	Yes	Yes	Global	Yes
log_syslog	Yes	Yes	Yes	Global	Yes
log_syslog_facility	Yes	Yes	Yes	Global	Yes
log_syslog_include_pid	Yes	Yes	Yes	Global	Yes
log_syslog_tag	Yes	Yes	Yes	Global	Yes
log_throttle_queries_not_using_indexes	Yes	Yes	Yes	Global	Yes
log_timestamps	Yes	Yes	Yes	Global	Yes
log-warnings	Yes	Yes	Yes	Global	Yes
long_query_time	Yes	Yes	Yes	Both	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
low_priority_updates	Yes	Yes	Yes	Both	Yes
lower_case_file_system			Yes	Global	No
lower_case_table_names	Yes	Yes	Yes	Global	No
mandatory_roles	Yes	Yes	Yes	Global	Yes
master_info_repository	Yes	Yes	Yes	Global	Yes
master_verify_checksum	Yes	Yes	Yes	Global	Yes
max_allowed_packet	Yes	Yes	Yes	Both	Yes
max_binlog_cache_size	Yes	Yes	Yes	Global	Yes
max_binlog_size	Yes	Yes	Yes	Global	Yes
max_binlog_stmt_cache_size	Yes	Yes	Yes	Global	Yes
max_connect_errors	Yes	Yes	Yes	Global	Yes
max_connections	Yes	Yes	Yes	Global	Yes
max_delayed_threads	Yes	Yes	Yes	Both	Yes
max_digest_length	Yes	Yes	Yes	Global	No
max_error_count	Yes	Yes	Yes	Both	Yes
max_execution_time	Yes	Yes	Yes	Both	Yes
max_heap_table_size	Yes	Yes	Yes	Both	Yes
max_insert_delayed_threads			Yes	Both	Yes
max_join_size	Yes	Yes	Yes	Both	Yes
max_length_for_sort_data	Yes	Yes	Yes	Both	Yes
max_points_in_geometry	Yes	Yes	Yes	Both	Yes
max_prepared_stmt_count	Yes	Yes	Yes	Global	Yes
max_relay_log_size	Yes	Yes	Yes	Global	Yes
max_seeks_for_key	Yes	Yes	Yes	Both	Yes
max_sort_length	Yes	Yes	Yes	Both	Yes
max_sp_recursion_depth	Yes	Yes	Yes	Both	Yes
max_tmp_tables			Yes	Both	Yes
max_user_connections	Yes	Yes	Yes	Both	Yes
max_write_lock_count	Yes	Yes	Yes	Global	Yes
mecab_rc_file	Yes	Yes	Yes	Global	No
metadata_locks_cache_size	Yes	Yes	Yes	Global	No
metadata_locks_hash_instances	Yes	Yes	Yes	Global	No
min_examined_row_limit	Yes	Yes	Yes	Both	Yes
multi_range_count	Yes	Yes	Yes	Both	Yes
myisam_data_pointer_size	Yes	Yes	Yes	Global	Yes
myisam_max_sort_file_size	Yes	Yes	Yes	Global	Yes
myisam_mmap_size	Yes	Yes	Yes	Global	No
myisam_recover_options	Yes	Yes	Yes	Global	No

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
myisam_repair_threads	Yes	Yes	Yes	Both	Yes
myisam_sort_buffer_size	Yes	Yes	Yes	Both	Yes
myisam_stats_method	Yes	Yes	Yes	Both	Yes
myisam_use_mmap	Yes	Yes	Yes	Global	Yes
mysql_firewall_mode	Yes	Yes	Yes	Global	Yes
mysql_firewall_trace	Yes	Yes	Yes	Global	Yes
mysql_native_password_proxy_users	Yes	Yes	Yes	Global	Yes
mysqlx_bind_address	Yes	Yes	Yes	Global	No
mysqlx_connect_timeout	Yes	Yes	Yes	Global	Yes
mysqlx_document_id_unique_prefix	Yes	Yes	Yes	Global	Yes
mysqlx_enable_hello_notice	Yes	Yes	Yes	Global	Yes
mysqlx_idle_worker_thread_timeout	Yes	Yes	Yes	Global	Yes
mysqlx_interactive_timeout	Yes	Yes	Yes	Global	Yes
mysqlx_max_allowed_packet	Yes	Yes	Yes	Global	Yes
mysqlx_max_connections	Yes	Yes	Yes	Global	Yes
mysqlx_min_worker_threads	Yes	Yes	Yes	Global	Yes
mysqlx_port	Yes	Yes	Yes	Global	No
mysqlx_port_open_timeout	Yes	Yes	Yes	Global	No
mysqlx_read_timeout	Yes	Yes	Yes	Session	Yes
mysqlx_socket	Yes	Yes	Yes	Global	No
mysqlx_ssl_ca	Yes	Yes	Yes	Global	No
mysqlx_ssl_capath	Yes	Yes	Yes	Global	No
mysqlx_ssl_cert	Yes	Yes	Yes	Global	No
mysqlx_ssl_cipher	Yes	Yes	Yes	Global	No
mysqlx_ssl_crl	Yes	Yes	Yes	Global	No
mysqlx_ssl_crlpath	Yes	Yes	Yes	Global	No
mysqlx_ssl_key	Yes	Yes	Yes	Global	No
mysqlx_wait_timeout	Yes	Yes	Yes	Session	Yes
mysqlx_write_timeout	Yes	Yes	Yes	Session	Yes
named_pipe	Yes	Yes	Yes	Global	No
named_pipe_full_access_group	Yes	Yes	Yes	Global	No
ndb-allow-copying-alter-table	Yes	Yes	Yes	Both	Yes
ndb_autoincrement_prefetch_sz	Yes	Yes	Yes	Both	Yes
ndb-batch-size	Yes	Yes			No
- *Variable*: ndb_batch_size			Yes	Global	No
ndb-blob-read-batch-bytes	Yes	Yes			Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
- *Variable*: ndb_blob_read_batch_bytes			Yes	Both	Yes
ndb-blob-write-batch-bytes	Yes	Yes			Yes
- *Variable*: ndb_blob_write_batch_bytes			Yes	Both	Yes
ndb_cache_check_time	Yes	Yes	Yes	Global	Yes
ndb_clear_apply_status	Yes		Yes	Global	Yes
ndb-cluster-connection-pool	Yes	Yes	Yes	Global	No
ndb-cluster-connection-pool-nodeids	Yes	Yes	Yes	Global	No
Ndb_conflict_last_conflict_epoch			Yes	Global	No
ndb_data_node_neighbour	Yes	Yes	Yes	Global	Yes
ndb_dbg_check_shares	Yes	Yes	Yes	Both	Yes
ndb-default-column-format	Yes	Yes			Yes
- *Variable*: ndb_default_column_format			Yes	Global	Yes
ndb_default_column_format	Yes	Yes	Yes	Global	Yes
ndb-deferred-constraints	Yes	Yes			Yes
- *Variable*: ndb_deferred_constraints			Yes	Both	Yes
ndb_deferred_constraints	Yes	Yes	Yes	Both	Yes
ndb-distribution	Yes	Yes			Yes
- *Variable*: ndb_distribution			Yes	Global	Yes
ndb_distribution	Yes	Yes	Yes	Global	Yes
ndb_eventbuffer_free_percent	Yes	Yes	Yes	Global	Yes
ndb_eventbuffer_max_alloc	Yes	Yes	Yes	Global	Yes
ndb_extra_logging	Yes	Yes	Yes	Global	Yes
ndb_force_send	Yes	Yes	Yes	Both	Yes
ndb_fully_replicated	Yes	Yes	Yes	Both	Yes
ndb_index_stat_enable	Yes	Yes	Yes	Both	Yes
ndb_index_stat_option	Yes	Yes	Yes	Both	Yes
ndb_join_pushdown			Yes	Both	Yes
ndb-log-apply-status	Yes	Yes			No

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
- *Variable*: ndb_log_apply_status			Yes	Global	No
ndb_log_apply_status	Yes	Yes	Yes	Global	No
ndb_log_bin	Yes		Yes	Both	Yes
ndb_log_binlog_index	Yes		Yes	Global	Yes
ndb-log-empty-epochs	Yes	Yes			Yes
- *Variable*: ndb_log_empty_epochs			Yes	Global	Yes
ndb_log_empty_epochs	Yes	Yes	Yes	Global	Yes
ndb-log-empty-update	Yes	Yes			Yes
- *Variable*: ndb_log_empty_update			Yes	Global	Yes
ndb_log_empty_update	Yes	Yes	Yes	Global	Yes
ndb-log-exclusive-reads	Yes	Yes			Yes
- *Variable*: ndb_log_exclusive_reads			Yes	Both	Yes
ndb_log_exclusive_reads	Yes	Yes	Yes	Both	Yes
ndb-log-orig	Yes	Yes			No
- *Variable*: ndb_log_orig			Yes	Global	No
ndb_log_orig	Yes	Yes	Yes	Global	No
ndb-log-transaction-id	Yes	Yes			No
- *Variable*: ndb_log_transaction_id			Yes	Global	No
ndb_log_transaction_id			Yes	Global	No
ndb-log-update-as-write	Yes	Yes			Yes
- *Variable*: ndb_log_update_as_write			Yes	Global	Yes
ndb-log-update-minimal	Yes	Yes			Yes
- *Variable*: ndb_log_update_minimal			Yes	Global	Yes
ndb_log_updated_only	Yes	Yes	Yes	Global	Yes
ndb_metadata_check	Yes	Yes	Yes	Global	Yes
ndb_metadata_check_interval	Yes	Yes	Yes	Global	Yes
ndb-optimization-delay	Yes	Yes			Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
- *Variable*: ndb_optimization_delay			Yes	Global	Yes
ndb_optimized_node_selection	Yes	Yes	Yes	Global	No
ndb_read_backup	Yes	Yes	Yes	Global	Yes
ndb_recv_thread_activation_threshold			Yes	Global	Yes
ndb_recv_thread_cpu_mask			Yes	Global	Yes
ndb_report_thresh_binlog_epoch_slip	Yes	Yes	Yes	Global	Yes
ndb_report_thresh_binlog_mem_usage	Yes	Yes	Yes	Global	Yes
ndb_row_checksum			Yes	Both	Yes
ndb_schema_dist_lock_wait_timeout	Yes	Yes	Yes	Global	Yes
ndb-schema-dist-timeout	Yes	Yes			No
- *Variable*: ndb_schema_dist_timeout			Yes	Global	No
ndb_schema_dist_timeout	Yes	Yes	Yes	Global	No
ndb_schema_dist_upgrade_allowed	Yes	Yes	Yes	Global	No
ndb_show_foreign_key_mock_tables	Yes	Yes	Yes	Global	Yes
ndb_slave_conflict_role	Yes	Yes	Yes	Global	Yes
Ndb_slave_max_replicated_epoch			Yes	Global	No
Ndb_system_name			Yes	Global	No
ndb_table_no_logging			Yes	Session	Yes
ndb_table_temporary			Yes	Session	Yes
ndb_use_copying_alter_table			Yes	Both	No
ndb_use_exact_count			Yes	Both	Yes
ndb_use_transactions	Yes	Yes	Yes	Both	Yes
ndb_version			Yes	Global	No
ndb_version_string			Yes	Global	No
ndb-wait-connected	Yes	Yes	Yes	Global	No
ndb-wait-setup	Yes	Yes	Yes	Global	No
ndbinfo_database			Yes	Global	No
ndbinfo_max_bytes	Yes		Yes	Both	Yes
ndbinfo_max_rows	Yes		Yes	Both	Yes
ndbinfo_offline			Yes	Global	Yes
ndbinfo_show_hidden	Yes		Yes	Both	Yes
ndbinfo_table_prefix	Yes		Yes	Both	Yes
ndbinfo_version			Yes	Global	No
net_buffer_length	Yes	Yes	Yes	Both	Yes
net_read_timeout	Yes	Yes	Yes	Both	Yes
net_retry_count	Yes	Yes	Yes	Both	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
net_write_timeout	Yes	Yes	Yes	Both	Yes
new	Yes	Yes	Yes	Both	Yes
ngram_token_size	Yes	Yes	Yes	Global	No
offline_mode	Yes	Yes	Yes	Global	Yes
old	Yes	Yes	Yes	Global	No
old_alter_table	Yes	Yes	Yes	Both	Yes
old_passwords	Yes	Yes	Yes	Both	Yes
open_files_limit	Yes	Yes	Yes	Global	No
optimizer_prune_level	Yes	Yes	Yes	Both	Yes
optimizer_search_depth	Yes	Yes	Yes	Both	Yes
optimizer_switch	Yes	Yes	Yes	Both	Yes
optimizer_trace	Yes	Yes	Yes	Both	Yes
optimizer_trace_features	Yes	Yes	Yes	Both	Yes
optimizer_trace_limit	Yes	Yes	Yes	Both	Yes
optimizer_trace_max_mem_size	Yes	Yes	Yes	Both	Yes
optimizer_trace_offset	Yes	Yes	Yes	Both	Yes
original_commit_timestamp			Yes	Session	Yes
original_server_version			Yes	Session	Yes
parser_max_mem_size	Yes	Yes	Yes	Both	Yes
partial_revokes	Yes	Yes	Yes	Global	Yes
password_history	Yes	Yes	Yes	Global	Yes
password_require_current	Yes	Yes	Yes	Global	Yes
password_reuse_interval	Yes	Yes	Yes	Global	Yes
performance_schema	Yes	Yes	Yes	Global	No
performance_schema_accounts_size	Yes	Yes	Yes	Global	No
performance_schema_digests_size	Yes	Yes	Yes	Global	No
performance_schema_error_size	Yes	Yes	Yes	Global	No
performance_schema_events_stages_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_stages_history_size	Yes	Yes	Yes	Global	No
performance_schema_events_statements_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_statements_history_size	Yes	Yes	Yes	Global	No
performance_schema_events_transactions_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_transactions_history_size	Yes	Yes	Yes	Global	No
performance_schema_events_waits_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_waits_history_size	Yes	Yes	Yes	Global	No
performance_schema_hosts_size	Yes	Yes	Yes	Global	No
performance_schema_max_cond_classes	Yes	Yes	Yes	Global	No
performance_schema_max_cond_instances	Yes	Yes	Yes	Global	No

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
performance_schema_max_digest_length	Yes	Yes	Yes	Global	No
performance_schema_max_digest_sample_age	Yes	Yes	Yes	Global	Yes
performance_schema_max_file_classes	Yes	Yes	Yes	Global	No
performance_schema_max_file_handles	Yes	Yes	Yes	Global	No
performance_schema_max_file_instances	Yes	Yes	Yes	Global	No
performance_schema_max_index_stat	Yes	Yes	Yes	Global	No
performance_schema_max_memory_classes	Yes	Yes	Yes	Global	No
performance_schema_max_metadata_locks	Yes	Yes	Yes	Global	No
performance_schema_max_mutex_classes	Yes	Yes	Yes	Global	No
performance_schema_max_mutex_instances	Yes	Yes	Yes	Global	No
performance_schema_max_prepared_statements_instances	Yes	Yes	Yes	Global	No
performance_schema_max_program_instances	Yes	Yes	Yes	Global	No
performance_schema_max_rwlock_classes	Yes	Yes	Yes	Global	No
performance_schema_max_rwlock_instances	Yes	Yes	Yes	Global	No
performance_schema_max_socket_classes	Yes	Yes	Yes	Global	No
performance_schema_max_socket_instances	Yes	Yes	Yes	Global	No
performance_schema_max_sql_text_length	Yes	Yes	Yes	Global	No
performance_schema_max_stage_classes	Yes	Yes	Yes	Global	No
performance_schema_max_statement_classes	Yes	Yes	Yes	Global	No
performance_schema_max_statement_stack	Yes	Yes	Yes	Global	No
performance_schema_max_table_handles	Yes	Yes	Yes	Global	No
performance_schema_max_table_instances	Yes	Yes	Yes	Global	No
performance_schema_max_table_lock_stat	Yes	Yes	Yes	Global	No
performance_schema_max_thread_classes	Yes	Yes	Yes	Global	No
performance_schema_max_thread_instances	Yes	Yes	Yes	Global	No
performance_schema_session_connect_attrs_size	Yes	Yes	Yes	Global	No
performance_schema_setup_actors_size	Yes	Yes	Yes	Global	No
performance_schema_setup_objects_size	Yes	Yes	Yes	Global	No
performance_schema_users_size	Yes	Yes	Yes	Global	No
persist_only_admin_x509_subject	Yes	Yes	Yes	Global	No
persisted_globals_load	Yes	Yes	Yes	Global	No
pid_file	Yes	Yes	Yes	Global	No
plugin_dir	Yes	Yes	Yes	Global	No
plugin-load	Yes	Yes	Yes	Global	No
plugin-load-add	Yes	Yes	Yes	Global	No
port	Yes	Yes	Yes	Global	No
preload_buffer_size	Yes	Yes	Yes	Both	Yes
print_identified_with_as_hex	Yes	Yes	Yes	Both	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
profiling			Yes	Both	Yes
profiling_history_size	Yes	Yes	Yes	Both	Yes
protocol_compression_algorithms	Yes	Yes	Yes	Global	Yes
protocol_version			Yes	Global	No
proxy_user			Yes	Session	No
pseudo_slave_mode			Yes	Session	Yes
pseudo_thread_id			Yes	Session	Yes
query_alloc_block_size	Yes	Yes	Yes	Both	Yes
query_cache_limit	Yes	Yes	Yes	Global	Yes
query_cache_min_res_unit	Yes	Yes	Yes	Global	Yes
query_cache_size	Yes	Yes	Yes	Global	Yes
query_cache_type	Yes	Yes	Yes	Both	Yes
query_cache_wlock_invalidate	Yes	Yes	Yes	Both	Yes
query_prealloc_size	Yes	Yes	Yes	Both	Yes
rand_seed1			Yes	Session	Yes
rand_seed2			Yes	Session	Yes
range_alloc_block_size	Yes	Yes	Yes	Both	Yes
range_optimizer_max_mem_size	Yes	Yes	Yes	Both	Yes
rbr_exec_mode			Yes	Both	Yes
read_buffer_size	Yes	Yes	Yes	Both	Yes
read_only	Yes	Yes	Yes	Global	Yes
read_rnd_buffer_size	Yes	Yes	Yes	Both	Yes
regexp_stack_limit	Yes	Yes	Yes	Global	Yes
regexp_time_limit	Yes	Yes	Yes	Global	Yes
relay_log	Yes	Yes	Yes	Global	No
relay_log_basename			Yes	Global	No
relay_log_index	Yes	Yes	Yes	Global	No
relay_log_info_file	Yes	Yes	Yes	Global	No
relay_log_info_repository	Yes	Yes	Yes	Global	Yes
relay_log_purge	Yes	Yes	Yes	Global	Yes
relay_log_recovery	Yes	Yes	Yes	Global	No
relay_log_space_limit	Yes	Yes	Yes	Global	No
report_host	Yes	Yes	Yes	Global	No
report_password	Yes	Yes	Yes	Global	No
report_port	Yes	Yes	Yes	Global	No
report_user	Yes	Yes	Yes	Global	No
require_secure_transport	Yes	Yes	Yes	Global	Yes
resultset_metadata			Yes	Session	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
rewriter_enabled			Yes	Global	Yes
rewriter_verbose			Yes	Global	Yes
rpl_read_size	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_master_enabled	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_master_timeout	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_master_trace_level	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_master_wait_for_slave_count	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_master_wait_no_slave	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_master_wait_point	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_slave_enabled	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_slave_trace_level	Yes	Yes	Yes	Global	Yes
rpl_stop_slave_timeout	Yes	Yes	Yes	Global	Yes
schema_definition_cache	Yes	Yes	Yes	Global	Yes
secondary_engine_cost_threshold			Yes	Session	Yes
secure-auth	Yes	Yes			Yes
- *Variable*: secure_auth			Yes	Global	Yes
secure_file_priv	Yes	Yes	Yes	Global	No
server-id	Yes	Yes			Yes
- *Variable*: server_id			Yes	Global	Yes
server-id-bits	Yes	Yes			No
- *Variable*: server_id_bits			Yes	Global	No
server_uuid			Yes	Global	No
session_track_gtids	Yes	Yes	Yes	Both	Yes
session_track_schema	Yes	Yes	Yes	Both	Yes
session_track_state_change	Yes	Yes	Yes	Both	Yes
session_track_system_variables	Yes	Yes	Yes	Both	Yes
session_track_transaction_info	Yes	Yes	Yes	Both	Yes
sha256_password_auto_generate_rsa_keys	Yes	Yes	Yes	Global	No
sha256_password_private_key_path	Yes	Yes	Yes	Global	No
sha256_password_proxy_users	Yes	Yes	Yes	Global	Yes
sha256_password_public_key_path	Yes	Yes	Yes	Global	No
shared_memory	Yes	Yes	Yes	Global	No
shared_memory_base_name	Yes	Yes	Yes	Global	No
show_compatibility_56	Yes	Yes	Yes	Global	Yes
show_create_table_skip_secondary_engine	Yes	Yes	Yes	Session	Yes
show_create_table_verbosity	Yes	Yes	Yes	Both	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
show_old_temporals	Yes	Yes	Yes	Both	Yes
skip_external_locking	Yes	Yes	Yes	Global	No
skip-name-resolve	Yes	Yes			No
- *Variable*: skip_name_resolve			Yes	Global	No
skip-networking	Yes	Yes			No
- *Variable*: skip_networking			Yes	Global	No
skip-show-database	Yes	Yes			No
- *Variable*: skip_show_database			Yes	Global	No
slave_allow_batching	Yes	Yes	Yes	Global	Yes
slave_checkpoint_group	Yes	Yes	Yes	Global	Yes
slave_checkpoint_period	Yes	Yes	Yes	Global	Yes
slave_compressed_protocol	Yes	Yes	Yes	Global	Yes
slave_exec_mode	Yes	Yes	Yes	Global	Yes
slave-load-tmpdir	Yes	Yes			No
- *Variable*: slave_load_tmpdir			Yes	Global	No
slave_max_allowed_packet	Yes	Yes	Yes	Global	Yes
slave-net-timeout	Yes	Yes			Yes
- *Variable*: slave_net_timeout			Yes	Global	Yes
slave_parallel_type	Yes	Yes	Yes	Global	Yes
slave_parallel_workers	Yes	Yes	Yes	Global	Yes
slave_pending_jobs_size_max	Yes	Yes	Yes	Global	Yes
slave_preserve_commit_order	Yes	Yes	Yes	Global	Yes
slave_rows_search_algorithms	Yes	Yes	Yes	Global	Yes
slave-skip-errors	Yes	Yes	Yes	Global	No
slave_sql_verify_checksum	Yes	Yes	Yes	Global	Yes
slave_transaction_retries	Yes	Yes	Yes	Global	Yes
slave_type_conversions	Yes	Yes	Yes	Global	No
slow_launch_time	Yes	Yes	Yes	Global	Yes
slow_query_log	Yes	Yes	Yes	Global	Yes
slow_query_log_file	Yes	Yes	Yes	Global	Yes
socket	Yes	Yes	Yes	Global	No
sort_buffer_size	Yes	Yes	Yes	Both	Yes
sql_auto_is_null			Yes	Both	Yes
sql_big_selects			Yes	Both	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
sql_buffer_result			Yes	Both	Yes
sql_log_bin			Yes	Session	Yes
sql_log_off			Yes	Both	Yes
sql-mode	Yes	Yes	Yes	Both	Yes
sql_notes			Yes	Both	Yes
sql_quote_show_create			Yes	Both	Yes
sql_require_primary_key	Yes	Yes	Yes	Both	Yes
sql_safe_updates			Yes	Both	Yes
sql_select_limit			Yes	Both	Yes
sql_slave_skip_counter			Yes	Global	Yes
sql_warnings			Yes	Both	Yes
ssl-ca	Yes	Yes	Yes	Global	Varies
ssl-capath	Yes	Yes	Yes	Global	Varies
ssl-cert	Yes	Yes	Yes	Global	Varies
ssl-cipher	Yes	Yes	Yes	Global	Varies
ssl-crl	Yes	Yes	Yes	Global	Varies
ssl-crlpath	Yes	Yes	Yes	Global	Varies
ssl_fips_mode	Yes	Yes	Yes	Global	Yes
ssl-key	Yes	Yes	Yes	Global	Varies
stored_program_cache	Yes	Yes	Yes	Global	Yes
stored_program_definition_cache	Yes	Yes	Yes	Global	Yes
super_read_only	Yes	Yes	Yes	Global	Yes
sync_binlog	Yes	Yes	Yes	Global	Yes
sync_master_info	Yes	Yes	Yes	Global	Yes
sync_relay_log	Yes	Yes	Yes	Global	Yes
sync_relay_log_info	Yes	Yes	Yes	Global	Yes
syseventlog.facility	Yes	Yes	Yes	Global	Yes
syseventlog.include_pid	Yes	Yes	Yes	Global	Yes
syseventlog.tag	Yes	Yes	Yes	Global	Yes
system_time_zone			Yes	Global	No
table_definition_cache	Yes	Yes	Yes	Global	Yes
table_encryption_privilege_check	Yes	Yes	Yes	Global	Yes
table_open_cache	Yes	Yes	Yes	Global	Yes
table_open_cache_instances	Yes	Yes	Yes	Global	No
tablespace_definition_cache	Yes	Yes	Yes	Global	Yes
temptable_max_ram	Yes	Yes	Yes	Global	Yes
temptable_use_mmap	Yes	Yes	Yes	Global	Yes
thread_cache_size	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
thread_handling	Yes	Yes	Yes	Global	No
thread_pool_algorithm	Yes	Yes	Yes	Global	No
thread_pool_high_priority_connection	Yes	Yes	Yes	Both	Yes
thread_pool_max_active_query_threads	Yes	Yes	Yes	Global	Yes
thread_pool_max_unused_threads	Yes	Yes	Yes	Global	Yes
thread_pool_prio_kickup_timer	Yes	Yes	Yes	Both	Yes
thread_pool_size	Yes	Yes	Yes	Global	No
thread_pool_stall_limit	Yes	Yes	Yes	Global	Yes
thread_stack	Yes	Yes	Yes	Global	No
time_format			Yes	Global	No
time_zone			Yes	Both	Yes
timestamp			Yes	Session	Yes
tls_ciphersuites	Yes	Yes	Yes	Global	Yes
tls_version	Yes	Yes	Yes	Global	Varies
tmp_table_size	Yes	Yes	Yes	Both	Yes
tmpdir	Yes	Yes	Yes	Global	No
transaction_alloc_block_size	Yes	Yes	Yes	Both	Yes
transaction_allow_batching			Yes	Session	Yes
transaction-isolation	Yes	Yes			Yes
- *Variable*: transaction_isolation			Yes	Both	Yes
transaction_prealloc_size	Yes	Yes	Yes	Both	Yes
transaction-read-only	Yes	Yes			Yes
- *Variable*: transaction_read_only			Yes	Both	Yes
transaction_write_set_extraction	Yes	Yes	Yes	Both	Yes
tx_isolation			Yes	Both	Yes
tx_read_only			Yes	Both	Yes
unique_checks			Yes	Both	Yes
updatable_views_with_limit	Yes	Yes	Yes	Both	Yes
use_secondary_engine			Yes	Session	Yes
validate_password_check_user_name	Yes	Yes	Yes	Global	Yes
validate_password_dictionary_file	Yes	Yes	Yes	Global	Yes
validate_password_length	Yes	Yes	Yes	Global	Yes
validate_password_mixed_case_count	Yes	Yes	Yes	Global	Yes
validate_password_number_count	Yes	Yes	Yes	Global	Yes
validate_password_policy	Yes	Yes	Yes	Global	Yes
validate_password_special_char_count	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
validate_password.check_user_name	Yes	Yes	Yes	Global	Yes
validate_password.dictionary_file	Yes	Yes	Yes	Global	Yes
validate_password.length	Yes	Yes	Yes	Global	Yes
validate_password.mixed_case_count	Yes	Yes	Yes	Global	Yes
validate_password.number_count	Yes	Yes	Yes	Global	Yes
validate_password.policy	Yes	Yes	Yes	Global	Yes
validate_password.special_char_count	Yes	Yes	Yes	Global	Yes
validate_user_plugins	Yes	Yes	Yes	Global	No
version			Yes	Global	No
version_comment			Yes	Global	No
version_compile_machine			Yes	Global	No
version_compile_os			Yes	Global	No
version_compile_zlib			Yes	Global	No
version_tokens_session	Yes	Yes	Yes	Both	Yes
version_tokens_session_number	Yes	Yes	Yes	Both	No
wait_timeout	Yes	Yes	Yes	Both	Yes
warning_count			Yes	Session	No
windowing_use_high_precision	Yes	Yes	Yes	Both	Yes

Notes:

1. This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

5.1.6 Server Status Variable Reference

The following table lists all status variables applicable within `mysqld`.

The table lists each variable's data type and scope. The last column indicates whether the scope for each variable is Global, Session, or both. Please see the corresponding item descriptions for details on setting and using the variables. Where appropriate, direct links to further information about the items are provided.

Table 5.3 Status Variable Summary

Variable Name	Variable Type	Variable Scope
Aborted_clients	Integer	Global
Aborted_connects	Integer	Global
Acl_cache_items_count	Integer	Global
Audit_log_current_size	Integer	Global
Audit_log_event_max_drop_size	Integer	Global
Audit_log_events	Integer	Global
Audit_log_events_filtered	Integer	Global
Audit_log_events_lost	Integer	Global
Audit_log_events_written	Integer	Global

Variable Name	Variable Type	Variable Scope
Audit_log_total_size	Integer	Global
Audit_log_write_waits	Integer	Global
Binlog_cache_disk_use	Integer	Global
Binlog_cache_use	Integer	Global
Binlog_stmt_cache_disk_use	Integer	Global
Binlog_stmt_cache_use	Integer	Global
Bytes_received	Integer	Both
Bytes_sent	Integer	Both
Caching_sha2_password_rsa_public_key	String	Global
Com_admin_commands	Integer	Both
Com_alter_db	Integer	Both
Com_alter_event	Integer	Both
Com_alter_function	Integer	Both
Com_alter_procedure	Integer	Both
Com_alter_resource_group	Integer	Global
Com_alter_server	Integer	Both
Com_alter_table	Integer	Both
Com_alter_tablespace	Integer	Both
Com_alter_user	Integer	Both
Com_alter_user_default_role	Integer	Global
Com_analyze	Integer	Both
Com_assign_to_keycache	Integer	Both
Com_begin	Integer	Both
Com_binlog	Integer	Both
Com_call_procedure	Integer	Both
Com_change_db	Integer	Both
Com_change_master	Integer	Both
Com_change_repl_filter	Integer	Both
Com_check	Integer	Both
Com_checksum	Integer	Both
Com_clone	Integer	Global
Com_commit	Integer	Both
Com_create_db	Integer	Both
Com_create_event	Integer	Both
Com_create_function	Integer	Both
Com_create_index	Integer	Both
Com_create_procedure	Integer	Both
Com_create_resource_group	Integer	Global

Variable Name	Variable Type	Variable Scope
Com_create_role	Integer	Global
Com_create_server	Integer	Both
Com_create_table	Integer	Both
Com_create_trigger	Integer	Both
Com_create_udf	Integer	Both
Com_create_user	Integer	Both
Com_create_view	Integer	Both
Com_dealloc_sql	Integer	Both
Com_delete	Integer	Both
Com_delete_multi	Integer	Both
Com_do	Integer	Both
Com_drop_db	Integer	Both
Com_drop_event	Integer	Both
Com_drop_function	Integer	Both
Com_drop_index	Integer	Both
Com_drop_procedure	Integer	Both
Com_drop_resource_group	Integer	Global
Com_drop_role	Integer	Global
Com_drop_server	Integer	Both
Com_drop_table	Integer	Both
Com_drop_trigger	Integer	Both
Com_drop_user	Integer	Both
Com_drop_view	Integer	Both
Com_empty_query	Integer	Both
Com_execute_sql	Integer	Both
Com_explain_other	Integer	Both
Com_flush	Integer	Both
Com_get_diagnostics	Integer	Both
Com_grant	Integer	Both
Com_grant_roles	Integer	Global
Com_group_replication_start	Integer	Global
Com_group_replication_stop	Integer	Global
Com_ha_close	Integer	Both
Com_ha_open	Integer	Both
Com_ha_read	Integer	Both
Com_help	Integer	Both
Com_insert	Integer	Both
Com_insert_select	Integer	Both

Variable Name	Variable Type	Variable Scope
Com_install_component	Integer	Global
Com_install_plugin	Integer	Both
Com_kill	Integer	Both
Com_load	Integer	Both
Com_lock_tables	Integer	Both
Com_optimize	Integer	Both
Com_preload_keys	Integer	Both
Com_prepare_sql	Integer	Both
Com_purge	Integer	Both
Com_purge_before_date	Integer	Both
Com_release_savepoint	Integer	Both
Com_rename_table	Integer	Both
Com_rename_user	Integer	Both
Com_repair	Integer	Both
Com_replace	Integer	Both
Com_replace_select	Integer	Both
Com_reset	Integer	Both
Com_resignal	Integer	Both
Com_revoke	Integer	Both
Com_revoke_all	Integer	Both
Com_revoke_roles	Integer	Global
Com_rollback	Integer	Both
Com_rollback_to_savepoint	Integer	Both
Com_savepoint	Integer	Both
Com_select	Integer	Both
Com_set_option	Integer	Both
Com_set_resource_group	Integer	Global
Com_set_role	Integer	Global
Com_show_authors	Integer	Both
Com_show_binlog_events	Integer	Both
Com_show_binlogs	Integer	Both
Com_show_charsets	Integer	Both
Com_show_collations	Integer	Both
Com_show_contributors	Integer	Both
Com_show_create_db	Integer	Both
Com_show_create_event	Integer	Both
Com_show_create_func	Integer	Both
Com_show_create_proc	Integer	Both

Variable Name	Variable Type	Variable Scope
Com_show_create_table	Integer	Both
Com_show_create_trigger	Integer	Both
Com_show_create_user	Integer	Both
Com_show_databases	Integer	Both
Com_show_engine_logs	Integer	Both
Com_show_engine_mutex	Integer	Both
Com_show_engine_status	Integer	Both
Com_show_errors	Integer	Both
Com_show_events	Integer	Both
Com_show_fields	Integer	Both
Com_show_function_code	Integer	Both
Com_show_function_status	Integer	Both
Com_show_grants	Integer	Both
Com_show_keys	Integer	Both
Com_show_master_status	Integer	Both
Com_show_ndb_status	Integer	Both
Com_show_open_tables	Integer	Both
Com_show_plugins	Integer	Both
Com_show_privileges	Integer	Both
Com_show_procedure_code	Integer	Both
Com_show_procedure_status	Integer	Both
Com_show_processlist	Integer	Both
Com_show_profile	Integer	Both
Com_show_profiles	Integer	Both
Com_show_relaylog_events	Integer	Both
Com_show_slave_hosts	Integer	Both
Com_show_slave_status	Integer	Both
Com_show_status	Integer	Both
Com_show_storage_engines	Integer	Both
Com_show_table_status	Integer	Both
Com_show_tables	Integer	Both
Com_show_triggers	Integer	Both
Com_show_variables	Integer	Both
Com_show_warnings	Integer	Both
Com_shutdown	Integer	Both
Com_signal	Integer	Both
Com_slave_start	Integer	Both
Com_slave_stop	Integer	Both

Variable Name	Variable Type	Variable Scope
Com_stmt_close	Integer	Both
Com_stmt_execute	Integer	Both
Com_stmt_fetch	Integer	Both
Com_stmt_prepare	Integer	Both
Com_stmt_reprepare	Integer	Both
Com_stmt_reset	Integer	Both
Com_stmt_send_long_data	Integer	Both
Com_truncate	Integer	Both
Com_uninstall_component	Integer	Global
Com_uninstall_plugin	Integer	Both
Com_unlock_tables	Integer	Both
Com_update	Integer	Both
Com_update_multi	Integer	Both
Com_xa_commit	Integer	Both
Com_xa_end	Integer	Both
Com_xa_prepare	Integer	Both
Com_xa_recover	Integer	Both
Com_xa_rollback	Integer	Both
Com_xa_start	Integer	Both
Compression	Integer	Session
Compression_algorithm	String	Global
Compression_level	Integer	Global
Connection_control_delay_generated	Integer	Global
Connection_errors_accept	Integer	Global
Connection_errors_internal	Integer	Global
Connection_errors_max_connections	Integer	Global
Connection_errors_peer_address	Integer	Global
Connection_errors_select	Integer	Global
Connection_errors_tcpwrap	Integer	Global
Connections	Integer	Global
Created_tmp_disk_tables	Integer	Both
Created_tmp_files	Integer	Global
Created_tmp_tables	Integer	Both
Current_tls_ca	File name	Global
Current_tls_capath	Directory name	Global
Current_tls_cert	File name	Global
Current_tls_cipher	String	Global
Current_tls_ciphersuites	String	Global

Variable Name	Variable Type	Variable Scope
Current_tls_crl	File name	Global
Current_tls_crlpath	Directory name	Global
Current_tls_key	File name	Global
Current_tls_version	String	Global
Delayed_errors	Integer	Global
Delayed_insert_threads	Integer	Global
Delayed_writes	Integer	Global
dragnet.Status	String	Global
Firewall_access_denied	Integer	Global
Firewall_access_granted	Integer	Global
Firewall_cached_entries	Integer	Global
Flush_commands	Integer	Global
group_replication_primary_member	String	Global
Handler_commit	Integer	Both
Handler_delete	Integer	Both
Handler_discover	Integer	Both
Handler_external_lock	Integer	Both
Handler_mrr_init	Integer	Both
Handler_prepare	Integer	Both
Handler_read_first	Integer	Both
Handler_read_key	Integer	Both
Handler_read_last	Integer	Both
Handler_read_next	Integer	Both
Handler_read_prev	Integer	Both
Handler_read_rnd	Integer	Both
Handler_read_rnd_next	Integer	Both
Handler_rollback	Integer	Both
Handler_savepoint	Integer	Both
Handler_savepoint_rollback	Integer	Both
Handler_update	Integer	Both
Handler_write	Integer	Both
Innodb_available_undo_logs	Integer	Global
Innodb_buffer_pool_bytes_data	Integer	Global
Innodb_buffer_pool_bytes_dirty	Integer	Global
Innodb_buffer_pool_dump_status	String	Global
Innodb_buffer_pool_load_status	String	Global
Innodb_buffer_pool_pages_data	Integer	Global
Innodb_buffer_pool_pages_dirty	Integer	Global

Variable Name	Variable Type	Variable Scope
Innodb_buffer_pool_pages_flushed	Integer	Global
Innodb_buffer_pool_pages_free	Integer	Global
Innodb_buffer_pool_pages_latched	Integer	Global
Innodb_buffer_pool_pages_misc	Integer	Global
Innodb_buffer_pool_pages_total	Integer	Global
Innodb_buffer_pool_read_ahead	Integer	Global
Innodb_buffer_pool_read_ahead_evicted	Integer	Global
Innodb_buffer_pool_read_ahead_rnd	Integer	Global
Innodb_buffer_pool_read_requests	Integer	Global
Innodb_buffer_pool_reads	Integer	Global
Innodb_buffer_pool_resize_status	String	Global
Innodb_buffer_pool_wait_free	Integer	Global
Innodb_buffer_pool_write_requests	Integer	Global
Innodb_data_fsyncs	Integer	Global
Innodb_data_pending_fsyncs	Integer	Global
Innodb_data_pending_reads	Integer	Global
Innodb_data_pending_writes	Integer	Global
Innodb_data_read	Integer	Global
Innodb_data_reads	Integer	Global
Innodb_data_writes	Integer	Global
Innodb_data_written	Integer	Global
Innodb_dblwr_pages_written	Integer	Global
Innodb_dblwr_writes	Integer	Global
Innodb_have_atomic_builtins	Integer	Global
Innodb_log_waits	Integer	Global
Innodb_log_write_requests	Integer	Global
Innodb_log_writes	Integer	Global
Innodb_num_open_files	Integer	Global
Innodb_os_log_fsyncs	Integer	Global
Innodb_os_log_pending_fsyncs	Integer	Global
Innodb_os_log_pending_writes	Integer	Global
Innodb_os_log_written	Integer	Global
Innodb_page_size	Integer	Global
Innodb_pages_created	Integer	Global
Innodb_pages_read	Integer	Global
Innodb_pages_written	Integer	Global
Innodb_row_lock_current_waits	Integer	Global
Innodb_row_lock_time	Integer	Global

Variable Name	Variable Type	Variable Scope
Innodb_row_lock_time_avg	Integer	Global
Innodb_row_lock_time_max	Integer	Global
Innodb_row_lock_waits	Integer	Global
Innodb_rows_deleted	Integer	Global
Innodb_rows_inserted	Integer	Global
Innodb_rows_read	Integer	Global
Innodb_rows_updated	Integer	Global
Innodb_system_rows_deleted	Integer	Global
Innodb_system_rows_inserted	Integer	Global
Innodb_system_rows_read	Integer	Global
Innodb_truncated_status_writes	Integer	Global
Key_blocks_not_flushed	Integer	Global
Key_blocks_unused	Integer	Global
Key_blocks_used	Integer	Global
Key_read_requests	Integer	Global
Key_reads	Integer	Global
Key_write_requests	Integer	Global
Key_writes	Integer	Global
Last_query_cost	Numeric	Session
Last_query_partial_plans	Integer	Session
Locked_connects	Integer	Global
Max_execution_time_exceeded	Integer	Both
Max_execution_time_set	Integer	Both
Max_execution_time_set_failed	Integer	Both
Max_used_connections	Integer	Global
Max_used_connections_time	Datetime	Global
mecab_charset	String	Global
Mysqlx_aborted_clients	Integer	Global
Mysqlx_address	String	Global
Mysqlx_bytes_received	Integer	Both
Mysqlx_bytes_sent	Integer	Both
Mysqlx_connection_accept_errors	Integer	Both
Mysqlx_connection_errors	Integer	Both
Mysqlx_connections_accepted	Integer	Global
Mysqlx_connections_closed	Integer	Global
Mysqlx_connections_rejected	Integer	Global
Mysqlx_crud_create_view	Integer	Both
Mysqlx_crud_delete	Integer	Both

Variable Name	Variable Type	Variable Scope
Mysqlx_crud_drop_view	Integer	Both
Mysqlx_crud_find	Integer	Both
Mysqlx_crud_insert	Integer	Both
Mysqlx_crud_modify_view	Integer	Both
Mysqlx_crud_update	Integer	Both
Mysqlx_cursor_close	Integer	Both
Mysqlx_cursor_fetch	Integer	Both
Mysqlx_cursor_open	Integer	Both
Mysqlx_errors_sent	Integer	Both
Mysqlx_errors_unknown_message_type	Integer	Both
Mysqlx_expect_close	Integer	Both
Mysqlx_expect_open	Integer	Both
Mysqlx_init_error	Integer	Both
Mysqlx_notice_global_sent	Integer	Both
Mysqlx_notice_other_sent	Integer	Both
Mysqlx_notice_warning_sent	Integer	Both
Mysqlx_notified_by_group_replication	Integer	Both
Mysqlx_port	String	Global
Mysqlx_prep_deallocate	Integer	Both
Mysqlx_prep_execute	Integer	Both
Mysqlx_prep_prepare	Integer	Both
Mysqlx_rows_sent	Integer	Both
Mysqlx_sessions	Integer	Global
Mysqlx_sessions_accepted	Integer	Global
Mysqlx_sessions_closed	Integer	Global
Mysqlx_sessions_fatal_error	Integer	Global
Mysqlx_sessions_killed	Integer	Global
Mysqlx_sessions_rejected	Integer	Global
Mysqlx_socket	String	Global
Mysqlx_ssl_accept_renegotiates	Integer	Global
Mysqlx_ssl_accepts	Integer	Global
Mysqlx_ssl_active	Integer	Both
Mysqlx_ssl_cipher	Integer	Both
Mysqlx_ssl_cipher_list	Integer	Both
Mysqlx_ssl_ctx_verify_depth	Integer	Both
Mysqlx_ssl_ctx_verify_mode	Integer	Both
Mysqlx_ssl_finished_accepts	Integer	Global
Mysqlx_ssl_server_not_after	Integer	Global

Variable Name	Variable Type	Variable Scope
Mysqlx_ssl_server_not_before	Integer	Global
Mysqlx_ssl_verify_depth	Integer	Global
Mysqlx_ssl_verify_mode	Integer	Global
Mysqlx_ssl_version	Integer	Both
Mysqlx_stmt_create_collection	Integer	Both
Mysqlx_stmt_create_collection_index	Integer	Both
Mysqlx_stmt_disable_notices	Integer	Both
Mysqlx_stmt_drop_collection	Integer	Both
Mysqlx_stmt_drop_collection_index	Integer	Both
Mysqlx_stmt_enable_notices	Integer	Both
Mysqlx_stmt_ensure_collection	String	Both
Mysqlx_stmt_execute_mysqlx	Integer	Both
Mysqlx_stmt_execute_sql	Integer	Both
Mysqlx_stmt_execute_xplugin	Integer	Both
Mysqlx_stmt_kill_client	Integer	Both
Mysqlx_stmt_list_clients	Integer	Both
Mysqlx_stmt_list_notices	Integer	Both
Mysqlx_stmt_list_objects	Integer	Both
Mysqlx_stmt_ping	Integer	Both
Mysqlx_worker_threads	Integer	Global
Mysqlx_worker_threads_active	Integer	Global
Ndb_api_bytes_received_count	Integer	Global
Ndb_api_bytes_received_count_session	Integer	Session
Ndb_api_bytes_received_count_slave	Integer	Global
Ndb_api_bytes_sent_count	Integer	Global
Ndb_api_bytes_sent_count_session	Integer	Session
Ndb_api_bytes_sent_count_slave	Integer	Global
Ndb_api_event_bytes_count	Integer	Global
Ndb_api_event_bytes_count_injector	Integer	Global
Ndb_api_event_data_count	Integer	Global
Ndb_api_event_data_count_injector	Integer	Global
Ndb_api_event_nondata_count	Integer	Global
Ndb_api_event_nondata_count_injector	Integer	Global
Ndb_api_pk_op_count	Integer	Global
Ndb_api_pk_op_count_session	Integer	Session
Ndb_api_pk_op_count_slave	Integer	Global
Ndb_api_pruned_scan_count	Integer	Global
Ndb_api_pruned_scan_count_session	Integer	Session

Variable Name	Variable Type	Variable Scope
Ndb_api_pruned_scan_count_slave	Integer	Global
Ndb_api_range_scan_count	Integer	Global
Ndb_api_range_scan_count_session	Integer	Session
Ndb_api_range_scan_count_slave	Integer	Global
Ndb_api_read_row_count	Integer	Global
Ndb_api_read_row_count_session	Integer	Session
Ndb_api_read_row_count_slave	Integer	Global
Ndb_api_scan_batch_count	Integer	Global
Ndb_api_scan_batch_count_session	Integer	Session
Ndb_api_scan_batch_count_slave	Integer	Global
Ndb_api_table_scan_count	Integer	Global
Ndb_api_table_scan_count_session	Integer	Session
Ndb_api_table_scan_count_slave	Integer	Global
Ndb_api_trans_abort_count	Integer	Global
Ndb_api_trans_abort_count_session	Integer	Session
Ndb_api_trans_abort_count_slave	Integer	Global
Ndb_api_trans_close_count	Integer	Global
Ndb_api_trans_close_count_session	Integer	Session
Ndb_api_trans_close_count_slave	Integer	Global
Ndb_api_trans_commit_count	Integer	Global
Ndb_api_trans_commit_count_session	Integer	Session
Ndb_api_trans_commit_count_slave	Integer	Global
Ndb_api_trans_local_read_row_count	Integer	Global
Ndb_api_trans_local_read_row_count_session	Integer	Session
Ndb_api_trans_local_read_row_count_slave	Integer	Global
Ndb_api_trans_start_count	Integer	Global
Ndb_api_trans_start_count_session	Integer	Session
Ndb_api_trans_start_count_slave	Integer	Global
Ndb_api_uk_op_count	Integer	Global
Ndb_api_uk_op_count_session	Integer	Session
Ndb_api_uk_op_count_slave	Integer	Global
Ndb_api_wait_exec_complete_count	Integer	Global
Ndb_api_wait_exec_complete_count_session	Integer	Session
Ndb_api_wait_exec_complete_count_slave	Integer	Global
Ndb_api_wait_meta_request_count	Integer	Global
Ndb_api_wait_meta_request_count_session	Integer	Session
Ndb_api_wait_meta_request_count_slave	Integer	Global
Ndb_api_wait_nanos_count	Integer	Global

Variable Name	Variable Type	Variable Scope
Ndb_api_wait_nanos_count_session	Integer	Session
Ndb_api_wait_nanos_count_slave	Integer	Global
Ndb_api_wait_scan_result_count	Integer	Global
Ndb_api_wait_scan_result_count_session	Integer	Session
Ndb_api_wait_scan_result_count_slave	Integer	Global
Ndb_cluster_node_id	Integer	Global
Ndb_config_from_host	Integer	Both
Ndb_config_from_port	Integer	Both
Ndb_conflict_fn_epoch	Integer	Global
Ndb_conflict_fn_epoch_trans	Integer	Global
Ndb_conflict_fn_epoch2	Integer	Global
Ndb_conflict_fn_epoch2_trans	Integer	Global
Ndb_conflict_fn_max	Integer	Global
Ndb_conflict_fn_old	Integer	Global
Ndb_conflict_last_stable_epoch	Integer	Global
Ndb_conflict_reflected_op_discard_count	Integer	Global
Ndb_conflict_reflected_op_prepare_count	Integer	Global
Ndb_conflict_refresh_op_count	Integer	Global
Ndb_conflict_trans_conflict_commit_count	Integer	Global
Ndb_conflict_trans_detect_iter_count	Integer	Global
Ndb_conflict_trans_reject_count	Integer	Global
Ndb_conflict_trans_row_conflict_count	Integer	Global
Ndb_conflict_trans_row_reject_count	Integer	Global
Ndb_epoch_delete_delete_count	Integer	Global
Ndb_execute_count	Integer	Global
Ndb_last_commit_epoch_server	Integer	Global
Ndb_last_commit_epoch_session	Integer	Session
Ndb_metadata_blacklist_size	Integer	Global
Ndb_metadata_detected_count	Integer	Global
Ndb_metadata_synced_count	Integer	Global
Ndb_cluster_node_id	Integer	Global
Ndb_number_of_data_nodes	Integer	Global
Ndb_pruned_scan_count	Integer	Global
Ndb_pushed_queries_defined	Integer	Global
Ndb_pushed_queries_dropped	Integer	Global
Ndb_pushed_queries_executed	Integer	Global
Ndb_pushed_reads	Integer	Global
Ndb_scan_count	Integer	Global

Variable Name	Variable Type	Variable Scope
Ndb_trans_hint_count_session	Integer	Both
Not_flushed_delayed_rows	Integer	Global
Ongoing_anonymous_gtid_violating_transaction_count	Integer	Global
Ongoing_anonymous_transaction_count	Integer	Global
Ongoing_automatic_gtid_violating_transaction_count	Integer	Global
Open_files	Integer	Global
Open_streams	Integer	Global
Open_table_definitions	Integer	Global
Open_tables	Integer	Both
Opened_files	Integer	Global
Opened_table_definitions	Integer	Both
Opened_tables	Integer	Both
Performance_schema_accounts_lost	Integer	Global
Performance_schema_cond_classes_lost	Integer	Global
Performance_schema_cond_instances_lost	Integer	Global
Performance_schema_digest_lost	Integer	Global
Performance_schema_file_classes_lost	Integer	Global
Performance_schema_file_handles_lost	Integer	Global
Performance_schema_file_instances_lost	Integer	Global
Performance_schema_hosts_lost	Integer	Global
Performance_schema_index_stat_lost	Integer	Global
Performance_schema_locker_lost	Integer	Global
Performance_schema_memory_classes_lost	Integer	Global
Performance_schema_metadata_lock_lost	Integer	Global
Performance_schema_mutex_classes_lost	Integer	Global
Performance_schema_mutex_instances_lost	Integer	Global
Performance_schema_nested_statement_lost	Integer	Global
Performance_schema_prepared_statements_lost	Integer	Global
Performance_schema_program_lost	Integer	Global
Performance_schema_rwlock_classes_lost	Integer	Global
Performance_schema_rwlock_instances_lost	Integer	Global
Performance_schema_session_connect_attrs_longest_seen	Integer	Global
Performance_schema_session_connect_attrs_lost	Integer	Global
Performance_schema_socket_classes_lost	Integer	Global
Performance_schema_socket_instances_lost	Integer	Global
Performance_schema_stage_classes_lost	Integer	Global
Performance_schema_statement_classes_lost	Integer	Global
Performance_schema_table_handles_lost	Integer	Global

Variable Name	Variable Type	Variable Scope
Performance_schema_table_instances_lost	Integer	Global
Performance_schema_table_lock_stat_lost	Integer	Global
Performance_schema_thread_classes_lost	Integer	Global
Performance_schema_thread_instances_lost	Integer	Global
Performance_schema_users_lost	Integer	Global
Prepared_stmt_count	Integer	Global
Qcache_free_blocks	Integer	Global
Qcache_free_memory	Integer	Global
Qcache_hits	Integer	Global
Qcache_inserts	Integer	Global
Qcache_lowmem_prunes	Integer	Global
Qcache_not_cached	Integer	Global
Qcache_queries_in_cache	Integer	Global
Qcache_total_blocks	Integer	Global
Queries	Integer	Both
Questions	Integer	Both
Rewriter_number_loaded_rules	Integer	Global
Rewriter_number_reloads	Integer	Global
Rewriter_number_rewritten_queries	Integer	Global
Rewriter_reload_error	Boolean	Global
Rpl_semi_sync_master_clients	Integer	Global
Rpl_semi_sync_master_net_avg_wait_time	Integer	Global
Rpl_semi_sync_master_net_wait_time	Integer	Global
Rpl_semi_sync_master_net_waits	Integer	Global
Rpl_semi_sync_master_no_times	Integer	Global
Rpl_semi_sync_master_no_tx	Integer	Global
Rpl_semi_sync_master_status	Boolean	Global
Rpl_semi_sync_master_timefunc_failures	Integer	Global
Rpl_semi_sync_master_tx_avg_wait_time	Integer	Global
Rpl_semi_sync_master_tx_wait_time	Integer	Global
Rpl_semi_sync_master_tx_waits	Integer	Global
Rpl_semi_sync_master_wait_pos_backtraverse	Integer	Global
Rpl_semi_sync_master_wait_sessions	Integer	Global
Rpl_semi_sync_master_yes_tx	Integer	Global
Rpl_semi_sync_slave_status	Boolean	Global
Rsa_public_key	String	Global
Secondary_engine_execution_count	Integer	Both
Select_full_join	Integer	Both

Variable Name	Variable Type	Variable Scope
Select_full_range_join	Integer	Both
Select_range	Integer	Both
Select_range_check	Integer	Both
Select_scan	Integer	Both
Slave_heartbeat_period	Numeric	Global
Slave_last_heartbeat	Datetime	Global
Slave_open_temp_tables	Integer	Global
Slave_received_heartbeats	Integer	Global
Slave_retried_transactions	Integer	Global
Slave_rows_last_search_algorithm_used	String	Global
Slave_running	String	Global
Slow_launch_threads	Integer	Both
Slow_queries	Integer	Both
Sort_merge_passes	Integer	Both
Sort_range	Integer	Both
Sort_rows	Integer	Both
Sort_scan	Integer	Both
Ssl_accept_renegotiates	Integer	Global
Ssl_accepts	Integer	Global
Ssl_callback_cache_hits	Integer	Global
Ssl_cipher	String	Both
Ssl_cipher_list	String	Both
Ssl_client_connects	Integer	Global
Ssl_connect_renegotiates	Integer	Global
Ssl_ctx_verify_depth	Integer	Global
Ssl_ctx_verify_mode	Integer	Global
Ssl_default_timeout	Integer	Both
Ssl_finished_accepts	Integer	Global
Ssl_finished_connects	Integer	Global
Ssl_server_not_after	Integer	Both
Ssl_server_not_before	Integer	Both
Ssl_session_cache_hits	Integer	Global
Ssl_session_cache_misses	Integer	Global
Ssl_session_cache_mode	String	Global
Ssl_session_cache_overflows	Integer	Global
Ssl_session_cache_size	Integer	Global
Ssl_session_cache_timeouts	Integer	Global
Ssl_sessions_reused	Integer	Both

Variable Name	Variable Type	Variable Scope
Ssl_used_session_cache_entries	Integer	Global
Ssl_verify_depth	Integer	Both
Ssl_verify_mode	Integer	Both
Ssl_version	String	Both
Table_locks_immediate	Integer	Global
Table_locks_waited	Integer	Global
Table_open_cache_hits	Integer	Both
Table_open_cache_misses	Integer	Both
Table_open_cache_overflows	Integer	Both
Tc_log_max_pages_used	Integer	Global
Tc_log_page_size	Integer	Global
Tc_log_page_waits	Integer	Global
Threads_cached	Integer	Global
Threads_connected	Integer	Global
Threads_created	Integer	Global
Threads_running	Integer	Global
Uptime	Integer	Global
Uptime_since_flush_status	Integer	Global
validate_password_dictionary_file_last_parsed	Datetime	Global
validate_password_dictionary_file_words_count	Integer	Global
validate_password.dictionary_file_last_parsed	Datetime	Global
validate_password.dictionary_file_words_count	Integer	Global

5.1.7 Server Command Options

When you start the `mysqld` server, you can specify program options using any of the methods described in Section 4.2.2, “Specifying Program Options”. The most common methods are to provide options in an option file or on the command line. However, in most cases it is desirable to make sure that the server uses the same options each time it runs. The best way to ensure this is to list them in an option file. See Section 4.2.2.2, “Using Option Files”. That section also describes option file format and syntax.

`mysqld` reads options from the `[mysqld]` and `[server]` groups. `mysqld_safe` reads options from the `[mysqld]`, `[server]`, `[mysqld_safe]`, and `[safe_mysqld]` groups. `mysql.server` reads options from the `[mysqld]` and `[mysql.server]` groups.

`mysqld` accepts many command options. For a brief summary, execute this command:

```
mysqld --help
```

To see the full list, use this command:

```
mysqld --verbose --help
```

Some of the items in the list are actually system variables that can be set at server startup. These can be displayed at runtime using the `SHOW VARIABLES` statement. Some items displayed by the preceding

`mysqld` command do not appear in `SHOW VARIABLES` output; this is because they are options only and not system variables.

The following list shows some of the most common server options. Additional options are described in other sections:

- Options that affect security: See Section 6.1.4, “Security-Related mysqld Options and Variables”.
- SSL-related options: See Command Options for Encrypted Connections.
- Binary log control options: See Section 5.4.4, “The Binary Log”.
- Replication-related options: See Section 17.1.6, “Replication and Binary Logging Options and Variables”.
- Options for loading plugins such as pluggable storage engines: See Section 5.6.1, “Installing and Uninstalling Plugins”.
- Options specific to particular storage engines: See Section 15.13, “InnoDB Startup Options and System Variables” and Section 16.2.1, “MyISAM Startup Options”.

Some options control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to an option that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server will adjust a value upward. For example, if you assign a value of 0 to an option for which the minimal value is 1024, the server will set the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some options take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued option is given as a relative path name, it will be located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

You can also set the values of server system variables at server startup by using variable names as options. To assign a value to a server system variable, use an option of the form `--var_name=value`. For example, `--sort_buffer_size=384M` sets the `sort_buffer_size` variable to a value of 384MB.

When you assign a value to a variable, MySQL might automatically correct the value to stay within a given range, or adjust the value to the closest permissible value if only certain values are permitted.

To restrict the maximum value to which a system variable can be set at runtime with the `SET` statement, specify this maximum by using an option of the form `--maximum-var_name=value` at server startup.

You can change the values of most system variables at runtime with the `SET` statement. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

Section 5.1.8, “Server System Variables”, provides a full description for all variables, and additional information for setting them at server startup and runtime. For information on changing system variables, see Section 5.1.1, “Configuring the Server”.

- `--help`, `-?`

Property	Value
Command-Line Format	`--help`

Display a short help message and exit. Use both the `--verbose` and `--help` options to see the full message.

- `--allow-suspicious-udfs`

Property	Value
Command-Line Format	`--allow-suspicious-udfs[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. See Section 29.4.2.6, “UDF Security Precautions”.

- `--ansi`

Property	Value
Command-Line Format	`--ansi`

Use standard (ANSI) SQL syntax instead of MySQL syntax. For more precise control over the server SQL mode, use the `--sql-mode` option instead. See Section 1.8, “MySQL Standards Compliance”, and Section 5.1.11, “Server SQL Modes”.

- `--basedir=dir_name`, `-b dir_name`

Property	Value
Command-Line Format	`--basedir=dir_name`
System Variable	`basedir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name
Default Value (>= 8.0.2)	`parent of mysqld installation directory`
Default Value (<= 8.0.1)	`configuration-dependent default`

The path to the MySQL installation directory. This option sets the `basedir` system variable.

The server executable determines its own full path name at startup and uses the parent of the directory in which it is located as the default `basedir` value. This in turn enables the server to use that `basedir` when searching for server-related information such as the `share` directory containing error messages.

- `--character-set-client-handshake`

Property	Value
Command-Line Format	`--character-set-client-handshake[={OFF|ON}]`
Type	Boolean
Default Value	`ON`

Do not ignore character set information sent by the client. To ignore client information and use the default server character set, use `--skip-character-set-client-handshake`; this makes MySQL behave like MySQL 4.0.

- `--chroot=dir_name`, `-r dir_name`

Property	Value
Command-Line Format	`--chroot=dir_name`
Type	Directory name

Put the `mysqld` server in a closed environment during startup by using the `chroot()` system call. This is a recommended security measure. Use of this option somewhat limits `LOAD DATA` and `SELECT ... INTO OUTFILE`.

- `--console`

Property	Value
Command-Line Format	`--console`
Platform Specific	Windows

(Windows only.) Cause the default error log destination to be the console. This affects log writers that base their own output destination on the default destination. See Section 5.4.2, “The Error Log”. `mysqld` does not close the console window if this option is used.

`--console` takes precedence over `--log-error` if both are given.

- `--core-file`

Property	Value
Command-Line Format	`--core-file[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Write a core file if `mysqld` dies. The name and location of the core file is system dependent. On Linux, a core file named `core.pid` is written to the current working directory of the process, which for `mysqld` is the data directory. *`pid`* represents the process ID of the server process. On macOS, a core file named `core.pid` is written to the `/cores` directory. On Solaris, use the `coreadm` command to specify where to write the core file and how to name it.

For some systems, to get a core file you must also specify the `--core-file-size` option to `mysqld_safe`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”. On some systems, such as Solaris, you do not get a core file if you are also using the `--user` option. There might be additional restrictions or limitations. For example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation.

The `innodb_buffer_pool_in_core_file` variable can be used to reduce the size of core files on operating systems that support it. For more information, see Section 15.8.3.7, “Excluding Buffer Pool Pages from Core Files”.

- `--daemonize`, `-D`

Property	Value
Command-Line Format	`--daemonize[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

This option causes the server to run as a traditional, forking daemon, permitting it to work with operating systems that use systemd for process control. For more information, see Section 2.5.9, “Managing MySQL Server with systemd”.

`--daemonize` is mutually exclusive with `--initialize` and `--initialize-insecure`.

If the server is started using the `--daemonize` option and is not connected to a tty device, a default error logging option of `--log-error=""` is used in the absence of an explicit logging option, to direct error output to the default log file.

`-D` is a synonym for `--daemonize`.

- `--datadir=`*`dir_name`*, `-h` *`dir_name`*

Property	Value
Command-Line Format	`--datadir=dir_name`
System Variable	`datadir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

The path to the MySQL server data directory. This option sets the `datadir` system variable. See the description of that variable.

- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`

Property	Value
Command-Line Format	`--debug[=debug_options]`
System Variable	`debug`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (Windows)	`d:t:i:O,\mysqld.trace`
Default Value (Unix)	`d:t:i:o,/tmp/mysqld.trace`

If MySQL is configured with the `-DWITH_DEBUG=1 CMake` option, you can use this option to get a trace file of what `mysqld` is doing. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:i:o,/tmp/mysqld.trace` on Unix and `d:t:i:O,\mysqld.trace` on Windows.

Using `-DWITH_DEBUG=1` to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

This option may be given multiple times. Values that begin with `+` or `-` are added to or subtracted from the previous value. For example, `--debug=T --debug=+P` sets the value to `P:T`.

For more information, see Section 29.5.4, “The DBUG Package”.

- `--debug-sync-timeout[=N]`

Property	Value
Command-Line Format	`--debug-sync-timeout[=#]`
Type	Integer

Controls whether the Debug Sync facility for testing and debugging is enabled. Use of Debug Sync requires that MySQL be configured with the `-DENABLE_DEBUG_SYNC=1 CMake` option (see Section 2.9.7, “MySQL Source-Configuration Options”). If Debug Sync is not compiled in, this option is not available. The option value is a timeout in seconds. The default value is 0, which disables Debug Sync. To enable it, specify a value greater than 0; this value also becomes the default timeout for individual synchronization points. If the option is given without a value, the timeout is set to 300 seconds.

For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `--default-time-zone=timezone`

Property	Value
Command-Line Format	`--default-time-zone=name`
Type	String

Set the default server time zone. This option sets the global `time_zone` system variable. If this option is not given, the default time zone is the same as the system time zone (given by the value of the `system_time_zone` system variable.

- `--defaults-extra-file=file_name`

Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name. This must be the first option on the command line if it is used.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

Read only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

Exception: Even with `--defaults-file`, `mysqld` reads `mysqld-auto.cnf`.

Note

This must be the first option on the command line if it is used, except that if the server is started with the `--defaults-file` and `--install` (or `--install-manual`) options, `--install` (or `--install-manual`) must be first.

For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--defaults-group-suffix=`*`str`*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqld` normally reads the `[mysqld]` group. If the `--defaults-group-suffix=_other` option is given, `mysqld` also reads the `[mysqld_other]` group.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--des-key-file=`*`file_name`*

Property	Value
Command-Line Format	`--des-key-file=file_name`
Deprecated	Yes (removed in 8.0.3)

  This option was removed in MySQL 8.0.3.

- `--early-plugin-load=`*`plugin_list`*

Property	Value
Command-Line Format	`--early-plugin-load=plugin_list`
Type	String
Default Value	`empty string`

  This option tells the server which plugins to load before loading mandatory built-in plugins and before storage engine initialization. If multiple `--early-plugin-load` options are given, only the last one is used.

  The option value is a semicolon-separated list of *`name=plugin_library`* and *`plugin_library`* values. Each *`name`* is the name of a plugin to load, and *`plugin_library`* is the name of the library file that contains the plugin code. If a plugin library is named without any preceding plugin name, the server loads all plugins in the library. The server looks for plugin library files in the directory named by the `plugin_dir` system variable.

  For example, if plugins named `myplug1` and `myplug2` have library files `myplug1.so` and `myplug2.so`, use this option to perform an early plugin load:

```
shell> mysqld --early-plugin-load="myplug1=myplug1.so;myplug2=myplug2.so"
```

  Quotes are used around the argument value because otherwise a semicolon (`;`) is interpreted as a special character by some command interpreters. (Unix shells treat it as a command terminator, for example.)

Each named plugin is loaded early for a single invocation of `mysqld` only. After a restart, the plugin is not loaded early unless `--early-plugin-load` is used again.

If the server is started using `--initialize` or `--initialize-insecure`, plugins specified by `--early-plugin-load` are not loaded.

If the server is run with `--help`, plugins specified by `--early-plugin-load` are loaded but not initialized. This behavior ensures that plugin options are displayed in the help message.

The default `--early-plugin-load` value is empty. To load the `keyring_file` plugin, you must use an explicit `--early-plugin-load` option with a nonempty value.

The `InnoDB` tablespace encryption feature relies on the `keyring_file` plugin for encryption key management, and the `keyring_file` plugin must be loaded prior to storage engine initialization to facilitate `InnoDB` recovery for encrypted tables. Administrators who want the `keyring_file` plugin loaded at startup should use the appropriate nonempty option value (for example, `keyring_file.so` on Unix and Unix-like systems and `keyring_file.dll` on Windows).

For information about `InnoDB` tablespace encryption, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”. For general information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

- `--exit-info[=flags]`, `-T [flags]`

Property	Value
Command-Line Format	`--exit-info[=flags]`
Type	Integer

This is a bitmask of different flags that you can use for debugging the `mysqld` server. Do not use this option unless you know *exactly* what it does!

- `--external-locking`

Property	Value
Command-Line Format	`--external-locking[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Enable external locking (system locking), which is disabled by default. If you use this option on a system on which `lockd` does not fully work (such as Linux), it is easy for `mysqld` to deadlock.

To disable external locking explicitly, use `--skip-external-locking`.

External locking affects only `MyISAM` table access. For more information, including conditions under which it can and cannot be used, see Section 8.11.5, “External Locking”.

- `--flush`

Property	Value
Command-Line Format	`--flush[={OFF|ON}]`
System Variable	`flush`
Scope	Global

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Flush (synchronize) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See Section B.4.3.3, “What to Do If MySQL Keeps Crashing”.

Note

If `--flush` is specified, the value of `flush_time` does not matter and changes to `flush_time` have no effect on flush behavior.

- `--gdb`

Property	Value
Command-Line Format	`--gdb[={OFF|ON}]`
Type	Boolean
Default Value	OFF

Install an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling. See Section 29.5, “Debugging and Porting MySQL”.

On Windows, this option also suppresses the forking that is used to implement the `RESTART` statement: Forking enables one process to act as a monitor to the other, which acts as the server. However, forking makes determining the server process to attach to for debugging more difficult, so starting the server with `--gdb` suppresses forking. For a server started with this option, `RESTART` simply exits and does not restart.

In non-debug settings, `--no-monitor` may be used to suppress forking the monitor process.

- `--initialize`, `-I`

Property	Value
Command-Line Format	`--initialize[={OFF|ON}]`
Type	Boolean
Default Value	OFF

This option is used to initialize a MySQL installation by creating the data directory and populating the tables in the `mysql` system database. For more information, see Section 2.10.1, “Initializing the Data Directory”.

When the server is started with `--initialize`, some functionality is unavailable that limits the statements permitted in any file named by the `init_file` system variable. For more information, see the description of that variable. In addition, the `disabled_storage_engines` system variable has no effect.

The `--ndbcluster` option is ignored when used together with `--initialize`.

`--initialize` is mutually exclusive with `--daemonize`.

`-I` is a synonym for `--initialize`.

- `--initialize-insecure`

Property	Value
Command-Line Format	`--initialize-insecure[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

This option is used to initialize a MySQL installation by creating the data directory and populating the tables in the `mysql` system database. This option implies `--initialize`. For more information, see the description of that option, and Section 2.10.1, “Initializing the Data Directory”.

`--initialize-insecure` is mutually exclusive with `--daemonize`.

- `--innodb-xxx`

Set an option for the `InnoDB` storage engine. The `InnoDB` options are listed in Section 15.13, “InnoDB Startup Options and System Variables”.

- `--install [service_name]`

Property	Value
Command-Line Format	`--install [service_name]`
Platform Specific	Windows

(Windows only) Install the server as a Windows service that starts automatically during Windows startup. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.3.4.8, “Starting MySQL as a Windows Service”.

Note

If the server is started with the `--defaults-file` and `--install` options, `--install` must be first.

- `--install-manual [service_name]`

Property	Value
Command-Line Format	`--install-manual [service_name]`
Platform Specific	Windows

(Windows only) Install the server as a Windows service that must be started manually. It does not start automatically during Windows startup. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.3.4.8, “Starting MySQL as a Windows Service”.

Note

If the server is started with the `--defaults-file` and `--install-manual` options, `--install-manual` must be first.

- `--language=lang_name, -L lang_name`

Property	Value
Command-Line Format	`--language=name`
Deprecated	Yes; use `lc-messages-dir` instead
System Variable	`language`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name
Default Value	`/usr/local/mysql/share/mysql/english/`

The language to use for error messages. *`lang_name`* can be given as the language name or as the full path name to the directory where the language files are installed. See Section 10.12, “Setting the Error Message Language”.

`--lc-messages-dir` and `--lc-messages` should be used rather than `--language`, which is deprecated (and handled as an alias for `--lc-messages-dir`). The `--language` option will be removed in a future MySQL release.

- `--large-pages`

Property	Value
Command-Line Format	`--large-pages[={OFF|ON}]`
System Variable	`large_pages`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Platform Specific	Linux
Type	Boolean
Default Value	`OFF`

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and operating system. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

MySQL supports the Linux implementation of large page support (which is called HugeTLB in Linux). See Section 8.12.3.2, “Enabling Large Page Support”. For Solaris support of large pages, see the description of the `--super-large-pages` option.

`--large-pages` is disabled by default.

- `--lc-messages=`*`locale_name`*

Property	Value
Command-Line Format	`--lc-messages=name`
System Variable	`lc_messages`

Property	Value
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	en_US

The locale to use for error messages. The default is en_US. The server converts the argument to a language name and combines it with the value of --lc-messages-dir to produce the location for the error message file. See Section 10.12, “Setting the Error Message Language”.

- --lc-messages-dir=*dir_name*

Property	Value
Command-Line Format	--lc-messages-dir=dir_name
System Variable	lc_messages_dir
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

The directory where error messages are located. The server uses the value together with the value of --lc-messages to produce the location for the error message file. See Section 10.12, “Setting the Error Message Language”.

- --local-service

Property	Value
Command-Line Format	--local-service

(Windows only) A --local-service option following the service name causes the server to run using the LocalService Windows account that has limited system privileges. If both --defaults-file and --local-service are given following the service name, they can be in any order. See Section 2.3.4.8, “Starting MySQL as a Windows Service”.

- --log-error[=*file_name*]

Property	Value
Command-Line Format	--log-error[=file_name]
System Variable	log_error
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

Set the default error log destination to the named file. This affects log writers that base their own output destination on the default destination. See Section 5.4.2, “The Error Log”.

If the option names no file, the default error log destination on Unix and Unix-like systems is a file named *`host_name`*`.err` in the data directory. The default destination on Windows is the same, unless the `--pid-file` option is specified. In that case, the file name is the PID file base name with a suffix of `.err` in the data directory.

If the option names a file, the default destination is that file (with an `.err` suffix added if the name has no suffix), located under the data directory unless an absolute path name is given to specify a different location.

If error log output cannot be redirected to the error log file, an error occurs and startup fails.

On Windows, `--console` takes precedence over `--log-error` if both are given. In this case, the default error log destination is the console rather than a file.

- `--log-isam[=`*`file_name`*`]`

Property	Value
Command-Line Format	`--log-isam[=file_name]`
Type	File name

Log all `MyISAM` changes to this file (used only when debugging `MyISAM`).

- `--log-raw`

Property	Value
Command-Line Format	`--log-raw[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Passwords in certain statements written to the general query log, slow query log, and binary log are rewritten by the server not to occur literally in plain text. Password rewriting can be suppressed for the general query log by starting the server with the `--log-raw` option. This option may be useful for diagnostic purposes, to see the exact text of statements as received by the server, but for security reasons is not recommended for production use.

If a query rewrite plugin is installed, the `--log-raw` option affects statement logging as follows:

- Without `--log-raw`, the server logs the statement returned by the query rewrite plugin. This may differ from the statement as received.
- With `--log-raw`, the server logs the original statement as received.

For more information, see Section 6.1.2.3, “Passwords and Logging”.

- `--log-short-format`

Property	Value
Command-Line Format	`--log-short-format[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Log less information to the slow query log, if it has been activated.

- `--log-tc=file_name`

Property	Value
Command-Line Format	`--log-tc=file_name`
Type	File name
Default Value	`tc.log`

The name of the memory-mapped transaction coordinator log file (for XA transactions that affect multiple storage engines when the binary log is disabled). The default name is `tc.log`. The file is created under the data directory if not given as a full path name. This option is unused.

- `--log-tc-size=size`

Property	Value
Command-Line Format	`--log-tc-size=#`
Type	Integer
Default Value	`6 * page size`
Minimum Value	`6 * page size`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The size in bytes of the memory-mapped transaction coordinator log. The default and minimum values are 6 times the page size, and the value must be a multiple of the page size.

- `--log-warnings[=level]`, `-W [level]`

Property	Value
Command-Line Format	`--log-warnings[=#]`
Deprecated	Yes (removed in 8.0.3)
System Variable	`log_warnings`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`2`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

This option was removed in MySQL 8.0.3. Use the `log_error_verbosity` system variable instead.

- `--memlock`

Property	Value
Command-Line Format	`--memlock[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Lock the `mysqld` process in memory. This option might help if you have a problem where the operating system is causing `mysqld` to swap to disk.

`--memlock` works on systems that support the `mlockall()` system call; this includes Solaris, most Linux distributions that use a 2.4 or higher kernel, and perhaps other Unix systems. On Linux systems, you can tell whether or not `mlockall()` (and thus this option) is supported by checking to see whether or not it is defined in the system `mman.h` file, like this:

```
shell> grep mlockall /usr/include/sys/mman.h
```

If `mlockall()` is supported, you should see in the output of the previous command something like the following:

```
extern int mlockall (int __flags) __THROW;
```

Important

Use of this option may require you to run the server as `root`, which, for reasons of security, is normally not a good idea. See Section 6.1.5, “How to Run MySQL as a Normal User”.

On Linux and perhaps other systems, you can avoid the need to run the server as `root` by changing the `limits.conf` file. See the notes regarding the memlock limit in Section 8.12.3.2, “Enabling Large Page Support”.

You must not try to use this option on a system that does not support the `mlockall()` system call; if you do so, `mysqld` will very likely crash as soon as you try to start it.

- `--myisam-block-size=N`

Property	Value
Command-Line Format	`--myisam-block-size=#`
Type	Integer
Default Value	`1024`
Minimum Value	`1024`
Maximum Value	`16384`

The block size to be used for `MyISAM` index pages.

- `--no-defaults`

Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read. This must be the first option on the command line if it is used.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-dd-upgrade`

Property	Value
Command-Line Format	`--no-dd-upgrade[={OFF|ON}]`
Introduced	8.0.4
Deprecated	8.0.16
Type	Boolean
Default Value	`OFF`

Note

This option is deprecated as of MySQL 8.0.16. It is superseded by the `--upgrade` option, which provides finer control over data dictionary and server upgrade behavior.

Prevent automatic upgrade of the data dictionary tables during the MySQL server startup process. This option is typically used when starting the MySQL server following an in-place upgrade of an existing installation to a newer MySQL version, which may include changes to data dictionary table definitions.

When `--no-dd-upgrade` is specified, and the server finds that its expected version of the data dictionary differs from the version stored in the data dictionary itself, startup fails with an error stating that data dictionary upgrade is prohibited;

```
[ERROR] [MY-011091] [Server] Data dictionary upgrade prohibited by the
command line option '--no_dd_upgrade'.
[ERROR] [MY-010020] [Server] Data Dictionary initialization failed.
```

During a normal startup, the data dictionary version of the server is compared to the version stored in the data dictionary to determine whether data dictionary table definitions should be upgraded. If an upgrade is necessary and supported, the server creates data dictionary tables with updated definitions, copies persisted metadata to the new tables, atomically replaces the old tables with the new ones, and reinitializes the data dictionary. If an upgrade is not necessary, startup continues without updating data dictionary tables.

- `--no-monitor`

Property	Value
Command-Line Format	`--no-monitor[={OFF|ON}]`
Introduced	8.0.12
Platform Specific	Windows
Type	Boolean
Default Value	`OFF`

(Windows only). This option suppresses the forking that is used to implement the `RESTART` statement: Forking enables one process to act as a monitor to the other, which acts as the server. For a server started with this option, `RESTART` simply exits and does not restart.

`--no-monitor` is not available prior to MySQL 8.0.12. The `--gdb` option can be used as a workaround.

- `--old-style-user-limits`

Property	Value
Command-Line Format	`--old-style-user-limits[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Enable old-style user limits. (Before MySQL 5.0.3, account resource limits were counted separately for each host from which a user connected rather than per account row in the `user` table.) See Section 6.2.20, “Setting Account Resource Limits”.

- `--performance-schema-xxx`

Configure a Performance Schema option. For details, see Section 26.14, “Performance Schema Command Options”.

- `--plugin-load=`*`plugin_list`*

Property	Value
Command-Line Format	`--plugin-load=plugin_list`
System Variable	`plugin_load`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

This option tells the server to load the named plugins at startup. If multiple `--plugin-load` options are given, only the last one is used. Additional plugins to load may be specified using `--plugin-load-add` options.

The option value is a semicolon-separated list of *`name=plugin_library`* and *`plugin_library`* values. Each *`name`* is the name of a plugin to load, and *`plugin_library`* is the name of the library file that contains the plugin code. If a plugin library is named without any preceding plugin name, the server loads all plugins in the library. The server looks for plugin library files in the directory named by the `plugin_dir` system variable.

For example, if plugins named `myplug1` and `myplug2` have library files `myplug1.so` and `myplug2.so`, use this option to perform an early plugin load:

```
shell> mysqld --plugin-load="myplug1=myplug1.so;myplug2=myplug2.so"
```

Quotes are used around the argument value here because otherwise semicolon (`;`) is interpreted as a special character by some command interpreters. (Unix shells treat it as a command terminator, for example.)

Each named plugin is loaded for a single invocation of `mysqld` only. After a restart, the plugin is not loaded unless `--plugin-load` is used again. This is in contrast to `INSTALL PLUGIN`, which adds an entry to the `mysql.plugins` table to cause the plugin to be loaded for every normal server startup.

Under normal startup, the server determines which plugins to load by reading the `mysql.plugins` system table. If the server is started with the `--skip-grant-tables` option, it does not consult the `mysql.plugins` table and does not load plugins listed there. `--plugin-load` enables plugins to be loaded even when `--skip-grant-tables` is given. `--plugin-load` also enables plugins to be loaded at startup that cannot be loaded at runtime.

For additional information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

- `--plugin-load-add=plugin_list`

Property	Value
Command-Line Format	`--plugin-load-add=plugin_list`
System Variable	`plugin_load_add`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

This option complements the `--plugin-load` option. `--plugin-load-add` adds a plugin or plugins to the set of plugins to be loaded at startup. The argument format is the same as for `--plugin-load`. `--plugin-load-add` can be used to avoid specifying a large set of plugins as a single long unwieldy `--plugin-load` argument.

`--plugin-load-add` can be given in the absence of `--plugin-load`, but any instance of `--plugin-load-add` that appears before `--plugin-load`. has no effect because `--plugin-load` resets the set of plugins to load. In other words, these options:

```
--plugin-load=x --plugin-load-add=y
```

are equivalent to this option:

```
--plugin-load="x;y"
```

But these options:

```
--plugin-load-add=y --plugin-load=x
```

are equivalent to this option:

```
--plugin-load=x
```

For additional information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

- `--plugin-xxx`

Specifies an option that pertains to a server plugin. For example, many storage engines can be built as plugins, and for such engines, options for them can be specified with a `--plugin` prefix. Thus, the `--innodb-file-per-table` option for `InnoDB` can be specified as `--plugin-innodb-file-per-table`.

For boolean options that can be enabled or disabled, the `--skip` prefix and other alternative formats are supported as well (see Section 4.2.2.4, “Program Option Modifiers”). For example, `--skip-plugin-innodb-file-per-table` disables `innodb-file-per-table`.

The rationale for the `--plugin` prefix is that it enables plugin options to be specified unambiguously if there is a name conflict with a built-in server option. For example, were a plugin writer to name a plugin “sql” and implement a “mode” option, the option name might be `--sql-mode`, which would conflict with the built-in option of the same name. In such cases, references to the conflicting name are resolved in favor of the built-in option. To avoid the ambiguity, users can specify the plugin option as `--plugin-sql-mode`. Use of the `--plugin` prefix for plugin options is recommended to avoid any question of ambiguity.

- `--port=`*`port_num`*, `-P` *`port_num`*

Property	Value
Command-Line Format	`--port=port_num`
System Variable	`port`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`3306`
Minimum Value	`0`
Maximum Value	`65535`

The port number to use when listening for TCP/IP connections. On Unix and Unix-like systems, the port number must be 1024 or higher unless the server is started by the `root` operating system user. Setting this option to 0 causes the default value to be used.

- `--port-open-timeout=`*`num`*

Property	Value
Command-Line Format	`--port-open-timeout=#`
Type	Integer
Default Value	`0`

On some systems, when the server is stopped, the TCP/IP port might not become available immediately. If the server is restarted quickly afterward, its attempt to reopen the port can fail. This option indicates how many seconds the server should wait for the TCP/IP port to become free if it cannot be opened. The default is not to wait.

- `--print-defaults`

Print the program name and all options that it gets from option files. Password values are masked. This must be the first option on the command line if it is used, except that it may be used immediately after `--defaults-file` or `--defaults-extra-file`.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--remove [service_name]`

Property	Value
Command-Line Format	`--remove [service_name]`
Platform Specific	Windows

(Windows only) Remove a MySQL Windows service. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.3.4.8, “Starting MySQL as a Windows Service”.

- `--safe-user-create`

Property	Value
Command-Line Format	`--safe-user-create[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement unless the user has the `INSERT` privilege for the `mysql.user` system table or any column in the table. If you want a user to have the ability to create new users that have those privileges that the user has the right to grant, you should grant the user the following privilege:

```
GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

This ensures that the user cannot change any privilege columns directly, but has to use the `GRANT` statement to give privileges to other users.

- `--secure-auth`

Property	Value
Command-Line Format	`--secure-auth[={OFF|ON}]`
Deprecated	Yes (removed in 8.0.3)
System Variable	`secure_auth`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`
Valid Values	`ON`

This option was removed in MySQL 8.0.3.

- `--skip-concurrent-insert`

Turn off the ability to select and insert at the same time on `MyISAM` tables. (This is to be used only if you think you have found a bug in this feature.) See Section 8.11.3, “Concurrent Inserts”.

- `--skip-grant-tables`

Property	Value
Command-Line Format	`--skip-grant-tables[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

This option causes the server to start without using the privilege system at all, which gives anyone with access to the server *unrestricted access to all databases*. You can cause a running server to start using the grant tables again by executing `mysqladmin flush-privileges` or `mysqladmin reload` command from a system shell, or by issuing a MySQL `FLUSH PRIVILEGES` statement after connecting to the server.

If the server is started with the `--skip-grant-tables` option to disable authentication checks, the server enables `--skip-networking` automatically to prevent remote connections.

This option also causes the server to suppress during its startup sequence the loading of user-defined functions (UDFs), scheduled events, and plugins that were installed with the `INSTALL PLUGIN` statement. To cause plugins to be loaded anyway, use the `--plugin-load` option. `--skip-grant-tables` also causes the `disabled_storage_engines` system variable to have no effect.

This option does not cause loading of server components to be suppressed during server startup.

`FLUSH PRIVILEGES` might be executed implicitly by other actions performed after startup (thus causing the server to start using the grant tables again). For example, `mysql_upgrade` flushes the privileges during the upgrade procedure.

- `--skip-host-cache`

Property	Value
Command-Line Format	`--skip-host-cache`

Disable use of the internal host cache for faster name-to-IP resolution. With the cache disabled, the server performs a DNS lookup every time a client connects.

Use of `--skip-host-cache` is similar to setting the `host_cache_size` system variable to 0, but `host_cache_size` is more flexible because it can also be used to resize, enable, or disable the host cache at runtime, not just at server startup.

If you start the server with `--skip-host-cache`, that does not prevent changes to the value of `host_cache_size`, but such changes have no effect and the cache is not re-enabled even if `host_cache_size` is set larger than 0.

For more information about how the host cache works, see Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

- `--skip-innodb`

Disable the `InnoDB` storage engine. In this case, because the default storage engine is `InnoDB`, the server will not start unless you also use `--default-storage-engine` and `--default-tmp-storage-engine` to set the default to some other engine for both permanent and `TEMPORARY` tables.

The `InnoDB` storage engine cannot be disabled, and the `--skip-innodb` option is deprecated and has no effect. Its use results in a warning. This option will be removed in a future MySQL release.

- `--skip-name-resolve`

Property	Value
Command-Line Format	`--skip-name-resolve[={OFF|ON}]`
System Variable	`skip_name_resolve`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Do not resolve host names when checking client connections. Use only IP addresses. If you use this option, all `Host` column values in the grant tables must be IP addresses. See Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

Depending on the network configuration of your system and the `Host` values for your accounts, clients may need to connect using an explicit `--host` option, such as `--host=127.0.0.1` or `--host=::1`.

An attempt to connect to the host `127.0.0.1` normally resolves to the `localhost` account. However, this fails if the server is run with the `--skip-name-resolve` option. If you plan to do that, make sure that an account exists that can accept a connection. For example, to be able to connect as `root` using `--host=127.0.0.1` or `--host=::1`, create these accounts:

```
CREATE USER 'root'@'127.0.0.1' IDENTIFIED BY 'root-password';
CREATE USER 'root'@'::1' IDENTIFIED BY 'root-password';
```

- `--skip-networking`

Property	Value
Command-Line Format	`--skip-networking`
System Variable	`skip_networking`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

Do not listen for TCP/IP connections at all. All interaction with `mysqld` must be made using named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are permitted. See Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

If the server is started with the `--skip-grant-tables` option to disable authentication checks, the server enables `--skip-networking` to prevent remote connections.

- `--ssl*`

Options that begin with `--ssl` specify whether to permit clients to connect using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--standalone`

Property	Value
Command-Line Format	`--standalone`
Platform Specific	Windows

Available on Windows only; instructs the MySQL server not to run as a service.

- `--super-large-pages`

Property	Value
Command-Line Format	`--super-large-pages[={OFF|ON}]`
Platform Specific	Solaris
Type	Boolean
Default Value	`OFF`

Standard use of large pages in MySQL attempts to use the largest size supported, up to 4MB. Under Solaris, a “super large pages” feature enables uses of pages up to 256MB. This feature is available for recent SPARC platforms. It can be enabled or disabled by using the `--super-large-pages` or `--skip-super-large-pages` option.

- `--symbolic-links`, `--skip-symbolic-links`

Property	Value
Command-Line Format	`--symbolic-links[={OFF|ON}]`
Deprecated	8.0.2
Type	Boolean
Default Value (>= 8.0.2)	`OFF`
Default Value (<= 8.0.1)	`ON`

Enable or disable symbolic link support. On Unix, enabling symbolic links means that you can link a `MyISAM` index file or data file to another directory with the `INDEX DIRECTORY` or `DATA DIRECTORY` option of the `CREATE TABLE` statement. If you delete or rename the table, the files that its symbolic links point to also are deleted or renamed. See Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”.

Note

Symbolic link support, along with the `--symbolic-links` option that controls it, is deprecated and will be removed in a future version of MySQL. In addition, the option is disabled by default. The related `have_symlink` system variable also is deprecated and will be removed in a future version of MySQL.

This option has no meaning on Windows.

- `--skip-show-database`

Property	Value
Command-Line Format	`--skip-show-database`
System Variable	`skip_show_database`

Property	Value
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

This option sets the `skip_show_database` system variable that controls who is permitted to use the `SHOW DATABASES` statement. See Section 5.1.8, “Server System Variables”.

- `--skip-stack-trace`

Property	Value
Command-Line Format	`--skip-stack-trace`

Do not write stack traces. This option is useful when you are running `mysqld` under a debugger. On some systems, you also must use this option to get a core file. See Section 29.5, “Debugging and Porting MySQL”.

- `--slow-start-timeout=`*`timeout`*

Property	Value
Command-Line Format	`--slow-start-timeout=#`
Type	Integer
Default Value	`15000`

This option controls the Windows service control manager's service start timeout. The value is the maximum number of milliseconds that the service control manager waits before trying to kill the windows service during startup. The default value is 15000 (15 seconds). If the MySQL service takes too long to start, you may need to increase this value. A value of 0 means there is no timeout.

- `--socket=`*`path`*

Property	Value
Command-Line Format	`--socket={file_name|pipe_name}`
System Variable	`socket`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value (Other)	`/tmp/mysql.sock`
Default Value (Windows)	`MySQL`

On Unix, this option specifies the Unix socket file to use when listening for local connections. The default value is `/tmp/mysql.sock`. If this option is given, the server creates the file in the data directory unless an absolute path name is given to specify a different directory. On Windows, the option specifies the pipe name to use when listening for local connections that use a named pipe. The default value is `MySQL` (not case sensitive).

- `--sql-mode=`*`value`*`[,`*`value`*`[,`*`value`*`...]]`

Property	Value
Command-Line Format	`--sql-mode=name`
System Variable	`sql_mode`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Set
Default Value (>= 8.0.11)	`ONLY_FULL_GROUP_BY STRICT_TRANS_TABLES NO_ZERO_IN_DATE NO_ZERO_DATE ERROR_FOR_DIVISION_BY_ZERO NO_ENGINE_SUBSTITUTION`
Default Value (<= 8.0.4)	`ONLY_FULL_GROUP_BY STRICT_TRANS_TABLES NO_ZERO_IN_DATE NO_ZERO_DATE ERROR_FOR_DIVISION_BY_ZERO NO_AUTO_CREATE_USER NO_ENGINE_SUBSTITUTION`

Valid Values (>= 8.0.11)	`ALLOW_INVALID_DATES`
	`ANSI_QUOTES`
	`ERROR_FOR_DIVISION_BY_ZERO`
	`HIGH_NOT_PRECEDENCE`
	`IGNORE_SPACE`
	`NO_AUTO_VALUE_ON_ZERO`
	`NO_BACKSLASH_ESCAPES`
	`NO_DIR_IN_CREATE`
	`NO_ENGINE_SUBSTITUTION`
	`NO_UNSIGNED_SUBTRACTION`
	`NO_ZERO_DATE`
	`NO_ZERO_IN_DATE`
	`ONLY_FULL_GROUP_BY`
	`PAD_CHAR_TO_FULL_LENGTH`
	`PIPES_AS_CONCAT`
	`REAL_AS_FLOAT`
	`STRICT_ALL_TABLES`
	`STRICT_TRANS_TABLES`
	`TIME_TRUNCATE_FRACTIONAL`

Property	Value
Valid Values (>= 8.0.1, <= 8.0.4)	ALLOW_INVALID_DATES ANSI_QUOTES ERROR_FOR_DIVISION_BY_ZERO HIGH_NOT_PRECEDENCE IGNORE_SPACE NO_AUTO_CREATE_USER NO_AUTO_VALUE_ON_ZERO NO_BACKSLASH_ESCAPES NO_DIR_IN_CREATE NO_ENGINE_SUBSTITUTION NO_FIELD_OPTIONS NO_KEY_OPTIONS NO_TABLE_OPTIONS NO_UNSIGNED_SUBTRACTION NO_ZERO_DATE NO_ZERO_IN_DATE ONLY_FULL_GROUP_BY PAD_CHAR_TO_FULL_LENGTH PIPES_AS_CONCAT REAL_AS_FLOAT STRICT_ALL_TABLES STRICT_TRANS_TABLES TIME_TRUNCATE_FRACTIONAL
Valid Values (8.0.0)	ALLOW_INVALID_DATES ANSI_QUOTES ERROR_FOR_DIVISION_BY_ZERO HIGH_NOT_PRECEDENCE IGNORE_SPACE NO_AUTO_CREATE_USER

Property	Value
	NO_AUTO_VALUE_ON_ZERO NO_BACKSLASH_ESCAPES NO_DIR_IN_CREATE NO_ENGINE_SUBSTITUTION NO_FIELD_OPTIONS NO_KEY_OPTIONS NO_TABLE_OPTIONS NO_UNSIGNED_SUBTRACTION NO_ZERO_DATE NO_ZERO_IN_DATE ONLY_FULL_GROUP_BY PAD_CHAR_TO_FULL_LENGTH PIPES_AS_CONCAT REAL_AS_FLOAT STRICT_ALL_TABLES STRICT_TRANS_TABLES

Set the SQL mode. See Section 5.1.11, “Server SQL Modes”.

Note

MySQL installation programs may configure the SQL mode during the installation process.

If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

- `--sysdate-is-now`

Property	Value
Command-Line Format	`--sysdate-is-now[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

`SYSDATE()` by default returns the time at which it executes, not the time at which the statement in which it occurs begins executing. This differs from the behavior of `NOW()`. This option causes `SYSDATE()` to be an alias for `NOW()`. For information about the implications for binary logging and replication, see

the description for `SYSDATE()` in Section 12.7, “Date and Time Functions” and for `SET TIMESTAMP` in Section 5.1.8, “Server System Variables”.

- `--tc-heuristic-recover={COMMIT|ROLLBACK}`

Property	Value
Command-Line Format	`--tc-heuristic-recover=name`
Type	Enumeration
Default Value	`COMMIT`
Valid Values	`COMMIT` `ROLLBACK`

The type of decision to use in the heuristic recovery process. To use this option, two or more storage engines that support XA transactions must be installed.

- `--temp-pool`

Property	Value
Command-Line Format	`--temp-pool[={OFF|ON}]`
Deprecated	Yes (removed in 8.0.1)
Type	Boolean
Default Value (Other)	`OFF`
Default Value (Linux)	`ON`

This option is obsolete and was removed in MySQL 8.0.1.

- `--transaction-isolation=level`

Property	Value
Command-Line Format	`--transaction-isolation=name`
System Variable	`transaction_isolation`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`REPEATABLE-READ`

Valid Values	`READ-UNCOMMITTED`
	`READ-COMMITTED`
	`REPEATABLE-READ`
	`SERIALIZABLE`

Sets the default transaction isolation level. The `level` value can be `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, or `SERIALIZABLE`. See Section 13.3.7, “SET TRANSACTION Syntax”.

The default transaction isolation level can also be set at runtime using the `SET TRANSACTION` statement or by setting the `transaction_isolation` system variable.

- `--transaction-read-only`

Property	Value
Command-Line Format	`--transaction-read-only[={OFF|ON}]`
System Variable	`transaction_read_only`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Sets the default transaction access mode. By default, read-only mode is disabled, so the mode is read/write.

To set the default transaction access mode at runtime, use the `SET TRANSACTION` statement or set the `transaction_read_only` system variable. See Section 13.3.7, “SET TRANSACTION Syntax”.

- `--tmpdir=dir_name`, `-t dir_name`

Property	Value
Command-Line Format	`--tmpdir=dir_name`
System Variable	`tmpdir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

The path of the directory to use for creating temporary files. It might be useful if your default `/tmp` directory resides on a partition that is too small to hold temporary tables. This option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (`:`) on Unix and semicolon characters (`;`) on Windows.

`--tmpdir` can be a non-permanent location, such as a directory on a memory-based file system or a directory that is cleared when the server host restarts. If the MySQL server is acting as a replication slave, and you are using a non-permanent location for `--tmpdir`, consider setting a different temporary directory for the slave using the `--slave-load-tmpdir` option. For a replication slave, the temporary files used to replicate `LOAD DATA` statements are stored in this directory, so with a permanent location they can survive machine restarts, although replication can now continue after a restart if the temporary files have been removed.

For more information about the storage location of temporary files, see Section B.4.3.5, “Where MySQL Stores Temporary Files”.

- `--upgrade=value`

Property	Value
Command-Line Format	`--upgrade=value`
Introduced	8.0.16
Type	Enumeration
Default Value	`AUTO`

Valid Values	`AUTO`
	`NONE`
	`MINIMAL`
	`FORCE`

This option controls whether and how the server performs an automatic upgrade at startup. Automatic upgrade involves two steps:

- Step 1: Data dictionary upgrade.

  This step upgrades:

  - The data dictionary tables in the `mysql` schema. If the actual data dictionary version is lower than the current expected version, the server upgrades the data dictionary. If it cannot, or is prevented from doing so, the server will not run.
  - The Performance Schema and `INFORMATION_SCHEMA`.

- Step 2: Server upgrade.

  This step comprises all other upgrade tasks. If the existing installation data has a lower MySQL version than the server expects, it must be upgraded:

  - The system tables in the `mysql` schema (the remaining non-data dictionary tables).
  - The `sys` schema.
  - User schemas.

For details about upgrade steps 1 and 2, see Section 2.11.3, “What the MySQL Upgrade Process Upgrades”.

These `--upgrade` option values are permitted:

- `AUTO`

  The server performs an automatic upgrade of anything it finds to be out of date (steps 1 and 2). This is the default action if `--upgrade` is not specified explicitly.

- `NONE`

  The server performs no automatic upgrade steps during the startup process (skips steps 1 and 2). Because this option value prevents a data dictionary upgrade, the server exits with an error if the data dictionary is found to be out of date:

```
[ERROR] [MY-013381] [Server] Server shutting down because upgrade is
required, yet prohibited by the command line option '--upgrade=NONE'.
[ERROR] [MY-010334] [Server] Failed to initialize DD Storage Engine
[ERROR] [MY-010020] [Server] Data Dictionary initialization failed.
```

- `MINIMAL`

  The server upgrades the data dictionary, the Performance Schema, and the `INFORMATION_SCHEMA`, if necessary (step 1). Note that following an upgrade with this option, Group Replication cannot be started, because system tables on which the replication internals depend are not updated, and reduced functionality might also be apparent in other areas.

- `FORCE`

  The server upgrades the data dictionary, the Performance Schema, and the `INFORMATION_SCHEMA`, if necessary (step 1). In addition, the server forces an upgrade of everything else (step 2). Expect server startup to take longer with this option because the server checks all objects in all schemas.

  `FORCE` is useful to force step 2 actions to be performed if the server thinks they are not necessary. For example, you may believe that a system table is missing or has become damaged and want to force a repair.

The following table summarizes the actions taken by the server for each option value.

Option Value	Server Performs Step 1?	Server Performs Step 2?
`AUTO`	If necessary	If necessary
`NONE`	No	No
`MINIMAL`	If necessary	No
`FORCE`	If necessary	Yes

- `--user={`*`user_name`*`|`*`user_id`*`}`, `-u {`*`user_name`*`|`*`user_id`*`}`

Property	Value
Command-Line Format	`--user=name`
Type	String

Run the `mysqld` server as the user having the name *`user_name`* or the numeric user ID *`user_id`*. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

This option is *mandatory* when starting `mysqld` as `root`. The server changes its user ID during its startup sequence, causing it to run as that particular user rather than as `root`. See Section 6.1.1, “Security Guidelines”.

To avoid a possible security hole where a user adds a `--user=root` option to a `my.cnf` file (thus causing the server to run as `root`), `mysqld` uses only the first `--user` option specified and produces a warning if there are multiple `--user` options. Options in `/etc/my.cnf` and `$MYSQL_HOME/my.cnf` are processed before command-line options, so it is recommended that you put a `--user` option in `/etc/my.cnf` and specify a value other than `root`. The option in `/etc/my.cnf` is found before any other `--user` options, which ensures that the server runs as a user other than `root`, and that a warning results if any other `--user` option is found.

- `--validate-config`

Property	Value
Command-Line Format	`--validate-config[={OFF|ON}]`
Introduced	8.0.16
Type	Boolean
Default Value	`OFF`

Validate the server startup configuration. If no errors are found, the server terminates with an exit code of 0. If an error is found, the server displays a diagnostic message and terminates with an exit code of 1. Warning and information messages may also be displayed, depending on the `log_error_verbosity` value, but do not produce immediate validation termination or an exit code of 1. For more information, see Section 5.1.3, “Server Configuration Validation”.

- `--verbose`, `-v`

  Use this option with the `--help` option for detailed help.

- `--version`, `-V`

  Display version information and exit.

5.1.8 Server System Variables

The MySQL server maintains many system variables that configure its operation. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically at runtime using the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can also use system variable values in expressions.

Setting a global system variable runtime value normally requires the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege. Setting a session system runtime variable value normally requires no special privileges and can be done by any user, although there are exceptions. For more information, see Section 5.1.9.1, “System Variable Privileges”

There are several ways to see the names and values of system variables:

- To see the values that a server will use based on its compiled-in defaults and any option files that it reads, use this command:

```
mysqld --verbose --help
```

- To see the values that a server will use based only on its compiled-in defaults, ignoring the settings in any option files, use this command:

```
mysqld --no-defaults --verbose --help
```

- To see the current values used by a running server, use the `SHOW VARIABLES` statement or the Performance Schema system variable tables. See Section 26.12.13, “Performance Schema System Variable Tables”.

This section provides a description of each system variable. For a system variable summary table, see Section 5.1.5, “Server System Variable Reference”. For more information about manipulation of system variables, see Section 5.1.9, “Using System Variables”.

For additional system variable information, see these sections:

- Section 5.1.9, “Using System Variables”, discusses the syntax for setting and displaying system variable values.
- Section 5.1.9.2, “Dynamic System Variables”, lists the variables that can be set at runtime.
- Information on tuning system variables can be found in Section 5.1.1, “Configuring the Server”.
- Section 15.13, “InnoDB Startup Options and System Variables”, lists `InnoDB` system variables.
- NDB Cluster System Variables, lists system variables which are specific to NDB Cluster.
- For information on server system variables specific to replication, see Section 17.1.6, “Replication and Binary Logging Options and Variables”.

Note

Some of the following variable descriptions refer to “enabling” or “disabling” a variable. These variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. Boolean variables can be set at startup to the values `ON`, `TRUE`, `OFF`, and `FALSE` (not case sensitive), as well as `1` and `0`. See Section 4.2.2.4, “Program Option Modifiers”.

Some system variables control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to a system variable that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server will adjust a value upward. For example, if you assign a value of 0 to a variable for which the minimal value is 1024, the server will set the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some system variables take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued variable is given as a relative path name, it will be located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

- `activate_all_roles_on_login`

Property	Value
Command-Line Format	`--activate-all-roles-on-login[={OFF|ON}]`
Introduced	8.0.2
System Variable	`activate_all_roles_on_login`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether to enable automatic activation of all granted roles when users log in to the server:

- If `activate_all_roles_on_login` is enabled, the server activates all roles granted to each account at login time. This takes precedence over default roles specified with `SET DEFAULT ROLE`.

- If `activate_all_roles_on_login` is disabled, the server activates the default roles specified with `SET DEFAULT ROLE`, if any, at login time.

Granted roles include those granted explicitly to the user and those named in the `mandatory_roles` system variable value.

`activate_all_roles_on_login` applies only at login time, and at the beginning of execution for stored programs and views that execute in definer context. To change the active roles within a session, use `SET ROLE`. To change the active roles for a stored program, the program body should execute `SET ROLE`.

- `admin_address`

Property	Value
Command-Line Format	`--admin-address=addr`
Introduced	8.0.14
System Variable	`admin_address`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The IP address on which to listen for TCP/IP connections on the administrative network interface (see Section 8.12.4.1, “How MySQL Handles Client Connections”). There is no default `admin_address` value. If this variable is not specified at startup, the server maintains no administrative interface. The server also has a `bind_address` system variable for configuring regular (nonadministrative) client TCP/IP connections. See Section 8.12.4.1, “How MySQL Handles Client Connections”.

If `admin_address` is specified, its value must satisfy these requirements:

- The value must be a single IPv4 address, IPv6 address, or host name.
- The value cannot specify a wildcard address format (`*`, `0.0.0.0`, or `::`).

An IP address can be specified as an IPv4 or IPv6 address. If the value is a host name, the server resolves the name to an IP address and binds to that address. If a host name resolves to multiple IP addresses, the server uses the first IPv4 address if there are any, or the first IPv6 address otherwise.

The server treats different types of addresses as follows:

- If the address is an IPv4-mapped address, the server accepts TCP/IP connections for that address, in either IPv4 or IPv6 format. For example, if the server is bound to `::ffff:127.0.0.1`, clients can connect using `--host=127.0.0.1` or `--host=::ffff:127.0.0.1`.
- If the address is a “regular” IPv4 or IPv6 address (such as `127.0.0.1` or `::1`), the server accepts TCP/IP connections only for that IPv4 or IPv6 address.

If binding to the address fails, the server produces an error and does not start.

The `admin_address` system variable is similar to the `bind_address` system variable that binds the server to an address for ordinary client connections, but with these differences:

  - `bind_address` permits multiple addresses. `admin_address` permits a single address.
  - `bind_address` permits wildcard addresses. `admin_address` does not.

- `admin_port`

Property	Value
Command-Line Format	`--admin-port=port_num`
Introduced	8.0.14
System Variable	`admin_port`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`33062`
Minimum Value	`0`
Maximum Value	`65535`

The TCP/IP port number to use for connections on the administrative network interface (see Section 8.12.4.1, “How MySQL Handles Client Connections”). Setting this variable to 0 causes the default value to be used.

Setting `admin_port` has no effect if `admin_address` is not specified because in that case the server maintains no administrative network interface.

- `authentication_windows_log_level`

Property	Value
Command-Line Format	`--authentication-windows-log-level=#`
Introduced	8.0.11
System Variable	`authentication_windows_log_level`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`2`
Minimum Value	`0`
Maximum Value	`4`

This variable is available only if the `authentication_windows` Windows authentication plugin is enabled and debugging code is enabled. See Section 6.4.1.6, “Windows Pluggable Authentication”.

This variable sets the logging level for the Windows authentication plugin. The following table shows the permitted values.

Value	Description
0	No logging
1	Log only error messages
2	Log level 1 messages and warning messages
3	Log level 2 messages and information notes
4	Log level 3 messages and debug messages

- `authentication_windows_use_principal_name`

Property	Value
Command-Line Format	`--authentication-windows-use-principal-name[={OFF|ON}]`
Introduced	8.0.11
System Variable	`authentication_windows_use_principal_name`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

This variable is available only if the `authentication_windows` Windows authentication plugin is enabled. See Section 6.4.1.6, “Windows Pluggable Authentication”.

A client that authenticates using the `InitSecurityContext()` function should provide a string identifying the service to which it connects (*`targetName`*). MySQL uses the principal name (UPN) of the account under which the server is running. The UPN has the form *`user_id@computer_name`* and need not be registered anywhere to be used. This UPN is sent by the server at the beginning of authentication handshake.

This variable controls whether the server sends the UPN in the initial challenge. By default, the variable is enabled. For security reasons, it can be disabled to avoid sending the server's account name to a client as cleartext. If the variable is disabled, the server always sends a `0x00` byte in the first challenge, the client does not specify *`targetName`*, and as a result, NTLM authentication is used.

If the server fails to obtain its UPN (which will happen primarily in environments that do not support Kerberos authentication), the UPN is not sent by the server and NTLM authentication is used.

- `autocommit`

Property	Value
Command-Line Format	`--autocommit[={OFF|ON}]`
System Variable	`autocommit`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean

Property	Value
Default Value	`ON`

The autocommit mode. If set to 1, all changes to a table take effect immediately. If set to 0, you must use `COMMIT` to accept a transaction or `ROLLBACK` to cancel it. If `autocommit` is 0 and you change it to 1, MySQL performs an automatic `COMMIT` of any open transaction. Another way to begin a transaction is to use a `START TRANSACTION` or `BEGIN` statement. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

By default, client connections begin with `autocommit` set to 1. To cause clients to begin with a default of 0, set the global `autocommit` value by starting the server with the `--autocommit=0` option. To set the variable using an option file, include these lines:

```
[mysqld]
autocommit=0
```

- `automatic_sp_privileges`

Property	Value
Command-Line Format	`--automatic-sp-privileges[={OFF|ON}]`
System Variable	`automatic_sp_privileges`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

When this variable has a value of 1 (the default), the server automatically grants the `EXECUTE` and `ALTER ROUTINE` privileges to the creator of a stored routine, if the user cannot already execute and alter or drop the routine. (The `ALTER ROUTINE` privilege is required to drop the routine.) The server also automatically drops those privileges from the creator when the routine is dropped. If `automatic_sp_privileges` is 0, the server does not automatically add or drop these privileges.

The creator of a routine is the account used to execute the `CREATE` statement for it. This might not be the same as the account named as the `DEFINER` in the routine definition.

See also Section 24.2.2, “Stored Routines and MySQL Privileges”.

- `auto_generate_certs`

Property	Value
Command-Line Format	`--auto-generate-certs[={OFF|ON}]`
System Variable	`auto_generate_certs`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

This variable controls whether the server autogenerates SSL key and certificate files in the data directory, if they do not already exist.

At startup, the server automatically generates server-side and client-side SSL certificate and key files in the data directory if the `auto_generate_certs` system variable is enabled, no SSL options other than `--ssl` are specified, and the server-side SSL files are missing from the data directory. These files enable secure client connections using SSL; see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

For more information about SSL file autogeneration, including file names and characteristics, see Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”

The `sha256_password_auto_generate_rsa_keys` and `caching_sha2_password_auto_generate_rsa_keys` system variables are related but control autogeneration of RSA key-pair files needed for secure password exchange using RSA over unencypted connections.

- `avoid_temporal_upgrade`

Property	Value
Command-Line Format	`--avoid-temporal-upgrade[={OFF|ON}]`
Deprecated	Yes
System Variable	`avoid_temporal_upgrade`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This variable controls whether `ALTER TABLE` implicitly upgrades temporal columns found to be in pre-5.6.4 format (`TIME`, `DATETIME`, and `TIMESTAMP` columns without support for fractional seconds precision). Upgrading such columns requires a table rebuild, which prevents any use of fast alterations that might otherwise apply to the operation to be performed.

This variable is disabled by default. Enabling it causes `ALTER TABLE` not to rebuild temporal columns and thereby be able to take advantage of possible fast alterations.

This variable is deprecated and will be removed in a future MySQL release.

- `back_log`

Property	Value
Command-Line Format	`--back-log=#`
System Variable	`back_log`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`1`
Maximum Value	`65535`

The number of outstanding connection requests MySQL can have. This comes into play when the main MySQL thread gets very many connection requests in a very short time. It then takes some time (although very little) for the main thread to check the connection and start a new thread. The `back_log` value indicates how many requests can be stacked during this short time before MySQL momentarily stops answering new requests. You need to increase this only if you expect a large number of connections in a short period of time.

In other words, this value is the size of the listen queue for incoming TCP/IP connections. Your operating system has its own limit on the size of this queue. The manual page for the Unix `listen()` system call should have more details. Check your OS documentation for the maximum value for this variable. `back_log` cannot be set higher than your operating system limit.

The default value is the value of `max_connections`, which enables the permitted backlog to adjust to the maximum permitted number of connections.

- `basedir`

Property	Value
Command-Line Format	`--basedir=dir_name`
System Variable	`basedir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name
Default Value (>= 8.0.2)	`parent of mysqld installation directory`
Default Value (<= 8.0.1)	`configuration-dependent default`

The path to the MySQL installation base directory.

- `big_tables`

Property	Value
Command-Line Format	`--big-tables[={OFF|ON}]`
System Variable	`big_tables`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

If enabled, the server stores all temporary tables on disk rather than in memory. This prevents most `The table tbl_name is full` errors for `SELECT` operations that require a large temporary table, but also slows down queries for which in-memory tables would suffice.

The default value for new connection is `OFF` (use in-memory temporary tables). Normally, it should never be necessary to enable this variable. When in-memory *internal* temporary tables are managed by the `TempTable` storage engine (the default), and the maximum amount of memory that can be occupied by the `TempTable` storage engine is exceeded, the `TempTable` storage engine starts storing data to temporary files on disk. When in-memory temporary tables are managed by the `MEMORY` storage engine, in-memory tables are automatically converted to disk-based tables as required. For more information, see Section 8.4.4, “Internal Temporary Table Use in MySQL”.

- `bind_address`

Property	Value
Command-Line Format	`--bind-address=addr`
System Variable	`bind_address`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`*`

The MySQL server listens on one or more network sockets for TCP/IP connections. Each socket is bound to one address, but it is possible for an address to map onto multiple network interfaces. To specify how the server should listen for TCP/IP connections, set the `bind_address` system variable at server startup. The server also has an `admin_address` system variable that enables administrative connections on a dedicated interface. See Section 8.12.4.1, “How MySQL Handles Client Connections”.

If `bind_address` is specified, its value must satisfy these requirements:

- Prior to MySQL 8.0.13, `bind_address` accepts a single address value, which may specify a single non-wildcard IP address or host name, or one of the wildcard address formats that permit listening on multiple network interfaces (`*`, `0.0.0.0`, or `::`).
- As of MySQL 8.0.13, `bind_address` accepts a single value as just described, or a list of comma-separated values. When the variable names a list of multiple values, each value must specify a single non-wildcard IP address or host name; none can specify a wildcard address format (`*`, `0.0.0.0`, or `::`).

IP addresses can be specified as IPv4 or IPv6 addresses. For any value that is a host name, the server resolves the name to an IP address and binds to that address. If a host name resolves to multiple IP addresses, the server uses the first IPv4 address if there are any, or the first IPv6 address otherwise.

The server treats different types of addresses as follows:

- If the address is `*`, the server accepts TCP/IP connections on all server host IPv4 interfaces, and, if the server host supports IPv6, on all IPv6 interfaces. Use this address to permit both IPv4 and IPv6 connections on all server interfaces. This value is the default. If the option specifies a list of multiple values, this value is not permitted.

- If the address is `0.0.0.0`, the server accepts TCP/IP connections on all server host IPv4 interfaces. If the option specifies a list of multiple values, this value is not permitted.
- If the address is `::`, the server accepts TCP/IP connections on all server host IPv4 and IPv6 interfaces. If the option specifies a list of multiple values, this value is not permitted.
- If the address is an IPv4-mapped address, the server accepts TCP/IP connections for that address, in either IPv4 or IPv6 format. For example, if the server is bound to `::ffff:127.0.0.1`, clients can connect using `--host=127.0.0.1` or `--host=::ffff:127.0.0.1`.
- If the address is a “regular” IPv4 or IPv6 address (such as `127.0.0.1` or `::1`), the server accepts TCP/IP connections only for that IPv4 or IPv6 address.

If binding to any address fails, the server produces an error and does not start.

Examples:

- `bind_address=*`

  The server listens on all IPv4 or IPv6 addresses, as specified by the `*` wildcard.

- `bind_address=198.51.100.20`

  The server listens only on the `198.51.100.20` IPv4 address.

- `bind_address=198.51.100.20,2001:db8:0:f101::1`

  The server listens on the `198.51.100.20` IPv4 address and the `2001:db8:0:f101::1` IPv6 address.

- `bind_address=198.51.100.20,*`

  This produces an error because wildcard addresses are not permitted when `bind_address` names a list of multiple values.

When `bind_address` names a single value (wildcard or non-wildcard), the server listens on a single socket, which for a wildcard address may be bound to multiple network interfaces. When `bind_address` names a list of multiple values, the server listens on one socket per value, with each socket bound to a single network interface. The number of sockets is linear with the number of values specified. Depending on operating system connection-acceptance efficiency, long value lists might incur a performance penalty for accepting TCP/IP connections.

Because file descriptors are allocated for listening sockets, it may be necessary to increase the `open_files_limit` system variable.

If you intend to bind the server to a specific address, be sure that the `mysql.user` system table contains an account with administrative privileges that you can use to connect to that address. Otherwise, you will not be able to shut down the server. For example, if you bind the server to `*`, you can connect to it using all existing accounts. But if you bind the server to `::1`, it accepts connections only on that address. In that case, first make sure that the `'root'@'::1'` account is present in the `mysql.user` table so you can still connect to the server to shut it down.

- `block_encryption_mode`

Property	Value
Command-Line Format	`--block-encryption-mode=#`
System Variable	`block_encryption_mode`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`aes-128-ecb`

This variable controls the block encryption mode for block-based algorithms such as AES. It affects encryption for `AES_ENCRYPT()` and `AES_DECRYPT()`.

`block_encryption_mode` takes a value in `aes-keylen-mode` format, where *`keylen`* is the key length in bits and *`mode`* is the encryption mode. The value is not case-sensitive. Permitted *`keylen`* values are 128, 192, and 256. Permitted *`mode`* values are `ECB`, `CBC`, `CFB1`, `CFB8`, `CFB128`, and `OFB`.

For example, this statement causes the AES encryption functions to use a key length of 256 bits and the CBC mode:

```
SET block_encryption_mode = 'aes-256-cbc';
```

An error occurs for attempts to set `block_encryption_mode` to a value containing an unsupported key length or a mode that the SSL library does not support.

- `bulk_insert_buffer_size`

Property	Value
Command-Line Format	`--bulk-insert-buffer-size=#`
System Variable	`bulk_insert_buffer_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`8388608`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

`MyISAM` uses a special tree-like cache to make bulk inserts faster for `INSERT ... SELECT`, `INSERT ... VALUES (...), (...), ...`, and `LOAD DATA` when adding data to nonempty tables. This variable limits the size of the cache tree in bytes per thread. Setting it to 0 disables this optimization. The default value is 8MB.

As of MySQL 8.0.14, setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

- `caching_sha2_password_auto_generate_rsa_keys`

Property	Value
Command-Line Format	`--caching-sha2-password-auto-generate-rsa-keys[={OFF|ON}]`
Introduced	8.0.4
System Variable	`caching_sha2_password_auto_generate_rsa_keys`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

The server uses this variable to determine whether to autogenerate RSA private/public key-pair files in the data directory if they do not already exist.

At startup, the server automatically generates RSA private/public key-pair files in the data directory if all of these conditions are true: The `sha256_password_auto_generate_rsa_keys` or `caching_sha2_password_auto_generate_rsa_keys` system variable is enabled; no RSA options are specified; the RSA files are missing from the data directory. These key-pair files enable secure password exchange using RSA over unencrypted connections for accounts authenticated by the `sha256_password` or `caching_sha2_password` plugin; see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

For more information about RSA file autogeneration, including file names and characteristics, see Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”

The `auto_generate_certs` system variable is related but controls autogeneration of SSL certificate and key files needed for secure connections using SSL.

- `caching_sha2_password_private_key_path`

Property	Value
Command-Line Format	`--caching-sha2-password-private-key-path=file_name`
Introduced	8.0.3
System Variable	`caching_sha2_password_private_key_path`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	`private_key.pem`

This variable specifies the path name of the RSA private key file for the `caching_sha2_password` authentication plugin. If the file is named as a relative path, it is interpreted relative to the server data directory. The file must be in PEM format.

Important

Because this file stores a private key, its access mode should be restricted so that only the MySQL server can read it.

For information about `caching_sha2_password`, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `caching_sha2_password_public_key_path`

Property	Value
Command-Line Format	`--caching-sha2-password-public-key-path=file_name`
Introduced	8.0.3
System Variable	`caching_sha2_password_public_key_path`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	`public_key.pem`

This variable specifies the path name of the RSA public key file for the `caching_sha2_password` authentication plugin. If the file is named as a relative path, it is interpreted relative to the server data directory. The file must be in PEM format.

For information about `caching_sha2_password`, including information about how clients request the RSA public key, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `character_set_client`

Property	Value
System Variable	`character_set_client`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (>= 8.0.1)	`utf8mb4`
Default Value (8.0.0)	`utf8`

The character set for statements that arrive from the client. The session value of this variable is set using the character set requested by the client when the client connects to the server. (Many clients support a `--default-character-set` option to enable this character set to be specified explicitly. See also Section 10.4, “Connection Character Sets and Collations”.) The global value of the variable is used to set the session value in cases when the client-requested value is unknown or not available, or the server is configured to ignore client requests:

- The client requests a character set not known to the server. For example, a Japanese-enabled client requests `sjis` when connecting to a server not configured with `sjis` support.
- The client is from a version of MySQL older than MySQL 4.1, and thus does not request a character set.
- `mysqld` was started with the `--skip-character-set-client-handshake` option, which causes it to ignore client character set configuration. This reproduces MySQL 4.0 behavior and is useful should you wish to upgrade the server without upgrading all the clients.

Some character sets cannot be used as the client character set. Attempting to use them as the `character_set_client` value produces an error. See Impermissible Client Character Sets.

- `character_set_connection`

Property	Value
System Variable	`character_set_connection`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (>= 8.0.1)	`utf8mb4`
Default Value (8.0.0)	`utf8`

The character set used for literals specified without a character set introducer and for number-to-string conversion. For information about introducers, see Section 10.3.8, “Character Set Introducers”.

- `character_set_database`

Property	Value
System Variable	`character_set_database`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (>= 8.0.1)	`utf8mb4`
Default Value (8.0.0)	`latin1`
Footnote	This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

The character set used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `character_set_server`.

As of MySQL 8.0.14, setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

The global `character_set_database` and `collation_database` system variables are deprecated and will be removed in a future version of MySQL.

Assigning a value to the session `character_set_database` and `collation_database` system variables is deprecated and assignments produce a warning. The session variables will become read only in a future version of MySQL and assignments will produce an error. It will remain possible to access the session variables to determine the database character set and collation for the default database.

- `character_set_filesystem`

Property	Value
Command-Line Format	`--character-set-filesystem=name`
System Variable	`character_set_filesystem`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`binary`

The file system character set. This variable is used to interpret string literals that refer to file names, such as in the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function. Such file names are converted from `character_set_client` to `character_set_filesystem` before the file opening attempt occurs. The default value is `binary`, which means that no conversion occurs. For systems on which multibyte file names are permitted, a different value may be more appropriate. For example, if the system represents file names using UTF-8, set `character_set_filesystem` to `'utf8mb4'`.

As of MySQL 8.0.14, setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

- `character_set_results`

Property	Value
System Variable	`character_set_results`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (>= 8.0.1)	`utf8mb4`
Default Value (8.0.0)	`utf8`

The character set used for returning query results to the client. This includes result data such as column values, result metadata such as column names, and error messages.

- `character_set_server`

Property	Value
Command-Line Format	`--character-set-server=name`
System Variable	`character_set_server`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (>= 8.0.1)	`utf8mb4`
Default Value (8.0.0)	`latin1`

The servers default character set. See Section 10.15, “Character Set Configuration”. If you set this variable, you should also set `collation_server` to specify the collation for the character set.

- `character_set_system`

Property	Value
System Variable	`character_set_system`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`utf8`

The character set used by the server for storing identifiers. The value is always `utf8`.

- `character_sets_dir`

Property	Value
Command-Line Format	`--character-sets-dir=dir_name`
System Variable	`character_sets_dir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `check_proxy_users`

Property	Value
Command-Line Format	`--check-proxy-users[={OFF|ON}]`
System Variable	`check_proxy_users`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Some authentication plugins implement proxy user mapping for themselves (for example, the PAM and Windows authentication plugins). Other authentication plugins do not support proxy users by default. Of these, some can request that the MySQL server itself map proxy users according to granted proxy privileges: mysql_native_password, sha256_password.

If the check_proxy_users system variable is enabled, the server performs proxy user mapping for any authentication plugins that make such a request. However, it may also be necessary to enable plugin-specific system variables to take advantage of server proxy user mapping support:

- For the mysql_native_password plugin, enable mysql_native_password_proxy_users.
- For the sha256_password plugin, enable sha256_password_proxy_users.

For information about user proxying, see Section 6.2.18, “Proxy Users”.

- collation_connection

Property	Value
System Variable	collation_connection
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The collation of the connection character set. collation_connection is important for comparisons of literal strings. For comparisons of strings with column values, collation_connection does not matter because columns have their own collation, which has a higher collation precedence (see Section 10.8.4, “Collation Coercibility in Expressions”).

- collation_database

Property	Value
System Variable	collation_database
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (>= 8.0.1)	utf8mb4_0900_ai_ci
Default Value (8.0.0)	latin1_swedish_ci
Footnote	This option is dynamic, but only the server should set this information. You should not set the value of this variable manually.

The collation used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as collation_server.

As of MySQL 8.0.18, setting the session value of this system variable is no longer a restricted operation.

The global `character_set_database` and `collation_database` system variables are deprecated and will be removed in a future version of MySQL.

Assigning a value to the session `character_set_database` and `collation_database` system variables is deprecated and assignments produce a warning. The session variables will become read only in a future version of MySQL and assignments will produce an error. It will remain possible to access the session variables to determine the database character set and collation for the default database.

- `collation_server`

Property	Value
Command-Line Format	`--collation-server=name`
System Variable	`collation_server`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (>= 8.0.1)	`utf8mb4_0900_ai_ci`
Default Value (8.0.0)	`latin1_swedish_ci`

The server's default collation. See Section 10.15, “Character Set Configuration”.

- `completion_type`

Property	Value
Command-Line Format	`--completion-type=#`
System Variable	`completion_type`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`NO_CHAIN`

Valid Values	`NO_CHAIN`
	`CHAIN`
	`RELEASE`
	`0`
	`1`
	`2`

The transaction completion type. This variable can take the values shown in the following table. The variable can be assigned using either the name values or corresponding integer values.

Value	Description
NO_CHAIN (or 0)	COMMIT and ROLLBACK are unaffected. This is the default value.
CHAIN (or 1)	COMMIT and ROLLBACK are equivalent to COMMIT AND CHAIN and ROLLBACK AND CHAIN, respectively. (A new transaction starts immediately with the same isolation level as the just-terminated transaction.)
RELEASE (or 2)	COMMIT and ROLLBACK are equivalent to COMMIT RELEASE and ROLLBACK RELEASE, respectively. (The server disconnects after terminating the transaction.)

completion_type affects transactions that begin with START TRANSACTION or BEGIN and end with COMMIT or ROLLBACK. It does not apply to implicit commits resulting from execution of the statements listed in Section 13.3.3, “Statements That Cause an Implicit Commit”. It also does not apply for XA COMMIT, XA ROLLBACK, or when autocommit=1.

- concurrent_insert

Property	Value
Command-Line Format	--concurrent-insert[=#]
System Variable	concurrent_insert
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	AUTO
Valid Values	NEVER AUTO ALWAYS 0 1 2

If AUTO (the default), MySQL permits INSERT and SELECT statements to run concurrently for MyISAM tables that have no free blocks in the middle of the data file. If you start mysqld with --skip-new, this variable is set to NEVER.

This variable can take the values shown in the following table. The variable can be assigned using either the name values or corresponding integer values.

Value	Description
NEVER (or 0)	Disables concurrent inserts
AUTO (or 1)	(Default) Enables concurrent insert for MyISAM tables that do not have holes
ALWAYS (or 2)	Enables concurrent inserts for all MyISAM tables, even those that have holes. For a table with a hole, new rows are inserted at the end of the table if it is in use by another thread. Otherwise, MySQL acquires a normal write lock and inserts the row into the hole.

See also Section 8.11.3, “Concurrent Inserts”.

- connect_timeout

Property	Value
Command-Line Format	--connect-timeout=#
System Variable	connect_timeout
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	10
Minimum Value	2
Maximum Value	31536000

The number of seconds that the mysqld server waits for a connect packet before responding with Bad handshake. The default value is 10 seconds.

Increasing the connect_timeout value might help if clients frequently encounter errors of the form Lost connection to MySQL server at '*XXX*', system error: *errno*.

- core_file

Property	Value
System Variable	core_file
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Whether to write a core file if the server crashes. This variable is set by the --core-file option.

- create_admin_listener_thread

Property	Value
Command-Line Format	`--create-admin-listener-thread[={OFF|ON}]`
Introduced	8.0.14
System Variable	`create_admin_listener_thread`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether to use a dedicated listening thread for client connections on the administrative network interface (see Section 8.12.4.1, “How MySQL Handles Client Connections”). The default is to implement the administrative interface using the listener thread used for ordinary connections.

Depending on factors such as platform type and workload, you may find one setting for this variable yields better performance than the other setting.

Setting `create_admin_listener_thread` has no effect if `admin_address` is not specified because in that case the server maintains no administrative network interface.

- `cte_max_recursion_depth`

Property	Value
Command-Line Format	`--cte-max-recursion-depth=#`
Introduced	8.0.3
System Variable	`cte_max_recursion_depth`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1000`
Minimum Value	`0`
Maximum Value	`4294967295`

The common table expression (CTE) maximum recursion depth. The server terminates execution of any CTE that recurses more levels than the value of this variable. For more information, see Limiting Common Table Expression Recursion.

- `datadir`

Property	Value
Command-Line Format	`--datadir=dir_name`
System Variable	`datadir`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	Directory name

The path to the MySQL server data directory. Relative paths are resolved with respect to the current directory. If the server will be started automatically (that is, in contexts for which you cannot assume what the current directory will be), it is best to specify the `datadir` value as an absolute path.

- `date_format`

  This system variable was removed in MySQL 8.0.3.

- `datetime_format`

  This system variable was removed in MySQL 8.0.3.

- `debug`

Property	Value
Command-Line Format	`--debug[=debug_options]`
System Variable	`debug`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (Windows)	`d:t:i:O,\mysqld.trace`
Default Value (Unix)	`d:t:i:o,/tmp/mysqld.trace`

This variable indicates the current debugging settings. It is available only for servers built with debugging support. The initial value comes from the value of instances of the `--debug` option given at server startup. The global and session values may be set at runtime.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Assigning a value that begins with `+` or `-` cause the value to added to or subtracted from the current value:

```
mysql> SET debug = 'T';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| T       |
+---------+

mysql> SET debug = '+P';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| P:T     |
+---------+
```

```
mysql> SET debug = '-P';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| T       |
+---------+
```

For more information, see Section 29.5.4, “The DBUG Package”.

- `debug_sync`

Property	Value
System Variable	`debug_sync`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

This variable is the user interface to the Debug Sync facility. Use of Debug Sync requires that MySQL be configured with the `-DENABLE_DEBUG_SYNC=1 CMake` option (see Section 2.9.7, “MySQL Source-Configuration Options”). If Debug Sync is not compiled in, this system variable is not available.

The global variable value is read only and indicates whether the facility is enabled. By default, Debug Sync is disabled and the value of `debug_sync` is `OFF`. If the server is started with `--debug-sync-timeout=N`, where *`N`* is a timeout value greater than 0, Debug Sync is enabled and the value of `debug_sync` is `ON - current signal` followed by the signal name. Also, *`N`* becomes the default timeout for individual synchronization points.

The session value can be read by any user and will have the same value as the global variable. The session value can be set to control synchronization points.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `default_authentication_plugin`

Property	Value
Command-Line Format	`--default-authentication-plugin=plugin_name`
System Variable	`default_authentication_plugin`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Enumeration
Default Value (>= 8.0.4)	`caching_sha2_password`
Default Value (<= 8.0.3)	`mysql_native_password`
Valid Values (>= 8.0.3)	`mysql_native_password`

Property	Value

	`sha256_password`
	
	`caching_sha2_password`

Valid Values (<= 8.0.2)	`mysql_native_password`
	
	`sha256_password`

The default authentication plugin. These values are permitted:

- `mysql_native_password`: Use MySQL native passwords; see Section 6.4.1.1, “Native Pluggable Authentication”.
- `sha256_password`: Use SHA-256 passwords; see Section 6.4.1.3, “SHA-256 Pluggable Authentication”.
- `caching_sha2_password`: Use SHA-256 passwords; see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

Note

In MySQL 8.0, `caching_sha2_password` is the default authentication plugin rather than `mysql_native_password`. For information about the implications of this change for server operation and compatibility of the server with clients and connectors, see caching_sha2_password as the Preferred Authentication Plugin.

The `default_authentication_plugin` value affects these aspects of server operation:

- It determines which authentication plugin the server assigns to new accounts created by `CREATE USER` and `GRANT` statements that do not explicitly specify an authentication plugin.
- For an account created with the following statement, the server associates the account with the default authentication plugin and assigns the account the given password, hashed as required by that plugin:

```
CREATE USER ... IDENTIFIED BY 'cleartext password';
```

- `default_collation_for_utf8mb4`

Property	Value
Introduced	8.0.11
System Variable	`default_collation_for_utf8mb4`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration

Valid Values	`utf8mb4_0900_ai_ci`
	
	`utf8mb4_general_ci`

For internal use by replication. This system variable is set to the default collation for the `utf8mb4` character set. The value of the variable is replicated from a master to a slave so that the slave can

correctly process data originating from a master with a different default collation for `utf8mb4`. This variable is primarily intended to support replication from a MySQL 5.7 or older master server to a MySQL 8.0 slave server, or group replication with a MySQL 5.7 primary node and one or more MySQL 8.0 secondaries. The default collation for `utf8mb4` in MySQL 5.7 is `utf8mb4_general_ci`, but `utf8mb4_0900_ai_ci` in MySQL 8.0. The variable is not present in releases earlier than MySQL 8.0, so if the slave does not receive a value for the variable, it assumes the master is from an earlier release and sets the value to the previous default collation `utf8mb4_general_ci`.

As of MySQL 8.0.18, setting the session value of this system variable is no longer a restricted operation.

The default `utf8mb4` collation is used in the following statements:

- `SHOW COLLATION` and `SHOW CHARACTER SET`.
- `CREATE TABLE` and `ALTER TABLE` having a `CHARACTER SET utf8mb4` clause without a `COLLATION` clause, either for the table character set or for a column character set.
- `CREATE DATABASE` and `ALTER DATABASE` having a `CHARACTER SET utf8mb4` clause without a `COLLATION` clause.
- Any statement containing a string literal of the form `_utf8mb4'*some text*'` without a `COLLATE` clause.

- `default_password_lifetime`

Property	Value
Command-Line Format	`--default-password-lifetime=#`
System Variable	`default_password_lifetime`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`65535`

This variable defines the global automatic password expiration policy. The default `default_password_lifetime` value is 0, which disables automatic password expiration. If the value of `default_password_lifetime` is a positive integer *`N`*, it indicates the permitted password lifetime; passwords must be changed every *`N`* days.

The global password expiration policy can be overridden as desired for individual accounts using the password expiration option of the `CREATE USER` and `ALTER USER` statements. See Section 6.2.15, “Password Management”.

- `default_storage_engine`

Property	Value
Command-Line Format	`--default-storage-engine=name`
System Variable	`default_storage_engine`
Scope	Global, Session

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`InnoDB`

The default storage engine for tables. See Chapter 16, *Alternative Storage Engines*. This variable sets the storage engine for permanent tables only. To set the storage engine for `TEMPORARY` tables, set the `default_tmp_storage_engine` system variable.

To see which storage engines are available and enabled, use the `SHOW ENGINES` statement or query the `INFORMATION_SCHEMA ENGINES` table.

If you disable the default storage engine at server startup, you must set the default engine for both permanent and `TEMPORARY` tables to a different engine or the server will not start.

- `default_table_encryption`

Property	Value
Command-Line Format	`--default-table-encryption[={OFF|ON}]`
Introduced	8.0.16
System Variable	`default_table_encryption`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	`OFF`

Defines the default encryption setting applied to schemas and general tablespaces when they are created without specifying an `ENCRYPTION` clause.

The `default_table_encryption` variable is only applicable to user-created schemas and general tablespaces. It does not govern encryption of the `mysql` system tablespace.

Setting the runtime value of `default_table_encryption` requires the `SYSTEM_VARIABLES_ADMIN` and `TABLE_ENCRYPTION_ADMIN` privileges, or the `SUPER` privilege.

`default_table_encryption` supports `SET PERSIST` and `SET PERSIST_ONLY` syntax. See Section 5.1.9.3, “Persisted System Variables”.

For more information, see Defining an Encryption Default for Schemas and General Tablespaces.

- `default_tmp_storage_engine`

Property	Value
Command-Line Format	`--default-tmp-storage-engine=name`
System Variable	`default_tmp_storage_engine`
Scope	Global, Session
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	Yes
Type	Enumeration
Default Value	`InnoDB`

The default storage engine for `TEMPORARY` tables (created with `CREATE TEMPORARY TABLE`). To set the storage engine for permanent tables, set the `default_storage_engine` system variable. Also see the discussion of that variable regarding possible values.

If you disable the default storage engine at server startup, you must set the default engine for both permanent and `TEMPORARY` tables to a different engine or the server will not start.

- `default_week_format`

Property	Value
Command-Line Format	`--default-week-format=#`
System Variable	`default_week_format`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`7`

The default mode value to use for the `WEEK()` function. See Section 12.7, “Date and Time Functions”.

- `delay_key_write`

Property	Value
Command-Line Format	`--delay-key-write[={OFF|ON|ALL}]`
System Variable	`delay_key_write`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`ALL`

This variable specifies how to use delayed key writes. It applies only to `MyISAM` tables. Delayed key writing causes key buffers not to be flushed between writes. See also Section 16.2.1, “MyISAM Startup Options”.

This variable can have one of the following values to affect handling of the DELAY_KEY_WRITE table option that can be used in CREATE TABLE statements.

Option	Description
OFF	DELAY_KEY_WRITE is ignored.
ON	MySQL honors any DELAY_KEY_WRITE option specified in CREATE TABLE statements. This is the default value.
ALL	All new opened tables are treated as if they were created with the DELAY_KEY_WRITE option enabled.

Note

If you set this variable to ALL, you should not use MyISAM tables from within another program (such as another MySQL server or myisamchk) when the tables are in use. Doing so leads to index corruption.

If DELAY_KEY_WRITE is enabled for a table, the key buffer is not flushed for the table on every index update, but only when the table is closed. This speeds up writes on keys a lot, but if you use this feature, you should add automatic checking of all MyISAM tables by starting the server with the myisam_recover_options system variable set (for example, myisam_recover_options='BACKUP,FORCE'). See Section 5.1.8, “Server System Variables”, and Section 16.2.1, “MyISAM Startup Options”.

Warning

If you enable external locking with --external-locking, there is no protection against index corruption for tables that use delayed key writes.

- delayed_insert_limit

Property	Value
Command-Line Format	--delayed-insert-limit=#
Deprecated	Yes
System Variable	delayed_insert_limit
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	100
Minimum Value	1
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

This system variable is deprecated (because DELAYED inserts are not supported), and will be removed in a future release.

- delayed_insert_timeout

Property	Value
Command-Line Format	`--delayed-insert-timeout=#`
Deprecated	Yes
System Variable	`delayed_insert_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`300`

This system variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `delayed_queue_size`

Property	Value
Command-Line Format	`--delayed-queue-size=#`
Deprecated	Yes
System Variable	`delayed_queue_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1000`
Minimum Value	`1`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

This system variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `disabled_storage_engines`

Property	Value
Command-Line Format	`--disabled-storage-engines=engine[,engine]...`
System Variable	`disabled_storage_engines`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`empty string`

This variable indicates which storage engines cannot be used to create tables or tablespaces. For example, to prevent new `MyISAM` or `FEDERATED` tables from being created, start the server with these lines in the server option file:

```
[mysqld]
disabled_storage_engines="MyISAM,FEDERATED"
```

By default, `disabled_storage_engines` is empty (no engines disabled), but it can be set to a comma-separated list of one or more engines (not case sensitive). Any engine named in the value cannot be used to create tables or tablespaces with `CREATE TABLE` or `CREATE TABLESPACE`, and cannot be used with `ALTER TABLE ... ENGINE` or `ALTER TABLESPACE ... ENGINE` to change the storage engine of existing tables or tablespaces. Attempts to do so result in an `ER_DISABLED_STORAGE_ENGINE` error.

`disabled_storage_engines` does not restrict other DDL statements for existing tables, such as `CREATE INDEX`, `TRUNCATE TABLE`, `ANALYZE TABLE`, `DROP TABLE`, or `DROP TABLESPACE`. This permits a smooth transition so that existing tables or tablespaces that use a disabled engine can be migrated to a permitted engine by means such as `ALTER TABLE ... ENGINE` *`permitted_engine`*.

It is permitted to set the `default_storage_engine` or `default_tmp_storage_engine` system variable to a storage engine that is disabled. This could cause applications to behave erratically or fail, although that might be a useful technique in a development environment for identifying applications that use disabled engines, so that they can be modified.

`disabled_storage_engines` is disabled and has no effect if the server is started with any of these options: `--initialize`, `--initialize-insecure`, `--skip-grant-tables`.

Note

Setting `disabled_storage_engines` might cause an issue with `mysql_upgrade`. For details, see Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

- `disconnect_on_expired_password`

Property	Value
Command-Line Format	`--disconnect-on-expired-password[={OFF|ON}]`
System Variable	`disconnect_on_expired_password`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

This variable controls how the server handles clients with expired passwords:

- If the client indicates that it can handle expires passwords, the value of `disconnect_on_expired_password` is irrelevant. The server permits the client to connect but puts it in sandbox mode.

- If the client does not indicate that it can handle expires passwords, the server handles the client according to the value of `disconnect_on_expired_password`:
  - If `disconnect_on_expired_password`: is enabled, the server disconnects the client.
  - If `disconnect_on_expired_password`: is disabled, the server permits the client to connect but puts it in sandbox mode.

For more information about the interaction of client and server settings relating to expired-password handling, see Section 6.2.16, “Server Handling of Expired Passwords”.

- `div_precision_increment`

Property	Value
Command-Line Format	`--div-precision-increment=#`
System Variable	`div_precision_increment`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`4`
Minimum Value	`0`
Maximum Value	`30`

This variable indicates the number of digits by which to increase the scale of the result of division operations performed with the `/` operator. The default value is 4. The minimum and maximum values are 0 and 30, respectively. The following example illustrates the effect of increasing the default value.

```
mysql> SELECT 1/7;
+--------+
| 1/7    |
+--------+
| 0.1429 |
+--------+
mysql> SET div_precision_increment = 12;
mysql> SELECT 1/7;
+----------------+
| 1/7            |
+----------------+
| 0.142857142857 |
+----------------+
```

- `dragnet.log_error_filter_rules`

Property	Value
Command-Line Format	`--dragnet.log-error-filter-rules=value`
Introduced	8.0.4
System Variable	`dragnet.log_error_filter_rules`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	String
Default Value	`IF prio>=INFORMATION THEN drop. IF EXISTS source_line THEN unset source_line.`

The filter rules that control operation of the `log_filter_dragnet` error log filter component. If `log_filter_dragnet` is not installed, `dragnet.log_error_filter_rules` is unavailable. If `log_filter_dragnet` is installed but not enabled, changes to `dragnet.log_error_filter_rules` have no effect.

As of MySQL 8.0.12, the `dragnet.Status` status variable can be consulted to determine the result of the most recent assignment to `dragnet.log_error_filter_rules`.

Prior to MySQL 8.0.12, successful assignments to `dragnet.log_error_filter_rules` at runtime produce a note confirming the new value:

```
mysql> SET GLOBAL dragnet.log_error_filter_rules = 'IF prio <> 0 THEN unset prio.';
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 4569
Message: filter configuration accepted:
         SET @@GLOBAL.dragnet.log_error_filter_rules=
         'IF prio!=ERROR THEN unset prio.';
```

The value displayed by `SHOW WARNINGS` indicates the “decompiled” canonical representation after the rule set has been successfully parsed and compiled into internal form. Semantically, this canonical form is identical to the value assigned to `dragnet.log_error_filter_rules`, but there may be some differences between the assigned and canonical values, as illustrated by the preceding example:

- The `<>` operator is changed to `!=`.
- The numeric priority of 0 is changed to the corresponding priority symbol `ERROR`.
- Optional spaces are removed.

For additional information, see Section 5.4.2.3, “Types of Error Log Filtering”, and Section 5.5.3, “Error Log Components”.

- `end_markers_in_json`

Property	Value
Command-Line Format	`--end-markers-in-json[={OFF|ON}]`
System Variable	`end_markers_in_json`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	`OFF`

Whether optimizer JSON output should add end markers. See MySQL Internals: The end_markers_in_json System Variable.

- `eq_range_index_dive_limit`

Property	Value
Command-Line Format	`--eq-range-index-dive-limit=#`
System Variable	`eq_range_index_dive_limit`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`200`
Minimum Value	`0`
Maximum Value	`4294967295`

This variable indicates the number of equality ranges in an equality comparison condition when the optimizer should switch from using index dives to index statistics in estimating the number of qualifying rows. It applies to evaluation of expressions that have either of these equivalent forms, where the optimizer uses a nonunique index to look up *`col_name`* values:

```
col_name IN(val1, ..., valN)
col_name = val1 OR ... OR col_name = valN
```

In both cases, the expression contains *`N`* equality ranges. The optimizer can make row estimates using index dives or index statistics. If `eq_range_index_dive_limit` is greater than 0, the optimizer uses existing index statistics instead of index dives if there are `eq_range_index_dive_limit` or more equality ranges. Thus, to permit use of index dives for up to *`N`* equality ranges, set `eq_range_index_dive_limit` to *`N`* + 1. To disable use of index statistics and always use index dives regardless of *`N`*, set `eq_range_index_dive_limit` to 0.

For more information, see Equality Range Optimization of Many-Valued Comparisons.

To update table index statistics for best estimates, use `ANALYZE TABLE`.

- `error_count`

The number of errors that resulted from the last statement that generated messages. This variable is read only. See Section 13.7.7.17, “SHOW ERRORS Syntax”.

- `event_scheduler`

Property	Value
Command-Line Format	`--event-scheduler[=value]`
System Variable	`event_scheduler`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Enumeration
Default Value (>= 8.0.3)	`ON`
Default Value (<= 8.0.2)	`OFF`

Valid Values	`ON`
	`OFF`
	`DISABLED`

This variable enables or disables, and starts or stops, the Event Scheduler. The possible status values are `ON`, `OFF`, and `DISABLED`, with the default being `OFF`. Turning the Event Scheduler `OFF` is not the same as disabling the Event Scheduler, which requires setting the status to `DISABLED`. This variable and its effects on the Event Scheduler's operation are discussed in greater detail in Section 24.4.2, “Event Scheduler Configuration”

- `explicit_defaults_for_timestamp`

Property	Value
Command-Line Format	`--explicit-defaults-for-timestamp[={OFF|ON}]`
Deprecated	Yes
System Variable	`explicit_defaults_for_timestamp`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value (>= 8.0.2)	`ON`
Default Value (<= 8.0.1)	`OFF`

This system variable determines whether the server enables certain nonstandard behaviors for default values and `NULL`-value handling in `TIMESTAMP` columns. By default, `explicit_defaults_for_timestamp` is enabled, which disables the nonstandard behaviors. Disabling `explicit_defaults_for_timestamp` results in a warning.

As of MySQL 8.0.18, setting the session value of this system variable is no longer a restricted operation.

If `explicit_defaults_for_timestamp` is disabled, the server enables the nonstandard behaviors and handles `TIMESTAMP` columns as follows:

- `TIMESTAMP` columns not explicitly declared with the `NULL` attribute are automatically declared with the `NOT NULL` attribute. Assigning such a column a value of `NULL` is permitted and sets the column to the current timestamp.
- The first `TIMESTAMP` column in a table, if not explicitly declared with the `NULL` attribute or an explicit `DEFAULT` or `ON UPDATE` attribute, is automatically declared with the `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` attributes.
- `TIMESTAMP` columns following the first one, if not explicitly declared with the `NULL` attribute or an explicit `DEFAULT` attribute, are automatically declared as `DEFAULT '0000-00-00 00:00:00'` (the

"zero" timestamp). For inserted rows that specify no explicit value for such a column, the column is assigned `'0000-00-00 00:00:00'` and no warning occurs.

Depending on whether strict SQL mode or the `NO_ZERO_DATE` SQL mode is enabled, a default value of `'0000-00-00 00:00:00'` may be invalid. Be aware that the `TRADITIONAL` SQL mode includes strict mode and `NO_ZERO_DATE`. See Section 5.1.11, "Server SQL Modes".

The nonstandard behaviors just described are deprecated and will be removed in a future MySQL release.

If `explicit_defaults_for_timestamp` is enabled, the server disables the nonstandard behaviors and handles `TIMESTAMP` columns as follows:

- It is not possible to assign a `TIMESTAMP` column a value of `NULL` to set it to the current timestamp. To assign the current timestamp, set the column to `CURRENT_TIMESTAMP` or a synonym such as `NOW()`.
- `TIMESTAMP` columns not explicitly declared with the `NOT NULL` attribute are automatically declared with the `NULL` attribute and permit `NULL` values. Assigning such a column a value of `NULL` sets it to `NULL`, not the current timestamp.
- `TIMESTAMP` columns declared with the `NOT NULL` attribute do not permit `NULL` values. For inserts that specify `NULL` for such a column, the result is either an error for a single-row insert or if strict SQL mode is enabled, or `'0000-00-00 00:00:00'` is inserted for multiple-row inserts with strict SQL mode disabled. In no case does assigning the column a value of `NULL` set it to the current timestamp.
- `TIMESTAMP` columns explicitly declared with the `NOT NULL` attribute and without an explicit `DEFAULT` attribute are treated as having no default value. For inserted rows that specify no explicit value for such a column, the result depends on the SQL mode. If strict SQL mode is enabled, an error occurs. If strict SQL mode is not enabled, the column is declared with the implicit default of `'0000-00-00 00:00:00'` and a warning occurs. This is similar to how MySQL treats other temporal types such as `DATETIME`.
- No `TIMESTAMP` column is automatically declared with the `DEFAULT CURRENT_TIMESTAMP` or `ON UPDATE CURRENT_TIMESTAMP` attributes. Those attributes must be explicitly specified.
- The first `TIMESTAMP` column in a table is not handled differently from `TIMESTAMP` columns following the first one.

If `explicit_defaults_for_timestamp` is disabled at server startup, this warning appears in the error log:

```
[Warning] TIMESTAMP with implicit DEFAULT value is deprecated.
Please use --explicit_defaults_for_timestamp server option (see
documentation for more details).
```

As indicated by the warning, to disable the deprecated nonstandard behaviors, enable the `explicit_defaults_for_timestamp` system variable at server startup.

Note

`explicit_defaults_for_timestamp` is itself deprecated because its only purpose is to permit control over deprecated `TIMESTAMP` behaviors that are to be removed in a future MySQL release. When removal of those behaviors occurs, `explicit_defaults_for_timestamp` will have no purpose and will be removed as well.

For additional information, see Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

- `external_user`

Property	Value
System Variable	`external_user`
Scope	Session
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this variable is `NULL`. See Section 6.2.18, “Proxy Users”.

- `flush`

Property	Value
Command-Line Format	`--flush[={OFF|ON}]`
System Variable	`flush`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

If `ON`, the server flushes (synchronizes) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See Section B.4.3.3, “What to Do If MySQL Keeps Crashing”. This variable is set to `ON` if you start `mysqld` with the `--flush` option.

Note

If `flush` is enabled, the value of `flush_time` does not matter and changes to `flush_time` have no effect on flush behavior.

- `flush_time`

Property	Value
Command-Line Format	`--flush-time=#`
System Variable	`flush_time`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`

Property	Value
Minimum Value	`0`

If this is set to a nonzero value, all tables are closed every `flush_time` seconds to free up resources and synchronize unflushed data to disk. This option is best used only on systems with minimal resources.

Note

If `flush` is enabled, the value of `flush_time` does not matter and changes to `flush_time` have no effect on flush behavior.

- `foreign_key_checks`

Property	Value
System Variable	`foreign_key_checks`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	`ON`

If set to 1 (the default), foreign key constraints for `InnoDB` tables are checked. If set to 0, foreign key constraints are ignored, with a couple of exceptions. When re-creating a table that was dropped, an error is returned if the table definition does not conform to the foreign key constraints referencing the table. Likewise, an `ALTER TABLE` operation returns an error if a foreign key definition is incorrectly formed. For more information, see Section 13.1.20.6, “Using FOREIGN KEY Constraints”.

Setting this variable has the same effect on `NDB` tables as it does for `InnoDB` tables. Typically you leave this setting enabled during normal operation, to enforce referential integrity. Disabling foreign key checking can be useful for reloading `InnoDB` tables in an order different from that required by their parent/child relationships. See Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”.

Setting `foreign_key_checks` to 0 also affects data definition statements: `DROP SCHEMA` drops a schema even if it contains tables that have foreign keys that are referred to by tables outside the schema, and `DROP TABLE` drops tables that have foreign keys that are referred to by other tables.

Note

Setting `foreign_key_checks` to 1 does not trigger a scan of the existing table data. Therefore, rows added to the table while `foreign_key_checks = 0` will not be verified for consistency.

Dropping an index required by a foreign key constraint is not permitted, even with `foreign_key_checks=0`. The foreign key constraint must be removed before dropping the index.

- `ft_boolean_syntax`

Property	Value
Command-Line Format	`--ft-boolean-syntax=name`
System Variable	`ft_boolean_syntax`

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`+ -><()~*:""&|`

The list of operators supported by boolean full-text searches performed using `IN BOOLEAN MODE`. See Section 12.9.2, “Boolean Full-Text Searches”.

The default variable value is `'+ -><()~*:""&|'`. The rules for changing the value are as follows:

- Operator function is determined by position within the string.
- The replacement value must be 14 characters.
- Each character must be an ASCII nonalphanumeric character.
- Either the first or second character must be a space.
- No duplicates are permitted except the phrase quoting operators in positions 11 and 12. These two characters are not required to be the same, but they are the only two that may be.
- Positions 10, 13, and 14 (which by default are set to `:`, `&`, and `|`) are reserved for future extensions.

- `ft_max_word_len`

Property	Value
Command-Line Format	`--ft-max-word-len=#`
System Variable	`ft_max_word_len`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Minimum Value	`10`

The maximum length of the word to be included in a `MyISAM FULLTEXT` index.

Note

`FULLTEXT` indexes on `MyISAM` tables must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_min_word_len`

Property	Value
Command-Line Format	`--ft-min-word-len=#`
System Variable	`ft_min_word_len`
Scope	Global

Property	Value
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	4
Minimum Value	1

The minimum length of the word to be included in a MyISAM FULLTEXT index.

Note

FULLTEXT indexes on MyISAM tables must be rebuilt after changing this variable. Use REPAIR TABLE *tbl_name* QUICK.

- ft_query_expansion_limit

Property	Value
Command-Line Format	--ft-query-expansion-limit=#
System Variable	ft_query_expansion_limit
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	20
Minimum Value	0
Maximum Value	1000

The number of top matches to use for full-text searches performed using WITH QUERY EXPANSION.

- ft_stopword_file

Property	Value
Command-Line Format	--ft-stopword-file=file_name
System Variable	ft_stopword_file
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

The file from which to read the list of stopwords for full-text searches on MyISAM tables. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. All the words from the file are used; comments are *not* honored. By default, a built-in list of stopwords is used (as defined in the storage/myisam/ft_static.c file). Setting this variable to the empty string ('') disables stopword filtering. See also Section 12.9.4, “Full-Text Stopwords”.

Note

`FULLTEXT` indexes on `MyISAM` tables must be rebuilt after changing this variable or the contents of the stopword file. Use `REPAIR TABLE tbl_name QUICK`.

- `general_log`

Property	Value
Command-Line Format	`--general-log[={OFF|ON}]`
System Variable	`general_log`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether the general query log is enabled. The value can be 0 (or `OFF`) to disable the log or 1 (or `ON`) to enable the log. The destination for log output is controlled by the `log_output` system variable; if that value is `NONE`, no log entries are written even if the log is enabled.

- `general_log_file`

Property	Value
Command-Line Format	`--general-log-file=file_name`
System Variable	`general_log_file`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	`host_name.log`

The name of the general query log file. The default value is *`host_name`*`.log`, but the initial value can be changed with the `--general_log_file` option.

- `generated_random_password_length`

Property	Value
Command-Line Format	`--generated-random-password-length=#`
Introduced	8.0.18
System Variable	`generated_random_password_length`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`20`

Property	Value
Minimum Value	5
Maximum Value	255

The maximum number of characters permitted in random passwords generated for `CREATE USER`, `ALTER USER`, and `SET PASSWORD` statements. For more information, see Random Password Generation.

- `group_concat_max_len`

Property	Value
Command-Line Format	`--group-concat-max-len=#`
System Variable	`group_concat_max_len`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	1024
Minimum Value	4
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

The maximum permitted result length in bytes for the `GROUP_CONCAT()` function. The default is 1024.

- `have_compress`

`YES` if the `zlib` compression library is available to the server, `NO` if not. If not, the `COMPRESS()` and `UNCOMPRESS()` functions cannot be used.

- `have_crypt`

This system variable was removed in MySQL 8.0.3.

- `have_dynamic_loading`

`YES` if `mysqld` supports dynamic loading of plugins, `NO` if not. If the value is `NO`, you cannot use options such as `--plugin-load` to load plugins at server startup, or the `INSTALL PLUGIN` statement to load plugins at runtime.

- `have_geometry`

`YES` if the server supports spatial data types, `NO` if not.

- `have_openssl`

This variable is an alias for `have_ssl`.

- `have_profiling`

YES if statement profiling capability is present, NO if not. If present, the profiling system variable controls whether this capability is enabled or disabled. See Section 13.7.7.31, “SHOW PROFILES Syntax”.

This variable is deprecated and will be removed in a future MySQL release.

- have_query_cache

  The query cache was removed in MySQL 8.0.3. have_query_cache is deprecated, always has a value of NO, and will be removed in a future MySQL release.

- have_rtree_keys

  YES if RTREE indexes are available, NO if not. (These are used for spatial indexes in MyISAM tables.)

- have_ssl

  YES if mysqld supports SSL connections, NO if not. DISABLED indicates that the server was compiled with SSL support, but was not started with the appropriate --ssl-*xxx* options. For more information, see Section 2.9.6, “Configuring SSL Library Support”.

- have_statement_timeout

Property	Value
System Variable	have_statement_timeout
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean

  Whether the statement execution timeout feature is available (see Statement Execution Time Optimizer Hints). The value can be NO if the background thread used by this feature could not be initialized.

- have_symlink

  YES if symbolic link support is enabled, NO if not. This is required on Unix for support of the DATA DIRECTORY and INDEX DIRECTORY table options. If the server is started with the --skip-symbolic-links option, the value is DISABLED.

  This variable has no meaning on Windows.

Note

Symbolic link support, along with the --symbolic-links option that controls it, is deprecated and will be removed in a future version of MySQL. In addition, the option is disabled by default. The related have_symlink system variable also is deprecated and will be removed in a future version of MySQL.

- histogram_generation_max_mem_size

Property	Value
Command-Line Format	--histogram-generation-max-mem-size=#
Introduced	8.0.2

Property	Value
System Variable	`histogram_generation_max_mem_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`20000000`
Minimum Value	`1000000`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The maximum amount of memory available for generating histogram statistics. See Section 8.9.6, “Optimizer Statistics”, and Section 13.7.3.1, “ANALYZE TABLE Syntax”.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

- `host_cache_size`

Property	Value
Command-Line Format	`--host-cache-size=#`
System Variable	`host_cache_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`0`
Maximum Value	`65536`

This variable controls the size of the host cache, as well as the size of the Performance Schema `host_cache` table that exposes the cache contents. Setting the size to 0 disables the host cache. Changing the cache size at runtime causes an implicit `FLUSH HOSTS` operation that clears the host cache, truncates the `host_cache` table, and unblocks any blocked hosts.

The default value is autosized to 128, plus 1 for a value of `max_connections` up to 500, plus 1 for every increment of 20 over 500 in the `max_connections` value, capped to a limit of 2000.

Using the `--skip-host-cache` option is similar to setting the `host_cache_size` system variable to 0, but `host_cache_size` is more flexible because it can also be used to resize, enable, and disable the host cache at runtime, not just at server startup. Starting the server with `--skip-host-cache` does not prevent changes to the value of `host_cache_size`, but such changes have no effect and the cache is not re-enabled even if `host_cache_size` is set larger than 0 at runtime.

For more information about how the host cache works, see Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

- `hostname`

Property	Value
System Variable	`hostname`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The server sets this variable to the server host name at startup.

- `identity`

This variable is a synonym for the `last_insert_id` variable. It exists for compatibility with other database systems. You can read its value with `SELECT @@identity`, and set it using `SET identity`.

- `init_connect`

Property	Value
Command-Line Format	`--init-connect=name`
System Variable	`init_connect`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

A string to be executed by the server for each client that connects. The string consists of one or more SQL statements, separated by semicolon characters.

For users that have the `CONNECTION_ADMIN` or `SUPER` privilege, the content of `init_connect` is not executed. This is done so that an erroneous value for `init_connect` does not prevent all clients from connecting. For example, the value might contain a statement that has a syntax error, thus causing client connections to fail. Not executing `init_connect` for users that have the `CONNECTION_ADMIN` or `SUPER` privilege enables them to open a connection and fix the `init_connect` value.

As of MySQL 8.0.5, `init_connect` execution is skipped for any client user with an expired password. This is done because such a user cannot execute arbitrary statements, and thus `init_connect` execution will fail, leaving the client unable to connect. Skipping `init_connect` execution enables the user to connect and change password.

The server discards any result sets produced by statements in the value of `init_connect`.

- `information_schema_stats_expiry`

Property	Value
Command-Line Format	`--information-schema-stats-expiry=#`
Introduced	8.0.3
System Variable	`information_schema_stats_expiry`
Scope	Global, Session

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	86400
Minimum Value	0
Maximum Value	31536000

Some INFORMATION_SCHEMA tables contain columns that provide table statistics:

```
STATISTICS.CARDINALITY
TABLES.AUTO_INCREMENT
TABLES.AVG_ROW_LENGTH
TABLES.CHECKSUM
TABLES.CHECK_TIME
TABLES.CREATE_TIME
TABLES.DATA_FREE
TABLES.DATA_LENGTH
TABLES.INDEX_LENGTH
TABLES.MAX_DATA_LENGTH
TABLES.TABLE_ROWS
TABLES.UPDATE_TIME
```

Those columns represent dynamic table metadata; that is, information that changes as table contents change.

By default, MySQL retrieves cached values for those columns from the mysql.index_stats and mysql.table_stats dictionary tables when the columns are queried, which is more efficient than retrieving statistics directly from the storage engine. If cached statistics are not available or have expired, MySQL retrieves the latest statistics from the storage engine and caches them in the mysql.index_stats and mysql.table_stats dictionary tables. Subsequent queries retrieve the cached statistics until the cached statistics expire.

The information_schema_stats_expiry session variable defines the period of time before cached statistics expire. The default is 86400 seconds (24 hours), but the time period can be extended to as much as one year.

To update cached values at any time for a given table, use ANALYZE TABLE.

To always retrieve the latest statistics directly from the storage engine and bypass cached values, set information_schema_stats_expiry to 0.

Querying statistics columns does not store or update statistics in the mysql.index_stats and mysql.table_stats dictionary tables under these circumstances:

- When cached statistics have not expired.
- When information_schema_stats_expiry is set to 0.
- When the server is started in read_only, super_read_only, transaction_read_only, or innodb_read_only mode.
- When the query also fetches Performance Schema data.

`information_schema_stats_expiry` is a session variable, and each client session can define its own expiration value. Statistics that are retrieved from the storage engine and cached by one session are available to other sessions.

For related information, see Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”.

- `init_file`

Property	Value
Command-Line Format	`--init-file=file_name`
System Variable	`init_file`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

If specified, this variable names a file containing SQL statements to be read and executed during the startup process. Prior to MySQL 8.0.18, each statement must be on a single line and should not include comments. As of MySQL 8.0.18, the acceptable format for statements in the file is expanded to support these constructs:

- `delimiter ;`, to set the statement delimiter to the `;` character.
- `delimiter $$`, to set the statement delimiter to the `$$` character sequence.
- Multiple statements on the same line, delimited by the current delimiter.
- Multiple-line statements.
- Comments from a `#` character to the end of the line.
- Comments from a `--` sequence to the end of the line.
- C-style comments from a `/*` sequence to the following `*/` sequence, including over multiple lines.
- Multiple-line string literals enclosed within either single quote (`'`) or double quote (`"`) characters.

If the server is started with the `--initialize` or `--initialize-insecure` option, it operates in bootstap mode and some functionality is unavailable that limits the statements permitted in the file. These include statements that relate to account management (such as `CREATE USER` or `GRANT`), replication, and global transaction identifiers. See Section 17.1.3, “Replication with Global Transaction Identifiers”.

- `innodb_xxx`

`InnoDB` system variables are listed in Section 15.13, “InnoDB Startup Options and System Variables”. These variables control many aspects of storage, memory use, and I/O patterns for `InnoDB` tables, and are especially important now that `InnoDB` is the default storage engine.

- `insert_id`

The value to be used by the following `INSERT` or `ALTER TABLE` statement when inserting an `AUTO_INCREMENT` value. This is mainly used with the binary log.

- `interactive_timeout`

Property	Value
Command-Line Format	`--interactive-timeout=#`
System Variable	`interactive_timeout`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`28800`
Minimum Value	`1`

The number of seconds the server waits for activity on an interactive connection before closing it. An interactive client is defined as a client that uses the `CLIENT_INTERACTIVE` option to `mysql_real_connect()`. See also `wait_timeout`.

- `internal_tmp_disk_storage_engine`

Property	Value
Command-Line Format	`--internal-tmp-disk-storage-engine=#`
Removed	8.0.16
System Variable	`internal_tmp_disk_storage_engine`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`INNODB`
Valid Values	`MYISAM` `INNODB`

Important

In MySQL 8.0.16 and later, on-disk internal temporary tables always use the `InnoDB` storage engine; as of MySQL 8.0.16, this variable has been removed and is thus no longer supported.

Prior to MySQL 8.0.16, this variable determines the storage engine used for on-disk internal temporary tables (see Storage Engine for On-Disk Internal Temporary Tables). Permitted values are `MYISAM` and `INNODB` (the default).

- `internal_tmp_mem_storage_engine`

Property	Value
Command-Line Format	`--internal-tmp-mem-storage-engine=#`
Introduced	8.0.2
System Variable	`internal_tmp_mem_storage_engine`

Property	Value
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Enumeration
Default Value	`TempTable`

Valid Values	`TempTable`
	
	`MEMORY`

The storage engine for in-memory internal temporary tables (see Section 8.4.4, “Internal Temporary Table Use in MySQL”). Permitted values are `TempTable` (the default) and `MEMORY`.

The optimizer uses the storage engine defined by `internal_tmp_mem_storage_engine` for in-memory internal temporary tables.

- `join_buffer_size`

Property	Value
Command-Line Format	`--join-buffer-size=#`
System Variable	`join_buffer_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`262144`
Minimum Value	`128`
Maximum Value (Other, 64-bit platforms)	`18446744073709547520`
Maximum Value (Other, 32-bit platforms)	`4294967295`
Maximum Value (Windows)	`4294967295`

The minimum size of the buffer that is used for plain index scans, range index scans, and joins that do not use indexes and thus perform full table scans. Normally, the best way to get fast joins is to add indexes. Increase the value of `join_buffer_size` to get a faster full join when adding indexes is not possible. One join buffer is allocated for each full join between two tables. For a complex join between several tables for which indexes are not used, multiple join buffers might be necessary.

Unless a Block Nested-Loop or Batched Key Access algorithm is used, there is no gain from setting the buffer larger than required to hold each matching row, and all joins allocate at least the minimum size, so use caution in setting this variable to a large value globally. It is better to keep the global setting small and change the session setting to a larger value only in sessions that are doing large joins, or change the setting on a per-query basis by using a `SET_VAR` optimizer hint (see Section 8.9.3, “Optimizer Hints”). Memory allocation time can cause substantial performance drops if the global size is larger than needed by most queries that use it.

When Block Nested-Loop is used, a larger join buffer can be beneficial up to the point where all required columns from all rows in the first table are stored in the join buffer. This depends on the query; the optimal size may be smaller than holding all rows from the first tables.

When Batched Key Access is used, the value of `join_buffer_size` defines how large the batch of keys is in each request to the storage engine. The larger the buffer, the more sequential access will be to the right hand table of a join operation, which can significantly improve performance.

The default is 256KB. The maximum permissible setting for `join_buffer_size` is 4GB−1. Larger values are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB−1 with a warning).

For additional information about join buffering, see Section 8.2.1.7, “Nested-Loop Join Algorithms”. For information about Batched Key Access, see Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”.

- `keep_files_on_create`

Property	Value
Command-Line Format	`--keep-files-on-create[={OFF|ON}]`
System Variable	`keep_files_on_create`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

If a `MyISAM` table is created with no `DATA DIRECTORY` option, the `.MYD` file is created in the database directory. By default, if `MyISAM` finds an existing `.MYD` file in this case, it overwrites it. The same applies to `.MYI` files for tables created with no `INDEX DIRECTORY` option. To suppress this behavior, set the `keep_files_on_create` variable to `ON` (1), in which case `MyISAM` will not overwrite existing files and returns an error instead. The default value is `OFF` (0).

If a `MyISAM` table is created with a `DATA DIRECTORY` or `INDEX DIRECTORY` option and an existing `.MYD` or `.MYI` file is found, MyISAM always returns an error. It will not overwrite a file in the specified directory.

- `key_buffer_size`

Property	Value
Command-Line Format	`--key-buffer-size=#`
System Variable	`key_buffer_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`8388608`
Minimum Value	`8`

Property	Value
Maximum Value (64-bit platforms)	`OS_PER_PROCESS_LIMIT`
Maximum Value (32-bit platforms)	`4294967295`

Index blocks for `MyISAM` tables are buffered and are shared by all threads. `key_buffer_size` is the size of the buffer used for index blocks. The key buffer is also known as the key cache.

The maximum permissible setting for `key_buffer_size` is 4GB−1 on 32-bit platforms. Larger values are permitted for 64-bit platforms. The effective maximum size might be less, depending on your available physical RAM and per-process RAM limits imposed by your operating system or hardware platform. The value of this variable indicates the amount of memory requested. Internally, the server allocates as much memory as possible up to this amount, but the actual allocation might be less.

You can increase the value to get better index handling for all reads and multiple writes; on a system whose primary function is to run MySQL using the `MyISAM` storage engine, 25% of the machine's total memory is an acceptable value for this variable. However, you should be aware that, if you make the value too large (for example, more than 50% of the machine's total memory), your system might start to page and become extremely slow. This is because MySQL relies on the operating system to perform file system caching for data reads, so you must leave some room for the file system cache. You should also consider the memory requirements of any other storage engines that you may be using in addition to `MyISAM`.

For even more speed when writing many rows at the same time, use `LOCK TABLES`. See Section 8.2.5.1, “Optimizing INSERT Statements”.

You can check the performance of the key buffer by issuing a `SHOW STATUS` statement and examining the `Key_read_requests`, `Key_reads`, `Key_write_requests`, and `Key_writes` status variables. (See Section 13.7.7, “SHOW Syntax”.) The `Key_reads/Key_read_requests` ratio should normally be less than 0.01. The `Key_writes/Key_write_requests` ratio is usually near 1 if you are using mostly updates and deletes, but might be much smaller if you tend to do updates that affect many rows at the same time or if you are using the `DELAY_KEY_WRITE` table option.

The fraction of the key buffer in use can be determined using `key_buffer_size` in conjunction with the `Key_blocks_unused` status variable and the buffer block size, which is available from the `key_cache_block_size` system variable:

```
1 - ((Key_blocks_unused * key_cache_block_size) / key_buffer_size)
```

This value is an approximation because some space in the key buffer is allocated internally for administrative structures. Factors that influence the amount of overhead for these structures include block size and pointer size. As block size increases, the percentage of the key buffer lost to overhead tends to decrease. Larger blocks results in a smaller number of read operations (because more keys are obtained per read), but conversely an increase in reads of keys that are not examined (if not all keys in a block are relevant to a query).

It is possible to create multiple `MyISAM` key caches. The size limit of 4GB applies to each cache individually, not as a group. See Section 8.10.2, “The MyISAM Key Cache”.

- `key_cache_age_threshold`

Property	Value
Command-Line Format	`--key-cache-age-threshold=#`
System Variable	`key_cache_age_threshold`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`300`
Minimum Value	`100`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

This value controls the demotion of buffers from the hot sublist of a key cache to the warm sublist. Lower values cause demotion to happen more quickly. The minimum value is 100. The default value is 300. See Section 8.10.2, “The MyISAM Key Cache”.

- `key_cache_block_size`

Property	Value
Command-Line Format	`--key-cache-block-size=#`
System Variable	`key_cache_block_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1024`
Minimum Value	`512`
Maximum Value	`16384`

The size in bytes of blocks in the key cache. The default value is 1024. See Section 8.10.2, “The MyISAM Key Cache”.

- `key_cache_division_limit`

Property	Value
Command-Line Format	`--key-cache-division-limit=#`
System Variable	`key_cache_division_limit`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value	`100`
Minimum Value	`1`
Maximum Value	`100`

The division point between the hot and warm sublists of the key cache buffer list. The value is the percentage of the buffer list to use for the warm sublist. Permissible values range from 1 to 100. The default value is 100. See Section 8.10.2, “The MyISAM Key Cache”.

- `large_files_support`

Property	Value
System Variable	`large_files_support`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

Whether `mysqld` was compiled with options for large file support.

- `large_pages`

Property	Value
Command-Line Format	`--large-pages[={OFF|ON}]`
System Variable	`large_pages`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Platform Specific	Linux
Type	Boolean
Default Value	`OFF`

Whether large page support is enabled (via the `--large-pages` option). See Section 8.12.3.2, “Enabling Large Page Support”.

- `large_page_size`

Property	Value
System Variable	`large_page_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`

If large page support is enabled, this shows the size of memory pages. Large memory pages are supported only on Linux; on other platforms, the value of this variable is always 0. See Section 8.12.3.2, “Enabling Large Page Support”.

- `last_insert_id`

  The value to be returned from `LAST_INSERT_ID()`. This is stored in the binary log when you use `LAST_INSERT_ID()` in a statement that updates a table. Setting this variable does not update the value returned by the `mysql_insert_id()` C API function.

- `lc_messages`

Property	Value
Command-Line Format	`--lc-messages=name`
System Variable	`lc_messages`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`en_US`

  The locale to use for error messages. The default is `en_US`. The server converts the argument to a language name and combines it with the value of `lc_messages_dir` to produce the location for the error message file. See Section 10.12, “Setting the Error Message Language”.

- `lc_messages_dir`

Property	Value
Command-Line Format	`--lc-messages-dir=dir_name`
System Variable	`lc_messages_dir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

  The directory where error messages are located. The server uses the value together with the value of `lc_messages` to produce the location for the error message file. See Section 10.12, “Setting the Error Message Language”.

- `lc_time_names`

Property	Value
Command-Line Format	`--lc-time-names=value`
System Variable	`lc_time_names`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	String

This variable specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()` and `MONTHNAME()` functions. Locale names are POSIX-style values such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting. For further information, see Section 10.16, “MySQL Server Locale Support”.

- `license`

Property	Value
System Variable	`license`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`GPL`

The type of license the server has.

- `local_infile`

Property	Value
Command-Line Format	`--local-infile[={OFF|ON}]`
System Variable	`local_infile`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value (>= 8.0.2)	`OFF`
Default Value (<= 8.0.1)	`ON`

This variable controls server-side `LOCAL` capability for `LOAD DATA` statements. Depending on the `local_infile` setting, the server refuses or permits local data loading by clients that have `LOCAL` enabled on the client side.

To explicitly cause the server to refuse or permit `LOAD DATA LOCAL` statements (regardless of how client programs and libraries are configured at build time or runtime), start `mysqld` with `local_infile` disabled or enabled, respectively. `local_infile` can also be set at runtime. For more information, see Section 6.1.6, “Security Issues with LOAD DATA LOCAL”.

- `lock_wait_timeout`

Property	Value
Command-Line Format	`--lock-wait-timeout=#`
System Variable	`lock_wait_timeout`

Property	Value
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	31536000
Minimum Value	1
Maximum Value	31536000

This variable specifies the timeout in seconds for attempts to acquire metadata locks. The permissible values range from 1 to 31536000 (1 year). The default is 31536000.

This timeout applies to all statements that use metadata locks. These include DML and DDL operations on tables, views, stored procedures, and stored functions, as well as `LOCK TABLES`, `FLUSH TABLES WITH READ LOCK`, and `HANDLER` statements.

This timeout does not apply to implicit accesses to system tables in the `mysql` database, such as grant tables modified by `GRANT` or `REVOKE` statements or table logging statements. The timeout does apply to system tables accessed directly, such as with `SELECT` or `UPDATE`.

The timeout value applies separately for each metadata lock attempt. A given statement can require more than one lock, so it is possible for the statement to block for longer than the `lock_wait_timeout` value before reporting a timeout error. When lock timeout occurs, `ER_LOCK_WAIT_TIMEOUT` is reported.

`lock_wait_timeout` also defines the amount of time that a `LOCK INSTANCE FOR BACKUP` statement waits for a lock before giving up.

- `locked_in_memory`

Property	Value
System Variable	`locked_in_memory`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

Whether `mysqld` was locked in memory with `--memlock`.

- `log_error`

Property	Value
Command-Line Format	`--log-error[=file_name]`
System Variable	`log_error`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

The default error log destination. If the destination is the console, the value is `stderr`. Otherwise, the destination is a file and the `log_error` value is the file name. See Section 5.4.2, “The Error Log”.

- `log_error_filter_rules`

Property	Value
Command-Line Format	`--log-error-filter-rules=value`
Introduced	8.0.2
Removed	8.0.4
System Variable	`log_error_filter_rules`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`set by server`

The filter rules for error logging. This variable is unused. It was removed in MySQL 8.0.4.

- `log_error_services`

Property	Value
Command-Line Format	`--log-error-services=value`
Introduced	8.0.2
System Variable	`log_error_services`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`log_filter_internal; log_sink_internal`

The components to enable for error logging. The variable may contain a list with 0, 1, or many elements. In the latter case, elements may be delimited by semicolon or (as of MySQL 8.0.12) comma, optionally followed by space. A given setting cannot use both semicolon and comma separators. Component order is significant because the server executes components in the order listed. Any loadable (not built in) component named in the `log_error_services` value must first be installed with `INSTALL COMPONENT`. For more information, see Section 5.4.2.1, “Error Log Component Configuration”.

- `log_error_suppression_list`

Property	Value
Command-Line Format	`--log-error-suppression-list=value`
Introduced	8.0.13
System Variable	`log_error_suppression_list`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	String
Default Value	`empty string`

The `log_error_suppression_list` system variable applies to events intended for the error log and specifies which events to suppress when they occur with a priority of `WARNING` or `INFORMATION`. For example, if a particular type of warning is considered undesirable “noise” in the error log because it occurs frequently but is not of interest, it can be suppressed. This variable affects filtering performed by the `log_filter_internal` error log filter component, which is enabled by default (see Section 5.5.3, “Error Log Components”). If `log_filter_internal` is disabled, `log_error_suppression_list` has no effect.

The `log_error_suppression_list` value may be the empty string for no suppression, or a list of one or more comma-separated values indicating the error codes to suppress. Error codes may be specified in symbolic or numeric form. A numeric code may be specified with or without the `MY-` prefix. Leading zeros in the numeric part are not significant. Examples of permitted code formats:

```
ER_SERVER_SHUTDOWN_COMPLETE
MY-000031
000031
MY-31
31
```

Symbolic values are preferable to numeric values for readability and portability. For information about the permitted error symbols and numbers, see Section B.3, “Error Message Reference”.

The effect of `log_error_suppression_list` combines with that of `log_error_verbosity`. For additional information, see Section 5.4.2.4, “Priority-Based Error Log Filtering (log_filter_internal)”.

- `log_error_verbosity`

Property	Value
Command-Line Format	`--log-error-verbosity=#`
System Variable	`log_error_verbosity`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.4)	2
Default Value (<= 8.0.3)	3
Minimum Value	1
Maximum Value	3

The `log_error_verbosity` system variable specifies the verbosity for handling events intended for the error log. This variable affects filtering performed by the `log_filter_internal` error log filter component, which is enabled by default (see Section 5.5.3, “Error Log Components”). If `log_filter_internal` is disabled, `log_error_verbosity` has no effect.

Events intended for the error log have a priority of `ERROR`, `WARNING`, or `INFORMATION`. `log_error_verbosity` controls verbosity based on which priorities to permit for messages written to the log, as shown in the following table.

Permitted Message Priorities	log_error_verbosity Value
`ERROR`	1
`ERROR`, `WARNING`	2
`ERROR`, `WARNING`, `INFORMATION`	3

There is also a priority of `SYSTEM`. System messages about non-error situations are printed to the error log regardless of the `log_error_verbosity` value. These messages include startup and shutdown messages, and some significant changes to settings.

The effect of `log_error_verbosity` combines with that of `log_error_suppression_list`. For additional information, see Section 5.4.2.4, “Priority-Based Error Log Filtering (log_filter_internal)”.

- `log_output`

Property	Value
Command-Line Format	`--log-output=name`
System Variable	`log_output`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Set
Default Value	`FILE`

Valid Values	`TABLE`
	`FILE`
	`NONE`

The destination or destinations for general query log and slow query log output. The value is a list one or more comma-separated words chosen from `TABLE`, `FILE`, and `NONE`. `TABLE` selects logging to the `general_log` and `slow_log` tables in the `mysql` system database. `FILE` selects logging to log files. `NONE` disables logging. If `NONE` is present in the value, it takes precedence over any other words that are present. `TABLE` and `FILE` can both be given to select both log output destinations.

This variable selects log output destinations, but does not enable log output. To do that, enable the `general_log` and `slow_query_log` system variables. For `FILE` logging, the `general_log_file` and `slow_query_log_file` system variables determine the log file locations. For more information, see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”.

- `log_queries_not_using_indexes`

Property	Value
Command-Line Format	`--log-queries-not-using-indexes[={OFF|ON}]`
System Variable	`log_queries_not_using_indexes`

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

If you enable this variable with the slow query log enabled, queries that are expected to retrieve all rows are logged. See Section 5.4.5, “The Slow Query Log”. This option does not necessarily mean that no index is used. For example, a query that uses a full index scan uses an index but would be logged because the index would not limit the number of rows.

- log_slow_admin_statements

Property	Value
Command-Line Format	--log-slow-admin-statements[={OFF|ON}]
System Variable	log_slow_admin_statements
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Include slow administrative statements in the statements written to the slow query log. Administrative statements include ALTER TABLE, ANALYZE TABLE, CHECK TABLE, CREATE INDEX, DROP INDEX, OPTIMIZE TABLE, and REPAIR TABLE.

- log_slow_extra

Property	Value
Command-Line Format	--log-slow-extra[={OFF|ON}]
Introduced	8.0.14
System Variable	log_slow_extra
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

If the slow query log is enabled and the output destination includes FILE, the server writes additional fields to log file lines that provide information about slow statements. See Section 5.4.5, “The Slow Query Log”. TABLE output is unaffected.

- log_syslog

Property	Value
Command-Line Format	`--log-syslog[={OFF|ON}]`
Deprecated	8.0.2 (removed in 8.0.13)
System Variable	`log_syslog`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value (Windows, <= 8.0.1)	`ON`
Default Value (Unix, <= 8.0.1)	`OFF`
Default Value (>= 8.0.2)	`ON` (when error logging to system log is enabled)

Prior to MySQL 8.0, this variable controlled whether to perform error logging to the system log (the Event Log on Windows, and `syslog` on Unix and Unix-like systems).

In MySQL 8.0, the `log_sink_syseventlog` log component implements error logging to the system log (see Section 5.4.2.7, “Error Logging to the System Log”), so this type of logging can be enabled by adding that component to the `log_error_services` system variable. `log_syslog` is removed. (Prior to MySQL 8.0.13, `log_syslog` exists but is deprecated and has no effect.)

- `log_syslog_facility`

Property	Value
Command-Line Format	`--log-syslog-facility=value`
Removed	8.0.13
System Variable	`log_syslog_facility`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`daemon`

This variable was removed in MySQL 8.0.13 and replaced by `syseventlog.facility`.

- `log_syslog_include_pid`

Property	Value
Command-Line Format	`--log-syslog-include-pid[={OFF|ON}]`
Removed	8.0.13
System Variable	`log_syslog_include_pid`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

This variable was removed in MySQL 8.0.13 and replaced by syseventlog.include_pid.

- log_syslog_tag

Property	Value
Command-Line Format	--log-syslog-tag=tag
Removed	8.0.13
System Variable	log_syslog_tag
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	empty string

This variable was removed in MySQL 8.0.13 and replaced by syseventlog.tag.

- log_timestamps

Property	Value
Command-Line Format	--log-timestamps=#
System Variable	log_timestamps
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	UTC
Valid Values	UTC SYSTEM

This variable controls the time zone of timestamps in messages written to the error log, and in general query log and slow query log messages written to files. It does not affect the time zone of general query log and slow query log messages written to tables (mysql.general_log, mysql.slow_log). Rows retrieved from those tables can be converted from the local system time zone to any desired time zone with CONVERT_TZ() or by setting the session time_zone system variable.

Permitted log_timestamps values are UTC (the default) and SYSTEM (local system time zone).

Timestamps are written using ISO 8601 / RFC 3339 format: YYYY-MM-DDThh:mm:ss.uuuuuu plus a tail value of Z signifying Zulu time (UTC) or ±hh:mm (an offset from UTC).

- log_throttle_queries_not_using_indexes

Property	Value
Command-Line Format	`--log-throttle-queries-not-using-indexes=#`
System Variable	`log_throttle_queries_not_using_indexes`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`

If `log_queries_not_using_indexes` is enabled, the `log_throttle_queries_not_using_indexes` variable limits the number of such queries per minute that can be written to the slow query log. A value of 0 (the default) means “no limit”. For more information, see Section 5.4.5, “The Slow Query Log”.

- `log_warnings`

Property	Value
Command-Line Format	`--log-warnings[=#]`
Deprecated	Yes (removed in 8.0.3)
System Variable	`log_warnings`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`2`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

This system variable was removed in MySQL 8.0.3. Use the `log_error_verbosity` system variable instead.

- `long_query_time`

Property	Value
Command-Line Format	`--long-query-time=#`
System Variable	`long_query_time`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Numeric
Default Value	`10`

Property	Value
Minimum Value	`0`

If a query takes longer than this many seconds, the server increments the `Slow_queries` status variable. If the slow query log is enabled, the query is logged to the slow query log file. This value is measured in real time, not CPU time, so a query that is under the threshold on a lightly loaded system might be above the threshold on a heavily loaded one. The minimum and default values of `long_query_time` are 0 and 10, respectively. The value can be specified to a resolution of microseconds. See Section 5.4.5, “The Slow Query Log”.

- `low_priority_updates`

Property	Value
Command-Line Format	`--low-priority-updates[={OFF|ON}]`
System Variable	`low_priority_updates`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

If set to `1`, all `INSERT`, `UPDATE`, `DELETE`, and `LOCK TABLE WRITE` statements wait until there is no pending `SELECT` or `LOCK TABLE READ` on the affected table. The same effect can be obtained using `{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...` to lower the priority of only one query. This variable affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`). See Section 8.11.2, “Table Locking Issues”.

- `lower_case_file_system`

Property	Value
System Variable	`lower_case_file_system`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean

This variable describes the case sensitivity of file names on the file system where the data directory is located. `OFF` means file names are case-sensitive, `ON` means they are not case-sensitive. This variable is read only because it reflects a file system attribute and setting it would have no effect on the file system.

- `lower_case_table_names`

Property	Value
Command-Line Format	`--lower-case-table-names[=#]`
System Variable	`lower_case_table_names`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2`

If set to 0, table names are stored as specified and comparisons are case-sensitive. If set to 1, table names are stored in lowercase on disk and comparisons are not case sensitive. If set to 2, table names are stored as given but compared in lowercase. This option also applies to database names and table aliases. For additional details, see Section 9.2.3, “Identifier Case Sensitivity”.

On Windows the default value is 1. On macOS, the default value is 2. On Linux, a value of 2 is not supported; the server forces the value to 0 instead.

You should *not* set `lower_case_table_names` to 0 if you are running MySQL on a system where the data directory resides on a case-insensitive file system (such as on Windows or macOS). It is an unsupported combination that could result in a hang condition when running an `INSERT INTO ... SELECT ... FROM` *`tbl_name`* operation with the wrong *`tbl_name`* letter case. With `MyISAM`, accessing table names using different letter cases could cause index corruption.

An error message is printed and the server exits if you attempt to start the server with `--lower_case_table_names=0` on a case-insensitive file system.

If you are using `InnoDB` tables, you should set this variable to 1 on all platforms to force names to be converted to lowercase.

The setting of this variable affects the behavior of replication filtering options with regard to case sensitivity. For more information, see Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”.

It is prohibited to start the server with a `lower_case_table_names` setting that is different from the setting used when the server was initialized. The restriction is necessary because collations used by various data dictionary table fields are determined by the setting defined when the server is initialized, and restarting the server with a different setting would introduce inconsistencies with respect to how identifiers are ordered and compared.

It is therefore necessary to configure `lower_case_table_names` to the desired setting before initializing the server. In most cases, this requires configuring `lower_case_table_names` in a MySQL option file before starting the MySQL server for the first time. For APT installations on Debian and Ubuntu, however, the server is initialized for you, and there is no opportunity to configure the setting in an option file beforehand. You must therefore use the `debconf-set-selection` utility prior to installing MySQL using APT to enable `lower_case_table_names`. To do so, run this command before installing MySQL using APT:

```
shell> sudo debconf-set-selections <<< "mysql-server mysql-server/lowercase-table-names select Enabled
```

Note

The ability to enable `lower_case_table_names` using `debconf-set-selections` was added in MySQL 8.0.17. Enabling `lower_case_table_names` sets the value to 1.

- `mandatory_roles`

Property	Value
Command-Line Format	`--mandatory-roles=value`
Introduced	8.0.2
System Variable	`mandatory_roles`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`empty string`

Roles the server should treat as mandatory. In effect, these roles are automatically granted to every user, although setting `mandatory_roles` does not actually change any user accounts, and the granted roles are not visible in the `mysql.role_edges` system table.

The variable value is a comma-separated list of role names. Example:

```
SET PERSIST mandatory_roles = '`role1`@`%`,`role2`,role3,role4@localhost';
```

Setting the runtime value of `mandatory_roles` requires the `ROLE_ADMIN` privilege, in addition to the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege normally required to set a global system variable runtime value.

Role names consist of a user part and host part in *`user_name@host_name`* format. The host part, if omitted, defaults to `%`. For additional information, see Section 6.2.5, “Specifying Role Names”.

The `mandatory_roles` value is a string, so user names and host names, if quoted, must be written in a fashion permitted for quoting within quoted strings.

Roles named in the value of `mandatory_roles` cannot be revoked with `REVOKE` or dropped with `DROP ROLE` or `DROP USER`.

To prevent sessions from being made system sessions by default, a role that has the `SYSTEM_USER` privilege cannot be listed in the value of the `mandatory_roles` system variable:

- If `mandatory_roles` is assigned a role at startup that has the `SYSTEM_USER` privilege, the server writes a message to the error log and exits.
- If `mandatory_roles` is assigned a role at runtime that has the `SYSTEM_USER` privilege, an error occurs and the `mandatory_roles` value remains unchanged.

Mandatory roles, like explicitly granted roles, do not take effect until activated (see Activating Roles). At login time, role activation occurs for all granted roles if the `activate_all_roles_on_login` system variable is enabled; otherwise, or for roles that are set as default roles otherwise. At runtime, `SET ROLE` activates roles.

Roles that do not exist when assigned to `mandatory_roles` but are created later may require special treatment to be considered mandatory. For details, see Defining Mandatory Roles.

`SHOW GRANTS` displays mandatory roles according to the rules described in Section 13.7.7.21, “SHOW GRANTS Syntax”.

- `max_allowed_packet`

Property	Value
Command-Line Format	`--max-allowed-packet=#`
System Variable	`max_allowed_packet`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.3)	`67108864`
Default Value (<= 8.0.2)	`4194304`
Minimum Value	`1024`
Maximum Value	`1073741824`

The maximum size of one packet or any generated/intermediate string, or any parameter sent by the `mysql_stmt_send_long_data()` C API function. The default is 64MB.

The packet message buffer is initialized to `net_buffer_length` bytes, but can grow up to `max_allowed_packet` bytes when needed. This value by default is small, to catch large (possibly incorrect) packets.

You must increase this value if you are using large `BLOB` columns or long strings. It should be as big as the largest `BLOB` you want to use. The protocol limit for `max_allowed_packet` is 1GB. The value should be a multiple of 1024; nonmultiples are rounded down to the nearest multiple.

When you change the message buffer size by changing the value of the `max_allowed_packet` variable, you should also change the buffer size on the client side if your client program permits it. The default `max_allowed_packet` value built in to the client library is 1GB, but individual client programs might override this. For example, `mysql` and `mysqldump` have defaults of 16MB and 24MB, respectively. They also enable you to change the client-side value by setting `max_allowed_packet` on the command line or in an option file.

The session value of this variable is read only. The client can receive up to as many bytes as the session value. However, the server will not send to the client more bytes than the current global `max_allowed_packet` value. (The global value could be less than the session value if the global value is changed after the client connects.)

- `max_connect_errors`

Property	Value
Command-Line Format	`--max-connect-errors=#`
System Variable	`max_connect_errors`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`100`
Minimum Value	`1`

Property	Value
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

After `max_connect_errors` successive connection requests from a host are interrupted without a successful connection, the server blocks that host from further connections. If a connection from a host is established successfully within fewer than `max_connect_errors` attempts after a previous connection was interrupted, the error count for the host is cleared to zero. However, once a host is blocked, flushing the host cache is the only way to unblock it. To flush the host cache, execute a `FLUSH HOSTS` statement, a `TRUNCATE TABLE` statement that truncates the Performance Schema `host_cache` table, or a `mysqladmin flush-hosts` command.

For more information about how the host cache works, see Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

- `max_connections`

Property	Value
Command-Line Format	`--max-connections=#`
System Variable	`max_connections`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	151
Minimum Value	1
Maximum Value	100000

The maximum permitted number of simultaneous client connections. For more information, see Section 8.12.4.1, “How MySQL Handles Client Connections”.

- `max_delayed_threads`

Property	Value
Command-Line Format	`--max-delayed-threads=#`
Deprecated	Yes
System Variable	`max_delayed_threads`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	20
Minimum Value	0
Maximum Value	16384

This system variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `max_digest_length`

Property	Value
Command-Line Format	`--max-digest-length=#`
System Variable	`max_digest_length`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1024`
Minimum Value	`0`
Maximum Value	`1048576`

The maximum number of bytes of memory reserved per session for computation of normalized statement digests. Once that amount of space is used during digest computation, truncation occurs: no further tokens from a parsed statement are collected or figure into its digest value. Statements that differ only after that many bytes of parsed tokens produce the same normalized statement digest and are considered identical if compared or if aggregated for digest statistics.

Decreasing the `max_digest_length` value reduces memory use but causes the digest value of more statements to become indistinguishable if they differ only at the end. Increasing the value permits longer statements to be distinguished but increases memory use, particularly for workloads that involve large numbers of simultaneous sessions (the server allocates `max_digest_length` bytes per session).

The parser uses this system variable as a limit on the maximum length of normalized statement digests that it computes. The Performance Schema, if it tracks statement digests, makes a copy of the digest value, using the `performance_schema_max_digest_length`. system variable as a limit on the maximum length of digests that it stores. Consequently, if `performance_schema_max_digest_length` is less than `max_digest_length`, digest values stored in the Performance Schema are truncated relative to the original digest values.

For more information about statement digesting, see Section 26.10, “Performance Schema Statement Digests and Sampling”.

- `max_error_count`

Property	Value
Command-Line Format	`--max-error-count=#`
System Variable	`max_error_count`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value (>= 8.0.3)	`1024`

Property	Value
Default Value (<= 8.0.2)	`64`
Minimum Value	`0`
Maximum Value	`65535`

The maximum number of error, warning, and information messages to be stored for display by the `SHOW ERRORS` and `SHOW WARNINGS` statements. This is the same as the number of condition areas in the diagnostics area, and thus the number of conditions that can be inspected by `GET DIAGNOSTICS`.

- `max_execution_time`

Property	Value
Command-Line Format	`--max-execution-time=#`
System Variable	`max_execution_time`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`0`

The execution timeout for `SELECT` statements, in milliseconds. If the value is 0, timeouts are not enabled.

`max_execution_time` applies as follows:

- The global `max_execution_time` value provides the default for the session value for new connections. The session value applies to `SELECT` executions executed within the session that include no `MAX_EXECUTION_TIME(N)` optimizer hint or for which *`N`* is 0.
- `max_execution_time` applies to read-only `SELECT` statements. Statements that are not read only are those that invoke a stored function that modifies data as a side effect.
- `max_execution_time` is ignored for `SELECT` statements in stored programs.

- `max_heap_table_size`

Property	Value
Command-Line Format	`--max-heap-table-size=#`
System Variable	`max_heap_table_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`16777216`
Minimum Value	`16384`
Maximum Value (64-bit platforms)	`18446744073709551615`

Property	Value
Maximum Value (32-bit platforms)	`4294967295`

This variable sets the maximum size to which user-created `MEMORY` tables are permitted to grow. The value of the variable is used to calculate `MEMORY` table `MAX_ROWS` values. Setting this variable has no effect on any existing `MEMORY` table, unless the table is re-created with a statement such as `CREATE TABLE` or altered with `ALTER TABLE` or `TRUNCATE TABLE`. A server restart also sets the maximum size of existing `MEMORY` tables to the global `max_heap_table_size` value.

This variable is also used in conjunction with `tmp_table_size` to limit the size of internal in-memory tables. See Section 8.4.4, “Internal Temporary Table Use in MySQL”.

`max_heap_table_size` is not replicated. See Section 17.5.1.21, “Replication and MEMORY Tables”, and Section 17.5.1.38, “Replication and Variables”, for more information.

- `max_insert_delayed_threads`

Property	Value
Deprecated	Yes
System Variable	`max_insert_delayed_threads`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

This variable is a synonym for `max_delayed_threads`.

This system variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `max_join_size`

Property	Value
Command-Line Format	`--max-join-size=#`
System Variable	`max_join_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`18446744073709551615`
Minimum Value	`1`
Maximum Value	`18446744073709551615`

Do not permit statements that probably need to examine more than `max_join_size` rows (for single-table statements) or row combinations (for multiple-table statements) or that are likely to do more than `max_join_size` disk seeks. By setting this value, you can catch statements where keys are not used properly and that would probably take a long time. Set it if your users tend to perform joins that lack a

`WHERE` clause, that take a long time, or that return millions of rows. For more information, see Using Safe-Updates Mode (--safe-updates).

Setting this variable to a value other than `DEFAULT` resets the value of `sql_big_selects` to `0`. If you set the `sql_big_selects` value again, the `max_join_size` variable is ignored.

- `max_length_for_sort_data`

Property	Value
Command-Line Format	`--max-length-for-sort-data=#`
System Variable	`max_length_for_sort_data`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value (>= 8.0.1)	`4096`
Default Value (8.0.0)	`1024`
Minimum Value	`4`
Maximum Value	`8388608`

The cutoff on the size of index values that determines which `filesort` algorithm to use. See Section 8.2.1.16, “ORDER BY Optimization”.

- `max_points_in_geometry`

Property	Value
Command-Line Format	`--max-points-in-geometry=#`
System Variable	`max_points_in_geometry`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`65536`
Minimum Value	`3`
Maximum Value	`1048576`

The maximum value of the *`points_per_circle`* argument to the `ST_Buffer_Strategy()` function.

- `max_prepared_stmt_count`

Property	Value
Command-Line Format	`--max-prepared-stmt-count=#`
System Variable	`max_prepared_stmt_count`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	16382
Minimum Value	0
Maximum Value (>= 8.0.18)	4194304
Maximum Value (<= 8.0.17)	1048576

This variable limits the total number of prepared statements in the server. It can be used in environments where there is the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. If the value is set lower than the current number of prepared statements, existing statements are not affected and can be used, but no new statements can be prepared until the current number drops below the limit. Setting the value to 0 disables prepared statements.

- max_seeks_for_key

Property	Value
Command-Line Format	--max-seeks-for-key=#
System Variable	max_seeks_for_key
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value (64-bit platforms)	18446744073709551615
Default Value (32-bit platforms)	4294967295
Minimum Value	1
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

Limit the assumed maximum number of seeks when looking up rows based on a key. The MySQL optimizer assumes that no more than this number of key seeks are required when searching for matching rows in a table by scanning an index, regardless of the actual cardinality of the index (see Section 13.7.7.22, “SHOW INDEX Syntax”). By setting this to a low value (say, 100), you can force MySQL to prefer indexes instead of table scans.

- max_sort_length

Property	Value
Command-Line Format	--max-sort-length=#
System Variable	max_sort_length
Scope	Global, Session

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	1024
Minimum Value	4
Maximum Value	8388608

The number of bytes to use when sorting data values. The server uses only the first `max_sort_length` bytes of each value and ignores the rest. Consequently, values that differ only after the first `max_sort_length` bytes compare as equal for `GROUP BY`, `ORDER BY`, and `DISTINCT` operations.

Increasing the value of `max_sort_length` may require increasing the value of `sort_buffer_size` as well. For details, see Section 8.2.1.16, “ORDER BY Optimization”

- `max_sp_recursion_depth`

Property	Value
Command-Line Format	`--max-sp-recursion-depth[=#]`
System Variable	`max_sp_recursion_depth`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Maximum Value	255

The number of times that any given stored procedure may be called recursively. The default value for this option is 0, which completely disables recursion in stored procedures. The maximum value is 255.

Stored procedure recursion increases the demand on thread stack space. If you increase the value of `max_sp_recursion_depth`, it may be necessary to increase thread stack size by increasing the value of `thread_stack` at server startup.

- `max_tmp_tables`

This system variable was removed in MySQL 8.0.3.

- `max_user_connections`

Property	Value
Command-Line Format	`--max-user-connections=#`
System Variable	`max_user_connections`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

The maximum number of simultaneous connections permitted to any given MySQL user account. A value of 0 (the default) means “no limit.”

This variable has a global value that can be set at server startup or runtime. It also has a read-only session value that indicates the effective simultaneous-connection limit that applies to the account associated with the current session. The session value is initialized as follows:

- If the user account has a nonzero `MAX_USER_CONNECTIONS` resource limit, the session `max_user_connections` value is set to that limit.
- Otherwise, the session `max_user_connections` value is set to the global value.

Account resource limits are specified using the `CREATE USER` or `ALTER USER` statement. See Section 6.2.20, “Setting Account Resource Limits”.

- `max_write_lock_count`

Property	Value
Command-Line Format	`--max-write-lock-count=#`
System Variable	`max_write_lock_count`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (64-bit platforms)	`18446744073709551615`
Default Value (32-bit platforms)	`4294967295`
Minimum Value	`1`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

After this many write locks, permit some pending read lock requests to be processed in between. Write lock requests have higher priority than read lock requests. However, if `max_write_lock_count` is set to some low value (say, 10), read lock requests may be preferred over pending write lock requests if the read lock requests have already been passed over in favor of 10 write lock requests. Normally this behavior does not occur because `max_write_lock_count` by default has a very large value.

- `mecab_rc_file`

Property	Value
Command-Line Format	`--mecab-rc-file=file_name`

Property	Value
System Variable	mecab_rc_file
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

The mecab_rc_file option is used when setting up the MeCab full-text parser.

The mecab_rc_file option defines the path to the mecabrc configuration file, which is the configuration file for MeCab. The option is read-only and can only be set at startup. The mecabrc configuration file is required to initialize MeCab.

For information about the MeCab full-text parser, see Section 12.9.9, “MeCab Full-Text Parser Plugin”.

For information about options that can be specified in the MeCab mecabrc configuration file, refer to the MeCab Documentation on the Google Developers site.

- metadata_locks_cache_size

Property	Value
Command-Line Format	--metadata-locks-cache-size=#
Deprecated	Yes (removed in 8.0.13)
System Variable	metadata_locks_cache_size
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	1024
Minimum Value	1
Maximum Value	1048576

This system variable was removed in MySQL 8.0.13.

- metadata_locks_hash_instances

Property	Value
Command-Line Format	--metadata-locks-hash-instances=#
Deprecated	Yes (removed in 8.0.13)
System Variable	metadata_locks_hash_instances
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	8

Property	Value
Minimum Value	1
Maximum Value	1024

This system variable was removed in MySQL 8.0.13.

- `min_examined_row_limit`

Property	Value
Command-Line Format	`--min-examined-row-limit=#`
System Variable	`min_examined_row_limit`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

Queries that examine fewer than this number of rows are not logged to the slow query log.

- `multi_range_count`

Property	Value
Command-Line Format	`--multi-range-count=#`
Deprecated	Yes (removed in 8.0.3)
System Variable	`multi_range_count`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	256
Minimum Value	1
Maximum Value	4294967295

This system variable was removed in MySQL 8.0.3.

- `myisam_data_pointer_size`

Property	Value
Command-Line Format	`--myisam-data-pointer-size=#`
System Variable	`myisam_data_pointer_size`

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	6
Minimum Value	2
Maximum Value	7

The default pointer size in bytes, to be used by `CREATE TABLE` for `MyISAM` tables when no `MAX_ROWS` option is specified. This variable cannot be less than 2 or larger than 7. The default value is 6. See Section B.4.2.11, “The table is full”.

- `myisam_max_sort_file_size`

Property	Value
Command-Line Format	`--myisam-max-sort-file-size=#`
System Variable	`myisam_max_sort_file_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (64-bit platforms)	9223372036854775807
Default Value (32-bit platforms)	2147483648

The maximum size of the temporary file that MySQL is permitted to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

If `MyISAM` index files exceed this size and disk space is available, increasing the value may help performance. The space must be available in the file system containing the directory where the original index file is located.

- `myisam_mmap_size`

Property	Value
Command-Line Format	`--myisam-mmap-size=#`
System Variable	`myisam_mmap_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value (64-bit platforms)	18446744073709551615

Property	Value
Default Value (32-bit platforms)	4294967295
Minimum Value	7
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

The maximum amount of memory to use for memory mapping compressed MyISAM files. If many compressed MyISAM tables are used, the value can be decreased to reduce the likelihood of memory-swapping problems.

- myisam_recover_options

Property	Value
Command-Line Format	--myisam-recover-options[=list]
System Variable	myisam_recover_options
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	OFF
Valid Values	OFF DEFAULT BACKUP FORCE QUICK

Set the MyISAM storage engine recovery mode. The variable value is any combination of the values of OFF, DEFAULT, BACKUP, FORCE, or QUICK. If you specify multiple values, separate them by commas. Specifying the variable with no value at server startup is the same as specifying DEFAULT, and specifying with an explicit value of "" disables recovery (same as a value of OFF). If recovery is enabled, each time mysqld opens a MyISAM table, it checks whether the table is marked as crashed or was not closed properly. (The last option works only if you are running with external locking disabled.) If this is the case, mysqld runs a check on the table. If the table was corrupted, mysqld attempts to repair it.

The following options affect how the repair works.

Option	Description
OFF	No recovery.
DEFAULT	Recovery without backup, forcing, or quick checking.
BACKUP	If the data file was changed during recovery, save a backup of the *tbl_name*.MYD file as *tbl_name-datetime*.BAK.

Option	Description
FORCE	Run recovery even if we would lose more than one row from the .MYD file.
QUICK	Do not check the rows in the table if there are not any delete blocks.

Before the server automatically repairs a table, it writes a note about the repair to the error log. If you want to be able to recover from most problems without user intervention, you should use the options BACKUP,FORCE. This forces a repair of a table even if some rows would be deleted, but it keeps the old data file as a backup so that you can later examine what happened.

See Section 16.2.1, “MyISAM Startup Options”.

- myisam_repair_threads

Property	Value
Command-Line Format	--myisam-repair-threads=#
System Variable	myisam_repair_threads
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1
Minimum Value	1
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

If this value is greater than 1, MyISAM table indexes are created in parallel (each index in its own thread) during the Repair by sorting process. The default value is 1.

Note

Multithreaded repair is still *beta-quality* code.

- myisam_sort_buffer_size

Property	Value
Command-Line Format	--myisam-sort-buffer-size=#
System Variable	myisam_sort_buffer_size
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	8388608
Minimum Value	4096

Property	Value
Maximum Value (Other, 64-bit platforms)	18446744073709551615
Maximum Value (Other, 32-bit platforms)	4294967295
Maximum Value (Windows, 64-bit platforms)	18446744073709551615
Maximum Value (Windows, 32-bit platforms)	4294967295

The size of the buffer that is allocated when sorting MyISAM indexes during a REPAIR TABLE or when creating indexes with CREATE INDEX or ALTER TABLE.

- myisam_stats_method

Property	Value
Command-Line Format	--myisam-stats-method=name
System Variable	myisam_stats_method
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	nulls_unequal
Valid Values	nulls_equal nulls_unequal nulls_ignored

How the server treats NULL values when collecting statistics about the distribution of index values for MyISAM tables. This variable has three possible values, nulls_equal, nulls_unequal, and nulls_ignored. For nulls_equal, all NULL index values are considered equal and form a single value group that has a size equal to the number of NULL values. For nulls_unequal, NULL values are considered unequal, and each NULL forms a distinct value group of size 1. For nulls_ignored, NULL values are ignored.

The method that is used for generating table statistics influences how the optimizer chooses indexes for query execution, as described in Section 8.3.8, “InnoDB and MyISAM Index Statistics Collection”.

- myisam_use_mmap

Property	Value
Command-Line Format	--myisam-use-mmap[={OFF|ON}]
System Variable	myisam_use_mmap
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Boolean
Default Value	`OFF`

Use memory mapping for reading and writing `MyISAM` tables.

- `mysql_native_password_proxy_users`

Property	Value
Command-Line Format	`--mysql-native-password-proxy-users[={OFF|ON}]`
System Variable	`mysql_native_password_proxy_users`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This variable controls whether the `mysql_native_password` built-in authentication plugin supports proxy users. It has no effect unless the `check_proxy_users` system variable is enabled. For information about user proxying, see Section 6.2.18, “Proxy Users”.

- `named_pipe`

Property	Value
Command-Line Format	`--named-pipe[={OFF|ON}]`
System Variable	`named_pipe`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Platform Specific	Windows
Type	Boolean
Default Value	`OFF`

(Windows only.) Indicates whether the server supports connections over named pipes.

- `named_pipe_full_access_group`

Property	Value
Command-Line Format	`--named-pipe-full-access-group=value`
Introduced	8.0.14
System Variable	`named_pipe_full_access_group`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Platform Specific	Windows

Property	Value
Type	String
Default Value	`*everyone*`

Valid Values	`*everyone*`
	
	`empty string`

(Windows only.) The access control granted to clients on the named pipe created by the MySQL server is set to the minimum necessary for successful communication when the `named_pipe` system variable is enabled to support named-pipe connections. Newer MySQL client software can open named pipe connections without any additional configuration, however, older client software may still require full access to open a named pipe connection.

This variable sets the name of a Windows local group whose members are granted sufficient access by the MySQL server to use older named-pipe clients. Initially, the value is set to `'*everyone*'` by default, which permits users of the Everyone group on Windows to continue using older clients until the older clients are upgraded. In contrast, setting the value to an empty string means that no Windows user will be granted full access to the named pipe. The default value `'*everyone*'` provides a language-independent way of referring to the Everyone group on Windows.

Ideally, a new Windows local group name (for example, `mysql_old_client_users`) should be created in Windows and then used to replace the default value for this variable when access to older client software is absolutely necessary. In this case, limit the membership of the group to as few users as possible, removing users from the group when their client software is upgraded. A non-member of the group who attempts to open a connection to MySQL with the older named-pipe client is denied access until the user is added to the group by a Windows administrator, and then the user logs out and logs in (required by Windows).

- `net_buffer_length`

Property	Value
Command-Line Format	`--net-buffer-length=#`
System Variable	`net_buffer_length`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`16384`
Minimum Value	`1024`
Maximum Value	`1048576`

Each client thread is associated with a connection buffer and result buffer. Both begin with a size given by `net_buffer_length` but are dynamically enlarged up to `max_allowed_packet` bytes as needed. The result buffer shrinks to `net_buffer_length` after each SQL statement.

This variable should not normally be changed, but if you have very little memory, you can set it to the expected length of statements sent by clients. If statements exceed this length, the connection buffer is automatically enlarged. The maximum value to which `net_buffer_length` can be set is 1MB.

The session value of this variable is read only.

- `net_read_timeout`

Property	Value
Command-Line Format	`--net-read-timeout=#`
System Variable	`net_read_timeout`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`30`
Minimum Value	`1`

The number of seconds to wait for more data from a connection before aborting the read. When the server is reading from the client, `net_read_timeout` is the timeout value controlling when to abort. When the server is writing to the client, `net_write_timeout` is the timeout value controlling when to abort. See also `slave_net_timeout`.

- `net_retry_count`

Property	Value
Command-Line Format	`--net-retry-count=#`
System Variable	`net_retry_count`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10`
Minimum Value	`1`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

If a read or write on a communication port is interrupted, retry this many times before giving up. This value should be set quite high on FreeBSD because internal interrupts are sent to all threads.

- `net_write_timeout`

Property	Value
Command-Line Format	`--net-write-timeout=#`
System Variable	`net_write_timeout`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value	`60`
Minimum Value	`1`

The number of seconds to wait for a block to be written to a connection before aborting the write. See also `net_read_timeout`.

- `new`

Property	Value
Command-Line Format	`--new[={OFF|ON}]`
System Variable	`new`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Disabled by	`skip-new`
Type	Boolean
Default Value	`OFF`

This variable was used in MySQL 4.0 to turn on some 4.1 behaviors, and is retained for backward compatibility. Its value is always `OFF`.

In NDB Cluster, setting this variable to `ON` makes it possible to employ partitioning types other than `KEY` or `LINEAR KEY` with `NDB` tables. *This feature is experimental only, and not supported in production*. For additional information, see User-defined partitioning and the NDB storage engine (NDB Cluster).

- `ngram_token_size`

Property	Value
Command-Line Format	`--ngram-token-size=#`
System Variable	`ngram_token_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`2`
Minimum Value	`1`
Maximum Value	`10`

Defines the n-gram token size for the n-gram full-text parser. The `ngram_token_size` option is read-only and can only be modified at startup. The default value is 2 (bigram). The maximum value is 10.

For more information about how to configure this variable, see Section 12.9.8, “ngram Full-Text Parser”.

- `offline_mode`

Property	Value
Command-Line Format	`--offline-mode[={OFF|ON}]`
System Variable	`offline_mode`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether the server is in “offline mode”, which has these characteristics:

- Connected client users who do not have the `CONNECTION_ADMIN` or `SUPER` privilege are disconnected on the next request, with an appropriate error. Disconnection includes terminating running statements and releasing locks. Such clients also cannot initiate new connections, and receive an appropriate error.
- Connected client users who have the `CONNECTION_ADMIN` or `SUPER` privilege are not disconnected, and can initiate new connections to manage the server.
- Replication slave threads are permitted to keep applying data to the server.

Only users who have the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege can control offline mode. To put a server in offline mode, change the value of the `offline_mode` system variable from `OFF` to `ON`. To resume normal operations, change `offline_mode` from `ON` to `OFF`. In offline mode, clients that are refused access receive an `ER_SERVER_OFFLINE_MODE` error.

- `old`

Property	Value
Command-Line Format	`--old[={OFF|ON}]`
System Variable	`old`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

`old` is a compatibility variable. It is disabled by default, but can be enabled at startup to revert the server to behaviors present in older versions.

When `old` is enabled, it changes the default scope of index hints to that used prior to MySQL 5.1.17. That is, index hints with no `FOR` clause apply only to how indexes are used for row retrieval and not to resolution of `ORDER BY` or `GROUP BY` clauses. (See Section 8.9.4, “Index Hints”.) Take care about enabling this in a replication setup. With statement-based binary logging, having different modes for the master and slaves might lead to replication errors.

- `old_alter_table`

Property	Value
Command-Line Format	`--old-alter-table[={OFF|ON}]`
System Variable	`old_alter_table`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

When this variable is enabled, the server does not use the optimized method of processing an `ALTER TABLE` operation. It reverts to using a temporary table, copying over the data, and then renaming the temporary table to the original, as used by MySQL 5.0 and earlier. For more information on the operation of `ALTER TABLE`, see Section 13.1.9, “ALTER TABLE Syntax”.

`ALTER TABLE ... DROP PARTITION` with `old_alter_table=ON` rebuilds the partitioned table and attempts to move data from the dropped partition to another partition with a compatible `PARTITION ... VALUES` definition. Data that cannot be moved to another partition is deleted. In earlier releases, `ALTER TABLE ... DROP PARTITION` with `old_alter_table=ON` deletes data stored in the partition and drops the partition.

- `old_passwords`

Property	Value
Command-Line Format	`--old-passwords=value`
Deprecated	Yes (removed in 8.0.11)
System Variable	`old_passwords`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`0`
Valid Values	`0` `2`

This system variable was removed in MySQL 8.0.11.

- `open_files_limit`

Property	Value
Command-Line Format	`--open-files-limit=#`
System Variable	`open_files_limit`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	`5000, with possible adjustment`
Minimum Value	`0`
Maximum Value	`platform dependent`

The number of file descriptors available to `mysqld` from the operating system. `mysqld` reserves descriptors with `setrlimit()`, using the value requested at startup by setting this variable or the `--open-files-limit` option to `mysqld_safe`. You should try increasing the value of this variable if `mysqld` gives you the error `Too many open files`. Internally, the maximum value for this variable is the maximum unsigned integer value, but the actual maximum is platform dependent.

The runtime variable value indicates the number of file descriptors actually permitted by the system and might differ from the value requested at startup. If the requested number of file descriptors cannot be allocated, `mysqld` writes a warning to the error log.

The effective `open_files_limit` value is based on the value specified at system startup (if any) and the values of `max_connections` and `table_open_cache`, using these formulas:

```
1) 10 + max_connections + (table_open_cache * 2)
2) max_connections * 5
3) operating system limit if positive
4) if operating system limit is Infinity:
   open_files_limit value specified at startup, 5000 if none
```

The server attempts to obtain the number of file descriptors using the maximum of those three values. If that many descriptors cannot be obtained, the server attempts to obtain as many as the system will permit.

The value is 0 on systems where MySQL cannot change the number of open files.

On Unix, the value cannot be set greater than `ulimit -n`.

- `optimizer_prune_level`

Property	Value
Command-Line Format	`--optimizer-prune-level=#`
System Variable	`optimizer_prune_level`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`1`
Minimum Value	`0`
Maximum Value	`1`

Controls the heuristics applied during query optimization to prune less-promising partial plans from the optimizer search space. A value of 0 disables heuristics so that the optimizer performs an exhaustive search. A value of 1 causes the optimizer to prune plans based on the number of rows retrieved by intermediate plans.

- `optimizer_search_depth`

Property	Value
Command-Line Format	`--optimizer-search-depth=#`
System Variable	`optimizer_search_depth`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`62`
Minimum Value	`0`
Maximum Value	`62`

The maximum depth of search performed by the query optimizer. Values larger than the number of relations in a query result in better query plans, but take longer to generate an execution plan for a query. Values smaller than the number of relations in a query return an execution plan quicker, but the resulting plan may be far from being optimal. If set to 0, the system automatically picks a reasonable value.

- `optimizer_switch`

Property	Value
Command-Line Format	`--optimizer-switch=value`
System Variable	`optimizer_switch`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Set

Valid Values (>= 8.0.13)	`batched_key_access={on|off}`
	`block_nested_loop={on|off}`
	`condition_fanout_filter={on|off}`
	`derived_merge={on|off}`
	`duplicateweedout={on|off}`
	`engine_condition_pushdown={on|off}`
	`firstmatch={on|off}`
	`index_condition_pushdown={on|off}`
	`index_merge={on|off}`
	`index_merge_intersection={on|off}`
	`index_merge_sort_union={on|off}`

Property	Value
	index_merge_union={on|off} loosescan={on|off} materialization={on|off} mrr={on|off} mrr_cost_based={on|off} semijoin={on|off} skip_scan={on|off} subquery_materialization_cost_based={on|off} use_index_extensions={on|off} use_invisible_indexes={on|off}
Valid Values (>= 8.0.3, <= 8.0.12)	batched_key_access={on|off} block_nested_loop={on|off} condition_fanout_filter={on|off} derived_merge={on|off} duplicateweedout={on|off} engine_condition_pushdown={on|off} firstmatch={on|off} index_condition_pushdown={on|off} index_merge={on|off} index_merge_intersection={on|off} index_merge_sort_union={on|off} index_merge_union={on|off} loosescan={on|off} materialization={on|off} mrr={on|off} mrr_cost_based={on|off} semijoin={on|off} subquery_materialization_cost_based={on|off}

Property	Value
	use_index_extensions={on|off} use_invisible_indexes={on|off}
Valid Values (<= 8.0.2)	batched_key_access={on|off} block_nested_loop={on|off} condition_fanout_filter={on|off} derived_merge={on|off} duplicateweedout={on|off} engine_condition_pushdown={on|off} firstmatch={on|off} index_condition_pushdown={on|off} index_merge={on|off} index_merge_intersection={on|off} index_merge_sort_union={on|off} index_merge_union={on|off} loosescan={on|off} materialization={on|off} mrr={on|off} mrr_cost_based={on|off} semijoin={on|off} subquery_materialization_cost_based={on|off} use_index_extensions={on|off}

The `optimizer_switch` system variable enables control over optimizer behavior. The value of this variable is a set of flags, each of which has a value of `on` or `off` to indicate whether the corresponding optimizer behavior is enabled or disabled. This variable has global and session values and can be changed at runtime. The global default can be set at server startup.

To see the current set of optimizer flags, select the variable value:

```
mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,index_merge_intersection=on,
                    engine_condition_pushdown=on,index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,block_nested_loop=on,
                    batched_key_access=off,materialization=on,semijoin=on,
```

```
                    loosescan=on,firstmatch=on,duplicateweedout=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on,condition_fanout_filter=on,
                    derived_merge=on,use_invisible_indexes=off,skip_scan=on,
                    hash_join=on
```

For more information about the syntax of this variable and the optimizer behaviors that it controls, see Section 8.9.2, “Switchable Optimizations”.

- `optimizer_trace`

Property	Value
Command-Line Format	`--optimizer-trace=value`
System Variable	`optimizer_trace`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

This variable controls optimizer tracing. For details, see MySQL Internals: Tracing the Optimizer.

- `optimizer_trace_features`

Property	Value
Command-Line Format	`--optimizer-trace-features=value`
System Variable	`optimizer_trace_features`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

This variable enables or disables selected optimizer tracing features. For details, see MySQL Internals: Tracing the Optimizer.

- `optimizer_trace_limit`

Property	Value
Command-Line Format	`--optimizer-trace-limit=#`
System Variable	`optimizer_trace_limit`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`

The maximum number of optimizer traces to display. For details, see MySQL Internals: Tracing the Optimizer.

- `optimizer_trace_max_mem_size`

Property	Value
Command-Line Format	`--optimizer-trace-max-mem-size=#`
System Variable	`optimizer_trace_max_mem_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value (>= 8.0.4)	`1048576`
Default Value (<= 8.0.3)	`16384`

The maximum cumulative size of stored optimizer traces. For details, see MySQL Internals: Tracing the Optimizer.

- `optimizer_trace_offset`

Property	Value
Command-Line Format	`--optimizer-trace-offset=#`
System Variable	`optimizer_trace_offset`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1`

The offset of optimizer traces to display. For details, see MySQL Internals: Tracing the Optimizer.

- `performance_schema_xxx`

Performance Schema system variables are listed in Section 26.15, “Performance Schema System Variables”. These variables may be used to configure Performance Schema operation.

- `parser_max_mem_size`

Property	Value
Command-Line Format	`--parser-max-mem-size=#`
System Variable	`parser_max_mem_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (64-bit platforms)	`18446744073709551615`
Default Value (32-bit platforms)	`4294967295`

Property	Value
Minimum Value	`10000000`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The maximum amount of memory available to the parser. The default value places no limit on memory available. The value can be reduced to protect against out-of-memory situations caused by parsing long or complex SQL statements.

- `partial_revokes`

Property	Value
Command-Line Format	`--partial-revokes[={OFF|ON}]`
Introduced	8.0.16
System Variable	`partial_revokes`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF` (if partial revokes do not exist) `ON` (if partial revokes exist)

Enabling this variable makes it possible to revoke privileges partially. Specifically, for users who have privileges at the global level, `partial_revokes` enables privileges for specific schemas to be revoked while leaving the privileges in place for other schemas. For example, a user who has the global `UPDATE` privilege can be restricted from exercising this privilege on the `mysql` system schema. (Or, stated another way, the user is enabled to exercise the `UPDATE` privilege on all schemas except the `mysql` schema.) In this sense, the user's global `UPDATE` privilege is partially revoked.

Once enabled, `partial_revokes` cannot be disabled if any account has privilege restrictions. If any such account exists, disabling `partial_revokes` fails:

- For attempts to disable `partial_revokes` at startup, the server logs an error message and enables `partial_revokes`.
- For attempts to disable `partial_revokes` at runtime, an error occurs and the `partial_revokes` value remains unchanged.

To disable `partial_revokes` in this case, first modify each account that has partially revoked privileges, either by re-granting the privileges or by removing the account.

For more information, including instructions for removing partial revokes, see Section 6.2.12, “Privilege Restriction Using Partial Revokes”.

- `password_history`

Property	Value
Command-Line Format	`--password-history=#`
Introduced	8.0.3
System Variable	`password_history`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

This variable defines the global policy for controlling reuse of previous passwords based on required minimum number of password changes. For an account password used previously, this variable indicates the number of subsequent account password changes that must occur before the password can be reused. If the value is 0 (the default), there is no reuse restriction based on number of password changes.

Changes to this variable apply immediately to all accounts defined with the `PASSWORD HISTORY DEFAULT` option.

The global number-of-changes password reuse policy can be overridden as desired for individual accounts using the `PASSWORD HISTORY` option of the `CREATE USER` and `ALTER USER` statements. See Section 6.2.15, “Password Management”.

- `password_require_current`

Property	Value
Command-Line Format	`--password-require-current[={OFF|ON}]`
Introduced	8.0.13
System Variable	`password_require_current`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This variable defines the global policy for controlling whether attempts to change an account password must specify the current password to be replaced.

Changes to this variable apply immediately to all accounts defined with the `PASSWORD REQUIRE CURRENT DEFAULT` option.

The global verification-required policy can be overridden as desired for individual accounts using the `PASSWORD REQUIRE` option of the `CREATE USER` and `ALTER USER` statements. See Section 6.2.15, “Password Management”.

- `password_reuse_interval`

Property	Value
Command-Line Format	`--password-reuse-interval=#`
Introduced	8.0.3
System Variable	`password_reuse_interval`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

This variable defines the global policy for controlling reuse of previous passwords based on time elapsed. For an account password used previously, this variable indicates the number of days that must pass before the password can be reused. If the value is 0 (the default), there is no reuse restriction based on time elapsed.

Changes to this variable apply immediately to all accounts defined with the `PASSWORD REUSE INTERVAL DEFAULT` option.

The global time-elapsed password reuse policy can be overridden as desired for individual accounts using the `PASSWORD REUSE INTERVAL` option of the `CREATE USER` and `ALTER USER` statements. See Section 6.2.15, “Password Management”.

- `persisted_globals_load`

Property	Value
Command-Line Format	`--persisted-globals-load[={OFF|ON}]`
System Variable	`persisted_globals_load`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether to load persisted configuration settings from the `mysqld-auto.cnf` file in the data directory. The server normally processes this file at startup after all other option files (see Section 4.2.2.2, “Using Option Files”). Disabling `persisted_globals_load` causes the server startup sequence to skip `mysqld-auto.cnf`.

To modify the contents of `mysqld-auto.cnf`, use the `SET PERSIST`, `SET PERSIST_ONLY`, and `RESET PERSIST` statements. See Section 5.1.9.3, “Persisted System Variables”.

- `persist_only_admin_x509_subject`

Property	Value
Command-Line Format	`--persist-only-admin-x509-subject=string`
Introduced	8.0.14
System Variable	`persist_only_admin_x509_subject`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`empty string`

`SET PERSIST` and `SET PERSIST_ONLY` enable system variables to be persisted to the `mysqld-auto.cnf` option file in the data directory (see Section 13.7.6.1, “SET Syntax for Variable Assignment”). Persisting system variables enables runtime configuration changes that affect subsequent server restarts, which is convenient for remote administration not requiring direct access to MySQL server host option files. However, some system variables are nonpersistible or acan be persisted only under certain restrictive conditions.

The `persist_only_admin_x509_subject` system variable specifies the SSL certificate X.509 Subject value that users must have to be able to persist system variables that are persist-restricted. The default value is the empty string, which disables the Subject check so that persist-restricted system variables cannot be persisted by any user.

If `persist_only_admin_x509_subject` is nonempty, users who connect to the server using an encrypted connection and supply an SSL certificate with the designated Subject value then can use `SET PERSIST_ONLY` to persist persist-restricted system variables. For information about persist-restricted system variables and instructions for configuring MySQL to enable `persist_only_admin_x509_subject`, see Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”.

- `pid_file`

Property	Value
Command-Line Format	`--pid-file=file_name`
System Variable	`pid_file`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

The path name of the file in which the server writes its process ID. The server creates the file in the data directory unless an absolute path name is given to specify a different directory. If you specify this variable, you must specify a value. If you do not specify this variable, MySQL uses a default value of *`host_name`*`.pid`, where *`host_name`* is the name of the host machine.

The process ID file is used by other programs such as `mysqld_safe` to determine the server's process ID. On Windows, this variable also affects the default error log file name. See Section 5.4.2, “The Error Log”.

- `plugin_dir`

Property	Value
Command-Line Format	`--plugin-dir=dir_name`
System Variable	`plugin_dir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name
Default Value	`BASEDIR/lib/plugin`

The path name of the plugin directory.

If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `secure_file_priv` to a directory where `SELECT` writes can be made safely.

- `port`

Property	Value
Command-Line Format	`--port=port_num`
System Variable	`port`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`3306`
Minimum Value	`0`
Maximum Value	`65535`

The number of the port on which the server listens for TCP/IP connections. This variable can be set with the `--port` option.

- `preload_buffer_size`

Property	Value
Command-Line Format	`--preload-buffer-size=#`
System Variable	`preload_buffer_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`32768`
Minimum Value	`1024`

Property	Value
Maximum Value	1073741824

The size of the buffer that is allocated when preloading indexes.

- print_identified_with_as_hex

Property	Value
Command-Line Format	--print-identified-with-as-hex[={OFF|ON}]
Introduced	8.0.17
System Variable	print_identified_with_as_hex
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Password hash values displayed in the IDENTIFIED WITH clause of output from SHOW CREATE USER may contain unprintable characters that have adverse effects on terminal displays and in other environments. Enabling print_identified_with_as_hex causes SHOW CREATE USER to display such hash values as hexadecimal strings rather than as regular string literals. Hash values that do not contain unprintable characters still display as regular string literals, even with this variable enabled.

- profiling

If set to 0 or OFF (the default), statement profiling is disabled. If set to 1 or ON, statement profiling is enabled and the SHOW PROFILE and SHOW PROFILES statements provide access to profiling information. See Section 13.7.7.31, “SHOW PROFILES Syntax”.

This variable is deprecated and will be removed in a future MySQL release.

- profiling_history_size

The number of statements for which to maintain profiling information if profiling is enabled. The default value is 15. The maximum value is 100. Setting the value to 0 effectively disables profiling. See Section 13.7.7.31, “SHOW PROFILES Syntax”.

This variable is deprecated and will be removed in a future MySQL release.

- protocol_compression_algorithms

Property	Value
Command-Line Format	--protocol-compression-algorithms=value
Introduced	8.0.18
System Variable	protocol_compression_algorithms
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Set

Property	Value
Default Value	`zlib,zstd,uncompressed`

Valid Values	`zlib`
	`zstd`
	`uncompressed`

The compression algorithms that the server permits for incoming connections. These include connections by client programs and by servers participating in master/slave replication or Group Replication. Compression does not apply to X Protocol connections or connections for `FEDERATED` tables.

The variable value is a list of one or more comma-separated compression algorithm names, in any order, chosen from the following items (not case-sensitive):

- `zlib`: Permit connections that use the `zlib` compression algorithm.
- `zstd`: Permit connections that use the `zstd` compression algorithm (zstd 1.3).
- `uncompressed`: Permit uncompressed connections. If this algorithm name is not included in the `protocol_compression_algorithms` value, the server does not permit uncompressed connections. It permits only compressed connections that use whichever other algorithms are specified in the value, and there is no fallback to uncompressed connections.

The default value of `zlib,zstd,uncompressed` indicates that the server permits all compression algorithms.

For more information, see Section 4.2.6, “Connection Compression Control”.

- `protocol_version`

Property	Value
System Variable	`protocol_version`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer

The version of the client/server protocol used by the MySQL server.

- `proxy_user`

Property	Value
System Variable	`proxy_user`
Scope	Session
Dynamic	No
SET_VAR Hint Applies	No
Type	String

If the current client is a proxy for another user, this variable is the proxy user account name. Otherwise, this variable is NULL. See Section 6.2.18, “Proxy Users”.

- pseudo_slave_mode

Property	Value
System Variable	pseudo_slave_mode
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

This system variable is for internal server use. pseudo_slave_mode assists with the correct handling of transactions that originated on older or newer servers than the server currently processing them. mysqlbinlog sets the value of pseudo_slave_mode to true before executing any SQL statements.

Setting the session value of this system variable is a restricted operation. The session user must have either the REPLICATION_APPLIER privilege (see Section 17.3.3, “Replication Privilege Checks”), or privileges sufficient to set restricted session variables (see Section 5.1.9.1, “System Variable Privileges”). However, note that the variable is not intended for users to set; it is set automatically by the replication infrastructure.

pseudo_slave_mode has the following effects on the handling of prepared XA transactions, which can be attached to or detached from the handling session (by default, the session that issues XA START):

- If true, and the handling session has executed an internal-use BINLOG statement, XA transactions are automatically detached from the session as soon as the first part of the transaction up to XA PREPARE finishes, so they can be committed or rolled back by any session that has the XA_RECOVER_ADMIN privilege.
- If false, XA transactions remain attached to the handling session as long as that session is alive, during which time no other session can commit the transaction. The prepared transaction is only detached if the session disconnects or the server restarts.

pseudo_slave_mode has the following effects on the original_commit_timestamp replication delay timestamp and the original_server_version system variable:

- If true, transactions that do not explicitly set original_commit_timestamp or original_server_version are assumed to originate on another, unknown server, so the value 0, meaning unknown, is assigned to both the timestamp and the system variable.
- If false, transactions that do not explicitly set original_commit_timestamp or original_server_version are assumed to originate on the current server, so the current timestamp and the current server's version are assigned to the timestamp and the system variable.

In MySQL 8.0.14 and later, pseudo_slave_mode has the following effects on the handling of a statement that sets one or more unsupported (removed or unknown) SQL modes:

- If true, the server ignores the unsupported mode and raises a warning.
- If false, the server rejects the statement with ER_UNSUPPORTED_SQL_MODE.

- pseudo_thread_id

Property	Value
System Variable	`pseudo_thread_id`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

This variable is for internal server use.

As of MySQL 8.0.14, setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

- `query_alloc_block_size`

Property	Value
Command-Line Format	`--query-alloc-block-size=#`
System Variable	`query_alloc_block_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`8192`
Minimum Value	`1024`
Maximum Value	`4294967295`
Block Size	`1024`

The allocation size of memory blocks that are allocated for objects created during statement parsing and execution. If you have problems with memory fragmentation, it might help to increase this parameter.

- `query_cache_limit`

Property	Value
Command-Line Format	`--query-cache-limit=#`
Deprecated	Yes (removed in 8.0.3)
System Variable	`query_cache_limit`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1048576`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`

Property	Value
Maximum Value (32-bit platforms)	4294967295

This system variable was removed in MySQL 8.0.3.

- query_cache_min_res_unit

Property	Value
Command-Line Format	--query-cache-min-res-unit=#
Deprecated	Yes (removed in 8.0.3)
System Variable	query_cache_min_res_unit
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	4096
Minimum Value	512
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

This system variable was removed in MySQL 8.0.3.

- query_cache_size

Property	Value
Command-Line Format	--query-cache-size=#
Deprecated	Yes (removed in 8.0.3)
System Variable	query_cache_size
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (64-bit platforms, >= 8.0.1)	0
Default Value (64-bit platforms, 8.0.0)	1048576
Default Value (32-bit platforms, >= 8.0.1)	0
Default Value (32-bit platforms, 8.0.0)	1048576
Minimum Value	0

Property	Value
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

This system variable was removed in MySQL 8.0.3.

- query_cache_type

Property	Value
Command-Line Format	--query-cache-type=#
Deprecated	Yes (removed in 8.0.3)
System Variable	query_cache_type
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	0
Valid Values	0 1 2

This system variable was removed in MySQL 8.0.3.

- query_cache_wlock_invalidate

Property	Value
Command-Line Format	--query-cache-wlock-invalidate[={OFF|ON}]
Deprecated	Yes (removed in 8.0.3)
System Variable	query_cache_wlock_invalidate
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

This system variable was removed in MySQL 8.0.3.

- query_prealloc_size

Property	Value
Command-Line Format	--query-prealloc-size=#
System Variable	query_prealloc_size

Property	Value
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	8192
Minimum Value	8192
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295
Block Size	1024

The size of the persistent buffer used for statement parsing and execution. This buffer is not freed between statements. If you are running complex queries, a larger `query_prealloc_size` value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations.

- `rand_seed1`

Property	Value
System Variable	rand_seed1
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

The `rand_seed1` and `rand_seed2` variables exist as session variables only, and can be set but not read. The variables—but not their values—are shown in the output of `SHOW VARIABLES`.

The purpose of these variables is to support replication of the `RAND()` function. For statements that invoke `RAND()`, the master passes two values to the slave, where they are used to seed the random number generator. The slave uses these values to set the session variables `rand_seed1` and `rand_seed2` so that `RAND()` on the slave generates the same value as on the master.

- `rand_seed2`

  See the description for `rand_seed1`.

- `range_alloc_block_size`

Property	Value
Command-Line Format	--range-alloc-block-size=#
System Variable	range_alloc_block_size
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes

Property	Value
Type	Integer
Default Value	`4096`
Minimum Value	`4096`
Maximum Value (64-bit platforms)	`18446744073709547520`
Maximum Value	`4294967295`
Block Size	`1024`

The size of blocks that are allocated when doing range optimization.

- `range_optimizer_max_mem_size`

Property	Value
Command-Line Format	`--range-optimizer-max-mem-size=#`
System Variable	`range_optimizer_max_mem_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`8388608`
Minimum Value	`0`
Maximum Value	`18446744073709551615`

The limit on memory consumption for the range optimizer. A value of 0 means “no limit.” If an execution plan considered by the optimizer uses the range access method but the optimizer estimates that the amount of memory needed for this method would exceed the limit, it abandons the plan and considers other plans. For more information, see Limiting Memory Use for Range Optimization.

- `rbr_exec_mode`

Property	Value
System Variable	`rbr_exec_mode`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`STRICT`
Valid Values	`IDEMPOTENT` `STRICT`

For internal use by `mysqlbinlog`. This variable switches the server between `IDEMPOTENT` mode and `STRICT` mode. `IDEMPOTENT` mode causes suppression of duplicate-key and no-key-found errors in `BINLOG` statements generated by `mysqlbinlog`. This mode is useful when replaying a row-based

binary log on a server that causes conflicts with existing data. `mysqlbinlog` sets this mode when you specify the `--idempotent` option by writing the following to the output:

```
SET SESSION RBR_EXEC_MODE=IDEMPOTENT;
```

As of MySQL 8.0.18, setting the session value of this system variable is no longer a restricted operation.

- `read_buffer_size`

Property	Value
Command-Line Format	`--read-buffer-size=#`
System Variable	`read_buffer_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`131072`
Minimum Value	`8200`
Maximum Value	`2147479552`

Each thread that does a sequential scan for a `MyISAM` table allocates a buffer of this size (in bytes) for each table it scans. If you do many sequential scans, you might want to increase this value, which defaults to 131072. The value of this variable should be a multiple of 4KB. If it is set to a value that is not a multiple of 4KB, its value will be rounded down to the nearest multiple of 4KB.

This option is also used in the following context for all storage engines:

- For caching the indexes in a temporary file (not a temporary table), when sorting rows for `ORDER BY`.
- For bulk insert into partitions.
- For caching results of nested queries.

`read_buffer_size` is also used in one other storage engine-specific way: to determine the memory block size for `MEMORY` tables.

For more information about memory use during different operations, see Section 8.12.3.1, “How MySQL Uses Memory”.

- `read_only`

Property	Value
Command-Line Format	`--read-only[={OFF|ON}]`
System Variable	`read_only`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

When the `read_only` system variable is enabled, the server permits no client updates except from users who have the `CONNECTION_ADMIN` or `SUPER` privilege. This variable is disabled by default.

The server also supports a `super_read_only` system variable (disabled by default), which has these effects:

- If `super_read_only` is enabled, the server prohibits client updates, even from users who have the `SUPER` privilege.
- Setting `super_read_only` to `ON` implicitly forces `read_only` to `ON`.
- Setting `read_only` to `OFF` implicitly forces `super_read_only` to `OFF`.

Even with `read_only` enabled, the server permits these operations:

- Updates performed by slave threads, if the server is a replication slave. In replication setups, it can be useful to enable `read_only` on slave servers to ensure that slaves accept updates only from the master server and not from clients.
- Writes to the system table `mysql.gtid_executed`, which stores GTIDs for executed transactions that are not present in the current binary log file.
- Use of `ANALYZE TABLE` or `OPTIMIZE TABLE` statements. The purpose of read-only mode is to prevent changes to table structure or contents. Analysis and optimization do not qualify as such changes. This means, for example, that consistency checks on read-only replication slaves can be performed with `mysqlcheck --all-databases --analyze`.
- Operations on `TEMPORARY` tables.
- Inserts into the log tables (`mysql.general_log` and `mysql.slow_log`); see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”.
- Updates to Performance Schema tables, such as `UPDATE` or `TRUNCATE TABLE` operations.

Changes to `read_only` on a master server are not replicated to slave servers. The value can be set on a slave server independent of the setting on the master.

The following conditions apply to attempts to enable `read_only` (including implicit attempts resulting from enabling `super_read_only`):

- The attempt fails and an error occurs if you have any explicit locks (acquired with `LOCK TABLES`) or have a pending transaction.
- The attempt blocks while other clients have any ongoing statement, active `LOCK TABLES WRITE`, or ongoing commit, until the locks are released and the statements and transactions end. While the attempt to enable `read_only` is pending, requests by other clients for table locks or to begin transactions also block until `read_only` has been set.
- The attempt blocks if there are active transactions that hold metadata locks, until those transactions end.
- `read_only` can be enabled while you hold a global read lock (acquired with `FLUSH TABLES WITH READ LOCK`) because that does not involve table locks.

- `read_rnd_buffer_size`

Property	Value
Command-Line Format	`--read-rnd-buffer-size=#`
System Variable	`read_rnd_buffer_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`262144`
Minimum Value	`1`
Maximum Value	`2147483647`

This variable is used for reads from `MyISAM` tables, and, for any storage engine, for Multi-Range Read optimization.

When reading rows from a `MyISAM` table in sorted order following a key-sorting operation, the rows are read through this buffer to avoid disk seeks. See Section 8.2.1.16, “ORDER BY Optimization”. Setting the variable to a large value can improve `ORDER BY` performance by a lot. However, this is a buffer allocated for each client, so you should not set the global variable to a large value. Instead, change the session variable only from within those clients that need to run large queries.

For more information about memory use during different operations, see Section 8.12.3.1, “How MySQL Uses Memory”. For information about Multi-Range Read optimization, see Section 8.2.1.11, “Multi-Range Read Optimization”.

- `regexp_stack_limit`

Property	Value
Command-Line Format	`--regexp-stack-limit=#`
Introduced	8.0.4
System Variable	`regexp_stack_limit`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`8000000`
Minimum Value	`0`
Maximum Value	`2147483647`

The maximum available memory in bytes for the internal stack used for regular expression matching operations performed by `REGEXP_LIKE()` and similar functions (see Section 12.5.2, “Regular Expressions”).

- `regexp_time_limit`

Property	Value
Command-Line Format	`--regexp-time-limit=#`

Property	Value
Introduced	8.0.4
System Variable	`regexp_time_limit`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`32`
Minimum Value	`0`
Maximum Value	`2147483647`

The time limit for regular expression matching operations performed by `REGEXP_LIKE()` and similar functions (see Section 12.5.2, “Regular Expressions”). This limit is expressed as the maximum permitted number of steps performed by the match engine, and thus affects execution time only indirectly. Typically, it is on the order of milliseconds.

- `require_secure_transport`

Property	Value
Command-Line Format	`--require-secure-transport[={OFF|ON}]`
System Variable	`require_secure_transport`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether client connections to the server are required to use some form of secure transport. When this variable is enabled, the server permits only TCP/IP connections that use SSL, or connections that use a socket file (on Unix) or shared memory (on Windows). The server rejects nonsecure connection attempts, which fail with an `ER_SECURE_TRANSPORT_REQUIRED` error.

This capability supplements per-account SSL requirements, which take precedence. For example, if an account is defined with `REQUIRE SSL`, enabling `require_secure_transport` does not make it possible to use the account to connect using a Unix socket file.

It is possible for a server to have no secure transports available. For example, a server on Windows supports no secure transports if started without specifying any SSL certificate or key files and with the `shared_memory` system variable disabled. Under these conditions, attempts to enable `require_secure_transport` at startup cause the server to write a message to the error log and exit. Attempts to enable the variable at runtime fail with an `ER_NO_SECURE_TRANSPORTS_CONFIGURED` error.

See also Configuring Encrypted Connections as Mandatory.

- `resultset_metadata`

Property	Value
Introduced	8.0.3
System Variable	`resultset_metadata`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`FULL`
Valid Values	`FULL` `NONE`

For connections for which metadata transfer is optional, the client sets the `resultset_metadata` system variable to control whether the server returns result set metadata. Permitted values are `FULL` (return all metadata; this is the default) and `NONE` (return no metadata).

For connections that are not metadata-optional, setting `resultset_metadata` to `NONE` produces an error.

For details about managing result set metadata transfer, see Section 28.7.27, “C API Optional Result Set Metadata”.

- `secondary_engine_cost_threshold`

Property	Value
Introduced	8.0.16
System Variable	`secondary_engine_cost_threshold`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Numeric
Default Value	`100000.000000`
Minimum Value	`0`
Maximum Value	`DBL_MAX (maximum double value)`

For future use.

- `schema_definition_cache`

Property	Value
Command-Line Format	`--schema-definition-cache=#`
System Variable	`schema_definition_cache`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	`256`
Minimum Value	`256`
Maximum Value	`524288`

Defines a limit for the number of schema definition objects, both used and unused, that can be kept in the dictionary object cache.

Unused schema definition objects are only kept in the dictionary object cache when the number in use is less than the capacity defined by `schema_definition_cache`.

A setting of `0` means that schema definition objects are only kept in the dictionary object cache while they are in use.

For more information, see Section 14.4, “Dictionary Object Cache”.

- `secure_auth`

Property	Value
Command-Line Format	`--secure-auth[={OFF|ON}]`
Deprecated	Yes (removed in 8.0.3)
System Variable	`secure_auth`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`
Valid Values	`ON`

This system variable was removed in MySQL 8.0.3.

- `secure_file_priv`

Property	Value
Command-Line Format	`--secure-file-priv=dir_name`
System Variable	`secure_file_priv`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`platform specific`

Valid Values	`empty string`
	`dirname`
	`NULL`

This variable is used to limit the effect of data import and export operations, such as those performed by the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function. These operations are permitted only to users who have the `FILE` privilege.

`secure_file_priv` may be set as follows:

- If empty, the variable has no effect. This is not a secure setting.
- If set to the name of a directory, the server limits import and export operations to work only with files in that directory. The directory must exist; the server will not create it.
- If set to `NULL`, the server disables import and export operations.

The default value is platform specific and depends on the value of the `INSTALL_LAYOUT` `CMake` option, as shown in the following table. To specify the default `secure_file_priv` value explicitly if you are building from source, use the `INSTALL_SECURE_FILE_PRIVDIR` `CMake` option.

`INSTALL_LAYOUT` Value	**Default `secure_file_priv` Value**
`STANDALONE`	empty
`DEB`, `RPM`, `SVR4`	`/var/lib/mysql-files`
Otherwise	`mysql-files` under the `CMAKE_INSTALL_PREFIX` value

The server checks the value of `secure_file_priv` at startup and writes a warning to the error log if the value is insecure. A non-`NULL` value is considered insecure if it is empty, or the value is the data directory or a subdirectory of it, or a directory that is accessible by all users. If `secure_file_priv` is set to a nonexistent path, the server writes an error message to the error log and exits.

- `server_id`

Property	**Value**
Command-Line Format	`--server-id=#`
System Variable	`server_id`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value (>= 8.0.3)	`1`
Default Value (<= 8.0.2)	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

Specifies the server ID. This variable is set by the `--server-id` option. The `server_id` system variable is set to 1 by default. The server can be started with this default ID, but when binary logging is enabled, an informational message is issued if you did not specify a server ID explicitly using the `--server-id` option.

For servers that are used in a replication topology, you must specify a unique server ID for each replication server, in the range from 1 to $2^{32} - 1$. “Unique” means that each ID must be different from every other ID in use by any other replication master or slave. For additional information, see

Section 17.1.6.2, “Replication Master Options and Variables”, and Section 17.1.6.3, “Replication Slave Options and Variables”.

If the server ID is set to 0, binary logging takes place, but a master with a server ID of 0 refuses any connections from slaves, and a slave with a server ID of 0 refuses to connect to a master. Note that although you can change the server ID dynamically to a nonzero value, doing so does not enable replication to start immediately. You must change the server ID and then restart the server to initialize the replication slave.

For more information, see Section 17.1.2.2, “Setting the Replication Slave Configuration”.

- `session_track_gtids`

Property	Value
Command-Line Format	`--session-track-gtids=value`
System Variable	`session_track_gtids`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`OFF`

Valid Values	`OFF`
	`OWN_GTID`
	`ALL_GTIDS`

Controls whether the server tracks GTIDs within the current session and returns them to the client. Depending on the variable value, at the end of executing each transaction, the server GTIDs are captured by the tracker and returned to the client. These `session_track_gtids` values are permitted:

- `OFF`: The tracker collects no GTIDs. This is the default.
- `OWN_GTID`: The tracker collects GTIDs generated by successfully committed read/write transactions.
- `ALL_GTIDS`: The tracker collects all GTIDs in the `gtid_executed` system variable at the time the current transaction commits, regardless of whether the transaction is read/write or read only.

`session_track_gtids` cannot be set within transactional context.

For more information about session state tracking, see Section 5.1.15, “Server Tracking of Client Session State Changes”.

- `session_track_schema`

Property	Value
Command-Line Format	`--session-track-schema[={OFF|ON}]`
System Variable	`session_track_schema`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Boolean
Default Value	`ON`

Controls whether the server tracks when the default schema (database) is set within the current session and notifies the client to make the schema name available.

If the schema name tracker is enabled, name notification occurs each time the default schema is set, even if the new schema name is the same as the old.

For more information about session state tracking, see Section 5.1.15, “Server Tracking of Client Session State Changes”.

- `session_track_state_change`

Property	Value
Command-Line Format	`--session-track-state-change[={OFF|ON}]`
System Variable	`session_track_state_change`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Controls whether the server tracks changes to the state of the current session and notifies the client when state changes occur. Changes can be reported for these attributes of client session state:

- The default schema (database).
- Session-specific values for system variables.
- User-defined variables.
- Temporary tables.
- Prepared statements.

If the session state tracker is enabled, notification occurs for each change that involves tracked session attributes, even if the new attribute values are the same as the old. For example, setting a user-defined variable to its current value results in a notification.

The `session_track_state_change` variable controls only notification of when changes occur, not what the changes are. For example, state-change notifications occur when the default schema is set or tracked session system variables are assigned, but the notification does not include the schema name or variable values. To receive notification of the schema name or session system variable values, use the `session_track_schema` or `session_track_system_variables` system variable, respectively.

Note

Assigning a value to `session_track_state_change` itself is not considered a state change and is not reported as such. However, if its name listed in the value

of `session_track_system_variables`, any assignments to it do result in notification of the new value.

For more information about session state tracking, see Section 5.1.15, “Server Tracking of Client Session State Changes”.

- `session_track_system_variables`

Property	Value
Command-Line Format	`--session-track-system-variables=#`
System Variable	`session_track_system_variables`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`time_zone, autocommit, character_set_client, character_set_results, character_set_connection`

Controls whether the server tracks assignments to session system variables and notifies the client of the name and value of each assigned variable. The variable value is a comma-separated list of variables for which to track assignments. By default, notification is enabled for `time_zone`, `autocommit`, `character_set_client`, `character_set_results`, and `character_set_connection`. (The latter three variables are those affected by `SET NAMES`.)

The special value `*` causes the server to track assignments to all session variables. If given, this value must be specified by itself without specific system variable names.

To disable notification of session variable assignments, set `session_track_system_variables` to the empty string.

If session system variable tracking is enabled, notification occurs for all assignments to tracked session variables, even if the new values are the same as the old.

For more information about session state tracking, see Section 5.1.15, “Server Tracking of Client Session State Changes”.

- `session_track_transaction_info`

Property	Value
Command-Line Format	`--session-track-transaction-info=value`
System Variable	`session_track_transaction_info`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`OFF`

Valid Values	`OFF`
	
	`STATE`

Property	Value
	CHARACTERISTICS

Controls whether the server tracks the state and characteristics of transactions within the current session and notifies the client to make this information available. These session_track_transaction_info values are permitted:

- OFF: Disable transaction state tracking. This is the default.
- STATE: Enable transaction state tracking without characteristics tracking. State tracking enables the client to determine whether a transaction is in progress and whether it could be moved to a different session without being rolled back.
- CHARACTERISTICS: Enable transaction state tracking, including characteristics tracking. Characteristics tracking enables the client to determine how to restart a transaction in another session so that it has the same characteristics as in the original session. The following characteristics are relevant for this purpose:

```
ISOLATION LEVEL
READ ONLY
READ WRITE
WITH CONSISTENT SNAPSHOT
```

For a client to safely relocate a transaction to another session, it must track not only transaction state but also transaction characteristics. In addition, the client must track the transaction_isolation and transaction_read_only system variables to correctly determine the session defaults. (To track these variables, list them in the value of the session_track_system_variables system variable.)

For more information about session state tracking, see Section 5.1.15, “Server Tracking of Client Session State Changes”.

- sha256_password_auto_generate_rsa_keys

Property	Value
Command-Line Format	--sha256-password-auto-generate-rsa-keys[={OFF|ON}]
System Variable	sha256_password_auto_generate_rsa_keys
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

The server uses this variable to determine whether to autogenerate RSA private/public key-pair files in the data directory if they do not already exist.

At startup, the server automatically generates RSA private/public key-pair files in the data directory if all of these conditions are true: The sha256_password_auto_generate_rsa_keys or caching_sha2_password_auto_generate_rsa_keys system variable is enabled; no RSA options are specified; the RSA files are missing from the data directory. These key-pair files enable secure password exchange using RSA over unencrypted connections for accounts authenticated by the

`sha256_password` or `caching_sha2_password` plugin; see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

For more information about RSA file autogeneration, including file names and characteristics, see Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”

The `auto_generate_certs` system variable is related but controls autogeneration of SSL certificate and key files needed for secure connections using SSL.

- `sha256_password_private_key_path`

Property	Value
Command-Line Format	`--sha256-password-private-key-path=file_name`
System Variable	`sha256_password_private_key_path`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	`private_key.pem`

The value of this variable is the path name of the RSA private key file for the `sha256_password` authentication plugin. If the file is named as a relative path, it is interpreted relative to the server data directory. The file must be in PEM format.

Important

Because this file stores a private key, its access mode should be restricted so that only the MySQL server can read it.

For information about `sha256_password`, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”.

- `sha256_password_proxy_users`

Property	Value
Command-Line Format	`--sha256-password-proxy-users[={OFF|ON}]`
System Variable	`sha256_password_proxy_users`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This variable controls whether the `sha256_password` built-in authentication plugin supports proxy users. It has no effect unless the `check_proxy_users` system variable is enabled. For information about user proxying, see Section 6.2.18, “Proxy Users”.

- `sha256_password_public_key_path`

Property	Value
Command-Line Format	`--sha256-password-public-key-path=file_name`
System Variable	`sha256_password_public_key_path`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	`public_key.pem`

The value of this variable is the path name of the RSA public key file for the `sha256_password` authentication plugin. If the file is named as a relative path, it is interpreted relative to the server data directory. The file must be in PEM format. Because this file stores a public key, copies can be freely distributed to client users. (Clients that explicitly specify a public key when connecting to the server using RSA password encryption must use the same public key as that used by the server.)

For information about `sha256_password`, including information about how clients specify the RSA public key, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”.

- `shared_memory`

Property	Value
Command-Line Format	`--shared-memory[={OFF|ON}]`
System Variable	`shared_memory`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Platform Specific	Windows
Type	Boolean
Default Value	`OFF`

(Windows only.) Whether the server permits shared-memory connections.

- `shared_memory_base_name`

Property	Value
Command-Line Format	`--shared-memory-base-name=name`
System Variable	`shared_memory_base_name`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Platform Specific	Windows
Type	String
Default Value	`MYSQL`

(Windows only.) The name of shared memory to use for shared-memory connections. This is useful when running multiple MySQL instances on a single physical machine. The default name is `MYSQL`. The name is case-sensitive.

This variable applies only if the server is started with the `shared_memory` system variable enabled to support shared-memory connections.

- `show_compatibility_56`

Property	Value
Command-Line Format	`--show-compatibility-56[={OFF|ON}]`
Deprecated	Yes (removed in 8.0.1)
System Variable	`show_compatibility_56`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This variable was used in the transition period during which system and status variable information in `INFORMATION_SCHEMA` tables was moved to Performance Schema tables. That transition period ended in MySQL 8.0.1, at which time this variable was removed. For advice on migrating away from the `INFORMATION_SCHEMA` tables to the Performance Schema tables, see Migrating to Performance Schema System and Status Variable Tables.

- `show_create_table_skip_secondary_engine`

Property	Value
Command-Line Format	`--show-create-table-skip-secondary-engine[={OFF|ON}]`
Introduced	8.0.18
System Variable	`show_create_table_skip_secondary_engine`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	`OFF`

Enabling `show_create_table_skip_secondary_engine` causes the `SECONDARY ENGINE` clause to be excluded from `SHOW CREATE TABLE` output, and from `CREATE TABLE` statements dumped by the `mysqldump` utility.

`mysqldump` provides the `--show-create-skip-secondary-engine` option. When specified, it enables the `show_create_table_skip_secondary_engine` system variable for the duration of the dump operation.

Attempting a `mysqldump` operation with the `--show-create-skip-secondary-engine` option on a release prior to MySQL 8.0.18 that does not support the `show_create_table_skip_secondary_engine` variable causes an error.

- show_create_table_verbosity

Property	Value
Command-Line Format	--show-create-table-verbosity[={OFF|ON}]
Introduced	8.0.11
System Variable	show_create_table_verbosity
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

SHOW CREATE TABLE normally does not show the ROW_FORMAT table option if the row format is the default format. Enabling this variable causes SHOW CREATE TABLE to display ROW_FORMAT regardless of whether it is the default format.

- show_old_temporals

Property	Value
Command-Line Format	--show-old-temporals[={OFF|ON}]
Deprecated	Yes
System Variable	show_old_temporals
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Whether SHOW CREATE TABLE output includes comments to flag temporal columns found to be in pre-5.6.4 format (TIME, DATETIME, and TIMESTAMP columns without support for fractional seconds precision). This variable is disabled by default. If enabled, SHOW CREATE TABLE output looks like this:

```
CREATE TABLE `mytbl` (
  `ts` timestamp /* 5.5 binary format */ NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `dt` datetime /* 5.5 binary format */ DEFAULT NULL,
  `t` time /* 5.5 binary format */ DEFAULT NULL
) DEFAULT CHARSET=utf8mb4
```

Output for the COLUMN_TYPE column of the INFORMATION_SCHEMA.COLUMNS table is affected similarly.

This variable is deprecated and will be removed in a future MySQL release.

- skip_external_locking

Property	Value
Command-Line Format	--skip-external-locking[={OFF|ON}]
System Variable	skip_external_locking

Property	Value
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

This is `OFF` if `mysqld` uses external locking (system locking), `ON` if external locking is disabled. This affects only `MyISAM` table access.

This variable is set by the `--external-locking` or `--skip-external-locking` option. External locking is disabled by default.

External locking affects only `MyISAM` table access. For more information, including conditions under which it can and cannot be used, see Section 8.11.5, “External Locking”.

- `skip_name_resolve`

Property	Value
Command-Line Format	`--skip-name-resolve[={OFF|ON}]`
System Variable	`skip_name_resolve`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This variable is set from the value of the `--skip-name-resolve` option. If it is `OFF`, `mysqld` resolves host names when checking client connections. If it is `ON`, `mysqld` uses only IP numbers; in this case, all `Host` column values in the grant tables must be IP addresses or `localhost`. See Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

- `skip_networking`

Property	Value
Command-Line Format	`--skip-networking`
System Variable	`skip_networking`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

This is `ON` if the server permits only local (non-TCP/IP) connections. On Unix, local connections use a Unix socket file. On Windows, local connections use a named pipe or shared memory. This variable can be set to `ON` with the `--skip-networking` option.

- `skip_show_database`

Property	Value
Command-Line Format	`--skip-show-database`
System Variable	`skip_show_database`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

This prevents people from using the `SHOW DATABASES` statement if they do not have the `SHOW DATABASES` privilege. This can improve security if you have concerns about users being able to see databases belonging to other users. Its effect depends on the `SHOW DATABASES` privilege: If the variable value is `ON`, the `SHOW DATABASES` statement is permitted only to users who have the `SHOW DATABASES` privilege, and the statement displays all database names. If the value is `OFF`, `SHOW DATABASES` is permitted to all users, but displays the names of only those databases for which the user has the `SHOW DATABASES` or other privilege.

Caution

Because any static global privilege is considered a privilege for all databases, any static global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`, except databases that have been restricted at the database level by partial revokes.

- `slow_launch_time`

Property	Value
Command-Line Format	`--slow-launch-time=#`
System Variable	`slow_launch_time`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`2`

If creating a thread takes longer than this many seconds, the server increments the `Slow_launch_threads` status variable.

- `slow_query_log`

Property	Value
Command-Line Format	`--slow-query-log[={OFF|ON}]`
System Variable	`slow_query_log`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean

Property	Value
Default Value	`OFF`

Whether the slow query log is enabled. The value can be 0 (or `OFF`) to disable the log or 1 (or `ON`) to enable the log. The destination for log output is controlled by the `log_output` system variable; if that value is `NONE`, no log entries are written even if the log is enabled.

"Slow" is determined by the value of the `long_query_time` variable. See Section 5.4.5, "The Slow Query Log".

- `slow_query_log_file`

Property	Value
Command-Line Format	`--slow-query-log-file=file_name`
System Variable	`slow_query_log_file`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	`host_name-slow.log`

The name of the slow query log file. The default value is *`host_name`*`-slow.log`, but the initial value can be changed with the `--slow_query_log_file` option.

- `socket`

Property	Value
Command-Line Format	`--socket={file_name|pipe_name}`
System Variable	`socket`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value (Other)	`/tmp/mysql.sock`
Default Value (Windows)	`MySQL`

On Unix platforms, this variable is the name of the socket file that is used for local client connections. The default is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

On Windows, this variable is the name of the named pipe that is used for local client connections. The default value is `MySQL` (not case-sensitive).

- `sort_buffer_size`

Property	Value
Command-Line Format	`--sort-buffer-size=#`
System Variable	`sort_buffer_size`

Property	Value
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	262144
Minimum Value	32768
Maximum Value (Other, 64-bit platforms)	18446744073709551615
Maximum Value (Other, 32-bit platforms)	4294967295
Maximum Value (Windows)	4294967295

Each session that must perform a sort allocates a buffer of this size. `sort_buffer_size` is not specific to any storage engine and applies in a general manner for optimization. At minimum the `sort_buffer_size` value must be large enough to accommodate fifteen tuples in the sort buffer. Also, increasing the value of `max_sort_length` may require increasing the value of `sort_buffer_size`. For more information, see Section 8.2.1.16, “ORDER BY Optimization”

If you see many `Sort_merge_passes` per second in `SHOW GLOBAL STATUS` output, you can consider increasing the `sort_buffer_size` value to speed up `ORDER BY` or `GROUP BY` operations that cannot be improved with query optimization or improved indexing.

The optimizer tries to work out how much space is needed but can allocate more, up to the limit. Setting it larger than required globally will slow down most queries that sort. It is best to increase it as a session setting, and only for the sessions that need a larger size. On Linux, there are thresholds of 256KB and 2MB where larger values may significantly slow down memory allocation, so you should consider staying below one of those values. Experiment to find the best value for your workload. See Section B.4.3.5, “Where MySQL Stores Temporary Files”.

The maximum permissible setting for `sort_buffer_size` is 4GB−1. Larger values are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB−1 with a warning).

- `sql_auto_is_null`

Property	Value
System Variable	`sql_auto_is_null`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	OFF

If this variable is enabled, then after a statement that successfully inserts an automatically generated `AUTO_INCREMENT` value, you can find that value by issuing a statement of the following form:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

If the statement returns a row, the value returned is the same as if you invoked the `LAST_INSERT_ID()` function. For details, including the return value after a multiple-row insert, see Section 12.15, "Information Functions". If no `AUTO_INCREMENT` value was successfully inserted, the `SELECT` statement returns no row.

The behavior of retrieving an `AUTO_INCREMENT` value by using an `IS NULL` comparison is used by some ODBC programs, such as Access. See Obtaining Auto-Increment Values. This behavior can be disabled by setting `sql_auto_is_null` to `OFF`.

Prior to MySQL 8.0.16, the transformation of `WHERE auto_col IS NULL` to `WHERE auto_col = LAST_INSERT_ID()` was performed only when the statement was executed, so that the value of `sql_auto_is_null` during execution determined whether the query was transformed. In MySQL 8.0.16 and later, the transformation is performed during statement preparation.

The default value of `sql_auto_is_null` is `OFF`.

- `sql_big_selects`

Property	Value
System Variable	`sql_big_selects`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	`ON`

If set to `OFF`, MySQL aborts `SELECT` statements that are likely to take a very long time to execute (that is, statements for which the optimizer estimates that the number of examined rows exceeds the value of `max_join_size`). This is useful when an inadvisable `WHERE` statement has been issued. The default value for a new connection is `ON`, which permits all `SELECT` statements.

If you set the `max_join_size` system variable to a value other than `DEFAULT`, `sql_big_selects` is set to `OFF`.

- `sql_buffer_result`

Property	Value
System Variable	`sql_buffer_result`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	`OFF`

If enabled, `sql_buffer_result` forces results from `SELECT` statements to be put into temporary tables. This helps MySQL free the table locks early and can be beneficial in cases where it takes a long time to send results to the client. The default value is `OFF`.

- `sql_log_off`

Property	Value
System Variable	`sql_log_off`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`
Valid Values	`OFF` (enable logging) `ON` (disable logging)

This variable controls whether logging to the general query log is disabled for the current session (assuming that the general query log itself is enabled). The default value is `OFF` (that is, enable logging). To disable or enable general query logging for the current session, set the session `sql_log_off` variable to `ON` or `OFF`.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

- `sql_mode`

Property	Value
Command-Line Format	`--sql-mode=name`
System Variable	`sql_mode`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Set
Default Value (>= 8.0.11)	`ONLY_FULL_GROUP_BY STRICT_TRANS_TABLES NO_ZERO_IN_DATE NO_ZERO_DATE ERROR_FOR_DIVISION_BY_ZERO NO_ENGINE_SUBSTITUTION`
Default Value (<= 8.0.4)	`ONLY_FULL_GROUP_BY STRICT_TRANS_TABLES NO_ZERO_IN_DATE NO_ZERO_DATE ERROR_FOR_DIVISION_BY_ZERO NO_AUTO_CREATE_USER NO_ENGINE_SUBSTITUTION`

Valid Values (>= 8.0.11)	`ALLOW_INVALID_DATES`
	
	`ANSI_QUOTES`
	
	`ERROR_FOR_DIVISION_BY_ZERO`
	
	`HIGH_NOT_PRECEDENCE`
	
	`IGNORE_SPACE`
	
	`NO_AUTO_VALUE_ON_ZERO`
	
	`NO_BACKSLASH_ESCAPES`

Property	Value
	NO_DIR_IN_CREATE NO_ENGINE_SUBSTITUTION NO_UNSIGNED_SUBTRACTION NO_ZERO_DATE NO_ZERO_IN_DATE ONLY_FULL_GROUP_BY PAD_CHAR_TO_FULL_LENGTH PIPES_AS_CONCAT REAL_AS_FLOAT STRICT_ALL_TABLES STRICT_TRANS_TABLES TIME_TRUNCATE_FRACTIONAL
Valid Values (>= 8.0.1, <= 8.0.4)	ALLOW_INVALID_DATES ANSI_QUOTES ERROR_FOR_DIVISION_BY_ZERO HIGH_NOT_PRECEDENCE IGNORE_SPACE NO_AUTO_CREATE_USER NO_AUTO_VALUE_ON_ZERO NO_BACKSLASH_ESCAPES NO_DIR_IN_CREATE NO_ENGINE_SUBSTITUTION NO_FIELD_OPTIONS NO_KEY_OPTIONS NO_TABLE_OPTIONS NO_UNSIGNED_SUBTRACTION NO_ZERO_DATE NO_ZERO_IN_DATE

Property	Value
	ONLY_FULL_GROUP_BY PAD_CHAR_TO_FULL_LENGTH PIPES_AS_CONCAT REAL_AS_FLOAT STRICT_ALL_TABLES STRICT_TRANS_TABLES TIME_TRUNCATE_FRACTIONAL
Valid Values (8.0.0)	ALLOW_INVALID_DATES ANSI_QUOTES ERROR_FOR_DIVISION_BY_ZERO HIGH_NOT_PRECEDENCE IGNORE_SPACE NO_AUTO_CREATE_USER NO_AUTO_VALUE_ON_ZERO NO_BACKSLASH_ESCAPES NO_DIR_IN_CREATE NO_ENGINE_SUBSTITUTION NO_FIELD_OPTIONS NO_KEY_OPTIONS NO_TABLE_OPTIONS NO_UNSIGNED_SUBTRACTION NO_ZERO_DATE NO_ZERO_IN_DATE ONLY_FULL_GROUP_BY PAD_CHAR_TO_FULL_LENGTH PIPES_AS_CONCAT REAL_AS_FLOAT STRICT_ALL_TABLES

Property	Value
	STRICT_TRANS_TABLES

The current server SQL mode, which can be set dynamically. For details, see Section 5.1.11, “Server SQL Modes”.

Note

MySQL installation programs may configure the SQL mode during the installation process.

If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

- sql_notes

Property	Value
System Variable	sql_notes
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

If enabled (the default), diagnostics of Note level increment warning_count and the server records them. If disabled, Note diagnostics do not increment warning_count and the server does not record them. mysqldump includes output to disable this variable so that reloading the dump file does not produce warnings for events that do not affect the integrity of the reload operation.

- sql_quote_show_create

Property	Value
System Variable	sql_quote_show_create
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

If enabled (the default), the server quotes identifiers for SHOW CREATE TABLE and SHOW CREATE DATABASE statements. If disabled, quoting is disabled. This option is enabled by default so that replication works for identifiers that require quoting. See Section 13.7.7.10, “SHOW CREATE TABLE Syntax”, and Section 13.7.7.6, “SHOW CREATE DATABASE Syntax”.

- sql_require_primary_key

Property	Value
Command-Line Format	--sql-require-primary-key[={OFF|ON}]

Property	Value
Introduced	8.0.13
System Variable	`sql_require_primary_key`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	`OFF`

Whether statements that create new tables or alter the structure of existing tables enforce the requirement that tables have a primary key.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Enabling this variable helps avoid performance problems in row-based replication that can occur when tables have no primary key. Suppose that a table has no primary key and an update or delete modifies multiple rows. On the master server, this operation can be performed using a single table scan but, when replicated using row-based replication, results in a table scan for each row to be modified on the slave. With a primary key, these table scans do not occur.

`sql_require_primary_key` applies to both base tables and `TEMPORARY` tables, and changes to its value are replicated to slave servers. As of MySQL 8.0.18, it applies only to storage engines that can participate in replication.

When enabled, `sql_require_primary_key` has these effects:

- Attempts to create a new table with no primary key fail with an error. This includes `CREATE TABLE ... LIKE`. It also includes `CREATE TABLE ... SELECT`, unless the `CREATE TABLE` part includes a primary key definition.
- Attempts to drop the primary key from an existing table fail with an error, with the exception that dropping the primary key and adding a primary key in the same `ALTER TABLE` statement is permitted.

  Dropping the primary key fails even if the table also contains a `UNIQUE NOT NULL` index.
- Attempts to import a table with no primary key fail with an error.

- `sql_safe_updates`

Property	Value
System Variable	`sql_safe_updates`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	`OFF`

If this variable is enabled, `UPDATE` and `DELETE` statements that do not use a key in the `WHERE` clause or a `LIMIT` clause produce an error. This makes it possible to catch `UPDATE` and `DELETE` statements

where keys are not used properly and that would probably change or delete a large number of rows. The default value is `OFF`.

For the `mysql` client, `sql_safe_updates` can be enabled by using the `--safe-updates` option. For more information, see Using Safe-Updates Mode (--safe-updates).

- `sql_select_limit`

Property	Value
System Variable	`sql_select_limit`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer

The maximum number of rows to return from `SELECT` statements. For more information, see Using Safe-Updates Mode (--safe-updates).

The default value for a new connection is the maximum number of rows that the server permits per table. Typical default values are $(2^{32})-1$ or $(2^{64})-1$. If you have changed the limit, the default value can be restored by assigning a value of `DEFAULT`.

If a `SELECT` has a `LIMIT` clause, the `LIMIT` takes precedence over the value of `sql_select_limit`.

- `sql_warnings`

Property	Value
System Variable	`sql_warnings`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This variable controls whether single-row `INSERT` statements produce an information string if warnings occur. The default is `OFF`. Set the value to `ON` to produce an information string.

- `ssl_ca`

Property	Value
Command-Line Format	`--ssl-ca=file_name`
System Variable	`ssl_ca`
Scope	Global
Dynamic (>= 8.0.16)	Yes
Dynamic (<= 8.0.15)	No
SET_VAR Hint Applies	No
Type	File name

The path to a file with a list of trusted SSL Certificate Authorities.

As of MySQL 8.0.16, this variable is dynamic and can be modified at runtime to affect the SSL context the server uses for new connections. See Server-Side Runtime Configuration for Encrypted Connections. Prior to MySQL 8.0.16, this variable can be set only at server startup.

- `ssl_capath`

Property	Value
Command-Line Format	`--ssl-capath=dir_name`
System Variable	`ssl_capath`
Scope	Global
Dynamic (>= 8.0.16)	Yes
Dynamic (<= 8.0.15)	No
SET_VAR Hint Applies	No
Type	Directory name

The path to a directory that contains trusted SSL CA certificates in PEM format.

As of MySQL 8.0.16, this variable is dynamic and can be modified at runtime to affect the SSL context the server uses for new connections. See Server-Side Runtime Configuration for Encrypted Connections. Prior to MySQL 8.0.16, this variable can be set only at server startup.

- `ssl_cert`

Property	Value
Command-Line Format	`--ssl-cert=file_name`
System Variable	`ssl_cert`
Scope	Global
Dynamic (>= 8.0.16)	Yes
Dynamic (<= 8.0.15)	No
SET_VAR Hint Applies	No
Type	File name

The name of the SSL certificate file to use for establishing secure connections.

As of MySQL 8.0.16, this variable is dynamic and can be modified at runtime to affect the SSL context the server uses for new connections. See Server-Side Runtime Configuration for Encrypted Connections. Prior to MySQL 8.0.16, this variable can be set only at server startup.

- `ssl_cipher`

Property	Value
Command-Line Format	`--ssl-cipher=name`
System Variable	`ssl_cipher`
Scope	Global
Dynamic (>= 8.0.16)	Yes

Property	Value
Dynamic (<= 8.0.15)	No
SET_VAR Hint Applies	No
Type	String

The list of permissible encryption ciphers for connections that use TLS protocols up through TLSv1.2.

As of MySQL 8.0.16, this variable is dynamic and can be modified at runtime to affect the SSL context the server uses for new connections. See Server-Side Runtime Configuration for Encrypted Connections. Prior to MySQL 8.0.16, this variable can be set only at server startup.

- `ssl_crl`

Property	Value
Command-Line Format	`--ssl-crl=file_name`
System Variable	`ssl_crl`
Scope	Global
Dynamic (>= 8.0.16)	Yes
Dynamic (<= 8.0.15)	No
SET_VAR Hint Applies	No
Type	File name

The path to a file containing certificate revocation lists in PEM format.

As of MySQL 8.0.16, this variable is dynamic and can be modified at runtime to affect the SSL context the server uses for new connections. See Server-Side Runtime Configuration for Encrypted Connections. Prior to MySQL 8.0.16, this variable can be set only at server startup.

- `ssl_crlpath`

Property	Value
Command-Line Format	`--ssl-crlpath=dir_name`
System Variable	`ssl_crlpath`
Scope	Global
Dynamic (>= 8.0.16)	Yes
Dynamic (<= 8.0.15)	No
SET_VAR Hint Applies	No
Type	Directory name

The path to a directory that contains files containing certificate revocation lists in PEM format.

As of MySQL 8.0.16, this variable is dynamic and can be modified at runtime to affect the SSL context the server uses for new connections. See Server-Side Runtime Configuration for Encrypted Connections. Prior to MySQL 8.0.16, this variable can be set only at server startup.

- `ssl_fips_mode`

Property	Value
Command-Line Format	`--ssl-fips-mode={OFF|ON|STRICT}`
Introduced	8.0.11
System Variable	`ssl_fips_mode`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`OFF`
Valid Values	`OFF` (or 0) `ON` (or 1) `STRICT` (or 2)

Controls whether to enable FIPS mode on the server side. The `ssl_fips_mode` system variable differs from other `--ssl-xxx` options in that it is not used to control whether the server permits encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

These `ssl_fips_mode` values are permitted:

- `OFF` (or 0): Disable FIPS mode.
- `ON` (or 1): Enable FIPS mode.
- `STRICT` (or 2): Enable “strict” FIPS mode.

Note

If the OpenSSL FIPS Object Module is not available, the only permitted value for `ssl_fips_mode` is `OFF`. In this case, setting `ssl_fips_mode` to `ON` or `STRICT` at startup causes the server to produce an error message and exit.

- `ssl_key`

Property	Value
Command-Line Format	`--ssl-key=file_name`
System Variable	`ssl_key`
Scope	Global
Dynamic (>= 8.0.16)	Yes
Dynamic (<= 8.0.15)	No
SET_VAR Hint Applies	No
Type	File name

The name of the SSL key file to use for establishing secure connections.

As of MySQL 8.0.16, this variable is dynamic and can be modified at runtime to affect the SSL context the server uses for new connections. See Server-Side Runtime Configuration for Encrypted Connections. Prior to MySQL 8.0.16, this variable can be set only at server startup.

- `stored_program_cache`

Property	Value
Command-Line Format	`--stored-program-cache=#`
System Variable	`stored_program_cache`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`256`
Minimum Value	`16`
Maximum Value	`524288`

Sets a soft upper limit for the number of cached stored routines per connection. The value of this variable is specified in terms of the number of stored routines held in each of the two caches maintained by the MySQL Server for, respectively, stored procedures and stored functions.

Whenever a stored routine is executed this cache size is checked before the first or top-level statement in the routine is parsed; if the number of routines of the same type (stored procedures or stored functions according to which is being executed) exceeds the limit specified by this variable, the corresponding cache is flushed and memory previously allocated for cached objects is freed. This allows the cache to be flushed safely, even when there are dependencies between stored routines.

The stored procedure and stored function caches exists in parallel with the stored program definition cache partition of the dictionary object cache. The stored procedure and stored function caches are per connection, while the stored program definition cache is shared. The existence of objects in the stored procedure and stored function caches have no dependence on the existence of objects in the stored program definition cache, and vice versa. For more information, see Section 14.4, “Dictionary Object Cache”.

- `stored_program_definition_cache`

Property	Value
Command-Line Format	`--stored-program-definition-cache=#`
System Variable	`stored_program_definition_cache`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`256`
Minimum Value	`256`
Maximum Value	`524288`

Defines a limit for the number of stored program definition objects, both used and unused, that can be kept in the dictionary object cache.

Unused stored program definition objects are only kept in the dictionary object cache when the number in use is less than the capacity defined by `stored_program_definition_cache`.

A setting of 0 means that stored program definition objects are only kept in the dictionary object cache while they are in use.

The stored program definition cache partition exists in parallel with the stored procedure and stored function caches that are configured using the `stored_program_cache` option.

The `stored_program_cache` option sets a soft upper limit for the number of cached stored procedures or functions per connection, and the limit is checked each time a connection executes a stored procedure or function. The stored program definition cache partition, on the other hand, is a shared cache that stores stored program definition objects for other purposes. The existence of objects in the stored program definition cache partition has no dependence on the existence of objects in the stored procedure cache or stored function cache, and vice versa.

For related information, see Section 14.4, “Dictionary Object Cache”.

- `super_read_only`

Property	Value
Command-Line Format	`--super-read-only[={OFF|ON}]`
System Variable	`super_read_only`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

If the `read_only` system variable is enabled, the server permits client updates only from users who have the `SUPER` privilege. If the `super_read_only` system variable is also enabled, the server prohibits client updates even from users who have `SUPER`. See the description of the `read_only` system variable for a description of read-only mode and information about how `read_only` and `super_read_only` interact.

Client updates prevented when `super_read_only` is enabled include operations that do not necessarily appear to be updates, such as `CREATE FUNCTION` (to install a UDF), `INSTALL PLUGIN`, and `INSTALL COMPONENT`. These operations are prohibited because they involve changes to tables in the `mysql` system schema.

Changes to `super_read_only` on a master server are not replicated to slave servers. The value can be set on a slave server independent of the setting on the master.

- `syseventlog.facility`

Property	Value
Command-Line Format	`--syseventlog.facility=value`
Introduced	8.0.13

Property	Value
System Variable	`syseventlog.facility`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`daemon`

The facility for error log output written to `syslog` (what type of program is sending the message). This variable is unavailable unless the `log_sink_syseventlog` error log component is installed. See Section 5.4.2.7, “Error Logging to the System Log”.

The permitted values can vary per operating system; consult your system `syslog` documentation.

This variable does not exist on Windows.

- `syseventlog.include_pid`

Property	Value
Command-Line Format	`--syseventlog.include-pid[={OFF|ON}]`
Introduced	8.0.13
System Variable	`syseventlog.include_pid`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether to include the server process ID in each line of error log output written to `syslog`. This variable is unavailable unless the `log_sink_syseventlog` error log component is installed. See Section 5.4.2.7, “Error Logging to the System Log”.

This variable does not exist on Windows.

- `syseventlog.tag`

Property	Value
Command-Line Format	`--syseventlog.tag=tag`
Introduced	8.0.13
System Variable	`syseventlog.tag`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`empty string`

The tag to be added to the server identifier in error log output written to `syslog` or the Windows Event Log. This variable is unavailable unless the `log_sink_syseventlog` error log component is installed. See Section 5.4.2.7, “Error Logging to the System Log”.

By default, no tag is set, so the server identifier is simply `MySQL` on Windows, and `mysqld` on other platforms. If a tag value of *`tag`* is specified, it is appended to the server identifier with a leading hyphen, resulting in a `syslog` identifier of `mysqld-`*`tag`* (or `MySQL-`*`tag`* on Windows).

On Windows, to use a tag that does not already exist, the server must be run from an account with Administrator privileges, to permit creation of a registry entry for the tag. Elevated privileges are not required if the tag already exists.

- `system_time_zone`

Property	Value
System Variable	`system_time_zone`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The server system time zone. When the server begins executing, it inherits a time zone setting from the machine defaults, possibly modified by the environment of the account used for running the server or the startup script. The value is used to set `system_time_zone`. Typically the time zone is specified by the `TZ` environment variable. It also can be specified using the `--timezone` option of the `mysqld_safe` script.

The `system_time_zone` variable differs from `time_zone`. Although they might have the same value, the latter variable is used to initialize the time zone for each client that connects. See Section 5.1.13, “MySQL Server Time Zone Support”.

- `table_definition_cache`

Property	Value
Command-Line Format	`--table-definition-cache=#`
System Variable	`table_definition_cache`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`400`
Maximum Value	`524288`

The number of table definitions that can be stored in the definition cache. If you use a large number of tables, you can create a large table definition cache to speed up opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the normal table cache. The minimum value is 400. The default value is based on the following formula, capped to a limit of 2000:

```
MIN(400 + table_open_cache / 2, 2000)
```

For InnoDB, table_definition_cache acts as a soft limit for the number of open table instances in the InnoDB data dictionary cache. If the number of open table instances exceeds the table_definition_cache setting, the LRU mechanism begins to mark table instances for eviction and eventually removes them from the data dictionary cache. The limit helps address situations in which significant amounts of memory would be used to cache rarely used table instances until the next server restart. The number of table instances with cached metadata could be higher than the limit defined by table_definition_cache, because parent and child table instances with foreign key relationships are not placed on the LRU list and are not subject to eviction from memory.

Additionally, table_definition_cache defines a soft limit for the number of InnoDB file-per-table tablespaces that can be open at one time, which is also controlled by innodb_open_files. If both table_definition_cache and innodb_open_files are set, the highest setting is used. If neither variable is set, table_definition_cache, which has a higher default value, is used. If the number of open tablespace file handles exceeds the limit defined by table_definition_cache or innodb_open_files, the LRU mechanism searches the tablespace file LRU list for files that are fully flushed and are not currently being extended. This process is performed each time a new tablespace is opened. If there are no “inactive” tablespaces, no tablespace files are closed.

The table definition cache exists in parallel with the table definition cache partition of the dictionary object cache. Both caches store table definitions but serve different parts of the MySQL server. Objects in one cache have no dependence on the existence objects in the other. For more information, see Section 14.4, “Dictionary Object Cache”.

- table_encryption_privilege_check

Property	Value
Command-Line Format	--table-encryption-privilege-check[={OFF|ON}]
Introduced	8.0.16
System Variable	table_encryption_privilege_check
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Controls the TABLE_ENCRYPTION_ADMIN privilege check that occurs when creating or altering a schema or general tablespace with encryption that differs from the default_table_encryption setting, or when creating or altering a table with an encryption setting that differs from the default schema encryption. The check is disabled by default.

Setting table_encryption_privilege_check at runtime requires the SUPER privilege.

table_encryption_privilege_check supports SET PERSIST and SET PERSIST_ONLY syntax. See Section 5.1.9.3, “Persisted System Variables”.

For more information, see Defining an Encryption Default for Schemas and General Tablespaces.

- table_open_cache

Property	Value
Command-Line Format	`--table-open-cache=#`
System Variable	`table_open_cache`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.4)	`4000`
Default Value (<= 8.0.3)	`2000`
Minimum Value	`1`
Maximum Value	`524288`

The number of open tables for all threads. Increasing this value increases the number of file descriptors that `mysqld` requires. You can check whether you need to increase the table cache by checking the `Opened_tables` status variable. See Section 5.1.10, “Server Status Variables”. If the value of `Opened_tables` is large and you do not use `FLUSH TABLES` often (which just forces all tables to be closed and reopened), then you should increase the value of the `table_open_cache` variable. For more information about the table cache, see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.

- `table_open_cache_instances`

Property	Value
Command-Line Format	`--table-open-cache-instances=#`
System Variable	`table_open_cache_instances`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`16`
Minimum Value	`1`
Maximum Value	`64`

The number of open tables cache instances. To improve scalability by reducing contention among sessions, the open tables cache can be partitioned into several smaller cache instances of size `table_open_cache` / `table_open_cache_instances` . A session needs to lock only one instance to access it for DML statements. This segments cache access among instances, permitting higher performance for operations that use the cache when there are many sessions accessing tables. (DDL statements still require a lock on the entire cache, but such statements are much less frequent than DML statements.)

A value of 8 or 16 is recommended on systems that routinely use 16 or more cores.

- `tablespace_definition_cache`

Property	Value
Command-Line Format	`--tablespace-definition-cache=#`

Property	Value
System Variable	tablespace_definition_cache
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	256
Minimum Value	256
Maximum Value	524288

Defines a limit for the number of tablespace definition objects, both used and unused, that can be kept in the dictionary object cache.

Unused tablespace definition objects are only kept in the dictionary object cache when the number in use is less than the capacity defined by tablespace_definition_cache.

A setting of 0 means that tablespace definition objects are only kept in the dictionary object cache while they are in use.

For more information, see Section 14.4, “Dictionary Object Cache”.

- temptable_max_ram

Property	Value
Command-Line Format	--temptable-max-ram=#
Introduced	8.0.2
System Variable	temptable_max_ram
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1073741824
Minimum Value	2097152
Maximum Value	2^64-1

Defines the maximum amount of memory that can be occupied by the TempTable storage engine before it starts storing data on disk. The default value is 1073741824 bytes (1GiB). For more information, see Section 8.4.4, “Internal Temporary Table Use in MySQL”.

- temptable_use_mmap

Property	Value
Command-Line Format	--temptable-use-mmap[={OFF|ON}]
Introduced	8.0.16
System Variable	temptable_use_mmap
Scope	Global

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

Defines whether the TempTable storage engine allocates space for internal in-memory temporary tables as memory-mapped temporary files when the amount of memory occupied by the TempTable storage engine exceeds the limit defined by the temptable_max_ram variable. When temptable_use_mmap is disabled, the TempTable storage engine uses InnoDB on-disk internal temporary tables instead. For more information, see Section 8.4.4, “Internal Temporary Table Use in MySQL”.

- thread_cache_size

Property	Value
Command-Line Format	--thread-cache-size=#
System Variable	thread_cache_size
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)
Minimum Value	0
Maximum Value	16384

How many threads the server should cache for reuse. When a client disconnects, the client's threads are put in the cache if there are fewer than thread_cache_size threads there. Requests for threads are satisfied by reusing threads taken from the cache if possible, and only when the cache is empty is a new thread created. This variable can be increased to improve performance if you have a lot of new connections. Normally, this does not provide a notable performance improvement if you have a good thread implementation. However, if your server sees hundreds of connections per second you should normally set thread_cache_size high enough so that most new connections use cached threads. By examining the difference between the Connections and Threads_created status variables, you can see how efficient the thread cache is. For details, see Section 5.1.10, “Server Status Variables”.

The default value is based on the following formula, capped to a limit of 100:

```
8 + (max_connections / 100)
```

- thread_handling

Property	Value
Command-Line Format	--thread-handling=name
System Variable	thread_handling
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

Property	Value
Type	Enumeration
Default Value	`one-thread-per-connection`

Valid Values	`no-threads`
	`one-thread-per-connection`
	`loaded-dynamically`

The thread-handling model used by the server for connection threads. The permissible values are `no-threads` (the server uses a single thread to handle one connection) and `one-thread-per-connection` (the server uses one thread to handle each client connection). `no-threads` is useful for debugging under Linux; see Section 29.5, “Debugging and Porting MySQL”.

- `thread_pool_algorithm`

Property	Value
Command-Line Format	`--thread-pool-algorithm=#`
Introduced	8.0.11
System Variable	`thread_pool_algorithm`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1`

This variable controls which algorithm the thread pool plugin uses:

- A value of 0 (the default) uses a conservative low-concurrency algorithm which is most well tested and is known to produce very good results.
- A value of 1 increases the concurrency and uses a more aggressive algorithm which at times has been known to perform 5–10% better on optimal thread counts, but has degrading performance as the number of connections increases. Its use should be considered as experimental and not supported.

This variable is available only if the thread pool plugin is enabled. See Section 5.6.3, “MySQL Enterprise Thread Pool”

- `thread_pool_high_priority_connection`

Property	Value
Command-Line Format	`--thread-pool-high-priority-connection=#`
Introduced	8.0.11
System Variable	`thread_pool_high_priority_connection`
Scope	Global, Session
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	1

This variable affects queuing of new statements prior to execution. If the value is 0 (false, the default), statement queuing uses both the low-priority and high-priority queues. If the value is 1 (true), queued statements always go to the high-priority queue.

This variable is available only if the thread pool plugin is enabled. See Section 5.6.3, “MySQL Enterprise Thread Pool”

- thread_pool_max_active_query_threads

Property	Value
Command-Line Format	--thread-pool-max-active-query-threads
Introduced	8.0.19
System Variable	thread_pool_max_active_query_threads
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	512

The maximum permissible number of active (running) query threads per group. If the value is 0, the thread pool plugin uses up to as many threads as are available.

- thread_pool_max_unused_threads

Property	Value
Command-Line Format	--thread-pool-max-unused-threads=#
Introduced	8.0.11
System Variable	thread_pool_max_unused_threads
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	4096

The maximum permitted number of unused threads in the thread pool. This variable makes it possible to limit the amount of memory used by sleeping threads.

A value of 0 (the default) means no limit on the number of sleeping threads. A value of *N* where *N* is greater than 0 means 1 consumer thread and *N*−1 reserve threads. In this case, if a thread is ready to sleep but the number of sleeping threads is already at the maximum, the thread exits rather than going to sleep.

A sleeping thread is either sleeping as a consumer thread or a reserve thread. The thread pool permits one thread to be the consumer thread when sleeping. If a thread goes to sleep and there is no existing consumer thread, it will sleep as a consumer thread. When a thread must be woken up, a consumer thread is selected if there is one. A reserve thread is selected only when there is no consumer thread to wake up.

This variable is available only if the thread pool plugin is enabled. See Section 5.6.3, “MySQL Enterprise Thread Pool”

- `thread_pool_prio_kickup_timer`

Property	Value
Command-Line Format	`--thread-pool-prio-kickup-timer=#`
Introduced	8.0.11
System Variable	`thread_pool_prio_kickup_timer`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1000`
Minimum Value	`0`
Maximum Value	`4294967294`

This variable affects statements waiting for execution in the low-priority queue. The value is the number of milliseconds before a waiting statement is moved to the high-priority queue. The default is 1000 (1 second). The range of values is 0 to $2^{32} - 2$.

This variable is available only if the thread pool plugin is enabled. See Section 5.6.3, “MySQL Enterprise Thread Pool”

- `thread_pool_size`

Property	Value
Command-Line Format	`--thread-pool-size=#`
Introduced	8.0.11
System Variable	`thread_pool_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	`16`
Minimum Value	`1`
Maximum Value (>= 8.0.19)	`512`
Maximum Value (<= 8.0.18)	`64`

The number of thread groups in the thread pool. This is the most important parameter controlling thread pool performance. It affects how many statements can execute simultaneously. The default value is 16, with a range from 1 to 64 of permissible values. If a value outside this range is specified, the thread pool plugin does not load and the server writes a message to the error log.

This variable is available only if the thread pool plugin is enabled. See Section 5.6.3, “MySQL Enterprise Thread Pool”

- `thread_pool_stall_limit`

Property	Value
Command-Line Format	`--thread-pool-stall-limit=#`
Introduced	8.0.11
System Variable	`thread_pool_stall_limit`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`6`
Minimum Value	`4`
Maximum Value	`600`

This variable affects executing statements. The value is the amount of time a statement has to finish after starting to execute before it becomes defined as stalled, at which point the thread pool permits the thread group to begin executing another statement. The value is measured in 10 millisecond units, so a value of 6 (the default) means 60ms. The range of values is 4 to 600 (40ms to 6s). Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that include long-running statements, to avoid starting too many new statements while the current ones execute.

This variable is available only if the thread pool plugin is enabled. See Section 5.6.3, “MySQL Enterprise Thread Pool”

- `thread_stack`

Property	Value
Command-Line Format	`--thread-stack=#`
System Variable	`thread_stack`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value (64-bit platforms, >= 8.0.2)	286720
Default Value (64-bit platforms, <= 8.0.1)	262144
Default Value (32-bit platforms, >= 8.0.2)	221184
Default Value (32-bit platforms, <= 8.0.1)	196608
Minimum Value	131072
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295
Block Size	1024

The stack size for each thread. The default is large enough for normal operation. If the thread stack size is too small, it limits the complexity of the SQL statements that the server can handle, the recursion depth of stored procedures, and other memory-consuming actions.

- `time_format`

  This system variable was removed in MySQL 8.0.3.

- `time_zone`

Property	Value
System Variable	time_zone
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies (>= 8.0.17)	Yes
SET_VAR Hint Applies (<= 8.0.16)	No
Type	String

The current time zone. This variable is used to initialize the time zone for each client that connects. By default, the initial value of this is `'SYSTEM'` (which means, “use the value of `system_time_zone`”). The value can be specified explicitly at server startup with the `--default-time-zone` option. See Section 5.1.13, “MySQL Server Time Zone Support”.

Note

If set to `SYSTEM`, every MySQL function call that requires a time zone calculation makes a system library call to determine the current system time zone. This call may be protected by a global mutex, resulting in contention.

- `timestamp`

Property	Value
System Variable	`timestamp`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Numeric

Set the time for this client. This is used to get the original timestamp if you use the binary log to restore rows. *`timestamp_value`* should be a Unix epoch timestamp (a value like that returned by `UNIX_TIMESTAMP()`, not a value in `'YYYY-MM-DD hh:mm:ss'` format) or `DEFAULT`.

Setting `timestamp` to a constant value causes it to retain that value until it is changed again. Setting `timestamp` to `DEFAULT` causes its value to be the current date and time as of the time it is accessed.

In MySQL 8.0, `timestamp` is a `DOUBLE` rather than `BIGINT` because its value includes a microseconds part.

`SET timestamp` affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`. The server can be started with the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`, in which case `SET timestamp` affects both functions.

- `tls_ciphersuites`

Property	Value
Command-Line Format	`--tls-ciphersuites=ciphersuite_list`
Introduced	8.0.16
System Variable	`tls_ciphersuites`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`empty string`

Which ciphersuites the server permits for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this variable depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `tls_version`

Property	Value
Command-Line Format	`--tls-version=protocol_list`
System Variable	`tls_version`
Scope	Global
Dynamic (>= 8.0.16)	Yes

Property	Value
Dynamic (<= 8.0.15)	No
SET_VAR Hint Applies	No
Type	String
Default Value (>= 8.0.16)	`TLSv1,TLSv1.1,TLSv1.2,TLSv1.3` (OpenSSL 1.1.1 and higher) `TLSv1,TLSv1.1,TLSv1.2` (otherwise)
Default Value (<= 8.0.15)	`TLSv1,TLSv1.1,TLSv1.2`

Which protocols the server permits for encrypted connections. The value is a list of one or more comma-separated protocol names. The protocols that can be named for this variable depend on the SSL library used to compile MySQL. Permitted protocols should be chosen such as not to leave “holes” in the list. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

As of MySQL 8.0.16, this variable is dynamic and can be modified at runtime to affect the SSL context the server uses for new connections. See Server-Side Runtime Configuration for Encrypted Connections. Prior to MySQL 8.0.16, this variable can be set only at server startup.

- `tmp_table_size`

Property	Value
Command-Line Format	`--tmp-table-size=#`
System Variable	`tmp_table_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`16777216`
Minimum Value	`1024`
Maximum Value	`18446744073709551615`

The maximum size of internal in-memory temporary tables. This variable does not apply to user-created `MEMORY` tables.

The actual limit is determined from whichever of the values of `tmp_table_size` and `max_heap_table_size` is smaller. If an in-memory temporary table exceeds the limit, MySQL automatically converts it to an on-disk temporary table. See Storage Engine for On-Disk Internal Temporary Tables, for more information.

Increase the value of `tmp_table_size` (and `max_heap_table_size` if necessary) if you do many advanced `GROUP BY` queries and you have lots of memory.

You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` and `Created_tmp_tables` variables.

See also Section 8.4.4, “Internal Temporary Table Use in MySQL”.

- `tmpdir`

Property	Value
Command-Line Format	`--tmpdir=dir_name`
System Variable	`tmpdir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

The path of the directory to use for creating temporary files. It might be useful if your default `/tmp` directory resides on a partition that is too small to hold temporary tables. This variable can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (`:`) on Unix and semicolon characters (`;`) on Windows.

`tmpdir` can be a non-permanent location, such as a directory on a memory-based file system or a directory that is cleared when the server host restarts. If the MySQL server is acting as a replication slave, and you are using a non-permanent location for `tmpdir`, consider setting a different temporary directory for the slave using the `slave_load_tmpdir` variable. For a replication slave, the temporary files used to replicate `LOAD DATA` statements are stored in this directory, so with a permanent location they can survive machine restarts, although replication can now continue after a restart if the temporary files have been removed.

For more information about the storage location of temporary files, see Section B.4.3.5, “Where MySQL Stores Temporary Files”.

- `transaction_alloc_block_size`

Property	Value
Command-Line Format	`--transaction-alloc-block-size=#`
System Variable	`transaction_alloc_block_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`8192`
Minimum Value	`1024`
Maximum Value	`131072`
Block Size	`1024`

The amount in bytes by which to increase a per-transaction memory pool which needs memory. See the description of `transaction_prealloc_size`.

- `transaction_isolation`

Property	Value
Command-Line Format	`--transaction-isolation=name`
System Variable	`transaction_isolation`

Property	Value
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	REPEATABLE-READ
Valid Values	READ-UNCOMMITTED READ-COMMITTED REPEATABLE-READ SERIALIZABLE

The transaction isolation level. The default is REPEATABLE-READ.

The transaction isolation level has three scopes: global, session, and next transaction. This three-scope implementation leads to some nonstandard isolation-level assignment semantics, as described later.

To set the global transaction isolation level at startup, use the --transaction-isolation server option.

At runtime, the isolation level can be set directly using the SET statement to assign a value to the transaction_isolation system variable, or indirectly using the SET TRANSACTION statement. If you set transaction_isolation directly to an isolation level name that contains a space, the name should be enclosed within quotation marks, with the space replaced by a dash. For example, use this SET statement to set the global value:

```
SET GLOBAL transaction_isolation = 'READ-COMMITTED';
```

Setting the global transaction_isolation value sets the isolation level for all subsequent sessions. Existing sessions are unaffected.

To set the session or next-level transaction_isolation value, use the SET statement. For most session system variables, these statements are equivalent ways to set the value:

```
SET @@SESSION.var_name = value;
SET SESSION var_name = value;
SET var_name = value;
SET @@var_name = value;
```

As mentioned previously, the transaction isolation level has a next-transaction scope, in addition to the global and session scopes. To enable the next-transaction scope to be set, SET syntax for assigning session system variable values has nonstandard semantics for transaction_isolation:

- To set the session isolation level, use any of these syntaxes:

  ```
  SET @@SESSION.transaction_isolation = value;
  SET SESSION transaction_isolation = value;
  SET transaction_isolation = value;
  ```

  For each of those syntaxes, these semantics apply:

- Sets the isolation level for all subsequent transactions performed within the session.
- Permitted within transactions, but does not affect the current ongoing transaction.
- If executed between transactions, overrides any preceding statement that sets the next-transaction isolation level.
- Corresponds to `SET SESSION TRANSACTION ISOLATION LEVEL` (with the `SESSION` keyword).

- To set the next-transaction isolation level, use this syntax:

```
SET @@transaction_isolation = value;
```

For that syntax, these semantics apply:

- Sets the isolation level only for the next single transaction performed within the session.
- Subsequent transactions revert to the session isolation level.
- Not permitted within transactions.
- Corresponds to `SET TRANSACTION ISOLATION LEVEL` (without the `SESSION` keyword).

For more information about `SET TRANSACTION` and its relationship to the `transaction_isolation` system variable, see Section 13.3.7, “SET TRANSACTION Syntax”.

- `transaction_prealloc_size`

Property	Value
Command-Line Format	`--transaction-prealloc-size=#`
System Variable	`transaction_prealloc_size`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`4096`
Minimum Value	`1024`
Maximum Value	`131072`
Block Size	`1024`

There is a per-transaction memory pool from which various transaction-related allocations take memory. The initial size of the pool in bytes is `transaction_prealloc_size`. For every allocation that cannot be satisfied from the pool because it has insufficient memory available, the pool is increased by `transaction_alloc_block_size` bytes. When the transaction ends, the pool is truncated to `transaction_prealloc_size` bytes.

By making `transaction_prealloc_size` sufficiently large to contain all statements within a single transaction, you can avoid many `malloc()` calls.

- `transaction_read_only`

Property	Value
Command-Line Format	`--transaction-read-only[={OFF|ON}]`
System Variable	`transaction_read_only`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

The transaction access mode. The value can be `OFF` (read/write; the default) or `ON` (read only).

The transaction access mode has three scopes: global, session, and next transaction. This three-scope implementation leads to some nonstandard access-mode assignment semantics, as described later.

To set the global transaction access mode at startup, use the `--transaction-read-only` server option.

At runtime, the access mode can be set directly using the `SET` statement to assign a value to the `transaction_read_only` system variable, or indirectly using the `SET TRANSACTION` statement. For example, use this `SET` statement to set the global value:

```
SET GLOBAL transaction_read_only = ON;
```

Setting the global `transaction_read_only` value sets the access mode for all subsequent sessions. Existing sessions are unaffected.

To set the session or next-level `transaction_read_only` value, use the `SET` statement. For most session system variables, these statements are equivalent ways to set the value:

```
SET @@SESSION.var_name = value;
SET SESSION var_name = value;
SET var_name = value;
SET @@var_name = value;
```

As mentioned previously, the transaction access mode has a next-transaction scope, in addition to the global and session scopes. To enable the next-transaction scope to be set, `SET` syntax for assigning session system variable values has nonstandard semantics for `transaction_read_only`,

- To set the session access mode, use any of these syntaxes:

  ```
  SET @@SESSION.transaction_read_only = value;
  SET SESSION transaction_read_only = value;
  SET transaction_read_only = value;
  ```

  For each of those syntaxes, these semantics apply:

  - Sets the access mode for all subsequent transactions performed within the session.
  - Permitted within transactions, but does not affect the current ongoing transaction.

- If executed between transactions, overrides any preceding statement that sets the next-transaction access mode.
- Corresponds to `SET SESSION TRANSACTION {READ WRITE | READ ONLY}` (with the `SESSION` keyword).

- To set the next-transaction access mode, use this syntax:

```
SET @@transaction_read_only = value;
```

For that syntax, these semantics apply:

- Sets the access mode only for the next single transaction performed within the session.
- Subsequent transactions revert to the session access mode.
- Not permitted within transactions.
- Corresponds to `SET TRANSACTION {READ WRITE | READ ONLY}` (without the `SESSION` keyword).

For more information about `SET TRANSACTION` and its relationship to the `transaction_read_only` system variable, see Section 13.3.7, “SET TRANSACTION Syntax”.

- `tx_isolation`

Property	Value
Deprecated	Yes (removed in 8.0.3)
System Variable	`tx_isolation`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`REPEATABLE-READ`

Valid Values	`READ-UNCOMMITTED`
	`READ-COMMITTED`
	`REPEATABLE-READ`
	`SERIALIZABLE`

This system variable was removed in MySQL 8.0.3. Use `transaction_isolation` instead.

- `tx_read_only`

Property	Value
Deprecated	Yes (removed in 8.0.3)
System Variable	`tx_read_only`
Scope	Global, Session

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

This system variable was removed in MySQL 8.0.3. Use transaction_read_only instead.

- unique_checks

Property	Value
System Variable	unique_checks
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	ON

If set to 1 (the default), uniqueness checks for secondary indexes in InnoDB tables are performed. If set to 0, storage engines are permitted to assume that duplicate keys are not present in input data. If you know for certain that your data does not contain uniqueness violations, you can set this to 0 to speed up large table imports to InnoDB.

Setting this variable to 0 does not *require* storage engines to ignore duplicate keys. An engine is still permitted to check for them and issue duplicate-key errors if it detects them.

- updatable_views_with_limit

Property	Value
Command-Line Format	--updatable-views-with-limit[={OFF|ON}]
System Variable	updatable_views_with_limit
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	1

This variable controls whether updates to a view can be made when the view does not contain all columns of the primary key defined in the underlying table, if the update statement contains a LIMIT clause. (Such updates often are generated by GUI tools.) An update is an UPDATE or DELETE statement. Primary key here means a PRIMARY KEY, or a UNIQUE index in which no column can contain NULL.

The variable can have two values:

- 1 or YES: Issue a warning only (not an error message). This is the default value.
- 0 or NO: Prohibit the update.

- use_secondary_engine

Property	Value
Introduced	8.0.13
System Variable	use_secondary_engine
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Enumeration
Default Value	ON
Valid Values	OFF ON FORCED

For future use.

- validate_password.*xxx*

  The validate_password component implements a set of system variables having names of the form validate_password.*xxx*. These variables affect password testing by that component; see Section 6.4.3.2, “Password Validation Options and Variables”.

- validate_user_plugins

Property	Value
Command-Line Format	--validate-user-plugins[={OFF|ON}]
System Variable	validate_user_plugins
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

  If this variable is enabled (the default), the server checks each user account and produces a warning if conditions are found that would make the account unusable:

  - The account requires an authentication plugin that is not loaded.
  - The account requires the sha256_password or caching_sha2_password authentication plugin but the server was started with neither SSL nor RSA enabled as required by the plugin.

  Enabling validate_user_plugins slows down server initialization and FLUSH PRIVILEGES. If you do not require the additional checking, you can disable this variable at startup to avoid the performance decrement.

- version

  The version number for the server. The value might also include a suffix indicating server build or configuration information. -debug indicates that the server was built with debugging support enabled.

- `version_comment`

Property	Value
System Variable	`version_comment`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The `CMake` configuration program has a `COMPILATION_COMMENT_SERVER` option that permits a comment to be specified when building MySQL. This variable contains the value of that comment. (Prior to MySQL 8.0.14, `version_comment` is set by the `COMPILATION_COMMENT` option.) See Section 2.9.7, “MySQL Source-Configuration Options”.

- `version_compile_machine`

Property	Value
System Variable	`version_compile_machine`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The type of the server binary.

- `version_compile_os`

Property	Value
System Variable	`version_compile_os`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The type of operating system on which MySQL was built.

- `version_compile_zlib`

Property	Value
Introduced	8.0.11
System Variable	`version_compile_zlib`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The version of the compiled-in `zlib` library.

- `wait_timeout`

Property	Value
Command-Line Format	`--wait-timeout=#`
System Variable	`wait_timeout`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`28800`
Minimum Value	`1`
Maximum Value (Other)	`31536000`
Maximum Value (Windows)	`2147483`

The number of seconds the server waits for activity on a noninteractive connection before closing it.

On thread startup, the session `wait_timeout` value is initialized from the global `wait_timeout` value or from the global `interactive_timeout` value, depending on the type of client (as defined by the `CLIENT_INTERACTIVE` connect option to `mysql_real_connect()`). See also `interactive_timeout`.

- `warning_count`

The number of errors, warnings, and notes that resulted from the last statement that generated messages. This variable is read only. See Section 13.7.7.40, “SHOW WARNINGS Syntax”.

- `windowing_use_high_precision`

Property	Value
Command-Line Format	`--windowing-use-high-precision[={OFF|ON}]`
Introduced	8.0.2
System Variable	`windowing_use_high_precision`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Boolean
Default Value	`ON`

Whether to compute window operations without loss of precision. See Section 8.2.1.21, “Window Function Optimization”.

5.1.9 Using System Variables

The MySQL server maintains many system variables that configure its operation. Section 5.1.8, “Server System Variables”, describes the meaning of these variables. Each system variable has a default value.

System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can also use system variable values in expressions.

Many system variables are built in. System variables may also be installed by server plugins or components:

- System variables implemented by a server plugin are exposed when the plugin is installed and have names that begin with the plugin name. For example, the `audit_log` plugin implements a system variable named `audit_log_policy`.
- System variables implemented by a server component are exposed when the component is installed and have names that begin with a component-specific prefix. For example, the `log_filter_dragnet` error log filter component implements a system variable named `log_error_filter_rules`, the full name of which is `dragnet.log_error_filter_rules`. To refer to this variable, use the full name.

There are two scopes in which system variables exist. Global variables affect the overall operation of the server. Session variables affect its operation for individual client connections. A given system variable can have both a global and a session value. Global and session system variables are related as follows:

- When the server starts, it initializes each global variable to its default value. These defaults can be changed by options specified on the command line or in an option file. (See Section 4.2.2, “Specifying Program Options”.)
- The server also maintains a set of session variables for each client that connects. The client's session variables are initialized at connect time using the current values of the corresponding global variables. For example, a client's SQL mode is controlled by the session `sql_mode` value, which is initialized when the client connects to the value of the global `sql_mode` value.

  For some system variables, the session value is not initialized from the corresponding global value; if so, that is indicated in the variable description.

System variable values can be set globally at server startup by using options on the command line or in an option file. At startup, the syntax for system variables is the same as for command options, so within variable names, dashes and underscores may be used interchangeably. For example, `--general_log=ON` and `--general-log=ON` are equivalent.

When you use a startup option to set a variable that takes a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3; that is, units of kilobytes, megabytes, or gigabytes, respectively. As of MySQL 8.0.14, a suffix can also be `T`, `P`, and `E` to indicate a multiplier of 1024^4, 1024^5 or 1024^6. Thus, the following command starts the server with an `InnoDB` log file size of 16 megabytes and a maximum packet size of one gigabyte:

```
mysqld --innodb-log-file-size=16M --max-allowed-packet=1G
```

Within an option file, those variables are set like this:

```
[mysqld]
innodb_log_file_size=16M
max_allowed_packet=1G
```

The lettercase of suffix letters does not matter; `16M` and `16m` are equivalent, as are `1G` and `1g`.

To restrict the maximum value to which a system variable can be set at runtime with the `SET` statement, specify this maximum by using an option of the form `--maximum-`*`var_name`*`=`*`value`* at server startup. For

example, to prevent the value of `innodb_log_file_size` from being increased to more than 32MB at runtime, use the option `--maximum-innodb-log-file-size=32M`.

Many system variables are dynamic and can be changed at runtime by using the `SET` statement. For a list, see Section 5.1.9.2, “Dynamic System Variables”. To change a system variable with `SET`, refer to it by name, optionally preceded by a modifier. At runtime, system variable names must be written using underscores, not dashes. The following examples briefly illustrate this syntax:

- Set a global system variable:

```
SET GLOBAL max_connections = 1000;
SET @@GLOBAL.max_connections = 1000;
```

- Persist a global system variable to the `mysqld-auto.cnf` file (and set the runtime value):

```
SET PERSIST max_connections = 1000;
SET @@PERSIST.max_connections = 1000;
```

- Persist a global system variable to the `mysqld-auto.cnf` file (without setting the runtime value):

```
SET PERSIST_ONLY back_log = 1000;
SET @@PERSIST_ONLY.back_log = 1000;
```

- Set a session system variable:

```
SET SESSION sql_mode = 'TRADITIONAL';
SET @@SESSION.sql_mode = 'TRADITIONAL';
SET @@sql_mode = 'TRADITIONAL';
```

For complete details about `SET` syntax, see Section 13.7.6.1, “SET Syntax for Variable Assignment”. For a description of the privilege requirements for setting and persisting system variables, see Section 5.1.9.1, “System Variable Privileges”

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

Note

Some system variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. However, to set such a variable on the command line or in an option file, you must set it to `1` or `0`; setting it to `ON` or `OFF` will not work. For example, on the command line, `--delay_key_write=1` works but `--delay_key_write=ON` does not.

To display system variable names and values, use the `SHOW VARIABLES` statement:

```
mysql> SHOW VARIABLES;
+---------------------------------+-----------------------------------+
| Variable_name                   | Value                             |
+---------------------------------+-----------------------------------+
| auto_increment_increment        | 1                                 |
| auto_increment_offset           | 1                                 |
| automatic_sp_privileges         | ON                                |
| back_log                        | 151                               |
| basedir                         | /home/mysql/                      |
| binlog_cache_size               | 32768                             |
| bulk_insert_buffer_size         | 8388608                           |
| character_set_client            | utf8                              |
| character_set_connection        | utf8                              |
| character_set_database          | utf8mb4                           |
| character_set_filesystem        | binary                            |
| character_set_results           | utf8                              |
| character_set_server            | utf8mb4                           |
| character_set_system            | utf8                              |
| character_sets_dir              | /home/mysql/share/mysql/charsets/ |
| collation_connection            | utf8_general_ci                   |
| collation_database              | utf8mb4_0900_ai_ci                |
| collation_server                | utf8mb4_0900_ai_ci                |
...
| innodb_autoextend_increment     | 8                                 |
| innodb_buffer_pool_size         | 8388608                           |
| innodb_commit_concurrency       | 0                                 |
| innodb_concurrency_tickets      | 500                               |
| innodb_data_file_path           | ibdata1:10M:autoextend            |
| innodb_data_home_dir            |                                   |
...
| version                         | 8.0.1-dmr-log                     |
| version_comment                 | Source distribution               |
| version_compile_machine         | i686                              |
| version_compile_os              | suse-linux                        |
| wait_timeout                    | 28800                             |
+---------------------------------+-----------------------------------+
```

With a `LIKE` clause, the statement displays only those variables that match the pattern. To obtain a specific variable name, use a `LIKE` clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the `%` wildcard character in a `LIKE` clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because `_` is a wildcard that matches any single character, you should escape it as `\_` to match it literally. In practice, this is rarely necessary.

For `SHOW VARIABLES`, if you specify neither `GLOBAL` nor `SESSION`, MySQL returns `SESSION` values.

The reason for requiring the `GLOBAL` keyword when setting `GLOBAL`-only variables but not when retrieving them is to prevent problems in the future:

- Were a `SESSION` variable to be removed that has the same name as a `GLOBAL` variable, a client with privileges sufficient to modify global variables might accidentally change the `GLOBAL` variable rather than just the `SESSION` variable for its own session.
- Were a `SESSION` variable to be added with the same name as a `GLOBAL` variable, a client that intends to change the `GLOBAL` variable might find only its own `SESSION` variable changed.

5.1.9.1 System Variable Privileges

A system variable can have a global value that affects server operation as a whole, a session value that affects only the current session, or both:

- For dynamic system variables, the `SET` statement can be used to change their global or session runtime value (or both), to affect operation of the current server instance. (For information about dynamic variables, see Section 5.1.9.2, “Dynamic System Variables”.)
- For certain global system variables, `SET` can be used to persist their value to the `mysqld-auto.cnf` file in the data directory, to affect server operation for subsequent startups. (For information about persisting system variables and the `mysqld-auto.cnf` file, see Section 5.1.9.3, “Persisted System Variables”.)
- For persisted global system variables, `RESET PERSIST` can be used to remove their value from `mysqld-auto.cnf`, to affect server operation for subsequent startups.

This section describes the privileges required for operations that assign values to system variables at runtime. This includes operations that affect runtime values, and operations that persist values.

To set a global system variable, use a `SET` statement with the appropriate keyword. These privileges apply:

- To set a global system variable runtime value, use the `SET GLOBAL` statement, which requires the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege.
- To persist a global system variable to the `mysqld-auto.cnf` file (and set the runtime value), use the `SET PERSIST` statement, which requires the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege.
- To persist a global system variable to the `mysqld-auto.cnf` file (without setting the runtime value), use the `SET PERSIST_ONLY` statement, which requires the `SYSTEM_VARIABLES_ADMIN` and `PERSIST_RO_VARIABLES_ADMIN` privileges. `SET PERSIST_ONLY` can be used for both dynamic and read-only system variables, but is particularly useful for persisting read-only variables, for which `SET PERSIST` cannot be used.
- Some global system variables are persist-restricted (see Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”). To persist these variables, use the `SET PERSIST_ONLY` statement, which requires the privileges described previously. In addition, you must connect to the server using an encrypted connection and supply an SSL certificate with the Subject value specified by the `persist_only_admin_x509_subject` system variable.

To remove a persisted global system variable from the `mysqld-auto.cnf` file, use the `RESET PERSIST` statement. These privileges apply:

- For dynamic system variables, `RESET PERSIST` requires the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege.
- For read-only system variables, `RESET PERSIST` requires the `SYSTEM_VARIABLES_ADMIN` and `PERSIST_RO_VARIABLES_ADMIN` privileges.
- For persist-restricted variables, `RESET PERSIST` does not require an encrypted connection to the server made using a particular SSL certificate.

If a global system variable has any exceptions to the preceding privilege requirements, the variable description indicates those exceptions. Examples include `default_table_encryption` and `mandatory_roles`, which require additional privileges. These additional privileges apply to operations that set the global runtime value, but not operations that persist the value.

To set a session system variable runtime value, use the `SET SESSION` statement. In contrast to setting global runtime values, setting session runtime values normally requires no special privileges and can be

done by any user to affect the current session. For some system variables, setting the session value may have effects outside the current session and thus is a restricted operation that can be done only by users who have a special privilege:

- As of MySQL 8.0.14, the privilege required is SESSION_VARIABLES_ADMIN.

Note

Any user who has SYSTEM_VARIABLES_ADMIN or SUPER effectively has SESSION_VARIABLES_ADMIN by implication and need not be granted SESSION_VARIABLES_ADMIN explicitly.

- Prior to MySQL 8.0.14, the privilege required is SYSTEM_VARIABLES_ADMIN or SUPER.

If a session system variable is restricted, the variable description indicates that restriction. Examples include binlog_format and sql_log_bin. Setting the session value of these variables affects binary logging for the current session, but may also have wider implications for the integrity of server replication and backups.

SESSION_VARIABLES_ADMIN enables administrators to minimize the privilege footprint of users who may previously have been granted SYSTEM_VARIABLES_ADMIN or SUPER for the purpose of enabling them to modify restricted session system variables. Suppose that an administrator has created the following role to confer the ability to set restricted session system variables:

```
CREATE ROLE set_session_sysvars;
GRANT SYSTEM_VARIABLES_ADMIN ON *.* TO set_session_sysvars;
```

Any user granted the set_session_sysvars role (and who has that role active) is able to set restricted session system variables. However, that user is also able to set global system variables, which may be undesirable.

By modifying the role to have SESSION_VARIABLES_ADMIN instead of SYSTEM_VARIABLES_ADMIN, the role privileges can be reduced to the ability to set restricted session system variables and nothing else. To modify the role, use these statements:

```
GRANT SESSION_VARIABLES_ADMIN ON *.* TO set_session_sysvars;
REVOKE SYSTEM_VARIABLES_ADMIN ON *.* FROM set_session_sysvars;
```

Modifying the role has an immediate effect: Any account granted the set_session_sysvars role no longer has SYSTEM_VARIABLES_ADMIN and is not able to set global system variables without being granted that ability explicitly. A similar GRANT/REVOKE sequence can be applied to any account that was granted SYSTEM_VARIABLES_ADMIN directly rather than by means of a role.

5.1.9.2 Dynamic System Variables

Many server system variables are dynamic and can be set at runtime. See Section 13.7.6.1, “SET Syntax for Variable Assignment”. For a description of the privilege requirements for setting system variables, see Section 5.1.9.1, “System Variable Privileges”

The following table lists all dynamic system variables applicable within mysqld.

The table lists each variable's data type and scope. The last column indicates whether the scope for each variable is Global, Session, or both. Please see the corresponding item descriptions for details on setting and using the variables. Where appropriate, direct links to further information about the items are provided.

Variables that have a type of “string” take a string value. Variables that have a type of “numeric” take a numeric value. Variables that have a type of “boolean” can be set to 0, 1, ON or OFF. Variables that are

marked as “enumeration” normally should be set to one of the available values for the variable, but can also be set to the number that corresponds to the desired enumeration value. For enumerated system variables, the first enumeration value corresponds to 0. This differs from the `ENUM` data type used for table columns, for which the first enumeration value corresponds to 1.

Table 5.4 Dynamic System Variable Summary

Variable Name	Variable Type	Variable Scope
activate_all_roles_on_login	Boolean	Global
audit_log_connection_policy	Enumeration	Global
audit_log_exclude_accounts	String	Global
audit_log_flush	Boolean	Global
audit_log_include_accounts	String	Global
audit_log_password_history_keep_days	Integer	Global
audit_log_read_buffer_size	Integer	Varies
audit_log_rotate_on_size	Integer	Global
audit_log_statement_policy	Enumeration	Global
authentication_ldap_sasl_auth_method_name	String	Global
authentication_ldap_sasl_bind_base_dn	String	Global
authentication_ldap_sasl_bind_root_dn	String	Global
authentication_ldap_sasl_bind_root_pwd	String	Global
authentication_ldap_sasl_ca_path	String	Global
authentication_ldap_sasl_group_search_attr	String	Global
authentication_ldap_sasl_group_search_filter	String	Global
authentication_ldap_sasl_init_pool_size	Integer	Global
authentication_ldap_sasl_log_status	Integer	Global
authentication_ldap_sasl_max_pool_size	Integer	Global
authentication_ldap_sasl_server_host	String	Global
authentication_ldap_sasl_server_port	Integer	Global
authentication_ldap_sasl_tls	Boolean	Global
authentication_ldap_sasl_user_search_attr	String	Global
authentication_ldap_simple_auth_method_name	String	Global
authentication_ldap_simple_bind_base_dn	String	Global
authentication_ldap_simple_bind_root_dn	String	Global
authentication_ldap_simple_bind_root_pwd	String	Global
authentication_ldap_simple_ca_path	String	Global
authentication_ldap_simple_group_search_attr	String	Global
authentication_ldap_simple_group_search_filter	String	Global
authentication_ldap_simple_init_pool_size	Integer	Global
authentication_ldap_simple_log_status	Integer	Global
authentication_ldap_simple_max_pool_size	Integer	Global
authentication_ldap_simple_server_host	String	Global

Variable Name	Variable Type	Variable Scope
authentication_ldap_simple_server_port	Integer	Global
authentication_ldap_simple_tls	Boolean	Global
authentication_ldap_simple_user_search_attr	String	Global
auto_increment_increment	Integer	Both
auto_increment_offset	Integer	Both
autocommit	Boolean	Both
automatic_sp_privileges	Boolean	Global
avoid_temporal_upgrade	Boolean	Global
big_tables	Boolean	Both
binlog_cache_size	Integer	Global
binlog_checksum	String	Global
binlog_direct_non_transactional_updates	Boolean	Both
binlog_encryption	Boolean	Global
binlog_error_action	Enumeration	Global
binlog_expire_logs_seconds	Integer	Global
binlog_format	Enumeration	Both
binlog_group_commit_sync_delay	Integer	Global
binlog_group_commit_sync_no_delay_count	Integer	Global
binlog_max_flush_queue_time	Integer	Global
binlog_order_commits	Boolean	Global
binlog_row_image	Enumeration	Both
binlog_row_metadata	Enumeration	Global
binlog_row_value_options	Set	Both
binlog_rows_query_log_events	Boolean	Both
binlog_stmt_cache_size	Integer	Global
binlog_transaction_dependency_history_size	Integer	Global
binlog_transaction_dependency_tracking	Enumeration	Global
block_encryption_mode	String	Both
bulk_insert_buffer_size	Integer	Both
character_set_client	String	Both
character_set_connection	String	Both
character_set_database	String	Both
character_set_filesystem	String	Both
character_set_results	String	Both
character_set_server	String	Both
check_proxy_users	Boolean	Global
clone_autotune_concurrency	Boolean	Global
clone_buffer_size	Integer	Global

Variable Name	Variable Type	Variable Scope
clone_ddl_timeout	Integer	Global
clone_enable_compression	Boolean	Global
clone_max_concurrency	Integer	Global
clone_max_data_bandwidth	Integer	Global
clone_max_network_bandwidth	Integer	Global
clone_ssl_ca	File name	Global
clone_ssl_cert	File name	Global
clone_ssl_key	File name	Global
clone_valid_donor_list	String	Global
collation_connection	String	Both
collation_database	String	Both
collation_server	String	Both
completion_type	Enumeration	Both
concurrent_insert	Enumeration	Global
connect_timeout	Integer	Global
connection_control_failed_connections_threshold	Integer	Global
connection_control_max_connection_delay	Integer	Global
connection_control_min_connection_delay	Integer	Global
cte_max_recursion_depth	Integer	Both
debug	String	Both
debug_sync	String	Session
default_collation_for_utf8mb4	Enumeration	Both
default_password_lifetime	Integer	Global
default_storage_engine	Enumeration	Both
default_table_encryption	Boolean	Both
default_tmp_storage_engine	Enumeration	Both
default_week_format	Integer	Both
delay_key_write	Enumeration	Global
delayed_insert_limit	Integer	Global
delayed_insert_timeout	Integer	Global
delayed_queue_size	Integer	Global
div_precision_increment	Integer	Both
dragnet.log_error_filter_rules	String	Global
end_markers_in_json	Boolean	Both
enforce_gtid_consistency	Enumeration	Global
eq_range_index_dive_limit	Integer	Both
event_scheduler	Enumeration	Global
expire_logs_days	Integer	Global

Variable Name	Variable Type	Variable Scope
explicit_defaults_for_timestamp	Boolean	Both
flush	Boolean	Global
flush_time	Integer	Global
foreign_key_checks	Boolean	Both
ft_boolean_syntax	String	Global
general_log	Boolean	Global
general_log_file	File name	Global
generated_random_password_length	Integer	Both
group_concat_max_len	Integer	Both
group_replication_allow_local_disjoint_gtids_join	Boolean	Global
group_replication_allow_local_lower_version_join	Boolean	Global
group_replication_auto_increment_increment	Integer	Global
group_replication_autorejoin_tries	Integer	Global
group_replication_bootstrap_group	Boolean	Global
group_replication_clone_threshold	Integer	Global
group_replication_communication_debug_options	String	Global
group_replication_communication_max_message_size	Integer	Global
group_replication_components_stop_timeout	Integer	Global
group_replication_compression_threshold	Integer	Global
group_replication_consistency	Enumeration	Both
group_replication_enforce_update_everywhere_checks	Boolean	Global
group_replication_exit_state_action	Enumeration	Global
group_replication_flow_control_applier_threshold	Integer	Global
group_replication_flow_control_certifier_threshold	Integer	Global
group_replication_flow_control_hold_percent	Integer	Global
group_replication_flow_control_max_commit_quota	Integer	Global
group_replication_flow_control_member_quota_percent	Integer	Global
group_replication_flow_control_min_quota	Integer	Global
group_replication_flow_control_min_recovery_quota	Integer	Global
group_replication_flow_control_mode	Enumeration	Global
group_replication_flow_control_period	Integer	Global
group_replication_flow_control_release_percent	Integer	Global
group_replication_force_members	String	Global
group_replication_group_name	String	Global
group_replication_group_seeds	String	Global
group_replication_gtid_assignment_block_size	Integer	Global
group_replication_ip_whitelist	String	Global
group_replication_local_address	String	Global

Variable Name	Variable Type	Variable Scope
group_replication_member_expel_timeout	Integer	Global
group_replication_member_weight	Integer	Global
group_replication_message_cache_size	Integer	Global
group_replication_poll_spin_loops	Integer	Global
group_replication_recovery_complete_at	Enumeration	Global
group_replication_recovery_compression_algorithm	Set	Global
group_replication_recovery_get_public_key	Boolean	Global
group_replication_recovery_public_key_path	File name	Global
group_replication_recovery_reconnect_interval	Integer	Global
group_replication_recovery_retry_count	Integer	Global
group_replication_recovery_ssl_ca	String	Global
group_replication_recovery_ssl_capath	String	Global
group_replication_recovery_ssl_cert	String	Global
group_replication_recovery_ssl_cipher	String	Global
group_replication_recovery_ssl_crl	File name	Global
group_replication_recovery_ssl_crlpath	Directory name	Global
group_replication_recovery_ssl_key	String	Global
group_replication_recovery_ssl_verify_server_cert	Boolean	Global
group_replication_recovery_use_ssl	Boolean	Global
group_replication_recovery_zstd_compression_level	Integer	Global
group_replication_single_primary_mode	Boolean	Global
group_replication_ssl_mode	Enumeration	Global
group_replication_start_on_boot	Boolean	Global
group_replication_transaction_size_limit	Integer	Global
group_replication_unreachable_majority_timeout	Integer	Global
gtid_executed_compression_period	Integer	Global
gtid_mode	Enumeration	Global
gtid_next	Enumeration	Session
gtid_purged	String	Global
histogram_generation_max_mem_size	Integer	Both
host_cache_size	Integer	Global
identity	Integer	Session
immediate_server_version	Integer	Session
information_schema_stats_expiry	Integer	Both
init_connect	String	Global
init_slave	String	Global
innodb_adaptive_flushing	Boolean	Global
innodb_adaptive_flushing_lwm	Integer	Global

Variable Name	Variable Type	Variable Scope
innodb_adaptive_hash_index	Boolean	Global
innodb_adaptive_max_sleep_delay	Integer	Global
innodb_api_bk_commit_interval	Integer	Global
innodb_api_trx_level	Integer	Global
innodb_autoextend_increment	Integer	Global
innodb_background_drop_list_empty	Boolean	Global
innodb_buffer_pool_dump_at_shutdown	Boolean	Global
innodb_buffer_pool_dump_now	Boolean	Global
innodb_buffer_pool_dump_pct	Integer	Global
innodb_buffer_pool_filename	File name	Global
innodb_buffer_pool_in_core_file	Boolean	Global
innodb_buffer_pool_load_abort	Boolean	Global
innodb_buffer_pool_load_now	Boolean	Global
innodb_buffer_pool_size	Integer	Global
innodb_change_buffer_max_size	Integer	Global
innodb_change_buffering	Enumeration	Global
innodb_change_buffering_debug	Integer	Global
innodb_checkpoint_disabled	Boolean	Global
innodb_checksum_algorithm	Enumeration	Global
innodb_cmp_per_index_enabled	Boolean	Global
innodb_commit_concurrency	Integer	Global
innodb_compress_debug	Enumeration	Global
innodb_compression_failure_threshold_pct	Integer	Global
innodb_compression_level	Integer	Global
innodb_compression_pad_pct_max	Integer	Global
innodb_concurrency_tickets	Integer	Global
innodb_ddl_log_crash_reset_debug	Boolean	Global
innodb_deadlock_detect	Boolean	Global
innodb_default_row_format	Enumeration	Global
innodb_disable_sort_file_cache	Boolean	Global
innodb_fast_shutdown	Integer	Global
innodb_fil_make_page_dirty_debug	Integer	Global
innodb_file_per_table	Boolean	Global
innodb_fill_factor	Integer	Global
innodb_flush_log_at_timeout	Integer	Global
innodb_flush_log_at_trx_commit	Enumeration	Global
innodb_flush_neighbors	Enumeration	Global
innodb_flush_sync	Boolean	Global

Variable Name	Variable Type	Variable Scope
innodb_flushing_avg_loops	Integer	Global
innodb_fsync_threshold	Integer	Global
innodb_ft_aux_table	String	Global
innodb_ft_enable_diag_print	Boolean	Global
innodb_ft_enable_stopword	Boolean	Both
innodb_ft_num_word_optimize	Integer	Global
innodb_ft_result_cache_limit	Integer	Global
innodb_ft_server_stopword_table	String	Global
innodb_ft_user_stopword_table	String	Both
innodb_idle_flush_pct	Integer	Global
innodb_io_capacity	Integer	Global
innodb_io_capacity_max	Integer	Global
innodb_limit_optimistic_insert_debug	Integer	Global
innodb_lock_wait_timeout	Integer	Both
innodb_log_buffer_size	Integer	Global
innodb_log_checkpoint_fuzzy_now	Boolean	Global
innodb_log_checkpoint_now	Boolean	Global
innodb_log_checksums	Boolean	Global
innodb_log_compressed_pages	Boolean	Global
innodb_log_spin_cpu_abs_lwm	Integer	Global
innodb_log_spin_cpu_pct_hwm	Integer	Global
innodb_log_wait_for_flush_spin_hwm	Integer	Global
innodb_log_write_ahead_size	Integer	Global
innodb_lru_scan_depth	Integer	Global
innodb_max_dirty_pages_pct	Numeric	Global
innodb_max_dirty_pages_pct_lwm	Numeric	Global
innodb_max_purge_lag	Integer	Global
innodb_max_purge_lag_delay	Integer	Global
innodb_max_undo_log_size	Integer	Global
innodb_merge_threshold_set_all_debug	Integer	Global
innodb_monitor_disable	String	Global
innodb_monitor_enable	String	Global
innodb_monitor_reset	Enumeration	Global
innodb_monitor_reset_all	Enumeration	Global
innodb_old_blocks_pct	Integer	Global
innodb_old_blocks_time	Integer	Global
innodb_online_alter_log_max_size	Integer	Global
innodb_optimize_fulltext_only	Boolean	Global

Variable Name	Variable Type	Variable Scope
innodb_parallel_read_threads	Integer	Session
innodb_print_all_deadlocks	Boolean	Global
innodb_print_ddl_logs	Boolean	Global
innodb_purge_batch_size	Integer	Global
innodb_purge_rseg_truncate_frequency	Integer	Global
innodb_random_read_ahead	Boolean	Global
innodb_read_ahead_threshold	Integer	Global
innodb_redo_log_archive_dirs	String	Global
innodb_redo_log_encrypt	Boolean	Global
innodb_replication_delay	Integer	Global
innodb_rollback_segments	Integer	Global
innodb_saved_page_number_debug	Integer	Global
innodb_spin_wait_delay	Integer	Global
innodb_spin_wait_pause_multiplier	Integer	Global
innodb_stats_auto_recalc	Boolean	Global
innodb_stats_include_delete_marked	Boolean	Global
innodb_stats_method	Enumeration	Global
innodb_stats_on_metadata	Boolean	Global
innodb_stats_persistent	Boolean	Global
innodb_stats_persistent_sample_pages	Integer	Global
innodb_stats_transient_sample_pages	Integer	Global
innodb_status_output	Boolean	Global
innodb_status_output_locks	Boolean	Global
innodb_strict_mode	Boolean	Both
innodb_sync_spin_loops	Integer	Global
innodb_table_locks	Boolean	Both
innodb_thread_concurrency	Integer	Global
innodb_thread_sleep_delay	Integer	Global
innodb_tmpdir	Directory name	Both
innodb_trx_purge_view_update_only_debug	Boolean	Global
innodb_trx_rseg_n_slots_debug	Integer	Global
innodb_undo_log_encrypt	Boolean	Global
innodb_undo_log_truncate	Boolean	Global
innodb_undo_logs	Integer	Global
innodb_undo_tablespaces	Integer	Global
insert_id	Integer	Session
interactive_timeout	Integer	Both
internal_tmp_disk_storage_engine	Enumeration	Global

Variable Name	Variable Type	Variable Scope
internal_tmp_mem_storage_engine	Enumeration	Both
join_buffer_size	Integer	Both
keep_files_on_create	Boolean	Both
key_buffer_size	Integer	Global
key_cache_age_threshold	Integer	Global
key_cache_block_size	Integer	Global
key_cache_division_limit	Integer	Global
keyring_aws_cmk_id	String	Global
keyring_aws_region	Enumeration	Global
keyring_encrypted_file_data	File name	Global
keyring_encrypted_file_password	String	Global
keyring_file_data	File name	Global
keyring_okv_conf_dir	Directory name	Global
keyring_operations	Boolean	Global
last_insert_id	Integer	Session
lc_messages	String	Both
lc_time_names	String	Both
local_infile	Boolean	Global
lock_wait_timeout	Integer	Both
log_bin_trust_function_creators	Boolean	Global
log_builtin_as_identified_by_password	Boolean	Global
log_error_filter_rules	String	Global
log_error_services	String	Global
log_error_suppression_list	String	Global
log_error_verbosity	Integer	Global
log_output	Set	Global
log_queries_not_using_indexes	Boolean	Global
log_slow_admin_statements	Boolean	Global
log_slow_extra	Boolean	Global
log_slow_slave_statements	Boolean	Global
log_statements_unsafe_for_binlog	Boolean	Global
log_syslog	Boolean	Global
log_syslog_facility	String	Global
log_syslog_include_pid	Boolean	Global
log_syslog_tag	String	Global
log_throttle_queries_not_using_indexes	Integer	Global
log_timestamps	Enumeration	Global
log_warnings	Integer	Global

Variable Name	Variable Type	Variable Scope
long_query_time	Numeric	Both
low_priority_updates	Boolean	Both
mandatory_roles	String	Global
master_info_repository	String	Global
master_verify_checksum	Boolean	Global
max_allowed_packet	Integer	Both
max_binlog_cache_size	Integer	Global
max_binlog_size	Integer	Global
max_binlog_stmt_cache_size	Integer	Global
max_connect_errors	Integer	Global
max_connections	Integer	Global
max_delayed_threads	Integer	Both
max_error_count	Integer	Both
max_execution_time	Integer	Both
max_heap_table_size	Integer	Both
max_insert_delayed_threads	Integer	Both
max_join_size	Integer	Both
max_length_for_sort_data	Integer	Both
max_points_in_geometry	Integer	Both
max_prepared_stmt_count	Integer	Global
max_relay_log_size	Integer	Global
max_seeks_for_key	Integer	Both
max_sort_length	Integer	Both
max_sp_recursion_depth	Integer	Both
max_tmp_tables	Integer	Both
max_user_connections	Integer	Both
max_write_lock_count	Integer	Global
min_examined_row_limit	Integer	Both
multi_range_count	Integer	Both
myisam_data_pointer_size	Integer	Global
myisam_max_sort_file_size	Integer	Global
myisam_repair_threads	Integer	Both
myisam_sort_buffer_size	Integer	Both
myisam_stats_method	Enumeration	Both
myisam_use_mmap	Boolean	Global
mysql_firewall_mode	Boolean	Global
mysql_firewall_trace	Boolean	Global
mysql_native_password_proxy_users	Boolean	Global

Variable Name	Variable Type	Variable Scope
mysqlx_connect_timeout	Integer	Global
mysqlx_document_id_unique_prefix	Integer	Global
mysqlx_enable_hello_notice	Boolean	Global
mysqlx_idle_worker_thread_timeout	Integer	Global
mysqlx_interactive_timeout	Integer	Global
mysqlx_max_allowed_packet	Integer	Global
mysqlx_max_connections	Integer	Global
mysqlx_min_worker_threads	Integer	Global
mysqlx_read_timeout	Integer	Session
mysqlx_wait_timeout	Integer	Session
mysqlx_write_timeout	Integer	Session
ndb_allow_copying_alter_table	Boolean	Both
ndb_autoincrement_prefetch_sz	Integer	Both
ndb_blob_read_batch_bytes	Integer	Both
ndb_blob_write_batch_bytes	Integer	Both
ndb_cache_check_time	Integer	Global
ndb_clear_apply_status	Boolean	Global
ndb_data_node_neighbour	Integer	Global
ndb_dbg_check_shares	Integer	Both
ndb_default_column_format	Enumeration	Global
ndb_default_column_format	Enumeration	Global
ndb_deferred_constraints	Integer	Both
ndb_deferred_constraints	Integer	Both
ndb_distribution	Enumeration	Global
ndb_distribution	Enumeration	Global
ndb_eventbuffer_free_percent	Integer	Global
ndb_eventbuffer_max_alloc	Integer	Global
ndb_extra_logging	Integer	Global
ndb_force_send	Boolean	Both
ndb_fully_replicated	Boolean	Both
ndb_index_stat_enable	Boolean	Both
ndb_index_stat_option	String	Both
ndb_join_pushdown	Boolean	Both
ndb_log_bin	Boolean	Both
ndb_log_binlog_index	Boolean	Global
ndb_log_empty_epochs	Boolean	Global
ndb_log_empty_epochs	Boolean	Global
ndb_log_empty_update	Boolean	Global

Variable Name	Variable Type	Variable Scope
ndb_log_empty_update	Boolean	Global
ndb_log_exclusive_reads	Boolean	Both
ndb_log_exclusive_reads	Boolean	Both
ndb_log_update_as_write	Boolean	Global
ndb_log_update_minimal	Boolean	Global
ndb_log_updated_only	Boolean	Global
ndb_metadata_check	Boolean	Global
ndb_metadata_check_interval	Integer	Global
ndb_optimization_delay	Integer	Global
ndb_read_backup	Boolean	Global
ndb_recv_thread_activation_threshold	Integer	Global
ndb_recv_thread_cpu_mask	Bitmap	Global
ndb_report_thresh_binlog_epoch_slip	Integer	Global
ndb_report_thresh_binlog_mem_usage	Integer	Global
ndb_row_checksum	Integer	Both
ndb_schema_dist_lock_wait_timeout	Integer	Global
ndb_show_foreign_key_mock_tables	Boolean	Global
ndb_slave_conflict_role	Enumeration	Global
ndb_table_no_logging	Boolean	Session
ndb_table_temporary	Boolean	Session
ndb_use_exact_count	Boolean	Both
ndb_use_transactions	Boolean	Both
ndbinfo_max_bytes	Integer	Both
ndbinfo_max_rows	Integer	Both
ndbinfo_offline	Boolean	Global
ndbinfo_show_hidden	Boolean	Both
ndbinfo_table_prefix	String	Both
net_buffer_length	Integer	Both
net_read_timeout	Integer	Both
net_retry_count	Integer	Both
net_write_timeout	Integer	Both
new	Boolean	Both
offline_mode	Boolean	Global
old_alter_table	Boolean	Both
old_passwords	Enumeration	Both
optimizer_prune_level	Integer	Both
optimizer_search_depth	Integer	Both
optimizer_switch	Set	Both

Variable Name	Variable Type	Variable Scope
optimizer_trace	String	Both
optimizer_trace_features	String	Both
optimizer_trace_limit	Integer	Both
optimizer_trace_max_mem_size	Integer	Both
optimizer_trace_offset	Integer	Both
original_commit_timestamp	Numeric	Session
original_server_version	Integer	Session
parser_max_mem_size	Integer	Both
partial_revokes	Boolean	Global
password_history	Integer	Global
password_require_current	Boolean	Global
password_reuse_interval	Integer	Global
performance_schema_max_digest_sample_age	Integer	Global
preload_buffer_size	Integer	Both
print_identified_with_as_hex	Boolean	Both
profiling	Boolean	Both
profiling_history_size	Integer	Both
protocol_compression_algorithms	Set	Global
pseudo_slave_mode	Integer	Session
pseudo_thread_id	Integer	Session
query_alloc_block_size	Integer	Both
query_cache_limit	Integer	Global
query_cache_min_res_unit	Integer	Global
query_cache_size	Integer	Global
query_cache_type	Enumeration	Both
query_cache_wlock_invalidate	Boolean	Both
query_prealloc_size	Integer	Both
rand_seed1	Integer	Session
rand_seed2	Integer	Session
range_alloc_block_size	Integer	Both
range_optimizer_max_mem_size	Integer	Both
rbr_exec_mode	Enumeration	Both
read_buffer_size	Integer	Both
read_only	Boolean	Global
read_rnd_buffer_size	Integer	Both
regexp_stack_limit	Integer	Global
regexp_time_limit	Integer	Global
relay_log_info_repository	String	Global

Variable Name	Variable Type	Variable Scope
relay_log_purge	Boolean	Global
require_secure_transport	Boolean	Global
resultset_metadata	Enumeration	Session
rewriter_enabled	Boolean	Global
rewriter_verbose	Integer	Global
rpl_read_size	Integer	Global
rpl_semi_sync_master_enabled	Boolean	Global
rpl_semi_sync_master_timeout	Integer	Global
rpl_semi_sync_master_trace_level	Integer	Global
rpl_semi_sync_master_wait_for_slave_count	Integer	Global
rpl_semi_sync_master_wait_no_slave	Boolean	Global
rpl_semi_sync_master_wait_point	Enumeration	Global
rpl_semi_sync_slave_enabled	Boolean	Global
rpl_semi_sync_slave_trace_level	Integer	Global
rpl_stop_slave_timeout	Integer	Global
schema_definition_cache	Integer	Global
secondary_engine_cost_threshold	Numeric	Session
secure_auth	Boolean	Global
server_id	Integer	Global
session_track_gtids	Enumeration	Both
session_track_schema	Boolean	Both
session_track_state_change	Boolean	Both
session_track_system_variables	String	Both
session_track_transaction_info	Enumeration	Both
sha256_password_proxy_users	Boolean	Global
show_compatibility_56	Boolean	Global
show_create_table_skip_secondary_engine	Boolean	Session
show_create_table_verbosity	Boolean	Both
show_old_temporals	Boolean	Both
slave_allow_batching	Boolean	Global
slave_checkpoint_group	Integer	Global
slave_checkpoint_period	Integer	Global
slave_compressed_protocol	Boolean	Global
slave_exec_mode	Enumeration	Global
slave_max_allowed_packet	Integer	Global
slave_net_timeout	Integer	Global
slave_parallel_type	Enumeration	Global
slave_parallel_workers	Integer	Global

Variable Name	Variable Type	Variable Scope
slave_pending_jobs_size_max	Integer	Global
slave_preserve_commit_order	Boolean	Global
slave_rows_search_algorithms	Set	Global
slave_sql_verify_checksum	Boolean	Global
slave_transaction_retries	Integer	Global
slow_launch_time	Integer	Global
slow_query_log	Boolean	Global
slow_query_log_file	File name	Global
sort_buffer_size	Integer	Both
sql_auto_is_null	Boolean	Both
sql_big_selects	Boolean	Both
sql_buffer_result	Boolean	Both
sql_log_bin	Boolean	Session
sql_log_off	Boolean	Both
sql_mode	Set	Both
sql_notes	Boolean	Both
sql_quote_show_create	Boolean	Both
sql_require_primary_key	Boolean	Both
sql_safe_updates	Boolean	Both
sql_select_limit	Integer	Both
sql_slave_skip_counter	Integer	Global
sql_warnings	Boolean	Both
ssl_ca	File name	Global
ssl_capath	Directory name	Global
ssl_cert	File name	Global
ssl_cipher	String	Global
ssl_crl	File name	Global
ssl_crlpath	Directory name	Global
ssl_fips_mode	Enumeration	Global
ssl_key	File name	Global
stored_program_cache	Integer	Global
stored_program_definition_cache	Integer	Global
super_read_only	Boolean	Global
sync_binlog	Integer	Global
sync_master_info	Integer	Global
sync_relay_log	Integer	Global
sync_relay_log_info	Integer	Global
syseventlog.facility	String	Global

Variable Name	Variable Type	Variable Scope
syseventlog.include_pid	Boolean	Global
syseventlog.tag	String	Global
table_definition_cache	Integer	Global
table_encryption_privilege_check	Boolean	Global
table_open_cache	Integer	Global
tablespace_definition_cache	Integer	Global
temptable_max_ram	Integer	Global
temptable_use_mmap	Boolean	Global
thread_cache_size	Integer	Global
thread_pool_high_priority_connection	Integer	Both
thread_pool_max_active_query_threads	Integer	Global
thread_pool_max_unused_threads	Integer	Global
thread_pool_prio_kickup_timer	Integer	Both
thread_pool_stall_limit	Integer	Global
time_zone	String	Both
timestamp	Numeric	Session
tls_ciphersuites	String	Global
tls_version	String	Global
tmp_table_size	Integer	Both
transaction_alloc_block_size	Integer	Both
transaction_allow_batching	Boolean	Session
transaction_isolation	Enumeration	Both
transaction_prealloc_size	Integer	Both
transaction_read_only	Boolean	Both
transaction_write_set_extraction	Enumeration	Both
tx_isolation	Enumeration	Both
tx_read_only	Boolean	Both
unique_checks	Boolean	Both
updatable_views_with_limit	Boolean	Both
use_secondary_engine	Enumeration	Session
validate_password_check_user_name	Boolean	Global
validate_password_dictionary_file	File name	Global
validate_password_length	Integer	Global
validate_password_mixed_case_count	Integer	Global
validate_password_number_count	Integer	Global
validate_password_policy	Enumeration	Global
validate_password_special_char_count	Integer	Global
validate_password.check_user_name	Boolean	Global

Variable Name	Variable Type	Variable Scope
validate_password.dictionary_file	File name	Global
validate_password.length	Integer	Global
validate_password.mixed_case_count	Integer	Global
validate_password.number_count	Integer	Global
validate_password.policy	Enumeration	Global
validate_password.special_char_count	Integer	Global
version_tokens_session	String	Both
wait_timeout	Integer	Both
windowing_use_high_precision	Boolean	Both

5.1.9.3 Persisted System Variables

The MySQL server maintains system variables that configure its operation. A system variable can have a global value that affects server operation as a whole, a session value that affects the current session, or both. Many system variables are dynamic and can be changed at runtime using the `SET` statement to affect operation of the current server instance. `SET` can also be used to persist certain global system variables to the `mysqld-auto.cnf` file in the data directory, to affect server operation for subsequent startups. `RESET PERSIST` removes persisted settings from `mysqld-auto.cnf`.

The following discussion describes aspects of persisting system variables:

- Overview of Persisted System Variables
- Syntax for Persisting System Variables
- Obtaining Information About Persisted System Variables
- Format and Server Handling of the mysqld-auto.cnf File

Overview of Persisted System Variables

The capability of persisting global system variables at runtime enables server configuration that persists across server startups. Although many system variables can be set at startup from a `my.cnf` option file, or at runtime using the `SET` statement, those methods of configuring the server either require login access to the server host, or do not provide the capability of persistently configuring the server at runtime or remotely:

- Modifying an option file requires direct access to that file, which requires login access to the MySQL server host. This is not always convenient.
- Modifying system variables with `SET GLOBAL` is a runtime capability that can be done from clients run locally or from remote hosts, but the changes affect only the currently running server instance. The settings are not persistent and do not carry over to subsequent server startups.

To augment administrative capabilities for server configuration beyond what is achievable by editing option files or using `SET GLOBAL`, MySQL provides variants of `SET` syntax that persist system variable settings to a file named `mysqld-auto.cnf` file in the data directory. Examples:

```
SET PERSIST max_connections = 1000;
SET @@PERSIST.max_connections = 1000;

SET PERSIST_ONLY back_log = 100;
```

```
SET @@PERSIST_ONLY.back_log = 100;
```

MySQL also provides a `RESET PERSIST` statement for removing persisted system variables from `mysqld-auto.cnf`.

Server configuration performed by persisting system variables has these characteristics:

- Persisted settings are made at runtime.
- Persisted settings are permanent. They apply across server restarts.
- Persisted settings can be made from local clients or clients who connect from a remote host. This provides the convenience of remotely configuring multiple MySQL servers from a central client host.
- To persist system variables, you need not have login access to the MySQL server host or file system access to option files. Ability to persist settings is controlled using the MySQL privilege system. See Section 5.1.9.1, “System Variable Privileges”.
- An administrator with sufficient privileges can reconfigure a server by persisting system variables, then cause the server to use the changed settings immediately by executing a `RESTART` statement.
- Persisted settings provide immediate feedback about errors. An error in a manually entered setting might not be discovered until much later. `SET` statements that persist system variables avoid the possibility of malformed settings because settings with syntax errors do not succeed and do not change server configuration.

Syntax for Persisting System Variables

These `SET` syntax options are available for persisting system variables:

- To persist a global system variable to the `mysqld-auto.cnf` option file in the data directory, precede the variable name by the `PERSIST` keyword or the `@@PERSIST.` qualifier:

  ```
  SET PERSIST max_connections = 1000;
  SET @@PERSIST.max_connections = 1000;
  ```

  Like `SET GLOBAL`, `SET PERSIST` sets the global variable runtime value, but also writes the variable setting to the `mysqld-auto.cnf` file (replacing any existing variable setting if there is one).

- To persist a global system variable to the `mysqld-auto.cnf` file without setting the global variable runtime value, precede the variable name by the `PERSIST_ONLY` keyword or the `@@PERSIST_ONLY.` qualifier:

  ```
  SET PERSIST_ONLY back_log = 1000;
  SET @@PERSIST_ONLY.back_log = 1000;
  ```

  Like `PERSIST`, `PERSIST_ONLY` writes the variable setting to `mysqld-auto.cnf`. However, unlike `PERSIST`, `PERSIST_ONLY` does not modify the global variable runtime value. This makes `PERSIST_ONLY` suitable for configuring read-only system variables that can be set only at server startup.

For more information about `SET`, see Section 13.7.6.1, “SET Syntax for Variable Assignment”.

These `RESET PERSIST` syntax options are available for removing persisted system variables:

- To remove all persisted variables from `mysqld-auto.cnf`, use `RESET PERSIST` without naming any system variable:

```
RESET PERSIST;
```

- To remove a specific persisted variable from `mysqld-auto.cnf`, name it in the statement:

```
RESET PERSIST system_var_name;
```

  This includes plugin system variables, even if the plugin is not currently installed. If the variable is not present in the file, an error occurs.

- To remove a specific persisted variable from `mysqld-auto.cnf`, but produce a warning rather than an error if the variable is not present in the file, add an `IF EXISTS` clause to the previous syntax:

```
RESET PERSIST IF EXISTS system_var_name;
```

For more information about `RESET PERSIST`, see Section 13.7.8.7, “RESET PERSIST Syntax”.

Using `SET` to persist a global system variable to a value of `DEFAULT` or to its literal default value assigns the variable its default value and adds a setting for the variable to `mysqld-auto.cnf`. To remove the variable from the file, use `RESET PERSIST`.

Some system variables cannot be persisted. See Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”.

A system variable implemented by a plugin can be persisted if the plugin is installed when the `SET` statement is executed. Assignment of the persisted plugin variable takes effect for subsequent server restarts if the plugin is still installed. If the plugin is no longer installed, the plugin variable will not exist when the server reads the `mysqld-auto.cnf` file. In this case, the server writes a warning to the error log and continues:

```
currently unknown variable 'var_name'
was read from the persisted config file
```

Obtaining Information About Persisted System Variables

The Performance Schema `persisted_variables` table provides an SQL interface to the `mysqld-auto.cnf` file, enabling its contents to be inspected at runtime using `SELECT` statements. See Section 26.12.13.1, “Performance Schema persisted_variables Table”.

The Performance Schema `variables_info` table contains information showing when and by which user each system variable was most recently set. See Section 26.12.13.2, “Performance Schema variables_info Table”.

`RESET PERSIST` affects the contents of the `persisted_variables` table because the table contents correspond to the contents of the `mysqld-auto.cnf` file. On the other hand, because `RESET PERSIST` does not change variable values, it has no effect on the contents of the `variables_info` table until the server is restarted.

Format and Server Handling of the mysqld-auto.cnf File

The `mysqld-auto.cnf` file uses a `JSON` format like this (reformatted slightly for readability):

```
{
```

```
  "Version": 1,
  "mysql_server": {
    "max_connections": {
      "Value": "152",
      "Metadata": {
        "Timestamp": 1.519921356e+15,
        "User": "root",
        "Host": "localhost"
      }
    },
    "transaction_isolation": {
      "Value": "READ-COMMITTED",
      "Metadata": {
        "Timestamp": 1.519921553e+15,
        "User": "root",
        "Host": "localhost"
      }
    },
    "mysql_server_static_options": {
      "innodb_api_enable_mdl": {
        "Value": "0",
        "Metadata": {
          "Timestamp": 1.519921573e+15,
          "User": "root",
          "Host": "localhost"
        }
      },
      "log_slave_updates": {
        "Value": "1",
        "Metadata": {
          "Timestamp": 1.519921582e+15,
          "User": "root",
          "Host": "localhost"
        }
      }
    }
  }
}
```

At startup, the server processes the `mysqld-auto.cnf` file after all other option files (see Section 4.2.2.2, “Using Option Files”). The server handles the file contents as follows:

- If the `persisted_globals_load` system variable is disabled, the server ignores the `mysqld-auto.cnf` file.
- Only read-only variables persisted using `SET PERSIST_ONLY` appear in the `"mysql_server_static_options"` section. All variables present inside this section are appended to the command line and processed with other command-line options.
- All remaining persisted variables are set by executing the equivalent of a `SET GLOBAL` statement later, just before the server starts listening for client connections. These settings therefore do not take effect until late in the startup process, which might be unsuitable for certain system variables. It may be preferable to set such variables in `my.cnf` rather than in `mysqld-auto.cnf`.

Management of the `mysqld-auto.cnf` file should be left to the server. Manipulation of the file should be performed only using `SET` and `RESET PERSIST` statements, not manually:

- Removal of the file results in a loss of all persisted settings at the next server startup. (This is permissible if your intent is to reconfigure the server without these settings.) To remove all settings in the file without removing the file itself, use this statement:

```
RESET PERSIST;
```

- Manual changes to the file may result in a parse error at server startup. In this case, the server reports an error and exits. If this issue occurs, start the server with the `persisted_globals_load` system variable disabled or with the `--no-defaults` option. Alternatively, remove the `mysqld-auto.cnf` file. However, as noted previously, removing this file results in a loss of all persisted settings.

5.1.9.4 Nonpersistible and Persist-Restricted System Variables

`SET PERSIST` and `SET PERSIST_ONLY` enable global system variables to be persisted to the `mysqld-auto.cnf` option file in the data directory (see Section 13.7.6.1, “SET Syntax for Variable Assignment”). However, not all system variables can be persisted, or can be persisted only under certain restrictive conditions. Here are some reasons why a system variable might be nonpersistible or persist-restricted:

- Session system variables cannot be persisted. Session variables cannot be set at server startup, so there is no reason to persist them.
- A global system variable might involve sensitive data such that it should be settable only by a user with direct access to the server host.
- A global system variable might be read only (that is, set only by the server). In this case, it cannot be set by users at all, whether at server startup or at runtime.
- A global system variable might be intended only for internal use.

Nonpersistible system variables cannot be persisted under any circumstances. As of MySQL 8.0.14, persist-restricted system variables can be persisted with `SET PERSIST_ONLY`, but only by users for which the following conditions are satisfied:

- The `persist_only_admin_x509_subject` system variable is set to an SSL certificate X.509 Subject value.
- The user connects to the server using an encrypted connection and supplies an SSL certificate with the designated Subject value.
- The user has sufficient privileges to use `SET PERSIST_ONLY` (see Section 5.1.9.1, “System Variable Privileges”).

For example, `protocol_version` is read only and set only by the server, so it cannot be persisted under any circumstances. On the other hand, `bind_address` is persist-restricted, so it can be set by users who satisfy the preceding conditions.

The following system variables are nonpersistible. This list may change with ongoing development.

```
audit_log_current_session
audit_log_filter_id
character_set_system
core_file
have_statement_timeout
have_symlink
hostname
innodb_version
large_files_support
large_page_size
license
locked_in_memory
log_bin
log_bin_basename
log_bin_index
lower_case_file_system
ndb_version
```

```
ndb_version_string
persist_only_admin_x509_subject
persisted_globals_load
protocol_version
relay_log_basename
relay_log_index
server_uuid
skip_external_locking
system_time_zone
version_comment
version_compile_machine
version_compile_os
version_compile_zlib
```

Persist-restricted system variables are those that are read only and can be set on the command line or in an option file, other than `persist_only_admin_x509_subject` and `persisted_globals_load`. This list may change with ongoing development.

```
audit_log_file
audit_log_format
auto_generate_certs
basedir
bind_address
caching_sha2_password_auto_generate_rsa_keys
caching_sha2_password_private_key_path
caching_sha2_password_public_key_path
character_sets_dir
daemon_memcached_engine_lib_name
daemon_memcached_engine_lib_path
daemon_memcached_option
datadir
default_authentication_plugin
ft_stopword_file
init_file
innodb_buffer_pool_load_at_startup
innodb_data_file_path
innodb_data_home_dir
innodb_dedicated_server
innodb_directories
innodb_force_load_corrupted
innodb_log_group_home_dir
innodb_page_size
innodb_read_only
innodb_temp_data_file_path
innodb_temp_tablespaces_dir
innodb_undo_directory
innodb_undo_tablespaces
keyring_encrypted_file_data
keyring_encrypted_file_password
lc_messages_dir
log_error
mecab_rc_file
named_pipe
pid_file
plugin_dir
port
relay_log
relay_log_info_file
secure_file_priv
sha256_password_auto_generate_rsa_keys
sha256_password_private_key_path
sha256_password_public_key_path
shared_memory
shared_memory_base_name
skip_networking
slave_load_tmpdir
```

```
socket
ssl_ca
ssl_capath
ssl_cert
ssl_crl
ssl_crlpath
ssl_key
tmpdir
version_tokens_session_number
```

To configure the server to enable persisting persist-restricted system variables, use this procedure:

1. Ensure that MySQL is configured to support encrypted connections. See Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

2. Designate an SSL certificate X.509 Subject value that signifies the ability to persist persist-restricted system variables, and generate a certificate that has that Subject. See Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.

3. Start the server with `persist_only_admin_x509_subject` set to the designated Subject value. For example, put these lines in your server `my.cnf` file:

   ```
   [mysqld]
   persist_only_admin_x509_subject="subject-value"
   ```

   The format of the Subject value is the same as used for `CREATE USER ... REQUIRE SUBJECT`. See Section 13.7.1.3, “CREATE USER Syntax”.

   You must perform this step directly on the MySQL server host because `persist_only_admin_x509_subject` itself cannot be persisted at runtime.

4. Restart the server.

5. Distribute the SSL certificate that has the designated Subject value to users who are to be permitted to persist persist-restricted system variables.

Suppose that `myclient-cert.pem` is the SSL certificate to be used by clients who can persist persist-restricted system variables. Display the certificate contents using the `openssl` command:

```
shell> openssl x509 -text -in myclient-cert.pem
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 2 (0x2)
    Signature Algorithm: md5WithRSAEncryption
        Issuer: C=US, ST=IL, L=Chicago, O=MyOrg, OU=CA, CN=MyCN
        Validity
            Not Before: Oct 18 17:03:03 2018 GMT
            Not After : Oct 15 17:03:03 2028 GMT
        Subject: C=US, ST=IL, L=Chicago, O=MyOrg, OU=client, CN=MyCN
...
```

The `openssl` output shows that the certificate Subject value is:

```
C=US, ST=IL, L=Chicago, O=MyOrg, OU=client, CN=MyCN
```

To specify the Subject for MySQL, use this format:

```
```

```
/C=US/ST=IL/L=Chicago/O=MyOrg/OU=client/CN=MyCN
```

Configure the server `my.cnf` file with the Subject value:

```
[mysqld]
persist_only_admin_x509_subject="/C=US/ST=IL/L=Chicago/O=MyOrg/OU=client/CN=MyCN"
```

Restart the server so that the new configuration takes effect.

Distribute the SSL certificate (and any other associated SSL files) to the appropriate users. Such a user then connects to the server with the certificate and any other SSL options required to establish an encrypted connection.

To use X.509, clients must specify the `--ssl-key` and `--ssl-cert` options to connect. It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified. For example:

```
shell> mysql --ssl-key=myclient-key.pem --ssl-cert=myclient-cert.pem --ssl-ca=mycacert.pem
```

Assuming that the user has sufficient privileges to use `SET PERSIST_ONLY`, persist-restricted system variables can be persisted like this:

```
mysql> SET PERSIST_ONLY socket = '/tmp/mysql.sock';
Query OK, 0 rows affected (0.00 sec)
```

If the server is not configured to enable persisting persist-restricted system variables, or the user does not satisfy the required conditions for that capability, an error occurs:

```
mysql> SET PERSIST_ONLY socket = '/tmp/mysql.sock';
ERROR 1238 (HY000): Variable 'socket' is a non persistent read only variable
```

5.1.9.5 Structured System Variables

A structured variable differs from a regular system variable in two respects:

- Its value is a structure with components that specify server parameters considered to be closely related.
- There might be several instances of a given type of structured variable. Each one has a different name and refers to a different resource maintained by the server.

MySQL supports one structured variable type, which specifies parameters governing the operation of key caches. A key cache structured variable has these components:

- `key_buffer_size`
- `key_cache_block_size`
- `key_cache_division_limit`
- `key_cache_age_threshold`

This section describes the syntax for referring to structured variables. Key cache variables are used for syntax examples, but specific details about how key caches operate are found elsewhere, in Section 8.10.2, “The MyISAM Key Cache”.

To refer to a component of a structured variable instance, you can use a compound name in *`instance_name.component_name`* format. Examples:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

For each structured system variable, an instance with the name of `default` is always predefined. If you refer to a component of a structured variable without any instance name, the `default` instance is used. Thus, `default.key_buffer_size` and `key_buffer_size` both refer to the same system variable.

Structured variable instances and components follow these naming rules:

- For a given type of structured variable, each instance must have a name that is unique *within* variables of that type. However, instance names need not be unique *across* structured variable types. For example, each structured variable has an instance named `default`, so `default` is not unique across variable types.
- The names of the components of each structured variable type must be unique across all system variable names. If this were not true (that is, if two different types of structured variables could share component member names), it would not be clear which default structured variable to use for references to member names that are not qualified by an instance name.
- If a structured variable instance name is not legal as an unquoted identifier, refer to it as a quoted identifier using backticks. For example, `hot-cache` is not legal, but `` `hot-cache` `` is.
- `global`, `session`, and `local` are not legal instance names. This avoids a conflict with notation such as `@@GLOBAL.`*`var_name`* for referring to nonstructured system variables.

Currently, the first two rules have no possibility of being violated because the only structured variable type is the one for key caches. These rules will assume greater significance if some other type of structured variable is created in the future.

With one exception, you can refer to structured variable components using compound names in any context where simple variable names can occur. For example, you can assign a value to a structured variable using a command-line option:

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

In an option file, use this syntax:

```
[mysqld]
hot_cache.key_buffer_size=64K
```

If you start the server with this option, it creates a key cache named `hot_cache` with a size of 64KB in addition to the default key cache that has a default size of 8MB.

Suppose that you start the server as follows:

```
shell> mysqld --key_buffer_size=256K \
         --extra_cache.key_buffer_size=128K \
         --extra_cache.key_cache_block_size=2048
```

In this case, the server sets the size of the default key cache to 256KB. (You could also have written `--default.key_buffer_size=256K`.) In addition, the server creates a second key cache named `extra_cache` that has a size of 128KB, with the size of block buffers for caching table index blocks set to 2048 bytes.

The following example starts the server with three different key caches having sizes in a 3:1:1 ratio:

```
shell> mysqld --key_buffer_size=6M \
         --hot_cache.key_buffer_size=2M \
         --cold_cache.key_buffer_size=2M
```

Structured variable values may be set and retrieved at runtime as well. For example, to set a key cache named `hot_cache` to a size of 10MB, use either of these statements:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
mysql> SET @@GLOBAL.hot_cache.key_buffer_size = 10*1024*1024;
```

To retrieve the cache size, do this:

```
mysql> SELECT @@GLOBAL.hot_cache.key_buffer_size;
```

However, the following statement does not work. The variable is not interpreted as a compound name, but as a simple string for a `LIKE` pattern-matching operation:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

This is the exception to being able to use structured variable names anywhere a simple variable name may occur.

5.1.10 Server Status Variables

The MySQL server maintains many status variables that provide information about its operation. You can view these variables and their values by using the `SHOW [GLOBAL | SESSION] STATUS` statement (see Section 13.7.7.35, “SHOW STATUS Syntax”). The optional `GLOBAL` keyword aggregates the values over all connections, and `SESSION` shows the values for the current connection.

```
mysql> SHOW GLOBAL STATUS;
+-----------------------------------+------------+
| Variable_name                     | Value      |
+-----------------------------------+------------+
| Aborted_clients                   | 0          |
| Aborted_connects                  | 0          |
| Bytes_received                    | 155372598  |
| Bytes_sent                        | 1176560426 |
...
| Connections                       | 30023      |
| Created_tmp_disk_tables           | 0          |
| Created_tmp_files                 | 3          |
| Created_tmp_tables                | 2          |
...
| Threads_created                   | 217        |
| Threads_running                   | 88         |
| Uptime                            | 1389872    |
+-----------------------------------+------------+
```

Many status variables are reset to 0 by the `FLUSH STATUS` statement.

This section provides a description of each status variable. For a status variable summary, see Section 5.1.6, “Server Status Variable Reference”.

The status variables have the following meanings.

- `Aborted_clients`

The number of connections that were aborted because the client died without closing the connection properly. See Section B.4.2.10, “Communication Errors and Aborted Connections”.

- `Aborted_connects`

  The number of failed attempts to connect to the MySQL server. See Section B.4.2.10, “Communication Errors and Aborted Connections”.

  For additional connection-related information, check the `Connection_errors_xxx` status variables and the `host_cache` table.

- `Binlog_cache_disk_use`

  The number of transactions that used the temporary binary log cache but that exceeded the value of `binlog_cache_size` and used a temporary file to store statements from the transaction.

  The number of nontransactional statements that caused the binary log transaction cache to be written to disk is tracked separately in the `Binlog_stmt_cache_disk_use` status variable.

- `Acl_cache_items_count`

  The number of cached privilege objects. Each object is the privilege combination of a user and its active roles.

- `Binlog_cache_use`

  The number of transactions that used the binary log cache.

- `Binlog_stmt_cache_disk_use`

  The number of nontransaction statements that used the binary log statement cache but that exceeded the value of `binlog_stmt_cache_size` and used a temporary file to store those statements.

- `Binlog_stmt_cache_use`

  The number of nontransactional statements that used the binary log statement cache.

- `Bytes_received`

  The number of bytes received from all clients.

- `Bytes_sent`

  The number of bytes sent to all clients.

- `Caching_sha2_password_rsa_public_key`

  The public key used by the `caching_sha2_password` authentication plugin for RSA key pair-based password exchange. The value is nonempty only if the server successfully initializes the private and public keys in the files named by the `caching_sha2_password_private_key_path` and `caching_sha2_password_public_key_path` system variables. The value of `Caching_sha2_password_rsa_public_key` comes from the latter file.

- `Com_xxx`

  The `Com_xxx` statement counter variables indicate the number of times each *`xxx`* statement has been executed. There is one status variable for each type of statement. For example, `Com_delete`

and Com_update count DELETE and UPDATE statements, respectively. Com_delete_multi and Com_update_multi are similar but apply to DELETE and UPDATE statements that use multiple-table syntax.

The discussion at the beginning of this section indicates how to relate these statement-counting status variables to other such variables.

All of the Com_stmt_*xxx* variables are increased even if a prepared statement argument is unknown or an error occurred during execution. In other words, their values correspond to the number of requests issued, not to the number of requests successfully completed.

The Com_stmt_*xxx* status variables are as follows:

- Com_stmt_prepare
- Com_stmt_execute
- Com_stmt_fetch
- Com_stmt_send_long_data
- Com_stmt_reset
- Com_stmt_close

Those variables stand for prepared statement commands. Their names refer to the COM_*xxx* command set used in the network layer. In other words, their values increase whenever prepared statement API calls such as mysql_stmt_prepare(), mysql_stmt_execute(), and so forth are executed. However, Com_stmt_prepare, Com_stmt_execute and Com_stmt_close also increase for PREPARE, EXECUTE, or DEALLOCATE PREPARE, respectively. Additionally, the values of the older statement counter variables Com_prepare_sql, Com_execute_sql, and Com_dealloc_sql increase for the PREPARE, EXECUTE, and DEALLOCATE PREPARE statements. Com_stmt_fetch stands for the total number of network round-trips issued when fetching from cursors.

Com_stmt_reprepare indicates the number of times statements were automatically reprepared by the server, for example, after metadata changes to tables or views referred to by the statement. A reprepare operation increments Com_stmt_reprepare, and also Com_stmt_prepare.

Com_explain_other indicates the number of EXPLAIN FOR CONNECTION statements executed. See Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”.

Com_change_repl_filter indicates the number of CHANGE REPLICATION FILTER statements executed.

- Compression

  Whether the client connection uses compression in the client/server protocol.

  As of MySQL 8.0.18, this status variable is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- Compression_algorithm

  The name of the compression algorithm in use for the current connection to the server. The value can be any algorithm permitted in the value of the protocol_compression_algorithms system variable. For example, the value is uncompressed if the connection does not use compression, or zlib if the connection uses the zlib algorithm.

For more information, see Section 4.2.6, “Connection Compression Control”.

This variable was added in MySQL 8.0.18.

- `Compression_level`

  The compression level in use for the current connection to the server. The value is 6 for `zlib` connections (the default `zlib` algorithm compression level), 1 to 22 for `zstd` connections, and 0 for `uncompressed` connections.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This variable was added in MySQL 8.0.18.

- `Connection_errors_`*`xxx`*

  These variables provide information about errors that occur during the client connection process. They are global only and represent error counts aggregated across connections from all hosts. These variables track errors not accounted for by the host cache (see Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”), such as errors that are not associated with TCP connections, occur very early in the connection process (even before an IP address is known), or are not specific to any particular IP address (such as out-of-memory conditions).

  - `Connection_errors_accept`

    The number of errors that occurred during calls to `accept()` on the listening port.

  - `Connection_errors_internal`

    The number of connections refused due to internal errors in the server, such as failure to start a new thread or an out-of-memory condition.

  - `Connection_errors_max_connections`

    The number of connections refused because the server `max_connections` limit was reached.

  - `Connection_errors_peer_address`

    The number of errors that occurred while searching for connecting client IP addresses.

  - `Connection_errors_select`

    The number of errors that occurred during calls to `select()` or `poll()` on the listening port. (Failure of this operation does not necessarily means a client connection was rejected.)

  - `Connection_errors_tcpwrap`

    The number of connections refused by the `libwrap` library.

- `Connections`

  The number of connection attempts (successful or not) to the MySQL server.

- `Created_tmp_disk_tables`

  The number of internal on-disk temporary tables created by the server while executing statements.

If an internal temporary table is created initially as an in-memory table but becomes too large, MySQL automatically converts it to an on-disk table. The maximum size for in-memory temporary tables is the minimum of the `tmp_table_size` and `max_heap_table_size` values. If `Created_tmp_disk_tables` is large, you may want to increase the `tmp_table_size` or `max_heap_table_size` value to lessen the likelihood that internal temporary tables in memory will be converted to on-disk tables.

You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` and `Created_tmp_tables` variables.

See also Section 8.4.4, “Internal Temporary Table Use in MySQL”.

- `Created_tmp_files`

  How many temporary files `mysqld` has created.

- `Created_tmp_tables`

  The number of internal temporary tables created by the server while executing statements.

  You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` and `Created_tmp_tables` variables.

  See also Section 8.4.4, “Internal Temporary Table Use in MySQL”.

  Each invocation of the `SHOW STATUS` statement uses an internal temporary table and increments the global `Created_tmp_tables` value.

- `Current_tls_ca`

  The active `ssl_ca` value in the SSL context that the server uses for new connections. This context value may differ from the current `ssl_ca` system variable value if the system variable has been changed but `ALTER INSTANCE RELOAD TLS` has not subsequently been executed to reconfigure the SSL context from the context-related system variable values and update the corresponding status variables. (This potential difference in values applies to each corresponding pair of context-related system and status variables. See Server-Side Runtime Configuration for Encrypted Connections.)

  This variable was added in MySQL 8.0.16.

- `Current_tls_capath`

  The active `ssl_capath` value in the SSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Current_tls_cert`

  The active `ssl_cert` value in the SSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Current_tls_cipher`

  The active `ssl_cipher` value in the SSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Current_tls_ciphersuites`

  The active `tls_ciphersuites` value in the SSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Current_tls_crl`

  The active `ssl_crl` value in the SSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Current_tls_crlpath`

  The active `ssl_crlpath` value in the SSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Current_tls_key`

  The active `ssl_key` value in the SSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Current_tls_version`

  The active `tls_version` value in the SSL context that the server uses for new connections. For notes about the relationship between this status variable and its corresponding system variable, see the description of `Current_tls_ca`.

  This variable was added in MySQL 8.0.16.

- `Delayed_errors`

  This status variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `Delayed_insert_threads`

  This status variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `Delayed_writes`

This status variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `dragnet.Status`

  The result of the most recent assignment to the `dragnet.log_error_filter_rules` system variable, empty if no such assignment has occurred.

  This variable was added in MySQL 8.0.12.

- `Flush_commands`

  The number of times the server flushes tables, whether because a user executed a `FLUSH TABLES` statement or due to internal server operation. It is also incremented by receipt of a `COM_REFRESH` packet. This is in contrast to `Com_flush`, which indicates how many `FLUSH` statements have been executed, whether `FLUSH TABLES`, `FLUSH LOGS`, and so forth.

- `group_replication_primary_member`

  Shows the primary member's UUID when the group is operating in single-primary mode. If the group is operating in multi-primary mode, shows an empty string.

  The `group_replication_primary_member` status variable has been deprecated and is scheduled to be removed in a future version.

- `Handler_commit`

  The number of internal `COMMIT` statements.

- `Handler_delete`

  The number of times that rows have been deleted from tables.

- `Handler_external_lock`

  The server increments this variable for each call to its `external_lock()` function, which generally occurs at the beginning and end of access to a table instance. There might be differences among storage engines. This variable can be used, for example, to discover for a statement that accesses a partitioned table how many partitions were pruned before locking occurred: Check how much the counter increased for the statement, subtract 2 (2 calls for the table itself), then divide by 2 to get the number of partitions locked.

- `Handler_mrr_init`

  The number of times the server uses a storage engine's own Multi-Range Read implementation for table access.

- `Handler_prepare`

  A counter for the prepare phase of two-phase commit operations.

- `Handler_read_first`

  The number of times the first entry in an index was read. If this value is high, it suggests that the server is doing a lot of full index scans (for example, `SELECT col1 FROM foo`, assuming that `col1` is indexed).

- `Handler_read_key`

The number of requests to read a row based on a key. If this value is high, it is a good indication that your tables are properly indexed for your queries.

- `Handler_read_last`

  The number of requests to read the last key in an index. With `ORDER BY`, the server will issue a first-key request followed by several next-key requests, whereas with `ORDER BY DESC`, the server will issue a last-key request followed by several previous-key requests.

- `Handler_read_next`

  The number of requests to read the next row in key order. This value is incremented if you are querying an index column with a range constraint or if you are doing an index scan.

- `Handler_read_prev`

  The number of requests to read the previous row in key order. This read method is mainly used to optimize `ORDER BY ... DESC`.

- `Handler_read_rnd`

  The number of requests to read a row based on a fixed position. This value is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan entire tables or you have joins that do not use keys properly.

- `Handler_read_rnd_next`

  The number of requests to read the next row in the data file. This value is high if you are doing a lot of table scans. Generally this suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.

- `Handler_rollback`

  The number of requests for a storage engine to perform a rollback operation.

- `Handler_savepoint`

  The number of requests for a storage engine to place a savepoint.

- `Handler_savepoint_rollback`

  The number of requests for a storage engine to roll back to a savepoint.

- `Handler_update`

  The number of requests to update a row in a table.

- `Handler_write`

  The number of requests to insert a row in a table.

- `Innodb_available_undo_logs`

  `Innodb_available_undo_logs` was removed in MySQL 8.0.2. The number of available rollback segments per tablespace may be retrieved using `SHOW VARIABLES LIKE 'innodb_rollback_segments';`

- `Innodb_buffer_pool_dump_status`

The progress of an operation to record the pages held in the InnoDB buffer pool, triggered by the setting of innodb_buffer_pool_dump_at_shutdown or innodb_buffer_pool_dump_now.

For related information and examples, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- Innodb_buffer_pool_load_status

  The progress of an operation to warm up the InnoDB buffer pool by reading in a set of pages corresponding to an earlier point in time, triggered by the setting of innodb_buffer_pool_load_at_startup or innodb_buffer_pool_load_now. If the operation introduces too much overhead, you can cancel it by setting innodb_buffer_pool_load_abort.

  For related information and examples, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- Innodb_buffer_pool_bytes_data

  The total number of bytes in the InnoDB buffer pool containing data. The number includes both dirty and clean pages. For more accurate memory usage calculations than with Innodb_buffer_pool_pages_data, when compressed tables cause the buffer pool to hold pages of different sizes.

- Innodb_buffer_pool_pages_data

  The number of pages in the InnoDB buffer pool containing data. The number includes both dirty and clean pages. When using compressed tables, the reported Innodb_buffer_pool_pages_data value may be larger than Innodb_buffer_pool_pages_total (Bug #59550).

- Innodb_buffer_pool_bytes_dirty

  The total current number of bytes held in dirty pages in the InnoDB buffer pool. For more accurate memory usage calculations than with Innodb_buffer_pool_pages_dirty, when compressed tables cause the buffer pool to hold pages of different sizes.

- Innodb_buffer_pool_pages_dirty

  The current number of dirty pages in the InnoDB buffer pool.

- Innodb_buffer_pool_pages_flushed

  The number of requests to flush pages from the InnoDB buffer pool.

- Innodb_buffer_pool_pages_free

  The number of free pages in the InnoDB buffer pool.

- Innodb_buffer_pool_pages_latched

  The number of latched pages in the InnoDB buffer pool. These are pages currently being read or written, or that cannot be flushed or removed for some other reason. Calculation of this variable is expensive, so it is available only when the UNIV_DEBUG system is defined at server build time.

- Innodb_buffer_pool_pages_misc

  The number of pages in the InnoDB buffer pool that are busy because they have been allocated for administrative overhead, such as row locks or the adaptive hash

index. This value can also be calculated as `Innodb_buffer_pool_pages_total` − `Innodb_buffer_pool_pages_free` − `Innodb_buffer_pool_pages_data`. When using compressed tables, `Innodb_buffer_pool_pages_misc` may report an out-of-bounds value (Bug #59550).

- `Innodb_buffer_pool_pages_total`

  The total size of the `InnoDB` buffer pool, in pages. When using compressed tables, the reported `Innodb_buffer_pool_pages_data` value may be larger than `Innodb_buffer_pool_pages_total` (Bug #59550)

- `Innodb_buffer_pool_read_ahead`

  The number of pages read into the `InnoDB` buffer pool by the read-ahead background thread.

- `Innodb_buffer_pool_read_ahead_evicted`

  The number of pages read into the `InnoDB` buffer pool by the read-ahead background thread that were subsequently evicted without having been accessed by queries.

- `Innodb_buffer_pool_read_ahead_rnd`

  The number of “random” read-aheads initiated by `InnoDB`. This happens when a query scans a large portion of a table but in random order.

- `Innodb_buffer_pool_read_requests`

  The number of logical read requests.

- `Innodb_buffer_pool_reads`

  The number of logical reads that `InnoDB` could not satisfy from the buffer pool, and had to read directly from disk.

- `Innodb_buffer_pool_resize_status`

  The status of an operation to resize the `InnoDB` buffer pool dynamically, triggered by setting the `innodb_buffer_pool_size` parameter dynamically. The `innodb_buffer_pool_size` parameter is dynamic, which allows you to resize the buffer pool without restarting the server. See Configuring InnoDB Buffer Pool Size Online for related information.

- `Innodb_buffer_pool_wait_free`

  Normally, writes to the `InnoDB` buffer pool happen in the background. When `InnoDB` needs to read or create a page and no clean pages are available, `InnoDB` flushes some dirty pages first and waits for that operation to finish. This counter counts instances of these waits. If `innodb_buffer_pool_size` has been set properly, this value should be small.

- `Innodb_buffer_pool_write_requests`

  The number of writes done to the `InnoDB` buffer pool.

- `Innodb_data_fsyncs`

  The number of `fsync()` operations so far. The frequency of `fsync()` calls is influenced by the setting of the `innodb_flush_method` configuration option.

- `Innodb_data_pending_fsyncs`

The current number of pending `fsync()` operations. The frequency of `fsync()` calls is influenced by the setting of the `innodb_flush_method` configuration option.

- `Innodb_data_pending_reads`

  The current number of pending reads.

- `Innodb_data_pending_writes`

  The current number of pending writes.

- `Innodb_data_read`

  The amount of data read since the server was started (in bytes).

- `Innodb_data_reads`

  The total number of data reads (OS file reads).

- `Innodb_data_writes`

  The total number of data writes.

- `Innodb_data_written`

  The amount of data written so far, in bytes.

- `Innodb_dblwr_pages_written`

  The number of pages that have been written to the doublewrite buffer. See Section 15.11.1, “InnoDB Disk I/O”.

- `Innodb_dblwr_writes`

  The number of doublewrite operations that have been performed. See Section 15.11.1, “InnoDB Disk I/O”.

- `Innodb_have_atomic_builtins`

  Indicates whether the server was built with atomic instructions.

- `Innodb_log_waits`

  The number of times that the log buffer was too small and a wait was required for it to be flushed before continuing.

- `Innodb_log_write_requests`

  The number of write requests for the `InnoDB` redo log.

- `Innodb_log_writes`

  The number of physical writes to the `InnoDB` redo log file.

- `Innodb_num_open_files`

  The number of files `InnoDB` currently holds open.

- `Innodb_os_log_fsyncs`

The number of `fsync()` writes done to the `InnoDB` redo log files.

- `Innodb_os_log_pending_fsyncs`

  The number of pending `fsync()` operations for the `InnoDB` redo log files.

- `Innodb_os_log_pending_writes`

  The number of pending writes to the `InnoDB` redo log files.

- `Innodb_os_log_written`

  The number of bytes written to the `InnoDB` redo log files.

- `Innodb_page_size`

  `InnoDB` page size (default 16KB). Many values are counted in pages; the page size enables them to be easily converted to bytes.

- `Innodb_pages_created`

  The number of pages created by operations on `InnoDB` tables.

- `Innodb_pages_read`

  The number of pages read from the `InnoDB` buffer pool by operations on `InnoDB` tables.

- `Innodb_pages_written`

  The number of pages written by operations on `InnoDB` tables.

- `Innodb_row_lock_current_waits`

  The number of row locks currently being waited for by operations on `InnoDB` tables.

- `Innodb_row_lock_time`

  The total time spent in acquiring row locks for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_time_avg`

  The average time to acquire a row lock for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_time_max`

  The maximum time to acquire a row lock for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_waits`

  The number of times operations on `InnoDB` tables had to wait for a row lock.

- `Innodb_rows_deleted`

  The number of rows deleted from `InnoDB` tables.

- `Innodb_rows_inserted`

  The number of rows inserted into `InnoDB` tables.

- `Innodb_rows_read`

  The number of rows read from `InnoDB` tables.

- `Innodb_rows_updated`

  The number of rows updated in `InnoDB` tables.

- `Innodb_system_rows_deleted`

  The number of rows deleted from `InnoDB` tables belonging to system-created schemas.

- `Innodb_system_rows_inserted`

  The number of rows inserted into `InnoDB` tables belonging to system-created schemas.

- `Innodb_system_rows_read`

  The number of rows read from `InnoDB` tables belonging to system-created schemas.

- `Innodb_truncated_status_writes`

  The number of times output from the `SHOW ENGINE INNODB STATUS` statement has been truncated.

- `Key_blocks_not_flushed`

  The number of key blocks in the `MyISAM` key cache that have changed but have not yet been flushed to disk.

- `Key_blocks_unused`

  The number of unused blocks in the `MyISAM` key cache. You can use this value to determine how much of the key cache is in use; see the discussion of `key_buffer_size` in Section 5.1.8, “Server System Variables”.

- `Key_blocks_used`

  The number of used blocks in the `MyISAM` key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.

- `Key_read_requests`

  The number of requests to read a key block from the `MyISAM` key cache.

- `Key_reads`

  The number of physical reads of a key block from disk into the `MyISAM` key cache. If `Key_reads` is large, then your `key_buffer_size` value is probably too small. The cache miss rate can be calculated as `Key_reads`/`Key_read_requests`.

- `Key_write_requests`

  The number of requests to write a key block to the `MyISAM` key cache.

- `Key_writes`

  The number of physical writes of a key block from the `MyISAM` key cache to disk.

- `Last_query_cost`

The total cost of the last compiled query as computed by the query optimizer. This is useful for comparing the cost of different query plans for the same query. The default value of 0 means that no query has been compiled yet. The default value is 0. `Last_query_cost` has session scope.

In MySQL 8.0.16 and later, this variable shows the cost of queries that have multiple query blocks, summing the cost estimates of each query block, estimating how many times non-cacheable subqueries are executed, and multiplying the cost of those query blocks by the number of subquery executions. (Bug #92766, Bug #28786951) Prior to MySQL 8.0.16, `Last_query_cost` was computed accurately only for simple, "flat" queries, but not for complex queries such as those containing subqueries or `UNION`. (For the latter, the value was set to 0.)

- `Last_query_partial_plans`

  The number of iterations the query optimizer made in execution plan construction for the previous query. `Last_query_cost` has session scope.

- `Locked_connects`

  The number of attempts to connect to locked user accounts. For information about account locking and unlocking, see Section 6.2.19, "Account Locking".

- `Max_execution_time_exceeded`

  The number of `SELECT` statements for which the execution timeout was exceeded.

- `Max_execution_time_set`

  The number of `SELECT` statements for which a nonzero execution timeout was set. This includes statements that include a nonzero `MAX_EXECUTION_TIME` optimizer hint, and statements that include no such hint but execute while the timeout indicated by the `max_execution_time` system variable is nonzero.

- `Max_execution_time_set_failed`

  The number of `SELECT` statements for which the attempt to set an execution timeout failed.

- `Max_used_connections`

  The maximum number of connections that have been in use simultaneously since the server started.

- `Max_used_connections_time`

  The time at which `Max_used_connections` reached its current value.

- `Not_flushed_delayed_rows`

  This status variable is deprecated (because `DELAYED` inserts are not supported), and will be removed in a future release.

- `mecab_charset`

  The character set currently used by the MeCab full-text parser plugin. For related information, see Section 12.9.9, "MeCab Full-Text Parser Plugin".

- `Ongoing_anonymous_transaction_count`

  Shows the number of ongoing transactions which have been marked as anonymous. This can be used to ensure that no further transactions are waiting to be processed.

- `Ongoing_anonymous_gtid_violating_transaction_count`

  This status variable is only available in debug builds. Shows the number of ongoing transactions which use `gtid_next=ANONYMOUS` and that violate GTID consistency.

- `Ongoing_automatic_gtid_violating_transaction_count`

  This status variable is only available in debug builds. Shows the number of ongoing transactions which use `gtid_next=AUTOMATIC` and that violate GTID consistency.

- `Open_files`

  The number of files that are open. This count includes regular files opened by the server. It does not include other types of files such as sockets or pipes. Also, the count does not include files that storage engines open using their own internal functions rather than asking the server level to do so.

- `Open_streams`

  The number of streams that are open (used mainly for logging).

- `Open_table_definitions`

  The number of cached table definitions.

- `Open_tables`

  The number of tables that are open.

- `Opened_files`

  The number of files that have been opened with `my_open()` (a `mysys` library function). Parts of the server that open files without using this function do not increment the count.

- `Opened_table_definitions`

  The number of table definitions that have been cached.

- `Opened_tables`

  The number of tables that have been opened. If `Opened_tables` is big, your `table_open_cache` value is probably too small.

- `Performance_schema_`*`xxx`*

  Performance Schema status variables are listed in Section 26.16, “Performance Schema Status Variables”. These variables provide information about instrumentation that could not be loaded or created due to memory constraints.

- `Prepared_stmt_count`

  The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_stmt_count` system variable.)

- `Qcache_free_blocks`

  This status variable was removed in MySQL 8.0.3.

- `Qcache_free_memory`

This status variable was removed in MySQL 8.0.3.

- `Qcache_hits`

  This status variable was removed in MySQL 8.0.3.

- `Qcache_inserts`

  This status variable was removed in MySQL 8.0.3.

- `Qcache_lowmem_prunes`

  This status variable was removed in MySQL 8.0.3.

- `Qcache_not_cached`

  This status variable was removed in MySQL 8.0.3.

- `Qcache_queries_in_cache`

  This status variable was removed in MySQL 8.0.3.

- `Qcache_total_blocks`

  This status variable was removed in MySQL 8.0.3.

- `Queries`

  The number of statements executed by the server. This variable includes statements executed within stored programs, unlike the `Questions` variable. It does not count `COM_PING` or `COM_STATISTICS` commands.

  The discussion at the beginning of this section indicates how to relate this statement-counting status variable to other such variables.

- `Questions`

  The number of statements executed by the server. This includes only statements sent to the server by clients and not statements executed within stored programs, unlike the `Queries` variable. This variable does not count `COM_PING`, `COM_STATISTICS`, `COM_STMT_PREPARE`, `COM_STMT_CLOSE`, or `COM_STMT_RESET` commands.

  The discussion at the beginning of this section indicates how to relate this statement-counting status variable to other such variables.

- `Rpl_semi_sync_master_clients`

  The number of semisynchronous slaves.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_net_avg_wait_time`

  The average time in microseconds the master waited for a slave reply. This variable is always `0`, is deprecated and it will be removed in a future version.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_net_wait_time`

The total time in microseconds the master waited for slave replies. This variable is always `0`, is deprecated and it will be removed in a future version.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_net_waits`

  The total number of times the master waited for slave replies.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_no_times`

  The number of times the master turned off semisynchronous replication.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_no_tx`

  The number of commits that were not acknowledged successfully by a slave.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_status`

  Whether semisynchronous replication currently is operational on the master. The value is `ON` if the plugin has been enabled and a commit acknowledgment has occurred. It is `OFF` if the plugin is not enabled or the master has fallen back to asynchronous replication due to commit acknowledgment timeout.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_timefunc_failures`

  The number of times the master failed when calling time functions such as `gettimeofday()`.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_tx_avg_wait_time`

  The average time in microseconds the master waited for each transaction.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_tx_wait_time`

  The total time in microseconds the master waited for transactions.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_tx_waits`

  The total number of times the master waited for transactions.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_wait_pos_backtraverse`

The total number of times the master waited for an event with binary coordinates lower than events waited for previously. This can occur when the order in which transactions start waiting for a reply is different from the order in which their binary log events are written.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_wait_sessions`

  The number of sessions currently waiting for slave replies.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_yes_tx`

  The number of commits that were acknowledged successfully by a slave.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_slave_status`

  Whether semisynchronous replication currently is operational on the slave. This is `ON` if the plugin has been enabled and the slave I/O thread is running, `OFF` otherwise.

  This variable is available only if the slave-side semisynchronous replication plugin is installed.

- `Rsa_public_key`

  The value of this variable is the public key used by the `sha256_password` authentication plugin for RSA key pair-based password exchange. The value is nonempty only if the server successfully initializes the private and public keys in the files named by the `sha256_password_private_key_path` and `sha256_password_public_key_path` system variables. The value of `Rsa_public_key` comes from the latter file.

  For information about `sha256_password`, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”.

- `Secondary_engine_execution_count`

  For future use. This variable was added in MySQL 8.0.13.

- `Select_full_join`

  The number of joins that perform table scans because they do not use indexes. If this value is not 0, you should carefully check the indexes of your tables.

- `Select_full_range_join`

  The number of joins that used a range search on a reference table.

- `Select_range`

  The number of joins that used ranges on the first table. This is normally not a critical issue even if the value is quite large.

- `Select_range_check`

  The number of joins without keys that check for key usage after each row. If this is not 0, you should carefully check the indexes of your tables.

- `Select_scan`

  The number of joins that did a full scan of the first table.

- `Slave_heartbeat_period`

  This variable is obsolete and was removed in MySQL 8.0.1. Instead, use the `HEARTBEAT_INTERVAL` column of the `replication_connection_configuration` table.

- `Slave_last_heartbeat`

  This variable is obsolete and was removed in MySQL 8.0.1. Instead, use the `LAST_HEARTBEAT_TIMESTAMP` column of the `replication_connection_status` table.

- `Slave_open_temp_tables`

  The number of temporary tables that the slave SQL thread currently has open. If the value is greater than zero, it is not safe to shut down the slave; see Section 17.5.1.30, “Replication and Temporary Tables”. This variable reports the total count of open temporary tables for *all* replication channels.

- `Slave_received_heartbeats`

  This variable is obsolete and was removed in MySQL 8.0.1. Instead, use the `COUNT_RECEIVED_HEARTBEATS` column of the `replication_connection_status` table.

- `Slave_retried_transactions`

  This variable is obsolete and was removed in MySQL 8.0.1. Instead, use the `COUNT_TRANSACTIONS_RETRIES` column of the `replication_applier_status` table.

- `Slave_rows_last_search_algorithm_used`

  The search algorithm that was most recently used by this slave to locate rows for row-based replication. The result shows whether the slave used indexes, a table scan, or hashing as the search algorithm for the last transaction executed on any channel.

  The method used depends on the setting for the `slave_rows_search_algorithms` system variable, and the keys that are available on the relevant table.

  This variable is available only for debug builds of MySQL.

- `Slave_running`

  This variable is obsolete and was removed in MySQL 8.0.1. Instead, use the `SERVICE_STATE` column of the `replication_connection_status` and `replication_applier_status` tables.

- `Slow_launch_threads`

  The number of threads that have taken more than `slow_launch_time` seconds to create.

- `Slow_queries`

  The number of queries that have taken more than `long_query_time` seconds. This counter increments regardless of whether the slow query log is enabled. For information about that log, see Section 5.4.5, “The Slow Query Log”.

- `Sort_merge_passes`

The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the `sort_buffer_size` system variable.

- `Sort_range`

  The number of sorts that were done using ranges.

- `Sort_rows`

  The number of sorted rows.

- `Sort_scan`

  The number of sorts that were done by scanning the table.

- `Ssl_accept_renegotiates`

  The number of negotiates needed to establish the connection.

- `Ssl_accepts`

  The number of accepted SSL connections.

- `Ssl_callback_cache_hits`

  The number of callback cache hits.

- `Ssl_cipher`

  The current encryption cipher (empty for unencrypted connections).

- `Ssl_cipher_list`

  The list of possible SSL ciphers (empty for non-SSL connections). If MySQL supports TLSv1.3, the value includes the possible TLSv1.3 ciphersuites. See Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `Ssl_client_connects`

  The number of SSL connection attempts to an SSL-enabled master.

- `Ssl_connect_renegotiates`

  The number of negotiates needed to establish the connection to an SSL-enabled master.

- `Ssl_ctx_verify_depth`

  The SSL context verification depth (how many certificates in the chain are tested).

- `Ssl_ctx_verify_mode`

  The SSL context verification mode.

- `Ssl_default_timeout`

  The default SSL timeout.

- `Ssl_finished_accepts`

The number of successful SSL connections to the server.

- `Ssl_finished_connects`

  The number of successful slave connections to an SSL-enabled master.

- `Ssl_server_not_after`

  The last date for which the SSL certificate is valid. To check SSL certificate expiration information, use this statement:

```
mysql> SHOW STATUS LIKE 'Ssl_server_not%';
+-----------------------+--------------------------+
| Variable_name         | Value                    |
+-----------------------+--------------------------+
| Ssl_server_not_after  | Apr 28 14:16:39 2025 GMT |
| Ssl_server_not_before | May  1 14:16:39 2015 GMT |
+-----------------------+--------------------------+
```

- `Ssl_server_not_before`

  The first date for which the SSL certificate is valid.

- `Ssl_session_cache_hits`

  The number of SSL session cache hits.

- `Ssl_session_cache_misses`

  The number of SSL session cache misses.

- `Ssl_session_cache_mode`

  The SSL session cache mode.

- `Ssl_session_cache_overflows`

  The number of SSL session cache overflows.

- `Ssl_session_cache_size`

  The SSL session cache size.

- `Ssl_session_cache_timeouts`

  The number of SSL session cache timeouts.

- `Ssl_sessions_reused`

  How many SSL connections were reused from the cache.

- `Ssl_used_session_cache_entries`

  How many SSL session cache entries were used.

- `Ssl_verify_depth`

  The verification depth for replication SSL connections.

- `Ssl_verify_mode`

The verification mode used by the server for a connection that uses SSL. The value is a bitmask; bits are defined in the `openssl/ssl.h` header file:

```
# define SSL_VERIFY_NONE                 0x00
# define SSL_VERIFY_PEER                 0x01
# define SSL_VERIFY_FAIL_IF_NO_PEER_CERT 0x02
# define SSL_VERIFY_CLIENT_ONCE          0x04
```

`SSL_VERIFY_PEER` indicates that the server asks for a client certificate. If the client supplies one, the server performs verification and proceeds only if verification is successful. `SSL_VERIFY_CLIENT_ONCE` indicates that a request for the client certificate will be done only in the initial handshake.

- `Ssl_version`

  The SSL protocol version of the connection (for example, TLSv1). If the connection is not encrypted, the value is empty.

- `Table_locks_immediate`

  The number of times that a request for a table lock could be granted immediately.

- `Table_locks_waited`

  The number of times that a request for a table lock could not be granted immediately and a wait was needed. If this is high and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication.

- `Table_open_cache_hits`

  The number of hits for open tables cache lookups.

- `Table_open_cache_misses`

  The number of misses for open tables cache lookups.

- `Table_open_cache_overflows`

  The number of overflows for the open tables cache. This is the number of times, after a table is opened or closed, a cache instance has an unused entry and the size of the instance is larger than `table_open_cache` / `table_open_cache_instances`.

- `Tc_log_max_pages_used`

  For the memory-mapped implementation of the log that is used by `mysqld` when it acts as the transaction coordinator for recovery of internal XA transactions, this variable indicates the largest number of pages used for the log since the server started. If the product of `Tc_log_max_pages_used` and `Tc_log_page_size` is always significantly less than the log size, the size is larger than necessary and can be reduced. (The size is set by the `--log-tc-size` option. This variable is unused: It is unneeded for binary log-based recovery, and the memory-mapped recovery log method is not used unless the number of storage engines that are capable of two-phase commit and that support XA transactions is greater than one. (`InnoDB` is the only applicable engine.)

- `Tc_log_page_size`

  The page size used for the memory-mapped implementation of the XA recovery log. The default value is determined using `getpagesize()`. This variable is unused for the same reasons as described for `Tc_log_max_pages_used`.

- `Tc_log_page_waits`

  For the memory-mapped implementation of the recovery log, this variable increments each time the server was not able to commit a transaction and had to wait for a free page in the log. If this value is large, you might want to increase the log size (with the `--log-tc-size` option). For binary log-based recovery, this variable increments each time the binary log cannot be closed because there are two-phase commits in progress. (The close operation waits until all such transactions are finished.)

- `Threads_cached`

  The number of threads in the thread cache.

- `Threads_connected`

  The number of currently open connections.

- `Threads_created`

  The number of threads created to handle connections. If `Threads_created` is big, you may want to increase the `thread_cache_size` value. The cache miss rate can be calculated as `Threads_created`/`Connections`.

- `Threads_running`

  The number of threads that are not sleeping.

- `Uptime`

  The number of seconds that the server has been up.

- `Uptime_since_flush_status`

  The number of seconds since the most recent `FLUSH STATUS` statement.

5.1.11 Server SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients, depending on the value of the `sql_mode` system variable. DBAs can set the global SQL mode to match site server operating requirements, and each application can set its session SQL mode to its own requirements.

Modes affect the SQL syntax MySQL supports and the data validation checks it performs. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

- Setting the SQL Mode
- The Most Important SQL Modes
- Full List of SQL Modes
- Combination SQL Modes
- Strict SQL Mode
- Comparison of the IGNORE Keyword and Strict SQL Mode

For answers to questions often asked about server SQL modes in MySQL, see Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”.

When working with InnoDB tables, consider also the innodb_strict_mode system variable. It enables additional error checks for InnoDB tables.

Setting the SQL Mode

The default SQL mode in MySQL 8.0 includes these modes: ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, and NO_ENGINE_SUBSTITUTION.

To set the SQL mode at server startup, use the --sql-mode="*modes*" option on the command line, or sql-mode="*modes*" in an option file such as my.cnf (Unix operating systems) or my.ini (Windows). *modes* is a list of different modes separated by commas. To clear the SQL mode explicitly, set it to an empty string using --sql-mode="" on the command line, or sql-mode="" in an option file.

Note

MySQL installation programs may configure the SQL mode during the installation process.

If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

To change the SQL mode at runtime, set the global or session sql_mode system variable using a SET statement:

```
SET GLOBAL sql_mode = 'modes';
SET SESSION sql_mode = 'modes';
```

Setting the GLOBAL variable requires the SYSTEM_VARIABLES_ADMIN or SUPER privilege and affects the operation of all clients that connect from that time on. Setting the SESSION variable affects only the current client. Each client can change its session sql_mode value at any time.

To determine the current global or session sql_mode setting, select its value:

```
SELECT @@GLOBAL.sql_mode;
SELECT @@SESSION.sql_mode;
```

Important

SQL mode and user-defined partitioning. Changing the server SQL mode after creating and inserting data into partitioned tables can cause major changes in the behavior of such tables, and could lead to loss or corruption of data. It is strongly recommended that you never change the SQL mode once you have created tables employing user-defined partitioning.

When replicating partitioned tables, differing SQL modes on the master and slave can also lead to problems. For best results, you should always use the same server SQL mode on the master and slave.

For more information, see Section 23.6, “Restrictions and Limitations on Partitioning”.

The Most Important SQL Modes

The most important sql_mode values are probably these:

- ANSI

This mode changes syntax and behavior to conform more closely to standard SQL. It is one of the special combination modes listed at the end of this section.

- `STRICT_TRANS_TABLES`

  If a value could not be inserted as given into a transactional table, abort the statement. For a nontransactional table, abort the statement if the value occurs in a single-row statement or the first row of a multiple-row statement. More details are given later in this section.

- `TRADITIONAL`

  Make MySQL behave like a “traditional” SQL database system. A simple description of this mode is “give an error instead of a warning” when inserting an incorrect value into a column. It is one of the special combination modes listed at the end of this section.

  Note

  With `TRADITIONAL` mode enabled, an `INSERT` or `UPDATE` aborts as soon as an error occurs. If you are using a nontransactional storage engine, this may not be what you want because data changes made prior to the error may not be rolled back, resulting in a “partially done” update.

When this manual refers to “strict mode,” it means a mode with either or both `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` enabled.

Full List of SQL Modes

The following list describes all supported SQL modes:

- `ALLOW_INVALID_DATES`

  Do not perform full checking of dates. Check only that the month is in the range from 1 to 12 and the day is in the range from 1 to 31. This may be useful for Web applications that obtain year, month, and day in three different fields and store exactly what the user inserted, without date validation. This mode applies to `DATE` and `DATETIME` columns. It does not apply `TIMESTAMP` columns, which always require a valid date.

  With `ALLOW_INVALID_DATES` disabled, the server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as `'2004-04-31'` are converted to `'0000-00-00'` and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable `ALLOW_INVALID_DATES`.

- `ANSI_QUOTES`

  Treat `"` as an identifier quote character (like the `` ` `` quote character) and not as a string quote character. You can still use `` ` `` to quote identifiers with this mode enabled. With `ANSI_QUOTES` enabled, you cannot use double quotation marks to quote literal strings because they are interpreted as identifiers.

- `ERROR_FOR_DIVISION_BY_ZERO`

  The `ERROR_FOR_DIVISION_BY_ZERO` mode affects handling of division by zero, which includes `MOD(N,0)`. For data-change operations (`INSERT`, `UPDATE`), its effect also depends on whether strict SQL mode is enabled.

  - If this mode is not enabled, division by zero inserts `NULL` and produces no warning.

- If this mode is enabled, division by zero inserts `NULL` and produces a warning.
- If this mode and strict mode are enabled, division by zero produces an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, division by zero inserts `NULL` and produces a warning.

For `SELECT`, division by zero returns `NULL`. Enabling `ERROR_FOR_DIVISION_BY_ZERO` causes a warning to be produced as well, regardless of whether strict mode is enabled.

`ERROR_FOR_DIVISION_BY_ZERO` is deprecated. `ERROR_FOR_DIVISION_BY_ZERO` is not part of strict mode, but should be used in conjunction with strict mode and is enabled by default. A warning occurs if `ERROR_FOR_DIVISION_BY_ZERO` is enabled without also enabling strict mode or vice versa.

Because `ERROR_FOR_DIVISION_BY_ZERO` is deprecated, it will be removed in a future MySQL release as a separate mode name and its effect included in the effects of strict SQL mode.

- `HIGH_NOT_PRECEDENCE`

  The precedence of the `NOT` operator is such that expressions such as `NOT a BETWEEN b AND c` are parsed as `NOT (a BETWEEN b AND c)`. In some older versions of MySQL, the expression was parsed as `(NOT a) BETWEEN b AND c`. The old higher-precedence behavior can be obtained by enabling the `HIGH_NOT_PRECEDENCE` SQL mode.

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 0
mysql> SET sql_mode = 'HIGH_NOT_PRECEDENCE';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 1
```

- `IGNORE_SPACE`

  Permit spaces between a function name and the `(` character. This causes built-in function names to be treated as reserved words. As a result, identifiers that are the same as function names must be quoted as described in Section 9.2, “Schema Object Names”. For example, because there is a `COUNT()` function, the use of `count` as a table name in the following statement causes an error:

```
mysql> CREATE TABLE count (i INT);
ERROR 1064 (42000): You have an error in your SQL syntax
```

  The table name should be quoted:

```
mysql> CREATE TABLE `count` (i INT);
Query OK, 0 rows affected (0.00 sec)
```

  The `IGNORE_SPACE` SQL mode applies to built-in functions, not to user-defined functions or stored functions. It is always permissible to have spaces after a UDF or stored function name, regardless of whether `IGNORE_SPACE` is enabled.

  For further discussion of `IGNORE_SPACE`, see Section 9.2.5, “Function Name Parsing and Resolution”.

- `NO_AUTO_VALUE_ON_ZERO`

NO_AUTO_VALUE_ON_ZERO affects handling of AUTO_INCREMENT columns. Normally, you generate the next sequence number for the column by inserting either NULL or 0 into it. NO_AUTO_VALUE_ON_ZERO suppresses this behavior for 0 so that only NULL generates the next sequence number.

This mode can be useful if 0 has been stored in a table's AUTO_INCREMENT column. (Storing 0 is not a recommended practice, by the way.) For example, if you dump the table with mysqldump and then reload it, MySQL normally generates new sequence numbers when it encounters the 0 values, resulting in a table with contents different from the one that was dumped. Enabling NO_AUTO_VALUE_ON_ZERO before reloading the dump file solves this problem. For this reason, mysqldump automatically includes in its output a statement that enables NO_AUTO_VALUE_ON_ZERO.

- NO_BACKSLASH_ESCAPES

  Disable the use of the backslash character (\) as an escape character within strings and identifiers. With this mode enabled, backslash becomes an ordinary character like any other.

- NO_DIR_IN_CREATE

  When creating a table, ignore all INDEX DIRECTORY and DATA DIRECTORY directives. This option is useful on slave replication servers.

- NO_ENGINE_SUBSTITUTION

  Control automatic substitution of the default storage engine when a statement such as CREATE TABLE or ALTER TABLE specifies a storage engine that is disabled or not compiled in.

  By default, NO_ENGINE_SUBSTITUTION is enabled.

  Because storage engines can be pluggable at runtime, unavailable engines are treated the same way:

  With NO_ENGINE_SUBSTITUTION disabled, for CREATE TABLE the default engine is used and a warning occurs if the desired engine is unavailable. For ALTER TABLE, a warning occurs and the table is not altered.

  With NO_ENGINE_SUBSTITUTION enabled, an error occurs and the table is not created or altered if the desired engine is unavailable.

- NO_UNSIGNED_SUBTRACTION

  Subtraction between integer values, where one is of type UNSIGNED, produces an unsigned result by default. If the result would otherwise have been negative, an error results:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CAST(0 AS UNSIGNED) - 1;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(cast(0 as unsigned) - 1)'
```

  If the NO_UNSIGNED_SUBTRACTION SQL mode is enabled, the result is negative:

```
mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|                      -1 |
+-------------------------+
```

If the result of such an operation is used to update an `UNSIGNED` integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if `NO_UNSIGNED_SUBTRACTION` is enabled. With strict SQL mode enabled, an error occurs and the column remains unchanged.

When `NO_UNSIGNED_SUBTRACTION` is enabled, the subtraction result is signed, *even if any operand is unsigned*. For example, compare the type of column `c2` in table `t1` with that of column `c2` in table `t2`:

```
mysql> SET sql_mode='';
mysql> CREATE TABLE test (c1 BIGINT UNSIGNED NOT NULL);
mysql> CREATE TABLE t1 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t1;
+-------+---------------------+------+-----+---------+-------+
| Field | Type                | Null | Key | Default | Extra |
+-------+---------------------+------+-----+---------+-------+
| c2    | bigint(21) unsigned | NO   |     | 0       |       |
+-------+---------------------+------+-----+---------+-------+

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
mysql> CREATE TABLE t2 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t2;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| c2    | bigint(21) | NO   |     | 0       |       |
+-------+------------+------+-----+---------+-------+
```

This means that `BIGINT UNSIGNED` is not 100% usable in all contexts. See Section 12.10, “Cast Functions and Operators”.

- `NO_ZERO_DATE`

  The `NO_ZERO_DATE` mode affects whether the server permits `'0000-00-00'` as a valid date. Its effect also depends on whether strict SQL mode is enabled.

  - If this mode is not enabled, `'0000-00-00'` is permitted and inserts produce no warning.
  - If this mode is enabled, `'0000-00-00'` is permitted and inserts produce a warning.
  - If this mode and strict mode are enabled, `'0000-00-00'` is not permitted and inserts produce an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, `'0000-00-00'` is permitted and inserts produce a warning.

  `NO_ZERO_DATE` is deprecated. `NO_ZERO_DATE` is not part of strict mode, but should be used in conjunction with strict mode and is enabled by default. A warning occurs if `NO_ZERO_DATE` is enabled without also enabling strict mode or vice versa.

  Because `NO_ZERO_DATE` is deprecated, it will be removed in a future MySQL release as a separate mode name and its effect included in the effects of strict SQL mode.

- `NO_ZERO_IN_DATE`

  The `NO_ZERO_IN_DATE` mode affects whether the server permits dates in which the year part is nonzero but the month or day part is 0. (This mode affects dates such as `'2010-00-01'` or `'2010-01-00'`, but not `'0000-00-00'`. To control whether the server permits `'0000-00-00'`, use the `NO_ZERO_DATE` mode.) The effect of `NO_ZERO_IN_DATE` also depends on whether strict SQL mode is enabled.

- If this mode is not enabled, dates with zero parts are permitted and inserts produce no warning.
- If this mode is enabled, dates with zero parts are inserted as `'0000-00-00'` and produce a warning.
- If this mode and strict mode are enabled, dates with zero parts are not permitted and inserts produce an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, dates with zero parts are inserted as `'0000-00-00'` and produce a warning.

`NO_ZERO_IN_DATE` is deprecated. `NO_ZERO_IN_DATE` is not part of strict mode, but should be used in conjunction with strict mode and is enabled by default. A warning occurs if `NO_ZERO_IN_DATE` is enabled without also enabling strict mode or vice versa.

Because `NO_ZERO_IN_DATE` is deprecated, it will be removed in a future MySQL release as a separate mode name and its effect included in the effects of strict SQL mode.

- `ONLY_FULL_GROUP_BY`

  Reject queries for which the select list, `HAVING` condition, or `ORDER BY` list refer to nonaggregated columns that are neither named in the `GROUP BY` clause nor are functionally dependent on (uniquely determined by) `GROUP BY` columns.

  A MySQL extension to standard SQL permits references in the `HAVING` clause to aliased expressions in the select list. The `HAVING` clause can refer to aliases regardless of whether `ONLY_FULL_GROUP_BY` is enabled.

  For additional discussion and examples, see Section 12.20.3, “MySQL Handling of GROUP BY”.

- `PAD_CHAR_TO_FULL_LENGTH`

  By default, trailing spaces are trimmed from `CHAR` column values on retrieval. If `PAD_CHAR_TO_FULL_LENGTH` is enabled, trimming does not occur and retrieved `CHAR` values are padded to their full length. This mode does not apply to `VARCHAR` columns, for which trailing spaces are retained on retrieval.

  Note

  As of MySQL 8.0.13, `PAD_CHAR_TO_FULL_LENGTH` is deprecated. It will be removed in a future version of MySQL.

```
mysql> CREATE TABLE t1 (c1 CHAR(10));
Query OK, 0 rows affected (0.37 sec)

mysql> INSERT INTO t1 (c1) VALUES('xy');
Query OK, 1 row affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+------+----------------+
| c1   | CHAR_LENGTH(c1) |
+------+----------------+
| xy   |              2 |
+------+----------------+
1 row in set (0.00 sec)

mysql> SET sql_mode = 'PAD_CHAR_TO_FULL_LENGTH';
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+------------+-----------------+
| c1         | CHAR_LENGTH(c1) |
+------------+-----------------+
| xy         |              10 |
+------------+-----------------+
1 row in set (0.00 sec)
```

- `PIPES_AS_CONCAT`

  Treat `||` as a string concatenation operator (same as `CONCAT()`) rather than as a synonym for `OR`.

- `REAL_AS_FLOAT`

  Treat `REAL` as a synonym for `FLOAT`. By default, MySQL treats `REAL` as a synonym for `DOUBLE`.

- `STRICT_ALL_TABLES`

  Enable strict SQL mode for all storage engines. Invalid data values are rejected. For details, see Strict SQL Mode.

- `STRICT_TRANS_TABLES`

  Enable strict SQL mode for transactional storage engines, and when possible for nontransactional storage engines. For details, see Strict SQL Mode.

- `TIME_TRUNCATE_FRACTIONAL`

  Control whether rounding or truncation occurs when inserting a `TIME`, `DATE`, or `TIMESTAMP` value with a fractional seconds part into a column having the same type but fewer fractional digits. The default behavior is to use rounding. If this mode is enabled, truncation occurs instead. The following sequence of statements illustrates the difference:

  ```
  CREATE TABLE t (id INT, tval TIME(1));
  SET sql_mode='';
  INSERT INTO t (id, tval) VALUES(1, 1.55);
  SET sql_mode='TIME_TRUNCATE_FRACTIONAL';
  INSERT INTO t (id, tval) VALUES(2, 1.55);
  ```

  The resulting table contents look like this, where the first value has been subject to rounding and the second to truncation:

  ```
  mysql> SELECT id, tval FROM t ORDER BY id;
  +------+------------+
  | id   | tval       |
  +------+------------+
  |    1 | 00:00:01.6 |
  |    2 | 00:00:01.5 |
  +------+------------+
  ```

  See also Section 11.3.5, “Fractional Seconds in Time Values”.

Combination SQL Modes

The following special modes are provided as shorthand for combinations of mode values from the preceding list.

- `ANSI`

Equivalent to REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, and ONLY_FULL_GROUP_BY.

ANSI mode also causes the server to return an error for queries where a set function *S* with an outer reference *S(outer_ref)* cannot be aggregated in the outer query against which the outer reference has been resolved. This is such a query:

```
SELECT * FROM t1 WHERE t1.a IN (SELECT MAX(t1.b) FROM t2 WHERE ...);
```

Here, MAX(t1.b) cannot aggregated in the outer query because it appears in the WHERE clause of that query. Standard SQL requires an error in this situation. If ANSI mode is not enabled, the server treats *S(outer_ref)* in such queries the same way that it would interpret *S(const)*.

See Section 1.8, “MySQL Standards Compliance”.

- TRADITIONAL

  TRADITIONAL is equivalent to STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, and NO_ENGINE_SUBSTITUTION.

Strict SQL Mode

Strict mode controls how MySQL handles invalid or missing values in data-change statements such as INSERT or UPDATE. A value can be invalid for several reasons. For example, it might have the wrong data type for the column, or it might be out of range. A value is missing when a new row to be inserted does not contain a value for a non-NULL column that has no explicit DEFAULT clause in its definition. (For a NULL column, NULL is inserted if the value is missing.) Strict mode also affects DDL statements such as CREATE TABLE.

If strict mode is not in effect, MySQL inserts adjusted values for invalid or missing values and produces warnings (see Section 13.7.7.40, “SHOW WARNINGS Syntax”). In strict mode, you can produce this behavior by using INSERT IGNORE or UPDATE IGNORE.

For statements such as SELECT that do not change data, invalid values generate a warning in strict mode, not an error.

Strict mode produces an error for attempts to create a key that exceeds the maximum key length. When strict mode is not enabled, this results in a warning and truncation of the key to the maximum key length.

Strict mode does not affect whether foreign key constraints are checked. foreign_key_checks can be used for that. (See Section 5.1.8, “Server System Variables”.)

Strict SQL mode is in effect if either STRICT_ALL_TABLES or STRICT_TRANS_TABLES is enabled, although the effects of these modes differ somewhat:

- For transactional tables, an error occurs for invalid or missing values in a data-change statement when either STRICT_ALL_TABLES or STRICT_TRANS_TABLES is enabled. The statement is aborted and rolled back.
- For nontransactional tables, the behavior is the same for either mode if the bad value occurs in the first row to be inserted or updated: The statement is aborted and the table remains unchanged. If the statement inserts or modifies multiple rows and the bad value occurs in the second or later row, the result depends on which strict mode is enabled:

- For `STRICT_ALL_TABLES`, MySQL returns an error and ignores the rest of the rows. However, because the earlier rows have been inserted or updated, the result is a partial update. To avoid this, use single-row statements, which can be aborted without changing the table.
- For `STRICT_TRANS_TABLES`, MySQL converts an invalid value to the closest valid value for the column and inserts the adjusted value. If a value is missing, MySQL inserts the implicit default value for the column data type. In either case, MySQL generates a warning rather than an error and continues processing the statement. Implicit defaults are described in Section 11.7, "Data Type Default Values".

Strict mode affects handling of division by zero, zero dates, and zeros in dates as follows:

- Strict mode affects handling of division by zero, which includes `MOD(N,0)`:

  For data-change operations (`INSERT`, `UPDATE`):

  - If strict mode is not enabled, division by zero inserts `NULL` and produces no warning.
  - If strict mode is enabled, division by zero produces an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, division by zero inserts `NULL` and produces a warning.

  For `SELECT`, division by zero returns `NULL`. Enabling strict mode causes a warning to be produced as well.

- Strict mode affects whether the server permits `'0000-00-00'` as a valid date:
  - If strict mode is not enabled, `'0000-00-00'` is permitted and inserts produce no warning.
  - If strict mode is enabled, `'0000-00-00'` is not permitted and inserts produce an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, `'0000-00-00'` is permitted and inserts produce a warning.
- Strict mode affects whether the server permits dates in which the year part is nonzero but the month or day part is 0 (dates such as `'2010-00-01'` or `'2010-01-00'`):
  - If strict mode is not enabled, dates with zero parts are permitted and inserts produce no warning.
  - If strict mode is enabled, dates with zero parts are not permitted and inserts produce an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, dates with zero parts are inserted as `'0000-00-00'` (which is considered valid with `IGNORE`) and produce a warning.

For more information about strict mode with respect to `IGNORE`, see Comparison of the IGNORE Keyword and Strict SQL Mode.

Strict mode affects handling of division by zero, zero dates, and zeros in dates in conjunction with the `ERROR_FOR_DIVISION_BY_ZERO`, `NO_ZERO_DATE`, and `NO_ZERO_IN_DATE` modes.

Comparison of the IGNORE Keyword and Strict SQL Mode

This section compares the effect on statement execution of the `IGNORE` keyword (which downgrades errors to warnings) and strict SQL mode (which upgrades warnings to errors). It describes which statements they affect, and which errors they apply to.

The following table presents a summary comparison of statement behavior when the default is to produce an error versus a warning. An example of when the default is to produce an error is inserting a `NULL` into a `NOT NULL` column. An example of when the default is to produce a warning is inserting a value of the wrong data type into a column (such as inserting the string `'abc'` into an integer column).

Operational Mode	When Statement Default is Error	When Statement Default is Warning
Without IGNORE or strict SQL mode	Error	Warning
With IGNORE	Warning	Warning (same as without IGNORE or strict SQL mode)
With strict SQL mode	Error (same as without IGNORE or strict SQL mode)	Error
With IGNORE and strict SQL mode	Warning	Warning

One conclusion to draw from the table is that when the IGNORE keyword and strict SQL mode are both in effect, IGNORE takes precedence. This means that, although IGNORE and strict SQL mode can be considered to have opposite effects on error handling, they do not cancel when used together.

The Effect of IGNORE on Statement Execution

Several statements in MySQL support an optional IGNORE keyword. This keyword causes the server to downgrade certain types of errors and generate warnings instead. For a multiple-row statement, IGNORE causes the statement to skip to the next row instead of aborting.

For example, if the table t has a primary key column i, attempting to insert the same value of i into multiple rows normally produces a duplicate-key error:

```
mysql> INSERT INTO t (i) VALUES(1),(1);
ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'
```

With IGNORE, the row containing the duplicate key still is not inserted, but a warning occurs instead of an error:

```
mysql> INSERT IGNORE INTO t (i) VALUES(1),(1);
Query OK, 1 row affected, 1 warning (0.01 sec)
Records: 2  Duplicates: 1  Warnings: 1

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1062 | Duplicate entry '1' for key 'PRIMARY' |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)
```

These statements support the IGNORE keyword:

- CREATE TABLE ... SELECT: IGNORE does not apply to the CREATE TABLE or SELECT parts of the statement but to inserts into the table of rows produced by the SELECT. Rows that duplicate an existing row on a unique key value are discarded.
- DELETE: IGNORE causes MySQL to ignore errors during the process of deleting rows.
- INSERT: With IGNORE, rows that duplicate an existing row on a unique key value are discarded. Rows set to values that would cause data conversion errors are set to the closest valid values instead.

  For partitioned tables where no partition matching a given value is found, IGNORE causes the insert operation to fail silently for rows containing the unmatched value.

- `LOAD DATA`, `LOAD XML`: With `IGNORE`, rows that duplicate an existing row on a unique key value are discarded.
- `UPDATE`: With `IGNORE`, rows for which duplicate-key conflicts occur on a unique key value are not updated. Rows updated to values that would cause data conversion errors are updated to the closest valid values instead.

The `IGNORE` keyword applies to the following errors:

```
ER_BAD_NULL_ERROR
ER_DUP_ENTRY
ER_DUP_ENTRY_WITH_KEY_NAME
ER_DUP_KEY
ER_NO_PARTITION_FOR_GIVEN_VALUE
ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT
ER_NO_REFERENCED_ROW_2
ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET
ER_ROW_IS_REFERENCED_2
ER_SUBQUERY_NO_1_ROW
ER_VIEW_CHECK_FAILED
```

The Effect of Strict SQL Mode on Statement Execution

The MySQL server can operate in different SQL modes, and can apply these modes differently for different clients, depending on the value of the `sql_mode` system variable. In “strict” SQL mode, the server upgrades certain warnings to errors.

For example, in non-strict SQL mode, inserting the string `'abc'` into an integer column results in conversion of the value to 0 and a warning:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t (i) VALUES('abc');
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+---------+------+-------------------------------------------------------+
| Level   | Code | Message                                               |
+---------+------+-------------------------------------------------------+
| Warning | 1366 | Incorrect integer value: 'abc' for column 'i' at row 1 |
+---------+------+-------------------------------------------------------+
1 row in set (0.00 sec)
```

In strict SQL mode, the invalid value is rejected with an error:

```
mysql> SET sql_mode = 'STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t (i) VALUES('abc');
ERROR 1366 (HY000): Incorrect integer value: 'abc' for column 'i' at row 1
```

For more information about possible settings of the `sql_mode` system variable, see Section 5.1.11, “Server SQL Modes”.

Strict SQL mode applies to the following statements under conditions for which some value might be out of range or an invalid row is inserted into or deleted from a table:

- `ALTER TABLE`

- `CREATE TABLE`
- `CREATE TABLE ... SELECT`
- `DELETE` (both single table and multiple table)
- `INSERT`
- `LOAD DATA`
- `LOAD XML`
- `SELECT SLEEP()`
- `UPDATE` (both single table and multiple table)

Within stored programs, individual statements of the types just listed execute in strict SQL mode if the program was defined while strict mode was in effect.

Strict SQL mode applies to the following errors, represent a class of errors in which an input value is either invalid or missing. A value is invalid if it has the wrong data type for the column or might be out of range. A value is missing if a new row to be inserted does not contain a value for a `NOT NULL` column that has no explicit `DEFAULT` clause in its definition.

```
ER_BAD_NULL_ERROR
ER_CUT_VALUE_GROUP_CONCAT
ER_DATA_TOO_LONG
ER_DATETIME_FUNCTION_OVERFLOW
ER_DIVISION_BY_ZERO
ER_INVALID_ARGUMENT_FOR_LOGARITHM
ER_NO_DEFAULT_FOR_FIELD
ER_NO_DEFAULT_FOR_VIEW_FIELD
ER_TOO_LONG_KEY
ER_TRUNCATED_WRONG_VALUE
ER_TRUNCATED_WRONG_VALUE_FOR_FIELD
ER_WARN_DATA_OUT_OF_RANGE
ER_WARN_NULL_TO_NOTNULL
ER_WARN_TOO_FEW_RECORDS
ER_WRONG_ARGUMENTS
ER_WRONG_VALUE_FOR_TYPE
WARN_DATA_TRUNCATED
```

5.1.12 IPv6 Support

Support for IPv6 in MySQL includes these capabilities:

- MySQL Server can accept TCP/IP connections from clients connecting over IPv6. For example, this command connects over IPv6 to the MySQL server on the local host:

  ```
  shell> mysql -h ::1
  ```

  To use this capability, two things must be true:

  - Your system must be configured to support IPv6. See Section 5.1.12.1, “Verifying System Support for IPv6”.
  - The default MySQL server configuration permits IPv6 connections in addition to IPv4 connections. To change the default configuration, start the server with the `bind_address` system variable set to an appropriate value. See Section 5.1.8, “Server System Variables”.

- MySQL account names permit IPv6 addresses to enable DBAs to specify privileges for clients that connect to the server over IPv6. See Section 6.2.4, “Specifying Account Names”. IPv6 addresses can be specified in account names in statements such as `CREATE USER`, `GRANT`, and `REVOKE`. For example:

```
mysql> CREATE USER 'bill'@'::1' IDENTIFIED BY 'secret';
mysql> GRANT SELECT ON mydb.* TO 'bill'@'::1';
```

- IPv6 functions enable conversion between string and internal format IPv6 address formats, and checking whether values represent valid IPv6 addresses. For example, `INET6_ATON()` and `INET6_NTOA()` are similar to `INET_ATON()` and `INET_NTOA()`, but handle IPv6 addresses in addition to IPv4 addresses. See Section 12.24, “Miscellaneous Functions”.

- From MySQL 8.0.14, Group Replication group members can use IPv6 addresses for communications within the group. A group can contain a mix of members using IPv6 and members using IPv4. See Section 18.4.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”.

The following sections describe how to set up MySQL so that clients can connect to the server over IPv6.

5.1.12.1 Verifying System Support for IPv6

Before MySQL Server can accept IPv6 connections, the operating system on your server host must support IPv6. As a simple test to determine whether that is true, try this command:

```
shell> ping6 ::1
16 bytes from ::1, icmp_seq=0 hlim=64 time=0.171 ms
16 bytes from ::1, icmp_seq=1 hlim=64 time=0.077 ms
...
```

To produce a description of your system's network interfaces, invoke `ifconfig -a` and look for IPv6 addresses in the output.

If your host does not support IPv6, consult your system documentation for instructions on enabling it. It might be that you need only reconfigure an existing network interface to add an IPv6 address. Or a more extensive change might be needed, such as rebuilding the kernel with IPv6 options enabled.

These links may be helpful in setting up IPv6 on various platforms:

- Windows
- Gentoo Linux
- Ubuntu Linux
- Linux (Generic)
- macOS

5.1.12.2 Configuring the MySQL Server to Permit IPv6 Connections

The MySQL server listens on one or more network sockets for TCP/IP connections. Each socket is bound to one address, but it is possible for an address to map onto multiple network interfaces.

Set the `bind_address` system variable at server startup to specify the TCP/IP connections that a server instance accepts. As of MySQL 8.0.13, you can specify multiple values for this option, including any combination of IPv6 addresses, IPv4 addresses, and host names that resolve to IPv6 or IPv4 addresses. Alternatively, you can specify one of the wildcard address formats that permit listening on multiple network interfaces. A value of *, which is the default, or a value of `::`, permit both IPv4 and IPv6 connections on all server host IPv4 and IPv6 interfaces. For more information, see the `bind_address` description in Section 5.1.8, “Server System Variables”.

5.1.12.3 Connecting Using the IPv6 Local Host Address

The following procedure shows how to configure MySQL to permit IPv6 connections by clients that connect to the local server using the `::1` local host address. The instructions given here assume that your system supports IPv6.

1. Start the MySQL server with an appropriate `bind_address` setting to permit it to accept IPv6 connections. For example, put the following lines in the server option file and restart the server:

   ```
   [mysqld]
   bind_address = *
   ```

   Specifying * (or `::`) as the value for `bind_address` permits both IPv4 and IPv6 connections on all server host IPv4 and IPv6 interfaces. If you want to bind the server to a specific list of addresses, you can do this as of MySQL 8.0.13 by specifying a comma-separated list of values for `bind_address`. This example specifies the local host addresses for both IPv4 and IPv6:

   ```
   [mysqld]
   bind_address = 127.0.0.1,::1
   ```

   For more information, see the `bind_address` description in Section 5.1.8, “Server System Variables”.

2. As an administrator, connect to the server and create an account for a local user who will connect from the `::1` local IPv6 host address:

   ```
   mysql> CREATE USER 'ipv6user'@'::1' IDENTIFIED BY 'ipv6pass';
   ```

   For the permitted syntax of IPv6 addresses in account names, see Section 6.2.4, “Specifying Account Names”. In addition to the `CREATE USER` statement, you can issue `GRANT` statements that give specific privileges to the account, although that is not necessary for the remaining steps in this procedure.

3. Invoke the `mysql` client to connect to the server using the new account:

   ```
   shell> mysql -h ::1 -u ipv6user -pipv6pass
   ```

4. Try some simple statements that show connection information:

   ```
   mysql> STATUS
   ...
   Connection:   ::1 via TCP/IP
   ...

   mysql> SELECT CURRENT_USER(), @@bind_address;
   +----------------+----------------+
   | CURRENT_USER() | @@bind_address |
   +----------------+----------------+
   | ipv6user@::1   | ::             |
   +----------------+----------------+
   ```

5.1.12.4 Connecting Using IPv6 Nonlocal Host Addresses

The following procedure shows how to configure MySQL to permit IPv6 connections by remote clients. It is similar to the preceding procedure for local clients, but the server and client hosts are distinct and each has its own nonlocal IPv6 address. The example uses these addresses:

```
Server host: 2001:db8:0:f101::1
```

```
Client host: 2001:db8:0:f101::2
```

These addresses are chosen from the nonroutable address range recommended by IANA for documentation purposes and suffice for testing on your local network. To accept IPv6 connections from clients outside the local network, the server host must have a public address. If your network provider assigns you an IPv6 address, you can use that. Otherwise, another way to obtain an address is to use an IPv6 broker; see Section 5.1.12.5, “Obtaining an IPv6 Address from a Broker”.

1. Start the MySQL server with an appropriate `bind_address` setting to permit it to accept IPv6 connections. For example, put the following lines in the server option file and restart the server:

   ```
   [mysqld]
   bind_address = *
   ```

   Specifying * (or `::`) as the value for `bind_address` permits both IPv4 and IPv6 connections on all server host IPv4 and IPv6 interfaces. If you want to bind the server to a specific list of addresses, you can do this as of MySQL 8.0.13 by specifying a comma-separated list of values for `bind_address`. This example specifies an IPv4 address as well as the required server host IPv6 address:

   ```
   [mysqld]
   bind_address = 198.51.100.20,2001:db8:0:f101::1
   ```

   For more information, see the `bind_address` description in Section 5.1.8, “Server System Variables”.

2. On the server host (`2001:db8:0:f101::1`), create an account for a user who will connect from the client host (`2001:db8:0:f101::2`):

   ```
   mysql> CREATE USER 'remoteipv6user'@'2001:db8:0:f101::2' IDENTIFIED BY 'remoteipv6pass';
   ```

3. On the client host (`2001:db8:0:f101::2`), invoke the `mysql` client to connect to the server using the new account:

   ```
   shell> mysql -h 2001:db8:0:f101::1 -u remoteipv6user -premoteipv6pass
   ```

4. Try some simple statements that show connection information:

   ```
   mysql> STATUS
   ...
   Connection:   2001:db8:0:f101::1 via TCP/IP
   ...

   mysql> SELECT CURRENT_USER(), @@bind_address;
   +----------------------------------+----------------+
   | CURRENT_USER()                   | @@bind_address |
   +----------------------------------+----------------+
   | remoteipv6user@2001:db8:0:f101::2 | ::             |
   +----------------------------------+----------------+
   ```

5.1.12.5 Obtaining an IPv6 Address from a Broker

If you do not have a public IPv6 address that enables your system to communicate over IPv6 outside your local network, you can obtain one from an IPv6 broker. The Wikipedia IPv6 Tunnel Broker page lists several brokers and their features, such as whether they provide static addresses and the supported routing protocols.

After configuring your server host to use a broker-supplied IPv6 address, start the MySQL server with an appropriate `bind_address` setting to permit the server to accept IPv6 connections. You can specify * (or

`::`) as the `bind_address` value, or bind the server to the specific IPv6 address provided by the broker. For more information, see the `bind_address` description in Section 5.1.8, “Server System Variables”.

Note that if the broker allocates dynamic addresses, the address provided for your system might change the next time you connect to the broker. If so, any accounts you create that name the original address become invalid. To bind to a specific address but avoid this change-of-address problem, you might be able to arrange with the broker for a static IPv6 address.

The following example shows how to use Freenet6 as the broker and the `gogoc` IPv6 client package on Gentoo Linux.

1. Create an account at Freenet6 by visiting this URL and signing up:

   ```
   http://gogonet.gogo6.com
   ```

2. After creating the account, go to this URL, sign in, and create a user ID and password for the IPv6 broker:

   ```
   http://gogonet.gogo6.com/page/freenet6-registration
   ```

3. As `root`, install `gogoc`:

   ```
   shell> emerge gogoc
   ```

4. Edit `/etc/gogoc/gogoc.conf` to set the `userid` and `password` values. For example:

   ```
   userid=gogouser
   passwd=gogopass
   ```

5. Start `gogoc`:

   ```
   shell> /etc/init.d/gogoc start
   ```

   To start `gogoc` each time your system boots, execute this command:

   ```
   shell> rc-update add gogoc default
   ```

6. Use `ping6` to try to ping a host:

   ```
   shell> ping6 ipv6.google.com
   ```

7. To see your IPv6 address:

   ```
   shell> ifconfig tun
   ```

5.1.13 MySQL Server Time Zone Support

This section describes the time zone settings maintained by MySQL, how to load the system tables required for named time support, how to stay current with time zone changes, and how to enable leap-second support.

- Time Zone Variables
- Populating the Time Zone Tables

- Staying Current with Time Zone Changes
- Time Zone Leap Second Support

For information about time zone settings in replication setups, please see Section 17.5.1, “Replication Features and Issues”.

Time Zone Variables

MySQL Server maintains several time zone settings:

- The system time zone. When the server starts, it attempts to determine the time zone of the host machine automatically and uses it to set the `system_time_zone` system variable. The value does not change thereafter.

  To explicitly specify the system time zone for MySQL Server at startup, set the `TZ` environment variable before you start `mysqld`. If you start the server using `mysqld_safe`, its `--timezone` option provides another way to set the system time zone. The permissible values for `TZ` and `--timezone` are system dependent. Consult your operating system documentation to see what values are acceptable.

- The server current time zone. The global `time_zone` system variable indicates the time zone the server currently is operating in. The initial `time_zone` value is `'SYSTEM'`, which indicates that the server time zone is the same as the system time zone.

  Note

  If set to `SYSTEM`, every MySQL function call that requires a time zone calculation makes a system library call to determine the current system time zone. This call may be protected by a global mutex, resulting in contention.

  The initial global server time zone value can be specified explicitly at startup with the `--default-time-zone` option on the command line, or you can use the following line in an option file:

```
default-time-zone='timezone'
```

  If you have the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege, you can set the global server time zone value at runtime with this statement:

```
SET GLOBAL time_zone = timezone;
```

- Per-session time zones. Each client that connects has its own session time zone setting, given by the session `time_zone` variable. Initially, the session variable takes its value from the global `time_zone` variable, but the client can change its own time zone with this statement:

```
SET time_zone = timezone;
```

The session time zone setting affects display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` or `CURTIME()`, and values stored in and retrieved from `TIMESTAMP` columns. Values for `TIMESTAMP` columns are converted from the session time zone to UTC for storage, and from UTC to the session time zone for retrieval.

The session time zone setting does not affect values displayed by functions such as `UTC_TIMESTAMP()` or values in `DATE`, `TIME`, or `DATETIME` columns. Nor are values in those data types stored in UTC; the time zone applies for them only when converting from `TIMESTAMP` values. If you want locale-specific arithmetic for `DATE`, `TIME`, or `DATETIME` values, convert them to UTC, perform the arithmetic, and then convert back.

The current global and session time zone values can be retrieved like this:

```
SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
```

`timezone` values can be given in several formats, none of which are case sensitive:

- As the value `'SYSTEM'`, indicating that the server time zone is the same as the system time zone.
- As a string indicating an offset from UTC, such as `'+10:00'` or `'-6:00'`.
- As a named time zone, such as `'Europe/Helsinki'`, `'US/Eastern'`, or `'MET'`. Named time zones can be used only if the time zone information tables in the `mysql` database have been created and populated.

Populating the Time Zone Tables

Several tables in the `mysql` system schema exist to store time zone information (see Section 5.3, “The mysql System Schema”). The MySQL installation procedure creates the time zone tables, but does not load them. To do so manually, use the following instructions.

Note

Loading the time zone information is not necessarily a one-time operation because the information changes occasionally. When such changes occur, applications that use the old rules become out of date and you may find it necessary to reload the time zone tables to keep the information used by your MySQL server current. See Staying Current with Time Zone Changes.

If your system has its own *zoneinfo* database (the set of files describing time zones), use the `mysql_tzinfo_to_sql` program to load the time zone tables. Examples of such systems are Linux, macOS, FreeBSD, and Solaris. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system has no zoneinfo database, you can use a downloadable package, as described later in this section.

To load the time zone tables from the command line, pass the zoneinfo directory path name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql
```

The `mysql` command shown here assumes that you connect to the server using an account such as `root` that has privileges for modifying tables in the `mysql` system schema. Adjust the connection parameters as required.

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

`mysql_tzinfo_to_sql` also can be used to load a single time zone file or generate leap second information:

- To load a single time zone file *`tz_file`* that corresponds to a time zone name *`tz_name`*, invoke `mysql_tzinfo_to_sql` like this:

  ```
  mysql_tzinfo_to_sql tz_file tz_name | mysql -u root -p mysql
  ```

  With this approach, you must execute a separate command to load the time zone file for each named zone that the server needs to know about.

- If your time zone must account for leap seconds, initialize leap second information like this, where *`tz_file`* is the name of your time zone file:

```
mysql_tzinfo_to_sql --leap tz_file | mysql -u root -p mysql
```

After running `mysql_tzinfo_to_sql`, restart the server so that it does not continue to use any previously cached time zone data.

If your system has no zoneinfo database (for example, Windows), you can use a package containing SQL statements that is available for download at the MySQL Developer Zone:

```
https://dev.mysql.com/downloads/timezones.html
```

Warning

Do *not* use a downloadable time zone package if your system has a zoneinfo database. Use the `mysql_tzinfo_to_sql` utility instead. Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

To use an SQL-statement time zone package that you have downloaded, unpack it, then load the unpacked file contents into the time zone tables:

```
mysql -u root -p mysql < file_name
```

Then restart the server.

Warning

Do *not* use a downloadable time zone package that contains `MyISAM` tables. That is intended for older MySQL versions. MySQL now uses `InnoDB` for the time zone tables. Trying to replace them with `MyISAM` tables will cause problems.

Staying Current with Time Zone Changes

When time zone rules change, applications that use the old rules become out of date. To stay current, it is necessary to make sure that your system uses current time zone information is used. For MySQL, there are multiple factors to consider in staying current:

- The operating system time affects the value that the MySQL server uses for times if its time zone is set to `SYSTEM`. Make sure that your operating system is using the latest time zone information. For most operating systems, the latest update or service pack prepares your system for the time changes. Check the website for your operating system vendor for an update that addresses the time changes.
- If you replace the system's `/etc/localtime` time zone file with a version that uses rules differing from those in effect at `mysqld` startup, restart `mysqld` so that it uses the updated rules. Otherwise, `mysqld` might not notice when the system changes its time.
- If you use named time zones with MySQL, make sure that the time zone tables in the `mysql` database are up to date:
  - If your system has its own zoneinfo database, reload the MySQL time zone tables whenever the zoneinfo database is updated.
  - For systems that do not have their own zoneinfo database, check the MySQL Developer Zone for updates. When a new update is available, download it and use it to replace the content of your current time zone tables.

For instructions for both methods, see Populating the Time Zone Tables. `mysqld` caches time zone information that it looks up, so after updating the time zone tables, restart `mysqld` to make sure that it does not continue to serve outdated time zone data.

If you are uncertain whether named time zones are available, for use either as the server's time zone setting or by clients that set their own time zone, check whether your time zone tables are empty. The following query determines whether the table that contains time zone names has any rows:

```
mysql> SELECT COUNT(*) FROM mysql.time_zone_name;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
```

A count of zero indicates that the table is empty. In this case, no applications currently are using named time zones, and you need not update the tables (unless you want to enable named time zone support). A count greater than zero indicates that the table is not empty and that its contents are available to be used for named time zone support. In this case, be sure to reload your time zone tables so that applications that use named time zones will obtain correct query results.

To check whether your MySQL installation is updated properly for a change in Daylight Saving Time rules, use a test like the one following. The example uses values that are appropriate for the 2007 DST 1-hour change that occurs in the United States on March 11 at 2 a.m.

The test uses this query:

```
SELECT
  CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') AS time1,
  CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') AS time2;
```

The two time values indicate the times at which the DST change occurs, and the use of named time zones requires that the time zone tables be used. The desired result is that both queries return the same result (the input time, converted to the equivalent value in the 'US/Central' time zone).

Before updating the time zone tables, you see an incorrect result like this:

```
+---------------------+---------------------+
| time1               | time2               |
+---------------------+---------------------+
| 2007-03-11 01:00:00 | 2007-03-11 02:00:00 |
+---------------------+---------------------+
```

After updating the tables, you should see the correct result:

```
+---------------------+---------------------+
| time1               | time2               |
+---------------------+---------------------+
| 2007-03-11 01:00:00 | 2007-03-11 01:00:00 |
+---------------------+---------------------+
```

Time Zone Leap Second Support

Leap second values are returned with a time part that ends with `:59:59`. This means that a function such as `NOW()` can return the same value for two or three consecutive seconds during the leap second. It remains true that literal temporal values having a time part that ends with `:59:60` or `:59:61` are considered invalid.

If it is necessary to search for `TIMESTAMP` values one second before the leap second, anomalous results may be obtained if you use a comparison with `'YYYY-MM-DD hh:mm:ss'` values. The following example demonstrates this. It changes the session time zone to UTC so there is no difference between internal `TIMESTAMP` values (which are in UTC) and displayed values (which have time zone correction applied).

```
mysql> CREATE TABLE t1 (
         a INT,
         ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
         PRIMARY KEY (ts)
       );
Query OK, 0 rows affected (0.01 sec)

mysql> -- change to UTC
mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:59'
mysql> SET timestamp = 1230767999;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:60'
mysql> SET timestamp = 1230768000;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (2);
Query OK, 1 row affected (0.00 sec)

mysql> -- values differ internally but display the same
mysql> SELECT a, ts, UNIX_TIMESTAMP(ts) FROM t1;
+------+---------------------+--------------------+
| a    | ts                  | UNIX_TIMESTAMP(ts) |
+------+---------------------+--------------------+
|    1 | 2008-12-31 23:59:59 |         1230767999 |
|    2 | 2008-12-31 23:59:59 |         1230768000 |
+------+---------------------+--------------------+
2 rows in set (0.00 sec)

mysql> -- only the non-leap value matches
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:59';
+------+---------------------+
| a    | ts                  |
+------+---------------------+
|    1 | 2008-12-31 23:59:59 |
+------+---------------------+
1 row in set (0.00 sec)

mysql> -- the leap value with seconds=60 is invalid
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:60';
Empty set, 2 warnings (0.00 sec)
```

To work around this, you can use a comparison based on the UTC value actually stored in the column, which has the leap second correction applied:

```
mysql> -- selecting using UNIX_TIMESTAMP value return leap value
mysql> SELECT * FROM t1 WHERE UNIX_TIMESTAMP(ts) = 1230768000;
+------+---------------------+
| a    | ts                  |
+------+---------------------+
|    2 | 2008-12-31 23:59:59 |
+------+---------------------+
```

```
1 row in set (0.00 sec)
```

5.1.14 Server-Side Help Support

MySQL Server supports a `HELP` statement that returns information from the MySQL Reference Manual (see Section 13.8.3, “HELP Syntax”). This information is stored in several tables in the `mysql` schema (see Section 5.3, “The mysql System Schema”). Proper operation of the `HELP` statement requires that these help tables be initialized.

For a new installation of MySQL using a binary or source distribution on Unix, help-table content initialization occurs when you initialize the data directory (see Section 2.10.1, “Initializing the Data Directory”). For an RPM distribution on Linux or binary distribution on Windows, content initialization occurs as part of the MySQL installation process.

For a MySQL upgrade using a binary distribution, help-table content is upgraded automatically by the server as of MySQL 8.0.16. Prior to MySQL 8.0.16, the content is not upgraded automatically, but you can upgrade it manually. Locate the `fill_help_tables.sql` file in the `share` or `share/mysql` directory. Change location into that directory and process the file with the `mysql` client as follows:

```
mysql -u root -p mysql < fill_help_tables.sql
```

The command shown here assumes that you connect to the server using an account such as `root` that has privileges for modifying tables in the `mysql` schema. Adjust the connection parameters as required.

Prior to MySQL 8.0.16, if you are working with Git and a MySQL development source tree, the source tree contains only a “stub” version of `fill_help_tables.sql`. To obtain a non-stub copy, use one from a source or binary distribution.

Note

Each MySQL series has its own series-specific reference manual, so help-table content is series specific as well. This has implications for replication because help-table content should match the MySQL series. If you load MySQL 8.0 help content into a MySQL 8.0 master server, it does not make sense to replicate that content to a slave server from a different MySQL series and for which that content is not appropriate. For this reason, as you upgrade individual servers in a replication scenario, you should upgrade each server's help tables, using the instructions given earlier. (Manual help-content upgrade is necessary only for replication servers from versions lower than 8.0.16. As mentioned in the preceding instructions, content upgrades occur automatically as of MySQL 8.0.16.)

5.1.15 Server Tracking of Client Session State Changes

The MySQL server implements several session state trackers. A client can enable these trackers to receive notification of changes to its session state.

One use for the tracker mechanism is to provide a means for MySQL connectors and client applications to determine whether any session context is available to permit session migration from one server to another. (To change sessions in a load-balanced environment, it is necessary to detect whether there is session state to take into consideration when deciding whether a switch can be made.)

Another use for the tracker mechanism is to permit applications to know when transactions can be moved from one session to another. Transaction state tracking enables this, which is useful for applications that may wish to move transactions from a busy server to one that is less loaded. For example, a load-

balancing connector managing a client connection pool could move transactions between available sessions in the pool.

However, session switching cannot be done at arbitrary times. If a session is in the middle of a transaction for which reads or writes have been done, switching to a different session implies a transaction rollback on the original session. A session switch must be done only when a transaction does not yet have any reads or writes performed within it.

Examples of when transactions might reasonably be switched:

- Immediately after `START TRANSACTION`
- After `COMMIT AND CHAIN`

In addition to knowing transaction state, it is useful to know transaction characteristics, so as to use the same characteristics if the transaction is moved to a different session. The following characteristics are relevant for this purpose:

```
READ ONLY
READ WRITE
ISOLATION LEVEL
WITH CONSISTENT SNAPSHOT
```

To support the preceding session-switching activities, notification is available for these types of client session state information:

- Changes to these attributes of client session state:
  - The default schema (database).
  - Session-specific values for system variables.
  - User-defined variables.
  - Temporary tables.
  - Prepared statements.

  The `session_track_state_change` system variable controls this tracker.
- Changes to the default schema name. The `session_track_schema` system variable controls this tracker.
- Changes to the session values of system variables. The `session_track_system_variables` system variable controls this tracker.
- Available GTIDs. The `session_track_gtids` system variable controls this tracker.
- Information about transaction state and characteristics. The `session_track_transaction_info` system variable controls this tracker.

For descriptions of the tracker-related system variables, see Section 5.1.8, “Server System Variables”. Those system variables permit control over which change notifications occur, but do not provide a way to access notification information. Notification occurs in the MySQL client/server protocol, which includes tracker information in OK packets so that session state changes can be detected. To enable client applications to extract state-change information from OK packets returned by the server, the MySQL C API provides a pair of functions:

- `mysql_session_track_get_first()` fetches the first part of the state-change information received from the server. See Section 28.7.7.69, “mysql_session_track_get_first()”.
- `mysql_session_track_get_next()` fetches any remaining state-change information received from the server. Following a successful call to `mysql_session_track_get_first()`, call this function repeatedly as long as it returns success. See Section 28.7.7.70, “mysql_session_track_get_next()”.

The `mysqltest` program has `disable_session_track_info` and `enable_session_track_info` commands that control whether session tracker notifications occur. You can use these commands to see from the command line what notifications SQL statements produce. Suppose that a file `testscript` contains the following `mysqltest` script:

```
DROP TABLE IF EXISTS test.t1;
CREATE TABLE test.t1 (i INT, f FLOAT);
--enable_session_track_info
SET @@SESSION.session_track_schema=ON;
SET @@SESSION.session_track_system_variables='*';
SET @@SESSION.session_track_state_change=ON;
USE information_schema;
SET NAMES 'utf8mb4';
SET @@SESSION.session_track_transaction_info='CHARACTERISTICS';
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION READ WRITE;
START TRANSACTION;
SELECT 1;
INSERT INTO test.t1 () VALUES();
INSERT INTO test.t1 () VALUES(1, RAND());
COMMIT;
```

Run the script as follows to see the information provided by the enabled trackers. For a description of the `Tracker:` information displayed by `mysqltest` for the various trackers, see Section 28.7.7.69, “mysql_session_track_get_first()”.

```
shell> mysqltest < testscript
DROP TABLE IF EXISTS test.t1;
CREATE TABLE test.t1 (i INT, f FLOAT);
SET @@SESSION.session_track_schema=ON;
SET @@SESSION.session_track_system_variables='*';
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- session_track_system_variables
-- *

SET @@SESSION.session_track_state_change=ON;
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- session_track_state_change
-- ON

USE information_schema;
-- Tracker : SESSION_TRACK_SCHEMA
-- information_schema

-- Tracker : SESSION_TRACK_STATE_CHANGE
-- 1

SET NAMES 'utf8mb4';
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- character_set_client
-- utf8mb4
-- character_set_connection
-- utf8mb4
-- character_set_results
-- utf8mb4
```

```
-- Tracker : SESSION_TRACK_STATE_CHANGE
-- 1

SET @@SESSION.session_track_transaction_info='CHARACTERISTICS';
-- Tracker : SESSION_TRACK_SYSTEM_VARIABLES
-- session_track_transaction_info
-- CHARACTERISTICS

-- Tracker : SESSION_TRACK_STATE_CHANGE
-- 1

-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
--

-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- ________

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
-- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SET TRANSACTION READ WRITE;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
-- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; SET TRANSACTION READ WRITE;

START TRANSACTION;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
-- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; START TRANSACTION READ WRITE;

-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T_______

SELECT 1;
1
1
-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T_____S_

INSERT INTO test.t1 () VALUES();
-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T___W_S_

INSERT INTO test.t1 () VALUES(1, RAND());
-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- T___WsS_

COMMIT;
-- Tracker : SESSION_TRACK_TRANSACTION_CHARACTERISTICS
--

-- Tracker : SESSION_TRACK_TRANSACTION_STATE
-- ________

ok
```

Preceding the `START TRANSACTION` statement, two `SET TRANSACTION` statements execute that set the isolation level and access mode characteristics for the next transaction. The `SESSION_TRACK_TRANSACTION_CHARACTERISTICS` value indicates those next-transaction values that have been set.

Following the `COMMIT` statement that ends the transaction, the `SESSION_TRACK_TRANSACTION_CHARACTERISTICS` value is reported as empty. This indicates that the next-transaction characteristics that were set preceding the start of the transaction have been reset, and that the session defaults apply. To track changes to those session defaults, track the session values of the `transaction_isolation` and `transaction_read_only` system variables.

5.1.16 Server Response to Signals

On Unix, signals can be sent to processes. `mysqld` responds to signals sent to it as follows:

- `SIGTERM` causes the server to shut down.
- `SIGHUP` causes the server to reload the grant tables and to flush tables, logs, the thread cache, and the host cache. These actions are like various forms of the `FLUSH` statement. The server also writes a status report to the error log that has this format:

```
Status information:

Current dir: /var/mysql/data/
Running threads: 0  Stack size: 196608
Current locks:

Key caches:
default
Buffer_size:       8388600
Block_size:           1024
Division_limit:        100
Age_limit:             300
blocks used:             0
not flushed:             0
w_requests:              0
writes:                  0
r_requests:              0
reads:                   0

handler status:
read_key:            0
read_next:           0
read_rnd             0
read_first:          1
write:               0
delete               0
update:              0

Table status:
Opened tables:          5
Open tables:            0
Open files:             7
Open streams:           0

Alarm status:
Active alarms:   1
Max used alarms: 2
Next alarm time: 67
```

5.1.17 The Server Shutdown Process

The server shutdown process takes place as follows:

1. The shutdown process is initiated.

   This can occur initiated several ways. For example, a user with the `SHUTDOWN` privilege can execute a `mysqladmin shutdown` command. `mysqladmin` can be used on any platform supported by MySQL. Other operating system-specific shutdown initiation methods are possible as well: The server shuts down on Unix when it receives a `SIGTERM` signal. A server running as a service on Windows shuts down when the services manager tells it to.

2. The server creates a shutdown thread if necessary.

Depending on how shutdown was initiated, the server might create a thread to handle the shutdown process. If shutdown was requested by a client, a shutdown thread is created. If shutdown is the result of receiving a `SIGTERM` signal, the signal thread might handle shutdown itself, or it might create a separate thread to do so. If the server tries to create a shutdown thread and cannot (for example, if memory is exhausted), it issues a diagnostic message that appears in the error log:

```
Error: Can't create thread to kill server
```

3. The server stops accepting new connections.

   To prevent new activity from being initiated during shutdown, the server stops accepting new client connections by closing the handlers for the network interfaces to which it normally listens for connections: the TCP/IP port, the Unix socket file, the Windows named pipe, and shared memory on Windows.

4. The server terminates current activity.

   For each thread associated with a client connection, the server breaks the connection to the client and marks the thread as killed. Threads die when they notice that they are so marked. Threads for idle connections die quickly. Threads that currently are processing statements check their state periodically and take longer to die. For additional information about thread termination, see Section 13.7.8.4, “KILL Syntax”, in particular for the instructions about killed `REPAIR TABLE` or `OPTIMIZE TABLE` operations on `MyISAM` tables.

   For threads that have an open transaction, the transaction is rolled back. If a thread is updating a nontransactional table, an operation such as a multiple-row `UPDATE` or `INSERT` may leave the table partially updated because the operation can terminate before completion.

   If the server is a master replication server, it treats threads associated with currently connected slaves like other client threads. That is, each one is marked as killed and exits when it next checks its state.

   If the server is a slave replication server, it stops the I/O and SQL threads, if they are active, before marking client threads as killed. The SQL thread is permitted to finish its current statement (to avoid causing replication problems), and then stops. If the SQL thread is in the middle of a transaction at this point, the server waits until the current replication event group (if any) has finished executing, or until the user issues a `KILL QUERY` or `KILL CONNECTION` statement. See also Section 13.4.2.7, “STOP SLAVE Syntax”. Since nontransactional statements cannot be rolled back, in order to guarantee crash-safe replication, only transactional tables should be used.

   Note

   To guarantee crash safety on the slave, you must run the slave with `--relay-log-recovery` enabled.

   See also Section 17.2.4, “Replication Relay and Status Logs”).

5. The server shuts down or closes storage engines.

   At this stage, the server flushes the table cache and closes all open tables.

   Each storage engine performs any actions necessary for tables that it manages. `InnoDB` flushes its buffer pool to disk (unless `innodb_fast_shutdown` is 2), writes the current LSN to the tablespace, and terminates its own internal threads. `MyISAM` flushes any pending index writes for a table.

6. The server exits.

To provide information to management processes, the server returns one of the exit codes described in the following list. The phrase in parentheses indicates the action taken by systemd in response to the code, for platforms on which systemd is used to manage the server.

- 0 = successful termination (no restart done)
- 1 = unsuccessful termination (no restart done)
- 2 = unsuccessful termination (restart done)

5.2 The MySQL Data Directory

Information managed by the MySQL server is stored under a directory known as the data directory. The following list briefly describes the items typically found in the data directory, with cross references for additional information:

- Data directory subdirectories. Each subdirectory of the data directory is a database directory and corresponds to a database managed by the server. All MySQL installations have certain standard databases:
  - The `mysql` directory corresponds to the `mysql` system schema, which contains information required by the MySQL server as it runs. This database contains data dictionary tables and system tables. See Section 5.3, “The mysql System Schema”.
  - The `performance_schema` directory corresponds to the Performance Schema, which provides information used to inspect the internal execution of the server at runtime. See Chapter 26, *MySQL Performance Schema*.
  - The `sys` directory corresponds to the `sys` schema, which provides a set of objects to help interpret Performance Schema information more easily. See Chapter 27, *MySQL sys Schema*.
  - The `ndbinfo` directory corresponds to the `ndbinfo` database that stores information specific to NDB Cluster (present only for installations built to include NDB Cluster). See Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”.

  Other subdirectories correspond to databases created by users or applications.

  Note

  `INFORMATION_SCHEMA` is a standard database, but its implementation uses no corresponding database directory.
- Log files written by the server. See Section 5.4, “MySQL Server Logs”.
- `InnoDB` tablespace and log files. See Chapter 15, *The InnoDB Storage Engine*.
- Default/autogenerated SSL and RSA certificate and key files. See Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.
- The server process ID file (while the server is running).
- The `mysqld-auto.cnf` file that stores persisted global system variable settings. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

Some items in the preceding list can be relocated elsewhere by reconfiguring the server. In addition, the `--datadir` option enables the location of the data directory itself to be changed. For a given MySQL installation, check the server configuration to determine whether items have been moved.

5.3 The mysql System Schema

The `mysql` schema is the system schema. It contains tables that store information required by the MySQL server as it runs. A broad categorization is that the `mysql` schema contains data dictionary tables that store database object metadata, and system tables used for other operational purposes. The following discussion further subdivides the set of system tables into smaller categories.

- Data Dictionary Tables
- Grant System Tables
- Object Information System Tables
- Log System Tables
- Server-Side Help System Tables
- Time Zone System Tables
- Replication System Tables
- Optimizer System Tables
- Miscellaneous System Tables

The remainder of this section enumerates the tables in each category, with cross references for additional information. Data dictionary tables and system tables use the `InnoDB` storage engine unless otherwise indicated.

`mysql` system tables and data dictionary tables reside in a single `InnoDB` tablespace file named `mysql.ibd` in the MySQL data directory. Previously, these tables were created in individual tablespace files in the `mysql` database directory.

Data-at-rest encryption can be enabled for the `mysql` system schema tablespace. For more information, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”.

Data Dictionary Tables

These tables comprise the data dictionary, which contains metadata about database objects. For additional information, see Chapter 14, *MySQL Data Dictionary*.

Important

The data dictionary is new in MySQL 8.0. A data dictionary-enabled server entails some general operational differences compared to previous MySQL releases. For details, see Section 14.7, “Data Dictionary Usage Differences”. Also, for upgrades to MySQL 8.0 from MySQL 5.7, the upgrade procedure differs somewhat from previous MySQL releases and requires that you verify the upgrade readiness of your installation by checking specific prerequisites. For more information, see Section 2.11, “Upgrading MySQL”, particularly Section 2.11.5, “Preparing Your Installation for Upgrade”.

- `catalogs`: Catalog information.
- `character_sets`: Information about available character sets.
- `check_constraints`: Information about `CHECK` constraints defined on tables. See Section 13.1.20.7, “CHECK Constraints”.

- `collations`: Information about collations for each character set.
- `column_statistics`: Histogram statistics for column values. See Section 8.9.6, “Optimizer Statistics”.
- `column_type_elements`: Information about types used by columns.
- `columns`: Information about columns in tables.
- `dd_properties`: A table that identifies data dictionary properties, such as its version. The server uses this to determine whether the data dictionary must be upgraded to a newer version.
- `events`: Information about Event Scheduler events. See Section 24.4, “Using the Event Scheduler”. The server loads events listed in this table during its startup sequence, unless started with the `--skip-grant-tables` option.
- `foreign_keys`, `foreign_key_column_usage`: Information about foreign keys.
- `index_column_usage`: Information about columns used by indexes.
- `index_partitions`: Information about partitions used by indexes.
- `index_stats`: Used to store dynamic index statistics generated when `ANALYZE TABLE` is executed.
- `indexes`: Information about table indexes.
- `innodb_ddl_log`: Stores DDL logs for crash-safe DDL operations.
- `parameter_type_elements`: Information about stored procedure and function parameters, and about return values for stored functions.
- `parameters`: Information about stored procedures and functions. See Section 24.2, “Using Stored Routines”.
- `resource_groups`: Information about resource groups. See Section 8.12.5, “Resource Groups”
- `routines`: Information about stored procedures and functions. See Section 24.2, “Using Stored Routines”.
- `schemata`: Information about schemata. In MySQL, a schema is a database, so this table provides information about databases.
- `st_spatial_reference_systems`: Information about available spatial reference systems for spatial data.
- `table_partition_values`: Information about values used by table partitions.
- `table_partitions`: Information about partitions used by tables.
- `table_stats`: Information about dynamic table statistics generated when `ANALYZE TABLE` is executed.
- `tables`: Information about tables in databases.
- `tablespace_files`: Information about files used by tablespaces.
- `tablespaces`: Information about active tablespaces.
- `triggers`: Information about triggers.

- `view_routine_usage`: Information about dependencies between views and stored functions used by them.
- `view_table_usage`: Used to track dependencies between views and their underlying tables.

Data dictionary tables are invisible. They cannot be read with `SELECT`, do not appear in the output of `SHOW TABLES`, are not listed in the `INFORMATION_SCHEMA.TABLES` table, and so forth. However, in most cases there are corresponding `INFORMATION_SCHEMA` tables that can be queried. Conceptually, the `INFORMATION_SCHEMA` provides a view through which MySQL exposes data dictionary metadata. For example, you cannot select from the `mysql.schemata` table directly:

```
mysql> SELECT * FROM mysql.schemata;
ERROR 3554 (HY000): Access to data dictionary table 'mysql.schemata' is rejected.
```

Instead, select that information from the corresponding `INFORMATION_SCHEMA` table:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA\G
*************************** 1. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: mysql
DEFAULT_CHARACTER_SET_NAME: utf8mb4
    DEFAULT_COLLATION_NAME: utf8mb4_0900_ai_ci
                  SQL_PATH: NULL
        DEFAULT_ENCRYPTION: NO
*************************** 2. row ***************************
              CATALOG_NAME: def
               SCHEMA_NAME: information_schema
DEFAULT_CHARACTER_SET_NAME: utf8
    DEFAULT_COLLATION_NAME: utf8_general_ci
                  SQL_PATH: NULL
        DEFAULT_ENCRYPTION: NO
...
```

There is no `INFORMATION_SCHEMA` table that corresponds exactly to `mysql.indexes`, but `INFORMATION_SCHEMA.STATISTICS` contains much of the same information.

As of yet, there are no `INFORMATION_SCHEMA` tables that correspond exactly to `mysql.foreign_keys`, `mysql.foreign_key_column_usage`. The standard SQL way to obtain foreign key information is by using the `INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS` and `KEY_COLUMN_USAGE` tables; these tables are now implemented as views on the `foreign_keys`, `foreign_key_column_usage`, and other data dictionary tables.

Some system tables from before MySQL 8.0 have been replaced by data dictionary tables and are no longer present in the `mysql` system schema:

- The `events` data dictionary table supersedes the `event` table from before MySQL 8.0.
- The `parameters` and `routines` data dictionary tables together supersede the `proc` table from before MySQL 8.0.

Grant System Tables

These system tables contain grant information about user accounts and the privileges held by them. For additional information about the structure, contents, and purpose of the these tables, see Section 6.2.3, “Grant Tables”.

As of MySQL 8.0, the grant tables are `InnoDB` (transactional) tables. Previously, these were `MyISAM` (nontransactional) tables. The change of grant-table storage engine underlies an accompanying change in MySQL 8.0 to the behavior of account-management statements such as `CREATE USER` and `GRANT`.

Previously, an account-management statement that named multiple users could succeed for some users and fail for others. The statements are now transactional and either succeed for all named users or roll back and have no effect if any error occurs.

Note

If MySQL is upgraded from an older version but the grant tables have not been upgraded from `MyISAM` to `InnoDB`, the server considers them read only and account-management statements produce an error. For upgrade instructions, see Section 2.11, “Upgrading MySQL”.

- `user`: User accounts, global privileges, and other nonprivilege columns.
- `global_grants`: Assignments of dynamic global privileges to users; see Static Versus Dynamic Privileges.
- `db`: Database-level privileges.
- `tables_priv`: Table-level privileges.
- `columns_priv`: Column-level privileges.
- `procs_priv`: Stored procedure and function privileges.
- `proxies_priv`: Proxy-user privileges.
- `default_roles`: This table lists default roles to be activated after a user connects and authenticates, or executes `SET ROLE DEFAULT`.
- `role_edges`: This table lists edges for role subgraphs.

  A given `user` table row might refer to a user account or a role. The server can distinquish whether a row represents a user account, a role, or both by consulting the `role_edges` table for information about relations between authentication IDs.
- `password_history`: Information about password changes.

Object Information System Tables

These system tables contain information about stored programs, components, user-defined functions, and server-side plugins:

- `component`: The registry for server components. Any components listed in this table are installed by a loader service during the server startup sequence. See Section 5.5, “MySQL Server Components”.
- `func`: Information about user-defined functions (UDFs). See Section 29.4, “Adding New Functions to MySQL”. The server loads UDFs listed in this table during its startup sequence, unless started with the `--skip-grant-tables` option.
- `plugin`: Information about server-side plugins. See Section 5.6.1, “Installing and Uninstalling Plugins”, and Section 29.2, “The MySQL Plugin API”. The server loads plugins listed in this table during its startup sequence, unless started with the `--skip-grant-tables` option.

Log System Tables

The server uses these system tables for logging:

- `general_log`: The general query log table.

- `slow_log`: The slow query log table.

Log tables use the `CSV` storage engine.

For more information, see Section 5.4, “MySQL Server Logs”.

Server-Side Help System Tables

These system tables contain server-side help information:

- `help_category`: Information about help categories.
- `help_keyword`: Keywords associated with help topics.
- `help_relation`: Mappings between help keywords and topics.
- `help_topic`: Help topic contents.

For more information, see Section 5.1.14, “Server-Side Help Support”.

Time Zone System Tables

These system tables contain time zone information:

- `time_zone`: Time zone IDs and whether they use leap seconds.
- `time_zone_leap_second`: When leap seconds occur.
- `time_zone_name`: Mappings between time zone IDs and names.
- `time_zone_transition`, `time_zone_transition_type`: Time zone descriptions.

For more information, see Section 5.1.13, “MySQL Server Time Zone Support”.

Replication System Tables

The server uses these system tables to support replication:

- `gtid_executed`: Table for storing GTID values. See mysql.gtid_executed Table.
- `ndb_binlog_index`: Binary log information for NDB Cluster replication. This table is created only if the server is built with `NDBCLUSTER` support. See Section 22.6.4, “NDB Cluster Replication Schema and Tables”.
- `slave_master_info`, `slave_relay_log_info`, `slave_worker_info`: Used to store replication information on slave servers. See Section 17.2.4, “Replication Relay and Status Logs”.

All of the tables just listed use the `InnoDB` storage engine.

Optimizer System Tables

These system tables are for use by the optimizer:

- `innodb_index_stats`, `innodb_table_stats`: Used for `InnoDB` persistent optimizer statistics. See Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.
- `server_cost`, `engine_cost`: The optimizer cost model uses tables that contain cost estimate information about operations that occur during query execution. `server_cost` contains optimizer cost estimates for general server operations. `engine_cost` contains estimates for operations specific to particular storage engines. See Section 8.9.5, “The Optimizer Cost Model”.

Miscellaneous System Tables

Other system tables do not fit the preceding categories:

- `audit_log_filter`, `audit_log_user`: If MySQL Enterprise Audit is installed, these tables provide persistent storage of audit log filter definitions and user accounts. See Audit Log Tables.
- `firewall_users`, `firewall_whitelist`: If MySQL Enterprise Firewall is installed, these tables provide persistent storage for information used by the firewall. See Section 6.4.7, “MySQL Enterprise Firewall”.
- `servers`: Used by the `FEDERATED` storage engine. See Section 16.8.2.2, “Creating a FEDERATED Table Using CREATE SERVER”.
- `innodb_dynamic_metadata`: Used by the `InnoDB` storage engine to store fast-changing table metadata such as auto-increment counter values and index tree corruption flags. Replaces the data dictionary buffer table that resided in the `InnoDB` system tablespace.

5.4 MySQL Server Logs

MySQL Server has several logs that can help you find out what activity is taking place.

Log Type	Information Written to Log
Error log	Problems encountered starting, running, or stopping `mysqld`
General query log	Established client connections and statements received from clients
Binary log	Statements that change data (also used for replication)
Relay log	Data changes received from a replication master server
Slow query log	Queries that took more than `long_query_time` seconds to execute
DDL log (metadata log)	Metadata operations performed by DDL statements

By default, no logs are enabled, except the error log on Windows. (The DDL log is always created when required, and has no user-configurable options; see Section 5.4.6, “The DDL Log”.) The following log-specific sections provide information about the server options that enable logging.

By default, the server writes files for all enabled logs in the data directory. You can force the server to close and reopen the log files (or in some cases switch to a new log file) by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement; execute `mysqladmin` with a `flush-logs` or `refresh` argument; or execute `mysqldump` with a `--flush-logs` or `--master-data` option. See Section 13.7.8.3, “FLUSH Syntax”, Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”, and Section 4.5.4, “`mysqldump` — A Database Backup Program”. In addition, the binary log is flushed when its size reaches the value of the `max_binlog_size` system variable.

You can control the general query and slow query logs during runtime. You can enable or disable logging, or change the log file name. You can tell the server to write general query and slow query entries to log tables, log files, or both. For details, see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”, Section 5.4.3, “The General Query Log”, and Section 5.4.5, “The Slow Query Log”.

The relay log is used only on slave replication servers, to hold data changes from the master server that must also be made on the slave. For discussion of relay log contents and configuration, see Section 17.2.4.1, “The Slave Relay Log”.

For information about log maintenance operations such as expiration of old log files, see Section 5.4.7, “Server Log Maintenance”.

For information about keeping logs secure, see Section 6.1.2.3, “Passwords and Logging”.

5.4.1 Selecting General Query Log and Slow Query Log Output Destinations

MySQL Server provides flexible control over the destination of output written to the general query log and the slow query log, if those logs are enabled. Possible destinations for log entries are log files or the `general_log` and `slow_log` tables in the `mysql` system database. File output, table output, or both can be selected.

- Log Control at Server Startup
- Log Control at Runtime
- Log Table Benefits and Characteristics

Log Control at Server Startup

The `log_output` system variable specifies the destination for log output. Setting this variable does not in itself enable the logs; they must be enabled separately.

- If `log_output` is not specified at startup, the default logging destination is `FILE`.
- If `log_output` is specified at startup, its value is a list one or more comma-separated words chosen from `TABLE` (log to tables), `FILE` (log to files), or `NONE` (do not log to tables or files). `NONE`, if present, takes precedence over any other specifiers.

The `general_log` system variable controls logging to the general query log for the selected log destinations. If specified at server startup, `general_log` takes an optional argument of 1 or 0 to enable or disable the log. To specify a file name other than the default for file logging, set the `general_log_file` variable. Similarly, the `slow_query_log` variable controls logging to the slow query log for the selected destinations and setting `slow_query_log_file` specifies a file name for file logging. If either log is enabled, the server opens the corresponding log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected.

Examples:

- To write general query log entries to the log table and the log file, use `--log_output=TABLE,FILE` to select both log destinations and `--general_log` to enable the general query log.
- To write general and slow query log entries only to the log tables, use `--log_output=TABLE` to select tables as the log destination and `--general_log` and `--slow_query_log` to enable both logs.
- To write slow query log entries only to the log file, use `--log_output=FILE` to select files as the log destination and `--slow_query_log` to enable the slow query log. In this case, because the default log destination is `FILE`, you could omit the `log_output` setting.

Log Control at Runtime

The system variables associated with log tables and files enable runtime control over logging:

- The `log_output` variable indicates the current logging destination. It can be modified at runtime to change the destination.
- The `general_log` and `slow_query_log` variables indicate whether the general query log and slow query log are enabled (`ON`) or disabled (`OFF`). You can set these variables at runtime to control whether the logs are enabled.

- The `general_log_file` and `slow_query_log_file` variables indicate the names of the general query log and slow query log files. You can set these variables at server startup or at runtime to change the names of the log files.
- To disable or enable general query logging for the current session, set the session `sql_log_off` variable to `ON` or `OFF`. (This assumes that the general query log itself is enabled.)

Log Table Benefits and Characteristics

The use of tables for log output offers the following benefits:

- Log entries have a standard format. To display the current structure of the log tables, use these statements:

```
SHOW CREATE TABLE mysql.general_log;
SHOW CREATE TABLE mysql.slow_log;
```

- Log contents are accessible through SQL statements. This enables the use of queries that select only those log entries that satisfy specific criteria. For example, to select log contents associated with a particular client (which can be useful for identifying problematic queries from that client), it is easier to do this using a log table than a log file.
- Logs are accessible remotely through any client that can connect to the server and issue queries (if the client has the appropriate log table privileges). It is not necessary to log in to the server host and directly access the file system.

The log table implementation has the following characteristics:

- In general, the primary purpose of log tables is to provide an interface for users to observe the runtime execution of the server, not to interfere with its runtime execution.
- `CREATE TABLE`, `ALTER TABLE`, and `DROP TABLE` are valid operations on a log table. For `ALTER TABLE` and `DROP TABLE`, the log table cannot be in use and must be disabled, as described later.
- By default, the log tables use the `CSV` storage engine that writes data in comma-separated values format. For users who have access to the `.CSV` files that contain log table data, the files are easy to import into other programs such as spreadsheets that can process CSV input.

  The log tables can be altered to use the `MyISAM` storage engine. You cannot use `ALTER TABLE` to alter a log table that is in use. The log must be disabled first. No engines other than `CSV` or `MyISAM` are legal for the log tables.

  Log Tables and “Too many open files” Errors. If you select `TABLE` as a log destination and the log tables use the `CSV` storage engine, you may find that disabling and enabling the general query log or slow query log repeatedly at runtime results in a number of open file descriptors for the `.CSV` file, possibly resulting in a “Too many open files” error. To work around this issue, execute `FLUSH TABLES` or ensure that the value of `open_files_limit` is greater than the value of `table_open_cache_instances`.
- To disable logging so that you can alter (or drop) a log table, you can use the following strategy. The example uses the general query log; the procedure for the slow query log is similar but uses the `slow_log` table and `slow_query_log` system variable.

```
SET @old_log_state = @@GLOBAL.general_log;
SET GLOBAL general_log = 'OFF';
ALTER TABLE mysql.general_log ENGINE = MyISAM;
```

```
SET GLOBAL general_log = @old_log_state;
```

- `TRUNCATE TABLE` is a valid operation on a log table. It can be used to expire log entries.
- `RENAME TABLE` is a valid operation on a log table. You can atomically rename a log table (to perform log rotation, for example) using the following strategy:

```
USE mysql;
DROP TABLE IF EXISTS general_log2;
CREATE TABLE general_log2 LIKE general_log;
RENAME TABLE general_log TO general_log_backup, general_log2 TO general_log;
```

- `CHECK TABLE` is a valid operation on a log table.
- `LOCK TABLES` cannot be used on a log table.
- `INSERT`, `DELETE`, and `UPDATE` cannot be used on a log table. These operations are permitted only internally to the server itself.
- `FLUSH TABLES WITH READ LOCK` and the state of the `read_only` system variable have no effect on log tables. The server can always write to the log tables.
- Entries written to the log tables are not written to the binary log and thus are not replicated to slave servers.
- To flush the log tables or log files, use `FLUSH TABLES` or `FLUSH LOGS`, respectively.
- Partitioning of log tables is not permitted.
- A `mysqldump` dump includes statements to recreate those tables so that they are not missing after reloading the dump file. Log table contents are not dumped.

5.4.2 The Error Log

This section discusses how to configure the MySQL server for logging of diagnostic messages to the error log. For information about selecting the error message character set or language, see Section 10.6, “Error Message Character Set”, or Section 10.12, “Setting the Error Message Language”.

The error log contains a record of `mysqld` startup and shutdown times. It also contains diagnostic messages such as errors, warnings, and notes that occur during server startup and shutdown, and while the server is running. For example, if `mysqld` notices that a table needs to be automatically checked or repaired, it writes a message to the error log.

On some operating systems, the error log contains a stack trace if `mysqld` exits abnormally. The trace can be used to determine where `mysqld` exited. See Section 29.5, “Debugging and Porting MySQL”.

If used to start `mysqld`, `mysqld_safe` may write messages to the error log. For example, when `mysqld_safe` notices abnormal `mysqld` exits, it restarts `mysqld` and writes a `mysqld restarted` message to the error log.

The following sections discuss aspects of configuring error logging.

5.4.2.1 Error Log Component Configuration

In MySQL 8.0, error logging uses the MySQL component architecture described at Section 5.5, “MySQL Server Components”. The error log subsystem consists of components that perform log event filtering and

writing, as well as a system variable that configures which components to enable to achieve the desired logging result.

This section discusses how to select components for error logging. For instructions specific to log filters, see Section 5.4.2.3, “Types of Error Log Filtering”. For instructions specific to the JSON and system log writers, see Section 5.4.2.6, “Error Logging in JSON Format”, and Section 5.4.2.7, “Error Logging to the System Log”. For additional details about all available log components, see Section 5.5.3, “Error Log Components”.

Component-based error logging offers these features:

- Log events can be filtered by filter components to affect the information available for writing.
- Log events are output by sink (writer) components. Multiple sink components can be enabled, to write error log output to multiple destinations.
- Built-in filter and writer components combine to implement the default error log format.
- A loadable writer enables logging in JSON format.
- A loadable writer enables logging to the system log.
- System variables control which log components to enable and how each component operates.

The `log_error_services` system variable controls which log components to enable for error logging. The variable may contain a list with 0, 1, or many elements. In the latter case, elements may be delimited by semicolon or (as of MySQL 8.0.12) comma, optionally followed by space. A given setting cannot use both semicolon and comma separators. Component order is significant because the server executes components in the order listed.

By default, `log_error_services` has this value:

```
mysql> SELECT @@GLOBAL.log_error_services;
+---------------------------------------+
| @@GLOBAL.log_error_services           |
+---------------------------------------+
| log_filter_internal; log_sink_internal |
+---------------------------------------+
```

That value indicates that log events first pass through the built-in filter component, `log_filter_internal`, then through the built-in log writer component, `log_sink_internal`. A filter modifies log events seen by components named later in the `log_error_services` value. A sink is a destination for log events. Typically, a sink processes log events into log messages that have a particular format and writes these messages to its associated output, such as a file or the system log.

Note

Assigning `log_error_services` a value that contains no writer components causes no log output to be written from that point.

The final component in the `log_error_services` value should be a writer. If the final component is a filter, the component has no effect because the filtered events are not sent to any writer.

The combination of `log_filter_internal` and `log_sink_internal` implements the default error log filtering and output behavior. The action of these components is affected by other server options and system variables:

- The output destination is determined by the `--log-error` option (and, on Windows, `--pid-file` and `--console`). These determine whether to write error messages to the console or a file and, if to a file, the error log file name. See Section 5.4.2.2, “Default Error Log Destination Configuration”.
- The `log_error_verbosity` and `log_error_suppression_list` system variables affect which types of log events `log_filter_internal` permits or suppresses. See Section 5.4.2.4, “Priority-Based Error Log Filtering (log_filter_internal)”.

To change the set of log components used for error logging, load components as necessary and modify the `log_error_services` value. Adding or removing log components is subject to these constraints:

- To enable a log component, first load it using `INSTALL COMPONENT` (unless it is built in or already loaded), then list the component in the `log_error_services` value.

  For a component to be permitted in the `log_error_services` value, it must be known. A component is known if it is built in, or if it is loadable and has been loaded using `INSTALL COMPONENT`. Attempts to name an unknown component at server startup cause `log_error_services` to be set to its default value. Attempts to name an unknown component at runtime produce an error and the `log_error_services` value remains unchanged.
- To disable a log component, remove it from the `log_error_services` value. Then, if the component is loadable and you also want to unload it, use `UNINSTALL COMPONENT`.

  Attempts to use `UNINSTALL COMPONENT` to unload a loadable component that is still named in the `log_error_services` value produce an error.

For example, to use the system log writer (`log_sink_syseventlog`) instead of the default writer (`log_sink_internal`), first load the writer component, then modify the `log_error_services` value:

```
INSTALL COMPONENT 'file://component_log_sink_syseventlog';
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_syseventlog';
```

Note

The URN to use for loading a log component with `INSTALL COMPONENT` is the component name prefixed with `file://component_`. For example, for the `log_sink_syseventlog` component, the corresponding URN is `file://component_log_sink_syseventlog`.

It is possible to configure multiple log writers, which enables sending output to multiple destinations. To enable the system log writer in addition to (rather than instead of) the default writer, set the `log_error_services` value like this:

```
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal; log_sink_syseventlog';
```

To revert to using only the default writer and unload the system log writer, execute these statements:

```
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal;
UNINSTALL COMPONENT 'file://component_log_sink_syseventlog';
```

To configure a log component to be enabled at each server startup, use this procedure:

1. If the component is loadable, load it at runtime using `INSTALL COMPONENT`. Loading the component registers it in the `mysql.component` system table so that the server loads it automatically for subsequent startups.

2. Set the `log_error_services` value at startup to include the component name. Set the value either in the server `my.cnf` file, or use `SET PERSIST`, which sets the value for the running MySQL instance and also saves the value to be used for subsequent server restarts; see Section 13.7.6.1, “SET Syntax for Variable Assignment”. A value set in `my.cnf` takes effect at the next restart. A value set using `SET PERSIST` takes effect immediately, and for subsequent restarts.

Suppose that you want to configure, for every server startup, use of the JSON log writer (`log_sink_json`) in addition to the built-in log filter and writer (`log_filter_internal`, `log_sink_internal`). First load the JSON writer if it is not loaded:

```
INSTALL COMPONENT 'file://component_log_sink_json';
```

Then set `log_error_services` to take effect at server startup. You can set it in `my.cnf`:

```
[mysqld]
log_error_services='log_filter_internal; log_sink_internal; log_sink_json'
```

Or you can set it using `SET PERSIST`:

```
SET PERSIST log_error_services = 'log_filter_internal; log_sink_internal; log_sink_json';
```

The order of components named in `log_error_services` is significant, particularly with respect to the relative order of filters and writers. Consider this `log_error_services` value:

```
log_filter_internal; log_sink_1; log_sink_2
```

In this case, log events pass to the built-in filter, then to the first writer, then to the second writer. Both writers receive the filtered log events.

Compare that to this `log_error_services` value:

```
log_sink_1; log_filter_internal; log_sink_2
```

In this case, log events pass to the first writer, then to the built-in filter, then to the second writer. The first writer receives unfiltered events. The second writer receives filtered events. You might configure error logging this way if you want one log that contains messages for all log events, and another log that contains messages only for a subset of log events.

5.4.2.2 Default Error Log Destination Configuration

This section discusses which server options configure the default error log destination, which can be the console or a named file. It also indicates which log writer components base their own output destination on the default destination.

In this discussion, “console” means `stderr`, the standard error output. This is your terminal or console window unless the standard error output has been redirected to a different destination.

The server interprets options that determine the default error log destination somewhat differently for Windows and Unix systems. Be sure to configure the destination using the information appropriate to your platform:

- For Windows, see Default Error Log Destination on Windows.

- For Unix and Unix-like systems, see Default Error Log Destination on Unix and Unix-Like Systems.

After the server interprets the default error log destination options, it sets the `log_error` system variable to indicate the default destination, which affects where several log writer components write error messages. See How the Default Error Log Destination Affects Log Writers.

- Default Error Log Destination on Windows
- Default Error Log Destination on Unix and Unix-Like Systems
- How the Default Error Log Destination Affects Log Writers

Default Error Log Destination on Windows

On Windows, `mysqld` uses the `--log-error`, `--pid-file`, and `--console` options to determine whether the default error log destination is the console or a file, and, if a file, the file name:

- If `--console` is given, the default destination is the console. (`--console` takes precedence over `--log-error` if both are given, and the following items regarding `--log-error` do not apply.)
- If `--log-error` is not given, or is given without naming a file, the default destination is a file named *`host_name`*`.err` in the data directory, unless the `--pid-file` option is specified. In that case, the file name is the PID file base name with a suffix of `.err` in the data directory.
- If `--log-error` is given to name a file, the default destination is that file (with an `.err` suffix added if the name has no suffix), located under the data directory unless an absolute path name is given to specify a different location.

If the default error log destination is the console, the server sets the `log_error` system variable to `stderr`. Otherwise, the default destination is a file and the server sets `log_error` to the file name.

Default Error Log Destination on Unix and Unix-Like Systems

On Unix and Unix-like systems, `mysqld` uses the `--log-error` option to determine whether the default error log destination is the console or a file, and, if a file, the file name:

- If `--log-error` is not given, the default destination is the console.
- If `--log-error` is given without naming a file, the default destination is a file named *`host_name`*`.err` in the data directory.
- If `--log-error` is given to name a file, the default destination is that file (with an `.err` suffix added if the name has no suffix), located under the data directory unless an absolute path name is given to specify a different location.
- If `--log-error` is given in an option file in a `[mysqld]`, `[server]`, or `[mysqld_safe]` section, `mysqld_safe` finds and uses the option, and passes it to `mysqld`.

Note

It is common for Yum or APT package installations to configure an error log file location under `/var/log` with an option like `log-error=/var/log/mysqld.log` in a server configuration file. Removing the file name from the option causes the *`host_name`*`.err` file in the data directory to be used.

If the default error log destination is the console, the server sets the `log_error` system variable to `stderr`. Otherwise, the default destination is a file and the server sets `log_error` to the file name.

How the Default Error Log Destination Affects Log Writers

After the server interprets the error log destination configuration options, it sets the `log_error` system variable to indicate the default error log destination. Log writer components may base their own output destination on the `log_error` value, or determine their destination independently of `log_error`

If `log_error` is `stderr`, the default error log destination is the console, and log writers that base their output destination on the default destination also write to the console:

- `log_sink_internal`, `log_sink_json`, `log_sink_test`: These writers write to the console. This is true even for writers such as `log_sink_json` that can be enabled multiple times; all instances write to the console.
- `log_sink_syseventlog`: This writer writes to the system log, regardless of the `log_error` value.

If `log_error` is not `stderr`, the default error log destination is a file and `log_error` indicates the file name. Log writers that base their output destination on the default destination base output file naming on that file name. (A writer might use exactly that name, or it might use some variant thereof.) Suppose that the `log_error` value *`file_name`*. Then log writers use the name like this:

- `log_sink_internal`, `log_sink_test`: These writers write to *`file_name`*.
- `log_sink_json`: Successive instances of this writer named in the `log_error_services` value write to files named *`file_name`* plus a numbered `.NN.json` suffix: *`file_name`*`.00.json`, *`file_name`*`.01.json`, and so forth.
- `log_sink_syseventlog`: This writer writes to the system log, regardless of the `log_error` value.

5.4.2.3 Types of Error Log Filtering

Error log configuration normally includes one log filter component and one or more log writer component. For error log filtering, MySQL offers a choice of components:

- `log_filter_internal`: This filter component provides error log filtering based on log event priority and error code, in combination with the `log_error_verbosity` and `log_error_suppression_list` system variables. `log_filter_internal` is built in and enabled by default. See Section 5.4.2.4, “Priority-Based Error Log Filtering (log_filter_internal)”.
- `log_filter_dragnet`: This filter component provides error log filtering based on user-supplied rules, in combination with the `dragnet.log_error_filter_rules` system variable. See Section 5.4.2.5, “Rule-Based Error Log Filtering (log_filter_dragnet)”.

5.4.2.4 Priority-Based Error Log Filtering (log_filter_internal)

The `log_filter_internal` log filter component implements a simple form of log filtering based on error event priority and error code. To affect how `log_filter_internal` permits or suppresses error, warning, and information events intended for the error log, set the `log_error_verbosity` and `log_error_suppression_list` system variables.

`log_filter_internal` is built in and enabled by default. If this filter is disabled, `log_error_verbosity` and `log_error_suppression_list` have no effect, so filtering must be modeled using another filter service instead where desired (for example, with individual filter rules when using `log_filter_dragnet`). For information about filter configuration, see Section 5.4.2.1, “Error Log Component Configuration”.

- Verbosity Filtering

- Suppression-List Filtering
- Verbosity and Suppression-List Interaction

Verbosity Filtering

Events intended for the error log have a priority of `ERROR`, `WARNING`, or `INFORMATION`. The `log_error_verbosity` system variable controls verbosity based on which priorities to permit for messages written to the log, as shown in the following table.

Permitted Message Priorities	log_error_verbosity Value
`ERROR`	1
`ERROR`, `WARNING`	2
`ERROR`, `WARNING`, `INFORMATION`	3

If `log_error_verbosity` is 2 or greater, the server logs messages about statements that are unsafe for statement-based logging. If the value is 3, the server logs aborted connections and access-denied errors for new connection attempts. See Section B.4.2.10, “Communication Errors and Aborted Connections”.

If you use replication, a `log_error_verbosity` value of 2 or greater is recommended, to obtain more information about what is happening, such as messages about network failures and reconnections.

If `log_error_verbosity` is 2 or greater on a slave server, the slave prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, and so forth.

There is also a priority of `SYSTEM`. System messages about non-error situations are printed to the error log regardless of the `log_error_verbosity` value. These messages include startup and shutdown messages, and some significant changes to settings.

In the MySQL error log, system messages are labeled as “System”. Other log writers might or might not follow the same convention, and in the resulting logs, system messages might be assigned the label used for the information priority level, such as “Note” or “Information”. If you apply any additional filtering or redirection for logging based on the labeling of messages, system messages do not override your filter, but are handled by it in the same way as other messages.

Suppression-List Filtering

The `log_error_suppression_list` system variable applies to events intended for the error log and specifies which events to suppress when they occur with a priority of `WARNING` or `INFORMATION`. For example, if a particular type of warning is considered undesirable “noise” in the error log because it occurs frequently but is not of interest, it can be suppressed.

The `log_error_suppression_list` value may be the empty string for no suppression, or a list of one or more comma-separated values indicating the error codes to suppress. Error codes may be specified in symbolic or numeric form. A numeric code may be specified with or without the `MY-` prefix. Leading zeros in the numeric part are not significant. Examples of permitted code formats:

```
ER_SERVER_SHUTDOWN_COMPLETE
MY-000031
000031
MY-31
31
```

Symbolic values are preferable to numeric values for readability and portability.

Although codes to be suppressed can be expressed in symbolic or numeric form, the numeric value of each code must be in a permitted range:

- 1 to 999: Global error codes that are used by the server as well as by clients.
- 10000 and higher: Server error codes intended to be written to the error log (not sent to clients).

In addition, each error code specified must actually be used by MySQL. Attempts to specify a code not within a permitted range or within a permitted range but not used by MySQL produce an error and the `log_error_suppression_list` value remains unchanged.

For information about error code ranges and the error symbols and numbers defined within each range, see Appendix B, *Errors, Error Codes, and Common Problems*.

The server can generate messages for a given error code at differing priorities, so suppression of a message associated with an error code listed in `log_error_suppression_list` depends on its priority. Suppose that the variable has a value of `'ER_PARSER_TRACE,MY-010001,10002'`. Then `log_error_suppression_list` has these effects:

- Messages for those codes are suppressed if generated with a priority of `WARNING` or `INFORMATION`.
- Messages generated with a priority of `ERROR` or `SYSTEM` are not suppressed.

Verbosity and Suppression-List Interaction

The effect of `log_error_verbosity` combines with that of `log_error_suppression_list`. Consider a server started with these settings:

```
[mysqld]
log_error_verbosity=2     # error and warning messages only
log_error_suppression_list='ER_PARSER_TRACE,MY-010001,10002'
```

In this case, `log_error_verbosity` permits messages with `ERROR` or `WARNING` priority and discards messages with `INFORMATION` priority. Of the nondiscarded messages, `log_error_suppression_list` discards messages with `WARNING` priority and any of the named error codes.

Note

The `log_error_verbosity` value of 2 shown in the example is also its default value, so the effect of this variable on `INFORMATION` messages is as just described by default, without an explicit setting. You must set `log_error_verbosity` to 3 if you want `log_error_suppression_list` to affect messages with `INFORMATION` priority.

Consider a server started with this setting:

```
[mysqld]
log_error_verbosity=1     # error messages only
```

In this case, `log_error_verbosity` permits messages with `ERROR` priority and discards messages with `WARNING` or `INFORMATION` priority. Setting `log_error_suppression_list` would have no effect because all error codes it might suppress are already discarded due to the `log_error_verbosity` setting.

5.4.2.5 Rule-Based Error Log Filtering (log_filter_dragnet)

The `log_filter_dragnet` log filter component enables log filtering based on user-defined rules. To define the applicable rules, set the `dragnet.log_error_filter_rules` system variable.

To enable the `log_filter_dragnet` filter, first load the filter component, then modify the `log_error_services` value. The following example enables `log_filter_dragnet` in combination with the built-in log writer:

```
INSTALL COMPONENT 'file://component_log_filter_dragnet';
SET GLOBAL log_error_services = 'log_filter_dragnet; log_sink_internal';
```

To set `log_error_services` to take effect at server startup, use the instructions at Section 5.4.2.1, “Error Log Component Configuration”. Those instructions apply to other error-logging system variables as well.

With `log_filter_dragnet` enabled, define its filter rules by setting the `dragnet.log_error_filter_rules` system variable. A rule set consists of zero or more rules, where each rule is an `IF` statement terminated by a period (`.`) character. If the variable value is empty (zero rules), no filtering occurs.

Example 1. This rule set drops information events, and, for other events, removes the `source_line` field:

```
SET GLOBAL dragnet.log_error_filter_rules =
  'IF prio>=INFORMATION THEN drop. IF EXISTS source_line THEN unset source_line.';
```

The effect is similar to the filtering performed by the `log_sink_internal` filter with a setting of `log_error_verbosity=2`.

Example 2: This rule limits information events to no more than one per 60 seconds:

```
SET GLOBAL dragnet.log_error_filter_rules =
  'IF prio>=INFORMATION THEN throttle 1/60.';
```

Once you have the filtering configuration set up as you desire, consider assigning `dragnet.log_error_filter_rules` using `SET PERSIST` rather than `SET GLOBAL` to make the setting persist across server restarts. Alternatively, add the setting to the server option file.

To stop using the filtering language, first remove it from the set of error logging components. Usually this means using a different filter component rather than no filter component. For example:

```
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_internal';
```

Again, consider using using `SET PERSIST` rather than `SET GLOBAL` to make the setting persist across server restarts.

Then uninstall the filter `log_filter_dragnet` component:

```
UNINSTALL COMPONENT 'file://component_log_filter_dragnet';
```

The following sections describe aspects of `log_filter_dragnet` operation in more detail:

- log_filter_dragnet Rule Language
- log_filter_dragnet Rule Actions

- log_filter_dragnet Rule Fields

log_filter_dragnet Rule Language

The following grammar defines the language for `log_filter_dragnet` filter rules. Each rule is an `IF` statement terminated by a period (`.`) character. The language is not case sensitive.

```
rule:
    IF condition THEN action
    [ELSEIF condition THEN action] ...
    [ELSE action]
    .

condition: {
    field comparator value
  | [NOT] EXISTS field
  | condition {AND | OR}  condition
}

action: {
    drop
  | throttle {count | count / window_size}
  | set field [:= | =] value
  | unset [field]
}

field: {
    core_field
  | optional_field
  | user_defined_field
}

core_field: {
    time
  | msg
  | prio
  | label
  | err_code
  | err_symbol
  | SQL_state
  | subsystem
}

optional_field: {
    OS_errno
  | OS_errmsg
  | user
  | host
  | thread
  | query_id
  | source_file
  | source_line
  | function
}

user_defined_field:
    sequence of characters in [a-zA-Z0-9_] class

comparator: {== | != | <> | >= | => | <= | =< | < | >}

value: {
    string_literal
  | integer_literal
  | float_literal
  | error_symbol
```

```
  | priority
}

count: integer_literal
window_size: integer_literal

string_literal:
    sequence of characters quoted as '...' or "..."

integer_literal:
    sequence of characters in [0-9] class

float_literal:
    integer_literal[.integer_literal]

error_symbol:
    valid MySQL error symbol such as ER_ACCESS_DENIED_ERROR or ER_STARTUP

priority: {
    ERROR
  | WARNING
  | INFORMATION
}
```

Simple conditions compare a field to a value or test field existence. To construct more complex conditions, use the `AND` and `OR` operators. Both operators have the same precedence and evaluate left to right.

To escape a character within a string, precede it by a backslash (`\`). A backslash is required to include backslash itself or the string-quoting character, optional for other characters.

For convenience, `log_filter_dragnet` supports symbolic names for comparisons to certain fields. Where applicable, symbolic values are preferable to numeric values for readability and portability.

- Event priority values 1, 2, and 3 can be specified as `ERROR`, `WARNING`, and `INFORMATION`. Priority symbols are recognized only in comparisons with the `prio` field. These comparisons are equivalent:

  ```
  IF prio == INFORMATION THEN ...
  IF prio == 3 THEN ...
  ```

- Error codes can be specified in numeric form or as the corresponding error symbol. For example, `ER_STARTUP` is the symbolic name for error `1408`, so these comparisons are equivalent:

  ```
  IF err_code == ER_STARTUP THEN ...
  IF err_code == 1408 THEN ...
  ```

  Error symbols are recognized only in comparisons with the `err_code` field and user-defined fields.

  To find the error symbol corresponding to a given error code number, use one of these methods:

  - Check the list of server errors at Section B.3.1, “Server Error Message Reference”.
  - Use the `perror` command, which when given an error number argument, displays information about the error, including its symbol.

  Suppose that a rule set with error numbers looks like this:

  ```
  IF err_code == 10927 OR err_code == 10914 THEN drop.
  IF err_code == 1131 THEN drop.
  ```

  Using `perror`, determine the error symbols:

```
shell> perror 10927 10914 1131
MySQL error code MY-010927 (ER_ACCESS_DENIED_FOR_USER_ACCOUNT_LOCKED):
Access denied for user '%-.48s'@'%-.64s'. Account is locked.
MySQL error code MY-010914 (ER_ABORTING_USER_CONNECTION):
Aborted connection %u to db: '%-.192s' user: '%-.48s' host:
'%-.64s' (%-.64s).
MySQL error code MY-001131 (ER_PASSWORD_ANONYMOUS_USER):
You are using MySQL as an anonymous user and anonymous users
are not allowed to change passwords
```

Substituting error symbols for numbers, the rule set becomes:

```
IF err_code == ER_ACCESS_DENIED_FOR_USER_ACCOUNT_LOCKED
  OR err_code == ER_ABORTING_USER_CONNECTION THEN drop.
IF err_code == ER_PASSWORD_ANONYMOUS_USER THEN drop.
```

Symbolic names can be specified as quoted strings for comparison with string fields, but in such cases the names are strings that have no special meaning and `log_filter_dragnet` does not resolve them to the corresponding numeric value. Also, typos may go undetected, whereas an error is thrown immediately on `SET` for attempts to use an unquoted symbol unknown to the server.

log_filter_dragnet Rule Actions

`log_filter_dragnet` supports these actions in filter rules:

- `drop`: Drop the current log event (do not log it).
- `throttle`: Apply rate limiting to reduce log verbosity for events matching particular conditions. The argument indicates a rate, in the form *`count`* or *`count`*/*`window_size`*. The *`count`* value indicates the permitted number of events to log per time window. The *`window_size`* value is the time window in seconds; if omitted, the default window is 60 seconds. Both values must be integer literals.

  This rule throttles plugin-shutdown messages to 5 per 60 seconds:

  ```
  IF err_code == ER_PLUGIN_SHUTTING_DOWN_PLUGIN THEN throttle 5.
  ```

  This rule throttles errors and warnings to 1000 per hour and information messages to 100 per hour:

  ```
  IF prio <= INFORMATION THEN throttle 1000/3600 ELSE throttle 100/3600.
  ```

- `set`: Assign a value to a field (and cause the field to exist if it did not already). In subsequent rules, `EXISTS` tests against the field name are true, and the new value can be tested by comparison conditions.
- `unset`: Discard a field. In subsequent rules, `EXISTS` tests against the field name are false, and comparisons of the field against any value are false.

  In the special case that the condition refers to exactly one field name, the field name following `unset` is optional and `unset` discards the named field. These rules are equivalent:

  ```
  IF myfield == 2 THEN unset myfield.
  IF myfield == 2 THEN unset.
  ```

log_filter_dragnet Rule Fields

`log_filter_dragnet` supports core, optional, and user-defined fields in rules:

- A core field is set up automatically for error events. However, its presence in the event is not guaranteed because a core field, like any type of field, may be unset by filter rules. If so, the field will be found missing by later rules within the rule set and by components that execute after the filter (such as log writers).
- An optional field is normally absent but may be present for certain event types. When present, an optional field provides additional event information as appropriate and available.
- A user-defined field is any field with a name that is not already defined as a core or optional field. A user-defined field does not exist until created with the `set` action.

As implied by the preceding description, any given field may be absent, either because it was not present in the first place, or was discarded by a filtering rule. For log writers, the effect of field absence is writer specific. For example, a writer might omit the field from the log message, indicate that the field is missing, or substitute a default. When in doubt, use a filter rule to unset the field, then check what the log writer does with it.

These fields are core fields:

- `time`

  The event timestamp.

- `msg`

  The event message string.

- `prio`

  The event priority, to indicate error, warning, or note/information event. This field corresponds to severity in `syslog`.

  In comparisons, each priority can be specified as a symbolic priority name or an integer literal. Priority symbols are recognized only in comparisons with the `prio` field. These comparisons are equivalent:

```
IF prio == INFORMATION THEN ...
IF prio == 3 THEN ...
```

  The following table shows the permitted priority levels.

Event Type	Priority Symbol	Numeric Priority
Error events	`ERROR`	1
Warning events	`WARNING`	2
Note/information events	`INFORMATION`	3

  There is also a priority of `SYSTEM`, but system messages cannot be filtered and are always written to the error log.

  In general, message priorities are determined as follows:

  Is the situation or event actionable?

  - Yes: Is the situation or event ignorable?
    - Yes: Priority is `WARNING`.

    - No: Priority is `ERROR`.
  - No: Is the situation or event mandatory?
    - Yes: Priority is `SYSTEM`.
    - No: Priority is `INFORMATION`.

  Priority values follow the principle that higher priorities have lower values, and vice versa. Priority values begin at 1 for the most severe events (errors) and increase for events with decreasing priority. For example, to discard events with lower priority than warnings, test for priority values higher than `WARNING`:

```
IF prio > WARNING THEN drop.
```

  The following examples show the `log_filter_dragnet` rules to achieve an effect similar to each `log_error_verbosity` value permitted by the `log_filter_internal` filter:

  - Errors only (`log_error_verbosity=1`):

```
IF prio > ERROR THEN drop.
```

  - Errors and warnings (`log_error_verbosity=2`):

```
IF prio > WARNING THEN drop.
```

  - Errors, warnings, and notes (`log_error_verbosity=3`):

```
IF prio > INFORMATION THEN drop.
```

    This rule can actually be omitted because there are no `prio` values greater than `INFORMATION`, so effectively it drops nothing.

- `err_code`

  The numeric event error code. In comparisons, the value to test can be specified as a symbolic error name or an integer literal. Error symbols are recognized only in comparisons with the `err_code` field and user-defined fields. These comparisons are equivalent:

```
IF err_code == ER_ACCESS_DENIED_ERROR THEN ...
IF err_code == 1045 THEN ...
```

- `err_symbol`

  The event error symbol, as a string (for example, `'ER_DUP_KEY'`). `err_symbol` values are intended more for identifying particular lines in log output than for use in filter rule comparisons because `log_filter_dragnet` does not resolve comparison values specified as strings to the equivalent numeric error code.

- `SQL_state`

  The event SQLSTATE value, as a string (for example, `'23000'`).

- `subsystem`

The subsystem in which the event occurred. Possible values are `InnoDB` (the `InnoDB` storage engine), `Repl` (the replication subsystem), `Server` (otherwise).

Optional fields fall into the following categories:

Additional information about the error, such as the error signaled by the operating system or the error lable:

- `OS_errno`

  The operating system error number.

- `OS_errmsg`

  The operating system error message.

- `label`

  The label corresponding to the `prio` value, as a string. Filter rules can change the label for log writers that support custom labels. `label` values are intended more for identifying particular lines in log output than for use in filter rule comparisons because `log_filter_dragnet` does not resolve comparison values specified as strings to the equivalent numeric priority.

Identification of the client for which the event occurred:

- `user`

  The client user.

- `host`

  The client host.

- `thread`

  The thread ID.

- `query_id`

  The query ID.

Debugging information:

- `source_file`

  The source file in which the event occurred. The file name should omit any leading path. For example, to test for the `sql/gis/distance.cc` file, write the comparison like this:

  ```
  IF source_file == "distance.cc" THEN ...
  ```

- `source_line`

  The line within the source file at which the event occurred.

- `function`

  The function in which the event occurred.

- `component`

The component or plugin in which the event occurred.

5.4.2.6 Error Logging in JSON Format

This section describes how to configure error logging using the built-in filter, `log_filter_internal`, and the JSON writer, `log_sink_json`, to take effect immediately and for subsequent server startups. For general information about configuring error logging, see Section 5.4.2.1, “Error Log Component Configuration”.

To enable the JSON writer, first load the writer component, then modify the `log_error_services` value:

```
INSTALL COMPONENT 'file://component_log_sink_json';
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_json';
```

To set `log_error_services` to take effect at server startup, use the instructions at Section 5.4.2.1, “Error Log Component Configuration”. Those instructions apply to other error-logging system variables as well.

It is permitted to name `log_sink_json` multiple times in the `log_error_services` value. For example, to write unfiltered events with one instance and filtered events with another instance, you could set `log_error_services` like this:

```
SET GLOBAL log_error_services = 'log_sink_json; log_filter_internal; log_sink_json';
```

The JSON log writer determines its output destination based on the default error log destination, which is given by the `log_error` system variable. If `log_error` names a file, the JSON writer bases output file naming on that file name, plus a numbered *`.NN`*`.json` suffix, with *`NN`* starting at 00. For example, if `log_error` is *`file_name`*, successive instances of `log_sink_json` named in the `log_error_services` value write to *`file_name`*`.00.json`, *`file_name`*`.01.json`, and so forth.

If `log_error` is `stderr`, the JSON writer writes to the console. If `log_json_writer` is named multiple times in the `log_error_services` value, they all write to the console, which is likely not useful.

5.4.2.7 Error Logging to the System Log

It is possible to have `mysqld` write the error log to the system log (the Event Log on Windows, and `syslog` on Unix and Unix-like systems).

This section describes how to configure error logging using the built-in filter, `log_filter_internal`, and the system log writer, `log_sink_syseventlog`, to take effect immediately and for subsequent server startups. For general information about configuring error logging, see Section 5.4.2.1, “Error Log Component Configuration”.

To enable the system log writer, first load the writer component, then modify the `log_error_services` value:

```
INSTALL COMPONENT 'file://component_log_sink_syseventlog';
SET GLOBAL log_error_services = 'log_filter_internal; log_sink_syseventlog';
```

To set `log_error_services` to take effect at server startup, use the instructions at Section 5.4.2.1, “Error Log Component Configuration”. Those instructions apply to other error-logging system variables as well.

Note

For MySQL 8.0 configuration, you must enable error logging to the system log explicitly. This differs from MySQL 5.7 and earlier, for which error logging to the system log is enabled by default on Windows, and on all platforms requires no component loading.

Error logging to the system log may require additional system configuration. Consult the system log documentation for your platform.

On Windows, error messages written to the Event Log within the Application log have these characteristics:

- Entries marked as `Error`, `Warning`, and `Note` are written to the Event Log, but not messages such as information statements from individual storage engines.
- Event Log entries have a source of `MySQL` (or `MySQL-`*`tag`* if `syseventlog.tag` is defined as *`tag`*).

On Unix and Unix-like systems, logging to the system log uses `syslog`. The following system variables affect `syslog` messages:

- `syseventlog.facility`: The default facility for `syslog` messages is `daemon`. Set this variable to specify a different facility.
- `syseventlog.include_pid`: Whether to include the server process ID in each line of `syslog` output.
- `syseventlog.tag`: This variable defines a tag to add to the server identifier (`mysqld`) in `syslog` messages. If defined, the tag is appended to the identifier with a leading hyphen.

Note

Prior to MySQL 8.0.13, use the `log_syslog_facility`, `log_syslog_include_pid`, and `log_syslog_tag` system variables rather than the `syseventlog.`*`xxx`* variables.

MySQL uses the custom label “System” for important system messages about non-error situations, such as startup, shutdown, and some significant changes to settings. In logs that do not support custom labels, including the Event Log on Windows, and `syslog` on Unix and Unix-like systems, system messages are assigned the label used for the information priority level. However, these messages are printed to the log even if the MySQL `log_error_verbosity` setting normally excludes messages at the information level.

When a log writer must fall back to a label of “Information” instead of “System” in this way, and the log event is further processed outside of the MySQL server (for example, filtered or forwarded by a `syslog` configuration), these events may by default be processed by the secondary application as being of “Information” priority rather than “System” priority.

5.4.2.8 Error Log Message Format

Each error log sink (writer) component has a characteristic output format it uses to write messages to its destination, but other factors may influence the content of the messages:

- The information available to the log writer. If a log filter component executed prior to execution of the writer component removes a log event attribute, that attribute is not available for writing. For information about log filtering, see Section 5.4.2.3, “Types of Error Log Filtering”.
- System variables may affect log writers. See System Variables That Affect Error Log Format.

For all log writers, the ID included in error log messages is that of the thread within `mysqld` responsible for writing the message. This indicates which part of the server produced the message, and is consistent with general query log and slow query log messages, which include the connection thread ID.

- Output Format for log_sink_internal
- Output Format for log_sink_json
- Output Format for log_sink_syseventlog
- Output Format for Early-Startup Logging
- System Variables That Affect Error Log Format

Output Format for log_sink_internal

This log writer produces the traditional error log output. It writes messages using this format:

```
timestamp thread_id [priority] [err_code] [subsystem] message
```

The `[` and `]` square bracket characters are literal characters in the message format. They do not indicate that fields are optional.

The `[err_code]` and `[subsystem]` fields were added in MySQL 8.0. They will be missing from logs generated by older servers. Log parsers can treat these fields as parts of the message text that will be present only for logs written by servers recent enough to include them. Parsers must treat the *`err_code`* part of `[err_code]` indicators as a string value.

Examples:

```
2018-03-22T12:35:47.538083Z 0 [Note] [MY-012487] [InnoDB] InnoDB: DDL log recovery : begin
2018-03-22T12:35:47.550565Z 0 [Warning] [MY-010068] [Server] CA certificate /var/mysql/sslinfo/cacert.pem is s
2018-03-22T12:35:47.669397Z 4 [Note] [MY-010051] [Server] Event Scheduler: scheduler thread started with id 4
2018-03-22T12:35:47.550939Z 0 [Note] [MY-010253] [Server] IPv6 is available.
```

Output Format for log_sink_json

The JSON-format log writer produces messages as JSON objects that contain key-value pairs. For example:

```
{ "prio": 3, "err_code": 10051, "subsystem": "Server",
  "source_file": "event_scheduler.cc", "function": "run",
  "msg": "Event Scheduler: scheduler thread started with id 4",
  "time": "2018-03-22T12:35:47.669397Z", "thread": 4,
  "err_symbol": "ER_SCHEDULER_STARTED", "SQL_state": "HY000",
  "label": "Note" }
```

Output Format for log_sink_syseventlog

The system log writer produces output that conforms to the system log format used on the local platform.

Output Format for Early-Startup Logging

The server generates some error log messages before startup options have been processed, and thus before it knows error log settings such as the `log_error_verbosity` and `log_timestamps` values, and which log components are to be used. The server handles error log messages that are generated early in the startup process as follows:

- Prior to MySQL 8.0.14, the server generates messages with the default timestamp, format, and verbosity level, and buffers them. After the startup options are processed and the error log configuration is known, the server flushes the buffered messages. Because these early messages use the default log configuration, they may differ from what is specified by the startup options. Also, the early messages

are not flushed to log writers other than the default. For example, logging to the JSON writer does not include these early messages because they are not in JSON format.

- As of MySQL 8.0.14, the server buffers log events rather than formatted log messages. This enables it to retroactively apply configuration settings to those events after the settings are known, with the result that flushed messages use the configured settings, not the defaults. Also, messages are flushed to all configured writers, not just the default writer.

  If a fatal error occurs before log configuration is known and the server must exit, the server so they are not lost. If no fatal error occurs but startup is excessively slow prior to processing startup options, the server periodically formats and flushes buffered messages using the logging defaults so as not to appear unresponsive. Although these behaviors are similar to pre-8.0.14 behavior in that the defaults are used, they are preferable to losing messages when exceptional conditions occur.

System Variables That Affect Error Log Format

The `log_timestamps` system variable controls the time zone of timestamps in messages written to the error log (as well as to general query log and slow query log files). Permitted values are `UTC` (the default) and `SYSTEM` (local system time zone).

5.4.2.9 Error Log File Flushing and Renaming

If you flush the error log using `FLUSH ERROR LOGS`, `FLUSH LOGS`, or `mysqladmin flush-logs`, the server closes and reopens any error log file to which it is writing. To rename an error log file, do so manually before flushing. Flushing the logs then opens a new file with the original file name. For example, assuming a log file name of *`host_name`*`.err`, to rename the file and create a new one, use the following commands:

```
mv host_name.err host_name.err-old
mysqladmin flush-logs
mv host_name.err-old backup-directory
```

On Windows, use `rename` rather than `mv`.

If the location of an error log file is not writable by the server, the log-flushing operation fails to create a new log file. For example, on Linux, the server might write the error log to the `/var/log/mysqld.log` file, where the `/var/log` directory is owned by `root` and is not writable by `mysqld`. For information about handling this case, see Section 5.4.7, “Server Log Maintenance”.

If the server is not writing to a named error log file, no error log file renaming occurs when the error log is flushed.

5.4.3 The General Query Log

The general query log is a general record of what `mysqld` is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to `mysqld`.

Each line that shows when a client connects also includes `using` *`connection_type`* to indicate the protocol used to establish the connection. *`connection_type`* is one of `TCP/IP` (TCP/IP connection established without SSL), `SSL/TLS` (TCP/IP connection established with SSL), `Socket` (Unix socket file connection), `Named Pipe` (Windows named pipe connection), or `Shared Memory` (Windows shared memory connection).

`mysqld` writes statements to the query log in the order that it receives them, which might differ from the order in which they are executed. This logging order is in contrast with that of the binary log, for which

statements are written after they are executed but before any locks are released. In addition, the query log may contain statements that only select data while such statements are never written to the binary log.

When using statement-based binary logging on a replication master server, statements received by its slaves are written to the query log of each slave. Statements are written to the query log of the master server if a client reads events with the `mysqlbinlog` utility and passes them to the server.

However, when using row-based binary logging, updates are sent as row changes rather than SQL statements, and thus these statements are never written to the query log when `binlog_format` is `ROW`. A given update also might not be written to the query log when this variable is set to `MIXED`, depending on the statement used. See Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”, for more information.

By default, the general query log is disabled. To specify the initial general query log state explicitly, use `--general_log[={0|1}]`. With no argument or an argument of 1, `--general_log` enables the log. With an argument of 0, this option disables the log. To specify a log file name, use `--general_log_file=file_name`. To specify the log destination, use the `log_output` system variable (as described in Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”).

Note

If you specify the `TABLE` log destination, see Log Tables and “Too many open files” Errors.

If you specify no name for the general query log file, the default name is *`host_name`*`.log`. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To disable or enable the general query log or change the log file name at runtime, use the global `general_log` and `general_log_file` system variables. Set `general_log` to 0 (or `OFF`) to disable the log or to 1 (or `ON`) to enable it. Set `general_log_file` to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

When the general query log is enabled, the server writes output to any destinations specified by the `log_output` system variable. If you enable the log, the server opens the log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected. If the destination is `NONE`, the server writes no queries even if the general log is enabled. Setting the log file name has no effect on logging if the log destination value does not contain `FILE`.

Server restarts and log flushing do not cause a new general query log file to be generated (although flushing closes and reopens it). To rename the file and create a new one, use the following commands:

```
shell> mv host_name.log host_name-old.log
shell> mysqladmin flush-logs
shell> mv host_name-old.log backup-directory
```

On Windows, use `rename` rather than `mv`.

You can also rename the general query log file at runtime by disabling the log:

```
SET GLOBAL general_log = 'OFF';
```

With the log disabled, rename the log file externally (for example, from the command line). Then enable the log again:

```
SET GLOBAL general_log = 'ON';
```

This method works on any platform and does not require a server restart.

To disable or enable general query logging for the current session, set the session `sql_log_off` variable to `ON` or `OFF`. (This assumes that the general query log itself is enabled.)

Passwords in statements written to the general query log are rewritten by the server not to occur literally in plain text. Password rewriting can be suppressed for the general query log by starting the server with the `--log-raw` option. This option may be useful for diagnostic purposes, to see the exact text of statements as received by the server, but for security reasons is not recommended for production use. See also Section 6.1.2.3, “Passwords and Logging”.

An implication of password rewriting is that statements that cannot be parsed (due, for example, to syntax errors) are not written to the general query log because they cannot be known to be password free. Use cases that require logging of all statements including those with errors should use the `--log-raw` option, bearing in mind that this also bypasses password rewriting.

Password rewriting occurs only when plain text passwords are expected. For statements with syntax that expect a password hash value, no rewriting occurs. If a plain text password is supplied erroneously for such syntax, the password is logged as given, without rewriting.

The `log_timestamps` system variable controls the time zone of timestamps in messages written to the general query log file (as well as to the slow query log file and the error log). It does not affect the time zone of general query log and slow query log messages written to log tables, but rows retrieved from those tables can be converted from the local system time zone to any desired time zone with `CONVERT_TZ()` or by setting the session `time_zone` system variable.

5.4.4 The Binary Log

The binary log contains “events” that describe database changes such as table creation operations or changes to table data. It also contains events for statements that potentially could have made changes (for example, a `DELETE` which matched no rows), unless row-based logging is used. The binary log also contains information about how long each statement took that updated data. The binary log has two important purposes:

- For replication, the binary log on a master replication server provides a record of the data changes to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See Section 17.2, “Replication Implementation”.
- Certain data recovery operations require use of the binary log. After a backup has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

The binary log is not used for statements such as `SELECT` or `SHOW` that do not modify data. To log all statements (for example, to identify a problem query), use the general query log. See Section 5.4.3, “The General Query Log”.

Running a server with binary logging enabled makes performance slightly slower. However, the benefits of the binary log in enabling you to set up replication and for restore operations generally outweigh this minor performance decrement.

The binary log is resilient to unexpected halts. Only complete events or transactions are logged or read back.

Passwords in statements written to the binary log are rewritten by the server not to occur literally in plain text. See also Section 6.1.2.3, “Passwords and Logging”.

From MySQL 8.0.14, binary log files and relay log files can be encrypted, helping to protect these files and the potentially sensitive data contained in them from being misused by outside attackers, and also from unauthorized viewing by users of the operating system where they are stored. You enable encryption on a MySQL server by setting the `binlog_encryption` system variable to `ON`. For more information, see Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”.

The following discussion describes some of the server options and variables that affect the operation of binary logging. For a complete list, see Section 17.1.6.4, “Binary Logging Options and Variables”.

Binary logging is enabled by default (the `log_bin` system variable is set to ON). The exception is if you use `mysqld` to initialize the data directory manually by invoking it with the `--initialize` or `--initialize-insecure` option, when binary logging is disabled by default, but can be enabled by specifying the `--log-bin` option.

To disable binary logging, you can specify the `--skip-log-bin` or `--disable-log-bin` option at startup. If either of these options is specified and `--log-bin` is also specified, the option specified later takes precedence.

The `--log-slave-updates` and `--slave-preserve-commit-order` options require binary logging. If you disable binary logging, either omit these options, or specify `--log-slave-updates=OFF` and `--skip-slave-preserve-commit-order`. MySQL disables these options by default when `--skip-log-bin` or `--disable-log-bin` is specified. If you specify `--log-slave-updates` or `--slave-preserve-commit-order` together with `--skip-log-bin` or `--disable-log-bin`, a warning or error message is issued.

The `--log-bin[=base_name]` option is used to specify the base name for binary log files. If you do not supply the `--log-bin` option, MySQL uses `binlog` as the default base name for the binary log files. For compatibility with earlier releases, if you supply the `--log-bin` option with no string or with an empty string, the base name defaults to *`host_name`*`-bin`, using the name of the host machine. It is recommended that you specify a base name, so that if the host name changes, you can easily continue to use the same binary log file names (see Section B.4.7, “Known Issues in MySQL”). If you supply an extension in the log name (for example, `--log-bin=`*`base_name.extension`*), the extension is silently removed and ignored.

`mysqld` appends a numeric extension to the binary log base name to generate binary log file names. The number increases each time the server creates a new log file, thus creating an ordered series of files. The server creates a new file in the series each time it starts or flushes the logs. The server also creates a new binary log file automatically after the current log's size reaches `max_binlog_size`. A binary log file may become larger than `max_binlog_size` if you are using large transactions because a transaction is written to the file in one piece, never split between files.

To keep track of which binary log files have been used, `mysqld` also creates a binary log index file that contains the names of the binary log files. By default, this has the same base name as the binary log file, with the extension `'.index'`. You can change the name of the binary log index file with the `--log-bin-index[=`*`file_name`*`]` option. You should not manually edit this file while `mysqld` is running; doing so would confuse `mysqld`.

The term “binary log file” generally denotes an individual numbered file containing database events. The term “binary log” collectively denotes the set of numbered binary log files plus the index file.

The default location for binary log files and the binary log index file is the data directory. You can use the `--log-bin` option to specify an alternative location, by adding a leading absolute path name to the base name to specify a different directory. When the server reads an entry from the binary log index file, which tracks the binary log files that have been used, it checks whether the entry contains a relative path. If it does, the relative part of the path is replaced with the absolute path set using the `--log-bin` option. An absolute path recorded in the binary log index file remains unchanged; in such a case, the index file

must be edited manually to enable a new path or paths to be used. The binary log file base name and any specified path are available as the `log_bin_basename` system variable.

In MySQL 5.7, a server ID had to be specified when binary logging was enabled, or the server would not start. In MySQL 8.0, the `server_id` system variable is set to 1 by default. The server can be started with this default ID when binary logging is enabled, but an informational message is issued if you do not specify a server ID explicitly using the `--server-id` option. For servers that are used in a replication topology, you must specify a unique nonzero server ID for each server.

A client that has privileges sufficient to set restricted session system variables (see Section 5.1.9.1, “System Variable Privileges”) can disable binary logging of its own statements by using a `SET sql_log_bin=OFF` statement.

By default, the server logs the length of the event as well as the event itself and uses this to verify that the event was written correctly. You can also cause the server to write checksums for the events by setting the `binlog_checksum` system variable. When reading back from the binary log, the master uses the event length by default, but can be made to use checksums if available by enabling the `master_verify_checksum` system variable. The slave I/O thread also verifies events received from the master. You can cause the slave SQL thread to use checksums if available when reading from the relay log by enabling the `slave_sql_verify_checksum` system variable.

The format of the events recorded in the binary log is dependent on the binary logging format. Three format types are supported: row-based logging, statement-based logging and mixed-base logging. The binary logging format used depends on the MySQL version. For general descriptions of the logging formats, see Section 5.4.4.1, “Binary Logging Formats”. For detailed information about the format of the binary log, see MySQL Internals: The Binary Log.

The server evaluates the `--binlog-do-db` and `--binlog-ignore-db` options in the same way as it does the `--replicate-do-db` and `--replicate-ignore-db` options. For information about how this is done, see Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”.

A replication slave server is started with the `log_slave_updates` system variable enabled by default, meaning that the slave writes to its own binary log any data modifications that are received from the replication master. The binary log must be enabled for this setting to work (see Section 17.1.6.3, “Replication Slave Options and Variables”). This setting enables the slave to act as a master to other slaves in chained replication.

You can delete all binary log files with the `RESET MASTER` statement, or a subset of them with `PURGE BINARY LOGS`. See Section 13.7.8.6, “RESET Syntax”, and Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

If you are using replication, you should not delete old binary log files on the master until you are sure that no slave still needs to use them. For example, if your slaves never run more than three days behind, once a day you can execute `mysqladmin flush-logs` on the master and then remove any logs that are more than three days old. You can remove the files manually, but it is preferable to use `PURGE BINARY LOGS`, which also safely updates the binary log index file for you (and which can take a date argument). See Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

You can display the contents of binary log files with the `mysqlbinlog` utility. This can be useful when you want to reprocess statements in the log for a recovery operation. For example, you can update a MySQL server from the binary log as follows:

```
shell> mysqlbinlog log_file | mysql -h server_name
```

`mysqlbinlog` also can be used to display replication slave relay log file contents because they are written using the same format as binary log files. For more information on the `mysqlbinlog` utility and how to use

it, see Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”. For more information about the binary log and recovery operations, see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

Binary logging is done immediately after a statement or transaction completes but before any locks are released or any commit is done. This ensures that the log is logged in commit order.

Updates to nontransactional tables are stored in the binary log immediately after execution.

Within an uncommitted transaction, all updates (UPDATE, DELETE, or INSERT) that change transactional tables such as InnoDB tables are cached until a COMMIT statement is received by the server. At that point, mysqld writes the entire transaction to the binary log before the COMMIT is executed.

Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a ROLLBACK statement at the end to ensure that the modifications to those tables are replicated.

When a thread that handles the transaction starts, it allocates a buffer of binlog_cache_size to buffer statements. If a statement is bigger than this, the thread opens a temporary file to store the transaction. The temporary file is deleted when the thread ends. From MySQL 8.0.17, if binary log encryption is active on the server, the temporary file is encrypted.

The Binlog_cache_use status variable shows the number of transactions that used this buffer (and possibly a temporary file) for storing statements. The Binlog_cache_disk_use status variable shows how many of those transactions actually had to use a temporary file. These two variables can be used for tuning binlog_cache_size to a large enough value that avoids the use of temporary files.

The max_binlog_cache_size system variable (default 4GB, which is also the maximum) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this many bytes, it fails and rolls back. The minimum value is 4096.

If you are using the binary log and row based logging, concurrent inserts are converted to normal inserts for CREATE ... SELECT or INSERT ... SELECT statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. If you are using statement-based logging, the original statement is written to the log.

The binary log format has some known limitations that can affect recovery from backups. See Section 17.5.1, “Replication Features and Issues”.

Binary logging for stored programs is done as described in Section 24.7, “Stored Program Binary Logging”.

Note that the binary log format differs in MySQL 8.0 from previous versions of MySQL, due to enhancements in replication. See Section 17.5.2, “Replication Compatibility Between MySQL Versions”.

If the server is unable to write to the binary log, flush binary log files, or synchronize the binary log to disk, the binary log on the replication master can become inconsistent and replication slaves can lose synchronization with the master. The binlog_error_action system variable controls the action taken if an error of this type is encountered with the binary log.

- The default setting, ABORT_SERVER, makes the server halt binary logging and shut down. At this point, you can identify and correct the cause of the error. On restart, recovery proceeds as in the case of an unexpected server halt (see Section 17.4.2, “Handling an Unexpected Halt of a Replication Slave”).
- The setting IGNORE_ERROR provides backward compatibility with older versions of MySQL. With this setting, the server continues the ongoing transaction and logs the error, then halts binary logging, but continues to perform updates. At this point, you can identify and correct the cause of the error. To resume binary logging, log_bin must be enabled again, which requires a server restart. Only use this

option if you require backward compatibility, and the binary log is non-essential on this MySQL server instance. For example, you might use the binary log only for intermittent auditing or debugging of the server, and not use it for replication from the server or rely on it for point-in-time restore operations.

By default, the binary log is synchronized to disk at each write (`sync_binlog=1`). If `sync_binlog` was not enabled, and the operating system or machine (not only the MySQL server) crashed, there is a chance that the last statements of the binary log could be lost. To prevent this, enable the `sync_binlog` system variable to synchronize the binary log to disk after every *`N`* commit groups. See Section 5.1.8, “Server System Variables”. The safest value for `sync_binlog` is 1 (the default), but this is also the slowest.

In earlier MySQL releases, there was a chance of inconsistency between the table content and binary log content if a crash occurred, even with `sync_binlog` set to 1. For example, if you are using `InnoDB` tables and the MySQL server processes a `COMMIT` statement, it writes many prepared transactions to the binary log in sequence, synchronizes the binary log, and then commits the transaction into `InnoDB`. If the server crashed between those two operations, the transaction would be rolled back by `InnoDB` at restart but still exist in the binary log. Such an issue was resolved in previous releases by enabling `InnoDB` support for two-phase commit in XA transactions. In 5.8.0 and higher, the `InnoDB` support for two-phase commit in XA transactions is always enabled.

`InnoDB` support for two-phase commit in XA transactions ensures that the binary log and `InnoDB` data files are synchronized. However, the MySQL server should also be configured to synchronize the binary log and the `InnoDB` logs to disk before committing the transaction. The `InnoDB` logs are synchronized by default, and `sync_binlog=1` ensures the binary log is synchronized. The effect of implicit `InnoDB` support for two-phase commit in XA transactions and `sync_binlog=1` is that at restart after a crash, after doing a rollback of transactions, the MySQL server scans the latest binary log file to collect transaction *`xid`* values and calculate the last valid position in the binary log file. The MySQL server then tells `InnoDB` to complete any prepared transactions that were successfully written to the to the binary log, and truncates the binary log to the last valid position. This ensures that the binary log reflects the exact data of `InnoDB` tables, and therefore the slave remains in synchrony with the master because it does not receive a statement which has been rolled back.

If the MySQL server discovers at crash recovery that the binary log is shorter than it should have been, it lacks at least one successfully committed `InnoDB` transaction. This should not happen if `sync_binlog=1` and the disk/file system do an actual sync when they are requested to (some do not), so the server prints an error message `The binary log file_name is shorter than its expected size`. In this case, this binary log is not correct and replication should be restarted from a fresh snapshot of the master's data.

The session values of the following system variables are written to the binary log and honored by the replication slave when parsing the binary log:

- `sql_mode` (except that the `NO_DIR_IN_CREATE` mode is not replicated; see Section 17.5.1.38, “Replication and Variables”)
- `foreign_key_checks`
- `unique_checks`
- `character_set_client`
- `collation_connection`
- `collation_database`
- `collation_server`
- `sql_auto_is_null`

5.4.4.1 Binary Logging Formats

The server uses several logging formats to record information in the binary log:

- Replication capabilities in MySQL originally were based on propagation of SQL statements from master to slave. This is called *statement-based logging*. You can cause this format to be used by starting the server with `--binlog-format=STATEMENT`.
- In *row-based logging* (the default), the master writes events to the binary log that indicate how individual table rows are affected. You can cause the server to use row-based logging by starting it with `--binlog-format=ROW`.
- A third option is also available: *mixed logging*. With mixed logging, statement-based logging is used by default, but the logging mode switches automatically to row-based in certain cases as described below. You can cause MySQL to use mixed logging explicitly by starting `mysqld` with the option `--binlog-format=MIXED`.

The logging format can also be set or limited by the storage engine being used. This helps to eliminate issues when replicating certain statements between a master and slave which are using different storage engines.

With statement-based replication, there may be issues with replicating nondeterministic statements. In deciding whether or not a given statement is safe for statement-based replication, MySQL determines whether it can guarantee that the statement can be replicated using statement-based logging. If MySQL cannot make this guarantee, it marks the statement as potentially unreliable and issues the warning, `Statement may not be safe to log in statement format`.

You can avoid these issues by using MySQL's row-based replication instead.

5.4.4.2 Setting The Binary Log Format

You can select the binary logging format explicitly by starting the MySQL server with `--binlog-format=`*`type`*. The supported values for *`type`* are:

- `STATEMENT` causes logging to be statement based.
- `ROW` causes logging to be row based. This is the default.
- `MIXED` causes logging to use mixed format.

The logging format also can be switched at runtime, although note that there are a number of situations in which you cannot do this, as discussed later in this section. Set the global value of the `binlog_format` system variable to specify the format for clients that connect subsequent to the change:

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';
mysql> SET GLOBAL binlog_format = 'ROW';
mysql> SET GLOBAL binlog_format = 'MIXED';
```

An individual client can control the logging format for its own statements by setting the session value of `binlog_format`:

```
mysql> SET SESSION binlog_format = 'STATEMENT';
mysql> SET SESSION binlog_format = 'ROW';
mysql> SET SESSION binlog_format = 'MIXED';
```

Changing the global `binlog_format` value requires privileges sufficient to set global system variables. Changing the session `binlog_format` value requires privileges sufficient to set restricted session system variables. See Section 5.1.9.1, “System Variable Privileges”.

There are several reasons why a client might want to set binary logging on a per-session basis:

- A session that makes many small changes to the database might want to use row-based logging.
- A session that performs updates that match many rows in the `WHERE` clause might want to use statement-based logging because it will be more efficient to log a few statements than many rows.
- Some statements require a lot of execution time on the master, but result in just a few rows being modified. It might therefore be beneficial to replicate them using row-based logging.

There are exceptions when you cannot switch the replication format at runtime:

- The replication format cannot be changed from within a stored function or a trigger.
- If the `NDB` storage engine is enabled.
- If a session has open temporary tables, the replication format cannot be changed for the session (`SET @@SESSION.binlog_format`).
- If any replication channel has open temporary tables, the replication format cannot be changed globally (`SET @@GLOBAL.binlog_format` or `SET @@PERSIST.binlog_format`).
- If any replication channel applier thread is currently running, the replication format cannot be changed globally (`SET @@GLOBAL.binlog_format` or `SET @@PERSIST.binlog_format`).

Trying to switch the replication format in any of these cases (or attempting to set the current replication format) results in an error. You can, however, use `PERSIST_ONLY` (`SET @@PERSIST_ONLY.binlog_format`) to change the replication format at any time, because this action does not modify the runtime global system variable value, and takes effect only after a server restart.

Switching the replication format at runtime is not recommended when any temporary tables exist, because temporary tables are logged only when using statement-based replication, whereas with row-based replication and mixed replication, they are not logged.

Switching the replication format while replication is ongoing can also cause issues. Each MySQL Server can set its own and only its own binary logging format (true whether `binlog_format` is set with global or session scope). This means that changing the logging format on a replication master does not cause a slave to change its logging format to match. When using `STATEMENT` mode, the `binlog_format` system variable is not replicated. When using `MIXED` or `ROW` logging mode, it is replicated but is ignored by the slave.

A replication slave is not able to convert binary log entries received in `ROW` logging format to `STATEMENT` format for use in its own binary log. The slave must therefore use `ROW` or `MIXED` format if the master does. Changing the binary logging format on the master from `STATEMENT` to `ROW` or `MIXED` while replication is ongoing to a slave with `STATEMENT` format can cause replication to fail with errors such as `Error executing row event: 'Cannot execute statement: impossible to write to binary log since statement is in row format and BINLOG_FORMAT = STATEMENT.'` Changing the binary logging format on the slave to `STATEMENT` format when the master is still using `MIXED` or `ROW` format also causes the same type of replication failure. To change the format safely, you must stop replication and ensure that the same change is made on both the master and the slave.

If you are using `InnoDB` tables and the transaction isolation level is `READ COMMITTED` or `READ UNCOMMITTED`, only row-based logging can be used. It is *possible* to change the logging format to `STATEMENT`, but doing so at runtime leads very rapidly to errors because `InnoDB` can no longer perform inserts.

With the binary log format set to `ROW`, many changes are written to the binary log using the row-based format. Some changes, however, still use the statement-based format. Examples include all DDL (data definition language) statements such as `CREATE TABLE`, `ALTER TABLE`, or `DROP TABLE`.

When row-based binary logging is used, the `binlog_row_event_max_size` system variable and its corresponding startup option `--binlog-row-event-max-size` set a soft limit on the maximum size of row events. The default value is 8192 bytes, and the value can only be changed at server startup. Where possible, rows stored in the binary log are grouped into events with a size not exceeding the value of this setting. If an event cannot be split, the maximum size can be exceeded.

The `--binlog-row-event-max-size` option is available for servers that are capable of row-based replication. Rows are stored into the binary log in chunks having a size in bytes not exceeding the value of this option. The value must be a multiple of 256. The default value is 8192.

Warning

When using *statement-based logging* for replication, it is possible for the data on the master and slave to become different if a statement is designed in such a way that the data modification is *nondeterministic*; that is, it is left to the will of the query optimizer. In general, this is not a good practice even outside of replication. For a detailed explanation of this issue, see Section B.4.7, “Known Issues in MySQL”.

For information about logs kept by replication slaves, see Section 17.2.4, “Replication Relay and Status Logs”.

5.4.4.3 Mixed Binary Logging Format

When running in `MIXED` logging format, the server automatically switches from statement-based to row-based logging under the following conditions:

- When a function contains `UUID()`.
- When one or more tables with `AUTO_INCREMENT` columns are updated and a trigger or stored function is invoked. Like all other unsafe statements, this generates a warning if `binlog_format = STATEMENT`.

  For more information, see Section 17.5.1.1, “Replication and AUTO_INCREMENT”.
- When the body of a view requires row-based replication, the statement creating the view also uses it. For example, this occurs when the statement creating a view uses the `UUID()` function.
- When a call to a UDF is involved.
- When `FOUND_ROWS()` or `ROW_COUNT()` is used. (Bug #12092, Bug #30244)
- When `USER()`, `CURRENT_USER()`, or `CURRENT_USER` is used. (Bug #28086)
- When one of the tables involved is a log table in the `mysql` database.
- When the `LOAD_FILE()` function is used. (Bug #39701)
- When a statement refers to one or more system variables. (Bug #31168)

  Exception. The following system variables, when used with session scope (only), do not cause the logging format to switch:

  - `auto_increment_increment`
  - `auto_increment_offset`
  - `character_set_client`
  - `character_set_connection`

- `character_set_database`
- `character_set_server`
- `collation_connection`
- `collation_database`
- `collation_server`
- `foreign_key_checks`
- `identity`
- `last_insert_id`
- `lc_time_names`
- `pseudo_thread_id`
- `sql_auto_is_null`
- `time_zone`
- `timestamp`
- `unique_checks`

For information about determining system variable scope, see Section 5.1.9, “Using System Variables”.

For information about how replication treats `sql_mode`, see Section 17.5.1.38, “Replication and Variables”.

In earlier releases, when mixed binary logging format was in use, if a statement was logged by row and the session that executed the statement had any temporary tables, all subsequent statements were treated as unsafe and logged in row-based format until all temporary tables in use by that session were dropped. As of MySQL 8.0, operations on temporary tables are not logged in mixed binary logging format, and the presence of temporary tables in the session has no impact on the logging mode used for each statement.

Note

A warning is generated if you try to execute a statement using statement-based logging that should be written using row-based logging. The warning is shown both in the client (in the output of `SHOW WARNINGS`) and through the `mysqld` error log. A warning is added to the `SHOW WARNINGS` table each time such a statement is executed. However, only the first statement that generated the warning for each client session is written to the error log to prevent flooding the log.

In addition to the decisions above, individual engines can also determine the logging format used when information in a table is updated. The logging capabilities of an individual engine can be defined as follows:

- If an engine supports row-based logging, the engine is said to be *row-logging capable*.
- If an engine supports statement-based logging, the engine is said to be *statement-logging capable*.

A given storage engine can support either or both logging formats. The following table lists the formats supported by each engine.

Storage Engine	Row Logging Supported	Statement Logging Supported
ARCHIVE	Yes	Yes
BLACKHOLE	Yes	Yes
CSV	Yes	Yes
EXAMPLE	Yes	No
FEDERATED	Yes	Yes
HEAP	Yes	Yes
InnoDB	Yes	Yes when the transaction isolation level is REPEATABLE READ or SERIALIZABLE; No otherwise.
MyISAM	Yes	Yes
MERGE	Yes	Yes
NDB	Yes	No

Whether a statement is to be logged and the logging mode to be used is determined according to the type of statement (safe, unsafe, or binary injected), the binary logging format (STATEMENT, ROW, or MIXED), and the logging capabilities of the storage engine (statement capable, row capable, both, or neither). (Binary injection refers to logging a change that must be logged using ROW format.)

Statements may be logged with or without a warning; failed statements are not logged, but generate errors in the log. This is shown in the following decision table. **Type**, **binlog_format**, **SLC**, and **RLC** columns outline the conditions, and **Error / Warning** and **Logged as** columns represent the corresponding actions. **SLC** stands for “statement-logging capable”, and **RLC** stands for “row-logging capable”.

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
*	*	No	No	Error: Cannot execute statement: Binary logging is impossible since at least one engine is involved that is both row-incapable and statement-incapable.	-
Safe	STATEMENT	Yes	No	-	STATEMENT
Safe	MIXED	Yes	No	-	STATEMENT
Safe	ROW	Yes	No	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = ROW and at least one table uses a storage engine that	-

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
				is not capable of row-based logging.	
Unsafe	STATEMENT	Yes	No	Warning: Unsafe statement binlogged in statement format, since BINLOG_FORMAT = STATEMENT	STATEMENT
Unsafe	MIXED	Yes	No	Error: Cannot execute statement: Binary logging of an unsafe statement is impossible when the storage engine is limited to statement-based logging, even if BINLOG_FORMAT = MIXED.	-
Unsafe	ROW	Yes	No	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = ROW and at least one table uses a storage engine that is not capable of row-based logging.	-
Row Injection	STATEMENT	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-
Row Injection	MIXED	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not	-

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
				capable of row-based logging.	
Row Injection	ROW	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-
Safe	STATEMENT	No	Yes	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine that is not capable of statement-based logging.	-
Safe	MIXED	No	Yes	-	ROW
Safe	ROW	No	Yes	-	ROW
Unsafe	STATEMENT	No	Yes	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine that is not capable of statement-based logging.	-
Unsafe	MIXED	No	Yes	-	ROW
Unsafe	ROW	No	Yes	-	ROW
Row Injection	STATEMENT	No	Yes	Error: Cannot execute row injection: Binary logging is not possible since BINLOG_FORMAT = STATEMENT.	-
Row Injection	MIXED	No	Yes	-	ROW

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
Row Injection	ROW	No	Yes	-	ROW
Safe	STATEMENT	Yes	Yes	-	STATEMENT
Safe	MIXED	Yes	Yes	-	STATEMENT
Safe	ROW	Yes	Yes	-	ROW
Unsafe	STATEMENT	Yes	Yes	Warning: Unsafe statement binlogged in statement format since BINLOG_FORMAT = STATEMENT.	STATEMENT
Unsafe	MIXED	Yes	Yes	-	ROW
Unsafe	ROW	Yes	Yes	-	ROW
Row Injection	STATEMENT	Yes	Yes	Error: Cannot execute row injection: Binary logging is not possible because BINLOG_FORMAT = STATEMENT.	-
Row Injection	MIXED	Yes	Yes	-	ROW
Row Injection	ROW	Yes	Yes	-	ROW

When a warning is produced by the determination, a standard MySQL warning is produced (and is available using SHOW WARNINGS). The information is also written to the mysqld error log. Only one error for each error instance per client connection is logged to prevent flooding the log. The log message includes the SQL statement that was attempted.

If a slave server has log_error_verbosity set to display warnings, the slave prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, statements that are unsafe for statement-based logging, and so forth.

5.4.4.4 Logging Format for Changes to mysql Database Tables

The contents of the grant tables in the mysql database can be modified directly (for example, with INSERT or DELETE) or indirectly (for example, with GRANT or CREATE USER). Statements that affect mysql database tables are written to the binary log using the following rules:

- Data manipulation statements that change data in mysql database tables directly are logged according to the setting of the binlog_format system variable. This pertains to statements such as INSERT, UPDATE, DELETE, REPLACE, DO, LOAD DATA, SELECT, and TRUNCATE TABLE.
- Statements that change the mysql database indirectly are logged as statements regardless of the value of binlog_format. This pertains to statements such as GRANT, REVOKE, SET PASSWORD, RENAME USER, CREATE (all forms except CREATE TABLE ... SELECT), ALTER (all forms), and DROP (all forms).

`CREATE TABLE ... SELECT` is a combination of data definition and data manipulation. The `CREATE TABLE` part is logged using statement format and the `SELECT` part is logged according to the value of `binlog_format`.

5.4.5 The Slow Query Log

The slow query log consists of SQL statements that take more than `long_query_time` seconds to execute and require at least `min_examined_row_limit` rows to be examined. The slow query log can be used to find queries that take a long time to execute and are therefore candidates for optimization. However, examining a long slow query log can be a time-consuming task. To make this easier, you can use the `mysqldumpslow` command to process a slow query log file and summarize its contents. See Section 4.6.9, "`mysqldumpslow` — Summarize Slow Query Log Files".

The time to acquire the initial locks is not counted as execution time. `mysqld` writes a statement to the slow query log after it has been executed and after all locks have been released, so log order might differ from execution order.

- Slow Query Log Parameters
- Slow Query Log Contents

Slow Query Log Parameters

The minimum and default values of `long_query_time` are 0 and 10, respectively. The value can be specified to a resolution of microseconds.

By default, administrative statements are not logged, nor are queries that do not use indexes for lookups. This behavior can be changed using `log_slow_admin_statements` and `log_queries_not_using_indexes`, as described later.

By default, the slow query log is disabled. To specify the initial slow query log state explicitly, use `--slow_query_log[={0|1}]`. With no argument or an argument of 1, `--slow_query_log` enables the log. With an argument of 0, this option disables the log. To specify a log file name, use `--slow_query_log_file=`*`file_name`*. To specify the log destination, use the `log_output` system variable (as described in Section 5.4.1, "Selecting General Query Log and Slow Query Log Output Destinations").

Note

If you specify the `TABLE` log destination, see Log Tables and "Too many open files" Errors.

If you specify no name for the slow query log file, the default name is *`host_name`*`-slow.log`. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To disable or enable the slow query log or change the log file name at runtime, use the global `slow_query_log` and `slow_query_log_file` system variables. Set `slow_query_log` to 0 to disable the log or to 1 to enable it. Set `slow_query_log_file` to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

The server writes less information to the slow query log if you use the `--log-short-format` option.

To include slow administrative statements in the slow query log, enable the `log_slow_admin_statements` system variable. Administrative statements include `ALTER TABLE`, `ANALYZE TABLE`, `CHECK TABLE`, `CREATE INDEX`, `DROP INDEX`, `OPTIMIZE TABLE`, and `REPAIR TABLE`.

To include queries that do not use indexes for row lookups in the statements written to the slow query log, enable the `log_queries_not_using_indexes` system variable. (Even with that variable enabled, the server does not log queries that would not benefit from the presence of an index due to the table having fewer than two rows.)

When queries that do not use an index are logged, the slow query log may grow quickly. It is possible to put a rate limit on these queries by setting the `log_throttle_queries_not_using_indexes` system variable. By default, this variable is 0, which means there is no limit. Positive values impose a per-minute limit on logging of queries that do not use indexes. The first such query opens a 60-second window within which the server logs queries up to the given limit, then suppresses additional queries. If there are suppressed queries when the window ends, the server logs a summary that indicates how many there were and the aggregate time spent in them. The next 60-second window begins when the server logs the next query that does not use indexes.

The server uses the controlling parameters in the following order to determine whether to write a query to the slow query log:

1. The query must either not be an administrative statement, or `log_slow_admin_statements` must be enabled.
2. The query must have taken at least `long_query_time` seconds, or `log_queries_not_using_indexes` must be enabled and the query used no indexes for row lookups.
3. The query must have examined at least `min_examined_row_limit` rows.
4. The query must not be suppressed according to the `log_throttle_queries_not_using_indexes` setting.

The `log_timestamps` system variable controls the time zone of timestamps in messages written to the slow query log file (as well as to the general query log file and the error log). It does not affect the time zone of general query log and slow query log messages written to log tables, but rows retrieved from those tables can be converted from the local system time zone to any desired time zone with `CONVERT_TZ()` or by setting the session `time_zone` system variable.

By default, a replication slave does not write replicated queries to the slow query log. To change this, enable the `log_slow_slave_statements` system variable. Note that if row-based replication is in use (`binlog_format=ROW`), `log_slow_slave_statements` has no effect. Queries are only added to the slave's slow query log when they are logged in statement format in the binary log, that is, when `binlog_format=STATEMENT` is set, or when `binlog_format=MIXED` is set and the statement is logged in statement format. Slow queries that are logged in row format when `binlog_format=MIXED` is set, or that are logged when `binlog_format=ROW` is set, are not added to the slave's slow query log, even if `log_slow_slave_statements` is enabled.

Slow Query Log Contents

When the slow query log is enabled, the server writes output to any destinations specified by the `log_output` system variable. If you enable the log, the server opens the log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected. If the destination is `NONE`, the server writes no queries even if the slow query log is enabled. Setting the log file name has no effect on logging if `FILE` is not selected as an output destination.

If the slow query log is enabled and `FILE` is selected as an output destination, each statement written to the log is preceded by a line that begins with a `#` character and has these fields (with all fields on a single line):

- `Query_time: duration`

The statement execution time in seconds.

- `Lock_time: duration`

  The time to acquire locks in seconds.

- `Rows_sent: N`

  The number of rows sent to the client.

- `Rows_examined:`

  The number of rows examined by the server layer (not counting any processing internal to storage engines).

Enabling the `log_slow_extra` system variable (available as of MySQL 8.0.14) causes the server to write the following extra fields to `FILE` output in addition to those just listed (`TABLE` output is unaffected). Some field descriptions refer to status variable names. Consult the status variable descriptions for more information. However, in the slow query log, the counters are per-statement values, not cumulative per-session values.

- `Thread_id: ID`

  The statement thread identifier.

- `Errno: error_number`

  The statement error number, or 0 if no error occurred.

- `Killed: N`

  If the statement was terminated, the error number indicating why, or 0 if the statement terminated normally.

- `Bytes_received: N`

  The `Bytes_received` value for the statement.

- `Bytes_sent: N`

  The `Bytes_sent` value for the statement.

- `Read_first: N`

  The `Handler_read_first` value for the statement.

- `Read_last: N`

  The `Handler_read_last` value for the statement.

- `Read_key: N`

  The `Handler_read_key` value for the statement.

- `Read_next: N`

  The `Handler_read_next` value for the statement.

- `Read_prev: N`

The `Handler_read_prev` value for the statement.

- `Read_rnd: N`

  The `Handler_read_rnd` value for the statement.

- `Read_rnd_next: N`

  The `Handler_read_rnd_next` value for the statement.

- `Sort_merge_passes: N`

  The `Sort_merge_passes` value for the statement.

- `Sort_range_count: N`

  The `Sort_range` value for the statement.

- `Sort_rows: N`

  The `Sort_rows` value for the statement.

- `Sort_scan_count: N`

  The `Sort_scan` value for the statement.

- `Created_tmp_disk_tables: N`

  The `Created_tmp_disk_tables` value for the statement.

- `Created_tmp_tables: N`

  The `Created_tmp_tables` value for the statement.

- `Start: timestamp`

  The statement execution start time.

- `End: timestamp`

  The statement execution end time.

A given slow query log file may contain a mix of lines with and without the extra fields added by enabling `log_slow_extra`. Log file analyzers can determine whether a line contains the additional fields by the field count.

Each statement written to the slow query log file is preceded by a `SET` statement that includes a timestamp. As of MySQL 8.0.14, the timestamp indicates when the slow statement began executing. Prior to 8.0.14, the timestamp indicates when the slow statement was logged (which occurs after the statement finishes executing).

Passwords in statements written to the slow query log are rewritten by the server not to occur literally in plain text. See Section 6.1.2.3, “Passwords and Logging”.

5.4.6 The DDL Log

The DDL log, or metadata log, records metadata operations generated by data definition statements such as `DROP TABLE` and `ALTER TABLE`. MySQL uses this log to recover from crashes occurring in the middle

of a metadata operation. When executing the statement `DROP TABLE t1, t2`, we need to ensure that both `t1` and `t2` are dropped, and that each table drop is complete. Another example of this type of SQL statement is `ALTER TABLE t3 DROP PARTITION p2`, where we must make certain that the partition is completely dropped and that its definition is removed from the list of partitions for table `t3`.

A record of metadata operations such as those just described are written to the file `ddl_log.log`, in the MySQL data directory. This is a binary file; it is not intended to be human-readable, and you should not attempt to modify its contents in any way.

`ddl_log.log` is not created until it is actually needed for recording metadata statements, and is removed following a successful start of `mysqld`. Thus, it is possible for this file not to be present on a MySQL server that is functioning in a completely normal manner.

Currently, `ddl_log.log` can hold up to 1048573 entries, equivalent to 4 GB in size. Once this limit is exceeded, you must rename or remove the file before it is possible to execute any additional DDL statements. This is a known issue which we are working to resolve (Bug #83708).

There are no user-configurable server options or variables associated with this file.

5.4.7 Server Log Maintenance

As described in Section 5.4, “MySQL Server Logs”, MySQL Server can create several different log files to help you see what activity is taking place. However, you must clean up these files regularly to ensure that the logs do not take up too much disk space.

When using MySQL with logging enabled, you may want to back up and remove old log files from time to time and tell MySQL to start logging to new files. See Section 7.2, “Database Backup Methods”.

On a Linux (Red Hat) installation, you can use the `mysql-log-rotate` script for this. If you installed MySQL from an RPM distribution, this script should have been installed automatically. Be careful with this script if you are using the binary log for replication. You should not remove binary logs until you are certain that their contents have been processed by all slaves.

On other systems, you must install a short script yourself that you start from `cron` (or its equivalent) for handling log files.

Binary log files are automatically removed after the server's binary log expiration period. Removal of the files can take place at startup and when the binary log is flushed. The default binary log expiration period is 30 days. You can specify an alternative expiration period using the `binlog_expire_logs_seconds` system variable. If you are using replication, you should specify an expiration period that is no lower than the maximum amount of time your slaves might lag behind the master. To remove binary logs on demand, use the `PURGE BINARY LOGS` statement (see Section 13.4.1.1, “PURGE BINARY LOGS Syntax”).

You can force MySQL to start using new log files by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement or execute a `mysqladmin flush-logs`, `mysqladmin refresh`, `mysqldump --flush-logs`, or `mysqldump --master-data` command. See Section 13.7.8.3, “FLUSH Syntax”, Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”, and Section 4.5.4, “`mysqldump` — A Database Backup Program”. In addition, the binary log is flushed when its size reaches the value of the `max_binlog_size` system variable.

`FLUSH LOGS` supports optional modifiers to enable selective flushing of individual logs (for example, `FLUSH BINARY LOGS`).

A log-flushing operation does the following:

- If general query logging or slow query logging to a log file is enabled, the server closes and reopens the general query log file or slow query log file.

- If binary logging is enabled, the server closes the current binary log file and opens a new log file with the next sequence number.
- If the server was started with the `--log-error` option to cause the error log to be written to a file, the server closes and reopens the log file.

The server creates a new binary log file when you flush the logs. However, it just closes and reopens the general and slow query log files. To cause new files to be created on Unix, rename the current log files before flushing them. At flush time, the server opens new log files with the original names. For example, if the general and slow query log files are named `mysql.log` and `mysql-slow.log`, you can use a series of commands like this:

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mv mysql-slow.log mysql-slow.old
shell> mysqladmin flush-logs
```

On Windows, use `rename` rather than `mv`.

At this point, you can make a backup of `mysql.old` and `mysql-slow.old` and then remove them from disk.

A similar strategy can be used to back up the error log file, if there is one.

You can rename the general query log or slow query log at runtime by disabling the log:

```
SET GLOBAL general_log = 'OFF';
SET GLOBAL slow_query_log = 'OFF';
```

With the logs disabled, rename the log files externally (for example, from the command line). Then enable the logs again:

```
SET GLOBAL general_log = 'ON';
SET GLOBAL slow_query_log = 'ON';
```

This method works on any platform and does not require a server restart.

Note

For the server to recreate a given log file after you have renamed the file externally, the file location must be writable by the server. This may not always be the case. For example, on Linux, the server might write the error log as `/var/log/mysqld.log`, where `/var/log` is owned by `root` and not writable by `mysqld`. In this case, the log-flushing operation will fail to create a new log file.

To handle this situation, you must manually create the new log file with the proper ownershiop after renaming the original log file. For example, execute these commands as `root`:

```
shell> mv /var/log/mysqld.log /var/log/mysqld.log.old
shell> install -omysql -gmysql -m0644 /dev/null /var/log/mysqld.log
```

5.5 MySQL Server Components

MySQL Server includes a component-based infrastructure for extending server capabilities. A component provides services that are available to the server and other components. (With respect to service use, the

server is a component, equal to other components.) Components interact with each other only through the services they provide.

MySQL distributions include several components that implement server extensions:

- Components for configuring error logging. See Section 5.4.2, “The Error Log”, and Section 5.5.3, “Error Log Components”.
- A component for checking passwords. See Section 6.4.3, “The Password Validation Component”.
- A component that enables applications to add their own message events to the audit log. See Section 6.4.6, “The Audit Message Component”.

System and status variables implemented by a server component are exposed when the component is installed and have names that begin with a component-specific prefix. For example, the `log_filter_dragnet` error log filter component implements a system variable named `log_error_filter_rules`, the full name of which is `dragnet.log_error_filter_rules`. To refer to this variable, use the full name.

The following sections describe how to install and uninstall components, and how to determine at runtime which components are installed and obtain information about them.

For information about the internal implementation of components, see the MySQL Server Doxygen documentation, available at https://dev.mysql.com/doc/index-other.html. For example, if you intend to write your own components, this information is important for understanding how components work.

5.5.1 Installing and Uninstalling Components

Server components must be loaded into the server before they can be used. MySQL supports component loading at runtime.

The `INSTALL COMPONENT` and `UNINSTALL COMPONENT` SQL statements enable component loading and unloading. For example:

```
INSTALL COMPONENT 'file://component_validate_password';
UNINSTALL COMPONENT 'file://component_validate_password';
```

A loader service handles component loading and unloading, and also lists loaded components in the `component` table of the `mysql` system database that serves as a registry.

The SQL statements for component manipulation affect server operation and the `mysql.component` system table as follows:

- `INSTALL COMPONENT` loads components into the server. The components become active immediately. The loader service also registers loaded components in the `mysql.component` system table. For subsequent server restarts, the loader service loads any components listed in `mysql.component` during the startup sequence. This occurs even if the server is started with the `--skip-grant-tables` option.
- `UNINSTALL COMPONENT` deactivates components and unloads them from the server. The loader service also unregisters the components from the `mysql.component` system table so that they are no longer loaded during the startup sequence for subsequent server restarts.

Compared to the corresponding `INSTALL PLUGIN` statement for server plugins, the `INSTALL COMPONENT` statement for components offers the significant advantage that it is not necessary to know any platform-specific file name suffix for naming the component. This means that a given `INSTALL COMPONENT` statement can be executed uniformly across platforms.

5.5.2 Obtaining Server Component Information

The `component` table in the `mysql` system database contains information about currently loaded components and shows which components have been registered with `INSTALL COMPONENT`. To see which components are installed, use this statement:

```
SELECT * FROM mysql.component;
```

5.5.3 Error Log Components

This section describes the characteristics of individual error log components. For general information about configuring error logging, see Section 5.4.2, “The Error Log”.

A log component can be a filter or a sink:

- A filter processes log events, to add, remove, or modify event fields, or to delete events entirely. The resulting events pass to the next log component named in the `log_error_services` system variable value.
- A sink is a destination (writer) for log events. Typically, a sink processes log events into log messages that have a particular format and writes these messages to its associated output, such as a file or the system log.

The server executes filters and sinks in the `log_error_services` value in the order they are named. The rightmost component should therefore be a sink. If the rightmost component is a filter, any changes it has on events have no effect on output.

The following sections describe individual log components, grouped by component type:

- Error Log Filter Components
- Error Log Sink Components

Component descriptions include these types of information:

- The component name and intended purpose.
- Whether the component is built in or must be loaded. For a loadable component, the description specifies the URN to use to load and unload the component with the `INSTALL COMPONENT` and `UNINSTALL COMPONENT` statements.
- Whether the component can be listed multiple times in the `log_error_services` value.
- For a sink component, the destination to which the component writes output.

Error Log Filter Components

Error log filter components implement filtering of error log events:

- If no filter component is enabled, no filtering occurs.
- Any enabled filter component affects log events only for components listed later in the `log_error_services` value. In particular, for any log sink component listed in `log_error_services` earlier than any filter component, no log event filtering occurs.

The log_filter_internal Component

- Purpose: Implements filtering based on log event priority and error code, in combination with the `log_error_verbosity` and `log_error_suppression_list` system variables. See Section 5.4.2.4, “Priority-Based Error Log Filtering (log_filter_internal)”.
- URN: This component is built in and need not be loaded with `INSTALL COMPONENT` before use.
- Multiple uses permitted: No.

If `log_filter_internal` is disabled, `log_error_verbosity` and `log_error_suppression_list` have no effect.

The log_filter_dragnet Component

- Purpose: Implements filtering based on the rules defined by the `dragnet.log_error_filter_rules` system variable setting. See Section 5.4.2.5, “Rule-Based Error Log Filtering (log_filter_dragnet)”.
- URN: `file://component_log_filter_dragnet`
- Multiple uses permitted: No.

Error Log Sink Components

Error log sink components are writers that implement error log output. If no sink component is enabled, no log output occurs.

Some sink component descriptions refer to the default error log destination. This is the console or a file and is indicated by the fault of the `log_error` system variable, determined as described in Section 5.4.2.2, “Default Error Log Destination Configuration”.

The log_sink_internal Component

- Purpose: Implements traditional error log message output format.
- URN: This component is built in and need not be loaded with `INSTALL COMPONENT` before use.
- Multiple uses permitted: No.
- Output destination: Writes to the default error log destination.

The log_sink_json Component

- Purpose: Implements JSON-format error logging. See Section 5.4.2.6, “Error Logging in JSON Format”.
- URN: `file://component_log_sink_json`
- Multiple uses permitted: Yes.
- Output destination: The JSON log writer determines its output destination based on the default error log destination, which is given by the `log_error` system variable:
  - If `log_error` names a file, the JSON writer bases output file naming on that file name, plus a numbered *`.NN`*`.json` suffix, with *`NN`* starting at 00. For example, if `log_error` is *`file_name`*, successive instances of `log_sink_json` named in the `log_error_services` value write to *`file_name`*`.00.json`, *`file_name`*`.01.json`, and so forth.
  - If `log_error` is `stderr`, the JSON writer writes to the console. If `log_json_writer` is named multiple times in the `log_error_services` value, they all write to the console, which is likely not useful.

The log_sink_syseventlog Component

- Purpose: Implements error logging to the system log. This is the Event Log on Windows, and `syslog` on Unix and Unix-like systems. See Section 5.4.2.7, “Error Logging to the System Log”.
- URN: `file://component_log_sink_syseventlog`
- Multiple uses permitted: No.
- Output destination: Writes to the system log. Does not use the default error log destination.

The log_sink_test Component

- Purpose: Intended for internal use in writing test cases. Not intended for production use.
- URN: `file://component_log_sink_test`
- Multiple uses permitted: Yes.
- Output destination: Writes to the default error log destination.

5.6 MySQL Server Plugins

MySQL supports a plugin API that enables creation of server components. Plugins can be loaded at server startup, or loaded and unloaded at runtime without restarting the server. The components supported by this interface include, but are not limited to, storage engines, `INFORMATION_SCHEMA` tables, full-text parser plugins, and server extensions.

MySQL distributions include several plugins that implement server extensions:

- Plugins for authenticating attempts by clients to connect to MySQL Server. Plugins are available for several authentication protocols. See Section 6.2.17, “Pluggable Authentication”.
- A connection-control plugin that enables administrators to introduce an increasing delay after a certain number of consecutive failed client connection attempts. See Section 6.4.2, “The Connection-Control Plugins”.
- A password-validation plugin implements password strength policies and assesses the strength of potential passwords. See Section 6.4.3, “The Password Validation Component”.
- Semisynchronous replication plugins implement an interface to replication capabilities that permit the master to proceed as long as at least one slave has responded to each transaction. See Section 17.4.9, “Semisynchronous Replication”.
- Group Replication enables you to create a highly available distributed MySQL service across a group of MySQL server instances, with data consistency, conflict detection and resolution, and group membership services all built-in. See Chapter 18, *Group Replication*.
- MySQL Enterprise Edition includes a thread pool plugin that manages connection threads to increase server performance by efficiently managing statement execution threads for large numbers of client connections. See Section 5.6.3, “MySQL Enterprise Thread Pool”.
- MySQL Enterprise Edition includes an audit plugin for monitoring and logging of connection and query activity. See Section 6.4.5, “MySQL Enterprise Audit”.
- MySQL Enterprise Edition includes a firewall plugin that implements an application-level firewall to enable database administrators to permit or deny SQL statement execution based on matching against whitelists of accepted statement patterns. See Section 6.4.7, “MySQL Enterprise Firewall”.

- Query rewrite plugins examine statements received by MySQL Server and possibly rewrite them before the server executes them. See Section 5.6.4, “The Rewriter Query Rewrite Plugin”, and Section 5.6.5, “The ddl_rewriter Plugin”.
- Version Tokens enables creation of and synchronization around server tokens that applications can use to prevent accessing incorrect or out-of-date data. Version Tokens is based on a plugin library that implements a `version_tokens` plugin and a set of user-defined functions. See Section 5.6.6, “Version Tokens”.
- Keyring plugins provide secure storage for sensitive information. See Section 6.4.4, “The MySQL Keyring”.
- X Plugin extends MySQL Server to be able to function as a document store. Running X Plugin enables MySQL Server to communicate with clients using the X Protocol, which is designed to expose the ACID compliant storage abilities of MySQL as a document store. See Section 20.5, “X Plugin”.
- Clone permits cloning `InnoDB` data from a local or remote MySQL server instance. See Section 5.6.7, “The Clone Plugin”.
- Test framework plugins test server services. For information about these plugins, see the Plugins for Testing Plugin Services section of the MySQL Server Doxygen documentation, available at https://dev.mysql.com/doc/index-other.html.

The following sections describe how to install and uninstall plugins, and how to determine at runtime which plugins are installed and obtain information about them. For information about writing plugins, see Section 29.2, “The MySQL Plugin API”.

5.6.1 Installing and Uninstalling Plugins

Server plugins must be loaded into the server before they can be used. MySQL supports plugin loading at server startup and runtime. It is also possible to control the activation state of loaded plugins at startup, and to unload them at runtime.

While a plugin is loaded, information about it is available from the `INFORMATION_SCHEMA.PLUGINS` table and the `SHOW PLUGINS` statement. See Section 5.6.2, “Obtaining Server Plugin Information”.

- Installing Plugins
- Controlling Plugin Activation State
- Uninstalling Plugins

Installing Plugins

Before a server plugin can be used, it must be installed using one of the following methods. In the descriptions, *`plugin_name`* stands for a plugin name such as `innodb`, `csv`, or `validate_password`.

Built-in plugins:

A built-in plugin is known by the server automatically. Normally, the server enables the plugin at startup. Some built-in plugins permit this to be changed with the `--plugin_name[=activation_state]` option.

Plugins registered in the `mysql.plugin` system table:

The `plugin` table in the `mysql` system database serves as a registry of plugins (other than built-in plugins, which need not be registered). At startup, the server loads each plugin listed in the table. Normally, for a plugin loaded from the `mysql.plugin` table, the server also enables the plugin. This can be changed with the `--plugin_name[=activation_state]` option.

If the server is started with the `--skip-grant-tables` option, it does not consult the `mysql.plugin` table and does not load the plugins listed there.

Plugins named with command-line options:

A plugin located in a plugin library file can be loaded at server startup with the `--plugin-load`, `--plugin-load-add`, or `--early-plugin-load` option. Normally, for a plugin loaded at startup, the server also enables the plugin. This can be changed with the `--plugin_name[=activation_state]` option.

The `--plugin-load` and `--plugin-load-add` options load plugins after built-in plugins and storage engines have initialized during the server startup sequence. The `--early-plugin-load` option is used to load plugins that must be available prior to initialization of built-in plugins and storage engines.

The value of each plugin-loading option is a semicolon-separated list of *`name=plugin_library`* and *`plugin_library`* values. Each *`name`* is the name of a plugin to load, and *`plugin_library`* is the name of the library file that contains the plugin code. If a plugin library is named without any preceding plugin name, the server loads all plugins in the library. The server looks for plugin library files in the directory named by the `plugin_dir` system variable.

Plugin-loading options do not register any plugin in the `mysql.plugin` table. For subsequent restarts, the server loads the plugin again only if `--plugin-load`, `--plugin-load-add`, or `--early-plugin-load` is given again. That is, the option produces a one-time plugin-installation operation that persists for a single server invocation.

`--plugin-load`, `--plugin-load-add`, and `--early-plugin-load` enable plugins to be loaded even when `--skip-grant-tables` is given (which causes the server to ignore the `mysql.plugin` table). `--plugin-load`, `--plugin-load-add`, and `--early-plugin-load` also enable plugins to be loaded at startup that cannot be loaded at runtime.

The `--plugin-load-add` option complements the `--plugin-load` option:

- Each instance of `--plugin-load` resets the set of plugins to load at startup, whereas `--plugin-load-add` adds a plugin or plugins to the set of plugins to be loaded without resetting the current set. Consequently, if multiple instances of `--plugin-load` are specified, only the last one takes effect. With multiple instances of `--plugin-load-add`, all of them take effect.
- The argument format is the same as for `--plugin-load`, but multiple instances of `--plugin-load-add` can be used to avoid specifying a large set of plugins as a single long unwieldy `--plugin-load` argument.
- `--plugin-load-add` can be given in the absence of `--plugin-load`, but any instance of `--plugin-load-add` that appears before `--plugin-load` has no effect because `--plugin-load` resets the set of plugins to load.

For example, these options:

```
--plugin-load=x --plugin-load-add=y
```

are equivalent to this option:

```
--plugin-load="x;y"
```

But these options:

```
--plugin-load-add=y --plugin-load=x
```

are equivalent to this option:

```
--plugin-load=x
```

Plugins installed with the `INSTALL PLUGIN` statement:

A plugin located in a plugin library file can be loaded at runtime with the `INSTALL PLUGIN` statement. The statement also registers the plugin in the `mysql.plugin` table to cause the server to load it on subsequent restarts. For this reason, `INSTALL PLUGIN` requires the `INSERT` privilege for the `mysql.plugin` table.

The plugin library file base name depends on your platform. Common suffixes are `.so` for Unix and Unix-like systems, `.dll` for Windows.

Example: The `--plugin-load` option installs a plugin at server startup. To install a plugin named `myplugin` from a plugin library file named `somepluglib.so`, use these lines in a `my.cnf` file:

```
[mysqld]
plugin-load=myplugin=somepluglib.so
```

In this case, the plugin is not registered in `mysql.plugin`. Restarting the server without the `--plugin-load` option causes the plugin not to be loaded at startup.

Alternatively, the `INSTALL PLUGIN` statement causes the server to load the plugin code from the library file at runtime:

```
INSTALL PLUGIN myplugin SONAME 'somepluglib.so';
```

`INSTALL PLUGIN` also causes “permanent” plugin registration: The plugin is listed in the `mysql.plugin` table to ensure that the server loads it on subsequent restarts.

Many plugins can be loaded either at server startup or at runtime. However, if a plugin is designed such that it must be loaded and initialized during server startup, attempts to load it at runtime using `INSTALL PLUGIN` produce an error:

```
mysql> INSTALL PLUGIN myplugin SONAME 'somepluglib.so';
ERROR 1721 (HY000): Plugin 'myplugin' is marked as not dynamically
installable. You have to stop the server to install it.
```

In this case, you must use `--plugin-load`, `--plugin-load-add`, or `--early-plugin-load`.

If a plugin is named both using a `--plugin-load`, `--plugin-load-add`, or `--early-plugin-load` option and (as a result of an earlier `INSTALL PLUGIN` statement) in the `mysql.plugin` table, the server starts but writes these messages to the error log:

```
[ERROR] Function 'plugin_name' already exists
[Warning] Couldn't load plugin named 'plugin_name'
with soname 'plugin_object_file'.
```

Controlling Plugin Activation State

If the server knows about a plugin when it starts (for example, because the plugin is named using a `--plugin-load` option or is registered in the `mysql.plugin` table), the server loads and enables the plugin by default. It is possible to control activation state for such a plugin using a `--plugin_name[=activation_state]` startup option, where *`plugin_name`* is the name of the plugin to affect, such as `innodb`, `csv`, or `validate_password`. As with other options, dashes and underscores

are interchangeable in option names. Also, activation state values are not case-sensitive. For example, `--my_plugin=ON` and `--my-plugin=on` are equivalent.

- `--plugin_name=OFF`

  Tells the server to disable the plugin. This may not be possible for certain built-in plugins, such as `mysql_native_password`.

- `--plugin_name[=ON]`

  Tells the server to enable the plugin. (Specifying the option as `--plugin_name` without a value has the same effect.) If the plugin fails to initialize, the server runs with the plugin disabled.

- `--plugin_name=FORCE`

  Tells the server to enable the plugin, but if plugin initialization fails, the server does not start. In other words, this option forces the server to run with the plugin enabled or not at all.

- `--plugin_name=FORCE_PLUS_PERMANENT`

  Like `FORCE`, but in addition prevents the plugin from being unloaded at runtime. If a user attempts to do so with `UNINSTALL PLUGIN`, an error occurs.

Plugin activation states are visible in the `LOAD_OPTION` column of the `INFORMATION_SCHEMA.PLUGINS` table.

Suppose that `CSV`, `BLACKHOLE`, and `ARCHIVE` are built-in pluggable storage engines and that you want the server to load them at startup, subject to these conditions: The server is permitted to run if `CSV` initialization fails, must require that `BLACKHOLE` initialization succeeds, and should disable `ARCHIVE`. To accomplish that, use these lines in an option file:

```
[mysqld]
csv=ON
blackhole=FORCE
archive=OFF
```

The `--enable-plugin_name` option format is a synonym for `--plugin_name=ON`. The `--disable-plugin_name` and `--skip-plugin_name` option formats are synonyms for `--plugin_name=OFF`.

If a plugin is disabled, either explicitly with `OFF` or implicitly because it was enabled with `ON` but failed to initialize, aspects of server operation that require the plugin will change. For example, if the plugin implements a storage engine, existing tables for the storage engine become inaccessible, and attempts to create new tables for the storage engine result in tables that use the default storage engine unless the `NO_ENGINE_SUBSTITUTION` SQL mode is enabled to cause an error to occur instead.

Disabling a plugin may require adjustment to other options. For example, if you start the server using `--skip-innodb` to disable `InnoDB`, other `innodb_xxx` options likely will need to be omitted at startup. In addition, because `InnoDB` is the default storage engine, it will not start unless you specify another available storage engine with `--default_storage_engine`. You must also set `--default_tmp_storage_engine`.

Uninstalling Plugins

At runtime, the `UNINSTALL PLUGIN` statement disables and uninstalls a plugin known to the server. The statement unloads the plugin and removes it from the `mysql.plugin` system table, if it is registered there. For this reason, `UNINSTALL PLUGIN` statement requires the `DELETE` privilege for the `mysql.plugin`

table. With the plugin no longer registered in the table, the server does not load the plugin automatically for subsequent restarts.

`UNINSTALL PLUGIN` can unload a plugin regardless of whether it was loaded at runtime with `INSTALL PLUGIN` or at startup with a plugin-loading option, subject to these conditions:

- It cannot unload plugins that are built in to the server. These can be identified as those that have a library name of `NULL` in the output from `INFORMATION_SCHEMA.PLUGINS` or `SHOW PLUGINS`.
- It cannot unload plugins for which the server was started with *`--plugin_name`*`=FORCE_PLUS_PERMANENT`, which prevents plugin unloading at runtime. These can be identified from the `LOAD_OPTION` column of the `INFORMATION_SCHEMA.PLUGINS` table.

To uninstall a plugin that currently is loaded at server startup with a plugin-loading option, use this procedure.

1. Remove any options related to the plugin from the `my.cnf` file.
2. Restart the server.
3. Plugins normally are installed using either a plugin-loading option at startup or with `INSTALL PLUGIN` at runtime, but not both. However, removing options for a plugin from the `my.cnf` file may not be sufficient to uninstall it if at some point `INSTALL PLUGIN` has also been used. If the plugin still appears in the output from `INFORMATION_SCHEMA.PLUGINS` or `SHOW PLUGINS`, use `UNINSTALL PLUGIN` to remove it from the `mysql.plugin` table. Then restart the server again.

5.6.2 Obtaining Server Plugin Information

There are several ways to determine which plugins are installed in the server:

- The `INFORMATION_SCHEMA.PLUGINS` table contains a row for each loaded plugin. Any that have a `PLUGIN_LIBRARY` value of `NULL` are built in and cannot be unloaded.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.PLUGINS\G
*************************** 1. row ***************************
           PLUGIN_NAME: binlog
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 50158.0
        PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
         PLUGIN_AUTHOR: MySQL AB
    PLUGIN_DESCRIPTION: This is a pseudo storage engine to represent the binlog in a transaction
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: FORCE
...
*************************** 10. row ***************************
           PLUGIN_NAME: InnoDB
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 50158.0
        PLUGIN_LIBRARY: ha_innodb_plugin.so
PLUGIN_LIBRARY_VERSION: 1.0
         PLUGIN_AUTHOR: Innobase Oy
    PLUGIN_DESCRIPTION: Supports transactions, row-level locking,
                        and foreign keys
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: ON
...
```

- The `SHOW PLUGINS` statement displays a row for each loaded plugin. Any that have a `Library` value of `NULL` are built in and cannot be unloaded.

```
mysql> SHOW PLUGINS\G
*************************** 1. row ***************************
   Name: binlog
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
...
*************************** 10. row ***************************
   Name: InnoDB
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: ha_innodb_plugin.so
License: GPL
...
```

- The `mysql.plugin` table shows which plugins have been registered with `INSTALL PLUGIN`. The table contains only plugin names and library file names, so it does not provide as much information as the `PLUGINS` table or the `SHOW PLUGINS` statement.

5.6.3 MySQL Enterprise Thread Pool

Note

MySQL Enterprise Thread Pool is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, https://www.mysql.com/products/.

MySQL Enterprise Edition includes MySQL Enterprise Thread Pool, implemented using a server plugin. The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. The thread pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The plugin implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections.

The thread pool addresses several problems of the model that uses one thread per connection:

- Too many thread stacks make CPU caches almost useless in highly parallel execution workloads. The thread pool promotes thread stack reuse to minimize the CPU cache footprint.
- With too many threads executing in parallel, context switching overhead is high. This also presents a challenging task to the operating system scheduler. The thread pool controls the number of active threads to keep the parallelism within the MySQL server at a level that it can handle and that is appropriate for the server host on which MySQL is executing.
- Too many transactions executing in parallel increases resource contention. In `InnoDB`, this increases the time spent holding central mutexes. The thread pool controls when transactions start to ensure that not too many execute in parallel.

Additional Resources

Section A.15, “MySQL 8.0 FAQ: MySQL Enterprise Thread Pool”

5.6.3.1 Thread Pool Components

The thread pool feature comprises these components:

- A plugin library file implements a plugin for the thread pool code as well as several associated monitoring tables that provide information about thread pool operation:

  - As of MySQL 8.0.14, the monitoring tables are Performance Schema tables; see Section 26.12.15, “Performance Schema Thread Pool Tables”.

  - Prior to MySQL 8.0.14, the monitoring tables are `INFORMATION_SCHEMA` tables; see Section 25.40, “INFORMATION_SCHEMA Thread Pool Tables”.

    The `INFORMATION_SCHEMA` tables now are deprecated and will be removed in a future MySQL version. Applications should transition away from the old tables to the new tables. For example, if an application uses this query:

    ```
    SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_STATE;
    ```

    The application should use this query instead:

    ```
    SELECT * FROM performance_schema.tp_thread_state;
    ```

    Note

    If you do not load all the monitoring tables, some or all MySQL Enterprise Monitor thread pool graphs will be empty.

  For a detailed description of how the thread pool works, see Section 5.6.3.3, “Thread Pool Operation”.

- Several system variables are related to the thread pool. The `thread_handling` system variable has a value of `loaded-dynamically` when the server successfully loads the thread pool plugin.

  The other related variables are implemented by the thread pool plugin; they are not available unless it is enabled:

  - `thread_pool_algorithm`: The concurrency algorithm to use for scheduling.
  - `thread_pool_high_priority_connection`: How to schedule statement execution for a session.
  - `thread_pool_prio_kickup_timer`: How long before the thread pool moves a statement awaiting execution from the low-priority queue to the high-priority queue.
  - `thread_pool_max_active_query_threads` How many active threads per group to permit.
  - `thread_pool_max_unused_threads`: How many sleeping threads to permit.
  - `thread_pool_size`: The number of thread groups in the thread pool. This is the most important parameter controlling thread pool performance.
  - `thread_pool_stall_limit`: The time before an executing statement is considered to be stalled.

  If any variable implemented by the plugin is set to an illegal value at startup, plugin initialization fails and the plugin does not load.

  For information about setting thread pool parameters, see Section 5.6.3.4, “Thread Pool Tuning”.

- The Performance Schema has instruments that expose information about the thread pool and may be used to investigate operational performance. To identify them, use this query:

  ```
  ```

```
SELECT * FROM performance_schema.setup_instruments
WHERE NAME LIKE '%thread_pool%';
```

For more information, see Chapter 26, *MySQL Performance Schema*.

5.6.3.2 Thread Pool Installation

This section describes how to install MySQL Enterprise Thread Pool. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `thread_pool`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

- Thread Pool Installation as of MySQL 8.0.14
- Thread Pool Installation Prior to MySQL 8.0.14

Thread Pool Installation as of MySQL 8.0.14

In MySQL 8.0.14 and higher, the thread pool monitoring tables are Performance Schema tables that are loaded and unloaded along with the thread pool plugin. The `INFORMATION_SCHEMA` versions of the tables are deprecated but still available; they are installed per the instructions in Thread Pool Installation Prior to MySQL 8.0.14.

To enable thread pool capability, load the plugin by starting the server with the `--plugin-load-add` option. To do this, put these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=thread_pool.so
```

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'thread%';
+-----------------------+---------------+
| PLUGIN_NAME           | PLUGIN_STATUS |
+-----------------------+---------------+
| thread_pool           | ACTIVE        |
+-----------------------+---------------+
```

To verify that the Performance Schema monitoring tables are available, examine the `INFORMATION_SCHEMA.TABLES` table or use the `SHOW TABLES` statement. For example:

```
mysql> SELECT TABLE_NAME
       FROM INFORMATION_SCHEMA.TABLES
       WHERE TABLE_SCHEMA = 'performance_schema'
       AND TABLE_NAME LIKE 'tp%';
+-----------------------+
| TABLE_NAME            |
+-----------------------+
| tp_thread_group_state |
| tp_thread_group_stats |
```

```
| tp_thread_state       |
+-----------------------+
```

If the server loads the thread pool plugin successfully, it sets the `thread_handling` system variable to `loaded-dynamically`.

If the plugin failed to initialize, check the server error log for diagnostic messages.

Thread Pool Installation Prior to MySQL 8.0.14

Prior to MySQL 8.0.14, the thread pool monitoring tables are plugins separate from the thread pool plugin and can be installed separately.

To enable thread pool capability, load the plugins to be used by starting the server with the `--plugin-load-add` option. For example, if you name only the plugin library file, the server loads all plugins that it contains (that is, the thread pool plugin and all the `INFORMATION_SCHEMA` tables). To do this, put these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=thread_pool.so
```

That is equivalent to loading all thread pool plugins by naming them individually:

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
plugin-load-add=tp_thread_state=thread_pool.so
plugin-load-add=tp_thread_group_state=thread_pool.so
plugin-load-add=tp_thread_group_stats=thread_pool.so
```

If desired, you can load individual plugins from the library file. To load the thread pool plugin but not the `INFORMATION_SCHEMA` tables, use an option like this:

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
```

To load the thread pool plugin and only the `TP_THREAD_STATE` `INFORMATION_SCHEMA` table, use options like this:

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
plugin-load-add=tp_thread_state=thread_pool.so
```

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'thread%' OR PLUGIN_NAME LIKE 'tp%';
+----------------------+---------------+
| PLUGIN_NAME          | PLUGIN_STATUS |
+----------------------+---------------+
| thread_pool          | ACTIVE        |
| TP_THREAD_STATE      | ACTIVE        |
| TP_THREAD_GROUP_STATE | ACTIVE        |
| TP_THREAD_GROUP_STATS | ACTIVE        |
+----------------------+---------------+
```

If the server loads the thread pool plugin successfully, it sets the `thread_handling` system variable to `loaded-dynamically`.

If a plugin failed to initialize, check the server error log for diagnostic messages.

5.6.3.3 Thread Pool Operation

The thread pool consists of a number of thread groups, each of which manages a set of client connections. As connections are established, the thread pool assigns them to thread groups in round-robin fashion.

The number of thread groups is configurable using the `thread_pool_size` system variable. The default number of groups is 16. For guidelines on setting this variable, see Section 5.6.3.4, “Thread Pool Tuning”.

The maximum number of threads per group is 4096 (or 4095 on some systems where one thread is used internally).

The thread pool separates connections and threads, so there is no fixed relationship between connections and the threads that execute statements received from those connections. This differs from the default thread-handling model that associates one thread with one connection such that a given thread executes all statements from its connection.

By default, the thread pool tries to ensure a maximum of one thread executing in each group at any time, but sometimes permits more threads to execute temporarily for best performance. The algorithm works in the following manner:

- Each thread group has a listener thread that listens for incoming statements from the connections assigned to the group. When a statement arrives, the thread group either begins executing it immediately or queues it for later execution:
  - Immediate execution occurs if the statement is the only one received and no statements are queued or currently executing.
  - Queuing occurs if the statement cannot begin executing immediately.
- If immediate execution occurs, execution is performed by the listener thread. (This means that temporarily no thread in the group is listening.) If the statement finishes quickly, the executing thread returns to listening for statements. Otherwise, the thread pool considers the statement stalled and starts another thread as a listener thread (creating it if necessary). To ensure that no thread group becomes blocked by stalled statements, the thread pool has a background thread that regularly monitors thread group states.

  By using the listening thread to execute a statement that can begin immediately, there is no need to create an additional thread if the statement finishes quickly. This ensures the most efficient execution possible in the case of a low number of concurrent threads.

  When the thread pool plugin starts, it creates one thread per group (the listener thread), plus the background thread. Additional threads are created as necessary to execute statements.
- The value of the `thread_pool_stall_limit` system variable determines the meaning of “finishes quickly” in the previous item. The default time before threads are considered stalled is 60ms but can be set to a maximum of 6s. This parameter is configurable to enable you to strike a balance appropriate for the server work load. Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that include long-running statements, to avoid starting too many new statements while the current ones execute.
- If `thread_pool_max_active_query_threads` is 0, the default algorithm applies as just described for determining the maximum number of active threads per group. The default algorithm takes stalled threads into account and may temporarily permit more active threads. If `thread_pool_max_active_query_threads` is greater than 0, it places a limit on the number of active threads per group.

- The thread pool focuses on limiting the number of concurrent short-running statements. Before an executing statement reaches the stall time, it prevents other statements from beginning to execute. If the statement executes past the stall time, it is permitted to continue but no longer prevents other statements from starting. In this way, the thread pool tries to ensure that in each thread group there is never more than one short-running statement, although there might be multiple long-running statements. It is undesirable to let long-running statements prevent other statements from executing because there is no limit on the amount of waiting that might be necessary. For example, on a replication master, a thread that is sending binary log events to a slave effectively runs forever.

- A statement becomes blocked if it encounters a disk I/O operation or a user level lock (row lock or table lock). The block would cause the thread group to become unused, so there are callbacks to the thread pool to ensure that the thread pool can immediately start a new thread in this group to execute another statement. When a blocked thread returns, the thread pool permits it to restart immediately.

- There are two queues, a high-priority queue and a low-priority queue. The first statement in a transaction goes to the low-priority queue. Any following statements for the transaction go to the high-priority queue if the transaction is ongoing (statements for it have begun executing), or to the low-priority queue otherwise. Queue assignment can be affected by enabling the `thread_pool_high_priority_connection` system variable, which causes all queued statements for a session to go into the high-priority queue.

  Statements for a nontransactional storage engine, or a transactional engine if `autocommit` is enabled, are treated as low-priority statements because in this case each statement is a transaction. Thus, given a mix of statements for `InnoDB` and `MyISAM` tables, the thread pool prioritizes those for `InnoDB` over those for `MyISAM` unless `autocommit` is enabled. With `autocommit` enabled, all statements will be low priority.

- When the thread group selects a queued statement for execution, it first looks in the high-priority queue, then in the low-priority queue. If a statement is found, it is removed from its queue and begins to execute.

- If a statement stays in the low-priority queue too long, the thread pool moves to the high-priority queue. The value of the `thread_pool_prio_kickup_timer` system variable controls the time before movement. For each thread group, a maximum of one statement per 10ms or 100 per second will be moved from the low-priority queue to the high-priority queue.

- The thread pool reuses the most active threads to obtain a much better use of CPU caches. This is a small adjustment that has a great impact on performance.

- While a thread executes a statement from a user connection, Performance Schema instrumentation accounts thread activity to the user connection. Otherwise, Performance Schema accounts activity to the thread pool.

Here are examples of conditions under which a thread group might have multiple threads started to execute statements:

- One thread begins executing a statement, but runs long enough to be considered stalled. The thread group permits another thread to begin executing another statement even through the first thread is still executing.

- One thread begins executing a statement, then becomes blocked and reports this back to the thread pool. The thread group permits another thread to begin executing another statement.

- One thread begins executing a statement, becomes blocked, but does not report back that it is blocked because the block does not occur in code that has been instrumented with thread pool callbacks. In this case, the thread appears to the thread group to be still running. If the block lasts long enough for

the statement to be considered stalled, the group permits another thread to begin executing another statement.

The thread pool is designed to be scalable across an increasing number of connections. It is also designed to avoid deadlocks that can arise from limiting the number of actively executing statements. It is important that threads that do not report back to the thread pool do not prevent other statements from executing and thus cause the thread pool to become deadlocked. Examples of such statements follow:

- Long-running statements. These would lead to all resources used by only a few statements and they could prevent all others from accessing the server.
- Binary log dump threads that read the binary log and send it to slaves. This is a kind of long-running "statement" that runs for a very long time, and that should not prevent other statements from executing.
- Statements blocked on a row lock, table lock, sleep, or any other blocking activity that has not been reported back to the thread pool by MySQL Server or a storage engine.

In each case, to prevent deadlock, the statement is moved to the stalled category when it does not complete quickly, so that the thread group can permit another statement to begin executing. With this design, when a thread executes or becomes blocked for an extended time, the thread pool moves the thread to the stalled category and for the rest of the statement's execution, it does not prevent other statements from executing.

The maximum number of threads that can occur is the sum of `max_connections` and `thread_pool_size`. This can happen in a situation where all connections are in execution mode and an extra thread is created per group to listen for more statements. This is not necessarily a state that happens often, but it is theoretically possible.

5.6.3.4 Thread Pool Tuning

This section provides guidelines on setting thread pool system variables for best performance, measured using a metric such as transactions per second.

`thread_pool_size` is the most important parameter controlling thread pool performance. It can be set only at server startup. Our experience in testing the thread pool indicates the following:

- If the primary storage engine is `InnoDB`, the optimal `thread_pool_size` setting is likely to be between 16 and 36, with the most common optimal values tending to be from 24 to 36. We have not seen any situation where the setting has been optimal beyond 36. There may be special cases where a value smaller than 16 is optimal.

  For workloads such as DBT2 and Sysbench, the optimum for `InnoDB` seems to be usually around 36. For very write-intensive workloads, the optimal setting can sometimes be lower.

- If the primary storage engine is `MyISAM`, the `thread_pool_size` setting should be fairly low. Optimal performance is often seen with values from 4 to 8. Higher values tend to have a slightly negative but not dramatic impact on performance.

Another system variable, `thread_pool_stall_limit`, is important for handling of blocked and long-running statements. If all calls that block the MySQL Server are reported to the thread pool, it would always know when execution threads are blocked. However, this may not always be true. For example, blocks could occur in code that has not been instrumented with thread pool callbacks. For such cases, the thread pool must be able to identify threads that appear to be blocked. This is done by means of a timeout, the length of which can be tuned using the `thread_pool_stall_limit` system variable. This parameter ensures that the server does not become completely blocked. The value of `thread_pool_stall_limit` has an upper limit of 6 seconds to prevent the risk of a deadlocked server.

`thread_pool_stall_limit` also enables the thread pool to handle long-running statements. If a long-running statement was permitted to block a thread group, all other connections assigned to the group would be blocked and unable to start execution until the long-running statement completed. In the worst case, this could take hours or even days.

The value of `thread_pool_stall_limit` should be chosen such that statements that execute longer than its value are considered stalled. Stalled statements generate a lot of extra overhead since they involve extra context switches and in some cases even extra thread creations. On the other hand, setting the `thread_pool_stall_limit` parameter too high means that long-running statements will block a number of short-running statements for longer than necessary. Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that include long-running statements, to avoid starting too many new statements while the current ones execute.

Suppose a server executes a workload where 99.9% of the statements complete within 100ms even when the server is loaded, and the remaining statements take between 100ms and 2 hours fairly evenly spread. In this case, it would make sense to set `thread_pool_stall_limit` to 10 (meaning 100ms). The default value of 60ms is suitable for servers that primarily execute very simple statements.

The `thread_pool_stall_limit` parameter can be changed at runtime to enable you to strike a balance appropriate for the server work load. Assuming that the `tp_thread_group_stats` table is enabled, you can use the following query to determine the fraction of executed statements that stalled:

```
SELECT SUM(STALLED_QUERIES_EXECUTED) / SUM(QUERIES_EXECUTED)
FROM performance_schema.tp_thread_group_stats;
```

This number should be as low as possible. To decrease the likelihood of statements stalling, increase the value of `thread_pool_stall_limit`.

When a statement arrives, what is the maximum time it can be delayed before it actually starts executing? Suppose that the following conditions apply:

- There are 200 statements queued in the low-priority queue.
- There are 10 statements queued in the high-priority queue.
- `thread_pool_prio_kickup_timer` is set to 10000 (10 seconds).
- `thread_pool_stall_limit` is set to 100 (1 second).

In the worst case, the 10 high-priority statements represent 10 transactions that continue executing for a long time. Thus, in the worst case, no statements will be moved to the high-priority queue because it will always already contain statements awaiting execution. After 10 seconds, the new statement is eligible to be moved to the high-priority queue. However, before it can be moved, all the statements before it must be moved as well. This could take another 2 seconds because a maximum of 100 statements per second are moved to the high-priority queue. Now when the statement reaches the high-priority queue, there could potentially be many long-running statements ahead of it. In the worst case, every one of those will become stalled and it will take 1 second for each statement before the next statement is retrieved from the high-priority queue. Thus, in this scenario, it will take 222 seconds before the new statement starts executing.

This example shows a worst case for an application. How to handle it depends on the application. If the application has high requirements for the response time, it should most likely throttle users at a higher level itself. Otherwise, it can use the thread pool configuration parameters to set some kind of a maximum waiting time.

5.6.4 The Rewriter Query Rewrite Plugin

MySQL supports query rewrite plugins that can examine and possibly modify SQL statements received by the server before the server executes them. See Query Rewrite Plugins.

MySQL distributions include a postparse query rewrite plugin named `Rewriter` and scripts for installing the plugin and its associated components. These components work together to provide statement-rewriting capability:

- A server-side plugin named `Rewriter` examines statements and may rewrite them, based on its in-memory cache of rewrite rules.
- These statements are subject to rewriting:
  - As of MySQL 8.0.12: `SELECT`, `INSERT`, `REPLACE`, `UPDATE`, and `DELETE`.
  - Prior to MySQL 8.0.12: `SELECT` only.

  Standalone statements and prepared statements are subject to rewriting. Statements occurring within view definitions or stored programs are not subject to rewriting.
- The `Rewriter` plugin uses a database named `query_rewrite` containing a table named `rewrite_rules`. The table provides persistent storage for the rules that the plugin uses to decide whether to rewrite statements. Users communicate with the plugin by modifying the set of rules stored in this table. The plugin communicates with users by setting the `message` column of table rows.
- The `query_rewrite` database contains a stored procedure named `flush_rewrite_rules()` that loads the contents of the rules table into the plugin.
- A user-defined function named `load_rewrite_rules()` is used by the `flush_rewrite_rules()` stored procedure.
- The `Rewriter` plugin exposes system variables that enable plugin configuration and status variables that provide runtime operational information.

The following sections describe how to install and use the `Rewriter` plugin, and provide reference information for its associated components.

5.6.4.1 Installing or Uninstalling the Rewriter Query Rewrite Plugin

Note

If installed, the `Rewriter` plugin involves some overhead even when disabled. To avoid this overhead, do not install the plugin unless you plan to use it.

To install or uninstall the `Rewriter` query rewrite plugin, choose the appropriate script located in the `share` directory of your MySQL installation:

- `install_rewriter.sql`: Choose this script to install the `Rewriter` plugin and its associated components.
- `uninstall_rewriter.sql`: Choose this script to uninstall the `Rewriter` plugin and its associated components.

Run the chosen script as follows:

```
shell> mysql -u root -p < install_rewriter.sql
Enter password: (enter root password here)
```

The example here uses the `install_rewriter.sql` installation script. Substitute `uninstall_rewriter.sql` if you are uninstalling the plugin.

Running an installation script should install and enable the plugin. To verify that, connect to the server and execute this statement:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'rewriter_enabled';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| rewriter_enabled | ON    |
+------------------+-------+
```

For usage instructions, see Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”. For reference information, see Section 5.6.4.3, “Rewriter Query Rewrite Plugin Reference”.

5.6.4.2 Using the Rewriter Query Rewrite Plugin

To enable or disable the plugin, enable or disable the `rewriter_enabled` system variable. By default, the `Rewriter` plugin is enabled when you install it (see Section 5.6.4.1, “Installing or Uninstalling the Rewriter Query Rewrite Plugin”). To set the initial plugin state explicitly, you can set the variable at server startup. For example, to enable the plugin in an option file, use these lines:

```
[mysqld]
rewriter_enabled=ON
```

It is also possible to enable or disable the plugin at runtime:

```
SET GLOBAL rewriter_enabled = ON;
SET GLOBAL rewriter_enabled = OFF;
```

Assuming that the `Rewriter` plugin is enabled, it examines and possibly modifies each rewritable statement received by the server. The plugin determines whether to rewrite statements based on its in-memory cache of rewriting rules, which are loaded from the `rewrite_rules` table in the `query_rewrite` database.

These statements are subject to rewriting:

- As of MySQL 8.0.12: `SELECT`, `INSERT`, `REPLACE`, `UPDATE`, and `DELETE`.
- Prior to MySQL 8.0.12: `SELECT` only.

Standalone statements and prepared statements are subject to rewriting. Statements occurring within view definitions or stored programs are not subject to rewriting.

- Adding Rewrite Rules
- How Statement Matching Works
- Rewriting Prepared Statements
- Rewriter Plugin Operational Information
- Rewriter Plugin Use of Character Sets

Adding Rewrite Rules

To add rules for the `Rewriter` plugin, add rows to the `rewrite_rules` table, then invoke the `flush_rewrite_rules()` stored procedure to load the rules from the table into the plugin. The following example creates a simple rule to match statements that select a single literal value:

```
INSERT INTO query_rewrite.rewrite_rules (pattern, replacement)
VALUES('SELECT ?', 'SELECT ? + 1');
```

The resulting table contents look like this:

```
mysql> SELECT * FROM query_rewrite.rewrite_rules\G
*************************** 1. row ***************************
                id: 1
           pattern: SELECT ?
  pattern_database: NULL
       replacement: SELECT ? + 1
           enabled: YES
           message: NULL
    pattern_digest: NULL
normalized_pattern: NULL
```

The rule specifies a pattern template indicating which `SELECT` statements to match, and a replacement template indicating how to rewrite matching statements. However, adding the rule to the `rewrite_rules` table is not sufficient to cause the `Rewriter` plugin to use the rule. You must invoke `flush_rewrite_rules()` to load the table contents into the plugin in-memory cache:

```
mysql> CALL query_rewrite.flush_rewrite_rules();
```

Tip

If your rewrite rules seem not to be working properly, make sure that you have reloaded the rules table by calling `flush_rewrite_rules()`.

When the plugin reads each rule from the rules table, it computes a normalized (statement digest) form from the pattern and a digest hash value, and uses them to update the `normalized_pattern` and `pattern_digest` columns:

```
mysql> SELECT * FROM query_rewrite.rewrite_rules\G
*************************** 1. row ***************************
                id: 1
           pattern: SELECT ?
  pattern_database: NULL
       replacement: SELECT ? + 1
           enabled: YES
           message: NULL
    pattern_digest: d1b44b0c19af710b5a679907e284acd2ddc285201794bc69a2389d77baedddae
normalized_pattern: select ?
```

For information about statement digesting, normalized statements, and digest hash values, see Section 26.10, “Performance Schema Statement Digests and Sampling”.

If a rule cannot be loaded due to some error, calling `flush_rewrite_rules()` produces an error:

```
mysql> CALL query_rewrite.flush_rewrite_rules();
ERROR 1644 (45000): Loading of some rule(s) failed.
```

When this occurs, the plugin writes an error message to the `message` column of the rule row to communicate the problem. Check the `rewrite_rules` table for rows with non-`NULL` `message` column values to see what problems exist.

Patterns use the same syntax as prepared statements (see Section 13.5.1, “PREPARE Syntax”). Within a pattern template, `?` characters act as parameter markers that match data values. Parameter markers

can be used only where data values should appear, not for SQL keywords, identifiers, and so forth. The ? characters should not be enclosed within quotation marks.

Like the pattern, the replacement can contain ? characters. For a statement that matches a pattern template, the plugin rewrites it, replacing ? parameter markers in the replacement using data values matched by the corresponding markers in the pattern. The result is a complete statement string. The plugin asks the server to parse it, and returns the result to the server as the representation of the rewritten statement.

After adding and loading the rule, check whether rewriting occurs according to whether statements match the rule pattern:

```
mysql> SELECT PI();
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
1 row in set (0.01 sec)

mysql> SELECT 10;
+--------+
| 10 + 1 |
+--------+
|     11 |
+--------+
1 row in set, 1 warning (0.00 sec)
```

No rewriting occurs for the first `SELECT` statement, but does for the second. The second statement illustrates that when the `Rewriter` plugin rewrites a statement, it produces a warning message. To view the message, use `SHOW WARNINGS`:

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1105
Message: Query 'SELECT 10' rewritten to 'SELECT 10 + 1' by a query rewrite plugin
```

A statement need not be rewritten to a statement of the same type. The following example loads a rule that rewrites `DELETE` statements to `UPDATE` statements:

```
INSERT INTO query_rewrite.rewrite_rules (pattern, replacement)
VALUES('DELETE FROM db1.t1 WHERE col = ?',
       'UPDATE db1.t1 SET col = NULL WHERE col = ?');
CALL query_rewrite.flush_rewrite_rules();
```

To enable or disable an existing rule, modify its `enabled` column and reload the table into the plugin. To disable rule 1:

```
UPDATE query_rewrite.rewrite_rules SET enabled = 'NO' WHERE id = 1;
CALL query_rewrite.flush_rewrite_rules();
```

This enables you to deactivate a rule without removing it from the table.

To re-enable rule 1:

```
UPDATE query_rewrite.rewrite_rules SET enabled = 'YES' WHERE id = 1;
CALL query_rewrite.flush_rewrite_rules();
```

The `rewrite_rules` table contains a `pattern_database` column that `Rewriter` uses for matching table names that are not qualified with a database name:

- Qualified table names in statements match qualified names in the pattern if corresponding database and table names are identical.
- Unqualified table names in statements match unqualified names in the pattern only if the default database is the same as `pattern_database` and the table names are identical.

Suppose that a table named `appdb.users` has a column named `id` and that applications are expected to select rows from the table using a query of one of these forms, where the second can be used only if `appdb` is the default database:

```
SELECT * FROM users WHERE appdb.id = id_value;
SELECT * FROM users WHERE id = id_value;
```

Suppose also that the `id` column is renamed to `user_id` (perhaps the table must be modified to add another type of ID and it is necessary to indicate more specifically what type of ID the `id` column represents).

The change means that applications must refer to `user_id` rather than `id` in the `WHERE` clause. But if there are old applications that cannot be written to change the `SELECT` queries they generate, they will no longer work properly. The `Rewriter` plugin can solve this problem. To match and rewrite statements whether or not they qualify the table name, add the following two rules and reload the rules table:

```
INSERT INTO query_rewrite.rewrite_rules
    (pattern, replacement) VALUES(
    'SELECT * FROM appdb.users WHERE id = ?',
    'SELECT * FROM appdb.users WHERE user_id = ?'
    );
INSERT INTO query_rewrite.rewrite_rules
    (pattern, replacement, pattern_database) VALUES(
    'SELECT * FROM users WHERE id = ?',
    'SELECT * FROM users WHERE user_id = ?',
    'appdb'
    );
CALL query_rewrite.flush_rewrite_rules();
```

`Rewriter` uses the first rule to match statements that use the qualified table name. It uses the second to match statements that used the unqualified name, but only if the default database is `appdb` (the value in `pattern_database`).

How Statement Matching Works

The `Rewriter` plugin uses statement digests and digest hash values to match incoming statements against rewrite rules in stages. The `max_digest_length` system variable determines the size of the buffer used for computing statement digests. Larger values enable computation of digests that distinguish longer statements. Smaller values use less memory but increase the likelihood of longer statements colliding with the same digest value.

The plugin matches each statement to the rewrite rules as follows:

1. Compute the statement digest hash value and compare it to the rule digest hash values. This is subject to false positives, but serves as a quick rejection test.
2. If the statement digest hash value matches any pattern digest hash values, match the normalized (statement digest) form of the statement to the normalized form of the matching rule patterns.

3. If the normalized statement matches a rule, compare the literal values in the statement and the pattern. A `?` character in the pattern matches any literal value in the statement. If the statement prepares a statement, `?` in the pattern also matches `?` in the statement. Otherwise, corresponding literals must be the same.

If multiple rules match a statement, it is nondeterministic which one the plugin uses to rewrite the statement.

If a pattern contains more markers than the replacement, the plugin discards excess data values. If a pattern contains fewer markers than the replacement, it is an error. The plugin notices this when the rules table is loaded, writes an error message to the `message` column of the rule row to communicate the problem, and sets the `Rewriter_reload_error` status variable to `ON`.

Rewriting Prepared Statements

Prepared statements are rewritten at parse time (that is, when they are prepared), not when they are executed later.

Prepared statements differ from nonprepared statements in that they may contain `?` characters as parameter markers. To match a `?` in a prepared statement, a `Rewriter` pattern must contain `?` in the same location. Suppose that a rewrite rule has this pattern:

```
SELECT ?, 3
```

The following table shows several prepared `SELECT` statements and whether the rule pattern matches them.

Prepared Statement	Whether Pattern Matches Statement
`PREPARE s AS 'SELECT 3, 3'`	Yes
`PREPARE s AS 'SELECT ?, 3'`	Yes
`PREPARE s AS 'SELECT 3, ?'`	No
`PREPARE s AS 'SELECT ?, ?'`	No

Rewriter Plugin Operational Information

The `Rewriter` plugin makes information available about its operation by means of several status variables:

```
mysql> SHOW GLOBAL STATUS LIKE 'Rewriter%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| Rewriter_number_loaded_rules      | 1     |
| Rewriter_number_reloads           | 5     |
| Rewriter_number_rewritten_queries | 1     |
| Rewriter_reload_error             | ON    |
+-----------------------------------+-------+
```

For descriptions of these variables, see Rewriter Query Rewrite Plugin Status Variables.

When you load the rules table by calling the `flush_rewrite_rules()` stored procedure, if an error occurs for some rule, the `CALL` statement produces an error, and the plugin sets the `Rewriter_reload_error` status variable to `ON`:

```
mysql> CALL query_rewrite.flush_rewrite_rules();
ERROR 1644 (45000): Loading of some rule(s) failed.

mysql> SHOW GLOBAL STATUS LIKE 'Rewriter_reload_error';
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Rewriter_reload_error | ON    |
+-----------------------+-------+
```

In this case, check the `rewrite_rules` table for rows with non-`NULL` `message` column values to see what problems exist.

Rewriter Plugin Use of Character Sets

When the `rewrite_rules` table is loaded into the `Rewriter` plugin, the plugin interprets statements using the current global value of the `character_set_client` system variable. If the global `character_set_client` value is changed subsequently, the rules table must be reloaded.

A client must have a session `character_set_client` value identical to what the global value was when the rules table was loaded or rule matching will not work for that client.

5.6.4.3 Rewriter Query Rewrite Plugin Reference

The following discussion serves as a reference to these components associated with the `Rewriter` query rewrite plugin:

- The `Rewriter` rules table in the `query_rewrite` database
- `Rewriter` procedures and functions
- `Rewriter` system and status variables

Rewriter Query Rewrite Plugin Rules Table

The `rewrite_rules` table in the `query_rewrite` database provides persistent storage for the rules that the `Rewriter` plugin uses to decide whether to rewrite statements.

Users communicate with the plugin by modifying the set of rules stored in this table. The plugin communicates information to users by setting the table's `message` column.

Note

The rules table is loaded into the plugin by the `flush_rewrite_rules` stored procedure. Unless that procedure has been called following the most recent table modification, the table contents do not necessarily correspond to the set of rules the plugin is using.

The `rewrite_rules` table has these columns:

- `id`

  The rule ID. This column is the table primary key. You can use the ID to uniquely identify any rule.

- `pattern`

  The template that indicates the pattern for statements that the rule matches. Use `?` to represent parameter markers that match data values.

- `pattern_database`

  The database used to match unqualified table names in statements. Qualified table names in statements match qualified names in the pattern if corresponding database and table names are identical. Unqualified table names in statements match unqualified names in the pattern only if the default database is the same as `pattern_database` and the table names are identical.

- `replacement`

  The template that indicates how to rewrite statements matching the `pattern` column value. Use `?` to represent parameter markers that match data values. In rewritten statements, the plugin replaces `?` parameter markers in `replacement` using data values matched by the corresponding markers in `pattern`.

- `enabled`

  Whether the rule is enabled. Load operations (performed by invoking the `flush_rewrite_rules()` stored procedure) load the rule from the table into the `Rewriter` in-memory cache only if this column is `YES`.

  This column makes it possible to deactivate a rule without removing it: Set the column to a value other than `YES` and reload the table into the plugin.

- `message`

  The plugin uses this column for communicating with users. If no error occurs when the rules table is loaded into memory, the plugin sets the `message` column to `NULL`. A non-`NULL` value indicates an error and the column contents are the error message. Errors can occur under these circumstances:

  - Either the pattern or the replacement is an incorrect SQL statement that produces syntax errors.
  - The replacement contains more `?` parameter markers than the pattern.

  If a load error occurs, the plugin also sets the `Rewriter_reload_error` status variable to `ON`.

- `pattern_digest`

  This column is used for debugging and diagnostics. If the column exists when the rules table is loaded into memory, the plugin updates it with the pattern digest. This column may be useful if you are trying to determine why some statement fails to be rewritten.

- `normalized_pattern`

  This column is used for debugging and diagnostics. If the column exists when the rules table is loaded into memory, the plugin updates it with the normalized form of the pattern. This column may be useful if you are trying to determine why some statement fails to be rewritten.

Rewriter Query Rewrite Plugin Procedures and Functions

`Rewriter` plugin operation uses a stored procedure that loads the rules table into its in-memory cache, and a helper user-defined function (UDF). Under normal operation, users invoke only the stored procedure. The UDF is intended to be invoked by the stored procedure, not directly by users.

- `flush_rewrite_rules()`

  This stored procedure uses the `load_rewrite_rules()` UDF to load the contents of the `rewrite_rules` table into the `Rewriter` in-memory cache.

Calling `flush_rewrite_rules()` implies `COMMIT`.

Invoke this procedure after you modify the rules table to cause the plugin to update its cache from the new table contents. If any errors occur, the plugin sets the `message` column for the appropriate rule rows in the table and sets the `Rewriter_reload_error` status variable to `ON`.

- `load_rewrite_rules()`

  This UDF is a helper routine used by the `flush_rewrite_rules()` stored procedure.

Rewriter Query Rewrite Plugin System Variables

The `Rewriter` query rewrite plugin supports the following system variables. These variables are available only if the plugin is installed (see Section 5.6.4.1, “Installing or Uninstalling the Rewriter Query Rewrite Plugin”).

- `rewriter_enabled`

Property	Value
System Variable	`rewriter_enabled`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether the `Rewriter` query rewrite plugin is enabled.

- `rewriter_verbose`

Property	Value
System Variable	`rewriter_verbose`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

For internal use.

Rewriter Query Rewrite Plugin Status Variables

The `Rewriter` query rewrite plugin supports the following status variables. These variables are available only if the plugin is installed (see Section 5.6.4.1, “Installing or Uninstalling the Rewriter Query Rewrite Plugin”).

- `Rewriter_number_loaded_rules`

  The number of rewrite plugin rewrite rules successfully loaded from the `rewrite_rules` table into memory for use by the `Rewriter` plugin.

- `Rewriter_number_reloads`

The number of times the `rewrite_rules` table has been loaded into the in-memory cache used by the `Rewriter` plugin.

- `Rewriter_number_rewritten_queries`

  The number of queries rewritten by the `Rewriter` query rewrite plugin since it was loaded.

- `Rewriter_reload_error`

  Whether an error occurred the most recent time that the `rewrite_rules` table was loaded into the in-memory cache used by the `Rewriter` plugin. If the value is `OFF`, no error occurred. If the value is `ON`, an error occurred; check the `message` column of the `rewriter_rules` table for error messages.

5.6.5 The ddl_rewriter Plugin

MySQL 8.0.16 and higher includes a `ddl_rewriter` plugin that modifies `CREATE TABLE` statements received by the server before it parses and executes them. The plugin removes `ENCRYPTION`, `DATA DIRECTORY`, and `INDEX DIRECTORY` clauses, which may be helpful when restoring tables from SQL dump files created from databases that are encrypted or that have their tables stored outside the data directory. For example, the plugin may enable restoring such dump files into an unencrypted instance or in an environment where the paths outside the data directory are not accessible.

Before using the `ddl_rewriter` plugin, install it according to the instructions provided in Section 5.6.5.1, “Installing or Uninstalling ddl_rewriter”.

`ddl_rewriter` examines SQL statements received by the server prior to parsing, rewriting them according to these conditions:

- `ddl_rewriter` considers only `CREATE TABLE` statements, and only if they are standalone statements that occur at the beginning of an input line or at the beginning of prepared statement text. `ddl_rewriter` does not consider `CREATE TABLE` statements within stored program definitions. Statements can extend over multiple lines.
- Within statements considered for rewrite, instances of the following clauses are rewritten and each instance replaced by a single space:
  - `ENCRYPTION`
  - `DATA DIRECTORY` (at the table and partition levels)
  - `INDEX DIRECTORY` (at the table and partition levels)
- Rewriting does not depend on lettercase.

If `ddl_rewriter` rewrites a statement, it generates a warning:

```
mysql> CREATE TABLE t (i INT) DATA DIRECTORY '/var/mysql/data';
Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1105
Message: Query 'CREATE TABLE t (i INT) DATA DIRECTORY '/var/mysql/data''
         rewritten to 'CREATE TABLE t (i INT) ' by a query rewrite plugin
1 row in set (0.00 sec)
```

If the general query log or binary log is enabled, the server writes to it statements as they appear after any rewriting by `ddl_rewriter`.

When installed, `ddl_rewriter` exposes the Performance Schema `memory/rewriter/ddl_rewriter` instrument for tracking plugin memory use. See Section 26.12.17.10, “Memory Summary Tables”

5.6.5.1 Installing or Uninstalling ddl_rewriter

This section describes how to install or uninstall the `ddl_rewriter` plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

Note

If installed, the `ddl_rewriter` plugin involves some minimal overhead even when disabled. To avoid this overhead, install `ddl_rewriter` only for the period during which you intend to use it.

The primary use case is modification of statements restored from dump files, so the typical usage pattern is: 1) Install the plugin; 2) restore the dump file or files; 3) uninstall the plugin.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `ddl_rewriter`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To install the `ddl_rewriter` plugin, use the `INSTALL PLUGIN` statement (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN ddl_rewriter SONAME 'ddl_rewriter.so';
```

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS, PLUGIN_TYPE
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'ddl%';
+-------------+---------------+-------------+
| PLUGIN_NAME  | PLUGIN_STATUS | PLUGIN_TYPE |
+-------------+---------------+-------------+
| ddl_rewriter | ACTIVE        | AUDIT       |
+-------------+---------------+-------------+
```

As the preceding result shows, `ddl_rewriter` is implemented as an audit plugin.

If the plugin failed to initialize, check the server error log for diagnostic messages.

Once installed as just described, `ddl_rewriter` remains installed until uninstalled. To remove it, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN ddl_rewriter;
```

If `ddl_rewriter` is installed, you can use the `--ddl-rewriter` option for subsequent server startups to control `ddl_rewriter` plugin activation. For example, to prevent the plugin from being enabled at runtime, use this option:

```
[mysqld]
ddl-rewriter=OFF
```

5.6.5.2 ddl_rewriter Plugin Options

This section describes the command options that control operation of the `ddl_rewriter` plugin. If values specified at startup time are incorrect, the `ddl_rewriter` plugin may fail to initialize properly and the server does not load it.

To control activation of the `ddl_rewriter` plugin, use this option:

- `--ddl-rewriter[=value]`

Property	Value
Command-Line Format	`--ddl-rewriter[=value]`
Introduced	8.0.16
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`FORCE`
	`FORCE_PLUS_PERMANENT`

This option controls how the server loads the `ddl_rewriter` plugin at startup. It is available only if the plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load` or `--plugin-load-add`. See Section 5.6.5.1, “Installing or Uninstalling ddl_rewriter”.

The option value should be one of those available for plugin-loading options, as described in Section 5.6.1, “Installing and Uninstalling Plugins”. For example, `--ddl-rewriter=OFF` disables the plugin at server startup.

5.6.6 Version Tokens

MySQL includes Version Tokens, a feature that enables creation of and synchronization around server tokens that applications can use to prevent accessing incorrect or out-of-date data.

The Version Tokens interface has these characteristics:

- Version tokens are pairs consisting of a name that serves as a key or identifier, plus a value.
- Version tokens can be locked. An application can use token locks to indicate to other cooperating applications that tokens are in use and should not be modified.
- Version token lists are established per server (for example, to specify the server assignment or operational state). In addition, an application that communicates with a server can register its own list of tokens that indicate the state it requires the server to be in. An SQL statement sent by the application to a server not in the required state produces an error. This is a signal to the application that it should seek a different server in the required state to receive the SQL statement.

The following sections describe the components of Version Tokens, discuss how to install and use it, and provide reference information for its components.

5.6.6.1 Version Tokens Components

Version Tokens is based on a plugin library that implements these components:

- A server-side plugin named `version_tokens` holds the list of version tokens associated with the server and subscribes to notifications for statement execution events. The `version_tokens` plugin uses the audit plugin API to monitor incoming statements from clients and matches each client's session-specific version token list against the server version token list. If there is a match, the plugin lets the statement through and the server continues to process it. Otherwise, the plugin returns an error to the client and the statement fails.
- A set of user-defined functions (UDFs) provides an SQL-level API for manipulating and inspecting the list of server version tokens maintained by the plugin. The `VERSION_TOKEN_ADMIN` or `SUPER` privilege is required to call any of the Version Token UDFs.
- When the `version_tokens` plugin loads, it defines the `VERSION_TOKEN_ADMIN` dynamic privilege. This privilege can be granted to users of the UDFs.
- A system variable enables clients to specify the list of version tokens that register the required server state. If the server has a different state when a client sends a statement, the client receives an error.

5.6.6.2 Installing or Uninstalling Version Tokens

Note

If installed, Version Tokens involves some overhead. To avoid this overhead, do not install it unless you plan to use it.

This section describes how to install or uninstall Version Tokens, which is implemented in a plugin library file containing a plugin and user-defined functions (UDFs). For general information about installing or uninstalling plugins and UDFs, see Section 5.6.1, “Installing and Uninstalling Plugins”, and Section 5.7.1, “Installing and Uninstalling User-Defined Functions”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `version_tokens`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To install the Version Tokens plugin and UDFs, use the `INSTALL PLUGIN` and `CREATE FUNCTION` statements (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN version_tokens SONAME 'version_token.so';
CREATE FUNCTION version_tokens_set RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_show RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_edit RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_delete RETURNS STRING
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_lock_shared RETURNS INT
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_lock_exclusive RETURNS INT
  SONAME 'version_token.so';
CREATE FUNCTION version_tokens_unlock RETURNS INT
```

```
  SONAME 'version_token.so';
```

You must install the UDFs to manage the server's version token list, but you must also install the plugin because the UDFs will not work correctly without it.

If the plugin and UDFs are used on a master replication server, install them on all slave servers as well to avoid replication problems.

Once installed as just described, the plugin and UDFs remain installed until uninstalled. To remove them, use the `UNINSTALL PLUGIN` and `DROP FUNCTION` statements:

```
UNINSTALL PLUGIN version_tokens;
DROP FUNCTION version_tokens_set;
DROP FUNCTION version_tokens_show;
DROP FUNCTION version_tokens_edit;
DROP FUNCTION version_tokens_delete;
DROP FUNCTION version_tokens_lock_shared;
DROP FUNCTION version_tokens_lock_exclusive;
DROP FUNCTION version_tokens_unlock;
```

5.6.6.3 Using Version Tokens

Before using Version Tokens, install it according to the instructions provided at Section 5.6.6.2, “Installing or Uninstalling Version Tokens”.

A scenario in which Version Tokens can be useful is a system that accesses a collection of MySQL servers but needs to manage them for load balancing purposes by monitoring them and adjusting server assignments according to load changes. Such a system comprises these components:

- The collection of MySQL servers to be managed.
- An administrative or management application that communicates with the servers and organizes them into high-availability groups. Groups serve different purposes, and servers within each group may have different assignments. Assignment of a server within a certain group can change at any time.
- Client applications that access the servers to retrieve and update data, choosing servers according to the purposes assigned them. For example, a client should not send an update to a read-only server.

Version Tokens permit server access to be managed according to assignment without requiring clients to repeatedly query the servers about their assignments:

- The management application performs server assignments and establishes version tokens on each server to reflect its assignment. The application caches this information to provide a central access point to it.

  If at some point the management application needs to change a server assignment (for example, to change it from permitting writes to read only), it changes the server's version token list and updates its cache.

- To improve performance, client applications obtain cache information from the management application, enabling them to avoid having to retrieve information about server assignments for each statement. Based on the type of statements it will issue (for example, reads versus writes), a client selects an appropriate server and connects to it.

- In addition, the client sends to the server its own client-specific version tokens to register the assignment it requires of the server. For each statement sent by the client to the server, the server compares its own token list with the client token list. If the server token list contains all tokens present in the client token list with the same values, there is a match and the server executes the statement.

On the other hand, perhaps the management application has changed the server assignment and its version token list. In this case, the new server assignment may now be incompatible with the client requirements. A token mismatch between the server and client token lists occurs and the server returns an error in reply to the statement. This is an indication to the client to refresh its version token information from the management application cache, and to select a new server to communicate with.

The client-side logic for detecting version token errors and selecting a new server can be implemented different ways:

- The client can handle all version token registration, mismatch detection, and connection switching itself.
- The logic for those actions can be implemented in a connector that manages connections between clients and MySQL servers. Such a connector might handle mismatch error detection and statement resending itself, or it might pass the error to the application and leave it to the application to resend the statement.

The following example illustrates the preceding discussion in more concrete form.

When Version Tokens initializes on a given server, the server's version token list is empty. Token list maintenance is performed by calling user-defined functions (UDFs). The `VERSION_TOKEN_ADMIN` or `SUPER` privilege is required to call any of the Version Token UDFs, so token list modification is expected to be done by a management or administrative application that has that privilege.

Suppose that a management application communicates with a set of servers that are queried by clients to access employee and product databases (named `emp` and `prod`, respectively). All servers are permitted to process data retrieval statements, but only some of them are permitted to make database updates. To handle this on a database-specific basis, the management application establishes a list of version tokens on each server. In the token list for a given server, token names represent database names and token values are `read` or `write` depending on whether the database must be used in read-only fashion or whether it can take reads and writes.

Client applications register a list of version tokens they require the server to match by setting a system variable. Variable setting occurs on a client-specific basis, so different clients can register different requirements. By default, the client token list is empty, which matches any server token list. When a client sets its token list to a nonempty value, matching may succeed or fail, depending on the server version token list.

To define the version token list for a server, the management application calls the `version_tokens_set()` UDF. (There are also UDFs for modifying and displaying the token list, described later.) For example, the application might send these statements to a group of three servers:

Server 1:

```
mysql> SELECT version_tokens_set('emp=read;prod=read');
+-----------------------------------------+
| version_tokens_set('emp=read;prod=read') |
+-----------------------------------------+
| 2 version tokens set.                   |
+-----------------------------------------+
```

Server 2:

```
mysql> SELECT version_tokens_set('emp=write;prod=read');
+-----------------------------------------+
| version_tokens_set('emp=write;prod=read') |
+-----------------------------------------+
```

```
| 2 version tokens set.                     |
+-------------------------------------------+
```

Server 3:

```
mysql> SELECT version_tokens_set('emp=read;prod=write');
+-------------------------------------------+
| version_tokens_set('emp=read;prod=write') |
+-------------------------------------------+
| 2 version tokens set.                     |
+-------------------------------------------+
```

The token list in each case is specified as a semicolon-separated list of *`name=value`* pairs. The resulting token list values result in these server assingments:

- Any server accepts reads for either database.
- Only server 2 accepts updates for the `emp` database.
- Only server 3 accepts updates for the `prod` database.

In addition to assigning each server a version token list, the management application also maintains a cache that reflects the server assignments.

Before communicating with the servers, a client application contacts the management application and retrieves information about server assignments. Then the client selects a server based on those assignments. Suppose that a client wants to perform both reads and writes on the `emp` database. Based on the preceding assignments, only server 2 qualifies. The client connects to server 2 and registers its server requirements there by setting its `version_tokens_session` system variable:

```
mysql> SET @@SESSION.version_tokens_session = 'emp=write';
```

For subsequent statements sent by the client to server 2, the server compares its own version token list to the client list to check whether they match. If so, statements execute normally:

```
mysql> UPDATE emp.employee SET salary = salary * 1.1 WHERE id = 4981;
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT last_name, first_name FROM emp.employee WHERE id = 4981;
+-----------+------------+
| last_name | first_name |
+-----------+------------+
| Smith     | Abe        |
+-----------+------------+
1 row in set (0.01 sec)
```

Discrepancies between the server and client version token lists can occur two ways:

- A token name in the `version_tokens_session` value is not present in the server token list. In this case, an `ER_VTOKEN_PLUGIN_TOKEN_NOT_FOUND` error occurs.
- A token value in the `version_tokens_session` value differs from the value of the corresponding token in the server token list. In this case, an `ER_VTOKEN_PLUGIN_TOKEN_MISMATCH` error occurs.

As long as the assignment of server 2 does not change, the client continues to use it for reads and writes. But suppose that the management application wants to change server assignments so that writes for the `emp` database must be sent to server 1 instead of server 2. To do this, it uses `version_tokens_edit()` to modify the `emp` token value on the two servers (and updates its cache of server assignments):

Server 1:

```
mysql> SELECT version_tokens_edit('emp=write');
+---------------------------------+
| version_tokens_edit('emp=write') |
+---------------------------------+
| 1 version tokens updated.        |
+---------------------------------+
```

Server 2:

```
mysql> SELECT version_tokens_edit('emp=read');
+--------------------------------+
| version_tokens_edit('emp=read') |
+--------------------------------+
| 1 version tokens updated.       |
+--------------------------------+
```

`version_tokens_edit()` modifies the named tokens in the server token list and leaves other tokens unchanged.

The next time the client sends a statement to server 2, its own token list no longer matches the server token list and an error occurs:

```
mysql> UPDATE emp.employee SET salary = salary * 1.1 WHERE id = 4982;
ERROR 3136 (42000): Version token mismatch for emp. Correct value read
```

In this case, the client should contact the management application to obtain updated information about server assignments, select a new server, and send the failed statement to the new server.

Note

Each client must cooperate with Version Tokens by sending only statements in accordance with the token list that it registers with a given server. For example, if a client registers a token list of `'emp=read'`, there is nothing in Version Tokens to prevent the client from sending updates for the `emp` database. The client itself must refrain from doing so.

For each statement received from a client, the server implicitly uses locking, as follows:

- Take a shared lock for each token named in the client token list (that is, in the `version_tokens_session` value)
- Perform the comparison between the server and client token lists
- Execute the statement or produce an error depending on the comparison result
- Release the locks

The server uses shared locks so that comparisons for multiple sessions can occur without blocking, while preventing changes to the tokens for any session that attempts to acquire an exclusive lock before it manipulates tokens of the same names in the server token list.

The preceding example uses only a few of the user-defined included in the Version Tokens plugin library, but there are others. One set of UDFs permits the server's list of version tokens to be manipulated and inspected. Another set of UDFs permits version tokens to be locked and unlocked.

These UDFs permit the server's list of version tokens to be created, changed, removed, and inspected:

- `version_tokens_set()` completely replaces the current list and assigns a new list. The argument is a semicolon-separated list of *`name=value`* pairs.
- `version_tokens_edit()` enables partial modifications to the current list. It can add new tokens or change the values of existing tokens. The argument is a semicolon-separated list of *`name=value`* pairs.
- `version_tokens_delete()` deletes tokens from the current list. The argument is a semicolon-separated list of token names.
- `version_tokens_show()` displays the current token list. It takes no argument.

Each of those functions, if successful, returns a binary string indicating what action occurred. The following example establishes the server token list, modifies it by adding a new token, deletes some tokens, and displays the resulting token list:

```
mysql> SELECT version_tokens_set('tok1=a;tok2=b');
+-------------------------------------+
| version_tokens_set('tok1=a;tok2=b') |
+-------------------------------------+
| 2 version tokens set.               |
+-------------------------------------+
mysql> SELECT version_tokens_edit('tok3=c');
+------------------------------+
| version_tokens_edit('tok3=c') |
+------------------------------+
| 1 version tokens updated.     |
+------------------------------+
mysql> SELECT version_tokens_delete('tok2;tok1');
+-----------------------------------+
| version_tokens_delete('tok2;tok1') |
+-----------------------------------+
| 2 version tokens deleted.          |
+-----------------------------------+
mysql> SELECT version_tokens_show();
+----------------------+
| version_tokens_show() |
+----------------------+
| tok3=c;              |
+----------------------+
```

Warnings occur if a token list is malformed:

```
mysql> SELECT version_tokens_set('tok1=a; =c');
+---------------------------------+
| version_tokens_set('tok1=a; =c') |
+---------------------------------+
| 1 version tokens set.           |
+---------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 42000
Message: Invalid version token pair encountered. The list provided
         is only partially updated.
1 row in set (0.00 sec)
```

As mentioned previously, version tokens are defined using a semicolon-separated list of *`name=value`* pairs. Consider this invocation of `version_tokens_set()`:

```
mysql> SELECT version_tokens_set('tok1=b;;; tok2= a = b ; tok1 = 1\'2 3"4')
+-----------------------------------------------------------+
| version_tokens_set('tok1=b;;; tok2= a = b ; tok1 = 1\'2 3"4') |
+-----------------------------------------------------------+
| 3 version tokens set.                                     |
+-----------------------------------------------------------+
```

Version Tokens interprets the argument as follows:

- Whitespace around names and values is ignored. Whitespace within names and values is permitted. (For `version_tokens_delete()`, which takes a list of names without values, whitespace around names is ignored.)
- There is no quoting mechanism.
- Order of tokens is not significant except that if a token list contains multiple instances of a given token name, the last value takes precedence over earlier values.

Given those rules, the preceding `version_tokens_set()` call results in a token list with two tokens: `tok1` has the value `1'2 3"4`, and `tok2` has the value `a = b`. To verify this, call `version_tokens_show()`:

```
mysql> SELECT version_tokens_show();
+--------------------------+
| version_tokens_show()    |
+--------------------------+
| tok2=a = b;tok1=1'2 3"4; |
+--------------------------+
```

If the token list contains two tokens, why did `version_tokens_set()` return the value `3 version tokens set`? That occurred because the original token list contained two definitions for `tok1`, and the second definition replaced the first.

The Version Tokens token-manipulation UDFs place these constraints on token names and values:

- Token names cannot contain `=` or `;` characters and have a maximum length of 64 characters.
- Token values cannot contain `;` characters. Length of values is constrained by the value of the `max_allowed_packet` system variable.
- Version Tokens treats token names and values as binary strings, so comparisons are case-sensitive.

Version Tokens also includes a set of UDFs enabling tokens to be locked and unlocked:

- `version_tokens_lock_exclusive()` acquires exclusive version token locks. It takes a list of one or more lock names and a timeout value.
- `version_tokens_lock_shared()` acquires shared version token locks. It takes a list of one or more lock names and a timeout value.
- `version_tokens_unlock()` releases version token locks (exclusive and shared). It takes no argument.

Each locking function returns nonzero for success. Otherwise, an error occurs:

```
mysql> SELECT version_tokens_lock_shared('lock1', 'lock2', 0);
+-----------------------------------------------+
| version_tokens_lock_shared('lock1', 'lock2', 0) |
```

```
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+

mysql> SELECT version_tokens_lock_shared(NULL, 0);
ERROR 3131 (42000): Incorrect locking service lock name '(null)'.
```

Locking using Version Tokens locking functions is advisory; applications must agree to cooperate.

It is possible to lock nonexisting token names. This does not create the tokens.

Note

Version Tokens locking functions are based on the locking service described at Section 29.3.1, “The Locking Service”, and thus have the same semantics for shared and exclusive locks. (Version Tokens uses the locking service routines built into the server, not the locking service UDF interface, so those UDFs need not be installed to use Version Tokens.) Locks acquired by Version Tokens use a locking service namespace of `version_token_locks`. Locking service locks can be monitored using the Performance Schema, so this is also true for Version Tokens locks. For details, see Locking Service Monitoring.

For the Version Tokens locking functions, token name arguments are used exactly as specified. Surrounding whitespace is not ignored and `=` and `;` characters are permitted. This is because Version Tokens simply passes the token names to be locked as is to the locking service.

5.6.6.4 Version Tokens Reference

The following discussion serves as a reference to these Version Tokens components:

- Version Tokens Functions
- Version Tokens System Variables

Version Tokens Functions

The Version Tokens plugin library includes several user-defined functions. One set of UDFs permits the server's list of version tokens to be manipulated and inspected. Another set of UDFs permits version tokens to be locked and unlocked. The `VERSION_TOKEN_ADMIN` or `SUPER` privilege is required to invoke any Version Tokens UDF.

The following UDFs permit the server's list of version tokens to be created, changed, removed, and inspected. Interpretation of *`name_list`* and *`token_list`* arguments (including whitespace handling) occurs as described in Section 5.6.6.3, “Using Version Tokens”, which provides details about the syntax for specifying tokens, as well as additional examples.

- `version_tokens_delete(`*`name_list`*`)`

  Deletes tokens from the server's list of version tokens using the *`name_list`* argument and returns a binary string that indicates the outcome of the operation. *`name_list`* is a semicolon-separated list of version token names to delete.

```
mysql> SELECT version_tokens_delete('tok1;tok3');
+-----------------------------------+
| version_tokens_delete('tok1;tok3') |
+-----------------------------------+
| 2 version tokens deleted.          |
```

```
+----------------------------------+
```

An argument of `NULL` is treated as an empty string, which has no effect on the token list.

`version_tokens_delete()` deletes the tokens named in its argument, if they exist. (It is not an error to delete nonexisting tokens.) To clear the token list entirely without knowing which tokens are in the list, pass `NULL` or a string containing no tokens to `version_tokens_set()`:

```
mysql> SELECT version_tokens_set(NULL);
+-----------------------------+
| version_tokens_set(NULL)    |
+-----------------------------+
| Version tokens list cleared. |
+-----------------------------+
mysql> SELECT version_tokens_set('');
+-----------------------------+
| version_tokens_set('')      |
+-----------------------------+
| Version tokens list cleared. |
+-----------------------------+
```

- `version_tokens_edit(`*`token_list`*`)`

  Modifies the server's list of version tokens using the *`token_list`* argument and returns a binary string that indicates the outcome of the operation. *`token_list`* is a semicolon-separated list of *`name=value`* pairs specifying the name of each token to be defined and its value. If a token exists, its value is updated with the given value. If a token does not exist, it is created with the given value. If the argument is `NULL` or a string containing no tokens, the token list remains unchanged.

```
mysql> SELECT version_tokens_set('tok1=value1;tok2=value2');
+-----------------------------------------------+
| version_tokens_set('tok1=value1;tok2=value2') |
+-----------------------------------------------+
| 2 version tokens set.                         |
+-----------------------------------------------+
mysql> SELECT version_tokens_edit('tok2=new_value2;tok3=new_value3');
+-------------------------------------------------------+
| version_tokens_edit('tok2=new_value2;tok3=new_value3') |
+-------------------------------------------------------+
| 2 version tokens updated.                             |
+-------------------------------------------------------+
```

- `version_tokens_set(`*`token_list`*`)`

  Replaces the server's list of version tokens with the tokens defined in the *`token_list`* argument and returns a binary string that indicates the outcome of the operation. *`token_list`* is a semicolon-separated list of *`name=value`* pairs specifying the name of each token to be defined and its value. If the argument is `NULL` or a string containing no tokens, the token list is cleared.

```
mysql> SELECT version_tokens_set('tok1=value1;tok2=value2');
+-----------------------------------------------+
| version_tokens_set('tok1=value1;tok2=value2') |
+-----------------------------------------------+
| 2 version tokens set.                         |
+-----------------------------------------------+
```

- `version_tokens_show()`

  Returns the server's list of version tokens as a binary string containing a semicolon-separated list of *`name=value`* pairs.

```
mysql> SELECT version_tokens_show();
+-------------------------+
| version_tokens_show()   |
+-------------------------+
| tok2=value2;tok1=value1; |
+-------------------------+
```

The following UDFs permit version tokens to be locked and unlocked:

- `version_tokens_lock_exclusive(token_name[, token_name] ..., timeout)`

  Acquires exclusive locks on one or more version tokens, specified by name as strings, timing out with an error if the locks are not acquired within the given timeout value.

```
mysql> SELECT version_tokens_lock_exclusive('lock1', 'lock2', 10);
+-----------------------------------------------------+
| version_tokens_lock_exclusive('lock1', 'lock2', 10) |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
```

- `version_tokens_lock_shared(token_name[, token_name] ..., timeout)`

  Acquires shared locks on one or more version tokens, specified by name as strings, timing out with an error if the locks are not acquired within the given timeout value.

```
mysql> SELECT version_tokens_lock_shared('lock1', 'lock2', 10);
+-------------------------------------------------+
| version_tokens_lock_shared('lock1', 'lock2', 10) |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
```

- `version_tokens_unlock()`

  Releases all locks that were acquired within the current session using `version_tokens_lock_exclusive()` and `version_tokens_lock_shared()`.

```
mysql> SELECT version_tokens_unlock();
+------------------------+
| version_tokens_unlock() |
+------------------------+
|                      1 |
+------------------------+
```

The locking functions share these characteristics:

- The return value is nonzero for success. Otherwise, an error occurs.
- Token names are strings.
- In contrast to argument handling for the UDFs that manipulate the server token list, whitespace surrounding token name arguments is not ignored and `=` and `;` characters are permitted.
- It is possible to lock nonexisting token names. This does not create the tokens.
- Timeout values are nonnegative integers representing the time in seconds to wait to acquire locks before timing out with an error. If the timeout is 0, there is no waiting and the function produces an error if locks cannot be acquired immediately.

- Version Tokens locking functions are based on the locking service described at Section 29.3.1, “The Locking Service”.

Version Tokens System Variables

Version Tokens supports the following system variables. These variables are unavailable unless the Version Tokens plugin is installed (see Section 5.6.6.2, “Installing or Uninstalling Version Tokens”).

System variables:

- `version_tokens_session`

Property	Value
Command-Line Format	`--version-tokens-session=value`
System Variable	`version_tokens_session`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`NULL`

The session value of this variable specifies the client version token list and indicates the tokens that the client session requires the server version token list to have.

If the `version_tokens_session` variable is `NULL` (the default) or has an empty value, any server version token list matches. (In effect, an empty value disables matching requirements.)

If the `version_tokens_session` variable has a nonempty value, any mismatch between its value and the server version token list results in an error for any statement the session sends to the server. A mismatch occurs under these conditions:

- A token name in the `version_tokens_session` value is not present in the server token list. In this case, an `ER_VTOKEN_PLUGIN_TOKEN_NOT_FOUND` error occurs.
- A token value in the `version_tokens_session` value differs from the value of the corresponding token in the server token list. In this case, an `ER_VTOKEN_PLUGIN_TOKEN_MISMATCH` error occurs.

It is not a mismatch for the server version token list to include a token not named in the `version_tokens_session` value.

Suppose that a management application has set the server token list as follows:

```
mysql> SELECT version_tokens_set('tok1=a;tok2=b;tok3=c');
+-----------------------------------------+
| version_tokens_set('tok1=a;tok2=b;tok3=c') |
+-----------------------------------------+
| 3 version tokens set.                   |
+-----------------------------------------+
```

A client registers the tokens it requires the server to match by setting its `version_tokens_session` value. Then, for each subsequent statement sent by the client, the server checks its token list against the client `version_tokens_session` value and produces an error if there is a mismatch:

```
mysql> SET @@SESSION.version_tokens_session = 'tok1=a;tok2=b';
mysql> SELECT 1;
+---+
| 1 |
+---+
| 1 |
+---+

mysql> SET @@SESSION.version_tokens_session = 'tok1=b';
mysql> SELECT 1;
ERROR 3136 (42000): Version token mismatch for tok1. Correct value a
```

The first `SELECT` succeeds because the client tokens `tok1` and `tok2` are present in the server token list and each token has the same value in the server list. The second `SELECT` fails because, although `tok1` is present in the server token list, it has a different value than specified by the client.

At this point, any statement sent by the client fails, unless the server token list changes such that it matches again. Suppose that the management application changes the server token list as follows:

```
mysql> SELECT version_tokens_edit('tok1=b');
+-------------------------------+
| version_tokens_edit('tok1=b') |
+-------------------------------+
| 1 version tokens updated.     |
+-------------------------------+
mysql> SELECT version_tokens_show();
+-----------------------+
| version_tokens_show() |
+-----------------------+
| tok3=c;tok1=b;tok2=b; |
+-----------------------+
```

Now the client `version_tokens_session` value matches the server token list and the client can once again successfully execute statements:

```
mysql> SELECT 1;
+---+
| 1 |
+---+
| 1 |
+---+
```

- `version_tokens_session_number`

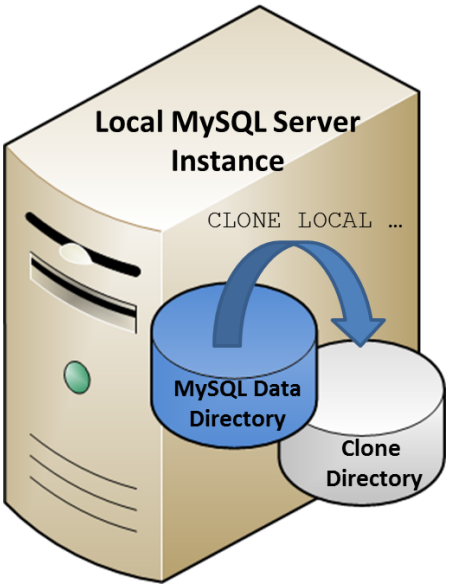
Property	Value
Command-Line Format	`--version-tokens-session-number=#`
System Variable	`version_tokens_session_number`
Scope	Global, Session
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`

This variable is for internal use.

5.6.7 The Clone Plugin

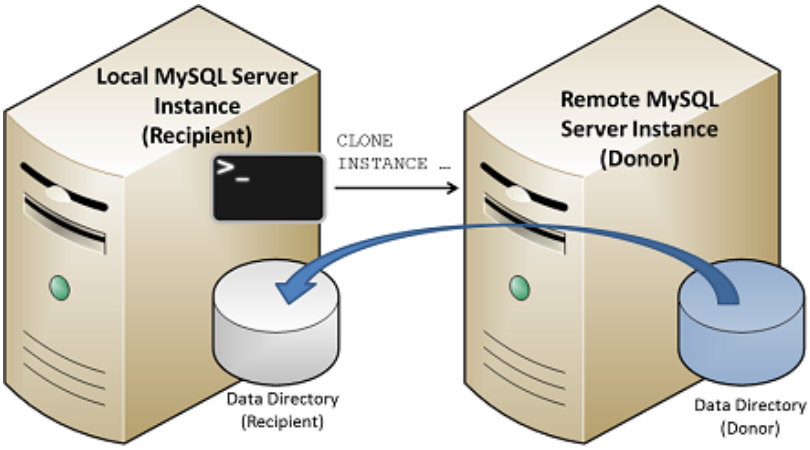
The clone plugin permits cloning data locally or from a remote MySQL server instance. Cloned data is a physical snapshot of data stored in `InnoDB` that includes schemas, tables, tablespaces, and data dictionary metadata. The cloned data comprises a fully functional data directory, which permits using the clone plugin for MySQL server provisioning.

Figure 5.1 Local Cloning Operation



A local cloning operation clones data from the MySQL server instance where the cloning operation is initiated to a directory on the same server or node where MySQL server instance runs.

Figure 5.2 Remote Cloning Operation



A remote cloning operation involves a local MySQL server instance (the “recipient”) where the cloning operation is initiated, and a remote MySQL server instance (the “donor”) where the source data is located. When a remote cloning operation is initiated on the recipient, cloned data is transferred over the network from the donor to the recipient. By default, a remote cloning operation removes the data in the recipient data directory and replaces it with the cloned data. Optionally, you can clone data to a different directory on the recipient to avoid removing existing data.

There is no difference with respect to data that is cloned by a local cloning operation as compared to a remote cloning operation. Both operations clone the same data.

The clone plugin supports replication. In addition to cloning data, a cloning operation extracts and transfers replication coordinates from the donor and applies them on the recipient, which enables using the clone plugin for provisioning Group Replication members and replication slaves. Using the clone plugin for provisioning is considerably faster and more efficient than replicating a large number of transactions (see Section 5.6.7.6, “Cloning for Replication”). Group Replication members can also be configured to use the clone plugin as an alternative method of recovery, so that members automatically choose the most efficient way to retrieve group data from seed members. For more information, see Section 18.4.3.1, “Cloning for Distributed Recovery”.

The clone plugin supports cloning of encrypted and page-compressed data. See Section 5.6.7.4, “Cloning Encrypted Data”, and Section 5.6.7.5, “Cloning Compressed Data”.

The clone plugin must be installed before you can use it. For installation instructions, see Section 5.6.7.1, “Installing the Clone Plugin”. For cloning instructions, see Section 5.6.7.2, “Cloning Data Locally”, and Section 5.6.7.3, “Cloning Remote Data”.

Performance Schema tables and instrumentation are provided for monitoring cloning operations. See Section 5.6.7.9, “Monitoring Cloning Operations”.

5.6.7.1 Installing the Clone Plugin

This section describes how to install and configure the clone plugin. For remote cloning operations, the clone plugin must be installed on the donor and recipient MySQL server instances.

For general information about installing or uninstalling plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, set the value of `plugin_dir` at server startup to tell the server the plugin directory location.

The plugin library file base name is `mysql_clone.so`. The file name suffix differs by platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in your `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=mysql_clone.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Note

The `--plugin-load-add` option cannot be used to load the clone plugin when restarting the server during an upgrade from a previous MySQL version. For example, after upgrading binaries or packages from MySQL 5.7 to MySQL 8.0, attempting to restart the server with `plugin-load-add=mysql_clone.so` causes this error: `[ERROR] [MY-013238] [Server] Error installing plugin 'clone': Cannot install during upgrade.` The workaround is to upgrade the server before attempting to start the server with `plugin-load-add=mysql_clone.so`.

Alternatively, to load the plugin at runtime, use this statement (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN clone SONAME 'mysql_clone.so';
```

`INSTALL PLUGIN` loads the plugin, and also registers it in the `mysql.plugins` system table to cause the plugin to be loaded for each subsequent normal server startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME = 'clone';
+-----------------------+---------------+
| PLUGIN_NAME           | PLUGIN_STATUS |
+-----------------------+---------------+
| clone                 | ACTIVE        |
+-----------------------+---------------+
```

If the plugin failed to initialize, check the server error log for clone or plugin-related diagnostic messages.

If the plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load-add`, you can use the `--clone` option at server startup to control the plugin activation state. For example, to load the plugin at startup and prevent it from being removed at runtime, use these options:

```
[mysqld]
plugin-load-add=mysql_clone.so
clone=FORCE_PLUS_PERMANENT
```

If you want to prevent the server from running without the clone plugin, use `--clone` with a value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

For more information about plugin activation states, see Controlling Plugin Activation State.

5.6.7.2 Cloning Data Locally

The clone plugin supports the following syntax for cloning data locally; that is, cloning data from the local MySQL data directory to another directory on the same server or node where the MySQL server instance runs:

```
CLONE LOCAL DATA DIRECTORY [=] 'clone_dir';
```

To use `CLONE` syntax, the clone plugin must be installed. For installation instructions, see Section 5.6.7.1, “Installing the Clone Plugin”.

The `BACKUP_ADMIN` privilege is required to execute `CLONE LOCAL DATA DIRECTORY` statements.

```
mysql> GRANT BACKUP_ADMIN ON *.* TO 'clone_user';
```

where *`clone_user`* is the MySQL user that will perform the cloning operation. The user you select to perform the cloning operation can be any MySQL user with the `BACKUP_ADMIN` privilege on *.*.

The following example demonstrates cloning data locally:

```
mysql> CLONE LOCAL DATA DIRECTORY = '/path/to/clone_dir';
```

where */path/to/clone_dir* is the full path of the local directory that data is cloned to. An absolute path is required, and the specified directory ("*clone_dir*") must not exist, but the specified path must be an existent path. The MySQL server must have the necessary write access to create the directory.

Note

A local cloning operation does not support cloning of user-created tables or tablespaces that reside outside of the data directory. Attempting to clone such tables or tablespaces causes the following error: `ERROR 1086 (HY000): File '/path/to/tablespace_name.ibd' already exists`. Cloning a tablespace with the same path as the source tablespace would cause a conflict and is therefore prohibited.

All other user-created `InnoDB` tables and tablespaces, the `InnoDB` system tablespace, redo logs, and undo tablespaces are cloned to the specified directory.

If desired, you can start the MySQL server on the cloned directory after the cloning operation is complete.

```
shell> mysqld_safe --datadir=clone_dir
```

where *clone_dir* is the directory that data was cloned to.

For information about monitoring cloning operation status and progress, see Section 5.6.7.9, "Monitoring Cloning Operations".

5.6.7.3 Cloning Remote Data

The clone plugin supports the following syntax for cloning remote data; that is, cloning data from a remote MySQL server instance (the donor) and transferring it to the MySQL instance where the cloning operation was initiated (the recipient).

```
CLONE INSTANCE FROM 'user'@'host':port
IDENTIFIED BY 'password'
[DATA DIRECTORY [=] 'clone_dir']
[REQUIRE [NO] SSL];
```

where:

- *user* is the clone user on the donor MySQL server instance.
- *password* is the *user* password.
- *host* is the `hostname` address of the donor MySQL server instance. Internet Protocol version 6 (IPv6) address format is not supported. An alias to the IPv6 address can be used instead. An IPv4 address can be used as is.
- *port* is the `port` number of the donor MySQL server instance. (The X Protocol port specified by `mysqlx_port` is not supported. Connecting to the donor MySQL server instance through MySQL Router is also not supported.)
- `DATA DIRECTORY [=] 'clone_dir'` is an optional clause used to specify a directory on the recipient for the data you are cloning. Use this option if you do not want to remove existing data in the recipient data directory. An absolute path is required, and the directory must not exist. The MySQL server must have the necessary write access to create the directory.

  When the optional `DATA DIRECTORY [=] 'clone_dir'` clause is not used, a cloning operation removes existing data in the recipient data directory, replaces it with the cloned data, and automatically restarts the server afterward.

- `[REQUIRE [NO] SSL]` explicitly specifies whether an encrypted connection is to be used or not when transferring cloned data over the network. An error is returned if the explicit specification cannot be satisfied. If an SSL clause is not specified, clone attempts to establish an encrypted connection by default, falling back to an insecure connection if the secure connection attempt fails. A secure connection is required when cloning encrypted data regardless of whether this clause is specified. For more information, see Configuring an Encrypted Connection for Cloning.

Note

By default, user-created `InnoDB` tables and tablespaces that reside in the data directory on the donor MySQL server instance are cloned to the data directory on the recipient MySQL server instance. If the `DATA DIRECTORY [=] 'clone_dir'` clause is specified, they are cloned to the specified directory.

User-created `InnoDB` tables and tablespaces that reside outside of the data directory on the donor MySQL server instance are cloned to the same path on the recipient MySQL server instance. An error is reported if a table or tablespace already exists.

By default, the `InnoDB` system tablespace, redo logs, and undo tablespaces are cloned to the same locations that are configured on the donor (as defined by `innodb_data_home_dir` and `innodb_data_file_path`, `innodb_log_group_home_dir`, and `innodb_undo_directory`, respectively). If the `DATA DIRECTORY [=] 'clone_dir'` clause is specified, those tablespaces and logs are cloned to the specified directory.

Remote Cloning Prerequisites

To perform a cloning operation, the clone plugin must be active on both the donor and recipient MySQL server instances. For installation instructions, see Section 5.6.7.1, “Installing the Clone Plugin”.

A MySQL user on the donor and recipient is required for executing the cloning operation (the “clone user”).

- On the donor, the clone user requires the `BACKUP_ADMIN` privilege for accessing and transferring data from the donor, and for blocking DDL during the cloning operation.
- On the recipient, the clone user requires the `CLONE_ADMIN` privilege for replacing recipient data, blocking DDL during the cloning operation, and automatically restarting the server. The `CLONE_ADMIN` privilege includes implicit `BACKUP_ADMIN` and `SHUTDOWN` privileges.

Instructions for creating the clone user and granting the required privileges are included in the remote cloning example that follows this prerequisite information.

The following prerequisites are checked when the `CLONE INSTANCE` statement is executed:

- The donor and recipient must have the same MySQL server version. The clone plugin is supported in MYSQL 8.0.17 and higher.

```
mysql> SHOW VARIABLES LIKE 'version';
 +---------------+--------+
| Variable_name | Value  |
+---------------+--------+
| version       | 8.0.17 |
+---------------+--------+
```

- The donor and recipient MySQL server instances must run on the same operating system and platform. For example, if the donor instance runs on a Linux 64-bit platform, the recipient instance must also run

on that platform. Refer to your operating system documentation for information about how to determine your operating system platform.

- The recipient must have enough disk space for the cloned data. By default, recipient data is removed prior to cloning the donor data, so you only require enough space for the donor data. If you clone to a named directory using the `DATA DIRECTORY` clause, you must have enough disk space for the existing recipient data and the cloned data. You can estimate the size of your data by checking the data directory size on your file system and the size of any tablespaces that reside outside of the data directory. When estimating data size on the donor, remember that only `InnoDB` data is cloned. If you store data in other storage engines, adjust your data size estimate accordingly.

- `InnoDB` permits creating some tablespace types outside of the data directory. If the donor MySQL server instance has tablespaces that reside outside of the data directory, the cloning operation must be able access those tablespaces. You can query the `INFORMATION_SCHEMA.FILES` table to identify tablespaces that reside outside of the data directory. Files that reside outside of the data directory have a fully qualified path to a directory other than the data directory.

```
mysql> SELECT FILE_NAME FROM INFORMATION_SCHEMA.FILES;
```

- Plugins that are active on the donor, including any keyring plugin, must also be active on the recipient. You can identify active plugins by issuing a `SHOW PLUGINS` statement or by querying the `INFORMATION_SCHEMA.PLUGINS` table.

- The donor and recipient must have the same MySQL server character set and collation. For information about MySQL server character set and collation configuration, see Section 10.15, “Character Set Configuration”.

- The same `innodb_page_size` and `innodb_data_file_path` settings are required on the donor and recipient. The `innodb_data_file_path` setting on the donor and recipient must specify the same number of data files of an equivalent size. You can check variable settings using `SHOW VARIABLES` syntax.

```
mysql> SHOW VARIABLES LIKE 'innodb_page_size';
mysql> SHOW VARIABLES LIKE 'innodb_data_file_path';
```

- If cloning encrypted or page-compressed data, the donor and recipient must have the same file system block size. For page-compressed data, the recipient file system must support sparse files and hole punching for hole punching to occur on the recipient. For information about these features and how to identify tables and tablespaces that use them, see Section 5.6.7.4, “Cloning Encrypted Data”, and Section 5.6.7.5, “Cloning Compressed Data”. To determine your file system block size, refer to your operating system documentation.

- A secure connection is required if you are cloning encrypted data. See Configuring an Encrypted Connection for Cloning.

- The `clone_valid_donor_list` setting on the recipient must include the host address of the donor MySQL server instance. You can only clone data from a host on the valid donor list. A MySQL user with the `SYSTEM_VARIABLES_ADMIN` privilege is required to configure this variable. Instructions for setting the `clone_valid_donor_list` variable are provided in the remote cloning example that follows this section. You can check the `clone_valid_donor_list` setting using `SHOW VARIABLES` syntax.

```
mysql> SHOW VARIABLES LIKE 'clone_valid_donor_list';
```

- There must be no other cloning operation running. Only a single cloning operation is permitted at a time. To determine if a clone operation is running, query the `clone_status` table. See Monitoring Cloning Operations using Performance Schema Clone Tables.

The following prerequisites also apply:

- Undo tablespace file names on the donor must be unique. When data is cloned to the recipient, undo tablespaces, regardless of their location on the donor, are cloned to the `innodb_undo_directory` location on the recipient or to the directory specified by the `DATA DIRECTORY [=] 'clone_dir'` clause, if used. Duplicate undo tablespace file names on the donor are not permitted for this reason. As of MySQL 8.0.18, an error is reported if duplicate undo tablespace file names are encountered during a cloning operation. Prior to MySQL 8.0.18, cloning undo tablespaces with the same file name could result in undo tablespace files being overwritten on the recipient.

  To view undo tablespace file names on the donor to ensure that they are unique, query `INFORMATION_SCHEMA.FILES`:

  ```
  mysql> SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES
         WHERE FILE_TYPE LIKE 'UNDO LOG';
  ```

  For information about dropping and adding undo tablespace files, see Section 15.6.3.4, “Undo Tablespaces”.

- By default, the recipient MySQL server instance is restarted (stopped and started) automatically after the data is cloned. For an automatic restart to occur, a monitoring process must be available on the recipient to detect server shutdowns. Otherwise, the cloning operation halts with the following error after the data is cloned, and the recipient MySQL server instance is shut down:

  ```
  ERROR 3707 (HY000): Restart server failed (mysqld is not managed by supervisor process).
  ```

  This error does not indicate a cloning failure. It means that the recipient MySQL server instance must be started again manually after the data is cloned. After starting the server manually, you can connect to the recipient MySQL server instance and check the Performance Schema clone tables to verify that the cloning operation completed successfully (see Monitoring Cloning Operations using Performance Schema Clone Tables.) The `RESTART` statement has the same monitoring process requirement. For more information, see Section 13.7.8.8, “RESTART Syntax”. This requirement is not applicable if cloning to a named directory using the `DATA DIRECTORY` clause, as an automatic restart is not performed in this case.

- Several variables control various aspects of a remote cloning operation. Before performing a remote cloning operation, review the variables and adjust settings as necessary to suit your computing environment. Clone variables are set on recipient MySQL server instance where the cloning operation is executed. See Section 5.6.7.12, “Clone System Variables”.

Cloning Remote Data

The following example demonstrates cloning remote data. By default, a remote cloning operation removes the data in the recipient data directory, replaces it with the cloned data, and restarts the MySQL server afterward.

The example assumes that remote cloning prerequisites are met. See Remote Cloning Prerequisites.

1. Login to the donor MySQL server instance with an administrative user account.

   a. Create a clone user with the `BACKUP_ADMIN` privilege.

   ```
   mysql> CREATE USER 'donor_clone_user'@'example.donor.host.com' IDENTIFIED BY 'password';
   mysql> GRANT BACKUP_ADMIN on *.* to 'donor_clone_user'@'example.donor.host.com';
   ```

   b. Install the clone plugin:

```
mysql> INSTALL PLUGIN clone SONAME 'mysql_clone.so';
```

2. Login to the recipient MySQL server instance with an administrative user account.

   a. Create a clone user with the `CLONE_ADMIN` privilege.

   ```
   mysql> CREATE USER 'recipient_clone_user'@'example.recipient.host.com' IDENTIFIED BY 'password';
   mysql> GRANT CLONE_ADMIN on *.* to 'recipient_clone_user'@'example.recipient.host.com';
   ```

   b. Install the clone plugin:

   ```
   mysql> INSTALL PLUGIN clone SONAME 'mysql_clone.so';
   ```

   c. Add the host address of the donor MySQL server instance to the `clone_valid_donor_list` variable setting.

   ```
   mysql> SET GLOBAL clone_valid_donor_list = 'example.donor.host.com:3306';
   ```

3. Log on to the recipient MySQL server instance as the clone user you created previously (`recipient_clone_user'@'example.recipient.host.com`) and execute the `CLONE INSTANCE` statement.

   ```
   mysql> CLONE INSTANCE FROM 'donor_clone_user'@'example.donor.host.com':3306
          IDENTIFIED BY 'password';
   ```

   After the data is cloned, the MySQL server instance on the recipient is restarted automatically.

   For information about monitoring cloning operation status and progress, see Section 5.6.7.9, “Monitoring Cloning Operations”.

Cloning to a Named Directory

By default, a remote cloning operation removes the data in the recipient data directory and replaces it with the cloned data. By cloning to a named directory, you can avoid removing existing data from the recipient data directory.

The procedure for cloning to a named directory is the same procedure described in Cloning Remote Data with one exception: The `CLONE INSTANCE` statement must include the `DATA DIRECTORY` clause. For example:

```
mysql> CLONE INSTANCE FROM 'user'@'example.donor.host.com':3306
       IDENTIFIED BY 'password'
       DATA DIRECTORY = '/path/to/clone_dir';
```

An absolute path is required, and the directory must not exist. The MySQL server must have the necessary write access to create the directory.

When cloning to a named directory, the recipient MySQL server instance is not restarted automatically after the data is cloned. If you want to restart the MySQL server on the named directory, you must do so manually:

```
shell> mysqld_safe --datadir=/path/to/clone_dir
```

where */path/to/clone_dir* is the path to the named directory on the recipient.

Configuring an Encrypted Connection for Cloning

You can configure an encrypted connection for remote cloning operations to protect data as it is cloned over the network. An encrypted connection is required by default when cloning encrypted data. (see Section 5.6.7.4, “Cloning Encrypted Data”.)

The instructions that follow describe how to configure the recipient MySQL server instance to use an encrypted connection. It is assumed that the donor MySQL server instance is already configured to use encrypted connections. If not, refer to Section 6.3.1, “Configuring MySQL to Use Encrypted Connections” for server-side configuration instructions.

To configure the recipient MySQL server instance to use an encrypted connection:

1. Make the client certificate and key files of the donor MySQL server instance available to the recipient host. Either distribute the files to the recipient host using a secure channel or place them on a mounted partition that is accessible to the recipient host. The client certificate and key files to make available include:

   - `ca.pem`

     The self-signed certificate authority (CA) file.

   - `client-cert.pem`

     The client public key certificate file.

   - `client-key.pem`

     The client private key file.

2. Configure the following SSL options on the recipient MySQL server instance.

   - `clone_ssl_ca`

     Specifies the path to the self-signed certificate authority (CA) file.

   - `clone_ssl_cert`

     Specifies the path to the client public key certificate file.

   - `clone_ssl_key`

     Specifies the path to the client private key file.

   For example:

   ```
   clone_ssl_ca=/path/to/ca.pem
   clone_ssl_cert=/path/to/client-cert.pem
   clone_ssl_key=/path/to/client-key.pem
   ```

3. To require that an encrypted connection is used, include the `REQUIRE SSL` clause when issuing the `CLONE` statement on the recipient.

   ```
   mysql> CLONE INSTANCE FROM 'user'@'example.donor.host.com':3306
          IDENTIFIED BY 'password'
          DATA DIRECTORY = '/path/to/clone_dir'
          REQUIRE SSL;
   ```

If an SSL clause is not specified, the clone plugin attempts to establish an encrypted connection by default, falling back to an unencrypted connection if the encrypted connection attempt fails.

Note

If you are cloning encrypted data, an encrypted connection is required by default regardless of whether the `REQUIRE SSL` clause is specified. Using `REQUIRE NO SSL` causes an error if you attempt to clone encrypted data.

5.6.7.4 Cloning Encrypted Data

Cloning of encrypted data is supported. The following requirements apply:

- A secure connection is required when cloning remote data to ensure safe transfer of unencrypted tablespace keys over the network. Tablespace keys are decrypted at the donor before transport and re-encrypted at the recipient using the recipient master key. An error is reported if an encrypted connection is not available or the `REQUIRE NO SSL` clause is used in the `CLONE INSTANCE` statement. For information about configuring an encrypted connection for cloning, see Configuring an Encrypted Connection for Cloning.
- When cloning data to a local data directory that uses a locally managed keyring, the same keyring must be used when starting the MySQL server on the clone directory.
- When cloning data to a remote data directory (the recipient directory) that uses a locally managed keyring, the recipient keyring must be used when starting the MySQL sever on the cloned directory.

Note

The `innodb_redo_log_encrypt` and `innodb_undo_log_encrypt` variable settings cannot be modified while a cloning operation is in progress.

For information about the data encryption feature, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”.

5.6.7.5 Cloning Compressed Data

Cloning of page-compressed data is supported. The following requirements apply when cloning remote data:

- The recipient file system must support sparse files and hole punching for hole punching to occur on the recipient.
- The donor and recipient file systems must have the same block size. If file system block sizes differ, an error similar to the following is reported: `ERROR 3868 (HY000): Clone Configuration FS Block Size: Donor value: 114688 is different from Recipient value: 4096`.

For information about the page compression feature, see Section 15.9.2, “InnoDB Page Compression”.

5.6.7.6 Cloning for Replication

The clone plugin supports replication. In addition to cloning data, a cloning operation extracts and transfers replication coordinates from the donor and applies them on the recipient, which enables using the clone plugin for provisioning Group Replication members and replication slaves. Using the clone plugin for provisioning is considerably faster and more efficient than replicating a large number of transactions.

Note

Group Replication members can also be configured to use the clone plugin as an alternative method of recovery, so that members automatically choose the most

efficient way to retrieve group data from seed members. For more information, see Section 18.4.3.1, “Cloning for Distributed Recovery”.

Both binary log position (filename, offset) and GTID coordinates are extracted and transferred from the donor MySQL server instance.

- Issue this query on a cloned MySQL server instance to view the binary log position that was applied:

```
mysql> SELECT BINLOG_FILE, BINLOG_POSITION FROM performance_schema.clone_status;
```

- Issue this query on a cloned MySQL server instance to view the GTID of last transaction cloned to the recipient:

```
mysql> SELECT @@GLOBAL.GTID_EXECUTED;
```

  If a recipient running with `GTID_MODE=ON` clones from a donor with `GTID_MODE=OFF`, both the GTID and binary log position replication coordinates are applied on the recipient.

The replication coordinates that are applied on the recipient permit initiating replication from the donor at a consistent position in the replication stream, assuming that the binary logs required for the recipient to catch up to the donor are not purged between the time that the data is cloned and the time that replication is started. If the required binary logs are not available, a replication handshake error is reported.

A cloned instance should be added to a group or cluster without excessive delay to avoid required binary logs being purged or the new member lagging behind significantly, requiring more recovery time.

The cloning operation does not configure or modify replication-related configuration settings on the recipient MySQL server instance.

To clone for replication, perform the following steps:

1. Clone data from the donor MySQL server instance to the recipient.

   - If you are adding a member to a Group Replication cluster, the donor must be a member in the cluster.
   - If you are adding a slave to a MySQL replication group, the donor must be a master or slave in the replication group.

   For cloning instructions, see Section 5.6.7.3, “Cloning Remote Data”.

2. After the cloning operation completes successfully, add the recipient MySQL server instance to the Group Replication cluster or MySQL replication group.

   The instructions that follow provide abbreviated examples for adding a recipient MySQL server instance to a Group Replication cluster, a replication group that uses GTID-based transactions as the replication data source, and a replication group that uses binary log file position based replication. Refer to the instructions that apply to your replication setup.

   - **Adding to a Group Replication cluster.** To add the recipient MySQL server instance to a Group Replication cluster, configure the instance for Group Replication, and add the instance to the group, as shown in this abbreviated example:

```
mysql> CHANGE MASTER TO MASTER_USER = 'user_name', MASTER_PASSWORD = 'password',
       ...
       FOR CHANNEL 'group_replication_recovery';
mysql> START GROUP_REPLICATION;
```

For detailed information about adding an instance to a Group Replication cluster, refer to Section 18.2.1.6, “Adding Instances to the Group”.

- **Adding to a replication group that uses GTID-based transactions.** To add a recipient MySQL server instance to a MySQL replication group that uses GTID-based transactions as the replication data source, configure the instance as required, and add the instance as shown in the following abbreviated example.

  The `CHANGE MASTER TO` statement must define the host address and port number of the donor instance, and the `MASTER_AUTO_POSITION` option should be enabled, as shown:

  ```
  mysql> CHANGE MASTER TO MASTER_HOST = 'donor_host_name', MASTER_PORT = donor_port_num,
         ...
         MASTER_AUTO_POSITION = 1;
  mysql> START SLAVE USER = 'user_name' PASSWORD = 'password';
  ```

  For detailed information about adding an instance to a MySQL replication group that uses GTID-based transactions as the replication data source, refer to Section 17.1.3, “Replication with Global Transaction Identifiers”.

- **Adding to a MySQL replication group that uses binary log file position.** To add a recipient MySQL server instance to a MySQL replication group that uses binary log file position based replication, configure the instance as required, and add the instance as shown in the following abbreviated example.

  Adding the recipient MySQL server instance to the group requires knowing the binary log position applied to the recipient instance.

  ```
  mysql> SELECT BINLOG_FILE, BINLOG_POSITION FROM performance_schema.clone_status;
  mysql> CHANGE MASTER TO MASTER_HOST = 'donor_host_name', MASTER_PORT = donor_port_num,
         ...
         MASTER_LOG_FILE = 'master_log_name',
         MASTER_LOG_POS = master_log_pos;
  mysql> START SLAVE USER = 'user_name' PASSWORD = 'password';
  ```

  For detailed information about adding an instance to a MySQL replication group that uses binary log file position based replication, refer to Section 17.1.2, “Setting Up Binary Log File Position Based Replication”.

5.6.7.7 Directories and Files Created During a Cloning Operation

When data is cloned, the following directories and files are created for internal use. They should not be modified.

- `#clone`: Contains internal clone files used by the cloning operation. Created in the directory that data is cloned to.
- `#ib_archive`: Contains internally archived log files, archived on the donor during the cloning operation.
- `*.#clone` files: Temporary data files created on the recipient while the existing data directory is replaced by a remote cloning operation.

5.6.7.8 Remote Cloning Operation Failure Handling

This section describes failure handing at different stages of a cloning operation.

1. Prerequisites are checked (see Remote Cloning Prerequisites).

- If a failure occurs during the prerequisite check, the `CLONE INSTANCE` operation reports an error.

2. A backup lock is taken to block DDL operations.

   - If the cloning operation is unable to obtain a DDL lock within the time limit specified by the `clone_ddl_timeout` variable, an error is reported.

3. User-created data (schemas, tables, tablespaces) and binary logs on the recipient are removed before data is cloned to the recipient data directory.

   - When user created data is removed from the recipient during a remote cloning operation, existing data in the recipient data directory is not saved and may be lost if a failure occurs. If the data to be replaced on the recipient is of importance, a backup should be taken before initiating a remote cloning operation.

     For informational purposes, warnings are printed to the server error log to specify when data removal starts and finishes:

     ```
     [Warning] [MY-013453] [InnoDB] Clone removing all user data for provisioning:
     Started...

     [Warning] [MY-013453] [InnoDB] Clone removing all user data for provisioning:
     Finished
     ```

     If a failure occurs while removing data, the recipient may be left with a partial set of schemas, tables, and tablespaces that existed before the cloning operation. Any time during the execution of a cloning operation or after a failure, the server is always in a consistent state.

4. Data is cloned from the donor. User-created data, dictionary metadata, and other system data are cloned.

   - If a failure occurs while cloning data, the cloning operation is rolled back and all cloned data removed. At this stage, the previously existing data on the recipient has also been removed, which leaves the recipient with no user data.

     Should this scenario occur, you can either rectify the cause of the failure and re-execute the cloning operation, or forgo the cloning operation and restore the recipient data from a backup taken before the cloning operation.

5. The server is restarted automatically (applies to remote cloning operations that do not clone to a named directory). During startup, typical server startup tasks are performed.

   - If the automatic server restart fails, you can restart the server manually to complete the cloning operation.

If a network error occurs during a cloning operation, the operation resumes if the error is resolved within five minutes. Otherwise, the operation aborts and returns an error.

5.6.7.9 Monitoring Cloning Operations

This section describes options for monitoring cloning operations.

- Monitoring Cloning Operations using Performance Schema Clone Tables
- Monitoring Cloning Operations Using Performance Schema Stage Events
- Monitoring Cloning Operations Using Performance Schema Clone Instrumentation

- The Com_clone Status Variable

Monitoring Cloning Operations using Performance Schema Clone Tables

A cloning operation may take some time to complete, depending on the amount of data and other factors related to data transfer. You can monitor the status and progress of a cloning operation on the recipient MySQL server instance using the `clone_status` and `clone_progress` Performance Schema tables.

Note

The `clone_status` and `clone_progress` Performance Schema tables can be used to monitor a cloning operation on the recipient MySQL server instance only. To monitor a cloning operation on the donor MySQL server instance, use the clone stage events, as described in Monitoring Cloning Operations Using Performance Schema Stage Events.

- The `clone_status` table provides the state of the current or last executed cloning operation. A clone operation has four possible states: `Not Started`, `In Progress`, `Completed`, and `Failed`.
- The `clone_progress` table provides progress information for the current or last executed clone operation, by stage. The stages of a cloning operation include `DROP DATA`, `FILE COPY`, `PAGE_COPY`, `REDO_COPY`, `FILE_SYNC`, `RESTART`, and `RECOVERY`.

The `SELECT` and `EXECUTE` privileges on the Performance Schema is required to access the Performance Schema clone tables.

To check the state of a cloning operation:

1. Connect to the recipient MySQL server instance.
2. Query the `clone_status` table:

```
mysql> SELECT STATE FROM performance_schema.clone_status;
+-----------+
| STATE     |
+-----------+
| Completed |
+-----------+
```

Should a failure occur during a cloning operation, you can query the `clone_status` table for error information:

```
mysql> SELECT STATE, ERROR_NO, ERROR_MESSAGE FROM performance_schema.clone_status;
+-----------+----------+---------------+
| STATE     | ERROR_NO | ERROR_MESSAGE |
+-----------+----------+---------------+
| Failed    |      xxx | "xxxxxxxxxx" |
+-----------+----------+---------------+
```

To review the details of each stage of a cloning operation:

1. Connect to the recipient MySQL server instance.
2. Query the `clone_progress` table. For example, the following query provides state and end time data for each stage of the cloning operation:

```
mysql> SELECT STAGE, STATE, END_TIME FROM performance_schema.clone_progress;
+-----------+-----------+----------------------------+
| stage     | state     | end_time                   |
+-----------+-----------+----------------------------+
| DROP DATA | Completed | 2019-01-27 22:45:43.141261 |
| FILE COPY | Completed | 2019-01-27 22:45:44.457572 |
```

```
| PAGE COPY | Completed | 2019-01-27 22:45:44.577330 |
| REDO COPY | Completed | 2019-01-27 22:45:44.679570 |
| FILE SYNC | Completed | 2019-01-27 22:45:44.918547 |
| RESTART   | Completed | 2019-01-27 22:45:48.583565 |
| RECOVERY  | Completed | 2019-01-27 22:45:49.626595 |
+-----------+-----------+----------------------------+
```

For other clone status and progress data points that you can monitor, refer to Section 26.12.16, “Performance Schema Clone Tables”.

Monitoring Cloning Operations Using Performance Schema Stage Events

A cloning operation may take some time to complete, depending on the amount of data and other factors related to data transfer. There are three stage events for monitoring the progress of a cloning operation. Each stage event reports `WORK_COMPLETED` and `WORK_ESTIMATED` values. Reported values are revised as the operation progresses.

This method of monitoring a cloning operation can be used on the donor or recipient MySQL server instance.

In order of occurrence, cloning operation stage events include:

- `stage/innodb/clone (file copy)`: Indicates progress of the file copy phase of the cloning operation. `WORK_ESTIMATED` and `WORK_COMPLETED` units are file chunks. The number of files to be transferred is known at the start of the file copy phase, and the number of chunks is estimated based on the number of files. `WORK_ESTIMATED` is set to the number of estimated file chunks. `WORK_COMPLETED` is updated after each chunk is sent.
- `stage/innodb/clone (page copy)`: Indicates progress of the page copy phase of cloning operation. `WORK_ESTIMATED` and `WORK_COMPLETED` units are pages. Once the file copy phase is completed, the number of pages to be transferred is known, and `WORK_ESTIMATED` is set to this value. `WORK_COMPLETED` is updated after each page is sent.
- `stage/innodb/clone (redo copy)`: Indicates progress of the redo copy phase of cloning operation. `WORK_ESTIMATED` and `WORK_COMPLETED` units are redo chunks. Once the page copy phase is completed, the number of redo chunks to be transferred is known, and `WORK_ESTIMATED` is set to this value. `WORK_COMPLETED` is updated after each chunk is sent.

The following example demonstrates how to enable `stage/innodb/clone%` event instruments and related consumer tables to monitor a cloning operation. For information about Performance Schema stage event instruments and related consumers, see Section 26.12.5, “Performance Schema Stage Event Tables”.

1. Enable the `stage/innodb/clone%` instruments:

```
mysql> UPDATE performance_schema.setup_instruments SET ENABLED = 'YES'
       WHERE NAME LIKE 'stage/innodb/clone%';
```

2. Enable the stage event consumer tables, which include `events_stages_current`, `events_stages_history`, and `events_stages_history_long`.

```
mysql> UPDATE performance_schema.setup_consumers SET ENABLED = 'YES'
       WHERE NAME LIKE '%stages%';
```

3. Run a cloning operation. In this example, a local data directory is cloned to a directory named `cloned_dir`.

```
```

```
mysql> CLONE LOCAL DATA DIRECTORY = '/path/to/cloned_dir';
```

4. Check the progress of the cloning operation by querying the Performance Schema `events_stages_current` table. The stage event shown differs depending on the cloning phase that is in progress. The `WORK_COMPLETED` column shows the work completed. The `WORK_ESTIMATED` column shows the work required in total.

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED FROM performance_schema.events_stages_current
       WHERE EVENT_NAME LIKE 'stage/innodb/clone%';
+-------------------------------+----------------+----------------+
| EVENT_NAME                    | WORK_COMPLETED | WORK_ESTIMATED |
+-------------------------------+----------------+----------------+
| stage/innodb/clone (redo copy) |              1 |              1 |
+-------------------------------+----------------+----------------+
```

The `events_stages_current` table returns an empty set if the cloning operation has finished. In this case, you can check the `events_stages_history` table to view event data for the completed operation. For example:

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED FROM events_stages_history
       WHERE EVENT_NAME LIKE 'stage/innodb/clone%';
+-------------------------------+----------------+----------------+
| EVENT_NAME                    | WORK_COMPLETED | WORK_ESTIMATED |
+-------------------------------+----------------+----------------+
| stage/innodb/clone (file copy) |            301 |            301 |
| stage/innodb/clone (page copy) |              0 |              0 |
| stage/innodb/clone (redo copy) |              1 |              1 |
+-------------------------------+----------------+----------------+
```

Monitoring Cloning Operations Using Performance Schema Clone Instrumentation

Performance Schema provides instrumentation for advanced performance monitoring of clone operations. To view the available clone instrumentation, issue the following query:

```
mysql> SELECT * FROM performance_schema.setup_instruments
       WHERE NAME LIKE WHERE NAME LIKE '%clone%';
+--------------------------------------------+---------+
| NAME                                       | ENABLED |
+--------------------------------------------+---------+
| wait/synch/mutex/innodb/clone_snapshot_mutex | NO      |
| wait/synch/mutex/innodb/clone_sys_mutex      | NO      |
| wait/synch/mutex/innodb/clone_task_mutex     | NO      |
| wait/io/file/innodb/innodb_clone_file        | YES     |
| stage/innodb/clone (file copy)               | YES     |
| stage/innodb/clone (redo copy)               | YES     |
| stage/innodb/clone (page copy)               | YES     |
| statement/abstract/clone                     | YES     |
| statement/clone/local                        | YES     |
| statement/clone/client                       | YES     |
| statement/clone/server                       | YES     |
| memory/innodb/clone                          | YES     |
| memory/clone/data                            | YES     |
+--------------------------------------------+---------+
```

Wait Instruments

Performance schema wait instruments track events that take time. Clone wait event instruments include:

- `wait/synch/mutex/innodb/clone_snapshot_mutex`: Tracks wait events for the clone snapshot mutex, which synchronizes access to the dynamic snapshot object (on the donor and recipient) between multiple clone threads.

- `wait/synch/mutex/innodb/clone_sys_mutex`: Tracks wait events for the clone sys mutex. There is one clone system object in a MySQL server instance. This mutex synchronizes access to the clone system object on the donor and recipient. It is acquired by clone threads and other foreground and background threads.
- `wait/synch/mutex/innodb/clone_task_mutex`: Tracks wait events for the clone task mutex, used for clone task management. The `clone_task_mutex` is acquired by clone threads.
- `wait/io/file/innodb/innodb_clone_file`: Tracks all I/O wait operations for files that clone operates on.

For information about monitoring `InnoDB` mutex waits, see Section 15.15.2, “Monitoring InnoDB Mutex Waits Using Performance Schema”. For information about monitoring wait events in general, see Section 26.12.4, “Performance Schema Wait Event Tables”.

Stage Instruments

Performance Schema stage events track steps that occur during the statement-execution process. Clone stage event instruments include:

- `stage/innodb/clone (file copy)`: Indicates progress of the file copy phase of the cloning operation.
- `stage/innodb/clone (redo copy)`: Indicates progress of the redo copy phase of cloning operation.
- `stage/innodb/clone (page copy)`: Indicates progress of the page copy phase of cloning operation.

For information about monitoring cloning operations using stage events, see Monitoring Cloning Operations Using Performance Schema Stage Events. For general information about monitoring stage events, see Section 26.12.5, “Performance Schema Stage Event Tables”.

Statement Instruments

Performance Schema statement events track statement execution. When a clone operation is initiated, the different statement types tracked by clone statement instruments may be executed in parallel. You can observe these statement events in the Performance Schema statement event tables. The number of statements that execute depends on the `clone_max_concurrency` and `clone_autotune_concurrency` settings.

Clone statement event instruments include:

- `statement/abstract/clone`: Tracks statement events for any clone operation before it is classified as a local, client, or server operation type.
- `statement/clone/local`: Tracks clone statement events for local clone operations; generated when executing a `CLONE LOCAL` statement.
- `statement/clone/client`: Tracks remote cloning statement events that occur on the recipient MySQL server instance; generated when executing a `CLONE INSTANCE` statement on the recipient.
- `statement/clone/server`: Tracks remote cloning statement events that occur on the donor MySQL server instance; generated when executing a `CLONE INSTANCE` statement on the recipient.

For information about monitoring Performance Schema statement events, see Section 26.12.6, “Performance Schema Statement Event Tables”.

Memory Instruments

Performance Schema memory instruments track memory usage. Clone memory usage instruments include:

- `memory/innodb/clone`: Tracks memory allocated by `InnoDB` for the dynamic snapshot.
- `memory/clone/data`: Tracks memory allocated by the clone plugin during a clone operation.

For information about monitoring memory usage using Performance Schema, see Section 26.12.17.10, “Memory Summary Tables”.

The Com_clone Status Variable

The `Com_clone` status variable provides a count of `CLONE` statement executions.

For more information, refer to the discussion about `Com_xxx` statement counter variables in Section 5.1.10, “Server Status Variables”.

5.6.7.10 Stopping a Cloning Operation

If necessary, you can stop a cloning operation with a `KILL QUERY` *`processlist_id`* statement.

On the recipient MySQL server instance, you can retrieve the processlist identifier (PID) for a cloning operation from the `PID` column of the `clone_status` table.

```
mysql> SELECT * FROM performance_schema.clone_status\G
*************************** 1. row ***************************
             ID: 1
            PID: 8
          STATE: In Progress
     BEGIN_TIME: 2019-07-15 11:58:36.767
       END_TIME: NULL
         SOURCE: LOCAL INSTANCE
    DESTINATION: /path/to/clone_dir/
       ERROR_NO: 0
  ERROR_MESSAGE:
    BINLOG_FILE:
BINLOG_POSITION: 0
  GTID_EXECUTED:
```

You can also retrieve the processlist identifier from the `ID` column of the `INFORMATION_SCHEMA PROCESSLIST` table, the `Id` column of `SHOW PROCESSLIST` output, or the `PROCESSLIST_ID` column of the Performance Schema `threads` table. These methods of obtaining the PID information can be used on the donor or recipient MySQL server instance.

5.6.7.11 Clone System Variable Reference

Table 5.5 Clone System Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
clone_autotune_concurrency	Yes	Yes	Yes		Global	Yes
clone_buffer_size	Yes	Yes	Yes		Global	Yes
clone_ddl_timeout	Yes	Yes	Yes		Global	Yes
clone_enable_compression	Yes	Yes	Yes		Global	Yes
clone_max_concurrency	Yes	Yes	Yes		Global	Yes
clone_max_data_bandwidth	Yes	Yes	Yes		Global	Yes
clone_max_network_bandwidth	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
clone_ssl_ca	Yes	Yes	Yes		Global	Ye
clone_ssl_cert	Yes	Yes	Yes		Global	Ye
clone_ssl_key	Yes	Yes	Yes		Global	Ye
clone_valid_donor_list	Yes	Yes	Yes		Global	Ye

5.6.7.12 Clone System Variables

This section describes the system variables that control operation of the clone plugin. If values specified at startup are incorrect, the clone plugin may fail to initialize properly and the server does not load it. In this case, the server may also produce error messages for other clone settings because it will not recognize them.

Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. They can be changed dynamically at runtime using the `SET` statement, which enables you to modify operation of the server without having to stop and restart it.

Setting a global system variable runtime value normally requires the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege. For more information, see Section 5.1.9.1, “System Variable Privileges”.

Clone variables are configured on the recipient MySQL server instance where the cloning operation is executed.

- `clone_autotune_concurrency`

Property	Value
Command-Line Format	`--clone-autotune-concurrency`
Introduced	8.0.17
System Variable	`clone_autotune_concurrency`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Enables dynamic spawning of threads for remote cloning operations. The `clone_max_concurrency` variable defines the maximum number of threads that can be spawned.

If `clone_autotune_concurrency` is disabled, `clone_max_concurrency` defines the actual number of threads spawned for a remote cloning operation.

- `clone_buffer_size`

Property	Value
Command-Line Format	`--clone-buffer-size`
Introduced	8.0.17
System Variable	`clone_buffer_size`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	4194304
Minimum Value	1048576
Maximum Value	268435456

Defines the size of the intermediate buffer used when transferring data during a cloning operation. The default value is 4 mebibytes (MiB). A larger buffer size may permit I/O device drivers to fetch data in parallel, which can improve cloning performance.

- clone_ddl_timeout

Property	Value
Command-Line Format	--clone-ddl-timeout
Introduced	8.0.17
System Variable	clone_ddl_timeout
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	300
Minimum Value	0
Maximum Value	2592000

The time in seconds to wait for a backup lock when executing a cloning operation. A cloning operation cannot run concurrently with DDL operations. A backup lock is required on the donor and recipient MySQL server instances. The cloning operation waits for current DDL operations to finish. Once backup locks are acquired, DDL operations must wait for the cloning operation to finish. A value of 0 means that no backup lock is to be taken for the cloning operation. In this case, the cloning operation fails with an error if a DDL operation is attempted concurrently.

- clone_enable_compression

Property	Value
Command-Line Format	--clone-enable-compression
Introduced	8.0.17
System Variable	clone_enable_compression
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Enables compression of data at the network layer during a remote cloning operation. Compression saves network bandwidth at the cost of CPU. Enabling compression may improve the data transfer rate.

- `clone_max_concurrency`

Property	Value
Command-Line Format	`--clone-max-concurrency`
Introduced	8.0.17
System Variable	`clone_max_concurrency`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`16`
Minimum Value	`1`
Maximum Value	`128`

Defines the maximum number of concurrent threads for a remote cloning operation. The default value is 16. A greater number of threads can improve cloning performance but also reduces the number of permitted simultaneous client connections, which can affect the performance of existing client connections.

If `clone_autotune_concurrency` is enabled (the default), `clone_max_concurrency` is the maximum number of threads that can be dynamically spawned for a remote cloning operation. If `clone_autotune_concurrency` is disabled, `clone_max_concurrency` defines the actual number of threads spawned for a remote cloning operation.

A minimum data transfer rate of 1 mebibyte (MiB) per thread is recommended for remote cloning operations. The data transfer rate for a remote cloning operation is controlled by the `clone_max_data_bandwidth` variable.

- `clone_max_data_bandwidth`

Property	Value
Command-Line Format	`--clone-max-data-bandwidth`
Introduced	8.0.17
System Variable	`clone_max_data_bandwidth`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1048576`

Defines the maximum data transfer rate in mebibytes (MiB) per second for a remote cloning operation. This variable helps manage the performance impact of a cloning operation. A limit should be set only when donor disk I/O bandwidth is saturated, affecting performance. A value of 0 means “unlimited”, which permits cloning operations to run at the highest possible data transfer rate.

The minimum data transfer rate is 1 MiB per second, per thread. For example, if there are 8 threads, the minimum transfer rate is 8 MiB per second. The `clone_max_concurrency` variable controls the maximum number threads spawned for a remote cloning operation.

The requested data transfer rate specified by `clone_max_data_bandwidth` may differ from the actual data transfer rate reported by the `DATA_SPEED` column in the `performance_schema.clone_progress` table.

- `clone_max_network_bandwidth`

Property	Value
Command-Line Format	`--clone-max-network-bandwidth`
Introduced	8.0.17
System Variable	`clone_max_network_bandwidth`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1048576`

Specifies the maximum approximate network transfer rate in mebibytes (MiB) per second for a remote cloning operation. This variable can be used to manage the performance impact of a cloning operation on network bandwidth. It should be set only when network bandwidth is saturated, affecting performance on the donor instance. A value of 0 means “unlimited”, which permits cloning at the highest possible data transfer rate over the network, providing the best performance.

- `clone_ssl_ca`

Property	Value
Command-Line Format	`--clone-ssl-ca=file_name`
Introduced	8.0.14
System Variable	`clone_ssl_ca`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	`empty string`

Specifies the path to the certificate authority (CA) file. Used to configure an encrypted connection for a remote cloning operation.

- `clone_ssl_cert`

Property	Value
Command-Line Format	`--clone-ssl-cert=file_name`
Introduced	8.0.14
System Variable	`clone_ssl_cert`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	`empty string`

Specifies the path to the public key certificate. Used to configure an encrypted connection for a remote cloning operation.

- `clone_ssl_key`

Property	Value
Command-Line Format	`--clone-ssl-key=file_name`
Introduced	8.0.14
System Variable	`clone_ssl_key`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	`empty string`

Specifies the path to the private key file. Used to configure an encrypted connection for a remote cloning operation.

- `clone_valid_donor_list`

Property	Value
Command-Line Format	`--clone-valid-donor-list=value`
Introduced	8.0.17
System Variable	`clone_valid_donor_list`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`empty string`

Defines valid donor host addresses for remote cloning operations. A comma-separated list of values is permitted in the following format: "`HOST1:PORT1,HOST2:PORT2,HOST3:PORT3`". Spaces are not permitted.

Internet Protocol version 6 (IPv6) address format is not supported. Internet Protocol version 6 (IPv6) address format is not supported. An alias to the IPv6 address can be used instead. An IPv4 address can be used as is.

5.6.7.13 Clone Plugin Limitations

The clone plugin is subject to these limitations:

- DDL, including `TRUNCATE TABLE`, is not permitted during a cloning operation. Concurrent DML is permitted.
- An instance cannot be cloned from a different MySQL server version. The donor and recipient must have the same MySQL server version. For example, you cannot clone between MySQL 5.7 and MySQL 8.0. The clone plugin is only supported in MySQL 8.0.17 and higher.
- Only a single MySQL instance can be cloned at a time. Cloning multiple MySQL instances in a single cloning operation is not supported.
- The X Protocol port specified by `mysqlx_port` is not supported for remote cloning operations (when specifying the port number of the donor MySQL server instance in a `CLONE INSTANCE` statement).
- The clone plugin does not support cloning of MySQL server configurations. The recipient MySQL server instance retains its configuration, including persisted system variable settings (see Section 5.1.9.3, “Persisted System Variables”.)
- The clone plugin does not support cloning of binary logs.
- The clone plugin only clones data stored in `InnoDB`. Other storage engine data is not cloned. `MyISAM` and `CSV` tables stored in any schema including the `sys` schema are cloned as empty tables.
- Connecting to the donor MySQL server instance through MySQL Router is not supported.
- Local cloning operations do not support cloning of general tablespaces that were created with an absolute path. A cloned tablespace file with the same path as the source tablespace file would cause a conflict.

5.7 MySQL Server User-Defined Functions

MySQL Server enables user-defined functions (UDFs) to be created and loaded into the server to extend server capabilities. Server capabilities can be implemented in whole or in part using UDFs. In addition, you can write your own UDFs.

The following sections describe how to install and uninstall UDFs, and how to determine at runtime which UDFs are installed and obtain information about them. For information about writing UDFs, see Section 29.4, “Adding New Functions to MySQL”.

5.7.1 Installing and Uninstalling User-Defined Functions

User-defined functions (UDFs) must be loaded into the server before they can be used. MySQL supports UDF loading at runtime.

To load a UDF, use the `CREATE FUNCTION` statement. For example:

```
CREATE FUNCTION metaphon
```

```
  RETURNS STRING
  SONAME 'udf_example.so';
```

The UDF file base name depends on your platform. Common suffixes are `.so` for Unix and Unix-like systems, `.dll` for Windows.

While a UDF is loaded, information about it is available from the Performance Schema `user_defined_functions` table. See Section 5.7.2, “Obtaining User-Defined Function Information”.

The statement also registers the UDF in the `mysql.func` system table to cause the server to load it on subsequent restarts. For this reason, `CREATE FUNCTION` requires the `INSERT` privilege for the `mysql` database.

To remove a UDF, use the `DROP FUNCTION` statement. For example:

```
DROP FUNCTION metaphon;
```

`CREATE FUNCTION` unloads the UDF and removes it from the `mysql.func` system table. For this reason, `DROP FUNCTION` statement requires the `DELETE` privilege for the `mysql` database. With the UDF no longer registered in the table, the server does not load the UDF automatically for subsequent restarts.

You cannnot use `CREATE FUNCTION` to reinstall a function that has previously been installed. To reinstall a function, first remove it with `DROP FUNCTION`, then install it again with `CREATE FUNCTION`. You would need to do this, for example, if you upgrade to a new version of MySQL that provides an updated implementation of the function, or you recompile a new version of a function that you have written. Otherwise, the server continues to use the old version.

If the server is started with the `--skip-grant-tables` option, it does not consult the `mysql.func` table and does not load the UDFs listed there.

5.7.2 Obtaining User-Defined Function Information

The Performance Schema `user_defined_functions` table contains information about the currently loaded user-defined functions:

```
SELECT * FROM performance_schema.user_defined_functions;
```

For more information, see Section 26.12.18.6, “The user_defined_functions Table”.

5.8 Running Multiple MySQL Instances on One Machine

In some cases, you might want to run multiple instances of MySQL on a single machine. You might want to test a new MySQL release while leaving an existing production setup undisturbed. Or you might want to give different users access to different `mysqld` servers that they manage themselves. (For example, you might be an Internet Service Provider that wants to provide independent MySQL installations for different customers.)

It is possible to use a different MySQL server binary per instance, or use the same binary for multiple instances, or any combination of the two approaches. For example, you might run a server from MySQL 5.7 and one from MySQL 8.0, to see how different versions handle a given workload. Or you might run multiple instances of the current production version, each managing a different set of databases.

Whether or not you use distinct server binaries, each instance that you run must be configured with unique values for several operating parameters. This eliminates the potential for conflict between instances.

Parameters can be set on the command line, in option files, or by setting environment variables. See Section 4.2.2, “Specifying Program Options”. To see the values used by a given instance, connect to it and execute a `SHOW VARIABLES` statement.

The primary resource managed by a MySQL instance is the data directory. Each instance should use a different data directory, the location of which is specified using the `--datadir=dir_name` option. For methods of configuring each instance with its own data directory, and warnings about the dangers of failing to do so, see Section 5.8.1, “Setting Up Multiple Data Directories”.

In addition to using different data directories, several other options must have different values for each server instance:

- `--port=port_num`

  `--port` controls the port number for TCP/IP connections. Alternatively, if the host has multiple network addresses, you can set the `bind_address` system variable to cause each server to listen to a different address.

- `--socket={file_name|pipe_name}`

  `--socket` controls the Unix socket file path on Unix or the named-pipe name on Windows. On Windows, it is necessary to specify distinct pipe names only for those servers configured to permit named-pipe connections.

- `--shared-memory-base-name=name`

  This option is used only on Windows. It designates the shared-memory name used by a Windows server to permit clients to connect using shared memory. It is necessary to specify distinct shared-memory names only for those servers configured to permit shared-memory connections.

- `--pid-file=file_name`

  This option indicates the path name of the file in which the server writes its process ID.

If you use the following log file options, their values must differ for each server:

- `--general_log_file=file_name`
- `--log-bin[=file_name]`
- `--slow_query_log_file=file_name`
- `--log-error[=file_name]`

For further discussion of log file options, see Section 5.4, “MySQL Server Logs”.

To achieve better performance, you can specify the following option differently for each server, to spread the load between several physical disks:

- `--tmpdir=dir_name`

Having different temporary directories also makes it easier to determine which MySQL server created any given temporary file.

If you have multiple MySQL installations in different locations, you can specify the base directory for each installation with the `--basedir=dir_name` option. This causes each instance to automatically use a different data directory, log files, and PID file because the default for each of those parameters is relative

to the base directory. In that case, the only other options you need to specify are the `--socket` and `--port` options. Suppose that you install different versions of MySQL using `tar` file binary distributions. These install in different locations, so you can start the server for each installation using the command `bin/mysqld_safe` under its corresponding base directory. `mysqld_safe` determines the proper `--basedir` option to pass to `mysqld`, and you need specify only the `--socket` and `--port` options to `mysqld_safe`.

As discussed in the following sections, it is possible to start additional servers by specifying appropriate command options or by setting environment variables. However, if you need to run multiple servers on a more permanent basis, it is more convenient to use option files to specify for each server those option values that must be unique to it. The `--defaults-file` option is useful for this purpose.

5.8.1 Setting Up Multiple Data Directories

Each MySQL Instance on a machine should have its own data directory. The location is specified using the `--datadir=dir_name` option.

There are different methods of setting up a data directory for a new instance:

- Create a new data directory.
- Copy an existing data directory.

The following discussion provides more detail about each method.

Warning

Normally, you should never have two servers that update data in the same databases. This may lead to unpleasant surprises if your operating system does not support fault-free system locking. If (despite this warning) you run multiple servers using the same data directory and they have logging enabled, you must use the appropriate options to specify log file names that are unique to each server. Otherwise, the servers try to log to the same files.

Even when the preceding precautions are observed, this kind of setup works only with `MyISAM` and `MERGE` tables, and not with any of the other storage engines. Also, this warning against sharing a data directory among servers always applies in an NFS environment. Permitting multiple MySQL servers to access a common data directory over NFS is a *very bad idea*. The primary problem is that NFS is the speed bottleneck. It is not meant for such use. Another risk with NFS is that you must devise a way to ensure that two or more servers do not interfere with each other. Usually NFS file locking is handled by the `lockd` daemon, but at the moment there is no platform that performs locking 100% reliably in every situation.

Create a New Data Directory

With this method, the data directory will be in the same state as when you first install MySQL. It will have the default set of MySQL accounts and no user data.

On Unix, initialize the data directory. See Section 2.10, “Postinstallation Setup and Testing”.

On Windows, the data directory is included in the MySQL distribution:

- MySQL Zip archive distributions for Windows contain an unmodified data directory. You can unpack such a distribution into a temporary location, then copy it `data` directory to where you are setting up the new instance.

- Windows MSI package installers create and set up the data directory that the installed server will use, but also create a pristine “template” data directory named `data` under the installation directory. After an installation has been performed using an MSI package, the template data directory can be copied to set up additional MySQL instances.

Copy an Existing Data Directory

With this method, any MySQL accounts or user data present in the data directory are carried over to the new data directory.

1. Stop the existing MySQL instance using the data directory. This must be a clean shutdown so that the instance flushes any pending changes to disk.
2. Copy the data directory to the location where the new data directory should be.
3. Copy the `my.cnf` or `my.ini` option file used by the existing instance. This serves as a basis for the new instance.
4. Modify the new option file so that any pathnames referring to the original data directory refer to the new data directory. Also, modify any other options that must be unique per instance, such as the TCP/IP port number and the log files. For a list of parameters that must be unique per instance, see Section 5.8, “Running Multiple MySQL Instances on One Machine”.
5. Start the new instance, telling it to use the new option file.

5.8.2 Running Multiple MySQL Instances on Windows

You can run multiple servers on Windows by starting them manually from the command line, each with appropriate operating parameters, or by installing several servers as Windows services and running them that way. General instructions for running MySQL from the command line or as a service are given in Section 2.3, “Installing MySQL on Microsoft Windows”. The following sections describe how to start each server with different values for those options that must be unique per server, such as the data directory. These options are listed in Section 5.8, “Running Multiple MySQL Instances on One Machine”.

5.8.2.1 Starting Multiple MySQL Instances at the Windows Command Line

The procedure for starting a single MySQL server manually from the command line is described in Section 2.3.4.6, “Starting MySQL from the Windows Command Line”. To start multiple servers this way, you can specify the appropriate options on the command line or in an option file. It is more convenient to place the options in an option file, but it is necessary to make sure that each server gets its own set of options. To do this, create an option file for each server and tell the server the file name with a `--defaults-file` option when you run it.

Suppose that you want to run one instance of `mysqld` on port 3307 with a data directory of `C:\mydata1`, and another instance on port 3308 with a data directory of `C:\mydata2`. Use this procedure:

1. Make sure that each data directory exists, including its own copy of the `mysql` database that contains the grant tables.
2. Create two option files. For example, create one file named `C:\my-opts1.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

   Create a second file named `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata2
port = 3308
```

3. Use the `--defaults-file` option to start each server with its own option file:

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts2.cnf
```

   Each server starts in the foreground (no new prompt appears until the server exits later), so you will need to issue those two commands in separate console windows.

To shut down the servers, connect to each using the appropriate port number:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 --host=127.0.0.1 --user=root --password shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 --host=127.0.0.1 --user=root --password shutdown
```

Servers configured as just described permit clients to connect over TCP/IP. If your version of Windows supports named pipes and you also want to permit named-pipe connections, specify options that enable the named pipe and specify its name. Each server that supports named-pipe connections must use a unique pipe name. For example, the `C:\my-opts1.cnf` file might be written like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

Modify `C:\my-opts2.cnf` similarly for use by the second server. Then start the servers as described previously.

A similar procedure applies for servers that you want to permit shared-memory connections. Enable such connections by starting the server with the `shared_memory` system variable enabled and specify a unique shared-memory name for each server by setting the `shared_memory_base_name` system variable.

5.8.2.2 Starting Multiple MySQL Instances as Windows Services

On Windows, a MySQL server can run as a Windows service. The procedures for installing, controlling, and removing a single MySQL service are described in Section 2.3.4.8, “Starting MySQL as a Windows Service”.

To set up multiple MySQL services, you must make sure that each instance uses a different service name in addition to the other parameters that must be unique per instance.

For the following instructions, suppose that you want to run the `mysqld` server from two different versions of MySQL that are installed at `C:\mysql-5.5.9` and `C:\mysql-8.0.19`, respectively. (This might be the case if you are running 5.5.9 as your production server, but also want to conduct tests using 8.0.19.)

To install MySQL as a Windows service, use the `--install` or `--install-manual` option. For information about these options, see Section 2.3.4.8, “Starting MySQL as a Windows Service”.

Based on the preceding information, you have several ways to set up multiple services. The following instructions describe some examples. Before trying any of them, shut down and remove any existing MySQL services.

- **Approach 1:** Specify the options for all services in one of the standard option files. To do this, use a different service name for each server. Suppose that you want to run the 5.5.9 `mysqld` using the service name of `mysqld1` and the 8.0.19 `mysqld` using the service name `mysqld2`. In this case, you can use the `[mysqld1]` group for 5.5.9 and the `[mysqld2]` group for 8.0.19. For example, you can set up `C:\my.cnf` like this:

  ```
  # options for mysqld1 service
  [mysqld1]
  basedir = C:/mysql-5.5.9
  port = 3307
  enable-named-pipe
  socket = mypipe1

  # options for mysqld2 service
  [mysqld2]
  basedir = C:/mysql-8.0.19
  port = 3308
  enable-named-pipe
  socket = mypipe2
  ```

  Install the services as follows, using the full server path names to ensure that Windows registers the correct executable program for each service:

  ```
  C:\> C:\mysql-5.5.9\bin\mysqld --install mysqld1
  C:\> C:\mysql-8.0.19\bin\mysqld --install mysqld2
  ```

  To start the services, use the services manager, or `NET START` or `SC START` with the appropriate service names:

  ```
  C:\> SC START mysqld1
  C:\> SC START mysqld2
  ```

  To stop the services, use the services manager, or use `NET STOP` or `SC STOP` with the appropriate service names:

  ```
  C:\> SC STOP mysqld1
  C:\> SC STOP mysqld2
  ```

- **Approach 2:** Specify options for each server in separate files and use `--defaults-file` when you install the services to tell each server what file to use. In this case, each file should list options using a `[mysqld]` group.

  With this approach, to specify options for the 5.5.9 `mysqld`, create a file `C:\my-opts1.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-5.5.9
  port = 3307
  enable-named-pipe
  socket = mypipe1
  ```

  For the 8.0.19 `mysqld`, create a file `C:\my-opts2.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-8.0.19
  port = 3308
  enable-named-pipe
  ```

```
socket = mypipe2
```

Install the services as follows (enter each command on a single line):

```
C:\> C:\mysql-5.5.9\bin\mysqld --install mysqld1
          --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql-8.0.19\bin\mysqld --install mysqld2
          --defaults-file=C:\my-opts2.cnf
```

When you install a MySQL server as a service and use a `--defaults-file` option, the service name must precede the option.

After installing the services, start and stop them the same way as in the preceding example.

To remove multiple services, use `SC DELETE mysqld_service_name` for each one. Alternatively, use `mysqld --remove` for each one, specifying a service name following the `--remove` option. If the service name is the default (`MySQL`), you can omit it when using `mysqld --remove`.

5.8.3 Running Multiple MySQL Instances on Unix

Note

The discussion here uses `mysqld_safe` to launch multiple instances of MySQL. For MySQL installation using an RPM distribution, server startup and shutdown is managed by systemd on several Linux platforms. On these platforms, `mysqld_safe` is not installed because it is unnecessary. For information about using systemd to handle multiple MySQL instances, see Section 2.5.9, “Managing MySQL Server with systemd”.

One way is to run multiple MySQL instances on Unix is to compile different servers with different default TCP/IP ports and Unix socket files so that each one listens on different network interfaces. Compiling in different base directories for each installation also results automatically in a separate, compiled-in data directory, log file, and PID file location for each server.

Assume that an existing 5.7 server is configured for the default TCP/IP port number (3306) and Unix socket file (`/tmp/mysql.sock`). To configure a new 8.0.19 server to have different operating parameters, use a `CMake` command something like this:

```
shell> cmake . -DMYSQL_TCP_PORT=port_number \
            -DMYSQL_UNIX_ADDR=file_name \
            -DCMAKE_INSTALL_PREFIX=/usr/local/mysql-8.0.19
```

Here, *`port_number`* and *`file_name`* must be different from the default TCP/IP port number and Unix socket file path name, and the `CMAKE_INSTALL_PREFIX` value should specify an installation directory different from the one under which the existing MySQL installation is located.

If you have a MySQL server listening on a given port number, you can use the following command to find out what operating parameters it is using for several important configurable variables, including the base directory and Unix socket file name:

```
shell> mysqladmin --host=host_name --port=port_number variables
```

With the information displayed by that command, you can tell what option values *not* to use when configuring an additional server.

If you specify `localhost` as the host name, `mysqladmin` defaults to using a Unix socket file connection rather than TCP/IP. To explicitly specify the connection protocol, use the `--protocol={TCP|SOCKET|PIPE|MEMORY}` option.

You need not compile a new MySQL server just to start with a different Unix socket file and TCP/IP port number. It is also possible to use the same server binary and start each invocation of it with different parameter values at runtime. One way to do so is by using command-line options:

```
shell> mysqld_safe --socket=file_name --port=port_number
```

To start a second server, provide different `--socket` and `--port` option values, and pass a `--datadir=dir_name` option to `mysqld_safe` so that the server uses a different data directory.

Alternatively, put the options for each server in a different option file, then start each server using a `--defaults-file` option that specifies the path to the appropriate option file. For example, if the option files for two server instances are named `/usr/local/mysql/my.cnf` and `/usr/local/mysql/my.cnf2`, start the servers like this: command:

```
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf2
```

Another way to achieve a similar effect is to use environment variables to set the Unix socket file name and TCP/IP port number:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> bin/mysqld --initialize --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

This is a quick way of starting a second server to use for testing. The nice thing about this method is that the environment variable settings apply to any client programs that you invoke from the same shell. Thus, connections for those clients are automatically directed to the second server.

Section 4.9, “MySQL Environment Variables”, includes a list of other environment variables you can use to affect MySQL programs.

On Unix, the `mysqld_multi` script provides another way to start multiple servers. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.

5.8.4 Using Client Programs in a Multiple-Server Environment

To connect with a client program to a MySQL server that is listening to different network interfaces from those compiled into your client, you can use one of the following methods:

- Start the client with `--host=host_name --port=port_number` to connect using TCP/IP to a remote server, with `--host=127.0.0.1 --port=port_number` to connect using TCP/IP to a local server, or with `--host=localhost --socket=file_name` to connect to a local server using a Unix socket file or a Windows named pipe.
- Start the client with `--protocol=TCP` to connect using TCP/IP, `--protocol=SOCKET` to connect using a Unix socket file, `--protocol=PIPE` to connect using a named pipe, or `--protocol=MEMORY` to connect using shared memory. For TCP/IP connections, you may also need to specify `--host` and `--port` options. For the other types of connections, you may need to specify a `--socket` option to specify a Unix socket file or Windows named-pipe name, or a `--shared-memory-base-name` option to specify the shared-memory name. Shared-memory connections are supported only on Windows.

- On Unix, set the `MYSQL_UNIX_PORT` and `MYSQL_TCP_PORT` environment variables to point to the Unix socket file and TCP/IP port number before you start your clients. If you normally use a specific socket file or port number, you can place commands to set these environment variables in your `.login` file so that they apply each time you log in. See Section 4.9, “MySQL Environment Variables”.

- Specify the default Unix socket file and TCP/IP port number in the `[client]` group of an option file. For example, you can use `C:\my.cnf` on Windows, or the `.my.cnf` file in your home directory on Unix. See Section 4.2.2.2, “Using Option Files”.

- In a C program, you can specify the socket file or port number arguments in the `mysql_real_connect()` call. You can also have the program read option files by calling `mysql_options()`. See Section 28.7.7, “C API Function Descriptions”.

- If you are using the Perl `DBD::mysql` module, you can read options from MySQL option files. For example:

  ```
  $dsn = "DBI:mysql:test;mysql_read_default_group=client;"
          . "mysql_read_default_file=/usr/local/mysql/data/my.cnf";
  $dbh = DBI->connect($dsn, $user, $password);
  ```

  See Section 28.9, “MySQL Perl API”.

  Other programming interfaces may provide similar capabilities for reading option files.

Chapter 6 Security

Table of Contents

6.1 General Security Issues 1066
6.1.1 Security Guidelines 1066
6.1.2 Keeping Passwords Secure 1068
6.1.3 Making MySQL Secure Against Attackers 1071
6.1.4 Security-Related mysqld Options and Variables 1072
6.1.5 How to Run MySQL as a Normal User 1073
6.1.6 Security Issues with LOAD DATA LOCAL 1074
6.1.7 Client Programming Security Guidelines 1075
6.2 Access Control and Account Management 1077
6.2.1 Account User Names and Passwords 1078
6.2.2 Privileges Provided by MySQL 1079
6.2.3 Grant Tables 1095
6.2.4 Specifying Account Names 1104
6.2.5 Specifying Role Names 1107
6.2.6 Access Control, Stage 1: Connection Verification 1107
6.2.7 Access Control, Stage 2: Request Verification 1110
6.2.8 Adding Accounts, Assigning Privileges, and Dropping Accounts 1112
6.2.9 Reserved Accounts 1115
6.2.10 Using Roles 1116
6.2.11 Account Categories 1123
6.2.12 Privilege Restriction Using Partial Revokes 1127
6.2.13 When Privilege Changes Take Effect 1134
6.2.14 Assigning Account Passwords 1135
6.2.15 Password Management 1136
6.2.16 Server Handling of Expired Passwords 1146
6.2.17 Pluggable Authentication 1148
6.2.18 Proxy Users 1153
6.2.19 Account Locking 1160
6.2.20 Setting Account Resource Limits 1161
6.2.21 Troubleshooting Problems Connecting to MySQL 1163
6.2.22 SQL-Based Account Activity Auditing 1168
6.3 Using Encrypted Connections 1169
6.3.1 Configuring MySQL to Use Encrypted Connections 1170
6.3.2 Encrypted Connection TLS Protocols and Ciphers 1177
6.3.3 Creating SSL and RSA Certificates and Keys 1183
6.3.4 Connecting to MySQL Remotely from Windows with SSH 1192
6.4 Security Components and Plugins 1193
6.4.1 Authentication Plugins 1194
6.4.2 The Connection-Control Plugins 1258
6.4.3 The Password Validation Component 1265
6.4.4 The MySQL Keyring 1277
6.4.5 MySQL Enterprise Audit 1312
6.4.6 The Audit Message Component 1383
6.4.7 MySQL Enterprise Firewall 1386
6.4.8 MySQL Enterprise Data Masking and De-Identification 1398
6.5 FIPS Support 1416

When thinking about security within a MySQL installation, you should consider a wide range of possible topics and how they affect the security of your MySQL server and related applications:

- General factors that affect security. These include choosing good passwords, not granting unnecessary privileges to users, ensuring application security by preventing SQL injections and data corruption, and others. See Section 6.1, “General Security Issues”.
- Security of the installation itself. The data files, log files, and the all the application files of your installation should be protected to ensure that they are not readable or writable by unauthorized parties. For more information, see Section 2.10, “Postinstallation Setup and Testing”.
- Access control and security within the database system itself, including the users and databases granted with access to the databases, views and stored programs in use within the database. For more information, see Section 6.2, “Access Control and Account Management”.
- The features offered by security-related plugins. See Section 6.4, “Security Components and Plugins”.
- Network security of MySQL and your system. The security is related to the grants for individual users, but you may also wish to restrict MySQL so that it is available only locally on the MySQL server host, or to a limited set of other hosts.
- Ensure that you have adequate and appropriate backups of your database files, configuration and log files. Also be sure that you have a recovery solution in place and test that you are able to successfully recover the information from your backups. See Chapter 7, *Backup and Recovery*.

6.1 General Security Issues

This section describes general security issues to be aware of and what you can do to make your MySQL installation more secure against attack or misuse. For information specifically about the access control system that MySQL uses for setting up user accounts and checking database access, see Section 2.10, “Postinstallation Setup and Testing”.

For answers to some questions that are often asked about MySQL Server security issues, see Section A.9, “MySQL 8.0 FAQ: Security”.

6.1.1 Security Guidelines

Anyone using MySQL on a computer connected to the Internet should read this section to avoid the most common security mistakes.

In discussing security, it is necessary to consider fully protecting the entire server host (not just the MySQL server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. We do not cover all aspects of availability and fault tolerance here.

MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that users can attempt to perform. There is also support for SSL-encrypted connections between MySQL clients and servers. Many of the concepts discussed here are not specific to MySQL at all; the same general ideas apply to almost all applications.

When running MySQL, follow these guidelines:

- **Do not ever give anyone (except MySQL `root` accounts) access to the `user` table in the `mysql` system database!** This is critical.
- Learn how the MySQL access privilege system works (see Section 6.2, “Access Control and Account Management”). Use the `GRANT` and `REVOKE` statements to control access to MySQL. Do not grant more privileges than necessary. Never grant privileges to all hosts.

Checklist:

- Try `mysql -u root`. If you are able to connect successfully to the server without being asked for a password, anyone can connect to your MySQL server as the MySQL `root` user with full privileges! Review the MySQL installation instructions, paying particular attention to the information about setting a `root` password. See Section 2.10.4, “Securing the Initial MySQL Account”.
- Use the `SHOW GRANTS` statement to check which accounts have access to what. Then use the `REVOKE` statement to remove those privileges that are not necessary.

- Do not store cleartext passwords in your database. If your computer becomes compromised, the intruder can take the full list of passwords and use them. Instead, use `SHA2()` or some other one-way hashing function and store the hash value.

  To prevent password recovery using rainbow tables, do not use these functions on a plain password; instead, choose some string to be used as a salt, and use hash(hash(password)+salt) values.

- Do not choose passwords from dictionaries. Special programs exist to break passwords. Even passwords like “xfish98” are very bad. Much better is “duag98” which contains the same word “fish” but typed one key to the left on a standard QWERTY keyboard. Another method is to use a password that is taken from the first characters of each word in a sentence (for example, “Four score and seven years ago” results in a password of “Fsasya”). The password is easy to remember and type, but difficult to guess for someone who does not know the sentence. In this case, you can additionally substitute digits for the number words to obtain the phrase “4 score and 7 years ago”, yielding the password “4sa7ya” which is even more difficult to guess.

- Invest in a firewall. This protects you from at least 50% of all types of exploits in any software. Put MySQL behind the firewall or in a demilitarized zone (DMZ).

  Checklist:

  - Try to scan your ports from the Internet using a tool such as `nmap`. MySQL uses port 3306 by default. This port should not be accessible from untrusted hosts. As a simple way to check whether your MySQL port is open, try the following command from some remote machine, where *`server_host`* is the host name or IP address of the host on which your MySQL server runs:

    ```
    shell> telnet server_host 3306
    ```

    If `telnet` hangs or the connection is refused, the port is blocked, which is how you want it to be. If you get a connection and some garbage characters, the port is open, and should be closed on your firewall or router, unless you really have a good reason to keep it open.

- Applications that access MySQL should not trust any data entered by users, and should be written using proper defensive programming techniques. See Section 6.1.7, “Client Programming Security Guidelines”.

- Do not transmit plain (unencrypted) data over the Internet. This information is accessible to everyone who has the time and ability to intercept it and use it for their own purposes. Instead, use an encrypted protocol such as SSL or SSH. MySQL supports internal SSL connections. Another technique is to use SSH port-forwarding to create an encrypted (and compressed) tunnel for the communication.

- Learn to use the `tcpdump` and `strings` utilities. In most cases, you can check whether MySQL data streams are unencrypted by issuing a command like the following:

  ```
  shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
  ```

This works under Linux and should work with small modifications under other systems.

Warning

If you do not see cleartext data, this does not always mean that the information actually is encrypted. If you need high security, consult with a security expert.

6.1.2 Keeping Passwords Secure

Passwords occur in several contexts within MySQL. The following sections provide guidelines that enable end users and administrators to keep these passwords secure and avoid exposing them. In addition, the `validate_password` plugin can be used to enforce a policy on acceptable password. See Section 6.4.3, "The Password Validation Component".

6.1.2.1 End-User Guidelines for Password Security

MySQL users should use the following guidelines to keep passwords secure.

When you run a client program to connect to the MySQL server, it is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed here, along with an assessment of the risks of each method. In short, the safest methods are to have the client program prompt for the password or to specify the password in a properly protected option file.

- Use the `mysql_config_editor` utility, which enables you to store authentication credentials in an encrypted login path file named `.mylogin.cnf`. The file can be read later by MySQL client programs to obtain authentication credentials for connecting to MySQL Server. See Section 4.6.7, "`mysql_config_editor` — MySQL Configuration Utility".

- Use a `--password=`*`password`* or `-p`*`password`* option on the command line. For example:

```
shell> mysql -u francis -pfrank db_name
```

Warning

This is convenient *but insecure*. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

If your operating environment is set up to display your current command in the title bar of your terminal window, the password remains visible as long as the command is running, even if the command has scrolled out of view in the window content area.

- Use the `--password` or `-p` option on the command line with no password value specified. In this case, the client program solicits the password interactively:

```
shell> mysql -u francis -p db_name
Enter password: ********
```

The `*` characters indicate where you enter your password. The password is not displayed as you enter it.

It is more secure to enter your password this way than to specify it on the command line because it is not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs noninteractively, there is no opportunity to enter the password from the keyboard. On some systems, you may even find that the first line of your script is read and interpreted (incorrectly) as your password.

- Store your password in an option file. For example, on Unix, you can list your password in the `[client]` section of the `.my.cnf` file in your home directory:

```
[client]
password=password
```

To keep the password safe, the file should not be accessible to anyone but yourself. To ensure this, set the file access mode to `400` or `600`. For example:

```
shell> chmod 600 .my.cnf
```

To name from the command line a specific option file containing the password, use the `--defaults-file=file_name` option, where `file_name` is the full path name to the file. For example:

```
shell> mysql --defaults-file=/home/francis/mysql-opts
```

Section 4.2.2.2, “Using Option Files”, discusses option files in more detail.

On Unix, the `mysql` client writes a record of executed statements to a history file (see Section 4.5.1.3, “mysql Client Logging”). By default, this file is named `.mysql_history` and is created in your home directory. Passwords can be written as plain text in SQL statements such as `CREATE USER` and `ALTER USER`, so if you use these statements, they are logged in the history file. To keep this file safe, use a restrictive access mode, the same way as described earlier for the `.my.cnf` file.

If your command interpreter is configured to maintain a history, any file in which the commands are saved will contain MySQL passwords entered on the command line. For example, `bash` uses `~/.bash_history`. Any such file should have a restrictive access mode.

6.1.2.2 Administrator Guidelines for Password Security

Database administrators should use the following guidelines to keep passwords secure.

MySQL stores passwords for user accounts in the `mysql.user` system table. Access to this table should never be granted to any nonadministrative accounts.

Account passwords can be expired so that users must reset them. See Section 6.2.15, “Password Management”, and Section 6.2.16, “Server Handling of Expired Passwords”.

The `validate_password` plugin can be used to enforce a policy on acceptable password. See Section 6.4.3, “The Password Validation Component”.

A user who has access to modify the plugin directory (the value of the `plugin_dir` system variable) or the `my.cnf` file that specifies the plugin directory location can replace plugins and modify the capabilities provided by plugins, including authentication plugins.

Files such as log files to which passwords might be written should be protected. See Section 6.1.2.3, “Passwords and Logging”.

6.1.2.3 Passwords and Logging

Passwords can be written as plain text in SQL statements such as `CREATE USER`, `GRANT` and `SET PASSWORD`. If such statements are logged by the MySQL server as written, passwords in them become visible to anyone with access to the logs.

Statement logging avoids writing passwords as cleartext for the following statements:

```
CREATE USER ... IDENTIFIED BY ...
ALTER USER ... IDENTIFIED BY ...
SET PASSWORD ...
SLAVE START ... PASSWORD = ...
CREATE SERVER ... OPTIONS(... PASSWORD ...)
ALTER SERVER ... OPTIONS(... PASSWORD ...)
```

Passwords in those statements are rewritten to not appear literally in statement text written to the general query log, slow query log, and binary log. Rewriting does not apply to other statements. In particular, `INSERT` or `UPDATE` statements for the `mysql.user` system table that refer to literal passwords are logged as is, so you should avoid such statements. (Direct modification of grant tables is discouraged, anyway.)

For the general query log, password rewriting can be suppressed by starting the server with the `--log-raw` option. For security reasons, this option is not recommended for production use. For diagnostic purposes, it may be useful to see the exact text of statements as received by the server.

By default, contents of audit log files produced by the audit log plugin are not encrypted and may contain sensitive information, such as the text of SQL statements. For security reasons, audit log files should be written to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log. See Section 6.4.5.3, “MySQL Enterprise Audit Security Considerations”.

Statements received by the server may be rewritten if a query rewrite plugin is installed (see Query Rewrite Plugins). In this case, the `--log-raw` option affects statement logging as follows:

- Without `--log-raw`, the server logs the statement returned by the query rewrite plugin. This may differ from the statement as received.
- With `--log-raw`, the server logs the original statement as received.

An implication of password rewriting is that statements that cannot be parsed (due, for example, to syntax errors) are not written to the general query log because they cannot be known to be password free. Use cases that require logging of all statements including those with errors should use the `--log-raw` option, bearing in mind that this also bypasses password rewriting.

Password rewriting occurs only when plain text passwords are expected. For statements with syntax that expect a password hash value, no rewriting occurs. If a plain text password is supplied erroneously for such syntax, the password is logged as given, without rewriting.

To guard log files against unwarranted exposure, locate them in a directory that restricts access to the server and the database administrator. If the server logs to tables in the `mysql` database, grant access to those tables only to the database administrator.

Replication slaves store the password for the replication master in the master info repository, which by default is a table in the `mysql` database named `slave_master_info`. The use of a file in the data directory for the master info repository is now deprecated, but still possible (see Section 17.2.4, “Replication Relay and Status Logs”). Ensure that the master info repository can be accessed only by the database administrator. An alternative to storing the password in the master info repository is to use the `START SLAVE` statement to specify credentials for connecting to the master.

Use a restricted access mode to protect database backups that include log tables or log files containing passwords.

6.1.3 Making MySQL Secure Against Attackers

When you connect to a MySQL server, you should use a password. The password is not transmitted as cleartext over the connection.

All other information is transferred as text, and can be read by anyone who is able to watch the connection. If the connection between the client and the server goes through an untrusted network, and you are concerned about this, you can use the compressed protocol to make traffic much more difficult to decipher. You can also use MySQL's internal SSL support to make the connection even more secure. See Section 6.3, “Using Encrypted Connections”. Alternatively, use SSH to get an encrypted TCP/IP connection between a MySQL server and a MySQL client. You can find an Open Source SSH client at http://www.openssh.org/, and a comparison of both Open Source and Commercial SSH clients at http://en.wikipedia.org/wiki/Comparison_of_SSH_clients.

To make a MySQL system secure, you should strongly consider the following suggestions:

- Require all MySQL accounts to have a password. A client program does not necessarily know the identity of the person running it. It is common for client/server applications that the user can specify any user name to the client program. For example, anyone can use the `mysql` program to connect as any other person simply by invoking it as `mysql -u other_user db_name` if `other_user` has no password. If all accounts have a password, connecting using another user's account becomes much more difficult.

  For a discussion of methods for setting passwords, see Section 6.2.14, “Assigning Account Passwords”.

- Make sure that the only Unix user account with read or write privileges in the database directories is the account that is used for running `mysqld`.

- Never run the MySQL server as the Unix `root` user. This is extremely dangerous, because any user with the `FILE` privilege is able to cause the server to create files as `root` (for example, `~root/.bashrc`). To prevent this, `mysqld` refuses to run as `root` unless that is specified explicitly using the `--user=root` option.

  `mysqld` can (and should) be run as an ordinary, unprivileged user instead. You can create a separate Unix account named `mysql` to make everything even more secure. Use this account only for administering MySQL. To start `mysqld` as a different Unix user, add a `user` option that specifies the user name in the `[mysqld]` group of the `my.cnf` option file where you specify server options. For example:

  ```
  [mysqld]
  user=mysql
  ```

  This causes the server to start as the designated user whether you start it manually or by using `mysqld_safe` or `mysql.server`. For more details, see Section 6.1.5, “How to Run MySQL as a Normal User”.

  Running `mysqld` as a Unix user other than `root` does not mean that you need to change the `root` user name in the `user` table. *User names for MySQL accounts have nothing to do with user names for Unix accounts*.

- Do not grant the `FILE` privilege to nonadministrative users. Any user that has this privilege can write a file anywhere in the file system with the privileges of the `mysqld` daemon. This includes the server's data directory containing the files that implement the privilege tables. To make `FILE`-privilege operations

a bit safer, files generated with `SELECT ... INTO OUTFILE` do not overwrite existing files and are writable by everyone.

The `FILE` privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file into a database table. This could be abused, for example, by using `LOAD DATA` to load `/etc/passwd` into a table, which then can be displayed with `SELECT`.

To limit the location in which files can be read and written, set the `secure_file_priv` system to a specific directory. See Section 5.1.8, “Server System Variables”.

- Encrypt binary log files and relay log files. Encryption helps to protect these files and the potentially sensitive data contained in them from being misused by outside attackers, and also from unauthorized viewing by users of the operating system where they are stored. You enable encryption on a MySQL server by setting the `binlog_encryption` system variable to `ON`. For more information, see Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”.

- Do not grant the `PROCESS` or `SUPER` privilege to nonadministrative users. The output of `mysqladmin processlist` and `SHOW PROCESSLIST` shows the text of any statements currently being executed, so any user who is permitted to see the server process list might be able to see statements issued by other users.

  `mysqld` reserves an extra connection for users who have the `CONNECTION_ADMIN` or `SUPER` privilege, so that a MySQL `root` user can log in and check server activity even if all normal connections are in use.

  The `SUPER` privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

- Do not permit the use of symlinks to tables. (This capability can be disabled with the `--skip-symbolic-links` option.) This is especially important if you run `mysqld` as `root`, because anyone that has write access to the server's data directory then could delete any file in the system! See Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”.

- Stored programs and views should be written using the security guidelines discussed in Section 24.6, “Stored Object Access Control”.

- If you do not trust your DNS, you should use IP addresses rather than host names in the grant tables. In any case, you should be very careful about creating grant table entries using host name values that contain wildcards.

- If you want to restrict the number of connections permitted to a single account, you can do so by setting the `max_user_connections` variable in `mysqld`. The `CREATE USER` and `ALTER USER` statements also support resource control options for limiting the extent of server use permitted to an account. See Section 13.7.1.3, “CREATE USER Syntax”, and Section 13.7.1.1, “ALTER USER Syntax”.

- If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `secure_file_priv` to a directory where `SELECT` writes can be made safely.

6.1.4 Security-Related mysqld Options and Variables

The following table shows `mysqld` options and system variables that affect security. For descriptions of each of these, see Section 5.1.7, “Server Command Options”, and Section 5.1.8, “Server System Variables”.

Table 6.1 Security Option and Variable Summary

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
allow-suspicious-udfs	Yes	Yes				
automatic_sp_privileges	Yes	Yes	Yes		Global	Ye
chroot	Yes	Yes				
des-key-file	Yes	Yes				
local_infile	Yes	Yes	Yes		Global	Ye
old_passwords	Yes	Yes	Yes		Both	Ye
safe-user-create	Yes	Yes				
secure-auth	Yes	Yes			Global	Ye
- *Variable*: secure_auth			Yes		Global	Ye
secure_file_priv	Yes	Yes	Yes		Global	No
skip-grant-tables	Yes	Yes				
skip-name-resolve	Yes	Yes			Global	No
- *Variable*: skip_name_resolve			Yes		Global	No
skip-networking	Yes	Yes			Global	No
- *Variable*: skip_networking			Yes		Global	No
skip-show-database	Yes	Yes			Global	No
- *Variable*: skip_show_database			Yes		Global	No

6.1.5 How to Run MySQL as a Normal User

On Windows, you can run the server as a Windows service using a normal user account.

On Linux, for installations performed using a MySQL repository or RPM packages, the MySQL server `mysqld` should be started by the local `mysql` operating system user. Starting by another operating system user is not supported by the init scripts that are included as part of the MySQL repositories.

On Unix (or Linux for installations performed using `tar.gz` packages) , the MySQL server `mysqld` can be started and run by any user. However, you should avoid running the server as the Unix `root` user for security reasons. To change `mysqld` to run as a normal unprivileged Unix user *`user_name`*, you must do the following:

1. Stop the server if it is running (use `mysqladmin shutdown`).
2. Change the database directories and files so that *`user_name`* has privileges to read and write files in them (you might need to do this as the Unix `root` user):

   ```
   shell> chown -R user_name /path/to/mysql/datadir
   ```

   If you do not do this, the server will not be able to access databases or tables when it runs as *`user_name`*.

If directories or files within the MySQL data directory are symbolic links, `chown -R` might not follow symbolic links for you. If it does not, you will also need to follow those links and change the directories and files they point to.

3. Start the server as user *`user_name`*. Another alternative is to start `mysqld` as the Unix `root` user and use the `--user=`*`user_name`* option. `mysqld` starts, then switches to run as the Unix user *`user_name`* before accepting any connections.

4. To start the server as the given user automatically at system startup time, specify the user name by adding a `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file or the `my.cnf` option file in the server's data directory. For example:

```
[mysqld]
user=user_name
```

If your Unix machine itself is not secured, you should assign passwords to the MySQL `root` account in the grant tables. Otherwise, any user with a login account on that machine can run the `mysql` client with a `--user=root` option and perform any operation. (It is a good idea to assign passwords to MySQL accounts in any case, but especially so when other login accounts exist on the server host.) See Section 2.10.4, “Securing the Initial MySQL Account”.

6.1.6 Security Issues with LOAD DATA LOCAL

The `LOAD DATA` statement can load a file located on the server host, or, if the `LOCAL` keyword is specified, on the client host.

There are two potential security issues with the `LOCAL` version of `LOAD DATA`:

- The transfer of the file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the `LOAD DATA` statement. Such a server could access any file on the client host to which the client user has read access. (A patched server could in fact reply with a file-transfer request to any statement, not just `LOAD DATA LOCAL`, so a more fundamental issue is that clients should not connect to untrusted servers.)

- In a Web environment where the clients are connecting from a Web server, a user could use `LOAD DATA LOCAL` to read any files that the Web server process has read access to (assuming that a user could run any statement against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not a remote program being run by users who connect to the Web server.

To avoid `LOAD DATA` issues, clients should avoid using `LOCAL`. To avoid connecting to untrusted servers, clients can establish a secure connection and verify the server identity by connecting using the `--ssl-mode=VERIFY_IDENTITY` option and the appropriate CA certificate.

To enable adminstrators and applications to manage the local data loading capability, `LOCAL` configuration works like this:

- On the server side:

  - The `local_infile` system variable controls server-side `LOCAL` capability. Depending on the `local_infile` setting, the server refuses or permits local data loading by clients that have `LOCAL` enabled on the client side. By default, `local_infile` is disabled.

  - To explicitly cause the server to refuse or permit `LOAD DATA LOCAL` statements (regardless of how client programs and libraries are configured at build time or runtime), start `mysqld` with `local_infile` disabled or enabled, respectively. `local_infile` can also be set at runtime.

- On the client side:

  - The `ENABLED_LOCAL_INFILE` `CMake` option controls the compiled-in default `LOCAL` capability for the MySQL client library. Clients that make no explicit arrangements therefore have `LOCAL` capability disabled or enabled according to the `ENABLED_LOCAL_INFILE` setting specified at MySQL build time.

    By default, the client library in MySQL binary distributions is compiled with `ENABLED_LOCAL_INFILE` disabled. If you compile MySQL from source, configure it with `ENABLED_LOCAL_INFILE` disabled or enabled based on whether clients that make no explicit arrangements should have `LOCAL` capability disabled or enabled, respectively.

  - Client programs that use the C API can control load data loading explicitly by invoking `mysql_options()` to disable or enable the `MYSQL_OPT_LOCAL_INFILE` option. See Section 28.7.7.50, “mysql_options()”.

  - For the `mysql` client, local data loading is disabled by default. To disable or enable it explicitly, use the `--local-infile=0` or `--local-infile[=1]` option.

  - For the `mysqlimport` client, local data loading is disabled by default. To disable or enable it explicitly, use the `--local=0` or `--local[=1]` option.

  - If you use `LOAD DATA LOCAL` in Perl scripts or other programs that read the `[client]` group from option files, you can add an `local-infile` option setting to that group. To prevent problems for programs that do not understand this option, specify it using the `loose-` prefix:

    ```
    [client]
    loose-local-infile=0
    ```

    or:

    ```
    [client]
    loose-local-infile=1
    ```

  - In all cases, successful use of a `LOCAL` load operation by a client also requires that the server permits it.

If `LOCAL` capability is disabled, on either the server or client side, a client that attempts to issue a `LOAD DATA LOCAL` statement receives the following error message:

```
ERROR 1148: The used command is not allowed with this MySQL version
```

6.1.7 Client Programming Security Guidelines

Applications that access MySQL should not trust any data entered by users, who can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like `; DROP DATABASE mysql;`. This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques, if you do not prepare for them.

A common mistake is to protect only string data values. Remember to check numeric data as well. If an application generates a query such as `SELECT * FROM table WHERE ID=234` when a user enters the value `234`, the user can enter the value `234 OR 1=1` to cause the application to generate the query `SELECT * FROM table WHERE ID=234 OR 1=1`. As a result, the server retrieves every row in the table. This exposes every row and causes excessive server load. The simplest way to protect from this type of attack is to use single quotation marks around the numeric constants: `SELECT * FROM table WHERE ID='234'`. If the user enters extra information, it all becomes part of the string. In a numeric context, MySQL automatically converts this string to a number and strips any trailing nonnumeric characters from it.

Sometimes people think that if a database contains only publicly available data, it need not be protected. This is incorrect. Even if it is permissible to display any row in the database, you should still protect against denial of service attacks (for example, those that are based on the technique in the preceding paragraph that causes the server to waste resources). Otherwise, your server becomes unresponsive to legitimate users.

Checklist:

- Enable strict SQL mode to tell the server to be more restrictive of what data values it accepts. See Section 5.1.11, “Server SQL Modes”.
- Try to enter single and double quotation marks (`'` and `"`) in all of your Web forms. If you get any kind of MySQL error, investigate the problem right away.
- Try to modify dynamic URLs by adding `%22` (`"`), `%23` (`#`), and `%27` (`'`) to them.
- Try to modify data types in dynamic URLs from numeric to character types using the characters shown in the previous examples. Your application should be safe against these and similar attacks.
- Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous!
- Check the size of data before passing it to MySQL.
- Have your application connect to the database using a user name different from the one you use for administrative purposes. Do not give your applications any access privileges they do not need.

Many application programming interfaces provide a means of escaping special characters in data values. Properly used, this prevents application users from entering values that cause the application to generate statements that have a different effect than you intend:

- MySQL C API: Use the `mysql_real_escape_string_quote()` API call.
- MySQL++: Use the `escape` and `quote` modifiers for query streams.
- PHP: Use either the `mysqli` or `pdo_mysql` extensions, and not the older `ext/mysql` extension. The preferred API's support the improved MySQL authentication protocol and passwords, as well as prepared statements with placeholders. See also Choosing an API.

  If the older `ext/mysql` extension must be used, then for escaping use the `mysql_real_escape_string_quote()` function and not `mysql_escape_string()` or `addslashes()` because only `mysql_real_escape_string_quote()` is character set-aware; the other functions can be “bypassed” when using (invalid) multibyte character sets.
- Perl DBI: Use placeholders or the `quote()` method.
- Ruby DBI: Use placeholders or the `quote()` method.

- Java JDBC: Use a `PreparedStatement` object and placeholders.

Other programming interfaces might have similar capabilities.

6.2 Access Control and Account Management

MySQL enables the creation of accounts that permit client users to connect to the server and access data managed by the server. The primary function of the MySQL privilege system is to authenticate a user who connects from a given host and to associate that user with privileges on a database such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. Additional functionality includes the ability to grant privileges for administrative operations.

To control which users can connect, each account can be assigned authentication credentials such as a password. The user interface to MySQL accounts consists of SQL statements such as `CREATE USER`, `GRANT`, and `REVOKE`. See Section 13.7.1, “Account Management Statements”.

The MySQL privilege system ensures that all users may perform only the operations permitted to them. As a user, when you connect to a MySQL server, your identity is determined by *the host from which you connect* and *the user name you specify*. When you issue requests after connecting, the system grants privileges according to your identity and *what you want to do*.

MySQL considers both your host name and user name in identifying you because there is no reason to assume that a given user name belongs to the same person on all hosts. For example, the user `joe` who connects from `office.example.com` need not be the same person as the user `joe` who connects from `home.example.com`. MySQL handles this by enabling you to distinguish users on different hosts that happen to have the same name: You can grant one set of privileges for connections by `joe` from `office.example.com`, and a different set of privileges for connections by `joe` from `home.example.com`. To see what privileges a given account has, use the `SHOW GRANTS` statement. For example:

```
SHOW GRANTS FOR 'joe'@'office.example.com';
SHOW GRANTS FOR 'joe'@'home.example.com';
```

Internally, the server stores privilege information in the grant tables of the `mysql` system database. The MySQL server reads the contents of these tables into memory when it starts and bases access-control decisions on the in-memory copies of the grant tables.

MySQL access control involves two stages when you run a client program that connects to the server:

Stage 1: The server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password.

Stage 2: Assuming that you can connect, the server checks each statement you issue to determine whether you have sufficient privileges to perform it. For example, if you try to select rows from a table in a database or drop a table from the database, the server verifies that you have the `SELECT` privilege for the table or the `DROP` privilege for the database.

For a more detailed description of what happens during each stage, see Section 6.2.6, “Access Control, Stage 1: Connection Verification”, and Section 6.2.7, “Access Control, Stage 2: Request Verification”. For help in diagnosing privilege-related problems, see Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”.

If your privileges are changed (either by yourself or someone else) while you are connected, those changes do not necessarily take effect immediately for the next statement that you issue. For details about the conditions under which the server reloads the grant tables, see Section 6.2.13, “When Privilege Changes Take Effect”.

There are some things that you cannot do with the MySQL privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.
- A password applies globally to an account. You cannot associate a password with a specific object such as a database, table, or routine.

6.2.1 Account User Names and Passwords

MySQL stores accounts in the `user` table of the `mysql` system database. An account is defined in terms of a user name and the client host or hosts from which the user can connect to the server. For information about account representation in the `user` table, see Section 6.2.3, “Grant Tables”.

An account may also have authentication credentials such as a password. The credentials are handled by the account authentication plugin. MySQL supports multiple authentication plugins. Some of them use built-in authentication methods, whereas others enable authentication using external authentication methods. See Section 6.2.17, “Pluggable Authentication”.

There are several distinctions between the way user names and passwords are used by MySQL and your operating system:

- User names, as used by MySQL for authentication purposes, have nothing to do with user names (login names) as used by Windows or Unix. On Unix, most MySQL clients by default try to log in using the current Unix user name as the MySQL user name, but that is for convenience only. The default can be overridden easily, because client programs permit any user name to be specified with a `-u` or `--user` option. This means that anyone can attempt to connect to the server using any user name, so you cannot make a database secure in any way unless all MySQL accounts have passwords. Anyone who specifies a user name for an account that has no password can connect successfully to the server.
- MySQL user names are up to 32 characters long. Operating system user names may have a different maximum length.

Warning

The MySQL user name length limit is hardcoded in MySQL servers and clients, and trying to circumvent it by modifying the definitions of the tables in the `mysql` database *does not work*.

You should never alter the structure of tables in the `mysql` database in any manner whatsoever except by means of the procedure that is described in Section 2.11, “Upgrading MySQL”. Attempting to redefine MySQL's system tables in any other fashion results in undefined and unsupported behavior. The server is free to ignore rows that become malformed as a result of such modifications.

- To authenticate client connections for accounts that use built-in authentication methods, the server uses passwords stored in the `user` table. These passwords are distinct from passwords for logging in to your operating system. There is no necessary connection between the “external” password you use to log in to a Windows or Unix machine and the password you use to access the MySQL server on that machine.

  If the server authenticates a client using some other plugin, the authentication method that the plugin implements may or may not use a password stored in the `user` table. In this case, it is possible that an external password is also used to authenticate to the MySQL server.

- Passwords stored in the `user` table are encrypted using plugin-specific algorithms.
- If the user name and password contain only ASCII characters, it is possible to connect to the server regardless of character set settings. To enable connections when the user name or password contain non-ASCII characters, client applications should call the `mysql_options()` C API function with the `MYSQL_SET_CHARSET_NAME` option and appropriate character set name as arguments. This causes authentication to take place using the specified character set. Otherwise, authentication fails unless the server default character set is the same as the encoding in the authentication defaults.

  Standard MySQL client programs support a `--default-character-set` option that causes `mysql_options()` to be called as just described. In addition, character set autodetection is supported as described in Section 10.4, “Connection Character Sets and Collations”. For programs that use a connector that is not based on the C API, the connector may provide an equivalent to `mysql_options()` that can be used instead. Check the connector documentation.

  The preceding notes do not apply for `ucs2`, `utf16`, and `utf32`, which are not permitted as client character sets.

The MySQL installation process populates the grant tables with an initial `root` account, as described in Section 2.10.4, “Securing the Initial MySQL Account”, which also discusses how to assign a password to it. Thereafter, you normally set up, modify, and remove MySQL accounts using statements such as `CREATE USER`, `DROP USER`, `GRANT`, and `REVOKE`. See Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”, and Section 13.7.1, “Account Management Statements”.

To connect to a MySQL server with a command-line client, specify user name and password options as necessary for the account that you want to use:

```
shell> mysql --user=finley --password db_name
```

If you prefer short options, the command looks like this:

```
shell> mysql -u finley -p db_name
```

If you omit the password value following the `--password` or `-p` option on the command line (as just shown), the client prompts for one. Alternatively, the password can be specified on the command line:

```
shell> mysql --user=finley --password=password db_name
shell> mysql -u finley -ppassword db_name
```

If you use the `-p` option, there must be *no space* between `-p` and the following password value.

Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. To avoid giving the password on the command line, use an option file or a login path file. See Section 4.2.2.2, “Using Option Files”, and Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”.

For additional information about specifying user names, passwords, and other connection parameters, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

6.2.2 Privileges Provided by MySQL

The privileges granted to a MySQL account determine which operations the account can perform. MySQL privileges differ in the contexts in which they apply and at different levels of operation:

- Administrative privileges enable users to manage operation of the MySQL server. These privileges are global because they are not specific to a particular database.

- Database privileges apply to a database and to all objects within it. These privileges can be granted for specific databases, or globally so that they apply to all databases.
- Privileges for database objects such as tables, indexes, views, and stored routines can be granted for specific objects within a database, for all objects of a given type within a database (for example, all tables in a database), or globally for all objects of a given type in all databases.

Privileges also differ in terms of whether they are static (built in to the server) or dynamic (defined at runtime). Whether a privilege is static or dynamic affects its availability to be granted to user accounts and roles. For information about the differences between static and dynamic privileges, see Static Versus Dynamic Privileges.)

Information about account privileges is stored in the grant tables in the `mysql` system database. For a description of the structure and contents of these tables, see Section 6.2.3, “Grant Tables”. The MySQL server reads the contents of the grant tables into memory when it starts, and reloads them under the circumstances indicated in Section 6.2.13, “When Privilege Changes Take Effect”. The server bases access-control decisions on the in-memory copies of the grant tables.

Important

Some MySQL releases introduce changes to the grant tables to add new privileges or features. To make sure that you can take advantage of any new capabilities, update your grant tables to the current structure whenever you upgrade MySQL. See Section 2.11, “Upgrading MySQL”.

The following sections summarize the available privileges, provide more detailed descriptions of each privilege, and offer usage guidelines.

- Summary of Available Privileges
- Static Privilege Descriptions
- Dynamic Privilege Descriptions
- Privilege-Granting Guidelines
- Static Versus Dynamic Privileges
- Migrating Accounts from SUPER to Dynamic Privileges

Summary of Available Privileges

The following table shows the static privilege names used in `GRANT` and `REVOKE` statements, along with the column name associated with each privilege in the grant tables and the context in which the privilege applies.

Table 6.2 Permissible Static Privileges for GRANT and REVOKE

Privilege	Grant Table Column	Context
`ALL [PRIVILEGES]`	Synonym for “all privileges”	Server administration
`ALTER`	`Alter_priv`	Tables
`ALTER ROUTINE`	`Alter_routine_priv`	Stored routines
`CREATE`	`Create_priv`	Databases, tables, or indexes
`CREATE ROLE`	`Create_role_priv`	Server administration
`CREATE ROUTINE`	`Create_routine_priv`	Stored routines

Privilege	Grant Table Column	Context
CREATE TABLESPACE	Create_tablespace_priv	Server administration
CREATE TEMPORARY TABLES	Create_tmp_table_priv	Tables
CREATE USER	Create_user_priv	Server administration
CREATE VIEW	Create_view_priv	Views
DELETE	Delete_priv	Tables
DROP	Drop_priv	Databases, tables, or views
DROP ROLE	Drop_role_priv	Server administration
EVENT	Event_priv	Databases
EXECUTE	Execute_priv	Stored routines
FILE	File_priv	File access on server host
GRANT OPTION	Grant_priv	Databases, tables, or stored routines
INDEX	Index_priv	Tables
INSERT	Insert_priv	Tables or columns
LOCK TABLES	Lock_tables_priv	Databases
PROCESS	Process_priv	Server administration
PROXY	See proxies_priv table	Server administration
REFERENCES	References_priv	Databases or tables
RELOAD	Reload_priv	Server administration
REPLICATION CLIENT	Repl_client_priv	Server administration
REPLICATION SLAVE	Repl_slave_priv	Server administration
SELECT	Select_priv	Tables or columns
SHOW DATABASES	Show_db_priv	Server administration
SHOW VIEW	Show_view_priv	Views
SHUTDOWN	Shutdown_priv	Server administration
SUPER	Super_priv	Server administration
TRIGGER	Trigger_priv	Tables
UPDATE	Update_priv	Tables or columns
USAGE	Synonym for “no privileges”	Server administration

The following table shows the dynamic privilege names used in GRANT and REVOKE statements, along with the context in which the privilege applies.

Table 6.3 Permissible Dynamic Privileges for GRANT and REVOKE

Privilege	Context
APPLICATION_PASSWORD_ADMIN	Dual password administration
AUDIT_ADMIN	Audit log administration
BACKUP_ADMIN	Backup administration
BINLOG_ADMIN	Backup and Replication administration
BINLOG_ENCRYPTION_ADMIN	Backup and Replication administration

Privilege	Context
CLONE_ADMIN	Clone administration
CONNECTION_ADMIN	Server administration
ENCRYPTION_KEY_ADMIN	Server administration
FIREWALL_ADMIN	Firewall administration
FIREWALL_USER	Firewall administration
GROUP_REPLICATION_ADMIN	Replication administration
INNODB_REDO_LOG_ARCHIVE	Redo log archiving administration
NDB_STORED_USER	NDB Cluster
PERSIST_RO_VARIABLES_ADMIN	Server administration
REPLICATION_APPLIER	PRIVILEGE_CHECKS_USER for a replication channel
REPLICATION_SLAVE_ADMIN	Replication administration
RESOURCE_GROUP_ADMIN	Resource group administration
RESOURCE_GROUP_USER	Resource group administration
ROLE_ADMIN	Server administration
SESSION_VARIABLES_ADMIN	Server administration
SET_USER_ID	Server administration
SYSTEM_USER	Server administration
SYSTEM_VARIABLES_ADMIN	Server administration
TABLE_ENCRYPTION_ADMIN	Server administration
VERSION_TOKEN_ADMIN	Server administration
XA_RECOVER_ADMIN	Server administration

Static Privilege Descriptions

Static privileges are built in to the server, in contrast to dynamic privileges, which are defined at runtime. The following list describes each static privilege available in MySQL.

Particular SQL statements might have more specific privilege requirements than indicated here. If so, the description for the statement in question provides the details.

- ALL, ALL PRIVILEGES

  These privilege specifiers are shorthand for “all privileges available at a given privilege level” (except GRANT OPTION). For example, granting ALL at the global or table level grants all global privileges or all table-level privileges, respectively.

- ALTER

  Enables use of the ALTER TABLE statement to change the structure of tables. ALTER TABLE also requires the CREATE and INSERT privileges. Renaming a table requires ALTER and DROP on the old table, CREATE, and INSERT on the new table.

- ALTER ROUTINE

  Enables use of statements that alter or drop stored routines (stored procedures and functions).

- `CREATE`

  Enables use of statements that create new databases and tables.

- `CREATE ROLE`

  Enables use of the `CREATE ROLE` statement. (The `CREATE USER` privilege also enables use of the `CREATE ROLE` statement.) See Section 6.2.10, “Using Roles”.

  The `CREATE ROLE` and `DROP ROLE` privileges are not as powerful as `CREATE USER` because they can be used only to create and drop accounts. They cannot be used as `CREATE USER` can be modify account attributes or rename accounts. See User and Role Interchangeability.

- `CREATE ROUTINE`

  Enables use of statements that create stored routines (stored procedures and functions).

- `CREATE TABLESPACE`

  Enables use of statements that create, alter, or drop tablespaces and log file groups.

- `CREATE TEMPORARY TABLES`

  Enables the creation of temporary tables using the `CREATE TEMPORARY TABLE` statement.

  After a session has created a temporary table, the server performs no further privilege checks on the table. The creating session can perform any operation on the table, such as `DROP TABLE`, `INSERT`, `UPDATE`, or `SELECT`. For more information, see Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”.

- `CREATE USER`

  Enables use of the `ALTER USER`, `CREATE ROLE`, `CREATE USER`, `DROP ROLE`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES` statements.

- `CREATE VIEW`

  Enables use of the `CREATE VIEW` statement.

- `DELETE`

  Enables rows to be deleted from tables in a database.

- `DROP`

  Enables use of statements that drop (remove) existing databases, tables, and views. The `DROP` privilege is required to use the `ALTER TABLE ... DROP PARTITION` statement on a partitioned table. The `DROP` privilege is also required for `TRUNCATE TABLE`.

- `DROP ROLE`

  Enables use of the `DROP ROLE` statement. (The `CREATE USER` privilege also enables use of the `DROP ROLE` statement.) See Section 6.2.10, “Using Roles”.

  The `CREATE ROLE` and `DROP ROLE` privileges are not as powerful as `CREATE USER` because they can be used only to create and drop accounts. They cannot be used as `CREATE USER` can be modify account attributes or rename accounts. See User and Role Interchangeability.

- `EVENT`

Enables use of statements that create, alter, drop, or display events for the Event Scheduler.

- `EXECUTE`

  Enables use of statements that execute stored routines (stored procedures and functions).

- `FILE`

  Affects the following operations and server behaviors:

  - Enables reading and writing files on the server host using the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function. A user who has the `FILE` privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.)
  - Enables creating new files in any directory where the MySQL server has write access. This includes the server's data directory containing the files that implement the privilege tables.
  - Enables use of the `DATA DIRECTORY` or `INDEX DIRECTORY` table option for the `CREATE TABLE` statement.

  As a security measure, the server does not overwrite existing files.

  To limit the location in which files can be read and written, set the `secure_file_priv` system variable to a specific directory. See Section 5.1.8, “Server System Variables”.

- `GRANT OPTION`

  Enables you to grant to or revoke from other users those privileges that you yourself possess.

- `INDEX`

  Enables use of statements that create or drop (remove) indexes. `INDEX` applies to existing tables. If you have the `CREATE` privilege for a table, you can include index definitions in the `CREATE TABLE` statement.

- `INSERT`

  Enables rows to be inserted into tables in a database. `INSERT` is also required for the `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` table-maintenance statements.

- `LOCK TABLES`

  Enables use of explicit `LOCK TABLES` statements to lock tables for which you have the `SELECT` privilege. This includes use of write locks, which prevents other sessions from reading the locked table.

- `PROCESS`

  Enables display of information about the threads executing within the server (that is, information about the statements being executed by sessions). The privilege enables use of `SHOW PROCESSLIST` or `mysqladmin processlist` to see threads belonging to other accounts; you can always see your own threads. The `PROCESS` privilege also enables use of `SHOW ENGINE`.

- `PROXY`

  Enables one user to impersonate or become known as another user. See Section 6.2.18, “Proxy Users”.

- `REFERENCES`

  Creation of a foreign key constraint requires the `REFERENCES` privilege for the parent table.

- `RELOAD`

  Enables use of the `FLUSH` statement. It also enables `mysqladmin` commands that are equivalent to `FLUSH` operations: `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, `flush-tables`, `flush-threads`, `refresh`, and `reload`.

  The `reload` command tells the server to reload the grant tables into memory. `flush-privileges` is a synonym for `reload`. The `refresh` command closes and reopens the log files and flushes all tables. The other `flush-`*`xxx`* commands perform functions similar to `refresh`, but are more specific and may be preferable in some instances. For example, if you want to flush just the log files, `flush-logs` is a better choice than `refresh`.

  The `RELOAD` privilege also enables use of the `RESET MASTER` and `RESET SLAVE` statements.

- `REPLICATION CLIENT`

  Enables use of the `SHOW MASTER STATUS`, `SHOW SLAVE STATUS`, and `SHOW BINARY LOGS` statements. Grant this privilege to accounts that are used by slave servers to connect to the current server as their master.

- `REPLICATION SLAVE`

  Enables the account to request updates that have been made to databases on the master server, using the `SHOW SLAVE HOSTS`, `SHOW RELAYLOG EVENTS`, and `SHOW BINLOG EVENTS` statements. This privilege is also required to use the `mysqlbinlog` options `--read-from-remote-server` (`-R`) and `--read-from-remote-master`. Grant this privilege to accounts that are used by slave servers to connect to the current server as their master.

- `SELECT`

  Enables rows to be selected from tables in a database. `SELECT` statements require the `SELECT` privilege only if they actually access tables. Some `SELECT` statements do not access tables and can be executed without permission for any database. For example, you can use `SELECT` as a simple calculator to evaluate expressions that make no reference to tables:

  ```
  SELECT 1+1;
  SELECT PI()*2;
  ```

  The `SELECT` privilege is also needed for other statements that read column values. For example, `SELECT` is needed for columns referenced on the right hand side of *`col_name`*=*`expr`* assignment in `UPDATE` statements or for columns named in the `WHERE` clause of `DELETE` or `UPDATE` statements.

  The `SELECT` privilege is needed for tables or views used with `EXPLAIN`, including any underlying tables in view definitions.

- `SHOW DATABASES`

  Enables the account to see database names by issuing the `SHOW DATABASE` statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the `--skip-show-database` option.

Caution

Because any static global privilege is considered a privilege for all databases, any static global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`, except databases that have been restricted at the database level by partial revokes.

- `SHOW VIEW`

  Enables use of the `SHOW CREATE VIEW` statement. This privilege is also needed for views used with `EXPLAIN`.

- `SHUTDOWN`

  Enables use of the `SHUTDOWN` and `RESTART` statements, the `mysqladmin shutdown` command, and the `mysql_shutdown()` C API function.

- `SUPER`

  `SUPER` is a powerful and far-reaching privilege and should not be granted lightly. If an account needs to perform only a subset of `SUPER` operations, it may be possible to achieve the desired privilege set by instead granting one or more dynamic privileges, each of which confers more limited capabilities. See Dynamic Privilege Descriptions.

  Note

  `SUPER` is deprecated and will be removed in a future version of MySQL. See Migrating Accounts from SUPER to Dynamic Privileges.

  `SUPER` affects the following operations and server behaviors:

  - Enables system variable changes at runtime:
    - Enables server configuration changes to global system variables with `SET GLOBAL` and `SET PERSIST`.

      The corresponding dynamic privilege is `SYSTEM_VARIABLES_ADMIN`.

    - Enables setting restricted session system variables that require a special privilege.

      The corresponding dynamic privilege is `SESSION_VARIABLES_ADMIN`.

    See also Section 5.1.9.1, “System Variable Privileges”.

  - Enables changes to global transaction characteristics (see Section 13.3.7, “SET TRANSACTION Syntax”).

    The corresponding dynamic privilege is `SYSTEM_VARIABLES_ADMIN`.

  - Enables the account to start and stop replication, including Group Replication.

    The corresponding dynamic privilege is `REPLICATION_SLAVE_ADMIN` for regular replication, `GROUP_REPLICATION_ADMIN` for Group Replication.

  - Enables use of the `CHANGE MASTER TO` and `CHANGE REPLICATION FILTER` statements.

The corresponding dynamic privilege is `REPLICATION_SLAVE_ADMIN`.

- Enables binary log control by means of the `PURGE BINARY LOGS` and `BINLOG` statements.

  The corresponding dynamic privilege is `BINLOG_ADMIN`.

- Enables setting the effective authorization ID when executing a view or stored program. A user with this privilege can specify any account in the `DEFINER` attribute of a view or stored program.

  The corresponding dynamic privilege is `SET_USER_ID`.

- Enables use of the `CREATE SERVER`, `ALTER SERVER`, and `DROP SERVER` statements.

- Enables use of the `mysqladmin debug` command.

- Enables `InnoDB` encryption key rotation.

  The corresponding dynamic privilege is `ENCRYPTION_KEY_ADMIN`.

- Enables execution of Version Tokens user-defined functions.

  The corresponding dynamic privilege is `VERSION_TOKEN_ADMIN`.

- Enables granting and revoking roles, use of the `WITH ADMIN OPTION` clause of the `GRANT` statement, and nonempty `<graphml>` element content in the result from the `ROLES_GRAPHML()` function.

  The corresponding dynamic privilege is `ROLE_ADMIN`.

- Enables control over client connections not permitted to non-`SUPER` accounts:

  - Enables use of the `KILL` statement or `mysqladmin kill` command to kill threads belonging to other accounts. (An account can always kill its own threads.)

  - The server does not execute `init_connect` system variable content when `SUPER` clients connect.

  - The server accepts one connection from a `SUPER` client even if the connection limit configured by the `max_connections` system variable is reached.

  - A server in offline mode (`offline_mode` enabled) does not terminate `SUPER` client connections at the next client request, and accepts new connections from `SUPER` clients.

  - Updates can be performed even when the `read_only` system variable is enabled. This applies to explicit table updates, and to use of account-management statements such as `GRANT` and `REVOKE` that update tables implicitly.

  The corresponding dynamic privilege for the preceding connection-control operations is `CONNECTION_ADMIN`.

You may also need the `SUPER` privilege to create or alter stored functions if binary logging is enabled, as described in Section 24.7, “Stored Program Binary Logging”.

- `TRIGGER`

  Enables trigger operations. You must have this privilege for a table to create, drop, execute, or display triggers for that table.

When a trigger is activated (by a user who has privileges to execute `INSERT`, `UPDATE`, or `DELETE` statements for the table associated with the trigger), trigger execution requires that the user who defined the trigger still have the `TRIGGER` privilege for the table.

- `UPDATE`

  Enables rows to be updated in tables in a database.

- `USAGE`

  This privilege specifier stands for “no privileges.” It is used at the global level with `GRANT` to specify clauses such as `WITH GRANT OPTION` without naming specific account privileges in the privilege list. `SHOW GRANTS` displays `USAGE` to indicate that an account has no privileges at a privilege level.

Dynamic Privilege Descriptions

Dynamic privileges are defined at runtime, in contrast to static privileges, which are built in to the server. The following list describes each dynamic privilege available in MySQL.

Most dynamic privileges are defined at server startup. Others are defined by a particular server component or plugin, as indicated in the privilege descriptions. In such cases, the privilege is unavailable unless the component or plugin that defines it is enabled.

Particular SQL statements might have more specific privilege requirements than indicated here. If so, the description for the statement in question provides the details.

- `APPLICATION_PASSWORD_ADMIN` (added in MySQL 8.0.14)

  For dual-password capability, this privilege enables use of the `RETAIN CURRENT PASSWORD` and `DISCARD OLD PASSWORD` clauses for `ALTER USER` and `SET PASSWORD` statements that apply to your own account. This privilege is required to manipulate your own secondary password because most users require only one password.

  If an account is to be permitted to manipulate secondary passwords for all accounts, it should be granted the `CREATE USER` privilege rather than `APPLICATION_PASSWORD_ADMIN`.

  For more information about use of dual passwords, see Section 6.2.15, “Password Management”.

- `AUDIT_ADMIN`

  Enables audit log configuration. This privilege is defined by the `audit_log` plugin; see Section 6.4.5, “MySQL Enterprise Audit”.

- `BACKUP_ADMIN`

  Enables execution of the `LOCK INSTANCE FOR BACKUP` statement and access to the Performance Schema `log_status` table.

  The `BACKUP_ADMIN` privilege is automatically granted to users with the `RELOAD` privilege when performing an in-place upgrade to MySQL 8.0 from an earlier version.

- `BINLOG_ADMIN`

  Enables binary log control by means of the `PURGE BINARY LOGS` and `BINLOG` statements.

- `BINLOG_ENCRYPTION_ADMIN`

Enables setting the system variable `binlog_encryption`, which activates or deactivates encryption for binary log files and relay log files. This ability is not provided by the `BINLOG_ADMIN`, `SYSTEM_VARIABLES_ADMIN`, or `SESSION_VARIABLES_ADMIN` privileges. The related system variable `binlog_rotate_encryption_master_key_at_startup`, which rotates the binary log master key automatically when the server is restarted, does not require this privilege.

- `CLONE_ADMIN`

  Enables execution of the `CLONE` statements. Includes `BACKUP_ADMIN` and `SHUTDOWN` privileges.

- `CONNECTION_ADMIN`

  Enables use of the `KILL` statement or `mysqladmin kill` command to kill threads belonging to other accounts. (An account can always kill its own threads.)

  Enables setting system variables related to client connections, or circumventing restrictions related to client connections. `CONNECTION_ADMIN` applies to the effects of these system variables:

  - `init_connect`: The server does not execute `init_connect` system variable content when `CONNECTION_ADMIN` clients connect.
  - `max_connections`: The server accepts one connection from a `CONNECTION_ADMIN` client even if the connection limit configured by the `max_connections` system variable is reached.
  - `offline_mode`: A server in offline mode (`offline_mode` enabled) does not terminate `CONNECTION_ADMIN` client connections at the next client request, and accepts new connections from `CONNECTION_ADMIN` clients.
  - `read_only`: Updates can be performed even when the `read_only` system variable is enabled. This applies to explicit table updates, and to use of account-management statements such as `GRANT` and `REVOKE` that update tables implicitly.

- `ENCRYPTION_KEY_ADMIN`

  Enables `InnoDB` encryption key rotation.

- `FIREWALL_ADMIN`

  Enables a user to administer firewall rules for any user. This privilege is defined by the `MYSQL_FIREWALL` plugin; see Section 6.4.7, “MySQL Enterprise Firewall”.

- `FIREWALL_USER`

  Enables users to update their own firewall rules. This privilege is defined by the `MYSQL_FIREWALL` plugin; see Section 6.4.7, “MySQL Enterprise Firewall”.

- `GROUP_REPLICATION_ADMIN`

  Enables the account to start and stop Group Replication using the `START GROUP REPLICATION` and `STOP GROUP REPLICATION` statements, to change the global setting for the `group_replication_consistency` system variable, and to use the `group_replication_set_write_concurrency()` and `group_replication_set_communication_protocol()` UDFs. Grant this privilege to accounts that are used to administer servers that are members of a replication group.

- `INNODB_REDO_LOG_ARCHIVE`

Enables the account to activate and deactivate redo log archiving.

- `NDB_STORED_USER`

  Enables the user or role and its privileges to be shared and synchronized between all `NDB`-enabled MySQL servers as soon as they join a given NDB Cluster. This privilege is available only if the `NDB` storage engine is enabled.

  Any changes to or revocations of privileges made for the given user or role are synchronized immediately with all connected MySQL servers (SQL nodes). You should be aware that there is no guarantee that multiple statements affecting privileges originating from different SQL nodes are executed on all SQL nodes in the same order. For this reason, it is highly recommended that all user administration be done from a single designated SQL node.

  `NDB_STORED_USER` is a global privilege and must be granted or revoked using `ON *.*`. Trying to set any other scope for this privilege results in an error. This privilege can be given to most application and administrative users, but it cannot be granted to system reserved accounts such as `mysql.session@localhost` or `mysql.infoschema@localhost`.

  A user that has been granted the `NDB_STORED_USER` privilege is stored in `NDB` (and thus shared by all SQL nodes), as is a role with this privilege. A user that is merely granted a role that has `NDB_STORED_USER` is *not* stored in `NDB`; each `NDB` stored user must be granted the privilege explicitly.

  For more detailed information about how this works in `NDB`, see Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”.

  The `NDB_STORED_USER` privilege is available beginning with NDB 8.0.18.

- `PERSIST_RO_VARIABLES_ADMIN`

  For users who also have `SYSTEM_VARIABLES_ADMIN`, `PERSIST_RO_VARIABLES_ADMIN` enables use of `SET PERSIST_ONLY` to persist global system variables to the `mysqld-auto.cnf` option file in the data directory. This statement is similar to `SET PERSIST` but does not modify the runtime global system variable value. This makes `SET PERSIST_ONLY` suitable for configuring read-only system variables that can be set only at server startup.

  See also Section 5.1.9.1, “System Variable Privileges”.

- `REPLICATION_APPLIER`

  Enables the account to act as the `PRIVILEGE_CHECKS_USER` for a replication channel, and to execute `BINLOG` statements in `mysqlbinlog` output. Grant this privilege to accounts that are assigned using `CHANGE MASTER TO` to provide a security context for replication channels, and to handle replication errors on those channels. As well as the `REPLICATION_APPLIER` privilege, you must also give the account the required privileges to execute the transactions received by the replication channel or contained in the `mysqlbinlog` output, for example to update the affected tables. For more information, see Section 17.3.3, “Replication Privilege Checks”.

- `REPLICATION_SLAVE_ADMIN`

  Enables the account to connect to the master server, start and stop replication using the `START SLAVE` and `STOP SLAVE` statements, and use the `CHANGE MASTER TO` and `CHANGE REPLICATION FILTER` statements. Grant this privilege to accounts that are used by slave servers to connect to the current server as their master. This privilege does not apply to Group Replication; use `GROUP_REPLICATION_ADMIN` for that.

- `RESOURCE_GROUP_ADMIN`

  Enables resource group management, consisting of creating, altering, and dropping resource groups, and assignment of threads and statements to resource groups. A user with this privilege can perform any operation relating to resource groups.

- `RESOURCE_GROUP_USER`

  Enables assigning threads and statements to resource groups. A user with this privilege can use the `SET RESOURCE GROUP` statement and the `RESOURCE_GROUP` optimizer hint.

- `ROLE_ADMIN`

  Enables granting and revoking roles, use of the `WITH ADMIN OPTION` clause of the `GRANT` statement, and nonempty `<graphml>` element content in the result from the `ROLES_GRAPHML()` function. Required to set the value of the `mandatory_roles` system variable.

- `SERVICE_CONNECTION_ADMIN`

  Enables connections to the network interface that permits only administrative connections (see Section 8.12.4.1, “How MySQL Handles Client Connections”).

- `SESSION_VARIABLES_ADMIN` (added in MySQL 8.0.14)

  For most system variables, setting the session value requires no special privileges and can be done by any user to affect the current session. For some system variables, setting the session value can have effects outside the current session and thus is a restricted operation. For these, the `SESSION_VARIABLES_ADMIN` privilege enables the user to set the session value.

  If a system variable is restricted and requires a special privilege to set the session value, the variable description indicates that restriction. Examples include `binlog_format`, `sql_log_bin`, and `sql_log_off`.

  Prior to MySQL 8.0.14 when `SESSION_VARIABLES_ADMIN` was added, restricted session system variables can be set only by users who have the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege.

  The `SESSION_VARIABLES_ADMIN` privilege is a subset of the `SYSTEM_VARIABLES_ADMIN` and `SUPER` privileges. A user who has either of those privileges is also permitted to set restricted session variables and effectively has `SESSION_VARIABLES_ADMIN` by implication and need not be granted `SESSION_VARIABLES_ADMIN` explicitly.

  See also Section 5.1.9.1, “System Variable Privileges”.

- `SET_USER_ID`

  Enables setting the effective authorization ID when executing a view or stored program. A user with this privilege can specify any account in the `DEFINER` attribute of a view or stored program.

- `SYSTEM_USER` (added in MySQL 8.0.16)

  The `SYSTEM_USER` privilege distinguishes system users from regular users:

  - A user with the `SYSTEM_USER` privilege is a system user.
  - A user without the `SYSTEM_USER` privilege is a regular user.

  The `SYSTEM_USER` privilege has an effect on the accounts to which a given user can apply its other privileges, as well as whether the user is protected from other accounts:

  - A system user can modify both system and regular accounts. That is, a user who has the appropriate privileges to perform a given operation on regular accounts is enabled by possession of `SYSTEM_USER` to also perform the operation on system accounts. A system account can be modified only by system users with appropriate privileges, not by regular users.
  - A regular user with appropriate privileges can modify regular accounts, but not system accounts. A regular account can be modified by both system and regular users with appropriate privileges.

  For more information, see Section 6.2.11, “Account Categories”.

  The protection against modification by regular accounts that is afforded to system accounts by the `SYSTEM_USER` privilege does not apply to regular accounts that have privileges on the `mysql` system schema and thus can directly modify the grant tables in that schema. For full protection, do not grant `mysql` schema privileges to regular accounts. See Protecting System Accounts Against Manipulation by Regular Accounts.

- `SYSTEM_VARIABLES_ADMIN`

  Affects the following operations and server behaviors:

  - Enables system variable changes at runtime:
    - Enables server configuration changes to global system variables with `SET GLOBAL` and `SET PERSIST`.
    - Enables server configuration changes to global system variables with `SET PERSIST_ONLY`, if the user also has `PERSIST_RO_VARIABLES_ADMIN`.
    - Enables setting restricted session system variables that require a special privilege. In effect, `SYSTEM_VARIABLES_ADMIN` implies `SESSION_VARIABLES_ADMIN` without explicitly granting `SESSION_VARIABLES_ADMIN`.

    See also Section 5.1.9.1, “System Variable Privileges”.
  - Enables changes to global transaction characteristics (see Section 13.3.7, “SET TRANSACTION Syntax”).

- `TABLE_ENCRYPTION_ADMIN` (added in MySQL 8.0.16)

  Enables a user to override default encryption settings when `table_encryption_privilege_check` is enabled; see Defining an Encryption Default for Schemas and General Tablespaces.

- `VERSION_TOKEN_ADMIN`

  Enables execution of Version Tokens user-defined functions. This privilege is defined by the `version_tokens` plugin; see Section 5.6.6, “Version Tokens”.

- `XA_RECOVER_ADMIN`

  Enables execution of the `XA RECOVER` statement; see Section 13.3.8.1, “XA Transaction SQL Syntax”.

  Prior to MySQL 8.0, any user could execute the `XA RECOVER` statement to discover the XID values for outstanding prepared XA transactions, possibly leading to commit or rollback of an XA transaction by a user other than the one who started it. In MySQL 8.0, `XA RECOVER` is permitted only to users who have the `XA_RECOVER_ADMIN` privilege, which is expected to be granted only to administrative users who have need for it. This might be the case, for example, for administrators of an XA application if it has

crashed and it is necessary to find outstanding transactions started by the application so they can be rolled back. This privilege requirement prevents users from discovering the XID values for outstanding prepared XA transactions other than their own. It does not affect normal commit or rollback of an XA transaction because the user who started it knows its XID.

Privilege-Granting Guidelines

It is a good idea to grant to an account only those privileges that it needs. You should exercise particular caution in granting the `FILE` and administrative privileges:

- `FILE` can be abused to read into a database table any files that the MySQL server can read on the server host. This includes all world-readable files and files in the server's data directory. The table can then be accessed using `SELECT` to transfer its contents to the client host.
- `GRANT OPTION` enables users to give their privileges to other users. Two users that have different privileges and with the `GRANT OPTION` privilege are able to combine privileges.
- `ALTER` may be used to subvert the privilege system by renaming tables.
- `SHUTDOWN` can be abused to deny service to other users entirely by terminating the server.
- `PROCESS` can be used to view the plain text of currently executing statements, including statements that set or change passwords.
- `SUPER` can be used to terminate other sessions or change how the server operates.
- Privileges granted for the `mysql` system database itself can be used to change passwords and other access privilege information:
  - Passwords are stored encrypted, so a malicious user cannot simply read them to know the plain text password. However, a user with write access to the `mysql.user` system table `authentication_string` column can change an account's password, and then connect to the MySQL server using that account.
  - `INSERT` or `UPDATE` granted for the `mysql` system database enable a user to add privileges or modify existing privileges, respectively.
  - `DROP` for the `mysql` system database enables a user to remote privilege tables, or even the database itself.

Static Versus Dynamic Privileges

MySQL supports static and dynamic privileges:

- Static privileges are built in to the server. They are always available to be granted to user accounts and cannot be unregistered.
- Dynamic privileges can be registered and unregistered at runtime. This affects their availability: A dynamic privilege that has not been registered cannot be granted.

For example, the `SELECT` and `INSERT` privileges are static and always available, whereas a dynamic privilege becomes available only if the server component that implements it has been enabled.

The remainder of this section describes how dynamic privileges work in MySQL. The discussion uses the term “components” but applies equally to plugins.

Note

Server administrators should be aware of which server components define dynamic privileges. For MySQL distributions, documentation of components that define dynamic privileges describes those privileges.

Third-party components may also define dynamic privileges; an administrator should understand those privileges and not install components that might conflict or compromise server operation. For example, one component conflicts with another if both define a privilege with the same name. Component developers can reduce the likelihood of this occurrence by choosing privilege names having a prefix based on the component name.

The server maintains the set of registered dynamic privileges internally in memory. Unregistration occurs at server shutdown.

Normally, a server component that defines dynamic privileges registers them when it is installed, during its initialization sequence. When uninstalled, a server component does not unregister its registered dynamic privileges. (This is current practice, not a requirement. That is, components could, but do not, unregister at any time privileges they register.)

No warning or error occurs for attempts to register an already registered dynamic privilege. Consider the following sequence of statements:

```
INSTALL COMPONENT 'my_component';
UNINSTALL COMPONENT 'my_component';
INSTALL COMPONENT 'my_component';
```

The first `INSTALL COMPONENT` statement registers any privileges defined by server component `my_component`, but `UNINSTALL COMPONENT` does not unregister them. For the second `INSTALL COMPONENT` statement, the component privileges it registers are found to be already registered, but no warnings or errors occur.

Dynamic privileges apply only at the global level. The server stores information about current assignments of dynamic privileges to user accounts in the `mysql.global_grants` system table:

- The server automatically registers privileges named in `global_grants` during server startup (unless the `--skip-grant-tables` option is given).
- The `GRANT` and `REVOKE` statements modify the contents of `global_grants`.
- Dynamic privilege assignments listed in `global_grants` are persistent. They are not removed at server shutdown.

Example: The following statement grants to user `u1` the privileges required to control replication (including Group Replication) on a slave server, and to modify system variables:

```
GRANT REPLICATION_SLAVE_ADMIN, GROUP_REPLICATION_ADMIN, BINLOG_ADMIN
ON *.* TO 'u1'@'localhost';
```

Granted dynamic privileges appear in the output from the `SHOW GRANTS` statement and the `INFORMATION_SCHEMA USER_PRIVILEGES` table.

For `GRANT` and `REVOKE` at the global level, any named privileges not recognized as static are checked against the current set of registered dynamic privileges and granted if found. Otherwise, an error occurs to indicate an unknown privilege identifier.

For GRANT and REVOKE the meaning of ALL [PRIVILEGES] at the global level includes all static global privileges, as well as all currently registered dynamic privileges:

- GRANT ALL at the global level grants all static global privileges and all currently registered dynamic privileges. A dynamic privilege registered subsequent to execution of the GRANT statement is not granted retroactively to any account.
- REVOKE ALL at the global level revokes all granted static global privileges and all granted dynamic privileges.

The FLUSH PRIVILEGES statement reads the global_grants table for dynamic privilege assignments and registers any unregistered privileges found there.

For descriptions of the dynamic privileges provided by MySQL Server and server components included in MySQL distributions, see Section 6.2.2, “Privileges Provided by MySQL”.

Migrating Accounts from SUPER to Dynamic Privileges

In MySQL 8.0, many operations that previously required the SUPER privilege are also associated with a dynamic privilege of more limited scope. (For descriptions of these privileges, see Section 6.2.2, “Privileges Provided by MySQL”.) Each such operation can be permitted to an account by granting the associated dynamic privilege rather than SUPER. This change improves security by enabling DBAs to avoid granting SUPER and tailor user privileges more closely to the operations permitted. SUPER is now deprecated and will be removed in a future version of MySQL.

When removal of SUPER occurs, operations that formerly required SUPER will fail unless accounts granted SUPER are migrated to the appropriate dynamic privileges. Use the following instructions to accomplish that goal so that accounts are ready prior to SUPER removal:

1. Execute this query to identify accounts that are granted SUPER:

   ```
   SELECT GRANTEE FROM INFORMATION_SCHEMA.USER_PRIVILEGES
   WHERE PRIVILEGE_TYPE = 'SUPER';
   ```

2. For each account identified by the preceding query, determine the operations for which it needs SUPER. Then grant the dynamic privileges corresponding to those operations, and revoke SUPER.

   For example, if 'u1'@'localhost' requires SUPER for binary log purging and system variable modification, these statements make the required changes to the account:

   ```
   GRANT BINLOG_ADMIN, SYSTEM_VARIABLES_ADMIN ON *.* TO 'u1'@'localhost';
   REVOKE SUPER ON *.* FROM 'u1'@'localhost';
   ```

   After you have modified all applicable accounts, the INFORMATION_SCHEMA query in the first step should produce an empty result set.

6.2.3 Grant Tables

The mysql system database includes several grant tables that contain information about user accounts and the privileges held by them. This section describes those tables. For information about other tables in the system database, see Section 5.3, “The mysql System Schema”.

The discussion here describes the underlying structure of the grant tables and how the server uses their contents when interacting with clients. However, normally you do not modify the grant tables directly. Modifications occur indirectly when you use account-management statements such as CREATE USER, GRANT, and REVOKE to set up accounts and control the privileges available to each one. See

Section 13.7.1, “Account Management Statements”. When you use such statements to perform account manipulations, the server modifies the grant tables on your behalf.

Note

Direct modification of grant tables using statements such as `INSERT`, `UPDATE`, or `DELETE` is discouraged and done at your own risk. The server is free to ignore rows that become malformed as a result of such modifications.

For any operation that modifies a grant table, the server checks whether the table has the expected structure and produces an error if not. To update the tables to the expected structure, perform the MySQL upgrade procedure. See Section 2.11, “Upgrading MySQL”.

- Grant Table Overview
- The user and db Grant Tables
- The tables_priv and columns_priv Grant Tables
- The procs_priv Grant Table
- The proxies_priv Grant Table
- The global_grants Grant Table
- The default_roles Grant Table
- The role_edges Grant Table
- The password_history Grant Table
- Grant Table Scope Column Properties
- Grant Table Privilege Column Properties

Grant Table Overview

These `mysql` database tables contain grant information:

- `user`: User accounts, static global privileges, and other nonprivilege columns.
- `global_grants`: Dynamic global privileges.
- `db`: Database-level privileges.
- `tables_priv`: Table-level privileges.
- `columns_priv`: Column-level privileges.
- `procs_priv`: Stored procedure and function privileges.
- `proxies_priv`: Proxy-user privileges.
- `default_roles`: Default user roles.
- `role_edges`: Edges for role subgraphs.
- `password_history`: Password change history.

For information about the differences between static and dynamic global privileges, see Static Versus Dynamic Privileges.)

In MySQL 8.0, grant tables use the `InnoDB` storage engine and are transactional. Before MySQL 8.0, grant tables used the `MyISAM` storage engine and were nontransactional. This change of grant table storage engine enables an accompanying change to the behavior of account-management statements such as `CREATE USER` or `GRANT`. Previously, an account-management statement that named multiple users could succeed for some users and fail for others. Now, each statement is transactional and either succeeds for all named users or rolls back and has no effect if any error occurs.

Each grant table contains scope columns and privilege columns:

- Scope columns determine the scope of each row in the tables; that is, the context in which the row applies. For example, a `user` table row with `Host` and `User` values of `'h1.example.net'` and `'bob'` applies to authenticating connections made to the server from the host `h1.example.net` by a client that specifies a user name of `bob`. Similarly, a `db` table row with `Host`, `User`, and `Db` column values of `'h1.example.net'`, `'bob'` and `'reports'` applies when `bob` connects from the host `h1.example.net` to access the `reports` database. The `tables_priv` and `columns_priv` tables contain scope columns indicating tables or table/column combinations to which each row applies. The `procs_priv` scope columns indicate the stored routine to which each row applies.
- Privilege columns indicate which privileges a table row grants; that is, which operations it permits to be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. Section 6.2.7, “Access Control, Stage 2: Request Verification”, describes the rules for this.

In addition, a grant table may contain columns used for purposes other than scope or privilege assessment.

The server uses the grant tables in the following manner:

- The `user` table scope columns determine whether to reject or permit incoming connections. For permitted connections, any privileges granted in the `user` table indicate the user's static global privileges. Any privileges granted in this table apply to *all* databases on the server.

  Caution

  Because any static global privilege is considered a privilege for all databases, any static global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`, except databases that have been restricted at the database level by partial revokes.

- The `global_grants` table lists current assignments of dynamic global privileges to user accounts. For each row, the scope columns determine which user has the privilege named in the privilege column.
- The `db` table scope columns determine which users can access which databases from which hosts. The privilege columns determine the permitted operations. A privilege granted at the database level applies to the database and to all objects in the database, such as tables and stored programs.
- The `tables_priv` and `columns_priv` tables are similar to the `db` table, but are more fine-grained: They apply at the table and column levels rather than at the database level. A privilege granted at the table level applies to the table and to all its columns. A privilege granted at the column level applies only to a specific column.
- The `procs_priv` table applies to stored routines (stored procedures and functions). A privilege granted at the routine level applies only to a single procedure or function.
- The `proxies_priv` table indicates which users can act as proxies for other users and whether a user can grant the `PROXY` privilege to other users.

- The `default_roles` and `role_edges` tables contain information about role relationships.
- The `password_history` table retains previously chosen passwords to enable restrictions on password reuse. See Section 6.2.15, “Password Management”.

The server reads the contents of the grant tables into memory when it starts. You can tell it to reload the tables by issuing a `FLUSH PRIVILEGES` statement or executing a `mysqladmin flush-privileges` or `mysqladmin reload` command. Changes to the grant tables take effect as indicated in Section 6.2.13, “When Privilege Changes Take Effect”.

When you modify an account, it is a good idea to verify that your changes have the intended effect. To check the privileges for a given account, use the `SHOW GRANTS` statement. For example, to determine the privileges that are granted to an account with user name and host name values of `bob` and `pc84.example.com`, use this statement:

```
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

To display nonprivilege properties of an account, use `SHOW CREATE USER`:

```
SHOW CREATE USER 'bob'@'pc84.example.com';
```

The user and db Grant Tables

The server uses the `user` and `db` tables in the `mysql` database at both the first and second stages of access control (see Section 6.2, “Access Control and Account Management”). The columns in the `user` and `db` tables are shown here.

Table 6.4 user and db Table Columns

Table Name	`user`	`db`
Scope columns	`Host`	`Host`
	`User`	`Db`
		`User`
Privilege columns	`Select_priv`	`Select_priv`
	`Insert_priv`	`Insert_priv`
	`Update_priv`	`Update_priv`
	`Delete_priv`	`Delete_priv`
	`Index_priv`	`Index_priv`
	`Alter_priv`	`Alter_priv`
	`Create_priv`	`Create_priv`
	`Drop_priv`	`Drop_priv`
	`Grant_priv`	`Grant_priv`
	`Create_view_priv`	`Create_view_priv`
	`Show_view_priv`	`Show_view_priv`
	`Create_routine_priv`	`Create_routine_priv`
	`Alter_routine_priv`	`Alter_routine_priv`
	`Execute_priv`	`Execute_priv`
	`Trigger_priv`	`Trigger_priv`

Table Name	user	db
	Event_priv	Event_priv
	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv
	References_priv	References_priv
	Reload_priv	
	Shutdown_priv	
	Process_priv	
	File_priv	
	Show_db_priv	
	Super_priv	
	Repl_slave_priv	
	Repl_client_priv	
	Create_user_priv	
	Create_tablespace_priv	
	Create_role_priv	
	Drop_role_priv	
Security columns	ssl_type	
	ssl_cipher	
	x509_issuer	
	x509_subject	
	plugin	
	authentication_string	
	password_expired	
	password_last_changed	
	password_lifetime	
	account_locked	
	Password_reuse_history	
	Password_reuse_time	
	Password_require_current	
	User_attributes	
Resource control columns	max_questions	
	max_updates	
	max_connections	
	max_user_connections	

The user table plugin and authentication_string columns store authentication plugin and credential information.

The server uses the plugin named in the plugin column of an account row to authenticate connection attempts for the account.

The `plugin` column must be nonempty. At startup, and at runtime when `FLUSH PRIVILEGES` is executed, the server checks `user` table rows. For any row with an empty `plugin` column, the server writes a warning to the error log of this form:

```
[Warning] User entry 'user_name'@'host_name' has an empty plugin
value. The user will be ignored and no one can login with this user
anymore.
```

To assign a plugin to an account that is missing one, use the `ALTER USER` statement.

The `password_expired` column permits DBAs to expire account passwords and require users to reset their password. The default `password_expired` value is `'N'`, but can be set to `'Y'` with the `ALTER USER` statement. After an account's password has been expired, all operations performed by the account in subsequent connections to the server result in an error until the user issues an `ALTER USER` statement to establish a new account password.

Note

Although it is possible to “reset” an expired password by setting it to its current value, it is preferable, as a matter of good policy, to choose a different password. DBAs can enforce non-reuse by establishing an appropriate password-reuse policy. See Password Reuse Policy.

`password_last_changed` is a `TIMESTAMP` column indicating when the password was last changed. The value is non-`NULL` only for accounts that use a MySQL built-in authentication plugin (`mysql_native_password`, `sha256_password`, or `caching_sha2_password`). The value is `NULL` for other accounts, such as those authenticated using an external authentication system.

`password_last_changed` is updated by the `CREATE USER`, `ALTER USER`, and `SET PASSWORD` statements, and by `GRANT` statements that create an account or change an account password.

`password_lifetime` indicates the account password lifetime, in days. If the password is past its lifetime (assessed using the `password_last_changed` column), the server considers the password expired when clients connect using the account. A value of *`N`* greater than zero means that the password must be changed every *`N`* days. A value of 0 disables automatic password expiration. If the value is `NULL` (the default), the global expiration policy applies, as defined by the `default_password_lifetime` system variable.

`account_locked` indicates whether the account is locked (see Section 6.2.19, “Account Locking”).

`Password_reuse_history` is the value of the `PASSWORD HISTORY` option for the account, or `NULL` for the default history.

`Password_reuse_time` is the value of the `PASSWORD REUSE INTERVAL` option for the account, or `NULL` for the default interval.

`Password_require_current` (added in MySQL 8.0.13) corresponds to the value of the `PASSWORD REQUIRE` option for the account, as shown by the following table.

Table 6.5 Permitted Password_require_current Values

Password_require_current Value	Corresponding PASSWORD REQUIRE Option
`'Y'`	`PASSWORD REQUIRE CURRENT`
`'N'`	`PASSWORD REQUIRE CURRENT OPTIONAL`

Password_require_current Value	Corresponding PASSWORD REQUIRE Option
NULL	PASSWORD REQUIRE CURRENT DEFAULT

User_attributes (added in MySQL 8.0.14) stores account attributes not stored in other columns:

- additional_password: The secondary password, if any. See Dual Password Support.
- Restrictions: Restriction lists, if any. Restrictions are added by partial-revoke operations. The attribute value is an array of elements that each have Database and Restrictions keys indicating the name of a restricted database and the applicable restrictions on it (see Section 6.2.12, “Privilege Restriction Using Partial Revokes”).

If no attributes apply, User_attributes is NULL.

Example: An account that has a secondary password and partially revoked database privileges has additional_password and Restrictions attributes in the column value:

```
mysql> SELECT User_attributes FROM mysql.User WHERE User = 'u'\G
*************************** 1. row ***************************
User_attributes: {"Restrictions":
                  [{"Database": "mysql", "Privileges": ["SELECT"]}],
                 "additional_password": "hashed_credentials"}
```

To determine which attributes are present, use the JSON_KEYS() function:

```
SELECT User, Host, JSON_KEYS(User_attributes)
FROM mysql.user WHERE User_attributes IS NOT NULL;
```

To extract a particular attribute, such as Restrictions, do this:

```
SELECT User, Host, User_attributes->>'$.Restrictions'
FROM mysql.user WHERE User_attributes->>'$.Restrictions' <> '';
```

The tables_priv and columns_priv Grant Tables

During the second stage of access control, the server performs request verification to ensure that each client has sufficient privileges for each request that it issues. In addition to the user and db grant tables, the server may also consult the tables_priv and columns_priv tables for requests that involve tables. The latter tables provide finer privilege control at the table and column levels. They have the columns shown in the following table.

Table 6.6 tables_priv and columns_priv Table Columns

Table Name	tables_priv	columns_priv
Scope columns	Host	Host
	Db	Db
	User	User
	Table_name	Table_name
		Column_name
Privilege columns	Table_priv	Column_priv

Table Name	tables_priv	columns_priv
	Column_priv	
Other columns	Timestamp	Timestamp
	Grantor	

The Timestamp and Grantor columns are set to the current timestamp and the CURRENT_USER value, respectively, but are otherwise unused.

The procs_priv Grant Table

For verification of requests that involve stored routines, the server may consult the procs_priv table, which has the columns shown in the following table.

Table 6.7 procs_priv Table Columns

Table Name	procs_priv
Scope columns	Host
	Db
	User
	Routine_name
	Routine_type
Privilege columns	Proc_priv
Other columns	Timestamp
	Grantor

The Routine_type column is an ENUM column with values of 'FUNCTION' or 'PROCEDURE' to indicate the type of routine the row refers to. This column enables privileges to be granted separately for a function and a procedure with the same name.

The Timestamp and Grantor columns are unused.

The proxies_priv Grant Table

The proxies_priv table records information about proxy accounts. It has these columns:

- Host, User: The proxy account; that is, the account that has the PROXY privilege for the proxied account.
- Proxied_host, Proxied_user: The proxied account.
- Grantor, Timestamp: Unused.
- With_grant: Whether the proxy account can grant the PROXY privilege to other accounts.

For an account to be able to grant the PROXY privilege to other accounts, it must have a row in the proxies_priv table with With_grant set to 1 and Proxied_host and Proxied_user set to indicate the account or accounts for which the privilege can be granted. For example, the 'root'@'localhost' account created during MySQL installation has a row in the proxies_priv table that enables granting the PROXY privilege for ''@'', that is, for all users and all hosts. This enables root to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. See Section 6.2.18, “Proxy Users”.

The global_grants Grant Table

The `global_grants` table lists current assignments of dynamic global privileges to user accounts. The table has these columns:

- `USER`, `HOST`: The user name and host name of the account to which the privilege is granted.
- `PRIV`: The privilege name.
- `WITH_GRANT_OPTION`: Whether the account can grant the privilege to other accounts.

The default_roles Grant Table

The `default_roles` table lists default user roles. It has these columns:

- `HOST`, `USER`: The account or role to which the default role applies.
- `DEFAULT_ROLE_HOST`, `DEFAULT_ROLE_USER`: The default role.

The role_edges Grant Table

The `role_edges` table lists edges for role subgraphs. It has these columns:

- `FROM_HOST`, `FROM_USER`: The account that is granted a role.
- `TO_HOST`, `TO_USER`: The role that is granted to the account.
- `WITH_ADMIN_OPTION`: Whether the account can grant the role to and revoke it from other accounts by using `WITH ADMIN OPTION`.

The password_history Grant Table

The `password_history` table contains information about password changes. It has these columns:

- `Host`, `User`: The account for which the password change occurred.
- `Password_timestamp`: The time when the password change occurred.
- `Password`: The new password hash value.

The `password_history` table accumulates a sufficient number of nonempty passwords per account to enable MySQL to perform checks against both the account password history length and reuse interval. Automatic pruning of entries that are outside both limits occurs when password-change attempts occur.

Note

The empty password does not count in the password history and is subject to reuse at any time.

If an account is renamed, its entries are renamed to match. If an account is dropped or its authentication plugin is changed, its entries are removed.

Grant Table Scope Column Properties

Scope columns in the grant tables contain strings. The default value for each is the empty string. The following table shows the number of characters permitted in each column.

Table 6.8 Grant Table Scope Column Lengths

Column Name	Maximum Permitted Characters
`Host`, `Proxied_host`	255 (60 prior to MySQL 8.0.17)
`User`, `Proxied_user`	32
`Db`	64
`Table_name`	64
`Column_name`	64
`Routine_name`	64

`Host` and `Proxied_host` values are converted to lowercase before being stored in the grant tables.

For access-checking purposes, comparisons of `User`, `Proxied_user`, `authentication_string`, `Db`, and `Table_name` values are case-sensitive. Comparisons of `Host`, `Proxied_host`, `Column_name`, and `Routine_name` values are not case-sensitive.

Grant Table Privilege Column Properties

The `user` and `db` tables list each privilege in a separate column that is declared as `ENUM('N','Y') DEFAULT 'N'`. In other words, each privilege can be disabled or enabled, with the default being disabled.

The `tables_priv`, `columns_priv`, and `procs_priv` tables declare the privilege columns as `SET` columns. Values in these columns can contain any combination of the privileges controlled by the table. Only those privileges listed in the column value are enabled.

Table 6.9 Set-Type Privilege Column Values

Table Name	Column Name	Possible Set Elements
`tables_priv`	`Table_priv`	`'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter', 'Create View', 'Show view', 'Trigger'`
`tables_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`
`columns_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`
`procs_priv`	`Proc_priv`	`'Execute', 'Alter Routine', 'Grant'`

Only the `user` and `global_grants` tables specify administrative privileges, such as `RELOAD`, `SHUTDOWN`, and `SYSTEM_VARIABLES_ADMIN`. Administrative operations are operations on the server itself and are not database-specific, so there is no reason to list these privileges in the other grant tables. Consequently, the server need consult only the `user` and `global_grants` tables to determine whether a user can perform an administrative operation.

The `FILE` privilege also is specified only in the `user` table. It is not an administrative privilege as such, but a user's ability to read or write files on the server host is independent of the database being accessed.

6.2.4 Specifying Account Names

MySQL account names consist of a user name and a host name, which enables creation of distinct accounts for users with the same user name who can connect from different hosts. This section describes how to write account names, including special values and wildcard rules.

MySQL role names are similar to account names, with some differences described at Section 6.2.5, “Specifying Role Names”.

In SQL statements such as `CREATE USER`, `GRANT`, and `SET PASSWORD`, account names follow these rules:

- Account name syntax is `'user_name'@'host_name'`.
- An account name consisting only of a user name is equivalent to `'user_name'@'%'`. For example, `'me'` is equivalent to `'me'@'%'`.
- The user name and host name need not be quoted if they are legal as unquoted identifiers. Quotes are necessary to specify a *`user_name`* string containing special characters (such as space or `-`), or a *`host_name`* string containing special characters or wildcard characters (such as `.` or `%`). For example, in the account name `'test-user'@'%.com'`, both the user name and host name parts require quotes.
- Quote user names and host names as identifiers or as strings, using either backticks (`` ` ``), single quotation marks (`'`), or double quotation marks (`"`). For string-quoting and identifier-quoting guidelines, see Section 9.1.1, “String Literals”, and Section 9.2, “Schema Object Names”.
- The user name and host name parts, if quoted, must be quoted separately. That is, write `'me'@'localhost'`, not `'me@localhost'`. The latter is actually equivalent to `'me@localhost'@'%'`.
- A reference to the `CURRENT_USER` or `CURRENT_USER()` function is equivalent to specifying the current client's user name and host name literally.

MySQL stores account names in grant tables in the `mysql` system database using separate columns for the user name and host name parts:

- The `user` table contains one row for each account. The `User` and `Host` columns store the user name and host name. This table also indicates which global privileges the account has.
- Other grant tables indicate privileges an account has for databases and objects within databases. These tables have `User` and `Host` columns to store the account name. Each row in these tables associates with the account in the `user` table that has the same `User` and `Host` values.
- For access-checking purposes, comparisons of User values are case-sensitive. Comparisons of Host values are not case sensitive.

For additional detail about the properties of user names and host names as stored in the grant tables, such as maximum length, see Grant Table Scope Column Properties.

User names and host names have certain special values or wildcard conventions, as described following.

The user name part of an account name is either a nonblank value that literally matches the user name for incoming connection attempts, or a blank value (empty string) that matches any user name. An account with a blank user name is an anonymous user. To specify an anonymous user in SQL statements, use a quoted empty user name part, such as `''@'localhost'`.

The host name part of an account name can take many forms, and wildcards are permitted:

- A host value can be a host name or an IP address (IPv4 or IPv6). The name `'localhost'` indicates the local host. The IP address `'127.0.0.1'` indicates the IPv4 loopback interface. The IP address `'::1'` indicates the IPv6 loopback interface.

- The `%` and `_` wildcard characters are permitted in host name or IP address values. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. For example, a host value of `'%'` matches any host name, whereas a value of `'%.mysql.com'` matches any host in the `mysql.com` domain. `'198.51.100.%'` matches any host in the 198.51.100 class C network.

  Because IP wildcard values are permitted in host values (for example, `'198.51.100.%'` to match every host on a subnet), someone could try to exploit this capability by naming a host `198.51.100.somewhere.com`. To foil such attempts, MySQL does not perform matching on host names that start with digits and a dot. For example, if a host is named `1.2.example.com`, its name never matches the host part of account names. An IP wildcard value can match only IP addresses, not host names.

- For a host value specified as an IPv4 address, a netmask can be given to indicate how many address bits to use for the network number. Netmask notation cannot be used for IPv6 addresses.

  The syntax is *`host_ip/netmask`*. For example:

  ```
  CREATE USER 'david'@'198.51.100.0/255.255.255.0';
  ```

  This enables `david` to connect from any client host having an IP address *`client_ip`* for which the following condition is true:

  ```
  client_ip & netmask = host_ip
  ```

  That is, for the `CREATE USER` statement just shown:

  ```
  client_ip & 255.255.255.0 = 198.51.100.0
  ```

  IP addresses that satisfy this condition range from `198.51.100.0` to `198.51.100.255`.

  A netmask typically begins with bits set to 1, followed by bits set to 0. Examples:

  - `198.0.0.0/255.0.0.0`: Any host on the 198 class A network
  - `198.51.100.0/255.255.0.0`: Any host on the 198.51 class B network
  - `198.51.100.0/255.255.255.0`: Any host on the 198.51.100 class C network
  - `198.51.100.1`: Only the host with this specific IP address

The server performs matching of host values in account names against the client host using the value returned by the system DNS resolver for the client host name or IP address. Except in the case that the account host value is specified using netmask notation, the server performs this comparison as a string match, even for an account host value given as an IP address. This means that you should specify account host values in the same format used by DNS. Here are examples of problems to watch out for:

- Suppose that a host on the local network has a fully qualified name of `host1.example.com`. If DNS returns name lookups for this host as `host1.example.com`, use that name in account host values. If DNS returns just `host1`, use `host1` instead.
- If DNS returns the IP address for a given host as `198.51.100.2`, that will match an account host value of `198.51.100.2` but not `198.051.100.2`. Similarly, it will match an account host pattern like `198.51.100.%` but not `198.051.100.%`.

To avoid problems like these, it is advisable to check the format in which your DNS returns host names and addresses. Use values in the same format in MySQL account names.

6.2.5 Specifying Role Names

MySQL role names refer to roles, which are named collections of privileges. For role usage examples, see Section 6.2.10, “Using Roles”.

Role names have syntax and semantics similar to account names; see Section 6.2.4, “Specifying Account Names”. As stored in the grant tables, they have the same properties as account names, which are described in Grant Table Scope Column Properties.

Role names differ from account names in these respects:

- The user part of role names cannot be blank. Thus, there is no “anonymous role” analogous to the concept of “anonymous user.”
- As for an account name, omitting the host part of a role name results in a host part of `'%'`. But unlike `'%'` in an account name, a host part of `'%'` in a role name has no wildcard properties. For example, for a name `'me'@'%'` used as a role name, the host part (`'%'`) is just a literal value; it has no “any host” matching property.
- Netmask notation in the host part of a role name has no significance.
- An account name is permitted to be `CURRENT_USER()` in several contexts. A role name is not.

It is possible for a row in the `mysql.user` system table to serve as both an account and a role. In this case, any special user or host name matching properties do not apply in contexts for which the name is used as a role name. For example, you cannot execute the following statement with the expectation that it will set the current session roles using all roles that have a user part of `myrole` and any host name:

```
SET ROLE 'myrole'@'%';
```

Instead, the statement sets the active role for the session to the role with exactly the name `'myrole'@'%'`.

For this reason, role names are often specified using only the user name part and letting the host name part implicitly be `'%'`. Specifying a role with a non-`'%'` host part can be useful if you intend to create a name that works both as a role an as a user account that is permitted to connect from the given host.

6.2.6 Access Control, Stage 1: Connection Verification

When you attempt to connect to a MySQL server, the server accepts or rejects the connection based on these conditions:

- Your identity and whether you can verify your identity by supplying the correct password
- Whether your account is locked or unlocked

The server checks credentials first, then account locking state. A failure for either step causes the server to deny access to you completely. Otherwise, the server accepts the connection, and then enters Stage 2 and waits for requests.

Credential checking is performed using the three `user` table scope columns (`Host`, `User`, and `authentication_string`). Locking state is recorded in the `user` table `account_locked` column. The server accepts the connection only if the `Host` and `User` columns in some `user` table row match the client host name and user name, the client supplies the password specified in that row, and the `account_locked` value is `'N'`. The rules for permissible `Host` and `User` values are given in Section 6.2.4, “Specifying Account Names”. Account locking can be changed with the `ALTER USER` statement.

Your identity is based on two pieces of information:

- The client host from which you connect
- Your MySQL user name

If the `User` column value is nonblank, the user name in an incoming connection must match exactly. If the `User` value is blank, it matches any user name. If the `user` table row that matches an incoming connection has a blank user name, the user is considered to be an anonymous user with no name, not a user with the name that the client actually specified. This means that a blank user name is used for all further access checking for the duration of the connection (that is, during Stage 2).

The `authentication_string` column can be blank. This is not a wildcard and does not mean that any password matches. It means that the user must connect without specifying a password. If the server authenticates a client using a plugin, the authentication method that the plugin implements may or may not use the password in the `authentication_string` column. In this case, it is possible that an external password is also used to authenticate to the MySQL server.

Nonblank `authentication_string` values in the `user` table represent encrypted passwords. MySQL does not store passwords as cleartext for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the password hashing method implemented by the account authentication plugin). The encrypted password then is used during the connection process when checking whether the password is correct. This is done without the encrypted password ever traveling over the connection. See Section 6.2.1, “Account User Names and Passwords”.

From MySQL's point of view, the encrypted password is the *real* password, so you should never give anyone access to it. In particular, *do not give nonadministrative users read access to tables in the `mysql` system database*.

The following table shows how various combinations of `User` and `Host` values in the `user` table apply to incoming connections.

`User` Value	**`Host` Value**	**Permissible Connections**
`'fred'`	`'h1.example.net'`	`fred`, connecting from `h1.example.net`
`''`	`'h1.example.net'`	Any user, connecting from `h1.example.net`
`'fred'`	`'%'`	`fred`, connecting from any host
`''`	`'%'`	Any user, connecting from any host
`'fred'`	`'%.example.net'`	`fred`, connecting from any host in the `example.net` domain
`'fred'`	`'x.example.%'`	`fred`, connecting from `x.example.net`, `x.example.com`, `x.example.edu`, and so on; this is probably not useful
`'fred'`	`'198.51.100.177'`	`fred`, connecting from the host with IP address `198.51.100.177`
`'fred'`	`'198.51.100.%'`	`fred`, connecting from any host in the `198.51.100` class C subnet
`'fred'`	`'198.51.100.0/255.255.255.0'`	Same as previous example

It is possible for the client host name and user name of an incoming connection to match more than one row in the `user` table. The preceding set of examples demonstrates this: Several of the entries shown match a connection from `h1.example.net` by `fred`.

When multiple matches are possible, the server must determine which of them to use. It resolves this issue as follows:

- Whenever the server reads the `user` table into memory, it sorts the rows.
- When a client attempts to connect, the server looks through the rows in sorted order.
- The server uses the first row that matches the client host name and user name.

The server uses sorting rules that order rows with the most-specific `Host` values first. Literal host names and IP addresses are the most specific. (The specificity of a literal IP address is not affected by whether it has a netmask, so `198.51.100.13` and `198.51.100.0/255.255.255.0` are considered equally specific.) The pattern `'%'` means “any host” and is least specific. The empty string `''` also means “any host” but sorts after `'%'`. Rows with the same `Host` value are ordered with the most-specific `User` values first (a blank `User` value means “any user” and is least specific). For rows with equally-specific `Host` and `User` values, the order is nondeterministic.

To see how this works, suppose that the `user` table looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| %         | root     | ...
| %         | jeffrey  | ...
| localhost | root     | ...
| localhost |          | ...
+-----------+----------+-
```

When the server reads the table into memory, it sorts the rows using the rules just described. The result after sorting looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| localhost | root     | ...
| localhost |          | ...
| %         | jeffrey  | ...
| %         | root     | ...
+-----------+----------+-
```

When a client attempts to connect, the server looks through the sorted rows and uses the first match found. For a connection from `localhost` by `jeffrey`, two of the rows from the table match: the one with `Host` and `User` values of `'localhost'` and `''`, and the one with values of `'%'` and `'jeffrey'`. The `'localhost'` row appears first in sorted order, so that is the one the server uses.

Here is another example. Suppose that the `user` table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| %              | jeffrey  | ...
| h1.example.net |          | ...
+----------------+----------+-
```

The sorted table looks like this:

```
+---------------+----------+-
| Host          | User     | ...
+---------------+----------+-
| h1.example.net |          | ...
| %             | jeffrey  | ...
+---------------+----------+-
```

A connection by `jeffrey` from `h1.example.net` is matched by the first row, whereas a connection by `jeffrey` from any host is matched by the second.

Note

It is a common misconception to think that, for a given user name, all rows that explicitly name that user are used first when the server attempts to find a match for the connection. This is not true. The preceding example illustrates this, where a connection from `h1.example.net` by `jeffrey` is first matched not by the row containing `'jeffrey'` as the `User` column value, but by the row with no user name. As a result, `jeffrey` is authenticated as an anonymous user, even though he specified a user name when connecting.

If you are able to connect to the server, but your privileges are not what you expect, you probably are being authenticated as some other account. To find out what account the server used to authenticate you, use the `CURRENT_USER()` function. (See Section 12.15, “Information Functions”.) It returns a value in *`user_name@host_name`* format that indicates the `User` and `Host` values from the matching `user` table row. Suppose that `jeffrey` connects and issues the following query:

```
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| @localhost     |
+----------------+
```

The result shown here indicates that the matching `user` table row had a blank `User` column value. In other words, the server is treating `jeffrey` as an anonymous user.

Another way to diagnose authentication problems is to print out the `user` table and sort it by hand to see where the first match is being made.

6.2.7 Access Control, Stage 2: Request Verification

After you establish a connection, the server enters Stage 2 of access control. For each request that you issue through that connection, the server determines what operation you want to perform, then checks whether you have sufficient privileges to do so. This is where the privilege columns in the grant tables come into play. These privileges can come from any of the `user`, `global_grants`, `db`, `tables_priv`, `columns_priv`, or `procs_priv` tables. (You may find it helpful to refer to Section 6.2.3, “Grant Tables”, which lists the columns present in each grant table.)

The `user` and `global_grants` tables grant global privileges. The rows in these tables for a given account indicate the account privileges that apply on a global basis no matter what the default database is. For example, if the `user` table grants you the `DELETE` privilege, you can delete rows from any table in any database on the server host. It is wise to grant privileges in the `user` table only to people who need them, such as database administrators. For other users, leave all privileges in the `user` table set to `'N'` and grant privileges at more specific levels only (for particular databases, tables, columns, or routines). It is also possible to grant database privileges globally but use partial revokes to restrict them from being exercised on specific databases (see Section 6.2.12, “Privilege Restriction Using Partial Revokes”).

The `db` table grants database-specific privileges. Values in the scope columns of this table can take the following forms:

- A blank `User` value matches the anonymous user. A nonblank value matches literally; there are no wildcards in user names.
- The wildcard characters `%` and `_` can be used in the `Host` and `Db` columns. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. If you want to use either character literally when granting privileges, you must escape it with a backslash. For example, to include the underscore character (_) as part of a database name, specify it as `\_` in the `GRANT` statement.
- A `'%'` or blank `Host` value means “any host.”
- A `'%'` or blank `Db` value means “any database.”

The server reads the `db` table into memory and sorts it at the same time that it reads the `user` table. The server sorts the `db` table based on the `Host`, `Db`, and `User` scope columns. As with the `user` table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching rows, it uses the first match that it finds.

The `tables_priv`, `columns_priv`, and `procs_priv` tables grant table-specific, column-specific, and routine-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters `%` and `_` can be used in the `Host` column. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator.
- A `'%'` or blank `Host` value means “any host.”
- The `Db`, `Table_name`, `Column_name`, and `Routine_name` columns cannot contain wildcards or be blank.

The server sorts the `tables_priv`, `columns_priv`, and `procs_priv` tables based on the `Host`, `Db`, and `User` columns. This is similar to `db` table sorting, but simpler because only the `Host` column can contain wildcards.

The server uses the sorted tables to verify each request that it receives. For requests that require administrative privileges such as `SHUTDOWN` or `RELOAD`, the server checks only the `user` and `global_privilege` tables because those are the only tables that specify administrative privileges. The server grants access if a row for the account in those tables permits the requested operation and denies access otherwise. For example, if you want to execute `mysqladmin shutdown` but your `user` table row does not grant the `SHUTDOWN` privilege to you, the server denies access without even checking the `db` table. (The latter table contains no `Shutdown_priv` column, so there is no need to check it.)

For database-related requests (`INSERT`, `UPDATE`, and so on), the server first checks the user's global privileges in the `user` table row (less any privilege restrictions imposed by partial revokes). If the row permits the requested operation, access is granted. If the global privileges in the `user` table are insufficient, the server determines the user's database-specific privileges from the `db` table:

The server looks in the `db` table for a match on the `Host`, `Db`, and `User` columns. The `Host` and `User` columns are matched to the connecting user's host name and MySQL user name. The `Db` column is matched to the database that the user wants to access. If there is no row for the `Host` and `User`, access is denied.

After determining the database-specific privileges granted by the `db` table rows, the server adds them to the global privileges granted by the `user` table. If the result permits the requested operation, access is granted. Otherwise, the server successively checks the user's table and column privileges in the

`tables_priv` and `columns_priv` tables, adds those to the user's privileges, and permits or denies access based on the result. For stored-routine operations, the server uses the `procs_priv` table rather than `tables_priv` and `columns_priv`.

Expressed in boolean terms, the preceding description of how a user's privileges are calculated may be summarized like this:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
OR routine privileges
```

It may not be apparent why, if the global privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database, table, and column privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute an `INSERT INTO ... SELECT` statement, you need both the `INSERT` and the `SELECT` privileges. Your privileges might be such that the `user` table row grants one privilege global and the `db` table row grants the other specifically for the relevant database. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either your global or database privileges alone. It must make an access-control decision based on the combined privileges.

6.2.8 Adding Accounts, Assigning Privileges, and Dropping Accounts

To manage MySQL accounts, use the SQL statements intended for that purpose:

- `CREATE USER` and `DROP USER` create and remove accounts.
- `GRANT` and `REVOKE` assign privileges to and revoke privileges from accounts.
- `SHOW GRANTS` displays account privilege assignments.

Account-management statements cause the server to make appropriate modifications to the underlying grant tables, which are discussed in Section 6.2.3, “Grant Tables”.

Note

Direct modification of grant tables using statements such as `INSERT`, `UPDATE`, or `DELETE` is discouraged and done at your own risk. The server is free to ignore rows that become malformed as a result of such modifications.

For any operation that modifies a grant table, the server checks whether the table has the expected structure and produces an error if not. To update the tables to the expected structure, perform the MySQL upgrade procedure. See Section 2.11, “Upgrading MySQL”.

Another option for creating accounts is to use the GUI tool MySQL Workbench. Also, several third-party programs offer capabilities for MySQL account administration. `phpMyAdmin` is one such program.

This section discusses the following topics:

- Creating Accounts and Granting Privileges
- Checking Account Privileges and Properties
- Revoking Account Privileges
- Dropping Accounts

For additional information about the statements discussed here, see Section 13.7.1, “Account Management Statements”.

Creating Accounts and Granting Privileges

The following examples show how to use the `mysql` client program to set up new accounts. These examples assume that the MySQL `root` account has the `CREATE USER` privilege and all privileges that it grants to other accounts.

At the command line, connect to the server as the MySQL `root` user, supplying the appropriate password at the password prompt:

```
shell> mysql -u root -p
Enter password: (enter root password here)
```

After connecting to the server, you can add new accounts. The following example uses `CREATE USER` and `GRANT` statements to set up four accounts (where you see `'password'`, substitute an appropriate password):

```
CREATE USER 'finley'@'localhost'
  IDENTIFIED BY 'password';
GRANT ALL
  ON *.*
  TO 'finley'@'localhost'
  WITH GRANT OPTION;

CREATE USER 'finley'@'%.example.com'
  IDENTIFIED BY 'password';
GRANT ALL
  ON *.*
  TO 'finley'@'%.example.com'
  WITH GRANT OPTION;

CREATE USER 'admin'@'localhost'
  IDENTIFIED BY 'password';
GRANT RELOAD,PROCESS
  ON *.*
  TO 'admin'@'localhost';

CREATE USER 'dummy'@'localhost';
```

The accounts created by those statements have the following properties:

- Two accounts have a user name of `finley`. Both are superuser accounts with full global privileges to do anything. The `'finley'@'localhost'` account can be used only when connecting from the local host. The `'finley'@'%.example.com'` account uses the `'%'` wildcard in the host part, so it can be used to connect from any host in the `example.com` domain.

  The `'finley'@'localhost'` account is necessary if there is an anonymous-user account for `localhost`. Without the `'finley'@'localhost'` account, that anonymous-user account takes precedence when `finley` connects from the local host and `finley` is treated as an anonymous user. The reason for this is that the anonymous-user account has a more specific `Host` column value than the `'finley'@'%'` account and thus comes earlier in the `user` table sort order. (For information about `user` table sorting, see Section 6.2.6, “Access Control, Stage 1: Connection Verification”.)

- The `'admin'@'localhost'` account can be used only by `admin` to connect from the local host. It is granted the global `RELOAD` and `PROCESS` administrative privileges. These privileges enable the `admin` user to execute the `mysqladmin reload`, `mysqladmin refresh`, and `mysqladmin flush-xxx`

commands, as well as `mysqladmin processlist` . No privileges are granted for accessing any databases. You could add such privileges using `GRANT` statements.

- The `'dummy'@'localhost'` account has no password (which is insecure and not recommended). This account can be used only to connect from the local host. No privileges are granted. It is assumed that you will grant specific privileges to the account using `GRANT` statements.

The previous example grants privileges at the global level. The next example creates three accounts and grants them access at lower levels; that is, to specific databases or objects within databases. Each account has a user name of `custom`, but the host name parts differ:

```
CREATE USER 'custom'@'localhost'
  IDENTIFIED BY 'password';
GRANT ALL
  ON bankaccount.*
  TO 'custom'@'localhost';

CREATE USER 'custom'@'host47.example.com'
  IDENTIFIED BY 'password';
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
  ON expenses.*
  TO 'custom'@'host47.example.com';

CREATE USER 'custom'@'%.example.com'
  IDENTIFIED BY 'password';
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
  ON customer.addresses
  TO 'custom'@'%.example.com';
```

The three accounts can be used as follows:

- The `'custom'@'localhost'` account has all database-level privileges to access the `bankaccount` database. The account can be used to connect to the server only from the local host.
- The `'custom'@'host47.example.com'` account has specific database-level privileges to access the `expenses` database. The account can be used to connect to the server only from the host `host47.example.com`.
- The `'custom'@'%.example.com'` account has specific table-level privileges to access the `addresses` table in the `customer` database, from any host in the `example.com` domain. The account can be used to connect to the server from all machines in the domain due to use of the `%` wildcard character in the host part of the account name.

Checking Account Privileges and Properties

To see the privileges for an account, use `SHOW GRANTS`:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-----------------------------------------------------+
| Grants for admin@localhost                          |
+-----------------------------------------------------+
| GRANT RELOAD, PROCESS ON *.* TO 'admin'@'localhost' |
+-----------------------------------------------------+
```

To see nonprivilege properties for an account, use `SHOW CREATE USER`:

```
mysql> SET print_identified_with_as_hex = ON;
mysql> SHOW CREATE USER 'admin'@'localhost'\G
*************************** 1. row ***************************
CREATE USER for admin@localhost: CREATE USER 'admin'@'localhost'
```

```
IDENTIFIED WITH 'caching_sha2_password'
AS 0x24412430303524301D0E17054E2241362B1419313C3E44326F294133734B30792F436E77764270373039612E32445250786D43
REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK
PASSWORD HISTORY DEFAULT
PASSWORD REUSE INTERVAL DEFAULT
PASSWORD REQUIRE CURRENT DEFAULT
```

Enabling the `print_identified_with_as_hex` system variable (available as of MySQL 8.0.17) causes `SHOW CREATE USER` to display hash values that contain unprintable characters as hexadecimal strings rather than as regular string literals.

Revoking Account Privileges

To revoke account privileges, use the `REVOKE` statement. Privileges can be revoked at different levels, just as they can be granted at different levels.

Revoke global privileges:

```
REVOKE ALL
  ON *.*
  FROM 'finley'@'%.example.com';

REVOKE RELOAD
  ON *.*
  FROM 'admin'@'localhost';
```

Revoke database-level privileges:

```
REVOKE CREATE,DROP
  ON expenses.*
  FROM 'custom'@'host47.example.com';
```

Revoke table-level privileges:

```
REVOKE INSERT,UPDATE,DELETE
  ON customer.addresses
  FROM 'custom'@'%.example.com';
```

To check the effect of privilege revocation, use `SHOW GRANTS`:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-----------------------------------------+
| Grants for admin@localhost              |
+-----------------------------------------+
| GRANT PROCESS ON *.* TO 'admin'@'localhost' |
+-----------------------------------------+
```

Dropping Accounts

To remove an account, use the `DROP USER` statement. For example, to drop some of the accounts created previously:

```
DROP USER 'finley'@'localhost';
DROP USER 'finley'@'%.example.com';
DROP USER 'admin'@'localhost';
DROP USER 'dummy'@'localhost';
```

6.2.9 Reserved Accounts

One part of the MySQL installation process is data directory initialization (see Section 2.10.1, “Initializing the Data Directory”). During data directory initialization, MySQL creates user accounts that should be considered reserved:

- `'root'@'localhost'`: Used for administrative purposes. This account has all privileges, is a system account, and can perform any operation.

  Strictly speaking, this account name is not reserved, in the sense that some installations rename the `root` account to something else to avoid exposing a highly privileged account with a well-known name.

- `'mysql.sys'@'localhost'`: Used as the `DEFINER` for `sys` schema objects. Use of the `mysql.sys` account avoids problems that occur if a DBA renames or removes the `root` account. This account is locked so that it cannot be used for client connections.

- `'mysql.session'@'localhost'`: Used internally by plugins to access the server. This account is locked so that it cannot be used for client connections. The account is a system account.

- `'mysql.infoschema'@'localhost'`: Used as the `DEFINER` for `INFORMATION_SCHEMA` views. Use of the `mysql.infoschema` account avoids problems that occur if a DBA renames or removes the root account. This account is locked so that it cannot be used for client connections.

6.2.10 Using Roles

A MySQL role is a named collection of privileges. Like user accounts, roles can have privileges granted to and revoked from them.

A user account can be granted roles, which grants to the account the privileges associated with each role. This enables assignment of sets of privileges to accounts and provides a convenient alternative to granting individual privileges, both for conceptualizing desired privilege assignments and implementing them.

The following list summarizes role-management capabilities provided by MySQL:

- `CREATE ROLE` and `DROP ROLE` create and remove roles.
- `GRANT` and `REVOKE` assign privileges to revoke privileges from user accounts and roles.
- `SHOW GRANTS` displays privilege and role assignments for user accounts and roles.
- `SET DEFAULT ROLE` specifies which account roles are active by default.
- `SET ROLE` changes the active roles within the current session.
- The `CURRENT_ROLE()` function displays the active roles within the current session.
- The `mandatory_roles` and `activate_all_roles_on_login` system variables enable defining mandatory roles and automatic activation of granted roles when users log in to the server.

For descriptions of individual role-manipulation statements (including the privileges required to use them), see Section 13.7.1, “Account Management Statements”. The following discussion provides examples of role usage. Unless otherwise specified, SQL statements shown here should be executed using a MySQL account with sufficient administrative privileges, such as the `root` account.

- Creating Roles and Granting Privileges to Them
- Defining Mandatory Roles
- Checking Role Privileges

- Activating Roles
- Revoking Roles or Role Privileges
- Dropping Roles
- User and Role Interchangeability

Creating Roles and Granting Privileges to Them

Consider this scenario:

- An application uses a database named `app_db`.
- Associated with the application, there can be accounts for developers who create and maintain the application, and for users who interact with it.
- Developers need full access to the database. Some users need only read access, others need read/write access.

To avoid granting privileges individually to possibly many user accounts, create roles as names for the required privilege sets. This makes it easy to grant the required privileges to user accounts, by granting the appropriate roles.

To create the roles, use the `CREATE ROLE` statement:

```
CREATE ROLE 'app_developer', 'app_read', 'app_write';
```

Role names are much like user account names and consist of a user part and host part in *`'user_name'@'host_name'`* format. The host part, if omitted, defaults to `'%'`. The user and host parts can be unquoted unless they contain special characters such as `-` or `%`. Unlike account names, the user part of role names cannot be blank. For additional information, see Section 6.2.5, “Specifying Role Names”.

To assign privileges to the roles, execute `GRANT` statements using the same syntax as for assigning privileges to user accounts:

```
GRANT ALL ON app_db.* TO 'app_developer';
GRANT SELECT ON app_db.* TO 'app_read';
GRANT INSERT, UPDATE, DELETE ON app_db.* TO 'app_write';
```

Now suppose that initially you require one developer account, two user accounts that need read-only access, and one user account that needs read/write access. Use `CREATE USER` to create the accounts:

```
CREATE USER 'dev1'@'localhost' IDENTIFIED BY 'dev1pass';
CREATE USER 'read_user1'@'localhost' IDENTIFIED BY 'read_user1pass';
CREATE USER 'read_user2'@'localhost' IDENTIFIED BY 'read_user2pass';
CREATE USER 'rw_user1'@'localhost' IDENTIFIED BY 'rw_user1pass';
```

To assign each user account its required privileges, you could use `GRANT` statements of the same form as just shown, but that requires enumerating individual privileges for each user. Instead, use an alternative `GRANT` syntax that permits granting roles rather than privileges:

```
GRANT 'app_developer' TO 'dev1'@'localhost';
GRANT 'app_read' TO 'read_user1'@'localhost', 'read_user2'@'localhost';
GRANT 'app_read', 'app_write' TO 'rw_user1'@'localhost';
```

The GRANT statement for the rw_user1 account grants the read and write roles, which combine to provide the required read and write privileges.

The GRANT syntax for granting roles to an account differs from the syntax for granting privileges: There is an ON clause to assign privileges, whereas there is no ON clause to assign roles. Because the syntaxes are distinct, you cannot mix assigning privileges and roles in the same statement. (It is permitted to assign both privileges and roles to an account, but you must use separate GRANT statements, each with syntax appropriate to what is to be granted.) As of MySQL 8.0.16, roles cannot be granted to anonymous users.

A role when created is locked, has no password, and is assigned the default authentication plugin. (These role attributes can be changed later with the ALTER USER statement, by users who have the global CREATE USER privilege.)

While locked, a role cannot be used to authenticate to the server. If unlocked, a role can be used to authenticate. This is because roles and users are both authorization identifiers with much in common and little to distinguish them. See also User and Role Interchangeability.

Defining Mandatory Roles

It is possible to specify roles as mandatory by naming them in the value of the mandatory_roles system variable. The server treats a mandatory role as granted to all users, so that it need not be granted explicitly to any account.

To specify mandatory roles at server startup, define mandatory_roles in your server my.cnf file:

```
[mysqld]
mandatory_roles='role1,role2@localhost,r3@%.example.com'
```

To set and persist mandatory_roles at runtime, use a statement like this:

```
SET PERSIST mandatory_roles = 'role1,role2@localhost,r3@%.example.com';
```

SET PERSIST sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. To change the value for the running MySQL instance without saving it for subsequent restarts, use the GLOBAL keyword rather than PERSIST. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

Setting mandatory_roles requires the ROLE_ADMIN privilege, in addition to the SYSTEM_VARIABLES_ADMIN or SUPER privilege normally required to set a global system variable.

Mandatory roles, like explicitly granted roles, do not take effect until activated (see Activating Roles). At login time, role activation occurs for all granted roles if the activate_all_roles_on_login system variable is enabled, or for roles that are set as default roles otherwise. At runtime, SET ROLE activates roles.

Roles named in the value of mandatory_roles cannot be revoked with REVOKE or dropped with DROP ROLE or DROP USER.

To prevent sessions from being made system sessions by default, a role that has the SYSTEM_USER privilege cannot be listed in the value of the mandatory_roles system variable:

- If mandatory_roles is assigned a role at startup that has the SYSTEM_USER privilege, the server writes a message to the error log and exits.
- If mandatory_roles is assigned a role at runtime that has the SYSTEM_USER privilege, an error occurs and the mandatory_roles value remains unchanged.

If a role named in `mandatory_roles` is not present in the `mysql.user` system table, the role is not granted to users. When the server attempts role activation for a user, it does not treat the nonexistent role as mandatory and writes a warning to the error log. If the role is created later and thus becomes valid, `FLUSH PRIVILEGES` may be necessary to cause the server to treat it as mandatory.

`SHOW GRANTS` displays mandatory roles according to the rules described in Section 13.7.7.21, “SHOW GRANTS Syntax”.

Checking Role Privileges

To verify the privileges assigned to an account, use `SHOW GRANTS`. For example:

```
mysql> SHOW GRANTS FOR 'dev1'@'localhost';
+-----------------------------------------------+
| Grants for dev1@localhost                     |
+-----------------------------------------------+
| GRANT USAGE ON *.* TO `dev1`@`localhost`      |
| GRANT `app_developer`@`%` TO `dev1`@`localhost` |
+-----------------------------------------------+
```

However, that shows each granted role without “expanding” it to the privileges the role represents. To show role privileges as well, add a `USING` clause naming the granted roles for which to display privileges:

```
mysql> SHOW GRANTS FOR 'dev1'@'localhost' USING 'app_developer';
+-----------------------------------------------------------+
| Grants for dev1@localhost                                 |
+-----------------------------------------------------------+
| GRANT USAGE ON *.* TO `dev1`@`localhost`                  |
| GRANT ALL PRIVILEGES ON `app_db`.* TO `dev1`@`localhost`  |
| GRANT `app_developer`@`%` TO `dev1`@`localhost`           |
+-----------------------------------------------------------+
```

Verify each other type of user similarly:

```
mysql> SHOW GRANTS FOR 'read_user1'@'localhost' USING 'app_read';
+-----------------------------------------------------+
| Grants for read_user1@localhost                     |
+-----------------------------------------------------+
| GRANT USAGE ON *.* TO `read_user1`@`localhost`      |
| GRANT SELECT ON `app_db`.* TO `read_user1`@`localhost` |
| GRANT `app_read`@`%` TO `read_user1`@`localhost`    |
+-----------------------------------------------------+
mysql> SHOW GRANTS FOR 'rw_user1'@'localhost' USING 'app_read', 'app_write';
+-------------------------------------------------------------------------------+
| Grants for rw_user1@localhost                                                 |
+-------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `rw_user1`@`localhost`                                  |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `app_db`.* TO `rw_user1`@`localhost` |
| GRANT `app_read`@`%`,`app_write`@`%` TO `rw_user1`@`localhost`                |
+-------------------------------------------------------------------------------+
```

`SHOW GRANTS` displays mandatory roles according to the rules described in Section 13.7.7.21, “SHOW GRANTS Syntax”.

Activating Roles

Roles granted to a user account can be active or inactive within account sessions. If a granted role is active within a session, its privileges apply; otherwise, they do not. To determine which roles are active within the current session, use the `CURRENT_ROLE()` function.

By default, granting a role to an account or naming it in the `mandatory_roles` system variable value does not automatically cause the role to become active within account sessions. For example, because thus far in the preceding discussion no `rw_user1` roles have been activated, if you connect to the server as `rw_user1` and invoke the `CURRENT_ROLE()` function, the result is `NONE` (no active roles):

```
mysql> SELECT CURRENT_ROLE();
+----------------+
| CURRENT_ROLE() |
+----------------+
| NONE           |
+----------------+
```

To specify which roles should become active each time a user connects to the server and authenticates, use `SET DEFAULT ROLE`. To set the default to all assigned roles for each account created earlier, use this statement:

```
SET DEFAULT ROLE ALL TO
  'dev1'@'localhost',
  'read_user1'@'localhost',
  'read_user2'@'localhost',
  'rw_user1'@'localhost';
```

Now if you connect as `rw_user1`, the initial value of `CURRENT_ROLE()` reflects the new default role assignments:

```
mysql> SELECT CURRENT_ROLE();
+--------------------------------+
| CURRENT_ROLE()                 |
+--------------------------------+
| `app_read`@`%`,`app_write`@`%` |
+--------------------------------+
```

To cause all explicitly granted and mandatory roles to be automatically activated when users connect to the server, enable the `activate_all_roles_on_login` system variable. By default, automatic role activation is disabled.

Within a session, a user can execute `SET ROLE` to change the set of active roles. For example, for `rw_user1`:

```
mysql> SET ROLE NONE; SELECT CURRENT_ROLE();
+----------------+
| CURRENT_ROLE() |
+----------------+
| NONE           |
+----------------+
mysql> SET ROLE ALL EXCEPT 'app_write'; SELECT CURRENT_ROLE();
+----------------+
| CURRENT_ROLE() |
+----------------+
| `app_read`@`%` |
+----------------+
mysql> SET ROLE DEFAULT; SELECT CURRENT_ROLE();
+--------------------------------+
| CURRENT_ROLE()                 |
+--------------------------------+
| `app_read`@`%`,`app_write`@`%` |
+--------------------------------+
```

The first `SET ROLE` statement deactivates all roles. The second makes `rw_user1` effectively read only. The third restores the default roles.

The effective user for stored program and view objects is subject to the `DEFINER` and `SQL SECURITY` attributes, which determine whether execution occurs in invoker or definer context (see Section 24.6, "Stored Object Access Control"):

- Stored program and view objects that execute in invoker context execute with the roles that are active within the current session.
- Stored program and view objects that execute in definer context execute with the default roles of the user named in their `DEFINER` attribute. If `activate_all_roles_on_login` is enabled, such objects execute with all roles granted to the `DEFINER` user, including mandatory roles. For stored programs, if execution should occur with roles different from the default, the program body should execute `SET ROLE` to activate the required roles.

Revoking Roles or Role Privileges

Just as roles can be granted to an account, they can be revoked from an account:

```
REVOKE role FROM user;
```

Roles named in the `mandatory_roles` system variable value cannot be revoked.

`REVOKE` can also be applied to a role to modify the privileges granted to it. This affects not only the role itself, but any account granted that role. Suppose that you want to temporarily make all application users read only. To do this, use `REVOKE` to revoke the modification privileges from the `app_write` role:

```
REVOKE INSERT, UPDATE, DELETE ON app_db.* FROM 'app_write';
```

As it happens, that leaves the role with no privileges at all, as can be seen using `SHOW GRANTS` (which demonstrates that this statement can be used with roles, not just users):

```
mysql> SHOW GRANTS FOR 'app_write';
+---------------------------------------+
| Grants for app_write@%                |
+---------------------------------------+
| GRANT USAGE ON *.* TO `app_write`@`%` |
+---------------------------------------+
```

Because revoking privileges from a role affects the privileges for any user who is assigned the modified role, `rw_user1` now has no table modification privileges (`INSERT`, `UPDATE`, and `DELETE` are no longer present):

```
mysql> SHOW GRANTS FOR 'rw_user1'@'localhost'
       USING 'app_read', 'app_write';
+-------------------------------------------------------------------+
| Grants for rw_user1@localhost                                     |
+-------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `rw_user1`@`localhost`                      |
| GRANT SELECT ON `app_db`.* TO `rw_user1`@`localhost`              |
| GRANT `app_read`@`%`,`app_write`@`%` TO `rw_user1`@`localhost` |
+-------------------------------------------------------------------+
```

In effect, the `rw_user1` read/write user has become a read-only user. This also occurs for any other accounts that are granted the `app_write` role, illustrating how use of roles makes it unnecessary to modify privileges for individual accounts.

To restore modification privileges to the role, simply re-grant them:

```
GRANT INSERT, UPDATE, DELETE ON app_db.* TO 'app_write';
```

Now `rw_user1` again has modification privileges, as do any other accounts granted the `app_write` role.

Dropping Roles

To drop roles, use `DROP ROLE`:

```
DROP ROLE 'app_read', 'app_write';
```

Dropping a role revokes it from every account to which it was granted.

Roles named in the `mandatory_roles` system variable value cannot be dropped.

User and Role Interchangeability

As has been hinted at earlier for `SHOW GRANTS`, which displays grants for user accounts or roles, accounts and roles can be used interchangeably.

One difference between roles and users is that `CREATE ROLE` creates an authorization identifier that is locked by default, whereas `CREATE USER` creates an authorization identifier that is unlocked by default. However, distinction is not immutable because a user with appropriate privileges can lock or unlock roles or users after they have been created.

If a database administrator has a preference that a specific authorization identifier must be a role, a name scheme can be used to communicate this intention. For example, you could use a `r_` prefix for all authorization identifiers that you intend to be roles and nothing else.

Another difference between roles and users lies in the privileges available for administering them:

- The `CREATE ROLE` and `DROP ROLE` privileges enable only use of the `CREATE ROLE` and `DROP ROLE` statements, respectively.
- The `CREATE USER` privilege enables use of the `ALTER USER`, `CREATE ROLE`, `CREATE USER`, `DROP ROLE`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES` statements.

Thus, the `CREATE ROLE` and `DROP ROLE` privileges are not as powerful as `CREATE USER` and may be granted to users who should only be permitted to create and drop roles, and not perform more general account manipulation.

With regard to privileges and interchangeability of users and roles, you can treat a user account like a role and grant that account to another user or a role. The effect is to grant the account's privileges and roles to the other user or role.

This set of statements demonstrates that you can grant a user to a user, a role to a user, a user to a role, or a role to a role:

```
CREATE USER 'u1';
CREATE ROLE 'r1';
GRANT SELECT ON db1.* TO 'u1';
GRANT SELECT ON db2.* TO 'r1';
CREATE USER 'u2';
CREATE ROLE 'r2';
GRANT 'u1', 'r1' TO 'u2';
GRANT 'u1', 'r1' TO 'r2';
```

The result in each case is to grant to the grantee object the privileges associated with the granted object. After executing those statements, each of `u2` and `r2` have been granted privileges from a user (`u1`) and a role (`r1`):

```
mysql> SHOW GRANTS FOR 'u2' USING 'u1', 'r1';
+-----------------------------------+
| Grants for u2@%                   |
+-----------------------------------+
| GRANT USAGE ON *.* TO `u2`@`%`    |
| GRANT SELECT ON `db1`.* TO `u2`@`%` |
| GRANT SELECT ON `db2`.* TO `u2`@`%` |
| GRANT `u1`@`%`,`r1`@`%` TO `u2`@`%` |
+-----------------------------------+
mysql> SHOW GRANTS FOR 'r2' USING 'u1', 'r1';
+-----------------------------------+
| Grants for r2@%                   |
+-----------------------------------+
| GRANT USAGE ON *.* TO `r2`@`%`    |
| GRANT SELECT ON `db1`.* TO `r2`@`%` |
| GRANT SELECT ON `db2`.* TO `r2`@`%` |
| GRANT `u1`@`%`,`r1`@`%` TO `r2`@`%` |
+-----------------------------------+
```

The preceding example is illustrative only, but interchangeability of user accounts and roles has practical application, such as in the following situation: Suppose that a legacy application development project began before the advent of roles in MySQL, so all user accounts associated with the project are granted privileges directly (rather than granted privileges by virtue of being granted roles). One of these accounts is a developer account that was originally granted privileges as follows:

```
CREATE USER 'old_app_dev'@'localhost' IDENTIFIED BY 'old_app_devpass';
GRANT ALL ON old_app.* TO 'old_app_dev'@'localhost';
```

If this developer leaves the project, it becomes necessary to assign the privileges to another user, or perhaps multiple users if development activies have expanded. Here are some ways to deal with the issue:

- Without using roles: Change the account password so the original developer cannot use it, and have a new developer use the account instead:

  ```
  ALTER USER 'old_app_dev'@'localhost' IDENTIFIED BY 'new_password';
  ```

- Using roles: Lock the account to prevent anyone from using it to connect to the server:

  ```
  ALTER USER 'old_app_dev'@'localhost' ACCOUNT LOCK;
  ```

  Then treat the account as a role. For each developer new to the project, create a new account and grant to it the original developer account:

  ```
  CREATE USER 'new_app_dev1'@'localhost' IDENTIFIED BY 'new_password';
  GRANT 'old_app_dev'@'localhost' TO 'new_app_dev1'@'localhost';
  ```

  The effect is to assign the original developer account privileges to the new account.

6.2.11 Account Categories

As of MySQL 8.0.16, MySQL incorporates the concept of user account categories, based on the `SYSTEM_USER` privilege.

- System and Regular Accounts
- Operations Affected by the SYSTEM_USER Privilege
- System and Regular Sessions
- Protecting System Accounts Against Manipulation by Regular Accounts

System and Regular Accounts

MySQL incorporates the concept of user account categories, with system and regular users distinguished according to whether they have the `SYSTEM_USER` privilege:

- A user with the `SYSTEM_USER` privilege is a system user.
- A user without the `SYSTEM_USER` privilege is a regular user.

The `SYSTEM_USER` privilege has an effect on the accounts to which a given user can apply its other privileges, as well as whether the user is protected from other accounts:

- A system user can modify both system and regular accounts. That is, a user who has the appropriate privileges to perform a given operation on regular accounts is enabled by possession of `SYSTEM_USER` to also perform the operation on system accounts. A system account can be modified only by system users with appropriate privileges, not by regular users.
- A regular user with appropriate privileges can modify regular accounts, but not system accounts. A regular account can be modified by both system and regular users with appropriate privileges.

If a user has the appropriate privileges to perform a given operation on regular accounts, `SYSTEM_USER` enables the user to also perform the operation on system accounts. `SYSTEM_USER` does not imply any other privilege, so the ability to perform a given account operation remains predicated on possession of any other required privileges. For example, if a user can grant the `SELECT` and `UPDATE` privileges to regular accounts, then with `SYSTEM_USER` the user can also grant `SELECT` and `UPDATE` to system accounts.

The distinction between system and regular accounts enables better control over certain account administration issues by protecting accounts that have the `SYSTEM_USER` privilege from accounts that do not have the privilege. For example, the `CREATE USER` privilege enables not only creation of new accounts, but modification and removal of existing accounts. Without the system user concept, a user who has the `CREATE USER` privilege can modify or drop any existing account, including the `root` account. The concept of system user enables restricting modifications to the `root` account (itself a system account) so they can be made only by system users. Regular users with the `CREATE USER` privilege can still modify or drop existing accounts, but only regular accounts.

Operations Affected by the SYSTEM_USER Privilege

The `SYSTEM_USER` privilege affects these operations:

- Account manipulation.

  Account manipulation includes creating and dropping accounts, granting and revoking privileges, changing account authentication characteristics such as credentials or authentication plugin, and changing other account characteristics such as password expiration policy.

  The `SYSTEM_USER` privilege is required to manipulate system accounts using account-management statements such as `CREATE USER` and `GRANT`. To prevent an account from modifying system accounts

this way, make it a regular account by not granting it the SYSTEM_USER privilege. (However, to fully protect system accounts against regular accounts, you must also withhold modification privileges for the mysql system schema from regular accounts. See Protecting System Accounts Against Manipulation by Regular Accounts.)

- Killing current sessions and statements executing within them.

  To kill a session or statement that is executing with the SYSTEM_USER privilege, your own session must have the SYSTEM_USER privilege, in addition to any other required privilege (CONNECTION_ADMIN or SUPER).

  Prior to MySQL 8.0.16, CONNECTION_ADMIN or SUPER are sufficient to kill any session or statement.

- Setting the DEFINER for stored objects.

  To set the DEFINER attribute for a stored object to an account that has the SYSTEM_USER privilege, you must have the SYSTEM_USER privilege, in addition to any other required privilege (SET_USER_ID or SUPER).

  Prior to MySQL 8.0.16, SET_USER_ID or SUPER are sufficient to specify any DEFINER value for stored objects.

- Specifying mandatory roles.

  A role that has the SYSTEM_USER privilege cannot be listed in the value of the mandatory_roles system variable.

  Prior to MySQL 8.0.16, any role can be listed in mandatory_roles.

System and Regular Sessions

Sessions executing within the server are distinguished as system or regular sessions, similar to the distinction between system and regular users:

- A session that possesses the SYSTEM_USER privilege is a system session.
- A session that does not possess the SYSTEM_USER privilege is a regular session.

A regular session is able to perform only operations permitted to regular users. A system session is additionally able to perform operations permitted only to system users.

The privileges possessed by a session are those granted directly to its underlying account, plus those granted to all roles currently active within the session. Thus, a session may be a system session because its account has been granted the SYSTEM_USER privilege directly, or because the session has activated a role that has the SYSTEM_USER privilege. Roles granted to an account that are not active within the session do not affect session privileges.

Because activating and deactivating roles can change the privileges possessed by sessions, a session may change from a regular session to a system session or vice versa. If a session activates or deactivates a role that has the SYSTEM_USER privilege, the appropriate change between regular and system session takes place immediately, for that session only:

- If a regular session activates a role with the SYSTEM_USER privilege, the session becomes a system session.
- If a system session deactivates a role with the SYSTEM_USER privilege, the session becomes a regular session, unless some other role with the SYSTEM_USER privilege remains active.

These operations have no effect on existing sessions:

- If the SYSTEM_USER privilege is granted to or revoked from an account, existing sessions for the account do not change between regular and system sessions. The grant or revoke operation affects only sessions for subsequent connections by the account.
- Statements executed by a stored object invoked within a session execute with the system or regular status of the parent session, even if the object DEFINER attribute names a system account.

Because role activation affects only sessions and not accounts, granting a role that has the SYSTEM_USER privilege to a regular account does not protect that account against regular users. The role protects only sessions for the account in which the role has been activated, and protects the session only against being killed by regular sessions.

Protecting System Accounts Against Manipulation by Regular Accounts

Account manipulation includes creating and dropping accounts, granting and revoking privileges, changing account authentication characteristics such as credentials or authentication plugin, and changing other account characteristics such as password expiration policy.

Account manipulation can be done two ways:

- By using account-management statements such as CREATE USER and GRANT. This is the preferred method.
- By direct grant-table modification using statements such as INSERT and UPDATE. This method is discouraged but possible for users with the appropriate privileges on the mysql system schema that contains the grant tables.

To fully protect system accounts against modification by a given account, make it a regular account and do not grant it modification privileges for the mysql schema:

- The SYSTEM_USER privilege is required to manipulate system accounts using account-management statements. To prevent an account from modifying system accounts this way, make it a regular account by not granting SYSTEM_USER to it. This includes not granting SYSTEM_USER to any roles granted to the account.
- Privileges for the mysql schema enable manipulation of system accounts through direct modification of the grant tables, even if the modifying account is a regular account. To restrict unauthorized direct modification of system accounts by a regular account, do not grant modification privileges for the mysql schema to the account (or any roles granted to the account). If a regular account must have global privileges that apply to all schemas, mysql schema modifications can be prevented using privilege restrictions imposed using partial revokes. See Section 6.2.12, “Privilege Restriction Using Partial Revokes”.

Note

Unlike withholding the SYSTEM_USER privilege, which prevents an account from modifying system accounts but not regular accounts, withholding mysql schema privileges prevents an account from modifying system accounts as well as regular accounts. This should not be an issue because, as mentioned, direct grant-table modification is discouraged.

Suppose that you want to create a user u1 who has all privileges on all schemas, except that u1 should be a regular user without the ability to modify system accounts. Assuming that the partial_revokes system variable is enabled, configure u1 as follows:

```
CREATE USER u1 IDENTIFIED BY 'password';

GRANT ALL ON *.* TO u1 WITH GRANT OPTION;
-- GRANT ALL includes SYSTEM_USER, so at this point
-- u1 can manipulate system or regular accounts

REVOKE SYSTEM_USER ON *.* FROM u1;
-- Revoking SYSTEM_USER makes u1 a regular user;
-- now u1 can use account-management statements
-- to manipulate only regular accounts

REVOKE ALL ON mysql.* FROM u1;
-- This partial revoke prevents u1 from directly
-- modifying grant tables to manipulate accounts
```

To prevent all `mysql` system schema access by an account, revoke all its privileges on the `mysql` schema, as just shown. It is also possible to permit partial `mysql` schema access, such as read-only access. The following example creates an account that has `SELECT`, `INSERT`, `UPDATE`, and `DELETE` privileges globally for all schemas, but only `SELECT` for the `mysql` schema:

```
CREATE USER u2 IDENTIFIED BY 'password';
GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u2;
REVOKE INSERT, UPDATE, DELETE ON mysql.* FROM u2;
```

Another possibility is to revoke all `mysql` schema privileges but grant access to specific `mysql` tables or columns. This can be done even with a partial revoke on `mysql`. The following statements enable read-only access to `u1` within the `mysql` schema, but only for the `db` table and the `Host` and `User` columns of the `user` table:

```
CREATE USER u3 IDENTIFIED BY 'password';
GRANT ALL ON *.* TO u3;
REVOKE ALL ON mysql.* FROM u3;
GRANT SELECT ON mysql.db TO u3;
GRANT SELECT(Host,User) ON mysql.user TO u3;
```

6.2.12 Privilege Restriction Using Partial Revokes

Prior to MySQL 8.0.16, it is not possible to grant privileges that apply globally except for certain schemas. As of MySQL 8.0.16, that is possible if the `partial_revokes` system variable is enabled. Specifically, for users who have privileges at the global level, `partial_revokes` enables privileges for specific schemas to be revoked while leaving the privileges in place for other schemas. Privilege restrictions thus imposed may be useful for administration of accounts that have global privileges but should not be permitted to access certain schemas. For example, it is possible to permit an account to modify any table except those in the `mysql` system schema.

- Using Partial Revokes
- Partial Revokes Versus Explicit Schema Grants
- Disabling Partial Revokes
- Partial Revokes and Replication

Note

For brevity, `CREATE USER` statements shown here do not include passwords. For production use, always assign account passwords.

Using Partial Revokes

The `partial_revokes` system variable controls whether privilege restrictions can be placed on accounts. By default, `partial_revokes` is disabled and attempts to partially revoke global privileges produce an error:

```
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT ON *.* TO u1;
mysql> REVOKE INSERT ON world.* FROM u1;
ERROR 1141 (42000): There is no such grant defined for user 'u1' on host '%'
```

To permit the `REVOKE` operation, enable `partial_revokes`:

```
SET PERSIST partial_revokes = ON;
```

`SET PERSIST` sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. To change the value for the running MySQL instance without saving it for subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

With `partial_revokes` enabled, the partial revoke succeeds:

```
mysql> REVOKE INSERT ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------+
| Grants for u1@%                         |
+-----------------------------------------+
| GRANT SELECT, INSERT ON *.* TO `u1`@`%` |
| REVOKE INSERT ON `world`.* FROM `u1`@`%` |
+-----------------------------------------+
```

`SHOW GRANTS` lists partial revokes as `REVOKE` statements in its output. The result indicates that `u1` has global `SELECT` and `INSERT` privileges, except that `INSERT` cannot be exercised for tables in the `world` schema. That is, access by `u1` to `world` tables is read only.

The server records privilege restrictions implemented through partial revokes in the `mysql.user` system table. If an account has partial revokes, its `User_attributes` column value has a `Restrictions` attribute:

```
mysql> SELECT User, Host, User_attributes->>'$.Restrictions'
       FROM mysql.user WHERE User_attributes->>'$.Restrictions' <> '';
+------+------+-------------------------------------------------------+
| User | Host | User_attributes->>'$.Restrictions'                    |
+------+------+-------------------------------------------------------+
| u1   | %    | [{"Database": "world", "Privileges": ["INSERT"]}] |
+------+------+-------------------------------------------------------+
```

Note

Although partial revokes can be imposed for any schema, privilege restrictions on the `mysql` system schema in particular are useful as part of a strategy for preventing regular accounts from modifying system accounts. See Protecting System Accounts Against Manipulation by Regular Accounts.

Partial revoke operations are subject to these conditions:

- Partial revokes must name the schema literally. Schema names that contain the `%` or `_` SQL wildcard characters (for example, `myschema%`) are not permitted.

- It is possible to use partial revokes to place restrictions on nonexistent schemas, but only if the revoked privilege is granted globally. If a privilege is not granted globally, revoking it for a nonexistent schema produces an error.
- Partial revokes apply at the schema level only. You cannot use partial revokes for privileges that apply only globally (such as `FILE` or `BINLOG_ADMIN`), or for table, column, or routine privileges.

As mentioned previously, partial revokes of schema-level privileges appear in `SHOW GRANTS` output as `REVOKE` statements. This differs from how `SHOW GRANTS` represents "plain" schema-level privileges:

- When granted, schema-level privileges are represented by their own `GRANT` statements in the output:

```
mysql> CREATE USER u1;
mysql> GRANT UPDATE ON mysql.* TO u1;
mysql> GRANT DELETE ON world.* TO u1;
mysql> SHOW GRANTS FOR u1;
+--------------------------------------+
| Grants for u1@%                      |
+--------------------------------------+
| GRANT USAGE ON *.* TO `u1`@`%`       |
| GRANT UPDATE ON `mysql`.* TO `u1`@`%` |
| GRANT DELETE ON `world`.* TO `u1`@`%` |
+--------------------------------------+
```

- When revoked, schema-level privileges simply disappear from the output. They do not appear as `REVOKE` statements:

```
mysql> REVOKE UPDATE ON mysql.* FROM u1;
mysql> REVOKE DELETE ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+--------------------------------+
| Grants for u1@%                |
+--------------------------------+
| GRANT USAGE ON *.* TO `u1`@`%` |
+--------------------------------+
```

When a user grants a privilege, any restriction the grantor has on the privilege is inherited by the grantee, unless the grantee already has the privilege without the restriction. Consider the following two users, one of whom has the global `SELECT` privilege:

```
CREATE USER u1, u2;
GRANT SELECT ON *.* TO u2;
```

Suppose that an administrative user `admin` has a global but partially revoked `SELECT` privilege:

```
mysql> CREATE USER admin;
mysql> GRANT SELECT ON *.* TO admin WITH GRANT OPTION;
mysql> REVOKE SELECT ON mysql.* FROM admin;
mysql> SHOW GRANTS FOR admin;
+-----------------------------------------------------+
| Grants for admin@%                                  |
+-----------------------------------------------------+
| GRANT SELECT ON *.* TO `admin`@`%` WITH GRANT OPTION |
| REVOKE SELECT ON `mysql`.* FROM `admin`@`%`         |
+-----------------------------------------------------+
```

If `admin` grants `SELECT` globally to `u1` and `u2`, the result differs for each user:

- If `admin` grants `SELECT` globally to `u1`, who has no `SELECT` privilege to begin with, `u1` inherits the `admin` privilege restriction:

```
mysql> GRANT SELECT ON *.* TO u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------+
| Grants for u1@%                         |
+-----------------------------------------+
| GRANT SELECT ON *.* TO `u1`@`%`         |
| REVOKE SELECT ON `mysql`.* FROM `u1`@`%` |
+-----------------------------------------+
```

- On the other hand, `u2` already holds a global `SELECT` privilege without restriction. `GRANT` can only add to a grantee's existing privileges, not reduce them, so if `admin` grants `SELECT` globally to `u2`, `u2` does not inherit the `admin` restriction:

```
mysql> GRANT SELECT ON *.* TO u2;
mysql> SHOW GRANTS FOR u2;
+---------------------------------+
| Grants for u2@%                 |
+---------------------------------+
| GRANT SELECT ON *.* TO `u2`@`%` |
+---------------------------------+
```

If a `GRANT` statement includes an `AS user` clause, the privilege restrictions applied are those on the user/role combination specified by the clause, rather than those on the user who executes the statement. For information about the `AS` clause, see Section 13.7.1.6, “GRANT Syntax”.

Restrictions on new privileges granted to an account are added to any existing restrictions for that account:

```
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u1;
mysql> REVOKE INSERT ON mysql.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------------------------+
| Grants for u1@%                                           |
+-----------------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%` |
| REVOKE INSERT ON `mysql`.* FROM `u1`@`%`                  |
+-----------------------------------------------------------+
mysql> REVOKE DELETE, UPDATE ON db2.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------------------------+
| Grants for u1@%                                           |
+-----------------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%` |
| REVOKE UPDATE, DELETE ON `db2`.* FROM `u1`@`%`            |
| REVOKE INSERT ON `mysql`.* FROM `u1`@`%`                  |
+-----------------------------------------------------------+
```

Aggregation of privilege restrictions applies both when privileges are partially revoked explicitly (as just shown) and when restrictions are inherited implicitly from the user who executes the statement or the user mentioned in an `AS user` clause.

If an account has a privilege restriction on a schema:

- The account cannot grant to other accounts a privilege on the restricted schema or any object within it.
- Another account that does not have the restriction can grant privileges to the restricted account for the restricted schema or objects within it. Suppose that an unrestricted user executes these statements:

```
CREATE USER u1;
```

```
GRANT SELECT, INSERT, UPDATE ON *.* TO u1;
REVOKE SELECT, INSERT, UPDATE ON mysql.* FROM u1;
GRANT SELECT ON mysql.user TO u1;          -- grant table privilege
GRANT SELECT(Host,User) ON mysql.db TO u1; -- grant column privileges
```

The resulting account has these privileges, with the ability to perform limited operations within the restricted schema:

```
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------------------------+
| Grants for u1@%                                           |
+-----------------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u1`@`%`           |
| REVOKE SELECT, INSERT, UPDATE ON `mysql`.* FROM `u1`@`%`  |
| GRANT SELECT (`Host`, `User`) ON `mysql`.`db` TO `u1`@`%` |
| GRANT SELECT ON `mysql`.`user` TO `u1`@`%`                |
+-----------------------------------------------------------+
```

If an account has a restriction on a global privilege, the restriction is removed by any of these actions:

- Granting the privilege globally to the account by an account that has no restriction on the privilege.
- Granting the privilege at the schema level.
- Revoking the privilege globally.

Consider a user `u1` who holds several privileges globally, but with restrictions on `INSERT`, `UPDATE` and `DELETE`:

```
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u1;
mysql> REVOKE INSERT, UPDATE, DELETE ON mysql.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------------------------+
| Grants for u1@%                                           |
+-----------------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%`   |
| REVOKE INSERT, UPDATE, DELETE ON `mysql`.* FROM `u1`@`%`  |
+-----------------------------------------------------------+
```

Granting a privilege globally to `u1` from an account with no restriction removes the privilege restriction. For example, to remove the `INSERT` restriction:

```
mysql> GRANT INSERT ON *.* TO u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------------------------+
| Grants for u1@%                                           |
+-----------------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%`   |
| REVOKE UPDATE, DELETE ON `mysql`.* FROM `u1`@`%`          |
+-----------------------------------------------------------+
```

Granting a privilege at the schema level to `u1` removes the privilege restriction. For example, to remove the `UPDATE` restriction:

```
mysql> GRANT UPDATE ON mysql.* TO u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------------------------+
| Grants for u1@%                                           |
+-----------------------------------------------------------+
```

```
| GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO `u1`@`%` |
| REVOKE DELETE ON `mysql`.* FROM `u1`@`%`                |
+---------------------------------------------------------+
```

Revoking a global privilege removes the privilege, including any restrictions on it. For example, to remove the `DELETE` restriction (at the cost of removing all `DELETE` access):

```
mysql> REVOKE DELETE ON *.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------------+
| Grants for u1@%                               |
+-----------------------------------------------+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u1`@`%` |
+-----------------------------------------------+
```

If an account has a privilege at both the global and schema levels, you must revoke it at the schema level twice to effect a partial revoke. Suppose that `u1` has these privileges, where `INSERT` is held both globally and on the `world` schema:

```
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT ON *.* TO u1;
mysql> GRANT INSERT ON world.* TO u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------+
| Grants for u1@%                         |
+-----------------------------------------+
| GRANT SELECT, INSERT ON *.* TO `u1`@`%` |
| GRANT INSERT ON `world`.* TO `u1`@`%`   |
+-----------------------------------------+
```

Revoking `INSERT` on `world` revokes the schema-level privilege (`SHOW GRANTS` no longer displays the schema-level `GRANT` statement):

```
mysql> REVOKE INSERT ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------+
| Grants for u1@%                         |
+-----------------------------------------+
| GRANT SELECT, INSERT ON *.* TO `u1`@`%` |
+-----------------------------------------+
```

Revoking `INSERT` on `world` again performs a partial revoke of the global privilege (`SHOW GRANTS` now includes a schema-level `REVOKE` statement):

```
mysql> REVOKE INSERT ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-----------------------------------------+
| Grants for u1@%                         |
+-----------------------------------------+
| GRANT SELECT, INSERT ON *.* TO `u1`@`%`  |
| REVOKE INSERT ON `world`.* FROM `u1`@`%` |
+-----------------------------------------+
```

Partial Revokes Versus Explicit Schema Grants

To provide access to accounts for some schemas but not others, partial revokes provide an alternative to the approach of explicitly granting schema-level access without granting global privileges. The two approaches have different advantages and disadvantages.

Granting schema-level privileges and not global privileges:

- Adding a new schema: The schema is inaccessible to existing accounts by default. For any account to which the schema should be accessible, the DBA must grant schema-level access.
- Adding a new account: The DBA must grant schema-level access for each schema to which the account should have access.

Granting global privileges in conjunction with partial revokes:

- Adding a new schema: The schema is accessible to existing accounts that have global privileges. For any such account to which the schema should be inaccessible, the DBA must add a partial revoke.
- Adding a new account: The DBA must grant the global privileges, plus a partial revoke on each restricted schema.

The approach that uses explicit schema-level grant is more convenient for accounts for which access is limited to a few schemas. The approach that uses partial revokes is more convenient for accounts with broad access to all schemas except a few.

Disabling Partial Revokes

Once enabled, `partial_revokes` cannot be disabled if any account has privilege restrictions. If any such account exists, disabling `partial_revokes` fails:

- For attempts to disable `partial_revokes` at startup, the server logs an error message and enables `partial_revokes`.
- For attempts to disable `partial_revokes` at runtime, an error occurs and the `partial_revokes` value remains unchanged.

To disable `partial_revokes` when restrictions exist, the restrictions first must be removed:

1. Determine which accounts have partial revokes:

   ```
   SELECT User, Host, User_attributes->>'$.Restrictions'
   FROM mysql.user WHERE User_attributes->>'$.Restrictions' <> '';
   ```

2. For each such account, remove its privilege restrictions. Suppose that the previous step shows account `u1` to have these restrictions:

   ```
   [{"Database": "world", "Privileges": ["INSERT", "DELETE"]
   ```

   Restriction removal can be done various ways:

   - Grant the privileges globally, without restrictions:

     ```
     GRANT INSERT, DELETE ON *.* TO u1;
     ```

   - Grant the privileges at the schema level:

     ```
     GRANT INSERT, DELETE ON world.* TO u1;
     ```

   - Revoke the privileges globally (assuming that they are no longer needed):

     ```
     REVOKE INSERT, DELETE ON *.* FROM u1;
     ```

   - Remove the account itself (assuming that it is no longer needed):

```
DROP USER u1;
```

After all privilege restrictions are removed, it is possible to disable partial revokes:

```
SET PERSIST partial_revokes = OFF;
```

Partial Revokes and Replication

In replication scenarios, if `partial_revokes` is enabled on any host, it must be enabled on all hosts. Otherwise, `REVOKE` statements to partially revoke a global privilege do not have the same effect for all hosts on which replication occurs, potentially resulting in replication inconsistencies or errors.

6.2.13 When Privilege Changes Take Effect

If the `mysqld` server is started without the `--skip-grant-tables` option, it reads all grant table contents into memory during its startup sequence. The in-memory tables become effective for access control at that point.

If you modify the grant tables indirectly using an account-management statement, the server notices these changes and loads the grant tables into memory again immediately. Account-management statements are described in Section 13.7.1, “Account Management Statements”. Examples include `GRANT`, `REVOKE`, `SET PASSWORD`, and `RENAME USER`.

If you modify the grant tables directly using statements such as `INSERT`, `UPDATE`, or `DELETE` (which is not recommended), the changes have no effect on privilege checking until you either tell the server to reload the tables or restart it. Thus, if you change the grant tables directly but forget to reload them, the changes have *no effect* until you restart the server. This may leave you wondering why your changes seem to make no difference!

To tell the server to reload the grant tables, perform a flush-privileges operation. This can be done by issuing a `FLUSH PRIVILEGES` statement or by executing a `mysqladmin flush-privileges` or `mysqladmin reload` command.

A grant table reload affects privileges for each existing client session as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect the next time the client executes a `USE db_name` statement.

Note

Client applications may cache the database name; thus, this effect may not be visible to them without actually changing to a different database.

- Global privileges and passwords are unaffected for a connected client. These changes take effect only in sessions for subsequent connections.

Changes to the set of active roles within a session take effect immediately, for that session only. The `SET ROLE` statement performs session role activation and deactivation (see Section 13.7.1.11, “SET ROLE Syntax”).

If the server is started with the `--skip-grant-tables` option, it does not read the grant tables or implement any access control. Any user can connect and perform any operation, *which is insecure.* To cause a server thus started to read the tables and enable access checking, flush the privileges.

6.2.14 Assigning Account Passwords

Required credentials for clients that connect to the MySQL server can include a password. This section describes how to assign passwords for MySQL accounts.

MySQL stores credentials in the `user` table in the `mysql` system database. Operations that assign or modify passwords are permitted only to users with the `CREATE USER` privilege, or, alternatively, privileges for the `mysql` database (`INSERT` privilege to create new accounts, `UPDATE` privilege to modify existing accounts). If the `read_only` system variable is enabled, use of account-modification statements such as `CREATE USER` or `ALTER USER` additionally requires the `CONNECTION_ADMIN` or `SUPER` privilege.

The discussion here summarizes syntax only for the most common password-assignment statements. For complete details on other possibilities, see Section 13.7.1.3, “CREATE USER Syntax”, Section 13.7.1.1, “ALTER USER Syntax”, and Section 13.7.1.10, “SET PASSWORD Syntax”.

MySQL uses plugins to perform client authentication; see Section 6.2.17, “Pluggable Authentication”. In password-assigning statements, the authentication plugin associated with an account performs any hashing required of a cleartext password specified. This enables MySQL to obfuscate passwords prior to storing them in the `mysql.user` system table. For the statements described here, MySQL automatically hashes the password specified. There are also syntax for `CREATE USER` and `ALTER USER` that permits hashed values to be specified literally. For details, see the descriptions of those statements.

To assign a password when you create a new account, use `CREATE USER` and include an `IDENTIFIED BY` clause:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

`CREATE USER` also supports syntax for specifying the account authentication plugin. See Section 13.7.1.3, “CREATE USER Syntax”.

To assign or change a password for an existing account, use the `ALTER USER` statement with an `IDENTIFIED BY` clause:

```
ALTER USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

If you are not connected as an anonymous user, you can change your own password without naming your own account literally:

```
ALTER USER USER() IDENTIFIED BY 'password';
```

To change an account password from the command line, use the `mysqladmin` command:

```
mysqladmin -u user_name -h host_name password "password"
```

The account for which this command sets the password is the one with a row in the `mysql.user` system table that matches *`user_name`* in the `User` column and the client host *from which you connect* in the `Host` column.

Warning

Setting a password using `mysqladmin` should be considered *insecure*. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their

initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

If you are using MySQL Replication, be aware that, currently, a password used by a replication slave as part of a `CHANGE MASTER TO` statement is effectively limited to 32 characters in length; if the password is longer, any excess characters are truncated. This is not due to any limit imposed by the MySQL Server generally, but rather is an issue specific to MySQL Replication. (For more information, see Bug #43439.)

6.2.15 Password Management

MySQL supports these password-management capabilities:

- Password expiration, to require passwords to be changed periodically.
- Password reuse restrictions, to prevent old passwords from being chosen again.
- Password verification, to require that password changes also specify the current password to be replaced.
- Dual passwords, to enable clients to connect using either a primary or secondary password.
- Password strength assessment, to require strong passwords.
- Random password generation, as an alternative to requiring explicit administrator-specified literal passwords.

The following sections these capabilities, except password strength assessment, which is implemented using the `validate_password` plugin and is described in Section 6.4.3, “The Password Validation Component”.

- Internal Versus External Credentials Storage
- Password Expiration Policy
- Password Reuse Policy
- Password Verification-Required Policy
- Dual Password Support
- Random Password Generation

Important

MySQL implements password-management capabilities using tables in the `mysql` system database. If you upgrade MySQL from an earlier version, your system tables might not be up to date. In that case, the server writes messages similar to these to the error log during the startup process (the exact numbers may vary):

```
[ERROR] Column count of mysql.user is wrong. Expected
49, found 47. The table is probably corrupted
[Warning] ACL table mysql.password_history missing.
Some operations may fail.
```

To correct the issue, perform the MySQL upgrade procedure. See Section 2.11, “Upgrading MySQL”. Until this is done, *password changes are not possible.*

Internal Versus External Credentials Storage

Some authentication plugins store account credentials internally to MySQL, in the `mysql.user` system table:

- `mysql_native_password`
- `caching_sha2_password`
- `sha256_password`

The discussion in this section applies to such authentication plugins because the password-management capabilities described here are handled by MySQL itself.

Other authentication plugins store account credentials externally to MySQL. For accounts that use plugins that perform authentication against an external credentials system, password management must be handled externally against that system as well.

For information about individual authentication plugins, see Section 6.4.1, “Authentication Plugins”.

Password Expiration Policy

MySQL enables database administrators to expire account passwords manually, and to establish a policy for automatic password expiration. Expiration policy can be established globally, and individual accounts can be set to either defer to the global policy or override the global policy with specific per-account behavior.

To expire an account password manually, use the `ALTER USER` statement:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE;
```

This operation marks the password expired in the corresponding row in the `mysql.user` system table.

Password expiration according to policy is automatic and is based on password age, which for a given account is assessed from the date and time of its most recent password change. The `mysql.user` system table indicates for each account when its password was last changed, and the server automatically treats the password as expired at client connection time if its age is greater than its permitted lifetime. This works with no explicit manual password expiration.

To establish automatic password-expiration policy globally, use the `default_password_lifetime` system variable. Its default value is 0, which disables automatic password expiration. If the value of `default_password_lifetime` is a positive integer *`N`*, it indicates the permitted password lifetime, such that passwords must be changed every *`N`* days.

Examples:

- To establish a global policy that passwords have a lifetime of approximately six months, start the server with these lines in a server `my.cnf` file:

  ```
  [mysqld]
  default_password_lifetime=180
  ```

- To establish a global policy such that passwords never expire, set `default_password_lifetime` to 0:

  ```
  [mysqld]
  default_password_lifetime=0
  ```

- `default_password_lifetime` can also be set and persisted at runtime:

```
SET PERSIST default_password_lifetime = 180;
SET PERSIST default_password_lifetime = 0;
```

  `SET PERSIST` sets the value for the running MySQL instance. It also saves the value to be used for subsequent server restarts; see Section 13.7.6.1, “SET Syntax for Variable Assignment”. To change the value for the running MySQL instance without saving it for subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`.

The global password-expiration policy applies to all accounts that have not been set to override it. To establish policy for individual accounts, use the `PASSWORD EXPIRE` option of the `CREATE USER` and `ALTER USER` statements. See Section 13.7.1.3, “CREATE USER Syntax”, and Section 13.7.1.1, “ALTER USER Syntax”.

Example account-specific statements:

- Require the password to be changed every 90 days:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 90 DAY;
```

  This expiration option overrides the global policy for all accounts named by the statement.

- Disable password expiration:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
```

  This expiration option overrides the global policy for all accounts named by the statement.

- Defer to the global expiration policy for all accounts named by the statement:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
```

When a client successfully connects, the server determines whether the account password has expired:

- The server checks whether the password has been manually expired.
- Otherwise, the server checks whether the password age is greater than its permitted lifetime according to the automatic password expiration policy. If so, the server considers the password expired.

If the password is expired (whether manually or automatically), the server either disconnects the client or restricts the operations permitted to it (see Section 6.2.16, “Server Handling of Expired Passwords”). Operations performed by a restricted client result in an error until the user establishes a new account password:

```
mysql> SELECT 1;
ERROR 1820 (HY000): You must reset your password using ALTER USER
statement before executing this statement.

mysql> ALTER USER USER() IDENTIFIED BY 'password';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT 1;
+---+
```

```
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

After the client resets the password, the server restores normal access for the session, as well as for subsequent connections that use the account. It is also possible for an administrative user to reset the account password, but any existing restricted sessions for that account remain restricted. A client using the account must disconnect and reconnect before statements can be executed successfully.

Note

Although it is possible to “reset” an expired password by setting it to its current value, it is preferable, as a matter of good policy, to choose a different password. DBAs can enforce non-reuse by establishing an appropriate password-reuse policy. See Password Reuse Policy.

Password Reuse Policy

MySQL enables restrictions to be placed on reuse of previous passwords. Reuse restrictions can be established based on number of password changes, time elapsed, or both. Reuse policy can be established globally, and individual accounts can be set to either defer to the global policy or override the global policy with specific per-account behavior.

The password history for an account consists of passwords it has been assigned in the past. MySQL can restrict new passwords from being chosen from this history:

- If an account is restricted on the basis of number of password changes, a new password cannot be chosen from a specified number of the most recent passwords. For example, if the minimum number of password changes is set to 3, a new password cannot be the same as any of the most recent 3 passwords.
- If an account is restricted based on time elapsed, a new password cannot be chosen from passwords in the history that are newer than a specified number of days. For example, if the password reuse interval is set to 60, a new password must not be among those previously chosen within the last 60 days.

Note

The empty password does not count in the password history and is subject to reuse at any time.

To establish password-reuse policy globally, use the `password_history` and `password_reuse_interval` system variables.

Examples:

- To prohibit reusing any of the last 6 passwords or passwords newer than 365 days, put these lines in the server `my.cnf` file:

```
[mysqld]
password_history=6
password_reuse_interval=365
```

- To set and persist the variables at runtime, use statements like this:

```
SET PERSIST password_history = 6;
```

```
SET PERSIST password_reuse_interval = 365;
```

`SET PERSIST` sets the value for the running MySQL instance. It also saves the value to be used for subsequent server restarts; see Section 13.7.6.1, “SET Syntax for Variable Assignment”. To change the value for the running MySQL instance without saving it for subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`.

The global password-reuse policy applies to all accounts that have not been set to override it. To establish policy for individual accounts, use the `PASSWORD HISTORY` and `PASSWORD REUSE INTERVAL` options of the `CREATE USER` and `ALTER USER` statements. See Section 13.7.1.3, “CREATE USER Syntax”, and Section 13.7.1.1, “ALTER USER Syntax”.

Example account-specific statements:

- Require a minimum of 5 password changes before permitting reuse:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD HISTORY 5;
ALTER USER 'jeffrey'@'localhost' PASSWORD HISTORY 5;
```

  This history-length option overrides the global policy for all accounts named by the statement.

- Require a minimum of 365 days elapsed before permitting reuse:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL 365 DAY;
ALTER USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL 365 DAY;
```

  This time-elapsed option overrides the global policy for all accounts named by the statement.

- To combine both types of reuse restrictions, use `PASSWORD HISTORY` and `PASSWORD REUSE INTERVAL` together:

```
CREATE USER 'jeffrey'@'localhost'
  PASSWORD HISTORY 5
  PASSWORD REUSE INTERVAL 365 DAY;
ALTER USER 'jeffrey'@'localhost'
  PASSWORD HISTORY 5
  PASSWORD REUSE INTERVAL 365 DAY;
```

  These options override both global policy reuse restrictions for all accounts named by the statement.

- Defer to the global policy for both types of reuse restrictions:

```
CREATE USER 'jeffrey'@'localhost'
  PASSWORD HISTORY DEFAULT
  PASSWORD REUSE INTERVAL DEFAULT;
ALTER USER 'jeffrey'@'localhost'
  PASSWORD HISTORY DEFAULT
  PASSWORD REUSE INTERVAL DEFAULT;
```

Password Verification-Required Policy

As of MySQL 8.0.13, it is possible to require that attempts to change an account password be verified by specifying the current password to be replaced. This enables DBAs to prevent users from changing a password without proving that they know the current password. Such changes could otherwise occur, for example, if one user walks away from a terminal session temporarily without logging out, and a malicious user uses the session to change the original user's MySQL password. This can have unfortunate consequences:

- The original user becomes unable to access MySQL until the account password is reset by an administrator.
- Until the password reset occurs, the malicious user can access MySQL with the benign user's changed credentials.

Password-verification policy can be established globally, and individual accounts can be set to either defer to the global policy or override the global policy with specific per-account behavior.

For each account, its `mysql.user` row indicates whether there is an account-specific setting requiring verification of the current password for password change attempts. The setting is established by the `PASSWORD REQUIRE` option of the `CREATE USER` and `ALTER USER` statements:

- If the account setting is `PASSWORD REQUIRE CURRENT`, password changes must specify the current password.
- If the account setting is `PASSWORD REQUIRE CURRENT OPTIONAL`, password changes may but need not specify the current password.
- If the account setting is `PASSWORD REQUIRE CURRENT DEFAULT`, the `password_require_current` system variable determines the verification-required policy for the account:
  - If `password_require_current` is enabled, password changes must specify the current password.
  - If `password_require_current` is disabled, password changes may but need not specify the current password.

In other words, if the account setting is not `PASSWORD REQUIRE CURRENT DEFAULT`, the account setting takes precedence over the global policy established by the `password_require_current` system variable. Otherwise, the account defers to the `password_require_current` setting.

By default, password verification is optional: `password_require_current` is disabled and accounts created with no `PASSWORD REQUIRE` option default to `PASSWORD REQUIRE CURRENT DEFAULT`.

The following table shows how per-account settings interact with `password_require_current` system variable values to determine account password verification-required policy.

Table 6.10 Password-Verification Policy

Per-Account Setting	password_require_current System Variable	Password Changes Require Current Password?
`PASSWORD REQUIRE CURRENT`	`OFF`	Yes
`PASSWORD REQUIRE CURRENT`	`ON`	Yes
`PASSWORD REQUIRE CURRENT OPTIONAL`	`OFF`	No
`PASSWORD REQUIRE CURRENT OPTIONAL`	`ON`	No
`PASSWORD REQUIRE CURRENT DEFAULT`	`OFF`	No
`PASSWORD REQUIRE CURRENT DEFAULT`	`ON`	Yes

Note

Privileged users can change any account password without specifying the current password, regardless of the verification-required policy. A privileged user is one

who has the global `CREATE USER` privilege or the `UPDATE` privilege for the `mysql` system database.

To establish password-verification policy globally, use the `password_require_current` system variable. Its default value is `OFF`, so it is not required that account password changes specify the current password.

Examples:

- To establish a global policy that password changes must specify the current password, start the server with these lines in a server `my.cnf` file:

```
[mysqld]
password_require_current=ON
```

- To set and persist `password_require_current` at runtime, use a statement such as one of these:

```
SET PERSIST password_require_current = ON;
SET PERSIST password_require_current = OFF;
```

  `SET PERSIST` sets the value for the running MySQL instance. It also saves the value to be used for subsequent server restarts; see Section 13.7.6.1, “SET Syntax for Variable Assignment”. To change the value for the running MySQL instance without saving it for subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`.

The global password verification-required policy applies to all accounts that have not been set to override it. To establish policy for individual accounts, use the `PASSWORD REQUIRE` options of the `CREATE USER` and `ALTER USER` statements. See Section 13.7.1.3, “CREATE USER Syntax”, and Section 13.7.1.1, “ALTER USER Syntax”.

Example account-specific statements:

- Require that password changes specify the current password:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT;
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT;
```

  This verification option overrides the global policy for all accounts named by the statement.

- Do not require that password changes specify the current password (the current password may but need not be given):

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT OPTIONAL;
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT OPTIONAL;
```

  This verification option overrides the global policy for all accounts named by the statement.

- Defer to the global password verification-required policy for all accounts named by the statement:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT DEFAULT;
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT DEFAULT;
```

Verification of the current password comes into play when a user changes a password using the `ALTER USER` or `SET PASSWORD` statement. The examples use `ALTER USER`, which is preferred over `SET PASSWORD`, but the principles described here are the same for both statements.

In password-change statements, a REPLACE clause specifies the current password to be replaced. Examples:

- Change the current user's password:

```
ALTER USER USER() IDENTIFIED BY 'auth_string' REPLACE 'current_auth_string';
```

- Change a named user's password:

```
ALTER USER 'jeffrey'@'localhost'
  IDENTIFIED BY 'auth_string'
  REPLACE 'current_auth_string';
```

- Change a named user's authentication plugin and password:

```
ALTER USER 'jeffrey'@'localhost'
  IDENTIFIED WITH caching_sha2_password BY 'auth_string'
  REPLACE 'current_auth_string';
```

The REPLACE clause works like this:

- REPLACE must be given if password changes for the account are required to specify the current password, as verification that the user attempting to make the change actually knows the current password.
- REPLACE is optional if password changes for the account may but need not specify the current password.
- If REPLACE is specified, it must specify the correct current password, or an error occurs. This is true even if REPLACE is optional.
- REPLACE can be specified only when changing the account password for the current user. (This means that in the examples just shown, the statements that explicitly name the account for jeffrey fail unless the current user is jeffrey.) This is true even if the change is attempted for another user by a privileged user; however, such a user can change any password without specifying REPLACE.
- REPLACE is omitted from the binary log to avoid writing cleartext passwords to it.

Dual Password Support

As of MySQL 8.0.14, user accounts are permitted to have dual passwords, designated as primary and secondary passwords. Dual-password capability makes it possible to seamlessly perform credential changes in scenarios like this:

- A system has a large number of MySQL servers, possibly involving replication.
- Multiple applications connect to different MySQL servers.
- Periodic credential changes must be made to the account or accounts used by the applications to connect to the servers.

Consider how a credential change must be performed in the preceding type of scenario when an account is permitted only a single password. In this case, there must be close cooperation in the timing of when the account password change is made and propagated throughout all servers, and when all applications that use the account are updated to use the new password. This process may involve downtime during which servers or applications are unavailable.

With dual passwords, credential changes can be made more easily, in phases, without requiring close cooperation, and without downtime:

1. For each affected account, establish a new primary password on the servers, retaining the current password as the secondary password. This enables servers to recognize either the primary or secondary password for each account, while applications can continue to connect to the servers using the same password as previously (which is now the secondary password).

2. After the password change has propagated to all servers, modify applications that use any affected account to connect using the account primary password.

3. After all applications have been migrated from the secondary passwords to the primary passwords, the secondary passwords are no longer needed and can be discarded. After this change has propagated to all servers, only the primary password for each account can be used to connect. The credential change is now complete.

MySQL implements dual-password capability with syntax that saves and discards secondary passwords:

- The `RETAIN CURRENT PASSWORD` clause for the `ALTER USER` and `SET PASSWORD` statements saves an account current password as its secondary password when you assign a new primary password.
- The `DISCARD OLD PASSWORD` clause for `ALTER USER` discards an account secondary password, leaving only the primary password.

Suppose that, for the previously described credential-change scenario, an account named `'appuser1'@'host1.example.com'` is used by applications to connect to servers, and that the account password is to be changed from `'password_a'` to `'password_b'`.

To perform this change of credentials, use `ALTER USER` as follows:

1. On each server that is not a replication slave, establish `'password_b'` as the new `appuser1` primary password, retaining the current password as the secondary password:

   ```
   ALTER USER 'appuser1'@'host1.example.com'
     IDENTIFIED BY 'password_b'
     RETAIN CURRENT PASSWORD;
   ```

2. Wait for the password change to replicate throughout the system to all slave servers.

3. Modify each application that uses the `appuser1` account so that it connects to the servers using a password of `'password_b'` rather than `'password_a'`.

4. At this point, the secondary password is no longer needed. On each server that is not a replication slave, discard the secondary password:

   ```
   ALTER USER 'appuser1'@'host1.example.com'
     DISCARD OLD PASSWORD;
   ```

5. After the discard-password change has replicated to all slave servers, the credential change is complete.

The `RETAIN CURRENT PASSWORD` and `DISCARD OLD PASSWORD` clauses have the following effects:

- `RETAIN CURRENT PASSWORD` retains an account current password as its secondary password, replacing any existing secondary password. The new password becomes the primary password, but clients can use the account to connect to the server using either the primary or secondary password.

(Exception: If the new password specified by the `ALTER USER` or `SET PASSWORD` statement is empty, the secondary password becomes empty as well, even if `RETAIN CURRENT PASSWORD` is given.)

- If you specify `RETAIN CURRENT PASSWORD` for an account that has an empty primary password, the statement fails.
- If an account has a secondary password and you change its primary password without specifying `RETAIN CURRENT PASSWORD`, the secondary password remains unchanged.
- For `ALTER USER`, if you change the authentication plugin assigned to the account, the secondary password is discarded. If you change the authentication plugin and also specify `RETAIN CURRENT PASSWORD`, the statement fails.
- For `ALTER USER`, `DISCARD OLD PASSWORD` discards the secondary password, if one exists. The account retains only its primary password, and clients can use the account to connect to the server only with the primary password.

Statements that modify secondary passwords require these privileges:

- The `APPLICATION_PASSWORD_ADMIN` privilege is required to use the `RETAIN CURRENT PASSWORD` or `DISCARD OLD PASSWORD` clause for `ALTER USER` and `SET PASSWORD` statements that apply to your own account. The privilege is required to manipulate your own secondary password because most users require only one password.
- If an account is to be permitted to manipulate secondary passwords for all accounts, it should be granted the `CREATE USER` privilege rather than `APPLICATION_PASSWORD_ADMIN`.

Random Password Generation

As of MySQL 8.0.18, the `CREATE USER`, `ALTER USER`, and `SET PASSWORD` statements have the capability of generating random passwords for user accounts, as an alternative to requiring explicit administrator-specified literal passwords. See the description of each statement for details about the syntax. This section describes the characteristics common to generated random passwords.

By default, generated random passwords have a length of 20 characters. This length is controlled by the `generated_random_password_length` system variable, which has a range from 5 to 255.

For each account for which a statement generates a random password, the statement stores the password in the `mysql.user` system table, hashed appropriately for the account authentication plugin. The statement also returns the cleartext password in a row of a result set to make it available to the user or application executing the statement. The result set columns are named `user`, `host`, and `generated password`, indicating the user name and host name values that identify the affected row in the `mysql.user` system table, and the cleartext generated password.

```
mysql> CREATE USER
       'u1'@'localhost' IDENTIFIED BY RANDOM PASSWORD,
       'u2'@'%.example.com' IDENTIFIED BY RANDOM PASSWORD,
       'u3'@'%.org' IDENTIFIED BY RANDOM PASSWORD;
+------+--------------+----------------------+
| user | host         | generated password   |
+------+--------------+----------------------+
| u1   | localhost    | BA;42VpXqQ@i+y{&TDFF |
| u2   | %.example.com | YX5>XRAJRP@>sn9azmD4 |
| u3   | %.org        | ;GfD44l,)C}PI/6)4TwZ |
+------+--------------+----------------------+
mysql> ALTER USER
       'u1'@'localhost' IDENTIFIED BY RANDOM PASSWORD,
       'u2'@'%.example.com' IDENTIFIED BY RANDOM PASSWORD;
+------+--------------+----------------------+
```

```
| user | host          | generated password   |
+------+---------------+----------------------+
| u1   | localhost     | yhXBrBp.;Y6abB)e_UWr |
| u2   | %.example.com | >M-vmjp9DTY6}hkp,RcC |
+------+---------------+----------------------+
mysql> SET PASSWORD FOR 'u3'@'%.org' TO RANDOM;
+------+-------+----------------------+
| user | host  | generated password   |
+------+-------+----------------------+
| u3   | %.org | o(._oNn)d;FC<vJIDg9M |
+------+-------+----------------------+
```

A `CREATE USER`, `ALTER USER`, or `SET PASSWORD` statement that generates a random password for an account is written to the binary log as a `CREATE USER` or `ALTER USER` statement with an `IDENTIFIED WITH auth_plugin AS 'auth_string'`, clause, where *`auth_plugin`* is the account authentication plugin and *`'auth_string'`* is the account hashed password value.

If the `validate_password` component is installed, the policy that it implements has no effect on generated passwords. (The purpose of password validation is to help humans create better passwords.)

6.2.16 Server Handling of Expired Passwords

MySQL provides password-expiration capability, which enables database administrators to require that users reset their password. Passwords can be expired manually, and on the basis of a policy for automatic expiration (see Section 6.2.15, “Password Management”).

The `ALTER USER` statement enables account password expiration. For example:

```
ALTER USER 'myuser'@'localhost' PASSWORD EXPIRE;
```

For each connection that uses an account with an expired password, the server either disconnects the client or restricts the client to “sandbox mode,” in which the server permits the client to perform only those operations necessary to reset the expired password. Which action is taken by the server depends on both client and server settings, as discussed later.

If the server disconnects the client, it returns an `ER_MUST_CHANGE_PASSWORD_LOGIN` error:

```
shell> mysql -u myuser -p
Password: ******
ERROR 1862 (HY000): Your password has expired. To log in you must
change it using a client that supports expired passwords.
```

If the server restricts the client to sandbox mode, these operations are permitted within the client session:

- The client can reset the account password with `ALTER USER` or `SET PASSWORD`. After that has been done, the server restores normal access for the session, as well as for subsequent connections that use the account.

  Note

  Although it is possible to “reset” an expired password by setting it to its current value, it is preferable, as a matter of good policy, to choose a different password. DBAs can enforce non-reuse by establishing an appropriate password-reuse policy. See Password Reuse Policy.

- The client can use the `SET` statement.

For any operation not permitted within the session, the server returns an `ER_MUST_CHANGE_PASSWORD` error:

```
mysql> USE performance_schema;
ERROR 1820 (HY000): You must reset your password using ALTER USER
statement before executing this statement.

mysql> SELECT 1;
ERROR 1820 (HY000): You must reset your password using ALTER USER
statement before executing this statement.
```

That is what normally happens for interactive invocations of the `mysql` client because by default such invocations are put in sandbox mode. To resume normal functioning, select a new password.

For noninteractive invocations of the `mysql` client (for example, in batch mode), the server normally disconnects the client if the password is expired. To permit noninteractive `mysql` invocations to stay connected so that the password can be changed (using the statements permitted in sandbox mode), add the `--connect-expired-password` option to the `mysql` command.

As mentioned previously, whether the server disconnects an expired-password client or restricts it to sandbox mode depends on a combination of client and server settings. The following discussion describes the relevant settings and how they interact.

Note

This discussion applies only for accounts with expired passwords. If a client connects using a nonexpired password, the server handles the client normally.

On the client side, a given client indicates whether it can handle sandbox mode for expired passwords. For clients that use the C client library, there are two ways to do this:

- Pass the `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` flag to `mysql_options()` prior to connecting:

  ```
  bool arg = 1;
  mysql_options(mysql,
                MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS,
                &arg);
  ```

  This is the technique used within the `mysql` client, which enables `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` if invoked interactively or with the `--connect-expired-password` option.

- Pass the `CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS` flag to `mysql_real_connect()` at connect time:

  ```
  MYSQL mysql;
  mysql_init(&mysql);
  if (!mysql_real_connect(&mysql,
                          host, user, password, db,
                          port, unix_socket,
                          CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS))
  {
    ... handle error ...
  }
  ```

Other MySQL Connectors have their own conventions for indicating readiness to handle sandbox mode. See the documentation for the Connector in which you are interested.

On the server side, if a client indicates that it can handle expired passwords, the server puts it in sandbox mode.

If a client does not indicate that it can handle expired passwords (or uses an older version of the client library that cannot so indicate), the server action depends on the value of the `disconnect_on_expired_password` system variable:

- If `disconnect_on_expired_password` is enabled (the default), the server disconnects the client with an `ER_MUST_CHANGE_PASSWORD_LOGIN` error.
- If `disconnect_on_expired_password` is disabled, the server puts the client in sandbox mode.

6.2.17 Pluggable Authentication

When a client connects to the MySQL server, the server uses the user name provided by the client and the client host to select the appropriate account row from the `mysql.user` system table. The server then authenticates the client, determining from the account row which authentication plugin applies to the client:

- If the server cannot find the plugin, an error occurs and the connection attempt is rejected.
- Otherwise, the server invokes that plugin to authenticate the user, and the plugin returns a status to the server indicating whether the user provided the correct password and is permitted to connect.

Pluggable authentication enables these important capabilities:

- **Choice of authentication methods.** Pluggable authentication makes it easy for DBAs to choose and change the authentication method used for individual MySQL accounts.
- **External authentication.** Pluggable authentication makes it possible for clients to connect to the MySQL server with credentials appropriate for authentication methods that store credentials elsewhere than in the `mysql.user` system table. For example, plugins can be created to use external authentication methods such as PAM, Windows login IDs, LDAP, or Kerberos.
- **Proxy users:** If a user is permitted to connect, an authentication plugin can return to the server a user name different from the name of the connecting user, to indicate that the connecting user is a proxy for another user (the proxied user). While the connection lasts, the proxy user is treated, for purposes of access control, as having the privileges of the proxied user. In effect, one user impersonates another. For more information, see Section 6.2.18, “Proxy Users”.

Note

If you start the server with the `--skip-grant-tables` option, authentication plugins are not used even if loaded because the server performs no client authentication and permits any client to connect. Because this is insecure, if the server is started with the `--skip-grant-tables` option, it enables `--skip-networking` automatically to prevent remote connections.

- Available Authentication Plugins
- Authentication Plugin Usage
- Authentication Plugin Client/Server Compatibility
- Authentication Plugin Connector-Writing Considerations
- Restrictions on Pluggable Authentication

Available Authentication Plugins

MySQL 8.0 provides these authentication plugins:

- A plugin that performs native authentication; that is, authentication based on the password hashing method in use from before the introduction of pluggable authentication in MySQL. The `mysql_native_password` plugin implements authentication based on this native password hashing method. See Section 6.4.1.1, “Native Pluggable Authentication”.
- Plugins that perform authentication using SHA-256 password hashing. This is stronger encryption than that available with native authentication. See Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.
- A client-side plugin that sends the password to the server without hashing or encryption. This plugin is used in conjunction with server-side plugins that require access to the password exactly as provided by the client user. See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.
- A plugin that performs external authentication using PAM (Pluggable Authentication Modules), enabling MySQL Server to use PAM to authenticate MySQL users. This plugin supports proxy users as well. See Section 6.4.1.5, “PAM Pluggable Authentication”.
- A plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password. This plugin supports proxy users as well. See Section 6.4.1.6, “Windows Pluggable Authentication”.
- Plugins that perform authentication using LDAP (Lightweight Directory Access Protocol) to authenticate MySQL users by accessing directory services such as X.500. These plugins support proxy users as well. See Section 6.4.1.7, “LDAP Pluggable Authentication”.
- A plugin that prevents all client connections to any account that uses it. Use cases for this plugin include proxied accounts that should never permit direct login but are accessed only through proxy accounts and accounts that must be able to execute stored programs and views with elevated privileges without exposing those privileges to ordinary users. See Section 6.4.1.8, “No-Login Pluggable Authentication”.
- A plugin that authenticates clients that connect from the local host through the Unix socket file. See Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”.
- A test plugin that checks account credentials and logs success or failure to the server error log. This plugin is intended for testing and development purposes, and as an example of how to write an authentication plugin. See Section 6.4.1.10, “Test Pluggable Authentication”.

Note

For information about current restrictions on the use of pluggable authentication, including which connectors support which plugins, see Restrictions on Pluggable Authentication.

Third-party connector developers should read that section to determine the extent to which a connector can take advantage of pluggable authentication capabilities and what steps to take to become more compliant.

If you are interested in writing your own authentication plugins, see Section 29.2.4.9, “Writing Authentication Plugins”.

Authentication Plugin Usage

This section provides general instructions for installing and using authentication plugins. For instructions specific to a given plugin, see the section that describes that plugin under Section 6.4.1, “Authentication Plugins”.

In general, pluggable authentication uses a pair of corresponding plugins on the server and client sides, so you use a given authentication method like this:

- If necessary, install the plugin library or libraries containing the appropriate plugins. On the server host, install the library containing the server-side plugin, so that the server can use it to authenticate client connections. Similarly, on each client host, install the library containing the client-side plugin for use by client programs. Authentication plugins that are built in need not be installed.
- For each MySQL account that you create, specify the appropriate server-side plugin to use for authentication. If the account is to use the default authentication plugin, the account-creation statement need not specify the plugin explicitly. The `default_authentication_plugin` system variable configures the default authentication plugin.
- When a client connects, the server-side plugin tells the client program which client-side plugin to use for authentication.

In the case that an account uses an authentication method that is the default for both the server and the client program, the server need not communicate to the client which client-side plugin to use, and a round trip in client/server negotiation can be avoided.

For standard MySQL clients such as `mysql` and `mysqladmin`, the `--default-auth=`*`plugin_name`* option can be specified on the command line as a hint about which client-side plugin the program can expect to use, although the server will override this if the server-side plugin associated with the user account requires a different client-side plugin.

If the client program does not find the client-side plugin library file, specify a `--plugin-dir=`*`dir_name`* option to indicate the plugin library directory location.

Authentication Plugin Client/Server Compatibility

Pluggable authentication enables flexibility in the choice of authentication methods for MySQL accounts, but in some cases client connections cannot be established due to authentication plugin incompatibility between the client and server.

The general compatibility principle for a successful client connection to a given account on a given server is that the client and server both must support the authentication *method* required by the account. Because authentication methods are implemented by authentication plugins, the client and server both must support the authentication *plugin* required by the account.

Authentication plugin incompatibilities can arise in various ways. Examples:

- Connect using a MySQL 5.7 client from 5.7.22 or lower to a MySQL 8.0 server account that authenticates with `caching_sha2_password`. This fails because the 5.7 client does not recognize the plugin, which was introduced in MySQL 8.0. (This issue is addressed in MySQL 5.7 as of 5.7.23, when `caching_sha2_password` client-side support was added to the MySQL client library and client programs.)
- Connect using a MySQL 5.5 client to a MySQL 5.6 server account that authenticates with `sha256_password`. This fails because the 5.5 client does not recognize the plugin, which was introduced in MySQL 5.6.
- Connect using a MySQL 5.7 client to a pre-5.7 server account that authenticates with `mysql_old_password`. This fails for multiple reasons. First, such a connection requires `--secure-auth=0`, which is no longer a supported option. Even were it supported, the 5.7 client does not recognize the plugin because it was removed in MySQL 5.7.

- Connect using a MySQL 5.7 client from a Community distribution to a MySQL 5.7 Enterprise server account that authenticates using one of the Enterprise-only LDAP authentication plugins. This fails because the Community client does not have access to the Enterprise plugin.

In general, these compatibility issues do not arise when connections are made between a client and server from the same MySQL distribution. When connections are made between a client and server from different MySQL series, issues can arise. These issues are inherent in the development process when MySQL introduces new authentication plugins or removes old ones. To minimize the potential for incompatibilities, regularly upgrade the server, clients, and connectors on a timely basis.

Authentication Plugin Connector-Writing Considerations

Various implementations of the MySQL client/server protocol exist. The `libmysqlclient` C API client library is one implementation. Some MySQL connectors (typically those not written in C) provide their own implementation. However, not all protocol implementations handle plugin authentication the same way. This section describes an authentication issue that protocol implementors should take into account.

In the client/server protocol, the server tells connecting clients which authentication plugin it considers the default. If the protocol implementation used by the client tries to load the default plugin and that plugin does not exist on the client side, the load operation fails. This is an unnecessary failure if the default plugin is not the plugin actually required by the account to which the client is trying to connect.

If a client/server protocol implementation does not have its own notion of default authentication plugin and always tries to load the default plugin specified by the server, it will fail with an error if that plugin is not available.

To avoid this problem, the protocol implementation used by the client should have its own default plugin and should use it as its first choice (or, alternatively, fall back to this default in case of failure to load the default plugin specified by the server). Example:

- In MySQL 5.7, `libmysqlclient` uses as its default choice either `mysql_native_password` or the plugin specified through the `MYSQL_DEFAULT_AUTH` option for `mysql_options()`.
- When a 5.7 client tries to connect to an 8.0 server, the server specifies `caching_sha2_password` as its default authentication plugin, but the client still sends credential details per either `mysql_native_password` or whatever is specified through `MYSQL_DEFAULT_AUTH`.
- The only time the client loads the plugin specified by the server is for a change-plugin request, but in that case it can be any plugin depending on the user account. In this case, the client must try to load the plugin, and if that plugin is not available, an error is not optional.

Restrictions on Pluggable Authentication

The first part of this section describes general restrictions on the applicability of the pluggable authentication framework described at Section 6.2.17, “Pluggable Authentication”. The second part describes how third-party connector developers can determine the extent to which a connector can take advantage of pluggable authentication capabilities and what steps to take to become more compliant.

The term “native authentication” used here refers to authentication against passwords stored in the `mysql.user` system table. This is the same authentication method provided by older MySQL servers, before pluggable authentication was implemented. “Windows native authentication” refers to authentication using the credentials of a user who has already logged in to Windows, as implemented by the Windows Native Authentication plugin (“Windows plugin” for short).

- General Pluggable Authentication Restrictions
- Pluggable Authentication and Third-Party Connectors

General Pluggable Authentication Restrictions

- **Connector/C, Connector/C++:** Clients that use these connectors can connect to the server only through accounts that use native authentication.

  Exception: A connector supports pluggable authentication if it was built to link to `libmysqlclient` dynamically (rather than statically) and it loads the current version of `libmysqlclient` if that version is installed, or if the connector is recompiled from source to link against the current `libmysqlclient`.

  For information about writing connectors to handle informatin from the server about the default server-side authentication plugin, see Authentication Plugin Connector-Writing Considerations.

- **Connector/NET:** Clients that use Connector/NET can connect to the server through accounts that use native authentication or Windows native authentication.

- **Connector/PHP:** Clients that use this connector can connect to the server only through accounts that use native authentication, when compiled using the MySQL native driver for PHP (`mysqlnd`).

- **Windows native authentication:** Connecting through an account that uses the Windows plugin requires Windows Domain setup. Without it, NTLM authentication is used and then only local connections are possible; that is, the client and server must run on the same computer.

- **Proxy users:** Proxy user support is available to the extent that clients can connect through accounts authenticated with plugins that implement proxy user capability (that is, plugins that can return a user name different from that of the connecting user). For example, the PAM and Windows plugins support proxy users. The `mysql_native_password` and `sha256_password` authentication plugins do not support proxy users by default, but can be configured to do so; see Server Support for Proxy User Mapping.

- **Replication**: Replication slaves can employ not only master accounts using native authentication, but can also connect through master accounts that use nonnative authentication if the required client-side plugin is available. If the plugin is built into `libmysqlclient`, it is available by default. Otherwise, the plugin must be installed on the slave side in the directory named by the slave `plugin_dir` system variable.

- **`FEDERATED` tables:** A `FEDERATED` table can access the remote table only through accounts on the remote server that use native authentication.

Pluggable Authentication and Third-Party Connectors

Third-party connector developers can use the following guidelines to determine readiness of a connector to take advantage of pluggable authentication capabilities and what steps to take to become more compliant:

- An existing connector to which no changes have been made uses native authentication and clients that use the connector can connect to the server only through accounts that use native authentication. *However, you should test the connector against a recent version of the server to verify that such connections still work without problem.*

  Exception: A connector might work with pluggable authentication without any changes if it links to `libmysqlclient` dynamically (rather than statically) and it loads the current version of `libmysqlclient` if that version is installed.

- To take advantage of pluggable authentication capabilities, a connector that is `libmysqlclient`-based should be relinked against the current version of `libmysqlclient`. This enables the connector to support connections though accounts that require client-side plugins now built into `libmysqlclient` (such as the cleartext plugin needed for PAM authentication and the Windows plugin needed for Windows native authentication). Linking with a current `libmysqlclient` also enables the connector to

access client-side plugins installed in the default MySQL plugin directory (typically the directory named by the default value of the local server's `plugin_dir` system variable).

If a connector links to `libmysqlclient` dynamically, it must be ensured that the newer version of `libmysqlclient` is installed on the client host and that the connector loads it at runtime.

- Another way for a connector to support a given authentication method is to implement it directly in the client/server protocol. Connector/NET uses this approach to provide support for Windows native authentication.
- If a connector should be able to load client-side plugins from a directory different from the default plugin directory, it must implement some means for client users to specify the directory. Possibilities for this include a command-line option or environment variable from which the connector can obtain the directory name. Standard MySQL client programs such as `mysql` and `mysqladmin` implement a `--plugin-dir` option. See also Section 28.7.17, “C API Client Plugin Functions”.
- Proxy user support by a connector depends, as described earlier in this section, on whether the authentication methods that it supports permit proxy users.

6.2.18 Proxy Users

The MySQL server authenticates client connections using authentication plugins. The plugin that authenticates a given connection may request that the connecting (external) user be treated as a different user for privilege-checking purposes. This enables the external user to be a proxy for the second user; that is, to assume the privileges of the second user:

- The external user is a “proxy user” (a user who can impersonate or become known as another user).
- The second user is a “proxied user” (a user whose identity and privileges can be assumed by a proxy user).

This section describes how the proxy user capability works. For general information about authentication plugins, see Section 6.2.17, “Pluggable Authentication”. For information about specific plugins, see Section 6.4.1, “Authentication Plugins”. For information about writing authentication plugins that support proxy users, see Implementing Proxy User Support in Authentication Plugins.

- Requirements for Proxy User Support
- Simple Proxy User Example
- Preventing Direct Login to Proxied Accounts
- Granting and Revoking the PROXY Privilege
- Default Proxy Users
- Default Proxy User and Anonymous User Conflicts
- Server Support for Proxy User Mapping
- Proxy User System Variables

Note

One administrative benefit to be gained by proxying is that the DBA can set up a single account with a set of privileges and then enable multiple proxy users to have those privileges without having to assign the privileges individually to each of those users. As an alternative to proxy users, DBAs may find that roles provide a suitable

way to map users onto specific sets of named privileges. Each user can be granted a given single role to, in effect, be granted the appropriate set of privileges. See Section 6.2.10, “Using Roles”.

Requirements for Proxy User Support

For proxying to occur for a given authentication plugin, these conditions must be satisfied:

- Proxying must be supported, either by the plugin itself, or by the MySQL server on behalf of the plugin. In the latter case, server support may need to be enabled explicitly; see Server Support for Proxy User Mapping.
- The account for the external proxy user must be set up to be authenticated by the plugin. Use the `CREATE USER` statement to associate an account with an authentication plugin, or `ALTER USER` to change its plugin.
- The account for the proxied user must exist and be granted the privileges to be assumed by the proxy user. Use the `CREATE USER` and `GRANT` statements for this.
- Normally, the proxied user is configured so that it can be used only in proxying scenaries and not for direct logins.
- The proxy user account must have the `PROXY` privilege for the proxied account. Use the `GRANT` statement for this.
- For a client connecting to the proxy account to be treated as a proxy user, the authentication plugin must return a user name different from the client user name, to indicate the user name of the proxied account that defines the privileges to be assumed by the proxy user.

  Alternatively, for plugins that are provided proxy mapping by the server, the proxied user is determined from the `PROXY` privilege held by the proxy user.

The proxy mechanism permits mapping only the external client user name to the proxied user name. There is no provision for mapping host names:

- When a client connects to the server, the server determines the proper account based on the user name passed by the client program and the host from which the client connects.
- If that account is a proxy account, the server attempts to determine the appropriate proxied account by finding a match for a proxied account using the user name returned by the authentication plugin and the host name of the proxy account. The host name in the proxied account is ignored.

Simple Proxy User Example

Consider the following account definitions:

```
-- create proxy account
CREATE USER 'employee_ext'@'localhost'
  IDENTIFIED WITH my_auth_plugin
  AS 'my_auth_string';

-- create proxied account and grant its privileges;
-- use mysql_no_login plugin to prevent direct login
CREATE USER 'employee'@'localhost'
  IDENTIFIED WITH mysql_no_login;
GRANT ALL
  ON employees.*
  TO 'employee'@'localhost';
```

```
-- grant to proxy account the
-- PROXY privilege for proxied account
GRANT PROXY
  ON 'employee'@'localhost'
  TO 'employee_ext'@'localhost';
```

When a client connects as `employee_ext` from the local host, MySQL uses the plugin named `my_auth_plugin` to perform authentication. Suppose that `my_auth_plugin` returns a user name of `employee` to the server, based on the content of '*my_auth_string*' and perhaps by consulting some external authentication system. The name `employee` differs from `employee_ext`, so returning `employee` serves as a request to the server to treat the `employee_ext` external user, for purposes of privilege checking, as the `employee` local user.

In this case, `employee_ext` is the proxy user and `employee` is the proxied user.

The server verifies that proxy authentication for `employee` is possible for the `employee_ext` user by checking whether `employee_ext` (the proxy user) has the `PROXY` privilege for `employee` (the proxied user). If this privilege has not been granted, an error occurs. Otherwise, `employee_ext` assumes the privileges of `employee`. The server checks statements executed during the client session by `employee_ext` against the privileges granted to `employee`. In this case, `employee_ext` can access tables in the `employees` database.

The proxied account, `employee`, uses the `mysql_no_login` authentication plugin to prevent clients from using the account to log in directly. (This assumes that the plugin is installed. For instructions, see Section 6.4.1.8, “No-Login Pluggable Authentication”.) For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

When proxying occurs, the `USER()` and `CURRENT_USER()` functions can be used to see the difference between the connecting user (the proxy user) and the account whose privileges apply during the current session (the proxied user). For the example just described, those functions return these values:

```
mysql> SELECT USER(), CURRENT_USER();
+------------------------+--------------------+
| USER()                 | CURRENT_USER()     |
+------------------------+--------------------+
| employee_ext@localhost | employee@localhost |
+------------------------+--------------------+
```

In the `CREATE USER` statement that creates the proxy user account, the `IDENTIFIED WITH` clause that names the proxy-supporting authentication plugin is optionally followed by an `AS 'auth_string'` clause specifying a string that the server passes to the plugin when the user connects. If present, the string provides information that helps the plugin determine how to map the proxy (external) client user name to a proxied user name. It is up to each plugin whether it requires the `AS` clause. If so, the format of the authentication string depends on how the plugin intends to use it. Consult the documentation for a given plugin for information about the authentication string values it accepts.

Preventing Direct Login to Proxied Accounts

Proxied accounts generally are intended to be used only by means of proxy accounts. That is, clients connect using a proxy account, then are mapped onto and assume the privileges of the appropriate proxied user.

There are multiple ways to ensure that a proxied account cannot be used directly:

- Associate the account with the `mysql_no_login` authentication plugin. In this case, the account cannot be used for direct logins under any circumstances. This assumes that the plugin is installed. For instructions, see Section 6.4.1.8, “No-Login Pluggable Authentication”.

- Include the ACCOUNT LOCK option when you create the account. See Section 13.7.1.3, “CREATE USER Syntax”. With this method, also include a password so that if the account is unlocked later, it cannot be accessed with no password. (If the validate_password component is enabled, it will not permit creating an account without a password, even if the account is locked. See Section 6.4.3, “The Password Validation Component”.)
- Create the account with a password but do not tell anyone else the password. If you do not let anyone know the password for the account, clients cannot use it to connect directly to the MySQL server.

Granting and Revoking the PROXY Privilege

The PROXY privilege is needed to enable an external user to connect as and have the privileges of another user. To grant this privilege, use the GRANT statement. For example:

```
GRANT PROXY ON 'proxied_user' TO 'proxy_user';
```

The statement creates a row in the mysql.proxies_priv grant table.

At connect time, *proxy_user* must represent a valid externally authenticated MySQL user, and *proxied_user* must represent a valid locally authenticated user. Otherwise, the connection attempt fails.

The corresponding REVOKE syntax is:

```
REVOKE PROXY ON 'proxied_user' FROM 'proxy_user';
```

MySQL GRANT and REVOKE syntax extensions work as usual. Examples:

```
-- grant PROXY to multiple accounts
GRANT PROXY ON 'a' TO 'b', 'c', 'd';

-- revoke PROXY from multiple accounts
REVOKE PROXY ON 'a' FROM 'b', 'c', 'd';

-- grant PROXY to an account and enable the account to grant
-- PROXY to the proxied account
GRANT PROXY ON 'a' TO 'd' WITH GRANT OPTION;

-- grant PROXY to default proxy account
GRANT PROXY ON 'a' TO ''@'';
```

The PROXY privilege can be granted in these cases:

- By a user that has GRANT PROXY ... WITH GRANT OPTION for *proxied_user*.
- By *proxied_user* for itself: The value of USER() must exactly match CURRENT_USER() and *proxied_user*, for both the user name and host name parts of the account name.

The initial root account created during MySQL installation has the PROXY ... WITH GRANT OPTION privilege for ''@'', that is, for all users and all hosts. This enables root to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. For example, root can do this:

```
CREATE USER 'admin'@'localhost'
  IDENTIFIED BY 'admin_password';
GRANT PROXY
  ON ''@''
  TO 'admin'@'localhost'
  WITH GRANT OPTION;
```

Those statements create an `admin` user that can manage all `GRANT PROXY` mappings. For example, `admin` can do this:

```
GRANT PROXY ON sally TO joe;
```

Default Proxy Users

To specify that some or all users should connect using a given authentication plugin, create a “blank” MySQL account with an empty user name and host name (`''@''`), associate it with that plugin, and let the plugin return the real authenticated user name (if different from the blank user). Suppose that there exists a plugin named `ldap_auth` that implements LDAP authentication and maps connecting users onto either a developer or manager account. To set up proxying of users onto these accounts, use the following statements:

```
-- create default proxy account
CREATE USER ''@''
  IDENTIFIED WITH ldap_auth
  AS 'O=Oracle, OU=MySQL';

-- create proxied accounts; use
-- mysql_no_login plugin to prevent direct login
CREATE USER 'developer'@'localhost'
  IDENTIFIED WITH mysql_no_login;
CREATE USER 'manager'@'localhost'
  IDENTIFIED WITH mysql_no_login;

-- grant to default proxy account the
-- PROXY privilege for proxied accounts
GRANT PROXY
  ON 'manager'@'localhost'
  TO ''@'';
GRANT PROXY
  ON 'developer'@'localhost'
  TO ''@'';
```

Now assume that a client connects as follows:

```
shell> mysql --user=myuser --password ...
Enter password: myuser_password
```

The server will not find `myuser` defined as a MySQL user. But because there is a blank user account (`''@''`) that matches the client user name and host name, the server authenticates the client against that account: The server invokes the `ldap_auth` authentication plugin and passes `myuser` and *`myuser_password`* to it as the user name and password.

If the `ldap_auth` plugin finds in the LDAP directory that *`myuser_password`* is not the correct password for `myuser`, authentication fails and the server rejects the connection.

If the password is correct and `ldap_auth` finds that `myuser` is a developer, it returns the user name `developer` to the MySQL server, rather than `myuser`. Returning a user name different from the client user name of `myuser` signals to the server that it should treat `myuser` as a proxy. The server verifies that `''@''` can authenticate as `developer` (because `''@''` has the `PROXY` privilege to do so) and accepts the connection. The session proceeds with `myuser` having the privileges of the `developer` proxied user. (These privileges should be set up by the DBA using `GRANT` statements, not shown.) The `USER()` and `CURRENT_USER()` functions return these values:

```
mysql> SELECT USER(), CURRENT_USER();
+------------------+--------------------+
```

```
| USER()           | CURRENT_USER()      |
+------------------+---------------------+
| myuser@localhost | developer@localhost |
+------------------+---------------------+
```

If the plugin instead finds in the LDAP directory that `myuser` is a manager, it returns `manager` as the user name and the session proceeds with `myuser` having the privileges of the `manager` proxied user.

```
mysql> SELECT USER(), CURRENT_USER();
+------------------+------------------+
| USER()           | CURRENT_USER()   |
+------------------+------------------+
| myuser@localhost | manager@localhost |
+------------------+------------------+
```

For simplicity, external authentication cannot be multilevel: Neither the credentials for `developer` nor those for `manager` are taken into account in the preceding example. However, they are still used if a client tries to connect and authenticate directly as the `developer` or `manager` account, which is why those proxied accounts should be protected against direct login (see Preventing Direct Login to Proxied Accounts).

Default Proxy User and Anonymous User Conflicts

If you intend to create a default proxy user, check for other existing “match any user” accounts that take precedence over the default proxy user because they can prevent that user from working as intended.

In the preceding discussion, the default proxy user account has `''` in the host part, which matches any host. If you set up a default proxy user, take care to also check whether nonproxy accounts exist with the same user part and `'%'` in the host part, because `'%'` also matches any host, but has precedence over `''` by the rules that the server uses to sort account rows internally (see Section 6.2.6, “Access Control, Stage 1: Connection Verification”).

Suppose that a MySQL installation includes these two accounts:

```
-- create default proxy account
CREATE USER ''@''
  IDENTIFIED WITH some_plugin
  AS 'some_auth_string';
-- create anonymous account
CREATE USER ''@'%'
  IDENTIFIED BY 'anon_user_password';
```

The first account (`''@''`) is intended as the default proxy user, used to authenticate connections for users who do not otherwise match a more-specific account. The second account (`''@'%'`) is an anonymous-user account, which might have been created, for example, to enable users without their own account to connect anonymously.

Both accounts have the same user part (`''`), which matches any user. And each account has a host part that matches any host. Nevertheless, there is a priority in account matching for connection attempts because the matching rules sort a host of `'%'` ahead of `''`. For accounts that do not match any more-specific account, the server attempts to authenticate them against `''@'%'` (the anonymous user) rather than `''@''` (the default proxy user). As a result, the default proxy account is never used.

To avoid this problem, use one of the following strategies:

- Remove the anonymous account so that it does not conflict with the default proxy user.
- Use a more-specific default proxy user that matches ahead of the anonymous user. For example, to permit only `localhost` proxy connections, use `''@'localhost'`:

```
CREATE USER ''@'localhost'
  IDENTIFIED WITH some_plugin
  AS 'some_auth_string';
```

In addition, modify any `GRANT PROXY` statements to name `''@'localhost'` rather than `''@''` as the proxy user.

Be aware that this strategy prevents anonymous-user connections from `localhost`.

- Use a named default account rather than an anonymous default account. For an example of this technique, consult the instructions for using the `authentication_windows` plugin. See Section 6.4.1.6, “Windows Pluggable Authentication”.

- Create multiple proxy users, one for local connections and one for “everything else” (remote connections). This can be useful particularly when local users should have different privileges from remote users.

  Create the proxy users:

  ```
  -- create proxy user for local connections
  CREATE USER ''@'localhost'
    IDENTIFIED WITH some_plugin
    AS 'some_auth_string';
  -- create proxy user for remote connections
  CREATE USER ''@'%'
    IDENTIFIED WITH some_plugin
    AS 'some_auth_string';
  ```

  Create the proxied users:

  ```
  -- create proxied user for local connections
  CREATE USER 'developer'@'localhost'
    IDENTIFIED WITH mysql_no_login;
  -- create proxied user for remote connections
  CREATE USER 'developer'@'%'
    IDENTIFIED WITH mysql_no_login;
  ```

  Grant to each proxy account the `PROXY` privilege for the corresponding proxied account:

  ```
  GRANT PROXY
    ON 'developer'@'localhost'
    TO ''@'localhost';
  GRANT PROXY
    ON 'developer'@'%'
    TO ''@'%';
  ```

  Finally, grant appropriate privileges to the local and remote proxied users (not shown).

  Assume that the `some_plugin`/`'some_auth_string'` combination causes `some_plugin` to map the client user name to `developer`. Local connections match the `''@'localhost'` proxy user, which maps to the `'developer'@'localhost'` proxied user. Remote connections match the `''@'%'` proxy user, which maps to the `'developer'@'%'` proxied user.

Server Support for Proxy User Mapping

Some authentication plugins implement proxy user mapping for themselves (for example, the PAM and Windows authentication plugins). Other authentication plugins do not support proxy users by default.

Of these, some can request that the MySQL server itself map proxy users according to granted proxy privileges: `mysql_native_password`, `sha256_password`. If the `check_proxy_users` system variable is enabled, the server performs proxy user mapping for any authentication plugins that make such a request:

- By default, `check_proxy_users` is disabled, so the server performs no proxy user mapping even for authentication plugins that request server support for proxy users.
- If `check_proxy_users` is enabled, it may also be necessary to enable a plugin-specific system variable to take advantage of server proxy user mapping support:
  - For the `mysql_native_password` plugin, enable `mysql_native_password_proxy_users`.
  - For the `sha256_password` plugin, enable `sha256_password_proxy_users`.

Proxy user mapping performed by the server is subject to these restrictions:

- The server will not proxy to or from an anonymous user, even if the associated `PROXY` privilege is granted.
- When a single account has been granted proxy privileges for more than one proxied account, server proxy user mapping is nondeterministic. Therefore, granting to a single account proxy privileges for multiple proxied accounts is discouraged.

Proxy User System Variables

Two system variables help trace the proxy login process:

- `proxy_user`: This value is `NULL` if proxying is not used. Otherwise, it indicates the proxy user account. For example, if a client authenticates through the `''@''` proxy account, this variable is set as follows:

```
mysql> SELECT @@proxy_user;
+--------------+
| @@proxy_user |
+--------------+
| ''@''        |
+--------------+
```

- `external_user`: Sometimes the authentication plugin may use an external user to authenticate to the MySQL server. For example, when using Windows native authentication, a plugin that authenticates using the windows API does not need the login ID passed to it. However, it still uses a Windows user ID to authenticate. The plugin may return this external user ID (or the first 512 UTF-8 bytes of it) to the server using the `external_user` read-only session variable. If the plugin does not set this variable, its value is `NULL`.

6.2.19 Account Locking

MySQL supports locking and unlocking user accounts using the `ACCOUNT LOCK` and `ACCOUNT UNLOCK` clauses for the `CREATE USER` and `ALTER USER` statements:

- When used with `CREATE USER`, these clauses specify the initial locking state for a new account. In the absence of either clause, the account is created in an unlocked state.

  If the `validate_password` component is enabled, it will not permit creating an account without a password, even if the account is locked. See Section 6.4.3, “The Password Validation Component”.

- When used with `ALTER USER`, these clauses specify the new locking state for an existing account. In the absence of either clause, the account locking state remains unchanged.

Account locking state is recorded in the `account_locked` column of the `mysql.user` system table. The output from `SHOW CREATE USER` indicates whether an account is locked or unlocked.

If a client attempts to connect to a locked account, the attempt fails. The server increments the `Locked_connects` status variable that indicates the number of attempts to connect to a locked account, returns an `ER_ACCOUNT_HAS_BEEN_LOCKED` error, and writes a message to the error log:

```
Access denied for user 'user_name'@'host_name'.
Account is locked.
```

Locking an account does not affect being able to connect using a proxy user that assumes the identity of the locked account. It also does not affect the ability to execute stored programs or views that have a `DEFINER` clause naming the locked account. That is, the ability to use a proxied account or stored programs or views is not affected by locking the account.

The account-locking capability depends on the presence of the `account_locked` column in the `mysql.user` system table. For upgrades from MySQL versions older than 5.7.6, perform the MySQL upgrade procedure to ensure that this column exists. See Section 2.11, “Upgrading MySQL”. For nonupgraded installations that have no `account_locked` column, the server treats all accounts as unlocked, and using the `ACCOUNT LOCK` or `ACCOUNT UNLOCK` clauses produces an error.

6.2.20 Setting Account Resource Limits

One means of restricting client use of MySQL server resources is to set the global `max_user_connections` system variable to a nonzero value. This limits the number of simultaneous connections that can be made by any given account, but places no limits on what a client can do once connected. In addition, setting `max_user_connections` does not enable management of individual accounts. Both types of control are of interest to MySQL administrators.

To address such concerns, MySQL permits limits for individual accounts on use of these server resources:

- The number of queries an account can issue per hour
- The number of updates an account can issue per hour
- The number of times an account can connect to the server per hour
- The number of simultaneous connections to the server by an account

Any statement that a client can issue counts against the query limit. Only statements that modify databases or tables count against the update limit.

An “account” in this context corresponds to a row in the `mysql.user` system table. That is, a connection is assessed against the `User` and `Host` values in the `user` table row that applies to the connection. For example, an account `'usera'@'%.example.com'` corresponds to a row in the `user` table that has `User` and `Host` values of `usera` and `%.example.com`, to permit `usera` to connect from any host in the `example.com` domain. In this case, the server applies resource limits in this row collectively to all connections by `usera` from any host in the `example.com` domain because all such connections use the same account.

Before MySQL 5.0, an “account” was assessed against the actual host from which a user connects. This older method of accounting may be selected by starting the server with the `--old-style-user-limits` option. In this case, if `usera` connects simultaneously from `host1.example.com` and `host2.example.com`, the server applies the account resource limits separately to each connection. If `usera` connects again from `host1.example.com`, the server applies the limits for that connection together with the existing connection from that host.

To establish resource limits for an account at account-creation time, use the `CREATE USER` statement. To modify the limits for an existing account, use `ALTER USER`. Provide a `WITH` clause that names each resource to be limited. The default value for each limit is zero (no limit). For example, to create a new account that can access the `customer` database, but only in a limited fashion, issue these statements:

```
mysql> CREATE USER 'francis'@'localhost' IDENTIFIED BY 'frank'
    ->     WITH MAX_QUERIES_PER_HOUR 20
    ->          MAX_UPDATES_PER_HOUR 10
    ->          MAX_CONNECTIONS_PER_HOUR 5
    ->          MAX_USER_CONNECTIONS 2;
```

The limit types need not all be named in the `WITH` clause, but those named can be present in any order. The value for each per-hour limit should be an integer representing a count per hour. For `MAX_USER_CONNECTIONS`, the limit is an integer representing the maximum number of simultaneous connections by the account. If this limit is set to zero, the global `max_user_connections` system variable value determines the number of simultaneous connections. If `max_user_connections` is also zero, there is no limit for the account.

To modify limits for an existing account, use an `ALTER USER` statement. The following statement changes the query limit for `francis` to 100:

```
mysql> ALTER USER 'francis'@'localhost' WITH MAX_QUERIES_PER_HOUR 100;
```

The statement modifies only the limit value specified and leaves the account otherwise unchanged.

To remove a limit, set its value to zero. For example, to remove the limit on how many times per hour `francis` can connect, use this statement:

```
mysql> ALTER USER 'francis'@'localhost' WITH MAX_CONNECTIONS_PER_HOUR 0;
```

As mentioned previously, the simultaneous-connection limit for an account is determined from the `MAX_USER_CONNECTIONS` limit and the `max_user_connections` system variable. Suppose that the global `max_user_connections` value is 10 and three accounts have individual resource limits specified as follows:

```
ALTER USER 'user1'@'localhost' WITH MAX_USER_CONNECTIONS 0;
ALTER USER 'user2'@'localhost' WITH MAX_USER_CONNECTIONS 5;
ALTER USER 'user3'@'localhost' WITH MAX_USER_CONNECTIONS 20;
```

`user1` has a connection limit of 10 (the global `max_user_connections` value) because it has a `MAX_USER_CONNECTIONS` limit of zero. `user2` and `user3` have connection limits of 5 and 20, respectively, because they have nonzero `MAX_USER_CONNECTIONS` limits.

The server stores resource limits for an account in the `user` table row corresponding to the account. The `max_questions`, `max_updates`, and `max_connections` columns store the per-hour limits, and the `max_user_connections` column stores the `MAX_USER_CONNECTIONS` limit. (See Section 6.2.3, “Grant Tables”.)

Resource-use counting takes place when any account has a nonzero limit placed on its use of any of the resources.

As the server runs, it counts the number of times each account uses resources. If an account reaches its limit on number of connections within the last hour, the server rejects further connections for the account until that hour is up. Similarly, if the account reaches its limit on the number of queries or updates,

the server rejects further queries or updates until the hour is up. In all such cases, the server issues appropriate error messages.

Resource counting occurs per account, not per client. For example, if your account has a query limit of 50, you cannot increase your limit to 100 by making two simultaneous client connections to the server. Queries issued on both connections are counted together.

The current per-hour resource-use counts can be reset globally for all accounts, or individually for a given account:

- To reset the current counts to zero for all accounts, issue a `FLUSH USER_RESOURCES` statement. The counts also can be reset by reloading the grant tables (for example, with a `FLUSH PRIVILEGES` statement or a `mysqladmin reload` command).
- The counts for an individual account can be reset to zero by setting any of its limits again. Specify a limit value equal to the value currently assigned to the account.

Per-hour counter resets do not affect the `MAX_USER_CONNECTIONS` limit.

All counts begin at zero when the server starts. Counts do not carry over through server restarts.

For the `MAX_USER_CONNECTIONS` limit, an edge case can occur if the account currently has open the maximum number of connections permitted to it: A disconnect followed quickly by a connect can result in an error (`ER_TOO_MANY_USER_CONNECTIONS` or `ER_USER_LIMIT_REACHED`) if the server has not fully processed the disconnect by the time the connect occurs. When the server finishes disconnect processing, another connection will once more be permitted.

6.2.21 Troubleshooting Problems Connecting to MySQL

If you encounter problems when you try to connect to the MySQL server, the following items describe some courses of action you can take to correct the problem.

- Make sure that the server is running. If it is not, clients cannot connect to it. For example, if an attempt to connect to the server fails with a message such as one of those following, one cause might be that the server is not running:

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

- It might be that the server is running, but you are trying to connect using a TCP/IP port, named pipe, or Unix socket file different from the one on which the server is listening. To correct this when you invoke a client program, specify a `--port` option to indicate the proper port number, or a `--socket` option to indicate the proper named pipe or Unix socket file. To find out where the socket file is, you can use this command:

```
shell> netstat -ln | grep mysql
```

- Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with `--skip-networking`, it will not accept TCP/IP connections at all. If the server was started with the `bind_address` system variable set to `127.0.0.1`, it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.

- Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or Windows Firewall may need to be configured not to block the MySQL port.

- The grant tables must be properly set up so that the server can use them for access control. For some distribution types (such as binary distributions on Windows, or RPM distributions on Linux), the installation process initializes the MySQL data directory, including the `mysql` system database containing the grant tables. For distributions that do not do this, you must initialize the data directory manually. For details, see Section 2.10, “Postinstallation Setup and Testing”.

  To determine whether you need to initialize the grant tables, look for a `mysql` directory under the data directory. (The data directory normally is named `data` or `var` and is located under your MySQL installation directory.) Make sure that you have a file named `user.MYD` in the `mysql` database directory. If not, initialize the data directory. After doing so and starting the server, you should be able to connect to the server.

- After a fresh installation, if you try to log on to the server as `root` without using a password, you might get the following error message.

  ```
  shell> mysql -u root
  ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: NO)
  ```

  It means a root password has already been assigned during installation and it has to be supplied. See Section 2.10.4, “Securing the Initial MySQL Account” on the different ways the password could have been assigned and, in some cases, how to find it. If you need to reset the root password, see instructions in Section B.4.3.2, “How to Reset the Root Password”. After you have found or reset your password, log on again as `root` using the `--password` (or `-p`) option:

  ```
  shell> mysql -u root -p
  Enter password:
  ```

  However, the server is going to let you connect as `root` without using a password if you have initialized MySQL using `mysqld --initialize-insecure` (see Section 2.10.1, “Initializing the Data Directory” for details). That is a security risk, so you should set a password for the `root` account; see Section 2.10.4, “Securing the Initial MySQL Account” for instructions.

- If you have updated an existing MySQL installation to a newer version, did you perform the MySQL upgrade procedure? If not, do so. The structure of the grant tables changes occasionally when new capabilities are added, so after an upgrade you should always make sure that your tables have the current structure. For instructions, see Section 2.11, “Upgrading MySQL”.

- If a client program receives the following error message when it tries to connect, it means that the server expects passwords in a newer format than the client is capable of generating:

  ```
  shell> mysql
  Client does not support authentication protocol requested
  by server; consider upgrading MySQL client
  ```

- Remember that client programs use connection parameters specified in option files or environment variables. If a client program seems to be sending incorrect default connection parameters when you have not specified them on the command line, check any applicable option files and your environment. For example, if you get `Access denied` when you run a client without any options, make sure that you have not specified an old password in any of your option files!

You can suppress the use of option files by a client program by invoking it with the `--no-defaults` option. For example:

```
shell> mysqladmin --no-defaults -u root version
```

The option files that clients use are listed in Section 4.2.2.2, “Using Option Files”. Environment variables are listed in Section 4.9, “MySQL Environment Variables”.

- If you get the following error, it means that you are using an incorrect `root` password:

```
shell> mysqladmin -u root -pxxxx ver
Access denied for user 'root'@'localhost' (using password: YES)
```

If the preceding error occurs even when you have not specified a password, it means that you have an incorrect password listed in some option file. Try the `--no-defaults` option as described in the previous item.

For information on changing passwords, see Section 6.2.14, “Assigning Account Passwords”.

If you have lost or forgotten the `root` password, see Section B.4.3.2, “How to Reset the Root Password”.

- `localhost` is a synonym for your local host name, and is also the default host to which clients try to connect if you specify no host explicitly.

  You can use a `--host=127.0.0.1` option to name the server host explicitly. This will make a TCP/IP connection to the local `mysqld` server. You can also use TCP/IP by specifying a `--host` option that uses the actual host name of the local host. In this case, the host name must be specified in a `user` table row on the server host, even though you are running the client program on the same host as the server.

- The `Access denied` error message tells you who you are trying to log in as, the client host from which you are trying to connect, and whether you were using a password. Normally, you should have one row in the `user` table that exactly matches the host name and user name that were given in the error message. For example, if you get an error message that contains `using password: NO`, it means that you tried to log in without a password.

- If you get an `Access denied` error when trying to connect to the database with `mysql -u` *`user_name`*, you may have a problem with the `user` table. Check this by executing `mysql -u root mysql` and issuing this SQL statement:

```
SELECT * FROM user;
```

The result should include a row with the `Host` and `User` columns matching your client's host name and your MySQL user name.

- If the following error occurs when you try to connect from a host other than the one on which the MySQL server is running, it means that there is no row in the `user` table with a `Host` value that matches the client host:

```
Host ... is not allowed to connect to this MySQL server
```

You can fix this by setting up an account for the combination of client host name and user name that you are using when trying to connect.

If you do not know the IP address or host name of the machine from which you are connecting, you should put a row with `'%'` as the `Host` column value in the `user` table. After trying to connect from the client machine, use a `SELECT USER()` query to see how you really did connect. Then change the `'%'` in the `user` table row to the actual host name that shows up in the log. Otherwise, your system is left insecure because it permits connections from any host for the given user name.

On Linux, another reason that this error might occur is that you are using a binary MySQL version that is compiled with a different version of the `glibc` library than the one you are using. In this case, you should either upgrade your operating system or `glibc`, or download a source distribution of MySQL version and compile it yourself. A source RPM is normally trivial to compile and install, so this is not a big problem.

- If you specify a host name when trying to connect, but get an error message where the host name is not shown or is an IP address, it means that the MySQL server got an error when trying to resolve the IP address of the client host to a name:

```
shell> mysqladmin -u root -pxxxx -h some_hostname ver
Access denied for user 'root'@'' (using password: YES)
```

If you try to connect as `root` and get the following error, it means that you do not have a row in the `user` table with a `User` column value of `'root'` and that `mysqld` cannot resolve the host name for your client:

```
Access denied for user ''@'unknown'
```

These errors indicate a DNS problem. To fix it, execute `mysqladmin flush-hosts` to reset the internal DNS host cache. See Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

Some permanent solutions are:

- Determine what is wrong with your DNS server and fix it.
- Specify IP addresses rather than host names in the MySQL grant tables.
- Put an entry for the client machine name in `/etc/hosts` on Unix or `\windows\hosts` on Windows.
- Start `mysqld` with the `--skip-name-resolve` option.
- Start `mysqld` with the `--skip-host-cache` option.
- On Unix, if you are running the server and the client on the same machine, connect to `localhost`. For connections to `localhost`, MySQL programs attempt to connect to the local server by using a Unix socket file, unless there are connection parameters specified to ensure that the client makes a TCP/IP connection. For more information, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.
- On Windows, if you are running the server and the client on the same machine and the server supports named pipe connections, connect to the host name `.` (period). Connections to `.` use a named pipe rather than TCP/IP.

- If `mysql -u root` works but `mysql -h your_hostname -u root` results in `Access denied` (where *`your_hostname`* is the actual host name of the local host), you may not have the correct name for your host in the `user` table. A common problem here is that the `Host` value in the `user` table row specifies an unqualified host name, but your system's name resolution routines return a fully qualified domain name (or vice versa). For example, if you have a row with host `'pluto'` in the

`user` table, but your DNS tells MySQL that your host name is `'pluto.example.com'`, the row does not work. Try adding a row to the `user` table that contains the IP address of your host as the `Host` column value. (Alternatively, you could add a row to the `user` table with a `Host` value that contains a wildcard (for example, `'pluto.%'`). However, use of `Host` values ending with `%` is *insecure* and is *not* recommended!)

- If `mysql -u user_name` works but `mysql -u user_name some_db` does not, you have not granted access to the given user for the database named `some_db`.
- If `mysql -u user_name` works when executed on the server host, but `mysql -h host_name -u user_name` does not work when executed on a remote client host, you have not enabled access to the server for the given user name from the remote host.
- If you cannot figure out why you get `Access denied`, remove from the `user` table all rows that have `Host` values containing wildcards (rows that contain `'%'` or `'_'` characters). A very common error is to insert a new row with `Host='%'` and `User='some_user'`, thinking that this enables you to specify `localhost` to connect from the same machine. The reason that this does not work is that the default privileges include a row with `Host='localhost'` and `User=''`. Because that row has a `Host` value `'localhost'` that is more specific than `'%'`, it is used in preference to the new row when connecting from `localhost`! The correct procedure is to insert a second row with `Host='localhost'` and `User='some_user'`, or to delete the row with `Host='localhost'` and `User=''`. After deleting the row, remember to issue a `FLUSH PRIVILEGES` statement to reload the grant tables. See also Section 6.2.6, “Access Control, Stage 1: Connection Verification”.
- If you are able to connect to the MySQL server, but get an `Access denied` message whenever you issue a `SELECT ... INTO OUTFILE` or `LOAD DATA` statement, your row in the `user` table does not have the `FILE` privilege enabled.
- If you change the grant tables directly (for example, by using `INSERT`, `UPDATE`, or `DELETE` statements) and your changes seem to be ignored, remember that you must execute a `FLUSH PRIVILEGES` statement or a `mysqladmin flush-privileges` command to cause the server to reload the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you change the `root` password with an `UPDATE` statement, you will not need to specify the new password until after you flush the privileges, because the server will not know you've changed the password yet!
- If your privileges seem to have changed in the middle of a session, it may be that a MySQL administrator has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in Section 6.2.13, “When Privilege Changes Take Effect”.
- If you have access problems with a Perl, PHP, Python, or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -ppassword db_name`. If you are able to connect using the `mysql` client, the problem lies with your program, not with the access privileges. (There is no space between `-p` and the password; you can also use the `--password=password` syntax to specify the password. If you use the `-p` or `--password` option with no password value, MySQL prompts you for the password.)
- For testing purposes, start the `mysqld` server with the `--skip-grant-tables` option. Then you can change the MySQL grant tables and use the `SHOW GRANTS` statement to check whether your modifications have the desired effect. When you are satisfied with your changes, execute `mysqladmin flush-privileges` to tell the `mysqld` server to reload the privileges. This enables you to begin using the new grant table contents without stopping and restarting the server.
- If everything else fails, start the `mysqld` server with a debugging option (for example, `--debug=d,general,query`). This prints host and user information about attempted connections, as well as information about each command issued. See Section 29.5.4, “The DBUG Package”.

- If you have any other problems with the MySQL grant tables and ask on the MySQL Community Slack, always provide a dump of the MySQL grant tables. You can dump the tables with the `mysqldump mysql` command. To file a bug report, see the instructions at Section 1.7, “How to Report Bugs or Problems”. In some cases, you may need to restart `mysqld` with `--skip-grant-tables` to run `mysqldump`.

6.2.22 SQL-Based Account Activity Auditing

Applications can use the following guidelines to perform SQL-based auditing that ties database activity to MySQL accounts.

MySQL accounts correspond to rows in the `mysql.user` system table. When a client connects successfully, the server authenticates the client to a particular row in this table. The `User` and `Host` column values in this row uniquely identify the account and correspond to the `'user_name'@'host_name'` format in which account names are written in SQL statements.

The account used to authenticate a client determines which privileges the client has. Normally, the `CURRENT_USER()` function can be invoked to determine which account this is for the client user. Its value is constructed from the `User` and `Host` columns of the `user` table row for the account.

However, there are circumstances under which the `CURRENT_USER()` value corresponds not to the client user but to a different account. This occurs in contexts when privilege checking is not based the client's account:

- Stored routines (procedures and functions) defined with the `SQL SECURITY DEFINER` characteristic
- Views defined with the `SQL SECURITY DEFINER` characteristic
- Triggers and events

In those contexts, privilege checking is done against the `DEFINER` account and `CURRENT_USER()` refers to that account, not to the account for the client who invoked the stored routine or view or who caused the trigger to activate. To determine the invoking user, you can call the `USER()` function, which returns a value indicating the actual user name provided by the client and the host from which the client connected. However, this value does not necessarily correspond directly to an account in the `user` table, because the `USER()` value never contains wildcards, whereas account values (as returned by `CURRENT_USER()`) may contain user name and host name wildcards.

For example, a blank user name matches any user, so an account of `''@'localhost'` enables clients to connect as an anonymous user from the local host with any user name. In this case, if a client connects as `user1` from the local host, `USER()` and `CURRENT_USER()` return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+-----------------+----------------+
| USER()          | CURRENT_USER() |
+-----------------+----------------+
| user1@localhost | @localhost     |
+-----------------+----------------+
```

The host name part of an account can contain wildcards, too. If the host name contains a `'%'` or `'_'` pattern character or uses netmask notation, the account can be used for clients connecting from multiple hosts and the `CURRENT_USER()` value will not indicate which one. For example, the account `'user2'@'%.example.com'` can be used by `user2` to connect from any host in the `example.com` domain. If `user2` connects from `remote.example.com`, `USER()` and `CURRENT_USER()` return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+-------------------------+--------------------+
| USER()                  | CURRENT_USER()     |
+-------------------------+--------------------+
| user2@remote.example.com | user2@%.example.com |
+-------------------------+--------------------+
```

If an application must invoke `USER()` for user auditing (for example, if it does auditing from within triggers) but must also be able to associate the `USER()` value with an account in the `user` table, it is necessary to avoid accounts that contain wildcards in the `User` or `Host` column. Specifically, do not permit `User` to be empty (which creates an anonymous-user account), and do not permit pattern characters or netmask notation in `Host` values. All accounts must have a nonempty `User` value and literal `Host` value.

With respect to the previous examples, the `''@'localhost'` and `'user2'@'%.example.com'` accounts should be changed not to use wildcards:

```
RENAME USER ''@'localhost' TO 'user1'@'localhost';
RENAME USER 'user2'@'%.example.com' TO 'user2'@'remote.example.com';
```

If `user2` must be able to connect from several hosts in the `example.com` domain, there should be a separate account for each host.

To extract the user name or host name part from a `CURRENT_USER()` or `USER()` value, use the `SUBSTRING_INDEX()` function:

```
mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(),'@',1);
+---------------------------------------+
| SUBSTRING_INDEX(CURRENT_USER(),'@',1) |
+---------------------------------------+
| user1                                 |
+---------------------------------------+

mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(),'@',-1);
+----------------------------------------+
| SUBSTRING_INDEX(CURRENT_USER(),'@',-1) |
+----------------------------------------+
| localhost                              |
+----------------------------------------+
```

6.3 Using Encrypted Connections

With an unencrypted connection between the MySQL client and the server, someone with access to the network could watch all your traffic and inspect the data being sent or received between client and server.

When you must move information over a network in a secure fashion, an unencrypted connection is unacceptable. To make any kind of data unreadable, use encryption. Encryption algorithms must include security elements to resist many kinds of known attacks such as changing the order of encrypted messages or replaying data twice.

MySQL supports encrypted connections between clients and the server using the TLS (Transport Layer Security) protocol. TLS is sometimes referred to as SSL (Secure Sockets Layer) but MySQL does not actually use the SSL protocol for encrypted connections because its encryption is weak (see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”).

TLS uses encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect data change, loss, or replay. TLS also incorporates algorithms that provide identity verification using the X.509 standard.

X.509 makes it possible to identify someone on the Internet. In basic terms, there should be some entity called a “Certificate Authority” (or CA) that assigns electronic certificates to anyone who needs them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key and a secret key). A certificate owner can present the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted using this public key can be decrypted only using the corresponding secret key, which is held by the owner of the certificate.

Support for encrypted connections in MySQL is provided using OpenSSL. For information about the encryption protocols and ciphers that OpenSSL supports, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

Note

From MySQL 8.0.11 to 8.0.17, it was possible to compile MySQL using wolfSSL as an alternative to OpenSSL. As of MySQL 8.0.18, support for wolfSSL is removed and all MySQL builds use OpenSSL.

By default, MySQL programs attempt to connect using encryption if the server supports encrypted connections, falling back to an unencrypted connection if an encrypted connection cannot be established. For information about options that affect use of encrypted connections, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections” and Command Options for Encrypted Connections.

MySQL performs encryption on a per-connection basis, and use of encryption for a given user can be optional or mandatory. This enables you to choose an encrypted or unencrypted connection according to the requirements of individual applications. For information on how to require users to use encrypted connections, see the discussion of the `REQUIRE` clause of the `CREATE USER` statement in Section 13.7.1.3, “CREATE USER Syntax”. See also the description of the `require_secure_transport` system variable at Section 5.1.8, “Server System Variables”

Encrypted connections can be used between master and slave replication servers. See Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”.

For information about using encrypted connections from the MySQL C API, see Section 28.7.22, “C API Encrypted Connection Support”.

It is also possible to connect using encryption from within an SSH connection to the MySQL server host. For an example, see Section 6.3.4, “Connecting to MySQL Remotely from Windows with SSH”.

6.3.1 Configuring MySQL to Use Encrypted Connections

Several options are available to indicate whether to use encrypted connections, and to specify the appropriate certificate and key files. This section provides general guidance about configuring the server and clients for encrypted connections:

- Server-Side Startup Configuration for Encrypted Connections
- Server-Side Runtime Configuration for Encrypted Connections
- Client-Side Configuration for Encrypted Connections
- Configuring Encrypted Connections as Mandatory

For a complete list of options related to establishment of encrypted connections, see Command Options for Encrypted Connections. Instructions for creating any required certificate and key files are available in Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.

Encrypted connections also can be used in these contexts:

- Between master and slave replication servers. See Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”.
- Among Group Replication servers. See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”.
- By client programs that are based on the MySQL C API. See Section 28.7.22, “C API Encrypted Connection Support”.

Server-Side Startup Configuration for Encrypted Connections

On the server side, the `--ssl` option specifies that the server permits but does not require encrypted connections. This option is enabled by default, so it need not be specified explicitly.

To require that clients connect using encrypted connections, enable the `require_secure_transport` system variable. See Configuring Encrypted Connections as Mandatory.

These options on the server side specify the certificate and key files the server uses when permitting clients to establish encrypted connections:

- `--ssl-ca`: The path name of the Certificate Authority (CA) certificate file. (`--ssl-capath` is similar but specifies the path name of a directory of CA certificate files.)
- `--ssl-cert`: The path name of the server public key certificate file. This certificate can be sent to the client and authenticated against the CA certificate that it has.
- `--ssl-key`: The path name of the server private key file.

For example, to enable the server for encrypted connections, start it with these lines in the `my.cnf` file, changing the file names as necessary:

```
[mysqld]
ssl-ca=ca.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

To specify in addition that clients are required to use encrypted connections, enable the `require_secure_transport` system variable:

```
[mysqld]
ssl-ca=ca.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
require_secure_transport=ON
```

Each certificate and key option names a file in PEM format. Should you need to create the required certificate and key files, see Section 6.3.3, “Creating SSL and RSA Certificates and Keys”. MySQL servers compiled using OpenSSL can generate missing certificate and key files automatically at startup. See Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”. Alternatively, if you have a MySQL source distribution, you can test your setup using the demonstration certificate and key files in its `mysql-test/std_data` directory.

The server performs certificate and key file autodiscovery. If `--ssl` is enabled (possibly along with `--ssl-cipher`) and other `--ssl-xxx` options are *not* given to configure encrypted connections explicitly, the server attempts to enable encrypted connection support automatically at startup:

- If the server discovers valid certificate and key files named `ca.pem`, `server-cert.pem`, and `server-key.pem` in the data directory, it enables support for encrypted connections by clients. (The files need not have been generated automatically; what matters is that they have those names and are valid.)
- If the server does not find valid certificate and key files in the data directory, it continues executing but without support for encrypted connections.

If the server automatically enables encrypted connection support, it writes a note to the error log. If the server discovers that the CA certificate is self-signed, it writes a warning to the error log. (The certificate is self-signed if created automatically by the server or manually using `mysql_ssl_rsa_setup`.)

MySQL also provides these options for server-side SSL control:

- `--ssl-cipher`: The list of permissible ciphers for connection encryption.
- `--ssl-crl`: The path name of the file containing certificate revocation lists. (`--ssl-crlpath` is similar but specifies the path name of a directory of certificate revocation-list files.)

The values of the `--ssl-xxx` options set the values of the corresponding system variables (`ssl_ca`, `ssl_cert`, `ssl_key`, and so forth).

To explicitly specify which encryption protocols and ciphersuites the server permits for encrypted connections, use the `tls_version` and `tls_ciphersuites` system variables; see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”. For example, you can set `tls_version` to prevent clients from using less-secure protocols.

Server-Side Runtime Configuration for Encrypted Connections

Certain encryption-related system variables can be set at runtime: `require_secure_transport`, and (as of MySQL 8.0.16) `tls_version` and `tls_ciphersuites`. If changed with `SET GLOBAL`, the new values apply to connections established subsequent to the change, and apply only until server restart. If changed with `SET PERSIST`, the new values also apply to subsequent server restarts. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

In addition, as of MySQL 8.0.16, the SSL context the server uses for new connections is reconfigurable at runtime. This capability may be useful, for example, to avoid restarting a MySQL server that has been running so long that its SSL certificate has expired.

The server creates the initial SSL context from the values that the context-related system variables have at startup. It also initializes a set of context-related status variables to indicate the values used in the context. The following table shows the system variables that define the SSL context and the corresponding status variables that indicate the currently active context values.

Table 6.11 Corresponding System and Status Variables Related to Server SSL Context

System Variable Name	Status Variable Name
`ssl_ca`	`Current_tls_ca`
`ssl_capath`	`Current_tls_capath`
`ssl_cert`	`Current_tls_cert`
`ssl_cipher`	`Current_tls_cipher`
`ssl_crl`	`Current_tls_crl`
`ssl_crlpath`	`Current_tls_crlpath`
`ssl_key`	`Current_tls_key`
`tls_ciphersuites`	`Current_tls_ciphersuites`

System Variable Name	Status Variable Name
tls_version	Current_tls_version

To reconfigure the SSL context at runtime, use this procedure:

1. Set any SSL context-related system variables that should be changed to their new values.
2. Execute ALTER INSTANCE RELOAD TLS. This statement reconfigures the active SSL context from the current values of the SSL context-related system variables. It also sets the context-related status variables to reflect the new active context values. The statement requires the CONNECTION_ADMIN privilege.
3. New connections established after execution of ALTER INSTANCE RELOAD TLS use the new SSL context. Existing connections remain unaffected. If existing connections should be terminated, use the KILL statement.

The members of each pair of system and status variables may have different values temporarily due to the way the reconfiguration procedure works:

- Changes to the system variables prior to ALTER INSTANCE RELOAD TLS do not change the SSL context. At this point, those changes have no effect on new connections, and corresponding context-related system and status variables may have different values. This enables you to make any changes required to the system variables, then update the active SSL context atomically with ALTER INSTANCE RELOAD TLS after all system variable changes have been made.
- After ALTER INSTANCE RELOAD TLS, corresponding system and status variables have the same values. This remains true until the next change to the system variables.

In some cases, ALTER INSTANCE RELOAD TLS by itself may suffice to reconfigure the SSL context, without changing any system variables. Suppose that the certificate in the file named by ssl_cert has expired. It is sufficient to replace the existing file contents with a nonexpired certificate and execute ALTER INSTANCE RELOAD TLS to cause the new file contents to be read and used for new connections.

By default, the RELOAD TLS action rolls back with an error and has no effect if the configuration values do not permit creation of a new SSL context. The previous context values continue to be used for new connections.

If the optional NO ROLLBACK ON ERROR clause is given and a new context cannot be created, rollback does not occur. Instead, a warning is generated and SSL is disabled for new connections.

The server-side --ssl option has an effect only at server startup on whether the server accepts SSL connections. It is ignored by, and has no effect on the operation of, ALTER INSTANCE RELOAD TLS. For example, you can use --ssl=0 to start the server with SSL connections disabled, then reconfigure SSL and execute ALTER INSTANCE RELOAD TLS to enable SSL connections at runtime.

ALTER INSTANCE RELOAD TLS changes only the SSL context the server itself uses for new connections. It does not affect the SSL context used by other enabled server plugins or components such as X Plugin or Group Replication. If you want to apply the reconfiguration to Group Replication's group communication connections, which take their settings from the server's SSL context-related system variables, you must issue STOP GROUP_REPLICATION followed by START GROUP_REPLICATION to stop and restart Group Replication.

Note

Prior to MySQL 8.0.16, the SSL context-related system variables are not dynamic. They can be set at server startup, but cannot be changed thereafter. These system

variables therefore determine the SSL context values the server uses for all new connections.

Client-Side Configuration for Encrypted Connections

By default, MySQL client programs attempt to establish an encrypted connection if the server supports encrypted connections, with further control available through the `--ssl-mode` option:

- In the absence of an `--ssl-mode` option, clients attempt to connect using encryption, falling back to an unencrypted connection if an encrypted connection cannot be established. This is also the behavior with an explicit `--ssl-mode=PREFFERED` option.
- With `--ssl-mode=REQUIRED`, clients require an encrypted connection and fail if one cannot be established.
- With `--ssl-mode=DISABLED`, clients use an unencrypted connection.
- With `--ssl-mode=VERIFY_CA` or `--ssl-mode=VERIFY_IDENTITY`, clients require an encrypted connection, and also perform verification against the server CA certificate and (with `VERIFY_IDENTITY`) against the server host name in its certificate.

Attempts to establish an unencrypted connection fail if the `require_secure_transport` system variable is enabled on the server side to cause the server to require encrypted connections. See Configuring Encrypted Connections as Mandatory.

The following options on the client side identify the certificate and key files clients use when establishing encrypted connections to the server. They are similar to the options used on the server side, but `--ssl-cert` and `--ssl-key` identify the client public and private key:

- `--ssl-ca`: The path name of the Certificate Authority (CA) certificate file. This option, if used, must specify the same certificate used by the server. (`--ssl-capath` is similar but specifies the path name of a directory of CA certificate files.)
- `--ssl-cert`: The path name of the client public key certificate file.
- `--ssl-key`: The path name of the client private key file.

For additional security relative to that provided by the default encryption, clients can supply a CA certificate matching the one used by the server and enable host name identity verification. In this way, the server and client place their trust in the same CA certificate and the client verifies that the host to which it connected is the one intended:

- To specify the CA certificate, use `--ssl-ca` (or `--ssl-capath`), and specify `--ssl-mode=VERIFY_CA`.
- To enable host name identity verification as well, use `--ssl-mode=VERIFY_IDENTITY` rather than `--ssl-mode=VERIFY_CA`.

Note

Host name identity verification with `VERIFY_IDENTITY` does not work with self-signed certificates that are created automatically by the server or manually using `mysql_ssl_rsa_setup` (see Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”). Such self-signed certificates do not contain the server name as the Common Name value.

Host name identity verification also does not work with certificates that specify the Common Name using wildcards because that name is compared verbatim to the server name.

MySQL also provides these options for client-side SSL control:

- `--ssl-cipher`: The list of permissible ciphers for connection encryption.
- `--ssl-crl`: The path name of the file containing certificate revocation lists. (`--ssl-crlpath` is similar but specifies the path name of a directory of certificate revocation-list files.)
- `--tls-version`, `--tls-ciphersuites`: The permitted encryption protocols and ciphersuites; see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

Depending on the encryption requirements of the MySQL account used by a client, the client may be required to specify certain options to connect using encryption to the MySQL server.

Suppose that you want to connect using an account that has no special encryption requirements or that was created using a `CREATE USER` statement that included the `REQUIRE SSL` clause. Assuming that the server supports encrypted connections, a client can connect using encryption with no `--ssl-mode` option or with an explicit `--ssl-mode=PREFFERED` option:

```
mysql
```

Or:

```
mysql --ssl-mode=PREFERRED
```

For an account created with a `REQUIRE SSL` clause, the connection attempt fails if an encrypted connection cannot be established. For an account with no special encryption requirements, the attempt falls back to an unencrypted connection if an encrypted connection cannot be established. To prevent fallback and fail if an encrypted connection cannot be obtained, connect like this:

```
mysql --ssl-mode=REQUIRED
```

If the account has more stringent security requirements, other options must be specified to establish an encrypted connection:

- For accounts created with a `REQUIRE X509` clause, clients must specify at least `--ssl-cert` and `--ssl-key`. In addition, `--ssl-ca` (or `--ssl-capath`) is recommended so that the public certificate provided by the server can be verified. For example:

  ```
  mysql --ssl-ca=ca.pem \
        --ssl-cert=client-cert.pem \
        --ssl-key=client-key.pem
  ```

- For accounts created with a `REQUIRE ISSUER` or `REQUIRE SUBJECT` clause, the encryption requirements are the same as for `REQUIRE X509`, but the certificate must match the issue or subject, respectively, specified in the account definition.

For additional information about the `REQUIRE` clause, see Section 13.7.1.3, “CREATE USER Syntax”.

To prevent use of encryption and override other `--ssl-xxx` options, invoke the client program with `--ssl-mode=DISABLED`:

```
mysql --ssl-mode=DISABLED
```

To determine whether the current connection with the server uses encryption, check the session value of the `Ssl_cipher` status variable. If the value is empty, the connection is not encrypted. Otherwise, the connection is encrypted and the value indicates the encryption cipher. For example:

```
mysql> SHOW SESSION STATUS LIKE 'Ssl_cipher';
+---------------+-------------------------+
| Variable_name | Value                   |
+---------------+-------------------------+
| Ssl_cipher    | DHE-RSA-AES128-GCM-SHA256 |
+---------------+-------------------------+
```

For the `mysql` client, an alternative is to use the `STATUS` or `\s` command and check the `SSL` line:

```
mysql> \s
...
SSL: Not in use
...
```

Or:

```
mysql> \s
...
SSL: Cipher in use is DHE-RSA-AES128-GCM-SHA256
...
```

Configuring Encrypted Connections as Mandatory

For some MySQL deployments it may be not only desirable but mandatory to use encrypted connections (for example, to satisfy regulatory requirements). This section discusses configuration settings that enable you to do this. These levels of control are available:

- You can configure the server to require the clients connect using encrypted connections.
- You can invoke individual client programs to require an encrypted connection, even if the server permits but does not require encryption.
- You can configure individual MySQL accounts to be usable only over encrypted connections.

To require that clients connect using encrypted connections, enable the `require_secure_transport` system variable. For example, put these lines in the server `my.cnf` file:

```
[mysqld]
require_secure_transport=ON
```

Alternatively, to set and persist the value at runtime, use this statement:

```
SET PERSIST require_secure_transport=ON;
```

`SET PERSIST` sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

With `require_secure_transport` enabled, client connections to the server are required to use some form of secure transport, and the server permits only TCP/IP connections that use SSL, or connections that use a socket file (on Unix) or shared memory (on Windows). The server rejects nonsecure connection attempts, which fail with an `ER_SECURE_TRANSPORT_REQUIRED` error.

To invoke a client program such that it requires an encrypted connection whether or not the server requires encryption, use an `--ssl-mode` option value of `REQUIRED`, `VERIFY_CA`, or `VERIFY_IDENTITY`. For example:

```
mysql --ssl-mode=REQUIRED
mysqldump --ssl-mode=VERIFY_CA
mysqladmin --ssl-mode=VERIFY_IDENTITY
```

To configure a MySQL account to be usable only over encrypted connections, include a `REQUIRE` clause in the `CREATE USER` statement that creates the account, specifying in that clause the encryption characteristics you require. For example, to require an encrypted connection and the use of a valid X.509 certificate, use `REQUIRE X509`:

```
CREATE USER 'jeffrey'@'localhost' REQUIRE X509;
```

For additional information about the `REQUIRE` clause, see Section 13.7.1.3, “CREATE USER Syntax”.

To modify existing accounts that have no encryption requirements, use the `ALTER USER` statement.

6.3.2 Encrypted Connection TLS Protocols and Ciphers

MySQL supports multiple TLS protocols and ciphers, and enables configuring which protocols and ciphers to permit for encrypted connections. It is also possible to determine which protocol and cipher the current session uses.

- Supported Connection TLS Protocols
- Connection TLS Protocol Configuration
- Connection Cipher Configuration
- Connection TLS Protocol Negotiation
- Monitoring Current Session TLS Protocol and Cipher

Supported Connection TLS Protocols

MySQL supports encrypted connections using the TLSv1, TLSv1.1, TLSv1.2, and TLSv1.3 protocols, listed in order from less secure to more secure. The set of protocols actually permitted for connections is subject to multiple factors:

- MySQL configuration. Permitted TLS protocols can be configured on both the server side and client side to include only a subset of the supported TLS protocols. The configuration on both sides must include at least one protocol in common or connection attempts cannot negotiate a protocol to use. For details, see Connection TLS Protocol Negotiation.
- System-wide host configuration. The host system may permit only certain TLS protocols, which means that MySQL connections cannot use nonpermitted protocols even if MySQL itself permits them:
  - Suppose that MySQL configuration permits TLSv1, TLSv1.1, and TLSv1.2, but your host system configuration permits only connections that use TLSv1.2 or higher. In this case, you cannot establish MySQL connections that use TLSv1 or TLSv1.1, even though MySQL is configured to permit them, because the host system does not permit them.
  - If MySQL configuration permits TLSv1, TLSv1.1, and TLSv1.2, but your host system configuration permits only connections that use TLSv1.3 or higher, you cannot establish MySQL connections at all, because no protocol permitted by MySQL is permitted by the host system.

  Workarounds for this issue include:

  - Change the system-wide host configuration to permit additional TLS protocols. Consult your operating system documentation for instructions. For example, your system may have an `/etc/ssl/openssl.cnf` file that contains these lines to restrict TLS protocols to TLSv1.2 or higher:

    ```
    [system_default_sect]
    MinProtocol = TLSv1.2
    ```

    Changing the value to a lower protocol version or `None` makes the system more permissive. However, this workaround has the disadvantage that permitting lower (less secure) protocols may have adverse security consequences.

  - If you cannot or prefer not to change the host system TLS configuration, change MySQL applications to use higher (more secure) TLS protocols that are permitted by the host system. This may not be possible for older versions of MySQL that support only lower protocol versions. For example, TLSv1 is the only supported protocol prior to MySQL 5.6.46, so attempts to connect to a pre-5.6.46 server fail even if the client is from a newer MySQL version that supports higher protocol versions. In such cases, an upgrade to a version of MySQL that supports additional TLS versions may be required.

- The SSL library. If the SSL library does not support a particular protocol, neither does MySQL, and any parts of the following discussion that specify that protocol do not apply.

Note

Support for the TLSv1.3 protocol is available as of MySQL 8.0.16 (with the exception of the Group Replication component, where support is available as of MySQL 8.0.18). To use the TLSv1.3 protocol, both the MySQL server and the client application must be compiled using OpenSSL 1.1.1 or higher.

Connection TLS Protocol Configuration

On the server side, the value of the `tls_version` system variable determines which TLS protocols a MySQL server permits for encrypted connections. The `tls_version` value applies to connections from clients and from slave servers using regular master/slave replication. The variable value is a list of one or more comma-separated protocol versions from this list (not case-sensitive): TLSv1, TLSv1.1, TLSv1.2, and (if available) TLSV1.3. By default, this variable lists all protocols supported by the SSL library used to compile MySQL. To determine the value of `tls_version` at runtime, use this statement:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'tls_version';
+---------------+-----------------------+
| Variable_name | Value                 |
+---------------+-----------------------+
| tls_version   | TLSv1,TLSv1.1,TLSv1.2 |
+---------------+-----------------------+
```

To change the value of `tls_version`, set it at server startup. For example, to permit connections that use the TLSv1.1 or TLSv1.2 protocol, but prohibit connections that use the less-secure TLSv1 protocol, use these lines in the server `my.cnf` file:

```
[mysqld]
tls_version=TLSv1.1,TLSv1.2
```

To be even more restrictive and permit only TLSv1.2 connections, set `tls_version` like this:

```
[mysqld]
```

```
tls_version=TLSv1.2
```

As of MySQL 8.0.16, `tls_version` can also be changed at runtime. See Server-Side Runtime Configuration for Encrypted Connections.

On the client side, the `--tls-version` option specifies which TLS protocols a client program permits for connections to the server. The format of the option value is the same as for the `tls_version` system variable described previously (a list of one or more comma-separated protocol versions).

For master/slave replication, the `MASTER_TLS_VERSION` option for the `CHANGE MASTER TO` statement specifies which TLS protocols a slave server permits for connections to the master. The format of the option value is the same as for the `tls_version` system variable described previously. See Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”.

The protocols that can be specified for `MASTER_TLS_VERSION` depend on the SSL library. This option is independent of and not affected by the server `tls_version` value. For example, a server that acts as a replication slave can be configured with `tls_version` set to TLSv1.3 to permit only incoming connections that use TLSv1.3, but also configured with `MASTER_TLS_VERSION` set to TLSv1.2 to permit only TLSv1.2 for outgoing slave connections to the master.

TLS protocol configuration affects which protocol a given connection uses, as described in Connection TLS Protocol Negotiation.

Permitted protocols should be chosen such as not to leave “holes” in the list. For example, these server configuration values do not have holes:

```
tls_version=TLSv1,TLSv1.1,TLSv1.2,TLSv1.3
tls_version=TLSv1.1,TLSv1.2,TLSv1.3
tls_version=TLSv1.2,TLSv1.3
tls_version=TLSv1.3
```

These values do have holes and should not be used:

```
tls_version=TLSv1,TLSv1.2
tls_version=TLSv1.1,TLSv1.3
```

The prohibition on holes also applies in other configuration contexts, such as for clients or replication slaves.

Connection Cipher Configuration

A default set of ciphers applies to encrypted connections, which can be overridden by explicitly configuring the permitted ciphers. During connection establishment, both sides of a connection must permit some cipher in common or the connection fails. Of the permitted ciphers common to both sides, the SSL library chooses the one supported by the provided certificate that has the highest priority.

To specify a cipher or ciphers applicable for encrypted connections that use TLS protocols up through TLSv1.2:

- The `--ssl-cipher` option is available for the server and for client programs.
- For master/slave replication, the `MASTER_SSL_CIPHER` option for the `CHANGE MASTER TO` statement specifies which ciphers a slave server permits for connections to the master. See Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”.

For encrypted connections that use TLSv1.3, OpenSSL 1.1.1 and higher supports the following ciphersuites, the first three of which are enabled by default:

```
TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_CCM_SHA256
TLS_AES_128_CCM_8_SHA256
```

To configure the permitted TLSv1.3 ciphersuites explicitly:

- On the server side, use the `tls_ciphersuites` system variable.
- On the client side, use the `--tls-ciphersuites` option.

In each case, the configuration value is a list of one or more colon-separated ciphersuite names.

Note

1. Ciphersuite support is available as of MySQL 8.0.16, but requires that both the MySQL server and the client application be compiled using OpenSSL 1.1.1 or higher.
2. If TLSv1.3 is used for replication connections to a replication slave, or in Group Replication for distributed recovery, the replication master or Group Replication group members must permit the use of at least one TLSv1.3 ciphersuite that is enabled by default.

A given cipher may work only with particular TLS protocols, which affects the TLS protocol negotiation process. See Connection TLS Protocol Negotiation.

To determine which ciphers a given server supports, check the session value of the `Ssl_cipher_list` status variable:

```
SHOW SESSION STATUS LIKE 'Ssl_cipher_list';
```

The `Ssl_cipher_list` status variable lists the possible SSL ciphers (empty for non-SSL connections). If MySQL supports TLSv1.3, the value includes the possible TLSv1.3 ciphersuites.

For encrypted connections that use TLS.v1.3, MySQL uses the SSL library default ciphersuite list.

For encrypted connections that use TLS protocols up through TLSv1.2, MySQL passes the following default cipher list to the SSL library.

```
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES128-SHA256
ECDHE-RSA-AES128-SHA256
ECDHE-ECDSA-AES256-SHA384
ECDHE-RSA-AES256-SHA384
DHE-RSA-AES128-GCM-SHA256
DHE-DSS-AES128-GCM-SHA256
DHE-RSA-AES128-SHA256
DHE-DSS-AES128-SHA256
DHE-DSS-AES256-GCM-SHA384
DHE-RSA-AES256-SHA256
DHE-DSS-AES256-SHA256
ECDHE-RSA-AES128-SHA
```

```
ECDHE-ECDSA-AES128-SHA
ECDHE-RSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA
DHE-DSS-AES128-SHA
DHE-RSA-AES128-SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
DHE-RSA-AES256-SHA
AES128-GCM-SHA256
DH-DSS-AES128-GCM-SHA256
ECDH-ECDSA-AES128-GCM-SHA256
AES256-GCM-SHA384
DH-DSS-AES256-GCM-SHA384
ECDH-ECDSA-AES256-GCM-SHA384
AES128-SHA256
DH-DSS-AES128-SHA256
ECDH-ECDSA-AES128-SHA256
AES256-SHA256
DH-DSS-AES256-SHA256
ECDH-ECDSA-AES256-SHA384
AES128-SHA
DH-DSS-AES128-SHA
ECDH-ECDSA-AES128-SHA
AES256-SHA
DH-DSS-AES256-SHA
ECDH-ECDSA-AES256-SHA
DHE-RSA-AES256-GCM-SHA384
DH-RSA-AES128-GCM-SHA256
ECDH-RSA-AES128-GCM-SHA256
DH-RSA-AES256-GCM-SHA384
ECDH-RSA-AES256-GCM-SHA384
DH-RSA-AES128-SHA256
ECDH-RSA-AES128-SHA256
DH-RSA-AES256-SHA256
ECDH-RSA-AES256-SHA384
ECDHE-RSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA
ECDHE-RSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA
DHE-DSS-AES128-SHA
DHE-RSA-AES128-SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA
DHE-RSA-AES256-SHA
AES128-SHA
DH-DSS-AES128-SHA
ECDH-ECDSA-AES128-SHA
AES256-SHA
DH-DSS-AES256-SHA
ECDH-ECDSA-AES256-SHA
DH-RSA-AES128-SHA
ECDH-RSA-AES128-SHA
DH-RSA-AES256-SHA
ECDH-RSA-AES256-SHA
DES-CBC3-SHA
```

These cipher restrictions are in place:

- The following ciphers are permanently restricted:

```
!DHE-DSS-DES-CBC3-SHA
!DHE-RSA-DES-CBC3-SHA
!ECDH-RSA-DES-CBC3-SHA
!ECDH-ECDSA-DES-CBC3-SHA
!ECDHE-RSA-DES-CBC3-SHA
!ECDHE-ECDSA-DES-CBC3-SHA
```

- The following categories of ciphers are permanently restricted:

```
!aNULL
!eNULL
!EXPORT
!LOW
!MD5
!DES
!RC2
!RC4
!PSK
!SSLv3
```

If the server is started with an `--ssl-cert` option specifying a certificate that uses any of the preceding restricted ciphers or cipher categories, the server starts with support for encrypted connections disabled.

Connection TLS Protocol Negotiation

Connection attempts in MySQL negotiate use of the highest TLS protocol version available on both sides for which a protocol-compatible encryption cipher is available on both sides. The negotiation process depends on factors such as the SSL library used to compile the server and client, the TLS protocol and encryption cipher configuration, and which key size is used:

- For a connection attempt to succeed, the server and client TLS protocol configuration must permit some protocol in common.
- Similarly, the server and client encryption cipher configuration must permit some cipher in common. A given cipher may work only with particular TLS protocols, so a protocol available to the negotiation process is not chosen unless there is also a compatible cipher.
- If TLSv1.3 is available, it is used if possible. (This means that server and client configuration both must permit TLSv1.3, and both must also permit some TLSv1.3-compatible encryption cipher.) Otherwise, MySQL continues through the list of available protocols, using TLSv1.2 if possible, and so forth. Negotiation proceeds from more secure protocols to less secure. Negotiation order is independent of the order in which protocols are configured. For example, negotiation order is the same regardless of whether `tls_version` has a value of `TLSv1,TLSv1.1,TLSv1.2,TLSv1.3` or `TLSv1.3,TLSv1.2,TLSv1.1,TLSv1`.
- TLSv1.2 does not work with all ciphers that have a key size of 512 bits or less. To use this protocol with such a key, use `--ssl-cipher` to specify the cipher name explicitly:

```
AES128-SHA
AES128-SHA256
AES256-SHA
AES256-SHA256
CAMELLIA128-SHA
CAMELLIA256-SHA
DES-CBC3-SHA
DHE-RSA-AES256-SHA
RC4-MD5
RC4-SHA
SEED-SHA
```

- For better security, use a certificate with an RSA key size of at least 2048 bits.

If the server and client do not have a permitted protocol in common, and a protocol-compatible cipher in common, the server terminates the connection request. Examples:

- If the server is configured with `tls_version=TLSv1.1,TLSv1.2`:

  - Connection attempts fail for clients invoked with `--tls-version=TLSv1`, and for older clients that support only TLSv1.
  - Similarly, connection attempts fail for replication slaves configured with `MASTER_TLS_VERSION = 'TLSv1'`, and for older slaves that support only TLSv1.
- If the server is configured with `tls_version=TLSv1` or is an older server that supports only TLSv1:
  - Connection attempts fail for clients invoked with `--tls-version=TLSv1.1,TLSv1.2`.
  - Similarly, connection attempts fail for replication slaves configured with `MASTER_TLS_VERSION = 'TLSv1.1,TLSv1.2'`.

MySQL permits specifying a list of protocols to support. This list is passed directly down to the underlying SSL library and is ultimately up to that library what protocols it actually enables from the supplied list. Please refer to the MySQL source code and the OpenSSL `SSL_CTX_new()` documentation for information about how the SSL library handles this.

Monitoring Current Session TLS Protocol and Cipher

To determine which encryption TLS protocol and cipher the current session uses, check the session values of the `Ssl_version` and `Ssl_cipher` status variables:

```
mysql> SELECT * FROM performance_schema.session_status
       WHERE VARIABLE_NAME IN ('Ssl_version','Ssl_cipher');
+---------------+-------------------------+
| VARIABLE_NAME | VARIABLE_VALUE          |
+---------------+-------------------------+
| Ssl_cipher    | DHE-RSA-AES128-GCM-SHA256 |
| Ssl_version   | TLSv1.2                 |
+---------------+-------------------------+
```

If the connection is not encrypted, both variables have an empty value.

6.3.3 Creating SSL and RSA Certificates and Keys

The following discussion describes how to create the files required for SSL and RSA support in MySQL. File creation can be performed using facilities provided by MySQL itself, or by invoking the `openssl` command directly.

SSL certificate and key files enable MySQL to support encrypted connections using SSL. See Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

RSA key files enable MySQL to support secure password exchange over unencrypted connections for accounts authenticated by the `sha256_password` or `caching_sha2_password` plugin. See Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

6.3.3.1 Creating SSL and RSA Certificates and Keys using MySQL

MySQL provides these ways to create the SSL certificate and key files and RSA key-pair files required to support encrypted connections using SSL and secure password exchange using RSA over unencrypted connections, if those files are missing:

- The server can autogenerate these files at startup, for MySQL distributions compiled using OpenSSL.

- Users can invoke the `mysql_ssl_rsa_setup` utility manually.
- For some distribution types, such as RPM packages, `mysql_ssl_rsa_setup` invocation occurs during data directory initialization. In this case, the MySQL distribution need not have been compiled using OpenSSL as long as the `openssl` command is available.

Important

Server autogeneration and `mysql_ssl_rsa_setup` help lower the barrier to using SSL by making it easier to generate the required files. However, certificates generated by these methods are self-signed, which may not be very secure. After you gain experience using such files, consider obtaining certificate/key material from a registered certificate authority.

- Automatic SSL and RSA File Generation
- Manual SSL and RSA File Generation Using mysql_ssl_rsa_setup
- SSL and RSA File Characteristics

Automatic SSL and RSA File Generation

For MySQL distributions compiled using OpenSSL, the MySQL server has the capability of automatically generating missing SSL and RSA files at startup. The `auto_generate_certs`, `sha256_password_auto_generate_rsa_keys`, and `caching_sha2_password_auto_generate_rsa_keys` system variables control automatic generation of these files. These variables are enabled by default. They can be enabled at startup and inspected but not set at runtime.

At startup, the server automatically generates server-side and client-side SSL certificate and key files in the data directory if the `auto_generate_certs` system variable is enabled, no SSL options other than `--ssl` are specified, and the server-side SSL files are missing from the data directory. These files enable encrypted client connections using SSL; see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

1. The server checks the data directory for SSL files with the following names:

```
ca.pem
server-cert.pem
server-key.pem
```

2. If any of those files are present, the server creates no SSL files. Otherwise, it creates them, plus some additional files:

```
ca.pem               Self-signed CA certificate
ca-key.pem           CA private key
server-cert.pem      Server certificate
server-key.pem       Server private key
client-cert.pem      Client certificate
client-key.pem       Client private key
```

3. If the server autogenerates SSL files, it uses the names of the `ca.pem`, `server-cert.pem`, and `server-key.pem` files to set the corresponding system variables (`ssl_ca`, `ssl_cert`, `ssl_key`).

At startup, the server automatically generates RSA private/public key-pair files in the data directory if all of these conditions are true: The `sha256_password_auto_generate_rsa_keys` or `caching_sha2_password_auto_generate_rsa_keys` system variable is enabled; no RSA

options are specified; the RSA files are missing from the data directory. These key-pair files enable secure password exchange using RSA over unencrypted connections for accounts authenticated by the `sha256_password` or `caching_sha2_password` plugin; see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

1. The server checks the data directory for RSA files with the following names:

```
private_key.pem      Private member of private/public key pair
public_key.pem       Public member of private/public key pair
```

2. If any of these files are present, the server creates no RSA files. Otherwise, it creates them.

3. If the server autogenerates the RSA files, it uses their names to set the corresponding system variables (`sha256_password_private_key_path` and `sha256_password_public_key_path`; `caching_sha2_password_private_key_path` and `caching_sha2_password_public_key_path`).

Manual SSL and RSA File Generation Using mysql_ssl_rsa_setup

MySQL distributions include a `mysql_ssl_rsa_setup` utility that can be invoked manually to generate SSL and RSA files. This utility is included with all MySQL distributions, but it does require that the `openssl` command be available. For usage instructions, see Section 4.4.3, “`mysql_ssl_rsa_setup` — Create SSL/RSA Files”.

SSL and RSA File Characteristics

SSL and RSA files created automatically by the server or by invoking `mysql_ssl_rsa_setup` have these characteristics:

- SSL and RSA keys are have a size of 2048 bits.
- The SSL CA certificate is self signed.
- The SSL server and client certificates are signed with the CA certificate and key, using the `sha256WithRSAEncryption` signature algorithm.
- SSL certificates use these Common Name (CN) values, with the appropriate certificate type (CA, Server, Client):

```
ca.pem:         MySQL_Server_suffix_Auto_Generated_CA_Certificate
server-cert.pm: MySQL_Server_suffix_Auto_Generated_Server_Certificate
client-cert.pm: MySQL_Server_suffix_Auto_Generated_Client_Certificate
```

  The *`suffix`* value is based on the MySQL version number. For files generated by `mysql_ssl_rsa_setup`, the suffix can be specified explicitly using the `--suffix` option.

  For files generated by the server, if the resulting CN values exceed 64 characters, the *`_suffix`* portion of the name is omitted.

- SSL files have blank values for Country (C), State or Province (ST), Organization (O), Organization Unit Name (OU) and email address.
- SSL files created by the server or by `mysql_ssl_rsa_setup` are valid for ten years from the time of generation.
- RSA files do not expire.

- SSL files have different serial numbers for each certificate/key pair (1 for CA, 2 for Server, 3 for Client).
- Files created automatically by the server are owned by the account that runs the server. Files created using `mysql_ssl_rsa_setup` are owned by the user who invoked that program. This can be changed on systems that support the `chown()` system call if the program is invoked by `root` and the `--uid` option is given to specify the user who should own the files.
- On Unix and Unix-like systems, the file access mode is 644 for certificate files (that is, world readable) and 600 for key files (that is, accessible only by the account that runs the server).

To see the contents of an SSL certificate (for example, to check the range of dates over which it is valid), invoke `openssl` directly:

```
openssl x509 -text -in ca.pem
openssl x509 -text -in server-cert.pem
openssl x509 -text -in client-cert.pem
```

It is also possible to check SSL certificate expiration information using this SQL statement:

```
mysql> SHOW STATUS LIKE 'Ssl_server_not%';
+-----------------------+--------------------------+
| Variable_name         | Value                    |
+-----------------------+--------------------------+
| Ssl_server_not_after  | Apr 28 14:16:39 2027 GMT |
| Ssl_server_not_before | May  1 14:16:39 2017 GMT |
+-----------------------+--------------------------+
```

6.3.3.2 Creating SSL Certificates and Keys Using openssl

This section describes how to use the `openssl` command to set up SSL certificate and key files for use by MySQL servers and clients. The first example shows a simplified procedure such as you might use from the command line. The second shows a script that contains more detail. The first two examples are intended for use on Unix and both use the `openssl` command that is part of OpenSSL. The third example describes how to set up SSL files on Windows.

Note

There are easier alternatives to generating the files required for SSL than the procedure described here: Let the server autogenerate them or use the `mysql_ssl_rsa_setup` program. See Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”.

Important

Whatever method you use to generate the certificate and key files, the Common Name value used for the server and client certificates/keys must each differ from the Common Name value used for the CA certificate. Otherwise, the certificate and key files will not work for servers compiled using OpenSSL. A typical error in this case is:

```
ERROR 2026 (HY000): SSL connection error:
error:00000001:lib(0):func(0):reason(1)
```

- Example 1: Creating SSL Files from the Command Line on Unix
- Example 2: Creating SSL Files Using a Script on Unix

- Example 3: Creating SSL Files on Windows

Example 1: Creating SSL Files from the Command Line on Unix

The following example shows a set of commands to create MySQL server and client certificate and key files. You will need to respond to several prompts by the `openssl` commands. To generate test files, you can press Enter to all prompts. To generate files for production use, you should provide nonempty responses.

```
# Create clean environment
rm -rf newcerts
mkdir newcerts && cd newcerts

# Create CA certificate
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 3600 \
        -key ca-key.pem -out ca.pem

# Create server certificate, remove passphrase, and sign it
# server-cert.pem = public key, server-key.pem = private key
openssl req -newkey rsa:2048 -days 3600 \
        -nodes -keyout server-key.pem -out server-req.pem
openssl rsa -in server-key.pem -out server-key.pem
openssl x509 -req -in server-req.pem -days 3600 \
        -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out server-cert.pem

# Create client certificate, remove passphrase, and sign it
# client-cert.pem = public key, client-key.pem = private key
openssl req -newkey rsa:2048 -days 3600 \
        -nodes -keyout client-key.pem -out client-req.pem
openssl rsa -in client-key.pem -out client-key.pem
openssl x509 -req -in client-req.pem -days 3600 \
        -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out client-cert.pem
```

After generating the certificates, verify them:

```
openssl verify -CAfile ca.pem server-cert.pem client-cert.pem
```

You should see a response like this:

```
server-cert.pem: OK
client-cert.pem: OK
```

To see the contents of a certificate (for example, to check the range of dates over which a certificate is valid), invoke `openssl` like this:

```
openssl x509 -text -in ca.pem
openssl x509 -text -in server-cert.pem
openssl x509 -text -in client-cert.pem
```

Now you have a set of files that can be used as follows:

- `ca.pem`: Use this as the argument to `--ssl-ca` on the server and client sides. (The CA certificate, if used, must be the same on both sides.)
- `server-cert.pem`, `server-key.pem`: Use these as the arguments to `--ssl-cert` and `--ssl-key` on the server side.
- `client-cert.pem`, `client-key.pem`: Use these as the arguments to `--ssl-cert` and `--ssl-key` on the client side.

For additional usage instructions, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

Example 2: Creating SSL Files Using a Script on Unix

Here is an example script that shows how to set up SSL certificate and key files for MySQL. After executing the script, use the files for SSL connections as described in Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

```
DIR=`pwd`/openssl
PRIV=$DIR/private

mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/ca.pem \
    -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ................++++++
# .........++++++
# writing new private key to '/home/jones/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
    $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..++++++
# ..........++++++
# writing new private key to '/home/jones/openssl/server-key.pem'
```

```
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca -cert $DIR/ca.pem -policy policy_anything \
    -out $DIR/server-cert.pem -config $DIR/openssl.cnf \
    -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
    $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..................................++++++
# ..........................................++++++
# writing new private key to '/home/jones/openssl/client-key.pem'
# Enter PEM pass phrase:
```

```
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#

openssl ca -cert $DIR/ca.pem -policy policy_anything \
    -out $DIR/client-cert.pem -config $DIR/openssl.cnf \
    -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/jones/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cat <<EOF > $DIR/my.cnf
[client]
ssl-ca=$DIR/ca.pem
ssl-cert=$DIR/client-cert.pem
ssl-key=$DIR/client-key.pem
[mysqld]
ssl-ca=$DIR/ca.pem
ssl-cert=$DIR/server-cert.pem
ssl-key=$DIR/server-key.pem
```

```
EOF
```

Example 3: Creating SSL Files on Windows

Download OpenSSL for Windows if it is not installed on your system. An overview of available packages can be seen here:

```
http://www.slproweb.com/products/Win32OpenSSL.html
```

Choose the Win32 OpenSSL Light or Win64 OpenSSL Light package, depending on your architecture (32-bit or 64-bit). The default installation location will be `C:\OpenSSL-Win32` or `C:\OpenSSL-Win64`, depending on which package you downloaded. The following instructions assume a default location of `C:\OpenSSL-Win32`. Modify this as necessary if you are using the 64-bit package.

If a message occurs during setup indicating `'...critical component is missing: Microsoft Visual C++ 2008 Redistributables'`, cancel the setup and download one of the following packages as well, again depending on your architecture (32-bit or 64-bit):

- Visual C++ 2008 Redistributables (x86), available at:

  ```
  http://www.microsoft.com/downloads/details.aspx?familyid=9B2DA534-3E03-4391-8A4D-074B9F2BC1BF
  ```

- Visual C++ 2008 Redistributables (x64), available at:

  ```
  http://www.microsoft.com/downloads/details.aspx?familyid=bd2a6171-e2d6-4230-b809-9a8d7548c1b6
  ```

After installing the additional package, restart the OpenSSL setup procedure.

During installation, leave the default `C:\OpenSSL-Win32` as the install path, and also leave the default option `'Copy OpenSSL DLL files to the Windows system directory'` selected.

When the installation has finished, add `C:\OpenSSL-Win32\bin` to the Windows System Path variable of your server (depending on your version of Windows, the following path-setting instructions might differ slightly):

1. On the Windows desktop, right-click the **My Computer** icon, and select **Properties**.
2. Select the **Advanced** tab from the **System Properties** menu that appears, and click the **Environment Variables** button.
3. Under **System Variables**, select **Path**, then click the **Edit** button. The **Edit System Variable** dialogue should appear.
4. Add `';C:\OpenSSL-Win32\bin'` to the end (notice the semicolon).
5. Press OK 3 times.
6. Check that OpenSSL was correctly integrated into the Path variable by opening a new command console (`Start>Run>cmd.exe`) and verifying that OpenSSL is available:

   ```
   Microsoft Windows [Version ...]
   Copyright (c) 2006 Microsoft Corporation. All rights reserved.

   C:\Windows\system32>cd \

   C:\>openssl
   OpenSSL> exit <<< If you see the OpenSSL prompt, installation was successful.
   ```

```
C:\>
```

After OpenSSL has been installed, use instructions similar to those from Example 1 (shown earlier in this section), with the following changes:

- Change the following Unix commands:

```
# Create clean environment
rm -rf newcerts
mkdir newcerts && cd newcerts
```

  On Windows, use these commands instead:

```
# Create clean environment
md c:\newcerts
cd c:\newcerts
```

- When a `'\'` character is shown at the end of a command line, this `'\'` character must be removed and the command lines entered all on a single line.

After generating the certificate and key files, to use them for SSL connections, see Section 6.3.1, "Configuring MySQL to Use Encrypted Connections".

6.3.3.3 Creating RSA Keys Using openssl

This section describes how to use the `openssl` command to set up the RSA key files that enable MySQL to support secure password exchange over unencrypted connections for accounts authenticated by the `sha256_password` and `caching_sha2_password` plugins.

Note

There are easier alternatives to generating the files required for RSA than the procedure described here: Let the server autogenerate them or use the `mysql_ssl_rsa_setup` program. See Section 6.3.3.1, "Creating SSL and RSA Certificates and Keys using MySQL".

To create the RSA private and public key-pair files, run these commands while logged into the system account used to run the MySQL server so the files will be owned by that account:

```
openssl genrsa -out private_key.pem 2048
openssl rsa -in private_key.pem -pubout -out public_key.pem
```

Those commands create 2,048-bit keys. To create stronger keys, use a larger value.

Then set the access modes for the key files. The private key should be readable only by the server, whereas the public key can be freely distributed to client users:

```
chmod 400 private_key.pem
chmod 444 public_key.pem
```

6.3.4 Connecting to MySQL Remotely from Windows with SSH

This section describes how to get an encrypted connection to a remote MySQL server with SSH. The information was provided by David Carlson `<dcarlson@mplcomm.com>`.

1. Install an SSH client on your Windows machine. For a comparison of SSH clients, see http://en.wikipedia.org/wiki/Comparison_of_SSH_clients.

2. Start your Windows SSH client. Set `Host_Name = yourmysqlserver_URL_or_IP`. Set `userid=your_userid` to log in to your server. This `userid` value might not be the same as the user name of your MySQL account.

3. Set up port forwarding. Either do a remote forward (Set `local_port: 3306`, `remote_host: yourmysqlservername_or_ip`, `remote_port: 3306`) or a local forward (Set `port: 3306`, `host: localhost`, `remote port: 3306`).

4. Save everything, otherwise you will have to redo it the next time.

5. Log in to your server with the SSH session you just created.

6. On your Windows machine, start some ODBC application (such as Access).

7. Create a new file in Windows and link to MySQL using the ODBC driver the same way you normally do, except type in `localhost` for the MySQL host server, not `yourmysqlservername`.

At this point, you should have an ODBC connection to MySQL, encrypted using SSH.

6.4 Security Components and Plugins

MySQL includes several components and plugins that implement security features:

- Plugins for authenticating attempts by clients to connect to MySQL Server. Plugins are available for several authentication protocols. For general discussion of the authentication process, see Section 6.2.17, “Pluggable Authentication”. For characteristics of specific authentication plugins, see Section 6.4.1, “Authentication Plugins”.

- A password-validation component for implementing password strength policies and assessing the strength of potential passwords. See Section 6.4.3, “The Password Validation Component”.

- Keyring plugins that provide secure storage for sensitive information. See Section 6.4.4, “The MySQL Keyring”.

- (MySQL Enterprise Edition only) MySQL Enterprise Audit, implemented using a server plugin, uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines. See Section 6.4.5, “MySQL Enterprise Audit”.

- A user-defined function enables applications to add their own message events to the audit log. See Section 6.4.6, “The Audit Message Component”.

- (MySQL Enterprise Edition only) MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against whitelists of accepted statement patterns. This helps harden MySQL Server against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics. See Section 6.4.7, “MySQL Enterprise Firewall”.

- (MySQL Enterprise Edition only) MySQL Enterprise Data Masking and De-Identification, implemented as a plugin library containing a plugin and a set of user-defined functions. Data masking hides sensitive information by replacing real values with substitutes. MySQL Enterprise Data Masking and De-Identification functions enable masking existing data using several methods such as obfuscation (removing identifying characteristics), generation of formatted random data, and data replacement or substitution. See Section 6.4.8, “MySQL Enterprise Data Masking and De-Identification”.

6.4.1 Authentication Plugins

The following sections describe pluggable authentication methods available in MySQL and the plugins that implement these methods. For general discussion of the authentication process, see Section 6.2.17, “Pluggable Authentication”.

The default plugin is indicated by the value of the `default_authentication_plugin` system variable.

6.4.1.1 Native Pluggable Authentication

MySQL includes a `mysql_native_password` plugin that implements native authentication; that is, authentication based on the password hashing method in use from before the introduction of pluggable authentication.

The following table shows the plugin names on the server and client sides.

Table 6.12 Plugin and Library Names for Native Password Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`mysql_native_password`
Client-side plugin	`mysql_native_password`
Library file	None (plugins are built in)

The following sections provide installation and usage information specific to native pluggable authentication:

- Installing Native Pluggable Authentication
- Using Native Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”.

Installing Native Pluggable Authentication

The `mysql_native_password` plugin exists in server and client forms:

- The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.
- The client-side plugin is built into the `libmysqlclient` client library and is available to any program linked against `libmysqlclient`.

Using Native Pluggable Authentication

MySQL client programs use `mysql_native_password` by default. The `--default-auth` option can be used as a hint about which client-side plugin the program can expect to use:

```
shell> mysql --default-auth=mysql_native_password ...
```

6.4.1.2 Caching SHA-2 Pluggable Authentication

MySQL provides two authentication plugins that implement SHA-256 hashing for user account passwords:

- `sha256_password`: Implements basic SHA-256 authentication.
- `caching_sha2_password`: Implements SHA-256 authentication (like `sha256_password`), but uses caching on the server side for better performance and has additional features for wider applicability.

This section describes the caching SHA-2 authentication plugin. For information about the original basic (noncaching) plugin, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”.

Important

In MySQL 8.0, `caching_sha2_password` is the default authentication plugin rather than `mysql_native_password`. For information about the implications of this change for server operation and compatibility of the server with clients and connectors, see caching_sha2_password as the Preferred Authentication Plugin.

Important

To connect to the server using an account that authenticates with the `caching_sha2_password` plugin, you must use either a secure connection or an unencrypted connection that supports password exchange using an RSA key pair, as described later in this section. Either way, the `caching_sha2_password` plugin uses MySQL's encryption capabilities. See Section 6.3, “Using Encrypted Connections”.

Note

In the name `sha256_password`, “sha256” refers to the 256-bit digest length the plugin uses for encryption. In the name `caching_sha2_password`, “sha2” refers more generally to the SHA-2 class of encryption algorithms, of which 256-bit encryption is one instance. The latter name choice leaves room for future expansion of possible digest lengths without changing the plugin name.

The `caching_sha2_password` plugin has these advantages, compared to `sha256_password`:

- On the server side, an in-memory cache enables faster reauthentication of users who have connected previously when they connect again.
- RSA-based password exchange is available regardless of the SSL library against which MySQL is linked.
- Support is provided for client connections that use the Unix socket-file and shared-memory protocols.

The following table shows the plugin names on the server and client sides.

Table 6.13 Plugin and Library Names for SHA-2 Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`caching_sha2_password`
Client-side plugin	`caching_sha2_password`
Library file	None (plugins are built in)

The following sections provide installation and usage information specific to caching SHA-2 pluggable authentication:

- Installing SHA-2 Pluggable Authentication
- Using SHA-2 Pluggable Authentication
- Cache Operation for SHA-2 Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”.

Installing SHA-2 Pluggable Authentication

The `caching_sha2_password` plugin exists in server and client forms:

- The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.
- The client-side plugin is built into the `libmysqlclient` client library and is available to any program linked against `libmysqlclient`.

The server-side plugin uses the `sha2_cache_cleaner` audit plugin as a helper to perform password cache management. `sha2_cache_cleaner`, like `caching_sha2_password`, is built in and need not be installed.

Using SHA-2 Pluggable Authentication

To set up an account that uses the `caching_sha2_password` plugin for SHA-256 password hashing, use the following statement, where *`password`* is the desired account password:

```
CREATE USER 'sha2user'@'localhost'
IDENTIFIED WITH caching_sha2_password BY 'password';
```

The server assigns the `caching_sha2_password` plugin to the account and uses it to encrypt the password using SHA-256, storing those values in the `plugin` and `authentication_string` columns of the `mysql.user` system table.

The preceding instructions do not assume that `caching_sha2_password` is the default authentication plugin. If `caching_sha2_password` is the default authentication plugin, a simpler `CREATE USER` syntax can be used.

To start the server with the default authentication plugin set to `caching_sha2_password`, put these lines in the server option file:

```
[mysqld]
default_authentication_plugin=caching_sha2_password
```

That causes the `caching_sha2_password` plugin to be used by default for new accounts. As a result, it is possible to create the account and set its password without naming the plugin explicitly:

```
CREATE USER 'sha2user'@'localhost' IDENTIFIED BY 'password';
```

Another consequence of setting `default_authentication_plugin` to `caching_sha2_password` is that, to use some other plugin for account creation, you must specify that plugin explicitly. For example, to use the `mysql_native_password` plugin, use this statement:

```
CREATE USER 'nativeuser'@'localhost'
IDENTIFIED WITH mysql_native_password BY 'password';
```

`caching_sha2_password` supports connections over secure transport. If you follow the RSA configuration procedure given later in this section, it also supports encrypted password exchange using RSA over unencrypted connections. RSA support has these characteristics:

- On the server side, two system variables name the RSA private and public key-pair files: `caching_sha2_password_private_key_path` and `caching_sha2_password_public_key_path`. The database administrator must set these variables at server startup if the key files to use have names that differ from the system variable default values.

- The server uses the `caching_sha2_password_auto_generate_rsa_keys` system variable to determine whether to automatically generate the RSA key-pair files. See Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.
- The `Caching_sha2_password_rsa_public_key` status variable displays the RSA public key value used by the `caching_sha2_password` authentication plugin.
- Clients that are in possession of the RSA public key can perform RSA key pair-based password exchange with the server during the connection process, as described later.
- For connections by accounts that authenticate with `caching_sha2_password` and RSA key pair-based password exchange, the server does not send the RSA public key to clients by default. Clients can use a client-side copy of the required public key, or request the public key from the server.

  Use of a trusted local copy of the public key enables the client to avoid a round trip in the client/server protocol, and is more secure than requesting the public key from the server. On the other hand, requesting the public key from the server is more convenient (it requires no management of a client-side file) and may be acceptable in secure network environments.

  - For command-line clients, use the `--server-public-key-path` option to specify the RSA public key file. Use the `--get-server-public-key` option to request the public key from the server. The following programs support the two options: `mysql`, `mysqlsh`, `mysqladmin`, `mysqlbinlog`, `mysqlcheck`, `mysqldump`, `mysqlimport`, `mysqlpump`, `mysqlshow`, `mysqlslap`, `mysqltest`, `mysql_upgrade`.
  - For programs that use the C API, call `mysql_options()` to specify the RSA public key file by passing the `MYSQL_SERVER_PUBLIC_KEY` option and the name of the file, or request the public key from the server by passing the `MYSQL_OPT_GET_SERVER_PUBLIC_KEY` option.
  - For replication slaves, use the `CHANGE MASTER TO` statement with the `MASTER_PUBLIC_KEY_PATH` option to specify the RSA public key file, or the `GET_MASTER_PUBLIC_KEY` option to request the public key from the master. For Group Replication, the `group_replication_recovery_public_key_path` and `group_replication_recovery_get_public_key` system variables serve the same purpose.

  In all cases, if the option is given to specify a valid public key file, it takes precedence over the option to request the public key from the server.

For clients that use the `caching_sha2_password` plugin, passwords are never exposed as cleartext when connecting to the server. How password transmission occurs depends on whether a secure connection or RSA encryption is used:

- If the connection is secure, an RSA key pair is unnecessary and is not used. This applies to encrypted TCP connections that use TLS, as well as Unix socket-file and shared-memory connections. The password is sent as cleartext but cannot be snooped because the connection is secure.
- If the connection is not secure, an RSA key pair is used. This applies to unencrypted TCP connections without TLS and named-pipe connections. RSA is used only for password exchange between client and server, to prevent password snooping. When the server receives the encrypted password, it decrypts it. A scramble is used in the encryption to prevent repeat attacks.

To enable use of an RSA key pair for password exchange during the client connection process, use the following procedure:

1. Create the RSA private and public key-pair files using the instructions in Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.

2. If the private and public key files are located in the data directory and are named `private_key.pem` and `public_key.pem` (the default values of the `caching_sha2_password_private_key_path` and `caching_sha2_password_public_key_path` system variables), the server uses them automatically at startup.

   Otherwise, to name the key files explicitly, set the system variables to the key file names in the server option file. If the files are located in the server data directory, you need not specify their full path names:

   ```
   [mysqld]
   caching_sha2_password_private_key_path=myprivkey.pem
   caching_sha2_password_public_key_path=mypubkey.pem
   ```

   If the key files are not located in the data directory, or to make their locations explicit in the system variable values, use full path names:

   ```
   [mysqld]
   caching_sha2_password_private_key_path=/usr/local/mysql/myprivkey.pem
   caching_sha2_password_public_key_path=/usr/local/mysql/mypubkey.pem
   ```

3. Restart the server, then connect to it and check the `Caching_sha2_password_rsa_public_key` status variable value. The value will differ from that shown here, but should be nonempty:

   ```
   mysql> SHOW STATUS LIKE 'Caching_sha2_password_rsa_public_key'\G
   *************************** 1. row ***************************
   Variable_name: Caching_sha2_password_rsa_public_key
           Value: -----BEGIN PUBLIC KEY-----
   MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDO9nRUDd+KvSZgY7cNBZMNpwX6
   MvE1PbJFXO7u18nJ9lwc99Du/E7lw6CVXw7VKrXPeHbVQUzGyUNkf45Nz/ckaaJa
   aLgJOBCIDmNVnyU54OT/1lcs2xiyfaDMe8fCJ64ZwTnKbY2gkt1IMjUAB5Ogd5kJ
   g8aV7EtKwyhHb0c30QIDAQAB
   -----END PUBLIC KEY-----
   ```

   If the value is empty, the server found some problem with the key files. Check the error log for diagnostic information.

After the server has been configured with the RSA key files, accounts that authenticate with the `caching_sha2_password` plugin have the option of using those key files to connect to the server. As mentioned previously, such accounts can use either a secure connection (in which case RSA is not used) or an unencrypted connection that performs password exchange using RSA. Suppose that an unencrypted connection is used. For example:

```
shell> mysql --ssl-mode=DISABLED -u sha2user -p
Enter password: password
```

For this connection attempt by `sha2user`, the server determines that `caching_sha2_password` is the appropriate authentication plugin and invokes it (because that was the plugin specified at `CREATE USER` time). The plugin finds that the connection is not encrypted and thus requires the password to be transmitted using RSA encryption. However, the server does not send the public key to the client, and the client provided no public key, so it cannot encrypt the password and the connection fails:

```
ERROR 2061 (HY000): Authentication plugin 'caching_sha2_password'
reported error: Authentication requires secure connection.
```

To request the RSA public key from the server, specify the `--get-server-public-key` option:

```
shell> mysql --ssl-mode=DISABLED -u sha2user -p --get-server-public-key
```

```
Enter password: password
```

In this case, the server sends the RSA public key to the client, which uses it to encrypt the password and returns the result to the server. The plugin uses the RSA private key on the server side to decrypt the password and accepts or rejects the connection based on whether the password is correct.

Alternatively, if the client has a file containing a local copy of the RSA public key required by the server, it can specify the file using the `--server-public-key-path` option:

```
shell> mysql --ssl-mode=DISABLED -u sha2user -p --server-public-key-path=file_name
Enter password: password
```

In this case, the client uses the public key to encrypt the password and returns the result to the server. The plugin uses the RSA private key on the server side to decrypt the password and accepts or rejects the connection based on whether the password is correct.

The public key value in the file named by the `--server-public-key-path` option should be the same as the key value in the server-side file named by the `caching_sha2_password_public_key_path` system variable. If the key file contains a valid public key value but the value is incorrect, an access-denied error occurs. If the key file does not contain a valid public key, the client program cannot use it.

Client users can obtain the RSA public key two ways:

- The database administrator can provide a copy of the public key file.
- A client user who can connect to the server some other way can use a `SHOW STATUS LIKE 'Caching_sha2_password_rsa_public_key'` statement and save the returned key value in a file.

Cache Operation for SHA-2 Pluggable Authentication

On the server side, the `caching_sha2_password` plugin uses an in-memory cache for faster authentication of clients who have connected previously. Entries consist of account-name/password-hash pairs. The cache works like this:

1. When a client connects, `caching_sha2_password` checks whether the client and password match some cache entry. If so, authentication succeeds.
2. If there is no matching cache entry, the plugin attempts to verify the client against the credentials in the `mysql.user` system table. If this succeeds, `caching_sha2_password` adds an entry for the client to the hash. Otherwise, authentication fails and the connection is rejected.

In this way, when a client first connects, authentication against the `mysql.user` system table occurs. When the client connects subsequently, faster authentication against the cache occurs.

Password cache operations other than adding entries are handled by the `sha2_cache_cleaner` audit plugin, which performs these actions on behalf of `caching_sha2_password`:

- It clears the cache entry for any account that is renamed or dropped, or any account for which the credentials or authentication plugin are changed.
- It empties the cache when the `FLUSH PRIVILEGES` statement is executed.
- It empties the cache at server shutdown. (This means the cache is not persistent across server restarts.)

Cache clearing operations affect the authentication requirements for subsequent client connections. For each user account, the first client connection for the user after any of the following operations must use a secure connection (made using TCP using TLS credentials, a Unix socket file, or shared memory) or RSA key pair-based password exchange:

- After account creation.
- After a password change for the account.
- After `RENAME USER` for the account.
- After `FLUSH PRIVILEGES`.

`FLUSH PRIVILEGES` clears the entire cache and affects all accounts that use the `caching_sha2_password` plugin. The other operations clear specific cache entries and affect only accounts that are part of the operation.

Once the user authenticates successfully, the account is entered into the cache and subsequent connections do not require a secure connection or the RSA key pair, until another cache clearing event occurs that affects the account. (When the cache can be used, the server uses a challenge-response mechanism that does not use cleartext password transmission and does not require a secure connection.)

6.4.1.3 SHA-256 Pluggable Authentication

MySQL provides two authentication plugins that implement SHA-256 hashing for user account passwords:

- `sha256_password`: Implements basic SHA-256 authentication.
- `caching_sha2_password`: Implements SHA-256 authentication (like `sha256_password`), but uses caching on the server side for better performance and has additional features for wider applicability.

This section describes the original noncaching SHA-2 authentication plugin. For information about the caching plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

Important

In MySQL 8.0, `caching_sha2_password` is the default authentication plugin rather than `mysql_native_password`. For information about the implications of this change for server operation and compatibility of the server with clients and connectors, see caching_sha2_password as the Preferred Authentication Plugin.

Because `caching_sha2_password` is the default authentication plugin in MySQL 8.0 and provides a superset of the capabilities of the `sha256_password` authentication plugin, `sha256_password` is deprecated and will be removed in a future MySQL version. MySQL accounts that authenticate using `sha256_password` should be migrated to use `caching_sha2_password` instead.

Important

To connect to the server using an account that authenticates with the `sha256_password` plugin, you must use either a TLS connection or an unencrypted connection that supports password exchange using an RSA key pair, as described later in this section. Either way, the `sha256_password` plugin uses MySQL's encryption capabilities. See Section 6.3, “Using Encrypted Connections”.

Note

In the name `sha256_password`, “sha256” refers to the 256-bit digest length the plugin uses for encryption. In the name `caching_sha2_password`, “sha2” refers more generally to the SHA-2 class of encryption algorithms, of which 256-bit encryption is one instance. The latter name choice leaves room for future expansion of possible digest lengths without changing the plugin name.

The following table shows the plugin names on the server and client sides.

Table 6.14 Plugin and Library Names for SHA-256 Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`sha256_password`
Client-side plugin	`sha256_password`
Library file	None (plugins are built in)

The following sections provide installation and usage information specific to SHA-256 pluggable authentication:

- Installing SHA-256 Pluggable Authentication
- Using SHA-256 Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”.

Installing SHA-256 Pluggable Authentication

The `sha256_password` plugin exists in server and client forms:

- The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.
- The client-side plugin is built into the `libmysqlclient` client library and is available to any program linked against `libmysqlclient`.

Using SHA-256 Pluggable Authentication

To set up an account that uses the `sha256_password` plugin for SHA-256 password hashing, use the following statement, where *`password`* is the desired account password:

```
CREATE USER 'sha256user'@'localhost'
IDENTIFIED WITH sha256_password BY 'password';
```

The server assigns the `sha256_password` plugin to the account and uses it to encrypt the password using SHA-256, storing those values in the `plugin` and `authentication_string` columns of the `mysql.user` system table.

The preceding instructions do not assume that `sha256_password` is the default authentication plugin. If `sha256_password` is the default authentication plugin, a simpler `CREATE USER` syntax can be used.

To start the server with the default authentication plugin set to `sha256_password`, put these lines in the server option file:

```
[mysqld]
default_authentication_plugin=sha256_password
```

That causes the `sha256_password` plugin to be used by default for new accounts. As a result, it is possible to create the account and set its password without naming the plugin explicitly:

```
CREATE USER 'sha256user'@'localhost' IDENTIFIED BY 'password';
```

Another consequence of setting `default_authentication_plugin` to `sha256_password` is that, to use some other plugin for account creation, you must specify that plugin explicitly. For example, to use the `mysql_native_password` plugin, use this statement:

```
CREATE USER 'nativeuser'@'localhost'
IDENTIFIED WITH mysql_native_password BY 'password';
```

`sha256_password` supports connections over secure transport. `sha256_password` also supports encrypted password exchange using RSA over unencrypted connections if MySQL is compiled using OpenSSL, and the MySQL server to which you wish to connect is configured to support RSA (using the RSA configuration procedure given later in this section).

RSA support has these characteristics:

- On the server side, two system variables name the RSA private and public key-pair files: `sha256_password_private_key_path` and `sha256_password_public_key_path`. The database administrator must set these variables at server startup if the key files to use have names that differ from the system variable default values.
- The server uses the `sha256_password_auto_generate_rsa_keys` system variable to determine whether to automatically generate the RSA key-pair files. See Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.
- The `Rsa_public_key` status variable displays the RSA public key value used by the `sha256_password` authentication plugin.
- Clients that are in possession of the RSA public key can perform RSA key pair-based password exchange with the server during the connection process, as described later.
- For connections by accounts that authenticate with `sha256_password` and RSA public key pair-based password exchange, the server sends the RSA public key to the client as needed. However, if a copy of the public key is available on the client host, the client can use it to save a round trip in the client/server protocol:
  - For these command-line clients, use the `--server-public-key-path` option to specify the RSA public key file: `mysql`, `mysqladmin`, `mysqlbinlog`, `mysqlcheck`, `mysqldump`, `mysqlimport`, `mysqlpump`, `mysqlshow`, `mysqlslap`, `mysqltest`, `mysql_upgrade`.
  - For programs that use the C API, call `mysql_options()` to specify the RSA public key file by passing the `MYSQL_SERVER_PUBLIC_KEY` option and the name of the file.
  - For replication slaves, use the `CHANGE MASTER TO` statement with the `MASTER_PUBLIC_KEY_PATH` option to specify the RSA public key file. For Group Replication, the `group_replication_recovery_get_public_key` system variable serves the same purpose.

For clients that use the `sha256_password` plugin, passwords are never exposed as cleartext when connecting to the server. How password transmission occurs depends on whether a secure connection or RSA encryption is used:

- If the connection is secure, an RSA key pair is unnecessary and is not used. This applies to encrypted connections that use TLS. The password is sent as cleartext but cannot be snooped because the connection is secure.
- If the connection is not secure, and an RSA key pair is available, the connection remains unencrypted. This applies to unencrypted connections without TLS. RSA is used only for password exchange between client and server, to prevent password snooping. When the server receives the encrypted password, it decrypts it. A scramble is used in the encryption to prevent repeat attacks.

- If a secure connection is not used and RSA encryption is not available, the connection attempt fails because the password cannot be sent without being exposed as cleartext.

Note

To use RSA password encryption with `sha256_password`, the client and server both must be compiled using OpenSSL, not just one of them.

Assuming that MySQL has been compiled using OpenSSL, use the following procedure to enable use of an RSA key pair for password exchange during the client connection process:

1. Create the RSA private and public key-pair files using the instructions in Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.

2. If the private and public key files are located in the data directory and are named `private_key.pem` and `public_key.pem` (the default values of the `sha256_password_private_key_path` and `sha256_password_public_key_path` system variables), the server uses them automatically at startup.

   Otherwise, to name the key files explicitly, set the system variables to the key file names in the server option file. If the files are located in the server data directory, you need not specify their full path names:

   ```
   [mysqld]
   sha256_password_private_key_path=myprivkey.pem
   sha256_password_public_key_path=mypubkey.pem
   ```

   If the key files are not located in the data directory, or to make their locations explicit in the system variable values, use full path names:

   ```
   [mysqld]
   sha256_password_private_key_path=/usr/local/mysql/myprivkey.pem
   sha256_password_public_key_path=/usr/local/mysql/mypubkey.pem
   ```

3. Restart the server, then connect to it and check the `Rsa_public_key` status variable value. The value will differ from that shown here, but should be nonempty:

   ```
   mysql> SHOW STATUS LIKE 'Rsa_public_key'\G
   *************************** 1. row ***************************
   Variable_name: Rsa_public_key
           Value: -----BEGIN PUBLIC KEY-----
   MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDO9nRUDd+KvSZgY7cNBZMNpwX6
   MvE1PbJFXO7u18nJ9lwc99Du/E7lw6CVXw7VKrXPeHbVQUzGyUNkf45Nz/ckaaJa
   aLgJOBCIDmNVnyU54OT/1lcs2xiyfaDMe8fCJ64ZwTnKbY2gkt1IMjUAB5Ogd5kJ
   g8aV7EtKwyhHb0c30QIDAQAB
   -----END PUBLIC KEY-----
   ```

   If the value is empty, the server found some problem with the key files. Check the error log for diagnostic information.

After the server has been configured with the RSA key files, accounts that authenticate with the `sha256_password` plugin have the option of using those key files to connect to the server. As mentioned previously, such accounts can use either a secure connection (in which case RSA is not used) or an unencrypted connection that performs password exchange using RSA. Suppose that an unencrypted connection is used. For example:

```
shell> mysql --ssl-mode=DISABLED -u sha256user -p
Enter password: password
```

For this connection attempt by `sha256user`, the server determines that `sha256_password` is the appropriate authentication plugin and invokes it (because that was the plugin specified at `CREATE USER` time). The plugin finds that the connection is not encrypted and thus requires the password to be transmitted using RSA encryption. In this case, the plugin sends the RSA public key to the client, which uses it to encrypt the password and returns the result to the server. The plugin uses the RSA private key on the server side to decrypt the password and accepts or rejects the connection based on whether the password is correct.

The server sends the RSA public key to the client as needed. However, if the client has a file containing a local copy of the RSA public key required by the server, it can specify the file using the `--server-public-key-path` option:

```
shell> mysql --ssl-mode=DISABLED -u sha256user -p --server-public-key-path=file_name
Enter password: password
```

The public key value in the file named by the `--server-public-key-path` option should be the same as the key value in the server-side file named by the `sha256_password_public_key_path` system variable. If the key file contains a valid public key value but the value is incorrect, an access-denied error occurs. If the key file does not contain a valid public key, the client program cannot use it. In this case, the `sha256_password` plugin sends the public key to the client as if no `--server-public-key-path` option had been specified.

Client users can obtain the RSA public key two ways:

- The database administrator can provide a copy of the public key file.
- A client user who can connect to the server some other way can use a `SHOW STATUS LIKE 'Rsa_public_key'` statement and save the returned key value in a file.

6.4.1.4 Client-Side Cleartext Pluggable Authentication

A client-side authentication plugin is available that enables clients to send passwords to the server as cleartext, without hashing or encryption. This plugin is built into the MySQL client library.

The following table shows the plugin name.

Table 6.15 Plugin and Library Names for Cleartext Authentication

Plugin or File	Plugin or File Name
Server-side plugin	None, see discussion
Client-side plugin	`mysql_clear_password`
Library file	None (plugin is built in)

Many client-side authentication plugins perform hashing or encryption of a password before the client sends it to the server. This enables clients to avoid sending passwords as cleartext.

Hashing or encryption cannot be done for authentication schemes that require the server to receive the password as entered on the client side. In such cases, the client-side `mysql_clear_password` plugin is used, which enables the client to send the password to the server as cleartext. There is no corresponding server-side plugin. Rather, `mysql_clear_password` can be used on the client side in concert with any server-side plugin that needs a cleartext password. (Examples are the PAM and simple LDAP authentication plugins; see Section 6.4.1.5, “PAM Pluggable Authentication”, and Section 6.4.1.7, “LDAP Pluggable Authentication”.)

The following discussion provides usage information specific to cleartext pluggable authentication. For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”.

Note

Sending passwords as cleartext may be a security problem in some configurations. To avoid problems if there is any possibility that the password would be intercepted, clients should connect to MySQL Server using a method that protects the password. Possibilities include SSL (see Section 6.3, "Using Encrypted Connections"), IPsec, or a private network.

To make inadvertent use of the `mysql_clear_password` plugin less likely, MySQL clients must explicitly enable it. This can be done in several ways:

- Set the `LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN` environment variable to a value that begins with `1`, `Y`, or `y`. This enables the plugin for all client connections.
- The `mysql`, `mysqladmin`, `mysqlcheck`, `mysqldump`, `mysqlshow`, and `mysqlslap` client programs support an `--enable-cleartext-plugin` option that enables the plugin on a per-invocation basis.
- The `mysql_options()` C API function supports a `MYSQL_ENABLE_CLEARTEXT_PLUGIN` option that enables the plugin on a per-connection basis. Also, any program that uses `libmysqlclient` and reads option files can enable the plugin by including an `enable-cleartext-plugin` option in an option group read by the client library.

6.4.1.5 PAM Pluggable Authentication

Note

PAM pluggable authentication is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition supports an authentication method that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as traditional Unix passwords or an LDAP directory.

PAM pluggable authentication provides these capabilities:

- External authentication: PAM authentication enables MySQL Server to accept connections from users defined outside the MySQL grant tables and that authenticate using methods supported by PAM.
- Proxy user support: PAM authentication can return to MySQL a user name different from the external user name passed by the client program, based on the PAM groups the external user is a member of and the authentication string provided. This means that the plugin can return the MySQL user that defines the privileges the external PAM-authenticated user should have. For example, an operating sytem user named `joe` can connect and have the privileges of a MySQL user named `developer`.

PAM pluggable authentication has been tested on Linux and macOS.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the `plugin_dir` system variable. For installation information, see Installing PAM Pluggable Authentication.

Table 6.16 Plugin and Library Names for PAM Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`authentication_pam`
Client-side plugin	`mysql_clear_password`

Plugin or File	Plugin or File Name
Library file	`authentication_pam.so`

The client-side `mysql_clear_password` cleartext plugin that communicates with the server-side PAM plugin is built into the `libmysqlclient` client library and is included in all distributions, including community distributions. Inclusion of the client-side cleartext plugin in all MySQL distributions enables clients from any distribution to connect to a server that has the server-side PAM plugin loaded.

The following sections provide installation and usage information specific to PAM pluggable authentication:

- How PAM Authentication of MySQL Users Works
- Installing PAM Pluggable Authentication
- Uninstalling PAM Pluggable Authentication
- Using PAM Pluggable Authentication
- PAM Unix Password Authentication without Proxy Users
- PAM LDAP Authentication without Proxy Users
- PAM Unix Password Authentication with Proxy Users and Group Mapping
- PAM Authentication Access to Unix Password Store
- PAM Authentication Debugging

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”. For information about the `mysql_clear_password` plugin, see Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”. For proxy user information, see Section 6.2.18, “Proxy Users”.

How PAM Authentication of MySQL Users Works

This section provides a general overview of how MySQL and PAM work together to authenticate MySQL users. For examples showing how to set up MySQL accounts to use specific PAM services, see Using PAM Pluggable Authentication.

1. The client program and the server communicate, with the client sending to the server the client user name (the operating system user name by default) and password:
   - The client user name is the external user name.
   - For accounts that use the PAM server-side authentication plugin, the corresponding client-side plugin is `mysql_clear_password`. This client-side plugin performs no password hashing, with the result that the client sends the password to the server as cleartext.
2. The server finds a matching MySQL account based on the external user name and the host from which the client connects. The PAM plugin uses the information passed to it by MySQL Server (such as user name, host name, password, and authentication string). When you define a MySQL account that authenticates using PAM, the authentication string contains:
   - A PAM service name, which is a name that the system administrator can use to refer to an authentication method for a particular application. There can be multiple applications associated with a single database server instance, so the choice of service name is left to the SQL application developer.
   - Optionally, if proxying is to be used, a mapping from PAM groups to MySQL user names.

3. The plugin uses the PAM service named in the authentication string to check the user credentials and returns `'Authentication succeeded, Username is user_name'` or `'Authentication failed'`. The password must be appropriate for the password store used by the PAM service. Examples:

   - For traditional Unix passwords, the service looks up passwords stored in the `/etc/shadow` file.
   - For LDAP, the service looks up passwords stored in an LDAP directory.

   If the credentials check fails, the server refuses the connection.

4. Otherwise, the authentication string indicates whether proxying occurs. If the string contains no PAM group mapping, proxying does not occur. In this case, the MySQL user name is the same as the external user name.

5. Otherwise, proxying is indicated based on the PAM group mapping, with the MySQL user name determined based on the first matching group in the mapping list. The meaning of “PAM group” depends on the PAM service. Examples:

   - For traditional Unix passwords, groups are Unix groups defined in the `/etc/group` file, possibly supplemented with additional PAM information in a file such as `/etc/security/group.conf`.
   - For LDAP, groups are LDAP groups defined in an LDAP directory.

   If the proxy user (the external user) has the `PROXY` privilege for the proxied MySQL user name, proxying occurs, with the proxy user assuming the privileges of the proxied user.

Installing PAM Pluggable Authentication

This section describes how to install the PAM authentication plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `authentication_pam`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=authentication_pam.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN authentication_pam SONAME 'authentication_pam.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%pam%';
+--------------------+---------------+
| PLUGIN_NAME        | PLUGIN_STATUS |
+--------------------+---------------+
| authentication_pam | ACTIVE        |
+--------------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the PAM plugin, see Using PAM Pluggable Authentication.

Uninstalling PAM Pluggable Authentication

The method used to uninstall the PAM authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN authentication_pam;
```

Using PAM Pluggable Authentication

This section describes in general terms how to use the PAM authentication plugin to connect from MySQL client programs to the server. The following sections provide instructions for using PAM authentication in specific ways. It is assumed that the server is running with the server-side PAM plugin enabled, as described in Installing PAM Pluggable Authentication.

To refer to the PAM authentication plugin in the `IDENTIFIED WITH` clause of a `CREATE USER` statement, use the name `authentication_pam`. For example:

```
CREATE USER user
  IDENTIFIED WITH authentication_pam
  AS 'auth_string';
```

The authentication string specifies the following types of information:

- The PAM service name (see How PAM Authentication of MySQL Users Works). Examples in the following discussion use a service name of `mysql-unix` for authentication using traditional Unix passwords, and `mysql-ldap` for authentication using LDAP.
- For proxy support, PAM provides a way for a PAM module to return to the server a MySQL user name other than the external user name passed by the client program when it connects to the server. Use the authentication string to control the mapping from external user names to MySQL user names. If you want to take advantage of proxy user capabilities, the authentication string must include this kind of mapping.

For example, if an account uses the `mysql-unix` PAM service name and should map operating system users in the `root` and `users` PAM groups to the `developer` and `data_entry` MySQL users, respectively, use a statement like this:

```
CREATE USER user
```

```
IDENTIFIED WITH authentication_pam
AS 'mysql-unix, root=developer, users=data_entry';
```

Authentication string syntax for the PAM authentication plugin follows these rules:

- The string consists of a PAM service name, optionally followed by a PAM group mapping list consisting of one or more keyword/value pairs each specifying a PAM group name and a MySQL user name:

  ```
  pam_service_name[,pam_group_name=mysql_user_name]...
  ```

  The plugin parses the authentication string for each connection attempt that uses the account. To minimize overhead, keep the string as short as possible.

- Each *`pam_group_name=mysql_user_name`* pair must be preceded by a comma.

- Leading and trailing spaces not inside double quotation marks are ignored.

- Unquoted *`pam_service_name`*, *`pam_group_name`*, and *`mysql_user_name`* values can contain anything except equal sign, comma, or space.

- If a *`pam_service_name`*, *`pam_group_name`*, or *`mysql_user_name`* value is quoted with double quotation marks, everything between the quotation marks is part of the value. This is necessary, for example, if the value contains space characters. All characters are legal except double quotation mark and backslash (`\`). To include either character, escape it with a backslash.

If the plugin successfully authenticates the external user name (the name passed by the client), it looks for a PAM group mapping list in the authentication string and, if present, uses it to return a different MySQL user name to the MySQL server based on which PAM groups the external user is a member of:

- If the authentication string contains no PAM group mapping list, the plugin returns the external name.

- If the authentication string does contain a PAM group mapping list, the plugin examines each *`pam_group_name=mysql_user_name`* pair in the list from left to right and tries to find a match for the *`pam_group_name`* value in a non-MySQL directory of the groups assigned to the authenticated user and returns *`mysql_user_name`* for the first match it finds. If the plugin finds no match for any PAM group, it returns the external name. If the plugin is not capable of looking up a group in a directory, it ignores the PAM group mapping list and returns the external name.

The following sections describe how to set up several authentication scenarios that use the PAM authentication plugin:

- No proxy users. This uses PAM only to check login names and passwords. Every external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use PAM authentication. (For a MySQL account of *`'user_name'@'host_name'`* to match the external user, *`user_name`* must be the external user name and *`host_name`* must match the host from which the client connects.) Authentication can be performed by various PAM-supported methods. Later discussion shows how to authenticate client credentials using traditional Unix passwords, and passwords in LDAP.

  PAM authentication, when not done through proxy users or PAM groups, requires the MySQL user name to be same as the operating system user name. MySQL user names are limited to 32 characters (see Section 6.2.3, “Grant Tables”), which limits PAM nonproxy authentication to Unix accounts with names of at most 32 characters.

- Proxy users only, with PAM group mapping. For this scenario, create one or more MySQL accounts that define different sets of privileges. (Ideally, nobody should connect using those accounts directly.) Then define a default user authenticating through PAM that uses some mapping scheme (usually based on the external PAM groups the users are members of) to map all the external user names to the few

MySQL accounts holding the privilege sets. Any client who connects and specifies an external user name as the client user name is mapped to one of the MySQL accounts and uses its privileges. The discussion shows how to set this up using traditional Unix passwords, but other PAM methods such as LDAP could be used instead.

Variations on these scenarios are possible:

- You can permit some users to log in directly (without proxying) but require others to connect through proxy accounts.
- You can use one PAM authentication method for some users, and another method for other users, by using differing PAM service names among your PAM-authenticated accounts. For example, you can use the `mysql-unix` PAM service for some users, and `mysql-ldap` for others.

The examples make the following assumptions. You might need to make some adjustments if your system is set up differently.

- The login name and password are `antonio` and *`antonio_password`*, respectively. Change these to correspond to the user you want to authenticate.
- The PAM configuration directory is `/etc/pam.d`.
- The PAM service name corresponds to the authentication method (`mysql-unix` or `mysql-ldap` in this discussion). To use a given PAM service, you must set up a PAM file with the same name in the PAM configuration directory (creating the file if it does not exist). In addition, you must name the PAM service in the authentication string of the `CREATE USER` statement for any account that authenticates using that PAM service.

The PAM authentication plugin checks at initialization time whether the `AUTHENTICATION_PAM_LOG` environment value is set in the server's startup environment. If so, the plugin enables logging of diagnostic messages to the standard output. Depending on how your server is started, the message might appear on the console or in the error log. These messages can be helpful for debugging PAM-related issues that occur when the plugin performs authentication. For more information, see PAM Authentication Debugging.

PAM Unix Password Authentication without Proxy Users

This authentication scenario uses PAM to check external users defined in terms of operating system user names and Unix passwords, without proxying. Every such external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use PAM authentication through traditional Unix password store.

Note

Traditional Unix passwords are checked using the `/etc/shadow` file. For information regarding possible issues related to this file, see PAM Authentication Access to Unix Password Store.

1. Verify that Unix authentication permits logins to the operating system with the user name `antonio` and password *`antonio_password`*.
2. Set up PAM to authenticate MySQL connections using traditional Unix passwords by creating a `mysql-unix` PAM service file named `/etc/pam.d/mysql-unix`. The file contents are system dependent, so check existing login-related files in the `/etc/pam.d` directory to see what they look like. On Linux, the `mysql-unix` file might look like this:

```
#%PAM-1.0
auth            include         password-auth
account         include         password-auth
```

For macOS, use `login` rather than `password-auth`.

The PAM file format might differ on some systems. For example, on Ubuntu and other Debian-based systems, use these file contents instead:

```
@include common-auth
@include common-account
@include common-session-noninteractive
```

3. Create a MySQL account with the same user name as the operating system user name and define it to authenticate using the PAM plugin and the `mysql-unix` PAM service:

```
CREATE USER 'antonio'@'localhost'
  IDENTIFIED WITH authentication_pam
  AS 'mysql-unix';
GRANT ALL PRIVILEGES
  ON mydb.*
  TO 'antonio'@'localhost';
```

   Here, the authentication string contains only the PAM service name, `mysql-unix`, which authenticates Unix passwords.

4. Use the `mysql` command-line client to connect to the MySQL server as `antonio`. For example:

```
shell> mysql --user=antonio --password --enable-cleartext-plugin
Enter password: antonio_password
```

   The server should permit the connection and the following query should return output as shown:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-------------------+-------------------+--------------+
| USER()            | CURRENT_USER()    | @@proxy_user |
+-------------------+-------------------+--------------+
| antonio@localhost | antonio@localhost | NULL         |
+-------------------+-------------------+--------------+
```

   This demonstrates that the `antonio` operating system user is authenticated to have the privileges granted to the `antonio` MySQL user, and that no proxying has occurred.

Note

The client-side `mysql_clear_password` authentication plugin leaves the password untouched, so client programs send it to the MySQL server as cleartext. This enables the password to be passed as is to PAM. A cleartext password is necessary to use the server-side PAM library, but may be a security problem in some configurations. These measures minimize the risk:

- To make inadvertent use of the `mysql_clear_password` plugin less likely, MySQL clients must explicitly enable it (for example, with the `--enable-cleartext-plugin` option). See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.
- To avoid password exposure with the `mysql_clear_password` plugin enabled, MySQL clients should connect to the MySQL server using an encrypted connection. See Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

PAM LDAP Authentication without Proxy Users

This authentication scenario uses PAM to check external users defined in terms of operating system user names and LDAP passwords, without proxying. Every such external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use PAM authentication through LDAP.

To use PAM LDAP pluggable authentication for MySQL, these prerequisites must be satisfied:

- An LDAP server must be available for the PAM LDAP service to communicate with.
- LDAP users to be authenticated by MySQL must be present in the directory managed by the LDAP server.

Note

Another way to use LDAP for MySQL user authentication is to use the LDAP-specific authentication plugins. See Section 6.4.1.7, “LDAP Pluggable Authentication”.

Configure MySQL for PAM LDAP authentication as follows:

1. Verify that Unix authentication permits logins to the operating system with the user name `antonio` and password *`antonio_password`*.
2. Set up PAM to authenticate MySQL connections using LDAP by creating a `mysql-ldap` PAM service file named `/etc/pam.d/mysql-ldap`. The file contents are system dependent, so check existing login-related files in the `/etc/pam.d` directory to see what they look like. On Linux, the `mysql-ldap` file might look like this:

   ```
   #%PAM-1.0
   auth        required    pam_ldap.so
   account     required    pam_ldap.so
   ```

   If PAM object files have a suffix different from `.so` on your system, substitute the correct suffix.

   The PAM file format might differ on some systems.
3. Create a MySQL account with the same user name as the operating system user name and define it to authenticate using the PAM plugin and the `mysql-ldap` PAM service:

   ```
   CREATE USER 'antonio'@'localhost'
     IDENTIFIED WITH authentication_pam
     AS 'mysql-ldap';
   GRANT ALL PRIVILEGES
     ON mydb.*
     TO 'antonio'@'localhost';
   ```

   Here, the authentication string contains only the PAM service name, `mysql-ldap`, which authenticates using LDAP.
4. Connecting to the server is the same as described in PAM Unix Password Authentication without Proxy Users.

PAM Unix Password Authentication with Proxy Users and Group Mapping

The authentication scheme described here uses proxying and PAM group mapping to map connecting MySQL users who authenticate using PAM onto other MySQL accounts that define different sets of privileges. Users do not connect directly through the accounts that define the privileges. Instead, they connect through a default proxy account authenticated using PAM, such that all the external users are

mapped to the MySQL accounts that hold the privileges. Any user who connects using the proxy account is mapped to one of those MySQL accounts, the privileges for which determine the database operations permitted to the external user.

The procedure shown here uses Unix password authentication. To use LDAP instead, see the early steps of PAM LDAP Authentication without Proxy Users.

Note

Traditional Unix passwords are checked using the `/etc/shadow` file. For information regarding possible issues related to this file, see PAM Authentication Access to Unix Password Store.

1. Verify that Unix authentication permits logins to the operating system with the user name `antonio` and password *`antonio_password`*.

2. Verify that `antonio` is a member of the `root` or `users` PAM group.

3. Set up PAM to authenticate the `mysql-unix` PAM service through operating system users by creating a file named `/etc/pam.d/mysql-unix`. The file contents are system dependent, so check existing login-related files in the `/etc/pam.d` directory to see what they look like. On Linux, the `mysql-unix` file might look like this:

   ```
   #%PAM-1.0
   auth            include         password-auth
   account         include         password-auth
   ```

   For macOS, use `login` rather than `password-auth`.

   The PAM file format might differ on some systems. For example, on Ubuntu and other Debian-based systems, use these file contents instead:

   ```
   @include common-auth
   @include common-account
   @include common-session-noninteractive
   ```

4. Create a default proxy user (`''@''`) that maps external PAM users to the proxied accounts:

   ```
   CREATE USER ''@''
     IDENTIFIED WITH authentication_pam
     AS 'mysql-unix, root=developer, users=data_entry';
   ```

   Here, the authentication string contains the PAM service name, `mysql-unix`, which authenticates Unix passwords. The authentication string also maps external users in the `root` and `users` PAM groups to the `developer` and `data_entry` MySQL user names, respectively.

   The PAM group mapping list following the PAM service name is required when you set up proxy users. Otherwise, the plugin cannot tell how to perform mapping from external user names to the proper proxied MySQL user names.

   Note

   If your MySQL installation has anonymous users, they might conflict with the default proxy user. For more information about this issue, and ways of dealing with it, see Default Proxy User and Anonymous User Conflicts.

5. Create the proxied accounts and grant to them the privileges required for MySQL access:

```
CREATE USER 'developer'@'localhost'
  IDENTIFIED WITH mysql_no_login;
CREATE USER 'data_entry'@'localhost'
  IDENTIFIED WITH mysql_no_login;

GRANT ALL PRIVILEGES
  ON mydevdb.*
  TO 'developer'@'localhost';
GRANT ALL PRIVILEGES
  ON mydb.*
  TO 'data_entry'@'localhost';
```

The proxied accounts use the `mysql_no_login` authentication plugin to prevent clients from using the accounts to log in directly to the MySQL server. Instead, it is expected that users who authenticate using PAM will use the `developer` or `data_entry` account by proxy based on their PAM group. (This assumes that the plugin is installed. For instructions, see Section 6.4.1.8, “No-Login Pluggable Authentication”.) For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

6. Grant to the proxy account the `PROXY` privilege for each proxied account:

```
GRANT PROXY
  ON 'developer'@'localhost'
  TO ''@'';
GRANT PROXY
  ON 'data_entry'@'localhost'
  TO ''@'';
```

7. Use the `mysql` command-line client to connect to the MySQL server as `antonio`.

```
shell> mysql --user=antonio --password --enable-cleartext-plugin
Enter password: antonio_password
```

The server authenticates the connection using the default `''@''` proxy account. The resulting privileges for `antonio` depend on which PAM groups `antonio` is a member of. If `antonio` is a member of the `root` PAM group, the PAM plugin maps `root` to the `developer` MySQL user name and returns that name to the server. The server verifies that `''@''` has the `PROXY` privilege for `developer` and permits the connection. The following query should return output as shown:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-------------------+--------------------+--------------+
| USER()            | CURRENT_USER()     | @@proxy_user |
+-------------------+--------------------+--------------+
| antonio@localhost | developer@localhost | ''@''        |
+-------------------+--------------------+--------------+
```

This demonstrates that the `antonio` operating system user is authenticated to have the privileges granted to the `developer` MySQL user, and that proxying occurred through the default proxy account.

If `antonio` is not a member of the `root` PAM group but is a member of the `users` PAM group, a similar process occurs, but the plugin maps `user` PAM group membership to the `data_entry` MySQL user name and returns that name to the server:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-------------------+---------------------+--------------+
| USER()            | CURRENT_USER()      | @@proxy_user |
+-------------------+---------------------+--------------+
```

```
| antonio@localhost | data_entry@localhost | ''@''        |
+-------------------+----------------------+--------------+
```

This demonstrates that the `antonio` operating system user is authenticated to have the privileges of the `data_entry` MySQL user, and that proxying occurred through the default proxy account.

Note

The client-side `mysql_clear_password` authentication plugin leaves the password untouched, so client programs send it to the MySQL server as cleartext. This enables the password to be passed as is to PAM. A cleartext password is necessary to use the server-side PAM library, but may be a security problem in some configurations. These measures minimize the risk:

- To make inadvertent use of the `mysql_clear_password` plugin less likely, MySQL clients must explicitly enable it (for example, with the `--enable-cleartext-plugin` option). See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.
- To avoid password exposure with the `mysql_clear_password` plugin enabled, MySQL clients should connect to the MySQL server using an encrypted connection. See Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

PAM Authentication Access to Unix Password Store

On some systems, Unix authentication uses a password store such as `/etc/shadow`, a file that typically has restricted access permissions. This can cause MySQL PAM-based authentication to fail. Unfortunately, the PAM implementation does not permit distinguishing “password could not be checked” (due, for example, to inability to read `/etc/shadow`) from “password does not match.” If you are using Unix password store for PAM authentication, you may be able to enable access to it from MySQL using one of the following methods:

- Assuming that the MySQL server is run from the `mysql` operating system account, put that account in the `shadow` group that has `/etc/shadow` access:
  1. Create a `shadow` group in `/etc/group`.
  2. Add the `mysql` operating system user to the `shadow` group in `/etc/group`.
  3. Assign `/etc/group` to the `shadow` group and enable the group read permission:

     ```
     chgrp shadow /etc/shadow
     chmod g+r /etc/shadow
     ```

  4. Restart the MySQL server.
- If you are using the `pam_unix` module and the `unix_chkpwd` utility, enable password store access as follows:

  ```
  chmod u-s /usr/sbin/unix_chkpwd
  setcap cap_dac_read_search+ep /usr/sbin/unix_chkpwd
  ```

  Adjust the path to `unix_chkpwd` as necessary for your platform.

PAM Authentication Debugging

The PAM authentication plugin checks at initialization time whether the `AUTHENTICATION_PAM_LOG` environment value is set (the value does not matter). If so, the plugin enables logging of diagnostic messages to the standard output. These messages may be helpful for debugging PAM-related issues that occur when the plugin performs authentication.

Some messages include reference to PAM plugin source files and line numbers, which enables plugin actions to be tied more closely to the location in the code where they occur.

Another technique for debugging connection failures and determining what is happening during connection attempts is to configure PAM authentication to permit all connections, then check the system log files. This technique should be used only on a *temporary* basis, and not on a production server.

Configure a PAM service file named `/etc/pam.d/mysql-any-password` with these contents (the format may differ on some systems):

```
#%PAM-1.0
auth        required    pam_permit.so
account     required    pam_permit.so
```

Create an account that uses the PAM plugin and names the `mysql-any-password` PAM service:

```
CREATE USER 'testuser'@'localhost'
  IDENTIFIED WITH authentication_pam
  AS 'mysql-any-password';
```

The `mysql-any-password` service file causes any authentication attempt to return true, even for incorrect passwords. If an authentication attempt fails, that tells you the configuration problem is on the MySQL side. Otherwise, the problem is on the operating system/PAM side. To see what might be happening, check system log files such as `/var/log/secure`, `/var/log/audit.log`, `/var/log/syslog`, or `/var/log/messages`.

After determining what the problem is, remove the `mysql-any-password` PAM service file to disable any-password access.

6.4.1.6 Windows Pluggable Authentication

Note

Windows pluggable authentication is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition for Windows supports an authentication method that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password.

The client and server exchange data packets in the authentication handshake. As a result of this exchange, the server creates a security context object that represents the identity of the client in the Windows OS. This identity includes the name of the client account. Windows pluggable authentication uses the identity of the client to check whether it is a given account or a member of a group. By default, negotiation uses Kerberos to authenticate, then NTLM if Kerberos is unavailable.

Windows pluggable authentication provides these capabilities:

- External authentication: Windows authentication enables MySQL Server to accept connections from users defined outside the MySQL grant tables who have logged in to Windows.

- Proxy user support: Windows authentication can return to MySQL a user name different from the external user name passed by the client program. This means that the plugin can return the MySQL user that defines the privileges the external Windows-authenticated user should have. For example, a Windows user named `joe` can connect and have the privileges of a MySQL user named `developer`.

The following table shows the plugin and library file names. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.17 Plugin and Library Names for Windows Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`authentication_windows`
Client-side plugin	`authentication_windows_client`
Library file	`authentication_windows.dll`

The library file includes only the server-side plugin. The client-side plugin is built into the `libmysqlclient` client library.

The server-side Windows authentication plugin is included only in MySQL Enterprise Edition. It is not included in MySQL community distributions. The client-side plugin is included in all distributions, including community distributions. This permits clients from any distribution to connect to a server that has the server-side plugin loaded.

The Windows authentication plugin is supported on any version of Windows supported by MySQL 8.0 (see https://www.mysql.com/support/supportedplatforms/database.html).

The following sections provide installation and usage information specific to Windows pluggable authentication:

- Installing Windows Pluggable Authentication
- Uninstalling Windows Pluggable Authentication
- Using Windows Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”. For proxy user information, see Section 6.2.18, “Proxy Users”.

Installing Windows Pluggable Authentication

This section describes how to install the Windows authentication plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file:

```
[mysqld]
plugin-load-add=authentication_windows.dll
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement:

```
INSTALL PLUGIN authentication_windows SONAME 'authentication_windows.dll';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%windows%';
+-----------------------+---------------+
| PLUGIN_NAME           | PLUGIN_STATUS |
+-----------------------+---------------+
| authentication_windows | ACTIVE        |
+-----------------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the Windows authentication plugin, see Using Windows Pluggable Authentication. Additional plugin control is provided by the `authentication_windows_use_principal_name` and `authentication_windows_log_level` system variables. See Section 5.1.8, “Server System Variables”.

Uninstalling Windows Pluggable Authentication

The method used to uninstall the Windows authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

  ```
  UNINSTALL PLUGIN authentication_windows;
  ```

In addition, remove any startup options that set Windows plugin-related system variables.

Using Windows Pluggable Authentication

The Windows authentication plugin supports the use of MySQL accounts such that users who have logged in to Windows can connect to the MySQL server without having to specify an additional password. It is assumed that the server is running with the server-side plugin enabled, as described in Installing Windows Pluggable Authentication. Once the DBA has enabled the server-side plugin and set up accounts to use it, clients can connect using those accounts with no other setup required on their part.

To refer to the Windows authentication plugin in the `IDENTIFIED WITH` clause of a `CREATE USER` statement, use the name `authentication_windows`. Suppose that the Windows users `Rafal` and `Tasha` should be permitted to connect to MySQL, as well as any users in the `Administrators` or `Power Users` group. To set this up, create a MySQL account named `sql_admin` that uses the Windows plugin for authentication:

```
CREATE USER sql_admin
```

```
  IDENTIFIED WITH authentication_windows
  AS 'Rafal, Tasha, Administrators, "Power Users"';
```

The plugin name is `authentication_windows`. The string following the `AS` keyword is the authentication string. It specifies that the Windows users named `Rafal` or `Tasha` are permitted to authenticate to the server as the MySQL user `sql_admin`, as are any Windows users in the `Administrators` or `Power Users` group. The latter group name contains a space, so it must be quoted with double quote characters.

After you create the `sql_admin` account, a user who has logged in to Windows can attempt to connect to the server using that account:

```
C:\> mysql --user=sql_admin
```

No password is required here. The `authentication_windows` plugin uses the Windows security API to check which Windows user is connecting. If that user is named `Rafal` or `Tasha`, or is a member of the `Administrators` or `Power Users` group, the server grants access and the client is authenticated as `sql_admin` and has whatever privileges are granted to the `sql_admin` account. Otherwise, the server denies access.

Authentication string syntax for the Windows authentication plugin follows these rules:

- The string consists of one or more user mappings separated by commas.
- Each user mapping associates a Windows user or group name with a MySQL user name:

  ```
  win_user_or_group_name=mysql_user_name
  win_user_or_group_name
  ```

  For the latter syntax, with no *`mysql_user_name`* value given, the implicit value is the MySQL user created by the `CREATE USER` statement. Thus, these statements are equivalent:

  ```
  CREATE USER sql_admin
    IDENTIFIED WITH authentication_windows
    AS 'Rafal, Tasha, Administrators, "Power Users"';

  CREATE USER sql_admin
    IDENTIFIED WITH authentication_windows
    AS 'Rafal=sql_admin, Tasha=sql_admin, Administrators=sql_admin,
        "Power Users"=sql_admin';
  ```

- Each backslash character (`\`) in a value must be doubled because backslash is the escape character in MySQL strings.
- Leading and trailing spaces not inside double quotation marks are ignored.
- Unquoted *`win_user_or_group_name`* and *`mysql_user_name`* values can contain anything except equal sign, comma, or space.
- If a *`win_user_or_group_name`* and or *`mysql_user_name`* value is quoted with double quotation marks, everything between the quotation marks is part of the value. This is necessary, for example, if the name contains space characters. All characters within double quotes are legal except double quotation mark and backslash. To include either character, escape it with a backslash.
- *`win_user_or_group_name`* values use conventional syntax for Windows principals, either local or in a domain. Examples (note the doubling of backslashes):

  ```
  ```

```
domain\\user
.\\user
domain\\group
.\\group
BUILTIN\\WellKnownGroup
```

When invoked by the server to authenticate a client, the plugin scans the authentication string left to right for a user or group match to the Windows user. If there is a match, the plugin returns the corresponding *`mysql_user_name`* to the MySQL server. If there is no match, authentication fails.

A user name match takes preference over a group name match. Suppose that the Windows user named `win_user` is a member of `win_group` and the authentication string looks like this:

```
'win_group = sql_user1, win_user = sql_user2'
```

When `win_user` connects to the MySQL server, there is a match both to `win_group` and to `win_user`. The plugin authenticates the user as `sql_user2` because the more-specific user match takes precedence over the group match, even though the group is listed first in the authentication string.

Windows authentication always works for connections from the same computer on which the server is running. For cross-computer connections, both computers must be registered with Windows Active Directory. If they are in the same Windows domain, it is unnecessary to specify a domain name. It is also possible to permit connections from a different domain, as in this example:

```
CREATE USER sql_accounting
  IDENTIFIED WITH authentication_windows
  AS 'SomeDomain\\Accounting';
```

Here `SomeDomain` is the name of the other domain. The backslash character is doubled because it is the MySQL escape character within strings.

MySQL supports the concept of proxy users whereby a client can connect and authenticate to the MySQL server using one account but while connected has the privileges of another account (see Section 6.2.18, “Proxy Users”). Suppose that you want Windows users to connect using a single user name but be mapped based on their Windows user and group names onto specific MySQL accounts as follows:

- The `local_user` and `MyDomain\domain_user` local and domain Windows users should map to the `local_wlad` MySQL account.
- Users in the `MyDomain\Developers` domain group should map to the `local_dev` MySQL account.
- Local machine administrators should map to the `local_admin` MySQL account.

To set this up, create a proxy account for Windows users to connect to, and configure this account so that users and groups map to the appropriate MySQL accounts (`local_wlad`, `local_dev`, `local_admin`). In addition, grant the MySQL accounts the privileges appropriate to the operations they need to perform. The following instructions use `win_proxy` as the proxy account, and `local_wlad`, `local_dev`, and `local_admin` as the proxied accounts.

1. Create the proxy MySQL account:

```
CREATE USER win_proxy
  IDENTIFIED WITH  authentication_windows
  AS 'local_user = local_wlad,
      MyDomain\\domain_user = local_wlad,
      MyDomain\\Developers = local_dev,
      BUILTIN\\Administrators = local_admin';
```

2. For proxying to work, the proxied accounts must exist, so create them:

```
CREATE USER local_wlad
  IDENTIFIED WITH mysql_no_login;
CREATE USER local_dev
  IDENTIFIED WITH mysql_no_login;
CREATE USER local_admin
  IDENTIFIED WITH mysql_no_login;
```

   The proxied accounts use the `mysql_no_login` authentication plugin to prevent clients from using the accounts to log in directly to the MySQL server. Instead, it is expected that users who authenticate using Windows will use the `win_proxy` proxy account. (This assumes that the plugin is installed. For instructions, see Section 6.4.1.8, “No-Login Pluggable Authentication”.) For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

   You should also execute `GRANT` statements (not shown) that grant each proxied account the privileges required for MySQL access.

3. Grant to the proxy account the `PROXY` privilege for each proxied account:

```
GRANT PROXY ON local_wlad TO win_proxy;
GRANT PROXY ON local_dev TO win_proxy;
GRANT PROXY ON local_admin TO win_proxy;
```

Now the Windows users `local_user` and `MyDomain\domain_user` can connect to the MySQL server as `win_proxy` and when authenticated have the privileges of the account given in the authentication string (in this case, `local_wlad`). A user in the `MyDomain\Developers` group who connects as `win_proxy` has the privileges of the `local_dev` account. A user in the `BUILTIN\Administrators` group has the privileges of the `local_admin` account.

To configure authentication so that all Windows users who do not have their own MySQL account go through a proxy account, substitute the default proxy account (`''@''`) for `win_proxy` in the preceding instructions. For information about default proxy accounts, see Section 6.2.18, “Proxy Users”.

Note

If your MySQL installation has anonymous users, they might conflict with the default proxy user. For more information about this issue, and ways of dealing with it, see Default Proxy User and Anonymous User Conflicts.

To use the Windows authentication plugin with Connector/NET connection strings in Connector/NET 6.4.4 and higher, see Using the Windows Native Authentication Plugin.

6.4.1.7 LDAP Pluggable Authentication

Note

LDAP pluggable authentication is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition supports an authentication method that enables MySQL Server to use LDAP (Lightweight Directory Access Protocol) to authenticate MySQL users by accessing directory services such as X.500. MySQL uses LDAP to fetch user, credential, and group information.

LDAP pluggable authentication provides these capabilities:

- External authentication: LDAP authentication enables MySQL Server to accept connections from users defined outside the MySQL grant tables in LDAP directories.
- Proxy user support: LDAP authentication can return to MySQL a user name different from the external user name passed by the client program, based on the LDAP groups the external user is a member of. This means that an LDAP plugin can return the MySQL user that defines the privileges the external LDAP-authenticated user should have. For example, an LDAP user named `joe` can connect and have the privileges of a MySQL user named `developer`, if the LDAP group for `joe` is `developer`.
- Security: Using TLS, connections to the LDAP server can be secure.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The files must be located in the directory named by the `plugin_dir` system variable.

Table 6.18 Plugin and Library Names for LDAP Authentication

Plugin or File	Plugin or File Name
Server-side plugin names	`authentication_ldap_sasl`, `authentication_ldap_simple`
Client-side plugin names	`authentication_ldap_sasl_client`, `mysql_clear_password`
Library file names	`authentication_ldap_sasl.so`, `authentication_ldap_sasl_client.so`, `authentication_ldap_simple.so`

The library files include only the `authentication_ldap_`*`XXX`* authentication plugins. The client-side `mysql_clear_password` plugin is built into the `libmysqlclient` client library.

The two server-side LDAP plugins each work with a specific client-side plugin:

- The server-side `authentication_ldap_simple` plugin performs simple LDAP authentication. For connections by accounts that use this plugin, client programs use the client-side `mysql_clear_password` plugin, which sends the password to the server as cleartext. No password hashing or encryption is used, so a secure connection between the MySQL client and server is recommended to prevent password exposure.
- The server-side `authentication_ldap_sasl` plugin performs SASL-based LDAP authentication. For connections by accounts that use this plugin, client programs use the client-side `authentication_ldap_sasl_client` plugin. The client-side and server-side SASL LDAP plugins use SASL messages for secure transmission of credentials within the LDAP protocol, to avoid sending the cleartext password between the MySQL client and server.

The following sections provide installation and usage information specific to LDAP pluggable authentication:

- Prerequisites for LDAP Pluggable Authentication
- How LDAP Authentication of MySQL Users Works
- Installing LDAP Pluggable Authentication
- Uninstalling LDAP Pluggable Authentication
- Using LDAP Pluggable Authentication
- Simple LDAP Authentication
- SASL-Based LDAP Authentication

- LDAP Authentication User DN Suffixes
- LDAP Authentication with Proxying
- LDAP Authentication Group Preference and Mapping Specification

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”. For information about the `mysql_clear_password` plugin, see Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”. For proxy user information, see Section 6.2.18, “Proxy Users”.

Note

If your system supports PAM and permits LDAP as a PAM authentication method, another way to use LDAP for MySQL user authentication is to use the server-side `authentication_pam` plugin. See Section 6.4.1.5, “PAM Pluggable Authentication”.

Prerequisites for LDAP Pluggable Authentication

To use LDAP pluggable authentication for MySQL, these prerequisites must be satisfied:

- An LDAP server must be available for the LDAP authentication plugins to communicate with.
- LDAP users to be authenticated by MySQL must be present in the directory managed by the LDAP server.
- An LDAP client library must be available on systems where the server-side `authentication_ldap_sasl` or `authentication_ldap_simple` plugin is used. Currently, supported libraries are the Windows native LDAP library, or the OpenLDAP library on non-Windows systems.
- To use SASL-based LDAP authentication:
  - The LDAP server must be configured to communicate with a SASL server.
  - A SASL client library must be is available on systems where the client-side `authentication_ldap_sasl_client` plugin is used. Currently, the only supported library is the Cyrus SASL library.

How LDAP Authentication of MySQL Users Works

This section provides a general overview of how MySQL and LDAP work together to authenticate MySQL users. For examples showing how to set up MySQL accounts to use specific LDAP authentication plugins, see Using LDAP Pluggable Authentication.

The client connects to the MySQL server, providing the MySQL client user name and the LDAP password:

- For simple LDAP authentication, the client-side and server-side plugins communicate the password as cleartext.
- For SASL-based LDAP authentication, the client-side and server-side plugins use SASL messages for secure transmission of credentials within the LDAP protocol, to avoid sending the cleartext password between the MySQL client and server.

If the client user name and host name match no MySQL account, the connection is rejected.

If there is a matching MySQL account, authentication against LDAP occurs. The LDAP server looks for an entry matching the user and authenticates the entry against the LDAP password:

- If the MySQL account names an the LDAP user distinguished name (DN), LDAP authentication uses that value and the LDAP password provided by the client. (To associate an LDAP user DN with a MySQL account, include a `BY` clause that specifies an authentication string in the `CREATE USER` statement that creates the account.)
- If the MySQL account names no LDAP user DN, LDAP authentication uses the user name and LDAP password provided by the client. In this case, the authentication plugin first binds to the LDAP server using the root DN and password as credentials to find the user DN based on the client user name, then authenticates that user DN against the LDAP password. This bind using the root credentials fails if the root DN and password are set to incorrect values, or are empty (not set) and the LDAP server does not permit anonymous connections.

If the LDAP server finds no match or multiple matches, authentication fails and the client connection is rejected.

If the LDAP server finds a single match, LDAP authentication succeeds (assuming that the password is correct), the LDAP server returns the LDAP entry, and the authentication plugin determines the name of the authenticated user based on that entry:

- If the LDAP entry has a group attribute (by default, the `cn` attribute), the plugin returns its value as the authenticated user name.
- If the LDAP entry has no group attribute, the authentication plugin returns the client user name as the authenticated user name.

The MySQL server compares the client user name with the authenticated user name to determine whether proxying occurs for the client session:

- If the names are the same, no proxying occurs: The MySQL account matching the client user name is used for privilege checking.
- If the names differ, proxying occurs: MySQL looks for an account matching the authenticated user name. That account becomes the proxied user, which is used for privilege checking. The MySQL account that matched the client user name is treated as the external proxy user.

Installing LDAP Pluggable Authentication

This section describes how to install the LDAP authentication plugins. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library files must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The server-side plugin library file base names are `authentication_ldap_sasl` and `authentication_ldap_simple`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugins at server startup, use `--plugin-load-add` options to name the library files that contain them. With this plugin-loading method, the options must be given each time the server starts. Also, specify values for any plugin-provided system variables you wish to configure.

Each server-side LDAP plugin exposes a set of system variables that enable its operation to be configured. Setting most of these is optional, but you must set the variables that specify the LDAP server host (so the plugin knows where to connect) and base distinguished name for LDAP bind operations (to limit the scope of searches and obtain faster searches). For details about all LDAP system variables, see Section 6.4.1.11, “Pluggable Authentication System Variables”.

To load the plugins and set the LDAP server host and base distinguished name for LDAP bind operations, put lines such as these in your `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=authentication_ldap_sasl.so
authentication_ldap_sasl_server_host=127.0.0.1
authentication_ldap_sasl_bind_base_dn="dc=example,dc=com"
plugin-load-add=authentication_ldap_simple.so
authentication_ldap_simple_server_host=127.0.0.1
authentication_ldap_simple_bind_base_dn="dc=example,dc=com"
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugins at runtime, use these statements (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN authentication_ldap_sasl
  SONAME 'authentication_ldap_sasl.so';
INSTALL PLUGIN authentication_ldap_simple
  SONAME 'authentication_ldap_simple.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

After installing the plugins at runtime, their system variables become available and you can add settings for them to your `my.cnf` file to configure the plugins for subsequent restarts. For example:

```
[mysqld]
authentication_ldap_sasl_server_host=127.0.0.1
authentication_ldap_sasl_bind_base_dn="dc=example,dc=com"
authentication_ldap_simple_server_host=127.0.0.1
authentication_ldap_simple_bind_base_dn="dc=example,dc=com"
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to set and persist the values at runtime, use these statements:

```
SET PERSIST authentication_ldap_sasl_server_host='127.0.0.1';
SET PERSIST authentication_ldap_sasl_bind_base_dn='dc=example,dc=com';
SET PERSIST authentication_ldap_simple_server_host='127.0.0.1';
SET PERSIST authentication_ldap_simple_bind_base_dn='dc=example,dc=com';
```

`SET PERSIST` sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. To change a value for the running MySQL instance without saving it for subsequent restarts, use the `GLOBAL` keyword rather than `PERSIST`. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%ldap%';
+----------------------------+---------------+
| PLUGIN_NAME                | PLUGIN_STATUS |
+----------------------------+---------------+
```

```
| authentication_ldap_sasl   | ACTIVE        |
| authentication_ldap_simple | ACTIVE        |
+----------------------------+---------------+
```

If a plugin failed to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with an LDAP plugin, see Using LDAP Pluggable Authentication.

Additional Notes for SELinux

On systems running EL6 or EL that have SELinux enabled, changes to the SELinux policy are required to enable the MySQL LDAP plugins to communicate with the LDAP service:

1. Create a file `mysqlldap.te` with these contents:

```
module mysqlldap 1.0;

require {
        type ldap_port_t;
        type mysqld_t;
        class tcp_socket name_connect;
}

#============= mysqld_t ==============

allow mysqld_t ldap_port_t:tcp_socket name_connect;
```

2. Compile the security policy module into a binary representation:

```
checkmodule -M -m mysqlldap.te -o mysqlldap.mod
```

3. Create an SELinux policy module package:

```
semodule_package -m mysqlldap.mod  -o mysqlldap.pp
```

4. Install the module package:

```
semodule -i mysqlldap.pp
```

5. When the SELinux policy changes has been made, restart the MySQL server:

```
service mysqld restart
```

Uninstalling LDAP Pluggable Authentication

The method used to uninstall the LDAP authentication plugins depends on how you installed them:

- If you installed the plugins at server startup using `--plugin-load-add` options, restart the server without those options.
- If you installed the plugins at runtime using `INSTALL PLUGIN`, they remain installed across server restarts. To uninstall them, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN authentication_ldap_sasl;
UNINSTALL PLUGIN authentication_ldap_simple;
```

In addition, remove from your `my.cnf` file any startup options that set LDAP plugin-related system variables. If you used `SET PERSIST` to persist LDAP system variables, use `RESET PERSIST` to remove the settings.

Using LDAP Pluggable Authentication

This section describes how to enable MySQL accounts to connect to the MySQL server using LDAP pluggable authentication. It is assumed that the server is running with the appropriate server-side plugins enabled, as described in Installing LDAP Pluggable Authentication, and that the appropriate client-side plugins are available on the client host.

This section does not describe LDAP configuration or administration. It is assumed that you are familiar with those topics.

The two server-side LDAP plugins each work with a specific client-side plugin:

- The server-side `authentication_ldap_simple` plugin performs simple LDAP authentication. For connections by accounts that use this plugin, client programs use the client-side `mysql_clear_password` plugin, which sends the password to the server as cleartext. No password hashing or encryption is used, so a secure connection between the MySQL client and server is recommended to prevent password exposure.
- The server-side `authentication_ldap_sasl` plugin performs SASL-based LDAP authentication. For connections by accounts that use this plugin, client programs use the client-side `authentication_ldap_sasl_client` plugin. The client-side and server-side SASL LDAP plugins use SASL messages for secure transmission of credentials within the LDAP protocol, to avoid sending the cleartext password between the MySQL client and server.

Overall requirements for LDAP authentication of MySQL users:

- There must be an LDAP directory entry for each user to be authenticated.
- There must be a MySQL user account that specifies a server-side LDAP authentication plugin and optionally names the associated LDAP user distinguished name (DN). (To associate an LDAP user DN with a MySQL account, include a `BY` clause in the `CREATE USER` statement that creates the account.) If an account names no LDAP string, LDAP authentication uses the user name specified by the client to find the LDAP entry.
- Client programs connect using the connection method appropriate for the server-side authentication plugin the MySQL account uses. For LDAP authentication, connections require the MySQL user name and LDAP password. In addition, for accounts that use the server-side `authentication_ldap_simple` plugin, invoke client programs with the `--enable-cleartext-plugin` option to enable the client-side `mysql_clear_password` plugin.

The instructions here assume the following scenario:

- MySQL users `betsy` and `boris` authenticate to the LDAP entries for `betsy_ldap` and `boris_ldap`, respectively. (It is not necessary that the MySQL and LDAP user names differ. The use of different names in this discussion helps clarify whether an operation context is MySQL or LDAP.)
- LDAP entries use the `uid` attribute to specify user names. This may vary depending on LDAP server. Some LDAP servers use the `cn` attribute for user names rather than `uid`. To change the attribute, modify the `authentication_ldap_simple_user_search_attr` or `authentication_ldap_sasl_user_search_attr` system variable appropriately.
- These LDAP entries are available in the directory managed by the LDAP server, to provide distinguished name values that uniquely identify each user:

```
uid=betsy_ldap,ou=People,dc=example,dc=com
uid=boris_ldap,ou=People,dc=example,dc=com
```

- `CREATE USER` statements that create MySQL accounts name an LDAP user in the `BY` clause, to indicate which LDAP entry the MySQL account authenticates against.

The instructions for setting up an account that uses LDAP authentication depend on which server-side LDAP plugin is used. The following sections describe several usage scenarios.

Simple LDAP Authentication

To configure a MySQL account for simple LDAP authentication, the `CREATE USER` statement specifies the `authentication_ldap_simple` plugin, and optionally names the LDAP user distinguished name (DN):

```
CREATE USER user
  IDENTIFIED WITH authentication_ldap_simple
  [BY 'LDAP user DN'];
```

Suppose that a MySQL user `betsy` has this entry in the LDAP directory:

```
uid=betsy_ldap,ou=People,dc=example,dc=com
```

Then the statement to create the MySQL account for `betsy` looks like this:

```
CREATE USER 'betsy'@'localhost'
  IDENTIFIED WITH authentication_ldap_simple
  AS 'uid=betsy_ldap,ou=People,dc=example,dc=com';
```

The authentication string specified in the `BY` clause does not include the LDAP password. That must be provided by the client user at connect time.

Clients connect to the MySQL server by providing the MySQL user name and LDAP password, and by enabling the client-side `mysql_clear_password` plugin:

```
shell> mysql --user=betsy --password --enable-cleartext-plugin
Enter password: betsy_password (betsy_ldap LDAP password)
```

Note

The client-side `mysql_clear_password` authentication plugin leaves the password untouched, so client programs send it to the MySQL server as cleartext. This enables the password to be passed as is to the LDAP server. A cleartext password is necessary to use the server-side LDAP library without SASL, but may be a security problem in some configurations. These measures minimize the risk:

- To make inadvertent use of the `mysql_clear_password` plugin less likely, MySQL clients must explicitly enable it (for example, with the `--enable-cleartext-plugin` option). See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.
- To avoid password exposure with the `mysql_clear_password` plugin enabled, MySQL clients should connect to the MySQL server using an encrypted connection. See Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

The authentication process occurs as follows:

1. The client-side plugin sends `betsy` and *`betsy_password`* as the client user name and LDAP password to the MySQL server.
2. The connection attempt matches the `'betsy'@'localhost'` account. The server-side LDAP plugin finds that this account has an authentication string of `'uid=betsy_ldap,ou=People,dc=example,dc=com'` to name the LDAP user DN. The plugin sends this string and the LDAP password to the LDAP server.
3. The LDAP server finds the LDAP entry for `betsy_ldap` and the password matches, so LDAP authentication succeeds.
4. The LDAP entry has no group attribute, so the server-side plugin returns the client user name (`betsy`) as the authenticated user. This is the same user name supplied by the client, so no proxying occurs and the client session uses the `'betsy'@'localhost'` account for privilege checking.

Had the matching LDAP entry contained a group attribute, that attribute value would have been the authenticated user name and, if the value differed from `betsy`, proxying would have occurred. For examples that use the group attribute, see LDAP Authentication with Proxying.

Had the `CREATE USER` statement contained no `BY` clause to specify the `betsy_ldap` LDAP distinguished name, authentication attempts would use the user name provided by the client (in this case, `betsy`). In the absence of an LDAP entry for `betsy`, authentication would fail.

SASL-Based LDAP Authentication

To configure a MySQL account for SASL LDAP authentication, the `CREATE USER` statement specifies the `authentication_ldap_sasl` plugin, and optionally names the LDAP user distinguished name (DN):

```
CREATE USER user
  IDENTIFIED WITH authentication_ldap_sasl
  [BY 'LDAP user DN'];
```

Suppose that a MySQL user `boris` has this entry in the LDAP directory:

```
uid=boris_ldap,ou=People,dc=example,dc=com
```

Then the statement to create the MySQL account for `boris` looks like this:

```
CREATE USER 'boris'@'localhost'
  IDENTIFIED WITH authentication_ldap_sasl
  AS 'uid=boris_ldap,ou=People,dc=example,dc=com';
```

The authentication string specified in the `BY` clause does not include the LDAP password. That must be provided by the client user at connect time.

Clients connect to the MySQL server by providing the MySQL user name and LDAP password:

```
shell> mysql --user=boris --password
Enter password: boris_password (boris_ldap LDAP password)
```

For the server-side `authentication_ldap_sasl` plugin, clients use the client-side `authentication_ldap_sasl_client` plugin. If a client program does not find the client-side plugin, specify a `--plugin-dir` option that names the directory where the plugin library file is installed.

The authentication process for `boris` is similar to that previously described for `betsy` with simple LDAP authentication, except that the client-side and server-side SASL LDAP plugins use SASL messages for secure transmission of credentials within the LDAP protocol, to avoid sending the cleartext password between the MySQL client and server.

LDAP Authentication User DN Suffixes

LDAP authentication plugins permit the authentication string that provides user DN information to begin with a `+` prefix character:

- In the absence of a `+` character, the authentication string value is treated as is without modification.
- If the authentication string begins with `+`, the plugin constructs the full user DN value from the user name sent by the client, together with the DN specified in the authentication string (with the `+` removed). In the constructed DN, the client user name becomes the value of the attribute that specifies LDAP user names. This is `uid` by default; to change the attribute, modify the `authentication_ldap_simple_user_search_attr` or `authentication_ldap_sasl_user_search_attr` system variable appropriately. The authentication string is stored as given in the `mysql.user` system table, with the full user DN constructed on the fly before authentication.

This account authentication string does not have `+` at the beginning, so it is taken as the full user DN:

```
CREATE USER 'baldwin'
  IDENTIFIED WITH authentication_ldap_simple
  AS 'uid=admin,ou=People,dc=example,dc=com';
```

The client connects with the user name specified in the account (`baldwin`). In this case, that name is not used because the authentication string has no prefix and thus fully specifies the user DN.

This account authentication string does have `+` at the beginning, so it is taken as just part of the user DN:

```
CREATE USER 'accounting'
  IDENTIFIED WITH authentication_ldap_simple
  AS '+ou=People,dc=example,dc=com';
```

The client connects with the user name specified in the account (`accounting`), which in this case is used as the `uid` attribute together with the authentication string to construct the user DN: `uid=accounting,ou=People,dc=example,dc=com`

The accounts in the preceding examples have a nonempty user name, so the client always connects to the MySQL server using the same name as specified in the account definition. If an account has an empty user name, such as the default anonymous `''@'%'` proxy account described in LDAP Authentication with Proxying, clients might connect to the MySQL server with varying user names. But the principle is the same: If the authentication string begins with `+`, the plugin uses the user name sent by the client together with the authentication string to construct the user DN.

LDAP Authentication with Proxying

LDAP authentication plugins support proxying, enabling a user to connect to the MySQL server as one user but assume the privileges of a different user. This section describes basic LDAP plugin proxy support. The LDAP plugins also support specification of group preference and proxy user mapping; see LDAP Authentication Group Preference and Mapping Specification.

The authentication scheme described here uses proxying based on mapping LDAP group attribute values to connecting MySQL users who authenticate using LDAP onto other MySQL accounts that define different

sets of privileges. Users do not connect directly through the accounts that define the privileges. Instead, they connect through a default proxy account authenticated with LDAP, such that all the external logins are mapped to the MySQL accounts that hold the privileges. Any user who connects using the proxy account is mapped to one of those MySQL accounts, the privileges for which determine the database operations permitted to the external user.

The instructions here assume the following scenario:

- LDAP entries use the `uid` and `cn` attributes to specify user name and group values, respectively. To use different user and group attribute names, set the appropriate system variables to configure the plugin:
  - For `authentication_ldap_simple`: Set `authentication_ldap_simple_user_search_attr` and `authentication_ldap_simple_group_search_attr`.
  - For `authentication_ldap_sasl`: Set `authentication_ldap_sasl_user_search_attr` and `authentication_ldap_sasl_group_search_attr`.
- These LDAP entries are available in the directory managed by the LDAP server, to provide distinguished name values that uniquely identify each user:

  ```
  uid=basha,ou=People,dc=example,dc=com,cn=accounting
  uid=basil,ou=People,dc=example,dc=com,cn=front_office
  ```

  The group attribute values will become the authenticated user names, so they name the `accounting` and `front_office` proxied accounts.
- The examples assume use of SASL LDAP authentication. Make the appropriate adjustments for simple LDAP authentication.

Create the default proxy MySQL account:

```
CREATE USER ''@'%'
  IDENTIFIED WITH authentication_ldap_simple;
```

The proxy account definition has no `AS 'auth_string'` clause to name an LDAP user DN. Thus:

- When clients connect, the client user name is used as the LDAP user name to search for.
- The matching LDAP entry is expected to include a group attribute naming the proxied MySQL account that defines the privileges the client should have.

Note

If your MySQL installation has anonymous users, they might conflict with the default proxy user. For more information about this issue, and ways of dealing with it, see Default Proxy User and Anonymous User Conflicts.

Create the proxied accounts and grant to them the privileges required for MySQL access:

```
CREATE USER 'accounting'@'localhost';
  IDENTIFIED WITH mysql_no_login;
CREATE USER 'front_office'@'localhost';
  IDENTIFIED WITH mysql_no_login;

GRANT ALL PRIVILEGES
  ON accountingdb.*
```

```
  TO 'accounting'@'localhost';
GRANT ALL PRIVILEGES
  ON frontdb.*
  TO 'front_office'@'localhost';
```

The proxied accounts use the `mysql_no_login` authentication plugin to prevent clients from using the accounts to log in directly to the MySQL server. Instead, it is expected that users who authenticate using LDAP will use the default `''@'%'` proxy account. (This assumes that the plugin is installed. For instructions, see Section 6.4.1.8, “No-Login Pluggable Authentication”.) For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

Grant to the proxy account the `PROXY` privilege for each proxied account:

```
GRANT PROXY
  ON 'accounting'@'localhost'
  TO ''@'%';
GRANT PROXY
  ON 'front_office'@'localhost'
  TO ''@'%';
```

Use the `mysql` command-line client to connect to the MySQL server as `basha`.

```
shell> mysql --user=basha --password
Enter password: basha_password (basha LDAP password)
```

Authentication occurs as follows:

1. The server authenticates the connection using the default `''@'%'` proxy account, for client user `basha`.
2. The matching LDAP entry is:

   ```
   uid=basha,ou=People,dc=example,dc=com,cn=accounting
   ```

3. The matching LDAP entry has group attribute `cn=accounting`, so `accounting` becomes the authenticated user.
4. The authenticated user differs from the client user name `basha`, with the result that `basha` is treated as a proxy for `accounting`, and `basha` assumes the privileges of the `accounting` account. The following query should return output as shown:

   ```
   mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
   +-----------------+---------------------+--------------+
   | USER()          | CURRENT_USER()      | @@proxy_user |
   +-----------------+---------------------+--------------+
   | basha@localhost | accounting@localhost | ''@'%'       |
   +-----------------+---------------------+--------------+
   ```

This demonstrates that `basha` uses the privileges granted to the `accounting` MySQL account, and that proxying occurred through the default proxy user account.

Now connect as `basil` instead:

```
shell> mysql --user=basil --password
Enter password: basil_password (basil LDAP password)
```

The authentication process for `basil` is similar to that previously described for `basha`:

1. The server authenticates the connection using the default `''@'%'` proxy account, for client user `basil`.
2. The matching LDAP entry is:

```
uid=basil,ou=People,dc=example,dc=com,cn=front_office
```

3. The matching LDAP entry has group attribute `cn=front_office`, so `front_office` becomes the authenticated user.
4. The authenticated user differs from the client user name `basil`, with the result that `basil` is treated as a proxy for `front_office`, and `basil` assumes the privileges of the `front_office` account. The following query should return output as shown:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+----------------+------------------------+--------------+
| USER()         | CURRENT_USER()         | @@proxy_user |
+----------------+------------------------+--------------+
| basil@localhost | front_office@localhost | ''@'%'       |
+----------------+------------------------+--------------+
```

This demonstrates that `basil` uses the privileges granted to the `front_office` MySQL account, and that proxying occurred through the default proxy user account.

LDAP Authentication Group Preference and Mapping Specification

As described in LDAP Authentication with Proxying, basic LDAP authentication proxying works by the principle that the plugin uses the first group name returned by the LDAP server as the MySQL proxy user account name. This simple capability does not enable specifying any preference about which group name to use if the LDAP server returns multiple names, or any name other than the group name as the proxy user name.

As of MySQL 8.0.14, for MySQL accounts that use LDAP authentication, the authentication string can specify the following information to enable greater proxying flexibility:

- A list of groups in preference order, such that the plugin uses the first group name in the list that matches a group returned by the LDAP server.
- A mapping from group names to proxy user names, such that a group name when matched can provide a specified name to use as the proxy user. This provides an alternative to using the group name as the proxy user.

Consider the following MySQL proxy account definition:

```
CREATE USER ''@'%'
  IDENTIFIED WITH authentication_ldap_sasl
  AS '+ou=People,dc=example,dc=com#grp1=usera,grp2,grp3=userc';
```

The authentication string has a user DN suffix `ou=People,dc=example,dc=com` prefixed by the `+` character. Thus, as described in LDAP Authentication User DN Suffixes, the full user DN is constructed from the user DN suffix as specified, plus the client user name as the `uid` attribute.

The remaining part of the authentication string begins with `#`, which signifies the beginning of group preference and mapping information. This part of the authentication string lists group names in the order `grp1`, `grp2`, `grp3`. The LDAP plugin compares that list with the set of group names returned by the LDAP server, looking in list order for a match against the returned names. The plugin uses the first match, or if there is no match, authentication fails.

Suppose that the LDAP server returns groups `grp3`, `grp2`, and `grp7`. The LDAP plugin uses `grp2` because it is the first group in the authentication string that matches, even though it is not the first group returned by the LDAP server. If the LDAP server returns `grp4`, `grp2`, and `grp1`, the plugin uses `grp1` even though `grp2` also matches. `grp1` has a precedence higher than `grp2` because it is listed earlier in the authentication string.

Assuming that the plugin finds a group name match, it performs mapping from that group name to the MySQL proxy user name, if there is one. For the example proxy account, mapping occurs as follows:

- If the matching group name is `grp1` or `grp3`, those are associated in the authentication string with user names `usera` and `userc`, respectively. The plugin uses the corresponding associated user name as the proxy user name.
- If the matching group name is `grp2`, there is no associated user name in the authentication string. The plugin uses `grp2` as the proxy user name.

If the LDAP server returns a group in DN format, the LDAP plugin parses the group DN to extract the group name from it.

To specify LDAP group preference and mapping information, these principles apply:

- Begin the group preference and mapping part of the authentication string with a `#` prefix character.
- The group preference and mapping specification is a list of one or more items, separated by commas. Each item has the form *`group_name=user_name`* or *`group_name`*. Items should be listed in group name preference order. For a group name selected by the plugin as a match from set of group names returned by the LDAP server, the two syntaxes differ in effect as follows:
  - For an item specified as *`group_name=user_name`* (with a user name), the group name maps to the user name, which is used as the MySQL proxy user name.
  - For an item specified as *`group_name`* (with no user name), the group name is used as the MySQL proxy user name.
- To quote a group or user name that contains special characters such as space, surround it by double quote (`"`) characters. For example, if an item has group and user names of `my group name` and `my user name`, it must be written in a group mapping using quotes:

  ```
  "my group name"="my user name"
  ```

  If an item has group and user names of `my_group_name` and `my_user_name` (which contain no special characters), it may but need not be written using quotes. Any of the following are valid:

  ```
  my_group_name=my_user_name
  my_group_name="my_user_name"
  "my_group_name"=my_user_name
  "my_group_name"="my_user_name"
  ```

- To escape a character, precede it by a backslash (`\`). This is useful particularly to include a literal double quote or backslash, which are otherwise not included literally.
- A user DN need not be present in the authentication string, but if present, it must precede the group preference and mapping part. A user DN can be given as a full user DN, or as a user DN suffix with a `+` prefix character.

6.4.1.8 No-Login Pluggable Authentication

The `mysql_no_login` server-side authentication plugin prevents all client connections to any account that uses it. Use cases for this plugin include:

- Accounts that must be able to execute stored programs and views with elevated privileges without exposing those privileges to ordinary users.
- Proxied accounts that should never permit direct login but are intended to be accessed only through proxy accounts.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.19 Plugin and Library Names for No-Login Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`mysql_no_login`
Client-side plugin	None
Library file	`mysql_no_login.so`

The following sections provide installation and usage information specific to no-login pluggable authentication:

- Installing No-Login Pluggable Authentication
- Uninstalling No-Login Pluggable Authentication
- Using No-Login Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”. For proxy user information, see Section 6.2.18, “Proxy Users”.

Installing No-Login Pluggable Authentication

This section describes how to install the no-login authentication plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `mysql_no_login`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=mysql_no_login.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN mysql_no_login SONAME 'mysql_no_login.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, "Obtaining Server Plugin Information"). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%login%';
+---------------+---------------+
| PLUGIN_NAME   | PLUGIN_STATUS |
+---------------+---------------+
| mysql_no_login | ACTIVE        |
+---------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the no-login plugin, see Using No-Login Pluggable Authentication.

Uninstalling No-Login Pluggable Authentication

The method used to uninstall the no-login authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN mysql_no_login;
```

Using No-Login Pluggable Authentication

This section describes how to use the no-login authentication plugin to prevent accounts from being used for connecting from MySQL client programs to the server. It is assumed that the server is running with the no-login plugin enabled, as described in Installing No-Login Pluggable Authentication.

To refer to the no-login authentication plugin in the `IDENTIFIED WITH` clause of a `CREATE USER` statement, use the name `mysql_no_login`.

An account that authenticates using `mysql_no_login` may be used as the `DEFINER` for stored program and view objects. If such an object definition also includes `SQL SECURITY DEFINER`, it executes with that account's privileges. DBAs can use this behavior to provide access to confidential or sensitive data that is exposed only through well-controlled interfaces.

The following example illustrates these principles. It defines an account that does not permit client connections, and associates with it a view that exposes only certain columns of the `mysql.user` system table:

```
CREATE DATABASE nologindb;
CREATE USER 'nologin'@'localhost'
  IDENTIFIED WITH mysql_no_login;
GRANT ALL ON nologindb.*
  TO 'nologin'@'localhost';
GRANT SELECT ON mysql.user
  TO 'nologin'@'localhost';
CREATE DEFINER = 'nologin'@'localhost'
```

```
  SQL SECURITY DEFINER
  VIEW nologindb.myview
  AS SELECT User, Host FROM mysql.user;
```

To provide protected access to the view to an ordinary user, do this:

```
GRANT SELECT ON nologindb.myview
  TO 'ordinaryuser'@'localhost';
```

Now the ordinary user can use the view to access the limited information it presents:

```
SELECT * FROM nologindb.myview;
```

Attempts by the user to access columns other than those exposed by the view result in an error, as do attempts to select from the view by users not granted access to it.

Note

Because the `nologin` account cannot be used directly, the operations required to set up objects that it uses must be performed by `root` or similar account that has the privileges required to create the objects and set `DEFINER` values.

The `mysql_no_login` plugin is also useful in proxying scenarios. (For a discussion of concepts involved in proxying, see Section 6.2.18, “Proxy Users”.) An account that authenticates using `mysql_no_login` may be used as a proxied user for proxy accounts:

```
-- create proxied account
CREATE USER 'proxied_user'@'localhost'
  IDENTIFIED WITH mysql_no_login;
-- grant privileges to proxied account
GRANT ...
  ON ...
  TO 'proxied_user'@'localhost';
-- permit proxy_user to be a proxy account for proxied account
GRANT PROXY
  ON 'proxied_user'@'localhost'
  TO 'proxy_user'@'localhost';
```

This enables clients to access MySQL through the proxy account (`proxy_user`) but not to bypass the proxy mechanism by connecting directly as the proxied user (`proxied_user`). A client who connects using the `proxy_user` account has the privileges of the `proxied_user` account, but `proxied_user` itself cannot be used to connect.

For alternative methods of protecting proxied accounts against direct use, see Preventing Direct Login to Proxied Accounts.

6.4.1.9 Socket Peer-Credential Pluggable Authentication

The server-side `auth_socket` authentication plugin authenticates clients that connect from the local host through the Unix socket file. The plugin uses the `SO_PEERCRED` socket option to obtain information about the user running the client program. Thus, the plugin can be used only on systems that support the `SO_PEERCRED` option, such as Linux.

The source code for this plugin can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

The following table shows the plugin and library file names. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.20 Plugin and Library Names for Socket Peer-Credential Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`auth_socket`
Client-side plugin	None, see discussion
Library file	`auth_socket.so`

The following sections provide installation and usage information specific to socket pluggable authentication:

- Installing Socket Pluggable Authentication
- Uninstalling Socket Pluggable Authentication
- Using Socket Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”.

Installing Socket Pluggable Authentication

This section describes how to install the socket authentication plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file:

```
[mysqld]
plugin-load-add=auth_socket.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement:

```
INSTALL PLUGIN auth_socket SONAME 'auth_socket.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%socket%';
+-------------+---------------+
| PLUGIN_NAME | PLUGIN_STATUS |
+-------------+---------------+
| auth_socket | ACTIVE        |
+-------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the socket plugin, see Using Socket Pluggable Authentication.

Uninstalling Socket Pluggable Authentication

The method used to uninstall the socket authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN auth_socket;
```

Using Socket Pluggable Authentication

The socket plugin checks whether the socket user name (the operating system user name) matches the MySQL user name specified by the client program to the server. If the names do not match, the plugin checks whether the socket user name matches the name specified in the `authentication_string` column of the `mysql.user` system table row. If a match is found, the plugin permits the connection. The `authentication_string` value can be specified using an `IDENTIFIED ...AS` clause with `CREATE USER` or `ALTER USER`.

Suppose that a MySQL account is created for an operating system user named `valerie` who is to be authenticated by the `auth_socket` plugin for connections from the local host through the socket file:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
```

If a user on the local host with a login name of `stefanie` invokes `mysql` with the option `--user=valerie` to connect through the socket file, the server uses `auth_socket` to authenticate the client. The plugin determines that the `--user` option value (`valerie`) differs from the client user's name (`stephanie`) and refuses the connection. If a user named `valerie` tries the same thing, the plugin finds that the user name and the MySQL user name are both `valerie` and permits the connection. However, the plugin refuses the connection even for `valerie` if the connection is made using a different protocol, such as TCP/IP.

To permit both the `valerie` and `stephanie` operating system users to access MySQL through socket file connections that use the account, this can be done two ways:

- Name both users at account-creation time, one following `CREATE USER`, and the other in the authentication string:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket AS 'stephanie';
```

- If you have already used `CREATE USER` to create the account for a single user, use `ALTER USER` to add the second user:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
ALTER USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket AS 'stephanie';
```

To access the account, both `valerie` and `stephanie` specify `--user=valerie` at connect time.

6.4.1.10 Test Pluggable Authentication

MySQL includes a test plugin that checks account credentials and logs success or failure to the server error log. This is a loadable plugin (not built in) and must be installed prior to use.

The test plugin source code is separate from the server source, unlike the built-in native plugin, so it can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

Note

This plugin is intended for testing and development purposes, and is not for use in production environments or on servers that are exposed to public networks.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.21 Plugin and Library Names for Test Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`test_plugin_server`
Client-side plugin	`auth_test_plugin`
Library file	`auth_test_plugin.so`

The following sections provide installation and usage information specific to test pluggable authentication:

- Installing Test Pluggable Authentication
- Uninstalling Test Pluggable Authentication
- Using Test Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.17, “Pluggable Authentication”.

Installing Test Pluggable Authentication

This section describes how to install the test authentication plugin. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=auth_test_plugin.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN test_plugin_server SONAME 'auth_test_plugin.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, "Obtaining Server Plugin Information"). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%test_plugin%';
+--------------------+---------------+
| PLUGIN_NAME        | PLUGIN_STATUS |
+--------------------+---------------+
| test_plugin_server | ACTIVE        |
+--------------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the test plugin, see Using Test Pluggable Authentication.

Uninstalling Test Pluggable Authentication

The method used to uninstall the test authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

  ```
  UNINSTALL PLUGIN test_plugin_server;
  ```

Using Test Pluggable Authentication

To use the test authentication plugin, create an account and name that plugin in the `IDENTIFIED WITH` clause:

```
CREATE USER 'testuser'@'localhost'
IDENTIFIED WITH test_plugin_server
BY 'testpassword';
```

Then provide the `--user` and `--password` options for that account when you connect to the server. For example:

```
shell> mysql --user=testuser --password
Enter password: testpassword
```

The plugin fetches the password as received from the client and compares it with the value stored in the `authentication_string` column of the account row in the `mysql.user` system table. If the two values match, the plugin returns the `authentication_string` value as the new effective user ID.

You can look in the server error log for a message indicating whether authentication succeeded (notice that the password is reported as the "user"):

```
[Note] Plugin test_plugin_server reported:
```

```
'successfully authenticated user testpassword'
```

6.4.1.11 Pluggable Authentication System Variables

These variables are unavailable unless the appropriate server-side plugin is installed:

- `authentication_ldap_sasl` for system variables with names of the form `authentication_ldap_sasl_xxx`
- `authentication_ldap_simple` for system variables with names of the form `authentication_ldap_simple_xxx`

Table 6.22 Authentication Plugin System Variable Summary

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
authentication_ldap_sasl_auth_method_name	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_bind_base_dn	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_bind_root_dn	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_bind_root_pwd	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_ca_path	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_group_search_attr	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_group_search_filter	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_init_pool_size	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_log_status	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_max_pool_size	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_server_host	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_server_port	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_tls	Yes	Yes	Yes		Global	Yes
authentication_ldap_sasl_user_search_attr	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_auth_method_name	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_bind_base_dn	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_bind_root_dn	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_bind_root_pwd	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_ca_path	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_group_search_attr	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_group_search_filter	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_init_pool_size	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_log_status	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_max_pool_size	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_server_host	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_server_port	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_tls	Yes	Yes	Yes		Global	Yes
authentication_ldap_simple_user_search_attr	Yes	Yes	Yes		Global	Yes
authentication_windows_log_level	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
authentication_windows_use_principal_name	Yes	Yes	Yes		Global	No

- `authentication_ldap_sasl_auth_method_name`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-auth-method-name=value`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_auth_method_name`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`SCRAM-SHA-1`

For SASL LDAP authentication, the authentication method name. Communication between the authentication plugin and the LDAP server occurs according to this authentication method. These authentication method values are permitted:

- `SCRAM-SHA-1`: Authentication uses a SASL challenge-response mechanism to ensure password security.

  The client-side `authentication_ldap_sasl_client` plugin communicates with the SASL server, using the password to create a challenge and obtain a SASL request buffer, then passes this buffer to the server-side `authentication_ldap_sasl` plugin. The client-side and server-side SASL LDAP plugins use SASL messages for secure transmission of credentials within the LDAP protocol, to avoid sending the cleartext password between the MySQL client and server.

- `authentication_ldap_sasl_bind_base_dn`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-bind-base-dn=value`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_bind_base_dn`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`NULL`

For SASL LDAP authentication, the base distinguished name (DN). This variable can be used to limit the scope of searches by anchoring them at a certain location (the “base”) within the search tree.

Suppose that members of one set of LDAP user entries each have this form:

```
uid=user_name,ou=People,dc=example,dc=com
```

And that members of another set of LDAP user entries each have this form:

```
uid=user_name,ou=Admin,dc=example,dc=com
```

Then searches work like this for different base DN values:

- If the base DN is `ou=People,dc=example,dc=com`: Searches find user entries only in the first set.
- If the base DN is `ou=Admin,dc=example,dc=com`: Searches find user entries only in the second set.
- If the base DN is `ou=dc=example,dc=com`: Searches find user entries in the first or second set.

In general, more specific base DN values result in faster searches because they limit the search scope more.

- `authentication_ldap_sasl_bind_root_dn`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-bind-root-dn=value`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_bind_root_dn`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`NULL`

For SASL LDAP authentication, the root distinguished name (DN). This variable is used in conjunction with `authentication_ldap_sasl_bind_root_pwd` as the credentials for authenticating to the LDAP server for the purpose of performing searches. Authentication uses either one or two LDAP bind operations, depending on whether the MySQL account names an LDAP user DN:

- If the account does not name a user DN: `authentication_ldap_sasl` performs an initial LDAP binding using `authentication_ldap_sasl_bind_root_dn` and `authentication_ldap_sasl_bind_root_pwd`. (These are both empty by default, so if they are not set, the LDAP server must permit anonymous connections.) The resulting bind LDAP handle is used to search for the user DN, based on the client user name. `authentication_ldap_sasl` performs a second bind using the user DN and client-supplied password.
- If the account does name a user DN: The first bind operation is unnecessary in this case. `authentication_ldap_sasl` performs a single bind using the user DN and client-supplied password. This is faster than if the MySQL account does not specify an LDAP user DN.

- `authentication_ldap_sasl_bind_root_pwd`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-bind-root-pwd=value`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_bind_root_pwd`
Scope	Global

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	NULL

For SASL LDAP authentication, the password for the root distinguished name. This variable is used in conjunction with authentication_ldap_sasl_bind_root_dn. See the description of that variable.

- authentication_ldap_sasl_ca_path

Property	Value
Command-Line Format	--authentication-ldap-sasl-ca-path=value
Introduced	8.0.11
System Variable	authentication_ldap_sasl_ca_path
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	NULL

For SASL LDAP authentication, the absolute path of the certificate authority file. Specify this file if it is desired that the authentication plugin perform verification of the LDAP server certificate.

Note

In addition to setting the authentication_ldap_sasl_ca_path variable to the file name, you must add the appropriate certificate authority certificates to the file and enable the authentication_ldap_sasl_tls system variable.

- authentication_ldap_sasl_group_search_attr

Property	Value
Command-Line Format	--authentication-ldap-sasl-group-search-attr=value
Introduced	8.0.11
System Variable	authentication_ldap_sasl_group_search_attr
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	cn

For SASL LDAP authentication, the name of the attribute that specifies group names in LDAP directory entries. If authentication_ldap_sasl_group_search_attr has its default value of cn, searches return the cn value as the group name. For example, if an LDAP entry with a uid value of user1 has a cn attribute of mygroup, searches for user1 return mygroup as the group name.

This variable should be the empty string if you want no group or proxy authentication.

If the group search attribute is `isMemberOf`, LDAP authentication directly retrieves the user attribute `isMemberOf` value and assigns it as group information. If the group search attribute is not `isMemberOf`, LDAP authentication searches for all groups where the user is a member. (The latter is the default behavior.) This behavior is based on how LDAP group information can be stored two ways: 1) A group entry can have an attribute named `memberUid` or `member` with a value that is a user name; 2) A user entry can have an attribute named `isMemberOf` with values that are group names.

- `authentication_ldap_sasl_group_search_filter`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-group-search-filter=value`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_group_search_filter`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`(|(&(objectClass=posixGroup)(memberUid=%s)) (&(objectClass=group)(member=%s)))`

For SASL LDAP authentication, the custom group search filter.

The search filter value can contain `{UA}` and `{UD}` notation to represent the user name and the full user DN. For example, `{UA}` is replaced with a user name such as `"admin"`, whereas `{UD}` is replaced with a use full DN such as `"uid=admin,ou=People,dc=example,dc=com"`. The following value is the default, which supports both OpenLDAP and Active Directory:

```
(|(&(objectClass=posixGroup)(memberUid={UA}))
  (&(objectClass=group)(member={UD})))
```

In some cases for the user scenario, `memberOf` is a simple user attribute that holds no group information. For additional flexibility, an optional `{GA}` prefix can be used with the group search attribute. Any group attribute with a {GA} prefix is treated as a user attribute having group names. For example, with a value of `{GA}MemberOf`, if the group value is the DN, the first attribute value from the group DN is returned as the group name.

- `authentication_ldap_sasl_init_pool_size`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-init-pool-size=#`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_init_pool_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value	`10`
Minimum Value	`0`
Maximum Value	`32767`

For SASL LDAP authentication, the initial size of the pool of connections to the LDAP server. Choose the value for this variable based on the average number of concurrent authentication requests to the LDAP server.

The plugin uses `authentication_ldap_sasl_init_pool_size` and `authentication_ldap_sasl_max_pool_size` together for connection-pool management:

- When the authentication plugin initializes, it creates `authentication_ldap_sasl_init_pool_size` connections, unless `authentication_ldap_sasl_max_pool_size=0` to disable pooling.
- If the plugin receives an anthentication request when there are no free connections in the current connection pool, the plugin can create a new connection, up to the maximum connection pool size given by `authentication_ldap_sasl_max_pool_size`.
- If the plugin receives a request when the pool size is already at its maximum and there are no free connections, authentication fails.
- When the plugin unloads, it closes all pooled connections.

Changes to plugin system variable settings may have no effect on connections already in the pool. For example, modifying the LDAP server host, port, or TLS settings does not affect existing connections. However, if the original variable values were invalid and the connection pool could not be initialized, the plugin attempts to reinitialize the pool for the next LDAP request. In this case, the new system variable values are used for the reinitialization attempt.

If `authentication_ldap_sasl_max_pool_size=0` to disable pooling, each LDAP connection opened by the plugin uses the values the system variables have at that time.

- `authentication_ldap_sasl_log_status`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-log-status=#`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_log_status`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value (>= 8.0.18)	`6`
Maximum Value (<= 8.0.17)	`5`

For SASL LDAP authentication, the logging level for messages written to the error log. The following table shows the permitted level values and their meanings.

Table 6.23 Log Levels for authentication_ldap_sasl_log_status

Option Value	Types of Messages Logged
1	No messages
2	Error messages
3	Error and warning messages
4	Error, warning, and information messages
5	Same as previous level plus debugging messages from MySQL
6	Same as previous level plus debugging messages from LDAP library

Log level 6 is available as of MySQL 8.0.18.

On the client side, messages can be logged to the standard output by setting the `AUTHENTICATION_LDAP_CLIENT_LOG` environment variable. The permitted and default values are the same as for `authentication_ldap_sasl_log_status`.

The `AUTHENTICATION_LDAP_CLIENT_LOG` environment variable applies only to SASL LDAP authentication. It has no effect for simple LDAP authentication because the client plugin in that case is `mysql_clear_password`, which knows nothing about LDAP operations.

- `authentication_ldap_sasl_max_pool_size`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-max-pool-size=#`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_max_pool_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1000`
Minimum Value	`0`
Maximum Value	`32767`

For SASL LDAP authentication, the maximum size of the pool of connections to the LDAP server. To disable connection pooling, set this variable to 0.

This variable is used in conjunction with `authentication_ldap_sasl_init_pool_size`. See the description of that variable.

- `authentication_ldap_sasl_server_host`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-server-host=host_name`

Property	Value
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_server_host`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

For SASL LDAP authentication, the LDAP server host. The permitted values for this variable depend on the authentication method:

- For `authentication_ldap_sasl_auth_method_name=SCRAM-SHA-1`: The LDAP server host can be a host name or IP address.

- `authentication_ldap_sasl_server_port`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-server-port=port_num`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_server_port`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`389`
Minimum Value	`1`
Maximum Value	`32376`

For SASL LDAP authentication, the LDAP server TCP/IP port number.

As of MySQL 8.0.14, if the LDAP port number is configured as 636 or 3269, the plugin uses LDAPS (LDAP over SSL) instead of LDAP. (LDAPS differs from `startTLS`.)

- `authentication_ldap_sasl_tls`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-tls[={OFF|ON}]`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_tls`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

For SASL LDAP authentication, whether connections by the plugin to the LDAP server are secure. If this variable is enabled, the plugin uses TLS to connect securely to the LDAP server. If you enable this variable, you may also wish to set the `authentication_ldap_sasl_ca_path` variable.

MySQL LDAP plugins support the StartTLS method, which initializes TLS on top of a plain LDAP connection. The `ldaps` method is deprecated and MySQL does not support it.

- `authentication_ldap_sasl_user_search_attr`

Property	Value
Command-Line Format	`--authentication-ldap-sasl-user-search-attr=value`
Introduced	8.0.11
System Variable	`authentication_ldap_sasl_user_search_attr`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`uid`

For SASL LDAP authentication, the name of the attribute that specifies user names in LDAP directory entries. If a user distinguished name is not provided, the authentication plugin searches for the name using this attribute. For example, if the `authentication_ldap_sasl_user_search_attr` value is `uid`, a search for the user name `user1` finds entries with a `uid` value of `user1`.

- `authentication_ldap_simple_auth_method_name`

Property	Value
Command-Line Format	`--authentication-ldap-simple-auth-method-name=value`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_auth_method_name`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`SIMPLE`

For simple LDAP authentication, the authentication method name. Communication between the authentication plugin and the LDAP server occurs according to this authentication method. These authentication method values are permitted:

- `SIMPLE`: This authentication method uses either one or two LDAP bind operations, depending on whether the MySQL account names an LDAP user distinguished name. See the description of `authentication_ldap_simple_bind_root_dn`.
- `AD-FOREST`: `authentication_ldap_simple` searches all the domains in the Active Directory forest, performing an LDAP bind to each Active Directory domain until the user is found in some domain.

Note

For simple LDAP authentication, it is recommended to also set TLS parameters to require that communication with the LDAP server take place over secure connections.

- `authentication_ldap_simple_bind_base_dn`

Property	Value
Command-Line Format	`--authentication-ldap-simple-bind-base-dn=value`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_bind_base_dn`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`NULL`

For simple LDAP authentication, the base distinguished name (DN). This variable can be used to limit the scope of searches by anchoring them at a certain location (the “base”) within the search tree.

Suppose that members of one set of LDAP user entries each have this form:

```
uid=user_name,ou=People,dc=example,dc=com
```

And that members of another set of LDAP user entries each have this form:

```
uid=user_name,ou=Admin,dc=example,dc=com
```

Then searches work like this for different base DN values:

- If the base DN is `ou=People,dc=example,dc=com`: Searches find user entries only in the first set.
- If the base DN is `ou=Admin,dc=example,dc=com`: Searches find user entries only in the second set.
- If the base DN is `ou=dc=example,dc=com`: Searches find user entries in the first or second set.

In general, more specific base DN values result in faster searches because they limit the search scope more.

- `authentication_ldap_simple_bind_root_dn`

Property	Value
Command-Line Format	`--authentication-ldap-simple-bind-root-dn=value`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_bind_root_dn`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	String
Default Value	NULL

For simple LDAP authentication, the root distinguished name (DN). This variable is used in conjunction with authentication_ldap_simple_bind_root_pwd as the credentials for authenticating to the LDAP server for the purpose of performing searches. Authentication uses either one or two LDAP bind operations, depending on whether the MySQL account names an LDAP user DN:

- If the account does not name a user DN: authentication_ldap_simple performs an initial LDAP binding using authentication_ldap_simple_bind_root_dn and authentication_ldap_simple_bind_root_pwd. (These are both empty by default, so if they are not set, the LDAP server must permit anonymous connections.) The resulting bind LDAP handle is used to search for the user DN, based on the client user name. authentication_ldap_simple performs a second bind using the user DN and client-supplied password.
- If the account does name a user DN: The first bind operation is unnecessary in this case. authentication_ldap_simple performs a single bind using the user DN and client-supplied password. This is faster than if the MySQL account does not specify an LDAP user DN.

- authentication_ldap_simple_bind_root_pwd

Property	Value
Command-Line Format	--authentication-ldap-simple-bind-root-pwd=value
Introduced	8.0.11
System Variable	authentication_ldap_simple_bind_root_pwd
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	NULL

For simple LDAP authentication, the password for the root distinguished name. This variable is used in conjunction with authentication_ldap_simple_bind_root_dn. See the description of that variable.

- authentication_ldap_simple_ca_path

Property	Value
Command-Line Format	--authentication-ldap-simple-ca-path=value
Introduced	8.0.11
System Variable	authentication_ldap_simple_ca_path
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

Property	Value
Default Value	`NULL`

For simple LDAP authentication, the absolute path of the certificate authority file. Specify this file if it is desired that the authentication plugin perform verification of the LDAP server certificate.

Note

In addition to setting the `authentication_ldap_simple_ca_path` variable to the file name, you must add the appropriate certificate authority certificates to the file and enable the `authentication_ldap_simple_tls` system variable.

- `authentication_ldap_simple_group_search_attr`

Property	Value
Command-Line Format	`--authentication-ldap-simple-group-search-attr=value`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_group_search_attr`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`cn`

For simple LDAP authentication, the name of the attribute that specifies group names in LDAP directory entries. If `authentication_ldap_simple_group_search_attr` has its default value of `cn`, searches return the `cn` value as the group name. For example, if an LDAP entry with a `uid` value of `user1` has a `cn` attribute of `mygroup`, searches for `user1` return `mygroup` as the group name.

If the group search attribute is `isMemberOf`, LDAP authentication directly retrieves the user attribute `isMemberOf` value and assigns it as group information. If the group search attribute is not `isMemberOf`, LDAP authentication searches for all groups where the user is a member. (The latter is the default behavior.) This behavior is based on how LDAP group information can be stored two ways: 1) A group entry can have an attribute named `memberUid` or `member` with a value that is a user name; 2) A user entry can have an attribute named `isMemberOf` with values that are group names.

- `authentication_ldap_simple_group_search_filter`

Property	Value
Command-Line Format	`--authentication-ldap-simple-group-search-filter=value`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_group_search_filter`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

Property	Value
Default Value	`(|(&(objectClass=posixGroup)(memberUid=%s))(&(objectClass=group)(member=%s)))`

For simple LDAP authentication, the custom group search filter.

The search filter value can contain `{UA}` and `{UD}` notation to represent the user name and the full user DN. For example, `{UA}` is replaced with a user name such as `"admin"`, whereas `{UD}` is replaced with a use full DN such as `"uid=admin,ou=People,dc=example,dc=com"`. The following value is the default, which supports both OpenLDAP and Active Directory:

```
(|(&(objectClass=posixGroup)(memberUid={UA}))
  (&(objectClass=group)(member={UD})))
```

In some cases for the user scenario, `memberOf` is a simple user attribute that holds no group information. For additional flexibility, an optional `{GA}` prefix can be used with the group search attribute. Any group attribute with a {GA} prefix is treated as a user attribute having group names. For example, with a value of `{GA}MemberOf`, if the group value is the DN, the first attribute value from the group DN is returned as the group name.

- `authentication_ldap_simple_init_pool_size`

Property	Value
Command-Line Format	`--authentication-ldap-simple-init-pool-size=#`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_init_pool_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value	`32767`

For simple LDAP authentication, the initial size of the pool of connections to the LDAP server. Choose the value for this variable based on the average number of concurrent authentication requests to the LDAP server.

The plugin uses `authentication_ldap_simple_init_pool_size` and `authentication_ldap_simple_max_pool_size` together for connection-pool management:

- When the authentication plugin initializes, it creates `authentication_ldap_simple_init_pool_size` connections, unless `authentication_ldap_simple_max_pool_size=0` to disable pooling.
- If the plugin receives an anthentication request when there are no free connections in the current connection pool, the plugin can create a new connection, up to the maximum connection pool size given by `authentication_ldap_simple_max_pool_size`.

- If the plugin receives a request when the pool size is already at its maximum and there are no free connections, authentication fails.
- When the plugin unloads, it closes all pooled connections.

Changes to plugin system variable settings may have no effect on connections already in the pool. For example, modifying the LDAP server host, port, or TLS settings does not affect existing connections. However, if the original variable values were invalid and the connection pool could not be initialized, the plugin attempts to reinitialize the pool for the next LDAP request. In this case, the new system variable values are used for the reinitialization attempt.

If `authentication_ldap_simple_max_pool_size=0` to disable pooling, each LDAP connection opened by the plugin uses the values the system variables have at that time.

- `authentication_ldap_simple_log_status`

Property	Value
Command-Line Format	`--authentication-ldap-simple-log-status=#`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_log_status`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value (>= 8.0.18)	`6`
Maximum Value (<= 8.0.17)	`5`

For simple LDAP authentication, the logging level for messages written to the error log. The following table shows the permitted level values and their meanings.

Table 6.24 Log Levels for authentication_ldap_simple_log_status

Option Value	Types of Messages Logged
1	No messages
2	Error messages
3	Error and warning messages
4	Error, warning, and information messages
5	Same as previous level plus debugging messages from MySQL
6	Same as previous level plus debugging messages from LDAP library

Log level 6 is available as of MySQL 8.0.18.

- `authentication_ldap_simple_max_pool_size`

Property	Value
Command-Line Format	`--authentication-ldap-simple-max-pool-size=#`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_max_pool_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1000`
Minimum Value	`0`
Maximum Value	`32767`

For simple LDAP authentication, the maximum size of the pool of connections to the LDAP server. To disable connection pooling, set this variable to 0.

This variable is used in conjunction with `authentication_ldap_simple_init_pool_size`. See the description of that variable.

- `authentication_ldap_simple_server_host`

Property	Value
Command-Line Format	`--authentication-ldap-simple-server-host=host_name`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_server_host`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

For simple LDAP authentication, the LDAP server host. The permitted values for this variable depend on the authentication method:

- For `authentication_ldap_simple_auth_method_name=SIMPLE`: The LDAP server host can be a host name or IP address.
- For `authentication_ldap_simple_auth_method_name=AD-FOREST`. The LDAP server host can be an Active Directory domain name. For example, for an LDAP server URL of `ldap://example.mem.local:389`, the server name can be `mem.local`.

  An Active Directory forest setup can have multiple domains (LDAP server IPs), which can be discovered using DNS. On Unix and Unix-like systems, some additional setup may be required to configure your DNS server with SRV records that specify the LDAP servers for the Active Directory domain. Suppose that your configuration has these properties:

  - The name server that provides information about Active Directory domains has IP address `10.172.166.100`.

- The LDAP servers have names `ldap1.mem.local` through `ldap3.mem.local` and IP addresses `10.172.166.101` through `10.172.166.103`.

You want the LDAP servers to be discoverable using SRV searches. For example, at the command line, a command like this should list the LDAP servers:

```
host -t SRV _ldap._tcp.mem.local
```

Perform the DNS configuration as follows:

1. Add a line to `/etc/resolv.conf` to specify the name server that provides information about Active Directory domains:

   ```
   nameserver 10.172.166.100
   ```

2. Configure the appropriate zone file for the name server with SRV records for the LDAP servers:

   ```
   _ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap1.mem.local.
   _ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap2.mem.local.
   _ldap._tcp.mem.local. 86400 IN SRV 0 100 389 ldap3.mem.local.
   ```

3. It may also be necessary to specify the IP address for the LDAP servers in `/etc/hosts` if the server host cannot be resolved. For example, add lines like this to the file:

   ```
   10.172.166.101 ldap1.mem.local
   10.172.166.102 ldap2.mem.local
   10.172.166.103 ldap3.mem.local
   ```

With the DNS configured as just described, the server-side LDAP plugin can discover the LDAP servers and will try to authenticate in all domains until authentication succeeds or there are no more servers.

Windows needs no such settings as just described. Given the LDAP server host in the `authentication_ldap_simple_server_host` value, the Windows LDAP library searches all domains and attempts to authenticate.

- `authentication_ldap_simple_server_port`

Property	Value
Command-Line Format	`--authentication-ldap-simple-server-port=port_num`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_server_port`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`389`
Minimum Value	`1`
Maximum Value	`32376`

For simple LDAP authentication, the LDAP server TCP/IP port number.

As of MySQL 8.0.14, if the LDAP port number is configured as 636 or 3269, the plugin uses LDAPS (LDAP over SSL) instead of LDAP. (LDAPS differs from `startTLS`.)

- `authentication_ldap_simple_tls`

Property	Value
Command-Line Format	`--authentication-ldap-simple-tls[={OFF|ON}]`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_tls`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

For simple LDAP authentication, whether connections by the plugin to the LDAP server are secure. If this variable is enabled, the plugin uses TLS to connect securely to the LDAP server. If you enable this variable, you may also wish to set the `authentication_ldap_simple_ca_path` variable.

MySQL LDAP plugins support the StartTLS method, which initializes TLS on top of a plain LDAP connection. The `ldaps` method is deprecated and MySQL does not support it.

- `authentication_ldap_simple_user_search_attr`

Property	Value
Command-Line Format	`--authentication-ldap-simple-user-search-attr=value`
Introduced	8.0.11
System Variable	`authentication_ldap_simple_user_search_attr`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`uid`

For simple LDAP authentication, the name of the attribute that specifies user names in LDAP directory entries. If a user distinguished name is not provided, the authentication plugin searches for the name using this attribute. For example, if the `authentication_ldap_simple_user_search_attr` value is `uid`, a search for the user name `user1` finds entries with a `uid` value of `user1`.

6.4.2 The Connection-Control Plugins

MySQL Server includes a plugin library that enables administrators to introduce an increasing delay in server response to clients after a certain number of consecutive failed connection attempts. This capability provides a deterrent that slows down brute force attacks that attempt to access MySQL user accounts. The plugin library contains two plugins:

- `CONNECTION_CONTROL` checks incoming connections and adds a delay to server responses as necessary. This plugin also exposes system variables that enable its operation to be configured and a status variable that provides rudimentary monitoring information.

  The `CONNECTION_CONTROL` plugin uses the audit plugin interface (see Section 29.2.4.8, "Writing Audit Plugins"). To collect information, it subscribes to the `MYSQL_AUDIT_CONNECTION_CLASSMASK` event class, and processes `MYSQL_AUDIT_CONNECTION_CONNECT` and `MYSQL_AUDIT_CONNECTION_CHANGE_USER` subevents to check whether the server should introduce a delay before responding to client connection attempts.

- `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` implements an `INFORMATION_SCHEMA` table that exposes more detailed monitoring information for failed connection attempts.

The following sections provide information about connection-control plugin installation and configuration. For information about the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table, see Section 25.41.1, "The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table".

6.4.2.1 Connection-Control Plugin Installation

This section describes how to install the connection-control plugins, `CONNECTION_CONTROL` and `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS`. For general information about installing plugins, see Section 5.6.1, "Installing and Uninstalling Plugins".

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `connection_control`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugins at server startup, use the `--plugin-load-add` option to name the library file that contains them. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
plugin-load-add=connection_control.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugins at runtime, use these statements (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN CONNECTION_CONTROL
  SONAME 'connection_control.so';
INSTALL PLUGIN CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS
  SONAME 'connection_control.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, "Obtaining Server Plugin Information"). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
```

```
      WHERE PLUGIN_NAME LIKE 'connection%';
+---------------------------------------+---------------+
| PLUGIN_NAME                           | PLUGIN_STATUS |
+---------------------------------------+---------------+
| CONNECTION_CONTROL                    | ACTIVE        |
| CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS | ACTIVE        |
+---------------------------------------+---------------+
```

If a plugin failed to initialize, check the server error log for diagnostic messages.

If the plugins have been previously registered with `INSTALL PLUGIN` or are loaded with `--plugin-load-add`, you can use the `--connection-control` and `--connection-control-failed-login-attempts` options at server startup to control plugin activation. For example, to load the plugins at startup and prevent them from being removed at runtime, use these options:

```
[mysqld]
plugin-load-add=connection_control.so
connection-control=FORCE_PLUS_PERMANENT
connection-control-failed-login-attempts=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without a given connection-control plugin, use an option value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

Note

It is possible to install one plugin without the other, but both must be installed for full connection-control capability. In particular, installing only the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` plugin is of little use because without the `CONNECTION_CONTROL` plugin to provide the data that populates the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table, retrievals from the table will always be empty.

- Connection Delay Configuration
- Connection Failure Assessment
- Connection Failure Monitoring

Connection Delay Configuration

To enable you to configure its operation, the `CONNECTION_CONTROL` plugin exposes several system variables:

- `connection_control_failed_connections_threshold`: The number of consecutive failed connection attempts permitted to clients before the server adds a delay for subsequent connection attempts.
- `connection_control_min_connection_delay`: The amount of delay to add for each consecutive connection failure above the threshold.
- `connection_control_max_connection_delay`: The maximum delay to add.

To entirely disable checking for failed connection attempts, set `connection_control_failed_connections_threshold` to zero. If `connection_control_failed_connections_threshold` is nonzero, the amount of delay is zero up through that many consecutive failed connection attempts. Thereafter, the amount of delay is the number of failed attempts above the threshold, multiplied by `connection_control_min_connection_delay` milliseconds. For example, with the default `connection_control_failed_connections_threshold`

and connection_control_min_connection_delay values of 3 and 1000, respectively, there is no delay for the first three consecutive failed connection attempts by a client, a delay of 1000 milliseconds for the fourth failed attempt, 2000 milliseconds for the fifth failed attempt, and so on, up to the maximum delay permitted by connection_control_max_connection_delay.

You can set the CONNECTION_CONTROL system variables at server startup or runtime. Suppose that you want to permit four consecutive failed connection attempts before the server starts delaying its responses, and to increase the delay by 1500 milliseconds for each additional failure after that. To set the relevant variables at server startup, put these lines in the server my.cnf file:

```
[mysqld]
plugin-load-add=connection_control.so
connection_control_failed_connections_threshold=4
connection_control_min_connection_delay=1500
```

To set and persist the variables at runtime, use these statements:

```
SET PERSIST connection_control_failed_connections_threshold = 4;
SET PERSIST connection_control_min_connection_delay = 1500;
```

SET PERSIST sets the value for the running MySQL instance. It also saves the value, causing it to be used for subsequent server restarts. To change a value for the running MySQL instance without saving it for subsequent restarts, use the GLOBAL keyword rather than PERSIST. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

The connection_control_min_connection_delay and connection_control_max_connection_delay system variables have fixed minimum and maximum values of 1000 and 2147483647, respectively. In addition, the permitted range of values of each variable also depends on the current value of the other:

- connection_control_min_connection_delay cannot be set greater than the current value of connection_control_max_connection_delay.
- connection_control_max_connection_delay cannot be set less than the current value of connection_control_min_connection_delay.

Thus, to make the changes required for some configurations, you might need to set the variables in a specific order. Suppose that the current minimum and maximum delays are 1000 and 2000, and that you want to set them to 3000 and 5000. You cannot first set connection_control_min_connection_delay to 3000 because that is greater than the current connection_control_max_connection_delay value of 2000. Instead, set connection_control_max_connection_delay to 5000, then set connection_control_min_connection_delay to 3000.

Connection Failure Assessment

When the CONNECTION_CONTROL plugin is installed, it checks connection attempts and tracks whether they fail or succeed. For this purpose, a failed connection attempt is one for which the client user and host match a known MySQL account but the provided credentials are incorrect, or do not match any known account.

Failed-connection counting is based on the user/host combination for each connection attempt. Determination of the applicable user name and host name takes proxying into account and occurs as follows:

- If the client user proxies another user, the proxying user's information is used. For example, if external_user@example.com proxies proxy_user@example.com, connection

counting uses the proxying user, `external_user@example.com`, rather than the proxied user, `proxy_user@example.com`. Both `external_user@example.com` and `proxy_user@example.com` must have valid entries in the `mysql.user` system table and a proxy relationship between them must be defined in the `mysql.proxies_priv` system table (see Section 6.2.18, “Proxy Users”).

- If the client user does not proxy another user, but does match a `mysql.user` entry, counting uses the `CURRENT_USER()` value corresponding to that entry. For example, if a user `user1` connecting from a host `host1.example.com` matches a `user1@host1.example.com` entry, counting uses `user1@host1.example.com`. If the user matches a `user1@%.example.com`, `user1@%.com`, or `user1@%` entry instead, counting uses `user1@%.example.com`, `user1@%.com`, or `user1@%`, respectively.

For the cases just described, the connection attempt matches some `mysql.user` entry, and whether the request succeeds or fails depends on whether the client provides the correct authentication credentials. For example, if the client presents an incorrect password, the connection attempt fails.

If the connection attempt matches no `mysql.user` entry, the attempt fails. In this case, no `CURRENT_USER()` value is available and connection-failure counting uses the user name provided by the client and the client host as determined by the server. For example, if a client attempts to connect as user `user2` from host `host2.example.com`, the user name part is available in the client request and the server determines the host information. The user/host combination used for counting is `user2@host2.example.com`.

Note

The server maintains information about which client hosts can possibly connect to the server (essentially the union of host values for `mysql.user` entries). If a client attempts to connect from any other host, the server rejects the attempt at an early stage of connection setup:

```
ERROR 1130 (HY000): Host 'host_name' is not
allowed to connect to this MySQL server
```

Because this type of rejection occurs so early, `CONNECTION_CONTROL` does not see it, and does not count it.

Connection Failure Monitoring

To monitor failed connections, use these information sources:

- The `Connection_control_delay_generated` status variable indicates the number of times the server added a delay to its response to a failed connection attempt. This does not count attempts that occur before reaching the threshold defined by the `connection_control_failed_connections_threshold` system variable.
- The `INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table provides information about the current number of consecutive failed connection attempts per client user/host combination. This counts all failed attempts, regardless of whether they were delayed.

Assigning a value to `connection_control_failed_connections_threshold` at runtime resets all accumulated failed-connection counters to zero, which has these visible effects:

- The `Connection_control_delay_generated` status variable is reset to zero.
- The `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table becomes empty.

6.4.2.2 Connection-Control System and Status Variables

This section describes the system and status variables that the `CONNECTION_CONTROL` plugin provides to enable its operation to be configured and monitored.

- Connection-Control System Variables
- Connection-Control Status Variables

Connection-Control System Variables

If the `CONNECTION_CONTROL` plugin is installed, it exposes these system variables:

- `connection_control_failed_connections_threshold`

Property	Value
Command-Line Format	`--connection-control-failed-connections-threshold=#`
Introduced	8.0.1
System Variable	`connection_control_failed_connections_threshold`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`3`
Minimum Value	`0`
Maximum Value	`2147483647`

The number of consecutive failed connection attempts permitted to clients before the server adds a delay for subsequent connection attempts:

- If the variable has a nonzero value *`N`*, the server adds a delay beginning with consecutive failed attempt *`N`*+1. If a client has reached the point where connection responses are delayed, the delay also occurs for the next subsequent successful connection.
- Setting this variable to zero disables failed-connection counting. In this case, the server never adds delays.

For information about how `connection_control_failed_connections_threshold` interacts with other connection-control system and status variables, see Section 6.4.2.1, “Connection-Control Plugin Installation”.

- `connection_control_max_connection_delay`

Property	Value
Command-Line Format	`--connection-control-max-connection-delay=#`
Introduced	8.0.1
System Variable	`connection_control_max_connection_delay`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	`2147483647`
Minimum Value	`1000`
Maximum Value	`2147483647`

The maximum delay in milliseconds for server response to failed connection attempts, if `connection_control_failed_connections_threshold` is greater than zero.

For information about how `connection_control_max_connection_delay` interacts with other connection-control system and status variables, see Section 6.4.2.1, “Connection-Control Plugin Installation”.

- `connection_control_min_connection_delay`

Property	Value
Command-Line Format	`--connection-control-min-connection-delay=#`
Introduced	8.0.1
System Variable	`connection_control_min_connection_delay`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1000`
Minimum Value	`1000`
Maximum Value	`2147483647`

The minimum delay in milliseconds for server response to failed connection attempts, if `connection_control_failed_connections_threshold` is greater than zero. This is also the amount by which the server increases the delay for additional successive failures once it begins delaying.

For information about how `connection_control_min_connection_delay` interacts with other connection-control system and status variables, see Section 6.4.2.1, “Connection-Control Plugin Installation”.

Connection-Control Status Variables

If the `CONNECTION_CONTROL` plugin is installed, it exposes this status variable:

- `Connection_control_delay_generated`

  The number of times the server added a delay to its response to a failed connection attempt. This does not count attempts that occur before reaching the threshold defined by the `connection_control_failed_connections_threshold` system variable.

  This variable provides a simple counter. For more detailed connection-control monitoring information, examine the `INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table; see Section 25.41.1, “The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table”.

Assigning a value to `connection_control_failed_connections_threshold` at runtime resets `Connection_control_delay_generated` to zero.

6.4.3 The Password Validation Component

The `validate_password` component serves to improve security by requiring account passwords and enabling strength testing of potential passwords. This component exposes system variables that enable you to configure password policy, and status variables for component monitoring.

Note

In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. (For general information about server components, see Section 5.5, “MySQL Server Components”.) The following instructions describe how to use the component, not the plugin. For instructions on using the plugin form of `validate_password`, see The Password Validation Plugin in MySQL 5.7 Reference Manual.

The plugin form of `validate_password` is still available but is deprecated and will be removed in a future version of MySQL. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.4.3.3, “Transitioning to the Password Validation Component”.

The `validate_password` component implements these capabilities:

- For SQL statements that assign a password supplied as a cleartext value, `validate_password` checks the password against the current password policy and rejects the password if it is weak (the statement returns an `ER_NOT_VALID_PASSWORD` error). This applies to the `ALTER USER`, `CREATE USER`, and `SET PASSWORD` statements.
- For `CREATE USER` statements, `validate_password` requires that a password be given, and that it satisfies the password policy. This is true even if an account is locked initially because otherwise unlocking the account later would cause it to become accessible without a password that satisfies the policy.
- `validate_password` implements a `VALIDATE_PASSWORD_STRENGTH()` SQL function that assesses the strength of potential passwords. This function takes a password argument and returns an integer from 0 (weak) to 100 (strong).

Note

For statements that assign or modify account passwords (`ALTER USER`, `CREATE USER`, and `SET PASSWORD`), the `validate_password` capabilities described here apply only to accounts that use an authentication plugin that stores credentials internally to MySQL. For accounts that use plugins that perform authentication against a credentials system external to MySQL, password management must be handled externally against that system as well. For more information about internal credentials storage, see Section 6.2.15, “Password Management”.

The preceding restriction does not apply to use of the `VALIDATE_PASSWORD_STRENGTH()` function because it does not affect accounts directly.

Examples:

- `validate_password` checks the cleartext password in the following statement. Under the default password policy, which requires passwords to be at least 8 characters long, the password is weak and the statement produces an error:

```
mysql> ALTER USER USER() IDENTIFIED BY 'abc';
ERROR 1819 (HY000): Your password does not satisfy the current
policy requirements
```

- Passwords specified as hashed values are not checked because the original password value is not available for checking:

```
mysql> ALTER USER 'jeffrey'@'localhost'
       IDENTIFIED WITH mysql_native_password
       AS '*0D3CED9BEC10A777AEC23CCC353A8C08A633045E';
Query OK, 0 rows affected (0.01 sec)
```

- This account-creation statement fails, even though the account is locked initially, because it does not include a password that satisfies the current password policy:

```
mysql> CREATE USER 'juanita'@'localhost' ACCOUNT LOCK;
ERROR 1819 (HY000): Your password does not satisfy the current
policy requirements
```

- To check a password, use the `VALIDATE_PASSWORD_STRENGTH()` function:

```
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('weak');
+-----------------------------------+
| VALIDATE_PASSWORD_STRENGTH('weak') |
+-----------------------------------+
|                                25 |
+-----------------------------------+
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('lessweak$_@123');
+-----------------------------------------------+
| VALIDATE_PASSWORD_STRENGTH('lessweak$_@123') |
+-----------------------------------------------+
|                                            50 |
+-----------------------------------------------+
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('N0Tweak$_@123!');
+-------------------------------------------+
| VALIDATE_PASSWORD_STRENGTH('N0Tweak$_@123!') |
+-------------------------------------------+
|                                       100 |
+-------------------------------------------+
```

To configure password checking, modify the system variables having names of the form `validate_password.xxx`; these are the parameters that control password policy. See Section 6.4.3.2, “Password Validation Options and Variables”.

If `validate_password` is not installed, the `validate_password.xxx` system variables are not available, passwords in statements are not checked, and the `VALIDATE_PASSWORD_STRENGTH()` function always returns 0. For example, without the plugin installed, accounts can be assigned passwords shorter than 8 characters, or no password at all.

Assuming that `validate_password` is installed, it implements three levels of password checking: `LOW`, `MEDIUM`, and `STRONG`. The default is `MEDIUM`; to change this, modify the value of `validate_password.policy`. The policies implement increasingly strict password tests. The following descriptions refer to default parameter values, which can be modified by changing the appropriate system variables.

- `LOW` policy tests password length only. Passwords must be at least 8 characters long. To change this length, modify `validate_password.length`.
- `MEDIUM` policy adds the conditions that passwords must contain at least 1 numeric character, 1 lowercase character, 1 uppercase character, and 1 special (nonalphanumeric) character. To change these values, modify `validate_password.number_count`, `validate_password.mixed_case_count`, and `validate_password.special_char_count`.
- `STRONG` policy adds the condition that password substrings of length 4 or longer must not match words in the dictionary file, if one has been specified. To specify the dictionary file, modify `validate_password.dictionary_file`.

In addition, `validate_password` supports the capability of rejecting passwords that match the user name part of the effective user account for the current session, either forward or in reverse. To provide control over this capability, `validate_password` exposes a `validate_password.check_user_name` system variable, which is enabled by default.

6.4.3.1 Password Validation Component Installation and Uninstallation

This section describes how to install and uninstall the `validate_password` password-validation component. For general information about installing and uninstalling components, see Section 5.5, “MySQL Server Components”.

Note

If you install MySQL 8.0 using the MySQL Yum repository, MySQL SLES Repository, or RPM packages provided by Oracle, the `validate_password` component is enabled by default after you start your MySQL Server for the first time.

Upgrades to MySQL 8.0 from 5.7 using Yum or RPM packages leave the `validate_password` plugin in place. To make the transition from the `validate_password` plugin to the `validate_password` component, see Section 6.4.3.3, “Transitioning to the Password Validation Component”.

To be usable by the server, the component library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To install the `validate_password` component, use this statement:

```
INSTALL COMPONENT 'file://component_validate_password';
```

Component installation is a one-time operation that need not be done per server startup. `INSTALL COMPONENT` loads the component, and also registers it in the `mysql.component` system table to cause it to be loaded during subsequent server startups.

To uninstall the `validate_password` component, use this statement:

```
UNINSTALL COMPONENT 'file://component_validate_password';
```

`UNINSTALL COMPONENT` unloads the component, and deregisters it from the `mysql.component` system table to cause it not to be loaded during subsequent server startups.

6.4.3.2 Password Validation Options and Variables

This section describes the system and status variables that `validate_password` provides to enable its operation to be configured and monitored.

- Password Validation Component System Variables
- Password Validation Component Status Variables
- Password Validation Plugin Options
- Password Validation Plugin System Variables
- Password Validation Plugin Status Variables

Password Validation Component System Variables

If the `validate_password` component is enabled, it exposes several system variables that enable configuration of password checking:

```
mysql> SHOW VARIABLES LIKE 'validate_password.%';
+--------------------------------------+--------+
| Variable_name                        | Value  |
+--------------------------------------+--------+
| validate_password.check_user_name    | ON     |
| validate_password.dictionary_file    |        |
| validate_password.length             | 8      |
| validate_password.mixed_case_count   | 1      |
| validate_password.number_count       | 1      |
| validate_password.policy             | MEDIUM |
| validate_password.special_char_count | 1      |
+--------------------------------------+--------+
```

To change how passwords are checked, you can set these system variables at server startup or at runtime. The following list describes the meaning of each variable.

- `validate_password.check_user_name`

Property	Value
Command-Line Format	`--validate-password.check-user-name[={OFF|ON}]`
Introduced	8.0.4
System Variable	`validate_password.check_user_name`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether `validate_password` compares passwords to the user name part of the effective user account for the current session and rejects them if they match. This variable is unavailable unless `validate_password` is installed.

By default, `validate_password.check_user_name` is enabled. This variable controls user name matching independent of the value of `validate_password.policy`.

When `validate_password.check_user_name` is enabled, it has these effects:

- Checking occurs in all contexts for which `validate_password` is invoked, which includes use of statements such as `ALTER USER` or `SET PASSWORD` to change the current user's password, and invocation of functions such as `VALIDATE_PASSWORD_STRENGTH()`.
- The user names used for comparison are taken from the values of the `USER()` and `CURRENT_USER()` functions for the current session. An implication is that a user who has sufficient privileges to set another user's password can set the password to that user's name, and cannot set that user's password to the name of the user executing the statement. For example, `'root'@'localhost'` can set the password for `'jeffrey'@'localhost'` to `'jeffrey'`, but cannot set the password to `'root'`.
- Only the user name part of the `USER()` and `CURRENT_USER()` function values is used, not the host name part. If a user name is empty, no comparison occurs.
- If a password is the same as the user name or its reverse, a match occurs and the password is rejected.
- User-name matching is case sensitive. The password and user name values are compared as binary strings on a byte-by-byte basis.
- If a password matches the user name, `VALIDATE_PASSWORD_STRENGTH()` returns 0 regardless of how other `validate_password` system variables are set.

- `validate_password.dictionary_file`

Property	Value
Command-Line Format	`--validate-password.dictionary-file=file_name`
Introduced	8.0.4
System Variable	`validate_password.dictionary_file`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name

The path name of the dictionary file that `validate_password` uses for checking passwords. This variable is unavailable unless `validate_password` is installed.

By default, this variable has an empty value and dictionary checks are not performed. For dictionary checks to occur, the variable value must be nonempty. If the file is named as a relative path, it is interpreted relative to the server data directory. File contents should be lowercase, one word per line. Contents are treated as having a character set of `utf8`. The maximum permitted file size is 1MB.

For the dictionary file to be used during password checking, the password policy must be set to 2 (`STRONG`); see the description of the `validate_password.policy` system variable. Assuming that is true, each substring of the password of length 4 up to 100 is compared to the words in the dictionary file. Any match causes the password to be rejected. Comparisons are not case sensitive.

For `VALIDATE_PASSWORD_STRENGTH()`, the password is checked against all policies, including `STRONG`, so the strength assessment includes the dictionary check regardless of the `validate_password.policy` value.

`validate_password.dictionary_file` can be set at runtime and assigning a value causes the named file to be read without a server restart.

- `validate_password.length`

Property	Value
Command-Line Format	`--validate-password.length=#`
Introduced	8.0.4
System Variable	`validate_password.length`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`8`
Minimum Value	`0`

The minimum number of characters that `validate_password` requires passwords to have. This variable is unavailable unless `validate_password` is installed.

The `validate_password.length` minimum value is a function of several other related system variables. The value cannot be set less than the value of this expression:

```
validate_password.number_count
+ validate_password.special_char_count
+ (2 * validate_password.mixed_case_count)
```

If `validate_password` adjusts the value of `validate_password.length` due to the preceding constraint, it writes a message to the error log.

- `validate_password.mixed_case_count`

Property	Value
Command-Line Format	`--validate-password.mixed-case-count=#`
Introduced	8.0.4
System Variable	`validate_password.mixed_case_count`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`0`

The minimum number of lowercase and uppercase characters that `validate_password` requires passwords to have if the password policy is `MEDIUM` or stronger. This variable is unavailable unless `validate_password` is installed.

For a given `validate_password.mixed_case_count` value, the password must have that many lowercase characters, and that many uppercase characters.

- `validate_password.number_count`

Property	Value
Command-Line Format	`--validate-password.number-count=#`
Introduced	8.0.4
System Variable	`validate_password.number_count`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`0`

The minimum number of numeric (digit) characters that `validate_password` requires passwords to have if the password policy is `MEDIUM` or stronger. This variable is unavailable unless `validate_password` is installed.

- `validate_password.policy`

Property	Value
Command-Line Format	`--validate-password.policy=value`
Introduced	8.0.4
System Variable	`validate_password.policy`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`1`

Valid Values	`0`
	`1`
	`2`

The password policy enforced by `validate_password`. This variable is unavailable unless `validate_password` is installed.

`validate_password.policy` affects how `validate_password` uses its other policy-setting system variables, except for checking passwords against user names, which is controlled independently by `validate_password.check_user_name`.

The `validate_password.policy` value can be specified using numeric values 0, 1, 2, or the corresponding symbolic values `LOW`, `MEDIUM`, `STRONG`. The following table describes the tests performed for each policy. For the length test, the required length is the value of the

`validate_password.length` system variable. Similarly, the required values for the other tests are given by other `validate_password.xxx` variables.

Policy	Tests Performed
`0` or `LOW`	Length
`1` or `MEDIUM`	Length; numeric, lowercase/uppercase, and special characters
`2` or `STRONG`	Length; numeric, lowercase/uppercase, and special characters; dictionary file

- `validate_password.special_char_count`

Property	Value
Command-Line Format	`--validate-password.special-char-count=#`
Introduced	8.0.4
System Variable	`validate_password.special_char_count`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`0`

The minimum number of nonalphanumeric characters that `validate_password` requires passwords to have if the password policy is `MEDIUM` or stronger. This variable is unavailable unless `validate_password` is installed.

Password Validation Component Status Variables

If the `validate_password` component is enabled, it exposes status variables that provide operational information:

```
mysql> SHOW STATUS LIKE 'validate_password.%';
+----------------------------------------------+---------------------+
| Variable_name                                | Value               |
+----------------------------------------------+---------------------+
| validate_password.dictionary_file_last_parsed | 2018-01-15 08:33:49 |
| validate_password.dictionary_file_words_count | 1902                |
+----------------------------------------------+---------------------+
```

The following list describes the meaning of each status variable.

- `validate_password.dictionary_file_last_parsed`

  When the dictionary file was last parsed. This variable is unavailable unless `validate_password` is installed.

- `validate_password.dictionary_file_words_count`

  The number of words read from the dictionary file. This variable is unavailable unless `validate_password` is installed.

Password Validation Plugin Options

Note

In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. The `validate_password` plugin is deprecated and will be removed in a future version of MySQL. Consequently, its options are also deprecated and will be removed. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.4.3.3, "Transitioning to the Password Validation Component".

To control activation of the `validate_password` plugin, use this option:

- `--validate-password[=value]`

Property	Value
Command-Line Format	`--validate-password[=value]`
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`FORCE`
	`FORCE_PLUS_PERMANENT`

This option controls how the server loads the deprecated `validate_password` plugin at startup. The value should be one of those available for plugin-loading options, as described in Section 5.6.1, "Installing and Uninstalling Plugins". For example, `--validate-password=FORCE_PLUS_PERMANENT` tells the server to load the plugin at startup and prevents it from being removed while the server is running.

This option is available only if the `validate_password` plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load-add`. See Section 6.4.3.1, "Password Validation Component Installation and Uninstallation".

Password Validation Plugin System Variables

Note

In MySQL 8.0, the `validate_password` plugin was reimplemented as the `validate_password` component. The `validate_password` plugin is deprecated and will be removed in a future version of MySQL. Consequently, its system variables are also deprecated and will be removed. Use the corresponding system variables of the `validate_password` component; see Password Validation Component System Variables. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.4.3.3, "Transitioning to the Password Validation Component".

- `validate_password_check_user_name`

Property	Value
Command-Line Format	`--validate-password-check-user-name[={OFF|ON}]`
System Variable	`validate_password_check_user_name`

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

This `validate_password` plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding `validate_password.check_user_name` system variable of the `validate_password` component instead.

- `validate_password_dictionary_file`

Property	Value
Command-Line Format	`--validate-password-dictionary-file=file_name`
System Variable	`validate_password_dictionary_file`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name

This `validate_password` plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding `validate_password.dictionary_file` system variable of the `validate_password` component instead.

- `validate_password_length`

Property	Value
Command-Line Format	`--validate-password-length=#`
System Variable	`validate_password_length`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`8`
Minimum Value	`0`

This `validate_password` plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding `validate_password.length` system variable of the `validate_password` component instead.

- `validate_password_mixed_case_count`

Property	Value
Command-Line Format	`--validate-password-mixed-case-count=#`
System Variable	`validate_password_mixed_case_count`

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1
Minimum Value	0

This validate_password plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding validate_password.mixed_case_count system variable of the validate_password component instead.

- validate_password_number_count

Property	Value
Command-Line Format	--validate-password-number-count=#
System Variable	validate_password_number_count
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1
Minimum Value	0

This validate_password plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding validate_password.number_count system variable of the validate_password component instead.

- validate_password_policy

Property	Value
Command-Line Format	--validate-password-policy=value
System Variable	validate_password_policy
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	1
Valid Values	0 1 2

This validate_password plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding validate_password.policy system variable of the validate_password component instead.

- validate_password_special_char_count

Property	Value
Command-Line Format	--validate-password-special-char-count=#
System Variable	validate_password_special_char_count
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1
Minimum Value	0

This validate_password plugin system variable is deprecated and will be removed in a future version of MySQL. Use the corresponding validate_password.special_char_count system variable of the validate_password component instead.

Password Validation Plugin Status Variables

Note

In MySQL 8.0, the validate_password plugin was reimplemented as the validate_password component. The validate_password plugin is deprecated and will be removed in a future version of MySQL. Consequently, its status variables are also deprecated and will be removed. Use the corresponding status variables of the validate_password component; see Password Validation Component Status Variables. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.4.3.3, "Transitioning to the Password Validation Component".

- validate_password_dictionary_file_last_parsed

This validate_password plugin status variable is deprecated and will be removed in a future version of MySQL. Use the corresponding validate_password.dictionary_file_last_parsed status variable of the validate_password component instead.

- validate_password_dictionary_file_words_count

This validate_password plugin status variable is deprecated and will be removed in a future version of MySQL. Use the corresponding validate_password.dictionary_file_words_count status variable of the validate_password component instead.

6.4.3.3 Transitioning to the Password Validation Component

Note

In MySQL 8.0, the validate_password plugin was reimplemented as the validate_password component. The validate_password plugin is deprecated and will be removed in a future version of MySQL.

MySQL installations that currently use the `validate_password` plugin should make the transition to using the `validate_password` component instead. To do so, use the following procedure. The procedure installs the component before uninstalling the plugin, to avoid having a time window during which no password validation occurs. (The component and plugin can be installed simultaneously. In this case, the server attempts to use the component, falling back to the plugin if the component is unavailable.)

1. Install the `validate_password` component:

   ```
   INSTALL COMPONENT 'file://component_validate_password';
   ```

2. Test the `validate_password` component to ensure that it works as expected. If you need to set any `validate_password.xxx` system variables, you can do so at runtime using `SET GLOBAL`. (Any option file changes that must be made are performed in the next step.)

3. Adjust any references to the plugin system and status variables to refer to the corresponding component system and status variables. Suppose that you configure the plugin at startup using an option file like this:

   ```
   [mysqld]
   validate-password=FORCE_PLUS_PERMANENT
   validate_password_dictionary_file=/usr/share/dict/words
   validate_password_length=10
   validate_password_number_count=2
   ```

   To adjust the option file, omit the `--validate-password` option (it applies only to the plugin, not the component), and modify the system variable references:

   ```
   [mysqld]
   validate_password.dictionary_file=/usr/share/dict/words
   validate_password.length=10
   validate_password.number_count=2
   ```

   Similar adjustments are needed for applications that refer at runtime to `validate_password` plugin system and status variables.

4. Uninstall the `validate_password` plugin:

   ```
   UNINSTALL PLUGIN validate_password;
   ```

   If the `validate_password` plugin is loaded at server startup using a `--plugin-load` or `--plugin-load-add` option, omit that option from the server startup procedure. For example, if the option is listed in a server option file, remove it from the file.

5. Restart the server.

6.4.4 The MySQL Keyring

MySQL Server supports a keyring that enables internal server components and plugins to securely store sensitive information for later retrieval. The implementation is plugin-based:

- The `keyring_file` plugin stores keyring data in a file local to the server host. This plugin is available in all MySQL distributions, Community Edition and Enterprise Edition included. See Section 6.4.4.2, “Using the keyring_file File-Based Plugin”.

- The `keyring_encrypted_file` plugin stores keyring data in an encrypted file local to the server host. This plugin is available in MySQL Enterprise Edition distributions. See Section 6.4.4.3, “Using the keyring_encrypted_file Keyring Plugin”.
- `keyring_okv` is a KMIP 1.1 plugin for use with KMIP-compatible back end keyring storage products such as Oracle Key Vault and Gemalto SafeNet KeySecure Appliance. This plugin is available in MySQL Enterprise Edition distributions. See Section 6.4.4.4, “Using the keyring_okv KMIP Plugin”.
- The `keyring_aws` plugin communicates with the Amazon Web Services Key Management Service for key generation and uses a local file for key storage. This plugin is available in MySQL Enterprise Edition distributions. See Section 6.4.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”.
- A MySQL server operational mode enables migration of keys between underlying keyring keystores. This enables DBAs to switch a MySQL installation from one keyring plugin to another. See Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”.
- An SQL interface for keyring key management is implemented as a set of user-defined functions (UDFs). See Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”.
- In MySQL 8.0.16 and higher, the `keyring_keys` table exposes metadata for keys in the keyring. Key metadata includes key IDs, key owners, and backend key IDs. The `keyring_keys` table does not expose any sensitive keyring data such as key contents. See Section 26.12.18.2, “The keyring_keys table”.

Warning

The `keyring_file` and `keyring_encrypted_file` plugins for encryption key management are not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

Uses for the keyring within MySQL include:

- The `InnoDB` storage engine uses the keyring to store its key for tablespace encryption. `InnoDB` can use any supported keyring plugin.
- MySQL Enterprise Audit uses the keyring to store the audit log file encryption password. The audit log plugin can use any supported keyring plugin.
- When binary log encryption has been activated for a MySQL server (by setting `binlog_encryption=ON`), the binary log encryption keys used to encrypt the file passwords for the binary log files and relay log files are stored in the keyring. Any supported keyring plugin can be used to store binary log encryption keys. Binary log encryption keys are retained as long as there are files on the server that were encrypted using them. When the binary log master key is rotated manually, all binary log encryption keys that no longer apply to any retained binary log files or relay log files are cleared from the keyring. If a retained binary log file or relay log file cannot be initialized for re-encryption, the relevant binary log encryption keys are not deleted in case the files can be recovered in the future. For example, this might be the case if a file listed in a binary log index file is currently unreadable, or if a channel fails to initialize. For more information, see Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”.

For general keyring installation instructions, see Section 6.4.4.1, “Keyring Plugin Installation”. For information specific to a given keyring plugin, see the section describing that plugin.

For information about using the keyring UDFs, see Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”.

Keyring plugins and UDFs access a keyring service that provides the interface for server components to the keyring. For information about accessing the keyring plugin service and writing keyring plugins, see Section 29.3.2, “The Keyring Service”, and Section 29.2.4.12, “Writing Keyring Plugins”.

6.4.4.1 Keyring Plugin Installation

Keyring service consumers require a keyring plugin to be installed. MySQL provides these plugin choices:

- `keyring_file`: A plugin that stores keyring data in a file local to the server host. Available in all MySQL distributions.
- `keyring_encrypted_file`: A plugin that stores keyring data in an encrypted file local to the server host. Available in MySQL Enterprise Edition distributions.
- `keyring_okv`: A plugin that uses KMIP-compatible back end keyring storage products such as Oracle Key Vault and Gemalto SafeNet KeySecure Appliance. Available in MySQL Enterprise Edition distributions.
- `keyring_aws`: A plugin that communicates with the Amazon Web Services Key Management Service as a back end for key generation and uses a local file for key storage. Available in MySQL Enterprise Edition distributions.

This section describes how to install the keyring plugin of your choosing. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

If you intend to use keyring user-defined functions (UDFs) in conjunction with the keyring plugin, install the UDFs following keyring installation using the instructions in Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

Installation for each keyring plugin is similar. The following instructions use `keyring_file`. Users of a different keyring plugin can substitute its name for `keyring_file`.

The `keyring_file` plugin library file base name is `keyring_file`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

Note

Only one keyring plugin should be enabled at a time. Enabling multiple keyring plugins is unsupported and results may not be as anticipated.

The keyring plugin must be loaded early during the server startup sequence so that server components can access it as necessary during their own initialization. For example, the `InnoDB` storage engine uses the keyring for tablespace encryption, so the keyring plugin must be loaded and available prior to `InnoDB` initialization.

To load the plugin, use the `--early-plugin-load` option to name the plugin library file that contains it. For example, on platforms where the plugin library file suffix is `.so`, use these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
early-plugin-load=keyring_file.so
```

Before starting the server, check the notes for your chosen keyring plugin to see whether it permits or requires additional configuration:

- For `keyring_file`: Section 6.4.4.2, “Using the keyring_file File-Based Plugin”.
- For `keyring_okv`: Section 6.4.4.4, “Using the keyring_okv KMIP Plugin”.
- For `keyring_aws`: Section 6.4.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”

After performing any plugin-specific configuration, verify plugin installation. With the MySQL server running, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'keyring%';
+--------------+---------------+
| PLUGIN_NAME  | PLUGIN_STATUS |
+--------------+---------------+
| keyring_file | ACTIVE        |
+--------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

If no keyring plugin is available when a server component tries to access the keyring service, the service cannot be used by that component. As a result, the component may fail to initialize or may initialize with limited functionality. For example, if `InnoDB` finds that there are encrypted tablespaces when it initializes, it attempts to access the keyring. If the keyring is unavailable, `InnoDB` can access only unencrypted tablespaces. To ensure that `InnoDB` can access encrypted tablespaces as well, use `--early-plugin-load` to load the keyring plugin.

Plugins can be loaded by other methods, such as the `--plugin-load` or `--plugin-load-add` option or the `INSTALL PLUGIN` statement. However, keyring plugins loaded using those methods may be available too late in the server startup sequence for certain server components, such as `InnoDB`:

- Plugin loading using `--plugin-load` or `--plugin-load-add` occurs after `InnoDB` initialization.
- Plugins installed using `INSTALL PLUGIN` are registered in the `mysql.plugin` system table and loaded automatically for subsequent server restarts. However, because `mysql.plugin` is an `InnoDB` table, any plugins named in it can be loaded during startup only after `InnoDB` initialization.

6.4.4.2 Using the keyring_file File-Based Plugin

The `keyring_file` plugin is a keyring plugin that stores keyring data in a file local to the server host.

Warning

The `keyring_file` plugin for encryption key management is not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

To install the `keyring_file` plugin, use the general keyring installation instructions found in Section 6.4.4.1, “Keyring Plugin Installation”, together with the configuration information specific to `keyring_file` found here.

To be usable during the server startup process, `keyring_file` must be loaded using the `--early-plugin-load` option. The `keyring_file_data` system variable optionally configures the location of

the file used by the `keyring_file` plugin for data storage. The default value is platform specific. To configure the file location explicitly, set the variable value at startup. For example, use these lines in the server `my.cnf` file (adjust the `.so` suffix and file location for your platform as necessary):

```
[mysqld]
early-plugin-load=keyring_file.so
keyring_file_data=/usr/local/mysql/mysql-keyring/keyring
```

Keyring operations are transactional: The `keyring_file` plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the `keyring_file_data` system variable with a suffix of `.backup`.

For additional information about `keyring_file_data`, see Section 6.4.4.11, “Keyring System Variables”.

To ensure that keys are flushed only when the correct keyring storage file exists, `keyring_file` stores a SHA-256 checksum of the keyring in the file. Before updating the file, the plugin verifies that it contains the expected checksum.

The `keyring_file` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the user-defined functions (UDFs) described in Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 29.3.2, “The Keyring Service”.

Example (using UDFs):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the key types permitted by `keyring_file`, see Section 6.4.4.7, “Supported Keyring Key Types and Lengths”.

6.4.4.3 Using the keyring_encrypted_file Keyring Plugin

Note

The `keyring_encrypted_file` plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The `keyring_encrypted_file` plugin is a keyring plugin that stores keyring data in an encrypted file local to the server host.

Warning

The `keyring_encrypted_file` plugin for encryption key management is not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

To install the `keyring_encrypted_file` plugin, use the general keyring installation instructions found in Section 6.4.4.1, “Keyring Plugin Installation”, together with the configuration information specific to `keyring_encrypted_file` found here.

To be usable during the server startup process, `keyring_encrypted_file` must be loaded using the `--early-plugin-load` option. To specify the password for encrypting the keyring data file, set the `keyring_encrypted_file_password` system variable. (The password is mandatory; if not specified at server startup, `keyring_encrypted_file` initialization fails.) The `keyring_encrypted_file_data` system variable optionally configures the location of the file used by the `keyring_encrypted_file` plugin for data storage. The default value is platform specific. To configure the file location explicitly, set the variable value at startup. For example, use these lines in the server `my.cnf` file (adjust the `.so` suffix and file location for your platform as necessary and substitute your chosen password):

```
[mysqld]
early-plugin-load=keyring_encrypted_file.so
keyring_encrypted_file_data=/usr/local/mysql/mysql-keyring/keyring-encrypted
keyring_encrypted_file_password=password
```

Because the `my.cnf` file stores a password when written as shown, it should have a restrictive mode and be accessible only to the account used to run the MySQL server.

Keyring operations are transactional: The `keyring_encrypted_file` plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the `keyring_encrypted_file_data` system variable with a suffix of `.backup`.

For additional information about the system variables used to configure the `keyring_encrypted_file` plugin, see Section 6.4.4.11, “Keyring System Variables”.

To ensure that keys are flushed only when the correct keyring storage file exists, `keyring_encrypted_file` stores a SHA-256 checksum of the keyring in the file. Before updating the file, the plugin verifies that it contains the expected checksum. In addition, `keyring_encrypted_file` encrypts file contents using AES before writing the file, and decrypts file contents after reading the file.

The `keyring_encrypted_file` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the user-defined functions (UDFs) described in Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 29.3.2, “The Keyring Service”.

Example (using UDFs):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the key types permitted by `keyring_encrypted_file`, see Section 6.4.4.7, “Supported Keyring Key Types and Lengths”.

6.4.4.4 Using the keyring_okv KMIP Plugin

Note

The `keyring_okv` plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The Key Management Interoperability Protocol (KMIP) enables communication of cryptographic keys between a key management server and its clients. The `keyring_okv` keyring plugin uses the KMIP 1.1

protocol to communicate securely as a client of a KMIP back end. Keyring material is generated exclusively by the back end, not by `keyring_okv`. The plugin works with these KMIP-compatible products:

- Oracle Key Vault
- Gemalto SafeNet KeySecure Appliance

The `keyring_okv` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by those functions are accessible at two levels:

- SQL interface: In SQL statements, call the user-defined functions (UDFs) described in Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”.
- C interface: In C-language code, call the keyring service functions described in Section 29.3.2, “The Keyring Service”.

Example (using UDFs):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

For information about the key types permitted by `keyring_okv`, Section 6.4.4.7, “Supported Keyring Key Types and Lengths”.

To install the `keyring_okv` plugin, use the general keyring installation instructions found in Section 6.4.4.1, “Keyring Plugin Installation”, together with the configuration information specific to `keyring_okv` found here.

- General keyring_okv Configuration
- Configuring keyring_okv for Oracle Key Vault
- Configuring keyring_okv for Gemalto SafeNet KeySecure Appliance
- Password-Protecting the keyring_okv Key File

General keyring_okv Configuration

Regardless of which KMIP back end the `keyring_okv` plugin uses for keyring storage, the `keyring_okv_conf_dir` system variable configures the location of the directory used by `keyring_okv` for its support files. The default value is empty, so you must set the variable to name a properly configured directory before the plugin can communicate with the KMIP back end. Unless you do so, `keyring_okv` writes a message to the error log during server startup that it cannot communicate:

```
[Warning] Plugin keyring_okv reported: 'For keyring_okv to be
initialized, please point the keyring_okv_conf_dir variable to a directory
containing Oracle Key Vault configuration file and ssl materials'
```

The `keyring_okv_conf_dir` variable must name a directory that contains the following items:

- `okvclient.ora`: A file that contains details of the KMIP back end with which `keyring_okv` will communicate.
- `ssl`: A directory that contains the certificate and key files required to establish a secure connection with the KMIP back end: `CA.pem`, `cert.pem`, and `key.pem`. If the key file is password-protected, the `ssl` directory can contain a single-line text file named `password.txt` containing the password needed to decrypt the key file.

Both the `okvclient.ora` file and `ssl` directory with the certificate and key files are required for `keyring_okv` to work properly. The procedure used to populate the configuration directory with these files depends on the KMIP back end used with `keyring_okv`, as described elsewhere.

The configuration directory used by `keyring_okv` as the location for its support files should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the `/usr/local/mysql/mysql-keyring-okv` directory, the following commands (executed as `root`) create the directory and set its mode and ownership:

```
cd /usr/local/mysql
mkdir mysql-keyring-okv
chmod 750 mysql-keyring-okv
chown mysql mysql-keyring-okv
chgrp mysql mysql-keyring-okv
```

To be usable during the server startup process, `keyring_okv` must be loaded using the `--early-plugin-load` option. Also, set the `keyring_okv_conf_dir` system variable to tell `keyring_okv` where to find its configuration directory. For example, use these lines in the server `my.cnf` file (adjust the `.so` suffix and directory location for your platform as necessary):

```
[mysqld]
early-plugin-load=keyring_okv.so
keyring_okv_conf_dir=/usr/local/mysql/mysql-keyring-okv
```

For additional information about `keyring_okv_conf_dir`, see Section 6.4.4.11, “Keyring System Variables”.

Configuring keyring_okv for Oracle Key Vault

The discussion here assumes that you are familiar with Oracle Key Vault. Some pertinent information sources:

- Oracle Key Vault site
- Oracle Key Vault documentation

In Oracle Key Vault terminology, clients that use Oracle Key Vault to store and retrieve security objects are called endpoints. To communicate with Oracle Key Vault, it is necessary to register as an endpoint and enroll by downloading and installing endpoint support files.

The following procedure briefly summarizes the process of setting up `keyring_okv` for use with Oracle Key Vault:

1. Create the configuration directory for the `keyring_okv` plugin to use.
2. Register an endpoint with Oracle Key Vault to obtain an enrollment token.
3. Use the enrollment token to obtain the `okvclient.jar` client software download.
4. Install the client software to populate the `keyring_okv` configuration directory that contains the Oracle Key Vault support files.

Use the following procedure to configure `keyring_okv` and Oracle Key Vault to work together. This description only summarizes how to interact with Oracle Key Vault. For details, visit the Oracle Key Vault site and consult the Oracle Key Vault Administrator's Guide.

1. Create the configuration directory that will contain the Oracle Key Vault support files, and make sure that the `keyring_okv_conf_dir` system variable is set to name that directory (for details, see General keyring_okv Configuration).
2. Log in to the Oracle Key Vault management console as a user who has the System Administrator role.
3. Select the Endpoints tab to arrive at the Endpoints page. On the Endpoints page, click Add.
4. Provide the required endpoint information and click Register. The endpoint type should be Other. Successful registration results in an enrollment token.
5. Log out from the Oracle Key Vault server.
6. Connect again to the Oracle Key Vault server, this time without logging in. Use the endpoint enrollment token to enroll and request the `okvclient.jar` software download. Save this file to your system.
7. Install the `okvclient.jar` file using the following command (you must have JDK 1.4 or higher):

   ```
   java -jar okvclient.jar -d dir_name [-v]
   ```

   The directory name following the `-d` option is the location in which to install extracted files. The `-v` option, if given, causes log information to be produced that may be useful if the command fails.

   When the command asks for an Oracle Key Vault endpoint password, do not provide one. Instead, press Enter. (The result is that no password will be required when the endpoint connects to Oracle Key Vault.)
8. The preceding command produces an `okvclient.ora` file, which should be in this location under the directory named by the `-d` option in the preceding `java -jar` command:

   ```
   install_dir/conf/okvclient.ora
   ```

   The file contents include lines that look something like this:

   ```
   SERVER=host_ip:port_num
   STANDBY_SERVER=host_ip:port_num
   ```

   The `keyring_okv` plugin attempts to communicate with the server running on the host named by the `SERVER` variable and falls back to `STANDBY_SERVER` if that fails:

   - For the `SERVER` variable, a setting in the `okvclient.ora` file is mandatory.
   - For the `STANDBY_SERVER` variable, a setting in the `okvclient.ora` file is optional.
9. Go to the Oracle Key Vault installer directory and test the setup by running this command:

   ```
   okvutil/bin/okvutil list
   ```

   The output should look something like this:

   ```
   Unique ID                            Type           Identifier
   255AB8DE-C97F-482C-E053-0100007F28B9 Symmetric Key -
   264BF6E0-A20E-7C42-E053-0100007FB29C Symmetric Key -
   ```

   For a fresh Oracle Key Vault server (a server without any key in it), the output looks like this instead, to indicate that there are no keys in the vault:

```
no objects found
```

10. Use this command to extract the `ssl` directory containing SSL materials from the `okvclient.jar` file:

    ```
    jar xf okvclient.jar ssl
    ```

11. Copy the Oracle Key Vault support files (the `okvclient.ora` file and the `ssl` directory) into the configuration directory.

12. (Optional) If you wish to password-protect the key file, use the instructions in Password-Protecting the keyring_okv Key File.

After completing the preceding procedure, restart the MySQL server. It loads the `keyring_okv` plugin and `keyring_okv` uses the files in its configuration directory to communicate with Oracle Key Vault.

Configuring keyring_okv for Gemalto SafeNet KeySecure Appliance

Gemalto SafeNet KeySecure Appliance uses the KMIP protocol (version 1.1 or 1.2). The `keyring_okv` keyring plugin (which supports KMIP 1.1) can use KeySecure as its KMIP back end for keyring storage.

Use the following procedure to configure `keyring_okv` and KeySecure to work together. The description only summarizes how to interact with KeySecure. For details, consult the section named Add a KMIP Server in the KeySecure User Guide.

1. Create the configuration directory that will contain the KeySecure support files, and make sure that the `keyring_okv_conf_dir` system variable is set to name that directory (for details, see General keyring_okv Configuration).

2. In the configuration directory, create a subdirectory named `ssl` to use for storing the required SSL certificate and key files.

3. In the configuration directory, create a file named `okvclient.ora`. It should have following format:

    ```
    SERVER=host_ip:port_num
    STANDBY_SERVER=host_ip:port_num
    ```

    For example, if KeySecure is running on host 198.51.100.20 and listening on port 9002, the `okvclient.ora` file looks like this:

    ```
    SERVER=198.51.100.20:9002
    STANDBY_SERVER=198.51.100.20:9002
    ```

4. Connect to the KeySecure Management Console as an administrator with credentials for Certificate Authorities access.

5. Navigate to Security >> Local CAs and create a local certificate authority (CA).

6. Go to Trusted CA Lists. Select Default and click on Properties. Then select Edit for Trusted Certificate Authority List and add the CA just created.

7. Download the CA and save it in the `ssl` directory as a file named `CA.pem`.

8. Navigate to Security >> Certificate Requests and create a certificate. Then you will be able to download a compressed `tar` file containing certificate PEM files.

9. Extract the PEM files from in the downloaded file. For example, if the file name is `csr_w_pk_pkcs8.gz`, decompress and unpack it using this command:

   ```
   tar zxvf csr_w_pk_pkcs8.gz
   ```

   Two files result from the extraction operation: `certificate_request.pem` and `private_key_pkcs8.pem`.

10. Use this `openssl` command to decrypt the private key and create a file named `key.pem`:

    ```
    openssl pkcs8 -in private_key_pkcs8.pem -out key.pem
    ```

11. Copy the `key.pem` file into the `ssl` directory.

12. Copy the certificate request in `certificate_request.pem` into the clipboard.

13. Navigate to Security >> Local CAs. Select the same CA that you created earlier (the one you downloaded to create the `CA.pem` file), and click Sign Request. Paste the Certificate Request from the clipboard, choose a certificate purpose of Client (the keyring is a client of KeySecure), and click Sign Request. The result is a certificate signed with the selected CA in a new page.

14. Copy the signed certificate to the clipboard, then save the clipboard contents as a file named `cert.pem` in the `ssl` directory.

15. (Optional) If you wish to password-protect the key file, use the instructions in Password-Protecting the keyring_okv Key File.

After completing the preceding procedure, restart the MySQL server. It loads the `keyring_okv` plugin and `keyring_okv` uses the files in its configuration directory to communicate with KeySecure.

Password-Protecting the keyring_okv Key File

You can optionally protect the key file with a password and supply a file containing the password to enable the key file to be decrypted. To so do, change location to the `ssl` directory and perform these steps:

1. Encrypt the `key.pem` key file. For example, use a command like this, and enter the encryption password at the prompts:

   ```
   shell> openssl rsa -des3 -in key.pem -out key.pem.new
   Enter PEM pass phrase:
   Verifying - Enter PEM pass phrase:
   ```

2. Save the encryption password in a single-line text file named `password.txt` in the `ssl` directory.

3. Verify that the encrypted key file can be decrypted using the following command. The decrypted file should display on the console:

   ```
   shell> openssl rsa -in key.pem.new -passin file:password.txt
   ```

4. Remove the original `key.pem` file and rename `key.pem.new` to `key.pem`.

5. Change the ownership and access mode of new `key.pem` file and `password.txt` file as necessary to ensure that they have the same restrictions as other files in the `ssl` directory.

6.4.4.5 Using the keyring_aws Amazon Web Services Keyring Plugin

Note

The `keyring_aws` plugin is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

The `keyring_aws` plugin is a keyring plugin that communicates with the Amazon Web Services Key Management Service (AWS KMS) as a back end for key generation and uses a local file for key storage. All keyring material is generated exclusively by the AWS server, not by `keyring_aws`.

`keyring_aws` is available on these platforms:

- Debian 8
- EL7
- macOS 10.13 and 10.14
- SLES 12
- Ubuntu 14.04 and 16.04
- Windows

The discussion here assumes that you are familiar with AWS in general and KMS in particular. Some pertinent information sources:

- AWS site
- KMS documentation

The following sections provide configuration and usage information for the `keyring_aws` keyring plugin:

- keyring_aws Configuration
- keyring_aws Operation
- keyring_aws Credential Changes

keyring_aws Configuration

To install the `keyring_aws` plugin, use the general keyring installation instructions found in Section 6.4.4.1, “Keyring Plugin Installation”, together with the plugin-specific configuration information found here.

The plugin library file contains the `keyring_aws` plugin and two user-defined functions (UDFs), `keyring_aws_rotate_cmk()` and `keyring_aws_rotate_keys()`.

To configure `keyring_aws`, you must obtain a secret access key that provides credentials for communicating with AWS KMS and write it to a configuration file:

1. Create an AWS KMS account.
2. Use AWS KMS to create a secret access key ID and secret access key. The access key serves to verify your identity and that of your applications.
3. Use the AWS KMS account to create a customer master key (CMK) ID. At MySQL startup, set the `keyring_aws_cmk_id` system variable to the CMK ID value. This variable is mandatory and there is no default. (Its value can be changed at runtime if desired using `SET GLOBAL`.)

4. If necessary, create the directory in which the configuration file will be located. The directory should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use `/usr/local/mysql/mysql-keyring/keyring_aws_conf` as the file name, the following commands (executed as `root`) create its parent directory and set the directory mode and ownership:

   ```
   shell> cd /usr/local/mysql
   shell> mkdir mysql-keyring
   shell> chmod 750 mysql-keyring
   shell> chown mysql mysql-keyring
   shell> chgrp mysql mysql-keyring
   ```

   At MySQL startup, set the `keyring_aws_conf_file` system variable to `/usr/local/mysql/mysql-keyring/keyring_aws_conf` to indicate the configuration file location to the server.

5. Prepare the `keyring_aws` configuration file, which should contain two lines:

   - Line 1: The secret access key ID
   - Line 2: The secret access key

   For example, if the key ID is `wwwwwwwwwwwwEXAMPLE` and the key is `xxxxxxxxxxxxx/yyyyyyy/zzzzzzzzEXAMPLEKEY`, the configuration file looks like this:

   ```
   wwwwwwwwwwwwEXAMPLE
   xxxxxxxxxxxxx/yyyyyyy/zzzzzzzzEXAMPLEKEY
   ```

To be usable during the server startup process, `keyring_aws` must be loaded using the `--early-plugin-load` option. The `keyring_aws_cmk_id` system variable is mandatory and configures the customer master key (CMK) ID obtained from the AWS KMS server. The `keyring_aws_conf_file` and `keyring_aws_data_file` system variables optionally configure the locations of the files used by the `keyring_aws` plugin for configuration information and data storage. The file location variable default values are platform specific. To configure the locations explicitly, set the variable values at startup. For example, use these lines in the server `my.cnf` file (adjust the `.so` suffix and file locations for your platform as necessary):

```
[mysqld]
early-plugin-load=keyring_aws.so
keyring_aws_cmk_id='arn:aws:kms:us-west-2:111122223333:key/abcd1234-ef56-ab12-cd34-ef56abcd1234'
keyring_aws_conf_file=/usr/local/mysql/mysql-keyring/keyring_aws_conf
keyring_aws_data_file=/usr/local/mysql/mysql-keyring/keyring_aws_data
```

For the `keyring_aws` plugin to start successfully, the configuration file must exist and contain valid secret access key information, initialized as described previously. The storage file need not exist. If it does not, `keyring_aws` attempts to create it (as well as its parent directory, if necessary).

For additional information about the system variables used to configure the `keyring_aws` plugin, see Section 6.4.4.11, “Keyring System Variables”.

Start the MySQL server and install the UDFs associated with the `keyring_aws` plugin. This is a one-time operation, performed by executing the following statements (adjust the `.so` suffix for your platform as necessary):

```
CREATE FUNCTION keyring_aws_rotate_cmk RETURNS INTEGER
  SONAME 'keyring_aws.so';
CREATE FUNCTION keyring_aws_rotate_keys RETURNS INTEGER
```

```
  SONAME 'keyring_aws.so';
```

For additional information about the `keyring_aws` UDFs, see Section 6.4.4.9, “Plugin-Specific Keyring Key-Management Functions”.

keyring_aws Operation

At plugin startup, the `keyring_aws` plugin reads the AWS secret access key ID and key from its configuration file. It also reads any encrypted keys contained in its storage file into its in-memory cache.

During operation, `keyring_aws` maintains encrypted keys in the in-memory cache and uses the storage file as local persistent storage. Each keyring operation is transactional: `keyring_aws` either successfully changes both the in-memory key cache and the keyring storage file, or the operation fails and the keyring state remains unchanged.

To ensure that keys are flushed only when the correct keyring storage file exists, `keyring_aws` stores a SHA-256 checksum of the keyring in the file. Before updating the file, the plugin verifies that it contains the expected checksum.

The `keyring_aws` plugin supports the functions that comprise the standard MySQL Keyring service interface. Keyring operations performed by these functions are accessible at two levels:

- C interface: In C-language code, call the keyring service functions described in Section 29.3.2, “The Keyring Service”.
- SQL interface: In SQL statements, call the user-defined functions (UDFs) described in Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”.

Example (using UDFs):

```
SELECT keyring_key_generate('MyKey', 'AES', 32);
SELECT keyring_key_remove('MyKey');
```

In addition, the `keyring_aws_rotate_cmk()` and `keyring_aws_rotate_keys()` UDFs “extend” the keyring plugin interface to provide AWS-related capabilities not covered by the standard keyring service interface. These capabilities are accessible only by calling the UDFs. There are no corresponding C-languge key service functions.

For information about the key types permitted by `keyring_aws`, see Section 6.4.4.7, “Supported Keyring Key Types and Lengths”.

keyring_aws Credential Changes

Assuming that the `keyring_aws` plugin has initialized properly at server startup, it is possible to change the credentials used for communicating with AWS KMS:

1. Use AWS KMS to create a new secret access key ID and secret access key.
2. Store the new credentials in the configuration file (the file named by the `keyring_aws_conf_file` system variable). The file format is as described previously.
3. Reinitialize the `keyring_aws` plugin so that it rereads the configuration file. Assuming that the new credentials are valid, the plugin should initialize successfully.

   There are two ways to reinitialize the plugin:

   - Restart the server. This is simpler and has no side effects, but is not suitable for installations that require minimal server downtime with as few restarts as possible.

- Reinitialize the plugin without restarting the server by executing the following statements (adjust the `.so` suffix for your platform as necessary):

```
UNINSTALL PLUGIN keyring_aws;
INSTALL PLUGIN keyring_aws SONAME 'keyring_aws.so';
```

Note

In addition to loading a plugin at runtime, `INSTALL PLUGIN` has the side effect of registering the plugin it in the `mysql.plugin` system table. Because of this, if you decide to stop using `keyring_aws`, it is not sufficient to remove the `--early-plugin-load` option from the set of options used to start the server. That stops the plugin from loading early, but the server still attempts to load it when it gets to the point in the startup sequence where it loads the plugins registered in `mysql.plugin`.

Consequently, if you execute the `UNINSTALL PLUGIN` plus `INSTALL PLUGIN` sequence just described to change the AWS KMS credentials, then to stop using `keyring_aws`, it is necessary to execute `UNINSTALL PLUGIN` again to unregister the plugin in addition to removing the `--early-plugin-load` option.

6.4.4.6 Migrating Keys Between Keyring Keystores

The MySQL server supports an operational mode that enables migration of keys between underlying keyring keystores. This enables DBAs to switch a MySQL installation from one keyring plugin to another. A migration server (that is, a server started in key migration mode) does not accept client connections. Instead, it runs only long enough to migrate keys, then exits. A migration server reports errors to the console (the standard error output).

It is possible to perform offline or online key migration:

- If you are sure that no running server on the local host is using the source or destination keystore, an offline migration is possible. In this case, the migration server can modify the keystores without possibility of a running server modifying keystore content during the migration.
- If a running server on the local host is using the source or destination keystore, an online migration must be performed. In this case, the migration server connects to the running server and instructs it to pause keyring operations while key migration is in progress.

The result of a key migration operation is that the destination keystore contains the keys it had prior to the migration, plus the keys from the source keystore. The source keystore is the same before and after the migration because keys are copied, not moved. If a key to be copied already exists in the destination keystore, an error occurs and the destination keystore is restored to its premigration state.

The user who invokes the server in key-migration mode must not be the `root` operating system user, and must have permission to read and write the keyring files.

To perform a key migration operation, determine which key migration options are needed. Migration options indicate which keyring plugins are involved, and whether to perform an offline or online migration:

- To indicate the source and destination keyring plugins, specify these options:
  - `--keyring-migration-source`: The source keyring plugin that manages the keys to be migrated.

- `--keyring-migration-destination`: The destination keyring plugin to which the migrated keys are to be copied.

These options tell the server to run in key migration mode. Both options are mandatory for all key migration operations. The source and destination plugins must differ, and the migration server must support both plugins.

- For an offline migration, no additional key migration options are needed.

Warning

Do not perform offline migration involving a keystore that is in use by a running server.

- For an online migration, some running server currently is using the source or destination keystore. Specify the key migration options that indicate how to connect to the running server. This is necessary so that the migration server can connect to the running server and tell it to pause keyring use during the migration operation.

  Use of any of the following options signifies an online migration:

  - `--keyring-migration-host`: The host where the running server is located. This is always the local host.
  - `--keyring-migration-user`, `--keyring-migration-password`: The user name and password for the account to use to connect to the running server.
  - `--keyring-migration-port`: For TCP/IP connections, the port number to connect to on the running server.
  - `--keyring-migration-socket`: For Unix socket file or Windows named pipe connections, the socket file or named pipe to connect to on the running server.

For additional details about the key migration options, see Section 6.4.4.10, “Keyring Command Options”.

Start the migration server with the key migration options determined as just described, possibly with other options. Keep the following considerations in mind:

- Other server options might be required, such as other configuration parameters for the two keyring plugins. For example, if `keyring_file` is one of the plugins, you must set the `keyring_file_data` system variable if the keyring data file location is not the default location. Other non-keyring options may be required as well. One way to specify these options is by using `--defaults-file` to name an option file that contains the required options.
- If you invoke the migration server from a system account different from that normally used to run MySQL, it might create keyring directories or files that are inaccessible to the server during normal operation. Suppose that `mysqld` normally runs as the `mysql` operating system user, but you invoke the migration server while logged in as `isabel`. Any new directories or files created by the migration server will be owned by `isabel`. Subsequent startup will fail when a server run as the `mysql` operating system user attempts to access file system objects owned by `isabel`.

  To avoid this issue, start the migration server as the `root` operating system user and provide a `--user=`*`user_name`* option, where *`user_name`* is the system account normally used to run MySQL.
- The migration server expects path name option values to be full paths. Relative path names may not be resolved as you expect.

Example command line for offline key migration:

```
mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
  --keyring-migration-source=keyring_file.so
  --keyring-migration-destination=keyring_encrypted_file.so
  --keyring_encrypted_file_password=password
```

Example command line for online key migration:

```
mysqld --defaults-file=/usr/local/mysql/etc/my.cnf
  --keyring-migration-source=keyring_file.so
  --keyring-migration-destination=keyring_encrypted_file.so
  --keyring_encrypted_file_password=password
  --keyring-migration-host=localhost
  --keyring-migration-user=root
  --keyring-migration-password=root_password
```

The key migration server performs the migration operation as follows:

1. (Online migration only) Connect to the running server using the connection options. The account used to connect must have the privileges required to modify the global `keyring_operations` system variable (`ENCRYPTION_KEY_ADMIN` in addition to either `SYSTEM_VARIABLES_ADMIN` or `SUPER`).
2. (Online migration only) Disable `keyring_operations` on the running server. (The running server must support `keyring_operations`.)
3. Load the source and destination keyring plugins.
4. Copy keys from the source keyring to the destination keyring.
5. Unload the keyring plugins.
6. (Online migration only) Enable `keyring_operations` on the running server.
7. (Online migration only) Disconnect from the running server.
8. Exit.

If an error occurs during key migration, any keys that were copied to the destination plugin are removed, leaving the destination keystore unchanged.

Important

For an online migration operation, the migration server takes care of enabling and disabling `keyring_operations` on the running server. However, if the migration server exits abnormally (for example, if someone forcibly terminates it), it is possible that `keyring_operations` will not have been re-enabled on the running server, leaving it unable to perform keyring operations. In this case, it may be necessary to connect to the running server and enable `keyring_operations` manually.

After a successful online key migration operation, the running server might need to be restarted:

- If the running server was using the source keystore, it need not be restarted after the migration.
- If the running server was using the source keystore before the migration but should use the destination keystore after the migration, it must be reconfigured to use the destination keyring plugin and restarted.
- If the running server was using the destination keystore and will continue to use it, it should be restarted after the migration to load all keys migrated into the destination keystore.

Note

MySQL server key migration mode supports pausing a single running server. To perform a key migration if multiple key servers are using the keystores involved, use this procedure:

1. Connect to each running server manually and set `keyring_operations=OFF`.
2. Use the migration server to perform an offline key migration.
3. Connect to each running server manually and set `keyring_operations=ON`.

All running servers must support the `keyring_operations=ON` system variable.

6.4.4.7 Supported Keyring Key Types and Lengths

MySQL Keyring supports keys of different types (encryption algorithms) and lengths:

- The available key types depend on which keyring plugin is installed.
- The permitted key lengths are subject to multiple factors:
  - General keyring UDF interface limits (for keys managed using one of the keyring UDFs described in Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”), or limits from back end implementations. These length limits can vary by key operation type.
  - In addition to the general limits, individual plugins may impose restrictions on key lengths per key type.

Table 6.25, “General Keyring Key Length Limits” shows the general key length limits. (The lower limits for `keyring_aws` are imposed by the AWS KMS interface, not the keyring UDFs.) Table 6.26, “Keyring Plugin Key Types and Lengths” shows for each keyring plugin the key types it permits, as well as any plugin-specific key-length restrictions.

Table 6.25 General Keyring Key Length Limits

Key Operation	Maximum Key Length
Generate key	16,384 bytes (2,048 prior to MySQL 8.0.18); 1,024 for `keyring_aws`
Store key	16,384 bytes (2,048 prior to MySQL 8.0.18); 4,096 for `keyring_aws`
Fetch key	16,384 bytes (2,048 prior to MySQL 8.0.18); 4,096 for `keyring_aws`

Table 6.26 Keyring Plugin Key Types and Lengths

Plugin Name	Permitted Key Type	Plugin-Specific Length Restrictions
`keyring_encrypted_file`	`AES`	None
	`DSA`	None
	`RSA`	None
	`SECRET`	None
`keyring_file`	`AES`	None
	`DSA`	None
	`RSA`	None

Plugin Name	Permitted Key Type	Plugin-Specific Length Restrictions
	SECRET	None
keyring_okv	AES SECRET	16, 24, or 32 bytes None
keyring_aws	AES SECRET	16, 24, or 32 bytes None

The SECRET key type, available as of MySQL 8.0.19, is intended for general-purpose storage of sensitive data using the MySQL keyring, and is supported by all keyring plugins. The keyring encrypts and decrypts SECRET data as a byte stream upon storage and retrieval.

Example keyring operations involving the SECRET key type:

```
SELECT keyring_key_generate('MySecret1', 'SECRET', 20);
SELECT keyring_key_remove('MySecret1');

SELECT keyring_key_store('MySecret2', 'SECRET', 'MySecretData');
SELECT keyring_key_fetch('MySecret2');
SELECT keyring_key_length_fetch('MySecret2');
SELECT keyring_key_type_fetch('MySecret2');
SELECT keyring_key_remove('MySecret2');
```

6.4.4.8 General-Purpose Keyring Key-Management Functions

MySQL Server supports a keyring service that enables internal server components and plugins to securely store sensitive information for later retrieval.

MySQL Server also includes an SQL interface for keyring key management, implemented as a set of general-purpose user-defined functions (UDFs) that access the functions provided by the internal keyring service. The keyring UDFs are contained in a plugin library file, which also contains a keyring_udf plugin that must be enabled prior to UDF invocation. For these UDFs to be used, a keyring plugin such as keyring_file or keyring_okv must be enabled.

The UDFs described here are general purpose and intended for use with any keyring plugin. A given keyring plugin might have UDFs of its own that are intended for use only with that plugin; see Section 6.4.4.9, “Plugin-Specific Keyring Key-Management Functions”.

The following sections provide installation instructions for the keyring UDFs and demonstrate how to use them. For information about the keyring service functions invoked by the UDFs, see Section 29.3.2, “The Keyring Service”. For general keyring information, see Section 6.4.4, “The MySQL Keyring”.

- Installing or Uninstalling General-Purpose Keyring Functions
- Using General-Purpose Keyring Functions
- General-Purpose Keyring Function Reference

Installing or Uninstalling General-Purpose Keyring Functions

This section describes how to install or uninstall the keyring user-defined functions (UDFs), which are implemented in a plugin library file that also contains a keyring_udf plugin. For general information about installing or uninstalling plugins and UDFs, see Section 5.6.1, “Installing and Uninstalling Plugins”, and Section 5.7.1, “Installing and Uninstalling User-Defined Functions”.

The keyring UDFs enable keyring key management operations, but the `keyring_udf` plugin must also be installed because the UDFs will not work correctly without it. Attempts to use the UDFs without the `keyring_udf` plugin result in an error.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `keyring_udf`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To install the `keyring_udf` plugin and the UDFs, use the `INSTALL PLUGIN` and `CREATE FUNCTION` statements (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN keyring_udf SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_generate RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_fetch RETURNS STRING
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_length_fetch RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_type_fetch RETURNS STRING
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_store RETURNS INTEGER
  SONAME 'keyring_udf.so';
CREATE FUNCTION keyring_key_remove RETURNS INTEGER
  SONAME 'keyring_udf.so';
```

If the plugin and UDFs are used on a master replication server, install them on all slave servers as well to avoid replication issues.

Once installed as just described, the plugin and UDFs remain installed until uninstalled. To remove them, use the `UNINSTALL PLUGIN` and `DROP FUNCTION` statements:

```
UNINSTALL PLUGIN keyring_udf;
DROP FUNCTION keyring_key_generate;
DROP FUNCTION keyring_key_fetch;
DROP FUNCTION keyring_key_length_fetch;
DROP FUNCTION keyring_key_type_fetch;
DROP FUNCTION keyring_key_store;
DROP FUNCTION keyring_key_remove;
```

Using General-Purpose Keyring Functions

Before using the keyring user-defined functions (UDFs), install them according to the instructions provided in Installing or Uninstalling General-Purpose Keyring Functions.

The keyring UDFs are subject to these constraints:

- To use any keyring UDF, the `keyring_udf` plugin must be enabled. Otherwise, an error occurs:

  ```
  ERROR 1123 (HY000): Can't initialize function 'keyring_key_generate';
  This function requires keyring_udf plugin which is not installed.
  Please install
  ```

  To install the `keyring_udf` plugin, see Installing or Uninstalling General-Purpose Keyring Functions.

- The keyring UDFs invoke keyring service functions (see Section 29.3.2, “The Keyring Service”). The service functions in turn use whatever keyring plugin is installed (for example, `keyring_file` or

keyring_okv). Therefore, to use any keyring UDF, some underlying keyring plugin must be enabled. Otherwise, an error occurs:

```
ERROR 3188 (HY000): Function 'keyring_key_generate' failed because
underlying keyring service returned an error. Please check if a
keyring plugin is installed and that provided arguments are valid
for the keyring you are using.
```

To install a keyring plugin, see Section 6.4.4.1, “Keyring Plugin Installation”.

- To use any keyring UDF, a user must possess the global EXECUTE privilege. Otherwise, an error occurs:

```
ERROR 1123 (HY000): Can't initialize function 'keyring_key_generate';
The user is not privileged to execute this function. User needs to
have EXECUTE
```

To grant the global EXECUTE privilege to a user, use this statement:

```
GRANT EXECUTE ON *.* TO user;
```

Alternatively, should you prefer to avoid granting the global EXECUTE privilege while still permitting users to access specific key-management operations, “wrapper” stored programs can be defined (a technique described later in this section).

- A key stored in the keyring by a given user can be manipulated later only by the same user. That is, the value of the CURRENT_USER() function at the time of key manipulation must have the same value as when the key was stored in the keyring. (This constraint rules out the use of the keyring UDFs for manipulation of instance-wide keys, such as those created by InnoDB to support tablespace encryption.)

  To enable multiple users to perform operations on the same key, “wrapper” stored programs can be defined (a technique described later in this section).

- Keyring UDFs support the key types and lengths supported by the underlying keyring plugin. For information about keys specific to a particular keyring plugin, see Section 6.4.4.7, “Supported Keyring Key Types and Lengths”.

To create a new random key and store it in the keyring, call keyring_key_generate(), passing to it an ID for the key, along with the key type (encryption method) and its length in bytes. The following call creates a 2,048-bit DSA-encrypted key named MyKey:

```
mysql> SELECT keyring_key_generate('MyKey', 'DSA', 256);
+-----------------------------------------+
| keyring_key_generate('MyKey', 'DSA', 256) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

A return value of 1 indicates success. If the key cannot be created, the return value is NULL and an error occurs. One reason this might be is that the underlying keyring plugin does not support the specified combination of key type and key length; see Section 6.4.4.7, “Supported Keyring Key Types and Lengths”.

To be able to check the return type regardless of whether an error occurs, use SELECT ... INTO @var_name and test the variable value:

```
mysql> SELECT keyring_key_generate('', '', -1) INTO @x;
```

```
ERROR 3188 (HY000): Function 'keyring_key_generate' failed because
underlying keyring service returned an error. Please check if a
keyring plugin is installed and that provided arguments are valid
for the keyring you are using.
mysql> SELECT @x;
+------+
| @x   |
+------+
| NULL |
+------+
mysql> SELECT keyring_key_generate('x', 'AES', 16) INTO @x;
mysql> SELECT @x;
+------+
| @x   |
+------+
|    1 |
+------+
```

This technique also applies to other keyring UDFs that for failure return a value and an error.

The ID passed to `keyring_key_generate()` provides a means by which to refer to the key in subsequent UDF calls. For example, use the key ID to retrieve its type as a string or its length in bytes as an integer:

```
mysql> SELECT keyring_key_type_fetch('MyKey');
+---------------------------------+
| keyring_key_type_fetch('MyKey') |
+---------------------------------+
| DSA                             |
+---------------------------------+
mysql> SELECT keyring_key_length_fetch('MyKey');
+----------------------------------+
| keyring_key_length_fetch('MyKey') |
+----------------------------------+
|                              256 |
+----------------------------------+
```

To retrieve a key value, pass the key ID to `keyring_key_fetch()`. The following example uses `HEX()` to display the key value because it may contain nonprintable characters. The example also uses a short key for brevity, but be aware that longer keys provide better security:

```
mysql> SELECT keyring_key_generate('MyShortKey', 'DSA', 8);
+---------------------------------------------+
| keyring_key_generate('MyShortKey', 'DSA', 8) |
+---------------------------------------------+
|                                           1 |
+---------------------------------------------+
mysql> SELECT HEX(keyring_key_fetch('MyShortKey'));
+--------------------------------------+
| HEX(keyring_key_fetch('MyShortKey')) |
+--------------------------------------+
| 1DB3B0FC3328A24C                     |
+--------------------------------------+
```

Keyring UDFs treat key IDs, types, and values as binary strings, so comparisons are case-sensitive. For example, IDs of `MyKey` and `mykey` refer to different keys.

To remove a key, pass the key ID to `keyring_key_remove()`:

```
mysql> SELECT keyring_key_remove('MyKey');
+---------------------------+
```

```
| keyring_key_remove('MyKey') |
+-----------------------------+
|                           1 |
+-----------------------------+
```

To obfuscate and store a key that you provide, pass the key ID, type, and value to `keyring_key_store()`:

```
mysql> SELECT keyring_key_store('AES_key', 'AES', 'Secret string');
+-----------------------------------------------------+
| keyring_key_store('AES_key', 'AES', 'Secret string') |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
```

As indicated previously, a user must have the global `EXECUTE` privilege to call keyring UDFs, and the user who stores a key in the keyring initially must be the same user who performs subsequent operations on the key later, as determined from the `CURRENT_USER()` value in effect for each UDF call. To permit key operations to users who do not have the global `EXECUTE` privilege or who may not be the key "owner," use this technique:

1. Define "wrapper" stored programs that encapsulate the required key operations and have a `DEFINER` value equal to the key owner.
2. Grant the `EXECUTE` privilege for specific stored programs to the individual users who should be able to invoke them.
3. If the operations implemented by the wrapper stored programs do not include key creation, create any necessary keys in advance, using the account named as the `DEFINER` in the stored program definitions.

This technique enables keys to be shared among users and provides to DBAs more fine-grained control over who can do what with keys, without having to grant global privileges.

The following example shows how to set up a shared key named `SharedKey` that is owned by the DBA, and a `get_shared_key()` stored function that provides access to the current key value. The value can be retrieved by any user with the `EXECUTE` privilege for that function, which is created in the `key_schema` schema.

From a MySQL administrative account (`'root'@'localhost'` in this example), create the administrative schema and the stored function to access the key:

```
mysql> CREATE SCHEMA key_schema;

mysql> CREATE DEFINER = 'root'@'localhost'
       FUNCTION key_schema.get_shared_key()
       RETURNS BLOB READS SQL DATA
       RETURN keyring_key_fetch('SharedKey');
```

From the administrative account, ensure that the shared key exists:

```
mysql> SELECT keyring_key_generate('SharedKey', 'DSA', 8);
+-----------------------------------------+
| keyring_key_generate('SharedKey', 'DSA', 8) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

```
+-------------------------------------------+
```

From the administrative account, create an ordinary user account to which key access is to be granted:

```
mysql> CREATE USER 'key_user'@'localhost'
       IDENTIFIED BY 'key_user_pwd';
```

From the `key_user` account, verify that, without the proper `EXECUTE` privilege, the new account cannot access the shared key:

```
mysql> SELECT HEX(key_schema.get_shared_key());
ERROR 1370 (42000): execute command denied to user 'key_user'@'localhost'
for routine 'key_schema.get_shared_key'
```

From the administrative account, grant `EXECUTE` to `key_user` for the stored function:

```
mysql> GRANT EXECUTE ON FUNCTION key_schema.get_shared_key
       TO 'key_user'@'localhost';
```

From the `key_user` account, verify that the key is now accessible:

```
mysql> SELECT HEX(key_schema.get_shared_key());
+----------------------------------+
| HEX(key_schema.get_shared_key()) |
+----------------------------------+
| 9BAFB9E75CEEB013                 |
+----------------------------------+
```

General-Purpose Keyring Function Reference

For each general-purpose keyring user-defined function (UDF), this section describes its purpose, calling sequence, and return value. For information about the conditions under which these UDFs can be invoked, see Using General-Purpose Keyring Functions.

- `keyring_key_fetch(`*`key_id`*`)`

  Given a key ID, deobfuscates and returns the key value.

  Arguments:

  - *`key_id`*: A string that specifies the key ID.

  Return value:

  Returns the key value as a string for success, `NULL` if the key does not exist, or `NULL` and an error for failure.

  Note

  Key values retrieved using `keyring_key_fetch()` are subject to the general keyring UDF limits described in Section 6.4.4.7, “Supported Keyring Key Types and Lengths”. A key value longer than that length can be stored using a keyring service function (see Section 29.3.2, “The Keyring Service”), but if retrieved using `keyring_key_fetch()` is truncated to the general keyring UDF limit.

  Example:

```
mysql> SELECT keyring_key_generate('RSA_key', 'RSA', 16);
+-----------------------------------------+
| keyring_key_generate('RSA_key', 'RSA', 16) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
mysql> SELECT HEX(keyring_key_fetch('RSA_key'));
+----------------------------------+
| HEX(keyring_key_fetch('RSA_key')) |
+----------------------------------+
| 91C2253B696064D3556984B6630F891A  |
+----------------------------------+
mysql> SELECT keyring_key_type_fetch('RSA_key');
+----------------------------------+
| keyring_key_type_fetch('RSA_key') |
+----------------------------------+
| RSA                              |
+----------------------------------+
mysql> SELECT keyring_key_length_fetch('RSA_key');
+-----------------------------------+
| keyring_key_length_fetch('RSA_key') |
+-----------------------------------+
|                                16 |
+-----------------------------------+
```

The example uses `HEX()` to display the key value because it may contain nonprintable characters. The example also uses a short key for brevity, but be aware that longer keys provide better security.

- `keyring_key_generate(`*`key_id`*`, `*`key_type`*`, `*`key_length`*`)`

  Generates a new random key with a given ID, type, and length, and stores it in the keyring. The type and length values must be consistent with the values supported by the underlying keyring plugin. See Section 6.4.4.7, “Supported Keyring Key Types and Lengths”.

  Arguments:

  - *`key_id`*: A string that specifies the key ID.
  - *`key_type`*: A string that specifies the key type.
  - *`key_length`*: An integer that specifies the key length in bytes.

  Return value:

  Returns 1 for success, or `NULL` and an error for failure.

  Example:

```
mysql> SELECT keyring_key_generate('RSA_key', 'RSA', 384);
+-----------------------------------------+
| keyring_key_generate('RSA_key', 'RSA', 384) |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
```

- `keyring_key_length_fetch(`*`key_id`*`)`

  Given a key ID, returns the key length.

  Arguments:

- *key_id*: A string that specifies the key ID.

Return value:

Returns the key length in bytes as an integer for success, NULL if the key does not exist, or NULL and an error for failure.

Example:

See the description of keyring_key_fetch().

- keyring_key_remove(*key_id*)

Removes the key with a given ID from the keyring.

Arguments:

- *key_id*: A string that specifies the key ID.

Return value:

Returns 1 for success, or NULL for failure.

Example:

```
mysql> SELECT keyring_key_remove('AES_key');
+------------------------------+
| keyring_key_remove('AES_key') |
+------------------------------+
|                            1 |
+------------------------------+
```

- keyring_key_store(*key_id*, *key_type*, *key*)

Obfuscates and stores a key in the keyring.

Arguments:

- *key_id*: A string that specifies the key ID.
- *key_type*: A string that specifies the key type.
- *key*: A string that specifies the key value.

Return value:

Returns 1 for success, or NULL and an error for failure.

Example:

```
mysql> SELECT keyring_key_store('new key', 'DSA', 'My key value');
+-----------------------------------------------------+
| keyring_key_store('new key', 'DSA', 'My key value') |
+-----------------------------------------------------+
|                                                   1 |
+-----------------------------------------------------+
```

- keyring_key_type_fetch(*key_id*)

Given a key ID, returns the key type.

Arguments:

- *`key_id`*: A string that specifies the key ID.

Return value:

Returns the key type as a string for success, `NULL` if the key does not exist, or `NULL` and an error for failure.

Example:

See the description of `keyring_key_fetch()`.

6.4.4.9 Plugin-Specific Keyring Key-Management Functions

For each keyring plugin-specific user-defined function (UDF), this section describes its purpose, calling sequence, and return value. For information about general-purpose keyring UDFs, see Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”.

- `keyring_aws_rotate_cmk()`

  Associated keyring plugin: `keyring_aws`

  `keyring_aws_rotate_cmk()` rotates the customer master key (CMK). Rotation changes only the key that AWS KMS uses for subsequent data key-encryption operations. AWS KMS maintains previous CMK versions, so keys generated using previous CMKs remain decryptable after rotation.

  Rotation changes the CMK value used inside AWS KMS but does not change the ID used to refer to it, so there is no need to change the `keyring_aws_cmk_id` system variable after calling `keyring_aws_rotate_cmk()`.

  This UDF requires the `SUPER` privilege.

  Arguments:

  None.

  Return value:

  Returns 1 for success, or `NULL` and an error for failure.

- `keyring_aws_rotate_keys()`

  Associated keyring plugin: `keyring_aws`

  `keyring_aws_rotate_keys()` rotates keys stored in the `keyring_aws` storage file named by the `keyring_aws_data_file` system variable. Rotation sends each key stored in the file to AWS KMS for re-encryption using the value of the `keyring_aws_cmk_id` system variable as the CMK value, and stores the new encrypted keys in the file.

  `keyring_aws_rotate_keys()` is useful for key re-encryption under these circumstances:

  - After rotating the CMK; that is, after invoking the `keyring_aws_rotate_cmk()` UDF
  - After changing the `keyring_aws_cmk_id` system variable to a different key value

This UDF requires the SUPER privilege.

Arguments:

None.

Return value:

Returns 1 for success, or NULL and an error for failure.

6.4.4.10 Keyring Command Options

MySQL supports the following keyring-related command-line options:

- `--keyring-migration-destination=plugin`

Property	Value
Command-Line Format	`--keyring-migration-destination=plugin_name`
Introduced	8.0.4
Type	String

The destination keyring plugin for key migration. See Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”. The format and interpretation of the option value is the same as described for the `--keyring-migration-source` option.

Note

`--keyring-migration-source` and `--keyring-migration-destination` are mandatory for all keyring migration operations. The source and destination plugins must differ, and the migration server must support both plugins.

- `--keyring-migration-host=host_name`

Property	Value
Command-Line Format	`--keyring-migration-host=host_name`
Introduced	8.0.4
Type	String
Default Value	`localhost`

The host location of the running server that is currently using one of the key migration keystores. See Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”. Migration always occurs on the local host, so the option always specifies a value for connecting to a local server, such as `localhost`, `127.0.0.1`, `::1`, or the local host IP address or host name.

- `--keyring-migration-password[=password]`

Property	Value
Command-Line Format	`--keyring-migration-password[=password]`
Introduced	8.0.4
Type	String

The password for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”. If you omit the *`password`* value following the option name on the command line, the server prompts for one.

Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. You can use an option file to avoid giving the password on the command line. In this case, the file should have a restrictive mode and be accessible only to the account used to run the migration server.

- `--keyring-migration-port=`*`port_num`*

Property	Value
Command-Line Format	`--keyring-migration-port=port_num`
Introduced	8.0.4
Type	Numeric
Default Value	`3306`

For TCP/IP connections, the port number for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”.

- `--keyring-migration-socket=`*`path`*

Property	Value
Command-Line Format	`--keyring-migration-socket={file_name|pipe_name}`
Introduced	8.0.4
Type	String

For Unix socket file or Windows named pipe connections, the socket file or named pipe for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”.

- `--keyring-migration-source=`*`plugin`*

Property	Value
Command-Line Format	`--keyring-migration-source=plugin_name`
Introduced	8.0.4
Type	String

The source keyring plugin for key migration. See Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”.

The option value is similar to that for `--plugin-load`, except that only one plugin library can be specified. The value is given as *`name=plugin_library`* or *`plugin_library`*. The *`name`* is the name of a plugin to load, and *`plugin_library`* is the name of the library file that contains the plugin code. If the plugin library is named without any preceding plugin name, the server loads all plugins in the library. The server looks for plugin library files in the directory named by the `plugin_dir` system variable.

Note

`--keyring-migration-source` and `--keyring-migration-destination` are mandatory for all keyring migration operations. The source and destination plugins must differ, and the migration server must support both plugins.

- `--keyring-migration-user=`*`user_name`*

Property	Value
Command-Line Format	`--keyring-migration-user=user_name`
Introduced	8.0.4
Type	String

The user name for connecting to the running server that is currently using one of the key migration keystores. See Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”.

6.4.4.11 Keyring System Variables

MySQL Keyring plugins support the following system variables. Use them to configure keyring plugin operation. These variables are unavailable unless the appropriate keyring plugin is installed (see Section 6.4.4.1, “Keyring Plugin Installation”).

- `keyring_aws_cmk_id`

Property	Value
Command-Line Format	`--keyring-aws-cmk-id=value`
Introduced	8.0.11
System Variable	`keyring_aws_cmk_id`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The customer master key (CMK) ID obtained from the AWS KMS server and used by the `keyring_aws` plugin. This variable is unavailable unless that plugin is installed, but if it is installed, a value for this variable is mandatory.

- `keyring_aws_conf_file`

Property	Value
Command-Line Format	`--keyring-aws-conf-file=file_name`
Introduced	8.0.11
System Variable	`keyring_aws_conf_file`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

Property	Value
Default Value	`platform specific`

The location of the configuration file for the `keyring_aws` keyring plugin. This variable is unavailable unless that plugin is installed.

At plugin startup, `keyring_aws` reads the AWS secret access key ID and key from the configuration file. For the `keyring_aws` plugin to start successfully, the configuration file must exist and contain valid secret access key information, initialized as described in Section 6.4.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”.

The default file name is `keyring_aws_conf`, located in the default keyring file directory. The location of this default directory is the same as for the `keyring_file_data` system variable. See the description of that variable for details, as well as for considerations to take into account if you create the directory manually.

- `keyring_aws_data_file`

Property	Value
Command-Line Format	`--keyring-aws-data-file`
Introduced	8.0.11
System Variable	`keyring_aws_data_file`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	`platform specific`

The location of the storage file for the `keyring_aws` keyring plugin. This variable is unavailable unless that plugin is installed.

At plugin startup, if the value assigned to `keyring_aws_data_file` specifies a file that does not exist, the `keyring_aws` plugin attempts to create it (as well as its parent directory, if necessary). If the file does exist, `keyring_aws` reads any encrypted keys contained in the file into its in-memory cache. `keyring_aws` does not cache unencrypted keys in memory.

The default file name is `keyring_aws_data`, located in the default keyring file directory. The location of this default directory is the same as for the `keyring_file_data` system variable. See the description of that variable for details, as well as for considerations to take into account if you create the directory manually.

- `keyring_aws_region`

Property	Value
Command-Line Format	`--keyring-aws-region=value`
Introduced	8.0.11
System Variable	`keyring_aws_region`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`us-east-1`

Valid Values	`ap-northeast-1`
	`ap-northeast-2`
	`ap-south-1`
	`ap-southeast-1`
	`ap-southeast-2`
	`eu-central-1`
	`eu-west-1`
	`sa-east-1`
	`us-east-1`
	`us-west-1`
	`us-west-2`

The AWS region.

- `keyring_encrypted_file_data`

Property	Value
Command-Line Format	`--keyring-encrypted-file-data=file_name`
Introduced	8.0.11
System Variable	`keyring_encrypted_file_data`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	`platform specific`

The path name of the data file used for secure data storage by the `keyring_encrypted_file` plugin. This variable is unavailable unless that plugin is installed. The file location should be in a directory considered for use only by keyring plugins. For example, do not locate the file under the data directory.

Keyring operations are transactional: The `keyring_encrypted_file` plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the `keyring_encrypted_file_data` system variable with a suffix of `.backup`.

Do not use the same `keyring_encrypted_file` data file for multiple MySQL instances. Each instance should have its own unique data file.

The default file name is `keyring_encrypted`, located in a directory that is platform specific and depends on the value of the `INSTALL_LAYOUT` `CMake` option, as shown in the following table. To specify the default directory for the file explicitly if you are building from source, use the `INSTALL_MYSQLKEYRINGDIR` `CMake` option.

`INSTALL_LAYOUT` Value	**Default `keyring_encrypted_file_data` Value**
`DEB`, `RPM`, `SVR4`	`/var/lib/mysql-keyring/keyring_encrypted`
Otherwise	`keyring/keyring_encrypted` under the `CMAKE_INSTALL_PREFIX` value

At plugin startup, if the value assigned to `keyring_encrypted_file_data` specifies a file that does not exist, the `keyring_encrypted_file` plugin attempts to create it (as well as its parent directory, if necessary).

If you create the directory manually, it should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the `/usr/local/mysql/mysql-keyring` directory, the following commands (executed as `root`) create the directory and set its mode and ownership:

```
cd /usr/local/mysql
mkdir mysql-keyring
chmod 750 mysql-keyring
chown mysql mysql-keyring
chgrp mysql mysql-keyring
```

If the `keyring_encrypted_file` plugin cannot create or access its data file, it writes an error message to the error log. If an attempted runtime assignment to `keyring_encrypted_file_data` results in an error, the variable value remains unchanged.

Important

Once the `keyring_encrypted_file` plugin has created its data file and started to use it, it is important not to remove the file. Loss of the file will cause data encrypted using its keys to become inaccessible. (It is permissible to rename or move the file, as long as you change the value of `keyring_encrypted_file_data` to match.)

- `keyring_encrypted_file_password`

Property	**Value**
Command-Line Format	`--keyring-encrypted-file-password=password`
Introduced	8.0.11
System Variable	`keyring_encrypted_file_password`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

The password used by the `keyring_encrypted_file` plugin. This variable is unavailable unless that plugin is installed. The password is mandatory for plugin operation; if not specified at server startup, `keyring_encrypted_file` initialization fails.

If this variable is specified in an option file, the file should have a restrictive mode and be accessible only to the account used to run the MySQL server.

Important

Once the `keyring_encrypted_file_password` value has been set, changing it does not rotate the keyring password and could make the server inaccessible. If an incorrect password is provided, the `keyring_encrypted_file` plugin cannot load keys from the encrypted keyring file.

The password value cannot be displayed at runtime with `SHOW VARIABLES` or the Performance Schema `global_variables` table because the display value is obfuscated.

- `keyring_file_data`

Property	Value
Command-Line Format	`--keyring-file-data=file_name`
System Variable	`keyring_file_data`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	File name
Default Value	`platform specific`

The path name of the data file used for secure data storage by the `keyring_file` plugin. This variable is unavailable unless that plugin is installed. The file location should be in a directory considered for use only by keyring plugins. For example, do not locate the file under the data directory.

Keyring operations are transactional: The `keyring_file` plugin uses a backup file during write operations to ensure that it can roll back to the original file if an operation fails. The backup file has the same name as the value of the `keyring_file_data` system variable with a suffix of `.backup`.

Do not use the same `keyring_file` data file for multiple MySQL instances. Each instance should have its own unique data file.

The default file name is `keyring`, located in a directory that is platform specific and depends on the value of the `INSTALL_LAYOUT` `CMake` option, as shown in the following table. To specify the default directory for the file explicitly if you are building from source, use the `INSTALL_MYSQLKEYRINGDIR` `CMake` option.

`INSTALL_LAYOUT` Value	Default `keyring_file_data` Value
`DEB`, `RPM`, `SVR4`	`/var/lib/mysql-keyring/keyring`
Otherwise	`keyring/keyring` under the `CMAKE_INSTALL_PREFIX` value

At plugin startup, if the value assigned to `keyring_file_data` specifies a file that does not exist, the `keyring_file` plugin attempts to create it (as well as its parent directory, if necessary).

If you create the directory manually, it should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the `/usr/`

`local/mysql/mysql-keyring` directory, the following commands (executed as `root`) create the directory and set its mode and ownership:

```
cd /usr/local/mysql
mkdir mysql-keyring
chmod 750 mysql-keyring
chown mysql mysql-keyring
chgrp mysql mysql-keyring
```

If the `keyring_file` plugin cannot create or access its data file, it writes an error message to the error log. If an attempted runtime assignment to `keyring_file_data` results in an error, the variable value remains unchanged.

Important

Once the `keyring_file` plugin has created its data file and started to use it, it is important not to remove the file. For example, `InnoDB` uses the file to store the master key used to decrypt the data in tables that use `InnoDB` tablespace encryption; see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”. Loss of the file will cause data in such tables to become inaccessible. (It is permissible to rename or move the file, as long as you change the value of `keyring_file_data` to match.) It is recommended that you create a separate backup of the keyring data file immediately after you create the first encrypted table and before and after master key rotation.

- `keyring_okv_conf_dir`

Property	Value
Command-Line Format	`--keyring-okv-conf-dir=dir_name`
Introduced	8.0.11
System Variable	`keyring_okv_conf_dir`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Directory name
Default Value	`empty string`

The path name of the directory that stores configuration information used by the `keyring_okv` plugin. This variable is unavailable unless that plugin is installed. The location should be a directory considered for use only by the `keyring_okv` plugin. For example, do not locate the directory under the data directory.

The default `keyring_okv_conf_dir` value is empty. For the `keyring_okv` plugin to be able to access Oracle Key Vault, the value must be set to a directory that contains Oracle Key Vault configuration and SSL materials. For instructions on setting up this directory, see Section 6.4.4.4, “Using the keyring_okv KMIP Plugin”.

The directory should have a restrictive mode and be accessible only to the account used to run the MySQL server. For example, on Unix and Unix-like systems, to use the `/usr/local/mysql/mysql-keyring-okv` directory, the following commands (executed as `root`) create the directory and set its mode and ownership:

```
cd /usr/local/mysql
mkdir mysql-keyring-okv
chmod 750 mysql-keyring-okv
chown mysql mysql-keyring-okv
chgrp mysql mysql-keyring-okv
```

If the value assigned to `keyring_okv_conf_dir` specifies a directory that does not exist, or that does not contain configuration information that enables a connection to Oracle Key Vault to be established, `keyring_okv` writes an error message to the error log. If an attempted runtime assignment to `keyring_okv_conf_dir` results in an error, the variable value and keyring operation remain unchanged.

- `keyring_operations`

Property	Value
Introduced	8.0.4
System Variable	`keyring_operations`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether keyring operations are enabled. This variable is used during key migration operations. See Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”. The privileges required to modify this variable are `ENCRYPTION_KEY_ADMIN` in addition to either `SYSTEM_VARIABLES_ADMIN` or `SUPER`.

6.4.5 MySQL Enterprise Audit

Note

MySQL Enterprise Audit is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin named `audit_log`. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring, logging, and blocking of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

After you install the audit plugin (see Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”), it writes an audit log file. By default, the file is named `audit.log` in the server data directory. To change the name of the file, set the `audit_log_file` system variable at server startup.

By default, audit log file contents are written in new-style XML format, without compression or encryption. To select the file format, set the `audit_log_format` system variable at server startup. For details on file format and contents, see Section 6.4.5.4, “Audit Log File Formats”.

For more information about controlling how logging occurs, including audit log file naming and format selection, see Section 6.4.5.5, “Audit Log Logging Configuration”. To perform filtering of audited events, see Section 6.4.5.6, “Audit Log Filtering”. For descriptions of the parameters used to configure the audit log plugin, see Audit Log Options and Variables.

If the audit log plugin is enabled, the Performance Schema (see Chapter 26, *MySQL Performance Schema*) has instrumentation for it. To identify the relevant instruments, use this query:

```
SELECT NAME FROM performance_schema.setup_instruments
WHERE NAME LIKE '%/alog/%';
```

6.4.5.1 Audit Log Components

MySQL Enterprise Audit is based on the audit log plugin and related components:

- A server-side plugin named `audit_log` examines auditable events and determines whether to write them to the audit log.
- User-defined functions enable manipulation of filtering definitions that control logging behavior, the encryption password, and log file reading.
- Tables in the `mysql` system database provide persistent storage of filter and user account data.
- System variables enable audit log configuration and status variables provide runtime operational information.
- An `AUDIT_ADMIN` privilege enable users to administer the audit log.

6.4.5.2 Installing or Uninstalling MySQL Enterprise Audit

This section describes how to install or uninstall MySQL Enterprise Audit, which is implemented using the audit log plugin and related components described in Section 6.4.5.1, “Audit Log Components”. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

Important

Read this entire section before following its instructions. Parts of the procedure differ depending on your environment.

Note

If installed, the `audit_log` plugin involves some minimal overhead even when disabled. To avoid this overhead, do not install MySQL Enterprise Audit unless you plan to use it.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To install MySQL Enterprise Audit, look in the `share` directory of your MySQL installation and choose the script that is appropriate for your platform. The available scripts differ in the suffix used to refer to the plugin library file:

- `audit_log_filter_win_install.sql`: Choose this script for Windows systems that use `.dll` as the file name suffix.
- `audit_log_filter_linux_install.sql`: Choose this script for Linux and similar systems that use `.so` as the file name suffix.

Run the script as follows. The example here uses the Linux installation script. Make the appropriate substitution for your system.

```
shell> mysql -u root -p < audit_log_filter_linux_install.sql
Enter password: (enter root password here)
```

Note

Some MySQL versions have introduced changes to the structure of the MySQL Enterprise Audit tables. To ensure that your tables are up to date for upgrades from earlier versions of MySQL 8.0, perform the MySQL upgrade procedure, making sure to use the option that forces an update (see Section 2.11, “Upgrading MySQL”). If you prefer to run the update statements only for the MySQL Enterprise Audit tables, see the following discussion.

As of MySQL 8.0.12, for new MySQL installations, the `USER` and `HOST` columns in the `audit_log_user` table used by MySQL Enterprise Audit have definitions that better correspond to the definitions of the `User` and `Host` columns in the `mysql.user` system table. For upgrades to an installation for which MySQL Enterprise Audit is already installed, it is recommended that you alter the table definitions as follows:

```
ALTER TABLE mysql.audit_log_user
  DROP FOREIGN KEY audit_log_user_ibfk_1;
ALTER TABLE mysql.audit_log_filter
  CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_as_ci;
ALTER TABLE mysql.audit_log_user
  CONVERT TO CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_as_ci;
ALTER TABLE mysql.audit_log_user
  MODIFY COLUMN USER VARCHAR(32);
ALTER TABLE mysql.audit_log_user
  ADD FOREIGN KEY (FILTERNAME) REFERENCES mysql.audit_log_filter(NAME);
```

Note

To use MySQL Enterprise Audit in the context of master/slave replication, Group Replication, or InnoDB cluster, you must prepare the slave or secondary nodes prior to running the installation script on the master or primary node. This is necessary because the `INSTALL PLUGIN` statement in the script is not replicated.

1. On each slave or secondary node, extract the `INSTALL PLUGIN` statement from the installation script and execute it manually.
2. On the master or primary node, run the installation script as described previously.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
```

```
      WHERE PLUGIN_NAME LIKE 'audit%';
+-------------+---------------+
| PLUGIN_NAME | PLUGIN_STATUS |
+-------------+---------------+
| audit_log   | ACTIVE        |
+-------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

After MySQL Enterprise Audit is installed, you can use the `--audit-log` option for subsequent server startups to control `audit_log` plugin activation. For example, to prevent the plugin from being removed at runtime, use this option:

```
[mysqld]
audit-log=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without the audit plugin, use `--audit-log` with a value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

Important

By default, rule-based audit log filtering logs no auditable events for any users. This differs from legacy audit log behavior, which logs all auditable events for all users (see Section 6.4.5.8, “Legacy Mode Audit Log Filtering”). Should you wish to produce log-everything behavior with rule-based filtering, create a simple filter to enable logging and assign it to the default account:

```
SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');
SELECT audit_log_filter_set_user('%', 'log_all');
```

The filter assigned to `%` is used for connections from any account that has no explicitly assigned filter (which initially is true for all accounts).

Once installed as just described, MySQL Enterprise Audit remains installed until uninstalled. To remove it, execute the following statements:

```
DROP TABLE IF EXISTS mysql.audit_log_user;
DROP TABLE IF EXISTS mysql.audit_log_filter;
UNINSTALL PLUGIN audit_log;
DROP FUNCTION audit_log_filter_set_filter;
DROP FUNCTION audit_log_filter_remove_filter;
DROP FUNCTION audit_log_filter_set_user;
DROP FUNCTION audit_log_filter_remove_user;
DROP FUNCTION audit_log_filter_flush;
DROP FUNCTION audit_log_encryption_password_get;
DROP FUNCTION audit_log_encryption_password_set;
DROP FUNCTION audit_log_read;
DROP FUNCTION audit_log_read_bookmark;
```

6.4.5.3 MySQL Enterprise Audit Security Considerations

By default, contents of audit log files produced by the audit log plugin are not encrypted and may contain sensitive information, such as the text of SQL statements. For security reasons, audit log files should be written to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log. The default file name is `audit.log` in the data directory. This can be changed by setting the `audit_log_file` system variable at server startup. Other audit log files may exist due to log rotation.

For additional security, enable audit log file encryption. See Audit Log File Encryption.

6.4.5.4 Audit Log File Formats

The MySQL server calls the audit log plugin to write an audit record to its log file whenever an auditable event occurs. Typically the first audit record written after plugin startup contains the server description and startup options. Elements following that one represent events such as client connect and disconnect events, executed SQL statements, and so forth. Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures. Contents of files referenced by statements such as `LOAD DATA` are not logged.

To select the log format that the audit log plugin uses to write its log file, set the `audit_log_format` system variable at server startup. These formats are available:

- New-style XML format (`audit_log_format=NEW`): An XML format that has better compatibility with Oracle Audit Vault than old-style XML format. MySQL 8.0 uses new-style XML format by default.
- Old-style XML format (`audit_log_format=OLD`): The original audit log format used by default in older MySQL series.
- JSON format (`audit_log_format=JSON`)

By default, audit log file contents are written in new-style XML format, without compression or encryption.

Note

For information about issues to consider when changing the log format, see Audit Log File Format.

The following sections describe the available audit logging formats:

- New-Style XML Audit Log File Format
- Old-Style XML Audit Log File Format
- JSON Audit Log File Format

New-Style XML Audit Log File Format

Here is a sample log file in new-style XML format (`audit_log_format=NEW`), reformatted slightly for readability:

```
<?xml version="1.0" encoding="utf-8"?>
<AUDIT>
 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:06:33 UTC</TIMESTAMP>
  <RECORD_ID>1_2017-10-16T14:06:33</RECORD_ID>
  <NAME>Audit</NAME>
  <SERVER_ID>1</SERVER_ID>
  <VERSION>1</VERSION>
  <STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
    --socket=/usr/local/mysql/mysql.sock
    --port=3306</STARTUP_OPTIONS>
  <OS_VERSION>i686-Linux</OS_VERSION>
  <MYSQL_VERSION>5.7.21-log</MYSQL_VERSION>
 </AUDIT_RECORD>
 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:09:38 UTC</TIMESTAMP>
  <RECORD_ID>2_2017-10-16T14:06:33</RECORD_ID>
  <NAME>Connect</NAME>
  <CONNECTION_ID>5</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root</USER>
```

```
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
  <CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
  <PRIV_USER>root</PRIV_USER>
  <PROXY_USER/>
  <DB>test</DB>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:09:38 UTC</TIMESTAMP>
  <RECORD_ID>6_2017-10-16T14:06:33</RECORD_ID>
  <NAME>Query</NAME>
  <CONNECTION_ID>5</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root[root] @ localhost [127.0.0.1]</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>drop_table</COMMAND_CLASS>
  <SQLTEXT>DROP TABLE IF EXISTS t</SQLTEXT>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:09:39 UTC</TIMESTAMP>
  <RECORD_ID>8_2017-10-16T14:06:33</RECORD_ID>
  <NAME>Quit</NAME>
  <CONNECTION_ID>5</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
  <CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:09:43 UTC</TIMESTAMP>
  <RECORD_ID>11_2017-10-16T14:06:33</RECORD_ID>
  <NAME>Quit</NAME>
  <CONNECTION_ID>6</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
  <CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
 </AUDIT_RECORD>
 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:09:45 UTC</TIMESTAMP>
  <RECORD_ID>12_2017-10-16T14:06:33</RECORD_ID>
  <NAME>NoAudit</NAME>
  <SERVER_ID>1</SERVER_ID>
 </AUDIT_RECORD>
</AUDIT>
```

The audit log file is written as XML, using UTF-8 (up to 4 bytes per character). The root element is `<AUDIT>`. The root element contains `<AUDIT_RECORD>` elements, each of which provides information about an audited event. When the audit log plugin begins writing a new log file, it writes the XML declaration and opening `<AUDIT>` root element tag. When the plugin closes a log file, it writes the closing `</AUDIT>` root element tag. The closing tag is not present while the file is open.

Elements within `<AUDIT_RECORD>` elements have these characteristics:

- Some elements appear in every `<AUDIT_RECORD>` element. Others are optional and may appear depending on the audit record type.
- Order of elements within an `<AUDIT_RECORD>` element is not guaranteed.
- Element values are not fixed length. Long values may be truncated as indicated in the element descriptions given later.
- The `<`, `>`, `"`, and `&` characters are encoded as `<`, `>`, `"`, and `&`, respectively. NUL bytes (U+00) are encoded as the `?` character.
- Characters not valid as XML characters are encoded using numeric character references. Valid XML characters are:

```
#x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
```

The following elements are mandatory in every `<AUDIT_RECORD>` element:

- `<NAME>`

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  Example:

  ```
  <NAME>Query</NAME>
  ```

  Some common `<NAME>` values:

  ```
  Audit    When auditing starts, which may be server startup time
  Connect  When a client connects, also known as logging in
  Query    An SQL statement (executed directly)
  Prepare  Preparation of an SQL statement; usually followed by Execute
  Execute  Execution of an SQL statement; usually follows Prepare
  Shutdown Server shutdown
  Quit     When a client disconnects
  NoAudit  Auditing has been turned off
  ```

  The possible values are `Audit`, `Binlog Dump`, `Change user`, `Close stmt`, `Connect Out`, `Connect`, `Create DB`, `Daemon`, `Debug`, `Delayed insert`, `Drop DB`, `Execute`, `Fetch`, `Field List`, `Init DB`, `Kill`, `Long Data`, `NoAudit`, `Ping`, `Prepare`, `Processlist`, `Query`, `Quit`, `Refresh`, `Register Slave`, `Reset stmt`, `Set option`, `Shutdown`, `Sleep`, `Statistics`, `Table Dump`, `Time`.

  With the exception of `Audit` and `NoAudit`, these values correspond to the `COM_xxx` command values listed in the `my_command.h` header file. For example, `Create DB` and `Change user` correspond to `COM_CREATE_DB` and `COM_CHANGE_USER`, respectively.

- `<RECORD_ID>`

A unique identifier for the audit record. The value is composed from a sequence number and timestamp, in the format *`SEQ_TIMESTAMP`*. When the audit log plugin opens the audit log file, it initializes the sequence number to the size of the audit log file, then increments the sequence by 1 for each record logged. The timestamp is a UTC value in *`YYYY-MM-DD`*`hh:mm:ss` format indicating the date and time when the audit log plugin opened the file.

Example:

```
<RECORD_ID>12_2017-10-16T14:06:33</RECORD_ID>
```

- `<TIMESTAMP>`

  A string representing a UTC value in *`YYYY-MM-DD`*`hh:mm:ss UTC` format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a `<TIMESTAMP>` value occurring after the statement finishes, not when it was received.

  Example:

```
<TIMESTAMP>2017-10-16T14:09:45 UTC</TIMESTAMP>
```

The following elements are optional in `<AUDIT_RECORD>` elements. Many of them occur only with specific `<NAME>` element values.

- `<COMMAND_CLASS>`

  A string that indicates the type of action performed.

  Example:

```
<COMMAND_CLASS>drop_table</COMMAND_CLASS>
```

  The values correspond to the `statement/sql/`*`xxx`* command counters. For example, *`xxx`* is `drop_table` and `select` for `DROP TABLE` and `SELECT` statements, respectively. The following statement displays the possible names:

```
SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%'
ORDER BY name;
```

- `<CONNECTION_ID>`

  An unsigned integer representing the client connection identifier. This is the same as the value returned by the `CONNECTION_ID()` function within the session.

  Example:

```
<CONNECTION_ID>127</CONNECTION_ID>
```

- `<CONNECTION_TYPE>`

  The security state of the connection to the server. Permitted values are `TCP/IP` (TCP/IP connection established without encryption), `SSL/TLS` (TCP/IP connection established with encryption), `Socket`

(Unix socket file connection), `Named Pipe` (Windows named pipe connection), and `Shared Memory` (Windows shared memory connection).

Example:

```
<CONNECTION_TYPE>SSL/TLS</CONNECTION_TYPE>
```

- `<DB>`

  A string representing the default database name.

  Example:

  ```
  <DB>test</DB>
  ```

- `<HOST>`

  A string representing the client host name.

  Example:

  ```
  <HOST>localhost</HOST>
  ```

- `<IP>`

  A string representing the client IP address.

  Example:

  ```
  <IP>127.0.0.1</IP>
  ```

- `<MYSQL_VERSION>`

  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable.

  Example:

  ```
  <MYSQL_VERSION>5.7.21-log</MYSQL_VERSION>
  ```

- `<OS_LOGIN>`

  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this element is empty. The value is the same as that of the `external_user` system variable (see Section 6.2.18, “Proxy Users”).

  Example:

  ```
  <OS_LOGIN>jeffrey</OS_LOGIN>
  ```

- `<OS_VERSION>`

  A string representing the operating system on which the server was built or is running.

Example:

```
<OS_VERSION>x86_64-Linux</OS_VERSION>
```

- `<PRIV_USER>`

  A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and may differ from the `<USER>` value.

  Example:

  ```
  <PRIV_USER>jeffrey</PRIV_USER>
  ```

- `<PROXY_USER>`

  A string representing the proxy user (see Section 6.2.18, “Proxy Users”). The value is empty if user proxying is not in effect.

  Example:

  ```
  <PROXY_USER>developer</PROXY_USER>
  ```

- `<SERVER_ID>`

  An unsigned integer representing the server ID. This is the same as the value of the `server_id` system variable.

  Example:

  ```
  <SERVER_ID>1</SERVER_ID>
  ```

- `<SQLTEXT>`

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example:

  ```
  <SQLTEXT>DELETE FROM t1</SQLTEXT>
  ```

- `<STARTUP_OPTIONS>`

  A string representing the options that were given on the command line or in option files when the MySQL server was started. The first option is the path to the server executable.

  Example:

  ```
  <STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
    --port=3306 --log_output=FILE</STARTUP_OPTIONS>
  ```

- `<STATUS>`

An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function. See the description for `<STATUS_CODE>` for information about how it differs from `<STATUS>`.

The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Section B.3.1, “Server Error Message Reference”.

Warnings are not logged.

Example:

```
<STATUS>1051</STATUS>
```

- `<STATUS_CODE>`

  An unsigned integer representing the command status: 0 for success, 1 if an error occurred.

  The `STATUS_CODE` value differs from the `STATUS` value: `STATUS_CODE` is 0 for success and 1 for error, which is compatible with the EZ_collector consumer for Audit Vault. `STATUS` is the value of the `mysql_errno()` C API function. This is 0 for success and nonzero for error, and thus is not necessarily 1 for error.

  Example:

  ```
  <STATUS_CODE>0</STATUS_CODE>
  ```

- `<USER>`

  A string representing the user name sent by the client. This may differ from the `<PRIV_USER>` value.

  Example:

  ```
  <USER>root[root] @ localhost [127.0.0.1]</USER>
  ```

- `<VERSION>`

  An unsigned integer representing the version of the audit log file format.

  Example:

  ```
  <VERSION>1</VERSION>
  ```

Old-Style XML Audit Log File Format

Here is a sample log file in old-style XML format (`audit_log_format=OLD`), reformatted slightly for readability:

```
<?xml version="1.0" encoding="utf-8"?>
<AUDIT>
  <AUDIT_RECORD
    TIMESTAMP="2017-10-16T14:25:00 UTC"
    RECORD_ID="1_2017-10-16T14:25:00"
    NAME="Audit"
    SERVER_ID="1"
    VERSION="1"
```

```
    STARTUP_OPTIONS="--port=3306"
    OS_VERSION="i686-Linux"
    MYSQL_VERSION="5.7.21-log"/>
  <AUDIT_RECORD
    TIMESTAMP="2017-10-16T14:25:24 UTC"
    RECORD_ID="2_2017-10-16T14:25:00"
    NAME="Connect"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="connect"
    CONNECTION_TYPE="SSL/TLS"
    PRIV_USER="root"
    PROXY_USER=""
    DB="test"/>

...

  <AUDIT_RECORD
    TIMESTAMP="2017-10-16T14:25:24 UTC"
    RECORD_ID="6_2017-10-16T14:25:00"
    NAME="Query"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root[root] @ localhost [127.0.0.1]"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="drop_table"
    SQLTEXT="DROP TABLE IF EXISTS t"/>

...

  <AUDIT_RECORD
    TIMESTAMP="2017-10-16T14:25:24 UTC"
    RECORD_ID="8_2017-10-16T14:25:00"
    NAME="Quit"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="connect"
    CONNECTION_TYPE="SSL/TLS"/>
  <AUDIT_RECORD
    TIMESTAMP="2017-10-16T14:25:32 UTC"
    RECORD_ID="12_2017-10-16T14:25:00"
    NAME="NoAudit"
    SERVER_ID="1"/>
</AUDIT>
```

The audit log file is written as XML, using UTF-8 (up to 4 bytes per character). The root element is `<AUDIT>`. The root element contains `<AUDIT_RECORD>` elements, each of which provides information about an audited event. When the audit log plugin begins writing a new log file, it writes the XML declaration and opening `<AUDIT>` root element tag. When the plugin closes a log file, it writes the closing `</AUDIT>` root element tag. The closing tag is not present while the file is open.

Attributes of `<AUDIT_RECORD>` elements have these characteristics:

- Some attributes appear in every <AUDIT_RECORD> element. Others are optional and may appear depending on the audit record type.
- Order of attributes within an <AUDIT_RECORD> element is not guaranteed.
- Attribute values are not fixed length. Long values may be truncated as indicated in the attribute descriptions given later.
- The <, >, ", and & characters are encoded as <, >, ", and &, respectively. NUL bytes (U+00) are encoded as the ? character.
- Characters not valid as XML characters are encoded using numeric character references. Valid XML characters are:

```
#x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
```

The following attributes are mandatory in every <AUDIT_RECORD> element:

- NAME

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  Example: NAME="Query"

  Some common NAME values:

```
Audit    When auditing starts, which may be server startup time
Connect  When a client connects, also known as logging in
Query    An SQL statement (executed directly)
Prepare  Preparation of an SQL statement; usually followed by Execute
Execute  Execution of an SQL statement; usually follows Prepare
Shutdown Server shutdown
Quit     When a client disconnects
NoAudit  Auditing has been turned off
```

  The possible values are Audit, Binlog Dump, Change user, Close stmt, Connect Out, Connect, Create DB, Daemon, Debug, Delayed insert, Drop DB, Execute, Fetch, Field List, Init DB, Kill, Long Data, NoAudit, Ping, Prepare, Processlist, Query, Quit, Refresh, Register Slave, Reset stmt, Set option, Shutdown, Sleep, Statistics, Table Dump, Time.

  With the exception of "Audit" and "NoAudit", these values correspond to the COM_*xxx* command values listed in the my_command.h header file. For example, "Create DB" and "Change user" correspond to COM_CREATE_DB and COM_CHANGE_USER, respectively.

- RECORD_ID

  A unique identifier for the audit record. The value is composed from a sequence number and timestamp, in the format *SEQ_TIMESTAMP*. When the audit log plugin opens the audit log file, it initializes the sequence number to the size of the audit log file, then increments the sequence by 1 for each record logged. The timestamp is a UTC value in *YYYY-MM-DD*T*hh:mm:ss* format indicating the date and time when the audit log plugin opened the file.

  Example: RECORD_ID="12_2017-10-16T14:25:00"

- TIMESTAMP

A string representing a UTC value in *YYYY-MM-DD*`T`*hh:mm:ss* `UTC` format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a `TIMESTAMP` value occurring after the statement finishes, not when it was received.

Example: `TIMESTAMP="2017-10-16T14:25:32 UTC"`

The following attributes are optional in `<AUDIT_RECORD>` elements. Many of them occur only for elements with specific values of the `NAME` attribute.

- `COMMAND_CLASS`

  A string that indicates the type of action performed.

  Example: `COMMAND_CLASS="drop_table"`

  The values correspond to the `statement/sql/`*xxx* command counters. For example, *xxx* is `drop_table` and `select` for `DROP TABLE` and `SELECT` statements, respectively. The following statement displays the possible names:

```
SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%'
ORDER BY name;
```

- `CONNECTION_ID`

  An unsigned integer representing the client connection identifier. This is the same as the value returned by the `CONNECTION_ID()` function within the session.

  Example: `CONNECTION_ID="127"`

- `CONNECTION_TYPE`

  The security state of the connection to the server. Permitted values are `TCP/IP` (TCP/IP connection established without encryption), `SSL/TLS` (TCP/IP connection established with encryption), `Socket` (Unix socket file connection), `Named Pipe` (Windows named pipe connection), and `Shared Memory` (Windows shared memory connection).

  Example: `CONNECTION_TYPE="SSL/TLS"`

- `DB`

  A string representing the default database name.

  Example: `DB="test"`

- `HOST`

  A string representing the client host name.

  Example: `HOST="localhost"`

- `IP`

  A string representing the client IP address.

  Example: `IP="127.0.0.1"`

- `MYSQL_VERSION`

  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable.

  Example: `MYSQL_VERSION="5.7.21-log"`

- `OS_LOGIN`

  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this attribute is empty. The value is the same as that of the `external_user` system variable (see Section 6.2.18, “Proxy Users”).

  Example: `OS_LOGIN="jeffrey"`

- `OS_VERSION`

  A string representing the operating system on which the server was built or is running.

  Example: `OS_VERSION="x86_64-Linux"`

- `PRIV_USER`

  A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and it may differ from the `USER` value.

  Example: `PRIV_USER="jeffrey"`

- `PROXY_USER`

  A string representing the proxy user (see Section 6.2.18, “Proxy Users”). The value is empty if user proxying is not in effect.

  Example: `PROXY_USER="developer"`

- `SERVER_ID`

  An unsigned integer representing the server ID. This is the same as the value of the `server_id` system variable.

  Example: `SERVER_ID="1"`

- `SQLTEXT`

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example: `SQLTEXT="DELETE FROM t1"`

- `STARTUP_OPTIONS`

  A string representing the options that were given on the command line or in option files when the MySQL server was started.

  Example: `STARTUP_OPTIONS="--port=3306 --log_output=FILE"`

- `STATUS`

  An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function. See the description for `STATUS_CODE` for information about how it differs from `STATUS`.

  The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Section B.3.1, “Server Error Message Reference”.

  Warnings are not logged.

  Example: `STATUS="1051"`

- `STATUS_CODE`

  An unsigned integer representing the command status: 0 for success, 1 if an error occurred.

  The `STATUS_CODE` value differs from the `STATUS` value: `STATUS_CODE` is 0 for success and 1 for error, which is compatible with the EZ_collector consumer for Audit Vault. `STATUS` is the value of the `mysql_errno()` C API function. This is 0 for success and nonzero for error, and thus is not necessarily 1 for error.

  Example: `STATUS_CODE="0"`

- `USER`

  A string representing the user name sent by the client. This may differ from the `PRIV_USER` value.

- `VERSION`

  An unsigned integer representing the version of the audit log file format.

  Example: `VERSION="1"`

JSON Audit Log File Format

For JSON-format audit logging (`audit_log_format=JSON`), the log file contents form a `JSON` array with each array element representing an audited event as a `JSON` hash of key-value pairs. Examples of complete event records appear later in this section. The following is an excerpt of partial events:

```
[
  {
    "timestamp": "2018-01-15 13:50:01",
    "id": 0,
    "class": "audit",
    "event": "startup",
    ...
  },
  {
    "timestamp": "2018-01-15 15:02:32",
    "id": 0,
    "class": "connection",
    "event": "connect",
    ...
  },
  ...
  {
    "timestamp": "2018-01-15 17:37:26",
    "id": 0,
    "class": "table_access",
```

```
    "event": "insert",
      ...
  }
  ...
]
```

The audit log file is written using UTF-8 (up to 4 bytes per character). When the audit log plugin begins writing a new log file, it writes the opening `[` array marker. When the plugin closes a log file, it writes the closing `]` array marker. The closing marker is not present while the file is open.

Items within audit records have these characteristics:

- Some items appear in every audit record. Others are optional and may appear depending on the audit record type.
- Order of items within an audit record is not guaranteed.
- Item values are not fixed length. Long values may be truncated as indicated in the item descriptions given later.
- The `"` and `\` characters are encoded as `\"` and `\\`, respectively.

The following examples show the JSON object formats for different event types (as indicated by the `class` and `event` items), reformatted slightly for readability:

Auditing startup event:

```
{ "timestamp": "2018-01-15 14:21:56",
  "id": 0,
  "class": "audit",
  "event": "startup",
  "connection_id": 0,
  "startup_data": { "server_id": 1,
                    "os_version": "i686-Linux",
                    "mysql_version": "5.7.21-log",
                    "args": ["/usr/local/mysql/bin/mysqld",
                             "--loose-audit-log-format=JSON",
                             "--log-error=log.err",
                             "--pid-file=mysqld.pid",
                             "--port=3306" ] } }
```

When the audit log plugin starts as a result of server startup (as opposed to being enabled at runtime), `connection_id` is set to 0, and `account` and `login` are not present.

Auditing shutdown event:

```
{ "timestamp": "2018-01-15 14:28:20",
  "id": 3,
  "class": "audit",
  "event": "shutdown",
  "connection_id": 0,
  "shutdown_data": { "server_id": 1 } }
```

When the audit log plugin is uninstalled as a result of server shutdown (as opposed to being disabled at runtime), `connection_id` is set to 0, and `account` and `login` are not present.

Connect or change-user event:

```
{ "timestamp": "2018-01-15 14:23:18",
  "id": 1,
```

```
  "class": "connection",
  "event": "connect",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "connection_data": { "connection_type": "ssl",
                       "status": 0,
                       "db": "test" } }
```

Disconnect event:

```
{ "timestamp": "2018-01-15 14:24:45",
  "id": 3,
  "class": "connection",
  "event": "disconnect",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "connection_data": { "connection_type": "ssl" } }
```

Query event:

```
{ "timestamp": "2018-01-15 14:23:35",
  "id": 2,
  "class": "general",
  "event": "status",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" },
  "general_data": { "command": "Query",
                    "sql_command": "show_variables",
                    "query": "SHOW VARIABLES",
                    "status": 0 } }
```

Table access event (read, delete, insert, update):

```
{ "timestamp": "2018-01-15 14:23:41",
  "id": 0,
  "class": "table_access",
  "event": "insert",
  "connection_id": 5,
  "account": { "user": "root", "host": "localhost" },
  "login": { "user": "root", "os": "", "ip": "127.0.0.1", "proxy": "" },
  "table_access_data": { "db": "test",
                         "table": "t1",
                         "query": "INSERT INTO t1 (i) VALUES(1),(2),(3)",
                         "sql_command": "insert" } }
```

The items in the following list appear at the top level of JSON-format audit records: Each item value is either a scalar or a `JSON` hash. For items that have a hash value, the description lists only the item names within that hash. For more complete descriptions of second-level hash items, see later in this section.

- `account`

  The MySQL account associated with the event. The value is a hash containing these items equivalent to the value of the `CURRENT_USER()` function within the section: `user`, `host`.

  Example:

  ```
  "account": { "user": "root", "host": "localhost" }
  ```

- `class`

  A string representing the event class. The class defines the type of event, when taken together with the `event` item that specifies the event subclass.

  Example:

  ```
  "class": "connection"
  ```

  The following table shows the permitted combinations of `class` and `event` values.

  Table 6.27 Audit Log Class and Event Combinations

Class Value	Permitted Event Values
`audit`	`startup`, `shutdown`
`connection`	`connect`, `change_user`, `disconnect`
`general`	`status`
`table_access_data`	`read`, `delete`, `insert`, `update`

- `connection_data`

  Information about a client connection. The value is a hash containing these items: `connection_type`, `status`, `db`. This item occurs only for audit records with a `class` value of `connection`.

  Example:

  ```
  "connection_data": { "connection_type": "ssl",
                       "status": 0,
                       "db": "test" }
  ```

- `connection_id`

  An unsigned integer representing the client connection identifier. This is the same as the value returned by the `CONNECTION_ID()` function within the session.

  Example:

  ```
  "connection_id": 5
  ```

- `event`

  A string representing the subclass of the event class. The subclass defines the type of event, when taken together with the `class` item that specifies the event class. For more information, see the `class` item description.

  Example:

  ```
  "event": "connect"
  ```

- `general_data`

  Information about an executed statement or command. The value is a hash containing these items: `command`, `sql_command`, `query`, `status`. This item occurs only for audit records with a `class` value of `general`.

Example:

```
"general_data": { "command": "Query",
                  "sql_command": "show_variables",
                  "query": "SHOW VARIABLES",
                  "status": 0 }
```

- `id`

  An unsigned integer representing an event ID.

  Example:

  ```
  "id": 2
  ```

  For audit records that have the same `timestamp` value, their `id` values distinguish them and form a sequence. Within the audit log, `timestamp`/`id` pairs are unique. These pairs are bookmarks that identify event locations within the log.

- `login`

  Information indicating how a client connected to the server. The value is a hash containing these items: `user`, `os`, `ip`, `proxy`.

  Example:

  ```
  "login": { "user": "root", "os": "", "ip": "::1", "proxy": "" }
  ```

- `shutdown_data`

  Information pertaining to audit log plugin termination. The value is a hash containing these items: `server_id` This item occurs only for audit records with `class` and `event` values of `audit` and `shutdown`, respectively.

  Example:

  ```
  "shutdown_data": { "server_id": 1 }
  ```

- `startup_data`

  Information pertaining to audit log plugin initialization. The value is a hash containing these items: `server_id`, `os_version`, `mysql_version`, `args`. This item occurs only for audit records with `class` and `event` values of `audit` and `startup`, respectively.

  Example:

  ```
  "startup_data": { "server_id": 1,
                    "os_version": "i686-Linux",
                    "mysql_version": "5.7.21-log",
                    "args": ["/usr/local/mysql/bin/mysqld",
                             "--loose-audit-log-format=JSON",
                             "--log-error=log.err",
                             "--pid-file=mysqld.pid",
                             "--port=3306" ] }
  ```

- `table_access_data`

Information about an access to a table. The value is a hash containing these items: `db`, `table`, `query`, `sql_command`, This item occurs only for audit records with a `class` value of `table_access`.

Example:

```
"table_access_data": { "db": "test",
                       "table": "t1",
                       "query": "INSERT INTO t1 (i) VALUES(1),(2),(3)",
                       "sql_command": "insert" }
```

- `timestamp`

  A string representing a UTC value in *`YYYY-MM-DD hh:mm:ss`* format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a `timestamp` value occurring after the statement finishes, not when it was received.

  Example:

  ```
  "timestamp": "2018-01-15 13:50:01"
  ```

  For audit records that have the same `timestamp` value, their `id` values distinguish them and form a sequence. Within the audit log, `timestamp`/`id` pairs are unique. These pairs are bookmarks that identify event locations within the log.

These items appear within hash values associated with top-level items of JSON-format audit records:

- `args`

  An array of options that were given on the command line or in option files when the MySQL server was started. The first option is the path to the server executable.

  Example:

  ```
  "args": ["/usr/local/mysql/bin/mysqld",
           "--loose-audit-log-format=JSON",
           "--log-error=log.err",
           "--pid-file=mysqld.pid",
           "--port=3306" ]
  ```

- `command`

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  Example:

  ```
  "command": "Query"
  ```

- `connection_type`

  The security state of the connection to the server. Permitted values are `tcp/ip` (TCP/IP connection established without encryption), `ssl` (TCP/IP connection established with encryption), `socket` (Unix socket file connection), `named_pipe` (Windows named pipe connection), and `shared_memory` (Windows shared memory connection).

Example:

```
"connection_type": "tcp/tcp"
```

- `db`

  A string representing a database name. For `connection_data`, it is the default database. For `table_access_data`, it is the table database.

  Example:

  ```
  "db": "test"
  ```

- `host`

  A string representing the client host name.

  Example:

  ```
  "host": "localhost"
  ```

- `ip`

  A string representing the client IP address.

  Example:

  ```
  "ip": "::1"
  ```

- `mysql_version`

  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable.

  Example:

  ```
  "mysql_version": "5.7.21-log"
  ```

- `os`

  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this attribute is empty. The value is the same as that of the `external_user` system variable. See Section 6.2.18, “Proxy Users”.

  Example:

  ```
  "os": "jeffrey"
  ```

- `os_version`

  A string representing the operating system on which the server was built or is running.

  Example:

```
"os_version": "i686-Linux"
```

- `proxy`

  A string representing the proxy user (see Section 6.2.18, “Proxy Users”). The value is empty if user proxying is not in effect.

  Example:

  ```
  "proxy": "developer"
  ```

- `query`

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example:

  ```
  "query": "DELETE FROM t1"
  ```

- `server_id`

  An unsigned integer representing the server ID. This is the same as the value of the `server_id` system variable.

  Example:

  ```
  "server_id": 1
  ```

- `sql_command`

  A string that indicates the SQL statement type.

  Example:

  ```
  "sql_command": "insert"
  ```

  The values correspond to the `statement/sql/xxx` command counters. For example, *`xxx`* is `drop_table` and `select` for `DROP TABLE` and `SELECT` statements, respectively. The following statement displays the possible names:

  ```
  SELECT REPLACE(EVENT_NAME, 'statement/sql/', '') AS name
  FROM performance_schema.events_statements_summary_global_by_event_name
  WHERE EVENT_NAME LIKE 'statement/sql/%'
  ORDER BY name;
  ```

- `status`

  An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function.

The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Section B.3.1, “Server Error Message Reference”.

Warnings are not logged.

Example:

```
"status": 1051
```

- `table`

  A string representing a table name.

  Example:

  ```
  "table": "t1"
  ```

- `user`

  A string representing a user name. The meaning differs depending on the item within which `user` occurs:

  - Within `account` items, `user` is a string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking.
  - Within `login` items, `user` is a string representing the user name sent by the client.

  Example:

  ```
  "user": "root"
  ```

6.4.5.5 Audit Log Logging Configuration

This section describes how to configure audit logging characteristics, such as the file to which the audit log plugin writes events, the format of written events, and whether to enable log file compression and encryption.

- Audit Log File Name
- Audit Log File Format
- Audit Log File Compression
- Audit Log File Encryption
- Audit Log File Manual Uncompression and Decryption
- Audit Log File Encryption Prior to MySQL 8.0.17
- Audit Log File Space Management and Name Rotation
- Audit Logging Write Strategy
- Audit Log File Reading

Note

Encryption capabilities described here apply as of MySQL 8.0.17, with the exception of the section that compares current encryption capabilities to the previous more-limited capabilities (see Audit Log File Encryption Prior to MySQL 8.0.17).

For additional information about the user-defined functions and system variables that affect audit logging, see Audit Log Functions, and Audit Log Options and Variables.

The audit log plugin can also control which audited events are written to the audit log file, based on event content or the account from which events originate. See Section 6.4.5.6, “Audit Log Filtering”.

Audit Log File Name

To configure the audit log file name, set the `audit_log_file` system variable at server startup. By default, the name is `audit.log` in the server data directory. For security reasons, write the audit log file to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log.

The plugin interprets the `audit_log_file` value as composed of a base name and an optional suffix. If compression or encryption are enabled, the effective file name (the name actually used to create the log file) differs from the configured file name because it has additional suffixes:

- If compression is enabled, the plugin adds a suffix of `.gz`.
- If encryption is enabled, the plugin adds a suffix of `.`*`pwd_id`*`.enc`, where *`pwd_id`* indicates which encryption password to use for log file operations.

The effective audit log file name is the name resulting from the addition of applicable compression and encryption suffixes to the configured file name. For example, if the configured `audit_log_file` value is `audit.log`, the effective file name is one of the values shown in the following table.

Enabled Features	Effective File Name
No compression or encryption	`audit.log`
Compression	`audit.log.gz`
Encryption	`audit.log.`*`pwd_id`*`.enc`
Compression, encryption	`audit.log.gz.`*`pwd_id`*`.enc`

`pwd_id` indicates the ID of the password used to encrypt or decrypt a file. *`pwd_id`* format is *`pwd_timestamp-seq`*, where:

- *`pwd_timestamp`* is a UTC value in *`YYYYMMDDThhmmss`* format indicating when the password was created.
- *`seq`* is a sequence number. Sequence numbers start at 1 and increase for passwords that have the same *`pwd_timestamp`* value.

Here are some example password ID values:

```
20190403T142359-1
20190403T142400-1
20190403T142400-2
```

The audit log plugin stores encryption passwords in the keyring (see Section 6.4.4, “The MySQL Keyring”). The IDs of audit log passwords in the keyring are based on *`pwd_id`* values, with a prefix of `audit_log-`. For the example password IDs just shown, the corresponding keyring IDs are:

```
audit_log-20190403T142359-1
audit_log-20190403T142400-1
audit_log-20190403T142400-2
```

The password currently used for encryption by the audit log plugin is the one having the largest *`pwd_timestamp`* value. If multiple passwords have the largest *`pwd_timestamp`* value, the current password is the one with the largest sequence number. For example, in the preceding set of password IDs, two of them have the largest timestamp, `20190403T142400`, so the current password is the one with the largest sequence number (2).

The audit log plugin performs certain actions during initialization and termination based on the effective audit log file name:

- During initialization, the plugin checks whether a file with the audit log file name already exists and renames it if so. (In this case, the plugin assumes that the previous server invocation exited unexpectedly with the audit log plugin running.) The plugin then writes to a new empty audit log file.
- During termination, the plugin renames the audit log file.
- File renaming (whether during plugin initialization or termination) occurs according to the usual rules for automatic log file rotation; see Automatic Audit Log File Rotation.

Audit Log File Format

To configure the audit log file format, set the `audit_log_format` system variable at server startup. By default, the format is `NEW` (new-style XML format). For details about each format, see Section 6.4.5.4, “Audit Log File Formats”.

If you change `audit_log_format`, it is recommended that you also change `audit_log_file`. Otherwise, there will be two sets of log files with the same base name but different formats.

Audit Log File Compression

Audit log file compression can be enabled for any logging format.

To configure audit log file compression, set the `audit_log_compression` system variable at server startup. Permitted values are `NONE` (no compression; the default) and `GZIP` (GNU Zip compression).

If both compression and encryption are enabled, compression occurs before encryption. To recover the original file manually, first decrypt it, then uncompress it. See Audit Log File Manual Uncompression and Decryption.

Audit Log File Encryption

Audit log file encryption can be enabled for any logging format. Encryption is based on user-defined passwords (with the exception of the initial password that the audit log plugin generates). To use this feature, the MySQL keyring must be enabled because audit logging uses it for password storage. Any keyring plugin can be used; for instructions, see Section 6.4.4, “The MySQL Keyring”.

To configure audit log file encryption, set the `audit_log_encryption` system variable at server startup. Permitted values are `NONE` (no encryption; the default) and `AES` (AES-256-CBC cipher encryption).

To set or get an encryption password, use these user-defined functions (UDFs):

- To set the current encryption password, invoke `audit_log_encryption_password_set()`. This function stores the new password in the keyring, If encryption is enabled, it also performs a log

file rotation operation that renames the current log file, and begins a new log file encrypted with the password. File renaming occurs according to the usual rules for automatic log file rotation; see Automatic Audit Log File Rotation.

- To get the current encryption password, invoke `audit_log_encryption_password_get()` with no argument. To get a password by ID, pass an argument specifying the keyring ID of the current password or an archived password.

  To determine which audit log keyring IDs exist, query the Performance Schema `keyring_keys` table:

```
mysql> SELECT * FROM performance_schema.keyring_keys
       WHERE KEY_ID LIKE 'audit_log%'
       ORDER BY KEY_ID;
+-----------------------------+-----------+----------------+
| KEY_ID                      | KEY_OWNER | BACKEND_KEY_ID |
+-----------------------------+-----------+----------------+
| audit_log-20190415T152248-1 |           |                |
| audit_log-20190415T153507-1 |           |                |
| audit_log-20190416T125122-1 |           |                |
| audit_log-20190416T141608-1 |           |                |
+-----------------------------+-----------+----------------+
```

For additional information about audit log encryption functions, see Audit Log Functions.

When the audit log plugin initializes, if it finds that log file encryption is enabled, it checks whether the keyring contains an audit log encryption password. If not, the plugin automatically generates a random initial encryption password and stores it in the keyring. To discover this password, invoke `audit_log_encryption_password_get()`.

If both compression and encryption are enabled, compression occurs before encryption. To recover the original file manually, first decrypt it, then uncompress it. See Audit Log File Manual Uncompression and Decryption.

Audit Log File Manual Uncompression and Decryption

Audit log files can be uncompressed and decrypted using standard tools. This should be done only for log files that have been closed (archived) and are no longer in use, not for the log file that the audit log plugin is currently writing. You can recognize archived log files because they have been renamed by the audit log plugin to include a timestamp in the file name just after the base name.

For this discussion, assume that `audit_log_file` is set to `audit.log`. In that case, an archived audit log file has one of the names shown in the following table.

Enabled Features	Archived File Name
No compression or encryption	`audit.`*`timestamp`*`.log`
Compression	`audit.`*`timestamp`*`.log.gz`
Encryption	`audit.`*`timestamp`*`.log.`*`pwd_id`*`.enc`
Compression, encryption	`audit.`*`timestamp`*`.log.gz.`*`pwd_id`*`.enc`

As discussed in Audit Log File Name, *`pwd_id`* format is *`pwd_timestamp-seq`*. Thus, the names of archived encrypted log files actually contain two timestamps. The first indicates rotation time, and the second indicates when the password was created.

Consider the following set of archived encrypted log file names:

```
audit.20190410T205827.log.20190403T185337-1.enc
audit.20190410T210243.log.20190403T185337-1.enc
audit.20190415T145309.log.20190414T223342-1.enc
audit.20190415T151322.log.20190414T223342-2.enc
```

Each file name has a unique rotation-time timestamp. By contrast, the password timestamps are not unique:

- The first two files have the same password ID and sequence number (`20190403T185337-1`). They have the same encryption password.
- The second two files have the same paswword ID (`20190414T223342`) but different sequence numbers (`1`, `2`). These files have different encryption passwords.

To uncompress a compressed log file manually, use `gunzip`, `gzip -d`, or equivalent command. For example:

```
gunzip -c audit.timestamp.log.gz > audit.timestamp.log
```

To decrypt an encrypted log file manually, use the `openssl` command. For example:

```
openssl enc -d -aes-256-cbc -pass pass:password -md sha256
    -in audit.timestamp.log.pwd_id.enc
    -out audit.timestamp.log
```

If both compression and encryption are enabled for audit logging, compression occurs before encryption. In this case, the file name has `.gz` and `.pwd_id.enc` suffixes added, corresponding to the order in which those operations occur. To recover the original file manually, perform the operations in reverse. That is, first decrypt the file, then uncompress it:

```
openssl enc -d -aes-256-cbc -pass pass:password -md sha256
    -in audit.timestamp.log.gz.pwd_id.enc
    -out audit.timestamp.log.gz
gunzip -c audit.timestamp.log.gz > audit.timestamp.log
```

Audit Log File Encryption Prior to MySQL 8.0.17

This section covers the differences in audit log file encryption capabilities prior to and as of MySQL 8.0.17, which is when password archiving and expiration were implemented. It also indicates how the audit log plugin handles upgrades to MySQL 8.0.17 or higher from versions lower than 8.0.17.

Feature	Prior to MySQL 8.0.17	As of MySQL 8.0.17
Number of passwords	Single password only	Multiple passwords permitted
Encrypted log file names	`.enc` suffix	`.pwd_id.enc` suffix
Password keyring ID	`audit_log`	`audit_log-pwd_id`
Password history	No	Yes
Password expiration	No	Yes

Prior to MySQL 8.0.17, there is no password history, so setting a new password makes the old password inaccessible, rendering MySQL Enterprise Audit unable to read log files encrypted with the old password. You must keep a record of the previous password should you need to decrypt those files manually.

If audit log file encryption is enabled when you upgrade to MySQL 8.0.17 or higher from a lower version, the audit log plugin performs these upgrade actions:

- During plugin initialization, the plugin checks for an encryption password with a keyring ID of `audit_log`. If it finds one, the plugin duplicates the password using a keyring ID in `audit_log-pwd_id` format and uses it as the current encryption password.
- Existing encrypted log files have a suffix of `.enc`. The plugin does not rename these to have a suffix of `.pwd_id.enc`, but can read them as long as the key with the ID of `audit_log` remains in the keyring.
- When password cleanup occurs, if the plugin expires any password with a keyring ID in `audit_log-pwd_id` format, it also expires the password with a keyring ID of `audit_log`, if it exists. (At this point, encrypted log files that have a suffix of `.enc` rather than `.pwd_id.enc` become unreadable by the plugin, so it is assumed that you no longer need them.)

Audit Log File Space Management and Name Rotation

The audit log file has the potential to grow very large and consume a lot of disk space. To enable management of the space used by its log files, the audit log plugin provides for log file rotation, either manually or automatically. Rotation capabilities use the `audit_log_flush` and `audit_log_rotate_on_size` system variables:

- By default, `audit_log_rotate_on_size=0` and no log rotation occurs unless performed manually. In this case, use `audit_log_flush` to close and reopen the current log file after manually renaming it.
- If `audit_log_rotate_on_size` is greater than 0, automatic rotation occurs when a write to the current log file causes its size to exceed this value. The audit log plugin closes the file, renames it, and opens a new log file. With automatic rotation enabled, `audit_log_flush` has no effect.
- Automatic rotation also occurs under several other conditions, described later.

Note

Renamed log files are not removed automatically. For example, with size-based log file rotation, renamed log files do not rotate off the end of the name sequence. Instead, they have unique names and accumulate indefinitely. To avoid excessive space use, remove old files periodically, backing them up first as necessary. If backed-up log files are encrypted, also back up the corresponding encryption passwords to a safe place, should you need to decrypt the files later.

The following discussion describes log file rotation methods in greater detail.

- Manual Audit Log File Rotation
- Automatic Audit Log File Rotation

Manual Audit Log File Rotation

If `audit_log_rotate_on_size=0` (the default), no log rotation occurs unless performed manually. In this case, the audit log plugin closes and reopens the log file when the `audit_log_flush` value changes from disabled to enabled. Log file renaming must be done externally to the server. Suppose that the log file name is `audit.log` and you want to maintain the three most recent log files, cycling through the names `audit.log.1` through `audit.log.3`. On Unix, perform rotation manually like this:

1. From the command line, rename the current log files:

```
mv audit.log.2 audit.log.3
mv audit.log.1 audit.log.2
mv audit.log audit.log.1
```

This strategy overwrites the current `audit.log.3` contents, placing a bound on the number of archived log files and the space they use.

2. At this point, the plugin is still writing to the current log file, which has been renamed to `audit.log.1`. Connect to the server and flush the log file so the plugin closes it and reopens a new `audit.log` file:

```
SET GLOBAL audit_log_flush = ON;
```

`audit_log_flush` is special in that its value remains `OFF` so that you need not disable it explicitly before enabling it again to perform another flush.

Note

If compression or encryption are enabled, log file names include suffixes that signify the enabled features, as well as a password ID if encryption is enabled. If file names include a password ID, be sure to retain the ID in the name of any files you rename manually so that the password to use for decryption operations can be determined.

Note

For JSON-format logging, renaming audit log files manually makes them unavailable to the log-reading functions because the audit log plugin no longer can determine that they are part of the log file sequence (see Audit Log File Reading). Consider setting `audit_log_rotate_on_size` greater than 0 to use size-based rotation instead.

Automatic Audit Log File Rotation

If `audit_log_rotate_on_size` is greater than 0, setting `audit_log_flush` has no effect. Instead, whenever a write to the current log file causes its size to exceed the `audit_log_rotate_on_size` value, the audit log plugin closes the file, renames it, and opens a new log file.

Automatic rotation also occurs under these conditions:

- During plugin initialization, if a file with the audit log file name already exists (see Audit Log File Name).
- During plugin termination.
- When the `audit_log_encryption_password_set()` function is called to set the encryption password, if encryption is enabled. (Rotation does not occur if encryption is disabled.)

The plugin renames the original file by inserting a timestamp just after its base name. For example, if the file name is `audit.log`, the plugin renames it to a value such as `audit.20180115T140633.log`. The timestamp is a UTC value in *`YYYYMMDDThhmmss`* format. The timestamp indicates rotation time for XML logging, and the timestamp of the last event written to the file for JSON logging.

Audit Logging Write Strategy

The audit log plugin can use any of several strategies for log writes. Regardless of strategy, logging occurs on a best-effort basis, with no guarantee of consistency.

To specify a write strategy, set the `audit_log_strategy` system variable at server startup. By default, the strategy value is `ASYNCHRONOUS` and the plugin logs asynchronously to a buffer, waiting if the buffer is full. It's possible to tell the plugin not to wait (`PERFORMANCE`) or to log synchronously, either using file system caching (`SEMISYNCHRONOUS`) or forcing output with a `sync()` call after each write request (`SYNCHRONOUS`).

For asynchronous write strategy, the `audit_log_buffer_size` system variable is the buffer size in bytes. Set this variable at server startup to change the buffer size. The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin does not allocate this buffer for nonasynchronous write strategies.

Asynchronous logging strategy has these characteristics:

- Minimal impact on server performance and scalability.
- Blocking of threads that generate audit events for the shortest possible time; that is, time to allocate the buffer plus time to copy the event to the buffer.
- Output goes to the buffer. A separate thread handles writes from the buffer to the log file.

With asynchronous logging, the integrity of the log file may be compromised if a problem occurs during a write to the file or if the plugin does not shut down cleanly (for example, in the event that the server host exits unexpectedly). To reduce this risk, set `audit_log_strategy` to use synchronous logging.

A disadvantage of `PERFORMANCE` strategy is that it drops events when the buffer is full. For a heavily loaded server, the audit log may have events missing.

Audit Log File Reading

The audit log plugin enables bookmarking and reading of JSON-format audit log files. (These capabilities do not apply to files written in other log formats.)

When the audit log plugin initializes and is configured for JSON logging, it uses the directory containing the audit log file (determined from the `audit_log_file` value) as the location to search for readable audit log files. To do this, it uses the value of `audit_log_file` to determine the file base name and suffix values, then looks for files with names that match the following pattern, where `[...]` indicates optional file name parts:

```
basename[.timestamp].suffix[.gz][[.pwd_id].enc]
```

A file is encrypted if its name ends with `.enc`. The plugin determines the keyring ID of the decryption password needed to read it as follows:

- If `.enc` is preceded by *`pwd_id`*, the keyring ID is `audit_log-`*`pwd_id`*.
- If `.enc` is not preceded by *`pwd_id`*, the file has an old name from before password history was implemented. The keyring ID is `audit_log`.

The plugin opens each file matching the pattern, checks that the file really contains JSON audit records, and sorts the files using the timestamps from the first record of each file to construct a list of files that are subject to use with the log-reading functions.

The plugin cannot include in the sequence files that were renamed manually and do not match the preceding pattern, or that were encrypted with a password no longer available in the keyring.

To read events from the audit log, use these user-defined functions (UDFs):

- `audit_log_read_bookmark()` returns a `JSON` string representing a bookmark for the most recently written audit log event. This bookmark is suitable for passing to `audit_log_read()` to indicate to that function where to begin reading. Example bookmark:

  ```
  { "timestamp": "2018-01-15 21:03:44", "id": 0 }
  ```

- `audit_log_read()` reads events from the audit log and returns a `JSON` string containing an array of audit events.

Example `audit_log_read()` invocation using the current bookmark:

```
mysql> SELECT audit_log_read(audit_log_read_bookmark());
+------------------------------------------------------------------------+
| audit_log_read(audit_log_read_bookmark())                              |
+------------------------------------------------------------------------+
| [ {"timestamp":"2018-01-15 22:41:24","id":0,"class":"connection", ... |
+------------------------------------------------------------------------+
```

Each event in the `audit_log_read()` return value is a `JSON` hash, except that the last array element may be a `JSON` `null` value to indicate no following events are available to read. For example:

```
[
  { "timestamp": "2018-01-15 22:08:08", "id": 10,
    "class": "general", "event": "status",
    ...
  },
  {
    "timestamp": "2018-01-15 22:08:08", "id": 11,
    "class": "connection", "event": "disconnect",
    ...
  },
  {
    "timestamp": "2018-01-15 13:39:33", "id": 0,
    "class": "connection", "event": "connect",
    ...
  },
  {
    "timestamp": "2018-01-15 13:39:33", "id": 1,
    "class": "general", "event": "status",
    ...
  },
  {
    "timestamp": "2018-01-15 13:39:33", "id": 2,
    "class": "connection", "event": "disconnect",
    ...
  },
  null
]
```

Use `audit_log_read()` like this:

- For the first call to `audit_log_read()` within a session, pass a bookmark indicating where to begin reading.
- If the final value of the returned array is not a `JSON` `null` value, there are more events following those just read and `audit_log_read()` can be called without or with a bookmark argument. Without an argument, reading continues with the next unread event. With a bookmark argument, reading continues from the bookmark.
- If the final value of the returned array is a `JSON` `null` value, there are no more events left to be read and the next call to `audit_log_read()` must include a bookmark argument.

A bookmark is a `JSON` hash that indicates where and how much to read. The following items are significant in the bookmark value (other items are ignored):

- `timestamp`, `id`: The location within the audit log of the first event to read. Both items must be present to completely specify a position.

- `max_array_length`: The maximum number of events to read from the log. If omitted, the default is to read to the end of the log or until the read buffer is full, whichever comes first.

The result returned from either log-reading function is a binary string. To use the string with functions that require a nonbinary string (such as the functions that manipulate `JSON` values), convert it to `utf8mb4`. Suppose that a bookmark has this value:

```
mysql> SELECT @mark := audit_log_read_bookmark() AS mark;
+-----------------------------------------------+
| mark                                          |
+-----------------------------------------------+
| { "timestamp": "2018-01-15 16:10:28", "id": 2 } |
+-----------------------------------------------+
```

Calling `audit_log_read()` with that bookmark can return multiple events. To set a limit on the number of events read by `audit_log_read()`, convert the bookmark to `utf8mb4`, then add to it a `max_array_length` item with a value of 1. For example, using the preceding bookmark, convert and modify it as follows:

```
mysql> SET @mark = CONVERT(@mark USING utf8mb4);
mysql> SET @mark := JSON_SET(@mark, '$.max_array_length', 1);
mysql> SELECT @mark;
+-------------------------------------------------------------------------+
| @mark                                                                   |
+-------------------------------------------------------------------------+
| {"id": 2, "timestamp": "2018-01-15 16:10:28", "max_array_length": 1} |
+-------------------------------------------------------------------------+
```

The modified bookmark, when passed to `audit_log_read()`, produces a result of a single audit record.

To set a limit on the number of bytes that `audit_log_read()` reads, set the `audit_log_read_buffer_size` system variable. As of MySQL 8.0.12, this variable has a default of 32KB and can be set at runtime. Each client should set its session value of `audit_log_read_buffer_size` appropriately for its use of `audit_log_read()`. Prior to MySQL 8.0.12, `audit_log_read_buffer_size` has a default of 1MB, affects all clients, and can be changed only at server startup.

Each call to `audit_log_read()` returns as many available items as fit within the buffer size, skipping items that do not fit within the buffer size. Given this behavior, consider these factors when assessing the proper buffer size for an application:

- There is a tradeoff between number of calls to `audit_log_read()` and items returned per call. With a smaller buffer size, calls return fewer items, so more calls are needed. With a larger buffer size, calls return more items, so fewer calls are needed.
- With a smaller buffer size, such as the default size of 32KB, there is a greater chance that items will exceed the buffer size and `audit_log_read()` will skip them. Skipped items generate warnings.

For additional information about audit log-reading functions, see Audit Log Functions.

6.4.5.6 Audit Log Filtering

Note

This section describes how audit log filtering works if the audit log plugin and the accompanying audit tables and UDFs are installed. If the plugin is installed

but not the accompanying audit tables and UDFs, the plugin operates in legacy filtering mode, described in Section 6.4.5.8, “Legacy Mode Audit Log Filtering”. Legacy mode is filtering behavior as it was prior to MySQL 5.7.13; that is, before the introduction of rule-based filtering.

The audit log plugin has the capability of controlling logging of audited events by filtering them:

- Audited events can be filtered using these characteristics:
  - User account
  - Audit event class
  - Audit event subclass
  - Value of event fields such as those that indicate operation status or SQL statement executed
- Audit filtering is rule based:
  - A filter definition creates a set of auditing rules. Definitions can be configured to include or exclude events for logging based on the characteristics just described.
  - Filter rules have the capability of blocking (aborting) execution of qualifying events, in addition to existing capabilities for event logging.
  - Multiple filters can be defined, and any given filter can be assigned to any number of user accounts.
  - It is possible to define a default filter to use with any user account that has no explicitly assigned filter.
- Audit filters can be defined, displayed, and modified using an SQL interface based on user-defined functions (UDFs).
- Audit filter definitions are stored in the tables in the `mysql` system database.
- Within a given session, the value of the read-only `audit_log_filter_id` system variable indicates whether a filter has been assigned to the session.

Note

By default, rule-based audit log filtering logs no auditable events for any users. To log all auditable events for all users, use the following statements, which create a simple filter to enable logging and assign it to the default account:

```
SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');
SELECT audit_log_filter_set_user('%', 'log_all');
```

The filter assigned to `%` is used for connections from any account that has no explicitly assigned filter (which initially is true for all accounts).

The following list briefly summarizes the UDFs that implement the SQL interface for audit filtering control:

- `audit_log_filter_set_filter()`: Define a filter
- `audit_log_filter_remove_filter()`: Remove a filter
- `audit_log_filter_set_user()`: Start filtering a user account
- `audit_log_filter_remove_user()`: Stop filtering a user account

- `audit_log_filter_flush()`: Flush manual changes to the filter tables to affect ongoing filtering

For usage examples and complete details about the filtering functions, see Using Audit Log Filtering Functions, and Audit Log Functions.

Audit log filtering functions are subject to these constraints:

- To use any filtering function, the `audit_log` plugin must be enabled or an error occurs. In addition, the audit tables must exist or an error occurs. To install the `audit_log` plugin and its accompanying UDFs and tables, see Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”.
- To use any filtering function, a user must possess the `SUPER` privilege or an error occurs. To grant the `SUPER` privilege to a user account, use this statement:

```
GRANT SUPER ON *.* TO user;
```

  Alternatively, should you prefer to avoid granting the `SUPER` privilege while still permitting users to access specific filtering functions, “wrapper” stored programs can be defined. This technique is described in the context of keyring UDFs in Using General-Purpose Keyring Functions; it can be adapted for use with filtering UDFs.

- The `audit_log` plugin operates in legacy mode if it is installed but the accompanying audit tables and functions are not created. The plugin writes these messages to the error log at server startup:

```
[Warning] Plugin audit_log reported: 'Failed to open the audit log filter tables.'
[Warning] Plugin audit_log reported: 'Audit Log plugin supports a filtering,
which has not been installed yet. Audit Log plugin will run in the legacy
mode, which will be disabled in the next release.'
```

  In legacy mode, filtering can be done based only on event account or status. For details, see Section 6.4.5.8, “Legacy Mode Audit Log Filtering”.

- Using Audit Log Filtering Functions

Using Audit Log Filtering Functions

Before using the audit log user-defined functions (UDFs), install them according to the instructions provided in Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”. The `SUPER` privilege is required to use any of these functions.

The audit log filtering functions enable filtering control by providing an interface to create, modify, and remove filter definitions and assign filters to user accounts.

Filter definitions are `JSON` values. For information about using `JSON` data in MySQL, see Section 11.6, “The JSON Data Type”. This section shows some simple filter definitions. For more information about filter definitions, see Section 6.4.5.7, “Writing Audit Log Filter Definitions”.

When a connection arrives, the audit log plugin determines which filter to use for the new session by searching for the user account name in the current filter assignments:

- If a filter is assigned to the user, the audit log uses that filter.
- Otherwise, if no user-specific filter assignment exists, but there is a filter assigned to the default account (`%`), the audit log uses the default filter.
- Otherwise, the audit log selects no audit events from the session for processing.

If a change-user operation occurs during a session (see Section 28.7.7.3, “mysql_change_user()”), filter assignment for the session is updated using the same rules but for the new user.

By default, no accounts have a filter assigned, so no processing of auditable events occurs for any account.

Suppose that instead you want the default to be to log only connection-related activity (for example, to see connect, change-user, and disconnect events, but not the SQL statements users execute while connected). To achieve this, define a filter (shown here named `log_conn_events`) that enables logging only of events in the `connection` class, and assign that filter to the default account, represented by the `%` account name:

```
SET @f = '{ "filter": { "class": { "name": "connection" } } }';
SELECT audit_log_filter_set_filter('log_conn_events', @f);
SELECT audit_log_filter_set_user('%', 'log_conn_events');
```

Now the audit log uses this default account filter for connections from any account that has no explicitly defined filter.

To assign a filter explicitly to a particular user account or accounts, define the filter, then assign it to the relevant accounts:

```
SELECT audit_log_filter_set_filter('log_all', '{ "filter": { "log": true } }');
SELECT audit_log_filter_set_user('user1@localhost', 'log_all');
SELECT audit_log_filter_set_user('user2@localhost', 'log_all');
```

Now full logging is enabled for `user1@localhost` and `user2@localhost`. Connections from other accounts continue to be filtered using the default account filter.

To disassociate a user account from its current filter, either unassign the filter or assign a different filter:

- To unassign the filter from the user account:

  ```
  SELECT audit_log_filter_remove_user('user1@localhost');
  ```

  Filtering of current sessions for the account remains unaffected. Subsequent connections from the account are filtered using the default account filter if there is one, and are not logged otherwise.

- To assign a different filter to the user account:

  ```
  SELECT audit_log_filter_set_filter('log_nothing', '{ "filter": { "log": false } }');
  SELECT audit_log_filter_set_user('user1@localhost', 'log_nothing');
  ```

  Filtering of current sessions for the account remains unaffected. Subsequent connections from the account are filtered using the new filter. For the filter shown here, that means no logging for new connections from `user1@localhost`.

For audit log filtering, user name and host name comparisons are case-sensitive. This differs from comparisons for privilege checking, for which host name comparisons are not case-sensitive.

To remove a filter, do this:

```
SELECT audit_log_filter_remove_filter('log_nothing');
```

Removing a filter also unassigns it from any users to whom it has been assigned, including any current sessions for those users.

The filtering UDFs just described affect audit filtering immediately and update the audit log tables in the `mysql` system database that store filters and user accounts (see Audit Log Tables). It is also possible to modify the audit log tables directly using statements such as `INSERT`, `UPDATE`, and `DELETE`, but such changes do not affect filtering immediately. To flush your changes and make them operational, call `audit_log_filter_flush()`:

```
SELECT audit_log_filter_flush();
```

Warning

`audit_log_filter_flush()` should be used only after modifying the audit tables directly, to force reloading all filters. Otherwise, this function should be avoided. It is, in effect, a simplified version of unloading and reloading the `audit_log` plugin with `UNINSTALL PLUGIN` plus `INSTALL PLUGIN`.

`audit_log_filter_flush()` affects all current sessions and detaches them from their previous filters. Current sessions are no longer logged unless they disconnect and reconnect, or execute a change-user operation.

To determine whether a filter has been assigned to the current session, check the session value of the read-only `audit_log_filter_id` system variable. If the value is 0, no filter is assigned. A nonzero value indicates the internally maintained ID of the assigned filter:

```
mysql> SELECT @@audit_log_filter_id;
+-----------------------+
| @@audit_log_filter_id |
+-----------------------+
|                     2 |
+-----------------------+
```

6.4.5.7 Writing Audit Log Filter Definitions

Filter definitions are `JSON` values. For information about using `JSON` data in MySQL, see Section 11.6, “The JSON Data Type”.

Filter definitions have this form, where *`actions`* indicates how filtering takes place:

```
{ "filter": actions }
```

The following discussion describes permitted constructs in filter definitions.

- Logging All Events
- Logging Specific Event Classes
- Logging Specific Event Subclasses
- Inclusive and Exclusive Logging
- Testing Event Field Values
- Blocking Execution of Specific Events
- Logical Operators
- Referencing Predefined Variables

- Referencing Predefined Functions
- Replacing a User Filter

Logging All Events

To explicitly enable or disable logging of all events, use a `log` element in the filter:

```
{
  "filter": { "log": true }
}
```

The `log` value can be either `true` or `false`.

The preceding filter enables logging of all events. It is equivalent to:

```
{
  "filter": { }
}
```

Logging behavior depends on the `log` value and whether `class` or `event` items are specified:

- With `log` specified, its given value is used.
- Without `log` specified, logging is `true` if no `class` or `event` item is specified, and `false` otherwise (in which case, `class` or `event` can include their own `log` item).

Logging Specific Event Classes

To log events of a specific class, use a `class` element in the filter, with its `name` field denoting the name of the class to log:

```
{
  "filter": {
    "class": { "name": "connection" }
  }
}
```

The `name` value can be `connection`, `general`, or `table_access` to log connection, general, or table-access events, respectively.

The preceding filter enables logging of events in the `connection` class. It is equivalent to the following filter with `log` items made explicit:

```
{
  "filter": {
    "log": false,
    "class": { "log": true,
               "name": "connection" }
  }
}
```

To enable logging of multiple classes, define the `class` value as a `JSON` array element that names the classes:

```
```

```
{
  "filter": {
    "class": [
      { "name": "connection" },
      { "name": "general" },
      { "name": "table_access" }
    ]
  }
}
```

Note

When multiple instances of a given item appear at the same level within a filter definition, the item values can be combined into a single instance of that item within an array value. The preceding definition can be written like this:

```
{
  "filter": {
    "class": [
      { "name": [ "connection", "general", "table_access" ] }
    ]
  }
}
```

Logging Specific Event Subclasses

To select specific event subclasses, use an `event` item containing a `name` item that names the subclasses. The default action for events selected by an `event` item is to log them. For example, this filter enables logging for the named event subclasses:

```
{
  "filter": {
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect" },
          { "name": "disconnect" }
        ]
      },
      { "name": "general" },
      {
        "name": "table_access",
        "event": [
          { "name": "insert" },
          { "name": "delete" },
          { "name": "update" }
        ]
      }
    ]
  }
}
```

The `event` item can also contain explicit `log` items to indicate whether to log qualifying events. This `event` item selects multiple events and explicitly indicates logging behavior for them:

```
"event": [
  { "name": "read", "log": false },
  { "name": "insert", "log": true },
  { "name": "delete", "log": true },
  { "name": "update", "log": true }
```

```
]
```

The `event` item can also indicate whether to block qualifying events, if it contains an `abort` item. For details, see Blocking Execution of Specific Events.

Table 6.28, "Event Class and Subclass Combinations" describes the permitted subclass values for each event class.

Table 6.28 Event Class and Subclass Combinations

Event Class	Event Subclass	Description
`connection`	`connect`	Connection initiation (successful or unsuccessful)
`connection`	`change_user`	User re-authentication with different user/password during session
`connection`	`disconnect`	Connection termination
`general`	`status`	General operation information
`message`	`internal`	Internally generated message
`message`	`user`	Message generated by `audit_api_message_emit_udf()`
`table_access`	`read`	Table read statements, such as `SELECT` or `INSERT INTO ... SELECT`
`table_access`	`delete`	Table delete statements, such as `DELETE` or `TRUNCATE TABLE`
`table_access`	`insert`	Table insert statements, such as `INSERT` or `REPLACE`
`table_access`	`update`	Table update statements, such as `UPDATE`

Table 6.29, "Log and Abort Characteristics Per Event Class and Subclass Combination" describes for each event subclass whether it can be logged or aborted.

Table 6.29 Log and Abort Characteristics Per Event Class and Subclass Combination

Event Class	Event Subclass	Can be Logged	Can be Aborted
`connection`	`connect`	Yes	No
`connection`	`change_user`	Yes	No
`connection`	`disconnect`	Yes	No
`general`	`status`	Yes	No
`message`	`internal`	Yes	Yes
`message`	`user`	Yes	Yes
`table_access`	`read`	Yes	Yes
`table_access`	`delete`	Yes	Yes
`table_access`	`insert`	Yes	Yes
`table_access`	`update`	Yes	Yes

Inclusive and Exclusive Logging

A filter can be defined in inclusive or exclusive mode:

- Inclusive mode logs only explicitly specified items.

- Exclusive mode logs everything but explicitly specified items.

To perform inclusive logging, disable logging globally and enable logging for specific classes. This filter logs `connect` and `disconnect` events in the `connection` class, and events in the `general` class:

```
{
  "filter": {
    "log": false,
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect", "log": true },
          { "name": "disconnect", "log": true }
        ]
      },
      { "name": "general", "log": true }
    ]
  }
}
```

To perform exclusive logging, enable logging globally and disable logging for specific classes. This filter logs everything except events in the `general` class:

```
{
  "filter": {
    "log": true,
    "class":
      { "name": "general", "log": false }
  }
}
```

This filter logs `change_user` events in the `connection` class, `message` events, and `table_access` events:

```
{
  "filter": {
    "log": true,
    "class": [
      {
        "name": "connection",
        "event": [
          { "name": "connect", "log": false },
          { "name": "disconnect", "log": false }
        ]
      },
      { "name": "general", "log": false }
    ]
  }
}
```

Testing Event Field Values

To enable logging based on specific event field values, specify a `field` item within the `log` item that indicates the field name and its expected value:

```
{
  "filter": {
    "class": {
    "name": "general",
      "event": {
```

```
        "name": "status",
        "log": {
          "field": { "name": "general_command.str", "value": "Query" }
        }
      }
    }
  }
}
```

Each event contains event class-specific fields that can be accessed from within a filter to perform custom filtering.

A connection event indicates when a connection-related activity occurs during a session, such as a user connecting to or disconnecting from the server. Table 6.30, “Connection Event Fields” indicates the permitted fields for connection events.

Table 6.30 Connection Event Fields

Field Name	Field Type	Description
`status`	integer	Event status: 0: OK Otherwise: Failed
`connection_id`	unsigned integer	Connection ID
`user.str`	string	User name specified during authentication
`user.length`	unsigned integer	User name length
`priv_user.str`	string	Authenticated user name (account user name)
`priv_user.length`	unsigned integer	Authenticated user name length
`external_user.str`	string	External user name (provided by third-party authentication plugin)
`external_user.length`	unsigned integer	External user name length
`proxy_user.str`	string	Proxy user name
`proxy_user.length`	unsigned integer	Proxy user name length
`host.str`	string	Connected user host
`host.length`	unsigned integer	Connected user host length
`ip.str`	string	Connected user IP address
`ip.length`	unsigned integer	Connected user IP address length
`database.str`	string	Database name specified at connect time
`database.length`	unsigned integer	Database name length
`connection_type`	integer	Connection type: or `"::undefined"`: Undefined or `"::tcp/ip"`: TCP/IP or `"::socket"`: Socket or `"::named_pipe"`: Named pipe or `"::ssl"`: TCP/IP with encryption

Field Name	Field Type	Description
		or "::shared_memory": Shared memory

The "::xxx" values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

A general event indicates the status code of an operation and its details. Table 6.31, "General Event Fields" indicates the permitted fields for general events.

Table 6.31 General Event Fields

Field Name	Field Type	Description
general_error_code	integer	Event status: 0: OK Otherwise: Failed
general_thread_id	unsigned integer	Connection/thread ID
general_user.str	string	User name specified during authentication
general_user.length	unsigned integer	User name length
general_command.str	string	Command name
general_command.length	unsigned integer	Command name length
general_query.str	string	SQL statement text
general_query.length	unsigned integer	SQL statement text length
general_host.str	string	Host name
general_host.length	unsigned integer	Host name length
general_sql_command.str	string	SQL command type name
general_sql_command.length	unsigned integer	SQL command type name length
general_external_user.str	string	External user name (provided by third-party authentication plugin)
general_external_user.length	unsigned integer	External user name length
general_ip.str	string	Connected user IP address
general_ip.length	unsigned integer	Connection user IP address length

general_command.str indicates a command name: Query, Execute, Quit, or Change user.

A general event with the general_command.str field set to Query or Execute contains general_sql_command.str set to a value that specifies the type of SQL command: alter_db, alter_db_upgrade, admin_commands, and so forth. These values can be seen as the last components of the Performance Schema instruments displayed by this statement:

```
mysql> SELECT NAME FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'statement/sql/%' ORDER BY NAME;
+---------------------------------------+
| NAME                                  |
+---------------------------------------+
| statement/sql/alter_db                |
| statement/sql/alter_db_upgrade        |
```

```
| statement/sql/alter_event              |
| statement/sql/alter_function           |
| statement/sql/alter_instance           |
| statement/sql/alter_procedure          |
| statement/sql/alter_server             |
...
```

A table-access event provides information about specific table accesses. Table 6.32, “Table-Access Event Fields” indicates the permitted fields for table-access events.

Table 6.32 Table-Access Event Fields

Field Name	Field Type	Description
`connection_id`	unsigned integer	Event connection ID
`sql_command_id`	integer	SQL command ID
`query.str`	string	SQL statement text
`query.length`	unsigned integer	SQL statement text length
`table_database.str`	string	Database name associated with event
`table_database.length`	unsigned integer	Database name length
`table_name.str`	string	Table name associated with event
`table_name.length`	unsigned integer	Table name length

The following list shows which statements produce which table-access events:

- `read` event:
  - `SELECT`
  - `INSERT ... SELECT` (for tables referenced in `SELECT` clause)
  - `REPLACE ... SELECT` (for tables referenced in `SELECT` clause)
  - `UPDATE ... WHERE` (for tables referenced in `WHERE` clause)
  - `HANDLER ... READ`
- `delete` event:
  - `DELETE`
  - `TRUNCATE TABLE`
- `insert` event:
  - `INSERT`
  - `INSERT ... SELECT` (for table referenced in `INSERT` clause)
  - `REPLACE`
  - `REPLACE ... SELECT` (for table referenced in `REPLACE` clause
  - `LOAD DATA`
  - `LOAD XML`

- `update` event:
  - `UPDATE`
  - `UPDATE ... WHERE` (for tables referenced in `UPDATE` clause)

Blocking Execution of Specific Events

`event` items can include an `abort` item that indicates whether to prevent qualifying events from executing. For example, `abort` enables rules to be written that block execution of specific SQL statements.

The `abort` item must appear within an `event` item. For example:

```
"event": {
  "name": qualifying event subclass names
  "abort": condition
}
```

For event subclasses selected by the `name` item, the `abort` action is true or false, depending on *`condition`* evaluation. If the condition evaluates to true, the event is blocked. Otherwise, the event continues executing.

The *`condition`* specification can be as simple as `true` or `false`, or it can be more complex such that evaluation depends on event characteristics.

This filter blocks `INSERT`, `UPDATE`, and `DELETE` statements:

```
{
  "filter": {
    "class": {
      "name": "table_access",
      "event": {
        "name": [ "insert", "update", "delete" ],
        "abort": true
      }
    }
  }
}
```

This more complex filter blocks the same statements, but only for a specific table (`finances.bank_account`):

```
{
  "filter": {
    "class": {
      "name": "table_access",
      "event": {
        "name": [ "insert", "update", "delete" ],
        "abort": {
          "and": [
            { "field": { "name": "table_database.str", "value": "finances" } },
            { "field": { "name": "table_name.str", "value": "bank_account" } }
          ]
        }
      }
    }
  }
}
```

Statements matched and blocked by the filter return an error to the client:

```
ERROR 1045 (28000): Statement was aborted by an audit log filter
```

Not all events can be blocked (see Table 6.29, “Log and Abort Characteristics Per Event Class and Subclass Combination”). For an event that cannot, the audit log writes a warning to the error log rather than blocking it.

For attempts to define a filter in which the `abort` item appears elsewhere than in an `event` item, an error occurs.

Logical Operators

Logical operators (`and`, `or`, `not`) can be used in `log` items. This permits construction of more advanced filtering configurations:

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "or": [
            {
              "and": [
                { "field": { "name": "general_command.str",    "value": "Query" } },
                { "field": { "name": "general_command.length", "value": 5 } }
              ]
            },
            {
              "and": [
                { "field": { "name": "general_command.str",    "value": "Execute" } },
                { "field": { "name": "general_command.length", "value": 7 } }
              ]
            }
          ]
        }
      }
    }
  }
}
```

Referencing Predefined Variables

To refer to a predefined variable in a `log` condition, use a `variable` item, which tests equality against a given value:

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "variable": {
            "name": "audit_log_connection_policy_value", "value": "::none"
          }
        }
      }
    }
```

```
  }
}
```

Each predefined variable corresponds to a system variable. By writing a filter that tests a predefined variable, you can modify filter operation by setting the corresponding system variable, without having to redefine the filter. For example, by writing a filter that tests the value of the `audit_log_connection_policy_value` predefined variable, you can modify filter operation by changing the value of the `audit_log_connection_policy` system variable.

The `audit_log_xxx_policy` system variables are used for the legacy mode audit log (see Section 6.4.5.8, “Legacy Mode Audit Log Filtering”). With rule-based audit log filtering, those variables remain visible (for example, using `SHOW VARIABLES`), but changes to them have no effect unless you write filters containing constructs that refer to them.

The following list describes the permitted predefined variables for `variable` items:

- `audit_log_connection_policy_value`

  This variable corresponds to the value of the `audit_log_connection_policy` system variable. The value is an unsigned integer. Table 6.33, “audit_log_connection_policy_value Values” shows the permitted values and the corresponding `audit_log_connection_policy` values.

  Table 6.33 audit_log_connection_policy_value Values

Value	Corresponding audit_log_connection_policy Value
`0` or `"::none"`	`NONE`
`1` or `"::errors"`	`ERRORS`
`2` or `"::all"`	`ALL`

  The `"::xxx"` values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

- `audit_log_policy_value`

  This variable corresponds to the value of the `audit_log_policy` system variable. The value is an unsigned integer. Table 6.34, “audit_log_policy_value Values” shows the permitted values and the corresponding `audit_log_policy` values.

  Table 6.34 audit_log_policy_value Values

Value	Corresponding audit_log_policy Value
`0` or `"::none"`	`NONE`
`1` or `"::logins"`	`LOGINS`
`2` or `"::all"`	`ALL`
`3` or `"::queries"`	`QUERIES`

  The `"::xxx"` values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

- `audit_log_statement_policy_value`

  This variable corresponds to the value of the `audit_log_statement_policy` system variable. The value is an unsigned integer. Table 6.35, “audit_log_statement_policy_value Values” shows the permitted values and the corresponding `audit_log_statement_policy` values.

Table 6.35 audit_log_statement_policy_value Values

Value	Corresponding audit_log_statement_policy Value
`0` or `"::none"`	`NONE`
`1` or `"::errors"`	`ERRORS`
`2` or `"::all"`	`ALL`

The `"::xxx"` values are symbolic pseudo-constants that may be given instead of the literal numeric values. They must be quoted as strings and are case-sensitive.

Referencing Predefined Functions

To refer to a predefined function in a `log` condition, use a `function` item, which takes `name` and `args` values to specify the function name and its arguments, respectively:

```
{
  "filter": {
    "class": {
      "name": "general",
      "event": {
        "name": "status",
        "log": {
          "function": {
            "name": "find_in_include_list",
            "args": [ { "string": [ { "field": "user.str" },
                                    { "string": "@"},
                                    { "field": "host.str" } ] } ]
          }
        }
      }
    }
  }
}
```

The function as specified in the `name` item should be the function name only, without parentheses or the argument list. Arguments in the `args` item, if there is one, must be given in the order listed in the function description. Arguments can refer to predefined variables, event fields, or string or numeric constants.

The preceding filter determines whether to log `general` class `status` events depending on whether the current user is found in the `audit_log_include_accounts` system variable. That user is constructed using fields in the event.

The following list describes the permitted predefined functions for `function` items:

- `audit_log_exclude_accounts_is_null()`

  Checks whether the `audit_log_exclude_accounts` system variable is `NULL`. This function can be helpful when defining filters that correspond to the legacy audit log implementation.

  Arguments:

  None.

- `audit_log_include_accounts_is_null()`

  Checks whether the `audit_log_include_accounts` system variable is `NULL`. This function can be helpful when defining filters that correspond to the legacy audit log implementation.

Arguments:

None.

- `debug_sleep(millisec)`

  Sleeps for the given number of milliseconds. This function is used during performance measurement.

  `debug_sleep()` is available for debug builds only.

  Arguments:

  - *`millisec`*: An unsigned integer that specifies the number of milliseconds to sleep.

- `find_in_exclude_list(account)`

  Checks whether an account string exists in the audit log exclude list (the value of the `audit_log_exclude_accounts` system variable).

  Arguments:

  - *`account`*: A string that specifies the user account name.

- `find_in_include_list(account)`

  Checks whether an account string exists in the audit log include list (the value of the `audit_log_include_accounts` system variable).

  Arguments:

  - *`account`*: A string that specifies the user account name.

- `string_find(text, substr)`

  Checks whether the `substr` value is contained in the `text` value. This search is case-sensitive.

  Arguments:

  - *`text`*: The text string to search.
  - *`substr`*: The substring to search for in *`text`*.

Replacing a User Filter

In some cases, the filter definition can be changed dynamically. To do this, define a `filter` configuration within an existing `filter`. For example:

```
{
  "filter": {
    "id": "main",
    "class": {
      "name": "table_access",
      "event": {
        "name": [ "update", "delete" ],
        "log": false,
        "filter": {
          "class": {
            "name": "general",
            "event" : { "name": "status",
```

```
                        "filter": { "ref": "main" } }
          },
          "activate": {
            "or": [
              { "field": { "name": "table_name.str", "value": "temp_1" } },
              { "field": { "name": "table_name.str", "value": "temp_2" } }
            ]
          }
        }
      }
    }
  }
}
```

A new filter is activated when the `activate` element within a subfilter evaluates to `true`. Using `activate` in a top-level `filter` is not permitted.

A new filter can be replaced with the original one by using a `ref` item inside the subfilter to refer to the original filter `id`.

The filter shown operates like this:

- The `main` filter waits for `table_access` events, either `update` or `delete`.
- If the `update` or `delete` `table_access` event occurs on the `temp_1` or `temp_2` table, the filter is replaced with the internal one (without an `id`, since there is no need to refer to it explicitly).
- If the end of the command is signalled (`general` / `status` event), an entry is written to the audit log file and the filter is replaced with the `main` filter.

The filter is useful to log statements that update or delete anything from the `temp_1` or `temp_2` tables, such as this one:

```
UPDATE temp_1, temp_3 SET temp_1.a=21, temp_3.a=23;
```

The statement generates multiple `table_access` events, but the audit log file will contain only `general` / `status` entries.

Note

Any `id` values used in the definition are evaluated with respect only to that definition. They have nothing to do with the value of the `audit_log_filter_id` system variable.

6.4.5.8 Legacy Mode Audit Log Filtering

Note

This section describes legacy audit log filtering, which applies if the `audit_log` plugin is installed but not the accompanying audit tables and UDFs needed for rule-based filtering.

The audit log plugin can filter audited events. This enables you to control whether audited events are written to the audit log file based on the account from which events originate or event status. Status filtering occurs separately for connection events and statement events.

- Event Filtering by Account
- Event Filtering by Status

Event Filtering by Account

To filter audited events based on the originating account, set one of these system variables at server startup or runtime:

- `audit_log_include_accounts`: The accounts to include in audit logging. If this variable is set, only these accounts are audited.
- `audit_log_exclude_accounts`: The accounts to exclude from audit logging. If this variable is set, all but these accounts are audited.

The value for either variable can be `NULL` or a string containing one or more comma-separated account names, each in *`user_name@host_name`* format. By default, both variables are `NULL`, in which case, no account filtering is done and auditing occurs for all accounts.

Modifications to `audit_log_include_accounts` or `audit_log_exclude_accounts` affect only connections created subsequent to the modification, not existing connections.

Example: To enable audit logging only for the `user1` and `user2` local host account accounts, set the `audit_log_include_accounts` system variable like this:

```
SET GLOBAL audit_log_include_accounts = 'user1@localhost,user2@localhost';
```

Only one of `audit_log_include_accounts` or `audit_log_exclude_accounts` can be non-`NULL` at a time:

- If you set `audit_log_include_accounts`, the server sets `audit_log_exclude_accounts` to `NULL`.
- If you attempt to set `audit_log_exclude_accounts`, an error occurs unless `audit_log_include_accounts` is `NULL`. In this case, you must first clear `audit_log_include_accounts` by setting it to `NULL`.

```
-- This sets audit_log_exclude_accounts to NULL
SET GLOBAL audit_log_include_accounts = value;

-- This fails because audit_log_include_accounts is not NULL
SET GLOBAL audit_log_exclude_accounts = value;

-- To set audit_log_exclude_accounts, first set
-- audit_log_include_accounts to NULL
SET GLOBAL audit_log_include_accounts = NULL;
SET GLOBAL audit_log_exclude_accounts = value;
```

If you inspect the value of either variable, be aware that `SHOW VARIABLES` displays `NULL` as an empty string. To avoid this, use `SELECT` instead:

```
mysql> SHOW VARIABLES LIKE 'audit_log_include_accounts';
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| audit_log_include_accounts |       |
+---------------------------+-------+
mysql> SELECT @@audit_log_include_accounts;
+-----------------------------+
| @@audit_log_include_accounts |
+-----------------------------+
| NULL                         |
```

```
+-----------------------------+
```

If a user name or host name requires quoting because it contains a comma, space, or other special character, quote it using single quotes. If the variable value itself is quoted with single quotes, double each inner single quote or escape it with a backslash. The following statements each enable audit logging for the local `root` account and are equivalent, even though the quoting styles differ:

```
SET GLOBAL audit_log_include_accounts = 'root@localhost';
SET GLOBAL audit_log_include_accounts = '''root''@''localhost''';
SET GLOBAL audit_log_include_accounts = '\'root\'@\'localhost\'';
SET GLOBAL audit_log_include_accounts = "'root'@'localhost'";
```

The last statement will not work if the `ANSI_QUOTES` SQL mode is enabled because in that mode double quotes signify identifier quoting, not string quoting.

Event Filtering by Status

To filter audited events based on status, set the following system variables at server startup or runtime. These variables apply only for legacy audit log filtering. For JSON audit log filtering, different status variables apply; see Audit Log Options and Variables.

- `audit_log_connection_policy`: Logging policy for connection events
- `audit_log_statement_policy`: Logging policy for statement events

Each variable takes a value of `ALL` (log all associated events; this is the default), `ERRORS` (log only failed events), or `NONE` (do not log events). For example, to log all statement events but only failed connection events, use these settings:

```
SET GLOBAL audit_log_statement_policy = ALL;
SET GLOBAL audit_log_connection_policy = ERRORS;
```

Another policy system variable, `audit_log_policy`, is available but does not afford as much control as `audit_log_connection_policy` and `audit_log_statement_policy`. It can be set only at server startup. At runtime, it is a read-only variable. It takes a value of `ALL` (log all events; this is the default), `LOGINS` (log connection events), `QUERIES` (log statement events), or `NONE` (do not log events). For any of those values, the audit log plugin logs all selected events without distinction as to success or failure. Use of `audit_log_policy` at startup works as follows:

- If you do not set `audit_log_policy` or set it to its default of `ALL`, any explicit settings for `audit_log_connection_policy` or `audit_log_statement_policy` apply as specified. If not specified, they default to `ALL`.
- If you set `audit_log_policy` to a non-`ALL` value, that value takes precedence over and is used to set `audit_log_connection_policy` and `audit_log_statement_policy`, as indicated in the following table. If you also set either of those variables to a value other than their default of `ALL`, the server writes a message to the error log to indicate that their values are being overridden.

Startup audit_log_policy Value	Resulting audit_log_connection_policy Value	Resulting audit_log_statement_policy Value
`LOGINS`	`ALL`	`NONE`
`QUERIES`	`NONE`	`ALL`
`NONE`	`NONE`	`NONE`

6.4.5.9 Audit Log Reference

The following discussion serves as a reference to MySQL Enterprise Audit components:

- Audit Log Tables
- Audit Log Functions
- Audit Log Option and Variable Reference
- Audit Log Options and Variables
- Audit Log Status Variables

To install the audit log tables and functions, use the instructions provided in Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”. Unless those components are installed, the `audit_log` plugin operates in legacy mode. See Section 6.4.5.8, “Legacy Mode Audit Log Filtering”.

Audit Log Tables

MySQL Enterprise Audit uses tables in the `mysql` system database for persistent storage of filter and user account data. The tables can be accessed only by users with privileges for that database. The tables use the `InnoDB` storage engine.

If these tables are missing, the `audit_log` plugin operates in legacy mode. See Section 6.4.5.8, “Legacy Mode Audit Log Filtering”.

The `audit_log_filter` table stores filter definitions. The table has these columns:

- `NAME`

  The filter name.

- `FILTER`

  The filter definition associated with the filter name. Definitions are stored as `JSON` values.

The `audit_log_user` table stores user account information. The table has these columns:

- `USER`

  The user name part of an account. For an account `user1@localhost`, the `USER` part is `user1`.

- `HOST`

  The host name part of an account. For an account `user1@localhost`, the `HOST` part is `localhost`.

- `FILTERNAME`

  The name of the filter assigned to the account. The filter name associates the account with a filter defined in the `audit_log_filter` table.

Audit Log Functions

This section describes, for each audit log user-defined function (UDF), its purpose, calling sequence, and return value. For information about the conditions under which these UDFs can be invoked, see Section 6.4.5.6, “Audit Log Filtering”.

Each audit log UDF returns a string that indicates whether the operation succeeded. `OK` indicates success. `ERROR: message` indicates failure.

These audit log UDFs are available:

- `audit_log_encryption_password_get([keyring_id])`

  This function fetches an audit log encryption password from the MySQL keyring, which must be enabled or an error occurs. Any keyring plugin can be used; for instructions, see Section 6.4.4, “The MySQL Keyring”.

  With no argument, the function retrieves the current encryption password as a binary string. An argument may be given to specify which audit log encryption password to retrieve. The argument must be the keyring ID of the current password or an archived password.

  For additional information about audit log encryption, see Audit Log File Encryption.

  Arguments:

  keyring_id: As of MySQL 8.0.17, this optional argument indicates the keyring ID of the password to retrieve. The maximum permitted length is 766 bytes. If omitted, the function retrieves the current password.

  Prior to MySQL 8.0.17, no argument is permitted. The function always retrieves the current password.

  Return value:

  The password string for success (up to 766 bytes), or `NULL` and an error for failure.

  Example:

  Retrieve the current password:

```
mysql> SELECT audit_log_encryption_password_get();
+-------------------------------------+
| audit_log_encryption_password_get() |
+-------------------------------------+
| secret                              |
+-------------------------------------+
```

  To retrieve a password by ID, you can determine which audit log keyring IDs exist by querying the Performance Schema `keyring_keys` table:

```
mysql> SELECT * FROM performance_schema.keyring_keys
       WHERE KEY_ID LIKE 'audit_log%'
       ORDER BY KEY_ID;
+----------------------------+-----------+----------------+
| KEY_ID                     | KEY_OWNER | BACKEND_KEY_ID |
+----------------------------+-----------+----------------+
| audit_log-20190415T152248-1 |           |                |
| audit_log-20190415T153507-1 |           |                |
| audit_log-20190416T125122-1 |           |                |
| audit_log-20190416T141608-1 |           |                |
+----------------------------+-----------+----------------+
mysql> SELECT audit_log_encryption_password_get('audit_log-20190416T125122-1');
+-----------------------------------------------------------------+
| audit_log_encryption_password_get('audit_log-20190416T125122-1') |
+-----------------------------------------------------------------+
| segreto                                                         |
```

```
+------------------------------------------------------------------+
```

- `audit_log_encryption_password_set(`*`password`*`)`

  Sets the current audit log encryption password to the argument and stores the password in the MySQL keyring. If encryption is enabled, the function performs a log file rotation operation that renames the current log file, and begins a new log file encrypted with the password. The keyring must be enabled or an error occurs. Any keyring plugin can be used; for instructions, see Section 6.4.4, "The MySQL Keyring".

  For additional information about audit log encryption, see Audit Log File Encryption.

  Arguments:

  `password`: The password string. The maximum permitted length is 766 bytes.

  Return value:

  1 for success, 0 for failure.

  Example:

```
mysql> SELECT audit_log_encryption_password_set(password);
+-------------------------------------------+
| audit_log_encryption_password_set(password) |
+-------------------------------------------+
| 1                                         |
+-------------------------------------------+
```

- `audit_log_filter_flush()`

  Calling any of the other filtering UDFs affects operational audit log filtering immediately and updates the audit log tables. If instead you modify the contents of those tables directly using statements such as `INSERT`, `UPDATE`, and `DELETE`, the changes do not affect filtering immediately. To flush your changes and make them operational, call `audit_log_filter_flush()`.

  Warning

  `audit_log_filter_flush()` should be used only after modifying the audit tables directly, to force reloading all filters. Otherwise, this function should be avoided. It is, in effect, a simplified version of unloading and reloading the `audit_log` plugin with `UNINSTALL PLUGIN` plus `INSTALL PLUGIN`.

  `audit_log_filter_flush()` affects all current sessions and detaches them from their previous filters. Current sessions are no longer logged unless they disconnect and reconnect, or execute a change-user operation.

  If this function fails, an error message is returned and the audit log is disabled until the next successful call to `audit_log_filter_flush()`.

  Arguments:

  None.

  Return value:

  A string that indicates whether the operation succeeded. `OK` indicates success. `ERROR: `*`message`* indicates failure.

Example:

```
mysql> SELECT audit_log_filter_flush();
+--------------------------+
| audit_log_filter_flush() |
+--------------------------+
| OK                       |
+--------------------------+
```

- `audit_log_filter_remove_filter(`*`filter_name`*`)`

  Given a filter name, removes the filter from the current set of filters. It is not an error for the filter not to exist.

  If a removed filter is assigned to any user accounts, those users stop being filtered (they are removed from the `audit_log_user` table). Termination of filtering includes any current sessions for those users: They are detached from the filter and no longer logged.

  Arguments:

  - *`filter_name`*: A string that specifies the filter name.

  Return value:

  A string that indicates whether the operation succeeded. `OK` indicates success. `ERROR: `*`message`* indicates failure.

  Example:

```
mysql> SELECT audit_log_filter_remove_filter('SomeFilter');
+-----------------------------------------------+
| audit_log_filter_remove_filter('SomeFilter') |
+-----------------------------------------------+
| OK                                            |
+-----------------------------------------------+
```

- `audit_log_filter_remove_user(`*`user_name`*`)`

  Given a user account name, cause the user to be no longer assigned to a filter. It is not an error if the user has no filter assigned. Filtering of current sessions for the user remains unaffected. New connections for the user are filtered using the default account filter if there is one, and are not logged otherwise.

  If the name is `%`, the function removes the default account filter that is used for any user account that has no explicitly assigned filter.

  Arguments:

  - *`user_name`*: The user account name as a string in *`user_name@host_name`* format, or `%` to represent the default account.

  Return value:

  A string that indicates whether the operation succeeded. `OK` indicates success. `ERROR: `*`message`* indicates failure.

  Example:

```
mysql> SELECT audit_log_filter_remove_user('user1@localhost');
+-----------------------------------------------+
| audit_log_filter_remove_user('user1@localhost') |
+-----------------------------------------------+
| OK                                            |
+-----------------------------------------------+
```

- `audit_log_filter_set_filter(`*`filter_name`*`, `*`definition`*`)`

  Given a filter name and definition, adds the filter to the current set of filters. If the filter already exists and is used by any current sessions, those sessions are detached from the filter and are no longer logged. This occurs because the new filter definition has a new filter ID that differs from its previous ID.

  Arguments:

  - *`filter_name`*: A string that specifies the filter name.
  - *`definition`*: A `JSON` value that specifies the filter definition.

  Return value:

  A string that indicates whether the operation succeeded. `OK` indicates success. `ERROR: `*`message`* indicates failure.

  Example:

```
mysql> SET @f = '{ "filter": { "log": false } }';
mysql> SELECT audit_log_filter_set_filter('SomeFilter', @f);
+-----------------------------------------------+
| audit_log_filter_set_filter('SomeFilter', @f) |
+-----------------------------------------------+
| OK                                            |
+-----------------------------------------------+
```

- `audit_log_filter_set_user(`*`user_name`*`, `*`filter_name`*`)`

  Given a user account name and a filter name, assigns the filter to the user. A user can be assigned only one filter, so if the user was already assigned a filter, the assignment is replaced. Filtering of current sessions for the user remains unaffected. New connections are filtered using the new filter.

  As a special case, the name `%` represents the default account. The filter is used for connections from any user account that has no explicitly assigned filter.

  Arguments:

  - *`user_name`*: The user account name as a string in *`user_name@host_name`* format, or `%` to represent the default account.
  - *`filter_name`*: A string that specifies the filter name.

  Return value:

  A string that indicates whether the operation succeeded. `OK` indicates success. `ERROR: `*`message`* indicates failure.

  Example:

```
mysql> SELECT audit_log_filter_set_user('user1@localhost', 'SomeFilter');
+-----------------------------------------------------------+
| audit_log_filter_set_user('user1@localhost', 'SomeFilter') |
+-----------------------------------------------------------+
| OK                                                        |
+-----------------------------------------------------------+
```

- `audit_log_read([arg])`

  Reads events from the audit log and returns a binary `JSON` string containing an array of audit events. If the audit log format is not `JSON`, an error occurs.

  Each event in the return value is a `JSON` hash, except that the last array element may be a `JSON null` value to indicate no following events are available to read.

  For the first call to `audit_log_read()` within a session, pass a bookmark indicating where to begin reading. If the final value of the returned array is not a `JSON null` value, there are more events following those just read and `audit_log_read()` can be called without or with a bookmark argument. Without an argument, reading continues with the next unread event. With a bookmark argument, reading continues from the bookmark.

  If the final value of the returned array is a `JSON null` value, there are no more events left to be read and the next call to `audit_log_read()` must include a bookmark argument.

  To obtain a bookmark for the most recently written event, call `audit_log_read_bookmark()`.

  For additional information about audit log-reading functions, see Audit Log File Reading.

  Arguments:

  `arg`: An optional bookmark, represented as a string containing a `JSON` hash that indicates where and how much to read. The following items are significant in the `arg` value (other items are ignored):

  - `timestamp`, `id`: The location within the audit log of the first event to read. Both items must be present to completely specify a position.
  - `max_array_length`: The maximum number of events to read from the log. If omitted, the default is to read to the end of the log or until the read buffer is full, whichever comes first.

  Return value:

  A binary `JSON` string containing an array of audit events for success, or `NULL` and an error for failure.

  Example:

```
mysql> SELECT audit_log_read(audit_log_read_bookmark());
+------------------------------------------------------------------------+
| audit_log_read(audit_log_read_bookmark())                              |
+------------------------------------------------------------------------+
| [ {"timestamp":"2018-01-15 22:41:24","id":0,"class":"connection", ... |
+------------------------------------------------------------------------+
```

- `audit_log_read_bookmark()`

  Returns a binary `JSON` string representing a bookmark for the most recently written audit log event. If the audit log format is not `JSON`, an error occurs.

The bookmark is a `JSON` hash with `timestamp` and `id` items indicating the event position within the audit log. It is suitable for passing to `audit_log_read()` to indicate to that function where to begin reading.

For additional information about audit log-reading functions, see Audit Log File Reading.

Arguments:

None.

Return value:

A binary `JSON` string containing a bookmark for success, or `NULL` and an error for failure.

Example:

```
mysql> SELECT audit_log_read_bookmark();
+------------------------------------------------+
| audit_log_read_bookmark()                      |
+------------------------------------------------+
| { "timestamp": "2018-01-15 21:03:44", "id": 0 } |
+------------------------------------------------+
```

Audit Log Option and Variable Reference

Table 6.36 Audit Log Option and Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
audit-log	Yes	Yes				
audit_log_buffer_size	Yes	Yes	Yes		Global	No
audit_log_compression	Yes	Yes	Yes		Global	No
audit_log_connection_policy	Yes	Yes	Yes		Global	Yes
audit_log_current_session			Yes		Both	No
Audit_log_current_size				Yes	Global	No
audit_log_encryption	Yes	Yes	Yes		Global	No
Audit_log_event_max_drop_size				Yes	Global	No
Audit_log_events				Yes	Global	No
Audit_log_events_filtered				Yes	Global	No
Audit_log_events_lost				Yes	Global	No
Audit_log_events_written				Yes	Global	No
audit_log_exclude_accounts	Yes	Yes	Yes		Global	Yes
audit_log_file	Yes	Yes	Yes		Global	No
audit_log_filter_id			Yes		Both	No
audit_log_flush			Yes		Global	Yes
audit_log_format	Yes	Yes	Yes		Global	No
audit_log_include_accounts	Yes	Yes	Yes		Global	Yes
audit_log_password_history_keep_days	Yes	Yes	Yes		Global	Yes
audit_log_policy	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
audit_log_read_buffer_size	Yes	Yes	Yes		Varies	Va
audit_log_rotate_on_size	Yes	Yes	Yes		Global	Ye
audit_log_statement_policy	Yes	Yes	Yes		Global	Ye
audit_log_strategy	Yes	Yes	Yes		Global	No
Audit_log_total_size				Yes	Global	No
Audit_log_write_waits				Yes	Global	No

Audit Log Options and Variables

This section describes the command options and system variables that configure operation of MySQL Enterprise Audit. If values specified at startup time are incorrect, the `audit_log` plugin may fail to initialize properly and the server does not load it. In this case, the server may also produce error messages for other audit log settings because it will not recognize them.

To configure activation of the audit log plugin, use this option:

- `--audit-log[=value]`

Property	Value
Command-Line Format	`--audit-log[=value]`
Introduced	8.0.11
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`FORCE`
	`FORCE_PLUS_PERMANENT`

This option controls how the server loads the `audit_log` plugin at startup. It is available only if the plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load` or `--plugin-load-add`. See Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”.

The option value should be one of those available for plugin-loading options, as described in Section 5.6.1, “Installing and Uninstalling Plugins”. For example, `--audit-log=FORCE_PLUS_PERMANENT` tells the server to load the plugin and prevent it from being removed while the server is running.

If the audit log plugin is enabled, it exposes several system variables that permit control over logging:

```
mysql> SHOW VARIABLES LIKE 'audit_log%';
+-------------------------------+--------------+
| Variable_name                 | Value        |
+-------------------------------+--------------+
| audit_log_buffer_size         | 1048576      |
| audit_log_connection_policy   | ALL          |
| audit_log_current_session     | OFF          |
| audit_log_exclude_accounts    |              |
| audit_log_file                | audit.log    |
| audit_log_filter_id           | 0            |
```

```
| audit_log_flush              | OFF          |
| audit_log_format             | NEW          |
| audit_log_include_accounts   |              |
| audit_log_policy             | ALL          |
| audit_log_rotate_on_size     | 0            |
| audit_log_statement_policy   | ALL          |
| audit_log_strategy           | ASYNCHRONOUS |
+------------------------------+--------------+
```

You can set any of these variables at server startup, and some of them at runtime. Those that are available only for legacy mode audit log filtering are so noted.

- `audit_log_buffer_size`

Property	Value
Command-Line Format	`--audit-log-buffer-size=#`
Introduced	8.0.11
System Variable	`audit_log_buffer_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1048576`
Minimum Value	`4096`
Maximum Value (64-bit platforms)	`18446744073709547520`
Maximum Value (32-bit platforms)	`4294967295`

When the audit log plugin writes events to the log asynchronously, it uses a buffer to store event contents prior to writing them. This variable controls the size of that buffer, in bytes. The server adjusts the value to a multiple of 4096. The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin allocates this buffer only if logging is asynchronous.

- `audit_log_compression`

Property	Value
Command-Line Format	`--audit-log-compression=value`
Introduced	8.0.11
System Variable	`audit_log_compression`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`NONE`
Valid Values	`NONE` `GZIP`

The type of compression for the audit log file. Permitted values are `NONE` (no compression; the default) and `GZIP` (GNU Zip compression). For more information, see Audit Log File Compression.

- `audit_log_connection_policy`

Property	Value
Command-Line Format	`--audit-log-connection-policy=value`
Introduced	8.0.11
System Variable	`audit_log_connection_policy`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`ALL`

Valid Values	`ALL`
	`ERRORS`
	`NONE`

Note

This variable applies only to legacy mode audit log filtering (see Section 6.4.5.8, “Legacy Mode Audit Log Filtering”).

The policy controlling how the audit log plugin writes connection events to its log file. The following table shows the permitted values.

Value	Description
`ALL`	Log all connection events
`ERRORS`	Log only failed connection events
`NONE`	Do not log connection events

Note

At server startup, any explicit value given for `audit_log_connection_policy` may be overridden if `audit_log_policy` is also specified, as described in Section 6.4.5.5, “Audit Log Logging Configuration”.

- `audit_log_current_session`

Property	Value
Introduced	8.0.11
System Variable	`audit_log_current_session`
Scope	Global, Session
Dynamic	No
SET_VAR Hint Applies	No

Property	Value
Type	Boolean
Default Value	`depends on filtering policy`

Whether audit logging is enabled for the current session. The session value of this variable is read only. It is set when the session begins based on the values of the `audit_log_include_accounts` and `audit_log_exclude_accounts` system variables. The audit log plugin uses the session value to determine whether to audit events for the session. (There is a global value, but the plugin does not use it.)

- `audit_log_encryption`

Property	Value
Command-Line Format	`--audit-log-encryption=value`
Introduced	8.0.11
System Variable	`audit_log_encryption`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`NONE`
Valid Values	`NONE` `AES`

The type of encryption for the audit log file. Permitted values are `NONE` (no encryption; the default) and `AES` (AES-256-CBC cipher encryption). For more information, see Audit Log File Encryption.

- `audit_log_exclude_accounts`

Property	Value
Command-Line Format	`--audit-log-exclude-accounts=value`
Introduced	8.0.11
System Variable	`audit_log_exclude_accounts`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`NULL`

Note

This variable applies only to legacy mode audit log filtering (see Section 6.4.5.8, “Legacy Mode Audit Log Filtering”).

The accounts for which events should not be logged. The value should be `NULL` or a string containing a list of one or more comma-separated account names. For more information, see Section 6.4.5.6, “Audit Log Filtering”.

Modifications to `audit_log_exclude_accounts` affect only connections created subsequent to the modification, not existing connections.

- `audit_log_file`

Property	Value
Command-Line Format	`--audit-log-file=file_name`
Introduced	8.0.11
System Variable	`audit_log_file`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	`audit.log`

The base name and suffix of the file to which the audit log plugin writes events. The default value is `audit.log`, regardless of logging format. To have the name suffix correspond to the format, set the name explicitly, choosing a different suffix (for example, `audit.xml` for XML format, `audit.json` for JSON format).

If the value of `audit_log_file` is a relative path name, the plugin interprets it relative to the data directory. If the value is a full path name, the plugin uses the value as is. A full path name may be useful if it is desirable to locate audit files on a separate file system or directory. For security reasons, write the audit log file to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log.

For details about how the audit log plugin interprets the `audit_log_file` value and the rules for file renaming that occurs at plugin initialization and termination, see Audit Log File Name.

The audit log plugin uses the directory containing the audit log file (determined from the `audit_log_file` value) as the location to search for readable audit log files. From these log files and the current file, the plugin constructs a list of the ones that are subject to use with the audit log bookmarking and reading functions. See Audit Log File Reading.

- `audit_log_filter_id`

Property	Value
Introduced	8.0.11
System Variable	`audit_log_filter_id`
Scope	Global, Session
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer

The session value of this variable indicates the internally maintained ID of the audit filter for the current session. A value of 0 means that the session has no filter assigned.

- `audit_log_flush`

Property	Value
Introduced	8.0.11
System Variable	`audit_log_flush`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

When this variable is set to enabled (1 or `ON`), the audit log plugin closes and reopens its log file to flush it. (The value remains `OFF` so that you need not disable it explicitly before enabling it again to perform another flush.) Enabling this variable has no effect unless `audit_log_rotate_on_size` is 0. For more information, see Section 6.4.5.5, “Audit Log Logging Configuration”.

- `audit_log_format`

Property	Value
Command-Line Format	`--audit-log-format=value`
Introduced	8.0.11
System Variable	`audit_log_format`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`NEW`

Valid Values	`OLD`
	`NEW`
	`JSON`

The audit log file format. Permitted values are `OLD` (old-style XML), `NEW` (new-style XML; the default), and `JSON`. For details about each format, see Section 6.4.5.4, “Audit Log File Formats”.

Note

For information about issues to consider when changing the log format, see Audit Log File Format.

- `audit_log_include_accounts`

Property	Value
Command-Line Format	`--audit-log-include-accounts=value`

Property	Value
Introduced	8.0.11
System Variable	`audit_log_include_accounts`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`NULL`

Note

This variable applies only to legacy mode audit log filtering (see Section 6.4.5.8, “Legacy Mode Audit Log Filtering”).

The accounts for which events should be logged. The value should be `NULL` or a string containing a list of one or more comma-separated account names. For more information, see Section 6.4.5.6, “Audit Log Filtering”.

Modifications to `audit_log_include_accounts` affect only connections created subsequent to the modification, not existing connections.

- `audit_log_password_history_keep_days`

Property	Value
Command-Line Format	`--audit-log-password-history-keep-days=#`
Introduced	8.0.17
System Variable	`audit_log_password_history_keep_days`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

When a new encryption password is created, the audit log plugin archives the previous password, if one exists, for later use. The `audit_log_password_history_keep_days` variable controls automatic removal of archived passwords. Its value indicates the number of days after which archived audit log encryption passwords are removed. The default is 0, which disables password removal; the password retention period is forever.

New audit log encryption passwords are created under these circumstances:

- During plugin initialization, if the plugin finds that log file encryption is enabled, it checks whether the keyring contains an audit log encryption password. If not, the plugin automatically generates a random initial encryption password.
- When the `audit_log_encryption_password_set()` function is called to set a specific password.

In each case, the plugin stores the password in the key ring and uses it to encrypt new log files.

Cleanup of old audit log encryption passwords occurs under these circumstances:

- During plugin initialization.
- When the `audit_log_encryption_password_set()` function is called.
- When the runtime value of `audit_log_password_history_keep_days` is changed from its current value to a value greater than 0. Runtime value changes may occur for `SET` statements that use the `GLOBAL` or `PERSIST` keyword, but not the `PERSIST_ONLY` keyword. `PERSIST_ONLY` writes the variable setting to `mysqld-auto.cnf`, but has no effect on the runtime value.

When password cleanup occurs, the current value of `audit_log_password_history_keep_days` determines which passwords to remove:

- If the value is *N* > 0, the plugin removes passwords more than *N* days old.
- If the value is 0, the plugin removes no passwords.

Note

Take care not to expire old passwords that are still needed to read archived encrypted log files.

If you normally leave password expiration disabled (`audit_log_password_history_keep_days=0`), it is possible to perform an on-demand cleanup operation by temporarily assigning the variable a value greater than zero. For example, to expire passwords older than 365 days, do this:

```
SET GLOBAL audit_log_password_history_keep_days = 365;
SET GLOBAL audit_log_password_history_keep_days = 0;
```

Setting the runtime value of `audit_log_password_history_keep_days` requires the `AUDIT_ADMIN` privilege, in addition to the `SYSTEM_VARIABLES_ADMIN` or `SUPER` privilege normally required to set a global system variable runtime value.

- `audit_log_policy`

Property	Value
Command-Line Format	`--audit-log-policy=value`
Introduced	8.0.11
System Variable	`audit_log_policy`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`ALL`

Valid Values	`ALL`
	`LOGINS`
	`QUERIES`

Property	Value
	NONE

Note

This variable applies only to legacy mode audit log filtering (see Section 6.4.5.8, “Legacy Mode Audit Log Filtering”).

The policy controlling how the audit log plugin writes events to its log file. The following table shows the permitted values.

Value	Description
ALL	Log all events
LOGINS	Log only login events
QUERIES	Log only query events
NONE	Log nothing (disable the audit stream)

audit_log_policy can be set only at server startup. At runtime, it is a read-only variable. Two other system variables, audit_log_connection_policy and audit_log_statement_policy, provide finer control over logging policy and can be set either at startup or at runtime. If you use audit_log_policy at startup instead of the other two variables, the server uses its value to set those variables. For more information about the policy variables and their interaction, see Section 6.4.5.5, “Audit Log Logging Configuration”.

- audit_log_read_buffer_size

Property	Value
Command-Line Format	--audit-log-read-buffer-size=#
Introduced	8.0.11
System Variable (>= 8.0.11)	audit_log_read_buffer_size
Scope (>= 8.0.12)	Global, Session
Scope (8.0.11)	Global
Dynamic (>= 8.0.12)	Yes
Dynamic (8.0.11)	No
SET_VAR Hint Applies (>= 8.0.11)	No
Type	Integer
Default Value (>= 8.0.12)	32768
Default Value (8.0.11)	1048576
Minimum Value (>= 8.0.12)	32768
Minimum Value (8.0.11)	1024
Maximum Value	4194304

The buffer size for reading from the audit log file, in bytes. The audit_log_read() function reads no more than this many bytes. Log file reading is supported only for JSON log format. For more information, see Audit Log File Reading.

As of MySQL 8.0.12, this variable has a default of 32KB and can be set at runtime. Each client should set its session value of `audit_log_read_buffer_size` appropriately for its use of `audit_log_read()`. Prior to MySQL 8.0.12, `audit_log_read_buffer_size` has a default of 1MB, affects all clients, and can be changed only at server startup.

- `audit_log_rotate_on_size`

Property	Value
Command-Line Format	`--audit-log-rotate-on-size=#`
Introduced	8.0.11
System Variable	`audit_log_rotate_on_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`

If the `audit_log_rotate_on_size` value is 0, the audit log plugin does not perform automatic log file rotation. Instead, use `audit_log_flush` to close and reopen the log on demand. In this case, manually rename the file externally to the server before flushing it.

If the `audit_log_rotate_on_size` value is greater than 0, automatic size-based log file rotation occurs. Whenever a write to the log file causes its size to exceed the `audit_log_rotate_on_size` value, the audit log plugin closes the current log file, renames it, and opens a new log file.

For more information about audit log file rotation, see Audit Log File Space Management and Name Rotation.

If you set this variable to a value that is not a multiple of 4096, it is truncated to the nearest multiple. (Thus, setting it to a value less than 4096 has the effect of setting it to 0 and no rotation occurs, except manually.)

- `audit_log_statement_policy`

Property	Value
Command-Line Format	`--audit-log-statement-policy=value`
Introduced	8.0.11
System Variable	`audit_log_statement_policy`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`ALL`

Valid Values	`ALL`
	`ERRORS`
	`NONE`

Note

This variable applies only to legacy mode audit log filtering (see Section 6.4.5.8, “Legacy Mode Audit Log Filtering”).

The policy controlling how the audit log plugin writes statement events to its log file. The following table shows the permitted values.

Value	Description
`ALL`	Log all statement events
`ERRORS`	Log only failed statement events
`NONE`	Do not log statement events

Note

At server startup, any explicit value given for `audit_log_statement_policy` may be overridden if `audit_log_policy` is also specified, as described in Section 6.4.5.5, “Audit Log Logging Configuration”.

- `audit_log_strategy`

Property	Value
Command-Line Format	`--audit-log-strategy=value`
Introduced	8.0.11
System Variable	`audit_log_strategy`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`ASYNCHRONOUS`

Valid Values	`ASYNCHRONOUS`
	`PERFORMANCE`
	`SEMISYNCHRONOUS`
	`SYNCHRONOUS`

The logging method used by the audit log plugin. These strategy values are permitted:

- `ASYNCHRONOUS`: Log asynchronously. Wait for space in the output buffer.
- `PERFORMANCE`: Log asynchronously. Drop requests for which there is insufficient space in the output buffer.
- `SEMISYNCHRONOUS`: Log synchronously. Permit caching by the operating system.
- `SYNCHRONOUS`: Log synchronously. Call `sync()` after each request.

Audit Log Status Variables

If the audit log plugin is enabled, it exposes several status variables that provide operational information. These variables are available for legacy mode audit filtering and JSON mode audit filtering.

- `Audit_log_current_size`

  The size of the current audit log file. The value increases when an event is written to the log and is reset to 0 when the log is rotated.

- `Audit_log_event_max_drop_size`

  The size of the largest dropped event in performance logging mode. For a description of logging modes, see Section 6.4.5.5, “Audit Log Logging Configuration”.

- `Audit_log_events`

  The number of events handled by the audit log plugin, whether or not they were written to the log based on filtering policy (see Section 6.4.5.5, “Audit Log Logging Configuration”).

- `Audit_log_events_filtered`

  The number of events handled by the audit log plugin that were filtered (not written to the log) based on filtering policy (see Section 6.4.5.5, “Audit Log Logging Configuration”).

- `Audit_log_events_lost`

  The number of events lost in performance logging mode because an event was larger than than the available audit log buffer space. This value may be useful for assessing how to set `audit_log_buffer_size` to size the buffer for performance mode. For a description of logging modes, see Section 6.4.5.5, “Audit Log Logging Configuration”.

- `Audit_log_events_written`

  The number of events written to the audit log.

- `Audit_log_total_size`

  The total size of events written to all audit log files. Unlike `Audit_log_current_size`, the value of `Audit_log_total_size` increases even when the log is rotated.

- `Audit_log_write_waits`

  The number of times an event had to wait for space in the audit log buffer in asynchronous logging mode. For a description of logging modes, see Section 6.4.5.5, “Audit Log Logging Configuration”.

6.4.5.10 Audit Log Restrictions

MySQL Enterprise Audit is subject to these general restrictions:

- Only SQL statements are logged. Changes made by no-SQL APIs, such as memcached, Node.JS, and the NDB API, are not logged.
- Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures.
- Contents of files referenced by statements such as `LOAD DATA` are not logged.

NDB Cluster. It is possible to use MySQL Enterprise Audit with MySQL NDB Cluster, subject to the following conditions:

- All changes to be logged must be done using the SQL interface. Changes using no-SQL interfaces, such as those provided by the NDB API, memcached, or ClusterJ, are not logged.
- The plugin must be installed on each MySQL server that is used to execute SQL on the cluster.
- Audit plugin data must be aggregated amongst all MySQL servers used with the cluster. This aggregation is the responsibility of the application or user.

6.4.6 The Audit Message Component

As of MySQL 8.0.14, the `audit_api_message_emit` component enables applications to add their own message events to the audit log, using the `audit_api_message_emit_udf()` user-defined function.

The `audit_api_message_emit` component cooperates with all plugins of audit type. For concreteness, examples use the `audit_log` plugin described in Section 6.4.5, “MySQL Enterprise Audit”.

- Installing or Uninstalling the Audit Message Component
- Audit Message Function

Installing or Uninstalling the Audit Message Component

To be usable by the server, the component library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To install the `audit_api_message_emit` component, use this statement:

```
INSTALL COMPONENT "file://component_audit_api_message_emit";
```

Component installation is a one-time operation that need not be done per server startup. `INSTALL COMPONENT` loads the component, and also registers it in the `mysql.component` system table to cause it to be loaded during subsequent server startups.

To uninstall the `audit_api_message_emit` component, use this statement:

```
UNINSTALL COMPONENT "file://component_audit_api_message_emit";
```

`UNINSTALL COMPONENT` unloads the component, and deregisters it from the `mysql.component` system table to cause it not to be loaded during subsequent server startups.

Installing or uninstalling the `audit_api_message_emit` component installs or uninstalls the `audit_api_message_emit_udf()` function that the component implements. It is not necessary to use `CREATE FUNCTION` or `DROP FUNCTION` to do so.

Audit Message Function

This section describes the `audit_api_message_emit_udf()` user-defined function (UDF) implemented by the `audit_api_message_emit` component.

Before using the audit message function, install the audit message component according to the instructions provided at Installing or Uninstalling the Audit Message Component.

- `audit_api_message_emit_udf(`*`component, producer, message`*`[, key, value] ...)`

  Adds a message event to the audit log. Message events include component, producer, and message strings of the caller's choosing, and optionally a set of key-value pairs.

An event posted by this UDF is sent to all enabled plugins of audit type, each of which handles the event according to its own rules. If no plugin of audit type is enabled, posting the event has no effect.

Arguments:

- *`component`*: A string that specifies a component name.
- *`producer`*: A string that specifies a producer name.
- *`message`*: A string that specifies the event message.
- *`key`*, *`value`*: Events may include 0 or more key-value pairs that specify an arbitrary application-provided data map. Each *`key`* argument is a string that specifies a name for its immediately following *`value`* argument. Each *`value`* argument specifies a value for its immediately following *`key`* argument. Each *`value`* can be a string or numeric value, or `NULL`.

Return value:

The string `OK` to indicate success. An error occurs if the function fails.

Example:

```
mysql> SELECT audit_api_message_emit_udf('component_text',
                                         'producer_text',
                                         'message_text',
                                         'key1', 'value1',
                                         'key2', 123,
                                         'key3', NULL) AS 'Message';
+---------+
| Message |
+---------+
| OK      |
+---------+
```

Additional information:

Each audit plugin that receives an event posted by `audit_api_message_emit_udf()` logs the event in plugin-specific format. For example, the `audit_log` plugin (see Section 6.4.5, “MySQL Enterprise Audit”) logs message values as follows, depending on the log format configured by the `audit_log_format` system variable:

- JSON format (`audit_log_format=JSON`):

```
{
  ...
  "class": "message",
  "event": "user",
  ...
  "message_data": {
    "component": "component_text",
    "producer": "producer_text",
    "message": "message_text",
    "map": {
      "key1": "value1",
      "key2": 123,
      "key3": null
    }
  }
}
```

- New-style XML format (`audit_log_format=NEW`):

```
<AUDIT_RECORD>
 ...
 <NAME>Message</NAME>
 ...
 <COMMAND_CLASS>user</COMMAND_CLASS>
 <COMPONENT>component_text</COMPONENT>
 <PRODUCER>producer_text</PRODUCER>
 <MESSAGE>message_text</MESSAGE>
 <MAP>
   <ELEMENT>
     <KEY>key1</KEY>
     <VALUE>value1</VALUE>
   </ELEMENT>
   <ELEMENT>
     <KEY>key2</KEY>
     <VALUE>123</VALUE>
   </ELEMENT>
   <ELEMENT>
     <KEY>key3</KEY>
     <VALUE/>
   </ELEMENT>
 </MAP>
</AUDIT_RECORD>
```

- Old-style XML format (`audit_log_format=OLD`):

```
<AUDIT_RECORD
  ...
  NAME="Message"
  ...
  COMMAND_CLASS="user"
  COMPONENT="component_text"
  PRODUCER="producer_text"
  MESSAGE="message_text"/>
```

Note

Message events logged in old-style XML format do not include the key-value map due to representational constraints imposed by this format.

Messages posted by `audit_api_message_emit_udf()` have an event class of `MYSQL_AUDIT_MESSAGE_CLASS` and a subclass of `MYSQL_AUDIT_MESSAGE_USER`. (Interally generated audit messages have the same class and a subclass of `MYSQL_AUDIT_MESSAGE_INTERNAL`; this subclass currently is unused.) To refer to such events in `audit_log` filtering rules, use a `class` element with a `name` value of `message`. For example:

```
{
  "filter": {
    "class": {
      "name": "message"
    }
  }
}
```

Should it be necessary to distinguish user-generated and internally generated message events, test the `subclass` value against `user` or `internal`.

Filtering based on the contents of the key-value map is not supported.

For information about writing filtering rules, see Section 6.4.5.6, “Audit Log Filtering”.

6.4.7 MySQL Enterprise Firewall

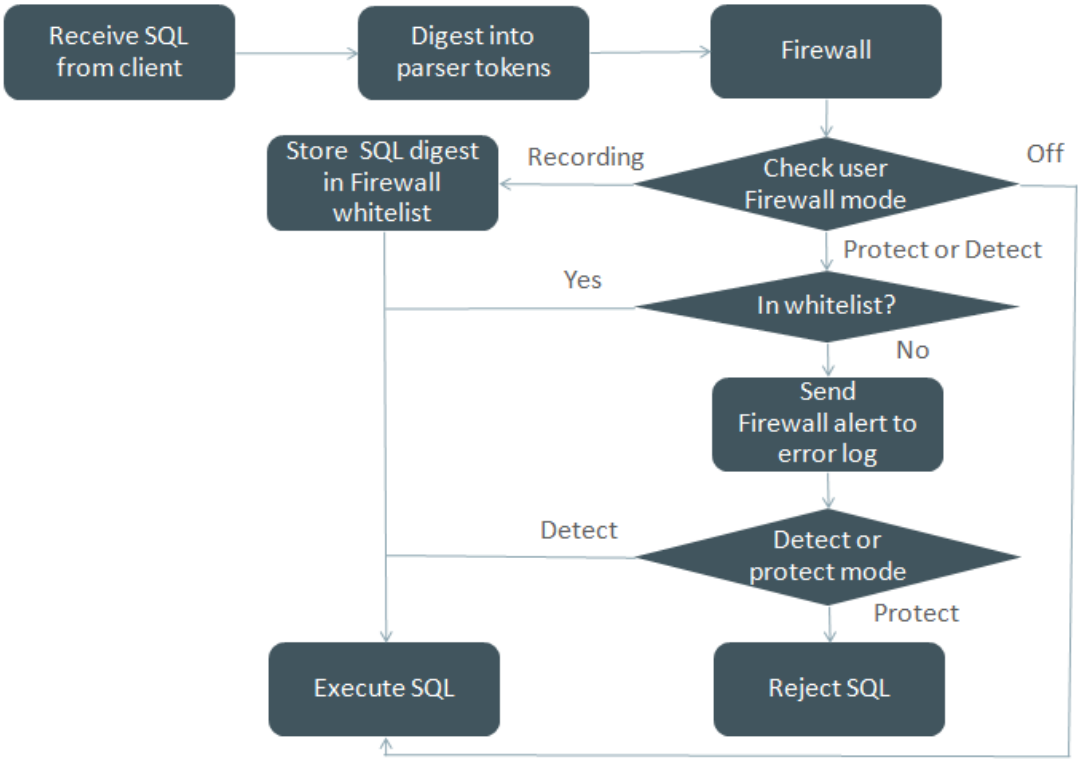
Note

MySQL Enterprise Firewall is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition includes MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against whitelists of accepted statement patterns. This helps harden MySQL Server against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics.

Each MySQL account registered with the firewall has its own statement whitelist, enabling protection to be tailored per account. For a given account, the firewall can operate in recording, protecting, or detecting mode, for training in the accepted statement patterns, active protection against unacceptable statements, or passive detection of unacceptable statements. The diagram illustrates how the firewall processes incoming statements in each mode.

Figure 6.1 MySQL Enterprise Firewall Operation



The following sections describe the components of MySQL Enterprise Firewall, discuss how to install and use it, and provide reference information for its components.

6.4.7.1 MySQL Enterprise Firewall Components

MySQL Enterprise Firewall is based on a plugin library that implements these components:

- A server-side plugin named `MYSQL_FIREWALL` examines SQL statements before they execute and, based on its in-memory cache, renders a decision whether to execute or reject each statement.
- Server-side plugins named `MYSQL_FIREWALL_USERS` and `MYSQL_FIREWALL_WHITELIST` implement `INFORMATION_SCHEMA` tables that provide views into the firewall data cache.
- System tables named `firewall_users` and `firewall_whitelist` in the `mysql` database provide persistent storage of firewall data.
- Stored procedures named `sp_set_firewall_mode()` and `sp_reload_firewall_rules()` perform tasks such as registering MySQL accounts with the firewall, establishing their operational mode, and managing transfer of firewall data between the cache and the underlying system tables.
- A set of user-defined functions provides an SQL-level API for lower-level tasks such as synchronizing the cache with the underlying system tables.
- System variables enable firewall configuration and status variables provide runtime operational information.
- `FIREWALL_ADMIN` and `FIREWALL_USER` privileges enable users to administer firewall rules for any user, and their own firewall rules, respectively.

6.4.7.2 Installing or Uninstalling MySQL Enterprise Firewall

MySQL Enterprise Firewall installation is a one-time operation that installs the components described in Section 6.4.7.1, “MySQL Enterprise Firewall Components”. Installation can be performed using a graphical interface or manually:

- On Windows, MySQL Installer includes an option to enable MySQL Enterprise Firewall for you.
- MySQL Workbench 6.3.4 or higher can install MySQL Enterprise Firewall, enable or disable an installed firewall, or uninstall the firewall.
- Manual MySQL Enterprise Firewall installation involves running a script located in the `share` directory of your MySQL installation.

Important

Read this entire section before following its instructions. Parts of the procedure differ depending on your environment.

Note

If installed, MySQL Enterprise Firewall involves some minimal overhead even when disabled. To avoid this overhead, do not install the firewall unless you plan to use it.

Note

MySQL Enterprise Firewall does not work together with the query cache. If the query cache is enabled, disable it before installing the firewall (see Query Cache Configuration).

For usage instructions, see Section 6.4.7.3, “Using MySQL Enterprise Firewall”. For reference information, see Section 6.4.7.4, “MySQL Enterprise Firewall Reference”.

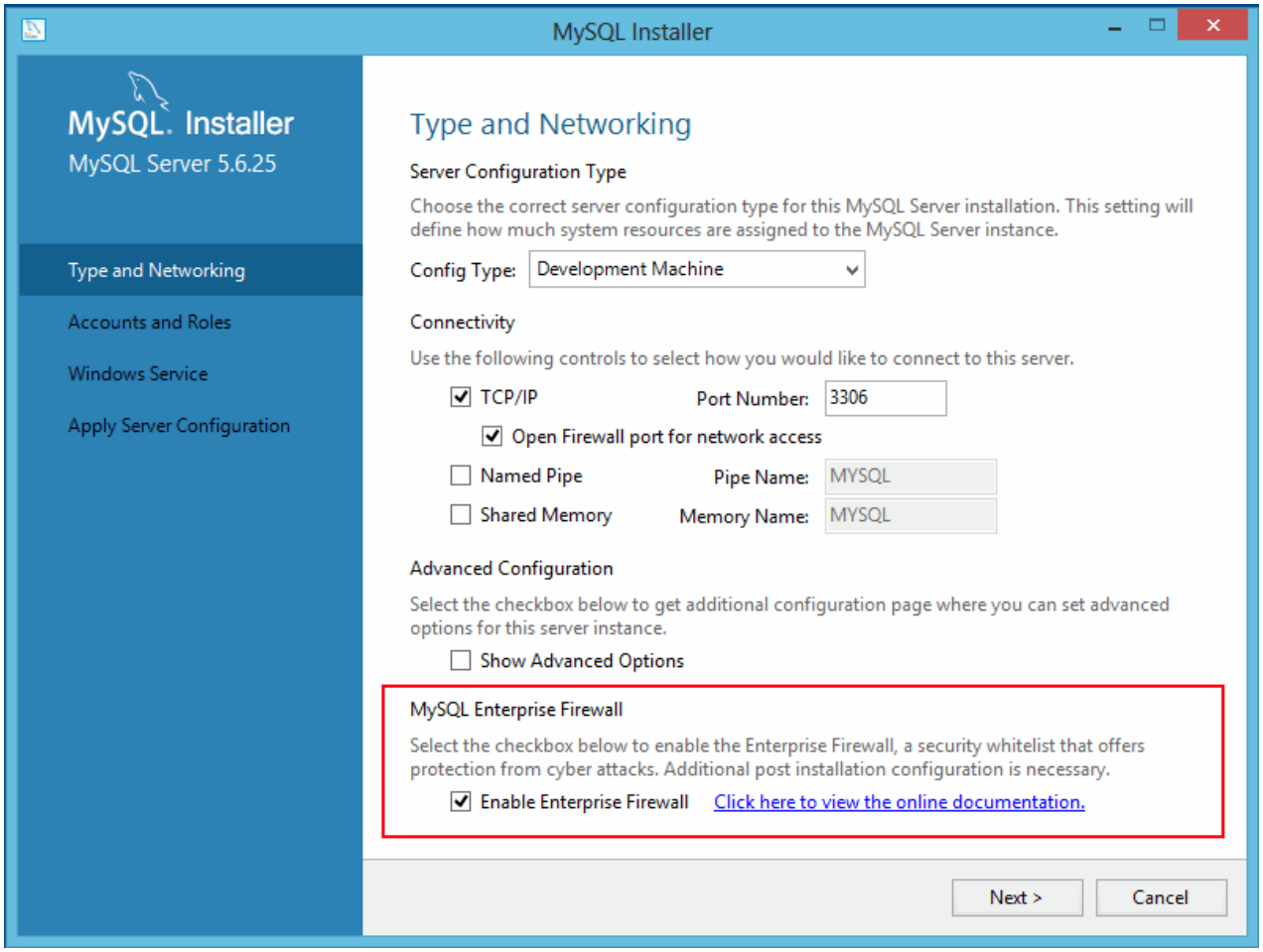
- Installing MySQL Enterprise Firewall
- Uninstalling MySQL Enterprise Firewall

Installing MySQL Enterprise Firewall

If MySQL Enterprise Firewall is already installed from an older version of MySQL, uninstall it using the instructions given later in this section and then restart your server before installing the current version. In this case, it is also necessary to register your configuration again.

On Windows, you can use MySQL Installer to install MySQL Enterprise Firewall, as shown in Figure 6.2, "MySQL Enterprise Firewall Installation on Windows". Check the **Enable Enterprise Firewall** checkbox. (**Open Firewall port for network access** has a different purpose. It refers to Windows Firewall and controls whether Windows blocks the TCP/IP port on which the MySQL server listens for client connections.)

Figure 6.2 MySQL Enterprise Firewall Installation on Windows



To install MySQL Enterprise Firewall using MySQL Workbench 6.3.4 or higher, see MySQL Enterprise Firewall Interface.

To install MySQL Enterprise Firewall manually, look in the `share` directory of your MySQL installation and choose the script that is appropriate for your platform. The available scripts differ in the suffix used to refer to the plugin library file:

- `win_install_firewall.sql`: Choose this script for Windows systems that use `.dll` as the file name suffix.

- `linux_install_firewall.sql`: Choose this script for Linux and similar systems that use `.so` as the file name suffix.

The installation script creates stored procedures in the default database, so choose a database to use. Then run the script as follows, naming the chosen database on the command line. The example here uses the `mysql` database and the Linux installation script. Make the appropriate substitutions for your system.

```
shell> mysql -u root -p mysql < linux_install_firewall.sql
Enter password: (enter root password here)
```

Note

To use MySQL Enterprise Firewall in the context of master/slave replication, Group Replication, or InnoDB cluster, you must prepare the slave or secondary nodes prior to running the installation script on the master or primary node. This is necessary because the `INSTALL PLUGIN` statements in the script are not replicated.

1. On each slave or secondary node, extract the `INSTALL PLUGIN` statements from the installation script and execute them manually.
2. On the master or primary node, run the installation script as described previously.

Installing MySQL Enterprise Firewall either using a graphical interface or manually should enable the firewall. To verify that, connect to the server and execute this statement:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'mysql_firewall_mode';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| mysql_firewall_mode | ON    |
+---------------------+-------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

Uninstalling MySQL Enterprise Firewall

MySQL Enterprise Firewall can be uninstalled using MySQL Workbench or manually.

To uninstall MySQL Enterprise Firewall using MySQL Workbench 6.3.4 or higher, see MySQL Enterprise Firewall Interface.

To uninstall MySQL Enterprise Firewall manually, execute the following statements. It is assumed that the stored procedures were created in the `mysql` database. Adjust the `DROP PROCEDURE` statements appropriately if the procedures were created in a different database.

```
DROP TABLE mysql.firewall_whitelist;
DROP TABLE mysql.firewall_users;
UNINSTALL PLUGIN mysql_firewall;
UNINSTALL PLUGIN mysql_firewall_whitelist;
UNINSTALL PLUGIN mysql_firewall_users;
DROP FUNCTION set_firewall_mode;
DROP FUNCTION normalize_statement;
DROP FUNCTION read_firewall_whitelist;
DROP FUNCTION read_firewall_users;
```

```
DROP FUNCTION mysql_firewall_flush_status;
DROP PROCEDURE mysql.sp_set_firewall_mode;
DROP PROCEDURE mysql.sp_reload_firewall_rules;
```

6.4.7.3 Using MySQL Enterprise Firewall

Before using MySQL Enterprise Firewall, install it according to the instructions provided in Section 6.4.7.2, “Installing or Uninstalling MySQL Enterprise Firewall”. Also, MySQL Enterprise Firewall does not work together with the query cache; disable the query cache if it is enabled (see Query Cache Configuration).

This section describes how to configure MySQL Enterprise Firewall using SQL statements. Alternatively, MySQL Workbench 6.3.4 or higher provides a graphical interface for firewall control. See MySQL Enterprise Firewall Interface.

To enable or disable the firewall, set the `mysql_firewall_mode` system variable. By default, this variable is enabled when the firewall is installed. To control the initial firewall state explicitly, you can set the variable at server startup. For example, to enable the firewall in an option file, use these lines:

```
[mysqld]
mysql_firewall_mode=ON
```

It is also possible to disable or enable the firewall at runtime:

```
mysql> SET GLOBAL mysql_firewall_mode = OFF;
mysql> SET GLOBAL mysql_firewall_mode = ON;
```

In addition to the global on/off firewall mode, each account registered with the firewall has its own operational mode. For an account in recording mode, the firewall learns an application's “fingerprint,” that is, the acceptable statement patterns that, taken together, form a whitelist. After training, switch the firewall to protecting mode to harden MySQL against access by statements that deviate from the fingerprint. For additional training, switch the firewall back to recording mode as necessary to update the whitelist with new statement patterns. An intrusion-detection mode is available that writes suspicious statements to the error log but does not deny access.

The firewall maintains whitelist rules on a per-account basis, enabling implementation of protection strategies such as these:

- For an application that has unique protection requirements, configure it to use an account that is not used for any other purpose.
- For applications that are related and share protection requirements, configure them as a group to use the same account.

Firewall operation is based on conversion of SQL statements to normalized digest form. Firewall digests are like the statement digests used by the Performance Schema (see Section 26.10, “Performance Schema Statement Digests and Sampling”). However, unlike the Performance Schema, the relevant digest-related system variable is `max_digest_length`.

For a connection from a registered account, the firewall converts each incoming statement to normalized form and processes it according to the account mode:

- In recording mode, the firewall adds the normalized statement to the account whitelist rules.
- In protecting mode, the firewall compares the normalized statement to the account whitelist rules. If there is a match, the statement passes and the server continues to process it. Otherwise, the server rejects

the statement and returns an error to the client. The firewall also writes the rejected statement to the error log if the `mysql_firewall_trace` system variable is enabled.

- In detecting mode, the firewall matches statements as in protecting mode, but writes nonmatching statements to the error log without denying access.

Accounts that have a mode of `OFF` or are not registered with the firewall are ignored by it.

To protect an account using MySQL Enterprise Firewall, follow these steps:

1. Register the account and put it in recording mode.
2. Connect to the MySQL server using the registered account and execute statements to be learned. This establishes the account's whitelist of accepted statements.
3. Switch the registered account to protecting mode.

The following example shows how to register an account with the firewall, use the firewall to learn acceptable statements for that account, and protect the account against execution of unacceptable statements. The example account, `'fwuser'@'localhost'`, is for use by an application that accesses tables in the `sakila` database. (This database is available at https://dev.mysql.com/doc/index-other.html.)

Note

The user and host parts of the account name are quoted separately for statements such as `CREATE USER` and `GRANT`, whereas to specify an account for use with a firewall component, name it as a single quoted string `'fwuser@localhost'`.

The convention for naming accounts as a single quoted string for firewall components means that you cannot use accounts that have embedded `@` characters in the user name.

Perform the steps in the following procedure using an administrative MySQL account, except those designated for execution by the account registered with the firewall. The default database should be `sakila` for statements executed using the registered account.

1. If necessary, create the account to be protected (choose an appropriate password) and grant it privileges for the `sakila` database:

```
mysql> CREATE USER 'fwuser'@'localhost' IDENTIFIED BY 'fWp@3sw0rd';
mysql> GRANT ALL ON sakila.* TO 'fwuser'@'localhost';
```

2. Use the `sp_set_firewall_mode()` stored procedure to register the account with the firewall and place it in recording mode (if the procedure is located in a database other than `mysql`, adjust the statement accordingly):

```
mysql> CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'RECORDING');
```

   During the course of its execution, the stored procedure invokes firewall user-defined functions, which may produce output of their own.

3. Using the registered account, connect to the server, then execute some statements that are legitimate for it:

```
mysql> SELECT first_name, last_name FROM customer WHERE customer_id = 1;
mysql> UPDATE rental SET return_date = NOW() WHERE rental_id = 1;
mysql> SELECT get_customer_balance(1, NOW());
```

The firewall converts the statements to digest form and records them in the account whitelist.

Note

Until the account executes statements in recording mode, its whitelist is empty, which is equivalent to “deny all.” If switched to protecting mode, the account will be effectively prohibited from executing statements.

4. At this point, the user and whitelist information is cached and can be seen in the firewall `INFORMATION_SCHEMA` tables:

```
mysql> SELECT MODE FROM INFORMATION_SCHEMA.MYSQL_FIREWALL_USERS
       WHERE USERHOST = 'fwuser@localhost';
+-----------+
| MODE      |
+-----------+
| RECORDING |
+-----------+
mysql> SELECT RULE FROM INFORMATION_SCHEMA.MYSQL_FIREWALL_WHITELIST
       WHERE USERHOST = 'fwuser@localhost';
+-----------------------------------------------------------------------------+
| RULE                                                                        |
+-----------------------------------------------------------------------------+
| SELECT `first_name` , `last_name` FROM `customer` WHERE `customer_id` = ?  |
| SELECT `get_customer_balance` ( ? , NOW ( ) )                               |
| UPDATE `rental` SET `return_date` = NOW ( ) WHERE `rental_id` = ?           |
| SELECT @@`version_comment` LIMIT ?                                          |
+-----------------------------------------------------------------------------+
```

Note

The `@@version_comment` rule comes from a statement sent automatically by the `mysql` client when you connect to the server as the registered user.

It is important to train the firewall under conditions matching application use. For example, a given MySQL connector might send statements to the server at the beginning of a connection to determine server characteristics and capabilities. If an application normally is used through that connector, train the firewall that way, too. That enables those initial statements to become part of the whitelist for the account associated with the application.

5. Use the stored procedure to switch the registered user to protecting mode:

```
mysql> CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'PROTECTING');
```

Important

Switching the account out of `RECORDING` mode synchronizes its firewall cache data to the underlying `mysql` system database tables for persistent storage. If you do not switch the mode for a user who is being recorded, the cached whitelist data is not written to the system tables and will be lost when the server is restarted.

6. Using the registered account, execute some acceptable and unacceptable statements. The firewall matches each one against the account whitelist and accepts or rejects it.

This statement is not identical to a training statement but produces the same normalized statement as one of them, so the firewall accepts it:

```
mysql> SELECT first_name, last_name FROM customer WHERE customer_id = '48';
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| ANN        | EVANS     |
+------------+-----------+
```

These statements do not match anything in the whitelist and each results in an error:

```
mysql> SELECT first_name, last_name FROM customer WHERE customer_id = 1 OR TRUE;
ERROR 1045 (28000): Statement was blocked by Firewall
mysql> SHOW TABLES LIKE 'customer%';
ERROR 1045 (28000): Statement was blocked by Firewall
mysql> TRUNCATE TABLE mysql.slow_log;
ERROR 1045 (28000): Statement was blocked by Firewall
```

The firewall also writes the rejected statements to the error log if the `mysql_firewall_trace` system variable is enabled. For example:

```
[Note] Plugin MYSQL_FIREWALL reported:
'ACCESS DENIED for fwuser@localhost. Reason: No match in whitelist.
Statement: TRUNCATE TABLE `mysql` . `slow_log` '
```

You can use these log messages in your efforts to identify the source of attacks.

7. You can log nonmatching statements as suspicious without denying access. To do this, put the account in intrusion-detecting mode:

```
mysql> CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'DETECTING');
```

8. Using the registered account, connect to the server, then execute a statement that does not match the whitelist:

```
mysql> SHOW TABLES LIKE 'customer%';
+-----------------------------+
| Tables_in_sakila (customer%) |
+-----------------------------+
| customer                    |
| customer_list               |
+-----------------------------+
```

In detecting mode, the firewall permits the nonmatching statement to execute but writes a message to the error log:

```
[Note] Plugin MYSQL_FIREWALL reported:
'SUSPICIOUS STATEMENT from 'fwuser@localhost'. Reason: No match in whitelist.
Statement: SHOW TABLES LIKE ? '
```

Note

Detection mode writes messages as Notes, which are information messages. To ensure that such messages appear in the error log and are not discarded, make sure that the `log_error_verbosity` system variable is set to a value of 3.

9. To assess firewall activity, examine its status variables:

```
mysql> SHOW GLOBAL STATUS LIKE 'Firewall%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Firewall_access_denied     | 3     |
| Firewall_access_granted    | 4     |
| Firewall_access_suspicious | 1     |
| Firewall_cached_entries    | 4     |
+----------------------------+-------+
```

   The variables indicate the number of statements rejected, accepted, logged as suspicious, and added to the cache, respectively. The `Firewall_access_granted` count is 4 because of the `@@version_comment` statement sent by the `mysql` client each of the three time you used it to connect as the registered user, plus the `SHOW TABLES` statement that was not blocked in `DETECTING` mode.

Should additional training for an account be necessary, switch it to recording mode again, then back to protecting mode after executing statements to be added to the whitelist.

6.4.7.4 MySQL Enterprise Firewall Reference

The following discussion serves as a reference to MySQL Enterprise Firewall components:

- MySQL Enterprise Firewall Tables
- MySQL Enterprise Firewall Procedures and Functions
- MySQL Enterprise Firewall System Variables
- MySQL Enterprise Firewall Status Variables

MySQL Enterprise Firewall Tables

MySQL Enterprise Firewall maintains account and whitelist information. It uses `INFORMATION_SCHEMA` tables to provide views into cached data, and tables in the `mysql` system database to store this data in persistent form. When enabled, the firewall bases its operational decisions on the cached data.

The `INFORMATION_SCHEMA` tables are accessible by anyone. The `mysql` tables can be accessed only by users with privileges for that database.

The `INFORMATION_SCHEMA.MYSQL_FIREWALL_USERS` and `mysql.firewall_users` tables list registered firewall accounts and their operational modes. The tables have these columns:

- `USERHOST`

  An account registered with the firewall. Each account has the format *`user_name@host_name`* and represents actual user and host names as authenticated by the server. Patterns and netmasks should not be used when registering users.

- `MODE`

  The current firewall operational mode for the account. The permitted mode values are `OFF`, `DETECTING`, `PROTECTING`, `RECORDING`, and `RESET`. For details about their meanings, see the description of `sp_set_firewall_mode()` in MySQL Enterprise Firewall Procedures and Functions.

The `INFORMATION_SCHEMA.MYSQL_FIREWALL_WHITELIST` and `mysql.firewall_whitelist` tables list registered firewall accounts and their whitelists. The tables have these columns:

- `USERHOST`

  An account registered with the firewall. The format is the same as for the user account tables.

- `RULE`

  A normalized statement indicating an acceptable statement pattern for the account. An account whitelist is the union of its rules.

- `ID`

  An integer column that is a primary key for the table. This column was added in MySQL 8.0.12.

MySQL Enterprise Firewall Procedures and Functions

MySQL Enterprise Firewall has stored procedures that perform tasks such as registering MySQL accounts with the firewall, establishing their operational mode, and managing transfer of firewall data between the cache and the underlying system tables. It also has a set of user-defined functions (UDFs) that provides an SQL-level API for lower-level tasks such as synchronizing the cache with the underlying system tables.

Under normal operation, the stored procedures implement the user interface. The UDFs are invoked by the stored procedures, not directly by users.

To invoke a stored procedure when the default database is not the database that contains the procedure, qualify the procedure name with the database name. For example:

```
CALL mysql.sp_set_firewall_mode(user, mode);
```

The following list describes each firewall stored procedure and UDF:

- `sp_reload_firewall_rules(user)`

  This stored procedure uses firewall UDFs to reset a registered account and reload the in-memory rules for it from the rules stored in the `mysql.firewall_whitelist` table. This procedure provides control over firewall operation for individual accounts.

  The *`user`* argument names the affected account, as a string in *`user_name@host_name`* format.

  Example:

  ```
  CALL mysql.sp_reload_firewall_rules('fwuser@localhost');
  ```

  Warning

  This procedure sets the account mode to `RESET`, which clears the account whitelist and sets its mode to `OFF`. If the account mode was not `OFF` prior to the `sp_reload_firewall_rules()` call, use `sp_set_firewall_mode()` to restore its previous mode after reloading the rules. For example, if the account was in `PROTECTING` mode, that is no longer true after calling `sp_reload_firewall_rules()` and you must set it to `PROTECTING` again explicitly.

- `sp_set_firewall_mode(user, mode)`

  This stored procedure registers a MySQL account with the firewall and establishes its operational mode. The procedure also invokes firewall UDFs as necessary to transfer firewall data between the cache and the underlying system tables. This procedure may be called even if the `mysql_firewall_mode` system

variable is `OFF`, although setting the mode for an account has no operational effect while the firewall is disabled.

The *`user`* argument names the affected account, as a string in *`user_name@host_name`* format.

The *`mode`* is the operational mode for the user, as a string. These mode values are permitted:

- `OFF`: Disable the firewall for the account.
- `DETECTING`: Intrusion-detection mode: Write suspicious (nonmatching) statements to the error log but do not deny access.
- `PROTECTING`: Protect the account by matching incoming statements against the account whitelist.
- `RECORDING`: Training mode: Record acceptable statements for the account. Incoming statements that do not immediately fail with a syntax error are recorded to become part of the account whitelist rules.
- `RESET`: Clear the account whitelist and set the account mode to `OFF`.

Switching the mode for an account to any mode but `RECORDING` synchronizes the firewall cache data to the underlying `mysql` system database tables for persistent storage. Switching the mode from `OFF` to `RECORDING` reloads the whitelist from the `mysql.firewall_whitelist` table into the cache.

If an account has an empty whitelist, setting its mode to `PROTECTING` produces an error message that is returned in a result set, but not an SQL error:

```
mysql> CALL mysql.sp_set_firewall_mode('a@b','PROTECTING');
+-----------------------------------------------------------------------+
| set_firewall_mode(arg_userhost, arg_mode)                             |
+-----------------------------------------------------------------------+
| ERROR: PROTECTING mode requested for a@b but the whitelist is empty. |
+-----------------------------------------------------------------------+
1 row in set (0.02 sec)

Query OK, 0 rows affected (0.02 sec)
```

- `mysql_firewall_flush_status()`

  This UDF resets several firewall status variables to 0:

  ```
  Firewall_access_denied
  Firewall_access_granted
  Firewall_access_suspicious
  ```

  Example:

  ```
  SELECT mysql_firewall_flush_status();
  ```

- `normalize_statement(`*`stmt`*`)`

  This UDF normalizes an SQL statement into the digest form used for whitelist rules.

  Example:

  ```
  SELECT normalize_statement('SELECT * FROM t1 WHERE c1 > 2');
  ```

- `read_firewall_users(`*`user, mode`*`)`

This aggregate UDF updates the firewall user cache through a `SELECT` statement on the `mysql.firewall_users` table.

Example:

```
SELECT read_firewall_users('fwuser@localhost', 'RECORDING')
FROM mysql.firewall_users;
```

- `read_firewall_whitelist(user, rule)`

  This aggregate UDF updates the recorded statement cache through a `SELECT` statement on the `mysql.firewall_whitelist` table.

  Example:

```
SELECT read_firewall_whitelist('fwuser@localhost', 'RECORDING')
FROM mysql.firewall_whitelist;
```

- `set_firewall_mode(user, mode)`

  This UDF manages the user cache and establishes the user operational mode.

  Example:

```
SELECT set_firewall_mode('fwuser@localhost', 'RECORDING');
```

MySQL Enterprise Firewall System Variables

MySQL Enterprise Firewall supports the following system variables. Use them to configure firewall operation. These variables are unavailable unless the firewall is installed (see Section 6.4.7.2, “Installing or Uninstalling MySQL Enterprise Firewall”).

- `mysql_firewall_mode`

Property	Value
Command-Line Format	`--mysql-firewall-mode[={OFF|ON}]`
Introduced	8.0.11
System Variable	`mysql_firewall_mode`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

  Whether MySQL Enterprise Firewall is enabled (the default) or disabled.

- `mysql_firewall_trace`

Property	Value
Command-Line Format	`--mysql-firewall-trace[={OFF|ON}]`
Introduced	8.0.11

Property	Value
System Variable	`mysql_firewall_trace`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether the MySQL Enterprise Firewall trace is enabled or disabled (the default). When `mysql_firewall_trace` is enabled, for `PROTECTING` mode, the firewall writes rejected statements to the error log.

MySQL Enterprise Firewall Status Variables

MySQL Enterprise Firewall supports the following status variables. Use them to obtain information about firewall operational status. These variables are unavailable unless the firewall is installed (see Section 6.4.7.2, "Installing or Uninstalling MySQL Enterprise Firewall"). Firewall status variables are set to 0 whenever the `MYSQL_FIREWALL` plugin is installed or the server is started. Many of them are reset to zero by the `mysql_firewall_flush_status()` UDF (see MySQL Enterprise Firewall Procedures and Functions).

- `Firewall_access_denied`

  The number of statements rejected by MySQL Enterprise Firewall.

- `Firewall_access_granted`

  The number of statements accepted by MySQL Enterprise Firewall.

- `Firewall_access_suspicious`

  The number of statements logged by MySQL Enterprise Firewall as suspicious for users who are in `DETECTING` mode.

- `Firewall_cached_entries`

  The number of statements recorded by MySQL Enterprise Firewall, including duplicates.

6.4.8 MySQL Enterprise Data Masking and De-Identification

Note

MySQL Enterprise Data Masking and De-Identification is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, https://www.mysql.com/products/.

As of MySQL 8.0.13, MySQL Enterprise Edition provides data masking and de-identification capabilities:

- Transformation of existing data to mask it and remove identifying characteristics, such as changing all digits of a credit card number but the last four to `'X'` characters.
- Generation of random data, such as email addresses and payment card numbers.

The way that applications use these capabilities depends on the purpose for which the data will be used and who will access it:

- Applications that use sensitive data may protect it by performing data masking and permitting use of partially masked data for client identification. Example: A call center may ask for clients to provide their last four Social Security number digits.
- Applications that require properly formatted data, but not necessarily the original data, can synthesize sample data. Example: An application developer who is testing data validators but has no access to original data may synthesize random data with the same format.

Example 1:

Medical research facilities can hold patient data that comprises a mix of personal and medical data. This may include genetic sequences (long strings), test results stored in JSON format, and other data types. Although the data may be used mostly by automated analysis software, access to genome data or test results of particular patients is still possible. In such cases, data masking should be used to render this information not personally identifiable.

Example 2:

A credit card processor company provides a set of services using sensitive data, such as:

- Processing a large number of financial transactions per second.
- Storing a large amount of transaction-related data.
- Protecting transaction-related data with strict requirements for personal data.
- Handling client complaints about transactions using reversible or partially masked data.

A typical transaction may include many types of sensitive information, including:

- Credit card number.
- Transaction type and amount.
- Merchant type.
- Transaction cryptogram (to confirm transaction legitimacy).
- Geolocation of GPS-equipped terminal (for fraud detection).

Those types of information may then be joined within a bank or other card-issuing financial institution with client personal data, such as:

- Full client name (either person or company).
- Address.
- Date of birth.
- Social Security number.
- Email address.
- Phone number.

Various employee roles within both the card processing company and the financial institution require access to that data. Some of these roles may require access only to masked data. Other roles may require access to the original data on a case-to-case basis, which is recorded in audit logs.

Masking and de-identification are core to regulatory compliance, so MySQL Enterprise Data Masking and De-Identification can help application developers satisfy privacy requirements:

- PCI – DSS: Payment Card Data.
- HIPAA: Privacy of Health Data, Health Information Technology for Economic and Clinical Health Act (HITECH Act).
- EU General Data Protection Directive (GDPR): Protection of Personal Data.
- Data Protection Act (UK): Protection of Personal Data.
- Sarbanes Oxley, GLBA, The USA Patriot Act, Identity Theft and Assumption Deterrence Act of 1998.
- FERPA – Student Data, NASD, CA SB1386 and AB 1950, State Data Protection Laws, Basel II.

The following sections describe the components of MySQL Enterprise Data Masking and De-Identification, discuss how to install and use it, and provide reference information for its components.

6.4.8.1 MySQL Enterprise Data Masking and De-Identification Components

MySQL Enterprise Data Masking and De-Identification is based on a plugin library that implements these components:

- A server-side plugin named `data_masking`.
- A set of user-defined functions (UDFs) provides an SQL-level API for performing masking and de-identification operations. Some of these functions require the `SUPER` privilege.

6.4.8.2 Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification

This section describes how to install or uninstall MySQL Enterprise Data Masking and De-Identification, which is implemented as a plugin library file containing a plugin and user-defined functions (UDFs). For general information about installing or uninstalling plugins and UDFs, see Section 5.6.1, “Installing and Uninstalling Plugins”, and Section 5.7.1, “Installing and Uninstalling User-Defined Functions”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `data_masking`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To install the MySQL Enterprise Data Masking and De-Identification plugin and UDFs, use the `INSTALL PLUGIN` and `CREATE FUNCTION` statements (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN data_masking SONAME 'data_masking.so';
CREATE FUNCTION gen_blacklist RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_dictionary RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_dictionary_drop RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_dictionary_load RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_range RETURNS INTEGER
  SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_email RETURNS STRING
  SONAME 'data_masking.so';
```

```
CREATE FUNCTION gen_rnd_pan RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_ssn RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION gen_rnd_us_phone RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION mask_inner RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION mask_outer RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION mask_pan RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION mask_pan_relaxed RETURNS STRING
  SONAME 'data_masking.so';
CREATE FUNCTION mask_ssn RETURNS STRING
  SONAME 'data_masking.so';
```

If the plugin and UDFs are used on a master replication server, install them on all slave servers as well to avoid replication issues.

Once installed as just described, the plugin and UDFs remain installed until uninstalled. To remove them, use the `UNINSTALL PLUGIN` and `DROP FUNCTION` statements:

```
UNINSTALL PLUGIN data_masking;
DROP FUNCTION gen_blacklist;
DROP FUNCTION gen_dictionary;
DROP FUNCTION gen_dictionary_drop;
DROP FUNCTION gen_dictionary_load;
DROP FUNCTION gen_range;
DROP FUNCTION gen_rnd_email;
DROP FUNCTION gen_rnd_pan;
DROP FUNCTION gen_rnd_ssn;
DROP FUNCTION gen_rnd_us_phone;
DROP FUNCTION mask_inner;
DROP FUNCTION mask_outer;
DROP FUNCTION mask_pan;
DROP FUNCTION mask_pan_relaxed;
DROP FUNCTION mask_ssn;
```

6.4.8.3 Using MySQL Enterprise Data Masking and De-Identification

Before using MySQL Enterprise Data Masking and De-Identification, install it according to the instructions provided at Section 6.4.8.2, “Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification”.

To use MySQL Enterprise Data Masking and De-Identification in applications, invoke the functions that are appropriate for the operations you wish to perform. For detailed function descriptions, see Section 6.4.8.4, “MySQL Enterprise Data Masking and De-Identification User-Defined Function Reference”. This section demonstrates how to use the functions to carry out some representative tasks. It first presents an overview of the available functions, followed by some examples of how the functions might be used in real-world context:

- Masking Data to Remove Identifying Characteristics
- Generating Random Data with Specific Characteristics
- Generating Random Data Using Dictionaries
- Using Masked Data for Customer Identification
- Creating Views that Display Masked Data

Masking Data to Remove Identifying Characteristics

MySQL provides general-purpose masking functions that mask arbitrary strings, and special-purpose masking functions that mask specific types of values.

General-Purpose Masking Functions

`mask_inner()` and `mask_outer()` are general-purpose functions that mask parts of arbitrary strings based on position within the string:

- `mask_inner()` masks the interior of its string argument, leaving the ends unmasked. Other arguments specify the sizes of the unmasked ends.

```
mysql> SELECT mask_inner('This is a string', 5, 1);
+--------------------------------------+
| mask_inner('This is a string', 5, 1) |
+--------------------------------------+
| This XXXXXXXXXg                      |
+--------------------------------------+
mysql> SELECT mask_inner('This is a string', 1, 5);
+--------------------------------------+
| mask_inner('This is a string', 1, 5) |
+--------------------------------------+
| TXXXXXXXXXXtring                     |
+--------------------------------------+
```

- `mask_outer()` does the reverse, masking the ends of its string argument, leaving the interior unmasked. Other arguments specify the sizes of the masked ends.

```
mysql> SELECT mask_outer('This is a string', 5, 1);
+--------------------------------------+
| mask_outer('This is a string', 5, 1) |
+--------------------------------------+
| XXXXXis a strinX                     |
+--------------------------------------+
mysql> SELECT mask_outer('This is a string', 1, 5);
+--------------------------------------+
| mask_outer('This is a string', 1, 5) |
+--------------------------------------+
| Xhis is a sXXXXX                     |
+--------------------------------------+
```

By default, `mask_inner()` and `mask_outer()` use `'X'` as the masking character, but permit an optional masking-character argument:

```
mysql> SELECT mask_inner('This is a string', 5, 1, '*');
+-----------------------------------------+
| mask_inner('This is a string', 5, 1, '*') |
+-----------------------------------------+
| This **********g                        |
+-----------------------------------------+
mysql> SELECT mask_outer('This is a string', 5, 1, '#');
+-----------------------------------------+
| mask_outer('This is a string', 5, 1, '#') |
+-----------------------------------------+
| #####is a strin#                        |
+-----------------------------------------+
```

Special-Purpose Masking Functions

Other masking functions expect a string argument representing a specific type of value and mask it to remove identifying characteristics.

Note

The examples here supply function arguments using the random value generation functions that return the appropriate type of value. For more information about generation functions, see Generating Random Data with Specific Characteristics.

Payment card Primary Account Number masking. Masking functions provide strict and relaxed masking of Primary Account Numbers.

- `mask_pan()` masks all but the last four digits of the number:

```
mysql> SELECT mask_pan(gen_rnd_pan());
+-------------------------+
| mask_pan(gen_rnd_pan()) |
+-------------------------+
| XXXXXXXXXXXX2461        |
+-------------------------+
```

- `mask_pan_relaxed()` is similar but does not mask the first six digits that indicate the payment card issuer unmasked:

```
mysql> SELECT mask_pan_relaxed(gen_rnd_pan());
+---------------------------------+
| mask_pan_relaxed(gen_rnd_pan()) |
+---------------------------------+
| 770630XXXXXX0807                |
+---------------------------------+
```

U.S. Social Security number masking. `mask_ssn()` masks all but the last four digits of the number:

```
mysql> SELECT mask_ssn(gen_rnd_ssn());
+-------------------------+
| mask_ssn(gen_rnd_ssn()) |
+-------------------------+
| XXX-XX-1723             |
+-------------------------+
```

Generating Random Data with Specific Characteristics

Several functions generate random values. These values can be used for testing, simulation, and so forth.

`gen_range()` returns a random integer selected from a given range:

```
mysql> SELECT gen_range(1, 10);
+------------------+
| gen_range(1, 10) |
+------------------+
|                6 |
+------------------+
```

`gen_rnd_email()` returns a random email address in the `example.com` domain:

```
mysql> SELECT gen_rnd_email();
+---------------------------+
| gen_rnd_email()           |
+---------------------------+
| ayxnq.xmkpvvy@example.com |
+---------------------------+
```

`gen_rnd_pan()` returns a random payment card Primary Account Number:

```
mysql> SELECT gen_rnd_pan();
```

(The `gen_rnd_pan()` function result is not shown because its return values should be used only for testing purposes, and not for publication. It cannot be guaranteed the number is not assigned to a legitimate payment account.)

`gen_rnd_ssn()` returns a random U.S. Social Security number with the first and second parts each chosen from a range not used for legitimate numbers:

```
mysql> SELECT gen_rnd_ssn();
+---------------+
| gen_rnd_ssn() |
+---------------+
| 912-45-1615   |
+---------------+
```

`gen_rnd_us_phone()` returns a random U.S. phone number in the 555 area code not used for legitimate numbers:

```
mysql> SELECT gen_rnd_us_phone();
+--------------------+
| gen_rnd_us_phone() |
+--------------------+
| 1-555-747-5627     |
+--------------------+
```

Generating Random Data Using Dictionaries

MySQL Enterprise Data Masking and De-Identification enables dictionaries to be used as sources of random values. To use a dictionary, it must first be loaded from a file and given a name. Each loaded dictionary becomes part of the dictionary registry. Items then can be selected from registered dictionaries and used as random values or as replacements for other values.

A valid dictionary file has these characteristics:

- The file contents are plain text, one term per line.
- Empty lines are ignored.
- The file must contain at least one term.

Suppose that a file named `de_cities.txt` contains these city names in Germany:

```
Berlin
Munich
Bremen
```

Also suppose that a file named `us_cities.txt` contains these city names in the United States:

```
Chicago
Houston
Phoenix
El Paso
Detroit
```

Assume that the `secure_file_priv` system variable is set to `/usr/local/mysql/mysql-files`. In that case, copy the dictionary files to that directory so that the MySQL server can access them. Then use `gen_dictionary_load()` to load the dictionaries into the dictionary registry and assign them names:

```
mysql> SELECT gen_dictionary_load('/usr/local/mysql/mysql-files/de_cities.txt', 'DE_Cities');
+-----------------------------------------------------------------------------+
| gen_dictionary_load('/usr/local/mysql/mysql-files/de_cities.txt', 'DE_Cities') |
+-----------------------------------------------------------------------------+
| Dictionary load success                                                     |
+-----------------------------------------------------------------------------+
mysql> SELECT gen_dictionary_load('/usr/local/mysql/mysql-files/us_cities.txt', 'US_Cities');
+-----------------------------------------------------------------------------+
| gen_dictionary_load('/usr/local/mysql/mysql-files/us_cities.txt', 'US_Cities') |
+-----------------------------------------------------------------------------+
| Dictionary load success                                                     |
+-----------------------------------------------------------------------------+
```

To select a random term from a dictionary, use `gen_dictionary()`:

```
mysql> SELECT gen_dictionary('DE_Cities');
+----------------------------+
| gen_dictionary('DE_Cities') |
+----------------------------+
| Berlin                     |
+----------------------------+
mysql> SELECT gen_dictionary('US_Cities');
+----------------------------+
| gen_dictionary('US_Cities') |
+----------------------------+
| Phoenix                    |
+----------------------------+
```

To select a random term from multiple dictionaries, randomly select one of the dictionaries, then select a term from it:

```
mysql> SELECT gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities'));
+-------------------------------------------------------------+
| gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities')) |
+-------------------------------------------------------------+
| Detroit                                                     |
+-------------------------------------------------------------+
mysql> SELECT gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities'));
+-------------------------------------------------------------+
| gen_dictionary(ELT(gen_range(1,2), 'DE_Cities', 'US_Cities')) |
+-------------------------------------------------------------+
| Bremen                                                      |
+-------------------------------------------------------------+
```

The `gen_blacklist()` function enables a term from one dictionary to be replaced by a term from another dictionary, which effects masking by substitution. Its arguments are the term to replace, the dictionary in which the term appears, and the dictionary from which to choose a replacement. For example, to substitute a U.S. city for a German city, or vice versa, use `gen_blacklist()` like this:

```
mysql> SELECT gen_blacklist('Munich', 'DE_Cities', 'US_Cities');
+-------------------------------------------------+
| gen_blacklist('Munich', 'DE_Cities', 'US_Cities') |
+-------------------------------------------------+
| Houston                                         |
+-------------------------------------------------+
mysql> SELECT gen_blacklist('El Paso', 'US_Cities', 'DE_Cities');
+-------------------------------------------------+
| gen_blacklist('El Paso', 'US_Cities', 'DE_Cities') |
+-------------------------------------------------+
| Bremen                                          |
+-------------------------------------------------+
```

If the term to replace is not in the first dictionary, `gen_blacklist()` returns it unchanged:

```
mysql> SELECT gen_blacklist('Moscow', 'DE_Cities', 'US_Cities');
+-------------------------------------------------+
| gen_blacklist('Moscow', 'DE_Cities', 'US_Cities') |
+-------------------------------------------------+
| Moscow                                          |
+-------------------------------------------------+
```

Using Masked Data for Customer Identification

At customer-service call centers, one common identity verification technique is to ask customers to provide their last four Social Security number (SSN) digits. For example, a customer might say her name is Joanna Bond and that her last four SSN digits are `0007`.

Suppose that a `customer` table containing customer records has these columns:

- `id`: Customer ID number.
- `first_name`: Customer first name.
- `last_name`: Customer last name.
- `ssn`: Customer Social Security number.

The application used by customer-service representatives to check the customer SSN might execute a query like this:

```
mysql> SELECT id, ssn
mysql> FROM customer
mysql> WHERE first_name = 'Joanna' AND last_name = 'Bond';
+-----+-------------+
| id  | ssn         |
+-----+-------------+
| 786 | 906-39-0007 |
+-----+-------------+
```

However, that exposes the SSN to the customer-service representative, who has no need to see anything but the last four digits. Instead, the application can use this query to display only the masked SSN:

```
mysql> SELECT id, mask_ssn(CONVERT(ssn USING binary))
mysql> FROM customer
mysql> WHERE first_name = 'Joanna' AND last_name = 'Bond';
+-----+-----------------------------------+
| id  | mask_ssn(CONVERT(ssn USING binary)) |
+-----+-----------------------------------+
| 786 | XXX-XX-0007                       |
+-----+-----------------------------------+
```

Now the representative sees only what is necessary, and customer privacy is preserved.

Why was the `CONVERT()` function used for the argument to `mask_ssn()`? Because `mask_ssn()` requires an argument of length 11, and because UDFs treat string arguments as binary strings, with one byte per character. Thus, even though `ssn` is defined as `VARCHAR(11)`, if the `ssn` column has a multibyte character set, it appears to be longer than 11 bytes when passed to a UDF, and an error occurs. Converting the value to a binary string ensures that the UDF sees an argument of length 11.

A similar technique may be needed for other data masking functions when string arguments do not have a single-byte character set.

Creating Views that Display Masked Data

If masked data from a table is used for multiple queries, it may be convenient to define a view that produces masked data. That way, applications can select from the view without performing masking in individual queries.

For example, a masking view on the `customer` table from the previous section can be defined like this:

```
CREATE VIEW masked_customer AS
SELECT id, first_name, last_name, mask_ssn(CONVERT(ssn USING binary)) AS ssn
FROM customer;
```

Then the query to look up a customer becomes simpler but still returns masked data:

```
mysql> SELECT id, ssn
mysql> FROM masked_customer
mysql> WHERE first_name = 'Joanna' AND last_name = 'Bond';
+-----+-------------+
| id  | ssn         |
+-----+-------------+
| 786 | XXX-XX-0007 |
+-----+-------------+
```

6.4.8.4 MySQL Enterprise Data Masking and De-Identification User-Defined Function Reference

The MySQL Enterprise Data Masking and De-Identification plugin library includes several user-defined functions (UDFs), which may be grouped into these categories:

- Data Masking Functions
- Random Data Generation Functions
- Random Data Dictionary-Based Functions

These UDFs treat string arguments as binary strings, which means they are implicitly case sensitive. In addition, string UDF return values are binary strings. If a string return value should be in a different character set, convert it. The following example shows how to convert the result of `gen_rnd_email()` to the `utf8mb4` character set:

```
SET @email = CONVERT(gen_rnd_email() USING utf8mb4);
```

It may also be necessary to convert string arguments, as illustrated in Using Masked Data for Customer Identification.

Data Masking Functions

Each function in this section performs a masking operation on its string argument and returns the masked result.

- `mask_inner(`*`str`*`, `*`margin1`*`, `*`margin2`*` [, `*`mask_char`*`])`

  Masks the interior part of a string, leaving the ends untouched, and returns the result. An optional masking character can be specified.

  Arguments:

  - *`str`*: The string to mask.

- *margin1*: A nonnegative integer that specifies the number of characters on the left end of the string to remain unmasked. If the value is 0, no left end characters remain unmasked.
- *margin2*: A nonnegative integer that specifies the number of characters on the right end of the string to remain unmasked. If the value is 0, no right end characters remain unmasked.
- *mask_char*: (Optional) The single character to use for masking. The default is 'X' if *mask_char* is not given.

  Because UDF string arguments are treated as binary strings, the masking character must be a single-byte character. Attempts to use a multibyte character produce an error.

Return value:

The masked string, or NULL if either margin is negative.

If the sum of the margin values is larger than the argument length, no masking occurs and the argument is returned unchanged.

Example:

```
mysql> SELECT mask_inner('abcdef', 1, 2), mask_inner('abcdef',0, 5);
+----------------------------+---------------------------+
| mask_inner('abcdef', 1, 2) | mask_inner('abcdef',0, 5) |
+----------------------------+---------------------------+
| aXXXef                     | Xbcdef                    |
+----------------------------+---------------------------+
mysql> SELECT mask_inner('abcdef', 1, 2, '*'), mask_inner('abcdef',0, 5, '#');
+---------------------------------+--------------------------------+
| mask_inner('abcdef', 1, 2, '*') | mask_inner('abcdef',0, 5, '#') |
+---------------------------------+--------------------------------+
| a***ef                          | #bcdef                         |
+---------------------------------+--------------------------------+
```

- mask_outer(*str*, *margin1*, *margin2* [, *mask_char*])

Masks the left and right ends of a string, leaving the interior unmasked, and returns the result. An optional masking character can be specified.

Arguments:

- *str*: The string to mask.
- *margin1*: A nonnegative integer that specifies the number of characters on the left end of the string to mask. If the value is 0, no left end characters are masked.
- *margin2*: A nonnegative integer that specifies the number of characters on the right end of the string to mask. If the value is 0, no right end characters are masked.
- *mask_char*: (Optional) The single character to use for masking. The default is 'X' if *mask_char* is not given.

  Because UDF string arguments are treated as binary strings, the masking character must be a single-byte character. Attempts to use a multibyte character produce an error.

Return value:

The masked string, or `NULL` if either margin is negative.

If the sum of the margin values is larger than the argument length, the entire argument is masked.

Example:

```
mysql> SELECT mask_outer('abcdef', 1, 2), mask_outer('abcdef',0, 5);
+---------------------------+--------------------------+
| mask_outer('abcdef', 1, 2) | mask_outer('abcdef',0, 5) |
+---------------------------+--------------------------+
| XbcdXX                    | aXXXXX                   |
+---------------------------+--------------------------+
mysql> SELECT mask_outer('abcdef', 1, 2, '*'), mask_outer('abcdef',0, 5, '#');
+--------------------------------+-------------------------------+
| mask_outer('abcdef', 1, 2, '*') | mask_outer('abcdef',0, 5, '#') |
+--------------------------------+-------------------------------+
| *bcd**                         | a#####                        |
+--------------------------------+-------------------------------+
```

- `mask_pan(str)`

  Masks a payment card Primary Account Number and returns the number with all but the last four digits replaced by `'X'` characters.

  Arguments:

  - *`str`*: The string to mask. The string must be a suitable length for the Primary Account Number, but is not otherwise checked.

  Return value:

  The masked payment number as a string. If the argument is shorter than required, it is returned unchanged.

  Example:

```
mysql> SELECT mask_pan(gen_rnd_pan());
+------------------------+
| mask_pan(gen_rnd_pan()) |
+------------------------+
| XXXXXXXXXXX9102        |
+------------------------+
mysql> SELECT mask_pan(gen_rnd_pan(19));
+---------------------------+
| mask_pan(gen_rnd_pan(19)) |
+---------------------------+
| XXXXXXXXXXXXXX8268       |
+---------------------------+
mysql> SELECT mask_pan('a*Z');
+-----------------+
| mask_pan('a*Z') |
+-----------------+
| a*Z             |
+-----------------+
```

- `mask_pan_relaxed(str)`

  Masks a payment card Primary Account Number and returns the number with all but the first six and last four digits replaced by `'X'` characters. The first six digits indicate the payment card issuer.

Arguments:

- *`str`*: The string to mask. The string must be a suitable length for the Primary Account Number, but is not otherwise checked.

Return value:

The masked payment number as a string. If the argument is shorter than required, it is returned unchanged.

Example:

```
mysql> SELECT mask_pan_relaxed(gen_rnd_pan());
+---------------------------------+
| mask_pan_relaxed(gen_rnd_pan()) |
+---------------------------------+
| 551279XXXXXX3108                |
+---------------------------------+
mysql> SELECT mask_pan_relaxed(gen_rnd_pan(19));
+----------------------------------+
| mask_pan_relaxed(gen_rnd_pan(19)) |
+----------------------------------+
| 462634XXXXXXXXX6739              |
+----------------------------------+
mysql> SELECT mask_pan_relaxed('a*Z');
+-------------------------+
| mask_pan_relaxed('a*Z') |
+-------------------------+
| a*Z                     |
+-------------------------+
```

- `mask_ssn(`*`str`*`)`

  Masks a U.S. Social Security number and returns the number with all but the last four digits replaced by `'X'` characters.

  Arguments:

  - *`str`*: The string to mask. The string must be 11 characters long, but is not otherwise checked.

  Return value:

  The masked Social Security number as a string, or `NULL` if the argument is not the correct length.

  Example:

```
mysql> SELECT mask_ssn('909-63-6922'), mask_ssn('abcdefghijk');
+-------------------------+-------------------------+
| mask_ssn('909-63-6922') | mask_ssn('abcdefghijk') |
+-------------------------+-------------------------+
| XXX-XX-6922             | XXX-XX-hijk             |
+-------------------------+-------------------------+
mysql> SELECT mask_ssn('909');
+-----------------+
| mask_ssn('909') |
+-----------------+
| NULL            |
+-----------------+
```

Random Data Generation Functions

The functions in this section generate random values for different types of data. When possible, generated values have characteristics reserved for demonstration or test values, to avoid having them mistaken for legitimate data. For example, `gen_rnd_us_phone()` returns a U.S. phone number that uses the 555 area code, which is not assigned to phone numbers in actual use. Individual function descriptions describe any exceptions to this principle.

- `gen_range(`*`lower`*`, `*`upper`*`)`

  Generates a random number chosen from a specified range.

  Arguments:

  - *`lower`*: An integer that specifies the lower boundary of the range.
  - *`upper`*: An integer that specifies the upper boundary of the range, which must not be less than the lower boundary.

  Return value:

  A random integer in the range from *`lower`* to *`upper`*, inclusive, or `NULL` if the *`upper`* argument is less than *`lower`*.

  Example:

```
mysql> SELECT gen_range(100, 200), gen_range(-1000, -800);
+--------------------+-----------------------+
| gen_range(100, 200) | gen_range(-1000, -800) |
+--------------------+-----------------------+
|                 177 |                  -917 |
+--------------------+-----------------------+
mysql> SELECT gen_range(1, 0);
+-----------------+
| gen_range(1, 0) |
+-----------------+
|            NULL |
+-----------------+
```

- `gen_rnd_email()`

  Generates a random email address in the `example.com` domain.

  Arguments:

  None.

  Return value:

  A random email address as a string.

  Example:

```
mysql> SELECT gen_rnd_email();
+---------------------------+
| gen_rnd_email()           |
+---------------------------+
| ijocv.mwvhhuf@example.com |
+---------------------------+
```

- `gen_rnd_pan([size])`

  Generates a random payment card Primary Account Number. The number passes the Luhn check (an algorithm that performs a checksum verification against a check digit).

  Warning

  Values returned from `gen_rnd_pan()` should be used only for test purposes, and are not suitable for publication. There is no way to guarantee that a given return value is not assigned to a legitimate payment account. Should it be necessary to publish a `gen_rnd_pan()` result, consider masking it with `mask_pan()` or `mask_pan_relaxed()`.

  Arguments:

  - *`size`*: (Optional) An integer that specifies the size of the result. The default is 16 if *`size`* is not given. If given, *`size`* must be an integer in the range from 12 to 19.

  Return value:

  A random payment number as a string, or `NULL` if a *`size`* argument outside the permitted range is given.

  Example:

```
mysql> SELECT mask_pan(gen_rnd_pan());
+-------------------------+
| mask_pan(gen_rnd_pan()) |
+-------------------------+
| XXXXXXXXXXXX5805        |
+-------------------------+
mysql> SELECT mask_pan(gen_rnd_pan(19));
+---------------------------+
| mask_pan(gen_rnd_pan(19)) |
+---------------------------+
| XXXXXXXXXXXXXXX5067       |
+---------------------------+
mysql> SELECT mask_pan_relaxed(gen_rnd_pan());
+---------------------------------+
| mask_pan_relaxed(gen_rnd_pan()) |
+---------------------------------+
| 398403XXXXXX9547                |
+---------------------------------+
mysql> SELECT mask_pan_relaxed(gen_rnd_pan(19));
+-----------------------------------+
| mask_pan_relaxed(gen_rnd_pan(19)) |
+-----------------------------------+
| 578416XXXXXXXXX6509               |
+-----------------------------------+
mysql> SELECT gen_rnd_pan(11), gen_rnd_pan(20);
+-----------------+-----------------+
| gen_rnd_pan(11) | gen_rnd_pan(20) |
+-----------------+-----------------+
| NULL            | NULL            |
+-----------------+-----------------+
```

- `gen_rnd_ssn()`

  Generates a random U.S. Social Security number in *`AAA-BB-CCCC`* format. The *`AAA`* part is greater than 900 and the *`BB`* part is less than 70, which are characteristics not used for legitimate Social Security numbers.

Arguments:

None.

Return value:

A random Social Security number as a string.

Example:

```
mysql> SELECT gen_rnd_ssn();
+---------------+
| gen_rnd_ssn() |
+---------------+
| 951-26-0058   |
+---------------+
```

- `gen_rnd_us_phone()`

  Generates a random U.S. phone number in `1-555-AAA-BBBB` format. The 555 area code is not used for legitimate phone numbers.

  Arguments:

  None.

  Return value:

  A random U.S. phone number as a string.

  Example:

```
mysql> SELECT gen_rnd_us_phone();
+--------------------+
| gen_rnd_us_phone() |
+--------------------+
| 1-555-682-5423     |
+--------------------+
```

Random Data Dictionary-Based Functions

The functions in this section manipulate dictionaries of terms and perform generation and masking operations based on them. Some of these functions require the SUPER privilege.

When a dictionary is loaded, it becomes part of the dictionary registry and is assigned a name to be used by other dictionary functions. Dictionaries are loaded from plain text files containing one term per line. Empty lines are ignored. To be valid, a dictionary file must contain at least one nonempty line.

- `gen_blacklist(str, dictionary_name, replacement_dictionary_name)`

  Replaces a term present in one dictionary with a term from a second dictionary and returns the replacement term. This masks the original term by substitution.

  Arguments:

  - *`str`*: A string that indicates the term to replace.
  - *`dictionary_name`*: A string that names the dictionary containing the term to replace.

- *replacement_dictionary_name*: A string that names the dictionary from which to choose the replacement term.

Return value:

A string randomly chosen from *replacement_dictionary_name* as a replacement for *str*, or *str* if it does not appear in *dictionary_name*, or NULL if either dictionary name is not in the dictionary registry.

If the term to replace appears in both dictionaries, it is possible for the return value to be the same term.

Example:

```
mysql> SELECT gen_blacklist('Berlin', 'DE_Cities', 'US_Cities');
+-----------------------------------------------+
| gen_blacklist('Berlin', 'DE_Cities', 'US_Cities') |
+-----------------------------------------------+
| Phoenix                                       |
+-----------------------------------------------+
```

- gen_dictionary(*dictionary_name*)

Returns a random term from a dictionary.

Arguments:

- *dictionary_name*: A string that names the dictionary from which to choose the term.

Return value:

A random term from the dictionary as a string, or NULL if the dictionary name is not in the dictionary registry.

Example:

```
mysql> SELECT gen_dictionary('mydict');
+--------------------------+
| gen_dictionary('mydict') |
+--------------------------+
| My term                  |
+--------------------------+
mysql> SELECT gen_dictionary('no-such-dict');
+-------------------------------+
| gen_dictionary('no-such-dict') |
+-------------------------------+
| NULL                          |
+-------------------------------+
```

- gen_dictionary_drop(*dictionary_name*)

Removes a dictionary from the dictionary registry.

This function requires the SUPER privilege.

Arguments:

- *dictionary_name*: A string that names the dictionary to remove from the dictionary registry.

Return value:

A string that indicates whether the drop operation succeeded. `Dictionary removed` indicates success. `Dictionary removal error` indicates failure.

Example:

```
mysql> SELECT gen_dictionary_drop('mydict');
+-----------------------------+
| gen_dictionary_drop('mydict') |
+-----------------------------+
| Dictionary removed            |
+-----------------------------+
mysql> SELECT gen_dictionary_drop('no-such-dict');
+-----------------------------------+
| gen_dictionary_drop('no-such-dict') |
+-----------------------------------+
| Dictionary removal error            |
+-----------------------------------+
```

- `gen_dictionary_load(`*`dictionary_path`*`, `*`dictionary_name`*`)`

  Loads a file into the dictionary registry and assigns the dictionary a name to be used with other functions that require a dictionary name argument.

  This function requires the `SUPER` privilege.

  Important

  Dictionaries are not persistent. Any dictionary used by applications must be loaded for each server startup.

  Once loaded into the registry, a dictionary is used as is, even if the underlying dictionary file changes. To reload a dictionary, first drop it with `gen_dictionary_drop()`, then load it again with `gen_dictionary_load()`.

  Arguments:

  - *`dictionary_path`*: A string that specifies the path name of the dictionary file.
  - *`dictionary_name`*: A string that provides a name for the dictionary.

  Return value:

  A string that indicates whether the load operation succeeded. `Dictionary load success` indicates success. `Dictionary load error` indicates failure. Dictionary load failure can occur for several reasons, including:

  - A dictionary with the given name is already loaded.
  - The dictionary file is not found.
  - The dictionary file contains no terms.
  - The `secure_file_priv` system variable is set and the dictionary file is not located in the directory named by the variable.

  Example:

```
mysql> SELECT gen_dictionary_load('/usr/local/mysql/mysql-files/mydict','mydict');
+----------------------------------------------------------------------+
| gen_dictionary_load('/usr/local/mysql/mysql-files/mydict','mydict') |
+----------------------------------------------------------------------+
| Dictionary load success                                              |
+----------------------------------------------------------------------+
mysql> SELECT gen_dictionary_load('/dev/null','null');
+---------------------------------------+
| gen_dictionary_load('/dev/null','null') |
+---------------------------------------+
| Dictionary load error                 |
+---------------------------------------+
```

6.5 FIPS Support

MySQL supports FIPS mode, if compiled using OpenSSL, and an OpenSSL library and FIPS Object Module are available at runtime.

FIPS mode on the server side applies to cryptographic operations performed by the server. This includes replication (master/slave and Group Replication) and X Plugin, which run within the server. FIPS mode also applies to attempts by clients to connect to the server.

The following sections describe FIPS mode and how to take advantage of it within MySQL:

- FIPS Overview
- System Requirements for FIPS Mode in MySQL
- Configuring FIPS Mode in MySQL

FIPS Overview

Federal Information Processing Standards 140-2 (FIPS 140-2) describes a security standard that can be required by Federal (US Government) agencies for cryptographic modules used to protect sensitive or valuable information. To be considered acceptable for such Federal use, a cryptographic module must be certified for FIPS 140-2. If a system intended to protect sensitive data lacks the proper FIPS 140-2 certificate, Federal agencies cannot purchase it.

Products such as OpenSSL can be used in FIPS mode, although the OpenSSL library itself is not validated for FIPS. Instead, the OpenSSL library is used with the OpenSSL FIPS Object Module to enable OpenSSL-based applications to operate in FIPS mode.

For general information about FIPS and its implementation in OpenSSL, these references may be helpful:

- National Institute of Standards and Technology FIPS PUB 140-2
- OpenSSL FIPS 140-2 Security Policy
- User Guide for the OpenSSL FIPS Object Module v2.0

Important

FIPS mode imposes conditions on cryptographic operations such as restrictions on acceptable encryption algorithms or requirements for longer key lengths. For OpenSSL, the exact FIPS behavior depends on the OpenSSL version. For details, refer to the OpenSSL FIPS User Guide.

System Requirements for FIPS Mode in MySQL

For MySQL to support FIPS mode, these system requirements must be satisfied:

- At build time, MySQL must be compiled using OpenSSL. FIPS mode cannot be used in MySQL if compilation uses a different SSL library.
- At runtime, the OpenSSL library and OpenSSL FIPS Object Module must be available as shared (dynamically linked) objects. It is possible to build statically linked OpenSSL objects, but MySQL will not use them.

FIPS mode has been tested for MySQL on EL7, but may work on other systems.

If your platform or operating system provides the OpenSSL FIPS Object Module, you can use it. Otherwise, you can build the OpenSSL library and FIPS Object Module from source. Use the instructions in the OpenSSL FIPS User Guide (see FIPS Overview).

Configuring FIPS Mode in MySQL

MySQL enables control of FIPS mode on the server side and the client side:

- The `ssl_fips_mode` system variable controls whether the server operates in FIPS mode.
- The `--ssl-fips-mode` client option controls whether a given MySQL client operates in FIPS mode.

The `ssl_fips_mode` system variable and `--ssl-fips-mode` client option permit these values:

- `OFF`: Disable FIPS mode.
- `ON`: Enable FIPS mode.
- `STRICT`: Enable “strict” FIPS mode.

On the server side, numeric `ssl_fips_mode` values of 0, 1, and 2 are equivalent to `OFF`, `ON`, and `STRICT`, respectivey.

Important

In general, `STRICT` imposes more restrictions than `ON`, but MySQL itself has no FIPS-specific code other than to specify to OpenSSL the FIPS mode value. The exact behavior of FIPS mode for `ON` or `STRICT` depends on the OpenSSL version. For details, refer to the OpenSSL FIPS User Guide (see FIPS Overview).

Note

If the OpenSSL FIPS Object Module is not available, the only permitted value for `ssl_fips_mode` and `--ssl-fips-mode` is `OFF`. An error occurs for attempts to set the FIPS mode to a different value.

FIPS mode on the server side applies to cryptographic operations performed by the server. This includes replication (master/slave and Group Replication) and X Plugin, which run within the server.

FIPS mode also applies to attempts by clients to connect to the server. When enabled, on either the client or server side, it restricts which of the supported encryption ciphers can be chosen. However, enabling FIPS mode does not require that an encrypted connection must be used, or that user credentials must be encrypted. For example, if FIPS mode is enabled, stronger cryptographic algorithms are required. In particular, MD5 is restricted, so trying to establish an encrypted connection using an encryption cipher

such as `RC4-MD5` does not work. But there is nothing about FIPS mode that prevents establishing an unencrypted connection. (To do that, you can use the `REQUIRE` clause for `CREATE USER` or `ALTER USER` for specific user accounts, or set the `require_secure_transport` system variable to affect all accounts.)

Chapter 7 Backup and Recovery

Table of Contents

7.1 Backup and Recovery Types 1420
7.2 Database Backup Methods 1423
7.3 Example Backup and Recovery Strategy 1425
7.3.1 Establishing a Backup Policy 1426
7.3.2 Using Backups for Recovery 1428
7.3.3 Backup Strategy Summary 1428
7.4 Using mysqldump for Backups 1429
7.4.1 Dumping Data in SQL Format with mysqldump 1429
7.4.2 Reloading SQL-Format Backups 1430
7.4.3 Dumping Data in Delimited-Text Format with mysqldump 1431
7.4.4 Reloading Delimited-Text Format Backups 1432
7.4.5 mysqldump Tips 1433
7.5 Point-in-Time (Incremental) Recovery Using the Binary Log 1435
7.5.1 Point-in-Time Recovery Using Event Times 1437
7.5.2 Point-in-Time Recovery Using Event Positions 1437
7.6 MyISAM Table Maintenance and Crash Recovery 1438
7.6.1 Using myisamchk for Crash Recovery 1439
7.6.2 How to Check MyISAM Tables for Errors 1439
7.6.3 How to Repair MyISAM Tables 1440
7.6.4 MyISAM Table Optimization 1442
7.6.5 Setting Up a MyISAM Table Maintenance Schedule 1443

It is important to back up your databases so that you can recover your data and be up and running again in case problems occur, such as system crashes, hardware failures, or users deleting data by mistake. Backups are also essential as a safeguard before upgrading a MySQL installation, and they can be used to transfer a MySQL installation to another system or to set up replication slave servers.

MySQL offers a variety of backup strategies from which you can choose the methods that best suit the requirements for your installation. This chapter discusses several backup and recovery topics with which you should be familiar:

- Types of backups: Logical versus physical, full versus incremental, and so forth.
- Methods for creating backups.
- Recovery methods, including point-in-time recovery.
- Backup scheduling, compression, and encryption.
- Table maintenance, to enable recovery of corrupt tables.

Additional Resources

Resources related to backup or to maintaining data availability include the following:

- Customers of MySQL Enterprise Edition can use the MySQL Enterprise Backup product for backups. For an overview of the MySQL Enterprise Backup product, see Section 30.2, “MySQL Enterprise Backup Overview”.

- A forum dedicated to backup issues is available at https://forums.mysql.com/list.php?28.
- Details for `mysqldump` can be found in Chapter 4, *MySQL Programs*.
- The syntax of the SQL statements described here is given in Chapter 13, *SQL Statement Syntax*.
- For additional information about `InnoDB` backup procedures, see Section 15.17.1, “InnoDB Backup”.
- Replication enables you to maintain identical data on multiple servers. This has several benefits, such as enabling client query load to be distributed over servers, availability of data even if a given server is taken offline or fails, and the ability to make backups with no impact on the master by using a slave server. See Chapter 17, *Replication*.
- MySQL InnoDB cluster is a collection of products that work together to provide a high availability solution. A group of MySQL servers can be configured to create a cluster using MySQL Shell. The cluster of servers has a single master, called the primary, which acts as the read-write master. Multiple secondary servers are replicas of the master. A minimum of three servers are required to create a high availability cluster. A client application is connected to the primary via MySQL Router. If the primary fails, a secondary is automatically promoted to the role of primary, and MySQL Router routes requests to the new primary.
- NDB Cluster provides a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. See Chapter 22, *MySQL NDB Cluster 8.0*, which provides information about MySQL NDB Cluster 8.0.

7.1 Backup and Recovery Types

This section describes the characteristics of different types of backups.

Physical (Raw) Versus Logical Backups

Physical backups consist of raw copies of the directories and files that store database contents. This type of backup is suitable for large, important databases that need to be recovered quickly when problems occur.

Logical backups save information represented as logical database structure (`CREATE DATABASE`, `CREATE TABLE` statements) and content (`INSERT` statements or delimited-text files). This type of backup is suitable for smaller amounts of data where you might edit the data values or table structure, or recreate the data on a different machine architecture.

Physical backup methods have these characteristics:

- The backup consists of exact copies of database directories and files. Typically this is a copy of all or part of the MySQL data directory.
- Physical backup methods are faster than logical because they involve only file copying without conversion.
- Output is more compact than for logical backup.
- Because backup speed and compactness are important for busy, important databases, the MySQL Enterprise Backup product performs physical backups. For an overview of the MySQL Enterprise Backup product, see Section 30.2, “MySQL Enterprise Backup Overview”.
- Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files. This may or may not provide for table-level granularity, depending on storage engine. For

example, `InnoDB` tables can each be in a separate file, or share file storage with other `InnoDB` tables; each `MyISAM` table corresponds uniquely to a set of files.

- In addition to databases, the backup can include any related files such as log or configuration files.
- Data from `MEMORY` tables is tricky to back up this way because their contents are not stored on disk. (The MySQL Enterprise Backup product has a feature where you can retrieve data from `MEMORY` tables during a backup.)
- Backups are portable only to other machines that have identical or similar hardware characteristics.
- Backups can be performed while the MySQL server is not running. If the server is running, it is necessary to perform appropriate locking so that the server does not change database contents during the backup. MySQL Enterprise Backup does this locking automatically for tables that require it.
- Physical backup tools include the `mysqlbackup` of MySQL Enterprise Backup for `InnoDB` or any other tables, or file system-level commands (such as `cp`, `scp`, `tar`, `rsync`) for `MyISAM` tables.
- For restore:
  - MySQL Enterprise Backup restores `InnoDB` and other tables that it backed up.
  - `ndb_restore` restores `NDB` tables.
  - Files copied at the file system level can be copied back to their original locations with file system commands.

Logical backup methods have these characteristics:

- The backup is done by querying the MySQL server to obtain database structure and content information.
- Backup is slower than physical methods because the server must access database information and convert it to logical format. If the output is written on the client side, the server must also send it to the backup program.
- Output is larger than for physical backup, particularly when saved in text format.
- Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This is true regardless of storage engine.
- The backup does not include log or configuration files, or other database-related files that are not part of databases.
- Backups stored in logical format are machine independent and highly portable.
- Logical backups are performed with the MySQL server running. The server is not taken offline.
- Logical backup tools include the `mysqldump` program and the `SELECT ... INTO OUTFILE` statement. These work for any storage engine, even `MEMORY`.
- To restore logical backups, SQL-format dump files can be processed using the `mysql` client. To load delimited-text files, use the `LOAD DATA` statement or the `mysqlimport` client.

Online Versus Offline Backups

Online backups take place while the MySQL server is running so that the database information can be obtained from the server. Offline backups take place while the server is stopped. This distinction can also

be described as “hot” versus “cold” backups; a “warm” backup is one where the server remains running but locked against modifying data while you access database files externally.

Online backup methods have these characteristics:

- The backup is less intrusive to other clients, which can connect to the MySQL server during the backup and may be able to access data depending on what operations they need to perform.
- Care must be taken to impose appropriate locking so that data modifications do not take place that would compromise backup integrity. The MySQL Enterprise Backup product does such locking automatically.

Offline backup methods have these characteristics:

- Clients can be affected adversely because the server is unavailable during backup. For that reason, such backups are often taken from a replication slave server that can be taken offline without harming availability.
- The backup procedure is simpler because there is no possibility of interference from client activity.

A similar distinction between online and offline applies for recovery operations, and similar characteristics apply. However, it is more likely that clients will be affected for online recovery than for online backup because recovery requires stronger locking. During backup, clients might be able to read data while it is being backed up. Recovery modifies data and does not just read it, so clients must be prevented from accessing data while it is being restored.

Local Versus Remote Backups

A local backup is performed on the same host where the MySQL server runs, whereas a remote backup is done from a different host. For some types of backups, the backup can be initiated from a remote host even if the output is written locally on the server. host.

- `mysqldump` can connect to local or remote servers. For SQL output (`CREATE` and `INSERT` statements), local or remote dumps can be done and generate output on the client. For delimited-text output (with the `--tab` option), data files are created on the server host.
- `SELECT ... INTO OUTFILE` can be initiated from a local or remote client host, but the output file is created on the server host.
- Physical backup methods typically are initiated locally on the MySQL server host so that the server can be taken offline, although the destination for copied files might be remote.

Snapshot Backups

Some file system implementations enable “snapshots” to be taken. These provide logical copies of the file system at a given point in time, without requiring a physical copy of the entire file system. (For example, the implementation may use copy-on-write techniques so that only parts of the file system modified after the snapshot time need be copied.) MySQL itself does not provide the capability for taking file system snapshots. It is available through third-party solutions such as Veritas, LVM, or ZFS.

Full Versus Incremental Backups

A full backup includes all data managed by a MySQL server at a given point in time. An incremental backup consists of the changes made to the data during a given time span (from one point in time to another). MySQL has different ways to perform full backups, such as those described earlier in this section.

Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes.

Full Versus Point-in-Time (Incremental) Recovery

A full recovery restores all data from a full backup. This restores the server instance to the state that it had when the backup was made. If that state is not sufficiently current, a full recovery can be followed by recovery of incremental backups made since the full backup, to bring the server to a more up-to-date state.

Incremental recovery is recovery of changes made during a given time span. This is also called point-in-time recovery because it makes a server's state current up to a given time. Point-in-time recovery is based on the binary log and typically follows a full recovery from the backup files that restores the server to its state when the backup was made. Then the data changes written in the binary log files are applied as incremental recovery to redo data modifications and bring the server up to the desired point in time.

Table Maintenance

Data integrity can be compromised if tables become corrupt. For `InnoDB` tables, this is not a typical issue. For programs to check `MyISAM` tables and repair them if problems are found, see Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

Backup Scheduling, Compression, and Encryption

Backup scheduling is valuable for automating backup procedures. Compression of backup output reduces space requirements, and encryption of the output provides better security against unauthorized access of backed-up data. MySQL itself does not provide these capabilities. The MySQL Enterprise Backup product can compress `InnoDB` backups, and compression or encryption of backup output can be achieved using file system utilities. Other third-party solutions may be available.

7.2 Database Backup Methods

This section summarizes some general methods for making backups.

Making a Hot Backup with MySQL Enterprise Backup

Customers of MySQL Enterprise Edition can use the MySQL Enterprise Backup product to do physical backups of entire instances or selected databases, tables, or both. This product includes features for incremental and compressed backups. Backing up the physical database files makes restore much faster than logical techniques such as the `mysqldump` command. `InnoDB` tables are copied using a hot backup mechanism. (Ideally, the `InnoDB` tables should represent a substantial majority of the data.) Tables from other storage engines are copied using a warm backup mechanism. For an overview of the MySQL Enterprise Backup product, see Section 30.2, “MySQL Enterprise Backup Overview”.

Making Backups with mysqldump

The `mysqldump` program can make backups. It can back up all kinds of tables. (See Section 7.4, “Using mysqldump for Backups”.)

For `InnoDB` tables, it is possible to perform an online backup that takes no locks on tables using the `--single-transaction` option to `mysqldump`. See Section 7.3.1, “Establishing a Backup Policy”.

Making Backups by Copying Table Files

MyISAM tables can be backed up by copying table files (`*.MYD`, `*.MYI` files, and associated `*.sdi` files). To get a consistent backup, stop the server or lock and flush the relevant tables:

```
FLUSH TABLES tbl_list WITH READ LOCK;
```

You need only a read lock; this enables other clients to continue to query the tables while you are making a copy of the files in the database directory. The flush is needed to ensure that the all active index pages are written to disk before you start the backup. See Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”, and Section 13.7.8.3, “FLUSH Syntax”.

You can also create a binary backup simply by copying the table files, as long as the server isn't updating anything. (But note that table file copying methods do not work if your database contains `InnoDB` tables. Also, even if the server is not actively updating data, `InnoDB` may still have modified data cached in memory and not flushed to disk.)

For an example of this backup method, refer to the export and import example in Section 13.2.5, “IMPORT TABLE Syntax”.

Making Delimited-Text File Backups

To create a text file containing a table's data, you can use `SELECT * INTO OUTFILE 'file_name' FROM tbl_name`. The file is created on the MySQL server host, not the client host. For this statement, the output file cannot already exist because permitting files to be overwritten constitutes a security risk. See Section 13.2.10, “SELECT Syntax”. This method works for any kind of data file, but saves only table data, not the table structure.

Another way to create text data files (along with files containing `CREATE TABLE` statements for the backed up tables) is to use `mysqldump` with the `--tab` option. See Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”.

To reload a delimited-text data file, use `LOAD DATA` or `mysqlimport`.

Making Incremental Backups by Enabling the Binary Log

MySQL supports incremental backups: You must start the server with the `--log-bin` option to enable binary logging; see Section 5.4.4, “The Binary Log”. The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you performed a backup. At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using `FLUSH LOGS`. This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained in Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”. The next time you do a full backup, you should also rotate the binary log using `FLUSH LOGS` or `mysqldump --flush-logs`. See Section 4.5.4, “`mysqldump` — A Database Backup Program”.

Making Backups Using Replication Slaves

If you have performance problems with your master server while making backups, one strategy that can help is to set up replication and perform backups on the slave rather than on the master. See Section 17.4.1, “Using Replication for Backups”.

If you are backing up a slave replication server, you should back up its master info and relay log info repositories (see Section 17.2.4, “Replication Relay and Status Logs”) when you back up the slave's databases, regardless of the backup method you choose. This information is always needed to resume replication after you restore the slave's data. If your slave is replicating `LOAD DATA` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The

slave needs these files to resume replication of any interrupted `LOAD DATA` operations. The location of this directory is the value of the `--slave-load-tmpdir` option. If the server was not started with that option, the directory location is the value of the `tmpdir` system variable.

Recovering Corrupt Tables

If you have to restore `MyISAM` tables that have become corrupt, try to recover them using `REPAIR TABLE` or `myisamchk -r` first. That should work in 99.9% of all cases. If `myisamchk` fails, see Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

Making Backups Using a File System Snapshot

If you are using a Veritas file system, you can make a backup like this:

1. From a client program, execute `FLUSH TABLES WITH READ LOCK`.
2. From another shell, execute `mount vxfs snapshot`.
3. From the first client, execute `UNLOCK TABLES`.
4. Copy files from the snapshot.
5. Unmount the snapshot.

Similar snapshot capabilities may be available in other file systems, such as LVM or ZFS.

7.3 Example Backup and Recovery Strategy

This section discusses a procedure for performing backups that enables you to recover data after several types of crashes:

- Operating system crash
- Power failure
- File system crash
- Hardware problem (hard drive, motherboard, and so forth)

The example commands do not include options such as `--user` and `--password` for the `mysqldump` and `mysql` client programs. You should include such options as necessary to enable client programs to connect to the MySQL server.

Assume that data is stored in the `InnoDB` storage engine, which has support for transactions and automatic crash recovery. Assume also that the MySQL server is under load at the time of the crash. If it were not, no recovery would ever be needed.

For cases of operating system crashes or power failures, we can assume that MySQL's disk data is available after a restart. The `InnoDB` data files might not contain consistent data due to the crash, but `InnoDB` reads its logs and finds in them the list of pending committed and noncommitted transactions that have not been flushed to the data files. `InnoDB` automatically rolls back those transactions that were not committed, and flushes to its data files those that were committed. Information about this recovery process is conveyed to the user through the MySQL error log. The following is an example log excerpt:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
```

```
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

For the cases of file system crashes or hardware problems, we can assume that the MySQL disk data is *not* available after a restart. This means that MySQL fails to start successfully because some blocks of disk data are no longer readable. In this case, it is necessary to reformat the disk, install a new one, or otherwise correct the underlying problem. Then it is necessary to recover our MySQL data from backups, which means that backups must already have been made. To make sure that is the case, design and implement a backup policy.

7.3.1 Establishing a Backup Policy

To be useful, backups must be scheduled regularly. A full backup (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, MySQL Enterprise Backup can perform a physical backup of an entire instance, with optimizations to minimize overhead and avoid disruption when backing up `InnoDB` data files; `mysqldump` provides online logical backup. This discussion uses `mysqldump`.

Assume that we make a full backup of all our `InnoDB` tables in all databases using the following command on Sunday at 1 p.m., when load is low:

```
shell> mysqldump --all-databases --master-data --single-transaction > backup_sunday_1_PM.sql
```

The resulting `.sql` file produced by `mysqldump` contains a set of SQL `INSERT` statements that can be used to reload the dumped tables at a later time.

This backup operation acquires a global read lock on all tables at the beginning of the dump (using `FLUSH TABLES WITH READ LOCK`). As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the `FLUSH` statement is issued, the backup operation may stall until those statements finish. After that, the dump becomes lock-free and does not disturb reads and writes on the tables.

It was assumed earlier that the tables to back up are `InnoDB` tables, so `--single-transaction` uses a consistent read and guarantees that data seen by `mysqldump` does not change. (Changes made by other clients to `InnoDB` tables are not seen by the `mysqldump` process.) If the backup operation includes nontransactional tables, consistency requires that they do not change during the backup. For example, for the `MyISAM` tables in the `mysql` database, there must be no administrative changes to MySQL accounts during the backup.

Full backups are necessary, but it is not always convenient to create them. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that has not changed since the previous full backup. It is more efficient to make an

initial full backup, and then to make incremental backups. The incremental backups are smaller and take less time to produce. The tradeoff is that, at recovery time, you cannot restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.

To make incremental backups, we need to save the incremental changes. In MySQL, these changes are represented in the binary log, so the MySQL server should always be started with the `--log-bin` option to enable that log. With binary logging enabled, the server writes each data change into a file while it updates data. Looking at the data directory of a MySQL server that was started with the `--log-bin` option and that has been running for some days, we find these MySQL binary log files:

```
-rw-rw---- 1 guilhem  guilhem   1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem  guilhem         4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem  guilhem        79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem  guilhem       508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem  guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem  guilhem    998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem  guilhem       361 Nov 14 10:07 gbichot2-bin.index
```

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is running, you can also tell it to close the current binary log file and begin a new one manually by issuing a `FLUSH LOGS` SQL statement or with a `mysqladmin flush-logs` command. `mysqldump` also has an option to flush the logs. The `.index` file in the data directory contains the list of all MySQL binary logs in the directory.

The MySQL binary logs are important for recovery because they form the set of incremental backups. If you make sure to flush the logs when you make your full backup, the binary log files created afterward contain all the data changes made since the backup. Let's modify the previous `mysqldump` command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so that the dump file contains the name of the new current binary log:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
         --all-databases > backup_sunday_1_PM.sql
```

After executing this command, the data directory contains a new binary log file, `gbichot2-bin.000007`, because the `--flush-logs` option causes the server to flush its logs. The `--master-data` option causes `mysqldump` to write binary log information to its output, so the resulting `.sql` dump file includes these lines:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG_POS=4;
```

Because the `mysqldump` command made a full backup, those lines mean two things:

- The dump file contains all changes made before any changes written to the `gbichot2-bin.000007` binary log file or higher.
- All data changes logged after the backup are not present in the dump file, but are present in the `gbichot2-bin.000007` binary log file or higher.

On Monday at 1 p.m., we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a `mysqladmin flush-logs` command creates `gbichot2-bin.000008`. All changes between the Sunday 1 p.m. full backup and Monday 1 p.m. will be in the `gbichot2-bin.000007` file. This incremental backup is important, so it is a good idea to copy it to a safe place. (For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday at 1 p.m., execute another `mysqladmin flush-logs` command. All changes between Monday 1 p.m. and Tuesday 1 p.m. will be in the `gbichot2-bin.000008` file (which also should be copied somewhere safe).

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
        --all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

Note

Deleting the MySQL binary logs with `mysqldump --delete-master-logs` can be dangerous if your server is a replication master server, because slave servers might not yet fully have processed the contents of the binary log. The description for the `PURGE BINARY LOGS` statement explains what should be verified before deleting the MySQL binary logs. See Section 13.4.1.1, "PURGE BINARY LOGS Syntax".

7.3.2 Using Backups for Recovery

Now, suppose that we have a catastrophic crash on Wednesday at 8 a.m. that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 p.m.). The full backup file is just a set of SQL statements, so restoring it is very easy:

```
shell> mysql < backup_sunday_1_PM.sql
```

At this point, the data is restored to its state as of Sunday 1 p.m.. To restore the changes made since then, we must use the incremental backups; that is, the `gbichot2-bin.000007` and `gbichot2-bin.000008` binary log files. Fetch the files if necessary from where they were backed up, and then process their contents like this:

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

We now have recovered the data to its state as of Tuesday 1 p.m., but still are missing the changes from that date to the date of the crash. To not lose them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID disks, SAN, ...) different from the place where it stores its data files, so that these logs were not on the destroyed disk. (That is, we can start the server with a `--log-bin` option that specifies a location on a different physical device from the one on which the data directory resides. That way, the logs are safe even if the device containing the directory is lost.) If we had done this, we would have the `gbichot2-bin.000009` file (and any subsequent files) at hand, and we could apply them using `mysqlbinlog` and `mysql` to restore the most recent data changes with no loss up to the moment of the crash:

```
shell> mysqlbinlog gbichot2-bin.000009 ... | mysql
```

For more information about using `mysqlbinlog` to process binary log files, see Section 7.5, "Point-in-Time (Incremental) Recovery Using the Binary Log".

7.3.3 Backup Strategy Summary

In case of an operating system crash or power failure, `InnoDB` itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

- Always run the MySQL server with the `--log-bin` option, or even `--log-bin=`*`log_name`*, where the log file name is located on some safe media different from the drive on which the data directory is located. If you have such safe media, this technique can also be good for disk load balancing (which results in a performance improvement).

- Make periodic full backups, using the `mysqldump` command shown earlier in Section 7.3.1, “Establishing a Backup Policy”, that makes an online, nonblocking backup.
- Make periodic incremental backups by flushing the logs with `FLUSH LOGS` or `mysqladmin flush-logs`.

7.4 Using mysqldump for Backups

This section describes how to use `mysqldump` to produce dump files, and how to reload dump files. A dump file can be used in several ways:

- As a backup to enable data recovery in case of data loss.
- As a source of data for setting up replication slaves.
- As a source of data for experimentation:
  - To make a copy of a database that you can use without changing the original data.
  - To test potential upgrade incompatibilities.

`mysqldump` produces two types of output, depending on whether the `--tab` option is given:

- Without `--tab`, `mysqldump` writes SQL statements to the standard output. This output consists of `CREATE` statements to create dumped objects (databases, tables, stored routines, and so forth), and `INSERT` statements to load data into tables. The output can be saved in a file and reloaded later using `mysql` to recreate the dumped objects. Options are available to modify the format of the SQL statements, and to control which objects are dumped.
- With `--tab`, `mysqldump` produces two output files for each dumped table. The server writes one file as tab-delimited text, one line per table row. This file is named *`tbl_name`*`.txt` in the output directory. The server also sends a `CREATE TABLE` statement for the table to `mysqldump`, which writes it as a file named *`tbl_name`*`.sql` in the output directory.

7.4.1 Dumping Data in SQL Format with mysqldump

This section describes how to use `mysqldump` to create SQL-format dump files. For information about reloading such dump files, see Section 7.4.2, “Reloading SQL-Format Backups”.

By default, `mysqldump` writes information as SQL statements to the standard output. You can save the output in a file:

```
shell> mysqldump [arguments] > file_name
```

To dump all databases, invoke `mysqldump` with the `--all-databases` option:

```
shell> mysqldump --all-databases > dump.sql
```

To dump only specific databases, name them on the command line and use the `--databases` option:

```
shell> mysqldump --databases db1 db2 db3 > dump.sql
```

The `--databases` option causes all names on the command line to be treated as database names. Without this option, `mysqldump` treats the first name as a database name and those following as table names.

With `--all-databases` or `--databases`, `mysqldump` writes `CREATE DATABASE` and `USE` statements prior to the dump output for each database. This ensures that when the dump file is reloaded, it creates each database if it does not exist and makes it the default database so database contents are loaded into the same database from which they came. If you want to cause the dump file to force a drop of each database before recreating it, use the `--add-drop-database` option as well. In this case, `mysqldump` writes a `DROP DATABASE` statement preceding each `CREATE DATABASE` statement.

To dump a single database, name it on the command line:

```
shell> mysqldump --databases test > dump.sql
```

In the single-database case, it is permissible to omit the `--databases` option:

```
shell> mysqldump test > dump.sql
```

The difference between the two preceding commands is that without `--databases`, the dump output contains no `CREATE DATABASE` or `USE` statements. This has several implications:

- When you reload the dump file, you must specify a default database name so that the server knows which database to reload.
- For reloading, you can specify a database name different from the original name, which enables you to reload the data into a different database.
- If the database to be reloaded does not exist, you must create it first.
- Because the output will contain no `CREATE DATABASE` statement, the `--add-drop-database` option has no effect. If you use it, it produces no `DROP DATABASE` statement.

To dump only specific tables from a database, name them on the command line following the database name:

```
shell> mysqldump test t1 t3 t7 > dump.sql
```

By default, if GTIDs are in use on the server where you create the dump file (`gtid_mode=ON`), `mysqldump` includes a `SET @@GLOBAL.gtid_purged` statement in the output to add the GTIDs from the `gtid_executed` set on the source server to the `gtid_purged` set on the target server. If you are dumping only specific databases or tables, it is important to note that the value that is included by `mysqldump` includes the GTIDs of all transactions in the `gtid_executed` set on the source server, even those that changed suppressed parts of the database, or other databases on the server that were not included in the partial dump. If you only replay one partial dump file on the target server, the extra GTIDs do not cause any problems with the future operation of that server. However, if you replay a second dump file on the target server that contains the same GTIDs (for example, another partial dump from the same source server), any `SET @@GLOBAL.gtid_purged` statement in the second dump file fails. To avoid this issue, either set the `mysqldump` option `--set-gtid-purged` to `OFF` or `COMMENTED` to output the second dump file without an active `SET @@GLOBAL.gtid_purged` statement, or remove the statement manually before replaying the dump file.

7.4.2 Reloading SQL-Format Backups

To reload a dump file written by `mysqldump` that consists of SQL statements, use it as input to the `mysql` client. If the dump file was created by `mysqldump` with the `--all-databases` or `--databases` option, it contains `CREATE DATABASE` and `USE` statements and it is not necessary to specify a default database into which to load the data:

```
shell> mysql < dump.sql
```

Alternatively, from within `mysql`, use a `source` command:

```
mysql> source dump.sql
```

If the file is a single-database dump not containing `CREATE DATABASE` and `USE` statements, create the database first (if necessary):

```
shell> mysqladmin create db1
```

Then specify the database name when you load the dump file:

```
shell> mysql db1 < dump.sql
```

Alternatively, from within `mysql`, create the database, select it as the default database, and load the dump file:

```
mysql> CREATE DATABASE IF NOT EXISTS db1;
mysql> USE db1;
mysql> source dump.sql
```

Note

For Windows PowerShell users: Because the "<" character is reserved for future use in PowerShell, an alternative approach is required, such as using quotes `cmd.exe /c "mysql < dump.sql"`.

7.4.3 Dumping Data in Delimited-Text Format with mysqldump

This section describes how to use `mysqldump` to create delimited-text dump files. For information about reloading such dump files, see Section 7.4.4, “Reloading Delimited-Text Format Backups”.

If you invoke `mysqldump` with the `--tab=dir_name` option, it uses *`dir_name`* as the output directory and dumps tables individually in that directory using two files for each table. The table name is the base name for these files. For a table named `t1`, the files are named `t1.sql` and `t1.txt`. The `.sql` file contains a `CREATE TABLE` statement for the table. The `.txt` file contains the table data, one line per table row.

The following command dumps the contents of the `db1` database to files in the `/tmp` database:

```
shell> mysqldump --tab=/tmp db1
```

The `.txt` files containing table data are written by the server, so they are owned by the system account used for running the server. The server uses `SELECT ... INTO OUTFILE` to write the files, so you must have the `FILE` privilege to perform this operation, and an error occurs if a given `.txt` file already exists.

The server sends the `CREATE` definitions for dumped tables to `mysqldump`, which writes them to `.sql` files. These files therefore are owned by the user who executes `mysqldump`.

It is best that `--tab` be used only for dumping a local server. If you use it with a remote server, the `--tab` directory must exist on both the local and remote hosts, and the `.txt` files will be written by the server in the remote directory (on the server host), whereas the `.sql` files will be written by `mysqldump` in the local directory (on the client host).

For `mysqldump --tab`, the server by default writes table data to `.txt` files one line per row with tabs between column values, no quotation marks around column values, and newline as the line terminator. (These are the same defaults as for `SELECT ... INTO OUTFILE`.)

To enable data files to be written using a different format, `mysqldump` supports these options:

- `--fields-terminated-by=`*`str`*

  The string for separating column values (default: tab).

- `--fields-enclosed-by=`*`char`*

  The character within which to enclose column values (default: no character).

- `--fields-optionally-enclosed-by=`*`char`*

  The character within which to enclose non-numeric column values (default: no character).

- `--fields-escaped-by=`*`char`*

  The character for escaping special characters (default: no escaping).

- `--lines-terminated-by=`*`str`*

  The line-termination string (default: newline).

Depending on the value you specify for any of these options, it might be necessary on the command line to quote or escape the value appropriately for your command interpreter. Alternatively, specify the value using hex notation. Suppose that you want `mysqldump` to quote column values within double quotation marks. To do so, specify double quote as the value for the `--fields-enclosed-by` option. But this character is often special to command interpreters and must be treated specially. For example, on Unix, you can quote the double quote like this:

```
--fields-enclosed-by='"'
```

On any platform, you can specify the value in hex:

```
--fields-enclosed-by=0x22
```

It is common to use several of the data-formatting options together. For example, to dump tables in comma-separated values format with lines terminated by carriage-return/newline pairs (`\r\n`), use this command (enter it on a single line):

```
shell> mysqldump --tab=/tmp --fields-terminated-by=,
         --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1
```

Should you use any of the data-formatting options to dump table data, you will need to specify the same format when you reload data files later, to ensure proper interpretation of the file contents.

7.4.4 Reloading Delimited-Text Format Backups

For backups produced with `mysqldump --tab`, each table is represented in the output directory by an `.sql` file containing the `CREATE TABLE` statement for the table, and a `.txt` file containing the table data. To reload a table, first change location into the output directory. Then process the `.sql` file with `mysql` to create an empty table and process the `.txt` file to load the data into the table:

```
shell> mysql db1 < t1.sql
shell> mysqlimport db1 t1.txt
```

An alternative to using `mysqlimport` to load the data file is to use the `LOAD DATA` statement from within the `mysql` client:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1;
```

If you used any data-formatting options with `mysqldump` when you initially dumped the table, you must use the same options with `mysqlimport` or `LOAD DATA` to ensure proper interpretation of the data file contents:

```
shell> mysqlimport --fields-terminated-by=,
         --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1 t1.txt
```

Or:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1
       FIELDS TERMINATED BY ',' FIELDS ENCLOSED BY '"'
       LINES TERMINATED BY '\r\n';
```

7.4.5 mysqldump Tips

This section surveys techniques that enable you to use `mysqldump` to solve specific problems:

- How to make a copy a database
- How to copy a database from one server to another
- How to dump stored programs (stored procedures and functions, triggers, and events)
- How to dump definitions and data separately

7.4.5.1 Making a Copy of a Database

```
shell> mysqldump db1 > dump.sql
shell> mysqladmin create db2
shell> mysql db2 < dump.sql
```

Do not use `--databases` on the `mysqldump` command line because that causes `USE db1` to be included in the dump file, which overrides the effect of naming `db2` on the `mysql` command line.

7.4.5.2 Copy a Database from one Server to Another

On Server 1:

```
shell> mysqldump --databases db1 > dump.sql
```

Copy the dump file from Server 1 to Server 2.

On Server 2:

```
shell> mysql < dump.sql
```

Use of `--databases` with the `mysqldump` command line causes the dump file to include `CREATE DATABASE` and `USE` statements that create the database if it does exist and make it the default database for the reloaded data.

Alternatively, you can omit `--databases` from the `mysqldump` command. Then you will need to create the database on Server 2 (if necessary) and specify it as the default database when you reload the dump file.

On Server 1:

```
shell> mysqldump db1 > dump.sql
```

On Server 2:

```
shell> mysqladmin create db1
shell> mysql db1 < dump.sql
```

You can specify a different database name in this case, so omitting `--databases` from the `mysqldump` command enables you to dump data from one database and load it into another.

7.4.5.3 Dumping Stored Programs

Several options control how `mysqldump` handles stored programs (stored procedures and functions, triggers, and events):

- `--events`: Dump Event Scheduler events
- `--routines`: Dump stored procedures and functions
- `--triggers`: Dump triggers for tables

The `--triggers` option is enabled by default so that when tables are dumped, they are accompanied by any triggers they have. The other options are disabled by default and must be specified explicitly to dump the corresponding objects. To disable any of these options explicitly, use its skip form: `--skip-events`, `--skip-routines`, or `--skip-triggers`.

7.4.5.4 Dumping Table Definitions and Content Separately

The `--no-data` option tells `mysqldump` not to dump table data, resulting in the dump file containing only statements to create the tables. Conversely, the `--no-create-info` option tells `mysqldump` to suppress `CREATE` statements from the output, so that the dump file contains only table data.

For example, to dump table definitions and data separately for the `test` database, use these commands:

```
shell> mysqldump --no-data test > dump-defs.sql
shell> mysqldump --no-create-info test > dump-data.sql
```

For a definition-only dump, add the `--routines` and `--events` options to also include stored routine and event definitions:

```
shell> mysqldump --no-data --routines --events test > dump-defs.sql
```

7.4.5.5 Using mysqldump to Test for Upgrade Incompatibilities

When contemplating a MySQL upgrade, it is prudent to install the newer version separately from your current production version. Then you can dump the database and database object definitions from the

production server and load them into the new server to verify that they are handled properly. (This is also useful for testing downgrades.)

On the production server:

```
shell> mysqldump --all-databases --no-data --routines --events > dump-defs.sql
```

On the upgraded server:

```
shell> mysql < dump-defs.sql
```

Because the dump file does not contain table data, it can be processed quickly. This enables you to spot potential incompatibilities without waiting for lengthy data-loading operations. Look for warnings or errors while the dump file is being processed.

After you have verified that the definitions are handled properly, dump the data and try to load it into the upgraded server.

On the production server:

```
shell> mysqldump --all-databases --no-create-info > dump-data.sql
```

On the upgraded server:

```
shell> mysql < dump-data.sql
```

Now check the table contents and run some test queries.

7.5 Point-in-Time (Incremental) Recovery Using the Binary Log

Point-in-time recovery refers to recovery of data changes made since a given point in time. Typically, this type of recovery is performed after restoring a full backup that brings the server to its state as of the time the backup was made. (The full backup can be made in several ways, such as those listed in Section 7.2, “Database Backup Methods”.) Point-in-time recovery then brings the server up to date incrementally from the time of the full backup to a more recent time.

Note

Many of the examples here use the `mysql` client to process binary log output produced by `mysqlbinlog`. If your binary log contains `\0` (null) characters, that output cannot be parsed by `mysql` unless you invoke it with the `--binary-mode` option.

The source of information for point-in-time recovery is the set of incremental backups represented by the binary log files generated subsequent to the full backup operation. Therefore, the server must be started with the `--log-bin` option to enable binary logging (see Section 5.4.4, “The Binary Log”).

To restore data from the binary log, you must know the name and location of the current binary log files. By default, the server creates binary log files in the data directory, but a path name can be specified with the `--log-bin` option to place the files in a different location. Section 5.4.4, “The Binary Log”.

To see a listing of all binary log files, use this statement:

```
mysql> SHOW BINARY LOGS;
```

To determine the name of the current binary log file, issue the following statement:

```
mysql> SHOW MASTER STATUS;
```

The `mysqlbinlog` utility converts the events in the binary log files from binary format to text so that they can be executed or viewed. `mysqlbinlog` has options for selecting sections of the binary log based on event times or position of events within the log. See Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

Executing events from the binary log causes the data modifications they represent to be redone. This enables recovery of data changes for a given span of time. To execute events from the binary log, process `mysqlbinlog` output using the `mysql` client:

```
shell> mysqlbinlog binlog_files | mysql -u root -p
```

If binary log files have been encrypted, which can be done from MySQL 8.0.14 onwards, `mysqlbinlog` cannot read them directly as in the above example, but can read them from the server using the `--read-from-remote-server` (`-R`) option. For example:

```
shell> mysqlbinlog --read-from-remote-server --host=host_name --port=3306  --user=root --password --ssl-mode=r
```

Here, the option `--ssl-mode=required` has been used to ensure that the data from the binary log files is protected in transit, because it is sent to `mysqlbinlog` in an unencrypted format.

Viewing log contents can be useful when you need to determine event times or positions to select partial log contents prior to executing events. To view events from the log, send `mysqlbinlog` output into a paging program:

```
shell> mysqlbinlog binlog_files | more
```

Alternatively, save the output in a file and view the file in a text editor:

```
shell> mysqlbinlog binlog_files > tmpfile
shell> ... edit tmpfile ...
```

Saving the output in a file is useful as a preliminary to executing the log contents with certain events removed, such as an accidental `DROP DATABASE`. You can delete from the file any statements not to be executed before executing its contents. After editing the file, execute the contents as follows:

```
shell> mysql -u root -p < tmpfile
```

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using different connections to the server causes problems if the first log file contains a `CREATE TEMPORARY TABLE` statement and the second log contains a statement that uses the temporary table. When the first `mysql` process terminates, the server drops the temporary table. When the second `mysql` process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* connection to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

When writing to a dump file while reading back from a binary log containing GTIDs (see Section 17.1.3, “Replication with Global Transaction Identifiers”), use the `--skip-gtids` option with `mysqlbinlog`, like this:

```
shell> mysqlbinlog --skip-gtids binlog.000001 >  /tmp/dump.sql
shell> mysqlbinlog --skip-gtids binlog.000002 >> /tmp/dump.sql
shell> mysql -u root -p -e "source /tmp/dump.sql"
```

7.5.1 Point-in-Time Recovery Using Event Times

To indicate the start and end times for recovery, specify the `--start-datetime` and `--stop-datetime` options for `mysqlbinlog`, in `DATETIME` format. As an example, suppose that exactly at 10:00 a.m. on April 20, 2005 an SQL statement was executed that deleted a large table. To restore the table and data, you could restore the previous night's backup, and then execute the following command:

```
shell> mysqlbinlog --stop-datetime="2005-04-20 9:59:59" \
        /var/log/mysql/bin.123456 | mysql -u root -p
```

This command recovers all of the data up until the date and time given by the `--stop-datetime` option. If you did not detect the erroneous SQL statement that was entered until hours later, you will probably also want to recover the activity that occurred afterward. Based on this, you could run `mysqlbinlog` again with a start date and time, like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 10:01:00" \
        /var/log/mysql/bin.123456 | mysql -u root -p
```

In this command, the SQL statements logged from 10:01 a.m. on will be re-executed. The combination of restoring of the previous night's dump file and the two `mysqlbinlog` commands restores everything up until one second before 10:00 a.m. and everything from 10:01 a.m. on.

To use this method of point-in-time recovery, you should examine the log to be sure of the exact times to specify for the commands. To display the log file contents without executing them, use this command:

```
shell> mysqlbinlog /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

Then open the `/tmp/mysql_restore.sql` file with a text editor to examine it.

Excluding specific changes by specifying times for `mysqlbinlog` does not work well if multiple statements executed at the same time as the one to be excluded.

7.5.2 Point-in-Time Recovery Using Event Positions

Instead of specifying dates and times, the `--start-position` and `--stop-position` options for `mysqlbinlog` can be used for specifying log positions. They work the same as the start and stop date options, except that you specify log position numbers rather than dates. Using positions may enable you to be more precise about which part of the log to recover, especially if many transactions occurred around

the same time as a damaging SQL statement. To determine the position numbers, run `mysqlbinlog` for a range of times near the time when the unwanted transaction was executed, but redirect the results to a text file for examination. This can be done like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 9:55:00" \
         --stop-datetime="2005-04-20 10:05:00" \
         /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

This command creates a small text file in the `/tmp` directory that contains the SQL statements around the time that the deleterious SQL statement was executed. Open this file with a text editor and look for the statement that you do not want to repeat. Determine the positions in the binary log for stopping and resuming the recovery and make note of them. Positions are labeled as `log_pos` followed by a number. After restoring the previous backup file, use the position numbers to process the binary log file. For example, you would use commands something like these:

```
shell> mysqlbinlog --stop-position=368312 /var/log/mysql/bin.123456 \
         | mysql -u root -p

shell> mysqlbinlog --start-position=368315 /var/log/mysql/bin.123456 \
         | mysql -u root -p
```

The first command recovers all the transactions up until the stop position given. The second command recovers all transactions from the starting position given until the end of the binary log. Because the output of `mysqlbinlog` includes `SET TIMESTAMP` statements before each SQL statement recorded, the recovered data and related MySQL logs will reflect the original times at which the transactions were executed.

7.6 MyISAM Table Maintenance and Crash Recovery

This section discusses how to use `myisamchk` to check or repair `MyISAM` tables (tables that have `.MYD` and `.MYI` files for storing data and indexes). For general `myisamchk` background, see Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”. Other table-repair information can be found at Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”.

You can use `myisamchk` to check, repair, or optimize database tables. The following sections describe how to perform these operations and how to set up a table maintenance schedule. For information about using `myisamchk` to get information about your tables, see Section 4.6.4.5, “Obtaining Table Information with myisamchk”.

Even though table repair with `myisamchk` is quite secure, it is always a good idea to make a backup *before* doing a repair or any maintenance operation that could make a lot of changes to a table.

`myisamchk` operations that affect indexes can cause `MyISAM FULLTEXT` indexes to be rebuilt with full-text parameters that are incompatible with the values used by the MySQL server. To avoid this problem, follow the guidelines in Section 4.6.4.1, “myisamchk General Options”.

`MyISAM` table maintenance can also be done using the SQL statements that perform operations similar to what `myisamchk` can do:

- To check `MyISAM` tables, use `CHECK TABLE`.
- To repair `MyISAM` tables, use `REPAIR TABLE`.
- To optimize `MyISAM` tables, use `OPTIMIZE TABLE`.
- To analyze `MyISAM` tables, use `ANALYZE TABLE`.

For additional information about these statements, see Section 13.7.3, “Table Maintenance Statements”.

These statements can be used directly or by means of the `mysqlcheck` client program. One advantage of these statements over `myisamchk` is that the server does all the work. With `myisamchk`, you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between `myisamchk` and the server.

7.6.1 Using myisamchk for Crash Recovery

This section describes how to check for and deal with data corruption in MySQL databases. If your tables become corrupted frequently, you should try to find the reason why. See Section B.4.3.3, “What to Do If MySQL Keeps Crashing”.

For an explanation of how `MyISAM` tables can become corrupted, see Section 16.2.4, “MyISAM Table Problems”.

If you run `mysqld` with external locking disabled (which is the default), you cannot reliably use `myisamchk` to check a table when `mysqld` is using the same table. If you can be certain that no one will access the tables through `mysqld` while you run `myisamchk`, you only have to execute `mysqladmin flush-tables` before you start checking the tables. If you cannot guarantee this, you must stop `mysqld` while you check the tables. If you run `myisamchk` to check tables that `mysqld` is updating at the same time, you may get a warning that a table is corrupt even when it is not.

If the server is run with external locking enabled, you can use `myisamchk` to check tables at any time. In this case, if the server tries to update a table that `myisamchk` is using, the server will wait for `myisamchk` to finish before it continues.

If you use `myisamchk` to repair or optimize tables, you *must* always ensure that the `mysqld` server is not using the table (this also applies if external locking is disabled). If you do not stop `mysqld`, you should at least do a `mysqladmin flush-tables` before you run `myisamchk`. Your tables *may become corrupted* if the server and `myisamchk` access the tables simultaneously.

When performing crash recovery, it is important to understand that each `MyISAM` table *`tbl_name`* in a database corresponds to the three files in the database directory shown in the following table.

File	Purpose
`tbl_name``.MYD`	Data file
`tbl_name``.MYI`	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

`myisamchk` works by creating a copy of the `.MYD` data file row by row. It ends the repair stage by removing the old `.MYD` file and renaming the new file to the original file name. If you use `--quick`, `myisamchk` does not create a temporary `.MYD` file, but instead assumes that the `.MYD` file is correct and generates only a new index file without touching the `.MYD` file. This is safe, because `myisamchk` automatically detects whether the `.MYD` file is corrupt and aborts the repair if it is. You can also specify the `--quick` option twice to `myisamchk`. In this case, `myisamchk` does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the `.MYD` file. Normally the use of two `--quick` options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running `myisamchk`.

7.6.2 How to Check MyISAM Tables for Errors

To check a `MyISAM` table, use the following commands:

- `myisamchk tbl_name`

  This finds 99.99% of all errors. What it cannot find is corruption that involves *only* the data file (which is very unusual). If you want to check a table, you should normally run `myisamchk` without options or with the `-s` (silent) option.

- `myisamchk -m tbl_name`

  This finds 99.999% of all errors. It first checks all index entries for errors and then reads through all rows. It calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

- `myisamchk -e tbl_name`

  This does a complete and thorough check of all data (`-e` means “extended check”). It does a check-read of every key for each row to verify that they indeed point to the correct row. This may take a long time for a large table that has many indexes. Normally, `myisamchk` stops after the first error it finds. If you want to obtain more information, you can add the `-v` (verbose) option. This causes `myisamchk` to keep going, up through a maximum of 20 errors.

- `myisamchk -e -i tbl_name`

  This is like the previous command, but the `-i` option tells `myisamchk` to print additional statistical information.

In most cases, a simple `myisamchk` command with no arguments other than the table name is sufficient to check a table.

7.6.3 How to Repair MyISAM Tables

The discussion in this section describes how to use `myisamchk` on `MyISAM` tables (extensions `.MYI` and `.MYD`).

You can also use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See Section 13.7.3.2, “CHECK TABLE Syntax”, and Section 13.7.3.5, “REPAIR TABLE Syntax”.

Symptoms of corrupted tables include queries that abort unexpectedly and observable errors such as these:

- Can't find file *`tbl_name`*`.MYI` (Errcode: *`nnn`*)
- Unexpected end of file
- Record file is crashed
- Got error *`nnn`* from table handler

To get more information about the error, run `perror` *`nnn`*, where *`nnn`* is the error number. The following example shows how to use `perror` to find the meanings for the most common error numbers that indicate a problem with a table:

```
shell> perror 126 127 132 134 135 136 141 144 145
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 132 = Old database file
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 135 = No more room in record file
MySQL error code 136 = No more room in index file
```

```
MySQL error code 141 = Duplicate unique key or constraint on write or update
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Note that error 135 (no more room in record file) and error 136 (no more room in index file) are not errors that can be fixed by a simple repair. In this case, you must use `ALTER TABLE` to increase the `MAX_ROWS` and `AVG_ROW_LENGTH` table option values:

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

If you do not know the current table option values, use `SHOW CREATE TABLE`.

For the other errors, you must repair your tables. `myisamchk` can usually detect and fix most problems that occur.

The repair process involves up to three stages, described here. Before you begin, you should change location to the database directory and check the permissions of the table files. On Unix, make sure that they are readable by the user that `mysqld` runs as (and to you, because you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

This section is for the cases where a table check fails (such as those described in Section 7.6.2, “How to Check MyISAM Tables for Errors”), or you want to use the extended features that `myisamchk` provides.

The `myisamchk` options used for table maintenance with are described in Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”. `myisamchk` also has variables that you can set to control memory allocation that may improve performance. See Section 4.6.4.6, “myisamchk Memory Usage”.

If you are going to repair a table from the command line, you must first stop the `mysqld` server. Note that when you do `mysqladmin shutdown` on a remote server, the `mysqld` server is still available for a while after `mysqladmin` returns, until all statement-processing has stopped and all index changes have been flushed to disk.

Stage 1: Checking your tables

Run `myisamchk *.MYI` or `myisamchk -e *.MYI` if you have more time. Use the `-s` (silent) option to suppress unnecessary information.

If the `mysqld` server is stopped, you should use the `--update-state` option to tell `myisamchk` to mark the table as “checked.”

You have to repair only those tables for which `myisamchk` announces an error. For such tables, proceed to Stage 2.

If you get unexpected errors when checking (such as `out of memory` errors), or if `myisamchk` crashes, go to Stage 3.

Stage 2: Easy safe repair

First, try `myisamchk -r -q tbl_name` (`-r -q` means “quick recovery mode”). This attempts to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work, and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use `myisamchk -r tbl_name` (`-r` means “recovery mode”). This removes incorrect rows and deleted rows from the data file and reconstructs the index file.

3. If the preceding step fails, use `myisamchk --safe-recover tbl_name`. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower).

Note

If you want a repair operation to go much faster, you should set the values of the `sort_buffer_size` and `key_buffer_size` variables each to about 25% of your available memory when running `myisamchk`.

If you get unexpected errors when repairing (such as `out of memory` errors), or if `myisamchk` crashes, go to Stage 3.

Stage 3: Difficult repair

You should reach this stage only if the first 16KB block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it is necessary to create a new index file. Do so as follows:

1. Move the data file to a safe place.
2. Use the table description file to create new (empty) data and index files:

```
shell> mysql db_name
```

```
mysql> SET autocommit=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

3. Copy the old data file back onto the newly created data file. (Do not just move the old file back onto the new file. You want to retain a copy in case something goes wrong.)

Important

If you are using replication, you should stop it prior to performing the above procedure, since it involves file system operations, and these are not logged by MySQL.

Go back to Stage 2. `myisamchk -r -q` should work. (This should not be an endless loop.)

You can also use the `REPAIR TABLE tbl_name USE_FRM` SQL statement, which performs the whole procedure automatically. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `REPAIR TABLE`. See Section 13.7.3.5, “REPAIR TABLE Syntax”.

7.6.4 MyISAM Table Optimization

To coalesce fragmented rows and eliminate wasted space that results from deleting or updating rows, run `myisamchk` in recovery mode:

```
shell> myisamchk -r tbl_name
```

You can optimize a table in the same way by using the `OPTIMIZE TABLE` SQL statement. `OPTIMIZE TABLE` does a table repair and a key analysis, and also sorts the index tree so that key lookups are faster. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `OPTIMIZE TABLE`. See Section 13.7.3.4, “OPTIMIZE TABLE Syntax”.

`myisamchk` has a number of other options that you can use to improve the performance of a table:

- `--analyze` or `-a`: Perform key distribution analysis. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use.
- `--sort-index` or `-S`: Sort the index blocks. This optimizes seeks and makes table scans that use indexes faster.
- `--sort-records=`*`index_num`* or `-R` *`index_num`*: Sort data rows according to a given index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index.

For a full description of all available options, see Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”.

7.6.5 Setting Up a MyISAM Table Maintenance Schedule

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. One way to check and repair `MyISAM` tables is with the `CHECK TABLE` and `REPAIR TABLE` statements. See Section 13.7.3, “Table Maintenance Statements”.

Another way to check tables is to use `myisamchk`. For maintenance purposes, you can use `myisamchk -s`. The `-s` option (short for `--silent`) causes `myisamchk` to run in silent mode, printing messages only when errors occur.

It is also a good idea to enable automatic `MyISAM` table checking. For example, whenever the machine has done a restart in the middle of an update, you usually need to check each table that could have been affected before it is used further. (These are “expected crashed tables.”) To cause the server to check `MyISAM` tables automatically, start it with the `myisam_recover_options` system variable set. See Section 5.1.8, “Server System Variables”.

You should also check your tables regularly during normal system operation. For example, you can run a `cron` job to check important tables once a week, using a line like this in a `crontab` file:

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

This prints out information about crashed tables so that you can examine and repair them as necessary.

To start with, execute `myisamchk -s` each night on all tables that have been updated during the last 24 hours. As you see that problems occur infrequently, you can back off the checking frequency to once a week or so.

Normally, MySQL tables need little maintenance. If you are performing many updates to `MyISAM` tables with dynamic-sized rows (tables with `VARCHAR`, `BLOB`, or `TEXT` columns) or have tables with many deleted rows you may want to defragment/reclaim space from the tables from time to time. You can do this by using `OPTIMIZE TABLE` on the tables in question. Alternatively, if you can stop the `mysqld` server for a while, change location into the data directory and use this command while the server is stopped:

```
shell> myisamchk -r -s --sort-index --myisam_sort_buffer_size=16M */*.MYI
```

Chapter 8 Optimization

Table of Contents

8.1 Optimization Overview 1447
8.2 Optimizing SQL Statements 1448
8.2.1 Optimizing SELECT Statements 1448
8.2.2 Optimizing Subqueries, Derived Tables, View References, and Common Table Expressions 1502
8.2.3 Optimizing INFORMATION_SCHEMA Queries 1516
8.2.4 Optimizing Performance Schema Queries 1519
8.2.5 Optimizing Data Change Statements 1520
8.2.6 Optimizing Database Privileges 1522
8.2.7 Other Optimization Tips 1522
8.3 Optimization and Indexes 1522
8.3.1 How MySQL Uses Indexes 1523
8.3.2 Primary Key Optimization 1524
8.3.3 SPATIAL Index Optimization 1524
8.3.4 Foreign Key Optimization 1525
8.3.5 Column Indexes 1525
8.3.6 Multiple-Column Indexes 1527
8.3.7 Verifying Index Usage 1528
8.3.8 InnoDB and MyISAM Index Statistics Collection 1528
8.3.9 Comparison of B-Tree and Hash Indexes 1530
8.3.10 Use of Index Extensions 1531
8.3.11 Optimizer Use of Generated Column Indexes 1534
8.3.12 Invisible Indexes 1536
8.3.13 Descending Indexes 1537
8.3.14 Indexed Lookups from TIMESTAMP Columns 1538
8.4 Optimizing Database Structure 1540
8.4.1 Optimizing Data Size 1540
8.4.2 Optimizing MySQL Data Types 1542
8.4.3 Optimizing for Many Tables 1544
8.4.4 Internal Temporary Table Use in MySQL 1545
8.4.5 Limits on Number of Databases and Tables 1549
8.4.6 Limits on Table Size 1549
8.4.7 Limits on Table Column Count and Row Size 1550
8.5 Optimizing for InnoDB Tables 1553
8.5.1 Optimizing Storage Layout for InnoDB Tables 1553
8.5.2 Optimizing InnoDB Transaction Management 1554
8.5.3 Optimizing InnoDB Read-Only Transactions 1555
8.5.4 Optimizing InnoDB Redo Logging 1556
8.5.5 Bulk Data Loading for InnoDB Tables 1557
8.5.6 Optimizing InnoDB Queries 1558
8.5.7 Optimizing InnoDB DDL Operations 1559
8.5.8 Optimizing InnoDB Disk I/O 1559
8.5.9 Optimizing InnoDB Configuration Variables 1562
8.5.10 Optimizing InnoDB for Systems with Many Tables 1564
8.6 Optimizing for MyISAM Tables 1564
8.6.1 Optimizing MyISAM Queries 1564
8.6.2 Bulk Data Loading for MyISAM Tables 1565

8.6.3 Optimizing REPAIR TABLE Statements ... 1566
8.7 Optimizing for MEMORY Tables ... 1568
8.8 Understanding the Query Execution Plan ... 1568
8.8.1 Optimizing Queries with EXPLAIN ... 1568
8.8.2 EXPLAIN Output Format ... 1569
8.8.3 Extended EXPLAIN Output Format ... 1583
8.8.4 Obtaining Execution Plan Information for a Named Connection ... 1585
8.8.5 Estimating Query Performance ... 1586
8.9 Controlling the Query Optimizer ... 1586
8.9.1 Controlling Query Plan Evaluation ... 1586
8.9.2 Switchable Optimizations ... 1587
8.9.3 Optimizer Hints ... 1592
8.9.4 Index Hints ... 1606
8.9.5 The Optimizer Cost Model ... 1608
8.9.6 Optimizer Statistics ... 1612
8.10 Buffering and Caching ... 1615
8.10.1 InnoDB Buffer Pool Optimization ... 1615
8.10.2 The MyISAM Key Cache ... 1615
8.10.3 Caching of Prepared Statements and Stored Programs ... 1620
8.11 Optimizing Locking Operations ... 1621
8.11.1 Internal Locking Methods ... 1622
8.11.2 Table Locking Issues ... 1624
8.11.3 Concurrent Inserts ... 1626
8.11.4 Metadata Locking ... 1626
8.11.5 External Locking ... 1630
8.12 Optimizing the MySQL Server ... 1631
8.12.1 Optimizing Disk I/O ... 1631
8.12.2 Using Symbolic Links ... 1632
8.12.3 Optimizing Memory Use ... 1635
8.12.4 Optimizing Network Use ... 1642
8.12.5 Resource Groups ... 1646
8.13 Measuring Performance (Benchmarking) ... 1651
8.13.1 Measuring the Speed of Expressions and Functions ... 1652
8.13.2 Using Your Own Benchmarks ... 1652
8.13.3 Measuring Performance with performance_schema ... 1652
8.14 Examining Thread Information ... 1653
8.14.1 Thread Command Values ... 1654
8.14.2 General Thread States ... 1656
8.14.3 Replication Master Thread States ... 1662
8.14.4 Replication Slave I/O Thread States ... 1663
8.14.5 Replication Slave SQL Thread States ... 1664
8.14.6 Replication Slave Connection Thread States ... 1665
8.14.7 NDB Cluster Thread States ... 1665
8.14.8 Event Scheduler Thread States ... 1666

This chapter explains how to optimize MySQL performance and provides examples. Optimization involves configuring, tuning, and measuring performance, at several levels. Depending on your job role (developer, DBA, or a combination of both), you might optimize at the level of individual SQL statements, entire applications, a single database server, or multiple networked database servers. Sometimes you can be proactive and plan in advance for performance, while other times you might troubleshoot a configuration or code issue after a problem occurs. Optimizing CPU and memory usage can also improve scalability, allowing the database to handle more load without slowing down.

8.1 Optimization Overview

Database performance depends on several factors at the database level, such as tables, queries, and configuration settings. These software constructs result in CPU and I/O operations at the hardware level, which you must minimize and make as efficient as possible. As you work on database performance, you start by learning the high-level rules and guidelines for the software side, and measuring performance using wall-clock time. As you become an expert, you learn more about what happens internally, and start measuring things such as CPU cycles and I/O operations.

Typical users aim to get the best database performance out of their existing software and hardware configurations. Advanced users look for opportunities to improve the MySQL software itself, or develop their own storage engines and hardware appliances to expand the MySQL ecosystem.

- Optimizing at the Database Level
- Optimizing at the Hardware Level
- Balancing Portability and Performance

Optimizing at the Database Level

The most important factor in making a database application fast is its basic design:

- Are the tables structured properly? In particular, do the columns have the right data types, and does each table have the appropriate columns for the type of work? For example, applications that perform frequent updates often have many tables with few columns, while applications that analyze large amounts of data often have few tables with many columns.
- Are the right indexes in place to make queries efficient?
- Are you using the appropriate storage engine for each table, and taking advantage of the strengths and features of each storage engine you use? In particular, the choice of a transactional storage engine such as `InnoDB` or a nontransactional one such as `MyISAM` can be very important for performance and scalability.

Note

`InnoDB` is the default storage engine for new tables. In practice, the advanced `InnoDB` performance features mean that `InnoDB` tables often outperform the simpler `MyISAM` tables, especially for a busy database.

- Does each table use an appropriate row format? This choice also depends on the storage engine used for the table. In particular, compressed tables use less disk space and so require less disk I/O to read and write the data. Compression is available for all kinds of workloads with `InnoDB` tables, and for read-only `MyISAM` tables.
- Does the application use an appropriate locking strategy? For example, by allowing shared access when possible so that database operations can run concurrently, and requesting exclusive access when appropriate so that critical operations get top priority. Again, the choice of storage engine is significant. The `InnoDB` storage engine handles most locking issues without involvement from you, allowing for better concurrency in the database and reducing the amount of experimentation and tuning for your code.
- Are all memory areas used for caching sized correctly? That is, large enough to hold frequently accessed data, but not so large that they overload physical memory and cause paging. The main memory areas to configure are the `InnoDB` buffer pool and the `MyISAM` key cache.

Optimizing at the Hardware Level

Any database application eventually hits hardware limits as the database becomes more and more busy. A DBA must evaluate whether it is possible to tune the application or reconfigure the server to avoid these bottlenecks, or whether more hardware resources are required. System bottlenecks typically arise from these sources:

- Disk seeks. It takes time for the disk to find a piece of data. With modern disks, the mean time for this is usually lower than 10ms, so we can in theory do about 100 seeks a second. This time improves slowly with new disks and is very hard to optimize for a single table. The way to optimize seek time is to distribute the data onto more than one disk.
- Disk reading and writing. When the disk is at the correct position, we need to read or write the data. With modern disks, one disk delivers at least 10–20MB/s throughput. This is easier to optimize than seeks because you can read in parallel from multiple disks.
- CPU cycles. When the data is in main memory, we must process it to get our result. Having large tables compared to the amount of memory is the most common limiting factor. But with small tables, speed is usually not the problem.
- Memory bandwidth. When the CPU needs more data than can fit in the CPU cache, main memory bandwidth becomes a bottleneck. This is an uncommon bottleneck for most systems, but one to be aware of.

Balancing Portability and Performance

To use performance-oriented SQL extensions in a portable MySQL program, you can wrap MySQL-specific keywords in a statement within `/*! */` comment delimiters. Other SQL servers ignore the commented keywords. For information about writing comments, see Section 9.6, “Comment Syntax”.

8.2 Optimizing SQL Statements

The core logic of a database application is performed through SQL statements, whether issued directly through an interpreter or submitted behind the scenes through an API. The tuning guidelines in this section help to speed up all kinds of MySQL applications. The guidelines cover SQL operations that read and write data, the behind-the-scenes overhead for SQL operations in general, and operations used in specific scenarios such as database monitoring.

8.2.1 Optimizing SELECT Statements

Queries, in the form of `SELECT` statements, perform all the lookup operations in the database. Tuning these statements is a top priority, whether to achieve sub-second response times for dynamic web pages, or to chop hours off the time to generate huge overnight reports.

Besides `SELECT` statements, the tuning techniques for queries also apply to constructs such as `CREATE TABLE...AS SELECT`, `INSERT INTO...SELECT`, and `WHERE` clauses in `DELETE` statements. Those statements have additional performance considerations because they combine write operations with the read-oriented query operations.

NDB Cluster supports a join pushdown optimization whereby a qualifying join is sent in its entirety to NDB Cluster data nodes, where it can be distributed among them and executed in parallel. For more information about this optimization, see Conditions for NDB pushdown joins,

The main considerations for optimizing queries are:

- To make a slow `SELECT ... WHERE` query faster, the first thing to check is whether you can add an index. Set up indexes on columns used in the `WHERE` clause, to speed up evaluation, filtering, and the

final retrieval of results. To avoid wasted disk space, construct a small set of indexes that speed up many related queries used in your application.

Indexes are especially important for queries that reference different tables, using features such as joins and foreign keys. You can use the `EXPLAIN` statement to determine which indexes are used for a `SELECT`. See Section 8.3.1, “How MySQL Uses Indexes” and Section 8.8.1, “Optimizing Queries with EXPLAIN”.

- Isolate and tune any part of the query, such as a function call, that takes excessive time. Depending on how the query is structured, a function could be called once for every row in the result set, or even once for every row in the table, greatly magnifying any inefficiency.
- Minimize the number of full table scans in your queries, particularly for big tables.
- Keep table statistics up to date by using the `ANALYZE TABLE` statement periodically, so the optimizer has the information needed to construct an efficient execution plan.
- Learn the tuning techniques, indexing techniques, and configuration parameters that are specific to the storage engine for each table. Both `InnoDB` and `MyISAM` have sets of guidelines for enabling and sustaining high performance in queries. For details, see Section 8.5.6, “Optimizing InnoDB Queries” and Section 8.6.1, “Optimizing MyISAM Queries”.
- You can optimize single-query transactions for `InnoDB` tables, using the technique in Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”.
- Avoid transforming the query in ways that make it hard to understand, especially if the optimizer does some of the same transformations automatically.
- If a performance issue is not easily solved by one of the basic guidelines, investigate the internal details of the specific query by reading the `EXPLAIN` plan and adjusting your indexes, `WHERE` clauses, join clauses, and so on. (When you reach a certain level of expertise, reading the `EXPLAIN` plan might be your first step for every query.)
- Adjust the size and properties of the memory areas that MySQL uses for caching. With efficient use of the `InnoDB` buffer pool, `MyISAM` key cache, and the MySQL query cache, repeated queries run faster because the results are retrieved from memory the second and subsequent times.
- Even for a query that runs fast using the cache memory areas, you might still optimize further so that they require less cache memory, making your application more scalable. Scalability means that your application can handle more simultaneous users, larger requests, and so on without experiencing a big drop in performance.
- Deal with locking issues, where the speed of your query might be affected by other sessions accessing the tables at the same time.

8.2.1.1 WHERE Clause Optimization

This section discusses optimizations that can be made for processing `WHERE` clauses. The examples use `SELECT` statements, but the same optimizations apply for `WHERE` clauses in `DELETE` and `UPDATE` statements.

Note

Because work on the MySQL optimizer is ongoing, not all of the optimizations that MySQL performs are documented here.

You might be tempted to rewrite your queries to make arithmetic operations faster, while sacrificing readability. Because MySQL does similar optimizations automatically, you can often avoid this work, and

leave the query in a more understandable and maintainable form. Some of the optimizations performed by MySQL follow:

- Removal of unnecessary parentheses:

```
   ((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- Constant folding:

```
   (a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Constant condition removal:

```
   (b>=5 AND b=5) OR (b=6 AND 5=5) OR (b=7 AND 5=6)
-> b=5 OR b=6
```

  In MySQL 8.0.14 and later, this takes place during preparation rather than during the optimization phase, which helps in simplification of joins. See Section 8.2.1.9, “Outer Join Optimization”, for further information and examples.

- Constant expressions used by indexes are evaluated only once.

- Beginning with MySQL 8.0.16, comparisons of columns of numeric types with constant values are checked and folded or removed for invalid or out-of-rage values:

```
# CREATE TABLE t (c TINYINT UNSIGNED NOT NULL);
  SELECT * FROM t WHERE c ≪ 256;
-≫ SELECT * FROM t WHERE 1;
```

  See Section 8.2.1.14, “Constant-Folding Optimization”, for more information.

- `COUNT(*)` on a single table without a `WHERE` is retrieved directly from the table information for `MyISAM` and `MEMORY` tables. This is also done for any `NOT NULL` expression when used with only one table.

- Early detection of invalid constant expressions. MySQL quickly detects that some `SELECT` statements are impossible and returns no rows.

- `HAVING` is merged with `WHERE` if you do not use `GROUP BY` or aggregate functions (`COUNT()`, `MIN()`, and so on).

- For each table in a join, a simpler `WHERE` is constructed to get a fast `WHERE` evaluation for the table and also to skip rows as soon as possible.

- All constant tables are read first before any other tables in the query. A constant table is any of the following:

  - An empty table or a table with one row.

  - A table that is used with a `WHERE` clause on a `PRIMARY KEY` or a `UNIQUE` index, where all index parts are compared to constant expressions and are defined as `NOT NULL`.

  All of the following tables are used as constant tables:

```
SELECT * FROM t WHERE primary_key=1;
```

```
SELECT * FROM t1,t2
  WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- The best join combination for joining the tables is found by trying all possibilities. If all columns in `ORDER BY` and `GROUP BY` clauses come from the same table, that table is preferred first when joining.
- If there is an `ORDER BY` clause and a different `GROUP BY` clause, or if the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue, a temporary table is created.
- If you use the `SQL_SMALL_RESULT` modifier, MySQL uses an in-memory temporary table.
- Each table index is queried, and the best index is used unless the optimizer believes that it is more efficient to use a table scan. At one time, a scan was used based on whether the best index spanned more than 30% of the table, but a fixed percentage no longer determines the choice between using an index or a scan. The optimizer now is more complex and bases its estimate on additional factors such as table size, number of rows, and I/O block size.
- In some cases, MySQL can read rows from the index without even consulting the data file. If all columns used from the index are numeric, only the index tree is used to resolve the query.
- Before each row is output, those that do not match the `HAVING` clause are skipped.

Some examples of queries that are very fast:

```
SELECT COUNT(*) FROM tbl_name;

SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;

SELECT MAX(key_part2) FROM tbl_name
  WHERE key_part1=constant;

SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... LIMIT 10;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

MySQL resolves the following queries using only the index tree, assuming that the indexed columns are numeric:

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;

SELECT COUNT(*) FROM tbl_name
  WHERE key_part1=val1 AND key_part2=val2;

SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use indexing to retrieve the rows in sorted order without a separate sorting pass:

```
SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... ;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

8.2.1.2 Range Optimization

The `range` access method uses a single index to retrieve a subset of table rows that are contained within one or several index value intervals. It can be used for a single-part or multiple-part index. The following sections describe conditions under which the optimizer uses range access.

- Range Access Method for Single-Part Indexes
- Range Access Method for Multiple-Part Indexes
- Equality Range Optimization of Many-Valued Comparisons
- Skip Scan Range Access Method
- Range Optimization of Row Constructor Expressions
- Limiting Memory Use for Range Optimization

Range Access Method for Single-Part Indexes

For a single-part index, index value intervals can be conveniently represented by corresponding conditions in the `WHERE` clause, denoted as *range conditions* rather than "intervals."

The definition of a range condition for a single-part index is as follows:

- For both `BTREE` and `HASH` indexes, comparison of a key part with a constant value is a range condition when using the `=`, `<=>`, `IN()`, `IS NULL`, or `IS NOT NULL` operators.
- Additionally, for `BTREE` indexes, comparison of a key part with a constant value is a range condition when using the `>`, `<`, `>=`, `<=`, `BETWEEN`, `!=`, or `<>` operators, or `LIKE` comparisons if the argument to `LIKE` is a constant string that does not start with a wildcard character.
- For all index types, multiple range conditions combined with `OR` or `AND` form a range condition.

"Constant value" in the preceding descriptions means one of the following:

- A constant from the query string
- A column of a `const` or `system` table from the same join
- The result of an uncorrelated subquery
- Any expression composed entirely from subexpressions of the preceding types

Here are some examples of queries with range conditions in the `WHERE` clause:

```
SELECT * FROM t1
  WHERE key_col > 1
  AND key_col < 10;

SELECT * FROM t1
  WHERE key_col = 1
  OR key_col IN (15,18,20);

SELECT * FROM t1
  WHERE key_col LIKE 'ab%'
  OR key_col BETWEEN 'bar' AND 'foo';
```

Some nonconstant values may be converted to constants during the optimizer constant propagation phase.

MySQL tries to extract range conditions from the `WHERE` clause for each of the possible indexes. During the extraction process, conditions that cannot be used for constructing the range condition are dropped, conditions that produce overlapping ranges are combined, and conditions that produce empty ranges are removed.

Consider the following statement, where `key1` is an indexed column and `nonkey` is not indexed:

```
SELECT * FROM t1 WHERE
  (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
  (key1 < 'bar' AND nonkey = 4) OR
  (key1 < 'uux' AND key1 > 'z');
```

The extraction process for key `key1` is as follows:

1. Start with original `WHERE` clause:

   ```
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
   (key1 < 'bar' AND nonkey = 4) OR
   (key1 < 'uux' AND key1 > 'z')
   ```

2. Remove `nonkey = 4` and `key1 LIKE '%b'` because they cannot be used for a range scan. The correct way to remove them is to replace them with `TRUE`, so that we do not miss any matching rows when doing the range scan. Replacing them with `TRUE` yields:

   ```
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
   (key1 < 'bar' AND TRUE) OR
   (key1 < 'uux' AND key1 > 'z')
   ```

3. Collapse conditions that are always true or false:

   - `(key1 LIKE 'abcde%' OR TRUE)` is always true
   - `(key1 < 'uux' AND key1 > 'z')` is always false

   Replacing these conditions with constants yields:

   ```
   (key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
   ```

   Removing unnecessary `TRUE` and `FALSE` constants yields:

   ```
   (key1 < 'abc') OR (key1 < 'bar')
   ```

4. Combining overlapping intervals into one yields the final condition to be used for the range scan:

   ```
   (key1 < 'bar')
   ```

In general (and as demonstrated by the preceding example), the condition used for a range scan is less restrictive than the `WHERE` clause. MySQL performs an additional check to filter out rows that satisfy the range condition but not the full `WHERE` clause.

The range condition extraction algorithm can handle nested `AND`/`OR` constructs of arbitrary depth, and its output does not depend on the order in which conditions appear in `WHERE` clause.

MySQL does not support merging multiple ranges for the `range` access method for spatial indexes. To work around this limitation, you can use a `UNION` with identical `SELECT` statements, except that you put each spatial predicate in a different `SELECT`.

Range Access Method for Multiple-Part Indexes

Range conditions on a multiple-part index are an extension of range conditions for a single-part index. A range condition on a multiple-part index restricts index rows to lie within one or several key tuple intervals. Key tuple intervals are defined over a set of key tuples, using ordering from the index.

For example, consider a multiple-part index defined as `key1(key_part1, key_part2, key_part3)`, and the following set of key tuples listed in key order:

```
key_part1  key_part2  key_part3
  NULL       1          'abc'
  NULL       1          'xyz'
  NULL       2          'foo'
   1         1          'abc'
   1         1          'xyz'
   1         2          'abc'
   2         1          'aaa'
```

The condition `key_part1 = 1` defines this interval:

```
(1,-inf,-inf) <= (key_part1,key_part2,key_part3) < (1,+inf,+inf)
```

The interval covers the 4th, 5th, and 6th tuples in the preceding data set and can be used by the range access method.

By contrast, the condition `key_part3 = 'abc'` does not define a single interval and cannot be used by the range access method.

The following descriptions indicate how range conditions work for multiple-part indexes in greater detail.

- For `HASH` indexes, each interval containing identical values can be used. This means that the interval can be produced only for conditions in the following form:

  ```
      key_part1 cmp const1
  AND key_part2 cmp const2
  AND ...
  AND key_partN cmp constN;
  ```

  Here, `const1`, `const2`, ... are constants, `cmp` is one of the `=`, `<=>`, or `IS NULL` comparison operators, and the conditions cover all index parts. (That is, there are `N` conditions, one for each part of an `N`-part index.) For example, the following is a range condition for a three-part `HASH` index:

  ```
  key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
  ```

  For the definition of what is considered to be a constant, see Range Access Method for Single-Part Indexes.

- For a `BTREE` index, an interval might be usable for conditions combined with `AND`, where each condition compares a key part with a constant value using `=`, `<=>`, `IS NULL`, `>`, `<`, `>=`, `<=`, `!=`, `<>`, `BETWEEN`, or `LIKE 'pattern'` (where `'pattern'` does not start with a wildcard). An interval can be used as long as it is possible to determine a single key tuple containing all rows that match the condition (or two intervals if `<>` or `!=` is used).

  The optimizer attempts to use additional key parts to determine the interval as long as the comparison operator is `=`, `<=>`, or `IS NULL`. If the operator is `>`, `<`, `>=`, `<=`, `!=`, `<>`, `BETWEEN`, or `LIKE`, the optimizer uses it but considers no more key parts. For the following expression, the optimizer uses `=` from the first comparison. It also uses `>=` from the second comparison but considers no further key parts and does not use the third comparison for interval construction:

  ```
  key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
  ```

  The single interval is:

```
('foo',10,-inf) < (key_part1,key_part2,key_part3) < ('foo',+inf,+inf)
```

It is possible that the created interval contains more rows than the initial condition. For example, the preceding interval includes the value `('foo', 11, 0)`, which does not satisfy the original condition.

- If conditions that cover sets of rows contained within intervals are combined with `OR`, they form a condition that covers a set of rows contained within the union of their intervals. If the conditions are combined with `AND`, they form a condition that covers a set of rows contained within the intersection of their intervals. For example, for this condition on a two-part index:

```
(key_part1 = 1 AND key_part2 < 2) OR (key_part1 > 5)
```

The intervals are:

```
(1,-inf) < (key_part1,key_part2) < (1,2)
(5,-inf) < (key_part1,key_part2)
```

In this example, the interval on the first line uses one key part for the left bound and two key parts for the right bound. The interval on the second line uses only one key part. The `key_len` column in the `EXPLAIN` output indicates the maximum length of the key prefix used.

In some cases, `key_len` may indicate that a key part was used, but that might be not what you would expect. Suppose that *`key_part1`* and *`key_part2`* can be `NULL`. Then the `key_len` column displays two key part lengths for the following condition:

```
key_part1 >= 1 AND key_part2 < 2
```

But, in fact, the condition is converted to this:

```
key_part1 >= 1 AND key_part2 IS NOT NULL
```

For a description of how optimizations are performed to combine or eliminate intervals for range conditions on a single-part index, see Range Access Method for Single-Part Indexes. Analogous steps are performed for range conditions on multiple-part indexes.

Equality Range Optimization of Many-Valued Comparisons

Consider these expressions, where *`col_name`* is an indexed column:

```
col_name IN(val1, ..., valN)
col_name = val1 OR ... OR col_name = valN
```

Each expression is true if *`col_name`* is equal to any of several values. These comparisons are equality range comparisons (where the “range” is a single value). The optimizer estimates the cost of reading qualifying rows for equality range comparisons as follows:

- If there is a unique index on *`col_name`*, the row estimate for each range is 1 because at most one row can have the given value.
- Otherwise, any index on *`col_name`* is nonunique and the optimizer can estimate the row count for each range using dives into the index or index statistics.

With index dives, the optimizer makes a dive at each end of a range and uses the number of rows in the range as the estimate. For example, the expression *`col_name`* `IN (10, 20, 30)` has three equality ranges and the optimizer makes two dives per range to generate a row estimate. Each pair of dives yields an estimate of the number of rows that have the given value.

Index dives provide accurate row estimates, but as the number of comparison values in the expression increases, the optimizer takes longer to generate a row estimate. Use of index statistics is less accurate than index dives but permits faster row estimation for large value lists.

The `eq_range_index_dive_limit` system variable enables you to configure the number of values at which the optimizer switches from one row estimation strategy to the other. To permit use of index dives for comparisons of up to *`N`* equality ranges, set `eq_range_index_dive_limit` to *`N`* + 1. To disable use of statistics and always use index dives regardless of *`N`*, set `eq_range_index_dive_limit` to 0.

To update table index statistics for best estimates, use `ANALYZE TABLE`.

Prior to MySQL 8.0, there is no way of skipping the use of index dives to estimate index usefulness, except by using the `eq_range_index_dive_limit` system variable. In MySQL 8.0, index dive skipping is possible for queries that satisfy all these conditions:

- The query is for a single table, not a join on multiple tables.
- A single-index `FORCE INDEX` index hint is present. The idea is that if index use is forced, there is nothing to be gained from the additional overhead of performing dives into the index.
- The index is nonunique and not a `FULLTEXT` index.
- No subquery is present.
- No `DISTINCT`, `GROUP BY`, or `ORDER BY` clause is present.

For `EXPLAIN FOR CONNECTION`, the output changes as follows if index dives are skipped:

- For traditional output, the `rows` and `filtered` values are `NULL`.
- For JSON output, `rows_examined_per_scan` and `rows_produced_per_join` do not appear, `skip_index_dive_due_to_force` is `true`, and cost calculations are not accurate.

Without `FOR CONNECTION`, `EXPLAIN` output does not change when index dives are skipped.

After execution of a query for which index dives are skipped, the corresponding row in the `INFORMATION_SCHEMA.OPTIMIZER_TRACE` table contains an `index_dives_for_range_access` value of `skipped_due_to_force_index`.

Skip Scan Range Access Method

Consider the following scenario:

```
CREATE TABLE t1 (f1 INT NOT NULL, f2 INT NOT NULL, PRIMARY KEY(f1, f2));
INSERT INTO t1 VALUES
  (1,1), (1,2), (1,3), (1,4), (1,5),
  (2,1), (2,2), (2,3), (2,4), (2,5);
INSERT INTO t1 SELECT f1, f2 + 5 FROM t1;
INSERT INTO t1 SELECT f1, f2 + 10 FROM t1;
INSERT INTO t1 SELECT f1, f2 + 20 FROM t1;
INSERT INTO t1 SELECT f1, f2 + 40 FROM t1;
ANALYZE TABLE t1;
```

```
EXPLAIN SELECT f1, f2 FROM t1 WHERE f2 > 40;
```

To execute this query, MySQL can choose an index scan to fetch all rows (the index includes all columns to be selected), then apply the `f2 > 40` condition from the `WHERE` clause to produce the final result set.

A range scan is more efficient than a full index scan, but cannot be used in this case because there is no condition on `f1`, the first index column. However, as of MySQL 8.0.13, the optimizer can perform multiple range scans, one for each value of `f1`, using a method called Skip Scan that is similar to Loose Index Scan (see Section 8.2.1.17, “GROUP BY Optimization”):

1. Skip between distinct values of the first index part, `f1` (the index prefix).
2. Perform a subrange scan on each distinct prefix value for the `f2 > 40` condition on the remaining index part.

For the data set shown earlier, the algorithm operates like this:

1. Get the first distinct value of the first key part (`f1 = 1`).
2. Construct the range based on the first and second key parts (`f1 = 1 AND f2 > 40`).
3. Perform a range scan.
4. Get the next distinct value of the first key part (`f1 = 2`).
5. Construct the range based on the first and second key parts (`f1 = 2 AND f2 > 40`).
6. Perform a range scan.

Using this strategy decreases the number of accessed rows because MySQL skips the rows that do not qualify for each constructed range. This Skip Scan access method is applicable under the following conditions:

- Table T has at least one compound index with key parts of the form ([A_1, ..., A_*k*,] B_1, ..., B_*m*, C [, D_1, ..., D_*n*]). Key parts A and D may be empty, but B and C must be nonempty.
- The query references only one table.
- The query does not use `GROUP BY` or `DISTINCT`.
- The query references only columns in the index.
- The predicates on A_1, ..., A_*k* must be equality predicates and they must be constants. This includes the `IN()` operator.
- The query must be a conjunctive query; that is, an `AND` of `OR` conditions: `(`*`cond1`*`(`*`key_part1`*`) OR `*`cond2`*`(`*`key_part1`*`)) AND (`*`cond1`*`(`*`key_part2`*`) OR ...) AND ...`
- There must be a range condition on C.
- Conditions on D columns are permitted. Conditions on D must be in conjunction with the range condition on C.

Use of Skip Scan is indicated in `EXPLAIN` output as follows:

- `Using index for skip scan` in the `Extra` column indicates that the loose index Skip Scan access method is used.
- If the index can be used for Skip Scan, the index should be visible in the `possible_keys` column.

Use of Skip Scan is indicated in optimizer trace output by a `"skip scan"` element of this form:

```
"skip_scan_range": {
  "type": "skip_scan",
  "index": index_used_for_skip_scan,
  "key_parts_used_for_access": [key_parts_used_for_access],
  "range": [range]
}
```

You may also see a `"best_skip_scan_summary"` element. If Skip Scan is chosen as the best range access variant, a `"chosen_range_access_summary"` is written. If Skip Scan is chosen as the overall best access method, a `"best_access_path"` element is present.

Use of Skip Scan is subject to the value of the `skip_scan` flag of the `optimizer_switch` system variable. See Section 8.9.2, “Switchable Optimizations”. By default, this flag is `on`. To disable it, set `skip_scan` to `off`.

In addition to using the `optimizer_switch` system variable to control optimizer use of Skip Scan session-wide, MySQL supports optimizer hints to influence the optimizer on a per-statement basis. See Section 8.9.3, “Optimizer Hints”.

Range Optimization of Row Constructor Expressions

The optimizer is able to apply the range scan access method to queries of this form:

```
SELECT ... FROM t1 WHERE ( col_1, col_2 ) IN (( 'a', 'b' ), ( 'c', 'd' ));
```

Previously, for range scans to be used, it was necessary to write the query as:

```
SELECT ... FROM t1 WHERE ( col_1 = 'a' AND col_2 = 'b' )
OR ( col_1 = 'c' AND col_2 = 'd' );
```

For the optimizer to use a range scan, queries must satisfy these conditions:

- Only `IN()` predicates are used, not `NOT IN()`.
- On the left side of the `IN()` predicate, the row constructor contains only column references.
- On the right side of the `IN()` predicate, row constructors contain only runtime constants, which are either literals or local column references that are bound to constants during execution.
- On the right side of the `IN()` predicate, there is more than one row constructor.

For more information about the optimizer and row constructors, see Section 8.2.1.22, “Row Constructor Expression Optimization”

Limiting Memory Use for Range Optimization

To control the memory available to the range optimizer, use the `range_optimizer_max_mem_size` system variable:

- A value of 0 means “no limit.”
- With a value greater than 0, the optimizer tracks the memory consumed when considering the range access method. If the specified limit is about to be exceeded, the range access method is abandoned and other methods, including a full table scan, are considered instead. This could be less optimal. If this

happens, the following warning occurs (where *N* is the current `range_optimizer_max_mem_size` value):

```
Warning    3170    Memory capacity of N bytes for
                   'range_optimizer_max_mem_size' exceeded. Range
                   optimization was not done for this query.
```

- For `UPDATE` and `DELETE` statements, if the optimizer falls back to a full table scan and the `sql_safe_updates` system variable is enabled, an error occurs rather than a warning because, in effect, no key is used to determine which rows to modify. For more information, see Using Safe-Updates Mode (--safe-updates).

For individual queries that exceed the available range optimization memory and for which the optimizer falls back to less optimal plans, increasing the `range_optimizer_max_mem_size` value may improve performance.

To estimate the amount of memory needed to process a range expression, use these guidelines:

- For a simple query such as the following, where there is one candidate key for the range access method, each predicate combined with `OR` uses approximately 230 bytes:

```
SELECT COUNT(*) FROM t
WHERE a=1 OR a=2 OR a=3 OR .. . a=N;
```

- Similarly for a query such as the following, each predicate combined with `AND` uses approximately 125 bytes:

```
SELECT COUNT(*) FROM t
WHERE a=1 AND b=1 AND c=1 ... N;
```

- For a query with `IN()` predicates:

```
SELECT COUNT(*) FROM t
WHERE a IN (1,2, ..., M) AND b IN (1,2, ..., N);
```

  Each literal value in an `IN()` list counts as a predicate combined with `OR`. If there are two `IN()` lists, the number of predicates combined with `OR` is the product of the number of literal values in each list. Thus, the number of predicates combined with `OR` in the preceding case is *M* × *N*.

8.2.1.3 Index Merge Optimization

The *Index Merge* access method retrieves rows with multiple `range` scans and merges their results into one. This access method merges index scans from a single table only, not scans across multiple tables. The merge can produce unions, intersections, or unions-of-intersections of its underlying scans.

Example queries for which Index Merge may be used:

```
SELECT * FROM tbl_name WHERE key1 = 10 OR key2 = 20;

SELECT * FROM tbl_name
  WHERE (key1 = 10 OR key2 = 20) AND non_key = 30;

SELECT * FROM t1, t2
  WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
  AND t2.key1 = t1.some_col;
```

```
SELECT * FROM t1, t2
  WHERE t1.key1 = 1
  AND (t2.key1 = t1.some_col OR t2.key2 = t1.some_col2);
```

Note

The Index Merge optimization algorithm has the following known limitations:

- If your query has a complex `WHERE` clause with deep `AND`/`OR` nesting and MySQL does not choose the optimal plan, try distributing terms using the following identity transformations:

```
(x AND y) OR z => (x OR z) AND (y OR z)
(x OR y) AND z => (x AND z) OR (y AND z)
```

- Index Merge is not applicable to full-text indexes.

In `EXPLAIN` output, the Index Merge method appears as `index_merge` in the `type` column. In this case, the `key` column contains a list of indexes used, and `key_len` contains a list of the longest key parts for those indexes.

The Index Merge access method has several algorithms, which are displayed in the `Extra` field of `EXPLAIN` output:

- `Using intersect(...)`
- `Using union(...)`
- `Using sort_union(...)`

The following sections describe these algorithms in greater detail. The optimizer chooses between different possible Index Merge algorithms and other access methods based on cost estimates of the various available options.

- Index Merge Intersection Access Algorithm
- Index Merge Union Access Algorithm
- Index Merge Sort-Union Access Algorithm
- Influencing Index Merge Optimization

Index Merge Intersection Access Algorithm

This access algorithm is applicable when a `WHERE` clause is converted to several range conditions on different keys combined with `AND`, and each condition is one of the following:

- An *`N`*-part expression of this form, where the index has exactly *`N`* parts (that is, all index parts are covered):

```
key_part1 = const1 AND key_part2 = const2 ... AND key_partN = constN
```

- Any range condition over the primary key of an `InnoDB` table.

Examples:

```
```

```
SELECT * FROM innodb_table
  WHERE primary_key < 10 AND key_col1 = 20;

SELECT * FROM tbl_name
  WHERE key1_part1 = 1 AND key1_part2 = 2 AND key2 = 2;
```

The Index Merge intersection algorithm performs simultaneous scans on all used indexes and produces the intersection of row sequences that it receives from the merged index scans.

If all columns used in the query are covered by the used indexes, full table rows are not retrieved (`EXPLAIN` output contains `Using index` in `Extra` field in this case). Here is an example of such a query:

```
SELECT COUNT(*) FROM t1 WHERE key1 = 1 AND key2 = 1;
```

If the used indexes do not cover all columns used in the query, full rows are retrieved only when the range conditions for all used keys are satisfied.

If one of the merged conditions is a condition over the primary key of an `InnoDB` table, it is not used for row retrieval, but is used to filter out rows retrieved using other conditions.

Index Merge Union Access Algorithm

The criteria for this algorithm are similar to those for the Index Merge intersection algorithm. The algorithm is applicable when the table's `WHERE` clause is converted to several range conditions on different keys combined with `OR`, and each condition is one of the following:

- An *`N`*-part expression of this form, where the index has exactly *`N`* parts (that is, all index parts are covered):

  ```
  key_part1 = const1 AND key_part2 = const2 ... AND key_partN = constN
  ```

- Any range condition over a primary key of an `InnoDB` table.

- A condition for which the Index Merge intersection algorithm is applicable.

Examples:

```
SELECT * FROM t1
  WHERE key1 = 1 OR key2 = 2 OR key3 = 3;

SELECT * FROM innodb_table
  WHERE (key1 = 1 AND key2 = 2)
     OR (key3 = 'foo' AND key4 = 'bar') AND key5 = 5;
```

Index Merge Sort-Union Access Algorithm

This access algorithm is applicable when the `WHERE` clause is converted to several range conditions combined by `OR`, but the Index Merge union algorithm is not applicable.

Examples:

```
SELECT * FROM tbl_name
  WHERE key_col1 < 10 OR key_col2 < 20;

SELECT * FROM tbl_name
  WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col = 30;
```

The difference between the sort-union algorithm and the union algorithm is that the sort-union algorithm must first fetch row IDs for all rows and sort them before returning any rows.

Influencing Index Merge Optimization

Use of Index Merge is subject to the value of the `index_merge`, `index_merge_intersection`, `index_merge_union`, and `index_merge_sort_union` flags of the `optimizer_switch` system variable. See Section 8.9.2, “Switchable Optimizations”. By default, all those flags are `on`. To enable only certain algorithms, set `index_merge` to `off`, and enable only such of the others as should be permitted.

In addition to using the `optimizer_switch` system variable to control optimizer use of the Index Merge algorithms session-wide, MySQL supports optimizer hints to influence the optimizer on a per-statement basis. See Section 8.9.3, “Optimizer Hints”.

8.2.1.4 Hash Join Optimization

Beginning with MySQL 8.0.18, MySQL employs a hash join for any query for which each join has an equi-join condition and uses no indexes, such as this one:

```
SELECT *
    FROM t1
    JOIN t2
        ON t1.c1=t2.c1;
```

A hash join is usually faster than and is intended to be used in such cases instead of the block nested loop algorithm (see Block Nested-Loop Join Algorithm) employed in previous versions of MySQL.

In the example just shown and the remaining examples in this section, we assume that the three tables `t1`, `t2`, and `t3` have been created using the following statements:

```
CREATE TABLE t1 (c1 INT, c2 INT);
CREATE TABLE t2 (c1 INT, c2 INT);
CREATE TABLE t3 (c1 INT, c2 INT);
```

You can see that a hash join is being employed by using `EXPLAIN FORMAT=TREE`, like this:

```
mysql> EXPLAIN FORMAT=TREE
    -> SELECT *
    ->     FROM t1
    ->     JOIN t2
    ->         ON t1.c1=t2.c1\G
*************************** 1. row ***************************
EXPLAIN: -> Inner hash join (t2.c1 = t1.c1)  (cost=0.70 rows=1)
    -> Table scan on t2  (cost=0.35 rows=1)
    -> Hash
        -> Table scan on t1  (cost=0.35 rows=1)
```

To see whether hash joins are being used for a given join, you must use `EXPLAIN` with the `FORMAT=TREE` option. `EXPLAIN ANALYZE` also displays information about hash joins used.

The hash join is used for queries involving multiple joins as well, as long as at least one join condition for each pair of tables is an equi-join, like the query shown here:

```
SELECT *
    FROM t1
    JOIN t2
        ON (t1.c1 = t2.c1 AND t1.c2 < t2.c2)
```

```
    JOIN t3
        ON (t2.c1 = t3.c1);
```

In cases like the one just shown, any extra conditions which are not equi-joins are applied as filters after the join is executed. This can be seen here in the output of `EXPLAIN FORMAT=TREE`:

```
mysql> EXPLAIN FORMAT=TREE
    -> SELECT *
    ->     FROM t1
    ->     JOIN t2
    ->         ON (t1.c1 = t2.c1 AND t1.c2 < t2.c2)
    ->     JOIN t3
    ->         ON (t2.c1 = t3.c1)\G
*************************** 1. row ***************************
EXPLAIN: -> Inner hash join (t3.c1 = t1.c1)  (cost=1.05 rows=1)
    -> Table scan on t3  (cost=0.35 rows=1)
    -> Hash
        -> Filter: (t1.c2 < t2.c2)  (cost=0.70 rows=1)
            -> Inner hash join (t2.c1 = t1.c1)  (cost=0.70 rows=1)
                -> Table scan on t2  (cost=0.35 rows=1)
                -> Hash
                    -> Table scan on t1  (cost=0.35 rows=1)
```

As also can be seen from the output just shown, multiple hash joins can be (and are) used for joins having multiple equi-join conditions.

A hash join cannot be used if any pair of joined tables does not have at least one equi-join condition, as can be seen here:

```
mysql> EXPLAIN FORMAT=TREE
    ->     SELECT *
    ->         FROM t1
    ->         JOIN t2
    ->             ON (t1.c1 = t2.c1)
    ->         JOIN t3
    ->             ON (t2.c1 < t3.c1)\G
*************************** 1. row ***************************
EXPLAIN: <not executable by iterator executor>
```

In this case, the slower block nested loop algorithm is employed, as in versions previous to MySQL 8.0.18 where no indexes are available:

```
mysql> EXPLAIN
    ->     SELECT *
    ->         FROM t1
    ->         JOIN t2
    ->             ON (t1.c1 = t2.c1)
    ->         JOIN t3
    ->             ON (t2.c1 < t3.c1)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: NULL
```

```
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: t2
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using where; Using join buffer (Block Nested Loop)
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: t3
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using where; Using join buffer (Block Nested Loop)
```

A hash join is also applied for a Cartesian product—that is, when no join condition is specified, as shown here:

```
mysql> EXPLAIN FORMAT=TREE
    -> SELECT *
    ->     FROM t1
    ->     JOIN t2
    ->     WHERE t1.c2 > 50\G
*************************** 1. row ***************************
EXPLAIN: -> Inner hash join  (cost=0.70 rows=1)
    -> Table scan on t2  (cost=0.35 rows=1)
    -> Hash
        -> Filter: (t1.c2 > 50)  (cost=0.35 rows=1)
            -> Table scan on t1  (cost=0.35 rows=1)
```

By default, MySQL employs hash joins whenever possible. It is possible to control whether hash joins are used in either of two ways:

- On the global or session level by using `hash_join=on` or `hash_join=off` as part of the setting for the `optimizer_switch` server system variable. The default is `hash_join=on`.
- On a case-by-case basis, by using one of the optimizer hints `HASH_JOIN` or `NO_HASH_JOIN` as part of a given join.

Memory usage by hash joins can be controlled using the `join_buffer_size` system variable; a hash join cannot use more memory than this amount. When the memory required for a hash join exceeds the amount available, MySQL handles this by using files on disk. If this happens, you should be aware that the join may not succeed if a hash join cannot fit into memory and it creates more files than set for `open_files_limit`. To avoid such problems, make either of the following changes:

- Increase `join_buffer_size` so that the hash join does not spill over to disk.
- Increase `open_files_limit`.

8.2.1.5 Engine Condition Pushdown Optimization

This optimization improves the efficiency of direct comparisons between a nonindexed column and a constant. In such cases, the condition is “pushed down” to the storage engine for evaluation. This optimization can be used only by the NDB storage engine.

For NDB Cluster, this optimization can eliminate the need to send nonmatching rows over the network between the cluster's data nodes and the MySQL server that issued the query, and can speed up queries where it is used by a factor of 5 to 10 times over cases where condition pushdown could be but is not used.

Suppose that an NDB Cluster table is defined as follows:

```
CREATE TABLE t1 (
    a INT,
    b INT,
    KEY(a)
) ENGINE=NDB;
```

Condition pushdown can be used with queries such as the one shown here, which includes a comparison between a nonindexed column and a constant:

```
SELECT a, b FROM t1 WHERE b = 10;
```

The use of condition pushdown can be seen in the output of EXPLAIN:

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE b = 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 10
        Extra: Using where with pushed condition
```

However, condition pushdown *cannot* be used with the following query:

```
SELECT a,b FROM t1 WHERE a = 10;
```

Condition pushdown is not applicable here because an index exists on column a. (An index access method would be more efficient and so would be chosen in preference to condition pushdown.)

Condition pushdown may also be employed when an indexed column is compared with a constant using a > or < operator:

```
mysql> EXPLAIN SELECT a, b FROM t1 WHERE a < 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: range
possible_keys: a
          key: a
      key_len: 5
          ref: NULL
         rows: 2
```

```
        Extra: Using where with pushed condition
```

Other supported comparisons for condition pushdown include the following:

- *`column`* `[NOT] LIKE` *`pattern`*

  `pattern` must be a string literal containing the pattern to be matched; for syntax, see Section 12.5.1, “String Comparison Functions and Operators”.

- *`column`* `IS [NOT] NULL`

- *`column`* `IN (`*`value_list`*`)`

  Each item in the *`value_list`* must be a constant, literal value.

- *`column`* `BETWEEN` *`constant1`* `AND` *`constant2`*

  `constant1` and *`constant2`* must each be a constant, literal value.

In all of the cases in the preceding list, it is possible for the condition to be converted into the form of one or more direct comparisons between a column and a constant.

Engine condition pushdown is enabled by default. To disable it at server startup, set the `optimizer_switch` system variable. For example, in a `my.cnf` file, use these lines:

```
[mysqld]
optimizer_switch=engine_condition_pushdown=off
```

At runtime, disable condition pushdown like this:

```
SET optimizer_switch='engine_condition_pushdown=off';
```

Limitations. Engine condition pushdown is subject to the following limitations:

- Condition pushdown is supported only by the `NDB` storage engine.
- Prior to NDB 8.0.18, columns could be compared with constants or expressions which evaluate to constant values only. In NDB 8.0.18 and later, columns can be compared with one another as long as they are of exactly the same type, including the same signedness, length, character set, precision, and scale, where these are applicable.
- Columns used in comparisons cannot be of any of the `BLOB` or `TEXT` types. This exclusion extends to `JSON`, `BIT`, and `ENUM` columns as well.
- A string value to be compared with a column must use the same collation as the column.
- Joins are not directly supported; conditions involving multiple tables are pushed separately where possible. Use extended `EXPLAIN` output to determine which conditions are actually pushed down. See Section 8.8.3, “Extended EXPLAIN Output Format”.

Previously, condition pushdown was limited to terms referring to column values from the same table to which the condition was being pushed. Beginning with NDB 8.0.16, column values from tables earlier in the query plan can also be referred to from pushed conditions. This reduces the number of rows which must be handled by the SQL node during join processing. Filtering can be also performed in parallel in the LDM threads, rather than in a single `mysqld` process. This has the potential to improve performance of queries by a significant margin.

Join algorithms cannot be combined with referring columns from previous tables in the following two situations:

1. When any of the referred previous tables are in a join buffer. In this case, each row retrieved from the scan-filtered table is matched against every row in the buffer. This means that there is no single specific row from which column values can be fetched from when generating the scan filter.

2. When the column originates from a child operation in a pushed join. This is because rows referenced from ancestor operations in the join have not yet been retrieved when the scan filter is generated.

8.2.1.6 Index Condition Pushdown Optimization

Index Condition Pushdown (ICP) is an optimization for the case where MySQL retrieves rows from a table using an index. Without ICP, the storage engine traverses the index to locate rows in the base table and returns them to the MySQL server which evaluates the `WHERE` condition for the rows. With ICP enabled, and if parts of the `WHERE` condition can be evaluated by using only columns from the index, the MySQL server pushes this part of the `WHERE` condition down to the storage engine. The storage engine then evaluates the pushed index condition by using the index entry and only if this is satisfied is the row read from the table. ICP can reduce the number of times the storage engine must access the base table and the number of times the MySQL server must access the storage engine.

Applicability of the Index Condition Pushdown optimization is subject to these conditions:

- ICP is used for the `range`, `ref`, `eq_ref`, and `ref_or_null` access methods when there is a need to access full table rows.

- ICP can be used for `InnoDB` and `MyISAM` tables, including partitioned `InnoDB` and `MyISAM` tables.

- For `InnoDB` tables, ICP is used only for secondary indexes. The goal of ICP is to reduce the number of full-row reads and thereby reduce I/O operations. For `InnoDB` clustered indexes, the complete record is already read into the `InnoDB` buffer. Using ICP in this case does not reduce I/O.

- ICP is not supported with secondary indexes created on virtual generated columns. `InnoDB` supports secondary indexes on virtual generated columns.

- Conditions that refer to subqueries cannot be pushed down.

- Conditions that refer to stored functions cannot be pushed down. Storage engines cannot invoke stored functions.

- Triggered conditions cannot be pushed down. (For information about triggered conditions, see Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”.)

To understand how this optimization works, first consider how an index scan proceeds when Index Condition Pushdown is not used:

1. Get the next row, first by reading the index tuple, and then by using the index tuple to locate and read the full table row.

2. Test the part of the `WHERE` condition that applies to this table. Accept or reject the row based on the test result.

Using Index Condition Pushdown, the scan proceeds like this instead:

1. Get the next row's index tuple (but not the full table row).

2. Test the part of the `WHERE` condition that applies to this table and can be checked using only index columns. If the condition is not satisfied, proceed to the index tuple for the next row.

3. If the condition is satisfied, use the index tuple to locate and read the full table row.

4. Test the remaining part of the `WHERE` condition that applies to this table. Accept or reject the row based on the test result.

`EXPLAIN` output shows `Using index condition` in the `Extra` column when Index Condition Pushdown is used. It does not show `Using index` because that does not apply when full table rows must be read.

Suppose that a table contains information about people and their addresses and that the table has an index defined as `INDEX (zipcode, lastname, firstname)`. If we know a person's `zipcode` value but are not sure about the last name, we can search like this:

```
SELECT * FROM people
  WHERE zipcode='95054'
  AND lastname LIKE '%etrunia%'
  AND address LIKE '%Main Street%';
```

MySQL can use the index to scan through people with `zipcode='95054'`. The second part (`lastname LIKE '%etrunia%'`) cannot be used to limit the number of rows that must be scanned, so without Index Condition Pushdown, this query must retrieve full table rows for all people who have `zipcode='95054'`.

With Index Condition Pushdown, MySQL checks the `lastname LIKE '%etrunia%'` part before reading the full table row. This avoids reading full rows corresponding to index tuples that match the `zipcode` condition but not the `lastname` condition.

Index Condition Pushdown is enabled by default. It can be controlled with the `optimizer_switch` system variable by setting the `index_condition_pushdown` flag:

```
SET optimizer_switch = 'index_condition_pushdown=off';
SET optimizer_switch = 'index_condition_pushdown=on';
```

See Section 8.9.2, “Switchable Optimizations”.

8.2.1.7 Nested-Loop Join Algorithms

MySQL executes joins between tables using a nested-loop algorithm or variations on it.

- Nested-Loop Join Algorithm
- Block Nested-Loop Join Algorithm

Nested-Loop Join Algorithm

A simple nested-loop join (NLJ) algorithm reads rows from the first table in a loop one at a time, passing each row to a nested loop that processes the next table in the join. This process is repeated as many times as there remain tables to be joined.

Assume that a join between three tables `t1`, `t2`, and `t3` is to be executed using the following join types:

```
Table   Join Type
t1      range
t2      ref
t3      ALL
```

If a simple NLJ algorithm is used, the join is processed like this:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions, send to client
    }
  }
}
```

Because the NLJ algorithm passes rows one at a time from outer loops to inner loops, it typically reads tables processed in the inner loops many times.

Block Nested-Loop Join Algorithm

A Block Nested-Loop (BNL) join algorithm uses buffering of rows read in outer loops to reduce the number of times that tables in inner loops must be read. For example, if 10 rows are read into a buffer and the buffer is passed to the next inner loop, each row read in the inner loop can be compared against all 10 rows in the buffer. This reduces by an order of magnitude the number of times the inner table must be read.

Prior to MySQL 8.0.18, this algorithm was applied for equi-joins when no indexes could be used; in MySQL 8.0.18 and later, the hash join optimization is employed in such cases. See Section 8.2.1.4, “Hash Join Optimization”.

MySQL join buffering has these characteristics:

- Join buffering can be used when the join is of type `ALL` or `index` (in other words, when no possible keys can be used, and a full scan is done, of either the data or index rows, respectively), or `range`. Use of buffering is also applicable to outer joins, as described in Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”.
- A join buffer is never allocated for the first nonconstant table, even if it would be of type `ALL` or `index`.
- Only columns of interest to a join are stored in its join buffer, not whole rows.
- The `join_buffer_size` system variable determines the size of each join buffer used to process a query.
- One buffer is allocated for each join that can be buffered, so a given query might be processed using multiple join buffers.
- A join buffer is allocated prior to executing the join and freed after the query is done.

For the example join described previously for the NLJ algorithm (without buffering), the join is done as follows using join buffering:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    store used columns from t1, t2 in join buffer
    if buffer is full {
      for each row in t3 {
        for each t1, t2 combination in join buffer {
          if row satisfies join conditions, send to client
        }
      }
      empty join buffer
    }
  }
}
```

```
if buffer is not empty {
  for each row in t3 {
    for each t1, t2 combination in join buffer {
      if row satisfies join conditions, send to client
    }
  }
}
```

If *`S`* is the size of each stored `t1`, `t2` combination in the join buffer and *`C`* is the number of combinations in the buffer, the number of times table `t3` is scanned is:

```
(S * C)/join_buffer_size + 1
```

The number of `t3` scans decreases as the value of `join_buffer_size` increases, up to the point when `join_buffer_size` is large enough to hold all previous row combinations. At that point, no speed is gained by making it larger.

8.2.1.8 Nested Join Optimization

The syntax for expressing joins permits nested joins. The following discussion refers to the join syntax described in Section 13.2.10.2, “JOIN Syntax”.

The syntax of *`table_factor`* is extended in comparison with the SQL Standard. The latter accepts only *`table_reference`*, not a list of them inside a pair of parentheses. This is a conservative extension if we consider each comma in a list of *`table_reference`* items as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

Is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

In MySQL, `CROSS JOIN` is syntactically equivalent to `INNER JOIN`; they can replace each other. In standard SQL, they are not equivalent. `INNER JOIN` is used with an `ON` clause; `CROSS JOIN` is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. Consider this join expression:

```
t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
   ON t1.a=t2.a
```

After removing parentheses and grouping operations to the left, that join expression transforms into this expression:

```
(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3
    ON t2.b=t3.b OR t2.b IS NULL
```

Yet, the two expressions are not equivalent. To see this, suppose that the tables `t1`, `t2`, and `t3` have the following state:

- Table `t1` contains rows `(1)`, `(2)`

- Table `t2` contains row `(1,101)`
- Table `t3` contains row `(101)`

In this case, the first expression returns a result set including the rows `(1,1,101,101)`, `(2,NULL,NULL,NULL)`, whereas the second expression returns the rows `(1,1,101,101)`, `(2,NULL,NULL,101)`:

```
mysql> SELECT *
       FROM t1
            LEFT JOIN
            (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
            ON t1.a=t2.a;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL | NULL |
+------+------+------+------+

mysql> SELECT *
       FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)
            LEFT JOIN t3
            ON t2.b=t3.b OR t2.b IS NULL;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL |  101 |
+------+------+------+------+
```

In the following example, an outer join operation is used together with an inner join operation:

```
t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```

That expression cannot be transformed into the following expression:

```
t1 LEFT JOIN t2 ON t1.a=t2.a, t3
```

For the given table states, the two expressions return different sets of rows:

```
mysql> SELECT *
       FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL | NULL |
+------+------+------+------+

mysql> SELECT *
       FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL |  101 |
+------+------+------+------+
```

Therefore, if we omit parentheses in a join expression with outer join operators, we might change the result set for the original expression.

More exactly, we cannot ignore parentheses in the right operand of the left outer join operation and in the left operand of a right join operation. In other words, we cannot ignore parentheses for the inner table expressions of outer join operations. Parentheses for the other operand (operand for the outer table) can be ignored.

The following expression:

```
(t1,t2) LEFT JOIN t3 ON P(t2.b,t3.b)
```

Is equivalent to this expression for any tables `t1,t2,t3` and any condition `P` over attributes `t2.b` and `t3.b`:

```
t1, t2 LEFT JOIN t3 ON P(t2.b,t3.b)
```

Whenever the order of execution of join operations in a join expression (*`joined_table`*) is not from left to right, we talk about nested joins. Consider the following queries:

```
SELECT * FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a
  WHERE t1.a > 1

SELECT * FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
  WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1
```

Those queries are considered to contain these nested joins:

```
t2 LEFT JOIN t3 ON t2.b=t3.b
t2, t3
```

In the first query, the nested join is formed with a left join operation. In the second query, it is formed with an inner join operation.

In the first query, the parentheses can be omitted: The grammatical structure of the join expression will dictate the same order of execution for join operations. For the second query, the parentheses cannot be omitted, although the join expression here can be interpreted unambiguously without them. In our extended syntax, the parentheses in `(t2, t3)` of the second query are required, although theoretically the query could be parsed without them: We still would have unambiguous syntactical structure for the query because `LEFT JOIN` and `ON` play the role of the left and right delimiters for the expression `(t2,t3)`.

The preceding examples demonstrate these points:

- For join expressions involving only inner joins (and not outer joins), parentheses can be removed and joins evaluated left to right. In fact, tables can be evaluated in any order.
- The same is not true, in general, for outer joins or for outer joins mixed with inner joins. Removal of parentheses may change the result.

Queries with nested outer joins are executed in the same pipeline manner as queries with inner joins. More exactly, a variation of the nested-loop join algorithm is exploited. Recall the algorithm by which the nested-loop join executes a query (see Section 8.2.1.7, “Nested-Loop Join Algorithms”). Suppose that a join query over 3 tables `T1,T2,T3` has this form:

```
SELECT * FROM T1 INNER JOIN T2 ON P1(T1,T2)
                 INNER JOIN T3 ON P2(T2,T3)
```

```
  WHERE P(T1,T2,T3)
```

Here, `P1(T1,T2)` and `P2(T3,T3)` are some join conditions (on expressions), whereas `P(T1,T2,T3)` is a condition over columns of tables `T1,T2,T3`.

The nested-loop join algorithm would execute this query in the following manner:

```
FOR each row t1 in T1 {
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

The notation `t1||t2||t3` indicates a row constructed by concatenating the columns of rows `t1`, `t2`, and `t3`. In some of the following examples, `NULL` where a table name appears means a row in which `NULL` is used for each column of that table. For example, `t1||t2||NULL` indicates a row constructed by concatenating the columns of rows `t1` and `t2`, and `NULL` for each column of `t3`. Such a row is said to be `NULL`-complemented.

Now consider a query with nested outer joins:

```
SELECT * FROM T1 LEFT JOIN
             (T2 LEFT JOIN T3 ON P2(T2,T3))
             ON P1(T1,T2)
  WHERE P(T1,T2,T3)
```

For this query, modify the nested-loop pattern to obtain:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF P(t1,t2,NULL) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

In general, for any nested loop for the first inner table in an outer join operation, a flag is introduced that is turned off before the loop and is checked after the loop. The flag is turned on when for the current row from the outer table a match from the table representing the inner operand is found. If at the end of the

loop cycle the flag is still off, no match has been found for the current row of the outer table. In this case, the row is complemented by NULL values for the columns of the inner tables. The result row is passed to the final check for the output or into the next nested loop, but only if the row satisfies the join condition of all embedded outer joins.

In the example, the outer join table expressed by the following expression is embedded:

```
(T2 LEFT JOIN T3 ON P2(T2,T3))
```

For the query with inner joins, the optimizer could choose a different order of nested loops, such as this one:

```
FOR each row t3 in T3 {
  FOR each row t2 in T2 such that P2(t2,t3) {
    FOR each row t1 in T1 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

For queries with outer joins, the optimizer can choose only such an order where loops for outer tables precede loops for inner tables. Thus, for our query with outer joins, only one nesting order is possible. For the following query, the optimizer evaluates two different nestings. In both nestings, T1 must be processed in the outer loop because it is used in an outer join. T2 and T3 are used in an inner join, so that join must be processed in the inner loop. However, because the join is an inner join, T2 and T3 can be processed in either order.

```
SELECT * T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)
  WHERE P(T1,T2,T3)
```

One nesting evaluates T2, then T3:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t1,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

The other nesting evaluates T3, then T2:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t3 in T3 such that P2(t1,t3) {
    FOR each row t2 in T2 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
```

```
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

When discussing the nested-loop algorithm for inner joins, we omitted some details whose impact on the performance of query execution may be huge. We did not mention so-called “pushed-down” conditions. Suppose that our `WHERE` condition `P(T1,T2,T3)` can be represented by a conjunctive formula:

```
P(T1,T2,T2) = C1(T1) AND C2(T2) AND C3(T3).
```

In this case, MySQL actually uses the following nested-loop algorithm for the execution of the query with inner joins:

```
FOR each row t1 in T1 such that C1(t1) {
  FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2)  {
    FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

You see that each of the conjuncts `C1(T1)`, `C2(T2)`, `C3(T3)` are pushed out of the most inner loop to the most outer loop where it can be evaluated. If `C1(T1)` is a very restrictive condition, this condition pushdown may greatly reduce the number of rows from table `T1` passed to the inner loops. As a result, the execution time for the query may improve immensely.

For a query with outer joins, the `WHERE` condition is to be checked only after it has been found that the current row from the outer table has a match in the inner tables. Thus, the optimization of pushing conditions out of the inner nested loops cannot be applied directly to queries with outer joins. Here we must introduce conditional pushed-down predicates guarded by the flags that are turned on when a match has been encountered.

Recall this example with outer joins:

```
P(T1,T2,T3)=C1(T1) AND C(T2) AND C3(T3)
```

For that example, the nested-loop algorithm using guarded pushed-down conditions looks like this:

```
FOR each row t1 in T1 such that C1(t1) {
  BOOL f1:=FALSE;
  FOR each row t2 in T2
      such that P1(t1,t2) AND (f1?C2(t2):TRUE) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3
        such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {
      IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {
        t:=t1||t2||t3; OUTPUT t;
      }
```

```
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1 && P(t1,NULL,NULL)) {
      t:=t1||NULL||NULL; OUTPUT t;
  }
}
```

In general, pushed-down predicates can be extracted from join conditions such as `P1(T1,T2)` and `P(T2,T3)`. In this case, a pushed-down predicate is guarded also by a flag that prevents checking the predicate for the `NULL`-complemented row generated by the corresponding outer join operation.

Access by key from one inner table to another in the same nested join is prohibited if it is induced by a predicate from the `WHERE` condition.

8.2.1.9 Outer Join Optimization

Outer joins include `LEFT JOIN` and `RIGHT JOIN`.

MySQL implements an *`A LEFT JOIN B join_specification`* as follows:

- Table *`B`* is set to depend on table *`A`* and all tables on which *`A`* depends.
- Table *`A`* is set to depend on all tables (except *`B`*) that are used in the `LEFT JOIN` condition.
- The `LEFT JOIN` condition is used to decide how to retrieve rows from table *`B`*. (In other words, any condition in the `WHERE` clause is not used.)
- All standard join optimizations are performed, with the exception that a table is always read after all tables on which it depends. If there is a circular dependency, an error occurs.
- All standard `WHERE` optimizations are performed.
- If there is a row in *`A`* that matches the `WHERE` clause, but there is no row in *`B`* that matches the `ON` condition, an extra *`B`* row is generated with all columns set to `NULL`.
- If you use `LEFT JOIN` to find rows that do not exist in some table and you have the following test: *`col_name`* `IS NULL` in the `WHERE` part, where *`col_name`* is a column that is declared as `NOT NULL`, MySQL stops searching for more rows (for a particular key combination) after it has found one row that matches the `LEFT JOIN` condition.

The `RIGHT JOIN` implementation is analogous to that of `LEFT JOIN` with the table roles reversed. Right joins are converted to equivalent left joins, as described in Section 8.2.1.10, “Outer Join Simplification”.

For a `LEFT JOIN`, if the `WHERE` condition is always false for the generated `NULL` row, the `LEFT JOIN` is changed to an inner join. For example, the `WHERE` clause would be false in the following query if `t2.column1` were `NULL`:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

Therefore, it is safe to convert the query to an inner join:

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

In MySQL 8.0.14 and later, trivial `WHERE` conditions arising from constant literal expressions are removed during preparation, rather than at a later stage in optimization, by which time joins have already been simplified. Earlier removal of trivial conditions allows the optimizer to convert outer joins to inner joins; this can result in improved plans for queries with outer joins containing trivial conditions in the `WHERE` clause, such as this one:

```
SELECT * FROM t1 LEFT JOIN t2 ON condition_1 WHERE condition_2 OR 0 = 1
```

The optimizer now sees during preparation that 0 = 1 is always false, making `OR 0 = 1` redundant, and removes it, leaving this:

```
SELECT * FROM t1 LEFT JOIN t2 ON condition_1 where condition_2
```

Now the optimizer can rewrite the query as an inner join, like this:

```
SELECT * FROM t1 JOIN t2 WHERE condition_1 AND condition_2
```

Now the optimizer can use table `t2` before table `t1` if doing so would result in a better query plan. To provide a hint about the table join order, use optimizer hints; see Section 8.9.3, “Optimizer Hints”. Alternatively, use `STRAIGHT_JOIN`; see Section 13.2.10, “SELECT Syntax”. However, `STRAIGHT_JOIN` may prevent indexes from being used because it disables semijoin transformations; see Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”.

8.2.1.10 Outer Join Simplification

Table expressions in the `FROM` clause of a query are simplified in many cases.

At the parser stage, queries with right outer join operations are converted to equivalent queries containing only left join operations. In the general case, the conversion is performed such that this right join:

```
(T1, ...) RIGHT JOIN (T2, ...) ON P(T1, ..., T2, ...)
```

Becomes this equivalent left join:

```
(T2, ...) LEFT JOIN (T1, ...) ON P(T1, ..., T2, ...)
```

All inner join expressions of the form `T1 INNER JOIN T2 ON P(T1,T2)` are replaced by the list `T1,T2`, `P(T1,T2)` being joined as a conjunct to the `WHERE` condition (or to the join condition of the embedding join, if there is any).

When the optimizer evaluates plans for outer join operations, it takes into consideration only plans where, for each such operation, the outer tables are accessed before the inner tables. The optimizer choices are limited because only such plans enable outer joins to be executed using the nested-loop algorithm.

Consider a query of this form, where `R(T2)` greatly narrows the number of matching rows from table `T2`:

```
SELECT * T1 LEFT JOIN T2 ON P1(T1,T2)
  WHERE P(T1,T2) AND R(T2)
```

If the query is executed as written, the optimizer has no choice but to access the less-restricted table `T1` before the more-restricted table `T2`, which may produce a very inefficient execution plan.

Instead, MySQL converts the query to a query with no outer join operation if the `WHERE` condition is null-rejected. (That is, it converts the outer join to an inner join.) A condition is said to be null-rejected for an outer join operation if it evaluates to `FALSE` or `UNKNOWN` for any `NULL`-complemented row generated for the operation.

Thus, for this outer join:

```
T1 LEFT JOIN T2 ON T1.A=T2.A
```

Conditions such as these are null-rejected because they cannot be true for any `NULL`-complemented row (with `T2` columns set to `NULL`):

```
T2.B IS NOT NULL
T2.B > 3
T2.C <= T1.C
T2.B < 2 OR T2.C > 1
```

Conditions such as these are not null-rejected because they might be true for a `NULL`-complemented row:

```
T2.B IS NULL
T1.B < 3 OR T2.B IS NOT NULL
T1.B < 3 OR T2.B > 3
```

The general rules for checking whether a condition is null-rejected for an outer join operation are simple:

- It is of the form `A IS NOT NULL`, where `A` is an attribute of any of the inner tables
- It is a predicate containing a reference to an inner table that evaluates to `UNKNOWN` when one of its arguments is `NULL`
- It is a conjunction containing a null-rejected condition as a conjunct
- It is a disjunction of null-rejected conditions

A condition can be null-rejected for one outer join operation in a query and not null-rejected for another. In this query, the `WHERE` condition is null-rejected for the second outer join operation but is not null-rejected for the first one:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 LEFT JOIN T3 ON T3.B=T1.B
  WHERE T3.C > 0
```

If the `WHERE` condition is null-rejected for an outer join operation in a query, the outer join operation is replaced by an inner join operation.

For example, in the preceding query, the second outer join is null-rejected and can be replaced by an inner join:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 INNER JOIN T3 ON T3.B=T1.B
  WHERE T3.C > 0
```

For the original query, the optimizer evaluates only plans compatible with the single table-access order `T1,T2,T3`. For the rewritten query, it additionally considers the access order `T3,T1,T2`.

A conversion of one outer join operation may trigger a conversion of another. Thus, the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 LEFT JOIN T3 ON T3.B=T2.B
  WHERE T3.C > 0
```

Is first converted to the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 INNER JOIN T3 ON T3.B=T2.B
  WHERE T3.C > 0
```

Which is equivalent to the query:

```
SELECT * FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3
  WHERE T3.C > 0 AND T3.B=T2.B
```

The remaining outer join operation can also be replaced by an inner join because the condition `T3.B=T2.B` is null-rejected. This results in a query with no outer joins at all:

```
SELECT * FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3
  WHERE T3.C > 0 AND T3.B=T2.B
```

Sometimes the optimizer succeeds in replacing an embedded outer join operation, but cannot convert the embedding outer join. The following query:

```
SELECT * FROM T1 LEFT JOIN
              (T2 LEFT JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A
  WHERE T3.C > 0
```

Is converted to:

```
SELECT * FROM T1 LEFT JOIN
              (T2 INNER JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A
  WHERE T3.C > 0
```

That can be rewritten only to the form still containing the embedding outer join operation:

```
SELECT * FROM T1 LEFT JOIN
              (T2,T3)
              ON (T2.A=T1.A AND T3.B=T2.B)
  WHERE T3.C > 0
```

Any attempt to convert an embedded outer join operation in a query must take into account the join condition for the embedding outer join together with the `WHERE` condition. In this query, the `WHERE` condition is not null-rejected for the embedded outer join, but the join condition of the embedding outer join `T2.A=T1.A AND T3.C=T1.C` is null-rejected:

```
SELECT * FROM T1 LEFT JOIN
              (T2 LEFT JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A AND T3.C=T1.C
  WHERE T3.D > 0 OR T1.D > 0
```

Consequently, the query can be converted to:

```
SELECT * FROM T1 LEFT JOIN
             (T2, T3)
             ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B
  WHERE T3.D > 0 OR T1.D > 0
```

8.2.1.11 Multi-Range Read Optimization

Reading rows using a range scan on a secondary index can result in many random disk accesses to the base table when the table is large and not stored in the storage engine's cache. With the Disk-Sweep Multi-Range Read (MRR) optimization, MySQL tries to reduce the number of random disk access for range scans by first scanning the index only and collecting the keys for the relevant rows. Then the keys are sorted and finally the rows are retrieved from the base table using the order of the primary key. The motivation for Disk-sweep MRR is to reduce the number of random disk accesses and instead achieve a more sequential scan of the base table data.

The Multi-Range Read optimization provides these benefits:

- MRR enables data rows to be accessed sequentially rather than in random order, based on index tuples. The server obtains a set of index tuples that satisfy the query conditions, sorts them according to data row ID order, and uses the sorted tuples to retrieve data rows in order. This makes data access more efficient and less expensive.
- MRR enables batch processing of requests for key access for operations that require access to data rows through index tuples, such as range index scans and equi-joins that use an index for the join attribute. MRR iterates over a sequence of index ranges to obtain qualifying index tuples. As these results accumulate, they are used to access the corresponding data rows. It is not necessary to acquire all index tuples before starting to read data rows.

The MRR optimization is not supported with secondary indexes created on virtual generated columns. `InnoDB` supports secondary indexes on virtual generated columns.

The following scenarios illustrate when MRR optimization can be advantageous:

Scenario A: MRR can be used for `InnoDB` and `MyISAM` tables for index range scans and equi-join operations.

1. A portion of the index tuples are accumulated in a buffer.
2. The tuples in the buffer are sorted by their data row ID.
3. Data rows are accessed according to the sorted index tuple sequence.

Scenario B: MRR can be used for `NDB` tables for multiple-range index scans or when performing an equi-join by an attribute.

1. A portion of ranges, possibly single-key ranges, is accumulated in a buffer on the central node where the query is submitted.
2. The ranges are sent to the execution nodes that access data rows.
3. The accessed rows are packed into packages and sent back to the central node.
4. The received packages with data rows are placed in a buffer.
5. Data rows are read from the buffer.

When MRR is used, the `Extra` column in `EXPLAIN` output shows `Using MRR`.

`InnoDB` and `MyISAM` do not use MRR if full table rows need not be accessed to produce the query result. This is the case if results can be produced entirely on the basis on information in the index tuples (through a covering index); MRR provides no benefit.

Two `optimizer_switch` system variable flags provide an interface to the use of MRR optimization. The `mrr` flag controls whether MRR is enabled. If `mrr` is enabled (`on`), the `mrr_cost_based` flag controls whether the optimizer attempts to make a cost-based choice between using and not using MRR (`on`) or uses MRR whenever possible (`off`). By default, `mrr` is `on` and `mrr_cost_based` is `on`. See Section 8.9.2, “Switchable Optimizations”.

For MRR, a storage engine uses the value of the `read_rnd_buffer_size` system variable as a guideline for how much memory it can allocate for its buffer. The engine uses up to `read_rnd_buffer_size` bytes and determines the number of ranges to process in a single pass.

8.2.1.12 Block Nested-Loop and Batched Key Access Joins

In MySQL, a Batched Key Access (BKA) Join algorithm is available that uses both index access to the joined table and a join buffer. The BKA algorithm supports inner join, outer join, and semijoin operations, including nested outer joins. Benefits of BKA include improved join performance due to more efficient table scanning. Also, the Block Nested-Loop (BNL) Join algorithm previously used only for inner joins is extended and can be employed for outer join and semijoin operations, including nested outer joins.

The following sections discuss the join buffer management that underlies the extension of the original BNL algorithm, the extended BNL algorithm, and the BKA algorithm. For information about semijoin strategies, see Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”

- Join Buffer Management for Block Nested-Loop and Batched Key Access Algorithms
- Block Nested-Loop Algorithm for Outer Joins and Semijoins
- Batched Key Access Joins
- Optimizer Hints for Block Nested-Loop and Batched Key Access Algorithms

Join Buffer Management for Block Nested-Loop and Batched Key Access Algorithms

MySQL can employ join buffers to execute not only inner joins without index access to the inner table, but also outer joins and semijoins that appear after subquery flattening. Moreover, a join buffer can be effectively used when there is an index access to the inner table.

The join buffer management code slightly more efficiently utilizes join buffer space when storing the values of the interesting row columns: No additional bytes are allocated in buffers for a row column if its value is `NULL`, and the minimum number of bytes is allocated for any value of the `VARCHAR` type.

The code supports two types of buffers, regular and incremental. Suppose that join buffer `B1` is employed to join tables `t1` and `t2` and the result of this operation is joined with table `t3` using join buffer `B2`:

- A regular join buffer contains columns from each join operand. If `B2` is a regular join buffer, each row *`r`* put into `B2` is composed of the columns of a row *`r1`* from `B1` and the interesting columns of a matching row *`r2`* from table `t3`.
- An incremental join buffer contains only columns from rows of the table produced by the second join operand. That is, it is incremental to a row from the first operand buffer. If `B2` is an incremental join buffer, it contains the interesting columns of the row *`r2`* together with a link to the row *`r1`* from `B1`.

Incremental join buffers are always incremental relative to a join buffer from an earlier join operation, so the buffer from the first join operation is always a regular buffer. In the example just given, the buffer `B1` used to join tables `t1` and `t2` must be a regular buffer.

Each row of the incremental buffer used for a join operation contains only the interesting columns of a row from the table to be joined. These columns are augmented with a reference to the interesting columns of the matched row from the table produced by the first join operand. Several rows in the incremental buffer can refer to the same row *r* whose columns are stored in the previous join buffers insofar as all these rows match row *r*.

Incremental buffers enable less frequent copying of columns from buffers used for previous join operations. This provides a savings in buffer space because in the general case a row produced by the first join operand can be matched by several rows produced by the second join operand. It is unnecessary to make several copies of a row from the first operand. Incremental buffers also provide a savings in processing time due to the reduction in copying time.

The `block_nested_loop` and `batched_key_access` flags of the `optimizer_switch` system variable control how the optimizer uses the Block Nested-Loop and Batched Key Access join algorithms. By default, `block_nested_loop` is `on` and `batched_key_access` is `off`. See Section 8.9.2, “Switchable Optimizations”. Optimizer hints may also be applied; see Optimizer Hints for Block Nested-Loop and Batched Key Access Algorithms.

For information about semijoin strategies, see Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”

Block Nested-Loop Algorithm for Outer Joins and Semijoins

The original implementation of the MySQL BNL algorithm is extended to support outer join and semijoin operations.

When these operations are executed with a join buffer, each row put into the buffer is supplied with a match flag.

If an outer join operation is executed using a join buffer, each row of the table produced by the second operand is checked for a match against each row in the join buffer. When a match is found, a new extended row is formed (the original row plus columns from the second operand) and sent for further extensions by the remaining join operations. In addition, the match flag of the matched row in the buffer is enabled. After all rows of the table to be joined have been examined, the join buffer is scanned. Each row from the buffer that does not have its match flag enabled is extended by `NULL` complements (`NULL` values for each column in the second operand) and sent for further extensions by the remaining join operations.

The `block_nested_loop` flag of the `optimizer_switch` system variable controls how the optimizer uses the Block Nested-Loop algorithm. By default, `block_nested_loop` is `on`. See Section 8.9.2, “Switchable Optimizations”. Optimizer hints may also be applied; see Optimizer Hints for Block Nested-Loop and Batched Key Access Algorithms.

In `EXPLAIN` output, use of BNL for a table is signified when the `Extra` value contains `Using join buffer (Block Nested Loop)` and the `type` value is `ALL`, `index`, or `range`.

For information about semijoin strategies, see Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”

Batched Key Access Joins

MySQL implements a method of joining tables called the Batched Key Access (BKA) join algorithm. BKA can be applied when there is an index access to the table produced by the second join operand. Like the

BNL join algorithm, the BKA join algorithm employs a join buffer to accumulate the interesting columns of the rows produced by the first operand of the join operation. Then the BKA algorithm builds keys to access the table to be joined for all rows in the buffer and submits these keys in a batch to the database engine for index lookups. The keys are submitted to the engine through the Multi-Range Read (MRR) interface (see Section 8.2.1.11, “Multi-Range Read Optimization”). After submission of the keys, the MRR engine functions perform lookups in the index in an optimal way, fetching the rows of the joined table found by these keys, and starts feeding the BKA join algorithm with matching rows. Each matching row is coupled with a reference to a row in the join buffer.

When BKA is used, the value of `join_buffer_size` defines how large the batch of keys is in each request to the storage engine. The larger the buffer, the more sequential access will be to the right hand table of a join operation, which can significantly improve performance.

For BKA to be used, the `batched_key_access` flag of the `optimizer_switch` system variable must be set to `on`. BKA uses MRR, so the `mrr` flag must also be `on`. Currently, the cost estimation for MRR is too pessimistic. Hence, it is also necessary for `mrr_cost_based` to be `off` for BKA to be used. The following setting enables BKA:

```
mysql> SET optimizer_switch='mrr=on,mrr_cost_based=off,batched_key_access=on';
```

There are two scenarios by which MRR functions execute:

- The first scenario is used for conventional disk-based storage engines such as `InnoDB` and `MyISAM`. For these engines, usually the keys for all rows from the join buffer are submitted to the MRR interface at once. Engine-specific MRR functions perform index lookups for the submitted keys, get row IDs (or primary keys) from them, and then fetch rows for all these selected row IDs one by one by request from BKA algorithm. Every row is returned with an association reference that enables access to the matched row in the join buffer. The rows are fetched by the MRR functions in an optimal way: They are fetched in the row ID (primary key) order. This improves performance because reads are in disk order rather than random order.

- The second scenario is used for remote storage engines such as `NDB`. A package of keys for a portion of rows from the join buffer, together with their associations, is sent by a MySQL Server (SQL node) to MySQL Cluster data nodes. In return, the SQL node receives a package (or several packages) of matching rows coupled with corresponding associations. The BKA join algorithm takes these rows and builds new joined rows. Then a new set of keys is sent to the data nodes and the rows from the returned packages are used to build new joined rows. The process continues until the last keys from the join buffer are sent to the data nodes, and the SQL node has received and joined all rows matching these keys. This improves performance because fewer key-bearing packages sent by the SQL node to the data nodes means fewer round trips between it and the data nodes to perform the join operation.

With the first scenario, a portion of the join buffer is reserved to store row IDs (primary keys) selected by index lookups and passed as a parameter to the MRR functions.

There is no special buffer to store keys built for rows from the join buffer. Instead, a function that builds the key for the next row in the buffer is passed as a parameter to the MRR functions.

In `EXPLAIN` output, use of BKA for a table is signified when the `Extra` value contains `Using join buffer (Batched Key Access)` and the `type` value is `ref` or `eq_ref`.

Optimizer Hints for Block Nested-Loop and Batched Key Access Algorithms

In addition to using the `optimizer_switch` system variable to control optimizer use of the BNL and BKA algorithms session-wide, MySQL supports optimizer hints to influence the optimizer on a per-statement basis. See Section 8.9.3, “Optimizer Hints”.

To use a BNL or BKA hint to enable join buffering for any inner table of an outer join, join buffering must be enabled for all inner tables of the outer join.

8.2.1.13 Condition Filtering

In join processing, prefix rows are those rows passed from one table in a join to the next. In general, the optimizer attempts to put tables with low prefix counts early in the join order to keep the number of row combinations from increasing rapidly. To the extent that the optimizer can use information about conditions on rows selected from one table and passed to the next, the more accurately it can compute row estimates and choose the best execution plan.

Without condition filtering, the prefix row count for a table is based on the estimated number of rows selected by the `WHERE` clause according to whichever access method the optimizer chooses. Condition filtering enables the optimizer to use other relevant conditions in the `WHERE` clause not taken into account by the access method, and thus improve its prefix row count estimates. For example, even though there might be an index-based access method that can be used to select rows from the current table in a join, there might also be additional conditions for the table in the `WHERE` clause that can filter (further restrict) the estimate for qualifying rows passed to the next table.

A condition contributes to the filtering estimate only if:

- It refers to the current table.
- It depends on a constant value or values from earlier tables in the join sequence.
- It was not already taken into account by the access method.

In `EXPLAIN` output, the `rows` column indicates the row estimate for the chosen access method, and the `filtered` column reflects the effect of condition filtering. `filtered` values are expressed as percentages. The maximum value is 100, which means no filtering of rows occurred. Values decreasing from 100 indicate increasing amounts of filtering.

The prefix row count (the number of rows estimated to be passed from the current table in a join to the next) is the product of the `rows` and `filtered` values. That is, the prefix row count is the estimated row count, reduced by the estimated filtering effect. For example, if `rows` is 1000 and `filtered` is 20%, condition filtering reduces the estimated row count of 1000 to a prefix row count of 1000 × 20% = 1000 × .2 = 200.

Consider the following query:

```
SELECT *
  FROM employee JOIN department ON employee.dept_no = department.dept_no
  WHERE employee.first_name = 'John'
  AND employee.hire_date BETWEEN '2018-01-01' AND '2018-06-01';
```

Suppose that the data set has these characteristics:

- The `employee` table has 1024 rows.
- The `department` table has 12 rows.
- Both tables have an index on `dept_no`.
- The `employee` table has an index on `first_name`.
- 8 rows satisfy this condition on `employee.first_name`:

```
employee.first_name = 'John'
```

- 150 rows satisfy this condition on `employee.hire_date`:

```
employee.hire_date BETWEEN '2018-01-01' AND '2018-06-01'
```

- 1 row satisfies both conditions:

```
employee.first_name = 'John'
AND employee.hire_date BETWEEN '2018-01-01' AND '2018-06-01'
```

Without condition filtering, `EXPLAIN` produces output like this:

```
+----+------------+--------+------------------+---------+---------+------+----------+
| id | table      | type   | possible_keys    | key     | ref     | rows | filtered |
+----+------------+--------+------------------+---------+---------+------+----------+
| 1  | employee   | ref    | name,h_date,dept | name    | const   | 8    | 100.00   |
| 1  | department | eq_ref | PRIMARY          | PRIMARY | dept_no | 1    | 100.00   |
+----+------------+--------+------------------+---------+---------+------+----------+
```

For `employee`, the access method on the `name` index picks up the 8 rows that match a name of `'John'`. No filtering is done (`filtered` is 100%), so all rows are prefix rows for the next table: The prefix row count is `rows` × `filtered` = 8 × 100% = 8.

With condition filtering, the optimizer additionally takes into account conditions from the `WHERE` clause not taken into account by the access method. In this case, the optimizer uses heuristics to estimate a filtering effect of 16.31% for the `BETWEEN` condition on `employee.hire_date`. As a result, `EXPLAIN` produces output like this:

```
+----+------------+--------+------------------+---------+---------+------+----------+
| id | table      | type   | possible_keys    | key     | ref     | rows | filtered |
+----+------------+--------+------------------+---------+---------+------+----------+
| 1  | employee   | ref    | name,h_date,dept | name    | const   | 8    | 16.31    |
| 1  | department | eq_ref | PRIMARY          | PRIMARY | dept_no | 1    | 100.00   |
+----+------------+--------+------------------+---------+---------+------+----------+
```

Now the prefix row count is `rows` × `filtered` = 8 × 16.31% = 1.3, which more closely reflects actual data set.

Normally, the optimizer does not calculate the condition filtering effect (prefix row count reduction) for the last joined table because there is no next table to pass rows to. An exception occurs for `EXPLAIN`: To provide more information, the filtering effect is calculated for all joined tables, including the last one.

To control whether the optimizer considers additional filtering conditions, use the `condition_fanout_filter` flag of the `optimizer_switch` system variable (see Section 8.9.2, “Switchable Optimizations”). This flag is enabled by default but can be disabled to suppress condition filtering (for example, if a particular query is found to yield better performance without it).

If the optimizer overestimates the effect of condition filtering, performance may be worse than if condition filtering is not used. In such cases, these techniques may help:

- If a column is not indexed, index it so that the optimizer has some information about the distribution of column values and can improve its row estimates.
- Similarly, if no column histogram information is available, generate a histogram (see Section 8.9.6, “Optimizer Statistics”).

- Change the join order. Ways to accomplish this include join-order optimizer hints (see Section 8.9.3, “Optimizer Hints”), `STRAIGHT_JOIN` immediately following the `SELECT`, and the `STRAIGHT_JOIN` join operator.
- Disable condition filtering for the session:

```
SET optimizer_switch = 'condition_fanout_filter=off';
```

Or, for a given query, using an optimizer hint:

```
SELECT /*+ SET_VAR(optimizer_switch = 'condition_fanout_filter=off') */ ...
```

8.2.1.14 Constant-Folding Optimization

Comparisons between constants and column values in which the constant value is out of range or of the wrong type with respect to the column type are now handled once during query optimization rather row-by-row than during execution. The comparisons that can be treated in this manner are `>`, `>=`, `<`, `<=`, `<>`/`!=`, `=`, and `<=>`.

Consider the table created by the following statement:

```
CREATE TABLE t (c TINYINT UNSIGNED NOT NULL);
```

The `WHERE` condition in the query `SELECT * FROM t WHERE c < 256` contains the integral constant 256 which is out of range for a `TINYINT UNSIGNED` column. Previously, this was handled by treating both operands as the larger type, but now, since any allowed value for `c` is less than the constant, the `WHERE` expression can instead be folded as `WHERE 1`, so that the query is rewritten as `SELECT * FROM t WHERE 1`.

This makes it possible for the optimizer to remove the `WHERE` expression altogether. If the column `c` were nullable (that is, defined only as `TINYINT UNSIGNED`) the query would be rewritten like this:

```
SELECT * FROM t WHERE ti IS NOT NULL
```

Folding is performed for constants compared to supported MySQL column types as follows:

- **Integer column type.** Integer types are compared with constants of the following types as described here:
  - **Integer value.** If the constant is out of range for the column type, the comparison is folded to `1` or `IS NOT NULL`, as already shown.

    If the constant is a range boundary, the comparison is folded to `=`. For example (using the same table as already defined):

```
mysql> EXPLAIN SELECT * FROM t WHERE c >= 255;
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
```

```
         rows: 5
     filtered: 20.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select `test`.`t`.`ti` AS `ti` from `test`.`t` where (`test`.`t`.`ti` = 255)
1 row in set (0.00 sec)
```

  - **Floating- or fixed-point value.** If the constant is one of the decimal types (such as `DECIMAL`, `REAL`, `DOUBLE`, or `FLOAT`) and has a nonzero decimal portion, it cannot be equal; fold accordingly. For other comparisons, round up or down to an integer value according to the sign, then perform a range check and handle as already described for integer-integer comparisons.

    A `REAL` value that is too small to be represented as `DECIMAL` is rounded to .01 or -.01 depending on the sign, then handled as a `DECIMAL`.

  - **String types.** Try to interpret the string value as an integer type, then handle the comparison as between integer values. If this fails, attempt to handle the value as a `REAL`.

- **DECIMAL or REAL column.** Decimal types are compared with constants of the following types as described here:

  - **Integer value.** Perform a range check against the column value's integer part. If no folding results, convert the constant to `DECIMAL` with the same number of decimal places as the column value, then check it as a `DECIMAL` (see next).

  - **DECIMAL or REAL value.** Check for overflow (that is, whether the constant has more digits in its integer part than allowed for the column's decimal type). If so, fold.

    If the constant has more significant fractional digits than column's type, truncate the constant. If the comparison operator is `=` or `<>`, fold. If the operator is `>=` or `<=`, adjust the operator due to truncation. For example, if column's type is `DECIMAL(3,1)`, `SELECT * FROM t WHERE f >= 10.13` becomes `SELECT * FROM t WHERE f > 10.1`.

    If the constant has fewer decimal digits than the column's type, convert it to a constant with same number of digits. For underflow of a `REAL` value (that is, too few fractional digits to represent it), convert the constant to decimal 0.

  - **String value.** If the value can be interpreted as an integer type, handle it as such. Otherwise, try to handle it as `REAL`.

- **FLOAT or DOUBLE column.** `FLOAT(m,n)` or `DOUBLE(m,n)` values compared with constants are handled as follows:

  If the value overflows the range of the column, fold.

  If the value has more than *n* decimals, truncate, compensating during folding. For `=` and `<>` comparisons, fold to `TRUE`, `FALSE`, or `IS [NOT] NULL` as described previously; for other operators, adjust the operator.

  If the value has more than `m` integer digits, fold.

Limitations. This optimization cannot be used in the following cases:

1. With comparisons using `BETWEEN` or `IN`.

2. With `BIT` columns or columns using date or time types.

3. During the preparation phase for a prepared statement, although it can be applied during the optimization phase when the prepared statement is actually executed. This due to the fact that, during statement preparation, the value of the constant is not yet known.

8.2.1.15 IS NULL Optimization

MySQL can perform the same optimization on *`col_name`* `IS NULL` that it can use for *`col_name`* = *`constant_value`*. For example, MySQL can use indexes and ranges to search for `NULL` with `IS NULL`.

Examples:

```
SELECT * FROM tbl_name WHERE key_col IS NULL;

SELECT * FROM tbl_name WHERE key_col <=> NULL;

SELECT * FROM tbl_name
  WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

If a `WHERE` clause includes a *`col_name`* `IS NULL` condition for a column that is declared as `NOT NULL`, that expression is optimized away. This optimization does not occur in cases when the column might produce `NULL` anyway (for example, if it comes from a table on the right side of a `LEFT JOIN`).

MySQL can also optimize the combination *`col_name = expr`* `OR` *`col_name`* `IS NULL`, a form that is common in resolved subqueries. `EXPLAIN` shows `ref_or_null` when this optimization is used.

This optimization can handle one `IS NULL` for any key part.

Some examples of queries that are optimized, assuming that there is an index on columns `a` and `b` of table `t2`:

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;

SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;

SELECT * FROM t1, t2
  WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT * FROM t1, t2
  WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);

SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)
  OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

`ref_or_null` works by first doing a read on the reference key, and then a separate search for rows with a `NULL` key value.

The optimization can handle only one `IS NULL` level. In the following query, MySQL uses key lookups only on the expression `(t1.a=t2.a AND t2.a IS NULL)` and is not able to use the key part on `b`:

```
SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL)
  OR (t1.b=t2.b AND t2.b IS NULL);
```

8.2.1.16 ORDER BY Optimization

This section describes when MySQL can use an index to satisfy an `ORDER BY` clause, the `filesort` operation used when an index cannot be used, and execution plan information available from the optimizer about `ORDER BY`.

An `ORDER BY` with and without `LIMIT` may return rows in different orders, as discussed in Section 8.2.1.19, “LIMIT Query Optimization”.

- Use of Indexes to Satisfy ORDER BY
- Use of filesort to Satisfy ORDER BY
- Influencing ORDER BY Optimization
- ORDER BY Execution Plan Information Available

Use of Indexes to Satisfy ORDER BY

In some cases, MySQL may use an index to satisfy an `ORDER BY` clause and avoid the extra sorting involved in performing a `filesort` operation.

The index may also be used even if the `ORDER BY` does not match the index exactly, as long as all unused portions of the index and all extra `ORDER BY` columns are constants in the `WHERE` clause. If the index does not contain all columns accessed by the query, the index is used only if index access is cheaper than other access methods.

Assuming that there is an index on `(key_part1, key_part2)`, the following queries may use the index to resolve the `ORDER BY` part. Whether the optimizer actually does so depends on whether reading the index is more efficient than a table scan if columns not in the index must also be read.

- In this query, the index on `(key_part1, key_part2)` enables the optimizer to avoid sorting:

```
SELECT * FROM t1
  ORDER BY key_part1, key_part2;
```

  However, the query uses `SELECT *`, which may select more columns than `key_part1` and `key_part2`. In that case, scanning an entire index and looking up table rows to find columns not in the index may be more expensive than scanning the table and sorting the results. If so, the optimizer probably will not use the index. If `SELECT *` selects only the index columns, the index will be used and sorting avoided.

  If `t1` is an `InnoDB` table, the table primary key is implicitly part of the index, and the index can be used to resolve the `ORDER BY` for this query:

```
SELECT pk, key_part1, key_part2 FROM t1
  ORDER BY key_part1, key_part2;
```

- In this query, `key_part1` is constant, so all rows accessed through the index are in `key_part2` order, and an index on `(key_part1, key_part2)` avoids sorting if the `WHERE` clause is selective enough to make an index range scan cheaper than a table scan:

```
SELECT * FROM t1
  WHERE key_part1 = constant
  ORDER BY key_part2;
```

- In the next two queries, whether the index is used is similar to the same queries without `DESC` shown previously:

```
SELECT * FROM t1
  ORDER BY key_part1 DESC, key_part2 DESC;

SELECT * FROM t1
  WHERE key_part1 = constant
  ORDER BY key_part2 DESC;
```

- Two columns in an `ORDER BY` can sort in the same direction (both `ASC`, or both `DESC`) or in opposite directions (one `ASC`, one `DESC`). A condition for index use is that the index must have the same homogeneity, but need not have the same actual direction.

  If a query mixes `ASC` and `DESC`, the optimizer can use an index on the columns if the index also uses corresponding mixed ascending and descending columns:

  ```
  SELECT * FROM t1
    ORDER BY key_part1 DESC, key_part2 ASC;
  ```

  The optimizer can use an index on (*`key_part1`*, *`key_part2`*) if *`key_part1`* is descending and *`key_part2`* is ascending. It can also use an index on those columns (with a backward scan) if *`key_part1`* is ascending and *`key_part2`* is descending. See Section 8.3.13, “Descending Indexes”

- In the next two queries, *`key_part1`* is compared to a constant. The index will be used if the `WHERE` clause is selective enough to make an index range scan cheaper than a table scan:

  ```
  SELECT * FROM t1
    WHERE key_part1 > constant
    ORDER BY key_part1 ASC;

  SELECT * FROM t1
    WHERE key_part1 < constant
    ORDER BY key_part1 DESC;
  ```

- In the next query, the `ORDER BY` does not name *`key_part1`*, but all rows selected have a constant *`key_part1`* value, so the index can still be used:

  ```
  SELECT * FROM t1
    WHERE key_part1 = constant1 AND key_part2 > constant2
    ORDER BY key_part2;
  ```

In some cases, MySQL *cannot* use indexes to resolve the `ORDER BY`, although it may still use indexes to find the rows that match the `WHERE` clause. Examples:

- The query uses `ORDER BY` on different indexes:

  ```
  SELECT * FROM t1 ORDER BY key1, key2;
  ```

- The query uses `ORDER BY` on nonconsecutive parts of an index:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key1_part1, key1_part3;
  ```

- The index used to fetch the rows differs from the one used in the `ORDER BY`:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
  ```

- The query uses `ORDER BY` with an expression that includes terms other than the index column name:

```
SELECT * FROM t1 ORDER BY ABS(key);
SELECT * FROM t1 ORDER BY -key;
```

- The query joins many tables, and the columns in the `ORDER BY` are not all from the first nonconstant table that is used to retrieve rows. (This is the first table in the `EXPLAIN` output that does not have a `const` join type.)
- The query has different `ORDER BY` and `GROUP BY` expressions.
- There is an index on only a prefix of a column named in the `ORDER BY` clause. In this case, the index cannot be used to fully resolve the sort order. For example, if only the first 10 bytes of a `CHAR(20)` column are indexed, the index cannot distinguish values past the 10th byte and a `filesort` is needed.
- The index does not store rows in order. For example, this is true for a `HASH` index in a `MEMORY` table.

Availability of an index for sorting may be affected by the use of column aliases. Suppose that the column `t1.a` is indexed. In this statement, the name of the column in the select list is `a`. It refers to `t1.a`, as does the reference to `a` in the `ORDER BY`, so the index on `t1.a` can be used:

```
SELECT a FROM t1 ORDER BY a;
```

In this statement, the name of the column in the select list is also `a`, but it is the alias name. It refers to `ABS(a)`, as does the reference to `a` in the `ORDER BY`, so the index on `t1.a` cannot be used:

```
SELECT ABS(a) AS a FROM t1 ORDER BY a;
```

In the following statement, the `ORDER BY` refers to a name that is not the name of a column in the select list. But there is a column in `t1` named `a`, so the `ORDER BY` refers to `t1.a` and the index on `t1.a` can be used. (The resulting sort order may be completely different from the order for `ABS(a)`, of course.)

```
SELECT ABS(a) AS b FROM t1 ORDER BY a;
```

Previously (MySQL 5.7 and lower), `GROUP BY` sorted implicitly under certain conditions. In MySQL 8.0, that no longer occurs, so specifying `ORDER BY NULL` at the end to suppress implicit sorting (as was done previously) is no longer necessary. However, query results may differ from previous MySQL versions. To produce a given sort order, provide an `ORDER BY` clause.

Use of filesort to Satisfy ORDER BY

If an index cannot be used to satisfy an `ORDER BY` clause, MySQL performs a `filesort` operation that reads table rows and sorts them. A `filesort` constitutes an extra sorting phase in query execution.

To obtain memory for `filesort` operations, as of MySQL 8.0.12, the optimizer allocates memory buffers incrementally as needed, up to the size indicated by the `sort_buffer_size` system variable, rather than allocating a fixed amount of `sort_buffer_size` bytes up front, as was done prior to MySQL 8.0.12. This enables users to set `sort_buffer_size` to larger values to speed up larger sorts, without concern for excessive memory use for small sorts. (This benefit may not occur for multiple concurrent sorts on Windows, which has a weak multithreaded `malloc`.)

A `filesort` operation uses temporary disk files as necessary if the result set is too large to fit in memory. Some types of queries are particularly suited to completely in-memory `filesort` operations. For example, the optimizer can use `filesort` to efficiently handle in memory, without temporary files, the `ORDER BY` operation for queries (and subqueries) of the following form:

```
SELECT ... FROM single_table ... ORDER BY non_index_column [DESC] LIMIT [M,]N;
```

Such queries are common in web applications that display only a few rows from a larger result set. Examples:

```
SELECT col1, ... FROM t1 ... ORDER BY name LIMIT 10;
SELECT col1, ... FROM t1 ... ORDER BY RAND() LIMIT 15;
```

Influencing ORDER BY Optimization

For slow `ORDER BY` queries for which `filesort` is not used, try lowering the `max_length_for_sort_data` system variable to a value that is appropriate to trigger a `filesort`. (A symptom of setting the value of this variable too high is a combination of high disk activity and low CPU activity.)

To increase `ORDER BY` speed, check whether you can get MySQL to use indexes rather than an extra sorting phase. If this is not possible, try the following strategies:

- Increase the `sort_buffer_size` variable value. Ideally, the value should be large enough for the entire result set to fit in the sort buffer (to avoid writes to disk and merge passes).

  Take into account that the size of column values stored in the sort buffer is affected by the `max_sort_length` system variable value. For example, if tuples store values of long string columns and you increase the value of `max_sort_length`, the size of sort buffer tuples increases as well and may require you to increase `sort_buffer_size`.

  To monitor the number of merge passes (to merge temporary files), check the `Sort_merge_passes` status variable.

- Increase the `read_rnd_buffer_size` variable value so that more rows are read at a time.

- Change the `tmpdir` system variable to point to a dedicated file system with large amounts of free space. The variable value can list several paths that are used in round-robin fashion; you can use this feature to spread the load across several directories. Separate the paths by colon characters (`:`) on Unix and semicolon characters (`;`) on Windows. The paths should name directories in file systems located on different *physical* disks, not different partitions on the same disk.

ORDER BY Execution Plan Information Available

With `EXPLAIN` (see Section 8.8.1, “Optimizing Queries with EXPLAIN”), you can check whether MySQL can use indexes to resolve an `ORDER BY` clause:

- If the `Extra` column of `EXPLAIN` output does not contain `Using filesort`, the index is used and a `filesort` is not performed.

- If the `Extra` column of `EXPLAIN` output contains `Using filesort`, the index is not used and a `filesort` is performed.

In addition, if a `filesort` is performed, optimizer trace output includes a `filesort_summary` block. For example:

```
"filesort_summary": {
  "rows": 100,
  "examined_rows": 100,
  "number_of_tmp_files": 0,
  "peak_memory_used": 25192,
```

```
  "sort_mode": "<sort_key, packed_additional_fields>"
}
```

`peak_memory_used` indicates the maximum memory used at any one time during the sort. This is a value up to but not necessarily as large as the value of the `sort_buffer_size` system variable. Prior to MySQL 8.0.12, the output shows `sort_buffer_size` instead, indicating the value of `sort_buffer_size`. (Prior to MySQL 8.0.12, the optimizer always allocates `sort_buffer_size` bytes for the sort buffer. As of 8.0.12, the optimizer allocates sort-buffer memory incrementally, beginning with a small amount and adding more as necessary, up to `sort_buffer_size` bytes.)

The `sort_mode` value provides information about the contents of tuples in the sort buffer:

- `<sort_key, rowid>`: This indicates that sort buffer tuples are pairs that contain the sort key value and row ID of the original table row. Tuples are sorted by sort key value and the row ID is used to read the row from the table.
- `<sort_key, additional_fields>`: This indicates that sort buffer tuples contain the sort key value and columns referenced by the query. Tuples are sorted by sort key value and column values are read directly from the tuple.
- `<sort_key, packed_additional_fields>`: Like the previous variant, but the additional columns are packed tightly together instead of using a fixed-length encoding.

`EXPLAIN` does not distinguish whether the optimizer does or does not perform a `filesort` in memory. Use of an in-memory `filesort` can be seen in optimizer trace output. Look for `filesort_priority_queue_optimization`. For information about the optimizer trace, see MySQL Internals: Tracing the Optimizer.

8.2.1.17 GROUP BY Optimization

The most general way to satisfy a `GROUP BY` clause is to scan the whole table and create a new temporary table where all rows from each group are consecutive, and then use this temporary table to discover groups and apply aggregate functions (if any). In some cases, MySQL is able to do much better than that and avoid creation of temporary tables by using index access.

The most important preconditions for using indexes for `GROUP BY` are that all `GROUP BY` columns reference attributes from the same index, and that the index stores its keys in order (as is true, for example, for a `BTREE` index, but not for a `HASH` index). Whether use of temporary tables can be replaced by index access also depends on which parts of an index are used in a query, the conditions specified for these parts, and the selected aggregate functions.

There are two ways to execute a `GROUP BY` query through index access, as detailed in the following sections. The first method applies the grouping operation together with all range predicates (if any). The second method first performs a range scan, and then groups the resulting tuples.

- Loose Index Scan
- Tight Index Scan

Loose Index Scan can also be used in the absence of `GROUP BY` under some conditions. See Skip Scan Range Access Method.

Loose Index Scan

The most efficient way to process `GROUP BY` is when an index is used to directly retrieve the grouping columns. With this access method, MySQL uses the property of some index types that the keys are ordered (for example, `BTREE`). This property enables use of lookup groups in an index without having

to consider all keys in the index that satisfy all `WHERE` conditions. This access method considers only a fraction of the keys in an index, so it is called a *Loose Index Scan*. When there is no `WHERE` clause, a Loose Index Scan reads as many keys as the number of groups, which may be a much smaller number than that of all keys. If the `WHERE` clause contains range predicates (see the discussion of the `range` join type in Section 8.8.1, “Optimizing Queries with EXPLAIN”), a Loose Index Scan looks up the first key of each group that satisfies the range conditions, and again reads the smallest possible number of keys. This is possible under the following conditions:

- The query is over a single table.
- The `GROUP BY` names only columns that form a leftmost prefix of the index and no other columns. (If, instead of `GROUP BY`, the query has a `DISTINCT` clause, all distinct attributes refer to columns that form a leftmost prefix of the index.) For example, if a table `t1` has an index on `(c1,c2,c3)`, Loose Index Scan is applicable if the query has `GROUP BY c1, c2`. It is not applicable if the query has `GROUP BY c2, c3` (the columns are not a leftmost prefix) or `GROUP BY c1, c2, c4` (`c4` is not in the index).
- The only aggregate functions used in the select list (if any) are `MIN()` and `MAX()`, and all of them refer to the same column. The column must be in the index and must immediately follow the columns in the `GROUP BY`.
- Any other parts of the index than those from the `GROUP BY` referenced in the query must be constants (that is, they must be referenced in equalities with constants), except for the argument of `MIN()` or `MAX()` functions.
- For columns in the index, full column values must be indexed, not just a prefix. For example, with `c1 VARCHAR(20), INDEX (c1(10))`, the index uses only a prefix of `c1` values and cannot be used for Loose Index Scan.

If Loose Index Scan is applicable to a query, the `EXPLAIN` output shows `Using index for group-by` in the `Extra` column.

Assume that there is an index `idx(c1,c2,c3)` on table `t1(c1,c2,c3,c4)`. The Loose Index Scan access method can be used for the following queries:

```
SELECT c1, c2 FROM t1 GROUP BY c1, c2;
SELECT DISTINCT c1, c2 FROM t1;
SELECT c1, MIN(c2) FROM t1 GROUP BY c1;
SELECT c1, c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > const GROUP BY c1, c2;
SELECT c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT c1, c2 FROM t1 WHERE c3 = const GROUP BY c1, c2;
```

The following queries cannot be executed with this quick select method, for the reasons given:

- There are aggregate functions other than `MIN()` or `MAX()`:

  ```
  SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
  ```

- The columns in the `GROUP BY` clause do not form a leftmost prefix of the index:

  ```
  SELECT c1, c2 FROM t1 GROUP BY c2, c3;
  ```

- The query refers to a part of a key that comes after the `GROUP BY` part, and for which there is no equality with a constant:

  ```
  SELECT c1, c3 FROM t1 GROUP BY c1, c2;
  ```

Were the query to include `WHERE c3 = const`, Loose Index Scan could be used.

The Loose Index Scan access method can be applied to other forms of aggregate function references in the select list, in addition to the `MIN()` and `MAX()` references already supported:

- `AVG(DISTINCT)`, `SUM(DISTINCT)`, and `COUNT(DISTINCT)` are supported. `AVG(DISTINCT)` and `SUM(DISTINCT)` take a single argument. `COUNT(DISTINCT)` can have more than one column argument.
- There must be no `GROUP BY` or `DISTINCT` clause in the query.
- The Loose Index Scan limitations described previously still apply.

Assume that there is an index `idx(c1,c2,c3)` on table `t1(c1,c2,c3,c4)`. The Loose Index Scan access method can be used for the following queries:

```
SELECT COUNT(DISTINCT c1), SUM(DISTINCT c1) FROM t1;

SELECT COUNT(DISTINCT c1, c2), COUNT(DISTINCT c2, c1) FROM t1;
```

Tight Index Scan

A Tight Index Scan may be either a full index scan or a range index scan, depending on the query conditions.

When the conditions for a Loose Index Scan are not met, it still may be possible to avoid creation of temporary tables for `GROUP BY` queries. If there are range conditions in the `WHERE` clause, this method reads only the keys that satisfy these conditions. Otherwise, it performs an index scan. Because this method reads all keys in each range defined by the `WHERE` clause, or scans the whole index if there are no range conditions, it is called a *Tight Index Scan*. With a Tight Index Scan, the grouping operation is performed only after all keys that satisfy the range conditions have been found.

For this method to work, it is sufficient that there be a constant equality condition for all columns in a query referring to parts of the key coming before or in between parts of the `GROUP BY` key. The constants from the equality conditions fill in any “gaps” in the search keys so that it is possible to form complete prefixes of the index. These index prefixes then can be used for index lookups. If the `GROUP BY` result requires sorting, and it is possible to form search keys that are prefixes of the index, MySQL also avoids extra sorting operations because searching with prefixes in an ordered index already retrieves all the keys in order.

Assume that there is an index `idx(c1,c2,c3)` on table `t1(c1,c2,c3,c4)`. The following queries do not work with the Loose Index Scan access method described previously, but still work with the Tight Index Scan access method.

- There is a gap in the `GROUP BY`, but it is covered by the condition `c2 = 'a'`:

  ```
  SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
  ```

- The `GROUP BY` does not begin with the first part of the key, but there is a condition that provides a constant for that part:

  ```
  SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
  ```

8.2.1.18 DISTINCT Optimization

`DISTINCT` combined with `ORDER BY` needs a temporary table in many cases.

Because `DISTINCT` may use `GROUP BY`, learn how MySQL works with columns in `ORDER BY` or `HAVING` clauses that are not part of the selected columns. See Section 12.20.3, “MySQL Handling of GROUP BY”.

In most cases, a `DISTINCT` clause can be considered as a special case of `GROUP BY`. For example, the following two queries are equivalent:

```
SELECT DISTINCT c1, c2, c3 FROM t1
WHERE c1 > const;

SELECT c1, c2, c3 FROM t1
WHERE c1 > const GROUP BY c1, c2, c3;
```

Due to this equivalence, the optimizations applicable to `GROUP BY` queries can be also applied to queries with a `DISTINCT` clause. Thus, for more details on the optimization possibilities for `DISTINCT` queries, see Section 8.2.1.17, “GROUP BY Optimization”.

When combining `LIMIT row_count` with `DISTINCT`, MySQL stops as soon as it finds *`row_count`* unique rows.

If you do not use columns from all tables named in a query, MySQL stops scanning any unused tables as soon as it finds the first match. In the following case, assuming that `t1` is used before `t2` (which you can check with `EXPLAIN`), MySQL stops reading from `t2` (for any particular row in `t1`) when it finds the first row in `t2`:

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

8.2.1.19 LIMIT Query Optimization

If you need only a specified number of rows from a result set, use a `LIMIT` clause in the query, rather than fetching the whole result set and throwing away the extra data.

MySQL sometimes optimizes a query that has a `LIMIT row_count` clause and no `HAVING` clause:

- If you select only a few rows with `LIMIT`, MySQL uses indexes in some cases when normally it would prefer to do a full table scan.
- If you combine `LIMIT row_count` with `ORDER BY`, MySQL stops sorting as soon as it has found the first *`row_count`* rows of the sorted result, rather than sorting the entire result. If ordering is done by using an index, this is very fast. If a filesort must be done, all rows that match the query without the `LIMIT` clause are selected, and most or all of them are sorted, before the first *`row_count`* are found. After the initial rows have been found, MySQL does not sort any remainder of the result set.

  One manifestation of this behavior is that an `ORDER BY` query with and without `LIMIT` may return rows in different order, as described later in this section.
- If you combine `LIMIT row_count` with `DISTINCT`, MySQL stops as soon as it finds *`row_count`* unique rows.
- In some cases, a `GROUP BY` can be resolved by reading the index in order (or doing a sort on the index), then calculating summaries until the index value changes. In this case, `LIMIT row_count` does not calculate any unnecessary `GROUP BY` values.
- As soon as MySQL has sent the required number of rows to the client, it aborts the query unless you are using `SQL_CALC_FOUND_ROWS`. In that case, the number of rows can be retrieved with `SELECT FOUND_ROWS()`. See Section 12.15, “Information Functions”.

- `LIMIT 0` quickly returns an empty set. This can be useful for checking the validity of a query. It can also be employed to obtain the types of the result columns within applications that use a MySQL API that makes result set metadata available. With the `mysql` client program, you can use the `--column-type-info` option to display result column types.
- If the server uses temporary tables to resolve a query, it uses the `LIMIT *row_count*` clause to calculate how much space is required.
- If an index is not used for `ORDER BY` but a `LIMIT` clause is also present, the optimizer may be able to avoid using a merge file and sort the rows in memory using an in-memory `filesort` operation.

If multiple rows have identical values in the `ORDER BY` columns, the server is free to return those rows in any order, and may do so differently depending on the overall execution plan. In other words, the sort order of those rows is nondeterministic with respect to the nonordered columns.

One factor that affects the execution plan is `LIMIT`, so an `ORDER BY` query with and without `LIMIT` may return rows in different orders. Consider this query, which is sorted by the `category` column but nondeterministic with respect to the `id` and `rating` columns:

```
mysql> SELECT * FROM ratings ORDER BY category;
+----+----------+--------+
| id | category | rating |
+----+----------+--------+
|  1 |        1 |    4.5 |
|  5 |        1 |    3.2 |
|  3 |        2 |    3.7 |
|  4 |        2 |    3.5 |
|  6 |        2 |    3.5 |
|  2 |        3 |    5.0 |
|  7 |        3 |    2.7 |
+----+----------+--------+
```

Including `LIMIT` may affect order of rows within each `category` value. For example, this is a valid query result:

```
mysql> SELECT * FROM ratings ORDER BY category LIMIT 5;
+----+----------+--------+
| id | category | rating |
+----+----------+--------+
|  1 |        1 |    4.5 |
|  5 |        1 |    3.2 |
|  4 |        2 |    3.5 |
|  3 |        2 |    3.7 |
|  6 |        2 |    3.5 |
+----+----------+--------+
```

In each case, the rows are sorted by the `ORDER BY` column, which is all that is required by the SQL standard.

If it is important to ensure the same row order with and without `LIMIT`, include additional columns in the `ORDER BY` clause to make the order deterministic. For example, if `id` values are unique, you can make rows for a given `category` value appear in `id` order by sorting like this:

```
mysql> SELECT * FROM ratings ORDER BY category, id;
+----+----------+--------+
| id | category | rating |
+----+----------+--------+
|  1 |        1 |    4.5 |
|  5 |        1 |    3.2 |
```

```
|  3 |        2 |    3.7 |
|  4 |        2 |    3.5 |
|  6 |        2 |    3.5 |
|  2 |        3 |    5.0 |
|  7 |        3 |    2.7 |
+----+----------+--------+

mysql> SELECT * FROM ratings ORDER BY category, id LIMIT 5;
+----+----------+--------+
| id | category | rating |
+----+----------+--------+
|  1 |        1 |    4.5 |
|  5 |        1 |    3.2 |
|  3 |        2 |    3.7 |
|  4 |        2 |    3.5 |
|  6 |        2 |    3.5 |
+----+----------+--------+
```

8.2.1.20 Function Call Optimization

MySQL functions are tagged internally as deterministic or nondeterministic. A function is nondeterministic if, given fixed values for its arguments, it can return different results for different invocations. Examples of nondeterministic functions: `RAND()`, `UUID()`.

If a function is tagged nondeterministic, a reference to it in a `WHERE` clause is evaluated for every row (when selecting from one table) or combination of rows (when selecting from a multiple-table join).

MySQL also determines when to evaluate functions based on types of arguments, whether the arguments are table columns or constant values. A deterministic function that takes a table column as argument must be evaluated whenever that column changes value.

Nondeterministic functions may affect query performance. For example, some optimizations may not be available, or more locking might be required. The following discussion uses `RAND()` but applies to other nondeterministic functions as well.

Suppose that a table `t` has this definition:

```
CREATE TABLE t (id INT NOT NULL PRIMARY KEY, col_a VARCHAR(100));
```

Consider these two queries:

```
SELECT * FROM t WHERE id = POW(1,2);
SELECT * FROM t WHERE id = FLOOR(1 + RAND() * 49);
```

Both queries appear to use a primary key lookup because of the equality comparison against the primary key, but that is true only for the first of them:

- The first query always produces a maximum of one row because `POW()` with constant arguments is a constant value and is used for index lookup.
- The second query contains an expression that uses the nondeterministic function `RAND()`, which is not constant in the query but in fact has a new value for every row of table `t`. Consequently, the query reads every row of the table, evaluates the predicate for each row, and outputs all rows for which the primary key matches the random value. This might be zero, one, or multiple rows, depending on the `id` column values and the values in the `RAND()` sequence.

The effects of nondeterminism are not limited to `SELECT` statements. This `UPDATE` statement uses a nondeterministic function to select rows to be modified:

```
UPDATE t SET col_a = some_expr WHERE id = FLOOR(1 + RAND() * 49);
```

Presumably the intent is to update at most a single row for which the primary key matches the expression. However, it might update zero, one, or multiple rows, depending on the `id` column values and the values in the `RAND()` sequence.

The behavior just described has implications for performance and replication:

- Because a nondeterministic function does not produce a constant value, the optimizer cannot use strategies that might otherwise be applicable, such as index lookups. The result may be a table scan.
- `InnoDB` might escalate to a range-key lock rather than taking a single row lock for one matching row.
- Updates that do not execute deterministically are unsafe for replication.

The difficulties stem from the fact that the `RAND()` function is evaluated once for every row of the table. To avoid multiple function evaluations, use one of these techniques:

- Move the expression containing the nondeterministic function to a separate statement, saving the value in a variable. In the original statement, replace the expression with a reference to the variable, which the optimizer can treat as a constant value:

  ```
  SET @keyval = FLOOR(1 + RAND() * 49);
  UPDATE t SET col_a = some_expr WHERE id = @keyval;
  ```

- Assign the random value to a variable in a derived table. This technique causes the variable to be assigned a value, once, prior to its use in the comparison in the `WHERE` clause:

  ```
  UPDATE /*+ NO_MERGE(dt) */ t, (SELECT FLOOR(1 + RAND() * 49) AS r) AS dt
  SET col_a = some_expr WHERE id = dt.r;
  ```

As mentioned previously, a nondeterministic expression in the `WHERE` clause might prevent optimizations and result in a table scan. However, it may be possible to partially optimize the `WHERE` clause if other expressions are deterministic. For example:

```
SELECT * FROM t WHERE partial_key=5 AND some_column=RAND();
```

If the optimizer can use `partial_key` to reduce the set of rows selected, `RAND()` is executed fewer times, which diminishes the effect of nondeterminism on optimization.

8.2.1.21 Window Function Optimization

Window functions affect the strategies the optimizer considers:

- Derived table merging for a subquery is disabled if the subquery has window functions. The subquery is always materialized.
- Semijoins are not applicable to window function optimization because semijoins apply to subqueries in `WHERE` and `JOIN ... ON`, which cannot contain window functions.
- The optimizer processes multiple windows that have the same ordering requirements in sequence, so sorting can be skipped for windows following the first one.
- The optimizer makes no attempt to merge windows that could be evaluated in a single step (for example, when multiple `OVER` clauses contain identical window definitions). The workaround is to define the window in a `WINDOW` clause and refer to the window name in the `OVER` clauses.

An aggregate function not used as a window function is aggregated in the outermost possible query. For example, in this query, MySQL sees that `COUNT(t1.b)` is something that cannot exist in the outer query because of its placement in the `WHERE` clause:

```
SELECT * FROM t1 WHERE t1.a = (SELECT COUNT(t1.b) FROM t2);
```

Consequently, MySQL aggregates inside the subquery, treating `t1.b` as a constant and returning the count of rows of `t2`.

Replacing `WHERE` with `HAVING` results in an error:

```
mysql> SELECT * FROM t1 HAVING t1.a = (SELECT COUNT(t1.b) FROM t2);
ERROR 1140 (42000): In aggregated query without GROUP BY, expression #1
of SELECT list contains nonaggregated column 'test.t1.a'; this is
incompatible with sql_mode=only_full_group_by
```

The error occurs because `COUNT(t1.b)` can exist in the `HAVING`, and so makes the outer query aggregated.

Window functions (including aggregate functions used as window functions) do not have the preceding complexity. They always aggregate in the subquery where they are written, never in the outer query.

Window function evaluation may be affected by the value of the `windowing_use_high_precision` system variable, which determines whether to compute window operations without loss of precision. By default, `windowing_use_high_precision` is enabled.

For some moving frame aggregates, the inverse aggregate function can be applied to remove values from the aggregate. This can improve performance but possibly with a loss of precision. For example, adding a very small floating-point value to a very large value causes the very small value to be “hidden” by the large value. When inverting the large value later, the effect of the small value is lost.

Loss of precision due to inverse aggregation is a factor only for operations on floating-point (approximate-value) data types. For other types, inverse aggregation is safe; this includes `DECIMAL`, which permits a fractional part but is an exact-value type.

For faster execution, MySQL always uses inverse aggregation when it is safe:

- For floating-point values, inverse aggregation is not always safe and might result in loss of precision. The default is to avoid inverse aggregation, which is slower but preserves precision. If it is permissible to sacrifice safety for speed, `windowing_use_high_precision` can be disabled to permit inverse aggregation.
- For nonfloating-point data types, inverse aggregation is always safe and is used regardless of the `windowing_use_high_precision` value.
- `windowing_use_high_precision` has no effect on `MIN()` and `MAX()`, which do not use inverse aggregation in any case.

For evaluation of the variance functions `STDDEV_POP()`, `STDDEV_SAMP()`, `VAR_POP()`, `VAR_SAMP()`, and their synonyms, evaluation can occur in optimized mode or default mode. Optimized mode may produce slightly different results in the last significant digits. If such differences are permissible, `windowing_use_high_precision` can be disabled to permit optimized mode.

For `EXPLAIN`, windowing execution plan information is too extensive to display in traditional output format. To see windowing information, use `EXPLAIN FORMAT=JSON` and look for the `windowing` element.

8.2.1.22 Row Constructor Expression Optimization

Row constructors permit simultaneous comparisons of multiple values. For example, these two statements are semantically equivalent:

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

In addition, the optimizer handles both expressions the same way.

The optimizer is less likely to use available indexes if the row constructor columns do not cover the prefix of an index. Consider the following table, which has a primary key on `(c1, c2, c3)`:

```
CREATE TABLE t1 (
  c1 INT, c2 INT, c3 INT, c4 CHAR(100),
  PRIMARY KEY(c1,c2,c3)
);
```

In this query, the `WHERE` clause uses all columns in the index. However, the row constructor itself does not cover an index prefix, with the result that the optimizer uses only `c1` (`key_len=4`, the size of `c1`):

```
mysql> EXPLAIN SELECT * FROM t1
       WHERE c1=1 AND (c2,c3) > (1,1)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: NULL
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 3
     filtered: 100.00
        Extra: Using where
```

In such cases, rewriting the row constructor expression using an equivalent nonconstructor expression may result in more complete index use. For the given query, the row constructor and equivalent nonconstructor expressions are:

```
(c2,c3) > (1,1)
c2 > 1 OR ((c2 = 1) AND (c3 > 1))
```

Rewriting the query to use the nonconstructor expression results in the optimizer using all three columns in the index (`key_len=12`):

```
mysql> EXPLAIN SELECT * FROM t1
       WHERE c1 = 1 AND (c2 > 1 OR ((c2 = 1) AND (c3 > 1)))\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: NULL
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 12
```

```
         ref: NULL
        rows: 3
    filtered: 100.00
       Extra: Using where
```

Thus, for better results, avoid mixing row constructors with `AND`/`OR` expressions. Use one or the other.

Under certain conditions, the optimizer can apply the range access method to `IN()` expressions that have row constructor arguments. See Range Optimization of Row Constructor Expressions.

8.2.1.23 Avoiding Full Table Scans

The output from `EXPLAIN` shows `ALL` in the `type` column when MySQL uses a full table scan to resolve a query. This usually happens under the following conditions:

- The table is so small that it is faster to perform a table scan than to bother with a key lookup. This is common for tables with fewer than 10 rows and a short row length.
- There are no usable restrictions in the `ON` or `WHERE` clause for indexed columns.
- You are comparing indexed columns with constant values and MySQL has calculated (based on the index tree) that the constants cover too large a part of the table and that a table scan would be faster. See Section 8.2.1.1, “WHERE Clause Optimization”.
- You are using a key with low cardinality (many rows match the key value) through another column. In this case, MySQL assumes that by using the key it probably will do many key lookups and that a table scan would be faster.

For small tables, a table scan often is appropriate and the performance impact is negligible. For large tables, try the following techniques to avoid having the optimizer incorrectly choose a table scan:

- Use `ANALYZE TABLE` *`tbl_name`* to update the key distributions for the scanned table. See Section 13.7.3.1, “ANALYZE TABLE Syntax”.
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

  ```
  SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
    WHERE t1.col_name=t2.col_name;
  ```

  See Section 8.9.4, “Index Hints”.
- Start `mysqld` with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.8, “Server System Variables”.

8.2.2 Optimizing Subqueries, Derived Tables, View References, and Common Table Expressions

The MySQL query optimizer has different strategies available to evaluate subqueries:

- For `IN` (or `=ANY`) subqueries, the optimizer has these choices:
  - Semijoin
  - Materialization

  - `EXISTS` strategy
- For `NOT IN` (or `<>ALL`) subqueries, the optimizer has these choices:
  - Materialization
  - `EXISTS` strategy

For derived tables, the optimizer has these choices (which also apply to view references and common table expressions):

- Merge the derived table into the outer query block
- Materialize the derived table to an internal temporary table

The following discussion provides more information about the preceding optimization strategies.

Note

A limitation on `UPDATE` and `DELETE` statements that use a subquery to modify a single table is that the optimizer does not use semijoin or materialization subquery optimizations. As a workaround, try rewriting them as multiple-table `UPDATE` and `DELETE` statements that use a join rather than a subquery.

8.2.2.1 Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations

A semijoin is a preparation-time transformation that enables multiple execution strategies such as table pullout, duplicate weedout, first match, loose scan, and materialization. The optimizer uses semijoin strategies to improve subquery execution, as described in this section.

For an inner join between two tables, the join returns a row from one table as many times as there are matches in the other table. But for some questions, the only information that matters is whether there is a match, not the number of matches. Suppose that there are tables named `class` and `roster` that list classes in a course curriculum and class rosters (students enrolled in each class), respectively. To list the classes that actually have students enrolled, you could use this join:

```
SELECT class.class_num, class.class_name
FROM class INNER JOIN roster
WHERE class.class_num = roster.class_num;
```

However, the result lists each class once for each enrolled student. For the question being asked, this is unnecessary duplication of information.

Assuming that `class_num` is a primary key in the `class` table, duplicate suppression is possible by using `SELECT DISTINCT`, but it is inefficient to generate all matching rows first only to eliminate duplicates later.

The same duplicate-free result can be obtained by using a subquery:

```
SELECT class_num, class_name
FROM class
WHERE class_num IN (SELECT class_num FROM roster);
```

Here, the optimizer can recognize that the `IN` clause requires the subquery to return only one instance of each class number from the `roster` table. In this case, the query can use a *semijoin*; that is, an operation that returns only one instance of each row in `class` that is matched by rows in `roster`.

In MySQL 8.0.16 and later, this strategy can also be employed with `EXISTS` subqueries by transforming an `EXISTS` subquery that appears at the top level of a `WHERE` or `ON` clause of an `IN` or `EXISTS` predicate that refers to a subquery:

```
WHERE ... EXISTS (SELECT ... FROM ...)

-> WHERE ... 1 IN (SELECT 1 FROM ...)
```

After this, the subquery operation can be handled as a semijoin.

Beginning with MySQL 8.0.17, the following subqueries are transformed into antijoins:

- `NOT IN (SELECT ... FROM ...)`
- `NOT EXISTS (SELECT ... FROM ...)`.
- `IN (SELECT ... FROM ...) IS NOT TRUE`
- `EXISTS (SELECT ... FROM ...) IS NOT TRUE`.
- `IN (SELECT ... FROM ...) IS FALSE`
- `EXISTS (SELECT ... FROM ...) IS FALSE`.

In short, any case in which a subquery of the form `IN (SELECT ... FROM ...)` or `EXISTS (SELECT ... FROM ...)` is negated is transformed into an antijoin.

An antijoin is an operation that returns only rows for which there is no match. Consider the query shown here:

```
SELECT class_num, class_name
FROM class
WHERE class_num NOT IN (SELECT class_num FROM roster);
```

This query is rewritten internally as the antijoin `SELECT class_num, class_name FROM class ANTIJOIN roster ON class_num`, which returns one instance of each row in `class` that is *not* matched by any rows in `roster`. This means that, for each row in `class`, as soon as a match is found in `roster`, the row in `class` can be discarded.

Antijoin transformations cannot in most cases be applied if the expressions being compared are nullable. An exception to this rule is that `(... NOT IN (SELECT ...)) IS NOT FALSE` and its equivalent `(... IN (SELECT ...)) IS NOT TRUE` can be transformed into antijoins.

Outer join and inner join syntax is permitted in the outer query specification, and table references may be base tables, derived tables, view references, or common table expressions.

In MySQL, a subquery must satisfy these criteria to be handled as a semijoin (or, in MySQL 8.0.17 and later, an antijoin if `NOT` modifies the subquery):

- It must be an `IN` (or `=ANY`) subquery that appears at the top level of the `WHERE` or `ON` clause, possibly as a term in an `AND` expression. For example:

  ```
  SELECT ...
  FROM ot1, ...
  WHERE (oe1, ...) IN (SELECT ie1, ... FROM it1, ... WHERE ...);
  ```

  Here, *`ot_i`* and *`it_i`* represent tables in the outer and inner parts of the query, and *`oe_i`* and *`ie_i`* represent expressions that refer to columns in the outer and inner tables.

In MySQL 8.0.17 and later, the subquery can also be the argument to an expression modified by `NOT`, `IS [NOT] TRUE`, or `IS [NOT] FALSE`.

- It must be a single `SELECT` without `UNION` constructs.
- It must not contain a `GROUP BY` or `HAVING` clause.
- It must not be implicitly grouped (it must contain no aggregate functions).
- It must not have `ORDER BY` with `LIMIT`.
- The statement must not use the `STRAIGHT_JOIN` join type in the outer query.
- The `STRAIGHT_JOIN` modifier must not be present.
- The number of outer and inner tables together must be less than the maximum number of tables permitted in a join.

The subquery may be correlated or uncorrelated. In MySQL 8.0.16 and later, decorrelation looks at equality predicates in the `WHERE` clause of a subquery used as the argument to `EXISTS`, and makes it possible to optimize it as if it was used within `IN (SELECT b FROM ...)`. The term *trivially correlated* means that the predicate is an equality predicate, that it is the sole predicate in the `WHERE` clause (or is combined with `AND`), and that one operand is from a table referenced in the subquery and the other operand is from the outer query block.

`DISTINCT` is permitted, as is `LIMIT` unless `ORDER BY` is also used.

If a subquery meets the preceding criteria, MySQL converts it to a semijoin (or, in MySQL 8.0.17 or later, an antijoin if applicable) and makes a cost-based choice from these strategies:

- Convert the subquery to a join, or use table pullout and run the query as an inner join between subquery tables and outer tables. Table pullout pulls a table out from the subquery to the outer query.
- Duplicate Weedout: Run the semijoin as if it was a join and remove duplicate records using a temporary table.
- FirstMatch: When scanning the inner tables for row combinations and there are multiple instances of a given value group, choose one rather than returning them all. This "shortcuts" scanning and eliminates production of unnecessary rows.
- LooseScan: Scan a subquery table using an index that enables a single value to be chosen from each subquery's value group.
- Materialize the subquery into an indexed temporary table that is used to perform a join, where the index is used to remove duplicates. The index might also be used later for lookups when joining the temporary table with the outer tables; if not, the table is scanned. For more information about materialization, see Section 8.2.2.2, “Optimizing Subqueries with Materialization”.

Each of these strategies can be enabled or disabled using the following `optimizer_switch` system variable flags:

- The `semijoin` flag controls whether semijoins are used. Starting with MySQL 8.0.17, this also applies to antijoins.
- If `semijoin` is enabled, the `firstmatch`, `loosescan`, `duplicateweedout`, and `materialization` flags enable finer control over the permitted semijoin strategies.

- If the `duplicateweedout` semijoin strategy is disabled, it is not used unless all other applicable strategies are also disabled.
- If `duplicateweedout` is disabled, on occasion the optimizer may generate a query plan that is far from optimal. This occurs due to heuristic pruning during greedy search, which can be avoided by setting `optimizer_prune_level=0`.

These flags are enabled by default. See Section 8.9.2, “Switchable Optimizations”.

The optimizer minimizes differences in handling of views and derived tables. This affects queries that use the `STRAIGHT_JOIN` modifier and a view with an `IN` subquery that can be converted to a semijoin. The following query illustrates this because the change in processing causes a change in transformation, and thus a different execution strategy:

```
CREATE VIEW v AS
SELECT *
FROM t1
WHERE a IN (SELECT b
            FROM t2);

SELECT STRAIGHT_JOIN *
FROM t3 JOIN v ON t3.x = v.a;
```

The optimizer first looks at the view and converts the `IN` subquery to a semijoin, then checks whether it is possible to merge the view into the outer query. Because the `STRAIGHT_JOIN` modifier in the outer query prevents semijoin, the optimizer refuses the merge, causing derived table evaluation using a materialized table.

`EXPLAIN` output indicates the use of semijoin strategies as follows:

- For extended `EXPLAIN` output, the text displayed by a following `SHOW WARNINGS` shows the rewritten query, which displays the semijoin structure. (See Section 8.8.3, “Extended EXPLAIN Output Format”.) From this you can get an idea about which tables were pulled out of the semijoin. If a subquery was converted to a semijoin, you will see that the subquery predicate is gone and its tables and `WHERE` clause were merged into the outer query join list and `WHERE` clause.
- Temporary table use for Duplicate Weedout is indicated by `Start temporary` and `End temporary` in the `Extra` column. Tables that were not pulled out and are in the range of `EXPLAIN` output rows covered by `Start temporary` and `End temporary` have their `rowid` in the temporary table.
- `FirstMatch(`*`tbl_name`*`)` in the `Extra` column indicates join shortcutting.
- `LooseScan(`*`m`*`..`*`n`*`)` in the `Extra` column indicates use of the LooseScan strategy. *`m`* and *`n`* are key part numbers.
- Temporary table use for materialization is indicated by rows with a `select_type` value of `MATERIALIZED` and rows with a `table` value of `<subquery`*`N`*`>`.

8.2.2.2 Optimizing Subqueries with Materialization

The optimizer uses materialization to enable more efficient subquery processing. Materialization speeds up query execution by generating a subquery result as a temporary table, normally in memory. The first time MySQL needs the subquery result, it materializes that result into a temporary table. Any subsequent time the result is needed, MySQL refers again to the temporary table. The optimizer may index the table with a hash index to make lookups fast and inexpensive. The index contains unique values to eliminate duplicates and make the table smaller.

Subquery materialization uses an in-memory temporary table when possible, falling back to on-disk storage if the table becomes too large. See Section 8.4.4, “Internal Temporary Table Use in MySQL”.

If materialization is not used, the optimizer sometimes rewrites a noncorrelated subquery as a correlated subquery. For example, the following `IN` subquery is noncorrelated (*`where_condition`* involves only columns from `t2` and not `t1`):

```
SELECT * FROM t1
WHERE t1.a IN (SELECT t2.b FROM t2 WHERE where_condition);
```

The optimizer might rewrite this as an `EXISTS` correlated subquery:

```
SELECT * FROM t1
WHERE EXISTS (SELECT t2.b FROM t2 WHERE where_condition AND t1.a=t2.b);
```

Subquery materialization using a temporary table avoids such rewrites and makes it possible to execute the subquery only once rather than once per row of the outer query.

For subquery materialization to be used in MySQL, the `optimizer_switch` system variable `materialization` flag must be enabled. (See Section 8.9.2, “Switchable Optimizations”.) With the `materialization` flag enabled, materialization applies to subquery predicates that appear anywhere (in the select list, `WHERE`, `ON`, `GROUP BY`, `HAVING`, or `ORDER BY`), for predicates that fall into any of these use cases:

- The predicate has this form, when no outer expression *`oe_i`* or inner expression *`ie_i`* is nullable. *`N`* is 1 or larger.

  ```
  (oe_1, oe_2, ..., oe_N) [NOT] IN (SELECT ie_1, i_2, ..., ie_N ...)
  ```

- The predicate has this form, when there is a single outer expression *`oe`* and inner expression *`ie`*. The expressions can be nullable.

  ```
  oe [NOT] IN (SELECT ie ...)
  ```

- The predicate is `IN` or `NOT IN` and a result of `UNKNOWN` (`NULL`) has the same meaning as a result of `FALSE`.

The following examples illustrate how the requirement for equivalence of `UNKNOWN` and `FALSE` predicate evaluation affects whether subquery materialization can be used. Assume that *`where_condition`* involves columns only from `t2` and not `t1` so that the subquery is noncorrelated.

This query is subject to materialization:

```
SELECT * FROM t1
WHERE t1.a IN (SELECT t2.b FROM t2 WHERE where_condition);
```

Here, it does not matter whether the `IN` predicate returns `UNKNOWN` or `FALSE`. Either way, the row from `t1` is not included in the query result.

An example where subquery materialization is not used is the following query, where `t2.b` is a nullable column:

```
SELECT * FROM t1
WHERE (t1.a,t1.b) NOT IN (SELECT t2.a,t2.b FROM t2
```

```
                        WHERE where_condition);
```

The following restrictions apply to the use of subquery materialization:

- The types of the inner and outer expressions must match. For example, the optimizer might be able to use materialization if both expressions are integer or both are decimal, but cannot if one expression is integer and the other is decimal.
- The inner expression cannot be a `BLOB`.

Use of `EXPLAIN` with a query provides some indication of whether the optimizer uses subquery materialization:

- Compared to query execution that does not use materialization, `select_type` may change from `DEPENDENT SUBQUERY` to `SUBQUERY`. This indicates that, for a subquery that would be executed once per outer row, materialization enables the subquery to be executed just once.
- For extended `EXPLAIN` output, the text displayed by a following `SHOW WARNINGS` includes `materialize` and `materialized-subquery`.

8.2.2.3 Optimizing Subqueries with the EXISTS Strategy

Certain optimizations are applicable to comparisons that use the `IN` (or `=ANY`) operator to test subquery results. This section discusses these optimizations, particularly with regard to the challenges that `NULL` values present. The last part of the discussion suggests how you can help the optimizer.

Consider the following subquery comparison:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

MySQL evaluates queries “from outside to inside.” That is, it first obtains the value of the outer expression *`outer_expr`*, and then runs the subquery and captures the rows that it produces.

A very useful optimization is to “inform” the subquery that the only rows of interest are those where the inner expression *`inner_expr`* is equal to *`outer_expr`*. This is done by pushing down an appropriate equality into the subquery's `WHERE` clause to make it more restrictive. The converted comparison looks like this:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

After the conversion, MySQL can use the pushed-down equality to limit the number of rows it must examine to evaluate the subquery.

More generally, a comparison of *`N`* values to a subquery that returns *`N`*-value rows is subject to the same conversion. If *`oe_i`* and *`ie_i`* represent corresponding outer and inner expression values, this subquery comparison:

```
(oe_1, ..., oe_N) IN
  (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

Becomes:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                          AND oe_1 = ie_1
                          AND ...
```

```
                            AND oe_N = ie_N)
```

For simplicity, the following discussion assumes a single pair of outer and inner expression values.

The conversion just described has its limitations. It is valid only if we ignore possible NULL values. That is, the “pushdown” strategy works as long as both of these conditions are true:

- *outer_expr* and *inner_expr* cannot be NULL.
- You need not distinguish NULL from FALSE subquery results. If the subquery is a part of an OR or AND expression in the WHERE clause, MySQL assumes that you do not care. Another instance where the optimizer notices that NULL and FALSE subquery results need not be distinguished is this construct:

  ```
  ... WHERE outer_expr IN (subquery)
  ```

  In this case, the WHERE clause rejects the row whether IN (*subquery*) returns NULL or FALSE.

When either or both of those conditions do not hold, optimization is more complex.

Suppose that *outer_expr* is known to be a non-NULL value but the subquery does not produce a row such that *outer_expr* = *inner_expr*. Then *outer_expr* IN (SELECT ...) evaluates as follows:

- NULL, if the SELECT produces any row where *inner_expr* is NULL
- FALSE, if the SELECT produces only non-NULL values or produces nothing

In this situation, the approach of looking for rows with *outer_expr* = *inner_expr* is no longer valid. It is necessary to look for such rows, but if none are found, also look for rows where *inner_expr* is NULL. Roughly speaking, the subquery can be converted to something like this:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND
        (outer_expr=inner_expr OR inner_expr IS NULL))
```

The need to evaluate the extra IS NULL condition is why MySQL has the ref_or_null access method:

```
mysql> EXPLAIN
       SELECT outer_expr IN (SELECT t2.maybe_null_key
                             FROM t2, t3 WHERE ...)
       FROM t1;
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: ref_or_null
possible_keys: maybe_null_key
          key: maybe_null_key
      key_len: 5
          ref: func
         rows: 2
        Extra: Using where; Using index
...
```

The unique_subquery and index_subquery subquery-specific access methods also have “or NULL” variants.

The additional `OR ... IS NULL` condition makes query execution slightly more complicated (and some optimizations within the subquery become inapplicable), but generally this is tolerable.

The situation is much worse when *`outer_expr`* can be `NULL`. According to the SQL interpretation of `NULL` as “unknown value,” `NULL IN (SELECT` *`inner_expr`* `...)` should evaluate to:

- `NULL`, if the `SELECT` produces any rows
- `FALSE`, if the `SELECT` produces no rows

For proper evaluation, it is necessary to be able to check whether the `SELECT` has produced any rows at all, so *`outer_expr = inner_expr`* cannot be pushed down into the subquery. This is a problem because many real world subqueries become very slow unless the equality can be pushed down.

Essentially, there must be different ways to execute the subquery depending on the value of *`outer_expr`*.

The optimizer chooses SQL compliance over speed, so it accounts for the possibility that *`outer_expr`* might be `NULL`:

- If *`outer_expr`* is `NULL`, to evaluate the following expression, it is necessary to execute the `SELECT` to determine whether it produces any rows:

  ```
  NULL IN (SELECT inner_expr FROM ... WHERE subquery_where)
  ```

  It is necessary to execute the original `SELECT` here, without any pushed-down equalities of the kind mentioned previously.

- On the other hand, when *`outer_expr`* is not `NULL`, it is absolutely essential that this comparison:

  ```
  outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
  ```

  Be converted to this expression that uses a pushed-down condition:

  ```
  EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
  ```

  Without this conversion, subqueries will be slow.

To solve the dilemma of whether or not to push down conditions into the subquery, the conditions are wrapped within “trigger” functions. Thus, an expression of the following form:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

Is converted into:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                          AND trigcond(outer_expr=inner_expr))
```

More generally, if the subquery comparison is based on several pairs of outer and inner expressions, the conversion takes this comparison:

```
(oe_1, ..., oe_N) IN (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

And converts it to this expression:

```
```

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                           AND trigcond(oe_1=ie_1)
                           AND ...
                           AND trigcond(oe_N=ie_N)
       )
```

Each `trigcond(X)` is a special function that evaluates to the following values:

- *X* when the “linked” outer expression *oe_i* is not `NULL`
- `TRUE` when the “linked” outer expression *oe_i* is `NULL`

Note

Trigger functions are *not* triggers of the kind that you create with `CREATE TRIGGER`.

Equalities that are wrapped within `trigcond()` functions are not first class predicates for the query optimizer. Most optimizations cannot deal with predicates that may be turned on and off at query execution time, so they assume any `trigcond(X)` to be an unknown function and ignore it. Triggered equalities can be used by those optimizations:

- Reference optimizations: `trigcond(X=Y [OR Y IS NULL])` can be used to construct `ref`, `eq_ref`, or `ref_or_null` table accesses.
- Index lookup-based subquery execution engines: `trigcond(X=Y)` can be used to construct `unique_subquery` or `index_subquery` accesses.
- Table-condition generator: If the subquery is a join of several tables, the triggered condition is checked as soon as possible.

When the optimizer uses a triggered condition to create some kind of index lookup-based access (as for the first two items of the preceding list), it must have a fallback strategy for the case when the condition is turned off. This fallback strategy is always the same: Do a full table scan. In `EXPLAIN` output, the fallback shows up as `Full scan on NULL key` in the `Extra` column:

```
mysql> EXPLAIN SELECT t1.col1,
       t1.col1 IN (SELECT t2.key1 FROM t2 WHERE t2.col2=t1.col2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
        ...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: index_subquery
possible_keys: key1
          key: key1
      key_len: 5
          ref: func
         rows: 2
        Extra: Using where; Full scan on NULL key
```

If you run `EXPLAIN` followed by `SHOW WARNINGS`, you can see the triggered condition:

```
*************************** 1. row ***************************
  Level: Note
   Code: 1003
```

```
Message: select `test`.`t1`.`col1` AS `col1`,
         <in_optimizer>(`test`.`t1`.`col1`,
         <exists>(<index_lookup>(<cache>(`test`.`t1`.`col1`) in t2
         on key1 checking NULL
         where (`test`.`t2`.`col2` = `test`.`t1`.`col2`) having
         trigcond(<is_not_null_test>(`test`.`t2`.`key1`))))) AS
         `t1.col1 IN (select t2.key1 from t2 where t2.col2=t1.col2)`
         from `test`.`t1`
```

The use of triggered conditions has some performance implications. A `NULL IN (SELECT ...)` expression now may cause a full table scan (which is slow) when it previously did not. This is the price paid for correct results (the goal of the trigger-condition strategy is to improve compliance, not speed).

For multiple-table subqueries, execution of `NULL IN (SELECT ...)` is particularly slow because the join optimizer does not optimize for the case where the outer expression is `NULL`. It assumes that subquery evaluations with `NULL` on the left side are very rare, even if there are statistics that indicate otherwise. On the other hand, if the outer expression might be `NULL` but never actually is, there is no performance penalty.

To help the query optimizer better execute your queries, use these suggestions:

- Declare a column as `NOT NULL` if it really is. This also helps other aspects of the optimizer by simplifying condition testing for the column.
- If you need not distinguish a `NULL` from `FALSE` subquery result, you can easily avoid the slow execution path. Replace a comparison that looks like this:

  ```
  outer_expr IN (SELECT inner_expr FROM ...)
  ```

  with this expression:

  ```
  (outer_expr IS NOT NULL) AND (outer_expr IN (SELECT inner_expr FROM ...))
  ```

  Then `NULL IN (SELECT ...)` is never evaluated because MySQL stops evaluating `AND` parts as soon as the expression result is clear.

  Another possible rewrite:

  ```
  EXISTS (SELECT inner_expr FROM ...
          WHERE inner_expr=outer_expr)
  ```

  This would apply when you need not distinguish `NULL` from `FALSE` subquery results, in which case you may actually want `EXISTS`.

The `subquery_materialization_cost_based` flag of the `optimizer_switch` system variable enables control over the choice between subquery materialization and `IN`-to-`EXISTS` subquery transformation. See Section 8.9.2, “Switchable Optimizations”.

8.2.2.4 Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization

The optimizer can handle derived table references using two strategies (which also apply to view references and common table expressions):

- Merge the derived table into the outer query block
- Materialize the derived table to an internal temporary table

Example 1:

```
SELECT * FROM (SELECT * FROM t1) AS derived_t1;
```

With merging of the derived table `derived_t1`, that query is executed similar to:

```
SELECT * FROM t1;
```

Example 2:

```
SELECT *
  FROM t1 JOIN (SELECT t2.f1 FROM t2) AS derived_t2 ON t1.f2=derived_t2.f1
  WHERE t1.f1 > 0;
```

With merging of the derived table `derived_t2`, that query is executed similar to:

```
SELECT t1.*, t2.f1
  FROM t1 JOIN t2 ON t1.f2=t2.f1
  WHERE t1.f1 > 0;
```

With materialization, `derived_t1` and `derived_t2` are each treated as a separate table within their respective queries.

The optimizer handles derived tables, view references, and common table expressions the same way: It avoids unnecessary materialization whenever possible, which enables pushing down conditions from the outer query to derived tables and produces more efficient execution plans. (For an example, see Section 8.2.2.2, “Optimizing Subqueries with Materialization”.)

If merging would result in an outer query block that references more than 61 base tables, the optimizer chooses materialization instead.

The optimizer propagates an `ORDER BY` clause in a derived table or view reference to the outer query block if these conditions are all true:

- The outer query is not grouped or aggregated.
- The outer query does not specify `DISTINCT`, `HAVING`, or `ORDER BY`.
- The outer query has this derived table or view reference as the only source in the `FROM` clause.

Otherwise, the optimizer ignores the `ORDER BY` clause.

The following means are available to influence whether the optimizer attempts to merge derived tables, view references, and common table expressions into the outer query block:

- The `MERGE` and `NO_MERGE` optimizer hints can be used. They apply assuming that no other rule prevents merging. See Section 8.9.3, “Optimizer Hints”.
- Similarly, you can use the `derived_merge` flag of the `optimizer_switch` system variable. See Section 8.9.2, “Switchable Optimizations”. By default, the flag is enabled to permit merging. Disabling the flag prevents merging and avoids `ER_UPDATE_TABLE_USED` errors.

  The `derived_merge` flag also applies to views that contain no `ALGORITHM` clause. Thus, if an `ER_UPDATE_TABLE_USED` error occurs for a view reference that uses an expression equivalent to the subquery, adding `ALGORITHM=TEMPTABLE` to the view definition prevents merging and takes precedence over the `derived_merge` value.

- It is possible to disable merging by using in the subquery any constructs that prevent merging, although these are not as explicit in their effect on materialization. Constructs that prevent merging are the same for derived tables, common table expressions, and view references:
  - Aggregate functions or window functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
  - `DISTINCT`
  - `GROUP BY`
  - `HAVING`
  - `LIMIT`
  - `UNION` or `UNION ALL`
  - Subqueries in the select list
  - Assignments to user variables
  - Refererences only to literal values (in this case, there is no underlying table)

If the optimizer chooses the materialization strategy rather than merging for a derived table, it handles the query as follows:

- The optimizer postpones derived table materialization until its contents are needed during query execution. This improves performance because delaying materialization may result in not having to do it at all. Consider a query that joins the result of a derived table to another table: If the optimizer processes that other table first and finds that it returns no rows, the join need not be carried out further and the optimizer can completely skip materializing the derived table.
- During query execution, the optimizer may add an index to a derived table to speed up row retrieval from it.

Consider the following `EXPLAIN` statement, for a `SELECT` query that contains a derived table:

```
EXPLAIN SELECT * FROM (SELECT * FROM t1) AS derived_t1;
```

The optimizer avoids materializing the derived table by delaying it until the result is needed during `SELECT` execution. In this case, the query is not executed (because it occurs in an `EXPLAIN` statement), so the result is never needed.

Even for queries that are executed, delay of derived table materialization may enable the optimizer to avoid materialization entirely. When this happens, query execution is quicker by the time needed to perform materialization. Consider the following query, which joins the result of a derived table to another table:

```
SELECT *
  FROM t1 JOIN (SELECT t2.f1 FROM t2) AS derived_t2
          ON t1.f2=derived_t2.f1
  WHERE t1.f1 > 0;
```

If the optimization processes `t1` first and the `WHERE` clause produces an empty result, the join must necessarily be empty and the derived table need not be materialized.

For cases when a derived table requires materialization, the optimizer may add an index to the materialized table to speed up access to it. If such an index enables `ref` access to the table, it can greatly reduce amount of data read during query execution. Consider the following query:

```
SELECT *
 FROM t1 JOIN (SELECT DISTINCT f1 FROM t2) AS derived_t2
         ON t1.f1=derived_t2.f1;
```

The optimizer constructs an index over column `f1` from `derived_t2` if doing so would enable use of `ref` access for the lowest cost execution plan. After adding the index, the optimizer can treat the materialized derived table the same as a regular table with an index, and it benefits similarly from the generated index. The overhead of index creation is negligible compared to the cost of query execution without the index. If `ref` access would result in higher cost than some other access method, the optimizer creates no index and loses nothing.

For optimizer trace output, a merged derived table or view reference is not shown as a node. Only its underlying tables appear in the top query's plan.

What is true for materialization of derived tables is also true for common table expressions (CTEs). In addition, the following considerations pertain specifically to CTEs.

If a CTE is materialized by a query, it is materialized once for the query, even if the query references it several times.

A recursive CTE is always materialized.

If a CTE is materialized, the optimizer automatically adds relevant indexes if it estimates that indexing will speed up access by the top-level statement to the CTE. This is similar to automatic indexing of derived tables, except that if the CTE is referenced multiple times, the optimizer may create multiple indexes, to speed up access by each reference in the most appropriate way.

The `MERGE` and `NO_MERGE` optimizer hints can be applied to CTEs. Each CTE reference in the top-level statement can have its own hint, permitting CTE references to be selectively merged or materialized. The following statement uses hints to indicate that `cte1` should be merged and `cte2` should be materialized:

```
WITH
  cte1 AS (SELECT a, b FROM table1),
  cte2 AS (SELECT c, d FROM table2)
SELECT /*+ MERGE(cte1) NO_MERGE(cte2) */ cte1.b, cte2.d
FROM cte1 JOIN cte2
WHERE cte1.a = cte2.c;
```

The `ALGORITHM` clause for `CREATE VIEW` does not affect materialization for any `WITH` clause preceding the `SELECT` statement in the view definition. Consider this statement:

```
CREATE ALGORITHM={TEMPTABLE|MERGE} VIEW v1 AS WITH ... SELECT ...
```

The `ALGORITHM` value affects materialization only of the `SELECT`, not the `WITH` clause.

Prior to MySQL 8.0.16, if `internal_tmp_disk_storage_engine=MYISAM`, an error occurred for any attempt to materialize a CTE using an on-disk temporary table, since for CTEs, the storage engine used for on-disk internal temporary tables could not be `MyISAM`. Beginning with MySQL 8.0.16, this is no longer an issue, since `TempTable` now always uses `InnoDB` for on-disk internal temporary tables.

As mentioned previously, a CTE, if materialized, is materialized once, even if referenced multiple times. To indicate one-time materialization, optimizer trace output contains an occurrence of `creating_tmp_table` plus one or more occurrences of `reusing_tmp_table`.

CTEs are similar to derived tables, for which the `materialized_from_subquery` node follows the reference. This is true for a CTE that is referenced multiple times, so there is no duplication of

`materialized_from_subquery` nodes (which would give the impression that the subquery is executed multiple times, and produce unnecessarily verbose output). Only one reference to the CTE has a complete `materialized_from_subquery` node with the description of its subquery plan. Other references have a reduced `materialized_from_subquery` node. The same idea applies to `EXPLAIN` output in `TRADITIONAL` format: Subqueries for other references are not shown.

8.2.3 Optimizing INFORMATION_SCHEMA Queries

Applications that monitor databases may make frequent use of `INFORMATION_SCHEMA` tables. To write queries for these tables most efficiently, use the following general guidelines:

- Try to query only `INFORMATION_SCHEMA` tables that are views on data dictionary tables.
- Try to query only for static metadata. Selecting columns or using retrieval conditions for dynamic metadata along with static metadata adds overhead to process the dynamic metadata.

Note

Comparison behavior for database and table names in `INFORMATION_SCHEMA` queries might differ from what you expect. For details, see Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”.

These `INFORMATION_SCHEMA` tables are implemented as views on data dictionary tables, so queries on them retrieve information from the data dictionary:

```
CHARACTER_SETS
CHECK_CONSTRAINTS
COLLATIONS
COLLATION_CHARACTER_SET_APPLICABILITY
COLUMNS
EVENTS
FILES
INNODB_COLUMNS
INNODB_DATAFILES
INNODB_FIELDS
INNODB_FOREIGN
INNODB_FOREIGN_COLS
INNODB_INDEXES
INNODB_TABLES
INNODB_TABLESPACES
INNODB_TABLESPACES_BRIEF
INNODB_TABLESTATS
KEY_COLUMN_USAGE
PARAMETERS
PARTITIONS
REFERENTIAL_CONSTRAINTS
RESOURCE_GROUPS
ROUTINES
SCHEMATA
STATISTICS
TABLES
TABLE_CONSTRAINTS
TRIGGERS
VIEWS
VIEW_ROUTINE_USAGE
VIEW_TABLE_USAGE
```

Some types of values, even for a non-view `INFORMATION_SCHEMA` table, are retrieved by lookups from the data dictionary. This includes values such as database and table names, table types, and storage engines.

Some INFORMATION_SCHEMA tables contain columns that provide table statistics:

```
STATISTICS.CARDINALITY
TABLES.AUTO_INCREMENT
TABLES.AVG_ROW_LENGTH
TABLES.CHECKSUM
TABLES.CHECK_TIME
TABLES.CREATE_TIME
TABLES.DATA_FREE
TABLES.DATA_LENGTH
TABLES.INDEX_LENGTH
TABLES.MAX_DATA_LENGTH
TABLES.TABLE_ROWS
TABLES.UPDATE_TIME
```

Those columns represent dynamic table metadata; that is, information that changes as table contents change.

By default, MySQL retrieves cached values for those columns from the mysql.index_stats and mysql.table_stats dictionary tables when the columns are queried, which is more efficient than retrieving statistics directly from the storage engine. If cached statistics are not available or have expired, MySQL retrieves the latest statistics from the storage engine and caches them in the mysql.index_stats and mysql.table_stats dictionary tables. Subsequent queries retrieve the cached statistics until the cached statistics expire.

The information_schema_stats_expiry session variable defines the period of time before cached statistics expire. The default is 86400 seconds (24 hours), but the time period can be extended to as much as one year.

To update cached values at any time for a given table, use ANALYZE TABLE.

Querying statistics columns does not store or update statistics in the mysql.index_stats and mysql.table_stats dictionary tables under these circumstances:

- When cached statistics have not expired.
- When information_schema_stats_expiry is set to 0.
- When the server is started in read_only, super_read_only, transaction_read_only, or innodb_read_only mode.
- When the query also fetches Performance Schema data.

information_schema_stats_expiry is a session variable, and each client session can define its own expiration value. Statistics that are retrieved from the storage engine and cached by one session are available to other sessions.

Note

If the innodb_read_only system variable is enabled, ANALYZE TABLE may fail because it cannot update statistics tables in the data dictionary, which use InnoDB. For ANALYZE TABLE operations that update the key distribution, failure may occur even if the operation updates the table itself (for example, if it is a MyISAM table). To obtain the updated distribution statistics, set information_schema_stats_expiry=0.

For INFORMATION_SCHEMA tables implemented as views on data dictionary tables, indexes on the underlying data dictionary tables permit the optimizer to construct efficient query execution plans. To see

the choices made by the optimizer, use `EXPLAIN`. To also see the query used by the server to execute an `INFORMATION_SCHEMA` query, use `SHOW WARNINGS` immediately following `EXPLAIN`.

Consider this statement, which identifies collations for the `utf8mb4` character set:

```
mysql> SELECT COLLATION_NAME
       FROM INFORMATION_SCHEMA.COLLATION_CHARACTER_SET_APPLICABILITY
       WHERE CHARACTER_SET_NAME = 'utf8mb4';
+----------------------------+
| COLLATION_NAME             |
+----------------------------+
| utf8mb4_general_ci         |
| utf8mb4_bin                |
| utf8mb4_unicode_ci         |
| utf8mb4_icelandic_ci       |
| utf8mb4_latvian_ci         |
| utf8mb4_romanian_ci        |
| utf8mb4_slovenian_ci       |
...
```

How does the server process that statement? To find out, use `EXPLAIN`:

```
mysql> EXPLAIN SELECT COLLATION_NAME
       FROM INFORMATION_SCHEMA.COLLATION_CHARACTER_SET_APPLICABILITY
       WHERE CHARACTER_SET_NAME = 'utf8mb4'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: cs
   partitions: NULL
         type: const
possible_keys: PRIMARY,name
          key: name
      key_len: 194
          ref: const
         rows: 1
     filtered: 100.00
        Extra: Using index
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: col
   partitions: NULL
         type: ref
possible_keys: character_set_id
          key: character_set_id
      key_len: 8
          ref: const
         rows: 68
     filtered: 100.00
        Extra: NULL
2 rows in set, 1 warning (0.01 sec)
```

To see the query used to statisfy that statement, use `SHOW WARNINGS`:

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select `mysql`.`col`.`name` AS `COLLATION_NAME`
         from `mysql`.`character_sets` `cs`
         join `mysql`.`collations` `col`
         where ((`mysql`.`col`.`character_set_id` = '45')
```

```
        and ('utf8mb4' = 'utf8mb4'))
```

As indicated by `SHOW WARNINGS`, the server handles the query on `COLLATION_CHARACTER_SET_APPLICABILITY` as a query on the `character_sets` and `collations` data dictionary tables in the `mysql` system database.

8.2.4 Optimizing Performance Schema Queries

Applications that monitor databases may make frequent use of Performance Schema tables. To write queries for these tables most efficiently, take advantage of their indexes. For example, include a `WHERE` clause that restricts retrieved rows based on comparison to specific values in an indexed column.

Most Performance Schema tables have indexes. Tables that do not are those that normally contain few rows or are unlikely to be queried frequently. Performance Schema indexes give the optimizer access to execution plans other than full table scans. These indexes also improve performance for related objects, such as `sys` schema views that use those tables.

To see whether a given Performance Schema table has indexes and what they are, use `SHOW INDEX` or `SHOW CREATE TABLE`:

```
mysql> SHOW INDEX FROM performance_schema.accounts\G
*************************** 1. row ***************************
        Table: accounts
   Non_unique: 0
     Key_name: ACCOUNT
 Seq_in_index: 1
  Column_name: USER
    Collation: NULL
  Cardinality: NULL
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: HASH
      Comment:
Index_comment:
      Visible: YES
*************************** 2. row ***************************
        Table: accounts
   Non_unique: 0
     Key_name: ACCOUNT
 Seq_in_index: 2
  Column_name: HOST
    Collation: NULL
  Cardinality: NULL
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: HASH
      Comment:
Index_comment:
      Visible: YES

mysql> SHOW CREATE TABLE performance_schema.rwlock_instances\G
*************************** 1. row ***************************
       Table: rwlock_instances
Create Table: CREATE TABLE `rwlock_instances` (
  `NAME` varchar(128) NOT NULL,
  `OBJECT_INSTANCE_BEGIN` bigint(20) unsigned NOT NULL,
  `WRITE_LOCKED_BY_THREAD_ID` bigint(20) unsigned DEFAULT NULL,
  `READ_LOCKED_BY_COUNT` int(10) unsigned NOT NULL,
  PRIMARY KEY (`OBJECT_INSTANCE_BEGIN`),
  KEY `NAME` (`NAME`),
  KEY `WRITE_LOCKED_BY_THREAD_ID` (`WRITE_LOCKED_BY_THREAD_ID`)
```

```
) ENGINE=PERFORMANCE_SCHEMA DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

To see the execution plan for a Performance Schema query and whether it uses any indexes, use `EXPLAIN`:

```
mysql> EXPLAIN SELECT * FROM performance_schema.accounts
       WHERE (USER,HOST) = ('root','localhost')\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: accounts
   partitions: NULL
         type: const
possible_keys: ACCOUNT
          key: ACCOUNT
      key_len: 278
          ref: const,const
         rows: 1
     filtered: 100.00
        Extra: NULL
```

The `EXPLAIN` output indicates that the optimizer uses the `accounts` table `ACCOUNT` index that comprises the `USER` and `HOST` columns.

Performance Schema indexes are virtual: They are a construct of the Performance Schema storage engine and use no memory or disk storage. The Performance Schema reports index information to the optimizer so that it can construct efficient execution plans. The Performance Schema in turn uses optimizer information about what to look for (for example, a particular key value), so that it can perform efficient lookups without building actual index structures. This implementation provides two important benefits:

- It entirely avoids the maintenance cost normally incurred for tables that undergo frequent updates.
- It reduces at an early stage of query execution the amount of data retrieved. For conditions on the indexed columns, the Performance Schema efficiently returns only table rows that satisfy the query conditions. Without an index, the Performance Schema would return all rows in the table, requiring that the optimizer later evaluate the conditions against each row to produce the final result.

Performance Schema indexes are predefined and cannot be dropped, added, or altered.

Performance Schema indexes are similar to hash indexes. For example:

- They are used only for equality comparisons that use the `=` or `<=>` operators.
- They are unordered. If a query result must have specific row ordering characteristics, include an `ORDER BY` clause.

For additional information about hash indexes, see Section 8.3.9, “Comparison of B-Tree and Hash Indexes”.

8.2.5 Optimizing Data Change Statements

This section explains how to speed up data change statements: `INSERT`, `UPDATE`, and `DELETE`. Traditional OLTP applications and modern web applications typically do many small data change operations, where concurrency is vital. Data analysis and reporting applications typically run data change operations that affect many rows at once, where the main considerations is the I/O to write large amounts of data and keep indexes up-to-date. For inserting and updating large volumes of data (known in the industry as ETL, for “extract-transform-load”), sometimes you use other SQL statements or external commands, that mimic the effects of `INSERT`, `UPDATE`, and `DELETE` statements.

8.2.5.1 Optimizing INSERT Statements

To optimize insert speed, combine many small operations into a single large operation. Ideally, you make a single connection, send the data for many new rows at once, and delay all index updates and consistency checking until the very end.

The time required for inserting a row is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)
- Sending query to server: (2)
- Parsing query: (2)
- Inserting row: (1 × size of row)
- Inserting indexes: (1 × number of indexes)
- Closing: (1)

This does not take into consideration the initial overhead to open tables, which is done once for each concurrently running query.

The size of the table slows down the insertion of indexes by log *N*, assuming B-tree indexes.

You can use the following methods to speed up inserts:

- If you are inserting many rows from the same client at the same time, use `INSERT` statements with multiple `VALUES` lists to insert several rows at a time. This is considerably faster (many times faster in some cases) than using separate single-row `INSERT` statements. If you are adding data to a nonempty table, you can tune the `bulk_insert_buffer_size` variable to make data insertion even faster. See Section 5.1.8, “Server System Variables”.
- When loading a table from a text file, use `LOAD DATA`. This is usually 20 times faster than using `INSERT` statements. See Section 13.2.7, “LOAD DATA Syntax”.
- Take advantage of the fact that columns have default values. Insert values explicitly only when the value to be inserted differs from the default. This reduces the parsing that MySQL must do and improves the insert speed.
- See Section 8.5.5, “Bulk Data Loading for InnoDB Tables” for tips specific to `InnoDB` tables.
- See Section 8.6.2, “Bulk Data Loading for MyISAM Tables” for tips specific to `MyISAM` tables.

8.2.5.2 Optimizing UPDATE Statements

An update statement is optimized like a `SELECT` query with the additional overhead of a write. The speed of the write depends on the amount of data being updated and the number of indexes that are updated. Indexes that are not changed do not get updated.

Another way to get fast updates is to delay updates and then do many updates in a row later. Performing multiple updates together is much quicker than doing one at a time if you lock the table.

For a `MyISAM` table that uses dynamic row format, updating a row to a longer total length may split the row. If you do this often, it is very important to use `OPTIMIZE TABLE` occasionally. See Section 13.7.3.4, “OPTIMIZE TABLE Syntax”.

8.2.5.3 Optimizing DELETE Statements

The time required to delete individual rows in a `MyISAM` table is exactly proportional to the number of indexes. To delete rows more quickly, you can increase the size of the key cache by increasing the `key_buffer_size` system variable. See Section 5.1.1, “Configuring the Server”.

To delete all rows from a `MyISAM` table, `TRUNCATE TABLE tbl_name` is faster than `DELETE FROM tbl_name`. Truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction or active table lock. See Section 13.1.37, “TRUNCATE TABLE Syntax”.

8.2.6 Optimizing Database Privileges

The more complex your privilege setup, the more overhead applies to all SQL statements. Simplifying the privileges established by `GRANT` statements enables MySQL to reduce permission-checking overhead when clients execute statements. For example, if you do not grant any table-level or column-level privileges, the server need not ever check the contents of the `tables_priv` and `columns_priv` tables. Similarly, if you place no resource limits on any accounts, the server does not have to perform resource counting. If you have a very high statement-processing load, consider using a simplified grant structure to reduce permission-checking overhead.

8.2.7 Other Optimization Tips

This section lists a number of miscellaneous tips for improving query processing speed:

- If your application makes several database requests to perform related updates, combining the statements into a stored routine can help performance. Similarly, if your application computes a single result based on several column values or large volumes of data, combining the computation into a UDF (user-defined function) can help performance. The resulting fast database operations are then available to be reused by other queries, applications, and even code written in different programming languages. See Section 24.2, “Using Stored Routines” and Section 29.4, “Adding New Functions to MySQL” for more information.
- To fix any compression issues that occur with `ARCHIVE` tables, use `OPTIMIZE TABLE`. See Section 16.5, “The ARCHIVE Storage Engine”.
- If possible, classify reports as “live” or as “statistical”, where data needed for statistical reports is created only from summary tables that are generated periodically from the live data.
- If you have data that does not conform well to a rows-and-columns table structure, you can pack and store data into a `BLOB` column. In this case, you must provide code in your application to pack and unpack information, but this might save I/O operations to read and write the sets of related values.
- With Web servers, store images and other binary assets as files, with the path name stored in the database rather than the file itself. Most Web servers are better at caching files than database contents, so using files is generally faster. (Although you must handle backups and storage issues yourself in this case.)
- If you need really high speed, look at the low-level MySQL interfaces. For example, by accessing the MySQL `InnoDB` or `MyISAM` storage engine directly, you could get a substantial speed increase compared to using the SQL interface.
- Replication can provide a performance benefit for some operations. You can distribute client retrievals among replication servers to split up the load. To avoid slowing down the master while making backups, you can make backups using a slave server. See Chapter 17, *Replication*.

8.3 Optimization and Indexes

The best way to improve the performance of `SELECT` operations is to create indexes on one or more of the columns that are tested in the query. The index entries act like pointers to the table rows, allowing the query to quickly determine which rows match a condition in the `WHERE` clause, and retrieve the other column values for those rows. All MySQL data types can be indexed.

Although it can be tempting to create an indexes for every possible column used in a query, unnecessary indexes waste space and waste time for MySQL to determine which indexes to use. Indexes also add to the cost of inserts, updates, and deletes because each index must be updated. You must find the right balance to achieve fast queries using the optimal set of indexes.

8.3.1 How MySQL Uses Indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. This is much faster than reading every row sequentially.

Most MySQL indexes (`PRIMARY KEY`, `UNIQUE`, `INDEX`, and `FULLTEXT`) are stored in B-trees. Exceptions: Indexes on spatial data types use R-trees; `MEMORY` tables also support hash indexes; `InnoDB` uses inverted lists for `FULLTEXT` indexes.

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in `MEMORY` tables) are described in Section 8.3.9, “Comparison of B-Tree and Hash Indexes”.

MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.6, “Multiple-Column Indexes”.
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

  For comparisons between nonbinary string columns, both columns should use the same character set. For example, comparing a `utf8` column with a `latin1` column precludes use of an index.

  Comparison of dissimilar columns (comparing a string column to a temporal or numeric column, for example) may prevent use of indexes if values cannot be compared directly without conversion. For a given value such as `1` in the numeric column, it might compare equal to any number of values in the string column such as `'1'`, `' 1'`, `'00001'`, or `'01.e1'`. This rules out use of any indexes for the string column.
- To find the `MIN()` or `MAX()` value for a specific indexed column *`key_col`*. This is optimized by a preprocessor that checks whether you are using `WHERE` *`key_part_N = constant`* on all key parts that occur before *`key_col`* in the index. In this case, MySQL does a single key lookup for each `MIN()`

or `MAX()` expression and replaces it with a constant. If all expressions are replaced with constants, the query returns at once. For example:

```
SELECT MIN(key_part2),MAX(key_part2)
  FROM tbl_name WHERE key_part1=10;
```

- To sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable index (for example, `ORDER BY key_part1, key_part2`). If all key parts are followed by `DESC`, the key is read in reverse order. (Or, if the index is a descending index, the key is read in forward order.) See Section 8.2.1.16, “ORDER BY Optimization”, Section 8.2.1.17, “GROUP BY Optimization”, and Section 8.3.13, “Descending Indexes”.

- In some cases, a query can be optimized to retrieve values without consulting the data rows. (An index that provides all the necessary results for a query is called a covering index.) If a query uses from a table only columns that are included in some index, the selected values can be retrieved from the index tree for greater speed:

```
SELECT key_part3 FROM tbl_name
  WHERE key_part1=1
```

Indexes are less important for queries on small tables, or big tables where report queries process most or all of the rows. When a query needs to access most of the rows, reading sequentially is faster than working through an index. Sequential reads minimize disk seeks, even if not all the rows are needed for the query. See Section 8.2.1.23, “Avoiding Full Table Scans” for details.

8.3.2 Primary Key Optimization

The primary key for a table represents the column or set of columns that you use in your most vital queries. It has an associated index, for fast query performance. Query performance benefits from the `NOT NULL` optimization, because it cannot include any `NULL` values. With the `InnoDB` storage engine, the table data is physically organized to do ultra-fast lookups and sorts based on the primary key column or columns.

If your table is big and important, but does not have an obvious column or set of columns to use as a primary key, you might create a separate column with auto-increment values to use as the primary key. These unique IDs can serve as pointers to corresponding rows in other tables when you join tables using foreign keys.

8.3.3 SPATIAL Index Optimization

MySQL permits creation of `SPATIAL` indexes on `NOT NULL` geometry-valued columns (see Section 11.5.10, “Creating Spatial Indexes”). The optimizer checks the `SRID` attribute for indexed columns to determine which spatial reference system (SRS) to use for comparisons, and uses calculations appropriate to the SRS. (Prior to MySQL 8.0, the optimizer performs comparisons of `SPATIAL` index values using Cartesian calculations; the results of such operations are undefined if the column contains values with non-Cartesian SRIDs.)

For comparisons to work properly, each column in a `SPATIAL` index must be SRID-restricted. That is, the column definition must include an explicit `SRID` attribute, and all column values must have the same SRID.

The optimizer considers `SPATIAL` indexes only for SRID-restricted columns:

- Indexes on columns restricted to a Cartesian SRID enable Cartesian bounding box computations.

- Indexes on columns restricted to a geographic SRID enable geographic bounding box computations.

The optimizer ignores `SPATIAL` indexes on columns that have no `SRID` attribute (and thus are not SRID-restricted). MySQL still maintains such indexes, as follows:

- They are updated for table modifications (`INSERT`, `UPDATE`, `DELETE`, and so forth). Updates occur as though the index was Cartesian, even though the column might contain a mix of Cartesian and geographical values.
- They exist only for backward compatibility (for example, the ability to perform a dump in MySQL 5.7 and restore in MySQL 8.0). Because `SPATIAL` indexes on columns that are not SRID-restricted are of no use to the optimizer, each such column should be modified:
  - Verify that all values within the column have the same SRID. To determine the SRIDs contained in a geometry column *`col_name`*, use the following query:

    ```
    SELECT DISTINCT ST_SRID(col_name) FROM tbl_name;
    ```

    If the query returns more than one row, the column contains a mix of SRIDs. In that case, modify its contents so all values have the same SRID.
  - Redefine the column to have an explicit `SRID` attribute.
  - Recreate the `SPATIAL` index.

8.3.4 Foreign Key Optimization

If a table has many columns, and you query many different combinations of columns, it might be efficient to split the less-frequently used data into separate tables with a few columns each, and relate them back to the main table by duplicating the numeric ID column from the main table. That way, each small table can have a primary key for fast lookups of its data, and you can query just the set of columns that you need using a join operation. Depending on how the data is distributed, the queries might perform less I/O and take up less cache memory because the relevant columns are packed together on disk. (To maximize performance, queries try to read as few data blocks as possible from disk; tables with only a few columns can fit more rows in each data block.)

8.3.5 Column Indexes

The most common type of index involves a single column, storing copies of the values from that column in a data structure, allowing fast lookups for the rows with the corresponding column values. The B-tree data structure lets the index quickly find a specific value, a set of values, or a range of values, corresponding to operators such as `=`, `>`, `≤`, `BETWEEN`, `IN`, and so on, in a `WHERE` clause.

The maximum number of indexes per table and the maximum index length is defined per storage engine. See Chapter 15, *The InnoDB Storage Engine*, and Chapter 16, *Alternative Storage Engines*. All storage engines support at least 16 indexes per table and a total index length of at least 256 bytes. Most storage engines have higher limits.

For additional information about column indexes, see Section 13.1.15, “CREATE INDEX Syntax”.

- Index Prefixes
- FULLTEXT Indexes
- Spatial Indexes
- Indexes in the MEMORY Storage Engine

Index Prefixes

With *col_name(N)* syntax in an index specification for a string column, you can create an index that uses only the first *N* characters of the column. Indexing only a prefix of column values in this way can make the index file much smaller. When you index a BLOB or TEXT column, you *must* specify a prefix length for the index. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 767 bytes long for InnoDB tables that use the REDUNDANT or COMPACT row format. The prefix length limit is 3072 bytes for InnoDB tables that use the DYNAMIC or COMPRESSED row format. For MyISAM tables, the prefix length limit is 1000 bytes.

Note

Prefix limits are measured in bytes, whereas the prefix length in CREATE TABLE, ALTER TABLE, and CREATE INDEX statements is interpreted as number of characters for nonbinary string types (CHAR, VARCHAR, TEXT) and number of bytes for binary string types (BINARY, VARBINARY, BLOB). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

If a search term exceeds the index prefix length, the index is used to exclude non-matching rows, and the remaining rows are examined for possible matches.

For additional information about index prefixes, see Section 13.1.15, “CREATE INDEX Syntax”.

FULLTEXT Indexes

FULLTEXT indexes are used for full-text searches. Only the InnoDB and MyISAM storage engines support FULLTEXT indexes and only for CHAR, VARCHAR, and TEXT columns. Indexing always takes place over the entire column and column prefix indexing is not supported. For details, see Section 12.9, “Full-Text Search Functions”.

Optimizations are applied to certain kinds of FULLTEXT queries against single InnoDB tables. Queries with these characteristics are particularly efficient:

- FULLTEXT queries that only return the document ID, or the document ID and the search rank.
- FULLTEXT queries that sort the matching rows in descending order of score and apply a LIMIT clause to take the top N matching rows. For this optimization to apply, there must be no WHERE clauses and only a single ORDER BY clause in descending order.
- FULLTEXT queries that retrieve only the COUNT(*) value of rows matching a search term, with no additional WHERE clauses. Code the WHERE clause as WHERE MATCH(*text*) AGAINST ('*other_text*'), without any > 0 comparison operator.

For queries that contain full-text expressions, MySQL evaluates those expressions during the optimization phase of query execution. The optimizer does not just look at full-text expressions and make estimates, it actually evaluates them in the process of developing an execution plan.

An implication of this behavior is that EXPLAIN for full-text queries is typically slower than for non-full-text queries for which no expression evaluation occurs during the optimization phase.

EXPLAIN for full-text queries may show Select tables optimized away in the Extra column due to matching occurring during optimization; in this case, no table access need occur during later execution.

Spatial Indexes

You can create indexes on spatial data types. `MyISAM` and `InnoDB` support R-tree indexes on spatial types. Other storage engines use B-trees for indexing spatial types (except for `ARCHIVE`, which does not support spatial type indexing).

Indexes in the MEMORY Storage Engine

The `MEMORY` storage engine uses `HASH` indexes by default, but also supports `BTREE` indexes.

8.3.6 Multiple-Column Indexes

MySQL can create composite indexes (that is, indexes on multiple columns). An index may consist of up to 16 columns. For certain data types, you can index a prefix of the column (see Section 8.3.5, “Column Indexes”).

MySQL can use multiple-column indexes for queries that test all the columns in the index, or queries that test just the first column, the first two columns, the first three columns, and so on. If you specify the columns in the right order in the index definition, a single composite index can speed up several kinds of queries on the same table.

A multiple-column index can be considered a sorted array, the rows of which contain values that are created by concatenating the values of the indexed columns.

Note

As an alternative to a composite index, you can introduce a column that is “hashed” based on information from other columns. If this column is short, reasonably unique, and indexed, it might be faster than a “wide” index on many columns. In MySQL, it is very easy to use this extra column:

```
SELECT * FROM tbl_name
  WHERE hash_col=MD5(CONCAT(val1,val2))
  AND col1=val1 AND col2=val2;
```

Suppose that a table has the following specification:

```
CREATE TABLE test (
    id         INT NOT NULL,
    last_name  CHAR(30) NOT NULL,
    first_name CHAR(30) NOT NULL,
    PRIMARY KEY (id),
    INDEX name (last_name,first_name)
);
```

The `name` index is an index over the `last_name` and `first_name` columns. The index can be used for lookups in queries that specify values in a known range for combinations of `last_name` and `first_name` values. It can also be used for queries that specify just a `last_name` value because that column is a leftmost prefix of the index (as described later in this section). Therefore, the `name` index is used for lookups in the following queries:

```
SELECT * FROM test WHERE last_name='Jones';

SELECT * FROM test
  WHERE last_name='Jones' AND first_name='John';
```

```
SELECT * FROM test
  WHERE last_name='Jones'
  AND (first_name='John' OR first_name='Jon');

SELECT * FROM test
  WHERE last_name='Jones'
  AND first_name >='M' AND first_name < 'N';
```

However, the `name` index is *not* used for lookups in the following queries:

```
SELECT * FROM test WHERE first_name='John';

SELECT * FROM test
  WHERE last_name='Jones' OR first_name='John';
```

Suppose that you issue the following `SELECT` statement:

```
SELECT * FROM tbl_name
  WHERE col1=val1 AND col2=val2;
```

If a multiple-column index exists on `col1` and `col2`, the appropriate rows can be fetched directly. If separate single-column indexes exist on `col1` and `col2`, the optimizer attempts to use the Index Merge optimization (see Section 8.2.1.3, “Index Merge Optimization”), or attempts to find the most restrictive index by deciding which index excludes more rows and using that index to fetch the rows.

If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`.

MySQL cannot use the index to perform lookups if the columns do not form a leftmost prefix of the index. Suppose that you have the `SELECT` statements shown here:

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

If an index exists on `(col1, col2, col3)`, only the first two queries use the index. The third and fourth queries do involve indexed columns, but do not use an index to perform lookups because `(col2)` and `(col2, col3)` are not leftmost prefixes of `(col1, col2, col3)`.

8.3.7 Verifying Index Usage

Always check whether all your queries really use the indexes that you have created in the tables. Use the `EXPLAIN` statement, as described in Section 8.8.1, “Optimizing Queries with EXPLAIN”.

8.3.8 InnoDB and MyISAM Index Statistics Collection

Storage engines collect statistics about tables for use by the optimizer. Table statistics are based on value groups, where a value group is a set of rows with the same key prefix value. For optimizer purposes, an important statistic is the average value group size.

MySQL uses the average value group size in the following ways:

- To estimate how many rows must be read for each `ref` access

- To estimate how many rows a partial join will produce; that is, the number of rows that an operation of this form will produce:

```
(...) JOIN tbl_name ON tbl_name.key = expr
```

As the average value group size for an index increases, the index is less useful for those two purposes because the average number of rows per lookup increases: For the index to be good for optimization purposes, it is best that each index value target a small number of rows in the table. When a given index value yields a large number of rows, the index is less useful and MySQL is less likely to use it.

The average value group size is related to table cardinality, which is the number of value groups. The `SHOW INDEX` statement displays a cardinality value based on *`N/S`*, where *`N`* is the number of rows in the table and *`S`* is the average value group size. That ratio yields an approximate number of value groups in the table.

For a join based on the `<=>` comparison operator, `NULL` is not treated differently from any other value: `NULL <=> NULL`, just as *`N <=> N`* for any other *`N`*.

However, for a join based on the `=` operator, `NULL` is different from non-`NULL` values: *`expr1 = expr2`* is not true when *`expr1`* or *`expr2`* (or both) are `NULL`. This affects `ref` accesses for comparisons of the form *`tbl_name.key = expr`*: MySQL will not access the table if the current value of *`expr`* is `NULL`, because the comparison cannot be true.

For `=` comparisons, it does not matter how many `NULL` values are in the table. For optimization purposes, the relevant value is the average size of the non-`NULL` value groups. However, MySQL does not currently enable that average size to be collected or used.

For `InnoDB` and `MyISAM` tables, you have some control over collection of table statistics by means of the `innodb_stats_method` and `myisam_stats_method` system variables, respectively. These variables have three possible values, which differ as follows:

- When the variable is set to `nulls_equal`, all `NULL` values are treated as identical (that is, they all form a single value group).

  If the `NULL` value group size is much higher than the average non-`NULL` value group size, this method skews the average value group size upward. This makes index appear to the optimizer to be less useful than it really is for joins that look for non-`NULL` values. Consequently, the `nulls_equal` method may cause the optimizer not to use the index for `ref` accesses when it should.

- When the variable is set to `nulls_unequal`, `NULL` values are not considered the same. Instead, each `NULL` value forms a separate value group of size 1.

  If you have many `NULL` values, this method skews the average value group size downward. If the average non-`NULL` value group size is large, counting `NULL` values each as a group of size 1 causes the optimizer to overestimate the value of the index for joins that look for non-`NULL` values. Consequently, the `nulls_unequal` method may cause the optimizer to use this index for `ref` lookups when other methods may be better.

- When the variable is set to `nulls_ignored`, `NULL` values are ignored.

If you tend to use many joins that use `<=>` rather than `=`, `NULL` values are not special in comparisons and one `NULL` is equal to another. In this case, `nulls_equal` is the appropriate statistics method.

The `innodb_stats_method` system variable has a global value; the `myisam_stats_method` system variable has both global and session values. Setting the global value affects statistics collection for tables from the corresponding storage engine. Setting the session value affects statistics collection only for the

current client connection. This means that you can force a table's statistics to be regenerated with a given method without affecting other clients by setting the session value of `myisam_stats_method`.

To regenerate `MyISAM` table statistics, you can use any of the following methods:

- Execute `myisamchk --stats_method=method_name --analyze`
- Change the table to cause its statistics to go out of date (for example, insert a row and then delete it), and then set `myisam_stats_method` and issue an `ANALYZE TABLE` statement

Some caveats regarding the use of `innodb_stats_method` and `myisam_stats_method`:

- You can force table statistics to be collected explicitly, as just described. However, MySQL may also collect statistics automatically. For example, if during the course of executing statements for a table, some of those statements modify the table, MySQL may collect statistics. (This may occur for bulk inserts or deletes, or some `ALTER TABLE` statements, for example.) If this happens, the statistics are collected using whatever value `innodb_stats_method` or `myisam_stats_method` has at the time. Thus, if you collect statistics using one method, but the system variable is set to the other method when a table's statistics are collected automatically later, the other method will be used.
- There is no way to tell which method was used to generate statistics for a given table.
- These variables apply only to `InnoDB` and `MyISAM` tables. Other storage engines have only one method for collecting table statistics. Usually it is closer to the `nulls_equal` method.

8.3.9 Comparison of B-Tree and Hash Indexes

Understanding the B-tree and hash data structures can help predict how different queries perform on different storage engines that use these data structures in their indexes, particularly for the `MEMORY` storage engine that lets you choose B-tree or hash indexes.

- B-Tree Index Characteristics
- Hash Index Characteristics

B-Tree Index Characteristics

A B-tree index can be used for column comparisons in expressions that use the `=`, `>`, `>=`, `<`, `<=`, or `BETWEEN` operators. The index also can be used for `LIKE` comparisons if the argument to `LIKE` is a constant string that does not start with a wildcard character. For example, the following `SELECT` statements use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

In the first statement, only rows with `'Patrick' <= key_col < 'Patricl'` are considered. In the second statement, only rows with `'Pat' <= key_col < 'Pau'` are considered.

The following `SELECT` statements do not use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

In the first statement, the `LIKE` value begins with a wildcard character. In the second statement, the `LIKE` value is not a constant.

If you use `... LIKE '%string%'` and *`string`* is longer than three characters, MySQL uses the *Turbo Boyer-Moore algorithm* to initialize the pattern for the string and then uses this pattern to perform the search more quickly.

A search using *`col_name`* `IS NULL` employs indexes if *`col_name`* is indexed.

Any index that does not span all `AND` levels in the `WHERE` clause is not used to optimize the query. In other words, to be able to use an index, a prefix of the index must be used in every `AND` group.

The following `WHERE` clauses use indexes:

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3

    /* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2

    /* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5

    /* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

These `WHERE` clauses do *not* use indexes:

```
    /* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2

    /*  Index is not used in both parts of the WHERE clause  */
... WHERE index=1 OR A=10

    /* No index spans all rows  */
... WHERE index_part1=1 OR index_part2=10
```

Sometimes MySQL does not use an index, even if one is available. One circumstance under which this occurs is when the optimizer estimates that using the index would require MySQL to access a very large percentage of the rows in the table. (In this case, a table scan is likely to be much faster because it requires fewer seeks.) However, if such a query uses `LIMIT` to retrieve only some of the rows, MySQL uses an index anyway, because it can much more quickly find the few rows to return in the result.

Hash Index Characteristics

Hash indexes have somewhat different characteristics from those just discussed:

- They are used only for equality comparisons that use the `=` or `<=>` operators (but are *very* fast). They are not used for comparison operators such as `<` that find a range of values. Systems that rely on this type of single-value lookup are known as “key-value stores”; to use MySQL for such applications, use hash indexes wherever possible.
- The optimizer cannot use a hash index to speed up `ORDER BY` operations. (This type of index cannot be used to search for the next entry in order.)
- MySQL cannot determine approximately how many rows there are between two values (this is used by the range optimizer to decide which index to use). This may affect some queries if you change a `MyISAM` or `InnoDB` table to a hash-indexed `MEMORY` table.
- Only whole keys can be used to search for a row. (With a B-tree index, any leftmost prefix of the key can be used to find rows.)

8.3.10 Use of Index Extensions

InnoDB automatically extends each secondary index by appending the primary key columns to it. Consider this table definition:

```
CREATE TABLE t1 (
  i1 INT NOT NULL DEFAULT 0,
  i2 INT NOT NULL DEFAULT 0,
  d DATE DEFAULT NULL,
  PRIMARY KEY (i1, i2),
  INDEX k_d (d)
) ENGINE = InnoDB;
```

This table defines the primary key on columns (i1, i2). It also defines a secondary index k_d on column (d), but internally InnoDB extends this index and treats it as columns (d, i1, i2).

The optimizer takes into account the primary key columns of the extended secondary index when determining how and whether to use that index. This can result in more efficient query execution plans and better performance.

The optimizer can use extended secondary indexes for ref, range, and index_merge index access, for Loose Index Scan access, for join and sorting optimization, and for MIN()/MAX() optimization.

The following example shows how execution plans are affected by whether the optimizer uses extended secondary indexes. Suppose that t1 is populated with these rows:

```
INSERT INTO t1 VALUES
(1, 1, '1998-01-01'), (1, 2, '1999-01-01'),
(1, 3, '2000-01-01'), (1, 4, '2001-01-01'),
(1, 5, '2002-01-01'), (2, 1, '1998-01-01'),
(2, 2, '1999-01-01'), (2, 3, '2000-01-01'),
(2, 4, '2001-01-01'), (2, 5, '2002-01-01'),
(3, 1, '1998-01-01'), (3, 2, '1999-01-01'),
(3, 3, '2000-01-01'), (3, 4, '2001-01-01'),
(3, 5, '2002-01-01'), (4, 1, '1998-01-01'),
(4, 2, '1999-01-01'), (4, 3, '2000-01-01'),
(4, 4, '2001-01-01'), (4, 5, '2002-01-01'),
(5, 1, '1998-01-01'), (5, 2, '1999-01-01'),
(5, 3, '2000-01-01'), (5, 4, '2001-01-01'),
(5, 5, '2002-01-01');
```

Now consider this query:

```
EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'
```

The execution plan depends on whether the extended index is used.

When the optimizer does not consider index extensions, it treats the index k_d as only (d). EXPLAIN for the query produces this result:

```
mysql> EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: ref
possible_keys: PRIMARY,k_d
          key: k_d
      key_len: 4
          ref: const
         rows: 5
```

```
        Extra: Using where; Using index
```

When the optimizer takes index extensions into account, it treats `k_d` as `(d, i1, i2)`. In this case, it can use the leftmost index prefix `(d, i1)` to produce a better execution plan:

```
mysql> EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: ref
possible_keys: PRIMARY,k_d
          key: k_d
      key_len: 8
          ref: const,const
         rows: 1
        Extra: Using index
```

In both cases, `key` indicates that the optimizer will use secondary index `k_d` but the `EXPLAIN` output shows these improvements from using the extended index:

- `key_len` goes from 4 bytes to 8 bytes, indicating that key lookups use columns `d` and `i1`, not just `d`.
- The `ref` value changes from `const` to `const,const` because the key lookup uses two key parts, not one.
- The `rows` count decreases from 5 to 1, indicating that `InnoDB` should need to examine fewer rows to produce the result.
- The `Extra` value changes from `Using where; Using index` to `Using index`. This means that rows can be read using only the index, without consulting columns in the data row.

Differences in optimizer behavior for use of extended indexes can also be seen with `SHOW STATUS`:

```
FLUSH TABLE t1;
FLUSH STATUS;
SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01';
SHOW STATUS LIKE 'handler_read%'
```

The preceding statements include `FLUSH TABLES` and `FLUSH STATUS` to flush the table cache and clear the status counters.

Without index extensions, `SHOW STATUS` produces this result:

```
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 0     |
| Handler_read_key      | 1     |
| Handler_read_last     | 0     |
| Handler_read_next     | 5     |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 0     |
+-----------------------+-------+
```

With index extensions, `SHOW STATUS` produces this result. The `Handler_read_next` value decreases from 5 to 1, indicating more efficient use of the index:

```
```

```
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 0     |
| Handler_read_key      | 1     |
| Handler_read_last     | 0     |
| Handler_read_next     | 1     |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 0     |
+-----------------------+-------+
```

The `use_index_extensions` flag of the `optimizer_switch` system variable permits control over whether the optimizer takes the primary key columns into account when determining how to use an `InnoDB` table's secondary indexes. By default, `use_index_extensions` is enabled. To check whether disabling use of index extensions will improve performance, use this statement:

```
SET optimizer_switch = 'use_index_extensions=off';
```

Use of index extensions by the optimizer is subject to the usual limits on the number of key parts in an index (16) and the maximum key length (3072 bytes).

8.3.11 Optimizer Use of Generated Column Indexes

MySQL supports indexes on generated columns. For example:

```
CREATE TABLE t1 (f1 INT, gc INT AS (f1 + 1) STORED, INDEX (gc));
```

The generated column, `gc`, is defined as the expression `f1 + 1`. The column is also indexed and the optimizer can take that index into account during execution plan construction. In the following query, the `WHERE` clause refers to `gc` and the optimizer considers whether the index on that column yields a more efficient plan:

```
SELECT * FROM t1 WHERE gc > 9;
```

The optimizer can use indexes on generated columns to generate execution plans, even in the absence of direct references in queries to those columns by name. This occurs if the `WHERE`, `ORDER BY`, or `GROUP BY` clause refers to an expression that matches the definition of some indexed generated column. The following query does not refer directly to `gc` but does use an expression that matches the definition of `gc`:

```
SELECT * FROM t1 WHERE f1 + 1 > 9;
```

The optimizer recognizes that the expression `f1 + 1` matches the definition of `gc` and that `gc` is indexed, so it considers that index during execution plan construction. You can see this using `EXPLAIN`:

```
mysql> EXPLAIN SELECT * FROM t1 WHERE f1 + 1 > 9\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: NULL
         type: range
possible_keys: gc
          key: gc
      key_len: 5
          ref: NULL
```

```
        rows: 1
    filtered: 100.00
       Extra: Using index condition
```

In effect, the optimizer has replaced the expression `f1 + 1` with the name of the generated column that matches the expression. That is also apparent in the rewritten query available in the extended `EXPLAIN` information displayed by `SHOW WARNINGS`:

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select `test`.`t1`.`f1` AS `f1`,`test`.`t1`.`gc`
         AS `gc` from `test`.`t1` where (`test`.`t1`.`gc` > 9)
```

The following restrictions and conditions apply to the optimizer's use of generated column indexes:

- For a query expression to match a generated column definition, the expression must be identical and it must have the same result type. For example, if the generated column expression is `f1 + 1`, the optimizer will not recognize a match if the query uses `1 + f1`, or if `f1 + 1` (an integer expression) is compared with a string.

- The optimization applies to these operators: `=`, `<`, `<=`, `>`, `>=`, `BETWEEN`, and `IN()`.

  For operators other than `BETWEEN` and `IN()`, either operand can be replaced by a matching generated column. For `BETWEEN` and `IN()`, only the first argument can be replaced by a matching generated column, and the other arguments must have the same result type. `BETWEEN` and `IN()` are not yet supported for comparisons involving JSON values.

- The generated column must be defined as an expression that contains at least a function call or one of the operators mentioned in the preceding item. The expression cannot consist of a simple reference to another column. For example, `gc INT AS (f1) STORED` consists only of a column reference, so indexes on `gc` are not considered.

- For comparisons of strings to indexed generated columns that compute a value from a JSON function that returns a quoted string, `JSON_UNQUOTE()` is needed in the column definition to remove the extra quotes from the function value. (For direct comparison of a string to the function result, the JSON comparator handles quote removal, but this does not occur for index lookups.) For example, instead of writing a column definition like this:

  ```
  doc_name TEXT AS (JSON_EXTRACT(jdoc, '$.name')) STORED
  ```

  Write it like this:

  ```
  doc_name TEXT AS (JSON_UNQUOTE(JSON_EXTRACT(jdoc, '$.name'))) STORED
  ```

  With the latter definition, the optimizer can detect a match for both of these comparisons:

  ```
  ... WHERE JSON_EXTRACT(jdoc, '$.name') = 'some_string' ...
  ... WHERE JSON_UNQUOTE(JSON_EXTRACT(jdoc, '$.name')) = 'some_string' ...
  ```

  Without `JSON_UNQUOTE()` in the column definition, the optimizer detects a match only for the first of those comparisons.

- If the optimizer picks the wrong index, an index hint can be used to disable it and force the optimizer to make a different choice.

8.3.12 Invisible Indexes

MySQL supports invisible indexes; that is, indexes that are not used by the optimizer. The feature applies to indexes other than primary keys (either explicit or implicit).

Indexes are visible by default. To control index visibility explicitly for a new index, use a `VISIBLE` or `INVISIBLE` keyword as part of the index definition for `CREATE TABLE`, `CREATE INDEX`, or `ALTER TABLE`:

```
CREATE TABLE t1 (
  i INT,
  j INT,
  k INT,
  INDEX i_idx (i) INVISIBLE
) ENGINE = InnoDB;
CREATE INDEX j_idx ON t1 (j) INVISIBLE;
ALTER TABLE t1 ADD INDEX k_idx (k) INVISIBLE;
```

To alter the visibility of an existing index, use a `VISIBLE` or `INVISIBLE` keyword with the `ALTER TABLE ... ALTER INDEX` operation:

```
ALTER TABLE t1 ALTER INDEX i_idx INVISIBLE;
ALTER TABLE t1 ALTER INDEX i_idx VISIBLE;
```

Information about whether an index is visible or invisible is available from the `INFORMATION_SCHEMA.STATISTICS` table or `SHOW INDEX` output. For example:

```
mysql> SELECT INDEX_NAME, IS_VISIBLE
       FROM INFORMATION_SCHEMA.STATISTICS
       WHERE TABLE_SCHEMA = 'db1' AND TABLE_NAME = 't1';
+------------+------------+
| INDEX_NAME | IS_VISIBLE |
+------------+------------+
| i_idx      | YES        |
| j_idx      | NO         |
| k_idx      | NO         |
+------------+------------+
```

Invisible indexes make it possible to test the effect of removing an index on query performance, without making a destructive change that must be undone should the index turn out to be required. Dropping and re-adding an index can be expensive for a large table, whereas making it invisible and visible are fast, in-place operations.

If an index made invisible actually is needed or used by the optimizer, there are several ways to notice the effect of its absence on queries for the table:

- Errors occur for queries that include index hints that refer to the invisible index.
- Performance Schema data shows an increase in workload for affected queries.
- Queries have different `EXPLAIN` execution plans.
- Queries appear in the slow query log that did not appear there previously.

The `use_invisible_indexes` flag of the `optimizer_switch` system variable controls whether the optimizer uses invisible indexes for query execution plan construction. If the flag is `off` (the default), the optimizer ignores invisible indexes (the same behavior as prior to the introduction of this flag). If the

flag is on, invisible indexes remain invisible but the optimizer takes them into account for execution plan construction.

Index visibility does not affect index maintenance. For example, an index continues to be updated per changes to table rows, and a unique index prevents insertion of duplicates into a column, regardless of whether the index is visible or invisible.

A table with no explicit primary key may still have an effective implicit primary key if it has any UNIQUE indexes on NOT NULL columns. In this case, the first such index places the same constraint on table rows as an explicit primary key and that index cannot be made invisible. Consider the following table definition:

```
CREATE TABLE t2 (
  i INT NOT NULL,
  j INT NOT NULL,
  UNIQUE j_idx (j)
) ENGINE = InnoDB;
```

The definition includes no explicit primary key, but the index on NOT NULL column j places the same constraint on rows as a primary key and cannot be made invisible:

```
mysql> ALTER TABLE t2 ALTER INDEX j_idx INVISIBLE;
ERROR 3522 (HY000): A primary key index cannot be invisible.
```

Now suppose that an explicit primary key is added to the table:

```
ALTER TABLE t2 ADD PRIMARY KEY (i);
```

The explicit primary key cannot be made invisible. In addition, the unique index on j no longer acts as an implicit primary key and as a result can be made invisible:

```
mysql> ALTER TABLE t2 ALTER INDEX j_idx INVISIBLE;
Query OK, 0 rows affected (0.03 sec)
```

8.3.13 Descending Indexes

MySQL supports descending indexes: DESC in an index definition is no longer ignored but causes storage of key values in descending order. Previously, indexes could be scanned in reverse order but at a performance penalty. A descending index can be scanned in forward order, which is more efficient. Descending indexes also make it possible for the optimizer to use multiple-column indexes when the most efficient scan order mixes ascending order for some columns and descending order for others.

Consider the following table definition, which contains two columns and four two-column index definitions for the various combinations of ascending and descending indexes on the columns:

```
CREATE TABLE t (
  c1 INT, c2 INT,
  INDEX idx1 (c1 ASC, c2 ASC),
  INDEX idx2 (c1 ASC, c2 DESC),
  INDEX idx3 (c1 DESC, c2 ASC),
  INDEX idx4 (c1 DESC, c2 DESC)
);
```

The table definition results in four distinct indexes. The optimizer can perform a forward index scan for each of the ORDER BY clauses and need not use a filesort operation:

```
ORDER BY c1 ASC, c2 ASC    -- optimizer can use idx1
```

```
ORDER BY c1 DESC, c2 DESC  -- optimizer can use idx4
ORDER BY c1 ASC, c2 DESC   -- optimizer can use idx2
ORDER BY c1 DESC, c2 ASC   -- optimizer can use idx3
```

Use of descending indexes is subject to these conditions:

- Descending indexes are supported only for the `InnoDB` storage engine, with these limitations:
  - Change buffering is not supported for a secondary index if the index contains a descending index key column or if the primary key includes a descending index column.
  - The `InnoDB` SQL parser does not use descending indexes. For `InnoDB` full-text search, this means that the index required on the `FTS_DOC_ID` column of the indexed table cannot be defined as a descending index. For more information, see Section 15.6.2.4, “InnoDB FULLTEXT Indexes”.
- Descending indexes are supported for all data types for which ascending indexes are available.
- Descending indexes are supported for ordinary (nongenerated) and generated columns (both `VIRTUAL` and `STORED`).
- `DISTINCT` can use any index containing matching columns, including descending key parts.
- Indexes that have descending key parts are not used for `MIN()`/`MAX()` optimization of queries that invoke aggregate functions but do not have a `GROUP BY` clause.
- Descending indexes are supported for `BTREE` but not `HASH` indexes. Descending indexes are not supported for `FULLTEXT` or `SPATIAL` indexes.

  Explicitly specified `ASC` and `DESC` designators for `HASH`, `FULLTEXT`, and `SPATIAL` indexes results in an error.

8.3.14 Indexed Lookups from TIMESTAMP Columns

Temporal values are stored in `TIMESTAMP` columns as UTC values, and values inserted into and retrieved from `TIMESTAMP` columns are converted between the session time zone and UTC. (This is the same type of conversion performed by the `CONVERT_TZ()` function. If the session time zone is UTC, there is effectively no time zone conversion.)

Due to conventions for local time zone changes such as Daylight Saving Time (DST), conversions between UTC and non-UTC time zones are not one-to-one in both directions. UTC values that are distinct may not be distinct in another time zone. The following example shows distinct UTC values that become identical in a non-UTC time zone:

```
mysql> CREATE TABLE tstable (ts TIMESTAMP);
mysql> SET time_zone = 'UTC'; -- insert UTC values
mysql> INSERT INTO tstable VALUES
       ('2018-10-28 00:30:00'),
       ('2018-10-28 01:30:00');
mysql> SELECT ts FROM tstable;
+---------------------+
| ts                  |
+---------------------+
| 2018-10-28 00:30:00 |
| 2018-10-28 01:30:00 |
+---------------------+
mysql> SET time_zone = 'MET'; -- retrieve non-UTC values
mysql> SELECT ts FROM tstable;
+---------------------+
| ts                  |
```

```
+---------------------+
| 2018-10-28 02:30:00 |
| 2018-10-28 02:30:00 |
+---------------------+
```

Note

To use named time zones such as `'MET'` or `'Europe/Amsterdam'`, the time zone tables must be properly set up. For instructions, see Section 5.1.13, “MySQL Server Time Zone Support”.

You can see that the two distinct UTC values are the same when converted to the `'MET'` time zone. This phenomenon can lead to different results for a given `TIMESTAMP` column query, depending on whether the optimizer uses an index to execute the query.

Suppose that a query selects values from the table shown earlier using a `WHERE` clause to search the `ts` column for a single specific value such as a user-provided timestamp literal:

```
SELECT ts FROM tstable
WHERE ts = 'literal';
```

Suppose further that the query executes under these conditions:

- The session time zone is not UTC and has a DST shift. For example:

  ```
  SET time_zone = 'MET';
  ```

- Unique UTC values stored in the `TIMESTAMP` column are not unique in the session time zone due to DST shifts. (The example shown earlier illustrates how this can occur.)

- The query specifies a search value that is within the hour of entry into DST in the session time zone.

Under those conditions, the comparison in the `WHERE` clause occurs in different ways for nonindexed and indexed lookups and leads to different results:

- If there is no index or the optimizer cannot use it, comparisons occur in the session time zone. The optimizer performs a table scan in which it retrieves each `ts` column value, converts it from UTC to the session time zone, and compares it to the search value (also interpreted in the session time zone):

  ```
  mysql> SELECT ts FROM tstable
         WHERE ts = '2018-10-28 02:30:00';
  +---------------------+
  | ts                  |
  +---------------------+
  | 2018-10-28 02:30:00 |
  | 2018-10-28 02:30:00 |
  +---------------------+
  ```

  Because the stored `ts` values are converted to the session time zone, it is possible for the query to return two timestamp values that are distinct as UTC values but equal in the session time zone: One value that occurs before the DST shift when clocks are changed, and one value that was occurs after the DST shift.

- If there is a usable index, comparisons occur in UTC. The optimizer performs an index scan, first converting the search value from the session time zone to UTC, then comparing the result to the UTC index entries:

  ```
  ```

```
mysql> ALTER TABLE tstable ADD INDEX (ts);
mysql> SELECT ts FROM tstable
       WHERE ts = '2018-10-28 02:30:00';
+---------------------+
| ts                  |
+---------------------+
| 2018-10-28 02:30:00 |
+---------------------+
```

In this case, the (converted) search value is matched only to index entries, and because the index entries for the distinct stored UTC values are also distinct, the search value can match only one of them.

Due to different optimizer operation for nonindexed and indexed lookups, the query produces different results in each case. The result from the nonindexed lookup returns all values that match in the session time zone. The indexed lookup cannot do so:

- It is performed within the storage engine, which knows only about UTC values.
- For the two distinct session time zone values that map to the same UTC value, the indexed lookup matches only the corresponding UTC index entry and returns only a single row.

In the preceding discussion, the data set stored in `tstable` happens to consist of distinct UTC values. In such cases, all index-using queries of the form shown match at most one index entry.

If the index is not `UNIQUE`, it is possible for the table (and the index) to store multiple instances of a given UTC value. For example, the `ts` column might contain multiple instances of the UTC value `'2018-10-28 00:30:00'`. In this case, the index-using query would return each of them (converted to the MET value `'2018-10-28 02:30:00'` in the result set). It remains true that index-using queries match the converted search value to a single value in the UTC index entries, rather than matching multiple UTC values that convert to the search value in the session time zone.

If it is important to return all `ts` values that match in the session time zone, the workaround is to suppress use of the index with an `IGNORE INDEX` hint:

```
mysql> SELECT ts FROM tstable
       IGNORE INDEX (ts)
       WHERE ts = '2018-10-28 02:30:00';
+---------------------+
| ts                  |
+---------------------+
| 2018-10-28 02:30:00 |
| 2018-10-28 02:30:00 |
+---------------------+
```

The same lack of one-to-one mapping for time zone conversions in both directions occurs in other contexts as well, such as conversions performed with the `FROM_UNIXTIME()` and `UNIX_TIMESTAMP()` functions. See Section 12.7, “Date and Time Functions”.

8.4 Optimizing Database Structure

In your role as a database designer, look for the most efficient way to organize your schemas, tables, and columns. As when tuning application code, you minimize I/O, keep related items together, and plan ahead so that performance stays high as the data volume increases. Starting with an efficient database design makes it easier for team members to write high-performing application code, and makes the database likely to endure as applications evolve and are rewritten.

8.4.1 Optimizing Data Size

Design your tables to minimize their space on the disk. This can result in huge improvements by reducing the amount of data written to and read from disk. Smaller tables normally require less main memory while their contents are being actively processed during query execution. Any space reduction for table data also results in smaller indexes that can be processed faster.

MySQL supports many different storage engines (table types) and row formats. For each table, you can decide which storage and indexing method to use. Choosing the proper table format for your application can give you a big performance gain. See Chapter 15, *The InnoDB Storage Engine*, and Chapter 16, *Alternative Storage Engines*.

You can get better performance for a table and minimize storage space by using the techniques listed here:

- Table Columns
- Row Format
- Indexes
- Joins
- Normalization

Table Columns

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory. For example, use the smaller integer types if possible to get smaller tables. `MEDIUMINT` is often a better choice than `INT` because a `MEDIUMINT` column uses 25% less space.
- Declare columns to be `NOT NULL` if possible. It makes SQL operations faster, by enabling better use of indexes and eliminating overhead for testing whether each value is `NULL`. You also save some storage space, one bit per column. If you really need `NULL` values in your tables, use them. Just avoid the default setting that allows `NULL` values in every column.

Row Format

- `InnoDB` tables are created using the `DYNAMIC` row format by default. To use a row format other than `DYNAMIC`, configure `innodb_default_row_format`, or specify the `ROW_FORMAT` option explicitly in a `CREATE TABLE` or `ALTER TABLE` statement.

  The compact family of row formats, which includes `COMPACT`, `DYNAMIC`, and `COMPRESSED`, decreases row storage space at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed it is likely to be faster. If it is a rare case that is limited by CPU speed, it might be slower.

  The compact family of row formats also optimizes `CHAR` column storage when using a variable-length character set such as `utf8mb3` or `utf8mb4`. With `ROW_FORMAT=REDUNDANT`, `CHAR(N)` occupies *N* × the maximum byte length of the character set. Many languages can be written primarily using single-byte `utf8` characters, so a fixed storage length often wastes space. With the compact family of rows formats, `InnoDB` allocates a variable amount of storage in the range of *N* to *N* × the maximum byte length of the character set for these columns by stripping trailing spaces. The minimum storage length is *N* bytes to facilitate in-place updates in typical cases. For more information, see Section 15.10, “InnoDB Row Formats”.

- To minimize space even further by storing table data in compressed form, specify `ROW_FORMAT=COMPRESSED` when creating `InnoDB` tables, or run the `myisampack` command on

an existing `MyISAM` table. (`InnoDB` compressed tables are readable and writable, while `MyISAM` compressed tables are read-only.)

- For `MyISAM` tables, if you do not have any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB` columns), a fixed-size row format is used. This is faster but may waste some space. See Section 16.2.3, “MyISAM Table Storage Formats”. You can hint that you want to have fixed length rows even if you have `VARCHAR` columns with the `CREATE TABLE` option `ROW_FORMAT=FIXED`.

Indexes

- The primary index of a table should be as short as possible. This makes identification of each row easy and efficient. For `InnoDB` tables, the primary key columns are duplicated in each secondary index entry, so a short primary key saves considerable space if you have many secondary indexes.
- Create only the indexes that you need to improve query performance. Indexes are good for retrieval, but slow down insert and update operations. If you access a table mostly by searching on a combination of columns, create a single composite index on them rather than a separate index for each column. The first part of the index should be the column most used. If you *always* use many columns when selecting from the table, the first column in the index should be the one with the most duplicates, to obtain better compression of the index.
- If it is very likely that a long string column has a unique prefix on the first number of characters, it is better to index only this prefix, using MySQL's support for creating an index on the leftmost part of the column (see Section 13.1.15, “CREATE INDEX Syntax”). Shorter indexes are faster, not only because they require less disk space, but because they also give you more hits in the index cache, and thus fewer disk seeks. See Section 5.1.1, “Configuring the Server”.

Joins

- In some circumstances, it can be beneficial to split into two a table that is scanned very often. This is especially true if it is a dynamic-format table and it is possible to use a smaller static format table that can be used to find the relevant rows when scanning the table.
- Declare columns with identical information in different tables with identical data types, to speed up joins based on the corresponding columns.
- Keep column names simple, so that you can use the same name across different tables and simplify join queries. For example, in a table named `customer`, use a column name of `name` instead of `customer_name`. To make your names portable to other SQL servers, consider keeping them shorter than 18 characters.

Normalization

- Normally, try to keep all data nonredundant (observing what is referred to in database theory as *third normal form*). Instead of repeating lengthy values such as names and addresses, assign them unique IDs, repeat these IDs as needed across multiple smaller tables, and join the tables in queries by referencing the IDs in the join clause.
- If speed is more important than disk space and the maintenance costs of keeping multiple copies of data, for example in a business intelligence scenario where you analyze all the data from large tables, you can relax the normalization rules, duplicating information or creating summary tables to gain more speed.

8.4.2 Optimizing MySQL Data Types

8.4.2.1 Optimizing for Numeric Data

- For unique IDs or other values that can be represented as either strings or numbers, prefer numeric columns to string columns. Since large numeric values can be stored in fewer bytes than the corresponding strings, it is faster and takes less memory to transfer and compare them.
- If you are using numeric data, it is faster in many cases to access information from a database (using a live connection) than to access a text file. Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses. You also save code in your application because you can avoid parsing the text file to find line and column boundaries.

8.4.2.2 Optimizing for Character and String Types

For character and string columns, follow these guidelines:

- Use binary collation order for fast comparison and sort operations, when you do not need language-specific collation features. You can use the `BINARY` operator to use binary collation within a particular query.
- When comparing values from different columns, declare those columns with the same character set and collation wherever possible, to avoid string conversions while running the query.
- For column values less than 8KB in size, use binary `VARCHAR` instead of `BLOB`. The `GROUP BY` and `ORDER BY` clauses can generate temporary tables, and these temporary tables can use the `MEMORY` storage engine if the original table does not contain any `BLOB` columns.
- If a table contains string columns such as name and address, but many queries do not retrieve those columns, consider splitting the string columns into a separate table and using join queries with a foreign key when necessary. When MySQL retrieves any value from a row, it reads a data block containing all the columns of that row (and possibly other adjacent rows). Keeping each row small, with only the most frequently used columns, allows more rows to fit in each data block. Such compact tables reduce disk I/O and memory usage for common queries.
- When you use a randomly generated value as a primary key in an `InnoDB` table, prefix it with an ascending value such as the current date and time if possible. When consecutive primary values are physically stored near each other, `InnoDB` can insert and retrieve them faster.
- See Section 8.4.2.1, “Optimizing for Numeric Data” for reasons why a numeric column is usually preferable to an equivalent string column.

8.4.2.3 Optimizing for BLOB Types

- When storing a large blob containing textual data, consider compressing it first. Do not use this technique when the entire table is compressed by `InnoDB` or `MyISAM`.
- For a table with several columns, to reduce memory requirements for queries that do not use the BLOB column, consider splitting the BLOB column into a separate table and referencing it with a join query when needed.
- Since the performance requirements to retrieve and display a BLOB value might be very different from other data types, you could put the BLOB-specific table on a different storage device or even a separate database instance. For example, to retrieve a BLOB might require a large sequential disk read that is better suited to a traditional hard drive than to an SSD device.
- See Section 8.4.2.2, “Optimizing for Character and String Types” for reasons why a binary `VARCHAR` column is sometimes preferable to an equivalent BLOB column.
- Rather than testing for equality against a very long text string, you can store a hash of the column value in a separate column, index that column, and test the hashed value in queries. (Use the `MD5()` or

`CRC32()` function to produce the hash value.) Since hash functions can produce duplicate results for different inputs, you still include a clause `AND blob_column = long_string_value` in the query to guard against false matches; the performance benefit comes from the smaller, easily scanned index for the hashed values.

8.4.3 Optimizing for Many Tables

Some techniques for keeping individual queries fast involve splitting data across many tables. When the number of tables runs into the thousands or even millions, the overhead of dealing with all these tables becomes a new performance consideration.

8.4.3.1 How MySQL Opens and Closes Tables

When you execute a `mysqladmin status` command, you should see something like this:

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

The `Open tables` value of 12 can be somewhat puzzling if you have fewer than 12 tables.

MySQL is multithreaded, so there may be many clients issuing queries for a given table simultaneously. To minimize the problem with multiple client sessions having different states on the same table, the table is opened independently by each concurrent session. This uses additional memory but normally increases performance. With `MyISAM` tables, one extra file descriptor is required for the data file for each client that has the table open. (By contrast, the index file descriptor is shared between all sessions.)

The `table_open_cache` and `max_connections` system variables affect the maximum number of files the server keeps open. If you increase one or both of these values, you may run up against a limit imposed by your operating system on the per-process number of open file descriptors. Many operating systems permit you to increase the open-files limit, although the method varies widely from system to system. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so.

`table_open_cache` is related to `max_connections`. For example, for 200 concurrent running connections, specify a table cache size of at least `200 * N`, where `N` is the maximum number of tables per join in any of the queries which you execute. You must also reserve some extra file descriptors for temporary tables and files.

Make sure that your operating system can handle the number of open file descriptors implied by the `table_open_cache` setting. If `table_open_cache` is set too high, MySQL may run out of file descriptors and exhibit symptoms such as refusing connections or failing to perform queries.

Also take into account that the `MyISAM` storage engine needs two file descriptors for each unique open table. To increase the number of file descriptors available to MySQL, set the `open_files_limit` system variable. See Section B.4.2.17, “File Not Found and Similar Errors”.

The cache of open tables is kept at a level of `table_open_cache` entries. The server autosizes the cache size at startup. To set the size explicitly, set the `table_open_cache` system variable at startup. MySQL may temporarily open more tables than this to execute queries, as described later in this section.

MySQL closes an unused table and removes it from the table cache under the following circumstances:

- When the cache is full and a thread tries to open a table that is not in the cache.
- When the cache contains more than `table_open_cache` entries and a table in the cache is no longer being used by any threads.

- When a table-flushing operation occurs. This happens when someone issues a `FLUSH TABLES` statement or executes a `mysqladmin flush-tables` or `mysqladmin refresh` command.

When the table cache fills up, the server uses the following procedure to locate a cache entry to use:

- Tables not currently in use are released, beginning with the table least recently used.
- If a new table must be opened, but the cache is full and no tables can be released, the cache is temporarily extended as necessary. When the cache is in a temporarily extended state and a table goes from a used to unused state, the table is closed and released from the cache.

A `MyISAM` table is opened for each concurrent access. This means the table needs to be opened twice if two threads access the same table or if a thread accesses the table twice in the same query (for example, by joining the table to itself). Each concurrent open requires an entry in the table cache. The first open of any `MyISAM` table takes two file descriptors: one for the data file and one for the index file. Each additional use of the table takes only one file descriptor for the data file. The index file descriptor is shared among all threads.

If you are opening a table with the `HANDLER *tbl_name* OPEN` statement, a dedicated table object is allocated for the thread. This table object is not shared by other threads and is not closed until the thread calls `HANDLER *tbl_name* CLOSE` or the thread terminates. When this happens, the table is put back in the table cache (if the cache is not full). See Section 13.2.4, “HANDLER Syntax”.

To determine whether your table cache is too small, check the `Opened_tables` status variable, which indicates the number of table-opening operations since the server started:

```
mysql> SHOW GLOBAL STATUS LIKE 'Opened_tables';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Opened_tables | 2741  |
+---------------+-------+
```

If the value is very large or increases rapidly, even when you have not issued many `FLUSH TABLES` statements, increase the `table_open_cache` value at server startup.

8.4.3.2 Disadvantages of Creating Many Tables in the Same Database

If you have many `MyISAM` tables in the same database directory, open, close, and create operations are slow. If you execute `SELECT` statements on many different tables, there is a little overhead when the table cache is full, because for every table that has to be opened, another must be closed. You can reduce this overhead by increasing the number of entries permitted in the table cache.

8.4.4 Internal Temporary Table Use in MySQL

In some cases, the server creates internal temporary tables while processing statements. Users have no direct control over when this occurs.

The server creates temporary tables under conditions such as these:

- Evaluation of `UNION` statements, with some exceptions described later.
- Evaluation of some views, such those that use the `TEMPTABLE` algorithm, `UNION`, or aggregation.
- Evaluation of derived tables (see Section 13.2.11.8, “Derived Tables”).

- Evaluation of common table expressions (see Section 13.2.13, “WITH Syntax (Common Table Expressions)”).
- Tables created for subquery or semijoin materialization (see Section 8.2.2, “Optimizing Subqueries, Derived Tables, View References, and Common Table Expressions”).
- Evaluation of statements that contain an `ORDER BY` clause and a different `GROUP BY` clause, or for which the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue.
- Evaluation of `DISTINCT` combined with `ORDER BY` may require a temporary table.
- For queries that use the `SQL_SMALL_RESULT` modifier, MySQL uses an in-memory temporary table, unless the query also contains elements (described later) that require on-disk storage.
- To evaluate `INSERT ... SELECT` statements that select from and insert into the same table, MySQL creates an internal temporary table to hold the rows from the `SELECT`, then inserts those rows into the target table. See Section 13.2.6.1, “INSERT ... SELECT Syntax”.
- Evaluation of multiple-table `UPDATE` statements.
- Evaluation of `GROUP_CONCAT()` or `COUNT(DISTINCT)` expressions.
- Evaluation of window functions (see Section 12.21, “Window Functions”) uses temporary tables as necessary.

To determine whether a statement requires a temporary table, use `EXPLAIN` and check the `Extra` column to see whether it says `Using temporary` (see Section 8.8.1, “Optimizing Queries with EXPLAIN”). `EXPLAIN` will not necessarily say `Using temporary` for derived or materialized temporary tables. For statements that use window functions, `EXPLAIN` with `FORMAT=JSON` always provides information about the windowing steps. If the windowing functions use temporary tables, it is indicated for each step.

When the server creates an internal temporary table (either in memory or on disk), it increments the `Created_tmp_tables` status variable. If the server creates the table on disk (either initially or by converting an in-memory table) it increments the `Created_tmp_disk_tables` status variable.

Some query conditions prevent the use of an in-memory temporary table, in which case the server uses an on-disk table instead:

- Presence of a `BLOB` or `TEXT` column in the table. However, the `TempTable` storage engine, which is the default storage engine for in-memory internal temporary tables in MySQL 8.0, supports binary large object types as of MySQL 8.0.13. See Internal Temporary Table Storage Engine.
- Presence of any string column with a maximum length larger than 512 (bytes for binary strings, characters for nonbinary strings) in the `SELECT` list, if `UNION` or `UNION ALL` is used.
- The `SHOW COLUMNS` and `DESCRIBE` statements use `BLOB` as the type for some columns, thus the temporary table used for the results is an on-disk table.

The server does not use a temporary table for `UNION` statements that meet certain qualifications. Instead, it retains from temporary table creation only the data structures necessary to perform result column typecasting. The table is not fully instantiated and no rows are written to or read from it; rows are sent directly to the client. The result is reduced memory and disk requirements, and smaller delay before the first row is sent to the client because the server need not wait until the last query block is executed. `EXPLAIN` and optimizer trace output reflects this execution strategy: The `UNION RESULT` query block is not present because that block corresponds to the part that reads from the temporary table.

These conditions qualify a `UNION` for evaluation without a temporary table:

- The union is `UNION ALL`, not `UNION` or `UNION DISTINCT`.
- There is no global `ORDER BY` clause.
- The union is not the top-level query block of an `{INSERT | REPLACE} ... SELECT ...` statement.

Internal Temporary Table Storage Engine

An internal temporary table can be held in memory and processed by the `TempTable` or `MEMORY` storage engine, or stored on disk by the `InnoDB` storage engine.

Storage Engine for In-Memory Internal Temporary Tables

The `internal_tmp_mem_storage_engine` session variable defines the storage engine for in-memory internal temporary tables. Permitted values are `TempTable` (the default) and `MEMORY`.

The `TempTable` storage engine provides efficient storage for `VARCHAR` and `VARBINARY` columns. Storage of other binary large object types is supported as of MySQL 8.0.13. The `temptable_max_ram` variable defines the maximum amount of RAM that can be occupied by the `TempTable` storage engine before it starts allocating space from disk in the form memory-mapped temporary files (the default) or `InnoDB` on-disk internal temporary tables. The default `temptable_max_ram` setting is 1GiB. The `temptable_use_mmap` variable (introduced in MySQL 8.0.16) controls whether the TempTable storage engine uses memory-mapped files or `InnoDB` on-disk internal temporary tables when the `temptable_max_ram` limit is exceeded. The default setting is `temptable_use_mmap=ON`.

Note

The `temptable_max_ram` setting does not account for the thread-local memory block allocated to each thread that uses the `TempTable` storage engine. The size of the thread-local memory block depends on the size of the thread's first memory allocation request. If the request is less than 1MB, which it is in most cases, the thread-local memory block size is 1MB. If the request is greater than 1MB, the thread-local memory block is approximately the same size as the initial memory request. The thread-local memory block is held in thread-local storage until thread exit.

Use of memory-mapped temporary files by the `TempTable` storage engine as an overflow mechanism for in-memory temporary tables is governed by these rules:

- Temporary files are created in the directory defined by the `tmpdir` variable.
- Temporary files are deleted immediately after they are created and opened, and therefore do not remain visible in the `tmpdir` directory. The space occupied by temporary files is held by the operating system while temporary files are open. The space is reclaimed when temporary files are closed by the `TempTable` storage engine, or when the `mysqld` process is shut down.
- Data is never moved between RAM and temporary files, within RAM, or between temporary files.
- New data is stored in RAM if space becomes available within the limit defined by `temptable_max_ram`. Otherwise, new data is stored in temporary files.
- If space becomes available in RAM after some of the data for a table is written to temporary files, it is possible for the remaining table data to be stored in RAM.

If the `TempTable` storage engine is configured to use `InnoDB` on-disk internal temporary tables as the overflow mechanism (`temptable_use_mmap=OFF`), an in-memory table that exceeds the `temptable_max_ram` limit is converted to an `InnoDB` on-disk internal temporary table, and any rows

belonging to that table are moved from memory to the InnoDB on-disk internal temporary table. The internal_tmp_disk_storage_engine (removed in MySQL 8.0.16) variable setting has no affect on the TempTable storage engine overflow mechanism.

Consider using InnoDB on-disk internal temporary tables as the TempTable overflow mechanism if the TempTable storage engine often exceeds the memory limit defined by the temptable_max_ram variable and uses excessive space in the temporary directory for memory-mapped files. This may occur due to use of large internal temporary tables or extensive use of internal temporary tables. InnoDB on-disk internal temporary tables are created in session temporary tablespaces, which reside in the data directory by default. For more information, see Section 15.6.3.5, "Temporary Tablespaces".

The memory/temptable/physical_ram and memory/temptable/physical_disk Performance Schema instruments can be used to monitor TempTable space allocation from memory and disk. memory/temptable/physical_ram reports the amount of allocated RAM. memory/temptable/physical_disk reports the amount of space allocated from disk when memory-mapped files are used as the TempTable overflow mechanism (temptable_use_mmap=ON). If the physical_disk instrument reports a value other than 0 and memory-mapped files are used as the TempTable overflow mechanism, the temptable_max_ram threshold was reached at some point. Data can be queried in Performance Schema memory summary tables such as memory_summary_global_by_event_name. See Section 26.12.17.10, "Memory Summary Tables".

When using the MEMORY storage engine for in-memory temporary tables, MySQL automatically converts an in-memory temporary table to an on-disk table if it becomes too large. The maximum size for in-memory temporary tables is defined by the tmp_table_size or max_heap_table_size value, whichever is smaller. This differs from MEMORY tables explicitly created with CREATE TABLE. For such tables, only the max_heap_table_size variable determines how large a table can grow, and there is no conversion to on-disk format.

Storage Engine for On-Disk Internal Temporary Tables

Starting with MySQL 8.0.16, the server always uses the InnoDB storage engine for managing internal temporary tables on disk.

In MySQL 8.0.15 and earlier, the internal_tmp_disk_storage_engine variable was used to define the storage engine used for on-disk internal temporary tables. This variable was removed in MySQL 8.0.16, and the storage engine used for this purpose is no longer user-configurable.

In MySQL 8.0.15 and earlier: For common table expressions (CTEs), the storage engine used for on-disk internal temporary tables cannot be MyISAM. If internal_tmp_disk_storage_engine=MYISAM, an error occurs for any attempt to materialize a CTE using an on-disk temporary table.

In MySQL 8.0.15 and earlier: When using internal_tmp_disk_storage_engine=INNODB, queries that generate on-disk internal temporary tables that exceed InnoDB row or column limits return Row size too large or Too many columns errors. The workaround is to set internal_tmp_disk_storage_engine to MYISAM.

Internal Temporary Table Storage Format

When in-memory internal temporary tables are managed by the TempTable storage engine, rows that include VARCHAR columns, VARBINARY columns, or other binary large object type columns (supported as of MySQL 8.0.13) are represented in memory by an array of cells, with each cell containing a NULL flag, the data length, and a data pointer. Column values are placed in consecutive order after the array, in a single region of memory, without padding. Each cell in the array uses 16 bytes of storage. The same storage format applies when the TempTable storage engine exceeds the temptable_max_ram limit and starts allocating space from disk as memory-mapped files or InnoDB on-disk internal temporary tables.

When in-memory internal temporary tables are managed by the MEMORY storage engine, fixed-length row format is used. VARCHAR and VARBINARY column values are padded to the maximum column length, in effect storing them as CHAR and BINARY columns.

Previous to MySQL 8.0.16, on-disk internal temporary tables were managed by the InnoDB or MyISAM storage engine (depending on the internal_tmp_disk_storage_engine setting). Both engines store internal temporary tables using dynamic-width row format. Columns take only as much storage as needed, which reduces disk I/O, space requirements, and processing time compared to on-disk tables that use fixed-length rows. Beginning with MySQL 8.0.16, internal_tmp_disk_storage_engine is not supported, and internal temporary tables on disk are always handled by InnoDB.

When using the MEMORY storage engine, statements can initially create an in-memory internal temporary table and then convert it to an on-disk table if the table becomes too large. In such cases, better performance might be achieved by skipping the conversion and creating the internal temporary table on disk to begin with. The big_tables variable can be used to force disk storage of internal temporary tables.

8.4.5 Limits on Number of Databases and Tables

MySQL has no limit on the number of databases. The underlying file system may have a limit on the number of directories.

MySQL has no limit on the number of tables. The underlying file system may have a limit on the number of files that represent tables. Individual storage engines may impose engine-specific constraints. InnoDB permits up to 4 billion tables.

8.4.6 Limits on Table Size

The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. For up-to-date information operating system file size limits, refer to the documentation specific to your operating system.

Windows users, please note that FAT and VFAT (FAT32) are *not* considered suitable for production use with MySQL. Use NTFS instead.

If you encounter a full-table error, there are several reasons why it might have occurred:

- The disk might be full.
- You are using InnoDB tables and have run out of room in an InnoDB tablespace file. The maximum tablespace size is also the maximum size for a table. For tablespace size limits, see Section 15.6.1.6, “Limits on InnoDB Tables”.

  Generally, partitioning of tables into multiple tablespace files is recommended for tables larger than 1TB in size.
- You have hit an operating system file size limit. For example, you are using MyISAM tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.
- You are using a MyISAM table and the space required for the table exceeds what is permitted by the internal pointer size. MyISAM permits data and index files to grow up to 256TB by default, but this limit can be changed up to the maximum permissible size of 65,536TB ($256^7 - 1$ bytes).

  If you need a MyISAM table that is larger than the default limit and your operating system supports large files, the CREATE TABLE statement supports AVG_ROW_LENGTH and MAX_ROWS options. See

Section 13.1.20, “CREATE TABLE Syntax”. The server uses these options to determine how large a table to permit.

If the pointer size is too small for an existing table, you can change the options with `ALTER TABLE` to increase a table's maximum permissible size. See Section 13.1.9, “ALTER TABLE Syntax”.

```
ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
```

You have to specify `AVG_ROW_LENGTH` only for tables with `BLOB` or `TEXT` columns; in this case, MySQL can't optimize the space required based only on the number of rows.

To change the default size limit for `MyISAM` tables, set the `myisam_data_pointer_size`, which sets the number of bytes used for internal row pointers. The value is used to set the pointer size for new tables if you do not specify the `MAX_ROWS` option. The value of `myisam_data_pointer_size` can be from 2 to 7. A value of 4 permits tables up to 4GB; a value of 6 permits tables up to 256TB.

You can check the maximum data and index sizes by using this statement:

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
```

You also can use `myisamchk -dv /path/to/table-index-file`. See Section 13.7.7, “SHOW Syntax”, or Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”.

Other ways to work around file-size limits for `MyISAM` tables are as follows:

- If your large table is read only, you can use `myisampack` to compress it. `myisampack` usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. `myisampack` also can merge multiple tables into a single table. See Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”.
- MySQL includes a `MERGE` library that enables you to handle a collection of `MyISAM` tables that have identical structure as a single `MERGE` table. See Section 16.7, “The MERGE Storage Engine”.

- You are using the `MEMORY` (`HEAP`) storage engine; in this case you need to increase the value of the `max_heap_table_size` system variable. See Section 5.1.8, “Server System Variables”.

8.4.7 Limits on Table Column Count and Row Size

This section describes limits on the number of columns in tables and the size of individual rows.

- Column Count Limits
- Row Size Limits

Column Count Limits

MySQL has hard limit of 4096 columns per table, but the effective maximum may be less for a given table. The exact column limit depends on several factors:

- The maximum row size for a table constrains the number (and possibly size) of columns because the total length of all columns cannot exceed this size. See Row Size Limits.
- The storage requirements of individual columns constrain the number of columns that fit within a given maximum row size. Storage requirements for some data types depend on factors such as storage engine, storage format, and character set. See Section 11.8, “Data Type Storage Requirements”.

- Storage engines may impose additional restrictions that limit table column count. For example, `InnoDB` has a limit of 1017 columns per table. See Section 15.6.1.6, “Limits on InnoDB Tables”. For information about other storage engines, see Chapter 16, *Alternative Storage Engines*.
- Functional key parts (see Section 13.1.15, “CREATE INDEX Syntax”) are implemented as hidden virtual generated stored columns, so each functional key part in a table index counts against the table total column limit.

Row Size Limits

The maximum row size for a given table is determined by several factors:

- The internal representation of a MySQL table has a maximum row size limit of 65,535 bytes, even if the storage engine is capable of supporting larger rows. `BLOB` and `TEXT` columns only contribute 9 to 12 bytes toward the row size limit because their contents are stored separately from the rest of the row.
- The maximum row size for an `InnoDB` table, which applies to data stored locally within a database page, is slightly less than half a page for 4KB, 8KB, 16KB, and 32KB `innodb_page_size` settings. For example, the maximum row size is slightly less than 8KB for the default 16KB `InnoDB` page size. For 64KB pages, the maximum row size is slightly less than 16KB. See Section 15.6.1.6, “Limits on InnoDB Tables”.

  If a row containing variable-length columns exceeds the `InnoDB` maximum row size, `InnoDB` selects variable-length columns for external off-page storage until the row fits within the `InnoDB` row size limit. The amount of data stored locally for variable-length columns that are stored off-page differs by row format. For more information, see Section 15.10, “InnoDB Row Formats”.
- Different storage formats use different amounts of page header and trailer data, which affects the amount of storage available for rows.
  - For information about `InnoDB` row formats, see Section 15.10, “InnoDB Row Formats”.
  - For information about `MyISAM` storage formats, see Section 16.2.3, “MyISAM Table Storage Formats”.

Row Size Limit Examples

- The MySQL maximum row size limit of 65,535 bytes is demonstrated in the following `InnoDB` and `MyISAM` examples. The limit is enforced regardless of storage engine, even though the storage engine may be capable of supporting larger rows.

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
       c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
       f VARCHAR(10000), g VARCHAR(6000)) ENGINE=InnoDB CHARACTER SET latin1;
ERROR 1118 (42000): Row size too large. The maximum row size for the used
table type, not counting BLOBs, is 65535. This includes storage overhead,
check the manual. You have to change some columns to TEXT or BLOBs
```

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
       c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
       f VARCHAR(10000), g VARCHAR(6000)) ENGINE=MyISAM CHARACTER SET latin1;
ERROR 1118 (42000): Row size too large. The maximum row size for the used
table type, not counting BLOBs, is 65535. This includes storage overhead,
check the manual. You have to change some columns to TEXT or BLOBs
```

  In the following `MyISAM` example, changing a column to `TEXT` avoids the 65,535-byte row size limit and permits the operation to succeed because `BLOB` and `TEXT` columns only contribute 9 to 12 bytes toward the row size.

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
       c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
       f VARCHAR(10000), g TEXT(6000)) ENGINE=MyISAM CHARACTER SET latin1;
Query OK, 0 rows affected (0.02 sec)
```

The operation succeeds for an `InnoDB` table because changing a column to `TEXT` avoids the MySQL 65,535-byte row size limit, and `InnoDB` off-page storage of variable-length columns avoids the `InnoDB` row size limit.

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
       c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
       f VARCHAR(10000), g TEXT(6000)) ENGINE=InnoDB CHARACTER SET latin1;
Query OK, 0 rows affected (0.02 sec)
```

- Storage for variable-length columns includes length bytes, which are counted toward the row size. For example, a `VARCHAR(255) CHARACTER SET utf8mb3` column takes two bytes to store the length of the value, so each value can take up to 767 bytes.

  The statement to create table `t1` succeeds because the columns require 32,765 + 2 bytes and 32,766 + 2 bytes, which falls within the maximum row size of 65,535 bytes:

  ```
  mysql> CREATE TABLE t1
         (c1 VARCHAR(32765) NOT NULL, c2 VARCHAR(32766) NOT NULL)
         ENGINE = InnoDB CHARACTER SET latin1;
  Query OK, 0 rows affected (0.02 sec)
  ```

  The statement to create table `t2` fails because, although the column length is within the maximum length of 65,535 bytes, two additional bytes are required to record the length, which causes the row size to exceed 65,535 bytes:

  ```
  mysql> CREATE TABLE t2
         (c1 VARCHAR(65535) NOT NULL)
         ENGINE = InnoDB CHARACTER SET latin1;
  ERROR 1118 (42000): Row size too large. The maximum row size for the used
  table type, not counting BLOBs, is 65535. This includes storage overhead,
  check the manual. You have to change some columns to TEXT or BLOBs
  ```

  Reducing the column length to 65,533 or less permits the statement to succeed.

  ```
  mysql> CREATE TABLE t2
         (c1 VARCHAR(65533) NOT NULL)
         ENGINE = InnoDB CHARACTER SET latin1;
  Query OK, 0 rows affected (0.01 sec)
  ```

- For `MyISAM` tables, `NULL` columns require additional space in the row to record whether their values are `NULL`. Each `NULL` column takes one bit extra, rounded up to the nearest byte.

  The statement to create table `t3` fails because `MyISAM` requires space for `NULL` columns in addition to the space required for variable-length column length bytes, causing the row size to exceed 65,535 bytes:

  ```
  mysql> CREATE TABLE t3
         (c1 VARCHAR(32765) NULL, c2 VARCHAR(32766) NULL)
         ENGINE = MyISAM CHARACTER SET latin1;
  ERROR 1118 (42000): Row size too large. The maximum row size for the used
  table type, not counting BLOBs, is 65535. This includes storage overhead,
  check the manual. You have to change some columns to TEXT or BLOBs
  ```

For information about `InnoDB` `NULL` column storage, see Section 15.10, “InnoDB Row Formats”.

- `InnoDB` restricts row size (for data stored locally within the database page) to slightly less than half a database page for 4KB, 8KB, 16KB, and 32KB `innodb_page_size` settings, and to slightly less than 16KB for 64KB pages.

  The statement to create table `t4` fails because the defined columns exceed the row size limit for a 16KB `InnoDB` page.

```
mysql> CREATE TABLE t4 (
       c1 CHAR(255),c2 CHAR(255),c3 CHAR(255),
       c4 CHAR(255),c5 CHAR(255),c6 CHAR(255),
       c7 CHAR(255),c8 CHAR(255),c9 CHAR(255),
       c10 CHAR(255),c11 CHAR(255),c12 CHAR(255),
       c13 CHAR(255),c14 CHAR(255),c15 CHAR(255),
       c16 CHAR(255),c17 CHAR(255),c18 CHAR(255),
       c19 CHAR(255),c20 CHAR(255),c21 CHAR(255),
       c22 CHAR(255),c23 CHAR(255),c24 CHAR(255),
       c25 CHAR(255),c26 CHAR(255),c27 CHAR(255),
       c28 CHAR(255),c29 CHAR(255),c30 CHAR(255),
       c31 CHAR(255),c32 CHAR(255),c33 CHAR(255)
       ) ENGINE=InnoDB ROW_FORMAT=DYNAMIC DEFAULT CHARSET latin1;
ERROR 1118 (42000): Row size too large (> 8126). Changing some columns to TEXT or BLOB may help.
In current row format, BLOB prefix of 0 bytes is stored inline.
```

8.5 Optimizing for InnoDB Tables

`InnoDB` is the storage engine that MySQL customers typically use in production databases where reliability and concurrency are important. `InnoDB` is the default storage engine in MySQL. This section explains how to optimize database operations for `InnoDB` tables.

8.5.1 Optimizing Storage Layout for InnoDB Tables

- Once your data reaches a stable size, or a growing table has increased by tens or some hundreds of megabytes, consider using the `OPTIMIZE TABLE` statement to reorganize the table and compact any wasted space. The reorganized tables require less disk I/O to perform full table scans. This is a straightforward technique that can improve performance when other techniques such as improving index usage or tuning application code are not practical.

  `OPTIMIZE TABLE` copies the data part of the table and rebuilds the indexes. The benefits come from improved packing of data within indexes, and reduced fragmentation within the tablespaces and on disk. The benefits vary depending on the data in each table. You may find that there are significant gains for some and not for others, or that the gains decrease over time until you next optimize the table. This operation can be slow if the table is large or if the indexes being rebuilt do not fit into the buffer pool. The first run after adding a lot of data to a table is often much slower than later runs.

- In `InnoDB`, having a long `PRIMARY KEY` (either a single column with a lengthy value, or several columns that form a long composite value) wastes a lot of disk space. The primary key value for a row is duplicated in all the secondary index records that point to the same row. (See Section 15.6.2.1, “Clustered and Secondary Indexes”.) Create an `AUTO_INCREMENT` column as the primary key if your primary key is long, or index a prefix of a long `VARCHAR` column instead of the entire column.

- Use the `VARCHAR` data type instead of `CHAR` to store variable-length strings or for columns with many `NULL` values. A `CHAR(N)` column always takes *`N`* characters to store data, even if the string is shorter or its value is `NULL`. Smaller tables fit better in the buffer pool and reduce disk I/O.

When using `COMPACT` row format (the default `InnoDB` format) and variable-length character sets, such as `utf8` or `sjis`, `CHAR(N)` columns occupy a variable amount of space, but still at least *`N`* bytes.

- For tables that are big, or contain lots of repetitive text or numeric data, consider using `COMPRESSED` row format. Less disk I/O is required to bring data into the buffer pool, or to perform full table scans. Before making a permanent decision, measure the amount of compression you can achieve by using `COMPRESSED` versus `COMPACT` row format.

8.5.2 Optimizing InnoDB Transaction Management

To optimize `InnoDB` transaction processing, find the ideal balance between the performance overhead of transactional features and the workload of your server. For example, an application might encounter performance issues if it commits thousands of times per second, and different performance issues if it commits only every 2-3 hours.

- The default MySQL setting `AUTOCOMMIT=1` can impose performance limitations on a busy database server. Where practical, wrap several related data change operations into a single transaction, by issuing `SET AUTOCOMMIT=0` or a `START TRANSACTION` statement, followed by a `COMMIT` statement after making all the changes.

  `InnoDB` must flush the log to disk at each transaction commit if that transaction made modifications to the database. When each change is followed by a commit (as with the default autocommit setting), the I/O throughput of the storage device puts a cap on the number of potential operations per second.

- Alternatively, for transactions that consist only of a single `SELECT` statement, turning on `AUTOCOMMIT` helps `InnoDB` to recognize read-only transactions and optimize them. See Section 8.5.3, “Optimizing InnoDB Read-Only Transactions” for requirements.

- Avoid performing rollbacks after inserting, updating, or deleting huge numbers of rows. If a big transaction is slowing down server performance, rolling it back can make the problem worse, potentially taking several times as long to perform as the original data change operations. Killing the database process does not help, because the rollback starts again on server startup.

  To minimize the chance of this issue occurring:

  - Increase the size of the buffer pool so that all the data change changes can be cached rather than immediately written to disk.
  - Set `innodb_change_buffering=all` so that update and delete operations are buffered in addition to inserts.
  - Consider issuing `COMMIT` statements periodically during the big data change operation, possibly breaking a single delete or update into multiple statements that operate on smaller numbers of rows.

  To get rid of a runaway rollback once it occurs, increase the buffer pool so that the rollback becomes CPU-bound and runs fast, or kill the server and restart with `innodb_force_recovery=3`, as explained in Section 15.17.2, “InnoDB Recovery”.

  This issue is expected to be infrequent with the default setting `innodb_change_buffering=all`, which allows update and delete operations to be cached in memory, making them faster to perform in the first place, and also faster to roll back if needed. Make sure to use this parameter setting on servers that process long-running transactions with many inserts, updates, or deletes.

- If you can afford the loss of some of the latest committed transactions if a crash occurs, you can set the `innodb_flush_log_at_trx_commit` parameter to 0. `InnoDB` tries to flush the log once per second anyway, although the flush is not guaranteed.

- When rows are modified or deleted, the rows and associated undo logs are not physically removed immediately, or even immediately after the transaction commits. The old data is preserved until transactions that started earlier or concurrently are finished, so that those transactions can access the previous state of modified or deleted rows. Thus, a long-running transaction can prevent `InnoDB` from purging data that was changed by a different transaction.
- When rows are modified or deleted within a long-running transaction, other transactions using the `READ COMMITTED` and `REPEATABLE READ` isolation levels have to do more work to reconstruct the older data if they read those same rows.
- When a long-running transaction modifies a table, queries against that table from other transactions do not make use of the covering index technique. Queries that normally could retrieve all the result columns from a secondary index, instead look up the appropriate values from the table data.

  If secondary index pages are found to have a `PAGE_MAX_TRX_ID` that is too new, or if records in the secondary index are delete-marked, `InnoDB` may need to look up records using a clustered index.

8.5.3 Optimizing InnoDB Read-Only Transactions

`InnoDB` can avoid the overhead associated with setting up the transaction ID (`TRX_ID` field) for transactions that are known to be read-only. A transaction ID is only needed for a transaction that might perform write operations or locking reads such as `SELECT ... FOR UPDATE`. Eliminating unnecessary transaction IDs reduces the size of internal data structures that are consulted each time a query or data change statement constructs a read view.

`InnoDB` detects read-only transactions when:

- The transaction is started with the `START TRANSACTION READ ONLY` statement. In this case, attempting to make changes to the database (for `InnoDB`, `MyISAM`, or other types of tables) causes an error, and the transaction continues in read-only state:

  ```
  ERROR 1792 (25006): Cannot execute statement in a READ ONLY transaction.
  ```

  You can still make changes to session-specific temporary tables in a read-only transaction, or issue locking queries for them, because those changes and locks are not visible to any other transaction.
- The `autocommit` setting is turned on, so that the transaction is guaranteed to be a single statement, and the single statement making up the transaction is a “non-locking” `SELECT` statement. That is, a `SELECT` that does not use a `FOR UPDATE` or `LOCK IN SHARED MODE` clause.
- The transaction is started without the `READ ONLY` option, but no updates or statements that explicitly lock rows have been executed yet. Until updates or explicit locks are required, a transaction stays in read-only mode.

Thus, for a read-intensive application such as a report generator, you can tune a sequence of `InnoDB` queries by grouping them inside `START TRANSACTION READ ONLY` and `COMMIT`, or by turning on the `autocommit` setting before running the `SELECT` statements, or simply by avoiding any data change statements interspersed with the queries.

For information about `START TRANSACTION` and `autocommit`, see Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

Note

Transactions that qualify as auto-commit, non-locking, and read-only (AC-NL-RO) are kept out of certain internal `InnoDB` data structures and are therefore not listed in `SHOW ENGINE INNODB STATUS` output.

8.5.4 Optimizing InnoDB Redo Logging

Consider the following guidelines for optimizing redo logging:

- Make your redo log files big, even as big as the buffer pool. When `InnoDB` has written the redo log files full, it must write the modified contents of the buffer pool to disk in a checkpoint. Small redo log files cause many unnecessary disk writes. Although historically big redo log files caused lengthy recovery times, recovery is now much faster and you can confidently use large redo log files.

  The size and number of redo log files are configured using the `innodb_log_file_size` and `innodb_log_files_in_group` configuration options. For information about modifying an existing redo log file configuration, see Changing the Number or Size of Redo Log Files.

- Consider increasing the size of the log buffer. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have transactions that update, insert, or delete many rows, making the log buffer larger saves disk I/O. Log buffer size is configured using the `innodb_log_buffer_size` configuration option, which can be configured dynamically in MySQL 8.0.

- Configure the `innodb_log_write_ahead_size` configuration option to avoid “read-on-write”. This option defines the write-ahead block size for the redo log. Set `innodb_log_write_ahead_size` to match the operating system or file system cache block size. Read-on-write occurs when redo log blocks are not entirely cached to the operating system or file system due to a mismatch between write-ahead block size for the redo log and operating system or file system cache block size.

  Valid values for `innodb_log_write_ahead_size` are multiples of the `InnoDB` log file block size (2^n). The minimum value is the `InnoDB` log file block size (512). Write-ahead does not occur when the minimum value is specified. The maximum value is equal to the `innodb_page_size` value. If you specify a value for `innodb_log_write_ahead_size` that is larger than the `innodb_page_size` value, the `innodb_log_write_ahead_size` setting is truncated to the `innodb_page_size` value.

  Setting the `innodb_log_write_ahead_size` value too low in relation to the operating system or file system cache block size results in read-on-write. Setting the value too high may have a slight impact on `fsync` performance for log file writes due to several blocks being written at once.

- Optimize the use of spin delay by user threads waiting for flushed redo. Spin delay helps reduce latency. During periods of low concurrency, reducing latency may be less of a priority, and avoiding the use of spin delay during these periods may reduce energy consumption. During periods of high concurrency, you may want to avoid expending processing power on spin delay so that it can be used for other work. The following system variables permit setting high and low watermark values that define boundaries for the use of spin delay.

  - `innodb_log_wait_for_flush_spin_hwm`: Defines the maximum average log flush time beyond which user threads no longer spin while waiting for flushed redo. The default value is 400 microseconds.

  - `innodb_log_spin_cpu_abs_lwm`: Defines the minimum amount of CPU usage below which user threads no longer spin while waiting for flushed redo. The value is expressed as a sum of CPU core usage. For example, The default value of 80 is 80% of a single CPU core. On a system with a multi-core processor, a value of 150 represents 100% usage of one CPU core plus 50% usage of a second CPU core.

  - `innodb_log_spin_cpu_pct_hwm`: Defines the maximum amount of CPU usage above which user threads no longer spin while waiting for flushed redo. The value is expressed as a percentage of the combined total processing power of all CPU cores. The default value is 50%. For example, 100%

usage of two CPU cores is 50% of the combined CPU processing power on a server with four CPU cores.

The `innodb_log_spin_cpu_pct_hwm` configuration option respects processor affinity. For example, if a server has 48 cores but the `mysqld` process is pinned to only four CPU cores, the other 44 CPU cores are ignored.

8.5.5 Bulk Data Loading for InnoDB Tables

These performance tips supplement the general guidelines for fast inserts in Section 8.2.5.1, “Optimizing INSERT Statements”.

- When importing data into `InnoDB`, turn off autocommit mode, because it performs a log flush to disk for every insert. To disable autocommit during your import operation, surround it with `SET autocommit` and `COMMIT` statements:

```
SET autocommit=0;
... SQL import statements ...
COMMIT;
```

  The `mysqldump` option `--opt` creates dump files that are fast to import into an `InnoDB` table, even without wrapping them with the `SET autocommit` and `COMMIT` statements.

- If you have `UNIQUE` constraints on secondary keys, you can speed up table imports by temporarily turning off the uniqueness checks during the import session:

```
SET unique_checks=0;
... SQL import statements ...
SET unique_checks=1;
```

  For big tables, this saves a lot of disk I/O because `InnoDB` can use its change buffer to write secondary index records in a batch. Be certain that the data contains no duplicate keys.

- If you have `FOREIGN KEY` constraints in your tables, you can speed up table imports by turning off the foreign key checks for the duration of the import session:

```
SET foreign_key_checks=0;
... SQL import statements ...
SET foreign_key_checks=1;
```

  For big tables, this can save a lot of disk I/O.

- Use the multiple-row `INSERT` syntax to reduce communication overhead between the client and the server if you need to insert many rows:

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

  This tip is valid for inserts into any table, not just `InnoDB` tables.

- When doing bulk inserts into tables with auto-increment columns, set `innodb_autoinc_lock_mode` to 2 (interleaved) instead of 1 (consecutive). See Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB” for details.

- When performing bulk inserts, it is faster to insert rows in `PRIMARY KEY` order. `InnoDB` tables use a clustered index, which makes it relatively fast to use data in the order of the `PRIMARY KEY`. Performing

bulk inserts in PRIMARY KEY order is particularly important for tables that do not fit entirely within the buffer pool.

- For optimal performance when loading data into an InnoDB FULLTEXT index, follow this set of steps:

  1. Define a column FTS_DOC_ID at table creation time, of type BIGINT UNSIGNED NOT NULL, with a unique index named FTS_DOC_ID_INDEX. For example:

```
CREATE TABLE t1 (
FTS_DOC_ID BIGINT unsigned NOT NULL AUTO_INCREMENT,
title varchar(255) NOT NULL DEFAULT '',
text mediumtext NOT NULL,
PRIMARY KEY (`FTS_DOC_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
CREATE UNIQUE INDEX FTS_DOC_ID_INDEX on t1(FTS_DOC_ID);
```

  2. Load the data into the table.

  3. Create the FULLTEXT index after the data is loaded.

Note

When adding FTS_DOC_ID column at table creation time, ensure that the FTS_DOC_ID column is updated when the FULLTEXT indexed column is updated, as the FTS_DOC_ID must increase monotonically with each INSERT or UPDATE. If you choose not to add the FTS_DOC_ID at table creation time and have InnoDB manage DOC IDs for you, InnoDB will add the FTS_DOC_ID as a hidden column with the next CREATE FULLTEXT INDEX call. This approach, however, requires a table rebuild which will impact performance.

8.5.6 Optimizing InnoDB Queries

To tune queries for InnoDB tables, create an appropriate set of indexes on each table. See Section 8.3.1, “How MySQL Uses Indexes” for details. Follow these guidelines for InnoDB indexes:

- Because each InnoDB table has a primary key (whether you request one or not), specify a set of primary key columns for each table, columns that are used in the most important and time-critical queries.

- Do not specify too many or too long columns in the primary key, because these column values are duplicated in each secondary index. When an index contains unnecessary data, the I/O to read this data and memory to cache it reduce the performance and scalability of the server.

- Do not create a separate secondary index for each column, because each query can only make use of one index. Indexes on rarely tested columns or columns with only a few different values might not be helpful for any queries. If you have many queries for the same table, testing different combinations of columns, try to create a small number of concatenated indexes rather than a large number of single-column indexes. If an index contains all the columns needed for the result set (known as a covering index), the query might be able to avoid reading the table data at all.

- If an indexed column cannot contain any NULL values, declare it as NOT NULL when you create the table. The optimizer can better determine which index is most effective to use for a query, when it knows whether each column contains NULL values.

- You can optimize single-query transactions for InnoDB tables, using the technique in Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”.

8.5.7 Optimizing InnoDB DDL Operations

- Many DDL operations on tables and indexes (`CREATE`, `ALTER`, and `DROP` statements) can be performed online. See Section 15.12, “InnoDB and Online DDL” for details.
- Online DDL support for adding secondary indexes means that you can generally speed up the process of creating and loading a table and associated indexes by creating the table without secondary indexes, then adding secondary indexes after the data is loaded.
- Use `TRUNCATE TABLE` to empty a table, not `DELETE FROM tbl_name`. Foreign key constraints can make a `TRUNCATE` statement work like a regular `DELETE` statement, in which case a sequence of commands like `DROP TABLE` and `CREATE TABLE` might be fastest.
- Because the primary key is integral to the storage layout of each `InnoDB` table, and changing the definition of the primary key involves reorganizing the whole table, always set up the primary key as part of the `CREATE TABLE` statement, and plan ahead so that you do not need to `ALTER` or `DROP` the primary key afterward.

8.5.8 Optimizing InnoDB Disk I/O

If you follow best practices for database design and tuning techniques for SQL operations, but your database is still slow due to heavy disk I/O activity, consider these disk I/O optimizations. If the Unix `top` tool or the Windows Task Manager shows that the CPU usage percentage with your workload is less than 70%, your workload is probably disk-bound.

- Increase buffer pool size

  When table data is cached in the `InnoDB` buffer pool, it can be accessed repeatedly by queries without requiring any disk I/O. Specify the size of the buffer pool with the `innodb_buffer_pool_size` option. This memory area is important enough that it is typically recommended that `innodb_buffer_pool_size` is configured to 50 to 75 percent of system memory. For more information see, Section 8.12.3.1, “How MySQL Uses Memory”.

- Adjust the flush method

  In some versions of GNU/Linux and Unix, flushing files to disk with the Unix `fsync()` call (which `InnoDB` uses by default) and similar methods is surprisingly slow. If database write performance is an issue, conduct benchmarks with the `innodb_flush_method` parameter set to `O_DSYNC`.

- Configure an fsync threshold

  By default, when `InnoDB` creates a new data file, such as a new log file or tablespace file, the file is fully written to the operating system cache before it is flushed to disk, which can cause a large amount of disk write activity to occur at once. To force smaller, periodic flushes of data from the operating system cache, you can use the `innodb_fsync_threshold` variable to define a threshold value, in bytes. When the byte threshold is reached, the contents of the operating system cache are flushed to disk. The default value of 0 forces the default behavior, which is to flush data to disk only after a file is fully written to the cache.

  Specifying a threshold to force smaller, periodic flushes may be beneficial in cases where multiple MySQL instances use the same storage devices. For example, creating a new MySQL instance and its associated data files could cause large surges of disk write activity, impeding the performance of other MySQL instances that use the same storage devices. Configuring a threshold helps avoid such surges in write activity.

- Use a noop or deadline I/O scheduler with native AIO on Linux

`InnoDB` uses the asynchronous I/O subsystem (native AIO) on Linux to perform read-ahead and write requests for data file pages. This behavior is controlled by the `innodb_use_native_aio` configuration option, which is enabled by default. With native AIO, the type of I/O scheduler has greater influence on I/O performance. Generally, noop and deadline I/O schedulers are recommended. Conduct benchmarks to determine which I/O scheduler provides the best results for your workload and environment. For more information, see Section 15.8.6, “Using Asynchronous I/O on Linux”.

- Use direct I/O on Solaris 10 for x86_64 architecture

  When using the `InnoDB` storage engine on Solaris 10 for x86_64 architecture (AMD Opteron), use direct I/O for `InnoDB`-related files to avoid degradation of `InnoDB` performance. To use direct I/O for an entire UFS file system used for storing `InnoDB`-related files, mount it with the `forcedirectio` option; see `mount_ufs(1M)`. (The default on Solaris 10/x86_64 is *not* to use this option.) To apply direct I/O only to `InnoDB` file operations rather than the whole file system, set `innodb_flush_method = O_DIRECT`. With this setting, `InnoDB` calls `directio()` instead of `fcntl()` for I/O to data files (not for I/O to log files).

- Use raw storage for data and log files with Solaris 2.6 or later

  When using the `InnoDB` storage engine with a large `innodb_buffer_pool_size` value on any release of Solaris 2.6 and up and any platform (sparc/x86/x64/amd64), conduct benchmarks with `InnoDB` data files and log files on raw devices or on a separate direct I/O UFS file system, using the `forcedirectio` mount option as described previously. (It is necessary to use the mount option rather than setting `innodb_flush_method` if you want direct I/O for the log files.) Users of the Veritas file system VxFS should use the `convosync=direct` mount option.

  Do not place other MySQL data files, such as those for `MyISAM` tables, on a direct I/O file system. Executables or libraries *must not* be placed on a direct I/O file system.

- Use additional storage devices

  Additional storage devices could be used to set up a RAID configuration. For related information, see Section 8.12.1, “Optimizing Disk I/O”.

  Alternatively, `InnoDB` tablespace data files and log files can be placed on different physical disks. For more information, refer to the following sections:

  - Section 15.8.1, “InnoDB Startup Configuration”
  - Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”
  - Creating a General Tablespace
  - Section 15.6.1.2, “Moving or Copying InnoDB Tables”

- Consider non-rotational storage

  Non-rotational storage generally provides better performance for random I/O operations; and rotational storage for sequential I/O operations. When distributing data and log files across rotational and non-rotational storage devices, consider the type of I/O operations that are predominantly performed on each file.

  Random I/O-oriented files typically include file-per-table and general tablespace data files, undo tablespace files, and temporary tablespace files. Sequential I/O-oriented files include `InnoDB` system tablespace files (due to doublewrite buffering and change buffering) and log files such as binary log files and redo log files.

Review settings for the following configuration options when using non-rotational storage:

- `innodb_checksum_algorithm`

  The `crc32` option uses a faster checksum algorithm and is recommended for fast storage systems.

- `innodb_flush_neighbors`

  Optimizes I/O for rotational storage devices. Disable it for non-rotational storage or a mix of rotational and non-rotational storage. It is disabled by default.

- `innodb_idle_flush_pct`

  Permits placing a limit on page flushing during idle periods, which can help extend the life of non-rotational storage devices. Introduced in MySQL 8.0.18.

- `innodb_io_capacity`

  The default setting of 200 is generally sufficient for a lower-end non-rotational storage device. For higher-end, bus-attached devices, consider a higher setting such as 1000.

- `innodb_io_capacity_max`

  The default value of 2000 is intended for workloads that use non-rotational storage. For a high-end, bus-attached non-rotational storage device, consider a higher setting such as 2500.

- `innodb_log_compressed_pages`

  If redo logs are on non-rotational storage, consider disabling this option to reduce logging. See Disable logging of compressed pages.

- `innodb_log_file_size`

  If redo logs are on non-rotational storage, configure this option to maximize caching and write combining.

- `innodb_page_size`

  Consider using a page size that matches the internal sector size of the disk. Early-generation SSD devices often have a 4KB sector size. Some newer devices have a 16KB sector size. The default `InnoDB` page size is 16KB. Keeping the page size close to the storage device block size minimizes the amount of unchanged data that is rewritten to disk.

- `binlog_row_image`

  If binary logs are on non-rotational storage and all tables have primary keys, consider setting this option to `minimal` to reduce logging.

Ensure that TRIM support is enabled for your operating system. It is typically enabled by default.

- Increase I/O capacity to avoid backlogs

  If throughput drops periodically because of `InnoDB` checkpoint operations, consider increasing the value of the `innodb_io_capacity` configuration option. Higher values cause more frequent flushing, avoiding the backlog of work that can cause dips in throughput.

- Lower I/O capacity if flushing does not fall behind

If the system is not falling behind with `InnoDB` flushing operations, consider lowering the value of the `innodb_io_capacity` configuration option. Typically, you keep this option value as low as practical, but not so low that it causes periodic drops in throughput as mentioned in the preceding bullet. In a typical scenario where you could lower the option value, you might see a combination like this in the output from `SHOW ENGINE INNODB STATUS`:

- History list length low, below a few thousand.
- Insert buffer merges close to rows inserted.
- Modified pages in buffer pool consistently well below `innodb_max_dirty_pages_pct` of the buffer pool. (Measure at a time when the server is not doing bulk inserts; it is normal during bulk inserts for the modified pages percentage to rise significantly.)
- `Log sequence number - Last checkpoint` is at less than 7/8 or ideally less than 6/8 of the total size of the `InnoDB` log files.

- Store system tablespace files on Fusion-io devices

  You can take advantage of a doublewrite buffer-related I/O optimization by storing system tablespace files (“ibdata files”) on Fusion-io devices that support atomic writes. In this case, doublewrite buffering (`innodb_doublewrite`) is automatically disabled and Fusion-io atomic writes are used for all data files. This feature is only supported on Fusion-io hardware and is only enabled for Fusion-io NVMFS on Linux. To take full advantage of this feature, an `innodb_flush_method` setting of `O_DIRECT` is recommended.

  Note

  Because the doublewrite buffer setting is global, doublewrite buffering is also disabled for data files residing on non-Fusion-io hardware.

- Disable logging of compressed pages

  When using the `InnoDB` table compression feature, images of re-compressed pages are written to the redo log when changes are made to compressed data. This behavior is controlled by `innodb_log_compressed_pages`, which is enabled by default to prevent corruption that can occur if a different version of the `zlib` compression algorithm is used during recovery. If you are certain that the `zlib` version will not change, disable `innodb_log_compressed_pages` to reduce redo log generation for workloads that modify compressed data.

8.5.9 Optimizing InnoDB Configuration Variables

Different settings work best for servers with light, predictable loads, versus servers that are running near full capacity all the time, or that experience spikes of high activity.

Because the `InnoDB` storage engine performs many of its optimizations automatically, many performance-tuning tasks involve monitoring to ensure that the database is performing well, and changing configuration options when performance drops. See Section 15.15, “InnoDB Integration with MySQL Performance Schema” for information about detailed `InnoDB` performance monitoring.

The main configuration steps you can perform include:

- Controlling the types of data change operations for which `InnoDB` buffers the changed data, to avoid frequent small disk writes. See Configuring Change Buffering. Because the default is to buffer all types of data change operations, only change this setting if you need to reduce the amount of buffering.

- Turning the adaptive hash indexing feature on and off using the `innodb_adaptive_hash_index` option. See Section 15.5.3, “Adaptive Hash Index” for more information. You might change this setting during periods of unusual activity, then restore it to its original setting.
- Setting a limit on the number of concurrent threads that `InnoDB` processes, if context switching is a bottleneck. See Section 15.8.4, “Configuring Thread Concurrency for InnoDB”.
- Controlling the amount of prefetching that `InnoDB` does with its read-ahead operations. When the system has unused I/O capacity, more read-ahead can improve the performance of queries. Too much read-ahead can cause periodic drops in performance on a heavily loaded system. See Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”.
- Increasing the number of background threads for read or write operations, if you have a high-end I/O subsystem that is not fully utilized by the default values. See Section 15.8.5, “Configuring the Number of Background InnoDB I/O Threads”.
- Controlling how much I/O `InnoDB` performs in the background. See Section 15.8.7, “Configuring InnoDB I/O Capacity”. You might scale back this setting if you observe periodic drops in performance.
- Controlling the algorithm that determines when `InnoDB` performs certain types of background writes. See Section 15.8.3.5, “Configuring Buffer Pool Flushing”. The algorithm works for some types of workloads but not others, so you might disable this feature if you observe periodic drops in performance.
- Taking advantage of multicore processors and their cache memory configuration, to minimize delays in context switching. See Section 15.8.8, “Configuring Spin Lock Polling”.
- Preventing one-time operations such as table scans from interfering with the frequently accessed data stored in the `InnoDB` buffer cache. See Section 15.8.3.3, “Making the Buffer Pool Scan Resistant”.
- Adjusting log files to a size that makes sense for reliability and crash recovery. `InnoDB` log files have often been kept small to avoid long startup times after a crash. Optimizations introduced in MySQL 5.5 speed up certain steps of the crash recovery process. In particular, scanning the redo log and applying the redo log are faster due to improved algorithms for memory management. If you have kept your log files artificially small to avoid long startup times, you can now consider increasing log file size to reduce the I/O that occurs due recycling of redo log records.
- Configuring the size and number of instances for the `InnoDB` buffer pool, especially important for systems with multi-gigabyte buffer pools. See Section 15.8.3.2, “Configuring Multiple Buffer Pool Instances”.
- Increasing the maximum number of concurrent transactions, which dramatically improves scalability for the busiest databases. See Section 15.6.6, “Undo Logs”.
- Moving purge operations (a type of garbage collection) into a background thread. See Section 15.8.9, “Purge Configuration”. To effectively measure the results of this setting, tune the other I/O-related and thread-related configuration settings first.
- Reducing the amount of switching that `InnoDB` does between concurrent threads, so that SQL operations on a busy server do not queue up and form a “traffic jam”. Set a value for the `innodb_thread_concurrency` option, up to approximately 32 for a high-powered modern system. Increase the value for the `innodb_concurrency_tickets` option, typically to 5000 or so. This combination of options sets a cap on the number of threads that `InnoDB` processes at any one time, and allows each thread to do substantial work before being swapped out, so that the number of waiting threads stays low and operations can complete without excessive context switching.

8.5.10 Optimizing InnoDB for Systems with Many Tables

- If you have configured non-persistent optimizer statistics (a non-default configuration), `InnoDB` computes index cardinality values for a table the first time that table is accessed after startup, instead of storing such values in the table. This step can take significant time on systems that partition the data into many tables. Since this overhead only applies to the initial table open operation, to "warm up" a table for later use, access it immediately after startup by issuing a statement such as `SELECT 1 FROM tbl_name LIMIT 1`.

  Optimizer statistics are persisted to disk by default, enabled by the `innodb_stats_persistent` configuration option. For information about persistent optimizer statistics, see Section 15.8.10.1, "Configuring Persistent Optimizer Statistics Parameters".

8.6 Optimizing for MyISAM Tables

The `MyISAM` storage engine performs best with read-mostly data or with low-concurrency operations, because table locks limit the ability to perform simultaneous updates. In MySQL, `InnoDB` is the default storage engine rather than `MyISAM`.

8.6.1 Optimizing MyISAM Queries

Some general tips for speeding up queries on `MyISAM` tables:

- To help MySQL better optimize queries, use `ANALYZE TABLE` or run `myisamchk --analyze` on a table after it has been loaded with data. This updates a value for each index part that indicates the average number of rows that have the same value. (For unique indexes, this is always 1.) MySQL uses this to decide which index to choose when you join two tables based on a nonconstant expression. You can check the result from the table analysis by using `SHOW INDEX FROM tbl_name` and examining the `Cardinality` value. `myisamchk --description --verbose` shows index distribution information.

- To sort an index and data according to an index, use `myisamchk --sort-index --sort-records=1` (assuming that you want to sort on index 1). This is a good way to make queries faster if you have a unique index from which you want to read all rows in order according to the index. The first time you sort a large table this way, it may take a long time.

- Try to avoid complex `SELECT` queries on `MyISAM` tables that are updated frequently, to avoid problems with table locking that occur due to contention between readers and writers.

- `MyISAM` supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. If it is important to be able to do this, consider using the table in ways that avoid deleting rows. Another possibility is to run `OPTIMIZE TABLE` to defragment the table after you have deleted a lot of rows from it. This behavior is altered by setting the `concurrent_insert` variable. You can force new rows to be appended (and therefore permit concurrent inserts), even in tables that have deleted rows. See Section 8.11.3, "Concurrent Inserts".

- For `MyISAM` tables that change frequently, try to avoid all variable-length columns (`VARCHAR`, `BLOB`, and `TEXT`). The table uses dynamic row format if it includes even a single variable-length column. See Chapter 16, *Alternative Storage Engines*.

- It is normally not useful to split a table into different tables just because the rows become large. In accessing a row, the biggest performance hit is the disk seek needed to find the first byte of the row. After finding the data, most modern disks can read the entire row fast enough for most applications. The only cases where splitting up a table makes an appreciable difference is if it is a `MyISAM` table using dynamic row format that you can change to a fixed row size, or if you very often need to scan the table but do not need most of the columns. See Chapter 16, *Alternative Storage Engines*.

- Use `ALTER TABLE ... ORDER BY expr1, expr2, ...` if you usually retrieve rows in `expr1, expr2, ...` order. By using this option after extensive changes to the table, you may be able to get higher performance.
- If you often need to calculate results such as counts based on information from a lot of rows, it may be preferable to introduce a new table and update the counter in real time. An update of the following form is very fast:

  ```
  UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
  ```

  This is very important when you use MySQL storage engines such as `MyISAM` that has only table-level locking (multiple readers with single writers). This also gives better performance with most database systems, because the row locking manager in this case has less to do.
- Use `OPTIMIZE TABLE` periodically to avoid fragmentation with dynamic-format `MyISAM` tables. See Section 16.2.3, “MyISAM Table Storage Formats”.
- Declaring a `MyISAM` table with the `DELAY_KEY_WRITE=1` table option makes index updates faster because they are not flushed to disk until the table is closed. The downside is that if something kills the server while such a table is open, you must ensure that the table is okay by running the server with the `myisam_recover_options` system variable set, or by running `myisamchk` before restarting the server. (However, even in this case, you should not lose anything by using `DELAY_KEY_WRITE`, because the key information can always be generated from the data rows.)
- Strings are automatically prefix- and end-space compressed in `MyISAM` indexes. See Section 13.1.15, “CREATE INDEX Syntax”.
- You can increase performance by caching queries or answers in your application and then executing many inserts or updates together. Locking the table during this operation ensures that the index cache is only flushed once after all updates.

8.6.2 Bulk Data Loading for MyISAM Tables

These performance tips supplement the general guidelines for fast inserts in Section 8.2.5.1, “Optimizing INSERT Statements”.

- For a `MyISAM` table, you can use concurrent inserts to add rows at the same time that `SELECT` statements are running, if there are no deleted rows in middle of the data file. See Section 8.11.3, “Concurrent Inserts”.
- With some extra work, it is possible to make `LOAD DATA` run even faster for a `MyISAM` table when the table has many indexes. Use the following procedure:

  1. Execute a `FLUSH TABLES` statement or a `mysqladmin flush-tables` command.
  2. Use `myisamchk --keys-used=0 -rq /path/to/db/tbl_name` to remove all use of indexes for the table.
  3. Insert data into the table with `LOAD DATA`. This does not update any indexes and therefore is very fast.
  4. If you intend only to read from the table in the future, use `myisampack` to compress it. See Section 16.2.3.3, “Compressed Table Characteristics”.
  5. Re-create the indexes with `myisamchk -rq /path/to/db/tbl_name`. This creates the index tree in memory before writing it to disk, which is much faster than updating the index during `LOAD DATA` because it avoids lots of disk seeks. The resulting index tree is also perfectly balanced.

6. Execute a `FLUSH TABLES` statement or a `mysqladmin flush-tables` command.

`LOAD DATA` performs the preceding optimization automatically if the `MyISAM` table into which you insert data is empty. The main difference between automatic optimization and using the procedure explicitly is that you can let `myisamchk` allocate much more temporary memory for the index creation than you might want the server to allocate for index re-creation when it executes the `LOAD DATA` statement.

You can also disable or enable the nonunique indexes for a `MyISAM` table by using the following statements rather than `myisamchk`. If you use these statements, you can skip the `FLUSH TABLES` operations:

```
ALTER TABLE tbl_name DISABLE KEYS;
ALTER TABLE tbl_name ENABLE KEYS;
```

- To speed up `INSERT` operations that are performed with multiple statements for nontransactional tables, lock your tables:

```
LOCK TABLES a WRITE;
INSERT INTO a VALUES (1,23),(2,34),(4,33);
INSERT INTO a VALUES (8,26),(6,29);
...
UNLOCK TABLES;
```

This benefits performance because the index buffer is flushed to disk only once, after all `INSERT` statements have completed. Normally, there would be as many index buffer flushes as there are `INSERT` statements. Explicit locking statements are not needed if you can insert all rows with a single `INSERT`.

Locking also lowers the total time for multiple-connection tests, although the maximum wait time for individual connections might go up because they wait for locks. Suppose that five clients attempt to perform inserts simultaneously as follows:

- Connection 1 does 1000 inserts
- Connections 2, 3, and 4 do 1 insert
- Connection 5 does 1000 inserts

If you do not use locking, connections 2, 3, and 4 finish before 1 and 5. If you use locking, connections 2, 3, and 4 probably do not finish before 1 or 5, but the total time should be about 40% faster.

`INSERT`, `UPDATE`, and `DELETE` operations are very fast in MySQL, but you can obtain better overall performance by adding locks around everything that does more than about five successive inserts or updates. If you do very many successive inserts, you could do a `LOCK TABLES` followed by an `UNLOCK TABLES` once in a while (each 1,000 rows or so) to permit other threads to access table. This would still result in a nice performance gain.

`INSERT` is still much slower for loading data than `LOAD DATA`, even when using the strategies just outlined.

- To increase performance for `MyISAM` tables, for both `LOAD DATA` and `INSERT`, enlarge the key cache by increasing the `key_buffer_size` system variable. See Section 5.1.1, “Configuring the Server”.

8.6.3 Optimizing REPAIR TABLE Statements

`REPAIR TABLE` for `MyISAM` tables is similar to using `myisamchk` for repair operations, and some of the same performance optimizations apply:

- `myisamchk` has variables that control memory allocation. You may be able to its improve performance by setting these variables, as described in Section 4.6.4.6, “myisamchk Memory Usage”.
- For `REPAIR TABLE`, the same principle applies, but because the repair is done by the server, you set server system variables instead of `myisamchk` variables. Also, in addition to setting memory-allocation variables, increasing the `myisam_max_sort_file_size` system variable increases the likelihood that the repair will use the faster filesort method and avoid the slower repair by key cache method. Set the variable to the maximum file size for your system, after checking to be sure that there is enough free space to hold a copy of the table files. The free space must be available in the file system containing the original table files.

Suppose that a `myisamchk` table-repair operation is done using the following options to set its memory-allocation variables:

```
--key_buffer_size=128M --myisam_sort_buffer_size=256M
--read_buffer_size=64M --write_buffer_size=64M
```

Some of those `myisamchk` variables correspond to server system variables:

`myisamchk` Variable	**System Variable**
`key_buffer_size`	`key_buffer_size`
`myisam_sort_buffer_size`	`myisam_sort_buffer_size`
`read_buffer_size`	`read_buffer_size`
`write_buffer_size`	none

Each of the server system variables can be set at runtime, and some of them (`myisam_sort_buffer_size`, `read_buffer_size`) have a session value in addition to a global value. Setting a session value limits the effect of the change to your current session and does not affect other users. Changing a global-only variable (`key_buffer_size`, `myisam_max_sort_file_size`) affects other users as well. For `key_buffer_size`, you must take into account that the buffer is shared with those users. For example, if you set the `myisamchk key_buffer_size` variable to 128MB, you could set the corresponding `key_buffer_size` system variable larger than that (if it is not already set larger), to permit key buffer use by activity in other sessions. However, changing the global key buffer size invalidates the buffer, causing increased disk I/O and slowdown for other sessions. An alternative that avoids this problem is to use a separate key cache, assign to it the indexes from the table to be repaired, and deallocate it when the repair is complete. See Section 8.10.2.2, “Multiple Key Caches”.

Based on the preceding remarks, a `REPAIR TABLE` operation can be done as follows to use settings similar to the `myisamchk` command. Here a separate 128MB key buffer is allocated and the file system is assumed to permit a file size of at least 100GB.

```
SET SESSION myisam_sort_buffer_size = 256*1024*1024;
SET SESSION read_buffer_size = 64*1024*1024;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
SET GLOBAL repair_cache.key_buffer_size = 128*1024*1024;
CACHE INDEX tbl_name IN repair_cache;
LOAD INDEX INTO CACHE tbl_name;
REPAIR TABLE tbl_name ;
SET GLOBAL repair_cache.key_buffer_size = 0;
```

If you intend to change a global variable but want to do so only for the duration of a `REPAIR TABLE` operation to minimally affect other users, save its value in a user variable and restore it afterward. For example:

```
SET @old_myisam_sort_buffer_size = @@GLOBAL.myisam_max_sort_file_size;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
REPAIR TABLE tbl_name ;
SET GLOBAL myisam_max_sort_file_size = @old_myisam_max_sort_file_size;
```

The system variables that affect `REPAIR TABLE` can be set globally at server startup if you want the values to be in effect by default. For example, add these lines to the server `my.cnf` file:

```
[mysqld]
myisam_sort_buffer_size=256M
key_buffer_size=1G
myisam_max_sort_file_size=100G
```

These settings do not include `read_buffer_size`. Setting `read_buffer_size` globally to a large value does so for all sessions and can cause performance to suffer due to excessive memory allocation for a server with many simultaneous sessions.

8.7 Optimizing for MEMORY Tables

Consider using `MEMORY` tables for noncritical data that is accessed often, and is read-only or rarely updated. Benchmark your application against equivalent `InnoDB` or `MyISAM` tables under a realistic workload, to confirm that any additional performance is worth the risk of losing data, or the overhead of copying data from a disk-based table at application start.

For best performance with `MEMORY` tables, examine the kinds of queries against each table, and specify the type to use for each associated index, either a B-tree index or a hash index. On the `CREATE INDEX` statement, use the clause `USING BTREE` or `USING HASH`. B-tree indexes are fast for queries that do greater-than or less-than comparisons through operators such as `>` or `BETWEEN`. Hash indexes are only fast for queries that look up single values through the `=` operator, or a restricted set of values through the `IN` operator. For why `USING BTREE` is often a better choice than the default `USING HASH`, see Section 8.2.1.23, “Avoiding Full Table Scans”. For implementation details of the different types of `MEMORY` indexes, see Section 8.3.9, “Comparison of B-Tree and Hash Indexes”.

8.8 Understanding the Query Execution Plan

Depending on the details of your tables, columns, indexes, and the conditions in your `WHERE` clause, the MySQL optimizer considers many techniques to efficiently perform the lookups involved in an SQL query. A query on a huge table can be performed without reading all the rows; a join involving several tables can be performed without comparing every combination of rows. The set of operations that the optimizer chooses to perform the most efficient query is called the “query execution plan”, also known as the `EXPLAIN` plan. Your goals are to recognize the aspects of the `EXPLAIN` plan that indicate a query is optimized well, and to learn the SQL syntax and indexing techniques to improve the plan if you see some inefficient operations.

8.8.1 Optimizing Queries with EXPLAIN

The `EXPLAIN` statement provides information about how MySQL executes statements:

- `EXPLAIN` works with `SELECT`, `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` statements.
- When `EXPLAIN` is used with an explainable statement, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. For information about using `EXPLAIN` to obtain execution plan information, see Section 8.8.2, “EXPLAIN Output Format”.

- When `EXPLAIN` is used with `FOR CONNECTION connection_id` rather than an explainable statement, it displays the execution plan for the statement executing in the named connection. See Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”.
- For `SELECT` statements, `EXPLAIN` produces additional execution plan information that can be displayed using `SHOW WARNINGS`. See Section 8.8.3, “Extended EXPLAIN Output Format”.
- `EXPLAIN` is useful for examining queries involving partitioned tables. See Section 23.3.5, “Obtaining Information About Partitions”.
- The `FORMAT` option can be used to select the output format. `TRADITIONAL` presents the output in tabular format. This is the default if no `FORMAT` option is present. `JSON` format displays the information in JSON format.

With the help of `EXPLAIN`, you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use `EXPLAIN` to check whether the optimizer joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in a `SELECT` statement, begin the statement with `SELECT STRAIGHT_JOIN` rather than just `SELECT`. (See Section 13.2.10, “SELECT Syntax”.) However, `STRAIGHT_JOIN` may prevent indexes from being used because it disables semijoin transformations. See Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”.

The optimizer trace may sometimes provide information complementary to that of `EXPLAIN`. However, the optimizer trace format and content are subject to change between versions. For details, see MySQL Internals: Tracing the Optimizer.

If you have a problem with indexes not being used when you believe that they should be, run `ANALYZE TABLE` to update table statistics, such as cardinality of keys, that can affect the choices the optimizer makes. See Section 13.7.3.1, “ANALYZE TABLE Syntax”.

Note

`EXPLAIN` can also be used to obtain information about the columns in a table. `EXPLAIN tbl_name` is synonymous with `DESCRIBE tbl_name` and `SHOW COLUMNS FROM tbl_name`. For more information, see Section 13.8.1, “DESCRIBE Syntax”, and Section 13.7.7.5, “SHOW COLUMNS Syntax”.

8.8.2 EXPLAIN Output Format

The `EXPLAIN` statement provides information about how MySQL executes statements. `EXPLAIN` works with `SELECT`, `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` statements.

`EXPLAIN` returns a row of information for each table used in the `SELECT` statement. It lists the tables in the output in the order that MySQL would read them while processing the statement. MySQL resolves all joins using a nested-loop join method. This means that MySQL reads a row from the first table, and then finds a matching row in the second table, the third table, and so on. When all tables are processed, MySQL outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

Note

MySQL Workbench has a Visual Explain capability that provides a visual representation of `EXPLAIN` output. See Tutorial: Using Explain to Improve Query Performance.

- EXPLAIN Output Columns
- EXPLAIN Join Types

- EXPLAIN Extra Information
- EXPLAIN Output Interpretation

EXPLAIN Output Columns

This section describes the output columns produced by `EXPLAIN`. Later sections provide additional information about the `type` and `Extra` columns.

Each output row from `EXPLAIN` provides information about one table. Each row contains the values summarized in Table 8.1, "EXPLAIN Output Columns", and described in more detail following the table. Column names are shown in the table's first column; the second column provides the equivalent property name shown in the output when `FORMAT=JSON` is used.

Table 8.1 EXPLAIN Output Columns

Column	JSON Name	Meaning
`id`	`select_id`	The `SELECT` identifier
`select_type`	None	The `SELECT` type
`table`	`table_name`	The table for the output row
`partitions`	`partitions`	The matching partitions
`type`	`access_type`	The join type
`possible_keys`	`possible_keys`	The possible indexes to choose
`key`	`key`	The index actually chosen
`key_len`	`key_length`	The length of the chosen key
`ref`	`ref`	The columns compared to the index
`rows`	`rows`	Estimate of rows to be examined
`filtered`	`filtered`	Percentage of rows filtered by table condition
`Extra`	None	Additional information

Note

JSON properties which are `NULL` are not displayed in JSON-formatted `EXPLAIN` output.

- `id` (JSON name: `select_id`)

  The `SELECT` identifier. This is the sequential number of the `SELECT` within the query. The value can be `NULL` if the row refers to the union result of other rows. In this case, the `table` column shows a value like `<unionM,N>` to indicate that the row refers to the union of the rows with `id` values of *`M`* and *`N`*.

- `select_type` (JSON name: none)

  The type of `SELECT`, which can be any of those shown in the following table. A JSON-formatted `EXPLAIN` exposes the `SELECT` type as a property of a `query_block`, unless it is `SIMPLE` or `PRIMARY`. The JSON names (where applicable) are also shown in the table.

`select_type` Value	JSON Name	Meaning
`SIMPLE`	None	Simple `SELECT` (not using `UNION` or subqueries)
`PRIMARY`	None	Outermost `SELECT`
`UNION`	None	Second or later `SELECT` statement in a `UNION`

`select_type` Value	JSON Name	Meaning
`DEPENDENT UNION`	`dependent` (`true`)	Second or later `SELECT` statement in a `UNION`, dependent on outer query
`UNION RESULT`	`union_result`	Result of a `UNION`.
`SUBQUERY`	None	First `SELECT` in subquery
`DEPENDENT SUBQUERY`	`dependent` (`true`)	First `SELECT` in subquery, dependent on outer query
`DERIVED`	None	Derived table
`DEPENDENT DERIVED`	`dependent` (`true`)	Derived table dependent on another table
`MATERIALIZED`	`materialized_from_subquery`	Materialized subquery
`UNCACHEABLE SUBQUERY`	`cacheable` (`false`)	A subquery for which the result cannot be cached and must be re-evaluated for each row of the outer query
`UNCACHEABLE UNION`	`cacheable` (`false`)	The second or later select in a `UNION` that belongs to an uncacheable subquery (see `UNCACHEABLE SUBQUERY`)

`DEPENDENT` typically signifies the use of a correlated subquery. See Section 13.2.11.7, “Correlated Subqueries”.

`DEPENDENT SUBQUERY` evaluation differs from `UNCACHEABLE SUBQUERY` evaluation. For `DEPENDENT SUBQUERY`, the subquery is re-evaluated only once for each set of different values of the variables from its outer context. For `UNCACHEABLE SUBQUERY`, the subquery is re-evaluated for each row of the outer context.

When you specify `FORMAT=JSON` with `EXPLAIN`, the output has no single property directly equivalent to `select_type`; the `query_block` property corresponds to a given `SELECT`. Properties equivalent to most of the `SELECT` subquery types just shown are available (an example being `materialized_from_subquery` for `MATERIALIZED`), and are displayed when appropriate. There are no JSON equivalents for `SIMPLE` or `PRIMARY`.

The `select_type` value for non-`SELECT` statements displays the statement type for affected tables. For example, `select_type` is `DELETE` for `DELETE` statements.

- `table` (JSON name: `table_name`)

  The name of the table to which the row of output refers. This can also be one of the following values:

  - `<union`*`M`*`,`*`N`*`>`: The row refers to the union of the rows with `id` values of *`M`* and *`N`*.
  - `<derived`*`N`*`>`: The row refers to the derived table result for the row with an `id` value of *`N`*. A derived table may result, for example, from a subquery in the `FROM` clause.
  - `<subquery`*`N`*`>`: The row refers to the result of a materialized subquery for the row with an `id` value of *`N`*. See Section 8.2.2.2, “Optimizing Subqueries with Materialization”.

- `partitions` (JSON name: `partitions`)

  The partitions from which records would be matched by the query. The value is `NULL` for nonpartitioned tables. See Section 23.3.5, “Obtaining Information About Partitions”.

- `type` (JSON name: `access_type`)

The join type. For descriptions of the different types, see `EXPLAIN` Join Types.

- `possible_keys` (JSON name: `possible_keys`)

  The `possible_keys` column indicates the indexes from which MySQL can choose to find the rows in this table. Note that this column is totally independent of the order of the tables as displayed in the output from `EXPLAIN`. That means that some of the keys in `possible_keys` might not be usable in practice with the generated table order.

  If this column is `NULL` (or undefined in JSON-formatted output), there are no relevant indexes. In this case, you may be able to improve the performance of your query by examining the `WHERE` clause to check whether it refers to some column or columns that would be suitable for indexing. If so, create an appropriate index and check the query with `EXPLAIN` again. See Section 13.1.9, “ALTER TABLE Syntax”.

  To see what indexes a table has, use `SHOW INDEX FROM *tbl_name*`.

- `key` (JSON name: `key`)

  The `key` column indicates the key (index) that MySQL actually decided to use. If MySQL decides to use one of the `possible_keys` indexes to look up rows, that index is listed as the key value.

  It is possible that `key` will name an index that is not present in the `possible_keys` value. This can happen if none of the `possible_keys` indexes are suitable for looking up rows, but all the columns selected by the query are columns of some other index. That is, the named index covers the selected columns, so although it is not used to determine which rows to retrieve, an index scan is more efficient than a data row scan.

  For `InnoDB`, a secondary index might cover the selected columns even if the query also selects the primary key because `InnoDB` stores the primary key value with each secondary index. If `key` is `NULL`, MySQL found no index to use for executing the query more efficiently.

  To force MySQL to use or ignore an index listed in the `possible_keys` column, use `FORCE INDEX`, `USE INDEX`, or `IGNORE INDEX` in your query. See Section 8.9.4, “Index Hints”.

  For `MyISAM` tables, running `ANALYZE TABLE` helps the optimizer choose better indexes. For `MyISAM` tables, `myisamchk --analyze` does the same. See Section 13.7.3.1, “ANALYZE TABLE Syntax”, and Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

- `key_len` (JSON name: `key_length`)

  The `key_len` column indicates the length of the key that MySQL decided to use. The value of `key_len` enables you to determine how many parts of a multiple-part key MySQL actually uses. If the `key` column says `NULL`, the `len_len` column also says `NULL`.

  Due to the key storage format, the key length is one greater for a column that can be `NULL` than for a `NOT NULL` column.

- `ref` (JSON name: `ref`)

  The `ref` column shows which columns or constants are compared to the index named in the `key` column to select rows from the table.

  If the value is `func`, the value used is the result of some function. To see which function, use `SHOW WARNINGS` following `EXPLAIN` to see the extended `EXPLAIN` output. The function might actually be an operator such as an arithmetic operator.

- `rows` (JSON name: `rows`)

  The `rows` column indicates the number of rows MySQL believes it must examine to execute the query.

  For `InnoDB` tables, this number is an estimate, and may not always be exact.

- `filtered` (JSON name: `filtered`)

  The `filtered` column indicates an estimated percentage of table rows that will be filtered by the table condition. The maximum value is 100, which means no filtering of rows occurred. Values decreasing from 100 indicate increasing amounts of filtering. `rows` shows the estimated number of rows examined and `rows` × `filtered` shows the number of rows that will be joined with the following table. For example, if `rows` is 1000 and `filtered` is 50.00 (50%), the number of rows to be joined with the following table is 1000 × 50% = 500.

- `Extra` (JSON name: none)

  This column contains additional information about how MySQL resolves the query. For descriptions of the different values, see `EXPLAIN` Extra Information.

  There is no single JSON property corresponding to the `Extra` column; however, values that can occur in this column are exposed as JSON properties, or as the text of the `message` property.

EXPLAIN Join Types

The `type` column of `EXPLAIN` output describes how tables are joined. In JSON-formatted output, these are found as values of the `access_type` property. The following list describes the join types, ordered from the best type to the worst:

- `system`

  The table has only one row (= system table). This is a special case of the `const` join type.

- `const`

  The table has at most one matching row, which is read at the start of the query. Because there is only one row, values from the column in this row can be regarded as constants by the rest of the optimizer. `const` tables are very fast because they are read only once.

  `const` is used when you compare all parts of a `PRIMARY KEY` or `UNIQUE` index to constant values. In the following queries, *`tbl_name`* can be used as a `const` table:

```
SELECT * FROM tbl_name WHERE primary_key=1;

SELECT * FROM tbl_name
  WHERE primary_key_part1=1 AND primary_key_part2=2;
```

- `eq_ref`

  One row is read from this table for each combination of rows from the previous tables. Other than the `system` and `const` types, this is the best possible join type. It is used when all parts of an index are used by the join and the index is a `PRIMARY KEY` or `UNIQUE NOT NULL` index.

  `eq_ref` can be used for indexed columns that are compared using the `=` operator. The comparison value can be a constant or an expression that uses columns from tables that are read before this table. In the following examples, MySQL can use an `eq_ref` join to process *`ref_table`*:

```
SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1=other_table.column
  AND ref_table.key_column_part2=1;
```

- `ref`

  All rows with matching index values are read from this table for each combination of rows from the previous tables. `ref` is used if the join uses only a leftmost prefix of the key or if the key is not a `PRIMARY KEY` or `UNIQUE` index (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this is a good join type.

  `ref` can be used for indexed columns that are compared using the `=` or `<=>` operator. In the following examples, MySQL can use a `ref` join to process *`ref_table`*:

  ```
  SELECT * FROM ref_table WHERE key_column=expr;

  SELECT * FROM ref_table,other_table
    WHERE ref_table.key_column=other_table.column;

  SELECT * FROM ref_table,other_table
    WHERE ref_table.key_column_part1=other_table.column
    AND ref_table.key_column_part2=1;
  ```

- `fulltext`

  The join is performed using a `FULLTEXT` index.

- `ref_or_null`

  This join type is like `ref`, but with the addition that MySQL does an extra search for rows that contain `NULL` values. This join type optimization is used most often in resolving subqueries. In the following examples, MySQL can use a `ref_or_null` join to process *`ref_table`*:

  ```
  SELECT * FROM ref_table
    WHERE key_column=expr OR key_column IS NULL;
  ```

  See Section 8.2.1.15, “IS NULL Optimization”.

- `index_merge`

  This join type indicates that the Index Merge optimization is used. In this case, the `key` column in the output row contains a list of indexes used, and `key_len` contains a list of the longest key parts for the indexes used. For more information, see Section 8.2.1.3, “Index Merge Optimization”.

- `unique_subquery`

  This type replaces `eq_ref` for some `IN` subqueries of the following form:

  ```
  value IN (SELECT primary_key FROM single_table WHERE some_expr)
  ```

  `unique_subquery` is just an index lookup function that replaces the subquery completely for better efficiency.

- `index_subquery`

This join type is similar to `unique_subquery`. It replaces `IN` subqueries, but it works for nonunique indexes in subqueries of the following form:

```
value IN (SELECT key_column FROM single_table WHERE some_expr)
```

- `range`

  Only rows that are in a given range are retrieved, using an index to select the rows. The `key` column in the output row indicates which index is used. The `key_len` contains the longest key part that was used. The `ref` column is `NULL` for this type.

  `range` can be used when a key column is compared to a constant using any of the `=`, `<>`, `>`, `>=`, `<`, `<=`, `IS NULL`, `<=>`, `BETWEEN`, `LIKE`, or `IN()` operators:

  ```
  SELECT * FROM tbl_name
    WHERE key_column = 10;

  SELECT * FROM tbl_name
    WHERE key_column BETWEEN 10 and 20;

  SELECT * FROM tbl_name
    WHERE key_column IN (10,20,30);

  SELECT * FROM tbl_name
    WHERE key_part1 = 10 AND key_part2 IN (10,20,30);
  ```

- `index`

  The `index` join type is the same as `ALL`, except that the index tree is scanned. This occurs two ways:

  - If the index is a covering index for the queries and can be used to satisfy all data required from the table, only the index tree is scanned. In this case, the `Extra` column says `Using index`. An index-only scan usually is faster than `ALL` because the size of the index usually is smaller than the table data.
  - A full table scan is performed using reads from the index to look up data rows in index order. `Uses index` does not appear in the `Extra` column.

  MySQL can use this join type when the query uses only columns that are part of a single index.

- `ALL`

  A full table scan is done for each combination of rows from the previous tables. This is normally not good if the table is the first table not marked `const`, and usually *very* bad in all other cases. Normally, you can avoid `ALL` by adding indexes that enable row retrieval from the table based on constant values or column values from earlier tables.

EXPLAIN Extra Information

The `Extra` column of `EXPLAIN` output contains additional information about how MySQL resolves the query. The following list explains the values that can appear in this column. Each item also indicates for JSON-formatted output which property displays the `Extra` value. For some of these, there is a specific property. The others display as the text of the `message` property.

If you want to make your queries as fast as possible, look out for `Extra` column values of `Using filesort` and `Using temporary`, or, in JSON-formatted `EXPLAIN` output, for `using_filesort` and `using_temporary_table` properties equal to `true`.

- `Child of 'table' pushed join@1` (JSON: `message` text)

  This table is referenced as the child of *`table`* in a join that can be pushed down to the NDB kernel. Applies only in NDB Cluster, when pushed-down joins are enabled. See the description of the `ndb_join_pushdown` server system variable for more information and examples.

- `const row not found` (JSON property: `const_row_not_found`)

  For a query such as `SELECT ... FROM tbl_name`, the table was empty.

- `Deleting all rows` (JSON property: `message`)

  For `DELETE`, some storage engines (such as `MyISAM`) support a handler method that removes all table rows in a simple and fast way. This `Extra` value is displayed if the engine uses this optimization.

- `Distinct` (JSON property: `distinct`)

  MySQL is looking for distinct values, so it stops searching for more rows for the current row combination after it has found the first matching row.

- `FirstMatch(tbl_name)` (JSON property: `first_match`)

  The semijoin FirstMatch join shortcutting strategy is used for *`tbl_name`*.

- `Full scan on NULL key` (JSON property: `message`)

  This occurs for subquery optimization as a fallback strategy when the optimizer cannot use an index-lookup access method.

- `Impossible HAVING` (JSON property: `message`)

  The `HAVING` clause is always false and cannot select any rows.

- `Impossible WHERE` (JSON property: `message`)

  The `WHERE` clause is always false and cannot select any rows.

- `Impossible WHERE noticed after reading const tables` (JSON property: `message`)

  MySQL has read all `const` (and `system`) tables and notice that the `WHERE` clause is always false.

- `LooseScan(m..n)` (JSON property: `message`)

  The semijoin LooseScan strategy is used. *`m`* and *`n`* are key part numbers.

- `No matching min/max row` (JSON property: `message`)

  No row satisfies the condition for a query such as `SELECT MIN(...) FROM ... WHERE condition`.

- `no matching row in const table` (JSON property: `message`)

  For a query with a join, there was an empty table or a table with no rows satisfying a unique index condition.

- `No matching rows after partition pruning` (JSON property: `message`)

  For `DELETE` or `UPDATE`, the optimizer found nothing to delete or update after partition pruning. It is similar in meaning to `Impossible WHERE` for `SELECT` statements.

- `No tables used` (JSON property: `message`)

The query has no `FROM` clause, or has a `FROM DUAL` clause.

For `INSERT` or `REPLACE` statements, `EXPLAIN` displays this value when there is no `SELECT` part. For example, it appears for `EXPLAIN INSERT INTO t VALUES(10)` because that is equivalent to `EXPLAIN INSERT INTO t SELECT 10 FROM DUAL`.

- `Not exists` (JSON property: `message`)

  MySQL was able to do a `LEFT JOIN` optimization on the query and does not examine more rows in this table for the previous row combination after it finds one row that matches the `LEFT JOIN` criteria. Here is an example of the type of query that can be optimized this way:

  ```
  SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
    WHERE t2.id IS NULL;
  ```

  Assume that `t2.id` is defined as `NOT NULL`. In this case, MySQL scans `t1` and looks up the rows in `t2` using the values of `t1.id`. If MySQL finds a matching row in `t2`, it knows that `t2.id` can never be `NULL`, and does not scan through the rest of the rows in `t2` that have the same `id` value. In other words, for each row in `t1`, MySQL needs to do only a single lookup in `t2`, regardless of how many rows actually match in `t2`.

  In MySQL 8.0.17 and later, this can also indicate that a `WHERE` condition of the form `NOT IN (*subquery*)` or `NOT EXISTS (*subquery*)` has been transformed internally into an antijoin. This removes the subquery and brings its tables into the plan for the topmost query, providing improved cost planning. By merging semijoins and antijoins, the optimizer can reorder tables in the execution plan more freely, in some casesresulting in a faster plan.

  You can see when an antijoin transformation is performed for a given query by checking the `Message` column from `SHOW WARNINGS` following execution of `EXPLAIN`, or in the output of `EXPLAIN FORMAT=TREE`.

  Note

  An antijoin is the complement of a semijoin `*table_a* JOIN *table_b* ON *condition*`. The antijoin returns all rows from `*table_a*` for which there is *no* row in `*table_b*` which matches `*condition*`.

- `Plan isn't ready yet` (JSON property: none)

  This value occurs with `EXPLAIN FOR CONNECTION` when the optimizer has not finished creating the execution plan for the statement executing in the named connection. If execution plan output comprises multiple lines, any or all of them could have this `Extra` value, depending on the progress of the optimizer in determining the full execution plan.

- `Range checked for each record (index map: *N*)` (JSON property: `message`)

  MySQL found no good index to use, but found that some of indexes might be used after column values from preceding tables are known. For each row combination in the preceding tables, MySQL checks whether it is possible to use a `range` or `index_merge` access method to retrieve rows. This is not very fast, but is faster than performing a join with no index at all. The applicability criteria are as described in Section 8.2.1.2, “Range Optimization”, and Section 8.2.1.3, “Index Merge Optimization”, with the exception that all column values for the preceding table are known and considered to be constants.

Indexes are numbered beginning with 1, in the same order as shown by `SHOW INDEX` for the table. The index map value *`N`* is a bitmask value that indicates which indexes are candidates. For example, a value of `0x19` (binary 11001) means that indexes 1, 4, and 5 will be considered.

- `Recursive` (JSON property: `recursive`)

  This indicates that the row applies to the recursive `SELECT` part of a recursive common table expression. See Section 13.2.13, “WITH Syntax (Common Table Expressions)”.

- `Rematerialize` (JSON property: `rematerialize`)

  `Rematerialize (X,...)` is displayed in the `EXPLAIN` row for table `T`, where `X` is any lateral derived table whose rematerialization is triggered when a new row of `T` is read. For example:

```
SELECT
  ...
FROM
  t,
  LATERAL (derived table that refers to t) AS dt
...
```

  The content of the derived table is rematerialized to bring it up to date each time a new row of `t` is processed by the top query.

- `Scanned N databases` (JSON property: `message`)

  This indicates how many directory scans the server performs when processing a query for `INFORMATION_SCHEMA` tables, as described in Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”. The value of *`N`* can be 0, 1, or `all`.

- `Select tables optimized away` (JSON property: `message`)

  The optimizer determined 1) that at most one row should be returned, and 2) that to produce this row, a deterministic set of rows must be read. When the rows to be read can be read during the optimization phase (for example, by reading index rows), there is no need to read any tables during query execution.

  The first condition is fulfilled when the query is implicitly grouped (contains an aggregate function but no `GROUP BY` clause). The second condition is fulfilled when one row lookup is performed per index used. The number of indexes read determines the number of rows to read.

  Consider the following implicitly grouped query:

```
SELECT MIN(c1), MIN(c2) FROM t1;
```

  Suppose that `MIN(c1)` can be retrieved by reading one index row and `MIN(c2)` can be retrieved by reading one row from a different index. That is, for each column `c1` and `c2`, there exists an index where the column is the first column of the index. In this case, one row is returned, produced by reading two deterministic rows.

  This `Extra` value does not occur if the rows to read are not deterministic. Consider this query:

```
SELECT MIN(c2) FROM t1 WHERE c1 <= 10;
```

  Suppose that `(c1, c2)` is a covering index. Using this index, all rows with `c1 <= 10` must be scanned to find the minimum `c2` value. By contrast, consider this query:

```
SELECT MIN(c2) FROM t1 WHERE c1 = 10;
```

In this case, the first index row with `c1 = 10` contains the minimum `c2` value. Only one row must be read to produce the returned row.

For storage engines that maintain an exact row count per table (such as `MyISAM`, but not `InnoDB`), this `Extra` value can occur for `COUNT(*)` queries for which the `WHERE` clause is missing or always true and there is no `GROUP BY` clause. (This is an instance of an implicitly grouped query where the storage engine influences whether a deterministic number of rows can be read.)

- `Skip_open_table`, `Open_frm_only`, `Open_full_table` (JSON property: `message`)

  These values indicate file-opening optimizations that apply to queries for `INFORMATION_SCHEMA` tables.

  - `Skip_open_table`: Table files do not need to be opened. The information is already available from the data dictionary.
  - `Open_frm_only`: Only the data dictionary need be read for table information.
  - `Open_full_table`: Unoptimized information lookup. Table information must be read from the data dictionary and by reading table files.

- `Start temporary`, `End temporary` (JSON property: `message`)

  This indicates temporary table use for the semijoin Duplicate Weedout strategy.

- `unique row not found` (JSON property: `message`)

  For a query such as `SELECT ... FROM tbl_name`, no rows satisfy the condition for a `UNIQUE` index or `PRIMARY KEY` on the table.

- `Using filesort` (JSON property: `using_filesort`)

  MySQL must do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the `WHERE` clause. The keys then are sorted and the rows are retrieved in sorted order. See Section 8.2.1.16, “ORDER BY Optimization”.

- `Using index` (JSON property: `using_index`)

  The column information is retrieved from the table using only information in the index tree without having to do an additional seek to read the actual row. This strategy can be used when the query uses only columns that are part of a single index.

  For `InnoDB` tables that have a user-defined clustered index, that index can be used even when `Using index` is absent from the `Extra` column. This is the case if `type` is `index` and `key` is `PRIMARY`.

- `Using index condition` (JSON property: `using_index_condition`)

  Tables are read by accessing index tuples and testing them first to determine whether to read full table rows. In this way, index information is used to defer (“push down”) reading full table rows unless it is necessary. See Section 8.2.1.6, “Index Condition Pushdown Optimization”.

- `Using index for group-by` (JSON property: `using_index_for_group_by`)

  Similar to the `Using index` table access method, `Using index for group-by` indicates that MySQL found an index that can be used to retrieve all columns of a `GROUP BY` or `DISTINCT` query

without any extra disk access to the actual table. Additionally, the index is used in the most efficient way so that for each group, only a few index entries are read. For details, see Section 8.2.1.17, “GROUP BY Optimization”.

- `Using index for skip scan` (JSON property: `using_index_for_skip_scan`)

  Indicates that the Skip Scan access method is used. See Skip Scan Range Access Method.

- `Using join buffer (Block Nested Loop)`, `Using join buffer (Batched Key Access)` (JSON property: `using_join_buffer`)

  Tables from earlier joins are read in portions into the join buffer, and then their rows are used from the buffer to perform the join with the current table. `(Block Nested Loop)` indicates use of the Block Nested-Loop algorithm and `(Batched Key Access)` indicates use of the Batched Key Access algorithm. That is, the keys from the table on the preceding line of the `EXPLAIN` output will be buffered, and the matching rows will be fetched in batches from the table represented by the line in which `Using join buffer` appears.

  In JSON-formatted output, the value of `using_join_buffer` is always either one of `Block Nested Loop` or `Batched Key Access`.

- `Using MRR` (JSON property: `message`)

  Tables are read using the Multi-Range Read optimization strategy. See Section 8.2.1.11, “Multi-Range Read Optimization”.

- `Using sort_union(...)`, `Using union(...)`, `Using intersect(...)` (JSON property: `message`)

  These indicate the particular algorithm showing how index scans are merged for the `index_merge` join type. See Section 8.2.1.3, “Index Merge Optimization”.

- `Using temporary` (JSON property: `using_temporary_table`)

  To resolve the query, MySQL needs to create a temporary table to hold the result. This typically happens if the query contains `GROUP BY` and `ORDER BY` clauses that list columns differently.

- `Using where` (JSON property: `attached_condition`)

  A `WHERE` clause is used to restrict which rows to match against the next table or send to the client. Unless you specifically intend to fetch or examine all rows from the table, you may have something wrong in your query if the `Extra` value is not `Using where` and the table join type is `ALL` or `index`.

  `Using where` has no direct counterpart in JSON-formatted output; the `attached_condition` property contains any `WHERE` condition used.

- `Using where with pushed condition` (JSON property: `message`)

  This item applies to `NDB` tables *only*. It means that NDB Cluster is using the Condition Pushdown optimization to improve the efficiency of a direct comparison between a nonindexed column and a constant. In such cases, the condition is “pushed down” to the cluster's data nodes and is evaluated on all data nodes simultaneously. This eliminates the need to send nonmatching rows over the network, and can speed up such queries by a factor of 5 to 10 times over cases where Condition Pushdown could be but is not used. For more information, see Section 8.2.1.5, “Engine Condition Pushdown Optimization”.

- `Zero limit` (JSON property: `message`)

  The query had a `LIMIT 0` clause and cannot select any rows.

EXPLAIN Output Interpretation

You can get a good indication of how good a join is by taking the product of the values in the `rows` column of the `EXPLAIN` output. This should tell you roughly how many rows MySQL must examine to execute the query. If you restrict queries with the `max_join_size` system variable, this row product also is used to determine which multiple-table `SELECT` statements to execute and which to abort. See Section 5.1.1, “Configuring the Server”.

The following example shows how a multiple-table join can be optimized progressively based on the information provided by `EXPLAIN`.

Suppose that you have the `SELECT` statement shown here and that you plan to examine it using `EXPLAIN`:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
               tt.ProjectReference, tt.EstimatedShipDate,
               tt.ActualShipDate, tt.ClientID,
               tt.ServiceCodes, tt.RepetitiveID,
               tt.CurrentProcess, tt.CurrentDPPerson,
               tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
               et_1.COUNTRY, do.CUSTNAME
        FROM tt, et, et AS et_1, do
        WHERE tt.SubmitTime IS NULL
          AND tt.ActualPC = et.EMPLOYID
          AND tt.AssignedPC = et_1.EMPLOYID
          AND tt.ClientID = do.CUSTNMBR;
```

For this example, make the following assumptions:

- The columns being compared have been declared as follows.

Table	Column	Data Type
`tt`	`ActualPC`	`CHAR(10)`
`tt`	`AssignedPC`	`CHAR(10)`
`tt`	`ClientID`	`CHAR(10)`
`et`	`EMPLOYID`	`CHAR(15)`
`do`	`CUSTNMBR`	`CHAR(15)`

- The tables have the following indexes.

Table	Index
`tt`	`ActualPC`
`tt`	`AssignedPC`
`tt`	`ClientID`
`et`	`EMPLOYID` (primary key)
`do`	`CUSTNMBR` (primary key)

- The `tt.ActualPC` values are not evenly distributed.

Initially, before any optimizations have been performed, the `EXPLAIN` statement produces the following information:

```
table type possible_keys key  key_len ref  rows  Extra
et    ALL  PRIMARY       NULL NULL    NULL 74
```

```
do    ALL  PRIMARY       NULL NULL    NULL 2135
et_1  ALL  PRIMARY       NULL NULL    NULL 74
tt    ALL  AssignedPC,   NULL NULL    NULL 3872
           ClientID,
           ActualPC
      Range checked for each record (index map: 0x23)
```

Because `type` is `ALL` for each table, this output indicates that MySQL is generating a Cartesian product of all the tables; that is, every combination of rows. This takes quite a long time, because the product of the number of rows in each table must be examined. For the case at hand, this product is 74 × 2135 × 74 × 3872 = 45,268,558,720 rows. If the tables were bigger, you can only imagine how long it would take.

One problem here is that MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. `tt.ActualPC` is declared as `CHAR(10)` and `et.EMPLOYID` is `CHAR(15)`, so there is a length mismatch.

To fix this disparity between column lengths, use `ALTER TABLE` to lengthen `ActualPC` from 10 characters to 15 characters:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Now `tt.ActualPC` and `et.EMPLOYID` are both `VARCHAR(15)`. Executing the `EXPLAIN` statement again produces this result:

```
table type   possible_keys key     key_len ref         rows    Extra
tt    ALL    AssignedPC,   NULL    NULL    NULL        3872    Using
             ClientID,                                        where
             ActualPC
do    ALL    PRIMARY       NULL    NULL    NULL        2135
      Range checked for each record (index map: 0x1)
et_1  ALL    PRIMARY       NULL    NULL    NULL        74
      Range checked for each record (index map: 0x1)
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC 1
```

This is not perfect, but is much better: The product of the `rows` values is less by a factor of 74. This version executes in a couple of seconds.

A second alteration can be made to eliminate the column length mismatches for the `tt.AssignedPC = et_1.EMPLOYID` and `tt.ClientID = do.CUSTNMBR` comparisons:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
                      MODIFY ClientID   VARCHAR(15);
```

After that modification, `EXPLAIN` produces the output shown here:

```
table type   possible_keys key      key_len ref           rows Extra
et    ALL    PRIMARY       NULL     NULL    NULL          74
tt    ref    AssignedPC,   ActualPC 15      et.EMPLOYID   52   Using
             ClientID,                                        where
             ActualPC
et_1  eq_ref PRIMARY       PRIMARY  15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY  15      tt.ClientID   1
```

At this point, the query is optimized almost as well as possible. The remaining problem is that, by default, MySQL assumes that values in the `tt.ActualPC` column are evenly distributed, and that is not the case for the `tt` table. Fortunately, it is easy to tell MySQL to analyze the key distribution:

```
mysql> ANALYZE TABLE tt;
```

With the additional index information, the join is perfect and `EXPLAIN` produces this result:

```
table type   possible_keys key     key_len ref           rows Extra
tt    ALL    AssignedPC    NULL    NULL    NULL          3872 Using
             ClientID,                                       where
             ActualPC
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC   1
et_1  eq_ref PRIMARY       PRIMARY 15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY 15      tt.ClientID   1
```

The `rows` column in the output from `EXPLAIN` is an educated guess from the MySQL join optimizer. Check whether the numbers are even close to the truth by comparing the `rows` product with the actual number of rows that the query returns. If the numbers are quite different, you might get better performance by using `STRAIGHT_JOIN` in your `SELECT` statement and trying to list the tables in a different order in the `FROM` clause. (However, `STRAIGHT_JOIN` may prevent indexes from being used because it disables semijoin transformations. See Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”.)

It is possible in some cases to execute statements that modify data when `EXPLAIN SELECT` is used with a subquery; for more information, see Section 13.2.11.8, “Derived Tables”.

8.8.3 Extended EXPLAIN Output Format

The `EXPLAIN` statement produces extra (“extended”) information that is not part of `EXPLAIN` output but can be viewed by issuing a `SHOW WARNINGS` statement following `EXPLAIN`. As of MySQL 8.0.12, extended information is available for `SELECT`, `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` statements. Prior to 8.0.12, extended information is available only for `SELECT` statements.

The `Message` value in `SHOW WARNINGS` output displays how the optimizer qualifies table and column names in the `SELECT` statement, what the `SELECT` looks like after the application of rewriting and optimization rules, and possibly other notes about the optimization process.

The extended information displayable with a `SHOW WARNINGS` statement following `EXPLAIN` is produced only for `SELECT` statements. `SHOW WARNINGS` displays an empty result for other explainable statements (`DELETE`, `INSERT`, `REPLACE`, and `UPDATE`).

Here is an example of extended `EXPLAIN` output:

```
mysql> EXPLAIN
       SELECT t1.a, t1.a IN (SELECT t2.a FROM t2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
         type: index
possible_keys: NULL
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 4
     filtered: 100.00
        Extra: Using index
*************************** 2. row ***************************
           id: 2
  select_type: SUBQUERY
        table: t2
         type: index
```

```
possible_keys: a
          key: a
      key_len: 5
          ref: NULL
         rows: 3
     filtered: 100.00
        Extra: Using index
2 rows in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select `test`.`t1`.`a` AS `a`,
         <in_optimizer>(`test`.`t1`.`a`,`test`.`t1`.`a` in
         ( <materialize> (/* select#2 */ select `test`.`t2`.`a`
         from `test`.`t2` where 1 having 1 ),
         <primary_index_lookup>(`test`.`t1`.`a` in
         <temporary table> on <auto_key>
         where ((`test`.`t1`.`a` = `materialized-subquery`.`a`))))) AS `t1.a
         IN (SELECT t2.a FROM t2)` from `test`.`t1`
1 row in set (0.00 sec)
```

Because the statement displayed by `SHOW WARNINGS` may contain special markers to provide information about query rewriting or optimizer actions, the statement is not necessarily valid SQL and is not intended to be executed. The output may also include rows with `Message` values that provide additional non-SQL explanatory notes about actions taken by the optimizer.

The following list describes special markers that can appear in the extended output displayed by `SHOW WARNINGS`:

- `<auto_key>`

  An automatically generated key for a temporary table.

- `<cache>(expr)`

  The expression (such as a scalar subquery) is executed once and the resulting value is saved in memory for later use. For results consisting of multiple values, a temporary table may be created and you will see `<temporary table>` instead.

- `<exists>(query fragment)`

  The subquery predicate is converted to an `EXISTS` predicate and the subquery is transformed so that it can be used together with the `EXISTS` predicate.

- `<in_optimizer>(query fragment)`

  This is an internal optimizer object with no user significance.

- `<index_lookup>(query fragment)`

  The query fragment is processed using an index lookup to find qualifying rows.

- `<if>(condition, expr1, expr2)`

  If the condition is true, evaluate to *`expr1`*, otherwise *`expr2`*.

- `<is_not_null_test>(expr)`

  A test to verify that the expression does not evaluate to `NULL`.

- `<materialize>(query fragment)`

  Subquery materialization is used.

- `` `materialized-subquery`.col_name ``

  A reference to the column *`col_name`* in an internal temporary table materialized to hold the result from evaluating a subquery.

- `<primary_index_lookup>(query fragment)`

  The query fragment is processed using a primary key lookup to find qualifying rows.

- `<ref_null_helper>(expr)`

  This is an internal optimizer object with no user significance.

- `/* select#N */ select_stmt`

  The `SELECT` is associated with the row in non-extended `EXPLAIN` output that has an `id` value of *`N`*.

- *`outer_tables`* `semi join (`*`inner_tables`*`)`

  A semijoin operation. *`inner_tables`* shows the tables that were not pulled out. See Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”.

- `<temporary table>`

  This represents an internal temporary table created to cache an intermediate result.

When some tables are of `const` or `system` type, expressions involving columns from these tables are evaluated early by the optimizer and are not part of the displayed statement. However, with `FORMAT=JSON`, some `const` table accesses are displayed as a `ref` access that uses a const value.

8.8.4 Obtaining Execution Plan Information for a Named Connection

To obtain the execution plan for an explainable statement executing in a named connection, use this statement:

```
EXPLAIN [options] FOR CONNECTION connection_id;
```

`EXPLAIN FOR CONNECTION` returns the `EXPLAIN` information that is currently being used to execute a query in a given connection. Because of changes to data (and supporting statistics) it may produce a different result from running `EXPLAIN` on the equivalent query text. This difference in behavior can be useful in diagnosing more transient performance problems. For example, if you are running a statement in one session that is taking a long time to complete, using `EXPLAIN FOR CONNECTION` in another session may yield useful information about the cause of the delay.

`connection_id` is the connection identifier, as obtained from the `INFORMATION_SCHEMA PROCESSLIST` table or the `SHOW PROCESSLIST` statement. If you have the `PROCESS` privilege, you can specify the identifier for any connection. Otherwise, you can specify the identifier only for your own connections. In all cases, you must have sufficient privileges to explain the query on the specified connection.

If the named connection is not executing a statement, the result is empty. Otherwise, `EXPLAIN FOR CONNECTION` applies only if the statement being executed in the named connection is explainable. This includes `SELECT`, `DELETE`, `INSERT`, `REPLACE`, and `UPDATE`. (However, `EXPLAIN FOR CONNECTION` does not work for prepared statements, even prepared statements of those types.)

If the named connection is executing an explainable statement, the output is what you would obtain by using `EXPLAIN` on the statement itself.

If the named connection is executing a statement that is not explainable, an error occurs. For example, you cannot name the connection identifier for your current session because `EXPLAIN` is not explainable:

```
mysql> SELECT CONNECTION_ID();
+-----------------+
| CONNECTION_ID() |
+-----------------+
|             373 |
+-----------------+
1 row in set (0.00 sec)

mysql> EXPLAIN FOR CONNECTION 373;
ERROR 1889 (HY000): EXPLAIN FOR CONNECTION command is supported
only for SELECT/UPDATE/INSERT/DELETE/REPLACE
```

The `Com_explain_other` status variable indicates the number of `EXPLAIN FOR CONNECTION` statements executed.

8.8.5 Estimating Query Performance

In most cases, you can estimate query performance by counting disk seeks. For small tables, you can usually find a row in one disk seek (because the index is probably cached). For bigger tables, you can estimate that, using B-tree indexes, you need this many seeks to find a row: `log(row_count) / log(index_block_length / 3 * 2 / (index_length + data_pointer_length)) + 1`.

In MySQL, an index block is usually 1,024 bytes and the data pointer is usually four bytes. For a 500,000-row table with a key value length of three bytes (the size of `MEDIUMINT`), the formula indicates `log(500,000)/log(1024/3*2/(3+4)) + 1` = `4` seeks.

This index would require storage of about 500,000 * 7 * 3/2 = 5.2MB (assuming a typical index buffer fill ratio of 2/3), so you probably have much of the index in memory and so need only one or two calls to read data to find the row.

For writes, however, you need four seek requests to find where to place a new index value and normally two seeks to update the index and write the row.

The preceding discussion does not mean that your application performance slowly degenerates by log *N*. As long as everything is cached by the OS or the MySQL server, things become only marginally slower as the table gets bigger. After the data gets too big to be cached, things start to go much slower until your applications are bound only by disk seeks (which increase by log *N*). To avoid this, increase the key cache size as the data grows. For `MyISAM` tables, the key cache size is controlled by the `key_buffer_size` system variable. See Section 5.1.1, “Configuring the Server”.

8.9 Controlling the Query Optimizer

MySQL provides optimizer control through system variables that affect how query plans are evaluated, switchable optimizations, optimizer and index hints, and the optimizer cost model.

The server also maintains statistics about column values, although the optimizer does not yet use this information.

8.9.1 Controlling Query Plan Evaluation

The task of the query optimizer is to find an optimal plan for executing an SQL query. Because the difference in performance between “good” and “bad” plans can be orders of magnitude (that is, seconds

versus hours or even days), most query optimizers, including that of MySQL, perform a more or less exhaustive search for an optimal plan among all possible query evaluation plans. For join queries, the number of possible plans investigated by the MySQL optimizer grows exponentially with the number of tables referenced in a query. For small numbers of tables (typically less than 7 to 10) this is not a problem. However, when larger queries are submitted, the time spent in query optimization may easily become the major bottleneck in the server's performance.

A more flexible method for query optimization enables the user to control how exhaustive the optimizer is in its search for an optimal query evaluation plan. The general idea is that the fewer plans that are investigated by the optimizer, the less time it spends in compiling a query. On the other hand, because the optimizer skips some plans, it may miss finding an optimal plan.

The behavior of the optimizer with respect to the number of plans it evaluates can be controlled using two system variables:

- The `optimizer_prune_level` variable tells the optimizer to skip certain plans based on estimates of the number of rows accessed for each table. Our experience shows that this kind of “educated guess” rarely misses optimal plans, and may dramatically reduce query compilation times. That is why this option is on (`optimizer_prune_level=1`) by default. However, if you believe that the optimizer missed a better query plan, this option can be switched off (`optimizer_prune_level=0`) with the risk that query compilation may take much longer. Note that, even with the use of this heuristic, the optimizer still explores a roughly exponential number of plans.

- The `optimizer_search_depth` variable tells how far into the “future” of each incomplete plan the optimizer should look to evaluate whether it should be expanded further. Smaller values of `optimizer_search_depth` may result in orders of magnitude smaller query compilation times. For example, queries with 12, 13, or more tables may easily require hours and even days to compile if `optimizer_search_depth` is close to the number of tables in the query. At the same time, if compiled with `optimizer_search_depth` equal to 3 or 4, the optimizer may compile in less than a minute for the same query. If you are unsure of what a reasonable value is for `optimizer_search_depth`, this variable can be set to 0 to tell the optimizer to determine the value automatically.

8.9.2 Switchable Optimizations

The `optimizer_switch` system variable enables control over optimizer behavior. Its value is a set of flags, each of which has a value of `on` or `off` to indicate whether the corresponding optimizer behavior is enabled or disabled. This variable has global and session values and can be changed at runtime. The global default can be set at server startup.

To see the current set of optimizer flags, select the variable value:

```
mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,index_merge_intersection=on,
                    engine_condition_pushdown=on,index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,block_nested_loop=on,
                    batched_key_access=off,materialization=on,semijoin=on,
                    loosescan=on,firstmatch=on,duplicateweedout=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on,condition_fanout_filter=on,
                    derived_merge=on,use_invisible_indexes=off,skip_scan=on,
                    hash_join=on
```

To change the value of `optimizer_switch`, assign a value consisting of a comma-separated list of one or more commands:

```
SET [GLOBAL|SESSION] optimizer_switch='command[,command]...';
```

Each *`command`* value should have one of the forms shown in the following table.

Command Syntax	Meaning
`default`	Reset every optimization to its default value
`opt_name``=default`	Set the named optimization to its default value
`opt_name``=off`	Disable the named optimization
`opt_name``=on`	Enable the named optimization

The order of the commands in the value does not matter, although the `default` command is executed first if present. Setting an *`opt_name`* flag to `default` sets it to whichever of `on` or `off` is its default value. Specifying any given *`opt_name`* more than once in the value is not permitted and causes an error. Any errors in the value cause the assignment to fail with an error, leaving the value of `optimizer_switch` unchanged.

The following list describes the permissible *`opt_name`* flag names, grouped by optimization strategy:

- Batched Key Access Flags
  - `batched_key_access` (default `off`)

    Controls use of BKA join algorithm.

  For `batched_key_access` to have any effect when set to `on`, the `mrr` flag must also be `on`. Currently, the cost estimation for MRR is too pessimistic. Hence, it is also necessary for `mrr_cost_based` to be `off` for BKA to be used.

  For more information, see Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”.
- Block Nested-Loop Flags
  - `block_nested_loop` (default `on`)

    Controls use of BNL join algorithm.

  For more information, see Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”.
- Condition Filtering Flags
  - `condition_fanout_filter` (default `on`)

    Controls use of condition filtering.

  For more information, see Section 8.2.1.13, “Condition Filtering”.
- Derived Table Merging Flags
  - `derived_merge` (default `on`)

    Controls merging of derived tables and views into outer query block.

  The `derived_merge` flag controls whether the optimizer attempts to merge derived tables, view references, and common table expressions into the outer query block, assuming that no other rule

prevents merging; for example, an `ALGORITHM` directive for a view takes precedence over the `derived_merge` setting. By default, the flag is `on` to enable merging.

For more information, see Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”.

- Engine Condition Pushdown Flags
  - `engine_condition_pushdown` (default `on`)

    Controls engine condition pushdown.

  For more information, see Section 8.2.1.5, “Engine Condition Pushdown Optimization”.
- Hash Join Flags
  - `hash_join` (default `on`)

    Controls hash joins (MySQL 8.0.18 and later).

  For more information, see Section 8.2.1.4, “Hash Join Optimization”.
- Index Condition Pushdown Flags
  - `index_condition_pushdown` (default `on`)

    Controls index condition pushdown.

  For more information, see Section 8.2.1.6, “Index Condition Pushdown Optimization”.
- Index Extensions Flags
  - `use_index_extensions` (default `on`)

    Controls use of index extensions.

  For more information, see Section 8.3.10, “Use of Index Extensions”.
- Index Merge Flags
  - `index_merge` (default `on`)

    Controls all Index Merge optimizations.
  - `index_merge_intersection` (default `on`)

    Controls the Index Merge Intersection Access optimization.
  - `index_merge_sort_union` (default `on`)

    Controls the Index Merge Sort-Union Access optimization.
  - `index_merge_union` (default `on`)

    Controls the Index Merge Union Access optimization.

  For more information, see Section 8.2.1.3, “Index Merge Optimization”.
- Index Visibility Flags

- `use_invisible_indexes` (default `off`)

  Controls use of invisible indexes.

For more information, see Section 8.3.12, “Invisible Indexes”.

- Multi-Range Read Flags
  - `mrr` (default `on`)

    Controls the Multi-Range Read strategy.

  - `mrr_cost_based` (default `on`)

    Controls use of cost-based MRR if `mrr=on`.

  For more information, see Section 8.2.1.11, “Multi-Range Read Optimization”.

- Skip Scan Flags
  - `skip_scan` (default `on`)

    Controls use of Skip Scan access method.

  For more information, see Skip Scan Range Access Method.

- Semijoin Flags
  - `semijoin` (default `on`)

    Controls all semijoin strategies.

    In MySQL 8.0.17 and later, this also applies to the antijoin optimization.

  - `duplicateweedout` (default `on`)

    Controls the semijoin Duplicate Weedout strategy.

  - `firstmatch` (default `on`)

    Controls the semijoin FirstMatch strategy.

  - `loosescan` (default `on`)

    Controls the semijoin LooseScan strategy (not to be confused with Loose Index Scan for `GROUP BY`).

  The `semijoin`, `firstmatch`, `loosescan`, and `duplicateweedout` flags enable control over semijoin strategies. The `semijoin` flag controls whether semijoins are used. If it is set to `on`, the `firstmatch` and `loosescan` flags enable finer control over the permitted semijoin strategies.

  If the `duplicateweedout` semijoin strategy is disabled, it is not used unless all other applicable strategies are also disabled.

  If `semijoin` and `materialization` are both `on`, semijoins also use materialization where applicable. These flags are `on` by default.

  For more information, see Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”.

- Subquery Materialization Flags

  - `materialization` (default `on`)

    Controls materialization (including semijoin materialization).

  - `subquery_materialization_cost_based` (default `on`)

    Use cost-based materialization choice.

  The `materialization` flag controls whether subquery materialization is used. If `semijoin` and `materialization` are both `on`, semijoins also use materialization where applicable. These flags are `on` by default.

  The `subquery_materialization_cost_based` flag enables control over the choice between subquery materialization and `IN`-to-`EXISTS` subquery transformation. If the flag is `on` (the default), the optimizer performs a cost-based choice between subquery materialization and `IN`-to-`EXISTS` subquery transformation if either method could be used. If the flag is `off`, the optimizer chooses subquery materialization over `IN`-to-`EXISTS` subquery transformation.

  For more information, see Section 8.2.2, “Optimizing Subqueries, Derived Tables, View References, and Common Table Expressions”.

When you assign a value to `optimizer_switch`, flags that are not mentioned keep their current values. This makes it possible to enable or disable specific optimizer behaviors in a single statement without affecting other behaviors. The statement does not depend on what other optimizer flags exist and what their values are. Suppose that all Index Merge optimizations are enabled:

```
mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on,
                    index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,
                    block_nested_loop=on,batched_key_access=off,
                    materialization=on,semijoin=on,loosescan=on,
                    firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on,
                    condition_fanout_filter=on
```

If the server is using the Index Merge Union or Index Merge Sort-Union access methods for certain queries and you want to check whether the optimizer will perform better without them, set the variable value like this:

```
mysql> SET optimizer_switch='index_merge_union=off,index_merge_sort_union=off';

mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=off,
                    index_merge_sort_union=off,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on,
                    index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,
                    block_nested_loop=on,batched_key_access=off,
                    materialization=on,semijoin=on,loosescan=on,
```

```
                    firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on,
                    condition_fanout_filter=on
```

8.9.3 Optimizer Hints

One means of control over optimizer strategies is to set the `optimizer_switch` system variable (see Section 8.9.2, “Switchable Optimizations”). Changes to this variable affect execution of all subsequent queries; to affect one query differently from another, it's necessary to change `optimizer_switch` before each one.

Another way to control the optimizer is by using optimizer hints, which can be specified within individual statements. Because optimizer hints apply on a per-statement basis, they provide finer control over statement execution plans than can be achieved using `optimizer_switch`. For example, you can enable an optimization for one table in a statement and disable the optimization for a different table. Hints within a statement take precedence over `optimizer_switch` flags.

Examples:

```
SELECT /*+ NO_RANGE_OPTIMIZATION(t3 PRIMARY, f2_idx) */ f1
  FROM t3 WHERE f1 > 30 AND f1 < 33;
SELECT /*+ BKA(t1) NO_BKA(t2) */ * FROM t1 INNER JOIN t2 WHERE ...;
SELECT /*+ NO_ICP(t1, t2) */ * FROM t1 INNER JOIN t2 WHERE ...;
SELECT /*+ SEMIJOIN(FIRSTMATCH, LOOSESCAN) */ * FROM t1 ...;
EXPLAIN SELECT /*+ NO_ICP(t1) */ * FROM t1 WHERE ...;
SELECT /*+ MERGE(dt) */ * FROM (SELECT * FROM t1) AS dt;
INSERT /*+ SET_VAR(foreign_key_checks=OFF) */ INTO t2 VALUES(2);
```

Optimizer hints, described here, differ from index hints, described in Section 8.9.4, “Index Hints”. Optimizer and index hints may be used separately or together.

- Optimizer Hint Overview
- Optimizer Hint Syntax
- Join-Order Optimizer Hints
- Table-Level Optimizer Hints
- Index-Level Optimizer Hints
- Subquery Optimizer Hints
- Statement Execution Time Optimizer Hints
- Variable-Setting Hint Syntax
- Resource Group Hint Syntax
- Optimizer Hints for Naming Query Blocks

Optimizer Hint Overview

Optimizer hints apply at different scope levels:

- Global: The hint affects the entire statement

- Query block: The hint affects a particular query block within a statement
- Table-level: The hint affects a particular table within a query block
- Index-level: The hint affects a particular index within a table

The following table summarizes the available optimizer hints, the optimizer strategies they affect, and the scope or scopes at which they apply. More details are given later.

Table 8.2 Optimizer Hints Available

Hint Name	Description	Applicable Scopes
`BKA`, `NO_BKA`	Affects Batched Key Access join processing	Query block, table
`BNL`, `NO_BNL`	Affects Block Nested-Loop join processing	Query block, table
`HASH_JOIN`, `NO_HASH_JOIN`	Affects Hash Join optimization	Table, index
`INDEX_MERGE`, `NO_INDEX_MERGE`	Affects Index Merge optimization	Table, index
`JOIN_FIXED_ORDER`	Use table order specified in `FROM` clause for join order	Query block
`JOIN_ORDER`	Use table order specified in hint for join order	Query block
`JOIN_PREFIX`	Use table order specified in hint for first tables of join order	Query block
`JOIN_SUFFIX`	Use table order specified in hint for last tables of join order	Query block
`MAX_EXECUTION_TIME`	Limits statement execution time	Global
`MERGE`, `NO_MERGE`	Affects derived table/view merging into outer query block	Table
`MRR`, `NO_MRR`	Affects Multi-Range Read optimization	Table, index
`NO_ICP`	Affects Index Condition Pushdown optimization	Table, index
`NO_RANGE_OPTIMIZATION`	Affects range optimization	Table, index
`QB_NAME`	Assigns name to query block	Query block
`RESOURCE_GROUP`	Set resource group during statement execution	Global
`SEMIJOIN`, `NO_SEMIJOIN`	Affects semijoin strategies; beginning with MySQL 8.0.17, this also applies to antijoins	Query block
`SKIP_SCAN`, `NO_SKIP_SCAN`	Affects Skip Scan optimization	Table, index
`SET_VAR`	Set variable during statement execution	Global
`SUBQUERY`	Affects materialization, `IN`-to-`EXISTS` subquery stratgies	Query block

Disabling an optimization prevents the optimizer from using it. Enabling an optimization means the optimizer is free to use the strategy if it applies to statement execution, not that the optimizer necessarily will use it.

Optimizer Hint Syntax

MySQL supports comments in SQL statements as described in Section 9.6, “Comment Syntax”. Optimizer hints must be specified within `/*+ ... */` comments. That is, optimizer hints use a variant of `/* ... */` C-style comment syntax, with a `+` character following the `/*` comment opening sequence. Examples:

```
/*+ BKA(t1) */
/*+ BNL(t1, t2) */
/*+ NO_RANGE_OPTIMIZATION(t4 PRIMARY) */
/*+ QB_NAME(qb2) */
```

Whitespace is permitted after the `+` character.

The parser recognizes optimizer hint comments after the initial keyword of `SELECT`, `UPDATE`, `INSERT`, `REPLACE`, and `DELETE` statements. Hints are permitted in these contexts:

- At the beginning of query and data change statements:

  ```
  SELECT /*+ ... */ ...
  INSERT /*+ ... */ ...
  REPLACE /*+ ... */ ...
  UPDATE /*+ ... */ ...
  DELETE /*+ ... */ ...
  ```

- At the beginning of query blocks:

  ```
  (SELECT /*+ ... */ ... )
  (SELECT ... ) UNION (SELECT /*+ ... */ ... )
  (SELECT /*+ ... */ ... ) UNION (SELECT /*+ ... */ ... )
  UPDATE ... WHERE x IN (SELECT /*+ ... */ ...)
  INSERT ... SELECT /*+ ... */ ...
  ```

- In hintable statements prefaced by `EXPLAIN`. For example:

  ```
  EXPLAIN SELECT /*+ ... */ ...
  EXPLAIN UPDATE ... WHERE x IN (SELECT /*+ ... */ ...)
  ```

  The implication is that you can use `EXPLAIN` to see how optimizer hints affect execution plans. Use `SHOW WARNINGS` immediately after `EXPLAIN` to see how hints are used. The extended `EXPLAIN` output displayed by a following `SHOW WARNINGS` indicates which hints were used. Ignored hints are not displayed.

A hint comment may contain multiple hints, but a query block cannot contain multiple hint comments. This is valid:

```
SELECT /*+ BNL(t1) BKA(t2) */ ...
```

But this is invalid:

```
SELECT /*+ BNL(t1) */ /* BKA(t2) */ ...
```

When a hint comment contains multiple hints, the possibility of duplicates and conflicts exists. The following general guidelines apply. For specific hint types, additional rules may apply, as indicated in the hint descriptions.

- Duplicate hints: For a hint such as `/*+ MRR(idx1) MRR(idx1) */`, MySQL uses the first hint and issues a warning about the duplicate hint.

- Conflicting hints: For a hint such as `/*+ MRR(idx1) NO_MRR(idx1) */`, MySQL uses the first hint and issues a warning about the second conflicting hint.

Query block names are identifiers and follow the usual rules about what names are valid and how to quote them (see Section 9.2, “Schema Object Names”).

Hint names, query block names, and strategy names are not case sensitive. References to table and index names follow the usual identifier case sensitivity rules (see Section 9.2.3, “Identifier Case Sensitivity”).

Join-Order Optimizer Hints

Join-order hints affect the order in which the optimizer joins tables.

Syntax of the `JOIN_FIXED_ORDER` hint:

```
hint_name([@query_block_name])
```

Syntax of other join-order hints:

```
hint_name([@query_block_name] tbl_name [, tbl_name] ...)
hint_name(tbl_name[@query_block_name] [, tbl_name[@query_block_name]] ...)
```

The syntax refers to these terms:

- *`hint_name`*: These hint names are permitted:
  - `JOIN_FIXED_ORDER`: Force the optimizer to join tables using the order in which they appear in the `FROM` clause. This is the same as specifying `SELECT STRAIGHT_JOIN`.
  - `JOIN_ORDER`: Instruct the optimizer to join tables using the specified table order. The hint applies to the named tables. The optimizer may place tables that are not named anywhere in the join order, including between specified tables.
  - `JOIN_PREFIX`: Instruct the optimizer to join tables using the specified table order for the first tables of the join execution plan. The hint applies to the named tables. The optimizer places all other tables after the named tables.
  - `JOIN_SUFFIX`: Instruct the optimizer to join tables using the specified table order for the last tables of the join execution plan. The hint applies to the named tables. The optimizer places all other tables before the named tables.
- *`tbl_name`*: The name of a table used in the statement. A hint that names tables applies to all tables that it names. The `JOIN_FIXED_ORDER` hint names no tables and applies to all tables in the `FROM` clause of the query block in which it occurs.

  If a table has an alias, hints must refer to the alias, not the table name.

  Table names in hints cannot be qualified with schema names.
- *`query_block_name`*: The query block to which the hint applies. If the hint includes no leading *`@query_block_name`*, the hint applies to the query block in which it occurs. For *`tbl_name@query_block_name`* syntax, the hint applies to the named table in the named query block. To assign a name to a query block, see Optimizer Hints for Naming Query Blocks.

Example:

```
SELECT
/*+ JOIN_PREFIX(t2, t5@subq2, t4@subq1)
    JOIN_ORDER(t4@subq1, t3)
    JOIN_SUFFIX(t1) */
COUNT(*) FROM t1 JOIN t2 JOIN t3
           WHERE t1.f1 IN (SELECT /*+ QB_NAME(subq1) */ f1 FROM t4)
             AND t2.f1 IN (SELECT /*+ QB_NAME(subq2) */ f1 FROM t5);
```

Hints control the behavior of semijoin tables that are merged to the outer query block. If subqueries `subq1` and `subq2` are converted to semijoins, tables `t4@subq1` and `t5@subq2` are merged to the outer query block. In this case, the hint specified in the outer query block controls the behavior of `t4@subq1`, `t5@subq2` tables.

The optimizer resolves join-order hints according to these principles:

- Multiple hint instances

  Only one `JOIN_PREFIX` and `JOIN_SUFFIX` hint of each type are applied. Any later hints of the same type are ignored with a warning. `JOIN_ORDER` can be specified several times.

  Examples:

  ```
  /*+ JOIN_PREFIX(t1) JOIN_PREFIX(t2) */
  ```

  The second `JOIN_PREFIX` hint is ignored with a warning.

  ```
  /*+ JOIN_PREFIX(t1) JOIN_SUFFIX(t2) */
  ```

  Both hints are applicable. No warning occurs.

  ```
  /*+ JOIN_ORDER(t1, t2) JOIN_ORDER(t2, t3) */
  ```

  Both hints are applicable. No warning occurs.

- Conflicting hints

  In some cases hints can conflict, such as when `JOIN_ORDER` and `JOIN_PREFIX` have table orders that are impossible to apply at the same time:

  ```
  SELECT /*+ JOIN_ORDER(t1, t2) JOIN_PREFIX(t2, t1) */ ... FROM t1, t2;
  ```

  In this case, the first specified hint is applied and subsequent conflicting hints are ignored with no warning. A valid hint that is impossible to apply is silently ignored with no warning.

- Ignored hints

  A hint is ignored if a table specified in the hint has a circular dependency.

  Example:

  ```
  /*+ JOIN_ORDER(t1, t2) JOIN_PREFIX(t2, t1) */
  ```

  The `JOIN_ORDER` hint sets table `t2` dependent on `t1`. The `JOIN_PREFIX` hint is ignored because table `t1` cannot be dependent on `t2`. Ignored hints are not displayed in extended `EXPLAIN` output.

- Interaction with `const` tables

The MySQL optimizer places `const` tables first in the join order, and the position of a `const` table cannot be affected by hints. References to `const` tables in join-order hints are ignored, although the hint is still applicable. For example, these are equivalent:

```
JOIN_ORDER(t1, const_tbl, t2)
JOIN_ORDER(t1, t2)
```

Accepted hints shown in extended `EXPLAIN` output include `const` tables as they were specified.

- Interaction with types of join operations

  MySQL supports several type of joins: `LEFT`, `RIGHT`, `INNER`, `CROSS`, `STRAIGHT_JOIN`. A hint that conflicts with the specified type of join is ignored with no warning.

  Example:

```
SELECT /*+ JOIN_PREFIX(t1, t2) */FROM t2 LEFT JOIN t1;
```

  Here a conflict occurs between the requested join order in the hint and the order required by the `LEFT JOIN`. The hint is ignored with no warning.

Table-Level Optimizer Hints

Table-level hints affect:

- Use of the Block Nested-Loop (BNL) and Batched Key Access (BKA) join-processing algorithms (see Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”).
- Whether derived tables, view references, or common table expressions should be merged into the outer query block, or materialized using an internal temporary table.

These hint types apply to specific tables, or all tables in a query block.

Syntax of table-level hints:

```
hint_name([@query_block_name] [tbl_name [, tbl_name] ...])
hint_name([tbl_name@query_block_name [, tbl_name@query_block_name] ...])
```

The syntax refers to these terms:

- *`hint_name`*: These hint names are permitted:
  - `BKA`, `NO_BKA`: Enable or disable BKA for the specified tables.
  - `BNL`, `NO_BNL`: Enable or disable BNL for the specified tables.
  - `HASH_JOIN`, `NO_HASH_JOIN`: Enable or disable use of a hash join for the specified tables.
  - `MERGE`, `NO_MERGE`: Enable merging for the specified tables, view references or common table expressions; or disable merging and use materialization instead.

Note

To use a BNL or BKA hint to enable join buffering for any inner table of an outer join, join buffering must be enabled for all inner tables of the outer join.

- *tbl_name*: The name of a table used in the statement. The hint applies to all tables that it names. If the hint names no tables, it applies to all tables of the query block in which it occurs.

  If a table has an alias, hints must refer to the alias, not the table name.

  Table names in hints cannot be qualified with schema names.

- *query_block_name*: The query block to which the hint applies. If the hint includes no leading *@query_block_name*, the hint applies to the query block in which it occurs. For *tbl_name@query_block_name* syntax, the hint applies to the named table in the named query block. To assign a name to a query block, see Optimizer Hints for Naming Query Blocks.

Examples:

```
SELECT /*+ NO_BKA(t1, t2) */ t1.* FROM t1 INNER JOIN t2 INNER JOIN t3;
SELECT /*+ NO_BNL() BKA(t1) */ t1.* FROM t1 INNER JOIN t2 INNER JOIN t3;
SELECT /*+ NO_MERGE(dt) */ * FROM (SELECT * FROM t1) AS dt;
```

A table-level hint applies to tables that receive records from previous tables, not sender tables. Consider this statement:

```
SELECT /*+ BNL(t2) */ FROM t1, t2;
```

If the optimizer chooses to process `t1` first, it applies a Block Nested-Loop join to `t2` by buffering the rows from `t1` before starting to read from `t2`. If the optimizer instead chooses to process `t2` first, the hint has no effect because `t2` is a sender table.

For the `MERGE` and `NO_MERGE` hints, these precedence rules apply:

- A hint takes precedence over any optimizer heuristic that is not a technical constraint. (If providing a hint as a suggestion has no effect, the optimizer has a reason for ignoring it.)
- A hint takes precedence over the `derived_merge` flag of the `optimizer_switch` system variable.
- For view references, an `ALGORITHM={MERGE|TEMPTABLE}` clause in the view definition takes precedence over a hint specified in the query referencing the view.

Index-Level Optimizer Hints

Index-level hints affect which index-processing strategies the optimizer uses for particular tables or indexes. These hint types affect use of Index Condition Pushdown (ICP), Multi-Range Read (MRR), Index Merge, and range optimizations (see Section 8.2.1, “Optimizing SELECT Statements”).

Syntax of index-level hints:

```
hint_name([@query_block_name] tbl_name [index_name [, index_name] ...])
hint_name(tbl_name@query_block_name [index_name [, index_name] ...])
```

The syntax refers to these terms:

- *hint_name*: These hint names are permitted:
  - `INDEX_MERGE`, `NO_INDEX_MERGE`: Enable or disable the Index Merge access method for the specified table or indexes. For information about this access method, see Section 8.2.1.3, “Index Merge Optimization”. These hints apply to all three Index Merge algorithms.

The `INDEX_MERGE` hint forces the optimizer to use Index Merge for the specified table using the specified set of indexes. If no index is specified, the optimizer considers all possible index combinations and selects the least expensive one. The hint may be ignored if the index combination is inapplicable to the given statement.

The `NO_INDEX_MERGE` hint disables Index Merge combinations that involve any of the specified indexes. If the hint specifies no indexes, Index Merge is not permitted for the table.

- `MRR`, `NO_MRR`: Enable or disable MRR for the specified table or indexes. MRR hints apply only to `InnoDB` and `MyISAM` tables. For information about this access method, see Section 8.2.1.11, “Multi-Range Read Optimization”.

- `NO_ICP`: Disable ICP for the specified table or indexes. By default, ICP is a candidate optimization strategy, so there is no hint for enabling it. For information about this access method, see Section 8.2.1.6, “Index Condition Pushdown Optimization”.

- `NO_RANGE_OPTIMIZATION`: Disable index range access for the specified table or indexes. This hint also disables Index Merge and Loose Index Scan for the table or indexes. By default, range access is a candidate optimization strategy, so there is no hint for enabling it.

  This hint may be useful when the number of ranges may be high and range optimization would require many resources.

- `SKIP_SCAN`, `NO_SKIP_SCAN`: Enable or disable the Skip Scan access method for the specified table or indexes. For information about this access method, see Skip Scan Range Access Method. These hints are available as of MySQL 8.0.13.

  The `SKIP_SCAN` hint forces the optimizer to use Skip Scan for the specified table using the specified set of indexes. If no index is specified, the optimizer considers all possible indexes and selects the least expensive one. The hint may be ignored if the index is inapplicable to the given statement.

  The `NO_SKIP_SCAN` hint disables Skip Scan for the specified indexes. If the hint specifies no indexes, Skip Scan is not permitted for the table.

- *`tbl_name`*: The table to which the hint applies.

- *`index_name`*: The name of an index in the named table. The hint applies to all indexes that it names. If the hint names no indexes, it applies to all indexes in the table.

  To refer to a primary key, use the name `PRIMARY`. To see the index names for a table, use `SHOW INDEX`.

- *`query_block_name`*: The query block to which the hint applies. If the hint includes no leading *`@query_block_name`*, the hint applies to the query block in which it occurs. For *`tbl_name@query_block_name`* syntax, the hint applies to the named table in the named query block. To assign a name to a query block, see Optimizer Hints for Naming Query Blocks.

Examples:

```
SELECT /*+ INDEX_MERGE(t1 f3, PRIMARY) */ f2 FROM t1
  WHERE f1 = 'o' AND f2 = f3 AND f3 <= 4;
SELECT /*+ MRR(t1) */ * FROM t1 WHERE f2 <= 3 AND 3 <= f3;
SELECT /*+ NO_RANGE_OPTIMIZATION(t3 PRIMARY, f2_idx) */ f1
  FROM t3 WHERE f1 > 30 AND f1 < 33;
INSERT INTO t3(f1, f2, f3)
  (SELECT /*+ NO_ICP(t2) */ t2.f1, t2.f2, t2.f3 FROM t1,t2
```

```
   WHERE t1.f1=t2.f1 AND t2.f2 BETWEEN t1.f1
   AND t1.f2 AND t2.f2 + 1 >= t1.f1 + 1);
SELECT /*+ SKIP_SCAN(t1 PRIMARY) */ f1, f2
  FROM t1 WHERE f2 > 40;
```

The following examples use the Index Merge hints, but other index-level hints follow the same principles regarding hint ignoring and precedence of optimizer hints in relation to the `optimizer_switch` system variable or index hints.

Assume that table `t1` has columns `a`, `b`, `c`, and `d`; and that indexes named `i_a`, `i_b`, and `i_c` exist on `a`, `b`, and `c`, respectively:

```
SELECT /*+ INDEX_MERGE(t1 i_a, i_b, i_c)*/ * FROM t1
  WHERE a = 1 AND b = 2 AND c = 3 AND d = 4;
```

Index Merge is used for `(i_a, i_b, i_c)` in this case.

```
SELECT /*+ INDEX_MERGE(t1 i_a, i_b, i_c)*/ * FROM t1
  WHERE b = 1 AND c = 2 AND d = 3;
```

Index Merge is used for `(i_b, i_c)` in this case.

```
/*+ INDEX_MERGE(t1 i_a, i_b) NO_INDEX_MERGE(t1 i_b) */
```

`NO_INDEX_MERGE` is ignored because there is a preceding hint for the same table.

```
/*+ NO_INDEX_MERGE(t1 i_a, i_b) INDEX_MERGE(t1 i_b) */
```

`INDEX_MERGE` is ignored because there is a preceding hint for the same table.

For the `INDEX_MERGE` and `NO_INDEX_MERGE` optimizer hints, these precedence rules apply:

- If an optimizer hint is specified and is applicable, it takes precedence over the Index Merge-related flags of the `optimizer_switch` system variable.

  ```
  SET optimizer_switch='index_merge_intersection=off';
  SELECT /*+ INDEX_MERGE(t1 i_b, i_c) */ * FROM t1
  WHERE b = 1 AND c = 2 AND d = 3;
  ```

  The hint takes precedence over `optimizer_switch`. Index Merge is used for `(i_b, i_c)` in this case.

  ```
  SET optimizer_switch='index_merge_intersection=on';
  SELECT /*+ INDEX_MERGE(t1 i_b) */ * FROM t1
  WHERE b = 1 AND c = 2 AND d = 3;
  ```

  The hint specifies only one index, so it is inapplicable, and the `optimizer_switch` flag (`on`) applies. Index Merge is used if the optimizer assesses it to be cost efficient.

  ```
  SET optimizer_switch='index_merge_intersection=off';
  SELECT /*+ INDEX_MERGE(t1 i_b) */ * FROM t1
  WHERE b = 1 AND c = 2 AND d = 3;
  ```

  The hint specifies only one index, so it is inapplicable, and the `optimizer_switch` flag (`off`) applies. Index Merge is not used.

- The `USE INDEX`, `FORCE INDEX`, and `IGNORE INDEX` index hints have higher priority than the `INDEX_MERGE` and `NO_INDEX_MERGE` optimizer hints.

```
/*+ INDEX_MERGE(t1 i_a, i_b, i_c) */ ... IGNORE INDEX i_a
```

  `IGNORE INDEX` takes precedence over `INDEX_MERGE`, so index `i_a` is excluded from the possible ranges for Index Merge.

```
/*+ NO_INDEX_MERGE(t1 i_a, i_b) */ ... FORCE INDEX i_a, i_b
```

  Index Merge is disallowed for `i_a, i_b` because of `FORCE INDEX`, but the optimizer is forced to use either `i_a` or `i_b` for `range` or `ref` access. There are no conflicts; both hints are applicable.

- If an `IGNORE INDEX` hint names multiple indexes, those indexes are unavailable for Index Merge.

- The `FORCE INDEX` and `USE INDEX` hints make only the named indexes to be available for Index Merge.

```
SELECT /*+ INDEX_MERGE(t1 i_a, i_b, i_c) */ a FROM t1
FORCE INDEX (i_a, i_b) WHERE c = 'h' AND a = 2 AND b = 'b';
```

  The Index Merge intersection access algorithm is used for `(i_a, i_b)`. The same is true if `FORCE INDEX` is changed to `USE INDEX`.

Subquery Optimizer Hints

Subquery hints affect whether to use semijoin transformations and which semijoin strategies to permit, and, when semijoins are not used, whether to use subquery materialization or `IN`-to-`EXISTS` transformations. For more information about these optimizations, see Section 8.2.2, “Optimizing Subqueries, Derived Tables, View References, and Common Table Expressions”.

Syntax of hints that affect semijoin strategies:

```
hint_name([@query_block_name] [strategy [, strategy] ...])
```

The syntax refers to these terms:

- *`hint_name`*: These hint names are permitted:

  - `SEMIJOIN`, `NO_SEMIJOIN`: Enable or disable the named semijoin strategies.

- *`strategy`*: A semijoin strategy to be enabled or disabled. These strategy names are permitted: `DUPSWEEDOUT`, `FIRSTMATCH`, `LOOSESCAN`, `MATERIALIZATION`.

  For `SEMIJOIN` hints, if no strategies are named, semijoin is used if possible based on the strategies enabled according to the `optimizer_switch` system variable. If strategies are named but inapplicable for the statement, `DUPSWEEDOUT` is used.

  For `NO_SEMIJOIN` hints, if no strategies are named, semijoin is not used. If strategies are named that rule out all applicable strategies for the statement, `DUPSWEEDOUT` is used.

If one subquery is nested within another and both are merged into a semijoin of an outer query, any specification of semijoin strategies for the innermost query are ignored. `SEMIJOIN` and `NO_SEMIJOIN` hints can still be used to enable or disable semijoin transformations for such nested subqueries.

If DUPSWEEDOUT is disabled, on occasion the optimizer may generate a query plan that is far from optimal. This occurs due to heuristic pruning during greedy search, which can be avoided by setting optimizer_prune_level=0.

Examples:

```
SELECT /*+ NO_SEMIJOIN(@subq1 FIRSTMATCH, LOOSESCAN) */ * FROM t2
  WHERE t2.a IN (SELECT /*+ QB_NAME(subq1) */ a FROM t3);
SELECT /*+ SEMIJOIN(@subq1 MATERIALIZATION, DUPSWEEDOUT) */ * FROM t2
  WHERE t2.a IN (SELECT /*+ QB_NAME(subq1) */ a FROM t3);
```

Syntax of hints that affect whether to use subquery materialization or IN-to-EXISTS transformations:

```
SUBQUERY([@query_block_name] strategy)
```

The hint name is always SUBQUERY.

For SUBQUERY hints, these *strategy* values are permitted: INTOEXISTS, MATERIALIZATION.

Examples:

```
SELECT id, a IN (SELECT /*+ SUBQUERY(MATERIALIZATION) */ a FROM t1) FROM t2;
SELECT * FROM t2 WHERE t2.a IN (SELECT /*+ SUBQUERY(INTOEXISTS) */ a FROM t1);
```

For semijoin and SUBQUERY hints, a leading *@query_block_name* specifies the query block to which the hint applies. If the hint includes no leading *@query_block_name*, the hint applies to the query block in which it occurs. To assign a name to a query block, see Optimizer Hints for Naming Query Blocks.

If a hint comment contains multiple subquery hints, the first is used. If there are other following hints of that type, they produce a warning. Following hints of other types are silently ignored.

Statement Execution Time Optimizer Hints

The MAX_EXECUTION_TIME hint is permitted only for SELECT statements. It places a limit *N* (a timeout value in milliseconds) on how long a statement is permitted to execute before the server terminates it:

```
MAX_EXECUTION_TIME(N)
```

Example with a timeout of 1 second (1000 milliseconds):

```
SELECT /*+ MAX_EXECUTION_TIME(1000) */ * FROM t1 INNER JOIN t2 WHERE ...
```

The MAX_EXECUTION_TIME(*N*) hint sets a statement execution timeout of *N* milliseconds. If this option is absent or *N* is 0, the statement timeout established by the max_execution_time system variable applies.

The MAX_EXECUTION_TIME hint is applicable as follows:

- For statements with multiple SELECT keywords, such as unions or statements with subqueries, MAX_EXECUTION_TIME applies to the entire statement and must appear after the first SELECT.
- It applies to read-only SELECT statements. Statements that are not read only are those that invoke a stored function that modifies data as a side effect.

- It does not apply to SELECT statements in stored programs and is ignored.

Variable-Setting Hint Syntax

The SET_VAR hint sets the session value of a system variable temporarily (for the duration of a single statement). Examples:

```
SELECT /*+ SET_VAR(sort_buffer_size = 16M) */ name FROM people ORDER BY name;
INSERT /*+ SET_VAR(foreign_key_checks=OFF) */ INTO t2 VALUES(2);
SELECT /*+ SET_VAR(optimizer_switch = 'mrr_cost_based=off') */ 1;
```

Syntax of the SET_VAR hint:

```
SET_VAR(var_name = value)
```

var_name names a system variable that has a session value (although not all such variables can be named, as explained later). *value* is the value to assign to the variable; the value must be a scalar.

SET_VAR makes a temporary variable change, as demonstrated by these statements:

```
mysql> SELECT @@unique_checks;
+-----------------+
| @@unique_checks |
+-----------------+
|               1 |
+-----------------+
mysql> SELECT /*+ SET_VAR(unique_checks=OFF) */ @@unique_checks;
+-----------------+
| @@unique_checks |
+-----------------+
|               0 |
+-----------------+
mysql> SELECT @@unique_checks;
+-----------------+
| @@unique_checks |
+-----------------+
|               1 |
+-----------------+
```

With SET_VAR, there is no need to save and restore the variable value. This enables you to replace multiple statements by a single statement. Consider this sequence of statements:

```
SET @saved_val = @@SESSION.var_name;
SET @@SESSION.var_name = value;
SELECT ...
SET @@SESSION.var_name = @saved_val;
```

The sequence can be replaced by this single statement:

```
SELECT /*+ SET_VAR(var_name = value) ...
```

Standalone SET statements permit any of these syntaxes for naming session variables:

```
SET SESSION var_name = value;
SET @@SESSION.var_name = value;
SET @@.var_name = value;
```

Because the SET_VAR hint applies only to session variables, session scope is implicit, and SESSION, @@SESSION., and @@ are neither needed nor permitted. Including explicit session-indicator syntax results in the SET_VAR hint being ignored with a warning.

Not all session variables are permitted for use with SET_VAR. Individual system variable descriptions indicate whether each variable is hintable; see Section 5.1.8, “Server System Variables”. You can also check a system variable at runtime by attempting to use it with SET_VAR. If the variable is not hintable, a warning occurs:

```
mysql> SELECT /*+ SET_VAR(collation_server = 'utf8') */ 1;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 4537
Message: Variable 'collation_server' cannot be set using SET_VAR hint.
```

SET_VAR syntax permits setting only a single variable, but multiple hints can be given to set multiple variables:

```
SELECT /*+ SET_VAR(optimizer_switch = 'mrr_cost_based=off')
           SET_VAR(max_heap_table_size = 1G) */ 1;
```

If several hints with the same variable name appear in the same statement, the first one is applied and the others are ignored with a warning:

```
SELECT /*+ SET_VAR(max_heap_table_size = 1G)
           SET_VAR(max_heap_table_size = 3G) */ 1;
```

In this case, the second hint is ignored with a warning that it is conflicting.

A SET_VAR hint is ignored with a warning if no system variable has the specified name or the variable value is incorrect:

```
SELECT /*+ SET_VAR(max_size = 1G) */ 1;
SELECT /*+ SET_VAR(optimizer_switch = 'mrr_cost_based=yes') */ 1;
```

For the first statement, there is no max_size variable. For the second statement, mrr_cost_flag takes values of on or off, so attempting to set it to yes is incorrect. In each case, the hint is ignored with a warning.

The SET_VAR hint is permitted only at the statement level. If used in a subquery, the hint is ignored with a warning.

Slave servers ignore SET_VAR hints in replicated statements to avoid the potential for security issues.

Resource Group Hint Syntax

The RESOURCE_GROUP optimizer hint is used for resource group management (see Section 8.12.5, “Resource Groups”). This hint assigns the thread that executes a statement to the named resource

group temporarily (for the duration of the statement). It requires the RESOURCE_GROUP_ADMIN or RESOURCE_GROUP_USER privilege.

Examples:

```
SELECT /*+ RESOURCE_GROUP(USR_default) */ name FROM people ORDER BY name;
INSERT /*+ RESOURCE_GROUP(Batch) */ INTO t2 VALUES(2);
```

Syntax of the RESOURCE_GROUP hint:

```
RESOURCE_GROUP(group_name)
```

group_name indicates the resource group to which the thread should be assigned for the duration of statement execution. If the group is nonexistent, a warning occurs and the hint is ignored.

The RESOURCE_GROUP hint must appear after the initial statement keyword (SELECT, INSERT, REPLACE, UPDATE, or DELETE).

An alternative to RESOURCE_GROUP is the SET RESOURCE GROUP statement, which nontemporarily assigns threads to a resource group. See Section 13.7.2.4, “SET RESOURCE GROUP Syntax”.

Optimizer Hints for Naming Query Blocks

Table-level, index-level, and subquery optimizer hints permit specific query blocks to be named as part of their argument syntax. To create these names, use the QB_NAME hint, which assigns a name to the query block in which it occurs:

```
QB_NAME(name)
```

QB_NAME hints can be used to make explicit in a clear way which query blocks other hints apply to. They also permit all non-query block name hints to be specified within a single hint comment for easier understanding of complex statements. Consider the following statement:

```
SELECT ...
  FROM (SELECT ...
  FROM (SELECT ... FROM ...)) ...
```

QB_NAME hints assign names to query blocks in the statement:

```
SELECT /*+ QB_NAME(qb1) */ ...
  FROM (SELECT /*+ QB_NAME(qb2) */ ...
  FROM (SELECT /*+ QB_NAME(qb3) */ ... FROM ...)) ...
```

Then other hints can use those names to refer to the appropriate query blocks:

```
SELECT /*+ QB_NAME(qb1) MRR(@qb1 t1) BKA(@qb2) NO_MRR(@qb3t1 idx1, id2) */ ...
  FROM (SELECT /*+ QB_NAME(qb2) */ ...
  FROM (SELECT /*+ QB_NAME(qb3) */ ... FROM ...)) ...
```

The resulting effect is as follows:

- MRR(@qb1 t1) applies to table t1 in query block qb1.
- BKA(@qb2) applies to query block qb2.
- NO_MRR(@qb3 t1 idx1, id2) applies to indexes idx1 and idx2 in table t1 in query block qb3.

Query block names are identifiers and follow the usual rules about what names are valid and how to quote them (see Section 9.2, “Schema Object Names”). For example, a query block name that contains spaces must be quoted, which can be done using backticks:

```
SELECT /*+ BKA(@`my hint name`) */ ...
  FROM (SELECT /*+ QB_NAME(`my hint name`) */ ...) ...
```

If the `ANSI_QUOTES` SQL mode is enabled, it is also possible to quote query block names within double quotation marks:

```
SELECT /*+ BKA(@"my hint name") */ ...
  FROM (SELECT /*+ QB_NAME("my hint name") */ ...) ...
```

8.9.4 Index Hints

Index hints give the optimizer information about how to choose indexes during query processing. Index hints, described here, differ from optimizer hints, described in Section 8.9.3, “Optimizer Hints”. Index and optimizer hints may be used separately or together.

Index hints apply only to `SELECT` statements. (They are accepted by the parser for `UPDATE` statements but are ignored and have no effect.)

Index hints are specified following a table name. (For the general syntax for specifying tables in a `SELECT` statement, see Section 13.2.10.2, “JOIN Syntax”.) The syntax for referring to an individual table, including index hints, looks like this:

```
tbl_name [[AS] alias] [index_hint_list]

index_hint_list:
    index_hint [index_hint] ...

index_hint:
    USE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
  | {IGNORE|FORCE} {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)

index_list:
    index_name [, index_name] ...
```

The `USE INDEX (index_list)` hint tells MySQL to use only one of the named indexes to find rows in the table. The alternative syntax `IGNORE INDEX (index_list)` tells MySQL to not use some particular index or indexes. These hints are useful if `EXPLAIN` shows that MySQL is using the wrong index from the list of possible indexes.

The `FORCE INDEX` hint acts like `USE INDEX (index_list)`, with the addition that a table scan is assumed to be *very* expensive. In other words, a table scan is used only if there is no way to use one of the named indexes to find rows in the table.

Each hint requires index names, not column names. To refer to a primary key, use the name `PRIMARY`. To see the index names for a table, use the `SHOW INDEX` statement or the `INFORMATION_SCHEMA.STATISTICS` table.

An *`index_name`* value need not be a full index name. It can be an unambiguous prefix of an index name. If a prefix is ambiguous, an error occurs.

Examples:

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
  WHERE col1=1 AND col2=2 AND col3=3;

SELECT * FROM table1 IGNORE INDEX (col3_index)
  WHERE col1=1 AND col2=2 AND col3=3;
```

The syntax for index hints has the following characteristics:

- It is syntactically valid to omit *`index_list`* for `USE INDEX`, which means “use no indexes.” Omitting *`index_list`* for `FORCE INDEX` or `IGNORE INDEX` is a syntax error.
- You can specify the scope of an index hint by adding a `FOR` clause to the hint. This provides more fine-grained control over optimizer selection of an execution plan for various phases of query processing. To affect only the indexes used when MySQL decides how to find rows in the table and how to process joins, use `FOR JOIN`. To influence index usage for sorting or grouping rows, use `FOR ORDER BY` or `FOR GROUP BY`.
- You can specify multiple index hints:

  ```
  SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX FOR ORDER BY (i2) ORDER BY a;
  ```

  It is not an error to name the same index in several hints (even within the same hint):

  ```
  SELECT * FROM t1 USE INDEX (i1) USE INDEX (i1,i1);
  ```

  However, it is an error to mix `USE INDEX` and `FORCE INDEX` for the same table:

  ```
  SELECT * FROM t1 USE INDEX FOR JOIN (i1) FORCE INDEX FOR JOIN (i2);
  ```

If an index hint includes no `FOR` clause, the scope of the hint is to apply to all parts of the statement. For example, this hint:

```
IGNORE INDEX (i1)
```

is equivalent to this combination of hints:

```
IGNORE INDEX FOR JOIN (i1)
IGNORE INDEX FOR ORDER BY (i1)
IGNORE INDEX FOR GROUP BY (i1)
```

In MySQL 5.0, hint scope with no `FOR` clause was to apply only to row retrieval. To cause the server to use this older behavior when no `FOR` clause is present, enable the `old` system variable at server startup. Take care about enabling this variable in a replication setup. With statement-based binary logging, having different modes for the master and slaves might lead to replication errors.

When index hints are processed, they are collected in a single list by type (`USE`, `FORCE`, `IGNORE`) and by scope (`FOR JOIN`, `FOR ORDER BY`, `FOR GROUP BY`). For example:

```
SELECT * FROM t1
  USE INDEX () IGNORE INDEX (i2) USE INDEX (i1) USE INDEX (i2);
```

is equivalent to:

```
SELECT * FROM t1
```

```
  USE INDEX (i1,i2) IGNORE INDEX (i2);
```

The index hints then are applied for each scope in the following order:

1. `{USE|FORCE} INDEX` is applied if present. (If not, the optimizer-determined set of indexes is used.)
2. `IGNORE INDEX` is applied over the result of the previous step. For example, the following two queries are equivalent:

```
SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX (i2) USE INDEX (i2);

SELECT * FROM t1 USE INDEX (i1);
```

For `FULLTEXT` searches, index hints work as follows:

- For natural language mode searches, index hints are silently ignored. For example, `IGNORE INDEX(i1)` is ignored with no warning and the index is still used.
- For boolean mode searches, index hints with `FOR ORDER BY` or `FOR GROUP BY` are silently ignored. Index hints with `FOR JOIN` or no `FOR` modifier are honored. In contrast to how hints apply for non-`FULLTEXT` searches, the hint is used for all phases of query execution (finding rows and retrieval, grouping, and ordering). This is true even if the hint is given for a non-`FULLTEXT` index.

  For example, the following two queries are equivalent:

```
SELECT * FROM t
  USE INDEX (index1)
  IGNORE INDEX (index1) FOR ORDER BY
  IGNORE INDEX (index1) FOR GROUP BY
  WHERE ... IN BOOLEAN MODE ... ;

SELECT * FROM t
  USE INDEX (index1)
  WHERE ... IN BOOLEAN MODE ... ;
```

8.9.5 The Optimizer Cost Model

To generate execution plans, the optimizer uses a cost model that is based on estimates of the cost of various operations that occur during query execution. The optimizer has a set of compiled-in default “cost constants” available to it to make decisions regarding execution plans.

The optimizer also has a database of cost estimates to use during execution plan construction. These estimates are stored in the `server_cost` and `engine_cost` tables in the `mysql` system database and are configurable at any time. The intent of these tables is to make it possible to easily adjust the cost estimates that the optimizer uses when it attempts to arrive at query execution plans.

- Cost Model General Operation
- The Cost Model Database
- Making Changes to the Cost Model Database

Cost Model General Operation

The configurable optimizer cost model works like this:

- The server reads the cost model tables into memory at startup and uses the in-memory values at runtime. Any non-`NULL` cost estimate specified in the tables takes precedence over the corresponding

compiled-in default cost constant. Any `NULL` estimate indicates to the optimizer to use the compiled-in default.

- At runtime, the server may reread the cost tables. This occurs when a storage engine is dynamically loaded or when a `FLUSH OPTIMIZER_COSTS` statement is executed.
- Cost tables enable server administrators to easily adjust cost estimates by changing entries in the tables. It is also easy to revert to a default by setting an entry's cost to `NULL`. The optimizer uses the in-memory cost values, so changes to the tables should be followed by `FLUSH OPTIMIZER_COSTS` to take effect.
- The in-memory cost estimates that are current when a client session begins apply throughout that session until it ends. In particular, if the server rereads the cost tables, any changed estimates apply only to subsequently started sessions. Existing sessions are unaffected.
- Cost tables are specific to a given server instance. The server does not replicate cost table changes to replication slaves.

The Cost Model Database

The optimizer cost model database consists of two tables in the `mysql` system database that contain cost estimate information for operations that occur during query execution:

- `server_cost`: Optimizer cost estimates for general server operations
- `engine_cost`: Optimizer cost estimates for operations specific to particular storage engines

The `server_cost` table contains these columns:

- `cost_name`

  The name of a cost estimate used in the cost model. The name is not case-sensitive. If the server does not recognize the cost name when it reads this table, it writes a warning to the error log.

- `cost_value`

  The cost estimate value. If the value is non-`NULL`, the server uses it as the cost. Otherwise, it uses the default estimate (the compiled-in value). DBAs can change a cost estimate by updating this column. If the server finds that the cost value is invalid (nonpositive) when it reads this table, it writes a warning to the error log.

  To override a default cost estimate (for an entry that specifies `NULL`), set the cost to a non-`NULL` value. To revert to the default, set the value to `NULL`. Then execute `FLUSH OPTIMIZER_COSTS` to tell the server to reread the cost tables.

- `last_update`

  The time of the last row update.

- `comment`

  A descriptive comment associated with the cost estimate. DBAs can use this column to provide information about why a cost estimate row stores a particular value.

- `default_value`

  The default (compiled-in) value for the cost estimate. This column is a read-only generated column that retains its value even if the associated cost estimate is changed. For rows added to the table at runtime, the value of this column is `NULL`.

The primary key for the `server_cost` table is the `cost_name` column, so it is not possible to create multiple entries for any cost estimate.

The server recognizes these `cost_name` values for the `server_cost` table:

- `disk_temptable_create_cost`, `disk_temptable_row_cost`

  The cost estimates for internally created temporary tables stored in a disk-based storage engine (either `InnoDB` or `MyISAM`). Increasing these values increases the cost estimate of using internal temporary tables and makes the optimizer prefer query plans with less use of them. For information about such tables, see Section 8.4.4, “Internal Temporary Table Use in MySQL”.

  The larger default values for these disk parameters compared to the default values for the corresponding memory parameters (`memory_temptable_create_cost`, `memory_temptable_row_cost`) reflects the greater cost of processing disk-based tables.

- `key_compare_cost`

  The cost of comparing record keys. Increasing this value causes a query plan that compares many keys to become more expensive. For example, a query plan that performs a `filesort` becomes relatively more expensive compared to a query plan that avoids sorting by using an index.

- `memory_temptable_create_cost`, `memory_temptable_row_cost`

  The cost estimates for internally created temporary tables stored in the `MEMORY` storage engine. Increasing these values increases the cost estimate of using internal temporary tables and makes the optimizer prefer query plans with less use of them. For information about such tables, see Section 8.4.4, “Internal Temporary Table Use in MySQL”.

  The smaller default values for these memory parameters compared to the default values for the corresponding disk parameters (`disk_temptable_create_cost`, `disk_temptable_row_cost`) reflects the lesser cost of processing memory-based tables.

- `row_evaluate_cost`

  The cost of evaluating record conditions. Increasing this value causes a query plan that examines many rows to become more expensive compared to a query plan that examines fewer rows. For example, a table scan becomes relatively more expensive compared to a range scan that reads fewer rows.

The `engine_cost` table contains these columns:

- `engine_name`

  The name of the storage engine to which this cost estimate applies. The name is not case-sensitive. If the value is `default`, it applies to all storage engines that have no named entry of their own. If the server does not recognize the engine name when it reads this table, it writes a warning to the error log.

- `device_type`

  The device type to which this cost estimate applies. The column is intended for specifying different cost estimates for different storage device types, such as hard disk drives versus solid state drives. Currently, this information is not used and 0 is the only permitted value.

- `cost_name`

  Same as in the `server_cost` table.

- `cost_value`

Same as in the `server_cost` table.

- `last_update`

  Same as in the `server_cost` table.

- `comment`

  Same as in the `server_cost` table.

- `default_value`

  The default (compiled-in) value for the cost estimate. This column is a read-only generated column that retains its value even if the associated cost estimate is changed. For rows added to the table at runtime, the value of this column is `NULL`, with the exception that if the row has the same `cost_name` value as one of the original rows, the `default_value` column will have the same value as that row.

The primary key for the `engine_cost` table is a tuple comprising the (`cost_name`, `engine_name`, `device_type`) columns, so it is not possible to create multiple entries for any combination of values in those columns.

The server recognizes these `cost_name` values for the `engine_cost` table:

- `io_block_read_cost`

  The cost of reading an index or data block from disk. Increasing this value causes a query plan that reads many disk blocks to become more expensive compared to a query plan that reads fewer disk blocks. For example, a table scan becomes relatively more expensive compared to a range scan that reads fewer blocks.

- `memory_block_read_cost`

  Similar to `io_block_read_cost`, but represents the cost of reading an index or data block from an in-memory database buffer.

If the `io_block_read_cost` and `memory_block_read_cost` values differ, the execution plan may change between two runs of the same query. Suppose that the cost for memory access is less than the cost for disk access. In that case, at server startup before data has been read into the buffer pool, you may get a different plan than after the query has been run because then the data will be in memory.

Making Changes to the Cost Model Database

For DBAs who wish to change the cost model parameters from their defaults, try doubling or halving the value and measuring the effect.

Changes to the `io_block_read_cost` and `memory_block_read_cost` parameters are most likely to yield worthwhile results. These parameter values enable cost models for data access methods to take into account the costs of reading information from different sources; that is, the cost of reading information from disk versus reading information already in a memory buffer. For example, all other things being equal, setting `io_block_read_cost` to a value larger than `memory_block_read_cost` causes the optimizer to prefer query plans that read information already held in memory to plans that must read from disk.

This example shows how to change the default value for `io_block_read_cost`:

```
UPDATE mysql.engine_cost
  SET cost_value = 2.0
  WHERE cost_name = 'io_block_read_cost';
FLUSH OPTIMIZER_COSTS;
```

This example shows how to change the value of `io_block_read_cost` only for the `InnoDB` storage engine:

```
INSERT INTO mysql.engine_cost
  VALUES ('InnoDB', 0, 'io_block_read_cost', 3.0,
  CURRENT_TIMESTAMP, 'Using a slower disk for InnoDB');
FLUSH OPTIMIZER_COSTS;
```

8.9.6 Optimizer Statistics

The `column_statistics` data dictionary table stores histogram statistics about column values, for use by the optimizer in constructing query execution plans. To perform histogram management, use the `ANALYZE TABLE` statement; see Section 13.7.3.1, “ANALYZE TABLE Syntax”.

The `column_statistics` table has these characteristics:

- The table contains statistics for columns of all data types except geometry types (spatial data) and `JSON`.
- The table is persistent so that column statistics need not be created each time the server starts.
- The server performs updates to the table; users do not.

The `column_statistics` table is not directly accessible by users because it is part of the data dictionary. Histogram information is available using `INFORMATION_SCHEMA.COLUMN_STATISTICS`, which is implemented as a view on the data dictionary table. `COLUMN_STATISTICS` has these columns:

- `SCHEMA_NAME`, `TABLE_NAME`, `COLUMN_NAME`: The names of the schema, table, and column for which the statistics apply.
- `HISTOGRAM`: A `JSON` value describing the column statistics, stored as a histogram.

Column histograms contain buckets for parts of the range of values stored in the column. Histograms are `JSON` objects to permit flexibility in the representation of column statistics. Here is a sample histogram object:

```
{
  "buckets": [
    [
      1,
      0.3333333333333333
    ],
    [
      2,
      0.6666666666666666
    ],
    [
      3,
      1
    ]
  ],
  "null-values": 0,
  "last-updated": "2017-03-24 13:32:40.000000",
  "sampling-rate": 1,
  "histogram-type": "singleton",
```

```
  "number-of-buckets-specified": 128,
  "data-type": "int",
  "collation-id": 8
}
```

Histogram objects have these keys:

- `buckets`: The histogram buckets. Bucket structure depends on the histogram type.

  For `singleton` histograms, buckets contain two values:

  - Value 1: The value for the bucket. The type depends on the column data type.
  - Value 2: A double representing the cumulative frequency for the value. For example, .25 and .75 indicate that 25% and 75% of the values in the column are less than or equal to the bucket value.

  For `equi-height` histograms, buckets contain four values:

  - Values 1, 2: The lower and upper inclusive values for the bucket. The type depends on the column data type.
  - Value 3: A double representing the cumulative frequency for the value. For example, .25 and .75 indicate that 25% and 75% of the values in the column are less than or equal to the bucket upper value.
  - Value 4: The number of distinct values in the range from the bucket lower value to its upper value.

- `null-values`: A number between 0.0 and 1.0 indicating the fraction of column values that are SQL `NULL` values. If 0, the column contains no `NULL` values.
- `last-updated`: When the histogram was generated, as a UTC value in *`YYYY-MM-DD hh:mm:ss.uuuuuu`* format.
- `sampling-rate`: A number between 0.0 and 1.0 indicating the fraction of data that was sampled to create the histogram. A value of 1 means that all of the data was read (no sampling).
- `histogram-type`: The histogram type:
  - `singleton`: One bucket represents one single value in the column. This histogram type is created when the number of distinct values in the column is less than or equal to the number of buckets specified in the `ANALYZE TABLE` statement that generated the histogram.
  - `equi-height`: One bucket represents a range of values. This histogram type is created when the number of distinct values in the column is greater than the number of buckets specified in the `ANALYZE TABLE` statement that generated the histogram.
- `number-of-buckets-specified`: The number of buckets specified in the `ANALYZE TABLE` statement that generated the histogram.
- `data-type`: The type of data this histogram contains. This is needed when reading and parsing histograms from persistent storage into memory. The value is one of `int`, `uint` (unsigned integer), `double`, `decimal`, `datetime`, or `string` (includes character and binary strings).
- `collation-id`: The collation ID for the histogram data. It is mostly meaningful when the `data-type` value is `string`. Values correspond to `ID` column values in the `INFORMATION_SCHEMA.COLLATIONS` table.

To extract particular values from the histogram objects, you can use `JSON` operations. For example:

```
mysql> SELECT
         TABLE_NAME, COLUMN_NAME,
         HISTOGRAM->>'$."data-type"' AS 'data-type',
         JSON_LENGTH(HISTOGRAM->>'$."buckets"') AS 'bucket-count'
       FROM INFORMATION_SCHEMA.COLUMN_STATISTICS;
+-----------------+-------------+-----------+--------------+
| TABLE_NAME      | COLUMN_NAME | data-type | bucket-count |
+-----------------+-------------+-----------+--------------+
| country         | Population  | int       |          226 |
| city            | Population  | int       |         1024 |
| countrylanguage | Language    | string    |          457 |
+-----------------+-------------+-----------+--------------+
```

The optimizer uses histogram statistics, if applicable, for columns of any data type for which statistics are collected. The optimizer applies histogram statistics to determine row estimates based on the selectivity (filtering effect) of column value comparisons against constant values. Predicates of these forms qualify for histogram use:

```
col_name = constant
col_name <> constant
col_name != constant
col_name > constant
col_name < constant
col_name >= constant
col_name <= constant
col_name IS NULL
col_name IS NOT NULL
col_name BETWEEN constant AND constant
col_name NOT BETWEEN constant AND constant
col_name IN (constant[, constant] ...)
col_name NOT IN (constant[, constant] ...)
```

For example, these statements contain predicates that qualify for histogram use:

```
SELECT * FROM orders WHERE amount BETWEEN 100.0 AND 300.0;
SELECT * FROM tbl WHERE col1 = 15 AND col2 > 100;
```

The requirement for comparison against a constant value includes functions that are constant, such as `ABS()` and `FLOOR()`:

```
SELECT * FROM tbl WHERE col1 < ABS(-34);
```

Histogram statistics are useful primarily for nonindexed columns. Adding an index to a column for which histogram statistics are applicable might also help the optimizer make row estimates. The tradeoffs are:

- An index must be updated when table data is modified.
- A histogram is created or updated only on demand, so it adds no overhead when table data is modified. On the other hand, the statistics become progressively more out of date when table modifications occur, until the next time they are updated.

The optimizer prefers range optimizer row estimates to those obtained from histogram statistics. If the optimizer determines that the range optimizer applies, it does not use histogram statistics.

For columns that are indexed, row estimates can be obtained for equality comparisons using index dives (see Section 8.2.1.2, “Range Optimization”). In this case, histogram statistics are not necessarily useful because index dives can yield better estimates.

In some cases, use of histogram statistics may not improve query execution (for example, if the statistics are out of date). To check whether this is the case, use `ANALYZE TABLE` to regenerate the histogram statistics, then run the query again.

Alternatively, to disable histogram statistics, use `ANALYZE TABLE` to drop them. A different method of disabling histogram statistics is to turn off the `condition_fanout_filter` flag of the `optimizer_switch` system variable (although this may disable other optimizations as well):

```
SET optimizer_switch='condition_fanout_filter=off';
```

If histogram statistics are used, the resulting effect is visible using `EXPLAIN`. Consider the following query, where no index is available for column `col1`:

```
SELECT * FROM t1 WHERE col1 < 24;
```

If histogram statistics indicate that 57% of the rows in `t1` satisfy the `col1 < 24` predicate, filtering can occur even in the absence of an index, and `EXPLAIN` shows 57.00 in the `filtered` column.

8.10 Buffering and Caching

MySQL uses several strategies that cache information in memory buffers to increase performance.

8.10.1 InnoDB Buffer Pool Optimization

`InnoDB` maintains a storage area called the buffer pool for caching data and indexes in memory. Knowing how the `InnoDB` buffer pool works, and taking advantage of it to keep frequently accessed data in memory, is an important aspect of MySQL tuning.

For an explanation of the inner workings of the `InnoDB` buffer pool, an overview of its LRU replacement algorithm, and general configuration information, see Section 15.5.1, “Buffer Pool”.

For additional `InnoDB` buffer pool configuration and tuning information, see these sections:

- Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
- Section 15.8.3.5, “Configuring Buffer Pool Flushing”
- Section 15.8.3.3, “Making the Buffer Pool Scan Resistant”
- Section 15.8.3.2, “Configuring Multiple Buffer Pool Instances”
- Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”
- Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”

8.10.2 The MyISAM Key Cache

To minimize disk I/O, the `MyISAM` storage engine exploits a strategy that is used by many database management systems. It employs a cache mechanism to keep the most frequently accessed table blocks in memory:

- For index blocks, a special structure called the *key cache* (or *key buffer*) is maintained. The structure contains a number of block buffers where the most-used index blocks are placed.

- For data blocks, MySQL uses no special cache. Instead it relies on the native operating system file system cache.

This section first describes the basic operation of the `MyISAM` key cache. Then it discusses features that improve key cache performance and that enable you to better control cache operation:

- Multiple sessions can access the cache concurrently.
- You can set up multiple key caches and assign table indexes to specific caches.

To control the size of the key cache, use the `key_buffer_size` system variable. If this variable is set equal to zero, no key cache is used. The key cache also is not used if the `key_buffer_size` value is too small to allocate the minimal number of block buffers (8).

When the key cache is not operational, index files are accessed using only the native file system buffering provided by the operating system. (In other words, table index blocks are accessed using the same strategy as that employed for table data blocks.)

An index block is a contiguous unit of access to the `MyISAM` index files. Usually the size of an index block is equal to the size of nodes of the index B-tree. (Indexes are represented on disk using a B-tree data structure. Nodes at the bottom of the tree are leaf nodes. Nodes above the leaf nodes are nonleaf nodes.)

All block buffers in a key cache structure are the same size. This size can be equal to, greater than, or less than the size of a table index block. Usually one these two values is a multiple of the other.

When data from any table index block must be accessed, the server first checks whether it is available in some block buffer of the key cache. If it is, the server accesses data in the key cache rather than on disk. That is, it reads from the cache or writes into it rather than reading from or writing to disk. Otherwise, the server chooses a cache block buffer containing a different table index block (or blocks) and replaces the data there by a copy of required table index block. As soon as the new index block is in the cache, the index data can be accessed.

If it happens that a block selected for replacement has been modified, the block is considered “dirty.” In this case, prior to being replaced, its contents are flushed to the table index from which it came.

Usually the server follows an *LRU (Least Recently Used)* strategy: When choosing a block for replacement, it selects the least recently used index block. To make this choice easier, the key cache module maintains all used blocks in a special list (*LRU chain*) ordered by time of use. When a block is accessed, it is the most recently used and is placed at the end of the list. When blocks need to be replaced, blocks at the beginning of the list are the least recently used and become the first candidates for eviction.

The `InnoDB` storage engine also uses an LRU algorithm, to manage its buffer pool. See Section 15.5.1, “Buffer Pool”.

8.10.2.1 Shared Key Cache Access

Threads can access key cache buffers simultaneously, subject to the following conditions:

- A buffer that is not being updated can be accessed by multiple sessions.
- A buffer that is being updated causes sessions that need to use it to wait until the update is complete.
- Multiple sessions can initiate requests that result in cache block replacements, as long as they do not interfere with each other (that is, as long as they need different index blocks, and thus cause different cache blocks to be replaced).

Shared access to the key cache enables the server to improve throughput significantly.

8.10.2.2 Multiple Key Caches

Note

As of MySQL 8.0, the compound-part structured-variable syntax discussed here for referring to multiple `MyISAM` key caches is deprecated.

Shared access to the key cache improves performance but does not eliminate contention among sessions entirely. They still compete for control structures that manage access to the key cache buffers. To reduce key cache access contention further, MySQL also provides multiple key caches. This feature enables you to assign different table indexes to different key caches.

Where there are multiple key caches, the server must know which cache to use when processing queries for a given `MyISAM` table. By default, all `MyISAM` table indexes are cached in the default key cache. To assign table indexes to a specific key cache, use the `CACHE INDEX` statement (see Section 13.7.8.2, “CACHE INDEX Syntax”). For example, the following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a `SET GLOBAL` parameter setting statement or by using server startup options. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

To destroy a key cache, set its size to zero:

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

You cannot destroy the default key cache. Any attempt to do this is ignored:

```
mysql> SET GLOBAL key_buffer_size = 0;

mysql> SHOW VARIABLES LIKE 'key_buffer_size';
+-----------------+---------+
| Variable_name   | Value   |
+-----------------+---------+
| key_buffer_size | 8384512 |
+-----------------+---------+
```

Key cache variables are structured system variables that have a name and components. For `keycache1.key_buffer_size`, `keycache1` is the cache variable name and `key_buffer_size` is the cache component. See Section 5.1.9.5, “Structured System Variables”, for a description of the syntax used for referring to structured key cache system variables.

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

For a busy server, you can use a strategy that involves three key caches:

- A “hot” key cache that takes up 20% of the space allocated for all key caches. Use this for tables that are heavily used for searches but that are not updated.
- A “cold” key cache that takes up 20% of the space allocated for all key caches. Use this cache for medium-sized, intensively modified tables, such as temporary tables.
- A “warm” key cache that takes up 60% of the key cache space. Employ this as the default key cache, to be used by default for all other tables.

One reason the use of three key caches is beneficial is that access to one key cache structure does not block access to the others. Statements that access tables assigned to one cache do not compete with statements that access tables assigned to another cache. Performance gains occur for other reasons as well:

- The hot cache is used only for retrieval queries, so its contents are never modified. Consequently, whenever an index block needs to be pulled in from disk, the contents of the cache block chosen for replacement need not be flushed first.
- For an index assigned to the hot cache, if there are no queries requiring an index scan, there is a high probability that the index blocks corresponding to nonleaf nodes of the index B-tree remain in the cache.
- An update operation most frequently executed for temporary tables is performed much faster when the updated node is in the cache and need not be read from disk first. If the size of the indexes of the temporary tables are comparable with the size of cold key cache, the probability is very high that the updated node is in the cache.

The `CACHE INDEX` statement sets up an association between a table and a key cache, but the association is lost each time the server restarts. If you want the association to take effect each time the server starts, one way to accomplish this is to use an option file: Include variable settings that configure your key caches, and an `init_file` system variable that names a file containing `CACHE INDEX` statements to be executed. For example:

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
init_file=/path/to/data-directory/mysqld_init.sql
```

The statements in `mysqld_init.sql` are executed each time the server starts. The file should contain one SQL statement per line. The following example assigns several tables each to `hot_cache` and `cold_cache`:

```
CACHE INDEX db1.t1, db1.t2, db2.t3 IN hot_cache
CACHE INDEX db1.t4, db2.t5, db2.t6 IN cold_cache
```

8.10.2.3 Midpoint Insertion Strategy

By default, the key cache management system uses a simple LRU strategy for choosing key cache blocks to be evicted, but it also supports a more sophisticated method called the *midpoint insertion strategy.*

When using the midpoint insertion strategy, the LRU chain is divided into two parts: a hot sublist and a warm sublist. The division point between two parts is not fixed, but the key cache management system takes care that the warm part is not “too short,” always containing at least `key_cache_division_limit` percent of the key cache blocks. `key_cache_division_limit` is a component of structured key cache variables, so its value is a parameter that can be set per cache.

When an index block is read from a table into the key cache, it is placed at the end of the warm sublist. After a certain number of hits (accesses of the block), it is promoted to the hot sublist. At present, the number of hits required to promote a block (3) is the same for all index blocks.

A block promoted into the hot sublist is placed at the end of the list. The block then circulates within this sublist. If the block stays at the beginning of the sublist for a long enough time, it is demoted to the warm sublist. This time is determined by the value of the `key_cache_age_threshold` component of the key cache.

The threshold value prescribes that, for a key cache containing *`N`* blocks, the block at the beginning of the hot sublist not accessed within the last *`N`* `* key_cache_age_threshold / 100` hits is to be moved to the beginning of the warm sublist. It then becomes the first candidate for eviction, because blocks for replacement always are taken from the beginning of the warm sublist.

The midpoint insertion strategy enables you to keep more-valued blocks always in the cache. If you prefer to use the plain LRU strategy, leave the `key_cache_division_limit` value set to its default of 100.

The midpoint insertion strategy helps to improve performance when execution of a query that requires an index scan effectively pushes out of the cache all the index blocks corresponding to valuable high-level B-tree nodes. To avoid this, you must use a midpoint insertion strategy with the `key_cache_division_limit` set to much less than 100. Then valuable frequently hit nodes are preserved in the hot sublist during an index scan operation as well.

8.10.2.4 Index Preloading

If there are enough blocks in a key cache to hold blocks of an entire index, or at least the blocks corresponding to its nonleaf nodes, it makes sense to preload the key cache with index blocks before starting to use it. Preloading enables you to put the table index blocks into a key cache buffer in the most efficient way: by reading the index blocks from disk sequentially.

Without preloading, the blocks are still placed into the key cache as needed by queries. Although the blocks will stay in the cache, because there are enough buffers for all of them, they are fetched from disk in random order, and not sequentially.

To preload an index into a cache, use the `LOAD INDEX INTO CACHE` statement. For example, the following statement preloads nodes (index blocks) of indexes of the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

The `IGNORE LEAVES` modifier causes only blocks for the nonleaf nodes of the index to be preloaded. Thus, the statement shown preloads all index blocks from `t1`, but only blocks for the nonleaf nodes from `t2`.

If an index has been assigned to a key cache using a `CACHE INDEX` statement, preloading places index blocks into that cache. Otherwise, the index is loaded into the default key cache.

8.10.2.5 Key Cache Block Size

It is possible to specify the size of the block buffers for an individual key cache using the `key_cache_block_size` variable. This permits tuning of the performance of I/O operations for index files.

The best performance for I/O operations is achieved when the size of read buffers is equal to the size of the native operating system I/O buffers. But setting the size of key nodes equal to the size of the I/O buffer does not always ensure the best overall performance. When reading the big leaf nodes, the server pulls in a lot of unnecessary data, effectively preventing reading other leaf nodes.

To control the size of blocks in the `.MYI` index file of `MyISAM` tables, use the `--myisam-block-size` option at server startup.

8.10.2.6 Restructuring a Key Cache

A key cache can be restructured at any time by updating its parameter values. For example:

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

If you assign to either the `key_buffer_size` or `key_cache_block_size` key cache component a value that differs from the component's current value, the server destroys the cache's old structure and creates a new one based on the new values. If the cache contains any dirty blocks, the server saves them to disk before destroying and re-creating the cache. Restructuring does not occur if you change other key cache parameters.

When restructuring a key cache, the server first flushes the contents of any dirty buffers to disk. After that, the cache contents become unavailable. However, restructuring does not block queries that need to use indexes assigned to the cache. Instead, the server directly accesses the table indexes using native file system caching. File system caching is not as efficient as using a key cache, so although queries execute, a slowdown can be anticipated. After the cache has been restructured, it becomes available again for caching indexes assigned to it, and the use of file system caching for the indexes ceases.

8.10.3 Caching of Prepared Statements and Stored Programs

For certain statements that a client might execute multiple times during a session, the server converts the statement to an internal structure and caches that structure to be used during execution. Caching enables the server to perform more efficiently because it avoids the overhead of reconverting the statement should it be needed again during the session. Conversion and caching occurs for these statements:

- Prepared statements, both those processed at the SQL level (using the `PREPARE` statement) and those processed using the binary client/server protocol (using the `mysql_stmt_prepare()` C API function). The `max_prepared_stmt_count` system variable controls the total number of statements the server caches. (The sum of the number of prepared statements across all sessions.)
- Stored programs (stored procedures and functions, triggers, and events). In this case, the server converts and caches the entire program body. The `stored_program_cache` system variable indicates the approximate number of stored programs the server caches per session.

The server maintains caches for prepared statements and stored programs on a per-session basis. Statements cached for one session are not accessible to other sessions. When a session ends, the server discards any statements cached for it.

When the server uses a cached internal statement structure, it must take care that the structure does not go out of date. Metadata changes can occur for an object used by the statement, causing a mismatch between the current object definition and the definition as represented in the internal statement structure. Metadata changes occur for DDL statements such as those that create, drop, alter, rename, or truncate tables, or that analyze, optimize, or repair tables. Table content changes (for example, with `INSERT` or `UPDATE`) do not change metadata, nor do `SELECT` statements.

Here is an illustration of the problem. Suppose that a client prepares this statement:

```
PREPARE s1 FROM 'SELECT * FROM t1';
```

The `SELECT *` expands in the internal structure to the list of columns in the table. If the set of columns in the table is modified with `ALTER TABLE`, the prepared statement goes out of date. If the server does not detect this change the next time the client executes `s1`, the prepared statement will return incorrect results.

To avoid problems caused by metadata changes to tables or views referred to by the prepared statement, the server detects these changes and automatically reprepares the statement when it is next executed. That is, the server reparses the statement and rebuilds the internal structure. Reparsing also occurs after referenced tables or views are flushed from the table definition cache, either implicitly to make room for new entries in the cache, or explicitly due to `FLUSH TABLES`.

Similarly, if changes occur to objects used by a stored program, the server reparses affected statements within the program.

The server also detects metadata changes for objects in expressions. These might be used in statements specific to stored programs, such as `DECLARE CURSOR` or flow-control statements such as `IF`, `CASE`, and `RETURN`.

To avoid reparsing entire stored programs, the server reparses affected statements or expressions within a program only as needed. Examples:

- Suppose that metadata for a table or view is changed. Reparsing occurs for a `SELECT *` within the program that accesses the table or view, but not for a `SELECT *` that does not access the table or view.
- When a statement is affected, the server reparses it only partially if possible. Consider this `CASE` statement:

  ```
  CASE case_expr
    WHEN when_expr1 ...
    WHEN when_expr2 ...
    WHEN when_expr3 ...
    ...
  END CASE
  ```

  If a metadata change affects only `WHEN when_expr3`, that expression is reparsed. *`case_expr`* and the other `WHEN` expressions are not reparsed.

Reparsing uses the default database and SQL mode that were in effect for the original conversion to internal form.

The server attempts reparsing up to three times. An error occurs if all attempts fail.

Reparsing is automatic, but to the extent that it occurs, diminishes prepared statement and stored program performance.

For prepared statements, the `Com_stmt_reprepare` status variable tracks the number of repreparations.

8.11 Optimizing Locking Operations

MySQL manages contention for table contents using locking:

- Internal locking is performed within the MySQL server itself to manage contention for table contents by multiple threads. This type of locking is internal because it is performed entirely by the server and involves no other programs. See Section 8.11.1, “Internal Locking Methods”.

- External locking occurs when the server and other programs lock `MyISAM` table files to coordinate among themselves which program can access the tables at which time. See Section 8.11.5, “External Locking”.

8.11.1 Internal Locking Methods

This section discusses internal locking; that is, locking performed within the MySQL server itself to manage contention for table contents by multiple sessions. This type of locking is internal because it is performed entirely by the server and involves no other programs. For locking performed on MySQL files by other programs, see Section 8.11.5, “External Locking”.

- Row-Level Locking
- Table-Level Locking
- Choosing the Type of Locking

Row-Level Locking

MySQL uses row-level locking for `InnoDB` tables to support simultaneous write access by multiple sessions, making them suitable for multi-user, highly concurrent, and OLTP applications.

To avoid deadlocks when performing multiple concurrent write operations on a single `InnoDB` table, acquire necessary locks at the start of the transaction by issuing a `SELECT ... FOR UPDATE` statement for each group of rows expected to be modified, even if the data change statements come later in the transaction. If transactions modify or lock more than one table, issue the applicable statements in the same order within each transaction. Deadlocks affect performance rather than representing a serious error, because `InnoDB` automatically detects deadlock conditions by default and rolls back one of the affected transactions.

On high concurrency systems, deadlock detection can cause a slowdown when numerous threads wait for the same lock. At times, it may be more efficient to disable deadlock detection and rely on the `innodb_lock_wait_timeout` setting for transaction rollback when a deadlock occurs. Deadlock detection can be disabled using the `innodb_deadlock_detect` configuration option.

Advantages of row-level locking:

- Fewer lock conflicts when different sessions access different rows.
- Fewer changes for rollbacks.
- Possible to lock a single row for a long time.

Table-Level Locking

MySQL uses table-level locking for `MyISAM`, `MEMORY`, and `MERGE` tables, permitting only one session to update those tables at a time. This locking level makes these storage engines more suitable for read-only, read-mostly, or single-user applications.

These storage engines avoid deadlocks by always requesting all needed locks at once at the beginning of a query and always locking the tables in the same order. The tradeoff is that this strategy reduces concurrency; other sessions that want to modify the table must wait until the current data change statement finishes.

Advantages of table-level locking:

- Relatively little memory required (row locking requires memory per row or group of rows locked)
- Fast when used on a large part of the table because only a single lock is involved.
- Fast if you often do `GROUP BY` operations on a large part of the data or must scan the entire table frequently.

MySQL grants table write locks as follows:

1. If there are no locks on the table, put a write lock on it.
2. Otherwise, put the lock request in the write lock queue.

MySQL grants table read locks as follows:

1. If there are no write locks on the table, put a read lock on it.
2. Otherwise, put the lock request in the read lock queue.

Table updates are given higher priority than table retrievals. Therefore, when a lock is released, the lock is made available to the requests in the write lock queue and then to the requests in the read lock queue. This ensures that updates to a table are not "starved" even when there is heavy `SELECT` activity for the table. However, if there are many updates for a table, `SELECT` statements wait until there are no more updates.

For information on altering the priority of reads and writes, see Section 8.11.2, "Table Locking Issues".

You can analyze the table lock contention on your system by checking the `Table_locks_immediate` and `Table_locks_waited` status variables, which indicate the number of times that requests for table locks could be granted immediately and the number that had to wait, respectively:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| Table_locks_immediate | 1151552 |
| Table_locks_waited    | 15324   |
+-----------------------+---------+
```

The Performance Schema lock tables also provide locking information. See Section 26.12.12, "Performance Schema Lock Tables".

The `MyISAM` storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a `MyISAM` table has no free blocks in the middle of the data file, rows are always inserted at the end of the data file. In this case, you can freely mix concurrent `INSERT` and `SELECT` statements for a `MyISAM` table without locks. That is, you can insert rows into a `MyISAM` table at the same time other clients are reading from it. Holes can result from rows having been deleted from or updated in the middle of the table. If there are holes, concurrent inserts are disabled but are enabled again automatically when all holes have been filled with new data. To control this behavior, use the `concurrent_insert` system variable. See Section 8.11.3, "Concurrent Inserts".

If you acquire a table lock explicitly with `LOCK TABLES`, you can request a `READ LOCAL` lock rather than a `READ` lock to enable other sessions to perform concurrent inserts while you have the table locked.

To perform many `INSERT` and `SELECT` operations on a table `t1` when concurrent inserts are not possible, you can insert rows into a temporary table `temp_t1` and update the real table with the rows from the temporary table:

```
mysql> LOCK TABLES t1 WRITE, temp_t1 WRITE;
mysql> INSERT INTO t1 SELECT * FROM temp_t1;
mysql> DELETE FROM temp_t1;
mysql> UNLOCK TABLES;
```

Choosing the Type of Locking

Generally, table locks are superior to row-level locks in the following cases:

- Most statements for the table are reads.
- Statements for the table are a mix of reads and writes, where writes are updates or deletes for a single row that can be fetched with one key read:

```
UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
DELETE FROM tbl_name WHERE unique_key_col=key_value;
```

- `SELECT` combined with concurrent `INSERT` statements, and very few `UPDATE` or `DELETE` statements.
- Many scans or `GROUP BY` operations on the entire table without any writers.

With higher-level locks, you can more easily tune applications by supporting locks of different types, because the lock overhead is less than for row-level locks.

Options other than row-level locking:

- Versioning (such as that used in MySQL for concurrent inserts) where it is possible to have one writer at the same time as many readers. This means that the database or table supports different views for the data depending on when access begins. Other common terms for this are “time travel,” “copy on write,” or “copy on demand.”
- Copy on demand is in many cases superior to row-level locking. However, in the worst case, it can use much more memory than using normal locks.
- Instead of using row-level locks, you can employ application-level locks, such as those provided by `GET_LOCK()` and `RELEASE_LOCK()` in MySQL. These are advisory locks, so they work only with applications that cooperate with each other. See Section 12.14, “Locking Functions”.

8.11.2 Table Locking Issues

`InnoDB` tables use row-level locking so that multiple sessions and applications can read from and write to the same table simultaneously, without making each other wait or producing inconsistent results. For this storage engine, avoid using the `LOCK TABLES` statement, because it does not offer any extra protection, but instead reduces concurrency. The automatic row-level locking makes these tables suitable for your busiest databases with your most important data, while also simplifying application logic since you do not need to lock and unlock tables. Consequently, the `InnoDB` storage engine is the default in MySQL.

MySQL uses table locking (instead of page, row, or column locking) for all storage engines except `InnoDB`. The locking operations themselves do not have much overhead. But because only one session can write to a table at any one time, for best performance with these other storage engines, use them primarily for tables that are queried often and rarely inserted into or updated.

- Performance Considerations Favoring InnoDB
- Workarounds for Locking Performance Issues

Performance Considerations Favoring InnoDB

When choosing whether to create a table using `InnoDB` or a different storage engine, keep in mind the following disadvantages of table locking:

- Table locking enables many sessions to read from a table at the same time, but if a session wants to write to a table, it must first get exclusive access, meaning it might have to wait for other sessions to finish with the table first. During the update, all other sessions that want to access this particular table must wait until the update is done.
- Table locking causes problems when a session is waiting because the disk is full and free space needs to become available before the session can proceed. In this case, all sessions that want to access the problem table are also put in a waiting state until more disk space is made available.
- A `SELECT` statement that takes a long time to run prevents other sessions from updating the table in the meantime, making the other sessions appear slow or unresponsive. While a session is waiting to get exclusive access to the table for updates, other sessions that issue `SELECT` statements will queue up behind it, reducing concurrency even for read-only sessions.

Workarounds for Locking Performance Issues

The following items describe some ways to avoid or reduce contention caused by table locking:

- Consider switching the table to the `InnoDB` storage engine, either using `CREATE TABLE ... ENGINE=INNODB` during setup, or using `ALTER TABLE ... ENGINE=INNODB` for an existing table. See Chapter 15, *The InnoDB Storage Engine* for more details about this storage engine.
- Optimize `SELECT` statements to run faster so that they lock tables for a shorter time. You might have to create some summary tables to do this.
- Start `mysqld` with `--low-priority-updates`. For storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`), this gives all statements that update (modify) a table lower priority than `SELECT` statements. In this case, the second `SELECT` statement in the preceding scenario would execute before the `UPDATE` statement, and would not wait for the first `SELECT` to finish.
- To specify that all updates issued in a specific connection should be done with low priority, set the `low_priority_updates` server system variable equal to 1.
- To give a specific `INSERT`, `UPDATE`, or `DELETE` statement lower priority, use the `LOW_PRIORITY` attribute.
- To give a specific `SELECT` statement higher priority, use the `HIGH_PRIORITY` attribute. See Section 13.2.10, “SELECT Syntax”.
- Start `mysqld` with a low value for the `max_write_lock_count` system variable to force MySQL to temporarily elevate the priority of all `SELECT` statements that are waiting for a table after a specific number of inserts to the table occur. This permits `READ` locks after a certain number of `WRITE` locks.
- If you have problems with `INSERT` combined with `SELECT`, consider switching to `MyISAM` tables, which support concurrent `SELECT` and `INSERT` statements. (See Section 8.11.3, “Concurrent Inserts”.)
- If you have problems with mixed `SELECT` and `DELETE` statements, the `LIMIT` option to `DELETE` may help. See Section 13.2.2, “DELETE Syntax”.
- Using `SQL_BUFFER_RESULT` with `SELECT` statements can help to make the duration of table locks shorter. See Section 13.2.10, “SELECT Syntax”.

- Splitting table contents into separate tables may help, by allowing queries to run against columns in one table, while updates are confined to columns in a different table.
- You could change the locking code in `mysys/thr_lock.c` to use a single queue. In this case, write locks and read locks would have the same priority, which might help some applications.

8.11.3 Concurrent Inserts

The `MyISAM` storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a `MyISAM` table has no holes in the data file (deleted rows in the middle), an `INSERT` statement can be executed to add rows to the end of the table at the same time that `SELECT` statements are reading rows from the table. If there are multiple `INSERT` statements, they are queued and performed in sequence, concurrently with the `SELECT` statements. The results of a concurrent `INSERT` may not be visible immediately.

The `concurrent_insert` system variable can be set to modify the concurrent-insert processing. By default, the variable is set to `AUTO` (or 1) and concurrent inserts are handled as just described. If `concurrent_insert` is set to `NEVER` (or 0), concurrent inserts are disabled. If the variable is set to `ALWAYS` (or 2), concurrent inserts at the end of the table are permitted even for tables that have deleted rows. See also the description of the `concurrent_insert` system variable.

If you are using the binary log, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. See Section 5.4.4, “The Binary Log”. In addition, for those statements a read lock is placed on the selected-from table such that inserts into that table are blocked. The effect is that concurrent inserts for that table must wait as well.

With `LOAD DATA`, if you specify `CONCURRENT` with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other sessions can retrieve data from the table while `LOAD DATA` is executing. Use of the `CONCURRENT` option affects the performance of `LOAD DATA` a bit, even if no other session is using the table at the same time.

If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` option if the server was started with that option. It also causes concurrent inserts not to be used.

For `LOCK TABLE`, the difference between `READ LOCAL` and `READ` is that `READ LOCAL` permits nonconflicting `INSERT` statements (concurrent inserts) to execute while the lock is held. However, this cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock.

8.11.4 Metadata Locking

MySQL uses metadata locking to manage concurrent access to database objects and to ensure data consistency. Metadata locking applies not just to tables, but also to schemas, stored programs (procedures, functions, triggers, scheduled events), tablespaces, user locks acquired with the `GET_LOCK()` function (see Section 12.14, “Locking Functions”), and locks acquired with the locking service described in Section 29.3.1, “The Locking Service”.

The Performance Schema `metadata_locks` table exposes metadata lock information, which can be useful for seeing which sessions hold locks, are blocked waiting for locks, and so forth. For details, see Section 26.12.12.3, “The metadata_locks Table”.

Metadata locking does involve some overhead, which increases as query volume increases. Metadata contention increases the more that multiple queries attempt to access the same objects.

Metadata locking is not a replacement for the table definition cache, and its mutexes and locks differ from the `LOCK_open` mutex. The following discussion provides some information about how metadata locking works.

- Metadata Lock Acquisition
- Metadata Lock Release

Metadata Lock Acquisition

If there are multiple waiters for a given lock, the highest-priority lock request is satisfied first, with an exception related to the `max_write_lock_count` system variable. Write lock requests have higher priority than read lock requests. However, if `max_write_lock_count` is set to some low value (say, 10), read lock requests may be preferred over pending write lock requests if the read lock requests have already been passed over in favor of 10 write lock requests. Normally this behavior does not occur because `max_write_lock_count` by default has a very large value.

Statements acquire metadata locks one by one, not simultaneously, and perform deadlock detection in the process.

DML statements normally acquire locks in the order in which tables are mentioned in the statement.

DDL statements, `LOCK TABLES`, and other similar statements try to reduce the number of possible deadlocks between concurrent DDL statements by acquiring locks on explicitly named tables in name order. Locks might be acquired in a different order for implicitly used tables (such as tables in foreign key relationships that also must be locked).

For example, `RENAME TABLE` is a DDL statement that acquires locks in name order:

- This `RENAME TABLE` statement renames `tbla` to something else, and renames `tblc` to `tbla`:

  ```
  RENAME TABLE tbla TO tbld, tblc TO tbla;
  ```

  The statement acquires metadata locks, in order, on `tbla`, `tblc`, and `tbld` (because `tbld` follows `tblc` in name order):

- This slightly different statement also renames `tbla` to something else, and renames `tblc` to `tbla`:

  ```
  RENAME TABLE tbla TO tblb, tblc TO tbla;
  ```

  In this case, the statement acquires metadata locks, in order, on `tbla`, `tblb`, and `tblc` (because `tblb` precedes `tblc` in name order):

Both statements acquire locks on `tbla` and `tblc`, in that order, but differ in whether the lock on the remaining table name is acquired before or after `tblc`.

Metadata lock acquisition order can make a difference in operation outcome when multiple transactions execute concurrently, as the following example illustrates.

Begin with two tables `x` and `x_new` that have identical structure. Three clients issue statements that involve these tables:

Client 1:

```
LOCK TABLE x WRITE, x_new WRITE;
```

The statement requests and acquires write locks in name order on `x` and `x_new`.

Client 2:

```
INSERT INTO x VALUES(1);
```

The statement requests and blocks waiting for a write lock on `x`.

Client 3:

```
RENAME TABLE x TO x_old, x_new TO x;
```

The statement requests exclusive locks in name order on `x`, `x_new`, and `x_old`, but blocks waiting for the lock on `x`.

Client 1:

```
UNLOCK TABLES;
```

The statement releases the write locks on `x` and `x_new`. The exclusive lock request for `x` by Client 3 has higher priority than the write lock request by Client 2, so Client 3 acquires its lock on `x`, then also on `x_new` and `x_old`, performs the renaming, and releases its locks. Client 2 then acquires its lock on `x`, performs the insert, and releases its lock.

Lock acquisition order results in the `RENAME TABLE` executing before the `INSERT`. The `x` into which the insert occurs is the table that was named `x_new` when Client 2 issued the insert and was renamed to `x` by Client 3:

```
mysql> SELECT * FROM x;
+------+
| i    |
+------+
|    1 |
+------+

mysql> SELECT * FROM x_old;
Empty set (0.01 sec)
```

Now begin instead with tables named `x` and `new_x` that have identical structure. Again, three clients issue statements that involve these tables:

Client 1:

```
LOCK TABLE x WRITE, new_x WRITE;
```

The statement requests and acquires write locks in name order on `new_x` and `x`.

Client 2:

```
INSERT INTO x VALUES(1);
```

The statement requests and blocks waiting for a write lock on `x`.

Client 3:

```
```

```
RENAME TABLE x TO old_x, new_x TO x;
```

The statement requests exclusive locks in name order on `new_x`, `old_x`, and `x`, but blocks waiting for the lock on `new_x`.

Client 1:

```
UNLOCK TABLES;
```

The statement releases the write locks on `x` and `new_x`. For `x`, the only pending request is by Client 2, so Client 2 acquires its lock, performs the insert, and releases the lock. For `new_x`, the only pending request is by Client 3, which is permitted to acquire that lock (and also the lock on `old_x`). The rename operation still blocks for the lock on `x` until the Client 2 insert finishes and releases its lock. Then Client 3 acquires the lock on `x`, performs the rename, and releases its lock.

In this case, lock acquisition order results in the `INSERT` executing before the `RENAME TABLE`. The `x` into which the insert occurs is the original `x`, now renamed to `old_x` by the rename operation:

```
mysql> SELECT * FROM x;
Empty set (0.01 sec)

mysql> SELECT * FROM old_x;
+------+
| i    |
+------+
|    1 |
+------+
```

If order of lock acquisition in concurrent statements makes a difference to an application in operation outcome, as in the preceding example, you may be able to adjust the table names to affect the order of lock acquisition.

Metadata locks are extended, as necessary, to tables related by a foreign key constraint to prevent conflicting DML and DDL operations from executing concurrently on the related tables. When updating a parent table, a metadata lock is taken on the child table while updating foreign key metadata. Foreign key metadata is owned by the child table.

Metadata Lock Release

To ensure transaction serializability, the server must not permit one session to perform a data definition language (DDL) statement on a table that is used in an uncompleted explicitly or implicitly started transaction in another session. The server achieves this by acquiring metadata locks on tables used within a transaction and deferring release of those locks until the transaction ends. A metadata lock on a table prevents changes to the table's structure. This locking approach has the implication that a table that is being used by a transaction within one session cannot be used in DDL statements by other sessions until the transaction ends.

This principle applies not only to transactional tables, but also to nontransactional tables. Suppose that a session begins a transaction that uses transactional table `t` and nontransactional table `nt` as follows:

```
START TRANSACTION;
SELECT * FROM t;
SELECT * FROM nt;
```

The server holds metadata locks on both `t` and `nt` until the transaction ends. If another session attempts a DDL or write lock operation on either table, it blocks until metadata lock release at transaction end. For example, a second session blocks if it attempts any of these operations:

```
DROP TABLE t;
ALTER TABLE t ...;
DROP TABLE nt;
ALTER TABLE nt ...;
LOCK TABLE t ... WRITE;
```

The same behavior applies for The `LOCK TABLES ... READ`. That is, explicitly or implicitly started transactions that update any table (transactional or nontransactional) will block and be blocked by `LOCK TABLES ... READ` for that table.

If the server acquires metadata locks for a statement that is syntactically valid but fails during execution, it does not release the locks early. Lock release is still deferred to the end of the transaction because the failed statement is written to the binary log and the locks protect log consistency.

In autocommit mode, each statement is in effect a complete transaction, so metadata locks acquired for the statement are held only to the end of the statement.

Metadata locks acquired during a `PREPARE` statement are released once the statement has been prepared, even if preparation occurs within a multiple-statement transaction.

As of MySQL 8.0.13, for XA transactions in `PREPARED` state, metadata locks are maintained across client disconnects and server restarts, until an `XA COMMIT` or `XA ROLLBACK` is executed.

8.11.5 External Locking

External locking is the use of file system locking to manage contention for `MyISAM` database tables by multiple processes. External locking is used in situations where a single process such as the MySQL server cannot be assumed to be the only process that requires access to tables. Here are some examples:

- If you run multiple servers that use the same database directory (not recommended), each server must have external locking enabled.
- If you use `myisamchk` to perform table maintenance operations on `MyISAM` tables, you must either ensure that the server is not running, or that the server has external locking enabled so that it locks table files as necessary to coordinate with `myisamchk` for access to the tables. The same is true for use of `myisampack` to pack `MyISAM` tables.

  If the server is run with external locking enabled, you can use `myisamchk` at any time for read operations such a checking tables. In this case, if the server tries to update a table that `myisamchk` is using, the server will wait for `myisamchk` to finish before it continues.

  If you use `myisamchk` for write operations such as repairing or optimizing tables, or if you use `myisampack` to pack tables, you *must* always ensure that the `mysqld` server is not using the table. If you do not stop `mysqld`, at least do a `mysqladmin flush-tables` before you run `myisamchk`. Your tables *may become corrupted* if the server and `myisamchk` access the tables simultaneously.

With external locking in effect, each process that requires access to a table acquires a file system lock for the table files before proceeding to access the table. If all necessary locks cannot be acquired, the process is blocked from accessing the table until the locks can be obtained (after the process that currently holds the locks releases them).

External locking affects server performance because the server must sometimes wait for other processes before it can access tables.

External locking is unnecessary if you run a single server to access a given data directory (which is the usual case) and if no other programs such as `myisamchk` need to modify tables while the server is

running. If you only *read* tables with other programs, external locking is not required, although `myisamchk` might report warnings if the server changes tables while `myisamchk` is reading them.

With external locking disabled, to use `myisamchk`, you must either stop the server while `myisamchk` executes or else lock and flush the tables before running `myisamchk`. (See System Factors.) To avoid this requirement, use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables.

For `mysqld`, external locking is controlled by the value of the `skip_external_locking` system variable. When this variable is enabled, external locking is disabled, and vice versa. External locking is disabled by default.

Use of external locking can be controlled at server startup by using the `--external-locking` or `--skip-external-locking` option.

If you do use external locking option to enable updates to `MyISAM` tables from many MySQL processes, do not start the server with the `delay_key_write` system variable set to `ALL` or use the `DELAY_KEY_WRITE=1` table option for any shared tables. Otherwise, index corruption can occur.

The easiest way to satisfy this condition is to always use `--external-locking` together with `--delay-key-write=OFF`. (This is not done by default because in many setups it is useful to have a mixture of the preceding options.)

8.12 Optimizing the MySQL Server

This section discusses optimization techniques for the database server, primarily dealing with system configuration rather than tuning SQL statements. The information in this section is appropriate for DBAs who want to ensure performance and scalability across the servers they manage; for developers constructing installation scripts that include setting up the database; and people running MySQL themselves for development, testing, and so on who want to maximize their own productivity.

8.12.1 Optimizing Disk I/O

This section describes ways to configure storage devices when you can devote more and faster storage hardware to the database server. For information about optimizing an `InnoDB` configuration to improve I/O performance, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- Disk seeks are a huge performance bottleneck. This problem becomes more apparent when the amount of data starts to grow so large that effective caching becomes impossible. For large databases where you access data more or less randomly, you can be sure that you need at least one disk seek to read and a couple of disk seeks to write things. To minimize this problem, use disks with low seek times.
- Increase the number of available disk spindles (and thereby reduce the seek overhead) by either symlinking files to different disks or striping the disks:
  - Using symbolic links

    This means that, for `MyISAM` tables, you symlink the index file and data files from their usual location in the data directory to another disk (that may also be striped). This makes both the seek and read times better, assuming that the disk is not used for other purposes as well. See Section 8.12.2, “Using Symbolic Links”.

    Symbolic links are not supported for use with `InnoDB` tables. However, it is possible to place `InnoDB` data and log files on different physical disks. For more information, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

  - Striping

Striping means that you have many disks and put the first block on the first disk, the second block on the second disk, and the *`N`*-th block on the (*`N MOD number_of_disks`*) disk, and so on. This means if your normal data size is less than the stripe size (or perfectly aligned), you get much better performance. Striping is very dependent on the operating system and the stripe size, so benchmark your application with different stripe sizes. See Section 8.13.2, “Using Your Own Benchmarks”.

The speed difference for striping is *very* dependent on the parameters. Depending on how you set the striping parameters and number of disks, you may get differences measured in orders of magnitude. You have to choose to optimize for random or sequential access.

- For reliability, you may want to use RAID 0+1 (striping plus mirroring), but in this case, you need 2 × *`N`* drives to hold *`N`* drives of data. This is probably the best option if you have the money for it. However, you may also have to invest in some volume-management software to handle it efficiently.
- A good option is to vary the RAID level according to how critical a type of data is. For example, store semi-important data that can be regenerated on a RAID 0 disk, but store really important data such as host information and logs on a RAID 0+1 or RAID *`N`* disk. RAID *`N`* can be a problem if you have many writes, due to the time required to update the parity bits.
- You can also set the parameters for the file system that the database uses:

  If you do not need to know when files were last accessed (which is not really useful on a database server), you can mount your file systems with the `-o noatime` option. That skips updates to the last access time in inodes on the file system, which avoids some disk seeks.

  On many operating systems, you can set a file system to be updated asynchronously by mounting it with the `-o async` option. If your computer is reasonably stable, this should give you better performance without sacrificing too much reliability. (This flag is on by default on Linux.)

Using NFS with MySQL

Caution is advised when considering using NFS with MySQL. Potential issues, which vary by operating system and NFS version, include:

- MySQL data and log files placed on NFS volumes becoming locked and unavailable for use. Locking issues may occur in cases where multiple instances of MySQL access the same data directory or where MySQL is shut down improperly, due to a power outage, for example. NFS version 4 addresses underlying locking issues with the introduction of advisory and lease-based locking. However, sharing a data directory among MySQL instances is not recommended.
- Data inconsistencies introduced due to messages received out of order or lost network traffic. To avoid this issue, use TCP with `hard` and `intr` mount options.
- Maximum file size limitations. NFS Version 2 clients can only access the lowest 2GB of a file (signed 32 bit offset). NFS Version 3 clients support larger files (up to 64 bit offsets). The maximum supported file size also depends on the local file system of the NFS server.

Using NFS within a professional SAN environment or other storage system tends to offer greater reliability than using NFS outside of such an environment. However, NFS within a SAN environment may be slower than directly attached or bus-attached non-rotational storage.

If you choose to use NFS, NFS Version 4 or later is recommended, as is testing your NFS setup thoroughly before deploying into a production environment.

8.12.2 Using Symbolic Links

You can move databases or tables from the database directory to other locations and replace them with symbolic links to the new locations. You might want to do this, for example, to move a database to a file system with more free space or increase the speed of your system by spreading your tables to different disks.

For `InnoDB` tables, use the `DATA DIRECTORY` clause on the `CREATE TABLE` statement instead of symbolic links, as explained in Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”. This new feature is a supported, cross-platform technique.

The recommended way to do this is to symlink entire database directories to a different disk. Symlink `MyISAM` tables only as a last resort.

To determine the location of your data directory, use this statement:

```
SHOW VARIABLES LIKE 'datadir';
```

8.12.2.1 Using Symbolic Links for Databases on Unix

On Unix, the way to symlink a database is first to create a directory on some disk where you have free space and then to create a soft link to it from the MySQL data directory.

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test /path/to/datadir
```

MySQL does not support linking one directory to multiple databases. Replacing a database directory with a symbolic link works as long as you do not make a symbolic link between databases. Suppose that you have a database `db1` under the MySQL data directory, and then make a symlink `db2` that points to `db1`:

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

The result is that, for any table `tbl_a` in `db1`, there also appears to be a table `tbl_a` in `db2`. If one client updates `db1.tbl_a` and another client updates `db2.tbl_a`, problems are likely to occur.

8.12.2.2 Using Symbolic Links for MyISAM Tables on Unix

Note

Symbolic link support as described here, along with the `--symbolic-links` option that controls it, is deprecated and will be removed in a future version of MySQL. In addition, the option is disabled by default.

Symlinks are fully supported only for `MyISAM` tables. For files used by tables for other storage engines, you may get strange problems if you try to use symbolic links. For `InnoDB` tables, use the alternative technique explained in Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory” instead.

Do not symlink tables on systems that do not have a fully operational `realpath()` call. (Linux and Solaris support `realpath()`). To determine whether your system supports symbolic links, check the value of the `have_symlink` system variable using this statement:

```
SHOW VARIABLES LIKE 'have_symlink';
```

The handling of symbolic links for `MyISAM` tables works as follows:

- In the data directory, you always have the data (`.MYD`) file and the index (`.MYI`) file. The data file and index file can be moved elsewhere and replaced in the data directory by symlinks.
- You can symlink the data file and the index file independently to different directories.
- To instruct a running MySQL server to perform the symlinking, use the `DATA DIRECTORY` and `INDEX DIRECTORY` options to `CREATE TABLE`. See Section 13.1.20, "CREATE TABLE Syntax". Alternatively, if `mysqld` is not running, symlinking can be accomplished manually using `ln -s` from the command line.

Note

The path used with either or both of the `DATA DIRECTORY` and `INDEX DIRECTORY` options may not include the MySQL `data` directory. (Bug #32167)

- `myisamchk` does not replace a symlink with the data file or index file. It works directly on the file to which the symlink points. Any temporary files are created in the directory where the data file or index file is located. The same is true for the `ALTER TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.
-

Note

When you drop a table that is using symlinks, *both the symlink and the file to which the symlink points are dropped*. This is an extremely good reason *not* to run `mysqld` as the `root` operating system user or permit operating system users to have write access to MySQL database directories.

- If you rename a table with `ALTER TABLE ... RENAME` or `RENAME TABLE` and you do not move the table to another database, the symlinks in the database directory are renamed to the new names and the data file and index file are renamed accordingly.
- If you use `ALTER TABLE ... RENAME` or `RENAME TABLE` to move a table to another database, the table is moved to the other database directory. If the table name changed, the symlinks in the new database directory are renamed to the new names and the data file and index file are renamed accordingly.
- If you are not using symlinks, start `mysqld` with the `--skip-symbolic-links` option to ensure that no one can use `mysqld` to drop or rename a file outside of the data directory.

These table symlink operations are not supported:

- `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.

8.12.2.3 Using Symbolic Links for Databases on Windows

On Windows, symbolic links can be used for database directories. This enables you to put a database directory at a different location (for example, on a different disk) by setting up a symbolic link to it. Use of database symlinks on Windows is similar to their use on Unix, although the procedure for setting up the link differs.

Suppose that you want to place the database directory for a database named `mydb` at `D:\data\mydb`. To do this, create a symbolic link in the MySQL data directory that points to `D:\data\mydb`. However, before creating the symbolic link, make sure that the `D:\data\mydb` directory exists by creating it if necessary. If you already have a database directory named `mydb` in the data directory, move it to `D:\data`. Otherwise, the symbolic link will be ineffective. To avoid problems, make sure that the server is not running when you move the database directory.

On Windows, you can create a symlink using the `mklink` command. This command requires administrative privileges.

1. Change location into the data directory:

```
C:\> cd \path\to\datadir
```

2. In the data directory, create a symlink named `mydb` that points to the location of the database directory:

```
C:\> mklink /d mydb D:\data\mydb
```

After this, all tables created in the database `mydb` are created in `D:\data\mydb`.

8.12.3 Optimizing Memory Use

8.12.3.1 How MySQL Uses Memory

MySQL allocates buffers and caches to improve performance of database operations. The default configuration is designed to permit a MySQL server to start on a virtual machine that has approximately 512MB of RAM. You can improve MySQL performance by increasing the values of certain cache and buffer-related system variables. You can also modify the default configuration to run MySQL on systems with limited memory.

The following list describes some of the ways that MySQL uses memory. Where applicable, relevant system variables are referenced. Some items are storage engine or feature specific.

- The `InnoDB` buffer pool is a memory area that holds cached `InnoDB` data for tables, indexes, and other auxiliary buffers. For efficiency of high-volume read operations, the buffer pool is divided into pages that can potentially hold multiple rows. For efficiency of cache management, the buffer pool is implemented as a linked list of pages; data that is rarely used is aged out of the cache, using a variation of the LRU algorithm. For more information, see Section 15.5.1, “Buffer Pool”.

  The size of the buffer pool is important for system performance:

  - `InnoDB` allocates memory for the entire buffer pool at server startup, using `malloc()` operations. The `innodb_buffer_pool_size` system variable defines the buffer pool size. Typically, a recommended `innodb_buffer_pool_size` value is 50 to 75 percent of system memory. `innodb_buffer_pool_size` can be configured dynamically, while the server is running. For more information, see Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”.

  - On systems with a large amount of memory, you can improve concurrency by dividing the buffer pool into multiple buffer pool instances. The `innodb_buffer_pool_instances` system variable defines the number of buffer pool instances.

  - A buffer pool that is too small may cause excessive churning as pages are flushed from the buffer pool only to be required again a short time later.

  - A buffer pool that is too large may cause swapping due to competition for memory.

- The storage engine interface enables the optimizer to provide information about the size of the record buffer to be used for scans that the optimizer estimates will read multiple rows. The buffer size can vary based on the size of the estimate. `InnoDB` uses this variable-size buffering capability to take advantage of row prefetching, and to reduce the overhead of latching and B-tree navigation.

- All threads share the `MyISAM` key buffer. The `key_buffer_size` system variable determines its size.

  For each `MyISAM` table the server opens, the index file is opened once; the data file is opened once for each concurrently running thread that accesses the table. For each concurrent thread, a table structure,

column structures for each column, and a buffer of size `3 * N` are allocated (where `N` is the maximum row length, not counting `BLOB` columns). A `BLOB` column requires five to eight bytes plus the length of the `BLOB` data. The `MyISAM` storage engine maintains one extra row buffer for internal use.

- The `myisam_use_mmap` system variable can be set to 1 to enable memory-mapping for all `MyISAM` tables.
- If an internal in-memory temporary table becomes too large (as determined using the `tmp_table_size` and `max_heap_table_size` system variables), MySQL automatically converts the table from in-memory to on-disk format. As of MySQL 8.0.16, on-disk temporary tables always use the InnoDB storage engine. (Previously, the storage engine employed for this purpose was determined by the `internal_tmp_disk_storage_engine` system variable, which is no longer supported.) You can increase the permissible temporary table size as described in Section 8.4.4, “Internal Temporary Table Use in MySQL”.

  For `MEMORY` tables explicitly created with `CREATE TABLE`, only the `max_heap_table_size` system variable determines how large a table can grow, and there is no conversion to on-disk format.
- The MySQL Performance Schema is a feature for monitoring MySQL server execution at a low level. The Performance Schema dynamically allocates memory incrementally, scaling its memory use to actual server load, instead of allocating required memory during server startup. Once memory is allocated, it is not freed until the server is restarted. For more information, see Section 26.17, “The Performance Schema Memory-Allocation Model”.
- Each thread that the server uses to manage client connections requires some thread-specific space. The following list indicates these and which system variables control their size:
  - A stack (`thread_stack`)
  - A connection buffer (`net_buffer_length`)
  - A result buffer (`net_buffer_length`)

  The connection buffer and result buffer each begin with a size equal to `net_buffer_length` bytes, but are dynamically enlarged up to `max_allowed_packet` bytes as needed. The result buffer shrinks to `net_buffer_length` bytes after each SQL statement. While a statement is running, a copy of the current statement string is also allocated.

  Each connection thread uses memory for computing statement digests. The server allocates `max_digest_length` bytes per session. See Section 26.10, “Performance Schema Statement Digests and Sampling”.
- All threads share the same base memory.
- When a thread is no longer needed, the memory allocated to it is released and returned to the system unless the thread goes back into the thread cache. In that case, the memory remains allocated.
- Each request that performs a sequential scan of a table allocates a *read buffer*. The `read_buffer_size` system variable determines the buffer size.
- When reading rows in an arbitrary sequence (for example, following a sort), a *random-read buffer* may be allocated to avoid disk seeks. The `read_rnd_buffer_size` system variable determines the buffer size.
- All joins are executed in a single pass, and most joins can be done without even using a temporary table. Most temporary tables are memory-based hash tables. Temporary tables with a large row length (calculated as the sum of all column lengths) or that contain `BLOB` columns are stored on disk.

- Most requests that perform a sort allocate a sort buffer and zero to two temporary files depending on the result set size. See Section B.4.3.5, “Where MySQL Stores Temporary Files”.
- Almost all parsing and calculating is done in thread-local and reusable memory pools. No memory overhead is needed for small items, thus avoiding the normal slow memory allocation and freeing. Memory is allocated only for unexpectedly large strings.
- For each table having `BLOB` columns, a buffer is enlarged dynamically to read in larger `BLOB` values. If you scan a table, the buffer grows as large as the largest `BLOB` value.
- MySQL requires memory and descriptors for the table cache. Handler structures for all in-use tables are saved in the table cache and managed as “First In, First Out” (FIFO). The `table_open_cache` system variable defines the initial table cache size; see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.

  MySQL also requires memory for the table definition cache. The `table_definition_cache` system variable defines the number of table definitions that can be stored in the table definition cache. If you use a large number of tables, you can create a large table definition cache to speed up the opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the table cache.
- A `FLUSH TABLES` statement or `mysqladmin flush-tables` command closes all tables that are not in use at once and marks all in-use tables to be closed when the currently executing thread finishes. This effectively frees most in-use memory. `FLUSH TABLES` does not return until all tables have been closed.
- The server caches information in memory as a result of `GRANT`, `CREATE USER`, `CREATE SERVER`, and `INSTALL PLUGIN` statements. This memory is not released by the corresponding `REVOKE`, `DROP USER`, `DROP SERVER`, and `UNINSTALL PLUGIN` statements, so for a server that executes many instances of the statements that cause caching, there will be an increase in memory use. This cached memory can be freed with `FLUSH PRIVILEGES`.
- In a replication topology, the following settings affect memory usage, and can be adjusted as required:
  - The `max_allowed_packet` system variable on a replication master limits the maximum message size that the master sends to its slaves for processing. This setting defaults to 64M.
  - The `slave_pending_jobs_size_max` system variable on a multi-threaded slave sets the maximum amount of memory that is made available for holding messages awaiting processing. This setting defaults to 128M. The memory is only allocated when needed, but it might be used if your replication topology handles large transactions sometimes. It is a soft limit, and larger transactions can be processed.
  - The `rpl_read_size` system variable on a replication master or slave controls the minimum amount of data in bytes that is read from the binary log files and relay log files. The default is 8192 bytes. A buffer the size of this value is allocated for each thread that reads from the binary log and relay log files, including dump threads on masters and coordinator threads on slaves.
  - The `binlog_transaction_dependency_history_size` system variable limits the number of row hashes held as an in-memory history.
  - The `max_binlog_cache_size` system variable specifies the upper limit of memory usage by an individual transaction.
  - The `max_binlog_stmt_cache_size` system variable specifies the upper limit of memory usage by the statement cache.

`ps` and other system status programs may report that `mysqld` uses a lot of memory. This may be caused by thread stacks on different memory addresses. For example, the Solaris version of `ps` counts the unused

memory between stacks as used memory. To verify this, check available swap with `swap -s`. We test `mysqld` with several memory-leakage detectors (both commercial and Open Source), so there should be no memory leaks.

Monitoring MySQL Memory Usage

The following example demonstrates how to use Performance Schema and sys schema to monitor MySQL memory usage.

Most Performance Schema memory instrumentation is disabled by default. Instruments can be enabled by updating the `ENABLED` column of the Performance Schema `setup_instruments` table. Memory instruments have names in the form of `memory/code_area/instrument_name`, where *`code_area`* is a value such as `sql` or `innodb`, and *`instrument_name`* is the instrument detail.

1. To view available MySQL memory instruments, query the Performance Schema `setup_instruments` table. The following query returns hundreds of memory instruments for all code areas.

```
mysql> SELECT * FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%memory%';
```

   You can narrow results by specifying a code area. For example, you can limit results to `InnoDB` memory instruments by specifying `innodb` as the code area.

```
mysql> SELECT * FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%memory/innodb%';
+----------------------------------------+---------+-------+
| NAME                                   | ENABLED | TIMED |
+----------------------------------------+---------+-------+
| memory/innodb/adaptive hash index      | NO      | NO    |
| memory/innodb/buf_buf_pool             | NO      | NO    |
| memory/innodb/dict_stats_bg_recalc_pool_t | NO      | NO    |
| memory/innodb/dict_stats_index_map_t   | NO      | NO    |
| memory/innodb/dict_stats_n_diff_on_level | NO      | NO    |
| memory/innodb/other                    | NO      | NO    |
| memory/innodb/row_log_buf              | NO      | NO    |
| memory/innodb/row_merge_sort           | NO      | NO    |
| memory/innodb/std                      | NO      | NO    |
| memory/innodb/trx_sys_t::rw_trx_ids    | NO      | NO    |
...
```

   Depending on your MySQL installation, code areas may include `performance_schema`, `sql`, `client`, `innodb`, `myisam`, `csv`, `memory`, `blackhole`, `archive`, `partition`, and others.

2. To enable memory instruments, add a `performance-schema-instrument` rule to your MySQL configuration file. For example, to enable all memory instruments, add this rule to your configuration file and restart the server:

```
performance-schema-instrument='memory/%=COUNTED'
```

   Note

   Enabling memory instruments at startup ensures that memory allocations that occur at startup are counted.

   After restarting the server, the `ENABLED` column of the Performance Schema `setup_instruments` table should report `YES` for memory instruments that you enabled. The `TIMED` column in the `setup_instruments` table is ignored for memory instruments because memory operations are not timed.

```
mysql> SELECT * FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%memory/innodb%';
+----------------------------------------+---------+-------+
| NAME                                   | ENABLED | TIMED |
+----------------------------------------+---------+-------+
| memory/innodb/adaptive hash index      | NO      | NO    |
| memory/innodb/buf_buf_pool             | NO      | NO    |
| memory/innodb/dict_stats_bg_recalc_pool_t | NO      | NO    |
| memory/innodb/dict_stats_index_map_t   | NO      | NO    |
| memory/innodb/dict_stats_n_diff_on_level | NO      | NO    |
| memory/innodb/other                    | NO      | NO    |
| memory/innodb/row_log_buf              | NO      | NO    |
| memory/innodb/row_merge_sort           | NO      | NO    |
| memory/innodb/std                      | NO      | NO    |
| memory/innodb/trx_sys_t::rw_trx_ids    | NO      | NO    |
...
```

3. Query memory instrument data. In this example, memory instrument data is queried in the Performance Schema `memory_summary_global_by_event_name` table, which summarizes data by `EVENT_NAME`. The `EVENT_NAME` is the name of the instrument.

   The following query returns memory data for the `InnoDB` buffer pool. For column descriptions, see Section 26.12.17.10, “Memory Summary Tables”.

   ```
   mysql> SELECT * FROM performance_schema.memory_summary_global_by_event_name
          WHERE EVENT_NAME LIKE 'memory/innodb/buf_buf_pool'\G
                     EVENT_NAME: memory/innodb/buf_buf_pool
                    COUNT_ALLOC: 1
                     COUNT_FREE: 0
       SUM_NUMBER_OF_BYTES_ALLOC: 137428992
        SUM_NUMBER_OF_BYTES_FREE: 0
                  LOW_COUNT_USED: 0
              CURRENT_COUNT_USED: 1
                 HIGH_COUNT_USED: 1
        LOW_NUMBER_OF_BYTES_USED: 0
   CURRENT_NUMBER_OF_BYTES_USED: 137428992
       HIGH_NUMBER_OF_BYTES_USED: 137428992
   ```

   The same underlying data can be queried using the `sys` schema `memory_global_by_current_bytes` table, which shows current memory usage within the server globally, broken down by allocation type.

   ```
   mysql> SELECT * FROM sys.memory_global_by_current_bytes
          WHERE event_name LIKE 'memory/innodb/buf_buf_pool'\G
   *************************** 1. row ***************************
          event_name: memory/innodb/buf_buf_pool
       current_count: 1
       current_alloc: 131.06 MiB
   current_avg_alloc: 131.06 MiB
          high_count: 1
          high_alloc: 131.06 MiB
      high_avg_alloc: 131.06 MiB
   ```

   This `sys` schema query aggregates currently allocated memory (`current_alloc`) by code area:

   ```
   mysql> SELECT SUBSTRING_INDEX(event_name,'/',2) AS
          code_area, FORMAT_BYTES(SUM(current_alloc))
          AS current_alloc
          FROM sys.x$memory_global_by_current_bytes
          GROUP BY SUBSTRING_INDEX(event_name,'/',2)
          ORDER BY SUM(current_alloc) DESC;
   +---------------------------+---------------+
   | code_area                 | current_alloc |
   +---------------------------+---------------+
   ```

```
| memory/innodb              | 843.24 MiB    |
| memory/performance_schema | 81.29 MiB     |
| memory/mysys               | 8.20 MiB      |
| memory/sql                 | 2.47 MiB      |
| memory/memory              | 174.01 KiB    |
| memory/myisam              | 46.53 KiB     |
| memory/blackhole           | 512 bytes     |
| memory/federated           | 512 bytes     |
| memory/csv                 | 512 bytes     |
| memory/vio                 | 496 bytes     |
+----------------------------+---------------+
```

Note

Prior to MySQL 8.0.16, `sys.format_bytes()` was used for `FORMAT_BYTES()`.

For more information about `sys` schema, see Chapter 27, *MySQL sys Schema*.

8.12.3.2 Enabling Large Page Support

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and operating system. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

In MySQL, large pages can be used by InnoDB, to allocate memory for its buffer pool and additional memory pool.

Standard use of large pages in MySQL attempts to use the largest size supported, up to 4MB. Under Solaris, a “super large pages” feature enables uses of pages up to 256MB. This feature is available for recent SPARC platforms. It can be enabled or disabled by using the `--super-large-pages` or `--skip-super-large-pages` option.

MySQL also supports the Linux implementation of large page support (which is called HugeTLB in Linux).

Before large pages can be used on Linux, the kernel must be enabled to support them and it is necessary to configure the HugeTLB memory pool. For reference, the HugeTBL API is documented in the `Documentation/vm/hugetlbpage.txt` file of your Linux sources.

The kernel for some recent systems such as Red Hat Enterprise Linux appear to have the large pages feature enabled by default. To check whether this is true for your kernel, use the following command and look for output lines containing “huge”:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

The nonempty command output indicates that large page support is present, but the zero values indicate that no pages are configured for use.

If your kernel needs to be reconfigured to support large pages, consult the `hugetlbpage.txt` file for instructions.

Assuming that your Linux kernel has large page support enabled, configure it for use by MySQL using the following commands. Normally, you put these in an `rc` file or equivalent startup file that is executed

during the system boot sequence, so that the commands execute each time the system starts. The commands should execute early in the boot sequence, before the MySQL server starts. Be sure to change the allocation numbers and the group number as appropriate for your system.

```
# Set the number of pages to be used.
# Each page is normally 2MB, so a value of 20 = 40MB.
# This command actually allocates memory, so this much
# memory must be available.
echo 20 > /proc/sys/vm/nr_hugepages

# Set the group number that is permitted to access this
# memory (102 in this case). The mysql user must be a
# member of this group.
echo 102 > /proc/sys/vm/hugetlb_shm_group

# Increase the amount of shmem permitted per segment
# (12G in this case).
echo 1560281088 > /proc/sys/kernel/shmmax

# Increase total amount of shared memory.  The value
# is the number of pages. At 4KB/page, 4194304 = 16GB.
echo 4194304 > /proc/sys/kernel/shmall
```

For MySQL usage, you normally want the value of `shmmax` to be close to the value of `shmall`.

To verify the large page configuration, check `/proc/meminfo` again as described previously. Now you should see some nonzero values:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      20
HugePages_Free:       20
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

The final step to make use of the `hugetlb_shm_group` is to give the `mysql` user an “unlimited” value for the memlock limit. This can be done either by editing `/etc/security/limits.conf` or by adding the following command to your `mysqld_safe` script:

```
ulimit -l unlimited
```

Adding the `ulimit` command to `mysqld_safe` causes the `root` user to set the memlock limit to `unlimited` before switching to the `mysql` user. (This assumes that `mysqld_safe` is started by `root`.)

Large page support in MySQL is disabled by default. To enable it, start the server with the `--large-pages` option. For example, you can use the following lines in the server `my.cnf` file:

```
[mysqld]
large-pages
```

With this option, `InnoDB` uses large pages automatically for its buffer pool and additional memory pool. If `InnoDB` cannot do this, it falls back to use of traditional memory and writes a warning to the error log: `Warning: Using conventional memory pool`

To verify that large pages are being used, check `/proc/meminfo` again:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      20
```

```
HugePages_Free:       20
HugePages_Rsvd:        2
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

8.12.4 Optimizing Network Use

8.12.4.1 How MySQL Handles Client Connections

This section describes aspects of how the MySQL server manages client connections.

- Network Interfaces and Connection Manager Threads
- Client Connection Thread Management
- Connection Volume Management
- Administrative Connection Management

Network Interfaces and Connection Manager Threads

The server is capable of listening for client connections on multiple network interfaces. Connection manager threads handle client connection requests on the network interfaces that the server listens to:

- On all platforms, one manager thread handles TCP/IP connection requests.
- On Unix, the same manager thread also handles Unix socket file connection requests.
- On Windows, a manager thread handles shared-memory connection requests, and another handles named-pipe connection requests.
- On all platforms, an additional network interface may be enabled to accept administrative TCP/IP connection requests. This interface can use the manager thread that handles “ordinary” TCP/IP requests, or a separate thread.

The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.

Client Connection Thread Management

Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

In this connection thread model, there are as many threads as there are clients currently connected, which has some disadvantages when server workload must scale to handle large numbers of connections. For example, thread creation and disposal becomes expensive. Also, each thread requires server and kernel resources, such as stack space. To accommodate a large number of simultaneous connections, the stack size per thread must be kept small, leading to a situation where it is either too small or the server consumes large amounts of memory. Exhaustion of other resources can occur as well, and scheduling overhead can become significant.

MySQL Enterprise Edition includes a thread pool plugin that provides an alternative thread-handling model designed to reduce overhead and improve performance. It implements a thread pool that increases server

performance by efficiently managing statement execution threads for large numbers of client connections. See Section 5.6.3, “MySQL Enterprise Thread Pool”.

To control and monitor how the server manages threads that handle client connections, several system and status variables are relevant. (See Section 5.1.8, “Server System Variables”, and Section 5.1.10, “Server Status Variables”.)

The `thread_cache_size` system variable determines the thread cache size. By default, the server autosizes the value at startup, but it can be set explicitly to override this default. A value of 0 disables caching, which causes a thread to be set up for each new connection and disposed of when the connection terminates. To enable *`N`* inactive connection threads to be cached, set `thread_cache_size` to *`N`* at server startup or at runtime. A connection thread becomes inactive when the client connection with which it was associated terminates.

To monitor the number of threads in the cache and how many threads have been created because a thread could not be taken from the cache, check the `Threads_cached` and `Threads_created` status variables.

When the thread stack is too small, this limits the complexity of the SQL statements which the server can handle, the recursion depth of stored procedures, and other memory-consuming actions. To set a stack size of *`N`* bytes for each thread, start the server with `thread_stack`. set to *`N`* at server startup.

Connection Volume Management

To control the maximum number of clients the server permits to connect simultaneously, set the `max_connections` system variable at server startup or at runtime. It may be necessary to increase `max_connections` if more clients attempt to connect simultaneously then the server is configured to handle (see Section B.4.2.6, “Too many connections”).

`mysqld` actually permits `max_connections` + 1 client connections. The extra connection is reserved for use by accounts that have the `CONNECTION_ADMIN` or `SUPER` privilege. By granting the privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See Section 13.7.7.29, “SHOW PROCESSLIST Syntax”. (The server also permits administrative connections on a dedicated interface. See Administrative Connection Management.)

If the server refuses a connection because the `max_connections` limit is reached, it increments the `Connection_errors_max_connections` status variable.

The maximum number of connections MySQL supports (that is, the maximum value to which `max_connections` can be set) depends on several factors:

- The quality of the thread library on a given platform.
- The amount of RAM available.
- The amount of RAM is used for each connection.
- The workload from each connection.
- The desired response time.
- The number of file descriptors available.

Linux or Solaris should be able to support at least 500 to 1000 simultaneous connections routinely and as many as 10,000 connections if you have many gigabytes of RAM available and the workload from each is low or the response time target undemanding.

Increasing the `max_connections` value increases the number of file descriptors that `mysqld` requires. If the required number of descriptors are not available, the server reduces the value of `max_connections`. For comments on file descriptor limits, see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.

Increasing the `open_files_limit` system variable may be necessary, which may also require raising the operating system limit on how many file descriptors can be used by MySQL. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so. See also Section B.4.2.17, “File Not Found and Similar Errors”.

Administrative Connection Management

As of MySQL 8.0.14, the server permits a TCP/IP port to be configured specifically for administrative connections. This provides an alternative to the single administrative connection that is permitted on the network interfaces used for ordinary connections even when `max_connections` connections are already established (see Connection Volume Management).

The administrative network interface has these characteristics:

- The interface is available only if the `admin_address` system variable is set at startup to indicate the IP address for the administrative interface. If no `admin_address` value is specified, the server maintains no administrative interface.
- The `admin_port` system variable specifies the interface TCP/IP port number (default 33062).
- There is no limit on the number of administrative connections.
- Connections are permitted only by users who have the `SERVICE_CONNECTION_ADMIN` privilege.

The `create_admin_listener_thread` system variable enables DBAs to choose at startup whether the administrative interface is implemented using the listener thread used for ordinary connections, or has its own separate thread. The default is to implement the administrative interface using the listener thread used for ordinary connections.

8.12.4.2 DNS Lookup Optimization and the Host Cache

The MySQL server maintains a host cache in memory that contains information about clients: IP address, host name, and error information. The Performance Schema `host_cache` table exposes the contents of the host cache so that it can be examined using `SELECT` statements. This may help you diagnose the causes of connection problems. See Section 26.12.18.1, “The host_cache Table”.

Note

The server uses the host cache only for nonlocal TCP connections. It does not use the cache for TCP connections established using a loopback interface address (for example, `127.0.0.1` or `::1`), or for connections established using a Unix socket file, named pipe, or shared memory.

- Host Cache Operation
- Host Cache Configuration

Host Cache Operation

The server uses the host cache for several purposes:

- By caching the results of IP-to-host name lookups, the server avoids doing a Domain Name System (DNS) lookup for each client connection. Instead, for a given host, it needs to perform a lookup only for the first connection from that host.

- The cache contains information about errors that occur during the connection process. Some errors are considered “blocking.” If too many of these occur successively from a given host without a successful connection, the server blocks further connections from that host. The `max_connect_errors` system variable determines the permitted number of successive errors before blocking occurs (see Section B.4.2.5, “Host 'host_name' is blocked”).

For each new client connection, the server uses the client IP address to check whether the client host name is in the host cache. If so, the server refuses or continues to process the connection request depending on whether or not the host is blocked. If the host is not in the cache, the server attempts to resolve the host name. First, it resolves the IP address to a host name and resolves that host name back to an IP address. Then it compares the result to the original IP address to ensure that they are the same. The server stores information about the result of this operation in the host cache. If the cache is full, the least recently used entry is discarded.

The server handles entries in the host cache like this:

1. When the first TCP client connection reaches the server from a given IP address, a new cache entry is created to record the client IP, host name, and client lookup validation flag. Initially, the host name is set to `NULL` and the flag is false. This entry is also used for subsequent client TCP connections from the same originating IP.

2. If the validation flag for the client IP entry is false, the server attempts an IP-to-host name-to-IP DNS resolution. If that is successful, the host name is updated with the resolved host name and the validation flag is set to true. If resolution is unsuccessful, the action taken depends on whether the error is permanent or transient. For permanent failures, the host name remains `NULL` and the validation flag is set to true. For transient failures, the host name and validation flag remain unchanged. (In this case, another DNS resolution attempt occurs the next time a client connects from this IP.)

3. If an error occurs while processing an incoming client connection from a given IP address, the server updates the corresponding error counters in the entry for that IP. For a description of the errors recorded, see Section 26.12.18.1, “The host_cache Table”.

The server performs host name resolution using the `gethostbyaddr()` and `gethostbyname()` system calls.

To unblock blocked hosts, flush the host cache by executing a `FLUSH HOSTS` statement, a `TRUNCATE TABLE` statement that truncates the Performance Schema `host_cache` table, or a `mysqladmin flush-hosts` command. `FLUSH HOSTS` and `mysqladmin flush-hosts` require the `RELOAD` privilege. `TRUNCATE TABLE` requires the `DROP` privilege for the `host_cache` table.

It is possible for a blocked host to become unblocked even without flushing the host cache if activity from other hosts has occurred since the last connection attempt from the blocked host. This can occur because the server discards the least recently used cache entry to make room for a new entry if the cache is full when a connection arrives from a client IP not in the cache. If the discarded entry is for a blocked host, that host becomes unblocked.

Some connection errors are not associated with TCP connections, occur very early in the connection process (even before an IP address is known), or are not specific to any particular IP address (such as out-of-memory conditions). For information about these errors, check the `Connection_errors_xxx` status variables (see Section 5.1.10, “Server Status Variables”).

Host Cache Configuration

The host cache is enabled by default. The `host_cache_size` system variable controls its size, as well as the size of the Performance Schema `host_cache` table that exposes the cache contents. The cache

size can be set at server startup and changed at runtime. For example, to set the size to 100 at startup, put these lines in the server `my.cnf` file:

```
[mysqld]
host_cache_size=200
```

To change the size to 300 at runtime, do this:

```
SET GLOBAL host_cache_size=300;
```

Setting `host_cache_size` to 0, either at server startup or at runtime, disables the host cache. With the cache disabled, the server performs a DNS lookup every time a client connects.

Changing the cache size at runtime causes an implicit `FLUSH HOSTS` operation that clears the host cache, truncates the `host_cache` table, and unblocks any blocked hosts.

Using the `--skip-host-cache` option is similar to setting the `host_cache_size` system variable to 0, but `host_cache_size` is more flexible because it can also be used to resize, enable, and disable the host cache at runtime, not just at server startup. Starting the server with `--skip-host-cache` does not prevent changes to the value of `host_cache_size`, but such changes have no effect and the cache is not re-enabled even if `host_cache_size` is set larger than 0 at runtime.

To disable DNS host name lookups, start the server with the `--skip-name-resolve` option. In this case, the server uses only IP addresses and not host names to match connecting hosts to rows in the MySQL grant tables. Only accounts specified in those tables using IP addresses can be used. (A client may not be able to connect if no account exists that specifies the client IP address.)

If you have a very slow DNS and many hosts, you might be able to improve performance either by disabling DNS lookups with `--skip-name-resolve` or by increasing the value of `host_cache_size` to make the host cache larger.

To disallow TCP/IP connections entirely, start the server with the `--skip-networking` option.

8.12.5 Resource Groups

MySQL supports creation and management of resource groups, and permits assigning threads running within the server to particular groups so that threads execute according to the resources available to the group. Group attributes enable control over its resources, to enable or restrict resource consumption by threads in the group. DBAs can modify these attributes as appropriate for different workloads.

Currently, CPU time is a manageable resource, represented by the concept of “virtual CPU” as a term that includes CPU cores, hyperthreads, hardware threads, and so forth. The server determines at startup how many virtual CPUs are available, and database administrators with appropriate privileges can associate these CPUs with resource groups and assign threads to groups.

For example, to manage execution of batch jobs that need not execute with high priority, a DBA can create a `Batch` resource group, and adjust its priority up or down depending on how busy the server is. (Perhaps batch jobs assigned to the group should run at lower priority during the day and at higher priority during the night.) The DBA can also adjust the set of CPUs available to the group. Groups can be enabled or disabled to control whether threads are assignable to them.

The following sections describe aspects of resource group use in MySQL:

- Resource Group Components
- Resource Group Attributes

- Resource Group Management
- Resource Group Replication
- Resource Group Restrictions

Important

On some platforms or MySQL server configurations, resource groups are unavailable or have limitations. In particular, Linux systems might require a manual step for some installation methods. For details, see Resource Group Restrictions.

Resource Group Components

These capabilities provide the SQL interface for resource group management in MySQL:

- SQL statements enable creating, altering, and dropping resource groups, and enable assigning threads to resource groups. An optimizer hint enables assigning individual statements to resource groups.
- Resource group privileges provide control over which users can perform resource group operations.
- The `INFORMATION_SCHEMA.RESOURCE_GROUPS` table exposes information about resource group definitions and the Performance Schema `threads` table shows the resource group assignment for each thread.
- Status variables provide execution counts for each management SQL statement.

Resource Group Attributes

Resource groups have attributes that define the group. All attributes can be set at group creation time. Some attributes are fixed at creation time; others can be modified any time thereafter.

These attributes are defined at resource group creation time and cannot be modified:

- Each group has a name. Resource group names are identifiers like table and column names, and need not be quoted in SQL statements unless they contain special characters or are reserved words. Group names are not case sensitive and may be up to 64 characters long.
- Each group has a type, which is either `SYSTEM` or `USER`. The resource group type affects the range of priority values assignable to the group, as described later. This attribute together with the differences in permitted priorities enables system threads to be identified so as to protect them from contention for CPU resources against user threads.

  System and user threads correspond to background and foreground threads as listed in the Performance Schema `threads` table.

These attributes are defined at resource group creation time and can be modified any time thereafter:

- The CPU affinity is the set of virtual CPUs the resource group can use. An affinity can be any nonempty subset of the available CPUs. If a group has no affinity, it can use all available CPUs.
- The thread priority is the execution priority for threads assigned to the resource group. Priority values range from -20 (highest priority) to 19 (lowest priority). The default priority is 0, for both system and user groups.

  System groups are permitted a higher priority than user groups, ensuring that user threads never have a higher priority than system threads:

  - For system resource groups, the permitted priority range is -20 to 0.
  - For user resource groups, the permitted priority range is 0 to 19.
- Each group can be enabled or disabled, affording administrators control over thread assignment. Threads can be assigned only to enabled groups.

Resource Group Management

By default, there is one system group and one user group, named `SYS_default` and `USR_default`, respectively. These default groups cannot be dropped and their attributes cannot be modified. Each default group has no CPU affinity and priority 0.

Newly created system and user threads are assigned to the `SYS_default` and `USR_default` groups, respectively.

For user-defined resource groups, all attributes are assigned at group creation time. After a group has been created, its attributes can be modified, with the exception of the name and type attributes.

To create and manage user-defined resource groups, use these SQL statements:

- `CREATE RESOURCE GROUP` creates a new group. See Section 13.7.2.2, “CREATE RESOURCE GROUP Syntax”.
- `ALTER RESOURCE GROUP` modifies an existing group. See Section 13.7.2.1, “ALTER RESOURCE GROUP Syntax”.
- `DROP RESOURCE GROUP` drops an existing group. See Section 13.7.2.3, “DROP RESOURCE GROUP Syntax”.

Those statements require the `RESOURCE_GROUP_ADMIN` privilege.

To manage resource group assignments, use these capabilities:

- `SET RESOURCE GROUP` assigns threads to a group. See Section 13.7.2.4, “SET RESOURCE GROUP Syntax”.
- The `RESOURCE_GROUP` optimizer hint assigns individual statements to a group. See Section 8.9.3, “Optimizer Hints”.

Those operations require the `RESOURCE_GROUP_ADMIN` or `RESOURCE_GROUP_USER` privilege.

Resource group definitions are stored in the `resource_groups` data dictionary table so that groups persist across server restarts. Because `resource_groups` is part of the data dictionary, it is not directly accessible by users. Resource group information is available using the `INFORMATION_SCHEMA.RESOURCE_GROUPS` table, which is implemented as a view on the data dictionary table. See Section 25.22, “The INFORMATION_SCHEMA RESOURCE_GROUPS Table”.

Initially, the `RESOURCE_GROUPS` table has these rows describing the default groups:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS\G
*************************** 1. row ***************************
   RESOURCE_GROUP_NAME: USR_default
   RESOURCE_GROUP_TYPE: USER
RESOURCE_GROUP_ENABLED: 1
              VCPU_IDS: 0-3
       THREAD_PRIORITY: 0
```

```
*************************** 2. row ***************************
   RESOURCE_GROUP_NAME: SYS_default
   RESOURCE_GROUP_TYPE: SYSTEM
RESOURCE_GROUP_ENABLED: 1
              VCPU_IDS: 0-3
       THREAD_PRIORITY: 0
```

The `THREAD_PRIORITY` values are 0, indicating the default priority. The `VCPU_IDS` values show a range comprising all available CPUs. For the default groups, the displayed value varies depending on the system on which the MySQL server runs.

Earlier discussion mentioned a scenario involving a resource group named `Batch` to manage execution of batch jobs that need not execute with high priority. To create such a group, use a statement similar to this:

```
CREATE RESOURCE GROUP Batch
  TYPE = USER
  VCPU = 2-3            -- assumes a system with at least 4 CPUs
  THREAD_PRIORITY = 10;
```

To verify that the resource group was created as expected, check the `RESOURCE_GROUPS` table:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.RESOURCE_GROUPS
       WHERE RESOURCE_GROUP_NAME = 'Batch'\G
*************************** 1. row ***************************
   RESOURCE_GROUP_NAME: Batch
   RESOURCE_GROUP_TYPE: USER
RESOURCE_GROUP_ENABLED: 1
              VCPU_IDS: 2-3
       THREAD_PRIORITY: 10
```

If the `THREAD_PRIORITY` value is 0 rather than 10, check whether your platform or system configuration limits the resource group capability; see Resource Group Restrictions.

To assign a thread to the `Batch` group, do this:

```
SET RESOURCE GROUP Batch FOR thread_id;
```

Thereafter, statements in the named thread execute with `Batch` group resources.

If a session's own current thread should be in the `Batch` group, execute this statement within the session:

```
SET RESOURCE GROUP Batch;
```

Thereafter, statements in the session execute with `Batch` group resources.

To execute a single statement using the `Batch` group, use the `RESOURCE_GROUP` optimizer hint:

```
INSERT /*+ RESOURCE_GROUP(Batch) */ INTO t2 VALUES(2);
```

Threads assigned to the `Batch` group execute with its resources, which can be modified as desired:

- For times when the system is highly loaded, decrease the number of CPUs assigned to the group, lower its priority, or (as shown) both:

  ```
  ALTER RESOURCE GROUP Batch
    VCPU = 3
    THREAD_PRIORITY = 19;
  ```

- For times when the system is lightly loaded, increase the number of CPUs assigned to the group, raise its priority, or (as shown) both:

```
ALTER RESOURCE GROUP Batch
  VCPU = 0-3
  THREAD_PRIORITY = 0;
```

Resource Group Replication

Resource group management is local to the server on which it occurs. Resource group SQL statements and modifications to the `resource_groups` data dictionary table are not written to the binary log and are not replicated.

Resource Group Restrictions

On some platforms or MySQL server configurations, resource groups are unavailable or have limitations:

- Resource groups are unavailable if the thread pool plugin is installed.
- Resource groups are unavailable on macOS, which provides no API for binding CPUs to a thread.
- On FreeBSD and Solaris, resource group thread priorities are ignored. (Effectively, all threads run at priority 0.) Attempts to change priorities result in a warning:

```
mysql> ALTER RESOURCE GROUP abc THREAD_PRIORITY = 10;
Query OK, 0 rows affected, 1 warning (0.18 sec)

mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------+
| Level   | Code | Message                                                         |
+---------+------+-----------------------------------------------------------------+
| Warning | 4560 | Attribute thread_priority is ignored (using default value). |
+---------+------+-----------------------------------------------------------------+
```

- On Linux, resource groups thread priorities are ignored unless the `CAP_SYS_NICE` capability is set. Granting `CAP_SYS_NICE` capability to a process enables a range of privileges; consult http://man7.org/linux/man-pages/man7/capabilities.7.html for the full list. Please be careful when enabling this capability.

  On Linux platforms using systemd and kernel support for Ambient Capabilities (Linux 4.3 or newer), the recommended way to enable `CAP_SYS_NICE` capability is to modify the MySQL service file and leave the `mysqld` binary unmodified. To adjust the service file for MySQL, use this procedure:

  1. Run the appropriate command for your platform:

     - Oracle Linux, Red Hat, and Fedora systems:

       ```
       shell> sudo systemctl edit mysqld
       ```

     - SUSE, Ubuntu, and Debian systems:

       ```
       shell> sudo systemctl edit mysql
       ```

  2. Using an editor, add the following text to the service file:

     ```
     [Service]
     AmbientCapabilities=CAP_SYS_NICE
     ```

3. Restart the MySQL service.

If you cannot enable the `CAP_SYS_NICE` capability as just described, it can be set manually using the `setcap` command, specifying the path name to the `mysqld` executable (this requires `sudo` access). You can check the capabilities using `getcap`. For example:

```
shell> sudo setcap cap_sys_nice+ep /path/to/mysqld
shell> getcap /path/to/mysqld
/path/to/mysqld = cap_sys_nice+ep
```

As a safety measure, restrict execution of the `mysqld` binary to the `root` user and users with `mysql` group membership:

```
shell> sudo chown root:mysql /path/to/mysqld
shell> sudo chmod 0750 /path/to/mysqld
```

Important

If manual use of `setcap` is required, it must be performed after each reinstall.

- On Windows, threads run at one of five thread priority levels. The resource group thread priority range of -20 to 19 maps onto those levels as indicated in the following table.

Table 8.3 Resource Group Thread Priority on Windows

Priority Range	Windows Priority Level
-20 to -10	`THREAD_PRIORITY_HIGHEST`
-9 to -1	`THREAD_PRIORITY_ABOVE_NORMAL`
0	`THREAD_PRIORITY_NORMAL`
1 to 10	`THREAD_PRIORITY_BELOW_NORMAL`
11 to 19	`THREAD_PRIORITY_LOWEST`

8.13 Measuring Performance (Benchmarking)

To measure performance, consider the following factors:

- Whether you are measuring the speed of a single operation on a quiet system, or how a set of operations (a “workload”) works over a period of time. With simple tests, you usually test how changing one aspect (a configuration setting, the set of indexes on a table, the SQL clauses in a query) affects performance. Benchmarks are typically long-running and elaborate performance tests, where the results could dictate high-level choices such as hardware and storage configuration, or how soon to upgrade to a new MySQL version.
- For benchmarking, sometimes you must simulate a heavy database workload to get an accurate picture.
- Performance can vary depending on so many different factors that a difference of a few percentage points might not be a decisive victory. The results might shift the opposite way when you test in a different environment.
- Certain MySQL features help or do not help performance depending on the workload. For completeness, always test performance with those features turned on and turned off. The most important feature to try with each workload is the adaptive hash index for `InnoDB` tables.

This section progresses from simple and direct measurement techniques that a single developer can do, to more complicated ones that require additional expertise to perform and interpret the results.

8.13.1 Measuring the Speed of Expressions and Functions

To measure the speed of a specific MySQL expression or function, invoke the `BENCHMARK()` function using the `mysql` client program. Its syntax is `BENCHMARK(loop_count,expr)`. The return value is always zero, but `mysql` prints a line displaying approximately how long the statement took to execute. For example:

```
mysql> SELECT BENCHMARK(1000000,1+1);
+------------------------+
| BENCHMARK(1000000,1+1) |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.32 sec)
```

This result was obtained on a Pentium II 400MHz system. It shows that MySQL can execute 1,000,000 simple addition expressions in 0.32 seconds on that system.

The built-in MySQL functions are typically highly optimized, but there may be some exceptions. `BENCHMARK()` is an excellent tool for finding out if some function is a problem for your queries.

8.13.2 Using Your Own Benchmarks

Benchmark your application and database to find out where the bottlenecks are. After fixing one bottleneck (or by replacing it with a “dummy” module), you can proceed to identify the next bottleneck. Even if the overall performance for your application currently is acceptable, you should at least make a plan for each bottleneck and decide how to solve it if someday you really need the extra performance.

A free benchmark suite is the Open Source Database Benchmark, available at http://osdb.sourceforge.net/.

It is very common for a problem to occur only when the system is very heavily loaded. We have had many customers who contact us when they have a (tested) system in production and have encountered load problems. In most cases, performance problems turn out to be due to issues of basic database design (for example, table scans are not good under high load) or problems with the operating system or libraries. Most of the time, these problems would be much easier to fix if the systems were not already in production.

To avoid problems like this, benchmark your whole application under the worst possible load:

- The `mysqlslap` program can be helpful for simulating a high load produced by multiple clients issuing queries simultaneously. See Section 4.5.8, “`mysqlslap` — Load Emulation Client”.
- You can also try benchmarking packages such as SysBench and DBT2, available at https://launchpad.net/sysbench, and http://osdldbt.sourceforge.net/#dbt2.

These programs or packages can bring a system to its knees, so be sure to use them only on your development systems.

8.13.3 Measuring Performance with performance_schema

You can query the tables in the `performance_schema` database to see real-time information about the performance characteristics of your server and the applications it is running. See Chapter 26, *MySQL Performance Schema* for details.

8.14 Examining Thread Information

When you are attempting to ascertain what your MySQL server is doing, it can be helpful to examine the process list, which is the set of threads currently executing within the server. Process list information is available from these sources:

- The `SHOW [FULL] PROCESSLIST` statement: Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
- The `SHOW PROFILE` statement: Section 13.7.7.31, “SHOW PROFILES Syntax”
- The `INFORMATION_SCHEMA PROCESSLIST` table: Section 25.19, “The INFORMATION_SCHEMA PROCESSLIST Table”
- The `mysqladmin processlist` command: Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
- The Performance Schema `threads` table, stage tables, and lock tables: Section 26.12.18, “Performance Schema Miscellaneous Tables”, Section 26.12.5, “Performance Schema Stage Event Tables”, Section 26.12.12, “Performance Schema Lock Tables”.

Access to `threads` does not require a mutex and has minimal impact on server performance. `INFORMATION_SCHEMA.PROCESSLIST` and `SHOW PROCESSLIST` have negative performance consequences because they require a mutex. `threads` also shows information about background threads, which `INFORMATION_SCHEMA.PROCESSLIST` and `SHOW PROCESSLIST` do not. This means that `threads` can be used to monitor activity the other thread information sources cannot.

You can always view information about your own threads. To view information about threads being executed for other accounts, you must have the `PROCESS` privilege.

Each process list entry contains several pieces of information:

- `Id` is the connection identifier for the client associated with the thread.
- `User` and `Host` indicate the account associated with the thread.
- `db` is the default database for the thread, or `NULL` if none is selected.
- `Command` and `State` indicate what the thread is doing.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.
- `Time` indicates how long the thread has been in its current state. The thread's notion of the current time may be altered in some cases: The thread can change the time with `SET TIMESTAMP = value`. For a thread running on a slave that is processing events from the master, the thread time is set to the time found in the events and thus reflects current time on the master and not the slave.
- `Info` contains the text of the statement being executed by the thread, or `NULL` if it is not executing one. By default, this value contains only the first 100 characters of the statement. To see the complete statements, use `SHOW FULL PROCESSLIST`.
- The `sys` schema `processlist` view, which presents information from the Performance Schema `threads` table in a more accessible format: Section 27.4.3.22, “The processlist and x$processlist Views”
- The `sys` schema `session` view, which presents information about user sessions (like the `sys` schema `processlist` view, but with background processes filtered out): Section 27.4.3.33, “The session and x$session Views”

The following sections list the possible `Command` values, and `State` values grouped by category. The meaning for some of these values is self-evident. For others, additional description is provided.

8.14.1 Thread Command Values

A thread can have any of the following `Command` values:

- `Binlog Dump`

  This is a thread on a master server for sending binary log contents to a slave server.

- `Change user`

  The thread is executing a change-user operation.

- `Close stmt`

  The thread is closing a prepared statement.

- `Connect`

  A replication slave is connected to its master.

- `Connect Out`

  A replication slave is connecting to its master.

- `Create DB`

  The thread is executing a create-database operation.

- `Daemon`

  This thread is internal to the server, not a thread that services a client connection.

- `Debug`

  The thread is generating debugging information.

- `Delayed insert`

  The thread is a delayed-insert handler.

- `Drop DB`

  The thread is executing a drop-database operation.

- `Error`

- `Execute`

  The thread is executing a prepared statement.

- `Fetch`

  The thread is fetching the results from executing a prepared statement.

- `Field List`

The thread is retrieving information for table columns.

- `Init DB`

  The thread is selecting a default database.

- `Kill`

  The thread is killing another thread.

- `Long Data`

  The thread is retrieving long data in the result of executing a prepared statement.

- `Ping`

  The thread is handling a server-ping request.

- `Prepare`

  The thread is preparing a prepared statement.

- `Processlist`

  The thread is producing information about server threads.

- `Query`

  The thread is executing a statement.

- `Quit`

  The thread is terminating.

- `Refresh`

  The thread is flushing table, logs, or caches, or resetting status variable or replication server information.

- `Register Slave`

  The thread is registering a slave server.

- `Reset stmt`

  The thread is resetting a prepared statement.

- `Set option`

  The thread is setting or resetting a client statement-execution option.

- `Shutdown`

  The thread is shutting down the server.

- `Sleep`

  The thread is waiting for the client to send a new statement to it.

- `Statistics`

The thread is producing server-status information.

- `Table Dump`

  The thread is sending table contents to a slave server.

- `Time`

  Unused.

8.14.2 General Thread States

The following list describes thread `State` values that are associated with general query processing and not more specialized activities such as replication. Many of these are useful only for finding bugs in the server.

- `After create`

  This occurs when the thread creates a table (including internal temporary tables), at the end of the function that creates the table. This state is used even if the table could not be created due to some error.

- `Analyzing`

  The thread is calculating a `MyISAM` table key distributions (for example, for `ANALYZE TABLE`).

- `checking permissions`

  The thread is checking whether the server has the required privileges to execute the statement.

- `Checking table`

  The thread is performing a table check operation.

- `cleaning up`

  The thread has processed one command and is preparing to free memory and reset certain state variables.

- `closing tables`

  The thread is flushing the changed table data to disk and closing the used tables. This should be a fast operation. If not, verify that you do not have a full disk and that the disk is not in very heavy use.

- `converting HEAP to ondisk`

  The thread is converting an internal temporary table from a `MEMORY` table to an on-disk table.

- `copy to tmp table`

  The thread is processing an `ALTER TABLE` statement. This state occurs after the table with the new structure has been created but before rows are copied into it.

  For a thread in this state, the Performance Schema can be used to obtain about the progress of the copy operation. See Section 26.12.5, “Performance Schema Stage Event Tables”.

- `Copying to group table`

If a statement has different `ORDER BY` and `GROUP BY` criteria, the rows are sorted by group and copied to a temporary table.

- `Copying to tmp table`

  The server is copying to a temporary table in memory.

- `altering table`

  The server is in the process of executing an in-place `ALTER TABLE`.

- `Copying to tmp table on disk`

  The server is copying to a temporary table on disk. The temporary result set has become too large (see Section 8.4.4, “Internal Temporary Table Use in MySQL”). Consequently, the thread is changing the temporary table from in-memory to disk-based format to save memory.

- `Creating index`

  The thread is processing `ALTER TABLE ... ENABLE KEYS` for a `MyISAM` table.

- `Creating sort index`

  The thread is processing a `SELECT` that is resolved using an internal temporary table.

- `creating table`

  The thread is creating a table. This includes creation of temporary tables.

- `Creating tmp table`

  The thread is creating a temporary table in memory or on disk. If the table is created in memory but later is converted to an on-disk table, the state during that operation will be `Copying to tmp table on disk`.

- `committing alter table to storage engine`

  The server has finished an in-place `ALTER TABLE` and is committing the result.

- `deleting from main table`

  The server is executing the first part of a multiple-table delete. It is deleting only from the first table, and saving columns and offsets to be used for deleting from the other (reference) tables.

- `deleting from reference tables`

  The server is executing the second part of a multiple-table delete and deleting the matched rows from the other tables.

- `discard_or_import_tablespace`

  The thread is processing an `ALTER TABLE ... DISCARD TABLESPACE` or `ALTER TABLE ... IMPORT TABLESPACE` statement.

- `end`

  This occurs at the end but before the cleanup of `ALTER TABLE`, `CREATE VIEW`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements.

- `executing`

  The thread has begun executing a statement.

- `Execution of init_command`

  The thread is executing statements in the value of the `init_command` system variable.

- `freeing items`

  The thread has executed a command. This state is usually followed by `cleaning up`.

- `FULLTEXT initialization`

  The server is preparing to perform a natural-language full-text search.

- `init`

  This occurs before the initialization of `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements. Actions taken by the server in this state include flushing the binary log and the `InnoDB` log.

  For the `end` state, the following operations could be happening:

  - Writing an event to the binary log
  - Freeing memory buffers, including for blobs

- `Killed`

  Someone has sent a `KILL` statement to the thread and it should abort next time it checks the kill flag. The flag is checked in each major loop in MySQL, but in some cases it might still take a short time for the thread to die. If the thread is locked by some other thread, the kill takes effect as soon as the other thread releases its lock.

- `Locking system tables`

  The thread is trying to lock a system table (for example, a time zone or log table).

- `logging slow query`

  The thread is writing a statement to the slow-query log.

- `login`

  The initial state for a connection thread until the client has been authenticated successfully.

- `manage keys`

  The server is enabling or disabling a table index.

- `NULL`

  This state is used for the `SHOW PROCESSLIST` state.

- `Opening system tables`

  The thread is trying to open a system table (for example, a time zone or log table).

- `Opening tables`

The thread is trying to open a table. This is should be very fast procedure, unless something prevents opening. For example, an `ALTER TABLE` or a `LOCK TABLE` statement can prevent opening a table until the statement is finished. It is also worth checking that your `table_open_cache` value is large enough.

For system tables, the `Opening system tables` state is used instead.

- `optimizing`

  The server is performing initial optimizations for a query.

- `preparing`

  This state occurs during query optimization.

- `Purging old relay logs`

  The thread is removing unneeded relay log files.

- `query end`

  This state occurs after processing a query but before the `freeing items` state.

- `Receiving from client`

  The server is reading a packet from the client.

- `Removing duplicates`

  The query was using `SELECT DISTINCT` in such a way that MySQL could not optimize away the distinct operation at an early stage. Because of this, MySQL requires an extra stage to remove all duplicated rows before sending the result to the client.

- `removing tmp table`

  The thread is removing an internal temporary table after processing a `SELECT` statement. This state is not used if no temporary table was created.

- `rename`

  The thread is renaming a table.

- `rename result table`

  The thread is processing an `ALTER TABLE` statement, has created the new table, and is renaming it to replace the original table.

- `Reopen tables`

  The thread got a lock for the table, but noticed after getting the lock that the underlying table structure changed. It has freed the lock, closed the table, and is trying to reopen it.

- `Repair by sorting`

  The repair code is using a sort to create indexes.

- `preparing for alter table`

  The server is preparing to execute an in-place `ALTER TABLE`.

- `Repair done`

  The thread has completed a multithreaded repair for a `MyISAM` table.

- `Repair with keycache`

  The repair code is using creating keys one by one through the key cache. This is much slower than `Repair by sorting`.

- `Rolling back`

  The thread is rolling back a transaction.

- `Saving state`

  For `MyISAM` table operations such as repair or analysis, the thread is saving the new table state to the `.MYI` file header. State includes information such as number of rows, the `AUTO_INCREMENT` counter, and key distributions.

- `Searching rows for update`

  The thread is doing a first phase to find all matching rows before updating them. This has to be done if the `UPDATE` is changing the index that is used to find the involved rows.

- `Sending data`

  The thread is reading and processing rows for a `SELECT` statement, and sending data to the client. Because operations occurring during this state tend to perform large amounts of disk access (reads), it is often the longest-running state over the lifetime of a given query.

- `Sending to client`

  The server is writing a packet to the client.

- `setup`

  The thread is beginning an `ALTER TABLE` operation.

- `Sorting for group`

  The thread is doing a sort to satisfy a `GROUP BY`.

- `Sorting for order`

  The thread is doing a sort to satisfy an `ORDER BY`.

- `Sorting index`

  The thread is sorting index pages for more efficient access during a `MyISAM` table optimization operation.

- `Sorting result`

  For a `SELECT` statement, this is similar to `Creating sort index`, but for nontemporary tables.

- `statistics`

  The server is calculating statistics to develop a query execution plan. If a thread is in this state for a long time, the server is probably disk-bound performing other work.

- `System lock`

  The thread has called `mysql_lock_tables()` and the thread state has not been updated since. This is a very general state that can occur for many reasons.

  For example, the thread is going to request or is waiting for an internal or external system lock for the table. This can occur when `InnoDB` waits for a table-level lock during execution of `LOCK TABLES`. If this state is being caused by requests for external locks and you are not using multiple `mysqld` servers that are accessing the same `MyISAM` tables, you can disable external system locks with the `--skip-external-locking` option. However, external locking is disabled by default, so it is likely that this option will have no effect. For `SHOW PROFILE`, this state means the thread is requesting the lock (not waiting for it).

  For system tables, the `Locking system tables` state is used instead.

- `update`

  The thread is getting ready to start updating the table.

- `Updating`

  The thread is searching for rows to update and is updating them.

- `updating main table`

  The server is executing the first part of a multiple-table update. It is updating only the first table, and saving columns and offsets to be used for updating the other (reference) tables.

- `updating reference tables`

  The server is executing the second part of a multiple-table update and updating the matched rows from the other tables.

- `User lock`

  The thread is going to request or is waiting for an advisory lock requested with a `GET_LOCK()` call. For `SHOW PROFILE`, this state means the thread is requesting the lock (not waiting for it).

- `User sleep`

  The thread has invoked a `SLEEP()` call.

- `Waiting for commit lock`

  `FLUSH TABLES WITH READ LOCK` is waiting for a commit lock.

- `Waiting for handler commit`

  The thread is waiting for a transaction to commit versus other parts of query processing.

- `Waiting for tables`

  The thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to reopen the table, it must wait until all other threads have closed the table in question.

  This notification takes place if another thread has used `FLUSH TABLES` or one of the following statements on the table in question: `FLUSH TABLES` *`tbl_name`*, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, or `OPTIMIZE TABLE`.

- `Waiting for table flush`

  The thread is executing `FLUSH TABLES` and is waiting for all threads to close their tables, or the thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to reopen the table, it must wait until all other threads have closed the table in question.

  This notification takes place if another thread has used `FLUSH TABLES` or one of the following statements on the table in question: `FLUSH TABLES` *`tbl_name`*, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, or `OPTIMIZE TABLE`.

- `Waiting for` *`lock_type`* `lock`

  The server is waiting to acquire a `THR_LOCK` lock or a lock from the metadata locking subsystem, where *`lock_type`* indicates the type of lock.

  This state indicates a wait for a `THR_LOCK`:

  - `Waiting for table level lock`

  These states indicate a wait for a metadata lock:

  - `Waiting for event metadata lock`
  - `Waiting for global read lock`
  - `Waiting for schema metadata lock`
  - `Waiting for stored function metadata lock`
  - `Waiting for stored procedure metadata lock`
  - `Waiting for table metadata lock`
  - `Waiting for trigger metadata lock`

  For information about table lock indicators, see Section 8.11.1, “Internal Locking Methods”. For information about metadata locking, see Section 8.11.4, “Metadata Locking”. To see which locks are blocking lock requests, use the Performance Schema lock tables described at Section 26.12.12, “Performance Schema Lock Tables”.

- `Waiting on cond`

  A generic state in which the thread is waiting for a condition to become true. No specific state information is available.

- `Writing to net`

  The server is writing a packet to the network.

8.14.3 Replication Master Thread States

The following list shows the most common states you may see in the `State` column for the master's `Binlog Dump` thread. If you see no `Binlog Dump` threads on a master server, this means that replication is not running; that is, that no slaves are currently connected.

- `Finished reading one binlog; switching to next binlog`

The thread has finished reading a binary log file and is opening the next one to send to the slave.

- `Master has sent all binlog to slave; waiting for more updates`

  The thread has read all remaining updates from the binary logs and sent them to the slave. The thread is now idle, waiting for new events to appear in the binary log resulting from new updates occurring on the master.

- `Sending binlog event to slave`

  Binary logs consist of *events*, where an event is usually an update plus some other information. The thread has read an event from the binary log and is now sending it to the slave.

- `Waiting to finalize termination`

  A very brief state that occurs as the thread is stopping.

8.14.4 Replication Slave I/O Thread States

The following list shows the most common states you see in the `State` column for a slave server I/O thread. This state also appears in the `Slave_IO_State` column displayed by `SHOW SLAVE STATUS`, so you can get a good view of what is happening by using that statement.

- `Checking master version`

  A state that occurs very briefly, after the connection to the master is established.

- `Connecting to master`

  The thread is attempting to connect to the master.

- `Queueing master event to the relay log`

  The thread has read an event and is copying it to the relay log so that the SQL thread can process it.

- `Reconnecting after a failed binlog dump request`

  The thread is trying to reconnect to the master.

- `Reconnecting after a failed master event read`

  The thread is trying to reconnect to the master. When connection is established again, the state becomes `Waiting for master to send event`.

- `Registering slave on master`

  A state that occurs very briefly after the connection to the master is established.

- `Requesting binlog dump`

  A state that occurs very briefly, after the connection to the master is established. The thread sends to the master a request for the contents of its binary logs, starting from the requested binary log file name and position.

- `Waiting for its turn to commit`

  A state that occurs when the slave thread is waiting for older worker threads to commit if `slave_preserve_commit_order` is enabled.

- `Waiting for master to send event`

  The thread has connected to the master and is waiting for binary log events to arrive. This can last for a long time if the master is idle. If the wait lasts for `slave_net_timeout` seconds, a timeout occurs. At that point, the thread considers the connection to be broken and makes an attempt to reconnect.

- `Waiting for master update`

  The initial state before `Connecting to master`.

- `Waiting for slave mutex on exit`

  A state that occurs briefly as the thread is stopping.

- `Waiting for the slave SQL thread to free enough relay log space`

  You are using a nonzero `relay_log_space_limit` value, and the relay logs have grown large enough that their combined size exceeds this value. The I/O thread is waiting until the SQL thread frees enough space by processing relay log contents so that it can delete some relay log files.

- `Waiting to reconnect after a failed binlog dump request`

  If the binary log dump request failed (due to disconnection), the thread goes into this state while it sleeps, then tries to reconnect periodically. The interval between retries can be specified using the `CHANGE MASTER TO` statement.

- `Waiting to reconnect after a failed master event read`

  An error occurred while reading (due to disconnection). The thread is sleeping for the number of seconds set by the `CHANGE MASTER TO` statement (default 60) before attempting to reconnect.

8.14.5 Replication Slave SQL Thread States

The following list shows the most common states you may see in the `State` column for a slave server SQL thread:

- `Killing slave`

  The thread is processing a `STOP SLAVE` statement.

- `Making temporary file (append) before replaying LOAD DATA INFILE`

  The thread is executing a `LOAD DATA` statement and is appending the data to a temporary file containing the data from which the slave will read rows.

- `Making temporary file (create) before replaying LOAD DATA INFILE`

  The thread is executing a `LOAD DATA` statement and is creating a temporary file containing the data from which the slave will read rows. This state can only be encountered if the original `LOAD DATA` statement was logged by a master running a version of MySQL lower than MySQL 5.0.3.

- `Reading event from the relay log`

  The thread has read an event from the relay log so that the event can be processed.

- `Slave has read all relay log; waiting for more updates`

  The thread has processed all events in the relay log files, and is now waiting for the I/O thread to write new events to the relay log.

- `Waiting for an event from Coordinator`

  Using the multithreaded slave (`slave_parallel_workers` is greater than 1), one of the slave worker threads is waiting for an event from the coordinator thread.

- `Waiting for slave mutex on exit`

  A very brief state that occurs as the thread is stopping.

- `Waiting for Slave Workers to free pending events`

  This waiting action occurs when the total size of events being processed by Workers exceeds the size of the `slave_pending_jobs_size_max` system variable. The Coordinator resumes scheduling when the size drops below this limit. This state occurs only when `slave_parallel_workers` is set greater than 0.

- `Waiting for the next event in relay log`

  The initial state before `Reading event from the relay log`.

- `Waiting until MASTER_DELAY seconds after master executed event`

  The SQL thread has read an event but is waiting for the slave delay to lapse. This delay is set with the `MASTER_DELAY` option of `CHANGE MASTER TO`.

The `Info` column for the SQL thread may also show the text of a statement. This indicates that the thread has read an event from the relay log, extracted the statement from it, and may be executing it.

8.14.6 Replication Slave Connection Thread States

These thread states occur on a replication slave but are associated with connection threads, not with the I/O or SQL threads.

- `Changing master`

  The thread is processing a `CHANGE MASTER TO` statement.

- `Killing slave`

  The thread is processing a `STOP SLAVE` statement.

- `Opening master dump table`

  This state occurs after `Creating table from master dump`.

- `Reading master dump table data`

  This state occurs after `Opening master dump table`.

- `Rebuilding the index on master dump table`

  This state occurs after `Reading master dump table data`.

8.14.7 NDB Cluster Thread States

- `Committing events to binlog`

- `Opening mysql.ndb_apply_status`

- `Processing events`

  The thread is processing events for binary logging.

- `Processing events from schema table`

  The thread is doing the work of schema replication.

- `Shutting down`

- `Syncing ndb table schema operation and binlog`

  This is used to have a correct binary log of schema operations for NDB.

- `Waiting for allowed to take ndbcluster global schema lock`

  The thread is waiting for permission to take a global schema lock.

- `Waiting for event from ndbcluster`

  The server is acting as an SQL node in an NDB Cluster, and is connected to a cluster management node.

- `Waiting for first event from ndbcluster`

- `Waiting for ndbcluster binlog update to reach current position`

- `Waiting for ndbcluster global schema lock`

  The thread is waiting for a global schema lock held by another thread to be released.

- `Waiting for ndbcluster to start`

- `Waiting for schema epoch`

  The thread is waiting for a schema epoch (that is, a global checkpoint).

8.14.8 Event Scheduler Thread States

These states occur for the Event Scheduler thread, threads that are created to execute scheduled events, or threads that terminate the scheduler.

- `Clearing`

  The scheduler thread or a thread that was executing an event is terminating and is about to end.

- `Initialized`

  The scheduler thread or a thread that will execute an event has been initialized.

- `Waiting for next activation`

  The scheduler has a nonempty event queue but the next activation is in the future.

- `Waiting for scheduler to stop`

  The thread issued `SET GLOBAL event_scheduler=OFF` and is waiting for the scheduler to stop.

- `Waiting on empty queue`

The scheduler's event queue is empty and it is sleeping.

Chapter 9 Language Structure

Table of Contents

9.1 Literal Values 1669
9.1.1 String Literals 1669
9.1.2 Numeric Literals 1672
9.1.3 Date and Time Literals 1672
9.1.4 Hexadecimal Literals 1675
9.1.5 Bit-Value Literals 1677
9.1.6 Boolean Literals 1679
9.1.7 NULL Values 1679
9.2 Schema Object Names 1679
9.2.1 Identifier Length Limits 1681
9.2.2 Identifier Qualifiers 1682
9.2.3 Identifier Case Sensitivity 1684
9.2.4 Mapping of Identifiers to File Names 1686
9.2.5 Function Name Parsing and Resolution 1687
9.3 Keywords and Reserved Words 1691
9.4 User-Defined Variables 1720
9.5 Expressions 1723
9.6 Comment Syntax 1727

This chapter discusses the rules for writing the following elements of SQL statements when using MySQL:

- Literal values such as strings and numbers
- Identifiers such as database, table, and column names
- Keywords and reserved words
- User-defined and system variables
- Comments

9.1 Literal Values

This section describes how to write literal values in MySQL. These include strings, numbers, hexadecimal and bit values, boolean values, and `NULL`. The section also covers various nuances that you may encounter when dealing with these basic types in MySQL.

9.1.1 String Literals

A string is a sequence of bytes or characters, enclosed within either single quote (`'`) or double quote (`"`) characters. Examples:

```
'a string'
"another string"
```

Quoted strings placed next to each other are concatenated to a single string. The following lines are equivalent:

```
'a string'
'a' ' ' 'string'
```

If the ANSI_QUOTES SQL mode is enabled, string literals can be quoted only within single quotation marks because a string quoted within double quotation marks is interpreted as an identifier.

A *binary string* is a string of bytes. Every binary string has a character set and collation named binary. A *nonbinary string* is a string of characters. It has a character set other than binary and a collation that is compatible with the character set.

For both types of strings, comparisons are based on the numeric values of the string unit. For binary strings, the unit is the byte; comparisons use numeric byte values. For nonbinary strings, the unit is the character and some character sets support multibyte characters; comparisons use numeric character code values. Character code ordering is a function of the string collation. (For more information, see Section 10.8.5, “The binary Collation Compared to _bin Collations”.)

A character string literal may have an optional character set introducer and COLLATE clause, to designate it as a string that uses a particular character set and collation:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1'string';
SELECT _binary'string';
SELECT _utf8'string' COLLATE utf8_danish_ci;
```

You can use N'literal' (or n'literal') to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For information about these forms of string syntax, see Section 10.3.7, “The National Character Set”, and Section 10.3.8, “Character Set Introducers”.

Within a string, certain sequences have special meaning unless the NO_BACKSLASH_ESCAPES SQL mode is enabled. Each of these sequences begins with a backslash (\), known as the *escape character*. MySQL recognizes the escape sequences shown in Table 9.1, “Special Character Escape Sequences”. For all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped. For example, \x is just x. These sequences are case-sensitive. For example, \b is interpreted as a backspace, but \B is interpreted as B. Escape processing is done according to the character set indicated by the character_set_connection system variable. This is true even for strings that are preceded by an introducer that indicates a different character set, as discussed in Section 10.3.6, “Character String Literal Character Set and Collation”.

Table 9.1 Special Character Escape Sequences

Escape Sequence	Character Represented by Sequence
\0	An ASCII NUL (X'00') character

Escape Sequence	Character Represented by Sequence
`\'`	A single quote (`'`) character
`\"`	A double quote (`"`) character
`\b`	A backspace character
`\n`	A newline (linefeed) character
`\r`	A carriage return character
`\t`	A tab character
`\Z`	ASCII 26 (Control+Z); see note following the table
`\\`	A backslash (`\`) character
`\%`	A `%` character; see note following the table
`\_`	A `_` character; see note following the table

The ASCII 26 character can be encoded as `\Z` to enable you to work around the problem that ASCII 26 stands for END-OF-FILE on Windows. ASCII 26 within a file causes problems if you try to use `mysql db_name < file_name`.

The `\%` and `\_` sequences are used to search for literal instances of `%` and `_` in pattern-matching contexts where they would otherwise be interpreted as wildcard characters. See the description of the `LIKE` operator in Section 12.5.1, “String Comparison Functions and Operators”. If you use `\%` or `\_` outside of pattern-matching contexts, they evaluate to the strings `\%` and `\_`, not to `%` and `_`.

There are several ways to include quote characters within a string:

- A `'` inside a string quoted with `'` may be written as `''`.
- A `"` inside a string quoted with `"` may be written as `""`.
- Precede the quote character by an escape character (`\`).
- A `'` inside a string quoted with `"` needs no special treatment and need not be doubled or escaped. In the same way, `"` inside a string quoted with `'` needs no special treatment.

The following `SELECT` statements demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";
+-------+---------+-----------+--------+--------+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT 'This\nIs\nFour\nLines';
+--------------------+
| This
Is
Four
Lines |
+--------------------+

mysql> SELECT 'disappearing\ backslash';
+------------------------+
```

```
| disappearing backslash |
+------------------------+
```

To insert binary data into a string column (such as a `BLOB` column), you should represent certain characters by escape sequences. Backslash (`\`) and the quote character used to quote the string must be escaped. In certain client environments, it may also be necessary to escape `NUL` or Control+Z. The `mysql` client truncates quoted strings containing `NUL` characters if they are not escaped, and Control+Z may be taken for END-OF-FILE on Windows if not escaped. For the escape sequences that represent each of these characters, see Table 9.1, “Special Character Escape Sequences”.

When writing application programs, any string that might contain any of these special characters must be properly escaped before the string is used as a data value in an SQL statement that is sent to the MySQL server. You can do this in two ways:

- Process the string with a function that escapes the special characters. In a C program, you can use the `mysql_real_escape_string_quote()` C API function to escape characters. See Section 28.7.7.56, “mysql_real_escape_string_quote()”. Within SQL statements that construct other SQL statements, you can use the `QUOTE()` function. The Perl DBI interface provides a `quote` method to convert special characters to the proper escape sequences. See Section 28.9, “MySQL Perl API”. Other language interfaces may provide a similar capability.
- As an alternative to explicitly escaping special characters, many MySQL APIs provide a placeholder capability that enables you to insert special markers into a statement string, and then bind data values to them when you issue the statement. In this case, the API takes care of escaping special characters in the values for you.

9.1.2 Numeric Literals

Number literals include exact-value (integer and `DECIMAL`) literals and approximate-value (floating-point) literals.

Integers are represented as a sequence of digits. Numbers may include `.` as a decimal separator. Numbers may be preceded by `-` or `+` to indicate a negative or positive value, respectively. Numbers represented in scientific notation with a mantissa and exponent are approximate-value numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar may be treated differently. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types. For more information about exact-value calculations, see Section 12.25, “Precision Math”.

The `FLOAT` and `DOUBLE` data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with `FLOAT` or `DOUBLE` are `DOUBLE PRECISION` and `REAL`.

An integer may be used in a floating-point context; it is interpreted as the equivalent floating-point number.

9.1.3 Date and Time Literals

Date and time values can be represented in several formats, such as quoted strings or as numbers, depending on the exact type of the value and other factors. For example, in contexts where MySQL expects a date, it interprets any of `'2015-07-21'`, `'20150721'`, and `20150721` as a date.

This section describes the acceptable formats for date and time literals. For more information about the temporal data types, such as the range of permitted values, consult these sections:

- Section 11.1.2, “Date and Time Type Overview”
- Section 11.3, “Date and Time Types”

Standard SQL and ODBC Date and Time Literals. Standard SQL permits temporal literals to be specified using a type keyword and a string. The space between the keyword and string is optional.

```
DATE 'str'
TIME 'str'
TIMESTAMP 'str'
```

MySQL recognizes those constructions and also the corresponding ODBC syntax:

```
{ d 'str' }
{ t 'str' }
{ ts 'str' }
```

MySQL uses the type keyword and these constructions produce `DATE`, `TIME`, and `DATETIME` values, respectively, including a trailing fractional seconds part if specified. The `TIMESTAMP` syntax produces a `DATETIME` value in MySQL because `DATETIME` has a range that more closely corresponds to the standard SQL `TIMESTAMP` type, which has a year range from `0001` to `9999`. (The MySQL `TIMESTAMP` year range is `1970` to `2038`.)

String and Numeric Literals in Date and Time Context. MySQL recognizes `DATE` values in these formats:

- As a string in either *`'YYYY-MM-DD'`* or *`'YY-MM-DD'`* format. A “relaxed” syntax is permitted: Any punctuation character may be used as the delimiter between date parts. For example, `'2012-12-31'`, `'2012/12/31'`, `'2012^12^31'`, and `'2012@12@31'` are equivalent.
- As a string with no delimiters in either *`'YYYYMMDD'`* or *`'YYMMDD'`* format, provided that the string makes sense as a date. For example, `'20070523'` and `'070523'` are interpreted as `'2007-05-23'`, but `'071332'` is illegal (it has nonsensical month and day parts) and becomes `'0000-00-00'`.
- As a number in either *`YYYYMMDD`* or *`YYMMDD`* format, provided that the number makes sense as a date. For example, `19830905` and `830905` are interpreted as `'1983-09-05'`.

MySQL recognizes `DATETIME` and `TIMESTAMP` values in these formats:

- As a string in either *`'YYYY-MM-DD hh:mm:ss'`* or *`'YY-MM-DD hh:mm:ss'`* format. A “relaxed” syntax is permitted here, too: Any punctuation character may be used as the delimiter between date parts or time parts. For example, `'2012-12-31 11:30:45'`, `'2012^12^31 11+30+45'`, `'2012/12/31 11*30*45'`, and `'2012@12@31 11^30^45'` are equivalent.

  The only delimiter recognized between a date and time part and a fractional seconds part is the decimal point.

  The date and time parts can be separated by `T` rather than a space. For example, `'2012-12-31 11:30:45'` `'2012-12-31T11:30:45'` are equivalent.

- As a string with no delimiters in either *'YYYYMMDDhhmmss'* or *'YYMMDDhhmmss'* format, provided that the string makes sense as a date. For example, '20070523091528' and '070523091528' are interpreted as '2007-05-23 09:15:28', but '071122129015' is illegal (it has a nonsensical minute part) and becomes '0000-00-00 00:00:00'.
- As a number in either *YYYYMMDDhhmmss* or *YYMMDDhhmmss* format, provided that the number makes sense as a date. For example, 19830905132800 and 830905132800 are interpreted as '1983-09-05 13:28:00'.

A DATETIME or TIMESTAMP value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized. For information about fractional seconds support in MySQL, see Section 11.3.5, “Fractional Seconds in Time Values”.

Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

- Year values in the range 70-99 are converted to 1970-1999.
- Year values in the range 00-69 are converted to 2000-2069.

See also Section 11.3.7, “Two-Digit Years in Dates”.

For values specified as strings that include date part delimiters, it is unnecessary to specify two digits for month or day values that are less than 10. '2015-6-9' is the same as '2015-06-09'. Similarly, for values specified as strings that include time part delimiters, it is unnecessary to specify two digits for hour, minute, or second values that are less than 10. '2015-10-30 1:2:3' is the same as '2015-10-30 01:02:03'.

Values specified as numbers should be 6, 8, 12, or 14 digits long. If a number is 8 or 14 digits long, it is assumed to be in *YYYYMMDD* or *YYYYMMDDhhmmss* format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in *YYMMDD* or *YYMMDDhhmmss* format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as nondelimited strings are interpreted according their length. For a string 8 or 14 characters long, the year is assumed to be given by the first 4 characters. Otherwise, the year is assumed to be given by the first 2 characters. The string is interpreted from left to right to find year, month, day, hour, minute, and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify '9903', thinking that represents March, 1999, MySQL converts it to the “zero” date value. This occurs because the year and month values are 99 and 03, but the day part is completely missing. However, you can explicitly specify a value of zero to represent missing month or day parts. For example, to insert the value '1999-03-00', use '990300'.

MySQL recognizes TIME values in these formats:

- As a string in *'D hh:mm:ss'* format. You can also use one of the following “relaxed” syntaxes: *'hh:mm:ss'*, *'hh:mm'*, *'D hh:mm'*, *'D hh'*, or *'ss'*. Here *D* represents days and can have a value from 0 to 34.
- As a string with no delimiters in *'hhmmss'* format, provided that it makes sense as a time. For example, '101112' is understood as '10:11:12', but '109712' is illegal (it has a nonsensical minute part) and becomes '00:00:00'.
- As a number in *hhmmss* format, provided that it makes sense as a time. For example, 101112 is understood as '10:11:12'. The following alternative formats are also understood: *ss*, *mmss*, or *hhmmss*.

A trailing fractional seconds part is recognized in the *'D hh:mm:ss.fraction'*, *'hh:mm:ss.fraction'*, *'hhmmss.fraction'*, and *hhmmss.fraction* time formats, where *fraction* is the fractional part in up to microseconds (6 digits) precision. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized. For information about fractional seconds support in MySQL, see Section 11.3.5, “Fractional Seconds in Time Values”.

For TIME values specified as strings that include a time part delimiter, it is unnecessary to specify two digits for hours, minutes, or seconds values that are less than 10. '8:3:2' is the same as '08:03:02'.

9.1.4 Hexadecimal Literals

Hexadecimal literal values are written using X'*val*' or 0x*val* notation, where *val* contains hexadecimal digits (0..9, A..F). Lettercase of the digits and of any leading X does not matter. A leading 0x is case-sensitive and cannot be written as 0X.

Legal hexadecimal literals:

```
X'01AF'
X'01af'
x'01AF'
x'01af'
0x01AF
0x01af
```

Illegal hexadecimal literals:

```
X'0G'   (G is not a hexadecimal digit)
0X01AF  (0X must be written as 0x)
```

Values written using X'*val*' notation must contain an even number of digits or a syntax error occurs. To correct the problem, pad the value with a leading zero:

```
mysql> SET @s = X'FFF';
ERROR 1064 (42000): You have an error in your SQL syntax;
check the manual that corresponds to your MySQL server
version for the right syntax to use near 'X'FFF''

mysql> SET @s = X'0FFF';
Query OK, 0 rows affected (0.00 sec)
```

Values written using 0x*val* notation that contain an odd number of digits are treated as having an extra leading 0. For example, 0xaaa is interpreted as 0x0aaa.

By default, a hexadecimal literal is a binary string, where each pair of hexadecimal digits represents a character:

```
mysql> SELECT X'4D7953514C', CHARSET(X'4D7953514C');
+---------------+------------------------+
| X'4D7953514C' | CHARSET(X'4D7953514C') |
+---------------+------------------------+
| MySQL         | binary                 |
+---------------+------------------------+
mysql> SELECT 0x5461626c65, CHARSET(0x5461626c65);
+--------------+-----------------------+
| 0x5461626c65 | CHARSET(0x5461626c65) |
```

```
+--------------+-----------------------+
| Table        | binary                |
+--------------+-----------------------+
```

A hexadecimal literal may have an optional character set introducer and `COLLATE` clause, to designate it as a string that uses a particular character set and collation:

```
[_charset_name] X'val' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1 X'4D7953514C';
SELECT _utf8 0x4D7953514C COLLATE utf8_danish_ci;
```

The examples use `X'val'` notation, but `0xval` notation permits introducers as well. For information about introducers, see Section 10.3.8, “Character Set Introducers”.

In numeric contexts, MySQL treats a hexadecimal literal like a `BIGINT` (64-bit integer). To ensure numeric treatment of a hexadecimal literal, use it in numeric context. Ways to do this include adding 0 or using `CAST(... AS UNSIGNED)`. For example, a hexadecimal literal assigned to a user-defined variable is a binary string by default. To assign the value as a number, use it in numeric context:

```
mysql> SET @v1 = X'41';
mysql> SET @v2 = X'41'+0;
mysql> SET @v3 = CAST(X'41' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+
```

An empty hexadecimal value (`X''`) evaluates to a zero-length binary string. Converted to a number, it produces 0:

```
mysql> SELECT CHARSET(X''), LENGTH(X'');
+--------------+-------------+
| CHARSET(X'') | LENGTH(X'') |
+--------------+-------------+
| binary       |           0 |
+--------------+-------------+
mysql> SELECT X''+0;
+-------+
| X''+0 |
+-------+
|     0 |
+-------+
```

The `X'val'` notation is based on standard SQL. The `0x` notation is based on ODBC, for which hexadecimal strings are often used to supply values for `BLOB` columns.

To convert a string or a number to a string in hexadecimal format, use the `HEX()` function:

```
mysql> SELECT HEX('cat');
+------------+
| HEX('cat') |
+------------+
```

```
| 636174     |
+------------+
mysql> SELECT X'636174';
+-----------+
| X'636174' |
+-----------+
| cat       |
+-----------+
```

For hexadecimal literals, bit operations are considered numeric context, but bit operations permit numeric or binary string arguments in MySQL 8.0 and higher. To explicitly specify binary string context for hexadecimal literals, use a `_binary` introducer for at least one of the arguments:

```
mysql> SET @v1 = X'000D' | X'0BC0';
mysql> SET @v2 = _binary X'000D' | X'0BC0';
mysql> SELECT HEX(@v1), HEX(@v2);
+----------+----------+
| HEX(@v1) | HEX(@v2) |
+----------+----------+
| BCD      | 0BCD     |
+----------+----------+
```

The displayed result appears similar for both bit operations, but the result without `_binary` is a `BIGINT` value, whereas the result with `_binary` is a binary string. Due to the difference in result types, the displayed values differ: High-order 0 digits are not displayed for the numeric result.

9.1.5 Bit-Value Literals

Bit-value literals are written using `b'val'` or `0bval` notation. *`val`* is a binary value written using zeros and ones. Lettercase of any leading `b` does not matter. A leading `0b` is case sensitive and cannot be written as `0B`.

Legal bit-value literals:

```
b'01'
B'01'
0b01
```

Illegal bit-value literals:

```
b'2'    (2 is not a binary digit)
0B01    (0B must be written as 0b)
```

By default, a bit-value literal is a binary string:

```
mysql> SELECT b'1000001', CHARSET(b'1000001');
+------------+---------------------+
| b'1000001' | CHARSET(b'1000001') |
+------------+---------------------+
| A          | binary              |
+------------+---------------------+
mysql> SELECT 0b1100001, CHARSET(0b1100001);
+-----------+--------------------+
| 0b1100001 | CHARSET(0b1100001) |
+-----------+--------------------+
| a         | binary             |
+-----------+--------------------+
```

A bit-value literal may have an optional character set introducer and `COLLATE` clause, to designate it as a string that uses a particular character set and collation:

```
[_charset_name] b'val' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1 b'1000001';
SELECT _utf8 0b1000001 COLLATE utf8_danish_ci;
```

The examples use `b'val'` notation, but `0bval` notation permits introducers as well. For information about introducers, see Section 10.3.8, “Character Set Introducers”.

In numeric contexts, MySQL treats a bit literal like an integer. To ensure numeric treatment of a bit literal, use it in numeric context. Ways to do this include adding 0 or using `CAST(... AS UNSIGNED)`. For example, a bit literal assigned to a user-defined variable is a binary string by default. To assign the value as a number, use it in numeric context:

```
mysql> SET @v1 = b'1100001';
mysql> SET @v2 = b'1100001'+0;
mysql> SET @v3 = CAST(b'1100001' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| a    |   97 |   97 |
+------+------+------+
```

An empty bit value (`b''`) evaluates to a zero-length binary string. Converted to a number, it produces 0:

```
mysql> SELECT CHARSET(b''), LENGTH(b'');
+--------------+-------------+
| CHARSET(b'') | LENGTH(b'') |
+--------------+-------------+
| binary       |           0 |
+--------------+-------------+
mysql> SELECT b''+0;
+-------+
| b''+0 |
+-------+
|     0 |
+-------+
```

Bit-value notation is convenient for specifying values to be assigned to `BIT` columns:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
mysql> INSERT INTO t SET b = b'0101';
```

Bit values in result sets are returned as binary values, which may not display well. To convert a bit value to printable form, use it in numeric context or use a conversion function such as `BIN()` or `HEX()`. High-order 0 digits are not displayed in the converted value.

```
mysql> SELECT b+0, BIN(b), OCT(b), HEX(b) FROM t;
+------+----------+--------+--------+
```

```
| b+0  | BIN(b)   | OCT(b) | HEX(b) |
+------+----------+--------+--------+
|  255 | 11111111 | 377    | FF     |
|   10 | 1010     | 12     | A      |
|    5 | 101      | 5      | 5      |
+------+----------+--------+--------+
```

For bit literals, bit operations are considered numeric context, but bit operations permit numeric or binary string arguments in MySQL 8.0 and higher. To explicitly specify binary string context for bit literals, use a `_binary` introducer for at least one of the arguments:

```
mysql> SET @v1 = b'000010101' | b'000101010';
mysql> SET @v2 = _binary b'000010101' | _binary b'000101010';
mysql> SELECT HEX(@v1), HEX(@v2);
+----------+----------+
| HEX(@v1) | HEX(@v2) |
+----------+----------+
| 3F       | 003F     |
+----------+----------+
```

The displayed result appears similar for both bit operations, but the result without `_binary` is a `BIGINT` value, whereas the result with `_binary` is a binary string. Due to the difference in result types, the displayed values differ: High-order 0 digits are not displayed for the numeric result.

9.1.6 Boolean Literals

The constants `TRUE` and `FALSE` evaluate to `1` and `0`, respectively. The constant names can be written in any lettercase.

```
mysql> SELECT TRUE, true, FALSE, false;
       -> 1, 1, 0, 0
```

9.1.7 NULL Values

The `NULL` value means “no data.” `NULL` can be written in any lettercase.

Be aware that the `NULL` value is different from values such as `0` for numeric types or the empty string for string types. For more information, see Section B.4.4.3, “Problems with NULL Values”.

For text file import or export operations performed with `LOAD DATA` or `SELECT ... INTO OUTFILE`, `NULL` is represented by the `\N` sequence. See Section 13.2.7, “LOAD DATA Syntax”.

For sorting with `ORDER BY`, `NULL` values sort before other values for ascending sorts, after other values for descending sorts.

9.2 Schema Object Names

Certain objects within MySQL, including database, table, index, column, alias, view, stored procedure, partition, tablespace, resource group and other object names are known as identifiers. This section describes the permissible syntax for identifiers in MySQL. Section 9.2.1, “Identifier Length Limits”, indicates the maximum length of each type of identifier. Section 9.2.3, “Identifier Case Sensitivity”, describes which types of identifiers are case-sensitive and under what conditions.

An identifier may be quoted or unquoted. If an identifier contains special characters or is a reserved word, you *must* quote it whenever you refer to it. (Exception: A reserved word that follows a period in a qualified

name must be an identifier, so it need not be quoted.) Reserved words are listed at Section 9.3, “Keywords and Reserved Words”.

Internally, identifiers are converted to and are stored as Unicode (UTF-8). The permissible Unicode characters in identifiers are those in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted. Identifiers thus may contain these characters:

- Permitted characters in unquoted identifiers:
  - ASCII: [0-9,a-z,A-Z$_] (basic Latin letters, digits 0-9, dollar, underscore)
  - Extended: U+0080 .. U+FFFF
- Permitted characters in quoted identifiers include the full Unicode Basic Multilingual Plane (BMP), except U+0000:
  - ASCII: U+0001 .. U+007F
  - Extended: U+0080 .. U+FFFF
- ASCII NUL (U+0000) and supplementary characters (U+10000 and higher) are not permitted in quoted or unquoted identifiers.
- Identifiers may begin with a digit but unless quoted may not consist solely of digits.
- Database, table, and column names cannot end with space characters.

The identifier quote character is the backtick (`):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

If the ANSI_QUOTES SQL mode is enabled, it is also permissible to quote identifiers within double quotation marks:

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax...
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

The ANSI_QUOTES mode causes the server to interpret double-quoted strings as identifiers. Consequently, when this mode is enabled, string literals must be enclosed within single quotation marks. They cannot be enclosed within double quotation marks. The server SQL mode is controlled as described in Section 5.1.11, “Server SQL Modes”.

Identifier quote characters can be included within an identifier if you quote the identifier. If the character to be included within the identifier is the same as that used to quote the identifier itself, then you need to double the character. The following statement creates a table named a`b that contains a column named c"d:

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
mysql> SELECT 1 AS `one`, 2 AS 'two';
+-----+-----+
| one | two |
+-----+-----+
|   1 |   2 |
+-----+-----+
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal.

It is recommended that you do not use names that begin with `Me` or `MeN`, where `M` and `N` are integers. For example, avoid using `1e` as an identifier, because an expression such as `1e+3` is ambiguous. Depending on context, it might be interpreted as the expression `1e + 3` or as the number `1e+3`.

Be careful when using `MD5()` to produce table names because it can produce names in illegal or ambiguous formats such as those just described.

A user variable cannot be used directly in an SQL statement as an identifier or as part of an identifier. See Section 9.4, “User-Defined Variables”, for more information and examples of workarounds.

Special characters in database and table names are encoded in the corresponding file system names as described in Section 9.2.4, “Mapping of Identifiers to File Names”.

9.2.1 Identifier Length Limits

The following table describes the maximum length for each type of identifier.

Identifier Type	Maximum Length (characters)
Database	64 (includes NDB Cluster 8.0.18 and later)
Table	64 (includes NDB Cluster 8.0.18 and later)
Column	64
Index	64
Constraint	64
Stored Program	64
View	64
Tablespace	64
Server	64
Log File Group	64
Alias	256 (see exception following table)

Identifier Type	Maximum Length (characters)
Compound Statement Label	16
User-Defined Variable	64
Resource Group	64

Aliases for column names in `CREATE VIEW` statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

Identifiers are stored using Unicode (UTF-8). This applies to identifiers in table definitions and to identifiers stored in the grant tables in the `mysql` database. The sizes of the identifier string columns in the grant tables are measured in characters. You can use multibyte characters without reducing the number of characters permitted for values stored in these columns.

Prior to NDB 8.0.18, NDB Cluster imposed a maximum length of 63 characters for names of databases and tables. As of NDB 8.0.18, this limitation is removed. See Section 22.1.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 8.0”.

Values such as user name and host names in MySQL account names are strings rather than identifiers. For information about the maximum length of such values as stored in grant tables, see Grant Table Scope Column Properties.

9.2.2 Identifier Qualifiers

Object names may be unqualified or qualified. An unqualified name is permitted in contexts where interpretation of the name is unambiguous. A qualified name includes at least one qualifier to clarify the interpretive context by overriding a default context or providing missing context.

For example, this statement creates a table using the unqualified name `t1`:

```
CREATE TABLE t1 (i INT);
```

Because `t1` includes no qualifier to specify a database, the statement creates the table in the default database. If there is no default database, an error occurs.

This statement creates a table using the qualified name `db1.t1`:

```
CREATE TABLE db1.t1 (i INT);
```

Because `db1.t1` includes a database qualifier `db1`, the statement creates `t1` in the database named `db1`, regardless of the default database. The qualifier *must* be specified if there is no default database. The qualifier *may* be specified if there is a default database, to specify a database different from the default, or to make the database explicit if the default is the same as the one specified.

Qualifiers have these characteristics:

- An unqualified name consists of a single identifier. A qualified name consists of multiple identifiers.
- The components of a multiple-part name must be separated by period (`.`) characters. The initial parts of a multiple-part name act as qualifiers that affect the context within which to interpret the final identifier.

- The qualifier character is a separate token and need not be contiguous with the associated identifiers. For example, *`tbl_name.col_name`* and *`tbl_name . col_name`* are equivalent.
- If any components of a multiple-part name require quoting, quote them individually rather than quoting the name as a whole. For example, write `` `my-table`.`my-column` ``, not `` `my-table.my-column` ``.
- A reserved word that follows a period in a qualified name must be an identifier, so in that context it need not be quoted.

The permitted qualifiers for object names depend on the object type:

- A database name is fully qualified and takes no qualifier:

```
CREATE DATABASE db1;
```

- A table, view, or stored program name may be given a database-name qualifier. Examples of unqualified and qualified names in `CREATE` statements:

```
CREATE TABLE mytable ...;
CREATE VIEW myview ...;
CREATE PROCEDURE myproc ...;
CREATE FUNCTION myfunc ...;
CREATE EVENT myevent ...;

CREATE TABLE mydb.mytable ...;
CREATE VIEW mydb.myview ...;
CREATE PROCEDURE mydb.myproc ...;
CREATE FUNCTION mydb.myfunc ...;
CREATE EVENT mydb.myevent ...;
```

- A trigger is associated with a table, so any qualifier applies to the table name:

```
CREATE TRIGGER mytrigger ... ON mytable ...;

CREATE TRIGGER mytrigger ... ON mydb.mytable ...;
```

- A column name may be given multiple qualifiers to indicate context in statements that reference it, as shown in the following table.

Column Reference	Meaning
`col_name`	Column *`col_name`* from whichever table used in the statement contains a column of that name
`tbl_name.col_name`	Column *`col_name`* from table *`tbl_name`* of the default database
`db_name.tbl_name.col_name`	Column *`col_name`* from table *`tbl_name`* of the database *`db_name`*

In other words, a column name may be given a table-name qualifier, which itself may be given a database-name qualifier. Examples of unqualified and qualified column references in `SELECT` statements:

```
SELECT c1 FROM mytable
WHERE c2 > 100;

SELECT mytable.c1 FROM mytable
WHERE mytable.c2 > 100;
```

```
SELECT mydb.mytable.c1 FROM mydb.mytable
WHERE mydb.mytable.c2 > 100;
```

You need not specify a qualifier for an object reference in a statement unless the unqualified reference is ambiguous. Suppose that column `c1` occurs only in table `t1`, `c2` only in `t2`, and `c` in both `t1` and `t2`. Any unqualified reference to `c` is ambiguous in a statement that refers to both tables and must be qualified as `t1.c` or `t2.c` to indicate which table you mean:

```
SELECT c1, c2, t1.c FROM t1 INNER JOIN t2
WHERE t2.c > 100;
```

Similarly, to retrieve from a table `t` in database `db1` and from a table `t` in database `db2` in the same statement, you must qualify the table references: For references to columns in those tables, qualifiers are required only for column names that appear in both tables. Suppose that column `c1` occurs only in table `db1.t`, `c2` only in `db2.t`, and `c` in both `db1.t` and `db2.t`. In this case, `c` is ambiguous and must be qualified but `c1` and `c2` need not be:

```
SELECT c1, c2, db1.t.c FROM db1.t INNER JOIN db2.t
WHERE db2.t.c > 100;
```

Table aliases enable qualified column references to be written more simply:

```
SELECT c1, c2, t1.c FROM db1.t AS t1 INNER JOIN db2.t AS t2
WHERE t2.c > 100;
```

9.2.3 Identifier Case Sensitivity

In MySQL, databases correspond to directories within the data directory. Each table within a database corresponds to at least one file within the database directory (and possibly more, depending on the storage engine). Triggers also correspond to files. Consequently, the case sensitivity of the underlying operating system plays a part in the case sensitivity of database, table, and trigger names. This means such names are not case-sensitive in Windows, but are case-sensitive in most varieties of Unix. One notable exception is macOS, which is Unix-based but uses a default file system type (HFS+) that is not case-sensitive. However, macOS also supports UFS volumes, which are case-sensitive just as on any Unix. See Section 1.8.1, “MySQL Extensions to Standard SQL”. The `lower_case_table_names` system variable also affects how the server handles identifier case sensitivity, as described later in this section.

Note

Although database, table, and trigger names are not case sensitive on some platforms, you should not refer to one of these using different cases within the same statement. The following statement would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Column, index, stored routine, event, and resource group names are not case-sensitive on any platform, nor are column aliases.

However, names of logfile groups are case-sensitive. This differs from standard SQL.

By default, table aliases are case-sensitive on Unix, but not so on Windows or macOS. The following statement would not work on Unix, because it refers to the alias both as `a` and as `A`:

```
```

```
mysql> SELECT col_name FROM tbl_name AS a
       WHERE a.col_name = 1 OR A.col_name = 2;
```

However, this same statement is permitted on Windows. To avoid problems caused by such differences, it is best to adopt a consistent convention, such as always creating and referring to databases and tables using lowercase names. This convention is recommended for maximum portability and ease of use.

How table and database names are stored on disk and used in MySQL is affected by the `lower_case_table_names` system variable. `lower_case_table_names` can take the values shown in the following table. This variable does *not* affect case sensitivity of trigger identifiers. On Unix, the default value of `lower_case_table_names` is 0. On Windows, the default value is 1. On macOS, the default value is 2.

`lower_case_table_names` can only be configured when initializing the server. Changing the `lower_case_table_names` setting after the server is initialized is prohibited.

Value	Meaning
`0`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement. Name comparisons are case sensitive. You should *not* set this variable to 0 if you are running MySQL on a system that has case-insensitive file names (such as Windows or macOS). If you force this variable to 0 with `--lower-case-table-names=0` on a case-insensitive file system and access `MyISAM` tablenames using different lettercases, index corruption may result.
`1`	Table names are stored in lowercase on disk and name comparisons are not case-sensitive. MySQL converts all table names to lowercase on storage and lookup. This behavior also applies to database names and table aliases.
`2`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement, but MySQL converts them to lowercase on lookup. Name comparisons are not case sensitive. This works *only* on file systems that are not case-sensitive! `InnoDB` table names and view names are stored in lowercase, as for `lower_case_table_names=1`.

If you are using MySQL on only one platform, you do not normally have to use a `lower_case_table_names` setting other than the default. However, you may encounter difficulties if you want to transfer tables between platforms that differ in file system case sensitivity. For example, on Unix, you can have two different tables named `my_table` and `MY_TABLE`, but on Windows these two names are considered identical. To avoid data transfer problems arising from lettercase of database or table names, you have two options:

- Use `lower_case_table_names=1` on all systems. The main disadvantage with this is that when you use `SHOW TABLES` or `SHOW DATABASES`, you do not see the names in their original lettercase.
- Use `lower_case_table_names=0` on Unix and `lower_case_table_names=2` on Windows. This preserves the lettercase of database and table names. The disadvantage of this is that you must ensure that your statements always refer to your database and table names with the correct lettercase on Windows. If you transfer your statements to Unix, where lettercase is significant, they do not work if the lettercase is incorrect.

  Exception: If you are using `InnoDB` tables and you are trying to avoid these data transfer problems, you should use `lower_case_table_names=1` on all platforms to force names to be converted to lowercase.

Object names may be considered duplicates if their uppercase forms are equal according to a binary collation. That is true for names of cursors, conditions, procedures, functions, savepoints, stored routine

parameters, stored program local variables, and plugins. It is not true for names of columns, constraints, databases, partitions, statements prepared with PREPARE, tables, triggers, users, and user-defined variables.

File system case sensitivity can affect searches in string columns of INFORMATION_SCHEMA tables. For more information, see Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”.

9.2.4 Mapping of Identifiers to File Names

There is a correspondence between database and table identifiers and names in the file system. For the basic structure, MySQL represents each database as a directory in the data directory, and depending upon the storage engine, each table may be represented by one or more files in the appropriate database directory.

For the data and index files, the exact representation on disk is storage engine specific. These files may be stored in the database directory, or the information may be stored in a separate file. InnoDB data is stored in the InnoDB data files. If you are using tablespaces with InnoDB, then the specific tablespace files you create are used instead.

Any character is legal in database or table identifiers except ASCII NUL (X'00'). MySQL encodes any characters that are problematic in the corresponding file system objects when it creates database directories or table files:

- Basic Latin letters (a..zA..Z), digits (0..9) and underscore (_) are encoded as is. Consequently, their case sensitivity directly depends on file system features.
- All other national letters from alphabets that have uppercase/lowercase mapping are encoded as shown in the following table. Values in the Code Range column are UCS-2 values.

Code Range	Pattern	Number	Used	Unused	Blocks
00C0..017F	[@][0..4][g..z]	5*20= 100	97	3	Latin-1 Supplement + Latin Extended-A
0370..03FF	[@][5..9][g..z]	5*20= 100	88	12	Greek and Coptic
0400..052F	[@][g..z][0..6]	20*7= 140	137	3	Cyrillic + Cyrillic Supplement
0530..058F	[@][g..z][7..8]	20*2= 40	38	2	Armenian
2160..217F	[@][g..z][9]	20*1= 20	16	4	Number Forms
0180..02AF	[@][g..z][a..k]	20*11=220	203	17	Latin Extended-B + IPA Extensions
1E00..1EFF	[@][g..z][l..r]	20*7= 140	136	4	Latin Extended Additional
1F00..1FFF	[@][g..z][s..z]	20*8= 160	144	16	Greek Extended
....	[@][a..f][g..z]	6*20= 120	0	120	RESERVED
24B6..24E9	[@][@][a..z]	26	26	0	Enclosed Alphanumerics
FF21..FF5A	[@][a..z][@]	26	26	0	Halfwidth and Fullwidth forms

One of the bytes in the sequence encodes lettercase. For example: LATIN CAPITAL LETTER A WITH GRAVE is encoded as @0G, whereas LATIN SMALL LETTER A WITH GRAVE is encoded as @0g. Here the third byte (G or g) indicates lettercase. (On a case-insensitive file system, both letters will be treated as the same.)

For some blocks, such as Cyrillic, the second byte determines lettercase. For other blocks, such as Latin1 Supplement, the third byte determines lettercase. If two bytes in the sequence are letters (as in Greek Extended), the leftmost letter character stands for lettercase. All other letter bytes must be in lowercase.

- All nonletter characters except underscore (_), as well as letters from alphabets that do not have uppercase/lowercase mapping (such as Hebrew) are encoded using hexadecimal representation using lowercase letters for hexadecimal digits `a..f`:

```
0x003F -> @003f
0xFFFF -> @ffff
```

  The hexadecimal values correspond to character values in the `ucs2` double-byte character set.

On Windows, some names such as `nul`, `prn`, and `aux` are encoded by appending `@@@` to the name when the server creates the corresponding file or directory. This occurs on all platforms for portability of the corresponding database object between platforms.

9.2.5 Function Name Parsing and Resolution

MySQL supports built-in (native) functions, user-defined functions (UDFs), and stored functions. This section describes how the server recognizes whether the name of a built-in function is used as a function call or as an identifier, and how the server determines which function to use in cases when functions of different types exist with a given name.

- Built-In Function Name Parsing
- Function Name Resolution

Built-In Function Name Parsing

The parser uses default rules for parsing names of built-in functions. These rules can be changed by enabling the `IGNORE_SPACE` SQL mode.

When the parser encounters a word that is the name of a built-in function, it must determine whether the name signifies a function call or is instead a nonexpression reference to an identifier such as a table or column name. For example, in the following statements, the first reference to `count` is a function call, whereas the second reference is a table name:

```
SELECT COUNT(*) FROM mytable;
CREATE TABLE count (i INT);
```

The parser should recognize the name of a built-in function as indicating a function call only when parsing what is expected to be an expression. That is, in nonexpression context, function names are permitted as identifiers.

However, some built-in functions have special parsing or implementation considerations, so the parser uses the following rules by default to distinguish whether their names are being used as function calls or as identifiers in nonexpression context:

- To use the name as a function call in an expression, there must be no whitespace between the name and the following `(` parenthesis character.
- Conversely, to use the function name as an identifier, it must not be followed immediately by a parenthesis.

The requirement that function calls be written with no whitespace between the name and the parenthesis applies only to the built-in functions that have special considerations. `COUNT` is one such name. The `sql/lex.h` source file lists the names of these special functions for which following whitespace determines their interpretation: names defined by the `SYM_FN()` macro in the `symbols[]` array.

The following list names the functions in MySQL 8.0 that are affected by the `IGNORE_SPACE` setting and listed as special in the `sql/lex.h` source file. You may find it easiest to treat the no-whitespace requirement as applying to all function calls.

- `ADDDATE`
- `BIT_AND`
- `BIT_OR`
- `BIT_XOR`
- `CAST`
- `COUNT`
- `CURDATE`
- `CURTIME`
- `DATE_ADD`
- `DATE_SUB`
- `EXTRACT`
- `GROUP_CONCAT`
- `MAX`
- `MID`
- `MIN`
- `NOW`
- `POSITION`
- `SESSION_USER`
- `STD`
- `STDDEV`
- `STDDEV_POP`
- `STDDEV_SAMP`
- `SUBDATE`
- `SUBSTR`
- `SUBSTRING`
- `SUM`

- SYSDATE
- SYSTEM_USER
- TRIM
- VARIANCE
- VAR_POP
- VAR_SAMP

For functions not listed as special in sql/lex.h, whitespace does not matter. They are interpreted as function calls only when used in expression context and may be used freely as identifiers otherwise. ASCII is one such name. However, for these nonaffected function names, interpretation may vary in expression context: *func_name* () is interpreted as a built-in function if there is one with the given name; if not, *func_name* () is interpreted as a user-defined function or stored function if one exists with that name.

The IGNORE_SPACE SQL mode can be used to modify how the parser treats function names that are whitespace-sensitive:

- With IGNORE_SPACE disabled, the parser interprets the name as a function call when there is no whitespace between the name and the following parenthesis. This occurs even when the function name is used in nonexpression context:

```
mysql> CREATE TABLE count(i INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'count(i INT)'
```

  To eliminate the error and cause the name to be treated as an identifier, either use whitespace following the name or write it as a quoted identifier (or both):

```
CREATE TABLE count (i INT);
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

- With IGNORE_SPACE enabled, the parser loosens the requirement that there be no whitespace between the function name and the following parenthesis. This provides more flexibility in writing function calls. For example, either of the following function calls are legal:

```
SELECT COUNT(*) FROM mytable;
SELECT COUNT (*) FROM mytable;
```

  However, enabling IGNORE_SPACE also has the side effect that the parser treats the affected function names as reserved words (see Section 9.3, “Keywords and Reserved Words”). This means that a space following the name no longer signifies its use as an identifier. The name can be used in function calls with or without following whitespace, but causes a syntax error in nonexpression context unless it is quoted. For example, with IGNORE_SPACE enabled, both of the following statements fail with a syntax error because the parser interprets count as a reserved word:

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

  To use the function name in nonexpression context, write it as a quoted identifier:

```
```

```
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

To enable the IGNORE_SPACE SQL mode, use this statement:

```
SET sql_mode = 'IGNORE_SPACE';
```

IGNORE_SPACE is also enabled by certain other composite modes such as ANSI that include it in their value:

```
SET sql_mode = 'ANSI';
```

Check Section 5.1.11, “Server SQL Modes”, to see which composite modes enable IGNORE_SPACE.

To minimize the dependency of SQL code on the IGNORE_SPACE setting, use these guidelines:

- Avoid creating UDFs or stored functions that have the same name as a built-in function.
- Avoid using function names in nonexpression context. For example, these statements use count (one of the affected function names affected by IGNORE_SPACE), so they fail with or without whitespace following the name if IGNORE_SPACE is enabled:

  ```
  CREATE TABLE count(i INT);
  CREATE TABLE count (i INT);
  ```

  If you must use a function name in nonexpression context, write it as a quoted identifier:

  ```
  CREATE TABLE `count`(i INT);
  CREATE TABLE `count` (i INT);
  ```

Function Name Resolution

The following rules describe how the server resolves references to function names for function creation and invocation:

- Built-in functions and user-defined functions

  An error occurs if you try to create a UDF with the same name as a built-in function.

- Built-in functions and stored functions

  It is possible to create a stored function with the same name as a built-in function, but to invoke the stored function it is necessary to qualify it with a schema name. For example, if you create a stored function named PI in the test schema, invoke it as test.PI() because the server resolves PI() without a qualifier as a reference to the built-in function. The server generates a warning if the stored function name collides with a built-in function name. The warning can be displayed with SHOW WARNINGS.

- User-defined functions and stored functions

  User-defined functions and stored functions share the same namespace, so you cannot create a UDF and a stored function with the same name.

The preceding function name resolution rules have implications for upgrading to versions of MySQL that implement new built-in functions:

- If you have already created a user-defined function with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF and `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name. Then modify any affected code to use the new name.
- If a new version of MySQL implements a built-in function with the same name as an existing stored function, you have two choices: Rename the stored function to use a nonconflicting name, or change calls to the function so that they use a schema qualifier (that is, use *`schema_name.func_name()`* syntax). In either case, modify any affected code accordingly.

9.3 Keywords and Reserved Words

Keywords are words that have significance in SQL. Certain keywords, such as `SELECT`, `DELETE`, or `BIGINT`, are reserved and require special treatment for use as identifiers such as table and column names. This may also be true for the names of built-in functions.

Nonreserved keywords are permitted as identifiers without quoting. Reserved words are permitted as identifiers if you quote them as described in Section 9.2, “Schema Object Names”:

```
mysql> CREATE TABLE interval (begin INT, end INT);
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'interval (begin INT, end INT)'
```

`BEGIN` and `END` are keywords but not reserved, so their use as identifiers does not require quoting. `INTERVAL` is a reserved keyword and must be quoted to be used as an identifier:

```
mysql> CREATE TABLE `interval` (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Exception: A word that follows a period in a qualified name must be an identifier, so it need not be quoted even if it is reserved:

```
mysql> CREATE TABLE mydb.interval (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Names of built-in functions are permitted as identifiers but may require care to be used as such. For example, `COUNT` is acceptable as a column name. However, by default, no whitespace is permitted in function invocations between the function name and the following `(` character. This requirement enables the parser to distinguish whether the name is used in a function call or in nonfunction context. For further details on recognition of function names, see Section 9.2.5, “Function Name Parsing and Resolution”.

The `INFORMATION_SCHEMA.KEYWORDS` table lists the words considered keywords by MySQL and indicates whether they are reserved. See Section 25.14, “The INFORMATION_SCHEMA KEYWORDS Table”.

- MySQL 8.0 Keywords and Reserved Words
- MySQL 8.0 New Keywords and Reserved Words
- MySQL 8.0 Removed Keywords and Reserved Words

MySQL 8.0 Keywords and Reserved Words

The following list shows the keywords and reserved words in MySQL 8.0, along with changes to individual words from version to version. Reserved keywords are marked with (R). In addition, `_FILENAME` is reserved.

At some point, you might upgrade to a higher version, so it is a good idea to have a look at future reserved words, too. You can find these in the manuals that cover higher versions of MySQL. Most of the reserved words in the list are forbidden by standard SQL as column or table names (for example, `GROUP`). A few are reserved because MySQL needs them and uses a `yacc` parser.

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

A

- `ACCESSIBLE` (R)
- `ACCOUNT`
- `ACTION`
- `ACTIVE`; added in 8.0.14 (nonreserved)
- `ADD` (R)
- `ADMIN`; became nonreserved in 8.0.12
- `AFTER`
- `AGAINST`
- `AGGREGATE`
- `ALGORITHM`
- `ALL` (R)
- `ALTER` (R)
- `ALWAYS`
- `ANALYSE`; removed in 8.0.1
- `ANALYZE` (R)
- `AND` (R)
- `ANY`
- `ARRAY` (R); added in 8.0.17 (reserved)
- `AS` (R)
- `ASC` (R)
- `ASCII`
- `ASENSITIVE` (R)
- `AT`
- `AUTOEXTEND_SIZE`
- `AUTO_INCREMENT`
- `AVG`

- AVG_ROW_LENGTH

B

- BACKUP
- BEFORE (R)
- BEGIN
- BETWEEN (R)
- BIGINT (R)
- BINARY (R)
- BINLOG
- BIT
- BLOB (R)
- BLOCK
- BOOL
- BOOLEAN
- BOTH (R)
- BTREE
- BUCKETS; added in 8.0.2 (nonreserved)
- BY (R)
- BYTE

C

- CACHE
- CALL (R)
- CASCADE (R)
- CASCADED
- CASE (R)
- CATALOG_NAME
- CHAIN
- CHANGE (R)
- CHANGED
- CHANNEL

- CHAR (R)
- CHARACTER (R)
- CHARSET
- CHECK (R)
- CHECKSUM
- CIPHER
- CLASS_ORIGIN
- CLIENT
- CLONE; added in 8.0.3 (nonreserved)
- CLOSE
- COALESCE
- CODE
- COLLATE (R)
- COLLATION
- COLUMN (R)
- COLUMNS
- COLUMN_FORMAT
- COLUMN_NAME
- COMMENT
- COMMIT
- COMMITTED
- COMPACT
- COMPLETION
- COMPONENT
- COMPRESSED
- COMPRESSION
- CONCURRENT
- CONDITION (R)
- CONNECTION
- CONSISTENT

- CONSTRAINT (R)
- CONSTRAINT_CATALOG
- CONSTRAINT_NAME
- CONSTRAINT_SCHEMA
- CONTAINS
- CONTEXT
- CONTINUE (R)
- CONVERT (R)
- CPU
- CREATE (R)
- CROSS (R)
- CUBE (R); became reserved in 8.0.1
- CUME_DIST (R); added in 8.0.2 (reserved)
- CURRENT
- CURRENT_DATE (R)
- CURRENT_TIME (R)
- CURRENT_TIMESTAMP (R)
- CURRENT_USER (R)
- CURSOR (R)
- CURSOR_NAME

D

- DATA
- DATABASE (R)
- DATABASES (R)
- DATAFILE
- DATE
- DATETIME
- DAY
- DAY_HOUR (R)
- DAY_MICROSECOND (R)

- DAY_MINUTE (R)
- DAY_SECOND (R)
- DEALLOCATE
- DEC (R)
- DECIMAL (R)
- DECLARE (R)
- DEFAULT (R)
- DEFAULT_AUTH
- DEFINER
- DEFINITION; added in 8.0.4 (nonreserved)
- DELAYED (R)
- DELAY_KEY_WRITE
- DELETE (R)
- DENSE_RANK (R); added in 8.0.2 (reserved)
- DESC (R)
- DESCRIBE (R)
- DESCRIPTION; added in 8.0.4 (nonreserved)
- DES_KEY_FILE; removed in 8.0.3
- DETERMINISTIC (R)
- DIAGNOSTICS
- DIRECTORY
- DISABLE
- DISCARD
- DISK
- DISTINCT (R)
- DISTINCTROW (R)
- DIV (R)
- DO
- DOUBLE (R)
- DROP (R)

- DUAL (R)
- DUMPFILE
- DUPLICATE
- DYNAMIC

E

- EACH (R)
- ELSE (R)
- ELSEIF (R)
- EMPTY (R); added in 8.0.4 (reserved)
- ENABLE
- ENCLOSED (R)
- ENCRYPTION
- END
- ENDS
- ENFORCED; added in 8.0.16 (nonreserved)
- ENGINE
- ENGINES
- ENUM
- ERROR
- ERRORS
- ESCAPE
- ESCAPED (R)
- EVENT
- EVENTS
- EVERY
- EXCEPT (R)
- EXCHANGE
- EXCLUDE; added in 8.0.2 (nonreserved)
- EXECUTE
- EXISTS (R)

- EXIT (R)
- EXPANSION
- EXPIRE
- EXPLAIN (R)
- EXPORT
- EXTENDED
- EXTENT_SIZE

F

- FALSE (R)
- FAST
- FAULTS
- FETCH (R)
- FIELDS
- FILE
- FILE_BLOCK_SIZE
- FILTER
- FIRST
- FIRST_VALUE (R); added in 8.0.2 (reserved)
- FIXED
- FLOAT (R)
- FLOAT4 (R)
- FLOAT8 (R)
- FLUSH
- FOLLOWING; added in 8.0.2 (nonreserved)
- FOLLOWS
- FOR (R)
- FORCE (R)
- FOREIGN (R)
- FORMAT
- FOUND

- FROM (R)
- FULL
- FULLTEXT (R)
- FUNCTION (R); became reserved in 8.0.1

G

- GENERAL
- GENERATED (R)
- GEOMCOLLECTION; added in 8.0.11 (nonreserved)
- GEOMETRY
- GEOMETRYCOLLECTION
- GET (R)
- GET_FORMAT
- GET_MASTER_PUBLIC_KEY; added in 8.0.4 (reserved); became nonreserved in 8.0.11
- GLOBAL
- GRANT (R)
- GRANTS
- GROUP (R)
- GROUPING (R); added in 8.0.1 (reserved)
- GROUPS (R); added in 8.0.2 (reserved)
- GROUP_REPLICATION

H

- HANDLER
- HASH
- HAVING (R)
- HELP
- HIGH_PRIORITY (R)
- HISTOGRAM; added in 8.0.2 (nonreserved)
- HISTORY; added in 8.0.3 (nonreserved)
- HOST
- HOSTS

- HOUR
- HOUR_MICROSECOND (R)
- HOUR_MINUTE (R)
- HOUR_SECOND (R)

I

- IDENTIFIED
- IF (R)
- IGNORE (R)
- IGNORE_SERVER_IDS
- IMPORT
- IN (R)
- INACTIVE; added in 8.0.14 (nonreserved)
- INDEX (R)
- INDEXES
- INFILE (R)
- INITIAL_SIZE
- INNER (R)
- INOUT (R)
- INSENSITIVE (R)
- INSERT (R)
- INSERT_METHOD
- INSTALL
- INSTANCE
- INT (R)
- INT1 (R)
- INT2 (R)
- INT3 (R)
- INT4 (R)
- INT8 (R)
- INTEGER (R)

- INTERVAL (R)
- INTO (R)
- INVISIBLE
- INVOKER
- IO
- IO_AFTER_GTIDS (R)
- IO_BEFORE_GTIDS (R)
- IO_THREAD
- IPC
- IS (R)
- ISOLATION
- ISSUER
- ITERATE (R)

J

- JOIN (R)
- JSON
- JSON_TABLE (R); added in 8.0.4 (reserved)

K

- KEY (R)
- KEYS (R)
- KEY_BLOCK_SIZE
- KILL (R)

L

- LAG (R); added in 8.0.2 (reserved)
- LANGUAGE
- LAST
- LAST_VALUE (R); added in 8.0.2 (reserved)
- LATERAL (R); added in 8.0.14 (reserved)
- LEAD (R); added in 8.0.2 (reserved)
- LEADING (R)

- LEAVE (R)
- LEAVES
- LEFT (R)
- LESS
- LEVEL
- LIKE (R)
- LIMIT (R)
- LINEAR (R)
- LINES (R)
- LINESTRING
- LIST
- LOAD (R)
- LOCAL
- LOCALTIME (R)
- LOCALTIMESTAMP (R)
- LOCK (R)
- LOCKED; added in 8.0.1 (nonreserved)
- LOCKS
- LOGFILE
- LOGS
- LONG (R)
- LONGBLOB (R)
- LONGTEXT (R)
- LOOP (R)
- LOW_PRIORITY (R)

M

- MASTER
- MASTER_AUTO_POSITION
- MASTER_BIND (R)
- MASTER_COMPRESSION_ALGORITHMS; added in 8.0.18 (nonreserved)

- MASTER_CONNECT_RETRY
- MASTER_DELAY
- MASTER_HEARTBEAT_PERIOD
- MASTER_HOST
- MASTER_LOG_FILE
- MASTER_LOG_POS
- MASTER_PASSWORD
- MASTER_PORT
- MASTER_PUBLIC_KEY_PATH; added in 8.0.4 (nonreserved)
- MASTER_RETRY_COUNT
- MASTER_SERVER_ID
- MASTER_SSL
- MASTER_SSL_CA
- MASTER_SSL_CAPATH
- MASTER_SSL_CERT
- MASTER_SSL_CIPHER
- MASTER_SSL_CRL
- MASTER_SSL_CRLPATH
- MASTER_SSL_KEY
- MASTER_SSL_VERIFY_SERVER_CERT (R)
- MASTER_TLS_CIPHERSUITES; added in 8.0.19 (nonreserved)
- MASTER_TLS_VERSION
- MASTER_USER
- MASTER_ZSTD_COMPRESSION_LEVEL; added in 8.0.18 (nonreserved)
- MATCH (R)
- MAXVALUE (R)
- MAX_CONNECTIONS_PER_HOUR
- MAX_QUERIES_PER_HOUR
- MAX_ROWS
- MAX_SIZE

- MAX_UPDATES_PER_HOUR
- MAX_USER_CONNECTIONS
- MEDIUM
- MEDIUMBLOB (R)
- MEDIUMINT (R)
- MEDIUMTEXT (R)
- MEMBER (R); added in 8.0.17 (reserved)
- MEMORY
- MERGE
- MESSAGE_TEXT
- MICROSECOND
- MIDDLEINT (R)
- MIGRATE
- MINUTE
- MINUTE_MICROSECOND (R)
- MINUTE_SECOND (R)
- MIN_ROWS
- MOD (R)
- MODE
- MODIFIES (R)
- MODIFY
- MONTH
- MULTILINESTRING
- MULTIPOINT
- MULTIPOLYGON
- MUTEX
- MYSQL_ERRNO

N

- NAME
- NAMES

- NATIONAL
- NATURAL (R)
- NCHAR
- NDB
- NDBCLUSTER
- NESTED; added in 8.0.4 (nonreserved)
- NETWORK_NAMESPACE; added in 8.0.16 (nonreserved)
- NEVER
- NEW
- NEXT
- NO
- NODEGROUP
- NONE
- NOT (R)
- NOWAIT; added in 8.0.1 (nonreserved)
- NO_WAIT
- NO_WRITE_TO_BINLOG (R)
- NTH_VALUE (R); added in 8.0.2 (reserved)
- NTILE (R); added in 8.0.2 (reserved)
- NULL (R)
- NULLS; added in 8.0.2 (nonreserved)
- NUMBER
- NUMERIC (R)
- NVARCHAR

O

- OF (R); added in 8.0.1 (reserved)
- OFFSET
- OJ; added in 8.0.16 (nonreserved)
- OLD; added in 8.0.14 (nonreserved)
- ON (R)

- ONE
- ONLY
- OPEN
- OPTIMIZE (R)
- OPTIMIZER_COSTS (R)
- OPTION (R)
- OPTIONAL; added in 8.0.13 (nonreserved)
- OPTIONALLY (R)
- OPTIONS
- OR (R)
- ORDER (R)
- ORDINALITY; added in 8.0.4 (nonreserved)
- ORGANIZATION; added in 8.0.4 (nonreserved)
- OTHERS; added in 8.0.2 (nonreserved)
- OUT (R)
- OUTER (R)
- OUTFILE (R)
- OVER (R); added in 8.0.2 (reserved)
- OWNER

P

- PACK_KEYS
- PAGE
- PARSER
- PARTIAL
- PARTITION (R)
- PARTITIONING
- PARTITIONS
- PASSWORD
- PATH; added in 8.0.4 (nonreserved)
- PERCENT_RANK (R); added in 8.0.2 (reserved)

- PERSIST; became nonreserved in 8.0.16
- PERSIST_ONLY; added in 8.0.2 (reserved); became nonreserved in 8.0.16
- PHASE
- PLUGIN
- PLUGINS
- PLUGIN_DIR
- POINT
- POLYGON
- PORT
- PRECEDES
- PRECEDING; added in 8.0.2 (nonreserved)
- PRECISION (R)
- PREPARE
- PRESERVE
- PREV
- PRIMARY (R)
- PRIVILEGES
- PRIVILEGE_CHECKS_USER; added in 8.0.18 (nonreserved)
- PROCEDURE (R)
- PROCESS; added in 8.0.11 (nonreserved)
- PROCESSLIST
- PROFILE
- PROFILES
- PROXY
- PURGE (R)

Q

- QUARTER
- QUERY
- QUICK

R

- RANDOM; added in 8.0.18 (nonreserved)
- RANGE (R)
- RANK (R); added in 8.0.2 (reserved)
- READ (R)
- READS (R)
- READ_ONLY
- READ_WRITE (R)
- REAL (R)
- REBUILD
- RECOVER
- RECURSIVE (R); added in 8.0.1 (reserved)
- REDOFILE; removed in 8.0.3
- REDO_BUFFER_SIZE
- REDUNDANT
- REFERENCE; added in 8.0.4 (nonreserved)
- REFERENCES (R)
- REGEXP (R)
- RELAY
- RELAYLOG
- RELAY_LOG_FILE
- RELAY_LOG_POS
- RELAY_THREAD
- RELEASE (R)
- RELOAD
- REMOTE; added in 8.0.3 (nonreserved); removed in 8.0.14
- REMOVE
- RENAME (R)
- REORGANIZE
- REPAIR
- REPEAT (R)

- REPEATABLE
- REPLACE (R)
- REPLICATE_DO_DB
- REPLICATE_DO_TABLE
- REPLICATE_IGNORE_DB
- REPLICATE_IGNORE_TABLE
- REPLICATE_REWRITE_DB
- REPLICATE_WILD_DO_TABLE
- REPLICATE_WILD_IGNORE_TABLE
- REPLICATION
- REQUIRE (R)
- RESET
- RESIGNAL (R)
- RESOURCE; added in 8.0.3 (nonreserved)
- RESPECT; added in 8.0.2 (nonreserved)
- RESTART; added in 8.0.4 (nonreserved)
- RESTORE
- RESTRICT (R)
- RESUME
- RETAIN; added in 8.0.14 (nonreserved)
- RETURN (R)
- RETURNED_SQLSTATE
- RETURNS
- REUSE; added in 8.0.3 (nonreserved)
- REVERSE
- REVOKE (R)
- RIGHT (R)
- RLIKE (R)
- ROLE; became nonreserved in 8.0.1
- ROLLBACK

- ROLLUP
- ROTATE
- ROUTINE
- ROW (R); became reserved in 8.0.2
- ROWS (R); became reserved in 8.0.2
- ROW_COUNT
- ROW_FORMAT
- ROW_NUMBER (R); added in 8.0.2 (reserved)
- RTREE

S

- SAVEPOINT
- SCHEDULE
- SCHEMA (R)
- SCHEMAS (R)
- SCHEMA_NAME
- SECOND
- SECONDARY; added in 8.0.16 (nonreserved)
- SECONDARY_ENGINE; added in 8.0.13 (nonreserved)
- SECONDARY_LOAD; added in 8.0.13 (nonreserved)
- SECONDARY_UNLOAD; added in 8.0.13 (nonreserved)
- SECOND_MICROSECOND (R)
- SECURITY
- SELECT (R)
- SENSITIVE (R)
- SEPARATOR (R)
- SERIAL
- SERIALIZABLE
- SERVER
- SESSION
- SET (R)

- SHARE
- SHOW (R)
- SHUTDOWN
- SIGNAL (R)
- SIGNED
- SIMPLE
- SKIP; added in 8.0.1 (nonreserved)
- SLAVE
- SLOW
- SMALLINT (R)
- SNAPSHOT
- SOCKET
- SOME
- SONAME
- SOUNDS
- SOURCE
- SPATIAL (R)
- SPECIFIC (R)
- SQL (R)
- SQLEXCEPTION (R)
- SQLSTATE (R)
- SQLWARNING (R)
- SQL_AFTER_GTIDS
- SQL_AFTER_MTS_GAPS
- SQL_BEFORE_GTIDS
- SQL_BIG_RESULT (R)
- SQL_BUFFER_RESULT
- SQL_CACHE; removed in 8.0.3
- SQL_CALC_FOUND_ROWS (R)
- SQL_NO_CACHE

- SQL_SMALL_RESULT (R)
- SQL_THREAD
- SQL_TSI_DAY
- SQL_TSI_HOUR
- SQL_TSI_MINUTE
- SQL_TSI_MONTH
- SQL_TSI_QUARTER
- SQL_TSI_SECOND
- SQL_TSI_WEEK
- SQL_TSI_YEAR
- SRID; added in 8.0.3 (nonreserved)
- SSL (R)
- STACKED
- START
- STARTING (R)
- STARTS
- STATS_AUTO_RECALC
- STATS_PERSISTENT
- STATS_SAMPLE_PAGES
- STATUS
- STOP
- STORAGE
- STORED (R)
- STRAIGHT_JOIN (R)
- STRING
- SUBCLASS_ORIGIN
- SUBJECT
- SUBPARTITION
- SUBPARTITIONS
- SUPER

- SUSPEND
- SWAPS
- SWITCHES
- SYSTEM (R); added in 8.0.3 (reserved)

T

- TABLE (R)
- TABLES
- TABLESPACE
- TABLE_CHECKSUM
- TABLE_NAME
- TEMPORARY
- TEMPTABLE
- TERMINATED (R)
- TEXT
- THAN
- THEN (R)
- THREAD_PRIORITY; added in 8.0.3 (nonreserved)
- TIES; added in 8.0.2 (nonreserved)
- TIME
- TIMESTAMP
- TIMESTAMPADD
- TIMESTAMPDIFF
- TINYBLOB (R)
- TINYINT (R)
- TINYTEXT (R)
- TO (R)
- TRAILING (R)
- TRANSACTION
- TRIGGER (R)
- TRIGGERS

- TRUE (R)
- TRUNCATE
- TYPE
- TYPES

U

- UNBOUNDED; added in 8.0.2 (nonreserved)
- UNCOMMITTED
- UNDEFINED
- UNDO (R)
- UNDOFILE
- UNDO_BUFFER_SIZE
- UNICODE
- UNINSTALL
- UNION (R)
- UNIQUE (R)
- UNKNOWN
- UNLOCK (R)
- UNSIGNED (R)
- UNTIL
- UPDATE (R)
- UPGRADE
- USAGE (R)
- USE (R)
- USER
- USER_RESOURCES
- USE_FRM
- USING (R)
- UTC_DATE (R)
- UTC_TIME (R)
- UTC_TIMESTAMP (R)

V

- VALIDATION
- VALUE
- VALUES (R)
- VARBINARY (R)
- VARCHAR (R)
- VARCHARACTER (R)
- VARIABLES
- VARYING (R)
- VCPU; added in 8.0.3 (nonreserved)
- VIEW
- VIRTUAL (R)
- VISIBLE

W

- WAIT
- WARNINGS
- WEEK
- WEIGHT_STRING
- WHEN (R)
- WHERE (R)
- WHILE (R)
- WINDOW (R); added in 8.0.2 (reserved)
- WITH (R)
- WITHOUT
- WORK
- WRAPPER
- WRITE (R)

X

- X509
- XA

- XID
- XML
- XOR (R)

Y

- YEAR
- YEAR_MONTH (R)

Z

- ZEROFILL (R)

MySQL 8.0 New Keywords and Reserved Words

The following list shows the keywords and reserved words that are added in MySQL 8.0, compared to MySQL 5.7. Reserved keywords are marked with (R).

A | B | C | D | E | F | G | H | I | J | L | M | N | O | P | R | S | T | U | V | W

A

- ACTIVE
- ADMIN
- ARRAY (R)

B

- BUCKETS

C

- CLONE
- COMPONENT
- CUME_DIST (R)

D

- DEFINITION
- DENSE_RANK (R)
- DESCRIPTION

E

- EMPTY (R)
- ENFORCED
- EXCEPT (R)

- EXCLUDE

F

- FIRST_VALUE (R)
- FOLLOWING

G

- GEOMCOLLECTION
- GET_MASTER_PUBLIC_KEY
- GROUPING (R)
- GROUPS (R)

H

- HISTOGRAM
- HISTORY

I

- INACTIVE
- INVISIBLE

J

- JSON_TABLE (R)

L

- LAG (R)
- LAST_VALUE (R)
- LATERAL (R)
- LEAD (R)
- LOCKED

M

- MASTER_COMPRESSION_ALGORITHMS
- MASTER_PUBLIC_KEY_PATH
- MASTER_TLS_CIPHERSUITES
- MASTER_ZSTD_COMPRESSION_LEVEL
- MEMBER (R)

N

- NESTED
- NETWORK_NAMESPACE
- NOWAIT
- NTH_VALUE (R)
- NTILE (R)
- NULLS

O

- OF (R)
- OJ
- OLD
- OPTIONAL
- ORDINALITY
- ORGANIZATION
- OTHERS
- OVER (R)

P

- PATH
- PERCENT_RANK (R)
- PERSIST
- PERSIST_ONLY
- PRECEDING
- PRIVILEGE_CHECKS_USER
- PROCESS

R

- RANDOM
- RANK (R)
- RECURSIVE (R)
- REFERENCE
- RESOURCE
- RESPECT

- RESTART
- RETAIN
- REUSE
- ROLE
- ROW_NUMBER (R)

S

- SECONDARY
- SECONDARY_ENGINE
- SECONDARY_LOAD
- SECONDARY_UNLOAD
- SKIP
- SRID
- SYSTEM (R)

T

- THREAD_PRIORITY
- TIES

U

- UNBOUNDED

V

- VCPU
- VISIBLE

W

- WINDOW (R)

MySQL 8.0 Removed Keywords and Reserved Words

The following list shows the keywords and reserved words that are removed in MySQL 8.0, compared to MySQL 5.7. Reserved keywords are marked with (R).

- ANALYSE
- DES_KEY_FILE
- PARSE_GCOL_EXPR
- REDOFILE
- SQL_CACHE

9.4 User-Defined Variables

You can store a value in a user-defined variable in one statement and refer to it later in another statement. This enables you to pass values from one statement to another.

User variables are written as `@var_name`, where the variable name *var_name* consists of alphanumeric characters, `.`, `_`, and `$`. A user variable name can contain other characters if you quote it as a string or identifier (for example, `@'my-var'`, `@"my-var"`, or `@`my-var``).

User-defined variables are session specific. A user variable defined by one client cannot be seen or used by other clients. (Exception: A user with access to the Performance Schema `user_variables_by_thread` table can see all user variables for all sessions.) All variables for a given client session are automatically freed when that client exits.

User variable names are not case-sensitive. Names have a maximum length of 64 characters.

One way to set a user-defined variable is by issuing a `SET` statement:

```
SET @var_name = expr [, @var_name = expr] ...
```

For `SET`, either `=` or `:=` can be used as the assignment operator.

User variables can be assigned a value from a limited set of data types: integer, decimal, floating-point, binary or nonbinary string, or `NULL` value. Assignment of decimal and real values does not preserve the precision or scale of the value. A value of a type other than one of the permissible types is converted to a permissible type. For example, a value having a temporal or spatial data type is converted to a binary string. A value having the `JSON` data type is converted to a string with a character set of `utf8mb4` and a collation of `utf8mb4_bin`.

If a user variable is assigned a nonbinary (character) string value, it has the same character set and collation as the string. The coercibility of user variables is implicit. (This is the same coercibility as for table column values.)

Hexadecimal or bit values assigned to user variables are treated as binary strings. To assign a hexadecimal or bit value as a number to a user variable, use it in numeric context. For example, add 0 or use `CAST(... AS UNSIGNED)`:

```
mysql> SET @v1 = X'41';
mysql> SET @v2 = X'41'+0;
mysql> SET @v3 = CAST(X'41' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+
mysql> SET @v1 = b'1000001';
mysql> SET @v2 = b'1000001'+0;
mysql> SET @v3 = CAST(b'1000001' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+
```

If the value of a user variable is selected in a result set, it is returned to the client as a string.

If you refer to a variable that has not been initialized, it has a value of `NULL` and a type of string.

User variables may be used in most contexts where expressions are permitted. This does not currently include contexts that explicitly require a literal value, such as in the `LIMIT` clause of a `SELECT` statement, or the `IGNORE N LINES` clause of a `LOAD DATA` statement.

Previous releases of MySQL made it possible to assign a value to a user variable in statements other than `SET`. This functionality is supported in MySQL 8.0 for backward compatibility but is subject to removal in a future release of MySQL.

When making an assignment in this way, you must use `:=` as the assignment operator; `=` is treated as the comparison operator in statements other than `SET`.

The order of evaluation for expressions involving user variables is undefined. For example, there is no guarantee that `SELECT @a, @a:=@a+1` evaluates `@a` first and then performs the assignment.

In addition, the default result type of a variable is based on its type at the beginning of the statement. This may have unintended effects if a variable holds a value of one type at the beginning of a statement in which it is also assigned a new value of a different type.

To avoid problems with this behavior, either do not assign a value to and read the value of the same variable within a single statement, or else set the variable to `0`, `0.0`, or `''` to define its type before you use it.

`HAVING`, `GROUP BY`, and `ORDER BY`, when referring to a variable that is assigned a value in the select expression list do not work as expected because the expression is evaluated on the client and thus can use stale column values from a previous row.

User variables are intended to provide data values. They cannot be used directly in an SQL statement as an identifier or as part of an identifier, such as in contexts where a table or database name is expected, or as a reserved word such as `SELECT`. This is true even if the variable is quoted, as shown in the following example:

```
mysql> SELECT c1 FROM t;
+----+
| c1 |
+----+
|  0 |
+----+
|  1 |
+----+
2 rows in set (0.00 sec)

mysql> SET @col = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+------+
| @col |
+------+
| c1   |
+------+
1 row in set (0.00 sec)

mysql> SELECT `@col` FROM t;
ERROR 1054 (42S22): Unknown column '@col' in 'field list'

mysql> SET @col = "`c1`";
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @col FROM t;
+------+
| @col |
+------+
| `c1` |
+------+
1 row in set (0.00 sec)
```

An exception to this principle that user variables cannot be used to provide identifiers, is when you are constructing a string for use as a prepared statement to execute later. In this case, user variables can be used to provide any part of the statement. The following example illustrates how this can be done:

```
mysql> SET @c = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SET @s = CONCAT("SELECT ", @c, " FROM t");
Query OK, 0 rows affected (0.00 sec)

mysql> PREPARE stmt FROM @s;
Query OK, 0 rows affected (0.04 sec)
Statement prepared

mysql> EXECUTE stmt;
+----+
| c1 |
+----+
|  0 |
+----+
|  1 |
+----+
2 rows in set (0.00 sec)

mysql> DEALLOCATE PREPARE stmt;
Query OK, 0 rows affected (0.00 sec)
```

See Section 13.5, “Prepared SQL Statement Syntax”, for more information.

A similar technique can be used in application programs to construct SQL statements using program variables, as shown here using PHP 5:

```
<?php
  $mysqli = new mysqli("localhost", "user", "pass", "test");

  if( mysqli_connect_errno() )
    die("Connection failed: %s\n", mysqli_connect_error());

  $col = "c1";

  $query = "SELECT $col FROM t";

  $result = $mysqli->query($query);

  while($row = $result->fetch_assoc())
  {
    echo "<p>" . $row["$col"] . "</p>\n";
  }

  $result->close();

  $mysqli->close();
?>
```

Assembling an SQL statement in this fashion is sometimes known as “Dynamic SQL”.

9.5 Expressions

This section lists the grammar rules that expressions must follow in MySQL and provides additional information about the types of terms that may appear in expressions.

- Expression Syntax
- Expression Term Notes
- Temporal Intervals

Expression Syntax

The following grammar rules define expression syntax in MySQL. The grammar shown here is based on that given in the `sql/sql_yacc.yy` file of MySQL source distributions. For additional information about some of the expression terms, see Expression Term Notes.

```
expr:
    expr OR expr
  | expr || expr
  | expr XOR expr
  | expr AND expr
  | expr && expr
  | NOT expr
  | ! expr
  | boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}
  | boolean_primary

boolean_primary:
    boolean_primary IS [NOT] NULL
  | boolean_primary <=> predicate
  | boolean_primary comparison_operator predicate
  | boolean_primary comparison_operator {ALL | ANY} (subquery)
  | predicate

comparison_operator: = | >= | > | <= | < | <> | !=

predicate:
    bit_expr [NOT] IN (subquery)
  | bit_expr [NOT] IN (expr [, expr] ...)
  | bit_expr [NOT] BETWEEN bit_expr AND predicate
  | bit_expr SOUNDS LIKE bit_expr
  | bit_expr [NOT] LIKE simple_expr [ESCAPE simple_expr]
  | bit_expr [NOT] REGEXP bit_expr
  | bit_expr

bit_expr:
    bit_expr | bit_expr
  | bit_expr & bit_expr
  | bit_expr << bit_expr
  | bit_expr >> bit_expr
  | bit_expr + bit_expr
  | bit_expr - bit_expr
  | bit_expr * bit_expr
  | bit_expr / bit_expr
  | bit_expr DIV bit_expr
  | bit_expr MOD bit_expr
  | bit_expr % bit_expr
  | bit_expr ^ bit_expr
  | bit_expr + interval_expr
  | bit_expr - interval_expr
  | simple_expr
```

```
simple_expr:
    literal
  | identifier
  | function_call
  | simple_expr COLLATE collation_name
  | param_marker
  | variable
  | simple_expr || simple_expr
  | + simple_expr
  | - simple_expr
  | ~ simple_expr
  | ! simple_expr
  | BINARY simple_expr
  | (expr [, expr] ...)
  | ROW (expr, expr [, expr] ...)
  | (subquery)
  | EXISTS (subquery)
  | {identifier expr}
  | match_expr
  | case_expr
  | interval_expr
```

For operator precedence, see Section 12.3.1, “Operator Precedence”. The precedence and meaning of some operators depends on the SQL mode:

- By default, `||` is a logical `OR` operator. With `PIPES_AS_CONCAT` enabled, `||` is string concatenation, with a precedence between `^` and the unary operators.
- By default, `!` has a higher precedence than `NOT`. With `HIGH_NOT_PRECEDENCE` enabled, `!` and `NOT` have the same precedence.

See Section 5.1.11, “Server SQL Modes”.

Expression Term Notes

For literal value syntax, see Section 9.1, “Literal Values”.

For identifier syntax, see Section 9.2, “Schema Object Names”.

Variables can be user variables, system variables, or stored program local variables or parameters:

- User variables: Section 9.4, “User-Defined Variables”
- System variables: Section 5.1.9, “Using System Variables”
- Stored program local variables: Section 13.6.4.1, “Local Variable DECLARE Syntax”
- Stored program parameters: Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”

`param_marker` is `?` as used in prepared statements for placeholders. See Section 13.5.1, “PREPARE Syntax”.

`(`*`subquery`*`)` indicates a subquery that returns a single value; that is, a scalar subquery. See Section 13.2.11.1, “The Subquery as Scalar Operand”.

`{`*`identifier expr`*`}` is ODBC escape syntax and is accepted for ODBC compatibility. The value is *`expr`*. The `{` and `}` curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

`match_expr` indicates a `MATCH` expression. See Section 12.9, “Full-Text Search Functions”.

`case_expr` indicates a `CASE` expression. See Section 12.4, “Control Flow Functions”.

`interval_expr` represents a temporal interval. See Temporal Intervals.

Temporal Intervals

`interval_expr` in expressions represents a temporal interval. Intervals have this syntax:

```
INTERVAL expr unit
```

`expr` represents a quantity. *`unit`* represents the unit for interpreting the quantity; it is a specifier such as `HOUR`, `DAY`, or `WEEK`. The `INTERVAL` keyword and the *`unit`* specifier are not case sensitive.

The following table shows the expected form of the *`expr`* argument for each *`unit`* value.

Table 9.2 Temporal Interval Expression and Unit Arguments

***`unit`* Value**	**Expected *`expr`* Format**
`MICROSECOND`	`MICROSECONDS`
`SECOND`	`SECONDS`
`MINUTE`	`MINUTES`
`HOUR`	`HOURS`
`DAY`	`DAYS`
`WEEK`	`WEEKS`
`MONTH`	`MONTHS`
`QUARTER`	`QUARTERS`
`YEAR`	`YEARS`
`SECOND_MICROSECOND`	`'SECONDS.MICROSECONDS'`
`MINUTE_MICROSECOND`	`'MINUTES:SECONDS.MICROSECONDS'`
`MINUTE_SECOND`	`'MINUTES:SECONDS'`
`HOUR_MICROSECOND`	`'HOURS:MINUTES:SECONDS.MICROSECONDS'`
`HOUR_SECOND`	`'HOURS:MINUTES:SECONDS'`
`HOUR_MINUTE`	`'HOURS:MINUTES'`
`DAY_MICROSECOND`	`'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'`
`DAY_SECOND`	`'DAYS HOURS:MINUTES:SECONDS'`
`DAY_MINUTE`	`'DAYS HOURS:MINUTES'`
`DAY_HOUR`	`'DAYS HOURS'`
`YEAR_MONTH`	`'YEARS-MONTHS'`

MySQL permits any punctuation delimiter in the *`expr`* format. Those shown in the table are the suggested delimiters.

Temporal intervals are used for certain functions, such as `DATE_ADD()` and `DATE_SUB()`:

```
mysql> SELECT DATE_ADD('2018-05-01',INTERVAL 1 DAY);
        -> '2018-05-02'
```

```
mysql> SELECT DATE_SUB('2018-05-01',INTERVAL 1 YEAR);
        -> '2017-05-01'
mysql> SELECT DATE_ADD('2020-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '2021-01-01 00:00:00'
mysql> SELECT DATE_ADD('2018-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '2019-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
    ->                 INTERVAL '1:1' MINUTE_SECOND);
        -> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2025-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '2024-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
    ->                 INTERVAL '-1 10' DAY_HOUR);
        -> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
    ->            INTERVAL '1.999999' SECOND_MICROSECOND);
        -> '1993-01-01 00:00:01.000001'
```

Temporal arithmetic also can be performed in expressions using `INTERVAL` together with the `+` or `-` operator:

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

`INTERVAL` *`expr unit`* is permitted on either side of the `+` operator if the expression on the other side is a date or datetime value. For the `-` operator, `INTERVAL` *`expr unit`* is permitted only on the right side, because it makes no sense to subtract a date or datetime value from an interval.

```
mysql> SELECT '2018-12-31 23:59:59' + INTERVAL 1 SECOND;
        -> '2019-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '2018-12-31';
        -> '2019-01-01'
mysql> SELECT '2025-01-01' - INTERVAL 1 SECOND;
        -> '2024-12-31 23:59:59'
```

The `EXTRACT()` function uses the same kinds of *`unit`* specifiers as `DATE_ADD()` or `DATE_SUB()`, but extracts parts from the date rather than performing date arithmetic:

```
mysql> SELECT EXTRACT(YEAR FROM '2019-07-02');
        -> 2019
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2019-07-02 01:02:03');
        -> 201907
```

Temporal intervals can be used in `CREATE EVENT` statements:

```
CREATE EVENT myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the *`unit`* keyword), MySQL assumes that you have left out the leftmost parts of the interval value. For example, if you specify a *`unit`* of `DAY_SECOND`, the value of *`expr`* is expected to have days, hours, minutes, and seconds parts. If you specify a value like `'1:10'`, MySQL assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, `'1:10'`

`DAY_SECOND` is interpreted in such a way that it is equivalent to `'1:10' MINUTE_SECOND`. This is analogous to the way that MySQL interprets `TIME` values as representing elapsed time rather than as a time of day.

`expr` is treated as a string, so be careful if you specify a nonstring value with `INTERVAL`. For example, with an interval specifier of `HOUR_MINUTE`, '6/4' is treated as 6 hours, four minutes, whereas `6/4` evaluates to `1.5000` and is treated as 1 hour, 5000 minutes:

```
mysql> SELECT '6/4', 6/4;
        -> 1.5000
mysql> SELECT DATE_ADD('2019-01-01', INTERVAL '6/4' HOUR_MINUTE);
        -> '2019-01-01 06:04:00'
mysql> SELECT DATE_ADD('2019-01-01', INTERVAL 6/4 HOUR_MINUTE);
        -> '2019-01-04 12:20:00'
```

To ensure interpretation of the interval value as you expect, a `CAST()` operation may be used. To treat `6/4` as 1 hour, 5 minutes, cast it to a `DECIMAL` value with a single fractional digit:

```
mysql> SELECT CAST(6/4 AS DECIMAL(3,1));
        -> 1.5
mysql> SELECT DATE_ADD('1970-01-01 12:00:00',
    ->                 INTERVAL CAST(6/4 AS DECIMAL(3,1)) HOUR_MINUTE);
        -> '1970-01-01 13:05:00'
```

If you add to or subtract from a date value something that contains a time part, the result is automatically converted to a datetime value:

```
mysql> SELECT DATE_ADD('2023-01-01', INTERVAL 1 DAY);
        -> '2023-01-02'
mysql> SELECT DATE_ADD('2023-01-01', INTERVAL 1 HOUR);
        -> '2023-01-01 01:00:00'
```

If you add `MONTH`, `YEAR_MONTH`, or `YEAR` and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month:

```
mysql> SELECT DATE_ADD('2019-01-30', INTERVAL 1 MONTH);
        -> '2019-02-28'
```

Date arithmetic operations require complete dates and do not work with incomplete dates such as `'2016-07-00'` or badly malformed dates:

```
mysql> SELECT DATE_ADD('2016-07-00', INTERVAL 1 DAY);
        -> NULL
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;
        -> NULL
```

9.6 Comment Syntax

MySQL Server supports three comment styles:

- From a `#` character to the end of the line.
- From a `--` sequence to the end of the line. In MySQL, the `--` (double-dash) comment style requires the second dash to be followed by at least one whitespace or control character (such as a space, tab, newline, and so on). This syntax differs slightly from standard SQL comment syntax, as discussed in Section 1.8.2.4, “'--' as the Start of a Comment”.

- From a `/*` sequence to the following `*/` sequence, as in the C programming language. This syntax enables a comment to extend over multiple lines because the beginning and closing sequences need not be on the same line.

The following example demonstrates all three comment styles:

```
mysql> SELECT 1+1;     # This comment continues to the end of line
mysql> SELECT 1+1;     -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

Nested comments are not supported, are deprecated, and will be removed in a future MySQL release. (Under some conditions, nested comments might be permitted, but usually are not, and users should avoid them.)

MySQL Server supports certain variants of C-style comments. These enable you to write code that includes MySQL extensions, but is still portable, by using comments of the following form:

```
/*! MySQL-specific code */
```

In this case, MySQL Server parses and executes the code within the comment as it would any other SQL statement, but other SQL servers will ignore the extensions. For example, MySQL Server recognizes the `STRAIGHT_JOIN` keyword in the following statement, but other servers will not:

```
SELECT /*! STRAIGHT_JOIN */ col1 FROM table1,table2 WHERE ...
```

If you add a version number after the `!` character, the syntax within the comment is executed only if the MySQL version is greater than or equal to the specified version number. The `KEY_BLOCK_SIZE` keyword in the following comment is executed only by servers from MySQL 5.1.10 or higher:

```
CREATE TABLE t1(a INT, KEY (a)) /*!50110 KEY_BLOCK_SIZE=1024 */;
```

The comment syntax just described applies to how the `mysqld` server parses SQL statements. The `mysql` client program also performs some parsing of statements before sending them to the server. (It does this to determine statement boundaries within a multiple-statement input line.) For information about differences between the server and `mysql` client parsers, see Section 4.5.1.6, “mysql Client Tips”.

Comments in `/*!12345 ... */` format are not stored on the server. If this format is used to comment stored programs, the comments are not retained in the program body.

Another variant of C-style comment syntax is used to specify optimizer hints. Hint comments include a `+` character following the `/*` comment opening sequence. Example:

```
SELECT /*+ BKA(t1) */ FROM ... ;
```

For more information, see Section 8.9.3, “Optimizer Hints”.

The use of short-form `mysql` commands such as `\C` within multiple-line `/* ... */` comments is not supported. Short-form commands do work within single-line `/*! ... */` version comments, as do `/*+ ... */` optimizer-hint comments, which are stored in object definitions. If there is a concern that

optimizer-hint comments may be stored in object definitions so that dump files when reloaded with `mysql` would result in execution of such commands, either invoke `mysql` with the `--binary-mode` option or use a reload client other than `mysql`.

Chapter 10 Character Sets, Collations, Unicode

Table of Contents

10.1 Character Sets and Collations in General 1732
10.2 Character Sets and Collations in MySQL 1733
10.2.1 Character Set Repertoire 1735
10.2.2 UTF-8 for Metadata 1737
10.3 Specifying Character Sets and Collations 1739
10.3.1 Collation Naming Conventions 1739
10.3.2 Server Character Set and Collation 1740
10.3.3 Database Character Set and Collation 1741
10.3.4 Table Character Set and Collation 1742
10.3.5 Column Character Set and Collation 1743
10.3.6 Character String Literal Character Set and Collation 1744
10.3.7 The National Character Set 1746
10.3.8 Character Set Introducers 1747
10.3.9 Examples of Character Set and Collation Assignment 1749
10.3.10 Compatibility with Other DBMSs 1750
10.4 Connection Character Sets and Collations 1750
10.5 Configuring Application Character Set and Collation 1757
10.6 Error Message Character Set 1758
10.7 Column Character Set Conversion 1759
10.8 Collation Issues 1760
10.8.1 Using COLLATE in SQL Statements 1761
10.8.2 COLLATE Clause Precedence 1761
10.8.3 Character Set and Collation Compatibility 1762
10.8.4 Collation Coercibility in Expressions 1762
10.8.5 The binary Collation Compared to _bin Collations 1763
10.8.6 Examples of the Effect of Collation 1766
10.8.7 Using Collation in INFORMATION_SCHEMA Searches 1767
10.9 Unicode Support 1769
10.9.1 The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding) 1771
10.9.2 The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding) 1772
10.9.3 The utf8 Character Set (Alias for utf8mb3) 1773
10.9.4 The ucs2 Character Set (UCS-2 Unicode Encoding) 1773
10.9.5 The utf16 Character Set (UTF-16 Unicode Encoding) 1773
10.9.6 The utf16le Character Set (UTF-16LE Unicode Encoding) 1774
10.9.7 The utf32 Character Set (UTF-32 Unicode Encoding) 1774
10.9.8 Converting Between 3-Byte and 4-Byte Unicode Character Sets 1774
10.10 Supported Character Sets and Collations 1777
10.10.1 Unicode Character Sets 1778
10.10.2 West European Character Sets 1785
10.10.3 Central European Character Sets 1787
10.10.4 South European and Middle East Character Sets 1788
10.10.5 Baltic Character Sets 1788
10.10.6 Cyrillic Character Sets 1789
10.10.7 Asian Character Sets 1789
10.10.8 The Binary Character Set 1794
10.11 Restrictions on Character Sets 1795
10.12 Setting the Error Message Language 1795
10.13 Adding a Character Set 1796

10.13.1 Character Definition Arrays 1798
10.13.2 String Collating Support for Complex Character Sets 1799
10.13.3 Multi-Byte Character Support for Complex Character Sets 1799
10.14 Adding a Collation to a Character Set 1799
10.14.1 Collation Implementation Types 1801
10.14.2 Choosing a Collation ID 1803
10.14.3 Adding a Simple Collation to an 8-Bit Character Set 1804
10.14.4 Adding a UCA Collation to a Unicode Character Set 1805
10.15 Character Set Configuration 1813
10.16 MySQL Server Locale Support 1814

MySQL includes character set support that enables you to store data using a variety of character sets and perform comparisons according to a variety of collations. You can specify character sets at the server, database, table, and column level.

This chapter discusses the following topics:

- What are character sets and collations?
- The multiple-level default system for character set assignment.
- Syntax for specifying character sets and collations.
- Affected functions and operations.
- Unicode support.
- The character sets and collations that are available, with notes.
- Selecting the language for error messages.
- Selecting the locale for day and month names.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the `utf8` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about configuring character sets for application use and character set-related issues in client/server communication, see Section 10.5, “Configuring Application Character Set and Collation”, and Section 10.4, “Connection Character Sets and Collations”.

10.1 Character Sets and Collations in General

A *character set* is a set of symbols and encodings. A *collation* is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: A, B, a, b. We give each letter a number: A = 0, B = 1, a = 2, b = 3. The letter A is a symbol, the number 0 is the *encoding* for A, and the combination of all four letters and their encodings is a *character set*.

Suppose that we want to compare two string values, A and B. The simplest way to do this is to look at the encodings: 0 for A and 1 for B. Because 0 is less than 1, we say A is less than B. What we've just done is

apply a collation to our character set. The collation is a set of rules (only one rule in this case): “compare the encodings.” We call this simplest of all possible collations a *binary* collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters `a` and `b` as equivalent to `A` and `B`; (2) then compare the encodings. We call this a *case-insensitive* collation. It is a little more complex than a binary collation.

In real life, most character sets have many characters: not just `A` and `B` but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with many special symbols and punctuation marks. Also in real life, most collations have many rules, not just for whether to distinguish lettercase, but also for whether to distinguish accents (an “accent” is a mark attached to a character as in German `Ö`), and for multiple-character mappings (such as the rule that `Ö` = `OE` in one of the two German collations).

MySQL can do these things for you:

- Store strings using a variety of character sets.
- Compare strings using a variety of collations.
- Mix strings with different character sets or collations in the same server, the same database, or even the same table.
- Enable specification of character set and collation at any level.

To use these features effectively, you must know what character sets and collations are available, how to change the defaults, and how they affect the behavior of string operators and functions.

10.2 Character Sets and Collations in MySQL

MySQL Server supports multiple character sets, including several Unicode character sets. To display the available character sets, use the `INFORMATION_SCHEMA CHARACTER_SETS` table or the `SHOW CHARACTER SET` statement. A partial listing follows. For more complete information, see Section 10.10, “Supported Character Sets and Collations”.

```
mysql> SHOW CHARACTER SET;
+----------+---------------------------------+---------------------+--------+
| Charset  | Description                     | Default collation   | Maxlen |
+----------+---------------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese        | big5_chinese_ci     |      2 |
| binary   | Binary pseudo charset           | binary              |      1 |
...
| latin1   | cp1252 West European            | latin1_swedish_ci   |      1 |
...
| ucs2     | UCS-2 Unicode                   | ucs2_general_ci     |      2 |
...
| utf8     | UTF-8 Unicode                   | utf8_general_ci     |      3 |
| utf8mb4  | UTF-8 Unicode                   | utf8mb4_0900_ai_ci  |      4 |
...
```

By default, the `SHOW CHARACTER SET` statement displays all available character sets. It takes an optional `LIKE` or `WHERE` clause that indicates which character set names to match. The following example shows some of the Unicode character sets (those based on Unicode Transformation Format):

```
mysql> SHOW CHARACTER SET LIKE 'utf%';
+---------+-----------------+--------------------+--------+
| Charset | Description     | Default collation  | Maxlen |
+---------+-----------------+--------------------+--------+
| utf16   | UTF-16 Unicode  | utf16_general_ci   |      4 |
```

```
| utf16le | UTF-16LE Unicode | utf16le_general_ci |      4 |
| utf32   | UTF-32 Unicode   | utf32_general_ci   |      4 |
| utf8    | UTF-8 Unicode    | utf8_general_ci    |      3 |
| utf8mb4 | UTF-8 Unicode    | utf8mb4_0900_ai_ci |      4 |
+---------+------------------+--------------------+--------+
```

A given character set always has at least one collation, and most character sets have several. To list the display collations for a character set, use the `INFORMATION_SCHEMA COLLATIONS` table or the `SHOW COLLATION` statement.

By default, the `SHOW COLLATION` statement displays all available collations. It takes an optional `LIKE` or `WHERE` clause that indicates which collation names to display. For example, to see the collations for the default character set, `utf8mb4`, use this statement:

```
mysql> SHOW COLLATION WHERE Charset = 'utf8mb4';
+----------------------------+---------+-----+---------+----------+---------+---------------+
| Collation                  | Charset | Id  | Default | Compiled | Sortlen | Pad_attribute |
+----------------------------+---------+-----+---------+----------+---------+---------------+
| utf8mb4_0900_ai_ci         | utf8mb4 | 255 | Yes     | Yes      |       0 | NO PAD        |
| utf8mb4_0900_as_ci         | utf8mb4 | 305 |         | Yes      |       0 | NO PAD        |
| utf8mb4_0900_as_cs         | utf8mb4 | 278 |         | Yes      |       0 | NO PAD        |
| utf8mb4_0900_bin           | utf8mb4 | 309 |         | Yes      |       1 | NO PAD        |
| utf8mb4_bin                | utf8mb4 |  46 |         | Yes      |       1 | PAD SPACE     |
| utf8mb4_croatian_ci        | utf8mb4 | 245 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_cs_0900_ai_ci      | utf8mb4 | 266 |         | Yes      |       0 | NO PAD        |
| utf8mb4_cs_0900_as_cs      | utf8mb4 | 289 |         | Yes      |       0 | NO PAD        |
| utf8mb4_czech_ci           | utf8mb4 | 234 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_danish_ci          | utf8mb4 | 235 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_da_0900_ai_ci      | utf8mb4 | 267 |         | Yes      |       0 | NO PAD        |
| utf8mb4_da_0900_as_cs      | utf8mb4 | 290 |         | Yes      |       0 | NO PAD        |
| utf8mb4_de_pb_0900_ai_ci   | utf8mb4 | 256 |         | Yes      |       0 | NO PAD        |
| utf8mb4_de_pb_0900_as_cs   | utf8mb4 | 279 |         | Yes      |       0 | NO PAD        |
| utf8mb4_eo_0900_ai_ci      | utf8mb4 | 273 |         | Yes      |       0 | NO PAD        |
| utf8mb4_eo_0900_as_cs      | utf8mb4 | 296 |         | Yes      |       0 | NO PAD        |
| utf8mb4_esperanto_ci       | utf8mb4 | 241 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_estonian_ci        | utf8mb4 | 230 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_es_0900_ai_ci      | utf8mb4 | 263 |         | Yes      |       0 | NO PAD        |
| utf8mb4_es_0900_as_cs      | utf8mb4 | 286 |         | Yes      |       0 | NO PAD        |
| utf8mb4_es_trad_0900_ai_ci | utf8mb4 | 270 |         | Yes      |       0 | NO PAD        |
| utf8mb4_es_trad_0900_as_cs | utf8mb4 | 293 |         | Yes      |       0 | NO PAD        |
| utf8mb4_et_0900_ai_ci      | utf8mb4 | 262 |         | Yes      |       0 | NO PAD        |
| utf8mb4_et_0900_as_cs      | utf8mb4 | 285 |         | Yes      |       0 | NO PAD        |
| utf8mb4_general_ci         | utf8mb4 |  45 |         | Yes      |       1 | PAD SPACE     |
| utf8mb4_german2_ci         | utf8mb4 | 244 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_hr_0900_ai_ci      | utf8mb4 | 275 |         | Yes      |       0 | NO PAD        |
| utf8mb4_hr_0900_as_cs      | utf8mb4 | 298 |         | Yes      |       0 | NO PAD        |
| utf8mb4_hungarian_ci       | utf8mb4 | 242 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_hu_0900_ai_ci      | utf8mb4 | 274 |         | Yes      |       0 | NO PAD        |
| utf8mb4_hu_0900_as_cs      | utf8mb4 | 297 |         | Yes      |       0 | NO PAD        |
| utf8mb4_icelandic_ci       | utf8mb4 | 225 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_is_0900_ai_ci      | utf8mb4 | 257 |         | Yes      |       0 | NO PAD        |
| utf8mb4_is_0900_as_cs      | utf8mb4 | 280 |         | Yes      |       0 | NO PAD        |
| utf8mb4_ja_0900_as_cs      | utf8mb4 | 303 |         | Yes      |       0 | NO PAD        |
| utf8mb4_ja_0900_as_cs_ks   | utf8mb4 | 304 |         | Yes      |      24 | NO PAD        |
| utf8mb4_latvian_ci         | utf8mb4 | 226 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_la_0900_ai_ci      | utf8mb4 | 271 |         | Yes      |       0 | NO PAD        |
| utf8mb4_la_0900_as_cs      | utf8mb4 | 294 |         | Yes      |       0 | NO PAD        |
| utf8mb4_lithuanian_ci      | utf8mb4 | 236 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_lt_0900_ai_ci      | utf8mb4 | 268 |         | Yes      |       0 | NO PAD        |
| utf8mb4_lt_0900_as_cs      | utf8mb4 | 291 |         | Yes      |       0 | NO PAD        |
| utf8mb4_lv_0900_ai_ci      | utf8mb4 | 258 |         | Yes      |       0 | NO PAD        |
| utf8mb4_lv_0900_as_cs      | utf8mb4 | 281 |         | Yes      |       0 | NO PAD        |
| utf8mb4_persian_ci         | utf8mb4 | 240 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_pl_0900_ai_ci      | utf8mb4 | 261 |         | Yes      |       0 | NO PAD        |
| utf8mb4_pl_0900_as_cs      | utf8mb4 | 284 |         | Yes      |       0 | NO PAD        |
```

```
| utf8mb4_polish_ci          | utf8mb4 | 229 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_romanian_ci        | utf8mb4 | 227 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_roman_ci           | utf8mb4 | 239 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_ro_0900_ai_ci      | utf8mb4 | 259 |         | Yes      |       0 | NO PAD        |
| utf8mb4_ro_0900_as_cs      | utf8mb4 | 282 |         | Yes      |       0 | NO PAD        |
| utf8mb4_ru_0900_ai_ci      | utf8mb4 | 306 |         | Yes      |       0 | NO PAD        |
| utf8mb4_ru_0900_as_cs      | utf8mb4 | 307 |         | Yes      |       0 | NO PAD        |
| utf8mb4_sinhala_ci         | utf8mb4 | 243 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_sk_0900_ai_ci      | utf8mb4 | 269 |         | Yes      |       0 | NO PAD        |
| utf8mb4_sk_0900_as_cs      | utf8mb4 | 292 |         | Yes      |       0 | NO PAD        |
| utf8mb4_slovak_ci          | utf8mb4 | 237 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_slovenian_ci       | utf8mb4 | 228 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_sl_0900_ai_ci      | utf8mb4 | 260 |         | Yes      |       0 | NO PAD        |
| utf8mb4_sl_0900_as_cs      | utf8mb4 | 283 |         | Yes      |       0 | NO PAD        |
| utf8mb4_spanish2_ci        | utf8mb4 | 238 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_spanish_ci         | utf8mb4 | 231 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_sv_0900_ai_ci      | utf8mb4 | 264 |         | Yes      |       0 | NO PAD        |
| utf8mb4_sv_0900_as_cs      | utf8mb4 | 287 |         | Yes      |       0 | NO PAD        |
| utf8mb4_swedish_ci         | utf8mb4 | 232 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_tr_0900_ai_ci      | utf8mb4 | 265 |         | Yes      |       0 | NO PAD        |
| utf8mb4_tr_0900_as_cs      | utf8mb4 | 288 |         | Yes      |       0 | NO PAD        |
| utf8mb4_turkish_ci         | utf8mb4 | 233 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_unicode_520_ci     | utf8mb4 | 246 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_unicode_ci         | utf8mb4 | 224 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_vietnamese_ci      | utf8mb4 | 247 |         | Yes      |       8 | PAD SPACE     |
| utf8mb4_vi_0900_ai_ci      | utf8mb4 | 277 |         | Yes      |       0 | NO PAD        |
| utf8mb4_vi_0900_as_cs      | utf8mb4 | 300 |         | Yes      |       0 | NO PAD        |
| utf8mb4_zh_0900_as_cs      | utf8mb4 | 308 |         | Yes      |       0 | NO PAD        |
+----------------------------+---------+-----+---------+----------+---------+---------------+
```

For more information about those collations, see Section 10.10.1, “Unicode Character Sets”.

Collations have these general characteristics:

- Two different character sets cannot have the same collation.
- Each character set has a *default collation*. For example, the default collations for `utf8mb4` and `latin1` are `utf8mb4_0900_ai_ci` and `latin1_swedish_ci`, respectively. The `INFORMATION_SCHEMA CHARACTER_SETS` table and the `SHOW CHARACTER SET` statement indicate the default collation for each character set. The `INFORMATION_SCHEMA COLLATIONS` table and the `SHOW COLLATION` statement have a column that indicates for each collation whether it is the default for its character set (`Yes` if so, empty if not).
- Collation names start with the name of the character set with which they are associated, generally followed by one or more suffixes indicating other collation characteristics. For additional information about naming conventions, see Section 10.3.1, “Collation Naming Conventions”.

When a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing an inappropriate collation, perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

10.2.1 Character Set Repertoire

The *repertoire* of a character set is the collection of characters in the set.

String expressions have a repertoire attribute, which can have two values:

- `ASCII`: The expression can contain only characters in the Unicode range `U+0000` to `U+007F`.
- `UNICODE`: The expression can contain characters in the Unicode range `U+0000` to `U+10FFFF`. This includes characters in the Basic Multilingual Plane (BMP) range (`U+0000` to `U+FFFF`) and supplementary characters outside the BMP range (`U+10000` to `U+10FFFF`).

The ASCII range is a subset of UNICODE range, so a string with ASCII repertoire can be converted safely without loss of information to the character set of any string with UNICODE repertoire or to a character set that is a superset of ASCII. (All MySQL character sets are supersets of ASCII with the exception of swe7, which reuses some punctuation characters for Swedish accented characters.) The use of repertoire enables character set conversion in expressions for many cases where MySQL would otherwise return an "illegal mix of collations" error.

The following discussion provides examples of expressions and their repertoires, and describes how the use of repertoire changes string expression evaluation:

- The repertoire for a string constant depends on string content and may differ from the repertoire of the string character set. Consider these statements:

  ```
  SET NAMES utf8; SELECT 'abc';
  SELECT _utf8'def';
  SELECT N'MySQL';
  ```

  Although the character set is utf8 in each of the preceding cases, the strings do not actually contain any characters outside the ASCII range, so their repertoire is ASCII rather than UNICODE.

- A column having the ascii character set has ASCII repertoire because of its character set. In the following table, c1 has ASCII repertoire:

  ```
  CREATE TABLE t1 (c1 CHAR(1) CHARACTER SET ascii);
  ```

  The following example illustrates how repertoire enables a result to be determined in a case where an error occurs without repertoire:

  ```
  CREATE TABLE t1 (
    c1 CHAR(1) CHARACTER SET latin1,
    c2 CHAR(1) CHARACTER SET ascii
  );
  INSERT INTO t1 VALUES ('a','b');
  SELECT CONCAT(c1,c2) FROM t1;
  ```

  Without repertoire, this error occurs:

  ```
  ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
  and (ascii_general_ci,IMPLICIT) for operation 'concat'
  ```

  Using repertoire, subset to superset (ascii to latin1) conversion can occur and a result is returned:

  ```
  +---------------+
  | CONCAT(c1,c2) |
  +---------------+
  | ab            |
  +---------------+
  ```

- Functions with one string argument inherit the repertoire of their argument. The result of UPPER(_utf8'*abc*') has ASCII repertoire because its argument has ASCII repertoire.

- For functions that return a string but do not have string arguments and use character_set_connection as the result character set, the result repertoire is ASCII if character_set_connection is ascii, and UNICODE otherwise:

  ```
  FORMAT(numeric_column, 4);
  ```

Use of repertoire changes how MySQL evaluates the following example:

```
SET NAMES ascii;
CREATE TABLE t1 (a INT, b VARCHAR(10) CHARACTER SET latin1);
INSERT INTO t1 VALUES (1,'b');
SELECT CONCAT(FORMAT(a, 4), b) FROM t1;
```

Without repertoire, this error occurs:

```
ERROR 1267 (HY000): Illegal mix of collations (ascii_general_ci,COERCIBLE)
and (latin1_swedish_ci,IMPLICIT) for operation 'concat'
```

With repertoire, a result is returned:

```
+-------------------------+
| CONCAT(FORMAT(a, 4), b) |
+-------------------------+
| 1.0000b                 |
+-------------------------+
```

- Functions with two or more string arguments use the “widest” argument repertoire for the result repertoire (`UNICODE` is wider than `ASCII`). Consider the following `CONCAT()` calls:

  ```
  CONCAT(_ucs2 X'0041', _ucs2 X'0042')
  CONCAT(_ucs2 X'0041', _ucs2 X'00C2')
  ```

  For the first call, the repertoire is `ASCII` because both arguments are within the range of the `ascii` character set. For the second call, the repertoire is `UNICODE` because the second argument is outside the `ascii` character set range.

- The repertoire for function return values is determined based only on the repertoire of the arguments that affect the result's character set and collation.

  ```
  IF(column1 < column2, 'smaller', 'greater')
  ```

  The result repertoire is `ASCII` because the two string arguments (the second argument and the third argument) both have `ASCII` repertoire. The first argument does not matter for the result repertoire, even if the expression uses string values.

10.2.2 UTF-8 for Metadata

Metadata is “the data about the data.” Anything that *describes* the database—as opposed to being the *contents* of the database—is metadata. Thus column names, database names, user names, version names, and most of the string results from `SHOW` are metadata. This is also true of the contents of tables in `INFORMATION_SCHEMA` because those tables by definition contain information about database objects.

Representation of metadata must satisfy these requirements:

- All metadata must be in the same character set. Otherwise, neither the `SHOW` statements nor `SELECT` statements for tables in `INFORMATION_SCHEMA` would work properly because different rows in the same column of the results of these operations would be in different character sets.

- Metadata must include all characters in all languages. Otherwise, users would not be able to name columns and tables using their own languages.

To satisfy both requirements, MySQL stores metadata in a Unicode character set, namely UTF-8. This does not cause any disruption if you never use accented or non-Latin characters. But if you do, you should be aware that metadata is in UTF-8.

The metadata requirements mean that the return values of the `USER()`, `CURRENT_USER()`, `SESSION_USER()`, `SYSTEM_USER()`, `DATABASE()`, and `VERSION()` functions have the UTF-8 character set by default.

The server sets the `character_set_system` system variable to the name of the metadata character set:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_system | utf8  |
+----------------------+-------+
```

Storage of metadata using Unicode does *not* mean that the server returns headers of columns and the results of `DESCRIBE` functions in the `character_set_system` character set by default. When you use `SELECT column1 FROM t`, the name `column1` itself is returned from the server to the client in the character set determined by the value of the `character_set_results` system variable, which has a default value of `utf8mb4`. If you want the server to pass metadata results back in a different character set, use the `SET NAMES` statement to force the server to perform character set conversion. `SET NAMES` sets the `character_set_results` and other related system variables. (See Section 10.4, “Connection Character Sets and Collations”.) Alternatively, a client program can perform the conversion after receiving the result from the server. It is more efficient for the client to perform the conversion, but this option is not always available for all clients.

If `character_set_results` is set to `NULL`, no conversion is performed and the server returns metadata using its original character set (the set indicated by `character_set_system`).

Error messages returned from the server to the client are converted to the client character set automatically, as with metadata.

If you are using (for example) the `USER()` function for comparison or assignment within a single statement, don't worry. MySQL performs some automatic conversion for you.

```
SELECT * FROM t1 WHERE USER() = latin1_column;
```

This works because the contents of `latin1_column` are automatically converted to UTF-8 before the comparison.

```
INSERT INTO t1 (latin1_column) SELECT USER();
```

This works because the contents of `USER()` are automatically converted to `latin1` before the assignment.

Although automatic conversion is not in the SQL standard, the standard does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings. For more information about coercion of strings, see Section 10.8.4, “Collation Coercibility in Expressions”.

10.3 Specifying Character Sets and Collations

There are default settings for character sets and collations at four levels: server, database, table, and column. The description in the following sections may appear complex, but it has been found in practice that multiple-level defaulting leads to natural and obvious results.

`CHARACTER SET` is used in clauses that specify a character set. `CHARSET` can be used as a synonym for `CHARACTER SET`.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the `utf8mb4` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8mb4';
```

For more information about character set-related issues in client/server communication, see Section 10.4, “Connection Character Sets and Collations”.

10.3.1 Collation Naming Conventions

MySQL collation names follow these conventions:

- A collation name starts with the name of the character set with which it is associated, generally followed by one or more suffixes indicating other collation characteristics. For example, `utf8mb4_general_ci` and `latin1_swedish_ci` are collations for the `utf8mb4` and `latin1` character sets, respectively. The `binary` character set has a single collation, also named `binary`, with no suffixes.
- A language-specific collation includes a locale code or language name. For example, `utf8mb4_tr_0900_ai_ci` and `utf8mb4_hu_0900_ai_ci` sort characters for the `utf8mb4` character set using the rules of Turkish and Hungarian, respectively. `utf8mb4_turkish_ci` and `utf8mb4_hungarian_ci` are similar but based on a less recent version of the Unicode Collation Algorithm.
- Collation suffixes indicate whether a collation is case and accent sensitive, or binary. The following table shows the suffixes used to indicate these characteristics.

Table 10.1 Collation Case/Accent Sensitivity Suffixes

Suffix	Meaning
`_ai`	Accent insensitive
`_as`	Accent sensitive
`_ci`	Case insensitive
`_cs`	Case sensitive
`_ks`	Kana sensitive
`_bin`	Binary

For nonbinary collation names that do not specify accent sensitivity, it is determined by case sensitivity. If a collation name does not contain `_ai` or `_as`, `_ci` in the name implies `_ai` and `_cs` in the name implies `_as`. For example, `latin1_general_ci` is explicitly case insensitive and implicitly accent insensitive, `latin1_general_cs` is explicitly case sensitive and implicitly accent sensitive, and `utf8mb4_0900_ai_ci` is explicitly case and accent insensitive.

For Japanese collations, the `_ks` suffix indicates that a collation is kana sensitive; that is, it distinguishes Katakana characters from Hiragana characters. Japanese collations without the `_ks` suffix are not kana sensitive and treat Katakana and Hiragana characters equal for sorting.

For the `binary` collation of the `binary` character set, comparisons are based on numeric byte values. For the `_bin` collation of a nonbinary character set, comparisons are based on numeric character code values, which differ from byte values for multibyte characters. For more information, see Section 10.8.5, "The binary Collation Compared to _bin Collations".

- For Unicode character sets, collation names may include a version number to indicate the version of the Unicode Collation Algorithm (UCA) on which the collation is based. UCA-based collations without a version number in the name use the version-4.0.0 UCA weight keys. For example:
  - `utf8mb4_0900_ai_ci` is based on UCA 9.0.0 weight keys (http://www.unicode.org/Public/UCA/9.0.0/allkeys.txt).
  - `utf8mb4_unicode_520_ci` is based on UCA 5.2.0 weight keys (http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt).
  - `utf8mb4_unicode_ci` (with no version named) is based on UCA 4.0.0 weight keys (http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt).
- For Unicode character sets, the *`xxx`*`_general_mysql500_ci` collations preserve the pre-5.1.24 ordering of the original *`xxx`*`_general_ci` collations and permit upgrades for tables created before MySQL 5.1.24 (Bug #27877).

10.3.2 Server Character Set and Collation

MySQL Server has a server character set and a server collation. These can be set at server startup on the command line or in an option file and changed at runtime.

Initially, the server character set and collation depend on the options that you use when you start `mysqld`. You can use `--character-set-server` for the character set. Along with it, you can add `--collation-server` for the collation. If you don't specify a character set, that is the same as saying `--character-set-server=utf8mb4`. If you specify only a character set (for example, `utf8mb4`) but not a collation, that is the same as saying `--character-set-server=utf8mb4 --collation-server=utf8mb4_0900_ai_ci` because `utf8mb4_0900_ai_ci` is the default collation for `utf8mb4`. Therefore, the following three commands all have the same effect:

```
mysqld
mysqld --character-set-server=utf8mb4
mysqld --character-set-server=utf8mb4 \
  --collation-server=utf8mb4_0900_ai_ci
```

One way to change the settings is by recompiling. To change the default server character set and collation when building from sources, use the `DEFAULT_CHARSET` and `DEFAULT_COLLATION` options for `CMake`. For example:

```
cmake . -DDEFAULT_CHARSET=latin1
```

Or:

```
cmake . -DDEFAULT_CHARSET=latin1 \
```

```
  -DDEFAULT_COLLATION=latin1_german1_ci
```

Both `mysqld` and `CMake` verify that the character set/collation combination is valid. If not, each program displays an error message and terminates.

The server character set and collation are used as default values if the database character set and collation are not specified in `CREATE DATABASE` statements. They have no other purpose.

The current server character set and collation can be determined from the values of the `character_set_server` and `collation_server` system variables. These variables can be changed at runtime.

10.3.3 Database Character Set and Collation

Every database has a database character set and a database collation. The `CREATE DATABASE` and `ALTER DATABASE` statements have optional clauses for specifying the database character set and collation:

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]

ALTER DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

The keyword `SCHEMA` can be used instead of `DATABASE`.

The `CHARACTER SET` and `COLLATE` clauses make it possible to create databases with different character sets and collations on the same MySQL server.

Database options are stored in the data dictionary and can be examined by checking the `INFORMATION_SCHEMA.SCHEMATA` table.

Example:

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL chooses the database character set and database collation in the following manner:

- If both `CHARACTER SET charset_name` and `COLLATE collation_name` are specified, character set `charset_name` and collation `collation_name` are used.
- If `CHARACTER SET charset_name` is specified without `COLLATE`, character set `charset_name` and its default collation are used. To see the default collation for each character set, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table.
- If `COLLATE collation_name` is specified without `CHARACTER SET`, the character set associated with `collation_name` and collation `collation_name` are used.
- Otherwise (neither `CHARACTER SET` nor `COLLATE` is specified), the server character set and server collation are used.

The character set and collation for the default database can be determined from the values of the `character_set_database` and `collation_database` system variables. The server sets these variables whenever the default database changes. If there is no default database, the variables have

the same value as the corresponding server-level system variables, `character_set_server` and `collation_server`.

To see the default character set and collation for a given database, use these statements:

```
USE db_name;
SELECT @@character_set_database, @@collation_database;
```

Alternatively, to display the values without changing the default database:

```
SELECT DEFAULT_CHARACTER_SET_NAME, DEFAULT_COLLATION_NAME
FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = 'db_name';
```

The database character set and collation affect these aspects of server operation:

- For `CREATE TABLE` statements, the database character set and collation are used as default values for table definitions if the table character set and collation are not specified. To override this, provide explicit `CHARACTER SET` and `COLLATE` table options.
- For `LOAD DATA` statements that include no `CHARACTER SET` clause, the server uses the character set indicated by the `character_set_database` system variable to interpret the information in the file. To override this, provide an explicit `CHARACTER SET` clause.
- For stored routines (procedures and functions), the database character set and collation in effect at routine creation time are used as the character set and collation of character data parameters for which the declaration includes no `CHARACTER SET` or a `COLLATE` attribute. To override this, provide `CHARACTER SET` and `COLLATE` explicitly.

10.3.4 Table Character Set and Collation

Every table has a table character set and a table collation. The `CREATE TABLE` and `ALTER TABLE` statements have optional clauses for specifying the table character set and collation:

```
CREATE TABLE tbl_name (column_list)
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]]

ALTER TABLE tbl_name
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]
```

Example:

```
CREATE TABLE t1 ( ... )
CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL chooses the table character set and collation in the following manner:

- If both `CHARACTER SET charset_name` and `COLLATE collation_name` are specified, character set *`charset_name`* and collation *`collation_name`* are used.
- If `CHARACTER SET charset_name` is specified without `COLLATE`, character set *`charset_name`* and its default collation are used. To see the default collation for each character set, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table.
- If `COLLATE collation_name` is specified without `CHARACTER SET`, the character set associated with *`collation_name`* and collation *`collation_name`* are used.

- Otherwise (neither `CHARACTER SET` nor `COLLATE` is specified), the database character set and collation are used.

The table character set and collation are used as default values for column definitions if the column character set and collation are not specified in individual column definitions. The table character set and collation are MySQL extensions; there are no such things in standard SQL.

10.3.5 Column Character Set and Collation

Every “character” column (that is, a column of type `CHAR`, `VARCHAR`, a `TEXT` type, or any synonym) has a column character set and a column collation. Column definition syntax for `CREATE TABLE` and `ALTER TABLE` has optional clauses for specifying the column character set and collation:

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

These clauses can also be used for `ENUM` and `SET` columns:

```
col_name {ENUM | SET} (val_list)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

Examples:

```
CREATE TABLE t1
(
    col1 VARCHAR(5)
      CHARACTER SET latin1
      COLLATE latin1_german1_ci
);

ALTER TABLE t1 MODIFY
    col1 VARCHAR(5)
      CHARACTER SET latin1
      COLLATE latin1_swedish_ci;
```

MySQL chooses the column character set and collation in the following manner:

- If both `CHARACTER SET charset_name` and `COLLATE collation_name` are specified, character set *`charset_name`* and collation *`collation_name`* are used.

  ```
  CREATE TABLE t1
  (
      col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci
  ) CHARACTER SET latin1 COLLATE latin1_bin;
  ```

  The character set and collation are specified for the column, so they are used. The column has character set `utf8` and collation `utf8_unicode_ci`.

- If `CHARACTER SET charset_name` is specified without `COLLATE`, character set *`charset_name`* and its default collation are used.

  ```
  CREATE TABLE t1
  (
      col1 CHAR(10) CHARACTER SET utf8
  ) CHARACTER SET latin1 COLLATE latin1_bin;
  ```

The character set is specified for the column, but the collation is not. The column has character set `utf8` and the default collation for `utf8`, which is `utf8_general_ci`. To see the default collation for each character set, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table.

- If `COLLATE collation_name` is specified without `CHARACTER SET`, the character set associated with `collation_name` and collation `collation_name` are used.

```
CREATE TABLE t1
(
    col1 CHAR(10) COLLATE utf8_polish_ci
) CHARACTER SET latin1 COLLATE latin1_bin;
```

The collation is specified for the column, but the character set is not. The column has collation `utf8_polish_ci` and the character set is the one associated with the collation, which is `utf8`.

- Otherwise (neither `CHARACTER SET` nor `COLLATE` is specified), the table character set and collation are used.

```
CREATE TABLE t1
(
    col1 CHAR(10)
) CHARACTER SET latin1 COLLATE latin1_bin;
```

Neither the character set nor collation is specified for the column, so the table defaults are used. The column has character set `latin1` and collation `latin1_bin`.

The `CHARACTER SET` and `COLLATE` clauses are standard SQL.

If you use `ALTER TABLE` to convert a column from one character set to another, MySQL attempts to map the data values, but if the character sets are incompatible, there may be data loss.

10.3.6 Character String Literal Character Set and Collation

Every character string literal has a character set and a collation.

For the simple statement `SELECT 'string'`, the string has the connection default character set and collation defined by the `character_set_connection` and `collation_connection` system variables.

A character string literal may have an optional character set introducer and `COLLATE` clause, to designate it as a string that uses a particular character set and collation:

```
[_charset_name]'string' [COLLATE collation_name]
```

The `_charset_name` expression is formally called an *introducer*. It tells the parser, “the string that follows uses character set `charset_name`.” An introducer does not change the string to the introducer character set like `CONVERT()` would do. It does not change the string value, although padding may occur. The introducer is just a signal. See Section 10.3.8, “Character Set Introducers”.

Examples:

```
SELECT 'abc';
SELECT _latin1'abc';
SELECT _binary'abc';
```

```
SELECT _utf8mb4'abc' COLLATE utf8mb4_danish_ci;
```

Character set introducers and the `COLLATE` clause are implemented according to standard SQL specifications.

MySQL determines the character set and collation of a character string literal in the following manner:

- If both *`_charset_name`* and `COLLATE` *`collation_name`* are specified, character set *`charset_name`* and collation *`collation_name`* are used. *`collation_name`* must be a permitted collation for *`charset_name`*.
- If *`_charset_name`* is specified but `COLLATE` is not specified, character set *`charset_name`* and its default collation are used. To see the default collation for each character set, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table.
- If *`_charset_name`* is not specified but `COLLATE` *`collation_name`* is specified, the connection default character set given by the `character_set_connection` system variable and collation *`collation_name`* are used. *`collation_name`* must be a permitted collation for the connection default character set.
- Otherwise (neither *`_charset_name`* nor `COLLATE` *`collation_name`* is specified), the connection default character set and collation given by the `character_set_connection` and `collation_connection` system variables are used.

Examples:

- A nonbinary string with `latin1` character set and `latin1_german1_ci` collation:

  ```
  SELECT _latin1'Müller' COLLATE latin1_german1_ci;
  ```

- A nonbinary string with `utf8mb4` character set and its default collation (that is, `utf8mb4_general_ci`):

  ```
  SELECT _utf8mb4'Müller';
  ```

- A binary string with `binary` character set and its default collation (that is, `binary`):

  ```
  SELECT _binary'Müller';
  ```

- A nonbinary string with the connection default character set and `utf8mb4_general_ci` collation (fails if the connection character set is not `utf8mb4`):

  ```
  SELECT 'Müller' COLLATE utf8mb4_general_ci;
  ```

- A string with the connection default character set and collation:

  ```
  SELECT 'Müller';
  ```

An introducer indicates the character set for the following string, but does not change how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by `character_set_connection`.

The following examples show that escape processing occurs using `character_set_connection` even in the presence of an introducer. The examples use `SET NAMES` (which changes `character_set_connection`, as discussed in Section 10.4, “Connection Character Sets and

Collations"), and display the resulting strings using the `HEX()` function so that the exact string contents can be seen.

Example 1:

```
mysql> SET NAMES latin1;
mysql> SELECT HEX('à\n'), HEX(_sjis'à\n');
+------------+-----------------+
| HEX('à\n')  | HEX(_sjis'à\n')  |
+------------+-----------------+
| E00A       | E00A            |
+------------+-----------------+
```

Here, `à` (hexadecimal value `E0`) is followed by `\n`, the escape sequence for newline. The escape sequence is interpreted using the `character_set_connection` value of `latin1` to produce a literal newline (hexadecimal value `0A`). This happens even for the second string. That is, the `_sjis` introducer does not affect the parser's escape processing.

Example 2:

```
mysql> SET NAMES sjis;
mysql> SELECT HEX('à\n'), HEX(_latin1'à\n');
+------------+------------------+
| HEX('à\n')  | HEX(_latin1'à\n')  |
+------------+------------------+
| E05C6E     | E05C6E           |
+------------+------------------+
```

Here, `character_set_connection` is `sjis`, a character set in which the sequence of `à` followed by `\` (hexadecimal values `05` and `5C`) is a valid multibyte character. Hence, the first two bytes of the string are interpreted as a single `sjis` character, and the `\` is not interpreted as an escape character. The following `n` (hexadecimal value `6E`) is not interpreted as part of an escape sequence. This is true even for the second string; the `_latin1` introducer does not affect escape processing.

10.3.7 The National Character Set

Standard SQL defines `NCHAR` or `NATIONAL CHAR` as a way to indicate that a `CHAR` column should use some predefined character set. MySQL uses `utf8` as this predefined character set. For example, these data type declarations are equivalent:

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

As are these:

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NVARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

You can use `N'literal'` (or `n'literal'`) to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
```

```
SELECT n'some text';
SELECT _utf8'some text';
```

10.3.8 Character Set Introducers

A character string literal, hexadecimal literal, or bit-value literal may have an optional character set introducer and `COLLATE` clause, to designate it as a string that uses a particular character set and collation:

```
[_charset_name] literal [COLLATE collation_name]
```

The *`_charset_name`* expression is formally called an *introducer*. It tells the parser, “the string that follows uses character set *`charset_name`*.” An introducer does not change the string to the introducer character set like `CONVERT()` would do. It does not change the string value, although padding may occur. The introducer is just a signal.

For character string literals, space between the introducer and the string is permitted but optional.

Examples:

```
SELECT 'abc';
SELECT _latin1'abc';
SELECT _binary'abc';
SELECT _utf8mb4'abc' COLLATE utf8mb4_danish_ci;

SELECT _latin1 X'4D7953514C';
SELECT _utf8mb4 0x4D7953514C COLLATE utf8mb4_danish_ci;

SELECT _latin1 b'1000001';
SELECT _utf8mb4 0b1000001 COLLATE utf8mb4_danish_ci;
```

Character set introducers and the `COLLATE` clause are implemented according to standard SQL specifications.

Character string literals can be designated as binary strings by using the `_binary` introducer. Hexadecimal literals and bit-value literals are binary strings by default, so `_binary` is permitted, but normally unnecessary. `_binary` may be useful to preserve a hexadecimal or bit literal as a binary string in contexts for which the literal is otherwise treated as a number. For example, bit operations permit numeric or binary string arguments in MySQL 8.0 and higher, but treat hexadecimal and bit literals as numbers by default. To explicitly specify binary string context for such literals, use a `_binary` introducer for at least one of the arguments:

```
mysql> SET @v1 = X'000D' | X'0BC0';
mysql> SET @v2 = _binary X'000D' | X'0BC0';
mysql> SELECT HEX(@v1), HEX(@v2);
+----------+----------+
| HEX(@v1) | HEX(@v2) |
+----------+----------+
| BCD      | 0BCD     |
+----------+----------+
```

The displayed result appears similar for both bit operations, but the result without `_binary` is a `BIGINT` value, whereas the result with `_binary` is a binary string. Due to the difference in result types, the displayed values differ: High-order 0 digits are not displayed for the numeric result.

MySQL determines the character set and collation of a character string literal, hexadecimal literal, or bit-value literal in the following manner:

- If both *_charset_name* and COLLATE *collation_name* are specified, character set *charset_name* and collation *collation_name* are used. *collation_name* must be a permitted collation for *charset_name*.
- If *_charset_name* is specified but COLLATE is not specified, character set *charset_name* and its default collation are used. To see the default collation for each character set, use the SHOW CHARACTER SET statement or query the INFORMATION_SCHEMA CHARACTER_SETS table.
- If *_charset_name* is not specified but COLLATE *collation_name* is specified:
  - For a character string literal, the connection default character set given by the character_set_connection system variable and collation *collation_name* are used. *collation_name* must be a permitted collation for the connection default character set.
  - For a hexadecimal literal or bit-value literal, the only permitted collation is binary because these types of literals are binary strings by default.
- Otherwise (neither *_charset_name* nor COLLATE *collation_name* is specified):
  - For a character string literal, the connection default character set and collation given by the character_set_connection and collation_connection system variables are used.
  - For a hexadecimal literal or bit-value literal, the character set and collation are binary.

Examples:

- Nonbinary strings with latin1 character set and latin1_german1_ci collation:

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
SELECT _latin1 X'0A0D' COLLATE latin1_german1_ci;
SELECT _latin1 b'0110' COLLATE latin1_german1_ci;
```

- Nonbinary strings with utf8mb4 character set and its default collation (that is, utf8mb4_0900_ai_ci):

```
SELECT _utf8mb4'Müller';
SELECT _utf8mb4 X'0A0D';
SELECT _utf8mb4 b'0110';
```

- Binary strings with binary character set and its default collation (that is, binary):

```
SELECT _binary'Müller';
SELECT X'0A0D';
SELECT b'0110';
```

  The hexadecimal literal and bit-value literal need no introducer because they are binary strings by default.

- A nonbinary string with the connection default character set and utf8mb4_general_ci collation (fails if the connection character set is not utf8mb4):

```
SELECT 'Müller' COLLATE utf8mb4_general_ci;
```

  This construction (COLLATE only) does not work for hexadecimal literals or bit literals because their character set is binary no matter the connection character set, and binary is not compatible with the utf8mb4_general_ci collation. The only permitted COLLATE clause in the absence of an introducer is COLLATE binary.

- A string with the connection default character set and collation:

```
SELECT 'Müller';
```

For character set literals, an introducer indicates the character set for the following string, but does not change how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by `character_set_connection`. For additional discussion and examples, see Section 10.3.6, “Character String Literal Character Set and Collation”.

10.3.9 Examples of Character Set and Collation Assignment

The following examples show how MySQL determines default character set and collation values.

Example 1: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Here we have a column with a `latin1` character set and a `latin1_german1_ci` collation. The definition is explicit, so that is straightforward. Notice that there is no problem with storing a `latin1` column in a `latin2` table.

Example 2: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

This time we have a column with a `latin1` character set and a default collation. Although it might seem natural, the default collation is not taken from the table level. Instead, because the default collation for `latin1` is always `latin1_swedish_ci`, column `c1` has a collation of `latin1_swedish_ci` (not `latin1_danish_ci`).

Example 3: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

We have a column with a default character set and a default collation. In this circumstance, MySQL checks the table level to determine the column character set and collation. Consequently, the character set for column `c1` is `latin1` and its collation is `latin1_danish_ci`.

Example 4: Database, Table, and Column Definition

```
CREATE DATABASE d1
    DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
```

```
    c1 CHAR(10)
);
```

We create a column without specifying its character set and collation. We're also not specifying a character set and a collation at the table level. In this circumstance, MySQL checks the database level to determine the table settings, which thereafter become the column settings.) Consequently, the character set for column `c1` is `latin2` and its collation is `latin2_czech_ci`.

10.3.10 Compatibility with Other DBMSs

For MaxDB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

10.4 Connection Character Sets and Collations

A “connection” is what a client program makes when it connects to the server, to begin a session within which it interacts with the server. The client sends SQL statements, such as queries, over the session connection. The server sends responses, such as result sets or error messages, over the connection back to the client.

- Connection Character Set and Collation System Variables
- Impermissible Client Character Sets
- Client Program Connection Character Set Configuration
- SQL Statements for Connection Character Set Configuration
- Connection Character Set Error Handling

Connection Character Set and Collation System Variables

Several character set and collation system variables relate to a client's interaction with the server. Some of these have been mentioned in earlier sections:

- The `character_set_server` and `collation_server` system variables indicate the server character set and collation. See Section 10.3.2, “Server Character Set and Collation”.
- The `character_set_database` and `collation_database` system variables indicate the character set and collation of the default database. See Section 10.3.3, “Database Character Set and Collation”.

Additional character set and collation system variables are involved in handling traffic for the connection between a client and the server. Every client has session-specific connection-related character set and collation system variables. These session system variable values are initialized at connect time, but can be changed within the session.

Several questions about character set and collation handling for client connections can be answered in terms of system variables:

- What character set are statements in when they leave the client?

  The server takes the `character_set_client` system variable to be the character set in which statements are sent by the client.

Note

Some character sets cannot be used as the client character set. See Impermissible Client Character Sets.

- What character set should the server translate statements to after receiving them?

  To determine this, the server uses the `character_set_connection` and `collation_connection` system variables:

  - The server converts statements sent by the client from `character_set_client` to `character_set_connection`. Exception: For string literals that have an introducer such as `_utf8mb4` or `_latin2`, the introducer determines the character set. See Section 10.3.8, “Character Set Introducers”.

  - `collation_connection` is important for comparisons of literal strings. For comparisons of strings with column values, `collation_connection` does not matter because columns have their own collation, which has a higher collation precedence (see Section 10.8.4, “Collation Coercibility in Expressions”).

- What character set should the server translate query results to before shipping them back to the client?

  The `character_set_results` system variable indicates the character set in which the server returns query results to the client. This includes result data such as column values, result metadata such as column names, and error messages.

  To tell the server to perform no conversion of result sets or error messages, set `character_set_results` to `NULL` or `binary`:

```
SET character_set_results = NULL;
SET character_set_results = binary;
```

  For more information about character sets and error messages, see Section 10.6, “Error Message Character Set”.

To see the values of the character set and collation system variables that apply to the current session, use this statement:

```
SELECT * FROM performance_schema.session_variables
WHERE VARIABLE_NAME IN (
'character_set_client', 'character_set_connection',
'character_set_results', 'collation_connection'
) ORDER BY VARIABLE_NAME;
```

The following simpler statements also display the connection variables, but include other related variables as well. They can be useful to see *all* character set and collation system variables:

```
SHOW SESSION VARIABLES LIKE 'character\_set\_%';
SHOW SESSION VARIABLES LIKE 'collation\_%';
```

Clients can fine-tune the settings for these variables, or depend on the defaults (in which case, you can skip the rest of this section). If you do not use the defaults, you must change the character settings *for each connection to the server.*

Impermissible Client Character Sets

The `character_set_client` system variable cannot be set to certain character sets:

```
ucs2
utf16
utf16le
utf32
```

Attempting to use any of those character sets as the client character set produces an error:

```
mysql> SET character_set_client = 'ucs2';
ERROR 1231 (42000): Variable 'character_set_client'
can't be set to the value of 'ucs2'
```

The same error occurs if any of those character sets are used in the following contexts, all of which result in an attempt to set `character_set_client` to the named character set:

- The `--default-character-set=charset_name` command option used by MySQL client programs such as `mysql` and `mysqladmin`.
- The `SET NAMES 'charset_name'` statement.
- The `SET CHARACTER SET 'charset_name'` statement.

Client Program Connection Character Set Configuration

When a client connects to the server, it indicates which character set it wants to use for communication with the server. (Actually, the client indicates the default collation for that character set, from which the server can determine the character set.) The server uses this information to set the `character_set_client`, `character_set_results`, `character_set_connection` system variables to the character set, and `collation_connection` to the character set default collation. In effect, the server performs the equivalent of a `SET NAMES` operation.

If the server does not support the requested character set or collation, it falls back to using the server character set and collation to configure the connection. For additional detail about this fallback behavior, see Connection Character Set Error Handling.

The `mysql`, `mysqladmin`, `mysqlcheck`, `mysqlimport`, and `mysqlshow` client programs determine the default character set to use as follows:

- In the absence of other information, each client uses the compiled-in default character set, usually `utf8mb4`.
- Each client can autodetect which character set to use based on the operating system setting, such as the value of the `LANG` or `LC_ALL` locale environment variable on Unix systems or the code page setting on Windows systems. For systems on which the locale is available from the OS, the client uses it to set the default character set rather than using the compiled-in default. For example, setting `LANG` to `ru_RU.KOI8-R` causes the `koi8r` character set to be used. Thus, users can configure the locale in their environment for use by MySQL clients.

  The OS character set is mapped to the closest MySQL character set if there is no exact match. If the client does not support the matching character set, it uses the compiled-in default. For example, `utf8` and `utf-8` map to `utf8mb4`, and `ucs2` is not supported as a connection character set, so it maps to the compiled-in default.

  C applications can use character set autodetection based on the OS setting by invoking `mysql_options()` as follows before connecting to the server:

```
mysql_options(mysql,
              MYSQL_SET_CHARSET_NAME,
              MYSQL_AUTODETECT_CHARSET_NAME);
```

- Each client supports a `--default-character-set` option, which enables users to specify the character set explicitly to override whatever default the client otherwise determines.

Note

Some character sets cannot be used as the client character set. Attempting to use them with `--default-character-set` produces an error. See Impermissible Client Character Sets.

With the `mysql` client, to use a character set different from the default, you could explicitly execute a `SET NAMES` statement every time you connect to the server (see Client Program Connection Character Set Configuration). To accomplish the same result more easily, specify the character set in your option file. For example, the following option file setting changes the three connection-related character set system variables set to `koi8r` each time you invoke `mysql`:

```
[mysql]
default-character-set=koi8r
```

If you are using the `mysql` client with auto-reconnect enabled (which is not recommended), it is preferable to use the `charset` command rather than `SET NAMES`. For example:

```
mysql> charset koi8r
Charset changed
```

The `charset` command issues a `SET NAMES` statement, and also changes the default character set that `mysql` uses when it reconnects after the connection has dropped.

When configuration client programs, you must also consider the environment within which they execute. See Section 10.5, “Configuring Application Character Set and Collation”.

SQL Statements for Connection Character Set Configuration

After a connection has been established, clients can change the character set and collation system variables for the current session. These variables can be changed individually using `SET` statements, but two more convenient statements affect the connection-related character set sytem variables as a group:

- `SET NAMES 'charset_name' [COLLATE 'collation_name']`

  `SET NAMES` indicates what character set the client will use to send SQL statements to the server. Thus, `SET NAMES 'cp1251'` tells the server, “future incoming messages from this client are in character set `cp1251`.” It also specifies the character set that the server should use for sending results back to the client. (For example, it indicates what character set to use for column values if you use a `SELECT` statement that produces a result set.)

  A `SET NAMES 'charset_name'` statement is equivalent to these three statements:

```
SET character_set_client = charset_name;
SET character_set_results = charset_name;
SET character_set_connection = charset_name;
```

Setting `character_set_connection` to *`charset_name`* also implicitly sets `collation_connection` to the default collation for *`charset_name`*. It is unnecessary to set that collation explicitly. To specify a particular collation to use for `collation_connection`, add a `COLLATE` clause:

```
SET NAMES 'charset_name' COLLATE 'collation_name'
```

- `SET CHARACTER SET 'charset_name'`

  `SET CHARACTER SET` is similar to `SET NAMES` but sets `character_set_connection` and `collation_connection` to `character_set_database` and `collation_database` (which, as mentioned previously, indicate the character set and collation of the default database).

  A `SET CHARACTER SET charset_name` statement is equivalent to these three statements:

  ```
  SET character_set_client = charset_name;
  SET character_set_results = charset_name;
  SET collation_connection = @@collation_database;
  ```

  Setting `collation_connection` also implicitly sets `character_set_connection` to the character set associated with the collation (equivalent to executing `SET character_set_connection = @@character_set_database`). It is unnecessary to set `character_set_connection` explicitly.

Note

Some character sets cannot be used as the client character set. Attempting to use them with `SET NAMES` or `SET CHARACTER SET` produces an error. See Impermissible Client Character Sets.

Example: Suppose that `column1` is defined as `CHAR(5) CHARACTER SET latin2`. If you do not say `SET NAMES` or `SET CHARACTER SET`, then for `SELECT column1 FROM t`, the server sends back all the values for `column1` using the character set that the client specified when it connected. On the other hand, if you say `SET NAMES 'latin1'` or `SET CHARACTER SET 'latin1'` before issuing the `SELECT` statement, the server converts the `latin2` values to `latin1` just before sending results back. Conversion may be lossy for characters that are not in both character sets.

Connection Character Set Error Handling

Attempts to use an inappropriate connection character set or collation can produce an error, or cause the server to fall back to its default character set and collation for a given connection. This section describes problems that can occur when configuring the connection character set. These problems can occur when establishing a connection or when changing the character set within an established connection.

- Connect-Time Error Handling
- Runtime Error Handling

Connect-Time Error Handling

Some character sets cannot be used as the client character set; see Impermissible Client Character Sets. If you specify a character set that is valid but not permitted as a client character set, the server returns an error:

```
shell> mysql --default-character-set=ucs2
```

```
ERROR 1231 (42000): Variable 'character_set_client' can't be set to
the value of 'ucs2'
```

If you specify a character set that the client does not recognize, it produces an error:

```
shell> mysql --default-character-set=bogus
mysql: Character set 'bogus' is not a compiled character set and is
not specified in the '/usr/local/mysql/share/charsets/Index.xml' file
ERROR 2019 (HY000): Can't initialize character set bogus
(path: /usr/local/mysql/share/charsets/)
```

If you specify a character set that the client recognizes but the server does not, the server falls back to its default character set and collation. Suppose that the server is configured to use `latin1` and `latin1_swedish_ci` as its defaults, and that it does not recognize `gb18030` as a valid character set. A client that specifies `--default-character-set=gb18030` is able to connect to the server, but the resulting character set is not what the client wants:

```
mysql> SHOW SESSION VARIABLES LIKE 'character\_set\_%';
+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| character_set_client     | latin1 |
| character_set_connection | latin1 |
...
| character_set_results    | latin1 |
...
+--------------------------+--------+
mysql> SHOW SESSION VARIABLES LIKE 'collation_connection';
+----------------------+-------------------+
| Variable_name        | Value             |
+----------------------+-------------------+
| collation_connection | latin1_swedish_ci |
+----------------------+-------------------+
```

You can see that the connection system variables have been set to reflect a character set and collation of `latin1` and `latin1_swedish_ci`. This occurs because the server cannot satisfy the client character set request and falls back to its defaults.

In this case, the client cannot use the character set that it wants because the server does not support it. The client must either be willing to use a different character set, or connect to a different server that supports the desired character set.

The same problem occurs in a more subtle context: When the client tells the server to use a character set that the server recognizes, but the default collation for that character set on the client side is not known on the server side. This occurs, for example, when a MySQL 8.0 client wants to connect to a MySQL 5.7 server using `utf8mb4` as the client character set. A client that specifies `--default-character-set=utf8mb4` is able to connect to the server. However, as in the previous example, the server falls back to its default character set and collation, not what the client requested:

```
mysql> SHOW SESSION VARIABLES LIKE 'character\_set\_%';
+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| character_set_client     | latin1 |
| character_set_connection | latin1 |
...
| character_set_results    | latin1 |
...
+--------------------------+--------+
mysql> SHOW SESSION VARIABLES LIKE 'collation_connection';
```

```
+----------------------+-------------------+
| Variable_name        | Value             |
+----------------------+-------------------+
| collation_connection | latin1_swedish_ci |
+----------------------+-------------------+
```

Why does this occur? After all, `utf8mb4` is known to the 8.0 client and the 5.7 server, so both of them recognize it. To understand this behavior, it is necessary to understand that when the client tells the server which character set it wants to use, it really tells the server the default collation for that character set. Therefore, the aforementioned behavior occurs due to a combination of factors:

- The default collation for `utf8mb4` differs between MySQL 5.7 and 8.0 (`utf8mb4_general_ci` for 5.7, `utf8mb4_0900_ai_ci` for 8.0).
- When the 8.0 client requests a character set of `utf8mb4`, what it sends to the server is the default 8.0 `utf8mb4` collation; that is, the `utf8mb4_0900_ai_ci`.
- `utf8mb4_0900_ai_ci` is implemented only as of MySQL 8.0, so the 5.7 server does not recognize it.
- Because the 5.7 server does not recognize `utf8mb4_0900_ai_ci`, it cannot satisfy the client character set request, and falls back to its default character set and collation (`latin1` and `latin1_swedish_ci`).

In this case, the client can still use `utf8mb4` by issuing a `SET NAMES 'utf8mb4'` statement after connecting. The resulting collation is the 5.7 default `utf8mb4` collation; that is, `utf8mb4_general_ci`. If the client additionally wants a collation of `utf8mb4_0900_ai_ci`, it cannot achieve that because the server does not recognize that collation. The client must either be willing to use a different `utf8mb4` collation, or connect to a server from MySQL 8.0 or higher.

Runtime Error Handling

Within an established connection, the client can request a change of connection character set and collation with `SET NAMES` or `SET CHARACTER SET`.

Some character sets cannot be used as the client character set; see Impermissible Client Character Sets. If you specify a character set that is valid but not permitted as a client character set, the server returns an error:

```
mysql> SET NAMES 'ucs2';
ERROR 1231 (42000): Variable 'character_set_client' can't be set to
the value of 'ucs2'
```

If the server does not recognize the character set (or the collation), it produces an error:

```
mysql> SET NAMES 'bogus';
ERROR 1115 (42000): Unknown character set: 'bogus'

mysql> SET NAMES 'utf8mb4' COLLATE 'bogus';
ERROR 1273 (HY000): Unknown collation: 'bogus'
```

Tip

A client that wants to verify whether its requested character set was honored by the server can execute the following statement after connecting and checking that the result is the expected character set:

```
SELECT @@character_set_client;
```

10.5 Configuring Application Character Set and Collation

For applications that store data using the default MySQL character set and collation (`utf8mb4`, `utf8mb4_0900_ai_ci`), no special configuration should be needed. If applications require data storage using a different character set or collation, you can configure character set information several ways:

- Specify character settings per database. For example, applications that use one database might use the default of `utf8mb4`, whereas applications that use another database might use `sjis`.
- Specify character settings at server startup. This causes the server to use the given settings for all applications that do not make other arrangements.
- Specify character settings at configuration time, if you build MySQL from source. This causes the server to use the given settings as the defaults for all applications, without having to specify them at server startup.

When different applications require different character settings, the per-database technique provides a good deal of flexibility. If most or all applications use the same character set, specifying character settings at server startup or configuration time may be most convenient.

For the per-database or server-startup techniques, the settings control the character set for data storage. Applications must also tell the server which character set to use for client/server communications, as described in the following instructions.

The examples shown here assume use of the `latin1` character set and `latin1_swedish_ci` collation in particular contexts as an alternative to the defaults of `utf8mb4` and `utf8mb4_0900_ai_ci`.

- **Specify character settings per database.** To create a database such that its tables will use a given default character set and collation for data storage, use a `CREATE DATABASE` statement like this:

  ```
  CREATE DATABASE mydb
    CHARACTER SET latin1
    COLLATE latin1_swedish_ci;
  ```

  Tables created in the database will use `latin1` and `latin1_swedish_ci` by default for any character columns.

  Applications that use the database should also configure their connection to the server each time they connect. This can be done by executing a `SET NAMES 'latin1'` statement after connecting. The statement can be used regardless of connection method (the `mysql` client, PHP scripts, and so forth).

  In some cases, it may be possible to configure the connection to use the desired character set some other way. For example, to connect using `mysql`, you can specify the `--default-character-set=latin1` command-line option to achieve the same effect as `SET NAMES 'latin1'`.

  For more information about configuring client connections, see Section 10.4, “Connection Character Sets and Collations”.

  Note

  If you use `ALTER DATABASE` to change the database default character set or collation, existing stored routines in the database that use those defaults must be dropped and recreated so that they use the new defaults. (In a stored routine, variables with character data types use the database defaults if the character set or collation are not specified explicitly. See Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.)

- **Specify character settings at server startup.** To select a character set and collation at server startup, use the `--character-set-server` and `--collation-server` options. For example, to specify the options in an option file, include these lines:

```
[mysqld]
character-set-server=latin1
collation-server=latin1_swedish_ci
```

  These settings apply server-wide and apply as the defaults for databases created by any application, and for tables created in those databases.

  It is still necessary for applications to configure their connection using `SET NAMES` or equivalent after they connect, as described previously. You might be tempted to start the server with the `--init_connect="SET NAMES 'latin1'"` option to cause `SET NAMES` to be executed automatically for each client that connects. However, this may yield inconsistent results because the `init_connect` value is not executed for users who have the `CONNECTION_ADMIN` or `SUPER` privilege.

- **Specify character settings at MySQL configuration time.** To select a character set and collation if you configure and build MySQL from source, use the `DEFAULT_CHARSET` and `DEFAULT_COLLATION` `CMake` options:

```
cmake . -DDEFAULT_CHARSET=latin1 \
  -DDEFAULT_COLLATION=latin1_swedish_ci
```

  The resulting server uses `latin1` and `latin1_swedish_ci` as the default for databases and tables and for client connections. It is unnecessary to use `--character-set-server` and `--collation-server` to specify those defaults at server startup. It is also unnecessary for applications to configure their connection using `SET NAMES` or equivalent after they connect to the server.

Regardless of how you configure the MySQL character set for application use, you must also consider the environment within which those applications execute. For example, if you will send statements using UTF-8 text taken from a file that you create in an editor, you should edit the file with the locale of your environment set to UTF-8 so that the file encoding is correct and so that the operating system handles it correctly. If you use the `mysql` client from within a terminal window, the window must be configured to use UTF-8 or characters may not display properly. For a script that executes in a Web environment, the script must handle character encoding properly for its interaction with the MySQL server, and it must generate pages that correctly indicate the encoding so that browsers know how to display the content of the pages. For example, you can include this `<meta>` tag within your `<head>` element:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

10.6 Error Message Character Set

This section describes how the MySQL server uses character sets for constructing error messages. For information about the language of error messages (rather than the character set), see Section 10.12, “Setting the Error Message Language”. For general information about configuring error logging, see Section 5.4.2, “The Error Log”.

- Character Set for Error Message Construction
- Character Set for Error Message Disposition

Character Set for Error Message Construction

The server constructs error messages as follows:

- The message template uses UTF-8 (`utf8mb3`).
- Parameters in the message template are replaced with values that apply to a specific error occurrence:
  - Identifiers such as table or column names use UTF-8 internally so they are copied as is.
  - Character (nonbinary) string values are converted from their character set to UTF-8.
  - Binary string values are copied as is for bytes in the range `0x20` to `0x7E`, and using `\x` hexadecimal encoding for bytes outside that range. For example, if a duplicate-key error occurs for an attempt to insert `0x41CF9F` into a `VARBINARY` unique column, the resulting error message uses UTF-8 with some bytes hexadecimal encoded:

    ```
    Duplicate entry 'A\xC3\x9F' for key 1
    ```

Character Set for Error Message Disposition

An error message, once constructed, can be written by the server to the error log or sent to clients:

- If the server writes the error message to the error log, it writes it in UTF-8, as constructed, without conversion to another character set.
- If the server sends the error message to a client program, the server converts it from UTF-8 to the character set specified by the `character_set_results` system variable. If `character_set_results` has a value of `NULL` or `binary`, no conversion occurs. No conversion occurs if the variable value is `utf8mb3` or `utf8mb4`, either, because those character sets have a repertoire that includes all UTF-8 characters used in message construction.

  If characters cannot be represented in `character_set_results`, some encoding may occur during the conversion. The encoding uses Unicode code point values:

  - Characters in the Basic Multilingual Plane (BMP) range (`0x0000` to `0xFFFF`) are written using `\`*nnnn* notation.
  - Characters outside the BMP range (`0x10000` to `0x10FFFF`) are written using `\+`*nnnnnn* notation.

  Clients can set `character_set_results` to control the character set in which they receive error messages. The variable can be set directly, or indirectly by means such as `SET NAMES`. For more information about `character_set_results`, see Section 10.4, “Connection Character Sets and Collations”.

10.7 Column Character Set Conversion

To convert a binary or nonbinary string column to use a particular character set, use `ALTER TABLE`. For successful conversion to occur, one of the following conditions must apply:

- If the column has a binary data type (`BINARY`, `VARBINARY`, `BLOB`), all the values that it contains must be encoded using a single character set (the character set you're converting the column to). If you use a binary column to store information in multiple character sets, MySQL has no way to know which values use which character set and cannot convert the data properly.
- If the column has a nonbinary data type (`CHAR`, `VARCHAR`, `TEXT`), its contents should be encoded in the column character set, not some other character set. If the contents are encoded in a different character set, you can convert the column to use a binary data type first, and then to a nonbinary column with the desired character set.

Suppose that a table `t` has a binary column named `col1` defined as `VARBINARY(50)`. Assuming that the information in the column is encoded using a single character set, you can convert it to a nonbinary column that has that character set. For example, if `col1` contains binary data representing characters in the `greek` character set, you can convert it as follows:

```
ALTER TABLE t MODIFY col1 VARCHAR(50) CHARACTER SET greek;
```

If your original column has a type of `BINARY(50)`, you could convert it to `CHAR(50)`, but the resulting values will be padded with `0x00` bytes at the end, which may be undesirable. To remove these bytes, use the `TRIM()` function:

```
UPDATE t SET col1 = TRIM(TRAILING 0x00 FROM col1);
```

Suppose that table `t` has a nonbinary column named `col1` defined as `CHAR(50) CHARACTER SET latin1` but you want to convert it to use `utf8` so that you can store values from many languages. The following statement accomplishes this:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET utf8;
```

Conversion may be lossy if the column contains characters that are not in both character sets.

A special case occurs if you have old tables from before MySQL 4.1 where a nonbinary column contains values that actually are encoded in a character set different from the server's default character set. For example, an application might have stored `sjis` values in a column, even though MySQL's default character set was different. It is possible to convert the column to use the proper character set but an additional step is required. Suppose that the server's default character set was `latin1` and `col1` is defined as `CHAR(50)` but its contents are `sjis` values. The first step is to convert the column to a binary data type, which removes the existing character set information without performing any character conversion:

```
ALTER TABLE t MODIFY col1 BLOB;
```

The next step is to convert the column to a nonbinary data type with the proper character set:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET sjis;
```

This procedure requires that the table not have been modified already with statements such as `INSERT` or `UPDATE` after an upgrade to MySQL 4.1 or higher. In that case, MySQL would store new values in the column using `latin1`, and the column will contain a mix of `sjis` and `latin1` values and cannot be converted properly.

If you specified attributes when creating a column initially, you should also specify them when altering the table with `ALTER TABLE`. For example, if you specified `NOT NULL` and an explicit `DEFAULT` value, you should also provide them in the `ALTER TABLE` statement. Otherwise, the resulting column definition will not include those attributes.

To convert all character columns in a table, the `ALTER TABLE ... CONVERT TO CHARACTER SET` *`charset`* statement may be useful. See Section 13.1.9, “ALTER TABLE Syntax”.

10.8 Collation Issues

The following sections discuss various aspects of character set collations.

10.8.1 Using COLLATE in SQL Statements

With the `COLLATE` clause, you can override whatever the default collation is for a comparison. `COLLATE` may be used in various parts of SQL statements. Here are some examples:

- With `ORDER BY`:

```
SELECT k
FROM t1
ORDER BY k COLLATE latin1_german2_ci;
```

- With `AS`:

```
SELECT k COLLATE latin1_german2_ci AS k1
FROM t1
ORDER BY k1;
```

- With `GROUP BY`:

```
SELECT k
FROM t1
GROUP BY k COLLATE latin1_german2_ci;
```

- With aggregate functions:

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- With `DISTINCT`:

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- With `WHERE`:

```
    SELECT *
    FROM t1
    WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

```
    SELECT *
    FROM t1
    WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
```

- With `HAVING`:

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

10.8.2 COLLATE Clause Precedence

The `COLLATE` clause has high precedence (higher than `||`), so the following two expressions are equivalent:

```
x || y COLLATE z
x || (y COLLATE z)
```

10.8.3 Character Set and Collation Compatibility

Each character set has one or more collations, but each collation is associated with one and only one character set. Therefore, the following statement causes an error message because the `latin2_bin` collation is not legal with the `latin1` character set:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

10.8.4 Collation Coercibility in Expressions

In the great majority of statements, it is obvious what collation MySQL uses to resolve a comparison operation. For example, in the following cases, it should be clear that the collation is the collation of column `x`:

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

However, with multiple operands, there can be ambiguity. For example:

```
SELECT x FROM T WHERE x = 'Y';
```

Should the comparison use the collation of the column `x`, or of the string literal `'Y'`? Both `x` and `'Y'` have collations, so which collation takes precedence?

A mix of collations may also occur in contexts other than comparison. For example, a multiple-argument concatenation operation such as `CONCAT(x,'Y')` combines its arguments to produce a single string. What collation should the result have?

To resolve questions like these, MySQL checks whether the collation of one item can be coerced to the collation of the other. MySQL assigns coercibility values as follows:

- An explicit `COLLATE` clause has a coercibility of 0 (not coercible at all).
- The concatenation of two strings with different collations has a coercibility of 1.
- The collation of a column or a stored routine parameter or local variable has a coercibility of 2.
- A “system constant” (the string returned by functions such as `USER()` or `VERSION()`) has a coercibility of 3.
- The collation of a literal has a coercibility of 4.
- The collation of a numeric or temporal value has a coercibility of 5.
- `NULL` or an expression that is derived from `NULL` has a coercibility of 6.

MySQL uses coercibility values with the following rules to resolve ambiguities:

- Use the collation with the lowest coercibility value.
- If both sides have the same coercibility, then:

- If both sides are Unicode, or both sides are not Unicode, it is an error.
- If one of the sides has a Unicode character set, and another side has a non-Unicode character set, the side with Unicode character set wins, and automatic character set conversion is applied to the non-Unicode side. For example, the following statement does not return an error:

  ```
  SELECT CONCAT(utf8_column, latin1_column) FROM t1;
  ```

  It returns a result that has a character set of `utf8` and the same collation as `utf8_column`. Values of `latin1_column` are automatically converted to `utf8` before concatenating.
- For an operation with operands from the same character set but that mix a `_bin` collation and a `_ci` or `_cs` collation, the `_bin` collation is used. This is similar to how operations that mix nonbinary and binary strings evaluate the operands as binary strings, except that it is for collations rather than data types.

Although automatic conversion is not in the SQL standard, the standard does say that every character set is (in terms of supported characters) a "subset" of Unicode. Because it is a well-known principle that "what applies to a superset can apply to a subset," we believe that a collation for Unicode can apply for comparisons with non-Unicode strings.

The following table illustrates some applications of the preceding rules.

Comparison	Collation Used
`column1 = 'A'`	Use collation of `column1`
`column1 = 'A' COLLATE x`	Use collation of `'A' COLLATE x`
`column1 COLLATE x = 'A' COLLATE y`	Error

To determine the coercibility of a string expression, use the `COERCIBILITY()` function (see Section 12.15, "Information Functions"):

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(VERSION());
        -> 3
mysql> SELECT COERCIBILITY('A');
        -> 4
mysql> SELECT COERCIBILITY(1000);
        -> 5
```

For implicit conversion of a numeric or temporal value to a string, such as occurs for the argument `1` in the expression `CONCAT(1, 'abc')`, the result is a character (nonbinary) string that has a character set and collation determined by the `character_set_connection` and `collation_connection` system variables. See Section 12.2, "Type Conversion in Expression Evaluation".

10.8.5 The binary Collation Compared to _bin Collations

This section describes how the `binary` collation for binary strings compares to `_bin` collations for nonbinary strings.

Binary strings (as stored using the `BINARY`, `VARBINARY`, and `BLOB` data types) have a character set and collation named `binary`. Binary strings are sequences of bytes and the numeric values of those bytes determine comparison and sort order.

Nonbinary strings (as stored using the `CHAR`, `VARCHAR`, and `TEXT` data types) have a character set and collation other than `binary`. A given nonbinary character set can have several collations, each of which defines a particular comparison and sort order for the characters in the set. One of these is the binary collation for the character set, indicated by a `_bin` suffix in the collation name. For example, the binary collations for `utf8` and `latin1` are named `utf8_bin` and `latin1_bin`, respectively. (`utf8mb4` is an exception that has two binary collations; see Section 10.10.1, “Unicode Character Sets”.)

The `binary` collation differs from `_bin` collations in several respects, discussed in the following sections:

- The Unit for Comparison and Sorting
- Character Set Conversion
- Lettercase Conversion
- Trailing Space Handling in Comparisons
- Trailing Space Handling for Inserts and Retrievals

The Unit for Comparison and Sorting

Binary strings are sequences of bytes. For the `binary` collation, comparison and sorting are based on numeric byte values. Nonbinary strings are sequences of characters, which might be multibyte. Collations for nonbinary strings define an ordering of the character values for comparison and sorting. For `_bin` collations, this ordering is based on numeric character code values, which is similar to ordering for binary strings except that character code values might be multibyte.

Character Set Conversion

A nonbinary string has a character set and is automatically converted to another character set in many cases, even when the string has a `_bin` collation:

- When assigning column values to another column that has a different character set:

```
UPDATE t1 SET utf8_bin_column=latin1_column;
INSERT INTO t1 (latin1_column) SELECT utf8_bin_column FROM t2;
```

- When assigning column values for `INSERT` or `UPDATE` using a string literal:

```
SET NAMES latin1;
INSERT INTO t1 (utf8_bin_column) VALUES ('string-in-latin1');
```

- When sending results from the server to a client:

```
SET NAMES latin1;
SELECT utf8_bin_column FROM t2;
```

For binary string columns, no conversion occurs. For cases similar to those preceding, the string value is copied byte-wise.

Lettercase Conversion

Collations for nonbinary character sets provide information about lettercase of characters, so characters in a nonbinary string can be converted from one lettercase to another, even for `_bin` collations that ignore lettercase for ordering:

```
mysql> SET NAMES utf8mb4 COLLATE utf8mb4_bin;
mysql> SELECT LOWER('aA'), UPPER('zZ');
+-------------+-------------+
| LOWER('aA') | UPPER('zZ') |
+-------------+-------------+
| aa          | ZZ          |
+-------------+-------------+
```

The concept of lettercase does not apply to bytes in a binary string. To perform lettercase conversion, the string must be converted to a nonbinary string:

```
mysql> SET NAMES binary;
mysql> SELECT LOWER('aA'), LOWER(CONVERT('aA' USING utf8mb4));
+-------------+-----------------------------------+
| LOWER('aA') | LOWER(CONVERT('aA' USING utf8mb4)) |
+-------------+-----------------------------------+
| aA          | aa                                |
+-------------+-----------------------------------+
```

Trailing Space Handling in Comparisons

Many MySQL collations have a pad attribute of `PAD SPACE`. The Unicode collations based on UCA 9.0.0 and higher have a pad attribute of `NO PAD`; see Section 10.10.1, “Unicode Character Sets”.

The pad attribute determines how trailing spaces are treated for comparison of nonbinary strings (`CHAR`, `VARCHAR`, and `TEXT` values):

- For `PAD SPACE` collations, trailing spaces are insignificant in comparisons; strings are compared without regard to any trailing spaces.
- `NO PAD` collations treat spaces at the end of strings like any other character.

The differing behaviors can be demonstrated using the two `utf8mb4` binary collations, one of which is `PAD SPACE`, the other of which is `NO PAD`. The example also shows how to use the `INFORMATION_SCHEMA COLLATIONS` table to determine the pad attribute for collations.

```
mysql> SELECT COLLATION_NAME, PAD_ATTRIBUTE
       FROM INFORMATION_SCHEMA.COLLATIONS
       WHERE COLLATION_NAME LIKE 'utf8mb4%bin';
+-----------------+---------------+
| COLLATION_NAME  | PAD_ATTRIBUTE |
+-----------------+---------------+
| utf8mb4_bin     | PAD SPACE     |
| utf8mb4_0900_bin | NO PAD       |
+-----------------+---------------+
mysql> SET NAMES utf8mb4 COLLATE utf8mb4_bin;
mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          1 |
+------------+
mysql> SET NAMES utf8mb4 COLLATE utf8mb4_0900_bin;
mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          0 |
+------------+
```

For binary strings, all characters are significant in comparisons, including trailing spaces:

```
mysql> SET NAMES binary;
mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          0 |
+------------+
```

Trailing Space Handling for Inserts and Retrievals

`CHAR(N)` columns store nonbinary strings. For inserts, values shorter than *`N`* characters are extended with spaces. For retrievals, trailing spaces are removed.

`BINARY(N)` columns store binary strings. For inserts, values shorter than *`N`* bytes are extended with `0x00` bytes. For retrievals, nothing is removed; a value of the declared length is always returned.

```
mysql> CREATE TABLE t1 (
         a CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin,
         b BINARY(10)
       );
mysql> INSERT INTO t1 VALUES ('a','a');
mysql> SELECT HEX(a), HEX(b) FROM t1;
+--------+----------------------+
| HEX(a) | HEX(b)               |
+--------+----------------------+
| 61     | 61000000000000000000 |
+--------+----------------------+
```

10.8.6 Examples of the Effect of Collation

Example 1: Sorting German Umlauts

Suppose that column `X` in table `T` has these `latin1` column values:

```
Muffler
Müller
MX Systems
MySQL
```

Suppose also that the column values are retrieved using the following statement:

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```

The following table shows the resulting order of the values if we use `ORDER BY` with different collations.

`latin1_swedish_ci`	`latin1_german1_ci`	`latin1_german2_ci`
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

The character that causes the different sort orders in this example is the U with two dots over it (ü), which the Germans call “U-umlaut.”

- The first column shows the result of the `SELECT` using the Swedish/Finnish collating rule, which says that U-umlaut sorts with Y.

- The second column shows the result of the `SELECT` using the German DIN-1 rule, which says that U-umlaut sorts with U.
- The third column shows the result of the `SELECT` using the German DIN-2 rule, which says that U-umlaut sorts with UE.

Example 2: Searching for German Umlauts

Suppose that you have three tables that differ only by the character set and collation used:

```
mysql> SET NAMES utf8;
mysql> CREATE TABLE german1 (
         c CHAR(10)
       ) CHARACTER SET latin1 COLLATE latin1_german1_ci;
mysql> CREATE TABLE german2 (
         c CHAR(10)
       ) CHARACTER SET latin1 COLLATE latin1_german2_ci;
mysql> CREATE TABLE germanutf8 (
         c CHAR(10)
       ) CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Each table contains two records:

```
mysql> INSERT INTO german1 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO german2 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO germanutf8 VALUES ('Bar'), ('Bär');
```

Two of the above collations have an `A = Ä` equality, and one has no such equality (`latin1_german2_ci`). For that reason, you'll get these results in comparisons:

```
mysql> SELECT * FROM german1 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bar  |
| Bär  |
+------+
mysql> SELECT * FROM german2 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bär  |
+------+
mysql> SELECT * FROM germanutf8 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bar  |
| Bär  |
+------+
```

This is not a bug but rather a consequence of the sorting properties of `latin1_german1_ci` and `utf8_unicode_ci` (the sorting shown is done according to the German DIN 5007 standard).

10.8.7 Using Collation in INFORMATION_SCHEMA Searches

String columns in `INFORMATION_SCHEMA` tables have a collation of `utf8_general_ci`, which is case insensitive. However, for values that correspond to objects that are represented in the file system, such as databases and tables, searches in `INFORMATION_SCHEMA` string columns can be case-sensitive or insensitive, depending on the characteristics of the underlying file system and the

`lower_case_table_names` system variable setting. For example, searches may be case-sensitive if the file system is case-sensitive. This section describes this behavior and how to modify it if necessary.

Suppose that a query searches the `SCHEMATA.SCHEMA_NAME` column for the `test` database. On Linux, file systems are case-sensitive, so comparisons of `SCHEMATA.SCHEMA_NAME` with `'test'` match, but comparisons with `'TEST'` do not:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME = 'test';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME = 'TEST';
Empty set (0.00 sec)
```

These results occur with the `lower_case_table_names` system variable set to 0. A `lower_case_table_names` setting of 1 or 2 causes the second query to return the same (nonempty) result as the first query.

Note

It is prohibited to start the server with a `lower_case_table_names` setting that is different from the setting used when the server was initialized.

On Windows or macOS, file systems are not case-sensitive, so comparisons match both `'test'` and `'TEST'`:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME = 'test';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME = 'TEST';
+-------------+
| SCHEMA_NAME |
+-------------+
| TEST        |
+-------------+
```

The value of `lower_case_table_names` makes no difference in this context.

The preceding behavior occurs because the `utf8_general_ci` collation is not used for `INFORMATION_SCHEMA` queries when searching for values that correspond to objects represented in the file system.

If the result of a string operation on an `INFORMATION_SCHEMA` column differs from expectations, a workaround is to use an explicit `COLLATE` clause to force a suitable collation (see Section 10.8.1, “Using COLLATE in SQL Statements”). For example, to perform a case-insensitive search, use `COLLATE` with the `INFORMATION_SCHEMA` column name:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
```

```
      WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'test';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
      WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'TEST';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+
```

You can also use the `UPPER()` or `LOWER()` function:

```
WHERE UPPER(SCHEMA_NAME) = 'TEST'
WHERE LOWER(SCHEMA_NAME) = 'test'
```

Although a case-insensitive comparison can be performed even on platforms with case-sensitive file systems, as just shown, it is not necessarily always the right thing to do. On such platforms, it is possible to have multiple objects with names that differ only in lettercase. For example, tables named `city`, `CITY`, and `City` can all exist simultaneously. Consider whether a search should match all such names or just one and write queries accordingly. The first of the following comparisons (with `utf8_bin`) is case sensitive; the others are not:

```
WHERE TABLE_NAME COLLATE utf8_bin = 'City'
WHERE TABLE_NAME COLLATE utf8_general_ci = 'city'
WHERE UPPER(TABLE_NAME) = 'CITY'
WHERE LOWER(TABLE_NAME) = 'city'
```

Searches in `INFORMATION_SCHEMA` string columns for values that refer to `INFORMATION_SCHEMA` itself do use the `utf8_general_ci` collation because `INFORMATION_SCHEMA` is a “virtual” database not represented in the file system. For example, comparisons with `SCHEMATA.SCHEMA_NAME` match `'information_schema'` or `'INFORMATION_SCHEMA'` regardless of platform:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
      WHERE SCHEMA_NAME = 'information_schema';
+--------------------+
| SCHEMA_NAME        |
+--------------------+
| information_schema |
+--------------------+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
      WHERE SCHEMA_NAME = 'INFORMATION_SCHEMA';
+--------------------+
| SCHEMA_NAME        |
+--------------------+
| information_schema |
+--------------------+
```

10.9 Unicode Support

The Unicode Standard includes characters from the Basic Multilingual Plane (BMP) and supplementary characters that lie outside the BMP. This section describes support for Unicode in MySQL. For information about the Unicode Standard itself, visit the Unicode Consortium website.

BMP characters have these characteristics:

- Their code point values are between 0 and 65535 (or `U+0000` and `U+FFFF`).
- They can be encoded in a variable-length encoding using 8, 16, or 24 bits (1 to 3 bytes).
- They can be encoded in a fixed-length encoding using 16 bits (2 bytes).
- They are sufficient for almost all characters in major languages.

Supplementary characters lie outside the BMP:

- Their code point values are between `U+10000` and `U+10FFFF`).
- Unicode support for supplementary characters requires character sets that have a range outside BMP characters and therefore take more space than BMP characters (up to 4 bytes per character).

The UTF-8 (Unicode Transformation Format with 8-bit units) method for encoding Unicode data is implemented according to RFC 3629, which describes encoding sequences that take from one to four bytes. The idea of UTF-8 is that various Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and Middle East script letters fit into a 2-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.
- Korean, Chinese, and Japanese ideographs use 3-byte or 4-byte sequences.

MySQL supports these Unicode character sets:

- `utf8mb4`: A UTF-8 encoding of the Unicode character set using one to four bytes per character.
- `utf8mb3`: A UTF-8 encoding of the Unicode character set using one to three bytes per character.
- `utf8`: An alias for `utf8mb3`.
- `ucs2`: The UCS-2 encoding of the Unicode character set using two bytes per character.
- `utf16`: The UTF-16 encoding for the Unicode character set using two or four bytes per character. Like `ucs2` but with an extension for supplementary characters.
- `utf16le`: The UTF-16LE encoding for the Unicode character set. Like `utf16` but little-endian rather than big-endian.
- `utf32`: The UTF-32 encoding for the Unicode character set using four bytes per character.

Note

The `utf8mb3` character set is deprecated and will be removed in a future MySQL release. Please use `utf8mb4` instead. Although `utf8` is currently an alias for `utf8mb3`, at some point `utf8` will become a reference to `utf8mb4`. To avoid ambiguity about the meaning of `utf8`, consider specifying `utf8mb4` explicitly for character set references instead of `utf8`.

Table 10.2, “Unicode Character Set General Characteristics”, summarizes the general characteristics of Unicode character sets supported by MySQL.

Table 10.2 Unicode Character Set General Characteristics

Character Set	Supported Characters	Required Storage Per Character
`utf8mb3`, `utf8`	BMP only	1, 2, or 3 bytes

Character Set	Supported Characters	Required Storage Per Character
ucs2	BMP only	2 bytes
utf8mb4	BMP and supplementary	1, 2, 3, or 4 bytes
utf16	BMP and supplementary	2 or 4 bytes
utf16le	BMP and supplementary	2 or 4 bytes
utf32	BMP and supplementary	4 bytes

Characters outside the BMP compare as REPLACEMENT CHARACTER and convert to `'?'` when converted to a Unicode character set that supports only BMP characters (utf8mb3 or ucs2).

If you use character sets that support supplementary characters and thus are “wider” than the BMP-only utf8mb3 and ucs2 character sets, there are potential incompatibility issues for your applications; see Section 10.9.8, “Converting Between 3-Byte and 4-Byte Unicode Character Sets”. That section also describes how to convert tables from the (3-byte) utf8mb3 to the (4-byte) utf8mb4, and what constraints may apply in doing so.

A similar set of collations is available for most Unicode character sets. For example, each has a Danish collation, the names of which are utf8mb4_danish_ci, utf8mb3_danish_ci, utf8_danish_ci, ucs2_danish_ci, utf16_danish_ci, and utf32_danish_ci. The exception is utf16le, which has only two collations. For information about Unicode collations and their differentiating properties, including collation properties for supplementary characters, see Section 10.10.1, “Unicode Character Sets”.

The MySQL implementation of UCS-2, UTF-16, and UTF-32 stores characters in big-endian byte order and does not use a byte order mark (BOM) at the beginning of values. Other database systems might use little-endian byte order or a BOM. In such cases, conversion of values will need to be performed when transferring data between those systems and MySQL. The implementation of UTF-16LE is little-endian.

MySQL uses no BOM for UTF-8 values.

Client applications that communicate with the server using Unicode should set the client character set accordingly (for example, by issuing a `SET NAMES 'utf8mb4'` statement). Some character sets cannot be used as the client character set. Attempting to use them with `SET NAMES` or `SET CHARACTER SET` produces an error. See Impermissible Client Character Sets.

The following sections provide additional detail on the Unicode character sets in MySQL.

10.9.1 The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding)

The utfmb4 character set has these characteristics:

- Supports BMP and supplementary characters.
- Requires a maximum of four bytes per multibyte character.

utf8mb4 contrasts with the utf8mb3 character set, which supports only BMP characters and uses a maximum of three bytes per character:

- For a BMP character, utf8mb4 and utf8mb3 have identical storage characteristics: same code values, same encoding, same length.
- For a supplementary character, utf8mb4 requires four bytes to store it, whereas utf8mb3 cannot store the character at all. When converting utf8mb3 columns to utf8mb4, you need not worry about converting supplementary characters because there will be none.

`utf8mb4` is a superset of `utf8mb3`, so for an operation such as the following concatenation, the result has character set `utf8mb4` and the collation of `utf8mb4_col`:

```
SELECT CONCAT(utf8mb3_col, utf8mb4_col);
```

Similarly, the following comparison in the `WHERE` clause works according to the collation of `utf8mb4_col`:

```
SELECT * FROM utf8mb3_tbl, utf8mb4_tbl
WHERE utf8mb3_tbl.utf8mb3_col = utf8mb4_tbl.utf8mb4_col;
```

For information about data type storage as it relates to multibyte character sets, see String Type Storage Requirements.

10.9.2 The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding)

The `utf8mb3` character set has these characteristics:

- Supports BMP characters only (no support for supplementary characters)
- Requires a maximum of three bytes per multibyte character.

Applications that use UTF-8 data but require supplementary character support should use `utf8mb4` rather than `utf8mb3` (see Section 10.9.1, “The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding)”).

Exactly the same set of characters is available in `utf8mb3` and `ucs2`. That is, they have the same repertoire.

`utf8` is an alias for `utf8mb3`; the character limit is implicit, rather than explicit in the name.

Note

The `utf8mb3` character set is deprecated and will be removed in a future MySQL release. Please use `utf8mb4` instead. Although `utf8` is currently an alias for `utf8mb3`, at some point `utf8` will become a reference to `utf8mb4`. To avoid ambiguity about the meaning of `utf8`, consider specifying `utf8mb4` explicitly for character set references instead of `utf8`.

`utf8mb3` can be used in `CHARACTER SET` clauses, and `utf8mb3_collation_substring` in `COLLATE` clauses, where *`collation_substring`* is `bin`, `czech_ci`, `danish_ci`, `esperanto_ci`, `estonian_ci`, and so forth. For example:

```
CREATE TABLE t (s1 CHAR(1) CHARACTER SET utf8mb3;
SELECT * FROM t WHERE s1 COLLATE utf8mb3_general_ci = 'x';
DECLARE x VARCHAR(5) CHARACTER SET utf8mb3 COLLATE utf8mb3_danish_ci;
SELECT CAST('a' AS CHAR CHARACTER SET utf8) COLLATE utf8_czech_ci;
```

MySQL immediately converts instances of `utf8mb3` in statements to `utf8`, so in statements such as `SHOW CREATE TABLE` or `SELECT CHARACTER_SET_NAME FROM INFORMATION_SCHEMA.COLUMNS` or `SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLUMNS`, users will see the name `utf8` or `utf8_collation_substring`.

`utf8mb3` is also valid in contexts other than `CHARACTER SET` clauses. For example:

```
```

```
mysqld --character-set-server=utf8mb3
```

```
SET NAMES 'utf8mb3'; /* and other SET statements that have similar effect */
SELECT _utf8mb3 'a';
```

For information about data type storage as it relates to multibyte character sets, see String Type Storage Requirements.

10.9.3 The utf8 Character Set (Alias for utf8mb3)

`utf8` is an alias for the `utf8mb3` character set. For more information, see Section 10.9.2, “The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding)”.

Note

The `utf8mb3` character set is deprecated and will be removed in a future MySQL release. Please use `utf8mb4` instead. Although `utf8` is currently an alias for `utf8mb3`, at some point `utf8` will become a reference to `utf8mb4`. To avoid ambiguity about the meaning of `utf8`, consider specifying `utf8mb4` explicitly for character set references instead of `utf8`.

10.9.4 The ucs2 Character Set (UCS-2 Unicode Encoding)

In UCS-2, every character is represented by a 2-byte Unicode code with the most significant byte first. For example: `LATIN CAPITAL LETTER A` has the code `0x0041` and it is stored as a 2-byte sequence: `0x00 0x41`. `CYRILLIC SMALL LETTER YERU` (Unicode `0x044B`) is stored as a 2-byte sequence: `0x04 0x4B`. For Unicode characters and their codes, please refer to the Unicode Consortium website.

The `ucs2` character set has these characteristics:

- Supports BMP characters only (no support for supplementary characters)
- Uses a fixed-length 16-bit encoding and requires two bytes per character.

10.9.5 The utf16 Character Set (UTF-16 Unicode Encoding)

The `utf16` character set is the `ucs2` character set with an extension that enables encoding of supplementary characters:

- For a BMP character, `utf16` and `ucs2` have identical storage characteristics: same code values, same encoding, same length.
- For a supplementary character, `utf16` has a special sequence for representing the character using 32 bits. This is called the “surrogate” mechanism: For a number greater than `0xffff`, take 10 bits and add them to `0xd800` and put them in the first 16-bit word, take 10 more bits and add them to `0xdc00` and put them in the next 16-bit word. Consequently, all supplementary characters require 32 bits, where the first 16 bits are a number between `0xd800` and `0xdbff`, and the last 16 bits are a number between `0xdc00` and `0xdfff`. Examples are in Section 15.5 Surrogates Area of the Unicode 4.0 document.

Because `utf16` supports surrogates and `ucs2` does not, there is a validity check that applies only in `utf16`: You cannot insert a top surrogate without a bottom surrogate, or vice versa. For example:

```
INSERT INTO t (ucs2_column) VALUES (0xd800); /* legal */
INSERT INTO t (utf16_column)VALUES (0xd800); /* illegal */
```

There is no validity check for characters that are technically valid but are not true Unicode (that is, characters that Unicode considers to be “unassigned code points” or “private use” characters or even “illegals” like `0xffff`). For example, since `U+F8FF` is the Apple Logo, this is legal:

```
INSERT INTO t (utf16_column)VALUES (0xf8ff); /* legal */
```

Such characters cannot be expected to mean the same thing to everyone.

Because MySQL must allow for the worst case (that one character requires four bytes) the maximum length of a `utf16` column or index is only half of the maximum length for a `ucs2` column or index. For example, the maximum length of a `MEMORY` table index key is 3072 bytes, so these statements create tables with the longest permitted indexes for `ucs2` and `utf16` columns:

```
CREATE TABLE tf (s1 VARCHAR(1536) CHARACTER SET ucs2) ENGINE=MEMORY;
CREATE INDEX i ON tf (s1);
CREATE TABLE tg (s1 VARCHAR(768) CHARACTER SET utf16) ENGINE=MEMORY;
CREATE INDEX i ON tg (s1);
```

10.9.6 The utf16le Character Set (UTF-16LE Unicode Encoding)

This is the same as `utf16` but is little-endian rather than big-endian.

10.9.7 The utf32 Character Set (UTF-32 Unicode Encoding)

The `utf32` character set is fixed length (like `ucs2` and unlike `utf16`). `utf32` uses 32 bits for every character, unlike `ucs2` (which uses 16 bits for every character), and unlike `utf16` (which uses 16 bits for some characters and 32 bits for others).

`utf32` takes twice as much space as `ucs2` and more space than `utf16`, but `utf32` has the same advantage as `ucs2` that it is predictable for storage: The required number of bytes for `utf32` equals the number of characters times 4. Also, unlike `utf16`, there are no tricks for encoding in `utf32`, so the stored value equals the code value.

To demonstrate how the latter advantage is useful, here is an example that shows how to determine a `utf8mb4` value given the `utf32` code value:

```
/* Assume code value = 100cc LINEAR B WHEELED CHARIOT */
CREATE TABLE tmp (utf32_col CHAR(1) CHARACTER SET utf32,
                  utf8mb4_col CHAR(1) CHARACTER SET utf8mb4);
INSERT INTO tmp VALUES (0x000100cc,NULL);
UPDATE tmp SET utf8mb4_col = utf32_col;
SELECT HEX(utf32_col),HEX(utf8mb4_col) FROM tmp;
```

MySQL is very forgiving about additions of unassigned Unicode characters or private-use-area characters. There is in fact only one validity check for `utf32`: No code value may be greater than `0x10ffff`. For example, this is illegal:

```
INSERT INTO t (utf32_column) VALUES (0x110000); /* illegal */
```

10.9.8 Converting Between 3-Byte and 4-Byte Unicode Character Sets

This section describes issues that you may face when converting character data between the `utf8mb3` and `utf8mb4` character sets.

Note

This discussion focuses primarily on converting between `utf8mb3` and `utf8mb4`, but similar principles apply to converting between the `ucs2` character set and character sets such as `utf16` or `utf32`.

The `utf8mb3` and `utf8mb4` character sets differ as follows:

- `utf8mb3` supports only characters in the Basic Multilingual Plane (BMP). `utf8mb4` additionally supports supplementary characters that lie outside the BMP.
- `utf8mb3` uses a maximum of three bytes per character. `utf8mb4` uses a maximum of four bytes per character.

Note

This discussion refers to the `utf8mb3` and `utf8mb4` character set names to be explicit about referring to 3-byte and 4-byte UTF-8 character set data. The exception is that in table definitions, `utf8` is used because MySQL converts instances of `utf8mb3` specified in such definitions to `utf8`, which is an alias for `utf8mb3`.

One advantage of converting from `utf8mb3` to `utf8mb4` is that this enables applications to use supplementary characters. One tradeoff is that this may increase data storage space requirements.

In terms of table content, conversion from `utf8mb3` to `utf8mb4` presents no problems:

- For a BMP character, `utf8mb4` and `utf8mb3` have identical storage characteristics: same code values, same encoding, same length.
- For a supplementary character, `utf8mb4` requires four bytes to store it, whereas `utf8mb3` cannot store the character at all. When converting `utf8mb3` columns to `utf8mb4`, you need not worry about converting supplementary characters because there will be none.

In terms of table structure, these are the primary potential incompatibilities:

- For the variable-length character data types (`VARCHAR` and the `TEXT` types), the maximum permitted length in characters is less for `utf8mb4` columns than for `utf8mb3` columns.
- For all character data types (`CHAR`, `VARCHAR`, and the `TEXT` types), the maximum number of characters that can be indexed is less for `utf8mb4` columns than for `utf8mb3` columns.

Consequently, to convert tables from `utf8mb3` to `utf8mb4`, it may be necessary to change some column or index definitions.

Tables can be converted from `utf8mb3` to `utf8mb4` by using `ALTER TABLE`. Suppose that a table has this definition:

```
CREATE TABLE t1 (
  col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,
  col2 CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL
) CHARACTER SET utf8;
```

The following statement converts `t1` to use `utf8mb4`:

```
ALTER TABLE t1
  DEFAULT CHARACTER SET utf8mb4,
```

```
MODIFY col1 CHAR(10)
  CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
MODIFY col2 CHAR(10)
  CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL;
```

The catch when converting from `utf8mb3` to `utf8mb4` is that the maximum length of a column or index key is unchanged in terms of *bytes*. Therefore, it is smaller in terms of *characters* because the maximum length of a character is four bytes instead of three. For the `CHAR`, `VARCHAR`, and `TEXT` data types, watch for these issues when converting your MySQL tables:

- Check all definitions of `utf8mb3` columns and make sure they will not exceed the maximum length for the storage engine.
- Check all indexes on `utf8mb3` columns and make sure they will not exceed the maximum length for the storage engine. Sometimes the maximum can change due to storage engine enhancements.

If the preceding conditions apply, you must either reduce the defined length of columns or indexes, or continue to use `utf8mb3` rather than `utf8mb4`.

Here are some examples where structural changes may be needed:

- A `TINYTEXT` column can hold up to 255 bytes, so it can hold up to 85 3-byte or 63 4-byte characters. Suppose that you have a `TINYTEXT` column that uses `utf8mb3` but must be able to contain more than 63 characters. You cannot convert it to `utf8mb4` unless you also change the data type to a longer type such as `TEXT`.

  Similarly, a very long `VARCHAR` column may need to be changed to one of the longer `TEXT` types if you want to convert it from `utf8mb3` to `utf8mb4`.

- `InnoDB` has a maximum index length of 767 bytes for tables that use `COMPACT` or `REDUNDANT` row format, so for `utf8mb3` or `utf8mb4` columns, you can index a maximum of 255 or 191 characters, respectively. If you currently have `utf8mb3` columns with indexes longer than 191 characters, you must index a smaller number of characters.

  In an `InnoDB` table that uses `COMPACT` or `REDUNDANT` row format, these column and index definitions are legal:

  ```
  col1 VARCHAR(500) CHARACTER SET utf8, INDEX (col1(255))
  ```

  To use `utf8mb4` instead, the index must be smaller:

  ```
  col1 VARCHAR(500) CHARACTER SET utf8mb4, INDEX (col1(191))
  ```

Note

For `InnoDB` tables that use `COMPRESSED` or `DYNAMIC` row format, index key prefixes longer than 767 bytes (up to 3072 bytes) are permitted. Tables created with these row formats enable you to index a maximum of 1024 or 768 characters for `utf8mb3` or `utf8mb4` columns, respectively. For related information, see Section 15.6.1.6, “Limits on InnoDB Tables”, and DYNAMIC Row Format.

The preceding types of changes are most likely to be required only if you have very long columns or indexes. Otherwise, you should be able to convert your tables from `utf8mb3` to `utf8mb4` without problems, using `ALTER TABLE` as described previously.

The following items summarize other potential incompatibilities:

- `SET NAMES 'utf8mb4'` causes use of the 4-byte character set for connection character sets. As long as no 4-byte characters are sent from the server, there should be no problems. Otherwise, applications that expect to receive a maximum of three bytes per character may have problems. Conversely, applications that expect to send 4-byte characters must ensure that the server understands them.
- For replication, if character sets that support supplementary characters are to be used on the master, all slaves must understand them as well.

  Also, keep in mind the general principle that if a table has different definitions on the master and slave, this can lead to unexpected results. For example, the differences in maximum index key length make it risky to use `utf8mb3` on the master and `utf8mb4` on the slave.

If you have converted to `utf8mb4`, `utf16`, `utf16le`, or `utf32`, and then decide to convert back to `utf8mb3` or `ucs2` (for example, to downgrade to an older version of MySQL), these considerations apply:

- `utf8mb3` and `ucs2` data should present no problems.
- The server must be recent enough to recognize definitions referring to the character set from which you are converting.
- For object definitions that refer to the `utf8mb4` character set, you can dump them with `mysqldump` prior to downgrading, edit the dump file to change instances of `utf8mb4` to `utf8`, and reload the file in the older server, as long as there are no 4-byte characters in the data. The older server will see `utf8` in the dump file object definitions and create new objects that use the (3-byte) `utf8` character set.

10.10 Supported Character Sets and Collations

This section indicates which character sets MySQL supports. There is one subsection for each group of related character sets. For each character set, the permissible collations are listed.

To list the available character sets and their default collations, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table. For example:

```
mysql> SHOW CHARACTER SET;
+----------+---------------------------------+---------------------+--------+
| Charset  | Description                     | Default collation   | Maxlen |
+----------+---------------------------------+---------------------+--------+
| armscii8 | ARMSCII-8 Armenian              | armscii8_general_ci |      1 |
| ascii    | US ASCII                        | ascii_general_ci    |      1 |
| big5     | Big5 Traditional Chinese        | big5_chinese_ci     |      2 |
| binary   | Binary pseudo charset           | binary              |      1 |
| cp1250   | Windows Central European        | cp1250_general_ci   |      1 |
| cp1251   | Windows Cyrillic                | cp1251_general_ci   |      1 |
| cp1256   | Windows Arabic                  | cp1256_general_ci   |      1 |
| cp1257   | Windows Baltic                  | cp1257_general_ci   |      1 |
| cp850    | DOS West European               | cp850_general_ci    |      1 |
| cp852    | DOS Central European            | cp852_general_ci    |      1 |
| cp866    | DOS Russian                     | cp866_general_ci    |      1 |
| cp932    | SJIS for Windows Japanese       | cp932_japanese_ci   |      2 |
| dec8     | DEC West European               | dec8_swedish_ci     |      1 |
| eucjpms  | UJIS for Windows Japanese       | eucjpms_japanese_ci |      3 |
| euckr    | EUC-KR Korean                   | euckr_korean_ci     |      2 |
| gb18030  | China National Standard GB18030 | gb18030_chinese_ci  |      4 |
| gb2312   | GB2312 Simplified Chinese       | gb2312_chinese_ci   |      2 |
| gbk      | GBK Simplified Chinese          | gbk_chinese_ci      |      2 |
| geostd8  | GEOSTD8 Georgian                | geostd8_general_ci  |      1 |
| greek    | ISO 8859-7 Greek                | greek_general_ci    |      1 |
| hebrew   | ISO 8859-8 Hebrew               | hebrew_general_ci   |      1 |
| hp8      | HP West European                | hp8_english_ci      |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak      | keybcs2_general_ci  |      1 |
```

```
| koi8r    | KOI8-R Relcom Russian           | koi8r_general_ci    |      1 |
| koi8u    | KOI8-U Ukrainian                | koi8u_general_ci    |      1 |
| latin1   | cp1252 West European            | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European     | latin2_general_ci   |      1 |
| latin5   | ISO 8859-9 Turkish              | latin5_turkish_ci   |      1 |
| latin7   | ISO 8859-13 Baltic              | latin7_general_ci   |      1 |
| macce    | Mac Central European            | macce_general_ci    |      1 |
| macroman | Mac West European               | macroman_general_ci |      1 |
| sjis     | Shift-JIS Japanese              | sjis_japanese_ci    |      2 |
| swe7     | 7bit Swedish                    | swe7_swedish_ci     |      1 |
| tis620   | TIS620 Thai                     | tis620_thai_ci      |      1 |
| ucs2     | UCS-2 Unicode                   | ucs2_general_ci     |      2 |
| ujis     | EUC-JP Japanese                 | ujis_japanese_ci    |      3 |
| utf16    | UTF-16 Unicode                  | utf16_general_ci    |      4 |
| utf16le  | UTF-16LE Unicode                | utf16le_general_ci  |      4 |
| utf32    | UTF-32 Unicode                  | utf32_general_ci    |      4 |
| utf8     | UTF-8 Unicode                   | utf8_general_ci     |      3 |
| utf8mb4  | UTF-8 Unicode                   | utf8mb4_0900_ai_ci  |      4 |
+----------+---------------------------------+---------------------+--------+
```

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

10.10.1 Unicode Character Sets

This section describes the collations available for Unicode character sets and their differentiating properties. For general information about Unicode, see Section 10.9, “Unicode Support”.

MySQL supports multiple Unicode character sets:

- `utf8mb4`: A UTF-8 encoding of the Unicode character set using one to four bytes per character.
- `utf8mb3`: A UTF-8 encoding of the Unicode character set using one to three bytes per character.
- `utf8`: An alias for `utf8mb3`.
- `ucs2`: The UCS-2 encoding of the Unicode character set using two bytes per character.
- `utf16`: The UTF-16 encoding for the Unicode character set using two or four bytes per character. Like `ucs2` but with an extension for supplementary characters.
- `utf16le`: The UTF-16LE encoding for the Unicode character set. Like `utf16` but little-endian rather than big-endian.
- `utf32`: The UTF-32 encoding for the Unicode character set using four bytes per character.

Note

The `utf8mb3` character set is deprecated and will be removed in a future MySQL release. Please use `utf8mb4` instead. Although `utf8` is currently an alias for `utf8mb3`, at some point `utf8` will become a reference to `utf8mb4`. To avoid ambiguity about the meaning of `utf8`, consider specifying `utf8mb4` explicitly for character set references instead of `utf8`.

`utf8mb4`, `utf16`, `utf16le`, and `utf32` support Basic Multilingual Plane (BMP) characters and supplementary characters that lie outside the BMP. `utf8` and `ucs2` support only BMP characters.

Most Unicode character sets have a general collation (indicated by `_general` in the name or by the absence of a language specifier), a binary collation (indicated by `_bin` in the name),

and several language-specific collations (indicated by language specifiers). For example, for `utf8mb4`, `utf8mb4_general_ci` and `utf8mb4_bin` are its general and binary collations, and `utf8mb4_danish_ci` is one of its language-specific collations.

Most character sets have a single binary collation. `utf8mb4` is an exception that has two: `utf8mb4_bin` and (as of MySQL 8.0.17) `utf8mb4_0900_bin`. These two binary collations have the same sort order but are distinguished by their pad attribute and collating weight characteristics. See Collation Pad Attributes, and Character Collating Weights.

Collation support for `utf16le` is limited. The only collations available are `utf16le_general_ci` and `utf16le_bin`. These are similar to `utf16_general_ci` and `utf16_bin`.

- Unicode Collation Algorithm (UCA) Versions
- Collation Pad Attributes
- Language-Specific Collations
- _general_ci Versus _unicode_ci Collations
- Character Collating Weights
- Miscellaneous Information

Unicode Collation Algorithm (UCA) Versions

MySQL implements the *`xxx`*`_unicode_ci` collations according to the Unicode Collation Algorithm (UCA) described at http://www.unicode.org/reports/tr10/. The collation uses the version-4.0.0 UCA weight keys: http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt. The *`xxx`*`_unicode_ci` collations have only partial support for the Unicode Collation Algorithm. Some characters are not supported, and combining marks are not fully supported. This affects primarily Vietnamese, Yoruba, and some smaller languages such as Navajo. A combined character is considered different from the same character written with a single unicode character in string comparisons, and the two characters are considered to have a different length (for example, as returned by the `CHAR_LENGTH()` function or in result set metadata).

Unicode collations based on UCA versions higher than 4.0.0 include the version in the collation name. Examples:

- `utf8mb4_unicode_520_ci` is based on UCA 5.2.0 weight keys (http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt),
- `utf8mb4_0900_ai_ci` is based on UCA 9.0.0 weight keys (http://www.unicode.org/Public/UCA/9.0.0/allkeys.txt).

The `LOWER()` and `UPPER()` functions perform case folding according to the collation of their argument. A character that has uppercase and lowercase versions only in a Unicode version higher than 4.0.0 is converted by these functions only if the argument collation uses a high enough UCA version.

Collation Pad Attributes

Collations based on UCA 9.0.0 and higher are faster than collations based on UCA versions prior to 9.0.0. They also have a pad attribute of `NO PAD`, in contrast to `PAD SPACE` as used in collations based on UCA versions prior to 9.0.0. For comparison of nonbinary strings, `NO PAD` collations treat spaces at the end of strings like any other character (see Trailing Space Handling in Comparisons).

To determine the pad attribute for a collation, use the `INFORMATION_SCHEMA COLLATIONS` table, which has a `PAD_ATTRIBUTE` column. For example:

```
mysql> SELECT COLLATION_NAME, PAD_ATTRIBUTE
       FROM INFORMATION_SCHEMA.COLLATIONS
       WHERE CHARACTER_SET_NAME = 'utf8mb4';
+----------------------------+---------------+
| COLLATION_NAME             | PAD_ATTRIBUTE |
+----------------------------+---------------+
| utf8mb4_general_ci         | PAD SPACE     |
| utf8mb4_bin                | PAD SPACE     |
| utf8mb4_unicode_ci         | PAD SPACE     |
| utf8mb4_icelandic_ci       | PAD SPACE     |
...
| utf8mb4_0900_ai_ci         | NO PAD        |
| utf8mb4_de_pb_0900_ai_ci   | NO PAD        |
| utf8mb4_is_0900_ai_ci      | NO PAD        |
...
| utf8mb4_ja_0900_as_cs      | NO PAD        |
| utf8mb4_ja_0900_as_cs_ks   | NO PAD        |
| utf8mb4_0900_as_ci         | NO PAD        |
| utf8mb4_ru_0900_ai_ci      | NO PAD        |
| utf8mb4_ru_0900_as_cs      | NO PAD        |
| utf8mb4_zh_0900_as_cs      | NO PAD        |
| utf8mb4_0900_bin           | NO PAD        |
+----------------------------+---------------+
```

Comparison of nonbinary string values (`CHAR`, `VARCHAR`, and `TEXT`) that have a `NO PAD` collation differ from other collations with respect to trailing spaces. For example, `'a'` and `'a '` compare as different strings, not the same string. This can be seen using the binary collations for `utf8mb4`. The pad attribute for `utf8mb4_bin` is `PAD SPACE`, whereas for `utf8mb4_0900_bin` it is `NO PAD`. Consequently, operations involving `utf8mb4_0900_bin` do not add trailing spaces, and comparisons involving strings with trailing spaces may differ for the two collations:

```
mysql> CREATE TABLE t1 (c CHAR(10) COLLATE utf8mb4_bin);
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO t1 VALUES('a');
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM t1 WHERE c = 'a ';
+------+
| c    |
+------+
| a    |
+------+
1 row in set (0.00 sec)

mysql> ALTER TABLE t1 MODIFY c CHAR(10) COLLATE utf8mb4_0900_bin;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t1 WHERE c = 'a ';
Empty set (0.00 sec)
```

Language-Specific Collations

MySQL implements language-specific Unicode collations if the ordering based only on the Unicode Collation Algorithm (UCA) does not work well for a language. Language-specific collations are UCA-based, with additional language tailoring rules. Examples of such rules appear later in this section. For questions about particular language orderings, unicode.org provides Common Locale Data Repository (CLDR) collation charts at http://www.unicode.org/cldr/charts/30/collation/index.html.

For example, the nonlanguage-specific `utf8mb4_0900_ai_ci` and language-specific `utf8mb4_`*`LOCALE`*`_0900_ai_ci` Unicode collations each have these characteristics:

- The collation is based on UCA 9.0.0 and CLDR v30, is accent insensitive, and case insensitive. These characteristics are indicated by `_0900`, `_ai`, and `_ci` in the collation name. Exception: `utf8mb4_la_0900_ai_ci` is not based on CLDR because Classical Latin is not defined in CLDR.
- The collation works for all characters in the range [U+0, U+10FFFF].
- If the collation is not language specific, it sorts all characters, including supplementary characters, in default order (described following). If the collation is language specific, it sorts characters of the language correctly according to language-specific rules, and characters not in the language in default order.
- By default, the collation sorts characters having a code point listed in the DUCET table (Default Unicode Collation Element Table) according to the weight value assigned in the table. The collation sorts characters not having a code point listed in the DUCET table using their implicit weight value, which is constructed according to the UCA.
- For non-language-specific collations, characters in contraction sequences are treated as separate characters. For language-specific collations, contractions might change character sorting order.

A collation name that includes a locale code or language name shown in the following table is a language-specific collation. Unicode character sets may include collations for one or more of these languages.

Table 10.3 Unicode Collation Language Specifiers

Language	Language Specifier
Chinese	`zh`
Classical Latin	`la` or `roman`
Croatian	`hr` or `croatian`
Czech	`cs` or `czech`
Danish	`da` or `danish`
Esperanto	`eo` or `esperanto`
Estonian	`et` or `estonian`
German phone book order	`de_pb` or `german2`
Hungarian	`hu` or `hungarian`
Icelandic	`is` or `icelandic`
Japanese	`ja`
Latvian	`lv` or `latvian`
Lithuanian	`lt` or `lithuanian`
Persian	`persian`
Polish	`pl` or `polish`
Romanian	`ro` or `romanian`
Russian	`ru`
Sinhala	`sinhala`
Slovak	`sk` or `slovak`
Slovenian	`sl` or `slovenian`
Modern Spanish	`es` or `spanish`
Traditional Spanish	`es_trad` or `spanish2`

Language	Language Specifier
Swedish	sv or swedish
Turkish	tr or turkish
Vietnamese	vi or vietnamese

Croatian collations are tailored for these Croatian letters: Č, Ć, Dž, Đ, Lj, Nj, Š, Ž.

Danish collations may also be used for Norwegian.

For Japanese, the utf8mb4 character set includes utf8mb4_ja_0900_as_cs and utf8mb4_ja_0900_as_cs_ks collations. Both collations are accent sensitive and case-sensitive. utf8mb4_ja_0900_as_cs_ks is also kana sensitive and distinguishes Katakana characters from Hiragana characters, whereas utf8mb4_ja_0900_as_cs treats Katakana and Hiragana characters as equal for sorting. Applications that require a Japanese collation but not kana sensitivity may use utf8mb4_ja_0900_as_cs for better sort performance. utf8mb4_ja_0900_as_cs uses three weight levels for sorting; utf8mb4_ja_0900_as_cs_ks uses four.

For Classical Latin collations that are accent insensitive, I and J compare as equal, and U and V compare as equal. I and J, and U and V compare as equal on the base letter level. In other words, J is regarded as an accented I, and U is regarded as an accented V.

Spanish collations are available for modern and traditional Spanish. For both, ñ (n-tilde) is a separate letter between n and o. In addition, for traditional Spanish, ch is a separate letter between c and d, and ll is a separate letter between l and m.

Traditional Spanish collations may also be used for Asturian and Galician.

Swedish collations include Swedish rules. For example, in Swedish, the following relationship holds, which is not something expected by a German or French speaker:

```
Ü = Y < Ö
```

_general_ci Versus _unicode_ci Collations

For any Unicode character set, operations performed using the *xxx*_general_ci collation are faster than those for the *xxx*_unicode_ci collation. For example, comparisons for the utf8_general_ci collation are faster, but slightly less correct, than comparisons for utf8_unicode_ci. The reason is that utf8_unicode_ci supports mappings such as expansions; that is, when one character compares as equal to combinations of other characters. For example, ß is equal to ss in German and some other languages. utf8_unicode_ci also supports contractions and ignorable characters. utf8_general_ci is a legacy collation that does not support expansions, contractions, or ignorable characters. It can make only one-to-one comparisons between characters.

To further illustrate, the following equalities hold in both utf8_general_ci and utf8_unicode_ci (for the effect of this in comparisons or searches, see Section 10.8.6, “Examples of the Effect of Collation”):

```
Ä = A
Ö = O
Ü = U
```

A difference between the collations is that this is true for utf8_general_ci:

```
ß = s
```

Whereas this is true for `utf8_unicode_ci`, which supports the German DIN-1 ordering (also known as dictionary order):

```
ß = ss
```

MySQL implements `utf8` language-specific collations if the ordering with `utf8_unicode_ci` does not work well for a language. For example, `utf8_unicode_ci` works fine for German dictionary order and French, so there is no need to create special `utf8` collations.

`utf8_general_ci` also is satisfactory for both German and French, except that ß is equal to `s`, and not to `ss`. If this is acceptable for your application, you should use `utf8_general_ci` because it is faster. If this is not acceptable (for example, if you require German dictionary order), use `utf8_unicode_ci` because it is more accurate.

If you require German DIN-2 (phone book) ordering, use the `utf8_german2_ci` collation, which compares the following sets of characters equal:

```
Ä = Æ = AE
Ö = Œ = OE
Ü = UE
ß = ss
```

`utf8_german2_ci` is similar to `latin1_german2_ci`, but the latter does not compare Æ equal to `AE` or Œ equal to `OE`. There is no `utf8_german_ci` corresponding to `latin1_german_ci` for German dictionary order because `utf8_general_ci` suffices.

Character Collating Weights

A character's collating weight is determined as follows:

- For all Unicode collations except the `_bin` (binary) collations, MySQL performs a table lookup to find a character's collating weight.
- For `_bin` collations except `utf8mb4_0900_bin`, the weight is based on the code point, possibly with leading zero bytes added.
- For `utf8mb4_0900_bin`, the weight is the `utf8mb4` encoding bytes. The sort order is the same as for `utf8mb4_bin`, but much faster.

Collating weights can be displayed using the `WEIGHT_STRING()` function. (See Section 12.5, “String Functions and Operators”.) If a collation uses a weight lookup table, but a character is not in the table (for example, because it is a “new” character), collating weight determination becomes more complex:

- For BMP characters in general collations (*`xxx`*`_general_ci`), the weight is the code point.
- For BMP characters in UCA collations (for example, *`xxx`*`_unicode_ci` and language-specific collations), the following algorithm applies:

```
if (code >= 0x3400 && code <= 0x4DB5)
  base= 0xFB80; /* CJK Ideograph Extension */
else if (code >= 0x4E00 && code <= 0x9FA5)
  base= 0xFB40; /* CJK Ideograph */
else
  base= 0xFBC0; /* All other characters */
aaaa= base +  (code >> 15);
bbbb= (code & 0x7FFF) | 0x8000;
```

The result is a sequence of two collating elements, `aaaa` followed by `bbbb`. For example:

```
mysql> SELECT HEX(WEIGHT_STRING(_ucs2 0x04CF COLLATE ucs2_unicode_ci));
+-----------------------------------------------------------+
| HEX(WEIGHT_STRING(_ucs2 0x04CF COLLATE ucs2_unicode_ci)) |
+-----------------------------------------------------------+
| FBC084CF                                                  |
+-----------------------------------------------------------+
```

Thus, `U+04cf CYRILLIC SMALL LETTER PALOCHKA` is, with all UCA 4.0.0 collations, greater than `U+04c0 CYRILLIC LETTER PALOCHKA`. With UCA 5.2.0 collations, all palochkas sort together.

- For supplementary characters in general collations, the weight is the weight for `0xfffd REPLACEMENT CHARACTER`. For supplementary characters in UCA 4.0.0 collations, their collating weight is `0xfffd`. That is, to MySQL, all supplementary characters are equal to each other, and greater than almost all BMP characters.

  An example with Deseret characters and `COUNT(DISTINCT)`:

  ```
  CREATE TABLE t (s1 VARCHAR(5) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
  INSERT INTO t VALUES (0xfffd);   /* REPLACEMENT CHARACTER */
  INSERT INTO t VALUES (0x010412); /* DESERET CAPITAL LETTER BEE */
  INSERT INTO t VALUES (0x010413); /* DESERET CAPITAL LETTER TEE */
  SELECT COUNT(DISTINCT s1) FROM t;
  ```

  The result is 2 because in the MySQL *`xxx`*`_unicode_ci` collations, the replacement character has a weight of `0x0dc6`, whereas Deseret Bee and Deseret Tee both have a weight of `0xfffd`. (Were the `utf32_general_ci` collation used instead, the result is 1 because all three characters have a weight of `0xfffd` in that collation.)

  An example with cuneiform characters and `WEIGHT_STRING()`:

  ```
  /*
  The four characters in the INSERT string are
  00000041  # LATIN CAPITAL LETTER A
  0001218F  # CUNEIFORM SIGN KAB
  000121A7  # CUNEIFORM SIGN KISH
  00000042  # LATIN CAPITAL LETTER B
  */
  CREATE TABLE t (s1 CHAR(4) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
  INSERT INTO t VALUES (0x00000410001218f000121a700000042);
  SELECT HEX(WEIGHT_STRING(s1)) FROM t;
  ```

  The result is:

  ```
  0E33 FFFD FFFD 0E4A
  ```

  `0E33` and `0E4A` are primary weights as in UCA 4.0.0. `FFFD` is the weight for KAB and also for KISH.

  The rule that all supplementary characters are equal to each other is nonoptimal but is not expected to cause trouble. These characters are very rare, so it is very rare that a multi-character string consists entirely of supplementary characters. In Japan, since the supplementary characters are obscure Kanji ideographs, the typical user does not care what order they are in, anyway. If you really want rows sorted by the MySQL rule and secondarily by code point value, it is easy:

  ```
  ORDER BY s1 COLLATE utf32_unicode_ci, s1 COLLATE utf32_bin
  ```

- For supplementary characters based on UCA versions higher than 4.0.0 (for example, *xxx*`_unicode_520_ci`), supplementary characters do not necessarily all have the same collating weight. Some have explicit weights from the UCA `allkeys.txt` file. Others have weights calculated from this algorithm:

```
aaaa= base +  (code >> 15);
bbbb= (code & 0x7FFF) | 0x8000;
```

There is a difference between “ordering by the character's code value” and “ordering by the character's binary representation,” a difference that appears only with `utf16_bin`, because of surrogates.

Suppose that `utf16_bin` (the binary collation for `utf16`) was a binary comparison “byte by byte” rather than “character by character.” If that were so, the order of characters in `utf16_bin` would differ from the order in `utf8_bin`. For example, the following chart shows two rare characters. The first character is in the range `E000`-`FFFF`, so it is greater than a surrogate but less than a supplementary. The second character is a supplementary.

```
Code point  Character                   utf8         utf16
----------  ---------                   ----         -----
0FF9D       HALFWIDTH KATAKANA LETTER N  EF BE 9D     FF 9D
10384       UGARITIC LETTER DELTA        F0 90 8E 84  D8 00 DF 84
```

The two characters in the chart are in order by code point value because `0xff9d` < `0x10384`. And they are in order by `utf8` value because `0xef` < `0xf0`. But they are not in order by `utf16` value, if we use byte-by-byte comparison, because `0xff` > `0xd8`.

So MySQL's `utf16_bin` collation is not “byte by byte.” It is “by code point.” When MySQL sees a supplementary-character encoding in `utf16`, it converts to the character's code-point value, and then compares. Therefore, `utf8_bin` and `utf16_bin` are the same ordering. This is consistent with the SQL:2008 standard requirement for a UCS_BASIC collation: “UCS_BASIC is a collation in which the ordering is determined entirely by the Unicode scalar values of the characters in the strings being sorted. It is applicable to the UCS character repertoire. Since every character repertoire is a subset of the UCS repertoire, the UCS_BASIC collation is potentially applicable to every character set. NOTE 11: The Unicode scalar value of a character is its code point treated as an unsigned integer.”

If the character set is `ucs2`, comparison is byte-by-byte, but `ucs2` strings should not contain surrogates, anyway.

Miscellaneous Information

The *xxx*`_general_mysql500_ci` collations preserve the pre-5.1.24 ordering of the original *xxx*`_general_ci` collations and permit upgrades for tables created before MySQL 5.1.24 (Bug #27877).

10.10.2 West European Character Sets

Western European character sets cover most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English.

- `ascii` (US ASCII) collations:
  - `ascii_bin`
  - `ascii_general_ci` (default)
- `cp850` (DOS West European) collations:

  - `cp850_bin`
  - `cp850_general_ci` (default)
- `dec8` (DEC Western European) collations:
  - `dec8_bin`
  - `dec8_swedish_ci` (default)
- `hp8` (HP Western European) collations:
  - `hp8_bin`
  - `hp8_english_ci` (default)
- `latin1` (cp1252 West European) collations:
  - `latin1_bin`
  - `latin1_danish_ci`
  - `latin1_general_ci`
  - `latin1_general_cs`
  - `latin1_german1_ci`
  - `latin1_german2_ci`
  - `latin1_spanish_ci`
  - `latin1_swedish_ci` (default)

  MySQL's `latin1` is the same as the Windows `cp1252` character set. This means it is the same as the official `ISO 8859-1` or IANA (Internet Assigned Numbers Authority) `latin1`, except that IANA `latin1` treats the code points between `0x80` and `0x9f` as "undefined," whereas `cp1252`, and therefore MySQL's `latin1`, assign characters for those positions. For example, `0x80` is the Euro sign. For the "undefined" entries in `cp1252`, MySQL translates `0x81` to Unicode `0x0081`, `0x8d` to `0x008d`, `0x8f` to `0x008f`, `0x90` to `0x0090`, and `0x9d` to `0x009d`.

  The `latin1_swedish_ci` collation is the default that probably is used by the majority of MySQL customers. Although it is frequently said that it is based on the Swedish/Finnish collation rules, there are Swedes and Finns who disagree with this statement.

  The `latin1_german1_ci` and `latin1_german2_ci` collations are based on the DIN-1 and DIN-2 standards, where DIN stands for *Deutsches Institut für Normung* (the German equivalent of ANSI). DIN-1 is called the "dictionary collation" and DIN-2 is called the "phone book collation." For an example of the effect this has in comparisons or when doing searches, see Section 10.8.6, "Examples of the Effect of Collation".

  - `latin1_german1_ci` (dictionary) rules:

```
Ä = A
Ö = O
Ü = U
ß = s
```

- `latin1_german2_ci` (phone-book) rules:

```
Ä = AE
Ö = OE
Ü = UE
ß = ss
```

  In the `latin1_spanish_ci` collation, `ñ` (n-tilde) is a separate letter between `n` and `o`.

- `macroman` (Mac West European) collations:
  - `macroman_bin`
  - `macroman_general_ci` (default)
- `swe7` (7bit Swedish) collations:
  - `swe7_bin`
  - `swe7_swedish_ci` (default)

10.10.3 Central European Character Sets

MySQL provides some support for character sets used in the Czech Republic, Slovakia, Hungary, Romania, Slovenia, Croatia, Poland, and Serbia (Latin).

- `cp1250` (Windows Central European) collations:
  - `cp1250_bin`
  - `cp1250_croatian_ci`
  - `cp1250_czech_cs`
  - `cp1250_general_ci` (default)
  - `cp1250_polish_ci`
- `cp852` (DOS Central European) collations:
  - `cp852_bin`
  - `cp852_general_ci` (default)
- `keybcs2` (DOS Kamenicky Czech-Slovak) collations:
  - `keybcs2_bin`
  - `keybcs2_general_ci` (default)
- `latin2` (ISO 8859-2 Central European) collations:
  - `latin2_bin`
  - `latin2_croatian_ci`
  - `latin2_czech_cs`

  - latin2_general_ci (default)
  - latin2_hungarian_ci
- macce (Mac Central European) collations:
  - macce_bin
  - macce_general_ci (default)

10.10.4 South European and Middle East Character Sets

South European and Middle Eastern character sets supported by MySQL include Armenian, Arabic, Georgian, Greek, Hebrew, and Turkish.

- armscii8 (ARMSCII-8 Armenian) collations:
  - armscii8_bin
  - armscii8_general_ci (default)
- cp1256 (Windows Arabic) collations:
  - cp1256_bin
  - cp1256_general_ci (default)
- geostd8 (GEOSTD8 Georgian) collations:
  - geostd8_bin
  - geostd8_general_ci (default)
- greek (ISO 8859-7 Greek) collations:
  - greek_bin
  - greek_general_ci (default)
- hebrew (ISO 8859-8 Hebrew) collations:
  - hebrew_bin
  - hebrew_general_ci (default)
- latin5 (ISO 8859-9 Turkish) collations:
  - latin5_bin
  - latin5_turkish_ci (default)

10.10.5 Baltic Character Sets

The Baltic character sets cover Estonian, Latvian, and Lithuanian languages.

- cp1257 (Windows Baltic) collations:
  - cp1257_bin

  - `cp1257_general_ci` (default)
  - `cp1257_lithuanian_ci`
- `latin7` (ISO 8859-13 Baltic) collations:
  - `latin7_bin`
  - `latin7_estonian_cs`
  - `latin7_general_ci` (default)
  - `latin7_general_cs`

10.10.6 Cyrillic Character Sets

The Cyrillic character sets and collations are for use with Belarusian, Bulgarian, Russian, Ukrainian, and Serbian (Cyrillic) languages.

- `cp1251` (Windows Cyrillic) collations:
  - `cp1251_bin`
  - `cp1251_bulgarian_ci`
  - `cp1251_general_ci` (default)
  - `cp1251_general_cs`
  - `cp1251_ukrainian_ci`
- `cp866` (DOS Russian) collations:
  - `cp866_bin`
  - `cp866_general_ci` (default)
- `koi8r` (KOI8-R Relcom Russian) collations:
  - `koi8r_bin`
  - `koi8r_general_ci` (default)
- `koi8u` (KOI8-U Ukrainian) collations:
  - `koi8u_bin`
  - `koi8u_general_ci` (default)

10.10.7 Asian Character Sets

The Asian character sets that we support include Chinese, Japanese, Korean, and Thai. These can be complicated. For example, the Chinese sets must allow for thousands of different characters. See Section 10.10.7.1, “The cp932 Character Set”, for additional information about the `cp932` and `sjis` character sets. See Section 10.10.7.2, “The gb18030 Character Set”, for additional information about character set support for the Chinese National Standard GB 18030.

For answers to some common questions and problems relating support for Asian character sets in MySQL, see Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”.

- `big5` (Big5 Traditional Chinese) collations:
  - `big5_bin`
  - `big5_chinese_ci` (default)
- `cp932` (SJIS for Windows Japanese) collations:
  - `cp932_bin`
  - `cp932_japanese_ci` (default)
- `eucjpms` (UJIS for Windows Japanese) collations:
  - `eucjpms_bin`
  - `eucjpms_japanese_ci` (default)
- `euckr` (EUC-KR Korean) collations:
  - `euckr_bin`
  - `euckr_korean_ci` (default)
- `gb2312` (GB2312 Simplified Chinese) collations:
  - `gb2312_bin`
  - `gb2312_chinese_ci` (default)
- `gbk` (GBK Simplified Chinese) collations:
  - `gbk_bin`
  - `gbk_chinese_ci` (default)
- `gb18030` (China National Standard GB18030) collations:
  - `gb18030_bin`
  - `gb18030_chinese_ci` (default)
  - `gb18030_unicode_520_ci`
- `sjis` (Shift-JIS Japanese) collations:
  - `sjis_bin`
  - `sjis_japanese_ci` (default)
- `tis620` (TIS620 Thai) collations:
  - `tis620_bin`
  - `tis620_thai_ci` (default)
- `ujis` (EUC-JP Japanese) collations:

- `ujis_bin`
- `ujis_japanese_ci` (default)

The `big5_chinese_ci` collation sorts on number of strokes.

10.10.7.1 The cp932 Character Set

Why is `cp932` needed?

In MySQL, the `sjis` character set corresponds to the `Shift_JIS` character set defined by IANA, which supports JIS X0201 and JIS X0208 characters. (See http://www.iana.org/assignments/character-sets.)

However, the meaning of “SHIFT JIS” as a descriptive term has become very vague and it often includes the extensions to `Shift_JIS` that are defined by various vendors.

For example, “SHIFT JIS” used in Japanese Windows environments is a Microsoft extension of `Shift_JIS` and its exact name is `Microsoft Windows Codepage : 932` or `cp932`. In addition to the characters supported by `Shift_JIS`, `cp932` supports extension characters such as NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.

Many Japanese users have experienced problems using these extension characters. These problems stem from the following factors:

- MySQL automatically converts character sets.
- Character sets are converted using Unicode (`ucs2`).
- The `sjis` character set does not support the conversion of these extension characters.
- There are several conversion rules from so-called “SHIFT JIS” to Unicode, and some characters are converted to Unicode differently depending on the conversion rule. MySQL supports only one of these rules (described later).

The MySQL `cp932` character set is designed to solve these problems.

Because MySQL supports character set conversion, it is important to separate IANA `Shift_JIS` and `cp932` into two different character sets because they provide different conversion rules.

How does `cp932` differ from `sjis`?

The `cp932` character set differs from `sjis` in the following ways:

- `cp932` supports NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.
- Some `cp932` characters have two different code points, both of which convert to the same Unicode code point. When converting from Unicode back to `cp932`, one of the code points must be selected. For this “round trip conversion,” the rule recommended by Microsoft is used. (See http://support.microsoft.com/kb/170559/EN-US/.)

  The conversion rule works like this:

  - If the character is in both JIS X 0208 and NEC special characters, use the code point of JIS X 0208.
  - If the character is in both NEC special characters and IBM selected characters, use the code point of NEC special characters.

- If the character is in both IBM selected characters and NEC selected—IBM extended characters, use the code point of IBM extended characters.

The table shown at https://msdn.microsoft.com/en-us/goglobal/cc305152.aspx provides information about the Unicode values of `cp932` characters. For `cp932` table entries with characters under which a four-digit number appears, the number represents the corresponding Unicode (`ucs2`) encoding. For table entries with an underlined two-digit value appears, there is a range of `cp932` character values that begin with those two digits. Clicking such a table entry takes you to a page that displays the Unicode value for each of the `cp932` characters that begin with those digits.

The following links are of special interest. They correspond to the encodings for the following sets of characters:

- NEC special characters (lead byte `0x87`):

```
https://msdn.microsoft.com/en-us/goglobal/gg674964
```

- NEC selected—IBM extended characters (lead byte `0xED` and `0xEE`):

```
https://msdn.microsoft.com/en-us/goglobal/gg671837
https://msdn.microsoft.com/en-us/goglobal/gg671838
```

- IBM selected characters (lead byte `0xFA`, `0xFB`, `0xFC`):

```
https://msdn.microsoft.com/en-us/goglobal/gg671839
https://msdn.microsoft.com/en-us/goglobal/gg671840
https://msdn.microsoft.com/en-us/goglobal/gg671841
```

- `cp932` supports conversion of user-defined characters in combination with `eucjpms`, and solves the problems with `sjis`/`ujis` conversion. For details, please refer to http://www.sljfaq.org/afaq/encodings.html.

For some characters, conversion to and from `ucs2` is different for `sjis` and `cp932`. The following tables illustrate these differences.

Conversion to `ucs2`:

`sjis`/`cp932` Value	**`sjis` -> `ucs2` Conversion**	**`cp932` -> `ucs2` Conversion**
5C	005C	005C
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225
817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1
81CA	00AC	FFE2

Conversion from `ucs2`:

ucs2 value	ucs2 -> sjis Conversion	ucs2 -> cp932 Conversion
005C	815F	5C
007E	7E	7E
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191
FFE1	3F	8192
FFE2	3F	81CA

Users of any Japanese character sets should be aware that using `--character-set-client-handshake` (or `--skip-character-set-client-handshake`) has an important effect. See Section 5.1.7, “Server Command Options”.

10.10.7.2 The gb18030 Character Set

In MySQL, the `gb18030` character set corresponds to the “Chinese National Standard GB 18030-2005: Information technology—Chinese coded character set”, which is the official character set of the People's Republic of China (PRC).

Characteristics of the MySQL gb18030 Character Set

- Supports all code points defined by the GB 18030-2005 standard. Unassigned code points in the ranges (GB+8431A439, GB+90308130) and (GB+E3329A36, GB+EF39EF39) are treated as '?' (0x3F). Conversion of unassigned code points return '?'.
- Supports UPPER and LOWER conversion for all GB18030 code points. Case folding defined by Unicode is also supported (based on `CaseFolding-6.3.0.txt`).
- Supports Conversion of data to and from other character sets.
- Supports SQL statements such as `SET NAMES`.
- Supports comparison between `gb18030` strings, and between `gb18030` strings and strings of other character sets. There is a conversion if strings have different character sets. Comparisons that include or ignore trailing spaces are also supported.
- The private use area (U+E000, U+F8FF) in Unicode is mapped to `gb18030`.
- There is no mapping between (U+D800, U+DFFF) and GB18030. Attempted conversion of code points in this range returns '?'.

- If an incoming sequence is illegal, an error or warning is returned. If an illegal sequence is used in `CONVERT()`, an error is returned. Otherwise, a warning is returned.
- For consistency with `utf8` and `utf8mb4`, UPPER is not supported for ligatures.
- Searches for ligatures also match uppercase ligatures when using the `gb18030_unicode_520_ci` collation.
- If a character has more than one uppercase character, the chosen uppercase character is the one whose lowercase is the character itself.
- The minimum multibyte length is 1 and the maximum is 4. The character set determines the length of a sequence using the first 1 or 2 bytes.

Supported Collations

- `gb18030_bin`: A binary collation.
- `gb18030_chinese_ci`: The default collation, which supports Pinyin. Sorting of non-Chinese characters is based on the order of the original sort key. The original sort key is `GB(UPPER(ch))` if `UPPER(ch)` exists. Otherwise, the original sort key is `GB(ch)`. Chinese characters are sorted according to the Pinyin collation defined in the Unicode Common Locale Data Repository (CLDR 24). Non-Chinese characters are sorted before Chinese characters with the exception of `GB+FE39FE39`, which is the code point maximum.
- `gb18030_unicode_520_ci`: A Unicode collation. Use this collation if you need to ensure that ligatures are sorted correctly.

10.10.8 The Binary Character Set

The `binary` character set is the chararcter set of binary strings, which are sequences of bytes. The `binary` character set has one collation, also named `binary`. Comparison and sorting are based on numeric byte values. The effect is that lettercase and accent differences are significant in comparisons. That is, the `binary` collation is case-sensitive and accent sensitive.

```
mysql> SET NAMES 'binary';
mysql> SELECT CHARSET('abc'), COLLATION('abc');
+----------------+------------------+
| CHARSET('abc') | COLLATION('abc') |
+----------------+------------------+
| binary         | binary           |
+----------------+------------------+
mysql> SELECT 'abc' = 'ABC', 'a' = 'ä';
+---------------+------------+
| 'abc' = 'ABC' | 'a' = 'ä'  |
+---------------+------------+
|             0 |          0 |
+---------------+------------+
```

For information about the differences between the `binary` collation of the `binary` character set and the `_bin` collations of nonbinary character sets, see Section 10.8.5, “The binary Collation Compared to _bin Collations”.

To convert a string expression to a binary string, any of these constructs are equivalent:

```
BINARY expr
CAST(expr AS BINARY)
CONVERT(expr USING BINARY)
```

If *expr* is a character string literal, the _binary introducer may be used to designate it as a binary string. For example:

```
_binary 'a'
```

The _binary introducer is permitted for hexadecimal literals and bit-value literals as well, but unnecessary; such literals are binary strings by default.

For more information about introducers, see Section 10.3.8, “Character Set Introducers”.

10.11 Restrictions on Character Sets

- Identifiers are stored in mysql database tables (user, db, and so forth) using utf8, but identifiers can contain only characters in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted in identifiers.
- The ucs2, utf16, utf16le, and utf32 character sets have the following restrictions:
  - None of them can be used as the client character set. See Impermissible Client Character Sets.
  - It is currently not possible to use LOAD DATA to load data files that use these character sets.
  - FULLTEXT indexes cannot be created on a column that uses any of these character sets. However, you can perform IN BOOLEAN MODE searches on the column without an index.
- The REGEXP and RLIKE operators work in byte-wise fashion, so they are not multibyte safe and may produce unexpected results with multibyte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

10.12 Setting the Error Message Language

By default, mysqld produces error messages in English, but they can be displayed instead in any of several other languages: Czech, Danish, Dutch, Estonian, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Norwegian-ny, Polish, Portuguese, Romanian, Russian, Slovak, Spanish, or Swedish. This applies to messages the server writes to the error log and sends to clients.

To select the language in which the server writes error messages, follow the instructions in this section. For information about changing the character set for error messages (rather than the language), see Section 10.6, “Error Message Character Set”. For general information about configuring error logging, see Section 5.4.2, “The Error Log”.

The server searches for the error message file using these rules:

- It looks for the file in a directory constructed from two system variable values, lc_messages_dir and lc_messages, with the latter converted to a language name. Suppose that you start the server using this command:

  ```
  mysqld --lc_messages_dir=/usr/share/mysql --lc_messages=fr_FR
  ```

  In this case, mysqld maps the locale fr_FR to the language french and looks for the error file in the /usr/share/mysql/french directory.

  By default, the language files are located in the share/mysql/*LANGUAGE* directory under the MySQL base directory.

- If the message file cannot be found in the directory constructed as just described, the server ignores the `lc_messages` value and uses only the `lc_messages_dir` value as the location in which to look.
- If the server cannot find the configured message file, it writes a message to the error log to indicate the problem and defaults to built-in English messages.

The `lc_messages_dir` system variable can be set only at server startup and has only a global read-only value at runtime. `lc_messages` can be set at server startup and has global and session values that can be modified at runtime. Thus, the error message language can be changed while the server is running, and each client can have its own error message language by setting its session `lc_messages` value to the desired locale name. For example, if the server is using the `fr_FR` locale for error messages, a client can execute this statement to receive error messages in English:

```
SET lc_messages = 'en_US';
```

10.13 Adding a Character Set

This section discusses the procedure for adding a character set to MySQL. The proper procedure depends on whether the character set is simple or complex:

- If the character set does not need special string collating routines for sorting and does not need multibyte character support, it is simple.
- If the character set needs either of those features, it is complex.

For example, `greek` and `swe7` are simple character sets, whereas `big5` and `czech` are complex character sets.

To use the following instructions, you must have a MySQL source distribution. In the instructions, *`MYSET`* represents the name of the character set that you want to add.

1. Add a `<charset>` element for *`MYSET`* to the `sql/share/charsets/Index.xml` file. Use the existing contents in the file as a guide to adding new contents. A partial listing for the `latin1` `<charset>` element follows:

   ```
   <charset name="latin1">
     <family>Western</family>
     <description>cp1252 West European</description>
     ...
     <collation name="latin1_swedish_ci" id="8" order="Finnish, Swedish">
       <flag>primary</flag>
       <flag>compiled</flag>
     </collation>
     <collation name="latin1_danish_ci" id="15" order="Danish"/>
     ...
     <collation name="latin1_bin" id="47" order="Binary">
       <flag>binary</flag>
       <flag>compiled</flag>
     </collation>
     ...
   </charset>
   ```

   The `<charset>` element must list all the collations for the character set. These must include at least a binary collation and a default (primary) collation. The default collation is often named using a suffix of `general_ci` (general, case insensitive). It is possible for the binary collation to be the default collation, but usually they are different. The default collation should have a `primary` flag. The binary collation should have a `binary` flag.

You must assign a unique ID number to each collation. The range of IDs from 1024 to 2047 is reserved for user-defined collations. To find the maximum of the currently used collation IDs, use this query:

```
SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
```

2. This step depends on whether you are adding a simple or complex character set. A simple character set requires only a configuration file, whereas a complex character set requires C source file that defines collation functions, multibyte functions, or both.

   For a simple character set, create a configuration file, *MYSET*.xml, that describes the character set properties. Create this file in the sql/share/charsets directory. You can use a copy of latin1.xml as the basis for this file. The syntax for the file is very simple:

   - Comments are written as ordinary XML comments (<!-- *text* -->).
   - Words within <map> array elements are separated by arbitrary amounts of whitespace.
   - Each word within <map> array elements must be a number in hexadecimal format.
   - The <map> array element for the <ctype> element has 257 words. The other <map> array elements after that have 256 words. See Section 10.13.1, “Character Definition Arrays”.
   - For each collation listed in the <charset> element for the character set in Index.xml, *MYSET*.xml must contain a <collation> element that defines the character ordering.

   For a complex character set, create a C source file that describes the character set properties and defines the support routines necessary to properly perform operations on the character set:

   - Create the file ctype-*MYSET*.c in the strings directory. Look at one of the existing ctype-*.c files (such as ctype-big5.c) to see what needs to be defined. The arrays in your file must have names like ctype_*MYSET*, to_lower_*MYSET*, and so on. These correspond to the arrays for a simple character set. See Section 10.13.1, “Character Definition Arrays”.
   - For each <collation> element listed in the <charset> element for the character set in Index.xml, the ctype-*MYSET*.c file must provide an implementation of the collation.
   - If the character set requires string collating functions, see Section 10.13.2, “String Collating Support for Complex Character Sets”.
   - If the character set requires multibyte character support, see Section 10.13.3, “Multi-Byte Character Support for Complex Character Sets”.

3. Modify the configuration information. Use the existing configuration information as a guide to adding information for *MYSYS*. The example here assumes that the character set has default and binary collations, but more lines are needed if *MYSET* has additional collations.

   a. Edit mysys/charset-def.c, and “register” the collations for the new character set.

      Add these lines to the “declaration” section:

      ```
      #ifdef HAVE_CHARSET_MYSET
      extern CHARSET_INFO my_charset_MYSET_general_ci;
      extern CHARSET_INFO my_charset_MYSET_bin;
      #endif
      ```

Add these lines to the “registration” section:

```
#ifdef HAVE_CHARSET_MYSET
  add_compiled_collation(&my_charset_MYSET_general_ci);
  add_compiled_collation(&my_charset_MYSET_bin);
#endif
```

b. If the character set uses `ctype-MYSET.c`, edit `strings/CMakeLists.txt` and add `ctype-MYSET.c` to the definition of the `STRINGS_SOURCES` variable.

c. Edit `cmake/character_sets.cmake`:

   i. Add *`MYSET`* to the value of with `CHARSETS_AVAILABLE` in alphabetic order.

   ii. Add *`MYSET`* to the value of `CHARSETS_COMPLEX` in alphabetic order. This is needed even for simple character sets, or `CMake` will not recognize `-DDEFAULT_CHARSET=MYSET`.

4. Reconfigure, recompile, and test.

10.13.1 Character Definition Arrays

Each simple character set has a configuration file located in the `sql/share/charsets` directory. For a character set named *`MYSYS`*, the file is named *`MYSET`*`.xml`. It uses `<map>` array elements to list character set properties. `<map>` elements appear within these elements:

- `<ctype>` defines attributes for each character.
- `<lower>` and `<upper>` list the lowercase and uppercase characters.
- `<unicode>` maps 8-bit character values to Unicode values.
- `<collation>` elements indicate character ordering for comparison and sorting, one element per collation. Binary collations need no `<map>` element because the character codes themselves provide the ordering.

For a complex character set as implemented in a `ctype-MYSET.c` file in the `strings` directory, there are corresponding arrays: `ctype_MYSET[]`, `to_lower_MYSET[]`, and so forth. Not every complex character set has all of the arrays. See also the existing `ctype-*.c` files for examples. See the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

Most of the arrays are indexed by character value and have 256 elements. The `<ctype>` array is indexed by character value + 1 and has 257 elements. This is a legacy convention for handling `EOF`.

`<ctype>` array elements are bit values. Each element describes the attributes of a single character in the character set. Each attribute is associated with a bitmask, as defined in `include/m_ctype.h`:

```
#define _MY_U   01      /* Upper case */
#define _MY_L   02      /* Lower case */
#define _MY_NMR 04      /* Numeral (digit) */
#define _MY_SPC 010     /* Spacing character */
#define _MY_PNT 020     /* Punctuation */
#define _MY_CTR 040     /* Control character */
#define _MY_B   0100    /* Blank */
#define _MY_X   0200    /* heXadecimal digit */
```

The `<ctype>` value for a given character should be the union of the applicable bitmask values that describe the character. For example, `'A'` is an uppercase character (`_MY_U`) as well as a hexadecimal digit (`_MY_X`), so its `ctype` value should be defined like this:

```
ctype['A'+1] = _MY_U | _MY_X = 01 | 0200 = 0201
```

The bitmask values in `m_ctype.h` are octal values, but the elements of the `<ctype>` array in *`MYSET`*`.xml` should be written as hexadecimal values.

The `<lower>` and `<upper>` arrays hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

```
lower['A'] should contain 'a'
upper['a'] should contain 'A'
```

Each `<collation>` array indicates how characters should be ordered for comparison and sorting purposes. MySQL sorts characters based on the values of this information. In some cases, this is the same as the `<upper>` array, which means that sorting is case-insensitive. For more complicated sorting rules (for complex character sets), see the discussion of string collating in Section 10.13.2, “String Collating Support for Complex Character Sets”.

10.13.2 String Collating Support for Complex Character Sets

For a simple character set named *`MYSET`*, sorting rules are specified in the *`MYSET`*`.xml` configuration file using `<map>` array elements within `<collation>` elements. If the sorting rules for your language are too complex to be handled with simple arrays, you must define string collating functions in the `ctype-`*`MYSET`*`.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `big5`, `czech`, `gbk`, `sjis`, and `tis160` character sets. Take a look at the `MY_COLLATION_HANDLER` structures to see how they are used. See also the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

10.13.3 Multi-Byte Character Support for Complex Character Sets

If you want to add support for a new character set named *`MYSET`* that includes multibyte characters, you must use multibyte character functions in the `ctype-`*`MYSET`*`.c` source file in the `strings` directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the `ctype-*.c` files in the `strings` directory, such as the files for the `euc_kr`, `gb2312`, `gbk`, `sjis`, and `ujis` character sets. Take a look at the `MY_CHARSET_HANDLER` structures to see how they are used. See also the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

10.14 Adding a Collation to a Character Set

A collation is a set of rules that defines how to compare and sort character strings. Each collation in MySQL belongs to a single character set. Every character set has at least one collation, and most have two or more collations.

A collation orders characters based on weights. Each character in a character set maps to a weight. Characters with equal weights compare as equal, and characters with unequal weights compare according to the relative magnitude of their weights.

The `WEIGHT_STRING()` function can be used to see the weights for the characters in a string. The value that it returns to indicate weights is a binary string, so it is convenient to use `HEX(WEIGHT_STRING(`*`str`*`))` to display the weights in printable form. The following example shows that

weights do not differ for lettercase for the letters in `'AaBb'` if it is a nonbinary case-insensitive string, but do differ if it is a binary string:

```
mysql> SELECT HEX(WEIGHT_STRING('AaBb' COLLATE latin1_swedish_ci));
+-----------------------------------------------------+
| HEX(WEIGHT_STRING('AaBb' COLLATE latin1_swedish_ci)) |
+-----------------------------------------------------+
| 41414242                                            |
+-----------------------------------------------------+
mysql> SELECT HEX(WEIGHT_STRING(BINARY 'AaBb'));
+----------------------------------+
| HEX(WEIGHT_STRING(BINARY 'AaBb')) |
+----------------------------------+
| 41614262                         |
+----------------------------------+
```

MySQL supports several collation implementations, as discussed in Section 10.14.1, “Collation Implementation Types”. Some of these can be added to MySQL without recompiling:

- Simple collations for 8-bit character sets.
- UCA-based collations for Unicode character sets.
- Binary (*`xxx`*`_bin`) collations.

The following sections describe how to add collations of the first two types to existing character sets. All existing character sets already have a binary collation, so there is no need here to describe how to add one.

Summary of the procedure for adding a new collation:

1. Choose a collation ID.
2. Add configuration information that names the collation and describes the character-ordering rules.
3. Restart the server.
4. Verify that the collation is present.

The instructions here cover only collations that can be added without recompiling MySQL. To add a collation that does require recompiling (as implemented by means of functions in a C source file), use the instructions in Section 10.13, “Adding a Character Set”. However, instead of adding all the information required for a complete character set, just modify the appropriate files for an existing character set. That is, based on what is already present for the character set's current collations, add data structures, functions, and configuration information for the new collation.

Note

If you modify an existing collation, that may affect the ordering of rows for indexes on columns that use the collation. In this case, rebuild any such indexes to avoid problems such as incorrect query results. See Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”.

Additional Resources

- The Unicode Collation Algorithm (UCA) specification: http://www.unicode.org/reports/tr10/
- The Locale Data Markup Language (LDML) specification: http://www.unicode.org/reports/tr35/

10.14.1 Collation Implementation Types

MySQL implements several types of collations:

Simple collations for 8-bit character sets

This kind of collation is implemented using an array of 256 weights that defines a one-to-one mapping from character codes to weights. `latin1_swedish_ci` is an example. It is a case-insensitive collation, so the uppercase and lowercase versions of a character have the same weights and they compare as equal.

```
mysql> SET NAMES 'latin1' COLLATE 'latin1_swedish_ci';
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX(WEIGHT_STRING('a')), HEX(WEIGHT_STRING('A'));
+-------------------------+-------------------------+
| HEX(WEIGHT_STRING('a')) | HEX(WEIGHT_STRING('A')) |
+-------------------------+-------------------------+
| 41                      | 41                      |
+-------------------------+-------------------------+
1 row in set (0.01 sec)

mysql> SELECT 'a' = 'A';
+-----------+
| 'a' = 'A' |
+-----------+
|         1 |
+-----------+
1 row in set (0.12 sec)
```

For implementation instructions, see Section 10.14.3, “Adding a Simple Collation to an 8-Bit Character Set”.

Complex collations for 8-bit character sets

This kind of collation is implemented using functions in a C source file that define how to order characters, as described in Section 10.13, “Adding a Character Set”.

Collations for non-Unicode multibyte character sets

For this type of collation, 8-bit (single-byte) and multibyte characters are handled differently. For 8-bit characters, character codes map to weights in case-insensitive fashion. (For example, the single-byte characters `'a'` and `'A'` both have a weight of `0x41`.) For multibyte characters, there are two types of relationship between character codes and weights:

- Weights equal character codes. `sjis_japanese_ci` is an example of this kind of collation. The multibyte character `'ぢ'` has a character code of `0x82C0`, and the weight is also `0x82C0`.

```
mysql> CREATE TABLE t1
       (c1 VARCHAR(2) CHARACTER SET sjis COLLATE sjis_japanese_ci);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t1 VALUES ('a'),('A'),(0x82C0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+-----------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+------+---------+-----------------------+
| a    | 61      | 41                    |
```

```
| A    | 41      | 41                    |
| ぢ    | 82C0    | 82C0                  |
+------+---------+-----------------------+
3 rows in set (0.00 sec)
```

- Character codes map one-to-one to weights, but a code is not necessarily equal to the weight. `gbk_chinese_ci` is an example of this kind of collation. The multibyte character `'膰'` has a character code of `0x81B0` but a weight of `0xC286`.

```
mysql> CREATE TABLE t1
       (c1 VARCHAR(2) CHARACTER SET gbk COLLATE gbk_chinese_ci);
Query OK, 0 rows affected (0.33 sec)

mysql> INSERT INTO t1 VALUES ('a'),('A'),(0x81B0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+-----------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+------+---------+-----------------------+
| a    | 61      | 41                    |
| A    | 41      | 41                    |
| 膰    | 81B0    | C286                  |
+------+---------+-----------------------+
3 rows in set (0.00 sec)
```

For implementation instructions, see Section 10.13, “Adding a Character Set”.

Collations for Unicode multibyte character sets

Some of these collations are based on the Unicode Collation Algorithm (UCA), others are not.

Non-UCA collations have a one-to-one mapping from character code to weight. In MySQL, such collations are case insensitive and accent insensitive. `utf8_general_ci` is an example: `'a'`, `'A'`, `'À'`, and `'á'` each have different character codes but all have a weight of `0x0041` and compare as equal.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_general_ci';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1
       (c1 CHAR(1) CHARACTER SET UTF8 COLLATE utf8_general_ci);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t1 VALUES ('a'),('A'),('À'),('á');
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+-----------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+------+---------+-----------------------+
| a    | 61      | 0041                  |
| A    | 41      | 0041                  |
| À    | C380    | 0041                  |
| á    | C3A1    | 0041                  |
+------+---------+-----------------------+
4 rows in set (0.00 sec)
```

UCA-based collations in MySQL have these properties:

- If a character has weights, each weight uses 2 bytes (16 bits).

- A character may have zero weights (or an empty weight). In this case, the character is ignorable. Example: "U+0000 NULL" does not have a weight and is ignorable.
- A character may have one weight. Example: `'a'` has a weight of `0x0E33`.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_unicode_ci';
Query OK, 0 rows affected (0.05 sec)

mysql> SELECT HEX('a'), HEX(WEIGHT_STRING('a'));
+----------+------------------------+
| HEX('a') | HEX(WEIGHT_STRING('a')) |
+----------+------------------------+
| 61       | 0E33                   |
+----------+------------------------+
1 row in set (0.02 sec)
```

- A character may have many weights. This is an expansion. Example: The German letter `'ß'` (SZ ligature, or SHARP S) has a weight of `0x0FEA0FEA`.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_unicode_ci';
Query OK, 0 rows affected (0.11 sec)

mysql> SELECT HEX('ß'), HEX(WEIGHT_STRING('ß'));
+-----------+-------------------------+
| HEX('ß')  | HEX(WEIGHT_STRING('ß'))  |
+-----------+-------------------------+
| C39F      | 0FEA0FEA                |
+-----------+-------------------------+
1 row in set (0.00 sec)
```

- Many characters may have one weight. This is a contraction. Example: `'ch'` is a single letter in Czech and has a weight of `0x0EE2`.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_czech_ci';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT HEX('ch'), HEX(WEIGHT_STRING('ch'));
+-----------+-------------------------+
| HEX('ch') | HEX(WEIGHT_STRING('ch')) |
+-----------+-------------------------+
| 6368      | 0EE2                    |
+-----------+-------------------------+
1 row in set (0.00 sec)
```

A many-characters-to-many-weights mapping is also possible (this is contraction with expansion), but is not supported by MySQL.

For implementation instructions, for a non-UCA collation, see Section 10.13, “Adding a Character Set”. For a UCA collation, see Section 10.14.4, “Adding a UCA Collation to a Unicode Character Set”.

Miscellaneous collations

There are also a few collations that do not fall into any of the previous categories.

10.14.2 Choosing a Collation ID

Each collation must have a unique ID. To add a collation, you must choose an ID value that is not currently used. MySQL supports two-byte collation IDs. The range of IDs from 1024 to 2047 is reserved for user-defined collations. The collation ID that you choose will appear in these contexts:

- The `ID` column of the `INFORMATION_SCHEMA.COLLATIONS` table.

- The `Id` column of `SHOW COLLATION` output.
- The `charsetnr` member of the `MYSQL_FIELD` C API data structure.
- The `number` member of the `MY_CHARSET_INFO` data structure returned by the `mysql_get_character_set_info()` C API function.

To determine the largest currently used ID, issue the following statement:

```
mysql> SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
+---------+
| MAX(ID) |
+---------+
|     210 |
+---------+
```

To display a list of all currently used IDs, issue this statement:

```
mysql> SELECT ID FROM INFORMATION_SCHEMA.COLLATIONS ORDER BY ID;
+-----+
| ID  |
+-----+
|   1 |
|   2 |
| ... |
|  52 |
|  53 |
|  57 |
|  58 |
| ... |
|  98 |
|  99 |
| 128 |
| 129 |
| ... |
| 210 |
+-----+
```

Warning

Before upgrading, you should save the configuration files that you change. If you upgrade in place, the process will replace the your modified files.

10.14.3 Adding a Simple Collation to an 8-Bit Character Set

This section describes how to add a simple collation for an 8-bit character set by writing the `<collation>` elements associated with a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. The example adds a collation named `latin1_test_ci` to the `latin1` character set.

1. Choose a collation ID, as shown in Section 10.14.2, “Choosing a Collation ID”. The following steps use an ID of 1024.
2. Modify the `Index.xml` and `latin1.xml` configuration files. These files are located in the directory named by the `character_sets_dir` system variable. You can check the variable value as follows, although the path name might be different on your system:

```
mysql> SHOW VARIABLES LIKE 'character_sets_dir';
+--------------------+----------------------------------------+
```

```
| Variable_name      | Value                                    |
+--------------------+------------------------------------------+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+--------------------+------------------------------------------+
```

3. Choose a name for the collation and list it in the `Index.xml` file. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. For example:

```
<charset name="latin1">
  ...
  <collation name="latin1_test_ci" id="1024"/>
  ...
</charset>
```

4. In the `latin1.xml` configuration file, add a `<collation>` element that names the collation and that contains a `<map>` element that defines a character code-to-weight mapping table for character codes 0 to 255. Each value within the `<map>` element must be a number in hexadecimal format.

```
<collation name="latin1_test_ci">
<map>
 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
 60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
 50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
 41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
 44 4E 4F 4F 4F 4F 5C D7 5C 55 55 55 59 59 DE DF
 41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
 44 4E 4F 4F 4F 4F 5C F7 5C 55 55 55 59 59 DE FF
</map>
</collation>
```

5. Restart the server and use this statement to verify that the collation is present:

```
mysql> SHOW COLLATION WHERE Collation = 'latin1_test_ci';
+---------------+---------+------+---------+----------+---------+
| Collation     | Charset | Id   | Default | Compiled | Sortlen |
+---------------+---------+------+---------+----------+---------+
| latin1_test_ci | latin1  | 1024 |         |          |       1 |
+---------------+---------+------+---------+----------+---------+
```

10.14.4 Adding a UCA Collation to a Unicode Character Set

This section describes how to add a UCA collation for a Unicode character set by writing the `<collation>` element within a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. It uses a subset of the Locale Data Markup Language (LDML) specification, which is available at http://www.unicode.org/reports/tr35/. With this method, you need not define the entire collation. Instead, you begin with an existing “base” collation and describe the new collation in terms of how it differs from the base collation. The following table lists the base collations of the Unicode character sets for which UCA collations can be defined. It is not possible to create user-defined UCA collations for `utf16le`; there is no `utf16le_unicode_ci` collation that would serve as the basis for such collations.

Table 10.4 MySQL Character Sets Available for User-Defined UCA Collations

Character Set	Base Collation
utf8	utf8_unicode_ci
ucs2	ucs2_unicode_ci
utf16	utf16_unicode_ci
utf32	utf32_unicode_ci

The following sections show how to add a collation that is defined using LDML syntax, and provide a summary of LDML rules supported in MySQL.

10.14.4.1 Defining a UCA Collation Using LDML Syntax

To add a UCA collation for a Unicode character set without recompiling MySQL, use the following procedure. If you are unfamiliar with the LDML rules used to describe the collation's sort characteristics, see Section 10.14.4.2, “LDML Syntax Supported in MySQL”.

The example adds a collation named `utf8_phone_ci` to the `utf8` character set. The collation is designed for a scenario involving a Web application for which users post their names and phone numbers. Phone numbers can be given in very different formats:

```
+7-12345-67
+7-12-345-67
+7 12 345 67
+7 (12) 345 67
+71234567
```

The problem raised by dealing with these kinds of values is that the varying permissible formats make searching for a specific phone number very difficult. The solution is to define a new collation that reorders punctuation characters, making them ignorable.

1. Choose a collation ID, as shown in Section 10.14.2, “Choosing a Collation ID”. The following steps use an ID of 1029.

2. To modify the `Index.xml` configuration file. This file is located in the directory named by the `character_sets_dir` system variable. You can check the variable value as follows, although the path name might be different on your system:

   ```
   mysql> SHOW VARIABLES LIKE 'character_sets_dir';
   +--------------------+----------------------------------------+
   | Variable_name      | Value                                  |
   +--------------------+----------------------------------------+
   | character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
   +--------------------+----------------------------------------+
   ```

3. Choose a name for the collation and list it in the `Index.xml` file. In addition, you'll need to provide the collation ordering rules. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. Within the `<collation>` element, provide a `<rules>` element containing the ordering rules:

   ```
   <charset name="utf8">
     ...
     <collation name="utf8_phone_ci" id="1029">
       <rules>
         <reset>\u0000</reset>
   ```

```
      <i>\u0020</i> <!-- space -->
      <i>\u0028</i> <!-- left parenthesis -->
      <i>\u0029</i> <!-- right parenthesis -->
      <i>\u002B</i> <!-- plus -->
      <i>\u002D</i> <!-- hyphen -->
    </rules>
  </collation>
  ...
</charset>
```

4. If you want a similar collation for other Unicode character sets, add other `<collation>` elements. For example, to define `ucs2_phone_ci`, add a `<collation>` element to the `<charset name="ucs2">` element. Remember that each collation must have its own unique ID.

5. Restart the server and use this statement to verify that the collation is present:

```
mysql> SHOW COLLATION WHERE Collation = 'utf8_phone_ci';
+---------------+---------+------+---------+----------+---------+
| Collation     | Charset | Id   | Default | Compiled | Sortlen |
+---------------+---------+------+---------+----------+---------+
| utf8_phone_ci | utf8    | 1029 |         |          |       8 |
+---------------+---------+------+---------+----------+---------+
```

Now test the collation to make sure that it has the desired properties.

Create a table containing some sample phone numbers using the new collation:

```
mysql> CREATE TABLE phonebook (
         name VARCHAR(64),
         phone VARCHAR(64) CHARACTER SET utf8 COLLATE utf8_phone_ci
       );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO phonebook VALUES ('Svoj','+7 912 800 80 02');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Hf','+7 (912) 800 80 04');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Bar','+7-912-800-80-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Ramil','(7912) 800 80 03');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Sanja','+380 (912) 8008005');
Query OK, 1 row affected (0.00 sec)
```

Run some queries to see whether the ignored punctuation characters are in fact ignored for comparison and sorting:

```
mysql> SELECT * FROM phonebook ORDER BY phone;
+-------+--------------------+
| name  | phone              |
+-------+--------------------+
| Sanja | +380 (912) 8008005 |
| Bar   | +7-912-800-80-01   |
| Svoj  | +7 912 800 80 02   |
| Ramil | (7912) 800 80 03   |
| Hf    | +7 (912) 800 80 04 |
+-------+--------------------+
5 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM phonebook WHERE phone='+7(912)800-80-01';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='79128008001';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='7 9 1 2 8 0 0 8 0 0 1';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)
```

10.14.4.2 LDML Syntax Supported in MySQL

This section describes the LDML syntax that MySQL recognizes. This is a subset of the syntax described in the LDML specification available at http://www.unicode.org/reports/tr35/, which should be consulted for further information. MySQL recognizes a large enough subset of the syntax that, in many cases, it is possible to download a collation definition from the Unicode Common Locale Data Repository and paste the relevant part (that is, the part between the `<rules>` and `</rules>` tags) into the MySQL `Index.xml` file. The rules described here are all supported except that character sorting occurs only at the primary level. Rules that specify differences at secondary or higher sort levels are recognized (and thus can be included in collation definitions) but are treated as equality at the primary level.

The MySQL server generates diagnostics when it finds problems while parsing the `Index.xml` file. See Section 10.14.4.3, “Diagnostics During Index.xml Parsing”.

Character Representation

Characters named in LDML rules can be written literally or in `\u`*nnnn* format, where *nnnn* is the hexadecimal Unicode code point value. For example, `A` and `á` can be written literally or as `\u0041` and `\u00E1`. Within hexadecimal values, the digits `A` through `F` are not case-sensitive; `\u00E1` and `\u00e1` are equivalent. For UCA 4.0.0 collations, hexadecimal notation can be used only for characters in the Basic Multilingual Plane, not for characters outside the BMP range of `0000` to `FFFF`. For UCA 5.2.0 collations, hexadecimal notation can be used for any character.

The `Index.xml` file itself should be written using UTF-8 encoding.

Syntax Rules

LDML has reset rules and shift rules to specify character ordering. Orderings are given as a set of rules that begin with a reset rule that establishes an anchor point, followed by shift rules that indicate how characters sort relative to the anchor point.

- A `<reset>` rule does not specify any ordering in and of itself. Instead, it “resets” the ordering for subsequent shift rules to cause them to be taken in relation to a given character. Either of the following rules resets subsequent shift rules to be taken in relation to the letter `'A'`:

```
<reset>A</reset>

<reset>\u0041</reset>
```

- The `<p>`, `<s>`, and `<t>` shift rules define primary, secondary, and tertiary differences of a character from another character:

  - Use primary differences to distinguish separate letters.
  - Use secondary differences to distinguish accent variations.
  - Use tertiary differences to distinguish lettercase variations.

  Either of these rules specifies a primary shift rule for the `'G'` character:

```
<p>G</p>

<p>\u0047</p>
```

- The `<i>` shift rule indicates that one character sorts identically to another. The following rules cause `'b'` to sort the same as `'a'`:

```
<reset>a</reset>
<i>b</i>
```

- Abbreviated shift syntax specifies multiple shift rules using a single pair of tags. The following table shows the correspondence between abbreviated syntax rules and the equivalent nonabbreviated rules.

  Table 10.5 Abbreviated Shift Syntax

Abbreviated Syntax	Nonabbreviated Syntax
`<pc>xyz</pc>`	`<p>x</p><p>y</p><p>z</p>`
`<sc>xyz</sc>`	`<s>x</s><s>y</s><s>z</s>`
`<tc>xyz</tc>`	`<t>x</t><t>y</t><t>z</t>`
`<ic>xyz</ic>`	`<i>x</i><i>y</i><i>z</i>`

- An expansion is a reset rule that establishes an anchor point for a multiple-character sequence. MySQL supports expansions 2 to 6 characters long. The following rules put `'z'` greater at the primary level than the sequence of three characters `'abc'`:

```
<reset>abc</reset>
<p>z</p>
```

- A contraction is a shift rule that sorts a multiple-character sequence. MySQL supports contractions 2 to 6 characters long. The following rules put the sequence of three characters `'xyz'` greater at the primary level than `'a'`:

```
<reset>a</reset>
<p>xyz</p>
```

- Long expansions and long contractions can be used together. These rules put the sequence of three characters `'xyz'` greater at the primary level than the sequence of three characters `'abc'`:

```
<reset>abc</reset>
```

```
<p>xyz</p>
```

- Normal expansion syntax uses `<x>` plus `<extend>` elements to specify an expansion. The following rules put the character `'k'` greater at the secondary level than the sequence `'ch'`. That is, `'k'` behaves as if it expands to a character after `'c'` followed by `'h'`:

  ```
  <reset>c</reset>
  <x><s>k</s><extend>h</extend></x>
  ```

  This syntax permits long sequences. These rules sort the sequence `'ccs'` greater at the tertiary level than the sequence `'cscs'`:

  ```
  <reset>cs</reset>
  <x><t>ccs</t><extend>cs</extend></x>
  ```

  The LDML specification describes normal expansion syntax as “tricky.” See that specification for details.

- Previous context syntax uses `<x>` plus `<context>` elements to specify that the context before a character affects how it sorts. The following rules put `'-'` greater at the secondary level than `'a'`, but only when `'-'` occurs after `'b'`:

  ```
  <reset>a</reset>
  <x><context>b</context><s>-</s></x>
  ```

- Previous context syntax can include the `<extend>` element. These rules put `'def'` greater at the primary level than `'aghi'`, but only when `'def'` comes after `'abc'`:

  ```
  <reset>a</reset>
  <x><context>abc</context><p>def</p><extend>ghi</extend></x>
  ```

- Reset rules permit a `before` attribute. Normally, shift rules after a reset rule indicate characters that sort after the reset character. Shift rules after a reset rule that has the `before` attribute indicate characters that sort before the reset character. The following rules put the character `'b'` immediately before `'a'` at the primary level:

  ```
  <reset before="primary">a</reset>
  <p>b</p>
  ```

  Permissible `before` attribute values specify the sort level by name or the equivalent numeric value:

  ```
  <reset before="primary">
  <reset before="1">

  <reset before="secondary">
  <reset before="2">

  <reset before="tertiary">
  <reset before="3">
  ```

- A reset rule can name a logical reset position rather than a literal character:

  ```
  <first_tertiary_ignorable/>
  <last_tertiary_ignorable/>
  <first_secondary_ignorable/>
  <last_secondary_ignorable/>
  <first_primary_ignorable/>
  ```

```
<last_primary_ignorable/>
<first_variable/>
<last_variable/>
<first_non_ignorable/>
<last_non_ignorable/>
<first_trailing/>
<last_trailing/>
```

These rules put `'z'` greater at the primary level than nonignorable characters that have a Default Unicode Collation Element Table (DUCET) entry and that are not CJK:

```
<reset><last_non_ignorable/></reset>
<p>z</p>
```

Logical positions have the code points shown in the following table.

Table 10.6 Logical Reset Position Code Points

Logical Position	Unicode 4.0.0 Code Point	Unicode 5.2.0 Code Point
`<first_non_ignorable/>`	U+02D0	U+02D0
`<last_non_ignorable/>`	U+A48C	U+1342E
`<first_primary_ignorable/>`	U+0332	U+0332
`<last_primary_ignorable/>`	U+20EA	U+101FD
`<first_secondary_ignorable/>`	U+0000	U+0000
`<last_secondary_ignorable/>`	U+FE73	U+FE73
`<first_tertiary_ignorable/>`	U+0000	U+0000
`<last_tertiary_ignorable/>`	U+FE73	U+FE73
`<first_trailing/>`	U+0000	U+0000
`<last_trailing/>`	U+0000	U+0000
`<first_variable/>`	U+0009	U+0009
`<last_variable/>`	U+2183	U+1D371

- The `<collation>` element permits a `shift-after-method` attribute that affects character weight calculation for shift rules. The attribute has these permitted values:

  - `simple`: Calculate character weights as for reset rules that do not have a `before` attribute. This is the default if the attribute is not given.

  - `expand`: Use expansions for shifts after reset rules.

  Suppose that `'0'` and `'1'` have weights of `0E29` and `0E2A` and we want to put all basic Latin letters between `'0'` and `'1'`:

```
<reset>0</reset>
<pc>abcdefghijklmnopqrstuvwxyz</pc>
```

For simple shift mode, weights are calculated as follows:

```
'a' has weight 0E29+1
'b' has weight 0E29+2
'c' has weight 0E29+3
...
```

However, there are not enough vacant positions to put 26 characters between `'0'` and `'1'`. The result is that digits and letters are intermixed.

To solve this, use `shift-after-method="expand"`. Then weights are calculated like this:

```
'a' has weight [0E29][233D+1]
'b' has weight [0E29][233D+2]
'c' has weight [0E29][233D+3]
...
```

`233D` is the UCA 4.0.0 weight for character `0xA48C`, which is the last nonignorable character (a sort of the greatest character in the collation, excluding CJK). UCA 5.2.0 is similar but uses `3ACA`, for character `0x1342E`.

MySQL-Specific LDML Extensions

An extension to LDML rules permits the `<collation>` element to include an optional `version` attribute in `<collation>` tags to indicate the UCA version on which the collation is based. If the `version` attribute is omitted, its default value is `4.0.0`. For example, this specification indicates a collation that is based on UCA 5.2.0:

```
<collation id="nnn" name="utf8_xxx_ci" version="5.2.0">
...
</collation>
```

10.14.4.3 Diagnostics During Index.xml Parsing

The MySQL server generates diagnostics when it finds problems while parsing the `Index.xml` file:

- Unknown tags are written to the error log. For example, the following message results if a collation definition contains a `<aaa>` tag:

  ```
  [Warning] Buffered warning: Unknown LDML tag:
  'charsets/charset/collation/rules/aaa'
  ```

- If collation initialization is not possible, the server reports an “Unknown collation” error, and also generates warnings explaining the problems, such as in the previous example. In other cases, when a collation description is generally correct but contains some unknown tags, the collation is initialized and is available for use. The unknown parts are ignored, but a warning is generated in the error log.

- Problems with collations generate warnings that clients can display with `SHOW WARNINGS`. Suppose that a reset rule contains an expansion longer than the maximum supported length of 6 characters:

  ```
  <reset>abcdefghi</reset>
  <i>x</i>
  ```

  An attempt to use the collation produces warnings:

  ```
  mysql> SELECT _utf8'test' COLLATE utf8_test_ci;
  ERROR 1273 (HY000): Unknown collation: 'utf8_test_ci'
  mysql> SHOW WARNINGS;
  +---------+------+----------------------------------------+
  | Level   | Code | Message                                |
  +---------+------+----------------------------------------+
  | Error   | 1273 | Unknown collation: 'utf8_test_ci'      |
  ```

```
| Warning | 1273 | Expansion is too long at 'abcdefghi=x' |
+---------+------+----------------------------------------+
```

10.15 Character Set Configuration

The MySQL server has a compiled-in default character set and collation. To change these defaults, use the `--character-set-server` and `--collation-server` options when you start the server. See Section 5.1.7, “Server Command Options”. The collation must be a legal collation for the default character set. To determine which collations are available for each character set, use the `SHOW COLLATION` statement or query the `INFORMATION_SCHEMA COLLATIONS` table.

If you try to use a character set that is not compiled into your binary, you might run into the following problems:

- If your program uses an incorrect path to determine where the character sets are stored (which is typically the `share/mysql/charsets` or `share/charsets` directory under the MySQL installation directory), this can be fixed by using the `--character-sets-dir` option when you run the program. For example, to specify a directory to be used by MySQL client programs, list it in the `[client]` group of your option file. The examples given here show what the setting might look like for Unix or Windows, respectively:

  ```
  [client]
  character-sets-dir=/usr/local/mysql/share/mysql/charsets

  [client]
  character-sets-dir="C:/Program Files/MySQL/MySQL Server 8.0/share/charsets"
  ```

- If the character set is a complex character set that cannot be loaded dynamically, you must recompile the program with support for the character set.

  For Unicode character sets, you can define collations without recompiling by using LDML notation. See Section 10.14.4, “Adding a UCA Collation to a Unicode Character Set”.

- If the character set is a dynamic character set, but you do not have a configuration file for it, you should install the configuration file for the character set from a new MySQL distribution.

- If your character set index file (`Index.xml`) does not contain the name for the character set, your program displays an error message:

  ```
  Character set 'charset_name' is not a compiled character set and is not
  specified in the '/usr/share/mysql/charsets/Index.xml' file
  ```

  To solve this problem, you should either get a new index file or manually add the name of any missing character sets to the current file.

You can force client programs to use specific character set as follows:

```
[client]
default-character-set=charset_name
```

This is normally unnecessary. However, when `character_set_system` differs from `character_set_server` or `character_set_client`, and you input characters manually (as database object identifiers, column values, or both), these may be displayed incorrectly in output from the client or the output itself may be formatted incorrectly. In such cases, starting the mysql client with `--default-character-set=system_character_set`—that is, setting the client character set to match the system character set—should fix the problem.

10.16 MySQL Server Locale Support

The locale indicated by the `lc_time_names` system variable controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()`, and `MONTHNAME()` functions.

`lc_time_names` does not affect the `STR_TO_DATE()` or `GET_FORMAT()` function.

The `lc_time_names` value does not affect the result from `FORMAT()`, but this function takes an optional third parameter that enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the `lc_time_names` system variable.

Locale names have language and region subtags listed by IANA (http://www.iana.org/assignments/language-subtag-registry) such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting, but you can set the value at server startup, or set the `GLOBAL` value at runtime if you have privileges sufficient to set global system variables; see Section 5.1.9.1, “System Variable Privileges”. Any client can examine the value of `lc_time_names` or set its `SESSION` value to affect the locale for its own connection.

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| en_US           |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----------------------+-------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----------------------+-------------------------+
| Friday                | January                 |
+-----------------------+-------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+----------------------------------------+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+----------------------------------------+
| Friday Fri January Jan                 |
+----------------------------------------+
1 row in set (0.00 sec)

mysql> SET lc_time_names = 'es_MX';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| es_MX           |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+-----------------------+-------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+-----------------------+-------------------------+
```

```
| viernes               | enero                  |
+-----------------------+------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+----------------------------------------+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+----------------------------------------+
| viernes vie enero ene                  |
+----------------------------------------+
1 row in set (0.00 sec)
```

The day or month name for each of the affected functions is converted from `utf8` to the character set indicated by the `character_set_connection` system variable.

`lc_time_names` may be set to any of the following locale values. The set of locales supported by MySQL may differ from those supported by your operating system.

Locale Value	Meaning
`ar_AE`: Arabic - United Arab Emirates	`ar_BH`: Arabic - Bahrain
`ar_DZ`: Arabic - Algeria	`ar_EG`: Arabic - Egypt
`ar_IN`: Arabic - India	`ar_IQ`: Arabic - Iraq
`ar_JO`: Arabic - Jordan	`ar_KW`: Arabic - Kuwait
`ar_LB`: Arabic - Lebanon	`ar_LY`: Arabic - Libya
`ar_MA`: Arabic - Morocco	`ar_OM`: Arabic - Oman
`ar_QA`: Arabic - Qatar	`ar_SA`: Arabic - Saudi Arabia
`ar_SD`: Arabic - Sudan	`ar_SY`: Arabic - Syria
`ar_TN`: Arabic - Tunisia	`ar_YE`: Arabic - Yemen
`be_BY`: Belarusian - Belarus	`bg_BG`: Bulgarian - Bulgaria
`ca_ES`: Catalan - Spain	`cs_CZ`: Czech - Czech Republic
`da_DK`: Danish - Denmark	`de_AT`: German - Austria
`de_BE`: German - Belgium	`de_CH`: German - Switzerland
`de_DE`: German - Germany	`de_LU`: German - Luxembourg
`el_GR`: Greek - Greece	`en_AU`: English - Australia
`en_CA`: English - Canada	`en_GB`: English - United Kingdom
`en_IN`: English - India	`en_NZ`: English - New Zealand
`en_PH`: English - Philippines	`en_US`: English - United States
`en_ZA`: English - South Africa	`en_ZW`: English - Zimbabwe
`es_AR`: Spanish - Argentina	`es_BO`: Spanish - Bolivia
`es_CL`: Spanish - Chile	`es_CO`: Spanish - Colombia
`es_CR`: Spanish - Costa Rica	`es_DO`: Spanish - Dominican Republic
`es_EC`: Spanish - Ecuador	`es_ES`: Spanish - Spain
`es_GT`: Spanish - Guatemala	`es_HN`: Spanish - Honduras
`es_MX`: Spanish - Mexico	`es_NI`: Spanish - Nicaragua
`es_PA`: Spanish - Panama	`es_PE`: Spanish - Peru
`es_PR`: Spanish - Puerto Rico	`es_PY`: Spanish - Paraguay

Locale Value	Meaning
es_SV: Spanish - El Salvador	es_US: Spanish - United States
es_UY: Spanish - Uruguay	es_VE: Spanish - Venezuela
et_EE: Estonian - Estonia	eu_ES: Basque - Basque
fi_FI: Finnish - Finland	fo_FO: Faroese - Faroe Islands
fr_BE: French - Belgium	fr_CA: French - Canada
fr_CH: French - Switzerland	fr_FR: French - France
fr_LU: French - Luxembourg	gl_ES: Galician - Spain
gu_IN: Gujarati - India	he_IL: Hebrew - Israel
hi_IN: Hindi - India	hr_HR: Croatian - Croatia
hu_HU: Hungarian - Hungary	id_ID: Indonesian - Indonesia
is_IS: Icelandic - Iceland	it_CH: Italian - Switzerland
it_IT: Italian - Italy	ja_JP: Japanese - Japan
ko_KR: Korean - Republic of Korea	lt_LT: Lithuanian - Lithuania
lv_LV: Latvian - Latvia	mk_MK: Macedonian - FYROM
mn_MN: Mongolia - Mongolian	ms_MY: Malay - Malaysia
nb_NO: Norwegian(Bokmål) - Norway	nl_BE: Dutch - Belgium
nl_NL: Dutch - The Netherlands	no_NO: Norwegian - Norway
pl_PL: Polish - Poland	pt_BR: Portugese - Brazil
pt_PT: Portugese - Portugal	rm_CH: Romansh - Switzerland
ro_RO: Romanian - Romania	ru_RU: Russian - Russia
ru_UA: Russian - Ukraine	sk_SK: Slovak - Slovakia
sl_SI: Slovenian - Slovenia	sq_AL: Albanian - Albania
sr_RS: Serbian - Yugoslavia	sv_FI: Swedish - Finland
sv_SE: Swedish - Sweden	ta_IN: Tamil - India
te_IN: Telugu - India	th_TH: Thai - Thailand
tr_TR: Turkish - Turkey	uk_UA: Ukrainian - Ukraine
ur_PK: Urdu - Pakistan	vi_VN: Vietnamese - Viet Nam
zh_CN: Chinese - China	zh_HK: Chinese - Hong Kong
zh_TW: Chinese - Taiwan Province of China	

Chapter 11 Data Types

Table of Contents

11.1 Data Type Overview 1818
11.1.1 Numeric Type Overview 1818
11.1.2 Date and Time Type Overview 1822
11.1.3 String Type Overview 1824
11.2 Numeric Types 1828
11.2.1 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT 1828
11.2.2 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC 1829
11.2.3 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE 1829
11.2.4 Bit-Value Type - BIT 1830
11.2.5 Numeric Type Attributes 1830
11.2.6 Out-of-Range and Overflow Handling 1831
11.3 Date and Time Types 1833
11.3.1 The DATE, DATETIME, and TIMESTAMP Types 1834
11.3.2 The TIME Type 1836
11.3.3 The YEAR Type 1836
11.3.4 Automatic Initialization and Updating for TIMESTAMP and DATETIME 1837
11.3.5 Fractional Seconds in Time Values 1840
11.3.6 Conversion Between Date and Time Types 1841
11.3.7 Two-Digit Years in Dates 1842
11.4 String Types 1843
11.4.1 The CHAR and VARCHAR Types 1843
11.4.2 The BINARY and VARBINARY Types 1845
11.4.3 The BLOB and TEXT Types 1846
11.4.4 The ENUM Type 1848
11.4.5 The SET Type 1851
11.5 Spatial Data Types 1853
11.5.1 Spatial Data Types 1855
11.5.2 The OpenGIS Geometry Model 1856
11.5.3 Supported Spatial Data Formats 1862
11.5.4 Geometry Well-Formedness and Validity 1865
11.5.5 Spatial Reference System Support 1866
11.5.6 Creating Spatial Columns 1867
11.5.7 Populating Spatial Columns 1867
11.5.8 Fetching Spatial Data 1869
11.5.9 Optimizing Spatial Analysis 1869
11.5.10 Creating Spatial Indexes 1869
11.5.11 Using Spatial Indexes 1871
11.6 The JSON Data Type 1872
11.7 Data Type Default Values 1890
11.8 Data Type Storage Requirements 1893
11.9 Choosing the Right Type for a Column 1897
11.10 Using Data Types from Other Database Engines 1898

MySQL supports a number of SQL data types in several categories: numeric types, date and time types, string (character and byte) types, spatial types, and the `JSON` data type. This chapter provides an overview of these data types, a more detailed description of the properties of the types in each category, and

a summary of the data type storage requirements. The initial overview is intentionally brief. The more detailed descriptions later in the chapter should be consulted for additional information about particular data types, such as the permissible formats in which you can specify values.

Data type descriptions use these conventions:

- For integer types, *M* indicates the maximum display width. For floating-point and fixed-point types, *M* is the total number of digits that can be stored (the precision). For string types, *M* is the maximum length. The maximum permissible value of *M* depends on the data type.
- *D* applies to floating-point and fixed-point types and indicates the number of digits following the decimal point (the scale). The maximum possible value is 30, but should be no greater than *M*−2.
- *fsp* applies to the `TIME`, `DATETIME`, and `TIMESTAMP` types and represents fractional seconds precision; that is, the number of digits following the decimal point for fractional parts of seconds. The *fsp* value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)
- Square brackets (`[` and `]`) indicate optional parts of type definitions.

11.1 Data Type Overview

11.1.1 Numeric Type Overview

A summary of the numeric data types follows. For additional information about properties and storage requirements of the numeric types, see Section 11.2, “Numeric Types”, and Section 11.8, “Data Type Storage Requirements”.

For integer types, *M* indicates the maximum display width. The maximum display width is 255. Display width is unrelated to the range of values a type can contain, as described in Section 11.2, “Numeric Types”.

For floating-point and fixed-point types, *M* is the total number of digits that can be stored.

As of MySQL 8.0.17, the display width attribute is deprecated for integer data types and will be removed in a future MySQL version.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

As of MySQL 8.0.17, the `ZEROFILL` attribute is deprecated for numeric data types and will be removed in a future MySQL version. Consider using an alternative means of producing the effect of this attribute. For example, applications could use the `LPAD()` function to zero-pad numbers up to the desired width, or they could store the formatted numbers in `CHAR` columns.

Numeric data types that permit the `UNSIGNED` attribute also permit `SIGNED`. However, these data types are signed by default, so the `SIGNED` attribute has no effect.

As of MySQL 8.0.17, the `UNSIGNED` attribute is deprecated for columns of type `FLOAT`, `DOUBLE`, and `DECIMAL` (and any synonyms) and will be removed in a future MySQL version. Consider using a simple `CHECK` constraint instead for such columns.

`SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

`SERIAL DEFAULT VALUE` in the definition of an integer column is an alias for `NOT NULL AUTO_INCREMENT UNIQUE`.

Warning

When you use subtraction between integer values where one is of type `UNSIGNED`, the result is unsigned unless the `NO_UNSIGNED_SUBTRACTION` SQL mode is enabled. See Section 12.10, “Cast Functions and Operators”.

- `BIT[(M)]`

  A bit-value type. *`M`* indicates the number of bits per value, from 1 to 64. The default is 1 if *`M`* is omitted.

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]`

  A very small integer. The signed range is `-128` to `127`. The unsigned range is `0` to `255`.

- `BOOL`, `BOOLEAN`

  These types are synonyms for `TINYINT(1)`. A value of zero is considered false. Nonzero values are considered true:

```
mysql> SELECT IF(0, 'true', 'false');
+-----------------------+
| IF(0, 'true', 'false') |
+-----------------------+
| false                 |
+-----------------------+

mysql> SELECT IF(1, 'true', 'false');
+-----------------------+
| IF(1, 'true', 'false') |
+-----------------------+
| true                  |
+-----------------------+

mysql> SELECT IF(2, 'true', 'false');
+-----------------------+
| IF(2, 'true', 'false') |
+-----------------------+
| true                  |
+-----------------------+
```

  However, the values `TRUE` and `FALSE` are merely aliases for `1` and `0`, respectively, as shown here:

```
mysql> SELECT IF(0 = FALSE, 'true', 'false');
+--------------------------------+
| IF(0 = FALSE, 'true', 'false') |
+--------------------------------+
| true                           |
+--------------------------------+

mysql> SELECT IF(1 = TRUE, 'true', 'false');
+-------------------------------+
| IF(1 = TRUE, 'true', 'false') |
+-------------------------------+
| true                          |
+-------------------------------+

mysql> SELECT IF(2 = TRUE, 'true', 'false');
+-------------------------------+
| IF(2 = TRUE, 'true', 'false') |
```

```
+------------------------------+
| false                        |
+------------------------------+

mysql> SELECT IF(2 = FALSE, 'true', 'false');
+--------------------------------+
| IF(2 = FALSE, 'true', 'false') |
+--------------------------------+
| false                          |
+--------------------------------+
```

The last two statements display the results shown because `2` is equal to neither `1` nor `0`.

- `SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`

  A small integer. The signed range is `-32768` to `32767`. The unsigned range is `0` to `65535`.

- `MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`

  A medium-sized integer. The signed range is `-8388608` to `8388607`. The unsigned range is `0` to `16777215`.

- `INT[(M)] [UNSIGNED] [ZEROFILL]`

  A normal-size integer. The signed range is `-2147483648` to `2147483647`. The unsigned range is `0` to `4294967295`.

- `INTEGER[(M)] [UNSIGNED] [ZEROFILL]`

  This type is a synonym for `INT`.

- `BIGINT[(M)] [UNSIGNED] [ZEROFILL]`

  A large integer. The signed range is `-9223372036854775808` to `9223372036854775807`. The unsigned range is `0` to `18446744073709551615`.

  `SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

  Some things you should be aware of with respect to `BIGINT` columns:

  - All arithmetic is done using signed `BIGINT` or `DOUBLE` values, so you should not use unsigned big integers larger than `9223372036854775807` (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a `BIGINT` value to a `DOUBLE`.

    MySQL can handle `BIGINT` in the following cases:

    - When using integers to store large unsigned values in a `BIGINT` column.
    - In `MIN(col_name)` or `MAX(col_name)`, where `col_name` refers to a `BIGINT` column.
    - When using operators (`+`, `-`, `*`, and so on) where both operands are integers.

  - You can always store an exact integer value in a `BIGINT` column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.

  - The `-`, `+`, and `*` operators use `BIGINT` arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than `9223372036854775807`.

- `DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]`

  A packed “exact” fixed-point number. *M* is the total number of digits (the precision) and *D* is the number of digits after the decimal point (the scale). The decimal point and (for negative numbers) the `-` sign are not counted in *M*. If *D* is 0, values have no decimal point or fractional part. The maximum number of digits (*M*) for `DECIMAL` is 65. The maximum number of supported decimals (*D*) is 30. If *D* is omitted, the default is 0. If *M* is omitted, the default is 10.

  `UNSIGNED`, if specified, disallows negative values. As of MySQL 8.0.17, the `UNSIGNED` attribute is deprecated for columns of type `DECIMAL` (and any synonyms) and will be removed in a future MySQL version. Consider using a simple `CHECK` constraint instead for such columns.

  All basic calculations (`+, -, *, /`) with `DECIMAL` columns are done with a precision of 65 digits.

- `DEC[(M[,D])] [UNSIGNED] [ZEROFILL]`, `NUMERIC[(M[,D])] [UNSIGNED] [ZEROFILL]`, `FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]`

  These types are synonyms for `DECIMAL`. The `FIXED` synonym is available for compatibility with other database systems.

- `FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]`

  A small (single-precision) floating-point number. Permissible values are `-3.402823466E+38` to `-1.175494351E-38`, `0`, and `1.175494351E-38` to `3.402823466E+38`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

  M is the total number of digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits permitted by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

  `FLOAT(M,D)` is a nonstandard MySQL extension. As of MySQL 8.0.17, this syntax is deprecated and support for it will be removed in a future MySQL version.

  `UNSIGNED`, if specified, disallows negative values. As of MySQL 8.0.17, the `UNSIGNED` attribute is deprecated for columns of type `FLOAT` (and any synonyms) and will be removed in a future MySQL version. Consider using a simple `CHECK` constraint instead for such columns.

  Using `FLOAT` might give you some unexpected problems because all calculations in MySQL are done with double precision. See Section B.4.4.7, “Solving Problems with No Matching Rows”.

- `FLOAT(p) [UNSIGNED] [ZEROFILL]`

  A floating-point number. *p* represents the precision in bits, but MySQL uses this value only to determine whether to use `FLOAT` or `DOUBLE` for the resulting data type. If *p* is from 0 to 24, the data type becomes `FLOAT` with no *M* or *D* values. If *p* is from 25 to 53, the data type becomes `DOUBLE` with no *M* or *D* values. The range of the resulting column is the same as for the single-precision `FLOAT` or double-precision `DOUBLE` data types described earlier in this section.

  `UNSIGNED`, if specified, disallows negative values. As of MySQL 8.0.17, the `UNSIGNED` attribute is deprecated for columns of type `FLOAT` (and any synonyms) and will be removed in a future MySQL version. Consider using a simple `CHECK` constraint instead for such columns.

  `FLOAT(p)` syntax is provided for ODBC compatibility.

- `DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]`

A normal-size (double-precision) floating-point number. Permissible values are `-1.7976931348623157E+308` to `-2.2250738585072014E-308`, `0`, and `2.2250738585072014E-308` to `1.7976931348623157E+308`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

`M` is the total number of digits and *`D`* is the number of digits following the decimal point. If *`M`* and *`D`* are omitted, values are stored to the limits permitted by the hardware. A double-precision floating-point number is accurate to approximately 15 decimal places.

`DOUBLE(`*`M`*`,`*`D`*`)` is a nonstandard MySQL extension. As of MySQL 8.0.17, this syntax is deprecated and support for it will be removed in a future MySQL version.

`UNSIGNED`, if specified, disallows negative values. As of MySQL 8.0.17, the `UNSIGNED` attribute is deprecated for columns of type `DOUBLE` (and any synonyms) and will be removed in a future MySQL version. Consider using a simple `CHECK` constraint instead for such columns.

- `DOUBLE PRECISION[(`*`M`*`,`*`D`*`)] [UNSIGNED] [ZEROFILL]`, `REAL[(`*`M`*`,`*`D`*`)] [UNSIGNED] [ZEROFILL]`

  These types are synonyms for `DOUBLE`. Exception: If the `REAL_AS_FLOAT` SQL mode is enabled, `REAL` is a synonym for `FLOAT` rather than `DOUBLE`.

11.1.2 Date and Time Type Overview

A summary of the temporal data types follows. For additional information about properties and storage requirements of the temporal types, see Section 11.3, “Date and Time Types”, and Section 11.8, “Data Type Storage Requirements”. For descriptions of functions that operate on temporal values, see Section 12.7, “Date and Time Functions”.

For the `DATE` and `DATETIME` range descriptions, “supported” means that although earlier values might work, there is no guarantee.

MySQL permits fractional seconds for `TIME`, `DATETIME`, and `TIMESTAMP` values, with up to microseconds (6 digits) precision. To define a column that includes a fractional seconds part, use the syntax *`type_name`*`(`*`fsp`*`)`, where *`type_name`* is `TIME`, `DATETIME`, or `TIMESTAMP`, and *`fsp`* is the fractional seconds precision. For example:

```
CREATE TABLE t1 (t TIME(3), dt DATETIME(6));
```

The *`fsp`* value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)

Any `TIMESTAMP` or `DATETIME` column in a table can have automatic initialization and updating properties.

- `DATE`

  A date. The supported range is `'1000-01-01'` to `'9999-12-31'`. MySQL displays `DATE` values in `'YYYY-MM-DD'` format, but permits assignment of values to `DATE` columns using either strings or numbers.

- `DATETIME[(`*`fsp`*`)]`

  A date and time combination. The supported range is `'1000-01-01 00:00:00.000000'` to `'9999-12-31 23:59:59.999999'`. MySQL displays `DATETIME` values in *`'YYYY-MM-DD`*

hh:mm:ss[.fraction]' format, but permits assignment of values to DATETIME columns using either strings or numbers.

An optional *fsp* value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

Automatic initialization and updating to the current date and time for DATETIME columns can be specified using DEFAULT and ON UPDATE column definition clauses, as described in Section 11.3.4, "Automatic Initialization and Updating for TIMESTAMP and DATETIME".

- TIMESTAMP[(*fsp*)]

  A timestamp. The range is '1970-01-01 00:00:01.000000' UTC to '2038-01-19 03:14:07.999999' UTC. TIMESTAMP values are stored as the number of seconds since the epoch ('1970-01-01 00:00:00' UTC). A TIMESTAMP cannot represent the value '1970-01-01 00:00:00' because that is equivalent to 0 seconds from the epoch and the value 0 is reserved for representing '0000-00-00 00:00:00', the "zero" TIMESTAMP value.

  An optional *fsp* value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

  The way the server handles TIMESTAMP definitions depends on the value of the explicit_defaults_for_timestamp system variable (see Section 5.1.8, "Server System Variables").

  If explicit_defaults_for_timestamp is enabled, there is no automatic assignment of the DEFAULT CURRENT_TIMESTAMP or ON UPDATE CURRENT_TIMESTAMP attributes to any TIMESTAMP column. They must be included explicitly in the column definition. Also, any TIMESTAMP not explicitly declared as NOT NULL permits NULL values.

  If explicit_defaults_for_timestamp is disabled, the server handles TIMESTAMP as follows:

  Unless specified otherwise, the first TIMESTAMP column in a table is defined to be automatically set to the date and time of the most recent modification if not explicitly assigned a value. This makes TIMESTAMP useful for recording the timestamp of an INSERT or UPDATE operation. You can also set any TIMESTAMP column to the current date and time by assigning it a NULL value, unless it has been defined with the NULL attribute to permit NULL values.

  Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP column definition clauses. By default, the first TIMESTAMP column has these properties, as previously noted. However, any TIMESTAMP column in a table can be defined to have these properties.

- TIME[(*fsp*)]

  A time. The range is '-838:59:59.000000' to '838:59:59.000000'. MySQL displays TIME values in '*hh:mm:ss[.fraction]*' format, but permits assignment of values to TIME columns using either strings or numbers.

  An optional *fsp* value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

- YEAR[(4)]

  A year in four-digit format. MySQL displays YEAR values in YYYY format, but permits assignment of values to YEAR columns using either strings or numbers. Values display as 1901 to 2155, and 0000.

For additional information about YEAR display format and interpretation of input values, see Section 11.3.3, “The YEAR Type”.

Note

MySQL 8.0 does not support the YEAR(2) data type permitted in older versions of MySQL. For instructions on converting to YEAR(4), see YEAR(2) Limitations and Migrating to YEAR(4) in MySQL 5.7 Reference Manual.

The SUM() and AVG() aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

11.1.3 String Type Overview

A summary of the string data types follows. For additional information about properties and storage requirements of the string types, see Section 11.4, “String Types”, and Section 11.8, “Data Type Storage Requirements”.

In some cases, MySQL may change a string column to a type different from that given in a CREATE TABLE or ALTER TABLE statement. See Section 13.1.20.8, “Silent Column Specification Changes”.

MySQL interprets length specifications in character column definitions in character units. This applies to CHAR, VARCHAR, and the TEXT types.

Column definitions for character string data types CHAR, VARCHAR, the TEXT types, ENUM, SET, and any synonyms) can specify the column character set and collation:

- CHARACTER SET specifies the character set. If desired, a collation for the character set can be specified with the COLLATE attribute, along with any other attributes. For example:

  ```
  CREATE TABLE t
  (
      c1 VARCHAR(20) CHARACTER SET utf8,
      c2 TEXT CHARACTER SET latin1 COLLATE latin1_general_cs
  );
  ```

  This table definition creates a column named c1 that has a character set of utf8 with the default collation for that character set, and a column named c2 that has a character set of latin1 and a case-sensitive collation.

  The rules for assigning the character set and collation when either or both of CHARACTER SET and the COLLATE attribute are missing are described in Section 10.3.5, “Column Character Set and Collation”.

  CHARSET is a synonym for CHARACTER SET.

- Specifying the CHARACTER SET binary attribute for a character string data type causes the column to be created as the corresponding binary string data type: CHAR becomes BINARY, VARCHAR becomes VARBINARY, and TEXT becomes BLOB. For the ENUM and SET data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

  ```
  ```

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET binary,
  c2 TEXT CHARACTER SET binary,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

The resulting table has this definition:

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

- The `BINARY` attribute is a nonstandard MySQL extension that is shorthand for specifying the binary (`_bin`) collation of the column character set (or of the table default character set if no column character set is specified). In this case, comparison and sorting are based on numeric character code values. Suppose that you specify a table using this definition:

  ```
  CREATE TABLE t
  (
    c1 VARCHAR(10) CHARACTER SET latin1 BINARY,
    c2 TEXT BINARY
  ) CHARACTER SET utf8mb4;
  ```

  The resulting table has this definition:

  ```
  CREATE TABLE t (
    c1 VARCHAR(10) CHARACTER SET latin1 COLLATE latin1_bin,
    c2 TEXT CHARACTER SET utf8mb4 COLLATE utf8mb4_bin
  ) CHARACTER SET utf8mb4;
  ```

  In MySQL 8.0, this nonstandard use of the `BINARY` attribute is ambiguous because the `utf8mb4` character set has multiple `_bin` collations. As of MySQL 8.0.17, the `BINARY` attribute is deprecated and support for it will be removed in a future MySQL version. Applications should be adjusted to use an explicit `_bin` collation instead.

  The use of `BINARY` to specify a data type or character set remains unchanged.

- The `ASCII` attribute is shorthand for `CHARACTER SET latin1`.

- The `UNICODE` attribute is shorthand for `CHARACTER SET ucs2`.

Character column comparison and sorting are based on the collation assigned to the column. For the `CHAR`, `VARCHAR`, `TEXT`, `ENUM`, and `SET` data types, you can declare a column with a binary (`_bin`) collation or the `BINARY` attribute to cause comparison and sorting to use the underlying character code values rather than a lexical ordering.

For additional information about use of character sets in MySQL, see Chapter 10, *Character Sets, Collations, Unicode*.

- `[NATIONAL] CHAR[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]`

  A fixed-length string that is always right-padded with spaces to the specified length when stored. *`M`* represents the column length in characters. The range of *`M`* is 0 to 255. If *`M`* is omitted, the length is 1.

Note

Trailing spaces are removed when `CHAR` values are retrieved unless the `PAD_CHAR_TO_FULL_LENGTH` SQL mode is enabled.

`CHAR` is shorthand for `CHARACTER`. `NATIONAL CHAR` (or its equivalent short form, `NCHAR`) is the standard SQL way to define that a `CHAR` column should use some predefined character set. MySQL uses `utf8` as this predefined character set. Section 10.3.7, “The National Character Set”.

The `CHAR BYTE` data type is an alias for the `BINARY` data type. This is a compatibility feature.

MySQL permits you to create a column of type `CHAR(0)`. This is useful primarily when you have to be compliant with old applications that depend on the existence of a column but that do not actually use its value. `CHAR(0)` is also quite nice when you need a column that can take only two values: A column that is defined as `CHAR(0) NULL` occupies only one bit and can take only the values `NULL` and `''` (the empty string).

- `[NATIONAL] VARCHAR(M) [CHARACTER SET charset_name] [COLLATE collation_name]`

  A variable-length string. *`M`* represents the maximum column length in characters. The range of *`M`* is 0 to 65,535. The effective maximum length of a `VARCHAR` is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. For example, `utf8` characters can require up to three bytes per character, so a `VARCHAR` column that uses the `utf8` character set can be declared to be a maximum of 21,844 characters. See Section 8.4.7, “Limits on Table Column Count and Row Size”.

  MySQL stores `VARCHAR` values as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A `VARCHAR` column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

  Note

  MySQL follows the standard SQL specification, and does *not* remove trailing spaces from `VARCHAR` values.

  `VARCHAR` is shorthand for `CHARACTER VARYING`. `NATIONAL VARCHAR` is the standard SQL way to define that a `VARCHAR` column should use some predefined character set. MySQL uses `utf8` as this predefined character set. Section 10.3.7, “The National Character Set”. `NVARCHAR` is shorthand for `NATIONAL VARCHAR`.

- `BINARY[(M)]`

  The `BINARY` type is similar to the `CHAR` type, but stores binary byte strings rather than nonbinary character strings. An optional length *`M`* represents the column length in bytes. If omitted, *`M`* defaults to 1.

- `VARBINARY(M)`

  The `VARBINARY` type is similar to the `VARCHAR` type, but stores binary byte strings rather than nonbinary character strings. *`M`* represents the maximum column length in bytes.

- `TINYBLOB`

  A `BLOB` column with a maximum length of 255 ($2^8 - 1$) bytes. Each `TINYBLOB` value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

- `TINYTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

  A `TEXT` column with a maximum length of 255 ($2^8 - 1$) characters. The effective maximum length is less if the value contains multibyte characters. Each `TINYTEXT` value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

- `BLOB[(M)]`

  A `BLOB` column with a maximum length of 65,535 ($2^{16} - 1$) bytes. Each `BLOB` value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

  An optional length *`M`* can be given for this type. If this is done, MySQL creates the column as the smallest `BLOB` type large enough to hold values *`M`* bytes long.

- `TEXT[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]`

  A `TEXT` column with a maximum length of 65,535 ($2^{16} - 1$) characters. The effective maximum length is less if the value contains multibyte characters. Each `TEXT` value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

  An optional length *`M`* can be given for this type. If this is done, MySQL creates the column as the smallest `TEXT` type large enough to hold values *`M`* characters long.

- `MEDIUMBLOB`

  A `BLOB` column with a maximum length of 16,777,215 ($2^{24} - 1$) bytes. Each `MEDIUMBLOB` value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

- `MEDIUMTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

  A `TEXT` column with a maximum length of 16,777,215 ($2^{24} - 1$) characters. The effective maximum length is less if the value contains multibyte characters. Each `MEDIUMTEXT` value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

- `LONGBLOB`

  A `BLOB` column with a maximum length of 4,294,967,295 or 4GB ($2^{32} - 1$) bytes. The effective maximum length of `LONGBLOB` columns depends on the configured maximum packet size in the client/server protocol and available memory. Each `LONGBLOB` value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

- `LONGTEXT [CHARACTER SET charset_name] [COLLATE collation_name]`

  A `TEXT` column with a maximum length of 4,294,967,295 or 4GB ($2^{32} - 1$) characters. The effective maximum length is less if the value contains multibyte characters. The effective maximum length of `LONGTEXT` columns also depends on the configured maximum packet size in the client/server protocol and available memory. Each `LONGTEXT` value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

- `ENUM('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]`

  An enumeration. A string object that can have only one value, chosen from the list of values `'value1'`, `'value2'`, `...`, `NULL` or the special `''` error value. `ENUM` values are represented internally as integers.

  An `ENUM` column can have a maximum of 65,535 distinct elements.

The maximum supported length of an individual `ENUM` element is *`M`* <= 255 and (*`M`* x *`w`*) <= 1020, where `M` is the element literal length and *`w`* is the number of bytes required for the maximum-length character in the character set.

- `SET('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]`

  A set. A string object that can have zero or more values, each of which must be chosen from the list of values `'value1'`, `'value2'`, `...` `SET` values are represented internally as integers.

  A `SET` column can have a maximum of 64 distinct members.

  The maximum supported length of an individual `SET` element is *`M`* <= 255 and (*`M`* x *`w`*) <= 1020, where `M` is the element literal length and *`w`* is the number of bytes required for the maximum-length character in the character set.

11.2 Numeric Types

MySQL supports all standard SQL numeric data types. These types include the exact numeric data types (`INTEGER`, `SMALLINT`, `DECIMAL`, and `NUMERIC`), as well as the approximate numeric data types (`FLOAT`, `REAL`, and `DOUBLE PRECISION`). The keyword `INT` is a synonym for `INTEGER`, and the keywords `DEC` and `FIXED` are synonyms for `DECIMAL`. MySQL treats `DOUBLE` as a synonym for `DOUBLE PRECISION` (a nonstandard extension). MySQL also treats `REAL` as a synonym for `DOUBLE PRECISION` (a nonstandard variation), unless the `REAL_AS_FLOAT` SQL mode is enabled.

The `BIT` data type stores bit values and is supported for `MyISAM`, `MEMORY`, `InnoDB`, and `NDB` tables.

For information about how MySQL handles assignment of out-of-range values to columns and overflow during expression evaluation, see Section 11.2.6, “Out-of-Range and Overflow Handling”.

For information about numeric type storage requirements, see Section 11.8, “Data Type Storage Requirements”.

The data type used for the result of a calculation on numeric operands depends on the types of the operands and the operations performed on them. For more information, see Section 12.6.1, “Arithmetic Operators”.

11.2.1 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT

MySQL supports the SQL standard integer types `INTEGER` (or `INT`) and `SMALLINT`. As an extension to the standard, MySQL also supports the integer types `TINYINT`, `MEDIUMINT`, and `BIGINT`. The following table shows the required storage and range for each integer type.

Table 11.1 Required Storage and Range for Integer Types Supported by MySQL

Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
`TINYINT`	1	`-128`	`0`	`127`	`255`
`SMALLINT`	2	`-32768`	`0`	`32767`	`65535`
`MEDIUMINT`	3	`-8388608`	`0`	`8388607`	`16777215`
`INT`	4	`-2147483648`	`0`	`2147483647`	`4294967295`

Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
BIGINT	8	-2^{63}	0	$2^{63}-1$	$2^{64}-1$

11.2.2 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC

The DECIMAL and NUMERIC types store exact numeric data values. These types are used when it is important to preserve exact precision, for example with monetary data. In MySQL, NUMERIC is implemented as DECIMAL, so the following remarks about DECIMAL apply equally to NUMERIC.

MySQL stores DECIMAL values in binary format. See Section 12.25, “Precision Math”.

In a DECIMAL column declaration, the precision and scale can be (and usually is) specified. For example:

```
salary DECIMAL(5,2)
```

In this example, 5 is the precision and 2 is the scale. The precision represents the number of significant digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point.

Standard SQL requires that DECIMAL(5,2) be able to store any value with five digits and two decimals, so values that can be stored in the salary column range from -999.99 to 999.99.

In standard SQL, the syntax DECIMAL(*M*) is equivalent to DECIMAL(*M*,0). Similarly, the syntax DECIMAL is equivalent to DECIMAL(*M*,0), where the implementation is permitted to decide the value of *M*. MySQL supports both of these variant forms of DECIMAL syntax. The default value of *M* is 10.

If the scale is 0, DECIMAL values contain no decimal point or fractional part.

The maximum number of digits for DECIMAL is 65, but the actual range for a given DECIMAL column can be constrained by the precision or scale for a given column. When such a column is assigned a value with more digits following the decimal point than are permitted by the specified scale, the value is converted to that scale. (The precise behavior is operating system-specific, but generally the effect is truncation to the permissible number of digits.)

11.2.3 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE

The FLOAT and DOUBLE types represent approximate numeric data values. MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

For FLOAT, the SQL standard permits an optional specification of the precision (but not the range of the exponent) in bits following the keyword FLOAT in parentheses; ; that is, FLOAT(*p*). MySQL also supports this optional precision specification, but the precision value in FLOAT(*p*) is used only to determine storage size. A precision from 0 to 23 results in a 4-byte single-precision FLOAT column. A precision from 24 to 53 results in an 8-byte double-precision DOUBLE column.

MySQL permits a nonstandard syntax: FLOAT(*M*,*D*) or REAL(*M*,*D*) or DOUBLE PRECISION(*M*,*D*). Here, (*M*,*D*) means than values can be stored with up to *M* digits in total, of which *D* digits may be after the decimal point. For example, a column defined as FLOAT(7,4) will look like -999.9999 when displayed. MySQL performs rounding when storing values, so if you insert 999.00009 into a FLOAT(7,4) column, the approximate result is 999.0001.

As of MySQL 8.0.17, the nonstandard FLOAT(*M*,*D*) and DOUBLE(*M*,*D*) syntax is deprecated and support for it will be removed in a future MySQL version.

Because floating-point values are approximate and not stored as exact values, attempts to treat them as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. For more information, see Section B.4.4.8, “Problems with Floating-Point Values”

For maximum portability, code requiring storage of approximate numeric data values should use `FLOAT` or `DOUBLE PRECISION` with no specification of precision or number of digits.

11.2.4 Bit-Value Type - BIT

The `BIT` data type is used to store bit values. A type of `BIT(M)` enables storage of *`M`*-bit values. *`M`* can range from 1 to 64.

To specify bit values, `b'value'` notation can be used. *`value`* is a binary value written using zeros and ones. For example, `b'111'` and `b'10000000'` represent 7 and 128, respectively. See Section 9.1.5, “Bit-Value Literals”.

If you assign a value to a `BIT(M)` column that is less than *`M`* bits long, the value is padded on the left with zeros. For example, assigning a value of `b'101'` to a `BIT(6)` column is, in effect, the same as assigning `b'000101'`.

NDB Cluster. The maximum combined size of all `BIT` columns used in a given `NDB` table must not exceed 4096 bits.

11.2.5 Numeric Type Attributes

MySQL supports an extension for optionally specifying the display width of integer data types in parentheses following the base keyword for the type. For example, `INT(4)` specifies an `INT` with a display width of four digits. This optional display width may be used by applications to display integer values having a width less than the width specified for the column by left-padding them with spaces. (That is, this width is present in the metadata returned with result sets. Whether it is used is up to the application.)

The display width does *not* constrain the range of values that can be stored in the column. Nor does it prevent values wider than the column display width from being displayed correctly. For example, a column specified as `SMALLINT(3)` has the usual `SMALLINT` range of `-32768` to `32767`, and values outside the range permitted by three digits are displayed in full using more than three digits.

When used in conjunction with the optional (nonstandard) `ZEROFILL` attribute, the default padding of spaces is replaced with zeros. For example, for a column declared as `INT(4) ZEROFILL`, a value of `5` is retrieved as `0005`.

Note

The `ZEROFILL` attribute is ignored for columns involved in expressions or `UNION` queries.

If you store values larger than the display width in an integer column that has the `ZEROFILL` attribute, you may experience problems when MySQL generates temporary tables for some complicated joins. In these cases, MySQL assumes that the data values fit within the column display width.

As of MySQL 8.0.17, the `ZEROFILL` attribute is deprecated for numeric data types, as is the display width attribute for integer data types. Support for `ZEROFILL` and display widths for integer data types will be removed in a future MySQL version. Consider using an alternative means of producing the effect of these attributes. For example, applications could use the `LPAD()` function to zero-pad numbers up to the desired width, or they could store the formatted numbers in `CHAR` columns.

All integer types can have an optional (nonstandard) `UNSIGNED` attribute. An unsigned type can be used to permit only nonnegative numbers in a column or when you need a larger upper numeric range for the column. For example, if an `INT` column is `UNSIGNED`, the size of the column's range is the same but its endpoints shift up, from `-2147483648` and `2147483647` to `0` and `4294967295`.

Floating-point and fixed-point types also can be `UNSIGNED`. As with integer types, this attribute prevents negative values from being stored in the column. Unlike the integer types, the upper range of column values remains the same. As of MySQL 8.0.17, the `UNSIGNED` attribute is deprecated for columns of type `FLOAT`, `DOUBLE`, and `DECIMAL` (and any synonyms) and will be removed in a future MySQL version. Consider using a simple `CHECK` constraint instead for such columns.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute.

Integer or floating-point data types can have the `AUTO_INCREMENT` attribute. When you insert a value of `NULL` into an indexed `AUTO_INCREMENT` column, the column is set to the next sequence value. Typically this is *`value`*`+1`, where *`value`* is the largest value for the column currently in the table. (`AUTO_INCREMENT` sequences begin with `1`.)

Storing `0` into an `AUTO_INCREMENT` column has the same effect as storing `NULL`, unless the `NO_AUTO_VALUE_ON_ZERO` SQL mode is enabled.

Inserting `NULL` to generate `AUTO_INCREMENT` values requires that the column be declared `NOT NULL`. If the column is declared `NULL`, inserting `NULL` stores a `NULL`. When you insert any other value into an `AUTO_INCREMENT` column, the column is set to that value and the sequence is reset so that the next automatically generated value follows sequentially from the inserted value.

Negative values for `AUTO_INCREMENT` columns are not supported.

`CHECK` constraints cannot refer to columns that have the `AUTO_INCREMENT` attribute, nor can the `AUTO_INCREMENT` attribute be added to existing columns that are used in `CHECK` constraints.

As of MySQL 8.0.17, `AUTO_INCREMENT` support is deprecated for `FLOAT` and `DOUBLE` columns and will be removed in a future MySQL version. Consider removing the `AUTO_INCREMENT` attribute from such columns, or convert them to an integer type.

11.2.6 Out-of-Range and Overflow Handling

When MySQL stores a value in a numeric column that is outside the permissible range of the column data type, the result depends on the SQL mode in effect at the time:

- If strict SQL mode is enabled, MySQL rejects the out-of-range value with an error, and the insert fails, in accordance with the SQL standard.
- If no restrictive modes are enabled, MySQL clips the value to the appropriate endpoint of the column data type range and stores the resulting value instead.

  When an out-of-range value is assigned to an integer column, MySQL stores the value representing the corresponding endpoint of the column data type range.

  When a floating-point or fixed-point column is assigned a value that exceeds the range implied by the specified (or default) precision and scale, MySQL stores the value representing the corresponding endpoint of that range.

Suppose that a table `t1` has this definition:

```
CREATE TABLE t1 (i1 TINYINT, i2 TINYINT UNSIGNED);
```

With strict SQL mode enabled, an out of range error occurs:

```
mysql> SET sql_mode = 'TRADITIONAL';
mysql> INSERT INTO t1 (i1, i2) VALUES(256, 256);
ERROR 1264 (22003): Out of range value for column 'i1' at row 1
mysql> SELECT * FROM t1;
Empty set (0.00 sec)
```

With strict SQL mode not enabled, clipping with warnings occurs:

```
mysql> SET sql_mode = '';
mysql> INSERT INTO t1 (i1, i2) VALUES(256, 256);
mysql> SHOW WARNINGS;
+---------+------+-------------------------------------------+
| Level   | Code | Message                                   |
+---------+------+-------------------------------------------+
| Warning | 1264 | Out of range value for column 'i1' at row 1 |
| Warning | 1264 | Out of range value for column 'i2' at row 1 |
+---------+------+-------------------------------------------+
mysql> SELECT * FROM t1;
+------+------+
| i1   | i2   |
+------+------+
|  127 |  255 |
+------+------+
```

When strict SQL mode is not enabled, column-assignment conversions that occur due to clipping are reported as warnings for `ALTER TABLE`, `LOAD DATA`, `UPDATE`, and multiple-row `INSERT` statements. In strict mode, these statements fail, and some or all the values are not inserted or changed, depending on whether the table is a transactional table and other factors. For details, see Section 5.1.11, “Server SQL Modes”.

Overflow during numeric expression evaluation results in an error. For example, the largest signed `BIGINT` value is 9223372036854775807, so the following expression produces an error:

```
mysql> SELECT 9223372036854775807 + 1;
ERROR 1690 (22003): BIGINT value is out of range in '(9223372036854775807 + 1)'
```

To enable the operation to succeed in this case, convert the value to unsigned;

```
mysql> SELECT CAST(9223372036854775807 AS UNSIGNED) + 1;
+-----------------------------------------+
| CAST(9223372036854775807 AS UNSIGNED) + 1 |
+-----------------------------------------+
|                      9223372036854775808 |
+-----------------------------------------+
```

Whether overflow occurs depends on the range of the operands, so another way to handle the preceding expression is to use exact-value arithmetic because `DECIMAL` values have a larger range than integers:

```
mysql> SELECT 9223372036854775807.0 + 1;
+--------------------------+
| 9223372036854775807.0 + 1 |
+--------------------------+
|     9223372036854775808.0 |
+--------------------------+
```

Subtraction between integer values, where one is of type `UNSIGNED`, produces an unsigned result by default. If the result would otherwise have been negative, an error results:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CAST(0 AS UNSIGNED) - 1;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(cast(0 as unsigned) - 1)'
```

If the NO_UNSIGNED_SUBTRACTION SQL mode is enabled, the result is negative:

```
mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+--------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+--------------------------+
|                       -1 |
+--------------------------+
```

If the result of such an operation is used to update an UNSIGNED integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if NO_UNSIGNED_SUBTRACTION is enabled. If strict SQL mode is enabled, an error occurs and the column remains unchanged.

11.3 Date and Time Types

The date and time types for representing temporal values are DATE, TIME, DATETIME, TIMESTAMP, and YEAR. Each temporal type has a range of valid values, as well as a “zero” value that may be used when you specify an invalid value that MySQL cannot represent. The TIMESTAMP type has special automatic updating behavior, described later. For temporal type storage requirements, see Section 11.8, “Data Type Storage Requirements”.

Keep in mind these general considerations when working with date and time types:

- MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to be assigned to or compared to a date or time type). For a description of the permitted formats for date and time types, see Section 9.1.3, “Date and Time Literals”. It is expected that you supply valid values. Unpredictable results may occur if you use values in other formats.
- Although MySQL tries to interpret values in several formats, date parts must always be given in year-month-day order (for example, '98-09-04'), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, '09-04-98', '04-09-98').
- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:
  - Year values in the range 70-99 are converted to 1970-1999.
  - Year values in the range 00-69 are converted to 2000-2069.

  See also Section 11.3.7, “Two-Digit Years in Dates”.
- Conversion of values from one temporal type to another occurs according to the rules in Section 11.3.6, “Conversion Between Date and Time Types”.
- MySQL automatically converts a date or time value to a number if the value is used in a numeric context and vice versa.
- By default, when MySQL encounters a value for a date or time type that is out of range or otherwise invalid for the type, it converts the value to the “zero” value for that type. The exception is that out-of-range TIME values are clipped to the appropriate endpoint of the TIME range.

- By setting the SQL mode to the appropriate value, you can specify more exactly what kind of dates you want MySQL to support. (See Section 5.1.11, “Server SQL Modes”.) You can get MySQL to accept certain dates, such as `'2009-11-31'`, by enabling the `ALLOW_INVALID_DATES` SQL mode. This is useful when you want to store a “possibly wrong” value which the user has specified (for example, in a web form) in the database for future processing. Under this mode, MySQL verifies only that the month is in the range from 1 to 12 and that the day is in the range from 1 to 31.
- MySQL permits you to store dates where the day or month and day are zero in a `DATE` or `DATETIME` column. This is useful for applications that need to store birthdates for which you may not know the exact date. In this case, you simply store the date as `'2009-00-00'` or `'2009-01-00'`. If you store dates such as these, you should not expect to get correct results for functions such as `DATE_SUB()` or `DATE_ADD()` that require complete dates. To disallow zero month or day parts in dates, enable the `NO_ZERO_IN_DATE` mode.
- MySQL permits you to store a “zero” value of `'0000-00-00'` as a “dummy date.” This is in some cases more convenient than using `NULL` values, and uses less data and index space. To disallow `'0000-00-00'`, enable the `NO_ZERO_DATE` mode.
- “Zero” date or time values used through Connector/ODBC are converted automatically to `NULL` because ODBC cannot handle such values.

The following table shows the format of the “zero” value for each type. The “zero” values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values `'0'` or `0`, which are easier to write. For temporal types that include a date part (`DATE`, `DATETIME`, and `TIMESTAMP`), use of these values may produce warning or errors. The precise behavior depends on which if any of strict SQL mode and the `NO_ZERO_DATE` SQL mode are enabled; see Section 5.1.11, “Server SQL Modes”.

Data Type	“Zero” Value
`DATE`	`'0000-00-00'`
`TIME`	`'00:00:00'`
`DATETIME`	`'0000-00-00 00:00:00'`
`TIMESTAMP`	`'0000-00-00 00:00:00'`
`YEAR`	`0000`

11.3.1 The DATE, DATETIME, and TIMESTAMP Types

The `DATE`, `DATETIME`, and `TIMESTAMP` types are related. This section describes their characteristics, how they are similar, and how they differ. MySQL recognizes `DATE`, `DATETIME`, and `TIMESTAMP` values in several formats, described in Section 9.1.3, “Date and Time Literals”. For the `DATE` and `DATETIME` range descriptions, “supported” means that although earlier values might work, there is no guarantee.

The `DATE` type is used for values with a date part but no time part. MySQL retrieves and displays `DATE` values in `'YYYY-MM-DD'` format. The supported range is `'1000-01-01'` to `'9999-12-31'`.

The `DATETIME` type is used for values that contain both date and time parts. MySQL retrieves and displays `DATETIME` values in *`'YYYY-MM-DD hh:mm:ss'`* format. The supported range is `'1000-01-01 00:00:00'` to `'9999-12-31 23:59:59'`.

The `TIMESTAMP` data type is used for values that contain both date and time parts. `TIMESTAMP` has a range of `'1970-01-01 00:00:01'` UTC to `'2038-01-19 03:14:07'` UTC.

A `DATETIME` or `TIMESTAMP` value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. In particular, any fractional part in a value inserted into a `DATETIME` or `TIMESTAMP`

column is stored rather than discarded. With the fractional part included, the format for these values is `'YYYY-MM-DD hh:mm:ss[.fraction]'`, the range for `DATETIME` values is `'1000-01-01 00:00:00.000000'` to `'9999-12-31 23:59:59.999999'`, and the range for `TIMESTAMP` values is `'1970-01-01 00:00:01.000000'` to `'2038-01-19 03:14:07.999999'`. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized. For information about fractional seconds support in MySQL, see Section 11.3.5, “Fractional Seconds in Time Values”.

The `TIMESTAMP` and `DATETIME` data types offer automatic initialization and updating to the current date and time. For more information, see Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

MySQL converts `TIMESTAMP` values from the current time zone to UTC for storage, and back from UTC to the current time zone for retrieval. (This does not occur for other types such as `DATETIME`.) By default, the current time zone for each connection is the server's time. The time zone can be set on a per-connection basis. As long as the time zone setting remains constant, you get back the same value you store. If you store a `TIMESTAMP` value, and then change the time zone and retrieve the value, the retrieved value is different from the value you stored. This occurs because the same time zone was not used for conversion in both directions. The current time zone is available as the value of the `time_zone` system variable. For more information, see Section 5.1.13, “MySQL Server Time Zone Support”.

Invalid `DATE`, `DATETIME`, or `TIMESTAMP` values are converted to the “zero” value of the appropriate type (`'0000-00-00'` or `'0000-00-00 00:00:00'`), if the SQL mode permits this conversion. The precise behavior depends on which if any of strict SQL mode and the `NO_ZERO_DATE` SQL mode are enabled; see Section 5.1.11, “Server SQL Modes”.

Be aware of certain properties of date value interpretation in MySQL:

- MySQL permits a “relaxed” format for values specified as strings, in which any punctuation character may be used as the delimiter between date parts or time parts. In some cases, this syntax can be deceiving. For example, a value such as `'10:11:12'` might look like a time value because of the `:`, but is interpreted as the year `'2010-11-12'` if used in a date context. The value `'10:45:15'` is converted to `'0000-00-00'` because `'45'` is not a valid month.

  The only delimiter recognized between a date and time part and a fractional seconds part is the decimal point.

- The server requires that month and day values be valid, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as `'2004-04-31'` are converted to `'0000-00-00'` and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable `ALLOW_INVALID_DATES`. See Section 5.1.11, “Server SQL Modes”, for more information.

- MySQL does not accept `TIMESTAMP` values that include a zero in the day or month column or values that are not a valid date. The sole exception to this rule is the special “zero” value `'0000-00-00 00:00:00'`, if the SQL mode permits this value. The precise behavior depends on which if any of strict SQL mode and the `NO_ZERO_DATE` SQL mode are enabled; see Section 5.1.11, “Server SQL Modes”.

- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

  - Year values in the range `00-69` are converted to `2000-2069`.

  - Year values in the range `70-99` are converted to `1970-1999`.

  See also Section 11.3.7, “Two-Digit Years in Dates”.

11.3.2 The TIME Type

MySQL retrieves and displays `TIME` values in *`'hh:mm:ss'`* format (or *`'hhh:mm:ss'`* format for large hours values). `TIME` values may range from `'-838:59:59'` to `'838:59:59'`. The hours part may be so large because the `TIME` type can be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

MySQL recognizes `TIME` values in several formats, some of which can include a trailing fractional seconds part in up to microseconds (6 digits) precision. See Section 9.1.3, “Date and Time Literals”. For information about fractional seconds support in MySQL, see Section 11.3.5, “Fractional Seconds in Time Values”. In particular, any fractional part in a value inserted into a `TIME` column is stored rather than discarded. With the fractional part included, the range for `TIME` values is `'-838:59:59.000000'` to `'838:59:59.000000'`.

Be careful about assigning abbreviated values to a `TIME` column. MySQL interprets abbreviated `TIME` values with colons as time of the day. That is, `'11:12'` means `'11:12:00'`, not `'00:11:12'`. MySQL interprets abbreviated values without colons using the assumption that the two rightmost digits represent seconds (that is, as elapsed time rather than as time of day). For example, you might think of `'1112'` and `1112` as meaning `'11:12:00'` (12 minutes after 11 o'clock), but MySQL interprets them as `'00:11:12'` (11 minutes, 12 seconds). Similarly, `'12'` and `12` are interpreted as `'00:00:12'`.

The only delimiter recognized between a time part and a fractional seconds part is the decimal point.

By default, values that lie outside the `TIME` range but are otherwise valid are clipped to the closest endpoint of the range. For example, `'-850:00:00'` and `'850:00:00'` are converted to `'-838:59:59'` and `'838:59:59'`. Invalid `TIME` values are converted to `'00:00:00'`. Note that because `'00:00:00'` is itself a valid `TIME` value, there is no way to tell, from a value of `'00:00:00'` stored in a table, whether the original value was specified as `'00:00:00'` or whether it was invalid.

For more restrictive treatment of invalid `TIME` values, enable strict SQL mode to cause errors to occur. See Section 5.1.11, “Server SQL Modes”.

11.3.3 The YEAR Type

The `YEAR` type is a 1-byte type used to represent year values. It can be declared as `YEAR` or `YEAR(4)` and has a display width of four characters.

Note

MySQL 8.0 does not support the `YEAR(2)` data type permitted in older versions of MySQL. For instructions on converting to `YEAR(4)`, see YEAR(2) Limitations and Migrating to YEAR(4) in MySQL 5.7 Reference Manual.

MySQL displays `YEAR` values in `YYYY` format, with a range of `1901` to `2155`, or `0000`.

You can specify input `YEAR` values in a variety of formats:

- As a 4-digit number in the range `1901` to `2155`.
- As a 4-digit string in the range `'1901'` to `'2155'`.
- As a 1- or 2-digit number in the range `1` to `99`. MySQL converts values in the ranges `1` to `69` and `70` to `99` to `YEAR` values in the ranges `2001` to `2069` and `1970` to `1999`.
- As a 1- or 2-digit string in the range `'0'` to `'99'`. MySQL converts values in the ranges `'0'` to `'69'` and `'70'` to `'99'` to `YEAR` values in the ranges `2000` to `2069` and `1970` to `1999`.

- The result of inserting a numeric `0` has a display value of `0000` and an internal value of `0000`. To insert zero and have it be interpreted as `2000`, specify it as a string `'0'` or `'00'`.
- As the result of a function that returns a value that is acceptable in a `YEAR` context, such as `NOW()`.

MySQL converts invalid `YEAR` values to `0000`.

See also Section 11.3.7, “Two-Digit Years in Dates”.

11.3.4 Automatic Initialization and Updating for TIMESTAMP and DATETIME

`TIMESTAMP` and `DATETIME` columns can be automatically initializated and updated to the current date and time (that is, the current timestamp).

For any `TIMESTAMP` or `DATETIME` column in a table, you can assign the current timestamp as the default value, the auto-update value, or both:

- An auto-initialized column is set to the current timestamp for inserted rows that specify no value for the column.
- An auto-updated column is automatically updated to the current timestamp when the value of any other column in the row is changed from its current value. An auto-updated column remains unchanged if all other columns are set to their current values. To prevent an auto-updated column from updating when other columns change, explicitly set it to its current value. To update an auto-updated column even when other columns do not change, explicitly set it to the value it should have (for example, set it to `CURRENT_TIMESTAMP`).

In addition, if the `explicit_defaults_for_timestamp` system variable is disabled, you can initialize or update any `TIMESTAMP` (but not `DATETIME`) column to the current date and time by assigning it a `NULL` value, unless it has been defined with the `NULL` attribute to permit `NULL` values.

To specify automatic properties, use the `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses in column definitions. The order of the clauses does not matter. If both are present in a column definition, either can occur first. Any of the synonyms for `CURRENT_TIMESTAMP` have the same meaning as `CURRENT_TIMESTAMP`. These are `CURRENT_TIMESTAMP()`, `NOW()`, `LOCALTIME`, `LOCALTIME()`, `LOCALTIMESTAMP`, and `LOCALTIMESTAMP()`.

Use of `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` is specific to `TIMESTAMP` and `DATETIME`. The `DEFAULT` clause also can be used to specify a constant (nonautomatic) default value (for example, `DEFAULT 0` or `DEFAULT '2000-01-01 00:00:00'`).

Note

The following examples use `DEFAULT 0`, a default that can produce warnings or errors depending on whether strict SQL mode or the `NO_ZERO_DATE` SQL mode is enabled. Be aware that the `TRADITIONAL` SQL mode includes strict mode and `NO_ZERO_DATE`. See Section 5.1.11, “Server SQL Modes”.

`TIMESTAMP` or `DATETIME` column definitions can specify the current timestamp for both the default and auto-update values, for one but not the other, or for neither. Different columns can have different combinations of automatic properties. The following rules describe the possibilities:

- With both `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP`, the column has the current timestamp for its default value and is automatically updated to the current timestamp.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
```

```
  dt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

- With a `DEFAULT` clause but no `ON UPDATE CURRENT_TIMESTAMP` clause, the column has the given default value and is not automatically updated to the current timestamp.

  The default depends on whether the `DEFAULT` clause specifies `CURRENT_TIMESTAMP` or a constant value. With `CURRENT_TIMESTAMP`, the default is the current timestamp.

  ```
  CREATE TABLE t1 (
    ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    dt DATETIME DEFAULT CURRENT_TIMESTAMP
  );
  ```

  With a constant, the default is the given value. In this case, the column has no automatic properties at all.

  ```
  CREATE TABLE t1 (
    ts TIMESTAMP DEFAULT 0,
    dt DATETIME DEFAULT 0
  );
  ```

- With an `ON UPDATE CURRENT_TIMESTAMP` clause and a constant `DEFAULT` clause, the column is automatically updated to the current timestamp and has the given constant default value.

  ```
  CREATE TABLE t1 (
    ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP,
    dt DATETIME DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP
  );
  ```

- With an `ON UPDATE CURRENT_TIMESTAMP` clause but no `DEFAULT` clause, the column is automatically updated to the current timestamp but does not have the current timestamp for its default value.

  The default in this case is type dependent. `TIMESTAMP` has a default of 0 unless defined with the `NULL` attribute, in which case the default is `NULL`.

  ```
  CREATE TABLE t1 (
    ts1 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,     -- default 0
    ts2 TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP -- default NULL
  );
  ```

  `DATETIME` has a default of `NULL` unless defined with the `NOT NULL` attribute, in which case the default is 0.

  ```
  CREATE TABLE t1 (
    dt1 DATETIME ON UPDATE CURRENT_TIMESTAMP,         -- default NULL
    dt2 DATETIME NOT NULL ON UPDATE CURRENT_TIMESTAMP -- default 0
  );
  ```

`TIMESTAMP` and `DATETIME` columns have no automatic properties unless they are specified explicitly, with this exception: If the `explicit_defaults_for_timestamp` system variable is disabled, the *first* `TIMESTAMP` column has both `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` if neither is specified explicitly. To suppress automatic properties for the first `TIMESTAMP` column, use one of these strategies:

- Enable the `explicit_defaults_for_timestamp` system variable. In this case, the `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses that specify automatic

initialization and updating are available, but are not assigned to any `TIMESTAMP` column unless explicitly included in the column definition.

- Alternatively, if `explicit_defaults_for_timestamp` is disabled, do either of the following:
  - Define the column with a `DEFAULT` clause that specifies a constant default value.
  - Specify the `NULL` attribute. This also causes the column to permit `NULL` values, which means that you cannot assign the current timestamp by setting the column to `NULL`. Assigning `NULL` sets the column to `NULL`, not the current timestamp. To assign the current timestamp, set the column to `CURRENT_TIMESTAMP` or a synonym such as `NOW()`.

Consider these table definitions:

```
CREATE TABLE t1 (
  ts1 TIMESTAMP DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t2 (
  ts1 TIMESTAMP NULL,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t3 (
  ts1 TIMESTAMP NULL DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
```

The tables have these properties:

- In each table definition, the first `TIMESTAMP` column has no automatic initialization or updating.
- The tables differ in how the `ts1` column handles `NULL` values. For `t1`, `ts1` is `NOT NULL` and assigning it a value of `NULL` sets it to the current timestamp. For `t2` and `t3`, `ts1` permits `NULL` and assigning it a value of `NULL` sets it to `NULL`.
- `t2` and `t3` differ in the default value for `ts1`. For `t2`, `ts1` is defined to permit `NULL`, so the default is also `NULL` in the absence of an explicit `DEFAULT` clause. For `t3`, `ts1` permits `NULL` but has an explicit default of 0.

If a `TIMESTAMP` or `DATETIME` column definition includes an explicit fractional seconds precision value anywhere, the same value must be used throughout the column definition. This is permitted:

```
CREATE TABLE t1 (
  ts TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP(6) ON UPDATE CURRENT_TIMESTAMP(6)
);
```

This is not permitted:

```
CREATE TABLE t1 (
  ts TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP(3)
);
```

TIMESTAMP Initialization and the NULL Attribute

If the `explicit_defaults_for_timestamp` system variable is disabled, `TIMESTAMP` columns by default are `NOT NULL`, cannot contain `NULL` values, and assigning `NULL` assigns the current timestamp. To permit a `TIMESTAMP` column to contain `NULL`, explicitly declare it with the `NULL` attribute. In this

case, the default value also becomes NULL unless overridden with a DEFAULT clause that specifies a different default value. DEFAULT NULL can be used to explicitly specify NULL as the default value. (For a TIMESTAMP column not declared with the NULL attribute, DEFAULT NULL is invalid.) If a TIMESTAMP column permits NULL values, assigning NULL sets it to NULL, not to the current timestamp.

The following table contains several TIMESTAMP columns that permit NULL values:

```
CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
```

A TIMESTAMP column that permits NULL values does *not* take on the current timestamp at insert time except under one of the following conditions:

- Its default value is defined as CURRENT_TIMESTAMP and no value is specified for the column
- CURRENT_TIMESTAMP or any of its synonyms such as NOW() is explicitly inserted into the column

In other words, a TIMESTAMP column defined to permit NULL values auto-initializes only if its definition includes DEFAULT CURRENT_TIMESTAMP:

```
CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP);
```

If the TIMESTAMP column permits NULL values but its definition does not include DEFAULT CURRENT_TIMESTAMP, you must explicitly insert a value corresponding to the current date and time. Suppose that tables t1 and t2 have these definitions:

```
CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');
CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT NULL);
```

To set the TIMESTAMP column in either table to the current timestamp at insert time, explicitly assign it that value. For example:

```
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
INSERT INTO t1 VALUES (NOW());
```

If the explicit_defaults_for_timestamp system variable is enabled, TIMESTAMP columns permit NULL values only if declared with the NULL attribute. Also, TIMESTAMP columns do not permit assigning NULL to assign the current timestamp, whether declared with the NULL or NOT NULL attribute. To assign the current timestamp, set the column to CURRENT_TIMESTAMP or a synonym such as NOW().

11.3.5 Fractional Seconds in Time Values

MySQL 8.0 has fractional seconds support for TIME, DATETIME, and TIMESTAMP values, with up to microseconds (6 digits) precision:

- To define a column that includes a fractional seconds part, use the syntax *type_name*(*fsp*), where *type_name* is TIME, DATETIME, or TIMESTAMP, and *fsp* is the fractional seconds precision. For example:

  ```
  CREATE TABLE t1 (t TIME(3), dt DATETIME(6));
  ```

The *fsp* value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)

- Inserting a TIME, DATE, or TIMESTAMP value with a fractional seconds part into a column of the same type but having fewer fractional digits results in rounding. Consider a table created and populated as follows:

```
CREATE TABLE fractest( c1 TIME(2), c2 DATETIME(2), c3 TIMESTAMP(2) );
INSERT INTO fractest VALUES
('17:51:04.777', '2018-09-08 17:51:04.777', '2018-09-08 17:51:04.777');
```

The temporal values are inserted into the table with rounding:

```
mysql> SELECT * FROM fractest;
+-------------+------------------------+------------------------+
| c1          | c2                     | c3                     |
+-------------+------------------------+------------------------+
| 17:51:04.78 | 2018-09-08 17:51:04.78 | 2018-09-08 17:51:04.78 |
+-------------+------------------------+------------------------+
```

No warning or error is given when such rounding occurs. This behavior follows the SQL standard.

To insert the values with truncation instead, enable the TIME_TRUNCATE_FRACTIONAL SQL mode:

```
SET @@sql_mode = sys.list_add(@@sql_mode, 'TIME_TRUNCATE_FRACTIONAL');
```

With that SQL mode enabled, the temporal values are inserted with truncation:

```
mysql> SELECT * FROM fractest;
+-------------+------------------------+------------------------+
| c1          | c2                     | c3                     |
+-------------+------------------------+------------------------+
| 17:51:04.77 | 2018-09-08 17:51:04.77 | 2018-09-08 17:51:04.77 |
+-------------+------------------------+------------------------+
```

- Functions that take temporal arguments accept values with fractional seconds. Return values from temporal functions include fractional seconds as appropriate. For example, NOW() with no argument returns the current date and time with no fractional part, but takes an optional argument from 0 to 6 to specify that the return value includes a fractional seconds part of that many digits.
- Syntax for temporal literals produces temporal values: DATE '*str*', TIME '*str*', and TIMESTAMP '*str*', and the ODBC-syntax equivalents. The resulting value includes a trailing fractional seconds part if specified. Previously, the temporal type keyword was ignored and these constructs produced the string value. See Standard SQL and ODBC Date and Time Literals

11.3.6 Conversion Between Date and Time Types

To some extent, you can convert a value from one temporal type to another. However, there may be some alteration of the value or loss of information. In all cases, conversion between temporal types is subject to the range of valid values for the resulting type. For example, although DATE, DATETIME, and TIMESTAMP values all can be specified using the same set of formats, the types do not all have the same range of values. TIMESTAMP values cannot be earlier than 1970 UTC or later than '2038-01-19 03:14:07' UTC. This means that a date such as '1968-01-01', while valid as a DATE or DATETIME value, is not valid as a TIMESTAMP value and is converted to 0.

Conversion of DATE values:

- Conversion to a DATETIME or TIMESTAMP value adds a time part of '00:00:00' because the DATE value contains no time information.
- Conversion to a TIME value is not useful; the result is '00:00:00'.

Conversion of DATETIME and TIMESTAMP values:

- Conversion to a DATE value takes fractional seconds into account and rounds the time part. For example, '1999-12-31 23:59:59.499' becomes '1999-12-31', whereas '1999-12-31 23:59:59.500' becomes '2000-01-01'.
- Conversion to a TIME value discards the date part because the TIME type contains no date information.

For conversion of TIME values to other temporal types, the value of CURRENT_DATE() is used for the date part. The TIME is interpreted as elapsed time (not time of day) and added to the date. This means that the date part of the result differs from the current date if the time value is outside the range from '00:00:00' to '23:59:59'.

Suppose that the current date is '2012-01-01'. TIME values of '12:00:00', '24:00:00', and '-12:00:00', when converted to DATETIME or TIMESTAMP values, result in '2012-01-01 12:00:00', '2012-01-02 00:00:00', and '2011-12-31 12:00:00', respectively.

Conversion of TIME to DATE is similar but discards the time part from the result: '2012-01-01', '2012-01-02', and '2011-12-31', respectively.

Explicit conversion can be used to override implicit conversion. For example, in comparison of DATE and DATETIME values, the DATE value is coerced to the DATETIME type by adding a time part of '00:00:00'. To perform the comparison by ignoring the time part of the DATETIME value instead, use the CAST() function in the following way:

```
date_col = CAST(datetime_col AS DATE)
```

Conversion of TIME and DATETIME values to numeric form (for example, by adding +0) depends on whether the value contains a fractional seconds part. TIME(N) or DATETIME(N) is converted to integer when N is 0 (or omitted) and to a DECIMAL value with N decimal digits when N is greater than 0:

```
mysql> SELECT CURTIME(), CURTIME()+0, CURTIME(3)+0;
+-----------+-------------+--------------+
| CURTIME() | CURTIME()+0 | CURTIME(3)+0 |
+-----------+-------------+--------------+
| 09:28:00  |       92800 |    92800.887 |
+-----------+-------------+--------------+
mysql> SELECT NOW(), NOW()+0, NOW(3)+0;
+---------------------+----------------+--------------------+
| NOW()               | NOW()+0        | NOW(3)+0           |
+---------------------+----------------+--------------------+
| 2012-08-15 09:28:00 | 20120815092800 | 20120815092800.889 |
+---------------------+----------------+--------------------+
```

11.3.7 Two-Digit Years in Dates

Date values with two-digit years are ambiguous because the century is unknown. Such values must be interpreted into four-digit form because MySQL stores years internally using four digits.

For DATETIME, DATE, and TIMESTAMP types, MySQL interprets dates specified with ambiguous year values using these rules:

- Year values in the range `00-69` are converted to `2000-2069`.
- Year values in the range `70-99` are converted to `1970-1999`.

For `YEAR`, the rules are the same, with this exception: A numeric `00` inserted into `YEAR(4)` results in `0000` rather than `2000`. To specify zero for `YEAR(4)` and have it be interpreted as `2000`, specify it as a string `'0'` or `'00'`.

Remember that these rules are only heuristics that provide reasonable guesses as to what your data values mean. If the rules used by MySQL do not produce the values you require, you must provide unambiguous input containing four-digit year values.

`ORDER BY` properly sorts `YEAR` values that have two-digit years.

Some functions like `MIN()` and `MAX()` convert a `YEAR` to a number. This means that a value with a two-digit year does not work properly with these functions. The fix in this case is to convert the `YEAR` to four-digit year format.

11.4 String Types

The string types are `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `BLOB`, `TEXT`, `ENUM`, and `SET`. This section describes how these types work and how to use them in your queries. For string type storage requirements, see Section 11.8, “Data Type Storage Requirements”.

11.4.1 The CHAR and VARCHAR Types

The `CHAR` and `VARCHAR` types are similar, but differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained.

The `CHAR` and `VARCHAR` types are declared with a length that indicates the maximum number of characters you want to store. For example, `CHAR(30)` can hold up to 30 characters.

The length of a `CHAR` column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When `CHAR` values are stored, they are right-padded with spaces to the specified length. When `CHAR` values are retrieved, trailing spaces are removed unless the `PAD_CHAR_TO_FULL_LENGTH` SQL mode is enabled.

Values in `VARCHAR` columns are variable-length strings. The length can be specified as a value from 0 to 65,535. The effective maximum length of a `VARCHAR` is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. See Section 8.4.7, “Limits on Table Column Count and Row Size”.

In contrast to `CHAR`, `VARCHAR` values are stored as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

If strict SQL mode is not enabled and you assign a value to a `CHAR` or `VARCHAR` column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.11, “Server SQL Modes”.

For `VARCHAR` columns, trailing spaces in excess of the column length are truncated prior to insertion and a warning is generated, regardless of the SQL mode in use. For `CHAR` columns, truncation of excess trailing spaces from inserted values is performed silently regardless of the SQL mode.

`VARCHAR` values are not padded when they are stored. Trailing spaces are retained when values are stored and retrieved, in conformance with standard SQL.

The following table illustrates the differences between CHAR and VARCHAR by showing the result of storing various string values into CHAR(4) and VARCHAR(4) columns (assuming that the column uses a single-byte character set such as latin1).

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

The values shown as stored in the last row of the table apply *only when not using strict mode*; if MySQL is running in strict mode, values that exceed the column length are *not stored*, and an error results.

InnoDB encodes fixed-length fields greater than or equal to 768 bytes in length as variable-length fields, which can be stored off-page. For example, a CHAR(255) column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with utf8mb4.

If a given value is stored into the CHAR(4) and VARCHAR(4) columns, the values retrieved from the columns are not always the same because trailing spaces are removed from CHAR columns upon retrieval. The following example illustrates this difference:

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO vc VALUES ('ab  ', 'ab  ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT CONCAT('(', v, ')'), CONCAT('(', c, ')') FROM vc;
+---------------------+---------------------+
| CONCAT('(', v, ')') | CONCAT('(', c, ')') |
+---------------------+---------------------+
| (ab  )              | (ab)                |
+---------------------+---------------------+
1 row in set (0.06 sec)
```

Values in CHAR and VARCHAR columns are sorted and compared according to the character set collation assigned to the column.

Most MySQL collations have a pad attribute of PAD SPACE. The exceptions are Unicode collations based on UCA 9.0.0 and higher, which have a pad attribute of NO PAD. (see Section 10.10.1, “Unicode Character Sets”).

To determine the pad attribute for a collation, use the INFORMATION_SCHEMA COLLATIONS table, which has a PAD_ATTRIBUTE column.

The pad attribute determines how trailing spaces are treated for comparison of nonbinary strings (CHAR, VARCHAR, and TEXT values). NO PAD collations treat spaces at the end of strings like any other character. For PAD SPACE collations, trailing spaces are insignificant in comparisons; strings are compared without regard to any trailing spaces. “Comparison” in this context does not include the LIKE pattern-matching operator, for which trailing spaces are significant. For example:

```
mysql> CREATE TABLE names (myname CHAR(10));
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO names VALUES ('Jones');
```

```
Query OK, 1 row affected (0.00 sec)

mysql> SELECT myname = 'Jones', myname = 'Jones  ' FROM names;
+--------------------+--------------------+
| myname = 'Jones'   | myname = 'Jones  ' |
+--------------------+--------------------+
|                  1 |                  1 |
+--------------------+--------------------+
1 row in set (0.00 sec)

mysql> SELECT myname LIKE 'Jones', myname LIKE 'Jones  ' FROM names;
+----------------------+----------------------+
| myname LIKE 'Jones'  | myname LIKE 'Jones  ' |
+----------------------+----------------------+
|                    1 |                    0 |
+----------------------+----------------------+
1 row in set (0.00 sec)
```

This is true for all MySQL versions, and is not affected by the server SQL mode.

Note

For more information about MySQL character sets and collations, see Chapter 10, *Character Sets, Collations, Unicode*. For additional information about storage requirements, see Section 11.8, “Data Type Storage Requirements”.

For those cases where trailing pad characters are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad characters will result in a duplicate-key error. For example, if a table contains `'a'`, an attempt to store `'a '` causes a duplicate-key error.

11.4.2 The BINARY and VARBINARY Types

The `BINARY` and `VARBINARY` types are similar to `CHAR` and `VARCHAR`, except that they contain binary strings rather than nonbinary strings. That is, they contain byte strings rather than character strings. This means they have the `binary` character set and collation, and comparison and sorting are based on the numeric values of the bytes in the values.

The permissible maximum length is the same for `BINARY` and `VARBINARY` as it is for `CHAR` and `VARCHAR`, except that the length for `BINARY` and `VARBINARY` is a length in bytes rather than in characters.

The `BINARY` and `VARBINARY` data types are distinct from the `CHAR BINARY` and `VARCHAR BINARY` data types. For the latter types, the `BINARY` attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary (`_bin`) collation for the column character set to be used, and the column itself contains nonbinary character strings rather than binary byte strings. For example, `CHAR(5) BINARY` is treated as `CHAR(5) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin`, assuming that the default character set is `utf8mb4`. This differs from `BINARY(5)`, which stores 5-bytes binary strings that have the `binary` character set and collation. For information about differences between binary strings and binary collations for nonbinary strings, see Section 10.8.5, “The binary Collation Compared to _bin Collations”.

If strict SQL mode is not enabled and you assign a value to a `BINARY` or `VARBINARY` column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For cases of truncation, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.11, “Server SQL Modes”.

When `BINARY` values are stored, they are right-padded with the pad value to the specified length. The pad value is `0x00` (the zero byte). Values are right-padded with `0x00` on insert, and no trailing bytes

are removed on select. All bytes are significant in comparisons, including `ORDER BY` and `DISTINCT` operations. `0x00` bytes and spaces are different in comparisons, with `0x00` < space.

Example: For a `BINARY(3)` column, `'a '` becomes `'a \0'` when inserted. `'a\0'` becomes `'a\0\0'` when inserted. Both inserted values remain unchanged when selected.

For `VARBINARY`, there is no padding on insert and no bytes are stripped on select. All bytes are significant in comparisons, including `ORDER BY` and `DISTINCT` operations. `0x00` bytes and spaces are different in comparisons, with `0x00` < space.

For those cases where trailing pad bytes are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad bytes will result in a duplicate-key error. For example, if a table contains `'a'`, an attempt to store `'a\0'` causes a duplicate-key error.

You should consider the preceding padding and stripping characteristics carefully if you plan to use the `BINARY` data type for storing binary data and you require that the value retrieved be exactly the same as the value stored. The following example illustrates how `0x00`-padding of `BINARY` values affects column value comparisons:

```
mysql> CREATE TABLE t (c BINARY(3));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET c = 'a';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a\0\0' from t;
+--------+---------+-------------+
| HEX(c) | c = 'a' | c = 'a\0\0' |
+--------+---------+-------------+
| 610000 |       0 |           1 |
+--------+---------+-------------+
1 row in set (0.09 sec)
```

If the value retrieved must be the same as the value specified for storage with no padding, it might be preferable to use `VARBINARY` or one of the `BLOB` data types instead.

11.4.3 The BLOB and TEXT Types

A `BLOB` is a binary large object that can hold a variable amount of data. The four `BLOB` types are `TINYBLOB`, `BLOB`, `MEDIUMBLOB`, and `LONGBLOB`. These differ only in the maximum length of the values they can hold. The four `TEXT` types are `TINYTEXT`, `TEXT`, `MEDIUMTEXT`, and `LONGTEXT`. These correspond to the four `BLOB` types and have the same maximum lengths and storage requirements. See Section 11.8, “Data Type Storage Requirements”.

`BLOB` values are treated as binary strings (byte strings). They have the `binary` character set and collation, and comparison and sorting are based on the numeric values of the bytes in column values. `TEXT` values are treated as nonbinary strings (character strings). They have a character set other than `binary`, and values are sorted and compared based on the collation of the character set.

If strict SQL mode is not enabled and you assign a value to a `BLOB` or `TEXT` column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.11, “Server SQL Modes”.

Truncation of excess trailing spaces from values to be inserted into `TEXT` columns always generates a warning, regardless of the SQL mode.

For TEXT and BLOB columns, there is no padding on insert and no bytes are stripped on select.

If a TEXT column is indexed, index entry comparisons are space-padded at the end. This means that, if the index requires unique values, duplicate-key errors will occur for values that differ only in the number of trailing spaces. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error. This is not true for BLOB columns.

In most respects, you can regard a BLOB column as a VARBINARY column that can be as large as you like. Similarly, you can regard a TEXT column as a VARCHAR column. BLOB and TEXT differ from VARBINARY and VARCHAR in the following ways:

- For indexes on BLOB and TEXT columns, you must specify an index prefix length. For CHAR and VARCHAR, a prefix length is optional. See Section 8.3.5, “Column Indexes”.
- BLOB and TEXT columns cannot have DEFAULT values.

If you use the BINARY attribute with a TEXT data type, the column is assigned the binary (_bin) collation of the column character set.

LONG and LONG VARCHAR map to the MEDIUMTEXT data type. This is a compatibility feature.

MySQL Connector/ODBC defines BLOB values as LONGVARBINARY and TEXT values as LONGVARCHAR.

Because BLOB and TEXT values can be extremely long, you might encounter some constraints in using them:

- Only the first max_sort_length bytes of the column are used when sorting. The default value of max_sort_length is 1024. You can make more bytes significant in sorting or grouping by increasing the value of max_sort_length at server startup or runtime. Any client can change the value of its session max_sort_length variable:

```
mysql> SET max_sort_length = 2000;
mysql> SELECT id, comment FROM t
    -> ORDER BY comment;
```

- Instances of BLOB or TEXT columns in the result of a query that is processed using a temporary table causes the server to use a table on disk rather than in memory because the MEMORY storage engine does not support those data types (see Section 8.4.4, “Internal Temporary Table Use in MySQL”). Use of disk incurs a performance penalty, so include BLOB or TEXT columns in the query result only if they are really needed. For example, avoid using SELECT *, which selects all columns.
- The maximum size of a BLOB or TEXT object is determined by its type, but the largest value you actually can transmit between the client and server is determined by the amount of available memory and the size of the communications buffers. You can change the message buffer size by changing the value of the max_allowed_packet variable, but you must do so for both the server and your client program. For example, both mysql and mysqldump enable you to change the client-side max_allowed_packet value. See Section 5.1.1, “Configuring the Server”, Section 4.5.1, “mysql — The MySQL Command-Line Client”, and Section 4.5.4, “mysqldump — A Database Backup Program”. You may also want to compare the packet sizes and the size of the data objects you are storing with the storage requirements, see Section 11.8, “Data Type Storage Requirements”

Each BLOB or TEXT value is represented internally by a separately allocated object. This is in contrast to all other data types, for which storage is allocated once per column when the table is opened.

In some cases, it may be desirable to store binary data such as media files in BLOB or TEXT columns. You may find MySQL's string handling functions useful for working with such data. See Section 12.5, “String Functions and Operators”. For security and other reasons, it is usually preferable to do so using

application code rather than giving application users the `FILE` privilege. You can discuss specifics for various languages and platforms in the MySQL Forums (http://forums.mysql.com/).

11.4.4 The ENUM Type

An `ENUM` is a string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation time. It has these advantages:

- Compact data storage in situations where a column has a limited set of possible values. The strings you specify as input values are automatically encoded as numbers. See Section 11.8, “Data Type Storage Requirements” for the storage requirements for `ENUM` types.
- Readable queries and output. The numbers are translated back to the corresponding strings in query results.

and these potential issues to consider:

- If you make enumeration values that look like numbers, it is easy to mix up the literal values with their internal index numbers, as explained in Enumeration Limitations.
- Using `ENUM` columns in `ORDER BY` clauses requires extra care, as explained in Enumeration Sorting.
- Creating and Using ENUM Columns
- Index Values for Enumeration Literals
- Handling of Enumeration Literals
- Empty or NULL Enumeration Values
- Enumeration Sorting
- Enumeration Limitations

Creating and Using ENUM Columns

An enumeration value must be a quoted string literal. For example, you can create a table with an `ENUM` column like this:

```
CREATE TABLE shirts (
    name VARCHAR(40),
    size ENUM('x-small', 'small', 'medium', 'large', 'x-large')
);
INSERT INTO shirts (name, size) VALUES ('dress shirt','large'), ('t-shirt','medium'),
  ('polo shirt','small');
SELECT name, size FROM shirts WHERE size = 'medium';
+---------+--------+
| name    | size   |
+---------+--------+
| t-shirt | medium |
+---------+--------+
UPDATE shirts SET size = 'small' WHERE size = 'large';
COMMIT;
```

Inserting 1 million rows into this table with a value of `'medium'` would require 1 million bytes of storage, as opposed to 6 million bytes if you stored the actual string `'medium'` in a `VARCHAR` column.

Index Values for Enumeration Literals

Each enumeration value has an index:

- The elements listed in the column specification are assigned index numbers, beginning with 1.
- The index value of the empty string error value is 0. This means that you can use the following `SELECT` statement to find rows into which invalid `ENUM` values were assigned:

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- The index of the `NULL` value is `NULL`.
- The term “index” here refers to a position within the list of enumeration values. It has nothing to do with table indexes.

For example, a column specified as `ENUM('Mercury', 'Venus', 'Earth')` can have any of the values shown here. The index of each value is also shown.

Value	Index
`NULL`	`NULL`
`''`	0
`'Mercury'`	1
`'Venus'`	2
`'Earth'`	3

An `ENUM` column can have a maximum of 65,535 distinct elements.

If you retrieve an `ENUM` value in a numeric context, the column value's index is returned. For example, you can retrieve numeric values from an `ENUM` column like this:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

Functions such as `SUM()` or `AVG()` that expect a numeric argument cast the argument to a number if necessary. For `ENUM` values, the index number is used in the calculation.

Handling of Enumeration Literals

Trailing spaces are automatically deleted from `ENUM` member values in the table definition when a table is created.

When retrieved, values stored into an `ENUM` column are displayed using the lettercase that was used in the column definition. Note that `ENUM` columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

If you store a number into an `ENUM` column, the number is treated as the index into the possible values, and the value stored is the enumeration member with that index. (However, this does *not* work with `LOAD DATA`, which treats all input as strings.) If the numeric value is quoted, it is still interpreted as an index if there is no matching string in the list of enumeration values. For these reasons, it is not advisable to define an `ENUM` column with enumeration values that look like numbers, because this can easily become confusing. For example, the following column has enumeration members with string values of `'0'`, `'1'`, and `'2'`, but numeric index values of `1`, `2`, and `3`:

```
numbers ENUM('0','1','2')
```

If you store `2`, it is interpreted as an index value, and becomes `'1'` (the value with index 2). If you store `'2'`, it matches an enumeration value, so it is stored as `'2'`. If you store `'3'`, it does not match any enumeration value, so it is treated as an index and becomes `'2'` (the value with index 3).

```
mysql> INSERT INTO t (numbers) VALUES(2),('2'),('3');
mysql> SELECT * FROM t;
+---------+
| numbers |
+---------+
| 1       |
| 2       |
| 2       |
+---------+
```

To determine all possible values for an `ENUM` column, use `SHOW COLUMNS FROM tbl_name LIKE 'enum_col'` and parse the `ENUM` definition in the `Type` column of the output.

In the C API, `ENUM` values are returned as strings. For information about using result set metadata to distinguish them from other strings, see Section 28.7.5, “C API Data Structures”.

Empty or NULL Enumeration Values

An enumeration value can also be the empty string (`''`) or `NULL` under certain circumstances:

- If you insert an invalid value into an `ENUM` (that is, a string not present in the list of permitted values), the empty string is inserted instead as a special error value. This string can be distinguished from a “normal” empty string by the fact that this string has the numeric value 0. See Index Values for Enumeration Literals for details about the numeric indexes for the enumeration values.

  If strict SQL mode is enabled, attempts to insert invalid `ENUM` values result in an error.

- If an `ENUM` column is declared to permit `NULL`, the `NULL` value is a valid value for the column, and the default value is `NULL`. If an `ENUM` column is declared `NOT NULL`, its default value is the first element of the list of permitted values.

Enumeration Sorting

`ENUM` values are sorted based on their index numbers, which depend on the order in which the enumeration members were listed in the column specification. For example, `'b'` sorts before `'a'` for `ENUM('b', 'a')`. The empty string sorts before nonempty strings, and `NULL` values sort before all other enumeration values.

To prevent unexpected results when using the `ORDER BY` clause on an `ENUM` column, use one of these techniques:

- Specify the `ENUM` list in alphabetic order.
- Make sure that the column is sorted lexically rather than by index number by coding `ORDER BY CAST(col AS CHAR)` or `ORDER BY CONCAT(col)`.

Enumeration Limitations

An enumeration value cannot be an expression, even one that evaluates to a string value.

For example, this `CREATE TABLE` statement does *not* work because the `CONCAT` function cannot be used to construct an enumeration value:

```
CREATE TABLE sizes (
    size ENUM('small', CONCAT('med','ium'), 'large')
);
```

You also cannot employ a user variable as an enumeration value. This pair of statements do *not* work:

```
SET @mysize = 'medium';

CREATE TABLE sizes (
    size ENUM('small', @mysize, 'large')
);
```

We strongly recommend that you do *not* use numbers as enumeration values, because it does not save on storage over the appropriate `TINYINT` or `SMALLINT` type, and it is easy to mix up the strings and the underlying number values (which might not be the same) if you quote the `ENUM` values incorrectly. If you do use a number as an enumeration value, always enclose it in quotation marks. If the quotation marks are omitted, the number is regarded as an index. See Handling of Enumeration Literals to see how even a quoted number could be mistakenly used as a numeric index value.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

11.4.5 The SET Type

A `SET` is a string object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created. `SET` column values that consist of multiple set members are specified with members separated by commas (`,`). A consequence of this is that `SET` member values should not themselves contain commas.

For example, a column specified as `SET('one', 'two') NOT NULL` can have any of these values:

```
''
'one'
'two'
'one,two'
```

A `SET` column can have a maximum of 64 distinct members.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

Trailing spaces are automatically deleted from `SET` member values in the table definition when a table is created.

When retrieved, values stored in a `SET` column are displayed using the lettercase that was used in the column definition. Note that `SET` columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

MySQL stores `SET` values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a `SET` value in a numeric context, the value retrieved has bits set corresponding to the set members that make up the column value. For example, you can retrieve numeric values from a `SET` column like this:

```
mysql> SELECT set_col+0 FROM tbl_name;
```

If a number is stored into a `SET` column, the bits that are set in the binary representation of the number determine the set members in the column value. For a column specified as `SET('a','b','c','d')`, the members have the following decimal and binary values.

SET Member	Decimal Value	Binary Value
`'a'`	`1`	`0001`

SET Member	Decimal Value	Binary Value
'b'	2	0010
'c'	4	0100
'd'	8	1000

If you assign a value of 9 to this column, that is 1001 in binary, so the first and fourth SET value members 'a' and 'd' are selected and the resulting value is 'a,d'.

For a value containing more than one SET element, it does not matter what order the elements are listed in when you insert the value. It also does not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value appears once, with elements listed according to the order in which they were specified at table creation time. Suppose that a column is specified as SET('a','b','c','d'):

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

If you insert the values 'a,d', 'd,a', 'a,d,d', 'a,d,a', and 'd,a,d':

```
mysql> INSERT INTO myset (col) VALUES
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Then all these values appear as 'a,d' when retrieved:

```
mysql> SELECT col FROM myset;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
5 rows in set (0.04 sec)
```

If you set a SET column to an unsupported value, the value is ignored and a warning is issued:

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;
+---------+------+-------------------------------------------+
| Level   | Code | Message                                   |
+---------+------+-------------------------------------------+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+---------+------+-------------------------------------------+
1 row in set (0.04 sec)

mysql> SELECT col FROM myset;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
```

```
| a,d  |
| a,d  |
+------+
6 rows in set (0.01 sec)
```

If strict SQL mode is enabled, attempts to insert invalid `SET` values result in an error.

`SET` values are sorted numerically. `NULL` values sort before non-`NULL` `SET` values.

Functions such as `SUM()` or `AVG()` that expect a numeric argument cast the argument to a number if necessary. For `SET` values, the cast operation causes the numeric value to be used.

Normally, you search for `SET` values using the `FIND_IN_SET()` function or the `LIKE` operator:

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

The first statement finds rows where *`set_col`* contains the *`value`* set member. The second is similar, but not the same: It finds rows where *`set_col`* contains *`value`* anywhere, even as a substring of another set member.

The following statements also are permitted:

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

The first of these statements looks for values containing the first set member. The second looks for an exact match. Be careful with comparisons of the second type. Comparing set values to *`'val1,val2'`* returns different results than comparing values to *`'val2,val1'`*. You should specify the values in the same order they are listed in the column definition.

To determine all possible values for a `SET` column, use `SHOW COLUMNS FROM` *`tbl_name`* `LIKE` *`set_col`* and parse the `SET` definition in the `Type` column of the output.

In the C API, `SET` values are returned as strings. For information about using result set metadata to distinguish them from other strings, see Section 28.7.5, “C API Data Structures”.

11.5 Spatial Data Types

The Open Geospatial Consortium (OGC) is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data.

The Open Geospatial Consortium publishes the *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC website at http://www.opengeospatial.org/standards/sfs.

Following the OGC specification, MySQL implements spatial extensions as a subset of the **SQL with Geometry Types** environment. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specification describes a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

MySQL spatial extensions enable the generation, storage, and analysis of geographic features:

- Data types for representing spatial values

- Functions for manipulating spatial values
- Spatial indexing for improved access times to spatial columns

The spatial data types and functions are available for `MyISAM`, `InnoDB`, `NDB`, and `ARCHIVE` tables. For indexing spatial columns, `MyISAM` and `InnoDB` support both `SPATIAL` and non-`SPATIAL` indexes. The other storage engines support non-`SPATIAL` indexes, as described in Section 13.1.15, “CREATE INDEX Syntax”.

A **geographic feature** is anything in the world that has a location. A feature can be:

- An entity. For example, a mountain, a pond, a city.
- A space. For example, town district, the tropics.
- A definable location. For example, a crossroad, as a particular place where two streets intersect.

Some documents use the term **geospatial feature** to refer to geographic features.

Geometry is another word that denotes a geographic feature. Originally the word **geometry** meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

The discussion here considers these terms synonymous: **geographic feature**, **geospatial feature**, **feature**, or **geometry**. The term most commonly used is **geometry**, defined as *a point or an aggregate of points representing anything in the world that has a location*.

The following material covers these topics:

- The spatial data types implemented in MySQL model
- The basis of the spatial extensions in the OpenGIS geometry model
- Data formats for representing spatial data
- How to use spatial data in MySQL
- Use of indexing for spatial data
- MySQL differences from the OpenGIS specification

For information about functions that operate on spatial data, see Section 12.16, “Spatial Analysis Functions”.

Additional Resources

These standards are important for the MySQL implementation of spatial operations:

- SQL/MM Part 3: Spatial.
- The Open Geospatial Consortium publishes the *OpenGIS® Implementation Standard for Geographic information*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. See in particular Simple Feature Access - Part 1: Common Architecture, and Simple Feature Access - Part 2: SQL Option. The Open Geospatial Consortium (OGC) maintains a website at http://www.opengeospatial.org/. The specification is available there at http://www.opengeospatial.org/standards/sfs. It contains additional information relevant to the material here.

- The grammar for spatial reference system (SRS) definitions is based on the grammar defined in *OpenGIS Implementation Specification: Coordinate Transformation Services*, Revision 1.00, OGC 01-009, January 12, 2001, Section 7.2. This specification is available at http://www.opengeospatial.org/standards/ct. For differences from that specification in SRS definitions as implemented in MySQL, see Section 13.1.19, “CREATE SPATIAL REFERENCE SYSTEM Syntax”.

If you have questions or concerns about the use of the spatial extensions to MySQL, you can discuss them in the GIS forum: https://forums.mysql.com/list.php?23.

11.5.1 Spatial Data Types

MySQL has spatial data types that correspond to OpenGIS classes. The basis for these types is described in Section 11.5.2, “The OpenGIS Geometry Model”.

Some spatial data types hold single geometry values:

- `GEOMETRY`
- `POINT`
- `LINESTRING`
- `POLYGON`

`GEOMETRY` can store geometry values of any type. The other single-value types (`POINT`, `LINESTRING`, and `POLYGON`) restrict their values to a particular geometry type.

The other spatial data types hold collections of values:

- `MULTIPOINT`
- `MULTILINESTRING`
- `MULTIPOLYGON`
- `GEOMETRYCOLLECTION`

`GEOMETRYCOLLECTION` can store a collection of objects of any type. The other collection types (`MULTIPOINT`, `MULTILINESTRING`, and `MULTIPOLYGON`) restrict collection members to those having a particular geometry type.

Example: To create a table named `geom` that has a column named `g` that can store values of any geometry type, use this statement:

```
CREATE TABLE geom (g GEOMETRY);
```

Columns with a spatial data type can have an `SRID` attribute, to explicitly indicate the spatial reference system (SRS) for values stored in the column. For example:

```
CREATE TABLE geom (
    p POINT SRID 0,
    g GEOMETRY NOT NULL SRID 4326
);
```

`SPATIAL` indexes can be created on spatial columns if they are `NOT NULL` and have a specific SRID, so if you plan to index the column, declare it with the `NOT NULL` and `SRID` attributes:

```
CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326);
```

`InnoDB` tables permit `SRID` values for Cartesian and geographic SRSs. `MyISAM` tables permit `SRID` values for Cartesian SRSs.

The `SRID` attribute makes a spatial column SRID-restricted, which has these implications:

- The column can contain only values with the given SRID. Attempts to insert values with a different SRID produce an error.
- The optimizer can use `SPATIAL` indexes on the column. See Section 8.3.3, “SPATIAL Index Optimization”.

Spatial columns with no `SRID` attribute are not SRID-restricted and accept values with any SRID. However, the optimizer cannot use `SPATIAL` indexes on them until the column definition is modified to include an `SRID` attribute, which may require that the column contents first be modified so that all values have the same SRID.

For other examples showing how to use spatial data types in MySQL, see Section 11.5.6, “Creating Spatial Columns”. For information about spatial reference systems, see Section 11.5.5, “Spatial Reference System Support”.

11.5.2 The OpenGIS Geometry Model

The set of geometry types proposed by OGC's **SQL with Geometry Types** environment is based on the **OpenGIS Geometry Model**. In this model, each geometric object has the following general properties:

- It is associated with a spatial reference system, which describes the coordinate space in which the object is defined.
- It belongs to some geometry class.

11.5.2.1 The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

- `Geometry` (noninstantiable)
  - `Point` (instantiable)
  - `Curve` (noninstantiable)
    - `LineString` (instantiable)
      - `Line`
      - `LinearRing`
  - `Surface` (noninstantiable)
    - `Polygon` (instantiable)
  - `GeometryCollection` (instantiable)
    - `MultiPoint` (instantiable)
    - `MultiCurve` (noninstantiable)

    - `MultiLineString` (instantiable)
  - `MultiSurface` (noninstantiable)
    - `MultiPolygon` (instantiable)

It is not possible to create objects in noninstantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

`Geometry` is the base class. It is an abstract class. The instantiable subclasses of `Geometry` are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base `Geometry` class has subclasses for `Point`, `Curve`, `Surface`, and `GeometryCollection`:

- `Point` represents zero-dimensional objects.
- `Curve` represents one-dimensional objects, and has subclass `LineString`, with sub-subclasses `Line` and `LinearRing`.
- `Surface` is designed for two-dimensional objects and has subclass `Polygon`.
- `GeometryCollection` has specialized zero-, one-, and two-dimensional collection classes named `MultiPoint`, `MultiLineString`, and `MultiPolygon` for modeling geometries corresponding to collections of `Points`, `LineStrings`, and `Polygons`, respectively. `MultiCurve` and `MultiSurface` are introduced as abstract superclasses that generalize the collection interfaces to handle `Curves` and `Surfaces`.

`Geometry`, `Curve`, `Surface`, `MultiCurve`, and `MultiSurface` are defined as noninstantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

`Point`, `LineString`, `Polygon`, `GeometryCollection`, `MultiPoint`, `MultiLineString`, and `MultiPolygon` are instantiable classes.

11.5.2.2 Geometry Class

`Geometry` is the root class of the hierarchy. It is a noninstantiable class but has a number of properties, described in the following list, that are common to all geometry values created from any of the `Geometry` subclasses. Particular subclasses have their own specific properties, described later.

Geometry Properties

A geometry value has the following properties:

- Its **type**. Each geometry belongs to one of the instantiable classes in the hierarchy.
- Its **SRID**, or spatial reference identifier. This value identifies the geometry's associated spatial reference system that describes the coordinate space in which the geometry object is defined.

  In MySQL, the SRID value is an integer associated with the geometry value. The maximum usable SRID value is $2^{32}-1$. If a larger value is given, only the lower 32 bits are used.

  SRID 0 represents an infinite flat Cartesian plane with no units assigned to its axes. To ensure SRID 0 behavior, create geometry values using SRID 0. SRID 0 is the default for new geometry values if no SRID is specified.

For computations on multiple geometry values, all values must have the same SRID or an error occurs.

- Its **coordinates** in its spatial reference system, represented as double-precision (8-byte) numbers. All nonempty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

  Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the **planar** coordinate system and the distance on the **geodetic** system (coordinates on the Earth's surface) are different things.

- Its **interior**, **boundary**, and **exterior**.

  Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.

- Its **MBR** (minimum bounding rectangle), or envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

  ```
  ((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
  ```

- Whether the value is **simple** or **nonsimple**. Geometry values of types (`LineString`, `MultiPoint`, `MultiLineString`) are either simple or nonsimple. Each type determines its own assertions for being simple or nonsimple.

- Whether the value is **closed** or **not closed**. Geometry values of types (`LineString`, `MultiString`) are either closed or not closed. Each type determines its own assertions for being closed or not closed.

- Whether the value is **empty** or **nonempty** A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a `NULL` value). An empty geometry is defined to be always simple and has an area of 0.

- Its **dimension**. A geometry can have a dimension of −1, 0, 1, or 2:

  - −1 for an empty geometry.
  - 0 for a geometry with no length and no area.
  - 1 for a geometry with nonzero length and zero area.
  - 2 for a geometry with nonzero area.

  `Point` objects have a dimension of zero. `LineString` objects have a dimension of 1. `Polygon` objects have a dimension of 2. The dimensions of `MultiPoint`, `MultiLineString`, and `MultiPolygon` objects are the same as the dimensions of the elements they consist of.

11.5.2.3 Point Class

A `Point` is a geometry that represents a single location in coordinate space.

`Point` Examples

- Imagine a large-scale map of the world with many cities. A `Point` object could represent each city.
- On a city map, a `Point` object could represent a bus stop.

`Point` Properties

- X-coordinate value.
- Y-coordinate value.
- `Point` is defined as a zero-dimensional geometry.
- The boundary of a `Point` is the empty set.

11.5.2.4 Curve Class

A `Curve` is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of `Curve` define the type of interpolation between points. `Curve` is a noninstantiable class.

`Curve` Properties

- A `Curve` has the coordinates of its points.
- A `Curve` is defined as a one-dimensional geometry.
- A `Curve` is simple if it does not pass through the same point twice, with the exception that a curve can still be simple if the start and end points are the same.
- A `Curve` is closed if its start point is equal to its endpoint.
- The boundary of a closed `Curve` is empty.
- The boundary of a nonclosed `Curve` consists of its two endpoints.
- A `Curve` that is simple and closed is a `LinearRing`.

11.5.2.5 LineString Class

A `LineString` is a `Curve` with linear interpolation between points.

`LineString` Examples

- On a world map, `LineString` objects could represent rivers.
- In a city map, `LineString` objects could represent streets.

`LineString` Properties

- A `LineString` has coordinates of segments, defined by each consecutive pair of points.
- A `LineString` is a `Line` if it consists of exactly two points.
- A `LineString` is a `LinearRing` if it is both closed and simple.

11.5.2.6 Surface Class

A `Surface` is a two-dimensional geometry. It is a noninstantiable class. Its only instantiable subclass is `Polygon`.

`Surface` Properties

- A `Surface` is defined as a two-dimensional geometry.

- The OpenGIS specification defines a simple `Surface` as a geometry that consists of a single “patch” that is associated with a single exterior boundary and zero or more interior boundaries.
- The boundary of a simple `Surface` is the set of closed curves corresponding to its exterior and interior boundaries.

11.5.2.7 Polygon Class

A `Polygon` is a planar `Surface` representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the `Polygon`.

`Polygon` Examples

- On a region map, `Polygon` objects could represent forests, districts, and so on.

`Polygon` Assertions

- The boundary of a `Polygon` consists of a set of `LinearRing` objects (that is, `LineString` objects that are both simple and closed) that make up its exterior and interior boundaries.
- A `Polygon` has no rings that cross. The rings in the boundary of a `Polygon` may intersect at a `Point`, but only as a tangent.
- A `Polygon` has no lines, spikes, or punctures.
- A `Polygon` has an interior that is a connected point set.
- A `Polygon` may have holes. The exterior of a `Polygon` with holes is not connected. Each hole defines a connected component of the exterior.

The preceding assertions make a `Polygon` a simple geometry.

11.5.2.8 GeometryCollection Class

A `GeomCollection` is a geometry that is a collection of zero or more geometries of any class.

`GeomCollection` and `GeometryCollection` are synonymous, with `GeomCollection` the preferred type name.

All the elements in a geometry collection must be in the same spatial reference system (that is, in the same coordinate system). There are no other constraints on the elements of a geometry collection, although the subclasses of `GeomCollection` described in the following sections may restrict membership. Restrictions may be based on:

- Element type (for example, a `MultiPoint` may contain only `Point` elements)
- Dimension
- Constraints on the degree of spatial overlap between elements

11.5.2.9 MultiPoint Class

A `MultiPoint` is a geometry collection composed of `Point` elements. The points are not connected or ordered in any way.

`MultiPoint` Examples

- On a world map, a `MultiPoint` could represent a chain of small islands.
- On a city map, a `MultiPoint` could represent the outlets for a ticket office.

`MultiPoint` Properties

- A `MultiPoint` is a zero-dimensional geometry.
- A `MultiPoint` is simple if no two of its `Point` values are equal (have identical coordinate values).
- The boundary of a `MultiPoint` is the empty set.

11.5.2.10 MultiCurve Class

A `MultiCurve` is a geometry collection composed of `Curve` elements. `MultiCurve` is a noninstantiable class.

`MultiCurve` Properties

- A `MultiCurve` is a one-dimensional geometry.
- A `MultiCurve` is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.
- A `MultiCurve` boundary is obtained by applying the “mod 2 union rule” (also known as the “odd-even rule”): A point is in the boundary of a `MultiCurve` if it is in the boundaries of an odd number of `Curve` elements.
- A `MultiCurve` is closed if all of its elements are closed.
- The boundary of a closed `MultiCurve` is always empty.

11.5.2.11 MultiLineString Class

A `MultiLineString` is a `MultiCurve` geometry collection composed of `LineString` elements.

`MultiLineString` Examples

- On a region map, a `MultiLineString` could represent a river system or a highway system.

11.5.2.12 MultiSurface Class

A `MultiSurface` is a geometry collection composed of surface elements. `MultiSurface` is a noninstantiable class. Its only instantiable subclass is `MultiPolygon`.

`MultiSurface` Assertions

- Surfaces within a `MultiSurface` have no interiors that intersect.
- Surfaces within a `MultiSurface` have boundaries that intersect at most at a finite number of points.

11.5.2.13 MultiPolygon Class

A `MultiPolygon` is a `MultiSurface` object composed of `Polygon` elements.

`MultiPolygon` Examples

- On a region map, a `MultiPolygon` could represent a system of lakes.

`MultiPolygon` Assertions

- A `MultiPolygon` has no two `Polygon` elements with interiors that intersect.
- A `MultiPolygon` has no two `Polygon` elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.
- A `MultiPolygon` may not have cut lines, spikes, or punctures. A `MultiPolygon` is a regular, closed point set.
- A `MultiPolygon` that has more than one `Polygon` has an interior that is not connected. The number of connected components of the interior of a `MultiPolygon` is equal to the number of `Polygon` values in the `MultiPolygon`.

`MultiPolygon` Properties

- A `MultiPolygon` is a two-dimensional geometry.
- A `MultiPolygon` boundary is a set of closed curves (`LineString` values) corresponding to the boundaries of its `Polygon` elements.
- Each `Curve` in the boundary of the `MultiPolygon` is in the boundary of exactly one `Polygon` element.
- Every `Curve` in the boundary of an `Polygon` element is in the boundary of the `MultiPolygon`.

11.5.3 Supported Spatial Data Formats

Two standard spatial data formats are used to represent geometry objects in queries:

- Well-Known Text (WKT) format
- Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format. (Internal format is like WKB but with an initial 4 bytes to indicate the SRID.)

There are functions available to convert between different data formats; see Section 12.16.6, “Geometry Format Conversion Functions”.

The following sections describe the spatial data formats MySQL uses:

- Well-Known Text (WKT) Format
- Well-Known Binary (WKB) Format
- Internal Geometry Storage Format

Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of geometry values is designed for exchanging geometry data in ASCII form. The OpenGIS specification provides a Backus-Naur grammar that specifies the formal production rules for writing WKT values (see Section 11.5, “Spatial Data Types”).

Examples of WKT representations of geometry objects:

- A `Point`:

```
POINT(15 20)
```

The point coordinates are specified with no separating comma. This differs from the syntax for the SQL `Point()` function, which requires a comma between the coordinates. Take care to use the syntax appropriate to the context of a given spatial operation. For example, the following statements both use `ST_X()` to extract the X-coordinate from a `Point` object. The first produces the object directly using the `Point()` function. The second uses a WKT representation converted to a `Point` with `ST_GeomFromText()`.

```
mysql> SELECT ST_X(Point(15, 20));
+--------------------+
| ST_X(POINT(15, 20)) |
+--------------------+
|                 15 |
+--------------------+

mysql> SELECT ST_X(ST_GeomFromText('POINT(15 20)'));
+---------------------------------------+
| ST_X(ST_GeomFromText('POINT(15 20)')) |
+---------------------------------------+
|                                    15 |
+---------------------------------------+
```

- A `LineString` with four points:

```
LINESTRING(0 0, 10 10, 20 25, 50 60)
```

The point coordinate pairs are separated by commas.

- A `Polygon` with one exterior ring and one interior ring:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- A `MultiPoint` with three `Point` values:

```
MULTIPOINT(0 0, 20 20, 60 60)
```

Spatial functions such as `ST_MPointFromText()` and `ST_GeomFromText()` that accept WKT-format representations of `MultiPoint` values permit individual points within values to be surrounded by parentheses. For example, both of the following function calls are valid:

```
ST_MPointFromText('MULTIPOINT (1 1, 2 2, 3 3)')
ST_MPointFromText('MULTIPOINT ((1 1), (2 2), (3 3))')
```

- A `MultiLineString` with two `LineString` values:

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- A `MultiPolygon` with two `Polygon` values:

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- A `GeometryCollection` consisting of two `Point` values and one `LineString`:

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation of geometric values is used for exchanging geometry data as binary streams represented by `BLOB` values containing geometric WKB information. This format is defined by the OpenGIS specification (see Section 11.5, “Spatial Data Types”). It is also defined in the ISO *SQL/MM Part 3: Spatial* standard.

WKB uses 1-byte unsigned integers, 4-byte unsigned integers, and 8-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to `POINT(1 -1)` consists of this sequence of 21 bytes, each represented by two hexadecimal digits:

```
0101000000000000000000F03F000000000000F0BF
```

The sequence consists of the components shown in the following table.

Table 11.2 WKB Components Example

Component	Size	Value
Byte order	1 byte	`01`
WKB type	4 bytes	`01000000`
X coordinate	8 bytes	`000000000000F03F`
Y coordinate	8 bytes	`000000000000F0BF`

Component representation is as follows:

- The byte order indicator is either 1 or 0 to signify little-endian or big-endian storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.
- The WKB type is a code that indicates the geometry type. MySQL uses values from 1 through 7 to indicate `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, and `GeometryCollection`.
- A `Point` value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values have more complex data structures, as detailed in the OpenGIS specification.

Internal Geometry Storage Format

MySQL stores geometry values using 4 bytes to indicate the SRID followed by the WKB representation of the value. For a description of WKB format, see Well-Known Binary (WKB) Format.

For the WKB part, these MySQL-specific considerations apply:

- The byte-order indicator byte is 1 because MySQL stores geometries as little-ending values.
- MySQL supports geometry types of `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, and `GeometryCollection`. Other geometry types are not supported.
- Only `GeometryCollection` can be empty. Such a value is stored with 0 elements.

- Polygon rings can be specified both clockwise and counterclockwise. MySQL flips the rings automatically when reading data.

Cartesian coordinates are stored in the length unit of the spatial reference system, with X values in the X coordinates and Y values in the Y coordinates. Axis directions are those specified by the spatial reference system.

Geographic coordinates are stored in the angle unit of the spatial reference system, with longitudes in the X coordinates and latitudes in the Y coordinates. Axis directions and the meridian are those specified by the spatial reference system.

The `LENGTH()` function returns the space in bytes required for value storage. Example:

```
mysql> SET @g = ST_GeomFromText('POINT(1 -1)');
mysql> SELECT LENGTH(@g);
+------------+
| LENGTH(@g) |
+------------+
|         25 |
+------------+
mysql> SELECT HEX(@g);
+----------------------------------------------------+
| HEX(@g)                                            |
+----------------------------------------------------+
| 000000000101000000000000000000F03F000000000000F0BF |
+----------------------------------------------------+
```

The value length is 25 bytes, made up of these components (as can be seen from the hexadecimal value):

- 4 bytes for integer SRID (0)
- 1 byte for integer byte order (1 = little-endian)
- 4 bytes for integer type information (1 = `Point`)
- 8 bytes for double-precision X coordinate (1)
- 8 bytes for double-precision Y coordinate (−1)

11.5.4 Geometry Well-Formedness and Validity

For geometry values, MySQL distinguishes between the concepts of syntactically well-formed and geometrically valid.

A geometry is syntactically well-formed if it satisfies conditions such as those in this (nonexhaustive) list:

- Linestrings have at least two points
- Polygons have at least one ring
- Polygon rings are closed (first and last points the same)
- Polygon rings have at least 4 points (minimum polygon is a triangle with first and last points the same)
- Collections are not empty (except `GeometryCollection`)

A geometry is geometrically valid if it is syntactically well-formed and satisfies conditions such as those in this (nonexhaustive) list:

- Polygons are not self-intersecting
- Polygon interior rings are inside the exterior ring
- Multipolygons do not have overlapping polygons

Spatial functions fail if a geometry is not syntactically well-formed. Spatial import functions that parse WKT or WKB values raise an error for attempts to create a geometry that is not syntactically well-formed. Syntactic well-formedness is also checked for attempts to store geometries into tables.

It is permitted to insert, select, and update geometrically invalid geometries, but they must be syntactically well-formed. Due to the computational expense, MySQL does not check explicitly for geometric validity. Spatial computations may detect some cases of invalid geometries and raise an error, but they may also return an undefined result without detecting the invalidity. Applications that require geometically valid geometries should check them using the `ST_IsValid()` function.

11.5.5 Spatial Reference System Support

A spatial reference system (SRS) for spatial data is a coordinate-based system for geographic locations.

There are different types of spatial reference systems:

- A projected SRS is a projection of a globe onto a flat surface; that is, a flat map. For example, a light bulb inside a globe that shines on a paper cylinder surrounding the globe projects a map onto the paper. The result is georeferenced: Each point maps to a place on the globe. The coordinate system on that plane is Cartesian using a length unit (meters, feet, and so forth), rather than degrees of longitude and latitude.

  The globes in this case are ellipsoids; that is, flattened spheres. Earth is a bit shorter in its North-South axis than its East-West axis, so a slightly flattened sphere is more correct, but perfect spheres permit faster calculations.

- A geographic SRS is a nonprojected SRS representing longitude-latitude (or latitude-longitude) coordinates on an ellipsoid, in any angular unit.

- The SRS denoted in MySQL by SRID 0 represents an infinite flat Cartesian plane with no units assigned to its axes. Unlike projected SRSs, it is not georeferenced and it does not necessarily represent Earth. It is an abstract plane that can be used for anything. SRID 0 is the default SRID for spatial data in MySQL.

MySQL maintains information about available spatial reference systems for spatial data in the data dictionary `mysql.st_spatial_reference_systems` table, which can store entries for projected and geographic SRSs. This data dictionary table is invisible, but SRS entry contents are available through the `INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS` table, implemented as a view on `mysql.st_spatial_reference_systems` (see Section 25.28, “The INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS Table”).

The following example shows what an SRS entry looks like:

```
mysql> SELECT *
       FROM INFORMATION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS
       WHERE SRS_ID = 4326\G
*************************** 1. row ***************************
                SRS_NAME: WGS 84
                  SRS_ID: 4326
            ORGANIZATION: EPSG
ORGANIZATION_COORDSYS_ID: 4326
              DEFINITION: GEOGCS["WGS 84",DATUM["World Geodetic System 1984",
```

```
                        SPHEROID["WGS 84",6378137,298.257223563,
                        AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],
                        PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
                        UNIT["degree",0.017453292519943278,
                        AUTHORITY["EPSG","9122"]],
                        AXIS["Lat",NORTH],AXIS["Long",EAST],
                        AUTHORITY["EPSG","4326"]]
            DESCRIPTION:
```

This entry describes the SRS used for GPS systems. It has a name (`SRS_NAME`) of WGS 84 and an ID (`SRS_ID`) of 4326, which is the ID used by the European Petroleum Survey Group (EPSG).

SRS definitions in the `DEFINITION` column are WKT values, represented as specified in the Open Geospatial Consortium document OGC 12-063r5.

`SRS_ID` values represent the same kind of values passed as the SRID argument to spatial functions. SRID 0 (the unitless Cartesian plane) is special. It is always a legal spatial reference system ID and can be used in any computations on spatial data that depend on SRID values.

For computations on multiple geometry values, all values must have the same SRID or an error occurs.

SRS definition parsing occurs on demand when definitions are needed by GIS functions. Parsed definitions are cached in the data dictionary cache so that parsing overhead is not incurred for every statement that needs SRS information.

To enable manipulation of SRS entries stored in the data dictionary, MySQL provides these SQL statements:

- `CREATE SPATIAL REFERENCE SYSTEM`: See Section 13.1.19, “CREATE SPATIAL REFERENCE SYSTEM Syntax”. The description for this statement includes additional information about SRS components.
- `DROP SPATIAL REFERENCE SYSTEM`: See Section 13.1.31, “DROP SPATIAL REFERENCE SYSTEM Syntax”.

11.5.6 Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with `CREATE TABLE` or `ALTER TABLE`. Spatial columns are supported for `MyISAM`, `InnoDB`, `NDB`, and `ARCHIVE` tables. See also the notes about spatial indexes under Section 11.5.10, “Creating Spatial Indexes”.

Columns with a spatial data type can have an SRID attribute, to explicitly indicate the spatial reference system (SRS) for values stored in the column. For implications of an SRID-restricted column, see Section 11.5.1, “Spatial Data Types”.

- Use the `CREATE TABLE` statement to create a table with a spatial column:

```
CREATE TABLE geom (g GEOMETRY);
```

- Use the `ALTER TABLE` statement to add or drop a spatial column to or from an existing table:

```
ALTER TABLE geom ADD pt POINT;
ALTER TABLE geom DROP pt;
```

11.5.7 Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values to internal geometry format:

- Perform the conversion directly in the `INSERT` statement:

```
INSERT INTO geom VALUES (ST_GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

- Perform the conversion prior to the `INSERT`:

```
SET @g = ST_GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

The following examples insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (ST_GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (ST_GeomFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (ST_GeomFromText(@g));
```

The preceding examples use `ST_GeomFromText()` to create geometry values. You can also use type-specific functions:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (ST_PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (ST_LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (ST_PolygonFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (ST_GeomCollFromText(@g));
```

A client application program that wants to use WKB representations of geometry values is responsible for sending correctly formed WKB in queries to the server. There are several ways to satisfy this requirement. For example:

- Inserting a `POINT(1 1)` value with hex literal syntax:

```
INSERT INTO geom VALUES
(ST_GeomFromWKB(X'0101000000000000000000F03F000000000000F03F'));
```

- An ODBC application can send a WKB representation, binding it to a placeholder using an argument of `BLOB` type:

```
INSERT INTO geom VALUES (ST_GeomFromWKB(?))
```

Other programming interfaces may support a similar placeholder mechanism.

- In a C program, you can escape a binary value using `mysql_real_escape_string_quote()` and include the result in a query string that is sent to the server. See Section 28.7.7.56, “mysql_real_escape_string_quote()”.

11.5.8 Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them to WKT or WKB format.

- Fetching spatial data in internal format:

  Fetching geometry values using internal format can be useful in table-to-table transfers:

  ```
  CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
  ```

- Fetching spatial data in WKT format:

  The `ST_AsText()` function converts a geometry from internal format to a WKT string.

  ```
  SELECT ST_AsText(g) FROM geom;
  ```

- Fetching spatial data in WKB format:

  The `ST_AsBinary()` function converts a geometry from internal format to a `BLOB` containing the WKB value.

  ```
  SELECT ST_AsBinary(g) FROM geom;
  ```

11.5.9 Optimizing Spatial Analysis

For `MyISAM` and `InnoDB` tables, search operations in columns containing spatial data can be optimized using `SPATIAL` indexes. The most typical operations are:

- Point queries that search for all objects that contain a given point
- Region queries that search for all objects that overlap a given region

MySQL uses **R-Trees with quadratic splitting** for `SPATIAL` indexes on spatial columns. A `SPATIAL` index is built using the minimum bounding rectangle (MBR) of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

It is also possible to create normal indexes on spatial columns. In a non-`SPATIAL` index, you must declare a prefix for any spatial column except for `POINT` columns.

`MyISAM` and `InnoDB` support both `SPATIAL` and non-`SPATIAL` indexes. Other storage engines support non-`SPATIAL` indexes, as described in Section 13.1.15, “CREATE INDEX Syntax”.

11.5.10 Creating Spatial Indexes

For `InnoDB` and `MyISAM` tables, MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but using the `SPATIAL` keyword. Columns in spatial indexes must be declared `NOT NULL`. The following examples demonstrate how to create spatial indexes:

- With `CREATE TABLE`:

```
CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326, SPATIAL INDEX(g));
```

- With `ALTER TABLE`:

```
CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326);
ALTER TABLE geom ADD SPATIAL INDEX(g);
```

- With `CREATE INDEX`:

```
CREATE TABLE geom (g GEOMETRY NOT NULL SRID 4326);
CREATE SPATIAL INDEX g ON geom (g);
```

`SPATIAL INDEX` creates an R-tree index. For storage engines that support nonspatial indexing of spatial columns, the engine creates a B-tree index. A B-tree index on spatial values is useful for exact-value lookups, but not for range scans.

The optimizer can use spatial indexes defined on columns that are SRID-restricted. For more information, see Section 11.5.1, “Spatial Data Types”, and Section 8.3.3, “SPATIAL Index Optimization”.

For more information on indexing spatial columns, see Section 13.1.15, “CREATE INDEX Syntax”.

To drop spatial indexes, use `ALTER TABLE` or `DROP INDEX`:

- With `ALTER TABLE`:

```
ALTER TABLE geom DROP INDEX g;
```

- With `DROP INDEX`:

```
DROP INDEX g ON geom;
```

Example: Suppose that a table `geom` contains more than 32,000 geometries, which are stored in the column `g` of type `GEOMETRY`. The table also has an `AUTO_INCREMENT` column `fid` for storing object ID values.

```
mysql> DESCRIBE geom;
+-------+----------+------+-----+---------+----------------+
| Field | Type     | Null | Key | Default | Extra          |
+-------+----------+------+-----+---------+----------------+
| fid   | int(11)  |      | PRI | NULL    | auto_increment |
| g     | geometry |      |     |         |                |
+-------+----------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+----------+
| count(*) |
+----------+
|    32376 |
+----------+
1 row in set (0.00 sec)
```

To add a spatial index on the column `g`, use this statement:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
```

```
Query OK, 32376 rows affected (4.05 sec)
Records: 32376  Duplicates: 0  Warnings: 0
```

11.5.11 Using Spatial Indexes

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as `MBRContains()` or `MBRWithin()` in the `WHERE` clause. The following query finds all objects that are in the given rectangle:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> SELECT fid,ST_AsText(g) FROM geom WHERE
    -> MBRContains(ST_GeomFromText(@poly),g);
+-----+-------------------------------------------------------------+
| fid | ST_AsText(g)                                                |
+-----+-------------------------------------------------------------+
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
+-----+-------------------------------------------------------------+
20 rows in set (0.00 sec)
```

Use `EXPLAIN` to check the way this query is executed:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> EXPLAIN SELECT fid,ST_AsText(g) FROM geom WHERE
    -> MBRContains(ST_GeomFromText(@poly),g)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: geom
         type: range
possible_keys: g
          key: g
      key_len: 32
          ref: NULL
         rows: 50
        Extra: Using where
```

```
1 row in set (0.00 sec)
```

Check what would happen without a spatial index:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> EXPLAIN SELECT fid,ST_AsText(g) FROM g IGNORE INDEX (g) WHERE
    -> MBRContains(ST_GeomFromText(@poly),g)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: geom
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 32376
        Extra: Using where
1 row in set (0.00 sec)
```

Executing the `SELECT` statement without the spatial index yields the same result but causes the execution time to rise from 0.00 seconds to 0.46 seconds:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> SELECT fid,ST_AsText(g) FROM geom IGNORE INDEX (g) WHERE
    -> MBRContains(ST_GeomFromText(@poly),g);
+-----+-------------------------------------------------------------+
| fid | ST_AsText(g)                                                |
+-----+-------------------------------------------------------------+
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
+-----+-------------------------------------------------------------+
20 rows in set (0.46 sec)
```

11.6 The JSON Data Type

- Creating JSON Values
- Normalization, Merging, and Autowrapping of JSON Values
- Searching and Modifying JSON Values
- JSON Path Syntax
- Comparison and Ordering of JSON Values
- Converting between JSON and non-JSON values
- Aggregation of JSON Values

MySQL supports a native `JSON` data type defined by RFC 7159 that enables efficient access to data in JSON (JavaScript Object Notation) documents. The `JSON` data type provides these advantages over storing JSON-format strings in a string column:

- Automatic validation of JSON documents stored in `JSON` columns. Invalid documents produce an error.
- Optimized storage format. JSON documents stored in `JSON` columns are converted to an internal format that permits quick read access to document elements. When the server later must read a JSON value stored in this binary format, the value need not be parsed from a text representation. The binary format is structured to enable the server to look up subobjects or nested values directly by key or array index without reading all values before or after them in the document.

MySQL 8.0 also supports the *JSON Merge Patch* format defined in RFC 7396, using the `JSON_MERGE_PATCH()` function. See the description of this function, as well as Normalization, Merging, and Autowrapping of JSON Values, for examples and further information.

Note

This discussion uses `JSON` in monotype to indicate specifically the JSON data type and “JSON” in regular font to indicate JSON data in general.

The space required to store a `JSON` document is roughly the same as for `LONGBLOB` or `LONGTEXT`; see Section 11.8, “Data Type Storage Requirements”, for more information. It is important to keep in mind that the size of any JSON document stored in a `JSON` column is limited to the value of the `max_allowed_packet` system variable. (When the server is manipulating a JSON value internally in memory, it can be larger than this; the limit applies when the server stores it.) You can obtain the amount of space required to store a JSON document using the `JSON_STORAGE_SIZE()` function; note that for a `JSON` column, the storage size—and thus the value returned by this function—is that used by the column prior to any partial updates that may have been performed on it (see the discussion of the JSON partial update optimization later in this section).

Prior to MySQL 8.0.13, a `JSON` column cannot have a non-`NULL` default value.

Along with the `JSON` data type, a set of SQL functions is available to enable operations on JSON values, such as creation, manipulation, and searching. The following discussion shows examples of these operations. For details about individual functions, see Section 12.17, “JSON Functions”.

A set of spatial functions for operating on GeoJSON values is also available. See Section 12.16.11, “Spatial GeoJSON Functions”.

`JSON` columns, like columns of other binary types, are not indexed directly; instead, you can create an index on a generated column that extracts a scalar value from the `JSON` column. See Indexing a Generated Column to Provide a JSON Column Index, for a detailed example.

The MySQL optimizer also looks for compatible indexes on virtual columns that match JSON expressions.

In MySQL 8.0.17 and later, the `InnoDB` storage engine supports multi-valued indexes on JSON arrays. See Multi-Valued Indexes.

MySQL NDB Cluster 8.0 supports `JSON` columns and MySQL JSON functions, including creation of an index on a column generated from a `JSON` column as a workaround for being unable to index a `JSON` column. A maximum of 3 `JSON` columns per `NDB` table is supported.

Partial Updates of JSON Values

In MySQL 8.0, the optimizer can perform a partial, in-place update of a `JSON` column instead of removing the old document and writing the new document in its entirety to the column. This optimization can be performed for an update that meets the following conditions:

- The column being updated was declared as `JSON`.
- The `UPDATE` statement uses any of the three functions `JSON_SET()`, `JSON_REPLACE()`, or `JSON_REMOVE()` to update the column. A direct assignment of the column value (for example, `UPDATE mytable SET jcol = '{"a": 10, "b": 25}'`) cannot be performed as a partial update.

  Updates of multiple `JSON` columns in a single `UPDATE` statement can be optimized in this fashion; MySQL can perform partial updates of only those columns whose values are updated using the three functions just listed.
- The input column and the target column must be the same column; a statement such as `UPDATE mytable SET jcol1 = JSON_SET(jcol2, '$.a', 100)` cannot be performed as a partial update.

  The update can use nested calls to any of the functions listed in the previous item, in any combination, as long as the input and target columns are the same.
- All changes replace existing array or object values with new ones, and do not add any new elements to the parent object or array.
- The value being replaced must be at least as large as the replacement value. In other words, the new value cannot be any larger than the old one.

  A possible exception to this requirement occurs when a previous partial update has left sufficient space for the larger value. You can use the function `JSON_STORAGE_FREE()` see how much space has been freed by any partial updates of a `JSON` column.

Such partial updates can be written to the binary log using a compact format that saves space; this can be enabled by setting the `binlog_row_value_options` system variable to `PARTIAL_JSON`. See the description of this variable for more information.

The next few sections provide basic information regarding the creation and manipulation of JSON values.

Creating JSON Values

A JSON array contains a list of values separated by commas and enclosed within `[` and `]` characters:

```
["abc", 10, null, true, false]
```

A JSON object contains a set of key-value pairs separated by commas and enclosed within `{` and `}` characters:

```
{"k1": "value", "k2": 10}
```

As the examples illustrate, JSON arrays and objects can contain scalar values that are strings or numbers, the JSON null literal, or the JSON boolean true or false literals. Keys in JSON objects must be strings. Temporal (date, time, or datetime) scalar values are also permitted:

```
["12:18:29.000000", "2015-07-29", "2015-07-29 12:18:29.000000"]
```

Nesting is permitted within JSON array elements and JSON object key values:

```
[99, {"id": "HK500", "cost": 75.99}, ["hot", "cold"]]
{"k1": "value", "k2": [10, 20]}
```

You can also obtain JSON values from a number of functions supplied by MySQL for this purpose (see Section 12.17.2, “Functions That Create JSON Values”) as well as by casting values of other types to the `JSON` type using `CAST(value AS JSON)` (see Converting between JSON and non-JSON values). The next several paragraphs describe how MySQL handles JSON values provided as input.

In MySQL, JSON values are written as strings. MySQL parses any string used in a context that requires a JSON value, and produces an error if it is not valid as JSON. These contexts include inserting a value into a column that has the `JSON` data type and passing an argument to a function that expects a JSON value (usually shown as *`json_doc`* or *`json_val`* in the documentation for MySQL JSON functions), as the following examples demonstrate:

- Attempting to insert a value into a `JSON` column succeeds if the value is a valid JSON value, but fails if it is not:

  ```
  mysql> CREATE TABLE t1 (jdoc JSON);
  Query OK, 0 rows affected (0.20 sec)

  mysql> INSERT INTO t1 VALUES('{"key1": "value1", "key2": "value2"}');
  Query OK, 1 row affected (0.01 sec)

  mysql> INSERT INTO t1 VALUES('[1, 2,');
  ERROR 3140 (22032) at line 2: Invalid JSON text:
  "Invalid value." at position 6 in value (or column) '[1, 2,'.
  ```

  Positions for “at position *N*” in such error messages are 0-based, but should be considered rough indications of where the problem in a value actually occurs.

- The `JSON_TYPE()` function expects a JSON argument and attempts to parse it into a JSON value. It returns the value's JSON type if it is valid and produces an error otherwise:

  ```
  mysql> SELECT JSON_TYPE('["a", "b", 1]');
  +---------------------------+
  | JSON_TYPE('["a", "b", 1]') |
  +---------------------------+
  | ARRAY                     |
  +---------------------------+

  mysql> SELECT JSON_TYPE('"hello"');
  +----------------------+
  | JSON_TYPE('"hello"') |
  +----------------------+
  | STRING               |
  +----------------------+
  ```

```
mysql> SELECT JSON_TYPE('hello');
ERROR 3146 (22032): Invalid data type for JSON data in argument 1
to function json_type; a JSON string or JSON type is required.
```

MySQL handles strings used in JSON context using the `utf8mb4` character set and `utf8mb4_bin` collation. Strings in other character sets are converted to `utf8mb4` as necessary. (For strings in the `ascii` or `utf8` character sets, no conversion is needed because `ascii` and `utf8` are subsets of `utf8mb4`.)

As an alternative to writing JSON values using literal strings, functions exist for composing JSON values from component elements. `JSON_ARRAY()` takes a (possibly empty) list of values and returns a JSON array containing those values:

```
mysql> SELECT JSON_ARRAY('a', 1, NOW());
+---------------------------------------+
| JSON_ARRAY('a', 1, NOW())             |
+---------------------------------------+
| ["a", 1, "2015-07-27 09:43:47.000000"] |
+---------------------------------------+
```

`JSON_OBJECT()` takes a (possibly empty) list of key-value pairs and returns a JSON object containing those pairs:

```
mysql> SELECT JSON_OBJECT('key1', 1, 'key2', 'abc');
+--------------------------------------+
| JSON_OBJECT('key1', 1, 'key2', 'abc') |
+--------------------------------------+
| {"key1": 1, "key2": "abc"}           |
+--------------------------------------+
```

`JSON_MERGE_PRESERVE()` takes two or more JSON documents and returns the combined result:

```
mysql> SELECT JSON_MERGE_PRESERVE('["a", 1]', '{"key": "value"}');
+-----------------------------------------------------+
| JSON_MERGE_PRESERVE('["a", 1]', '{"key": "value"}') |
+-----------------------------------------------------+
| ["a", 1, {"key": "value"}]                          |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

For information about the merging rules, see Normalization, Merging, and Autowrapping of JSON Values.

(MySQL 8.0.3 and later also support `JSON_MERGE_PATCH()`, which has somewhat different behavior. See JSON_MERGE_PATCH() compared with JSON_MERGE_PRESERVE(), for information about the differences between these two functions.)

JSON values can be assigned to user-defined variables:

```
mysql> SET @j = JSON_OBJECT('key', 'value');
mysql> SELECT @j;
+------------------+
| @j               |
+------------------+
| {"key": "value"} |
+------------------+
```

However, user-defined variables cannot be of `JSON` data type, so although `@j` in the preceding example looks like a JSON value and has the same character set and collation as a JSON value, it does *not* have the `JSON` data type. Instead, the result from `JSON_OBJECT()` is converted to a string when assigned to the variable.

Strings produced by converting JSON values have a character set of `utf8mb4` and a collation of `utf8mb4_bin`:

```
mysql> SELECT CHARSET(@j), COLLATION(@j);
+-------------+---------------+
| CHARSET(@j) | COLLATION(@j) |
+-------------+---------------+
| utf8mb4     | utf8mb4_bin   |
+-------------+---------------+
```

Because `utf8mb4_bin` is a binary collation, comparison of JSON values is case-sensitive.

```
mysql> SELECT JSON_ARRAY('x') = JSON_ARRAY('X');
+-----------------------------------+
| JSON_ARRAY('x') = JSON_ARRAY('X') |
+-----------------------------------+
|                                 0 |
+-----------------------------------+
```

Case sensitivity also applies to the JSON `null`, `true`, and `false` literals, which always must be written in lowercase:

```
mysql> SELECT JSON_VALID('null'), JSON_VALID('Null'), JSON_VALID('NULL');
+--------------------+--------------------+--------------------+
| JSON_VALID('null') | JSON_VALID('Null') | JSON_VALID('NULL') |
+--------------------+--------------------+--------------------+
|                  1 |                  0 |                  0 |
+--------------------+--------------------+--------------------+

mysql> SELECT CAST('null' AS JSON);
+----------------------+
| CAST('null' AS JSON) |
+----------------------+
| null                 |
+----------------------+
1 row in set (0.00 sec)

mysql> SELECT CAST('NULL' AS JSON);
ERROR 3141 (22032): Invalid JSON text in argument 1 to function cast_as_json:
"Invalid value." at position 0 in 'NULL'.
```

Case sensitivity of the JSON literals differs from that of the SQL `NULL`, `TRUE`, and `FALSE` literals, which can be written in any lettercase:

```
mysql> SELECT ISNULL(null), ISNULL(Null), ISNULL(NULL);
+--------------+--------------+--------------+
| ISNULL(null) | ISNULL(Null) | ISNULL(NULL) |
+--------------+--------------+--------------+
|            1 |            1 |            1 |
+--------------+--------------+--------------+
```

Sometimes it may be necessary or desirable to insert quote characters (`"` or `'`) into a JSON document. Assume for this example that you want to insert some JSON objects containing strings representing sentences that state some facts about MySQL, each paired with an appropriate keyword, into a table created using the SQL statement shown here:

```
mysql> CREATE TABLE facts (sentence JSON);
```

Among these keyword-sentence pairs is this one:

```
mascot: The MySQL mascot is a dolphin named "Sakila".
```

One way to insert this as a JSON object into the `facts` table is to use the MySQL `JSON_OBJECT()` function. In this case, you must escape each quote character using a backslash, as shown here:

```
mysql> INSERT INTO facts VALUES
    >   (JSON_OBJECT("mascot", "Our mascot is a dolphin named \"Sakila\"."));
```

This does not work in the same way if you insert the value as a JSON object literal, in which case, you must use the double backslash escape sequence, like this:

```
mysql> INSERT INTO facts VALUES
    >   ('{"mascot": "Our mascot is a dolphin named \\"Sakila\\"."}');
```

Using the double backslash keeps MySQL from performing escape sequence processing, and instead causes it to pass the string literal to the storage engine for processing. After inserting the JSON object in either of the ways just shown, you can see that the backslashes are present in the JSON column value by doing a simple `SELECT`, like this:

```
mysql> SELECT sentence FROM facts;
+-------------------------------------------------------+
| sentence                                              |
+-------------------------------------------------------+
| {"mascot": "Our mascot is a dolphin named \"Sakila\"."} |
+-------------------------------------------------------+
```

To look up this particular sentence employing `mascot` as the key, you can use the column-path operator `->`, as shown here:

```
mysql> SELECT col->"$.mascot" FROM qtest;
+-----------------------------------------+
| col->"$.mascot"                         |
+-----------------------------------------+
| "Our mascot is a dolphin named \"Sakila\"." |
+-----------------------------------------+
1 row in set (0.00 sec)
```

This leaves the backslashes intact, along with the surrounding quote marks. To display the desired value using `mascot` as the key, but without including the surrounding quote marks or any escapes, use the inline path operator `->>`, like this:

```
mysql> SELECT sentence->>"$.mascot" FROM facts;
+---------------------------------------+
| sentence->>"$.mascot"                 |
+---------------------------------------+
| Our mascot is a dolphin named "Sakila". |
+---------------------------------------+
```

Note

The previous example does not work as shown if the `NO_BACKSLASH_ESCAPES` server SQL mode is enabled. If this mode is set, a single backslash instead of double backslashes can be used to insert the JSON object literal, and the backslashes are preserved. If you use the `JSON_OBJECT()` function when

performing the insert and this mode is set, you must alternate single and double quotes, like this:

```
mysql> INSERT INTO facts VALUES
    > (JSON_OBJECT('mascot', 'Our mascot is a dolphin named "Sakila".'));
```

See the description of the `JSON_UNQUOTE()` function for more information about the effects of this mode on escaped characters in JSON values.

Normalization, Merging, and Autowrapping of JSON Values

When a string is parsed and found to be a valid JSON document, it is also normalized. This means that members with keys that duplicate a key found later in the document, reading from left to right, are discarded. The object value produced by the following `JSON_OBJECT()` call includes only the second `key1` element because that key name occurs earlier in the value, as shown here:

```
mysql> SELECT JSON_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def');
+-----------------------------------------------------+
| JSON_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def') |
+-----------------------------------------------------+
| {"key1": "def", "key2": "abc"}                      |
+-----------------------------------------------------+
```

Normalization is also performed when values are inserted into JSON columns, as shown here:

```
mysql> CREATE TABLE t1 (c1 JSON);

mysql> INSERT INTO t1 VALUES
    >     ('{"x": 17, "x": "red"}'),
    >     ('{"x": 17, "x": "red", "x": [3, 5, 7]}');

mysql> SELECT c1 FROM t1;
+------------------+
| c1               |
+------------------+
| {"x": "red"}     |
| {"x": [3, 5, 7]} |
+------------------+
```

This “last duplicate key wins” behavior is suggested by RFC 7159 and is implemented by most JavaScript parsers. (Bug #86866, Bug #26369555)

In versions of MySQL prior to 8.0.3, members with keys that duplicated a key found earlier in the document were discarded. The object value produced by the following `JSON_OBJECT()` call does not include the second `key1` element because that key name occurs earlier in the value:

```
mysql> SELECT JSON_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def');
+-----------------------------------------------------+
| JSON_OBJECT('key1', 1, 'key2', 'abc', 'key1', 'def') |
+-----------------------------------------------------+
| {"key1": 1, "key2": "abc"}                          |
+-----------------------------------------------------+
```

Prior to MySQL 8.0.3, this “first duplicate key wins” normalization was also performed when inserting values into JSON columns.

```
mysql> CREATE TABLE t1 (c1 JSON);
```

```
mysql> INSERT INTO t1 VALUES
     >     ('{"x": 17, "x": "red"}'),
     >     ('{"x": 17, "x": "red", "x": [3, 5, 7]}');

mysql> SELECT c1 FROM t1;
+-----------+
| c1        |
+-----------+
| {"x": 17} |
| {"x": 17} |
+-----------+
```

MySQL also discards extra whitespace between keys, values, or elements in the original JSON document. To make lookups more efficient, it also sorts the keys of a JSON object. *You should be aware that the result of this ordering is subject to change and not guaranteed to be consistent across releases.*

MySQL functions that produce JSON values (see Section 12.17.2, “Functions That Create JSON Values”) always return normalized values.

Merging JSON Values

Two merging algorithms are supported in MySQL 8.0.3 (and later), implemented by the functions `JSON_MERGE_PRESERVE()` and `JSON_MERGE_PATCH()`. These differ in how they handle duplicate keys: `JSON_MERGE_PRESERVE()` retains values for duplicate keys, while `JSON_MERGE_PATCH()` discards all but the last value. The next few paragraphs explain how each of these two functions handles the merging of different combinations of JSON documents (that is, of objects and arrays).

Note

`JSON_MERGE_PRESERVE()` is the same as the `JSON_MERGE()` function found in previous versions of MySQL (renamed in MySQL 8.0.3). `JSON_MERGE()` is still supported as an alias for `JSON_MERGE_PRESERVE()` in MySQL 8.0, but is deprecated and subject to removal in a future release.

Merging arrays. In contexts that combine multiple arrays, the arrays are merged into a single array. `JSON_MERGE_PRESERVE()` does this by concatenating arrays named later to the end of the first array. `JSON_MERGE_PATCH()` considers each argument as an array consisting of a single element (thus having 0 as its index) and then applies “last duplicate key wins” logic to select only the last argument. You can compare the results shown by this query:

```
mysql> SELECT
    ->   JSON_MERGE_PRESERVE('[1, 2]', '["a", "b", "c"]', '[true, false]') AS Preserve,
    ->   JSON_MERGE_PATCH('[1, 2]', '["a", "b", "c"]', '[true, false]') AS Patch\G
*************************** 1. row ***************************
Preserve: [1, 2, "a", "b", "c", true, false]
   Patch: [true, false]
```

Multiple objects when merged produce a single object. `JSON_MERGE_PRESERVE()` handles multiple objects having the same key by combining all unique values for that key in an array; this array is then used as the value for that key in the result. `JSON_MERGE_PATCH()` discards values for which duplicate keys are found, working from left to right, so that the result contains only the last value for that key. The following query illustrates the difference in the results for the duplicate key `a`:

```
mysql> SELECT
    ->   JSON_MERGE_PRESERVE('{"a": 1, "b": 2}', '{"c": 3, "a": 4}', '{"c": 5, "d": 3}') AS Preserve,
    ->   JSON_MERGE_PATCH('{"a": 3, "b": 2}', '{"c": 3, "a": 4}', '{"c": 5, "d": 3}') AS Patch\G
*************************** 1. row ***************************
Preserve: {"a": [1, 4], "b": 2, "c": [3, 5], "d": 3}
```

```
   Patch: {"a": 4, "b": 2, "c": 5, "d": 3}
```

Nonarray values used in a context that requires an array value are autowrapped: The value is surrounded by `[` and `]` characters to convert it to an array. In the following statement, each argument is autowrapped as an array (`[1]`, `[2]`). These are then merged to produce a single result array; as in the previous two cases, `JSON_MERGE_PRESERVE()` combines values having the same key while `JSON_MERGE_PATCH()` discards values for all duplicate keys except the last, as shown here:

```
mysql> SELECT
   ->   JSON_MERGE_PRESERVE('1', '2') AS Preserve,
   ->   JSON_MERGE_PATCH('1', '2') AS Patch\G
*************************** 1. row ***************************
Preserve: [1, 2]
   Patch: 2
```

Array and object values are merged by autowrapping the object as an array and merging the arrays by combining values or by “last duplicate key wins” according to the choice of merging function (`JSON_MERGE_PRESERVE()` or `JSON_MERGE_PATCH()`, respectively), as can be seen in this example:

```
mysql> SELECT
   ->   JSON_MERGE_PRESERVE('[10, 20]', '{"a": "x", "b": "y"}') AS Preserve,
   ->   JSON_MERGE_PATCH('[10, 20]', '{"a": "x", "b": "y"}') AS Patch\G
*************************** 1. row ***************************
Preserve: [10, 20, {"a": "x", "b": "y"}]
   Patch: {"a": "x", "b": "y"}
```

Searching and Modifying JSON Values

A JSON path expression selects a value within a JSON document.

Path expressions are useful with functions that extract parts of or modify a JSON document, to specify where within that document to operate. For example, the following query extracts from a JSON document the value of the member with the `name` key:

```
mysql> SELECT JSON_EXTRACT('{"id": 14, "name": "Aztalan"}', '$.name');
+-------------------------------------------------------+
| JSON_EXTRACT('{"id": 14, "name": "Aztalan"}', '$.name') |
+-------------------------------------------------------+
| "Aztalan"                                             |
+-------------------------------------------------------+
```

Path syntax uses a leading `$` character to represent the JSON document under consideration, optionally followed by selectors that indicate successively more specific parts of the document:

- A period followed by a key name names the member in an object with the given key. The key name must be specified within double quotation marks if the name without quotes is not legal within path expressions (for example, if it contains a space).
- `[`*`N`*`]` appended to a *`path`* that selects an array names the value at position *`N`* within the array. Array positions are integers beginning with zero. If *`path`* does not select an array value, *`path`*[0] evaluates to the same value as *`path`*:

  ```
  mysql> SELECT JSON_SET('"x"', '$[0]', 'a');
  +-----------------------------+
  | JSON_SET('"x"', '$[0]', 'a') |
  +-----------------------------+
  | "a"                         |
  ```

```
+----------------------------+
1 row in set (0.00 sec)
```

- `[M to N]` specifies a subset or range of array values starting with the value at position *`M`*, and ending with the value at position *`N`*.

  `last` is supported as a synonym for the index of the rightmost array element. Relative addressing of array elements is also supported. If *`path`* does not select an array value, *`path`*[last] evaluates to the same value as *`path`*, as shown later in this section (see Rightmost array element).

- Paths can contain `*` or `**` wildcards:

  - `.[*]` evaluates to the values of all members in a JSON object.
  - `[*]` evaluates to the values of all elements in a JSON array.
  - *`prefix`*`**`*`suffix`* evaluates to all paths that begin with the named prefix and end with the named suffix.

- A path that does not exist in the document (evaluates to nonexistent data) evaluates to `NULL`.

Let `$` refer to this JSON array with three elements:

```
[3, {"a": [5, 6], "b": 10}, [99, 100]]
```

Then:

- `$[0]` evaluates to `3`.
- `$[1]` evaluates to `{"a": [5, 6], "b": 10}`.
- `$[2]` evaluates to `[99, 100]`.
- `$[3]` evaluates to `NULL` (it refers to the fourth array element, which does not exist).

Because `$[1]` and `$[2]` evaluate to nonscalar values, they can be used as the basis for more-specific path expressions that select nested values. Examples:

- `$[1].a` evaluates to `[5, 6]`.
- `$[1].a[1]` evaluates to `6`.
- `$[1].b` evaluates to `10`.
- `$[2][0]` evaluates to `99`.

As mentioned previously, path components that name keys must be quoted if the unquoted key name is not legal in path expressions. Let `$` refer to this value:

```
{"a fish": "shark", "a bird": "sparrow"}
```

The keys both contain a space and must be quoted:

- `$."a fish"` evaluates to `shark`.
- `$."a bird"` evaluates to `sparrow`.

Paths that use wildcards evaluate to an array that can contain multiple values:

```
mysql> SELECT JSON_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.*');
+---------------------------------------------------------+
| JSON_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.*') |
+---------------------------------------------------------+
| [1, 2, [3, 4, 5]]                                       |
+---------------------------------------------------------+
mysql> SELECT JSON_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.c[*]');
+-----------------------------------------------------------+
| JSON_EXTRACT('{"a": 1, "b": 2, "c": [3, 4, 5]}', '$.c[*]') |
+-----------------------------------------------------------+
| [3, 4, 5]                                                 |
+-----------------------------------------------------------+
```

In the following example, the path `$**.b` evaluates to multiple paths (`$.a.b` and `$.c.b`) and produces an array of the matching path values:

```
mysql> SELECT JSON_EXTRACT('{"a": {"b": 1}, "c": {"b": 2}}', '$**.b');
+---------------------------------------------------------+
| JSON_EXTRACT('{"a": {"b": 1}, "c": {"b": 2}}', '$**.b') |
+---------------------------------------------------------+
| [1, 2]                                                  |
+---------------------------------------------------------+
```

Ranges from JSON arrays. You can use ranges with the `to` keyword to specify subsets of JSON arrays. For example, `$[1 to 3]` includes the second, third, and fourth elements of an array, as shown here:

```
mysql> SELECT JSON_EXTRACT('[1, 2, 3, 4, 5]', '$[1 to 3]');
+-----------------------------------------------+
| JSON_EXTRACT('[1, 2, 3, 4, 5]', '$[1 to 3]') |
+-----------------------------------------------+
| [2, 3, 4]                                     |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

The syntax is *`M to N`*, where *`M`* and *`N`* are, respectively, the first and last indexes of a range of elements from a JSON array. *`N`* must be greater than *`M`*; *`M`* must be greater than or equal to 0. Array elements are indexed beginning with 0.

You can use ranges in contexts where wildcards are supported.

Rightmost array element. The `last` keyword is supported as a synonym for the index of the last element in an array. Expressions of the form `last - N` can be used for relative addressing, and within range definitions, like this:

```
mysql> SELECT JSON_EXTRACT('[1, 2, 3, 4, 5]', '$[last-3 to last-1]');
+-------------------------------------------------------+
| JSON_EXTRACT('[1, 2, 3, 4, 5]', '$[last-3 to last-1]') |
+-------------------------------------------------------+
| [2, 3, 4]                                             |
+-------------------------------------------------------+
1 row in set (0.01 sec)
```

If the path is evaluated against a value that is not an array, the result of the evaluation is the same as if the value had been wrapped in a single-element array:

```
mysql> SELECT JSON_REPLACE('"Sakila"', '$[last]', 10);
+-----------------------------------------+
| JSON_REPLACE('"Sakila"', '$[last]', 10) |
```

```
+----------------------------------------+
| 10                                     |
+----------------------------------------+
1 row in set (0.00 sec)
```

You can use *`column->path`* with a JSON column identifier and JSON path expression as a synonym for `JSON_EXTRACT(`*`column, path`*`)`. See Section 12.17.3, “Functions That Search JSON Values”, for more information. See also Indexing a Generated Column to Provide a JSON Column Index.

Some functions take an existing JSON document, modify it in some way, and return the resulting modified document. Path expressions indicate where in the document to make changes. For example, the `JSON_SET()`, `JSON_INSERT()`, and `JSON_REPLACE()` functions each take a JSON document, plus one or more path-value pairs that describe where to modify the document and the values to use. The functions differ in how they handle existing and nonexisting values within the document.

Consider this document:

```
mysql> SET @j = '["a", {"b": [true, false]}, [10, 20]]';
```

`JSON_SET()` replaces values for paths that exist and adds values for paths that do not exist:.

```
mysql> SELECT JSON_SET(@j, '$[1].b[0]', 1, '$[2][2]', 2);
+-----------------------------------------+
| JSON_SET(@j, '$[1].b[0]', 1, '$[2][2]', 2) |
+-----------------------------------------+
| ["a", {"b": [1, false]}, [10, 20, 2]]      |
+-----------------------------------------+
```

In this case, the path `$[1].b[0]` selects an existing value (`true`), which is replaced with the value following the path argument (`1`). The path `$[2][2]` does not exist, so the corresponding value (`2`) is added to the value selected by `$[2]`.

`JSON_INSERT()` adds new values but does not replace existing values:

```
mysql> SELECT JSON_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2);
+-----------------------------------------+
| JSON_INSERT(@j, '$[1].b[0]', 1, '$[2][2]', 2) |
+-----------------------------------------+
| ["a", {"b": [true, false]}, [10, 20, 2]]      |
+-----------------------------------------+
```

`JSON_REPLACE()` replaces existing values and ignores new values:

```
mysql> SELECT JSON_REPLACE(@j, '$[1].b[0]', 1, '$[2][2]', 2);
+-----------------------------------------+
| JSON_REPLACE(@j, '$[1].b[0]', 1, '$[2][2]', 2) |
+-----------------------------------------+
| ["a", {"b": [1, false]}, [10, 20]]             |
+-----------------------------------------+
```

The path-value pairs are evaluated left to right. The document produced by evaluating one pair becomes the new value against which the next pair is evaluated.

`JSON_REMOVE()` takes a JSON document and one or more paths that specify values to be removed from the document. The return value is the original document minus the values selected by paths that exist within the document:

```
mysql> SELECT JSON_REMOVE(@j, '$[2]', '$[1].b[1]', '$[1].b[1]');
+--------------------------------------------------+
| JSON_REMOVE(@j, '$[2]', '$[1].b[1]', '$[1].b[1]') |
+--------------------------------------------------+
| ["a", {"b": [true]}]                             |
+--------------------------------------------------+
```

The paths have these effects:

- `$[2]` matches `[10, 20]` and removes it.
- The first instance of `$[1].b[1]` matches `false` in the `b` element and removes it.
- The second instance of `$[1].b[1]` matches nothing: That element has already been removed, the path no longer exists, and has no effect.

JSON Path Syntax

Many of the JSON functions supported by MySQL and described elsewhere in this Manual (see Section 12.17, “JSON Functions”) require a path expression in order to identify a specific element in a JSON document. A path consists of the path's scope followed by one or more path legs. For paths used in MySQL JSON functions, the scope is always the document being searched or otherwise operated on, represented by a leading `$` character. Path legs are separated by period characters (`.`). Cells in arrays are represented by `[N]`, where *`N`* is a non-negative integer. Names of keys must be double-quoted strings or valid ECMAScript identifiers (see `http://www.ecma-international.org/ecma-262/5.1/#sec-7.6`). Path expressions, like JSON text, should be encoded using the `ascii`, `utf8`, or `utf8mb4` character set. Other character encodings are implicitly coerced to `utf8mb4`. The complete syntax is shown here:

```
pathExpression:
    scope[(pathLeg)*]

pathLeg:
    member | arrayLocation | doubleAsterisk

member:
    period ( keyName | asterisk )

arrayLocation:
    leftBracket ( nonNegativeInteger | asterisk ) rightBracket

keyName:
    ESIdentifier | doubleQuotedString

doubleAsterisk:
    '**'

period:
    '.'

asterisk:
    '*'

leftBracket:
    '['

rightBracket:
    ']'
```

As noted previously, in MySQL, the scope of the path is always the document being operated on, represented as `$`. You can use `'$'` as a synonym for the document in JSON path expressions.

Note

Some implementations support column references for scopes of JSON paths; currently, MySQL does not support these.

The wildcard `*` and `**` tokens are used as follows:

- `.*` represents the values of all members in the object.
- `[*]` represents the values of all cells in the array.
- `[`*`prefix`*`]**`*`suffix`* represents all paths beginning with *`prefix`* and ending with *`suffix`*. *`prefix`* is optional, while *`suffix`* is required; in other words, a path may not end in `**`.

  In addition, a path may not contain the sequence `***`.

For path syntax examples, see the descriptions of the various JSON functions that take paths as arguments, such as `JSON_CONTAINS_PATH()`, `JSON_SET()`, and `JSON_REPLACE()`. For examples which include the use of the `*` and `**` wildcards, see the description of the `JSON_SEARCH()` function.

MySQL 8.0.2 and later also supports range notation for subsets of JSON arrays using the `to` keyword (such as `$[2 to 10]`), as well as the `last` keyword as a synonym for the rightmost element of an array. See Searching and Modifying JSON Values, for more information and examples.

Comparison and Ordering of JSON Values

JSON values can be compared using the `=`, `<`, `<=`, `>`, `>=`, `<>`, `!=`, and `<=>` operators.

The following comparison operators and functions are not yet supported with JSON values:

- `BETWEEN`
- `IN()`
- `GREATEST()`
- `LEAST()`

A workaround for the comparison operators and functions just listed is to cast JSON values to a native MySQL numeric or string data type so they have a consistent non-JSON scalar type.

Comparison of JSON values takes place at two levels. The first level of comparison is based on the JSON types of the compared values. If the types differ, the comparison result is determined solely by which type has higher precedence. If the two values have the same JSON type, a second level of comparison occurs using type-specific rules.

The following list shows the precedences of JSON types, from highest precedence to the lowest. (The type names are those returned by the `JSON_TYPE()` function.) Types shown together on a line have the same precedence. Any value having a JSON type listed earlier in the list compares greater than any value having a JSON type listed later in the list.

```
BLOB
BIT
OPAQUE
DATETIME
TIME
DATE
```

```
BOOLEAN
ARRAY
OBJECT
STRING
INTEGER, DOUBLE
NULL
```

For JSON values of the same precedence, the comparison rules are type specific:

- `BLOB`

  The first *`N`* bytes of the two values are compared, where *`N`* is the number of bytes in the shorter value. If the first *`N`* bytes of the two values are identical, the shorter value is ordered before the longer value.

- `BIT`

  Same rules as for `BLOB`.

- `OPAQUE`

  Same rules as for `BLOB`. `OPAQUE` values are values that are not classified as one of the other types.

- `DATETIME`

  A value that represents an earlier point in time is ordered before a value that represents a later point in time. If two values originally come from the MySQL `DATETIME` and `TIMESTAMP` types, respectively, they are equal if they represent the same point in time.

- `TIME`

  The smaller of two time values is ordered before the larger one.

- `DATE`

  The earlier date is ordered before the more recent date.

- `ARRAY`

  Two JSON arrays are equal if they have the same length and values in corresponding positions in the arrays are equal.

  If the arrays are not equal, their order is determined by the elements in the first position where there is a difference. The array with the smaller value in that position is ordered first. If all values of the shorter array are equal to the corresponding values in the longer array, the shorter array is ordered first.

  Example:

  ```
  [] < ["a"] < ["ab"] < ["ab", "cd", "ef"] < ["ab", "ef"]
  ```

- `BOOLEAN`

  The JSON false literal is less than the JSON true literal.

- `OBJECT`

  Two JSON objects are equal if they have the same set of keys, and each key has the same value in both objects.

  Example:

```
{"a": 1, "b": 2} = {"b": 2, "a": 1}
```

The order of two objects that are not equal is unspecified but deterministic.

- `STRING`

  Strings are ordered lexically on the first *`N`* bytes of the `utf8mb4` representation of the two strings being compared, where *`N`* is the length of the shorter string. If the first *`N`* bytes of the two strings are identical, the shorter string is considered smaller than the longer string.

  Example:

  ```
  "a" < "ab" < "b" < "bc"
  ```

  This ordering is equivalent to the ordering of SQL strings with collation `utf8mb4_bin`. Because `utf8mb4_bin` is a binary collation, comparison of JSON values is case-sensitive:

  ```
  "A" < "a"
  ```

- `INTEGER`, `DOUBLE`

  JSON values can contain exact-value numbers and approximate-value numbers. For a general discussion of these types of numbers, see Section 9.1.2, “Numeric Literals”.

  The rules for comparing native MySQL numeric types are discussed in Section 12.2, “Type Conversion in Expression Evaluation”, but the rules for comparing numbers within JSON values differ somewhat:

  - In a comparison between two columns that use the native MySQL `INT` and `DOUBLE` numeric types, respectively, it is known that all comparisons involve an integer and a double, so the integer is converted to double for all rows. That is, exact-value numbers are converted to approximate-value numbers.
  - On the other hand, if the query compares two JSON columns containing numbers, it cannot be known in advance whether numbers will be integer or double. To provide the most consistent behavior across all rows, MySQL converts approximate-value numbers to exact-value numbers. The resulting ordering is consistent and does not lose precision for the exact-value numbers. For example, given the scalars 9223372036854775805, 9223372036854775806, 9223372036854775807 and 9.223372036854776e18, the order is such as this:

    ```
    9223372036854775805 < 9223372036854775806 < 9223372036854775807
    < 9.223372036854776e18 = 9223372036854776000 < 9223372036854776001
    ```

  Were JSON comparisons to use the non-JSON numeric comparison rules, inconsistent ordering could occur. The usual MySQL comparison rules for numbers yield these orderings:

  - Integer comparison:

    ```
    9223372036854775805 < 9223372036854775806 < 9223372036854775807
    ```

    (not defined for 9.223372036854776e18)

  - Double comparison:

    ```
    ```

```
9223372036854775805 = 9223372036854775806 = 9223372036854775807 = 9.223372036854776e18
```

For comparison of any JSON value to SQL `NULL`, the result is `UNKNOWN`.

For comparison of JSON and non-JSON values, the non-JSON value is converted to JSON according to the rules in the following table, then the values compared as described previously.

Converting between JSON and non-JSON values

The following table provides a summary of the rules that MySQL follows when casting between JSON values and values of other types:

Table 11.3 JSON Conversion Rules

other type	CAST(other type AS JSON)	CAST(JSON AS other type)
JSON	No change	No change
utf8 character type (`utf8mb4`, `utf8`, `ascii`)	The string is parsed into a JSON value.	The JSON value is serialized into a `utf8mb4` string.
Other character types	Other character encodings are implicitly converted to `utf8mb4` and treated as described for utf8 character type.	The JSON value is serialized into a `utf8mb4` string, then cast to the other character encoding. The result may not be meaningful.
`NULL`	Results in a `NULL` value of type JSON.	Not applicable.
Geometry types	The geometry value is converted into a JSON document by calling `ST_AsGeoJSON()`.	Illegal operation. Workaround: Pass the result of `CAST(json_val AS CHAR)` to `ST_GeomFromGeoJSON()`.
All other types	Results in a JSON document consisting of a single scalar value.	Succeeds if the JSON document consists of a single scalar value of the target type and that scalar value can be cast to the target type. Otherwise, returns `NULL` and produces a warning.

`ORDER BY` and `GROUP BY` for JSON values works according to these principles:

- Ordering of scalar JSON values uses the same rules as in the preceding discussion.
- For ascending sorts, SQL `NULL` orders before all JSON values, including the JSON null literal; for descending sorts, SQL `NULL` orders after all JSON values, including the JSON null literal.
- Sort keys for JSON values are bound by the value of the `max_sort_length` system variable, so keys that differ only after the first `max_sort_length` bytes compare as equal.
- Sorting of nonscalar values is not currently supported and a warning occurs.

For sorting, it can be beneficial to cast a JSON scalar to some other native MySQL type. For example, if a column named `jdoc` contains JSON objects having a member consisting of an `id` key and a nonnegative value, use this expression to sort by `id` values:

```
ORDER BY CAST(JSON_EXTRACT(jdoc, '$.id') AS UNSIGNED)
```

If there happens to be a generated column defined to use the same expression as in the `ORDER BY`, the MySQL optimizer recognizes that and considers using the index for the query execution plan. See Section 8.3.11, “Optimizer Use of Generated Column Indexes”.

Aggregation of JSON Values

For aggregation of JSON values, SQL `NULL` values are ignored as for other data types. Non-`NULL` values are converted to a numeric type and aggregated, except for `MIN()`, `MAX()`, and `GROUP_CONCAT()`. The conversion to number should produce a meaningful result for JSON values that are numeric scalars, although (depending on the values) truncation and loss of precision may occur. Conversion to number of other JSON values may not produce a meaningful result.

11.7 Data Type Default Values

Data type specifications can have explicit or implicit default values.

A `DEFAULT value` clause in a data type specification explicitly indicates a default value for a column. Examples:

```
CREATE TABLE t1 (
  i     INT DEFAULT -1,
  c     VARCHAR(10) DEFAULT '',
  price DOUBLE(16,2) DEFAULT 0.00
);
```

`SERIAL DEFAULT VALUE` is a special case. In the definition of an integer column, it is an alias for `NOT NULL AUTO_INCREMENT UNIQUE`.

Some aspects of explicit `DEFAULT` clause handling are version dependent, as described following.

- Handling of Explicit Defaults as of MySQL 8.0.13
- Handling of Explicit Defaults Prior to MySQL 8.0.13
- Handling of Implicit Defaults

Handling of Explicit Defaults as of MySQL 8.0.13

The default value specified in a `DEFAULT` clause can be a literal constant or an expression. With one exception, enclose expression default values within parentheses to distinguish them from literal constant default values. Examples:

```
CREATE TABLE t1 (
  -- literal defaults
  i INT         DEFAULT 0,
  c VARCHAR(10) DEFAULT '',
  -- expression defaults
  f FLOAT       DEFAULT (RAND() * RAND()),
  b BINARY(16)  DEFAULT (UUID_TO_BIN(UUID())),
  d DATE        DEFAULT (CURRENT_DATE + INTERVAL 1 YEAR),
  p POINT       DEFAULT (Point(0,0)),
  j JSON        DEFAULT (JSON_ARRAY())
);
```

The exception is that, for `TIMESTAMP` and `DATETIME` columns, you can specify the `CURRENT_TIMESTAMP` function as the default, without enclosing parentheses. See Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

The `BLOB`, `TEXT`, `GEOMETRY`, and `JSON` data types can be assigned a default value only if the value is written as an expression, even if the expression value is a literal:

- This is permitted (literal default specified as expression):

```
CREATE TABLE t2 (b BLOB DEFAULT ('abc'));
```

- This produces an error (literal default not specified as expression):

```
CREATE TABLE t2 (b BLOB DEFAULT 'abc');
```

Expression default values must adhere to the following rules. An error occurs if an expression contains disallowed constructs.

- Literals, built-in functions (both deterministic and nondeterministic), and operators are permitted.
- Subqueries, parameters, variables, stored functions, and user-defined functions are not permitted.
- An expression default value cannot depend on a column that has the `AUTO_INCREMENT` attribute.
- An expression default value for one column can refer to other table columns, with the exception that references to generated columns or columns with expression default values must be to columns that occur earlier in the table definition. That is, expression default values cannot contain forward references to generated columns or columns with expression default values.

  The ordering constraint also applies to the use of `ALTER TABLE` to reorder table columns. If the resulting table would have an expression default value that contains a forward reference to a generated column or column with an expression default value, the statement fails.

Note

If any component of an expression default value depends on the SQL mode, different results may occur for different uses of the table unless the SQL mode is the same during all uses.

For `CREATE TABLE ... LIKE` and `CREATE TABLE ... SELECT`, the destination table preserves expression default values from the original table.

If an expression default value refers to a nondeterministic function, any statement that causes the expression to be evaluated is unsafe for statement-based replication. This includes statements such as `INSERT`, `UPDATE`, and `ALTER TABLE`. In this situation, if binary logging is disabled, the statement is executed as normal. If binary logging is enabled and `binlog_format` is set to `STATEMENT`, the statement is logged and executed but a warning message is written to the error log, because replication slaves might diverge. When `binlog_format` is set to `MIXED` or `ROW`, the statement is not executed and an error message is written to the error log.

When inserting a new row, the default value for a column with an expression default can be inserted either by omitting the column name or by specifying the column as `DEFAULT` (just as for columns with literal defaults):

```
mysql> CREATE TABLE t4 (uid BINARY(16) DEFAULT (UUID_TO_BIN(UUID())));
mysql> INSERT INTO t4 () VALUES();
mysql> INSERT INTO t4 () VALUES(DEFAULT);
mysql> SELECT BIN_TO_UUID(uid) AS uid FROM t4;
+--------------------------------------+
| uid                                  |
+--------------------------------------+
| f1109174-94c9-11e8-971d-3bf1095aa633 |
| f110cf9a-94c9-11e8-971d-3bf1095aa633 |
```

```
+-------------------------------------+
```

However, the use of `DEFAULT(col_name)` to specify the default value for a named column is permitted only for columns that have a literal default value, not for columns that have an expression default value.

Not all storage engines permit expression default values. For those that do not, an `ER_UNSUPPORTED_ACTION_ON_DEFAULT_VAL_GENERATED` error occurs.

If a default value evaluates to a data type that differs from the declared column type, implicit coercion to the declared type occurs according to the usual MySQL type-conversion rules. See Section 12.2, “Type Conversion in Expression Evaluation”.

Handling of Explicit Defaults Prior to MySQL 8.0.13

With one exception, the default value specified in a `DEFAULT` clause must be a literal constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as `NOW()` or `CURRENT_DATE`. The exception is that, for `TIMESTAMP` and `DATETIME` columns, you can specify `CURRENT_TIMESTAMP` as the default. See Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

The `BLOB`, `TEXT`, `GEOMETRY`, and `JSON` data types cannot be assigned a default value.

If a default value evaluates to a data type that differs from the declared column type, implicit coercion to the declared type occurs according to the usual MySQL type-conversion rules. See Section 12.2, “Type Conversion in Expression Evaluation”.

Handling of Implicit Defaults

If a data type specification includes no explicit `DEFAULT` value, MySQL determines the default value as follows:

If the column can take `NULL` as a value, the column is defined with an explicit `DEFAULT NULL` clause.

If the column cannot take `NULL` as a value, MySQL defines the column with no explicit `DEFAULT` clause. Exception: If the column is defined as part of a `PRIMARY KEY` but not explicitly as `NOT NULL`, MySQL creates it as a `NOT NULL` column (because `PRIMARY KEY` columns must be `NOT NULL`).

For data entry into a `NOT NULL` column that has no explicit `DEFAULT` clause, if an `INSERT` or `REPLACE` statement includes no value for the column, or an `UPDATE` statement sets the column to `NULL`, MySQL handles the column according to the SQL mode in effect at the time:

- If strict SQL mode is enabled, an error occurs for transactional tables and the statement is rolled back. For nontransactional tables, an error occurs, but if this happens for the second or subsequent row of a multiple-row statement, the preceding rows will have been inserted.
- If strict mode is not enabled, MySQL sets the column to the implicit default value for the column data type.

Suppose that a table `t` is defined as follows:

```
CREATE TABLE t (i INT NOT NULL);
```

In this case, `i` has no explicit default, so in strict mode each of the following statements produce an error and no row is inserted. When not using strict mode, only the third statement produces an error; the implicit

default is inserted for the first two statements, but the third fails because `DEFAULT(i)` cannot produce a value:

```
INSERT INTO t VALUES();
INSERT INTO t VALUES(DEFAULT);
INSERT INTO t VALUES(DEFAULT(i));
```

See Section 5.1.11, “Server SQL Modes”.

For a given table, the `SHOW CREATE TABLE` statement displays which columns have an explicit `DEFAULT` clause.

Implicit defaults are defined as follows:

- For numeric types, the default is `0`, with the exception that for integer or floating-point types declared with the `AUTO_INCREMENT` attribute, the default is the next value in the sequence.
- For date and time types other than `TIMESTAMP`, the default is the appropriate “zero” value for the type. This is also true for `TIMESTAMP` if the `explicit_defaults_for_timestamp` system variable is enabled (see Section 5.1.8, “Server System Variables”). Otherwise, for the first `TIMESTAMP` column in a table, the default value is the current date and time. See Section 11.3, “Date and Time Types”.
- For string types other than `ENUM`, the default value is the empty string. For `ENUM`, the default is the first enumeration value.

11.8 Data Type Storage Requirements

- InnoDB Table Storage Requirements
- NDB Table Storage Requirements
- Numeric Type Storage Requirements
- Date and Time Type Storage Requirements
- String Type Storage Requirements
- Spatial Type Storage Requirements
- JSON Storage Requirements

The storage requirements for table data on disk depend on several factors. Different storage engines represent data types and store raw data differently. Table data might be compressed, either for a column or an entire row, complicating the calculation of storage requirements for a table or column.

Despite differences in storage layout on disk, the internal MySQL APIs that communicate and exchange information about table rows use a consistent data structure that applies across all storage engines.

This section includes guidelines and information for the storage requirements for each data type supported by MySQL, including the internal format and size for storage engines that use a fixed-size representation for data types. Information is listed by category or storage engine.

The internal representation of a table has a maximum row size of 65,535 bytes, even if the storage engine is capable of supporting larger rows. This figure excludes `BLOB` or `TEXT` columns, which contribute only 9 to 12 bytes toward this size. For `BLOB` and `TEXT` data, the information is stored internally in a different area of memory than the row buffer. Different storage engines handle the allocation and storage of this

data in different ways, according to the method they use for handling the corresponding types. For more information, see Chapter 16, *Alternative Storage Engines*, and Section 8.4.7, “Limits on Table Column Count and Row Size”.

InnoDB Table Storage Requirements

See Section 15.10, “InnoDB Row Formats” for information about storage requirements for `InnoDB` tables.

NDB Table Storage Requirements

Important

`NDB` tables use *4-byte alignment*; all `NDB` data storage is done in multiples of 4 bytes. Thus, a column value that would typically take 15 bytes requires 16 bytes in an `NDB` table. For example, in `NDB` tables, the `TINYINT`, `SMALLINT`, `MEDIUMINT`, and `INTEGER` (`INT`) column types each require 4 bytes storage per record due to the alignment factor.

Each `BIT(M)` column takes *`M`* bits of storage space. Although an individual `BIT` column is *not* 4-byte aligned, `NDB` reserves 4 bytes (32 bits) per row for the first 1-32 bits needed for `BIT` columns, then another 4 bytes for bits 33-64, and so on.

While a `NULL` itself does not require any storage space, `NDB` reserves 4 bytes per row if the table definition contains any columns allowing `NULL`, up to 32 `NULL` columns. (If an NDB Cluster table is defined with more than 32 `NULL` columns up to 64 `NULL` columns, then 8 bytes per row are reserved.)

Every table using the `NDB` storage engine requires a primary key; if you do not define a primary key, a “hidden” primary key is created by `NDB`. This hidden primary key consumes 31-35 bytes per table record.

You can use the `ndb_size.pl` Perl script to estimate `NDB` storage requirements. It connects to a current MySQL (not NDB Cluster) database and creates a report on how much space that database would require if it used the `NDB` storage engine. See Section 22.4.28, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator” for more information.

Numeric Type Storage Requirements

Data Type	Storage Required
`TINYINT`	1 byte
`SMALLINT`	2 bytes
`MEDIUMINT`	3 bytes
`INT`, `INTEGER`	4 bytes
`BIGINT`	8 bytes
`FLOAT(`*`p`*`)`	4 bytes if 0 <= *`p`* <= 24, 8 bytes if 25 <= *`p`* <= 53
`FLOAT`	4 bytes
`DOUBLE [PRECISION]`, `REAL`	8 bytes
`DECIMAL(`*`M`*`,`*`D`*`)`, `NUMERIC(`*`M`*`,`*`D`*`)`	Varies; see following discussion
`BIT(`*`M`*`)`	approximately (*`M`*+7)/8 bytes

Values for `DECIMAL` (and `NUMERIC`) columns are represented using a binary format that packs nine decimal (base 10) digits into four bytes. Storage for the integer and fractional parts of each value are

determined separately. Each multiple of nine digits requires four bytes, and the “leftover” digits require some fraction of four bytes. The storage required for excess digits is given by the following table.

Leftover Digits	Number of Bytes
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4

Date and Time Type Storage Requirements

For `TIME`, `DATETIME`, and `TIMESTAMP` columns, the storage required for tables created before MySQL 5.6.4 differs from tables created from 5.6.4 on. This is due to a change in 5.6.4 that permits these types to have a fractional part, which requires from 0 to 3 bytes.

Data Type	Storage Required Before MySQL 5.6.4	Storage Required as of MySQL 5.6.4
`YEAR`	1 byte	1 byte
`DATE`	3 bytes	3 bytes
`TIME`	3 bytes	3 bytes + fractional seconds storage
`DATETIME`	8 bytes	5 bytes + fractional seconds storage
`TIMESTAMP`	4 bytes	4 bytes + fractional seconds storage

As of MySQL 5.6.4, storage for `YEAR` and `DATE` remains unchanged. However, `TIME`, `DATETIME`, and `TIMESTAMP` are represented differently. `DATETIME` is packed more efficiently, requiring 5 rather than 8 bytes for the nonfractional part, and all three parts have a fractional part that requires from 0 to 3 bytes, depending on the fractional seconds precision of stored values.

Fractional Seconds Precision	Storage Required
0	0 bytes
1, 2	1 byte
3, 4	2 bytes
5, 6	3 bytes

For example, `TIME(0)`, `TIME(2)`, `TIME(4)`, and `TIME(6)` use 3, 4, 5, and 6 bytes, respectively. `TIME` and `TIME(0)` are equivalent and require the same storage.

For details about internal representation of temporal values, see MySQL Internals: Important Algorithms and Structures.

String Type Storage Requirements

In the following table, *M* represents the declared column length in characters for nonbinary string types and bytes for binary string types. *L* represents the actual length in bytes of a given string value.

Data Type	Storage Required
CHAR(*M*)	The compact family of InnoDB row formats optimize storage for variable-length character sets. See COMPACT Row Format Storage Characteristics. Otherwise, *M* × *w* bytes, <= *M* <= 255, where *w* is the number of bytes required for the maximum-length character in the character set.
BINARY(*M*)	*M* bytes, 0 <= *M* <= 255
VARCHAR(*M*), VARBINARY(*M*)	*L* + 1 bytes if column values require 0 − 255 bytes, *L* + 2 bytes if values may require more than 255 bytes
TINYBLOB, TINYTEXT	*L* + 1 bytes, where $L < 2^8$
BLOB, TEXT	*L* + 2 bytes, where $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	*L* + 3 bytes, where $L < 2^{24}$
LONGBLOB, LONGTEXT	*L* + 4 bytes, where $L < 2^{32}$
ENUM('*value1*','*value2*',...)	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
SET('*value1*','*value2*',...)	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

Variable-length string types are stored using a length prefix plus data. The length prefix requires from one to four bytes depending on the data type, and the value of the prefix is *L* (the byte length of the string). For example, storage for a MEDIUMTEXT value requires *L* bytes to store the value plus three bytes to store the length of the value.

To calculate the number of bytes used to store a particular CHAR, VARCHAR, or TEXT column value, you must take into account the character set used for that column and whether the value contains multibyte characters. In particular, when using a utf8 Unicode character set, you must keep in mind that not all characters use the same number of bytes. utf8mb3 and utf8mb4 character sets can require up to three and four bytes per character, respectively. For a breakdown of the storage used for different categories of utf8mb3 or utf8mb4 characters, see Section 10.9, “Unicode Support”.

VARCHAR, VARBINARY, and the BLOB and TEXT types are variable-length types. For each, the storage requirements depend on these factors:

- The actual length of the column value
- The column's maximum possible length
- The character set used for the column, because some character sets contain multibyte characters

For example, a VARCHAR(255) column can hold a string with a maximum length of 255 characters. Assuming that the column uses the latin1 character set (one byte per character), the actual storage required is the length of the string (*L*), plus one byte to record the length of the string. For the string 'abcd', *L* is 4 and the storage requirement is five bytes. If the same column is instead declared to use the ucs2 double-byte character set, the storage requirement is 10 bytes: The length of 'abcd' is eight bytes and the column requires two bytes to store lengths because the maximum length is greater than 255 (up to 510 bytes).

The effective maximum number of *bytes* that can be stored in a VARCHAR or VARBINARY column is subject to the maximum row size of 65,535 bytes, which is shared among all columns. For a VARCHAR column that

stores multibyte characters, the effective maximum number of *characters* is less. For example, `utf8mb4` characters can require up to four bytes per character, so a `VARCHAR` column that uses the `utf8mb4` character set can be declared to be a maximum of 16,383 characters. See Section 8.4.7, “Limits on Table Column Count and Row Size”.

`InnoDB` encodes fixed-length fields greater than or equal to 768 bytes in length as variable-length fields, which can be stored off-page. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.

The `NDB` storage engine supports variable-width columns. This means that a `VARCHAR` column in an NDB Cluster table requires the same amount of storage as would any other storage engine, with the exception that such values are 4-byte aligned. Thus, the string `'abcd'` stored in a `VARCHAR(50)` column using the `latin1` character set requires 8 bytes (rather than 5 bytes for the same column value in a `MyISAM` table).

`TEXT` and `BLOB` columns are implemented differently in `NDB`; each row in a `TEXT` column is made up of two separate parts. One of these is of fixed size (256 bytes), and is actually stored in the original table. The other consists of any data in excess of 256 bytes, which is stored in a hidden table. The rows in this second table are always 2000 bytes long. This means that the size of a `TEXT` column is 256 if *`size`* <= 256 (where *`size`* represents the size of the row); otherwise, the size is 256 + *`size`* + (2000 × (*`size`* − 256) % 2000).

The size of an `ENUM` object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations having between 256 and 65,535 possible values. See Section 11.4.4, “The ENUM Type”.

The size of a `SET` object is determined by the number of different set members. If the set size is *`N`*, the object occupies `(`*`N`*`+7)/8` bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A `SET` can have a maximum of 64 members. See Section 11.4.5, “The SET Type”.

Spatial Type Storage Requirements

MySQL stores geometry values using 4 bytes to indicate the SRID followed by the WKB representation of the value. The `LENGTH()` function returns the space in bytes required for value storage.

For descriptions of WKB and internal storage formats for spatial values, see Section 11.5.3, “Supported Spatial Data Formats”.

JSON Storage Requirements

In general, the storage requirement for a `JSON` column is approximately the same as for a `LONGBLOB` or `LONGTEXT` column; that is, the space consumed by a JSON document is roughly the same as it would be for the document's string representation stored in a column of one of these types. However, there is an overhead imposed by the binary encoding, including metadata and dictionaries needed for lookup, of the individual values stored in the JSON document. For example, a string stored in a JSON document requires 4 to 10 bytes additional storage, depending on the length of the string and the size of the object or array in which it is stored.

In addition, MySQL imposes a limit on the size of any JSON document stored in a `JSON` column such that it cannot be any larger than the value of `max_allowed_packet`.

11.9 Choosing the Right Type for a Column

For optimum storage, you should try to use the most precise type in all cases. For example, if an integer column is used for values in the range from `1` to `99999`, `MEDIUMINT UNSIGNED` is the best type. Of the types that represent all the required values, this type uses the least amount of storage.

All basic calculations (`+`, `-`, `*`, and `/`) with `DECIMAL` columns are done with precision of 65 decimal (base 10) digits. See Section 11.1.1, “Numeric Type Overview”.

If accuracy is not too important or if speed is the highest priority, the `DOUBLE` type may be good enough. For high precision, you can always convert to a fixed-point type stored in a `BIGINT`. This enables you to do all calculations with 64-bit integers and then convert results back to floating-point values as necessary.

11.10 Using Data Types from Other Database Engines

To facilitate the use of code written for SQL implementations from other vendors, MySQL maps data types as shown in the following table. These mappings make it easier to import table definitions from other database systems into MySQL.

Other Vendor Type	MySQL Type
`BOOL`	`TINYINT`
`BOOLEAN`	`TINYINT`
`CHARACTER VARYING(M)`	`VARCHAR(M)`
`FIXED`	`DECIMAL`
`FLOAT4`	`FLOAT`
`FLOAT8`	`DOUBLE`
`INT1`	`TINYINT`
`INT2`	`SMALLINT`
`INT3`	`MEDIUMINT`
`INT4`	`INT`
`INT8`	`BIGINT`
`LONG VARBINARY`	`MEDIUMBLOB`
`LONG VARCHAR`	`MEDIUMTEXT`
`LONG`	`MEDIUMTEXT`
`MIDDLEINT`	`MEDIUMINT`
`NUMERIC`	`DECIMAL`

Data type mapping occurs at table creation time, after which the original type specifications are discarded. If you create a table with types used by other vendors and then issue a `DESCRIBE tbl_name` statement, MySQL reports the table structure using the equivalent MySQL types. For example:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);
Query OK, 0 rows affected (0.00 sec)

mysql> DESCRIBE t;
+-------+---------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| a     | tinyint(1)    | YES  |     | NULL    |       |
| b     | double        | YES  |     | NULL    |       |
| c     | mediumtext    | YES  |     | NULL    |       |
| d     | decimal(10,0) | YES  |     | NULL    |       |
+-------+---------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Chapter 12 Functions and Operators

Table of Contents

12.1 Function and Operator Reference 1901
12.2 Type Conversion in Expression Evaluation 1914
12.3 Operators 1917
12.3.1 Operator Precedence 1918
12.3.2 Comparison Functions and Operators 1919
12.3.3 Logical Operators 1926
12.3.4 Assignment Operators 1928
12.4 Control Flow Functions 1929
12.5 String Functions and Operators 1932
12.5.1 String Comparison Functions and Operators 1948
12.5.2 Regular Expressions 1951
12.5.3 Character Set and Collation of Function Results 1962
12.6 Numeric Functions and Operators 1963
12.6.1 Arithmetic Operators 1964
12.6.2 Mathematical Functions 1966
12.7 Date and Time Functions 1975
12.8 What Calendar Is Used By MySQL? 1997
12.9 Full-Text Search Functions 1997
12.9.1 Natural Language Full-Text Searches 1999
12.9.2 Boolean Full-Text Searches 2002
12.9.3 Full-Text Searches with Query Expansion 2008
12.9.4 Full-Text Stopwords 2009
12.9.5 Full-Text Restrictions 2013
12.9.6 Fine-Tuning MySQL Full-Text Search 2014
12.9.7 Adding a Collation for Full-Text Indexing 2017
12.9.8 ngram Full-Text Parser 2019
12.9.9 MeCab Full-Text Parser Plugin 2021
12.10 Cast Functions and Operators 2026
12.11 XML Functions 2032
12.12 Bit Functions and Operators 2043
12.13 Encryption and Compression Functions 2055
12.14 Locking Functions 2062
12.15 Information Functions 2065
12.16 Spatial Analysis Functions 2076
12.16.1 Spatial Function Reference 2076
12.16.2 Argument Handling by Spatial Functions 2079
12.16.3 Functions That Create Geometry Values from WKT Values 2080
12.16.4 Functions That Create Geometry Values from WKB Values 2082
12.16.5 MySQL-Specific Functions That Create Geometry Values 2084
12.16.6 Geometry Format Conversion Functions 2085
12.16.7 Geometry Property Functions 2087
12.16.8 Spatial Operator Functions 2100
12.16.9 Functions That Test Spatial Relations Between Geometry Objects 2104
12.16.10 Spatial Geohash Functions 2110
12.16.11 Spatial GeoJSON Functions 2112
12.16.12 Spatial Convenience Functions 2114
12.17 JSON Functions 2118
12.17.1 JSON Function Reference 2118

12.17.2 Functions That Create JSON Values 2120
12.17.3 Functions That Search JSON Values 2121
12.17.4 Functions That Modify JSON Values 2134
12.17.5 Functions That Return JSON Value Attributes 2143
12.17.6 JSON Table Functions 2146
12.17.7 JSON Schema Validation Functions 2150
12.17.8 JSON Utility Functions 2155
12.18 Functions Used with Global Transaction Identifiers (GTIDs) 2161
12.19 MySQL Enterprise Encryption Functions 2163
12.19.1 MySQL Enterprise Encryption Installation 2163
12.19.2 MySQL Enterprise Encryption Usage and Examples 2164
12.19.3 MySQL Enterprise Encryption Function Reference 2166
12.19.4 MySQL Enterprise Encryption Function Descriptions 2167
12.20 Aggregate (GROUP BY) Functions 2171
12.20.1 Aggregate (GROUP BY) Function Descriptions 2171
12.20.2 GROUP BY Modifiers 2181
12.20.3 MySQL Handling of GROUP BY 2187
12.20.4 Detection of Functional Dependence 2190
12.21 Window Functions 2194
12.21.1 Window Function Descriptions 2194
12.21.2 Window Function Concepts and Syntax 2200
12.21.3 Window Function Frame Specification 2203
12.21.4 Named Windows 2207
12.21.5 Window Function Restrictions 2209
12.22 Performance Schema Functions 2209
12.23 Internal Functions 2212
12.24 Miscellaneous Functions 2213
12.25 Precision Math 2228
12.25.1 Types of Numeric Values 2229
12.25.2 DECIMAL Data Type Characteristics 2229
12.25.3 Expression Handling 2230
12.25.4 Rounding Behavior 2232
12.25.5 Precision Math Examples 2233

Expressions can be used at several points in SQL statements, such as in the `ORDER BY` or `HAVING` clauses of `SELECT` statements, in the `WHERE` clause of a `SELECT`, `DELETE`, or `UPDATE` statement, or in `SET` statements. Expressions can be written using literal values, column values, `NULL`, built-in functions, stored functions, user-defined functions, and operators. This chapter describes the functions and operators that are permitted for writing expressions in MySQL. Instructions for writing stored functions and user-defined functions are given in Section 24.2, “Using Stored Routines”, and Section 29.4, “Adding New Functions to MySQL”. See Section 9.2.5, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for a particular function or operator.

Note

By default, there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function calls and references to tables or columns that happen to have the same name as a function. However, spaces around function arguments are permitted.

You can tell the MySQL server to accept spaces after function names by starting it with the `--sql-mode=IGNORE_SPACE` option. (See Section 5.1.11, “Server SQL Modes”.) Individual client programs can

request this behavior by using the `CLIENT_IGNORE_SPACE` option for `mysql_real_connect()`. In either case, all function names become reserved words.

For the sake of brevity, most examples in this chapter display the output from the `mysql` program in abbreviated form. Rather than showing examples in this format:

```
mysql> SELECT MOD(29,9);
+-----------+
| mod(29,9) |
+-----------+
|         2 |
+-----------+
1 rows in set (0.00 sec)
```

This format is used instead:

```
mysql> SELECT MOD(29,9);
       -> 2
```

12.1 Function and Operator Reference

Table 12.1 Functions and Operators

Name	Description
`ABS()`	Return the absolute value
`ACOS()`	Return the arc cosine
`ADDDATE()`	Add time values (intervals) to a date value
`ADDTIME()`	Add time
`AES_DECRYPT()`	Decrypt using AES
`AES_ENCRYPT()`	Encrypt using AES
`AND`, `&&`	Logical AND
`ANY_VALUE()`	Suppress ONLY_FULL_GROUP_BY value rejection
`ASCII()`	Return numeric value of left-most character
`ASIN()`	Return the arc sine
`=`	Assign a value (as part of a `SET` statement, or as part of the `SET` clause in an `UPDATE` statement)
`:=`	Assign a value
`ASYMMETRIC_DECRYPT()`	Decrypt ciphertext using private or public key
`ASYMMETRIC_DERIVE()`	Derive symmetric key from asymmetric keys
`ASYMMETRIC_ENCRYPT()`	Encrypt cleartext using private or public key
`ASYMMETRIC_SIGN()`	Generate signature from digest
`ASYMMETRIC_VERIFY()`	Verify that signature matches digest
`ATAN()`	Return the arc tangent
`ATAN2()`, `ATAN()`	Return the arc tangent of the two arguments
`AVG()`	Return the average value of the argument
`BENCHMARK()`	Repeatedly execute an expression
`BETWEEN ... AND ...`	Whether a value is within a range of values

Name	Description
BIN()	Return a string containing binary representation of a number
BIN_TO_UUID()	Convert binary UUID to string
BINARY	Cast a string to a binary string
BIT_AND()	Return bitwise AND
BIT_COUNT()	Return the number of bits that are set
BIT_LENGTH()	Return length of argument in bits
BIT_OR()	Return bitwise OR
BIT_XOR()	Return bitwise XOR
&	Bitwise AND
~	Bitwise inversion
|	Bitwise OR
^	Bitwise XOR
CAN_ACCESS_COLUMN()	Internal use only
CAN_ACCESS_DATABASE()	Internal use only
CAN_ACCESS_TABLE()	Internal use only
CAN_ACCESS_VIEW()	Internal use only
CASE	Case operator
CAST()	Cast a value as a certain type
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
CHAR()	Return the character for each integer passed
CHAR_LENGTH()	Return number of characters in argument
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CHARSET()	Return the character set of the argument
COALESCE()	Return the first non-NULL argument
COERCIBILITY()	Return the collation coercibility value of the string argument
COLLATION()	Return the collation of the string argument
COMPRESS()	Return result as a binary string
CONCAT()	Return concatenated string
CONCAT_WS()	Return concatenate with separator
CONNECTION_ID()	Return the connection ID (thread ID) for the connection
CONV()	Convert numbers between different number bases
CONVERT()	Cast a value as a certain type
CONVERT_TZ()	Convert from one time zone to another
COS()	Return the cosine
COT()	Return the cotangent
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values

Name	Description
CRC32()	Compute a cyclic redundancy check value
CREATE_ASYMMETRIC_PRIV_KEY()	Create private key
CREATE_ASYMMETRIC_PUB_KEY()	Create public key
CREATE_DH_PARAMETERS()	Generate shared DH secret
CREATE_DIGEST()	Generate digest from string
CUME_DIST()	Cumulative distribution value
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_ROLE()	Return the current active roles
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURRENT_USER(), CURRENT_USER	The authenticated user name and host name
CURTIME()	Return the current time
DATABASE()	Return the default (current) database name
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
DECODE()	Decode a string encrypted using ENCODE()
DEFAULT()	Return the default value for a table column
DEGREES()	Convert radians to degrees
DENSE_RANK()	Rank of current row within its partition, without gaps
DES_DECRYPT()	Decrypt a string
DES_ENCRYPT()	Encrypt a string
DIV	Integer division
/	Division operator
ELT()	Return string at index number
ENCODE()	Encode a string
ENCRYPT()	Encrypt a string
=	Equal operator
<=>	NULL-safe equal to operator

Name	Description
EXP()	Raise to the power of
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
EXTRACT()	Extract part of a date
ExtractValue()	Extract a value from an XML string using XPath notation
FIELD()	Index (position) of first argument in subsequent arguments
FIND_IN_SET()	Index (position) of first argument within second argument
FIRST_VALUE()	Value of argument from first row of window frame
FLOOR()	Return the largest integer value not greater than the argument
FORMAT()	Return a number formatted to specified number of decimal places
FORMAT_BYTES()	Convert byte count to value with units
FORMAT_PICO_TIME()	Convert time in picoseconds to value with units
FOUND_ROWS()	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
FROM_BASE64()	Decode base64 encoded string and return result
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format Unix timestamp as a date
GeomCollection()	Construct geometry collection from geometries
GeometryCollection()	Construct geometry collection from geometries
GET_DD_COLUMN_PRIVILEGES()	Internal use only
GET_DD_CREATE_OPTIONS()	Internal use only
GET_DD_INDEX_SUB_PART_LENGTH()	Internal use only
GET_FORMAT()	Return a date format string
GET_LOCK()	Get a named lock
>	Greater than operator
>=	Greater than or equal operator
GREATEST()	Return the largest argument
GROUP_CONCAT()	Return a concatenated string
GROUPING()	Distinguish super-aggregate ROLLUP rows from regular rows
GTID_SUBSET()	Return true if all GTIDs in subset are also in set; otherwise false.
GTID_SUBTRACT()	Return all GTIDs in set that are not in subset.
HEX()	Hexadecimal representation of decimal or string value
HOUR()	Extract the hour
ICU_VERSION()	ICU library version
IF()	If/else construct
IFNULL()	Null if/else construct
IN()	Whether a value is within a set of values

Name	Description
INET_ATON()	Return the numeric value of an IP address
INET_NTOA()	Return the IP address from a numeric value
INET6_ATON()	Return the numeric value of an IPv6 address
INET6_NTOA()	Return the IPv6 address from a numeric value
INSERT()	Insert substring at specified position up to specified number of characters
INSTR()	Return the index of the first occurrence of substring
INTERNAL_AUTO_INCREMENT()	Internal use only
INTERNAL_AVG_ROW_LENGTH()	Internal use only
INTERNAL_CHECK_TIME()	Internal use only
INTERNAL_CHECKSUM()	Internal use only
INTERNAL_DATA_FREE()	Internal use only
INTERNAL_DATA_LENGTH()	Internal use only
INTERNAL_DD_CHAR_LENGTH()	Internal use only
INTERNAL_GET_COMMENT_OR_ERROR()	Internal use only
INTERNAL_GET_VIEW_WARNING_OR_ERROR()	Internal use only
INTERNAL_INDEX_COLUMN_CARDINALITY()	Internal use only
INTERNAL_INDEX_LENGTH()	Internal use only
INTERNAL_KEYS_DISABLED()	Internal use only
INTERNAL_MAX_DATA_LENGTH()	Internal use only
INTERNAL_TABLE_ROWS()	Internal use only
INTERNAL_UPDATE_TIME()	Internal use only
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS_FREE_LOCK()	Whether the named lock is free
IS_IPV4()	Whether argument is an IPv4 address
IS_IPV4_COMPAT()	Whether argument is an IPv4-compatible address
IS_IPV4_MAPPED()	Whether argument is an IPv4-mapped address
IS_IPV6()	Whether argument is an IPv6 address
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
IS_USED_LOCK()	Whether the named lock is in use; return connection identifier if true
IS_UUID()	Whether argument is a valid UUID
ISNULL()	Test whether the argument is NULL
JSON_ARRAY()	Create JSON array
JSON_ARRAY_APPEND()	Append data to JSON document

Name	Description
JSON_ARRAY_INSERT()	Insert into JSON array
JSON_ARRAYAGG()	Return result set as a single JSON array
->	Return value from JSON column after evaluating path; equivalent to JSON_EXTRACT().
JSON_CONTAINS()	Whether JSON document contains specific object at path
JSON_CONTAINS_PATH()	Whether JSON document contains any data at path
JSON_DEPTH()	Maximum depth of JSON document
JSON_EXTRACT()	Return data from JSON document
->>	Return value from JSON column after evaluating path and unquoting the result; equivalent to JSON_UNQUOTE(JSON_EXTRACT()).
JSON_INSERT()	Insert data into JSON document
JSON_KEYS()	Array of keys from JSON document
JSON_LENGTH()	Number of elements in JSON document
JSON_MERGE() (deprecated 8.0.3)	Merge JSON documents, preserving duplicate keys. Deprecated synonym for JSON_MERGE_PRESERVE()
JSON_MERGE_PATCH()	Merge JSON documents, replacing values of duplicate keys
JSON_MERGE_PRESERVE()	Merge JSON documents, preserving duplicate keys
JSON_OBJECT()	Create JSON object
JSON_OBJECTAGG()	Return result set as a single JSON object
JSON_OVERLAPS()	Compares two JSON documents, returns TRUE (1) if these have any key-value pairs or array elements in common, otherwise FALSE (0)
JSON_PRETTY()	Print a JSON document in human-readable format
JSON_QUOTE()	Quote JSON document
JSON_REMOVE()	Remove data from JSON document
JSON_REPLACE()	Replace values in JSON document
JSON_SCHEMA_VALID()	Validate JSON document against JSON schema; returns TRUE/1 if document validates against schema, or FALSE/0 if it does not
JSON_SCHEMA_VALIDATION_REPORT()	Validate JSON document against JSON schema; returns report in JSON format on outcome on validation including success or failure and reasons for failure
JSON_SEARCH()	Path to value within JSON document
JSON_SET()	Insert data into JSON document
JSON_STORAGE_FREE()	Freed space within binary representation of JSON column value following partial update
JSON_STORAGE_SIZE()	Space used for storage of binary representation of a JSON document
JSON_TABLE()	Return data from a JSON expression as a relational table
JSON_TYPE()	Type of JSON value

Name	Description
JSON_UNQUOTE()	Unquote JSON value
JSON_VALID()	Whether JSON value is valid
LAG()	Value of argument from row lagging current row within partition
LAST_DAY	Return the last day of the month for the argument
LAST_INSERT_ID()	Value of the AUTOINCREMENT column for the last INSERT
LAST_VALUE()	Value of argument from last row of window frame
LCASE()	Synonym for LOWER()
LEAD()	Value of argument from row leading current row within partition
LEAST()	Return the smallest argument
LEFT()	Return the leftmost number of characters as specified
<<	Left shift
LENGTH()	Return the length of a string in bytes
<	Less than operator
<=	Less than or equal operator
LIKE	Simple pattern matching
LineString()	Construct LineString from Point values
LN()	Return the natural logarithm of the argument
LOAD_FILE()	Load the named file
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
LOCATE()	Return the position of the first occurrence of substring
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
MATCH	Perform full-text search
MAX()	Return the maximum value
MBRContains()	Whether MBR of one geometry contains MBR of another
MBRCoveredBy()	Whether one MBR is covered by another

Name	Description
MBRCovers()	Whether one MBR covers another
MBRDisjoint()	Whether MBRs of two geometries are disjoint
MBREquals()	Whether MBRs of two geometries are equal
MBRIntersects()	Whether MBRs of two geometries intersect
MBROverlaps()	Whether MBRs of two geometries overlap
MBRTouches()	Whether MBRs of two geometries touch
MBRWithin()	Whether MBR of one geometry is within MBR of another
MD5()	Calculate MD5 checksum
MEMBER OF()	Returns true (1) if first operand matches any element of JSON array passed as second operand, otherwise returns false (0)
MICROSECOND()	Return the microseconds from argument
MID()	Return a substring starting from the specified position
MIN()	Return the minimum value
-	Minus operator
MINUTE()	Return the minute from the argument
MOD()	Return the remainder
%, MOD	Modulo operator
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
MultiLineString()	Contruct MultiLineString from LineString values
MultiPoint()	Construct MultiPoint from Point values
MultiPolygon()	Construct MultiPolygon from Polygon values
NAME_CONST()	Cause the column to have the given name
NOT, !	Negates value
NOT BETWEEN ... AND ...	Whether a value is not within a range of values
!=, <>	Not equal operator
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
NOW()	Return the current date and time
NTH_VALUE()	Value of argument from N-th row of window frame
NTILE()	Bucket number of current row within its partition.
NULLIF()	Return NULL if expr1 = expr2
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
OR, ||	Logical OR
ORD()	Return character code for leftmost character of the argument
PASSWORD()	Calculate and return a password string

Name	Description
PERCENT_RANK()	Percentage rank value
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
PI()	Return the value of pi
+	Addition operator
Point()	Construct Point from coordinates
Polygon()	Construct Polygon from LineString arguments
POSITION()	Synonym for LOCATE()
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
PS_CURRENT_THREAD_ID()	Performance Schema thread ID for current thread
PS_THREAD_ID()	Performance Schema thread ID for given thread
QUARTER()	Return the quarter from a date argument
QUOTE()	Escape the argument for use in an SQL statement
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
RANDOM_BYTES()	Return a random byte vector
RANK()	Rank of current row within its partition, with gaps
REGEXP	Whether string matches regular expression
REGEXP_INSTR()	Starting index of substring matching regular expression
REGEXP_LIKE()	Whether string matches regular expression
REGEXP_REPLACE()	Replace substrings matching regular expression
REGEXP_SUBSTR()	Return substring matching regular expression
RELEASE_ALL_LOCKS()	Release all current named locks
RELEASE_LOCK()	Release the named lock
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
>>	Right shift
RLIKE	Whether string matches regular expression
ROLES_GRAPHML()	Return a GraphML document representing memory role subgraphs
ROUND()	Round the argument
ROW_COUNT()	The number of rows updated
ROW_NUMBER()	Number of current row within its partition
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces

Name	Description
SCHEMA()	Synonym for DATABASE()
SEC_TO_TIME()	Converts seconds to 'hh:mm:ss' format
SECOND()	Return the second (0-59)
SESSION_USER()	Synonym for USER()
SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
SHA2()	Calculate an SHA-2 checksum
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SLEEP()	Sleep for a number of seconds
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds
SPACE()	Return a string of the specified number of spaces
SQRT()	Return the square root of the argument
ST_Area()	Return Polygon or MultiPolygon area
ST_AsBinary(), ST_AsWKB()	Convert from internal geometry format to WKB
ST_AsGeoJSON()	Generate GeoJSON object from geometry
ST_AsText(), ST_AsWKT()	Convert from internal geometry format to WKT
ST_Buffer()	Return geometry of points within given distance from geometry
ST_Buffer_Strategy()	Produce strategy option for ST_Buffer()
ST_Centroid()	Return centroid as a point
ST_Contains()	Whether one geometry contains another
ST_ConvexHull()	Return convex hull of geometry
ST_Crosses()	Whether one geometry crosses another
ST_Difference()	Return point set difference of two geometries
ST_Dimension()	Dimension of geometry
ST_Disjoint()	Whether one geometry is disjoint from another
ST_Distance()	The distance of one geometry from another
ST_Distance_Sphere()	Minimum distance on earth between two geometries
ST_EndPoint()	End Point of LineString
ST_Envelope()	Return MBR of geometry
ST_Equals()	Whether one geometry is equal to another
ST_ExteriorRing()	Return exterior ring of Polygon
ST_GeoHash()	Produce a geohash value
ST_GeomCollFromText(), ST_GeometryCollectionFromText(), ST_GeomCollFromTxt()	Return geometry collection from WKT
ST_GeomCollFromWKB(), ST_GeometryCollectionFromWKB()	Return geometry collection from WKB
ST_GeometryN()	Return N-th geometry from geometry collection

Name	Description
ST_GeometryType()	Return name of geometry type
ST_GeomFromGeoJSON()	Generate geometry from GeoJSON object
ST_GeomFromText(), ST_GeometryFromText()	Return geometry from WKT
ST_GeomFromWKB(), ST_GeometryFromWKB()	Return geometry from WKB
ST_InteriorRingN()	Return N-th interior ring of Polygon
ST_Intersection()	Return point set intersection of two geometries
ST_Intersects()	Whether one geometry intersects another
ST_IsClosed()	Whether a geometry is closed and simple
ST_IsEmpty()	Placeholder function
ST_IsSimple()	Whether a geometry is simple
ST_IsValid()	Whether a geometry is valid
ST_LatFromGeoHash()	Return latitude from geohash value
ST_Latitude()	Return latitude of Point
ST_Length()	Return length of LineString
ST_LineFromText(), ST_LineStringFromText()	Construct LineString from WKT
ST_LineFromWKB(), ST_LineStringFromWKB()	Construct LineString from WKB
ST_LongFromGeoHash()	Return longitude from geohash value
ST_Longitude()	Return longitude of Point
ST_MakeEnvelope()	Rectangle around two points
ST_MLineFromText(), ST_MultiLineStringFromText()	Construct MultiLineString from WKT
ST_MLineFromWKB(), ST_MultiLineStringFromWKB()	Construct MultiLineString from WKB
ST_MPointFromText(), ST_MultiPointFromText()	Construct MultiPoint from WKT
ST_MPointFromWKB(), ST_MultiPointFromWKB()	Construct MultiPoint from WKB
ST_MPolyFromText(), ST_MultiPolygonFromText()	Construct MultiPolygon from WKT
ST_MPolyFromWKB(), ST_MultiPolygonFromWKB()	Construct MultiPolygon from WKB
ST_NumGeometries()	Return number of geometries in geometry collection
ST_NumInteriorRing(), ST_NumInteriorRings()	Return number of interior rings in Polygon
ST_NumPoints()	Return number of points in LineString
ST_Overlaps()	Whether one geometry overlaps another
ST_PointFromGeoHash()	Convert geohash value to POINT value

Name	Description
ST_PointFromText()	Construct Point from WKT
ST_PointFromWKB()	Construct Point from WKB
ST_PointN()	Return N-th point from LineString
ST_PolyFromText(), ST_PolygonFromText()	Construct Polygon from WKT
ST_PolyFromWKB(), ST_PolygonFromWKB()	Construct Polygon from WKB
ST_Simplify()	Return simplified geometry
ST_SRID()	Return spatial reference system ID for geometry
ST_StartPoint()	Start Point of LineString
ST_SwapXY()	Return argument with X/Y coordinates swapped
ST_SymDifference()	Return point set symmetric difference of two geometries
ST_Touches()	Whether one geometry touches another
ST_Transform()	Transform coordinates of geometry
ST_Union()	Return point set union of two geometries
ST_Validate()	Return validated geometry
ST_Within()	Whether one geometry is within another
ST_X()	Return X coordinate of Point
ST_Y()	Return Y coordinate of Point
STATEMENT_DIGEST()	Compute statement digest hash value
STATEMENT_DIGEST_TEXT()	Compute normalized statement digest
STD()	Return the population standard deviation
STDDEV()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
STR_TO_DATE()	Convert a string to a date
STRCMP()	Compare two strings
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBSTR()	Return the substring as specified
SUBSTRING()	Return the substring as specified
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
SUBTIME()	Subtract times
SUM()	Return the sum
SYSDATE()	Return the time at which the function executes
SYSTEM_USER()	Synonym for USER()
TAN()	Return the tangent of the argument
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time

Name	Description
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
*	Multiplication operator
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_BASE64()	Return the argument converted to a base-64 string
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
TRIM()	Remove leading and trailing spaces
TRUNCATE()	Truncate to specified number of decimal places
UCASE()	Synonym for UPPER()
-	Change the sign of the argument
UNCOMPRESS()	Uncompress a string compressed
UNCOMPRESSED_LENGTH()	Return the length of a string before compression
UNHEX()	Return a string containing hex representation of a number
UNIX_TIMESTAMP()	Return a Unix timestamp
UpdateXML()	Return replaced XML fragment
UPPER()	Convert to uppercase
USER()	The user name and host name provided by the client
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
UUID()	Return a Universal Unique Identifier (UUID)
UUID_SHORT()	Return an integer-valued universal identifier
UUID_TO_BIN()	Convert string UUID to binary
VALIDATE_PASSWORD_STRENGTH()	Determine strength of password
VALUES()	Define the values to be used during an INSERT
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance
VERSION()	Return a string that indicates the MySQL server version
WAIT_FOR_EXECUTED_GTID_SET()	Wait until the given GTIDs have executed on the slave.
WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS() (deprecated 8.0.18)	Deprecated. Use WAIT_FOR_EXECUTED_GTID_SET().
WEEK()	Return the week number

Name	Description
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
WEIGHT_STRING()	Return the weight string for a string
XOR	Logical XOR
YEAR()	Return the year
YEARWEEK()	Return the year and week

12.2 Type Conversion in Expression Evaluation

When an operator is used with operands of different types, type conversion occurs to make the operands compatible. Some conversions occur implicitly. For example, MySQL automatically converts strings to numbers as necessary, and vice versa.

```
mysql> SELECT 1+'1';
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

It is also possible to convert a number to a string explicitly using the CAST() function. Conversion occurs implicitly with the CONCAT() function because it expects string arguments.

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
        -> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
        -> 38.8, '38.8'
```

See later in this section for information about the character set of implicit number-to-string conversions, and for modified rules that apply to CREATE TABLE ... SELECT statements.

The following rules describe how conversion occurs for comparison operations:

- If one or both arguments are NULL, the result of the comparison is NULL, except for the NULL-safe <=> equality comparison operator. For NULL <=> NULL, the result is true. No conversion is needed.
- If both arguments in a comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- Hexadecimal values are treated as binary strings if not compared to a number.
- If one of the arguments is a TIMESTAMP or DATETIME column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more ODBC-friendly. This is not done for the arguments to IN(). To be safe, always use complete datetime, date, or time strings when doing comparisons. For example, to achieve best results when using BETWEEN with date or time values, use CAST() to explicitly convert the values to the desired data type.

  A single-row subquery from a table or tables is not considered a constant. For example, if a subquery returns an integer to be compared to a DATETIME value, the comparison is done as two integers. The integer is not converted to a temporal value. To compare the operands as DATETIME values, use CAST() to explicitly convert the subquery value to DATETIME.

- If one of the arguments is a decimal value, comparison depends on the other argument. The arguments are compared as decimal values if the other argument is a decimal or integer value, or as floating-point values if the other argument is a floating-point value.
- In all other cases, the arguments are compared as floating-point (real) numbers. For example, a comparison of string and numeric operands takes places as a comparison of floating-point numbers.

For information about conversion of values from one temporal type to another, see Section 11.3.6, “Conversion Between Date and Time Types”.

Comparison of JSON values takes place at two levels. The first level of comparison is based on the JSON types of the compared values. If the types differ, the comparison result is determined solely by which type has higher precedence. If the two values have the same JSON type, a second level of comparison occurs using type-specific rules. For comparison of JSON and non-JSON values, the non-JSON value is converted to JSON and the values compared as JSON values. For details, see Comparison and Ordering of JSON Values.

The following examples illustrate conversion of strings to numbers for comparison operations:

```
mysql> SELECT 1 > '6x';
        -> 0
mysql> SELECT 7 > '6x';
        -> 1
mysql> SELECT 0 > 'x6';
        -> 0
mysql> SELECT 0 = 'x6';
        -> 1
```

For comparisons of a string column with a number, MySQL cannot use an index on the column to look up the value quickly. If *`str_col`* is an indexed string column, the index cannot be used when performing the lookup in the following statement:

```
SELECT * FROM tbl_name WHERE str_col=1;
```

The reason for this is that there are many different strings that may convert to the value `1`, such as `'1'`, `' 1'`, or `'1a'`.

Comparisons that use floating-point numbers (or values that are converted to floating-point numbers) are approximate because such numbers are inexact. This might lead to results that appear inconsistent:

```
mysql> SELECT '18015376320243458' = 18015376320243458;
        -> 1
mysql> SELECT '18015376320243459' = 18015376320243459;
        -> 0
```

Such results can occur because the values are converted to floating-point numbers, which have only 53 bits of precision and are subject to rounding:

```
mysql> SELECT '18015376320243459'+0.0;
        -> 1.8015376320243e+16
```

Furthermore, the conversion from string to floating-point and from integer to floating-point do not necessarily occur the same way. The integer may be converted to floating-point by the CPU, whereas the string is converted digit by digit in an operation that involves floating-point multiplications.

The results shown will vary on different systems, and can be affected by factors such as computer architecture or the compiler version or optimization level. One way to avoid such problems is to use `CAST()` so that a value is not converted implicitly to a float-point number:

```
mysql> SELECT CAST('18015376320243459' AS UNSIGNED) = 18015376320243459;
        -> 1
```

For more information about floating-point comparisons, see Section B.4.4.8, “Problems with Floating-Point Values”.

The server includes `dtoa`, a conversion library that provides the basis for improved conversion between string or `DECIMAL` values and approximate-value (`FLOAT`/`DOUBLE`) numbers:

- Consistent conversion results across platforms, which eliminates, for example, Unix versus Windows conversion differences.
- Accurate representation of values in cases where results previously did not provide sufficient precision, such as for values close to IEEE limits.
- Conversion of numbers to string format with the best possible precision. The precision of `dtoa` is always the same or better than that of the standard C library functions.

Because the conversions produced by this library differ in some cases from non-`dtoa` results, the potential exists for incompatibilities in applications that rely on previous results. For example, applications that depend on a specific exact result from previous conversions might need adjustment to accommodate additional precision.

The `dtoa` library provides conversions with the following properties. *`D`* represents a value with a `DECIMAL` or string representation, and *`F`* represents a floating-point number in native binary (IEEE) format.

- *`F`* -> *`D`* conversion is done with the best possible precision, returning *`D`* as the shortest string that yields *`F`* when read back in and rounded to the nearest value in native binary format as specified by IEEE.
- *`D`* -> *`F`* conversion is done such that *`F`* is the nearest native binary number to the input decimal string *`D`*.

These properties imply that *`F`* -> *`D`* -> *`F`* conversions are lossless unless *`F`* is `-inf`, `+inf`, or `NaN`. The latter values are not supported because the SQL standard defines them as invalid values for `FLOAT` or `DOUBLE`.

For *`D`* -> *`F`* -> *`D`* conversions, a sufficient condition for losslessness is that *`D`* uses 15 or fewer digits of precision, is not a denormal value, `-inf`, `+inf`, or `NaN`. In some cases, the conversion is lossless even if *`D`* has more than 15 digits of precision, but this is not always the case.

Implicit conversion of a numeric or temporal value to string produces a value that has a character set and collation determined by the `character_set_connection` and `collation_connection` system variables. (These variables commonly are set with `SET NAMES`. For information about connection character sets, see Section 10.4, “Connection Character Sets and Collations”.)

This means that such a conversion results in a character (nonbinary) string (a `CHAR`, `VARCHAR`, or `LONGTEXT` value), except in the case that the connection character set is set to `binary`. In that case, the conversion result is a binary string (a `BINARY`, `VARBINARY`, or `LONGBLOB` value).

For integer expressions, the preceding remarks about expression *evaluation* apply somewhat differently for expression *assignment*; for example, in a statement such as this:

```
CREATE TABLE t SELECT integer_expr;
```

In this case, the table in the column resulting from the expression has type `INT` or `BIGINT` depending on the length of the integer expression. If the maximum length of the expression does not fit in an `INT`, `BIGINT` is used instead. The length is taken from the `max_length` value of the `SELECT` result set

metadata (see Section 28.7.5, “C API Data Structures”). This means that you can force a `BIGINT` rather than `INT` by use of a sufficiently long expression:

```
CREATE TABLE t SELECT 00000000000000000000;
```

12.3 Operators

Table 12.2 Operators

Name	Description
`AND`, `&&`	Logical AND
`=`	Assign a value (as part of a `SET` statement, or as part of the `SET` clause in an `UPDATE` statement)
`:=`	Assign a value
`BETWEEN ... AND ...`	Whether a value is within a range of values
`BINARY`	Cast a string to a binary string
`&`	Bitwise AND
`~`	Bitwise inversion
`|`	Bitwise OR
`^`	Bitwise XOR
`CASE`	Case operator
`DIV`	Integer division
`/`	Division operator
`=`	Equal operator
`<=>`	NULL-safe equal to operator
`>`	Greater than operator
`>=`	Greater than or equal operator
`IN()`	Whether a value is within a set of values
`IS`	Test a value against a boolean
`IS NOT`	Test a value against a boolean
`IS NOT NULL`	NOT NULL value test
`IS NULL`	NULL value test
`->`	Return value from JSON column after evaluating path; equivalent to JSON_EXTRACT().
`->>`	Return value from JSON column after evaluating path and unquoting the result; equivalent to JSON_UNQUOTE(JSON_EXTRACT()).
`<<`	Left shift
`<`	Less than operator
`<=`	Less than or equal operator
`LIKE`	Simple pattern matching
`MEMBER OF()`	Returns true (1) if first operand matches any element of JSON array passed as second operand, otherwise returns false (0)

Name	Description
-	Minus operator
%, MOD	Modulo operator
NOT, !	Negates value
NOT BETWEEN ... AND ...	Whether a value is not within a range of values
!=, <>	Not equal operator
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OR, ||	Logical OR
+	Addition operator
REGEXP	Whether string matches regular expression
>>	Right shift
RLIKE	Whether string matches regular expression
SOUNDS LIKE	Compare sounds
*	Multiplication operator
-	Change the sign of the argument
XOR	Logical XOR

12.3.1 Operator Precedence

Operator precedences are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

```
INTERVAL
BINARY, COLLATE
!
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
= (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
AND, &&
XOR
OR, ||
= (assignment), :=
```

The precedence of = depends on whether it is used as a comparison operator (=) or as an assignment operator (=). When used as a comparison operator, it has the same precedence as <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, and IN(). When used as an assignment operator, it has the same precedence as :=. Section 13.7.6.1, “SET Syntax for Variable Assignment”, and Section 9.4, “User-Defined Variables”, explain how MySQL determines which interpretation of = should apply.

For operators that occur at the same precedence level within an expression, evaluation proceeds left to right, with the exception that assignments evaluate right to left.

The precedence and meaning of some operators depends on the SQL mode:

- By default, `||` is a logical `OR` operator. With `PIPES_AS_CONCAT` enabled, `||` is string concatenation, with a precedence between `^` and the unary operators.
- By default, `!` has a higher precedence than `NOT`. With `HIGH_NOT_PRECEDENCE` enabled, `!` and `NOT` have the same precedence.

See Section 5.1.11, “Server SQL Modes”.

The precedence of operators determines the order of evaluation of terms in an expression. To override this order and group terms explicitly, use parentheses. For example:

```
mysql> SELECT 1+2*3;
        -> 7
mysql> SELECT (1+2)*3;
        -> 9
```

12.3.2 Comparison Functions and Operators

Table 12.3 Comparison Operators

Name	Description
`BETWEEN ... AND ...`	Whether a value is within a range of values
`COALESCE()`	Return the first non-NULL argument
`=`	Equal operator
`<=>`	NULL-safe equal to operator
`>`	Greater than operator
`>=`	Greater than or equal operator
`GREATEST()`	Return the largest argument
`IN()`	Whether a value is within a set of values
`INTERVAL()`	Return the index of the argument that is less than the first argument
`IS`	Test a value against a boolean
`IS NOT`	Test a value against a boolean
`IS NOT NULL`	NOT NULL value test
`IS NULL`	NULL value test
`ISNULL()`	Test whether the argument is NULL
`LEAST()`	Return the smallest argument
`<`	Less than operator
`<=`	Less than or equal operator
`LIKE`	Simple pattern matching
`NOT BETWEEN ... AND ...`	Whether a value is not within a range of values
`!=, <>`	Not equal operator
`NOT IN()`	Whether a value is not within a set of values
`NOT LIKE`	Negation of simple pattern matching

Name	Description
STRCMP()	Compare two strings

Comparison operations result in a value of 1 (TRUE), 0 (FALSE), or NULL. These operations work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as necessary.

The following relational comparison operators can be used to compare not only scalar operands, but row operands:

```
= > < >= <= <> !=
```

The descriptions for those operators later in this section detail how they work with row operands. For additional examples of row comparisons in the context of row subqueries, see Section 13.2.11.5, “Row Subqueries”.

Some of the functions in this section return values other than 1 (TRUE), 0 (FALSE), or NULL. LEAST() and GREATEST() are examples of such functions; Section 12.2, “Type Conversion in Expression Evaluation”, describes the rules for comparison operations performed by these and similar functions for determining their return values.

Note

In previous versions of MySQL, when evaluating an expression containing LEAST() or GREATEST(), the server attempted to guess the context in which the function was used, and to coerce the function's arguments to the data type of the expression as a whole. For example, the arguments to LEAST("11", "45", "2") are evaluated and sorted as strings, so that this expression returns "11". In MySQL 8.0.3 and earlier, when evaluating the expression LEAST("11", "45", "2") + 0, the server converted the arguments to integers (anticipating the addition of integer 0 to the result) before sorting them, thus returning 2.

Beginning with MySQL 8.0.4, the server no longer attempts to infer context in this fashion. Instead, the function is executed using the arguments as provided, performing data type conversions to one or more of the arguments if and only if they are not all of the same type. Any type coercion mandated by an expression that makes use of the return value is now performed following function execution. This means that, in MySQl 8.0.4 and later, LEAST("11", "45", "2") + 0 evaluates to "11" + 0 and thus to integer 11. (Bug #83895, Bug #25123839)

To convert a value to a specific type for comparison purposes, you can use the CAST() function. String values can be converted to a different character set using CONVERT(). See Section 12.10, “Cast Functions and Operators”.

By default, string comparisons are not case-sensitive and use the current character set. The default is utf8mb4.

- =

  Equal:

  ```
  mysql> SELECT 1 = 0;
  ```

```
        -> 0
mysql> SELECT '0' = 0;
        -> 1
mysql> SELECT '0.0' = 0;
        -> 1
mysql> SELECT '0.01' = 0;
        -> 0
mysql> SELECT '.01' = 0.01;
        -> 1
```

For row comparisons, `(a, b) = (x, y)` is equivalent to:

```
(a = x) AND (b = y)
```

- `<=>`

  `NULL`-safe equal. This operator performs an equality comparison like the `=` operator, but returns `1` rather than `NULL` if both operands are `NULL`, and `0` rather than `NULL` if one operand is `NULL`.

  The `<=>` operator is equivalent to the standard SQL `IS NOT DISTINCT FROM` operator.

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
        -> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
        -> 1, NULL, NULL
```

For row comparisons, `(a, b) <=> (x, y)` is equivalent to:

```
(a <=> x) AND (b <=> y)
```

- `<>`, `!=`

  Not equal:

```
mysql> SELECT '.01' <> '0.01';
        -> 1
mysql> SELECT .01 <> '0.01';
        -> 0
mysql> SELECT 'zapp' <> 'zappp';
        -> 1
```

For row comparisons, `(a, b) <> (x, y)` and `(a, b) != (x, y)` are equivalent to:

```
(a <> x) OR (b <> y)
```

- `<=`

  Less than or equal:

```
mysql> SELECT 0.1 <= 2;
        -> 1
```

For row comparisons, `(a, b) <= (x, y)` is equivalent to:

```
(a < x) OR ((a = x) AND (b <= y))
```

- `<`

Less than:

```
mysql> SELECT 2 < 2;
        -> 0
```

For row comparisons, `(a, b) < (x, y)` is equivalent to:

```
(a < x) OR ((a = x) AND (b < y))
```

- `>=`

  Greater than or equal:

  ```
  mysql> SELECT 2 >= 2;
          -> 1
  ```

  For row comparisons, `(a, b) >= (x, y)` is equivalent to:

  ```
  (a > x) OR ((a = x) AND (b >= y))
  ```

- `>`

  Greater than:

  ```
  mysql> SELECT 2 > 2;
          -> 0
  ```

  For row comparisons, `(a, b) > (x, y)` is equivalent to:

  ```
  (a > x) OR ((a = x) AND (b > y))
  ```

- *`expr`* `BETWEEN` *`min`* `AND` *`max`*

  If *`expr`* is greater than or equal to *`min`* and *`expr`* is less than or equal to *`max`*, `BETWEEN` returns `1`, otherwise it returns `0`. This is equivalent to the expression `(`*`min`* `<=` *`expr`* `AND` *`expr`* `<=` *`max`*`)` if all the arguments are of the same type. Otherwise type conversion takes place according to the rules described in Section 12.2, “Type Conversion in Expression Evaluation”, but applied to all the three arguments.

  ```
  mysql> SELECT 2 BETWEEN 1 AND 3, 2 BETWEEN 3 and 1;
          -> 1, 0
  mysql> SELECT 1 BETWEEN 2 AND 3;
          -> 0
  mysql> SELECT 'b' BETWEEN 'a' AND 'c';
          -> 1
  mysql> SELECT 2 BETWEEN 2 AND '3';
          -> 1
  mysql> SELECT 2 BETWEEN 2 AND 'x-3';
          -> 0
  ```

  For best results when using `BETWEEN` with date or time values, use `CAST()` to explicitly convert the values to the desired data type. Examples: If you compare a `DATETIME` to two `DATE` values, convert the `DATE` values to `DATETIME` values. If you use a string constant such as `'2001-1-1'` in a comparison to a `DATE`, cast the string to a `DATE`.

- *expr* NOT BETWEEN *min* AND *max*

  This is the same as NOT (*expr* BETWEEN *min* AND *max*).

- COALESCE(*value*,...)

  Returns the first non-NULL value in the list, or NULL if there are no non-NULL values.

  The return type of COALESCE() is the aggregated type of the argument types.

```
mysql> SELECT COALESCE(NULL,1);
        -> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
        -> NULL
```

- GREATEST(*value1*,*value2*,...)

  With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for LEAST().

```
mysql> SELECT GREATEST(2,0);
        -> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
        -> 767.0
mysql> SELECT GREATEST('B','A','C');
        -> 'C'
```

  GREATEST() returns NULL if any argument is NULL.

- *expr* IN (*value*,...)

  Returns 1 (true) if *expr* is equal to any of the values in the IN() list, else returns 0 (false).

  Type conversion takes place according to the rules described in Section 12.2, “Type Conversion in Expression Evaluation”, applied to all the arguments. If no type conversion is needed for the values in the IN() list, they are all non-JSON constants of the same type, and *expr* can be compared to each of them as a value of the same type (possibly after type conversion), an optimization takes place. The values the list are sorted and the search for *expr* is done using a binary search, which makes the IN() operation very quick.

```
mysql> SELECT 2 IN (0,3,5,7);
        -> 0
mysql> SELECT 'wefwf' IN ('wee','wefwf','weg');
        -> 1
```

  IN() can be used to compare row constructors:

```
mysql> SELECT (3,4) IN ((1,2), (3,4));
        -> 1
mysql> SELECT (3,4) IN ((1,2), (3,5));
        -> 0
```

  You should never mix quoted and unquoted values in an IN() list because the comparison rules for quoted values (such as strings) and unquoted values (such as numbers) differ. Mixing types may therefore lead to inconsistent results. For example, do not write an IN() expression like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN (1,2,'a');
```

Instead, write it like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN ('1','2','a');
```

Implicit type conversion may produce nonintuitive results:

```
mysql> SELECT 'a' IN (0), 0 IN ('b');
        -> 1, 1
```

In both cases, the comparison values are converted to floating-point values, yielding 0.0 in each case, and a comparison result of 1 (true).

The number of values in the `IN()` list is only limited by the `max_allowed_packet` value.

To comply with the SQL standard, `IN()` returns `NULL` not only if the expression on the left hand side is `NULL`, but also if no match is found in the list and one of the expressions in the list is `NULL`.

`IN()` syntax can also be used to write certain types of subqueries. See Section 13.2.11.3, “Subqueries with ANY, IN, or SOME”.

- *`expr`* `NOT IN (`*`value`*`,...)`

  This is the same as `NOT (`*`expr`* `IN (`*`value`*`,...))`.

- `INTERVAL(`*`N`*`,`*`N1`*`,`*`N2`*`,`*`N3`*`,...)`

  Returns `0` if *`N`* < *`N1`*, `1` if *`N`* < *`N2`* and so on or `-1` if *`N`* is `NULL`. All arguments are treated as integers. It is required that *`N1`* < *`N2`* < *`N3`* < `...` < *`Nn`* for this function to work correctly. This is because a binary search is used (very fast).

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
        -> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
        -> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
        -> 0
```

- `IS` *`boolean_value`*

  Tests a value against a boolean value, where *`boolean_value`* can be `TRUE`, `FALSE`, or `UNKNOWN`.

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
        -> 1, 1, 1
```

- `IS NOT` *`boolean_value`*

  Tests a value against a boolean value, where *`boolean_value`* can be `TRUE`, `FALSE`, or `UNKNOWN`.

```
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
        -> 1, 1, 0
```

- `IS NULL`

  Tests whether a value is `NULL`.

```
```

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
        -> 0, 0, 1
```

To work well with ODBC programs, MySQL supports the following extra features when using `IS NULL`:

- If `sql_auto_is_null` variable is set to 1, then after a statement that successfully inserts an automatically generated `AUTO_INCREMENT` value, you can find that value by issuing a statement of the following form:

  ```
  SELECT * FROM tbl_name WHERE auto_col IS NULL
  ```

  If the statement returns a row, the value returned is the same as if you invoked the `LAST_INSERT_ID()` function. For details, including the return value after a multiple-row insert, see Section 12.15, “Information Functions”. If no `AUTO_INCREMENT` value was successfully inserted, the `SELECT` statement returns no row.

  The behavior of retrieving an `AUTO_INCREMENT` value by using an `IS NULL` comparison can be disabled by setting `sql_auto_is_null = 0`. See Section 5.1.8, “Server System Variables”.

  The default value of `sql_auto_is_null` is 0.

- For `DATE` and `DATETIME` columns that are declared as `NOT NULL`, you can find the special date `'0000-00-00'` by using a statement like this:

  ```
  SELECT * FROM tbl_name WHERE date_column IS NULL
  ```

  This is needed to get some ODBC applications to work because ODBC does not support a `'0000-00-00'` date value.

  See Obtaining Auto-Increment Values, and the description for the `FLAG_AUTO_IS_NULL` option at Connector/ODBC Connection Parameters.

- `IS NOT NULL`

  Tests whether a value is not `NULL`.

  ```
  mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
          -> 1, 1, 0
  ```

- `ISNULL(expr)`

  If *`expr`* is `NULL`, `ISNULL()` returns `1`, otherwise it returns `0`.

  ```
  mysql> SELECT ISNULL(1+1);
          -> 0
  mysql> SELECT ISNULL(1/0);
          -> 1
  ```

  `ISNULL()` can be used instead of `=` to test whether a value is `NULL`. (Comparing a value to `NULL` using `=` always yields `NULL`.)

  The `ISNULL()` function shares some special behaviors with the `IS NULL` comparison operator. See the description of `IS NULL`.

- `LEAST(value1,value2,...)`

With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

- If any argument is `NULL`, the result is `NULL`. No comparison is needed.
- If all arguments are integer-valued, they are compared as integers.
- If at least one argument is double precision, they are compared as double-precision values. Otherwise, if at least one argument is a `DECIMAL` value, they are compared as `DECIMAL` values.
- If the arguments comprise a mix of numbers and strings, they are compared as strings.
- If any argument is a nonbinary (character) string, the arguments are compared as nonbinary strings.
- In all other cases, the arguments are compared as binary strings.

The return type of `LEAST()` is the aggregated type of the comparison argument types.

```
mysql> SELECT LEAST(2,0);
        -> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
        -> 3.0
mysql> SELECT LEAST('B','A','C');
        -> 'A'
```

12.3.3 Logical Operators

Table 12.4 Logical Operators

Name	Description
`AND`, `&&`	Logical AND
`NOT`, `!`	Negates value
`OR`, `||`	Logical OR
`XOR`	Logical XOR

In SQL, all logical operators evaluate to `TRUE`, `FALSE`, or `NULL` (`UNKNOWN`). In MySQL, these are implemented as 1 (`TRUE`), 0 (`FALSE`), and `NULL`. Most of this is common to different SQL database servers, although some servers may return any nonzero value for `TRUE`.

MySQL evaluates any nonzero, non-`NULL` value to `TRUE`. For example, the following statements all assess to `TRUE`:

```
mysql> SELECT 10 IS TRUE;
-> 1
mysql> SELECT -10 IS TRUE;
-> 1
mysql> SELECT 'string' IS NOT NULL;
-> 1
```

- `NOT`, `!`

  Logical NOT. Evaluates to `1` if the operand is `0`, to `0` if the operand is nonzero, and `NOT NULL` returns `NULL`.

  ```
  mysql> SELECT NOT 10;
  ```

```
        -> 0
mysql> SELECT NOT 0;
        -> 1
mysql> SELECT NOT NULL;
        -> NULL
mysql> SELECT ! (1+1);
        -> 0
mysql> SELECT ! 1+1;
        -> 1
```

The last example produces `1` because the expression evaluates the same way as `(!1)+1`.

The `!`, operator is a nonstandard MySQL extension. As of MySQL 8.0.17, this operator is deprecated and support for it will be removed in a future MySQL version. Applications should be adjusted to use the standard SQL `NOT` operator.

- `AND`, `&&`

  Logical AND. Evaluates to `1` if all operands are nonzero and not `NULL`, to `0` if one or more operands are `0`, otherwise `NULL` is returned.

```
mysql> SELECT 1 AND 1;
        -> 1
mysql> SELECT 1 AND 0;
        -> 0
mysql> SELECT 1 AND NULL;
        -> NULL
mysql> SELECT 0 AND NULL;
        -> 0
mysql> SELECT NULL AND 0;
        -> 0
```

The `&&`, operator is a nonstandard MySQL extension. As of MySQL 8.0.17, this operator is deprecated and support for it will be removed in a future MySQL version. Applications should be adjusted to use the standard SQL `AND` operator.

- `OR`, `||`

  Logical OR. When both operands are non-`NULL`, the result is `1` if any operand is nonzero, and `0` otherwise. With a `NULL` operand, the result is `1` if the other operand is nonzero, and `NULL` otherwise. If both operands are `NULL`, the result is `NULL`.

```
mysql> SELECT 1 OR 1;
        -> 1
mysql> SELECT 1 OR 0;
        -> 1
mysql> SELECT 0 OR 0;
        -> 0
mysql> SELECT 0 OR NULL;
        -> NULL
mysql> SELECT 1 OR NULL;
        -> 1
```

Note

If the `PIPES_AS_CONCAT` SQL mode is enabled, `||` signifies the SQL-standard string concatenation operator (like `CONCAT()`).

The `||`, operator is a nonstandard MySQL extension. As of MySQL 8.0.17, this operator is deprecated and support for it will be removed in a future MySQL version. Applications should be adjusted to use the

standard SQL `OR` operator. Exception: Deprecation does not apply if `PIPES_AS_CONCAT` is enabled because, in that case, `||` signifies string concatentation.

- `XOR`

  Logical XOR. Returns `NULL` if either operand is `NULL`. For non-`NULL` operands, evaluates to `1` if an odd number of operands is nonzero, otherwise `0` is returned.

```
mysql> SELECT 1 XOR 1;
        -> 0
mysql> SELECT 1 XOR 0;
        -> 1
mysql> SELECT 1 XOR NULL;
        -> NULL
mysql> SELECT 1 XOR 1 XOR 1;
        -> 1
```

  `a XOR b` is mathematically equal to `(a AND (NOT b)) OR ((NOT a) and b)`.

12.3.4 Assignment Operators

Table 12.5 Assignment Operators

Name	Description
`=`	Assign a value (as part of a `SET` statement, or as part of the `SET` clause in an `UPDATE` statement)
`:=`	Assign a value

- `:=`

  Assignment operator. Causes the user variable on the left hand side of the operator to take on the value to its right. The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same `SET` statement. You can perform multiple assignments in the same statement.

  Unlike `=`, the `:=` operator is never interpreted as a comparison operator. This means you can use `:=` in any valid SQL statement (not just in `SET` statements) to assign a value to a variable.

```
mysql> SELECT @var1, @var2;
        -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
        -> 1, 1
mysql> SELECT @var1, @var2;
        -> 1, 1

mysql> SELECT @var1:=COUNT(*) FROM t1;
        -> 4
mysql> SELECT @var1;
        -> 4
```

  You can make value assignments using `:=` in other statements besides `SELECT`, such as `UPDATE`, as shown here:

```
mysql> SELECT @var1;
        -> 4
mysql> SELECT * FROM t1;
        -> 1, 3, 5, 7

mysql> UPDATE t1 SET c1 = 2 WHERE c1 = @var1:= 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT @var1;
        -> 1
mysql> SELECT * FROM t1;
        -> 2, 3, 5, 7
```

While it is also possible both to set and to read the value of the same variable in a single SQL statement using the `:=` operator, this is not recommended. Section 9.4, “User-Defined Variables”, explains why you should avoid doing this.

- `=`

  This operator is used to perform value assignments in two cases, described in the next two paragraphs.

  Within a `SET` statement, `=` is treated as an assignment operator that causes the user variable on the left hand side of the operator to take on the value to its right. (In other words, when used in a `SET` statement, `=` is treated identically to `:=`.) The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same `SET` statement.

  In the `SET` clause of an `UPDATE` statement, `=` also acts as an assignment operator; in this case, however, it causes the column named on the left hand side of the operator to assume the value given to the right, provided any `WHERE` conditions that are part of the `UPDATE` are met. You can make multiple assignments in the same `SET` clause of an `UPDATE` statement.

  In any other context, `=` is treated as a comparison operator.

```
mysql> SELECT @var1, @var2;
        -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
        -> 1, 1
mysql> SELECT @var1, @var2;
        -> 1, 1
```

  For more information, see Section 13.7.6.1, “SET Syntax for Variable Assignment”, Section 13.2.12, “UPDATE Syntax”, and Section 13.2.11, “Subquery Syntax”.

12.4 Control Flow Functions

Table 12.6 Flow Control Operators

Name	Description
`CASE`	Case operator
`IF()`	If/else construct
`IFNULL()`	Null if/else construct

Name	Description
NULLIF()	Return NULL if expr1 = expr2

- `CASE value WHEN [compare_value] THEN result [WHEN [compare_value] THEN result ...] [ELSE result] END`

  `CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END`

  The first `CASE` syntax returns the *`result`* for the first *`value=compare_value`* comparison that is true. The second syntax returns the result for the first condition that is true. If no comparison or condition is true, the result after `ELSE` is returned, or `NULL` if there is no `ELSE` part.

  Note

  The syntax of the `CASE` *expr* described here differs slightly from that of the SQL `CASE` *statement* described in Section 13.6.5.1, “CASE Syntax”, for use inside stored programs. The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

  The return type of a `CASE` expression result is the aggregated type of all result values:

  - If all types are numeric, the aggregated type is also numeric:
    - If at least one argument is double precision, the result is double precision.
    - Otherwise, if at least one argument is `DECIMAL`, the result is `DECIMAL`.
    - Otherwise, the result is an integer type (with one exception):
      - If all integer types are all signed or all unsigned, the result is the same sign and the precision is the highest of all specified integer types (that is, `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, or `BIGINT`).
      - If there is a combination of signed and unsigned integer types, the result is signed and the precision may be higher. For example, if the types are signed `INT` and unsigned `INT`, the result is signed `BIGINT`.
      - The exception is unsigned `BIGINT` combined with any signed integer type. The result is `DECIMAL` with sufficient precision and scale 0.
  - If all types are `BIT`, the result is `BIT`. Otherwise, `BIT` arguments are treated similar to `BIGINT`.
  - If all types are `YEAR`, the result is `YEAR`. Otherwise, `YEAR` arguments are treated similar to `INT`.
  - If all types are character string (`CHAR` or `VARCHAR`), the result is `VARCHAR` with maximum length determined by the longest character length of the operands.
  - If all types are character or binary string, the result is `VARBINARY`.
  - `SET` and `ENUM` are treated similar to `VARCHAR`; the result is `VARCHAR`.
  - If all types are `JSON`, the result is `JSON`.
  - If all types are temporal, the result is temporal:

  - If all temporal types are `DATE`, `TIME`, or `TIMESTAMP`, the result is `DATE`, `TIME`, or `TIMESTAMP`, respectively.
  - Otherwise, for a mix of temporal types, the result is `DATETIME`.
- If all types are `GEOMETRY`, the result is `GEOMETRY`.
- If any type is `BLOB`, the result is `BLOB`.
- For all other type combinations, the result is `VARCHAR`.
- Literal `NULL` operands are ignored for type aggregation.

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
    ->     WHEN 2 THEN 'two' ELSE 'more' END;
        -> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
        -> 'true'
mysql> SELECT CASE BINARY 'B'
    ->     WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
        -> NULL
```

- `IF(`*`expr1`*`,`*`expr2`*`,`*`expr3`*`)`

  If *`expr1`* is `TRUE` (*`expr1`* `<> 0` and *`expr1`* `<> NULL`), `IF()` returns *`expr2`*. Otherwise, it returns *`expr3`*.

  Note

  There is also an `IF` *statement*, which differs from the `IF()` *function* described here. See Section 13.6.5.2, “IF Syntax”.

  If only one of *`expr2`* or *`expr3`* is explicitly `NULL`, the result type of the `IF()` function is the type of the non-`NULL` expression.

  The default return type of `IF()` (which may matter when it is stored into a temporary table) is calculated as follows:

  - If *`expr2`* or *`expr3`* produce a string, the result is a string.

    If *`expr2`* and *`expr3`* are both strings, the result is case-sensitive if either string is case sensitive.
  - If *`expr2`* or *`expr3`* produce a floating-point value, the result is a floating-point value.
  - If *`expr2`* or *`expr3`* produce an integer, the result is an integer.

```
mysql> SELECT IF(1>2,2,3);
        -> 3
mysql> SELECT IF(1<2,'yes','no');
        -> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
        -> 'no'
```

- `IFNULL(`*`expr1`*`,`*`expr2`*`)`

  If *`expr1`* is not `NULL`, `IFNULL()` returns *`expr1`*; otherwise it returns *`expr2`*.

```
```

```
mysql> SELECT IFNULL(1,0);
        -> 1
mysql> SELECT IFNULL(NULL,10);
        -> 10
mysql> SELECT IFNULL(1/0,10);
        -> 10
mysql> SELECT IFNULL(1/0,'yes');
        -> 'yes'
```

The default return type of `IFNULL(expr1,expr2)` is the more “general” of the two expressions, in the order `STRING`, `REAL`, or `INTEGER`. Consider the case of a table based on expressions or where MySQL must internally store a value returned by `IFNULL()` in a temporary table:

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| test  | varbinary(4) | NO   |     |         |       |
+-------+--------------+------+-----+---------+-------+
```

In this example, the type of the `test` column is `VARBINARY(4)` (a string type).

- `NULLIF(expr1,expr2)`

  Returns `NULL` if *`expr1 = expr2`* is true, otherwise returns *`expr1`*. This is the same as `CASE WHEN` *`expr1 = expr2`* `THEN NULL ELSE` *`expr1`* `END`.

  The return value has the same type as the first argument.

```
mysql> SELECT NULLIF(1,1);
        -> NULL
mysql> SELECT NULLIF(1,2);
        -> 1
```

Note

MySQL evaluates *`expr1`* twice if the arguments are not equal.

12.5 String Functions and Operators

Table 12.7 String Functions and Operators

Name	Description
`ASCII()`	Return numeric value of left-most character
`BIN()`	Return a string containing binary representation of a number
`BIT_LENGTH()`	Return length of argument in bits
`CHAR()`	Return the character for each integer passed
`CHAR_LENGTH()`	Return number of characters in argument
`CHARACTER_LENGTH()`	Synonym for CHAR_LENGTH()
`CONCAT()`	Return concatenated string
`CONCAT_WS()`	Return concatenate with separator
`ELT()`	Return string at index number

Name	Description
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD()	Index (position) of first argument in subsequent arguments
FIND_IN_SET()	Index (position) of first argument within second argument
FORMAT()	Return a number formatted to specified number of decimal places
FROM_BASE64()	Decode base64 encoded string and return result
HEX()	Hexadecimal representation of decimal or string value
INSERT()	Insert substring at specified position up to specified number of characters
INSTR()	Return the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LOAD_FILE()	Load the named file
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MATCH	Perform full-text search
MID()	Return a substring starting from the specified position
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
ORD()	Return character code for leftmost character of the argument
POSITION()	Synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Whether string matches regular expression
REGEXP_INSTR()	Starting index of substring matching regular expression
REGEXP_LIKE()	Whether string matches regular expression
REGEXP_REPLACE()	Replace substrings matching regular expression
REGEXP_SUBSTR()	Return substring matching regular expression
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string

Name	Description
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Whether string matches regular expression
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds
SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings
SUBSTR()	Return the substring as specified
SUBSTRING()	Return the substring as specified
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
TO_BASE64()	Return the argument converted to a base-64 string
TRIM()	Remove leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Return a string containing hex representation of a number
UPPER()	Convert to uppercase
WEIGHT_STRING()	Return the weight string for a string

String-valued functions return NULL if the length of the result would be greater than the value of the max_allowed_packet system variable. See Section 5.1.1, “Configuring the Server”.

For functions that operate on string positions, the first position is numbered 1.

For functions that take length arguments, noninteger arguments are rounded to the nearest integer.

- ASCII(*str*)

  Returns the numeric value of the leftmost character of the string *str*. Returns 0 if *str* is the empty string. Returns NULL if *str* is NULL. ASCII() works for 8-bit characters.

```
mysql> SELECT ASCII('2');
        -> 50
mysql> SELECT ASCII(2);
        -> 50
mysql> SELECT ASCII('dx');
        -> 100
```

  See also the ORD() function.

- BIN(*N*)

  Returns a string representation of the binary value of *N*, where *N* is a longlong (BIGINT) number. This is equivalent to CONV(*N*,10,2). Returns NULL if *N* is NULL.

```
mysql> SELECT BIN(12);
```

```
        -> '1100'
```

- `BIT_LENGTH(str)`

  Returns the length of the string *`str`* in bits.

```
mysql> SELECT BIT_LENGTH('text');
        -> 32
```

- `CHAR(N,... [USING charset_name])`

  `CHAR()` interprets each argument *`N`* as an integer and returns a string consisting of the characters given by the code values of those integers. `NULL` values are skipped.

```
mysql> SELECT CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
        -> 'MMM'
```

  `CHAR()` arguments larger than 255 are converted into multiple result bytes. For example, `CHAR(256)` is equivalent to `CHAR(1,0)`, and `CHAR(256*256)` is equivalent to `CHAR(1,0,0)`:

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+----------------+----------------+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+----------------+----------------+
| 0100           | 0100           |
+----------------+----------------+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+------------------+--------------------+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+------------------+--------------------+
| 010000           | 010000             |
+------------------+--------------------+
```

  By default, `CHAR()` returns a binary string. To produce a string in a given character set, use the optional `USING` clause:

```
mysql> SELECT CHARSET(CHAR(X'65')), CHARSET(CHAR(X'65' USING utf8));
+----------------------+------------------------------+
| CHARSET(CHAR(X'65')) | CHARSET(CHAR(X'65' USING utf8)) |
+----------------------+------------------------------+
| binary               | utf8                         |
+----------------------+------------------------------+
```

  If `USING` is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from `CHAR()` becomes `NULL`.

- `CHAR_LENGTH(str)`

  Returns the length of the string *`str`*, measured in characters. A multibyte character counts as a single character. This means that for a string containing five 2-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

- `CHARACTER_LENGTH(str)`

  `CHARACTER_LENGTH()` is a synonym for `CHAR_LENGTH()`.

- `CONCAT(str1,str2,...)`

Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent nonbinary string form.

`CONCAT()` returns `NULL` if any argument is `NULL`.

```
mysql> SELECT CONCAT('My', 'S', 'QL');
        -> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
        -> NULL
mysql> SELECT CONCAT(14.3);
        -> '14.3'
```

For quoted strings, concatenation can be performed by placing the strings next to each other:

```
mysql> SELECT 'My' 'S' 'QL';
        -> 'MySQL'
```

- `CONCAT_WS(separator,str1,str2,...)`

  `CONCAT_WS()` stands for Concatenate With Separator and is a special form of `CONCAT()`. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is `NULL`, the result is `NULL`.

  ```
  mysql> SELECT CONCAT_WS(',','First name','Second name','Last Name');
          -> 'First name,Second name,Last Name'
  mysql> SELECT CONCAT_WS(',','First name',NULL,'Last Name');
          -> 'First name,Last Name'
  ```

  `CONCAT_WS()` does not skip empty strings. However, it does skip any `NULL` values after the separator argument.

- `ELT(N,str1,str2,str3,...)`

  `ELT()` returns the *N*th element of the list of strings: *str1* if *N* = 1, *str2* if *N* = 2, and so on. Returns `NULL` if *N* is less than 1 or greater than the number of arguments. `ELT()` is the complement of `FIELD()`.

  ```
  mysql> SELECT ELT(1, 'Aa', 'Bb', 'Cc', 'Dd');
          -> 'Aa'
  mysql> SELECT ELT(4, 'Aa', 'Bb', 'Cc', 'Dd');
          -> 'Dd'
  ```

- `EXPORT_SET(bits,on,off[,separator[,number_of_bits]])`

  Returns a string such that for every bit set in the value *bits*, you get an *on* string and for every bit not set in the value, you get an *off* string. Bits in *bits* are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the *separator* string (the default being the comma character `,`). The number of bits examined is given by *number_of_bits*, which has a default of 64 if not specified. *number_of_bits* is silently clipped to 64 if larger than 64. It is treated as an unsigned integer, so a value of −1 is effectively the same as 64.

  ```
  mysql> SELECT EXPORT_SET(5,'Y','N',',',4);
          -> 'Y,N,Y,N'
  ```

```
mysql> SELECT EXPORT_SET(6,'1','0',',',10);
       -> '0,1,1,0,0,0,0,0,0,0'
```

- `FIELD(str,str1,str2,str3,...)`

  Returns the index (position) of *`str`* in the *`str1`*, *`str2`*, *`str3`*, `...` list. Returns `0` if *`str`* is not found.

  If all arguments to `FIELD()` are strings, all arguments are compared as strings. If all arguments are numbers, they are compared as numbers. Otherwise, the arguments are compared as double.

  If *`str`* is `NULL`, the return value is `0` because `NULL` fails equality comparison with any value. `FIELD()` is the complement of `ELT()`.

```
mysql> SELECT FIELD('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff');
       -> 2
mysql> SELECT FIELD('Gg', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff');
       -> 0
```

- `FIND_IN_SET(str,strlist)`

  Returns a value in the range of 1 to *`N`* if the string *`str`* is in the string list *`strlist`* consisting of *`N`* substrings. A string list is a string composed of substrings separated by `,` characters. If the first argument is a constant string and the second is a column of type `SET`, the `FIND_IN_SET()` function is optimized to use bit arithmetic. Returns `0` if *`str`* is not in *`strlist`* or if *`strlist`* is the empty string. Returns `NULL` if either argument is `NULL`. This function does not work properly if the first argument contains a comma (`,`) character.

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
       -> 2
```

- `FORMAT(X,D[,locale])`

  Formats the number *`X`* to a format like `'#,###,###.##'`, rounded to *`D`* decimal places, and returns the result as a string. If *`D`* is `0`, the result has no decimal point or fractional part.

  The optional third parameter enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the `lc_time_names` system variable (see Section 10.16, “MySQL Server Locale Support”). If no locale is specified, the default is `'en_US'`.

```
mysql> SELECT FORMAT(12332.123456, 4);
       -> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
       -> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
       -> '12,332'
mysql> SELECT FORMAT(12332.2,2,'de_DE');
       -> '12.332,20'
```

- `FROM_BASE64(str)`

  Takes a string encoded with the base-64 encoded rules used by `TO_BASE64()` and returns the decoded result as a binary string. The result is `NULL` if the argument is `NULL` or not a valid base-64 string. See the description of `TO_BASE64()` for details about the encoding and decoding rules.

```
mysql> SELECT TO_BASE64('abc'), FROM_BASE64(TO_BASE64('abc'));
       -> 'JWJj', 'abc'
```

- `HEX(str)`, `HEX(N)`

  For a string argument `str`, `HEX()` returns a hexadecimal string representation of `str` where each byte of each character in `str` is converted to two hexadecimal digits. (Multibyte characters therefore become more than two digits.) The inverse of this operation is performed by the `UNHEX()` function.

  For a numeric argument `N`, `HEX()` returns a hexadecimal string representation of the value of `N` treated as a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,16)`. The inverse of this operation is performed by `CONV(HEX(N),16,10)`.

```
mysql> SELECT X'616263', HEX('abc'), UNHEX(HEX('abc'));
        -> 'abc', 616263, 'abc'
mysql> SELECT HEX(255), CONV(HEX(255),16,10);
        -> 'FF', 255
```

- `INSERT(str,pos,len,newstr)`

  Returns the string `str`, with the substring beginning at position `pos` and `len` characters long replaced by the string `newstr`. Returns the original string if `pos` is not within the length of the string. Replaces the rest of the string from position `pos` if `len` is not within the length of the rest of the string. Returns `NULL` if any argument is `NULL`.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
        -> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
        -> 'QuWhat'
```

  This function is multibyte safe.

- `INSTR(str,substr)`

  Returns the position of the first occurrence of substring `substr` in string `str`. This is the same as the two-argument form of `LOCATE()`, except that the order of the arguments is reversed.

```
mysql> SELECT INSTR('foobarbar', 'bar');
        -> 4
mysql> SELECT INSTR('xbar', 'foobar');
        -> 0
```

  This function is multibyte safe, and is case-sensitive only if at least one argument is a binary string.

- `LCASE(str)`

  `LCASE()` is a synonym for `LOWER()`.

  `LCASE()` used in a view is rewritten as `LOWER()` when storing the view's definition. (Bug #12844279)

- `LEFT(str,len)`

  Returns the leftmost `len` characters from the string `str`, or `NULL` if any argument is `NULL`.

```
mysql> SELECT LEFT('foobarbar', 5);
        -> 'fooba'
```

  This function is multibyte safe.

- `LENGTH(str)`

  Returns the length of the string *`str`*, measured in bytes. A multibyte character counts as multiple bytes. This means that for a string containing five 2-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

```
mysql> SELECT LENGTH('text');
        -> 4
```

Note

The `Length()` OpenGIS spatial function is named `ST_Length()` in MySQL.

- `LOAD_FILE(file_name)`

  Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full path name to the file, and you must have the `FILE` privilege. The file must be readable by the server and its size less than `max_allowed_packet` bytes. If the `secure_file_priv` system variable is set to a nonempty directory name, the file to be loaded must be located in that directory. (Prior to MySQL 8.0.17, the file must be readable by all, not just readable by the server.)

  If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

  The `character_set_filesystem` system variable controls interpretation of file names that are given as literal strings.

```
mysql> UPDATE t
           SET blob_col=LOAD_FILE('/tmp/picture')
           WHERE id=1;
```

- `LOCATE(substr,str)`, `LOCATE(substr,str,pos)`

  The first syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*. The second syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*, starting at position *`pos`*. Returns `0` if *`substr`* is not in *`str`*. Returns `NULL` if any argument is `NULL`.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
        -> 7
```

  This function is multibyte safe, and is case-sensitive only if at least one argument is a binary string.

- `LOWER(str)`

  Returns the string *`str`* with all characters changed to lowercase according to the current character set mapping. The default is `utf8mb4`.

```
mysql> SELECT LOWER('QUADRATICALLY');
        -> 'quadratically'
```

`LOWER()` (and `UPPER()`) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion, convert the string to a nonbinary string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING utf8mb4));
+------------+-----------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING utf8mb4)) |
+------------+-----------------------------------+
| New York    | new york                          |
+------------+-----------------------------------+
```

For collations of Unicode character sets, `LOWER()` and `UPPER()` work according to the Unicode Collation Algorithm (UCA) version in the collation name, if there is one, and UCA 4.0.0 if no version is specified. For example, `utf8mb4_0900_ai_ci` and `utf8_unicode_520_ci` work according to UCA 9.0.0 and 5.2.0, respectively, whereas `utf8_unicode_ci` works according to UCA 4.0.0. See Section 10.10.1, “Unicode Character Sets”.

This function is multibyte safe.

`LCASE()` used within views is rewritten as `LOWER()`.

- `LPAD(str,len,padstr)`

  Returns the string *`str`*, left-padded with the string *`padstr`* to a length of *`len`* characters. If *`str`* is longer than *`len`*, the return value is shortened to *`len`* characters.

  ```
  mysql> SELECT LPAD('hi',4,'??');
         -> '??hi'
  mysql> SELECT LPAD('hi',1,'??');
         -> 'h'
  ```

- `LTRIM(str)`

  Returns the string *`str`* with leading space characters removed.

  ```
  mysql> SELECT LTRIM('  barbar');
         -> 'barbar'
  ```

  This function is multibyte safe.

- `MAKE_SET(bits,str1,str2,...)`

  Returns a set value (a string containing substrings separated by `,` characters) consisting of the strings that have the corresponding bit in *`bits`* set. *`str1`* corresponds to bit 0, *`str2`* to bit 1, and so on. `NULL` values in *`str1`*, *`str2`*, `...` are not appended to the result.

  ```
  mysql> SELECT MAKE_SET(1,'a','b','c');
         -> 'a'
  mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
         -> 'hello,world'
  mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
         -> 'hello'
  mysql> SELECT MAKE_SET(0,'a','b','c');
         -> ''
  ```

- `MID(str,pos,len)`

`MID(str,pos,len)` is a synonym for `SUBSTRING(str,pos,len)`.

- `OCT(N)`

  Returns a string representation of the octal value of `N`, where `N` is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,8)`. Returns `NULL` if `N` is `NULL`.

```
mysql> SELECT OCT(12);
        -> '14'
```

- `OCTET_LENGTH(str)`

  `OCTET_LENGTH()` is a synonym for `LENGTH()`.

- `ORD(str)`

  If the leftmost character of the string `str` is a multibyte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
  (1st byte code)
+ (2nd byte code * 256)
+ (3rd byte code * 256^2) ...
```

  If the leftmost character is not a multibyte character, `ORD()` returns the same value as the `ASCII()` function.

```
mysql> SELECT ORD('2');
        -> 50
```

- `POSITION(substr IN str)`

  `POSITION(substr IN str)` is a synonym for `LOCATE(substr,str)`.

- `QUOTE(str)`

  Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotation marks and with each instance of backslash (`\`), single quote (`'`), ASCII `NUL`, and Control+Z preceded by a backslash. If the argument is `NULL`, the return value is the word “NULL” without enclosing single quotation marks.

```
mysql> SELECT QUOTE('Don\'t!');
        -> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
        -> NULL
```

  For comparison, see the quoting rules for literal strings and within the C API in Section 9.1.1, “String Literals”, and Section 28.7.7.56, “mysql_real_escape_string_quote()”.

- `REPEAT(str,count)`

  Returns a string consisting of the string `str` repeated `count` times. If `count` is less than 1, returns an empty string. Returns `NULL` if `str` or `count` are `NULL`.

```
mysql> SELECT REPEAT('MySQL', 3);
        -> 'MySQLMySQLMySQL'
```

- `REPLACE(str,from_str,to_str)`

  Returns the string *`str`* with all occurrences of the string *`from_str`* replaced by the string *`to_str`*. `REPLACE()` performs a case-sensitive match when searching for *`from_str`*.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

  This function is multibyte safe.

- `REVERSE(str)`

  Returns the string *`str`* with the order of the characters reversed.

```
mysql> SELECT REVERSE('abc');
        -> 'cba'
```

  This function is multibyte safe.

- `RIGHT(str,len)`

  Returns the rightmost *`len`* characters from the string *`str`*, or `NULL` if any argument is `NULL`.

```
mysql> SELECT RIGHT('foobarbar', 4);
        -> 'rbar'
```

  This function is multibyte safe.

- `RPAD(str,len,padstr)`

  Returns the string *`str`*, right-padded with the string *`padstr`* to a length of *`len`* characters. If *`str`* is longer than *`len`*, the return value is shortened to *`len`* characters.

```
mysql> SELECT RPAD('hi',5,'?');
        -> 'hi???'
mysql> SELECT RPAD('hi',1,'?');
        -> 'h'
```

  This function is multibyte safe.

- `RTRIM(str)`

  Returns the string *`str`* with trailing space characters removed.

```
mysql> SELECT RTRIM('barbar   ');
        -> 'barbar'
```

  This function is multibyte safe.

- `SOUNDEX(str)`

  Returns a soundex string from *`str`*. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the `SOUNDEX()` function returns an arbitrarily long string. You can use `SUBSTRING()` on the result to get a standard soundex string. All nonalphabetic characters in *`str`* are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

Important

When using `SOUNDEX()`, you should be aware of the following limitations:

- This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reliable results.
- This function is not guaranteed to provide consistent results with strings that use multibyte character sets, including `utf-8`. See Bug #22638 for more information.

```
mysql> SELECT SOUNDEX('Hello');
        -> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
        -> 'Q36324'
```

Note

This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

- *`expr1`* `SOUNDS LIKE` *`expr2`*

  This is the same as `SOUNDEX(`*`expr1`*`) = SOUNDEX(`*`expr2`*`)`.

- `SPACE(`*`N`*`)`

  Returns a string consisting of *`N`* space characters.

```
mysql> SELECT SPACE(6);
        -> '      '
```

- `SUBSTR(`*`str`*`,`*`pos`*`)`, `SUBSTR(`*`str`* `FROM` *`pos`*`)`, `SUBSTR(`*`str`*`,`*`pos`*`,`*`len`*`)`, `SUBSTR(`*`str`* `FROM` *`pos`* `FOR` *`len`*`)`

  `SUBSTR()` is a synonym for `SUBSTRING()`.

- `SUBSTRING(`*`str`*`,`*`pos`*`)`, `SUBSTRING(`*`str`* `FROM` *`pos`*`)`, `SUBSTRING(`*`str`*`,`*`pos`*`,`*`len`*`)`, `SUBSTRING(`*`str`* `FROM` *`pos`* `FOR` *`len`*`)`

  The forms without a *`len`* argument return a substring from string *`str`* starting at position *`pos`*. The forms with a *`len`* argument return a substring *`len`* characters long from string *`str`*, starting at position *`pos`*. The forms that use `FROM` are standard SQL syntax. It is also possible to use a negative value for *`pos`*. In this case, the beginning of the substring is *`pos`* characters from the end of the string, rather than the beginning. A negative value may be used for *`pos`* in any of the forms of this function.

  For all forms of `SUBSTRING()`, the position of the first character in the string from which the substring is to be extracted is reckoned as `1`.

```
mysql> SELECT SUBSTRING('Quadratically',5);
        -> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
        -> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
        -> 'ratica'
```

```
mysql> SELECT SUBSTRING('Sakila', -3);
        -> 'ila'
mysql> SELECT SUBSTRING('Sakila', -5, 3);
        -> 'aki'
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
        -> 'ki'
```

This function is multibyte safe.

If *`len`* is less than 1, the result is the empty string.

- `SUBSTRING_INDEX(`*`str`*`,`*`delim`*`,`*`count`*`)`

  Returns the substring from string *`str`* before *`count`* occurrences of the delimiter *`delim`*. If *`count`* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *`count`* is negative, everything to the right of the final delimiter (counting from the right) is returned. `SUBSTRING_INDEX()` performs a case-sensitive match when searching for *`delim`*.

  ```
  mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
          -> 'www.mysql'
  mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
          -> 'mysql.com'
  ```

  This function is multibyte safe.

- `TO_BASE64(`*`str`*`)`

  Converts the string argument to base-64 encoded form and returns the result as a character string with the connection character set and collation. If the argument is not a string, it is converted to a string before conversion takes place. The result is `NULL` if the argument is `NULL`. Base-64 encoded strings can be decoded using the `FROM_BASE64()` function.

  ```
  mysql> SELECT TO_BASE64('abc'), FROM_BASE64(TO_BASE64('abc'));
          -> 'JWJj', 'abc'
  ```

  Different base-64 encoding schemes exist. These are the encoding and decoding rules used by `TO_BASE64()` and `FROM_BASE64()`:

  - The encoding for alphabet value 62 is `'+'`.
  - The encoding for alphabet value 63 is `'/'`.
  - Encoded output consists of groups of 4 printable characters. Each 3 bytes of the input data are encoded using 4 characters. If the last group is incomplete, it is padded with `'='` characters to a length of 4.
  - A newline is added after each 76 characters of encoded output to divide long output into multiple lines.
  - Decoding recognizes and ignores newline, carriage return, tab, and space.

- `TRIM([{BOTH | LEADING | TRAILING} [`*`remstr`*`] FROM] `*`str`*`)`, `TRIM([`*`remstr`*` FROM] `*`str`*`)`

  Returns the string *`str`* with all *`remstr`* prefixes or suffixes removed. If none of the specifiers `BOTH`, `LEADING`, or `TRAILING` is given, `BOTH` is assumed. *`remstr`* is optional and, if not specified, spaces are removed.

  ```
  ```

```
mysql> SELECT TRIM('  bar   ');
        -> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
        -> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
        -> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
        -> 'barx'
```

This function is multibyte safe.

- `UCASE(str)`

  `UCASE()` is a synonym for `UPPER()`.

  `UCASE()` used within views is rewritten as `UPPER()`.

- `UNHEX(str)`

  For a string argument *`str`*, `UNHEX(str)` interprets each pair of characters in the argument as a hexadecimal number and converts it to the byte represented by the number. The return value is a binary string.

```
mysql> SELECT UNHEX('4D7953514C');
        -> 'MySQL'
mysql> SELECT X'4D7953514C';
        -> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
        -> 'string'
mysql> SELECT HEX(UNHEX('1267'));
        -> '1267'
```

  The characters in the argument string must be legal hexadecimal digits: `'0'` .. `'9'`, `'A'` .. `'F'`, `'a'` .. `'f'`. If the argument contains any nonhexadecimal digits, the result is `NULL`:

```
mysql> SELECT UNHEX('GG');
+-------------+
| UNHEX('GG') |
+-------------+
| NULL        |
+-------------+
```

  A `NULL` result can occur if the argument to `UNHEX()` is a `BINARY` column, because values are padded with 0x00 bytes when stored but those bytes are not stripped on retrieval. For example, `'41'` is stored into a `CHAR(3)` column as `'41 '` and retrieved as `'41'` (with the trailing pad space stripped), so `UNHEX()` for the column value returns `'A'`. By contrast `'41'` is stored into a `BINARY(3)` column as `'41\0'` and retrieved as `'41\0'` (with the trailing pad `0x00` byte not stripped). `'\0'` is not a legal hexadecimal digit, so `UNHEX()` for the column value returns `NULL`.

  For a numeric argument *`N`*, the inverse of `HEX(N)` is not performed by `UNHEX()`. Use `CONV(HEX(N),16,10)` instead. See the description of `HEX()`.

- `UPPER(str)`

  Returns the string *`str`* with all characters changed to uppercase according to the current character set mapping. The default is `utf8mb4`.

```
mysql> SELECT UPPER('Hej');
        -> 'HEJ'
```

See the description of `LOWER()` for information that also applies to `UPPER()`. This included information about how to perform lettercase conversion of binary strings (`BINARY`, `VARBINARY`, `BLOB`) for which these functions are ineffective, and information about case folding for Unicode character sets.

This function is multibyte safe.

`UCASE()` used within views is rewritten as `UPPER()`.

- `WEIGHT_STRING(`*`str`* `[AS {CHAR|BINARY}(`*`N`*`)] [`*`flags`*`])`

This function returns the weight string for the input string. The return value is a binary string that represents the comparison and sorting value of the string. It has these properties:

- If `WEIGHT_STRING(`*`str1`*`)` = `WEIGHT_STRING(`*`str2`*`)`, then *`str1`* = *`str2`* (*`str1`* and *`str2`* are considered equal)
- If `WEIGHT_STRING(`*`str1`*`)` < `WEIGHT_STRING(`*`str2`*`)`, then *`str1`* < *`str2`* (*`str1`* sorts before *`str2`*)

`WEIGHT_STRING()` is a debugging function intended for internal use. Its behavior can change without notice between MySQL versions. It can be used for testing and debugging of collations, especially if you are adding a new collation. See Section 10.14, “Adding a Collation to a Character Set”.

This list briefly summarizes the arguments. More details are given in the discussion following the list.

- *`str`*: The input string expression.
- `AS` clause: Optional; cast the input string to a given type and length.
- *`flags`*: Optional; unused.

The input string, *`str`*, is a string expression. If the input is a nonbinary (character) string such as a `CHAR`, `VARCHAR`, or `TEXT` value, the return value contains the collation weights for the string. If the input is a binary (byte) string such as a `BINARY`, `VARBINARY`, or `BLOB` value, the return value is the same as the input (the weight for each byte in a binary string is the byte value). If the input is `NULL`, `WEIGHT_STRING()` returns `NULL`.

Examples:

```
mysql> SET @s = _utf8mb4 'AB' COLLATE utf8mb4_0900_ai_ci;
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+-------------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+-------------------------+
| AB   | 4142    | 1C471C60                |
+------+---------+-------------------------+
```

```
mysql> SET @s = _utf8mb4 'ab' COLLATE utf8mb4_0900_ai_ci;
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+-------------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+-------------------------+
| ab   | 6162    | 1C471C60                |
+------+---------+-------------------------+
```

```
mysql> SET @s = CAST('AB' AS BINARY);
```

```
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+------------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+------------------------+
| AB   | 4142    | 4142                   |
+------+---------+------------------------+
```

```
mysql> SET @s = CAST('ab' AS BINARY);
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+------------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+------------------------+
| ab   | 6162    | 6162                   |
+------+---------+------------------------+
```

The preceding examples use `HEX()` to display the `WEIGHT_STRING()` result. Because the result is a binary value, `HEX()` can be especially useful when the result contains nonprinting values, to display it in printable form:

```
mysql> SET @s = CONVERT(X'C39F' USING utf8) COLLATE utf8_czech_ci;
mysql> SELECT HEX(WEIGHT_STRING(@s));
+------------------------+
| HEX(WEIGHT_STRING(@s)) |
+------------------------+
| 0FEA0FEA               |
+------------------------+
```

For non-`NULL` return values, the data type of the value is `VARBINARY` if its length is within the maximum length for `VARBINARY`, otherwise the data type is `BLOB`.

The `AS` clause may be given to cast the input string to a nonbinary or binary string and to force it to a given length:

- `AS CHAR(N)` casts the string to a nonbinary string and pads it on the right with spaces to a length of *`N`* characters. *`N`* must be at least 1. If *`N`* is less than the length of the input string, the string is truncated to *`N`* characters. No warning occurs for truncation.
- `AS BINARY(N)` is similar but casts the string to a binary string, *`N`* is measured in bytes (not characters), and padding uses `0x00` bytes (not spaces).

```
mysql> SET NAMES 'latin1';
mysql> SELECT HEX(WEIGHT_STRING('ab' AS CHAR(4)));
+-------------------------------------+
| HEX(WEIGHT_STRING('ab' AS CHAR(4))) |
+-------------------------------------+
| 41422020                            |
+-------------------------------------+
mysql> SET NAMES 'utf8';
mysql> SELECT HEX(WEIGHT_STRING('ab' AS CHAR(4)));
+-------------------------------------+
| HEX(WEIGHT_STRING('ab' AS CHAR(4))) |
+-------------------------------------+
| 0041004200200020                    |
+-------------------------------------+
```

```
mysql> SELECT HEX(WEIGHT_STRING('ab' AS BINARY(4)));
+---------------------------------------+
| HEX(WEIGHT_STRING('ab' AS BINARY(4))) |
+---------------------------------------+
| 61620000                              |
```

```
+------------------------------------+
```

The *`flags`* clause currently is unused.

12.5.1 String Comparison Functions and Operators

Table 12.8 String Comparison Functions and Operators

Name	Description
LIKE	Simple pattern matching
NOT LIKE	Negation of simple pattern matching
STRCMP()	Compare two strings

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This affects only comparisons.

Normally, if any expression in a string comparison is case sensitive, the comparison is performed in case-sensitive fashion.

- *`expr`* LIKE *`pat`* [ESCAPE '*`escape_char`*']

  Pattern matching using an SQL pattern. Returns 1 (TRUE) or 0 (FALSE). If either *`expr`* or *`pat`* is NULL, the result is NULL.

  The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

  Per the SQL standard, LIKE performs matching on a per-character basis, thus it can produce results different from the = comparison operator:

```
mysql> SELECT 'ä' LIKE 'ae' COLLATE latin1_german2_ci;
+-----------------------------------------+
| 'ä' LIKE 'ae' COLLATE latin1_german2_ci |
+-----------------------------------------+
|                                       0 |
+-----------------------------------------+
mysql> SELECT 'ä' = 'ae' COLLATE latin1_german2_ci;
+-------------------------------------+
| 'ä' = 'ae' COLLATE latin1_german2_ci |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

  In particular, trailing spaces are significant, which is not true for CHAR or VARCHAR comparisons performed with the = operator:

```
mysql> SELECT 'a' = 'a ', 'a' LIKE 'a ';
+------------+--------------+
| 'a' = 'a ' | 'a' LIKE 'a ' |
+------------+--------------+
|          1 |            0 |
+------------+--------------+
1 row in set (0.00 sec)
```

  With LIKE you can use the following two wildcard characters in the pattern:

  - % matches any number of characters, even zero characters.

- `_` matches exactly one character.

```
mysql> SELECT 'David!' LIKE 'David_';
        -> 1
mysql> SELECT 'David!' LIKE '%D%v%';
        -> 1
```

To test for literal instances of a wildcard character, precede it by the escape character. If you do not specify the `ESCAPE` character, `\` is assumed.

- `\%` matches one `%` character.
- `\_` matches one `_` character.

```
mysql> SELECT 'David!' LIKE 'David\_';
        -> 0
mysql> SELECT 'David_' LIKE 'David\_';
        -> 1
```

To specify a different escape character, use the `ESCAPE` clause:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
        -> 1
```

The escape sequence should be empty or one character long. The expression must evaluate as a constant at execution time. If the `NO_BACKSLASH_ESCAPES` SQL mode is enabled, the sequence cannot be empty.

The following two statements illustrate that string comparisons are not case-sensitive unless one of the operands is case-sensitive (uses a case-sensitive collation or is a binary string):

```
mysql> SELECT 'abc' LIKE 'ABC';
        -> 1
mysql> SELECT 'abc' LIKE _utf8mb4 'ABC' COLLATE utf8mb4_0900_as_cs;
        -> 0
mysql> SELECT 'abc' LIKE _utf8mb4 'ABC' COLLATE utf8mb4_bin;
        -> 0
mysql> SELECT 'abc' LIKE BINARY 'ABC';
        -> 0
```

As an extension to standard SQL, MySQL permits `LIKE` on numeric expressions.

```
mysql> SELECT 10 LIKE '1%';
        -> 1
```

Note

Because MySQL uses C escape syntax in strings (for example, `\n` to represent a newline character), you must double any `\` that you use in `LIKE` strings. For example, to search for `\n`, specify it as `\\n`. To search for `\`, specify it as `\\\\`; this is because the backslashes are stripped once by the parser and again when the pattern match is made, leaving a single backslash to be matched against.

Exception: At the end of the pattern string, backslash can be specified as `\\`. At the end of the string, backslash stands for itself because there is nothing following to escape. Suppose that a table contains the following values:

```
mysql> SELECT filename FROM t1;
+--------------+
| filename     |
+--------------+
| C:           |
| C:\          |
| C:\Programs  |
| C:\Programs\ |
+--------------+
```

To test for values that end with backslash, you can match the values using either of the following patterns:

```
mysql> SELECT filename, filename LIKE '%\\' FROM t1;
+--------------+---------------------+
| filename     | filename LIKE '%\\' |
+--------------+---------------------+
| C:           |                   0 |
| C:\          |                   1 |
| C:\Programs  |                   0 |
| C:\Programs\ |                   1 |
+--------------+---------------------+

mysql> SELECT filename, filename LIKE '%\\\\' FROM t1;
+--------------+----------------------+
| filename     | filename LIKE '%\\\\' |
+--------------+----------------------+
| C:           |                    0 |
| C:\          |                    1 |
| C:\Programs  |                    0 |
| C:\Programs\ |                    1 |
+--------------+----------------------+
```

- *`expr`* `NOT LIKE` *`pat`* `[ESCAPE '`*`escape_char`*`']`

  This is the same as `NOT (`*`expr`* `LIKE` *`pat`* `[ESCAPE '`*`escape_char`*`'])`.

Note

Aggregate queries involving `NOT LIKE` comparisons with columns containing `NULL` may yield unexpected results. For example, consider the following table and data:

```
CREATE TABLE foo (bar VARCHAR(10));

INSERT INTO foo VALUES (NULL), (NULL);
```

The query `SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%';` returns `0`. You might assume that `SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%';` would return `2`. However, this is not the case: The second query returns `0`. This is because `NULL NOT LIKE` *`expr`* always returns `NULL`, regardless of the value of *`expr`*. The same is true for aggregate queries involving `NULL` and comparisons using `NOT RLIKE` or `NOT REGEXP`. In such cases, you must test explicitly for `NOT NULL` using `OR` (and not `AND`), as shown here:

```
SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%' OR bar IS NULL;
```

- `STRCMP(expr1,expr2)`

  `STRCMP()` returns `0` if the strings are the same, `-1` if the first argument is smaller than the second according to the current sort order, and `1` otherwise.

```
mysql> SELECT STRCMP('text', 'text2');
        -> -1
mysql> SELECT STRCMP('text2', 'text');
        -> 1
mysql> SELECT STRCMP('text', 'text');
        -> 0
```

  `STRCMP()` performs the comparison using the collation of the arguments.

```
mysql> SET @s1 = _utf8mb4 'x' COLLATE utf8mb4_0900_ai_ci;
mysql> SET @s2 = _utf8mb4 'X' COLLATE utf8mb4_0900_ai_ci;
mysql> SET @s3 = _utf8mb4 'x' COLLATE utf8mb4_0900_as_cs;
mysql> SET @s4 = _utf8mb4 'X' COLLATE utf8mb4_0900_as_cs;
mysql> SELECT STRCMP(@s1, @s2), STRCMP(@s3, @s4);
+------------------+------------------+
| STRCMP(@s1, @s2) | STRCMP(@s3, @s4) |
+------------------+------------------+
|                0 |               -1 |
+------------------+------------------+
```

  If the collations are incompatible, one of the arguments must be converted to be compatible with the other. See Section 10.8.4, “Collation Coercibility in Expressions”.

```
mysql> SET @s1 = _utf8mb4 'x' COLLATE utf8mb4_0900_ai_ci;
mysql> SET @s2 = _utf8mb4 'X' COLLATE utf8mb4_0900_ai_ci;
mysql> SET @s3 = _utf8mb4 'x' COLLATE utf8mb4_0900_as_cs;
mysql> SET @s4 = _utf8mb4 'X' COLLATE utf8mb4_0900_as_cs;
-->
mysql> SELECT STRCMP(@s1, @s3);
ERROR 1267 (HY000): Illegal mix of collations (utf8mb4_0900_ai_ci,IMPLICIT)
and (utf8mb4_0900_as_cs,IMPLICIT) for operation 'strcmp'
mysql> SELECT STRCMP(@s1, @s3 COLLATE utf8mb4_0900_ai_ci);
+--------------------------------------------+
| STRCMP(@s1, @s3 COLLATE utf8mb4_0900_ai_ci) |
+--------------------------------------------+
|                                          0 |
+--------------------------------------------+
```

12.5.2 Regular Expressions

Table 12.9 Regular Expression Functions and Operators

Name	Description
`NOT REGEXP`	Negation of REGEXP
`REGEXP`	Whether string matches regular expression
`REGEXP_INSTR()`	Starting index of substring matching regular expression
`REGEXP_LIKE()`	Whether string matches regular expression
`REGEXP_REPLACE()`	Replace substrings matching regular expression

Name	Description
REGEXP_SUBSTR()	Return substring matching regular expression
RLIKE	Whether string matches regular expression

A regular expression is a powerful way of specifying a pattern for a complex search. This section discusses the functions and operators available for regular expression matching and illustrates, with examples, some of the special characters and constructs that can be used for regular expression operations. See also Section 3.3.4.7, “Pattern Matching”.

MySQL implements regular expression support using International Components for Unicode (ICU), which provides full Unicode support and is multibyte safe. (Prior to MySQL 8.0.4, MySQL used Henry Spencer's implementation of regular expressions, which operates in byte-wise fashion and is not multibyte safe. For information about ways in which applications that use regular expressions may be affected by the implementation change, see Regular Expression Compatibility Considerations.)

- Regular Expression Functions and Operators
- Regular Expression Syntax
- Regular Expression Resource Control
- Regular Expression Compatibility Considerations

Regular Expression Functions and Operators

- *expr* NOT REGEXP *pat*, *expr* NOT RLIKE *pat*

  This is the same as NOT (*expr* REGEXP *pat*).

- *expr* REGEXP *pat*, *expr* RLIKE *pat*

  Returns 1 if the string *expr* matches the regular expression specified by the pattern *pat*, 0 otherwise. If *expr* or *pat* is NULL, the return value is NULL.

  REGEXP and RLIKE are synonyms for REGEXP_LIKE().

  For additional information about how matching occurs, see the description for REGEXP_LIKE().

```
mysql> SELECT 'Michael!' REGEXP '.*';
+-----------------------+
| 'Michael!' REGEXP '.*' |
+-----------------------+
|                     1 |
+-----------------------+
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
+--------------------------------------+
| 'new*\n*line' REGEXP 'new\\*.\\*line' |
+--------------------------------------+
|                                    0 |
+--------------------------------------+
mysql> SELECT 'a' REGEXP '^[a-d]';
+--------------------+
| 'a' REGEXP '^[a-d]' |
+--------------------+
|                  1 |
+--------------------+
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
+---------------+----------------------+
| 'a' REGEXP 'A' | 'a' REGEXP BINARY 'A' |
```

```
+---------------+----------------------+
|             1 |                    0 |
+---------------+----------------------+
```

- `REGEXP_INSTR(expr, pat[, pos[, occurrence[, return_option[, match_type]]]])`

  Returns the starting index of the substring of the string *`expr`* that matches the regular expression specified by the pattern *`pat`*, 0 if there is no match. If *`expr`* or *`pat`* is `NULL`, the return value is `NULL`. Character indexes begin at 1.

  `REGEXP_INSTR()` takes these optional arguments:

  - *`pos`*: The position in *`expr`* at which to start the search. If omitted, the default is 1.
  - *`occurrence`*: Which occurrence of a match to search for. If omitted, the default is 1.
  - *`return_option`*: Which type of position to return. If this value is 0, `REGEXP_INSTR()` returns the position of the matched substring's first character. If this value is 1, `REGEXP_INSTR()` returns the position following the matched substring. If omitted, the default is 0.
  - *`match_type`*: A string that specifies how to perform matching. The meaning is as described for `REGEXP_LIKE()`.

  For additional information about how matching occurs, see the description for `REGEXP_LIKE()`.

```
mysql> SELECT REGEXP_INSTR('dog cat dog', 'dog');
+-----------------------------------+
| REGEXP_INSTR('dog cat dog', 'dog') |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
mysql> SELECT REGEXP_INSTR('dog cat dog', 'dog', 2);
+--------------------------------------+
| REGEXP_INSTR('dog cat dog', 'dog', 2) |
+--------------------------------------+
|                                    9 |
+--------------------------------------+
mysql> SELECT REGEXP_INSTR('aa aaa aaaa', 'a{2}');
+-----------------------------------+
| REGEXP_INSTR('aa aaa aaaa', 'a{2}') |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
mysql> SELECT REGEXP_INSTR('aa aaa aaaa', 'a{4}');
+-----------------------------------+
| REGEXP_INSTR('aa aaa aaaa', 'a{4}') |
+-----------------------------------+
|                                 8 |
+-----------------------------------+
```

- `REGEXP_LIKE(expr, pat[, match_type])`

  Returns 1 if the string *`expr`* matches the regular expression specified by the pattern *`pat`*, 0 otherwise. If *`expr`* or *`pat`* is `NULL`, the return value is `NULL`.

  The pattern can be an extended regular expression, the syntax for which is discussed in Regular Expression Syntax. The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

  The optional *`match_type`* argument is a string that may contain any or all the following characters specifying how to perform matching:

- `c`: Case sensitive matching.
- `i`: Case insensitive matching.
- `m`: Multiple-line mode. Recognize line terminators within the string. The default behavior is to match line terminators only at the start and end of the string expression.
- `n`: The `.` character matches line terminators. The default is for `.` matching to stop at the end of a line.
- `u`: Unix-only line endings. Only the newline character is recognized as a line ending by the `.`, `^`, and `$` match operators.

If characters specifying contradictory options are specified within *`match_type`*, the rightmost one takes precedence.

By default, regular expression operations use the character set and collation of the *`expr`* and *`pat`* arguments when deciding the type of a character and performing the comparison. If the arguments have different character sets or collations, coercibility rules apply as described in Section 10.8.4, "Collation Coercibility in Expressions". Arguments may be specified with explicit collation indicators to change comparison behavior.

```
mysql> SELECT REGEXP_LIKE('CamelCase', 'CAMELCASE');
+-------------------------------------+
| REGEXP_LIKE('CamelCase', 'CAMELCASE') |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
mysql> SELECT REGEXP_LIKE('CamelCase', 'CAMELCASE' COLLATE utf8mb4_0900_as_cs);
+-------------------------------------------------------------------+
| REGEXP_LIKE('CamelCase', 'CAMELCASE' COLLATE utf8mb4_0900_as_cs) |
+-------------------------------------------------------------------+
|                                                                 0 |
+-------------------------------------------------------------------+
```

`match_type` may be specified with the `c` or `i` characters to override the default case sensitivity. Exception: If either argument is a binary string, the arguments are handled in case-sensitive fashion as binary strings, even if *`match_type`* contains the `i` character.

Note

Because MySQL uses the C escape syntax in strings (for example, `\n` to represent the newline character), you must double any `\` that you use in your *`expr`* and *`pat`* arguments.

```
mysql> SELECT REGEXP_LIKE('Michael!', '.*');
+-------------------------------+
| REGEXP_LIKE('Michael!', '.*') |
+-------------------------------+
|                             1 |
+-------------------------------+
mysql> SELECT REGEXP_LIKE('new*\n*line', 'new\\*.\\*line');
+--------------------------------------------+
| REGEXP_LIKE('new*\n*line', 'new\\*.\\*line') |
+--------------------------------------------+
|                                          0 |
+--------------------------------------------+
mysql> SELECT REGEXP_LIKE('a', '^[a-d]');
+---------------------------+
```

```
| REGEXP_LIKE('a', '^[a-d]') |
+----------------------------+
|                          1 |
+----------------------------+
mysql> SELECT REGEXP_LIKE('a', 'A'), REGEXP_LIKE('a', BINARY 'A');
+----------------------+-----------------------------+
| REGEXP_LIKE('a', 'A') | REGEXP_LIKE('a', BINARY 'A') |
+----------------------+-----------------------------+
|                    1 |                           0 |
+----------------------+-----------------------------+
```

```
mysql> SELECT REGEXP_LIKE('abc', 'ABC');
+---------------------------+
| REGEXP_LIKE('abc', 'ABC') |
+---------------------------+
|                         1 |
+---------------------------+
mysql> SELECT REGEXP_LIKE('abc', 'ABC', 'c');
+--------------------------------+
| REGEXP_LIKE('abc', 'ABC', 'c') |
+--------------------------------+
|                              0 |
+--------------------------------+
```

- REGEXP_REPLACE(*expr*, *pat*, *repl*[, *pos*[, *occurrence*[, *match_type*]]])

  Replaces occurrences in the string *expr* that match the regular expression specified by the pattern *pat* with the replacement string *repl*, and returns the resulting string. If *expr*, *pat*, or *repl* is NULL, the return value is NULL.

  REGEXP_REPLACE() takes these optional arguments:

  - *pos*: The position in *expr* at which to start the search. If omitted, the default is 1.
  - *occurrence*: Which occurrence of a match to replace. If omitted, the default is 0 (which means “replace all occurrences”).
  - *match_type*: A string that specifies how to perform matching. The meaning is as described for REGEXP_LIKE().

  Prior to MySQL 8.0.17, the result returned by this function used the UTF-16 character set; in MySQL 8.0.17 and later, the character set and collation of the expression searched for matches is used. (Bug #94203, Bug #29308212)

  For additional information about how matching occurs, see the description for REGEXP_LIKE().

```
mysql> SELECT REGEXP_REPLACE('a b c', 'b', 'X');
+-----------------------------------+
| REGEXP_REPLACE('a b c', 'b', 'X') |
+-----------------------------------+
| a X c                             |
+-----------------------------------+
mysql> SELECT REGEXP_REPLACE('abc def ghi', '[a-z]+', 'X', 1, 3);
+-------------------------------------------------------+
| REGEXP_REPLACE('abc def ghi', '[a-z]+', 'X', 1, 3) |
+-------------------------------------------------------+
| abc def X                                             |
+-------------------------------------------------------+
```

- REGEXP_SUBSTR(*expr*, *pat*[, *pos*[, *occurrence*[, *match_type*]]])

Returns the substring of the string *`expr`* that matches the regular expression specified by the pattern *`pat`*, `NULL` if there is no match. If *`expr`* or *`pat`* is `NULL`, the return value is `NULL`.

`REGEXP_SUBSTR()` takes these optional arguments:

- *`pos`*: The position in *`expr`* at which to start the search. If omitted, the default is 1.
- *`occurrence`*: Which occurrence of a match to search for. If omitted, the default is 1.
- *`match_type`*: A string that specifies how to perform matching. The meaning is as described for `REGEXP_LIKE()`.

Prior to MySQL 8.0.17, the result returned by this function used the `UTF-16` character set; in MySQL 8.0.17 and later, the character set and collation of the expression searched for matches is used. (Bug #94203, Bug #29308212)

For additional information about how matching occurs, see the description for `REGEXP_LIKE()`.

```
mysql> SELECT REGEXP_SUBSTR('abc def ghi', '[a-z]+');
+----------------------------------------+
| REGEXP_SUBSTR('abc def ghi', '[a-z]+') |
+----------------------------------------+
| abc                                    |
+----------------------------------------+
mysql> SELECT REGEXP_SUBSTR('abc def ghi', '[a-z]+', 1, 3);
+-------------------------------------------------+
| REGEXP_SUBSTR('abc def ghi', '[a-z]+', 1, 3) |
+-------------------------------------------------+
| ghi                                             |
+-------------------------------------------------+
```

Regular Expression Syntax

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression `hello` matches `hello` and nothing else.

Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression `hello|world` contains the `|` alternation operator and matches either the `hello` or `world`.

As a more complex example, the regular expression `B[an]*s` matches any of the strings `Bananas`, `Baaaaas`, `Bs`, and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

The following list covers some of the basic special characters and constructs that can be used in regular expressions. For information about the full regular expression syntax supported by the ICU library used to implement regular expression support, visit the International Components for Unicode website.

- `^`

  Match the beginning of a string.

```
mysql> SELECT REGEXP_LIKE('fo\nfo', '^fo$');                    -> 0
mysql> SELECT REGEXP_LIKE('fofo', '^fo');                       -> 1
```

- `$`

  Match the end of a string.

```
mysql> SELECT REGEXP_LIKE('fo\no', '^fo\no$');                -> 1
mysql> SELECT REGEXP_LIKE('fo\no', '^fo$');                   -> 0
```

- `.`

  Match any character (including carriage return and newline, although to match these in the middle of a string, the `m` (multiple line) match-control character or the `(?m)` within-pattern modifier must be given).

```
mysql> SELECT REGEXP_LIKE('fofo', '^f.*$');                   -> 1
mysql> SELECT REGEXP_LIKE('fo\r\nfo', '^f.*$');               -> 0
mysql> SELECT REGEXP_LIKE('fo\r\nfo', '^f.*$', 'm');          -> 1
mysql> SELECT REGEXP_LIKE('fo\r\nfo', '(?m)^f.*$');          -> 1
```

- `a*`

  Match any sequence of zero or more `a` characters.

```
mysql> SELECT REGEXP_LIKE('Ban', '^Ba*n');                    -> 1
mysql> SELECT REGEXP_LIKE('Baaan', '^Ba*n');                  -> 1
mysql> SELECT REGEXP_LIKE('Bn', '^Ba*n');                     -> 1
```

- `a+`

  Match any sequence of one or more `a` characters.

```
mysql> SELECT REGEXP_LIKE('Ban', '^Ba+n');                    -> 1
mysql> SELECT REGEXP_LIKE('Bn', '^Ba+n');                     -> 0
```

- `a?`

  Match either zero or one `a` character.

```
mysql> SELECT REGEXP_LIKE('Bn', '^Ba?n');                     -> 1
mysql> SELECT REGEXP_LIKE('Ban', '^Ba?n');                    -> 1
mysql> SELECT REGEXP_LIKE('Baan', '^Ba?n');                   -> 0
```

- `de|abc`

  Alternation; match either of the sequences `de` or `abc`.

```
mysql> SELECT REGEXP_LIKE('pi', 'pi|apa');                    -> 1
mysql> SELECT REGEXP_LIKE('axe', 'pi|apa');                   -> 0
mysql> SELECT REGEXP_LIKE('apa', 'pi|apa');                   -> 1
mysql> SELECT REGEXP_LIKE('apa', '^(pi|apa)$');               -> 1
mysql> SELECT REGEXP_LIKE('pi', '^(pi|apa)$');                -> 1
mysql> SELECT REGEXP_LIKE('pix', '^(pi|apa)$');               -> 0
```

- `(abc)*`

  Match zero or more instances of the sequence `abc`.

```
mysql> SELECT REGEXP_LIKE('pi', '^(pi)*$');                   -> 1
mysql> SELECT REGEXP_LIKE('pip', '^(pi)*$');                  -> 0
mysql> SELECT REGEXP_LIKE('pipi', '^(pi)*$');                 -> 1
```

- `{1}`, `{2,3}`

Repetition; `{n}` and `{m,n}` notation provide a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. *m* and *n* are integers.

- `a*`

  Can be written as `a{0,}`.

- `a+`

  Can be written as `a{1,}`.

- `a?`

  Can be written as `a{0,1}`.

To be more precise, `a{n}` matches exactly *n* instances of `a`. `a{n,}` matches *n* or more instances of `a`. `a{m,n}` matches *m* through *n* instances of `a`, inclusive. If both *m* and *n* are given, *m* must be less than or equal to *n*.

```
mysql> SELECT REGEXP_LIKE('abcde', 'a[bcd]{2}e');             -> 0
mysql> SELECT REGEXP_LIKE('abcde', 'a[bcd]{3}e');             -> 1
mysql> SELECT REGEXP_LIKE('abcde', 'a[bcd]{1,10}e');          -> 1
```

- `[a-dX]`, `[^a-dX]`

  Matches any character that is (or is not, if `^` is used) either `a`, `b`, `c`, `d` or `X`. A `-` character between two other characters forms a range that matches all characters from the first character to the second. For example, `[0-9]` matches any decimal digit. To include a literal `]` character, it must immediately follow the opening bracket `[`. To include a literal `-` character, it must be written first or last. Any character that does not have a defined special meaning inside a `[]` pair matches only itself.

```
mysql> SELECT REGEXP_LIKE('aXbc', '[a-dXYZ]');                -> 1
mysql> SELECT REGEXP_LIKE('aXbc', '^[a-dXYZ]$');              -> 0
mysql> SELECT REGEXP_LIKE('aXbc', '^[a-dXYZ]+$');             -> 1
mysql> SELECT REGEXP_LIKE('aXbc', '^[^a-dXYZ]+$');            -> 0
mysql> SELECT REGEXP_LIKE('gheis', '^[^a-dXYZ]+$');           -> 1
mysql> SELECT REGEXP_LIKE('gheisa', '^[^a-dXYZ]+$');          -> 0
```

- `[=character_class=]`

  Within a bracket expression (written using `[` and `]`), `[=character_class=]` represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if `o` and `(+)` are the members of an equivalence class, `[[=o=]]`, `[[=(+)=]]`, and `[o(+)]` are all synonymous. An equivalence class may not be used as an endpoint of a range.

- `[:character_class:]`

  Within a bracket expression (written using `[` and `]`), `[:character_class:]` represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the `ctype(3)` manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

Character Class Name	Meaning
`alnum`	Alphanumeric characters
`alpha`	Alphabetic characters
`blank`	Whitespace characters
`cntrl`	Control characters
`digit`	Digit characters
`graph`	Graphic characters
`lower`	Lowercase alphabetic characters
`print`	Graphic or space characters
`punct`	Punctuation characters
`space`	Space, tab, newline, and carriage return
`upper`	Uppercase alphabetic characters
`xdigit`	Hexadecimal digit characters

```
mysql> SELECT REGEXP_LIKE('justalnums', '[[:alnum:]]+');       -> 1
mysql> SELECT REGEXP_LIKE('!!', '[[:alnum:]]+');               -> 0
```

To use a literal instance of a special character in a regular expression, precede it by two backslash (\) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string `1+2` that contains the special `+` character, only the last of the following regular expressions is the correct one:

```
mysql> SELECT REGEXP_LIKE('1+2', '1+2');                       -> 0
mysql> SELECT REGEXP_LIKE('1+2', '1\+2');                      -> 0
mysql> SELECT REGEXP_LIKE('1+2', '1\\+2');                     -> 1
```

Regular Expression Resource Control

`REGEXP_LIKE()` and similar functions use resources that can be controlled by setting system variables:

- The match engine uses memory for its internal stack. To control the maximum available memory for the stack in bytes, set the `regexp_stack_limit` system variable.
- The match engine operates in steps. To control the maximum number of steps performed by the engine (and thus indirectly the execution time), set the `regexp_time_limit` system variable. Because this limit is expressed as number of steps, it affects execution time only indirectly. Typically, it is on the order of milliseconds.

Regular Expression Compatibility Considerations

Prior to MySQL 8.0.4, MySQL used the Henry Spencer regular expression library to support regular expression operations, rather than International Components for Unicode (ICU). The following discussion describes differences between the Spencer and ICU libraries that may affect applications:

- With the Spencer library, the `REGEXP` and `RLIKE` operators work in byte-wise fashion, so they are not multibyte safe and may produce unexpected results with multibyte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

ICU has full Unicode support and is multibyte safe. Its regular expression functions treat all strings as UTF-16. You should keep in mind that positional indexes are based on 16-bit chunks and not on code points. This means that, when passed to such functions, characters using more than one chunk may produce unanticipated results, such as those shown here:

```
mysql> SELECT REGEXP_INSTR('####b', 'b');
+--------------------------+
| REGEXP_INSTR('??b', 'b') |
+--------------------------+
|                        5 |
+--------------------------+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_INSTR('####bxxx', 'b', 4);
+-------------------------------+
| REGEXP_INSTR('??bxxx', 'b', 4) |
+-------------------------------+
|                             5 |
+-------------------------------+
1 row in set (0.00 sec)
```

Characters within the Unicode Basic Multilingual Plane, which includes characters used by most modern languages, are safe in this regard:

```
mysql> SELECT REGEXP_INSTR('бжb', 'b');
+---------------------------+
| REGEXP_INSTR('бжb', 'b')  |
+---------------------------+
|                         3 |
+---------------------------+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_INSTR('עבb', 'b');
+---------------------------+
| REGEXP_INSTR('עבb', 'b')  |
+---------------------------+
|                         3 |
+---------------------------+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_INSTR('µå周çб', '周');
+-----------------------------------+
| REGEXP_INSTR('µå周çб', '周')      |
+-----------------------------------+
|                                 3 |
+-----------------------------------+
1 row in set (0.00 sec)
```

Emoji, such as the “sushi” character ## (U+1F363) used in the first two examples, are not included in the Basic Multilingual Plane, but rather in Unicode's Supplementary Multilingual Plane. Another issue can arise with emoji and other 4-byte characters when REGEXP_SUBSTR() or a similar function begins searching in the middle of a character. Each of the two statements in the following example starts from the second 2-byte position in the first argument. The first statement works on a string consisting solely of 2-byte (BMP) characters. The second statement contains 4-byte characters which are incorrectly interpreted in the result because the first two bytes are stripped off and so the remainder of the character data is misaligned.

```
mysql> SELECT REGEXP_SUBSTR('周周周周', '.*', 2);
```

```
+--------------------------------------+
| REGEXP_SUBSTR('周周周周', '.*', 2)     |
+--------------------------------------+
| 周周周                                 |
+--------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT REGEXP_SUBSTR('########', '.*', 2);
+-------------------------------+
| REGEXP_SUBSTR('????', '.*', 2) |
+-------------------------------+
| ?#揑#揑#揑                      |
+-------------------------------+
1 row in set (0.00 sec)
```

- For the `.` operator, the Spencer library matches line-terminator characters (carriage return, newline) anywhere in string expressions, including in the middle. To match line terminator characters in the middle of strings with ICU, specify the `m` match-control character.
- The Spencer library supports word-beginning and word-end boundary markers (`[[:<:]]` and `[[:>:]]` notation). ICU does not. For ICU, you can use `\b` to match word boundaries; double the backslash because MySQL interprets it as the escape character within strings.
- The Spencer library supports collating element bracket expressions (`[.characters.]` notation). ICU does not.
- For repetition counts (`{n}` and `{m,n}` notation), the Spencer library has a maximum of 255. ICU has no such limit, although the maximum number of match engine steps can be limited by setting the `regexp_time_limit` system variable.
- ICU interprets parentheses as metacharacters. To specify a literal open or close parenthesis `(` in a regular expression, it must be escaped:

```
mysql> SELECT REGEXP_LIKE('(', '(');
ERROR 3692 (HY000): Mismatched parenthesis in regular expression.
mysql> SELECT REGEXP_LIKE('(', '\\(');
+-------------------------+
| REGEXP_LIKE('(', '\\(') |
+-------------------------+
|                       1 |
+-------------------------+
mysql> SELECT REGEXP_LIKE(')', ')');
ERROR 3692 (HY000): Mismatched parenthesis in regular expression.
mysql> SELECT REGEXP_LIKE(')', '\\)');
+-------------------------+
| REGEXP_LIKE(')', '\\)') |
+-------------------------+
|                       1 |
+-------------------------+
```

- ICU also interprets square brackets as metacharacters, but only the opening square bracket need be escaped to be used as a literal character:

```
mysql> SELECT REGEXP_LIKE('[', '[');
ERROR 3696 (HY000): The regular expression contains an
unclosed bracket expression.
mysql> SELECT REGEXP_LIKE('[', '\\[');
+-------------------------+
| REGEXP_LIKE('[', '\\[') |
+-------------------------+
|                       1 |
+-------------------------+
```

```
mysql> SELECT REGEXP_LIKE(']', ']');
+----------------------+
| REGEXP_LIKE(']', ']') |
+----------------------+
|                    1 |
+----------------------+
```

12.5.3 Character Set and Collation of Function Results

MySQL has many operators and functions that return a string. This section answers the question: What is the character set and collation of such a string?

For simple functions that take string input and return a string result as output, the output's character set and collation are the same as those of the principal input value. For example, `UPPER(X)` returns a string with the same character string and collation as *X*. The same applies for `INSTR()`, `LCASE()`, `LOWER()`, `LTRIM()`, `MID()`, `REPEAT()`, `REPLACE()`, `REVERSE()`, `RIGHT()`, `RPAD()`, `RTRIM()`, `SOUNDEX()`, `SUBSTRING()`, `TRIM()`, `UCASE()`, and `UPPER()`.

Note

The `REPLACE()` function, unlike all other functions, always ignores the collation of the string input and performs a case-sensitive comparison.

If a string input or function result is a binary string, the string has the `binary` character set and collation. This can be checked by using the `CHARSET()` and `COLLATION()` functions, both of which return `binary` for a binary string argument:

```
mysql> SELECT CHARSET(BINARY 'a'), COLLATION(BINARY 'a');
+--------------------+----------------------+
| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |
+--------------------+----------------------+
| binary              | binary               |
+--------------------+----------------------+
```

For operations that combine multiple string inputs and return a single string output, the “aggregation rules” of standard SQL apply for determining the collation of the result:

- If an explicit `COLLATE Y` occurs, use *Y*.
- If explicit `COLLATE Y` and `COLLATE Z` occur, raise an error.
- Otherwise, if all collations are *Y*, use *Y*.
- Otherwise, the result has no collation.

For example, with `CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END`, the resulting collation is *X*. The same applies for `UNION`, `||`, `CONCAT()`, `ELT()`, `GREATEST()`, `IF()`, and `LEAST()`.

For operations that convert to character data, the character set and collation of the strings that result from the operations are defined by the `character_set_connection` and `collation_connection` system variables that determine the default connection character set and collation (see Section 10.4, “Connection Character Sets and Collations”). This applies only to `BIN_TO_UUID()`, `CAST()`, `CONV()`, `FORMAT()`, `HEX()`, and `SPACE()`.

An exception to the preceding priniciple occurs for expressions for virtual generated columns. In such expressions, the table character set is used for `BIN_TO_UUID()`, `CONV()`, or `HEX()` results, regardless of connection character set.

If there is any question about the character set or collation of the result returned by a string function, use the `CHARSET()` or `COLLATION()` function to find out:

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+---------------+-----------------+-------------------+
| USER()        | CHARSET(USER()) | COLLATION(USER()) |
+---------------+-----------------+-------------------+
| test@localhost | utf8            | utf8_general_ci   |
+---------------+-----------------+-------------------+
mysql> SELECT CHARSET(COMPRESS('abc')), COLLATION(COMPRESS('abc'));
+--------------------------+----------------------------+
| CHARSET(COMPRESS('abc')) | COLLATION(COMPRESS('abc')) |
+--------------------------+----------------------------+
| binary                   | binary                     |
+--------------------------+----------------------------+
```

12.6 Numeric Functions and Operators

Table 12.10 Numeric Functions and Operators

Name	Description
ABS()	Return the absolute value
ACOS()	Return the arc cosine
ASIN()	Return the arc sine
ATAN()	Return the arc tangent
ATAN2(), ATAN()	Return the arc tangent of the two arguments
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
CONV()	Convert numbers between different number bases
COS()	Return the cosine
COT()	Return the cotangent
CRC32()	Compute a cyclic redundancy check value
DEGREES()	Convert radians to degrees
DIV	Integer division
/	Division operator
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
LN()	Return the natural logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
-	Minus operator
MOD()	Return the remainder
%, MOD	Modulo operator
PI()	Return the value of pi
+	Addition operator
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power

Name	Description
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
*	Multiplication operator
TRUNCATE()	Truncate to specified number of decimal places
-	Change the sign of the argument

12.6.1 Arithmetic Operators

Table 12.11 Arithmetic Operators

Name	Description
DIV	Integer division
/	Division operator
-	Minus operator
%, MOD	Modulo operator
+	Addition operator
*	Multiplication operator
-	Change the sign of the argument

The usual arithmetic operators are available. The result is determined according to the following rules:

- In the case of `-`, `+`, and `*`, the result is calculated with `BIGINT` (64-bit) precision if both operands are integers.
- If both operands are integers and any of them are unsigned, the result is an unsigned integer. For subtraction, if the `NO_UNSIGNED_SUBTRACTION` SQL mode is enabled, the result is signed even if any operand is unsigned.
- If any of the operands of a `+`, `-`, `/`, `*`, `%` is a real or string value, the precision of the result is the precision of the operand with the maximum precision.
- In division performed with `/`, the scale of the result when using two exact-value operands is the scale of the first operand plus the value of the `div_precision_increment` system variable (which is 4 by default). For example, the result of the expression `5.05 / 0.014` has a scale of six decimal places (`360.714286`).

These rules are applied for each operation, such that nested calculations imply the precision of each component. Hence, `(14620 / 9432456) / (24250 / 9432456)`, resolves first to `(0.0014) / (0.0026)`, with the final result having 8 decimal places (`0.60288653`).

Because of these rules and the way they are applied, care should be taken to ensure that components and subcomponents of a calculation use the appropriate level of precision. See Section 12.10, “Cast Functions and Operators”.

For information about handling of overflow in numeric expression evaluation, see Section 11.2.6, “Out-of-Range and Overflow Handling”.

Arithmetic operators apply to numbers. For other types of values, alternative operations may be available. For example, to add date values, use `DATE_ADD()`; see Section 12.7, “Date and Time Functions”.

- `+`

  Addition:

```
mysql> SELECT 3+5;
        -> 8
```

- `-`

  Subtraction:

```
mysql> SELECT 3-5;
        -> -2
```

- `-`

  Unary minus. This operator changes the sign of the operand.

```
mysql> SELECT - 2;
        -> -2
```

Note

If this operator is used with a `BIGINT`, the return value is also a `BIGINT`. This means that you should avoid using `-` on integers that may have the value of -2^{63}.

- `*`

  Multiplication:

```
mysql> SELECT 3*5;
        -> 15
mysql> SELECT 18014398509481984*18014398509481984.0;
        -> 324518553658426726783156020576256.0
mysql> SELECT 18014398509481984*18014398509481984;
        -> out-of-range error
```

  The last expression produces an error because the result of the integer multiplication exceeds the 64-bit range of `BIGINT` calculations. (See Section 11.2, “Numeric Types”.)

- `/`

  Division:

```
mysql> SELECT 3/5;
        -> 0.60
```

  Division by zero produces a `NULL` result:

```
mysql> SELECT 102/(1-1);
```

```
        -> NULL
```

A division is calculated with `BIGINT` arithmetic only if performed in a context where its result is converted to an integer.

- `DIV`

  Integer division. Discards from the division result any fractional part to the right of the decimal point.

  If either operand has a noninteger type, the operands are converted to `DECIMAL` and divided using `DECIMAL` arithmetic before converting the result to `BIGINT`. If the result exceeds `BIGINT` range, an error occurs.

```
mysql> SELECT 5 DIV 2, -5 DIV 2, 5 DIV -2, -5 DIV -2;
        -> 2, -2, -2, 2
```

- *`N`* `%` *`M`*, *`N`* `MOD` *`M`*

  Modulo operation. Returns the remainder of *`N`* divided by *`M`*. For more information, see the description for the `MOD()` function in Section 12.6.2, “Mathematical Functions”.

12.6.2 Mathematical Functions

Table 12.12 Mathematical Functions

Name	Description
`ABS()`	Return the absolute value
`ACOS()`	Return the arc cosine
`ASIN()`	Return the arc sine
`ATAN()`	Return the arc tangent
`ATAN2()`, `ATAN()`	Return the arc tangent of the two arguments
`CEIL()`	Return the smallest integer value not less than the argument
`CEILING()`	Return the smallest integer value not less than the argument
`CONV()`	Convert numbers between different number bases
`COS()`	Return the cosine
`COT()`	Return the cotangent
`CRC32()`	Compute a cyclic redundancy check value
`DEGREES()`	Convert radians to degrees
`EXP()`	Raise to the power of
`FLOOR()`	Return the largest integer value not greater than the argument
`LN()`	Return the natural logarithm of the argument
`LOG()`	Return the natural logarithm of the first argument
`LOG10()`	Return the base-10 logarithm of the argument
`LOG2()`	Return the base-2 logarithm of the argument
`MOD()`	Return the remainder
`PI()`	Return the value of pi
`POW()`	Return the argument raised to the specified power

Name	Description
POWER()	Return the argument raised to the specified power
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
TRUNCATE()	Truncate to specified number of decimal places

All mathematical functions return NULL in the event of an error.

- ABS(*X*)

  Returns the absolute value of *X*.

```
mysql> SELECT ABS(2);
        -> 2
mysql> SELECT ABS(-32);
        -> 32
```

  This function is safe to use with BIGINT values.

- ACOS(*X*)

  Returns the arc cosine of *X*, that is, the value whose cosine is *X*. Returns NULL if *X* is not in the range -1 to 1.

```
mysql> SELECT ACOS(1);
        -> 0
mysql> SELECT ACOS(1.0001);
        -> NULL
mysql> SELECT ACOS(0);
        -> 1.5707963267949
```

- ASIN(*X*)

  Returns the arc sine of *X*, that is, the value whose sine is *X*. Returns NULL if *X* is not in the range -1 to 1.

```
mysql> SELECT ASIN(0.2);
        -> 0.20135792079033
mysql> SELECT ASIN('foo');

+-------------+
| ASIN('foo') |
+-------------+
|           0 |
+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
```

```
| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
+---------+------+-----------------------------------------+
```

- `ATAN(X)`

  Returns the arc tangent of *X*, that is, the value whose tangent is *X*.

```
mysql> SELECT ATAN(2);
        -> 1.1071487177941
mysql> SELECT ATAN(-2);
        -> -1.1071487177941
```

- `ATAN(Y,X)`, `ATAN2(Y,X)`

  Returns the arc tangent of the two variables *X* and *Y*. It is similar to calculating the arc tangent of *Y / X*, except that the signs of both arguments are used to determine the quadrant of the result.

```
mysql> SELECT ATAN(-2,2);
        -> -0.78539816339745
mysql> SELECT ATAN2(PI(),0);
        -> 1.5707963267949
```

- `CEIL(X)`

  `CEIL()` is a synonym for `CEILING()`.

- `CEILING(X)`

  Returns the smallest integer value not less than *X*.

```
mysql> SELECT CEILING(1.23);
        -> 2
mysql> SELECT CEILING(-1.23);
        -> -1
```

  For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- `CONV(N,from_base,to_base)`

  Converts numbers between different number bases. Returns a string representation of the number *N*, converted from base *from_base* to base *to_base*. Returns `NULL` if any argument is `NULL`. The argument *N* is interpreted as an integer, but may be specified as an integer or a string. The minimum base is `2` and the maximum base is `36`. If *from_base* is a negative number, *N* is regarded as a signed number. Otherwise, *N* is treated as unsigned. `CONV()` works with 64-bit precision.

```
mysql> SELECT CONV('a',16,2);
        -> '1010'
mysql> SELECT CONV('6E',18,8);
        -> '172'
mysql> SELECT CONV(-17,10,-18);
        -> '-H'
mysql> SELECT CONV(10+'10'+'10'+X'0a',10,10);
        -> '40'
```

- `COS(X)`

  Returns the cosine of *X*, where *X* is given in radians.

```
mysql> SELECT COS(PI());
       -> -1
```

- `COT(X)`

  Returns the cotangent of *X*.

```
mysql> SELECT COT(12);
       -> -1.5726734063977
mysql> SELECT COT(0);
       -> out-of-range error
```

- `CRC32(expr)`

  Computes a cyclic redundancy check value and returns a 32-bit unsigned value. The result is `NULL` if the argument is `NULL`. The argument is expected to be a string and (if possible) is treated as one if it is not.

```
mysql> SELECT CRC32('MySQL');
       -> 3259397556
mysql> SELECT CRC32('mysql');
       -> 2501908538
```

- `DEGREES(X)`

  Returns the argument *X*, converted from radians to degrees.

```
mysql> SELECT DEGREES(PI());
       -> 180
mysql> SELECT DEGREES(PI() / 2);
       -> 90
```

- `EXP(X)`

  Returns the value of *e* (the base of natural logarithms) raised to the power of *X*. The inverse of this function is `LOG()` (using a single argument only) or `LN()`.

```
mysql> SELECT EXP(2);
       -> 7.3890560989307
mysql> SELECT EXP(-2);
       -> 0.13533528323661
mysql> SELECT EXP(0);
       -> 1
```

- `FLOOR(X)`

  Returns the largest integer value not greater than *X*.

```
mysql> SELECT FLOOR(1.23), FLOOR(-1.23);
       -> 1, -2
```

  For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- `FORMAT(X,D)`

Formats the number *X* to a format like `'#,###,###.##'`, rounded to *D* decimal places, and returns the result as a string. For details, see Section 12.5, “String Functions and Operators”.

- `HEX(N_or_S)`

  This function can be used to obtain a hexadecimal representation of a decimal number or a string; the manner in which it does so varies according to the argument's type. See this function's description in Section 12.5, “String Functions and Operators”, for details.

- `LN(X)`

  Returns the natural logarithm of *X*; that is, the base-*e* logarithm of *X*. If *X* is less than or equal to 0.0E0, the function returns `NULL` and a warning “Invalid argument for logarithm” is reported.

```
mysql> SELECT LN(2);
        -> 0.69314718055995
mysql> SELECT LN(-2);
        -> NULL
```

  This function is synonymous with `LOG(X)`. The inverse of this function is the `EXP()` function.

- `LOG(X)`, `LOG(B,X)`

  If called with one parameter, this function returns the natural logarithm of *X*. If *X* is less than or equal to 0.0E0, the function returns `NULL` and a warning “Invalid argument for logarithm” is reported.

  The inverse of this function (when called with a single argument) is the `EXP()` function.

```
mysql> SELECT LOG(2);
        -> 0.69314718055995
mysql> SELECT LOG(-2);
        -> NULL
```

  If called with two parameters, this function returns the logarithm of *X* to the base *B*. If *X* is less than or equal to 0, or if *B* is less than or equal to 1, then `NULL` is returned.

```
mysql> SELECT LOG(2,65536);
        -> 16
mysql> SELECT LOG(10,100);
        -> 2
mysql> SELECT LOG(1,100);
        -> NULL
```

  `LOG(B,X)` is equivalent to `LOG(X) / LOG(B)`.

- `LOG2(X)`

  Returns the base-2 logarithm of *X*. If *X* is less than or equal to 0.0E0, the function returns `NULL` and a warning “Invalid argument for logarithm” is reported.

```
mysql> SELECT LOG2(65536);
        -> 16
mysql> SELECT LOG2(-100);
        -> NULL
```

  `LOG2()` is useful for finding out how many bits a number requires for storage. This function is equivalent to the expression `LOG(X) / LOG(2)`.

- `LOG10(X)`

  Returns the base-10 logarithm of *X*. If *X* is less than or equal to 0.0E0, the function returns `NULL` and a warning “Invalid argument for logarithm” is reported.

```
mysql> SELECT LOG10(2);
        -> 0.30102999566398
mysql> SELECT LOG10(100);
        -> 2
mysql> SELECT LOG10(-100);
        -> NULL
```

  `LOG10(X)` is equivalent to `LOG(10,X)`.

- `MOD(N,M)`, `N % M`, `N MOD M`

  Modulo operation. Returns the remainder of *N* divided by *M*.

```
mysql> SELECT MOD(234, 10);
        -> 4
mysql> SELECT 253 % 7;
        -> 1
mysql> SELECT MOD(29,9);
        -> 2
mysql> SELECT 29 MOD 9;
        -> 2
```

  This function is safe to use with `BIGINT` values.

  `MOD()` also works on values that have a fractional part and returns the exact remainder after division:

```
mysql> SELECT MOD(34.5,3);
        -> 1.5
```

  `MOD(N,0)` returns `NULL`.

- `PI()`

  Returns the value of π (pi). The default number of decimal places displayed is seven, but MySQL uses the full double-precision value internally.

```
mysql> SELECT PI();
        -> 3.141593
mysql> SELECT PI()+0.000000000000000000;
        -> 3.141592653589793116
```

- `POW(X,Y)`

  Returns the value of *X* raised to the power of *Y*.

```
mysql> SELECT POW(2,2);
        -> 4
mysql> SELECT POW(2,-2);
        -> 0.25
```

- `POWER(X,Y)`

  This is a synonym for `POW()`.

- `RADIANS(X)`

  Returns the argument *X*, converted from degrees to radians. (Note that π radians equals 180 degrees.)

```
mysql> SELECT RADIANS(90);
       -> 1.5707963267949
```

- `RAND([N])`

  Returns a random floating-point value *v* in the range `0` <= *v* < `1.0`. To obtain a random integer *R* in the range *i* <= *R* < *j*, use the expression `FLOOR(i + RAND() * (j − i))`. For example, to obtain a random integer in the range the range `7` <= *R* < `12`, use the following statement:

```
SELECT FLOOR(7 + (RAND() * 5));
```

  If an integer argument *N* is specified, it is used as the seed value:

  - With a constant initializer argument, the seed is initialized once when the statement is prepared, prior to execution.
  - With a nonconstant initializer argument (such as a column name), the seed is initialized with the value for each invocation of `RAND()`.

  One implication of this behavior is that for equal argument values, `RAND(N)` returns the same value each time, and thus produces a repeatable sequence of column values. In the following example, the sequence of values produced by `RAND(3)` is the same both places it occurs.

```
mysql> CREATE TABLE t (i INT);
Query OK, 0 rows affected (0.42 sec)

mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT i, RAND() FROM t;
+------+------------------+
| i    | RAND()           |
+------+------------------+
|    1 | 0.61914388706828 |
|    2 | 0.93845168309142 |
|    3 | 0.83482678498591 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+------+------------------+
| i    | RAND(3)          |
+------+------------------+
|    1 | 0.90576975597606 |
|    2 | 0.37307905813035 |
|    3 | 0.14808605345719 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND() FROM t;
+------+------------------+
| i    | RAND()           |
+------+------------------+
|    1 | 0.35877890638893 |
```

```
|    2 | 0.28941420772058 |
|    3 | 0.37073435016976 |
+------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT i, RAND(3) FROM t;
+------+------------------+
| i    | RAND(3)          |
+------+------------------+
|    1 | 0.90576975597606 |
|    2 | 0.37307905813035 |
|    3 | 0.14808605345719 |
+------+------------------+
3 rows in set (0.01 sec)
```

`RAND()` in a `WHERE` clause is evaluated for every row (when selecting from one table) or combination of rows (when selecting from a multiple-table join). Thus, for optimizer purposes, `RAND()` is not a constant value and cannot be used for index optimizations. For more information, see Section 8.2.1.20, “Function Call Optimization”.

Use of a column with `RAND()` values in an `ORDER BY` or `GROUP BY` clause may yield unexpected results because for either clause a `RAND()` expression can be evaluated multiple times for the same row, each time returning a different result. If the goal is to retrieve rows in random order, you can use a statement like this:

```
SELECT * FROM tbl_name ORDER BY RAND();
```

To select a random sample from a set of rows, combine `ORDER BY RAND()` with `LIMIT`:

```
SELECT * FROM table1, table2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000;
```

`RAND()` is not meant to be a perfect random generator. It is a fast way to generate random numbers on demand that is portable between platforms for the same MySQL version.

This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `ROUND(X)`, `ROUND(X,D)`

  Rounds the argument *X* to *D* decimal places. The rounding algorithm depends on the data type of *X*. *D* defaults to 0 if not specified. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

```
mysql> SELECT ROUND(-1.23);
        -> -1
mysql> SELECT ROUND(-1.58);
        -> -2
mysql> SELECT ROUND(1.58);
        -> 2
mysql> SELECT ROUND(1.298, 1);
        -> 1.3
mysql> SELECT ROUND(1.298, 0);
        -> 1
mysql> SELECT ROUND(23.298, -1);
        -> 20
```

The return value has the same type as the first argument (assuming that it is integer, double, or decimal). This means that for an integer argument, the result is an integer (no decimal places):

```
mysql> SELECT ROUND(150.000,2), ROUND(150,2);
+------------------+--------------+
| ROUND(150.000,2) | ROUND(150,2) |
+------------------+--------------+
|           150.00 |          150 |
+------------------+--------------+
```

`ROUND()` uses the following rules depending on the type of the first argument:

- For exact-value numbers, `ROUND()` uses the “round half away from zero” or “round toward nearest” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.
- For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` uses the “round to nearest even” rule: A value with a fractional part exactly halfway between two integers is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

For more information, see Section 12.25, “Precision Math”.

- `SIGN(X)`

  Returns the sign of the argument as `-1`, `0`, or `1`, depending on whether *X* is negative, zero, or positive.

```
mysql> SELECT SIGN(-32);
        -> -1
mysql> SELECT SIGN(0);
        -> 0
mysql> SELECT SIGN(234);
        -> 1
```

- `SIN(X)`

  Returns the sine of *X*, where *X* is given in radians.

```
mysql> SELECT SIN(PI());
        -> 1.2246063538224e-16
mysql> SELECT ROUND(SIN(PI()));
        -> 0
```

- `SQRT(X)`

  Returns the square root of a nonnegative number *X*.

```
mysql> SELECT SQRT(4);
        -> 2
mysql> SELECT SQRT(20);
        -> 4.4721359549996
```

```
mysql> SELECT SQRT(-16);
        -> NULL
```

- `TAN(X)`

  Returns the tangent of *X*, where *X* is given in radians.

```
mysql> SELECT TAN(PI());
        -> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
        -> 1.5574077246549
```

- `TRUNCATE(X,D)`

  Returns the number *X*, truncated to *D* decimal places. If *D* is `0`, the result has no decimal point or fractional part. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

```
mysql> SELECT TRUNCATE(1.223,1);
        -> 1.2
mysql> SELECT TRUNCATE(1.999,1);
        -> 1.9
mysql> SELECT TRUNCATE(1.999,0);
        -> 1
mysql> SELECT TRUNCATE(-1.999,1);
        -> -1.9
mysql> SELECT TRUNCATE(122,-2);
       -> 100
mysql> SELECT TRUNCATE(10.28*100,0);
       -> 1028
```

  All numbers are rounded toward zero.

12.7 Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See Section 11.3, “Date and Time Types”, for a description of the range of values each date and time type has and the valid formats in which values may be specified.

Table 12.13 Date and Time Functions

Name	Description
`ADDDATE()`	Add time values (intervals) to a date value
`ADDTIME()`	Add time
`CONVERT_TZ()`	Convert from one time zone to another
`CURDATE()`	Return the current date
`CURRENT_DATE()`, `CURRENT_DATE`	Synonyms for CURDATE()
`CURRENT_TIME()`, `CURRENT_TIME`	Synonyms for CURTIME()
`CURRENT_TIMESTAMP()`, `CURRENT_TIMESTAMP`	Synonyms for NOW()
`CURTIME()`	Return the current time
`DATE()`	Extract the date part of a date or datetime expression
`DATE_ADD()`	Add time values (intervals) to a date value
`DATE_FORMAT()`	Format date as specified

Name	Description
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT()	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format Unix timestamp as a date
GET_FORMAT()	Return a date format string
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'hh:mm:ss' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments

Name	Description
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
UNIX_TIMESTAMP()	Return a Unix timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

Here is an example that uses date functions. The following query selects all rows with a *`date_col`* value from within the last 30 days:

```
mysql> SELECT something FROM tbl_name
    -> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

The query also selects rows with dates that lie in the future.

Functions that expect date values usually accept datetime values and ignore the time part. Functions that expect time values usually accept datetime values and ignore the date part.

Functions that return the current date or time each are evaluated only once per query at the start of query execution. This means that multiple references to a function such as NOW() within a single query always produce the same result. (For our purposes, a single query also includes a call to a stored program (stored routine, trigger, or event) and all subprograms called by that program.) This principle also applies to CURDATE(), CURTIME(), UTC_DATE(), UTC_TIME(), UTC_TIMESTAMP(), and to any of their synonyms.

The CURRENT_TIMESTAMP(), CURRENT_TIME(), CURRENT_DATE(), and FROM_UNIXTIME() functions return values in the current session time zone, which is available as the session value of the time_zone system variable. In addition, UNIX_TIMESTAMP() assumes that its argument is a datetime value in the session time zone. See Section 5.1.13, “MySQL Server Time Zone Support”.

Some date functions can be used with “zero” dates or incomplete dates such as '2001-11-00', whereas others cannot. Functions that extract parts of dates typically work with incomplete dates and thus can return 0 when you might otherwise expect a nonzero value. For example:

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
       -> 0, 0
```

Other functions expect complete dates and return NULL for incomplete dates. These include functions that perform date arithmetic or that map parts of dates to names. For example:

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
        -> NULL
mysql> SELECT DAYNAME('2006-05-00');
        -> NULL
```

Several functions are more strict when passed a `DATE()` function value as their argument and reject incomplete dates with a day part of zero. These functions are affected: `CONVERT_TZ()`, `DATE_ADD()`, `DATE_SUB()`, `DAYOFYEAR()`, `LAST_DAY()` (permits a day part of zero), `TIMESTAMPDIFF()`, `TO_DAYS()`, `TO_SECONDS()`, `WEEK()`, `WEEKDAY()`, `WEEKOFYEAR()`, `YEARWEEK()`.

Fractional seconds for `TIME`, `DATETIME`, and `TIMESTAMP` values are supported, with up to microsecond precision. Functions that take temporal arguments accept values with fractional seconds. Return values from temporal functions include fractional seconds as appropriate.

- `ADDDATE(`*`date`*`,INTERVAL `*`expr unit`*`)`, `ADDDATE(`*`expr`*`,`*`days`*`)`

  When invoked with the `INTERVAL` form of the second argument, `ADDDATE()` is a synonym for `DATE_ADD()`. The related function `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL` *`unit`* argument, see Temporal Intervals.

  ```
  mysql> SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
          -> '2008-02-02'
  mysql> SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
          -> '2008-02-02'
  ```

  When invoked with the *`days`* form of the second argument, MySQL treats it as an integer number of days to be added to *`expr`*.

  ```
  mysql> SELECT ADDDATE('2008-01-02', 31);
          -> '2008-02-02'
  ```

- `ADDTIME(`*`expr1`*`,`*`expr2`*`)`

  `ADDTIME()` adds *`expr2`* to *`expr1`* and returns the result. *`expr1`* is a time or datetime expression, and *`expr2`* is a time expression.

  ```
  mysql> SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
          -> '2008-01-02 01:01:01.000001'
  mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
          -> '03:00:01.999997'
  ```

- `CONVERT_TZ(`*`dt`*`,`*`from_tz`*`,`*`to_tz`*`)`

  `CONVERT_TZ()` converts a datetime value *`dt`* from the time zone given by *`from_tz`* to the time zone given by *`to_tz`* and returns the resulting value. Time zones are specified as described in Section 5.1.13, “MySQL Server Time Zone Support”. This function returns `NULL` if the arguments are invalid.

  If the value falls out of the supported range of the `TIMESTAMP` type when converted from *`from_tz`* to UTC, no conversion occurs. The `TIMESTAMP` range is described in Section 11.1.2, “Date and Time Type Overview”.

  ```
  mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
          -> '2004-01-01 13:00:00'
  mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');
          -> '2004-01-01 22:00:00'
  ```

Note

To use named time zones such as `'MET'` or `'Europe/Amsterdam'`, the time zone tables must be properly set up. For instructions, see Section 5.1.13, “MySQL Server Time Zone Support”.

- `CURDATE()`

  Returns the current date as a value in *`'YYYY-MM-DD'`* or *`YYYYMMDD`* format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURDATE();
        -> '2008-06-13'
mysql> SELECT CURDATE() + 0;
        -> 20080613
```

- `CURRENT_DATE`, `CURRENT_DATE()`

  `CURRENT_DATE` and `CURRENT_DATE()` are synonyms for `CURDATE()`.

- `CURRENT_TIME`, `CURRENT_TIME([fsp])`

  `CURRENT_TIME` and `CURRENT_TIME()` are synonyms for `CURTIME()`.

- `CURRENT_TIMESTAMP`, `CURRENT_TIMESTAMP([fsp])`

  `CURRENT_TIMESTAMP` and `CURRENT_TIMESTAMP()` are synonyms for `NOW()`.

- `CURTIME([fsp])`

  Returns the current time as a value in *`'hh:mm:ss'`* or *`hhmmss`* format, depending on whether the function is used in a string or numeric context. The value is expressed in the session time zone.

  If the *`fsp`* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT CURTIME();
        -> '23:50:26'
mysql> SELECT CURTIME() + 0;
        -> 235026.000000
```

- `DATE(expr)`

  Extracts the date part of the date or datetime expression *`expr`*.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
        -> '2003-12-31'
```

- `DATEDIFF(expr1,expr2)`

  `DATEDIFF()` returns *`expr1`* − *`expr2`* expressed as a value in days from one date to the other. *`expr1`* and *`expr2`* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
        -> 1
mysql> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
```

```
        -> -31
```

- `DATE_ADD(date,INTERVAL expr unit)`, `DATE_SUB(date,INTERVAL expr unit)`

  These functions perform date arithmetic. The *`date`* argument specifies the starting date or datetime value. *`expr`* is an expression specifying the interval value to be added or subtracted from the starting date. *`expr`* is evaluated as a string; it may start with a `-` for negative intervals. *`unit`* is a keyword indicating the units in which the expression should be interpreted.

  For more information about temporal interval syntax, including a full list of *`unit`* specifiers, the expected form of the *`expr`* argument for each *`unit`* value, and rules for operand interpretation in temporal arithmetic, see Temporal Intervals.

  The return value depends on the arguments:

  - `DATE` if the *`date`* argument is a `DATE` value and your calculations involve only `YEAR`, `MONTH`, and `DAY` parts (that is, no time parts).
  - `DATETIME` if the first argument is a `DATETIME` (or `TIMESTAMP`) value, or if the first argument is a `DATE` and the *`unit`* value uses `HOURS`, `MINUTES`, or `SECONDS`.
  - String otherwise.

  To ensure that the result is `DATETIME`, you can use `CAST()` to convert the first argument to `DATETIME`.

```
mysql> SELECT DATE_ADD('2018-05-01',INTERVAL 1 DAY);
        -> '2018-05-02'
mysql> SELECT DATE_SUB('2018-05-01',INTERVAL 1 YEAR);
        -> '2017-05-01'
mysql> SELECT DATE_ADD('2020-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '2021-01-01 00:00:00'
mysql> SELECT DATE_ADD('2018-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '2019-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
    ->                 INTERVAL '1:1' MINUTE_SECOND);
        -> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2025-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '2024-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
    ->                 INTERVAL '-1 10' DAY_HOUR);
        -> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
    ->            INTERVAL '1.999999' SECOND_MICROSECOND);
        -> '1993-01-01 00:00:01.000001'
```

- `DATE_FORMAT(date,format)`

  Formats the *`date`* value according to the *`format`* string.

  The specifiers shown in the following table may be used in the *`format`* string. The `%` character is required before format specifier characters. The specifiers apply to other functions as well: `STR_TO_DATE()`, `TIME_FORMAT()`, `UNIX_TIMESTAMP()`.

Specifier	Description
%a	Abbreviated weekday name (Sun..Sat)
%b	Abbreviated month name (Jan..Dec)
%c	Month, numeric (0..12)
%D	Day of the month with English suffix (0th, 1st, 2nd, 3rd, …)
%d	Day of the month, numeric (00..31)
%e	Day of the month, numeric (0..31)
%f	Microseconds (000000..999999)
%H	Hour (00..23)
%h	Hour (01..12)
%I	Hour (01..12)
%i	Minutes, numeric (00..59)
%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (*hh:mm:ss* followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (*hh:mm:ss*)
%U	Week (00..53), where Sunday is the first day of the week; WEEK() mode 0
%u	Week (00..53), where Monday is the first day of the week; WEEK() mode 1
%V	Week (01..53), where Sunday is the first day of the week; WEEK() mode 2; used with %X
%v	Week (01..53), where Monday is the first day of the week; WEEK() mode 3; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric (two digits)
%%	A literal % character
%*x*	*x*, for any "*x*" not listed above

Ranges for the month and day specifiers begin with zero due to the fact that MySQL permits the storing of incomplete dates such as `'2014-00-00'`.

The language used for day and month names and abbreviations is controlled by the value of the `lc_time_names` system variable (Section 10.16, “MySQL Server Locale Support”).

For the `%U`, `%u`, `%V`, and `%v` specifiers, see the description of the `WEEK()` function for information about the mode values. The mode affects how week numbering occurs.

`DATE_FORMAT()` returns a string with a character set and collation given by `character_set_connection` and `collation_connection` so that it can return month and weekday names containing non-ASCII characters.

```
mysql> SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
        -> 'Sunday October 2009'
mysql> SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
        -> '22:23:00'
mysql> SELECT DATE_FORMAT('1900-10-04 22:23:00',
    ->                 '%D %y %a %d %m %b %j');
        -> '4th 00 Thu 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
    ->                 '%H %k %I %r %T %S %w');
        -> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
        -> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
        -> '00'
```

- `DATE_SUB(`*`date`*`,INTERVAL `*`expr unit`*`)`

  See the description for `DATE_ADD()`.

- `DAY(`*`date`*`)`

  `DAY()` is a synonym for `DAYOFMONTH()`.

- `DAYNAME(`*`date`*`)`

  Returns the name of the weekday for *`date`*. The language used for the name is controlled by the value of the `lc_time_names` system variable (Section 10.16, “MySQL Server Locale Support”).

```
mysql> SELECT DAYNAME('2007-02-03');
        -> 'Saturday'
```

- `DAYOFMONTH(`*`date`*`)`

  Returns the day of the month for *`date`*, in the range `1` to `31`, or `0` for dates such as `'0000-00-00'` or `'2008-00-00'` that have a zero day part.

```
mysql> SELECT DAYOFMONTH('2007-02-03');
        -> 3
```

- `DAYOFWEEK(`*`date`*`)`

  Returns the weekday index for *`date`* (`1` = Sunday, `2` = Monday, …, `7` = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('2007-02-03');
       -> 7
```

- `DAYOFYEAR(date)`

  Returns the day of the year for *`date`*, in the range `1` to `366`.

```
mysql> SELECT DAYOFYEAR('2007-02-03');
       -> 34
```

- `EXTRACT(unit FROM date)`

  The `EXTRACT()` function uses the same kinds of *`unit`* specifiers as `DATE_ADD()` or `DATE_SUB()`, but extracts parts from the date rather than performing date arithmetic. For information on the *`unit`* argument, see Temporal Intervals.

```
mysql> SELECT EXTRACT(YEAR FROM '2019-07-02');
       -> 2019
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2019-07-02 01:02:03');
       -> 201907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '2019-07-02 01:02:03');
       -> 20102
mysql> SELECT EXTRACT(MICROSECOND
    ->                FROM '2003-01-02 10:30:00.000123');
       -> 123
```

- `FROM_DAYS(N)`

  Given a day number *`N`*, returns a `DATE` value.

```
mysql> SELECT FROM_DAYS(730669);
       -> '2000-07-03'
```

  Use `FROM_DAYS()` with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582). See Section 12.8, “What Calendar Is Used By MySQL?”.

- `FROM_UNIXTIME(unix_timestamp[,format])`

  Returns a representation of the *`unix_timestamp`* argument as a value in *`'YYYY-MM-DD hh:mm:ss'`* or *`YYYYMMDDhhmmss.uuuuuu`* format, depending on whether the function is used in a string or numeric context. *`unix_timestamp`* is an internal timestamp value representing seconds since `'1970-01-01 00:00:00'` UTC, such as produced by the `UNIX_TIMESTAMP()` function.

  The return value is expressed in the session time zone. (Clients can set the session time zone as described in Section 5.1.13, “MySQL Server Time Zone Support”.) The *`format`* string, if given, is used to format the result the same way as described in the entry for the `DATE_FORMAT()` function.

```
mysql> SELECT FROM_UNIXTIME(1447430881);
       -> '2015-11-13 10:08:01'
mysql> SELECT FROM_UNIXTIME(1447430881) + 0;
       -> 20151113100801
mysql> SELECT FROM_UNIXTIME(1447430881,
    ->                      '%Y %D %M %h:%i:%s %x');
       -> '2015 13th November 10:08:01 2015'
```

Note

If you use `UNIX_TIMESTAMP()` and `FROM_UNIXTIME()` to convert between values in a non-UTC time zone and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For details, see the description of the `UNIX_TIMESTAMP()` function.

- `GET_FORMAT({DATE|TIME|DATETIME}, {'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'})`

  Returns a format string. This function is useful in combination with the `DATE_FORMAT()` and the `STR_TO_DATE()` functions.

  The possible values for the first and second arguments result in several possible format strings (for the specifiers used, see the table in the `DATE_FORMAT()` function description). ISO format refers to ISO 9075, not ISO 8601.

Function Call	Result
`GET_FORMAT(DATE,'USA')`	`'%m.%d.%Y'`
`GET_FORMAT(DATE,'JIS')`	`'%Y-%m-%d'`
`GET_FORMAT(DATE,'ISO')`	`'%Y-%m-%d'`
`GET_FORMAT(DATE,'EUR')`	`'%d.%m.%Y'`
`GET_FORMAT(DATE,'INTERNAL')`	`'%Y%m%d'`
`GET_FORMAT(DATETIME,'USA')`	`'%Y-%m-%d %H.%i.%s'`
`GET_FORMAT(DATETIME,'JIS')`	`'%Y-%m-%d %H:%i:%s'`
`GET_FORMAT(DATETIME,'ISO')`	`'%Y-%m-%d %H:%i:%s'`
`GET_FORMAT(DATETIME,'EUR')`	`'%Y-%m-%d %H.%i.%s'`
`GET_FORMAT(DATETIME,'INTERNAL')`	`'%Y%m%d%H%i%s'`
`GET_FORMAT(TIME,'USA')`	`'%h:%i:%s %p'`
`GET_FORMAT(TIME,'JIS')`	`'%H:%i:%s'`
`GET_FORMAT(TIME,'ISO')`	`'%H:%i:%s'`
`GET_FORMAT(TIME,'EUR')`	`'%H.%i.%s'`
`GET_FORMAT(TIME,'INTERNAL')`	`'%H%i%s'`

  `TIMESTAMP` can also be used as the first argument to `GET_FORMAT()`, in which case the function returns the same values as for `DATETIME`.

```
mysql> SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
        -> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
        -> '2003-10-31'
```

- `HOUR(time)`

  Returns the hour for *`time`*. The range of the return value is `0` to `23` for time-of-day values. However, the range of `TIME` values actually is much larger, so `HOUR` can return values greater than `23`.

```
mysql> SELECT HOUR('10:05:03');
        -> 10
mysql> SELECT HOUR('272:59:59');
```

```
        -> 272
```

- `LAST_DAY(date)`

  Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns `NULL` if the argument is invalid.

```
mysql> SELECT LAST_DAY('2003-02-05');
        -> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
        -> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
        -> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
        -> NULL
```

- `LOCALTIME`, `LOCALTIME([fsp])`

  `LOCALTIME` and `LOCALTIME()` are synonyms for `NOW()`.

- `LOCALTIMESTAMP`, `LOCALTIMESTAMP([fsp])`

  `LOCALTIMESTAMP` and `LOCALTIMESTAMP()` are synonyms for `NOW()`.

- `MAKEDATE(year,dayofyear)`

  Returns a date, given year and day-of-year values. *`dayofyear`* must be greater than 0 or the result is `NULL`.

```
mysql> SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
        -> '2011-01-31', '2011-02-01'
mysql> SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
        -> '2011-12-31', '2014-12-31'
mysql> SELECT MAKEDATE(2011,0);
        -> NULL
```

- `MAKETIME(hour,minute,second)`

  Returns a time value calculated from the *`hour`*, *`minute`*, and *`second`* arguments.

  The *`second`* argument can have a fractional part.

```
mysql> SELECT MAKETIME(12,15,30);
        -> '12:15:30'
```

- `MICROSECOND(expr)`

  Returns the microseconds from the time or datetime expression *`expr`* as a number in the range from `0` to `999999`.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
        -> 123456
mysql> SELECT MICROSECOND('2019-12-31 23:59:59.000010');
        -> 10
```

- `MINUTE(time)`

  Returns the minute for *`time`*, in the range `0` to `59`.

```
mysql> SELECT MINUTE('2008-02-03 10:05:03');
        -> 5
```

- `MONTH(date)`

  Returns the month for *`date`*, in the range `1` to `12` for January to December, or `0` for dates such as `'0000-00-00'` or `'2008-00-00'` that have a zero month part.

```
mysql> SELECT MONTH('2008-02-03');
        -> 2
```

- `MONTHNAME(date)`

  Returns the full name of the month for *`date`*. The language used for the name is controlled by the value of the `lc_time_names` system variable (Section 10.16, “MySQL Server Locale Support”).

```
mysql> SELECT MONTHNAME('2008-02-03');
        -> 'February'
```

- `NOW([fsp])`

  Returns the current date and time as a value in *`'YYYY-MM-DD hh:mm:ss'`* or *`YYYYMMDDhhmmss`* format, depending on whether the function is used in a string or numeric context. The value is expressed in the session time zone.

  If the *`fsp`* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT NOW();
        -> '2007-12-15 23:50:26'
mysql> SELECT NOW() + 0;
        -> 20071215235026.000000
```

  `NOW()` returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, `NOW()` returns the time at which the function or triggering statement began to execute.) This differs from the behavior for `SYSDATE()`, which returns the exact time at which it executes.

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:36 |        0 | 2006-04-12 13:47:36 |
+---------------------+----------+---------------------+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:44 |        0 | 2006-04-12 13:47:46 |
+---------------------+----------+---------------------+
```

  In addition, the `SET TIMESTAMP` statement affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`. Setting the timestamp to a nonzero value causes each subsequent invocation of `NOW()` to return that

value. Setting the timestamp to zero cancels this effect so that `NOW()` once again returns the current date and time.

See the description for `SYSDATE()` for additional information about the differences between the two functions.

- `PERIOD_ADD(P,N)`

  Adds `N` months to period `P` (in the format `YYMM` or `YYYYMM`). Returns a value in the format `YYYYMM`.

  Note

  The period argument `P` is *not* a date value.

```
mysql> SELECT PERIOD_ADD(200801,2);
        -> 200803
```

- `PERIOD_DIFF(P1,P2)`

  Returns the number of months between periods `P1` and `P2`. `P1` and `P2` should be in the format `YYMM` or `YYYYMM`. Note that the period arguments `P1` and `P2` are *not* date values.

```
mysql> SELECT PERIOD_DIFF(200802,200703);
        -> 11
```

- `QUARTER(date)`

  Returns the quarter of the year for `date`, in the range `1` to `4`.

```
mysql> SELECT QUARTER('2008-04-01');
        -> 2
```

- `SECOND(time)`

  Returns the second for `time`, in the range `0` to `59`.

```
mysql> SELECT SECOND('10:05:03');
        -> 3
```

- `SEC_TO_TIME(seconds)`

  Returns the `seconds` argument, converted to hours, minutes, and seconds, as a `TIME` value. The range of the result is constrained to that of the `TIME` data type. A warning occurs if the argument corresponds to a value outside that range.

```
mysql> SELECT SEC_TO_TIME(2378);
        -> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
        -> 3938
```

- `STR_TO_DATE(str,format)`

  This is the inverse of the `DATE_FORMAT()` function. It takes a string `str` and a format string `format`. `STR_TO_DATE()` returns a `DATETIME` value if the format string contains both date and time parts, or

a DATE or TIME value if the string contains only date or time parts. If the date, time, or datetime value extracted from *str* is illegal, STR_TO_DATE() returns NULL and produces a warning.

The server scans *str* attempting to match *format* to it. The format string can contain literal characters and format specifiers beginning with %. Literal characters in *format* must match literally in *str*. Format specifiers in *format* must match a date or time part in *str*. For the specifiers that can be used in *format*, see the DATE_FORMAT() function description.

```
mysql> SELECT STR_TO_DATE('01,5,2013','%d,%m,%Y');
        -> '2013-05-01'
mysql> SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y');
        -> '2013-05-01'
```

Scanning starts at the beginning of *str* and fails if *format* is found not to match. Extra characters at the end of *str* are ignored.

```
mysql> SELECT STR_TO_DATE('a09:30:17','a%h:%i:%s');
        -> '09:30:17'
mysql> SELECT STR_TO_DATE('a09:30:17','%h:%i:%s');
        -> NULL
mysql> SELECT STR_TO_DATE('09:30:17a','%h:%i:%s');
        -> '09:30:17'
```

Unspecified date or time parts have a value of 0, so incompletely specified values in *str* produce a result with some or all parts set to 0:

```
mysql> SELECT STR_TO_DATE('abc','abc');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('9','%m');
        -> '0000-09-00'
mysql> SELECT STR_TO_DATE('9','%s');
        -> '00:00:09'
```

Range checking on the parts of date values is as described in Section 11.3.1, “The DATE, DATETIME, and TIMESTAMP Types”. This means, for example, that “zero” dates or dates with part values of 0 are permitted unless the SQL mode is set to disallow such values.

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
        -> '2004-04-31'
```

If the NO_ZERO_DATE or NO_ZERO_IN_DATE SQL mode is enabled, zero dates or part of dates are disallowed. In that case, STR_TO_DATE() returns NULL and generates a warning:

```
mysql> SET sql_mode = '';
mysql> SELECT STR_TO_DATE('15:35:00', '%H:%i:%s');
+------------------------------------+
| STR_TO_DATE('15:35:00', '%H:%i:%s') |
+------------------------------------+
| 15:35:00                           |
+------------------------------------+
mysql> SET sql_mode = 'NO_ZERO_IN_DATE';
mysql> SELECT STR_TO_DATE('15:35:00', '%h:%i:%s');
+------------------------------------+
| STR_TO_DATE('15:35:00', '%h:%i:%s') |
+------------------------------------+
| NULL                               |
```

```
+-----------------------------------+
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1411
Message: Incorrect datetime value: '15:35:00' for function str_to_date
```

Note

You cannot use format `"%X%V"` to convert a year-week string to a date because the combination of a year and week does not uniquely identify a year and month if the week crosses a month boundary. To convert a year-week to a date, you should also specify the weekday:

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
        -> '2004-10-18'
```

- `SUBDATE(`*`date`*`,INTERVAL `*`expr unit`*`)`, `SUBDATE(`*`expr`*`,`*`days`*`)`

  When invoked with the `INTERVAL` form of the second argument, `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL` *`unit`* argument, see the discussion for `DATE_ADD()`.

```
mysql> SELECT DATE_SUB('2008-01-02', INTERVAL 31 DAY);
        -> '2007-12-02'
mysql> SELECT SUBDATE('2008-01-02', INTERVAL 31 DAY);
        -> '2007-12-02'
```

  The second form enables the use of an integer value for *`days`*. In such cases, it is interpreted as the number of days to be subtracted from the date or datetime expression *`expr`*.

```
mysql> SELECT SUBDATE('2008-01-02 12:00:00', 31);
        -> '2007-12-02 12:00:00'
```

- `SUBTIME(`*`expr1`*`,`*`expr2`*`)`

  `SUBTIME()` returns *`expr1`* − *`expr2`* expressed as a value in the same format as *`expr1`*. *`expr1`* is a time or datetime expression, and *`expr2`* is a time expression.

```
mysql> SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002');
        -> '2007-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
        -> '-00:59:59.999999'
```

- `SYSDATE([`*`fsp`*`])`

  Returns the current date and time as a value in *`'YYYY-MM-DD hh:mm:ss'`* or *`YYYYMMDDhhmmss`* format, depending on whether the function is used in a string or numeric context.

  If the *`fsp`* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

  `SYSDATE()` returns the time at which it executes. This differs from the behavior for `NOW()`, which returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, `NOW()` returns the time at which the function or triggering statement began to execute.)

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
```

```
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:36 |        0 | 2006-04-12 13:47:36 |
+---------------------+----------+---------------------+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:44 |        0 | 2006-04-12 13:47:46 |
+---------------------+----------+---------------------+
```

In addition, the `SET TIMESTAMP` statement affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`.

Because `SYSDATE()` can return different values even within the same statement, and is not affected by `SET TIMESTAMP`, it is nondeterministic and therefore unsafe for replication if statement-based binary logging is used. If that is a problem, you can use row-based logging.

Alternatively, you can use the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`. This works if the option is used on both the master and the slave.

The nondeterministic nature of `SYSDATE()` also means that indexes cannot be used for evaluating expressions that refer to it.

- `TIME(expr)`

  Extracts the time part of the time or datetime expression *`expr`* and returns it as a string.

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
        -> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
        -> '01:02:03.000123'
```

- `TIMEDIFF(expr1,expr2)`

  `TIMEDIFF()` returns *`expr1`* − *`expr2`* expressed as a time value. *`expr1`* and *`expr2`* are time or date-and-time expressions, but both must be of the same type.

  The result returned by `TIMEDIFF()` is limited to the range allowed for `TIME` values. Alternatively, you can use either of the functions `TIMESTAMPDIFF()` and `UNIX_TIMESTAMP()`, both of which return integers.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
    ->                 '2000:01:01 00:00:00.000001');
        -> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('2008-12-31 23:59:59.000001',
    ->                 '2008-12-30 01:01:01.000002');
        -> '46:58:57.999999'
```

- `TIMESTAMP(expr)`, `TIMESTAMP(expr1,expr2)`

  With a single argument, this function returns the date or datetime expression *`expr`* as a datetime value. With two arguments, it adds the time expression *`expr2`* to the date or datetime expression *`expr1`* and returns the result as a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
        -> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
        -> '2004-01-01 00:00:00'
```

- `TIMESTAMPADD(unit,interval,datetime_expr)`

  Adds the integer expression *`interval`* to the date or datetime expression *`datetime_expr`*. The unit for *`interval`* is given by the *`unit`* argument, which should be one of the following values: `MICROSECOND` (microseconds), `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, or `YEAR`.

  The *`unit`* value may be specified using one of keywords as shown, or with a prefix of `SQL_TSI_`. For example, `DAY` and `SQL_TSI_DAY` both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
        -> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
        -> '2003-01-09'
```

- `TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)`

  Returns *`datetime_expr2`* − *`datetime_expr1`*, where *`datetime_expr1`* and *`datetime_expr2`* are date or datetime expressions. One expression may be a date and the other a datetime; a date value is treated as a datetime having the time part `'00:00:00'` where necessary. The unit for the result (an integer) is given by the *`unit`* argument. The legal values for *`unit`* are the same as those listed in the description of the `TIMESTAMPADD()` function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
        -> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
        -> -1
mysql> SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
        -> 128885
```

Note

The order of the date or datetime arguments for this function is the opposite of that used with the `TIMESTAMP()` function when invoked with 2 arguments.

- `TIME_FORMAT(time,format)`

  This is used like the `DATE_FORMAT()` function, but the *`format`* string may contain format specifiers only for hours, minutes, seconds, and microseconds. Other specifiers produce a `NULL` value or `0`.

  If the *`time`* value contains an hour part that is greater than `23`, the `%H` and `%k` hour format specifiers produce a value larger than the usual range of `0..23`. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
        -> '100 100 04 04 4'
```

- `TIME_TO_SEC(time)`

  Returns the *`time`* argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
        -> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
        -> 2378
```

- `TO_DAYS(date)`

  Given a date *`date`*, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
        -> 728779
mysql> SELECT TO_DAYS('2007-10-07');
        -> 733321
```

  `TO_DAYS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See Section 12.8, “What Calendar Is Used By MySQL?”, for details.

  Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in Section 11.3, “Date and Time Types”. For example, `'2008-10-07'` and `'08-10-07'` are seen as identical dates:

```
mysql> SELECT TO_DAYS('2008-10-07'), TO_DAYS('08-10-07');
        -> 733687, 733687
```

  In MySQL, the zero date is defined as `'0000-00-00'`, even though this date is itself considered invalid. This means that, for `'0000-00-00'` and `'0000-01-01'`, `TO_DAYS()` returns the values shown here:

```
mysql> SELECT TO_DAYS('0000-00-00');
+-----------------------+
| to_days('0000-00-00') |
+-----------------------+
|                  NULL |
+-----------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT TO_DAYS('0000-01-01');
+-----------------------+
| to_days('0000-01-01') |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)
```

  This is true whether or not the `ALLOW_INVALID_DATES` SQL server mode is enabled.

- `TO_SECONDS(expr)`

  Given a date or datetime *`expr`*, returns the number of seconds since the year 0. If *`expr`* is not a valid date or datetime value, returns `NULL`.

```
mysql> SELECT TO_SECONDS(950501);
        -> 62966505600
mysql> SELECT TO_SECONDS('2009-11-29');
        -> 63426672000
mysql> SELECT TO_SECONDS('2009-11-29 13:43:32');
        -> 63426721412
mysql> SELECT TO_SECONDS( NOW() );
        -> 63426721458
```

Like `TO_DAYS()`, `TO_SECONDS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See Section 12.8, “What Calendar Is Used By MySQL?”, for details.

Like `TO_DAYS()`, `TO_SECONDS()`, converts two-digit year values in dates to four-digit form using the rules in Section 11.3, “Date and Time Types”.

In MySQL, the zero date is defined as `'0000-00-00'`, even though this date is itself considered invalid. This means that, for `'0000-00-00'` and `'0000-01-01'`, `TO_SECONDS()` returns the values shown here:

```
mysql> SELECT TO_SECONDS('0000-00-00');
+--------------------------+
| TO_SECONDS('0000-00-00') |
+--------------------------+
|                     NULL |
+--------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+---------+------+---------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT TO_SECONDS('0000-01-01');
+--------------------------+
| TO_SECONDS('0000-01-01') |
+--------------------------+
|                    86400 |
+--------------------------+
1 row in set (0.00 sec)
```

This is true whether or not the `ALLOW_INVALID_DATES` SQL server mode is enabled.

- `UNIX_TIMESTAMP([date])`

  If `UNIX_TIMESTAMP()` is called with no *`date`* argument, it returns a Unix timestamp representing seconds since `'1970-01-01 00:00:00'` UTC.

  If `UNIX_TIMESTAMP()` is called with a *`date`* argument, it returns the value of the argument as seconds since `'1970-01-01 00:00:00'` UTC. The server interprets *`date`* as a value in the session time zone and converts it to an internal Unix timestamp value in UTC. (Clients can set the session time zone as described in Section 5.1.13, “MySQL Server Time Zone Support”.) The *`date`* argument may be a `DATE`, `DATETIME`, or `TIMESTAMP` string, or a number in *`YYMMDD`*, *`YYMMDDhhmmss`*, *`YYYYMMDD`*, or

YYYYMMDDhhmmss format. If the argument includes a time part, it may optionally include a fractional seconds part.

The return value is an integer if no argument is given or the argument does not include a fractional seconds part, or DECIMAL if an argument is given that includes a fractional seconds part.

When the *date* argument is a TIMESTAMP column, UNIX_TIMESTAMP() returns the internal timestamp value directly, with no implicit "string-to-Unix-timestamp" conversion.

The valid range of argument values is the same as for the TIMESTAMP data type: '1970-01-01 00:00:01.000000' UTC to '2038-01-19 03:14:07.999999' UTC. If you pass an out-of-range date to UNIX_TIMESTAMP(), it returns 0.

```
mysql> SELECT UNIX_TIMESTAMP();
        -> 1447431666
mysql> SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19');
        -> 1447431619
mysql> SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19.012');
        -> 1447431619.012
```

If you use UNIX_TIMESTAMP() and FROM_UNIXTIME() to convert between values in a non-UTC time zone and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For example, due to conventions for local time zone changes such as Daylight Saving Time (DST), it is possible for UNIX_TIMESTAMP() to map two values that are distinct in a non-UTC time zone to the same Unix timestamp value. FROM_UNIXTIME() will map that value back to only one of the original values. Here is an example, using values that are distinct in the MET time zone:

```
mysql> SET time_zone = 'MET';
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT FROM_UNIXTIME(1111885200);
+---------------------------+
| FROM_UNIXTIME(1111885200) |
+---------------------------+
| 2005-03-27 03:00:00       |
+---------------------------+
```

Note

To use named time zones such as 'MET' or 'Europe/Amsterdam', the time zone tables must be properly set up. For instructions, see Section 5.1.13, "MySQL Server Time Zone Support".

If you want to subtract UNIX_TIMESTAMP() columns, you might want to cast them to signed integers. See Section 12.10, "Cast Functions and Operators".

- UTC_DATE, UTC_DATE()

Returns the current UTC date as a value in *'YYYY-MM-DD'* or *YYYYMMDD* format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
        -> '2003-08-14', 20030814
```

- `UTC_TIME`, `UTC_TIME([fsp])`

  Returns the current UTC time as a value in *'hh:mm:ss'* or *hhmmss* format, depending on whether the function is used in a string or numeric context.

  If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
        -> '18:07:53', 180753.000000
```

- `UTC_TIMESTAMP`, `UTC_TIMESTAMP([fsp])`

  Returns the current UTC date and time as a value in *'YYYY-MM-DD hh:mm:ss'* or *YYYYMMDDhhmmss* format, depending on whether the function is used in a string or numeric context.

  If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
        -> '2003-08-14 18:08:04', 20030814180804.000000
```

- `WEEK(date[,mode])`

  This function returns the week number for *date*. The two-argument form of `WEEK()` enables you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from `0` to `53` or from `1` to `53`. If the *mode* argument is omitted, the value of the `default_week_format` system variable is used. See Section 5.1.8, “Server System Variables”.

  The following table describes how the *mode* argument works.

Mode	First day of week	Range	Week 1 is the first week …
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with 4 or more days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with 4 or more days this year
4	Sunday	0-53	with 4 or more days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with 4 or more days this year
7	Monday	1-53	with a Monday in this year

  For *mode* values with a meaning of “with 4 or more days this year,” weeks are numbered according to ISO 8601:1988:

  - If the week containing January 1 has 4 or more days in the new year, it is week 1.

- Otherwise, it is the last week of the previous year, and the next week is week 1.

```
mysql> SELECT WEEK('2008-02-20');
        -> 7
mysql> SELECT WEEK('2008-02-20',0);
        -> 7
mysql> SELECT WEEK('2008-02-20',1);
        -> 8
mysql> SELECT WEEK('2008-12-31',1);
        -> 53
```

If a date falls in the last week of the previous year, MySQL returns `0` if you do not use `2`, `3`, `6`, or `7` as the optional *`mode`* argument:

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
        -> 2000, 0
```

One might argue that `WEEK()` should return `52` because the given date actually occurs in the 52nd week of 1999. `WEEK()` returns `0` instead so that the return value is "the week number in the given year." This makes use of the `WEEK()` function reliable when combined with other functions that extract a date part from a date.

If you prefer a result evaluated with respect to the year that contains the first day of the week for the given date, use `0`, `2`, `5`, or `7` as the optional *`mode`* argument.

```
mysql> SELECT WEEK('2000-01-01',2);
        -> 52
```

Alternatively, use the `YEARWEEK()` function:

```
mysql> SELECT YEARWEEK('2000-01-01');
        -> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
        -> '52'
```

- `WEEKDAY(`*`date`*`)`

  Returns the weekday index for *`date`* (`0` = Monday, `1` = Tuesday, … `6` = Sunday).

```
mysql> SELECT WEEKDAY('2008-02-03 22:23:00');
        -> 6
mysql> SELECT WEEKDAY('2007-11-06');
        -> 1
```

- `WEEKOFYEAR(`*`date`*`)`

  Returns the calendar week of the date as a number in the range from `1` to `53`. `WEEKOFYEAR()` is a compatibility function that is equivalent to `WEEK(`*`date`*`,3)`.

```
mysql> SELECT WEEKOFYEAR('2008-02-20');
        -> 8
```

- `YEAR(`*`date`*`)`

  Returns the year for *`date`*, in the range `1000` to `9999`, or `0` for the "zero" date.

```
mysql> SELECT YEAR('1987-01-01');
        -> 1987
```

- `YEARWEEK(date)`, `YEARWEEK(date,mode)`

  Returns year and week for a date. The year in the result may be different from the year in the date argument for the first and the last week of the year.

  The *`mode`* argument works exactly like the *`mode`* argument to `WEEK()`. For the single-argument syntax, a *`mode`* value of 0 is used. Unlike `WEEK()`, the value of `default_week_format` does not influence `YEARWEEK()`.

```
mysql> SELECT YEARWEEK('1987-01-01');
        -> 198652
```

  The week number is different from what the `WEEK()` function would return (`0`) for optional arguments `0` or `1`, as `WEEK()` then returns the week in the context of the given year.

12.8 What Calendar Is Used By MySQL?

MySQL uses what is known as a *proleptic Gregorian calendar*.

Every country that has switched from the Julian to the Gregorian calendar has had to discard at least ten days during the switch. To see how this works, consider the month of October 1582, when the first Julian-to-Gregorian switch occurred.

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

There are no dates between October 4 and October 15. This discontinuity is called the *cutover*. Any dates before the cutover are Julian, and any dates following the cutover are Gregorian. Dates during a cutover are nonexistent.

A calendar applied to dates when it was not actually in use is called *proleptic*. Thus, if we assume there was never a cutover and Gregorian rules always rule, we have a proleptic Gregorian calendar. This is what is used by MySQL, as is required by standard SQL. For this reason, dates prior to the cutover stored as MySQL `DATE` or `DATETIME` values must be adjusted to compensate for the difference. It is important to realize that the cutover did not occur at the same time in all countries, and that the later it happened, the more days were lost. For example, in Great Britain, it took place in 1752, when Wednesday September 2 was followed by Thursday September 14. Russia remained on the Julian calendar until 1918, losing 13 days in the process, and what is popularly referred to as its “October Revolution” occurred in November according to the Gregorian calendar.

12.9 Full-Text Search Functions

`MATCH (col1,col2,...) AGAINST (expr [search_modifier])`

```
search_modifier:
  {
       IN NATURAL LANGUAGE MODE
     | IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
```

```
    | IN BOOLEAN MODE
    | WITH QUERY EXPANSION
}
```

MySQL has support for full-text indexing and searching:

- A full-text index in MySQL is an index of type `FULLTEXT`.
- Full-text indexes can be used only with `InnoDB` or `MyISAM` tables, and can be created only for `CHAR`, `VARCHAR`, or `TEXT` columns.
- `MySQL` provides a built-in full-text ngram parser that supports Chinese, Japanese, and Korean (CJK), and an installable MeCab full-text parser plugin for Japanese. Parsing differences are outlined in Section 12.9.8, “ngram Full-Text Parser”, and Section 12.9.9, “MeCab Full-Text Parser Plugin”.
- A `FULLTEXT` index definition can be given in the `CREATE TABLE` statement when a table is created, or added later using `ALTER TABLE` or `CREATE INDEX`.
- For large data sets, it is much faster to load your data into a table that has no `FULLTEXT` index and then create the index after that, than to load data into a table that has an existing `FULLTEXT` index.

Full-text searching is performed using `MATCH() ... AGAINST` syntax. `MATCH()` takes a comma-separated list that names the columns to be searched. `AGAINST` takes a string to search for, and an optional modifier that indicates what type of search to perform. The search string must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.

There are three types of full-text searches:

- A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators, with the exception of double quote (") characters. The stopword list applies. For more information about stopword lists, see Section 12.9.4, “Full-Text Stopwords”.

  Full-text searches are natural language searches if the `IN NATURAL LANGUAGE MODE` modifier is given or if no modifier is given. For more information, see Section 12.9.1, “Natural Language Full-Text Searches”.

- A boolean search interprets the search string using the rules of a special query language. The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual. Certain common words (stopwords) are omitted from the search index and do not match if present in the search string. The `IN BOOLEAN MODE` modifier specifies a boolean search. For more information, see Section 12.9.2, “Boolean Full-Text Searches”.

- A query expansion search is a modification of a natural language search. The search string is used to perform a natural language search. Then words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search. The `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` or `WITH QUERY EXPANSION` modifier specifies a query expansion search. For more information, see Section 12.9.3, “Full-Text Searches with Query Expansion”.

For information about `FULLTEXT` query performance, see Section 8.3.5, “Column Indexes”.

For more information about `InnoDB FULLTEXT` indexes, see Section 15.6.2.4, “InnoDB FULLTEXT Indexes”.

Constraints on full-text searching are listed in Section 12.9.5, “Full-Text Restrictions”.

The myisam_ftdump utility dumps the contents of a MyISAM full-text index. This may be helpful for debugging full-text queries. See Section 4.6.3, “myisam_ftdump — Display Full-Text Index information”.

12.9.1 Natural Language Full-Text Searches

By default or with the IN NATURAL LANGUAGE MODE modifier, the MATCH() function performs a natural language search for a string against a *text collection*. A collection is a set of one or more columns included in a FULLTEXT index. The search string is given as the argument to AGAINST(). For each row in the table, MATCH() returns a relevance value; that is, a similarity measure between the search string and the text in that row in the columns named in the MATCH() list.

```
mysql> CREATE TABLE articles (
          id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
          title VARCHAR(200),
          body TEXT,
          FULLTEXT (title,body)
        ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO articles (title,body) VALUES
        ('MySQL Tutorial','DBMS stands for DataBase ...'),
        ('How To Use MySQL Well','After you went through a ...'),
        ('Optimizing MySQL','In this tutorial we will show ...'),
        ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
        ('MySQL vs. YourSQL','In the following database comparison ...'),
        ('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM articles
        WHERE MATCH (title,body)
        AGAINST ('database' IN NATURAL LANGUAGE MODE);
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
+----+-------------------+------------------------------------------+
2 rows in set (0.00 sec)
```

By default, the search is performed in case-insensitive fashion. To perform a case-sensitive full-text search, use a case-sensitive or binary collation for the indexed columns. For example, a column that uses the utf8mb4 character set of can be assigned a collation of utf8mb4_0900_as_cs or utf8mb4_bin to make it case-sensitive for full-text searches.

When MATCH() is used in a WHERE clause, as in the example shown earlier, the rows returned are automatically sorted with the highest relevance first. Relevance values are nonnegative floating-point numbers. Zero relevance means no similarity. Relevance is computed based on the number of words in the row (document), the number of unique words in the row, the total number of words in the collection, and the number of rows that contain a particular word.

Note

The term “document” may be used interchangeably with the term “row”, and both terms refer to the indexed part of the row. The term “collection” refers to the indexed columns and encompasses all rows.

To simply count matches, you could use a query like this:

```
mysql> SELECT COUNT(*) FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' IN NATURAL LANGUAGE MODE);
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

You might find it quicker to rewrite the query as follows:

```
mysql> SELECT
    COUNT(IF(MATCH (title,body) AGAINST ('database' IN NATURAL LANGUAGE MODE), 1, NULL))
    AS count
    FROM articles;
+-------+
| count |
+-------+
|     2 |
+-------+
1 row in set (0.03 sec)
```

The first query does some extra work (sorting the results by relevance) but also can use an index lookup based on the `WHERE` clause. The index lookup might make the first query faster if the search matches few rows. The second query performs a full table scan, which might be faster than the index lookup if the search term was present in most rows.

For natural-language full-text searches, the columns named in the `MATCH()` function must be the same columns included in some `FULLTEXT` index in your table. For the preceding query, note that the columns named in the `MATCH()` function (`title` and `body`) are the same as those named in the definition of the `article` table's `FULLTEXT` index. To search the `title` or `body` separately, you would create separate `FULLTEXT` indexes for each column.

You can also perform a boolean search or a search with query expansion. These search types are described in Section 12.9.2, “Boolean Full-Text Searches”, and Section 12.9.3, “Full-Text Searches with Query Expansion”.

A full-text search that uses an index can name columns only from a single table in the `MATCH()` clause because an index cannot span multiple tables. For `MyISAM` tables, a boolean search can be done in the absence of an index (albeit more slowly), in which case it is possible to name columns from multiple tables.

The preceding example is a basic illustration that shows how to use the `MATCH()` function where rows are returned in order of decreasing relevance. The next example shows how to retrieve the relevance values explicitly. Returned rows are not ordered because the `SELECT` statement includes neither `WHERE` nor `ORDER BY` clauses:

```
mysql> SELECT id, MATCH (title,body)
    AGAINST ('Tutorial' IN NATURAL LANGUAGE MODE) AS score
    FROM articles;
+----+----------------------+
| id | score                |
+----+----------------------+
|  1 | 0.22764469683170319 |
|  2 |                    0 |
|  3 | 0.22764469683170319 |
|  4 |                    0 |
|  5 |                    0 |
|  6 |                    0 |
+----+----------------------+
```

```
6 rows in set (0.00 sec)
```

The following example is more complex. The query returns the relevance values and it also sorts the rows in order of decreasing relevance. To achieve this result, specify `MATCH()` twice: once in the `SELECT` list and once in the `WHERE` clause. This causes no additional overhead, because the MySQL optimizer notices that the two `MATCH()` calls are identical and invokes the full-text search code only once.

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
    ('Security implications of running MySQL as root'
    IN NATURAL LANGUAGE MODE) AS score
    FROM articles WHERE MATCH (title,body) AGAINST
    ('Security implications of running MySQL as root'
    IN NATURAL LANGUAGE MODE);
+----+-------------------------------------+-----------------+
| id | body                                | score           |
+----+-------------------------------------+-----------------+
|  4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
|  6 | When configured properly, MySQL ... | 1.3114095926285 |
+----+-------------------------------------+-----------------+
2 rows in set (0.00 sec)
```

A phrase that is enclosed within double quote (`"`) characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words and performs a search in the `FULLTEXT` index for the words. Nonword characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, `"test phrase"` matches `"test, phrase"`. If the phrase contains no words that are in the index, the result is empty. For example, if all words are either stopwords or shorter than the minimum length of indexed words, the result is empty.

The MySQL `FULLTEXT` implementation regards any sequence of true word characters (letters, digits, and underscores) as a word. That sequence may also contain apostrophes (`'`), but not more than one in a row. This means that `aaa'bbb` is regarded as one word, but `aaa''bbb` is regarded as two words. Apostrophes at the beginning or the end of a word are stripped by the `FULLTEXT` parser; `'aaa'bbb'` would be parsed as `aaa'bbb`.

The built-in `FULLTEXT` parser determines where words start and end by looking for certain delimiter characters; for example, (space), `,` (comma), and `.` (period). If words are not separated by delimiters (as in, for example, Chinese), the built-in `FULLTEXT` parser cannot determine where a word begins or ends. To be able to add words or other indexed terms in such languages to a `FULLTEXT` index that uses the built-in `FULLTEXT` parser, you must preprocess them so that they are separated by some arbitrary delimiter. Alternatively, you can create `FULLTEXT` indexes using the ngram parser plugin (for Chinese, Japanese, or Korean) or the MeCab parser plugin (for Japanese).

It is possible to write a plugin that replaces the built-in full-text parser. For details, see Section 29.2, “The MySQL Plugin API”. For example parser plugin source code, see the `plugin/fulltext` directory of a MySQL source distribution.

Some words are ignored in full-text searches:

- Any word that is too short is ignored. The default minimum length of words that are found by full-text searches is three characters for `InnoDB` search indexes, or four characters for `MyISAM`. You can control the cutoff by setting a configuration option before creating the index: `innodb_ft_min_token_size` configuration option for `InnoDB` search indexes, or `ft_min_word_len` for `MyISAM`.

Note

This behavior does not apply to `FULLTEXT` indexes that use the ngram parser. For the ngram parser, token length is defined by the `ngram_token_size` option.

- Words in the stopword list are ignored. A stopword is a word such as “the” or “some” that is so common that it is considered to have zero semantic value. There is a built-in stopword list, but it can be overridden by a user-defined list. The stopword lists and related configuration options are different for `InnoDB` search indexes and `MyISAM` ones. Stopword processing is controlled by the configuration options `innodb_ft_enable_stopword`, `innodb_ft_server_stopword_table`, and `innodb_ft_user_stopword_table` for `InnoDB` search indexes, and `ft_stopword_file` for `MyISAM` ones.

See Section 12.9.4, “Full-Text Stopwords” to view default stopword lists and how to change them. The default minimum word length can be changed as described in Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”.

Every correct word in the collection and in the query is weighted according to its significance in the collection or query. Thus, a word that is present in many documents has a lower weight, because it has lower semantic value in this particular collection. Conversely, if the word is rare, it receives a higher weight. The weights of the words are combined to compute the relevance of the row. This technique works best with large collections.

MyISAM Limitation

For very small tables, word distribution does not adequately reflect their semantic value, and this model may sometimes produce bizarre results for search indexes on `MyISAM` tables. For example, although the word “MySQL” is present in every row of the `articles` table shown earlier, a search for the word in a `MyISAM` search index produces no results:

```
mysql> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('MySQL' IN NATURAL LANGUAGE MODE);
Empty set (0.00 sec)
```

The search result is empty because the word “MySQL” is present in at least 50% of the rows, and so is effectively treated as a stopword. This filtering technique is more suitable for large data sets, where you might not want the result set to return every second row from a 1GB table, than for small data sets where it might cause poor results for popular terms.

The 50% threshold can surprise you when you first try full-text searching to see how it works, and makes `InnoDB` tables more suited to experimentation with full-text searches. If you create a `MyISAM` table and insert only one or two rows of text into it, every word in the text occurs in at least 50% of the rows. As a result, no search returns any results until the table contains more rows. Users who need to bypass the 50% limitation can build search indexes on `InnoDB` tables, or use the boolean search mode explained in Section 12.9.2, “Boolean Full-Text Searches”.

12.9.2 Boolean Full-Text Searches

MySQL can perform boolean full-text searches using the `IN BOOLEAN MODE` modifier. With this modifier, certain characters have special meaning at the beginning or end of words in the search string. In the following query, the `+` and `-` operators indicate that a word must be present or absent, respectively, for a match to occur. Thus, the query retrieves all the rows that contain the word “MySQL” but that do *not* contain the word “YourSQL”:

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
    AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
```

```
+----+-----------------------+------------------------------------+
| id | title                 | body                               |
+----+-----------------------+------------------------------------+
|  1 | MySQL Tutorial        | DBMS stands for DataBase ...       |
|  2 | How To Use MySQL Well | After you went through a ...       |
|  3 | Optimizing MySQL      | In this tutorial we will show ...  |
|  4 | 1001 MySQL Tricks     | 1. Never run mysqld as root. 2. ... |
|  6 | MySQL Security        | When configured properly, MySQL ... |
+----+-----------------------+------------------------------------+
```

Note

In implementing this feature, MySQL uses what is sometimes referred to as *implied Boolean logic*, in which

- `+` stands for `AND`
- `-` stands for `NOT`
- [*no operator*] implies `OR`

Boolean full-text searches have these characteristics:

- They do not automatically sort rows in order of decreasing relevance.
- `InnoDB` tables require a `FULLTEXT` index on all columns of the `MATCH()` expression to perform boolean queries. Boolean queries against a `MyISAM` search index can work even without a `FULLTEXT` index, although a search executed in this fashion would be quite slow.
- The minimum and maximum word length full-text parameters apply to `FULLTEXT` indexes created using the built-in `FULLTEXT` parser and MeCab parser plugin. `innodb_ft_min_token_size` and `innodb_ft_max_token_size` are used for `InnoDB` search indexes. `ft_min_word_len` and `ft_max_word_len` are used for `MyISAM` search indexes.

  Minimum and maximum word length full-text parameters do not apply to `FULLTEXT` indexes created using the ngram parser. ngram token size is defined by the `ngram_token_size` option.
- The stopword list applies, controlled by `innodb_ft_enable_stopword`, `innodb_ft_server_stopword_table`, and `innodb_ft_user_stopword_table` for `InnoDB` search indexes, and `ft_stopword_file` for `MyISAM` ones.
- `InnoDB` full-text search does not support the use of multiple operators on a single search word, as in this example: `'++apple'`. Use of multiple operators on a single search word returns a syntax error to standard out. MyISAM full-text search will successfully process the same search ignoring all operators except for the operator immediately adjacent to the search word.
- `InnoDB` full-text search only supports leading plus or minus signs. For example, `InnoDB` supports `'+apple'` but does not support `'apple+'`. Specifying a trailing plus or minus sign causes `InnoDB` to report a syntax error.
- `InnoDB` full-text search does not support the use of a leading plus sign with wildcard (`'+*'`), a plus and minus sign combination (`'+-'`), or leading a plus and minus sign combination (`'+-apple'`). These invalid queries return a syntax error.
- `InnoDB` full-text search does not support the use of the `@` symbol in boolean full-text searches. The `@` symbol is reserved for use by the `@distance` proximity search operator.
- They do not use the 50% threshold that applies to `MyISAM` search indexes.

The boolean full-text search capability supports the following operators:

- `+`

  A leading or trailing plus sign indicates that this word *must* be present in each row that is returned. `InnoDB` only supports leading plus signs.

- `-`

  A leading or trailing minus sign indicates that this word must *not* be present in any of the rows that are returned. `InnoDB` only supports leading minus signs.

  Note: The `-` operator acts only to exclude rows that are otherwise matched by other search terms. Thus, a boolean-mode search that contains only terms preceded by `-` returns an empty result. It does not return “all rows except those containing any of the excluded terms.”

- (no operator)

  By default (when neither `+` nor `-` is specified), the word is optional, but the rows that contain it are rated higher. This mimics the behavior of `MATCH() ... AGAINST()` without the `IN BOOLEAN MODE` modifier.

- `@distance`

  This operator works on `InnoDB` tables only. It tests whether two or more words all start within a specified distance from each other, measured in words. Specify the search words within a double-quoted string immediately before the `@distance` operator, for example, `MATCH(col1) AGAINST('"word1 word2 word3" @8' IN BOOLEAN MODE)`

- `> <`

  These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The `>` operator increases the contribution and the `<` operator decreases it. See the example following this list.

- `( )`

  Parentheses group words into subexpressions. Parenthesized groups can be nested.

- `~`

  A leading tilde acts as a negation operator, causing the word's contribution to the row's relevance to be negative. This is useful for marking “noise” words. A row containing such a word is rated lower than others, but is not excluded altogether, as it would be with the `-` operator.

- `*`

  The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it is *appended* to the word to be affected. Words match if they begin with the word preceding the `*` operator.

  If a word is specified with the truncation operator, it is not stripped from a boolean query, even if it is too short or a stopword. Whether a word is too short is determined from the `innodb_ft_min_token_size` setting for `InnoDB` tables, or `ft_min_word_len` for `MyISAM` tables. These options are not applicable to `FULLTEXT` indexes that use the ngram parser.

  The wildcarded word is considered as a prefix that must be present at the start of one or more words. If the minimum word length is 4, a search for `'+word +the*'` could return fewer rows than a search for `'+word +the'`, because the second query ignores the too-short search term `the`.

- `"`

A phrase that is enclosed within double quote (`"`) characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words and performs a search in the `FULLTEXT` index for the words. Nonword characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, `"test phrase"` matches `"test, phrase"`.

If the phrase contains no words that are in the index, the result is empty. The words might not be in the index because of a combination of factors: if they do not exist in the text, are stopwords, or are shorter than the minimum length of indexed words.

The following examples demonstrate some search strings that use boolean full-text operators:

- `'apple banana'`

  Find rows that contain at least one of the two words.

- `'+apple +juice'`

  Find rows that contain both words.

- `'+apple macintosh'`

  Find rows that contain the word “apple”, but rank rows higher if they also contain “macintosh”.

- `'+apple -macintosh'`

  Find rows that contain the word “apple” but not “macintosh”.

- `'+apple ~macintosh'`

  Find rows that contain the word “apple”, but if the row also contains the word “macintosh”, rate it lower than if row does not. This is “softer” than a search for `'+apple -macintosh'`, for which the presence of “macintosh” causes the row not to be returned at all.

- `'+apple +(>turnover <strudel)'`

  Find rows that contain the words “apple” and “turnover”, or “apple” and “strudel” (in any order), but rank “apple turnover” higher than “apple strudel”.

- `'apple*'`

  Find rows that contain words such as “apple”, “apples”, “applesauce”, or “applet”.

- `'"some words"'`

  Find rows that contain the exact phrase “some words” (for example, rows that contain “some words of wisdom” but not “some noise words”). Note that the `"` characters that enclose the phrase are operator characters that delimit the phrase. They are not the quotation marks that enclose the search string itself.

Relevancy Rankings for InnoDB Boolean Mode Search

`InnoDB` full-text search is modeled on the Sphinx full-text search engine, and the algorithms used are based on BM25 and TF-IDF ranking algorithms. For these reasons, relevancy rankings for `InnoDB` boolean full-text search may differ from `MyISAM` relevancy rankings.

`InnoDB` uses a variation of the “term frequency-inverse document frequency” (`TF-IDF`) weighting system to rank a document's relevance for a given full-text search query. The `TF-IDF` weighting is based on how frequently a word appears in a document, offset by how frequently the word appears in all documents in

the collection. In other words, the more frequently a word appears in a document, and the less frequently the word appears in the document collection, the higher the document is ranked.

How Relevancy Ranking is Calculated

The term frequency (`TF`) value is the number of times that a word appears in a document. The inverse document frequency (`IDF`) value of a word is calculated using the following formula, where `total_records` is the number of records in the collection, and `matching_records` is the number of records that the search term appears in.

```
${IDF} = log10( ${total_records} / ${matching_records} )
```

When a document contains a word multiple times, the IDF value is multiplied by the TF value:

```
${TF} * ${IDF}
```

Using the `TF` and `IDF` values, the relevancy ranking for a document is calculated using this formula:

```
${rank} = ${TF} * ${IDF} * ${IDF}
```

The formula is demonstrated in the following examples.

Relevancy Ranking for a Single Word Search

This example demonstrates the relevancy ranking calculation for a single-word search.

```
mysql> CREATE TABLE articles (
id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
title VARCHAR(200),
body TEXT,
FULLTEXT (title,body)
) ENGINE=InnoDB;
Query OK, 0 rows affected (1.04 sec)

mysql> INSERT INTO articles (title,body) VALUES
('MySQL Tutorial','This database tutorial ...'),
("How To Use MySQL",'After you went through a ...'),
('Optimizing Your Database','In this database tutorial ...'),
('MySQL vs. YourSQL','When comparing databases ...'),
('MySQL Security','When configured properly, MySQL ...'),
('Database, Database, Database','database database database'),
('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
('MySQL Full-Text Indexes', 'MySQL fulltext indexes use a ..');
Query OK, 8 rows affected (0.06 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> SELECT id, title, body, MATCH (title,body)  AGAINST ('database' IN BOOLEAN MODE)
AS score FROM articles ORDER BY score DESC;
+----+-----------------------------+-------------------------------------+----------------------+
| id | title                       | body                                | score                |
+----+-----------------------------+-------------------------------------+----------------------+
|  6 | Database, Database, Database | database database database          |  1.0886961221694946 |
|  3 | Optimizing Your Database     | In this database tutorial ...       | 0.36289870738983154 |
|  1 | MySQL Tutorial               | This database tutorial ...          | 0.18144935369491577 |
|  2 | How To Use MySQL             | After you went through a ...        |                   0 |
|  4 | MySQL vs. YourSQL            | When comparing databases ...        |                   0 |
|  5 | MySQL Security               | When configured properly, MySQL ... |                   0 |
|  7 | 1001 MySQL Tricks            | 1. Never run mysqld as root. 2. ... |                   0 |
|  8 | MySQL Full-Text Indexes      | MySQL fulltext indexes use a ..     |                   0 |
+----+-----------------------------+-------------------------------------+----------------------+
8 rows in set (0.00 sec)
```

There are 8 records in total, with 3 that match the “database” search term. The first record (`id 6`) contains the search term 6 times and has a relevancy ranking of `1.0886961221694946`. This ranking value is

calculated using a TF value of 6 (the “database” search term appears 6 times in record id 6) and an IDF value of 0.42596873216370745, which is calculated as follows (where 8 is the total number of records and 3 is the number of records that the search term appears in):

```
${IDF} = log10( 8 / 3 ) = 0.42596873216370745
```

The TF and IDF values are then entered into the ranking formula:

```
${rank} = ${TF} * ${IDF} * ${IDF}
```

Performing the calculation in the MySQL command-line client returns a ranking value of 1.088696164686938.

```
mysql> SELECT 6*log10(8/3)*log10(8/3);
+-------------------------+
| 6*log10(8/3)*log10(8/3) |
+-------------------------+
|       1.088696164686938 |
+-------------------------+
1 row in set (0.00 sec)
```

Note

You may notice a slight difference in the ranking values returned by the SELECT ... MATCH ... AGAINST statement and the MySQL command-line client (1.0886961221694946 versus 1.088696164686938). The difference is due to how the casts between integers and floats/doubles are performed internally by InnoDB (along with related precision and rounding decisions), and how they are performed elsewhere, such as in the MySQL command-line client or other types of calculators.

Relevancy Ranking for a Multiple Word Search

This example demonstrates the relevancy ranking calculation for a multiple-word full-text search based on the articles table and data used in the previous example.

If you search on more than one word, the relevancy ranking value is a sum of the relevancy ranking value for each word, as shown in this formula:

```
${rank} = ${TF} * ${IDF} * ${IDF} + ${TF} * ${IDF} * ${IDF}
```

Performing a search on two terms ('mysql tutorial') returns the following results:

```
mysql> SELECT id, title, body, MATCH (title,body)  AGAINST ('mysql tutorial' IN BOOLEAN MODE)
    AS score FROM articles ORDER BY score DESC;
+----+-----------------------------+-----------------------------------+----------------------+
| id | title                       | body                              | score                |
+----+-----------------------------+-----------------------------------+----------------------+
|  1 | MySQL Tutorial              | This database tutorial ...        |   0.7405621409416199 |
|  3 | Optimizing Your Database    | In this database tutorial ...     |   0.3624762296676636 |
|  5 | MySQL Security              | When configured properly, MySQL ... | 0.031219376251101494 |
|  8 | MySQL Full-Text Indexes     | MySQL fulltext indexes use a ..   | 0.031219376251101494 |
|  2 | How To Use MySQL            | After you went through a ...      | 0.015609688125550747 |
|  4 | MySQL vs. YourSQL           | When comparing databases ...      | 0.015609688125550747 |
|  7 | 1001 MySQL Tricks           | 1. Never run mysqld as root. 2. ... | 0.015609688125550747 |
|  6 | Database, Database, Database | database database database        |                    0 |
+----+-----------------------------+-----------------------------------+----------------------+
8 rows in set (0.00 sec)
```

In the first record (id 8), 'mysql' appears once and 'tutorial' appears twice. There are six matching records for 'mysql' and two matching records for 'tutorial'. The MySQL command-line client returns the expected ranking value when inserting these values into the ranking formula for a multiple word search:

```
mysql> SELECT (1*log10(8/6)*log10(8/6)) + (2*log10(8/2)*log10(8/2));
+-----------------------------------------------------+
| (1*log10(8/6)*log10(8/6)) + (2*log10(8/2)*log10(8/2)) |
+-----------------------------------------------------+
|                                  0.7405621541938003 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

Note

The slight difference in the ranking values returned by the `SELECT ... MATCH ... AGAINST` statement and the MySQL command-line client is explained in the preceding example.

12.9.3 Full-Text Searches with Query Expansion

Full-text search supports query expansion (and in particular, its variant “blind query expansion”). This is generally useful when a search phrase is too short, which often means that the user is relying on implied knowledge that the full-text search engine lacks. For example, a user searching for “database” may really mean that “MySQL”, “Oracle”, “DB2”, and “RDBMS” all are phrases that should match “databases” and should be returned, too. This is implied knowledge.

Blind query expansion (also known as automatic relevance feedback) is enabled by adding `WITH QUERY EXPANSION` or `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` following the search phrase. It works by performing the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. Thus, if one of these documents contains the word “databases” and the word “MySQL”, the second search finds the documents that contain the word “MySQL” even if they do not contain the word “database”. The following example shows this difference:

```
mysql> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' IN NATURAL LANGUAGE MODE);
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
+----+-------------------+------------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' WITH QUERY EXPANSION);
+----+----------------------+------------------------------------------+
| id | title                | body                                     |
+----+----------------------+------------------------------------------+
|  5 | MySQL vs. YourSQL    | In the following database comparison ... |
|  1 | MySQL Tutorial       | DBMS stands for DataBase ...             |
|  3 | Optimizing MySQL     | In this tutorial we will show ...        |
|  6 | MySQL Security       | When configured properly, MySQL ...      |
|  2 | How To Use MySQL Well | After you went through a ...             |
|  4 | 1001 MySQL Tricks    | 1. Never run mysqld as root. 2. ...      |
+----+----------------------+------------------------------------------+
6 rows in set (0.00 sec)
```

Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell “Maigret”. A search for “Megre and the reluctant witnesses” finds only “Maigret and the Reluctant Witnesses” without query expansion. A search with query expansion finds all books with the word “Maigret” on the second pass.

Note

Because blind query expansion tends to increase noise significantly by returning nonrelevant documents, use it only when a search phrase is short.

12.9.4 Full-Text Stopwords

The stopword list is loaded and searched for full-text queries using the server character set and collation (the values of the `character_set_server` and `collation_server` system variables). False hits or misses might occur for stopword lookups if the stopword file or columns used for full-text indexing or searches have a character set or collation different from `character_set_server` or `collation_server`.

Case sensitivity of stopword lookups depends on the server collation. For example, lookups are case insensitive if the collation is `utf8mb4_0900_ai_ci`, whereas lookups are case-sensitive if the collation is `utf8mb4_0900_as_cs` or `utf8mb4_bin`.

- Stopwords for InnoDB Search Indexes
- Stopwords for MyISAM Search Indexes

Stopwords for InnoDB Search Indexes

`InnoDB` has a relatively short list of default stopwords, because documents from technical, literary, and other sources often use short words as keywords or in significant phrases. For example, you might search for “to be or not to be” and expect to get a sensible result, rather than having all those words ignored.

To see the default `InnoDB` stopword list, query the `INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD` table.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD;
+-------+
| value |
+-------+
| a     |
| about |
| an    |
| are   |
| as    |
| at    |
| be    |
| by    |
| com   |
| de    |
| en    |
| for   |
| from  |
| how   |
| i     |
| in    |
| is    |
| it    |
| la    |
| of    |
| on    |
| or    |
| that  |
| the   |
| this  |
| to    |
| was   |
```

```
| what  |
| when  |
| where |
| who   |
| will  |
| with  |
| und   |
| the   |
| www   |
+-------+
36 rows in set (0.00 sec)
```

To define your own stopword list for all `InnoDB` tables, define a table with the same structure as the `INNODB_FT_DEFAULT_STOPWORD` table, populate it with stopwords, and set the value of the `innodb_ft_server_stopword_table` option to a value in the form *`db_name/table_name`* before creating the full-text index. The stopword table must have a single `VARCHAR` column named `value`. The following example demonstrates creating and configuring a new global stopword table for `InnoDB`.

```
-- Create a new stopword table

mysql> CREATE TABLE my_stopwords(value VARCHAR(30)) ENGINE = INNODB;
Query OK, 0 rows affected (0.01 sec)

-- Insert stopwords (for simplicity, a single stopword is used in this example)

mysql> INSERT INTO my_stopwords(value) VALUES ('Ishmael');
Query OK, 1 row affected (0.00 sec)

-- Create the table

mysql> CREATE TABLE opening_lines (
id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
opening_line TEXT(500),
author VARCHAR(200),
title VARCHAR(200)
) ENGINE=InnoDB;
Query OK, 0 rows affected (0.01 sec)

-- Insert data into the table

mysql> INSERT INTO opening_lines(opening_line,author,title) VALUES
('Call me Ishmael.','Herman Melville','Moby-Dick'),
('A screaming comes across the sky.','Thomas Pynchon','Gravity\'s Rainbow'),
('I am an invisible man.','Ralph Ellison','Invisible Man'),
('Where now? Who now? When now?','Samuel Beckett','The Unnamable'),
('It was love at first sight.','Joseph Heller','Catch-22'),
('All this happened, more or less.','Kurt Vonnegut','Slaughterhouse-Five'),
('Mrs. Dalloway said she would buy the flowers herself.','Virginia Woolf','Mrs. Dalloway'),
('It was a pleasure to burn.','Ray Bradbury','Fahrenheit 451');
Query OK, 8 rows affected (0.00 sec)
Records: 8  Duplicates: 0  Warnings: 0

-- Set the innodb_ft_server_stopword_table option to the new stopword table

mysql> SET GLOBAL innodb_ft_server_stopword_table = 'test/my_stopwords';
Query OK, 0 rows affected (0.00 sec)

-- Create the full-text index (which rebuilds the table if no FTS_DOC_ID column is defined)

mysql> CREATE FULLTEXT INDEX idx ON opening_lines(opening_line);
Query OK, 0 rows affected, 1 warning (1.17 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

Verify that the specified stopword ('Ishmael') does not appear by querying the words in `INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE`.

Note

By default, words less than 3 characters in length or greater than 84 characters in length do not appear in an `InnoDB` full-text search index. Maximum and minimum word length values are configurable using the `innodb_ft_max_token_size` and `innodb_ft_min_token_size` variables. This default behavior does not apply to the ngram parser plugin. ngram token size is defined by the `ngram_token_size` option.

```
mysql> SET GLOBAL innodb_ft_aux_table='test/opening_lines';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT word FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE LIMIT 15;
+-----------+
| word      |
+-----------+
| across    |
| all       |
| burn      |
| buy       |
| call      |
| comes     |
| dalloway  |
| first     |
| flowers   |
| happened  |
| herself   |
| invisible |
| less      |
| love      |
| man       |
+-----------+
15 rows in set (0.00 sec)
```

To create stopword lists on a table-by-table basis, create other stopword tables and use the `innodb_ft_user_stopword_table` option to specify the stopword table that you want to use before you create the full-text index.

Stopwords for MyISAM Search Indexes

The stopword file is loaded and searched using `latin1` if `character_set_server` is `ucs2`, `utf16`, `utf16le`, or `utf32`.

To override the default stopword list for MyISAM tables, set the `ft_stopword_file` system variable. (See Section 5.1.8, “Server System Variables”.) The variable value should be the path name of the file containing the stopword list, or the empty string to disable stopword filtering. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. After changing the value of this variable or the contents of the stopword file, restart the server and rebuild your `FULLTEXT` indexes.

The stopword list is free-form, separating stopwords with any nonalphanumeric character such as newline, space, or comma. Exceptions are the underscore character (`_`) and a single apostrophe (`'`) which are treated as part of a word. The character set of the stopword list is the server's default character set; see Section 10.3.2, “Server Character Set and Collation”.

The following list shows the default stopwords for `MyISAM` search indexes. In a MySQL source distribution, you can find this list in the `storage/myisam/ft_static.c` file.

```
a's           able         about        above        according
accordingly   across       actually     after        afterwards
again         against      ain't        all          allow
```

allows	almost	alone	along	already
also	although	always	am	among
amongst	an	and	another	any
anybody	anyhow	anyone	anything	anyway
anyways	anywhere	apart	appear	appreciate
appropriate	are	aren't	around	as
aside	ask	asking	associated	at
available	away	awfully	be	became
because	become	becomes	becoming	been
before	beforehand	behind	being	believe
below	beside	besides	best	better
between	beyond	both	brief	but
by	c'mon	c's	came	can
can't	cannot	cant	cause	causes
certain	certainly	changes	clearly	co
com	come	comes	concerning	consequently
consider	considering	contain	containing	contains
corresponding	could	couldn't	course	currently
definitely	described	despite	did	didn't
different	do	does	doesn't	doing
don't	done	down	downwards	during
each	edu	eg	eight	either
else	elsewhere	enough	entirely	especially
et	etc	even	ever	every
everybody	everyone	everything	everywhere	ex
exactly	example	except	far	few
fifth	first	five	followed	following
follows	for	former	formerly	forth
four	from	further	furthermore	get
gets	getting	given	gives	go
goes	going	gone	got	gotten
greetings	had	hadn't	happens	hardly
has	hasn't	have	haven't	having
he	he's	hello	help	hence
her	here	here's	hereafter	hereby
herein	hereupon	hers	herself	hi
him	himself	his	hither	hopefully
how	howbeit	however	i'd	i'll
i'm	i've	ie	if	ignored
immediate	in	inasmuch	inc	indeed
indicate	indicated	indicates	inner	insofar
instead	into	inward	is	isn't
it	it'd	it'll	it's	its
itself	just	keep	keeps	kept
know	known	knows	last	lately
later	latter	latterly	least	less
lest	let	let's	like	liked
likely	little	look	looking	looks
ltd	mainly	many	may	maybe
me	mean	meanwhile	merely	might
more	moreover	most	mostly	much
must	my	myself	name	namely
nd	near	nearly	necessary	need
needs	neither	never	nevertheless	new
next	nine	no	nobody	non
none	noone	nor	normally	not
nothing	novel	now	nowhere	obviously
of	off	often	oh	ok
okay	old	on	once	one
ones	only	onto	or	other
others	otherwise	ought	our	ours
ourselves	out	outside	over	overall
own	particular	particularly	per	perhaps
placed	please	plus	possible	presumably
probably	provides	que	quite	qv
rather	rd	re	really	reasonably
regarding	regardless	regards	relatively	respectively

```
right         said          same          saw           say
saying        says          second        secondly      see
seeing        seem          seemed        seeming       seems
seen          self          selves        sensible      sent
serious       seriously     seven         several       shall
she           should        shouldn't     since         six
so            some          somebody      somehow       someone
something     sometime      sometimes     somewhat      somewhere
soon          sorry         specified     specify       specifying
still         sub           such          sup           sure
t's           take          taken         tell          tends
th            than          thank         thanks        thanx
that          that's        thats         the           their
theirs        them          themselves    then          thence
there         there's       thereafter    thereby       therefore
therein       theres        thereupon     these         they
they'd        they'll       they're       they've       think
third         this          thorough      thoroughly    those
though        three         through       throughout    thru
thus          to            together      too           took
toward        towards       tried         tries         truly
try           trying        twice         two           un
under         unfortunately unless        unlikely      until
unto          up            upon          us            use
used          useful        uses          using         usually
value         various       very          via           viz
vs            want          wants         was           wasn't
way           we            we'd          we'll         we're
we've         welcome       well          went          were
weren't       what          what's        whatever      when
whence        whenever      where         where's       whereafter
whereas       whereby       wherein       whereupon     wherever
whether       which         while         whither       who
who's         whoever       whole         whom          whose
why           will          willing       wish          with
within        without       won't         wonder        would
wouldn't      yes           yet           you           you'd
you'll        you're        you've        your          yours
yourself      yourselves    zero
```

12.9.5 Full-Text Restrictions

- Full-text searches are supported for `InnoDB` and `MyISAM` tables only.
- Full-text searches are not supported for partitioned tables. See Section 23.6, “Restrictions and Limitations on Partitioning”.
- Full-text searches can be used with most multibyte character sets. The exception is that for Unicode, the `utf8` character set can be used, but not the `ucs2` character set. Although `FULLTEXT` indexes on `ucs2` columns cannot be used, you can perform `IN BOOLEAN MODE` searches on a `ucs2` column that has no such index.

  The remarks for `utf8` also apply to `utf8mb4`, and the remarks for `ucs2` also apply to `utf16`, `utf16le`, and `utf32`.
- Ideographic languages such as Chinese and Japanese do not have word delimiters. Therefore, the built-in full-text parser *cannot determine where words begin and end in these and other such languages*.

  A character-based ngram full-text parser that supports Chinese, Japanese, and Korean (CJK), and a word-based MeCab parser plugin that supports Japanese are provided for use with `InnoDB` and `MyISAM` tables.

- Although the use of multiple character sets within a single table is supported, all columns in a `FULLTEXT` index must use the same character set and collation.
- The `MATCH()` column list must match exactly the column list in some `FULLTEXT` index definition for the table, unless this `MATCH()` is `IN BOOLEAN MODE` on a `MyISAM` table. For `MyISAM` tables, boolean-mode searches can be done on nonindexed columns, although they are likely to be slow.
- The argument to `AGAINST()` must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.
- Index hints are more limited for `FULLTEXT` searches than for non-`FULLTEXT` searches. See Section 8.9.4, “Index Hints”.
- For `InnoDB`, all DML operations (`INSERT`, `UPDATE`, `DELETE`) involving columns with full-text indexes are processed at transaction commit time. For example, for an `INSERT` operation, an inserted string is tokenized and decomposed into individual words. The individual words are then added to full-text index tables when the transaction is committed. As a result, full-text searches only return committed data.
- The '%' character is not a supported wildcard character for full-text searches.

12.9.6 Fine-Tuning MySQL Full-Text Search

MySQL's full-text search capability has few user-tunable parameters. You can exert more control over full-text searching behavior if you have a MySQL source distribution because some changes require source code modifications. See Section 2.9, “Installing MySQL from Source”.

Full-text search is carefully tuned for effectiveness. Modifying the default behavior in most cases can actually decrease effectiveness. *Do not alter the MySQL sources unless you know what you are doing*.

Most full-text variables described in this section must be set at server startup time. A server restart is required to change them; they cannot be modified while the server is running.

Some variable changes require that you rebuild the `FULLTEXT` indexes in your tables. Instructions for doing so are given later in this section.

- Configuring Minimum and Maximum Word Length
- Configuring the Natural Language Search Threshold
- Modifying Boolean Full-Text Search Operators
- Character Set Modifications
- Rebuilding InnoDB Full-Text Indexes
- Optimizing InnoDB Full-Text Indexes
- Rebuilding MyISAM Full-Text Indexes

Configuring Minimum and Maximum Word Length

The minimum and maximum lengths of words to be indexed are defined by the `innodb_ft_min_token_size` and `innodb_ft_max_token_size` for `InnoDB` search indexes, and `ft_min_word_len` and `ft_max_word_len` for `MyISAM` ones.

Note

Minimum and maximum word length full-text parameters do not apply to `FULLTEXT` indexes created using the ngram parser. ngram token size is defined by the `ngram_token_size` option.

After changing any of these options, rebuild your FULLTEXT indexes for the change to take effect. For example, to make two-character words searchable, you could put the following lines in an option file:

```
[mysqld]
innodb_ft_min_token_size=2
ft_min_word_len=2
```

Then restart the server and rebuild your FULLTEXT indexes. For MyISAM tables, note the remarks regarding myisamchk in the instructions that follow for rebuilding MyISAM full-text indexes.

Configuring the Natural Language Search Threshold

For MyISAM search indexes, the 50% threshold for natural language searches is determined by the particular weighting scheme chosen. To disable it, look for the following line in storage/myisam/ftdefs.h:

```
#define GWS_IN_USE GWS_PROB
```

Change that line to this:

```
#define GWS_IN_USE GWS_FREQ
```

Then recompile MySQL. There is no need to rebuild the indexes in this case.

Note

By making this change, you *severely* decrease MySQL's ability to provide adequate relevance values for the MATCH() function. If you really need to search for such common words, it would be better to search using IN BOOLEAN MODE instead, which does not observe the 50% threshold.

Modifying Boolean Full-Text Search Operators

To change the operators used for boolean full-text searches on MyISAM tables, set the ft_boolean_syntax system variable. (InnoDB does not have an equivalent setting.) This variable can be changed while the server is running, but you must have privileges sufficient to set global system variables (see Section 5.1.9.1, “System Variable Privileges”). No rebuilding of indexes is necessary in this case.

Character Set Modifications

For the built-in full-text parser, you can change the set of characters that are considered word characters in several ways, as described in the following list. After making the modification, rebuild the indexes for each table that contains any FULLTEXT indexes. Suppose that you want to treat the hyphen character ('-') as a word character. Use one of these methods:

- Modify the MySQL source: In storage/innobase/handler/ha_innodb.cc (for InnoDB), or in storage/myisam/ftdefs.h (for MyISAM), see the true_word_char() and misc_word_char() macros. Add '-' to one of those macros and recompile MySQL.
- Modify a character set file: This requires no recompilation. The true_word_char() macro uses a “character type” table to distinguish letters and numbers from other characters. . You can edit the contents of the <ctype><map> array in one of the character set XML files to specify that '-' is a “letter.” Then use the given character set for your FULLTEXT indexes. For information about the <ctype><map> array format, see Section 10.13.1, “Character Definition Arrays”.

- Add a new collation for the character set used by the indexed columns, and alter the columns to use that collation. For general information about adding collations, see Section 10.14, “Adding a Collation to a Character Set”. For an example specific to full-text indexing, see Section 12.9.7, “Adding a Collation for Full-Text Indexing”.

Rebuilding InnoDB Full-Text Indexes

For the changes to take effect, `FULLTEXT` indexes must be rebuilt after modifying any of the following full-text index variables: `innodb_ft_min_token_size`; `innodb_ft_max_token_size`; `innodb_ft_server_stopword_table`; `innodb_ft_user_stopword_table`; `innodb_ft_enable_stopword`; `ngram_token_size`. Modifying `innodb_ft_min_token_size`, `innodb_ft_max_token_size`, or `ngram_token_size` requires restarting the server.

To rebuild `FULLTEXT` indexes for an `InnoDB` table, use `ALTER TABLE` with the `DROP INDEX` and `ADD INDEX` options to drop and re-create each index.

Optimizing InnoDB Full-Text Indexes

Running `OPTIMIZE TABLE` on a table with a full-text index rebuilds the full-text index, removing deleted Document IDs and consolidating multiple entries for the same word, where possible.

To optimize a full-text index, enable `innodb_optimize_fulltext_only` and run `OPTIMIZE TABLE`.

```
mysql> set GLOBAL innodb_optimize_fulltext_only=ON;
Query OK, 0 rows affected (0.01 sec)

mysql> OPTIMIZE TABLE opening_lines;
+--------------------+----------+----------+----------+
| Table              | Op       | Msg_type | Msg_text |
+--------------------+----------+----------+----------+
| test.opening_lines | optimize | status   | OK       |
+--------------------+----------+----------+----------+
1 row in set (0.01 sec)
```

To avoid lengthy rebuild times for full-text indexes on large tables, you can use the `innodb_ft_num_word_optimize` option to perform the optimization in stages. The `innodb_ft_num_word_optimize` option defines the number of words that are optimized each time `OPTIMIZE TABLE` is run. The default setting is 2000, which means that 2000 words are optimized each time `OPTIMIZE TABLE` is run. Subsequent `OPTIMIZE TABLE` operations continue from where the preceding `OPTIMIZE TABLE` operation ended.

Rebuilding MyISAM Full-Text Indexes

If you modify full-text variables that affect indexing (`ft_min_word_len`, `ft_max_word_len`, or `ft_stopword_file`), or if you change the stopword file itself, you must rebuild your `FULLTEXT` indexes after making the changes and restarting the server.

To rebuild the `FULLTEXT` indexes for a `MyISAM` table, it is sufficient to do a `QUICK` repair operation:

```
mysql> REPAIR TABLE tbl_name QUICK;
```

Alternatively, use `ALTER TABLE` as just described. In some cases, this may be faster than a repair operation.

Each table that contains any `FULLTEXT` index must be repaired as just shown. Otherwise, queries for the table may yield incorrect results, and modifications to the table will cause the server to see the table as corrupt and in need of repair.

If you use `myisamchk` to perform an operation that modifies `MyISAM` table indexes (such as repair or analyze), the `FULLTEXT` indexes are rebuilt using the *default* full-text parameter values for minimum word length, maximum word length, and stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in `MyISAM` index files. To avoid the problem if you have modified the minimum or maximum word length or stopword file values used by the server, specify the same `ft_min_word_len`, `ft_max_word_len`, and `ft_stopword_file` values for `myisamchk` that you use for `mysqld`. For example, if you have set the minimum word length to 3, you can repair a table with `myisamchk` like this:

```
myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that `myisamchk` and the server use the same values for full-text parameters, place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using `myisamchk` for `MyISAM` table index modification is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE` statements. These statements are performed by the server, which knows the proper full-text parameter values to use.

12.9.7 Adding a Collation for Full-Text Indexing

This section describes how to add a new collation for full-text searches using the built-in full-text parser. The sample collation is like `latin1_swedish_ci` but treats the `'-'` character as a letter rather than as a punctuation character so that it can be indexed as a word character. General information about adding collations is given in Section 10.14, “Adding a Collation to a Character Set”; it is assumed that you have read it and are familiar with the files involved.

To add a collation for full-text indexing, use the following procedure. The instructions here add a collation for a simple character set, which as discussed in Section 10.14, “Adding a Collation to a Character Set”, can be created using a configuration file that describes the character set properties. For a complex character set such as Unicode, create collations using C source files that describe the character set properties.

1. Add a collation to the `Index.xml` file. The collation ID must be unused, so choose a value different from 1000 if that ID is already taken on your system.

   ```
   <charset name="latin1">
   ...
   <collation name="latin1_fulltext_ci" id="1000"/>
   </charset>
   ```

2. Declare the sort order for the collation in the `latin1.xml` file. In this case, the order can be copied from `latin1_swedish_ci`:

   ```
   <collation name="latin1_fulltext_ci">
   <map>
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
   10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
   ```

```
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
41 41 41 41 5C 5B 5C 43 45 45 45 45 49 49 49 49
44 4E 4F 4F 4F 4F 5D D7 D8 55 55 55 59 59 DE DF
41 41 41 41 5C 5B 5C 43 45 45 45 45 49 49 49 49
44 4E 4F 4F 4F 4F 5D F7 D8 55 55 55 59 59 DE FF
</map>
</collation>
```

3. Modify the `ctype` array in `latin1.xml`. Change the value corresponding to 0x2D (which is the code for the `'-'` character) from 10 (punctuation) to 01 (uppercase letter). In the following array, this is the element in the fourth row down, third value from the end.

```
<ctype>
<map>
00
20 20 20 20 20 20 20 20 20 28 28 28 28 28 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
48 10 10 10 10 10 10 10 10 10 10 10 01 10 10 10
84 84 84 84 84 84 84 84 84 10 10 10 10 10 10 10
10 81 81 81 81 81 81 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 01 10 10 10 10 10
10 82 82 82 82 82 82 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 02 02 02 02 10 10 10 10 20
10 00 10 02 10 10 10 10 10 10 01 10 01 00 01 00
00 10 10 10 10 10 10 10 10 10 02 10 02 00 02 01
48 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 10 01 01 01 01 01 01 01 02
02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 10 02 02 02 02 02 02 02 02
</map>
</ctype>
```

4. Restart the server.

5. To employ the new collation, include it in the definition of columns that are to use it:

```
mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected (0.13 sec)

mysql> CREATE TABLE t1 (
    a TEXT CHARACTER SET latin1 COLLATE latin1_fulltext_ci,
    FULLTEXT INDEX(a)
    ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.47 sec)
```

6. Test the collation to verify that hyphen is considered as a word character:

```
mysql> INSERT INTO t1 VALUEs ('----'),('....'),('abcd');
Query OK, 3 rows affected (0.22 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t1 WHERE MATCH a AGAINST ('----' IN BOOLEAN MODE);
```

```
+------+
| a    |
+------+
| ---- |
+------+
1 row in set (0.00 sec)
```

12.9.8 ngram Full-Text Parser

The built-in MySQL full-text parser uses the white space between words as a delimiter to determine where words begin and end, which is a limitation when working with ideographic languages that do not use word delimiters. To address this limitation, MySQL provides an ngram full-text parser that supports Chinese, Japanese, and Korean (CJK). The ngram full-text parser is supported for use with `InnoDB` and `MyISAM`.

Note

MySQL also provides a MeCab full-text parser plugin for Japanese, which tokenizes documents into meaningful words. For more information, see Section 12.9.9, “MeCab Full-Text Parser Plugin”.

An ngram is a contiguous sequence of *n* characters from a given sequence of text. The ngram parser tokenizes a sequence of text into a contiguous sequence of *n* characters. For example, you can tokenize “abcd” for different values of *n* using the ngram full-text parser.

```
n=1: 'a', 'b', 'c', 'd'
n=2: 'ab', 'bc', 'cd'
n=3: 'abc', 'bcd'
n=4: 'abcd'
```

The ngram full-text parser is a built-in server plugin. As with other built-in server plugins, it is automatically loaded when the server is started.

The full-text search syntax described in Section 12.9, “Full-Text Search Functions” applies to the ngram parser plugin. Differences in parsing behavior are described in this section. Full-text-related configuration options, except for minimum and maximum word length options (`innodb_ft_min_token_size`, `innodb_ft_max_token_size`, `ft_min_word_len`, `ft_max_word_len`) are also applicable.

Configuring ngram Token Size

The ngram parser has a default ngram token size of 2 (bigram). For example, with a token size of 2, the ngram parser parses the string “abc def” into four tokens: “ab”, “bc”, “de” and “ef”.

ngram token size is configurable using the `ngram_token_size` configuration option, which has a minimum value of 1 and maximum value of 10.

Typically, `ngram_token_size` is set to the size of the largest token that you want to search for. If you only intend to search for single characters, set `ngram_token_size` to 1. A smaller token size produces a smaller full-text search index, and faster searches. If you need to search for words comprised of more than one character, set `ngram_token_size` accordingly. For example, “Happy Birthday” is “生日快乐” in simplified Chinese, where “生日” is “birthday”, and “快乐” translates as “happy”. To search on two-character words such as these, set `ngram_token_size` to a value of 2 or higher.

As a read-only variable, `ngram_token_size` may only be set as part of a startup string or in a configuration file:

- Startup string:

```
mysqld --ngram_token_size=2
```

- Configuration file:

```
[mysqld]
ngram_token_size=2
```

Note

The following minimum and maximum word length configuration options are ignored for `FULLTEXT` indexes that use the ngram parser: `innodb_ft_min_token_size`, `innodb_ft_max_token_size`, `ft_min_word_len`, and `ft_max_word_len`.

Creating a FULLTEXT Index that Uses the ngram Parser

To create a `FULLTEXT` index that uses the ngram parser, specify `WITH PARSER ngram` with `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX`.

The following example demonstrates creating a table with an `ngram FULLTEXT` index, inserting sample data (Simplified Chinese text), and viewing tokenized data in the `INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE` table.

```
mysql> USE test;

mysql> CREATE TABLE articles (
      id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
      title VARCHAR(200),
      body TEXT,
      FULLTEXT (title,body) WITH PARSER ngram
    ) ENGINE=InnoDB CHARACTER SET utf8mb4;

mysql> SET NAMES utf8mb4;

INSERT INTO articles (title,body) VALUES
    ('数据库管理','在本教程中我将向你展示如何管理数据库'),
    ('数据库应用开发','学习开发数据库应用程序');

mysql> SET GLOBAL innodb_ft_aux_table="test/articles";

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE ORDER BY doc_id, position;
```

To add a `FULLTEXT` index to an existing table, you can use `ALTER TABLE` or `CREATE INDEX`. For example:

```
CREATE TABLE articles (
      id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
      title VARCHAR(200),
      body TEXT
     ) ENGINE=InnoDB CHARACTER SET utf8;

ALTER TABLE articles ADD FULLTEXT INDEX ft_index (title,body) WITH PARSER ngram;

# Or:

CREATE FULLTEXT INDEX ft_index ON articles (title,body) WITH PARSER ngram;
```

ngram Parser Space Handling

The ngram parser eliminates spaces when parsing. For example:

- “ab cd” is parsed to “ab”, “cd”

- “a bc” is parsed to “bc”

ngram Parser Stopword Handling

The built-in MySQL full-text parser compares words to entries in the stopword list. If a word is equal to an entry in the stopword list, the word is excluded from the index. For the ngram parser, stopword handling is performed differently. Instead of excluding tokens that are equal to entries in the stopword list, the ngram parser excludes tokens that *contain* stopwords. For example, assuming `ngram_token_size=2`, a document that contains “a,b” is parsed to “a,” and “,b”. If a comma (“,”) is defined as a stopword, both “a,” and “,b” are excluded from the index because they contain a comma.

By default, the ngram parser uses the default stopword list, which contains a list of English stopwords. For a stopword list applicable to Chinese, Japanese, or Korean, you must create your own. For information about creating a stopword list, see Section 12.9.4, “Full-Text Stopwords”.

Stopwords greater in length than `ngram_token_size` are ignored.

ngram Parser Term Search

For *natural language mode* search, the search term is converted to a union of ngram terms. For example, the string “abc” (assuming `ngram_token_size=2`) is converted to “ab bc”. Given two documents, one containing “ab” and the other containing “abc”, the search term “ab bc” matches both documents.

For *boolean mode search*, the search term is converted to an ngram phrase search. For example, the string 'abc' (assuming `ngram_token_size=2`) is converted to '“ab bc”'. Given two documents, one containing 'ab' and the other containing 'abc', the search phrase '“ab bc”' only matches the document containing 'abc'.

ngram Parser Wildcard Search

Because an ngram `FULLTEXT` index contains only ngrams, and does not contain information about the beginning of terms, wildcard searches may return unexpected results. The following behaviors apply to wildcard searches using ngram `FULLTEXT` search indexes:

- If the prefix term of a wildcard search is shorter than ngram token size, the query returns all indexed rows that contain ngram tokens starting with the prefix term. For example, assuming `ngram_token_size=2`, a search on “a*” returns all rows starting with “a”.
- If the prefix term of a wildcard search is longer than ngram token size, the prefix term is converted to an ngram phrase and the wildcard operator is ignored. For example, assuming `ngram_token_size=2`, an “abc*” wildcard search is converted to “ab bc”.

ngram Parser Phrase Search

Phrase searches are converted to ngram phrase searches. For example, The search phrase “abc” is converted to “ab bc”, which returns documents containing “abc” and “ab bc”.

The search phrase “abc def” is converted to “ab bc de ef”, which returns documents containing “abc def” and “ab bc de ef”. A document that contains “abcdef” is not returned.

12.9.9 MeCab Full-Text Parser Plugin

The built-in MySQL full-text parser uses the white space between words as a delimiter to determine where words begin and end, which is a limitation when working with ideographic languages that do not use word

delimiters. To address this limitation for Japanese, MySQL provides a MeCab full-text parser plugin. The MeCab full-text parser plugin is supported for use with `InnoDB` and `MyISAM`.

Note

MySQL also provides an ngram full-text parser plugin that supports Japanese. For more information, see Section 12.9.8, “ngram Full-Text Parser”.

The MeCab full-text parser plugin is a full-text parser plugin for Japanese that tokenizes a sequence of text into meaningful words. For example, MeCab tokenizes “データベース管理” (“Database Management”) into “データベース” (“Database”) and “管理” (“Management”). By comparison, the ngram full-text parser tokenizes text into a contiguous sequence of *n* characters, where *n* represents a number between 1 and 10.

In addition to tokenizing text into meaningful words, MeCab indexes are typically smaller than ngram indexes, and MeCab full-text searches are generally faster. One drawback is that it may take longer for the MeCab full-text parser to tokenize documents, compared to the ngram full-text parser.

The full-text search syntax described in Section 12.9, “Full-Text Search Functions” applies to the MeCab parser plugin. Differences in parsing behavior are described in this section. Full-text related configuration options are also applicable.

For additional information about the MeCab parser, refer to the MeCab: Yet Another Part-of-Speech and Morphological Analyzer project on Github.

Installing the MeCab Parser Plugin

The MeCab parser plugin requires `mecab` and `mecab-ipadic`.

On supported Fedora, Debian and Ubuntu platforms (except Ubuntu 12.04 where the system `mecab` version is too old), MySQL dynamically links to the system `mecab` installation if it is installed to the default location. On other supported Unix-like platforms, `libmecab.so` is statically linked in `libpluginmecab.so`, which is located in the MySQL plugin directory. `mecab-ipadic` is included in MySQL binaries and is located in *`MYSQL_HOME`*`\lib\mecab`.

You can install `mecab` and `mecab-ipadic` using a native package management utility (on Fedora, Debian, and Ubuntu), or you can build `mecab` and `mecab-ipadic` from source. For information about installing `mecab` and `mecab-ipadic` using a native package management utility, see Installing MeCab From a Binary Distribution (Optional). If you want to build `mecab` and `mecab-ipadic` from source, see Building MeCab From Source (Optional).

On Windows, `libmecab.dll` is found in the MySQL `bin` directory. `mecab-ipadic` is located in *`MYSQL_HOME`*`/lib/mecab`.

To install and configure the MeCab parser plugin, perform the following steps:

1. In the MySQL configuration file, set the `mecab_rc_file` configuration option to the location of the `mecabrc` configuration file, which is the configuration file for MeCab. If you are using the MeCab package distributed with MySQL, the `mecabrc` file is located in `MYSQL_HOME/lib/mecab/etc/`.

   ```
   [mysqld]
   loose-mecab-rc-file=MYSQL_HOME/lib/mecab/etc/mecabrc
   ```

   The `loose` prefix is an option modifier. The `mecab_rc_file` option is not recognized by MySQL until the MeCaB parser plugin is installed but it must be set before attempting to install the MeCaB parser plugin. The `loose` prefix allows you restart MySQL without encountering an error due to an unrecognized variable.

If you use your own MeCab installation, or build MeCab from source, the location of the `mecabrc` configuration file may differ.

For information about the MySQL configuration file and its location, see Section 4.2.2.2, “Using Option Files”.

2. Also in the MySQL configuration file, set the minimum token size to 1 or 2, which are the values recommended for use with the MeCab parser. For `InnoDB` tables, minimum token size is defined by the `innodb_ft_min_token_size` configuration option, which has a default value of 3. For `MyISAM` tables, minimum token size is defined by `ft_min_word_len`, which has a default value of 4.

```
[mysqld]
innodb_ft_min_token_size=1
```

3. Modify the `mecabrc` configuration file to specify the dictionary you want to use. The `mecab-ipadic` package distributed with MySQL binaries includes three dictionaries (`ipadic_euc-jp`, `ipadic_sjis`, and `ipadic_utf-8`). The `mecabrc` configuration file packaged with MySQL contains and entry similar to the following:

```
dicdir =  /path/to/mysql/lib/mecab/lib/mecab/dic/ipadic_euc-jp
```

To use the `ipadic_utf-8` dictionary, for example, modify the entry as follows:

```
dicdir=MYSQL_HOME/lib/mecab/dic/ipadic_utf-8
```

If you are using your own MeCab installation or have built MeCab from source, the default `dicdir` entry in the `mecabrc` file will differ, as will the dictionaries and their location.

Note

After the MeCab parser plugin is installed, you can use the `mecab_charset` status variable to view the character set used with MeCab. The three MeCab dictionaries provided with the MySQL binary support the following character sets.

- The `ipadic_euc-jp` dictionary supports the `ujis` and `eucjpms` character sets.
- The `ipadic_sjis` dictionary supports the `sjis` and `cp932` character sets.
- The `ipadic_utf-8` dictionary supports the `utf8` and `utf8mb4` character sets.

`mecab_charset` only reports the first supported character set. For example, the `ipadic_utf-8` dictionary supports both `utf8` and `utf8mb4`. `mecab_charset` always reports `utf8` when this dictionary is in use.

4. Restart MySQL.

5. Install the MeCab parser plugin:

The MeCab parser plugin is installed using `INSTALL PLUGIN` syntax. The plugin name is `mecab`, and the shared library name is `libpluginmecab.so`. For additional information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

```
INSTALL PLUGIN mecab SONAME 'libpluginmecab.so';
```

Once installed, the MeCab parser plugin loads at every normal MySQL restart.

6. Verify that the MeCab parser plugin is loaded using the `SHOW PLUGINS` statement.

```
mysql> SHOW PLUGINS;
```

A `mecab` plugin should appear in the list of plugins.

Creating a FULLTEXT Index that uses the MeCab Parser

To create a `FULLTEXT` index that uses the mecab parser, specify `WITH PARSER ngram` with `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX`.

This example demonstrates creating a table with a `mecab` `FULLTEXT` index, inserting sample data, and viewing tokenized data in the `INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE` table:

```
mysql> USE test;

mysql> CREATE TABLE articles (
      id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
      title VARCHAR(200),
      body TEXT,
      FULLTEXT (title,body) WITH PARSER mecab
    ) ENGINE=InnoDB CHARACTER SET utf8;

mysql> SET NAMES utf8;

mysql> INSERT INTO articles (title,body) VALUES
    ('データベース管理','このチュートリアルでは、私はどのようにデータベースを管理する方法を紹介します'),
    ('データベースアプリケーション開発','データベースアプリケーションを開発することを学ぶ');

mysql> SET GLOBAL innodb_ft_aux_table="test/articles";

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE ORDER BY doc_id, position;
```

To add a `FULLTEXT` index to an existing table, you can use `ALTER TABLE` or `CREATE INDEX`. For example:

```
CREATE TABLE articles (
      id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
      title VARCHAR(200),
      body TEXT
     ) ENGINE=InnoDB CHARACTER SET utf8;

ALTER TABLE articles ADD FULLTEXT INDEX ft_index (title,body) WITH PARSER mecab;

# Or:

CREATE FULLTEXT INDEX ft_index ON articles (title,body) WITH PARSER mecab;
```

MeCab Parser Space Handling

The MeCab parser uses spaces as separators in query strings. For example, the MeCab parser tokenizes データベース管理 as データベース and 管理.

MeCab Parser Stopword Handling

By default, the MeCab parser uses the default stopword list, which contains a short list of English stopwords. For a stopword list applicable to Japanese, you must create your own. For information about creating stopword lists, see Section 12.9.4, “Full-Text Stopwords”.

MeCab Parser Term Search

For natural language mode search, the search term is converted to a union of tokens. For example, データベース管理 is converted to データベース 管理.

```
SELECT COUNT(*) FROM articles WHERE MATCH(title,body) AGAINST('データベース管理' IN NATURAL LANGUAGE MODE);
```

For boolean mode search, the search term is converted to a search phrase. For example, データベース管理 is converted to データベース 管理.

```
SELECT COUNT(*) FROM articles WHERE MATCH(title,body) AGAINST('データベース管理' IN BOOLEAN MODE);
```

MeCab Parser Wildcard Search

Wildcard search terms are not tokenized. A search on データベース管理* is performed on the prefix, データベース管理.

```
SELECT COUNT(*) FROM articles WHERE MATCH(title,body) AGAINST('データベース*' IN BOOLEAN MODE);
```

MeCab Parser Phrase Search

Phrases are tokenized. For example, データベース管理 is tokenized as データベース 管理.

```
SELECT COUNT(*) FROM articles WHERE MATCH(title,body) AGAINST('"データベース管理"' IN BOOLEAN MODE);
```

Installing MeCab From a Binary Distribution (Optional)

This section describes how to install `mecab` and `mecab-ipadic` from a binary distribution using a native package management utility. For example, on Fedora, you can use Yum to perform the installation:

```
yum mecab-devel
```

On Debian or Ubuntu, you can perform an APT installation:

```
apt-get install mecab
apt-get install mecab-ipadic
```

Installing MeCab From Source (Optional)

If you want to build `mecab` and `mecab-ipadic` from source, basic installation steps are provided below. For additional information, refer to the MeCab documentation.

1. Download the tar.gz packages for `mecab` and `mecab-ipadic` from http://taku910.github.io/mecab/#download. As of February, 2016, the latest available packages are `mecab-0.996.tar.gz` and `mecab-ipadic-2.7.0-20070801.tar.gz`.

2. Install `mecab`:

   ```
   tar zxfv mecab-0.996.tar
   cd mecab-0.996
   ./configure
   make
   make check
   su
   make install
   ```

3. Install `mecab-ipadic`:

   ```
   tar zxfv mecab-ipadic-2.7.0-20070801.tar
   cd mecab-ipadic-2.7.0-20070801
   ./configure
   make
   su
   make install
   ```

4. Compile MySQL using the `WITH_MECAB` CMake option. Set the `WITH_MECAB` option to `system` if you have installed `mecab` and `mecab-ipadic` to the default location.

```
-DWITH_MECAB=system
```

If you defined a custom installation directory, set `WITH_MECAB` to the custom directory. For example:

```
-DWITH_MECAB=/path/to/mecab
```

12.10 Cast Functions and Operators

Table 12.14 Cast Functions and Operators

Name	Description
`BINARY`	Cast a string to a binary string
`CAST()`	Cast a value as a certain type
`CONVERT()`	Cast a value as a certain type

Cast functions and operators enable conversion of values from one data type to another.

`CONVERT()` with a `USING` clause provides a way to convert data between different character sets:

```
CONVERT(expr USING transcoding_name)
```

In MySQL, transcoding names are the same as the corresponding character set names.

Examples:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8_table (utf8_column)
    SELECT CONVERT(latin1_column USING utf8) FROM latin1_table;
```

You can also use `CONVERT()` without `USING` or `CAST()` to convert strings between different character sets:

```
CONVERT(string, CHAR[(N)] CHARACTER SET charset_name)
CAST(string AS CHAR[(N)] CHARACTER SET charset_name)
```

Examples:

```
SELECT CONVERT('test', CHAR CHARACTER SET utf8);
SELECT CAST('test' AS CHAR CHARACTER SET utf8);
```

If you specify `CHARACTER SET charset_name` as just shown, the resulting character set and collation are *`charset_name`* and the default collation of *`charset_name`*. If you omit `CHARACTER SET charset_name`, the resulting character set and collation are defined by the `character_set_connection` and `collation_connection` system variables that determine the default connection character set and collation (see Section 10.4, “Connection Character Sets and Collations”).

A `COLLATE` clause is not permitted within a `CONVERT()` or `CAST()` call, but you can apply it to the function result. For example, this is legal:

```
SELECT CAST('test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

But this is illegal:

```
SELECT CAST('test' AS CHAR CHARACTER SET utf8 COLLATE utf8_bin);
```

Normally, you cannot compare a `BLOB` value or other binary string in case-insensitive fashion because binary strings use the `binary` character set, which has no collation with the concept of lettercase. To perform a case-insensitive comparison, use the `CONVERT()` or `CAST()` function to convert the value to a nonbinary string. Comparisons of the resulting string use its collation. For example, if the conversion result character set has a case-insensitive collation, a `LIKE` operation is not case-sensitive:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1)
  FROM tbl_name;
```

To use a different character set, substitute its name for `latin1` in the preceding statement. To specify a particular collation for the converted string, use a `COLLATE` clause following the `CONVERT()` call:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) COLLATE latin1_german1_ci
  FROM tbl_name;
```

`CONVERT()` and `CAST()` can be used more generally for comparing strings that are represented in different character sets. For example, a comparison of these strings results in an error because they have different character sets:

```
mysql> SET @s1 = _latin1 'abc', @s2 = _latin2 'abc';
mysql> SELECT @s1 = @s2;
ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
and (latin2_general_ci,IMPLICIT) for operation '='
```

Converting one of the strings to a character set compatible with the other enables the comparison to occur without error:

```
mysql> SELECT @s1 = CONVERT(@s2 USING latin1);
+---------------------------------+
| @s1 = CONVERT(@s2 USING latin1) |
+---------------------------------+
|                               1 |
+---------------------------------+
```

For string literals, another way to specify the character set is to use a character set introducer (`_latin1` and `_latin2` in the preceding example are instances of introducers). Unlike conversion functions such as `CAST()`, or `CONVERT()`, which convert a string from one character set to another, an introducer designates a string literal as having a particular character set, with no conversion involved. For more information, see Section 10.3.8, “Character Set Introducers”.

Character set conversion is also useful preceding lettercase conversion of binary strings. `LOWER()` and `UPPER()` are ineffective when applied directly to binary strings because the concept of lettercase does not apply. To perform lettercase conversion of a binary string, first convert it to a nonbinary string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING utf8mb4));
+-------------+-------------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING utf8mb4)) |
+-------------+-------------------------------------+
| New York    | new york                            |
+-------------+-------------------------------------+
```

If you convert an indexed column using `BINARY`, `CAST()`, or `CONVERT()`, MySQL may not be able to use the index efficiently.

The cast functions are useful for creating a column with a specific type in a `CREATE TABLE ... SELECT` statement:

```
mysql> CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE) AS c1;
mysql> SHOW CREATE TABLE new_table\G
*************************** 1. row ***************************
       Table: new_table
Create Table: CREATE TABLE `new_table` (
  `c1` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

The cast functions are useful for sorting `ENUM` columns in lexical order. Normally, sorting of `ENUM` columns occurs using the internal numeric values. Casting the values to `CHAR` results in a lexical sort:

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

`CAST()` also changes the result if you use it as part of a more complex expression such as `CONCAT('Date: ',CAST(NOW() AS DATE))`.

For temporal values, there is little need to use `CAST()` to extract data in different formats. Instead, use a function such as `EXTRACT()`, `DATE_FORMAT()`, or `TIME_FORMAT()`. See Section 12.7, “Date and Time Functions”.

To cast a string to a number, you normally need do nothing other than use the string value in numeric context:

```
mysql> SELECT 1+'1';
       -> 2
```

That is also true for hexadecimal and bit literals, which are binary strings by default:

```
mysql> SELECT X'41', X'41'+0;
        -> 'A', 65
mysql> SELECT b'1100001', b'1100001'+0;
        -> 'a', 97
```

A string used in an arithmetic operation is converted to a floating-point number during expression evaluation.

A number used in string context is converted to a string:

```
mysql> SELECT CONCAT('hello you ',2);
        -> 'hello you 2'
```

For information about implicit conversion of numbers to strings, see Section 12.2, “Type Conversion in Expression Evaluation”.

MySQL supports arithmetic with both signed and unsigned 64-bit values. For numeric operators (such as `+` or `-`) where one of the operands is an unsigned integer, the result is unsigned by default (see Section 12.6.1, “Arithmetic Operators”). To override this, use the `SIGNED` or `UNSIGNED` cast operator to cast a value to a signed or unsigned 64-bit integer, respectively.

```
mysql> SELECT 1 - 2;
        -> -1
mysql> SELECT CAST(1 - 2 AS UNSIGNED);
        -> 18446744073709551615
mysql> SELECT CAST(CAST(1 - 2 AS UNSIGNED) AS SIGNED);
        -> -1
```

If either operand is a floating-point value, the result is a floating-point value and is not affected by the preceding rule. (In this context, `DECIMAL` column values are regarded as floating-point values.)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
        -> -1.0
```

The SQL mode affects the result of conversion operations (see Section 5.1.11, “Server SQL Modes”). Examples:

- For conversion of a “zero” date string to a date, `CONVERT()` and `CAST()` return `NULL` and produce a warning when the `NO_ZERO_DATE` SQL mode is enabled.
- For integer subtraction, if the `NO_UNSIGNED_SUBTRACTION` SQL mode is enabled, the subtraction result is signed even if any operand is unsigned.

The following list describes the available cast functions and operators:

- `BINARY` *`expr`*

  The `BINARY` operator converts the expression to a binary string. A common use for `BINARY` is to force a character string comparison to be done byte by byte rather than character by character, in effect becoming case-sensitive. The `BINARY` operator also causes trailing spaces in comparisons to be significant.

  ```
  mysql> SELECT 'a' = 'A';
          -> 1
  mysql> SELECT BINARY 'a' = 'A';
          -> 0
  mysql> SELECT 'a' = 'a ';
          -> 1
  mysql> SELECT BINARY 'a' = 'a ';
          -> 0
  ```

  In a comparison, `BINARY` affects the entire operation; it can be given before either operand with the same result.

  For purposes of converting a string expression to a binary string, these constructs are equivalent:

  ```
  BINARY expr
  CAST(expr AS BINARY)
  CONVERT(expr USING BINARY)
  ```

  If a value is a string literal, it can be designated as a binary string without performing any conversion by using the `_binary` character set introducer:

  ```
  mysql> SELECT 'a' = 'A';
          -> 1
  mysql> SELECT _binary 'a' = 'A';
          -> 0
  ```

  For information about introducers, see Section 10.3.8, “Character Set Introducers”.

  The `BINARY` operator in expressions differs in effect from the `BINARY` attribute in character column definitions. A character column defined with the `BINARY` attribute is assigned table default character set and the binary (`_bin`) collation of that character set. Every nonbinary character set has a `_bin` collation. For example, the binary collation for the `utf8` character set is `utf8_bin`, so if the table default character set is `utf8`, these two column definitions are equivalent:

```
CHAR(10) BINARY
CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin
```

The use of `CHARACTER SET binary` in the definition of a `CHAR`, `VARCHAR`, or `TEXT` column causes the column to be treated as the corresponding binary string data type. For example, the following pairs of definitions are equivalent:

```
CHAR(10) CHARACTER SET binary
BINARY(10)

VARCHAR(10) CHARACTER SET binary
VARBINARY(10)

TEXT CHARACTER SET binary
BLOB
```

- `CAST(expr AS type [ARRAY])`

  The `CAST()` function takes an expression of any type and produces a result value of the specified type, similar to `CONVERT()`. For more information, see the description of `CONVERT()`.

  In MySQL 8.0.17 and later, `InnoDB` allows the use of an additional `ARRAY` keyword for creating a multi-valued index on a `JSON` array as part of `CREATE INDEX`, `CREATE TABLE`, and `ALTER TABLE` statements. `ARRAY` is not supported except when used to create a multi-valued index in one of these statements, in which case it is required. The column being indexed must be a column of type `JSON`. (`CONVERT()` does *not* support multi-valued index creation or the `ARRAY` keyword.) The *`type`* following the `AS` keyword may be any of the types supported by `CAST()`, with the exception of `BINARY`, which is not supported for this purpose. Multi-Valued Indexes, provides syntax information and examples, as well as other relevant information.

  `CAST()` is standard SQL syntax.

- `CONVERT(expr,type)`, `CONVERT(expr USING transcoding_name)`

  The `CONVERT()` function takes an expression of any type and produces a result value of the specified type.

  Discussion of `CONVERT(expr, type)` syntax here also applies to `CAST(expr AS type)`, which is equivalent.

  `CONVERT(... USING ...)` is standard SQL syntax. The non-`USING` form of `CONVERT()` is ODBC syntax.

  `CONVERT()` with `USING` converts data between different character sets. In MySQL, transcoding names are the same as the corresponding character set names. For example, this statement converts the string `'abc'` in the default character set to the corresponding string in the `utf8` character set:

  ```
  SELECT CONVERT('abc' USING utf8);
  ```

  `CONVERT()` without `USING` and `CAST()` take an expression and a *`type`* value specifying the result type. These *`type`* values are permitted:

  - `BINARY[(N)]`

Produces a string with the `BINARY` data type. See Section 11.4.2, “The BINARY and VARBINARY Types” for a description of how this affects comparisons. If the optional length *`N`* is given, `BINARY(N)` causes the cast to use no more than *`N`* bytes of the argument. Values shorter than *`N`* bytes are padded with `0x00` bytes to a length of *`N`*.

- `CHAR[(N)] [charset_info]`

  Produces a string with the `CHAR` data type. If the optional length *`N`* is given, `CHAR(N)` causes the cast to use no more than *`N`* characters of the argument. No padding occurs for values shorter than *`N`* characters.

  With no *`charset_info`* clause, `CHAR` produces a string with the default character set. To specify the character set explicitly, these *`charset_info`* values are permitted:

  - `CHARACTER SET charset_name`: Produces a string with the given character set.
  - `ASCII`: Shorthand for `CHARACTER SET latin1`.
  - `UNICODE`: Shorthand for `CHARACTER SET ucs2`.

  In all cases, the string has the default collation for the character set.

- `DATE`

  Produces a `DATE` value.

- `DATETIME`

  Produces a `DATETIME` value.

- `DECIMAL[(M[,D])]`

  Produces a `DECIMAL` value. If the optional *`M`* and *`D`* values are given, they specify the maximum number of digits (the precision) and the number of digits following the decimal point (the scale).

- `DOUBLE`

  Produces a `DOUBLE` result. Added in MySQL 8.0.17.

- FLOAT[(*P*)]

  If the precision *`P`* is not specified, produces a result of type `FLOAT`. If *`P`* is provided and 0 <= < *`P`* <= 24, the result is of type `FLOAT`. If 25 <= *`P`* <= 53, the result is of type `DOUBLE`. If *`P`* < 0 or *`P`* > 53, an error is returned. Added in MySQL 8.0.17.

- `JSON`

  Produces a `JSON` value. For details on the rules for conversion of values between `JSON` and other types, see Comparison and Ordering of JSON Values.

- `NCHAR[(N)]`

  Like `CHAR`, but produces a string with the national character set. See Section 10.3.7, “The National Character Set”.

  Unlike `CHAR`, `NCHAR` does not permit trailing character set information to be specified.

- `REAL`

  Produces a result of type `REAL`. This is actually `FLOAT` if `REAL_AS_FLOAT` SQL mode is enabled; otherwise the result is of type `DOUBLE`.

- `SIGNED [INTEGER]`

  Produces a signed integer value.

- `TIME`

  Produces a `TIME` value.

- `UNSIGNED [INTEGER]`

  Produces an unsigned integer value.

12.11 XML Functions

Table 12.15 XML Functions

Name	Description
`ExtractValue()`	Extract a value from an XML string using XPath notation
`UpdateXML()`	Return replaced XML fragment

This section discusses XML and related functionality in MySQL.

Note

It is possible to obtain XML-formatted output from MySQL in the `mysql` and `mysqldump` clients by invoking them with the `--xml` option. See Section 4.5.1, "`mysql` — The MySQL Command-Line Client", and Section 4.5.4, "`mysqldump` — A Database Backup Program".

Two functions providing basic XPath 1.0 (XML Path Language, version 1.0) capabilities are available. Some basic information about XPath syntax and usage is provided later in this section; however, an in-depth discussion of these topics is beyond the scope of this manual, and you should refer to the XML Path Language (XPath) 1.0 standard for definitive information. A useful resource for those new to XPath or who desire a refresher in the basics is the Zvon.org XPath Tutorial, which is available in several languages.

Note

These functions remain under development. We continue to improve these and other aspects of XML and XPath functionality in MySQL 8.0 and onwards. You may discuss these, ask questions about them, and obtain help from other users with them in the MySQL XML User Forum.

XPath expressions used with these functions support user variables and local stored program variables. User variables are weakly checked; variables local to stored programs are strongly checked (see also Bug #26518):

- **User variables (weak checking).** Variables using the syntax `$@variable_name` (that is, user variables) are not checked. No warnings or errors are issued by the server if a variable has the wrong type or has previously not been assigned a value. This also means the user is fully responsible for any typographical errors, since no warnings will be given if (for example) `$@myvariable` is used where `$@myvariable` was intended.

Example:

```
mysql> SET @xml = '<a><b>X</b><b>Y</b></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @i =1, @j = 2;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @i, ExtractValue(@xml, '//b[$@i]');
+------+-------------------------------+
| @i   | ExtractValue(@xml, '//b[$@i]') |
+------+-------------------------------+
|    1 | X                             |
+------+-------------------------------+
1 row in set (0.00 sec)

mysql> SELECT @j, ExtractValue(@xml, '//b[$@j]');
+------+-------------------------------+
| @j   | ExtractValue(@xml, '//b[$@j]') |
+------+-------------------------------+
|    2 | Y                             |
+------+-------------------------------+
1 row in set (0.00 sec)

mysql> SELECT @k, ExtractValue(@xml, '//b[$@k]');
+------+-------------------------------+
| @k   | ExtractValue(@xml, '//b[$@k]') |
+------+-------------------------------+
| NULL |                               |
+------+-------------------------------+
1 row in set (0.00 sec)
```

- **Variables in stored programs (strong checking).** Variables using the syntax *`$variable_name`* can be declared and used with these functions when they are called inside stored programs. Such variables are local to the stored program in which they are defined, and are strongly checked for type and value.

  Example:

```
mysql> DELIMITER |

mysql> CREATE PROCEDURE myproc ()
    -> BEGIN
    ->   DECLARE i INT DEFAULT 1;
    ->   DECLARE xml VARCHAR(25) DEFAULT '<a>X</a><a>Y</a><a>Z</a>';
    ->
    ->   WHILE i < 4 DO
    ->     SELECT xml, i, ExtractValue(xml, '//a[$i]');
    ->     SET i = i+1;
    ->   END WHILE;
    -> END |
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;

mysql> CALL myproc();
+-------------------------+---+-----------------------------+
| xml                     | i | ExtractValue(xml, '//a[$i]') |
+-------------------------+---+-----------------------------+
| <a>X</a><a>Y</a><a>Z</a> | 1 | X                           |
+-------------------------+---+-----------------------------+
1 row in set (0.00 sec)
```

```
+-------------------------+---+-----------------------------+
| xml                     | i | ExtractValue(xml, '//a[$i]') |
+-------------------------+---+-----------------------------+
| <a>X</a><a>Y</a><a>Z</a> | 2 | Y                           |
+-------------------------+---+-----------------------------+
1 row in set (0.01 sec)

+-------------------------+---+-----------------------------+
| xml                     | i | ExtractValue(xml, '//a[$i]') |
+-------------------------+---+-----------------------------+
| <a>X</a><a>Y</a><a>Z</a> | 3 | Z                           |
+-------------------------+---+-----------------------------+
1 row in set (0.01 sec)
```

Parameters. Variables used in XPath expressions inside stored routines that are passed in as parameters are also subject to strong checking.

Expressions containing user variables or variables local to stored programs must otherwise (except for notation) conform to the rules for XPath expressions containing variables as given in the XPath 1.0 specification.

Note

A user variable used to store an XPath expression is treated as an empty string. Because of this, it is not possible to store an XPath expression as a user variable. (Bug #32911)

- `ExtractValue(xml_frag, xpath_expr)`

  `ExtractValue()` takes two string arguments, a fragment of XML markup *`xml_frag`* and an XPath expression *`xpath_expr`* (also known as a *locator*); it returns the text (`CDATA`) of the first text node which is a child of the element or elements matched by the XPath expression.

  Using this function is the equivalent of performing a match using the *`xpath_expr`* after appending `/text()`. In other words, `ExtractValue('<a><b>Sakila</b></a>', '/a/b')` and `ExtractValue('<a><b>Sakila</b></a>', '/a/b/text()')` produce the same result.

  If multiple matches are found, the content of the first child text node of each matching element is returned (in the order matched) as a single, space-delimited string.

  If no matching text node is found for the expression (including the implicit `/text()`)—for whatever reason, as long as *`xpath_expr`* is valid, and *`xml_frag`* consists of elements which are properly nested and closed—an empty string is returned. No distinction is made between a match on an empty element and no match at all. This is by design.

  If you need to determine whether no matching element was found in *`xml_frag`* or such an element was found but contained no child text nodes, you should test the result of an expression that uses the XPath `count()` function. For example, both of these statements return an empty string, as shown here:

```
mysql> SELECT ExtractValue('<a><b/></a>', '/a/b');
+-------------------------------------+
| ExtractValue('<a><b/></a>', '/a/b') |
+-------------------------------------+
|                                     |
+-------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c/></a>', '/a/b');
+-------------------------------------+
| ExtractValue('<a><c/></a>', '/a/b') |
```

```
+-------------------------------------+
|                                     |
+-------------------------------------+
1 row in set (0.00 sec)
```

However, you can determine whether there was actually a matching element using the following:

```
mysql> SELECT ExtractValue('<a><b/></a>', 'count(/a/b)');
+-------------------------------------+
| ExtractValue('<a><b/></a>', 'count(/a/b)') |
+-------------------------------------+
| 1                                   |
+-------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c/></a>', 'count(/a/b)');
+-------------------------------------+
| ExtractValue('<a><c/></a>', 'count(/a/b)') |
+-------------------------------------+
| 0                                   |
+-------------------------------------+
1 row in set (0.01 sec)
```

Important

`ExtractValue()` returns only `CDATA`, and does not return any tags that might be contained within a matching tag, nor any of their content (see the result returned as `val1` in the following example).

```
mysql> SELECT
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '/a') AS val1,
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '/a/b') AS val2,
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '//b') AS val3,
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '/b') AS val4,
    ->   ExtractValue('<a>ccc<b>ddd</b><b>eee</b></a>', '//b') AS val5;

+------+------+------+------+---------+
| val1 | val2 | val3 | val4 | val5    |
+------+------+------+------+---------+
| ccc  | ddd  | ddd  |      | ddd eee |
+------+------+------+------+---------+
```

This function uses the current SQL collation for making comparisons with `contains()`, performing the same collation aggregation as other string functions (such as `CONCAT()`), in taking into account the collation coercibility of their arguments; see Section 10.8.4, “Collation Coercibility in Expressions”, for an explanation of the rules governing this behavior.

(Previously, binary—that is, case-sensitive—comparison was always used.)

`NULL` is returned if *`xml_frag`* contains elements which are not properly nested or closed, and a warning is generated, as shown in this example:

```
mysql> SELECT ExtractValue('<a>c</a><b', '//a');
+---------------------------------+
| ExtractValue('<a>c</a><b', '//a') |
+---------------------------------+
| NULL                            |
+---------------------------------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1525
Message: Incorrect XML value: 'parse error at line 1 pos 11:
         END-OF-INPUT unexpected ('>' wanted)'
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
+-----------------------------------+
| ExtractValue('<a>c</a><b/>', '//a') |
+-----------------------------------+
| c                                 |
+-----------------------------------+
1 row in set (0.00 sec)
```

- `UpdateXML(xml_target, xpath_expr, new_xml)`

  This function replaces a single portion of a given fragment of XML markup *`xml_target`* with a new XML fragment *`new_xml`*, and then returns the changed XML. The portion of *`xml_target`* that is replaced matches an XPath expression *`xpath_expr`* supplied by the user.

  If no expression matching *`xpath_expr`* is found, or if multiple matches are found, the function returns the original *`xml_target`* XML fragment. All three arguments should be strings.

```
mysql> SELECT
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>') AS val1,
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val2,
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '//b', '<e>fff</e>') AS val3,
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val4,
    ->   UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val5
    -> \G

*************************** 1. row ***************************
val1: <e>fff</e>
val2: <a><b>ccc</b><d></d></a>
val3: <a><e>fff</e><d></d></a>
val4: <a><b>ccc</b><e>fff</e></a>
val5: <a><d></d><b>ccc</b><d></d></a>
```

Note

A discussion in depth of XPath syntax and usage are beyond the scope of this manual. Please see the XML Path Language (XPath) 1.0 specification for definitive information. A useful resource for those new to XPath or who are wishing a refresher in the basics is the Zvon.org XPath Tutorial, which is available in several languages.

Descriptions and examples of some basic XPath expressions follow:

- */tag*

  Matches *<tag/>* if and only if *<tag/>* is the root element.

  Example: */a* has a match in *<a>* because it matches the outermost (root) tag. It does not match the inner *a* element in *<a/>* because in this instance it is the child of another element.

- */tag1/tag2*

  Matches *<tag2/>* if and only if it is a child of *<tag1/>*, and *<tag1/>* is the root element.

Example: `/a/b` matches the *b* element in the XML fragment `<a><b/></a>` because it is a child of the root element *a*. It does not have a match in `<b><a/></b>` because in this case, *b* is the root element (and hence the child of no other element). Nor does the XPath expression have a match in `<a><c><b/></c></a>`; here, *b* is a descendant of *a*, but not actually a child of *a*.

This construct is extendable to three or more elements. For example, the XPath expression `/a/b/c` matches the *c* element in the fragment `<a><b><c/></b></a>`.

- *`//tag`*

  Matches any instance of `<tag>`.

  Example: `//a` matches the *a* element in any of the following: `<a><b><c/></b></a>`; `<c><a><b/></a></b>`; `<c><b><a/></b></c>`.

  `//` can be combined with `/`. For example, `//a/b` matches the *b* element in either of the fragments `<a><b/></a>` or `<c><a><b/></a></c>`.

Note

`//tag` is the equivalent of `/descendant-or-self::*/tag`. A common error is to confuse this with `/descendant-or-self::tag`, although the latter expression can actually lead to very different results, as can be seen here:

```
mysql> SET @xml = '<a><b><c>w</c><b>x</b><d>y</d>z</b></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @xml;
+----------------------------------------+
| @xml                                   |
+----------------------------------------+
| <a><b><c>w</c><b>x</b><d>y</d>z</b></a> |
+----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//b[1]');
+-------------------------------+
| ExtractValue(@xml, '//b[1]') |
+-------------------------------+
| x z                           |
+-------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//b[2]');
+-------------------------------+
| ExtractValue(@xml, '//b[2]') |
+-------------------------------+
|                               |
+-------------------------------+
1 row in set (0.01 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::*/b[1]');
+-------------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::*/b[1]') |
+-------------------------------------------------+
| x z                                             |
+-------------------------------------------------+
1 row in set (0.06 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::*/b[2]');
+-------------------------------------------------+
```

```
| ExtractValue(@xml, '/descendant-or-self::*/b[2]') |
+---------------------------------------------------+
|                                                   |
+---------------------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT ExtractValue(@xml, '/descendant-or-self::b[1]');
+---------------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::b[1]') |
+---------------------------------------------------+
| z                                                 |
+---------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::b[2]');
+---------------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::b[2]') |
+---------------------------------------------------+
| x                                                 |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

- The `*` operator acts as a “wildcard” that matches any element. For example, the expression `/*/b` matches the *b* element in either of the XML fragments `<a><b/></a>` or `<c><b/></c>`. However, the expression does not produce a match in the fragment `<b><a/></b>` because *b* must be a child of some other element. The wildcard may be used in any position: The expression `/*/b/*` will match any child of a *b* element that is itself not the root element.

- You can match any of several locators using the `|` (`UNION`) operator. For example, the expression `//b|//c` matches all *b* and *c* elements in the XML target.

- It is also possible to match an element based on the value of one or more of its attributes. This done using the syntax *`tag[@attribute="value"]`*. For example, the expression `//b[@id="idB"]` matches the second *b* element in the fragment `<a><b id="idA"/><c/><b id="idB"/></a>`. To match against *any* element having *`attribute="value"`*, use the XPath expression `//*[attribute="value"]`.

  To filter multiple attribute values, simply use multiple attribute-comparison clauses in succession. For example, the expression `//b[@c="x"][@d="y"]` matches the element `<b c="x" d="y"/>` occurring anywhere in a given XML fragment.

  To find elements for which the same attribute matches any of several values, you can use multiple locators joined by the `|` operator. For example, to match all *b* elements whose *c* attributes have either of the values 23 or 17, use the expression `//b[@c="23"]|//b[@c="17"]`. You can also use the logical `or` operator for this purpose: `//b[@c="23" or @c="17"]`.

Note

The difference between `or` and `|` is that `or` joins conditions, while `|` joins result sets.

XPath Limitations. The XPath syntax supported by these functions is currently subject to the following limitations:

- Nodeset-to-nodeset comparison (such as `'/a/b[@c=@d]'`) is not supported.

- All of the standard XPath comparison operators are supported. (Bug #22823)

- Relative locator expressions are resolved in the context of the root node. For example, consider the following query and result:

```
mysql> SELECT ExtractValue(
    ->   '<a><b c="1">X</b><b c="2">Y</b></a>',
    ->    'a/b'
    -> ) AS result;
+--------+
| result |
+--------+
| X Y    |
+--------+
1 row in set (0.03 sec)
```

In this case, the locator `a/b` resolves to `/a/b`.

Relative locators are also supported within predicates. In the following example, `d[../@c="1"]` is resolved as `/a/b[@c="1"]/d`:

```
mysql> SELECT ExtractValue(
    ->      '<a>
    ->        <b c="1"><d>X</d></b>
    ->        <b c="2"><d>X</d></b>
    ->      </a>',
    ->      'a/b/d[../@c="1"]')
    -> AS result;
+--------+
| result |
+--------+
| X      |
+--------+
1 row in set (0.00 sec)
```

- Locators prefixed with expressions that evaluate as scalar values—including variable references, literals, numbers, and scalar function calls—are not permitted, and their use results in an error.
- The `::` operator is not supported in combination with node types such as the following:
  - *`axis`*`::comment()`
  - *`axis`*`::text()`
  - *`axis`*`::processing-instructions()`
  - *`axis`*`::node()`

  However, name tests (such as *`axis`*`::`*`name`* and *`axis`*`::*`) are supported, as shown in these examples:

```
mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::b');
+-----------------------------------------------------+
| ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::b') |
+-----------------------------------------------------+
| x                                                   |
+-----------------------------------------------------+
1 row in set (0.02 sec)

mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::*');
+-----------------------------------------------------+
| ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::*') |
+-----------------------------------------------------+
| x y                                                 |
+-----------------------------------------------------+
1 row in set (0.01 sec)
```

- “Up-and-down” navigation is not supported in cases where the path would lead “above” the root element. That is, you cannot use expressions which match on descendants of ancestors of a given element, where one or more of the ancestors of the current element is also an ancestor of the root element (see Bug #16321).
- The following XPath functions are not supported, or have known issues as indicated:
  - `id()`
  - `lang()`
  - `local-name()`
  - `name()`
  - `namespace-uri()`
  - `normalize-space()`
  - `starts-with()`
  - `string()`
  - `substring-after()`
  - `substring-before()`
  - `translate()`
- The following axes are not supported:
  - `following-sibling`
  - `following`
  - `preceding-sibling`
  - `preceding`

XPath expressions passed as arguments to `ExtractValue()` and `UpdateXML()` may contain the colon character (`:`) in element selectors, which enables their use with markup employing XML namespaces notation. For example:

```
mysql> SET @xml = '<a>111<b:c>222<d>333</d><e:f>444</e:f></b:c></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//e:f');
+-----------------------------+
| ExtractValue(@xml, '//e:f') |
+-----------------------------+
| 444                         |
+-----------------------------+
1 row in set (0.00 sec)

mysql> SELECT UpdateXML(@xml, '//b:c', '<g:h>555</g:h>');
+-------------------------------------------+
| UpdateXML(@xml, '//b:c', '<g:h>555</g:h>') |
+-------------------------------------------+
| <a>111<g:h>555</g:h></a>                  |
+-------------------------------------------+
```

```
1 row in set (0.00 sec)
```

This is similar in some respects to what is permitted by Apache Xalan and some other parsers, and is much simpler than requiring namespace declarations or the use of the `namespace-uri()` and `local-name()` functions.

Error handling. For both `ExtractValue()` and `UpdateXML()`, the XPath locator used must be valid and the XML to be searched must consist of elements which are properly nested and closed. If the locator is invalid, an error is generated:

```
mysql> SELECT ExtractValue('<a>c</a><b/>', '/&a');
ERROR 1105 (HY000): XPATH syntax error: '&a'
```

If *`xml_frag`* does not consist of elements which are properly nested and closed, `NULL` is returned and a warning is generated, as shown in this example:

```
mysql> SELECT ExtractValue('<a>c</a><b', '//a');
+-----------------------------------+
| ExtractValue('<a>c</a><b', '//a') |
+-----------------------------------+
| NULL                              |
+-----------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1525
Message: Incorrect XML value: 'parse error at line 1 pos 11:
         END-OF-INPUT unexpected ('>' wanted)'
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
+------------------------------------+
| ExtractValue('<a>c</a><b/>', '//a') |
+------------------------------------+
| c                                  |
+------------------------------------+
1 row in set (0.00 sec)
```

Important

The replacement XML used as the third argument to `UpdateXML()` is *not* checked to determine whether it consists solely of elements which are properly nested and closed.

XPath Injection. *code injection* occurs when malicious code is introduced into the system to gain unauthorized access to privileges and data. It is based on exploiting assumptions made by developers about the type and content of data input from users. XPath is no exception in this regard.

A common scenario in which this can happen is the case of application which handles authorization by matching the combination of a login name and password with those found in an XML file, using an XPath expression like this one:

```
//user[login/text()='neapolitan' and password/text()='1c3cr34m']/attribute::id
```

This is the XPath equivalent of an SQL statement like this one:

```
SELECT id FROM users WHERE login='neapolitan' AND password='1c3cr34m';
```

A PHP application employing XPath might handle the login process like this:

```
<?php

  $file     =   "users.xml";

  $login    =   $POST["login"];
  $password =   $POST["password"];

  $xpath = "//user[login/text()=$login and password/text()=$password]/attribute::id";

  if( file_exists($file) )
  {
    $xml = simplexml_load_file($file);

    if($result = $xml->xpath($xpath))
      echo "You are now logged in as user $result[0].";
    else
      echo "Invalid login name or password.";
  }
  else
    exit("Failed to open $file.");

?>
```

No checks are performed on the input. This means that a malevolent user can “short-circuit” the test by entering `' or 1=1` for both the login name and password, resulting in `$xpath` being evaluated as shown here:

```
//user[login/text()='' or 1=1 and password/text()='' or 1=1]/attribute::id
```

Since the expression inside the square brackets always evaluates as `true`, it is effectively the same as this one, which matches the `id` attribute of every `user` element in the XML document:

```
//user/attribute::id
```

One way in which this particular attack can be circumvented is simply by quoting the variable names to be interpolated in the definition of `$xpath`, forcing the values passed from a Web form to be converted to strings:

```
$xpath = "//user[login/text()='$login' and password/text()='$password']/attribute::id";
```

This is the same strategy that is often recommended for preventing SQL injection attacks. In general, the practices you should follow for preventing XPath injection attacks are the same as for preventing SQL injection:

- Never accepted untested data from users in your application.
- Check all user-submitted data for type; reject or convert data that is of the wrong type
- Test numeric data for out of range values; truncate, round, or reject values that are out of range. Test strings for illegal characters and either strip them out or reject input containing them.
- Do not output explicit error messages that might provide an unauthorized user with clues that could be used to compromise the system; log these to a file or database table instead.

Just as SQL injection attacks can be used to obtain information about database schemas, so can XPath injection be used to traverse XML files to uncover their structure, as discussed in Amit Klein's paper Blind XPath Injection (PDF file, 46KB).

It is also important to check the output being sent back to the client. Consider what can happen when we use the MySQL `ExtractValue()` function:

```
mysql> SELECT ExtractValue(
    ->     LOAD_FILE('users.xml'),
    ->     '//user[login/text()="" or 1=1 and password/text()="" or 1=1]/attribute::id'
    -> ) AS id;
+-------------------------------+
| id                            |
+-------------------------------+
| 00327 13579 02403 42354 28570 |
+-------------------------------+
1 row in set (0.01 sec)
```

Because `ExtractValue()` returns multiple matches as a single space-delimited string, this injection attack provides every valid ID contained within `users.xml` to the user as a single row of output. As an extra safeguard, you should also test output before returning it to the user. Here is a simple example:

```
mysql> SELECT @id = ExtractValue(
    ->     LOAD_FILE('users.xml'),
    ->     '//user[login/text()="" or 1=1 and password/text()="" or 1=1]/attribute::id'
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT IF(
    ->     INSTR(@id, ' ') = 0,
    ->     @id,
    ->     'Unable to retrieve user ID')
    -> AS singleID;
+---------------------------+
| singleID                  |
+---------------------------+
| Unable to retrieve user ID |
+---------------------------+
1 row in set (0.00 sec)
```

In general, the guidelines for returning data to users securely are the same as for accepting user input. These can be summed up as:

- Always test outgoing data for type and permissible values.
- Never permit unauthorized users to view error messages that might provide information about the application that could be used to exploit it.

12.12 Bit Functions and Operators

Table 12.16 Bit Functions and Operators

Name	Description
`BIT_COUNT()`	Return the number of bits that are set
`&`	Bitwise AND
`~`	Bitwise inversion
`|`	Bitwise OR
`^`	Bitwise XOR
`<<`	Left shift
`>>`	Right shift

Bit functions and operators comprise `BIT_COUNT()`, `BIT_AND()`, `BIT_OR()`, `BIT_XOR()`, `&`, `|`, `^`, `~`, `<<`, and `>>`. (The `BIT_AND()`, `BIT_OR()`, and `BIT_XOR()` aggregate functions are described in Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”.) Prior to MySQL 8.0, bit functions and operators required `BIGINT` (64-bit integer) arguments and returned `BIGINT` values, so they had a maximum range of 64 bits. Non-`BIGINT` arguments were converted to `BIGINT` prior to performing the operation and truncation could occur.

In MySQL 8.0, bit functions and operators permit binary string type arguments (`BINARY`, `VARBINARY`, and the `BLOB` types) and return a value of like type, which enables them to take arguments and produce return values larger than 64 bits. Nonbinary string arguments are converted to `BIGINT` and processed as such, as before.

An implication of this change in behavior is that bit operations on binary string arguments might produce a different result in MySQL 8.0 than in 5.7. For information about how to prepare in MySQL 5.7 for potential incompatibilities between MySQL 5.7 and 8.0, see Bit Functions and Operators, in MySQL 5.7 Reference Manual.

- Bit Operations Prior to MySQL 8.0
- Bit Operations in MySQL 8.0
- Binary String Bit-Operation Examples
- Bitwise AND, OR, and XOR Operations
- Bitwise Complement and Shift Operations
- BIT_COUNT() Operations
- BIT_AND(), BIT_OR(), and BIT_XOR() Operations
- Special Handling of Hexadecimal Literals, Bit Literals, and NULL Literals
- Bit-Operation Incompatibilities with MySQL 5.7

Bit Operations Prior to MySQL 8.0

Bit operations prior to MySQL 8.0 handle only unsigned 64-bit integer argument and result values (that is, unsigned `BIGINT` values). Conversion of arguments of other types to `BIGINT` occurs as necessary. Examples:

- This statement operates on numeric literals, treated as unsigned 64-bit integers:

```
mysql> SELECT 127 | 128, 128 << 2, BIT_COUNT(15);
+-----------+----------+--------------+
| 127 | 128 | 128 << 2 | BIT_COUNT(15) |
+-----------+----------+--------------+
|       255 |      512 |             4 |
+-----------+----------+--------------+
```

- This statement performs to-number conversions on the string arguments (`'127'` to `127`, and so forth) before performing the same operations as the first statement and producing the same results:

```
mysql> SELECT '127' | '128', '128' << 2, BIT_COUNT('15');
+---------------+------------+-----------------+
| '127' | '128' | '128' << 2 | BIT_COUNT('15') |
+---------------+------------+-----------------+
```

```
|           255 |        512 |               4 |
+---------------+------------+-----------------+
```

- This statement uses hexadecimal literals for the bit-operation arguments. MySQL by default treats hexadecimal literals as binary strings, but in numeric context evaluates them as numbers (see Section 9.1.4, “Hexadecimal Literals”). Prior to MySQL 8.0, numeric context includes bit operations. Examples:

```
mysql> SELECT X'7F' | X'80', X'80' << 2, BIT_COUNT(X'0F');
+--------------+------------+-----------------+
| X'7F' | X'80' | X'80' << 2 | BIT_COUNT(X'0F') |
+--------------+------------+-----------------+
|           255 |        512 |               4 |
+---------------+------------+-----------------+
```

  Handling of bit-value literals in bit operations is similar to hexadecimal literals (that is, as numbers).

Bit Operations in MySQL 8.0

MySQL 8.0 extends bit operations to handle binary string arguments directly (without conversion) and produce binary string results. (Arguments that are not integers or binary strings are still converted to integers, as before.) This extension enhances bit operations in the following ways:

- Bit operations become possible on values longer than 64 bits.
- It is easier to perform bit operations on values that are more naturally represented as binary strings than as integers.

For example, consider UUID values and IPv6 addresses, which have human-readable text formats like this:

```
UUID: 6ccd780c-baba-1026-9564-5b8c656024db
IPv6: fe80::219:d1ff:fe91:1a72
```

It is cumbersome to operate on text strings in those formats. An alternative is convert them to fixed-length binary strings without delimiters. `UUID_TO_BIN()` and `INET6_ATON()` each produce a value of data type `BINARY(16)`, a binary string 16 bytes (128 bits) long. The following statements illustrate this (`HEX()` is used to produce displayable values):

```
mysql> SELECT HEX(UUID_TO_BIN('6ccd780c-baba-1026-9564-5b8c656024db'));
+--------------------------------------------------------+
| HEX(UUID_TO_BIN('6ccd780c-baba-1026-9564-5b8c656024db')) |
+--------------------------------------------------------+
| 6CCD780CBABA102695645B8C656024DB                       |
+--------------------------------------------------------+
mysql> SELECT HEX(INET6_ATON('fe80::219:d1ff:fe91:1a72'));
+-------------------------------------------+
| HEX(INET6_ATON('fe80::219:d1ff:fe91:1a72')) |
+-------------------------------------------+
| FE800000000000000219D1FFFE911A72          |
+-------------------------------------------+
```

Those binary values are easily manipulable with bit operations to perform actions such as extracting the timestamp from UUID values, or extracting the network and host parts of IPv6 addresses. (For examples, see later in this discussion.)

Arguments that count as binary strings include column values, routine parameters, local variables, and user-defined variables that have a binary string type: `BINARY`, `VARBINARY`, or one of the `BLOB` types.

What about hexadecimal literals and bit literals? Recall that those are binary strings by default in MySQL, but numbers in numeric context. How are they handled for bit operations in MySQL 8.0? Does MySQL continue to evaluate them in numeric context, as is done prior to MySQL 8.0? Or do bit operations evaluate them as binary strings, now that binary strings can be handled “natively” without conversion?

Answer: It has been common to specify arguments to bit operations using hexadecimal literals or bit literals with the intent that they represent numbers, so MySQL continues to evaluate bit operations in numeric context when all bit arguments are hexadecimal or bit literals, for backward compatility. If you require evaluation as binary strings instead, that is easily accomplished: Use the `_binary` introducer for at least one literal.

- These bit operations evaluate the hexadecimal literals and bit literals as integers:

```
mysql> SELECT X'40' | X'01', b'11110001' & b'01001111';
+--------------+--------------------------+
| X'40' | X'01' | b'11110001' & b'01001111' |
+--------------+--------------------------+
|           65 |                       65 |
+--------------+--------------------------+
```

- These bit operations evaluate the hexadecimal literals and bit literals as binary strings, due to the `_binary` introducer:

```
mysql> SELECT _binary X'40' | X'01', b'11110001' & _binary b'01001111';
+----------------------+----------------------------------+
| _binary X'40' | X'01' | b'11110001' & _binary b'01001111' |
+----------------------+----------------------------------+
| A                    | A                                |
+----------------------+----------------------------------+
```

Although the bit operations in both statements produce a result with a numeric value of 65, the second statement operates in binary-string context, for which 65 is ASCII A.

In numeric evaluation context, permitted values of hexadecimal literal and bit literal arguments have a maximum of 64 bits, as do results. By contrast, in binary-string evaluation context, permitted arguments (and results) can exceed 64 bits:

```
mysql> SELECT _binary X'4040404040404040' | X'0102030405060708';
+--------------------------------------------------+
| _binary X'4040404040404040' | X'0102030405060708' |
+--------------------------------------------------+
| ABCDEFGH                                         |
+--------------------------------------------------+
```

There are several ways to refer to a hexadecimal literal or bit literal in a bit operation to cause binary-string evaluation:

```
_binary literal
BINARY literal
CAST(literal AS BINARY)
```

Another way to produce binary-string evaluation of hexadecimal literals or bit literals is to assign them to user-defined variables, which results in variables that have a binary string type:

```
mysql> SET @v1 = X'40', @v2 = X'01', @v3 = b'11110001', @v4 = b'01001111';
mysql> SELECT @v1 | @v2, @v3 & @v4;
+-----------+-----------+
```

```
| @v1 | @v2 | @v3 & @v4 |
+-----------+-----------+
| A         | A         |
+-----------+-----------+
```

In binary-string context, bitwise operation arguments must have the same length or an `ER_INVALID_BITWISE_OPERANDS_SIZE` error occurs:

```
mysql> SELECT _binary X'40' | X'0001';
ERROR 3513 (HY000): Binary operands of bitwise
operators must be of equal length
```

To satisfy the equal-length requirement, pad the shorter value with leading zero digits or, if the longer value begins with leading zero digits and a shorter result value is acceptable, strip them:

```
mysql> SELECT _binary X'0040' | X'0001';
+--------------------------+
| _binary X'0040' | X'0001' |
+--------------------------+
|  A                       |
+--------------------------+
mysql> SELECT _binary X'40' | X'01';
+----------------------+
| _binary X'40' | X'01' |
+----------------------+
| A                    |
+----------------------+
```

Padding or stripping can also be accomplished using functions such as `LPAD()`, `RPAD()`, `SUBSTR()`, or `CAST()`. In such cases, the expression arguments are no longer all literals and `_binary` becomes unnecessary. Examples:

```
mysql> SELECT LPAD(X'40', 2, X'00') | X'0001';
+--------------------------------+
| LPAD(X'40', 2, X'00') | X'0001' |
+--------------------------------+
|  A                             |
+--------------------------------+
mysql> SELECT X'40' | SUBSTR(X'0001', 2, 1);
+-----------------------------+
| X'40' | SUBSTR(X'0001', 2, 1) |
+-----------------------------+
| A                           |
+-----------------------------+
```

Binary String Bit-Operation Examples

The following example illustrates use of bit operations to extract parts of a UUID value, in this case, the timestamp and IEEE 802 node number. This technique requires bitmasks for each extracted part.

Convert the text UUID to the corresponding 16-byte binary value so that it can be manipulated using bit operations in binary-string context:

```
mysql> SET @uuid = UUID_TO_BIN('6ccd780c-baba-1026-9564-5b8c656024db');
mysql> SELECT HEX(@uuid);
+----------------------------------+
| HEX(@uuid)                       |
+----------------------------------+
| 6CCD780CBABA102695645B8C656024DB |
+----------------------------------+
```

Construct bitmasks for the timestamp and node number parts of the value. The timestamp comprises the first three parts (64 bits, bits 0 to 63) and the node number is the last part (48 bits, bits 80 to 127):

```
mysql> SET @ts_mask = CAST(X'FFFFFFFFFFFFFFFF' AS BINARY(16));
mysql> SET @node_mask = CAST(X'FFFFFFFFFFFF' AS BINARY(16)) >> 80;
mysql> SELECT HEX(@ts_mask);
+----------------------------------+
| HEX(@ts_mask)                    |
+----------------------------------+
| FFFFFFFFFFFFFFFF0000000000000000 |
+----------------------------------+
mysql> SELECT HEX(@node_mask);
+----------------------------------+
| HEX(@node_mask)                  |
+----------------------------------+
| 00000000000000000000FFFFFFFFFFFF |
+----------------------------------+
```

The `CAST(... AS BINARY(16))` function is used here because the masks must be the same length as the UUID value against which they are applied. The same result can be produced using other functions to pad the masks to the required length:

```
SET @ts_mask= RPAD(X'FFFFFFFFFFFFFFFF' , 16, X'00');
SET @node_mask = LPAD(X'FFFFFFFFFFFF', 16, X'00') ;
```

Use the masks to extract the timestamp and node number parts:

```
mysql> SELECT HEX(@uuid & @ts_mask) AS 'timestamp part';
+----------------------------------+
| timestamp part                   |
+----------------------------------+
| 6CCD780CBABA10260000000000000000 |
+----------------------------------+
mysql> SELECT HEX(@uuid & @node_mask) AS 'node part';
+----------------------------------+
| node part                        |
+----------------------------------+
| 00000000000000000005B8C656024DB |
+----------------------------------+
```

The preceding example uses these bit operations: right shift (`>>`) and bitwise AND (`&`).

Note

`UUID_TO_BIN()` takes a flag that causes some bit rearrangement in the resulting binary UUID value. If you use that flag, modify the extraction masks accordingly.

The next example uses bit operations to extract the network and host parts of an IPv6 address. Suppose that the network part has a length of 80 bits. Then the host part has a length of 128 − 80 = 48 bits. To extract the network and host parts of the address, convert it to a binary string, then use bit operations in binary-string context.

Convert the text IPv6 address to the corresponding binary string:

```
mysql> SET @ip = INET6_ATON('fe80::219:d1ff:fe91:1a72');
```

Define the network length in bits:

```
mysql> SET @net_len = 80;
```

Construct network and host masks by shifting the all-ones address left or right. To do this, begin with the address `::`, which is shorthand for all zeros, as you can see by converting it to a binary string like this:

```
mysql> SELECT HEX(INET6_ATON('::')) AS 'all zeros';
+----------------------------------+
| all zeros                        |
+----------------------------------+
| 00000000000000000000000000000000 |
+----------------------------------+
```

To produce the complementary value (all ones), use the `~` operator to invert the bits:

```
mysql> SELECT HEX(~INET6_ATON('::')) AS 'all ones';
+----------------------------------+
| all ones                         |
+----------------------------------+
| FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF |
+----------------------------------+
```

Shift the all-ones value left or right to produce the network and host masks:

```
mysql> SET @net_mask = ~INET6_ATON('::') << (128 - @net_len);
mysql> SET @host_mask = ~INET6_ATON('::') >> @net_len;
```

Display the masks to verify that they cover the correct parts of the address:

```
mysql> SELECT INET6_NTOA(@net_mask) AS 'network mask';
+-----------------------------+
| network mask                |
+-----------------------------+
| ffff:ffff:ffff:ffff:ffff:: |
+-----------------------------+
mysql> SELECT INET6_NTOA(@host_mask) AS 'host mask';
+------------------------+
| host mask              |
+------------------------+
| ::ffff:255.255.255.255 |
+------------------------+
```

Extract and display the network and host parts of the address:

```
mysql> SET @net_part = @ip & @net_mask;
mysql> SET @host_part = @ip & @host_mask;
mysql> SELECT INET6_NTOA(@net_part) AS 'network part';
+-----------------+
| network part    |
+-----------------+
| fe80::219:0:0:0 |
+-----------------+
mysql> SELECT INET6_NTOA(@host_part) AS 'host part';
+------------------+
| host part        |
+------------------+
| ::d1ff:fe91:1a72 |
+------------------+
```

The preceding example uses these bit operations: Complement (`~`), left shift (`<<`), and bitwise AND (`&`).

The remaining discussion provides details on argument handling for each group of bit operations, more information about literal-value handling in bit operations, and potential incompatibilities between MySQL 8.0 and older MySQL versions.

Bitwise AND, OR, and XOR Operations

For `&`, `|`, and `^` bit operations, the result type depends on whether the arguments are evaluated as binary strings or numbers:

- Binary-string evaluation occurs when the arguments have a binary string type, and at least one of them is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument conversion to unsigned 64-bit integers as necessary.
- Binary-string evaluation produces a binary string of the same length as the arguments. If the arguments have unequal lengths, an `ER_INVALID_BITWISE_OPERANDS_SIZE` error occurs. Numeric evaluation produces an unsigned 64-bit integer.

Examples of numeric evaluation:

```
mysql> SELECT 64 | 1, X'40' | X'01';
+--------+---------------+
| 64 | 1 | X'40' | X'01' |
+--------+---------------+
|     65 |            65 |
+--------+---------------+
```

Examples of binary-string evaluation:

```
mysql> SELECT _binary X'40' | X'01';
+-----------------------+
| _binary X'40' | X'01' |
+-----------------------+
| A                     |
+-----------------------+
mysql> SET @var1 = X'40', @var2 = X'01';
mysql> SELECT @var1 | @var2;
+---------------+
| @var1 | @var2 |
+---------------+
| A             |
+---------------+
```

Bitwise Complement and Shift Operations

For `~`, `<<`, and `>>` bit operations, the result type depends on whether the bit argument is evaluated as a binary string or number:

- Binary-string evaluation occurs when the bit argument has a binary string type, and is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument conversion to an unsigned 64-bit integer as necessary.
- Binary-string evaluation produces a binary string of the same length as the bit argument. Numeric evaluation produces an unsigned 64-bit integer.

For shift operations, bits shifted off the end of the value are lost without warning, regardless of the argument type. In particular, if the shift count is greater or equal to the number of bits in the bit argument, all bits in the result are 0.

Examples of numeric evaluation:

```
mysql> SELECT ~0, 64 << 2, X'40' << 2;
+----------------------+---------+------------+
| ~0                   | 64 << 2 | X'40' << 2 |
+----------------------+---------+------------+
```

```
| 18446744073709551615 |     256 |        256 |
+----------------------+---------+------------+
```

Examples of binary-string evaluation:

```
mysql> SELECT HEX(_binary X'1111000022220000' >> 16);
+---------------------------------------+
| HEX(_binary X'1111000022220000' >> 16) |
+---------------------------------------+
| 0000111100002222                      |
+---------------------------------------+
mysql> SELECT HEX(_binary X'1111000022220000' << 16);
+---------------------------------------+
| HEX(_binary X'1111000022220000' << 16) |
+---------------------------------------+
| 0000222200000000                      |
+---------------------------------------+
mysql> SET @var1 = X'F0F0F0F0';
mysql> SELECT HEX(~@var1);
+-------------+
| HEX(~@var1) |
+-------------+
| 0F0F0F0F    |
+-------------+
```

BIT_COUNT() Operations

The `BIT_COUNT()` function always returns an unsigned 64-bit integer, or `NULL` if the argument is `NULL`.

```
mysql> SELECT BIT_COUNT(127);
+---------------+
| BIT_COUNT(127) |
+---------------+
|             7 |
+---------------+
mysql> SELECT BIT_COUNT(b'010101'), BIT_COUNT(_binary b'010101');
+---------------------+-----------------------------+
| BIT_COUNT(b'010101') | BIT_COUNT(_binary b'010101') |
+---------------------+-----------------------------+
|                   3 |                           3 |
+---------------------+-----------------------------+
```

BIT_AND(), BIT_OR(), and BIT_XOR() Operations

For the `BIT_AND()`, `BIT_OR()`, and `BIT_XOR()` bit functions, the result type depends on whether the function argument values are evaluated as binary strings or numbers:

- Binary-string evaluation occurs when the argument values have a binary string type, and the argument is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument value conversion to unsigned 64-bit integers as necessary.
- Binary-string evaluation produces a binary string of the same length as the argument values. If argument values have unequal lengths, an `ER_INVALID_BITWISE_OPERANDS_SIZE` error occurs. If the argument size exceeds 511 bytes, an `ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE` error occurs. Numeric evaluation produces an unsigned 64-bit integer.

`NULL` values do not affect the result unless all values are `NULL`. In that case, the result is a neutral value having the same length as the length of the argument values (all bits 1 for `BIT_AND()`, all bits 0 for `BIT_OR()`, and `BIT_XOR()`).

Example:

```
mysql> CREATE TABLE t (group_id INT, a VARBINARY(6));
mysql> INSERT INTO t VALUES (1, NULL);
mysql> INSERT INTO t VALUES (1, NULL);
mysql> INSERT INTO t VALUES (2, NULL);
mysql> INSERT INTO t VALUES (2, X'1234');
mysql> INSERT INTO t VALUES (2, X'FF34');
mysql> SELECT HEX(BIT_AND(a)), HEX(BIT_OR(a)), HEX(BIT_XOR(a))
       FROM t GROUP BY group_id;
+-----------------+----------------+-----------------+
| HEX(BIT_AND(a)) | HEX(BIT_OR(a)) | HEX(BIT_XOR(a)) |
+-----------------+----------------+-----------------+
| FFFFFFFFFFFF    | 000000000000   | 000000000000    |
| 1234            | FF34           | ED00            |
+-----------------+----------------+-----------------+
```

Special Handling of Hexadecimal Literals, Bit Literals, and NULL Literals

For backward compatibility, MySQL 8.0 evaluates bit operations in numeric context when all bit arguments are hexadecimal literals, bit literals, or `NULL` literals. That is, bit operations on binary-string bit arguments do not use binary-string evaluation if all bit arguments are unadorned hexadecimal literals, bit literals, or `NULL` literals. (This does not apply to such literals if they are written with a `_binary` introducer, `BINARY` operator, or other way of specifying them explicitly as binary strings.)

The literal handling just described is the same as prior to MySQL 8.0. Examples:

- These bit operations evaluate the literals in numeric context and produce a `BIGINT` result:

  ```
  b'0001' | b'0010'
  X'0008' << 8
  ```

- These bit operations evaluate `NULL` in numeric context and produce a `BIGINT` result that has a `NULL` value:

  ```
  NULL & NULL
  NULL >> 4
  ```

In MySQL 8.0, you can cause those operations to evaluate the arguments in binary-string context by indicating explicitly that at least one argument is a binary string:

```
_binary b'0001' | b'0010'
_binary X'0008' << 8
BINARY NULL & NULL
BINARY NULL >> 4
```

The result of the last two expressions is `NULL`, just as without the `BINARY` operator, but the data type of the result is a binary string type rather than an integer type.

Bit-Operation Incompatibilities with MySQL 5.7

Because bit operations can handle binary string arguments natively in MySQL 8.0, some expressions produce a different result in MySQL 8.0 than in 5.7. The five problematic expression types to watch out for are:

```
nonliteral_binary { & | ^ } binary
binary  { & | ^ } nonliteral_binary
nonliteral_binary { << >> } anything
~ nonliteral_binary
```

```
AGGR_BIT_FUNC(nonliteral_binary)
```

Those expressions return `BIGINT` in MySQL 5.7, binary string in 8.0.

Explanation of notation:

- `{ op1 op2 ... }`: List of operators that apply to the given expression type.
- *`binary`*: Any kind of binary string argument, including a hexadecimal literal, bit literal, or `NULL` literal.
- *`nonliteral_binary`*: An argument that is a binary string value other than a hexadecimal literal, bit literal, or `NULL` literal.
- *`AGGR_BIT_FUNC`*: An aggregate function that takes bit-value arguments: `BIT_AND()`, `BIT_OR()`, `BIT_XOR()`.

For information about how to prepare in MySQL 5.7 for potential incompatibilities between MySQL 5.7 and 8.0, see Bit Functions and Operators, in MySQL 5.7 Reference Manual.

The following list describes available bit functions and operators:

- `|`

  Bitwise OR.

  The result type depends on whether the arguments are evaluated as binary strings or numbers:

  - Binary-string evaluation occurs when the arguments have a binary string type, and at least one of them is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument conversion to unsigned 64-bit integers as necessary.
  - Binary-string evaluation produces a binary string of the same length as the arguments. If the arguments have unequal lengths, an `ER_INVALID_BITWISE_OPERANDS_SIZE` error occurs. Numeric evaluation produces an unsigned 64-bit integer.

  For more information, see the introductory discussion in this section.

```
mysql> SELECT 29 | 15;
        -> 31
mysql> SELECT _binary X'40404040' | X'01020304';
        -> 'ABCD'
```

- `&`

  Bitwise AND.

  The result type depends on whether the arguments are evaluated as binary strings or numbers:

  - Binary-string evaluation occurs when the arguments have a binary string type, and at least one of them is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument conversion to unsigned 64-bit integers as necessary.
  - Binary-string evaluation produces a binary string of the same length as the arguments. If the arguments have unequal lengths, an `ER_INVALID_BITWISE_OPERANDS_SIZE` error occurs. Numeric evaluation produces an unsigned 64-bit integer.

  For more information, see the introductory discussion in this section.

```
mysql> SELECT 29 & 15;
        -> 13
mysql> SELECT HEX(_binary X'FF' & b'11110000');
        -> 'F0'
```

- `^`

  Bitwise XOR.

  The result type depends on whether the arguments are evaluated as binary strings or numbers:

  - Binary-string evaluation occurs when the arguments have a binary string type, and at least one of them is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument conversion to unsigned 64-bit integers as necessary.
  - Binary-string evaluation produces a binary string of the same length as the arguments. If the arguments have unequal lengths, an `ER_INVALID_BITWISE_OPERANDS_SIZE` error occurs. Numeric evaluation produces an unsigned 64-bit integer.

  For more information, see the introductory discussion in this section.

```
mysql> SELECT 1 ^ 1;
        -> 0
mysql> SELECT 1 ^ 0;
        -> 1
mysql> SELECT 11 ^ 3;
        -> 8
mysql> SELECT HEX(_binary X'FEDC' ^ X'1111');
        -> 'EFCD'
```

- `<<`

  Shifts a longlong (`BIGINT`) number or binary string to the left.

  The result type depends on whether the bit argument is evaluated as a binary string or number:

  - Binary-string evaluation occurs when the bit argument has a binary string type, and is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument conversion to an unsigned 64-bit integer as necessary.
  - Binary-string evaluation produces a binary string of the same length as the bit argument. Numeric evaluation produces an unsigned 64-bit integer.

  Bits shifted off the end of the value are lost without warning, regardless of the argument type. In particular, if the shift count is greater or equal to the number of bits in the bit argument, all bits in the result are 0.

  For more information, see the introductory discussion in this section.

```
mysql> SELECT 1 << 2;
        -> 4
mysql> SELECT HEX(_binary X'00FF00FF00FF' << 8);
        -> 'FF00FF00FF00'
```

- `>>`

  Shifts a longlong (`BIGINT`) number or binary string to the right.

  The result type depends on whether the bit argument is evaluated as a binary string or number:

- Binary-string evaluation occurs when the bit argument has a binary string type, and is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument conversion to an unsigned 64-bit integer as necessary.
- Binary-string evaluation produces a binary string of the same length as the bit argument. Numeric evaluation produces an unsigned 64-bit integer.

Bits shifted off the end of the value are lost without warning, regardless of the argument type. In particular, if the shift count is greater or equal to the number of bits in the bit argument, all bits in the result are 0.

For more information, see the introductory discussion in this section.

```
mysql> SELECT 4 >> 2;
        -> 1
mysql> SELECT HEX(_binary X'00FF00FF00FF' >> 8);
        -> '0000FF00FF00'
```

- `~`

  Invert all bits.

  The result type depends on whether the bit argument is evaluated as a binary string or number:

  - Binary-string evaluation occurs when the bit argument has a binary string type, and is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument conversion to an unsigned 64-bit integer as necessary.
  - Binary-string evaluation produces a binary string of the same length as the bit argument. Numeric evaluation produces an unsigned 64-bit integer.

  For more information, see the introductory discussion in this section.

```
mysql> SELECT 5 & ~1;
        -> 4
mysql> SELECT HEX(~X'0000FFFF1111EEEE');
        -> 'FFFF0000EEEE1111'
```

- `BIT_COUNT(N)`

  Returns the number of bits that are set in the argument *`N`* as an unsigned 64-bit integer, or `NULL` if the argument is `NULL`.

```
mysql> SELECT BIT_COUNT(64), BIT_COUNT(BINARY 64);
        -> 1, 7
mysql> SELECT BIT_COUNT('64'), BIT_COUNT(_binary '64');
        -> 1, 7
mysql> SELECT BIT_COUNT(X'40'), BIT_COUNT(_binary X'40');
        -> 1, 1
```

12.13 Encryption and Compression Functions

Table 12.17 Encryption Functions

Name	Description
`AES_DECRYPT()`	Decrypt using AES

Name	Description
AES_ENCRYPT()	Encrypt using AES
ASYMMETRIC_DECRYPT()	Decrypt ciphertext using private or public key
ASYMMETRIC_DERIVE()	Derive symmetric key from asymmetric keys
ASYMMETRIC_ENCRYPT()	Encrypt cleartext using private or public key
ASYMMETRIC_SIGN()	Generate signature from digest
ASYMMETRIC_VERIFY()	Verify that signature matches digest
COMPRESS()	Return result as a binary string
CREATE_ASYMMETRIC_PRIV_KEY()	Create private key
CREATE_ASYMMETRIC_PUB_KEY()	Create public key
CREATE_DH_PARAMETERS()	Generate shared DH secret
CREATE_DIGEST()	Generate digest from string
DECODE()	Decode a string encrypted using ENCODE()
DES_DECRYPT()	Decrypt a string
DES_ENCRYPT()	Encrypt a string
ENCODE()	Encode a string
ENCRYPT()	Encrypt a string
MD5()	Calculate MD5 checksum
PASSWORD()	Calculate and return a password string
RANDOM_BYTES()	Return a random byte vector
SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
SHA2()	Calculate an SHA-2 checksum
STATEMENT_DIGEST()	Compute statement digest hash value
STATEMENT_DIGEST_TEXT()	Compute normalized statement digest
UNCOMPRESS()	Uncompress a string compressed
UNCOMPRESSED_LENGTH()	Return the length of a string before compression
VALIDATE_PASSWORD_STRENGTH()	Determine strength of password

Many encryption and compression functions return strings for which the result might contain arbitrary byte values. If you want to store these results, use a column with a VARBINARY or BLOB binary string data type. This will avoid potential problems with trailing space removal or character set conversion that would change data values, such as may occur if you use a nonbinary string data type (CHAR, VARCHAR, TEXT).

Some encryption functions return strings of ASCII characters: MD5(), SHA(), SHA1(), SHA2(), STATEMENT_DIGEST(), STATEMENT_DIGEST_TEXT(). Their return value is a string that has a character set and collation determined by the character_set_connection and collation_connection system variables. This is a nonbinary string unless the character set is binary.

If an application stores values from a function such as MD5() or SHA1() that returns a string of hex digits, more efficient storage and comparisons can be obtained by converting the hex representation to binary using UNHEX() and storing the result in a BINARY(*N*) column. Each pair of hexadecimal digits requires one byte in binary form, so the value of *N* depends on the length of the hex string. *N* is 16 for an MD5() value and 20 for a SHA1() value. For SHA2(), *N* ranges from 28 to 32 depending on the argument specifying the desired bit length of the result.

The size penalty for storing the hex string in a `CHAR` column is at least two times, up to eight times if the value is stored in a column that uses the `utf8` character set (where each character uses 4 bytes). Storing the string also results in slower comparisons because of the larger values and the need to take character set collation rules into account.

Suppose that an application stores `MD5()` string values in a `CHAR(32)` column:

```
CREATE TABLE md5_tbl (md5_val CHAR(32), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(MD5('abcdef'), ...);
```

To convert hex strings to more compact form, modify the application to use `UNHEX()` and `BINARY(16)` instead as follows:

```
CREATE TABLE md5_tbl (md5_val BINARY(16), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(UNHEX(MD5('abcdef')), ...);
```

Applications should be prepared to handle the very rare case that a hashing function produces the same value for two different input values. One way to make collisions detectable is to make the hash column a primary key.

Note

Exploits for the MD5 and SHA-1 algorithms have become known. You may wish to consider using another one-way encryption function described in this section instead, such as `SHA2()`.

Caution

Passwords or other sensitive values supplied as arguments to encryption functions are sent as cleartext to the MySQL server unless an SSL connection is used. Also, such values will appear in any MySQL logs to which they are written. To avoid these types of exposure, applications can encrypt sensitive values on the client side before sending them to the server. The same considerations apply to encryption keys. To avoid exposing these, applications can use stored procedures to encrypt and decrypt values on the server side.

- `AES_DECRYPT(`*`crypt_str`*`,`*`key_str`*`[,`*`init_vector`*`])`

  This function decrypts data using the official AES (Advanced Encryption Standard) algorithm. For more information, see the description of `AES_ENCRYPT()`.

  The optional initialization vector argument, *`init_vector`*. Statements that use `AES_DECRYPT()` are unsafe for statement-based replication.

- `AES_ENCRYPT(`*`str`*`,`*`key_str`*`[,`*`init_vector`*`])`

  `AES_ENCRYPT()` and `AES_DECRYPT()` implement encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as “Rijndael.” The AES standard permits various key lengths. By default these functions implement AES with a 128-bit key length. Key lengths of 196 or 256 bits can be used, as described later. The key length is a trade off between performance and security.

  `AES_ENCRYPT()` encrypts the string *`str`* using the key string *`key_str`* and returns a binary string containing the encrypted output. `AES_DECRYPT()` decrypts the encrypted string *`crypt_str`* using the key string *`key_str`* and returns the original plaintext string. If either function argument is `NULL`, the function returns `NULL`.

The *str* and *crypt_str* arguments can be any length, and padding is automatically added to *str* so it is a multiple of a block as required by block-based algorithms such as AES. This padding is automatically removed by the AES_DECRYPT() function. The length of *crypt_str* can be calculated using this formula:

```
16 * (trunc(string_length / 16) + 1)
```

For a key length of 128 bits, the most secure way to pass a key to the *key_str* argument is to create a truly random 128-bit value and pass it as a binary value. For example:

```
INSERT INTO t
VALUES (1,AES_ENCRYPT('text',UNHEX('F3229A0B371ED2D9441B830D21A390C3')));
```

A passphrase can be used to generate an AES key by hashing the passphrase. For example:

```
INSERT INTO t
VALUES (1,AES_ENCRYPT('text', UNHEX(SHA2('My secret passphrase',512))));
```

Do not pass a password or passphrase directly to *crypt_str*, hash it first. Previous versions of this documentation suggested the former approach, but it is no longer recommended as the examples shown here are more secure.

If AES_DECRYPT() detects invalid data or incorrect padding, it returns NULL. However, it is possible for AES_DECRYPT() to return a non-NULL value (possibly garbage) if the input data or the key is invalid.

AES_ENCRYPT() and AES_DECRYPT() permit control of the block encryption mode and take an optional *init_vector* initialization vector argument:

- The block_encryption_mode system variable controls the mode for block-based encryption algorithms. Its default value is aes-128-ecb, which signifies encryption using a key length of 128 bits and ECB mode. For a description of the permitted values of this variable, see Section 5.1.8, “Server System Variables”.
- The optional *init_vector* argument provides an initialization vector for block encryption modes that require it.

For modes that require the optional *init_vector* argument, it must be 16 bytes or longer (bytes in excess of 16 are ignored). An error occurs if *init_vector* is missing.

For modes that do not require *init_vector*, it is ignored and a warning is generated if it is specified.

A random string of bytes to use for the initialization vector can be produced by calling RANDOM_BYTES(16). For encryption modes that require an initialization vector, the same vector must be used for encryption and decryption.

```
mysql> SET block_encryption_mode = 'aes-256-cbc';
mysql> SET @key_str = SHA2('My secret passphrase',512);
mysql> SET @init_vector = RANDOM_BYTES(16);
mysql> SET @crypt_str = AES_ENCRYPT('text',@key_str,@init_vector);
mysql> SELECT AES_DECRYPT(@crypt_str,@key_str,@init_vector);
+-------------------------------------------------+
| AES_DECRYPT(@crypt_str,@key_str,@init_vector) |
+-------------------------------------------------+
| text                                            |
+-------------------------------------------------+
```

The following table lists each permitted block encryption mode and whether the initialization vector argument is required.

Block Encryption Mode	Initialization Vector Required
ECB	No
CBC	Yes
CFB1	Yes
CFB8	Yes
CFB128	Yes
OFB	Yes

Statements that use `AES_ENCRYPT()` or `AES_DECRYPT()` are unsafe for statement-based replication.

- `COMPRESS(string_to_compress)`

  Compresses a string and returns the result as a binary string. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`. The compressed string can be uncompressed with `UNCOMPRESS()`.

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
        -> 21
mysql> SELECT LENGTH(COMPRESS(''));
        -> 0
mysql> SELECT LENGTH(COMPRESS('a'));
        -> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
        -> 15
```

  The compressed string contents are stored the following way:

  - Empty strings are stored as empty strings.
  - Nonempty strings are stored as a 4-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra `.` character is added to avoid problems with endspace trimming should the result be stored in a `CHAR` or `VARCHAR` column. (However, use of nonbinary string data types such as `CHAR` or `VARCHAR` to store compressed strings is not recommended anyway because character set conversion may occur. Use a `VARBINARY` or `BLOB` binary string column instead.)

- `DECODE(crypt_str,pass_str)`

  This function was removed in MySQL 8.0.3.

  Consider using `AES_ENCRYPT()` and `AES_DECRYPT()` instead.

- `DES_DECRYPT(crypt_str[,key_str])`

  This function was removed in MySQL 8.0.3.

  Consider using `AES_ENCRYPT()` and `AES_DECRYPT()` instead.

- `DES_ENCRYPT(str[,{key_num|key_str}])`

  This function was removed in MySQL 8.0.3.

Consider using AES_ENCRYPT() and AES_DECRYPT() instead.

- ENCODE(*str*,*pass_str*)

  This function was removed in MySQL 8.0.3.

  Consider using AES_ENCRYPT() and AES_DECRYPT() instead.

- ENCRYPT(*str*[,*salt*])

  This function was removed in MySQL 8.0.3. For one-way hashing, consider using SHA2() instead.

- MD5(*str*)

  Calculates an MD5 128-bit checksum for the string. The value is returned as a string of 32 hexadecimal digits, or NULL if the argument was NULL. The return value can, for example, be used as a hash key. See the notes at the beginning of this section about storing hash values efficiently.

  The return value is a string in the connection character set.

  If FIPS mode is enabled, MD5() returns NULL. See Section 6.5, “FIPS Support”.

```
mysql> SELECT MD5('testing');
        -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

  This is the “RSA Data Security, Inc. MD5 Message-Digest Algorithm.”

  See the note regarding the MD5 algorithm at the beginning this section.

- PASSWORD(*str*)

  This function was removed in MySQL 8.0.11.

- RANDOM_BYTES(*len*)

  This function returns a binary string of *len* random bytes generated using the random number generator of the SSL library. Permitted values of *len* range from 1 to 1024. For values outside that range, RANDOM_BYTES() generates a warning and returns NULL.

  RANDOM_BYTES() can be used to provide the initialization vector for the AES_DECRYPT() and AES_ENCRYPT() functions. For use in that context, *len* must be at least 16. Larger values are permitted, but bytes in excess of 16 are ignored.

  RANDOM_BYTES() generates a random value, which makes its result nondeterministic. Consequently, statements that use this function are unsafe for statement-based replication.

- SHA1(*str*), SHA(*str*)

  Calculates an SHA-1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a string of 40 hexadecimal digits, or NULL if the argument was NULL. One of the possible uses for this function is as a hash key. See the notes at the beginning of this section about storing hash values efficiently. SHA() is synonymous with SHA1().

  The return value is a string in the connection character set.

```
mysql> SELECT SHA1('abc');
        -> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

`SHA1()` can be considered a cryptographically more secure equivalent of `MD5()`. However, see the note regarding the MD5 and SHA-1 algorithms at the beginning this section.

- `SHA2(str, hash_length)`

  Calculates the SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, and SHA-512). The first argument is the plaintext string to be hashed. The second argument indicates the desired bit length of the result, which must have a value of 224, 256, 384, 512, or 0 (which is equivalent to 256). If either argument is `NULL` or the hash length is not one of the permitted values, the return value is `NULL`. Otherwise, the function result is a hash value containing the desired number of bits. See the notes at the beginning of this section about storing hash values efficiently.

  The return value is a string in the connection character set.

```
mysql> SELECT SHA2('abc', 224);
        -> '23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7'
```

  This function works only if MySQL has been configured with SSL support. See Section 6.3, “Using Encrypted Connections”.

  `SHA2()` can be considered cryptographically more secure than `MD5()` or `SHA1()`.

- `STATEMENT_DIGEST(statement)`

  Given an SQL statement as a string, returns the statement digest hash value as a string in the connection character set, or `NULL` if the argument is `NULL`. The related `STATEMENT_DIGEST_TEXT()` function returns the normalized statement digest. For information about statement digesting, see Section 26.10, “Performance Schema Statement Digests and Sampling”.

  Both functions use the MySQL parser to parse the statement. If parsing fails, an error occurs. The error message includes the parse error only if the statement is provided as a literal string.

  The `max_digest_length` system variable determines the maximum number of bytes available to these functions for computing normalized statement digests.

```
mysql> SET @stmt = 'SELECT * FROM mytable WHERE cola = 10 AND colb = 20';
mysql> SELECT STATEMENT_DIGEST(@stmt);
+------------------------------------------------------------------+
| STATEMENT_DIGEST(@stmt)                                          |
+------------------------------------------------------------------+
| 3bb95eeade896657c4526e74ff2a2862039d0a0fe8a9e7155b5fe492cbd78387 |
+------------------------------------------------------------------+
mysql> SELECT STATEMENT_DIGEST_TEXT(@stmt);
+--------------------------------------------------------+
| STATEMENT_DIGEST_TEXT(@stmt)                           |
+--------------------------------------------------------+
| SELECT * FROM `mytable` WHERE `cola` = ? AND `colb` = ?  |
+--------------------------------------------------------+
```

- `STATEMENT_DIGEST_TEXT(statement)`

  Given an SQL statement as a string, returns the normalized statement digest as a string in the connection character set, or `NULL` if the argument is `NULL`. For additional discussion and examples, see the description of the related `STATEMENT_DIGEST()` function.

- `UNCOMPRESS(string_to_uncompress)`

Uncompresses a string compressed by the `COMPRESS()` function. If the argument is not a compressed value, the result is `NULL`. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
        -> 'any string'
mysql> SELECT UNCOMPRESS('any string');
        -> NULL
```

- `UNCOMPRESSED_LENGTH(compressed_string)`

  Returns the length that the compressed string had before being compressed.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
        -> 30
```

- `VALIDATE_PASSWORD_STRENGTH(str)`

  Given an argument representing a plaintext password, this function returns an integer to indicate how strong the password is. The return value ranges from 0 (weak) to 100 (strong).

  Password assessment by `VALIDATE_PASSWORD_STRENGTH()` is done by the `validate_password` component. If that component is not installed, the function always returns 0. For information about installing `validate_password`, see Section 6.4.3, “The Password Validation Component”. To examine or configure the parameters that affect password testing, check or set the system variables implemented by `validate_password`. See Section 6.4.3.2, “Password Validation Options and Variables”.

  The password is subjected to increasingly strict tests and the return value reflects which tests were satisfied, as shown in the following table. In addition, if the `validate_password.check_user_name` system variable is enabled and the password matches the user name, `VALIDATE_PASSWORD_STRENGTH()` returns 0 regardless of how other `validate_password` system variables are set.

Password Test	Return Value
Length < 4	0
Length ≥ 4 and < `validate_password.length`	25
Satisfies policy 1 (`LOW`)	50
Satisfies policy 2 (`MEDIUM`)	75
Satisfies policy 3 (`STRONG`)	100

12.14 Locking Functions

This section describes functions used to manipulate user-level locks.

Table 12.18 Locking Functions

Name	Description
`GET_LOCK()`	Get a named lock
`IS_FREE_LOCK()`	Whether the named lock is free
`IS_USED_LOCK()`	Whether the named lock is in use; return connection identifier if true

Name	Description
RELEASE_ALL_LOCKS()	Release all current named locks
RELEASE_LOCK()	Release the named lock

- GET_LOCK(*str*,*timeout*)

  Tries to obtain a lock with a name given by the string *str*, using a timeout of *timeout* seconds. A negative *timeout* value means infinite timeout. The lock is exclusive. While held by one session, other sessions cannot obtain a lock of the same name.

  Returns 1 if the lock was obtained successfully, 0 if the attempt timed out (for example, because another client has previously locked the name), or NULL if an error occurred (such as running out of memory or the thread was killed with mysqladmin kill).

  A lock obtained with GET_LOCK() is released explicitly by executing RELEASE_LOCK() or implicitly when your session terminates (either normally or abnormally). Locks obtained with GET_LOCK() are not released when transactions commit or roll back.

  GET_LOCK() is implemented using the metadata locking (MDL) subsystem. Multiple simultaneous locks can be acquired and GET_LOCK() does not release any existing locks. For example, suppose that you execute these statements:

```
SELECT GET_LOCK('lock1',10);
SELECT GET_LOCK('lock2',10);
SELECT RELEASE_LOCK('lock2');
SELECT RELEASE_LOCK('lock1');
```

  The second GET_LOCK() acquires a second lock and both RELEASE_LOCK() calls return 1 (success).

  It is even possible for a given session to acquire multiple locks for the same name. Other sessions cannot acquire a lock with that name until the acquiring session releases all its locks for the name.

  Uniquely named locks acquired with GET_LOCK() appear in the Performance Schema metadata_locks table. The OBJECT_TYPE column says USER LEVEL LOCK and the OBJECT_NAME column indicates the lock name. In the case that multiple locks are acquired for the *same* name, only the first lock for the name registers a row in the metadata_locks table. Subsequent locks for the name increment a counter in the lock but do not acquire additional metadata locks. The metadata_locks row for the lock is deleted when the last lock instance on the name is released.

  The capability of acquiring multiple locks means there is the possibility of deadlock among clients. When this happens, the server chooses a caller and terminates its lock-acquisition request with an ER_USER_LOCK_DEADLOCK error. This error does not cause transactions to roll back.

  MySQL enforces a maximum length on lock names of 64 characters.

  GET_LOCK() can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked within one session, GET_LOCK() blocks any request by another session for a lock with the same name. This enables clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also enables a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form *db_name.str* or *app_name.str*.

If multiple clients are waiting for a lock, the order in which they will acquire it is undefined. Applications should not assume that clients will acquire the lock in the same order that they issued the lock requests.

`GET_LOCK()` is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

Caution

With the capability of acquiring multiple named locks, it is possible for a single statement to acquire a large number of locks. For example:

```
INSERT INTO ... SELECT GET_LOCK(t1.col_name) FROM t1;
```

These types of statements may have certain adverse effects. For example, if the statement fails part way through and rolls back, locks acquired up to the point of failure will still exist. If the intent is for there to be a correspondence between rows inserted and locks acquired, that intent will not be satisfied. Also, if it is important that locks are granted in a certain order, be aware that result set order may differ depending on which execution plan the optimizer chooses. For these reasons, it may be best to limit applications to a single lock-acquisition call per statement.

A different locking interface is available as either a plugin service or a set of user-defined functions. This interface provides lock namespaces and distinct read and write locks, unlike the interface provided by `GET_LOCK()` and related functions. For details, see Section 29.3.1, “The Locking Service”.

- `IS_FREE_LOCK(`*`str`*`)`

  Checks whether the lock named *`str`* is free to use (that is, not locked). Returns `1` if the lock is free (no one is using the lock), `0` if the lock is in use, and `NULL` if an error occurs (such as an incorrect argument).

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `IS_USED_LOCK(`*`str`*`)`

  Checks whether the lock named *`str`* is in use (that is, locked). If so, it returns the connection identifier of the client session that holds the lock. Otherwise, it returns `NULL`.

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `RELEASE_ALL_LOCKS()`

  Releases all named locks held by the current session and returns the number of locks released (0 if there were none)

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `RELEASE_LOCK(`*`str`*`)`

  Releases the lock named by the string *`str`* that was obtained with `GET_LOCK()`. Returns `1` if the lock was released, `0` if the lock was not established by this thread (in which case the lock is not released), and `NULL` if the named lock did not exist. The lock does not exist if it was never obtained by a call to `GET_LOCK()` or if it has previously been released.

The DO statement is convenient to use with RELEASE_LOCK(). See Section 13.2.3, “DO Syntax”.

This function is unsafe for statement-based replication. A warning is logged if you use this function when binlog_format is set to STATEMENT.

12.15 Information Functions

Table 12.19 Information Functions

Name	Description
BENCHMARK()	Repeatedly execute an expression
CHARSET()	Return the character set of the argument
COERCIBILITY()	Return the collation coercibility value of the string argument
COLLATION()	Return the collation of the string argument
CONNECTION_ID()	Return the connection ID (thread ID) for the connection
CURRENT_ROLE()	Return the current active roles
CURRENT_USER(), CURRENT_USER	The authenticated user name and host name
DATABASE()	Return the default (current) database name
FOUND_ROWS()	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
ICU_VERSION()	ICU library version
LAST_INSERT_ID()	Value of the AUTOINCREMENT column for the last INSERT
ROLES_GRAPHML()	Return a GraphML document representing memory role subgraphs
ROW_COUNT()	The number of rows updated
SCHEMA()	Synonym for DATABASE()
SESSION_USER()	Synonym for USER()
SYSTEM_USER()	Synonym for USER()
USER()	The user name and host name provided by the client
VERSION()	Return a string that indicates the MySQL server version

- BENCHMARK(*count*,*expr*)

  The BENCHMARK() function executes the expression *expr* repeatedly *count* times. It may be used to time how quickly MySQL processes the expression. The result value is 0, or NULL for inappropriate arguments such as a NULL or negative repeat count.

  The intended use is from within the mysql client, which reports query execution times:

```
mysql> SELECT BENCHMARK(1000000,AES_ENCRYPT('hello','goodbye'));
+-------------------------------------------------+
| BENCHMARK(1000000,AES_ENCRYPT('hello','goodbye')) |
+-------------------------------------------------+
|                                               0 |
+-------------------------------------------------+
1 row in set (4.74 sec)
```

The time reported is elapsed time on the client end, not CPU time on the server end. It is advisable to execute `BENCHMARK()` several times, and to interpret the result with regard to how heavily loaded the server machine is.

`BENCHMARK()` is intended for measuring the runtime performance of scalar expressions, which has some significant implications for the way that you use it and interpret the results:

- Only scalar expressions can be used. Although the expression can be a subquery, it must return a single column and at most a single row. For example, `BENCHMARK(10, (SELECT * FROM t))` will fail if the table `t` has more than one column or more than one row.
- Executing a `SELECT expr` statement `N` times differs from executing `SELECT BENCHMARK(N, expr)` in terms of the amount of overhead involved. The two have very different execution profiles and you should not expect them to take the same amount of time. The former involves the parser, optimizer, table locking, and runtime evaluation `N` times each. The latter involves only runtime evaluation `N` times, and all the other components just once. Memory structures already allocated are reused, and runtime optimizations such as local caching of results already evaluated for aggregate functions can alter the results. Use of `BENCHMARK()` thus measures performance of the runtime component by giving more weight to that component and removing the “noise” introduced by the network, parser, optimizer, and so forth.

- `CHARSET(str)`

  Returns the character set of the string argument.

```
mysql> SELECT CHARSET('abc');
        -> 'utf8'
mysql> SELECT CHARSET(CONVERT('abc' USING latin1));
        -> 'latin1'
mysql> SELECT CHARSET(USER());
        -> 'utf8'
```

- `COERCIBILITY(str)`

  Returns the collation coercibility value of the string argument.

```
mysql> SELECT COERCIBILITY('abc' COLLATE utf8_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(USER());
        -> 3
mysql> SELECT COERCIBILITY('abc');
        -> 4
mysql> SELECT COERCIBILITY(1000);
        -> 5
```

  The return values have the meanings shown in the following table. Lower values have higher precedence.

Coercibility	Meaning	Example
0	Explicit collation	Value with `COLLATE` clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value, stored routine parameter or local variable

Coercibility	Meaning	Example
3	System constant	`USER()` return value
4	Coercible	Literal string
5	Numeric	Numeric or temporal value
5	Ignorable	`NULL` or an expression derived from `NULL`

For more information, see Section 10.8.4, “Collation Coercibility in Expressions”.

- `COLLATION(str)`

  Returns the collation of the string argument.

```
mysql> SELECT COLLATION('abc');
        -> 'utf8_general_ci'
mysql> SELECT COLLATION(_utf8mb4'abc');
        -> 'utf8mb4_0900_ai_ci'
mysql> SELECT COLLATION(_latin1'abc');
        -> 'latin1_swedish_ci'
```

- `CONNECTION_ID()`

  Returns the connection ID (thread ID) for the connection. Every connection has an ID that is unique among the set of currently connected clients.

  The value returned by `CONNECTION_ID()` is the same type of value as displayed in the `ID` column of the `INFORMATION_SCHEMA.PROCESSLIST` table, the `Id` column of `SHOW PROCESSLIST` output, and the `PROCESSLIST_ID` column of the Performance Schema `threads` table.

```
mysql> SELECT CONNECTION_ID();
        -> 23786
```

- `CURRENT_ROLE()`

  Returns a `utf8` string containing the current active roles for the current session, separated by commas, or `NONE` if there are none. The value reflects the setting of the `sql_quote_show_create` system variable.

  Suppose that an account is granted roles as follows:

```
GRANT 'r1', 'r2' TO 'u1'@'localhost';
SET DEFAULT ROLE ALL TO 'u1'@'localhost';
```

  In sessions for `u1`, the initial `CURRENT_ROLE()` value names the default account roles. Using `SET ROLE` changes that:

```
mysql> SELECT CURRENT_ROLE();
+-------------------+
| CURRENT_ROLE()    |
+-------------------+
| `r1`@`%`,`r2`@`%` |
+-------------------+
mysql> SET ROLE 'r1'; SELECT CURRENT_ROLE();
+----------------+
| CURRENT_ROLE() |
```

```
+----------------+
| `r1`@`%`       |
+----------------+
```

- `CURRENT_USER`, `CURRENT_USER()`

  Returns the user name and host name combination for the MySQL account that the server used to authenticate the current client. This account determines your access privileges. The return value is a string in the `utf8` character set.

  The value of `CURRENT_USER()` can differ from the value of `USER()`.

```
mysql> SELECT USER();
        -> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user ''@'localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
        -> '@localhost'
```

  The example illustrates that although the client specified a user name of `davida` (as indicated by the value of the `USER()` function), the server authenticated the client using an anonymous user account (as seen by the empty user name part of the `CURRENT_USER()` value). One way this might occur is that there is no account listed in the grant tables for `davida`.

  Within a stored program or view, `CURRENT_USER()` returns the account for the user who defined the object (as given by its `DEFINER` value) unless defined with the `SQL SECURITY INVOKER` characteristic. In the latter case, `CURRENT_USER()` returns the object's invoker.

  Triggers and events have no option to define the `SQL SECURITY` characteristic, so for these objects, `CURRENT_USER()` returns the account for the user who defined the object. To return the invoker, use `USER()` or `SESSION_USER()`.

  The following statements support use of the `CURRENT_USER()` function to take the place of the name of (and, possibly, a host for) an affected user or a definer; in such cases, `CURRENT_USER()` is expanded where and as needed:

  - `DROP USER`
  - `RENAME USER`
  - `GRANT`
  - `REVOKE`
  - `CREATE FUNCTION`
  - `CREATE PROCEDURE`
  - `CREATE TRIGGER`
  - `CREATE EVENT`
  - `CREATE VIEW`
  - `ALTER EVENT`
  - `ALTER VIEW`

- `SET PASSWORD`

For information about the implications that this expansion of `CURRENT_USER()` has for replication, see Section 17.5.1.8, “Replication of CURRENT_USER()”.

- `DATABASE()`

Returns the default (current) database name as a string in the `utf8` character set. If there is no default database, `DATABASE()` returns `NULL`. Within a stored routine, the default database is the database that the routine is associated with, which is not necessarily the same as the database that is the default in the calling context.

```
mysql> SELECT DATABASE();
       -> 'test'
```

If there is no default database, `DATABASE()` returns `NULL`.

- `FOUND_ROWS()`

Note

The `SQL_CALC_FOUND_ROWS` query modifier and accompanying `FOUND_ROWS()` function are deprecated as of MySQL 8.0.17 and will be removed in a future MySQL version. As a replacement, considering executing your query with `LIMIT`, and then a second query with `COUNT(*)` and without `LIMIT` to determine whether there are additional rows. For example, instead of these queries:

```
SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name WHERE id > 100 LIMIT 10;
SELECT FOUND_ROWS();
```

Use these queries instead:

```
SELECT * FROM tbl_name WHERE id > 100 LIMIT 10;
SELECT COUNT(*) WHERE id > 100;
```

`COUNT(*)` is subject to certain optimizations. `SQL_CALC_FOUND_ROWS` causes some optimizations to be disabled.

A `SELECT` statement may include a `LIMIT` clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the `LIMIT`, but without running the statement again. To obtain this row count, include an `SQL_CALC_FOUND_ROWS` option in the `SELECT` statement, and then invoke `FOUND_ROWS()` afterward:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
    -> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

The second `SELECT` returns a number indicating how many rows the first `SELECT` would have returned had it been written without the `LIMIT` clause.

In the absence of the `SQL_CALC_FOUND_ROWS` option in the most recent successful `SELECT` statement, `FOUND_ROWS()` returns the number of rows in the result set returned by that statement. If the statement includes a `LIMIT` clause, `FOUND_ROWS()` returns the number of rows up to the limit. For example, `FOUND_ROWS()` returns 10 or 60, respectively, if the statement includes `LIMIT 10` or `LIMIT 50, 10`.

The row count available through FOUND_ROWS() is transient and not intended to be available past the statement following the SELECT SQL_CALC_FOUND_ROWS statement. If you need to refer to the value later, save it:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;
mysql> SET @rows = FOUND_ROWS();
```

If you are using SELECT SQL_CALC_FOUND_ROWS, MySQL must calculate how many rows are in the full result set. However, this is faster than running the query again without LIMIT, because the result set need not be sent to the client.

SQL_CALC_FOUND_ROWS and FOUND_ROWS() can be useful in situations when you want to restrict the number of rows that a query returns, but also determine the number of rows in the full result set without running the query again. An example is a Web script that presents a paged display containing links to the pages that show other sections of a search result. Using FOUND_ROWS() enables you to determine how many other pages are needed for the rest of the result.

The use of SQL_CALC_FOUND_ROWS and FOUND_ROWS() is more complex for UNION statements than for simple SELECT statements, because LIMIT may occur at multiple places in a UNION. It may be applied to individual SELECT statements in the UNION, or global to the UNION result as a whole.

The intent of SQL_CALC_FOUND_ROWS for UNION is that it should return the row count that would be returned without a global LIMIT. The conditions for use of SQL_CALC_FOUND_ROWS with UNION are:

- The SQL_CALC_FOUND_ROWS keyword must appear in the first SELECT of the UNION.
- The value of FOUND_ROWS() is exact only if UNION ALL is used. If UNION without ALL is used, duplicate removal occurs and the value of FOUND_ROWS() is only approximate.
- If no LIMIT is present in the UNION, SQL_CALC_FOUND_ROWS is ignored and returns the number of rows in the temporary table that is created to process the UNION.

Beyond the cases described here, the behavior of FOUND_ROWS() is undefined (for example, its value following a SELECT statement that fails with an error).

Important

FOUND_ROWS() is not replicated reliably using statement-based replication. This function is automatically replicated using row-based replication.

- ICU_VERSION()

  The version of the International Components for Unicode (ICU) library used to support regular expression operations (see Section 12.5.2, “Regular Expressions”). This function is primarily intended for use in test cases.

- LAST_INSERT_ID(), LAST_INSERT_ID(*expr*)

  With no argument, LAST_INSERT_ID() returns a BIGINT UNSIGNED (64-bit) value representing the first automatically generated value successfully inserted for an AUTO_INCREMENT column as a result of the most recently executed INSERT statement. The value of LAST_INSERT_ID() remains unchanged if no rows are successfully inserted.

  With an argument, LAST_INSERT_ID() returns an unsigned integer.

For example, after inserting a row that generates an AUTO_INCREMENT value, you can get the value like this:

```
mysql> SELECT LAST_INSERT_ID();
        -> 195
```

The currently executing statement does not affect the value of LAST_INSERT_ID(). Suppose that you generate an AUTO_INCREMENT value with one statement, and then refer to LAST_INSERT_ID() in a multiple-row INSERT statement that inserts rows into a table with its own AUTO_INCREMENT column. The value of LAST_INSERT_ID() will remain stable in the second statement; its value for the second and later rows is not affected by the earlier row insertions. (However, if you mix references to LAST_INSERT_ID() and LAST_INSERT_ID(*expr*), the effect is undefined.)

If the previous statement returned an error, the value of LAST_INSERT_ID() is undefined. For transactional tables, if the statement is rolled back due to an error, the value of LAST_INSERT_ID() is left undefined. For manual ROLLBACK, the value of LAST_INSERT_ID() is not restored to that before the transaction; it remains as it was at the point of the ROLLBACK.

Within the body of a stored routine (procedure or function) or a trigger, the value of LAST_INSERT_ID() changes the same way as for statements executed outside the body of these kinds of objects. The effect of a stored routine or trigger upon the value of LAST_INSERT_ID() that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of LAST_INSERT_ID(), the changed value is seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements will not see a changed value.

The ID that was generated is maintained in the server on a *per-connection basis*. This means that the value returned by the function to a given client is the first AUTO_INCREMENT value generated for most recent statement affecting an AUTO_INCREMENT column *by that client*. This value cannot be affected by other clients, even if they generate AUTO_INCREMENT values of their own. This behavior ensures that each client can retrieve its own ID without concern for the activity of other clients, and without the need for locks or transactions.

The value of LAST_INSERT_ID() is not changed if you set the AUTO_INCREMENT column of a row to a non-“magic” value (that is, a value that is not NULL and not 0).

Important

If you insert multiple rows using a single INSERT statement, LAST_INSERT_ID() returns the value generated for the *first* inserted row *only*. The reason for this is to make it possible to reproduce easily the same INSERT statement against some other server.

For example:

```
mysql> USE test;

mysql> CREATE TABLE t (
       id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
       name VARCHAR(10) NOT NULL
       );
```

```
mysql> INSERT INTO t VALUES (NULL, 'Bob');

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
+----+------+

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+

mysql> INSERT INTO t VALUES
       (NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
|  2 | Mary |
|  3 | Jane |
|  4 | Lisa |
+----+------+

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                2 |
+------------------+
```

Although the second `INSERT` statement inserted three new rows into `t`, the ID generated for the first of these rows was `2`, and it is this value that is returned by `LAST_INSERT_ID()` for the following `SELECT` statement.

If you use `INSERT IGNORE` and the row is ignored, the `LAST_INSERT_ID()` remains unchanged from the current value (or 0 is returned if the connection has not yet performed a successful `INSERT`) and, for non-transactional tables, the `AUTO_INCREMENT` counter is not incremented. For `InnoDB` tables, the `AUTO_INCREMENT` counter is incremented if `innodb_autoinc_lock_mode` is set to `1` or `2`, as demonstrated in the following example:

```
mysql> USE test;

mysql> SELECT @@innodb_autoinc_lock_mode;
+----------------------------+
| @@innodb_autoinc_lock_mode |
+----------------------------+
|                          1 |
+----------------------------+

mysql> CREATE TABLE `t` (
       `id` INT(11) NOT NULL AUTO_INCREMENT,
       `val` INT(11) DEFAULT NULL,
       PRIMARY KEY (`id`),
       UNIQUE KEY `i1` (`val`)
       ) ENGINE=InnoDB DEFAULT CHARSET=latin1;

# Insert two rows
```

```
mysql> INSERT INTO t (val) VALUES (1),(2);

# With auto_increment_offset=1, the inserted rows
# result in an AUTO_INCREMENT value of 3

mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `val` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `i1` (`val`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1

# LAST_INSERT_ID() returns the first automatically generated
# value that is successfully inserted for the AUTO_INCREMENT column

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+

# The attempted insertion of duplicate rows fail but errors are ignored

mysql> INSERT IGNORE INTO t (val) VALUES (1),(2);
Query OK, 0 rows affected (0.00 sec)
Records: 2  Duplicates: 2  Warnings: 0

# With innodb_autoinc_lock_mode=1, the AUTO_INCREMENT counter
# is incremented for the ignored rows

mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `val` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `i1` (`val`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1

# The LAST_INSERT_ID is unchanged because the previous insert was unsuccessful

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+
```

For more information, see Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”.

If *`expr`* is given as an argument to `LAST_INSERT_ID()`, the value of the argument is returned by the function and is remembered as the next value to be returned by `LAST_INSERT_ID()`. This can be used to simulate sequences:

1. Create a table to hold the sequence counter and initialize it:

```
mysql> CREATE TABLE sequence (id INT NOT NULL);
mysql> INSERT INTO sequence VALUES (0);
```

2. Use the table to generate sequence numbers like this:

```
mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
mysql> SELECT LAST_INSERT_ID();
```

The `UPDATE` statement increments the sequence counter and causes the next call to `LAST_INSERT_ID()` to return the updated value. The `SELECT` statement retrieves that value. The `mysql_insert_id()` C API function can also be used to get the value. See Section 28.7.7.38, “mysql_insert_id()”.

You can generate sequences without calling `LAST_INSERT_ID()`, but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. It is multi-user safe because multiple clients can issue the `UPDATE` statement and get their own sequence value with the `SELECT` statement (or `mysql_insert_id()`), without affecting or being affected by other clients that generate their own sequence values.

Note that `mysql_insert_id()` is only updated after `INSERT` and `UPDATE` statements, so you cannot use the C API function to retrieve the value for `LAST_INSERT_ID(`*`expr`*`)` after executing other SQL statements like `SELECT` or `SET`.

- `ROLES_GRAPHML()`

  Returns a `utf8` string containing a GraphML document representing memory role subgraphs. The `ROLE_ADMIN` or `SUPER` privilege is required to see content in the `<graphml>` element. Otherwise, the result shows only an empty element:

```
mysql> SELECT ROLES_GRAPHML();
+-----------------------------------------------------+
| ROLES_GRAPHML()                                     |
+-----------------------------------------------------+
| <?xml version="1.0" encoding="UTF-8"?><graphml /> |
+-----------------------------------------------------+
```

- `ROW_COUNT()`

  `ROW_COUNT()` returns a value as follows:

  - DDL statements: 0. This applies to statements such as `CREATE TABLE` or `DROP TABLE`.
  - DML statements other than `SELECT`: The number of affected rows. This applies to statements such as `UPDATE`, `INSERT`, or `DELETE` (as before), but now also to statements such as `ALTER TABLE` and `LOAD DATA`.
  - `SELECT`: -1 if the statement returns a result set, or the number of rows “affected” if it does not. For example, for `SELECT * FROM t1`, `ROW_COUNT()` returns -1. For `SELECT * FROM t1 INTO OUTFILE '`*`file_name`*`'`, `ROW_COUNT()` returns the number of rows written to the file.
  - `SIGNAL` statements: 0.

  For `UPDATE` statements, the affected-rows value by default is the number of rows actually changed. If you specify the `CLIENT_FOUND_ROWS` flag to `mysql_real_connect()` when connecting to `mysqld`, the affected-rows value is the number of rows “found”; that is, matched by the `WHERE` clause.

  For `REPLACE` statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

For `INSERT ... ON DUPLICATE KEY UPDATE` statements, the affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the `CLIENT_FOUND_ROWS` flag, the affected-rows value is 1 (not 0) if an existing row is set to its current values.

The `ROW_COUNT()` value is similar to the value from the `mysql_affected_rows()` C API function and the row count that the `mysql` client displays following statement execution.

```
mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           3 |
+-------------+
1 row in set (0.00 sec)

mysql> DELETE FROM t WHERE i IN(1,2);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           2 |
+-------------+
1 row in set (0.00 sec)
```

Important

`ROW_COUNT()` is not replicated reliably using statement-based replication. This function is automatically replicated using row-based replication.

- `SCHEMA()`

  This function is a synonym for `DATABASE()`.

- `SESSION_USER()`

  `SESSION_USER()` is a synonym for `USER()`.

- `SYSTEM_USER()`

  `SYSTEM_USER()` is a synonym for `USER()`.

Note

The `SYSTEM_USER()` function is distinct from the `SYSTEM_USER` privilege. The former returns the current MySQL account name. The latter distinguishes the system user and regular user account categories (see Section 6.2.11, “Account Categories”).

- `USER()`

  Returns the current MySQL user name and host name as a string in the `utf8` character set.

```
mysql> SELECT USER();
        -> 'davida@localhost'
```

The value indicates the user name you specified when connecting to the server, and the client host from which you connected. The value can be different from that of `CURRENT_USER()`.

- `VERSION()`

  Returns a string that indicates the MySQL server version. The string uses the `utf8` character set. The value might have a suffix in addition to the version number. See the description of the `version` system variable in Section 5.1.8, “Server System Variables”.

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

```
mysql> SELECT VERSION();
        -> '8.0.19-standard'
```

12.16 Spatial Analysis Functions

MySQL provides functions to perform various operations on spatial data. These functions can be grouped into several major categories according to the type of operation they perform:

- Functions that create geometries in various formats (WKT, WKB, internal)
- Functions that convert geometries between formats
- Functions that access qualitative or quantitative properties of a geometry
- Functions that describe relations between two geometries
- Functions that create new geometries from existing ones

For general background about MySQL support for using spatial data, see Section 11.5, “Spatial Data Types”.

12.16.1 Spatial Function Reference

The following table lists each spatial function and provides a short description of each one.

Table 12.20 Spatial Functions

Name	Description
`GeomCollection()`	Construct geometry collection from geometries
`GeometryCollection()`	Construct geometry collection from geometries
`LineString()`	Construct LineString from Point values
`MBRContains()`	Whether MBR of one geometry contains MBR of another
`MBRCoveredBy()`	Whether one MBR is covered by another
`MBRCovers()`	Whether one MBR covers another
`MBRDisjoint()`	Whether MBRs of two geometries are disjoint
`MBREquals()`	Whether MBRs of two geometries are equal
`MBRIntersects()`	Whether MBRs of two geometries intersect
`MBROverlaps()`	Whether MBRs of two geometries overlap
`MBRTouches()`	Whether MBRs of two geometries touch

Name	Description
MBRWithin()	Whether MBR of one geometry is within MBR of another
MultiLineString()	Contruct MultiLineString from LineString values
MultiPoint()	Construct MultiPoint from Point values
MultiPolygon()	Construct MultiPolygon from Polygon values
Point()	Construct Point from coordinates
Polygon()	Construct Polygon from LineString arguments
ST_Area()	Return Polygon or MultiPolygon area
ST_AsBinary(), ST_AsWKB()	Convert from internal geometry format to WKB
ST_AsGeoJSON()	Generate GeoJSON object from geometry
ST_AsText(), ST_AsWKT()	Convert from internal geometry format to WKT
ST_Buffer()	Return geometry of points within given distance from geometry
ST_Buffer_Strategy()	Produce strategy option for ST_Buffer()
ST_Centroid()	Return centroid as a point
ST_Contains()	Whether one geometry contains another
ST_ConvexHull()	Return convex hull of geometry
ST_Crosses()	Whether one geometry crosses another
ST_Difference()	Return point set difference of two geometries
ST_Dimension()	Dimension of geometry
ST_Disjoint()	Whether one geometry is disjoint from another
ST_Distance()	The distance of one geometry from another
ST_Distance_Sphere()	Minimum distance on earth between two geometries
ST_EndPoint()	End Point of LineString
ST_Envelope()	Return MBR of geometry
ST_Equals()	Whether one geometry is equal to another
ST_ExteriorRing()	Return exterior ring of Polygon
ST_GeoHash()	Produce a geohash value
ST_GeomCollFromText(), ST_GeometryCollectionFromText(), ST_GeomCollFromTxt()	Return geometry collection from WKT
ST_GeomCollFromWKB(), ST_GeometryCollectionFromWKB()	Return geometry collection from WKB
ST_GeometryN()	Return N-th geometry from geometry collection
ST_GeometryType()	Return name of geometry type
ST_GeomFromGeoJSON()	Generate geometry from GeoJSON object
ST_GeomFromText(), ST_GeometryFromText()	Return geometry from WKT
ST_GeomFromWKB(), ST_GeometryFromWKB()	Return geometry from WKB
ST_InteriorRingN()	Return N-th interior ring of Polygon

Name	Description
ST_Intersection()	Return point set intersection of two geometries
ST_Intersects()	Whether one geometry intersects another
ST_IsClosed()	Whether a geometry is closed and simple
ST_IsEmpty()	Placeholder function
ST_IsSimple()	Whether a geometry is simple
ST_IsValid()	Whether a geometry is valid
ST_LatFromGeoHash()	Return latitude from geohash value
ST_Latitude()	Return latitude of Point
ST_Length()	Return length of LineString
ST_LineFromText(), ST_LineStringFromText()	Construct LineString from WKT
ST_LineFromWKB(), ST_LineStringFromWKB()	Construct LineString from WKB
ST_LongFromGeoHash()	Return longitude from geohash value
ST_Longitude()	Return longitude of Point
ST_MakeEnvelope()	Rectangle around two points
ST_MLineFromText(), ST_MultiLineStringFromText()	Construct MultiLineString from WKT
ST_MLineFromWKB(), ST_MultiLineStringFromWKB()	Construct MultiLineString from WKB
ST_MPointFromText(), ST_MultiPointFromText()	Construct MultiPoint from WKT
ST_MPointFromWKB(), ST_MultiPointFromWKB()	Construct MultiPoint from WKB
ST_MPolyFromText(), ST_MultiPolygonFromText()	Construct MultiPolygon from WKT
ST_MPolyFromWKB(), ST_MultiPolygonFromWKB()	Construct MultiPolygon from WKB
ST_NumGeometries()	Return number of geometries in geometry collection
ST_NumInteriorRing(), ST_NumInteriorRings()	Return number of interior rings in Polygon
ST_NumPoints()	Return number of points in LineString
ST_Overlaps()	Whether one geometry overlaps another
ST_PointFromGeoHash()	Convert geohash value to POINT value
ST_PointFromText()	Construct Point from WKT
ST_PointFromWKB()	Construct Point from WKB
ST_PointN()	Return N-th point from LineString
ST_PolyFromText(), ST_PolygonFromText()	Construct Polygon from WKT
ST_PolyFromWKB(), ST_PolygonFromWKB()	Construct Polygon from WKB

Name	Description
ST_Simplify()	Return simplified geometry
ST_SRID()	Return spatial reference system ID for geometry
ST_StartPoint()	Start Point of LineString
ST_SwapXY()	Return argument with X/Y coordinates swapped
ST_SymDifference()	Return point set symmetric difference of two geometries
ST_Touches()	Whether one geometry touches another
ST_Transform()	Transform coordinates of geometry
ST_Union()	Return point set union of two geometries
ST_Validate()	Return validated geometry
ST_Within()	Whether one geometry is within another
ST_X()	Return X coordinate of Point
ST_Y()	Return Y coordinate of Point

12.16.2 Argument Handling by Spatial Functions

Spatial values, or geometries, have the properties described at Section 11.5.2.2, “Geometry Class”. The following discussion lists general spatial function argument-handling characteristics. Specific functions or groups of functions may have additional argument-handling characteristics, as discussed in the sections where those function descriptions occur.

Spatial functions are defined only for valid geometry values.

The spatial reference identifier (SRID) of a geometry identifies the coordinate space in which the geometry is defined. In MySQL, the SRID value is an integer associated with the geometry value. The maximum usable SRID value is $2^{32}-1$. If a larger value is given, only the lower 32 bits are used.

SRID 0 represents an infinite flat Cartesian plane with no units assigned to its axes. To ensure SRID 0 behavior, create geometry values using SRID 0. SRID 0 is the default for new geometry values if no SRID is specified.

Geometry values produced by any spatial function inherit the SRID of the geometry arguments.

Spatial functions that take multiple geometry arguments require those arguments to have the same SRID value (that is, same value in the lower 32 bits). Assuming equal SRIDs, spatial functions do nothing with them after performing the equality check; geometry values are implicitly handled using Cartesian coordinates (SRID 0). If a spatial function returns ER_GIS_DIFFERENT_SRIDS, it means that the geometry arguments did not all have the same SRID. You must modify them to have the same SRID.

The Open Geospatial Consortium guidelines require that input polygons already be closed, so unclosed polygons are rejected as invalid rather than being closed.

Empty geometry-collection handling is as follows: An empty WKT input geometry collection may be specified as 'GEOMETRYCOLLECTION()'. This is also the output WKT resulting from a spatial operation that produces an empty geometry collection.

During parsing of a nested geometry collection, the collection is flattened and its basic components are used in various GIS operations to compute results. This provides additional flexibility to users because it is unnecessary to be concerned about the uniqueness of geometry data. Nested geometry collections may be produced from nested GIS function calls without having to be explicitly flattened first.

12.16.3 Functions That Create Geometry Values from WKT Values

These functions take as arguments a Well-Known Text (WKT) representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry. For a description of WKT format, see Well-Known Text (WKT) Format.

Functions in this section detect arguments in either Cartesian or geographic spatial reference systems (SRSs), and return results appropriate to the SRS.

`ST_GeomFromText()` accepts a WKT value of any geometry type as its first argument. Other functions provide type-specific construction functions for construction of geometry values of each geometry type.

Functions such as `ST_MPointFromText()` and `ST_GeomFromText()` that accept WKT-format representations of `MultiPoint` values permit individual points within values to be surrounded by parentheses. For example, both of the following function calls are valid:

```
ST_MPointFromText('MULTIPOINT (1 1, 2 2, 3 3)')
ST_MPointFromText('MULTIPOINT ((1 1), (2 2), (3 3))')
```

Functions such as `ST_GeomFromText()` that accept WKT geometry collection arguments understand both OpenGIS `'GEOMETRYCOLLECTION EMPTY'` standard syntax and MySQL `'GEOMETRYCOLLECTION()'` nonstandard syntax. Functions such as `ST_AsWKT()` that produce WKT values produce `'GEOMETRYCOLLECTION EMPTY'` standard syntax:

```
mysql> SET @s1 = ST_GeomFromText('GEOMETRYCOLLECTION()');
mysql> SET @s2 = ST_GeomFromText('GEOMETRYCOLLECTION EMPTY');
mysql> SELECT ST_AsWKT(@s1), ST_AsWKT(@s2);
+--------------------------+--------------------------+
| ST_AsWKT(@s1)            | ST_AsWKT(@s2)            |
+--------------------------+--------------------------+
| GEOMETRYCOLLECTION EMPTY | GEOMETRYCOLLECTION EMPTY |
+--------------------------+--------------------------+
```

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any geometry argument is `NULL` or is not a syntactically well-formed geometry, or if the SRID argument is `NULL`, the return value is `NULL`.
- By default, geographic coordinates (latitude, longitude) are interpreted as in the order specified by the spatial reference system of geometry arguments. An optional *`options`* argument may be given to override the default axis order. `options` consists of a list of comma-separated *`key=value`*. The onlypermitted *`key`* value is `axis-order`, with permitted values of `lat-long`, `long-lat` and `srid-defined` (the default).

  If the *`options`* argument is `NULL`, the return value is `NULL`. If the *`options`* argument is invalid, an error occurs to indicate why.
- If an SRID argument refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:
  - If a longitude value is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.
  - If a latitude value is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

These functions are available for creating geometries from WKT values:

- `ST_GeomCollFromText(wkt[, srid [, options]])`, `ST_GeometryCollectionFromText(wkt[, srid [, options]])`, `ST_GeomCollFromTxt(wkt[, srid [, options]])`

  Constructs a `GeometryCollection` value using its WKT representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

```
mysql> SET @g = "MULTILINESTRING((10 10, 11 11), (9 9, 10 10))";
mysql> SELECT ST_AsText(ST_GeomCollFromText(@g));
+---------------------------------------------+
| ST_AsText(ST_GeomCollFromText(@g))          |
+---------------------------------------------+
| MULTILINESTRING((10 10,11 11),(9 9,10 10)) |
+---------------------------------------------+
```

- `ST_GeomFromText(wkt[, srid [, options]])`, `ST_GeometryFromText(wkt[, srid [, options]])`

  Constructs a geometry value of any type using its WKT representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_LineFromText(wkt[, srid [, options]])`, `ST_LineStringFromText(wkt[, srid [, options]])`

  Constructs a `LineString` value using its WKT representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_MLineFromText(wkt[, srid [, options]])`, `ST_MultiLineStringFromText(wkt[, srid [, options]])`

  Constructs a `MultiLineString` value using its WKT representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_MPointFromText(wkt[, srid [, options]])`, `ST_MultiPointFromText(wkt[, srid [, options]])`

  Constructs a `MultiPoint` value using its WKT representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_MPolyFromText(wkt[, srid [, options]])`, `ST_MultiPolygonFromText(wkt[, srid [, options]])`

  Constructs a `MultiPolygon` value using its WKT representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_PointFromText(wkt[, srid [, options]])`

Constructs a `Point` value using its WKT representation and SRID.

`ST_PointFromText()` handles its arguments as described in the introduction to this section.

- `ST_PolyFromText(wkt[, srid [, options]])`, `ST_PolygonFromText(wkt[, srid [, options]])`

  Constructs a `Polygon` value using its WKT representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

12.16.4 Functions That Create Geometry Values from WKB Values

These functions take as arguments a `BLOB` containing a Well-Known Binary (WKB) representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry. For a description of WKB format, see Well-Known Binary (WKB) Format.

Functions in this section detect arguments in either Cartesian or geographic spatial reference systems (SRSs), and return results appropriate to the SRS.

`ST_GeomFromWKB()` accepts a WKB value of any geometry type as its first argument. Other functions provide type-specific construction functions for construction of geometry values of each geometry type.

Prior to MySQL 8.0, these functions also accepted geometry objects as returned by the functions in Section 12.16.5, “MySQL-Specific Functions That Create Geometry Values”. Geometry arguments are no longer permitted and produce an error. To migrate calls from using geometry arguments to using WKB arguments, follow these guidelines:

- Rewrite constructs such as `ST_GeomFromWKB(Point(0, 0))` as `Point(0, 0)`.
- Rewrite constructs such as `ST_GeomFromWKB(Point(0, 0), 4326)` as `ST_SRID(Point(0, 0), 4326)` or `ST_GeomFromWKB(ST_AsWKB(Point(0, 0)), 4326)`.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If the WKB or SRID argument is `NULL`, the return value is `NULL`.
- By default, geographic coordinates (latitude, longitude) are interpreted as in the order specified by the spatial reference system of geometry arguments. An optional *`options`* argument may be given to override the default axis order. `options` consists of a list of comma-separated *`key=value`*. The onlypermitted *`key`* value is `axis-order`, with permitted values of `lat-long`, `long-lat` and `srid-defined` (the default).

  If the *`options`* argument is `NULL`, the return value is `NULL`. If the *`options`* argument is invalid, an error occurs to indicate why.
- If an SRID argument refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:
  - If a longitude value is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.
  - If a latitude value is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

These functions are available for creating geometries from WKB values:

- `ST_GeomCollFromWKB(wkb[, srid [, options]])`, `ST_GeometryCollectionFromWKB(wkb[, srid [, options]])`

  Constructs a `GeometryCollection` value using its WKB representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_GeomFromWKB(wkb[, srid [, options]])`, `ST_GeometryFromWKB(wkb[, srid [, options]])`

  Constructs a geometry value of any type using its WKB representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_LineFromWKB(wkb[, srid [, options]])`, `ST_LineStringFromWKB(wkb[, srid [, options]])`

  Constructs a `LineString` value using its WKB representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_MLineFromWKB(wkb[, srid [, options]])`, `ST_MultiLineStringFromWKB(wkb[, srid [, options]])`

  Constructs a `MultiLineString` value using its WKB representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_MPointFromWKB(wkb[, srid [, options]])`, `ST_MultiPointFromWKB(wkb[, srid [, options]])`

  Constructs a `MultiPoint` value using its WKB representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_MPolyFromWKB(wkb[, srid [, options]])`, `ST_MultiPolygonFromWKB(wkb[, srid [, options]])`

  Constructs a `MultiPolygon` value using its WKB representation and SRID.

  These functions handle their arguments as described in the introduction to this section.

- `ST_PointFromWKB(wkb[, srid [, options]])`

  Constructs a `Point` value using its WKB representation and SRID.

  `ST_PointFromWKB()` handles its arguments as described in the introduction to this section.

- `ST_PolyFromWKB(wkb[, srid [, options]])`, `ST_PolygonFromWKB(wkb[, srid [, options]])`

  Constructs a `Polygon` value using its WKB representation and SRID.

These functions handle their arguments as described in the introduction to this section.

12.16.5 MySQL-Specific Functions That Create Geometry Values

MySQL provides a set of useful nonstandard functions for creating geometry values. The functions described in this section are MySQL extensions to the OpenGIS specification.

These functions produce geometry objects from either WKB values or geometry objects as arguments. If any argument is not a proper WKB or geometry representation of the proper object type, the return value is `NULL`.

For example, you can insert the geometry return value from `Point()` directly into a `POINT` column:

```
INSERT INTO t1 (pt_col) VALUES(Point(1,2));
```

- `GeomCollection(g [, g] ...)`

  Constructs a `GeomCollection` value from the geometry arguments.

  `GeomCollection()` returns all the proper geometries contained in the arguments even if a nonsupported geometry is present.

  `GeomCollection()` with no arguments is permitted as a way to create an empty geometry. Also, functions such as `ST_GeomFromText()` that accept WKT geometry collection arguments understand both OpenGIS `'GEOMETRYCOLLECTION EMPTY'` standard syntax and MySQL `'GEOMETRYCOLLECTION()'` nonstandard syntax.

  `GeomCollection()` and `GeometryCollection()` are synonymous, with `GeomCollection()` the preferred function.

- `GeometryCollection(g [, g] ...)`

  Constructs a `GeomCollection` value from the geometry arguments.

  `GeometryCollection()` returns all the proper geometries contained in the arguments even if a nonsupported geometry is present.

  `GeometryCollection()` with no arguments is permitted as a way to create an empty geometry. Also, functions such as `ST_GeomFromText()` that accept WKT geometry collection arguments understand both OpenGIS `'GEOMETRYCOLLECTION EMPTY'` standard syntax and MySQL `'GEOMETRYCOLLECTION()'` nonstandard syntax.

  `GeomCollection()` and `GeometryCollection()` are synonymous, with `GeomCollection()` the preferred function.

- `LineString(pt [, pt] ...)`

  Constructs a `LineString` value from a number of `Point` or WKB `Point` arguments. If the number of arguments is less than two, the return value is `NULL`.

- `MultiLineString(ls [, ls] ...)`

  Constructs a `MultiLineString` value using `LineString` or WKB `LineString` arguments.

- `MultiPoint(pt [, pt2] ...)`

  Constructs a `MultiPoint` value using `Point` or WKB `Point` arguments.

- `MultiPolygon(poly [, poly] ...)`

  Constructs a `MultiPolygon` value from a set of `Polygon` or WKB `Polygon` arguments.

- `Point(x, y)`

  Constructs a `Point` using its coordinates.

- `Polygon(ls [, ls] ...)`

  Constructs a `Polygon` value from a number of `LineString` or WKB `LineString` arguments. If any argument does not represent a `LinearRing` (that is, not a closed and simple `LineString`), the return value is `NULL`.

12.16.6 Geometry Format Conversion Functions

MySQL supports the functions listed in this section for converting geometry values from internal geometry format to WKT or WKB format, or for swapping the order of X and Y coordinates.

There are also functions to convert a string from WKT or WKB format to internal geometry format. See Section 12.16.3, “Functions That Create Geometry Values from WKT Values”, and Section 12.16.4, “Functions That Create Geometry Values from WKB Values”.

Functions such as `ST_GeomFromText()` that accept WKT geometry collection arguments understand both OpenGIS `'GEOMETRYCOLLECTION EMPTY'` standard syntax and MySQL `'GEOMETRYCOLLECTION()'` nonstandard syntax. Another way to produce an empty geometry collection is by calling `GeometryCollection()` with no arguments. Functions such as `ST_AsWKT()` that produce WKT values produce `'GEOMETRYCOLLECTION EMPTY'` standard syntax:

```
mysql> SET @s1 = ST_GeomFromText('GEOMETRYCOLLECTION()');
mysql> SET @s2 = ST_GeomFromText('GEOMETRYCOLLECTION EMPTY');
mysql> SELECT ST_AsWKT(@s1), ST_AsWKT(@s2);
+-------------------------+-------------------------+
| ST_AsWKT(@s1)           | ST_AsWKT(@s2)           |
+-------------------------+-------------------------+
| GEOMETRYCOLLECTION EMPTY | GEOMETRYCOLLECTION EMPTY |
+-------------------------+-------------------------+
mysql> SELECT ST_AsWKT(GeomCollection());
+---------------------------+
| ST_AsWKT(GeomCollection()) |
+---------------------------+
| GEOMETRYCOLLECTION EMPTY   |
+---------------------------+
```

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL`, the return value is `NULL`.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument is in an undefined spatial reference system, the axes are output in the order they appear in the geometry and an `ER_WARN_SRS_NOT_FOUND_AXIS_ORDER` warning occurs.
- By default, geographic coordinates (latitude, longitude) are interpreted as in the order specified by the spatial reference system of geometry arguments. An optional *`options`* argument may be given to override the default axis order. `options` consists of a list of comma-separated *`key=value`*. The onlypermitted *`key`* value is `axis-order`, with permitted values of `lat-long`, `long-lat` and `srid-defined` (the default).

If the *options* argument is NULL, the return value is NULL. If the *options* argument is invalid, an error occurs to indicate why.

- Otherwise, the return value is non-NULL.

These functions are available for format conversions or coordinate swapping:

- ST_AsBinary(*g* [, *options*]), ST_AsWKB(*g* [, *options*])

  Converts a value in internal geometry format to its WKB representation and returns the binary result.

  The function return value has geographic coordinates (latitude, longitude) in the order specified by the spatial reference system that applies to the geometry argument. An optional *options* argument may be given to override the default axis order.

  ST_AsBinary() and ST_AsWKB() handle their arguments as described in the introduction to this section.

```
mysql> SET @g = ST_LineFromText('LINESTRING(0 5,5 10,10 15)', 4326);
mysql> SELECT ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g)));
+---------------------------------------+
| ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g))) |
+---------------------------------------+
| LINESTRING(5 0,10 5,15 10)              |
+---------------------------------------+
mysql> SELECT ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g, 'axis-order=long-lat')));
+-------------------------------------------------------------+
| ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g, 'axis-order=long-lat'))) |
+-------------------------------------------------------------+
| LINESTRING(0 5,5 10,10 15)                                  |
+-------------------------------------------------------------+
mysql> SELECT ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g, 'axis-order=lat-long')));
+-------------------------------------------------------------+
| ST_AsText(ST_GeomFromWKB(ST_AsWKB(@g, 'axis-order=lat-long'))) |
+-------------------------------------------------------------+
| LINESTRING(5 0,10 5,15 10)                                  |
+-------------------------------------------------------------+
```

- ST_AsText(*g* [, *options*]), ST_AsWKT(*g* [, *options*])

  Converts a value in internal geometry format to its WKT representation and returns the string result.

  The function return value has geographic coordinates (latitude, longitude) in the order specified by the spatial reference system that applies to the geometry argument. An optional *options* argument may be given to override the default axis order.

  ST_AsText() and ST_AsWKT() handle their arguments as described in the introduction to this section.

```
mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_GeomFromText(@g));
+------------------------------+
| ST_AsText(ST_GeomFromText(@g)) |
+------------------------------+
| LINESTRING(1 1,2 2,3 3)        |
+------------------------------+
```

  Output for MultiPoint values includes parentheses around each point. For example:

```
```

```
mysql> SELECT ST_AsText(ST_GeomFromText(@mp));
+--------------------------------+
| ST_AsText(ST_GeomFromText(@mp)) |
+--------------------------------+
| MULTIPOINT((1 1),(2 2),(3 3))   |
+--------------------------------+
```

- `ST_SwapXY(g)`

  Accepts an argument in internal geometry format, swaps the X and Y values of each coordinate pair within the geometry, and returns the result.

  `ST_SwapXY()` handles its arguments as described in the introduction to this section.

```
mysql> SET @g = ST_LineFromText('LINESTRING(0 5,5 10,10 15)');
mysql> SELECT ST_AsText(@g);
+---------------------------+
| ST_AsText(@g)             |
+---------------------------+
| LINESTRING(0 5,5 10,10 15) |
+---------------------------+
mysql> SELECT ST_AsText(ST_SwapXY(@g));
+---------------------------+
| ST_AsText(ST_SwapXY(@g))   |
+---------------------------+
| LINESTRING(5 0,10 5,15 10) |
+---------------------------+
```

12.16.7 Geometry Property Functions

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such functions return `NULL` if the argument is of an incorrect geometry type. For example, the `ST_Area()` polygon function returns `NULL` if the object type is neither `Polygon` nor `MultiPolygon`.

12.16.7.1 General Geometry Property Functions

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL`, the return value is `NULL`.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument has an SRID value that refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- If any SRID argument is not within the range of a 32-bit unsigned integer, an `ER_DATA_OUT_OF_RANGE` error occurs.
- If any SRID argument refers to an undefined SRS, an `ER_SRS_NOT_FOUND` error occurs.
- Otherwise, the return value is non-`NULL`.

These functions are available for obtaining geometry properties:

- `ST_Dimension(g)`

Returns the inherent dimension of the geometry value *g*. The dimension can be −1, 0, 1, or 2. The meaning of these values is given in Section 11.5.2.2, “Geometry Class”.

`ST_Dimension()` handles its arguments as described in the introduction to this section.

```
mysql> SELECT ST_Dimension(ST_GeomFromText('LineString(1 1,2 2)'));
+------------------------------------------------------+
| ST_Dimension(ST_GeomFromText('LineString(1 1,2 2)')) |
+------------------------------------------------------+
|                                                    1 |
+------------------------------------------------------+
```

- `ST_Envelope(g)`

  Returns the minimum bounding rectangle (MBR) for the geometry value *g*. The result is returned as a `Polygon` value that is defined by the corner points of the bounding box:

  ```
  POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
  ```

  ```
  mysql> SELECT ST_AsText(ST_Envelope(ST_GeomFromText('LineString(1 1,2 2)')));
  +---------------------------------------------------------------+
  | ST_AsText(ST_Envelope(ST_GeomFromText('LineString(1 1,2 2)'))) |
  +---------------------------------------------------------------+
  | POLYGON((1 1,2 1,2 2,1 2,1 1))                                |
  +---------------------------------------------------------------+
  ```

  If the argument is a point or a vertical or horizontal line segment, `ST_Envelope()` returns the point or the line segment as its MBR rather than returning an invalid polygon:

  ```
  mysql> SELECT ST_AsText(ST_Envelope(ST_GeomFromText('LineString(1 1,1 2)')));
  +---------------------------------------------------------------+
  | ST_AsText(ST_Envelope(ST_GeomFromText('LineString(1 1,1 2)'))) |
  +---------------------------------------------------------------+
  | LINESTRING(1 1,1 2)                                           |
  +---------------------------------------------------------------+
  ```

  `ST_Envelope()` handles its arguments as described in the introduction to this section, with this exception:

  - If the geometry has an SRID value for a geographic spatial reference system (SRS), an `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` error occurs.

- `ST_GeometryType(g)`

  Returns a binary string indicating the name of the geometry type of which the geometry instance *g* is a member. The name corresponds to one of the instantiable `Geometry` subclasses.

  `ST_GeometryType()` handles its arguments as described in the introduction to this section.

  ```
  mysql> SELECT ST_GeometryType(ST_GeomFromText('POINT(1 1)'));
  +---------------------------------------------+
  | ST_GeometryType(ST_GeomFromText('POINT(1 1)')) |
  +---------------------------------------------+
  | POINT                                       |
  +---------------------------------------------+
  ```

- `ST_IsEmpty(g)`

This function is a placeholder that returns 1 for an empty geometry collection value or 0 otherwise.

The only valid empty geometry is represented in the form of an empty geometry collection value. MySQL does not support GIS `EMPTY` values such as `POINT EMPTY`.

`ST_IsEmpty()` handles its arguments as described in the introduction to this section.

- `ST_IsSimple(g)`

  Returns 1 if the geometry value *g* is simple according to the ISO *SQL/MM Part 3: Spatial* standard. `ST_IsSimple()` returns 0 if the argument is not simple.

  The descriptions of the instantiable geometric classes given under Section 11.5.2, “The OpenGIS Geometry Model” include the specific conditions that cause class instances to be classified as not simple.

  `ST_IsSimple()` handles its arguments as described in the introduction to this section, with this exception:

  - If the geometry has a geographic SRS with a longitude or latitude that is out of range, an error occurs:
    - If any longitude argument is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.
    - If any latitude argument is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

    Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

- `ST_SRID(g[, srid])`

  With a single argument representing a valid geometry object *g*, `ST_SRID()` returns an integer indicating the ID of the spatial reference system (SRS) associated with *g*.

  With the optional second argument representing a valid SRID value, `ST_SRID()` returns an object with the same type as its first argument with an SRID value equal to the second argument. This only sets the SRID value of the object; it does not perform any transformation of coordinate values.

  `ST_SRID()` handles its arguments as described in the introduction to this section, with this exception:

  - For the single-argument syntax, `ST_SRID()` returns the geometry SRID even if it refers to an undefined SRS. An `ER_SRS_NOT_FOUND` error does not occur.

  `ST_SRID(g, target_srid)` and `ST_Transform(g, target_srid)` differ as follows:

  - `ST_SRID()` changes the geometry SRID value without transforming its coordinates.
  - `ST_Transform()` transforms the geometry coordinates in addition to changing its SRID value.

```
mysql> SET @g = ST_GeomFromText('LineString(1 1,2 2)', 0);
mysql> SELECT ST_SRID(@g);
+-------------+
| ST_SRID(@g) |
+-------------+
|           0 |
+-------------+
mysql> SET @g = ST_SRID(@g, 4326);
```

```
mysql> SELECT ST_SRID(@g);
+-------------+
| ST_SRID(@g) |
+-------------+
|        4326 |
+-------------+
```

It is possible to create a geometry in a particular SRID by passing to `ST_SRID()` the result of one of the MySQL-specific functions for creating spatial values, along with an SRID value. For example:

```
SET @g1 = ST_SRID(Point(1, 1), 4326);
```

However, that method creates the geometry in SRID 0, then casts it to SRID 4326 (WGS 84). A preferable alternative is to create the geometry with the correct spatial reference system to begin with. For example:

```
SET @g1 = ST_PointFromText('POINT(1 1)', 4326);
SET @g1 = ST_GeomFromText('POINT(1 1)', 4326);
```

The two-argument form of `ST_SRID()` is useful for tasks such as correcting or changing the SRS of geometries that have an incorrect SRID.

12.16.7.2 Point Property Functions

A `Point` consists of X and Y coordinates, which may be obtained using the `ST_X()` and `ST_Y()` functions, respectively. These functions also permit an optional second argument that specifies an X or Y coordinate value, in which case the function result is the `Point` object from the first argument with the appropriate coordinate modified to be equal to the second argument.

For `Point` objects that have a geographic spatial reference system (SRS), the longitude and latitude may be obtained using the `ST_Longitude()` and `ST_Latitude()` functions, respectively. These functions also permit an optional second argument that specifies a longitude or latitude value, in which case the function result is the `Point` object from the first argument with the longitude or latitude modified to be equal to the second argument.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL`, the return value is `NULL`.
- If any geometry argument is a valid geometry but not a `Point` object, an `ER_UNEXPECTED_GEOMETRY_TYPE` error occurs.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument has an SRID value that refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- If an X or Y coordinate argument is provided and the value is `-inf`, `+inf`, or `NaN`, an `ER_DATA_OUT_OF_RANGE` error occurs.
- If a longitude or latitude argument is out of range, an error occurs:
  - If any longitude argument is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.
  - If any latitude argument is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

- Otherwise, the return value is non-`NULL`.

These functions are available for obtaining point properties:

- `ST_Latitude(p [, new_latitude_val])`

  With a single argument representing a valid `Point` object *p* that has a geographic spatial reference system (SRS), `ST_Latitude()` returns the latitude value of *p* as a double-precision number.

  With the optional second argument representing a valid latitude value, `ST_Latitude()` returns a `Point` object like the first argument with its latitude equal to the second argument.

  `ST_Latitude()` handles its arguments as described in the introduction to this section, with the addition that if the `Point` object is valid but does not have a geographic SRS, an `ER_SRS_NOT_GEOGRAPHIC` error occurs.

```
mysql> SET @pt = ST_GeomFromText('POINT(45 90)', 4326);
mysql> SELECT ST_Latitude(@pt);
+------------------+
| ST_Latitude(@pt) |
+------------------+
|               45 |
+------------------+
mysql> SELECT ST_AsText(ST_Latitude(@pt, 10));
+---------------------------------+
| ST_AsText(ST_Latitude(@pt, 10)) |
+---------------------------------+
| POINT(10 90)                    |
+---------------------------------+
```

  This function was added in MySQL 8.0.12.

- `ST_Longitude(p [, new_longitude_val])`

  With a single argument representing a valid `Point` object *p* that has a geographic spatial reference system (SRS), `ST_Longitude()` returns the longitude value of *p* as a double-precision number.

  With the optional second argument representing a valid longitude value, `ST_Longitude()` returns a `Point` object like the first argument with its longitude equal to the second argument.

  `ST_Longitude()` handles its arguments as described in the introduction to this section, with the addition that if the `Point` object is valid but does not have a geographic SRS, an `ER_SRS_NOT_GEOGRAPHIC` error occurs.

```
mysql> SET @pt = ST_GeomFromText('POINT(45 90)', 4326);
mysql> SELECT ST_Longitude(@pt);
+-------------------+
| ST_Longitude(@pt) |
+-------------------+
|                90 |
+-------------------+
mysql> SELECT ST_AsText(ST_Longitude(@pt, 10));
+----------------------------------+
| ST_AsText(ST_Longitude(@pt, 10)) |
+----------------------------------+
| POINT(45 10)                     |
+----------------------------------+
```

This function was added in MySQL 8.0.12.

- `ST_X(p[, new_x_val])`

  With a single argument representing a valid `Point` object *p*, `ST_X()` returns the X-coordinate value of *p* as a double-precision number. As of MySQL 8.0.12, the X coordinate is considered to refer to the axis that appears first in the `Point` spatial reference system (SRS) definition.

  With the optional second argument, `ST_X()` returns a `Point` object like the first argument with its X coordinate equal to the second argument. As of MySQL 8.0.12, if the `Point` object has a geographic SRS, the second argument must be in the proper range for longitude or latitude values.

  `ST_X()` handles its arguments as described in the introduction to this section.

```
mysql> SELECT ST_X(Point(56.7, 53.34));
+--------------------------+
| ST_X(Point(56.7, 53.34)) |
+--------------------------+
|                     56.7 |
+--------------------------+
mysql> SELECT ST_AsText(ST_X(Point(56.7, 53.34), 10.5));
+-----------------------------------------+
| ST_AsText(ST_X(Point(56.7, 53.34), 10.5)) |
+-----------------------------------------+
| POINT(10.5 53.34)                       |
+-----------------------------------------+
```

- `ST_Y(p[, new_y_val])`

  With a single argument representing a valid `Point` object *p*, `ST_Y()` returns the Y-coordinate value of *p* as a double-precision number. As of MySQL 8.0.12, the Y coordinate is considered to refer to the axis that appears second in the `Point` spatial reference system (SRS) definition.

  With the optional second argument, `ST_Y()` returns a `Point` object like the first argument with its Y coordinate equal to the second argument. As of MySQL 8.0.12, if the `Point` object has a geographic SRS, the second argument must be in the proper range for longitude or latitude values.

  `ST_Y()` handles its arguments as described in the introduction to this section.

```
mysql> SELECT ST_Y(Point(56.7, 53.34));
+--------------------------+
| ST_Y(Point(56.7, 53.34)) |
+--------------------------+
|                    53.34 |
+--------------------------+
mysql> SELECT ST_AsText(ST_Y(Point(56.7, 53.34), 10.5));
+-----------------------------------------+
| ST_AsText(ST_Y(Point(56.7, 53.34), 10.5)) |
+-----------------------------------------+
| POINT(56.7 10.5)                        |
+-----------------------------------------+
```

12.16.7.3 LineString and MultiLineString Property Functions

A `LineString` consists of `Point` values. You can extract particular points of a `LineString`, count the number of points that it contains, or obtain its length.

Some functions in this section also work for `MultiLineString` values.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL` or any geometry argument is an empty geometry, the return value is `NULL`.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument has an SRID value that refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- Otherwise, the return value is non-`NULL`.

These functions are available for obtaining linestring properties:

- `ST_EndPoint(`*`ls`*`)`

  Returns the `Point` that is the endpoint of the `LineString` value *`ls`*.

  `ST_EndPoint()` handles its arguments as described in the introduction to this section.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_EndPoint(ST_GeomFromText(@ls)));
+-------------------------------------------+
| ST_AsText(ST_EndPoint(ST_GeomFromText(@ls))) |
+-------------------------------------------+
| POINT(3 3)                                |
+-------------------------------------------+
```

- `ST_IsClosed(`*`ls`*`)`

  For a `LineString` value *`ls`*, `ST_IsClosed()` returns 1 if *`ls`* is closed (that is, its `ST_StartPoint()` and `ST_EndPoint()` values are the same).

  For a `MultiLineString` value *`ls`*, `ST_IsClosed()` returns 1 if *`ls`* is closed (that is, the `ST_StartPoint()` and `ST_EndPoint()` values are the same for each `LineString` in *`ls`*).

  `ST_IsClosed()` returns 0 if *`ls`* is not closed, and `NULL` if *`ls`* is `NULL`.

  `ST_IsClosed()` handles its arguments as described in the introduction to this section, with this exception:

  - If the geometry has an SRID value for a geographic spatial reference system (SRS), an `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` error occurs.

```
mysql> SET @ls1 = 'LineString(1 1,2 2,3 3,2 2)';
mysql> SET @ls2 = 'LineString(1 1,2 2,3 3,1 1)';

mysql> SELECT ST_IsClosed(ST_GeomFromText(@ls1));
+-----------------------------------+
| ST_IsClosed(ST_GeomFromText(@ls1)) |
+-----------------------------------+
|                                 0 |
+-----------------------------------+

mysql> SELECT ST_IsClosed(ST_GeomFromText(@ls2));
+-----------------------------------+
| ST_IsClosed(ST_GeomFromText(@ls2)) |
+-----------------------------------+
|                                 1 |
+-----------------------------------+
```

```
mysql> SET @ls3 = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';

mysql> SELECT ST_IsClosed(ST_GeomFromText(@ls3));
+-----------------------------------+
| ST_IsClosed(ST_GeomFromText(@ls3)) |
+-----------------------------------+
|                                 0 |
+-----------------------------------+
```

- `ST_Length(ls [, unit])`

  Returns a double-precision number indicating the length of the `LineString` or `MultiLineString` value *`ls`* in its associated spatial reference system. The length of a `MultiLineString` value is equal to the sum of the lengths of its elements.

  `ST_Length()` computes a result as follows:

  - If the geometry is a valid `LineString` in a Cartesian SRS, the return value is the Cartesian length of the geometry.
  - If the geometry is a valid `MultiLineString` in a Cartesian SRS, the return value is the sum of the Cartesian lengths of its elements.
  - If the geometry is a valid `LineString` in a geographic SRS, the return value is the geodetic length of the geometry in that SRS, in meters.
  - If the geometry is a valid `MultiLineString` in a geographic SRS, the return value is the sum of the geodetic lengths of its elements in that SRS, in meters.

  `ST_Length()` handles its arguments as described in the introduction to this section, with these exceptions:

  - If the geometry is not a `LineString` or `MultiLineString`, the return value is `NULL`.
  - If the geometry is geometrically invalid, either the result is an undefined length (that is, it can be any number), or an error occurs.
  - If the length computation result is `+inf`, an `ER_DATA_OUT_OF_RANGE` error occurs.
  - If the geometry has a geographic SRS with a longitude or latitude that is out of range, an error occurs:
    - If any longitude argument is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.
    - If any latitude argument is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

    Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

  As of MySQL 8.0.16, `ST_Length()` permits an optional *`unit`* argument that specifies the linear unit for the returned length value. These rules apply:

  - If a unit is specified but not supported by MySQL, an `ER_UNIT_NOT_FOUND` error occurs.
  - If a supported linear unit is specified and the SRID is 0, an `ER_GEOMETRY_IN_UNKNOWN_LENGTH_UNIT` error occurs.
  - If a supported linear unit is specified and the SRID is not 0, the result is in that unit.

- If a unit is not specified, the result is in the unit of the SRS of the geometries, whether Cartesian or geographic. Currently, all MySQL SRSs are expressed in meters.

A unit is supported if it is found in the `INFORMATION_SCHEMA ST_UNITS_OF_MEASURE` table. See Section 25.29, “The INFORMATION_SCHEMA ST_UNITS_OF_MEASURE Table”.

```
mysql> SET @ls = ST_GeomFromText('LineString(1 1,2 2,3 3)');
mysql> SELECT ST_Length(@ls);
+--------------------+
| ST_Length(@ls)     |
+--------------------+
| 2.8284271247461903 |
+--------------------+

mysql> SET @mls = ST_GeomFromText('MultiLineString((1 1,2 2,3 3),(4 4,5 5))');
mysql> SELECT ST_Length(@mls);
+-------------------+
| ST_Length(@mls)   |
+-------------------+
| 4.242640687119286 |
+-------------------+

mysql> SET @ls = ST_GeomFromText('LineString(1 1,2 2,3 3)', 4326);
mysql> SELECT ST_Length(@ls);
+-------------------+
| ST_Length(@ls)    |
+-------------------+
| 313701.9623204328 |
+-------------------+
mysql> SELECT ST_Length(@ls, 'metre');
+-------------------------+
| ST_Length(@ls, 'metre') |
+-------------------------+
|       313701.9623204328 |
+-------------------------+
mysql> SELECT ST_Length(@ls, 'foot');
+------------------------+
| ST_Length(@ls, 'foot') |
+------------------------+
|     1029205.9131247795 |
+------------------------+
```

- `ST_NumPoints(ls)`

  Returns the number of `Point` objects in the `LineString` value *`ls`*.

  `ST_NumPoints()` handles its arguments as described in the introduction to this section.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_NumPoints(ST_GeomFromText(@ls));
+-----------------------------------+
| ST_NumPoints(ST_GeomFromText(@ls)) |
+-----------------------------------+
|                                 3 |
+-----------------------------------+
```

- `ST_PointN(ls, N)`

  Returns the *`N`*-th `Point` in the `Linestring` value *`ls`*. Points are numbered beginning with 1.

  `ST_PointN()` handles its arguments as described in the introduction to this section.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_PointN(ST_GeomFromText(@ls),2));
+--------------------------------------------+
| ST_AsText(ST_PointN(ST_GeomFromText(@ls),2)) |
+--------------------------------------------+
| POINT(2 2)                                 |
+--------------------------------------------+
```

- `ST_StartPoint(ls)`

  Returns the `Point` that is the start point of the `LineString` value *`ls`*.

  `ST_StartPoint()` handles its arguments as described in the introduction to this section.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_StartPoint(ST_GeomFromText(@ls)));
+----------------------------------------------+
| ST_AsText(ST_StartPoint(ST_GeomFromText(@ls))) |
+----------------------------------------------+
| POINT(1 1)                                   |
+----------------------------------------------+
```

12.16.7.4 Polygon and MultiPolygon Property Functions

Functions in this section return properties of `Polygon` or `MultiPolygon` values.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL` or any geometry argument is an empty geometry, the return value is `NULL`.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument has an SRID value that refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- For functions that take multiple geometry arguments, if those arguments do not have the same SRID, an `ER_GIS_DIFFERENT_SRIDS` error occurs.
- Otherwise, the return value is non-`NULL`.

These functions are available for obtaining polygon properties:

- `ST_Area({poly|mpoly})`

  Returns a double-precision number indicating the area of the `Polygon` or `MultiPolygon` argument, as measured in its spatial reference system.

  As of MySQL 8.0.13, `ST_Area()` handles its arguments as described in the introduction to this section, with these exceptions:

  - If the geometry is geometrically invalid, either the result is an undefined area (that is, it can be any number), or an error occurs.
  - If the geometry is valid but is not a `Polygon` or `MultiPolygon` object, an `ER_UNEXPECTED_GEOMETRY_TYPE` error occurs.
  - If the geometry is a valid `Polygon` in a Cartesian SRS, the result is the Cartesian area of the polygon.

- If the geometry is a valid `MultiPolygon` in a Cartesian SRS, the result is the sum of the Cartesian area of the polygons.
- If the geometry is a valid `Polygon` in a geographic SRS, the result is the geodetic area of the polygon in that SRS, in square meters.
- If the geometry is a valid `MultiPolygon` in a geographic SRS, the result is the sum of geodetic area of the polygons in that SRS, in square meters.
- If an area computation results in `+inf`, an `ER_DATA_OUT_OF_RANGE` error occurs.
- If the geometry has a geographic SRS with a longitude or latitude that is out of range, an error occurs:
  - If any longitude argument is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.
  - If any latitude argument is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

  Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

Prior to MySQL 8.0.13, `ST_Area()` handles its arguments as described in the introduction to this section, with these exceptions:

- For arguments of dimension 0 or 1, the result is 0.
- If a geometry is empty, the return value is 0 rather than `NULL`.
- For a geometry collection, the result is the sum of the area values of all components. If the geometry collection is empty, its area is returned as 0.
- If the geometry has an SRID value for a geographic spatial reference system (SRS), an `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` error occurs.

```
mysql> SET @poly =
       'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
mysql> SELECT ST_Area(ST_GeomFromText(@poly));
+---------------------------------+
| ST_Area(ST_GeomFromText(@poly)) |
+---------------------------------+
|                               4 |
+---------------------------------+

mysql> SET @mpoly =
       'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT ST_Area(ST_GeomFromText(@mpoly));
+----------------------------------+
| ST_Area(ST_GeomFromText(@mpoly)) |
+----------------------------------+
|                                8 |
+----------------------------------+
```

- `ST_Centroid(`*{poly|mpoly}*`)`

  Returns the mathematical centroid for the `Polygon` or `MultiPolygon` argument as a `Point`. The result is not guaranteed to be on the `MultiPolygon`.

This function processes geometry collections by computing the centroid point for components of highest dimension in the collection. Such components are extracted and made into a single `MultiPolygon`, `MultiLineString`, or `MultiPoint` for centroid computation.

`ST_Centroid()` handles its arguments as described in the introduction to this section, with these exceptions:

- The return value is `NULL` for the additional condition that the argument is an empty geometry collection.
- If the geometry has an SRID value for a geographic spatial reference system (SRS), an `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` error occurs.

```
mysql> SET @poly =
       ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7,5 5))');
mysql> SELECT ST_GeometryType(@poly),ST_AsText(ST_Centroid(@poly));
+------------------------+-------------------------------------------+
| ST_GeometryType(@poly) | ST_AsText(ST_Centroid(@poly))             |
+------------------------+-------------------------------------------+
| POLYGON                | POINT(4.958333333333333 4.958333333333333) |
+------------------------+-------------------------------------------+
```

- `ST_ExteriorRing(poly)`

  Returns the exterior ring of the `Polygon` value *`poly`* as a `LineString`.

  `ST_ExteriorRing()` handles its arguments as described in the introduction to this section.

```
mysql> SET @poly =
       'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT ST_AsText(ST_ExteriorRing(ST_GeomFromText(@poly)));
+-------------------------------------------------+
| ST_AsText(ST_ExteriorRing(ST_GeomFromText(@poly))) |
+-------------------------------------------------+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)                 |
+-------------------------------------------------+
```

- `ST_InteriorRingN(poly, N)`

  Returns the *`N`*-th interior ring for the `Polygon` value *`poly`* as a `LineString`. Rings are numbered beginning with 1.

  `ST_InteriorRingN()` handles its arguments as described in the introduction to this section.

```
mysql> SET @poly =
       'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT ST_AsText(ST_InteriorRingN(ST_GeomFromText(@poly),1));
+-------------------------------------------------+
| ST_AsText(ST_InteriorRingN(ST_GeomFromText(@poly),1)) |
+-------------------------------------------------+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)                 |
+-------------------------------------------------+
```

- `ST_NumInteriorRing(poly)`, `ST_NumInteriorRings(poly)`

  Returns the number of interior rings in the `Polygon` value *`poly`*.

`ST_NumInteriorRing()` and `ST_NuminteriorRings()` handle their arguments as described in the introduction to this section.

```
mysql> SET @poly =
       'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT ST_NumInteriorRings(ST_GeomFromText(@poly));
+-------------------------------------------+
| ST_NumInteriorRings(ST_GeomFromText(@poly)) |
+-------------------------------------------+
|                                         1 |
+-------------------------------------------+
```

12.16.7.5 GeometryCollection Property Functions

These functions return properties of `GeometryCollection` values.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL` or any geometry argument is an empty geometry, the return value is `NULL`.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument has an SRID value that refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- Otherwise, the return value is non-`NULL`.

These functions are available for obtaining geometry collection properties:

- `ST_GeometryN(gc, N)`

  Returns the *`N`*-th geometry in the `GeometryCollection` value *`gc`*. Geometries are numbered beginning with 1.

  `ST_GeometryN()` handles its arguments as described in the introduction to this section.

  ```
  mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
  mysql> SELECT ST_AsText(ST_GeometryN(ST_GeomFromText(@gc),1));
  +-------------------------------------------------+
  | ST_AsText(ST_GeometryN(ST_GeomFromText(@gc),1)) |
  +-------------------------------------------------+
  | POINT(1 1)                                      |
  +-------------------------------------------------+
  ```

- `ST_NumGeometries(gc)`

  Returns the number of geometries in the `GeometryCollection` value *`gc`*.

  `ST_NumGeometries()` handles its arguments as described in the introduction to this section.

  ```
  mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
  mysql> SELECT ST_NumGeometries(ST_GeomFromText(@gc));
  +---------------------------------------+
  | ST_NumGeometries(ST_GeomFromText(@gc)) |
  +---------------------------------------+
  |                                     2 |
  +---------------------------------------+
  ```

12.16.8 Spatial Operator Functions

OpenGIS proposes a number of functions that can produce geometries. They are designed to implement spatial operators.

These functions support all argument type combinations except those that are inapplicable according to the Open Geospatial Consortium specification.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL`, the return value is `NULL`.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument has an SRID value that refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- For functions that take multiple geometry arguments, if those arguments do not have the same SRID, an `ER_GIS_DIFFERENT_SRIDS` error occurs.
- If any geometry argument has an SRID value for a geographic SRS, an `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` error occurs.
- Otherwise, the return value is non-`NULL`.

These spatial operator functions are available:

- `ST_Buffer(`*g*`, `*d*`[, `*strategy1*`[, `*strategy2*`[, `*strategy3*`]]])`

  Returns a geometry that represents all points whose distance from the geometry value *g* is less than or equal to a distance of *d*.

  If the geometry argument is empty, `ST_Buffer()` returns an empty geometry.

  If the distance is 0, `ST_Buffer()` returns the geometry argument unchanged:

```
mysql> SET @pt = ST_GeomFromText('POINT(0 0)');
mysql> SELECT ST_AsText(ST_Buffer(@pt, 0));
+------------------------------+
| ST_AsText(ST_Buffer(@pt, 0)) |
+------------------------------+
| POINT(0 0)                   |
+------------------------------+
```

  `ST_Buffer()` supports negative distances for `Polygon` and `MultiPolygon` values, and for geometry collections containing `Polygon` or `MultiPolygon` values. The result may be an empty geometry.

  `ST_Buffer()` permits up to three optional strategy arguments following the distance argument. Strategies influence buffer computation. These arguments are byte string values produced by the `ST_Buffer_Strategy()` function, to be used for point, join, and end strategies:

  - Point strategies apply to `Point` and `MultiPoint` geometries. If no point strategy is specified, the default is `ST_Buffer_Strategy('point_circle', 32)`.
  - Join strategies apply to `LineString`, `MultiLineString`, `Polygon`, and `MultiPolygon` geometries. If no join strategy is specified, the default is `ST_Buffer_Strategy('join_round', 32)`.

- End strategies apply to `LineString` and `MultiLineString` geometries. If no end strategy is specified, the default is `ST_Buffer_Strategy('end_round', 32)`.

Up to one strategy of each type may be specified, and they may be given in any order.

`ST_Buffer()` handles its arguments as described in the introduction to this section, with these exceptions:

- For a negative distance for `Point`, `MultiPoint`, `LineString`, and `MultiLineString` values, and for geometry collections not containing any `Polygon` or `MultiPolygon` values, an `ER_WRONG_ARGUMENTS` error occurs.
- If multiple strategies of a given type are specified, an `ER_WRONG_ARGUMENTS` error occurs.

```
mysql> SET @pt = ST_GeomFromText('POINT(0 0)');
mysql> SET @pt_strategy = ST_Buffer_Strategy('point_square');
mysql> SELECT ST_AsText(ST_Buffer(@pt, 2, @pt_strategy));
+-----------------------------------------+
| ST_AsText(ST_Buffer(@pt, 2, @pt_strategy)) |
+-----------------------------------------+
| POLYGON((-2 -2,2 -2,2 2,-2 2,-2 -2))       |
+-----------------------------------------+
```

```
mysql> SET @ls = ST_GeomFromText('LINESTRING(0 0,0 5,5 5)');
mysql> SET @end_strategy = ST_Buffer_Strategy('end_flat');
mysql> SET @join_strategy = ST_Buffer_Strategy('join_round', 10);
mysql> SELECT ST_AsText(ST_Buffer(@ls, 5, @end_strategy, @join_strategy))
+-------------------------------------------------------------+
| ST_AsText(ST_Buffer(@ls, 5, @end_strategy, @join_strategy))   |
+-------------------------------------------------------------+
| POLYGON((5 5,5 10,0 10,-3.5355339059327373 8.535533905932738, |
| -5 5,-5 0,0 0,5 0,5 5))                                       |
+-------------------------------------------------------------+
```

- `ST_Buffer_Strategy(`*`strategy`*`[, `*`points_per_circle`*`])`

  This function returns a strategy byte string for use with `ST_Buffer()` to influence buffer computation.

  Information about strategies is available at Boost.org.

  The first argument must be a string indicating a strategy option:

  - For point strategies, permitted values are `'point_circle'` and `'point_square'`.
  - For join strategies, permitted values are `'join_round'` and `'join_miter'`.
  - For end strategies, permitted values are `'end_round'` and `'end_flat'`.

  If the first argument is `'point_circle'`, `'join_round'`, `'join_miter'`, or `'end_round'`, the *`points_per_circle`* argument must be given as a positive numeric value. The maximum *`points_per_circle`* value is the value of the `max_points_in_geometry` system variable.

  For examples, see the description of `ST_Buffer()`.

  `ST_Buffer_Strategy()` handles its arguments as described in the introduction to this section, with these exceptions:

  - If any argument is invalid, an `ER_WRONG_ARGUMENTS` error occurs.

  - If the first argument is `'point_square'` or `'end_flat'`, the *`points_per_circle`* argument must not be given or an `ER_WRONG_ARGUMENTS` error occurs.

- `ST_ConvexHull(`*`g`*`)`

  Returns a geometry that represents the convex hull of the geometry value *`g`*.

  This function computes a geometry's convex hull by first checking whether its vertex points are colinear. The function returns a linear hull if so, a polygon hull otherwise. This function processes geometry collections by extracting all vertex points of all components of the collection, creating a `MultiPoint` value from them, and computing its convex hull.

  `ST_ConvexHull()` handles its arguments as described in the introduction to this section, with this exception:

  - The return value is `NULL` for the additional condition that the argument is an empty geometry collection.

```
mysql> SET @g = 'MULTIPOINT(5 0,25 0,15 10,15 25)';
mysql> SELECT ST_AsText(ST_ConvexHull(ST_GeomFromText(@g)));
+---------------------------------------------+
| ST_AsText(ST_ConvexHull(ST_GeomFromText(@g))) |
+---------------------------------------------+
| POLYGON((5 0,25 0,15 25,5 0))               |
+---------------------------------------------+
```

- `ST_Difference(`*`g1, g2`*`)`

  Returns a geometry that represents the point set difference of the geometry values *`g1`* and *`g2`*.

  `ST_Difference()` handles its arguments as described in the introduction to this section.

```
mysql> SET @g1 = Point(1,1), @g2 = Point(2,2);
mysql> SELECT ST_AsText(ST_Difference(@g1, @g2));
+-----------------------------------+
| ST_AsText(ST_Difference(@g1, @g2)) |
+-----------------------------------+
| POINT(1 1)                        |
+-----------------------------------+
```

- `ST_Intersection(`*`g1, g2`*`)`

  Returns a geometry that represents the point set intersection of the geometry values *`g1`* and *`g2`*.

  `ST_Intersection()` handles its arguments as described in the introduction to this section.

```
mysql> SET @g1 = ST_GeomFromText('LineString(1 1, 3 3)');
mysql> SET @g2 = ST_GeomFromText('LineString(1 3, 3 1)');
mysql> SELECT ST_AsText(ST_Intersection(@g1, @g2));
+-------------------------------------+
| ST_AsText(ST_Intersection(@g1, @g2)) |
+-------------------------------------+
| POINT(2 2)                          |
+-------------------------------------+
```

- `ST_SymDifference(`*`g1, g2`*`)`

Returns a geometry that represents the point set symmetric difference of the geometry values *g1* and *g2*, which is defined as:

```
g1 symdifference g2 := (g1 union g2) difference (g1 intersection g2)
```

Or, in function call notation:

```
ST_SymDifference(g1, g2) = ST_Difference(ST_Union(g1, g2), ST_Intersection(g1, g2))
```

`ST_SymDifference()` handles its arguments as described in the introduction to this section.

```
mysql> SET @g1 = Point(1,1), @g2 = Point(2,2);
mysql> SELECT ST_AsText(ST_SymDifference(@g1, @g2));
+--------------------------------------+
| ST_AsText(ST_SymDifference(@g1, @g2)) |
+--------------------------------------+
| MULTIPOINT((1 1),(2 2))              |
+--------------------------------------+
```

- `ST_Transform(g, target_srid)`

  Transforms a geometry from one spatial reference system (SRS) to another. The return value is a geometry of the same type as the input geometry with all coordinates transformed to the target SRID, *target_srid*. Transformation support is limited to geographic SRSs, unless the SRID of the geometry argument is the same as the target SRID value, in which case the return value is the input geometry for any valid SRS.

  `ST_Transform()` handles its arguments as described in the introduction to this section, with these exceptions:

  - Geometry arguments that have an SRID value for a geographic SRS do not produce an error.
  - If the geometry or target SRID argument has an SRID value that refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
  - If the geometry is in an SRS that `ST_Transform()` cannot transform from, an `ER_TRANSFORM_SOURCE_SRS_NOT_SUPPORTED` error occurs.
  - If the target SRID is in an SRS that `ST_Transform()` cannot transform to, an `ER_TRANSFORM_TARGET_SRS_NOT_SUPPORTED` error occurs.
  - If the geometry is in an SRS that is not WGS 84 and has no TOWGS84 clause, an `ER_TRANSFORM_SOURCE_SRS_MISSING_TOWGS84` error occurs.
  - If the target SRID is in an SRS that is not WGS 84 and has no TOWGS84 clause, an `ER_TRANSFORM_TARGET_SRS_MISSING_TOWGS84` error occurs.

  `ST_SRID(g, target_srid)` and `ST_Transform(g, target_srid)` differ as follows:

  - `ST_SRID()` changes the geometry SRID value without transforming its coordinates.
  - `ST_Transform()` transforms the geometry coordinates in addition to changing its SRID value.

```
mysql> SET @p = ST_GeomFromText('POINT(52.381389 13.064444)', 4326);
mysql> SELECT ST_AsText(@p);
```

```
+---------------------------+
| ST_AsText(@p)             |
+---------------------------+
| POINT(52.381389 13.064444) |
+---------------------------+
mysql> SET @p = ST_Transform(@p, 4230);
mysql> SELECT ST_AsText(@p);
+-----------------------------------------+
| ST_AsText(@p)                           |
+-----------------------------------------+
| POINT(52.38208611407426 13.065520672345304) |
+-----------------------------------------+
```

- `ST_Union(g1, g2)`

  Returns a geometry that represents the point set union of the geometry values *`g1`* and *`g2`*.

  `ST_Union()` handles its arguments as described in the introduction to this section.

```
mysql> SET @g1 = ST_GeomFromText('LineString(1 1, 3 3)');
mysql> SET @g2 = ST_GeomFromText('LineString(1 3, 3 1)');
mysql> SELECT ST_AsText(ST_Union(@g1, @g2));
+-------------------------------------+
| ST_AsText(ST_Union(@g1, @g2))       |
+-------------------------------------+
| MULTILINESTRING((1 1,3 3),(1 3,3 1)) |
+-------------------------------------+
```

In addition, Section 12.16.7, “Geometry Property Functions”, discusses several functions that construct new geometries from existing ones. See that section for descriptions of these functions:

- `ST_Envelope(g)`
- `ST_StartPoint(ls)`
- `ST_EndPoint(ls)`
- `ST_PointN(ls, N)`
- `ST_ExteriorRing(poly)`
- `ST_InteriorRingN(poly, N)`
- `ST_GeometryN(gc, N)`

12.16.9 Functions That Test Spatial Relations Between Geometry Objects

The functions described in this section take two geometries as arguments and return a qualitative or quantitative relation between them.

MySQL implements two sets of functions using function names defined by the OpenGIS specification. One set tests the relationship between two geometry values using precise object shapes, the other set uses object minimum bounding rectangles (MBRs).

12.16.9.1 Spatial Relation Functions That Use Object Shapes

The OpenGIS specification defines the following functions to test the relationship between two geometry values `g1` and `g2`, using precise object shapes. The return values 1 and 0 indicate true and false, respectively, except for `ST_Distance()`, which returns distance values.

Functions in this section detect arguments in either Cartesian or geographic spatial reference systems (SRSs), and return results appropriate to the SRS.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL` or any geometry argument is an empty geometry, the return value is `NULL`.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- For functions that take multiple geometry arguments, if those arguments do not have the same SRID, an `ER_GIS_DIFFERENT_SRIDS` error occurs.
- If any geometry argument is geometrically invalid, either the result is true or false (it is undefined which), or an error occurs.
- For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:
  - If a longitude value is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.
  - If a latitude value is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

  Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.
- Otherwise, the return value is non-`NULL`.

These object-shape functions are available for testing geometry relationships:

- `ST_Contains(`*`g1, g2`*`)`

  Returns 1 or 0 to indicate whether *`g1`* completely contains *`g2`*. This tests the opposite relationship as `ST_Within()`.

  `ST_Contains()` handles its arguments as described in the introduction to this section.

- `ST_Crosses(`*`g1, g2`*`)`

  Two geometries *spatially cross* if their spatial relation has the following properties:

  - Unless `g1` and `g2` are both of dimension 1: `g1` crosses `g2` if the interior of `g2` has points in common with the interior of `g1`, but `g2` does not cover the entire interior of `g1`.
  - If both `g1` and `g2` are of dimension 1: If the lines cross each other in a finite number of points (that is, no common line segments, only single points in common).

  This function returns 1 or 0 to indicate whether *`g1`* spatially crosses *`g2`*.

  `ST_Crosses()` handles its arguments as described in the introduction to this section except that the return value is `NULL` for these additional conditions:

  - `g1` is of dimension 2 (`Polygon` or `MultiPolygon`).
  - *`g2`* is of dimension 1 (`Point` or `MultiPoint`).

- `ST_Disjoint(`*`g1, g2`*`)`

  Returns 1 or 0 to indicate whether *`g1`* is spatially disjoint from (does not intersect) *`g2`*.

`ST_Disjoint()` handles its arguments as described in the introduction to this section.

- `ST_Distance(`*`g1`*`, `*`g2`*` [, `*`unit`*`])`

  Returns the distance between *`g1`* and *`g2`*, measured in the length unit of the spatial reference system (SRS) of the geometry arguments, or in the unit of the optional *`unit`* argument if that is specified.

  This function processes geometry collections by returning the shortest distance among all combinations of the components of the two geometry arguments.

  `ST_Distance()` handles its geometry arguments as described in the introduction to this section, with these exceptions:

  - `ST_Distance()` detects arguments in a geographic (ellipsoidal) spatial reference system and returns the geodetic distance on the ellipsoid. As of MySQL 8.0.18, `ST_Distance()` supports distance calculations for geographic SRS arguments of all geometry types. Prior to MySQL 8.0.18, the only permitted geographic argument types are `Point` and `Point`, or `Point` and `MultiPoint` (in any argument order). If called with other geometry type argument combinations in a geographic SRS, an `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` error occurs.
  - If any argument is geometrically invalid, either the result is an undefined distance (that is, it can be any number), or an error occurs.
  - If an intermediate or final result produces `NaN` or a negative number, an `ER_GIS_INVALID_DATA` error occurs.

  As of MySQL 8.0.14, `ST_Distance()` permits an optional *`unit`* argument that specifies the linear unit for the returned distance value. These rules apply:

  - If a unit is specified but not supported by MySQL, an `ER_UNIT_NOT_FOUND` error occurs.
  - If a supported linear unit is specified and the SRID is 0, an `ER_GEOMETRY_IN_UNKNOWN_LENGTH_UNIT` error occurs.
  - If a supported linear unit is specified and the SRID is not 0, the result is in that unit.
  - If a unit is not specified, the result is in the unit of the SRS of the geometries, whether Cartesian or geographic. Currently, all MySQL SRSs are expressed in meters.

  A unit is supported if it is found in the `INFORMATION_SCHEMA ST_UNITS_OF_MEASURE` table. See Section 25.29, “The INFORMATION_SCHEMA ST_UNITS_OF_MEASURE Table”.

```
mysql> SET @g1 = Point(1,1);
mysql> SET @g2 = Point(2,2);
mysql> SELECT ST_Distance(@g1, @g2);
+----------------------+
| ST_Distance(@g1, @g2) |
+----------------------+
|    1.4142135623730951 |
+----------------------+

mysql> SET @g1 = ST_GeomFromText('POINT(1 1)', 4326);
mysql> SET @g2 = ST_GeomFromText('POINT(2 2)', 4326);
mysql> SELECT ST_Distance(@g1, @g2);
+----------------------+
| ST_Distance(@g1, @g2) |
+----------------------+
|     156874.3859490455 |
```

```
+----------------------+
mysql> SELECT ST_Distance(@g1, @g2, 'metre');
+-------------------------------+
| ST_Distance(@g1, @g2, 'metre') |
+-------------------------------+
|              156874.3859490455 |
+-------------------------------+
mysql> SELECT ST_Distance(@g1, @g2, 'foot');
+-------------------------------+
| ST_Distance(@g1, @g2, 'foot') |
+-------------------------------+
|             514679.7439273146 |
+-------------------------------+
```

For the special case of distance calculations on a sphere, see the `ST_Distance_Sphere()` function.

- `ST_Equals(g1, g2)`

  Returns 1 or 0 to indicate whether *g1* is spatially equal to *g2*.

  `ST_Equals()` handles its arguments as described in the introduction to this section, except that it does not return `NULL` for empty geometry arguments.

```
mysql> SET @g1 = Point(1,1), @g2 = Point(2,2);
mysql> SELECT ST_Equals(@g1, @g1), ST_Equals(@g1, @g2);
+---------------------+---------------------+
| ST_Equals(@g1, @g1) | ST_Equals(@g1, @g2) |
+---------------------+---------------------+
|                   1 |                   0 |
+---------------------+---------------------+
```

- `ST_Intersects(g1, g2)`

  Returns 1 or 0 to indicate whether *g1* spatially intersects *g2*.

  `ST_Intersects()` handles its arguments as described in the introduction to this section.

- `ST_Overlaps(g1, g2)`

  Two geometries *spatially overlap* if they intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

  This function returns 1 or 0 to indicate whether *g1* spatially overlaps *g2*.

  `ST_Overlaps()` handles its arguments as described in the introduction to this section except that the return value is `NULL` for the additional condition that the dimensions of the two geometries are not equal.

- `ST_Touches(g1, g2)`

  Two geometries *spatially touch* if their interiors do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

  This function returns 1 or 0 to indicate whether *g1* spatially touches *g2*.

  `ST_Touches()` handles its arguments as described in the introduction to this section except that the return value is `NULL` for the additional condition that both geometries are of dimension 0 (`Point` or `MultiPoint`).

- `ST_Within(g1, g2)`

Returns 1 or 0 to indicate whether *g1* is spatially within *g2*. This tests the opposite relationship as `ST_Contains()`.

`ST_Within()` handles its arguments as described in the introduction to this section.

12.16.9.2 Spatial Relation Functions That Use Minimum Bounding Rectangles

MySQL provides several MySQL-specific functions that test the relationship between minimum bounding rectangles (MBRs) of two geometries `g1` and `g2`. The return values 1 and 0 indicate true and false, respectively.

The bounding box of a point is interpreted as a point that is both boundary and interior.

The bounding box of a straight horizontal or vertical line is interpreted as a line where the interior of the line is also boundary. The endpoints are boundary points.

If any of the parameters are geometry collections, the interior, boundary, and exterior of those parameters are those of the union of all elements in the collection.

Functions in this section detect arguments in either Cartesian or geographic spatial reference systems (SRSs), and return results appropriate to the SRS.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL` or an empty geometry, the return value is `NULL`.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- For functions that take multiple geometry arguments, if those arguments do not have the same SRID, an `ER_GIS_DIFFERENT_SRIDS` error occurs.
- If any argument is geometrically invalid, either the result is true or false (it is undefined which), or an error occurs.
- For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:
  - If a longitude value is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.
  - If a latitude value is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

  Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.
- Otherwise, the return value is non-`NULL`.

These MBR functions are available for testing geometry relationships:

- `MBRContains(`*g1, g2*`)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* contains the minimum bounding rectangle of *g2*. This tests the opposite relationship as `MBRWithin()`.

  `MBRContains()` handles its arguments as described in the introduction to this section.

```
mysql> SET @g1 = ST_GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = ST_GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRWithin(@g2,@g1);
+----------------------+--------------------+
| MBRContains(@g1,@g2) | MBRWithin(@g2,@g1) |
+----------------------+--------------------+
|                    1 |                  1 |
+----------------------+--------------------+
```

- `MBRCoveredBy(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* is covered by the minimum bounding rectangle of *g2*. This tests the opposite relationship as `MBRCovers()`.

  `MBRCoveredBy()` handles its arguments as described in the introduction to this section.

```
mysql> SET @g1 = ST_GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = ST_GeomFromText('Point(1 1)');
mysql> SELECT MBRCovers(@g1,@g2), MBRCoveredby(@g1,@g2);
+--------------------+----------------------+
| MBRCovers(@g1,@g2) | MBRCoveredby(@g1,@g2) |
+--------------------+----------------------+
|                  1 |                    0 |
+--------------------+----------------------+
mysql> SELECT MBRCovers(@g2,@g1), MBRCoveredby(@g2,@g1);
+--------------------+----------------------+
| MBRCovers(@g2,@g1) | MBRCoveredby(@g2,@g1) |
+--------------------+----------------------+
|                  0 |                    1 |
+--------------------+----------------------+
```

- `MBRCovers(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* covers the minimum bounding rectangle of *g2*. This tests the opposite relationship as `MBRCoveredBy()`. See the description of `MBRCoveredBy()` for examples.

  `MBRCovers()` handles its arguments as described in the introduction to this section.

- `MBRDisjoint(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

  `MBRDisjoint()` handles its arguments as described in the introduction to this section.

- `MBREquals(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* are the same.

  `MBREquals()` handles its arguments as described in the introduction to this section, except that it does not return `NULL` for empty geometry arguments.

- `MBRIntersects(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* intersect.

`MBRIntersects()` handles its arguments as described in the introduction to this section.

- `MBROverlaps(g1, g2)`

  Two geometries *spatially overlap* if they intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

  This function returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* overlap.

  `MBROverlaps()` handles its arguments as described in the introduction to this section.

- `MBRTouches(g1, g2)`

  Two geometries *spatially touch* if their interiors do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

  This function returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* touch.

  `MBRTouches()` handles its arguments as described in the introduction to this section.

- `MBRWithin(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* is within the minimum bounding rectangle of *g2*. This tests the opposite relationship as `MBRContains()`.

  `MBRWithin()` handles its arguments as described in the introduction to this section.

```
mysql> SET @g1 = ST_GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = ST_GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+--------------------+--------------------+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+--------------------+--------------------+
|                  1 |                  0 |
+--------------------+--------------------+
```

12.16.10 Spatial Geohash Functions

Geohash is a system for encoding latitude and longitude coordinates of arbitrary precision into a text string. Geohash values are strings that contain only characters chosen from `"0123456789bcdefghjkmnpqrstuvwxyz"`.

The functions in this section enable manipulation of geohash values, which provides applications the capabilities of importing and exporting geohash data, and of indexing and searching geohash values.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL`, the return value is `NULL`.
- If any argument is invalid, an error occurs.
- If any argument has a longitude or latitude that is out of range, an error occurs:
  - If any longitude argument is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.

  - If any latitude argument is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

  Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

- If any point argument does not have SRID 0 or 4326, an `ER_SRS_NOT_FOUND` error occurs. *`point`* argument SRID validity is not checked.
- If any SRID argument refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- If any SRID argument is not within the range of a 32-bit unsigned integer, an `ER_DATA_OUT_OF_RANGE` error occurs.
- Otherwise, the return value is non-`NULL`.

These geohash functions are available:

- `ST_GeoHash(`*`longitude, latitude, max_length`*`)`, `ST_GeoHash(`*`point, max_length`*`)`

  Returns a geohash string in the connection character set and collation.

  For the first syntax, the *`longitude`* must be a number in the range [−180, 180], and the *`latitude`* must be a number in the range [−90, 90]. For the second syntax, a `POINT` value is required, where the X and Y coordinates are in the valid ranges for longitude and latitude, respectively.

  The resulting string is no longer than *`max_length`* characters, which has an upper limit of 100. The string might be shorter than *`max_length`* characters because the algorithm that creates the geohash value continues until it has created a string that is either an exact representation of the location or *`max_length`* characters, whichever comes first.

  `ST_GeoHash()` handles its arguments as described in the introduction to this section.

```
mysql> SELECT ST_GeoHash(180,0,10), ST_GeoHash(-180,-90,15);
+----------------------+------------------------+
| ST_GeoHash(180,0,10) | ST_GeoHash(-180,-90,15) |
+----------------------+------------------------+
| xbpbpbpbpb           | 000000000000000         |
+----------------------+------------------------+
```

- `ST_LatFromGeoHash(`*`geohash_str`*`)`

  Returns the latitude from a geohash string value, as a double-precision number in the range [−90, 90].

  The `ST_LatFromGeoHash()` decoding function reads no more than 433 characters from the *`geohash_str`* argument. That represents the upper limit on information in the internal representation of coordinate values. Characters past the 433rd are ignored, even if they are otherwise illegal and produce an error.

  `ST_LatFromGeoHash()` handles its arguments as described in the introduction to this section.

```
mysql> SELECT ST_LatFromGeoHash(ST_GeoHash(45,-20,10));
+-----------------------------------------+
| ST_LatFromGeoHash(ST_GeoHash(45,-20,10)) |
+-----------------------------------------+
|                                     -20 |
+-----------------------------------------+
```

- `ST_LongFromGeoHash(`*`geohash_str`*`)`

Returns the longitude from a geohash string value, as a double-precision number in the range [−180, 180].

The remarks in the description of `ST_LatFromGeoHash()` regarding the maximum number of characters processed from the *`geohash_str`* argument also apply to `ST_LongFromGeoHash()`.

`ST_LongFromGeoHash()` handles its arguments as described in the introduction to this section.

```
mysql> SELECT ST_LongFromGeoHash(ST_GeoHash(45,-20,10));
+-----------------------------------------+
| ST_LongFromGeoHash(ST_GeoHash(45,-20,10)) |
+-----------------------------------------+
|                                      45 |
+-----------------------------------------+
```

- `ST_PointFromGeoHash(`*`geohash_str, srid`*`)`

  Returns a `POINT` value containing the decoded geohash value, given a geohash string value.

  The X and Y coordinates of the point are the longitude in the range [−180, 180] and the latitude in the range [−90, 90], respectively.

  The *`srid`* argument is an 32-bit unsigned integer.

  The remarks in the description of `ST_LatFromGeoHash()` regarding the maximum number of characters processed from the *`geohash_str`* argument also apply to `ST_PointFromGeoHash()`.

  `ST_PointFromGeoHash()` handles its arguments as described in the introduction to this section.

```
mysql> SET @gh = ST_GeoHash(45,-20,10);
mysql> SELECT ST_AsText(ST_PointFromGeoHash(@gh,0));
+---------------------------------------+
| ST_AsText(ST_PointFromGeoHash(@gh,0)) |
+---------------------------------------+
| POINT(45 -20)                         |
+---------------------------------------+
```

12.16.11 Spatial GeoJSON Functions

This section describes functions for converting between GeoJSON documents and spatial values. GeoJSON is an open standard for encoding geometric/geographical features. For more information, see http://geojson.org. The functions discussed here follow GeoJSON specification revision 1.0.

GeoJSON supports the same geometric/geographic data types that MySQL supports. Feature and FeatureCollection objects are not supported, except that geometry objects are extracted from them. CRS support is limited to values that identify an SRID.

MySQL also supports a native `JSON` data type and a set of SQL functions to enable operations on JSON values. For more information, see Section 11.6, “The JSON Data Type”, and Section 12.17, “JSON Functions”.

- `ST_AsGeoJSON(`*`g [, max_dec_digits [, options]`*`])`

  Generates a GeoJSON object from the geometry *`g`*. The object string has the connection character set and collation.

  If any argument is `NULL`, the return value is `NULL`. If any non-`NULL` argument is invalid, an error occurs.

max_dec_digits, if specified, limits the number of decimal digits for coordinates and causes rounding of output. If not specified, this argument defaults to its maximum value of $2^{32} - 1$. The minimum is 0.

options, if specified, is a bitmask. The following table shows the permitted flag values. If the geometry argument has an SRID of 0, no CRS object is produced even for those flag values that request one.

Flag Value	Meaning
0	No options. This is the default if *options* is not specified.
1	Add a bounding box to the output.
2	Add a short-format CRS URN to the output. The default format is a short format (EPSG:*srid*).
4	Add a long-format CRS URN (urn:ogc:def:crs:EPSG::*srid*). This flag overrides flag 2. For example, option values of 5 and 7 mean the same (add a bounding box and a long-format CRS URN).

```
mysql> SELECT ST_AsGeoJSON(ST_GeomFromText('POINT(11.11111 12.22222)'),2);
+-------------------------------------------------------------+
| ST_AsGeoJSON(ST_GeomFromText('POINT(11.11111 12.22222)'),2) |
+-------------------------------------------------------------+
| {"type": "Point", "coordinates": [11.11, 12.22]}            |
+-------------------------------------------------------------+
```

- ST_GeomFromGeoJSON(*str* [, *options* [, *srid*]])

Parses a string *str* representing a GeoJSON object and returns a geometry.

If any argument is NULL, the return value is NULL. If any non-NULL argument is invalid, an error occurs.

options, if given, describes how to handle GeoJSON documents that contain geometries with coordinate dimensions higher than 2. The following table shows the permitted *options* values.

Option Value	Meaning
1	Reject the document and produce an error. This is the default if *options* is not specified.
2, 3, 4	Accept the document and strip off the coordinates for higher coordinate dimensions.

options values of 2, 3, and 4 currently produce the same effect. If geometries with coordinate dimensions higher than 2 are supported in the future, these values will produce different effects.

The *srid* argument, if given, must be a 32-bit unsigned integer. If not given, the geometry return value has an SRID of 4326.

If *srid* refers to an undefined spatial reference system (SRS), an ER_SRS_NOT_FOUND error occurs.

For geographic SRS geometry arguments, if any argument has a longitude or latitude that is out of range, an error occurs:

- If a longitude value is not in the range (−180, 180], an ER_LONGITUDE_OUT_OF_RANGE error occurs.
- If a latitude value is not in the range [−90, 90], an ER_LATITUDE_OUT_OF_RANGE error occurs.

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

GeoJSON geometry, feature, and feature collection objects may have a `crs` property. The parsing function parses named CRS URNs in the `urn:ogc:def:crs:EPSG::srid` and `EPSG:srid` namespaces, but not CRSs given as link objects. Also, `urn:ogc:def:crs:OGC:1.3:CRS84` is recognized as SRID 4326. If an object has a CRS that is not understood, an error occurs, with the exception that if the optional *`srid`* argument is given, any CRS is ignored even if it is invalid.

If a `crs` member that specifies an SRID different from the top-level object SRID is found at a lower level of the GeoJSON document, an `ER_INVALID_GEOJSON_CRS_NOT_TOP_LEVEL` error occurs.

As specified in the GeoJSON specification, parsing is case sensitive for the `type` member of the GeoJSON input (`Point`, `LineString`, and so forth). The specification is silent regarding case sensitivity for other parsing, which in MySQL is not case-sensitive.

This example shows the parsing result for a simple GeoJSON object:

```
mysql> SET @json = '{ "type": "Point", "coordinates": [102.0, 0.0]}';
mysql> SELECT ST_AsText(ST_GeomFromGeoJSON(@json));
+---------------------------------------+
| ST_AsText(ST_GeomFromGeoJSON(@json)) |
+---------------------------------------+
| POINT(102 0)                          |
+---------------------------------------+
```

12.16.12 Spatial Convenience Functions

The functions in this section provide convenience operations on geometry values.

Unless otherwise specified, functions in this section handle their arguments as follows:

- If any argument is `NULL`, the return value is `NULL`.
- If any geometry argument is not a syntactically well-formed geometry, an `ER_GIS_INVALID_DATA` error occurs.
- If any geometry argument has an SRID value that refers to an undefined spatial reference system (SRS), an `ER_SRS_NOT_FOUND` error occurs.
- For functions that take multiple geometry arguments, if those arguments do not have the same SRID, an `ER_GIS_DIFFERENT_SRIDS` error occurs.
- Otherwise, the return value is non-`NULL`.

These convenience functions are available:

- `ST_Distance_Sphere(`*`g1`*`, `*`g2`*` [, `*`radius`*`])`

  Returns the mimimum spherical distance between `Point` or `MultiPoint` arguments on a sphere, in meters. (For general-purpose distance calculations, see the `ST_Distance()` function.) The optional *`radius`* argument should be given in meters.

  If both geometry parameters are valid Cartesian `Point` or `MultiPoint` values in SRID 0, the return value is shortest distance between the two geometries on a sphere with the provided radius. If omitted, the default radius is 6,370,986 meters, Point X and Y coordinates are interpreted as longitude and latitude, respectively, in degrees.

If both geometry parameters are valid `Point` or `MultiPoint` values in a geographic spatial reference system (SRS), the return value is the shortest distance between the two geometries on a sphere with the provided radius. If omitted, the default radius is equal to the mean radius, defined as (2a+b)/3, where a is the semi-major axis and b is the semi-minor axis of the SRS.

`ST_Distance_Sphere()` handles its arguments as described in the introduction to this section, with these exceptions:

- Supported geometry argument combinations are `Point` and `Point`, or `Point` and `MultiPoint` (in any argument order). If at least one of the geometries is neither `Point` nor `MultiPoint`, and its SRID is 0, an `ER_NOT_IMPLEMENTED_FOR_CARTESIAN_SRS` error occurs. If at least one of the geometries is neither `Point` nor `MultiPoint`, and its SRID refers to a geographic SRS, an `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` error occurs. If any geometry refers to a projected SRS, an `ER_NOT_IMPLEMENTED_FOR_PROJECTED_SRS` error occurs.
- If any argument has a longitude or latitude that is out of range, an error occurs:
  - If any longitude argument is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.
  - If any latitude argument is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

  Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.
- If the *`radius`* argument is present but not positive, an `ER_NONPOSITIVE_RADIUS` error occurs.
- If the distance exceeds the range of a double-precision number, an `ER_STD_OVERFLOW_ERROR` error occurs.

```
mysql> SET @pt1 = ST_GeomFromText('POINT(0 0)');
mysql> SET @pt2 = ST_GeomFromText('POINT(180 0)');
mysql> SELECT ST_Distance_Sphere(@pt1, @pt2);
+--------------------------------+
| ST_Distance_Sphere(@pt1, @pt2) |
+--------------------------------+
|             20015042.813723423 |
+--------------------------------+
```

- `ST_IsValid(`*`g`*`)`

Returns 1 if the argument is geometrically valid, 0 if the argument is not geometrically valid. Geometry validity is defined by the OGC specification.

The only valid empty geometry is represented in the form of an empty geometry collection value. `ST_IsValid()` returns 1 in this case. MySQL does not support GIS `EMPTY` values such as `POINT EMPTY`.

`ST_IsValid()` handles its arguments as described in the introduction to this section, with this exception:

- If the geometry has a geographic SRS with a longitude or latitude that is out of range, an error occurs:
  - If any longitude argument is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.

- If any latitude argument is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

Ranges shown are in degrees. If an SRS uses another unit, the range uses the corresponding values in its unit. The exact range limits deviate slightly due to floating-point arithmetic.

```
mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0,-0.00 0,0.0 0)');
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 0, 1 1)');
mysql> SELECT ST_IsValid(@ls1);
+------------------+
| ST_IsValid(@ls1) |
+------------------+
|                0 |
+------------------+
mysql> SELECT ST_IsValid(@ls2);
+------------------+
| ST_IsValid(@ls2) |
+------------------+
|                1 |
+------------------+
```

- `ST_MakeEnvelope(`*`pt1, pt2`*`)`

Returns the rectangle that forms the envelope around two points, as a `Point`, `LineString`, or `Polygon`.

Calculations are done using the Cartesian coordinate system rather than on a sphere, spheroid, or on earth.

Given two points *`pt1`* and *`pt2`*, `ST_MakeEnvelope()` creates the result geometry on an abstract plane like this:

- If *`pt1`* and *`pt2`* are equal, the result is the point *`pt1`*.
- Otherwise, if `(`*`pt1, pt2`*`)` is a vertical or horizontal line segment, the result is the line segment `(`*`pt1, pt2`*`)`.
- Otherwise, the result is a polygon using *`pt1`* and *`pt2`* as diagonal points.

The result geometry has an SRID of 0.

`ST_MakeEnvelope()` handles its arguments as described in the introduction to this section, with these exceptions:

- If the arguments are not `Point` values, an `ER_WRONG_ARGUMENTS` error occurs.
- An `ER_GIS_INVALID_DATA` error occurs for the additional condition that any coordinate value of the two points is infinite or `NaN`.
- If any geometry has an SRID value for a geographic spatial reference system (SRS), an `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` error occurs.

```
mysql> SET @pt1 = ST_GeomFromText('POINT(0 0)');
mysql> SET @pt2 = ST_GeomFromText('POINT(1 1)');
mysql> SELECT ST_AsText(ST_MakeEnvelope(@pt1, @pt2));
+---------------------------------------+
| ST_AsText(ST_MakeEnvelope(@pt1, @pt2)) |
+---------------------------------------+
```

```
| POLYGON((0 0,1 0,1 1,0 1,0 0))         |
+----------------------------------------+
```

- `ST_Simplify(g, max_distance)`

  Simplifies a geometry using the Douglas-Peucker algorithm and returns a simplified value of the same type.

  The geometry may be any geometry type, although the Douglas-Peucker algorithm may not actually process every type. A geometry collection is processed by giving its components one by one to the simplification algorithm, and the returned geometries are put into a geometry collection as result.

  The *`max_distance`* argument is the distance (in units of the input coordinates) of a vertex to other segments to be removed. Vertices within this distance of the simplified linestring are removed.

  According to Boost.Geometry, geometries might become invalid as a result of the simplification process, and the process might create self-intersections. To check the validity of the result, pass it to `ST_IsValid()`.

  `ST_Simplify()` handles its arguments as described in the introduction to this section, with this exception:

  - If the *`max_distance`* argument is not positive, or is `NaN`, an `ER_WRONG_ARGUMENTS` error occurs.

```
mysql> SET @g = ST_GeomFromText('LINESTRING(0 0,0 1,1 1,1 2,2 2,2 3,3 3)');
mysql> SELECT ST_AsText(ST_Simplify(@g, 0.5));
+---------------------------------+
| ST_AsText(ST_Simplify(@g, 0.5)) |
+---------------------------------+
| LINESTRING(0 0,0 1,1 1,2 3,3 3) |
+---------------------------------+
mysql> SELECT ST_AsText(ST_Simplify(@g, 1.0));
+---------------------------------+
| ST_AsText(ST_Simplify(@g, 1.0)) |
+---------------------------------+
| LINESTRING(0 0,3 3)             |
+---------------------------------+
```

- `ST_Validate(g)`

  Validates a geometry according to the OGC specification. A geometry can be syntactically well-formed (WKB value plus SRID) but geometrically invalid. For example, this polygon is geometrically invalid: `POLYGON((0 0, 0 0, 0 0, 0 0, 0 0))`

  `ST_Validate()` returns the geometry if it is syntactically well-formed and is geometrically valid, `NULL` if the argument is not syntactically well-formed or is not geometrically valid or is `NULL`.

  `ST_Validate()` can be used to filter out invalid geometry data, although at a cost. For applications that require more precise results not tainted by invalid data, this penalty may be worthwhile.

  If the geometry argument is valid, it is returned as is, except that if an input `Polygon` or `MultiPolygon` has clockwise rings, those rings are reversed before checking for validity. If the geometry is valid, the value with the reversed rings is returned.

  The only valid empty geometry is represented in the form of an empty geometry collection value. `ST_Validate()` returns it directly without further checks in this case.

As of MySQL 8.0.13, `ST_Validate()` handles its arguments as described in the introduction to this section, with these exceptions:

- If the geometry has a geographic SRS with a longitude or latitude that is out of range, an error occurs:
  - If any longitude argument is not in the range (−180, 180], an `ER_LONGITUDE_OUT_OF_RANGE` error occurs.
  - If any latitude argument is not in the range [−90, 90], an `ER_LATITUDE_OUT_OF_RANGE` error occurs.

  Ranges shown are in degrees. The exact range limits deviate slightly due to floating-point arithmetic.

Prior to MySQL 8.0.13, `ST_Validate()` handles its arguments as described in the introduction to this section, with these exceptions:

- If the geometry is not syntactically well-formed, the return value is `NULL`. An `ER_GIS_INVALID_DATA` error does not occur.
- If the geometry has an SRID value for a geographic spatial reference system (SRS), an `ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS` error occurs.

```
mysql> SET @ls1 = ST_GeomFromText('LINESTRING(0 0)');
mysql> SET @ls2 = ST_GeomFromText('LINESTRING(0 0, 1 1)');
mysql> SELECT ST_AsText(ST_Validate(@ls1));
+-----------------------------+
| ST_AsText(ST_Validate(@ls1)) |
+-----------------------------+
| NULL                        |
+-----------------------------+
mysql> SELECT ST_AsText(ST_Validate(@ls2));
+-----------------------------+
| ST_AsText(ST_Validate(@ls2)) |
+-----------------------------+
| LINESTRING(0 0,1 1)         |
+-----------------------------+
```

12.17 JSON Functions

The functions described in this section perform operations on JSON values. For discussion of the `JSON` data type and additional examples showing how to use these functions, see Section 11.6, “The JSON Data Type”.

For functions that take a JSON argument, an error occurs if the argument is not a valid JSON value. Arguments parsed as JSON are indicated by *`json_doc`*; arguments indicated by *`val`* are not parsed.

A set of spatial functions for operating on GeoJSON values is also available. See Section 12.16.11, “Spatial GeoJSON Functions”.

12.17.1 JSON Function Reference

Table 12.21 JSON Functions

Name	Description
`JSON_ARRAY()`	Create JSON array
`JSON_ARRAY_APPEND()`	Append data to JSON document

Name	Description
JSON_ARRAY_INSERT()	Insert into JSON array
->	Return value from JSON column after evaluating path; equivalent to JSON_EXTRACT().
JSON_CONTAINS()	Whether JSON document contains specific object at path
JSON_CONTAINS_PATH()	Whether JSON document contains any data at path
JSON_DEPTH()	Maximum depth of JSON document
JSON_EXTRACT()	Return data from JSON document
->>	Return value from JSON column after evaluating path and unquoting the result; equivalent to JSON_UNQUOTE(JSON_EXTRACT()).
JSON_INSERT()	Insert data into JSON document
JSON_KEYS()	Array of keys from JSON document
JSON_LENGTH()	Number of elements in JSON document
JSON_MERGE() (deprecated 8.0.3)	Merge JSON documents, preserving duplicate keys. Deprecated synonym for JSON_MERGE_PRESERVE()
JSON_MERGE_PATCH()	Merge JSON documents, replacing values of duplicate keys
JSON_MERGE_PRESERVE()	Merge JSON documents, preserving duplicate keys
JSON_OBJECT()	Create JSON object
JSON_OVERLAPS()	Compares two JSON documents, returns TRUE (1) if these have any key-value pairs or array elements in common, otherwise FALSE (0)
JSON_PRETTY()	Print a JSON document in human-readable format
JSON_QUOTE()	Quote JSON document
JSON_REMOVE()	Remove data from JSON document
JSON_REPLACE()	Replace values in JSON document
JSON_SCHEMA_VALID()	Validate JSON document against JSON schema; returns TRUE/1 if document validates against schema, or FALSE/0 if it does not
JSON_SCHEMA_VALIDATION_REPORT()	Validate JSON document against JSON schema; returns report in JSON format on outcome on validation including success or failure and reasons for failure
JSON_SEARCH()	Path to value within JSON document
JSON_SET()	Insert data into JSON document
JSON_STORAGE_FREE()	Freed space within binary representation of JSON column value following partial update
JSON_STORAGE_SIZE()	Space used for storage of binary representation of a JSON document
JSON_TABLE()	Return data from a JSON expression as a relational table
JSON_TYPE()	Type of JSON value
JSON_UNQUOTE()	Unquote JSON value
JSON_VALID()	Whether JSON value is valid

Name	Description
`MEMBER OF()`	Returns true (1) if first operand matches any element of JSON array passed as second operand, otherwise returns false (0)

MySQL supports two aggregate JSON functions `JSON_ARRAYAGG()` and `JSON_OBJECTAGG()`. See Section 12.20, “Aggregate (GROUP BY) Functions”, for descriptions of these.

MySQL also supports “pretty-printing” of JSON values in an easy-to-read format, using the `JSON_PRETTY()` function. You can see how much storage space a given JSON value takes up, and how much space remains for additional storage, using `JSON_STORAGE_SIZE()` and `JSON_STORAGE_FREE()`, respectively. For complete descriptions of these functions, see Section 12.17.8, “JSON Utility Functions”.

12.17.2 Functions That Create JSON Values

The functions listed in this section compose JSON values from component elements.

- `JSON_ARRAY([val[, val] ...])`

  Evaluates a (possibly empty) list of values and returns a JSON array containing those values.

```
mysql> SELECT JSON_ARRAY(1, "abc", NULL, TRUE, CURTIME());
+---------------------------------------------+
| JSON_ARRAY(1, "abc", NULL, TRUE, CURTIME()) |
+---------------------------------------------+
| [1, "abc", null, true, "11:30:24.000000"]   |
+---------------------------------------------+
```

- `JSON_OBJECT([key, val[, key, val] ...])`

  Evaluates a (possibly empty) list of key-value pairs and returns a JSON object containing those pairs. An error occurs if any key name is `NULL` or the number of arguments is odd.

```
mysql> SELECT JSON_OBJECT('id', 87, 'name', 'carrot');
+---------------------------------------+
| JSON_OBJECT('id', 87, 'name', 'carrot') |
+---------------------------------------+
| {"id": 87, "name": "carrot"}            |
+---------------------------------------+
```

- `JSON_QUOTE(string)`

  Quotes a string as a JSON value by wrapping it with double quote characters and escaping interior quote and other characters, then returning the result as a `utf8mb4` string. Returns `NULL` if the argument is `NULL`.

  This function is typically used to produce a valid JSON string literal for inclusion within a JSON document.

  Certain special characters are escaped with backslashes per the escape sequences shown in Table 12.22, “JSON_UNQUOTE() Special Character Escape Sequences”.

```
mysql> SELECT JSON_QUOTE('null'), JSON_QUOTE('"null"');
+--------------------+----------------------+
| JSON_QUOTE('null') | JSON_QUOTE('"null"') |
+--------------------+----------------------+
| "null"             | "\"null\""           |
+--------------------+----------------------+
mysql> SELECT JSON_QUOTE('[1, 2, 3]');
```

```
+------------------------+
| JSON_QUOTE('[1, 2, 3]') |
+------------------------+
| "[1, 2, 3]"            |
+------------------------+
```

You can also obtain JSON values by casting values of other types to the `JSON` type using `CAST(value AS JSON)`; see Converting between JSON and non-JSON values, for more information.

Two aggregate functions generating JSON values are available. `JSON_ARRAYAGG()` returns a result set as a single JSON array, and `JSON_OBJECTAGG()` returns a result set as a single JSON object. For more information, see Section 12.20, “Aggregate (GROUP BY) Functions”.

12.17.3 Functions That Search JSON Values

The functions in this section perform search or comparison operations on JSON values to extract data from them, report whether data exists at a location within them, or report the path to data within them. The `MEMBER OF()` operator is also documented herein.

- `JSON_CONTAINS(target, candidate[, path])`

  Indicates by returning 1 or 0 whether a given *`candidate`* JSON document is contained within a *`target`* JSON document, or—if a *`path`* argument was supplied—whether the candidate is found at a specific path within the target. Returns `NULL` if any argument is `NULL`, or if the path argument does not identify a section of the target document. An error occurs if *`target`* or *`candidate`* is not a valid JSON document, or if the *`path`* argument is not a valid path expression or contains a `*` or `**` wildcard.

  To check only whether any data exists at the path, use `JSON_CONTAINS_PATH()` instead.

  The following rules define containment:

  - A candidate scalar is contained in a target scalar if and only if they are comparable and are equal. Two scalar values are comparable if they have the same `JSON_TYPE()` types, with the exception that values of types `INTEGER` and `DECIMAL` are also comparable to each other.
  - A candidate array is contained in a target array if and only if every element in the candidate is contained in some element of the target.
  - A candidate nonarray is contained in a target array if and only if the candidate is contained in some element of the target.
  - A candidate object is contained in a target object if and only if for each key in the candidate there is a key with the same name in the target and the value associated with the candidate key is contained in the value associated with the target key.

  Otherwise, the candidate value is not contained in the target document.

  Starting with MySQL 8.0.17, queries using `JSON_CONTAINS()` on `InnoDB` tables can be optimized using multi-valued indexes; see Multi-Valued Indexes, for more information.

```
mysql> SET @j = '{"a": 1, "b": 2, "c": {"d": 4}}';
mysql> SET @j2 = '1';
mysql> SELECT JSON_CONTAINS(@j, @j2, '$.a');
+-------------------------------+
| JSON_CONTAINS(@j, @j2, '$.a') |
+-------------------------------+
|                             1 |
+-------------------------------+
```

```
mysql> SELECT JSON_CONTAINS(@j, @j2, '$.b');
+-------------------------------+
| JSON_CONTAINS(@j, @j2, '$.b') |
+-------------------------------+
|                             0 |
+-------------------------------+

mysql> SET @j2 = '{"d": 4}';
mysql> SELECT JSON_CONTAINS(@j, @j2, '$.a');
+-------------------------------+
| JSON_CONTAINS(@j, @j2, '$.a') |
+-------------------------------+
|                             0 |
+-------------------------------+
mysql> SELECT JSON_CONTAINS(@j, @j2, '$.c');
+-------------------------------+
| JSON_CONTAINS(@j, @j2, '$.c') |
+-------------------------------+
|                             1 |
+-------------------------------+
```

- `JSON_CONTAINS_PATH(`*`json_doc`*`, `*`one_or_all`*`, `*`path`*`[, `*`path`*`] ...)`

  Returns 0 or 1 to indicate whether a JSON document contains data at a given path or paths. Returns `NULL` if any argument is `NULL`. An error occurs if the *`json_doc`* argument is not a valid JSON document, any *`path`* argument is not a valid path expression, or *`one_or_all`* is not `'one'` or `'all'`.

  To check for a specific value at a path, use `JSON_CONTAINS()` instead.

  The return value is 0 if no specified path exists within the document. Otherwise, the return value depends on the *`one_or_all`* argument:

  - `'one'`: 1 if at least one path exists within the document, 0 otherwise.
  - `'all'`: 1 if all paths exist within the document, 0 otherwise.

```
mysql> SET @j = '{"a": 1, "b": 2, "c": {"d": 4}}';
mysql> SELECT JSON_CONTAINS_PATH(@j, 'one', '$.a', '$.e');
+-------------------------------------------+
| JSON_CONTAINS_PATH(@j, 'one', '$.a', '$.e') |
+-------------------------------------------+
|                                         1 |
+-------------------------------------------+
mysql> SELECT JSON_CONTAINS_PATH(@j, 'all', '$.a', '$.e');
+-------------------------------------------+
| JSON_CONTAINS_PATH(@j, 'all', '$.a', '$.e') |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+
mysql> SELECT JSON_CONTAINS_PATH(@j, 'one', '$.c.d');
+---------------------------------------+
| JSON_CONTAINS_PATH(@j, 'one', '$.c.d') |
+---------------------------------------+
|                                     1 |
+---------------------------------------+
mysql> SELECT JSON_CONTAINS_PATH(@j, 'one', '$.a.d');
+---------------------------------------+
| JSON_CONTAINS_PATH(@j, 'one', '$.a.d') |
+---------------------------------------+
|                                     0 |
+---------------------------------------+
```

- `JSON_EXTRACT(`*`json_doc`*`, `*`path`*`[, `*`path`*`] ...)`

Returns data from a JSON document, selected from the parts of the document matched by the *`path`* arguments. Returns `NULL` if any argument is `NULL` or no paths locate a value in the document. An error occurs if the *`json_doc`* argument is not a valid JSON document or any *`path`* argument is not a valid path expression.

The return value consists of all values matched by the *`path`* arguments. If it is possible that those arguments could return multiple values, the matched values are autowrapped as an array, in the order corresponding to the paths that produced them. Otherwise, the return value is the single matched value.

```
mysql> SELECT JSON_EXTRACT('[10, 20, [30, 40]]', '$[1]');
+-----------------------------------------+
| JSON_EXTRACT('[10, 20, [30, 40]]', '$[1]') |
+-----------------------------------------+
| 20                                      |
+-----------------------------------------+
mysql> SELECT JSON_EXTRACT('[10, 20, [30, 40]]', '$[1]', '$[0]');
+--------------------------------------------------+
| JSON_EXTRACT('[10, 20, [30, 40]]', '$[1]', '$[0]') |
+--------------------------------------------------+
| [20, 10]                                         |
+--------------------------------------------------+
mysql> SELECT JSON_EXTRACT('[10, 20, [30, 40]]', '$[2][*]');
+-----------------------------------------+
| JSON_EXTRACT('[10, 20, [30, 40]]', '$[2][*]') |
+-----------------------------------------+
| [30, 40]                                |
+-----------------------------------------+
```

MySQL supports the `->` operator as shorthand for this function as used with 2 arguments where the left hand side is a `JSON` column identifier (not an expression) and the right hand side is the JSON path to be matched within the column.

- *`column->path`*

  The `->` operator serves as an alias for the `JSON_EXTRACT()` function when used with two arguments, a column identifier on the left and a JSON path on the right that is evaluated against the JSON document (the column value). You can use such expressions in place of column identifiers wherever they occur in SQL statements.

  The two `SELECT` statements shown here produce the same output:

```
mysql> SELECT c, JSON_EXTRACT(c, "$.id"), g
     > FROM jemp
     > WHERE JSON_EXTRACT(c, "$.id") > 1
     > ORDER BY JSON_EXTRACT(c, "$.name");
+------------------------------+-----------+------+
| c                            | c->"$.id" | g    |
+------------------------------+-----------+------+
| {"id": "3", "name": "Barney"} | "3"       |    3 |
| {"id": "4", "name": "Betty"}  | "4"       |    4 |
| {"id": "2", "name": "Wilma"}  | "2"       |    2 |
+------------------------------+-----------+------+
3 rows in set (0.00 sec)

mysql> SELECT c, c->"$.id", g
     > FROM jemp
     > WHERE c->"$.id" > 1
     > ORDER BY c->"$.name";
+------------------------------+-----------+------+
| c                            | c->"$.id" | g    |
```

```
+------------------------------+-----------+------+
| {"id": "3", "name": "Barney"} | "3"       |    3 |
| {"id": "4", "name": "Betty"}  | "4"       |    4 |
| {"id": "2", "name": "Wilma"}  | "2"       |    2 |
+------------------------------+-----------+------+
3 rows in set (0.00 sec)
```

This functionality is not limited to `SELECT`, as shown here:

```
mysql> ALTER TABLE jemp ADD COLUMN n INT;
Query OK, 0 rows affected (0.68 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> UPDATE jemp SET n=1 WHERE c->"$.id" = "4";
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT c, c->"$.id", g, n
     > FROM jemp
     > WHERE JSON_EXTRACT(c, "$.id") > 1
     > ORDER BY c->"$.name";
+------------------------------+-----------+------+------+
| c                            | c->"$.id" | g    | n    |
+------------------------------+-----------+------+------+
| {"id": "3", "name": "Barney"} | "3"       |    3 | NULL |
| {"id": "4", "name": "Betty"}  | "4"       |    4 |    1 |
| {"id": "2", "name": "Wilma"}  | "2"       |    2 | NULL |
+------------------------------+-----------+------+------+
3 rows in set (0.00 sec)

mysql> DELETE FROM jemp WHERE c->"$.id" = "4";
Query OK, 1 row affected (0.04 sec)

mysql> SELECT c, c->"$.id", g, n
     > FROM jemp
     > WHERE JSON_EXTRACT(c, "$.id") > 1
     > ORDER BY c->"$.name";
+------------------------------+-----------+------+------+
| c                            | c->"$.id" | g    | n    |
+------------------------------+-----------+------+------+
| {"id": "3", "name": "Barney"} | "3"       |    3 | NULL |
| {"id": "2", "name": "Wilma"}  | "2"       |    2 | NULL |
+------------------------------+-----------+------+------+
2 rows in set (0.00 sec)
```

(See Indexing a Generated Column to Provide a JSON Column Index, for the statements used to create and populate the table just shown.)

This also works with JSON array values, as shown here:

```
mysql> CREATE TABLE tj10 (a JSON, b INT);
Query OK, 0 rows affected (0.26 sec)

mysql> INSERT INTO tj10
     > VALUES ("[3,10,5,17,44]", 33), ("[3,10,5,17,[22,44,66]]", 0);
Query OK, 1 row affected (0.04 sec)

mysql> SELECT a->"$[4]" FROM tj10;
+--------------+
| a->"$[4]"    |
+--------------+
| 44           |
| [22, 44, 66] |
+--------------+
```

```
2 rows in set (0.00 sec)

mysql> SELECT * FROM tj10 WHERE a->"$[0]" = 3;
+-----------------------------+------+
| a                           | b    |
+-----------------------------+------+
| [3, 10, 5, 17, 44]          |   33 |
| [3, 10, 5, 17, [22, 44, 66]] |    0 |
+-----------------------------+------+
2 rows in set (0.00 sec)
```

Nested arrays are supported. An expression using `->` evaluates as `NULL` if no matching key is found in the target JSON document, as shown here:

```
mysql> SELECT * FROM tj10 WHERE a->"$[4][1]" IS NOT NULL;
+-----------------------------+------+
| a                           | b    |
+-----------------------------+------+
| [3, 10, 5, 17, [22, 44, 66]] |    0 |
+-----------------------------+------+

mysql> SELECT a->"$[4][1]" FROM tj10;
+--------------+
| a->"$[4][1]" |
+--------------+
| NULL         |
| 44           |
+--------------+
2 rows in set (0.00 sec)
```

This is the same behavior as seen in such cases when using `JSON_EXTRACT()`:

```
mysql> SELECT JSON_EXTRACT(a, "$[4][1]") FROM tj10;
+----------------------------+
| JSON_EXTRACT(a, "$[4][1]") |
+----------------------------+
| NULL                       |
| 44                         |
+----------------------------+
2 rows in set (0.00 sec)
```

- *`column->>path`*

  This is an improved, unquoting extraction operator. Whereas the `->` operator simply extracts a value, the `->>` operator in addition unquotes the extracted result. In other words, given a `JSON` column value *`column`* and a path expression *`path`*, the following three expressions return the same value:

  - `JSON_UNQUOTE( JSON_EXTRACT(`*`column, path`*`) )`
  - `JSON_UNQUOTE(`*`column -> path`*`)`
  - *`column->>path`*

  The `->>` operator can be used wherever `JSON_UNQUOTE(JSON_EXTRACT())` would be allowed. This includes (but is not limited to) `SELECT` lists, `WHERE` and `HAVING` clauses, and `ORDER BY` and `GROUP BY` clauses.

  The next few statements demonstrate some `->>` operator equivalences with other expressions in the `mysql` client:

```
mysql> SELECT * FROM jemp WHERE g > 2;
+------------------------------+------+
| c                            | g    |
+------------------------------+------+
| {"id": "3", "name": "Barney"} |    3 |
| {"id": "4", "name": "Betty"}  |    4 |
+------------------------------+------+
2 rows in set (0.01 sec)

mysql> SELECT c->'$.name' AS name
    ->     FROM jemp WHERE g > 2;
+----------+
| name     |
+----------+
| "Barney" |
| "Betty"  |
+----------+
2 rows in set (0.00 sec)

mysql> SELECT JSON_UNQUOTE(c->'$.name') AS name
    ->     FROM jemp WHERE g > 2;
+--------+
| name   |
+--------+
| Barney |
| Betty  |
+--------+
2 rows in set (0.00 sec)

mysql> SELECT c->>'$.name' AS name
    ->     FROM jemp WHERE g > 2;
+--------+
| name   |
+--------+
| Barney |
| Betty  |
+--------+
2 rows in set (0.00 sec)
```

See Indexing a Generated Column to Provide a JSON Column Index, for the SQL statements used to create and populate the `jemp` table in the set of examples just shown.

This operator can also be used with JSON arrays, as shown here:

```
mysql> CREATE TABLE tj10 (a JSON, b INT);
Query OK, 0 rows affected (0.26 sec)

mysql> INSERT INTO tj10 VALUES
    ->     ('[3,10,5,"x",44]', 33),
    ->     ('[3,10,5,17,[22,"y",66]]', 0);
Query OK, 2 rows affected (0.04 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT a->"$[3]", a->"$[4][1]" FROM tj10;
+-----------+--------------+
| a->"$[3]" | a->"$[4][1]" |
+-----------+--------------+
| "x"       | NULL         |
| 17        | "y"          |
+-----------+--------------+
2 rows in set (0.00 sec)

mysql> SELECT a->>"$[3]", a->>"$[4][1]" FROM tj10;
+------------+---------------+
| a->>"$[3]" | a->>"$[4][1]" |
```

```
+------------+--------------+
| x          | NULL         |
| 17         | y            |
+------------+--------------+
2 rows in set (0.00 sec)
```

As with `->`, the `->>` operator is always expanded in the output of `EXPLAIN`, as the following example demonstrates:

```
mysql> EXPLAIN SELECT c->>'$.name' AS name
    ->     FROM jemp WHERE g > 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: jemp
   partitions: NULL
         type: range
possible_keys: i
          key: i
      key_len: 5
          ref: NULL
         rows: 2
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select
json_unquote(json_extract(`jtest`.`jemp`.`c`,'$.name')) AS `name` from
`jtest`.`jemp` where (`jtest`.`jemp`.`g` > 2)
1 row in set (0.00 sec)
```

This is similar to how MySQL expands the `->` operator in the same circumstances.

- `JSON_KEYS(`*`json_doc`*`[, `*`path`*`])`

  Returns the keys from the top-level value of a JSON object as a JSON array, or, if a *`path`* argument is given, the top-level keys from the selected path. Returns `NULL` if any argument is `NULL`, the *`json_doc`* argument is not an object, or *`path`*, if given, does not locate an object. An error occurs if the *`json_doc`* argument is not a valid JSON document or the *`path`* argument is not a valid path expression or contains a `*` or `**` wildcard.

  The result array is empty if the selected object is empty. If the top-level value has nested subobjects, the return value does not include keys from those subobjects.

  ```
  mysql> SELECT JSON_KEYS('{"a": 1, "b": {"c": 30}}');
  +---------------------------------------+
  | JSON_KEYS('{"a": 1, "b": {"c": 30}}') |
  +---------------------------------------+
  | ["a", "b"]                            |
  +---------------------------------------+
  mysql> SELECT JSON_KEYS('{"a": 1, "b": {"c": 30}}', '$.b');
  +--------------------------------------------+
  | JSON_KEYS('{"a": 1, "b": {"c": 30}}', '$.b') |
  +--------------------------------------------+
  | ["c"]                                      |
  +--------------------------------------------+
  ```

- `JSON_OVERLAPS(`*`json_doc1, json_doc2`*`)`

Compares two JSON documents. Returns true (1) if the two document have any key-value pairs or array elements in common. If both arguments are scalars, the function performs a simple equality test.

This function serves as counterpart to `JSON_CONTAINS()`, which requires all elements of the array searched for to be present in the array searched in. Thus, `JSON_CONTAINS()` performs an `AND` operation on search keys, while `JSON_OVERLAPS()` performs an `OR` operation.

Queries on JSON columns of `InnoDB` tables using `JSON_OVERLAPS()` in the `WHERE` clause can be optimized using multi-valued indexes. Multi-Valued Indexes, provides detailed information and examples.

When two comparing two arrays, `JSON_OVERLAPS()` returns true if they share one or more array elements in common, and false if they do not:

```
mysql> SELECT JSON_OVERLAPS("[1,3,5,7]", "[2,5,7]");
+---------------------------------------+
| JSON_OVERLAPS("[1,3,5,7]", "[2,5,7]") |
+---------------------------------------+
|                                     1 |
+---------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_OVERLAPS("[1,3,5,7]", "[2,6,7]");
+---------------------------------------+
| JSON_OVERLAPS("[1,3,5,7]", "[2,6,7]") |
+---------------------------------------+
|                                     1 |
+---------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_OVERLAPS("[1,3,5,7]", "[2,6,8]");
+---------------------------------------+
| JSON_OVERLAPS("[1,3,5,7]", "[2,6,8]") |
+---------------------------------------+
|                                     0 |
+---------------------------------------+
1 row in set (0.00 sec)
```

Partial matches are treated as no match, as shown here:

```
mysql> SELECT JSON_OVERLAPS('[[1,2],[3,4],5]', '[1,[2,3],[4,5]]');
+-----------------------------------------------------+
| JSON_OVERLAPS('[[1,2],[3,4],5]', '[1,[2,3],[4,5]]') |
+-----------------------------------------------------+
|                                                   0 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

When comparing objects, the result is true if they have at least one key-value pair in common.

```
mysql> SELECT JSON_OVERLAPS('{"a":1,"b":10,"d":10}', '{"c":1,"e":10,"f":1,"d":10}');
+-------------------------------------------------------------------------+
| JSON_OVERLAPS('{"a":1,"b":10,"d":10}', '{"c":1,"e":10,"f":1,"d":10}') |
+-------------------------------------------------------------------------+
|                                                                       1 |
+-------------------------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_OVERLAPS('{"a":1,"b":10,"d":10}', '{"a":5,"e":10,"f":1,"d":20}');
+-------------------------------------------------------------------------+
| JSON_OVERLAPS('{"a":1,"b":10,"d":10}', '{"a":5,"e":10,"f":1,"d":20}') |
```

```
+-----------------------------------------------------------------------------+
|                                                                           0 |
+-----------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

If two scalars are used as the arguments to the function, `JSON_OVERLAPS()` performs a simple test for equality:

```
mysql> SELECT JSON_OVERLAPS('5', '5');
+-------------------------+
| JSON_OVERLAPS('5', '5') |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_OVERLAPS('5', '6');
+-------------------------+
| JSON_OVERLAPS('5', '6') |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set (0.00 sec)
```

When comparing a scalar with an array, `JSON_OVERLAPS()` attempts to treat the scalar as an array element. In this example, the second argument `6` is interpreted as `[6]`, as shown here:

```
mysql> SELECT JSON_OVERLAPS('[4,5,6,7]', '6');
+---------------------------------+
| JSON_OVERLAPS('[4,5,6,7]', '6') |
+---------------------------------+
|                               1 |
+---------------------------------+
1 row in set (0.00 sec)
```

The function does not perform type conversions:

```
mysql> SELECT JSON_OVERLAPS('[4,5,"6",7]', '6');
+----------------------------------+
| JSON_OVERLAPS('[4,5,"6",7]', '6') |
+----------------------------------+
|                                0 |
+----------------------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_OVERLAPS('[4,5,6,7]', '"6"');
+----------------------------------+
| JSON_OVERLAPS('[4,5,6,7]', '"6"') |
+----------------------------------+
|                                0 |
+----------------------------------+
1 row in set (0.00 sec)
```

`JSON_OVERLAPS()` was added in MySQL 8.0.17.

- `JSON_SEARCH(`*`json_doc`*`, `*`one_or_all`*`, `*`search_str`*`[, `*`escape_char`*`[, `*`path`*`] ...])`

  Returns the path to the given string within a JSON document. Returns `NULL` if any of the *`json_doc`*, *`search_str`*, or *`path`* arguments are `NULL`; no *`path`* exists within the document; or *`search_str`* is not found. An error occurs if the *`json_doc`* argument is not a valid JSON document, any *`path`* argument is

not a valid path expression, *one_or_all* is not 'one' or 'all', or *escape_char* is not a constant expression.

The *one_or_all* argument affects the search as follows:

- 'one': The search terminates after the first match and returns one path string. It is undefined which match is considered first.
- 'all': The search returns all matching path strings such that no duplicate paths are included. If there are multiple strings, they are autowrapped as an array. The order of the array elements is undefined.

Within the *search_str* search string argument, the % and _ characters work as for the LIKE operator: % matches any number of characters (including zero characters), and _ matches exactly one character.

To specify a literal % or _ character in the search string, precede it by the escape character. The default is \ if the *escape_char* argument is missing or NULL. Otherwise, *escape_char* must be a constant that is empty or one character.

For more information about matching and escape character behavior, see the description of LIKE in Section 12.5.1, “String Comparison Functions and Operators”. For escape character handling, a difference from the LIKE behavior is that the escape character for JSON_SEARCH() must evaluate to a constant at compile time, not just at execution time. For example, if JSON_SEARCH() is used in a prepared statement and the *escape_char* argument is supplied using a ? parameter, the parameter value might be constant at execution time, but is not at compile time.

```
mysql> SET @j = '["abc", [{"k": "10"}, "def"], {"x":"abc"}, {"y":"bcd"}]';

mysql> SELECT JSON_SEARCH(@j, 'one', 'abc');
+--------------------------------+
| JSON_SEARCH(@j, 'one', 'abc') |
+--------------------------------+
| "$[0]"                         |
+--------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', 'abc');
+--------------------------------+
| JSON_SEARCH(@j, 'all', 'abc') |
+--------------------------------+
| ["$[0]", "$[2].x"]             |
+--------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', 'ghi');
+--------------------------------+
| JSON_SEARCH(@j, 'all', 'ghi') |
+--------------------------------+
| NULL                           |
+--------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '10');
+--------------------------------+
| JSON_SEARCH(@j, 'all', '10') |
+--------------------------------+
| "$[1][0].k"                    |
+--------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$');
+-----------------------------------------+
| JSON_SEARCH(@j, 'all', '10', NULL, '$') |
+-----------------------------------------+
| "$[1][0].k"                             |
+-----------------------------------------+
```

```
mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$[*]');
+-----------------------------------------+
| JSON_SEARCH(@j, 'all', '10', NULL, '$[*]') |
+-----------------------------------------+
| "$[1][0].k"                             |
+-----------------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$**.k');
+-----------------------------------------+
| JSON_SEARCH(@j, 'all', '10', NULL, '$**.k') |
+-----------------------------------------+
| "$[1][0].k"                              |
+-----------------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$[*][0].k');
+-----------------------------------------------+
| JSON_SEARCH(@j, 'all', '10', NULL, '$[*][0].k') |
+-----------------------------------------------+
| "$[1][0].k"                                    |
+-----------------------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$[1]');
+-----------------------------------------+
| JSON_SEARCH(@j, 'all', '10', NULL, '$[1]') |
+-----------------------------------------+
| "$[1][0].k"                             |
+-----------------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '10', NULL, '$[1][0]');
+-----------------------------------------------+
| JSON_SEARCH(@j, 'all', '10', NULL, '$[1][0]') |
+-----------------------------------------------+
| "$[1][0].k"                                    |
+-----------------------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', 'abc', NULL, '$[2]');
+-----------------------------------------+
| JSON_SEARCH(@j, 'all', 'abc', NULL, '$[2]') |
+-----------------------------------------+
| "$[2].x"                                |
+-----------------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '%a%');
+-------------------------------+
| JSON_SEARCH(@j, 'all', '%a%') |
+-------------------------------+
| ["$[0]", "$[2].x"]            |
+-------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%');
+-------------------------------+
| JSON_SEARCH(@j, 'all', '%b%') |
+-------------------------------+
| ["$[0]", "$[2].x", "$[3].y"]  |
+-------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%', NULL, '$[0]');
+-----------------------------------------+
| JSON_SEARCH(@j, 'all', '%b%', NULL, '$[0]') |
+-----------------------------------------+
| "$[0]"                                  |
+-----------------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%', NULL, '$[2]');
+-----------------------------------------+
| JSON_SEARCH(@j, 'all', '%b%', NULL, '$[2]') |
```

```
+-----------------------------------------+
| "$[2].x"                                |
+-----------------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%', NULL, '$[1]');
+-----------------------------------------+
| JSON_SEARCH(@j, 'all', '%b%', NULL, '$[1]') |
+-----------------------------------------+
| NULL                                    |
+-----------------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%', '', '$[1]');
+-----------------------------------------+
| JSON_SEARCH(@j, 'all', '%b%', '', '$[1]') |
+-----------------------------------------+
| NULL                                    |
+-----------------------------------------+

mysql> SELECT JSON_SEARCH(@j, 'all', '%b%', '', '$[3]');
+-----------------------------------------+
| JSON_SEARCH(@j, 'all', '%b%', '', '$[3]') |
+-----------------------------------------+
| "$[3].y"                                |
+-----------------------------------------+
```

For more information about the JSON path syntax supported by MySQL, including rules governing the wildcard operators `*` and `**`, see JSON Path Syntax.

- *`value`* `MEMBER OF(`*`json_array`*`)`

  Returns true (1) if *`value`* is an element of *`json_array`*, otherwise returns false (0). *`value`* must be a scalar or a JSON document; if it is a scalar, the operator attempts to treat it as an element of a JSON array.

  Queries using `MEMBER OF()` on JSON columns of `InnoDB` tables in the `WHERE` clause can be optimized using multi-valued indexes. See Multi-Valued Indexes, for detailed information and examples.

  Simple scalars are treated as array values, as shown here:

```
mysql> SELECT 17 MEMBER OF('[23, "abc", 17, "ab", 10]');
+-----------------------------------------+
| 17 MEMBER OF('[23, "abc", 17, "ab", 10]') |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT 'ab' MEMBER OF('[23, "abc", 17, "ab", 10]');
+-----------------------------------------+
| 'ab' MEMBER OF('[23, "abc", 17, "ab", 10]') |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
1 row in set (0.00 sec)
```

  Partial matches of array element values do not match:

```
mysql> SELECT 7 MEMBER OF('[23, "abc", 17, "ab", 10]');
+-----------------------------------------+
| 7 MEMBER OF('[23, "abc", 17, "ab", 10]') |
+-----------------------------------------+
|                                       0 |
```

```
+----------------------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT 'a' MEMBER OF('[23, "abc", 17, "ab", 10]');
+-------------------------------------------+
| 'a' MEMBER OF('[23, "abc", 17, "ab", 10]') |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+
1 row in set (0.00 sec)
```

Conversions to and from string types are not performed:

```
mysql> SELECT
    -> 17 MEMBER OF('[23, "abc", "17", "ab", 10]'),
    -> "17" MEMBER OF('[23, "abc", 17, "ab", 10]')\G
*************************** 1. row ***************************
17 MEMBER OF('[23, "abc", "17", "ab", 10]'): 0
"17" MEMBER OF('[23, "abc", 17, "ab", 10]'): 0
1 row in set (0.00 sec)
```

To use this operator with a value which itself an array, it is necessary to cast it explicitly as a JSON array. You can do this with `CAST(... AS JSON)`:

```
mysql> SELECT CAST('[4,5]' AS JSON) MEMBER OF('[[3,4],[4,5]]');
+-------------------------------------------------+
| CAST('[4,5]' AS JSON) MEMBER OF('[[3,4],[4,5]]') |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
1 row in set (0.00 sec)
```

It is also possible to perform the necessary cast using the `JSON_ARRAY()` function, like this:

```
mysql> SELECT JSON_ARRAY(4,5) MEMBER OF('[[3,4],[4,5]]');
+-----------------------------------------+
| JSON_ARRAY(4,5) MEMBER OF('[[3,4],[4,5]]') |
+-----------------------------------------+
|                                       1 |
+-----------------------------------------+
1 row in set (0.00 sec)
```

Any JSON objects used as values to be tested or which appear in the target array must be coerced to the correct type using `CAST(... AS JSON)` or `JSON_OBJECT()`. In addition, a target array containing JSON objects must itself be cast using `JSON_ARRAY`. This is demonstrated in the following sequence of statements:

```
mysql> SET @a = CAST('{"a":1}' AS JSON);
Query OK, 0 rows affected (0.00 sec)

mysql> SET @b = JSON_OBJECT("b", 2);
Query OK, 0 rows affected (0.00 sec)

mysql> SET @c = JSON_ARRAY(17, @b, "abc", @a, 23);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a MEMBER OF(@c), @b MEMBER OF(@c);
+------------------+------------------+
| @a MEMBER OF(@c) | @b MEMBER OF(@c) |
+------------------+------------------+
```

```
|                1 |                 1 |
+------------------+-------------------+
1 row in set (0.00 sec)
```

The `MEMBER OF()` operator was added in MySQL 8.0.17.

12.17.4 Functions That Modify JSON Values

The functions in this section modify JSON values and return the result.

- `JSON_ARRAY_APPEND(json_doc, path, val[, path, val] ...)`

  Appends values to the end of the indicated arrays within a JSON document and returns the result. Returns `NULL` if any argument is `NULL`. An error occurs if the *`json_doc`* argument is not a valid JSON document or any *`path`* argument is not a valid path expression or contains a `*` or `**` wildcard.

  The path-value pairs are evaluated left to right. The document produced by evaluating one pair becomes the new value against which the next pair is evaluated.

  If a path selects a scalar or object value, that value is autowrapped within an array and the new value is added to that array. Pairs for which the path does not identify any value in the JSON document are ignored.

```
mysql> SET @j = '["a", ["b", "c"], "d"]';
mysql> SELECT JSON_ARRAY_APPEND(@j, '$[1]', 1);
+---------------------------------+
| JSON_ARRAY_APPEND(@j, '$[1]', 1) |
+---------------------------------+
| ["a", ["b", "c", 1], "d"]       |
+---------------------------------+
mysql> SELECT JSON_ARRAY_APPEND(@j, '$[0]', 2);
+---------------------------------+
| JSON_ARRAY_APPEND(@j, '$[0]', 2) |
+---------------------------------+
| [["a", 2], ["b", "c"], "d"]     |
+---------------------------------+
mysql> SELECT JSON_ARRAY_APPEND(@j, '$[1][0]', 3);
+------------------------------------+
| JSON_ARRAY_APPEND(@j, '$[1][0]', 3) |
+------------------------------------+
| ["a", [["b", 3], "c"], "d"]        |
+------------------------------------+

mysql> SET @j = '{"a": 1, "b": [2, 3], "c": 4}';
mysql> SELECT JSON_ARRAY_APPEND(@j, '$.b', 'x');
+-----------------------------------+
| JSON_ARRAY_APPEND(@j, '$.b', 'x')  |
+-----------------------------------+
| {"a": 1, "b": [2, 3, "x"], "c": 4} |
+-----------------------------------+
mysql> SELECT JSON_ARRAY_APPEND(@j, '$.c', 'y');
+-------------------------------------+
| JSON_ARRAY_APPEND(@j, '$.c', 'y')    |
+-------------------------------------+
| {"a": 1, "b": [2, 3], "c": [4, "y"]} |
+-------------------------------------+

mysql> SET @j = '{"a": 1}';
mysql> SELECT JSON_ARRAY_APPEND(@j, '$', 'z');
+---------------------------------+
| JSON_ARRAY_APPEND(@j, '$', 'z') |
+---------------------------------+
| [{"a": 1}, "z"]                 |
```

```
+--------------------------------+
```

In MySQL 5.7, this function was named `JSON_APPEND()`. That name is no longer supported in MySQL 8.0.

- `JSON_ARRAY_INSERT(json_doc, path, val[, path, val] ...)`

  Updates a JSON document, inserting into an array within the document and returning the modified document. Returns `NULL` if any argument is `NULL`. An error occurs if the *`json_doc`* argument is not a valid JSON document or any *`path`* argument is not a valid path expression or contains a `*` or `**` wildcard or does not end with an array element identifier.

  The path-value pairs are evaluated left to right. The document produced by evaluating one pair becomes the new value against which the next pair is evaluated.

  Pairs for which the path does not identify any array in the JSON document are ignored. If a path identifies an array element, the corresponding value is inserted at that element position, shifting any following values to the right. If a path identifies an array position past the end of an array, the value is inserted at the end of the array.

```
mysql> SET @j = '["a", {"b": [1, 2]}, [3, 4]]';
mysql> SELECT JSON_ARRAY_INSERT(@j, '$[1]', 'x');
+-------------------------------------+
| JSON_ARRAY_INSERT(@j, '$[1]', 'x') |
+-------------------------------------+
| ["a", "x", {"b": [1, 2]}, [3, 4]]  |
+-------------------------------------+
mysql> SELECT JSON_ARRAY_INSERT(@j, '$[100]', 'x');
+-------------------------------------+
| JSON_ARRAY_INSERT(@j, '$[100]', 'x') |
+-------------------------------------+
| ["a", {"b": [1, 2]}, [3, 4], "x"]    |
+-------------------------------------+
mysql> SELECT JSON_ARRAY_INSERT(@j, '$[1].b[0]', 'x');
+---------------------------------------+
| JSON_ARRAY_INSERT(@j, '$[1].b[0]', 'x') |
+---------------------------------------+
| ["a", {"b": ["x", 1, 2]}, [3, 4]]       |
+---------------------------------------+
mysql> SELECT JSON_ARRAY_INSERT(@j, '$[2][1]', 'y');
+-------------------------------------+
| JSON_ARRAY_INSERT(@j, '$[2][1]', 'y') |
+-------------------------------------+
| ["a", {"b": [1, 2]}, [3, "y", 4]]     |
+-------------------------------------+
mysql> SELECT JSON_ARRAY_INSERT(@j, '$[0]', 'x', '$[2][1]', 'y');
+-----------------------------------------------------+
| JSON_ARRAY_INSERT(@j, '$[0]', 'x', '$[2][1]', 'y') |
+-----------------------------------------------------+
| ["x", "a", {"b": [1, 2]}, [3, 4]]                   |
+-----------------------------------------------------+
```

  Earlier modifications affect the positions of the following elements in the array, so subsequent paths in the same `JSON_ARRAY_INSERT()` call should take this into account. In the final example, the second path inserts nothing because the path no longer matches anything after the first insert.

- `JSON_INSERT(json_doc, path, val[, path, val] ...)`

  Inserts data into a JSON document and returns the result. Returns `NULL` if any argument is `NULL`. An error occurs if the *`json_doc`* argument is not a valid JSON document or any *`path`* argument is not a valid path expression or contains a `*` or `**` wildcard.

The path-value pairs are evaluated left to right. The document produced by evaluating one pair becomes the new value against which the next pair is evaluated.

A path-value pair for an existing path in the document is ignored and does not overwrite the existing document value. A path-value pair for a nonexisting path in the document adds the value to the document if the path identifies one of these types of values:

- A member not present in an existing object. The member is added to the object and associated with the new value.
- A position past the end of an existing array. The array is extended with the new value. If the existing value is not an array, it is autowrapped as an array, then extended with the new value.

Otherwise, a path-value pair for a nonexisting path in the document is ignored and has no effect.

For a comparison of `JSON_INSERT()`, `JSON_REPLACE()`, and `JSON_SET()`, see the discussion of `JSON_SET()`.

```
mysql> SET @j = '{ "a": 1, "b": [2, 3]}';
mysql> SELECT JSON_INSERT(@j, '$.a', 10, '$.c', '[true, false]');
+-------------------------------------------------+
| JSON_INSERT(@j, '$.a', 10, '$.c', '[true, false]') |
+-------------------------------------------------+
| {"a": 1, "b": [2, 3], "c": "[true, false]"}        |
+-------------------------------------------------+
```

The third and final value listed in the result is a quoted string and not an array like the second one (which is not quoted in the output); no casting of values to the JSON type is performed. To insert the array as an array, you must perform such casts explicitly, as shown here:

```
mysql> SELECT JSON_INSERT(@j, '$.a', 10, '$.c', CAST('[true, false]' AS JSON));
+-----------------------------------------------------------------+
| JSON_INSERT(@j, '$.a', 10, '$.c', CAST('[true, false]' AS JSON)) |
+-----------------------------------------------------------------+
| {"a": 1, "b": [2, 3], "c": [true, false]}                       |
+-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

- `JSON_MERGE(`*`json_doc`*`, `*`json_doc`*`[, `*`json_doc`*`] ...)`

  Merges two or more JSON documents. Synonym for `JSON_MERGE_PRESERVE()`; deprecated in MySQL 8.0.3 and subject to removal in a future release.

```
mysql> SELECT JSON_MERGE('[1, 2]', '[true, false]');
+---------------------------------------+
| JSON_MERGE('[1, 2]', '[true, false]') |
+---------------------------------------+
| [1, 2, true, false]                   |
+---------------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1287
Message: 'JSON_MERGE' is deprecated and will be removed in a future release. \
 Please use JSON_MERGE_PRESERVE/JSON_MERGE_PATCH instead
1 row in set (0.00 sec)
```

For additional examples, see the entry for `JSON_MERGE_PRESERVE()`.

- `JSON_MERGE_PATCH(json_doc, json_doc[, json_doc] ...)`

  Performs an RFC 7396 compliant merge of two or more JSON documents and returns the merged result, without preserving members having duplicate keys. Raises an error if at least one of the documents passed as arguments to this function is not valid.

  Note

  For an explanation and example of the differences between this function and `JSON_MERGE_PRESERVE()`, see JSON_MERGE_PATCH() compared with JSON_MERGE_PRESERVE().

  `JSON_MERGE_PATCH()` performs a merge as follows:

  1. If the first argument is not an object, the result of the merge is the same as if an empty object had been merged with the second argument.
  2. If the second argument is not an object, the result of the merge is the second argument.
  3. If both arguments are objects, the result of the merge is an object with the following members:
     - All members of the first object which do not have a corresponding member with the same key in the second object.
     - All members of the second object which do not have a corresponding key in the first object, and whose value is not the JSON `null` literal.
     - All members with a key that exists in both the first and the second object, and whose value in the second object is not the JSON `null` literal. The values of these members are the results of recursively merging the value in the first object with the value in the second object.

  For additional information, see Normalization, Merging, and Autowrapping of JSON Values.

```
mysql> SELECT JSON_MERGE_PATCH('[1, 2]', '[true, false]');
+-----------------------------------------+
| JSON_MERGE_PATCH('[1, 2]', '[true, false]') |
+-----------------------------------------+
| [true, false]                           |
+-----------------------------------------+

mysql> SELECT JSON_MERGE_PATCH('{"name": "x"}', '{"id": 47}');
+-------------------------------------------------+
| JSON_MERGE_PATCH('{"name": "x"}', '{"id": 47}') |
+-------------------------------------------------+
| {"id": 47, "name": "x"}                         |
+-------------------------------------------------+

mysql> SELECT JSON_MERGE_PATCH('1', 'true');
+------------------------------+
| JSON_MERGE_PATCH('1', 'true') |
+------------------------------+
| true                         |
+------------------------------+

mysql> SELECT JSON_MERGE_PATCH('[1, 2]', '{"id": 47}');
+---------------------------------------+
| JSON_MERGE_PATCH('[1, 2]', '{"id": 47}') |
```

```
+----------------------------------------+
| {"id": 47}                             |
+----------------------------------------+

mysql> SELECT JSON_MERGE_PATCH('{ "a": 1, "b":2 }',
    >     '{ "a": 3, "c":4 }');
+-----------------------------------------------------------+
| JSON_MERGE_PATCH('{ "a": 1, "b":2 }','{ "a": 3, "c":4 }') |
+-----------------------------------------------------------+
| {"a": 3, "b": 2, "c": 4}                                  |
+-----------------------------------------------------------+

mysql> SELECT JSON_MERGE_PATCH('{ "a": 1, "b":2 }','{ "a": 3, "c":4 }',
    >     '{ "a": 5, "d":6 }');
+-------------------------------------------------------------------------------------+
| JSON_MERGE_PATCH('{ "a": 1, "b":2 }','{ "a": 3, "c":4 }','{ "a": 5, "d":6 }') |
+-------------------------------------------------------------------------------------+
| {"a": 5, "b": 2, "c": 4, "d": 6}                                                    |
+-------------------------------------------------------------------------------------+
```

You can use this function to remove a member by specifying `null` as the value of the same member in the seond argument, as shown here:

```
mysql> SELECT JSON_MERGE_PATCH('{"a":1, "b":2}', '{"b":null}');
+-------------------------------------------------+
| JSON_MERGE_PATCH('{"a":1, "b":2}', '{"b":null}') |
+-------------------------------------------------+
| {"a": 1}                                        |
+-------------------------------------------------+
```

This example shows that the function operates in a recursive fashion; that is, values of members are not limited to scalars, but rather can themselves be JSON documents:

```
mysql> SELECT JSON_MERGE_PATCH('{"a":{"x":1}}', '{"a":{"y":2}}');
+-------------------------------------------------+
| JSON_MERGE_PATCH('{"a":{"x":1}}', '{"a":{"y":2}}') |
+-------------------------------------------------+
| {"a": {"x": 1, "y": 2}}                         |
+-------------------------------------------------+
```

`JSON_MERGE_PATCH()` is supported in MySQL 8.0.3 and later.

JSON_MERGE_PATCH() compared with JSON_MERGE_PRESERVE(). The behavior of `JSON_MERGE_PATCH()` is the same as that of `JSON_MERGE_PRESERVE()`, with the following two exceptions:

- `JSON_MERGE_PATCH()` removes any member in the first object with a matching key in the second object, provided that the value associated with the key in the second object is not JSON `null`.
- If the second object has a member with a key matching a member in the first object, `JSON_MERGE_PATCH()` *replaces* the value in the first object with the value in the second object, whereas `JSON_MERGE_PRESERVE()` *appends* the second value to the first value.

This example compares the results of merging the same 3 JSON objects, each having a matching key `"a"`, with each of these two functions:

```
mysql> SET @x = '{ "a": 1, "b": 2 }',
    >     @y = '{ "a": 3, "c": 4 }',
    >     @z = '{ "a": 5, "d": 6 }';
```

```
mysql> SELECT  JSON_MERGE_PATCH(@x, @y, @z)    AS Patch,
    ->         JSON_MERGE_PRESERVE(@x, @y, @z) AS Preserve\G
*************************** 1. row ***************************
   Patch: {"a": 5, "b": 2, "c": 4, "d": 6}
Preserve: {"a": [1, 3, 5], "b": 2, "c": 4, "d": 6}
```

- `JSON_MERGE_PRESERVE(json_doc, json_doc[, json_doc] ...)`

  Merges two or more JSON documents and returns the merged result. Returns `NULL` if any argument is `NULL`. An error occurs if any argument is not a valid JSON document.

  Merging takes place according to the following rules. For additional information, see Normalization, Merging, and Autowrapping of JSON Values.

  - Adjacent arrays are merged to a single array.
  - Adjacent objects are merged to a single object.
  - A scalar value is autowrapped as an array and merged as an array.
  - An adjacent array and object are merged by autowrapping the object as an array and merging the two arrays.

```
mysql> SELECT JSON_MERGE_PRESERVE('[1, 2]', '[true, false]');
+-----------------------------------------------+
| JSON_MERGE_PRESERVE('[1, 2]', '[true, false]') |
+-----------------------------------------------+
| [1, 2, true, false]                           |
+-----------------------------------------------+

mysql> SELECT JSON_MERGE_PRESERVE('{"name": "x"}', '{"id": 47}');
+-------------------------------------------------------+
| JSON_MERGE_PRESERVE('{"name": "x"}', '{"id": 47}') |
+-------------------------------------------------------+
| {"id": 47, "name": "x"}                               |
+-------------------------------------------------------+

mysql> SELECT JSON_MERGE_PRESERVE('1', 'true');
+---------------------------------+
| JSON_MERGE_PRESERVE('1', 'true') |
+---------------------------------+
| [1, true]                       |
+---------------------------------+

mysql> SELECT JSON_MERGE_PRESERVE('[1, 2]', '{"id": 47}');
+-------------------------------------------+
| JSON_MERGE_PRESERVE('[1, 2]', '{"id": 47}') |
+-------------------------------------------+
| [1, 2, {"id": 47}]                        |
+-------------------------------------------+

mysql> SELECT JSON_MERGE_PRESERVE('{ "a": 1, "b": 2 }',
    >    '{ "a": 3, "c": 4 }');
+-----------------------------------------------------------------+
| JSON_MERGE_PRESERVE('{ "a": 1, "b": 2 }','{ "a": 3, "c":4 }') |
+-----------------------------------------------------------------+
| {"a": [1, 3], "b": 2, "c": 4}                                   |
+-----------------------------------------------------------------+

mysql> SELECT JSON_MERGE_PRESERVE('{ "a": 1, "b": 2 }','{ "a": 3, "c": 4 }',
    >    '{ "a": 5, "d": 6 }');
+-------------------------------------------------------------------------------------+
| JSON_MERGE_PRESERVE('{ "a": 1, "b": 2 }','{ "a": 3, "c": 4 }','{ "a": 5, "d": 6 }') |
```

```
+-------------------------------------------------------------------------------+
| {"a": [1, 3, 5], "b": 2, "c": 4, "d": 6}                                      |
+-------------------------------------------------------------------------------+
```

This function was added in MySQL 8.0.3 as a synonym for `JSON_MERGE()`. The `JSON_MERGE()` function is now deprecated, and is subject to removal in a future release of MySQL.

This function is similar to but differs from `JSON_MERGE_PATCH()` in significant respects; see JSON_MERGE_PATCH() compared with JSON_MERGE_PRESERVE(), for more information.

- `JSON_REMOVE(`*`json_doc`*`, `*`path`*`[, `*`path`*`] ...)`

  Removes data from a JSON document and returns the result. Returns `NULL` if any argument is `NULL`. An error occurs if the *`json_doc`* argument is not a valid JSON document or any *`path`* argument is not a valid path expression or is `$` or contains a `*` or `**` wildcard.

  The *`path`* arguments are evaluated left to right. The document produced by evaluating one path becomes the new value against which the next path is evaluated.

  It is not an error if the element to be removed does not exist in the document; in that case, the path does not affect the document.

```
mysql> SET @j = '["a", ["b", "c"], "d"]';
mysql> SELECT JSON_REMOVE(@j, '$[1]');
+-------------------------+
| JSON_REMOVE(@j, '$[1]') |
+-------------------------+
| ["a", "d"]              |
+-------------------------+
```

- `JSON_REPLACE(`*`json_doc`*`, `*`path`*`, `*`val`*`[, `*`path`*`, `*`val`*`] ...)`

  Replaces existing values in a JSON document and returns the result. Returns `NULL` if any argument is `NULL`. An error occurs if the *`json_doc`* argument is not a valid JSON document or any *`path`* argument is not a valid path expression or contains a `*` or `**` wildcard.

  The path-value pairs are evaluated left to right. The document produced by evaluating one pair becomes the new value against which the next pair is evaluated.

  A path-value pair for an existing path in the document overwrites the existing document value with the new value. A path-value pair for a nonexisting path in the document is ignored and has no effect.

  In MySQL 8.0.4, the optimizer can perform a partial, in-place update of a `JSON` column instead of removing the old document and writing the new document in its entirety to the column. This optimization can be performed for an update statement that uses the `JSON_REPLACE()` function and meets the conditions outlined in Partial Updates of JSON Values.

  For a comparison of `JSON_INSERT()`, `JSON_REPLACE()`, and `JSON_SET()`, see the discussion of `JSON_SET()`.

```
mysql> SET @j = '{ "a": 1, "b": [2, 3]}';
mysql> SELECT JSON_REPLACE(@j, '$.a', 10, '$.c', '[true, false]');
+-----------------------------------------------------+
| JSON_REPLACE(@j, '$.a', 10, '$.c', '[true, false]') |
+-----------------------------------------------------+
| {"a": 10, "b": [2, 3]}                              |
+-----------------------------------------------------+
```

- `JSON_SET(json_doc, path, val[, path, val] ...)`

  Inserts or updates data in a JSON document and returns the result. Returns `NULL` if any argument is `NULL` or *`path`*, if given, does not locate an object. An error occurs if the *`json_doc`* argument is not a valid JSON document or any *`path`* argument is not a valid path expression or contains a `*` or `**` wildcard.

  The path-value pairs are evaluated left to right. The document produced by evaluating one pair becomes the new value against which the next pair is evaluated.

  A path-value pair for an existing path in the document overwrites the existing document value with the new value. A path-value pair for a nonexisting path in the document adds the value to the document if the path identifies one of these types of values:

  - A member not present in an existing object. The member is added to the object and associated with the new value.
  - A position past the end of an existing array. The array is extended with the new value. If the existing value is not an array, it is autowrapped as an array, then extended with the new value.

  Otherwise, a path-value pair for a nonexisting path in the document is ignored and has no effect.

  In MySQL 8.0.4, the optimizer can perform a partial, in-place update of a `JSON` column instead of removing the old document and writing the new document in its entirety to the column. This optimization can be performed for an update statement that uses the `JSON_SET()` function and meets the conditions outlined in Partial Updates of JSON Values.

  The `JSON_SET()`, `JSON_INSERT()`, and `JSON_REPLACE()` functions are related:

  - `JSON_SET()` replaces existing values and adds nonexisting values.
  - `JSON_INSERT()` inserts values without replacing existing values.
  - `JSON_REPLACE()` replaces *only* existing values.

  The following examples illustrate these differences, using one path that does exist in the document (`$.a`) and another that does not exist (`$.c`):

```
mysql> SET @j = '{ "a": 1, "b": [2, 3]}';
mysql> SELECT JSON_SET(@j, '$.a', 10, '$.c', '[true, false]');
+----------------------------------------------+
| JSON_SET(@j, '$.a', 10, '$.c', '[true, false]') |
+----------------------------------------------+
| {"a": 10, "b": [2, 3], "c": "[true, false]"}    |
+----------------------------------------------+
mysql> SELECT JSON_INSERT(@j, '$.a', 10, '$.c', '[true, false]');
+-----------------------------------------------+
| JSON_INSERT(@j, '$.a', 10, '$.c', '[true, false]') |
+-----------------------------------------------+
| {"a": 1, "b": [2, 3], "c": "[true, false]"}        |
+-----------------------------------------------+
mysql> SELECT JSON_REPLACE(@j, '$.a', 10, '$.c', '[true, false]');
+------------------------------------------------+
| JSON_REPLACE(@j, '$.a', 10, '$.c', '[true, false]') |
+------------------------------------------------+
| {"a": 10, "b": [2, 3]}                          |
+------------------------------------------------+
```

- `JSON_UNQUOTE(json_val)`

Unquotes JSON value and returns the result as a `utf8mb4` string. Returns `NULL` if the argument is `NULL`. An error occurs if the value starts and ends with double quotes but is not a valid JSON string literal.

Within a string, certain sequences have special meaning unless the `NO_BACKSLASH_ESCAPES` SQL mode is enabled. Each of these sequences begins with a backslash (`\`), known as the *escape character*. MySQL recognizes the escape sequences shown in Table 12.22, "JSON_UNQUOTE() Special Character Escape Sequences". For all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped. For example, `\x` is just `x`. These sequences are case-sensitive. For example, `\b` is interpreted as a backspace, but `\B` is interpreted as `B`.

Table 12.22 JSON_UNQUOTE() Special Character Escape Sequences

Escape Sequence	Character Represented by Sequence
`\"`	A double quote (") character
`\b`	A backspace character
`\f`	A formfeed character
`\n`	A newline (linefeed) character
`\r`	A carriage return character
`\t`	A tab character
`\\`	A backslash (\) character
`\uXXXX`	UTF-8 bytes for Unicode value *XXXX*

Two simple examples of the use of this function are shown here:

```
mysql> SET @j = '"abc"';
mysql> SELECT @j, JSON_UNQUOTE(@j);
+-------+------------------+
| @j    | JSON_UNQUOTE(@j) |
+-------+------------------+
| "abc" | abc              |
+-------+------------------+
mysql> SET @j = '[1, 2, 3]';
mysql> SELECT @j, JSON_UNQUOTE(@j);
+-----------+------------------+
| @j        | JSON_UNQUOTE(@j) |
+-----------+------------------+
| [1, 2, 3] | [1, 2, 3]        |
+-----------+------------------+
```

The following set of examples shows how `JSON_UNQUOTE` handles escapes with `NO_BACKSLASH_ESCAPES` disabled and enabled:

```
mysql> SELECT @@sql_mode;
+------------+
| @@sql_mode |
+------------+
|            |
+------------+

mysql> SELECT JSON_UNQUOTE('"\\t\\u0032"');
+------------------------------+
| JSON_UNQUOTE('"\\t\\u0032"') |
```

```
+-----------------------------+
|       2                     |
+-----------------------------+

mysql> SET @@sql_mode = 'NO_BACKSLASH_ESCAPES';
mysql> SELECT JSON_UNQUOTE('"\\t\\u0032"');
+-----------------------------+
| JSON_UNQUOTE('"\\t\\u0032"') |
+-----------------------------+
| \t\u0032                    |
+-----------------------------+

mysql> SELECT JSON_UNQUOTE('"\t\u0032"');
+---------------------------+
| JSON_UNQUOTE('"\t\u0032"') |
+---------------------------+
|       2                       |
+---------------------------+
```

12.17.5 Functions That Return JSON Value Attributes

The functions in this section return attributes of JSON values.

- `JSON_DEPTH(`*`json_doc`*`)`

  Returns the maximum depth of a JSON document. Returns `NULL` if the argument is `NULL`. An error occurs if the argument is not a valid JSON document.

  An empty array, empty object, or scalar value has depth 1. A nonempty array containing only elements of depth 1 or nonempty object containing only member values of depth 1 has depth 2. Otherwise, a JSON document has depth greater than 2.

```
mysql> SELECT JSON_DEPTH('{}'), JSON_DEPTH('[]'), JSON_DEPTH('true');
+------------------+------------------+--------------------+
| JSON_DEPTH('{}') | JSON_DEPTH('[]') | JSON_DEPTH('true') |
+------------------+------------------+--------------------+
|                1 |                1 |                  1 |
+------------------+------------------+--------------------+
mysql> SELECT JSON_DEPTH('[10, 20]'), JSON_DEPTH('[[], {}]');
+------------------------+------------------------+
| JSON_DEPTH('[10, 20]') | JSON_DEPTH('[[], {}]') |
+------------------------+------------------------+
|                      2 |                      2 |
+------------------------+------------------------+
mysql> SELECT JSON_DEPTH('[10, {"a": 20}]');
+-------------------------------+
| JSON_DEPTH('[10, {"a": 20}]') |
+-------------------------------+
|                             3 |
+-------------------------------+
```

- `JSON_LENGTH(`*`json_doc`*`[, `*`path`*`])`

  Returns the length of a JSON document, or, if a *`path`* argument is given, the length of the value within the document identified by the path. Returns `NULL` if any argument is `NULL` or the *`path`* argument does not identify a value in the document. An error occurs if the *`json_doc`* argument is not a valid JSON document or the *`path`* argument is not a valid path expression or contains a `*` or `**` wildcard.

  The length of a document is determined as follows:

  - The length of a scalar is 1.

- The length of an array is the number of array elements.
- The length of an object is the number of object members.
- The length does not count the length of nested arrays or objects.

```
mysql> SELECT JSON_LENGTH('[1, 2, {"a": 3}]');
+--------------------------------+
| JSON_LENGTH('[1, 2, {"a": 3}]') |
+--------------------------------+
|                              3 |
+--------------------------------+
mysql> SELECT JSON_LENGTH('{"a": 1, "b": {"c": 30}}');
+---------------------------------------+
| JSON_LENGTH('{"a": 1, "b": {"c": 30}}') |
+---------------------------------------+
|                                     2 |
+---------------------------------------+
mysql> SELECT JSON_LENGTH('{"a": 1, "b": {"c": 30}}', '$.b');
+------------------------------------------------+
| JSON_LENGTH('{"a": 1, "b": {"c": 30}}', '$.b') |
+------------------------------------------------+
|                                              1 |
+------------------------------------------------+
```

- `JSON_TYPE(`*`json_val`*`)`

  Returns a `utf8mb4` string indicating the type of a JSON value. This can be an object, an array, or a scalar type, as shown here:

```
mysql> SET @j = '{"a": [10, true]}';
mysql> SELECT JSON_TYPE(@j);
+---------------+
| JSON_TYPE(@j) |
+---------------+
| OBJECT        |
+---------------+
mysql> SELECT JSON_TYPE(JSON_EXTRACT(@j, '$.a'));
+-----------------------------------+
| JSON_TYPE(JSON_EXTRACT(@j, '$.a')) |
+-----------------------------------+
| ARRAY                             |
+-----------------------------------+
mysql> SELECT JSON_TYPE(JSON_EXTRACT(@j, '$.a[0]'));
+--------------------------------------+
| JSON_TYPE(JSON_EXTRACT(@j, '$.a[0]')) |
+--------------------------------------+
| INTEGER                              |
+--------------------------------------+
mysql> SELECT JSON_TYPE(JSON_EXTRACT(@j, '$.a[1]'));
+--------------------------------------+
| JSON_TYPE(JSON_EXTRACT(@j, '$.a[1]')) |
+--------------------------------------+
| BOOLEAN                              |
+--------------------------------------+
```

  `JSON_TYPE()` returns `NULL` if the argument is `NULL`:

```
mysql> SELECT JSON_TYPE(NULL);
+-----------------+
| JSON_TYPE(NULL) |
```

```
+-----------------+
| NULL            |
+-----------------+
```

An error occurs if the argument is not a valid JSON value:

```
mysql> SELECT JSON_TYPE(1);
ERROR 3146 (22032): Invalid data type for JSON data in argument 1
to function json_type; a JSON string or JSON type is required.
```

For a non-`NULL`, non-error result, the following list describes the possible `JSON_TYPE()` return values:

- Purely JSON types:
  - `OBJECT`: JSON objects
  - `ARRAY`: JSON arrays
  - `BOOLEAN`: The JSON true and false literals
  - `NULL`: The JSON null literal
- Numeric types:
  - `INTEGER`: MySQL `TINYINT`, `SMALLINT`, `MEDIUMINT` and `INT` and `BIGINT` scalars
  - `DOUBLE`: MySQL `DOUBLE` `FLOAT` scalars
  - `DECIMAL`: MySQL `DECIMAL` and `NUMERIC` scalars
- Temporal types:
  - `DATETIME`: MySQL `DATETIME` and `TIMESTAMP` scalars
  - `DATE`: MySQL `DATE` scalars
  - `TIME`: MySQL `TIME` scalars
- String types:
  - `STRING`: MySQL `utf8` character type scalars: `CHAR`, `VARCHAR`, `TEXT`, `ENUM`, and `SET`
- Binary types:
  - `BLOB`: MySQL binary type scalars including `BINARY`, `VARBINARY`, `BLOB`, and `BIT`
- All other types:
  - `OPAQUE` (raw bits)

- `JSON_VALID(`*`val`*`)`

  Returns 0 or 1 to indicate whether a value is valid JSON. Returns `NULL` if the argument is `NULL`.

```
mysql> SELECT JSON_VALID('{"a": 1}');
+------------------------+
| JSON_VALID('{"a": 1}') |
+------------------------+
```

```
|                     1 |
+-----------------------+
mysql> SELECT JSON_VALID('hello'), JSON_VALID('"hello"');
+--------------------+----------------------+
| JSON_VALID('hello') | JSON_VALID('"hello"') |
+--------------------+----------------------+
|                  0 |                    1 |
+--------------------+----------------------+
```

12.17.6 JSON Table Functions

This section contains information about JSON functions that convert JSON data to tabular data. In MySQL 8.0.4 and later, one such function—`JSON_TABLE()`—is supported.

- `JSON_TABLE(expr, path COLUMNS (column_list) [AS] alias)`

  Extracts data from a JSON document and returns it as a relational table having the specified columns. The complete syntax for this function is shown here:

```
JSON_TABLE(
    expr,
    path COLUMNS (column_list)
)   [AS] alias

column_list:
    column[, column][, ...]

column:
    name FOR ORDINALITY
    |  name type PATH string path [on_error] [on_empty]
    |  name type EXISTS PATH string path
    |  NESTED [PATH] path COLUMNS (column_list)

on_error:
    {NULL | ERROR | DEFAULT json_string} ON ERROR

on_empty:
    {NULL | ERROR | DEFAULT json_string} ON EMPTY
```

  `expr`: This is an expression that returns JSON data. This can be a constant (`'{"a":1}'`), a column (`t1.json_data`, given table `t1` specified prior to `JSON_TABLE()` in the `FROM` clause), or a function call (`JSON_EXTRACT(t1,jsn_data,'$.post.comments')`).

  `path`: A JSON path expression, which is applied to the data source. We refer to the JSON value matching the path as the *row source*; this is used to generate a row of relational data. The `COLUMNS` clause evaluates the row source, finds specific JSON values within the row source, and returns those JSON values as SQL values in individual columns of a row of relational data.

  The *`alias`* is required. The usual rules for table aliases apply (see Section 9.2, “Schema Object Names”).

  `JSON_TABLE()` supports four types of columns, described in the following list:

  1. *`name`* `FOR ORDINALITY`: This type enumerates rows in the `COLUMNS` clause; the column named *`name`* is a counter whose type is `UNSIGNED INT`, and whose initial value is 1. This is equivalent to specifying a column as `AUTO_INCREMENT` in a `CREATE TABLE` statement, and can be used to distinguish parent rows with the same value for multiple rows generated by a `NESTED [PATH]` clause.

2. *`name type PATH string_path [on_error] [on_empty]`*: Columns of this type are used to extract values specified by *`string_path`*. *`type`* is a MySQL data type. `JSON_TABLE()` extracts data as JSON then coerces it to the column type, using the regular automatic type conversion applying to JSON data in MySQL. The exact behavior depends on the column type: If the column type is an SQL type, then only a scalar value can be saved in the column. Saving an object or array triggers the *`on error`* clause; this also occurs when an error takes place during coercion from the value saved as JSON to the table column, such as trying to save the string `'asd'` to an integer column. A missing value triggers the *`on_empty`* clause.

   The optional *`on_error`* clause determines what `JSON_TABLE()` does when saving an object or array:

   - `NULL ON ERROR`: The column is set to `NULL`; this is the default behavior. If an error occurs during type coercion, a warning is thrown.
   - `ERROR ON ERROR`: An error is thrown.
   - `DEFAULT` *`json string`* `ON ERROR`: The *`json_string`* is parsed as JSON (provided that it is valid) and stored instead of the object or array. A warning is thrown if the error is caused by type coercion. Column type rules also apply to the default value.

   When a value saved to a column is truncated, such as saving 3.14159 in a `DECIMAL(10,1)` column, a warning is issued independently of any `ON ERROR` option. When multiple values are truncated in a single statement, the warning is issued only once.

   The optional *`on empty`* clause determines what `JSON_TABLE()` does in the event that data is missing (depending on type). This clause is also triggered on a column in a `NESTED PATH` clause when the latter has no match and a `NULL` complemented row is produced for it. *`on empty`* takes one of the following values:

   - `NULL ON EMPTY`: The column is set to `NULL`; this is the default behavior.
   - `ERROR ON EMPTY`: An error is thrown.
   - `DEFAULT` *`json_string`* `ON EMPTY`: the provided *`json_string`* is parsed as JSON, as long as it is valid, and stored instead of the missing value. Column type rules also apply to the default value.

   This query demonstrates the use of the `ON ERROR` and `ON EMPTY` options. The row corresponding to `{"b":1}` is empty for the path `"$.a"`, and attempting to save `[1,2]` as a scalar produces an error; these rows are highlighted in the output shown.

```
mysql> SELECT *
    -> FROM
    ->   JSON_TABLE(
    ->     '[{"a":"3"},{"a":2},{"b":1},{"a":0},{"a":[1,2]}]',
    ->     "$[*]"
    ->     COLUMNS(
    ->       rowid FOR ORDINALITY,
    ->       ac VARCHAR(100) PATH "$.a" DEFAULT '999' ON ERROR DEFAULT '111' ON EMPTY,
    ->       aj JSON PATH "$.a" DEFAULT '{"x": 333}' ON EMPTY,
    ->       bx INT EXISTS PATH "$.b"
    ->     )
    ->   ) AS tt;

+-------+------+------------+------+
| rowid | ac   | aj         | bx   |
```

```
+-------+------+------------+------+
|     1 | 3    | "3"        |    0 |
|     2 | 2    | 2          |    0 |
|     3 | 111  | {"x": 333} |    1 |
|     4 | 0    | 0          |    0 |
|     5 | 999  | [1, 2]     |    0 |
+-------+------+------------+------+
5 rows in set (0.00 sec)
```

3. *`name type`* `EXISTS PATH` *`path`*: This column returns 1 if any data is present at the location specified by *`path`*, and 0 otherwise. *`type`* can be any valid MySQL data type, but should normally be specified as some variety of `INT`.

4. `NESTED [PATH]` *`path`* `COLUMNS (`*`column_list`*`)`: This flattens nested objects or arrays in JSON data into a single row along with the JSON values from the parent object or array. Using multiple `PATH` options allows projection of JSON values from multiple levels of nesting into a single row.

   The *`path`* is relative to the parent path row path of `JSON_TABLE()`, or the path of the parent `NESTED [PATH]` clause in the event of nested paths.

Column names are subject to the usual rules and limitations governing table column names. See Section 9.2, “Schema Object Names”.

All JSON and JSON path expressions are checked for validity; an invalid expression of either type causes an error.

Each match for the *`path`* preceding the `COLUMNS` keyword maps to an individual row in the result table. For example, the following query gives the result shown here:

```
mysql> SELECT *
    -> FROM
    ->   JSON_TABLE(
    ->     '[{"x":2,"y":"8"},{"x":"3","y":"7"},{"x":"4","y":6}]',
    ->     "$[*]" COLUMNS(
    ->       xval VARCHAR(100) PATH "$.x",
    ->       yval VARCHAR(100) PATH "$.y"
    ->     )
    ->   ) AS  jt1;

+------+------+
| xval | yval |
+------+------+
| 2    | 8    |
| 3    | 7    |
| 4    | 6    |
+------+------+
```

The expression `"$[*]"` matches each element of the array. You can filter the rows in the result by modifying the path. For example, using `"$[1]"` limits extraction to the second element of the JSON array used as the source, as shown here:

```
mysql> SELECT *
    -> FROM
    ->   JSON_TABLE(
    ->     '[{"x":2,"y":"8"},{"x":"3","y":"7"},{"x":"4","y":6}]',
    ->     "$[1]" COLUMNS(
    ->       xval VARCHAR(100) PATH "$.x",
    ->       yval VARCHAR(100) PATH "$.y"
    ->     )
```

```
    ->   ) AS  jt1;

+------+------+
| xval | yval |
+------+------+
| 3    | 7    |
+------+------+
```

Within a column definition, `"$"` passes the entire match to the column; `"$.x"` and `"$.y"` pass only the values corresponding to the keys `x` and `y`, respectively, within that match. For more information, see JSON Path Syntax.

`NESTED PATH` (or simply `NESTED`; `PATH` is optional) produces a set of records for each match in the `COLUMNS` clause to which it belongs. If there is no match, all columns of the nested path are set to `NULL`. This implements an outer join between the topmost clause and `NESTED [PATH]`. An inner join can be emulated by applying a suitable condition in the `WHERE` clause, as shown here:

```
mysql> SELECT *
    -> FROM
    ->   JSON_TABLE(
    ->     '[ {"a": 1, "b": [11,111]}, {"a": 2, "b": [22,222]}, {"a":3}]',
    ->     '$[*]' COLUMNS(
    ->             a INT PATH '$.a',
    ->             NESTED PATH '$.b[*]' COLUMNS (b INT PATH '$')
    ->            )
    ->    ) AS jt
    -> WHERE b IS NOT NULL;

+------+------+
| a    | b    |
+------+------+
|    1 |   11 |
|    1 |  111 |
|    2 |   22 |
|    2 |  222 |
+------+------+
```

Sibling nested paths—that is, two or more instances of `NESTED [PATH]` in the same `COLUMNS` clause—are processed one after another, one at a time. While one nested path is producing records, columns of any sibling nested path expressions are set to `NULL`. This means that the total number of records for a single match within a single containing `COLUMNS` clause is the sum and not the product of all records produced by `NESTED [PATH]` modifiers, as shown here:

```
mysql> SELECT *
    -> FROM
    ->   JSON_TABLE(
    ->     '[{"a": 1, "b": [11,111]}, {"a": 2, "b": [22,222]}]',
    ->     '$[*]' COLUMNS(
    ->         a INT PATH '$.a',
    ->         NESTED PATH '$.b[*]' COLUMNS (b1 INT PATH '$'),
    ->         NESTED PATH '$.b[*]' COLUMNS (b2 INT PATH '$')
    ->     )
    -> ) AS jt;

+------+------+------+
| a    | b1   | b2   |
+------+------+------+
|    1 |   11 | NULL |
|    1 |  111 | NULL |
|    1 | NULL |   11 |
|    1 | NULL |  111 |
|    2 |   22 | NULL |
```

```
|    2 |  222 | NULL |
|    2 | NULL |   22 |
|    2 | NULL |  222 |
+------+------+------+
```

A `FOR ORDINALITY` column enumerates records produced by the `COLUMNS` clause, and can be used to distinguish parent records of a nested path, especially if values in parent records are the same, as can be seen here:

```
mysql> SELECT *
    -> FROM
    ->   JSON_TABLE(
    ->     '[{"a": "a_val",
    '>       "b": [{"c": "c_val", "l": [1,2]}]},
    '>     {"a": "a_val",
    '>       "b": [{"c": "c_val","l": [11]}, {"c": "c_val", "l": [22]}]}]',
    ->     '$[*]' COLUMNS(
    ->       top_ord FOR ORDINALITY,
    ->       apath VARCHAR(10) PATH '$.a',
    ->       NESTED PATH '$.b[*]' COLUMNS (
    ->         bpath VARCHAR(10) PATH '$.c',
    ->         ord FOR ORDINALITY,
    ->         NESTED PATH '$.l[*]' COLUMNS (lpath varchar(10) PATH '$')
    ->         )
    ->     )
    -> ) as jt;

+---------+---------+---------+------+-------+
| top_ord | apath   | bpath   | ord  | lpath |
+---------+---------+---------+------+-------+
|       1 |  a_val  |  c_val  |    1 | 1     |
|       1 |  a_val  |  c_val  |    1 | 2     |
|       2 |  a_val  |  c_val  |    1 | 11    |
|       2 |  a_val  |  c_val  |    2 | 22    |
+---------+---------+---------+------+-------+
```

The source document contains an array of two elements; each of these elements produces two rows. The values of `apath` and `bpath` are the same over the entire result set; this means that they cannot be used to determine whether `lpath` values came from the same or different parents. The value of the `ord` column remains the same as the set of records having `top_ord` equal to 1, so these two values are from a single object. The remaining two values are from different objects, since they have different values in the `ord` column.

12.17.7 JSON Schema Validation Functions

Beginning with MySQL 8.0.17, MySQL supports validation of JSON documents against JSON schemas conforming to Draft 4 of the JSON Schema specification. This can be done using either of the functions detailed in this section, both of which take two orguments, a JSON schema, and a JSON document which is validated against the schema. `JSON_SCHEMA_VALID()` returns true if the document is validates against the schema, and false if it is not; `JSON_SCHEMA_VALIDATION_REPORT()` provides a report in JSON format on the validation.

Both functions handle null or invalid input as follows:

- If at least one of the arguments is `NULL`, the function returns `NULL`.
- If at least one of the arguments is not valid JSON, the function raises an error (`ER_INVALID_TYPE_FOR_JSON`)
- In addition, if the schema is not a valid JSON object, the function returns `ER_INVALID_JSON_TYPE`.

MySQL supports the `required` attribute in JSON schemas to enforce the inclusion of required properties (see the examples in the function descriptions).

MySQL does not support external resources in JSON schemas; using the `$ref` keyword causes `JSON_SCHEMA_VALID()` to fail with `ER_NOT_SUPPORTED_YET`.

Note

MySQL supports regular expression patterns in JSON schema, which supports but silently ignores invalid patterns (see the description of `JSON_SCHEMA_VALID()` for an example).

These functions are described in detail in the following list:

- `JSON_SCHEMA_VALID(`*`schema`*`,`*`document`*`)`

  Validates a JSON *`document`* against a JSON *`schema`*. Both *`schema`* and *`document`* are required. The schema must be a valid JSON object; the document must be a valid JSON document. Provided that these conditions are met: If the document validates against the schema, the function returns true (1); otherwise, it returns false (0).

  In this example, we set a user variable `@schema` to the value of a a JSON schema for geographical coordinates, and another one `@document` to the value of a JSON document containing one such coordinate. We then verify that `@document` validates according to `@schema` by using them as the arguments to `JSON_SCHEMA_VALID()`:

```
mysql> SET @schema = '{
    '>  "id": "http://json-schema.org/geo",
    '> "$schema": "http://json-schema.org/draft-04/schema#",
    '> "description": "A geographical coordinate",
    '> "type": "object",
    '> "properties": {
    '>   "latitude": {
    '>     "type": "number",
    '>     "minimum": -90,
    '>     "maximum": 90
    '>   },
    '>   "longitude": {
    '>     "type": "number",
    '>     "minimum": -180,
    '>     "maximum": 180
    '>   }
    '> },
    '> "required": ["latitude", "longitude"]
    '>}';
Query OK, 0 rows affected (0.01 sec)


mysql> SET @document = '{
    '> "latitude": 63.444697,
    '> "longitude": 10.445118
    '>}';
Query OK, 0 rows affected (0.00 sec)


mysql> SELECT JSON_SCHEMA_VALID(@schema, @document);
+---------------------------------------+
| JSON_SCHEMA_VALID(@schema, @document) |
+---------------------------------------+
|                                     1 |
+---------------------------------------+
1 row in set (0.00 sec)
```

Since `@schema` contains the `required` attribute, we can set `@document` to a value that is otherwise valid but does not contain the required properties, then test it against `@schema`, like this:

```
mysql> SET @document = '{}';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT JSON_SCHEMA_VALID(@schema, @document);
+--------------------------------------+
| JSON_SCHEMA_VALID(@schema, @document) |
+--------------------------------------+
|                                    0 |
+--------------------------------------+
1 row in set (0.00 sec)
```

If we now set the value of `@schema` to the same JSON schema but without the `required` attribute, `@document` validates because it is a valid JSON object, even though it contains no properties, as shown here:

```
mysql> SET @schema = '{
    '> "id": "http://json-schema.org/geo",
    '> "$schema": "http://json-schema.org/draft-04/schema#",
    '> "description": "A geographical coordinate",
    '> "type": "object",
    '> "properties": {
    '>   "latitude": {
    '>     "type": "number",
    '>     "minimum": -90,
    '>     "maximum": 90
    '>   },
    '>   "longitude": {
    '>     "type": "number",
    '>     "minimum": -180,
    '>     "maximum": 180
    '>   }
    '> }
    '>}';
Query OK, 0 rows affected (0.00 sec)


mysql> SELECT JSON_SCHEMA_VALID(@schema, @document);
+--------------------------------------+
| JSON_SCHEMA_VALID(@schema, @document) |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
1 row in set (0.00 sec)
```

JSON Schema has support for specifying regular expression patterns for strings, but the implementation used by MySQL silently ignores invalid patterns. This means that `JSON_SCHEMA_VALID()` can return true even when a regular expression pattern is invalid, as shown here:

```
mysql> SELECT JSON_SCHEMA_VALID('{"type":"string","pattern":"("}', '"abc"');
+-------------------------------------------------------------+
| JSON_SCHEMA_VALID('{"type":"string","pattern":"("}', '"abc"') |
+-------------------------------------------------------------+
|                                                           1 |
+-------------------------------------------------------------+
1 row in set (0.04 sec)
```

- `JSON_SCHEMA_VALIDATION_REPORT(`*`schema`*`,`*`document`*`)`

Validates a JSON *`document`* against a JSON *`schema`*. Both *`schema`* and *`document`* are required. As with JSON_VALID_SCHEMA(), the schema must be a valid JSON object, and the document must be a valid JSON document. Provided that these conditions are met, the function returns a report, as a JSON document, on the outcome of the validation. If the JSON document is considered valid according to the JSON Schema, the function returns a JSON object with one property `valid` having the value "true". If the JSON document fails validation, the function returns a JSON object which includes the properties listed here:

- `valid`: Always "false" for a failed schema validation
- `reason`: A human-readable string containing the reason for the failure
- `schema-location`: A JSON pointer URI fragment identifier indicating where in the JSON schema the validation failed (see Note following this list)
- `document-location`: A JSON pointer URI fragment identifier indicating where in the JSON document the validation failed (see Note following this list)
- `schema-failed-keyword`: A string containing the name of the keyword or property in the JSON schema that was violated

Note

JSON pointer URI fragment identifiers are defined in RFC 6901 - JavaScript Object Notation (JSON) Pointer. (These are *not* the same as the JSON path notation used by `JSON_EXTRACT()` and other MySQL JSON functions.) In this notation, `#` represents the entire document, and `#/myprop` represents the portion of the document included in the top-level property named `myprop`. See the specification just cited and the examples shown later in this section for more information.

In this example, we set a user variable `@schema` to the value of a a JSON schema for geographical coordinates, and another one `@document` to the value of a JSON document containing one such coordinate. We then verify that `@document` validates according to `@schema` by using them as the arguments to `JSON_SCHEMA_VALIDATION_REORT()`:

```
mysql> SET @schema = '{
    '>  "id": "http://json-schema.org/geo",
    '> "$schema": "http://json-schema.org/draft-04/schema#",
    '> "description": "A geographical coordinate",
    '> "type": "object",
    '> "properties": {
    '>   "latitude": {
    '>     "type": "number",
    '>     "minimum": -90,
    '>     "maximum": 90
    '>   },
    '>   "longitude": {
    '>     "type": "number",
    '>     "minimum": -180,
    '>     "maximum": 180
    '>   }
    '> },
    '> "required": ["latitude", "longitude"]
    '>}';
Query OK, 0 rows affected (0.01 sec)

mysql> SET @document = '{
```

```
    '> "latitude": 63.444697,
    '> "longitude": 10.445118
    '>}';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT JSON_SCHEMA_VALIDATION_REPORT(@schema, @document);
+-------------------------------------------------+
| JSON_SCHEMA_VALIDATION_REPORT(@schema, @document) |
+-------------------------------------------------+
| {"valid": true}                                 |
+-------------------------------------------------+
1 row in set (0.00 sec)
```

Now we set `@document` such that it specifies an illegal value for one of its properties, like this:

```
mysql> SET @document = '{
    '> "latitude": 63.444697,
    '> "longitude": 310.445118
    '> }';
```

Validation of `@document` now fails when tested with `JSON_SCHEMA_VALIDATION_REPORT()`. The output from the function call contains detailed information about the failure (with the function wrapped by `JSON_PRETTY()` to provide better formatting), as shown here:

```
mysql> SELECT JSON_PRETTY(JSON_SCHEMA_VALIDATION_REPORT(@schema, @document))\G
*************************** 1. row ***************************
JSON_PRETTY(JSON_SCHEMA_VALIDATION_REPORT(@schema, @document)): {
  "valid": false,
  "reason": "The JSON document location '#/longitude' failed requirement 'maximum' at JSON Schema location '
  "schema-location": "#/properties/longitude",
  "document-location": "#/longitude",
  "schema-failed-keyword": "maximum"
}
1 row in set (0.00 sec)
```

Since `@schema` contains the `required` attribute, we can set `@document` to a value that is otherwise valid but does not contain the required properties, then test it against `@schema`. The output of `JSON_SCHEMA_VALIDATION_REPORT()` shows that validation fails due to lack of a required element, like this:

```
mysql> SET @document = '{}';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT JSON_PRETTY(JSON_SCHEMA_VALIDATION_REPORT(@schema, @document))\G
*************************** 1. row ***************************
JSON_PRETTY(JSON_SCHEMA_VALIDATION_REPORT(@schema, @document)): {
  "valid": false,
  "reason": "The JSON document location '#' failed requirement 'required' at JSON Schema location '#'",
  "schema-location": "#",
  "document-location": "#",
  "schema-failed-keyword": "required"
}
1 row in set (0.00 sec)
```

If we now set the value of `@schema` to the same JSON schema but without the `required` attribute, `@document` validates because it is a valid JSON object, even though it contains no properties, as shown here:

```
mysql> SET @schema = '{
    '> "id": "http://json-schema.org/geo",
```

```
    '> "$schema": "http://json-schema.org/draft-04/schema#",
    '> "description": "A geographical coordinate",
    '> "type": "object",
    '> "properties": {
    '>   "latitude": {
    '>     "type": "number",
    '>     "minimum": -90,
    '>     "maximum": 90
    '>   },
    '>   "longitude": {
    '>     "type": "number",
    '>     "minimum": -180,
    '>     "maximum": 180
    '>   }
    '> }
    '>}';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT JSON_SCHEMA_VALIDATION_REPORT(@schema, @document);
+--------------------------------------------------+
| JSON_SCHEMA_VALIDATION_REPORT(@schema, @document) |
+--------------------------------------------------+
| {"valid": true}                                  |
+--------------------------------------------------+
1 row in set (0.00 sec)
```

12.17.8 JSON Utility Functions

This section documents utility functions that act on JSON values, or strings that can be parsed as JSON values. `JSON_PRETTY()` prints out a JSON value in a format that is easy to read. `JSON_STORAGE_SIZE()` and `JSON_STORAGE_FREE()` show, respectively, the amount of storage space used by a given JSON value and the amount of space remaining in a `JSON` column following a partial update.

- `JSON_PRETTY(`*`json_val`*`)`

  Provides pretty-printing of JSON values similar to that implemented in PHP and by other languages and database systems. The value supplied must be a JSON value or a valid string representation of a JSON value. Extraneous whitespaces and newlines present in this value have no effect on the output. For a `NULL` value, the function returns `NULL`. If the value is not a JSON document, or if it cannot be parsed as one, the function fails with an error.

  Formatting of the output from this function adheres to the following rules:

  - Each array element or object member appears on a separate line, indented by one additional level as compared to its parent.
  - Each level of indentation adds two leading spaces.
  - A comma separating individual array elements or object members is printed before the newline that separates the two elements or members.
  - The key and the value of an object member are separated by a colon followed by a space (`': '`).
  - An empty object or array is printed on a single line. No space is printed between the opening and closing brace.
  - Special characters in string scalars and key names are escaped employing the same rules used by the `JSON_QUOTE()` function.

```
mysql> SELECT JSON_PRETTY('123'); # scalar
+--------------------+
| JSON_PRETTY('123') |
+--------------------+
| 123                |
+--------------------+

mysql> SELECT JSON_PRETTY("[1,3,5]"); # array
+------------------------+
| JSON_PRETTY("[1,3,5]") |
+------------------------+
| [
  1,
  3,
  5
]      |
+------------------------+

mysql> SELECT JSON_PRETTY('{"a":"10","b":"15","x":"25"}'); # object
+-------------------------------------------+
| JSON_PRETTY('{"a":"10","b":"15","x":"25"}') |
+-------------------------------------------+
| {
  "a": "10",
  "b": "15",
  "x": "25"
}   |
+-------------------------------------------+

mysql> SELECT JSON_PRETTY('["a",1,{"key1":
    '>    "value1"},"5",     "77" ,
    '>       {"key2":["value3","valueX",
    '> "valueY"]},"j", "2"   ]')\G  # nested arrays and objects
*************************** 1. row ***************************
JSON_PRETTY('["a",1,{"key1":
             "value1"},"5",     "77" ,
                {"key2":["value3","valuex",
          "valuey"]},"j", "2"   ]'): [
  "a",
  1,
  {
    "key1": "value1"
  },
  "5",
  "77",
  {
    "key2": [
      "value3",
      "valuex",
      "valuey"
    ]
  },
  "j",
  "2"
]
```

- `JSON_STORAGE_FREE(json_val)`

  For a `JSON` column value, this function shows how much storage space was freed in its binary representation after it was updated in place using `JSON_SET()`, `JSON_REPLACE()`, or `JSON_REMOVE()`. The argument can also be a valid JSON document or a string which can be parsed as one—either as a literal value or as the value of a user variable—in which case the function returns 0. It returns a positive, nonzero value if the argument is a `JSON` column value which has been updated as described previously, such that its binary representation takes up less space than it did prior to the update. For a `JSON` column which has been updated such that its binary representation is the same as

or larger than before, or if the update was not able to take advantage of a partial update, it returns 0; it returns `NULL` if the argument is `NULL`.

If *`json_val`* is not `NULL`, and neither is a valid JSON document nor can be successfully parsed as one, an error results.

In this example, we create a table containing a `JSON` column, then insert a row containing a JSON object:

```
mysql> CREATE TABLE jtable (jcol JSON);
Query OK, 0 rows affected (0.38 sec)

mysql> INSERT INTO jtable VALUES
    ->     ('{"a": 10, "b": "wxyz", "c": "[true, false]"}');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT * FROM jtable;
+-------------------------------------------+
| jcol                                      |
+-------------------------------------------+
| {"a": 10, "b": "wxyz", "c": "[true, false]"} |
+-------------------------------------------+
1 row in set (0.00 sec)
```

Now we update the column value using `JSON_SET()` such that a partial update can be performed; in this case, we replace the value pointed to by the `c` key (the array `[true, false]`) with one that takes up less space (the integer `1`):

```
mysql> UPDATE jtable
    ->     SET jcol = JSON_SET(jcol, "$.a", 10, "$.b", "wxyz", "$.c", 1);
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM jtable;
+-------------------------------+
| jcol                          |
+-------------------------------+
| {"a": 10, "b": "wxyz", "c": 1} |
+-------------------------------+
1 row in set (0.00 sec)

mysql> SELECT JSON_STORAGE_FREE(jcol) FROM jtable;
+-------------------------+
| JSON_STORAGE_FREE(jcol) |
+-------------------------+
|                      14 |
+-------------------------+
1 row in set (0.00 sec)
```

The effects of successive partial updates on this free space are cumulative, as shown in this example using `JSON_SET()` to reduce the space taken up by the value having key `b` (and making no other changes):

```
mysql> UPDATE jtable
    ->     SET jcol = JSON_SET(jcol, "$.a", 10, "$.b", "wx", "$.c", 1);
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT JSON_STORAGE_FREE(jcol) FROM jtable;
+-------------------------+
| JSON_STORAGE_FREE(jcol) |
```

```
+-------------------------+
|                      16 |
+-------------------------+
1 row in set (0.00 sec)
```

Updating the column without using `JSON_SET()`, `JSON_REPLACE()`, or `JSON_REMOVE()` means that the optimizer cannot perform the update in place; in this case, `JSON_STORAGE_FREE()` returns 0, as shown here:

```
mysql> UPDATE jtable SET jcol = '{"a": 10, "b": 1}';
Query OK, 1 row affected (0.05 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT JSON_STORAGE_FREE(jcol) FROM jtable;
+-------------------------+
| JSON_STORAGE_FREE(jcol) |
+-------------------------+
|                       0 |
+-------------------------+
1 row in set (0.00 sec)
```

Partial updates of JSON documents can be performed only on column values. For a user variable that stores a JSON value, the value is always completely replaced, even when the update is performed using `JSON_SET()`:

```
mysql> SET @j = '{"a": 10, "b": "wxyz", "c": "[true, false]"}';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @j = JSON_SET(@j, '$.a', 10, '$.b', 'wxyz', '$.c', '1');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @j, JSON_STORAGE_FREE(@j) AS Free;
+---------------------------------+------+
| @j                              | Free |
+---------------------------------+------+
| {"a": 10, "b": "wxyz", "c": "1"} |    0 |
+---------------------------------+------+
1 row in set (0.00 sec)
```

For a JSON literal, this function always returns 0:

```
mysql> SELECT JSON_STORAGE_FREE('{"a": 10, "b": "wxyz", "c": "1"}') AS Free;
+------+
| Free |
+------+
|    0 |
+------+
1 row in set (0.00 sec)
```

- `JSON_STORAGE_SIZE(`*`json_val`*`)`

  This function returns the number of bytes used to store the binary representation of a JSON document. When the argument is a `JSON` column, this is the space used to store the JSON document as it was inserted into the column, prior to any partial updates that may have been performed on it afterwards. *`json_val`* must be a valid JSON document or a string which can be parsed as one. In the case where it is string, the function returns the amount of storage space in the JSON binary representation that is created by parsing the string as JSON and converting it to binary. It returns `NULL` if the argument is `NULL`.

An error results when *`json_val`* is not `NULL`, and is not—or cannot be successfully parsed as—a JSON document.

To illustrate this function's behavior when used with a `JSON` column as its argument, we create a table named `jtable` containing a `JSON` column `jcol`, insert a JSON value into the table, then obtain the storage space used by this column with `JSON_STORAGE_SIZE()`, as shown here:

```
mysql> CREATE TABLE jtable (jcol JSON);
Query OK, 0 rows affected (0.42 sec)

mysql> INSERT INTO jtable VALUES
    ->     ('{"a": 1000, "b": "wxyz", "c": "[1, 3, 5, 7]"}');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT
    ->     jcol,
    ->     JSON_STORAGE_SIZE(jcol) AS Size,
    ->     JSON_STORAGE_FREE(jcol) AS Free
    -> FROM jtable;
+-----------------------------------------------+------+------+
| jcol                                          | Size | Free |
+-----------------------------------------------+------+------+
| {"a": 1000, "b": "wxyz", "c": "[1, 3, 5, 7]"} |   47 |    0 |
+-----------------------------------------------+------+------+
1 row in set (0.00 sec)
```

According to the output of `JSON_STORAGE_SIZE()`, the JSON document inserted into the column takes up 47 bytes. We also checked the amount of space freed by any previous partial updates of the column using `JSON_STORAGE_FREE()`; since no updates have yet been performed, this is 0, as expected.

Next we perform an `UPDATE` on the table that should result in a partial update of the document stored in `jcol`, and then test the result as shown here:

```
mysql> UPDATE jtable SET jcol =
    ->     JSON_SET(jcol, "$.b", "a");
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT
    ->     jcol,
    ->     JSON_STORAGE_SIZE(jcol) AS Size,
    ->     JSON_STORAGE_FREE(jcol) AS Free
    -> FROM jtable;
+-----------------------------------------+------+------+
| jcol                                    | Size | Free |
+-----------------------------------------+------+------+
| {"a": 1000, "b": "a", "c": "[1, 3, 5, 7]"} |   47 |    3 |
+-----------------------------------------+------+------+
1 row in set (0.00 sec)
```

The value returned by `JSON_STORAGE_FREE()` in the previous query indicates that a partial update of the JSON document was performed, and that this freed 3 bytes of space used to store it. The result returned by `JSON_STORAGE_SIZE()` is unchanged by the partial update.

Partial updates are supported for updates using `JSON_SET()`, `JSON_REPLACE()`, or `JSON_REMOVE()`. The direct assignment of a value to a `JSON` column cannot be partially updated; following such an update, `JSON_STORAGE_SIZE()` always shows the storage used for the newly-set value:

```
mysql> UPDATE jtable
```

```
mysql>     SET jcol = '{"a": 4.55, "b": "wxyz", "c": "[true, false]"}';
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT
    ->     jcol,
    ->     JSON_STORAGE_SIZE(jcol) AS Size,
    ->     JSON_STORAGE_FREE(jcol) AS Free
    -> FROM jtable;
+-----------------------------------------------+------+------+
| jcol                                          | Size | Free |
+-----------------------------------------------+------+------+
| {"a": 4.55, "b": "wxyz", "c": "[true, false]"} |   56 |    0 |
+-----------------------------------------------+------+------+
1 row in set (0.00 sec)
```

A JSON user variable cannot be partially updated. This means that this function always shows the space currently used to store a JSON document in a user variable:

```
mysql> SET @j = '[100, "sakila", [1, 3, 5], 425.05]';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @j, JSON_STORAGE_SIZE(@j) AS Size;
+-----------------------------------+------+
| @j                                | Size |
+-----------------------------------+------+
| [100, "sakila", [1, 3, 5], 425.05] |   45 |
+-----------------------------------+------+
1 row in set (0.00 sec)

mysql> SET @j = JSON_SET(@j, '$[1]', "json");
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @j, JSON_STORAGE_SIZE(@j) AS Size;
+-----------------------------------+------+
| @j                                | Size |
+-----------------------------------+------+
| [100, "json", [1, 3, 5], 425.05] |   43 |
+-----------------------------------+------+
1 row in set (0.00 sec)

mysql> SET @j = JSON_SET(@j, '$[2][0]', JSON_ARRAY(10, 20, 30));
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @j, JSON_STORAGE_SIZE(@j) AS Size;
+-------------------------------------------+------+
| @j                                        | Size |
+-------------------------------------------+------+
| [100, "json", [[10, 20, 30], 3, 5], 425.05] |   56 |
+-------------------------------------------+------+
1 row in set (0.00 sec)
```

For a JSON literal, this function always returns the current storage space used:

```
mysql> SELECT
    ->     JSON_STORAGE_SIZE('[100, "sakila", [1, 3, 5], 425.05]') AS A,
    ->     JSON_STORAGE_SIZE('{"a": 1000, "b": "a", "c": "[1, 3, 5, 7]"}') AS B,
    ->     JSON_STORAGE_SIZE('{"a": 1000, "b": "wxyz", "c": "[1, 3, 5, 7]"}') AS C,
    ->     JSON_STORAGE_SIZE('[100, "json", [[10, 20, 30], 3, 5], 425.05]') AS D;
+----+----+----+----+
| A  | B  | C  | D  |
+----+----+----+----+
| 45 | 44 | 47 | 56 |
+----+----+----+----+
```

```
1 row in set (0.00 sec)
```

12.18 Functions Used with Global Transaction Identifiers (GTIDs)

The functions described in this section are used with GTID-based replication. It is important to keep in mind that all of these functions take string representations of GTID sets as arguments. As such, the GTID sets must always be quoted when used with them. See GTID Sets for more information.

The union of two GTID sets is simply their representations as strings, joined together with an interposed comma. In other words, you can define a very simple function for obtaining the union of two GTID sets, similar to that created here:

```
CREATE FUNCTION GTID_UNION(g1 TEXT, g2 TEXT)
    RETURNS TEXT DETERMINISTIC
    RETURN CONCAT(g1,',',g2);
```

For more information about GTIDs and how these GTID functions are used in practice, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

Table 12.23 GTID Functions

Name	Description
`GTID_SUBSET()`	Return true if all GTIDs in subset are also in set; otherwise false.
`GTID_SUBTRACT()`	Return all GTIDs in set that are not in subset.
`WAIT_FOR_EXECUTED_GTID_SET()`	Wait until the given GTIDs have executed on the slave.
`WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()` (deprecated 8.0.18)	Deprecated. Use `WAIT_FOR_EXECUTED_GTID_SET()`.

- `GTID_SUBSET(`*`set1`*`,`*`set2`*`)`

  Given two sets of global transaction identifiers *`set1`* and *`set2`*, returns true if all GTIDs in *`set1`* are also in *`set2`*. Returns false otherwise.

  The GTID sets used with this function are represented as strings, as shown in the following examples:

```
mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
*************************** 1. row ***************************
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 1
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23-25',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
*************************** 1. row ***************************
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23-25',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 1
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
*************************** 1. row ***************************
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 0
1 row in set (0.00 sec)
```

- `GTID_SUBTRACT(`*`set1`*`,`*`set2`*`)`

Given two sets of global transaction identifiers *`set1`* and *`set2`*, returns only those GTIDs from *`set1`* that are not in *`set2`*.

All GTID sets used with this function are represented as strings and must be quoted, as shown in these examples:

```
mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21')\G
*************************** 1. row ***************************
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:22-57
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25')\G
*************************** 1. row ***************************
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:26-57
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:23-24')\G
*************************** 1. row ***************************
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:23-24'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:21-22:25-57
1 row in set (0.01 sec)
```

- `WAIT_FOR_EXECUTED_GTID_SET(`*`gtid_set`*`[, `*`timeout`*`])`

  Wait until the server has applied all of the transactions whose global transaction identifiers are contained in *`gtid_set`*; that is, until the condition GTID_SUBSET(*`gtid_subset`*, `@@GLOBAL.gtid_executed`) holds. See Section 17.1.3.1, “GTID Format and Storage” for a definition of GTID sets.

  If a timeout is specified, and *`timeout`* seconds elapse before all of the transactions in the GTID set have been applied, the function stops waiting. *`timeout`* is optional, and the default timeout is 0 seconds, in which case the function always waits until all of the transactions in the GTID set have been applied.

  `WAIT_FOR_EXECUTED_GTID_SET()` monitors all the GTIDs that are applied on the server, including transactions that arrive from all replication channels and user clients. It does not take into account whether replication channels have been started or stopped.

  For more information, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

  GTID sets used with this function are represented as strings and so must be quoted as shown in the following example:

  ```
  mysql> SELECT WAIT_FOR_EXECUTED_GTID_SET('3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5');
         -> 0
  ```

  For a syntax description for GTID sets, see Section 17.1.3.1, “GTID Format and Storage”.

  For `WAIT_FOR_EXECUTED_GTID_SET()`, the return value is the state of the query, where 0 represents success, and 1 represents timeout. Any other failures generate an error.

  `gtid_mode` cannot be changed to OFF while any client is using this function to wait for GTIDs to be applied.

- `WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS(`*`gtid_set`*`[, `*`timeout`*`][,`*`channel`*`])`

`WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()` is deprecated. Use `WAIT_FOR_EXECUTED_GTID_SET()` instead, which works regardless of the replication channel or user client through which the specified transactions arrive on the server.

12.19 MySQL Enterprise Encryption Functions

Note

MySQL Enterprise Encryption is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, https://www.mysql.com/products/.

MySQL Enterprise Edition includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. These functions enable Enterprise applications to perform the following operations:

- Implement added data protection using public-key asymmetric cryptography
- Create public and private keys and digital signatures
- Perform asymmetric encryption and decryption
- Use cryptographic hashing for digital signing and data verification and validation

MySQL Enterprise Encryption supports the RSA, DSA, and DH cryptographic algorithms.

MySQL Enterprise Encryption is supplied as a user-defined function (UDF) library, from which individual functions can be installed individually.

12.19.1 MySQL Enterprise Encryption Installation

MySQL Enterprise Encryption functions are located in a user-defined function (UDF) library file installed in the plugin directory (the directory named by the `plugin_dir` system variable). The UDF library base name is `openssl_udf` and the suffix is platform dependent. For example, the file name on Linux or Windows is `openssl_udf.so` or `openssl_udf.dll`, respectively.

To install functions from the library file, use the `CREATE FUNCTION` statement. To load all functions from the library, use this set of statements (adjust the file name suffix as necessary):

```
CREATE FUNCTION asymmetric_decrypt RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_derive RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_encrypt RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_sign RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_verify RETURNS INTEGER
  SONAME 'openssl_udf.so';
CREATE FUNCTION create_asymmetric_priv_key RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION create_asymmetric_pub_key RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION create_dh_parameters RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION create_digest RETURNS STRING
```

```
  SONAME 'openssl_udf.so';
```

Once installed, UDFs remain installed across server restarts. To unload UDFs, use the `DROP FUNCTION` statement. For example, to unload the key-generation functions, do this:

```
DROP FUNCTION create_asymmetric_priv_key;
DROP FUNCTION create_asymmetric_pub_key;
```

In the `CREATE FUNCTION` and `DROP FUNCTION` statements, the function names must be specified in lowercase. This differs from their use at function invocation time, for which you can use any lettercase.

The `CREATE FUNCTION` and `DROP FUNCTION` statements require the `INSERT` and `DROP` privilege, respectively, for the `mysql` database.

12.19.2 MySQL Enterprise Encryption Usage and Examples

To use MySQL Enterprise Encryption in applications, invoke the functions that are appropriate for the operations you wish to perform. This section demonstrates how to carry out some representative tasks:

- Create a private/public key pair using RSA encryption
- Use the private key to encrypt data and the public key to decrypt it
- Generate a digest from a string
- Use the digest with a key pair
- Create a symmetric key
- Limit CPU usage by key-generation operations

Create a private/public key pair using RSA encryption

```
-- Encryption algorithm; can be 'DSA' or 'DH' instead
SET @algo = 'RSA';
-- Key length in bits; make larger for stronger keys
SET @key_len = 1024;

-- Create private key
SET @priv = CREATE_ASYMMETRIC_PRIV_KEY(@algo, @key_len);
-- Derive corresponding public key from private key, using same algorithm
SET @pub = CREATE_ASYMMETRIC_PUB_KEY(@algo, @priv);
```

Now you can use the key pair to encrypt and decrypt data, sign and verify data, or generate symmetric keys.

Use the private key to encrypt data and the public key to decrypt it

This requires that the members of the key pair be RSA keys.

```
SET @ciphertext = ASYMMETRIC_ENCRYPT(@algo, 'My secret text', @priv);
SET @plaintext = ASYMMETRIC_DECRYPT(@algo, @ciphertext, @pub);
```

Conversely, you can encrypt using the public key and decrypt using the private key.

```
```

```
SET @ciphertext = ASYMMETRIC_ENCRYPT(@algo, 'My secret text', @pub);
SET @plaintext = ASYMMETRIC_DECRYPT(@algo, @ciphertext, @priv);
```

In either case, the algorithm specified for the encryption and decryption functions must match that used to generate the keys.

Generate a digest from a string

```
-- Digest type; can be 'SHA256', 'SHA384', or 'SHA512' instead
SET @dig_type = 'SHA224';

-- Generate digest string
SET @dig = CREATE_DIGEST(@dig_type, 'My text to digest');
```

Use the digest with a key pair

The key pair can be used to sign data, then verify that the signature matches the digest.

```
-- Encryption algorithm; could be 'DSA' instead; keys must
-- have been created using same algorithm
SET @algo = 'RSA';

-- Generate signature for digest and verify signature against digest
SET @sig = ASYMMETRIC_SIGN(@algo, @dig, @priv, @dig_type);
-- Verify signature against digest
SET @verf = ASYMMETRIC_VERIFY(@algo, @dig, @sig, @pub, @dig_type);
```

Create a symmetric key

This requires DH private/public keys as inputs, created using a shared symmetric secret. Create the secret by passing the key length to `CREATE_DH_PARAMETERS()`, then pass the secret as the “key length” to `CREATE_ASYMMETRIC_PRIV_KEY()`.

```
-- Generate DH shared symmetric secret
SET @dhp = CREATE_DH_PARAMETERS(1024);
-- Generate DH key pairs
SET @algo = 'DH';
SET @priv1 = CREATE_ASYMMETRIC_PRIV_KEY(@algo, @dhp);
SET @pub1 = CREATE_ASYMMETRIC_PUB_KEY(@algo, @priv1);
SET @priv2 = CREATE_ASYMMETRIC_PRIV_KEY(@algo, @dhp);
SET @pub2 = CREATE_ASYMMETRIC_PUB_KEY(@algo, @priv2);

-- Generate symmetric key using public key of first party,
-- private key of second party
SET @sym1 = ASYMMETRIC_DERIVE(@pub1, @priv2);

-- Or use public key of second party, private key of first party
SET @sym2 = ASYMMETRIC_DERIVE(@pub2, @priv1);
```

Key string values can be created at runtime and stored into a variable or table using `SET`, `SELECT`, or `INSERT`:

```
SET @priv1 = CREATE_ASYMMETRIC_PRIV_KEY('RSA', 1024);
SELECT CREATE_ASYMMETRIC_PRIV_KEY('RSA', 1024) INTO @priv2;
INSERT INTO t (key_col) VALUES(CREATE_ASYMMETRIC_PRIV_KEY('RSA', 1024));
```

Key string values stored in files can be read using the `LOAD_FILE()` function by users who have the `FILE` privilege.

Digest and signature strings can be handled similarly.

Limit CPU usage by key-generation operations

The `CREATE_ASYMMETRIC_PRIV_KEY()` and `CREATE_DH_PARAMETERS()` encryption functions take a key-length parameter, and the amount of CPU resources required by these functions increases as the key length increases. For some installations, this might result in unacceptable CPU usage if applications frequently generate excessively long keys.

OpenSSL imposes a minimum key length of 1,024 bits for all keys. OpenSSL also imposes a maximum key length of 10,000 bits and 16,384 bits for DSA and RSA keys, respectively, for `CREATE_ASYMMETRIC_PRIV_KEY()`, and a maximum key length of 10,000 bits for `CREATE_DH_PARAMETERS()`. If those maximum values are too high, three environment variables are available to enable MySQL server administrators to set lower maximum lengths for key generation, and thereby to limit CPU usage:

- `MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD`: Maximum DSA key length in bits for `CREATE_ASYMMETRIC_PRIV_KEY()`. The minimum and maximum values for this variable are 1,024 and 10,000.
- `MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD`: Maximum RSA key length in bits for `CREATE_ASYMMETRIC_PRIV_KEY()`. The minimum and maximum values for this variable are 1,024 and 16,384.
- `MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD`: Maximum key length in bits for `CREATE_DH_PARAMETERS()`. The minimum and maximum values for this variable are 1,024 and 10,000.

To use any of these environment variables, set them in the environment of the process that starts the server. If set, their values take precedence over the maximum key lengths imposed by OpenSSL. For example, to set a maximum key length of 4,096 bits for DSA and RSA keys for `CREATE_ASYMMETRIC_PRIV_KEY()`, set these variables:

```
export MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD=4096
export MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD=4096
```

The example uses Bourne shell syntax. The syntax for other shells may differ.

12.19.3 MySQL Enterprise Encryption Function Reference

Table 12.24 MySQL Enterprise Encryption Functions

Name	Description
`ASYMMETRIC_DECRYPT()`	Decrypt ciphertext using private or public key
`ASYMMETRIC_DERIVE()`	Derive symmetric key from asymmetric keys
`ASYMMETRIC_ENCRYPT()`	Encrypt cleartext using private or public key
`ASYMMETRIC_SIGN()`	Generate signature from digest
`ASYMMETRIC_VERIFY()`	Verify that signature matches digest
`CREATE_ASYMMETRIC_PRIV_KEY()`	Create private key
`CREATE_ASYMMETRIC_PUB_KEY()`	Create public key
`CREATE_DH_PARAMETERS()`	Generate shared DH secret
`CREATE_DIGEST()`	Generate digest from string

12.19.4 MySQL Enterprise Encryption Function Descriptions

MySQL Enterprise Encryption functions have these general characteristics:

- For arguments of the wrong type or an incorrect number of arguments, each function returns an error.
- If the arguments are not suitable to permit a function to perform the requested operation, it returns `NULL` or 0 as appropriate. This occurs, for example, if a function does not support a specified algorithm, a key length is too short or long, or a string expected to be a key string in PEM format is not a valid key. (OpenSSL imposes its own key-length limits, and server administrators can impose additional limits on maximum key length by setting environment variables. See Section 12.19.2, "MySQL Enterprise Encryption Usage and Examples".)
- The underlying SSL library takes care of randomness initialization.

Several of the functions take an encryption algorithm argument. The following table summarizes the supported algorithms by function.

Table 12.25 Supported Algorithms by Function

Function	Supported Algorithms
`ASYMMETRIC_DECRYPT()`	RSA
`ASYMMETRIC_DERIVE()`	DH
`ASYMMETRIC_ENCRYPT()`	RSA
`ASYMMETRIC_SIGN()`	RSA, DSA
`ASYMMETRIC_VERIFY()`	RSA, DSA
`CREATE_ASYMMETRIC_PRIV_KEY()`	RSA, DSA, DH
`CREATE_ASYMMETRIC_PUB_KEY()`	RSA, DSA, DH
`CREATE_DH_PARAMETERS()`	DH

Note

Although you can create keys using any of the RSA, DSA, or DH encryption algorithms, other functions that take key arguments might accept only certain types of keys. For example, `ASYMMETRIC_ENCRYPT()` and `ASYMMETRIC_DECRYPT()` accept only RSA keys.

The following descriptions describe the calling sequences for MySQL Enterprise Encryption functions. For additional examples and discussion, see Section 12.19.2, "MySQL Enterprise Encryption Usage and Examples".

- `ASYMMETRIC_DECRYPT(`*`algorithm, crypt_str, key_str`*`)`

  Decrypts an encrypted string using the given algorithm and key string, and returns the resulting plaintext as a binary string. If decryption fails, the result is `NULL`.

  `key_str` must be a valid key string in PEM format. For successful decryption, it must be the public or private key string corresponding to the private or public key string used with `ASYMMETRIC_ENCRYPT()` to produce the encrypted string. *`algorithm`* indicates the encryption algorithm used to create the key.

  Supported *`algorithm`* values: `'RSA'`

  For a usage example, see the description of `ASYMMETRIC_ENCRYPT()`.

- `ASYMMETRIC_DERIVE(pub_key_str, priv_key_str)`

  Derives a symmetric key using the private key of one party and the public key of another, and returns the resulting key as a binary string. If key derivation fails, the result is `NULL`.

  `pub_key_str` and *`priv_key_str`* must be valid key strings in PEM format. They must be created using the DH algorithm.

  Suppose that you have two pairs of public and private keys:

```
SET @dhp = CREATE_DH_PARAMETERS(1024);
SET @priv1 = CREATE_ASYMMETRIC_PRIV_KEY('DH', @dhp);
SET @pub1 = CREATE_ASYMMETRIC_PUB_KEY('DH', @priv1);
SET @priv2 = CREATE_ASYMMETRIC_PRIV_KEY('DH', @dhp);
SET @pub2 = CREATE_ASYMMETRIC_PUB_KEY('DH', @priv2);
```

  Suppose further that you use the private key from one pair and the public key from the other pair to create a symmetric key string. Then this symmetric key identity relationship holds:

```
ASYMMETRIC_DERIVE(@pub1, @priv2) = ASYMMETRIC_DERIVE(@pub2, @priv1)
```

- `ASYMMETRIC_ENCRYPT(algorithm, str, key_str)`

  Encrypts a string using the given algorithm and key string, and returns the resulting ciphertext as a binary string. If encryption fails, the result is `NULL`.

  The *`str`* length cannot be greater than the *`key_str`* length − 11, in bytes

  `key_str` must be a valid key string in PEM format. *`algorithm`* indicates the encryption algorithm used to create the key.

  Supported *`algorithm`* values: `'RSA'`

  To encrypt a string, pass a private or public key string to `ASYMMETRIC_ENCRYPT()`. To recover the original unencrypted string, pass the encrypted string to `ASYMMETRIC_DECRYPT()`, along with the public or private key string correponding to the private or public key string used for encryption.

```
-- Generate private/public key pair
SET @priv = CREATE_ASYMMETRIC_PRIV_KEY('RSA', 1024);
SET @pub = CREATE_ASYMMETRIC_PUB_KEY('RSA', @priv);

-- Encrypt using private key, decrypt using public key
SET @ciphertext = ASYMMETRIC_ENCRYPT('RSA', 'The quick brown fox', @priv);
SET @plaintext = ASYMMETRIC_DECRYPT('RSA', @ciphertext, @pub);

-- Encrypt using public key, decrypt using private key
SET @ciphertext = ASYMMETRIC_ENCRYPT('RSA', 'The quick brown fox', @pub);
SET @plaintext = ASYMMETRIC_DECRYPT('RSA', @ciphertext, @priv);
```

  Suppose that:

```
SET @s = a string to be encrypted
SET @priv = a valid private RSA key string in PEM format
SET @pub = the corresponding public RSA key string in PEM format
```

  Then these identity relationships hold:

```
```

```
ASYMMETRIC_DECRYPT('RSA', ASYMMETRIC_ENCRYPT('RSA', @s, @priv), @pub) = @s
ASYMMETRIC_DECRYPT('RSA', ASYMMETRIC_ENCRYPT('RSA', @s, @pub), @priv) = @s
```

- `ASYMMETRIC_SIGN(algorithm, digest_str, priv_key_str, digest_type)`

  Signs a digest string using a private key string, and returns the signature as a binary string. If signing fails, the result is `NULL`.

  `digest_str` is the digest string. It can be generated by calling `CREATE_DIGEST()`. *`digest_type`* indicates the digest algorithm used to generate the digest string.

  `priv_key_str` is the private key string to use for signing the digest string. It must be a valid key string in PEM format. *`algorithm`* indicates the encryption algorithm used to create the key.

  Supported *`algorithm`* values: `'RSA'`, `'DSA'`

  Supported *`digest_type`* values: `'SHA224'`, `'SHA256'`, `'SHA384'`, `'SHA512'`

  For a usage example, see the description of `ASYMMETRIC_VERIFY()`.

- `ASYMMETRIC_VERIFY(algorithm, digest_str, sig_str, pub_key_str, digest_type)`

  Verifies whether the signature string matches the digest string, and returns 1 or 0 to indicate whether verification succeeded or failed.

  `digest_str` is the digest string. It can be generated by calling `CREATE_DIGEST()`. *`digest_type`* indicates the digest algorithm used to generate the digest string.

  `sig_str` is the signature string. It can be generated by calling `ASYMMETRIC_SIGN()`.

  `pub_key_str` is the public key string of the signer. It corresponds to the private key passed to `ASYMMETRIC_SIGN()` to generate the signature string and must be a valid key string in PEM format. *`algorithm`* indicates the encryption algorithm used to create the key.

  Supported *`algorithm`* values: `'RSA'`, `'DSA'`

  Supported *`digest_type`* values: `'SHA224'`, `'SHA256'`, `'SHA384'`, `'SHA512'`

```
-- Set the encryption algorithm and digest type
SET @algo = 'RSA';
SET @dig_type = 'SHA224';

-- Create private/public key pair
SET @priv = CREATE_ASYMMETRIC_PRIV_KEY(@algo, 1024);
SET @pub = CREATE_ASYMMETRIC_PUB_KEY(@algo, @priv);

-- Generate digest from string
SET @dig = CREATE_DIGEST(@dig_type, 'The quick brown fox');

-- Generate signature for digest and verify signature against digest
SET @sig = ASYMMETRIC_SIGN(@algo, @dig, @priv, @dig_type);
SET @verf = ASYMMETRIC_VERIFY(@algo, @dig, @sig, @pub, @dig_type);
```

- `CREATE_ASYMMETRIC_PRIV_KEY(algorithm, {key_len|dh_secret})`

  Creates a private key using the given algorithm and key length or DH secret, and returns the key as a binary string in PEM format. If key generation fails, the result is `NULL`.

  Supported *`algorithm`* values: `'RSA'`, `'DSA'`, `'DH'`

Supported *key_len* values: The minimum key length in bits is 1,024. The maximum key length depends on the algorithm: 16,384 for RSA and 10,000 for DSA. These key-length limits are constraints imposed by OpenSSL. Server administrators can impose additional limits on maximum key length by setting environment variables. See Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”.

For DH keys, pass a shared DH secret instead of a key length. To create the secret, pass the key length to `CREATE_DH_PARAMETERS()`.

This example creates a 2,048-bit DSA private key, then derives a public key from the private key:

```
SET @priv = CREATE_ASYMMETRIC_PRIV_KEY('DSA', 2048);
SET @pub = CREATE_ASYMMETRIC_PUB_KEY('DSA', @priv);
```

For an example showing DH key generation, see the description of `ASYMMETRIC_DERIVE()`.

Some general considerations in choosing key lengths and encryption algorithms:

- The strength of encryption for private and public keys increases with the key size, but the time for key generation increases as well.
- Generation of DH keys takes much longer than RSA or RSA keys.
- Asymmetric encryption functions are slower than symmetric functions. If performance is an important factor and the functions are to be used very frequently, you are better off using symmetric encryption. For example, consider using `AES_ENCRYPT()` and `AES_DECRYPT()`.

- `CREATE_ASYMMETRIC_PUB_KEY(`*algorithm*`, `*priv_key_str*`)`

  Derives a public key from the given private key using the given algorithm, and returns the key as a binary string in PEM format. If key derivation fails, the result is `NULL`.

  priv_key_str must be a valid key string in PEM format. *algorithm* indicates the encryption algorithm used to create the key.

  Supported *algorithm* values: `'RSA'`, `'DSA'`, `'DH'`

  For a usage example, see the description of `CREATE_ASYMMETRIC_PRIV_KEY()`.

- `CREATE_DH_PARAMETERS(`*key_len*`)`

  Creates a shared secret for generating a DH private/public key pair and returns a binary string that can be passed to `CREATE_ASYMMETRIC_PRIV_KEY()`. If secret generation fails, the result is null.

  Supported *key_len* values: The minimum and maximum key lengths in bits are 1,024 and 10,000. These key-length limits are constraints imposed by OpenSSL. Server administrators can impose additional limits on maximum key length by setting environment variables. See Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”.

  For an example showing how to use the return value for generating symmetric keys, see the description of `ASYMMETRIC_DERIVE()`.

  ```
  SET @dhp = CREATE_DH_PARAMETERS(1024);
  ```

- `CREATE_DIGEST(`*digest_type*`, `*str*`)`

Creates a digest from the given string using the given digest type, and returns the digest as a binary string. If digest generation fails, the result is `NULL`.

Supported *`digest_type`* values: `'SHA224'`, `'SHA256'`, `'SHA384'`, `'SHA512'`

```
SET @dig = CREATE_DIGEST('SHA512', The quick brown fox');
```

The resulting digest string is suitable for use with `ASYMMETRIC_SIGN()` and `ASYMMETRIC_VERIFY()`.

12.20 Aggregate (GROUP BY) Functions

12.20.1 Aggregate (GROUP BY) Function Descriptions

This section describes group (aggregate) functions that operate on sets of values.

Table 12.26 Aggregate (GROUP BY) Functions

Name	Description
`AVG()`	Return the average value of the argument
`BIT_AND()`	Return bitwise AND
`BIT_OR()`	Return bitwise OR
`BIT_XOR()`	Return bitwise XOR
`COUNT()`	Return a count of the number of rows returned
`COUNT(DISTINCT)`	Return the count of a number of different values
`GROUP_CONCAT()`	Return a concatenated string
`JSON_ARRAYAGG()`	Return result set as a single JSON array
`JSON_OBJECTAGG()`	Return result set as a single JSON object
`MAX()`	Return the maximum value
`MIN()`	Return the minimum value
`STD()`	Return the population standard deviation
`STDDEV()`	Return the population standard deviation
`STDDEV_POP()`	Return the population standard deviation
`STDDEV_SAMP()`	Return the sample standard deviation
`SUM()`	Return the sum
`VAR_POP()`	Return the population standard variance
`VAR_SAMP()`	Return the sample variance
`VARIANCE()`	Return the population standard variance

Unless otherwise stated, group functions ignore `NULL` values.

If you use a group function in a statement containing no `GROUP BY` clause, it is equivalent to grouping on all rows. For more information, see Section 12.20.3, “MySQL Handling of GROUP BY”.

Most aggregate functions can be used as window functions. Those that can be used this way are signified in their syntax description by `[`*`over_clause`*`]`, representing an optional `OVER` clause. *`over_clause`* is described in Section 12.21.2, “Window Function Concepts and Syntax”, which also includes other information about window function usage.

For numeric arguments, the variance and standard deviation functions return a `DOUBLE` value. The `SUM()` and `AVG()` functions return a `DECIMAL` value for exact-value arguments (integer or `DECIMAL`), and a `DOUBLE` value for approximate-value arguments (`FLOAT` or `DOUBLE`).

The `SUM()` and `AVG()` aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

Functions such as `SUM()` or `AVG()` that expect a numeric argument cast the argument to a number if necessary. For `SET` or `ENUM` values, the cast operation causes the underlying numeric value to be used.

The `BIT_AND()`, `BIT_OR()`, and `BIT_XOR()` aggregate functions perform bit operations. Prior to MySQL 8.0, bit functions and operators required `BIGINT` (64-bit integer) arguments and returned `BIGINT` values, so they had a maximum range of 64 bits. Non-`BIGINT` arguments were converted to `BIGINT` prior to performing the operation and truncation could occur.

In MySQL 8.0, bit functions and operators permit binary string type arguments (`BINARY`, `VARBINARY`, and the `BLOB` types) and return a value of like type, which enables them to take arguments and produce return values larger than 64 bits. For discussion about argument evaluation and result types for bit operations, see the introductory discussion in Section 12.12, “Bit Functions and Operators”.

- `AVG([DISTINCT] expr) [over_clause]`

  Returns the average value of *`expr`*. The `DISTINCT` option can be used to return the average of the distinct values of *`expr`*.

  If there are no matching rows, `AVG()` returns `NULL`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”; it cannot be used with `DISTINCT`.

  ```
  mysql> SELECT student_name, AVG(test_score)
         FROM student
         GROUP BY student_name;
  ```

- `BIT_AND(expr) [over_clause]`

  Returns the bitwise `AND` of all bits in *`expr`*.

  The result type depends on whether the function argument values are evaluated as binary strings or numbers:

  - Binary-string evaluation occurs when the argument values have a binary string type, and the argument is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument value conversion to unsigned 64-bit integers as necessary.
  - Binary-string evaluation produces a binary string of the same length as the argument values. If argument values have unequal lengths, an `ER_INVALID_BITWISE_OPERANDS_SIZE` error occurs. If the argument size exceeds 511 bytes, an `ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE` error occurs. Numeric evaluation produces an unsigned 64-bit integer.

  If there are no matching rows, `BIT_AND()` returns a neutral value (all bits set to 1) having the same length as the argument values.

`NULL` values do not affect the result unless all values are `NULL`. In that case, the result is a neutral value having the same length as the argument values.

For more information discussion about argument evaluation and result types, see the introductory discussion in Section 12.12, “Bit Functions and Operators”.

As of MySQL 8.0.12, this function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

- `BIT_OR(`*`expr`*`) [`*`over_clause`*`]`

  Returns the bitwise `OR` of all bits in *`expr`*.

  The result type depends on whether the function argument values are evaluated as binary strings or numbers:

  - Binary-string evaluation occurs when the argument values have a binary string type, and the argument is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument value conversion to unsigned 64-bit integers as necessary.
  - Binary-string evaluation produces a binary string of the same length as the argument values. If argument values have unequal lengths, an `ER_INVALID_BITWISE_OPERANDS_SIZE` error occurs. If the argument size exceeds 511 bytes, an `ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE` error occurs. Numeric evaluation produces an unsigned 64-bit integer.

  If there are no matching rows, `BIT_OR()` returns a neutral value (all bits set to 0) having the same length as the argument values.

  `NULL` values do not affect the result unless all values are `NULL`. In that case, the result is a neutral value having the same length as the argument values.

  For more information discussion about argument evaluation and result types, see the introductory discussion in Section 12.12, “Bit Functions and Operators”.

  As of MySQL 8.0.12, this function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

- `BIT_XOR(`*`expr`*`) [`*`over_clause`*`]`

  Returns the bitwise `XOR` of all bits in *`expr`*.

  The result type depends on whether the function argument values are evaluated as binary strings or numbers:

  - Binary-string evaluation occurs when the argument values have a binary string type, and the argument is not a hexadecimal literal, bit literal, or `NULL` literal. Numeric evaluation occurs otherwise, with argument value conversion to unsigned 64-bit integers as necessary.
  - Binary-string evaluation produces a binary string of the same length as the argument values. If argument values have unequal lengths, an `ER_INVALID_BITWISE_OPERANDS_SIZE` error occurs. If the argument size exceeds 511 bytes, an `ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE` error occurs. Numeric evaluation produces an unsigned 64-bit integer.

  If there are no matching rows, `BIT_XOR()` returns a neutral value (all bits set to 0) having the same length as the argument values.

`NULL` values do not affect the result unless all values are `NULL`. In that case, the result is a neutral value having the same length as the argument values.

For more information discussion about argument evaluation and result types, see the introductory discussion in Section 12.12, “Bit Functions and Operators”.

As of MySQL 8.0.12, this function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

- `COUNT(`*`expr`*`) [`*`over_clause`*`]`

  Returns a count of the number of non-`NULL` values of *`expr`* in the rows retrieved by a `SELECT` statement. The result is a `BIGINT` value.

  If there are no matching rows, `COUNT()` returns `0`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

```
mysql> SELECT student.student_name,COUNT(*)
       FROM student,course
       WHERE student.student_id=course.student_id
       GROUP BY student_name;
```

  `COUNT(*)` is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain `NULL` values.

  For transactional storage engines such as `InnoDB`, storing an exact row count is problematic. Multiple transactions may be occurring at the same time, each of which may affect the count.

  `InnoDB` does not keep an internal count of rows in a table because concurrent transactions might “see” different numbers of rows at the same time. Consequently, `SELECT COUNT(*)` statements only count rows visible to the current transaction.

  As of MySQL 8.0.13, `SELECT COUNT(*) FROM` *`tbl_name`* query performance for `InnoDB` tables is optimized for single-threaded workloads if there are no extra clauses such as `WHERE` or `GROUP BY`.

  `InnoDB` processes `SELECT COUNT(*)` statements by traversing the smallest available secondary index unless an index or optimizer hint directs the optimizer to use a different index. If a secondary index is not present, `InnoDB` processes `SELECT COUNT(*)` statements by scanning the clustered index.

  Processing `SELECT COUNT(*)` statements takes some time if index records are not entirely in the buffer pool. For a faster count, create a counter table and let your application update it according to the inserts and deletes it does. However, this method may not scale well in situations where thousands of concurrent transactions are initiating updates to the same counter table. If an approximate row count is sufficient, use `SHOW TABLE STATUS`.

  `InnoDB` handles `SELECT COUNT(*)` and `SELECT COUNT(1)` operations in the same way. There is no performance difference.

  For `MyISAM` tables, `COUNT(*)` is optimized to return very quickly if the `SELECT` retrieves from one table, no other columns are retrieved, and there is no `WHERE` clause. For example:

```
mysql> SELECT COUNT(*) FROM student;
```

This optimization only applies to `MyISAM` tables, because an exact row count is stored for this storage engine and can be accessed very quickly. `COUNT(1)` is only subject to the same optimization if the first column is defined as `NOT NULL`.

- `COUNT(DISTINCT expr,[expr...])`

  Returns a count of the number of rows with different non-`NULL` *`expr`* values.

  If there are no matching rows, `COUNT(DISTINCT)` returns `0`.

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

  In MySQL, you can obtain the number of distinct expression combinations that do not contain `NULL` by giving a list of expressions. In standard SQL, you would have to do a concatenation of all expressions inside `COUNT(DISTINCT ...)`.

- `GROUP_CONCAT(expr)`

  This function returns a string result with the concatenated non-`NULL` values from a group. It returns `NULL` if there are no non-`NULL` values. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

```
mysql> SELECT student_name,
         GROUP_CONCAT(test_score)
       FROM student
       GROUP BY student_name;
```

  Or:

```
mysql> SELECT student_name,
         GROUP_CONCAT(DISTINCT test_score
                      ORDER BY test_score DESC SEPARATOR ' ')
       FROM student
       GROUP BY student_name;
```

  In MySQL, you can get the concatenated values of expression combinations. To eliminate duplicate values, use the `DISTINCT` clause. To sort values in the result, use the `ORDER BY` clause. To sort in reverse order, add the `DESC` (descending) keyword to the name of the column you are sorting by in the `ORDER BY` clause. The default is ascending order; this may be specified explicitly using the `ASC` keyword. The default separator between values in a group is comma (`,`). To specify a separator explicitly, use `SEPARATOR` followed by the string literal value that should be inserted between group values. To eliminate the separator altogether, specify `SEPARATOR ''`.

  The result is truncated to the maximum length that is given by the `group_concat_max_len` system variable, which has a default value of 1024. The value can be set higher, although the effective maximum length of the return value is constrained by the value of `max_allowed_packet`. The syntax to change the value of `group_concat_max_len` at runtime is as follows, where *`val`* is an unsigned integer:

```
SET [GLOBAL | SESSION] group_concat_max_len = val;
```

The return value is a nonbinary or binary string, depending on whether the arguments are nonbinary or binary strings. The result type is `TEXT` or `BLOB` unless `group_concat_max_len` is less than or equal to 512, in which case the result type is `VARCHAR` or `VARBINARY`.

See also `CONCAT()` and `CONCAT_WS()`: Section 12.5, “String Functions and Operators”.

- `JSON_ARRAYAGG(col_or_expr) [over_clause]`

  Aggregates a result set as a single `JSON` array whose elements consist of the rows. The order of elements in this array is undefined. The function acts on a column or an expression that evaluates to a single value. Returns `NULL` if the result contains no rows, or in the event of an error.

  As of MySQL 8.0.14, this function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

```
mysql> SELECT o_id, attribute, value FROM t3;
+------+-----------+--------+
| o_id | attribute | value  |
+------+-----------+--------+
|    2 | color     | red    |
|    2 | fabric    | silk   |
|    3 | color     | green  |
|    3 | shape     | square |
+------+-----------+--------+
4 rows in set (0.00 sec)

mysql> SELECT o_id, JSON_ARRAYAGG(attribute) AS attributes
       FROM t3 GROUP BY o_id;
+------+---------------------+
| o_id | attributes          |
+------+---------------------+
|    2 | ["color", "fabric"] |
|    3 | ["color", "shape"]  |
+------+---------------------+
2 rows in set (0.00 sec)
```

- `JSON_OBJECTAGG(key, value) [over_clause]`

  Takes two column names or expressions as arguments, the first of these being used as a key and the second as a value, and returns a JSON object containing key-value pairs. Returns `NULL` if the result contains no rows, or in the event of an error. An error occurs if any key name is `NULL` or the number of arguments is not equal to 2.

  As of MySQL 8.0.14, this function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

```
mysql> SELECT o_id, attribute, value FROM t3;
+------+-----------+--------+
| o_id | attribute | value  |
+------+-----------+--------+
|    2 | color     | red    |
|    2 | fabric    | silk   |
|    3 | color     | green  |
|    3 | shape     | square |
+------+-----------+--------+
4 rows in set (0.00 sec)

mysql> SELECT o_id, JSON_OBJECTAGG(attribute, value)
       FROM t3 GROUP BY o_id;
+------+----------------------------------------+
```

```
| o_id | JSON_OBJECTAGG(attribute, value)      |
+------+---------------------------------------+
|    2 | {"color": "red", "fabric": "silk"}    |
|    3 | {"color": "green", "shape": "square"} |
+------+---------------------------------------+
2 rows in set (0.00 sec)
```

Duplicate key handling

When the result of this function is normalized, values having duplicate keys are discarded. In keeping with the MySQL `JSON` data type specification that does not permit duplicate keys, only the last value encountered is used with that key in the returned object (“last duplicate key wins”). This means that the result of using this function on columns from a `SELECT` can depend on the order in which in the rows are returned, which is not guaranteed. When used as a window function, if there are duplicate keys within a frame, only the last value for the key is present in the result. The value for the key from last row in the frame is deterministic if the `ORDER BY` specification guarantees that the values have a specific order. If not, the resulting value of the key is nondeterministic. Consider the following:

```
mysql> CREATE TABLE t(c VARCHAR(10), i INT);
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO t VALUES
       ('key', 3), ('key', 4), ('key', 5);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT c, i FROM t;
+------+------+
| c    | i    |
+------+------+
| key  |    3 |
| key  |    4 |
| key  |    5 |
+------+------+
3 rows in set (0.00 sec)

mysql> SELECT JSON_OBJECTAGG(c, i) FROM t;
+----------------------+
| JSON_OBJECTAGG(c, i) |
+----------------------+
| {"key": 5}           |
+----------------------+
1 row in set (0.00 sec)

mysql> DELETE FROM t;
Query OK, 3 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES
       ('key', 3), ('key', 5), ('key', 4);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT c, i FROM t;
+------+------+
| c    | i    |
+------+------+
| key  |    3 |
| key  |    5 |
| key  |    4 |
+------+------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT JSON_OBJECTAGG(c, i) FROM t;
+----------------------+
| JSON_OBJECTAGG(c, i) |
+----------------------+
| {"key": 4}           |
+----------------------+
1 row in set (0.00 sec)
```

The key chosen from the last query is nondeterministic. If you prefer a particular key ordering, you can invoke `JSON_OBJECTAGG()` as a window function by including an `OVER` clause with an `ORDER BY` specification to impose a particular order on frame rows. The following examples show what happens with and without `ORDER BY` for a few different frame specifications.

Without `ORDER BY`, the frame is the entire partition:

```
mysql> SELECT JSON_OBJECTAGG(c, i)
       OVER () AS json_object FROM t;
+-------------+
| json_object |
+-------------+
| {"key": 4}  |
| {"key": 4}  |
| {"key": 4}  |
+-------------+
```

With `ORDER BY`, where the frame is the default of `RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW` (in both ascending and descending order):

```
mysql> SELECT JSON_OBJECTAGG(c, i)
       OVER (ORDER BY i) AS json_object FROM t;
+-------------+
| json_object |
+-------------+
| {"key": 3}  |
| {"key": 4}  |
| {"key": 5}  |
+-------------+
mysql> SELECT JSON_OBJECTAGG(c, i)
       OVER (ORDER BY i DESC) AS json_object FROM t;
+-------------+
| json_object |
+-------------+
| {"key": 5}  |
| {"key": 4}  |
| {"key": 3}  |
+-------------+
```

With `ORDER BY` and an explicit frame of the entire partition:

```
mysql> SELECT JSON_OBJECTAGG(c, i)
       OVER (ORDER BY i
            ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
        AS json_object
       FROM t;
+-------------+
| json_object |
+-------------+
| {"key": 5}  |
| {"key": 5}  |
| {"key": 5}  |
+-------------+
```

To return a particular key value (such as the smallest or largest), include a `LIMIT` clause in the appropriate query. For example:

```
mysql> SELECT JSON_OBJECTAGG(c, i)
       OVER (ORDER BY i) AS json_object FROM t LIMIT 1;
+-------------+
| json_object |
+-------------+
| {"key": 3}  |
+-------------+
mysql> SELECT JSON_OBJECTAGG(c, i)
       OVER (ORDER BY i DESC) AS json_object FROM t LIMIT 1;
+-------------+
| json_object |
+-------------+
| {"key": 5}  |
+-------------+
```

See Normalization, Merging, and Autowrapping of JSON Values, for additional information and examples.

- `MAX([DISTINCT] expr) [over_clause]`

  Returns the maximum value of *`expr`*. `MAX()` may take a string argument; in such cases, it returns the maximum string value. See Section 8.3.1, “How MySQL Uses Indexes”. The `DISTINCT` keyword can be used to find the maximum of the distinct values of *`expr`*, however, this produces the same result as omitting `DISTINCT`.

  If there are no matching rows, `MAX()` returns `NULL`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”; it cannot be used with `DISTINCT`.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
       FROM student
       GROUP BY student_name;
```

  For `MAX()`, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set. This differs from how `ORDER BY` compares them.

- `MIN([DISTINCT] expr) [over_clause]`

  Returns the minimum value of *`expr`*. `MIN()` may take a string argument; in such cases, it returns the minimum string value. See Section 8.3.1, “How MySQL Uses Indexes”. The `DISTINCT` keyword can be used to find the minimum of the distinct values of *`expr`*, however, this produces the same result as omitting `DISTINCT`.

  If there are no matching rows, `MIN()` returns `NULL`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”; it cannot be used with `DISTINCT`.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
       FROM student
       GROUP BY student_name;
```

For `MIN()`, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set. This differs from how `ORDER BY` compares them.

- `STD(expr) [over_clause]`

  Returns the population standard deviation of *`expr`*. `STD()` is a synonym for the standard SQL function `STDDEV_POP()`, provided as a MySQL extension.

  If there are no matching rows, `STD()` returns `NULL`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

- `STDDEV(expr) [over_clause]`

  Returns the population standard deviation of *`expr`*. `STDDEV()` is a synonym for the standard SQL function `STDDEV_POP()`, provided for compatibility with Oracle.

  If there are no matching rows, `STDDEV()` returns `NULL`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

- `STDDEV_POP(expr) [over_clause]`

  Returns the population standard deviation of *`expr`* (the square root of `VAR_POP()`). You can also use `STD()` or `STDDEV()`, which are equivalent but not standard SQL.

  If there are no matching rows, `STDDEV_POP()` returns `NULL`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

- `STDDEV_SAMP(expr) [over_clause]`

  Returns the sample standard deviation of *`expr`* (the square root of `VAR_SAMP()`.

  If there are no matching rows, `STDDEV_SAMP()` returns `NULL`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

- `SUM([DISTINCT] expr) [over_clause]`

  Returns the sum of *`expr`*. If the return set has no rows, `SUM()` returns `NULL`. The `DISTINCT` keyword can be used to sum only the distinct values of *`expr`*.

  If there are no matching rows, `SUM()` returns `NULL`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”; it cannot be used with `DISTINCT`.

- `VAR_POP(expr) [over_clause]`

  Returns the population standard variance of *`expr`*. It considers rows as the whole population, not as a sample, so it has the number of rows as the denominator. You can also use `VARIANCE()`, which is equivalent but is not standard SQL.

If there are no matching rows, `VAR_POP()` returns `NULL`.

This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

- `VAR_SAMP(`*`expr`*`) [`*`over_clause`*`]`

  Returns the sample variance of *`expr`*. That is, the denominator is the number of rows minus one.

  If there are no matching rows, `VAR_SAMP()` returns `NULL`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

- `VARIANCE(`*`expr`*`) [`*`over_clause`*`]`

  Returns the population standard variance of *`expr`*. `VARIANCE()` is a synonym for the standard SQL function `VAR_POP()`, provided as a MySQL extension.

  If there are no matching rows, `VARIANCE()` returns `NULL`.

  This function executes as a window function if *`over_clause`* is present. *`over_clause`* is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

12.20.2 GROUP BY Modifiers

The `GROUP BY` clause permits a `WITH ROLLUP` modifier that causes summary output to include extra rows that represent higher-level (that is, super-aggregate) summary operations. `ROLLUP` thus enables you to answer questions at multiple levels of analysis with a single query. For example, `ROLLUP` can be used to provide support for OLAP (Online Analytical Processing) operations.

Suppose that a `sales` table has `year`, `country`, `product`, and `profit` columns for recording sales profitability:

```
CREATE TABLE sales
(
    year    INT,
    country VARCHAR(20),
    product VARCHAR(32),
    profit  INT
);
```

To summarize table contents per year, use a simple `GROUP BY` like this:

```
mysql> SELECT year, SUM(profit) AS profit
       FROM sales
       GROUP BY year;
+------+--------+
| year | profit |
+------+--------+
| 2000 |   4525 |
| 2001 |   3010 |
+------+--------+
```

The output shows the total (aggregate) profit for each year. To also determine the total profit summed over all years, you must add up the individual values yourself or run an additional query. Or you can use `ROLLUP`, which provides both levels of analysis with a single query. Adding a `WITH ROLLUP` modifier to the

`GROUP BY` clause causes the query to produce another (super-aggregate) row that shows the grand total over all year values:

```
mysql> SELECT year, SUM(profit) AS profit
       FROM sales
       GROUP BY year WITH ROLLUP;
+------+--------+
| year | profit |
+------+--------+
| 2000 |   4525 |
| 2001 |   3010 |
| NULL |   7535 |
+------+--------+
```

The `NULL` value in the `year` column identifies the grand total super-aggregate line.

`ROLLUP` has a more complex effect when there are multiple `GROUP BY` columns. In this case, each time there is a change in value in any but the last grouping column, the query produces an extra super-aggregate summary row.

For example, without `ROLLUP`, a summary of the `sales` table based on `year`, `country`, and `product` might look like this, where the output indicates summary values only at the year/country/product level of analysis:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product;
+------+---------+------------+--------+
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2000 | India   | Calculator |    150 |
| 2000 | India   | Computer   |   1200 |
| 2000 | USA     | Calculator |     75 |
| 2000 | USA     | Computer   |   1500 |
| 2001 | Finland | Phone      |     10 |
| 2001 | USA     | Calculator |     50 |
| 2001 | USA     | Computer   |   2700 |
| 2001 | USA     | TV         |    250 |
+------+---------+------------+--------+
```

With `ROLLUP` added, the query produces several extra rows:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+--------+
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2000 | Finland | NULL       |   1600 |
| 2000 | India   | Calculator |    150 |
| 2000 | India   | Computer   |   1200 |
| 2000 | India   | NULL       |   1350 |
| 2000 | USA     | Calculator |     75 |
| 2000 | USA     | Computer   |   1500 |
| 2000 | USA     | NULL       |   1575 |
| 2000 | NULL    | NULL       |   4525 |
| 2001 | Finland | Phone      |     10 |
| 2001 | Finland | NULL       |     10 |
| 2001 | USA     | Calculator |     50 |
```

```
| 2001 | USA     | Computer   |   2700 |
| 2001 | USA     | TV         |    250 |
| 2001 | USA     | NULL       |   3000 |
| 2001 | NULL    | NULL       |   3010 |
| NULL | NULL    | NULL       |   7535 |
+------+---------+------------+--------+
```

Now the output includes summary information at four levels of analysis, not just one:

- Following each set of product rows for a given year and country, an extra super-aggregate summary row appears showing the total for all products. These rows have the `product` column set to `NULL`.
- Following each set of rows for a given year, an extra super-aggregate summary row appears showing the total for all countries and products. These rows have the `country` and `products` columns set to `NULL`.
- Finally, following all other rows, an extra super-aggregate summary row appears showing the grand total for all years, countries, and products. This row has the `year`, `country`, and `products` columns set to `NULL`.

The `NULL` indicators in each super-aggregate row are produced when the row is sent to the client. The server looks at the columns named in the `GROUP BY` clause following the leftmost one that has changed value. For any column in the result set with a name that matches any of those names, its value is set to `NULL`. (If you specify grouping columns by column position, the server identifies which columns to set to `NULL` by position.)

Because the `NULL` values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you can test them as `NULL` values only in the select list or `HAVING` clause. You cannot test them as `NULL` values in join conditions or the `WHERE` clause to determine which rows to select. For example, you cannot add `WHERE product IS NULL` to the query to eliminate from the output all but the super-aggregate rows.

The `NULL` values do appear as `NULL` on the client side and can be tested as such using any MySQL client programming interface. However, at this point, you cannot distinguish whether a `NULL` represents a regular grouped value or a super-aggregate value. To test the distinction, use the `GROUPING()` function, described later.

Previously, MySQL did not allow the use of `DISTINCT` or `ORDER BY` in a query having a `WITH ROLLUP` option. This restriction is lifted in MySQL 8.0.12 and later. (Bug #87450, Bug #86311, Bug #26640100, Bug #26073513)

For `GROUP BY ... WITH ROLLUP` queries, to test whether `NULL` values in the result represent super-aggregate values, the `GROUPING()` function is available for use in the select list, `HAVING` clause, and (as of MySQL 8.0.12) `ORDER BY` clause. For example, `GROUPING(year)` returns 1 when `NULL` in the `year` column occurs in a super-aggregate row, and 0 otherwise. Similarly, `GROUPING(country)` and `GROUPING(product)` return 1 for super-aggregate `NULL` values in the `country` and `product` columns, respectively:

```
mysql> SELECT
         year, country, product, SUM(profit) AS profit,
         GROUPING(year) AS grp_year,
         GROUPING(country) AS grp_country,
         GROUPING(product) AS grp_product
       FROM sales
       GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+--------+----------+-------------+-------------+
| year | country | product    | profit | grp_year | grp_country | grp_product |
+------+---------+------------+--------+----------+-------------+-------------+
| 2000 | Finland | Computer   |   1500 |        0 |           0 |           0 |
| 2000 | Finland | Phone      |    100 |        0 |           0 |           0 |
```

```
| 2000 | Finland | NULL       |   1600 |        0 |           0 |           1 |
| 2000 | India   | Calculator |    150 |        0 |           0 |           0 |
| 2000 | India   | Computer   |   1200 |        0 |           0 |           0 |
| 2000 | India   | NULL       |   1350 |        0 |           0 |           1 |
| 2000 | USA     | Calculator |     75 |        0 |           0 |           0 |
| 2000 | USA     | Computer   |   1500 |        0 |           0 |           0 |
| 2000 | USA     | NULL       |   1575 |        0 |           0 |           1 |
| 2000 | NULL    | NULL       |   4525 |        0 |           1 |           1 |
| 2001 | Finland | Phone      |     10 |        0 |           0 |           0 |
| 2001 | Finland | NULL       |     10 |        0 |           0 |           1 |
| 2001 | USA     | Calculator |     50 |        0 |           0 |           0 |
| 2001 | USA     | Computer   |   2700 |        0 |           0 |           0 |
| 2001 | USA     | TV         |    250 |        0 |           0 |           0 |
| 2001 | USA     | NULL       |   3000 |        0 |           0 |           1 |
| 2001 | NULL    | NULL       |   3010 |        0 |           1 |           1 |
| NULL | NULL    | NULL       |   7535 |        1 |           1 |           1 |
+------+---------+------------+--------+----------+-------------+-------------+
```

Instead of displaying the `GROUPING()` results directly, you can use `GROUPING()` to substitute labels for super-aggregate `NULL` values:

```
mysql> SELECT
         IF(GROUPING(year), 'All years', year) AS year,
         IF(GROUPING(country), 'All countries', country) AS country,
         IF(GROUPING(product), 'All products', product) AS product,
         SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product WITH ROLLUP;
+-----------+---------------+-------------+--------+
| year      | country       | product     | profit |
+-----------+---------------+-------------+--------+
| 2000      | Finland       | Computer    |   1500 |
| 2000      | Finland       | Phone       |    100 |
| 2000      | Finland       | All products |   1600 |
| 2000      | India         | Calculator  |    150 |
| 2000      | India         | Computer    |   1200 |
| 2000      | India         | All products |   1350 |
| 2000      | USA           | Calculator  |     75 |
| 2000      | USA           | Computer    |   1500 |
| 2000      | USA           | All products |   1575 |
| 2000      | All countries | All products |   4525 |
| 2001      | Finland       | Phone       |     10 |
| 2001      | Finland       | All products |     10 |
| 2001      | USA           | Calculator  |     50 |
| 2001      | USA           | Computer    |   2700 |
| 2001      | USA           | TV          |    250 |
| 2001      | USA           | All products |   3000 |
| 2001      | All countries | All products |   3010 |
| All years | All countries | All products |   7535 |
+-----------+---------------+-------------+--------+
```

With multiple expression arguments, `GROUPING()` returns a result representing a bitmask the combines the results for each expression, with the lowest-order bit corresponding to the result for the rightmost expression. For example, `GROUPING(year, country, product)` is evaluated like this:

```
  result for GROUPING(product)
+ result for GROUPING(country) << 1
+ result for GROUPING(year) << 2
```

The result of such a `GROUPING()` is nonzero if any of the expressions represents a super-aggregate `NULL`, so you can return only the super-aggregate rows and filter out the regular grouped rows like this:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
```

```
       FROM sales
       GROUP BY year, country, product WITH ROLLUP
       HAVING GROUPING(year, country, product) <> 0;
+------+---------+---------+--------+
| year | country | product | profit |
+------+---------+---------+--------+
| 2000 | Finland | NULL    |   1600 |
| 2000 | India   | NULL    |   1350 |
| 2000 | USA     | NULL    |   1575 |
| 2000 | NULL    | NULL    |   4525 |
| 2001 | Finland | NULL    |     10 |
| 2001 | USA     | NULL    |   3000 |
| 2001 | NULL    | NULL    |   3010 |
| NULL | NULL    | NULL    |   7535 |
+------+---------+---------+--------+
```

The `sales` table contains no `NULL` values, so all `NULL` values in a `ROLLUP` result represent super-aggregate values. When the data set contains `NULL` values, `ROLLUP` summaries may contain `NULL` values not only in super-aggregate rows, but also in regular grouped rows. `GROUPING()` enables these to be distinguished. Suppose that table `t1` contains a simple data set with two grouping factors for a set of quantity values, where `NULL` indicates something like “other” or “unknown”:

```
mysql> SELECT * FROM t1;
+------+-------+----------+
| name | size  | quantity |
+------+-------+----------+
| ball | small |       10 |
| ball | large |       20 |
| ball | NULL  |        5 |
| hoop | small |       15 |
| hoop | large |        5 |
| hoop | NULL  |        3 |
+------+-------+----------+
```

A simple `ROLLUP` operation produces these results, in which it is not so easy to distinguish `NULL` values in super-aggregate rows from `NULL` values in regular grouped rows:

```
mysql> SELECT name, size, SUM(quantity) AS quantity
       FROM t1
       GROUP BY name, size WITH ROLLUP;
+------+-------+----------+
| name | size  | quantity |
+------+-------+----------+
| ball | NULL  |        5 |
| ball | large |       20 |
| ball | small |       10 |
| ball | NULL  |       35 |
| hoop | NULL  |        3 |
| hoop | large |        5 |
| hoop | small |       15 |
| hoop | NULL  |       23 |
| NULL | NULL  |       58 |
+------+-------+----------+
```

Using `GROUPING()` to substitute labels for the super-aggregate `NULL` values makes the result easier to interpret:

```
mysql> SELECT
         IF(GROUPING(name) = 1, 'All items', name) AS name,
         IF(GROUPING(size) = 1, 'All sizes', size) AS size,
         SUM(quantity) AS quantity
       FROM t1
       GROUP BY name, size WITH ROLLUP;
```

```
+-----------+-----------+----------+
| name      | size      | quantity |
+-----------+-----------+----------+
| ball      | NULL      |        5 |
| ball      | large     |       20 |
| ball      | small     |       10 |
| ball      | All sizes |       35 |
| hoop      | NULL      |        3 |
| hoop      | large     |        5 |
| hoop      | small     |       15 |
| hoop      | All sizes |       23 |
| All items | All sizes |       58 |
+-----------+-----------+----------+
```

Other Considerations When using ROLLUP

The following discussion lists some behaviors specific to the MySQL implementation of `ROLLUP`.

Prior to MySQL 8.0.12, when you use `ROLLUP`, you cannot also use an `ORDER BY` clause to sort the results. In other words, `ROLLUP` and `ORDER BY` were mutually exclusive in MySQL. However, you still have some control over sort order. To work around the restriction that prevents using `ROLLUP` with `ORDER BY` and achieve a specific sort order of grouped results, generate the grouped result set as a derived table and apply `ORDER BY` to it. For example:

```
mysql> SELECT * FROM
         (SELECT year, SUM(profit) AS profit
         FROM sales GROUP BY year WITH ROLLUP) AS dt
       ORDER BY year DESC;
+------+--------+
| year | profit |
+------+--------+
| 2001 |   3010 |
| 2000 |   4525 |
| NULL |   7535 |
+------+--------+
```

As of MySQL 8.0.12, `ORDER BY` and `ROLLUP` can be used together, which enables the use of `ORDER BY` and `GROUPING()` to achieve a specific sort order of grouped results. For example:

```
mysql> SELECT year, SUM(profit) AS profit
       FROM sales
       GROUP BY year WITH ROLLUP
       ORDER BY GROUPING(year) DESC;
+------+--------+
| year | profit |
+------+--------+
| NULL |   7535 |
| 2000 |   4525 |
| 2001 |   3010 |
+------+--------+
```

In both cases, the super-aggregate summary rows sort with the rows from which they are calculated, and their placement depends on sort order (at the end for ascending sort, at the beginning for descending sort).

`LIMIT` can be used to restrict the number of rows returned to the client. `LIMIT` is applied after `ROLLUP`, so the limit applies against the extra rows added by `ROLLUP`. For example:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product WITH ROLLUP
       LIMIT 5;
+------+---------+------------+--------+
```

```
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2000 | Finland | NULL       |   1600 |
| 2000 | India   | Calculator |    150 |
| 2000 | India   | Computer   |   1200 |
+------+---------+------------+--------+
```

Using `LIMIT` with `ROLLUP` may produce results that are more difficult to interpret, because there is less context for understanding the super-aggregate rows.

A MySQL extension permits a column that does not appear in the `GROUP BY` list to be named in the select list. (For information about nonaggregated columns and `GROUP BY`, see Section 12.20.3, “MySQL Handling of GROUP BY”.) In this case, the server is free to choose any value from this nonaggregated column in summary rows, and this includes the extra rows added by `WITH ROLLUP`. For example, in the following query, `country` is a nonaggregated column that does not appear in the `GROUP BY` list and values chosen for this column are nondeterministic:

```
mysql> SELECT year, country, SUM(profit) AS profit
       FROM sales
       GROUP BY year WITH ROLLUP;
+------+---------+--------+
| year | country | profit |
+------+---------+--------+
| 2000 | India   |   4525 |
| 2001 | USA     |   3010 |
| NULL | USA     |   7535 |
+------+---------+--------+
```

This behavior is permitted when the `ONLY_FULL_GROUP_BY` SQL mode is not enabled. If that mode is enabled, the server rejects the query as illegal because `country` is not listed in the `GROUP BY` clause. With `ONLY_FULL_GROUP_BY` enabled, you can still execute the query by using the `ANY_VALUE()` function for nondeterministic-value columns:

```
mysql> SELECT year, ANY_VALUE(country) AS country, SUM(profit) AS profit
       FROM sales
       GROUP BY year WITH ROLLUP;
+------+---------+--------+
| year | country | profit |
+------+---------+--------+
| 2000 | India   |   4525 |
| 2001 | USA     |   3010 |
| NULL | USA     |   7535 |
+------+---------+--------+
```

12.20.3 MySQL Handling of GROUP BY

SQL-92 and earlier does not permit queries for which the select list, `HAVING` condition, or `ORDER BY` list refer to nonaggregated columns that are not named in the `GROUP BY` clause. For example, this query is illegal in standard SQL-92 because the nonaggregated `name` column in the select list does not appear in the `GROUP BY`:

```
SELECT o.custid, c.name, MAX(o.payment)
  FROM orders AS o, customers AS c
  WHERE o.custid = c.custid
  GROUP BY o.custid;
```

For the query to be legal in SQL-92, the `name` column must be omitted from the select list or named in the `GROUP BY` clause.

SQL:1999 and later permits such nonaggregates per optional feature T301 if they are functionally dependent on `GROUP BY` columns: If such a relationship exists between `name` and `custid`, the query is legal. This would be the case, for example, were `custid` a primary key of `customers`.

MySQL implements detection of functional dependence. If the `ONLY_FULL_GROUP_BY` SQL mode is enabled (which it is by default), MySQL rejects queries for which the select list, `HAVING` condition, or `ORDER BY` list refer to nonaggregated columns that are neither named in the `GROUP BY` clause nor are functionally dependent on them.

If `ONLY_FULL_GROUP_BY` is disabled, a MySQL extension to the standard SQL use of `GROUP BY` permits the select list, `HAVING` condition, or `ORDER BY` list to refer to nonaggregated columns even if the columns are not functionally dependent on `GROUP BY` columns. This causes MySQL to accept the preceding query. In this case, the server is free to choose any value from each group, so unless they are the same, the values chosen are nondeterministic, which is probably not what you want. Furthermore, the selection of values from each group cannot be influenced by adding an `ORDER BY` clause. Result set sorting occurs after values have been chosen, and `ORDER BY` does not affect which value within each group the server chooses. Disabling `ONLY_FULL_GROUP_BY` is useful primarily when you know that, due to some property of the data, all values in each nonaggregated column not named in the `GROUP BY` are the same for each group.

You can achieve the same effect without disabling `ONLY_FULL_GROUP_BY` by using `ANY_VALUE()` to refer to the nonaggregated column.

The following discussion demonstrates functional dependence, the error message MySQL produces when functional dependence is absent, and ways of causing MySQL to accept a query in the absence of functional dependence.

This query might be invalid with `ONLY_FULL_GROUP_BY` enabled because the nonaggregated `address` column in the select list is not named in the `GROUP BY` clause:

```
SELECT name, address, MAX(age) FROM t GROUP BY name;
```

The query is valid if `name` is a primary key of `t` or is a unique `NOT NULL` column. In such cases, MySQL recognizes that the selected column is functionally dependent on a grouping column. For example, if `name` is a primary key, its value determines the value of `address` because each group has only one value of the primary key and thus only one row. As a result, there is no randomness in the choice of `address` value in a group and no need to reject the query.

The query is invalid if `name` is not a primary key of `t` or a unique `NOT NULL` column. In this case, no functional dependency can be inferred and an error occurs:

```
mysql> SELECT name, address, MAX(age) FROM t GROUP BY name;
ERROR 1055 (42000): Expression #2 of SELECT list is not in GROUP
BY clause and contains nonaggregated column 'mydb.t.address' which
is not functionally dependent on columns in GROUP BY clause; this
is incompatible with sql_mode=only_full_group_by
```

If you know that, *for a given data set,* each `name` value in fact uniquely determines the `address` value, `address` is effectively functionally dependent on `name`. To tell MySQL to accept the query, you can use the `ANY_VALUE()` function:

```
SELECT name, ANY_VALUE(address), MAX(age) FROM t GROUP BY name;
```

Alternatively, disable `ONLY_FULL_GROUP_BY`.

The preceding example is quite simple, however. In particular, it is unlikely you would group on a single primary key column because every group would contain only one row. For addtional examples demonstrating functional dependence in more complex queries, see Section 12.20.4, “Detection of Functional Dependence”.

If a query has aggregate functions and no `GROUP BY` clause, it cannot have nonaggregated columns in the select list, `HAVING` condition, or `ORDER BY` list with `ONLY_FULL_GROUP_BY` enabled:

```
mysql> SELECT name, MAX(age) FROM t;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression
#1 of SELECT list contains nonaggregated column 'mydb.t.name'; this
is incompatible with sql_mode=only_full_group_by
```

Without `GROUP BY`, there is a single group and it is nondeterministic which `name` value to choose for the group. Here, too, `ANY_VALUE()` can be used, if it is immaterial which `name` value MySQL chooses:

```
SELECT ANY_VALUE(name), MAX(age) FROM t;
```

`ONLY_FULL_GROUP_BY` also affects handling of queries that use `DISTINCT` and `ORDER BY`. Consider the case of a table `t` with three columns `c1`, `c2`, and `c3` that contains these rows:

```
c1 c2 c3
1  2  A
3  4  B
1  2  C
```

Suppose that we execute the following query, expecting the results to be ordered by `c3`:

```
SELECT DISTINCT c1, c2 FROM t ORDER BY c3;
```

To order the result, duplicates must be eliminated first. But to do so, should we keep the first row or the third? This arbitrary choice influences the retained value of `c3`, which in turn influences ordering and makes it arbitrary as well. To prevent this problem, a query that has `DISTINCT` and `ORDER BY` is rejected as invalid if any `ORDER BY` expression does not satisfy at least one of these conditions:

- The expression is equal to one in the select list
- All columns referenced by the expression and belonging to the query's selected tables are elements of the select list

Another MySQL extension to standard SQL permits references in the `HAVING` clause to aliased expressions in the select list. For example, the following query returns `name` values that occur only once in table `orders`:

```
SELECT name, COUNT(name) FROM orders
  GROUP BY name
  HAVING COUNT(name) = 1;
```

The MySQL extension permits the use of an alias in the `HAVING` clause for the aggregated column:

```
SELECT name, COUNT(name) AS c FROM orders
  GROUP BY name
  HAVING c = 1;
```

Standard SQL permits only column expressions in `GROUP BY` clauses, so a statement such as this is invalid because `FLOOR(value/100)` is a noncolumn expression:

```
SELECT id, FLOOR(value/100)
  FROM tbl_name
  GROUP BY id, FLOOR(value/100);
```

MySQL extends standard SQL to permit noncolumn expressions in `GROUP BY` clauses and considers the preceding statement valid.

Standard SQL also does not permit aliases in `GROUP BY` clauses. MySQL extends standard SQL to permit aliases, so another way to write the query is as follows:

```
SELECT id, FLOOR(value/100) AS val
  FROM tbl_name
  GROUP BY id, val;
```

The alias `val` is considered a column expression in the `GROUP BY` clause.

In the presence of a noncolumn expression in the `GROUP BY` clause, MySQL recognizes equality between that expression and expressions in the select list. This means that with `ONLY_FULL_GROUP_BY` SQL mode enabled, the query containing `GROUP BY id, FLOOR(value/100)` is valid because that same `FLOOR()` expression occurs in the select list. However, MySQL does not try to recognize functional dependence on `GROUP BY` noncolumn expressions, so the following query is invalid with `ONLY_FULL_GROUP_BY` enabled, even though the third selected expression is a simple formula of the `id` column and the `FLOOR()` expression in the `GROUP BY` clause:

```
SELECT id, FLOOR(value/100), id+FLOOR(value/100)
  FROM tbl_name
  GROUP BY id, FLOOR(value/100);
```

A workaround is to use a derived table:

```
SELECT id, F, id+F
  FROM
    (SELECT id, FLOOR(value/100) AS F
     FROM tbl_name
     GROUP BY id, FLOOR(value/100)) AS dt;
```

12.20.4 Detection of Functional Dependence

The following discussion provides several examples of the ways in which MySQL detects functional dependencies. The examples use this notation:

```
{X} -> {Y}
```

Understand this as "*X* uniquely determines *Y*," which also means that *Y* is functionally dependent on *X*.

The examples use the `world` database, which can be downloaded from https://dev.mysql.com/doc/index-other.html. You can find details on how to install the database on the same page.

- Functional Dependencies Derived from Keys
- Functional Dependencies Derived from Multiple-Column Keys and from Equalities
- Functional Dependency Special Cases
- Functional Dependencies and Views
- Combinations of Functional Dependencies

Functional Dependencies Derived from Keys

The following query selects, for each country, a count of spoken languages:

```
SELECT co.Name, COUNT(*)
FROM countrylanguage cl, country co
WHERE cl.CountryCode = co.Code
GROUP BY co.Code;
```

`co.Code` is a primary key of `co`, so all columns of `co` are functionally dependent on it, as expressed using this notation:

```
{co.Code} -> {co.*}
```

Thus, `co.name` is functionally dependent on `GROUP BY` columns and the query is valid.

A `UNIQUE` index over a `NOT NULL` column could be used instead of a primary key and the same functional dependence would apply. (This is not true for a `UNIQUE` index that permits `NULL` values because it permits multiple `NULL` values and in that case uniqueness is lost.)

Functional Dependencies Derived from Multiple-Column Keys and from Equalities

This query selects, for each country, a list of all spoken languages and how many people speak them:

```
SELECT co.Name, cl.Language,
cl.Percentage * co.Population / 100.0 AS SpokenBy
FROM countrylanguage cl, country co
WHERE cl.CountryCode = co.Code
GROUP BY cl.CountryCode, cl.Language;
```

The pair (`cl.CountryCode`, `cl.Language`) is a two-column composite primary key of `cl`, so that column pair uniquely determines all columns of `cl`:

```
{cl.CountryCode, cl.Language} -> {cl.*}
```

Moreover, because of the equality in the `WHERE` clause:

```
{cl.CountryCode} -> {co.Code}
```

And, because `co.Code` is primary key of `co`:

```
{co.Code} -> {co.*}
```

“Uniquely determines” relationships are transitive, therefore:

```
{cl.CountryCode, cl.Language} -> {cl.*,co.*}
```

As a result, the query is valid.

As with the previous example, a `UNIQUE` key over `NOT NULL` columns could be used instead of a primary key.

An `INNER JOIN` condition can be used instead of `WHERE`. The same functional dependencies apply:

```
SELECT co.Name, cl.Language,
cl.Percentage * co.Population/100.0 AS SpokenBy
FROM countrylanguage cl INNER JOIN country co
```

```
ON cl.CountryCode = co.Code
GROUP BY cl.CountryCode, cl.Language;
```

Functional Dependency Special Cases

Whereas an equality test in a `WHERE` condition or `INNER JOIN` condition is symmetric, an equality test in an outer join condition is not, because tables play different roles.

Assume that referential integrity has been accidentally broken and there exists a row of `countrylanguage` without a corresponding row in `country`. Consider the same query as in the previous example, but with a `LEFT JOIN`:

```
SELECT co.Name, cl.Language,
cl.Percentage * co.Population/100.0 AS SpokenBy
FROM countrylanguage cl LEFT JOIN country co
ON cl.CountryCode = co.Code
GROUP BY cl.CountryCode, cl.Language;
```

For a given value of `cl.CountryCode`, the value of `co.Code` in the join result is either found in a matching row (determined by `cl.CountryCode`) or is `NULL`-complemented if there is no match (also determined by `cl.CountryCode`). In each case, this relationship applies:

```
{cl.CountryCode} -> {co.Code}
```

`cl.CountryCode` is itself functionally dependent on {`cl.CountryCode`, `cl.Language`} which is a primary key.

If in the join result `co.Code` is `NULL`-complemented, `co.Name` is as well. If `co.Code` is not `NULL`-complemented, then because `co.Code` is a primary key, it determines `co.Name`. Therefore, in all cases:

```
{co.Code} -> {co.Name}
```

Which yields:

```
{cl.CountryCode, cl.Language} -> {cl.*,co.*}
```

As a result, the query is valid.

However, suppose that the tables are swapped, as in this query:

```
SELECT co.Name, cl.Language,
cl.Percentage * co.Population/100.0 AS SpokenBy
FROM country co LEFT JOIN countrylanguage cl
ON cl.CountryCode = co.Code
GROUP BY cl.CountryCode, cl.Language;
```

Now this relationship does *not* apply:

```
{cl.CountryCode, cl.Language} -> {cl.*,co.*}
```

Indeed, all `NULL`-complemented rows made for `cl` will be put into a single group (they have both `GROUP BY` columns equal to `NULL`), and inside this group the value of `co.Name` can vary. The query is invalid and MySQL rejects it.

Functional dependence in outer joins is thus linked to whether determinant columns belong to the left or right side of the `LEFT JOIN`. Determination of functional dependence becomes more complex if there are nested outer joins or the join condition does not consist entirely of equality comparisons.

Functional Dependencies and Views

Suppose that a view on countries produces their code, their name in uppercase, and how many different official languages they have:

```
CREATE VIEW Country2 AS
SELECT co.Code, UPPER(co.Name) AS UpperName,
COUNT(cl.Language) AS OfficialLanguages
FROM country AS co JOIN countrylanguage AS cl
ON cl.CountryCode = co.Code
WHERE cl.isOfficial = 'T'
GROUP BY co.Code;
```

This definition is valid because:

```
{co.Code} -> {co.*}
```

In the view result, the first selected column is `co.Code`, which is also the group column and thus determines all other selected expressions:

```
{Country2.Code} -> {Country2.*}
```

MySQL understands this and uses this information, as described following.

This query displays countries, how many different official languages they have, and how many cities they have, by joining the view with the `city` table:

```
SELECT co2.Code, co2.UpperName, co2.OfficialLanguages,
COUNT(*) AS Cities
FROM country2 AS co2 JOIN city ci
ON ci.CountryCode = co2.Code
GROUP BY co2.Code;
```

This query is valid because, as seen previously:

```
{co2.Code} -> {co2.*}
```

MySQL is able to discover a functional dependency in the result of a view and use that to validate a query which uses the view. The same would be true if `country2` were a derived table (or common table expression), as in:

```
SELECT co2.Code, co2.UpperName, co2.OfficialLanguages,
COUNT(*) AS Cities
FROM
(
 SELECT co.Code, UPPER(co.Name) AS UpperName,
 COUNT(cl.Language) AS OfficialLanguages
 FROM country AS co JOIN countrylanguage AS cl
 ON cl.CountryCode=co.Code
 WHERE cl.isOfficial='T'
 GROUP BY co.Code
) AS co2
JOIN city ci ON ci.CountryCode = co2.Code
GROUP BY co2.Code;
```

Combinations of Functional Dependencies

MySQL is able to combine all of the preceding types of functional dependencies (key based, equality based, view based) to validate more complex queries.

12.21 Window Functions

MySQL supports window functions that, for each row from a query, perform a calculation using rows related to that row. The following sections discuss how to use window functions, including descriptions of the `OVER` and `WINDOW` clauses. The first section provides descriptions of the nonaggregate window functions. For descriptions of the aggregate window functions, see Section 12.20.1, "Aggregate (GROUP BY) Function Descriptions".

For information about optimization and window functions, see Section 8.2.1.21, "Window Function Optimization".

12.21.1 Window Function Descriptions

This section describes nonaggregate window functions that, for each row from a query, perform a calculation using rows related to that row. Most aggregate functions also can be used as window functions; see Section 12.20.1, "Aggregate (GROUP BY) Function Descriptions".

For window function usage information and examples, and definitions of terms such as the `OVER` clause, window, partition, frame, and peer, see Section 12.21.2, "Window Function Concepts and Syntax".

Table 12.27 Window Functions

Name	Description
`CUME_DIST()`	Cumulative distribution value
`DENSE_RANK()`	Rank of current row within its partition, without gaps
`FIRST_VALUE()`	Value of argument from first row of window frame
`LAG()`	Value of argument from row lagging current row within partition
`LAST_VALUE()`	Value of argument from last row of window frame
`LEAD()`	Value of argument from row leading current row within partition
`NTH_VALUE()`	Value of argument from N-th row of window frame
`NTILE()`	Bucket number of current row within its partition.
`PERCENT_RANK()`	Percentage rank value
`RANK()`	Rank of current row within its partition, with gaps
`ROW_NUMBER()`	Number of current row within its partition

In the following function descriptions, *`over_clause`* represents the `OVER` clause, described in Section 12.21.2, "Window Function Concepts and Syntax". Some window functions permit a *`null_treatment`* clause that specifies how to handle `NULL` values when calculating results. This clause is optional. It is part of the SQL standard, but the MySQL implementation permits only `RESPECT NULLS` (which is also the default). This means that `NULL` values are considered when calculating results. `IGNORE NULLS` is parsed, but produces an error.

- `CUME_DIST()` *`over_clause`*

  Returns the cumulative distribution of a value within a group of values; that is, the percentage of partition values less than or equal to the value in the current row. This represents the number of rows preceding or peer with the current row in the window ordering of the window partition divided by the total number of rows in the window partition. Return values range from 0 to 1.

  This function should be used with `ORDER BY` to sort partition rows into the desired order. Without `ORDER BY`, all rows are peers and have value *`N`*/*`N`* = 1, where *`N`* is the partition size.

over_clause is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

The following query shows, for the set of values in the `val` column, the `CUME_DIST()` value for each row, as well as the percentage rank value returned by the similar `PERCENT_RANK()` function. For reference, the query also displays row numbers using `ROW_NUMBER()`:

```
mysql> SELECT
         val,
         ROW_NUMBER()   OVER w AS 'row_number',
         CUME_DIST()    OVER w AS 'cume_dist',
         PERCENT_RANK() OVER w AS 'percent_rank'
       FROM numbers
       WINDOW w AS (ORDER BY val);
+------+------------+--------------------+--------------+
| val  | row_number | cume_dist          | percent_rank |
+------+------------+--------------------+--------------+
|    1 |          1 | 0.2222222222222222 |            0 |
|    1 |          2 | 0.2222222222222222 |            0 |
|    2 |          3 | 0.3333333333333333 |         0.25 |
|    3 |          4 | 0.6666666666666666 |        0.375 |
|    3 |          5 | 0.6666666666666666 |        0.375 |
|    3 |          6 | 0.6666666666666666 |        0.375 |
|    4 |          7 | 0.8888888888888888 |         0.75 |
|    4 |          8 | 0.8888888888888888 |         0.75 |
|    5 |          9 |                  1 |            1 |
+------+------------+--------------------+--------------+
```

- `DENSE_RANK()` *over_clause*

  Returns the rank of the current row within its partition, without gaps. Peers are considered ties and receive the same rank. This function assigns consecutive ranks to peer groups; the result is that groups of size greater than one do not produce noncontiguous rank numbers. For an example, see the `RANK()` function description.

  This function should be used with `ORDER BY` to sort partition rows into the desired order. Without `ORDER BY`, all rows are peers.

  over_clause is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

- `FIRST_VALUE(`*expr*`)` [*null_treatment*] *over_clause*

  Returns the value of *expr* from the first row of the window frame.

  over_clause is as described in Section 12.21.2, “Window Function Concepts and Syntax”. *null_treatment* is as described in the section introduction.

  The following query demonstrates `FIRST_VALUE()`, `LAST_VALUE()`, and two instances of `NTH_VALUE()`:

```
mysql> SELECT
         time, subject, val,
         FIRST_VALUE(val)  OVER w AS 'first',
         LAST_VALUE(val)   OVER w AS 'last',
         NTH_VALUE(val, 2) OVER w AS 'second',
         NTH_VALUE(val, 4) OVER w AS 'fourth'
       FROM observations
       WINDOW w AS (PARTITION BY subject ORDER BY time
                    ROWS UNBOUNDED PRECEDING);
+----------+---------+------+-------+------+--------+--------+
| time     | subject | val  | first | last | second | fourth |
```

```
+----------+---------+------+-------+------+--------+--------+
| 07:00:00 | st113   |   10 |    10 |   10 |   NULL |   NULL |
| 07:15:00 | st113   |    9 |    10 |    9 |      9 |   NULL |
| 07:30:00 | st113   |   25 |    10 |   25 |      9 |   NULL |
| 07:45:00 | st113   |   20 |    10 |   20 |      9 |     20 |
| 07:00:00 | xh458   |    0 |     0 |    0 |   NULL |   NULL |
| 07:15:00 | xh458   |   10 |     0 |   10 |     10 |   NULL |
| 07:30:00 | xh458   |    5 |     0 |    5 |     10 |   NULL |
| 07:45:00 | xh458   |   30 |     0 |   30 |     10 |     30 |
| 08:00:00 | xh458   |   25 |     0 |   25 |     10 |     30 |
+----------+---------+------+-------+------+--------+--------+
```

Each function uses the rows in the current frame, which, per the window definition shown, extends from the first partition row to the current row. For the `NTH_VALUE()` calls, the current frame does not always include the requested row; in such cases, the return value is `NULL`.

- `LAG(`*`expr [, N[, default]]`*`)` [*`null_treatment`*] *`over_clause`*

  Returns the value of *`expr`* from the row that lags (precedes) the current row by *`N`* rows within its partition. If there is no such row, the return value is *`default`*. For example, if *`N`* is 3, the return value is *`default`* for the first two rows. If *`N`* or *`default`* are missing, the defaults are 1 and `NULL`, respectively.

  `N` must be a literal nonnegative integer. If *`N`* is 0, *`expr`* is evaluated for the current row.

  `over_clause` is as described in Section 12.21.2, “Window Function Concepts and Syntax”. *`null_treatment`* is as described in the section introduction.

  `LAG()` (and the similar `LEAD()` function) are often used to compute differences between rows. The following query shows a set of time-ordered observations and, for each one, the `LAG()` and `LEAD()` values from the adjoining rows, as well as the differences between the current and adjoining rows:

```
mysql> SELECT
         t, val,
         LAG(val)        OVER w AS 'lag',
         LEAD(val)       OVER w AS 'lead',
         val - LAG(val)  OVER w AS 'lag diff',
         val - LEAD(val) OVER w AS 'lead diff'
       FROM series
       WINDOW w AS (ORDER BY t);
+----------+------+------+------+----------+-----------+
| t        | val  | lag  | lead | lag diff | lead diff |
+----------+------+------+------+----------+-----------+
| 12:00:00 |  100 | NULL |  125 |     NULL |       -25 |
| 13:00:00 |  125 |  100 |  132 |       25 |        -7 |
| 14:00:00 |  132 |  125 |  145 |        7 |       -13 |
| 15:00:00 |  145 |  132 |  140 |       13 |         5 |
| 16:00:00 |  140 |  145 |  150 |       -5 |       -10 |
| 17:00:00 |  150 |  140 |  200 |       10 |       -50 |
| 18:00:00 |  200 |  150 | NULL |       50 |      NULL |
+----------+------+------+------+----------+-----------+
```

In the example, the `LAG()` and `LEAD()` calls use the default *`N`* and *`default`* values of 1 and `NULL`, respectively.

The first row shows what happens when there is no previous row for `LAG()`: The function returns the *`default`* value (in this case, `NULL`). The last row shows the same thing when there is no next row for `LEAD()`.

`LAG()` and `LEAD()` also serve to compute sums rather than differences. Consider this data set, which contains the first few numbers of the Fibonacci series:

```
mysql> SELECT n FROM fib ORDER BY n;
+------+
| n    |
+------+
|    1 |
|    1 |
|    2 |
|    3 |
|    5 |
|    8 |
+------+
```

The following query shows the `LAG()` and `LEAD()` values for the rows adjacent to the current row. It also uses those functions to add to the current row value the values from the preceding and following rows. The effect is to generate the next number in the Fibonacci series, and the next number after that:

```
mysql> SELECT
         n,
         LAG(n, 1, 0)      OVER w AS 'lag',
         LEAD(n, 1, 0)     OVER w AS 'lead',
         n + LAG(n, 1, 0)  OVER w AS 'next_n',
         n + LEAD(n, 1, 0) OVER w AS 'next_next_n'
       FROM fib
       WINDOW w AS (ORDER BY n);
+------+------+------+--------+-------------+
| n    | lag  | lead | next_n | next_next_n |
+------+------+------+--------+-------------+
|    1 |    0 |    1 |      1 |           2 |
|    1 |    1 |    2 |      2 |           3 |
|    2 |    1 |    3 |      3 |           5 |
|    3 |    2 |    5 |      5 |           8 |
|    5 |    3 |    8 |      8 |          13 |
|    8 |    5 |    0 |     13 |           8 |
+------+------+------+--------+-------------+
```

One way to generate the initial set of Fibonacci numbers is to use a recursive common table expression. For an example, see Fibonacci Series Generation.

- `LAST_VALUE(`*`expr`*`)` [*`null_treatment`*] *`over_clause`*

  Returns the value of *`expr`* from the last row of the window frame.

  `over_clause` is as described in Section 12.21.2, “Window Function Concepts and Syntax”. *`null_treatment`* is as described in the section introduction.

  For an example, see the `FIRST_VALUE()` function description.

- `LEAD(`*`expr [, N[, default]]`*`)` [*`null_treatment`*] *`over_clause`*

  Returns the value of *`expr`* from the row that leads (follows) the current row by *`N`* rows within its partition. If there is no such row, the return value is *`default`*. For example, if *`N`* is 3, the return value is *`default`* for the last two rows. If *`N`* or *`default`* are missing, the defaults are 1 and `NULL`, respectively.

  `N` must be a literal nonnegative integer. If *`N`* is 0, *`expr`* is evaluated for the current row.

  `over_clause` is as described in Section 12.21.2, “Window Function Concepts and Syntax”. *`null_treatment`* is as described in the section introduction.

  For an example, see the `LAG()` function description.

- `NTH_VALUE(expr, N) [from_first_last] [null_treatment] over_clause`

  Returns the value of *`expr`* from the *`N`*-th row of the window frame. If there is no such row, the return value is `NULL`.

  `N` must be a literal positive integer.

  `from_first_last` is part of the SQL standard, but the MySQL implementation permits only `FROM FIRST` (which is also the default). This means that calculations begin at the first row of the window. `FROM LAST` is parsed, but produces an error. To obtain the same effect as `FROM LAST` (begin calculations at the last row of the window), use `ORDER BY` to sort in reverse order.

  `over_clause` is as described in Section 12.21.2, “Window Function Concepts and Syntax”. *`null_treatment`* is as described in the section introduction.

  For an example, see the `FIRST_VALUE()` function description.

- `NTILE(N) over_clause`

  Divides a partition into *`N`* groups (buckets), assigns each row in the partition its bucket number, and returns the bucket number of the current row within its partition. For example, if *`N`* is 4, `NTILE()` divides rows into four buckets. If *`N`* is 100, `NTILE()` divides rows into 100 buckets.

  `N` must be a literal positive integer. Bucket number return values range from 1 to *`N`*.

  This function should be used with `ORDER BY` to sort partition rows into the desired order.

  `over_clause` is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

  The following query shows, for the set of values in the `val` column, the percentile values resulting from dividing the rows into two or four groups. For reference, the query also displays row numbers using `ROW_NUMBER()`:

```
mysql> SELECT
         val,
         ROW_NUMBER() OVER w AS 'row_number',
         NTILE(2)     OVER w AS 'ntile2',
         NTILE(4)     OVER w AS 'ntile4'
       FROM numbers
       WINDOW w AS (ORDER BY val);
+------+------------+--------+--------+
| val  | row_number | ntile2 | ntile4 |
+------+------------+--------+--------+
|    1 |          1 |      1 |      1 |
|    1 |          2 |      1 |      1 |
|    2 |          3 |      1 |      1 |
|    3 |          4 |      1 |      2 |
|    3 |          5 |      1 |      2 |
|    3 |          6 |      2 |      3 |
|    4 |          7 |      2 |      3 |
|    4 |          8 |      2 |      4 |
|    5 |          9 |      2 |      4 |
+------+------------+--------+--------+
```

- `PERCENT_RANK() over_clause`

  Returns the percentage of partition values less than the value in the current row, excluding the highest value. Return values range from 0 to 1 and represent the row relative rank, calculated as the result of this formula, where *`rank`* is the row rank and *`rows`* is the number of partition rows:

```
(rank - 1) / (rows - 1)
```

This function should be used with `ORDER BY` to sort partition rows into the desired order. Without `ORDER BY`, all rows are peers.

`over_clause` is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

For an example, see the `CUME_DIST()` function description.

- `RANK()` *`over_clause`*

  Returns the rank of the current row within its partition, with gaps. Peers are considered ties and receive the same rank. This function does not assign consecutive ranks to peer groups if groups of size greater than one exist; the result is noncontiguous rank numbers.

  This function should be used with `ORDER BY` to sort partition rows into the desired order. Without `ORDER BY`, all rows are peers.

  `over_clause` is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

  The following query shows the difference between `RANK()`, which produces ranks with gaps, and `DENSE_RANK()`, which produces ranks without gaps. The query shows rank values for each member of a set of values in the `val` column, which contains some duplicates. `RANK()` assigns peers (the duplicates) the same rank value, and the next greater value has a rank higher by the number of peers minus one. `DENSE_RANK()` also assigns peers the same rank value, but the next higher value has a rank one greater. For reference, the query also displays row numbers using `ROW_NUMBER()`:

```
mysql> SELECT
         val,
         ROW_NUMBER() OVER w AS 'row_number',
         RANK()       OVER w AS 'rank',
         DENSE_RANK() OVER w AS 'dense_rank'
       FROM numbers
       WINDOW w AS (ORDER BY val);
+------+------------+------+------------+
| val  | row_number | rank | dense_rank |
+------+------------+------+------------+
|    1 |          1 |    1 |          1 |
|    1 |          2 |    1 |          1 |
|    2 |          3 |    3 |          2 |
|    3 |          4 |    4 |          3 |
|    3 |          5 |    4 |          3 |
|    3 |          6 |    4 |          3 |
|    4 |          7 |    7 |          4 |
|    4 |          8 |    7 |          4 |
|    5 |          9 |    9 |          5 |
+------+------------+------+------------+
```

- `ROW_NUMBER()` *`over_clause`*

  Returns the number of the current row within its partition. Rows numbers range from 1 to the number of partition rows.

  `ORDER BY` affects the order in which rows are numbered. Without `ORDER BY`, row numbering is nondeterministic.

  `ROW_NUMBER()` assigns peers different row numbers. To assign peers the same value, use `RANK()` or `DENSE_RANK()`. For an example, see the `RANK()` function description.

over_clause is as described in Section 12.21.2, “Window Function Concepts and Syntax”.

12.21.2 Window Function Concepts and Syntax

This section describes how to use window functions. Examples use the same sales information data set as found in the discussion of the GROUPING() function in Section 12.20.2, “GROUP BY Modifiers”:

```
mysql> SELECT * FROM sales ORDER BY country, year, product;
+------+---------+------------+--------+
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2001 | Finland | Phone      |     10 |
| 2000 | India   | Calculator |     75 |
| 2000 | India   | Calculator |     75 |
| 2000 | India   | Computer   |   1200 |
| 2000 | USA     | Calculator |     75 |
| 2000 | USA     | Computer   |   1500 |
| 2001 | USA     | Calculator |     50 |
| 2001 | USA     | Computer   |   1500 |
| 2001 | USA     | Computer   |   1200 |
| 2001 | USA     | TV         |    150 |
| 2001 | USA     | TV         |    100 |
+------+---------+------------+--------+
```

A window function performs an aggregate-like operation on a set of query rows. However, whereas an aggregate operation groups query rows into a single result row, a window function produces a result for each query row:

- The row for which function evaluation occurs is called the current row.
- The query rows related to the current row over which function evaluation occurs comprise the window for the current row.

For example, using the sales information table, these two queries perform aggregate operations that produce a single global sum for all rows taken as a group, and sums grouped per country:

```
mysql> SELECT SUM(profit) AS total_profit
       FROM sales;
+--------------+
| total_profit |
+--------------+
|         7535 |
+--------------+
mysql> SELECT country, SUM(profit) AS country_profit
       FROM sales
       GROUP BY country
       ORDER BY country;
+---------+----------------+
| country | country_profit |
+---------+----------------+
| Finland |           1610 |
| India   |           1350 |
| USA     |           4575 |
+---------+----------------+
```

By contrast, window operations do not collapse groups of query rows to a single output row. Instead, they produce a result for each row. Like the preceding queries, the following query uses SUM(), but this time as a window function:

```
mysql> SELECT
         year, country, product, profit,
         SUM(profit) OVER() AS total_profit,
         SUM(profit) OVER(PARTITION BY country) AS country_profit
       FROM sales
       ORDER BY country, year, product, profit;
+------+---------+------------+--------+--------------+----------------+
| year | country | product    | profit | total_profit | country_profit |
+------+---------+------------+--------+--------------+----------------+
| 2000 | Finland | Computer   |   1500 |         7535 |           1610 |
| 2000 | Finland | Phone      |    100 |         7535 |           1610 |
| 2001 | Finland | Phone      |     10 |         7535 |           1610 |
| 2000 | India   | Calculator |     75 |         7535 |           1350 |
| 2000 | India   | Calculator |     75 |         7535 |           1350 |
| 2000 | India   | Computer   |   1200 |         7535 |           1350 |
| 2000 | USA     | Calculator |     75 |         7535 |           4575 |
| 2000 | USA     | Computer   |   1500 |         7535 |           4575 |
| 2001 | USA     | Calculator |     50 |         7535 |           4575 |
| 2001 | USA     | Computer   |   1200 |         7535 |           4575 |
| 2001 | USA     | Computer   |   1500 |         7535 |           4575 |
| 2001 | USA     | TV         |    100 |         7535 |           4575 |
| 2001 | USA     | TV         |    150 |         7535 |           4575 |
+------+---------+------------+--------+--------------+----------------+
```

Each window operation in the query is signified by inclusion of an `OVER` clause that specifies how to partition query rows into groups for processing by the window function:

- The first `OVER` clause is empty, which treats the entire set of query rows as a single partition. The window function thus produces a global sum, but does so for each row.
- The second `OVER` clause partitions rows by country, producing a sum per partition (per country). The function produces this sum for each partition row.

Window functions are permitted only in the select list and `ORDER BY` clause. Query result rows are determined from the `FROM` clause, after `WHERE`, `GROUP BY`, and `HAVING` processing, and windowing execution occurs before `ORDER BY`, `LIMIT`, and `SELECT DISTINCT`.

The `OVER` clause is permitted for many aggregate functions, which therefore can be used as window or nonwindow functions, depending on whether the `OVER` clause is present or absent:

```
AVG()
BIT_AND()
BIT_OR()
BIT_XOR()
COUNT()
JSON_ARRAYAGG()
JSON_OBJECTAGG()
MAX()
MIN()
STDDEV_POP(), STDDEV(), STD()
STDDEV_SAMP()
SUM()
VAR_POP(), VARIANCE()
VAR_SAMP()
```

For details about each aggregate function, see Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”.

MySQL also supports nonaggregate functions that are used only as window functions. For these, the `OVER` clause is mandatory:

```
CUME_DIST()
DENSE_RANK()
FIRST_VALUE()
LAG()
LAST_VALUE()
LEAD()
NTH_VALUE()
NTILE()
PERCENT_RANK()
RANK()
ROW_NUMBER()
```

For details about each nonaggregate function, see Section 12.21.1, “Window Function Descriptions”.

As an example of one of those nonaggregate window functions, this query uses `ROW_NUMBER()`, which produces the row number of each row within its partition. In this case, rows are numbered per country. By default, partition rows are unordered and row numbering is nondeterministic. To sort partition rows, include an `ORDER BY` clause within the window definition. The query uses unordered and ordered partitions (the `row_num1` and `row_num2` columns) to illustrate the difference between omitting and including `ORDER BY`:

```
mysql> SELECT
         year, country, product, profit,
         ROW_NUMBER() OVER(PARTITION BY country) AS row_num1,
         ROW_NUMBER() OVER(PARTITION BY country ORDER BY year, product) AS row_num2
       FROM sales;
+------+---------+------------+--------+----------+----------+
| year | country | product    | profit | row_num1 | row_num2 |
+------+---------+------------+--------+----------+----------+
| 2000 | Finland | Computer   |   1500 |        2 |        1 |
| 2000 | Finland | Phone      |    100 |        1 |        2 |
| 2001 | Finland | Phone      |     10 |        3 |        3 |
| 2000 | India   | Calculator |     75 |        2 |        1 |
| 2000 | India   | Calculator |     75 |        3 |        2 |
| 2000 | India   | Computer   |   1200 |        1 |        3 |
| 2000 | USA     | Calculator |     75 |        5 |        1 |
| 2000 | USA     | Computer   |   1500 |        4 |        2 |
| 2001 | USA     | Calculator |     50 |        2 |        3 |
| 2001 | USA     | Computer   |   1500 |        3 |        4 |
| 2001 | USA     | Computer   |   1200 |        7 |        5 |
| 2001 | USA     | TV         |    150 |        1 |        6 |
| 2001 | USA     | TV         |    100 |        6 |        7 |
+------+---------+------------+--------+----------+----------+
```

As mentioned previously, to use a window function (or treat an aggregate function as a window function), include an `OVER` clause following the function call. The `OVER` clause has two forms:

```
over_clause:
    {OVER (window_spec) | OVER window_name}
```

Both forms define how the window function should process query rows. They differ in whether the window is defined directly in the `OVER` clause, or supplied by a reference to a named window defined elsewhere in the query:

- In the first case, the window specification appears directly in the `OVER` clause, between the parentheses.
- In the second case, *`window_name`* is the name for a window specification defined by a `WINDOW` clause elsewhere in the query. For details, see Section 12.21.4, “Named Windows”.

For `OVER (window_spec)` syntax, the window specification has several parts, all optional:

```
window_spec:
    [window_name] [partition_clause] [order_clause] [frame_clause]
```

If `OVER()` is empty, the window consists of all query rows and the window function computes a result using all rows. Otherwise, the clauses present within the parentheses determine which query rows are used to compute the function result and how they are partitioned and ordered:

- *`window_name`*: The name of a window defined by a `WINDOW` clause elsewhere in the query. If *`window_name`* appears by itself within the `OVER` clause, it completely defines the window. If partitioning, ordering, or framing clauses are also given, they modify interpretation of the named window. For details, see Section 12.21.4, “Named Windows”.

- *`partition_clause`*: A `PARTITION BY` clause indicates how to divide the query rows into groups. The window function result for a given row is based on the rows of the partition that contains the row. If `PARTITION BY` is omitted, there is a single partition consisting of all query rows.

  Note

  Partitioning for window functions differs from table partitioning. For information about table partitioning, see Chapter 23, *Partitioning*.

  `partition_clause` has this syntax:

  ```
  partition_clause:
      PARTITION BY expr [, expr] ...
  ```

  Standard SQL requires `PARTITION BY` to be followed by column names only. A MySQL extension is to permit expressions, not just column names. For example, if a table contains a `TIMESTAMP` column named `ts`, standard SQL permits `PARTITION BY ts` but not `PARTITION BY HOUR(ts)`, whereas MySQL permits both.

- *`order_clause`*: An `ORDER BY` clause indicates how to sort rows in each partition. Partition rows that are equal according to the `ORDER BY` clause are considered peers. If `ORDER BY` is omitted, partition rows are unordered, with no processing order implied, and all partition rows are peers.

  `order_clause` has this syntax:

  ```
  order_clause:
      ORDER BY expr [ASC|DESC] [, expr [ASC|DESC]] ...
  ```

  Each `ORDER BY` expression optionally can be followed by `ASC` or `DESC` to indicate sort direction. The default is `ASC` if no direction is specified. `NULL` values sort first for ascending sorts, last for descending sorts.

  An `ORDER BY` in a window definition applies within individual partitions. To sort the result set as a whole, include an `ORDER BY` at the query top level.

- *`frame_clause`*: A frame is a subset of the current partition and the frame clause specifies how to define the subset. The frame clause has many subclauses of its own. For details, see Section 12.21.3, “Window Function Frame Specification”.

12.21.3 Window Function Frame Specification

The definition of a window used with a window function can include a frame clause. A frame is a subset of the current partition and the frame clause specifies how to define the subset.

Frames are determined with respect to the current row, which enables a frame to move within a partition depending on the location of the current row within its partition. Examples:

- By defining a frame to be all rows from the partition start to the current row, you can compute running totals for each row.
- By defining a frame as extending *N* rows on either side of the current row, you can compute rolling averages.

The following query demonstrates the use of moving frames to compute running totals within each group of time-ordered `level` values, as well as rolling averages computed from the current row and the rows that immediately precede and follow it:

```
mysql> SELECT
         time, subject, val,
         SUM(val) OVER (PARTITION BY subject ORDER BY time
                        ROWS UNBOUNDED PRECEDING)
           AS running_total,
         AVG(val) OVER (PARTITION BY subject ORDER BY time
                        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
           AS running_average
       FROM observations;
+----------+---------+------+---------------+-----------------+
| time     | subject | val  | running_total | running_average |
+----------+---------+------+---------------+-----------------+
| 07:00:00 | st113   |   10 |            10 |          9.5000 |
| 07:15:00 | st113   |    9 |            19 |         14.6667 |
| 07:30:00 | st113   |   25 |            44 |         18.0000 |
| 07:45:00 | st113   |   20 |            64 |         22.5000 |
| 07:00:00 | xh458   |    0 |             0 |          5.0000 |
| 07:15:00 | xh458   |   10 |            10 |          5.0000 |
| 07:30:00 | xh458   |    5 |            15 |         15.0000 |
| 07:45:00 | xh458   |   30 |            45 |         20.0000 |
| 08:00:00 | xh458   |   25 |            70 |         27.5000 |
+----------+---------+------+---------------+-----------------+
```

For the `running_average` column, there is no frame row preceding the first one or following the last. In these cases, `AVG()` computes the average of the rows that are available.

Aggregate functions used as window functions operate on rows in the current row frame, as do these nonaggregate window functions:

```
FIRST_VALUE()
LAST_VALUE()
NTH_VALUE()
```

Standard SQL specifies that window functions that operate on the entire partition should have no frame clause. MySQL permits a frame clause for such functions but ignores it. These functions use the entire partition even if a frame is specified:

```
CUME_DIST()
DENSE_RANK()
LAG()
LEAD()
NTILE()
PERCENT_RANK()
RANK()
ROW_NUMBER()
```

The frame clause, if given, has this syntax:

```
frame_clause:
    frame_units frame_extent

frame_units:
    {ROWS | RANGE}
```

In the absence of a frame clause, the default frame depends on whether an `ORDER BY` clause is present, as described later in this section.

The *`frame_units`* value indicates the type of relationship between the current row and frame rows:

- `ROWS`: The frame is defined by beginning and ending row positions. Offsets are differences in row numbers from the current row number.
- `RANGE`: The frame is defined by rows within a value range. Offsets are differences in row values from the current row value.

The *`frame_extent`* value indicates the start and end points of the frame. You can specify just the start of the frame (in which case the current row is implicitly the end) or use `BETWEEN` to specify both frame endpoints:

```
frame_extent:
    {frame_start | frame_between}

frame_between:
    BETWEEN frame_start AND frame_end

frame_start, frame_end: {
    CURRENT ROW
  | UNBOUNDED PRECEDING
  | UNBOUNDED FOLLOWING
  | expr PRECEDING
  | expr FOLLOWING
}
```

With `BETWEEN` syntax, *`frame_start`* must not occur later than *`frame_end`*.

The permitted *`frame_start`* and *`frame_end`* values have these meanings:

- `CURRENT ROW`: For `ROWS`, the bound is the current row. For `RANGE`, the bound is the peers of the current row.
- `UNBOUNDED PRECEDING`: The bound is the first partition row.
- `UNBOUNDED FOLLOWING`: The bound is the last partition row.
- *`expr`* `PRECEDING`: For `ROWS`, the bound is *`expr`* rows before the current row. For `RANGE`, the bound is the rows with values equal to the current row value minus *`expr`*; if the current row value is `NULL`, the bound is the peers of the row.

  For *`expr`* `PRECEDING` (and *`expr`* `FOLLOWING`), *`expr`* can be a `?` parameter marker (for use in a prepared statement), a nonnegative numeric literal, or a temporal interval of the form `INTERVAL` *`val unit`*. For `INTERVAL` expressions, *`val`* specifies nonnegative interval value, and *`unit`* is a keyword indicating the units in which the value should be interpreted. (For details about the permitted *`units`* specifiers, see the description of the `DATE_ADD()` function in Section 12.7, “Date and Time Functions”.)

  `RANGE` on a numeric or temporal *`expr`* requires `ORDER BY` on a numeric or temporal expression, respectively.

  Examples of valid *`expr`* `PRECEDING` and *`expr`* `FOLLOWING` indicators:

```
10 PRECEDING
INTERVAL 5 DAY PRECEDING
5 FOLLOWING
INTERVAL '2:30' MINUTE_SECOND FOLLOWING
```

- *expr* `FOLLOWING`: For `ROWS`, the bound is *expr* rows after the current row. For `RANGE`, the bound is the rows with values equal to the current row value plus *expr*; if the current row value is `NULL`, the bound is the peers of the row.

  For permitted values of *expr*, see the description of *expr* `PRECEDING`.

The following query demonstrates `FIRST_VALUE()`, `LAST_VALUE()`, and two instances of `NTH_VALUE()`:

```
mysql> SELECT
         time, subject, val,
         FIRST_VALUE(val)  OVER w AS 'first',
         LAST_VALUE(val)   OVER w AS 'last',
         NTH_VALUE(val, 2) OVER w AS 'second',
         NTH_VALUE(val, 4) OVER w AS 'fourth'
       FROM observations
       WINDOW w AS (PARTITION BY subject ORDER BY time
                    ROWS UNBOUNDED PRECEDING);
+----------+---------+------+-------+------+--------+--------+
| time     | subject | val  | first | last | second | fourth |
+----------+---------+------+-------+------+--------+--------+
| 07:00:00 | st113   |   10 |    10 |   10 |   NULL |   NULL |
| 07:15:00 | st113   |    9 |    10 |    9 |      9 |   NULL |
| 07:30:00 | st113   |   25 |    10 |   25 |      9 |   NULL |
| 07:45:00 | st113   |   20 |    10 |   20 |      9 |     20 |
| 07:00:00 | xh458   |    0 |     0 |    0 |   NULL |   NULL |
| 07:15:00 | xh458   |   10 |     0 |   10 |     10 |   NULL |
| 07:30:00 | xh458   |    5 |     0 |    5 |     10 |   NULL |
| 07:45:00 | xh458   |   30 |     0 |   30 |     10 |     30 |
| 08:00:00 | xh458   |   25 |     0 |   25 |     10 |     30 |
+----------+---------+------+-------+------+--------+--------+
```

Each function uses the rows in the current frame, which, per the window definition shown, extends from the first partition row to the current row. For the `NTH_VALUE()` calls, the current frame does not always include the requested row; in such cases, the return value is `NULL`.

In the absence of a frame clause, the default frame depends on whether an `ORDER BY` clause is present:

- With `ORDER BY`: The default frame includes rows from the partition start through the current row, including all peers of the current row (rows equal to the current row according to the `ORDER BY` clause). The default is equivalent to this frame specification:

  ```
  RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
  ```

- Without `ORDER BY`: The default frame includes all partition rows (because, without `ORDER BY`, all partition rows are peers). The default is equivalent to this frame specification:

  ```
  RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
  ```

Because the default frame differs depending on presence or absence of `ORDER BY`, adding `ORDER BY` to a query to get deterministic results may change the results. (For example, the values produced by `SUM()` might change.) To obtain the same results but ordered per `ORDER BY`, provide an explicit frame specification to be used regardless of whether `ORDER BY` is present.

The meaning of a frame specification can be nonobvious when the current row value is NULL. Assuming that to be the case, these examples illustrate how various frame specifications apply:

- ORDER BY X ASC RANGE BETWEEN 10 FOLLOWING AND 15 FOLLOWING

  The frame starts at NULL and stops at NULL, thus includes only rows with value NULL.

- ORDER BY X ASC RANGE BETWEEN 10 FOLLOWING AND UNBOUNDED FOLLOWING

  The frame starts at NULL and stops at the end of the partition. Because an ASC sort puts NULL values first, the frame is the entire partition.

- ORDER BY X DESC RANGE BETWEEN 10 FOLLOWING AND UNBOUNDED FOLLOWING

  The frame starts at NULL and stops at the end of the partition. Because a DESC sort puts NULL values last, the frame is only the NULL values.

- ORDER BY X ASC RANGE BETWEEN 10 PRECEDING AND UNBOUNDED FOLLOWING

  The frame starts at NULL and stops at the end of the partition. Because an ASC sort puts NULL values first, the frame is the entire partition.

- ORDER BY X ASC RANGE BETWEEN 10 PRECEDING AND 10 FOLLOWING

  The frame starts at NULL and stops at NULL, thus includes only rows with value NULL.

- ORDER BY X ASC RANGE BETWEEN 10 PRECEDING AND 1 PRECEDING

  The frame starts at NULL and stops at NULL, thus includes only rows with value NULL.

- ORDER BY X ASC RANGE BETWEEN UNBOUNDED PRECEDING AND 10 FOLLOWING

  The frame starts at the beginning of the partition and stops at rows with value NULL. Because an ASC sort puts NULL values first, the frame is only the NULL values.

12.21.4 Named Windows

Windows can be defined and given names by which to refer to them in OVER clauses. To do this, use a WINDOW clause. If present in a query, the WINDOW clause falls between the positions of the HAVING and ORDER BY clauses, and has this syntax:

```
WINDOW window_name AS (window_spec)
    [, window_name AS (window_spec)] ...
```

For each window definition, *window_name* is the window name, and *window_spec* is the same type of window specification as given between the parentheses of an OVER clause, as described in Section 12.21.2, “Window Function Concepts and Syntax”:

```
window_spec:
    [window_name] [partition_clause] [order_clause] [frame_clause]
```

A WINDOW clause is useful for queries in which multiple OVER clauses would otherwise define the same window. Instead, you can define the window once, give it a name, and refer to the name in the OVER clauses. Consider this query, which defines the same window multiple times:

```
SELECT
  val,
```

```
  ROW_NUMBER() OVER (ORDER BY val) AS 'row_number',
  RANK()       OVER (ORDER BY val) AS 'rank',
  DENSE_RANK() OVER (ORDER BY val) AS 'dense_rank'
FROM numbers;
```

The query can be written more simply by using `WINDOW` to define the window once and referring to the window by name in the `OVER` clauses:

```
SELECT
  val,
  ROW_NUMBER() OVER w AS 'row_number',
  RANK()       OVER w AS 'rank',
  DENSE_RANK() OVER w AS 'dense_rank'
FROM numbers
WINDOW w AS (ORDER BY val);
```

A named window also makes it easier to experiment with the window definition to see the effect on query results. You need only modify the window definition in the `WINDOW` clause, rather than multiple `OVER` clause definitions.

If an `OVER` clause uses `OVER (window_name ...)` rather than `OVER window_name`, the named window can be modified by the addition of other clauses. For example, this query defines a window that includes partitioning, and uses `ORDER BY` in the `OVER` clauses to modify the window in different ways:

```
SELECT
  DISTINCT year, country,
  FIRST_VALUE(year) OVER (w ORDER BY year ASC) AS first,
  FIRST_VALUE(year) OVER (w ORDER BY year DESC) AS last
FROM sales
WINDOW w AS (PARTITION BY country);
```

An `OVER` clause can only add properties to a named window, not modify them. If the named window definition includes a partitioning, ordering, or framing property, the `OVER` clause that refers to the window name cannot also include the same kind of property or an error occurs:

- This construct is permitted because the window definition and the referring `OVER` clause do not contain the same kind of properties:

  ```
  OVER (w ORDER BY country)
  ... WINDOW w AS (PARTITION BY country)
  ```

- This construct is not permitted because the `OVER` clause specifies `PARTITION BY` for a named window that already has `PARTITION BY`:

  ```
  OVER (w PARTITION BY year)
  ... WINDOW w AS (PARTITION BY country)
  ```

The definition of a named window can itself begin with a *`window_name`*. In such cases, forward and backward references are permitted, but not cycles:

- This is permitted; it contains forward and backward references but no cycles:

  ```
  WINDOW w1 AS (w2), w2 AS (), w3 AS (w1)
  ```

- This is not permitted because it contains a cycle:

  ```
  WINDOW w1 AS (w2), w2 AS (w3), w3 AS (w1)
  ```

12.21.5 Window Function Restrictions

The SQL standard imposes a constraint on window functions that they cannot be used in `UPDATE` or `DELETE` statements to update rows. Using such functions in a subquery of these statements (to select rows) is permitted.

MySQL does not support these window function features:

- `DISTINCT` syntax for aggregate window functions.
- Nested window functions.
- Dynamic frame endpoints that depend on the value of the current row.

The parser recognizes these window constructs which nevertheless are not supported:

- The `GROUPS` frame units specifier is parsed, but produces an error. Only `ROWS` and `RANGE` are supported.
- The `EXCLUDE` clause for frame specification is parsed, but produces an error.
- `IGNORE NULLS` is parsed, but produces an error. Only `RESPECT NULLS` is supported.
- `FROM LAST` is parsed, but produces an error. Only `FROM FIRST` is supported.

12.22 Performance Schema Functions

As of MySQL 8.0.16, MySQL includes built-in SQL functions that format or retrieve Performance Schema data, and that may be used as equivalents for the corresponding `sys` schema stored functions. The built-in functions can be invoked in any schema and require no qualifier, unlike the `sys` functions, which require either a `sys.` schema qualifier or that `sys` be the current schema.

Table 12.28 Performance Schema Functions

Name	Description
`FORMAT_BYTES()`	Convert byte count to value with units
`FORMAT_PICO_TIME()`	Convert time in picoseconds to value with units
`PS_CURRENT_THREAD_ID()`	Performance Schema thread ID for current thread
`PS_THREAD_ID()`	Performance Schema thread ID for given thread

The built-in functions supersede the corresponding `sys` functions, which are deprecated and will be removed in a future MySQL version. Applications that use the `sys` functions should be adjusted to use the built-in functions instead, keeping in mind some minor differences between the `sys` functions and the built-in functions. For details about these differences, see the function descriptions in this section.

- `FORMAT_BYTES(`*`count`*`)`

  Given a numeric byte count, converts it to human-readable format and returns a string consisting of a value and a units indicator. The string contains the number of bytes rounded to 2 decimal places and a minimum of 3 significant digits. Numbers less than 1024 bytes are represented as whole numbers and are not rounded.

  The units indicator depends on the size of the byte-count argument as shown in the following table.

Argument Value	Result Units	Result Units Indicator
Up to 1023	bytes	bytes

Argument Value	Result Units	Result Units Indicator
Up to $1024^2 - 1$	kibibytes	KiB
Up to $1024^3 - 1$	mebibytes	MiB
Up to $1024^4 - 1$	gibibytes	GiB
Up to $1024^5 - 1$	tebibytes	TiB
Up to $1024^6 - 1$	pebibytes	PiB
1024^6 and up	exbibytes	EiB

```
mysql> SELECT FORMAT_BYTES(512), FORMAT_BYTES(18446644073709551615);
+------------------+-----------------------------------+
| FORMAT_BYTES(512) | FORMAT_BYTES(18446644073709551615) |
+------------------+-----------------------------------+
|  512 bytes        | 16.00 EiB                          |
+------------------+-----------------------------------+
```

`FORMAT_BYTES()` was added in MySQL 8.0.16. It may be used instead of the `sys` schema `format_bytes()` function, keeping in mind this difference:

- `FORMAT_BYTES()` uses the `EiB` units indicator. `sys.format_bytes()` does not.

- `FORMAT_PICO_TIME(time_val)`

  Given a numeric Performance Schema latency or wait time in picoseconds, converts it to human-readable format and returns a string consisting of a value and a units indicator. The string contains the decimal time rounded to 2 decimal places and a minimum of 3 significant digits. Times under 1 nanosecond are represented as whole numbers and are not rounded.

  The units indicator depends on the size of the time-value argument as shown in the following table.

Argument Value	Result Units	Result Units Indicator
Up to $10^3 - 1$	picoseconds	ps
Up to $10^6 - 1$	nanoseconds	ns
Up to $10^9 - 1$	microseconds	us
Up to $10^{12} - 1$	milliseconds	ms
Up to $60 \times 10^{12} - 1$	seconds	s
Up to $3.6 \times 10^{15} - 1$	minutes	min
Up to $8.64 \times 10^{16} - 1$	hours	h
8.64×10^{16} and up	days	d

```
mysql> SELECT FORMAT_PICO_TIME(3501), FORMAT_PICO_TIME(188732396662000);
+-----------------------+---------------------------------+
| FORMAT_PICO_TIME(3501) | FORMAT_PICO_TIME(188732396662000) |
+-----------------------+---------------------------------+
| 3.50 ns               | 3.15 min                        |
+-----------------------+---------------------------------+
```

`FORMAT_PICO_TIME()` was added in MySQL 8.0.16. It may be used instead of the `sys` schema `format_time()` function, keeping in mind these differences:

  - To indicate minutes, `sys.format_time()` uses the `m` units indicator, whereas `FORMAT_PICO_TIME()` uses `min`.
  - `sys.format_time()` uses the `w` (weeks) units indicator. `FORMAT_PICO_TIME()` does not.

- `PS_CURRENT_THREAD_ID()`

  Returns a `BIGINT UNSIGNED` value representing the Performance Schema thread ID assigned to the current connection.

  The thread ID return value is a value of the type given in the `THREAD_ID` column of Performance Schema tables.

  Performance Schema configuration affects `PS_CURRENT_THREAD_ID()` the same way as for `PS_THREAD_ID()`. For details, see the description of that function.

```
mysql> SELECT PS_CURRENT_THREAD_ID();
+-----------------------+
| PS_CURRENT_THREAD_ID() |
+-----------------------+
|                    52 |
+-----------------------+
mysql> SELECT PS_THREAD_ID(CONNECTION_ID());
+------------------------------+
| PS_THREAD_ID(CONNECTION_ID()) |
+------------------------------+
|                           52 |
+------------------------------+
```

  `PS_CURRENT_THREAD_ID()` was added in MySQL 8.0.16. It may be used as a shortcut for invoking the `sys` schema `ps_thread_id()` function with an argument of `NULL` or `CONNECTION_ID()`.

- `PS_THREAD_ID(`*`connection_id`*`)`

  Given a connection ID, returns a `BIGINT UNSIGNED` value representing the Performance Schema thread ID assigned to the connection ID, or `NULL` if no thread ID exists for the connection ID. The latter can occur for threads that are not instrumented.

  The connection ID argument is a value of the type given in the `PROCESSLIST_ID` column of the Performance Schema `threads` table or the `Id` column of `SHOW PROCESSLIST` output.

  The thread ID return value is a value of the type given in the `THREAD_ID` column of Performance Schema tables.

  Performance Schema configuration affects `PS_THREAD_ID()` operation as follows. (These remarks also apply to `PS_CURRENT_THREAD_ID()`.)

  - Disabling the `thread_instrumentation` consumer disables statistics from being collected and aggregated at the thread level, but has no effect on `PS_THREAD_ID()`.
  - If `performance_schema_max_thread_instances` is not 0, the Performance Schema allocates memory for thread statistics and assigns an internal ID to each thread for which instance memory is available. If there are threads for which instance memory is not available, `PS_THREAD_ID()` returns `NULL`; in this case, `Performance_schema_thread_instances_lost` will be nonzero.
  - If `performance_schema_max_thread_instances` is 0, the Performance Schema allocates no thread memory and `PS_THREAD_ID()` returns `NULL`.

- If the Performance Schema itself is disabled, `PS_THREAD_ID()` produces an error.

```
mysql> SELECT PS_THREAD_ID(6);
+-----------------+
| PS_THREAD_ID(6) |
+-----------------+
|              45 |
+-----------------+
```

`PS_THREAD_ID()` was added in MySQL 8.0.16. It may be used instead of the `sys` schema `ps_thread_id()` function, keeping in mind this difference:

- With an argument of `NULL`, `sys.ps_thread_id()` returns the thread ID for the current connection, whereas `PS_THREAD_ID()` returns `NULL`. To obtain the current connection thread ID, use `PS_CURRENT_THREAD_ID()` instead.

12.23 Internal Functions

Table 12.29 Internal Functions

Name	Description
`CAN_ACCESS_COLUMN()`	Internal use only
`CAN_ACCESS_DATABASE()`	Internal use only
`CAN_ACCESS_TABLE()`	Internal use only
`CAN_ACCESS_VIEW()`	Internal use only
`GET_DD_COLUMN_PRIVILEGES()`	Internal use only
`GET_DD_CREATE_OPTIONS()`	Internal use only
`GET_DD_INDEX_SUB_PART_LENGTH()`	Internal use only
`INTERNAL_AUTO_INCREMENT()`	Internal use only
`INTERNAL_AVG_ROW_LENGTH()`	Internal use only
`INTERNAL_CHECK_TIME()`	Internal use only
`INTERNAL_CHECKSUM()`	Internal use only
`INTERNAL_DATA_FREE()`	Internal use only
`INTERNAL_DATA_LENGTH()`	Internal use only
`INTERNAL_DD_CHAR_LENGTH()`	Internal use only
`INTERNAL_GET_COMMENT_OR_ERROR()`	Internal use only
`INTERNAL_GET_VIEW_WARNING_OR_ERROR()`	Internal use only
`INTERNAL_INDEX_COLUMN_CARDINALITY()`	Internal use only
`INTERNAL_INDEX_LENGTH()`	Internal use only
`INTERNAL_KEYS_DISABLED()`	Internal use only
`INTERNAL_MAX_DATA_LENGTH()`	Internal use only
`INTERNAL_TABLE_ROWS()`	Internal use only
`INTERNAL_UPDATE_TIME()`	Internal use only

The functions listed in this section are intended only for internal use by the server. Attempts by users to invoke them result in an error.

- CAN_ACCESS_COLUMN(*ARGS*)
- CAN_ACCESS_DATABASE(*ARGS*)
- CAN_ACCESS_TABLE(*ARGS*)
- CAN_ACCESS_VIEW(*ARGS*)
- GET_DD_COLUMN_PRIVILEGES(*ARGS*)
- GET_DD_CREATE_OPTIONS(*ARGS*)
- GET_DD_INDEX_SUB_PART_LENGTH(*ARGS*)
- INTERNAL_AUTO_INCREMENT(*ARGS*)
- INTERNAL_AVG_ROW_LENGTH(*ARGS*)
- INTERNAL_CHECK_TIME(*ARGS*)
- INTERNAL_CHECKSUM(*ARGS*)
- INTERNAL_DATA_FREE(*ARGS*)
- INTERNAL_DATA_LENGTH(*ARGS*)
- INTERNAL_DD_CHAR_LENGTH(*ARGS*)
- INTERNAL_GET_COMMENT_OR_ERROR(*ARGS*)
- INTERNAL_GET_VIEW_WARNING_OR_ERROR(*ARGS*)
- INTERNAL_INDEX_COLUMN_CARDINALITY(*ARGS*)
- INTERNAL_INDEX_LENGTH(*ARGS*)
- INTERNAL_KEYS_DISABLED(*ARGS*)
- INTERNAL_MAX_DATA_LENGTH(*ARGS*)
- INTERNAL_TABLE_ROWS(*ARGS*)
- INTERNAL_UPDATE_TIME(*ARGS*)
- IS_VISIBLE_DD_OBJECT(*ARGS*)

12.24 Miscellaneous Functions

Table 12.30 Miscellaneous Functions

Name	Description
ANY_VALUE()	Suppress ONLY_FULL_GROUP_BY value rejection
BIN_TO_UUID()	Convert binary UUID to string
DEFAULT()	Return the default value for a table column
GROUPING()	Distinguish super-aggregate ROLLUP rows from regular rows
INET_ATON()	Return the numeric value of an IP address
INET_NTOA()	Return the IP address from a numeric value
INET6_ATON()	Return the numeric value of an IPv6 address

Name	Description
INET6_NTOA()	Return the IPv6 address from a numeric value
IS_IPV4()	Whether argument is an IPv4 address
IS_IPV4_COMPAT()	Whether argument is an IPv4-compatible address
IS_IPV4_MAPPED()	Whether argument is an IPv4-mapped address
IS_IPV6()	Whether argument is an IPv6 address
IS_UUID()	Whether argument is a valid UUID
MASTER_POS_WAIT()	Block until the slave has read and applied all updates up to the specified position
NAME_CONST()	Cause the column to have the given name
SLEEP()	Sleep for a number of seconds
UUID()	Return a Universal Unique Identifier (UUID)
UUID_SHORT()	Return an integer-valued universal identifier
UUID_TO_BIN()	Convert string UUID to binary
VALUES()	Define the values to be used during an INSERT

- ANY_VALUE(*arg*)

  This function is useful for GROUP BY queries when the ONLY_FULL_GROUP_BY SQL mode is enabled, for cases when MySQL rejects a query that you know is valid for reasons that MySQL cannot determine. The function return value and type are the same as the return value and type of its argument, but the function result is not checked for the ONLY_FULL_GROUP_BY SQL mode.

  For example, if name is a nonindexed column, the following query fails with ONLY_FULL_GROUP_BY enabled:

```
mysql> SELECT name, address, MAX(age) FROM t GROUP BY name;
ERROR 1055 (42000): Expression #2 of SELECT list is not in GROUP
BY clause and contains nonaggregated column 'mydb.t.address' which
is not functionally dependent on columns in GROUP BY clause; this
is incompatible with sql_mode=only_full_group_by
```

  The failure occurs because address is a nonaggregated column that is neither named among GROUP BY columns nor functionally dependent on them. As a result, the address value for rows within each name group is nondeterministic. There are multiple ways to cause MySQL to accept the query:

  - Alter the table to make name a primary key or a unique NOT NULL column. This enables MySQL to determine that address is functionally dependent on name; that is, address is uniquely determined by name. (This technique is inapplicable if NULL must be permitted as a valid name value.)
  - Use ANY_VALUE() to refer to address:

```
SELECT name, ANY_VALUE(address), MAX(age) FROM t GROUP BY name;
```

    In this case, MySQL ignores the nondeterminism of address values within each name group and accepts the query. This may be useful if you simply do not care which value of a nonaggregated column is chosen for each group. ANY_VALUE() is not an aggregate function, unlike functions such as SUM() or COUNT(). It simply acts to suppress the test for nondeterminism.

- Disable ONLY_FULL_GROUP_BY. This is equivalent to using ANY_VALUE() with ONLY_FULL_GROUP_BY enabled, as described in the previous item.

ANY_VALUE() is also useful if functional dependence exists between columns but MySQL cannot determine it. The following query is valid because age is functionally dependent on the grouping column age-1, but MySQL cannot tell that and rejects the query with ONLY_FULL_GROUP_BY enabled:

```
SELECT age FROM t GROUP BY age-1;
```

To cause MySQL to accept the query, use ANY_VALUE():

```
SELECT ANY_VALUE(age) FROM t GROUP BY age-1;
```

ANY_VALUE() can be used for queries that refer to aggregate functions in the absence of a GROUP BY clause:

```
mysql> SELECT name, MAX(age) FROM t;
ERROR 1140 (42000): In aggregated query without GROUP BY, expression
#1 of SELECT list contains nonaggregated column 'mydb.t.name'; this
is incompatible with sql_mode=only_full_group_by
```

Without GROUP BY, there is a single group and it is nondeterministic which name value to choose for the group. ANY_VALUE() tells MySQL to accept the query:

```
SELECT ANY_VALUE(name), MAX(age) FROM t;
```

It may be that, due to some property of a given data set, you know that a selected nonaggregated column is effectively functionally dependent on a GROUP BY column. For example, an application may enforce uniqueness of one column with respect to another. In this case, using ANY_VALUE() for the effectively functionally dependent column may make sense.

For additional discussion, see Section 12.20.3, “MySQL Handling of GROUP BY”.

- BIN_TO_UUID(*binary_uuid*), BIN_TO_UUID(*binary_uuid*, *swap_flag*)

  BIN_TO_UUID() is the inverse of UUID_TO_BIN(). It converts a binary UUID to a string UUID and returns the result. The binary value should be a UUID as a VARBINARY(16) value. The return value is a utf8 string of five hexadecimal numbers separated by dashes. (For details about this format, see the UUID() function description.) If the UUID argument is NULL, the return value is NULL. If any argument is invalid, an error occurs.

  BIN_TO_UUID() takes one or two arguments:

  - The one-argument form takes a binary UUID value. The UUID value is assumed not to have its time-low and time-high parts swapped. The string result is in the same order as the binary argument.
  - The two-argument form takes a binary UUID value and a swap-flag value:
    - If *swap_flag* is 0, the two-argument form is equivalent to the one-argument form. The string result is in the same order as the binary argument.
    - If *swap_flag* is 1, the UUID value is assumed to have its time-low and time-high parts swapped. These parts are swapped back to their original position in the result value.

For usage examples and information about time-part swapping, see the `UUID_TO_BIN()` function description.

- `DEFAULT(col_name)`

  Returns the default value for a table column. An error results if the column has no default value.

  The use of `DEFAULT(col_name)` to specify the default value for a named column is permitted only for columns that have a literal default value, not for columns that have an expression default value.

```
mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
```

- `FORMAT(X,D)`

  Formats the number *`X`* to a format like `'#,###,###.##'`, rounded to *`D`* decimal places, and returns the result as a string. For details, see Section 12.5, “String Functions and Operators”.

- `GROUPING(expr [, expr] ...)`

  For `GROUP BY` queries that include a `WITH ROLLUP` modifier, the `ROLLUP` operation produces super-aggregate output rows where `NULL` represents the set of all values. The `GROUPING()` function enables you to distinguish `NULL` values for super-aggregate rows from `NULL` values in regular grouped rows.

  `GROUPING()` is permitted only in the select list or `HAVING` clause.

  Each argument to `GROUPING()` must be an expression that exactly matches an expression in the `GROUP BY` clause. The expression cannot be a positional specifier. For each expression, `GROUPING()` produces 1 if the expression value in the current row is a `NULL` representing a super-aggregate value. Otherwise, `GROUPING()` produces 0, indicating that the expression value is a `NULL` for a regular result row or is not `NULL`.

  Suppose that table `t1` contains these rows, where `NULL` indicates something like “other” or “unknown”:

```
mysql> SELECT * FROM t1;
+------+-------+----------+
| name | size  | quantity |
+------+-------+----------+
| ball | small |       10 |
| ball | large |       20 |
| ball | NULL  |        5 |
| hoop | small |       15 |
| hoop | large |        5 |
| hoop | NULL  |        3 |
+------+-------+----------+
```

  A summary of the table without `WITH ROLLUP` looks like this:

```
mysql> SELECT name, size, SUM(quantity) AS quantity
       FROM t1
       GROUP BY name, size;
+------+-------+----------+
| name | size  | quantity |
+------+-------+----------+
| ball | small |       10 |
| ball | large |       20 |
| ball | NULL  |        5 |
| hoop | small |       15 |
| hoop | large |        5 |
```

```
| hoop | NULL  |        3 |
+------+-------+----------+
```

The result contains `NULL` values, but those do not represent super-aggregate rows because the query does not include `WITH ROLLUP`.

Adding `WITH ROLLUP` produces super-aggregate summary rows containing additional `NULL` values. However, without comparing this result to the previous one, it is not easy to see which `NULL` values occur in super-aggregate rows and which occur in regular grouped rows:

```
mysql> SELECT name, size, SUM(quantity) AS quantity
       FROM t1
       GROUP BY name, size WITH ROLLUP;
+------+-------+----------+
| name | size  | quantity |
+------+-------+----------+
| ball | NULL  |        5 |
| ball | large |       20 |
| ball | small |       10 |
| ball | NULL  |       35 |
| hoop | NULL  |        3 |
| hoop | large |        5 |
| hoop | small |       15 |
| hoop | NULL  |       23 |
| NULL | NULL  |       58 |
+------+-------+----------+
```

To distinguish `NULL` values in super-aggregate rows from those in regular grouped rows, use `GROUPING()`, which returns 1 only for super-aggregate `NULL` values:

```
mysql> SELECT
         name, size, SUM(quantity) AS quantity,
         GROUPING(name) AS grp_name,
         GROUPING(size) AS grp_size
       FROM t1
       GROUP BY name, size WITH ROLLUP;
+------+-------+----------+----------+----------+
| name | size  | quantity | grp_name | grp_size |
+------+-------+----------+----------+----------+
| ball | NULL  |        5 |        0 |        0 |
| ball | large |       20 |        0 |        0 |
| ball | small |       10 |        0 |        0 |
| ball | NULL  |       35 |        0 |        1 |
| hoop | NULL  |        3 |        0 |        0 |
| hoop | large |        5 |        0 |        0 |
| hoop | small |       15 |        0 |        0 |
| hoop | NULL  |       23 |        0 |        1 |
| NULL | NULL  |       58 |        1 |        1 |
+------+-------+----------+----------+----------+
```

Common uses for `GROUPING()`:

- Substitute a label for super-aggregate `NULL` values:

```
mysql> SELECT
         IF(GROUPING(name) = 1, 'All items', name) AS name,
         IF(GROUPING(size) = 1, 'All sizes', size) AS size,
         SUM(quantity) AS quantity
       FROM t1
       GROUP BY name, size WITH ROLLUP;
+-----------+-----------+----------+
```

```
| name      | size      | quantity |
+-----------+-----------+----------+
| ball      | NULL      |        5 |
| ball      | large     |       20 |
| ball      | small     |       10 |
| ball      | All sizes |       35 |
| hoop      | NULL      |        3 |
| hoop      | large     |        5 |
| hoop      | small     |       15 |
| hoop      | All sizes |       23 |
| All items | All sizes |       58 |
+-----------+-----------+----------+
```

- Return only super-aggregate lines by filtering out the regular grouped lines:

```
mysql> SELECT name, size, SUM(quantity) AS quantity
       FROM t1
       GROUP BY name, size WITH ROLLUP
       HAVING GROUPING(name) = 1 OR GROUPING(size) = 1;
+------+------+----------+
| name | size | quantity |
+------+------+----------+
| ball | NULL |       35 |
| hoop | NULL |       23 |
| NULL | NULL |       58 |
+------+------+----------+
```

`GROUPING()` permits multiple expression arguments. In this case, the `GROUPING()` return value represents a bitmask combined from the results for each expression, where the lowest-order bit corresponds to the result for the rightmost expression. For example, with three expression arguments, `GROUPING(expr1, expr2, expr3)` is evaluated like this:

```
  result for GROUPING(expr3)
+ result for GROUPING(expr2) << 1
+ result for GROUPING(expr1) << 2
```

The following query shows how `GROUPING()` results for single arguments combine for a multiple-argument call to produce a bitmask value:

```
mysql> SELECT
         name, size, SUM(quantity) AS quantity,
         GROUPING(name) AS grp_name,
         GROUPING(size) AS grp_size,
       GROUPING(name, size) AS grp_all
       FROM t1
       GROUP BY name, size WITH ROLLUP;
+------+-------+----------+----------+----------+---------+
| name | size  | quantity | grp_name | grp_size | grp_all |
+------+-------+----------+----------+----------+---------+
| ball | NULL  |        5 |        0 |        0 |       0 |
| ball | large |       20 |        0 |        0 |       0 |
| ball | small |       10 |        0 |        0 |       0 |
| ball | NULL  |       35 |        0 |        1 |       1 |
| hoop | NULL  |        3 |        0 |        0 |       0 |
| hoop | large |        5 |        0 |        0 |       0 |
| hoop | small |       15 |        0 |        0 |       0 |
| hoop | NULL  |       23 |        0 |        1 |       1 |
| NULL | NULL  |       58 |        1 |        1 |       3 |
+------+-------+----------+----------+----------+---------+
```

With multiple expression arguments, the `GROUPING()` return value is nonzero if any expression represents a super-aggregate value. Multiple-argument `GROUPING()` syntax thus provides a simpler way to write the earlier query that returned only super-aggregate rows, by using a single multiple-argument `GROUPING()` call rather than multiple single-argument calls:

```
mysql> SELECT name, size, SUM(quantity) AS quantity
       FROM t1
       GROUP BY name, size WITH ROLLUP
       HAVING GROUPING(name, size) <> 0;
+------+------+----------+
| name | size | quantity |
+------+------+----------+
| ball | NULL |       35 |
| hoop | NULL |       23 |
| NULL | NULL |       58 |
+------+------+----------+
```

Use of `GROUPING()` is subject to these limitations:

- Do not use subquery `GROUP BY` expressions as `GROUPING()` arguments because matching might fail. For example, matching fails for this query:

  ```
  mysql> SELECT GROUPING((SELECT MAX(name) FROM t1))
         FROM t1
         GROUP BY (SELECT MAX(name) FROM t1) WITH ROLLUP;
  ERROR 3580 (HY000): Argument #1 of GROUPING function is not in GROUP BY
  ```

- `GROUP BY` literal expressions should not be used within a `HAVING` clause as `GROUPING()` arguments. Due to differences between when the optimizer evaluates `GROUP BY` and `HAVING`, matching may succeed but `GROUPING()` evaluation does not produce the expected result. Consider this query:

  ```
  SELECT a AS f1, 'w' AS f2
  FROM t
  GROUP BY f1, f2 WITH ROLLUP
  HAVING GROUPING(f2) = 1;
  ```

  `GROUPING()` is evaluated earlier for the literal constant expression than for the `HAVING` clause as a whole and returns 0. To check whether a query such as this is affected, use `EXPLAIN` and look for `Impossible having` in the `Extra` column.

For more information about `WITH ROLLUP` and `GROUPING()`, see Section 12.20.2, “GROUP BY Modifiers”.

- `INET_ATON(`*`expr`*`)`

Given the dotted-quad representation of an IPv4 network address as a string, returns an integer that represents the numeric value of the address in network byte order (big endian). `INET_ATON()` returns `NULL` if it does not understand its argument.

```
mysql> SELECT INET_ATON('10.0.5.9');
        -> 167773449
```

For this example, the return value is calculated as $10\times256^3 + 0\times256^2 + 5\times256 + 9$.

`INET_ATON()` may or may not return a non-`NULL` result for short-form IP addresses (such as `'127.1'` as a representation of `'127.0.0.1'`). Because of this, `INET_ATON()`a should not be used for such addresses.

Note

To store values generated by `INET_ATON()`, use an `INT UNSIGNED` column rather than `INT`, which is signed. If you use a signed column, values corresponding to IP addresses for which the first octet is greater than 127 cannot be stored correctly. See Section 11.2.6, “Out-of-Range and Overflow Handling”.

- `INET_NTOA(`*`expr`*`)`

  Given a numeric IPv4 network address in network byte order, returns the dotted-quad string representation of the address as a string in the connection character set. `INET_NTOA()` returns `NULL` if it does not understand its argument.

```
mysql> SELECT INET_NTOA(167773449);
        -> '10.0.5.9'
```

- `INET6_ATON(`*`expr`*`)`

  Given an IPv6 or IPv4 network address as a string, returns a binary string that represents the numeric value of the address in network byte order (big endian). Because numeric-format IPv6 addresses require more bytes than the largest integer type, the representation returned by this function has the `VARBINARY` data type: `VARBINARY(16)` for IPv6 addresses and `VARBINARY(4)` for IPv4 addresses. If the argument is not a valid address, `INET6_ATON()` returns `NULL`.

  The following examples use `HEX()` to display the `INET6_ATON()` result in printable form:

```
mysql> SELECT HEX(INET6_ATON('fdfe::5a55:caff:fefa:9089'));
        -> 'FDFE0000000000005A55CAFFFEFA9089'
mysql> SELECT HEX(INET6_ATON('10.0.5.9'));
        -> '0A000509'
```

  `INET6_ATON()` observes several constraints on valid arguments. These are given in the following list along with examples.

  - A trailing zone ID is not permitted, as in `fe80::3%1` or `fe80::3%eth0`.
  - A trailing network mask is not permitted, as in `2001:45f:3:ba::/64` or `198.51.100.0/24`.
  - For values representing IPv4 addresses, only classless addresses are supported. Classful addresses such as `198.51.1` are rejected. A trailing port number is not permitted, as in `198.51.100.2:8080`. Hexadecimal numbers in address components are not permitted, as in `198.0xa0.1.2`. Octal numbers are not supported: `198.51.010.1` is treated as `198.51.10.1`, not `198.51.8.1`. These IPv4 constraints also apply to IPv6 addresses that have IPv4 address parts, such as IPv4-compatible or IPv4-mapped addresses.

  To convert an IPv4 address *`expr`* represented in numeric form as an `INT` value to an IPv6 address represented in numeric form as a `VARBINARY` value, use this expression:

```
INET6_ATON(INET_NTOA(expr))
```

  For example:

```
mysql> SELECT HEX(INET6_ATON(INET_NTOA(167773449)));
       -> '0A000509'
```

- `INET6_NTOA(expr)`

  Given an IPv6 or IPv4 network address represented in numeric form as a binary string, returns the string representation of the address as a string in the connection character set. If the argument is not a valid address, `INET6_NTOA()` returns `NULL`.

  `INET6_NTOA()` has these properties:

  - It does not use operating system functions to perform conversions, thus the output string is platform independent.
  - The return string has a maximum length of 39 (4 x 8 + 7). Given this statement:

    ```
    CREATE TABLE t AS SELECT INET6_NTOA(expr) AS c1;
    ```

    The resulting table would have this definition:

    ```
    CREATE TABLE t (c1 VARCHAR(39) CHARACTER SET utf8 DEFAULT NULL);
    ```

  - The return string uses lowercase letters for IPv6 addresses.

```
mysql> SELECT INET6_NTOA(INET6_ATON('fdfe::5a55:caff:fefa:9089'));
       -> 'fdfe::5a55:caff:fefa:9089'
mysql> SELECT INET6_NTOA(INET6_ATON('10.0.5.9'));
       -> '10.0.5.9'

mysql> SELECT INET6_NTOA(UNHEX('FDFE0000000000005A55CAFFFEFA9089'));
       -> 'fdfe::5a55:caff:fefa:9089'
mysql> SELECT INET6_NTOA(UNHEX('0A000509'));
       -> '10.0.5.9'
```

- `IS_IPV4(expr)`

  Returns 1 if the argument is a valid IPv4 address specified as a string, 0 otherwise.

```
mysql> SELECT IS_IPV4('10.0.5.9'), IS_IPV4('10.0.5.256');
       -> 1, 0
```

  For a given argument, if `IS_IPV4()` returns 1, `INET_ATON()` (and `INET6_ATON()`) will return non-`NULL`. The converse statement is not true: In some cases, `INET_ATON()` returns non-`NULL` when `IS_IPV4()` returns 0.

  As implied by the preceding remarks, `IS_IPV4()` is more strict than `INET_ATON()` about what constitutes a valid IPv4 address, so it may be useful for applications that need to perform strong checks against invalid values. Alternatively, use `INET6_ATON()` to convert IPv4 addresses to internal form and check for a `NULL` result (which indicates an invalid address). `INET6_ATON()` is equally strong as `IS_IPV4()` about checking IPv4 addresses.

- `IS_IPV4_COMPAT(expr)`

This function takes an IPv6 address represented in numeric form as a binary string, as returned by `INET6_ATON()`. It returns 1 if the argument is a valid IPv4-compatible IPv6 address, 0 otherwise. IPv4-compatible addresses have the form `::ipv4_address`.

```
mysql> SELECT IS_IPV4_COMPAT(INET6_ATON('::10.0.5.9'));
        -> 1
mysql> SELECT IS_IPV4_COMPAT(INET6_ATON('::ffff:10.0.5.9'));
        -> 0
```

The IPv4 part of an IPv4-compatible address can also be represented using hexadecimal notation. For example, `198.51.100.1` has this raw hexadecimal value:

```
mysql> SELECT HEX(INET6_ATON('198.51.100.1'));
        -> 'C6336401'
```

Expressed in IPv4-compatible form, `::198.51.100.1` is equivalent to `::c0a8:0001` or (without leading zeros) `::c0a8:1`

```
mysql> SELECT
    ->   IS_IPV4_COMPAT(INET6_ATON('::198.51.100.1')),
    ->   IS_IPV4_COMPAT(INET6_ATON('::c0a8:0001')),
    ->   IS_IPV4_COMPAT(INET6_ATON('::c0a8:1'));
        -> 1, 1, 1
```

- `IS_IPV4_MAPPED(expr)`

  This function takes an IPv6 address represented in numeric form as a binary string, as returned by `INET6_ATON()`. It returns 1 if the argument is a valid IPv4-mapped IPv6 address, 0 otherwise. IPv4-mapped addresses have the form `::ffff:ipv4_address`.

  ```
  mysql> SELECT IS_IPV4_MAPPED(INET6_ATON('::10.0.5.9'));
          -> 0
  mysql> SELECT IS_IPV4_MAPPED(INET6_ATON('::ffff:10.0.5.9'));
          -> 1
  ```

  As with `IS_IPV4_COMPAT()` the IPv4 part of an IPv4-mapped address can also be represented using hexadecimal notation:

  ```
  mysql> SELECT
      ->   IS_IPV4_MAPPED(INET6_ATON('::ffff:198.51.100.1')),
      ->   IS_IPV4_MAPPED(INET6_ATON('::ffff:c0a8:0001')),
      ->   IS_IPV4_MAPPED(INET6_ATON('::ffff:c0a8:1'));
          -> 1, 1, 1
  ```

- `IS_IPV6(expr)`

  Returns 1 if the argument is a valid IPv6 address specified as a string, 0 otherwise. This function does not consider IPv4 addresses to be valid IPv6 addresses.

  ```
  mysql> SELECT IS_IPV6('10.0.5.9'), IS_IPV6('::1');
          -> 0, 1
  ```

  For a given argument, if `IS_IPV6()` returns 1, `INET6_ATON()` will return non-`NULL`.

- `IS_UUID(string_uuid)`

Returns 1 if the argument is a valid string-format UUID, 0 if the argument is not a valid UUID, and `NULL` if the argument is `NULL`.

"Valid" means that the value is in a format that can be parsed. That is, it has the correct length and contains only the permitted characters (hexadecimal digits in any lettercase and, optionally, dashes and curly braces). This format is most common:

```
aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee
```

These other formats are also permitted:

```
aaaaaaaabbbbccccddddeeeeeeeeeeee
{aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee}
```

For the meanings of fields within the value, see the `UUID()` function description.

```
mysql> SELECT IS_UUID('6ccd780c-baba-1026-9564-5b8c656024db');
+-------------------------------------------------+
| IS_UUID('6ccd780c-baba-1026-9564-5b8c656024db') |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
mysql> SELECT IS_UUID('6CCD780C-BABA-1026-9564-5B8C656024DB');
+-------------------------------------------------+
| IS_UUID('6CCD780C-BABA-1026-9564-5B8C656024DB') |
+-------------------------------------------------+
|                                               1 |
+-------------------------------------------------+
mysql> SELECT IS_UUID('6ccd780cbaba102695645b8c656024db');
+-------------------------------------------+
| IS_UUID('6ccd780cbaba102695645b8c656024db') |
+-------------------------------------------+
|                                         1 |
+-------------------------------------------+
mysql> SELECT IS_UUID('{6ccd780c-baba-1026-9564-5b8c656024db}');
+---------------------------------------------------+
| IS_UUID('{6ccd780c-baba-1026-9564-5b8c656024db}') |
+---------------------------------------------------+
|                                                 1 |
+---------------------------------------------------+
mysql> SELECT IS_UUID('6ccd780c-baba-1026-9564-5b8c6560');
+-------------------------------------------+
| IS_UUID('6ccd780c-baba-1026-9564-5b8c6560') |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+
mysql> SELECT IS_UUID(RAND());
+-----------------+
| IS_UUID(RAND()) |
+-----------------+
|               0 |
+-----------------+
```

- `MASTER_POS_WAIT(log_name,log_pos[,timeout][,channel])`

  This function is useful for control of master/slave synchronization. It blocks until the slave has read and applied all updates up to the specified position in the master log. The return value is the number of log events the slave had to wait for to advance to the specified position. The function returns `NULL` if the slave SQL thread is not started, the slave's master information is not initialized, the arguments are incorrect, or an error occurs. It returns `-1` if the timeout has been exceeded. If the slave SQL thread

stops while `MASTER_POS_WAIT()` is waiting, the function returns `NULL`. If the slave is past the specified position, the function returns immediately.

On a multithreaded slave, the function waits until expiry of the limit set by the `slave_checkpoint_group` or `slave_checkpoint_period` system variable, when the checkpoint operation is called to update the status of the slave. Depending on the setting for the system variables, the function might therefore return some time after the specified position was reached.

If a *`timeout`* value is specified, `MASTER_POS_WAIT()` stops waiting when *`timeout`* seconds have elapsed. *`timeout`* must be greater than 0; a zero or negative *`timeout`* means no timeout.

The optional *`channel`* value enables you to name which replication channel the function applies to. See Section 17.2.3, “Replication Channels” for more information.

This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `NAME_CONST(`*`name`*`,`*`value`*`)`

  Returns the given value. When used to produce a result set column, `NAME_CONST()` causes the column to have the given name. The arguments should be constants.

```
mysql> SELECT NAME_CONST('myname', 14);
+--------+
| myname |
+--------+
|     14 |
+--------+
```

  This function is for internal use only. The server uses it when writing statements from stored programs that contain references to local program variables, as described in Section 24.7, “Stored Program Binary Logging”. You might see this function in the output from `mysqlbinlog`.

  For your applications, you can obtain exactly the same result as in the example just shown by using simple aliasing, like this:

```
mysql> SELECT 14 AS myname;
+--------+
| myname |
+--------+
|     14 |
+--------+
1 row in set (0.00 sec)
```

  See Section 13.2.10, “SELECT Syntax”, for more information about column aliases.

- `SLEEP(`*`duration`*`)`

  Sleeps (pauses) for the number of seconds given by the *`duration`* argument, then returns 0. The duration may have a fractional part. If the argument is `NULL` or negative, `SLEEP()` produces a warning, or an error in strict SQL mode.

  When sleep returns normally (without interruption), it returns 0:

```
mysql> SELECT SLEEP(1000);
+-------------+
| SLEEP(1000) |
```

```
+-------------+
|           0 |
+-------------+
```

When `SLEEP()` is the only thing invoked by a query that is interrupted, it returns 1 and the query itself returns no error. This is true whether the query is killed or times out:

- This statement is interrupted using `KILL QUERY` from another session:

```
mysql> SELECT SLEEP(1000);
+-------------+
| SLEEP(1000) |
+-------------+
|           1 |
+-------------+
```

- This statement is interrupted by timing out:

```
mysql> SELECT /*+ MAX_EXECUTION_TIME(1) */ SLEEP(1000);
+-------------+
| SLEEP(1000) |
+-------------+
|           1 |
+-------------+
```

When `SLEEP()` is only part of a query that is interrupted, the query returns an error:

- This statement is interrupted using `KILL QUERY` from another session:

```
mysql> SELECT 1 FROM t1 WHERE SLEEP(1000);
ERROR 1317 (70100): Query execution was interrupted
```

- This statement is interrupted by timing out:

```
mysql> SELECT /*+ MAX_EXECUTION_TIME(1000) */ 1 FROM t1 WHERE SLEEP(1000);
ERROR 3024 (HY000): Query execution was interrupted, maximum statement
execution time exceeded
```

This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `UUID()`

Returns a Universal Unique Identifier (UUID) generated according to RFC 4122, “A Universally Unique IDentifier (UUID) URN Namespace” (http://www.ietf.org/rfc/rfc4122.txt).

A UUID is designed as a number that is globally unique in space and time. Two calls to `UUID()` are expected to generate two different values, even if these calls are performed on two separate devices not connected to each other.

Warning

Although `UUID()` values are intended to be unique, they are not necessarily unguessable or unpredictable. If unpredictability is required, UUID values should be generated some other way.

`UUID()` returns a value that conforms to UUID version 1 as described in RFC 4122. The value is a 128-bit number represented as a `utf8` string of five hexadecimal numbers in `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` format:

- The first three numbers are generated from the low, middle, and high parts of a timestamp. The high part also includes the UUID version number.
- The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).
- The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host device has no Ethernet card, or it is unknown how to find the hardware address of an interface on the host operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have *very* low probability.

  The MAC address of an interface is taken into account only on FreeBSD, Linux, and Windows. On other operating systems, MySQL uses a randomly generated 48-bit number.

```
mysql> SELECT UUID();
        -> '6ccd780c-baba-1026-9564-5b8c656024db'
```

To convert between string and binary UUID values, use the `UUID_TO_BIN()` and `BIN_TO_UUID()` functions. To check whether a string is a valid UUID value, use the `IS_UUID()` function.

This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `UUID_SHORT()`

Returns a “short” universal identifier as a 64-bit unsigned integer. Values returned by `UUID_SHORT()` differ from the string-format 128-bit identifiers returned by the `UUID()` function and have different uniqueness properties. The value of `UUID_SHORT()` is guaranteed to be unique if the following conditions hold:

- The `server_id` value of the current server is between 0 and 255 and is unique among your set of master and slave servers
- You do not set back the system time for your server host between `mysqld` restarts
- You invoke `UUID_SHORT()` on average fewer than 16 million times per second between `mysqld` restarts

The `UUID_SHORT()` return value is constructed this way:

```
  (server_id & 255) << 56
+ (server_startup_time_in_seconds << 24)
+ incremented_variable++;
```

```
mysql> SELECT UUID_SHORT();
        -> 92395783831158784
```

Note

`UUID_SHORT()` does not work with statement-based replication.

- `UUID_TO_BIN(`*`string_uuid`*`)`, `UUID_TO_BIN(`*`string_uuid, swap_flag`*`)`

  Converts a string UUID to a binary UUID and returns the result. (The `IS_UUID()` function description lists the permitted string UUID formats.) The return binary UUID is a `VARBINARY(16)` value. If the UUID argument is `NULL`, the return value is `NULL`. If any argument is invalid, an error occurs.

  `UUID_TO_BIN()` takes one or two arguments:

  - The one-argument form takes a string UUID value. The binary result is in the same order as the string argument.
  - The two-argument form takes a string UUID value and a flag value:
    - If *`swap_flag`* is 0, the two-argument form is equivalent to the one-argument form. The binary result is in the same order as the string argument.
    - If *`swap_flag`* is 1, the format of the return value differs: The time-low and time-high parts (the first and third groups of hexadecimal digits, respectively) are swapped. This moves the more rapidly varying part to the right and can improve indexing efficiency if the result is stored in an indexed column.

  Time-part swapping assumes the use of UUID version 1 values, such as are generated by the `UUID()` function. For UUID values produced by other means that do not follow version 1 format, time-part swapping provides no benefit. For details about version 1 format, see the `UUID()` function description.

  Suppose that you have the following string UUID value:

```
mysql> SET @uuid = '6ccd780c-baba-1026-9564-5b8c656024db';
```

  To convert the string UUID to binary with or without time-part swapping, use `UUID_TO_BIN()`:

```
mysql> SELECT HEX(UUID_TO_BIN(@uuid));
+----------------------------------+
| HEX(UUID_TO_BIN(@uuid))          |
+----------------------------------+
| 6CCD780CBABA102695645B8C656024DB |
+----------------------------------+
mysql> SELECT HEX(UUID_TO_BIN(@uuid, 0));
+----------------------------------+
| HEX(UUID_TO_BIN(@uuid, 0))       |
+----------------------------------+
| 6CCD780CBABA102695645B8C656024DB |
+----------------------------------+
mysql> SELECT HEX(UUID_TO_BIN(@uuid, 1));
+----------------------------------+
| HEX(UUID_TO_BIN(@uuid, 1))       |
+----------------------------------+
| 1026BABA6CCD780C95645B8C656024DB |
+----------------------------------+
```

  To convert a binary UUID returned by `UUID_TO_BIN()` to a string UUID, use `BIN_TO_UUID()`. If you produce a binary UUID by calling `UUID_TO_BIN()` with a second argument of 1 to swap time parts, you

should also pass a second argument of 1 to `BIN_TO_UUID()` to unswap the time parts when converting the binary UUID back to a string UUID:

```
mysql> SELECT BIN_TO_UUID(UUID_TO_BIN(@uuid));
+--------------------------------------+
| BIN_TO_UUID(UUID_TO_BIN(@uuid))      |
+--------------------------------------+
| 6ccd780c-baba-1026-9564-5b8c656024db |
+--------------------------------------+
mysql> SELECT BIN_TO_UUID(UUID_TO_BIN(@uuid,0),0);
+--------------------------------------+
| BIN_TO_UUID(UUID_TO_BIN(@uuid,0),0)  |
+--------------------------------------+
| 6ccd780c-baba-1026-9564-5b8c656024db |
+--------------------------------------+
mysql> SELECT BIN_TO_UUID(UUID_TO_BIN(@uuid,1),1);
+--------------------------------------+
| BIN_TO_UUID(UUID_TO_BIN(@uuid,1),1)  |
+--------------------------------------+
| 6ccd780c-baba-1026-9564-5b8c656024db |
+--------------------------------------+
```

If the use of time-part swapping is not the same for the conversion in both directions, the original UUID will not be recovered properly:

```
mysql> SELECT BIN_TO_UUID(UUID_TO_BIN(@uuid,0),1);
+--------------------------------------+
| BIN_TO_UUID(UUID_TO_BIN(@uuid,0),1)  |
+--------------------------------------+
| baba1026-780c-6ccd-9564-5b8c656024db |
+--------------------------------------+
mysql> SELECT BIN_TO_UUID(UUID_TO_BIN(@uuid,1),0);
+--------------------------------------+
| BIN_TO_UUID(UUID_TO_BIN(@uuid,1),0)  |
+--------------------------------------+
| 1026baba-6ccd-780c-9564-5b8c656024db |
+--------------------------------------+
```

- `VALUES(col_name)`

  In an `INSERT ... ON DUPLICATE KEY UPDATE` statement, you can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of `col_name` that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` function is meaningful only in the `ON DUPLICATE KEY UPDATE` clause of `INSERT` statements and returns `NULL` otherwise. See Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”.

  ```
  mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
      -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
  ```

12.25 Precision Math

MySQL provides support for precision math: numeric value handling that results in extremely accurate results and a high degree control over invalid values. Precision math is based on these two features:

- SQL modes that control how strict the server is about accepting or rejecting invalid data.

- The MySQL library for fixed-point arithmetic.

These features have several implications for numeric operations and provide a high degree of compliance with standard SQL:

- **Precise calculations**: For exact-value numbers, calculations do not introduce floating-point errors. Instead, exact precision is used. For example, MySQL treats a number such as `.0001` as an exact value rather than as an approximation, and summing it 10,000 times produces a result of exactly `1`, not a value that is merely “close” to 1.
- **Well-defined rounding behavior**: For exact-value numbers, the result of `ROUND()` depends on its argument, not on environmental factors such as how the underlying C library works.
- **Platform independence**: Operations on exact numeric values are the same across different platforms such as Windows and Unix.
- **Control over handling of invalid values**: Overflow and division by zero are detectable and can be treated as errors. For example, you can treat a value that is too large for a column as an error rather than having the value truncated to lie within the range of the column's data type. Similarly, you can treat division by zero as an error rather than as an operation that produces a result of `NULL`. The choice of which approach to take is determined by the setting of the server SQL mode.

The following discussion covers several aspects of how precision math works, including possible incompatibilities with older applications. At the end, some examples are given that demonstrate how MySQL handles numeric operations precisely. For information about controlling the SQL mode, see Section 5.1.11, “Server SQL Modes”.

12.25.1 Types of Numeric Values

The scope of precision math for exact-value operations includes the exact-value data types (integer and `DECIMAL` types) and exact-value numeric literals. Approximate-value data types and numeric literals are handled as floating-point numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar may be treated differently. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types.

The `FLOAT` and `DOUBLE` data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with `FLOAT` or `DOUBLE` are `DOUBLE PRECISION` and `REAL`.

12.25.2 DECIMAL Data Type Characteristics

This section discusses the characteristics of the `DECIMAL` data type (and its synonyms), with particular regard to the following topics:

- Maximum number of digits
- Storage format

- Storage requirements
- The nonstandard MySQL extension to the upper range of `DECIMAL` columns

The declaration syntax for a `DECIMAL` column is `DECIMAL(M,D)`. The ranges of values for the arguments are as follows:

- *`M`* is the maximum number of digits (the precision). It has a range of 1 to 65.
- *`D`* is the number of digits to the right of the decimal point (the scale). It has a range of 0 to 30 and must be no larger than *`M`*.

If *`D`* is omitted, the default is 0. If *`M`* is omitted, the default is 10.

The maximum value of 65 for *`M`* means that calculations on `DECIMAL` values are accurate up to 65 digits. This limit of 65 digits of precision also applies to exact-value numeric literals, so the maximum range of such literals differs from before.

Values for `DECIMAL` columns are stored using a binary format that packs nine decimal digits into 4 bytes. The storage requirements for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires 4 bytes, and any remaining digits left over require some fraction of 4 bytes. The storage required for remaining digits is given by the following table.

Leftover Digits	Number of Bytes
0	0
1–2	1
3–4	2
5–6	3
7–9	4

For example, a `DECIMAL(18,9)` column has nine digits on either side of the decimal point, so the integer part and the fractional part each require 4 bytes. A `DECIMAL(20,6)` column has fourteen integer digits and six fractional digits. The integer digits require four bytes for nine of the digits and 3 bytes for the remaining five digits. The six fractional digits require 3 bytes.

`DECIMAL` columns do not store a leading `+` character or `-` character or leading `0` digits. If you insert `+0003.1` into a `DECIMAL(5,1)` column, it is stored as `3.1`. For negative numbers, a literal `-` character is not stored.

`DECIMAL` columns do not permit values larger than the range implied by the column definition. For example, a `DECIMAL(3,0)` column supports a range of `-999` to `999`. A `DECIMAL(M,D)` column permits up to *`M`* - *`D`* digits to the left of the decimal point.

The SQL standard requires that the precision of `NUMERIC(M,D)` be *exactly* *`M`* digits. For `DECIMAL(M,D)`, the standard requires a precision of at least *`M`* digits but permits more. In MySQL, `DECIMAL(M,D)` and `NUMERIC(M,D)` are the same, and both have a precision of exactly *`M`* digits.

For a full explanation of the internal format of `DECIMAL` values, see the file `strings/decimal.c` in a MySQL source distribution. The format is explained (with an example) in the `decimal2bin()` function.

12.25.3 Expression Handling

With precision math, exact-value numbers are used as given whenever possible. For example, numbers in comparisons are used exactly as given without a change in value. In strict SQL mode, for `INSERT` into a

column with an exact data type (`DECIMAL` or integer), a number is inserted with its exact value if it is within the column range. When retrieved, the value should be the same as what was inserted. (If strict SQL mode is not enabled, truncation for `INSERT` is permissible.)

Handling of a numeric expression depends on what kind of values the expression contains:

- If any approximate values are present, the expression is approximate and is evaluated using floating-point arithmetic.
- If no approximate values are present, the expression contains only exact values. If any exact value contains a fractional part (a value following the decimal point), the expression is evaluated using `DECIMAL` exact arithmetic and has a precision of 65 digits. The term “exact” is subject to the limits of what can be represented in binary. For example, `1.0/3.0` can be approximated in decimal notation as `.333...`, but not written as an exact number, so `(1.0/3.0)*3.0` does not evaluate to exactly `1.0`.
- Otherwise, the expression contains only integer values. The expression is exact and is evaluated using integer arithmetic and has a precision the same as `BIGINT` (64 bits).

If a numeric expression contains any strings, they are converted to double-precision floating-point values and the expression is approximate.

Inserts into numeric columns are affected by the SQL mode, which is controlled by the `sql_mode` system variable. (See Section 5.1.11, “Server SQL Modes”.) The following discussion mentions strict mode (selected by the `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` mode values) and `ERROR_FOR_DIVISION_BY_ZERO`. To turn on all restrictions, you can simply use `TRADITIONAL` mode, which includes both strict mode values and `ERROR_FOR_DIVISION_BY_ZERO`:

```
SET sql_mode='TRADITIONAL';
```

If a number is inserted into an exact type column (`DECIMAL` or integer), it is inserted with its exact value if it is within the column range and precision.

If the value has too many digits in the fractional part, rounding occurs and a note is generated. Rounding is done as described in Section 12.25.4, “Rounding Behavior”. Truncation due to rounding of the fractional part is not an error, even in strict mode.

If the value has too many digits in the integer part, it is too large (out of range) and is handled as follows:

- If strict mode is not enabled, the value is truncated to the nearest legal value and a warning is generated.
- If strict mode is enabled, an overflow error occurs.

Underflow is not detected, so underflow handling is undefined.

For inserts of strings into numeric columns, conversion from string to number is handled as follows if the string has nonnumeric contents:

- A string that does not begin with a number cannot be used as a number and produces an error in strict mode, or a warning otherwise. This includes the empty string.
- A string that begins with a number can be converted, but the trailing nonnumeric portion is truncated. If the truncated portion contains anything other than spaces, this produces an error in strict mode, or a warning otherwise.

By default, division by zero produces a result of `NULL` and no warning. By setting the SQL mode appropriately, division by zero can be restricted.

With the ERROR_FOR_DIVISION_BY_ZERO SQL mode enabled, MySQL handles division by zero differently:

- If strict mode is not enabled, a warning occurs.
- If strict mode is enabled, inserts and updates involving division by zero are prohibited, and an error occurs.

In other words, inserts and updates involving expressions that perform division by zero can be treated as errors, but this requires ERROR_FOR_DIVISION_BY_ZERO in addition to strict mode.

Suppose that we have this statement:

```
INSERT INTO t SET i = 1/0;
```

This is what happens for combinations of strict and ERROR_FOR_DIVISION_BY_ZERO modes.

sql_mode Value	Result
'' (Default)	No warning, no error; i is set to NULL.
strict	No warning, no error; i is set to NULL.
ERROR_FOR_DIVISION_BY_ZERO	Warning, no error; i is set to NULL.
strict,ERROR_FOR_DIVISION_BY_ZERO	Error condition; no row is inserted.

12.25.4 Rounding Behavior

This section discusses precision math rounding for the ROUND() function and for inserts into columns with exact-value types (DECIMAL and integer).

The ROUND() function rounds differently depending on whether its argument is exact or approximate:

- For exact-value numbers, ROUND() uses the “round half up” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative. (In other words, it is rounded toward zero.)
- For approximate-value numbers, the result depends on the C library. On many systems, this means that ROUND() uses the “round to nearest even” rule: A value with a fractional part exactly half way between two integers is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

For inserts into a DECIMAL or integer column, the target is an exact data type, so rounding uses “round half away from zero,” regardless of whether the value to be inserted is exact or approximate:

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 2

mysql> SHOW WARNINGS;
+-------+------+---------------------------------------+
| Level | Code | Message                               |
+-------+------+---------------------------------------+
| Note  | 1265 | Data truncated for column 'd' at row 1 |
| Note  | 1265 | Data truncated for column 'd' at row 2 |
+-------+------+---------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT d FROM t;
+------+
| d    |
+------+
|    3 |
|    3 |
+------+
2 rows in set (0.00 sec)
```

The `SHOW WARNINGS` statement displays the notes that are generated by truncation due to rounding of the fractional part. Such truncation is not an error, even in strict SQL mode (see Section 12.25.3, “Expression Handling”).

12.25.5 Precision Math Examples

This section provides some examples that show precision math query results in MySQL. These examples demonstrate the principles described in Section 12.25.3, “Expression Handling”, and Section 12.25.4, “Rounding Behavior”.

Example 1. Numbers are used with their exact value as given when possible:

```
mysql> SELECT (.1 + .2) = .3;
+----------------+
| (.1 + .2) = .3 |
+----------------+
|              1 |
+----------------+
```

For floating-point values, results are inexact:

```
mysql> SELECT (.1E0 + .2E0) = .3E0;
+----------------------+
| (.1E0 + .2E0) = .3E0 |
+----------------------+
|                    0 |
+----------------------+
```

Another way to see the difference in exact and approximate value handling is to add a small number to a sum many times. Consider the following stored procedure, which adds `.0001` to a variable 1,000 times.

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE d DECIMAL(10,4) DEFAULT 0;
  DECLARE f FLOAT DEFAULT 0;
  WHILE i < 10000 DO
    SET d = d + .0001;
    SET f = f + .0001E0;
```

```
    SET i = i + 1;
  END WHILE;
  SELECT d, f;
END;
```

The sum for both `d` and `f` logically should be 1, but that is true only for the decimal calculation. The floating-point calculation introduces small errors:

```
+--------+------------------+
| d      | f                |
+--------+------------------+
| 1.0000 | 0.99999999999991 |
+--------+------------------+
```

Example 2. Multiplication is performed with the scale required by standard SQL. That is, for two numbers *`X1`* and *`X2`* that have scale *`S1`* and *`S2`*, the scale of the result is *`S1 + S2`*:

```
mysql> SELECT .01 * .01;
+-----------+
| .01 * .01 |
+-----------+
| 0.0001    |
+-----------+
```

Example 3. Rounding behavior for exact-value numbers is well-defined:

Rounding behavior (for example, with the `ROUND()` function) is independent of the implementation of the underlying C library, which means that results are consistent from platform to platform.

- Rounding for exact-value columns (`DECIMAL` and integer) and exact-valued numbers uses the “round half away from zero” rule. A value with a fractional part of .5 or greater is rounded away from zero to the nearest integer, as shown here:

  ```
  mysql> SELECT ROUND(2.5), ROUND(-2.5);
  +------------+-------------+
  | ROUND(2.5) | ROUND(-2.5) |
  +------------+-------------+
  | 3          | -3          |
  +------------+-------------+
  ```

- Rounding for floating-point values uses the C library, which on many systems uses the “round to nearest even” rule. A value with a fractional part exactly half way between two integers is rounded to the nearest even integer:

  ```
  mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
  +--------------+---------------+
  | ROUND(2.5E0) | ROUND(-2.5E0) |
  +--------------+---------------+
  |            2 |            -2 |
  +--------------+---------------+
  ```

Example 4. In strict mode, inserting a value that is out of range for a column causes an error, rather than truncation to a legal value.

When MySQL is not running in strict mode, truncation to a legal value occurs:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
|  127 |
+------+
1 row in set (0.00 sec)
```

However, an error occurs if strict mode is in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 128;
ERROR 1264 (22003): Out of range value adjusted for column 'i' at row 1

mysql> SELECT i FROM t;
Empty set (0.00 sec)
```

Example 5: In strict mode and with `ERROR_FOR_DIVISION_BY_ZERO` set, division by zero causes an error, not a result of `NULL`.

In nonstrict mode, division by zero has a result of `NULL`:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
| NULL |
+------+
1 row in set (0.03 sec)
```

However, division by zero is an error if the proper SQL modes are in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
ERROR 1365 (22012): Division by 0

mysql> SELECT i FROM t;
```

```
Empty set (0.01 sec)
```

Example 6. Exact-value literals are evaluated as exact values.

Approximate-value literals are evaluated using floating point, but exact-value literals are handled as `DECIMAL`:

```
mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> DESCRIBE t;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| a     | decimal(2,1) unsigned | NO   |     | 0.0     |       |
| b     | double               | NO   |     | 0       |       |
+-------+----------------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

Example 7. If the argument to an aggregate function is an exact numeric type, the result is also an exact numeric type, with a scale at least that of the argument.

Consider these statements:

```
mysql> CREATE TABLE t (i INT, d DECIMAL, f FLOAT);
mysql> INSERT INTO t VALUES(1,1,1);
mysql> CREATE TABLE y SELECT AVG(i), AVG(d), AVG(f) FROM t;
```

The result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type:

```
mysql> DESCRIBE y;
+--------+---------------+------+-----+---------+-------+
| Field  | Type          | Null | Key | Default | Extra |
+--------+---------------+------+-----+---------+-------+
| AVG(i) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(d) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(f) | double        | YES  |     | NULL    |       |
+--------+---------------+------+-----+---------+-------+
```

The result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type.

Chapter 13 SQL Statement Syntax

Table of Contents

13.1 Data Definition Statements 2238
13.1.1 Atomic Data Definition Statement Support 2238
13.1.2 ALTER DATABASE Syntax 2244
13.1.3 ALTER EVENT Syntax 2245
13.1.4 ALTER FUNCTION Syntax 2247
13.1.5 ALTER INSTANCE Syntax 2247
13.1.6 ALTER LOGFILE GROUP Syntax 2248
13.1.7 ALTER PROCEDURE Syntax 2249
13.1.8 ALTER SERVER Syntax 2250
13.1.9 ALTER TABLE Syntax 2250
13.1.10 ALTER TABLESPACE Syntax 2274
13.1.11 ALTER VIEW Syntax 2276
13.1.12 CREATE DATABASE Syntax 2276
13.1.13 CREATE EVENT Syntax 2277
13.1.14 CREATE FUNCTION Syntax 2282
13.1.15 CREATE INDEX Syntax 2282
13.1.16 CREATE LOGFILE GROUP Syntax 2297
13.1.17 CREATE PROCEDURE and CREATE FUNCTION Syntax 2299
13.1.18 CREATE SERVER Syntax 2304
13.1.19 CREATE SPATIAL REFERENCE SYSTEM Syntax 2305
13.1.20 CREATE TABLE Syntax 2310
13.1.21 CREATE TABLESPACE Syntax 2358
13.1.22 CREATE TRIGGER Syntax 2365
13.1.23 CREATE VIEW Syntax 2367
13.1.24 DROP DATABASE Syntax 2371
13.1.25 DROP EVENT Syntax 2372
13.1.26 DROP FUNCTION Syntax 2372
13.1.27 DROP INDEX Syntax 2372
13.1.28 DROP LOGFILE GROUP Syntax 2373
13.1.29 DROP PROCEDURE and DROP FUNCTION Syntax 2373
13.1.30 DROP SERVER Syntax 2373
13.1.31 DROP SPATIAL REFERENCE SYSTEM Syntax 2374
13.1.32 DROP TABLE Syntax 2374
13.1.33 DROP TABLESPACE Syntax 2375
13.1.34 DROP TRIGGER Syntax 2377
13.1.35 DROP VIEW Syntax 2377
13.1.36 RENAME TABLE Syntax 2377
13.1.37 TRUNCATE TABLE Syntax 2379
13.2 Data Manipulation Statements 2381
13.2.1 CALL Syntax 2381
13.2.2 DELETE Syntax 2382
13.2.3 DO Syntax 2386
13.2.4 HANDLER Syntax 2387
13.2.5 IMPORT TABLE Syntax 2389
13.2.6 INSERT Syntax 2391
13.2.7 LOAD DATA Syntax 2399
13.2.8 LOAD XML Syntax 2410
13.2.9 REPLACE Syntax 2418

13.2.10 SELECT Syntax 2421
13.2.11 Subquery Syntax 2438
13.2.12 UPDATE Syntax 2455
13.2.13 WITH Syntax (Common Table Expressions) 2458
13.3 Transactional and Locking Statements 2470
13.3.1 START TRANSACTION, COMMIT, and ROLLBACK Syntax 2470
13.3.2 Statements That Cannot Be Rolled Back 2473
13.3.3 Statements That Cause an Implicit Commit 2474
13.3.4 SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax 2475
13.3.5 LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Syntax 2475
13.3.6 LOCK TABLES and UNLOCK TABLES Syntax 2476
13.3.7 SET TRANSACTION Syntax 2482
13.3.8 XA Transactions 2485
13.4 Replication Statements 2490
13.4.1 SQL Statements for Controlling Master Servers 2490
13.4.2 SQL Statements for Controlling Slave Servers 2493
13.4.3 SQL Statements for Controlling Group Replication 2511
13.5 Prepared SQL Statement Syntax 2515
13.5.1 PREPARE Syntax 2519
13.5.2 EXECUTE Syntax 2519
13.5.3 DEALLOCATE PREPARE Syntax 2519
13.6 Compound-Statement Syntax 2520
13.6.1 BEGIN ... END Compound-Statement Syntax 2520
13.6.2 Statement Label Syntax 2520
13.6.3 DECLARE Syntax 2521
13.6.4 Variables in Stored Programs 2522
13.6.5 Flow Control Statements 2523
13.6.6 Cursors 2528
13.6.7 Condition Handling 2530
13.6.8 Restrictions on Condition Handling 2559
13.7 Database Administration Statements 2560
13.7.1 Account Management Statements 2560
13.7.2 Resource Group Management Statements 2604
13.7.3 Table Maintenance Statements 2608
13.7.4 Component, Plugin, and User-Defined Function Statements 2620
13.7.5 CLONE Syntax 2624
13.7.6 SET Syntax 2625
13.7.7 SHOW Syntax 2631
13.7.8 Other Administrative Statements 2689
13.8 Utility Statements 2701
13.8.1 DESCRIBE Syntax 2701
13.8.2 EXPLAIN Syntax 2701
13.8.3 HELP Syntax 2704
13.8.4 USE Syntax 2706

This chapter describes the syntax for the SQL statements supported by MySQL.

13.1 Data Definition Statements

13.1.1 Atomic Data Definition Statement Support

MySQL 8.0 supports atomic Data Definition Language (DDL) statements. This feature is referred to as *atomic DDL*. An atomic DDL statement combines the data dictionary updates, storage engine operations, and binary log writes associated with a DDL operation into a single, atomic transaction. The transaction is

either committed, with applicable changes persisted to the data dictionary, storage engine, and binary log, or is rolled back, even if the server halts during the operation.

Atomic DDL is made possible by the introduction of the MySQL data dictionary in MySQL 8.0. In earlier MySQL versions, metadata was stored in metadata files, nontransactional tables, and storage engine-specific dictionaries, which necessitated intermediate commits. Centralized, transactional metadata storage provided by the MySQL data dictionary removed this barrier, making it possible to restructure DDL statement operations into atomic transactions.

The atomic DDL feature is described under the following topics in this section:

- Supported DDL Statements
- Atomic DDL Characteristics
- Changes in DDL Statement Behavior
- Storage Engine Support
- Viewing DDL Logs

Supported DDL Statements

The atomic DDL feature supports both table and non-table DDL statements. Table-related DDL operations require storage engine support, whereas non-table DDL operations do not. Currently, only the `InnoDB` storage engine supports atomic DDL.

- Supported table DDL statements include `CREATE`, `ALTER`, and `DROP` statements for databases, tablespaces, tables, and indexes, and the `TRUNCATE TABLE` statement.
- Supported non-table DDL statements include:
  - `CREATE` and `DROP` statements, and, if applicable, `ALTER` statements for stored programs, triggers, views, and user-defined functions (UDFs).
  - Account management statements: `CREATE`, `ALTER`, `DROP`, and, if applicable, `RENAME` statements for users and roles, as well as `GRANT` and `REVOKE` statements.

The following statements are not supported by the atomic DDL feature:

- Table-related DDL statements that involve a storage engine other than `InnoDB`.
- `INSTALL PLUGIN` and `UNINSTALL PLUGIN` statements.
- `INSTALL COMPONENT` and `UNINSTALL COMPONENT` statements.
- `CREATE SERVER`, `ALTER SERVER`, and `DROP SERVER` statements.

Atomic DDL Characteristics

The characteristics of atomic DDL statements include the following:

- Metadata updates, binary log writes, and storage engine operations, where applicable, are combined into a single transaction.
- There are no intermediate commits at the SQL layer during the DDL operation.
- Where applicable:

  - The state of data dictionary, routine, event, and UDF caches is consistent with the status of the DDL operation, meaning that caches are updated to reflect whether or not the DDL operation was completed successfully or rolled back.
  - The storage engine methods involved in a DDL operation do not perform intermediate commits, and the storage engine registers itself as part of the DDL transaction.
  - The storage engine supports redo and rollback of DDL operations, which is performed in the *Post-DDL* phase of the DDL operation.
- The visible behaviour of DDL operations is atomic, which changes the behavior of some DDL statements. See Changes in DDL Statement Behavior.

Note

DDL statements, atomic or otherwise, implicitly end any transaction that is active in the current session, as if you had done a `COMMIT` before executing the statement. This means that DDL statements cannot be performed within another transaction, within transaction control statements such as `START TRANSACTION ... COMMIT`, or combined with other statements within the same transaction.

Changes in DDL Statement Behavior

This section describes changes in DDL statement behavior due to the introduction of atomic DDL support.

- `DROP TABLE` operations are fully atomic if all named tables use an atomic DDL-supported storage engine. The statement either drops all tables successfully or is rolled back.

  `DROP TABLE` fails with an error if a named table does not exist, and no changes are made, regardless of the storage engine. This change in behavior is demonstrated in the following example, where the `DROP TABLE` statement fails because a named table does not exist:

```
mysql> CREATE TABLE t1 (c1 INT);
mysql> DROP TABLE t1, t2;
ERROR 1051 (42S02): Unknown table 'test.t2'
mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
+----------------+
```

  Prior to the introduction of atomic DDL, `DROP TABLE` reports an error for the named table that does not exist but succeeds for the named table that does exist:

```
mysql> CREATE TABLE t1 (c1 INT);
mysql> DROP TABLE t1, t2;
ERROR 1051 (42S02): Unknown table 'test.t2'
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

Note

Due to this change in behavior, a partially completed `DROP TABLE` statement on a MySQL 5.7 master fails when replicated on a MySQL 8.0 slave. To avoid this failure scenario, use `IF EXISTS` syntax in `DROP TABLE` statements to prevent errors from occurring for tables that do not exist.

- `DROP DATABASE` is atomic if all tables use an atomic DDL-supported storage engine. The statement either drops all objects successfully or is rolled back. However, removal of the database directory from the file system occurs last and is not part of the atomic transaction. If removal of the database directory fails due to a file system error or server halt, the `DROP DATABASE` transaction is not rolled back.

- For tables that do not use an atomic DDL-supported storage engine, table deletion occurs outside of the atomic `DROP TABLE` or `DROP DATABASE` transaction. Such table deletions are written to the binary log individually, which limits the discrepancy between the storage engine, data dictionary, and binary log to one table at most in the case of an interrupted `DROP TABLE` or `DROP DATABASE` operation. For operations that drop multiple tables, the tables that do not use an atomic DDL-supported storage engine are dropped before tables that do.

- `CREATE TABLE`, `ALTER TABLE`, `RENAME TABLE`, `TRUNCATE TABLE`, `CREATE TABLESPACE`, and `DROP TABLESPACE` operations for tables that use an atomic DDL-supported storage engine are either fully committed or rolled back if the server halts during their operation. In earlier MySQL releases, interruption of these operations could cause discrepancies between the storage engine, data dictionary, and binary log, or leave behind orphan files. `RENAME TABLE` operations are only atomic if all named tables use an atomic DDL-supported storage engine.

- `DROP VIEW` fails if a named view does not exist, and no changes are made. The change in behavior is demonstrated in this example, where the `DROP VIEW` statement fails because a named view does not exist:

```
mysql> CREATE VIEW test.viewA AS SELECT * FROM t;
mysql> DROP VIEW test.viewA, test.viewB;
ERROR 1051 (42S02): Unknown table 'test.viewB'
mysql> SHOW FULL TABLES IN test WHERE TABLE_TYPE LIKE 'VIEW';
+----------------+------------+
| Tables_in_test | Table_type |
+----------------+------------+
| viewA          | VIEW       |
+----------------+------------+
```

  Prior to the introduction of atomic DDL, `DROP VIEW` returns an error for the named view that does not exist but succeeds for the named view that does exist:

```
mysql> CREATE VIEW test.viewA AS SELECT * FROM t;
mysql> DROP VIEW test.viewA, test.viewB;
ERROR 1051 (42S02): Unknown table 'test.viewB'
mysql> SHOW FULL TABLES IN test WHERE TABLE_TYPE LIKE 'VIEW';
Empty set (0.00 sec)
```

  Note

  Due to this change in behavior, a partially completed `DROP VIEW` operation on a MySQL 5.7 master fails when replicated on a MySQL 8.0 slave. To avoid this failure scenario, use `IF EXISTS` syntax in `DROP VIEW` statements to prevent an error from occurring for views that do not exist.

- Partial execution of account management statements is no longer permitted. Account management statements either succeed for all named users or roll back and have no effect if an error occurs. In earlier MySQL versions, account management statements that name multiple users could succeed for some users and fail for others.

  The change in behavior is demonstrated in this example, where the second `CREATE USER` statement returns an error but fails because it cannot succeed for all named users.

```
mysql> CREATE USER userA;
mysql> CREATE USER userA, userB;
ERROR 1396 (HY000): Operation CREATE USER failed for 'userA'@'%'
mysql> SELECT User FROM mysql.user WHERE User LIKE 'user%';
+-------+
| User  |
+-------+
| userA |
+-------+
```

Prior to the introduction of atomic DDL, the second `CREATE USER` statement returns an error for the named user that does not exist but succeeds for the named user that does exist:

```
mysql> CREATE USER userA;
mysql> CREATE USER userA, userB;
ERROR 1396 (HY000): Operation CREATE USER failed for 'userA'@'%'
mysql> SELECT User FROM mysql.user WHERE User LIKE 'user%';
+-------+
| User  |
+-------+
| userA |
| userB |
+-------+
```

Note

Due to this change in behavior, partially completed account management statements on a MySQL 5.7 master fail when replicated on a MySQL 8.0 slave. To avoid this failure scenario, use `IF EXISTS` or `IF NOT EXISTS` syntax, as appropriate, in account management statements to prevent errors related to named users.

Storage Engine Support

Currently, only the `InnoDB` storage engine supports atomic DDL. Storage engines that do not support atomic DDL are exempted from DDL atomicity. DDL operations involving exempted storage engines remain capable of introducing inconsistencies that can occur when operations are interrupted or only partially completed.

To support redo and rollback of DDL operations, `InnoDB` writes DDL logs to the `mysql.innodb_ddl_log` table, which is a hidden data dictionary table that resides in the `mysql.ibd` data dictionary tablespace.

To view DDL logs that are written to the `mysql.innodb_ddl_log` table during a DDL operation, enable the `innodb_print_ddl_logs` configuration option. For more information, see Viewing DDL Logs.

Note

The redo logs for changes to the `mysql.innodb_ddl_log` table are flushed to disk immediately regardless of the `innodb_flush_log_at_trx_commit` setting. Flushing the redo logs immediately avoids situations where data files are modified by DDL operations but the redo logs for changes to the `mysql.innodb_ddl_log` table resulting from those operations are not persisted to disk. Such a situation could cause errors during rollback or recovery.

The `InnoDB` storage engine executes DDL operations in phases. DDL operations such as `ALTER TABLE` may perform the *Prepare* and *Perform* phases multiple times prior to the *Commit* phase.

1. *Prepare*: Create the required objects and write the DDL logs to the `mysql.innodb_ddl_log` table. The DDL logs define how to roll forward and roll back the DDL operation.
2. *Perform*: Perform the DDL operation. For example, perform a create routine for a `CREATE TABLE` operation.
3. *Commit*: Update the data dictionary and commit the data dictionary transaction.
4. *Post-DDL*: Replay and remove DDL logs from the `mysql.innodb_ddl_log` table. To ensure that rollback can be performed safely without introducing inconsistencies, file operations such as renaming or removing data files are performed in this final phase. This phase also removes dynamic metadata from the `mysql.innodb_dynamic_metadata` data dictionary table for `DROP TABLE`, `TRUNCATE TABLE`, and other DDL operations that rebuild the table.

DDL logs are replayed and removed from the `mysql.innodb_ddl_log` table during the *Post-DDL* phase, regardless of whether the transaction is committed or rolled back. DDL logs should only remain in the `mysql.innodb_ddl_log` table if the server is halted during a DDL operation. In this case, the DDL logs are replayed and removed after recovery.

In a recovery situation, a DDL transaction may be committed or rolled back when the server is restarted. If the data dictionary transaction that was performed during the *Commit* phase of a DDL operation is present in the redo log and binary log, the operation is considered successful and is rolled forward. Otherwise, the incomplete data dictionary transaction is rolled back when `InnoDB` replays data dictionary redo logs, and the DDL transaction is rolled back.

Viewing DDL Logs

To view DDL logs that are written to the `mysql.innodb_ddl_log` data dictionary table during atomic DDL operations that involve the `InnoDB` storage engine, enable `innodb_print_ddl_logs` to have MySQL write the DDL logs to `stderr`. Depending on the host operating system and MySQL configuration, `stderr` may be the error log, terminal, or console window. See Section 5.4.2.2, “Default Error Log Destination Configuration”.

`InnoDB` writes DDL logs to the `mysql.innodb_ddl_log` table to support redo and rollback of DDL operations. The `mysql.innodb_ddl_log` table is a hidden data dictionary table that resides in the `mysql.ibd` data dictionary tablespace. Like other hidden data dictionary tables, the `mysql.innodb_ddl_log` table cannot be accessed directly in non-debug versions of MySQL. (See Section 14.1, “Data Dictionary Schema”.) The structure of the `mysql.innodb_ddl_log` table corresponds to this definition:

```
CREATE TABLE mysql.innodb_ddl_log (
  id BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  thread_id BIGINT UNSIGNED NOT NULL,
  type INT UNSIGNED NOT NULL,
  space_id INT UNSIGNED,
  page_no INT UNSIGNED,
  index_id BIGINT UNSIGNED,
  table_id BIGINT UNSIGNED,
  old_file_path VARCHAR(512) COLLATE UTF8_BIN,
  new_file_path VARCHAR(512) COLLATE UTF8_BIN,
  KEY(thread_id)
);
```

- `id`: A unique identifier for a DDL log record.
- `thread_id`: Each DDL log record is assigned a `thread_id`, which is used to replay and remove DDL logs that belong to a particular DDL transaction. DDL transactions that involve multiple data file operations generate multiple DDL log records.

- `type`: The DDL operation type. Types include `FREE` (drop an index tree), `DELETE` (delete a file), `RENAME` (rename a file), or `DROP` (drop metadata from the `mysql.innodb_dynamic_metadata` data dictionary table).
- `space_id`: The tablespace ID.
- `page_no`: A page that contains allocation information; an index tree root page, for example.
- `index_id`: The index ID.
- `table_id`: The table ID.
- `old_file_path`: The old tablespace file path. Used by DDL operations that create or drop tablespace files; also used by DDL operations that rename a tablespace.
- `new_file_path`: The new tablespace file path. Used by DDL operations that rename tablespace files.

This example demonstrates enabling `innodb_print_ddl_logs` to view DDL logs written to `strderr` for a `CREATE TABLE` operation.

```
mysql> SET GLOBAL innodb_print_ddl_logs=1;
mysql> CREATE TABLE t1 (c1 INT) ENGINE = InnoDB;
```

```
[Note] [000000] InnoDB: DDL log insert : [DDL record: DELETE SPACE, id=18, thread_id=7,
space_id=5, old_file_path=./test/t1.ibd]
[Note] [000000] InnoDB: DDL log delete : by id 18
[Note] [000000] InnoDB: DDL log insert : [DDL record: REMOVE CACHE, id=19, thread_id=7,
table_id=1058, new_file_path=test/t1]
[Note] [000000] InnoDB: DDL log delete : by id 19
[Note] [000000] InnoDB: DDL log insert : [DDL record: FREE, id=20, thread_id=7,
space_id=5, index_id=132, page_no=4]
[Note] [000000] InnoDB: DDL log delete : by id 20
[Note] [000000] InnoDB: DDL log post ddl : begin for thread id : 7
[Note] [000000] InnoDB: DDL log post ddl : end for thread id : 7
```

13.1.2 ALTER DATABASE Syntax

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification ...

alter_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | DEFAULT ENCRYPTION [=] {'Y' | 'N'}
```

`ALTER DATABASE` enables you to change the overall characteristics of a database. These characteristics are stored in the data dictionary. To use `ALTER DATABASE`, you need the `ALTER` privilege on the database. `ALTER SCHEMA` is a synonym for `ALTER DATABASE`.

The database name can be omitted from the first syntax, in which case the statement applies to the default database.

National Language Characteristics

The `CHARACTER SET` clause changes the default database character set. The `COLLATE` clause changes the default database collation. Chapter 10, *Character Sets, Collations, Unicode*, discusses character set and collation names.

You can see what character sets and collations are available using, respectively, the SHOW CHARACTER SET and SHOW COLLATION statements. See Section 13.7.7.3, “SHOW CHARACTER SET Syntax”, and Section 13.7.7.4, “SHOW COLLATION Syntax”, for more information.

If you change the default character set or collation for a database, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults. (In a stored routine, variables with character data types use the database defaults if the character set or collation are not specified explicitly. See Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.)

Encryption

The DEFAULT ENCRYPTION clause, introduced in MySQL 8.0.16, defines the default database encryption, which is inherited by tables created in the database. Changing the default encryption of the database does not alter the encryption of existing tables associated with the schema. Only newly-created tables inherit the default database encryption. An ALTER DATABASE statement executed without a DEFAULT ENCRYPTION clause retains the existing default database encryption. If the table_encryption_privilege_check variable is enabled, the TABLE_ENCRYPTION_ADMIN privilege is required to specify a default encryption setting that differs from the default_table_encryption setting. For more information, see Defining an Encryption Default for Schemas and General Tablespaces.

13.1.3 ALTER EVENT Syntax

```
ALTER
    [DEFINER = user]
    EVENT event_name
    [ON SCHEDULE schedule]
    [ON COMPLETION [NOT] PRESERVE]
    [RENAME TO new_event_name]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    [DO event_body]
```

The ALTER EVENT statement changes one or more of the characteristics of an existing event without the need to drop and recreate it. The syntax for each of the DEFINER, ON SCHEDULE, ON COMPLETION, COMMENT, ENABLE / DISABLE, and DO clauses is exactly the same as when used with CREATE EVENT. (See Section 13.1.13, “CREATE EVENT Syntax”.)

Any user can alter an event defined on a database for which that user has the EVENT privilege. When a user executes a successful ALTER EVENT statement, that user becomes the definer for the affected event.

ALTER EVENT works only with an existing event:

```
mysql> ALTER EVENT no_such_event
     >     ON SCHEDULE
     >       EVERY '2:3' DAY_HOUR;
ERROR 1517 (HY000): Unknown event 'no_such_event'
```

In each of the following examples, assume that the event named myevent is defined as shown here:

```
CREATE EVENT myevent
    ON SCHEDULE
      EVERY 6 HOUR
    COMMENT 'A sample comment.'
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

The following statement changes the schedule for myevent from once every six hours starting immediately to once every twelve hours, starting four hours from the time the statement is run:

```
ALTER EVENT myevent
    ON SCHEDULE
      EVERY 12 HOUR
    STARTS CURRENT_TIMESTAMP + INTERVAL 4 HOUR;
```

It is possible to change multiple characteristics of an event in a single statement. This example changes the SQL statement executed by `myevent` to one that deletes all records from `mytable`; it also changes the schedule for the event such that it executes once, one day after this `ALTER EVENT` statement is run.

```
ALTER EVENT myevent
    ON SCHEDULE
      AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
    DO
      TRUNCATE TABLE myschema.mytable;
```

Specify the options in an `ALTER EVENT` statement only for those characteristics that you want to change; omitted options keep their existing values. This includes any default values for `CREATE EVENT` such as `ENABLE`.

To disable `myevent`, use this `ALTER EVENT` statement:

```
ALTER EVENT myevent
    DISABLE;
```

The `ON SCHEDULE` clause may use expressions involving built-in MySQL functions and user variables to obtain any of the *`timestamp`* or *`interval`* values which it contains. You cannot use stored routines or user-defined functions in such expressions, and you cannot use any table references; however, you can use `SELECT FROM DUAL`. This is true for both `ALTER EVENT` and `CREATE EVENT` statements. References to stored routines, user-defined functions, and tables in such cases are specifically not permitted, and fail with an error (see Bug #22830).

Although an `ALTER EVENT` statement that contains another `ALTER EVENT` statement in its `DO` clause appears to succeed, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

To rename an event, use the `ALTER EVENT` statement's `RENAME TO` clause. This statement renames the event `myevent` to `yourevent`:

```
ALTER EVENT myevent
    RENAME TO yourevent;
```

You can also move an event to a different database using `ALTER EVENT ... RENAME TO ...` and *`db_name.event_name`* notation, as shown here:

```
ALTER EVENT olddb.myevent
    RENAME TO newdb.myevent;
```

To execute the previous statement, the user executing it must have the `EVENT` privilege on both the `olddb` and `newdb` databases.

Note

There is no `RENAME EVENT` statement.

The value `DISABLE ON SLAVE` is used on a replication slave instead of `ENABLE` or `DISABLE` to indicate an event that was created on the master and replicated to the slave, but that is not executed on the slave.

Normally, DISABLE ON SLAVE is set automatically as required; however, there are some circumstances under which you may want or need to change it manually. See Section 17.5.1.16, “Replication of Invoked Features”, for more information.

13.1.4 ALTER FUNCTION Syntax

```
ALTER FUNCTION func_name [characteristic ...]

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

This statement can be used to change the characteristics of a stored function. More than one change may be specified in an ALTER FUNCTION statement. However, you cannot change the parameters or body of a stored function using this statement; to make such changes, you must drop and re-create the function using DROP FUNCTION and CREATE FUNCTION.

You must have the ALTER ROUTINE privilege for the function. (That privilege is granted automatically to the function creator.) If binary logging is enabled, the ALTER FUNCTION statement might also require the SUPER privilege, as described in Section 24.7, “Stored Program Binary Logging”.

13.1.5 ALTER INSTANCE Syntax

```
ALTER INSTANCE instance_action

instance_action: {
    ROTATE INNODB MASTER KEY
  | ROTATE BINLOG MASTER KEY
  | RELOAD TLS [NO ROLLBACK ON ERROR]
}
```

ALTER INSTANCE defines actions applicable to a MySQL server instance. The statement supports these actions:

- ALTER INSTANCE ROTATE INNODB MASTER KEY

  This action rotates the master encryption key used for InnoDB tablespace encryption. Key rotation requires the ENCRYPTION_KEY_ADMIN or SUPER privilege. To perform this action, a keyring plugin must be installed and configured. For instructions, see Section 6.4.4, “The MySQL Keyring”.

  ALTER INSTANCE ROTATE INNODB MASTER KEY supports concurrent DML. However, it cannot be run concurrently with CREATE TABLE ... ENCRYPTION or ALTER TABLE ... ENCRYPTION operations, and locks are taken to prevent conflicts that could arise from concurrent execution of these statements. If one of the conflicting statements is running, it must complete before another can proceed.

  ALTER INSTANCE ROTATE INNODB MASTER KEY statements are written to the binary log so that they can be executed on replicated servers.

  For additional ALTER INSTANCE ROTATE INNODB MASTER KEY usage information, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”.

- ALTER INSTANCE ROTATE BINLOG MASTER KEY

  This action rotates the binary log master key used for binary log encryption. Key rotation for the binary log master key requires the BINLOG_ENCRYPTION_ADMIN or SUPER privilege. The statement cannot be

used if the `binlog_encryption` system variable is set to `OFF`. To perform this action, a keyring plugin must be installed and configured. For instructions, see Section 6.4.4, “The MySQL Keyring”.

`ALTER INSTANCE ROTATE BINLOG MASTER KEY` actions are not written to the binary log and are not executed on replication slaves. Binary log master key rotation can therefore be carried out in replication environments including a mix of MySQL versions. To schedule regular rotation of the binary log master key on all applicable master and slave servers, you can enable the MySQL Event Scheduler on each server and issue the `ALTER INSTANCE ROTATE BINLOG MASTER KEY` statement using a `CREATE EVENT` statement. If you rotate the binary log master key because you suspect that the current or any of the previous binary log master keys might have been compromised, issue the statement on every applicable master and slave server, which enables you to verify immediate compliance.

For additional `ALTER INSTANCE ROTATE BINLOG MASTER KEY` usage information, including what to do if the process does not complete correctly or is interrupted by an unexpected server halt, see Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”.

- `ALTER INSTANCE RELOAD TLS`

  This action reconfigures the SSL context from the current values of the system variables that define the context. It also updates the status variables that reflect the active context values. This action requires the `CONNECTION_ADMIN` privilege.

  By default, the `RELOAD TLS` action rolls back with an error and has no effect if the configuration values do not permit creation of a new SSL context. The previous context values continue to be used for new connections.

  If the optional `NO ROLLBACK ON ERROR` clause is given and a new context cannot be created, rollback does not occur. Instead, a warning is generated and SSL is disabled for new connections.

  `ALTER INSTANCE RELOAD TLS` statements are not written to the binary log (and thus are not replicated). SSL configuration is local and depends on local files not necessarily present on all servers involved.

  For additional information about reconfiguring the SSL context, including which system and status variables are context-related, see Server-Side Runtime Configuration for Encrypted Connections.

13.1.6 ALTER LOGFILE GROUP Syntax

```
ALTER LOGFILE GROUP logfile_group
    ADD UNDOFILE 'file_name'
    [INITIAL_SIZE [=] size]
    [WAIT]
    ENGINE [=] engine_name
```

This statement adds an `UNDO` file named '*`file_name`*' to an existing log file group *`logfile_group`*. An `ALTER LOGFILE GROUP` statement has one and only one `ADD UNDOFILE` clause. No `DROP UNDOFILE` clause is currently supported.

Note

All NDB Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and an undo log file with the same name, or an undo log file and a data file with the same name.

The optional `INITIAL_SIZE` parameter sets the `UNDO` file's initial size in bytes; if not specified, the initial size defaults to 134217728 (128 MB). You may optionally follow *`size`* with a one-letter abbreviation for an

order of magnitude, similar to those used in `my.cnf`. Generally, this is one of the letters `M` (megabytes) or `G` (gigabytes). (Bug #13116514, Bug #16104705, Bug #62858)

On 32-bit systems, the maximum supported value for `INITIAL_SIZE` is 4294967296 (4 GB). (Bug #29186)

The minimum allowed value for `INITIAL_SIZE` is 1048576 (1 MB). (Bug #29574)

Note

`WAIT` is parsed but otherwise ignored. This keyword currently has no effect, and is intended for future expansion.

The `ENGINE` parameter (required) determines the storage engine which is used by this log file group, with *`engine_name`* being the name of the storage engine. Currently, the only accepted values for *`engine_name`* are "`NDBCLUSTER`" and "`NDB`". The two values are equivalent.

Here is an example, which assumes that the log file group `lg_3` has already been created using `CREATE LOGFILE GROUP` (see Section 13.1.16, "CREATE LOGFILE GROUP Syntax"):

```
ALTER LOGFILE GROUP lg_3
    ADD UNDOFILE 'undo_10.dat'
    INITIAL_SIZE=32M
    ENGINE=NDBCLUSTER;
```

When `ALTER LOGFILE GROUP` is used with `ENGINE = NDBCLUSTER` (alternatively, `ENGINE = NDB`), an `UNDO` log file is created on each NDB Cluster data node. You can verify that the `UNDO` files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example:

```
mysql> SELECT FILE_NAME, LOGFILE_GROUP_NUMBER, EXTRA
    -> FROM INFORMATION_SCHEMA.FILES
    -> WHERE LOGFILE_GROUP_NAME = 'lg_3';
+------------+--------------------+----------------+
| FILE_NAME  | LOGFILE_GROUP_NUMBER | EXTRA          |
+------------+--------------------+----------------+
| newdata.dat |                  0 | CLUSTER_NODE=3 |
| newdata.dat |                  0 | CLUSTER_NODE=4 |
| undo_10.dat |                 11 | CLUSTER_NODE=3 |
| undo_10.dat |                 11 | CLUSTER_NODE=4 |
+------------+--------------------+----------------+
4 rows in set (0.01 sec)
```

(See Section 25.11, "The INFORMATION_SCHEMA FILES Table".)

Memory used for `UNDO_BUFFER_SIZE` comes from the global pool whose size is determined by the value of the `SharedGlobalMemory` data node configuration parameter. This includes any default value implied for this option by the setting of the `InitialLogFileGroup` data node configuration parameter.

`ALTER LOGFILE GROUP` is useful only with Disk Data storage for NDB Cluster. For more information, see Section 22.5.13, "NDB Cluster Disk Data Tables".

13.1.7 ALTER PROCEDURE Syntax

```
ALTER PROCEDURE proc_name [characteristic ...]

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
```

This statement can be used to change the characteristics of a stored procedure. More than one change may be specified in an `ALTER PROCEDURE` statement. However, you cannot change the parameters or body of a stored procedure using this statement; to make such changes, you must drop and re-create the procedure using `DROP PROCEDURE` and `CREATE PROCEDURE`.

You must have the `ALTER ROUTINE` privilege for the procedure. By default, that privilege is granted automatically to the procedure creator. This behavior can be changed by disabling the `automatic_sp_privileges` system variable. See Section 24.2.2, “Stored Routines and MySQL Privileges”.

13.1.8 ALTER SERVER Syntax

```
ALTER SERVER  server_name
    OPTIONS (option [, option] ...)
```

Alters the server information for *`server_name`*, adjusting any of the options permitted in the `CREATE SERVER` statement. The corresponding fields in the `mysql.servers` table are updated accordingly. This statement requires the `SUPER` privilege.

For example, to update the `USER` option:

```
ALTER SERVER s OPTIONS (USER 'sally');
```

`ALTER SERVER` causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

`ALTER SERVER` is not written to the binary log, regardless of the logging format that is in use.

13.1.9 ALTER TABLE Syntax

```
ALTER TABLE tbl_name
    [alter_specification [, alter_specification] ...]
    [partition_options]

alter_specification:
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX|KEY} [index_name]
        [index_type] (key_part,...) [index_option] ...
  | ADD {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name]
        (key_part,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY
        [index_type] (key_part,...)
        [index_option] ...
  | ADD [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
        [index_name] [index_type] (key_part,...)
        [index_option] ...
  | ADD [CONSTRAINT [symbol]] FOREIGN KEY
        [index_name] (col_name,...)
        reference_definition
  | ADD check_constraint_definition
  | DROP CHECK symbol
  | ALTER CHECK symbol [NOT] ENFORCED
  | ALGORITHM [=] {DEFAULT|INSTANT|INPLACE|COPY}
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | ALTER INDEX index_name {VISIBLE | INVISIBLE}
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
        [FIRST|AFTER col_name]
  | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
```

```
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | {DISABLE|ENABLE} KEYS
  | {DISCARD|IMPORT} TABLESPACE
  | DROP [COLUMN] col_name
  | DROP {INDEX|KEY} index_name
  | DROP PRIMARY KEY
  | DROP FOREIGN KEY fk_symbol
  | FORCE
  | LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
  | MODIFY [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ORDER BY col_name [, col_name] ...
  | RENAME COLUMN old_col_name TO new_col_name
  | RENAME {INDEX|KEY} old_index_name TO new_index_name
  | RENAME [TO|AS] new_tbl_name
  | {WITHOUT|WITH} VALIDATION

partition_options:
    partition_option [partition_option] ...

partition_option:
    ADD PARTITION (partition_definition)
  | DROP PARTITION partition_names
  | DISCARD PARTITION {partition_names | ALL} TABLESPACE
  | IMPORT PARTITION {partition_names | ALL} TABLESPACE
  | TRUNCATE PARTITION {partition_names | ALL}
  | COALESCE PARTITION number
  | REORGANIZE PARTITION partition_names INTO (partition_definitions)
  | EXCHANGE PARTITION partition_name WITH TABLE tbl_name [{WITH|WITHOUT} VALIDATION]
  | ANALYZE PARTITION {partition_names | ALL}
  | CHECK PARTITION {partition_names | ALL}
  | OPTIMIZE PARTITION {partition_names | ALL}
  | REBUILD PARTITION {partition_names | ALL}
  | REPAIR PARTITION {partition_names | ALL}
  | REMOVE PARTITIONING

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | {VISIBLE | INVISIBLE}

check_constraint_definition:
    [CONSTRAINT [symbol]] CHECK (expr) [[NOT] ENFORCED]

table_options:
    table_option [[,] table_option] ...

table_option:
    AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | COMPRESSION [=] {'ZLIB'|'LZ4'|'NONE'}
  | CONNECTION [=] 'connect_string'
  | {DATA|INDEX} DIRECTORY [=] 'absolute path to directory'
  | DELAY_KEY_WRITE [=] {0 | 1}
  | ENCRYPTION [=] {'Y' | 'N'}
  | ENGINE [=] engine_name
```

```
  | INSERT_METHOD [=] { NO | FIRST | LAST }
  | KEY_BLOCK_SIZE [=] value
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PACK_KEYS [=] {0 | 1 | DEFAULT}
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
  | STATS_AUTO_RECALC [=] {DEFAULT|0|1}
  | STATS_PERSISTENT [=] {DEFAULT|0|1}
  | STATS_SAMPLE_PAGES [=] value
  | TABLESPACE tablespace_name [STORAGE {DISK|MEMORY}]
  | UNION [=] (tbl_name[,tbl_name]...)

partition_options:
    (see CREATE TABLE options)
```

`ALTER TABLE` changes the structure of a table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change characteristics such as the storage engine used for the table or the table comment.

- To use `ALTER TABLE`, you need `ALTER`, `CREATE`, and `INSERT` privileges for the table. Renaming a table requires `ALTER` and `DROP` on the old table, `ALTER`, `CREATE`, and `INSERT` on the new table.
- Following the table name, specify the alterations to be made. If none are given, `ALTER TABLE` does nothing.
- The syntax for many of the permissible alterations is similar to clauses of the `CREATE TABLE` statement. *`column_definition`* clauses use the same syntax for `ADD` and `CHANGE` as for `CREATE TABLE`. For more information, see Section 13.1.20, “CREATE TABLE Syntax”.
- The word `COLUMN` is optional and can be omitted, except for `RENAME COLUMN` (to distinguish a column-renaming operation from the `RENAME` table-renaming operation).
- Multiple `ADD`, `ALTER`, `DROP`, and `CHANGE` clauses are permitted in a single `ALTER TABLE` statement, separated by commas. This is a MySQL extension to standard SQL, which permits only one of each clause per `ALTER TABLE` statement. For example, to drop multiple columns in a single statement, do this:

  ```
  ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
  ```

- If a storage engine does not support an attempted `ALTER TABLE` operation, a warning may result. Such warnings can be displayed with `SHOW WARNINGS`. See Section 13.7.7.40, “SHOW WARNINGS Syntax”. For information on troubleshooting `ALTER TABLE`, see Section B.4.6.1, “Problems with ALTER TABLE”.
- For information about generated columns, see Section 13.1.9.2, “ALTER TABLE and Generated Columns”.
- For usage examples, see Section 13.1.9.3, “ALTER TABLE Examples”.
- `InnoDB` in MySQL 8.0.17 and later supports addition of multi-valued indexes on JSON columns using a *`key_part`* specification can take the form `(CAST json_path AS type ARRAY)`. See Multi-Valued Indexes, for detailed information regarding multi-valued index creation and usage of, as well as restrictions and limitations on multi-valued indexes.
- With the `mysql_info()` C API function, you can find out how many rows were copied by `ALTER TABLE`. See Section 28.7.7.36, “mysql_info()”.

There are several additional aspects to the `ALTER TABLE` statement, described under the following topics in this section:

- Table Options
- Performance and Space Requirements
- Concurrency Control
- Adding and Dropping Columns
- Renaming, Redefining, and Reordering Columns
- Primary Keys and Indexes
- Foreign Keys and Other Constraints
- Changing the Character Set
- Discarding and Importing InnoDB Tablespaces
- Row Order for MyISAM Tables
- Partitioning Options

Table Options

`table_options` signifies table options of the kind that can be used in the `CREATE TABLE` statement, such as `ENGINE`, `AUTO_INCREMENT`, `AVG_ROW_LENGTH`, `MAX_ROWS`, `ROW_FORMAT`, or `TABLESPACE`.

For descriptions of all table options, see Section 13.1.20, “CREATE TABLE Syntax”. However, `ALTER TABLE` ignores `DATA DIRECTORY` and `INDEX DIRECTORY` when given as table options. `ALTER TABLE` permits them only as partitioning options, and requires that you have the `FILE` privilege.

Use of table options with `ALTER TABLE` provides a convenient way of altering single table characteristics. For example:

- If `t1` is currently not an `InnoDB` table, this statement changes its storage engine to `InnoDB`:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

  - See Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB” for considerations when switching tables to the `InnoDB` storage engine.
  - When you specify an `ENGINE` clause, `ALTER TABLE` rebuilds the table. This is true even if the table already has the specified storage engine.
  - Running `ALTER TABLE` *`tbl_name`* `ENGINE=INNODB` on an existing `InnoDB` table performs a “null” `ALTER TABLE` operation, which can be used to defragment an `InnoDB` table, as described in Section 15.11.4, “Defragmenting a Table”. Running `ALTER TABLE` *`tbl_name`* `FORCE` on an `InnoDB` table performs the same function.
  - `ALTER TABLE` *`tbl_name`* `ENGINE=INNODB` and `ALTER TABLE` *`tbl_name`* `FORCE` use online DDL. For more information, see Section 15.12, “InnoDB and Online DDL”.
  - The outcome of attempting to change the storage engine of a table is affected by whether the desired storage engine is available and the setting of the `NO_ENGINE_SUBSTITUTION` SQL mode, as described in Section 5.1.11, “Server SQL Modes”.
  - To prevent inadvertent loss of data, `ALTER TABLE` cannot be used to change the storage engine of a table to `MERGE` or `BLACKHOLE`.

- To change the `InnoDB` table to use compressed row-storage format:

```
ALTER TABLE t1 ROW_FORMAT = COMPRESSED;
```

- The `ENCRYPTION` clause enables or disables page-level data encryption for an `InnoDB` table. A keyring plugin must be installed and configured to enable encryption.

  If the `table_encryption_privilege_check` variable is enabled, the `TABLE_ENCRYPTION_ADMIN` privilege is required to use an `ENCRYPTION` clause with a setting that differs from the default schema encryption setting.

  Prior to MySQL 8.0.16, the `ENCRYPTION` clause was only supported when altering tables residing in file-per-table tablespaces. As of MySQL 8.0.16, the `ENCRYPTION` clause is also supported for tables residing in general tablespaces.

  For tables that reside in general tablespaces, table and tablespace encryption must match.

  Altering table encryption by moving a table to a different tablespace or changing the storage engine is not permitted without explicitly specifying an `ENCRYPTION` clause.

  As of MySQL 8.0.16, specifying an `ENCRYPTION` clause with a value other than `'N'` or `''` is not permitted if the table uses storage engine that does not support encryption. Previously, the clause was accepted. Attempting to create a table without an `ENCRYPTION` clause in an encryption-enabled schema using a storage engine that does not support encryption is also not permitted.

  For more information, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”.

- To reset the current auto-increment value:

```
ALTER TABLE t1 AUTO_INCREMENT = 13;
```

  You cannot reset the counter to a value less than or equal to the value that is currently in use. For both `InnoDB` and `MyISAM`, if the value is less than or equal to the maximum value currently in the `AUTO_INCREMENT` column, the value is reset to the current maximum `AUTO_INCREMENT` column value plus one.

- To change the default table character set:

```
ALTER TABLE t1 CHARACTER SET = utf8;
```

  See also Changing the Character Set.

- To add (or change) a table comment:

```
ALTER TABLE t1 COMMENT = 'New table comment';
```

- Use `ALTER TABLE` with the `TABLESPACE` option to move `InnoDB` tables between existing general tablespaces, file-per-table tablespaces, and the system tablespace. See Moving Tables Between Tablespaces Using ALTER TABLE.

  - `ALTER TABLE ... TABLESPACE` operations always cause a full table rebuild, even if the `TABLESPACE` attribute has not changed from its previous value.

  - `ALTER TABLE ... TABLESPACE` syntax does not support moving a table from a temporary tablespace to a persistent tablespace.

  - The `DATA DIRECTORY` clause, which is supported with `CREATE TABLE ... TABLESPACE`, is not supported with `ALTER TABLE ... TABLESPACE`, and is ignored if specified.
  - For more information about the capabilities and limitations of the `TABLESPACE` option, see `CREATE TABLE`.

- MySQL NDB Cluster 8.0 supports setting `NDB_TABLE` options for controlling a table's partition balance (fragment count type), read-from-any-replica capability, full replication, or any combination of these, as part of the table comment for an `ALTER TABLE` statement in the same manner as for `CREATE TABLE`, as shown in this example:

```
ALTER TABLE t1 COMMENT = "NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RA_BY_NODE";
```

  Bear in mind that `ALTER TABLE ... COMMENT ...` discards any existing comment for the table. See Setting NDB_TABLE options, for additional information and examples.

To verify that the table options were changed as intended, use `SHOW CREATE TABLE`, or query the `INFORMATION_SCHEMA.TABLES` table.

Performance and Space Requirements

`ALTER TABLE` operations are processed using one of the following algorithms:

- `COPY`: Operations are performed on a copy of the original table, and table data is copied from the original table to the new table row by row. Concurrent DML is not permitted.
- `INPLACE`: Operations avoid copying table data but may rebuild the table in place. An exclusive metadata lock on the table may be taken briefly during preparation and execution phases of the operation. Typically, concurrent DML is supported.
- `INSTANT`: Operations only modify metadata in the data dictionary. No exclusive metadata locks are taken on the table during preparation and execution, and table data is unaffected, making operations instantaneous. Concurrent DML is permitted. (Introduced in MySQL 8.0.12)

The `ALGORITHM` clause is optional. If the `ALGORITHM` clause is omitted, MySQL uses `ALGORITHM=INSTANT` for storage engines and `ALTER TABLE` clauses that support it. Otherwise, `ALGORITHM=INPLACE` is used. If `ALGORITHM=INPLACE` is not supported, `ALGORITHM=COPY` is used.

Specifying an `ALGORITHM` clause requires the operation to use the specified algorithm for clauses and storage engines that support it, or fail with an error otherwise. Specifying `ALGORITHM=DEFAULT` is the same as omitting the `ALGORITHM` clause.

`ALTER TABLE` operations that use the `COPY` algorithm wait for other operations that are modifying the table to complete. After alterations are applied to the table copy, data is copied over, the original table is deleted, and the table copy is renamed to the name of the original table. While the `ALTER TABLE` operation executes, the original table is readable by other sessions (with the exception noted shortly). Updates and writes to the table started after the `ALTER TABLE` operation begins are stalled until the new table is ready, then are automatically redirected to the new table. The temporary copy of the table is created in the database directory of the original table unless it is a `RENAME TO` operation that moves the table to a database that resides in a different directory.

The exception referred to earlier is that `ALTER TABLE` blocks reads (not just writes) at the point where it is ready to clear outdated table structures from the table and table definition caches. At this point, it must acquire an exclusive lock. To do so, it waits for current readers to finish, and blocks new reads and writes.

An ALTER TABLE operation that uses the COPY algorithm prevents concurrent DML operations. Concurrent queries are still allowed. That is, a table-copying operation always includes at least the concurrency restrictions of LOCK=SHARED (allow queries but not DML). You can further restrict concurrency for operations that support the LOCK clause by specifying LOCK=EXCLUSIVE, which prevents DML and queries. For more information, see Concurrency Control.

To force use of the COPY algorithm for an ALTER TABLE operation that would otherwise not use it, specify ALGORITHM=COPY or enable the old_alter_table system variable. If there is a conflict between the old_alter_table setting and an ALGORITHM clause with a value other than DEFAULT, the ALGORITHM clause takes precedence.

For InnoDB tables, an ALTER TABLE operation that uses the COPY algorithm on a table that resides in a shared tablespace can increase the amount of space used by the tablespace. Such operations require as much additional space as the data in the table plus indexes. For a table residing in a shared tablespace, the additional space used during the operation is not released back to the operating system as it is for a table that resides in a file-per-table tablespace.

For information about space requirements for online DDL operations, see Section 15.12.3, “Online DDL Space Requirements”.

ALTER TABLE operations that support the INPLACE algorithm include:

- ALTER TABLE operations supported by the InnoDB online DDL feature. See Section 15.12.1, “Online DDL Operations”.
- Renaming a table. MySQL renames files that correspond to the table *tbl_name* without making a copy. (You can also use the RENAME TABLE statement to rename tables. See Section 13.1.36, “RENAME TABLE Syntax”.) Privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.
- Operations that only modify table metadata. These operations are immediate because the server does not touch table contents. Metadata-only operations include:
  - Renaming a column. In NDB Cluster 8.0.18 and later, this operation can also be performed online.
  - Changing the default value of a column (except for NDB tables).
  - Modifying the definition of an ENUM or SET column by adding new enumeration or set members to the *end* of the list of valid member values, as long as the storage size of the data type does not change. For example, adding a member to a SET column that has 8 members changes the required storage per value from 1 byte to 2 bytes; this requires a table copy. Adding members in the middle of the list causes renumbering of existing members, which requires a table copy.
  - Changing the definition of a spatial column to remove the SRID attribute. (Adding or changing an SRID attribute does require a rebuild and cannot be done in place because the server must verify that all values have the specified SRID value.)
  - As of MySQL 8.0.14, changing a column character set, when these conditions apply:
    - The column data type is CHAR, VARCHAR, a TEXT type, or ENUM.
    - The character set change is from utf8mb3 to utf8mb4, or any character set to binary.
    - There is no index on the column.
  - As of MySQL 8.0.14, changing a generated column, when these conditions apply:

  - For `InnoDB` tables, statements that modify generated stored columns but do not change their type, expression, or nullability.
  - For non-`InnoDB` tables, statements that modify generated stored or virtual columns but do not change their type, expression, or nullability.

  An example of such a change is a change to the column comment.

- Renaming an index.
- Adding or dropping a secondary index, for `InnoDB` and `NDB` tables. See Section 15.12.1, “Online DDL Operations”.
- For `NDB` tables, operations that add and drop indexes on variable-width columns. These operations occur online, without table copying and without blocking concurrent DML actions for most of their duration. See Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”.
- Modifying index visibility with an `ALTER INDEX` operation.
- Column modifications of tables containing generated columns that depend on columns with a `DEFAULT` value if the modified columns are not involved in the generated column expressions. For example, changing the `NULL` property of a separate column can be done in place without a table rebuild.

`ALTER TABLE` operations that support the `INSTANT` algorithm include:

- Adding a column. This feature is referred to as “Instant `ADD COLUMN`”. Limitations apply. See Section 15.12.1, “Online DDL Operations”.
- Adding or dropping a virtual column.
- Adding or dropping a column default value.
- Modifying the definition of an `ENUM` or `SET` column. The same restrictions apply as described above for `ALGORITHM=INSTANT`.
- Changing the index type.
- Renaming a table. The same restrictions apply as described above for `ALGORITHM=INSTANT`.

For more information about operations that support `ALGORITHM=INSTANT`, see Section 15.12.1, “Online DDL Operations”.

`ALTER TABLE` upgrades MySQL 5.5 temporal columns to 5.6 format for `ADD COLUMN`, `CHANGE COLUMN`, `MODIFY COLUMN`, `ADD INDEX`, and `FORCE` operations. This conversion cannot be done using the `INPLACE` algorithm because the table must be rebuilt, so specifying `ALGORITHM=INPLACE` in these cases results in an error. Specify `ALGORITHM=COPY` if necessary.

If an `ALTER TABLE` operation on a multicolumn index used to partition a table by `KEY` changes the order of the columns, it can only be performed using `ALGORITHM=COPY`.

The `WITHOUT VALIDATION` and `WITH VALIDATION` clauses affect whether `ALTER TABLE` performs an in-place operation for virtual generated column modifications. See Section 13.1.9.2, “ALTER TABLE and Generated Columns”.

NDB Cluster 8.0 supports online operations using the same `ALGORITHM=INPLACE` syntax used with the standard MySQL Server. See Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”, for more information.

`ALTER TABLE` with `DISCARD ... PARTITION ... TABLESPACE` or `IMPORT ... PARTITION ... TABLESPACE` does not create any temporary tables or temporary partition files.

`ALTER TABLE` with `ADD PARTITION`, `DROP PARTITION`, `COALESCE PARTITION`, `REBUILD PARTITION`, or `REORGANIZE PARTITION` does not create temporary tables (except when used with `NDB` tables); however, these operations can and do create temporary partition files.

`ADD` or `DROP` operations for `RANGE` or `LIST` partitions are immediate operations or nearly so. `ADD` or `COALESCE` operations for `HASH` or `KEY` partitions copy data between all partitions, unless `LINEAR HASH` or `LINEAR KEY` was used; this is effectively the same as creating a new table, although the `ADD` or `COALESCE` operation is performed partition by partition. `REORGANIZE` operations copy only changed partitions and do not touch unchanged ones.

For `MyISAM` tables, you can speed up index re-creation (the slowest part of the alteration process) by setting the `myisam_sort_buffer_size` system variable to a high value.

Concurrency Control

For `ALTER TABLE` operations that support it, you can use the `LOCK` clause to control the level of concurrent reads and writes on a table while it is being altered. Specifying a non-default value for this clause enables you to require a certain amount of concurrent access or exclusivity during the alter operation, and halts the operation if the requested degree of locking is not available.

Only `LOCK = DEFAULT` is permitted for operations that use `ALGORITHM=INSTANT`. The other `LOCK` clause parameters are not applicable.

The parameters for the `LOCK` clause are:

- ```
  LOCK = DEFAULT
  ```

  Maximum level of concurrency for the given `ALGORITHM` clause (if any) and `ALTER TABLE` operation: Permit concurrent reads and writes if supported. If not, permit concurrent reads if supported. If not, enforce exclusive access.

- ```
  LOCK = NONE
  ```

  If supported, permit concurrent reads and writes. Otherwise, an error occurs.

- ```
  LOCK = SHARED
  ```

  If supported, permit concurrent reads but block writes. Writes are blocked even if concurrent writes are supported by the storage engine for the given `ALGORITHM` clause (if any) and `ALTER TABLE` operation. If concurrent reads are not supported, an error occurs.

- ```
  LOCK = EXCLUSIVE
  ```

  Enforce exclusive access. This is done even if concurrent reads/writes are supported by the storage engine for the given `ALGORITHM` clause (if any) and `ALTER TABLE` operation.

Adding and Dropping Columns

Use `ADD` to add new columns to a table, and `DROP` to remove existing columns. `DROP` *`col_name`* is a MySQL extension to standard SQL.

To add a column at a specific position within a table row, use `FIRST` or `AFTER` *`col_name`*. The default is to add the column last.

If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use the `DROP TABLE` statement instead.

If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well. If you use `CHANGE` or `MODIFY` to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.

For `ALTER TABLE ... ADD`, if the column has an expression default value that uses a nondeterministic function, the statement may produce a warning or error. For details, see Section 17.1.3.6, “Restrictions on Replication with GTIDs”.

Renaming, Redefining, and Reordering Columns

The `CHANGE`, `MODIFY`, `RENAME COLUMN`, and `ALTER` clauses enable the names and definitions of existing columns to be altered. They have these comparative characteristics:

- `CHANGE`:
  - Can rename a column and change its definition, or both.
  - Has more capability than `MODIFY` or `RENAME COLUMN`, but at the expense of convenience for some operations. `CHANGE` requires naming the column twice if not renaming it, and requires respecifying the column definition if only renaming it.
  - With `FIRST` or `AFTER`, can reorder columns.
- `MODIFY`:
  - Can change a column definition but not its name.
  - More convenient than `CHANGE` to change a column definition without renaming it.
  - With `FIRST` or `AFTER`, can reorder columns.
- `RENAME COLUMN`:
  - Can change a column name but not its definition.
  - More convenient than `CHANGE` to rename a column without changing its definition.
- `ALTER`: Used only to change a column default value.

`CHANGE` is a MySQL extension to standard SQL. `MODIFY` and `RENAME COLUMN` are MySQL extensions for Oracle compatibility.

To alter a column to change both its name and definition, use `CHANGE`, specifying the old and new names and the new definition. For example, to rename an `INT NOT NULL` column from `a` to `b` and change its definition to use the `BIGINT` data type while retaining the `NOT NULL` attribute, do this:

```
ALTER TABLE t1 CHANGE a b BIGINT NOT NULL;
```

To change a column definition but not its name, use `CHANGE` or `MODIFY`. With `CHANGE`, the syntax requires two column names, so you must specify the same name twice to leave the name unchanged. For example, to change the definition of column `b`, do this:

```
ALTER TABLE t1 CHANGE b b INT NOT NULL;
```

`MODIFY` is more convenient to change the definition without changing the name because it requires the column name only once:

```
ALTER TABLE t1 MODIFY b INT NOT NULL;
```

To change a column name but not its definition, use `CHANGE` or `RENAME COLUMN`. With `CHANGE`, the syntax requires a column definition, so to leave the definition unchanged, you must respecify the definition the column currently has. For example, to rename an `INT NOT NULL` column from `b` to `a`, do this:

```
ALTER TABLE t1 CHANGE b a INT NOT NULL;
```

`RENAME COLUMN` is more convenient to change the name without changing the definition because it requires only the old and new names:

```
ALTER TABLE t1 RENAME COLUMN b TO a;
```

In general, you cannot rename a column to a name that already exists in the table. However, this is sometimes not the case, such as when you swap names or move them through a cycle. If a table has columns named `a`, `b`, and `c`, these are valid operations:

```
-- swap a and b
ALTER TABLE t1 RENAME COLUMN a TO b,
               RENAME COLUMN b TO a;
-- "rotate" a, b, c through a cycle
ALTER TABLE t1 RENAME COLUMN a TO b,
               RENAME COLUMN b TO c,
               RENAME COLUMN c TO a;
```

For column definition changes using `CHANGE` or `MODIFY`, the definition must include the data type and all attributes that should apply to the new column, other than index attributes such as `PRIMARY KEY` or `UNIQUE`. Attributes present in the original definition but not specified for the new definition are not carried forward. Suppose that a column `col1` is defined as `INT UNSIGNED DEFAULT 1 COMMENT 'my column'` and you modify the column as follows, intending to change only `INT` to `BIGINT`:

```
ALTER TABLE t1 MODIFY col1 BIGINT;
```

That statement changes the data type from `INT` to `BIGINT`, but it also drops the `UNSIGNED`, `DEFAULT`, and `COMMENT` attributes. To retain them, the statement must include them explicitly:

```
ALTER TABLE t1 MODIFY col1 BIGINT UNSIGNED DEFAULT 1 COMMENT 'my column';
```

For data type changes using `CHANGE` or `MODIFY`, MySQL tries to convert existing column values to the new type as well as possible.

Warning

This conversion may result in alteration of data. For example, if you shorten a string column, values may be truncated. To prevent the operation from succeeding if conversions to the new data type would result in loss of data, enable strict SQL mode before using `ALTER TABLE` (see Section 5.1.11, “Server SQL Modes”).

If you use CHANGE or MODIFY to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.

For columns renamed by CHANGE or RENAME COLUMN, MySQL automatically renames these references to the renamed column:

- Indexes that refer to the old column, including invisible indexes and disabled MyISAM indexes.
- Foreign keys that refer to the old column.

For columns renamed by CHANGE or RENAME COLUMN, MySQL does not automatically rename these references to the renamed column:

- Generated column and partition expressions that refer to the renamed column. You must use CHANGE to redefine such expressions in the same ALTER TABLE statement as the one that renames the column.
- Views and stored programs that refer to the renamed column. You must manually alter the definition of these objects to refer to the new column name.

To reorder columns within a table, use FIRST and AFTER in CHANGE or MODIFY operations.

ALTER ... SET DEFAULT or ALTER ... DROP DEFAULT specify a new default value for a column or remove the old default value, respectively. If the old default is removed and the column can be NULL, the new default is NULL. If the column cannot be NULL, MySQL assigns a default value as described in Section 11.7, “Data Type Default Values”.

Primary Keys and Indexes

DROP PRIMARY KEY drops the primary key. If there is no primary key, an error occurs. For information about the performance characteristics of primary keys, especially for InnoDB tables, see Section 8.3.2, “Primary Key Optimization”.

If you add a UNIQUE INDEX or PRIMARY KEY to a table, MySQL stores it before any nonunique index to permit detection of duplicate keys as early as possible.

DROP INDEX removes an index. This is a MySQL extension to standard SQL. See Section 13.1.27, “DROP INDEX Syntax”. To determine index names, use SHOW INDEX FROM *tbl_name*.

Some storage engines permit you to specify an index type when creating an index. The syntax for the *index_type* specifier is USING *type_name*. For details about USING, see Section 13.1.15, “CREATE INDEX Syntax”. The preferred position is after the column list. Support for use of the option before the column list will be removed in a future MySQL release.

index_option values specify additional options for an index. USING is one such option. For details about permissible *index_option* values, see Section 13.1.15, “CREATE INDEX Syntax”.

RENAME INDEX *old_index_name* TO *new_index_name* renames an index. This is a MySQL extension to standard SQL. The content of the table remains unchanged. *old_index_name* must be the name of an existing index in the table that is not dropped by the same ALTER TABLE statement. *new_index_name* is the new index name, which cannot duplicate the name of an index in the resulting table after changes have been applied. Neither index name can be PRIMARY.

If you use ALTER TABLE on a MyISAM table, all nonunique indexes are created in a separate batch (as for REPAIR TABLE). This should make ALTER TABLE much faster when you have many indexes.

For MyISAM tables, key updating can be controlled explicitly. Use ALTER TABLE ... DISABLE KEYS to tell MySQL to stop updating nonunique indexes. Then use ALTER TABLE ... ENABLE KEYS to

re-create missing indexes. `MyISAM` does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should give a considerable speedup. Using `ALTER TABLE ... DISABLE KEYS` requires the `INDEX` privilege in addition to the privileges mentioned earlier.

While the nonunique indexes are disabled, they are ignored for statements such as `SELECT` and `EXPLAIN` that otherwise would use them.

After an `ALTER TABLE` statement, it may be necessary to run `ANALYZE TABLE` to update index cardinality information. See Section 13.7.7.22, “SHOW INDEX Syntax”.

The `ALTER INDEX` operation permits an index to be made visible or invisible. An invisible index is not used by the optimizer. Modification of index visibility applies to indexes other than primary keys (either explicit or implicit). This feature is storage engine neutral (supported for any engine). For more information, see Section 8.3.12, “Invisible Indexes”.

Foreign Keys and Other Constraints

The `FOREIGN KEY` and `REFERENCES` clauses are supported by the `InnoDB` and `NDB` storage engines, which implement `ADD [CONSTRAINT [symbol]] FOREIGN KEY [index_name] (...) REFERENCES ... (...)`. See Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”. For other storage engines, the clauses are parsed but ignored.

For `ALTER TABLE`, unlike `CREATE TABLE`, `ADD FOREIGN KEY` ignores *`index_name`* if given and uses an automatically generated foreign key name. As a workaround, include the `CONSTRAINT` clause to specify the foreign key name:

```
ADD CONSTRAINT name FOREIGN KEY (....) ...
```

Important

MySQL silently ignores inline `REFERENCES` specifications, where the references are defined as part of the column specification. MySQL accepts only `REFERENCES` clauses defined as part of a separate `FOREIGN KEY` specification.

Note

Partitioned `InnoDB` tables do not support foreign keys. This restriction does not apply to `NDB` tables, including those explicitly partitioned by `[LINEAR] KEY`. For more information, see Section 23.6.2, “Partitioning Limitations Relating to Storage Engines”.

MySQL Server and NDB Cluster both support the use of `ALTER TABLE` to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

Adding and dropping a foreign key in the same `ALTER TABLE` statement is supported for `ALTER TABLE ... ALGORITHM=INPLACE` but not for `ALTER TABLE ... ALGORITHM=COPY`.

The server prohibits changes to foreign key columns that have the potential to cause loss of referential integrity. A workaround is to use `ALTER TABLE ... DROP FOREIGN KEY` before changing the column definition and `ALTER TABLE ... ADD FOREIGN KEY` afterward. Examples of prohibited changes include:

- Changes to the data type of foreign key columns that may be unsafe. For example, changing `VARCHAR(20)` to `VARCHAR(30)` is permitted, but changing it to `VARCHAR(1024)` is not because that alters the number of length bytes required to store individual values.
- Changing a `NULL` column to `NOT NULL` in non-strict mode is prohibited to prevent converting `NULL` values to default non-`NULL` values, for which there are no corresponding values in the referenced table. The operation is permitted in strict mode, but an error is returned if any such conversion is required.

`ALTER TABLE tbl_name RENAME new_tbl_name` changes internally generated foreign key constraint names and user-defined foreign key constraint names that begin with the string "*`tbl_name`*_ibfk_" to reflect the new table name. `InnoDB` interprets foreign key constraint names that begin with the string "*`tbl_name`*_ibfk_" as internally generated names.

Prior to MySQL 8.0.16, `ALTER TABLE` permits only the following limited version of `CHECK` constraint-adding syntax, which is parsed and ignored:

```
ADD CHECK (expr)
```

As of MySQL 8.0.16, `ALTER TABLE` permits `CHECK` constraints for existing tables to be added, dropped, or altered:

- Add a new constraint:

  ```
  ALTER TABLE tbl_name
      ADD CONSTRAINT [symbol] CHECK (expr) [[NOT] ENFORCED];
  ```

  The meaning of constraint syntax elements is the same as for `CREATE TABLE`. See Section 13.1.20.7, "CHECK Constraints".

- Drop an existing constraint named *`symbol`*:

  ```
  ALTER TABLE tbl_name
      DROP CHECK symbol;
  ```

- Alter whether an existing constraint named *`symbol`* is enforced:

  ```
  ALTER TABLE tbl_name
      ALTER CHECK symbol [NOT] ENFORCED;
  ```

The `DROP CHECK` and `ALTER CHECK` clauses are MySQL extensions to standard SQL.

If a table alteration causes a violation of an enforced `CHECK` constraint, an error occurs and the table is not modified. Examples of operations for which an error occurs:

- Attempts to add the `AUTO_INCREMENT` attribute to a column that is used in a `CHECK` constraint.
- Attempts to add an enforced `CHECK` constraint or enforce a nonenforced `CHECK` constraint for which existing rows violate the constraint condition.
- Attempts to modify, rename, or drop a column that is used in a `CHECK` constraint, unless that constraint is also dropped in the same statement. Exception: If a `CHECK` constraint refers only to a single column, dropping the column automatically drops the constraint.

`ALTER TABLE tbl_name RENAME new_tbl_name` changes internally generated and user-defined `CHECK` constraint names that begin with the string "*`tbl_name`*_chk_" to reflect the new table name. MySQL interprets `CHECK` constraint names that begin with the string "*`tbl_name`*_chk_" as internally generated names.

Changing the Character Set

To change the table default character set and all character columns (CHAR, VARCHAR, TEXT) to a new character set, use a statement like this:

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
```

The statement also changes the collation of all character columns. If you specify no COLLATE clause to indicate which collation to use, the statement uses default collation for the character set. If this collation is inappropriate for the intended table use (for example, if it would change from a case-sensitive collation to a case-insensitive collation), specify a collation explicitly.

For a column that has a data type of VARCHAR or one of the TEXT types, CONVERT TO CHARACTER SET changes the data type as necessary to ensure that the new column is long enough to store as many characters as the original column. For example, a TEXT column has two length bytes, which store the byte-length of values in the column, up to a maximum of 65,535. For a latin1 TEXT column, each character requires a single byte, so the column can store up to 65,535 characters. If the column is converted to utf8, each character might require up to three bytes, for a maximum possible length of 3 × 65,535 = 196,605 bytes. That length does not fit in a TEXT column's length bytes, so MySQL converts the data type to MEDIUMTEXT, which is the smallest string type for which the length bytes can record a value of 196,605. Similarly, a VARCHAR column might be converted to MEDIUMTEXT.

To avoid data type changes of the type just described, do not use CONVERT TO CHARACTER SET. Instead, use MODIFY to change individual columns. For example:

```
ALTER TABLE t MODIFY latin1_text_col TEXT CHARACTER SET utf8;
ALTER TABLE t MODIFY latin1_varchar_col VARCHAR(M) CHARACTER SET utf8;
```

If you specify CONVERT TO CHARACTER SET binary, the CHAR, VARCHAR, and TEXT columns are converted to their corresponding binary string types (BINARY, VARBINARY, BLOB). This means that the columns no longer will have a character set and a subsequent CONVERT TO operation will not apply to them.

If *charset_name* is DEFAULT in a CONVERT TO CHARACTER SET operation, the character set named by the character_set_database system variable is used.

Warning

The CONVERT TO operation converts column values between the original and named character sets. This is *not* what you want if you have a column in one character set (like latin1) but the stored values actually use some other, incompatible character set (like utf8). In this case, you have to do the following for each such column:

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

The reason this works is that there is no conversion when you convert to or from BLOB columns.

To change only the *default* character set for a table, use this statement:

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

The word DEFAULT is optional. The default character set is the character set that is used if you do not specify the character set for columns that you add to a table later (for example, with ALTER TABLE ... ADD column).

When the foreign_key_checks system variable is enabled, which is the default setting, character set conversion is not permitted on tables that include a character string column used in a foreign key constraint. The workaround is to disable foreign_key_checks before performing the character set conversion. You must perform the conversion on both tables involved in the foreign key constraint before re-enabling foreign_key_checks. If you re-enable foreign_key_checks after converting only one of the tables, an ON DELETE CASCADE or ON UPDATE CASCADE operation could corrupt data in the referencing table due to implicit conversion that occurs during these operations (Bug #45290, Bug #74816).

Discarding and Importing InnoDB Tablespaces

An InnoDB table created in its own file-per-table tablespace can be discarded and imported using the DISCARD TABLESPACE and IMPORT TABLESPACE options. These options can be used to import a file-per-table tablespace from a backup or to copy a file-per-table tablespace from one database server to another. See Section 15.6.3.7, “Copying Tablespaces to Another Instance”.

Row Order for MyISAM Tables

ORDER BY enables you to create the new table with the rows in a specific order. This option is useful primarily when you know that you query the rows in a certain order most of the time. By using this option after major changes to the table, you might be able to get higher performance. In some cases, it might make sorting easier for MySQL if the table is in order by the column that you want to order it by later.

Note

The table does not remain in the specified order after inserts and deletes.

ORDER BY syntax permits one or more column names to be specified for sorting, each of which optionally can be followed by ASC or DESC to indicate ascending or descending sort order, respectively. The default is ascending order. Only column names are permitted as sort criteria; arbitrary expressions are not permitted. This clause should be given last after any other clauses.

ORDER BY does not make sense for InnoDB tables because InnoDB always orders table rows according to the clustered index.

When used on a partitioned table, ALTER TABLE ... ORDER BY orders rows within each partition only.

Partitioning Options

partition_options signifies options that can be used with partitioned tables for repartitioning, to add, drop, discard, import, merge, and split partitions, and to perform partitioning maintenance.

It is possible for an ALTER TABLE statement to contain a PARTITION BY or REMOVE PARTITIONING clause in an addition to other alter specifications, but the PARTITION BY or REMOVE PARTITIONING clause must be specified last after any other specifications. The ADD PARTITION, DROP PARTITION, DISCARD PARTITION, IMPORT PARTITION, COALESCE PARTITION, REORGANIZE PARTITION, EXCHANGE PARTITION, ANALYZE PARTITION, CHECK PARTITION, and REPAIR PARTITION options cannot be combined with other alter specifications in a single ALTER TABLE, since the options just listed act on individual partitions.

For more information about partition options, see Section 13.1.20, “CREATE TABLE Syntax”, and Section 13.1.9.1, “ALTER TABLE Partition Operations”. For information about and examples of ALTER

`TABLE ... EXCHANGE PARTITION` statements, see Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”.

13.1.9.1 ALTER TABLE Partition Operations

Partitioning-related clauses for `ALTER TABLE` can be used with partitioned tables for repartitioning, to add, drop, discard, import, merge, and split partitions, and to perform partitioning maintenance.

- Simply using a *`partition_options`* clause with `ALTER TABLE` on a partitioned table repartitions the table according to the partitioning scheme defined by the *`partition_options`*. This clause always begins with `PARTITION BY`, and follows the same syntax and other rules as apply to the *`partition_options`* clause for `CREATE TABLE` (for more detailed information, see Section 13.1.20, “CREATE TABLE Syntax”), and can also be used to partition an existing table that is not already partitioned. For example, consider a (nonpartitioned) table defined as shown here:

```
CREATE TABLE t1 (
    id INT,
    year_col INT
);
```

This table can be partitioned by `HASH`, using the `id` column as the partitioning key, into 8 partitions by means of this statement:

```
ALTER TABLE t1
    PARTITION BY HASH(id)
    PARTITIONS 8;
```

MySQL supports an `ALGORITHM` option with `[SUB]PARTITION BY [LINEAR] KEY`. `ALGORITHM=1` causes the server to use the same key-hashing functions as MySQL 5.1 when computing the placement of rows in partitions; `ALGORITHM=2` means that the server employs the key-hashing functions implemented and used by default for new `KEY` partitioned tables in MySQL 5.5 and later. (Partitioned tables created with the key-hashing functions employed in MySQL 5.5 and later cannot be used by a MySQL 5.1 server.) Not specifying the option has the same effect as using `ALGORITHM=2`. This option is intended for use chiefly when upgrading or downgrading `[LINEAR] KEY` partitioned tables between MySQL 5.1 and later MySQL versions, or for creating tables partitioned by `KEY` or `LINEAR KEY` on a MySQL 5.5 or later server which can be used on a MySQL 5.1 server.

The table that results from using an `ALTER TABLE ... PARTITION BY` statement must follow the same rules as one created using `CREATE TABLE ... PARTITION BY`. This includes the rules governing the relationship between any unique keys (including any primary key) that the table might have, and the column or columns used in the partitioning expression, as discussed in Section 23.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”. The `CREATE TABLE ... PARTITION BY` rules for specifying the number of partitions also apply to `ALTER TABLE ... PARTITION BY`.

The *`partition_definition`* clause for `ALTER TABLE ADD PARTITION` supports the same options as the clause of the same name for the `CREATE TABLE` statement. (See Section 13.1.20, “CREATE TABLE Syntax”, for the syntax and description.) Suppose that you have the partitioned table created as shown here:

```
CREATE TABLE t1 (
    id INT,
    year_col INT
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
```

```
    PARTITION p2 VALUES LESS THAN (1999)
);
```

You can add a new partition `p3` to this table for storing values less than `2002` as follows:

```
ALTER TABLE t1 ADD PARTITION (PARTITION p3 VALUES LESS THAN (2002));
```

`DROP PARTITION` can be used to drop one or more `RANGE` or `LIST` partitions. This statement cannot be used with `HASH` or `KEY` partitions; instead, use `COALESCE PARTITION` (see later in this section). Any data that was stored in the dropped partitions named in the *`partition_names`* list is discarded. For example, given the table `t1` defined previously, you can drop the partitions named `p0` and `p1` as shown here:

```
ALTER TABLE t1 DROP PARTITION p0, p1;
```

Note

`DROP PARTITION` does not work with tables that use the `NDB` storage engine. See Section 23.3.1, “Management of RANGE and LIST Partitions”, and Section 22.1.7, “Known Limitations of NDB Cluster”.

`ADD PARTITION` and `DROP PARTITION` do not currently support `IF [NOT] EXISTS`.

The `DISCARD PARTITION ... TABLESPACE` and `IMPORT PARTITION ... TABLESPACE` options extend the Transportable Tablespace feature to individual `InnoDB` table partitions. Each `InnoDB` table partition has its own tablespace file (`.ibd` file). The Transportable Tablespace feature makes it easy to copy the tablespaces from a running MySQL server instance to another running instance, or to perform a restore on the same instance. Both options take a comma-separated list of one or more partition names. For example:

```
ALTER TABLE t1 DISCARD PARTITION p2, p3 TABLESPACE;
```

```
ALTER TABLE t1 IMPORT PARTITION p2, p3 TABLESPACE;
```

When running `DISCARD PARTITION ... TABLESPACE` and `IMPORT PARTITION ... TABLESPACE` on subpartitioned tables, both partition and subpartition names are allowed. When a partition name is specified, subpartitions of that partition are included.

The Transportable Tablespace feature also supports copying or restoring partitioned `InnoDB` tables (all partitions at once). For additional information, see Section 15.6.3.7, “Copying Tablespaces to Another Instance”, as well as, Transportable Tablespace Examples.

Renames of partitioned tables are supported. You can rename individual partitions indirectly using `ALTER TABLE ... REORGANIZE PARTITION`; however, this operation copies the partition's data.

To delete rows from selected partitions, use the `TRUNCATE PARTITION` option. This option takes a list of one or more comma-separated partition names. Consider the table `t1` created by this statement:

```
CREATE TABLE t1 (
    id INT,
    year_col INT
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
```

```
    PARTITION p2 VALUES LESS THAN (1999),
    PARTITION p3 VALUES LESS THAN (2003),
    PARTITION p4 VALUES LESS THAN (2007)
);
```

To delete all rows from partition `p0`, use the following statement:

```
ALTER TABLE t1 TRUNCATE PARTITION p0;
```

The statement just shown has the same effect as the following `DELETE` statement:

```
DELETE FROM t1 WHERE year_col < 1991;
```

When truncating multiple partitions, the partitions do not have to be contiguous: This can greatly simplify delete operations on partitioned tables that would otherwise require very complex `WHERE` conditions if done with `DELETE` statements. For example, this statement deletes all rows from partitions `p1` and `p3`:

```
ALTER TABLE t1 TRUNCATE PARTITION p1, p3;
```

An equivalent `DELETE` statement is shown here:

```
DELETE FROM t1 WHERE
    (year_col >= 1991 AND year_col < 1995)
    OR
    (year_col >= 2003 AND year_col < 2007);
```

If you use the `ALL` keyword in place of the list of partition names, the statement acts on all table partitions.

`TRUNCATE PARTITION` merely deletes rows; it does not alter the definition of the table itself, or of any of its partitions.

To verify that the rows were dropped, check the `INFORMATION_SCHEMA.PARTITIONS` table, using a query such as this one:

```
SELECT PARTITION_NAME, TABLE_ROWS
    FROM INFORMATION_SCHEMA.PARTITIONS
    WHERE TABLE_NAME = 't1';
```

`COALESCE PARTITION` can be used with a table that is partitioned by `HASH` or `KEY` to reduce the number of partitions by *`number`*. Suppose that you have created table `t2` as follows:

```
CREATE TABLE t2 (
    name VARCHAR (30),
    started DATE
)
PARTITION BY HASH( YEAR(started) )
PARTITIONS 6;
```

To reduce the number of partitions used by `t2` from 6 to 4, use the following statement:

```
ALTER TABLE t2 COALESCE PARTITION 2;
```

The data contained in the last *`number`* partitions will be merged into the remaining partitions. In this case, partitions 4 and 5 will be merged into the first 4 partitions (the partitions numbered 0, 1, 2, and 3).

To change some but not all the partitions used by a partitioned table, you can use `REORGANIZE PARTITION`. This statement can be used in several ways:

- To merge a set of partitions into a single partition. This is done by naming several partitions in the *`partition_names`* list and supplying a single definition for *`partition_definition`*.
- To split an existing partition into several partitions. Accomplish this by naming a single partition for *`partition_names`* and providing multiple *`partition_definitions`*.
- To change the ranges for a subset of partitions defined using `VALUES LESS THAN` or the value lists for a subset of partitions defined using `VALUES IN`.

Note

For partitions that have not been explicitly named, MySQL automatically provides the default names `p0`, `p1`, `p2`, and so on. The same is true with regard to subpartitions.

For more detailed information about and examples of `ALTER TABLE ... REORGANIZE PARTITION` statements, see Section 23.3.1, “Management of RANGE and LIST Partitions”.

- To exchange a table partition or subpartition with a table, use the `ALTER TABLE ... EXCHANGE PARTITION` statement—that is, to move any existing rows in the partition or subpartition to the nonpartitioned table, and any existing rows in the nonpartitioned table to the table partition or subpartition.

  For usage information and examples, see Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”.

- Several options provide partition maintenance and repair functionality analogous to that implemented for nonpartitioned tables by statements such as `CHECK TABLE` and `REPAIR TABLE` (which are also supported for partitioned tables; for more information, see Section 13.7.3, “Table Maintenance Statements”). These include `ANALYZE PARTITION`, `CHECK PARTITION`, `OPTIMIZE PARTITION`, `REBUILD PARTITION`, and `REPAIR PARTITION`. Each of these options takes a *`partition_names`* clause consisting of one or more names of partitions, separated by commas. The partitions must already exist in the target table. You can also use the `ALL` keyword in place of *`partition_names`*, in which case the statement acts on all table partitions. For more information and examples, see Section 23.3.4, “Maintenance of Partitions”.

  `InnoDB` does not currently support per-partition optimization; `ALTER TABLE ... OPTIMIZE PARTITION` causes the entire table to rebuilt and analyzed, and an appropriate warning to be issued. (Bug #11751825, Bug #42822) To work around this problem, use `ALTER TABLE ... REBUILD PARTITION` and `ALTER TABLE ... ANALYZE PARTITION` instead.

  The `ANALYZE PARTITION`, `CHECK PARTITION`, `OPTIMIZE PARTITION`, and `REPAIR PARTITION` options are not supported for tables which are not partitioned.

- `REMOVE PARTITIONING` enables you to remove a table's partitioning without otherwise affecting the table or its data. This option can be combined with other `ALTER TABLE` options such as those used to add, drop, or rename columns or indexes.

- Using the `ENGINE` option with `ALTER TABLE` changes the storage engine used by the table without affecting the partitioning. The target storage engine must provide its own partitioning handler. Only the `InnoDB` and `NDB` storage engines have native partitioning handlers; `NDB` is not currently supported in MySQL 8.0.

It is possible for an ALTER TABLE statement to contain a PARTITION BY or REMOVE PARTITIONING clause in an addition to other alter specifications, but the PARTITION BY or REMOVE PARTITIONING clause must be specified last after any other specifications.

The ADD PARTITION, DROP PARTITION, COALESCE PARTITION, REORGANIZE PARTITION, ANALYZE PARTITION, CHECK PARTITION, and REPAIR PARTITION options cannot be combined with other alter specifications in a single ALTER TABLE, since the options just listed act on individual partitions. For more information, see Section 13.1.9.1, “ALTER TABLE Partition Operations”.

Only a single instance of any one of the following options can be used in a given ALTER TABLE statement: PARTITION BY, ADD PARTITION, DROP PARTITION, TRUNCATE PARTITION, EXCHANGE PARTITION, REORGANIZE PARTITION, or COALESCE PARTITION, ANALYZE PARTITION, CHECK PARTITION, OPTIMIZE PARTITION, REBUILD PARTITION, REMOVE PARTITIONING.

For example, the following two statements are invalid:

```
ALTER TABLE t1 ANALYZE PARTITION p1, ANALYZE PARTITION p2;

ALTER TABLE t1 ANALYZE PARTITION p1, CHECK PARTITION p2;
```

In the first case, you can analyze partitions p1 and p2 of table t1 concurrently using a single statement with a single ANALYZE PARTITION option that lists both of the partitions to be analyzed, like this:

```
ALTER TABLE t1 ANALYZE PARTITION p1, p2;
```

In the second case, it is not possible to perform ANALYZE and CHECK operations on different partitions of the same table concurrently. Instead, you must issue two separate statements, like this:

```
ALTER TABLE t1 ANALYZE PARTITION p1;
ALTER TABLE t1 CHECK PARTITION p2;
```

REBUILD operations are currently unsupported for subpartitions. The REBUILD keyword is expressly disallowed with subpartitions, and causes ALTER TABLE to fail with an error if so used.

CHECK PARTITION and REPAIR PARTITION operations fail when the partition to be checked or repaired contains any duplicate key errors.

For more information about these statements, see Section 23.3.4, “Maintenance of Partitions”.

13.1.9.2 ALTER TABLE and Generated Columns

ALTER TABLE operations permitted for generated columns are ADD, MODIFY, and CHANGE.

- Generated columns can be added.

```
CREATE TABLE t1 (c1 INT);
ALTER TABLE t1 ADD COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED;
```

- The data type and expression of generated columns can be modified.

```
CREATE TABLE t1 (c1 INT, c2 INT GENERATED ALWAYS AS (c1 + 1) STORED);
ALTER TABLE t1 MODIFY COLUMN c2 TINYINT GENERATED ALWAYS AS (c1 + 5) STORED;
```

- Generated columns can be renamed or dropped, if no other column refers to them.

```
CREATE TABLE t1 (c1 INT, c2 INT GENERATED ALWAYS AS (c1 + 1) STORED);
ALTER TABLE t1 CHANGE c2 c3 INT GENERATED ALWAYS AS (c1 + 1) STORED;
```

```
ALTER TABLE t1 DROP COLUMN c3;
```

- Virtual generated columns cannot be altered to stored generated columns, or vice versa. To work around this, drop the column, then add it with the new definition.

```
CREATE TABLE t1 (c1 INT, c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL);
ALTER TABLE t1 DROP COLUMN c2;
ALTER TABLE t1 ADD COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED;
```

- Nongenerated columns can be altered to stored but not virtual generated columns.

```
CREATE TABLE t1 (c1 INT, c2 INT);
ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED;
```

- Stored but not virtual generated columns can be altered to nongenerated columns. The stored generated values become the values of the nongenerated column.

```
CREATE TABLE t1 (c1 INT, c2 INT GENERATED ALWAYS AS (c1 + 1) STORED);
ALTER TABLE t1 MODIFY COLUMN c2 INT;
```

- `ADD COLUMN` is not an in-place operation for stored columns (done without using a temporary table) because the expression must be evaluated by the server. For stored columns, indexing changes are done in place, and expression changes are not done in place. Changes to column comments are done in place.

- For non-partitioned tables, `ADD COLUMN` and `DROP COLUMN` are in-place operations for virtual columns. However, adding or dropping a virtual column cannot be performed in place in combination with other `ALTER TABLE` operations.

  For partitioned tables, `ADD COLUMN` and `DROP COLUMN` are not in-place operations for virtual columns.

- `InnoDB` supports secondary indexes on virtual generated columns. Adding or dropping a secondary index on a virtual generated column is an in-place operation. For more information, see Section 13.1.20.10, “Secondary Indexes and Generated Columns”.

- When a `VIRTUAL` generated column is added to a table or modified, it is not ensured that data being calculated by the generated column expression will not be out of range for the column. This can lead to inconsistent data being returned and unexpectedly failed statements. To permit control over whether validation occurs for such columns, `ALTER TABLE` supports `WITHOUT VALIDATION` and `WITH VALIDATION` clauses:

  - With `WITHOUT VALIDATION` (the default if neither clause is specified), an in-place operation is performed (if possible), data integrity is not checked, and the statement finishes more quickly. However, later reads from the table might report warnings or errors for the column if values are out of range.

  - With `WITH VALIDATION`, `ALTER TABLE` copies the table. If an out-of-range or any other error occurs, the statement fails. Because a table copy is performed, the statement takes longer.

  `WITHOUT VALIDATION` and `WITH VALIDATION` are permitted only with `ADD COLUMN`, `CHANGE COLUMN`, and `MODIFY COLUMN` operations. Otherwise, an `ER_WRONG_USAGE` error occurs.

- If expression evaluation causes truncation or provides incorrect input to a function, the `ALTER TABLE` statement terminates with an error and the DDL operation is rejected.

- An `ALTER TABLE` statement that changes the default value of a column *`col_name`* may also change the value of a generated column expression that refers to the column using *`col_name`*,

which may change the value of a generated column expression that refers to the column using `DEFAULT(col_name)`. For this reason, `ALTER TABLE` operations that change the definition of a column cause a table rebuild if any generated column expression uses `DEFAULT()`.

13.1.9.3 ALTER TABLE Examples

Begin with a table `t1` created as shown here:

```
CREATE TABLE t1 (a INTEGER, b CHAR(10));
```

To rename the table from `t1` to `t2`:

```
ALTER TABLE t1 RENAME t2;
```

To change column `a` from `INTEGER` to `TINYINT NOT NULL` (leaving the name the same), and to change column `b` from `CHAR(10)` to `CHAR(20)` as well as renaming it from `b` to `c`:

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new `TIMESTAMP` column named `d`:

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

To add an index on column `d` and a `UNIQUE` index on column `a`:

```
ALTER TABLE t2 ADD INDEX (d), ADD UNIQUE (a);
```

To remove column `c`:

```
ALTER TABLE t2 DROP COLUMN c;
```

To add a new `AUTO_INCREMENT` integer column named `c`:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
  ADD PRIMARY KEY (c);
```

We indexed `c` (as a `PRIMARY KEY`) because `AUTO_INCREMENT` columns must be indexed, and we declare `c` as `NOT NULL` because primary key columns cannot be `NULL`.

For `NDB` tables, it is also possible to change the storage type used for a table or column. For example, consider an `NDB` table created as shown here:

```
mysql> CREATE TABLE t1 (c1 INT) TABLESPACE ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.27 sec)
```

To convert this table to disk-based storage, you can use the following `ALTER TABLE` statement:

```
mysql> ALTER TABLE t1 TABLESPACE ts_1 STORAGE DISK;
Query OK, 0 rows affected (2.99 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
```

```
Create Table: CREATE TABLE `t1` (
  `c1` int(11) DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */
ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

It is not necessary that the tablespace was referenced when the table was originally created; however, the tablespace must be referenced by the `ALTER TABLE`:

```
mysql> CREATE TABLE t2 (c1 INT) ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.00 sec)

mysql> ALTER TABLE t2 STORAGE DISK;
ERROR 1005 (HY000): Can't create table 'c.#sql-1750_3' (errno: 140)
mysql> ALTER TABLE t2 TABLESPACE ts_1 STORAGE DISK;
Query OK, 0 rows affected (3.42 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> SHOW CREATE TABLE t2\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t2` (
  `c1` int(11) DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */
ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

To change the storage type of an individual column, you can use `ALTER TABLE ... MODIFY [COLUMN]`. For example, suppose you create an NDB Cluster Disk Data table with two columns, using this `CREATE TABLE` statement:

```
mysql> CREATE TABLE t3 (c1 INT, c2 INT)
    ->     TABLESPACE ts_1 STORAGE DISK ENGINE NDB;
Query OK, 0 rows affected (1.34 sec)
```

To change column `c2` from disk-based to in-memory storage, include a STORAGE MEMORY clause in the column definition used by the ALTER TABLE statement, as shown here:

```
mysql> ALTER TABLE t3 MODIFY c2 INT STORAGE MEMORY;
Query OK, 0 rows affected (3.14 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can make an in-memory column into a disk-based column by using `STORAGE DISK` in a similar fashion.

Column `c1` uses disk-based storage, since this is the default for the table (determined by the table-level `STORAGE DISK` clause in the `CREATE TABLE` statement). However, column `c2` uses in-memory storage, as can be seen here in the output of SHOW `CREATE TABLE`:

```
mysql> SHOW CREATE TABLE t3\G
*************************** 1. row ***************************
       Table: t3
Create Table: CREATE TABLE `t3` (
  `c1` int(11) DEFAULT NULL,
  `c2` int(11) /*!50120 STORAGE MEMORY */ DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.02 sec)
```

When you add an `AUTO_INCREMENT` column, column values are filled in with sequence numbers automatically. For `MyISAM` tables, you can set the first sequence number by executing `SET INSERT_ID=value` before `ALTER TABLE` or by using the `AUTO_INCREMENT=value` table option.

With MyISAM tables, if you do not change the AUTO_INCREMENT column, the sequence number is not affected. If you drop an AUTO_INCREMENT column and then add another AUTO_INCREMENT column, the numbers are resequenced beginning with 1.

When replication is used, adding an AUTO_INCREMENT column to a table might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an AUTO_INCREMENT number. Assuming that you want to add an AUTO_INCREMENT column to the table t1, the following statements produce a new table t2 identical to t1 but with an AUTO_INCREMENT column:

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

This assumes that the table t1 has columns col1 and col2.

This set of statements will also produce a new table t2 identical to t1, with the addition of an AUTO_INCREMENT column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important

To guarantee the same ordering on both master and slave, *all* columns of t1 must be referenced in the ORDER BY clause.

Regardless of the method used to create and populate the copy having the AUTO_INCREMENT column, the final step is to drop the original table and then rename the copy:

```
DROP TABLE t1;
ALTER TABLE t2 RENAME t1;
```

13.1.10 ALTER TABLESPACE Syntax

```
ALTER [UNDO] TABLESPACE tablespace_name
  NDB only:
    {ADD|DROP} DATAFILE 'file_name'
    [INITIAL_SIZE [=] size]
    [WAIT]
  InnoDB and NDB:
    [RENAME TO tablespace_name]
  InnoDB only:
    [SET {ACTIVE|INACTIVE}]
    [ENCRYPTION [=] {'Y' | 'N'}]
  InnoDB and NDB:
    [ENGINE [=] engine_name]
```

This statement is used with NDB and InnoDB tablespaces. It can be used to add a new data file to, or to drop a data file from an NDB tablespace. It can also be used to rename an NDB Cluster Disk Data tablespace, rename an InnoDB general tablespace, encrypt an InnoDB general tablespace, or mark an InnoDB undo tablespace as active or inactive.

The UNDO keyword, introduced in MySQL 8.0.14, is used with the SET {ACTIVE|INACTIVE} clause to mark an InnoDB undo tablespace as active or inactive. For more information, see Section 15.6.3.4, “Undo Tablespaces”.

The ADD DATAFILE variant enables you to specify an initial size for an NDB Disk Data tablespace using an INITIAL_SIZE clause, where *size* is measured in bytes; the default value is 134217728 (128 MB). You may optionally follow *size* with a one-letter abbreviation for an order of magnitude, similar to those used in my.cnf. Generally, this is one of the letters M (megabytes) or G (gigabytes).

On 32-bit systems, the maximum supported value for INITIAL_SIZE is 4294967296 (4 GB). (Bug #29186)

INITIAL_SIZE is rounded, explicitly, as for CREATE TABLESPACE.

Once a data file has been created, its size cannot be changed; however, you can add more data files to an NDB tablespace using additional ALTER TABLESPACE ... ADD DATAFILE statements.

When ALTER TABLESPACE ... ADD DATAFILE is used with ENGINE = NDB, a data file is created on each Cluster data node, but only one row is generated in the INFORMATION_SCHEMA.FILES table. See the description of this table, as well as Section 22.5.13.1, “NDB Cluster Disk Data Objects”, for more information. ADD DATAFILE is not supported with InnoDB tablespaces.

Using DROP DATAFILE with ALTER TABLESPACE drops the data file '*file_name*' from an NDB tablespace. You cannot drop a data file from a tablespace which is in use by any table; in other words, the data file must be empty (no extents used). See Section 22.5.13.1, “NDB Cluster Disk Data Objects”. In addition, any data file to be dropped must previously have been added to the tablespace with CREATE TABLESPACE or ALTER TABLESPACE. DROP DATAFILE is not supported with InnoDB tablespaces.

WAIT is parsed but otherwise ignored. It is intended for future expansion.

The ENGINE clause, which specifies the storage engine used by the tablespace, is deprecated and will be removed in a future release. The tablespace storage engine is known by the data dictionary, making the ENGINE clause obsolete. If the storage engine is specified, it must match the tablespace storage engine defined in the data dictionary. The only values for *engine_name* compatible with NDB tablespaces are NDB and NDBCLUSTER.

RENAME TO operations are implicitly performed in autocommit mode, regardless of the autocommit setting.

A RENAME TO operation cannot be performed while LOCK TABLES or FLUSH TABLES WITH READ LOCK is in effect for tables that reside in the tablespace.

Exclusive metadata locks are taken on tables that reside in a general tablespace while the tablespace is renamed, which prevents concurrent DDL. Concurrent DML is supported.

The CREATE TABLESPACE privilege is required to rename an InnoDB general tablespace.

The ENCRYPTION clause enables or disables page-level data encryption for an InnoDB general tablespace or the mysql system tablespace. Encryption support for general tablespaces was introduced in MySQL 8.0.13. Encryption support for the mysql system tablespace was introduced in MySQL 8.0.16.

A keyring plugin must be installed and configured before encryption can be enabled.

As of MySQL 8.0.16, if the table_encryption_privilege_check variable is enabled, the TABLE_ENCRYPTION_ADMIN privilege is required to alter a general tablespace with an ENCRYPTION clause setting that differs from the default_table_encryption setting.

Enabling encryption for a general tablespace fails if any table in the tablespace belongs to a schema defined with DEFAULT ENCRYPTION='N'. Similarly, disabling encryption fails if any table in the general tablespace belongs to a schema defined with DEFAULT ENCRYPTION='Y'. The DEFAULT ENCRYPTION schema option was introduced in MySQL 8.0.16.

If an ALTER TABLESPACE statement executed on a general tablespace does not include an ENCRYPTION clause, the tablespace retains its current encryption status, regardless of the default_table_encryption setting.

When a general tablespace or the mysql system tablespace is encrypted, all tables residing in the tablespace are encrypted. Likewise, a table created in an encrypted tablespace is encrypted.

The INPLACE algorithm is used when altering the ENCRYPTION attribute of a general tablespace or the mysql system tablespace. The INPLACE algorithm permits concurrent DML on tables that reside in the tablespace. Concurrent DDL is blocked.

For more information, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”.

13.1.11 ALTER VIEW Syntax

```
ALTER
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = user]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement changes the definition of a view, which must exist. The syntax is similar to that for CREATE VIEW see Section 13.1.23, “CREATE VIEW Syntax”). This statement requires the CREATE VIEW and DROP privileges for the view, and some privilege for each column referred to in the SELECT statement. ALTER VIEW is permitted only to the definer or users with the SET_USER_ID or SUPER privilege.

13.1.12 CREATE DATABASE Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | DEFAULT ENCRYPTION [=] {'Y' | 'N'}
```

CREATE DATABASE creates a database with the given name. To use this statement, you need the CREATE privilege for the database. CREATE SCHEMA is a synonym for CREATE DATABASE.

An error occurs if the database exists and you did not specify IF NOT EXISTS.

CREATE DATABASE is not permitted within a session that has an active LOCK TABLES statement.

create_specification options specify database characteristics. Database characteristics are stored in the data dictionary.

- The CHARACTER SET clause specifies the default database character set. The COLLATE clause specifies the default database collation. Chapter 10, *Character Sets, Collations, Unicode*, discusses character set and collation names.
- The DEFAULT ENCRYPTION clause, introduced in MySQL 8.0.16, defines the default database encryption, which is inherited by tables created in the database. If the DEFAULT ENCRYPTION clause is not specified, the default_table_encryption setting defines the default database encryption. If the table_encryption_privilege_check variable is enabled, the TABLE_ENCRYPTION_ADMIN privilege is required to specify a default encryption setting that differs from

the `default_table_encryption` setting. For more information, see Defining an Encryption Default for Schemas and General Tablespaces.

A database in MySQL is implemented as a directory containing files that correspond to tables in the database. Because there are no tables in a database when it is initially created, the `CREATE DATABASE` statement creates only a directory under the MySQL data directory. Rules for permissible database names are given in Section 9.2, “Schema Object Names”. If a database name contains special characters, the name for the database directory contains encoded versions of those characters as described in Section 9.2.4, “Mapping of Identifiers to File Names”.

Creating a database directory by manually creating a directory under the data directory (for example, with `mkdir`) is unsupported in MySQL 8.0.

MySQL has no limit on the number of databases. The underlying file system may have a limit on the number of directories.

You can also use the `mysqladmin` program to create databases. See Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”.

13.1.13 CREATE EVENT Syntax

```
CREATE
    [DEFINER = user]
    EVENT
    [IF NOT EXISTS]
    event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    DO event_body;

schedule:
    AT timestamp [+ INTERVAL interval] ...
  | EVERY interval
    [STARTS timestamp [+ INTERVAL interval] ...]
    [ENDS timestamp [+ INTERVAL interval] ...]

interval:
    quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
              WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
              DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

This statement creates and schedules a new event. The event will not run unless the Event Scheduler is enabled. For information about checking Event Scheduler status and enabling it if necessary, see Section 24.4.2, “Event Scheduler Configuration”.

`CREATE EVENT` requires the `EVENT` privilege for the schema in which the event is to be created. If the `DEFINER` clause is present, the privileges required depend on the *`user`* value, as discussed in Section 24.6, “Stored Object Access Control”.

The minimum requirements for a valid `CREATE EVENT` statement are as follows:

- The keywords `CREATE EVENT` plus an event name, which uniquely identifies the event in a database schema.
- An `ON SCHEDULE` clause, which determines when and how often the event executes.
- A `DO` clause, which contains the SQL statement to be executed by an event.

This is an example of a minimal `CREATE EVENT` statement:

```
CREATE EVENT myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

The previous statement creates an event named `myevent`. This event executes once—one hour following its creation—by running an SQL statement that increments the value of the `myschema.mytable` table's `mycol` column by 1.

The *`event_name`* must be a valid MySQL identifier with a maximum length of 64 characters. Event names are not case-sensitive, so you cannot have two events named `myevent` and `MyEvent` in the same schema. In general, the rules governing event names are the same as those for names of stored routines. See Section 9.2, “Schema Object Names”.

An event is associated with a schema. If no schema is indicated as part of *`event_name`*, the default (current) schema is assumed. To create an event in a specific schema, qualify the event name with a schema using *`schema_name.event_name`* syntax.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at event execution time. If the `DEFINER` clause is present, the *`user`* value should be a MySQL account specified as *`'user_name'@'host_name'`*, `CURRENT_USER`, or `CURRENT_USER()`. The permitted *`user`* values depend on the privileges you hold, as discussed in Section 24.6, “Stored Object Access Control”. Also see that section for additional information about event security.

If the `DEFINER` clause is omitted, the default definer is the user who executes the `CREATE EVENT` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

Within an event body, the `CURRENT_USER` function returns the account used to check privileges at event execution time, which is the `DEFINER` user. For information about user auditing within events, see Section 6.2.22, “SQL-Based Account Activity Auditing”.

`IF NOT EXISTS` has the same meaning for `CREATE EVENT` as for `CREATE TABLE`: If an event named *`event_name`* already exists in the same schema, no action is taken, and no error results. (However, a warning is generated in such cases.)

The `ON SCHEDULE` clause determines when, how often, and for how long the *`event_body`* defined for the event repeats. This clause takes one of two forms:

- `AT` *`timestamp`* is used for a one-time event. It specifies that the event executes one time only at the date and time given by *`timestamp`*, which must include both the date and time, or must be an expression that resolves to a datetime value. You may use a value of either the `DATETIME` or `TIMESTAMP` type for this purpose. If the date is in the past, a warning occurs, as shown here:

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2006-02-10 23:59:01 |
+---------------------+
1 row in set (0.04 sec)

mysql> CREATE EVENT e_totals
    ->     ON SCHEDULE AT '2006-02-10 23:59:00'
    ->     DO INSERT INTO test.totals VALUES (NOW());
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
```

```
   Code: 1588
Message: Event execution time is in the past and ON COMPLETION NOT
         PRESERVE is set. The event was dropped immediately after
         creation.
```

`CREATE EVENT` statements which are themselves invalid—for whatever reason—fail with an error.

You may use `CURRENT_TIMESTAMP` to specify the current date and time. In such a case, the event acts as soon as it is created.

To create an event which occurs at some point in the future relative to the current date and time—such as that expressed by the phrase “three weeks from now”—you can use the optional clause `+ INTERVAL` *`interval`*. The *`interval`* portion consists of two parts, a quantity and a unit of time, and follows the syntax rules described in Temporal Intervals, except that you cannot use any units keywords that involving microseconds when defining an event. With some interval types, complex time units may be used. For example, “two minutes and ten seconds” can be expressed as `+ INTERVAL '2:10' MINUTE_SECOND`.

You can also combine intervals. For example, `AT CURRENT_TIMESTAMP + INTERVAL 3 WEEK + INTERVAL 2 DAY` is equivalent to “three weeks and two days from now”. Each portion of such a clause must begin with `+ INTERVAL`.

- To repeat actions at a regular interval, use an `EVERY` clause. The `EVERY` keyword is followed by an *`interval`* as described in the previous discussion of the `AT` keyword. (`+ INTERVAL` is *not* used with `EVERY`.) For example, `EVERY 6 WEEK` means “every six weeks”.

  Although `+ INTERVAL` clauses are not permitted in an `EVERY` clause, you can use the same complex time units permitted in a `+ INTERVAL`.

  An `EVERY` clause may contain an optional `STARTS` clause. `STARTS` is followed by a *`timestamp`* value that indicates when the action should begin repeating, and may also use `+ INTERVAL` *`interval`* to specify an amount of time “from now”. For example, `EVERY 3 MONTH STARTS CURRENT_TIMESTAMP + INTERVAL 1 WEEK` means “every three months, beginning one week from now”. Similarly, you can express “every two weeks, beginning six hours and fifteen minutes from now” as `EVERY 2 WEEK STARTS CURRENT_TIMESTAMP + INTERVAL '6:15' HOUR_MINUTE`. Not specifying `STARTS` is the same as using `STARTS CURRENT_TIMESTAMP`—that is, the action specified for the event begins repeating immediately upon creation of the event.

  An `EVERY` clause may contain an optional `ENDS` clause. The `ENDS` keyword is followed by a *`timestamp`* value that tells MySQL when the event should stop repeating. You may also use `+ INTERVAL` *`interval`* with `ENDS`; for instance, `EVERY 12 HOUR STARTS CURRENT_TIMESTAMP + INTERVAL 30 MINUTE ENDS CURRENT_TIMESTAMP + INTERVAL 4 WEEK` is equivalent to “every twelve hours, beginning thirty minutes from now, and ending four weeks from now”. Not using `ENDS` means that the event continues executing indefinitely.

  `ENDS` supports the same syntax for complex time units as `STARTS` does.

  You may use `STARTS`, `ENDS`, both, or neither in an `EVERY` clause.

  If a repeating event does not terminate within its scheduling interval, the result may be multiple instances of the event executing simultaneously. If this is undesirable, you should institute a mechanism to prevent simultaneous instances. For example, you could use the `GET_LOCK()` function, or row or table locking.

The `ON SCHEDULE` clause may use expressions involving built-in MySQL functions and user variables to obtain any of the *`timestamp`* or *`interval`* values which it contains. You may not use stored functions or user-defined functions in such expressions, nor may you use any table references; however, you may use `SELECT FROM DUAL`. This is true for both `CREATE EVENT` and `ALTER EVENT` statements. References

to stored functions, user-defined functions, and tables in such cases are specifically not permitted, and fail with an error (see Bug #22830).

Times in the `ON SCHEDULE` clause are interpreted using the current session `time_zone` value. This becomes the event time zone; that is, the time zone that is used for event scheduling and is in effect within the event as it executes. These times are converted to UTC and stored along with the event time zone internally. This enables event execution to proceed as defined regardless of any subsequent changes to the server time zone or daylight saving time effects. For additional information about representation of event times, see Section 24.4.4, “Event Metadata”. See also Section 13.7.7.18, “SHOW EVENTS Syntax”, and Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”.

Normally, once an event has expired, it is immediately dropped. You can override this behavior by specifying `ON COMPLETION PRESERVE`. Using `ON COMPLETION NOT PRESERVE` merely makes the default nonpersistent behavior explicit.

You can create an event but prevent it from being active using the `DISABLE` keyword. Alternatively, you can use `ENABLE` to make explicit the default status, which is active. This is most useful in conjunction with `ALTER EVENT` (see Section 13.1.3, “ALTER EVENT Syntax”).

A third value may also appear in place of `ENABLE` or `DISABLE`; `DISABLE ON SLAVE` is set for the status of an event on a replication slave to indicate that the event was created on the master and replicated to the slave, but is not executed on the slave. See Section 17.5.1.16, “Replication of Invoked Features”.

You may supply a comment for an event using a `COMMENT` clause. *`comment`* may be any string of up to 64 characters that you wish to use for describing the event. The comment text, being a string literal, must be surrounded by quotation marks.

The `DO` clause specifies an action carried by the event, and consists of an SQL statement. Nearly any valid MySQL statement that can be used in a stored routine can also be used as the action statement for a scheduled event. (See Section 24.8, “Restrictions on Stored Programs”.) For example, the following event `e_hourly` deletes all rows from the `sessions` table once per hour, where this table is part of the `site_activity` schema:

```
CREATE EVENT e_hourly
    ON SCHEDULE
      EVERY 1 HOUR
    COMMENT 'Clears out sessions table each hour.'
    DO
      DELETE FROM site_activity.sessions;
```

MySQL stores the `sql_mode` system variable setting in effect when an event is created or altered, and always executes the event with this setting in force, *regardless of the current server SQL mode when the event begins executing*.

A `CREATE EVENT` statement that contains an `ALTER EVENT` statement in its `DO` clause appears to succeed; however, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

Note

Statements such as `SELECT` or `SHOW` that merely return a result set have no effect when used in an event; the output from these is not sent to the MySQL Monitor, nor is it stored anywhere. However, you can use statements such as `SELECT ... INTO` and `INSERT INTO ... SELECT` that store a result. (See the next example in this section for an instance of the latter.)

The schema to which an event belongs is the default schema for table references in the `DO` clause. Any references to tables in other schemas must be qualified with the proper schema name.

As with stored routines, you can use compound-statement syntax in the `DO` clause by using the `BEGIN` and `END` keywords, as shown here:

```
delimiter |

CREATE EVENT e_daily
    ON SCHEDULE
      EVERY 1 DAY
    COMMENT 'Saves total number of sessions then clears the table each day'
    DO
      BEGIN
        INSERT INTO site_activity.totals (time, total)
          SELECT CURRENT_TIMESTAMP, COUNT(*)
            FROM site_activity.sessions;
        DELETE FROM site_activity.sessions;
      END |

delimiter ;
```

This example uses the `delimiter` command to change the statement delimiter. See Section 24.1, “Defining Stored Programs”.

More complex compound statements, such as those used in stored routines, are possible in an event. This example uses local variables, an error handler, and a flow control construct:

```
delimiter |

CREATE EVENT e
    ON SCHEDULE
      EVERY 5 SECOND
    DO
      BEGIN
        DECLARE v INTEGER;
        DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END;

        SET v = 0;

        WHILE v < 5 DO
          INSERT INTO t1 VALUES (0);
          UPDATE t2 SET s1 = s1 + 1;
          SET v = v + 1;
        END WHILE;
    END |

delimiter ;
```

There is no way to pass parameters directly to or from events; however, it is possible to invoke a stored routine with parameters within an event:

```
CREATE EVENT e_call_myproc
    ON SCHEDULE
      AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
    DO CALL myproc(5, 27);
```

If an event's definer has privileges sufficient to set global system variables (see Section 5.1.9.1, “System Variable Privileges”), the event can read and write global variables. As granting such privileges entails a potential for abuse, extreme care must be taken in doing so.

Generally, any statements that are valid in stored routines may be used for action statements executed by events. For more information about statements permissible within stored routines, see Section 24.2.1,

"Stored Routine Syntax". You can create an event as part of a stored routine, but an event cannot be created by another event.

13.1.14 CREATE FUNCTION Syntax

The `CREATE FUNCTION` statement is used to create stored functions and user-defined functions (UDFs):

- For information about creating stored functions, see Section 13.1.17, "CREATE PROCEDURE and CREATE FUNCTION Syntax".
- For information about creating user-defined functions, see Section 13.7.4.1, "CREATE FUNCTION Syntax for User-Defined Functions".

13.1.15 CREATE INDEX Syntax

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (key_part,...)
    [index_option]
    [algorithm_option | lock_option] ...

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | {VISIBLE | INVISIBLE}

index_type:
    USING {BTREE | HASH}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

Normally, you create all indexes on a table at the time the table itself is created with `CREATE TABLE`. See Section 13.1.20, "CREATE TABLE Syntax". This guideline is especially important for `InnoDB` tables, where the primary key determines the physical layout of rows in the data file. `CREATE INDEX` enables you to add indexes to existing tables.

`CREATE INDEX` is mapped to an `ALTER TABLE` statement to create indexes. See Section 13.1.9, "ALTER TABLE Syntax". `CREATE INDEX` cannot be used to create a `PRIMARY KEY`; use `ALTER TABLE` instead. For more information about indexes, see Section 8.3.1, "How MySQL Uses Indexes".

`InnoDB` supports secondary indexes on virtual columns. For more information, see Section 13.1.20.10, "Secondary Indexes and Generated Columns".

When the `innodb_stats_persistent` setting is enabled, run the `ANALYZE TABLE` statement for an `InnoDB` table after creating an index on that table.

Beginning with MySQL 8.0.17, the *`expr`* for a *`key_part`* specification can take the form `(CAST` *`json_path`* `AS` *`type`* `ARRAY)` to create a multi-valued index on a `JSON` column. See Multi-Valued Indexes.

An index specification of the form `(`*`key_part1, key_part2, ...`*`)` creates an index with multiple key parts. Index key values are formed by concatenating the values of the given key parts. For example

`(col1, col2, col3)` specifies a multiple-column index with index keys consisting of values from `col1`, `col2`, and `col3`.

A *`key_part`* specification can end with `ASC` or `DESC` to specify whether index values are stored in ascending or descending order. The default is ascending if no order specifier is given. `ASC` and `DESC` are not permitted for `HASH` indexes. `ASC` and `DESC` are also not supported for multi-valued indexes. As of MySQL 8.0.12, `ASC` and `DESC` are not permitted for `SPATIAL` indexes.

The following sections describe different aspects of the `CREATE INDEX` statement:

- Column Prefix Key Parts
- Functional Key Parts
- Unique Indexes
- Full-Text Indexes
- Multi-Valued Indexes
- Spatial Indexes
- Index Options
- Table Copying and Locking Options

Column Prefix Key Parts

For string columns, indexes can be created that use only the leading part of column values, using *`col_name(length)`* syntax to specify an index prefix length:

- Prefixes can be specified for `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` key parts.
- Prefixes *must* be specified for `BLOB` and `TEXT` key parts. Additionally, `BLOB` and `TEXT` columns can be indexed only for `InnoDB`, `MyISAM`, and `BLACKHOLE` tables.
- Prefix *limits* are measured in bytes. However, prefix *lengths* for index specifications in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements are interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

  Prefix support and lengths of prefixes (where supported) are storage engine dependent. For example, a prefix can be up to 767 bytes long for `InnoDB` tables that use the `REDUNDANT` or `COMPACT` row format. The prefix length limit is 3072 bytes for `InnoDB` tables that use the `DYNAMIC` or `COMPRESSED` row format. For `MyISAM` tables, the prefix length limit is 1000 bytes. The `NDB` storage engine does not support prefixes (see Section 22.1.7.6, “Unsupported or Missing Features in NDB Cluster”).

If a specified index prefix exceeds the maximum column data type size, `CREATE INDEX` handles the index as follows:

- For a nonunique index, either an error occurs (if strict SQL mode is enabled), or the index length is reduced to lie within the maximum column data type size and a warning is produced (if strict SQL mode is not enabled).
- For a unique index, an error occurs regardless of SQL mode because reducing the index length might enable insertion of nonunique entries that do not meet the specified uniqueness requirement.

The statement shown here creates an index using the first 10 characters of the `name` column (assuming that `name` has a nonbinary string type):

```
CREATE INDEX part_of_name ON customer (name(10));
```

If names in the column usually differ in the first 10 characters, lookups performed using this index should not be much slower than using an index created from the entire `name` column. Also, using column prefixes for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up `INSERT` operations.

Functional Key Parts

A “normal” index indexes column values or prefixes of column values. For example, in the following table, the index entry for a given `t1` row includes the full `col1` value and a prefix of the `col2` value consisting of its first 10 characters:

```
CREATE TABLE t1 (
  col1 VARCHAR(10),
  col2 VARCHAR(20),
  INDEX (col1, col2(10))
);
```

MySQL 8.0.13 and higher supports functional key parts that index expression values rather than column or column prefix values. Use of functional key parts enables indexing of values not stored directly in the table. Examples:

```
CREATE TABLE t1 (col1 INT, col2 INT, INDEX func_index ((ABS(col1))));
CREATE INDEX idx1 ON t1 ((col1 + col2));
CREATE INDEX idx2 ON t1 ((col1 + col2), (col1 - col2), col1);
ALTER TABLE t1 ADD INDEX ((col1 * 40) DESC);
```

An index with multiple key parts can mix nonfunctional and functional key parts.

`ASC` and `DESC` are supported for functional key parts.

Functional key parts must adhere to the following rules. An error occurs if a key part definition contains disallowed constructs.

- In index definitions, enclose expressions within parentheses to distinguish them from columns or column prefixes. For example, this is permitted; the expressions are enclosed within parentheses:

  ```
  INDEX ((col1 + col2), (col3 - col4))
  ```

  This produces an error; the expressions are not enclosed within parentheses:

  ```
  INDEX (col1 + col2, col3 - col4)
  ```

- A functional key part cannot consist solely of a column name. For example, this is not permitted:

  ```
  INDEX ((col1), (col2))
  ```

  Instead, write the key parts as nonfunctional key parts, without parentheses:

  ```
  INDEX (col1, col2)
  ```

- A functional key part expression cannot refer to column prefixes. For a workaround, see the discussion of `SUBSTRING()` and `CAST()` later in this section.
- Functional key parts are not permitted in foreign key specifications.

For `CREATE TABLE ... LIKE`, the destination table preserves functional key parts from the original table.

Functional indexes are implemented as hidden virtual generated columns, which has these implications:

- Each functional key part counts against the limit on total number of table columns; see Section 8.4.7, “Limits on Table Column Count and Row Size”.
- Functional key parts inherit all restrictions that apply to generated columns. Examples:
  - Only functions permitted for generated columns are permitted for functional key parts.
  - Subqueries, parameters, variables, stored functions, and user-defined functions are not permitted.

  For more information about applicable restrictions, see Section 13.1.20.9, “CREATE TABLE and Generated Columns”, and Section 13.1.9.2, “ALTER TABLE and Generated Columns”.
- The virtual generated column itself requires no storage. The index itself takes up storage space as any other index.

`UNIQUE` is supported for indexes that include functional key parts. However, primary keys cannot include functional key parts. A primary key requires the generated column to be stored, but functional key parts are implemented as virtual generated columns, not stored generated columns.

`SPATIAL` and `FULLTEXT` indexes cannot have functional key parts.

If a table contains no primary key, `InnoDB` automatically promotes the first `UNIQUE NOT NULL` index to the primary key. This is not supported for `UNIQUE NOT NULL` indexes that have functional key parts.

Nonfunctional indexes raise a warning if there are duplicate indexes. Indexes that contain functional key parts do not have this feature.

To remove a column that is referenced by a functional key part, the index must be removed first. Otherwise, an error occurs.

Although nonfunctional key parts support a prefix length specification, this is not possible for functional key parts. The solution is to use `SUBSTRING()` (or `CAST()`, as described later in this section). For a functional key part containing the `SUBSTRING()` function to be used in a query, the `WHERE` clause must contain `SUBSTRING()` with the same arguments. In the following example, only the second `SELECT` is able to use the index because that is the only query in which the arguments to `SUBSTRING()` match the index specification:

```
CREATE TABLE tbl (
  col1 LONGTEXT,
  INDEX idx1 ((SUBSTRING(col1, 1, 10)))
);
SELECT * FROM tbl WHERE SUBSTRING(col1, 1, 9) = '123456789';
SELECT * FROM tbl WHERE SUBSTRING(col1, 1, 10) = '1234567890';
```

Functional key parts enable indexing of values that cannot be indexed otherwise, such as `JSON` values. However, this must be done correctly to achieve the desired effect. For example, this syntax does not work:

```
CREATE TABLE employees (
  data JSON,
  INDEX ((data->>'$.name'))
);
```

The syntax fails because:

- The `->>` operator translates into `JSON_UNQUOTE(JSON_EXTRACT(...))`.
- `JSON_UNQUOTE()` returns a value with a data type of `LONGTEXT`, and the hidden generated column thus is assigned the same data type.
- MySQL cannot index `LONGTEXT` columns specified without a prefix length on the key part, and prefix lengths are not permitted in functional key parts.

To index the `JSON` column, you could try using the `CAST()` function as follows:

```
CREATE TABLE employees (
  data JSON,
  INDEX ((CAST(data->>'$.name' AS CHAR(30))))
);
```

The hidden generated column is assigned the `VARCHAR(30)` data type, which can be indexed. But this approach produces a new issue when trying to use the index:

- `CAST()` returns a string with the collation `utf8mb4_0900_ai_ci` (the server default collation).
- `JSON_UNQUOTE()` returns a string with the collation `utf8mb4_bin` (hard coded).

As a result, there is a collation mismatch between the indexed expression in the preceding table definition and the `WHERE` clause expression in the following query:

```
SELECT * FROM employees WHERE data->>'$.name' = 'James';
```

The index is not used because the expressions in the query and the index differ. To support this kind of scenario for functional key parts, the optimizer automatically strips `CAST()` when looking for an index to use, but *only* if the collation of the indexed expression matches that of the query expression. For an index with a functional key part to be used, either of the following two solutions work (although they differ somewhat in effect):

- Solution 1. Assign the indexed expression the same collation as `JSON_UNQUOTE()`:

  ```
  CREATE TABLE employees (
    data JSON,
    INDEX idx ((CAST(data->>"$.name" AS CHAR(30)) COLLATE utf8mb4_bin))
  );
  INSERT INTO employees VALUES
    ('{ "name": "james", "salary": 9000 }'),
    ('{ "name": "James", "salary": 10000 }'),
    ('{ "name": "Mary", "salary": 12000 }'),
    ('{ "name": "Peter", "salary": 8000 }');
  SELECT * FROM employees WHERE data->>'$.name' = 'James';
  ```

  The `->>` operator is the same as `JSON_UNQUOTE(JSON_EXTRACT(...))`, and `JSON_UNQUOTE()` returns a string with collation `utf8mb4_bin`. The comparison is thus case sensitive, and only one row matches:

  ```
  +-----------------------------------+
  | data                              |
  +-----------------------------------+
  | {"name": "James", "salary": 10000} |
  +-----------------------------------+
  ```

- Solution 2. Specify the full expression in the query:

  ```
  ```

```
CREATE TABLE employees (
  data JSON,
  INDEX idx ((CAST(data->>"$.name" AS CHAR(30))))
);
INSERT INTO employees VALUES
  ('{ "name": "james", "salary": 9000 }'),
  ('{ "name": "James", "salary": 10000 }'),
  ('{ "name": "Mary", "salary": 12000 }'),
  ('{ "name": "Peter", "salary": 8000 }');
SELECT * FROM employees WHERE CAST(data->>'$.name' AS CHAR(30)) = 'James';
```

`CAST()` returns a string with collation `utf8mb4_0900_ai_ci`, so the comparison case insensitive and two rows match:

```
+-----------------------------------+
| data                              |
+-----------------------------------+
| {"name": "james", "salary": 9000}  |
| {"name": "James", "salary": 10000} |
+-----------------------------------+
```

Be aware that although the optimizer supports automatically stripping `CAST()` with indexed generated columns, the following approach does not work because it produces a different result with and without an index (Bug#27337092):

```
mysql> CREATE TABLE employees (
         data JSON,
         generated_col VARCHAR(30) AS (CAST(data->>'$.name' AS CHAR(30)))
       );
Query OK, 0 rows affected, 1 warning (0.03 sec)

mysql> INSERT INTO employees (data)
       VALUES ('{"name": "james"}'), ('{"name": "James"}');
Query OK, 2 rows affected, 1 warning (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 1

mysql> SELECT * FROM employees WHERE data->>'$.name' = 'James';
+------------------+---------------+
| data             | generated_col |
+------------------+---------------+
| {"name": "James"} | James         |
+------------------+---------------+
1 row in set (0.00 sec)

mysql> ALTER TABLE employees ADD INDEX idx (generated_col);
Query OK, 0 rows affected, 1 warning (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> SELECT * FROM employees WHERE data->>'$.name' = 'James';
+------------------+---------------+
| data             | generated_col |
+------------------+---------------+
| {"name": "james"} | james         |
| {"name": "James"} | James         |
+------------------+---------------+
2 rows in set (0.01 sec)
```

Unique Indexes

A `UNIQUE` index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. If you specify a prefix value for a column in a `UNIQUE` index, the column values must be unique within the prefix length. A `UNIQUE` index permits multiple `NULL` values for columns that can contain `NULL`.

If a table has a `PRIMARY KEY` or `UNIQUE NOT NULL` index that consists of a single column that has an integer type, you can use `_rowid` to refer to the indexed column in `SELECT` statements, as follows:

- `_rowid` refers to the `PRIMARY KEY` column if there is a `PRIMARY KEY` consisting of a single integer column. If there is a `PRIMARY KEY` but it does not consist of a single integer column, `_rowid` cannot be used.
- Otherwise, `_rowid` refers to the column in the first `UNIQUE NOT NULL` index if that index consists of a single integer column. If the first `UNIQUE NOT NULL` index does not consist of a single integer column, `_rowid` cannot be used.

Full-Text Indexes

`FULLTEXT` indexes are supported only for `InnoDB` and `MyISAM` tables and can include only `CHAR`, `VARCHAR`, and `TEXT` columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 12.9, “Full-Text Search Functions”, for details of operation.

Multi-Valued Indexes

As of MySQL 8.0.17, `InnoDB` supports multi-valued indexes. A multi-valued index is a secondary index defined on a column that stores an array of values. A “normal” index has one index record for each data record (1:1). A multi-valued index can have multiple index records for a single data record (N:1). Multi-valued indexes are intended for indexing `JSON` arrays. For example, a multi-valued index defined on the array of zip codes in the following JSON document creates an index record for each zip code, with each index record referencing the same data record.

```
{
    "user":"Bob",
    "user_id":31,
    "zipcode":[94477,94536]
}
```

Creating multi-valued Indexes

You can create a multi-valued index in a `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX` statement. This requires using `CAST(... AS ... ARRAY)` in the index definition, which casts same-typed scalar values in a `JSON` array to an SQL data type array. A virtual column is then generated transparently with the values in the SQL data type array; finally, a functional index (also referred to as a virtual index) is created on the virtual column. It is the functional index defined on the virtual column of values from the SQL data type array that forms the multi-valued index.

The examples in the following list show the three different ways in which a multi-valued index `zips` can be created on an array `$.zipcode` on a `JSON` column `custinfo` in a table named `customers`. In each case, the JSON array is cast to an SQL data type array of `UNSIGNED` integer values.

- `CREATE TABLE` only:

```
CREATE TABLE customers (
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    custinfo JSON,
    INDEX zips( (CAST(custinfo->'$.zip' AS UNSIGNED ARRAY)) )
    );
```

- `CREATE TABLE` plus `ALTER TABLE`:

```
CREATE TABLE customers (
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    custinfo JSON
    );

ALTER TABLE customers
    ADD INDEX zips( (CAST(custinfo->'$.zip' AS UNSIGNED ARRAY)) );
```

- `CREATE TABLE` plus `CREATE INDEX`:

```
CREATE TABLE customers (
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    custinfo JSON
    );


CREATE INDEX zips ON customers ( (CAST(custinfo->'$.zip' AS UNSIGNED ARRAY)) );
```

A multi-valued index can also be defined as part of a composite index. This example shows a composite index that includes two single-valued parts (for the `id` and `modified` columns), and one multi-valued part (for the `custinfo` column):

```
CREATE TABLE customers (
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    custinfo JSON
    );

ALTER TABLE customers
    ADD INDEX comp(
        id, modified, (CAST(custinfo->'$.zipcode' AS UNSIGNED ARRAY))
    );
```

Only one multi-valued key part can be used in a composite index. The multi-valued key part may be used in any order relative to the other parts of the key. In other words, the `ALTER TABLE` statement just shown could have used `comp(id, (CAST(custinfo->'$.zipcode' AS UNSIGNED ARRAY), modified))` (or any other ordering) and still have been valid.

Using multi-valued Indexes

The optimizer uses a multi-valued index to fetch records when the following functions are specified in a `WHERE` clause:

- `MEMBER OF()`
- `JSON_CONTAINS()`
- `JSON_OVERLAPS()`

We can demonstrate this by creating and populating the `customers` table using the following `CREATE TABLE` and `INSERT` statements:

```
mysql> CREATE TABLE customers (
    ->     id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->     modified DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    ->     custinfo JSON
    ->     );
Query OK, 0 rows affected (0.51 sec)
```

```
mysql> INSERT INTO customers VALUES
    ->     (NULL, NOW(), '{"user":"Jack","user_id":37,"zipcode":[94582,94536]}'),
    ->     (NULL, NOW(), '{"user":"Jill","user_id":22,"zipcode":[94568,94507,94582]}'),
    ->     (NULL, NOW(), '{"user":"Bob","user_id":31,"zipcode":[94477,94536]}'),
    ->     (NULL, NOW(), '{"user":"Mary","user_id":72,"zipcode":[94536]}'),
    ->     (NULL, NOW(), '{"user":"Ted","user_id":56,"zipcode":[94507,94582]}');
Query OK, 5 rows affected (0.07 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

First we execute three queries on the `customers` table, one each using `MEMBER OF()`, `JSON_CONTAINS()`, and `JSON_OVERLAPS()`, with the result from each query shown here:

```
mysql> SELECT * FROM customers
    ->     WHERE 94507 MEMBER OF(custinfo->'$.zipcode');
+----+---------------------+--------------------------------------------------------------------------+
| id | modified            | custinfo                                                                 |
+----+---------------------+--------------------------------------------------------------------------+
|  2 | 2019-06-29 22:23:12 | {"user": "Jill", "user_id": 22, "zipcode": [94568, 94507, 94582]} |
|  3 | 2019-06-29 22:23:12 | {"user": "Bob", "user_id": 31, "zipcode": [94477, 94507]}         |
|  5 | 2019-06-29 22:23:12 | {"user": "Ted", "user_id": 56, "zipcode": [94507, 94582]}         |
+----+---------------------+--------------------------------------------------------------------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM customers
    ->     WHERE JSON_CONTAINS(custinfo->'$.zipcode', CAST('[94507,94582]' AS JSON));
+----+---------------------+--------------------------------------------------------------------------+
| id | modified            | custinfo                                                                 |
+----+---------------------+--------------------------------------------------------------------------+
|  2 | 2019-06-29 22:23:12 | {"user": "Jill", "user_id": 22, "zipcode": [94568, 94507, 94582]} |
|  5 | 2019-06-29 22:23:12 | {"user": "Ted", "user_id": 56, "zipcode": [94507, 94582]}         |
+----+---------------------+--------------------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM customers
    ->     WHERE JSON_OVERLAPS(custinfo->'$.zipcode', CAST('[94507,94582]' AS JSON));
+----+---------------------+--------------------------------------------------------------------------+
| id | modified            | custinfo                                                                 |
+----+---------------------+--------------------------------------------------------------------------+
|  1 | 2019-06-29 22:23:12 | {"user": "Jack", "user_id": 37, "zipcode": [94582, 94536]}        |
|  2 | 2019-06-29 22:23:12 | {"user": "Jill", "user_id": 22, "zipcode": [94568, 94507, 94582]} |
|  3 | 2019-06-29 22:23:12 | {"user": "Bob", "user_id": 31, "zipcode": [94477, 94507]}         |
|  5 | 2019-06-29 22:23:12 | {"user": "Ted", "user_id": 56, "zipcode": [94507, 94582]}         |
+----+---------------------+--------------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

Next, we run `EXPLAIN` on each of the previous three queries:

```
mysql> EXPLAIN SELECT * FROM customers
    ->     WHERE 94507 MEMBER OF(custinfo->'$.zipcode');
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+
|  1 | SIMPLE      | customers | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    5 |   100.00 |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM customers
    ->     WHERE JSON_CONTAINS(custinfo->'$.zipcode', CAST('[94507,94582]' AS JSON));
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+
|  1 | SIMPLE      | customers | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    5 |   100.00 |
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+----------+
1 row in set, 1 warning (0.00 sec)
```

```
mysql> EXPLAIN SELECT * FROM customers
    ->     WHERE JSON_OVERLAPS(custinfo->'$.zipcode', CAST('[94507,94582]' AS JSON));
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+-------
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref  | rows | filtere
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+-------
|  1 | SIMPLE      | customers | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    5 |   100.0
+----+-------------+-----------+------------+------+---------------+------+---------+------+------+-------
1 row in set, 1 warning (0.01 sec)
```

None of the three queries just shown are able to use any keys. To solve this problem, we can add a multi-valued index on the `zipcode` array in the `JSON` column (`custinfo`), like this:

```
mysql> ALTER TABLE customers
    ->     ADD INDEX zips( (CAST(custinfo->'$.zipcode' AS UNSIGNED ARRAY)) );
Query OK, 0 rows affected (0.47 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

When we run the previous `EXPLAIN` statements again, we can now observe that the queries can (and do) use the index `zips` that was just created:

```
mysql> EXPLAIN SELECT * FROM customers
    ->     WHERE 94507 MEMBER OF(custinfo->'$.zipcode');
+----+-------------+-----------+------------+------+---------------+------+---------+-------+------+-------
| id | select_type | table     | partitions | type | possible_keys | key  | key_len | ref   | rows | filtere
+----+-------------+-----------+------------+------+---------------+------+---------+-------+------+-------
|  1 | SIMPLE      | customers | NULL       | ref  | zips          | zips | 9       | const |    1 |   100.0
+----+-------------+-----------+------------+------+---------------+------+---------+-------+------+-------
1 row in set, 1 warning (0.00 sec)

mysql> EXPLAIN SELECT * FROM customers
    ->     WHERE JSON_OVERLAPS(custinfo->'$.zipcode', CAST('[94507,94582]' AS JSON));
+----+-------------+-----------+------------+-------+---------------+------+---------+------+------+-------
| id | select_type | table     | partitions | type  | possible_keys | key  | key_len | ref  | rows | filtere
+----+-------------+-----------+------------+-------+---------------+------+---------+------+------+-------
|  1 | SIMPLE      | customers | NULL       | range | zips          | zips | 9       | NULL |    6 |   100.0
+----+-------------+-----------+------------+-------+---------------+------+---------+------+------+-------
1 row in set, 1 warning (0.01 sec)

mysql> EXPLAIN SELECT * FROM customers
    ->     WHERE JSON_CONTAINS(custinfo->'$.zipcode', CAST('[94507,94582]' AS JSON));
+----+-------------+-----------+------------+-------+---------------+------+---------+------+------+-------
| id | select_type | table     | partitions | type  | possible_keys | key  | key_len | ref  | rows | filtere
+----+-------------+-----------+------------+-------+---------------+------+---------+------+------+-------
|  1 | SIMPLE      | customers | NULL       | range | zips          | zips | 9       | NULL |    6 |   100.0
+----+-------------+-----------+------------+-------+---------------+------+---------+------+------+-------
1 row in set, 1 warning (0.00 sec)
```

A multi-valued index can be defined as a unique key. If defined as a unique key, attempting to insert a value already present in the multi-valued index returns a duplicate key error. If duplicate values are already present, attempting to add a unique multi-valued index fails, as shown here:

```
mysql> ALTER TABLE customers DROP INDEX zips;
Query OK, 0 rows affected (0.55 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE customers
    ->     ADD UNIQUE INDEX zips((CAST(custinfo->'$.zipcode' AS UNSIGNED ARRAY)));
ERROR 1062 (23000): Duplicate entry '[94507, ' for key 'zips'
mysql> ALTER TABLE customers
    ->     ADD INDEX zips((CAST(custinfo->'$.zipcode' AS UNSIGNED ARRAY)));
Query OK, 0 rows affected (0.36 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Characteristics of Multi-Valued Indexes

Multi-valued indexes have the additional characteristics listed here:

- DML operations that affect multi-valued indexes are handled in the same way as DML operations that affect a normal index, with the only difference being that there may be more than one insert or update for a single clustered index record.
- Nullability and multi-valued indexes:
  - If multi-valued key part has an empty array, no entries are added to the index, and the data record is not accessible by an index scan.
  - If multi-valued key part generation returns a `NULL` value, a single entry containing `NULL` is added to the multi-valued index. If the key part is defined as `NOT NULL`, an error is reported.
  - If the typed array column is set to `NULL`, the storage engine stores single record containing `NULL` that points to the data record.
  - `JSON` null values are not permitted in indexed arrays. If any returned value is `NULL`, it is treated as a JSON null and an `Invalid JSON value` error is reported.
- Because multi-valued indexes are virtual indexes on virtual columns, they must adhere to the same rules as secondary indexes on virtual generated columns.
- Index records are not added for empty arrays.

Limitations and Restrictions on Multi-valued Indexes

Multi-valued indexes are subject to the limitations and restrictions listed here:

- Only one multi-valued key part is permitted per multi-valued index. However, the `CAST(... AS ... ARRAY)` expression can refer to multiple arrays within a `JSON` document, as shown here:

  ```
  CAST(data->'$.arr[*][*]' AS UNSIGNED ARRAY)
  ```

  In this case, all values matching the path expression are stored in the index as a single flat array.
- An index with a multi-valued key part does not support ordering and therefore cannot be used as a primary key. For the same reason, a multi-valued index cannot be defined using the `ASC` or `DESC` keyword.
- A multi-valued index cannot be a covering index.
- The number of keys that can be indexed for a single data record is based on the number of keys and total key length permitted by the storage engine. `InnoDB` limits are 655335 keys and 10000 bytes for the total key length. `InnoDB` limits are determined by the amount of data can be stored on a single undo log page. A single undo log page should permit indexing up to 1250 integer values, for example.
- The only type of expression that is permitted in a multi-valued key part is a `JSON` path. The path need not point to an existing element in a JSON document inserted into the indexed column, but must itself be syntactically valid.
- Because index records for the same clustered index record are dispersed throughout a multi-valued index, a multi-valued index does not support range scans or index-only scans.
- Multi-valued indexes are not permitted in foreign key specifications.
- Index prefixes cannot be defined for multi-valued indexes.

- Multi-valued indexes cannot be defined on data cast as `BINARY` (see the description of the `CAST()` function).
- Online creation of a multi-value index is not supported, which means the operation uses `ALGORITHM=COPY`. See Performance and Space Requirements.
- Character sets and collations other than the following two combinations of character set and collation are not supported for multi-valued indexes:

  1. The `binary` character set with the default `binary` collation
  2. The `utf8mb4` character set with the default `utf8mb4_0900_as_cs` collation (see following)

  It is possible for different binary strings to be mapped to the same character or characters when using the `utf8mb4_0900_as_cs` collation. Multi-value indexes remove duplicate values before they are stored, so any duplicate values are lost, and selecting one of these does not return any results. For example:

```
mysql> CREATE TABLE t1 (
    ->     f1 JSON,
    ->     UNIQUE KEY i1( (CAST(f1->"$[*]" AS CHAR(10) ARRAY)) )
    -> );
mysql> INSERT INTO t1(f1)
    ->  VALUES (JSON_ARRAY(_utf8mb4 0xE284AB, _utf8mb4 0xc385));
mysql> SELECT * FROM t1 WHERE _utf8mb4 0xc385 MEMBER OF (f1->"$[*]");
+------------+
| f1         |
+------------+
| ["Å", "Å"] |
+------------+
mysql> SELECT * FROM t1 WHERE _utf8mb4 0xE284AB MEMBER OF (f1->"$[*]");
Empty set (0,00 sec)
```

  You can (and should) anticipate such issues in advance by sanitizing data prior to inserting it.

- As with other indexes on columns of `InnoDB` tables, a multi-valued index cannot be created with `USING HASH`; attempting to do so results in a warning: `This storage engine does not support the HASH index algorithm, storage engine default was used instead`. (`USING BTREE` is supported as usual.)

Spatial Indexes

The `MyISAM`, `InnoDB`, `NDB`, and `ARCHIVE` storage engines support spatial columns such as `POINT` and `GEOMETRY`. (Section 11.5, “Spatial Data Types”, describes the spatial data types.) However, support for spatial column indexing varies among engines. Spatial and nonspatial indexes on spatial columns are available according to the following rules.

Spatial indexes on spatial columns have these characteristics:

- Available only for `InnoDB` and `MyISAM` tables. Specifying `SPATIAL INDEX` for other storage engines results in an error.
- As of MySQL 8.0.12, an index on a spatial column *must* be a `SPATIAL` index. The `SPATIAL` keyword is thus optional but implicit for creating an index on a spatial column.
- Available for single spatial columns only. A spatial index cannot be created over multiple spatial columns.
- Indexed columns must be `NOT NULL`.

- Column prefix lengths are prohibited. The full width of each column is indexed.
- Not permitted for a primary key or unique index.

Nonspatial indexes on spatial columns (created with `INDEX`, `UNIQUE`, or `PRIMARY KEY`) have these characteristics:

- Permitted for any storage engine that supports spatial columns except `ARCHIVE`.
- Columns can be `NULL` unless the index is a primary key.
- The index type for a non-`SPATIAL` index depends on the storage engine. Currently, B-tree is used.
- Permitted for a column that can have `NULL` values only for `InnoDB`, `MyISAM`, and `MEMORY` tables.

Index Options

Following the key part list, index options can be given. An *`index_option`* value can be any of the following:

- `KEY_BLOCK_SIZE [=]` *`value`*

  For `MyISAM` tables, `KEY_BLOCK_SIZE` optionally specifies the size in bytes to use for index key blocks. The value is treated as a hint; a different size could be used if necessary. A `KEY_BLOCK_SIZE` value specified for an individual index definition overrides a table-level `KEY_BLOCK_SIZE` value.

  `KEY_BLOCK_SIZE` is not supported at the index level for `InnoDB` tables. See Section 13.1.20, “CREATE TABLE Syntax”.

- *`index_type`*

  Some storage engines permit you to specify an index type when creating an index. For example:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index ON lookup (id) USING BTREE;
```

  Table 13.1, “Index Types Per Storage Engine” shows the permissible index type values supported by different storage engines. Where multiple index types are listed, the first one is the default when no index type specifier is given. Storage engines not listed in the table do not support an *`index_type`* clause in index definitions.

Table 13.1 Index Types Per Storage Engine

Storage Engine	Permissible Index Types
`InnoDB`	`BTREE`
`MyISAM`	`BTREE`
`MEMORY`/`HEAP`	`HASH`, `BTREE`
`NDB`	`HASH`, `BTREE` (see note in text)

The *`index_type`* clause cannot be used for `FULLTEXT INDEX` or (prior to MySQL 8.0.12) `SPATIAL INDEX` specifications. Full-text index implementation is storage engine dependent. Spatial indexes are implemented as R-tree indexes.

If you specify an index type that is not valid for a given storage engine, but another index type is available that the engine can use without affecting query results, the engine uses the available type.

The parser recognizes RTREE as a type name. As of MySQL 8.0.12, this is permitted only for SPATIAL indexes. Prior to 8.0.12, RTREE cannot be specified for any storage engine.

BTREE indexes are implemented by the NDB storage engine as T-tree indexes.

Note

For indexes on NDB table columns, the USING option can be specified only for a unique index or primary key. USING HASH prevents the creation of an ordered index; otherwise, creating a unique index or primary key on an NDB table automatically results in the creation of both an ordered index and a hash index, each of which indexes the same set of columns.

For unique indexes that include one or more NULL columns of an NDB table, the hash index can be used only to look up literal values, which means that IS [NOT] NULL conditions require a full scan of the table. One workaround is to make sure that a unique index using one or more NULL columns on such a table is always created in such a way that it includes the ordered index; that is, avoid employing USING HASH when creating the index.

If you specify an index type that is not valid for a given storage engine, but another index type is available that the engine can use without affecting query results, the engine uses the available type. The parser recognizes RTREE as a type name, but currently this cannot be specified for any storage engine.

Note

Use of the *index_type* option before the ON *tbl_name* clause is deprecated; support for use of the option in this position will be removed in a future MySQL release. If an *index_type* option is given in both the earlier and later positions, the final option applies.

TYPE *type_name* is recognized as a synonym for USING *type_name*. However, USING is the preferred form.

The following tables show index characteristics for the storage engines that support the *index_type* option.

Table 13.2 InnoDB Storage Engine Index Characteristics

Index Class	Index Type	Stores NULL VALUES	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
Primary key	BTREE	No	No	N/A	N/A
Unique	BTREE	Yes	Yes	Index	Index
Key	BTREE	Yes	Yes	Index	Index
FULLTEXT	N/A	Yes	Yes	Table	Table
SPATIAL	N/A	No	No	N/A	N/A

Table 13.3 MyISAM Storage Engine Index Characteristics

Index Class	Index Type	Stores NULL VALUES	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
Primary key	BTREE	No	No	N/A	N/A
Unique	BTREE	Yes	Yes	Index	Index

Index Class	Index Type	Stores NULL VALUES	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
Key	BTREE	Yes	Yes	Index	Index
FULLTEXT	N/A	Yes	Yes	Table	Table
SPATIAL	N/A	No	No	N/A	N/A

Table 13.4 MEMORY Storage Engine Index Characteristics

Index Class	Index Type	Stores NULL VALUES	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
Primary key	BTREE	No	No	N/A	N/A
Unique	BTREE	Yes	Yes	Index	Index
Key	BTREE	Yes	Yes	Index	Index
Primary key	HASH	No	No	N/A	N/A
Unique	HASH	Yes	Yes	Index	Index
Key	HASH	Yes	Yes	Index	Index

Table 13.5 NDB Storage Engine Index Characteristics

Index Class	Index Type	Stores NULL VALUES	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
Primary key	BTREE	No	No	Index	Index
Unique	BTREE	Yes	Yes	Index	Index
Key	BTREE	Yes	Yes	Index	Index
Primary key	HASH	No	No	Table (see note 1)	Table (see note 1)
Unique	HASH	Yes	Yes	Table (see note 1)	Table (see note 1)
Key	HASH	Yes	Yes	Table (see note 1)	Table (see note 1)

Table note:

1. `USING HASH` prevents creation of an implicit ordered index.

- `WITH PARSER` *`parser_name`*

  This option can be used only with `FULLTEXT` indexes. It associates a parser plugin with the index if full-text indexing and searching operations need special handling. `InnoDB` and `MyISAM` support full-text parser plugins. See Full-Text Parser Plugins and Section 29.2.4.4, “Writing Full-Text Parser Plugins” for more information.

- `COMMENT '`*`string`*`'`

  Index definitions can include an optional comment of up to 1024 characters.

  The `MERGE_THRESHOLD` for index pages can be configured for individual indexes using the *`index_option`* `COMMENT` clause of the `CREATE INDEX` statement. For example:

```
CREATE TABLE t1 (id INT);
CREATE INDEX id_index ON t1 (id) COMMENT 'MERGE_THRESHOLD=40';
```

If the page-full percentage for an index page falls below the MERGE_THRESHOLD value when a row is deleted or when a row is shortened by an update operation, InnoDB attempts to merge the index page with a neighboring index page. The default MERGE_THRESHOLD value is 50, which is the previously hardcoded value.

MERGE_THRESHOLD can also be defined at the index level and table level using CREATE TABLE and ALTER TABLE statements. For more information, see Section 15.8.11, “Configuring the Merge Threshold for Index Pages”.

- VISIBLE, INVISIBLE

  Specify index visibility. Indexes are visible by default. An invisible index is not used by the optimizer. Specification of index visibility applies to indexes other than primary keys (either explicit or implicit). For more information, see Section 8.3.12, “Invisible Indexes”.

Table Copying and Locking Options

ALGORITHM and LOCK clauses may be given to influence the table copying method and level of concurrency for reading and writing the table while its indexes are being modified. They have the same meaning as for the ALTER TABLE statement. For more information, see Section 13.1.9, “ALTER TABLE Syntax”

NDB Cluster supports online operations using the same ALGORITHM=INPLACE syntax used with the standard MySQL Server. See Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”, for more information.

13.1.16 CREATE LOGFILE GROUP Syntax

```
CREATE LOGFILE GROUP logfile_group
    ADD UNDOFILE 'undo_file'
    [INITIAL_SIZE [=] initial_size]
    [UNDO_BUFFER_SIZE [=] undo_buffer_size]
    [REDO_BUFFER_SIZE [=] redo_buffer_size]
    [NODEGROUP [=] nodegroup_id]
    [WAIT]
    [COMMENT [=] 'string']
    ENGINE [=] engine_name
```

This statement creates a new log file group named *logfile_group* having a single UNDO file named '*undo_file*'. A CREATE LOGFILE GROUP statement has one and only one ADD UNDOFILE clause. For rules covering the naming of log file groups, see Section 9.2, “Schema Object Names”.

Note

All NDB Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a log file group with the same name, or a tablespace and a data file with the same name.

There can be only one log file group per NDB Cluster instance at any given time.

The optional INITIAL_SIZE parameter sets the UNDO file's initial size; if not specified, it defaults to 128M (128 megabytes). The optional UNDO_BUFFER_SIZE parameter sets the size used by the UNDO buffer for the log file group; The default value for UNDO_BUFFER_SIZE is 8M (eight megabytes); this value cannot exceed the amount of system memory available. Both of these parameters are specified in bytes. You may

optionally follow either or both of these with a one-letter abbreviation for an order of magnitude, similar to those used in `my.cnf`. Generally, this is one of the letters `M` (for megabytes) or `G` (for gigabytes).

Memory used for `UNDO_BUFFER_SIZE` comes from the global pool whose size is determined by the value of the `SharedGlobalMemory` data node configuration parameter. This includes any default value implied for this option by the setting of the `InitialLogFileGroup` data node configuration parameter.

The maximum permitted for `UNDO_BUFFER_SIZE` is 629145600 (600 MB).

On 32-bit systems, the maximum supported value for `INITIAL_SIZE` is 4294967296 (4 GB). (Bug #29186)

The minimum allowed value for `INITIAL_SIZE` is 1048576 (1 MB).

The `ENGINE` option determines the storage engine to be used by this log file group, with *`engine_name`* being the name of the storage engine. In MySQL 8.0, this must be `NDB` (or `NDBCLUSTER`). If `ENGINE` is not set, MySQL tries to use the engine specified by the `default_storage_engine` server system variable (formerly `storage_engine`). In any case, if the engine is not specified as `NDB` or `NDBCLUSTER`, the `CREATE LOGFILE GROUP` statement appears to succeed but actually fails to create the log file group, as shown here:

```
mysql> CREATE LOGFILE GROUP lg1
    ->     ADD UNDOFILE 'undo.dat' INITIAL_SIZE = 10M;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-------+------+------------------------------------------------------------------------------------------
| Level | Code | Message
+-------+------+------------------------------------------------------------------------------------------
| Error | 1478 | Table storage engine 'InnoDB' does not support the create option 'TABLESPACE or LOGFILE GROUP
+-------+------+------------------------------------------------------------------------------------------
1 row in set (0.00 sec)

mysql> DROP LOGFILE GROUP lg1 ENGINE = NDB;
ERROR 1529 (HY000): Failed to drop LOGFILE GROUP

mysql> CREATE LOGFILE GROUP lg1
    ->     ADD UNDOFILE 'undo.dat' INITIAL_SIZE = 10M
    ->     ENGINE = NDB;
Query OK, 0 rows affected (2.97 sec)
```

The fact that the `CREATE LOGFILE GROUP` statement does not actually return an error when a non-`NDB` storage engine is named, but rather appears to succeed, is a known issue which we hope to address in a future release of NDB Cluster.

`REDO_BUFFER_SIZE`, `NODEGROUP`, `WAIT`, and `COMMENT` are parsed but ignored, and so have no effect in MySQL 8.0. These options are intended for future expansion.

When used with `ENGINE [=] NDB`, a log file group and associated `UNDO` log file are created on each Cluster data node. You can verify that the `UNDO` files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example:

```
mysql> SELECT LOGFILE_GROUP_NAME, LOGFILE_GROUP_NUMBER, EXTRA
    -> FROM INFORMATION_SCHEMA.FILES
    -> WHERE FILE_NAME = 'undo_10.dat';
+--------------------+----------------------+----------------+
| LOGFILE_GROUP_NAME | LOGFILE_GROUP_NUMBER | EXTRA          |
+--------------------+----------------------+----------------+
```

```
| lg_3               |                   11 | CLUSTER_NODE=3 |
| lg_3               |                   11 | CLUSTER_NODE=4 |
+--------------------+----------------------+----------------+
2 rows in set (0.06 sec)
```

`CREATE LOGFILE GROUP` is useful only with Disk Data storage for NDB Cluster. See Section 22.5.13, “NDB Cluster Disk Data Tables”.

13.1.17 CREATE PROCEDURE and CREATE FUNCTION Syntax

```
CREATE
    [DEFINER = user]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = user]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic:
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }

routine_body:
    Valid SQL routine statement
```

These statements create stored routines. By default, a routine is associated with the default database. To associate the routine explicitly with a given database, specify the name as *`db_name.sp_name`* when you create it.

The `CREATE FUNCTION` statement is also used in MySQL to support UDFs (user-defined functions). See Section 29.4, “Adding New Functions to MySQL”. A UDF can be regarded as an external stored function. Stored functions share their namespace with UDFs. See Section 9.2.5, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

To invoke a stored procedure, use the `CALL` statement (see Section 13.2.1, “CALL Syntax”). To invoke a stored function, refer to it in an expression. The function returns a value during expression evaluation.

`CREATE PROCEDURE` and `CREATE FUNCTION` require the `CREATE ROUTINE` privilege. If the `DEFINER` clause is present, the privileges required depend on the *`user`* value, as discussed in Section 24.6, “Stored Object Access Control”. If binary logging is enabled, `CREATE FUNCTION` might require the `SUPER` privilege, as discussed in Section 24.7, “Stored Program Binary Logging”.

By default, MySQL automatically grants the `ALTER ROUTINE` and `EXECUTE` privileges to the routine creator. This behavior can be changed by disabling the `automatic_sp_privileges` system variable. See Section 24.2.2, “Stored Routines and MySQL Privileges”.

The `DEFINER` and `SQL SECURITY` clauses specify the security context to be used when checking access privileges at routine execution time, as described later in this section.

If the routine name is the same as the name of a built-in SQL function, a syntax error occurs unless you use a space between the name and the following parenthesis when defining the routine or invoking it later. For this reason, avoid using the names of existing SQL functions for your own stored routines.

The `IGNORE_SPACE` SQL mode applies to built-in functions, not to stored routines. It is always permissible to have spaces after a stored routine name, regardless of whether `IGNORE_SPACE` is enabled.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of `()` should be used. Parameter names are not case sensitive.

Each parameter is an `IN` parameter by default. To specify otherwise for a parameter, use the keyword `OUT` or `INOUT` before the parameter name.

Note

Specifying a parameter as `IN`, `OUT`, or `INOUT` is valid only for a `PROCEDURE`. For a `FUNCTION`, parameters are always regarded as `IN` parameters.

An `IN` parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns. An `OUT` parameter passes a value from the procedure back to the caller. Its initial value is `NULL` within the procedure, and its value is visible to the caller when the procedure returns. An `INOUT` parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

For each `OUT` or `INOUT` parameter, pass a user-defined variable in the `CALL` statement that invokes the procedure so that you can obtain its value when the procedure returns. If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `OUT` or `INOUT` parameter. If you are calling the procedure from within a trigger, you can also pass `NEW.col_name` as an `OUT` or `INOUT` parameter.

For information about the effect of unhandled conditions on procedure parameters, see Section 13.6.7.8, “Condition Handling and OUT or INOUT Parameters”.

Routine parameters cannot be referenced in statements prepared within the routine; see Section 24.8, “Restrictions on Stored Programs”.

The following example shows a simple stored procedure that uses an `OUT` parameter:

```
mysql> delimiter //

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO param1 FROM t;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+------+
| @a   |
+------+
| 3    |
```

```
+------+
1 row in set (0.00 sec)
```

The example uses the `mysql` client `delimiter` command to change the statement delimiter from `;` to `//` while the procedure is being defined. This enables the `;` delimiter used in the procedure body to be passed through to the server rather than being interpreted by `mysql` itself. See Section 24.1, “Defining Stored Programs”.

The `RETURNS` clause may be specified only for a `FUNCTION`, for which it is mandatory. It indicates the return type of the function, and the function body must contain a `RETURN value` statement. If the `RETURN` statement returns a value of a different type, the value is coerced to the proper type. For example, if a function specifies an `ENUM` or `SET` value in the `RETURNS` clause, but the `RETURN` statement returns an integer, the value returned from the function is the string for the corresponding `ENUM` member of set of `SET` members.

The following example function takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use `delimiter` because the function definition contains no internal `;` statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
    -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

Parameter types and function return types can be declared to use any valid data type. The `COLLATE` attribute can be used if preceded by a `CHARACTER SET` specification.

The *`routine_body`* consists of a valid SQL routine statement. This can be a simple statement such as `SELECT` or `INSERT`, or a compound statement written using `BEGIN` and `END`. Compound statements can contain declarations, loops, and other control structure statements. The syntax for these statements is described in Section 13.6, “Compound-Statement Syntax”. In practice, stored functions tend to use compound statements, unless the body consists of a single `RETURN` statement.

MySQL permits routines to contain DDL statements, such as `CREATE` and `DROP`. MySQL also permits stored procedures (but not stored functions) to contain SQL transaction statements such as `COMMIT`. Stored functions may not contain statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.

Statements that return a result set can be used within a stored procedure but not within a stored function. This prohibition includes `SELECT` statements that do not have an `INTO var_list` clause and other statements such as `SHOW`, `EXPLAIN`, and `CHECK TABLE`. For statements that can be determined at function definition time to return a result set, a `Not allowed to return a result set from a function` error occurs (`ER_SP_NO_RETSET`). For statements that can be determined only at runtime to return a result set, a `PROCEDURE %s can't return a result set in the given context` error occurs (`ER_SP_BADSELECT`).

`USE` statements within stored routines are not permitted. When a routine is invoked, an implicit `USE db_name` is performed (and undone when the routine terminates). The causes the routine to have the

given default database while it executes. References to objects in databases other than the routine default database should be qualified with the appropriate database name.

For additional information about statements that are not permitted in stored routines, see Section 24.8, “Restrictions on Stored Programs”.

For information about invoking stored procedures from within programs written in a language that has a MySQL interface, see Section 13.2.1, “CALL Syntax”.

MySQL stores the `sql_mode` system variable setting in effect when a routine is created or altered, and always executes the routine with this setting in force, *regardless of the current server SQL mode when the routine begins executing*.

The switch from the SQL mode of the invoker to that of the routine occurs after evaluation of arguments and assignment of the resulting values to routine parameters. If you define a routine in strict SQL mode but invoke it in nonstrict mode, assignment of arguments to routine parameters does not take place in strict mode. If you require that expressions passed to a routine be assigned in strict SQL mode, you should invoke the routine with strict mode in effect.

The `COMMENT` characteristic is a MySQL extension, and may be used to describe the stored routine. This information is displayed by the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements.

The `LANGUAGE` characteristic indicates the language in which the routine is written. The server ignores this characteristic; only SQL routines are supported.

A routine is considered “deterministic” if it always produces the same result for the same input parameters, and “not deterministic” otherwise. If neither `DETERMINISTIC` nor `NOT DETERMINISTIC` is given in the routine definition, the default is `NOT DETERMINISTIC`. To declare that a function is deterministic, you must specify `DETERMINISTIC` explicitly.

Assessment of the nature of a routine is based on the “honesty” of the creator: MySQL does not check that a routine declared `DETERMINISTIC` is free of statements that produce nondeterministic results. However, misdeclaring a routine might affect results or affect performance. Declaring a nondeterministic routine as `DETERMINISTIC` might lead to unexpected results by causing the optimizer to make incorrect execution plan choices. Declaring a deterministic routine as `NONDETERMINISTIC` might diminish performance by causing available optimizations not to be used.

If binary logging is enabled, the `DETERMINISTIC` characteristic affects which routine definitions MySQL accepts. See Section 24.7, “Stored Program Binary Logging”.

A routine that contains the `NOW()` function (or its synonyms) or `RAND()` is nondeterministic, but it might still be replication-safe. For `NOW()`, the binary log includes the timestamp and replicates correctly. `RAND()` also replicates correctly as long as it is called only a single time during the execution of a routine. (You can consider the routine execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

Several characteristics provide information about the nature of data use by the routine. In MySQL, these characteristics are advisory only. The server does not use them to constrain what kinds of statements a routine will be permitted to execute.

- `CONTAINS SQL` indicates that the routine does not contain statements that read or write data. This is the default if none of these characteristics is given explicitly. Examples of such statements are `SET @x = 1` or `DO RELEASE_LOCK('abc')`, which execute but neither read nor write data.
- `NO SQL` indicates that the routine contains no SQL statements.
- `READS SQL DATA` indicates that the routine contains statements that read data (for example, `SELECT`), but not statements that write data.

- `MODIFIES SQL DATA` indicates that the routine contains statements that may write data (for example, `INSERT` or `DELETE`).

The `SQL SECURITY` characteristic can be `DEFINER` or `INVOKER` to specify the security context; that is, whether the routine executes using the privileges of the account named in the routine `DEFINER` clause or the user who invokes it. This account must have permission to access the database with which the routine is associated. The default value is `DEFINER`. The user who invokes the routine must have the `EXECUTE` privilege for it, as must the `DEFINER` account if the routine executes in definer security context.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at routine execution time for routines that have the `SQL SECURITY DEFINER` characteristic.

If the `DEFINER` clause is present, the *`user`* value should be a MySQL account specified as *`'user_name'@'host_name'`*, `CURRENT_USER`, or `CURRENT_USER()`. The permitted *`user`* values depend on the privileges you hold, as discussed in Section 24.6, “Stored Object Access Control”. Also see that section for additional information about stored routine security.

If the `DEFINER` clause is omitted, the default definer is the user who executes the `CREATE PROCEDURE` or `CREATE FUNCTION` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

Within the body of a stored routine that is defined with the `SQL SECURITY DEFINER` characteristic, the `CURRENT_USER` function returns the routine's `DEFINER` value. For information about user auditing within stored routines, see Section 6.2.22, “SQL-Based Account Activity Auditing”.

Consider the following procedure, which displays a count of the number of MySQL accounts listed in the `mysql.user` system table:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

The procedure is assigned a `DEFINER` account of `'admin'@'localhost'` no matter which user defines it. It executes with the privileges of that account no matter which user invokes it (because the default security characteristic is `DEFINER`). The procedure succeeds or fails depending on whether invoker has the `EXECUTE` privilege for it and `'admin'@'localhost'` has the `SELECT` privilege for the `mysql.user` table.

Now suppose that the procedure is defined with the `SQL SECURITY INVOKER` characteristic:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
SQL SECURITY INVOKER
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

The procedure still has a `DEFINER` of `'admin'@'localhost'`, but in this case, it executes with the privileges of the invoking user. Thus, the procedure succeeds or fails depending on whether the invoker has the `EXECUTE` privilege for it and the `SELECT` privilege for the `mysql.user` table.

The server handles the data type of a routine parameter, local routine variable created with `DECLARE`, or function return value as follows:

- Assignments are checked for data type mismatches and overflow. Conversion and overflow problems result in warnings, or errors in strict SQL mode.
- Only scalar values can be assigned. For example, a statement such as `SET x = (SELECT 1, 2)` is invalid.

- For character data types, if `CHARACTER SET` is includedd in the declaration, the specified character set and its default collation is used. If the `COLLATE` attribute is also present, that collation is used rather than the default collation.

  If `CHARACTER SET` and `COLLATE` are not present, the database character set and collation in effect at routine creation time are used. To avoid having the server use the database character set and collation, provide an explicit `CHARACTER SET` and a `COLLATE` attribute for character data parameters.

  If you change the database default character set or collation, stored routines that use the database defaults must be dropped and recreated so that they use the new defaults.

  The database character set and collation are given by the value of the `character_set_database` and `collation_database` system variables. For more information, see Section 10.3.3, “Database Character Set and Collation”.

13.1.18 CREATE SERVER Syntax

```
CREATE SERVER server_name
    FOREIGN DATA WRAPPER wrapper_name
    OPTIONS (option [, option] ...)

option:
  { HOST character-literal
  | DATABASE character-literal
  | USER character-literal
  | PASSWORD character-literal
  | SOCKET character-literal
  | OWNER character-literal
  | PORT numeric-literal }
```

This statement creates the definition of a server for use with the `FEDERATED` storage engine. The `CREATE SERVER` statement creates a new row in the `servers` table in the `mysql` database. This statement requires the `SUPER` privilege.

The *`server_name`* should be a unique reference to the server. Server definitions are global within the scope of the server, it is not possible to qualify the server definition to a specific database. *`server_name`* has a maximum length of 64 characters (names longer than 64 characters are silently truncated), and is case insensitive. You may specify the name as a quoted string.

The *`wrapper_name`* is an identifier and may be quoted with single quotation marks.

For each *`option`* you must specify either a character literal or numeric literal. Character literals are UTF-8, support a maximum length of 64 characters and default to a blank (empty) string. String literals are silently truncated to 64 characters. Numeric literals must be a number between 0 and 9999, default value is 0.

Note

The `OWNER` option is currently not applied, and has no effect on the ownership or operation of the server connection that is created.

The `CREATE SERVER` statement creates an entry in the `mysql.servers` table that can later be used with the `CREATE TABLE` statement when creating a `FEDERATED` table. The options that you specify will be used to populate the columns in the `mysql.servers` table. The table columns are `Server_name`, `Host`, `Db`, `Username`, `Password`, `Port` and `Socket`.

For example:

```
CREATE SERVER s
```

```
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'Remote', HOST '198.51.100.106', DATABASE 'test');
```

Be sure to specify all options necessary to establish a connection to the server. The user name, host name, and database name are mandatory. Other options might be required as well, such as password.

The data stored in the table can be used when creating a connection to a `FEDERATED` table:

```
CREATE TABLE t (s1 INT) ENGINE=FEDERATED CONNECTION='s';
```

For more information, see Section 16.8, “The FEDERATED Storage Engine”.

`CREATE SERVER` causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

`CREATE SERVER` is not written to the binary log, regardless of the logging format that is in use.

13.1.19 CREATE SPATIAL REFERENCE SYSTEM Syntax

```
CREATE OR REPLACE SPATIAL REFERENCE SYSTEM
    srid srs_attribute ...

CREATE SPATIAL REFERENCE SYSTEM
    [IF NOT EXISTS]
    srid srs_attribute ...

srs_attribute: {
    NAME 'srs_name'
  | DEFINITION 'definition'
  | ORGANIZATION 'org_name' IDENTIFIED BY org_id
  | DESCRIPTION 'description'
}

srid, org_id: 32-bit unsigned integer
```

This statement creates a spatial reference system (SRS) definition and stores it in the data dictionary. The definition can be inspected using the `INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS` table. This statement requires the `SUPER` privilege.

If neither `OR REPLACE` nor `IF NOT EXISTS` is specified, an error occurs if an SRS definition with the SRID value already exists.

With `CREATE OR REPLACE` syntax, any existing SRS definition with the same SRID value is replaced, unless the SRID value is used by some column in an existing table. In that case, an error occurs. For example:

```
mysql> CREATE OR REPLACE SPATIAL REFERENCE SYSTEM 4326 ...;
ERROR 3716 (SR005): Can't modify SRID 4326. There is at
least one column depending on it.
```

To identify which column or columns use the SRID, use this query:

```
SELECT * FROM INFORMATION_SCHEMA.ST_GEOMETRY_COLUMNS WHERE SRS_ID=4326;
```

With `CREATE ... IF NOT EXISTS` syntax, any existing SRS definition with the same SRID value causes the new definition to be ignored and a warning occurs.

SRID values must be in the range of 32-bit unsigned integers, with these restrictions:

- SRID 0 is a valid SRID but cannot be used with `CREATE SPATIAL REFERENCE SYSTEM`.

- If the value is in a reserved SRID range, a warning occurs. Reserved ranges are [0, 32767] (reserved by EPSG), [60,000,000, 69,999,999] (reserved by EPSG), and [2,000,000,000, 2,147,483,647] (reserved by MySQL). EPSG stands for the European Petroleum Survey Group.
- Users should not create SRSs with SRIDs in the reserved ranges. Doing so runs the risk that the SRIDs will conflict with future SRS definitions distributed with MySQL, with the result that the new system-provided SRSs are not installed for MySQL upgrades or that the user-defined SRSs are overwritten.

Attributes for the statement must satisfy these conditions:

- Attributes can be given in any order, but no attribute can be given more than once.
- The `NAME` and `DEFINITION` attributes are mandatory.
- The `NAME` *`srs_name`* attribute value must be unique. The combination of the `ORGANIZATION` *`org_name`* and *`org_id`* attribute values must be unique.
- The `NAME` *`srs_name`* attribute value and `ORGANIZATION` *`org_name`* attribute value cannot be empty or begin or end with whitespace.
- String values in attribute specifications cannot contain control characters, including newline.
- The following table shows the maximum lengths for string attribute values.

Table 13.6 CREATE SPATIAL REFERENCE SYSTEM Attribute Lengths

Attribute	Maximum Length (characters)
`NAME`	80
`DEFINITION`	4096
`ORGANIZATION`	256
`DESCRIPTION`	2048

Here is an example `CREATE SPATIAL REFERENCE SYSTEM` statement. The `DEFINITION` value is reformatted across multiple lines for readability. (For the statement to be legal, the value actually must be given on a single line.)

```
CREATE SPATIAL REFERENCE SYSTEM 4120
NAME 'Greek'
ORGANIZATION 'EPSG' IDENTIFIED BY 4120
DEFINITION
  'GEOGCS["Greek",DATUM["Greek",SPHEROID["Bessel 1841",
  6377397.155,299.1528128,AUTHORITY["EPSG","7004"]],
  AUTHORITY["EPSG","6120"]],PRIMEM["Greenwich",0,
  AUTHORITY["EPSG","8901"]],UNIT["degree",0.017453292519943278,
  AUTHORITY["EPSG","9122"]],AXIS["Lat",NORTH],AXIS["Lon",EAST],
  AUTHORITY["EPSG","4120"]]';
```

The grammar for SRS definitions is based on the grammar defined in *OpenGIS Implementation Specification: Coordinate Transformation Services*, Revision 1.00, OGC 01-009, January 12, 2001, Section 7.2. This specification is available at http://www.opengeospatial.org/standards/ct.

MySQL incorporates these changes to the specification:

- Only the `<horz cs>` production rule is implemented (that is, geographic and projected SRSs).
- There is an optional, nonstandard `<authority>` clause for `<parameter>`. This makes it possible to recognize projection parameters by authority instead of name.

- The specification does not make `AXIS` clauses mandatory in `GEOGCS` spatial reference system definitions. However, if there are no `AXIS` clauses, MySQL cannot determine whether a definition has axes in latitude-longitude order or longitude-latitude order. MySQL enforces the nonstandard requirement that each `GEOGCS` definition must include two `AXIS` clauses. One must be `NORTH` or `SOUTH`, and the other `EAST` or `WEST`. The `AXIS` clause order determines whether the definition has axes in latitude-longitude order or longitude-latitude order.
- SRS definitions may not contain newlines.

If an SRS definition specifies an authority code for the projection (which is recommended), an error occurs if the definition is missing mandatory parameters. In this case, the error message indicates what the problem is. The projection methods and mandatory parameters that MySQL supports are shown in Table 13.7, “Supported Spatial Reference System Projection Methods” and Table 13.8, “Spatial Reference System Projection Parameters”.

For additional information about writing SRS definitions for MySQL, see Geographic Spatial Reference Systems in MySQL 8.0 and Projected Spatial Reference Systems in MySQL 8.0

The following table shows the projection methods that MySQL supports. MySQL permits unknown projection methods but cannot check the defintion for mandatory paramters and cannot convert spatial data to or from an unknown projection. For detailed explanations of how each projection works, including formulas, see EPSG Guidance Note 7-2.

Table 13.7 Supported Spatial Reference System Projection Methods

EPSG Code	Projection Name	Mandatory Parameters (EPSG Codes)
1024	Popular Visualisation Pseudo Mercator	8801, 8802, 8806, 8807
1027	Lambert Azimuthal Equal Area (Spherical)	8801, 8802, 8806, 8807
1028	Equidistant Cylindrical	8823, 8802, 8806, 8807
1029	Equidistant Cylindrical (Spherical)	8823, 8802, 8806, 8807
1041	Krovak (North Orientated)	8811, 8833, 1036, 8818, 8819, 8806, 8807
1042	Krovak Modified	8811, 8833, 1036, 8818, 8819, 8806, 8807, 8617, 8618, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035
1043	Krovak Modified (North Orientated)	8811, 8833, 1036, 8818, 8819, 8806, 8807, 8617, 8618, 1026, 1027, 1028, 1029, 1030, 1031, 1032, 1033, 1034, 1035
1051	Lambert Conic Conformal (2SP Michigan)	8821, 8822, 8823, 8824, 8826, 8827, 1038
1052	Colombia Urban	8801, 8802, 8806, 8807, 1039
9801	Lambert Conic Conformal (1SP)	8801, 8802, 8805, 8806, 8807
9802	Lambert Conic Conformal (2SP)	8821, 8822, 8823, 8824, 8826, 8827
9803	Lambert Conic Conformal (2SP Belgium)	8821, 8822, 8823, 8824, 8826, 8827

EPSG Code	Projection Name	Mandatory Parameters (EPSG Codes)
9804	Mercator (variant A)	8801, 8802, 8805, 8806, 8807
9805	Mercator (variant B)	8823, 8802, 8806, 8807
9806	Cassini-Soldner	8801, 8802, 8806, 8807
9807	Transverse Mercator	8801, 8802, 8805, 8806, 8807
9808	Transverse Mercator (South Orientated)	8801, 8802, 8805, 8806, 8807
9809	Oblique Stereographic	8801, 8802, 8805, 8806, 8807
9810	Polar Stereographic (variant A)	8801, 8802, 8805, 8806, 8807
9811	New Zealand Map Grid	8801, 8802, 8806, 8807
9812	Hotine Oblique Mercator (variant A)	8811, 8812, 8813, 8814, 8815, 8806, 8807
9813	Laborde Oblique Mercator	8811, 8812, 8813, 8815, 8806, 8807
9815	Hotine Oblique Mercator (variant B)	8811, 8812, 8813, 8814, 8815, 8816, 8817
9816	Tunisia Mining Grid	8821, 8822, 8826, 8827
9817	Lambert Conic Near-Conformal	8801, 8802, 8805, 8806, 8807
9818	American Polyconic	8801, 8802, 8806, 8807
9819	Krovak	8811, 8833, 1036, 8818, 8819, 8806, 8807
9820	Lambert Azimuthal Equal Area	8801, 8802, 8806, 8807
9822	Albers Equal Area	8821, 8822, 8823, 8824, 8826, 8827
9824	Transverse Mercator Zoned Grid System	8801, 8830, 8831, 8805, 8806, 8807
9826	Lambert Conic Conformal (West Orientated)	8801, 8802, 8805, 8806, 8807
9828	Bonne (South Orientated)	8801, 8802, 8806, 8807
9829	Polar Stereographic (variant B)	8832, 8833, 8806, 8807
9830	Polar Stereographic (variant C)	8832, 8833, 8826, 8827
9831	Guam Projection	8801, 8802, 8806, 8807
9832	Modified Azimuthal Equidistant	8801, 8802, 8806, 8807
9833	Hyperbolic Cassini-Soldner	8801, 8802, 8806, 8807
9834	Lambert Cylindrical Equal Area (Spherical)	8823, 8802, 8806, 8807
9835	Lambert Cylindrical Equal Area	8823, 8802, 8806, 8807

The following table shows the projection parameters that MySQL recognizes. Recognition occurs primarily by authority code. If there is no authority code, MySQL falls back to case-insensitive string matching on the parameter name. For details about each parameter, look it up by code in the EPSG Online Registry.

Table 13.8 Spatial Reference System Projection Parameters

EPSG Code	Fallback Name (Recognized by MySQL)	EPSG Name
1026	c1	C1
1027	c2	C2
1028	c3	C3
1029	c4	C4
1030	c5	C5
1031	c6	C6
1032	c7	C7
1033	c8	C8
1034	c9	C9
1035	c10	C10
1036	azimuth	Co-latitude of cone axis
1038	ellipsoid_scale_factor	Ellipsoid scaling factor
1039	projection_plane_height_at_origin	Projection plane origin height
8617	evaluation_point_ordinate_1	Ordinate 1 of evaluation point
8618	evaluation_point_ordinate_2	Ordinate 2 of evaluation point
8801	latitude_of_origin	Latitude of natural origin
8802	central_meridian	Longitude of natural origin
8805	scale_factor	Scale factor at natural origin
8806	false_easting	False easting
8807	false_northing	False northing
8811	latitude_of_center	Latitude of projection centre
8812	longitude_of_center	Longitude of projection centre
8813	azimuth	Azimuth of initial line
8814	rectified_grid_angle	Angle from Rectified to Skew Grid
8815	scale_factor	Scale factor on initial line
8816	false_easting	Easting at projection centre
8817	false_northing	Northing at projection centre
8818	pseudo_standard_parallel_1	Latitude of pseudo standard parallel
8819	scale_factor	Scale factor on pseudo standard parallel
8821	latitude_of_origin	Latitude of false origin
8822	central_meridian	Longitude of false origin
8823	standard_parallel_1, standard_parallel1	Latitude of 1st standard parallel
8824	standard_parallel_2, standard_parallel2	Latitude of 2nd standard parallel

EPSG Code	Fallback Name (Recognized by MySQL)	EPSG Name
8826	false_easting	Easting at false origin
8827	false_northing	Northing at false origin
8830	initial_longitude	Initial longitude
8831	zone_width	Zone width
8832	standard_parallel	Latitude of standard parallel
8833	longitude_of_center	Longitude of origin

13.1.20 CREATE TABLE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options]
    [partition_options]
    [IGNORE | REPLACE]
    [AS] query_expression

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    { LIKE old_tbl_name | (LIKE old_tbl_name) }

create_definition:
    col_name column_definition
  | {INDEX|KEY} [index_name] [index_type] (key_part,...)
      [index_option] ...
  | {FULLTEXT|SPATIAL} [INDEX|KEY] [index_name] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] PRIMARY KEY
      [index_type] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] UNIQUE [INDEX|KEY]
      [index_name] [index_type] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] FOREIGN KEY
      [index_name] (col_name,...)
      reference_definition
  | check_constraint_definition

column_definition:
    data_type [NOT NULL | NULL] [DEFAULT {literal | (expr)} ]
      [AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
      [COMMENT 'string']
      [COLLATE collation_name]
      [COLUMN_FORMAT {FIXED|DYNAMIC|DEFAULT}]
      [STORAGE {DISK|MEMORY}]
      [reference_definition]
      [check_constraint_definition]
  | data_type
      [COLLATE collation_name]
      [GENERATED ALWAYS] AS (expr)
      [VIRTUAL | STORED] [NOT NULL | NULL]
      [UNIQUE [KEY]] [[PRIMARY] KEY]
      [COMMENT 'string']
      [reference_definition]
      [check_constraint_definition]
```

```
data_type:
    (see Chapter 11, Data Types)

key_part: {col_name [(length)] | (expr)} [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | {VISIBLE | INVISIBLE}

check_constraint_definition:
    [CONSTRAINT [symbol]] CHECK (expr) [[NOT] ENFORCED]

reference_definition:
    REFERENCES tbl_name (key_part,...)
      [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
      [ON DELETE reference_option]
      [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options:
    table_option [[,] table_option] ...

table_option:
    AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | COMPRESSION [=] {'ZLIB'|'LZ4'|'NONE'}
  | CONNECTION [=] 'connect_string'
  | {DATA|INDEX} DIRECTORY [=] 'absolute path to directory'
  | DELAY_KEY_WRITE [=] {0 | 1}
  | ENCRYPTION [=] {'Y' | 'N'}
  | ENGINE [=] engine_name
  | INSERT_METHOD [=] { NO | FIRST | LAST }
  | KEY_BLOCK_SIZE [=] value
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PACK_KEYS [=] {0 | 1 | DEFAULT}
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
  | STATS_AUTO_RECALC [=] {DEFAULT|0|1}
  | STATS_PERSISTENT [=] {DEFAULT|0|1}
  | STATS_SAMPLE_PAGES [=] value
  | TABLESPACE tablespace_name [STORAGE {DISK|MEMORY}]
  | UNION [=] (tbl_name[,tbl_name]...)

partition_options:
    PARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY [ALGORITHM={1|2}] (column_list)
        | RANGE{(expr) | COLUMNS(column_list)}
        | LIST{(expr) | COLUMNS(column_list)} }
    [PARTITIONS num]
    [SUBPARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY [ALGORITHM={1|2}] (column_list) }
```

```
      [SUBPARTITIONS num]
    ]
    [(partition_definition [, partition_definition] ...)]

partition_definition:
    PARTITION partition_name
        [VALUES
            {LESS THAN {(expr | value_list) | MAXVALUE}
            |
            IN (value_list)}]
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'string' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]
        [(subpartition_definition [, subpartition_definition] ...)]

subpartition_definition:
    SUBPARTITION logical_name
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'string' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]

query_expression:
    SELECT ...   (Some valid select or union statement)
```

`CREATE TABLE` creates a table with the given name. You must have the `CREATE` privilege for the table.

By default, tables are created in the default database, using the `InnoDB` storage engine. An error occurs if the table exists, if there is no default database, or if the database does not exist.

MySQL has no limit on the number of tables. The underlying file system may have a limit on the number of files that represent tables. Individual storage engines may impose engine-specific constraints. `InnoDB` permits up to 4 billion tables.

For information about the physical representation of a table, see Section 13.1.20.2, “Files Created by CREATE TABLE”.

The original `CREATE TABLE` statement, including all specifications and table options are stored by MySQL when the table is created. For more information, see Section 13.1.20.1, “CREATE TABLE Statement Retention”.

There are several aspects to the `CREATE TABLE` statement, described under the following topics in this section:

- Table Name
- Temporary Tables
- Table Cloning and Copying
- Column Data Types and Attributes
- Indexes, Foreign Keys, and CHECK Constraints
- Table Options
- Table Partitioning

Table Name

- *tbl_name*

  The table name can be specified as *db_name.tbl_name* to create the table in a specific database. This works regardless of whether there is a default database, assuming that the database exists. If you use quoted identifiers, quote the database and table names separately. For example, write `mydb`.`mytbl`, not `mydb.mytbl`.

  Rules for permissible table names are given in Section 9.2, “Schema Object Names”.

- IF NOT EXISTS

  Prevents an error from occurring if the table exists. However, there is no verification that the existing table has a structure identical to that indicated by the CREATE TABLE statement.

Temporary Tables

You can use the TEMPORARY keyword when creating a table. A TEMPORARY table is visible only within the current session, and is dropped automatically when the session is closed. For more information, see Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”.

Table Cloning and Copying

- LIKE

  Use CREATE TABLE ... LIKE to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

  For more information, see Section 13.1.20.4, “CREATE TABLE ... LIKE Syntax”.

- [AS] *query_expression*

  To create one table from another, add a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl AS SELECT * FROM orig_tbl;
```

  For more information, see Section 13.1.20.5, “CREATE TABLE ... SELECT Syntax”.

- IGNORE|REPLACE

  The IGNORE and REPLACE options indicate how to handle rows that duplicate unique key values when copying a table using a SELECT statement.

  For more information, see Section 13.1.20.5, “CREATE TABLE ... SELECT Syntax”.

Column Data Types and Attributes

There is a hard limit of 4096 columns per table, but the effective maximum may be less for a given table and depends on the factors discussed in Section 8.4.7, “Limits on Table Column Count and Row Size”.

- *data_type*

data_type represents the data type in a column definition. For a full description of the syntax available for specifying column data types, as well as information about the properties of each type, see Chapter 11, *Data Types*.

- Some attributes do not apply to all data types. AUTO_INCREMENT applies only to integer and floating-point types. Prior to MySQL 8.0.13, DEFAULT does not apply to the BLOB, TEXT, GEOMETRY, and JSON types.

- Character data types (CHAR, VARCHAR, the TEXT types, ENUM, SET, and any synonyms) synonyms) can include CHARACTER SET to specify the character set for the column. CHARSET is a synonym for CHARACTER SET. A collation for the character set can be specified with the COLLATE attribute, along with any other attributes. For details, see Chapter 10, *Character Sets, Collations, Unicode*. Example:

  ```
  CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
  ```

  MySQL 8.0 interprets length specifications in character column definitions in characters. Lengths for BINARY and VARBINARY are in bytes.

- For CHAR, VARCHAR, BINARY, and VARBINARY columns, indexes can be created that use only the leading part of column values, using *col_name(length)* syntax to specify an index prefix length. BLOB and TEXT columns also can be indexed, but a prefix length *must* be given. Prefix lengths are given in characters for nonbinary string types and in bytes for binary string types. That is, index entries consist of the first *length* characters of each column value for CHAR, VARCHAR, and TEXT columns, and the first *length* bytes of each column value for BINARY, VARBINARY, and BLOB columns. Indexing only a prefix of column values like this can make the index file much smaller. For additional information about index prefixes, see Section 13.1.15, “CREATE INDEX Syntax”.

  Only the InnoDB and MyISAM storage engines support indexing on BLOB and TEXT columns. For example:

  ```
  CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
  ```

  If a specified index prefix exceeds the maximum column data type size, CREATE TABLE handles the index as follows:

  - For a nonunique index, either an error occurs (if strict SQL mode is enabled), or the index length is reduced to lie within the maximum column data type size and a warning is produced (if strict SQL mode is not enabled).

  - For a unique index, an error occurs regardless of SQL mode because reducing the index length might enable insertion of nonunique entries that do not meet the specified uniqueness requirement.

- JSON columns cannot be indexed. You can work around this restriction by creating an index on a generated column that extracts a scalar value from the JSON column. See Indexing a Generated Column to Provide a JSON Column Index, for a detailed example.

- NOT NULL | NULL

  If neither NULL nor NOT NULL is specified, the column is treated as though NULL had been specified.

  In MySQL 8.0, only the InnoDB, MyISAM, and MEMORY storage engines support indexes on columns that can have NULL values. In other cases, you must declare indexed columns as NOT NULL or an error results.

- `DEFAULT`

  Specifies a default value for a column. For more information about default value handling, including the case that a column definition includes no explicit `DEFAULT` value, see Section 11.7, “Data Type Default Values”.

  If the `NO_ZERO_DATE` or `NO_ZERO_IN_DATE` SQL mode is enabled and a date-valued default is not correct according to that mode, `CREATE TABLE` produces a warning if strict SQL mode is not enabled and an error if strict mode is enabled. For example, with `NO_ZERO_IN_DATE` enabled, `c1 DATE DEFAULT '2010-00-00'` produces a warning.

- `AUTO_INCREMENT`

  An integer or floating-point column can have the additional attribute `AUTO_INCREMENT`. When you insert a value of `NULL` (recommended) or `0` into an indexed `AUTO_INCREMENT` column, the column is set to the next sequence value. Typically this is *`value`*`+1`, where *`value`* is the largest value for the column currently in the table. `AUTO_INCREMENT` sequences begin with `1`.

  To retrieve an `AUTO_INCREMENT` value after inserting a row, use the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function. See Section 12.15, “Information Functions”, and Section 28.7.7.38, “mysql_insert_id()”.

  If the `NO_AUTO_VALUE_ON_ZERO` SQL mode is enabled, you can store `0` in `AUTO_INCREMENT` columns as `0` without generating a new sequence value. See Section 5.1.11, “Server SQL Modes”.

  There can be only one `AUTO_INCREMENT` column per table, it must be indexed, and it cannot have a `DEFAULT` value. An `AUTO_INCREMENT` column works properly only if it contains only positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to avoid precision problems when numbers “wrap” over from positive to negative and also to ensure that you do not accidentally get an `AUTO_INCREMENT` column that contains `0`.

  For `MyISAM` tables, you can specify an `AUTO_INCREMENT` secondary column in a multiple-column key. See Section 3.6.9, “Using AUTO_INCREMENT”.

  To make MySQL compatible with some ODBC applications, you can find the `AUTO_INCREMENT` value for the last inserted row with the following query:

  ```
  SELECT * FROM tbl_name WHERE auto_col IS NULL
  ```

  This method requires that `sql_auto_is_null` variable is not set to 0. See Section 5.1.8, “Server System Variables”.

  For information about `InnoDB` and `AUTO_INCREMENT`, see Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”. For information about `AUTO_INCREMENT` and MySQL Replication, see Section 17.5.1.1, “Replication and AUTO_INCREMENT”.

- `COMMENT`

  A comment for a column can be specified with the `COMMENT` option, up to 1024 characters long. The comment is displayed by the `SHOW CREATE TABLE` and `SHOW FULL COLUMNS` statements.

- `COLUMN_FORMAT`

  In NDB Cluster, it is also possible to specify a data storage format for individual columns of `NDB` tables using `COLUMN_FORMAT`. Permissible column formats are `FIXED`, `DYNAMIC`, and `DEFAULT`. `FIXED` is used to specify fixed-width storage, `DYNAMIC` permits the column to be variable-width, and `DEFAULT`

causes the column to use fixed-width or variable-width storage as determined by the column's data type (possibly overridden by a `ROW_FORMAT` specifier).

For `NDB` tables, the default value for `COLUMN_FORMAT` is `FIXED`.

In NDB Cluster, the maximum possible offset for a column defined with `COLUMN_FORMAT=FIXED` is 8188 bytes. For more information and possible workarounds, see Section 22.1.7.5, “Limits Associated with Database Objects in NDB Cluster”.

`COLUMN_FORMAT` currently has no effect on columns of tables using storage engines other than `NDB`. MySQL 8.0 silently ignores `COLUMN_FORMAT`.

- `STORAGE`

  For `NDB` tables, it is possible to specify whether the column is stored on disk or in memory by using a `STORAGE` clause. `STORAGE DISK` causes the column to be stored on disk, and `STORAGE MEMORY` causes in-memory storage to be used. The `CREATE TABLE` statement used must still include a `TABLESPACE` clause:

```
mysql> CREATE TABLE t1 (
    ->     c1 INT STORAGE DISK,
    ->     c2 INT STORAGE MEMORY
    -> ) ENGINE NDB;
ERROR 1005 (HY000): Can't create table 'c.t1' (errno: 140)

mysql> CREATE TABLE t1 (
    ->     c1 INT STORAGE DISK,
    ->     c2 INT STORAGE MEMORY
    -> ) TABLESPACE ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.06 sec)
```

  For `NDB` tables, `STORAGE DEFAULT` is equivalent to `STORAGE MEMORY`.

  The `STORAGE` clause has no effect on tables using storage engines other than `NDB`. The `STORAGE` keyword is supported only in the build of `mysqld` that is supplied with NDB Cluster; it is not recognized in any other version of MySQL, where any attempt to use the `STORAGE` keyword causes a syntax error.

- `GENERATED ALWAYS`

  Used to specify a generated column expression. For information about generated columns, see Section 13.1.20.9, “CREATE TABLE and Generated Columns”.

  Stored generated columns can be indexed. `InnoDB` supports secondary indexes on virtual generated columns. See Section 13.1.20.10, “Secondary Indexes and Generated Columns”.

Indexes, Foreign Keys, and CHECK Constraints

Several keywords apply to creation of indexes, foreign keys, and `CHECK` constraints. For general background in addition to the following descriptions, see Section 13.1.15, “CREATE INDEX Syntax”, Section 13.1.20.6, “Using FOREIGN KEY Constraints”, and Section 13.1.20.7, “CHECK Constraints”.

- `CONSTRAINT` *`symbol`*

  The `CONSTRAINT` *`symbol`* clause may be given to name a constraint. If the clause is not given, or a *`symbol`* is not included following the `CONSTRAINT` keyword, MySQL automatically generates a constraint name, with the exception noted below. The *`symbol`* value, if used, must be unique per schema (database), per constraint type. A duplicate *`symbol`* results in an error.

Note

If the `CONSTRAINT symbol` clause is not given in a foreign key definition, or a `symbol` is not included following the `CONSTRAINT` keyword, MySQL uses the foreign key index name up to MySQL 8.0.15, and automatically generates a constraint name thereafter.

The SQL standard specifies that all types of constraints (primary key, unique index, foreign key, check) belong to the same namespace. In MySQL, each constraint type has its own namespace per schema. Consequently, names for each type of constraint must be unique per schema.

- `PRIMARY KEY`

  A unique index where all key columns must be defined as `NOT NULL`. If they are not explicitly declared as `NOT NULL`, MySQL declares them so implicitly (and silently). A table can have only one `PRIMARY KEY`. The name of a `PRIMARY KEY` is always `PRIMARY`, which thus cannot be used as the name for any other kind of index.

  If you do not have a `PRIMARY KEY` and an application asks for the `PRIMARY KEY` in your tables, MySQL returns the first `UNIQUE` index that has no `NULL` columns as the `PRIMARY KEY`.

  In `InnoDB` tables, keep the `PRIMARY KEY` short to minimize storage overhead for secondary indexes. Each secondary index entry contains a copy of the primary key columns for the corresponding row. (See Section 15.6.2.1, “Clustered and Secondary Indexes”.)

  In the created table, a `PRIMARY KEY` is placed first, followed by all `UNIQUE` indexes, and then the nonunique indexes. This helps the MySQL optimizer to prioritize which index to use and also more quickly to detect duplicated `UNIQUE` keys.

  A `PRIMARY KEY` can be a multiple-column index. However, you cannot create a multiple-column index using the `PRIMARY KEY` key attribute in a column specification. Doing so only marks that single column as primary. You must use a separate `PRIMARY KEY(key_part, ...)` clause.

  If a table has a `PRIMARY KEY` or `UNIQUE NOT NULL` index that consists of a single column that has an integer type, you can use `_rowid` to refer to the indexed column in `SELECT` statements, as described in Unique Indexes.

  In MySQL, the name of a `PRIMARY KEY` is `PRIMARY`. For other indexes, if you do not assign a name, the index is assigned the same name as the first indexed column, with an optional suffix (`_2`, `_3`, `...`) to make it unique. You can see index names for a table using `SHOW INDEX FROM tbl_name`. See Section 13.7.7.22, “SHOW INDEX Syntax”.

- `KEY | INDEX`

  `KEY` is normally a synonym for `INDEX`. The key attribute `PRIMARY KEY` can also be specified as just `KEY` when given in a column definition. This was implemented for compatibility with other database systems.

- `UNIQUE`

  A `UNIQUE` index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. For all engines, a `UNIQUE` index permits multiple `NULL` values for columns that can contain `NULL`. If you specify a prefix value for a column in a `UNIQUE` index, the column values must be unique within the prefix length.

If a table has a PRIMARY KEY or UNIQUE NOT NULL index that consists of a single column that has an integer type, you can use _rowid to refer to the indexed column in SELECT statements, as described in Unique Indexes.

- FULLTEXT

  A FULLTEXT index is a special type of index used for full-text searches. Only the InnoDB and MyISAM storage engines support FULLTEXT indexes. They can be created only from CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 12.9, “Full-Text Search Functions”, for details of operation. A WITH PARSER clause can be specified as an *index_option* value to associate a parser plugin with the index if full-text indexing and searching operations need special handling. This clause is valid only for FULLTEXT indexes. InnoDB and MyISAM support full-text parser plugins. See Full-Text Parser Plugins and Section 29.2.4.4, “Writing Full-Text Parser Plugins” for more information.

- SPATIAL

  You can create SPATIAL indexes on spatial data types. Spatial types are supported only for InnoDB and MyISAM tables, and indexed columns must be declared as NOT NULL. See Section 11.5, “Spatial Data Types”.

- FOREIGN KEY

  MySQL supports foreign keys, which let you cross-reference related data across tables, and foreign key constraints, which help keep this spread-out data consistent. For definition and option information, see *reference_definition*, and *reference_option*.

  Partitioned tables employing the InnoDB storage engine do not support foreign keys. See Section 23.6, “Restrictions and Limitations on Partitioning”, for more information.

- CHECK

  The CHECK clause enables the creation of constraints to be checked for data values in table rows. See Section 13.1.20.7, “CHECK Constraints”.

- *key_part*

  - A *key_part* specification can end with ASC or DESC to specify whether index values are stored in ascending or descending order. The default is ascending if no order specifier is given.

  - Prefixes, defined by the *length* attribute, can be up to 767 bytes long for InnoDB tables that use the REDUNDANT or COMPACT row format. The prefix length limit is 3072 bytes for InnoDB tables that use the DYNAMIC or COMPRESSED row format. For MyISAM tables, the prefix length limit is 1000 bytes.

    Prefix *limits* are measured in bytes. However, prefix *lengths* for index specifications in CREATE TABLE, ALTER TABLE, and CREATE INDEX statements are interpreted as number of characters for nonbinary string types (CHAR, VARCHAR, TEXT) and number of bytes for binary string types (BINARY, VARBINARY, BLOB). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

  - Beginning with MySQL 8.0.17, the *expr* for a *key_part* specification can take the form (CAST *json_path* AS *type* ARRAY) to create a multi-valued index on a JSON column. Multi-Valued Indexes, provides detailed information regarding creation of, usage of, and restrictions and limitations on multi-valued indexes.

- *index_type*

Some storage engines permit you to specify an index type when creating an index. The syntax for the *index_type* specifier is USING *type_name*.

Example:

```
CREATE TABLE lookup
  (id INT, INDEX USING BTREE (id))
  ENGINE = MEMORY;
```

The preferred position for USING is after the index column list. It can be given before the column list, but support for use of the option in that position is deprecated and will be removed in a future MySQL release.

- *index_option*

  index_option values specify additional options for an index.

  - KEY_BLOCK_SIZE

    For MyISAM tables, KEY_BLOCK_SIZE optionally specifies the size in bytes to use for index key blocks. The value is treated as a hint; a different size could be used if necessary. A KEY_BLOCK_SIZE value specified for an individual index definition overrides the table-level KEY_BLOCK_SIZE value.

    For information about the table-level KEY_BLOCK_SIZE attribute, see Table Options.

  - WITH PARSER

    The WITH PARSER option can only be used with FULLTEXT indexes. It associates a parser plugin with the index if full-text indexing and searching operations need special handling. InnoDB and MyISAM support full-text parser plugins. If you have a MyISAM table with an associated full-text parser plugin, you can convert the table to InnoDB using ALTER TABLE.

  - COMMENT

    In MySQL 8.0, index definitions can include an optional comment of up to 1024 characters.

    You can set the InnoDB MERGE_THRESHOLD value for an individual index using the *index_option* COMMENT clause. See Section 15.8.11, “Configuring the Merge Threshold for Index Pages”.

  For more information about permissible *index_option* values, see Section 13.1.15, “CREATE INDEX Syntax”. For more information about indexes, see Section 8.3.1, “How MySQL Uses Indexes”.

- *reference_definition*

  For *reference_definition* syntax details and examples, see Section 13.1.20.6, “Using FOREIGN KEY Constraints”. For information specific to foreign keys in InnoDB, see Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”.

  InnoDB and NDB tables support checking of foreign key constraints. The columns of the referenced table must always be explicitly named. Both ON DELETE and ON UPDATE actions on foreign keys are supported. For more detailed information and examples, see Section 13.1.20.6, “Using FOREIGN KEY Constraints”. For information specific to foreign keys in InnoDB, see Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”.

  For other storage engines, MySQL Server parses and ignores the FOREIGN KEY and REFERENCES syntax in CREATE TABLE statements. See Section 1.8.2.3, “Foreign Key Differences”.

Important

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including `InnoDB`, recognizes or enforces the `MATCH` clause used in referential integrity constraint definitions. Use of an explicit `MATCH` clause will not have the specified effect, and also causes `ON DELETE` and `ON UPDATE` clauses to be ignored. For these reasons, specifying `MATCH` should be avoided.

The `MATCH` clause in the SQL standard controls how `NULL` values in a composite (multiple-column) foreign key are handled when comparing to a primary key. `InnoDB` essentially implements the semantics defined by `MATCH SIMPLE`, which permit a foreign key to be all or partially `NULL`. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table. It is possible to implement other semantics using triggers.

Additionally, MySQL requires that the referenced columns be indexed for performance. However, `InnoDB` does not enforce any requirement that the referenced columns be declared `UNIQUE` or `NOT NULL`. The handling of foreign key references to nonunique keys or keys that contain `NULL` values is not well defined for operations such as `UPDATE` or `DELETE CASCADE`. You are advised to use foreign keys that reference only keys that are both `UNIQUE` (or `PRIMARY`) and `NOT NULL`.

MySQL parses but ignores “inline `REFERENCES` specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. MySQL accepts `REFERENCES` clauses only when specified as part of a separate `FOREIGN KEY` specification.

- *`reference_option`*

  For information about the `RESTRICT`, `CASCADE`, `SET NULL`, `NO ACTION`, and `SET DEFAULT` options, see Section 13.1.20.6, “Using FOREIGN KEY Constraints”.

Table Options

Table options are used to optimize the behavior of the table. In most cases, you do not have to specify any of them. These options apply to all storage engines unless otherwise indicated. Options that do not apply to a given storage engine may be accepted and remembered as part of the table definition. Such options then apply if you later use `ALTER TABLE` to convert the table to use a different storage engine.

- `ENGINE`

  Specifies the storage engine for the table, using one of the names shown in the following table. The engine name can be unquoted or quoted. The quoted name `'DEFAULT'` is recognized but ignored.

Storage Engine	Description
`InnoDB`	Transaction-safe tables with row locking and foreign keys. The default storage engine for new tables. See Chapter 15, *The InnoDB Storage Engine*, and in particular Section 15.1, “Introduction to InnoDB” if you have MySQL experience but are new to `InnoDB`.
`MyISAM`	The binary portable storage engine that is primarily used for read-only or read-mostly workloads. See Section 16.2, “The MyISAM Storage Engine”.

Storage Engine	Description
MEMORY	The data for this storage engine is stored only in memory. See Section 16.3, “The MEMORY Storage Engine”.
CSV	Tables that store rows in comma-separated values format. See Section 16.4, “The CSV Storage Engine”.
ARCHIVE	The archiving storage engine. See Section 16.5, “The ARCHIVE Storage Engine”.
EXAMPLE	An example engine. See Section 16.9, “The EXAMPLE Storage Engine”.
FEDERATED	Storage engine that accesses remote tables. See Section 16.8, “The FEDERATED Storage Engine”.
HEAP	This is a synonym for MEMORY.
MERGE	A collection of MyISAM tables used as one table. Also known as MRG_MyISAM. See Section 16.7, “The MERGE Storage Engine”.
NDB	Clustered, fault-tolerant, memory-based tables, supporting transactions and foreign keys. Also known as NDBCLUSTER. See Chapter 22, *MySQL NDB Cluster 8.0*.

By default, if a storage engine is specified that is not available, the statement fails with an error. You can override this behavior by removing NO_ENGINE_SUBSTITUTION from the server SQL mode (see Section 5.1.11, “Server SQL Modes”) so that MySQL allows substitution of the specified engine with the default storage engine instead. Normally in such cases, this is InnoDB, which is the default value for the default_storage_engine system variable. When NO_ENGINE_SUBSTITUTION is disabled, a warning occurs if the storage engine specification is not honored.

- AUTO_INCREMENT

  The initial AUTO_INCREMENT value for the table. In MySQL 8.0, this works for MyISAM, MEMORY, InnoDB, and ARCHIVE tables. To set the first auto-increment value for engines that do not support the AUTO_INCREMENT table option, insert a “dummy” row with a value one less than the desired value after creating the table, and then delete the dummy row.

  For engines that support the AUTO_INCREMENT table option in CREATE TABLE statements, you can also use ALTER TABLE *tbl_name* AUTO_INCREMENT = *N* to reset the AUTO_INCREMENT value. The value cannot be set lower than the maximum value currently in the column.

- AVG_ROW_LENGTH

  An approximation of the average row length for your table. You need to set this only for large tables with variable-size rows.

  When you create a MyISAM table, MySQL uses the product of the MAX_ROWS and AVG_ROW_LENGTH options to decide how big the resulting table is. If you don't specify either option, the maximum size for MyISAM data and index files is 256TB by default. (If your operating system does not support files that large, table sizes are constrained by the file size limit.) If you want to keep down the pointer sizes to make the index smaller and faster and you don't really need big files, you can decrease the default pointer size by setting the myisam_data_pointer_size system variable. (See Section 5.1.8, “Server System Variables”.) If you want all your tables to be able to grow above the default limit and are willing to have your tables slightly slower and larger than necessary, you can increase the default pointer size by setting this variable. Setting the value to 7 permits table sizes up to 65,536TB.

- [DEFAULT] CHARACTER SET

Specifies a default character set for the table. CHARSET is a synonym for CHARACTER SET. If the character set name is DEFAULT, the database character set is used.

- CHECKSUM

  Set this to 1 if you want MySQL to maintain a live checksum for all rows (that is, a checksum that MySQL updates automatically as the table changes). This makes the table a little slower to update, but also makes it easier to find corrupted tables. The CHECKSUM TABLE statement reports the checksum. (MyISAM only.)

- [DEFAULT] COLLATE

  Specifies a default collation for the table.

- COMMENT

  A comment for the table, up to 2048 characters long.

  You can set the InnoDB MERGE_THRESHOLD value for a table using the *table_option* COMMENT clause. See Section 15.8.11, “Configuring the Merge Threshold for Index Pages”.

  Setting NDB_TABLE options. The table comment in a CREATE TABLE that creates an NDB table or an ALTER TABLE statement which alters one can also be used to specify one to four of the NDB_TABLE options NOLOGGING, READ_BACKUP, PARTITION_BALANCE, or FULLY_REPLICATED as a set of name-value pairs, separated by commas if need be, immediately following the string NDB_TABLE= that begins the quoted comment text. An example statement using this syntax is shown here (emphasized text):

  ```
  CREATE TABLE t1 (
      c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
      c2 VARCHAR(100),
      c3 VARCHAR(100) )
  ENGINE=NDB
  COMMENT="NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RP_BY_NODE";
  ```

  Spaces are not permitted within the quoted string. The string is case-insensitive.

  The comment is displayed as part of the ouput of SHOW CREATE TABLE. The text of the comment is also available as the TABLE_COMMENT column of the MySQL Information Schema TABLES table.

  This comment syntax is also supported with ALTER TABLE statements for NDB tables. Keep in mind that a table comment used with ALTER TABLE replaces any existing comment which the table might have had perviously.

  Setting the MERGE_THRESHOLD option in table comments is not supported for NDB tables (it is ignored).

  For complete syntax information and examples, see Section 13.1.20.11, “Setting NDB_TABLE Options”.

- COMPRESSION

  The compression algorithm used for page level compression for InnoDB tables. Supported values include Zlib, LZ4, and None. The COMPRESSION attribute was introduced with the transparent page compression feature. Page compression is only supported with InnoDB tables that reside in file-per-table tablespaces, and is only available on Linux and Windows platforms that support sparse files and hole punching. For more information, see Section 15.9.2, “InnoDB Page Compression”.

- CONNECTION

The connection string for a FEDERATED table.

Note

Older versions of MySQL used a COMMENT option for the connection string.

- DATA DIRECTORY, INDEX DIRECTORY

  For InnoDB, the DATA DIRECTORY='*directory*' clause permits creating a file-per-table tablespace outside of the data directory. The tablespace data file is created in the specified directory, inside a subdirectory with the same name as the schema. The innodb_file_per_table variable must be enabled to use the DATA DIRECTORY clause. The full directory path must be specified. For more information, see Section 15.6.3.6, "Creating a Tablespace Outside of the Data Directory".

  When creating MyISAM tables, you can use the DATA DIRECTORY='*directory*' clause, the INDEX DIRECTORY='*directory*' clause, or both. They specify where to put a MyISAM table's data file and index file, respectively. Unlike InnoDB tables, MySQL does not create subdirectories that correspond to the database name when creating a MyISAM table with a DATA DIRECTORY or INDEX DIRECTORY option. Files are created in the directory that is specified.

  You must have the FILE privilege to use the DATA DIRECTORY or INDEX DIRECTORY table option.

  Important

  Table-level DATA DIRECTORY and INDEX DIRECTORY options are ignored for partitioned tables. (Bug #32091)

  These options work only when you are not using the --skip-symbolic-links option. Your operating system must also have a working, thread-safe realpath() call. See Section 8.12.2.2, "Using Symbolic Links for MyISAM Tables on Unix", for more complete information.

  If a MyISAM table is created with no DATA DIRECTORY option, the .MYD file is created in the database directory. By default, if MyISAM finds an existing .MYD file in this case, it overwrites it. The same applies to .MYI files for tables created with no INDEX DIRECTORY option. To suppress this behavior, start the server with the --keep_files_on_create option, in which case MyISAM will not overwrite existing files and returns an error instead.

  If a MyISAM table is created with a DATA DIRECTORY or INDEX DIRECTORY option and an existing .MYD or .MYI file is found, MyISAM always returns an error. It will not overwrite a file in the specified directory.

  Important

  You cannot use path names that contain the MySQL data directory with DATA DIRECTORY or INDEX DIRECTORY. This includes partitioned tables and individual table partitions. (See Bug #32167.)

- DELAY_KEY_WRITE

  Set this to 1 if you want to delay key updates for the table until the table is closed. See the description of the delay_key_write system variable in Section 5.1.8, "Server System Variables". (MyISAM only.)

- ENCRYPTION

The ENCRYPTION clause enables or disables page-level data encryption for an InnoDB table. A keyring plugin must be installed and configured before encryption can be enabled. Prior to MySQL 8.0.16, the ENCRYPTION clause can only be specified when creating a table in an a file-per-table tablespace. As of MySQL 8.0.16, the ENCRYPTION clause can also be specified when creating a table in a general tablespace.

As of MySQL 8.0.16, a table inherits the default schema encryption if an ENCRYPTION clause is not specified. If the table_encryption_privilege_check variable is enabled, the TABLE_ENCRYPTION_ADMIN privilege is required to create a table with an ENCRYPTION clause setting that differs from the default schema encryption. When creating a table in a general tablespace, table and tablespace encryption must match.

As of MySQL 8.0.16, specifying an ENCRYPTION clause with a value other than 'N' or '' is not permitted when using a storage engine that does not support encryption. Previously, the clause was accepted.

For more information, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”.

- INSERT_METHOD

  If you want to insert data into a MERGE table, you must specify with INSERT_METHOD the table into which the row should be inserted. INSERT_METHOD is an option useful for MERGE tables only. Use a value of FIRST or LAST to have inserts go to the first or last table, or a value of NO to prevent inserts. See Section 16.7, “The MERGE Storage Engine”.

- KEY_BLOCK_SIZE

  For MyISAM tables, KEY_BLOCK_SIZE optionally specifies the size in bytes to use for index key blocks. The value is treated as a hint; a different size could be used if necessary. A KEY_BLOCK_SIZE value specified for an individual index definition overrides the table-level KEY_BLOCK_SIZE value.

  For InnoDB tables, KEY_BLOCK_SIZE specifies the page size in kilobytes to use for compressed InnoDB tables. The KEY_BLOCK_SIZE value is treated as a hint; a different size could be used by InnoDB if necessary. KEY_BLOCK_SIZE can only be less than or equal to the innodb_page_size value. A value of 0 represents the default compressed page size, which is half of the innodb_page_size value. Depending on innodb_page_size, possible KEY_BLOCK_SIZE values include 0, 1, 2, 4, 8, and 16. See Section 15.9.1, “InnoDB Table Compression” for more information.

  Oracle recommends enabling innodb_strict_mode when specifying KEY_BLOCK_SIZE for InnoDB tables. When innodb_strict_mode is enabled, specifying an invalid KEY_BLOCK_SIZE value returns an error. If innodb_strict_mode is disabled, an invalid KEY_BLOCK_SIZE value results in a warning, and the KEY_BLOCK_SIZE option is ignored.

  The Create_options column in response to SHOW TABLE STATUS reports the actual KEY_BLOCK_SIZE used by the table, as does SHOW CREATE TABLE.

  InnoDB only supports KEY_BLOCK_SIZE at the table level.

  KEY_BLOCK_SIZE is not supported with 32KB and 64KB innodb_page_size values. InnoDB table compression does not support these pages sizes.

  InnoDB does not support the KEY_BLOCK_SIZE option when creating temporary tables.

- MAX_ROWS

The maximum number of rows you plan to store in the table. This is not a hard limit, but rather a hint to the storage engine that the table must be able to store at least this many rows.

Important

The use of `MAX_ROWS` with `NDB` tables to control the number of table partitions is deprecated. It remains supported in later versions for backward compatibility, but is subject to removal in a future release. Use PARTITION_BALANCE instead; see Setting NDB_TABLE options.

The `NDB` storage engine treats this value as a maximum. If you plan to create very large NDB Cluster tables (containing millions of rows), you should use this option to insure that `NDB` allocates sufficient number of index slots in the hash table used for storing hashes of the table's primary keys by setting `MAX_ROWS = 2 * rows`, where *`rows`* is the number of rows that you expect to insert into the table.

The maximum `MAX_ROWS` value is 4294967295; larger values are truncated to this limit.

- `MIN_ROWS`

  The minimum number of rows you plan to store in the table. The `MEMORY` storage engine uses this option as a hint about memory use.

- `PACK_KEYS`

  Takes effect only with `MyISAM` tables. Set this option to 1 if you want to have smaller indexes. This usually makes updates slower and reads faster. Setting the option to 0 disables all packing of keys. Setting it to `DEFAULT` tells the storage engine to pack only long `CHAR`, `VARCHAR`, `BINARY`, or `VARBINARY` columns.

  If you do not use `PACK_KEYS`, the default is to pack strings, but not numbers. If you use `PACK_KEYS=1`, numbers are packed as well.

  When packing binary number keys, MySQL uses prefix compression:

  - Every key needs one extra byte to indicate how many bytes of the previous key are the same for the next key.
  - The pointer to the row is stored in high-byte-first order directly after the key, to improve compression.

  This means that if you have many equal keys on two consecutive rows, all following “same” keys usually only take two bytes (including the pointer to the row). Compare this to the ordinary case where the following keys takes `storage_size_for_key + pointer_size` (where the pointer size is usually 4). Conversely, you get a significant benefit from prefix compression only if you have many numbers that are the same. If all keys are totally different, you use one byte more per key, if the key is not a key that can have `NULL` values. (In this case, the packed key length is stored in the same byte that is used to mark if a key is `NULL`.)

- `PASSWORD`

  This option is unused.

- `ROW_FORMAT`

  Defines the physical format in which the rows are stored.

When executing a CREATE TABLE statement with strict mode disabled, if you specify a row format that is not supported by the storage engine that is used for the table, the table is created using that storage engine's default row format. The actual row format of the table is reported in the Row_format and Create_options columns in response to SHOW TABLE STATUS. SHOW CREATE TABLE also reports the actual row format of the table.

Row format choices differ depending on the storage engine used for the table.

For InnoDB tables:

- The default row format is defined by innodb_default_row_format, which has a default setting of DYNAMIC. The default row format is used when the ROW_FORMAT option is not defined or when ROW_FORMAT=DEFAULT is used.

  If the ROW_FORMAT option is not defined, or if ROW_FORMAT=DEFAULT is used, operations that rebuild a table also silently change the row format of the table to the default defined by innodb_default_row_format. For more information, see Defining the Row Format of a Table.

- For more efficient InnoDB storage of data types, especially BLOB types, use the DYNAMIC. See DYNAMIC Row Format for requirements associated with the DYNAMIC row format.

- To enable compression for InnoDB tables, specify ROW_FORMAT=COMPRESSED. The ROW_FORMAT=COMPRESSED option is not supported when creating temporary tables. See Section 15.9, “InnoDB Table and Page Compression” for requirements associated with the COMPRESSED row format.

- The row format used in older versions of MySQL can still be requested by specifying the REDUNDANT row format.

- When you specify a non-default ROW_FORMAT clause, consider also enabling the innodb_strict_mode configuration option.

- ROW_FORMAT=FIXED is not supported. If ROW_FORMAT=FIXED is specified while innodb_strict_mode is disabled, InnoDB issues a warning and assumes ROW_FORMAT=DYNAMIC. If ROW_FORMAT=FIXED is specified while innodb_strict_mode is enabled, which is the default, InnoDB returns an error.

- For additional information about InnoDB row formats, see Section 15.10, “InnoDB Row Formats”.

For MyISAM tables, the option value can be FIXED or DYNAMIC for static or variable-length row format. myisampack sets the type to COMPRESSED. See Section 16.2.3, “MyISAM Table Storage Formats”.

For NDB tables, the default ROW_FORMAT is DYNAMIC.

- STATS_AUTO_RECALC

  Specifies whether to automatically recalculate persistent statistics for an InnoDB table. The value DEFAULT causes the persistent statistics setting for the table to be determined by the innodb_stats_auto_recalc configuration option. The value 1 causes statistics to be recalculated when 10% of the data in the table has changed. The value 0 prevents automatic recalculation for this table; with this setting, issue an ANALYZE TABLE statement to recalculate the statistics after making substantial changes to the table. For more information about the persistent statistics feature, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

- STATS_PERSISTENT

Specifies whether to enable persistent statistics for an `InnoDB` table. The value `DEFAULT` causes the persistent statistics setting for the table to be determined by the `innodb_stats_persistent` configuration option. The value `1` enables persistent statistics for the table, while the value `0` turns off this feature. After enabling persistent statistics through a `CREATE TABLE` or `ALTER TABLE` statement, issue an `ANALYZE TABLE` statement to calculate the statistics, after loading representative data into the table. For more information about the persistent statistics feature, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

- `STATS_SAMPLE_PAGES`

  The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by `ANALYZE TABLE`. For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

- `TABLESPACE`

  The `TABLESPACE` clause can be used to create a table in an existing general tablespace, a file-per-table tablespace, or the system tablespace.

```
CREATE TABLE tbl_name ... TABLESPACE [=] tablespace_name
```

  The general tablespace that you specify must exist prior to using the `TABLESPACE` clause. For information about general tablespaces, see Section 15.6.3.3, “General Tablespaces”.

  The *`tablespace_name`* is a case-sensitive identifier. It may be quoted or unquoted. The forward slash character (“/”) is not permitted. Names beginning with “innodb_” are reserved for special use.

  To create a table in the system tablespace, specify `innodb_system` as the tablespace name.

```
CREATE TABLE tbl_name ... TABLESPACE [=] innodb_system
```

  Using `TABLESPACE [=] innodb_system`, you can place a table of any uncompressed row format in the system tablespace regardless of the `innodb_file_per_table` setting. For example, you can add a table with `ROW_FORMAT=DYNAMIC` to the system tablespace using `TABLESPACE [=] innodb_system`.

  To create a table in a file-per-table tablespace, specify `innodb_file_per_table` as the tablespace name.

```
CREATE TABLE tbl_name ... TABLESPACE [=] innodb_file_per_table
```

Note

If `innodb_file_per_table` is enabled, you need not specify `TABLESPACE=innodb_file_per_table` to create an `InnoDB` file-per-table tablespace. `InnoDB` tables are created in file-per-table tablespaces by default when `innodb_file_per_table` is enabled.

  The `DATA DIRECTORY` clause is permitted with `CREATE TABLE ... TABLESPACE=innodb_file_per_table` but is otherwise not supported for use in combination with the `TABLESPACE` clause.

Note

Support for `TABLESPACE = innodb_file_per_table` and `TABLESPACE = innodb_temporary` clauses with `CREATE TEMPORARY TABLE` is deprecated as of MySQL 8.0.13 and will be removed in a future version of MySQL.

The `STORAGE` table option is employed only with `NDB` tables. `STORAGE` determines the type of storage used (disk or memory), and can be either `DISK` or `MEMORY`.

`TABLESPACE ... STORAGE DISK` assigns a table to an NDB Cluster Disk Data tablespace. The tablespace must already have been created using `CREATE TABLESPACE`. See Section 22.5.13, "NDB Cluster Disk Data Tables", for more information.

Important

A `STORAGE` clause cannot be used in a `CREATE TABLE` statement without a `TABLESPACE` clause.

- `UNION`

  Used to access a collection of identical `MyISAM` tables as one. This works only with `MERGE` tables. See Section 16.7, "The MERGE Storage Engine".

  You must have `SELECT`, `UPDATE`, and `DELETE` privileges for the tables you map to a `MERGE` table.

Note

Formerly, all tables used had to be in the same database as the `MERGE` table itself. This restriction no longer applies.

Table Partitioning

`partition_options` can be used to control partitioning of the table created with `CREATE TABLE`.

Not all options shown in the syntax for *`partition_options`* at the beginning of this section are available for all partitioning types. Please see the listings for the following individual types for information specific to each type, and see Chapter 23, *Partitioning*, for more complete information about the workings of and uses for partitioning in MySQL, as well as additional examples of table creation and other statements relating to MySQL partitioning.

Partitions can be modified, merged, added to tables, and dropped from tables. For basic information about the MySQL statements to accomplish these tasks, see Section 13.1.9, "ALTER TABLE Syntax". For more detailed descriptions and examples, see Section 23.3, "Partition Management".

- `PARTITION BY`

  If used, a *`partition_options`* clause begins with `PARTITION BY`. This clause contains the function that is used to determine the partition; the function returns an integer value ranging from 1 to *`num`*, where *`num`* is the number of partitions. (The maximum number of user-defined partitions which a table may contain is 1024; the number of subpartitions—discussed later in this section—is included in this maximum.)

Note

The expression (*expr*) used in a PARTITION BY clause cannot refer to any columns not in the table being created; such references are specifically not permitted and cause the statement to fail with an error. (Bug #29444)

- HASH(*expr*)

  Hashes one or more columns to create a key for placing and locating rows. *expr* is an expression using one or more table columns. This can be any valid MySQL expression (including MySQL functions) that yields a single integer value. For example, these are both valid CREATE TABLE statements using PARTITION BY HASH:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5))
    PARTITION BY HASH(col1);

CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATETIME)
    PARTITION BY HASH ( YEAR(col3) );
```

  You may not use either VALUES LESS THAN or VALUES IN clauses with PARTITION BY HASH.

  PARTITION BY HASH uses the remainder of *expr* divided by the number of partitions (that is, the modulus). For examples and additional information, see Section 23.2.4, “HASH Partitioning”.

  The LINEAR keyword entails a somewhat different algorithm. In this case, the number of the partition in which a row is stored is calculated as the result of one or more logical AND operations. For discussion and examples of linear hashing, see Section 23.2.4.1, “LINEAR HASH Partitioning”.

- KEY(*column_list*)

  This is similar to HASH, except that MySQL supplies the hashing function so as to guarantee an even data distribution. The *column_list* argument is simply a list of 1 or more table columns (maximum: 16). This example shows a simple table partitioned by key, with 4 partitions:

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY KEY(col3)
    PARTITIONS 4;
```

  For tables that are partitioned by key, you can employ linear partitioning by using the LINEAR keyword. This has the same effect as with tables that are partitioned by HASH. That is, the partition number is found using the & operator rather than the modulus (see Section 23.2.4.1, “LINEAR HASH Partitioning”, and Section 23.2.5, “KEY Partitioning”, for details). This example uses linear partitioning by key to distribute data between 5 partitions:

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY LINEAR KEY(col3)
    PARTITIONS 5;
```

  The ALGORITHM={1|2} option is supported with [SUB]PARTITION BY [LINEAR] KEY. ALGORITHM=1 causes the server to use the same key-hashing functions as MySQL 5.1; ALGORITHM=2 means that the server employs the key-hashing functions implemented and used by default for new KEY partitioned tables in MySQL 5.5 and later. (Partitioned tables created with the key-hashing functions employed in MySQL 5.5 and later cannot be used by a MySQL 5.1 server.) Not specifying the option has the same effect as using ALGORITHM=2. This option is intended for use chiefly when upgrading or downgrading [LINEAR] KEY partitioned tables between MySQL 5.1 and later MySQL versions, or for

creating tables partitioned by KEY or LINEAR KEY on a MySQL 5.5 or later server which can be used on a MySQL 5.1 server. For more information, see Section 13.1.9.1, “ALTER TABLE Partition Operations”.

mysqldump in MySQL 5.7 (and later) writes this option encased in versioned comments, like this:

```
CREATE TABLE t1 (a INT)
/*!50100 PARTITION BY KEY */ /*!50611 ALGORITHM = 1 */ /*!50100 ()
      PARTITIONS 3 */
```

This causes MySQL 5.6.10 and earlier servers to ignore the option, which would otherwise cause a syntax error in those versions. If you plan to load a dump made on a MySQL 5.7 server where you use tables that are partitioned or subpartitioned by KEY into a MySQL 5.6 server previous to version 5.6.11, be sure to consult Changes in MySQL 5.6, before proceeding. (The information found there also applies if you are loading a dump containing KEY partitioned or subpartitioned tables made from a MySQL 5.7—actually 5.6.11 or later—server into a MySQL 5.5.30 or earlier server.)

Also in MySQL 5.6.11 and later, ALGORITHM=1 is shown when necessary in the output of SHOW CREATE TABLE using versioned comments in the same manner as mysqldump. ALGORITHM=2 is always omitted from SHOW CREATE TABLE output, even if this option was specified when creating the original table.

You may not use either VALUES LESS THAN or VALUES IN clauses with PARTITION BY KEY.

- RANGE(*expr*)

In this case, *expr* shows a range of values using a set of VALUES LESS THAN operators. When using range partitioning, you must define at least one partition using VALUES LESS THAN. You cannot use VALUES IN with range partitioning.

Note

For tables partitioned by RANGE, VALUES LESS THAN must be used with either an integer literal value or an expression that evaluates to a single integer value. In MySQL 8.0, you can overcome this limitation in a table that is defined using PARTITION BY RANGE COLUMNS, as described later in this section.

Suppose that you have a table that you wish to partition on a column containing year values, according to the following scheme.

Partition Number:	Years Range:
0	1990 and earlier
1	1991 to 1994
2	1995 to 1998
3	1999 to 2002
4	2003 to 2005
5	2006 and later

A table implementing such a partitioning scheme can be realized by the CREATE TABLE statement shown here:

```
CREATE TABLE t1 (
    year_col  INT,
    some_data INT
```

```
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (1999),
    PARTITION p3 VALUES LESS THAN (2002),
    PARTITION p4 VALUES LESS THAN (2006),
    PARTITION p5 VALUES LESS THAN MAXVALUE
);
```

`PARTITION ... VALUES LESS THAN ...` statements work in a consecutive fashion. `VALUES LESS THAN MAXVALUE` works to specify “leftover” values that are greater than the maximum value otherwise specified.

`VALUES LESS THAN` clauses work sequentially in a manner similar to that of the `case` portions of a `switch ... case` block (as found in many programming languages such as C, Java, and PHP). That is, the clauses must be arranged in such a way that the upper limit specified in each successive `VALUES LESS THAN` is greater than that of the previous one, with the one referencing `MAXVALUE` coming last of all in the list.

- `RANGE COLUMNS(`*`column_list`*`)`

  This variant on `RANGE` facilitates partition pruning for queries using range conditions on multiple columns (that is, having conditions such as `WHERE a = 1 AND b < 10` or `WHERE a = 1 AND b = 10 AND c < 10`). It enables you to specify value ranges in multiple columns by using a list of columns in the `COLUMNS` clause and a set of column values in each `PARTITION ... VALUES LESS THAN (`*`value_list`*`)` partition definition clause. (In the simplest case, this set consists of a single column.) The maximum number of columns that can be referenced in the *`column_list`* and *`value_list`* is 16.

  The *`column_list`* used in the `COLUMNS` clause may contain only names of columns; each column in the list must be one of the following MySQL data types: the integer types; the string types; and time or date column types. Columns using `BLOB`, `TEXT`, `SET`, `ENUM`, `BIT`, or spatial data types are not permitted; columns that use floating-point number types are also not permitted. You also may not use functions or arithmetic expressions in the `COLUMNS` clause.

  The `VALUES LESS THAN` clause used in a partition definition must specify a literal value for each column that appears in the `COLUMNS()` clause; that is, the list of values used for each `VALUES LESS THAN` clause must contain the same number of values as there are columns listed in the `COLUMNS` clause. An attempt to use more or fewer values in a `VALUES LESS THAN` clause than there are in the `COLUMNS` clause causes the statement to fail with the error `Inconsistency in usage of column lists for partitioning...`. You cannot use `NULL` for any value appearing in `VALUES LESS THAN`. It is possible to use `MAXVALUE` more than once for a given column other than the first, as shown in this example:

  ```
  CREATE TABLE rc (
      a INT NOT NULL,
      b INT NOT NULL
  )
  PARTITION BY RANGE COLUMNS(a,b) (
      PARTITION p0 VALUES LESS THAN (10,5),
      PARTITION p1 VALUES LESS THAN (20,10),
      PARTITION p2 VALUES LESS THAN (50,MAXVALUE),
      PARTITION p3 VALUES LESS THAN (65,MAXVALUE),
      PARTITION p4 VALUES LESS THAN (MAXVALUE,MAXVALUE)
  );
  ```

  Each value used in a `VALUES LESS THAN` value list must match the type of the corresponding column exactly; no conversion is made. For example, you cannot use the string `'1'` for a value that matches a

column that uses an integer type (you must use the numeral `1` instead), nor can you use the numeral `1` for a value that matches a column that uses a string type (in such a case, you must use a quoted string: `'1'`).

For more information, see Section 23.2.1, “RANGE Partitioning”, and Section 23.4, “Partition Pruning”.

- `LIST(expr)`

  This is useful when assigning partitions based on a table column with a restricted set of possible values, such as a state or country code. In such a case, all rows pertaining to a certain state or country can be assigned to a single partition, or a partition can be reserved for a certain set of states or countries. It is similar to `RANGE`, except that only `VALUES IN` may be used to specify permissible values for each partition.

  `VALUES IN` is used with a list of values to be matched. For instance, you could create a partitioning scheme such as the following:

```
CREATE TABLE client_firms (
    id   INT,
    name VARCHAR(35)
)
PARTITION BY LIST (id) (
    PARTITION r0 VALUES IN (1, 5, 9, 13, 17, 21),
    PARTITION r1 VALUES IN (2, 6, 10, 14, 18, 22),
    PARTITION r2 VALUES IN (3, 7, 11, 15, 19, 23),
    PARTITION r3 VALUES IN (4, 8, 12, 16, 20, 24)
);
```

  When using list partitioning, you must define at least one partition using `VALUES IN`. You cannot use `VALUES LESS THAN` with `PARTITION BY LIST`.

  Note

  For tables partitioned by `LIST`, the value list used with `VALUES IN` must consist of integer values only. In MySQL 8.0, you can overcome this limitation using partitioning by `LIST COLUMNS`, which is described later in this section.

- `LIST COLUMNS(column_list)`

  This variant on `LIST` facilitates partition pruning for queries using comparison conditions on multiple columns (that is, having conditions such as `WHERE a = 5 AND b = 5` or `WHERE a = 1 AND b = 10 AND c = 5`). It enables you to specify values in multiple columns by using a list of columns in the `COLUMNS` clause and a set of column values in each `PARTITION ... VALUES IN (value_list)` partition definition clause.

  The rules governing regarding data types for the column list used in `LIST COLUMNS(column_list)` and the value list used in `VALUES IN(value_list)` are the same as those for the column list used in `RANGE COLUMNS(column_list)` and the value list used in `VALUES LESS THAN(value_list)`, respectively, except that in the `VALUES IN` clause, `MAXVALUE` is not permitted, and you may use `NULL`.

  There is one important difference between the list of values used for `VALUES IN` with `PARTITION BY LIST COLUMNS` as opposed to when it is used with `PARTITION BY LIST`. When used with `PARTITION BY LIST COLUMNS`, each element in the `VALUES IN` clause must be a *set* of column values; the number of values in each set must be the same as the number of columns used in the `COLUMNS` clause, and the data types of these values must match those of the columns (and occur in the same order). In the simplest case, the set consists of a single column. The maximum number of columns that can be used in the *`column_list`* and in the elements making up the *`value_list`* is 16.

The table defined by the following `CREATE TABLE` statement provides an example of a table using `LIST COLUMNS` partitioning:

```
CREATE TABLE lc (
    a INT NULL,
    b INT NULL
)
PARTITION BY LIST COLUMNS(a,b) (
    PARTITION p0 VALUES IN( (0,0), (NULL,NULL) ),
    PARTITION p1 VALUES IN( (0,1), (0,2), (0,3), (1,1), (1,2) ),
    PARTITION p2 VALUES IN( (1,0), (2,0), (2,1), (3,0), (3,1) ),
    PARTITION p3 VALUES IN( (1,3), (2,2), (2,3), (3,2), (3,3) )
);
```

- `PARTITIONS` *`num`*

  The number of partitions may optionally be specified with a `PARTITIONS` *`num`* clause, where *`num`* is the number of partitions. If both this clause *and* any `PARTITION` clauses are used, *`num`* must be equal to the total number of any partitions that are declared using `PARTITION` clauses.

  Note

  Whether or not you use a `PARTITIONS` clause in creating a table that is partitioned by `RANGE` or `LIST`, you must still include at least one `PARTITION VALUES` clause in the table definition (see below).

- `SUBPARTITION BY`

  A partition may optionally be divided into a number of subpartitions. This can be indicated by using the optional `SUBPARTITION BY` clause. Subpartitioning may be done by `HASH` or `KEY`. Either of these may be `LINEAR`. These work in the same way as previously described for the equivalent partitioning types. (It is not possible to subpartition by `LIST` or `RANGE`.)

  The number of subpartitions can be indicated using the `SUBPARTITIONS` keyword followed by an integer value.

- Rigorous checking of the value used in `PARTITIONS` or `SUBPARTITIONS` clauses is applied and this value must adhere to the following rules:

  - The value must be a positive, nonzero integer.
  - No leading zeros are permitted.
  - The value must be an integer literal, and cannot not be an expression. For example, `PARTITIONS 0.2E+01` is not permitted, even though `0.2E+01` evaluates to `2`. (Bug #15890)

- *`partition_definition`*

  Each partition may be individually defined using a *`partition_definition`* clause. The individual parts making up this clause are as follows:

  - `PARTITION` *`partition_name`*

    Specifies a logical name for the partition.

  - `VALUES`

For range partitioning, each partition must include a `VALUES LESS THAN` clause; for list partitioning, you must specify a `VALUES IN` clause for each partition. This is used to determine which rows are to be stored in this partition. See the discussions of partitioning types in Chapter 23, *Partitioning*, for syntax examples.

- `[STORAGE] ENGINE`

  MySQL accepts a `[STORAGE] ENGINE` option for both `PARTITION` and `SUBPARTITION`. Currently, the only way in which this option can be used is to set all partitions or all subpartitions to the same storage engine, and an attempt to set different storage engines for partitions or subpartitions in the same table will give rise to the error `ERROR 1469 (HY000): The mix of handlers in the partitions is not permitted in this version of MySQL`.

- `COMMENT`

  An optional `COMMENT` clause may be used to specify a string that describes the partition. Example:

  ```
  COMMENT = 'Data for the years previous to 1999'
  ```

  The maximum length for a partition comment is 1024 characters.

- `DATA DIRECTORY` and `INDEX DIRECTORY`

  `DATA DIRECTORY` and `INDEX DIRECTORY` may be used to indicate the directory where, respectively, the data and indexes for this partition are to be stored. Both the *`data_dir`* and the *`index_dir`* must be absolute system path names.

  You must have the `FILE` privilege to use the `DATA DIRECTORY` or `INDEX DIRECTORY` partition option.

  Example:

  ```
  CREATE TABLE th (id INT, name VARCHAR(30), adate DATE)
  PARTITION BY LIST(YEAR(adate))
  (
    PARTITION p1999 VALUES IN (1995, 1999, 2003)
      DATA DIRECTORY = '/var/appdata/95/data'
      INDEX DIRECTORY = '/var/appdata/95/idx',
    PARTITION p2000 VALUES IN (1996, 2000, 2004)
      DATA DIRECTORY = '/var/appdata/96/data'
      INDEX DIRECTORY = '/var/appdata/96/idx',
    PARTITION p2001 VALUES IN (1997, 2001, 2005)
      DATA DIRECTORY = '/var/appdata/97/data'
      INDEX DIRECTORY = '/var/appdata/97/idx',
    PARTITION p2002 VALUES IN (1998, 2002, 2006)
      DATA DIRECTORY = '/var/appdata/98/data'
      INDEX DIRECTORY = '/var/appdata/98/idx'
  );
  ```

  `DATA DIRECTORY` and `INDEX DIRECTORY` behave in the same way as in the `CREATE TABLE` statement's *`table_option`* clause as used for `MyISAM` tables.

  One data directory and one index directory may be specified per partition. If left unspecified, the data and indexes are stored by default in the table's database directory.

  The `DATA DIRECTORY` and `INDEX DIRECTORY` options are ignored for creating partitioned tables if `NO_DIR_IN_CREATE` is in effect.

- `MAX_ROWS` and `MIN_ROWS`

  May be used to specify, respectively, the maximum and minimum number of rows to be stored in the partition. The values for *`max_number_of_rows`* and *`min_number_of_rows`* must be positive integers. As with the table-level options with the same names, these act only as “suggestions” to the server and are not hard limits.

- `TABLESPACE`

  May be used to designate an `InnoDB` file-per-table tablespace for the partition by specifying `TABLESPACE \`innodb_file_per_table\``. All partitions must belong to the same storage engine.

  Placing `InnoDB` table partitions in shared `InnoDB` tablespaces is not supported. Shared tablespaces include the `InnoDB` system tablespace and general tablespaces.

- *`subpartition_definition`*

  The partition definition may optionally contain one or more *`subpartition_definition`* clauses. Each of these consists at a minimum of the `SUBPARTITION` *`name`*, where *`name`* is an identifier for the subpartition. Except for the replacement of the `PARTITION` keyword with `SUBPARTITION`, the syntax for a subpartition definition is identical to that for a partition definition.

  Subpartitioning must be done by `HASH` or `KEY`, and can be done only on `RANGE` or `LIST` partitions. See Section 23.2.6, “Subpartitioning”.

Partitioning by Generated Columns

Partitioning by generated columns is permitted. For example:

```
CREATE TABLE t1 (
  s1 INT,
  s2 INT AS (EXP(s1)) STORED
)
PARTITION BY LIST (s2) (
  PARTITION p1 VALUES IN (1)
);
```

Partitioning sees a generated column as a regular column, which enables workarounds for limitations on functions that are not permitted for partitioning (see Section 23.6.3, “Partitioning Limitations Relating to Functions”). The preceding example demonstrates this technique: `EXP()` cannot be used directly in the `PARTITION BY` clause, but a generated column defined using `EXP()` is permitted.

13.1.20.1 CREATE TABLE Statement Retention

The original `CREATE TABLE` statement, including all specifications and table options are stored by MySQL when the table is created. The information is retained so that if you change storage engines, collations or other settings using an `ALTER TABLE` statement, the original table options specified are retained. This enables you to change between `InnoDB` and `MyISAM` table types even though the row formats supported by the two engines are different.

Because the text of the original statement is retained, but due to the way that certain values and options may be silently reconfigured, the active table definition (accessible through `DESCRIBE` or with `SHOW TABLE STATUS`) and the table creation string (accessible through `SHOW CREATE TABLE`) may report different values.

For InnoDB tables, SHOW CREATE TABLE and the Create_options column reported by SHOW TABLE STATUS show the actual ROW_FORMAT and KEY_BLOCK_SIZE attributes used by the table. In previous MySQL releases, the originally specified values for these attributes were reported.

13.1.20.2 Files Created by CREATE TABLE

For an InnoDB table created in a file-per-table tablespace or general tablespace, table data and associated indexes are stored in a .ibd file in the database directory. When an InnoDB table is created in the system tablespace, table data and indexes are stored in the ibdata* files that represent the system tablespace. The innodb_file_per_table option controls whether tables are created in file-per-table tablespaces or the system tablespace, by default. The TABLESPACE option can be used to place a table in a file-per-table tablespace, general tablespace, or the system tablespace, regardless of the innodb_file_per_table setting.

For MyISAM tables, the storage engine creates data and index files. Thus, for each MyISAM table *tbl_name*, there are two disk files.

File	Purpose
tbl_name.MYD	Data file
tbl_name.MYI	Index file

Chapter 16, *Alternative Storage Engines*, describes what files each storage engine creates to represent tables. If a table name contains special characters, the names for the table files contain encoded versions of those characters as described in Section 9.2.4, “Mapping of Identifiers to File Names”.

13.1.20.3 CREATE TEMPORARY TABLE Syntax

You can use the TEMPORARY keyword when creating a table. A TEMPORARY table is visible only within the current session, and is dropped automatically when the session is closed. This means that two different sessions can use the same temporary table name without conflicting with each other or with an existing non-TEMPORARY table of the same name. (The existing table is hidden until the temporary table is dropped.)

InnoDB does not support compressed temporary tables. When innodb_strict_mode is enabled (the default), CREATE TEMPORARY TABLE returns an error if ROW_FORMAT=COMPRESSED or KEY_BLOCK_SIZE is specified. If innodb_strict_mode is disabled, warnings are issued and the temporary table is created using a non-compressed row format. The innodb_file_per-table option does not affect the creation of InnoDB temporary tables.

CREATE TABLE causes an implicit commit, except when used with the TEMPORARY keyword. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

TEMPORARY tables have a very loose relationship with databases (schemas). Dropping a database does not automatically drop any TEMPORARY tables created within that database.

To create a temporary table, you must have the CREATE TEMPORARY TABLES privilege. After a session has created a temporary table, the server performs no further privilege checks on the table. The creating session can perform any operation on the table, such as DROP TABLE, INSERT, UPDATE, or SELECT.

One implication of this behavior is that a session can manipulate its temporary tables even if the current user has no privilege to create them. Suppose that the current user does not have the CREATE TEMPORARY TABLES privilege but is able to execute a definer-context stored procedure that executes with the privileges of a user who does have CREATE TEMPORARY TABLES and that creates a temporary table.

While the procedure executes, the session uses the privileges of the defining user. After the procedure returns, the effective privileges revert to those of the current user, which can still see the temporary table and perform any operation on it.

You cannot use `CREATE TEMPORARY TABLE ... LIKE` to create an empty table based on the definition of a table that resides in the `mysql` tablespace, `InnoDB` system tablespace (`innodb_system`), or a general tablespace. The tablespace definition for such a table includes a `TABLESPACE` attribute that defines the tablespace where the table resides, and the aforementioned tablespaces do not support temporary tables. To create a temporary table based on the definition of such a table, use this syntax instead:

```
CREATE TEMPORARY TABLE new_tbl SELECT * FROM orig_tbl LIMIT 0;
```

Note

Support for `TABLESPACE = innodb_file_per_table` and `TABLESPACE = innodb_temporary` clauses with `CREATE TEMPORARY TABLE` is deprecated as of MySQL 8.0.13 and will be removed in a future version of MySQL.

13.1.20.4 CREATE TABLE ... LIKE Syntax

Use `CREATE TABLE ... LIKE` to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

The copy is created using the same version of the table storage format as the original table. The `SELECT` privilege is required on the original table.

`LIKE` works only for base tables, not for views.

Important

You cannot execute `CREATE TABLE` or `CREATE TABLE ... LIKE` while a `LOCK TABLES` statement is in effect.

`CREATE TABLE ... LIKE` makes the same checks as `CREATE TABLE`. This means that if the current SQL mode is different from the mode in effect when the original table was created, the table definition might be considered invalid for the new mode and the statement will fail.

For `CREATE TABLE ... LIKE`, the destination table preserves generated column information from the original table.

For `CREATE TABLE ... LIKE`, the destination table preserves expression default values from the original table.

For `CREATE TABLE ... LIKE`, the destination table preserves `CHECK` constraints from the original table, except that all the constraint names are generated.

`CREATE TABLE ... LIKE` does not preserve any `DATA DIRECTORY` or `INDEX DIRECTORY` table options that were specified for the original table, or any foreign key definitions.

If the original table is a `TEMPORARY` table, `CREATE TABLE ... LIKE` does not preserve `TEMPORARY`. To create a `TEMPORARY` destination table, use `CREATE TEMPORARY TABLE ... LIKE`.

Tables created in the `mysql` tablespace, the `InnoDB` system tablespace (`innodb_system`), or general tablespaces include a `TABLESPACE` attribute in the table definition, which defines the tablespace where the table resides. Due to a temporary regression, `CREATE TABLE ... LIKE` preserves the `TABLESPACE` attribute and creates the table in the defined tablespace regardless of the `innodb_file_per_table` setting. To avoid the `TABLESPACE` attribute when creating an empty table based on the definition of such a table, use this syntax instead:

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl LIMIT 0;
```

13.1.20.5 CREATE TABLE ... SELECT Syntax

You can create one table from another by adding a `SELECT` statement at the end of the `CREATE TABLE` statement:

```
CREATE TABLE new_tbl [AS] SELECT * FROM orig_tbl;
```

MySQL creates new columns for all elements in the `SELECT`. For example:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
    ->        PRIMARY KEY (a), KEY(b))
    ->        ENGINE=MyISAM SELECT b,c FROM test2;
```

This creates a `MyISAM` table with three columns, `a`, `b`, and `c`. The `ENGINE` option is part of the `CREATE TABLE` statement, and should not be used following the `SELECT`; this would result in a syntax error. The same is true for other `CREATE TABLE` options such as `CHARSET`.

Notice that the columns from the `SELECT` statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
mysql> SELECT * FROM foo;
+---+
| n |
+---+
| 1 |
+---+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM bar;
+------+---+
| m    | n |
+------+---+
| NULL | 1 |
+------+---+
1 row in set (0.00 sec)
```

For each row in table `foo`, a row is inserted in `bar` with the values from `foo` and default values for the new columns.

In a table resulting from `CREATE TABLE ... SELECT`, columns named only in the `CREATE TABLE` part come first. Columns named in both parts or only in the `SELECT` part come after that. The data type of `SELECT` columns can be overridden by also specifying the column in the `CREATE TABLE` part.

If any errors occur while copying the data to the table, it is automatically dropped and not created.

You can precede the SELECT by IGNORE or REPLACE to indicate how to handle rows that duplicate unique key values. With IGNORE, rows that duplicate an existing row on a unique key value are discarded. With REPLACE, new rows replace rows that have the same unique key value. If neither IGNORE nor REPLACE is specified, duplicate unique key values result in an error. For more information, see Comparison of the IGNORE Keyword and Strict SQL Mode.

Because the ordering of the rows in the underlying SELECT statements cannot always be determined, CREATE TABLE ... IGNORE SELECT and CREATE TABLE ... REPLACE SELECT statements are flagged as unsafe for statement-based replication. Such statements produce a warning in the error log when using statement-based mode and are written to the binary log using the row-based format when using MIXED mode. See also Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

CREATE TABLE ... SELECT does not automatically create any indexes for you. This is done intentionally to make the statement as flexible as possible. If you want to have indexes in the created table, you should specify these before the SELECT statement:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

For CREATE TABLE ... SELECT, the destination table does not preserve information about whether columns in the selected-from table are generated columns. The SELECT part of the statement cannot assign values to generated columns in the destination table.

For CREATE TABLE ... SELECT, the destination table does preserve expression default values from the original table.

Some conversion of data types might occur. For example, the AUTO_INCREMENT attribute is not preserved, and VARCHAR columns can become CHAR columns. Retrained attributes are NULL (or NOT NULL) and, for those columns that have them, CHARACTER SET, COLLATION, COMMENT, and the DEFAULT clause.

When creating a table with CREATE TABLE ... SELECT, make sure to alias any function calls or expressions in the query. If you do not, the CREATE statement might fail or result in undesirable column names.

```
CREATE TABLE artists_and_works
  SELECT artist.name, COUNT(work.artist_id) AS number_of_works
  FROM artist LEFT JOIN work ON artist.id = work.artist_id
  GROUP BY artist.id;
```

You can also explicitly specify the data type for a column in the created table:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

For CREATE TABLE ... SELECT, if IF NOT EXISTS is given and the target table exists, nothing is inserted into the destination table, and the statement is not logged.

To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts during CREATE TABLE ... SELECT.

You cannot use FOR UPDATE as part of the SELECT in a statement such as CREATE TABLE *new_table* SELECT ... FROM *old_table* ... . If you attempt to do so, the statement fails.

13.1.20.6 Using FOREIGN KEY Constraints

MySQL supports foreign keys, which let you cross-reference related data across tables, and foreign key constraints, which help keep this spread-out data consistent. The essential syntax for a foreign key constraint definition in a `CREATE TABLE` or `ALTER TABLE` statement looks like this:

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (col_name, ...)
    REFERENCES tbl_name (col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

`index_name` represents a foreign key ID. The *`index_name`* value is ignored if there is already an explicitly defined index on the child table that can support the foreign key. Otherwise, MySQL implicitly creates a foreign key index that is named according to the following rules:

- If defined, the `CONSTRAINT` *`symbol`* value is used. Otherwise, the `FOREIGN KEY` *`index_name`* value is used.
- If neither a `CONSTRAINT` *`symbol`* or `FOREIGN KEY` *`index_name`* is defined, the foreign key index name is generated using the name of the referencing foreign key column.

Foreign keys definitions are subject to the following conditions:

- Foreign key relationships involve a parent table that holds the central data values, and a child table with identical values pointing back to its parent. The `FOREIGN KEY` clause is specified in the child table. The parent and child tables must use the same storage engine. They must not be `TEMPORARY` tables.

  In MySQL 8.0, creation of a foreign key constraint requires the `REFERENCES` privilege for the parent table.

- Corresponding columns in the foreign key and the referenced key must have similar data types. *The size and sign of integer types must be the same*. The length of string types need not be the same. For nonbinary (character) string columns, the character set and collation must be the same.

- When `foreign_key_checks` is enabled, which is the default setting, character set conversion is not permitted on tables that include a character string column used in a foreign key constraint. The workaround is described in Section 13.1.9, “ALTER TABLE Syntax”.

- MySQL requires indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. In the referencing table, there must be an index where the foreign key columns are listed as the *first* columns in the same order. Such an index is created on the referencing table automatically if it does not exist. This index might be silently dropped later, if you create another index that can be used to enforce the foreign key constraint. *`index_name`*, if given, is used as described previously.

- `InnoDB` permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are the *first* columns in the same order. Hidden columns that `InnoDB` adds to an index are also considered (see Section 15.6.2.1, “Clustered and Secondary Indexes”).

  `NDB` requires an explicit unique key (or primary key) on any column referenced as a foreign key.

- Index prefixes on foreign key columns are not supported. One consequence of this is that `BLOB` and `TEXT` columns cannot be included in a foreign key because indexes on those columns must always include a prefix length.

- If the `CONSTRAINT symbol` clause is given, the `symbol` value, if used, must be unique in the database. A duplicate `symbol` will result in an error similar to: `ERROR 1022 (2300): Can't write; duplicate key in table '#sql- 464_1'`. If the clause is not given, or a `symbol` is not included following the `CONSTRAINT` keyword, MySQL uses the foreign key index name up to MySQL 8.0.15, and automatically generates a constraint name thereafter.

- `InnoDB` does not currently support foreign keys for tables with user-defined partitioning. This includes both parent and child tables.

  This restriction does not apply for `NDB` tables that are partitioned by `KEY` or `LINEAR KEY` (the only user partitioning types supported by the `NDB` storage engine); these may have foreign key references or be the targets of such references.

- For `NDB` tables, `ON UPDATE CASCADE` is not supported where the reference is to the parent table's primary key.

- As of NDB 8.0.16: For `NDB` tables, `ON DELETE CASCADE` is not supported where the child table contains one or more columns of any of the `TEXT` or `BLOB` types. (Bug #89511, Bug #27484882)

Additional aspects of `FOREIGN KEY` constraint usage are described under the following topics in this section:

- Referential Actions
- Examples of Foreign Key Clauses
- Adding Foreign Keys
- Dropping Foreign Keys
- Foreign Keys and Other MySQL Statements
- Foreign Keys and the ANSI/ISO SQL Standard
- Foreign Key Metadata
- Foreign Key Errors

Referential Actions

This section describes how foreign keys help guarantee referential integrity.

For storage engines supporting foreign keys, MySQL rejects any `INSERT` or `UPDATE` operation that attempts to create a foreign key value in a child table if there is no a matching candidate key value in the parent table.

When an `UPDATE` or `DELETE` operation affects a key value in the parent table that has matching rows in the child table, the result depends on the *referential action* specified using `ON UPDATE` and `ON DELETE` subclauses of the `FOREIGN KEY` clause. MySQL supports five options regarding the action to be taken, listed here:

- `CASCADE`: Delete or update the row from the parent table, and automatically delete or update the matching rows in the child table. Both `ON DELETE CASCADE` and `ON UPDATE CASCADE` are supported. Between two tables, do not define several `ON UPDATE CASCADE` clauses that act on the same column in the parent table or in the child table.

  If a `FOREIGN KEY` clause is defined on both tables in a foreign key relationship, making both tables a parent and child, an `ON UPDATE CASCADE` or `ON DELETE CASCADE` subclause defined for one

`FOREIGN KEY` clause must be defined for the other in order for cascading operations to succeed. If an `ON UPDATE CASCADE` or `ON DELETE CASCADE` subclause is only defined for one `FOREIGN KEY` clause, cascading operations fail with an error.

Note

Cascaded foreign key actions do not activate triggers.

- `SET NULL`: Delete or update the row from the parent table, and set the foreign key column or columns in the child table to `NULL`. Both `ON DELETE SET NULL` and `ON UPDATE SET NULL` clauses are supported.

  If you specify a `SET NULL` action, *make sure that you have not declared the columns in the child table as `NOT NULL`*.

- `RESTRICT`: Rejects the delete or update operation for the parent table. Specifying `RESTRICT` (or `NO ACTION`) is the same as omitting the `ON DELETE` or `ON UPDATE` clause.

- `NO ACTION`: A keyword from standard SQL. In MySQL, equivalent to `RESTRICT`. The MySQL Server rejects the delete or update operation for the parent table if there is a related foreign key value in the referenced table. Some database systems have deferred checks, and `NO ACTION` is a deferred check. In MySQL, foreign key constraints are checked immediately, so `NO ACTION` is the same as `RESTRICT`.

- `SET DEFAULT`: This action is recognized by the MySQL parser, but both `InnoDB` and `NDB` reject table definitions containing `ON DELETE SET DEFAULT` or `ON UPDATE SET DEFAULT` clauses.

For an `ON DELETE` or `ON UPDATE` that is not specified, the default action is always `NO ACTION`.

As the default, an `ON DELETE NO ACTION` or `ON UPDATE NO ACTION` clause that is specified explicitly does not appear in `SHOW CREATE TABLE` output or in tables dumped with `mysqldump`. `RESTRICT`, which is an equivalent non-default keyword, does appear `SHOW CREATE TABLE` output and in tables dumped with `mysqldump`.

MySQL supports foreign key references between one column and another within a table. (A column cannot have a foreign key reference to itself.) In these cases, “child table records” really refers to dependent records within the same table.

A foreign key constraint on a stored generated column cannot use `CASCADE`, `SET NULL`, or `SET DEFAULT` as `ON UPDATE` referential actions, nor can it use `SET NULL` or `SET DEFAULT` as `ON DELETE` referential actions.

A foreign key constraint on the base column of a stored generated column cannot use `CASCADE`, `SET NULL`, or `SET DEFAULT` as `ON UPDATE` or `ON DELETE` referential actions.

A foreign key constraint cannot reference a virtual generated column.

For `InnoDB` restrictions related to foreign keys and generated columns, see Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”.

Examples of Foreign Key Clauses

Here is a simple example that relates `parent` and `child` tables through a single-column foreign key:

```
CREATE TABLE parent (
    id INT NOT NULL,
    PRIMARY KEY (id)
```

```
) ENGINE=INNODB;

CREATE TABLE child (
    id INT,
    parent_id INT,
    INDEX par_ind (parent_id),
    FOREIGN KEY (parent_id)
        REFERENCES parent(id)
        ON DELETE CASCADE
) ENGINE=INNODB;
```

A more complex example in which a `product_order` table has foreign keys for two other tables. One foreign key references a two-column index in the `product` table. The other references a single-column index in the `customer` table:

```
CREATE TABLE product (
    category INT NOT NULL, id INT NOT NULL,
    price DECIMAL,
    PRIMARY KEY(category, id)
)   ENGINE=INNODB;

CREATE TABLE customer (
    id INT NOT NULL,
    PRIMARY KEY (id)
)   ENGINE=INNODB;

CREATE TABLE product_order (
    no INT NOT NULL AUTO_INCREMENT,
    product_category INT NOT NULL,
    product_id INT NOT NULL,
    customer_id INT NOT NULL,

    PRIMARY KEY(no),
    INDEX (product_category, product_id),
    INDEX (customer_id),

    FOREIGN KEY (product_category, product_id)
      REFERENCES product(category, id)
      ON UPDATE CASCADE ON DELETE RESTRICT,

    FOREIGN KEY (customer_id)
      REFERENCES customer(id)
)   ENGINE=INNODB;
```

Adding Foreign Keys

You can add a new foreign key constraint to an existing table by using `ALTER TABLE`. The syntax relating to foreign keys for this statement is shown here:

```
ALTER TABLE tbl_name
    ADD [CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (col_name, ...)
    REFERENCES tbl_name (col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
```

The foreign key can be self referential (referring to the same table). When you add a foreign key constraint to a table using `ALTER TABLE`, *remember to create the required indexes first.*

Dropping Foreign Keys

You can also use `ALTER TABLE` to drop foreign keys, using the syntax shown here:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

If the FOREIGN KEY clause included a CONSTRAINT name when you created the foreign key, you can refer to that name to drop the foreign key. Otherwise, the *fk_symbol* value is generated internally when the foreign key is created. To find out the symbol value when you want to drop a foreign key, use a SHOW CREATE TABLE statement, as shown here:

```
mysql> SHOW CREATE TABLE ibtest11c\G
*************************** 1. row ***************************
       Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default '',
  `C` varchar(175) default NULL,
  PRIMARY KEY  (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`, `D`)
REFERENCES `ibtest11a` (`A`, `D`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`, `C`)
REFERENCES `ibtest11a` (`B`, `C`)
ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=INNODB CHARSET=utf8mb4
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY `0_38775`;
```

Adding and dropping a foreign key in the same ALTER TABLE statement is supported for ALTER TABLE ... ALGORITHM=INPLACE but is unsupported for ALTER TABLE ... ALGORITHM=COPY.

Foreign Keys and Other MySQL Statements

Table and column identifiers in a FOREIGN KEY ... REFERENCES ... clause can be quoted within backticks (`). Alternatively, double quotation marks (") can be used if the ANSI_QUOTES SQL mode is enabled. The setting of the lower_case_table_names system variable is also taken into account.

You can view a child table's foreign key definitions as part of the output of the SHOW CREATE TABLE statement:

```
SHOW CREATE TABLE tbl_name;
```

You can also obtain information about foreign keys by querying the INFORMATION_SCHEMA.KEY_COLUMN_USAGE table.

You can find information about foreign keys used by InnoDB tables in the INNODB_FOREIGN and INNODB_FOREIGN_COLS tables, also in the INFORMATION_SCHEMA database.

mysqldump produces correct definitions of tables in the dump file, including the foreign keys for child tables.

To make it easier to reload dump files for tables that have foreign key relationships, mysqldump automatically includes a statement in the dump output to set foreign_key_checks to 0. This avoids problems with tables having to be reloaded in a particular order when the dump is reloaded. It is also possible to set this variable manually:

```
mysql> SET foreign_key_checks = 0;
mysql> SOURCE dump_file_name;
mysql> SET foreign_key_checks = 1;
```

This enables you to import the tables in any order if the dump file contains tables that are not correctly ordered for foreign keys. It also speeds up the import operation. Setting `foreign_key_checks` to 0 can also be useful for ignoring foreign key constraints during `LOAD DATA` and `ALTER TABLE` operations. However, even if `foreign_key_checks = 0`, MySQL does not permit the creation of a foreign key constraint where a column references a nonmatching column type. Also, if a table has foreign key constraints, `ALTER TABLE` cannot be used to alter the table to use another storage engine. To change the storage engine, you must drop any foreign key constraints first.

You cannot issue `DROP TABLE` for a table that is referenced by a `FOREIGN KEY` constraint, unless you do `SET foreign_key_checks = 0`. When you drop a table, any constraints that were defined in the statement used to create that table are also dropped.

If you re-create a table that was dropped, it must have a definition that conforms to the foreign key constraints referencing it. It must have the correct column names and types, and it must have indexes on the referenced keys, as stated earlier. If these are not satisfied, MySQL returns Error 1005 and refers to Error 150 in the error message, which means that a foreign key constraint was not correctly formed. Similarly, if an `ALTER TABLE` fails due to Error 150, this means that a foreign key definition would be incorrectly formed for the altered table.

For `InnoDB` tables, you can obtain a detailed explanation of the most recent `InnoDB` foreign key error in the MySQL Server, by checking the output of `SHOW ENGINE INNODB STATUS`.

MySQL extends metadata locks, as necessary, to tables that are related by a foreign key constraint. Extending metadata locks prevents conflicting DML and DDL operations from executing concurrently on related tables. This feature also enables updates to foreign key metadata when a parent table is modified. In earlier MySQL releases, foreign key metadata, which is owned by the child table, could not be updated safely.

If a table is locked explicitly with `LOCK TABLES`, any tables related by a foreign key constraint are opened and locked implicitly. For foreign key checks, a shared read-only lock (`LOCK TABLES READ`) is taken on related tables. For cascading updates, a shared-nothing write lock (`LOCK TABLES WRITE`) is taken on related tables that are involved in the operation.

Foreign Keys and the ANSI/ISO SQL Standard

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including `InnoDB`, recognizes or enforces the `MATCH` clause used in referential-integrity constraint definitions. Use of an explicit `MATCH` clause will not have the specified effect, and also causes `ON DELETE` and `ON UPDATE` clauses to be ignored. For these reasons, specifying `MATCH` should be avoided.

The `MATCH` clause in the SQL standard controls how `NULL` values in a composite (multiple-column) foreign key are handled when comparing to a primary key. MySQL essentially implements the semantics defined by `MATCH SIMPLE`, which permit a foreign key to be all or partially `NULL`. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table. It is possible to implement other semantics using triggers.

Additionally, MySQL requires that the referenced columns be indexed for performance reasons. However, the system does not enforce a requirement that the referenced columns be `UNIQUE` or be declared `NOT NULL`. The handling of foreign key references to nonunique keys or keys that contain `NULL` values is not well defined for operations such as `UPDATE` or `DELETE CASCADE`. You are advised to use foreign keys that reference only `UNIQUE` (including `PRIMARY`) and `NOT NULL` keys.

Furthermore, MySQL parses but ignores “inline `REFERENCES` specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. MySQL accepts `REFERENCES` clauses only when specified as part of a separate `FOREIGN KEY` specification. For storage engines that do not support foreign keys (such as `MyISAM`), MySQL Server parses and ignores foreign key specifications.

Foreign Key Metadata

The `INFORMATION_SCHEMA.KEY_COLUMN_USAGE` table identifies the key columns that have constraints. Metadata specific to `InnoDB` foreign keys is found in the `INNODB_SYS_FOREIGN` and `INNODB_SYS_FOREIGN_COLS` tables.

Foreign Key Errors

In the event of a foreign key error involving `InnoDB` tables (usually Error 150 in the MySQL Server), information about the most recent `InnoDB` foreign key error can be obtained by checking `SHOW ENGINE INNODB STATUS` output.

Warning

If a user has table-level privileges for all parent tables, `ER_NO_REFERENCED_ROW_2` and `ER_ROW_IS_REFERENCED_2` error messages for foreign key operations expose information about parent tables. If a user does not have table-level privileges for all parent tables, more generic error messages are displayed instead (`ER_NO_REFERENCED_ROW` and `ER_ROW_IS_REFERENCED`).

An exception is that, for stored programs defined to execute with `DEFINER` privileges, the user against which privileges are assessed is the user in the program `DEFINER` clause, not the invoking user. If that user has table-level parent table privileges, parent table information is still displayed. In this case, it is the responsibility of the stored program creator to hide the information by including appropriate condition handlers.

13.1.20.7 CHECK Constraints

Prior to MySQL 8.0.16, `CREATE TABLE` permits only the following limited version of table `CHECK` constraint syntax, which is parsed and ignored:

```
CHECK (expr)
```

As of MySQL 8.0.16, `CREATE TABLE` permits the core features of table and column `CHECK` constraints, for all storage engines. `CREATE TABLE` permits the following `CHECK` constraint syntax, for both table constraints and column constraints:

```
[CONSTRAINT [symbol]] CHECK (expr) [[NOT] ENFORCED]
```

The optional *`symbol`* specifies a name for the constraint. If omitted, MySQL generates a name from the table name, a literal `_chk_`, and an ordinal number (1, 2, 3, ...). Constraint names have a maximum length of 64 characters. They are case sensitive, but not accent sensitive.

`expr` specifies the constraint condition as a boolean expression that must evaluate to `TRUE` or `UNKNOWN` (for `NULL` values) for each row of the table. If the condition evaluates to `FALSE`, it fails and a constraint violation occurs. The effect of a violation depends on the statement being executed, as described later in this section.

The optional enforcement clause indicates whether the constraint is enforced:

- If omitted or specified as `ENFORCED`, the constraint is created and enforced.
- If specified as `NOT ENFORCED`, the constraint is created but not enforced.

A `CHECK` constraint is specified as either a table constraint or column constraint:

- A table constraint does not appear within a column definition and can refer to any table column or columns. Forward references are permitted to columns appearing later in the table definition.
- A column constraint appears within a column definition and can refer only to that column.

Consider this table definition:

```
CREATE TABLE t1
(
  CHECK (c1 <> c2),
  c1 INT CHECK (c1 > 10),
  c2 INT CONSTRAINT c2_positive CHECK (c2 > 0),
  c3 INT CHECK (c3 < 100),
  CONSTRAINT c1_nonzero CHECK (c1 <> 0),
  CHECK (c1 > c3)
);
```

The definition includes table constraints and column constraints, in named and unnamed formats:

- The first constraint is a table constraint: It occurs outside any column definition, so it can (and does) refer to multiple table columns. This constraint contains forward references to columns not defined yet. No constraint name is specified, so MySQL generates a name.
- The next three constraints are column constraints: Each occurs within a column definition, and thus can refer only to the column being defined. One of the constraints is named explicitly. MySQL generates a name for each of the other two.
- The last two contraints are table constraints. One of them is named explicitly. MySQL generates a name for the other one.

As mentioned, MySQL generates a name for any `CHECK` constraint specified without one. To see the names generated for the preceding table definition, use `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) DEFAULT NULL,
  `c2` int(11) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  CONSTRAINT `c1_nonzero` CHECK ((`c1` <> 0)),
  CONSTRAINT `c2_positive` CHECK ((`c2` > 0)),
  CONSTRAINT `t1_chk_1` CHECK ((`c1` <> `c2`)),
  CONSTRAINT `t1_chk_2` CHECK ((`c1` > 10)),
  CONSTRAINT `t1_chk_3` CHECK ((`c3` < 100)),
  CONSTRAINT `t1_chk_4` CHECK ((`c1` > `c3`))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

The SQL standard specifies that all types of constraints (primary key, unique index, foreign key, check) belong to the same namespace. In MySQL, each constraint type has its own namespace per schema (database). Consequently, `CHECK` constraint names must be unique per schema; no two tables

in the same schema can share a CHECK constraint name. (Exception: A TEMPORARY table hides a non-TEMPORARY table of the same name, so it can have the same CHECK constraint names as well.)

Beginning generated constraint names with the table name helps ensure schema uniqueness because table names also must be unique within the schema.

CHECK condition expressions must adhere to the following rules. An error occurs if an expression contains disallowed constructs.

- Nongenerated and generated columns are permitted, except columns with the AUTO_INCREMENT attribute and columns in other tables.
- Literals, deterministic built-in functions, and operators are permitted. A function is deterministic if, given the same data in tables, multiple invocations produce the same result, independently of the connected user. Examples of functions that are nondeterministic and fail this definition: CONNECTION_ID(), CURRENT_USER(), NOW().
- Stored functions and user-defined functions are not permitted.
- Stored procedure and function parameters are not permitted.
- Variables (system variables, user-defined variables, and stored program local variables) are not permitted.
- Subqueries are not permitted.

Foreign key referential actions (ON UPDATE, ON DELETE) are prohibited on columns used in CHECK constraints. Likewise, CHECK constraints are prohibited on columns used in foreign key referential actions.

CHECK constraints are evaluated for INSERT, UPDATE, REPLACE, LOAD DATA, and LOAD XML statements and an error occurs if a constraint evaluates to FALSE. If an error occurs, handling of changes already applied differs for transactional and nontransactional storage engines, and also depends on whether strict SQL mode is in effect, as described in Strict SQL Mode.

CHECK constraints are evaluated for INSERT IGNORE, UPDATE IGNORE, LOAD DATA ... IGNORE, and LOAD XML ... IGNORE statements and a warning occurs if a constraint evaluates to FALSE. The insert or update for any offending row is skipped.

If the constraint expression evaluates to a data type that differs from the declared column type, implicit coercion to the declared type occurs according to the usual MySQL type-conversion rules. See Section 12.2, “Type Conversion in Expression Evaluation”. If type conversion fails or results in a loss of precision, an error occurs.

Note

Constraint expression evaluation uses the SQL mode in effect at evaluation time. If any component of the expression depends on the SQL mode, different results may occur for different uses of the table unless the SQL mode is the same during all uses.

13.1.20.8 Silent Column Specification Changes

In some cases, MySQL silently changes column specifications from those given in a CREATE TABLE or ALTER TABLE statement. These might be changes to a data type, to attributes associated with a data type, or to an index specification.

All changes are subject to the internal row-size limit of 65,535 bytes, which may cause some attempts at data type changes to fail. See Section 8.4.7, “Limits on Table Column Count and Row Size”.

- Columns that are part of a `PRIMARY KEY` are made `NOT NULL` even if not declared that way.
- Trailing spaces are automatically deleted from `ENUM` and `SET` member values when the table is created.
- MySQL maps certain data types used by other SQL database vendors to MySQL types. See Section 11.10, “Using Data Types from Other Database Engines”.
- If you include a `USING` clause to specify an index type that is not permitted for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.
- If strict SQL mode is not enabled, a `VARCHAR` column with a length specification greater than 65535 is converted to `TEXT`, and a `VARBINARY` column with a length specification greater than 65535 is converted to `BLOB`. Otherwise, an error occurs in either of these cases.
- Specifying the `CHARACTER SET binary` attribute for a character data type causes the column to be created as the corresponding binary data type: `CHAR` becomes `BINARY`, `VARCHAR` becomes `VARBINARY`, and `TEXT` becomes `BLOB`. For the `ENUM` and `SET` data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET binary,
  c2 TEXT CHARACTER SET binary,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

The resulting table has this definition:

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

To see whether MySQL used a data type other than the one you specified, issue a `DESCRIBE` or `SHOW CREATE TABLE` statement after creating or altering the table.

Certain other data type changes can occur if you compress a table using `myisampack`. See Section 16.2.3.3, “Compressed Table Characteristics”.

13.1.20.9 CREATE TABLE and Generated Columns

`CREATE TABLE` supports the specification of generated columns. Values of a generated column are computed from an expression included in the column definition.

Generated columns are also supported by the `NDB` storage engine.

The following simple example shows a table that stores the lengths of the sides of right triangles in the `sidea` and `sideb` columns, and computes the length of the hypotenuse in `sidec` (the square root of the sums of the squares of the other sides):

```
```

```
CREATE TABLE triangle (
  sidea DOUBLE,
  sideb DOUBLE,
  sidec DOUBLE AS (SQRT(sidea * sidea + sideb * sideb))
);
INSERT INTO triangle (sidea, sideb) VALUES(1,1),(3,4),(6,8);
```

Selecting from the table yields this result:

```
mysql> SELECT * FROM triangle;
+-------+-------+--------------------+
| sidea | sideb | sidec              |
+-------+-------+--------------------+
|     1 |     1 | 1.4142135623730951 |
|     3 |     4 |                  5 |
|     6 |     8 |                 10 |
+-------+-------+--------------------+
```

Any application that uses the `triangle` table has access to the hypotenuse values without having to specify the expression that calculates them.

Generated column definitions have this syntax:

```
col_name data_type [GENERATED ALWAYS] AS (expr)
  [VIRTUAL | STORED] [NOT NULL | NULL]
  [UNIQUE [KEY]] [[PRIMARY] KEY]
  [COMMENT 'string']
```

`AS (expr)` indicates that the column is generated and defines the expression used to compute column values. `AS` may be preceded by `GENERATED ALWAYS` to make the generated nature of the column more explicit. Constructs that are permitted or prohibited in the expression are discussed later.

The `VIRTUAL` or `STORED` keyword indicates how column values are stored, which has implications for column use:

- `VIRTUAL`: Column values are not stored, but are evaluated when rows are read, immediately after any `BEFORE` triggers. A virtual column takes no storage.

  `InnoDB` supports secondary indexes on virtual columns. See Section 13.1.20.10, “Secondary Indexes and Generated Columns”.

- `STORED`: Column values are evaluated and stored when rows are inserted or updated. A stored column does require storage space and can be indexed.

The default is `VIRTUAL` if neither keyword is specified.

It is permitted to mix `VIRTUAL` and `STORED` columns within a table.

Other attributes may be given to indicate whether the column is indexed or can be `NULL`, or provide a comment.

Generated column expressions must adhere to the following rules. An error occurs if an expression contains disallowed constructs.

- Literals, deterministic built-in functions, and operators are permitted. A function is deterministic if, given the same data in tables, multiple invocations produce the same result, independently of the connected user. Examples of functions that are nondeterministic and fail this definition: `CONNECTION_ID()`, `CURRENT_USER()`, `NOW()`.

- Stored functions and user-defined functions are not permitted.
- Stored procedure and function parameters are not permitted.
- Variables (system variables, user-defined variables, and stored program local variables) are not permitted.
- Subqueries are not permitted.
- A generated column definition can refer to other generated columns, but only those occurring earlier in the table definition. A generated column definition can refer to any base (nongenerated) column in the table whether its definition occurs earlier or later.
- The `AUTO_INCREMENT` attribute cannot be used in a generated column definition.
- An `AUTO_INCREMENT` column cannot be used as a base column in a generated column definition.
- If expression evaluation causes truncation or provides incorrect input to a function, the `CREATE TABLE` statement terminates with an error and the DDL operation is rejected.

If the expression evaluates to a data type that differs from the declared column type, implicit coercion to the declared type occurs according to the usual MySQL type-conversion rules. See Section 12.2, “Type Conversion in Expression Evaluation”.

Note

Expression evaluation uses the SQL mode in effect at evaluation time. If any component of the expression depends on the SQL mode, different results may occur for different uses of the table unless the SQL mode is the same during all uses.

For `CREATE TABLE ... LIKE`, the destination table preserves generated column information from the original table.

For `CREATE TABLE ... SELECT`, the destination table does not preserve information about whether columns in the selected-from table are generated columns. The `SELECT` part of the statement cannot assign values to generated columns in the destination table.

Partitioning by generated columns is permitted. See Table Partitioning.

A foreign key constraint on a stored generated column cannot use `CASCADE`, `SET NULL`, or `SET DEFAULT` as `ON UPDATE` referential actions, nor can it use `SET NULL` or `SET DEFAULT` as `ON DELETE` referential actions.

A foreign key constraint on the base column of a stored generated column cannot use `CASCADE`, `SET NULL`, or `SET DEFAULT` as `ON UPDATE` or `ON DELETE` referential actions.

A foreign key constraint cannot reference a virtual generated column.

For `InnoDB` restrictions related to foreign keys and generated columns, see Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”.

Triggers cannot use `NEW.col_name` or use `OLD.col_name` to refer to generated columns.

For `INSERT`, `REPLACE`, and `UPDATE`, if a generated column is inserted into, replaced, or updated explicitly, the only permitted value is `DEFAULT`.

A generated column in a view is considered updatable because it is possible to assign to it. However, if such a column is updated explicitly, the only permitted value is `DEFAULT`.

Generated columns have several use cases, such as these:

- Virtual generated columns can be used as a way to simplify and unify queries. A complicated condition can be defined as a generated column and referred to from multiple queries on the table to ensure that all of them use exactly the same condition.
- Stored generated columns can be used as a materialized cache for complicated conditions that are costly to calculate on the fly.
- Generated columns can simulate functional indexes: Use a generated column to define a functional expression and index it. This can be useful for working with columns of types that cannot be indexed directly, such as `JSON` columns; see Indexing a Generated Column to Provide a JSON Column Index, for a detailed example.

  For stored generated columns, the disadvantage of this approach is that values are stored twice; once as the value of the generated column and once in the index.
- If a generated column is indexed, the optimizer recognizes query expressions that match the column definition and uses indexes from the column as appropriate during query execution, even if a query does not refer to the column directly by name. For details, see Section 8.3.11, “Optimizer Use of Generated Column Indexes”.

Example:

Suppose that a table `t1` contains `first_name` and `last_name` columns and that applications frequently construct the full name using an expression like this:

```
SELECT CONCAT(first_name,' ',last_name) AS full_name FROM t1;
```

One way to avoid writing out the expression is to create a view `v1` on `t1`, which simplifies applications by enabling them to select `full_name` directly without using an expression:

```
CREATE VIEW v1 AS
SELECT *, CONCAT(first_name,' ',last_name) AS full_name FROM t1;

SELECT full_name FROM v1;
```

A generated column also enables applications to select `full_name` directly without the need to define a view:

```
CREATE TABLE t1 (
  first_name VARCHAR(10),
  last_name VARCHAR(10),
  full_name VARCHAR(255) AS (CONCAT(first_name,' ',last_name))
);

SELECT full_name FROM t1;
```

13.1.20.10 Secondary Indexes and Generated Columns

`InnoDB` supports secondary indexes on virtual generated columns. Other index types are not supported. A secondary index defined on a virtual column is sometimes referred to as a “virtual index”.

A secondary index may be created on one or more virtual columns or on a combination of virtual columns and regular columns or stored generated columns. Secondary indexes that include virtual columns may be defined as `UNIQUE`.

When a secondary index is created on a virtual generated column, generated column values are materialized in the records of the index. If the index is a covering index (one that includes all the columns retrieved by a query), generated column values are retrieved from materialized values in the index structure instead of computed “on the fly”.

There are additional write costs to consider when using a secondary index on a virtual column due to computation performed when materializing virtual column values in secondary index records during `INSERT` and `UPDATE` operations. Even with additional write costs, secondary indexes on virtual columns may be preferable to generated *stored* columns, which are materialized in the clustered index, resulting in larger tables that require more disk space and memory. If a secondary index is not defined on a virtual column, there are additional costs for reads, as virtual column values must be computed each time the column's row is examined.

Values of an indexed virtual column are MVCC-logged to avoid unnecessary recomputation of generated column values during rollback or during a purge operation. The data length of logged values is limited by the index key limit of 767 bytes for `COMPACT` and `REDUNDANT` row formats, and 3072 bytes for `DYNAMIC` and `COMPRESSED` row formats.

Adding or dropping a secondary index on a virtual column is an in-place operation.

Indexing a Generated Column to Provide a JSON Column Index

As noted elsewhere, `JSON` columns cannot be indexed directly. To create an index that references such a column indirectly, you can define a generated column that extracts the information that should be indexed, then create an index on the generated column, as shown in this example:

```
mysql> CREATE TABLE jemp (
    ->     c JSON,
    ->     g INT GENERATED ALWAYS AS (c->"$.id")),
    ->     INDEX i (g)
    -> );
Query OK, 0 rows affected (0.28 sec)

mysql> INSERT INTO jemp (c) VALUES
     >   ('{"id": "1", "name": "Fred"}'), ('{"id": "2", "name": "Wilma"}'),
     >   ('{"id": "3", "name": "Barney"}'), ('{"id": "4", "name": "Betty"}');
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT c->>"$.name" AS name
     >     FROM jemp WHERE g > 2;
+--------+
| name   |
+--------+
| Barney |
| Betty  |
+--------+
2 rows in set (0.00 sec)

mysql> EXPLAIN SELECT c->>"$.name" AS name
     >    FROM jemp WHERE g > 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: jemp
   partitions: NULL
         type: range
```

```
possible_keys: i
          key: i
      key_len: 5
          ref: NULL
         rows: 2
     filtered: 100.00
        Extra: Using where
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select json_unquote(json_extract(`test`.`jemp`.`c`,'$.name'))
AS `name` from `test`.`jemp` where (`test`.`jemp`.`g` > 2)
1 row in set (0.00 sec)
```

(We have wrapped the output from the last statement in this example to fit the viewing area.)

When you use `EXPLAIN` on a `SELECT` or other SQL statement containing one or more expressions that use the `->` or `->>` operator, these expressions are translated into their equivalents using `JSON_EXTRACT()` and (if needed) `JSON_UNQUOTE()` instead, as shown here in the output from `SHOW WARNINGS` immediately following this `EXPLAIN` statement:

```
mysql> EXPLAIN SELECT c->>"$.name"
     > FROM jemp WHERE g > 2 ORDER BY c->"$.name"\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: jemp
   partitions: NULL
         type: range
possible_keys: i
          key: i
      key_len: 5
          ref: NULL
         rows: 2
     filtered: 100.00
        Extra: Using where; Using filesort
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select json_unquote(json_extract(`test`.`jemp`.`c`,'$.name')) AS
`c->>"$.name"` from `test`.`jemp` where (`test`.`jemp`.`g` > 2) order by
json_extract(`test`.`jemp`.`c`,'$.name')
1 row in set (0.00 sec)
```

See the descriptions of the `->` and `->>` operators, as well as those of the `JSON_EXTRACT()` and `JSON_UNQUOTE()` functions, for additional information and examples.

This technique also can be used to provide indexes that indirectly reference columns of other types that cannot be indexed directly, such as `GEOMETRY` columns.

JSON columns and indirect indexing in NDB Cluster

It is also possible to use indirect indexing of JSON columns in MySQL NDB Cluster, subject to the following conditions:

1. `NDB` handles a `JSON` column value internally as a `BLOB`. This means that any `NDB` table having one or more JSON columns must have a primary key, else it cannot be recorded in the binary log.

2. The NDB storage engine does not support indexing of virtual columns. Since the default for generated columns is VIRTUAL, you must specify explicitly the generated column to which to apply the indirect index as STORED.

The **CREATE TABLE** statement used to create the table jempn shown here is a version of the jemp table shown previously, with modifications making it compatible with NDB:

```
CREATE TABLE jempn (
  a BIGINT(20) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  c JSON DEFAULT NULL,
  g INT GENERATED ALWAYS AS (c->"$.name") STORED,
  INDEX i (g)
) ENGINE=NDB;
```

We can populate this table using the following INSERT statement:

```
INSERT INTO jempn (a, c) VALUES
  (NULL, '{"id": "1", "name": "Fred"}'),
  (NULL, '{"id": "2", "name": "Wilma"}'),
  (NULL, '{"id": "3", "name": "Barney"}'),
  (NULL, '{"id": "4", "name": "Betty"}');
```

Now NDB can use index i, as shown here:

```
mysql> EXPLAIN SELECT c->>"$.name" AS name
          FROM jempn WHERE g > 2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: jempn
   partitions: p0,p1
         type: range
possible_keys: i
          key: i
      key_len: 5
          ref: NULL
         rows: 3
     filtered: 100.00
        Extra: Using where with pushed condition (`test`.`jempn`.`g` > 2)
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select
json_unquote(json_extract(`test`.`jempn`.`c`,'$.name')) AS `name` from
`test`.`jempn` where (`test`.`jempn`.`g` > 2)
1 row in set (0.00 sec)
```

You should keep in mind that a stored generated column uses DataMemory, and that an index on such a column uses IndexMemory.

13.1.20.11 Setting NDB_TABLE Options

In MySQL NDB Cluster, the table comment in a CREATE TABLE or ALTER TABLE statement can also be used to specify an NDB_TABLE option, which consists of one or more name-value pairs, separated by commas if need be, following the string NDB_TABLE=. Complete syntax for names and values syntax is shown here:

```
COMMENT="NDB_TABLE=ndb_table_option[,ndb_table_option[,...]]"

ndb_table_option:
    NOLOGGING={1|0}
  | READ_BACKUP={1|0}
  | PARTITION_BALANCE={FOR_RP_BY_NODE|FOR_RA_BY_NODE|FOR_RP_BY_LDM
                       |FOR_RA_BY_LDM|FOR_RA_BY_LDM_X_2
                       |FOR_RA_BY_LDM_X_3|FOR_RA_BY_LDM_X_4}
  | FULLY_REPLICATED={1|0}
```

Spaces are not permitted within the quoted string. The string is case-insensitive.

The four NDB table options that can be set as part of a comment in this way are described in more detail in the next few paragraphs.

NOLOGGING: Using 1 corresponds to having ndb_table_no_logging enabled, but has no actual effect. Provided as a placeholder, mostly for completeness of ALTER TABLE statements.

READ_BACKUP: Setting this option to 1 has the same effect as though ndb_read_backup were enabled; enables reading from any replica. You can set READ_BACKUP for an existing table online, using an ALTER TABLE statement similar to one of those shown here:

```
ALTER TABLE ... ALGORITHM=INPLACE, COMMENT="NDB_TABLE=READ_BACKUP=1";

ALTER TABLE ... ALGORITHM=INPLACE, COMMENT="NDB_TABLE=READ_BACKUP=0";
```

For more information about the ALGORITHM option for ALTER TABLE, see Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”.

PARTITION_BALANCE: Provides additional control over assignment and placement of partitions. The following four schemes are supported:

1. FOR_RP_BY_NODE: One partition per node.

   Only one LDM on each node stores a primary partition. Each partition is stored in the same LDM (same ID) on all nodes.

2. FOR_RA_BY_NODE: One partition per node group.

   Each node stores a single partition, which can be either a primary replica or a backup replica. Each partition is stored in the same LDM on all nodes.

3. FOR_RP_BY_LDM: One partition for each LDM on each node; the default.

   This is the same behavior as prior to MySQL NDB Cluster 7.5.2, except for a slightly different mapping of partitions to LDMs, starting with LDM 0 and placing one partition per node group, then moving on to the next LDM.

   This is the setting used if READ_BACKUP is set to 1.

4. FOR_RA_BY_LDM: One partition per LDM in each node group.

   These partitions can be primary or backup partitions.

5. FOR_RA_BY_LDM_X_2: Two partitions per LDM in each node group.

   These partitions can be primary or backup partitions.

6. FOR_RA_BY_LDM_X_3: Three partitions per LDM in each node group.

These partitions can be primary or backup partitions.

7. FOR_RA_BY_LDM_X_4: Four partitions per LDM in each node group.

   These partitions can be primary or backup partitions.

PARTITION_BALANCE is the preferred interface for setting the number of partitions per table. Using MAX_ROWS to force the number of partitions is deprecated but continues to be supported for backward compatibility; it is subject to removal in a future release of MySQL NDB Cluster. (Bug #81759, Bug #23544301)

FULLY_REPLICATED controls whether the table is fully replicated, that is, whether each data node has a complete copy of the table. To enable full replication of the table, use FULLY_REPLICATED=1.

This setting can also be controlled using the ndb_fully_replicated system variable. Setting it to ON enables the option by default for all new NDB tables; the default is OFF. The ndb_data_node_neighbour system variable is also used for fully replicated tables, to ensure that when a fully replicated table is accessed, we access the data node which is local to this MySQL Server.

An example of a CREATE TABLE statement using such a comment when creating an NDB table is shown here:

```
mysql> CREATE TABLE t1 (
    >     c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    >     c2 VARCHAR(100),
    >     c3 VARCHAR(100) )
    > ENGINE=NDB
    >
COMMENT="NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RP_BY_NODE";
```

The comment is displayed as part of the ouput of SHOW CREATE TABLE. The text of the comment is also available from querying the MySQL Information Schema TABLES table, as in this example:

```
mysql> SELECT TABLE_NAME, TABLE_SCHEMA, TABLE_COMMENT
    > FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME="t1";
+------------+--------------+------------------------------------------------------------+
| TABLE_NAME | TABLE_SCHEMA | TABLE_COMMENT                                              |
+------------+--------------+------------------------------------------------------------+
| t1         | c            | NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RP_BY_NODE |
| t1         | d            |                                                            |
+------------+--------------+------------------------------------------------------------+
2 rows in set (0.00 sec)
```

This comment syntax is also supported with ALTER TABLE statements for NDB tables. Keep in mind that a table comment used with ALTER TABLE replaces any existing comment which the table might have.

```
mysql> ALTER TABLE t1 COMMENT="NDB_TABLE=PARTITION_BALANCE=FOR_RA_BY_NODE";
Query OK, 0 rows affected (0.40 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT TABLE_NAME, TABLE_SCHEMA, TABLE_COMMENT
    > FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME="t1";
+------------+--------------+-------------------------------------------------+
| TABLE_NAME | TABLE_SCHEMA | TABLE_COMMENT                                   |
+------------+--------------+-------------------------------------------------+
| t1         | c            | NDB_TABLE=PARTITION_BALANCE=FOR_RA_BY_NODE      |
| t1         | d            |                                                 |
+------------+--------------+-------------------------------------------------+
2 rows in set (0.01 sec)
```

You can also see the value of the PARTITION_BALANCE option in the output of ndb_desc. ndb_desc also shows whether the READ_BACKUP and FULLY_REPLICATED options are set for the table. See the description of this program for more information.

Because the READ_BACKUP value was not carried over to the new comment set by the ALTER TABLE statement, there is no longer a way using SQL to retrieve the value previously set for it. To keep this from happening, it is suggested that you preserve any such values from the existing comment string, like this:

```
mysql> SELECT TABLE_NAME, TABLE_SCHEMA, TABLE_COMMENT
     > FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME="t1";
+------------+--------------+------------------------------------------------------------+
| TABLE_NAME | TABLE_SCHEMA | TABLE_COMMENT                                              |
+------------+--------------+------------------------------------------------------------+
| t1         | c            | NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RP_BY_NODE |
| t1         | d            |                                                            |
+------------+--------------+------------------------------------------------------------+
2 rows in set (0.00 sec)

mysql> ALTER TABLE t1 COMMENT="NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RA_BY_NODE";
Query OK, 0 rows affected (1.56 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SELECT TABLE_NAME, TABLE_SCHEMA, TABLE_COMMENT
     > FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME="t1";
+------------+--------------+------------------------------------------------------------------+
| TABLE_NAME | TABLE_SCHEMA | TABLE_COMMENT                                                    |
+------------+--------------+------------------------------------------------------------------+
| t1         | c            | NDB_TABLE=READ_BACKUP=0,PARTITION_BALANCE=FOR_RA_BY_NODE       |
| t1         | d            |                                                                  |
+------------+--------------+------------------------------------------------------------------+
2 rows in set (0.01 sec)
```

13.1.21 CREATE TABLESPACE Syntax

```
CREATE [UNDO] TABLESPACE tablespace_name

  InnoDB and NDB:
    [ADD DATAFILE 'file_name']

  InnoDB only:
    [FILE_BLOCK_SIZE = value]
    [ENCRYPTION [=] {'Y' | 'N'}]

  NDB only:
    USE LOGFILE GROUP logfile_group
    [EXTENT_SIZE [=] extent_size]
    [INITIAL_SIZE [=] initial_size]
    [AUTOEXTEND_SIZE [=] autoextend_size]
    [MAX_SIZE [=] max_size]
    [NODEGROUP [=] nodegroup_id]
    [WAIT]
    [COMMENT [=] 'string']

  InnoDB and NDB:
    [ENGINE [=] engine_name]
```

This statement is used to create a tablespace. The precise syntax and semantics depend on the storage engine used. In standard MySQL releases, this is always an InnoDB tablespace. MySQL NDB Cluster also supports tablespaces using the NDB storage engine.

- Considerations for InnoDB
- Considerations for NDB Cluster

- Options
- Notes
- InnoDB Examples
- NDB Example

Considerations for InnoDB

`CREATE TABLESPACE` syntax is used to create general tablespaces or undo tablespaces. The `UNDO` keyword, introduced in MySQL 8.0.14, must be specified to create an undo tablespace.

A general tablespace is a shared tablespace. It can hold multiple tables, and supports all table row formats. General tablespaces can be created in a location relative to or independent of the data directory.

After creating an `InnoDB` general tablespace, use `CREATE TABLE` *`tbl_name`* `... TABLESPACE [=]` *`tablespace_name`* or `ALTER TABLE` *`tbl_name`* `TABLESPACE [=]` *`tablespace_name`* to add tables to the tablespace. For more information, see Section 15.6.3.3, “General Tablespaces”.

Undo tablespaces contain undo logs. Undo tablespaces can be created in a chosen location by specifying a fully qualified data file path. For more information, see Section 15.6.3.4, “Undo Tablespaces”.

Considerations for NDB Cluster

This statement is used to create a tablespace, which can contain one or more data files, providing storage space for NDB Cluster Disk Data tables (see Section 22.5.13, “NDB Cluster Disk Data Tables”). One data file is created and added to the tablespace using this statement. Additional data files may be added to the tablespace by using the `ALTER TABLESPACE` statement (see Section 13.1.10, “ALTER TABLESPACE Syntax”).

Note

All NDB Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a log file group with the same name, or a tablespace and a data file with the same name.

A log file group of one or more `UNDO` log files must be assigned to the tablespace to be created with the `USE LOGFILE GROUP` clause. *`logfile_group`* must be an existing log file group created with `CREATE LOGFILE GROUP` (see Section 13.1.16, “CREATE LOGFILE GROUP Syntax”). Multiple tablespaces may use the same log file group for `UNDO` logging.

When setting `EXTENT_SIZE` or `INITIAL_SIZE`, you may optionally follow the number with a one-letter abbreviation for an order of magnitude, similar to those used in `my.cnf`. Generally, this is one of the letters `M` (for megabytes) or `G` (for gigabytes).

`INITIAL_SIZE` and `EXTENT_SIZE` are subject to rounding as follows:

- `EXTENT_SIZE` is rounded up to the nearest whole multiple of 32K.
- `INITIAL_SIZE` is rounded *down* to the nearest whole multiple of 32K; this result is rounded up to the nearest whole multiple of `EXTENT_SIZE` (after any rounding).

The rounding just described is done explicitly, and a warning is issued by the MySQL Server when any such rounding is performed. The rounded values are also used by the NDB kernel for calculating `INFORMATION_SCHEMA.FILES` column values and other purposes. However, to avoid an unexpected result, we suggest that you always use whole multiples of 32K in specifying these options.

When `CREATE TABLESPACE` is used with `ENGINE [=] NDB`, a tablespace and associated data file are created on each Cluster data node. You can verify that the data files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. (See the example later in this section.)

(See Section 25.11, "The INFORMATION_SCHEMA FILES Table".)

Options

- `ADD DATAFILE`: Defines the name of a tablespace data file. The `ADD DATAFILE` clause is required when creating undo tablespaces. Otherwise, it is optional as of MySQL 8.0.14.

  An `InnoDB` tablespace supports only a single data file, whose name must include a `.ibd` extension. An NDB Cluster tablespace supports multiple data files which can have any legal file names; more data files can be added to an NDB Cluster tablespace following its creation by using an `ALTER TABLESPACE` statement.

  To place a general tablespace data file in a location outside of the data directory, include a fully qualified path or a path relative to the data directory. Only a fully qualified path is permitted for undo tablespaces. If you do not specify a path, a general tablespace is created in the data directory. An undo tablespace created without specifying a path is created in the directory defined by the `innodb_undo_directory` variable. If the `innodb_undo_directory` variable is undefined, undo tablespaces are created in the data directory.

  Creating a general tablespace in a subdirectory under the data directory is not supported to avoid conflicts with implicitly created file-per-table tablespaces. When creating a general tablespace or undo tablespace outside of the data directory, the directory must exist and must be known to `InnoDB` prior to creating the tablespace. To make a directory known to `InnoDB`, add it to the `innodb_directories` value or to one of the variables whose values are appended to the `innodb_directories` value. `innodb_directories` is a read-only variable. Configuring it requires restarting the server.

  The *`file_name`*, including any specified path, must be quoted with single or double quotations marks. File names (not counting the file extension) and directory names must be at least one byte in length. Zero length file names and directory names are not supported.

  If the `ADD DATAFILE` clause is not specified when creating a tablespace, a tablespace data file with a unique file name is created implicitly. The unique file name is a 128 bit UUID formatted into five groups of hexadecimal numbers separated by dashes (*`aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee`*). A file extension is added if required by the storage engine. An `.ibd` file extension is added for `InnoDB` general tablespace data files. In a replication environment, the data file name created on the master is not the same as the data file name created on the slave.

  As of MySQL 8.0.17, the `ADD DATAFILE` clause does not permit circular directory references when creating an `InnoDB` tablespace. For example, the circular directory reference (`/../`) in the following statement is not permitted:

  ```
  CREATE TABLESPACE ts1 ADD DATAFILE ts1.ibd 'any_directory/../ts1.ibd';
  ```

  An exception to this restriction exists on Linux, where a circular directory reference is permitted if the preceding directory is a symbolic link. For example, the data file path in the example above is permitted if *`any_directory`* is a symbolic link. (It is still permitted for data file paths to begin with '`../`'.)

- `FILE_BLOCK_SIZE`: This option—which is specific to `InnoDB` general tablespaces, and is ignored by `NDB`—defines the block size for the tablespace data file. Values can be specified in bytes or kilobytes. For example, an 8 kilobyte file block size can be specified as 8192 or 8K. If you do not specify this option, `FILE_BLOCK_SIZE` defaults to the `innodb_page_size` value.

`FILE_BLOCK_SIZE` is required when you intend to use the tablespace for storing compressed `InnoDB` tables (`ROW_FORMAT=COMPRESSED`). In this case, you must define the tablespace `FILE_BLOCK_SIZE` when creating the tablespace.

If `FILE_BLOCK_SIZE` is equal the `innodb_page_size` value, the tablespace can contain only tables having an uncompressed row format (`COMPACT`, `REDUNDANT`, and `DYNAMIC`). Tables with a `COMPRESSED` row format have a different physical page size than uncompressed tables. Therefore, compressed tables cannot coexist in the same tablespace as uncompressed tables.

For a general tablespace to contain compressed tables, `FILE_BLOCK_SIZE` must be specified, and the `FILE_BLOCK_SIZE` value must be a valid compressed page size in relation to the `innodb_page_size` value. Also, the physical page size of the compressed table (`KEY_BLOCK_SIZE`) must be equal to `FILE_BLOCK_SIZE/1024`. For example, if `innodb_page_size=16K`, and `FILE_BLOCK_SIZE=8K`, the `KEY_BLOCK_SIZE` of the table must be 8. For more information, see Section 15.6.3.3, “General Tablespaces”.

- `USE LOGFILE GROUP`: Required for `NDB`, this is the name of a log file group previously created using `CREATE LOGFILE GROUP`. Not supported for `InnoDB`, where it fails with an error.

- `EXTENT_SIZE`: This option is specific to NDB, and is not supported by InnoDB, where it fails with an error. `EXTENT_SIZE` sets the size, in bytes, of the extents used by any files belonging to the tablespace. The default value is 1M. The minimum size is 32K, and theoretical maximum is 2G, although the practical maximum size depends on a number of factors. In most cases, changing the extent size does not have any measurable effect on performance, and the default value is recommended for all but the most unusual situations.

  An *extent* is a unit of disk space allocation. One extent is filled with as much data as that extent can contain before another extent is used. In theory, up to 65,535 (64K) extents may used per data file; however, the recommended maximum is 32,768 (32K). The recommended maximum size for a single data file is 32G—that is, 32K extents × 1 MB per extent. In addition, once an extent is allocated to a given partition, it cannot be used to store data from a different partition; an extent cannot store data from more than one partition. This means, for example that a tablespace having a single datafile whose `INITIAL_SIZE` (described in the following item) is 256 MB and whose `EXTENT_SIZE` is 128M has just two extents, and so can be used to store data from at most two different disk data table partitions.

  You can see how many extents remain free in a given data file by querying the `INFORMATION_SCHEMA.FILES` table, and so derive an estimate for how much space remains free in the file. For further discussion and examples, see Section 25.11, “The INFORMATION_SCHEMA FILES Table”.

- `INITIAL_SIZE`: This option is specific to `NDB`, and is not supported by `InnoDB`, where it fails with an error.

  The `INITIAL_SIZE` parameter sets the total size in bytes of the data file that was specific using `ADD DATAFILE`. Once this file has been created, its size cannot be changed; however, you can add more data files to the tablespace using `ALTER TABLESPACE ... ADD DATAFILE`.

  `INITIAL_SIZE` is optional; its default value is 134217728 (128 MB).

  On 32-bit systems, the maximum supported value for `INITIAL_SIZE` is 4294967296 (4 GB).

- `AUTOEXTEND_SIZE`: Currently ignored by MySQL; reserved for possible future use. Has no effect in any release of MySQL 8.0 or MySQL NDB Cluster 8.0, regardless of the storage engine used.

- `MAX_SIZE`: Currently ignored by MySQL; reserved for possible future use. Has no effect in any release of MySQL 8.0 or MySQL NDB Cluster 8.0, regardless of the storage engine used.

- `NODEGROUP`: Currently ignored by MySQL; reserved for possible future use. Has no effect in any release of MySQL 8.0 or MySQL NDB Cluster 8.0, regardless of the storage engine used.
- `WAIT`: Currently ignored by MySQL; reserved for possible future use. Has no effect in any release of MySQL 8.0 or MySQL NDB Cluster 8.0, regardless of the storage engine used.
- `COMMENT`: Currently ignored by MySQL; reserved for possible future use. Has no effect in any release of MySQL 8.0 or MySQL NDB Cluster 8.0, regardless of the storage engine used.
- The `ENCRYPTION` clause enables or disables page-level data encryption for an `InnoDB` general tablespace. Encryption support for general tablespaces was introduced in MySQL 8.0.13.

  As of MySQL 8.0.16, if the `ENCRYPTION` clause is not specified, the `default_table_encryption` setting controls whether encryption is enabled. The `ENCRYPTION` clause overrides the `default_table_encryption` setting. However, if the `table_encryption_privilege_check` variable is enabled, the `TABLE_ENCRYPTION_ADMIN` privilege is required to use an `ENCRYPTION` clause setting that differs from the `default_table_encryption` setting.

  A keyring plugin must be installed and configured before an encryption-enabled tablespace can be created.

  When a general tablespace is encrypted, all tables residing in the tablespace are encrypted. Likewise, a table created in an encrypted tablespace is encrypted.

  For more information, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
- `ENGINE`: Defines the storage engine which uses the tablespace, where *`engine_name`* is the name of the storage engine. Currently, only the `InnoDB` storage engine is supported by standard MySQL 8.0 releases. MySQL NDB Cluster supports both `NDB` and `InnoDB` tablespaces. The value of the `default_storage_engine` system variable is used for `ENGINE` if the option is not specified.

Notes

- For the rules covering the naming of MySQL tablespaces, see Section 9.2, “Schema Object Names”. In addition to these rules, the slash character (“/”) is not permitted, nor can you use names beginning with `innodb_`, as this prefix is reserved for system use.
- Creation of temporary general tablespaces is not supported.
- General tablespaces do not support temporary tables.
- The `TABLESPACE` option may be used with `CREATE TABLE` or `ALTER TABLE` to assign an `InnoDB` table partition or subpartition to a file-per-table tablespace. All partitions must belong to the same storage engine. Assigning table partitions to shared `InnoDB` tablespaces is not supported. Shared tablespaces include the `InnoDB` system tablespace and general tablespaces.
- General tablespaces support the addition of tables of any row format using `CREATE TABLE ... TABLESPACE`. `innodb_file_per_table` does not need to be enabled.
- `innodb_strict_mode` is not applicable to general tablespaces. Tablespace management rules are strictly enforced independently of `innodb_strict_mode`. If `CREATE TABLESPACE` parameters are incorrect or incompatible, the operation fails regardless of the `innodb_strict_mode` setting. When a table is added to a general tablespace using `CREATE TABLE ... TABLESPACE` or `ALTER TABLE ... TABLESPACE`, `innodb_strict_mode` is ignored but the statement is evaluated as if `innodb_strict_mode` is enabled.
- Use `DROP TABLESPACE` to remove a tablespace. All tables must be dropped from a tablespace using `DROP TABLE` prior to dropping the tablespace. Before dropping an NDB Cluster tablespace you

must also remove all its data files using one or more `ALTER TABLESPACE ... DROP DATAFILE` statements. See Section 22.5.13.1, “NDB Cluster Disk Data Objects”.

- All parts of an `InnoDB` table added to an `InnoDB` general tablespace reside in the general tablespace, including indexes and `BLOB` pages.

  For an `NDB` table assigned to a tablespace, only those columns which are not indexed are stored on disk, and actually use the tablespace data files. Indexes and indexed columns for all `NDB` tables are always kept in memory.

- Similar to the system tablespace, truncating or dropping tables stored in a general tablespace creates free space internally in the general tablespace .ibd data file which can only be used for new `InnoDB` data. Space is not released back to the operating system as it is for file-per-table tablespaces.

- A general tablespace is not associated with any database or schema.

- `ALTER TABLE ... DISCARD TABLESPACE` and `ALTER TABLE ...IMPORT TABLESPACE` are not supported for tables that belong to a general tablespace.

- The server uses tablespace-level metadata locking for DDL that references general tablespaces. By comparison, the server uses table-level metadata locking for DDL that references file-per-table tablespaces.

- A generated or existing tablespace cannot be changed to a general tablespace.

- There is no conflict between general tablespace names and file-per-table tablespace names. The “/” character, which is present in file-per-table tablespace names, is not permitted in general tablespace names.

- `mysqldump` and `mysqlpump` do not dump `InnoDB` `CREATE TABLESPACE` statements.

InnoDB Examples

This example demonstrates creating a general tablespace and adding three uncompressed tables of different row formats.

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' ENGINE=INNODB;

mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=REDUNDANT;

mysql> CREATE TABLE t2 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=COMPACT;

mysql> CREATE TABLE t3 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=DYNAMIC;
```

This example demonstrates creating a general tablespace and adding a compressed table. The example assumes a default `innodb_page_size` value of 16K. The `FILE_BLOCK_SIZE` of 8192 requires that the compressed table have a `KEY_BLOCK_SIZE` of 8.

```
mysql> CREATE TABLESPACE `ts2` ADD DATAFILE 'ts2.ibd' FILE_BLOCK_SIZE = 8192 Engine=InnoDB;

mysql> CREATE TABLE t4 (c1 INT PRIMARY KEY) TABLESPACE ts2 ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8;
```

This example demonstrates creating a general tablespace without specifying the `ADD DATAFILE` clause, which is optional as of MySQL 8.0.14.

```
mysql> CREATE TABLESPACE `ts3` ENGINE=INNODB;
```

This example demonstrates creating an undo tablespace.

```
mysql> CREATE UNDO TABLESPACE undo_003 ADD DATAFILE 'undo_003.ibu';
```

NDB Example

Suppose that you wish to create an NDB Cluster Disk Data tablespace named `myts` using a datafile named `mydata-1.dat`. An `NDB` tablespace always requires the use of a log file group consisting of one or more undo log files. For this example, we first create a log file group named `mylg` that contains one undo long file named `myundo-1.dat`, using the `CREATE LOGFILE GROUP` statement shown here:

```
mysql> CREATE LOGFILE GROUP myg1
    ->     ADD UNDOFILE 'myundo-1.dat'
    ->     ENGINE=NDB;
Query OK, 0 rows affected (3.29 sec)
```

Now you can create the tablespace previously described using the following statement:

```
mysql> CREATE TABLESPACE myts
    ->     ADD DATAFILE 'mydata-1.dat'
    ->     USE LOGFILE GROUP mylg
    ->     ENGINE=NDB;
Query OK, 0 rows affected (2.98 sec)
```

You can now create a Disk Data table using a `CREATE TABLE` statement with the `TABLESPACE` and `STORAGE DISK` options, similar to what is shown here:

```
mysql> CREATE TABLE mytable (
    ->     id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->     lname VARCHAR(50) NOT NULL,
    ->     fname VARCHAR(50) NOT NULL,
    ->     dob DATE NOT NULL,
    ->     joined DATE NOT NULL,
    ->     INDEX(last_name, first_name)
    -> )
    ->     TABLESPACE myts STORAGE DISK
    ->     ENGINE=NDB;
Query OK, 0 rows affected (1.41 sec)
```

It is important to note that only the `dob` and `joined` columns from `mytable` are actually stored on disk, due to the fact that the `id`, `lname`, and `fname` columns are all indexed.

As mentioned previously, when `CREATE TABLESPACE` is used with `ENGINE [=] NDB`, a tablespace and associated data file are created on each NDB Cluster data node. You can verify that the data files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table, as shown here:

```
mysql> SELECT FILE_NAME, FILE_TYPE, LOGFILE_GROUP_NAME, STATUS, EXTRA
    ->     FROM INFORMATION_SCHEMA.FILES
    ->     WHERE TABLESPACE_NAME = 'myts';

+--------------+------------+--------------------+--------+----------------+
| file_name    | file_type  | logfile_group_name | status | extra          |
+--------------+------------+--------------------+--------+----------------+
| mydata-1.dat | DATAFILE   | mylg               | NORMAL | CLUSTER_NODE=5 |
| mydata-1.dat | DATAFILE   | mylg               | NORMAL | CLUSTER_NODE=6 |
| NULL         | TABLESPACE | mylg               | NORMAL | NULL           |
+--------------+------------+--------------------+--------+----------------+
```

```
3 rows in set (0.01 sec)
```

For additional information and examples, see Section 22.5.13.1, “NDB Cluster Disk Data Objects”.

13.1.22 CREATE TRIGGER Syntax

```
CREATE
    [DEFINER = user]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. The trigger becomes associated with the table named *`tbl_name`*, which must refer to a permanent table. You cannot associate a trigger with a `TEMPORARY` table or a view.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

This section describes `CREATE TRIGGER` syntax. For additional discussion, see Section 24.3.1, “Trigger Syntax and Examples”.

`CREATE TRIGGER` requires the `TRIGGER` privilege for the table associated with the trigger. If the `DEFINER` clause is present, the privileges required depend on the *`user`* value, as discussed in Section 24.6, “Stored Object Access Control”. If binary logging is enabled, `CREATE TRIGGER` might require the `SUPER` privilege, as discussed in Section 24.7, “Stored Program Binary Logging”.

The `DEFINER` clause determines the security context to be used when checking access privileges at trigger activation time, as described later in this section.

`trigger_time` is the trigger action time. It can be `BEFORE` or `AFTER` to indicate that the trigger activates before or after each row to be modified.

Basic column value checks occur prior to trigger activation, so you cannot use `BEFORE` triggers to convert values inappropriate for the column type to valid values.

`trigger_event` indicates the kind of operation that activates the trigger. These *`trigger_event`* values are permitted:

- `INSERT`: The trigger activates whenever a new row is inserted into the table (for example, through `INSERT`, `LOAD DATA`, and `REPLACE` statements).
- `UPDATE`: The trigger activates whenever a row is modified (for example, through `UPDATE` statements).
- `DELETE`: The trigger activates whenever a row is deleted from the table (for example, through `DELETE` and `REPLACE` statements). `DROP TABLE` and `TRUNCATE TABLE` statements on the table do *not* activate this trigger, because they do not use `DELETE`. Dropping a partition does not activate `DELETE` triggers, either.

The *trigger_event* does not represent a literal type of SQL statement that activates the trigger so much as it represents a type of table operation. For example, an INSERT trigger activates not only for INSERT statements but also LOAD DATA statements because both statements insert rows into a table.

A potentially confusing example of this is the INSERT INTO ... ON DUPLICATE KEY UPDATE ... syntax: a BEFORE INSERT trigger activates for every row, followed by either an AFTER INSERT trigger or both the BEFORE UPDATE and AFTER UPDATE triggers, depending on whether there was a duplicate key for the row.

Note

Cascaded foreign key actions do not activate triggers.

It is possible to define multiple triggers for a given table that have the same trigger event and action time. For example, you can have two BEFORE UPDATE triggers for a table. By default, triggers that have the same trigger event and action time activate in the order they were created. To affect trigger order, specify a *trigger_order* clause that indicates FOLLOWS or PRECEDES and the name of an existing trigger that also has the same trigger event and action time. With FOLLOWS, the new trigger activates after the existing trigger. With PRECEDES, the new trigger activates before the existing trigger.

trigger_body is the statement to execute when the trigger activates. To execute multiple statements, use the BEGIN ... END compound statement construct. This also enables you to use the same statements that are permitted within stored routines. See Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”. Some statements are not permitted in triggers; see Section 24.8, “Restrictions on Stored Programs”.

Within the trigger body, you can refer to columns in the subject table (the table associated with the trigger) by using the aliases OLD and NEW. OLD.*col_name* refers to a column of an existing row before it is updated or deleted. NEW.*col_name* refers to the column of a new row to be inserted or an existing row after it is updated.

Triggers cannot use NEW.*col_name* or use OLD.*col_name* to refer to generated columns. For information about generated columns, see Section 13.1.20.9, “CREATE TABLE and Generated Columns”.

MySQL stores the sql_mode system variable setting in effect when a trigger is created, and always executes the trigger body with this setting in force, *regardless of the current server SQL mode when the trigger begins executing*.

The DEFINER clause specifies the MySQL account to be used when checking access privileges at trigger activation time. If the DEFINER clause is present, the *user* value should be a MySQL account specified as '*user_name*'@'*host_name*', CURRENT_USER, or CURRENT_USER(). The permitted *user* values depend on the privileges you hold, as discussed in Section 24.6, “Stored Object Access Control”. Also see that section for additional information about trigger security.

If the DEFINER clause is omitted, the default definer is the user who executes the CREATE TRIGGER statement. This is the same as specifying DEFINER = CURRENT_USER explicitly.

MySQL takes the DEFINER user into account when checking trigger privileges as follows:

- At CREATE TRIGGER time, the user who issues the statement must have the TRIGGER privilege.
- At trigger activation time, privileges are checked against the DEFINER user. This user must have these privileges:
  - The TRIGGER privilege for the subject table.

- The SELECT privilege for the subject table if references to table columns occur using OLD.*col_name* or NEW.*col_name* in the trigger body.
- The UPDATE privilege for the subject table if table columns are targets of SET NEW.*col_name* = *value* assignments in the trigger body.
- Whatever other privileges normally are required for the statements executed by the trigger.

Within a trigger body, the CURRENT_USER function returns the account used to check privileges at trigger activation time. This is the DEFINER user, not the user whose actions caused the trigger to be activated. For information about user auditing within triggers, see Section 6.2.22, “SQL-Based Account Activity Auditing”.

If you use LOCK TABLES to lock a table that has triggers, the tables used within the trigger are also locked, as described in LOCK TABLES and Triggers.

For additional discussion of trigger use, see Section 24.3.1, “Trigger Syntax and Examples”.

13.1.23 CREATE VIEW Syntax

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = user]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

The CREATE VIEW statement creates a new view, or replaces an existing view if the OR REPLACE clause is given. If the view does not exist, CREATE OR REPLACE VIEW is the same as CREATE VIEW. If the view does exist, CREATE OR REPLACE VIEW replaces it.

For information about restrictions on view use, see Section 24.9, “Restrictions on Views”.

The *select_statement* is a SELECT statement that provides the definition of the view. (Selecting from the view selects, in effect, using the SELECT statement.) The *select_statement* can select from base tables or other views.

The view definition is “frozen” at creation time and is not affected by subsequent changes to the definitions of the underlying tables. For example, if a view is defined as SELECT * on a table, new columns added to the table later do not become part of the view, and columns dropped from the table will result in an error when selecting from the view.

The ALGORITHM clause affects how MySQL processes the view. The DEFINER and SQL SECURITY clauses specify the security context to be used when checking access privileges at view invocation time. The WITH CHECK OPTION clause can be given to constrain inserts or updates to rows in tables referenced by the view. These clauses are described later in this section.

The CREATE VIEW statement requires the CREATE VIEW privilege for the view, and some privilege for each column selected by the SELECT statement. For columns used elsewhere in the SELECT statement, you must have the SELECT privilege. If the OR REPLACE clause is present, you must also have the DROP privilege for the view. If the DEFINER clause is present, the privileges required depend on the *user* value, as discussed in Section 24.6, “Stored Object Access Control”.

When a view is referenced, privilege checking occurs as described later in this section.

A view belongs to a database. By default, a new view is created in the default database. To create the view explicitly in a given database, use *db_name.view_name* syntax to qualify the view name with the database name:

```
CREATE VIEW test.v AS SELECT * FROM t;
```

Unqualified table or view names in the SELECT statement are also interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the appropriate database name.

Within a database, base tables and views share the same namespace, so a base table and a view cannot have the same name.

Columns retrieved by the SELECT statement can be simple references to table columns, or expressions that use functions, constant values, operators, and so forth.

A view must have unique column names with no duplicates, just like a base table. By default, the names of the columns retrieved by the SELECT statement are used for the view column names. To define explicit names for the view columns, specify the optional *column_list* clause as a list of comma-separated identifiers. The number of names in *column_list* must be the same as the number of columns retrieved by the SELECT statement.

A view can be created from many kinds of SELECT statements. It can refer to base tables or other views. It can use joins, UNION, and subqueries. The SELECT need not even refer to any tables:

```
CREATE VIEW v_today (today) AS SELECT CURRENT_DATE;
```

The following example defines a view that selects two columns from another table as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
+------+-------+-------+
```

A view definition is subject to the following restrictions:

- The SELECT statement cannot refer to system variables or user-defined variables.
- Within a stored program, the SELECT statement cannot refer to program parameters or local variables.
- The SELECT statement cannot refer to prepared statement parameters.
- Any table or view referred to in the definition must exist. If, after the view has been created, a table or view that the definition refers to is dropped, use of the view results in an error. To check a view definition for problems of this kind, use the CHECK TABLE statement.
- The definition cannot refer to a TEMPORARY table, and you cannot create a TEMPORARY view.
- You cannot associate a trigger with a view.
- Aliases for column names in the SELECT statement are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

`ORDER BY` is permitted in a view definition, but it is ignored if you select from a view using a statement that has its own `ORDER BY`.

For other options or clauses in the definition, they are added to the options or clauses of the statement that references the view, but the effect is undefined. For example, if a view definition includes a `LIMIT` clause, and you select from the view using a statement that has its own `LIMIT` clause, it is undefined which limit applies. This same principle applies to options such as `ALL`, `DISTINCT`, or `SQL_SMALL_RESULT` that follow the `SELECT` keyword, and to clauses such as `INTO`, `FOR UPDATE`, `FOR SHARE`, `LOCK IN SHARE MODE`, and `PROCEDURE`.

The results obtained from a view may be affected if you change the query processing environment by changing system variables:

```
mysql> CREATE VIEW v (mycol) AS SELECT 'abc';
Query OK, 0 rows affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-------+
| mycol |
+-------+
| mycol |
+-------+
1 row in set (0.01 sec)

mysql> SET sql_mode = 'ANSI_QUOTES';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-------+
| mycol |
+-------+
| abc   |
+-------+
1 row in set (0.00 sec)
```

The `DEFINER` and `SQL SECURITY` clauses determine which MySQL account to use when checking access privileges for the view when a statement is executed that references the view. The valid `SQL SECURITY` characteristic values are `DEFINER` (the default) and `INVOKER`. These indicate that the required privileges must be held by the user who defined or invoked the view, respectively.

If the `DEFINER` clause is present, the *`user`* value should be a MySQL account specified as *`'user_name'@'host_name'`*, `CURRENT_USER`, or `CURRENT_USER()`. The permitted *`user`* values depend on the privileges you hold, as discussed in Section 24.6, “Stored Object Access Control”. Also see that section for additional information about view security.

If the `DEFINER` clause is omitted, the default definer is the user who executes the `CREATE VIEW` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

Within a view definition, the `CURRENT_USER` function returns the view's `DEFINER` value by default. For views defined with the `SQL SECURITY INVOKER` characteristic, `CURRENT_USER` returns the account for the view's invoker. For information about user auditing within views, see Section 6.2.22, “SQL-Based Account Activity Auditing”.

Within a stored routine that is defined with the `SQL SECURITY DEFINER` characteristic, `CURRENT_USER` returns the routine's `DEFINER` value. This also affects a view defined within such a routine, if the view definition contains a `DEFINER` value of `CURRENT_USER`.

MySQL checks view privileges like this:

- At view definition time, the view creator must have the privileges needed to use the top-level objects accessed by the view. For example, if the view definition refers to table columns, the creator must have some privilege for each column in the select list of the definition, and the `SELECT` privilege for each column used elsewhere in the definition. If the definition refers to a stored function, only the privileges needed to invoke the function can be checked. The privileges required at function invocation time can be checked only as it executes: For different invocations, different execution paths within the function might be taken.
- The user who references a view must have appropriate privileges to access it (`SELECT` to select from it, `INSERT` to insert into it, and so forth.)
- When a view has been referenced, privileges for objects accessed by the view are checked against the privileges held by the view `DEFINER` account or invoker, depending on whether the `SQL SECURITY` characteristic is `DEFINER` or `INVOKER`, respectively.
- If reference to a view causes execution of a stored function, privilege checking for statements executed within the function depend on whether the function `SQL SECURITY` characteristic is `DEFINER` or `INVOKER`. If the security characteristic is `DEFINER`, the function runs with the privileges of the `DEFINER` account. If the characteristic is `INVOKER`, the function runs with the privileges determined by the view's `SQL SECURITY` characteristic.

Example: A view might depend on a stored function, and that function might invoke other stored routines. For example, the following view invokes a stored function `f()`:

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);
```

Suppose that `f()` contains a statement such as this:

```
IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```

The privileges required for executing statements within `f()` need to be checked when `f()` executes. This might mean that privileges are needed for `p1()` or `p2()`, depending on the execution path within `f()`. Those privileges must be checked at runtime, and the user who must possess the privileges is determined by the `SQL SECURITY` values of the view `v` and the function `f()`.

The `DEFINER` and `SQL SECURITY` clauses for views are extensions to standard SQL. In standard SQL, views are handled using the rules for `SQL SECURITY DEFINER`. The standard says that the definer of the view, which is the same as the owner of the view's schema, gets applicable privileges on the view (for example, `SELECT`) and may grant them. MySQL has no concept of a schema “owner”, so MySQL adds a clause to identify the definer. The `DEFINER` clause is an extension where the intent is to have what the standard has; that is, a permanent record of who defined the view. This is why the default `DEFINER` value is the account of the view creator.

The optional `ALGORITHM` clause is a MySQL extension to standard SQL. It affects how MySQL processes the view. `ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`. For more information, see Section 24.5.2, “View Processing Algorithms”, as well as Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”.

Some views are updatable. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one

relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable.

A generated column in a view is considered updatable because it is possible to assign to it. However, if such a column is updated explicitly, the only permitted value is `DEFAULT`. For information about generated columns, see Section 13.1.20.9, “CREATE TABLE and Generated Columns”.

The `WITH CHECK OPTION` clause can be given for an updatable view to prevent inserts or updates to rows except those for which the `WHERE` clause in the *`select_statement`* is true.

In a `WITH CHECK OPTION` clause for an updatable view, the `LOCAL` and `CASCADED` keywords determine the scope of check testing when the view is defined in terms of another view. The `LOCAL` keyword restricts the `CHECK OPTION` only to the view being defined. `CASCADED` causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is `CASCADED`.

For more information about updatable views and the `WITH CHECK OPTION` clause, see Section 24.5.3, “Updatable and Insertable Views”, and Section 24.5.4, “The View WITH CHECK OPTION Clause”.

13.1.24 DROP DATABASE Syntax

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

`DROP DATABASE` drops all tables in the database and deletes the database. Be *very* careful with this statement! To use `DROP DATABASE`, you need the `DROP` privilege on the database. `DROP SCHEMA` is a synonym for `DROP DATABASE`.

Important

When a database is dropped, privileges granted specifically for the database are *not* automatically dropped. They must be dropped manually. See Section 13.7.1.6, “GRANT Syntax”.

`IF EXISTS` is used to prevent an error from occurring if the database does not exist.

If the default database is dropped, the default database is unset (the `DATABASE()` function returns `NULL`).

If you use `DROP DATABASE` on a symbolically linked database, both the link and the original database are deleted.

`DROP DATABASE` returns the number of tables that were removed.

The `DROP DATABASE` statement removes from the given database directory those files and directories that MySQL itself may create during normal operation. This includes all files with the extensions shown in the following list:

- `.BAK`
- `.DAT`
- `.HSH`
- `.MRG`
- `.MYD`
- `.MYI`
- `.cfg`

- `.db`
- `.ibd`
- `.ndb`

If other files or directories remain in the database directory after MySQL removes those just listed, the database directory cannot be removed. In this case, you must remove any remaining files or directories manually and issue the `DROP DATABASE` statement again.

Dropping a database does not remove any `TEMPORARY` tables that were created in that database. `TEMPORARY` tables are automatically removed when the session that created them ends. See Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”.

You can also drop databases with `mysqladmin`. See Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”.

13.1.25 DROP EVENT Syntax

```
DROP EVENT [IF EXISTS] event_name
```

This statement drops the event named *`event_name`*. The event immediately ceases being active, and is deleted completely from the server.

If the event does not exist, the error `ERROR 1517 (HY000): Unknown event 'event_name'` results. You can override this and cause the statement to generate a warning for nonexistent events instead using `IF EXISTS`.

This statement requires the `EVENT` privilege for the schema to which the event to be dropped belongs.

13.1.26 DROP FUNCTION Syntax

The `DROP FUNCTION` statement is used to drop stored functions and user-defined functions (UDFs):

- For information about dropping stored functions, see Section 13.1.29, “DROP PROCEDURE and DROP FUNCTION Syntax”.
- For information about dropping user-defined functions, see Section 13.7.4.2, “DROP FUNCTION Syntax”.

13.1.27 DROP INDEX Syntax

```
DROP INDEX index_name ON tbl_name
    [algorithm_option | lock_option] ...

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

`DROP INDEX` drops the index named *`index_name`* from the table *`tbl_name`*. This statement is mapped to an `ALTER TABLE` statement to drop the index. See Section 13.1.9, “ALTER TABLE Syntax”.

To drop a primary key, the index name is always `PRIMARY`, which must be specified as a quoted identifier because `PRIMARY` is a reserved word:

```
DROP INDEX `PRIMARY` ON t;
```

Indexes on variable-width columns of NDB tables are dropped online; that is, without any table copying. The table is not locked against access from other NDB Cluster API nodes, although it is locked against other operations on the *same* API node for the duration of the operation. This is done automatically by the server whenever it determines that it is possible to do so; you do not have to use any special SQL syntax or server options to cause it to happen.

ALGORITHM and LOCK clauses may be given to influence the table copying method and level of concurrency for reading and writing the table while its indexes are being modified. They have the same meaning as for the ALTER TABLE statement. For more information, see Section 13.1.9, “ALTER TABLE Syntax”

MySQL NDB Cluster supports online operations using the same ALGORITHM=INPLACE syntax supported in the standard MySQL Server. See Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”, for more information.

13.1.28 DROP LOGFILE GROUP Syntax

```
DROP LOGFILE GROUP logfile_group
    ENGINE [=] engine_name
```

This statement drops the log file group named *logfile_group*. The log file group must already exist or an error results. (For information on creating log file groups, see Section 13.1.16, “CREATE LOGFILE GROUP Syntax”.)

Important

Before dropping a log file group, you must drop all tablespaces that use that log file group for UNDO logging.

The required ENGINE clause provides the name of the storage engine used by the log file group to be dropped. Currently, the only permitted values for *engine_name* are NDB and NDBCLUSTER.

DROP LOGFILE GROUP is useful only with Disk Data storage for NDB Cluster. See Section 22.5.13, “NDB Cluster Disk Data Tables”.

13.1.29 DROP PROCEDURE and DROP FUNCTION Syntax

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

This statement is used to drop a stored procedure or function. That is, the specified routine is removed from the server. You must have the ALTER ROUTINE privilege for the routine. (If the automatic_sp_privileges system variable is enabled, that privilege and EXECUTE are granted automatically to the routine creator when the routine is created and dropped from the creator when the routine is dropped. See Section 24.2.2, “Stored Routines and MySQL Privileges”.)

The IF EXISTS clause is a MySQL extension. It prevents an error from occurring if the procedure or function does not exist. A warning is produced that can be viewed with SHOW WARNINGS.

DROP FUNCTION is also used to drop user-defined functions (see Section 13.7.4.2, “DROP FUNCTION Syntax”).

13.1.30 DROP SERVER Syntax

```
DROP SERVER [ IF EXISTS ] server_name
```

Drops the server definition for the server named *`server_name`*. The corresponding row in the `mysql.servers` table is deleted. This statement requires the `SUPER` privilege.

Dropping a server for a table does not affect any `FEDERATED` tables that used this connection information when they were created. See Section 13.1.18, “CREATE SERVER Syntax”.

`DROP SERVER` causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

`DROP SERVER` is not written to the binary log, regardless of the logging format that is in use.

13.1.31 DROP SPATIAL REFERENCE SYSTEM Syntax

```
DROP SPATIAL REFERENCE SYSTEM
    [IF EXISTS]
    srid

srid: 32-bit unsigned integer
```

This statement removes a spatial reference system (SRS) definition from the data dictionary. It requires the `SUPER` privilege.

Example:

```
DROP SPATIAL REFERENCE SYSTEM 4120;
```

If no SRS definition with the SRID value exists, an error occurs unless `IF EXISTS` is specified. In that case, a warning occurs rather than an error.

If the SRID value is used by some column in an existing table, an error occurs. For example:

```
mysql> DROP SPATIAL REFERENCE SYSTEM 4326;
ERROR 3716 (SR005): Can't modify SRID 4326. There is at
least one column depending on it.
```

To identify which column or columns use the SRID, use this query:

```
SELECT * FROM INFORMATION_SCHEMA.ST_GEOMETRY_COLUMNS WHERE SRS_ID=4326;
```

SRID values must be in the range of 32-bit unsigned integers, with these restrictions:

- SRID 0 is a valid SRID but cannot be used with `DROP SPATIAL REFERENCE SYSTEM`.
- If the value is in a reserved SRID range, a warning occurs. Reserved ranges are [0, 32767] (reserved by EPSG), [60,000,000, 69,999,999] (reserved by EPSG), and [2,000,000,000, 2,147,483,647] (reserved by MySQL). EPSG stands for the European Petroleum Survey Group.
- Users should not drop SRSs with SRIDs in the reserved ranges. If system-installed SRSs are dropped, the SRS definitions may be recreated for MySQL upgrades.

13.1.32 DROP TABLE Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

`DROP TABLE` removes one or more tables. You must have the `DROP` privilege for each table.

Be careful with this statement! For each table, it removes the table definition and all table data. If the table is partitioned, the statement removes the table definition, all its partitions, all data stored in those partitions, and all partition definitions associated with the dropped table.

Dropping a table also drops any triggers for the table.

`DROP TABLE` causes an implicit commit, except when used with the `TEMPORARY` keyword. See Section 13.3.3, "Statements That Cause an Implicit Commit".

Important

When a table is dropped, privileges granted specifically for the table are *not* automatically dropped. They must be dropped manually. See Section 13.7.1.6, "GRANT Syntax".

If any tables named in the argument list do not exist, `DROP TABLE` behavior depends on whether the `IF EXISTS` clause is given:

- Without `IF EXISTS`, the statement fails with an error indicating which nonexisting tables it was unable to drop, and no changes are made.
- With `IF EXISTS`, no error occurs for nonexisting tables. The statement drops all named tables that do exist, and generates a `NOTE` diagnostic for each nonexistent table. These notes can be displayed with `SHOW WARNINGS`. See Section 13.7.7.40, "SHOW WARNINGS Syntax".

`IF EXISTS` can also be useful for dropping tables in unusual circumstances under which there is an entry in the data dictionary but no table managed by the storage engine. (For example, if an abnormal server exit occurs after removal of the table from the storage engine but before removal of the data dictionary entry.)

The `TEMPORARY` keyword has the following effects:

- The statement drops only `TEMPORARY` tables.
- The statement does not cause an implicit commit.
- No access rights are checked. A `TEMPORARY` table is visible only with the session that created it, so no check is necessary.

Including the `TEMPORARY` keyword is a good way to prevent accidentally dropping non-`TEMPORARY` tables.

The `RESTRICT` and `CASCADE` keywords do nothing. They are permitted to make porting easier from other database systems.

`DROP TABLE` is not supported with all `innodb_force_recovery` settings. See Section 15.20.2, "Forcing InnoDB Recovery".

13.1.33 DROP TABLESPACE Syntax

```
DROP [UNDO] TABLESPACE tablespace_name
    [ENGINE [=] engine_name]
```

This statement drops a tablespace that was previously created using `CREATE TABLESPACE`. It is supported by the `NDB` and `InnoDB` storage engines.

The `UNDO` keyword, introduced in MySQL 8.0.14, must be specified to drop an undo tablespace. Only undo tablespaces created using `CREATE UNDO TABLESPACE` syntax can be dropped. An undo tablespace

must be in an `empty` state before it can be dropped. For more information, see Section 15.6.3.4, “Undo Tablespaces”.

`ENGINE` sets the storage engine that uses the tablespace, where *`engine_name`* is the name of the storage engine. Currently, the values `InnoDB` and `NDB` are supported. If not set, the value of `default_storage_engine` is used. If it is not the same as the storage engine used to create the tablespace, the `DROP TABLESPACE` statement fails.

`tablespace_name` is a case-sensitive identifier in MySQL.

For an `InnoDB` general tablespace, all tables must be dropped from the tablespace prior to a `DROP TABLESPACE` operation. If the tablespace is not empty, `DROP TABLESPACE` returns an error.

An `NDB` tablespace to be dropped must not contain any data files; in other words, before you can drop an `NDB` tablespace, you must first drop each of its data files using `ALTER TABLESPACE ... DROP DATAFILE`.

Notes

- A general `InnoDB` tablespace is not deleted automatically when the last table in the tablespace is dropped. The tablespace must be dropped explicitly using `DROP TABLESPACE` *`tablespace_name`*.
- A `DROP DATABASE` operation can drop tables that belong to a general tablespace but it cannot drop the tablespace, even if the operation drops all tables that belong to the tablespace. The tablespace must be dropped explicitly using `DROP TABLESPACE` *`tablespace_name`*.
- Similar to the system tablespace, truncating or dropping tables stored in a general tablespace creates free space internally in the general tablespace .ibd data file which can only be used for new `InnoDB` data. Space is not released back to the operating system as it is for file-per-table tablespaces.

InnoDB Examples

This example demonstrates how to drop an `InnoDB` general tablespace. The general tablespace `ts1` is created with a single table. Before dropping the tablespace, the table must be dropped.

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' Engine=InnoDB;

mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts10 Engine=InnoDB;

mysql> DROP TABLE t1;

mysql> DROP TABLESPACE ts1;
```

This example demonstrates dropping an undo tablespace. An undo tablespace must be in an `empty` state before it can be dropped. For more information, see Section 15.6.3.4, “Undo Tablespaces”.

```
mysql> DROP UNDO TABLESPACE undo_003;
```

NDB Example

This example shows how to drop an `NDB` tablespace `myts` having a data file named `mydata-1.dat` after first creating the tablespace, and assumes the existence of a log file group named `mylg` (see Section 13.1.16, “CREATE LOGFILE GROUP Syntax”).

```
mysql> CREATE TABLESPACE myts
    ->     ADD DATAFILE 'mydata-1.dat'
    ->     USE LOGFILE GROUP mylg
```

```
    ->     ENGINE=NDB;
```

You must remove all data files from the tablespace using `ALTER TABLESPACE`, as shown here, before it can be dropped:

```
mysql> ALTER TABLESPACE myts
    ->     DROP DATAFILE 'mydata-1.dat'
    ->     ENGINE=NDB;

mysql> DROP TABLESPACE myts;
```

13.1.34 DROP TRIGGER Syntax

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

This statement drops a trigger. The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. `DROP TRIGGER` requires the `TRIGGER` privilege for the table associated with the trigger.

Use `IF EXISTS` to prevent an error from occurring for a trigger that does not exist. A `NOTE` is generated for a nonexistent trigger when using `IF EXISTS`. See Section 13.7.7.40, “SHOW WARNINGS Syntax”.

Triggers for a table are also dropped if you drop the table.

13.1.35 DROP VIEW Syntax

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
    [RESTRICT | CASCADE]
```

`DROP VIEW` removes one or more views. You must have the `DROP` privilege for each view.

If any views named in the argument list do not exist, the statement fails with an error indicating by name which nonexisting views it was unable to drop, and no changes are made.

Note

In MySQL 5.7 and earlier, `DROP VIEW` returns an error if any views named in the argument list do not exist, but also drops all views in the list that do exist. Due to the change in behavior in MySQL 8.0, a partially completed `DROP VIEW` operation on a MySQL 5.7 master fails when replicated on a MySQL 8.0 slave. To avoid this failure scenario, use `IF EXISTS` syntax in `DROP VIEW` statements to prevent an error from occurring for views that do not exist. For more information, see Section 13.1.1, “Atomic Data Definition Statement Support”.

The `IF EXISTS` clause prevents an error from occurring for views that don't exist. When this clause is given, a `NOTE` is generated for each nonexistent view. See Section 13.7.7.40, “SHOW WARNINGS Syntax”.

`RESTRICT` and `CASCADE`, if given, are parsed and ignored.

13.1.36 RENAME TABLE Syntax

```
RENAME TABLE
    tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

`RENAME TABLE` renames one or more tables. You must have `ALTER` and `DROP` privileges for the original table, and `CREATE` and `INSERT` privileges for the new table.

For example, to rename a table named `old_table` to `new_table`, use this statement:

```
RENAME TABLE old_table TO new_table;
```

That statement is equivalent to the following `ALTER TABLE` statement:

```
ALTER TABLE old_table RENAME new_table;
```

`RENAME TABLE`, unlike `ALTER TABLE`, can rename multiple tables within a single statement:

```
RENAME TABLE old_table1 TO new_table1,
             old_table2 TO new_table2,
             old_table3 TO new_table3;
```

Renaming operations are performed left to right. Thus, to swap two table names, do this (assuming that a table with the intermediary name `tmp_table` does not already exist):

```
RENAME TABLE old_table TO tmp_table,
             new_table TO old_table,
             tmp_table TO new_table;
```

Metadata locks on tables are acquired in name order, which in some cases can make a difference in operation outcome when multiple transactions execute concurrently. See Section 8.11.4, “Metadata Locking”.

As of MySQL 8.0.13, you can rename tables locked with a `LOCK TABLES` statement, provided that they are locked with a `WRITE` lock or are the product of renaming `WRITE`-locked tables from earlier steps in a multiple-table rename operation. For example, this is permitted:

```
LOCK TABLE old_table1 WRITE;
RENAME TABLE old_table1 TO new_table1,
             new_table1 TO new_table2;
```

This is not permitted:

```
LOCK TABLE old_table1 READ;
RENAME TABLE old_table1 TO new_table1,
             new_table1 TO new_table2;
```

Prior to MySQL 8.0.13, to execute `RENAME TABLE`, there must be no tables locked with `LOCK TABLES`.

With the transaction table locking conditions satisfied, the rename operation is done atomically; no other session can access any of the tables while the rename is in progress.

If any errors occur during a `RENAME TABLE`, the statement fails and no changes are made.

You can use `RENAME TABLE` to move a table from one database to another:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

Using this method to move all tables from one database to a different one in effect renames the database (an operation for which MySQL has no single statement), except that the original database continues to exist, albeit with no tables.

Like `RENAME TABLE`, `ALTER TABLE ... RENAME` can also be used to move a table to a different database. Regardless of the statement used, if the rename operation would move the table to a database located on a different file system, the success of the outcome is platform specific and depends on the underlying operating system calls used to move table files.

If a table has triggers, attempts to rename the table into a different database fail with a `Trigger in wrong schema` (`ER_TRG_IN_WRONG_SCHEMA`) error.

An unencrypted table can be moved to an encryption-enabled database and vice versa. However, if the `table_encryption_privilege_check` variable is enabled, the `TABLE_ENCRYPTION_ADMIN` privilege is required if the table encryption setting differs from the default database encryption.

To rename `TEMPORARY` tables, `RENAME TABLE` does not work. Use `ALTER TABLE` instead.

`RENAME TABLE` works for views, except that views cannot be renamed into a different database.

Any privileges granted specifically for a renamed table or view are not migrated to the new name. They must be changed manually.

`RENAME TABLE` *`tbl_name`* `TO` *`new_tbl_name`* changes internally generated foreign key constraint names and user-defined foreign key constraint names that begin with the string "*`tbl_name`*_ibfk_" to reflect the new table name. `InnoDB` interprets foreign key constraint names that begin with the string "*`tbl_name`*_ibfk_" as internally generated names.

Foreign key constraint names that point to the renamed table are automatically updated unless there is a conflict, in which case the statement fails with an error. A conflict occurs if the renamed constraint name already exists. In such cases, you must drop and re-create the foreign keys for them to function properly.

`RENAME TABLE` *`tbl_name`* `TO` *`new_tbl_name`* changes internally generated and user-defined `CHECK` constraint names that begin with the string "*`tbl_name`*_chk_" to reflect the new table name. MySQL interprets `CHECK` constraint names that begin with the string "*`tbl_name`*_chk_" as internally generated names. Example:

```
mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `i1` int(11) DEFAULT NULL,
  `i2` int(11) DEFAULT NULL,
  CONSTRAINT `t1_chk_1` CHECK ((`i1` > 0)),
  CONSTRAINT `t1_chk_2` CHECK ((`i2` < 0))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.02 sec)

mysql> RENAME TABLE t1 TO t3;
Query OK, 0 rows affected (0.03 sec)

mysql> SHOW CREATE TABLE t3\G
*************************** 1. row ***************************
       Table: t3
Create Table: CREATE TABLE `t3` (
  `i1` int(11) DEFAULT NULL,
  `i2` int(11) DEFAULT NULL,
  CONSTRAINT `t3_chk_1` CHECK ((`i1` > 0)),
  CONSTRAINT `t3_chk_2` CHECK ((`i2` < 0))
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.01 sec)
```

13.1.37 TRUNCATE TABLE Syntax

```
TRUNCATE [TABLE] tbl_name
```

`TRUNCATE TABLE` empties a table completely. It requires the `DROP` privilege. Logically, `TRUNCATE TABLE` is similar to a `DELETE` statement that deletes all rows, or a sequence of `DROP TABLE` and `CREATE TABLE` statements.

To achieve high performance, `TRUNCATE TABLE` bypasses the DML method of deleting data. Thus, it does not cause `ON DELETE` triggers to fire, it cannot be performed for `InnoDB` tables with parent-child foreign key relationships, and it cannot be rolled back like a DML operation. However, `TRUNCATE TABLE` operations on tables that use an atomic DDL-supported storage engine are either fully committed or rolled back if the server halts during their operation. For more information, see Section 13.1.1, “Atomic Data Definition Statement Support”.

Although `TRUNCATE TABLE` is similar to `DELETE`, it is classified as a DDL statement rather than a DML statement. It differs from `DELETE` in the following ways:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one, particularly for large tables.
- Truncate operations cause an implicit commit, and so cannot be rolled back. See Section 13.3.3, “Statements That Cause an Implicit Commit”.
- Truncation operations cannot be performed if the session holds an active table lock.
- `TRUNCATE TABLE` fails for an `InnoDB` table or `NDB` table if there are any `FOREIGN KEY` constraints from other tables that reference the table. Foreign key constraints between columns of the same table are permitted.
- Truncation operations do not return a meaningful value for the number of deleted rows. The usual result is “0 rows affected,” which should be interpreted as “no information.”
- As long as the table definition is valid, the table can be re-created as an empty table with `TRUNCATE TABLE`, even if the data or index files have become corrupted.
- Any `AUTO_INCREMENT` value is reset to its start value. This is true even for `MyISAM` and `InnoDB`, which normally do not reuse sequence values.
- When used with partitioned tables, `TRUNCATE TABLE` preserves the partitioning; that is, the data and index files are dropped and re-created, while the partition definitions are unaffected.
- The `TRUNCATE TABLE` statement does not invoke `ON DELETE` triggers.
- Truncating a corrupted `InnoDB` table is supported.

`TRUNCATE TABLE` for a table closes all handlers for the table that were opened with `HANDLER OPEN`.

`TRUNCATE TABLE` is treated for purposes of binary logging and replication as `DROP TABLE` followed by `CREATE TABLE`—that is, as DDL rather than DML. This is due to the fact that, when using `InnoDB` and other transactional storage engines where the transaction isolation level does not permit statement-based logging (`READ COMMITTED` or `READ UNCOMMITTED`), the statement was not logged and replicated when using `STATEMENT` or `MIXED` logging mode. (Bug #36763) However, it is still applied on replication slaves using `InnoDB` in the manner described previously.

In MySQL 5.7 and earlier, on a system with a large buffer pool and `innodb_adaptive_hash_index` enabled, a `TRUNCATE TABLE` operation could cause a temporary drop in system performance due to an LRU scan that occurred when removing the table's adaptive hash index entries (Bug #68184).

The remapping of `TRUNCATE TABLE` to `DROP TABLE` and `CREATE TABLE` in MySQL 8.0 avoids the problematic LRU scan.

`TRUNCATE TABLE` can be used with Performance Schema summary tables, but the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. See Section 26.12.17, "Performance Schema Summary Tables".

13.2 Data Manipulation Statements

13.2.1 CALL Syntax

```
CALL sp_name([parameter[,...]])
CALL sp_name[()]
```

The `CALL` statement invokes a stored procedure that was defined previously with `CREATE PROCEDURE`.

Stored procedures that take no arguments can be invoked without parentheses. That is, `CALL p()` and `CALL p` are equivalent.

`CALL` can pass back values to its caller using parameters that are declared as `OUT` or `INOUT` parameters. When the procedure returns, a client program can also obtain the number of rows affected for the final statement executed within the routine: At the SQL level, call the `ROW_COUNT()` function; from the C API, call the `mysql_affected_rows()` function.

For information about the effect of unhandled conditions on procedure parameters, see Section 13.6.7.8, "Condition Handling and OUT or INOUT Parameters".

To get back a value from a procedure using an `OUT` or `INOUT` parameter, pass the parameter by means of a user variable, and then check the value of the variable after the procedure returns. (If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.) For an `INOUT` parameter, initialize its value before passing it to the procedure. The following procedure has an `OUT` parameter that the procedure sets to the current server version, and an `INOUT` value that the procedure increments by one from its current value:

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
  # Set value of OUT parameter
  SELECT VERSION() INTO ver_param;
  # Increment value of INOUT parameter
  SET incr_param = incr_param + 1;
END;
```

Before calling the procedure, initialize the variable to be passed as the `INOUT` parameter. After calling the procedure, the values of the two variables will have been set or modified:

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+--------------------+------------+
| @version           | @increment |
+--------------------+------------+
| 8.0.3-rc-debug-log |         11 |
+--------------------+------------+
```

In prepared `CALL` statements used with `PREPARE` and `EXECUTE`, placeholders can be used for `IN` parameters, `OUT`, and `INOUT` parameters. These types of parameters can be used as follows:

```
mysql> SET @increment = 10;
mysql> PREPARE s FROM 'CALL p(?, ?)';
mysql> EXECUTE s USING @version, @increment;
mysql> SELECT @version, @increment;
+--------------------+------------+
| @version           | @increment |
+--------------------+------------+
| 8.0.3-rc-debug-log |         11 |
+--------------------+------------+
```

To write C programs that use the `CALL` SQL statement to execute stored procedures that produce result sets, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. `CLIENT_MULTI_RESULTS` must also be enabled if `CALL` is used to execute any stored procedure that contains prepared statements. It cannot be determined when such a procedure is loaded whether those statements will produce result sets, so it is necessary to assume that they will.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). `CLIENT_MULTI_RESULTS` is enabled by default.

To process the result of a `CALL` statement executed using `mysql_query()` or `mysql_real_query()`, use a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see Section 28.7.23, “C API Multiple Statement Execution Support”.

C programs can use the prepared-statement interface to execute `CALL` statements and access `OUT` and `INOUT` parameters. This is done by processing the result of a `CALL` statement using a loop that calls `mysql_stmt_next_result()` to determine whether there are more results. For an example, see Section 28.7.25, “C API Prepared CALL Statement Support”. Languages that provide a MySQL interface can use prepared `CALL` statements to directly retrieve `OUT` and `INOUT` procedure parameters.

Metadata changes to objects referred to by stored programs are detected and cause automatic reparsing of the affected statements when the program is next executed. For more information, see Section 8.10.3, “Caching of Prepared Statements and Stored Programs”.

13.2.2 DELETE Syntax

`DELETE` is a DML statement that removes rows from a table.

A `DELETE` statement can start with a `WITH` clause to define common table expressions accessible within the `DELETE`. See Section 13.2.13, “WITH Syntax (Common Table Expressions)”.

Single-Table Syntax

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name [[AS] tbl_alias]
    [PARTITION (partition_name [, partition_name] ...)]
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

The `DELETE` statement deletes rows from *`tbl_name`* and returns the number of deleted rows. To check the number of deleted rows, call the `ROW_COUNT()` function described in Section 12.15, “Information Functions”.

Main Clauses

The conditions in the optional `WHERE` clause identify which rows to delete. With no `WHERE` clause, all rows are deleted.

where_condition is an expression that evaluates to true for each row to be deleted. It is specified as described in Section 13.2.10, “SELECT Syntax”.

If the ORDER BY clause is specified, the rows are deleted in the order that is specified. The LIMIT clause places a limit on the number of rows that can be deleted. These clauses apply to single-table deletes, but not multi-table deletes.

Multiple-Table Syntax

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

Privileges

You need the DELETE privilege on a table to delete rows from it. You need only the SELECT privilege for any columns that are only read, such as those named in the WHERE clause.

Performance

When you do not need to know the number of deleted rows, the TRUNCATE TABLE statement is a faster way to empty a table than a DELETE statement with no WHERE clause. Unlike DELETE, TRUNCATE TABLE cannot be used within a transaction or if you have a lock on the table. See Section 13.1.37, “TRUNCATE TABLE Syntax” and Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”.

The speed of delete operations may also be affected by factors discussed in Section 8.2.5.3, “Optimizing DELETE Statements”.

To ensure that a given DELETE statement does not take too much time, the MySQL-specific LIMIT *row_count* clause for DELETE specifies the maximum number of rows to be deleted. If the number of rows to delete is larger than the limit, repeat the DELETE statement until the number of affected rows is less than the LIMIT value.

Subqueries

You cannot delete from a table and select from the same table in a subquery.

Partitioned Table Support

DELETE supports explicit partition selection using the PARTITION option, which takes a list of the comma-separated names of one or more partitions or subpartitions (or both) from which to select rows to be dropped. Partitions not included in the list are ignored. Given a partitioned table t with a partition named p0, executing the statement DELETE FROM t PARTITION (p0) has the same effect on the table as executing ALTER TABLE t TRUNCATE PARTITION (p0); in both cases, all rows in partition p0 are dropped.

PARTITION can be used along with a WHERE condition, in which case the condition is tested only on rows in the listed partitions. For example, DELETE FROM t PARTITION (p0) WHERE c < 5 deletes rows only from partition p0 for which the condition c < 5 is true; rows in any other partitions are not checked and thus not affected by the DELETE.

The PARTITION option can also be used in multiple-table DELETE statements. You can use up to one such option per table named in the FROM option.

For more information and examples, see Section 23.5, “Partition Selection”.

Auto-Increment Columns

If you delete the row containing the maximum value for an AUTO_INCREMENT column, the value is not reused for a MyISAM or InnoDB table. If you delete all rows in the table with DELETE FROM *tbl_name* (without a WHERE clause) in autocommit mode, the sequence starts over for all storage engines except InnoDB and MyISAM. There are some exceptions to this behavior for InnoDB tables, as discussed in Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”.

For MyISAM tables, you can specify an AUTO_INCREMENT secondary column in a multiple-column key. In this case, reuse of values deleted from the top of the sequence occurs even for MyISAM tables. See Section 3.6.9, “Using AUTO_INCREMENT”.

Modifiers

The DELETE statement supports the following modifiers:

- If you specify the LOW_PRIORITY modifier, the server delays execution of the DELETE until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).
- For MyISAM tables, if you use the QUICK modifier, the storage engine does not merge index leaves during delete, which may speed up some kinds of delete operations.
- The IGNORE modifier causes MySQL to ignore errors during the process of deleting rows. (Errors encountered during the parsing stage are processed in the usual manner.) Errors that are ignored due to the use of IGNORE are returned as warnings. For more information, see Comparison of the IGNORE Keyword and Strict SQL Mode.

Order of Deletion

If the DELETE statement includes an ORDER BY clause, rows are deleted in the order specified by the clause. This is useful primarily in conjunction with LIMIT. For example, the following statement finds rows matching the WHERE clause, sorts them by timestamp_column, and deletes the first (oldest) one:

```
DELETE FROM somelog WHERE user = 'jcole'
ORDER BY timestamp_column LIMIT 1;
```

ORDER BY also helps to delete rows in an order required to avoid referential integrity violations.

InnoDB Tables

If you are deleting many rows from a large table, you may exceed the lock table size for an InnoDB table. To avoid this problem, or simply to minimize the time that the table remains locked, the following strategy (which does not use DELETE at all) might be helpful:

1. Select the rows *not* to be deleted into an empty table that has the same structure as the original table:

   ```
   INSERT INTO t_copy SELECT * FROM t WHERE ... ;
   ```

2. Use RENAME TABLE to atomically move the original table out of the way and rename the copy to the original name:

```
RENAME TABLE t TO t_old, t_copy TO t;
```

3. Drop the original table:

```
DROP TABLE t_old;
```

No other sessions can access the tables involved while `RENAME TABLE` executes, so the rename operation is not subject to concurrency problems. See Section 13.1.36, “RENAME TABLE Syntax”.

MyISAM Tables

In `MyISAM` tables, deleted rows are maintained in a linked list and subsequent `INSERT` operations reuse old row positions. To reclaim unused space and reduce file sizes, use the `OPTIMIZE TABLE` statement or the `myisamchk` utility to reorganize tables. `OPTIMIZE TABLE` is easier to use, but `myisamchk` is faster. See Section 13.7.3.4, “OPTIMIZE TABLE Syntax”, and Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”.

The `QUICK` modifier affects whether index leaves are merged for delete operations. `DELETE QUICK` is most useful for applications where index values for deleted rows are replaced by similar index values from rows inserted later. In this case, the holes left by deleted values are reused.

`DELETE QUICK` is not useful when deleted values lead to underfilled index blocks spanning a range of index values for which new inserts occur again. In this case, use of `QUICK` can lead to wasted space in the index that remains unreclaimed. Here is an example of such a scenario:

1. Create a table that contains an indexed `AUTO_INCREMENT` column.
2. Insert many rows into the table. Each insert results in an index value that is added to the high end of the index.
3. Delete a block of rows at the low end of the column range using `DELETE QUICK`.

In this scenario, the index blocks associated with the deleted index values become underfilled but are not merged with other index blocks due to the use of `QUICK`. They remain underfilled when new inserts occur, because new rows do not have index values in the deleted range. Furthermore, they remain underfilled even if you later use `DELETE` without `QUICK`, unless some of the deleted index values happen to lie in index blocks within or adjacent to the underfilled blocks. To reclaim unused index space under these circumstances, use `OPTIMIZE TABLE`.

If you are going to delete many rows from a table, it might be faster to use `DELETE QUICK` followed by `OPTIMIZE TABLE`. This rebuilds the index rather than performing many index block merge operations.

Multi-Table Deletes

You can specify multiple tables in a `DELETE` statement to delete rows from one or more tables depending on the condition in the `WHERE` clause. You cannot use `ORDER BY` or `LIMIT` in a multiple-table `DELETE`. The *`table_references`* clause lists the tables involved in the join, as described in Section 13.2.10.2, “JOIN Syntax”.

For the first multiple-table syntax, only matching rows from the tables listed before the `FROM` clause are deleted. For the second multiple-table syntax, only matching rows from the tables listed in the `FROM` clause (before the `USING` clause) are deleted. The effect is that you can delete rows from many tables at the same time and have additional tables that are used only for searching:

```
```

```
DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

Or:

```
DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

These statements use all three tables when searching for rows to delete, but delete matching rows only from tables `t1` and `t2`.

The preceding examples use `INNER JOIN`, but multiple-table `DELETE` statements can use other types of join permitted in `SELECT` statements, such as `LEFT JOIN`. For example, to delete rows that exist in `t1` that have no match in `t2`, use a `LEFT JOIN`:

```
DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL;
```

The syntax permits `.*` after each *`tbl_name`* for compatibility with `Access`.

If you use a multiple-table `DELETE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, you should delete from a single table and rely on the `ON DELETE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly.

Note

If you declare an alias for a table, you must use the alias when referring to the table:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

Table aliases in a multiple-table `DELETE` should be declared only in the *`table_references`* part of the statement. Elsewhere, alias references are permitted but not alias declarations.

Correct:

```
DELETE a1, a2 FROM t1 AS a1 INNER JOIN t2 AS a2
WHERE a1.id=a2.id;

DELETE FROM a1, a2 USING t1 AS a1 INNER JOIN t2 AS a2
WHERE a1.id=a2.id;
```

Incorrect:

```
DELETE t1 AS a1, t2 AS a2 FROM t1 INNER JOIN t2
WHERE a1.id=a2.id;

DELETE FROM t1 AS a1, t2 AS a2 USING t1 INNER JOIN t2
WHERE a1.id=a2.id;
```

Table aliases are also supported for single-table `DELETE` statements beginning with MySQL 8.0.16. (Bug #89410,Bug #27455809)

13.2.3 DO Syntax

```
DO expr [, expr] ...
```

`DO` executes the expressions but does not return any results. In most respects, `DO` is shorthand for `SELECT expr, ...`, but has the advantage that it is slightly faster when you do not care about the result.

`DO` is useful primarily with functions that have side effects, such as `RELEASE_LOCK()`.

Example: This `SELECT` statement pauses, but also produces a result set:

```
mysql> SELECT SLEEP(5);
+----------+
| SLEEP(5) |
+----------+
|        0 |
+----------+
1 row in set (5.02 sec)
```

`DO`, on the other hand, pauses without producing a result set.:

```
mysql> DO SLEEP(5);
Query OK, 0 rows affected (4.99 sec)
```

This could be useful, for example in a stored function or trigger, which prohibit statements that produce result sets.

`DO` only executes expressions. It cannot be used in all cases where `SELECT` can be used. For example, `DO id FROM t1` is invalid because it references a table.

13.2.4 HANDLER Syntax

```
HANDLER tbl_name OPEN [ [AS] alias]

HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]

HANDLER tbl_name CLOSE
```

The `HANDLER` statement provides direct access to table storage engine interfaces. It is available for `InnoDB` and `MyISAM` tables.

The `HANDLER ... OPEN` statement opens a table, making it accessible using subsequent `HANDLER ... READ` statements. This table object is not shared by other sessions and is not closed until the session calls `HANDLER ... CLOSE` or the session terminates.

If you open the table using an alias, further references to the open table with other `HANDLER` statements must use the alias rather than the table name. If you do not use an alias, but open the table using a table name qualified by the database name, further references must use the unqualified table name. For example, for a table opened using `mydb.mytable`, further references must use `mytable`.

The first `HANDLER ... READ` syntax fetches a row where the index specified satisfies the given values and the `WHERE` condition is met. If you have a multiple-column index, specify the index column values as a comma-separated list. Either specify values for all the columns in the index, or specify values for a leftmost prefix of the index columns. Suppose that an index `my_idx` includes three columns named `col_a`, `col_b`, and `col_c`, in that order. The `HANDLER` statement can specify values for all three columns in the index, or for the columns in a leftmost prefix. For example:

```
HANDLER ... READ my_idx = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... READ my_idx = (col_a_val,col_b_val) ...
HANDLER ... READ my_idx = (col_a_val) ...
```

To employ the HANDLER interface to refer to a table's PRIMARY KEY, use the quoted identifier `PRIMARY`:

```
HANDLER tbl_name READ `PRIMARY` ...
```

The second HANDLER ... READ syntax fetches a row from the table in index order that matches the WHERE condition.

The third HANDLER ... READ syntax fetches a row from the table in natural row order that matches the WHERE condition. It is faster than HANDLER *tbl_name* READ *index_name* when a full table scan is desired. Natural row order is the order in which rows are stored in a MyISAM table data file. This statement works for InnoDB tables as well, but there is no such concept because there is no separate data file.

Without a LIMIT clause, all forms of HANDLER ... READ fetch a single row if one is available. To return a specific number of rows, include a LIMIT clause. It has the same syntax as for the SELECT statement. See Section 13.2.10, “SELECT Syntax”.

HANDLER ... CLOSE closes a table that was opened with HANDLER ... OPEN.

There are several reasons to use the HANDLER interface instead of normal SELECT statements:

- HANDLER is faster than SELECT:
  - A designated storage engine handler object is allocated for the HANDLER ... OPEN. The object is reused for subsequent HANDLER statements for that table; it need not be reinitialized for each one.
  - There is less parsing involved.
  - There is no optimizer or query-checking overhead.
  - The handler interface does not have to provide a consistent look of the data (for example, dirty reads are permitted), so the storage engine can use optimizations that SELECT does not normally permit.
- HANDLER makes it easier to port to MySQL applications that use a low-level ISAM-like interface. (See Section 15.19, “InnoDB memcached Plugin” for an alternative way to adapt applications that use the key-value store paradigm.)
- HANDLER enables you to traverse a database in a manner that is difficult (or even impossible) to accomplish with SELECT. The HANDLER interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database.

HANDLER is a somewhat low-level statement. For example, it does not provide consistency. That is, HANDLER ... OPEN does *not* take a snapshot of the table, and does *not* lock the table. This means that after a HANDLER ... OPEN statement is issued, table data can be modified (by the current session or other sessions) and these modifications might be only partially visible to HANDLER ... NEXT or HANDLER ... PREV scans.

An open handler can be closed and marked for reopen, in which case the handler loses its position in the table. This occurs when both of the following circumstances are true:

- Any session executes FLUSH TABLES or DDL statements on the handler's table.
- The session in which the handler is open executes non-HANDLER statements that use tables.

`TRUNCATE TABLE` for a table closes all handlers for the table that were opened with `HANDLER OPEN`.

If a table is flushed with `FLUSH TABLES tbl_name WITH READ LOCK` was opened with `HANDLER`, the handler is implicitly flushed and loses its position.

13.2.5 IMPORT TABLE Syntax

```
IMPORT TABLE FROM sdi_file [, sdi_file] ...
```

The `IMPORT TABLE` statement imports `MyISAM` tables based on information contained in `.sdi` (serialized dictionary information) metadata files. `IMPORT TABLE` requires the `FILE` privilege to read the `.sdi` and table content files, and the `CREATE` privilege for the table to be created.

Tables can be exported from one server using `mysqldump` to write a file of SQL statements and imported into another server using `mysql` to process the dump file. `IMPORT TABLE` provides a faster alternative using the “raw” table files.

Prior to import, the files that provide the table content must be placed in the appropriate schema directory for the import server, and the `.sdi` file must be located in a directory accessible to the server. For example, the `.sdi` file can be placed in the directory named by the `secure_file_priv` system variable, or (if `secure_file_priv` is empty) in a directory under the server data directory.

The following example describes how to export `MyISAM` tables named `employees` and `managers` from the `hr` schema of one server and import them into the `hr` schema of another server. The example uses these assumptions (to perform a similar operation on your own system, modify the path names as appropriate):

- For the export server, *`export_basedir`* represents its base directory, and its data directory is *`export_basedir`*`/data`.
- For the import server, *`import_basedir`* represents its base directory, and its data directory is *`import_basedir`*`/data`.
- Table files are exported from the export server into the `/tmp/export` directory and this directory is secure (not accessible to other users).
- The import server uses `/tmp/mysql-files` as the directory named by its `secure_file_priv` system variable.

To export tables from the export server, use this procedure:

1. Ensure a consistent snapshot by executing this statement to lock the tables so that they cannot be modified during export:

   ```
   mysql> FLUSH TABLES hr.employees, hr.managers WITH READ LOCK;
   ```

   While the lock is in effect, the tables can still be used, but only for read access.

2. At the file system level, copy the `.sdi` and table content files from the `hr` schema directory to the secure export directory:

   - The `.sdi` file is located in the `hr` schema directory, but might not have exactly the same basename as the table name. For example, the `.sdi` files for the `employees` and `managers` tables might be named `employees_125.sdi` and `managers_238.sdi`.
   - For a `MyISAM` table, the content files are its `.MYD` data file and `.MYI` index file.

Given those file names, the copy commands look like this:

```
shell> cd export_basedir/data/hr
shell> cp employees_125.sdi /tmp/export
shell> cp managers_238.sdi /tmp/export
shell> cp employees.{MYD,MYI} /tmp/export
shell> cp managers.{MYD,MYI} /tmp/export
```

3. Unlock the tables:

```
mysql> UNLOCK TABLES;
```

To import tables into the import server, use this procedure:

1. The import schema must exist. If necessary, execute this statement to create it:

```
mysql> CREATE SCHEMA hr;
```

2. At the file system level, copy the `.sdi` files to the import server `secure_file_priv` directory, `/tmp/mysql-files`. Also, copy the table content files to the `hr` schema directory:

```
shell> cd /tmp/export
shell> cp employees_125.sdi /tmp/mysql-files
shell> cp managers_238.sdi /tmp/mysql-files
shell> cp employees.{MYD,MYI} import_basedir/data/hr
shell> cp managers.{MYD,MYI} import_basedir/data/hr
```

3. Import the tables by executing an `IMPORT TABLE` statement that names the `.sdi` files:

```
mysql> IMPORT TABLE FROM
       '/tmp/mysql-files/employees.sdi',
       '/tmp/mysql-files/managers.sdi';
```

The `.sdi` file need not be placed in the import server directory named by the `secure_file_priv` system variable if that variable is empty; it can be in any directory accessible to the server, including the schema directory for the imported table. If the `.sdi` file is placed in that directory, however, it may be rewritten; the import operation creates a new `.sdi` file for the table, which will overwrite the old `.sdi` file if the operation uses the same file name for the new file.

Each *`sdi_file`* value must be a string literal that names the `.sdi` file for a table or is a pattern that matches `.sdi` files. If the string is a pattern, any leading directory path and the `.sdi` file name suffix must be given literally. Pattern characters are permitted only in the base name part of the file name:

- `?` matches any single character
- `*` matches any sequence of characters, including no characters

Using a pattern, the previous `IMPORT TABLE` statement could have been written like this (assuming that the `/tmp/mysql-files` directory contains no other `.sdi` files matching the pattern):

```
IMPORT TABLE FROM '/tmp/mysql-files/*.sdi';
```

To interpret the location of `.sdi` file path names, the server uses the same rules for `IMPORT TABLE` as the server-side rules for `LOAD DATA` (that is, the non-`LOCAL` rules). See Section 13.2.7, “LOAD DATA Syntax”, paying particular attention to the rules used to interpret relative path names.

`IMPORT TABLE` fails if the `.sdi` or table files cannot be located. After importing a table, the server attempts to open it and reports as warnings any problems detected. To attempt a repair to correct any reported issues, use `REPAIR TABLE`.

`IMPORT TABLE` is not written to the binary log.

Restrictions and Limitations

`IMPORT TABLE` applies only to non-`TEMPORARY` `MyISAM` tables. It does not apply to tables created with a transactional storage engine, tables created with `CREATE TEMPORARY TABLE`, or views.

The table data and index files must be placed in the schema directory for the import server prior to the import operation, unless the table as defined on the export server uses the `DATA DIRECTORY` or `INDEX DIRECTORY` table options. In that case, modify the import procedure using one of these alternatives before executing the `IMPORT TABLE` statement:

- Put the data and index files into the same directory on the import server host as on the export server host, and create symlinks in the import server schema directory to those files.
- Put the data and index files into an import server host directory different from that on the export server host, and create symlinks in the import server schema directory to those files. In addition, modify the `.sdi` file to reflect the different file locations.
- Put the data and index files into the schema directory on the import server host, and modify the `.sdi` file to remove the data and index directory table options.

Any collation IDs stored in the `.sdi` file must refer to the same collations on the export and import servers.

Trigger information for a table is not serialized into the table `.sdi` file, so triggers are not restored by the import operation.

Some edits to an `.sdi` file are permissible prior to executing the `IMPORT TABLE` statement, whereas others are problematic or may even cause the import operation to fail:

- Changing the data directory and index directory table options is required if the locations of the data and index files differ between the export and import servers.
- Changing the schema name is required to import the table into a different schema on the import server than on the export server.
- Changing schema and table names may be required to accommodate differences between file system case-sensitivity semantics on the export and import servers or differences in `lower_case_table_names` settings. Changing the table names in the `.sdi` file may require renaming the table files as well.
- In some cases, changes to column definitions are permitted. Changing data types is likely to cause problems.

13.2.6 INSERT Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    {VALUES | VALUE} (value_list) [, (value_list)] ...
    [ON DUPLICATE KEY UPDATE assignment_list]
```

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    SET assignment_list
    [ON DUPLICATE KEY UPDATE assignment_list]

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    SELECT ...
    [ON DUPLICATE KEY UPDATE assignment_list]

value:
    {expr | DEFAULT}

value_list:
    value [, value] ...

assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

`INSERT` inserts new rows into an existing table. The `INSERT ... VALUES` and `INSERT ... SET` forms of the statement insert rows based on explicitly specified values. The `INSERT ... SELECT` form inserts rows selected from another table or tables. `INSERT` with an `ON DUPLICATE KEY UPDATE` clause enables existing rows to be updated if a row to be inserted would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`.

For additional information about `INSERT ... SELECT` and `INSERT ... ON DUPLICATE KEY UPDATE`, see Section 13.2.6.1, “INSERT ... SELECT Syntax”, and Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”.

In MySQL 8.0, the `DELAYED` keyword is accepted but ignored by the server. For the reasons for this, see Section 13.2.6.3, “INSERT DELAYED Syntax”,

Inserting into a table requires the `INSERT` privilege for the table. If the `ON DUPLICATE KEY UPDATE` clause is used and a duplicate key causes an `UPDATE` to be performed instead, the statement requires the `UPDATE` privilege for the columns to be updated. For columns that are read but not modified you need only the `SELECT` privilege (such as for a column referenced only on the right hand side of an *`col_name`*=*`expr`* assignment in an `ON DUPLICATE KEY UPDATE` clause).

When inserting into a partitioned table, you can control which partitions and subpartitions accept new rows. The `PARTITION` option takes a list of the comma-separated names of one or more partitions or subpartitions (or both) of the table. If any of the rows to be inserted by a given `INSERT` statement do not match one of the partitions listed, the `INSERT` statement fails with the error `Found a row not matching the given partition set`. For more information and examples, see Section 23.5, “Partition Selection”.

You can use `REPLACE` instead of `INSERT` to overwrite old rows. `REPLACE` is the counterpart to `INSERT IGNORE` in the treatment of new rows that contain unique key values that duplicate old rows: The new rows replace the old rows rather than being discarded. See Section 13.2.9, “REPLACE Syntax”.

`tbl_name` is the table into which rows should be inserted. Specify the columns for which the statement provides values as follows:

- Provide a parenthesized list of comma-separated column names following the table name. In this case, a value for each named column must be provided by the `VALUES` list or the `SELECT` statement.

- If you do not specify a list of column names for `INSERT ... VALUES` or `INSERT ... SELECT`, values for every column in the table must be provided by the `VALUES` list or the `SELECT` statement. If you do not know the order of the columns in the table, use `DESCRIBE tbl_name` to find out.
- A `SET` clause indicates columns explicitly by name, together with the value to assign each one.

Column values can be given in several ways:

- If strict SQL mode is not enabled, any column not explicitly given a value is set to its default (explicit or implicit) value. For example, if you specify a column list that does not name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in Section 11.7, “Data Type Default Values”. See also Section 1.8.3.3, “Constraints on Invalid Data”.

  If strict SQL mode is enabled, an `INSERT` statement generates an error if it does not specify an explicit value for every column that has no default value. See Section 5.1.11, “Server SQL Modes”.

- If both the column list and the `VALUES` list are empty, `INSERT` creates a row with each column set to its default value:

  ```
  INSERT INTO tbl_name () VALUES();
  ```

  If strict mode is not enabled, MySQL uses the implicit default value for any column that has no explicitly defined default. If strict mode is enabled, an error occurs if any column has no default value.

- Use the keyword `DEFAULT` to set a column explicitly to its default value. This makes it easier to write `INSERT` statements that assign values to all but a few columns, because it enables you to avoid writing an incomplete `VALUES` list that does not include a value for each column in the table. Otherwise, you must provide the list of column names corresponding to each value in the `VALUES` list.

- If a generated column is inserted into explicitly, the only permitted value is `DEFAULT`. For information about generated columns, see Section 13.1.20.9, “CREATE TABLE and Generated Columns”.

- In expressions, you can use `DEFAULT(col_name)` to produce the default value for column *`col_name`*.

- Type conversion of an expression *`expr`* that provides a column value might occur if the expression data type does not match the column data type. Conversion of a given value can result in different inserted values depending on the column type. For example, inserting the string `'1999.0e-2'` into an `INT`, `FLOAT`, `DECIMAL(10,6)`, or `YEAR` column inserts the value `1999`, `19.9921`, `19.992100`, or `1999`, respectively. The value stored in the `INT` and `YEAR` columns is `1999` because the string-to-number conversion looks only at as much of the initial part of the string as may be considered a valid integer or year. For the `FLOAT` and `DECIMAL` columns, the string-to-number conversion considers the entire string a valid numeric value.

- An expression *`expr`* can refer to any column that was set earlier in a value list. For example, you can do this because the value for `col2` refers to `col1`, which has previously been assigned:

  ```
  INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
  ```

  But the following is not legal, because the value for `col1` refers to `col2`, which is assigned after `col1`:

  ```
  INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
  ```

  An exception occurs for columns that contain `AUTO_INCREMENT` values. Because `AUTO_INCREMENT` values are generated after other value assignments, any reference to an `AUTO_INCREMENT` column in the assignment returns a `0`.

`INSERT` statements that use `VALUES` syntax can insert multiple rows. To do this, include multiple lists of comma-separated column values, with lists enclosed within parentheses and separated by commas. Example:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

Each values list must contain exactly as many values as are to be inserted per row. The following statement is invalid because it contains one list of nine values, rather than three lists of three values each:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

`VALUE` is a synonym for `VALUES` in this context. Neither implies anything about the number of values lists, nor about the number of values per list. Either may be used whether there is a single values list or multiple lists, and regardless of the number of values per list.

The affected-rows value for an `INSERT` can be obtained using the `ROW_COUNT()` SQL function or the `mysql_affected_rows()` C API function. See Section 12.15, “Information Functions”, and Section 28.7.7.1, “mysql_affected_rows()”.

If you use an `INSERT ... VALUES` statement with multiple value lists or `INSERT ... SELECT`, the statement returns an information string in this format:

```
Records: N1 Duplicates: N2 Warnings: N3
```

If you are using the C API, the information string can be obtained by invoking the `mysql_info()` function. See Section 28.7.7.36, “mysql_info()”.

`Records` indicates the number of rows processed by the statement. (This is not necessarily the number of rows actually inserted because `Duplicates` can be nonzero.) `Duplicates` indicates the number of rows that could not be inserted because they would duplicate some existing unique index value. `Warnings` indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting `NULL` into a column that has been declared `NOT NULL`. For multiple-row `INSERT` statements or `INSERT INTO ... SELECT` statements, the column is set to the implicit default value for the column data type. This is `0` for numeric types, the empty string (`''`) for string types, and the “zero” value for date and time types. `INSERT INTO ... SELECT` statements are handled the same way as multiple-row inserts because the server does not examine the result set from the `SELECT` to see whether it returns a single row. (For a single-row `INSERT`, no warning occurs when `NULL` is inserted into a `NOT NULL` column. Instead, the statement fails with an error.)
- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the closest endpoint of the range.
- Assigning a value such as `'10.34 a'` to a numeric column. The trailing nonnumeric text is stripped off and the remaining numeric part is inserted. If the string value has no leading numeric part, the column is set to `0`.
- Inserting a string into a string column (`CHAR`, `VARCHAR`, `TEXT`, or `BLOB`) that exceeds the column's maximum length. The value is truncated to the column's maximum length.
- Inserting a value into a date or time column that is illegal for the data type. The column is set to the appropriate zero value for the type.

- For `INSERT` examples involving `AUTO_INCREMET` column values, see Section 3.6.9, “Using AUTO_INCREMENT”.

  If `INSERT` inserts a row into a table that has an `AUTO_INCREMENT` column, you can find the value used for that column by using the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function.

  Note

  These two functions do not always behave identically. The behavior of `INSERT` statements with respect to `AUTO_INCREMENT` columns is discussed further in Section 12.15, “Information Functions”, and Section 28.7.7.38, “mysql_insert_id()”.

The `INSERT` statement supports the following modifiers:

- If you use the `LOW_PRIORITY` modifier, execution of the `INSERT` is delayed until no other clients are reading from the table. This includes other clients that began reading while existing clients are reading, and while the `INSERT LOW_PRIORITY` statement is waiting. It is possible, therefore, for a client that issues an `INSERT LOW_PRIORITY` statement to wait for a very long time.

  `LOW_PRIORITY` affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

  Note

  `LOW_PRIORITY` should normally not be used with `MyISAM` tables because doing so disables concurrent inserts. See Section 8.11.3, “Concurrent Inserts”.

- If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` option if the server was started with that option. It also causes concurrent inserts not to be used. See Section 8.11.3, “Concurrent Inserts”.

  `HIGH_PRIORITY` affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- If you use the `IGNORE` modifier, errors that occur while executing the `INSERT` statement are ignored. For example, without `IGNORE`, a row that duplicates an existing `UNIQUE` index or `PRIMARY KEY` value in the table causes a duplicate-key error and the statement is aborted. With `IGNORE`, the row is discarded and no error occurs. Ignored errors generate warnings instead.

  `IGNORE` has a similar effect on inserts into partitioned tables where no partition matching a given value is found. Without `IGNORE`, such `INSERT` statements are aborted with an error. When `INSERT IGNORE` is used, the insert operation fails silently for rows containing the unmatched value, but inserts rows that are matched. For an example, see Section 23.2.2, “LIST Partitioning”.

  Data conversions that would trigger errors abort the statement if `IGNORE` is not specified. With `IGNORE`, invalid values are adjusted to the closest values and inserted; warnings are produced but the statement does not abort. You can determine with the `mysql_info()` C API function how many rows were actually inserted into the table.

  For more information, see Comparison of the IGNORE Keyword and Strict SQL Mode.

- If you specify `ON DUPLICATE KEY UPDATE`, and a row is inserted that would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row occurs. The affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the `CLIENT_FOUND_ROWS` flag to the `mysql_real_connect()` C API

function when connecting to `mysqld`, the affected-rows value is 1 (not 0) if an existing row is set to its current values. See Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”.

- `INSERT DELAYED` was deprecated in MySQL 5.6, and is scheduled for eventual removal. In MySQL 8.0, the `DELAYED` modifier is accepted but ignored. Use `INSERT` (without `DELAYED`) instead. See Section 13.2.6.3, “INSERT DELAYED Syntax”.

An `INSERT` statement affecting a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks only those partitions into which rows are actually inserted. (For storage engines such as `InnoDB` that employ row-level locking, no locking of partitions takes place.) For more information, see Partitioning and Locking.

13.2.6.1 INSERT ... SELECT Syntax

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    SELECT ...
    [ON DUPLICATE KEY UPDATE assignment_list]

value:
    {expr | DEFAULT}

assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

With `INSERT ... SELECT`, you can quickly insert many rows into a table from the result of a `SELECT` statement, which can select from one or many tables. For example:

```
INSERT INTO tbl_temp2 (fld_id)
  SELECT tbl_temp1.fld_order_id
  FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

The following conditions hold for `INSERT ... SELECT` statements:

- Specify `IGNORE` to ignore rows that would cause duplicate-key violations.
- The target table of the `INSERT` statement may appear in the `FROM` clause of the `SELECT` part of the query. However, you cannot insert into a table and select from the same table in a subquery.

  When selecting from and inserting into the same table, MySQL creates an internal temporary table to hold the rows from the `SELECT` and then inserts those rows into the target table. However, you cannot use `INSERT INTO t ... SELECT ... FROM t` when `t` is a `TEMPORARY` table, because `TEMPORARY` tables cannot be referred to twice in the same statement. See Section 8.4.4, “Internal Temporary Table Use in MySQL”, and Section B.4.6.2, “TEMPORARY Table Problems”.
- `AUTO_INCREMENT` columns work as usual.
- To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts for `INSERT ... SELECT` statements (see Section 8.11.3, “Concurrent Inserts”).
- To avoid ambiguous column reference problems when the `SELECT` and the `INSERT` refer to the same table, provide a unique alias for each table used in the `SELECT` part, and qualify column names in that part with the appropriate alias.

You can explicitly select which partitions or subpartitions (or both) of the source or target table (or both) are to be used with a `PARTITION` option following the name of the table. When `PARTITION` is used with the name of the source table in the `SELECT` portion of the statement, rows are selected only from the partitions or subpartitions named in its partition list. When `PARTITION` is used with the name of the target table for the `INSERT` portion of the statement, it must be possible to insert all rows selected into the partitions or subpartitions named in the partition list following the option. Otherwise, the `INSERT ... SELECT` statement fails. For more information and examples, see Section 23.5, “Partition Selection”.

For `INSERT ... SELECT` statements, see Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax” for conditions under which the `SELECT` columns can be referred to in an `ON DUPLICATE KEY UPDATE` clause.

The order in which a `SELECT` statement with no `ORDER BY` clause returns rows is nondeterministic. This means that, when using replication, there is no guarantee that such a `SELECT` returns rows in the same order on the master and the slave, which can lead to inconsistencies between them. To prevent this from occurring, always write `INSERT ... SELECT` statements that are to be replicated using an `ORDER BY` clause that produces the same row order on the master and the slave. See also Section 17.5.1.18, “Replication and LIMIT”.

Due to this issue, `INSERT ... SELECT ON DUPLICATE KEY UPDATE` and `INSERT IGNORE ... SELECT` statements are flagged as unsafe for statement-based replication. Such statements produce a warning in the error log when using statement-based mode and are written to the binary log using the row-based format when using `MIXED` mode. (Bug #11758262, Bug #50439)

See also Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

An `INSERT ... SELECT` statement affecting partitioned tables using a storage engine such as `MyISAM` that employs table-level locks locks all partitions of the target table; however, only those partitions that are actually read from the source table are locked. (This does not occur with tables using storage engines such as `InnoDB` that employ row-level locking.) For more information, see Partitioning and Locking.

13.2.6.2 INSERT ... ON DUPLICATE KEY UPDATE Syntax

If you specify an `ON DUPLICATE KEY UPDATE` clause and a row to be inserted would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row occurs. For example, if column `a` is declared as `UNIQUE` and contains the value `1`, the following two statements have similar effect:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=c+1;

UPDATE t1 SET c=c+1 WHERE a=1;
```

(The effects are not identical for an `InnoDB` table where `a` is an auto-increment column. With an auto-increment column, an `INSERT` statement increases the auto-increment value but `UPDATE` does not.)

If column `b` is also unique, the `INSERT` is equivalent to this `UPDATE` statement instead:

```
UPDATE t1 SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

If `a=1 OR b=2` matches several rows, only *one* row is updated. In general, you should try to avoid using an `ON DUPLICATE KEY UPDATE` clause on tables with multiple unique indexes.

With `ON DUPLICATE KEY UPDATE`, the affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the

CLIENT_FOUND_ROWS flag to the mysql_real_connect() C API function when connecting to mysqld, the affected-rows value is 1 (not 0) if an existing row is set to its current values.

If a table contains an AUTO_INCREMENT column and INSERT ... ON DUPLICATE KEY UPDATE inserts or updates a row, the LAST_INSERT_ID() function returns the AUTO_INCREMENT value.

The ON DUPLICATE KEY UPDATE clause can contain multiple column assignments, separated by commas.

In assignment value expressions in the ON DUPLICATE KEY UPDATE clause, you can use the VALUES(*col_name*) function to refer to column values from the INSERT portion of the INSERT ... ON DUPLICATE KEY UPDATE statement. In other words, VALUES(*col_name*) in the ON DUPLICATE KEY UPDATE clause refers to the value of *col_name* that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The VALUES() function is meaningful only in the ON DUPLICATE KEY UPDATE clause or INSERT statements and returns NULL otherwise. Example:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3),(4,5,6)
  ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

That statement is identical to the following two statements:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=3;
INSERT INTO t1 (a,b,c) VALUES (4,5,6)
  ON DUPLICATE KEY UPDATE c=9;
```

For INSERT ... SELECT statements, these rules apply regarding acceptable forms of SELECT query expressions that you can refer to in an ON DUPLICATE KEY UPDATE clause:

- References to columns from queries on a single table, which may be a derived table.
- References to columns from queries on a join over multiple tables.
- References to columns from DISTINCT queries.
- References to columns in other tables, as long as the SELECT does not use GROUP BY. One side effect is that you must qualify references to nonunique column names.

References to columns from a UNION are not supported. To work around this restriction, rewrite the UNION as a derived table so that its rows can be treated as a single-table result set. For example, this statement produces an error:

```
INSERT INTO t1 (a, b)
  SELECT c, d FROM t2
  UNION
  SELECT e, f FROM t3
ON DUPLICATE KEY UPDATE b = b + c;
```

Instead, use an equivalent statement that rewrites the UNION as a derived table:

```
INSERT INTO t1 (a, b)
SELECT * FROM
  (SELECT c, d FROM t2
   UNION
   SELECT e, f FROM t3) AS dt
```

```
ON DUPLICATE KEY UPDATE b = b + c;
```

The technique of rewriting a query as a derived table also enables references to columns from `GROUP BY` queries.

Because the results of `INSERT ... SELECT` statements depend on the ordering of rows from the `SELECT` and this order cannot always be guaranteed, it is possible when logging `INSERT ... SELECT ON DUPLICATE KEY UPDATE` statements for the master and the slave to diverge. Thus, `INSERT ... SELECT ON DUPLICATE KEY UPDATE` statements are flagged as unsafe for statement-based replication. Such statements produce a warning in the error log when using statement-based mode and are written to the binary log using the row-based format when using `MIXED` mode. An `INSERT ... ON DUPLICATE KEY UPDATE` statement against a table having more than one unique or primary key is also marked as unsafe. (Bug #11765650, Bug #58637)

See also Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

An `INSERT ... ON DUPLICATE KEY UPDATE` on a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks any partitions of the table in which a partitioning key column is updated. (This does not occur with tables using storage engines such as `InnoDB` that employ row-level locking.) For more information, see Partitioning and Locking.

13.2.6.3 INSERT DELAYED Syntax

```
INSERT DELAYED ...
```

The `DELAYED` option for the `INSERT` statement is a MySQL extension to standard SQL. In previous versions of MySQL, it can be used for certain kinds of tables (such as `MyISAM`), such that when a client uses `INSERT DELAYED`, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

`DELAYED` inserts and replaces were deprecated in MySQL 5.6. In MySQL 8.0, `DELAYED` is not supported. The server recognizes but ignores the `DELAYED` keyword, handles the insert as a nondelayed insert, and generates an `ER_WARN_LEGACY_SYNTAX_CONVERTED` warning (“INSERT DELAYED is no longer supported. The statement was converted to INSERT”). The `DELAYED` keyword is scheduled for removal in a future release.

13.2.7 LOAD DATA Syntax

```
LOAD DATA
    [LOW_PRIORITY | CONCURRENT] [LOCAL]
    INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var
        [, col_name_or_user_var] ...)]
```

```
    [SET col_name={expr | DEFAULT},
        [, col_name={expr | DEFAULT}] ...]
```

The `LOAD DATA` statement reads rows from a text file into a table at a very high speed. `LOAD DATA` is the complement of `SELECT ... INTO OUTFILE`. (See Section 13.2.10.1, “SELECT ... INTO Syntax”.) To write data from a table to a file, use `SELECT ... INTO OUTFILE`. To read the file back into a table, use `LOAD DATA`. The syntax of the `FIELDS` and `LINES` clauses is the same for both statements.

You can also load data files by using the `mysqlimport` utility; see Section 4.5.5, “`mysqlimport` — A Data Import Program”. `mysqlimport` operates by sending a `LOAD DATA` statement to the server.

For more information about the efficiency of `INSERT` versus `LOAD DATA` and speeding up `LOAD DATA`, see Section 8.2.5.1, “Optimizing INSERT Statements”.

- Partitioned Table Support
- Input File Name, Location, and Content Interpretation
- Replication Considerations
- Concurrency Considerations
- Duplicate-Key Handling
- Index Handling
- Field and Line Handling
- Column List Specification
- Input Preprocessing
- Statement Result Information
- Miscellaneous Topics

Partitioned Table Support

`LOAD DATA` supports explicit partition selection using the `PARTITION` option with a list of one or more comma-separated names of partitions, subpartitions, or both. When this option is used, if any rows from the file cannot be inserted into any of the partitions or subpartitions named in the list, the statement fails with the error `Found a row not matching the given partition set`. For more information and examples, see Section 23.5, “Partition Selection”.

For partitioned tables using storage engines that employ table locks, such as `MyISAM`, `LOAD DATA` cannot prune any partition locks. This does not apply to tables using storage engines that employ row-level locking, such as `InnoDB`. For more information, see Partitioning and Locking.

Input File Name, Location, and Content Interpretation

The file name must be given as a literal string. On Windows, specify backslashes in path names as forward slashes or doubled backslashes. The `character_set_filesystem` system variable controls the interpretation of the file name character set.

The server uses the character set indicated by the `character_set_database` system variable to interpret the information in the file. `SET NAMES` and the setting of `character_set_client` do not affect interpretation of input. If the contents of the input file use a character set that differs from the default, it is

usually preferable to specify the character set of the file by using the `CHARACTER SET` clause. A character set of `binary` specifies “no conversion.”

`LOAD DATA` interprets all fields in the file as having the same character set, regardless of the data types of the columns into which field values are loaded. For proper interpretation of file contents, you must ensure that it was written with the correct character set. For example, if you write a data file with `mysqldump -T` or by issuing a `SELECT ... INTO OUTFILE` statement in `mysql`, be sure to use a `--default-character-set` option so that output is written in the character set to be used when the file is loaded with `LOAD DATA`.

Note

It is not possible to load data files that use the `ucs2`, `utf16`, `utf16le`, or `utf32` character set.

Replication Considerations

`LOAD DATA` is considered unsafe for statement-based replication. If you do use `LOAD DATA` when `binlog_format=STATEMENT` is set, a temporary file containing the data is created on the replication slave where the changes are applied. If binary log encryption is active on the server, note that this temporary file is not encrypted. When encryption is required, be sure to use row-based or mixed binary logging format instead, which do not create the temporary files. For more information on the interaction between `LOAD DATA` and replication, see Section 17.5.1.19, “Replication and LOAD DATA”.

Concurrency Considerations

If you use the `LOW_PRIORITY` modifier, execution of the `LOAD DATA` statement is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

If you specify the `CONCURRENT` modifier with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other threads can retrieve data from the table while `LOAD DATA` is executing. This modifier affects the performance of `LOAD DATA` a bit, even if no other thread is using the table at the same time.

The `LOCAL` modifier affects expected location of the file and error handling, as described later. `LOCAL` works only if your server and your client both have been configured to permit it. For example, if `mysqld` was started with the `local_infile` system variable disabled, `LOCAL` does not work. See Section 6.1.6, “Security Issues with LOAD DATA LOCAL”.

The `LOCAL` modifier affects where the file is expected to be found:

- If `LOCAL` is specified, the file is read by the client program on the client host and sent to the server. The file can be given as a full path name to specify its exact location. If given as a relative path name, the name is interpreted relative to the directory in which the client program was started.

  When using `LOCAL` with `LOAD DATA`, a copy of the file is created in the directory where the MySQL server stores temporary files. See Section B.4.3.5, “Where MySQL Stores Temporary Files”. Lack of sufficient space for the copy in this directory can cause the `LOAD DATA LOCAL` statement to fail.

- If `LOCAL` is not specified, the file must be located on the server host and is read directly by the server. The server uses the following rules to locate the file:

  - If the file name is an absolute path name, the server uses it as given.
  - If the file name is a relative path name with one or more leading components, the server searches for the file relative to the server's data directory.

- If a file name with no leading components is given, the server looks for the file in the database directory of the default database.

In the non-`LOCAL` case, these rules mean that a file named as `./myfile.txt` is read from the server's data directory, whereas the file named as `myfile.txt` is read from the database directory of the default database. For example, if `db1` is the default database, the following `LOAD DATA` statement reads the file `data.txt` from the database directory for `db1`, even though the statement explicitly loads the file into a table in the `db2` database:

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Note

The server also uses the non-`LOCAL` rules to locate `.sdi` files for the `IMPORT TABLE` statement.

Non-`LOCAL` load operations read text files located on the server. For security reasons, such operations require that you have the `FILE` privilege. See Section 6.2.2, “Privileges Provided by MySQL”. Also, non-`LOCAL` load operations are subject to the `secure_file_priv` system variable setting. If the variable value is a nonempty directory name, the file to be loaded must be located in that directory. If the variable value is empty (which is insecure), the file need only be readable by the server.

Using `LOCAL` is a bit slower than letting the server access the files directly, because the file contents must be sent over the connection by the client to the server. On the other hand, you do not need the `FILE` privilege to load local files.

`LOCAL` also affects error handling:

- With `LOAD DATA`, data-interpretation and duplicate-key errors terminate the operation.
- With `LOAD DATA LOCAL`, data-interpretation and duplicate-key errors become warnings and the operation continues because the server has no way to stop transmission of the file in the middle of the operation. For duplicate-key errors, this is the same as if `IGNORE` is specified. `IGNORE` is explained further later in this section.

Duplicate-Key Handling

The `REPLACE` and `IGNORE` modifiers control handling of input rows that duplicate existing rows on unique key values:

- If you specify `REPLACE`, input rows replace existing rows. In other words, rows that have the same value for a primary key or unique index as an existing row. See Section 13.2.9, “REPLACE Syntax”.
- If you specify `IGNORE`, rows that duplicate an existing row on a unique key value are discarded. For more information, see Comparison of the IGNORE Keyword and Strict SQL Mode.
- If you do not specify either modifier, the behavior depends on whether the `LOCAL` modifier is specified. Without `LOCAL`, an error occurs when a duplicate key value is found, and the rest of the text file is ignored. With `LOCAL`, the default behavior is the same as if `IGNORE` is specified; this is because the server has no way to stop transmission of the file in the middle of the operation.

Index Handling

To ignore foreign key constraints during the load operation, execute a `SET foreign_key_checks = 0` statement before executing `LOAD DATA`.

If you use `LOAD DATA` on an empty `MyISAM` table, all nonunique indexes are created in a separate batch (as for `REPAIR TABLE`). Normally, this makes `LOAD DATA` much faster when you have many indexes. In some extreme cases, you can create the indexes even faster by turning them off with `ALTER TABLE ... DISABLE KEYS` before loading the file into the table and using `ALTER TABLE ... ENABLE KEYS` to re-create the indexes after loading the file. See Section 8.2.5.1, “Optimizing INSERT Statements”.

Field and Line Handling

For both the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements, the syntax of the `FIELDS` and `LINES` clauses is the same. Both clauses are optional, but `FIELDS` must precede `LINES` if both are specified.

If you specify a `FIELDS` clause, each of its subclauses (`TERMINATED BY`, `[OPTIONALLY] ENCLOSED BY`, and `ESCAPED BY`) is also optional, except that you must specify at least one of them. Arguments to these clauses are permitted to contain only ASCII characters.

If you specify no `FIELDS` or `LINES` clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
LINES TERMINATED BY '\n' STARTING BY ''
```

Backslash is the MySQL escape character within strings in SQL statements. Thus, to specify a literal backslash, you must specify two backslashes for the value to be interpreted as a single backslash. The escape sequences `'\t'` and `'\n'` specify tab and newline characters, respectively.

In other words, the defaults cause `LOAD DATA` to act as follows when reading input:

- Look for line boundaries at newlines.
- Do not skip any line prefix.
- Break lines into fields at tabs.
- Do not expect fields to be enclosed within any quoting characters.
- Interpret characters preceded by the escape character `\` as escape sequences. For example, `\t`, `\n`, and `\\` signify tab, newline, and backslash, respectively. See the discussion of `FIELDS ESCAPED BY` later for the full list of escape sequences.

Conversely, the defaults cause `SELECT ... INTO OUTFILE` to act as follows when writing output:

- Write tabs between fields.
- Do not enclose fields within any quoting characters.
- Use `\` to escape instances of tab, newline, or `\` that occur within field values.
- Write newlines at the ends of lines.

Note

For a text file generated on a Windows system, proper file reading might require `LINES TERMINATED BY '\r\n'` because Windows programs typically use two characters as a line terminator. Some programs, such as `WordPad`, might use `\r` as a line terminator when writing files. To read such files, use `LINES TERMINATED BY '\r'`.

If all the input lines have a common prefix that you want to ignore, you can use `LINES STARTING BY 'prefix_string'` to skip the prefix *and anything before it*. If a line does not include the prefix, the entire line is skipped. Suppose that you issue the following statement:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
  FIELDS TERMINATED BY ','  LINES STARTING BY 'xxx';
```

If the data file looks like this:

```
xxx"abc",1
something xxx"def",2
"ghi",3
```

The resulting rows will be `("abc",1)` and `("def",2)`. The third row in the file is skipped because it does not contain the prefix.

The `IGNORE number LINES` option can be used to ignore lines at the start of the file. For example, you can use `IGNORE 1 LINES` to skip an initial header line containing column names:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;
```

When you use `SELECT ... INTO OUTFILE` in tandem with `LOAD DATA` to write data from a database into a file and then read the file back into the database later, the field- and line-handling options for both statements must match. Otherwise, `LOAD DATA` will not interpret the contents of the file properly. Suppose that you use `SELECT ... INTO OUTFILE` to write a file with fields delimited by commas:

```
SELECT * INTO OUTFILE 'data.txt'
  FIELDS TERMINATED BY ','
  FROM table2;
```

To read the comma-delimited file, the correct statement would be:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY ',';
```

If instead you tried to read the file with the statement shown following, it wouldn't work because it instructs `LOAD DATA` to look for tabs between fields:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

`LOAD DATA` can be used to read files obtained from external sources. For example, many programs can export data in comma-separated values (CSV) format, such that lines have fields separated by commas and enclosed within double quotation marks, with an initial line of column names. If the lines in such a file are terminated by carriage return/newline pairs, the statement shown here illustrates the field- and line-handling options you would use to load the file:

```
LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;
```

If the input values are not necessarily enclosed within quotation marks, use `OPTIONALLY` before the `ENCLOSED BY` option.

Any of the field- or line-handling options can specify an empty string (''). If not empty, the FIELDS [OPTIONALLY] ENCLOSED BY and FIELDS ESCAPED BY values must be a single character. The FIELDS TERMINATED BY, LINES STARTING BY, and LINES TERMINATED BY values can be more than one character. For example, to write lines that are terminated by carriage return/linefeed pairs, or to read a file containing such lines, specify a LINES TERMINATED BY '\r\n' clause.

To read a file containing jokes that are separated by lines consisting of %%, you can do this

```
CREATE TABLE jokes
  (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joke TEXT NOT NULL);
LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
  FIELDS TERMINATED BY ''
  LINES TERMINATED BY '\n%%\n' (joke);
```

FIELDS [OPTIONALLY] ENCLOSED BY controls quoting of fields. For output (SELECT ... INTO OUTFILE), if you omit the word OPTIONALLY, all fields are enclosed by the ENCLOSED BY character. An example of such output (using a comma as the field delimiter) is shown here:

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

If you specify OPTIONALLY, the ENCLOSED BY character is used only to enclose values from columns that have a string data type (such as CHAR, BINARY, TEXT, or ENUM):

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Occurrences of the ENCLOSED BY character within a field value are escaped by prefixing them with the ESCAPED BY character. Also, if you specify an empty ESCAPED BY value, it is possible to inadvertently generate output that cannot be read properly by LOAD DATA. For example, the preceding output just shown would appear as follows if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

For input, the ENCLOSED BY character, if present, is stripped from the ends of field values. (This is true regardless of whether OPTIONALLY is specified; OPTIONALLY has no effect on input interpretation.) Occurrences of the ENCLOSED BY character preceded by the ESCAPED BY character are interpreted as part of the current field value.

If the field begins with the ENCLOSED BY character, instances of that character are recognized as terminating a field value only if followed by the field or line TERMINATED BY sequence. To avoid ambiguity, occurrences of the ENCLOSED BY character within a field value can be doubled and are interpreted as a single instance of the character. For example, if ENCLOSED BY '"' is specified, quotation marks are handled as shown here:

```
"The ""BIG"" boss"  -> The "BIG" boss
```

```
The "BIG" boss      -> The "BIG" boss
The ""BIG"" boss    -> The ""BIG"" boss
```

`FIELDS ESCAPED BY` controls how to read or write special characters:

- For input, if the `FIELDS ESCAPED BY` character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. Some two-character sequences that are exceptions, where the first character is the escape character. These sequences are shown in the following table (using `\` for the escape character). The rules for `NULL` handling are described later in this section.

Character	Escape Sequence
`\0`	An ASCII NUL (`X'00'`) character
`\b`	A backspace character
`\n`	A newline (linefeed) character
`\r`	A carriage return character
`\t`	A tab character.
`\Z`	ASCII 26 (Control+Z)
`\N`	NULL

  For more information about `\`-escape syntax, see Section 9.1.1, “String Literals”.

  If the `FIELDS ESCAPED BY` character is empty, escape-sequence interpretation does not occur.

- For output, if the `FIELDS ESCAPED BY` character is not empty, it is used to prefix the following characters on output:

  - The `FIELDS ESCAPED BY` character.
  - The `FIELDS [OPTIONALLY] ENCLOSED BY` character.
  - The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values, if the `ENCLOSED BY` character is empty or unspecified.
  - ASCII `0` (what is actually written following the escape character is ASCII `0`, not a zero-valued byte).

  If the `FIELDS ESCAPED BY` character is empty, no characters are escaped and `NULL` is output as `NULL`, not `\N`. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

In certain cases, field- and line-handling options interact:

- If `LINES TERMINATED BY` is an empty string and `FIELDS TERMINATED BY` is nonempty, lines are also terminated with `FIELDS TERMINATED BY`.
- If the `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` values are both empty (`''`), a fixed-row (nondelimited) format is used. With fixed-row format, no delimiters are used between fields (but you can still have a line terminator). Instead, column values are read and written using a field width wide enough to hold all values in the field. For `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, and `BIGINT`, the field widths are 4, 6, 8, 11, and 20, respectively, no matter what the declared display width is.

  `LINES TERMINATED BY` is still used to separate lines. If a line does not contain all fields, the rest of the columns are set to their default values. If you do not have a line terminator, you should set this to `''`. In this case, the text file must contain all fields for each row.

Fixed-row format also affects handling of `NULL` values, as described later.

Note

Fixed-size format does not work if you are using a multibyte character set.

Handling of `NULL` values varies according to the `FIELDS` and `LINES` options in use:

- For the default `FIELDS` and `LINES` values, `NULL` is written as a field value of `\N` for output, and a field value of `\N` is read as `NULL` for input (assuming that the `ESCAPED BY` character is `\`).
- If `FIELDS ENCLOSED BY` is not empty, a field containing the literal word `NULL` as its value is read as a `NULL` value. This differs from the word `NULL` enclosed within `FIELDS ENCLOSED BY` characters, which is read as the string `'NULL'`.
- If `FIELDS ESCAPED BY` is empty, `NULL` is written as the word `NULL`.
- With fixed-row format (which is used when `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` are both empty), `NULL` is written as an empty string. This causes both `NULL` values and empty strings in the table to be indistinguishable when written to the file because both are written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

An attempt to load `NULL` into a `NOT NULL` column causes assignment of the implicit default value for the column's data type and a warning, or an error in strict SQL mode. Implicit default values are discussed in Section 11.7, “Data Type Default Values”.

Some cases are not supported by `LOAD DATA`:

- Fixed-size rows (`FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` both empty) and `BLOB` or `TEXT` columns.
- If you specify one separator that is the same as or a prefix of another, `LOAD DATA` cannot interpret the input properly. For example, the following `FIELDS` clause would cause problems:

```
FIELDS TERMINATED BY '"' ENCLOSED BY '"'
```

- If `FIELDS ESCAPED BY` is empty, a field value that contains an occurrence of `FIELDS ENCLOSED BY` or `LINES TERMINATED BY` followed by the `FIELDS TERMINATED BY` value causes `LOAD DATA` to stop reading a field or line too early. This happens because `LOAD DATA` cannot properly determine where the field or line value ends.

Column List Specification

The following example loads all columns of the `persondata` table:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

By default, when no column list is provided at the end of the `LOAD DATA` statement, input lines are expected to contain a field for each table column. If you want to load only some of a table's columns, specify a column list:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata
(col_name_or_user_var [, col_name_or_user_var] ...);
```

You must also specify a column list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, MySQL cannot tell how to match input fields with table columns.

Input Preprocessing

Each *`col_name_or_user_var`* value is either a column name or a user variable. With user variables, the `SET` clause enables you to perform preprocessing transformations on their values before assigning the result to columns.

User variables in the `SET` clause can be used in several ways. The following example uses the first input column directly for the value of `t1.column1`, and assigns the second input column to a user variable that is subjected to a division operation before being used for the value of `t1.column2`:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @var1)
  SET column2 = @var1/100;
```

The `SET` clause can be used to supply values not derived from the input file. The following statement sets `column3` to the current date and time:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, column2)
  SET column3 = CURRENT_TIMESTAMP;
```

You can also discard an input value by assigning it to a user variable and not assigning the variable to a table column:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @dummy, column2, @dummy, column3);
```

Use of the column/variable list and `SET` clause is subject to the following restrictions:

- Assignments in the `SET` clause should have only column names on the left hand side of assignment operators.
- You can use subqueries in the right hand side of `SET` assignments. A subquery that returns a value to be assigned to a column may be a scalar subquery only. Also, you cannot use a subquery to select from the table that is being loaded.
- Lines ignored by an `IGNORE` clause are not processed for the column/variable list or `SET` clause.
- User variables cannot be used when loading data with fixed-row format because user variables do not have a display width.

When processing an input line, `LOAD DATA` splits it into fields and uses the values according to the column/variable list and the `SET` clause, if they are present. Then the resulting row is inserted into the table. If there are `BEFORE INSERT` or `AFTER INSERT` triggers for the table, they are activated before or after inserting the row, respectively.

If an input line has too many fields, the extra fields are ignored and the number of warnings is incremented.

If an input line has too few fields, the table columns for which input fields are missing are set to their default values. Default value assignment is described in Section 11.7, “Data Type Default Values”.

An empty field value is interpreted different from a missing field:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to `0`.
- For date and time types, the column is set to the appropriate “zero” value for the type. See Section 11.3, “Date and Time Types”.

These are the same values that result if you assign an empty string explicitly to a string, numeric, or date or time type explicitly in an `INSERT` or `UPDATE` statement.

Treatment of empty or incorrect field values differs from that just described if the SQL mode is set to a restrictive value. For example, if `sql_mode` is set to `TRADITIONAL`, conversion of an empty value or a value such as `'x'` for a numeric column results in an error, not conversion to 0. (With `LOCAL` or `IGNORE`, warnings occur rather than errors, even with a restrictive `sql_mode` value, and the row is inserted using the same closest-value behavior used for nonrestrictive SQL modes. This occurs because the server has no way to stop transmission of the file in the middle of the operation.)

`TIMESTAMP` columns are set to the current date and time only if there is a `NULL` value for the column (that is, `\N`) and the column is not declared to permit `NULL` values, or if the `TIMESTAMP` column's default value is the current timestamp and it is omitted from the field list when a field list is specified.

`LOAD DATA` regards all input as strings, so you cannot use numeric values for `ENUM` or `SET` columns the way you can with `INSERT` statements. All `ENUM` and `SET` values must be specified as strings.

`BIT` values cannot be loaded directly using binary notation (for example, `b'011010'`). To work around this, use the `SET` clause to strip off the leading `b'` and trailing `'` and perform a base-2 to base-10 conversion so that MySQL loads the values into the `BIT` column properly:

```
shell> cat /tmp/bit_test.txt
b'10'
b'1111111'
shell> mysql test
mysql> LOAD DATA INFILE '/tmp/bit_test.txt'
       INTO TABLE bit_test (@var1)
       SET b = CAST(CONV(MID(@var1, 3, LENGTH(@var1)-3), 2, 10) AS UNSIGNED);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT BIN(b+0) FROM bit_test;
+----------+
| BIN(b+0) |
+----------+
| 10       |
| 1111111  |
+----------+
2 rows in set (0.00 sec)
```

For `BIT` values in `0b` binary notation (for example, `0b011010`), use this `SET` clause instead to strip off the leading `0b`:

```
SET b = CAST(CONV(MID(@var1, 3, LENGTH(@var1)-2), 2, 10) AS UNSIGNED)
```

Statement Result Information

When the `LOAD DATA` statement finishes, it returns an information string in the following format:

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

Warnings occur under the same circumstances as when values are inserted using the `INSERT` statement (see Section 13.2.6, “INSERT Syntax”), except that `LOAD DATA` also generates warnings when there are too few or too many fields in the input row.

You can use `SHOW WARNINGS` to get a list of the first `max_error_count` warnings as information about what went wrong. See Section 13.7.7.40, “SHOW WARNINGS Syntax”.

If you are using the C API, you can get information about the statement by calling the `mysql_info()` function. See Section 28.7.7.36, “mysql_info()”.

Miscellaneous Topics

On Unix, if you need `LOAD DATA` to read from a pipe, you can use the following technique (the example loads a listing of the `/` directory into the table `db1.t1`):

```
mkfifo /mysql/data/db1/ls.dat
chmod 666 /mysql/data/db1/ls.dat
find / -ls > /mysql/data/db1/ls.dat &
mysql -e "LOAD DATA INFILE 'ls.dat' INTO TABLE t1" db1
```

Here you must run the command that generates the data to be loaded and the `mysql` commands either on separate terminals, or run the data generation process in the background (as shown in the preceding example). If you do not do this, the pipe will block until data is read by the `mysql` process.

13.2.8 LOAD XML Syntax

```
LOAD XML
    [LOW_PRIORITY | CONCURRENT] [LOCAL]
    INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE [db_name.]tbl_name
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<tagname>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var
        [, field_name_or_user_var] ...)]
    [SET col_name={expr | DEFAULT},
        [, col_name={expr | DEFAULT}] ...]
```

The `LOAD XML` statement reads data from an XML file into a table. The *`file_name`* must be given as a literal string. The *`tagname`* in the optional `ROWS IDENTIFIED BY` clause must also be given as a literal string, and must be surrounded by angle brackets (`<` and `>`).

`LOAD XML` acts as the complement of running the `mysql` client in XML output mode (that is, starting the client with the `--xml` option). To write data from a table to an XML file, you can invoke the `mysql` client with the `--xml` and `-e` options from the system shell, as shown here:

```
shell> mysql --xml -e 'SELECT * FROM mydb.mytable' > file.xml
```

To read the file back into a table, use `LOAD XML`. By default, the `<row>` element is considered to be the equivalent of a database table row; this can be changed using the `ROWS IDENTIFIED BY` clause.

This statement supports three different XML formats:

- Column names as attributes and column values as attribute values:

```
<row column1="value1" column2="value2" .../>
```

- Column names as tags and column values as the content of these tags:

```
<row>
  <column1>value1</column1>
  <column2>value2</column2>
</row>
```

- Column names are the `name` attributes of `<field>` tags, and values are the contents of these tags:

```
<row>
  <field name='column1'>value1</field>
  <field name='column2'>value2</field>
</row>
```

  This is the format used by other MySQL tools, such as `mysqldump`.

All three formats can be used in the same XML file; the import routine automatically detects the format for each row and interprets it correctly. Tags are matched based on the tag or attribute name and the column name.

The following clauses work essentially the same way for `LOAD XML` as they do for `LOAD DATA`:

- `LOW_PRIORITY` or `CONCURRENT`
- `LOCAL`
- `REPLACE` or `IGNORE`
- `CHARACTER SET`
- `SET`

See Section 13.2.7, “LOAD DATA Syntax”, for more information about these clauses.

`(field_name_or_user_var, ...)` is a list of one or more comma-separated XML fields or user variables. The name of a user variable used for this purpose must match the name of a field from the XML file, prefixed with `@`. You can use field names to select only desired fields. User variables can be employed to store the corresponding field values for subsequent re-use.

The `IGNORE` *`number`* `LINES` or `IGNORE` *`number`* `ROWS` clause causes the first *`number`* rows in the XML file to be skipped. It is analogous to the `LOAD DATA` statement's `IGNORE ... LINES` clause.

Suppose that we have a table named `person`, created as shown here:

```
USE test;

CREATE TABLE person (
    person_id INT NOT NULL PRIMARY KEY,
    fname VARCHAR(40) NULL,
    lname VARCHAR(40) NULL,
    created TIMESTAMP
);
```

Suppose further that this table is initially empty.

Now suppose that we have a simple XML file `person.xml`, whose contents are as shown here:

```
<list>
  <person person_id="1" fname="Kapek" lname="Sainnouine"/>
  <person person_id="2" fname="Sajon" lname="Rondela"/>
  <person person_id="3"><fname>Likame</fname><lname>Örrtmons</lname></person>
  <person person_id="4"><fname>Slar</fname><lname>Manlanth</lname></person>
  <person><field name="person_id">5</field><field name="fname">Stoma</field>
    <field name="lname">Milu</field></person>
  <person><field name="person_id">6</field><field name="fname">Nirtam</field>
    <field name="lname">Sklöd</field></person>
  <person person_id="7"><fname>Sungam</fname><lname>Dulbåd</lname></person>
  <person person_id="8" fname="Sraref" lname="Encmelt"/>
</list>
```

Each of the permissible XML formats discussed previously is represented in this example file.

To import the data in `person.xml` into the `person` table, you can use this statement:

```
mysql> LOAD XML LOCAL INFILE 'person.xml'
    ->   INTO TABLE person
    ->   ROWS IDENTIFIED BY '<person>';

Query OK, 8 rows affected (0.00 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0
```

Here, we assume that `person.xml` is located in the MySQL data directory. If the file cannot be found, the following error results:

```
ERROR 2 (HY000): File '/person.xml' not found (Errcode: 2)
```

The `ROWS IDENTIFIED BY '<person>'` clause means that each `<person>` element in the XML file is considered equivalent to a row in the table into which the data is to be imported. In this case, this is the `person` table in the `test` database.

As can be seen by the response from the server, 8 rows were imported into the `test.person` table. This can be verified by a simple `SELECT` statement:

```
mysql> SELECT * FROM person;
+-----------+--------+------------+---------------------+
| person_id | fname  | lname      | created             |
+-----------+--------+------------+---------------------+
|         1 | Kapek  | Sainnouine | 2007-07-13 16:18:47 |
|         2 | Sajon  | Rondela    | 2007-07-13 16:18:47 |
|         3 | Likame | Örrtmons   | 2007-07-13 16:18:47 |
|         4 | Slar   | Manlanth   | 2007-07-13 16:18:47 |
|         5 | Stoma  | Nilu       | 2007-07-13 16:18:47 |
|         6 | Nirtam | Sklöd      | 2007-07-13 16:18:47 |
|         7 | Sungam | Dulbåd     | 2007-07-13 16:18:47 |
|         8 | Sreraf | Encmelt    | 2007-07-13 16:18:47 |
+-----------+--------+------------+---------------------+
8 rows in set (0.00 sec)
```

This shows, as stated earlier in this section, that any or all of the 3 permitted XML formats may appear in a single file and be read using `LOAD XML`.

The inverse of the import operation just shown—that is, dumping MySQL table data into an XML file—can be accomplished using the `mysql` client from the system shell, as shown here:

```
shell> mysql --xml -e "SELECT * FROM test.person" > person-dump.xml
shell> cat person-dump.xml
<?xml version="1.0"?>

<resultset statement="SELECT * FROM test.person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
 <field name="person_id">1</field>
 <field name="fname">Kapek</field>
 <field name="lname">Sainnouine</field>
  </row>

  <row>
 <field name="person_id">2</field>
 <field name="fname">Sajon</field>
 <field name="lname">Rondela</field>
  </row>

  <row>
 <field name="person_id">3</field>
 <field name="fname">Likema</field>
 <field name="lname">Örrtmons</field>
  </row>

  <row>
 <field name="person_id">4</field>
 <field name="fname">Slar</field>
 <field name="lname">Manlanth</field>
  </row>

  <row>
 <field name="person_id">5</field>
 <field name="fname">Stoma</field>
 <field name="lname">Nilu</field>
  </row>

  <row>
 <field name="person_id">6</field>
 <field name="fname">Nirtam</field>
 <field name="lname">Sklöd</field>
  </row>

  <row>
 <field name="person_id">7</field>
 <field name="fname">Sungam</field>
 <field name="lname">Dulbåd</field>
  </row>

  <row>
 <field name="person_id">8</field>
 <field name="fname">Sreraf</field>
 <field name="lname">Encmelt</field>
  </row>
</resultset>
```

Note

The `--xml` option causes the `mysql` client to use XML formatting for its output; the `-e` option causes the client to execute the SQL statement immediately following the option. See Section 4.5.1, “`mysql` — The MySQL Command-Line Client”.

You can verify that the dump is valid by creating a copy of the `person` table and importing the dump file into the new table, like this:

```
mysql> USE test;
```

```
mysql> CREATE TABLE person2 LIKE person;
Query OK, 0 rows affected (0.00 sec)

mysql> LOAD XML LOCAL INFILE 'person-dump.xml'
    ->   INTO TABLE person2;
Query OK, 8 rows affected (0.01 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT * FROM person2;
+-----------+--------+------------+---------------------+
| person_id | fname  | lname      | created             |
+-----------+--------+------------+---------------------+
|         1 | Kapek  | Sainnouine | 2007-07-13 16:18:47 |
|         2 | Sajon  | Rondela    | 2007-07-13 16:18:47 |
|         3 | Likema | Örrtmons   | 2007-07-13 16:18:47 |
|         4 | Slar   | Manlanth   | 2007-07-13 16:18:47 |
|         5 | Stoma  | Nilu       | 2007-07-13 16:18:47 |
|         6 | Nirtam | Sklöd      | 2007-07-13 16:18:47 |
|         7 | Sungam | Dulbåd     | 2007-07-13 16:18:47 |
|         8 | Sreraf | Encmelt    | 2007-07-13 16:18:47 |
+-----------+--------+------------+---------------------+
8 rows in set (0.00 sec)
```

There is no requirement that every field in the XML file be matched with a column in the corresponding table. Fields which have no corresponding columns are skipped. You can see this by first emptying the `person2` table and dropping the `created` column, then using the same `LOAD XML` statement we just employed previously, like this:

```
mysql> TRUNCATE person2;
Query OK, 8 rows affected (0.26 sec)

mysql> ALTER TABLE person2 DROP COLUMN created;
Query OK, 0 rows affected (0.52 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE person2\G
*************************** 1. row ***************************
       Table: person2
Create Table: CREATE TABLE `person2` (
  `person_id` int(11) NOT NULL,
  `fname` varchar(40) DEFAULT NULL,
  `lname` varchar(40) DEFAULT NULL,
  PRIMARY KEY (`person_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> LOAD XML LOCAL INFILE 'person-dump.xml'
    ->   INTO TABLE person2;
Query OK, 8 rows affected (0.01 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT * FROM person2;
+-----------+--------+------------+
| person_id | fname  | lname      |
+-----------+--------+------------+
|         1 | Kapek  | Sainnouine |
|         2 | Sajon  | Rondela    |
|         3 | Likema | Örrtmons   |
|         4 | Slar   | Manlanth   |
|         5 | Stoma  | Nilu       |
|         6 | Nirtam | Sklöd      |
|         7 | Sungam | Dulbåd     |
|         8 | Sreraf | Encmelt    |
+-----------+--------+------------+
8 rows in set (0.00 sec)
```

The order in which the fields are given within each row of the XML file does not affect the operation of `LOAD XML`; the field order can vary from row to row, and is not required to be in the same order as the corresponding columns in the table.

As mentioned previously, you can use a `(field_name_or_user_var, ...)` list of one or more XML fields (to select desired fields only) or user variables (to store the corresponding field values for later use). User variables can be especially useful when you want to insert data from an XML file into table columns whose names do not match those of the XML fields. To see how this works, we first create a table named `individual` whose structure matches that of the `person` table, but whose columns are named differently:

```
mysql> CREATE TABLE individual (
    ->     individual_id INT NOT NULL PRIMARY KEY,
    ->     name1 VARCHAR(40) NULL,
    ->     name2 VARCHAR(40) NULL,
    ->     made TIMESTAMP
    -> );
Query OK, 0 rows affected (0.42 sec)
```

In this case, you cannot simply load the XML file directly into the table, because the field and column names do not match:

```
mysql> LOAD XML INFILE '../bin/person-dump.xml' INTO TABLE test.individual;
ERROR 1263 (22004): Column set to default value; NULL supplied to NOT NULL column 'individual_id' at row 1
```

This happens because the MySQL server looks for field names matching the column names of the target table. You can work around this problem by selecting the field values into user variables, then setting the target table's columns equal to the values of those variables using `SET`. You can perform both of these operations in a single statement, as shown here:

```
mysql> LOAD XML INFILE '../bin/person-dump.xml'
    ->     INTO TABLE test.individual (@person_id, @fname, @lname, @created)
    ->     SET individual_id=@person_id, name1=@fname, name2=@lname, made=@created;
Query OK, 8 rows affected (0.05 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT * FROM individual;
+---------------+--------+------------+---------------------+
| individual_id | name1  | name2      | made                |
+---------------+--------+------------+---------------------+
|             1 | Kapek  | Sainnouine | 2007-07-13 16:18:47 |
|             2 | Sajon  | Rondela    | 2007-07-13 16:18:47 |
|             3 | Likema | Örrtmons   | 2007-07-13 16:18:47 |
|             4 | Slar   | Manlanth   | 2007-07-13 16:18:47 |
|             5 | Stoma  | Nilu       | 2007-07-13 16:18:47 |
|             6 | Nirtam | Sklöd      | 2007-07-13 16:18:47 |
|             7 | Sungam | Dulbåd     | 2007-07-13 16:18:47 |
|             8 | Srraf  | Encmelt    | 2007-07-13 16:18:47 |
+---------------+--------+------------+---------------------+
8 rows in set (0.00 sec)
```

The names of the user variables *must* match those of the corresponding fields from the XML file, with the addition of the required `@` prefix to indicate that they are variables. The user variables need not be listed or assigned in the same order as the corresponding fields.

Using a `ROWS IDENTIFIED BY '<tagname>'` clause, it is possible to import data from the same XML file into database tables with different definitions. For this example, suppose that you have a file named `address.xml` which contains the following XML:

```
<?xml version="1.0"?>

<list>
  <person person_id="1">
    <fname>Robert</fname>
    <lname>Jones</lname>
    <address address_id="1" street="Mill Creek Road" zip="45365" city="Sidney"/>
    <address address_id="2" street="Main Street" zip="28681" city="Taylorsville"/>
  </person>

  <person person_id="2">
    <fname>Mary</fname>
    <lname>Smith</lname>
    <address address_id="3" street="River Road" zip="80239" city="Denver"/>
    <!-- <address address_id="4" street="North Street" zip="37920" city="Knoxville"/> -->
  </person>

</list>
```

You can again use the `test.person` table as defined previously in this section, after clearing all the existing records from the table and then showing its structure as shown here:

```
mysql< TRUNCATE person;
Query OK, 0 rows affected (0.04 sec)

mysql< SHOW CREATE TABLE person\G
*************************** 1. row ***************************
       Table: person
Create Table: CREATE TABLE `person` (
  `person_id` int(11) NOT NULL,
  `fname` varchar(40) DEFAULT NULL,
  `lname` varchar(40) DEFAULT NULL,
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`person_id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

Now create an `address` table in the `test` database using the following `CREATE TABLE` statement:

```
CREATE TABLE address (
    address_id INT NOT NULL PRIMARY KEY,
    person_id INT NULL,
    street VARCHAR(40) NULL,
    zip INT NULL,
    city VARCHAR(40) NULL,
    created TIMESTAMP
);
```

To import the data from the XML file into the `person` table, execute the following `LOAD XML` statement, which specifies that rows are to be specified by the `<person>` element, as shown here;

```
mysql> LOAD XML LOCAL INFILE 'address.xml'
    ->   INTO TABLE person
    ->   ROWS IDENTIFIED BY '<person>';
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
```

You can verify that the records were imported using a `SELECT` statement:

```
mysql> SELECT * FROM person;
+-----------+--------+-------+---------------------+
```

```
| person_id | fname  | lname | created             |
+-----------+--------+-------+---------------------+
|         1 | Robert | Jones | 2007-07-24 17:37:06 |
|         2 | Mary   | Smith | 2007-07-24 17:37:06 |
+-----------+--------+-------+---------------------+
2 rows in set (0.00 sec)
```

Since the `<address>` elements in the XML file have no corresponding columns in the `person` table, they are skipped.

To import the data from the `<address>` elements into the `address` table, use the `LOAD XML` statement shown here:

```
mysql> LOAD XML LOCAL INFILE 'address.xml'
    ->   INTO TABLE address
    ->   ROWS IDENTIFIED BY '<address>';
Query OK, 3 rows affected (0.00 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0
```

You can see that the data was imported using a `SELECT` statement such as this one:

```
mysql> SELECT * FROM address;
+------------+-----------+-----------------+-------+--------------+---------------------+
| address_id | person_id | street          | zip   | city         | created             |
+------------+-----------+-----------------+-------+--------------+---------------------+
|          1 |         1 | Mill Creek Road | 45365 | Sidney       | 2007-07-24 17:37:37 |
|          2 |         1 | Main Street     | 28681 | Taylorsville | 2007-07-24 17:37:37 |
|          3 |         2 | River Road      | 80239 | Denver       | 2007-07-24 17:37:37 |
+------------+-----------+-----------------+-------+--------------+---------------------+
3 rows in set (0.00 sec)
```

The data from the `<address>` element that is enclosed in XML comments is not imported. However, since there is a `person_id` column in the `address` table, the value of the `person_id` attribute from the parent `<person>` element for each `<address>` *is* imported into the `address` table.

Security Considerations. As with the `LOAD DATA` statement, the transfer of the XML file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the `LOAD XML` statement. Such a server could access any file on the client host to which the client user has read access.

In a Web environment, clients usually connect to MySQL from a Web server. A user that can run any command against the MySQL server can use `LOAD XML LOCAL` to read any files to which the Web server process has read access. In this environment, the client with respect to the MySQL server is actually the Web server, not the remote program being run by the user who connects to the Web server.

You can disable loading of XML files from clients by starting the server with `--local-infile=0` or `--local-infile=OFF`. This option can also be used when starting the `mysql` client to disable `LOAD XML` for the duration of the client session.

To prevent a client from loading XML files from the server, do not grant the `FILE` privilege to the corresponding MySQL user account, or revoke this privilege if the client user account already has it.

Important

Revoking the `FILE` privilege (or not granting it in the first place) keeps the user only from executing the `LOAD XML` statement (as well as the `LOAD_FILE()` function; it does *not* prevent the user from executing `LOAD XML LOCAL`. To disallow this statement, you must start the server or the client with `--local-infile=OFF`.

> In other words, the `FILE` privilege affects only whether the client can read files on the server; it has no bearing on whether the client can read files on the local file system.

For partitioned tables using storage engines that employ table locks, such as `MyISAM`, any locks caused by `LOAD XML` perform locks on all partitions of the table. This does not apply to tables using storage engines which employ row-level locking, such as `InnoDB`. For more information, see Partitioning and Locking.

13.2.9 REPLACE Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    {VALUES | VALUE} (value_list) [, (value_list)] ...

REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    SET assignment_list

REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    SELECT ...

value:
    {expr | DEFAULT}

value_list:
    value [, value] ...

assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

`REPLACE` works exactly like `INSERT`, except that if an old row in the table has the same value as a new row for a `PRIMARY KEY` or a `UNIQUE` index, the old row is deleted before the new row is inserted. See Section 13.2.6, “INSERT Syntax”.

`REPLACE` is a MySQL extension to the SQL standard. It either inserts, or *deletes* and inserts. For another MySQL extension to standard SQL—that either inserts or *updates*—see Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”.

`DELAYED` inserts and replaces were deprecated in MySQL 5.6. In MySQL 8.0, `DELAYED` is not supported. The server recognizes but ignores the `DELAYED` keyword, handles the replace as a nondelayed replace, and generates an `ER_WARN_LEGACY_SYNTAX_CONVERTED` warning. (“REPLACE DELAYED is no longer supported. The statement was converted to REPLACE.”) The `DELAYED` keyword will be removed in a future release.

Note

`REPLACE` makes sense only if a table has a `PRIMARY KEY` or `UNIQUE` index. Otherwise, it becomes equivalent to `INSERT`, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values specified in the REPLACE statement. Any missing columns are set to their default values, just as happens for INSERT. You cannot refer to values from the current row and use them in the new row. If you use an assignment such as SET *col_name* = *col_name* + 1, the reference to the column name on the right hand side is treated as DEFAULT(*col_name*), so the assignment is equivalent to SET *col_name* = DEFAULT(*col_name*) + 1.

To use REPLACE, you must have both the INSERT and DELETE privileges for the table.

If a generated column is replaced explicitly, the only permitted value is DEFAULT. For information about generated columns, see Section 13.1.20.9, “CREATE TABLE and Generated Columns”.

REPLACE supports explicit partition selection using the PARTITION keyword with a list of comma-separated names of partitions, subpartitions, or both. As with INSERT, if it is not possible to insert the new row into any of these partitions or subpartitions, the REPLACE statement fails with the error `Found a row not matching the given partition set`. For more information and examples, see Section 23.5, “Partition Selection”.

The REPLACE statement returns a count to indicate the number of rows affected. This is the sum of the rows deleted and inserted. If the count is 1 for a single-row REPLACE, a row was inserted and no rows were deleted. If the count is greater than 1, one or more old rows were deleted before the new row was inserted. It is possible for a single row to replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

The affected-rows count makes it easy to determine whether REPLACE only added a row or whether it also replaced any rows: Check whether the count is 1 (added) or greater (replaced).

If you are using the C API, the affected-rows count can be obtained using the mysql_affected_rows() function.

You cannot replace into a table and select from the same table in a subquery.

MySQL uses the following algorithm for REPLACE (and LOAD DATA ... REPLACE):

1. Try to insert the new row into the table
2. While the insertion fails because a duplicate-key error occurs for a primary key or unique index:
   a. Delete from the table the conflicting row that has the duplicate key value
   b. Try again to insert the new row into the table

It is possible that in the case of a duplicate-key error, a storage engine may perform the REPLACE as an update rather than a delete plus insert, but the semantics are the same. There are no user-visible effects other than a possible difference in how the storage engine increments Handler_*xxx* status variables.

Because the results of REPLACE ... SELECT statements depend on the ordering of rows from the SELECT and this order cannot always be guaranteed, it is possible when logging these statements for the master and the slave to diverge. For this reason, REPLACE ... SELECT statements are flagged as unsafe for statement-based replication. such statements produce a warning in the error log when using statement-based mode and are written to the binary log using the row-based format when using MIXED mode. See also Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

When modifying an existing table that is not partitioned to accommodate partitioning, or, when modifying the partitioning of an already partitioned table, you may consider altering the table's primary key (see Section 23.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”). You should be aware that, if you do

this, the results of REPLACE statements may be affected, just as they would be if you modified the primary key of a nonpartitioned table. Consider the table created by the following CREATE TABLE statement:

```
CREATE TABLE test (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  data VARCHAR(64) DEFAULT NULL,
  ts TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id)
);
```

When we create this table and run the statements shown in the mysql client, the result is as follows:

```
mysql> REPLACE INTO test VALUES (1, 'Old', '2014-08-20 18:47:00');
Query OK, 1 row affected (0.04 sec)

mysql> REPLACE INTO test VALUES (1, 'New', '2014-08-20 18:47:42');
Query OK, 2 rows affected (0.04 sec)

mysql> SELECT * FROM test;
+----+------+---------------------+
| id | data | ts                  |
+----+------+---------------------+
|  1 | New  | 2014-08-20 18:47:42 |
+----+------+---------------------+
1 row in set (0.00 sec)
```

Now we create a second table almost identical to the first, except that the primary key now covers 2 columns, as shown here (emphasized text):

```
CREATE TABLE test2 (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  data VARCHAR(64) DEFAULT NULL,
  ts TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id, ts)
);
```

When we run on test2 the same two REPLACE statements as we did on the original test table, we obtain a different result:

```
mysql> REPLACE INTO test2 VALUES (1, 'Old', '2014-08-20 18:47:00');
Query OK, 1 row affected (0.05 sec)

mysql> REPLACE INTO test2 VALUES (1, 'New', '2014-08-20 18:47:42');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM test2;
+----+------+---------------------+
| id | data | ts                  |
+----+------+---------------------+
|  1 | Old  | 2014-08-20 18:47:00 |
|  1 | New  | 2014-08-20 18:47:42 |
+----+------+---------------------+
2 rows in set (0.00 sec)
```

This is due to the fact that, when run on test2, both the id and ts column values must match those of an existing row for the row to be replaced; otherwise, a row is inserted.

A REPLACE statement affecting a partitioned table using a storage engine such as MyISAM that employs table-level locks locks only those partitions containing rows that match the REPLACE statement WHERE clause, as long as none of the table partitioning columns are updated; otherwise the entire table is locked.

(For storage engines such as InnoDB that employ row-level locking, no locking of partitions takes place.) For more information, see Partitioning and Locking.

13.2.10 SELECT Syntax

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}, ... [WITH ROLLUP]]
    [HAVING where_condition]
    [WINDOW window_name AS (window_spec)
        [, window_name AS (window_spec)] ...]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR {UPDATE | SHARE} [OF tbl_name [, tbl_name] ...] [NOWAIT | SKIP LOCKED]
      | LOCK IN SHARE MODE]]
```

SELECT is used to retrieve rows selected from one or more tables, and can include UNION statements and subqueries. See Section 13.2.10.3, “UNION Syntax”, and Section 13.2.11, “Subquery Syntax”. A SELECT statement can start with a WITH clause to define common table expressions accessible within the SELECT. See Section 13.2.13, “WITH Syntax (Common Table Expressions)”.

The most commonly used clauses of SELECT statements are these:

- Each *select_expr* indicates a column that you want to retrieve. There must be at least one *select_expr*.
- *table_references* indicates the table or tables from which to retrieve rows. Its syntax is described in Section 13.2.10.2, “JOIN Syntax”.
- SELECT supports explicit partition selection using the PARTITION with a list of partitions or subpartitions (or both) following the name of the table in a *table_reference* (see Section 13.2.10.2, “JOIN Syntax”). In this case, rows are selected only from the partitions listed, and any other partitions of the table are ignored. For more information and examples, see Section 23.5, “Partition Selection”.

  SELECT ... PARTITION from tables using storage engines such as MyISAM that perform table-level locks (and thus partition locks) lock only the partitions or subpartitions named by the PARTITION option.

  For more information, see Partitioning and Locking.
- The WHERE clause, if given, indicates the condition or conditions that rows must satisfy to be selected. *where_condition* is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no WHERE clause.

  In the WHERE expression, you can use any of the functions and operators that MySQL supports, except for aggregate (summary) functions. See Section 9.5, “Expressions”, and Chapter 12, *Functions and Operators*.

SELECT can also be used to retrieve rows computed without reference to any table.

For example:

```
mysql> SELECT 1 + 1;
        -> 2
```

You are permitted to specify DUAL as a dummy table name in situations where no tables are referenced:

```
mysql> SELECT 1 + 1 FROM DUAL;
        -> 2
```

DUAL is purely for the convenience of people who require that all SELECT statements should have FROM and possibly other clauses. MySQL may ignore the clauses. MySQL does not require FROM DUAL if no tables are referenced.

In general, clauses used must be given in exactly the order shown in the syntax description. For example, a HAVING clause must come after any GROUP BY clause and before any ORDER BY clause. The exception is that the INTO clause can appear either as shown in the syntax description or immediately following the *select_expr* list. For more information about INTO, see Section 13.2.10.1, “SELECT ... INTO Syntax”.

The list of *select_expr* terms comprises the select list that indicates which columns to retrieve. Terms specify a column or expression or can use *-shorthand:

- A select list consisting only of a single unqualified * can be used as shorthand to select all columns from all tables:

  ```
  SELECT * FROM t1 INNER JOIN t2 ...
  ```

- *tbl_name*.* can be used as a qualified shorthand to select all columns from the named table:

  ```
  SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ...
  ```

- Use of an unqualified * with other items in the select list may produce a parse error. To avoid this problem, use a qualified *tbl_name*.* reference

  ```
  SELECT AVG(score), t1.* FROM t1 ...
  ```

The following list provides additional information about other SELECT clauses:

- A *select_expr* can be given an alias using AS *alias_name*. The alias is used as the expression's column name and can be used in GROUP BY, ORDER BY, or HAVING clauses. For example:

  ```
  SELECT CONCAT(last_name,', ',first_name) AS full_name
    FROM mytable ORDER BY full_name;
  ```

  The AS keyword is optional when aliasing a *select_expr* with an identifier. The preceding example could have been written like this:

  ```
  SELECT CONCAT(last_name,', ',first_name) full_name
    FROM mytable ORDER BY full_name;
  ```

  However, because the AS is optional, a subtle problem can occur if you forget the comma between two *select_expr* expressions: MySQL interprets the second as an alias name. For example, in the following statement, columnb is treated as an alias name:

```
SELECT columna columnb FROM mytable;
```

For this reason, it is good practice to be in the habit of using AS explicitly when specifying column aliases.

It is not permissible to refer to a column alias in a WHERE clause, because the column value might not yet be determined when the WHERE clause is executed. See Section B.4.4.4, “Problems with Column Aliases”.

- The FROM *table_references* clause indicates the table or tables from which to retrieve rows. If you name more than one table, you are performing a join. For information on join syntax, see Section 13.2.10.2, “JOIN Syntax”. For each table specified, you can optionally specify an alias.

```
tbl_name [[AS] alias] [index_hint]
```

The use of index hints provides the optimizer with information about how to choose indexes during query processing. For a description of the syntax for specifying these hints, see Section 8.9.4, “Index Hints”.

You can use SET max_seeks_for_key=*value* as an alternative way to force MySQL to prefer key scans instead of table scans. See Section 5.1.8, “Server System Variables”.

- You can refer to a table within the default database as *tbl_name*, or as *db_name*.*tbl_name* to specify a database explicitly. You can refer to a column as *col_name*, *tbl_name*.*col_name*, or *db_name*.*tbl_name*.*col_name*. You need not specify a *tbl_name* or *db_name*.*tbl_name* prefix for a column reference unless the reference would be ambiguous. See Section 9.2.2, “Identifier Qualifiers”, for examples of ambiguity that require the more explicit column reference forms.

- A table reference can be aliased using *tbl_name* AS *alias_name* or *tbl_name* *alias_name*. These statements are equivalent:

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
  WHERE t1.name = t2.name;

SELECT t1.name, t2.salary FROM employee t1, info t2
  WHERE t1.name = t2.name;
```

- Columns selected for output can be referred to in ORDER BY and GROUP BY clauses using column names, column aliases, or column positions. Column positions are integers and begin with 1:

```
SELECT college, region, seed FROM tournament
  ORDER BY region, seed;

SELECT college, region AS r, seed AS s FROM tournament
  ORDER BY r, s;

SELECT college, region, seed FROM tournament
  ORDER BY 2, 3;
```

To sort in reverse order, add the DESC (descending) keyword to the name of the column in the ORDER BY clause that you are sorting by. The default is ascending order; this can be specified explicitly using the ASC keyword.

If ORDER BY occurs within a subquery and also is applied in the outer query, the outermost ORDER BY takes precedence. For example, results for the following statement are sorted in descending order, not ascending order:

```
(SELECT ... ORDER BY a) ORDER BY a DESC;
```

Use of column positions is deprecated because the syntax has been removed from the SQL standard.

- Prior to MySQL 8.0.13, MySQL supported a nonstandard syntax extension that permitted explicit `ASC` or `DESC` designators for `GROUP BY` columns. MySQL 8.0.12 and later supports `ORDER BY` with grouping functions so that use of this extension is no longer necessary. (Bug #86312, Bug #26073525) This also means you can sort on an arbitrary column or columns when using `GROUP BY`, like this:

  ```
  SELECT a, b, COUNT(c) AS t FROM test_table GROUP BY a,b ORDER BY a,t DESC;
  ```

  As of MySQL 8.0.13, the `GROUP BY` extension is no longer supported: `ASC` or `DESC` designators for `GROUP BY` columns are not permitted.

- When you use `ORDER BY` or `GROUP BY` to sort a column in a `SELECT`, the server sorts values using only the initial number of bytes indicated by the `max_sort_length` system variable.

- MySQL extends the use of `GROUP BY` to permit selecting fields that are not mentioned in the `GROUP BY` clause. If you are not getting the results that you expect from your query, please read the description of `GROUP BY` found in Section 12.20, “Aggregate (GROUP BY) Functions”.

- `GROUP BY` permits a `WITH ROLLUP` modifier. See Section 12.20.2, “GROUP BY Modifiers”.

  Previously, it was not permitted to use `ORDER BY` in a query having a `WITH ROLLUP` modifier. This restriction is lifted as of MySQL 8.0.12. See Section 12.20.2, “GROUP BY Modifiers”.

- The `HAVING` clause is applied nearly last, just before items are sent to the client, with no optimization. (`LIMIT` is applied after `HAVING`.)

  The SQL standard requires that `HAVING` must reference only columns in the `GROUP BY` clause or columns used in aggregate functions. However, MySQL supports an extension to this behavior, and permits `HAVING` to refer to columns in the `SELECT` list and columns in outer subqueries as well.

  If the `HAVING` clause refers to a column that is ambiguous, a warning occurs. In the following statement, `col2` is ambiguous because it is used as both an alias and a column name:

  ```
  SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
  ```

  Preference is given to standard SQL behavior, so if a `HAVING` column name is used both in `GROUP BY` and as an aliased column in the output column list, preference is given to the column in the `GROUP BY` column.

- Do not use `HAVING` for items that should be in the `WHERE` clause. For example, do not write the following:

  ```
  SELECT col_name FROM tbl_name HAVING col_name > 0;
  ```

  Write this instead:

  ```
  SELECT col_name FROM tbl_name WHERE col_name > 0;
  ```

- The `HAVING` clause can refer to aggregate functions, which the `WHERE` clause cannot:

  ```
  ```

```
SELECT user, MAX(salary) FROM users
  GROUP BY user HAVING MAX(salary) > 10;
```

(This did not work in some older versions of MySQL.)

- MySQL permits duplicate column names. That is, there can be more than one *`select_expr`* with the same name. This is an extension to standard SQL. Because MySQL also permits `GROUP BY` and `HAVING` to refer to *`select_expr`* values, this can result in an ambiguity:

```
SELECT 12 AS a, a FROM t GROUP BY a;
```

  In that statement, both columns have the name `a`. To ensure that the correct column is used for grouping, use different names for each *`select_expr`*.

- The `WINDOW` clause, if present, defines named windows that can be referred to by window functions. For details, see Section 12.21.4, “Named Windows”.

- MySQL resolves unqualified column or alias references in `ORDER BY` clauses by searching in the *`select_expr`* values, then in the columns of the tables in the `FROM` clause. For `GROUP BY` or `HAVING` clauses, it searches the `FROM` clause before searching in the *`select_expr`* values. (For `GROUP BY` and `HAVING`, this differs from the pre-MySQL 5.0 behavior that used the same rules as for `ORDER BY`.)

- The `LIMIT` clause can be used to constrain the number of rows returned by the `SELECT` statement. `LIMIT` takes one or two numeric arguments, which must both be nonnegative integer constants, with these exceptions:

  - Within prepared statements, `LIMIT` parameters can be specified using `?` placeholder markers.

  - Within stored programs, `LIMIT` parameters can be specified using integer-valued routine parameters or local variables.

  With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1):

```
SELECT * FROM tbl LIMIT 5,10;  # Retrieve rows 6-15
```

  To retrieve all rows from a certain offset up to the end of the result set, you can use some large number for the second parameter. This statement retrieves all rows from the 96th row to the last:

```
SELECT * FROM tbl LIMIT 95,18446744073709551615;
```

  With one argument, the value specifies the number of rows to return from the beginning of the result set:

```
SELECT * FROM tbl LIMIT 5;     # Retrieve first 5 rows
```

  In other words, `LIMIT` *`row_count`* is equivalent to `LIMIT 0,` *`row_count`*.

  For prepared statements, you can use placeholders. The following statements will return one row from the `tbl` table:

```
SET @a=1;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';
EXECUTE STMT USING @a;
```

  The following statements will return the second to sixth row from the `tbl` table:

```
SET @skip=1; SET @numrows=5;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
EXECUTE STMT USING @skip, @numrows;
```

For compatibility with PostgreSQL, MySQL also supports the `LIMIT row_count OFFSET offset` syntax.

If `LIMIT` occurs within a subquery and also is applied in the outer query, the outermost `LIMIT` takes precedence. For example, the following statement produces two rows, not one:

```
(SELECT ... LIMIT 1) LIMIT 2;
```

- The `SELECT ... INTO` form of `SELECT` enables the query result to be written to a file or stored in variables. For more information, see Section 13.2.10.1, “SELECT ... INTO Syntax”.

- If you use `FOR UPDATE` with a storage engine that uses page or row locks, rows examined by the query are write-locked until the end of the current transaction.

  You cannot use `FOR UPDATE` as part of the `SELECT` in a statement such as `CREATE TABLE new_table SELECT ... FROM old_table ...`. (If you attempt to do so, the statement is rejected with the error `Can't update table 'old_table' while 'new_table' is being created`.)

  `FOR SHARE` and `LOCK IN SHARE MODE` set shared locks that permit other transactions to read the examined rows but not to update or delete them. `FOR SHARE` and `LOCK IN SHARE MODE` are equivalent. However, `FOR SHARE`, like `FOR UPDATE`, supports `NOWAIT`, `SKIP LOCKED`, and `OF tbl_name` options. `FOR SHARE` is a replacement for `LOCK IN SHARE MODE`, but `LOCK IN SHARE MODE` remains available for backward compatibility.

  `NOWAIT` causes a `FOR UPDATE` or `FOR SHARE` query to execute immediately, returning an error if a row lock cannot be obtained due to a lock held by another transaction.

  `SKIP LOCKED` causes a `FOR UPDATE` or `FOR SHARE` query to execute immediately, excluding rows from the result set that are locked by another transaction.

  `NOWAIT` and `SKIP LOCKED` options are unsafe for statement-based replication.

  Note

  Queries that skip locked rows return an inconsistent view of the data. `SKIP LOCKED` is therefore not suitable for general transactional work. However, it may be used to avoid lock contention when multiple sessions access the same queue-like table.

  `OF tbl_name` applies `FOR UPDATE` and `FOR SHARE` queries to named tables. For example:

```
SELECT * FROM t1, t2 FOR SHARE OF t1 FOR UPDATE OF t2;
```

  All tables referenced by the query block are locked when `OF tbl_name` is omitted. Consequently, using a locking clause without `OF tbl_name` in combination with another locking clause returns an error. Specifying the same table in multiple locking clauses returns an error. If an alias is specified as the table name in the `SELECT` statement, a locking clause may only use the alias. If the `SELECT` statement does not specify an alias explicitly, the locking clause may only specify the actual table name.

For more information about FOR UPDATE and FOR SHARE, see Section 15.7.2.4, “Locking Reads”. For additional information about NOWAIT and SKIP LOCKED options, see Locking Read Concurrency with NOWAIT and SKIP LOCKED.

Following the SELECT keyword, you can use a number of modifiers that affect the operation of the statement. HIGH_PRIORITY, STRAIGHT_JOIN, and modifiers beginning with SQL_ are MySQL extensions to standard SQL.

- The ALL and DISTINCT modifiers specify whether duplicate rows should be returned. ALL (the default) specifies that all matching rows should be returned, including duplicates. DISTINCT specifies removal of duplicate rows from the result set. It is an error to specify both modifiers. DISTINCTROW is a synonym for DISTINCT.

  In MySQL 8.0.12 and later, DISTINCT can be used with a query that also uses WITH ROLLUP. (Bug #87450, Bug #26640100)

- HIGH_PRIORITY gives the SELECT higher priority than a statement that updates a table. You should use this only for queries that are very fast and must be done at once. A SELECT HIGH_PRIORITY query that is issued while the table is locked for reading runs even if there is an update statement waiting for the table to be free. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).

  HIGH_PRIORITY cannot be used with SELECT statements that are part of a UNION.

- STRAIGHT_JOIN forces the optimizer to join the tables in the order in which they are listed in the FROM clause. You can use this to speed up a query if the optimizer joins the tables in nonoptimal order. STRAIGHT_JOIN also can be used in the *table_references* list. See Section 13.2.10.2, “JOIN Syntax”.

  STRAIGHT_JOIN does not apply to any table that the optimizer treats as a const or system table. Such a table produces a single row, is read during the optimization phase of query execution, and references to its columns are replaced with the appropriate column values before query execution proceeds. These tables will appear first in the query plan displayed by EXPLAIN. See Section 8.8.1, “Optimizing Queries with EXPLAIN”. This exception may not apply to const or system tables that are used on the NULL-complemented side of an outer join (that is, the right-side table of a LEFT JOIN or the left-side table of a RIGHT JOIN.

- SQL_BIG_RESULT or SQL_SMALL_RESULT can be used with GROUP BY or DISTINCT to tell the optimizer that the result set has many rows or is small, respectively. For SQL_BIG_RESULT, MySQL directly uses disk-based temporary tables if they are created, and prefers sorting to using a temporary table with a key on the GROUP BY elements. For SQL_SMALL_RESULT, MySQL uses in-memory temporary tables to store the resulting table instead of using sorting. This should not normally be needed.

- SQL_BUFFER_RESULT forces the result to be put into a temporary table. This helps MySQL free the table locks early and helps in cases where it takes a long time to send the result set to the client. This modifier can be used only for top-level SELECT statements, not for subqueries or following UNION.

- SQL_CALC_FOUND_ROWS tells MySQL to calculate how many rows there would be in the result set, disregarding any LIMIT clause. The number of rows can then be retrieved with SELECT FOUND_ROWS(). See Section 12.15, “Information Functions”.

Note

The SQL_CALC_FOUND_ROWS query modifier and accompanying FOUND_ROWS() function are deprecated as of MySQL 8.0.17 and will be removed in a future

MySQL version. See the FOUND_ROWS() description for information about an alternative strategy.

- The SQL_CACHE and SQL_NO_CACHE modifiers were used with the query cache prior to MySQL 8.0. The query cache was removed in MySQL 8.0. The SQL_CACHE modifier was removed as well. SQL_NO_CACHE is deprecated, has no effect, and will be removed in a future MySQL release.

A SELECT from a partitioned table using a storage engine such as MyISAM that employs table-level locks locks only those partitions containing rows that match the SELECT statement WHERE clause. (This does not occur with storage engines such as InnoDB that employ row-level locking.) For more information, see Partitioning and Locking.

13.2.10.1 SELECT ... INTO Syntax

The SELECT ... INTO form of SELECT enables a query result to be stored in variables or written to a file:

- SELECT ... INTO *var_list* selects column values and stores them into variables.
- SELECT ... INTO OUTFILE writes the selected rows to a file. Column and line terminators can be specified to produce a specific output format.
- SELECT ... INTO DUMPFILE writes a single row to a file without any formatting.

The SELECT syntax description (see Section 13.2.10, "SELECT Syntax") shows the INTO clause near the end of the statement. It is also possible to use INTO immediately following the *select_expr* list.

An INTO clause should not be used in a nested SELECT because such a SELECT must return its result to the outer context.

The INTO clause can name a list of one or more variables, which can be user-defined variables, stored procedure or function parameters, or stored program local variables. (Within a prepared SELECT ... INTO OUTFILE statement, only user-defined variables are permitted;see Section 13.6.4.2, "Local Variable Scope and Resolution".)

The selected values are assigned to the variables. The number of variables must match the number of columns. The query should return a single row. If the query returns no rows, a warning with error code 1329 occurs (No data), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (Result consisted of more than one row). If it is possible that the statement may retrieve multiple rows, you can use LIMIT 1 to limit the result set to a single row.

```
SELECT id, data INTO @x, @y FROM test.t1 LIMIT 1;
```

User variable names are not case-sensitive. See Section 9.4, "User-Defined Variables".

The SELECT ... INTO OUTFILE '*file_name*' form of SELECT writes the selected rows to a file. The file is created on the server host, so you must have the FILE privilege to use this syntax. *file_name* cannot be an existing file, which among other things prevents files such as /etc/passwd and database tables from being destroyed. The character_set_filesystem system variable controls the interpretation of the file name.

The SELECT ... INTO OUTFILE statement is intended primarily to let you very quickly dump a table to a text file on the server machine. If you want to create the resulting file on some other host than the server host, you normally cannot use SELECT ... INTO OUTFILE since there is no way to write a path to the file relative to the server host's file system.

However, if the MySQL client software is installed on the remote machine, you can instead use a client command such as `mysql -e "SELECT ..." > file_name` to generate the file on the client host.

It is also possible to create the resulting file on a different host other than the server host, if the location of the file on the remote host can be accessed using a network-mapped path on the server's file system. In this case, the presence of `mysql` (or some other MySQL client program) is not required on the target host.

`SELECT ... INTO OUTFILE` is the complement of `LOAD DATA`. Column values are written converted to the character set specified in the `CHARACTER SET` clause. If no such clause is present, values are dumped using the `binary` character set. In effect, there is no character set conversion. If a result set contains columns in several character sets, the output data file will as well and you may not be able to reload the file correctly.

The syntax for the *`export_options`* part of the statement consists of the same `FIELDS` and `LINES` clauses that are used with the `LOAD DATA` statement. See Section 13.2.7, “LOAD DATA Syntax”, for information about the `FIELDS` and `LINES` clauses, including their default values and permissible values.

`FIELDS ESCAPED BY` controls how to write special characters. If the `FIELDS ESCAPED BY` character is not empty, it is used when necessary to avoid ambiguity as a prefix that precedes following characters on output:

- The `FIELDS ESCAPED BY` character
- The `FIELDS [OPTIONALLY] ENCLOSED BY` character
- The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values
- ASCII `NUL` (the zero-valued byte; what is actually written following the escape character is ASCII `0`, not a zero-valued byte)

The `FIELDS TERMINATED BY`, `ENCLOSED BY`, `ESCAPED BY`, or `LINES TERMINATED BY` characters *must* be escaped so that you can read the file back in reliably. ASCII `NUL` is escaped to make it easier to view with some pagers.

The resulting file does not have to conform to SQL syntax, so nothing else need be escaped.

If the `FIELDS ESCAPED BY` character is empty, no characters are escaped and `NULL` is output as `NULL`, not `\N`. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

Here is an example that produces a file in the comma-separated values (CSV) format used by many programs:

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  LINES TERMINATED BY '\n'
  FROM test_table;
```

If you use `INTO DUMPFILE` instead of `INTO OUTFILE`, MySQL writes only one row into the file, without any column or line termination and without performing any escape processing. This is useful if you want to store a `BLOB` value in a file.

Note

Any file created by `INTO OUTFILE` or `INTO DUMPFILE` is owned by the operating system user under whose account `mysqld` run. (You should *never* run `mysqld` as `root` for this and other reasons.) As of MySQL 8.0.17, the umask for file creation

> is 0640; you must have sufficient access privileges to manipulate the file contents. Prior to MySQL 8.0.17, the umask is 0666 and the file is writable by all users on the server host.
>
> If the `secure_file_priv` system variable is set to a nonempty directory name, the file to be written must be located in that directory.

In the context of `SELECT ... INTO` statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log. For additional information, see Section 24.4.5, "Event Scheduler Status".

13.2.10.2 JOIN Syntax

MySQL supports the following `JOIN` syntax for the *`table_references`* part of `SELECT` statements and multiple-table `DELETE` and `UPDATE` statements:

```
table_references:
    escaped_table_reference [, escaped_table_reference] ...

escaped_table_reference:
    table_reference
  | { OJ table_reference }

table_reference:
    table_factor
  | joined_table

table_factor:
    tbl_name [PARTITION (partition_names)]
        [[AS] alias] [index_hint_list]
  | table_subquery [AS] alias [(col_list)]
  | ( table_references )

joined_table:
    table_reference {[INNER | CROSS] JOIN | STRAIGHT_JOIN} table_factor [join_specification]
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_specification
  | table_reference NATURAL [INNER | {LEFT|RIGHT} [OUTER]] JOIN table_factor

join_specification:
    ON search_condition
  | USING (join_column_list)

join_column_list:
    column_name [, column_name] ...

index_hint_list:
    index_hint [, index_hint] ...

index_hint:
    USE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
  | {IGNORE|FORCE} {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)

index_list:
    index_name [, index_name] ...
```

A table reference is also known as a join expression.

A table reference (when it refers to a partitioned table) may contain a `PARTITION` option, including a list of comma-separated partitions, subpartitions, or both. This option follows the name of the table and precedes

any alias declaration. The effect of this option is that rows are selected only from the listed partitions or subpartitions. Any partitions or subpartitions not named in the list are ignored. For more information and examples, see Section 23.5, “Partition Selection”.

The syntax of *`table_factor`* is extended in MySQL in comparison with standard SQL. The standard accepts only *`table_reference`*, not a list of them inside a pair of parentheses.

This is a conservative extension if each comma in a list of *`table_reference`* items is considered as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                 ON (t2.a = t1.a AND t3.b = t1.b AND t4.c = t1.c)
```

is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                 ON (t2.a = t1.a AND t3.b = t1.b AND t4.c = t1.c)
```

In MySQL, `JOIN`, `CROSS JOIN`, and `INNER JOIN` are syntactic equivalents (they can replace each other). In standard SQL, they are not equivalent. `INNER JOIN` is used with an `ON` clause, `CROSS JOIN` is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. MySQL also supports nested joins. See Section 8.2.1.8, “Nested Join Optimization”.

Index hints can be specified to affect how the MySQL optimizer makes use of indexes. For more information, see Section 8.9.4, “Index Hints”. Optimizer hints and the `optimizer_switch` system variable are other ways to influence optimizer use of indexes. See Section 8.9.3, “Optimizer Hints”, and Section 8.9.2, “Switchable Optimizations”.

The following list describes general factors to take into account when writing joins:

- A table reference can be aliased using *`tbl_name`* `AS` *`alias_name`* or *`tbl_name alias_name`*:

  ```
  SELECT t1.name, t2.salary
    FROM employee AS t1 INNER JOIN info AS t2 ON t1.name = t2.name;

  SELECT t1.name, t2.salary
    FROM employee t1 INNER JOIN info t2 ON t1.name = t2.name;
  ```

- A *`table_subquery`* is also known as a derived table or subquery in the `FROM` clause. See Section 13.2.11.8, “Derived Tables”. Such subqueries *must* include an alias to give the subquery result a table name, and may optionally include a list of table column names in parentheses. A trivial example follows:

  ```
  SELECT * FROM (SELECT 1, 2, 3) AS t1;
  ```

- The maximum number of tables that can be referenced in a single join is 61. This includes a join handled by merging derived tables and views in the `FROM` clause into the outer query block (see Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”).

- `INNER JOIN` and `,` (comma) are semantically equivalent in the absence of a join condition: both produce a Cartesian product between the specified tables (that is, each and every row in the first table is joined to each and every row in the second table).

However, the precedence of the comma operator is less than that of `INNER JOIN`, `CROSS JOIN`, `LEFT JOIN`, and so on. If you mix comma joins with the other join types when there is a join condition, an error of the form `Unknown column 'col_name' in 'on clause'` may occur. Information about dealing with this problem is given later in this section.

- The *`search_condition`* used with `ON` is any conditional expression of the form that can be used in a `WHERE` clause. Generally, the `ON` clause serves for conditions that specify how to join tables, and the `WHERE` clause restricts which rows to include in the result set.
- If there is no matching row for the right table in the `ON` or `USING` part in a `LEFT JOIN`, a row with all columns set to `NULL` is used for the right table. You can use this fact to find rows in a table that have no counterpart in another table:

```
SELECT left_tbl.*
  FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id
  WHERE right_tbl.id IS NULL;
```

  This example finds all rows in `left_tbl` with an `id` value that is not present in `right_tbl` (that is, all rows in `left_tbl` with no corresponding row in `right_tbl`). See Section 8.2.1.9, “Outer Join Optimization”.
- The `USING(`*`join_column_list`*`)` clause names a list of columns that must exist in both tables. If tables `a` and `b` both contain columns `c1`, `c2`, and `c3`, the following join compares corresponding columns from the two tables:

```
a LEFT JOIN b USING (c1, c2, c3)
```

- The `NATURAL [LEFT] JOIN` of two tables is defined to be semantically equivalent to an `INNER JOIN` or a `LEFT JOIN` with a `USING` clause that names all columns that exist in both tables.
- `RIGHT JOIN` works analogously to `LEFT JOIN`. To keep code portable across databases, it is recommended that you use `LEFT JOIN` instead of `RIGHT JOIN`.
- The `{ OJ ... }` syntax shown in the join syntax description exists only for compatibility with ODBC. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

```
SELECT left_tbl.*
    FROM { OJ left_tbl LEFT OUTER JOIN right_tbl
           ON left_tbl.id = right_tbl.id }
    WHERE right_tbl.id IS NULL;
```

  You can use other types of joins within `{ OJ ... }`, such as `INNER JOIN` or `RIGHT OUTER JOIN`. This helps with compatibility with some third-party applications, but is not official ODBC syntax.
- `STRAIGHT_JOIN` is similar to `JOIN`, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer processes the tables in a suboptimal order.

Some join examples:

```
SELECT * FROM table1, table2;

SELECT * FROM table1 INNER JOIN table2 ON table1.id = table2.id;

SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.id;
```

```
SELECT * FROM table1 LEFT JOIN table2 USING (id);

SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.id
  LEFT JOIN table3 ON table2.id = table3.id;
```

Natural joins and joins with `USING`, including outer join variants, are processed according to the SQL:2003 standard:

- Redundant columns of a `NATURAL` join do not appear. Consider this set of statements:

```
CREATE TABLE t1 (i INT, j INT);
CREATE TABLE t2 (k INT, j INT);
INSERT INTO t1 VALUES(1, 1);
INSERT INTO t2 VALUES(1, 1);
SELECT * FROM t1 NATURAL JOIN t2;
SELECT * FROM t1 JOIN t2 USING (j);
```

  In the first `SELECT` statement, column `j` appears in both tables and thus becomes a join column, so, according to standard SQL, it should appear only once in the output, not twice. Similarly, in the second SELECT statement, column `j` is named in the `USING` clause and should appear only once in the output, not twice.

  Thus, the statements produce this output:

```
+------+------+------+
| j    | i    | k    |
+------+------+------+
|    1 |    1 |    1 |
+------+------+------+
+------+------+------+
| j    | i    | k    |
+------+------+------+
|    1 |    1 |    1 |
+------+------+------+
```

  Redundant column elimination and column ordering occurs according to standard SQL, producing this display order:

  - First, coalesced common columns of the two joined tables, in the order in which they occur in the first table
  - Second, columns unique to the first table, in order in which they occur in that table
  - Third, columns unique to the second table, in order in which they occur in that table

  The single result column that replaces two common columns is defined using the coalesce operation. That is, for two `t1.a` and `t2.a` the resulting single join column `a` is defined as `a = COALESCE(t1.a, t2.a)`, where:

```
COALESCE(x, y) = (CASE WHEN x IS NOT NULL THEN x ELSE y END)
```

  If the join operation is any other join, the result columns of the join consist of the concatenation of all columns of the joined tables.

  A consequence of the definition of coalesced columns is that, for outer joins, the coalesced column contains the value of the non-`NULL` column if one of the two columns is always `NULL`. If neither or both columns are `NULL`, both common columns have the same value, so it doesn't matter which one

is chosen as the value of the coalesced column. A simple way to interpret this is to consider that a coalesced column of an outer join is represented by the common column of the inner table of a `JOIN`. Suppose that the tables `t1(a, b)` and `t2(a, c)` have the following contents:

```
t1    t2
----  ----
1 x   2 z
2 y   3 w
```

Then, for this join, column `a` contains the values of `t1.a`:

```
mysql> SELECT * FROM t1 NATURAL LEFT JOIN t2;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    1 | x    | NULL |
|    2 | y    | z    |
+------+------+------+
```

By contrast, for this join, column `a` contains the values of `t2.a`.

```
mysql> SELECT * FROM t1 NATURAL RIGHT JOIN t2;
+------+------+------+
| a    | c    | b    |
+------+------+------+
|    2 | z    | y    |
|    3 | w    | NULL |
+------+------+------+
```

Compare those results to the otherwise equivalent queries with `JOIN ... ON`:

```
mysql> SELECT * FROM t1 LEFT JOIN t2 ON (t1.a = t2.a);
+------+------+------+------+
| a    | b    | a    | c    |
+------+------+------+------+
|    1 | x    | NULL | NULL |
|    2 | y    |    2 | z    |
+------+------+------+------+
```

```
mysql> SELECT * FROM t1 RIGHT JOIN t2 ON (t1.a = t2.a);
+------+------+------+------+
| a    | b    | a    | c    |
+------+------+------+------+
|    2 | y    |    2 | z    |
| NULL | NULL |    3 | w    |
+------+------+------+------+
```

- A `USING` clause can be rewritten as an `ON` clause that compares corresponding columns. However, although `USING` and `ON` are similar, they are not quite the same. Consider the following two queries:

```
a LEFT JOIN b USING (c1, c2, c3)
a LEFT JOIN b ON a.c1 = b.c1 AND a.c2 = b.c2 AND a.c3 = b.c3
```

With respect to determining which rows satisfy the join condition, both joins are semantically identical.

With respect to determining which columns to display for `SELECT *` expansion, the two joins are not semantically identical. The `USING` join selects the coalesced value of corresponding columns, whereas the `ON` join selects all columns from all tables. For the `USING` join, `SELECT *` selects these values:

```
COALESCE(a.c1, b.c1), COALESCE(a.c2, b.c2), COALESCE(a.c3, b.c3)
```

For the `ON` join, `SELECT *` selects these values:

```
a.c1, a.c2, a.c3, b.c1, b.c2, b.c3
```

With an inner join, `COALESCE(a.c1, b.c1)` is the same as either `a.c1` or `b.c1` because both columns will have the same value. With an outer join (such as `LEFT JOIN`), one of the two columns can be `NULL`. That column is omitted from the result.

- An `ON` clause can refer only to its operands.

  Example:

  ```
  CREATE TABLE t1 (i1 INT);
  CREATE TABLE t2 (i2 INT);
  CREATE TABLE t3 (i3 INT);
  SELECT * FROM t1 JOIN t2 ON (i1 = i3) JOIN t3;
  ```

  The statement fails with an `Unknown column 'i3' in 'on clause'` error because `i3` is a column in `t3`, which is not an operand of the `ON` clause. To enable the join to be processed, rewrite the statement as follows:

  ```
  SELECT * FROM t1 JOIN t2 JOIN t3 ON (i1 = i3);
  ```

- `JOIN` has higher precedence than the comma operator (`,`), so the join expression `t1, t2 JOIN t3` is interpreted as `(t1, (t2 JOIN t3))`, not as `((t1, t2) JOIN t3)`. This affects statements that use an `ON` clause because that clause can refer only to columns in the operands of the join, and the precedence affects interpretation of what those operands are.

  Example:

  ```
  CREATE TABLE t1 (i1 INT, j1 INT);
  CREATE TABLE t2 (i2 INT, j2 INT);
  CREATE TABLE t3 (i3 INT, j3 INT);
  INSERT INTO t1 VALUES(1, 1);
  INSERT INTO t2 VALUES(1, 1);
  INSERT INTO t3 VALUES(1, 1);
  SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);
  ```

  The `JOIN` takes precedence over the comma operator, so the operands for the `ON` clause are `t2` and `t3`. Because `t1.i1` is not a column in either of the operands, the result is an `Unknown column 't1.i1' in 'on clause'` error.

  To enable the join to be processed, use either of these strategies:

  - Group the first two tables explicitly with parentheses so that the operands for the `ON` clause are `(t1, t2)` and `t3`:

    ```
    SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);
    ```

  - Avoid the use of the comma operator and use `JOIN` instead:

    ```
    ```

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (t1.i1 = t3.i3);
```

The same precedence interpretation also applies to statements that mix the comma operator with `INNER JOIN`, `CROSS JOIN`, `LEFT JOIN`, and `RIGHT JOIN`, all of which have higher precedence than the comma operator.

- A MySQL extension compared to the SQL:2003 standard is that MySQL permits you to qualify the common (coalesced) columns of `NATURAL` or `USING` joins, whereas the standard disallows that.

13.2.10.3 UNION Syntax

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

`UNION` is used to combine the result from multiple `SELECT` statements into a single result set.

The column names from the first `SELECT` statement are used as the column names for the results returned. Selected columns listed in corresponding positions of each `SELECT` statement should have the same data type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.)

If the data types of corresponding `SELECT` columns do not match, the types and lengths of the columns in the `UNION` result take into account the values retrieved by all of the `SELECT` statements. For example, consider the following:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+---------------+
| REPEAT('a',1) |
+---------------+
| a             |
| bbbbbbbbbb    |
+---------------+
```

The `SELECT` statements are normal select statements, but with the following restrictions:

- Only the last `SELECT` statement can use `INTO OUTFILE`. (However, the entire `UNION` result is written to the file.)
- `HIGH_PRIORITY` cannot be used with `SELECT` statements that are part of a `UNION`. If you specify it for the first `SELECT`, it has no effect. If you specify it for any subsequent `SELECT` statements, a syntax error results.

The default behavior for `UNION` is that duplicate rows are removed from the result. The optional `DISTINCT` keyword has no effect other than the default because it also specifies duplicate-row removal. With the optional `ALL` keyword, duplicate-row removal does not occur and the result includes all matching rows from all the `SELECT` statements.

You can mix `UNION ALL` and `UNION DISTINCT` in the same query. Mixed `UNION` types are treated such that a `DISTINCT` union overrides any `ALL` union to its left. A `DISTINCT` union can be produced explicitly by using `UNION DISTINCT` or implicitly by using `UNION` with no following `DISTINCT` or `ALL` keyword.

To apply `ORDER BY` or `LIMIT` to an individual `SELECT`, place the clause inside the parentheses that enclose the `SELECT`:

```
(SELECT a FROM t1 WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
```

```
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

However, use of `ORDER BY` for individual `SELECT` statements implies nothing about the order in which the rows appear in the final result because `UNION` by default produces an unordered set of rows. Therefore, the use of `ORDER BY` in this context is typically in conjunction with `LIMIT`, so that it is used to determine the subset of the selected rows to retrieve for the `SELECT`, even though it does not necessarily affect the order of those rows in the final `UNION` result. If `ORDER BY` appears without `LIMIT` in a `SELECT`, it is optimized away because it will have no effect anyway.

To use an `ORDER BY` or `LIMIT` clause to sort or limit the entire `UNION` result, parenthesize the individual `SELECT` statements and place the `ORDER BY` or `LIMIT` after the last one. The following example uses both clauses:

```
(SELECT a FROM t1 WHERE a=10 AND B=1)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

A statement without parentheses is equivalent to one parenthesized as just shown.

This kind of `ORDER BY` cannot use column references that include a table name (that is, names in *`tbl_name.col_name`* format). Instead, provide a column alias in the first `SELECT` statement and refer to the alias in the `ORDER BY`. (Alternatively, refer to the column in the `ORDER BY` using its column position. However, use of column positions is deprecated.)

Also, if a column to be sorted is aliased, the `ORDER BY` clause *must* refer to the alias, not the column name. The first of the following statements will work, but the second will fail with an `Unknown column 'a' in 'order clause'` error:

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY b;
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY a;
```

To cause rows in a `UNION` result to consist of the sets of rows retrieved by each `SELECT` one after the other, select an additional column in each `SELECT` to use as a sort column and add an `ORDER BY` following the last `SELECT`:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col;
```

To additionally maintain sort order within individual `SELECT` results, add a secondary column to the `ORDER BY` clause:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col, col1a;
```

Use of an additional column also enables you to determine which `SELECT` each row comes from. Extra columns can provide other identifying information as well, such as a string that indicates a table name.

`UNION` queries with an aggregate function in an `ORDER BY` clause are rejected with an `ER_AGGREGATE_ORDER_FOR_UNION` error. Example:

```
```

```
SELECT 1 AS foo UNION SELECT 2 ORDER BY MAX(1);
```

13.2.11 Subquery Syntax

A subquery is a `SELECT` statement within another statement.

All subquery forms and operations that the SQL standard requires are supported, as well as a few features that are MySQL-specific.

Here is an example of a subquery:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

In this example, `SELECT * FROM t1 ...` is the *outer query* (or *outer statement*), and `(SELECT column1 FROM t2)` is the *subquery*. We say that the subquery is *nested* within the outer query, and in fact it is possible to nest subqueries within other subqueries, to a considerable depth. A subquery must always appear within parentheses.

The main advantages of subqueries are:

- They allow queries that are *structured* so that it is possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- Many people find subqueries more readable than complex joins or unions. Indeed, it was the innovation of subqueries that gave people the original idea of calling the early SQL “Structured Query Language.”

Here is an example statement that shows the major points about subquery syntax as specified by the SQL standard and supported in MySQL:

```
DELETE FROM t1
WHERE s11 > ANY
 (SELECT COUNT(*) /* no hint */ FROM t2
  WHERE NOT EXISTS
   (SELECT * FROM t3
    WHERE ROW(5*t2.s1,77)=
     (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
      (SELECT * FROM t5) AS t5)));
```

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries. Subqueries that return a particular kind of result often can be used only in certain contexts, as described in the following sections.

There are few restrictions on the type of statements in which subqueries can be used. A subquery can contain many of the keywords or clauses that an ordinary `SELECT` can contain: `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT`, joins, index hints, `UNION` constructs, comments, functions, and so on.

A subquery's outer statement can be any one of: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET`, or `DO`.

In MySQL, you cannot modify a table and select from the same table in a subquery. This applies to statements such as `DELETE`, `INSERT`, `REPLACE`, `UPDATE`, and (because subqueries can be used in the `SET` clause) `LOAD DATA`.

For information about how the optimizer handles subqueries, see Section 8.2.2, “Optimizing Subqueries, Derived Tables, View References, and Common Table Expressions”. For a discussion of restrictions on

subquery use, including performance issues for certain forms of subquery syntax, see Section 13.2.11.13, “Restrictions on Subqueries”.

13.2.11.1 The Subquery as Scalar Operand

In its simplest form, a subquery is a scalar subquery that returns a single value. A scalar subquery is a simple operand, and you can use it almost anywhere a single column value or literal is legal, and you can expect it to have those characteristics that all operands have: a data type, a length, an indication that it can be `NULL`, and so on. For example:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

The subquery in this `SELECT` returns a single value (`'abcde'`) that has a data type of `CHAR`, a length of 5, a character set and collation equal to the defaults in effect at `CREATE TABLE` time, and an indication that the value in the column can be `NULL`. Nullability of the value selected by a scalar subquery is not copied because if the subquery result is empty, the result is `NULL`. For the subquery just shown, if `t1` were empty, the result would be `NULL` even though `s2` is `NOT NULL`.

There are a few contexts in which a scalar subquery cannot be used. If a statement permits only a literal value, you cannot use a subquery. For example, `LIMIT` requires literal integer arguments, and `LOAD DATA` requires a literal string file name. You cannot use subqueries to supply these values.

When you see examples in the following sections that contain the rather spartan construct `(SELECT column1 FROM t1)`, imagine that your own code contains much more diverse and complex constructions.

Suppose that we make two tables:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

Then perform a `SELECT`:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

The result is `2` because there is a row in `t2` containing a column `s1` that has a value of `2`.

A scalar subquery can be part of an expression, but remember the parentheses, even if the subquery is an operand that provides an argument for a function. For example:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

13.2.11.2 Comparisons Using Subqueries

The most common use of a subquery is in the form:

```
non_subquery_operand comparison_operator (subquery)
```

Where *`comparison_operator`* is one of these operators:

```
=  >  <  >=  <=  <>  !=  <=>
```

For example:

```
... WHERE 'a' = (SELECT column1 FROM t1)
```

MySQL also permits this construct:

```
non_subquery_operand LIKE (subquery)
```

At one time the only legal place for a subquery was on the right side of a comparison, and you might still find some old DBMSs that insist on this.

Here is an example of a common-form subquery comparison that you cannot do with a join. It finds all the rows in table `t1` for which the `column1` value is equal to a maximum value in table `t2`:

```
SELECT * FROM t1
  WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Here is another example, which again is impossible with a join because it involves aggregating for one of the tables. It finds all rows in table `t1` containing a value that occurs twice in a given column:

```
SELECT * FROM t1 AS t
  WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

For a comparison of the subquery to a scalar, the subquery must return a scalar. For a comparison of the subquery to a row constructor, the subquery must be a row subquery that returns a row with the same number of values as the row constructor. See Section 13.2.11.5, “Row Subqueries”.

13.2.11.3 Subqueries with ANY, IN, or SOME

Syntax:

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

Where *`comparison_operator`* is one of these operators:

```
=  >  <  >=  <=  <>  !=
```

The `ANY` keyword, which must follow a comparison operator, means “return `TRUE` if the comparison is `TRUE` for `ANY` of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is `TRUE` if table `t2` contains `(21,14,7)` because there is a value `7` in `t2` that is less than `10`. The expression is `FALSE` if table `t2` contains `(20,10)`, or if table `t2` is empty. The expression is *unknown* (that is, `NULL`) if table `t2` contains `(NULL,NULL,NULL)`.

When used with a subquery, the word `IN` is an alias for `= ANY`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

`IN` and `= ANY` are not synonyms when used with an expression list. `IN` can take an expression list, but `= ANY` cannot. See Section 12.3.2, “Comparison Functions and Operators”.

`NOT IN` is not an alias for `<> ANY`, but for `<> ALL`. See Section 13.2.11.4, “Subqueries with ALL”.

The word `SOME` is an alias for `ANY`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Use of the word `SOME` is rare, but this example shows why it might be useful. To most people, the English phrase “a is not equal to any b” means “there is no b which is equal to a,” but that is not what is meant by the SQL syntax. The syntax means “there is some b to which a is not equal.” Using `<> SOME` instead helps ensure that everyone understands the true meaning of the query.

13.2.11.4 Subqueries with ALL

Syntax:

```
operand comparison_operator ALL (subquery)
```

The word `ALL`, which must follow a comparison operator, means “return `TRUE` if the comparison is `TRUE` for `ALL` of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is `TRUE` if table `t2` contains `(-5,0,+5)` because `10` is greater than all three values in `t2`. The expression is `FALSE` if table `t2` contains `(12,6,NULL,-100)` because there is a single value `12` in table `t2` that is greater than `10`. The expression is *unknown* (that is, `NULL`) if table `t2` contains `(0,NULL,1)`.

Finally, the expression is `TRUE` if table `t2` is empty. So, the following expression is `TRUE` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

But this expression is `NULL` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

In addition, the following expression is `NULL` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

In general, *tables containing `NULL` values* and *empty tables* are “edge cases.” When writing subqueries, always consider whether you have taken those two possibilities into account.

`NOT IN` is an alias for `<> ALL`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

13.2.11.5 Row Subqueries

Scalar or column subqueries return a single value or a column of values. A *row subquery* is a subquery variant that returns a single row and can thus return more than one column value. Legal operators for row subquery comparisons are:

```
=  >  <  >=  <=  <>  !=  <=>
```

Here are two examples:

```
SELECT * FROM t1
  WHERE (col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
SELECT * FROM t1
  WHERE ROW(col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
```

For both queries, if the table `t2` contains a single row with `id = 10`, the subquery returns a single row. If this row has `col3` and `col4` values equal to the `col1` and `col2` values of any rows in `t1`, the `WHERE` expression is `TRUE` and each query returns those `t1` rows. If the `t2` row `col3` and `col4` values are not equal the `col1` and `col2` values of any `t1` row, the expression is `FALSE` and the query returns an empty result set. The expression is *unknown* (that is, `NULL`) if the subquery produces no rows. An error occurs if the subquery produces multiple rows because a row subquery can return at most one row.

For information about how each operator works for row comparisons, see Section 12.3.2, “Comparison Functions and Operators”.

The expressions `(1,2)` and `ROW(1,2)` are sometimes called *row constructors*. The two are equivalent. The row constructor and the row returned by the subquery must contain the same number of values.

A row constructor is used for comparisons with subqueries that return two or more columns. When a subquery returns a single column, this is regarded as a scalar value and not as a row, so a row constructor cannot be used with a subquery that does not return at least two columns. Thus, the following query fails with a syntax error:

```
SELECT * FROM t1 WHERE ROW(1) = (SELECT column1 FROM t2)
```

Row constructors are legal in other contexts. For example, the following two statements are semantically equivalent (and are handled in the same way by the optimizer):

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

The following query answers the request, “find all rows in table `t1` that also exist in table `t2`”:

```
SELECT column1,column2,column3
  FROM t1
  WHERE (column1,column2,column3) IN
         (SELECT column1,column2,column3 FROM t2);
```

For more information about the optimizer and row constructors, see Section 8.2.1.22, “Row Constructor Expression Optimization”

13.2.11.6 Subqueries with EXISTS or NOT EXISTS

If a subquery returns any rows at all, `EXISTS subquery` is `TRUE`, and `NOT EXISTS subquery` is `FALSE`. For example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Traditionally, an `EXISTS` subquery starts with `SELECT *`, but it could begin with `SELECT 5` or `SELECT column1` or anything at all. MySQL ignores the `SELECT` list in such a subquery, so it makes no difference.

For the preceding example, if `t2` contains any rows, even rows with nothing but `NULL` values, the `EXISTS` condition is `TRUE`. This is actually an unlikely example because a `[NOT] EXISTS` subquery almost always contains correlations. Here are some more realistic examples:

- What kind of store is present in one or more cities?

```
SELECT DISTINCT store_type FROM stores
  WHERE EXISTS (SELECT * FROM cities_stores
                WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in no cities?

```
SELECT DISTINCT store_type FROM stores
  WHERE NOT EXISTS (SELECT * FROM cities_stores
                    WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in all cities?

```
SELECT DISTINCT store_type FROM stores s1
  WHERE NOT EXISTS (
    SELECT * FROM cities WHERE NOT EXISTS (
      SELECT * FROM cities_stores
       WHERE cities_stores.city = cities.city
       AND cities_stores.store_type = stores.store_type));
```

The last example is a double-nested `NOT EXISTS` query. That is, it has a `NOT EXISTS` clause within a `NOT EXISTS` clause. Formally, it answers the question “does a city exist with a store that is not in `Stores`”? But it is easier to say that a nested `NOT EXISTS` answers the question “is *x* `TRUE` for all *y*?”

13.2.11.7 Correlated Subqueries

A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query. For example:

```
SELECT * FROM t1
  WHERE column1 = ANY (SELECT column1 FROM t2
                       WHERE t2.column2 = t1.column2);
```

Notice that the subquery contains a reference to a column of `t1`, even though the subquery's `FROM` clause does not mention a table `t1`. So, MySQL looks outside the subquery, and finds `t1` in the outer query.

Suppose that table `t1` contains a row where `column1 = 5` and `column2 = 6`; meanwhile, table `t2` contains a row where `column1 = 5` and `column2 = 7`. The simple expression `... WHERE column1 = ANY (SELECT column1 FROM t2)` would be `TRUE`, but in this example, the `WHERE` clause within the subquery is `FALSE` (because `(5,6)` is not equal to `(5,7)`), so the expression as a whole is `FALSE`.

Scoping rule: MySQL evaluates from inside to outside. For example:

```
SELECT column1 FROM t1 AS x
  WHERE x.column1 = (SELECT column1 FROM t2 AS x
    WHERE x.column1 = (SELECT column1 FROM t3
      WHERE x.column2 = t3.column1));
```

In this statement, `x.column2` must be a column in table `t2` because `SELECT column1 FROM t2 AS x ...` renames `t2`. It is not a column in table `t1` because `SELECT column1 FROM t1 ...` is an outer query that is *farther out*.

For subqueries in `HAVING` or `ORDER BY` clauses, MySQL also looks for column names in the outer select list.

For certain cases, a correlated subquery is optimized. For example:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

Otherwise, they are inefficient and likely to be slow. Rewriting the query as a join might improve performance.

Aggregate functions in correlated subqueries may contain outer references, provided the function contains nothing but outer references, and provided the function is not contained in another function or expression.

13.2.11.8 Derived Tables

This section discusses general characteristics of derived tables. For information about lateral derived tables preceded by the `LATERAL` keyword, see Section 13.2.11.9, “Lateral Derived Tables”.

A derived table is an expression that generates a table within the scope of a query `FROM` clause. For example, a subquery in a `SELECT` statement `FROM` clause is a derived table:

```
SELECT ... FROM (subquery) [AS] tbl_name ...
```

The `JSON_TABLE()` function generates a table and provides another way to create a derived table:

```
SELECT * FROM JSON_TABLE(arg_list) [AS] tbl_name ...
```

The `[AS] tbl_name` clause is mandatory because every table in a `FROM` clause must have a name. Any columns in the derived table must have unique names. Alternatively, *`tbl_name`* may be followed by a parenthesized list of names for the derived table columns:

```
SELECT ... FROM (subquery) [AS] tbl_name (col_list) ...
```

The number of column names must be the same as the number of table columns.

For the sake of illustration, assume that you have this table:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Here is how to use a subquery in the `FROM` clause, using the example table:

```
INSERT INTO t1 VALUES (1,'1',1.0);
INSERT INTO t1 VALUES (2,'2',2.0);
SELECT sb1,sb2,sb3
  FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
  WHERE sb1 > 1;
```

Result:

```
+------+------+------+
```

```
| sb1  | sb2  | sb3  |
+------+------+------+
|    2 | 2    |    4 |
+------+------+------+
```

Here is another example: Suppose that you want to know the average of a set of sums for a grouped table. This does not work:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

However, this query provides the desired information:

```
SELECT AVG(sum_column1)
  FROM (SELECT SUM(column1) AS sum_column1
        FROM t1 GROUP BY column1) AS t1;
```

Notice that the column name used within the subquery (`sum_column1`) is recognized in the outer query.

The column names for a derived table come from its select list:

```
mysql> SELECT * FROM (SELECT 1, 2, 3, 4) AS dt;
+---+---+---+---+
| 1 | 2 | 3 | 4 |
+---+---+---+---+
| 1 | 2 | 3 | 4 |
+---+---+---+---+
```

To provide column names explicitly, follow the derived table name with a parenthesized list of column names:

```
mysql> SELECT * FROM (SELECT 1, 2, 3, 4) AS dt (a, b, c, d);
+---+---+---+---+
| a | b | c | d |
+---+---+---+---+
| 1 | 2 | 3 | 4 |
+---+---+---+---+
```

A derived table can return a scalar, column, row, or table.

Derived tables are subject to these restrictions:

- A derived table cannot be a correlated subquery.
- A derived table cannot contain references to other tables of the same `SELECT`.
- Prior to MySQL 8.0.14, a derived table cannot contain outer references. This is a MySQL restriction that is lifted in MySQL 8.0.14, not a restriction of the SQL standard. For example, the derived table `dt` in the following query contains a reference `t1.b` to the table `t1` in the outer query:

  ```
  SELECT * FROM t1
  WHERE t1.d > (SELECT AVG(dt.a)
                  FROM (SELECT SUM(t2.a) AS a
                        FROM t2
                        WHERE t2.b = t1.b GROUP BY t2.c) dt
                WHERE dt.a > 10);
  ```

  The query is valid in MySQL 8.0.14 and higher. Before 8.0.14, it produces an error: `Unknown column 't1.b' in 'where clause'`

The optimizer determines information about derived tables in such a way that `EXPLAIN` does not need to materialize them. See Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”.

It is possible under certain circumstances that using `EXPLAIN SELECT` will modify table data. This can occur if the outer query accesses any tables and an inner query invokes a stored function that changes one or more rows of a table. Suppose that there are two tables `t1` and `t2` in database `d1`, and a stored function `f1` that modifies `t2`, created as shown here:

```
CREATE DATABASE d1;
USE d1;
CREATE TABLE t1 (c1 INT);
CREATE TABLE t2 (c1 INT);
CREATE FUNCTION f1(p1 INT) RETURNS INT
  BEGIN
    INSERT INTO t2 VALUES (p1);
    RETURN p1;
  END;
```

Referencing the function directly in an `EXPLAIN SELECT` has no effect on `t2`, as shown here:

```
mysql> SELECT * FROM t2;
Empty set (0.02 sec)

mysql> EXPLAIN SELECT f1(5)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: NULL
   partitions: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: No tables used
1 row in set (0.01 sec)

mysql> SELECT * FROM t2;
Empty set (0.01 sec)
```

This is because the `SELECT` statement did not reference any tables, as can be seen in the `table` and `Extra` columns of the output. This is also true of the following nested `SELECT`:

```
mysql> EXPLAIN SELECT NOW() AS a1, (SELECT f1(5)) AS a2\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: No tables used
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
```

```
+-------+------+-----------------------------------------+
| Level | Code | Message                                 |
+-------+------+-----------------------------------------+
| Note  | 1249 | Select 2 was reduced during optimization |
+-------+------+-----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

However, if the outer `SELECT` references any tables, the optimizer executes the statement in the subquery as well, with the result that `t2` is modified:

```
mysql> EXPLAIN SELECT * FROM t1 AS a1, (SELECT f1(5)) AS a2\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: <derived2>
   partitions: NULL
         type: system
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: NULL
*************************** 2. row ***************************
           id: 1
  select_type: PRIMARY
        table: a1
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: NULL
*************************** 3. row ***************************
           id: 2
  select_type: DERIVED
        table: NULL
   partitions: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: No tables used
3 rows in set (0.00 sec)

mysql> SELECT * FROM t2;
+------+
| c1   |
+------+
|    5 |
+------+
1 row in set (0.00 sec)
```

This also means that an `EXPLAIN SELECT` statement such as the one shown here may take a long time to execute because the `BENCHMARK()` function is executed once for each row in `t1`:

```
EXPLAIN SELECT * FROM t1 AS a1, (SELECT BENCHMARK(1000000, MD5(NOW())));
```

13.2.11.9 Lateral Derived Tables

A derived table cannot normally refer to (depend on) columns of preceding tables in the same `FROM` clause. As of MySQL 8.0.14, a derived table may be defined as a lateral derived table to specify that such references are permitted.

Nonlateral derived tables are specified using the syntax discussed in Section 13.2.11.8, “Derived Tables”. The syntax for a lateral derived table is the same as for a nonlateral derived table except that the keyword `LATERAL` is specified before the derived table specification. The `LATERAL` keyword must precede each table to be used as a lateral derived table.

Lateral derived tables are subject to these restrictions:

- A lateral derived table can occur only in a `FROM` clause, either in a list of tables separated with commas or in a join specification (`JOIN`, `INNER JOIN`, `CROSS JOIN`, `LEFT [OUTER] JOIN`, or `RIGHT [OUTER] JOIN`).
- If a lateral derived table is in the right operand of a join clause and contains a reference to the left operand, the join operation must be an `INNER JOIN`, `CROSS JOIN`, or `LEFT [OUTER] JOIN`.

  If the table is in the left operand and contains a reference to the right operand, the join operation must be an `INNER JOIN`, `CROSS JOIN`, or `RIGHT [OUTER] JOIN`.
- If a lateral derived table references an aggregate function, the function's aggregation query cannot be the one that owns the `FROM` clause in which the lateral derived table occurs.
- Per the SQL standard, a table function has an implicit `LATERAL`, so it behaves as in MySQL 8.0 versions prior to 8.0.14. However, per the standard, the `LATERAL` word is not allowed before `JSON_TABLE()`, even though it is implicit.

The following discussion shows how lateral derived tables make possible certain SQL operations that cannot be done with nonlateral derived tables or that require less-efficient workarounds.

Suppose that we want to solve this problem: Given a table of people in a sales force (where each row describes a member of the sales force), and a table of all sales (where each row describes a sale: salesperson, customer, amount, date), determine the size and customer of the largest sale for each salesperson. This problem can be approached two ways.

First approach to solving the problem: For each salesperson, calculate the maximum sale size, and also find the customer who provided this maximum. In MySQL, that can be done like this:

```
SELECT
  salesperson.name,
  -- find maximum sale size for this salesperson
  (SELECT MAX(amount) AS amount
    FROM all_sales
    WHERE all_sales.salesperson_id = salesperson.id)
  AS amount,
  -- find customer for this maximum size
  (SELECT customer_name
    FROM all_sales
    WHERE all_sales.salesperson_id = salesperson.id
    AND all_sales.amount =
         -- find maximum size, again
         (SELECT MAX(amount) AS amount
```

```
          FROM all_sales
          WHERE all_sales.salesperson_id = salesperson.id))
  AS customer_name
FROM
  salesperson;
```

That query is inefficient because it calculates the maximum size twice per salesperson (once in the first subquery and once in the second).

We can try to achieve an efficiency gain by calculating the maximum once per salesperson and “caching” it in a derived table, as shown by this modified query:

```
SELECT
  salesperson.name,
  max_sale.amount,
  max_sale_customer.customer_name
FROM
  salesperson,
  -- calculate maximum size, cache it in transient derived table max_sale
  (SELECT MAX(amount) AS amount
    FROM all_sales
    WHERE all_sales.salesperson_id = salesperson.id)
  AS max_sale,
  -- find customer, reusing cached maximum size
  (SELECT customer_name
    FROM all_sales
    WHERE all_sales.salesperson_id = salesperson.id
    AND all_sales.amount =
        -- the cached maximum size
        max_sale.amount)
  AS max_sale_customer;
```

However, the query is illegal in SQL-92 because derived tables cannot depend on other tables in the same `FROM` clause. Derived tables must be constant over the query's duration, not contain references to columns of other `FROM` clause tables. As written, the query produces this error:

```
ERROR 1054 (42S22): Unknown column 'salesperson.id' in 'where clause'
```

In SQL:1999, the query becomes legal if the derived tables are preceded by the `LATERAL` keyword (which means “this derived table depends on previous tables on its left side”):

```
SELECT
  salesperson.name,
  max_sale.amount,
  max_sale_customer.customer_name
FROM
  salesperson,
  -- calculate maximum size, cache it in transient derived table max_sale
  LATERAL
  (SELECT MAX(amount) AS amount
    FROM all_sales
    WHERE all_sales.salesperson_id = salesperson.id)
  AS max_sale,
  -- find customer, reusing cached maximum size
  LATERAL
  (SELECT customer_name
    FROM all_sales
    WHERE all_sales.salesperson_id = salesperson.id
    AND all_sales.amount =
        -- the cached maximum size
        max_sale.amount)
  AS max_sale_customer;
```

A lateral derived table need not be constant and is brought up to date each time a new row from a preceding table on which it depends is processed by the top query.

Second approach to solving the problem: A different solution could be used if a subquery in the `SELECT` list could return multiple columns:

```
SELECT
  salesperson.name,
  -- find maximum size and customer at same time
  (SELECT amount, customer_name
    FROM all_sales
    WHERE all_sales.salesperson_id = salesperson.id
    ORDER BY amount DESC LIMIT 1)
FROM
  salesperson;
```

That is efficient but illegal. It does not work because such subqueries can return only a single column:

```
ERROR 1241 (21000): Operand should contain 1 column(s)
```

One attempt at rewriting the query is to select multiple columns from a derived table:

```
SELECT
  salesperson.name,
  max_sale.amount,
  max_sale.customer_name
FROM
  salesperson,
  -- find maximum size and customer at same time
  (SELECT amount, customer_name
    FROM all_sales
    WHERE all_sales.salesperson_id = salesperson.id
    ORDER BY amount DESC LIMIT 1)
  AS max_sale;
```

However, that also does not work. The derived table is dependent on the `salesperson` table and thus fails without `LATERAL`:

```
ERROR 1054 (42S22): Unknown column 'salesperson.id' in 'where clause'
```

Adding the `LATERAL` keyword makes the query legal:

```
SELECT
  salesperson.name,
  max_sale.amount,
  max_sale.customer_name
FROM
  salesperson,
  -- find maximum size and customer at same time
  LATERAL
  (SELECT amount, customer_name
    FROM all_sales
    WHERE all_sales.salesperson_id = salesperson.id
    ORDER BY amount DESC LIMIT 1)
  AS max_sale;
```

In short, `LATERAL` is the efficient solution to all drawbacks in the two approaches just discussed.

13.2.11.10 Subquery Errors

There are some errors that apply only to subqueries. This section describes them.

- Unsupported subquery syntax:

```
ERROR 1235 (ER_NOT_SUPPORTED_YET)
SQLSTATE = 42000
Message = "This version of MySQL doesn't yet support
'LIMIT & IN/ALL/ANY/SOME subquery'"
```

  This means that MySQL does not support statements of the following form:

```
SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
```

- Incorrect number of columns from subquery:

```
ERROR 1241 (ER_OPERAND_COL)
SQLSTATE = 21000
Message = "Operand should contain 1 column(s)"
```

  This error occurs in cases like this:

```
SELECT (SELECT column1, column2 FROM t2) FROM t1;
```

  You may use a subquery that returns multiple columns, if the purpose is row comparison. In other contexts, the subquery must be a scalar operand. See Section 13.2.11.5, “Row Subqueries”.

- Incorrect number of rows from subquery:

```
ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
SQLSTATE = 21000
Message = "Subquery returns more than 1 row"
```

  This error occurs for statements where the subquery must return at most one row but returns multiple rows. Consider the following example:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

  If `SELECT column1 FROM t2` returns just one row, the previous query will work. If the subquery returns more than one row, error 1242 will occur. In that case, the query should be rewritten as:

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- Incorrectly used table in subquery:

```
Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"
```

  This error occurs in cases such as the following, which attempts to modify a table and select from the same table in the subquery:

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

You can use a subquery for assignment within an `UPDATE` statement because subqueries are legal in `UPDATE` and `DELETE` statements as well as in `SELECT` statements. However, you cannot use the same table (in this case, table `t1`) for both the subquery `FROM` clause and the update target.

For transactional storage engines, the failure of a subquery causes the entire statement to fail. For nontransactional storage engines, data modifications made before the error was encountered are preserved.

13.2.11.11 Optimizing Subqueries

Development is ongoing, so no optimization tip is reliable for the long term. The following list provides some interesting tricks that you might want to play with. See also Section 8.2.2, “Optimizing Subqueries, Derived Tables, View References, and Common Table Expressions”.

- Use subquery clauses that affect the number or order of the rows in the subquery. For example:

```
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT column1 FROM t2 ORDER BY column1);
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT DISTINCT column1 FROM t2);
SELECT * FROM t1 WHERE EXISTS
  (SELECT * FROM t2 LIMIT 1);
```

- Replace a join with a subquery. For example, try this:

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
  SELECT column1 FROM t2);
```

Instead of this:

```
SELECT DISTINCT t1.column1 FROM t1, t2
  WHERE t1.column1 = t2.column1;
```

- Some subqueries can be transformed to joins for compatibility with older versions of MySQL that do not support subqueries. However, in some cases, converting a subquery to a join may improve performance. See Section 13.2.11.12, “Rewriting Subqueries as Joins”.

- Move clauses from outside to inside the subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

Instead of this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

For another example, use this query:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

Instead of this query:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- Use a row subquery instead of a correlated subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

  Instead of this query:

```
SELECT * FROM t1
  WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
                AND t2.column2=t1.column2);
```

- Use `NOT (a = ANY (...))` rather than `a <> ALL (...)`.
- Use `x = ANY (table containing (1,2))` rather than `x=1 OR x=2`.
- Use `= ANY` rather than `EXISTS`.
- For uncorrelated subqueries that always return one row, `IN` is always slower than `=`. For example, use this query:

```
SELECT * FROM t1
  WHERE t1.col_name = (SELECT a FROM t2 WHERE b = some_const);
```

  Instead of this query:

```
SELECT * FROM t1
  WHERE t1.col_name IN (SELECT a FROM t2 WHERE b = some_const);
```

These tricks might cause programs to go faster or slower. Using MySQL facilities like the `BENCHMARK()` function, you can get an idea about what helps in your own situation. See Section 12.15, “Information Functions”.

Some optimizations that MySQL itself makes are:

- MySQL executes uncorrelated subqueries only once. Use `EXPLAIN` to make sure that a given subquery really is uncorrelated.
- MySQL rewrites `IN`, `ALL`, `ANY`, and `SOME` subqueries in an attempt to take advantage of the possibility that the select-list columns in the subquery are indexed.
- MySQL replaces subqueries of the following form with an index-lookup function, which `EXPLAIN` describes as a special join type (`unique_subquery` or `index_subquery`):

```
... IN (SELECT indexed_column FROM single_table ...)
```

- MySQL enhances expressions of the following form with an expression involving `MIN()` or `MAX()`, unless `NULL` values or empty sets are involved:

```
value {ALL|ANY|SOME} {> | < | >= | <=} (uncorrelated subquery)
```

  For example, this `WHERE` clause:

```
WHERE 5 > ALL (SELECT x FROM t)
```

  might be treated by the optimizer like this:

```
WHERE 5 > (SELECT MAX(x) FROM t)
```

See also MySQL Internals: How MySQL Transforms Subqueries.

13.2.11.12 Rewriting Subqueries as Joins

Sometimes there are other ways to test membership in a set of values than by using a subquery. Also, on some occasions, it is not only possible to rewrite a query without a subquery, but it can be more efficient to make use of some of these techniques rather than to use subqueries. One of these is the `IN()` construct:

For example, this query:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

Can be rewritten as:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

The queries:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

Can be rewritten as:

```
SELECT table1.*
  FROM table1 LEFT JOIN table2 ON table1.id=table2.id
  WHERE table2.id IS NULL;
```

A `LEFT [OUTER] JOIN` can be faster than an equivalent subquery because the server might be able to optimize it better—a fact that is not specific to MySQL Server alone. Prior to SQL-92, outer joins did not exist, so subqueries were the only way to do certain things. Today, MySQL Server and many other modern database systems offer a wide range of outer join types.

MySQL Server supports multiple-table `DELETE` statements that can be used to efficiently delete rows based on information from one table or even from many tables at the same time. Multiple-table `UPDATE` statements are also supported. See Section 13.2.2, “DELETE Syntax”, and Section 13.2.12, “UPDATE Syntax”.

13.2.11.13 Restrictions on Subqueries

- In general, you cannot modify a table and select from the same table in a subquery. For example, this limitation applies to statements of the following forms:

  ```
  DELETE FROM t WHERE ... (SELECT ... FROM t ...);
  UPDATE t ... WHERE col = (SELECT ... FROM t ...);
  {INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
  ```

  Exception: The preceding prohibition does not apply if for the modified table you are using a derived table and that derived table is materialized rather than merged into the outer query. (See Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”.) Example:

```
UPDATE t ... WHERE col = (SELECT * FROM (SELECT ... FROM t...) AS dt ...);
```

Here the result from the derived table is materialized as a temporary table, so the relevant rows in `t` have already been selected by the time the update to `t` takes place.

In general, you may be able to influence the optimizer to materialize a derived table by adding a `NO_MERGE` optimizer hint. See Section 8.9.3, “Optimizer Hints”.

- Row comparison operations are only partially supported:
  - For *`expr [NOT] IN subquery`*, *`expr`* can be an *`n`*-tuple (specified using row constructor syntax) and the subquery can return rows of *`n`*-tuples. The permitted syntax is therefore more specifically expressed as *`row_constructor [NOT] IN table_subquery`*
  - For *`expr op {ALL|ANY|SOME} subquery`*, *`expr`* must be a scalar value and the subquery must be a column subquery; it cannot return multiple-column rows.

  In other words, for a subquery that returns rows of *`n`*-tuples, this is supported:

```
(expr_1, ..., expr_n) [NOT] IN table_subquery
```

  But this is not supported:

```
(expr_1, ..., expr_n) op {ALL|ANY|SOME} subquery
```

  The reason for supporting row comparisons for `IN` but not for the others is that `IN` is implemented by rewriting it as a sequence of `=` comparisons and `AND` operations. This approach cannot be used for `ALL`, `ANY`, or `SOME`.

- Subqueries in the `FROM` clause cannot be correlated subqueries. They are materialized in whole (evaluated to produce a result set) during query execution, so they cannot be evaluated per row of the outer query. The optimizer delays materialization until the result is needed, which may permit materialization to be avoided. See Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”.

- MySQL does not support `LIMIT` in subqueries for certain subquery operators:

```
mysql> SELECT * FROM t1
       WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1);
ERROR 1235 (42000): This version of MySQL doesn't yet support
 'LIMIT & IN/ALL/ANY/SOME subquery'
```

- MySQL permits a subquery to refer to a stored function that has data-modifying side effects such as inserting rows into a table. For example, if `f()` inserts rows, the following query can modify data:

```
SELECT ... WHERE x IN (SELECT f() ...);
```

  This behavior is an extension to the SQL standard. In MySQL, it can produce nondeterministic results because `f()` might be executed a different number of times for different executions of a given query depending on how the optimizer chooses to handle it.

  For statement-based or mixed-format replication, one implication of this indeterminism is that such a query can produce different results on the master and its slaves.

13.2.12 UPDATE Syntax

`UPDATE` is a DML statement that modifies rows in a table.

An `UPDATE` statement can start with a `WITH` clause to define common table expressions accessible within the `UPDATE`. See Section 13.2.13, “WITH Syntax (Common Table Expressions)”.

Single-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET assignment_list
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]

value:
    {expr | DEFAULT}

assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET assignment_list
    [WHERE where_condition]
```

For the single-table syntax, the `UPDATE` statement updates columns of existing rows in the named table with new values. The `SET` clause indicates which columns to modify and the values they should be given. Each value can be given as an expression, or the keyword `DEFAULT` to set a column explicitly to its default value. The `WHERE` clause, if given, specifies the conditions that identify which rows to update. With no `WHERE` clause, all rows are updated. If the `ORDER BY` clause is specified, the rows are updated in the order that is specified. The `LIMIT` clause places a limit on the number of rows that can be updated.

For the multiple-table syntax, `UPDATE` updates rows in each table named in *`table_references`* that satisfy the conditions. Each matching row is updated once, even if it matches the conditions multiple times. For multiple-table syntax, `ORDER BY` and `LIMIT` cannot be used.

For partitioned tables, both the single-single and multiple-table forms of this statement support the use of a `PARTITION` option as part of a table reference. This option takes a list of one or more partitions or subpartitions (or both). Only the partitions (or subpartitions) listed are checked for matches, and a row that is not in any of these partitions or subpartitions is not updated, whether it satisfies the *`where_condition`* or not.

Note

Unlike the case when using `PARTITION` with an `INSERT` or `REPLACE` statement, an otherwise valid `UPDATE ... PARTITION` statement is considered successful even if no rows in the listed partitions (or subpartitions) match the *`where_condition`*.

For more information and examples, see Section 23.5, “Partition Selection”.

`where_condition` is an expression that evaluates to true for each row to be updated. For expression syntax, see Section 9.5, “Expressions”.

`table_references` and *`where_condition`* are specified as described in Section 13.2.10, “SELECT Syntax”.

You need the UPDATE privilege only for columns referenced in an UPDATE that are actually updated. You need only the SELECT privilege for any columns that are read but not modified.

The UPDATE statement supports the following modifiers:

- With the LOW_PRIORITY modifier, execution of the UPDATE is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).
- With the IGNORE modifier, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur on a unique key value are not updated. Rows updated to values that would cause data conversion errors are updated to the closest valid values instead. For more information, see Comparison of the IGNORE Keyword and Strict SQL Mode.

UPDATE IGNORE statements, including those having an ORDER BY clause, are flagged as unsafe for statement-based replication. (This is because the order in which the rows are updated determines which rows are ignored.) Such statements produce a warning in the error log when using statement-based mode and are written to the binary log using the row-based format when using MIXED mode. (Bug #11758262, Bug #50439) See Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”, for more information.

If you access a column from the table to be updated in an expression, UPDATE uses the current value of the column. For example, the following statement sets col1 to one more than its current value:

```
UPDATE t1 SET col1 = col1 + 1;
```

The second assignment in the following statement sets col2 to the current (updated) col1 value, not the original col1 value. The result is that col1 and col2 have the same value. This behavior differs from standard SQL.

```
UPDATE t1 SET col1 = col1 + 1, col2 = col1;
```

Single-table UPDATE assignments are generally evaluated from left to right. For multiple-table updates, there is no guarantee that assignments are carried out in any particular order.

If you set a column to the value it currently has, MySQL notices this and does not update it.

If you update a column that has been declared NOT NULL by setting to NULL, an error occurs if strict SQL mode is enabled; otherwise, the column is set to the implicit default value for the column data type and the warning count is incremented. The implicit default value is 0 for numeric types, the empty string ('') for string types, and the “zero” value for date and time types. See Section 11.7, “Data Type Default Values”.

If a generated column is updated explicitly, the only permitted value is DEFAULT. For information about generated columns, see Section 13.1.20.9, “CREATE TABLE and Generated Columns”.

UPDATE returns the number of rows that were actually changed. The mysql_info() C API function returns the number of rows that were matched and updated and the number of warnings that occurred during the UPDATE.

You can use LIMIT *row_count* to restrict the scope of the UPDATE. A LIMIT clause is a rows-matched restriction. The statement stops as soon as it has found *row_count* rows that satisfy the WHERE clause, whether or not they actually were changed.

If an UPDATE statement includes an ORDER BY clause, the rows are updated in the order specified by the clause. This can be useful in certain situations that might otherwise result in an error. Suppose that a table

`t` contains a column `id` that has a unique index. The following statement could fail with a duplicate-key error, depending on the order in which rows are updated:

```
UPDATE t SET id = id + 1;
```

For example, if the table contains 1 and 2 in the `id` column and 1 is updated to 2 before 2 is updated to 3, an error occurs. To avoid this problem, add an `ORDER BY` clause to cause the rows with larger `id` values to be updated before those with smaller values:

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

You can also perform `UPDATE` operations covering multiple tables. However, you cannot use `ORDER BY` or `LIMIT` with a multiple-table `UPDATE`. The *`table_references`* clause lists the tables involved in the join. Its syntax is described in Section 13.2.10.2, “JOIN Syntax”. Here is an example:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

The preceding example shows an inner join that uses the comma operator, but multiple-table `UPDATE` statements can use any type of join permitted in `SELECT` statements, such as `LEFT JOIN`.

If you use a multiple-table `UPDATE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, update a single table and rely on the `ON UPDATE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly. See Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”.

You cannot update a table and select from the same table in a subquery.

An `UPDATE` on a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks only those partitions containing rows that match the `UPDATE` statement `WHERE` clause, as long as none of the table partitioning columns are updated. (For storage engines such as `InnoDB` that employ row-level locking, no locking of partitions takes place.) For more information, see Partitioning and Locking.

13.2.13 WITH Syntax (Common Table Expressions)

A common table expression (CTE) is a named temporary result set that exists within the scope of a single statement and that can be referred to later within that statement, possibly multiple times. The following discussion describes how to write statements that use CTEs.

- Common Table Expression Syntax
- Recursive Common Table Expressions
- Limiting Common Table Expression Recursion
- Recursive Common Table Expression Examples
- Common Table Expressions Compared to Similar Constructs

For information about CTE optimization, see Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”.

Additional Resources

These articles contain additional information about using CTEs in MySQL, including many examples:

- MySQL 8.0 Labs: [Recursive] Common Table Expressions in MySQL (CTEs)
- MySQL 8.0 Labs: [Recursive] Common Table Expressions in MySQL (CTEs), Part Two – how to generate series
- MySQL 8.0 Labs: [Recursive] Common Table Expressions in MySQL (CTEs), Part Three – hierarchies
- MySQL 8.0.1: [Recursive] Common Table Expressions in MySQL (CTEs), Part Four – depth-first or breadth-first traversal, transitive closure, cycle avoidance

Common Table Expression Syntax

To specify common table expressions, use a `WITH` clause that has one or more comma-separated subclauses. Each subclause provides a subquery that produces a result set, and associates a name with the subquery. The following example defines CTEs named `cte1` and `cte2` in the `WITH` clause, and refers to them in the top-level `SELECT` that follows the `WITH` clause:

```
WITH
  cte1 AS (SELECT a, b FROM table1),
  cte2 AS (SELECT c, d FROM table2)
SELECT b, d FROM cte1 JOIN cte2
WHERE cte1.a = cte2.c;
```

In the statement containing the `WITH` clause, each CTE name can be referenced to access the corresponding CTE result set.

A CTE name can be referenced in other CTEs, enabling CTEs to be defined based on other CTEs.

A CTE can refer to itself to define a recursive CTE. Common applications of recursive CTEs include series generation and traversal of hierarchical or tree-structured data.

Common table expressions are an optional part of the syntax for DML statements. They are defined using a `WITH` clause:

```
with_clause:
    WITH [RECURSIVE]
        cte_name [(col_name [, col_name] ...)] AS (subquery)
        [, cte_name [(col_name [, col_name] ...)] AS (subquery)] ...
```

`cte_name` names a single common table expression and can be used as a table reference in the statement containing the `WITH` clause.

The *`subquery`* part of `AS (subquery)` is called the “subquery of the CTE” and is what produces the CTE result set. The parentheses following `AS` are required.

A common table expression is recursive if its subquery refers to its own name. The `RECURSIVE` keyword must be included if any CTE in the `WITH` clause is recursive. For more information, see Recursive Common Table Expressions.

Determination of column names for a given CTE occurs as follows:

- If a parenthesized list of names follows the CTE name, those names are the column names:

```
WITH cte (col1, col2) AS
(
  SELECT 1, 2
```

```
  UNION ALL
  SELECT 3, 4
)
SELECT col1, col2 FROM cte;
```

The number of names in the list must be the same as the number of columns in the result set.

- Otherwise, the column names come from the select list of the first `SELECT` within the `AS (subquery)` part:

```
WITH cte AS
(
  SELECT 1 AS col1, 2 AS col2
  UNION ALL
  SELECT 3, 4
)
SELECT col1, col2 FROM cte;
```

A `WITH` clause is permitted in these contexts:

- At the beginning of `SELECT`, `UPDATE`, and `DELETE` statements.

```
WITH ... SELECT ...
WITH ... UPDATE ...
WITH ... DELETE ...
```

- At the beginning of subqueries (including derived table subqueries):

```
SELECT ... WHERE id IN (WITH ... SELECT ...) ...
SELECT * FROM (WITH ... SELECT ...) AS dt ...
```

- Immediately preceding `SELECT` for statements that include a `SELECT` statement:

```
INSERT ... WITH ... SELECT ...
REPLACE ... WITH ... SELECT ...
CREATE TABLE ... WITH ... SELECT ...
CREATE VIEW ... WITH ... SELECT ...
DECLARE CURSOR ... WITH ... SELECT ...
EXPLAIN ... WITH ... SELECT ...
```

Only one `WITH` clause is permitted at the same level. `WITH` followed by `WITH` at the same level is not permitted, so this is illegal:

```
WITH cte1 AS (...) WITH cte2 AS (...) SELECT ...
```

To make the statement legal, use a single `WITH` clause that separates the subclauses by a comma:

```
WITH cte1 AS (...), cte2 AS (...) SELECT ...
```

However, a statement can contain multiple `WITH` clauses if they occur at different levels:

```
WITH cte1 AS (SELECT 1)
SELECT * FROM (WITH cte2 AS (SELECT 2) SELECT * FROM cte2 JOIN cte1) AS dt;
```

A `WITH` clause can define one or more common table expressions, but each CTE name must be unique to the clause. This is illegal:

```
WITH cte1 AS (...), cte1 AS (...) SELECT ...
```

To make the statement legal, define the CTEs with unique names:

```
WITH cte1 AS (...), cte2 AS (...) SELECT ...
```

A CTE can refer to itself or to other CTEs:

- A self-referencing CTE is recursive.
- A CTE can refer to CTEs defined earlier in the same `WITH` clause, but not those defined later.

  This constraint rules out mutually-recursive CTEs, where `cte1` references `cte2` and `cte2` references `cte1`. One of those references must be to a CTE defined later, which is not permitted.
- A CTE in a given query block can refer to CTEs defined in query blocks at a more outer level, but not CTEs defined in query blocks at a more inner level.

For resolving references to objects with the same names, derived tables hide CTEs; and CTEs hide base tables, `TEMPORARY` tables, and views. Name resolution occurs by searching for objects in the same query block, then proceeding to outer blocks in turn while no object with the name is found.

Like derived tables, a CTE cannot contain outer references prior to MySQL 8.0.14. This is a MySQL restriction that is lifted in MySQL 8.0.14, not a restriction of the SQL standard. For additional syntax considerations specific to recursive CTEs, see Recursive Common Table Expressions.

Recursive Common Table Expressions

A recursive common table expression is one having a subquery that refers to its own name. For example:

```
WITH RECURSIVE cte (n) AS
(
  SELECT 1
  UNION ALL
  SELECT n + 1 FROM cte WHERE n < 5
)
SELECT * FROM cte;
```

When executed, the statement produces this result, a single column containing a simple linear sequence:

```
+------+
| n    |
+------+
|    1 |
|    2 |
|    3 |
|    4 |
|    5 |
+------+
```

A recursive CTE has this structure:

- The `WITH` clause must begin with `WITH RECURSIVE` if any CTE in the `WITH` clause refers to itself. (If no CTE refers to itself, `RECURSIVE` is permitted but not required.)

  If you forget `RECURSIVE` for a recursive CTE, this error is a likely result:

```
ERROR 1146 (42S02): Table 'cte_name' doesn't exist
```

- The recursive CTE subquery has two parts, separated by `UNION [ALL]` or `UNION DISTINCT`:

```
SELECT ...      -- return initial row set
UNION ALL
SELECT ...      -- return additional row sets
```

  The first `SELECT` produces the initial row or rows for the CTE and does not refer to the CTE name. The second `SELECT` produces additional rows and recurses by referring to the CTE name in its `FROM` clause. Recursion ends when this part produces no new rows. Thus, a recursive CTE consists of a nonrecursive `SELECT` part followed by a recursive `SELECT` part.

  Each `SELECT` part can itself be a union of multiple `SELECT` statements.

- The types of the CTE result columns are inferred from the column types of the nonrecursive `SELECT` part only, and the columns are all nullable. For type determination, the recursive `SELECT` part is ignored.
- If the nonrecursive and recursive parts are separated by `UNION DISTINCT`, duplicate rows are eliminated. This is useful for queries that perform transitive closures, to avoid infinite loops.
- Each iteration of the recursive part operates only on the rows produced by the previous iteration. If the recursive part has multiple query blocks, iterations of each query block are scheduled in unspecified order, and each query block operates on rows that have been produced either by its previous iteration or by other query blocks since that previous iteration's end.

The recursive CTE subquery shown earlier has this nonrecursive part that retrieves a single row to produce the initial row set:

```
SELECT 1
```

The CTE subquery also has this recursive part:

```
SELECT n + 1 FROM cte WHERE n < 5
```

At each iteration, that `SELECT` produces a row with a new value one greater than the value of `n` from the previous row set. The first iteration operates on the initial row set (`1`) and produces `1+1=2`; the second iteration operates on the first iteration's row set (`2`) and produces `2+1=3`; and so forth. This continues until recursion ends, which occurs when `n` is no longer less than 5.

If the recursive part of a CTE produces wider values for a column than the nonrecursive part, it may be necessary to widen the column in the nonrecursive part to avoid data truncation. Consider this statement:

```
WITH RECURSIVE cte AS
(
  SELECT 1 AS n, 'abc' AS str
  UNION ALL
  SELECT n + 1, CONCAT(str, str) FROM cte WHERE n < 3
)
SELECT * FROM cte;
```

In nonstrict SQL mode, the statement produces this output:

```
```

```
+------+------+
| n    | str  |
+------+------+
|    1 | abc  |
|    2 | abc  |
|    3 | abc  |
+------+------+
```

The `str` column values are all `'abc'` because the nonrecursive `SELECT` determines the column widths. Consequently, the wider `str` values produced by the recursive `SELECT` are truncated.

In strict SQL mode, the statement produces an error:

```
ERROR 1406 (22001): Data too long for column 'str' at row 1
```

To address this issue, so that the statement does not produce truncation or errors, use `CAST()` in the nonrecursive `SELECT` to make the `str` column wider:

```
WITH RECURSIVE cte AS
(
  SELECT 1 AS n, CAST('abc' AS CHAR(20)) AS str
  UNION ALL
  SELECT n + 1, CONCAT(str, str) FROM cte WHERE n < 3
)
SELECT * FROM cte;
```

Now the statement produces this result, without truncation:

```
+------+--------------+
| n    | str          |
+------+--------------+
|    1 | abc          |
|    2 | abcabc       |
|    3 | abcabcabcabc |
+------+--------------+
```

Columns are accessed by name, not position, which means that columns in the recursive part can access columns in the nonrecursive part that have a different position, as this CTE illustrates:

```
WITH RECURSIVE cte AS
(
  SELECT 1 AS n, 1 AS p, -1 AS q
  UNION ALL
  SELECT n + 1, q * 2, p * 2 FROM cte WHERE n < 5
)
SELECT * FROM cte;
```

Because `p` in one row is derived from `q` in the previous row, and vice versa, the positive and negative values swap positions in each successive row of the output:

```
+------+------+------+
| n    | p    | q    |
+------+------+------+
|    1 |    1 |   -1 |
|    2 |   -2 |    2 |
|    3 |    4 |   -4 |
|    4 |   -8 |    8 |
|    5 |   16 |  -16 |
+------+------+------+
```

Some syntax constraints apply within recursive CTE subqueries:

- The recursive `SELECT` part must not contain these constructs:
  - Aggregate functions such as `SUM()`
  - Window functions
  - `GROUP BY`
  - `ORDER BY`
  - `LIMIT`
  - `DISTINCT`

  This constraint does not apply to the nonrecursive `SELECT` part of a recursive CTE. The prohibition on `DISTINCT` applies only to `UNION` members; `UNION DISTINCT` is permitted.
- The recursive `SELECT` part must reference the CTE only once and only in its `FROM` clause, not in any subquery. It can reference tables other than the CTE and join them with the CTE. If used in a join like this, the CTE must not be on the right side of a `LEFT JOIN`.

These constraints come from the SQL standard, other than the MySQL-specific exclusions of `ORDER BY`, `LIMIT`, and `DISTINCT`.

For recursive CTEs, `EXPLAIN` output rows for recursive `SELECT` parts display `Recursive` in the `Extra` column.

Cost estimates displayed by `EXPLAIN` represent cost per iteration, which might differ considerably from total cost. The optimizer cannot predict the number of iterations because it cannot predict when the `WHERE` clause will become false.

CTE actual cost may also be affected by result set size. A CTE that produces many rows may require an internal temporary table large enough to be converted from in-memory to on-disk format and may suffer a performance penalty. If so, increasing the permitted in-memory temporary table size may improve performance; see Section 8.4.4, “Internal Temporary Table Use in MySQL”.

Limiting Common Table Expression Recursion

It is important for recursive CTEs that the recursive `SELECT` part include a condition to terminate recursion. As a development technique to guard against a runaway recursive CTE, you can force termination by placing a limit on execution time:

- The `cte_max_recursion_depth` system variable enforces a limit on the number of recursion levels for CTEs. The server terminates execution of any CTE that recurses more levels than the value of this variable.
- The `max_execution_time` system variable enforces an execution timeout for `SELECT` statements executed within the current session.
- The `MAX_EXECUTION_TIME` optimizer hint enforces a per-query execution timeout for the `SELECT` statement in which it appears.

Suppose that a recursive CTE is mistakenly written with no recursion execution termination condition:

```
WITH RECURSIVE cte (n) AS
(
  SELECT 1
  UNION ALL
  SELECT n + 1 FROM cte
)
SELECT * FROM cte;
```

By default, `cte_max_recursion_depth` has a value of 1000, causing the CTE to terminate when it recurses past 1000 levels. Applications can change the session value to adjust for their requirements:

```
SET SESSION cte_max_recursion_depth = 10;      -- permit only shallow recursion
SET SESSION cte_max_recursion_depth = 1000000; -- permit deeper recursion
```

You can also set the global `cte_max_recursion_depth` value to affect all sessions that begin subsequently.

For queries that execute and thus recurse slowly or in contexts for which there is reason to set the `cte_max_recursion_depth` value very high, another way to guard against deep recursion is to set a per-session timeout. To do so, execute a statement like this prior to executing the CTE statement:

```
SET max_execution_time = 1000; -- impose one second timeout
```

Alternatively, include an optimizer hint within the CTE statement itself:

```
WITH RECURSIVE cte (n) AS
(
  SELECT 1
  UNION ALL
  SELECT n + 1 FROM cte
)
SELECT /*+ MAX_EXECUTION_TIME(1000) */ * FROM cte;
```

If a recursive query without an execution time limit enters an infinite loop, you can terminate it from another session using `KILL QUERY`. Within the session itself, the client program used to run the query might provide a way to kill the query. For example, in `mysql`, typing **Control+C** interrupts the current statement.

Recursive Common Table Expression Examples

As mentioned previously, recursive common table expressions (CTEs) are frequently used for series generation and traversing hierarchical or tree-structured data. This section shows some simple examples of these techniques.

- Fibonacci Series Generation
- Date Series Generation
- Hierarchical Data Traversal

Fibonacci Series Generation

A Fibonacci series begins with the two numbers 0 and 1 (or 1 and 1) and each number after that is the sum of the previous two numbers. A recursive common table expression can generate a Fibonacci series if each row produced by the recursive `SELECT` has access to the two previous numbers from the series. The following CTE generates a 10-number series using 0 and 1 as the first two numbers:

```
WITH RECURSIVE fibonacci (n, fib_n, next_fib_n) AS
(
  SELECT 1, 0, 1
  UNION ALL
  SELECT n + 1, next_fib_n, fib_n + next_fib_n
    FROM fibonacci WHERE n < 10
)
SELECT * FROM fibonacci;
```

The CTE produces this result:

```
+------+-------+------------+
| n    | fib_n | next_fib_n |
+------+-------+------------+
|    1 |     0 |          1 |
|    2 |     1 |          1 |
|    3 |     1 |          2 |
|    4 |     2 |          3 |
|    5 |     3 |          5 |
|    6 |     5 |          8 |
|    7 |     8 |         13 |
|    8 |    13 |         21 |
|    9 |    21 |         34 |
|   10 |    34 |         55 |
+------+-------+------------+
```

How the CTE works:

- `n` is a display column to indicate that the row contains the `n`-th Fibonacci number. For example, the 8th Fibonacci number is 13.
- The `fib_n` column displays Fibonacci number `n`.
- The `next_fib_n` column displays the next Fibonacci number after number `n`. This column provides the next series value to the next row, so that row can produce the sum of the two previous series values in its `fib_n` column.
- Recursion ends when `n` reaches 10. This is an arbitrary choice, to limit the output to a small set of rows.

The preceding output shows the entire CTE result. To select just part of it, add an appropriate `WHERE` clause to the top-level `SELECT`. For example, to select the 8th Fibonacci number, do this:

```
mysql> WITH RECURSIVE fibonacci ...
       ...
       SELECT fib_n FROM fibonacci WHERE n = 8;
+-------+
| fib_n |
+-------+
|    13 |
+-------+
```

Date Series Generation

A common table expression can generate a series of successive dates, which is useful for generating summaries that include a row for all dates in the series, including dates not represented in the summarized data.

Suppose that a table of sales numbers contains these rows:

```
mysql> SELECT * FROM sales ORDER BY date, price;
```

```
+------------+--------+
| date       | price  |
+------------+--------+
| 2017-01-03 | 100.00 |
| 2017-01-03 | 200.00 |
| 2017-01-06 |  50.00 |
| 2017-01-08 |  10.00 |
| 2017-01-08 |  20.00 |
| 2017-01-08 | 150.00 |
| 2017-01-10 |   5.00 |
+------------+--------+
```

This query summarizes the sales per day:

```
mysql> SELECT date, SUM(price) AS sum_price
       FROM sales
       GROUP BY date
       ORDER BY date;
+------------+-----------+
| date       | sum_price |
+------------+-----------+
| 2017-01-03 |    300.00 |
| 2017-01-06 |     50.00 |
| 2017-01-08 |    180.00 |
| 2017-01-10 |      5.00 |
+------------+-----------+
```

However, that result contains “holes” for dates not represented in the range of dates spanned by the table. A result that represents all dates in the range can be produced using a recursive CTE to generate that set of dates, joined with a `LEFT JOIN` to the sales data.

Here is the CTE to generate the date range series:

```
WITH RECURSIVE dates (date) AS
(
  SELECT MIN(date) FROM sales
  UNION ALL
  SELECT date + INTERVAL 1 DAY FROM dates
  WHERE date + INTERVAL 1 DAY <= (SELECT MAX(date) FROM sales)
)
SELECT * FROM dates;
```

The CTE produces this result:

```
+------------+
| date       |
+------------+
| 2017-01-03 |
| 2017-01-04 |
| 2017-01-05 |
| 2017-01-06 |
| 2017-01-07 |
| 2017-01-08 |
| 2017-01-09 |
| 2017-01-10 |
+------------+
```

How the CTE works:

- The nonrecursive `SELECT` produces the lowest date in the date range spanned by the `sales` table.
- Each row produced by the recursive `SELECT` adds one day to the date produced by the previous row.

- Recursion ends after the dates reach the highest date in the date range spanned by the `sales` table.

Joining the CTE with a `LEFT JOIN` against the `sales` table produces the sales summary with a row for each date in the range:

```
WITH RECURSIVE dates (date) AS
(
  SELECT MIN(date) FROM sales
  UNION ALL
  SELECT date + INTERVAL 1 DAY FROM dates
  WHERE date + INTERVAL 1 DAY <= (SELECT MAX(date) FROM sales)
)
SELECT dates.date, COALESCE(SUM(price), 0) AS sum_price
FROM dates LEFT JOIN sales ON dates.date = sales.date
GROUP BY dates.date
ORDER BY dates.date;
```

The output looks like this:

```
+------------+-----------+
| date       | sum_price |
+------------+-----------+
| 2017-01-03 |    300.00 |
| 2017-01-04 |      0.00 |
| 2017-01-05 |      0.00 |
| 2017-01-06 |     50.00 |
| 2017-01-07 |      0.00 |
| 2017-01-08 |    180.00 |
| 2017-01-09 |      0.00 |
| 2017-01-10 |      5.00 |
+------------+-----------+
```

Some points to note:

- Are the queries inefficient, particularly the one with the `MAX()` subquery executed for each row in the recursive `SELECT`? Checking with `EXPLAIN` shows that the subqueries are optimized away for efficiency.
- The use of `COALESCE()` avoids displaying `NULL` in the `sum_price` column on days for which no sales data occur in the `sales` table.

Hierarchical Data Traversal

Recursive common table expressions are useful for traversing data that forms a hierarchy. Consider these statements that create a small data set that shows, for each employee in a company, the employee name and ID number, and the ID of the employee's manager. The top-level employee (the CEO), has a manager ID of `NULL` (no manager).

```
CREATE TABLE employees (
  id         INT PRIMARY KEY NOT NULL,
  name       VARCHAR(100) NOT NULL,
  manager_id INT NULL,
  INDEX (manager_id),
FOREIGN KEY (manager_id) REFERENCES employees (id)
);
INSERT INTO employees VALUES
(333, "Yasmina", NULL),  # Yasmina is the CEO (manager_id is NULL)
(198, "John", 333),      # John has ID 198 and reports to 333 (Yasmina)
(692, "Tarek", 333),
(29, "Pedro", 198),
```

```
(4610, "Sarah", 29),
(72, "Pierre", 29),
(123, "Adil", 692);
```

The resulting data set looks like this:

```
mysql> SELECT * FROM employees ORDER BY id;
+------+---------+------------+
| id   | name    | manager_id |
+------+---------+------------+
|   29 | Pedro   |        198 |
|   72 | Pierre  |         29 |
|  123 | Adil    |        692 |
|  198 | John    |        333 |
|  333 | Yasmina |       NULL |
|  692 | Tarek   |        333 |
| 4610 | Sarah   |         29 |
+------+---------+------------+
```

To produce the organizational chart with the management chain for each employee (that is, the path from CEO to employee), use a recursive CTE:

```
WITH RECURSIVE employee_paths (id, name, path) AS
(
  SELECT id, name, CAST(id AS CHAR(200))
    FROM employees
    WHERE manager_id IS NULL
  UNION ALL
  SELECT e.id, e.name, CONCAT(ep.path, ',', e.id)
    FROM employee_paths AS ep JOIN employees AS e
      ON ep.id = e.manager_id
)
SELECT * FROM employee_paths ORDER BY path;
```

The CTE produces this output:

```
+------+---------+-----------------+
| id   | name    | path            |
+------+---------+-----------------+
|  333 | Yasmina | 333             |
|  198 | John    | 333,198         |
|   29 | Pedro   | 333,198,29      |
| 4610 | Sarah   | 333,198,29,4610 |
|   72 | Pierre  | 333,198,29,72   |
|  692 | Tarek   | 333,692         |
|  123 | Adil    | 333,692,123     |
+------+---------+-----------------+
```

How the CTE works:

- The nonrecursive `SELECT` produces the row for the CEO (the row with a `NULL` manager ID).

  The `path` column is widened to `CHAR(200)` to ensure that there is room for the longer `path` values produced by the recursive `SELECT`.

- Each row produced by the recursive `SELECT` finds all employees who report directly to an employee produced by a previous row. For each such employee, the row includes the employee ID and name, and the employee management chain. The chain is the manager's chain, with the employee ID added to the end.

- Recursion ends when employees have no others who report to them.

To find the path for a specific employee or employees, add a `WHERE` clause to the top-level `SELECT`. For example, to display the results for Tarek and Sarah, modify that `SELECT` like this:

```
mysql> WITH RECURSIVE ...
       ...
       SELECT * FROM employees_extended
       WHERE id IN (692, 4610)
       ORDER BY path;
+------+-------+-----------------+
| id   | name  | path            |
+------+-------+-----------------+
| 4610 | Sarah | 333,198,29,4610 |
|  692 | Tarek | 333,692         |
+------+-------+-----------------+
```

Common Table Expressions Compared to Similar Constructs

Common table expressions (CTEs) are similar to derived tables in some ways:

- Both constructs are named.
- Both constructs exist for the scope of a single statement.

Because of these similarities, CTEs and derived tables often can be used interchangeably. As a trivial example, these statements are equivalent:

```
WITH cte AS (SELECT 1) SELECT * FROM cte;
SELECT * FROM (SELECT 1) AS dt;
```

However, CTEs have some advantages over derived tables:

- A derived table can be referenced only a single time within a query. A CTE can be referenced multiple times. To use multiple instances of a derived table result, you must derive the result multiple times.
- A CTE can be self-referencing (recursive).
- One CTE can refer to another.
- A CTE may be easier to read when its definition appears at the beginning of the statement rather than embedded within it.

CTEs are similar to tables created with `CREATE [TEMPORARY] TABLE` but need not be defined or dropped explicitly. For a CTE, you need no privileges to create tables.

13.3 Transactional and Locking Statements

MySQL supports local transactions (within a given client session) through statements such as `SET autocommit`, `START TRANSACTION`, `COMMIT`, and `ROLLBACK`. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”. XA transaction support enables MySQL to participate in distributed transactions as well. See Section 13.3.8, “XA Transactions”.

13.3.1 START TRANSACTION, COMMIT, and ROLLBACK Syntax

```
START TRANSACTION
    [transaction_characteristic [, transaction_characteristic] ...]
```

```
transaction_characteristic: {
    WITH CONSISTENT SNAPSHOT
  | READ WRITE
  | READ ONLY
}

BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

These statements provide control over use of transactions:

- `START TRANSACTION` or `BEGIN` start a new transaction.
- `COMMIT` commits the current transaction, making its changes permanent.
- `ROLLBACK` rolls back the current transaction, canceling its changes.
- `SET autocommit` disables or enables the default autocommit mode for the current session.

By default, MySQL runs with autocommit mode enabled. This means that, when not otherwise inside a transaction, each statement is atomic, as if it were surrounded by `START TRANSACTION` and `COMMIT`. You cannot use `ROLLBACK` to undo the effect; however, if an error occurs during statement execution, the statement is rolled back.

To disable autocommit mode implicitly for a single series of statements, use the `START TRANSACTION` statement:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

With `START TRANSACTION`, autocommit remains disabled until you end the transaction with `COMMIT` or `ROLLBACK`. The autocommit mode then reverts to its previous state.

`START TRANSACTION` permits several modifiers that control transaction characteristics. To specify multiple modifiers, separate them by commas.

- The `WITH CONSISTENT SNAPSHOT` modifier starts a consistent read for storage engines that are capable of it. This applies only to `InnoDB`. The effect is the same as issuing a `START TRANSACTION` followed by a `SELECT` from any `InnoDB` table. See Section 15.7.2.3, “Consistent Nonlocking Reads”. The `WITH CONSISTENT SNAPSHOT` modifier does not change the current transaction isolation level, so it provides a consistent snapshot only if the current isolation level is one that permits a consistent read. The only isolation level that permits a consistent read is `REPEATABLE READ`. For all other isolation levels, the `WITH CONSISTENT SNAPSHOT` clause is ignored. A warning is generated when the `WITH CONSISTENT SNAPSHOT` clause is ignored.
- The `READ WRITE` and `READ ONLY` modifiers set the transaction access mode. They permit or prohibit changes to tables used in the transaction. The `READ ONLY` restriction prevents the transaction from modifying or locking both transactional and nontransactional tables that are visible to other transactions; the transaction can still modify or lock temporary tables.

  MySQL enables extra optimizations for queries on `InnoDB` tables when the transaction is known to be read-only. Specifying `READ ONLY` ensures these optimizations are applied in cases where the read-only status cannot be determined automatically. See Section 8.5.3, “Optimizing InnoDB Read-Only Transactions” for more information.

If no access mode is specified, the default mode applies. Unless the default has been changed, it is read/write. It is not permitted to specify both `READ WRITE` and `READ ONLY` in the same statement.

In read-only mode, it remains possible to change tables created with the `TEMPORARY` keyword using DML statements. Changes made with DDL statements are not permitted, just as with permanent tables.

For additional information about transaction access mode, including ways to change the default mode, see Section 13.3.7, "SET TRANSACTION Syntax".

If the `read_only` system variable is enabled, explicitly starting a transaction with `START TRANSACTION READ WRITE` requires the `CONNECTION_ADMIN` or `SUPER` privilege.

Important

Many APIs used for writing MySQL client applications (such as JDBC) provide their own methods for starting transactions that can (and sometimes should) be used instead of sending a `START TRANSACTION` statement from the client. See Chapter 28, *Connectors and APIs*, or the documentation for your API, for more information.

To disable autocommit mode explicitly, use the following statement:

```
SET autocommit=0;
```

After disabling autocommit mode by setting the `autocommit` variable to zero, changes to transaction-safe tables (such as those for `InnoDB` or `NDB`) are not made permanent immediately. You must use `COMMIT` to store your changes to disk or `ROLLBACK` to ignore the changes.

`autocommit` is a session variable and must be set for each session. To disable autocommit mode for each new connection, see the description of the `autocommit` system variable at Section 5.1.8, "Server System Variables".

`BEGIN` and `BEGIN WORK` are supported as aliases of `START TRANSACTION` for initiating a transaction. `START TRANSACTION` is standard SQL syntax, is the recommended way to start an ad-hoc transaction, and permits modifiers that `BEGIN` does not.

The `BEGIN` statement differs from the use of the `BEGIN` keyword that starts a `BEGIN ... END` compound statement. The latter does not begin a transaction. See Section 13.6.1, "BEGIN ... END Compound-Statement Syntax".

Note

Within all stored programs (stored procedures and functions, triggers, and events), the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. Begin a transaction in this context with `START TRANSACTION` instead.

The optional `WORK` keyword is supported for `COMMIT` and `ROLLBACK`, as are the `CHAIN` and `RELEASE` clauses. `CHAIN` and `RELEASE` can be used for additional control over transaction completion. The value of the `completion_type` system variable determines the default completion behavior. See Section 5.1.8, "Server System Variables".

The `AND CHAIN` clause causes a new transaction to begin as soon as the current one ends, and the new transaction has the same isolation level as the just-terminated transaction. The new transaction also uses the same access mode (`READ WRITE` or `READ ONLY`) as the just-terminated transaction. The `RELEASE` clause causes the server to disconnect the current client session after terminating the current

transaction. Including the `NO` keyword suppresses `CHAIN` or `RELEASE` completion, which can be useful if the `completion_type` system variable is set to cause chaining or release completion by default.

Beginning a transaction causes any pending transaction to be committed. See Section 13.3.3, “Statements That Cause an Implicit Commit”, for more information.

Beginning a transaction also causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

For best results, transactions should be performed using only tables managed by a single transaction-safe storage engine. Otherwise, the following problems can occur:

- If you use tables from more than one transaction-safe storage engine (such as `InnoDB`), and the transaction isolation level is not `SERIALIZABLE`, it is possible that when one transaction commits, another ongoing transaction that uses the same tables will see only some of the changes made by the first transaction. That is, the atomicity of transactions is not guaranteed with mixed engines and inconsistencies can result. (If mixed-engine transactions are infrequent, you can use `SET TRANSACTION ISOLATION LEVEL` to set the isolation level to `SERIALIZABLE` on a per-transaction basis as necessary.)
- If you use tables that are not transaction-safe within a transaction, changes to those tables are stored at once, regardless of the status of autocommit mode.
- If you issue a `ROLLBACK` statement after updating a nontransactional table within a transaction, an `ER_WARNING_NOT_COMPLETE_ROLLBACK` warning occurs. Changes to transaction-safe tables are rolled back, but not changes to nontransaction-safe tables.

Each transaction is stored in the binary log in one chunk, upon `COMMIT`. Transactions that are rolled back are not logged. (**Exception**: Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a `ROLLBACK` statement at the end to ensure that modifications to the nontransactional tables are replicated.) See Section 5.4.4, “The Binary Log”.

You can change the isolation level or access mode for transactions with the `SET TRANSACTION` statement. See Section 13.3.7, “SET TRANSACTION Syntax”.

Rolling back can be a slow operation that may occur implicitly without the user having explicitly asked for it (for example, when an error occurs). Because of this, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the session, not only for explicit rollbacks performed with the `ROLLBACK` statement but also for implicit rollbacks.

Note

In MySQL 8.0, `BEGIN`, `COMMIT`, and `ROLLBACK` are not affected by `--replicate-do-db` or `--replicate-ignore-db` rules.

13.3.2 Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, those that create, drop, or alter tables or stored routines.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back in such cases by issuing a `ROLLBACK` statement.

13.3.3 Statements That Cause an Implicit Commit

The statements listed in this section (and any synonyms for them) implicitly end any transaction active in the current session, as if you had done a `COMMIT` before executing the statement.

Most of these statements also cause an implicit commit after executing. The intent is to handle each such statement in its own special transaction. Transaction-control and locking statements are exceptions: If an implicit commit occurs before execution, another does not occur after.

- **Data definition language (DDL) statements that define or modify database objects.** `ALTER EVENT`, `ALTER FUNCTION`, `ALTER PROCEDURE`, `ALTER SERVER`, `ALTER TABLE`, `ALTER VIEW`, `CREATE DATABASE`, `CREATE EVENT`, `CREATE FUNCTION`, `CREATE INDEX`, `CREATE PROCEDURE`, `CREATE ROLE`, `CREATE SERVER`, `CREATE SPATIAL REFERENCE SYSTEM`, `CREATE TABLE`, `CREATE TRIGGER`, `CREATE VIEW`, `DROP DATABASE`, `DROP EVENT`, `DROP FUNCTION`, `DROP INDEX`, `DROP PROCEDURE`, `DROP ROLE`, `DROP SERVER`, `DROP SPATIAL REFERENCE SYSTEM`, `DROP TABLE`, `DROP TRIGGER`, `DROP VIEW`, `INSTALL PLUGIN`, `RENAME TABLE`, `TRUNCATE TABLE`, `UNINSTALL PLUGIN`.

  `CREATE TABLE` and `DROP TABLE` statements do not commit a transaction if the `TEMPORARY` keyword is used. (This does not apply to other operations on temporary tables such as `ALTER TABLE` and `CREATE INDEX`, which do cause a commit.) However, although no implicit commit occurs, neither can the statement be rolled back, which means that the use of such statements causes transactional atomicity to be violated. For example, if you use `CREATE TEMPORARY TABLE` and then roll back the transaction, the table remains in existence.

  The `CREATE TABLE` statement in `InnoDB` is processed as a single transaction. This means that a `ROLLBACK` from the user does not undo `CREATE TABLE` statements the user made during that transaction.

  `CREATE TABLE ... SELECT` causes an implicit commit before and after the statement is executed when you are creating nontemporary tables. (No commit occurs for `CREATE TEMPORARY TABLE ... SELECT`.)

- **Statements that implicitly use or modify tables in the `mysql` database.** `ALTER USER`, `CREATE USER`, `DROP USER`, `GRANT`, `RENAME USER`, `REVOKE`, `SET PASSWORD`.

- **Transaction-control and locking statements.** `BEGIN`, `LOCK TABLES`, `SET autocommit = 1` (if the value is not already 1), `START TRANSACTION`, `UNLOCK TABLES`.

  `UNLOCK TABLES` commits a transaction only if any tables currently have been locked with `LOCK TABLES` to acquire nontransactional table locks. A commit does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table-level locks.

  Transactions cannot be nested. This is a consequence of the implicit commit performed for any current transaction when you issue a `START TRANSACTION` statement or one of its synonyms.

  Statements that cause an implicit commit cannot be used in an XA transaction while the transaction is in an `ACTIVE` state.

  The `BEGIN` statement differs from the use of the `BEGIN` keyword that starts a `BEGIN ... END` compound statement. The latter does not cause an implicit commit. See Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”.

- **Data loading statements.** `LOAD DATA`. `LOAD DATA` causes an implicit commit only for tables using the `NDB` storage engine.

- **Administrative statements.** `ANALYZE TABLE`, `CACHE INDEX`, `CHECK TABLE`, `FLUSH`, `LOAD INDEX INTO CACHE`, `OPTIMIZE TABLE`, `REPAIR TABLE`, `RESET` (but not `RESET PERSIST`).

- **Replication control statements**. `START SLAVE`, `STOP SLAVE`, `RESET SLAVE`, `CHANGE MASTER TO`.

13.3.4 SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

`InnoDB` supports the SQL statements `SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, `RELEASE SAVEPOINT` and the optional `WORK` keyword for `ROLLBACK`.

The `SAVEPOINT` statement sets a named transaction savepoint with a name of *`identifier`*. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The `ROLLBACK TO SAVEPOINT` statement rolls back a transaction to the named savepoint without terminating the transaction. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but `InnoDB` does *not* release the row locks that were stored in memory after the savepoint. (For a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

If the `ROLLBACK TO SAVEPOINT` statement returns the following error, it means that no savepoint with the specified name exists:

```
ERROR 1305 (42000): SAVEPOINT identifier does not exist
```

The `RELEASE SAVEPOINT` statement removes the named savepoint from the set of savepoints of the current transaction. No commit or rollback occurs. It is an error if the savepoint does not exist.

All savepoints of the current transaction are deleted if you execute a `COMMIT`, or a `ROLLBACK` that does not name a savepoint.

A new savepoint level is created when a stored function is invoked or a trigger is activated. The savepoints on previous levels become unavailable and thus do not conflict with savepoints on the new level. When the function or trigger terminates, any savepoints it created are released and the previous savepoint level is restored.

13.3.5 LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Syntax

```
LOCK INSTANCE FOR BACKUP

UNLOCK INSTANCE
```

`LOCK INSTANCE FOR BACKUP` acquires an instance-level *backup lock* that permits DML during an online backup while preventing operations that could result in an inconsistent snapshot.

Executing the `LOCK INSTANCE FOR BACKUP` statement requires the `BACKUP_ADMIN` privilege. The `BACKUP_ADMIN` privilege is automatically granted to users with the `RELOAD` privilege when performing an in-place upgrade to MySQL 8.0 from an earlier version.

Multiple sessions can hold a backup lock simultaneously.

`UNLOCK INSTANCE` releases a backup lock held by the current session. A backup lock held by a session is also released if the session is terminated.

`LOCK INSTANCE FOR BACKUP` prevents files from being created, renamed, or removed. `REPAIR TABLE` `TRUNCATE TABLE`, `OPTIMIZE TABLE`, and account management statements are blocked. See Section 13.7.1, “Account Management Statements”. Operations that modify `InnoDB` files that are not recorded in the `InnoDB` redo log are also blocked.

`LOCK INSTANCE FOR BACKUP` permits DDL operations that only affect user-created temporary tables. In effect, files that belong to user-created temporary tables can be created, renamed, or removed while a backup lock is held. Creation of binary log files is also permitted.

A backup lock acquired by `LOCK INSTANCE FOR BACKUP` is independent of transactional locks and locks taken by `FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK`, and the following sequences of statements are permitted:

```
LOCK INSTANCE FOR BACKUP;
FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK;
UNLOCK TABLES;
UNLOCK INSTANCE;
```

```
FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK;
LOCK INSTANCE FOR BACKUP;
UNLOCK INSTANCE;
UNLOCK TABLES;
```

The `lock_wait_timeout` setting defines the amount of time that a `LOCK INSTANCE FOR BACKUP` statement waits to acquire a lock before giving up.

13.3.6 LOCK TABLES and UNLOCK TABLES Syntax

```
LOCK TABLES
    tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type] ...

lock_type: {
    READ [LOCAL]
  | [LOW_PRIORITY] WRITE
}

UNLOCK TABLES
```

MySQL enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

Locks may be used to emulate transactions or to get more speed when updating tables. This is explained in more detail in Table-Locking Restrictions and Conditions.

`LOCK TABLES` explicitly acquires table locks for the current client session. Table locks can be acquired for base tables or views. You must have the `LOCK TABLES` privilege, and the `SELECT` privilege for each object to be locked.

For view locking, `LOCK TABLES` adds all base tables used in the view to the set of tables to be locked and locks them automatically. If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly, as described in LOCK TABLES and Triggers.

If you lock a table explicitly with `LOCK TABLES`, any tables related by a foreign key constraint are opened and locked implicitly. For foreign key checks, a shared read-only lock (`LOCK TABLES READ`) is taken on

related tables. For cascading updates, a shared-nothing write lock (`LOCK TABLES WRITE`) is taken on related tables that are involved in the operation.

`UNLOCK TABLES` explicitly releases any table locks held by the current session. `LOCK TABLES` implicitly releases any table locks held by the current session before acquiring new locks.

Another use for `UNLOCK TABLES` is to release the global read lock acquired with the `FLUSH TABLES WITH READ LOCK` statement, which enables you to lock all tables in all databases. See Section 13.7.8.3, "FLUSH Syntax". (This is a very convenient way to get backups if you have a file system such as Veritas that can take snapshots in time.)

A table lock protects only against inappropriate reads or writes by other sessions. A session holding a `WRITE` lock can perform table-level operations such as `DROP TABLE` or `TRUNCATE TABLE`. For sessions holding a `READ` lock, `DROP TABLE` and `TRUNCATE TABLE` operations are not permitted.

The following discussion applies only to non-`TEMPORARY` tables. `LOCK TABLES` is permitted (but ignored) for a `TEMPORARY` table. The table can be accessed freely by the session within which it was created, regardless of what other locking may be in effect. No lock is necessary because no other session can see the table.

- Table Lock Acquisition
- Table Lock Release
- Interaction of Table Locking and Transactions
- LOCK TABLES and Triggers
- Table-Locking Restrictions and Conditions

Table Lock Acquisition

To acquire table locks within the current session, use the `LOCK TABLES` statement, which acquires metadata locks (see Section 8.11.4, "Metadata Locking").

The following lock types are available:

`READ [LOCAL]` lock:

- The session that holds the lock can read the table (but not write it).
- Multiple sessions can acquire a `READ` lock for the table at the same time.
- Other sessions can read the table without explicitly acquiring a `READ` lock.
- The `LOCAL` modifier enables nonconflicting `INSERT` statements (concurrent inserts) by other sessions to execute while the lock is held. (See Section 8.11.3, "Concurrent Inserts".) However, `READ LOCAL` cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock. For `InnoDB` tables, `READ LOCAL` is the same as `READ`.

`[LOW_PRIORITY] WRITE` lock:

- The session that holds the lock can read and write the table.
- Only the session that holds the lock can access the table. No other session can access it until the lock is released.

- Lock requests for the table by other sessions block while the `WRITE` lock is held.
- The `LOW_PRIORITY` modifier has no effect. In previous versions of MySQL, it affected locking behavior, but this is no longer true. It is now deprecated and its use produces a warning. Use `WRITE` without `LOW_PRIORITY` instead.

`WRITE` locks normally have higher priority than `READ` locks to ensure that updates are processed as soon as possible. This means that if one session obtains a `READ` lock and then another session requests a `WRITE` lock, subsequent `READ` lock requests wait until the session that requested the `WRITE` lock has obtained the lock and released it. (An exception to this policy can occur for small values of the `max_write_lock_count` system variable; see Section 8.11.4, “Metadata Locking”.)

If the `LOCK TABLES` statement must wait due to locks held by other sessions on any of the tables, it blocks until all locks can be acquired.

A session that requires locks must acquire all the locks that it needs in a single `LOCK TABLES` statement. While the locks thus obtained are held, the session can access only the locked tables. For example, in the following sequence of statements, an error occurs for the attempt to access `t2` because it was not locked in the `LOCK TABLES` statement:

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

Tables in the `INFORMATION_SCHEMA` database are an exception. They can be accessed without being locked explicitly even while a session holds table locks obtained with `LOCK TABLES`.

You cannot refer to a locked table multiple times in a single query using the same name. Use aliases instead, and obtain a separate lock for the table and each alias:

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

The error occurs for the first `INSERT` because there are two references to the same name for a locked table. The second `INSERT` succeeds because the references to the table use different names.

If your statements refer to a table by means of an alias, you must lock the table using that same alias. It does not work to lock the table without specifying the alias:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

Conversely, if you lock a table using an alias, you must refer to it in your statements using that alias:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

Note

`LOCK TABLES` or `UNLOCK TABLES`, when applied to a partitioned table, always locks or unlocks the entire table; these statements do not support partition lock pruning. See Partitioning and Locking.

Table Lock Release

When the table locks held by a session are released, they are all released at the same time. A session can release its locks explicitly, or locks may be released implicitly under certain conditions.

- A session can release its locks explicitly with `UNLOCK TABLES`.
- If a session issues a `LOCK TABLES` statement to acquire a lock while already holding locks, its existing locks are released implicitly before the new locks are granted.
- If a session begins a transaction (for example, with `START TRANSACTION`), an implicit `UNLOCK TABLES` is performed, which causes existing locks to be released. (For additional information about the interaction between table locking and transactions, see Interaction of Table Locking and Transactions.)

If the connection for a client session terminates, whether normally or abnormally, the server implicitly releases all table locks held by the session (transactional and nontransactional). If the client reconnects, the locks will no longer be in effect. In addition, if the client had an active transaction, the server rolls back the transaction upon disconnect, and if reconnect occurs, the new session begins with autocommit enabled. For this reason, clients may wish to disable auto-reconnect. With auto-reconnect in effect, the client is not notified if reconnect occurs but any table locks or current transaction will have been lost. With auto-reconnect disabled, if the connection drops, an error occurs for the next statement issued. The client can detect the error and take appropriate action such as reacquiring the locks or redoing the transaction. See Section 28.7.28, “C API Automatic Reconnection Control”.

Note

If you use `ALTER TABLE` on a locked table, it may become unlocked. For example, if you attempt a second `ALTER TABLE` operation, the result may be an error `Table 'tbl_name' was not locked with LOCK TABLES`. To handle this, lock the table again prior to the second alteration. See also Section B.4.6.1, “Problems with ALTER TABLE”.

Interaction of Table Locking and Transactions

`LOCK TABLES` and `UNLOCK TABLES` interact with the use of transactions as follows:

- `LOCK TABLES` is not transaction-safe and implicitly commits any active transaction before attempting to lock the tables.
- `UNLOCK TABLES` implicitly commits any active transaction, but only if `LOCK TABLES` has been used to acquire table locks. For example, in the following set of statements, `UNLOCK TABLES` releases the global read lock but does not commit the transaction because no table locks are in effect:

```
FLUSH TABLES WITH READ LOCK;
START TRANSACTION;
SELECT ... ;
UNLOCK TABLES;
```

- Beginning a transaction (for example, with `START TRANSACTION`) implicitly commits any current transaction and releases existing table locks.

- `FLUSH TABLES WITH READ LOCK` acquires a global read lock and not table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits. For example, `START TRANSACTION` does not release the global read lock. See Section 13.7.8.3, “FLUSH Syntax”.
- Other statements that implicitly cause transactions to be committed do not release existing table locks. For a list of such statements, see Section 13.3.3, “Statements That Cause an Implicit Commit”.
- The correct way to use `LOCK TABLES` and `UNLOCK TABLES` with transactional tables, such as `InnoDB` tables, is to begin a transaction with `SET autocommit = 0` (not `START TRANSACTION`) followed by `LOCK TABLES`, and to not call `UNLOCK TABLES` until you commit the transaction explicitly. For example, if you need to write to table `t1` and read from table `t2`, you can do this:

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

  When you call `LOCK TABLES`, `InnoDB` internally takes its own table lock, and MySQL takes its own table lock. `InnoDB` releases its internal table lock at the next commit, but for MySQL to release its table lock, you have to call `UNLOCK TABLES`. You should not have `autocommit = 1`, because then `InnoDB` releases its internal table lock immediately after the call of `LOCK TABLES`, and deadlocks can very easily happen. `InnoDB` does not acquire the internal table lock at all if `autocommit = 1`, to help old applications avoid unnecessary deadlocks.
- `ROLLBACK` does not release table locks.

LOCK TABLES and Triggers

If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly:

- The locks are taken as the same time as those acquired explicitly with the `LOCK TABLES` statement.
- The lock on a table used in a trigger depends on whether the table is used only for reading. If so, a read lock suffices. Otherwise, a write lock is used.
- If a table is locked explicitly for reading with `LOCK TABLES`, but needs to be locked for writing because it might be modified within a trigger, a write lock is taken rather than a read lock. (That is, an implicit write lock needed due to the table's appearance within a trigger causes an explicit read lock request for the table to be converted to a write lock request.)

Suppose that you lock two tables, `t1` and `t2`, using this statement:

```
LOCK TABLES t1 WRITE, t2 READ;
```

If `t1` or `t2` have any triggers, tables used within the triggers will also be locked. Suppose that `t1` has a trigger defined like this:

```
CREATE TRIGGER t1_a_ins AFTER INSERT ON t1 FOR EACH ROW
BEGIN
  UPDATE t4 SET count = count+1
      WHERE id = NEW.id AND EXISTS (SELECT a FROM t3);
  INSERT INTO t2 VALUES(1, 2);
END;
```

The result of the `LOCK TABLES` statement is that `t1` and `t2` are locked because they appear in the statement, and `t3` and `t4` are locked because they are used within the trigger:

- `t1` is locked for writing per the `WRITE` lock request.
- `t2` is locked for writing, even though the request is for a `READ` lock. This occurs because `t2` is inserted into within the trigger, so the `READ` request is converted to a `WRITE` request.
- `t3` is locked for reading because it is only read from within the trigger.
- `t4` is locked for writing because it might be updated within the trigger.

Table-Locking Restrictions and Conditions

You can safely use `KILL` to terminate a session that is waiting for a table lock. See Section 13.7.8.4, “KILL Syntax”.

`LOCK TABLES` and `UNLOCK TABLES` cannot be used within stored programs.

Tables in the `performance_schema` database cannot be locked with `LOCK TABLES`, except the `setup_xxx` tables.

The following statements are prohibited while a `LOCK TABLES` statement is in effect: `CREATE TABLE`, `CREATE TABLE ... LIKE`, `CREATE VIEW`, `DROP VIEW`, and DDL statements on stored functions and procedures and events.

For some operations, system tables in the `mysql` database must be accessed. For example, the `HELP` statement requires the contents of the server-side help tables, and `CONVERT_TZ()` might need to read the time zone tables. The server implicitly locks the system tables for reading as necessary so that you need not lock them explicitly. These tables are treated as just described:

```
mysql.help_category
mysql.help_keyword
mysql.help_relation
mysql.help_topic
mysql.time_zone
mysql.time_zone_leap_second
mysql.time_zone_name
mysql.time_zone_transition
mysql.time_zone_transition_type
```

If you want to explicitly place a `WRITE` lock on any of those tables with a `LOCK TABLES` statement, the table must be the only one locked; no other table can be locked with the same statement.

Normally, you do not need to lock tables, because all single `UPDATE` statements are atomic; no other session can interfere with any other currently executing SQL statement. However, there are a few cases when locking tables may provide an advantage:

- If you are going to run many operations on a set of `MyISAM` tables, it is much faster to lock the tables you are going to use. Locking `MyISAM` tables speeds up inserting, updating, or deleting on them because MySQL does not flush the key cache for the locked tables until `UNLOCK TABLES` is called. Normally, the key cache is flushed after each SQL statement.

  The downside to locking the tables is that no session can update a `READ`-locked table (including the one holding the lock) and no session can access a `WRITE`-locked table other than the one holding the lock.

- If you are using tables for a nontransactional storage engine, you must use `LOCK TABLES` if you want to ensure that no other session modifies the tables between a `SELECT` and an `UPDATE`. The example shown here requires `LOCK TABLES` to execute safely:

```
LOCK TABLES trans READ, customer WRITE;
SELECT SUM(value) FROM trans WHERE customer_id=some_id;
UPDATE customer
  SET total_value=sum_from_previous_statement
  WHERE customer_id=some_id;
UNLOCK TABLES;
```

Without `LOCK TABLES`, it is possible that another session might insert a new row in the `trans` table between execution of the `SELECT` and `UPDATE` statements.

You can avoid using `LOCK TABLES` in many cases by using relative updates (`UPDATE customer SET value=value+new_value`) or the `LAST_INSERT_ID()` function.

You can also avoid locking tables in some cases by using the user-level advisory lock functions `GET_LOCK()` and `RELEASE_LOCK()`. These locks are saved in a hash table in the server and implemented with `pthread_mutex_lock()` and `pthread_mutex_unlock()` for high speed. See Section 12.14, “Locking Functions”.

See Section 8.11.1, “Internal Locking Methods”, for more information on locking policy.

13.3.7 SET TRANSACTION Syntax

```
SET [GLOBAL | SESSION] TRANSACTION
    transaction_characteristic [, transaction_characteristic] ...

transaction_characteristic: {
    ISOLATION LEVEL level
  | access_mode
}

level: {
     REPEATABLE READ
   | READ COMMITTED
   | READ UNCOMMITTED
   | SERIALIZABLE
}

access_mode: {
     READ WRITE
   | READ ONLY
}
```

This statement specifies transaction characteristics. It takes a list of one or more characteristic values separated by commas. Each characteristic value sets the transaction isolation level or access mode. The isolation level is used for operations on `InnoDB` tables. The access mode specifies whether transactions operate in read/write or read-only mode.

In addition, `SET TRANSACTION` can include an optional `GLOBAL` or `SESSION` keyword to indicate the scope of the statement.

- Transaction Isolation Levels
- Transaction Access Mode
- Transaction Characteristic Scope

Transaction Isolation Levels

To set the transaction isolation level, use an `ISOLATION LEVEL level` clause. It is not permitted to specify multiple `ISOLATION LEVEL` clauses in the same `SET TRANSACTION` statement.

The default isolation level is `REPEATABLE READ`. Other permitted values are `READ COMMITTED`, `READ UNCOMMITTED`, and `SERIALIZABLE`. For information about these isolation levels, see Section 15.7.2.1, “Transaction Isolation Levels”.

Transaction Access Mode

To set the transaction access mode, use a `READ WRITE` or `READ ONLY` clause. It is not permitted to specify multiple access-mode clauses in the same `SET TRANSACTION` statement.

By default, a transaction takes place in read/write mode, with both reads and writes permitted to tables used in the transaction. This mode may be specified explicitly using `SET TRANSACTION` with an access mode of `READ WRITE`.

If the transaction access mode is set to `READ ONLY`, changes to tables are prohibited. This may enable storage engines to make performance improvements that are possible when writes are not permitted.

In read-only mode, it remains possible to change tables created with the `TEMPORARY` keyword using DML statements. Changes made with DDL statements are not permitted, just as with permanent tables.

The `READ WRITE` and `READ ONLY` access modes also may be specified for an individual transaction using the `START TRANSACTION` statement.

Transaction Characteristic Scope

You can set transaction characteristics globally, for the current session, or for the next transaction only:

- With the `GLOBAL` keyword:
  - The statement applies globally for all subsequent sessions.
  - Existing sessions are unaffected.
- With the `SESSION` keyword:
  - The statement applies to all subsequent transactions performed within the current session.
  - The statement is permitted within transactions, but does not affect the current ongoing transaction.
  - If executed between transactions, the statement overrides any preceding statement that sets the next-transaction value of the named characteristics.
- Without any `SESSION` or `GLOBAL` keyword:
  - The statement applies only to the next single transaction performed within the session.
  - Subsequent transactions revert to using the session value of the named characteristics.
  - The statement is not permitted within transactions:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.02 sec)

mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
ERROR 1568 (25001): Transaction characteristics can't be changed
while a transaction is in progress
```

A change to global transaction characteristics requires the `CONNECTION_ADMIN` or `SUPER` privilege. Any session is free to change its session characteristics (even in the middle of a transaction), or the characteristics for its next transaction (prior to the start of that transaction).

To set the global isolation level at server startup, use the `--transaction-isolation=level` option on the command line or in an option file. Values of *`level`* for this option use dashes rather than spaces, so the permissible values are `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, or `SERIALIZABLE`.

Similarly, to set the global transaction access mode at server startup, use the `--transaction-read-only` option. The default is `OFF` (read/write mode) but the value can be set to `ON` for a mode of read only.

For example, to set the isolation level to `REPEATABLE READ` and the access mode to `READ WRITE`, use these lines in the `[mysqld]` section of an option file:

```
[mysqld]
transaction-isolation = REPEATABLE-READ
transaction-read-only = OFF
```

At runtime, characteristics at the global, session, and next-transaction scope levels can be set indirectly using the `SET TRANSACTION` statement, as described previously. They can also be set directly using the `SET` statement to assign values to the `transaction_isolation` and `transaction_read_only` system variables:

- `SET TRANSACTION` permits optional `GLOBAL` and `SESSION` keywords for setting transaction characteristics at different scope levels.
- The `SET` statement for assigning values to the `transaction_isolation` and `transaction_read_only` system variables has syntaxes for setting these variables at different scope levels.

The following tables show the characteristic scope level set by each `SET TRANSACTION` and variable-assignment syntax.

Table 13.9 SET TRANSACTION Syntax for Transaction Characteristics

Syntax	Affected Characteristic Scope
`SET GLOBAL TRANSACTION` *`transaction_characteristic`*	Global
`SET SESSION TRANSACTION` *`transaction_characteristic`*	Session
`SET TRANSACTION` *`transaction_characteristic`*	Next transaction only

Table 13.10 SET Syntax for Transaction Characteristics

Syntax	Affected Characteristic Scope
`SET GLOBAL` *`var_name = value`*	Global
`SET @@GLOBAL.`*`var_name = value`*	Global
`SET PERSIST` *`var_name = value`*	Global
`SET @@PERSIST.`*`var_name = value`*	Global
`SET PERSIST_ONLY` *`var_name = value`*	No runtime effect
`SET @@PERSIST_ONLY.`*`var_name = value`*	No runtime effect
`SET SESSION` *`var_name = value`*	Session
`SET @@SESSION.`*`var_name = value`*	Session
`SET` *`var_name = value`*	Session

Syntax	Affected Characteristic Scope
`SET @@var_name = value`	Next transaction only

It is possible to check the global and session values of transaction characteristics at runtime:

```
SELECT @@GLOBAL.transaction_isolation, @@GLOBAL.transaction_read_only;
SELECT @@SESSION.transaction_isolation, @@SESSION.transaction_read_only;
```

13.3.8 XA Transactions

Support for XA transactions is available for the `InnoDB` storage engine. The MySQL XA implementation is based on the X/Open CAE document *Distributed Transaction Processing: The XA Specification*. This document is published by The Open Group and available at http://www.opengroup.org/public/pubs/catalog/c193.htm. Limitations of the current XA implementation are described in Section 13.3.8.3, “Restrictions on XA Transactions”.

On the client side, there are no special requirements. The XA interface to a MySQL server consists of SQL statements that begin with the `XA` keyword. MySQL client programs must be able to send SQL statements and to understand the semantics of the XA statement interface. They do not need be linked against a recent client library. Older client libraries also will work.

Among the MySQL Connectors, MySQL Connector/J 5.0.0 and higher supports XA directly, by means of a class interface that handles the XA SQL statement interface for you.

XA supports distributed transactions, that is, the ability to permit multiple separate transactional resources to participate in a global transaction. Transactional resources often are RDBMSs but may be other kinds of resources.

A global transaction involves several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends ACID properties “up a level” so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties. (As with nondistributed transactions, `SERIALIZABLE` may be preferred if your applications are sensitive to read phenomena. `REPEATABLE READ` may not be sufficient for distributed transactions.)

Some examples of distributed transactions:

- An application may act as an integration tool that combines a messaging service with an RDBMS. The application makes sure that transactions dealing with message sending, retrieval, and processing that also involve a transactional database all happen in a global transaction. You can think of this as “transactional email.”
- An application performs actions that involve different database servers, such as a MySQL server and an Oracle server (or multiple MySQL servers), where actions that involve multiple servers must happen as part of a global transaction, rather than as separate transactions local to each server.
- A bank keeps account information in an RDBMS and distributes and receives money through automated teller machines (ATMs). It is necessary to ensure that ATM actions are correctly reflected in the accounts, but this cannot be done with the RDBMS alone. A global transaction manager integrates the ATM and database resources to ensure overall consistency of financial transactions.

Applications that use global transactions involve one or more Resource Managers and a Transaction Manager:

- A Resource Manager (RM) provides access to transactional resources. A database server is one kind of resource manager. It must be possible to either commit or roll back transactions managed by the RM.

- A Transaction Manager (TM) coordinates the transactions that are part of a global transaction. It communicates with the RMs that handle each of these transactions. The individual transactions within a global transaction are “branches” of the global transaction. Global transactions and their branches are identified by a naming scheme described later.

The MySQL implementation of XA enables a MySQL server to act as a Resource Manager that handles XA transactions within a global transaction. A client program that connects to the MySQL server acts as the Transaction Manager.

To carry out a global transaction, it is necessary to know which components are involved, and bring each component to a point when it can be committed or rolled back. Depending on what each component reports about its ability to succeed, they must all commit or roll back as an atomic group. That is, either all components must commit, or all components must roll back. To manage a global transaction, it is necessary to take into account that any component or the connecting network might fail.

The process for executing a global transaction uses two-phase commit (2PC). This takes place after the actions performed by the branches of the global transaction have been executed.

1. In the first phase, all branches are prepared. That is, they are told by the TM to get ready to commit. Typically, this means each RM that manages a branch records the actions for the branch in stable storage. The branches indicate whether they are able to do this, and these results are used for the second phase.

2. In the second phase, the TM tells the RMs whether to commit or roll back. If all branches indicated when they were prepared that they will be able to commit, all branches are told to commit. If any branch indicated when it was prepared that it will not be able to commit, all branches are told to roll back.

In some cases, a global transaction might use one-phase commit (1PC). For example, when a Transaction Manager finds that a global transaction consists of only one transactional resource (that is, a single branch), that resource can be told to prepare and commit at the same time.

13.3.8.1 XA Transaction SQL Syntax

To perform XA transactions in MySQL, use the following statements:

```
XA {START|BEGIN} xid [JOIN|RESUME]

XA END xid [SUSPEND [FOR MIGRATE]]

XA PREPARE xid

XA COMMIT xid [ONE PHASE]

XA ROLLBACK xid

XA RECOVER [CONVERT XID]
```

For `XA START`, the `JOIN` and `RESUME` clauses are recognized but have no effect.

For `XA END` the `SUSPEND [FOR MIGRATE]` clause is recognized but has no effect.

Each XA statement begins with the `XA` keyword, and most of them require an *`xid`* value. An *`xid`* is an XA transaction identifier. It indicates which transaction the statement applies to. *`xid`* values are supplied by the client, or generated by the MySQL server. An *`xid`* value has from one to three parts:

```
xid: gtrid [, bqual [, formatID ]]
```

gtrid is a global transaction identifier, *bqual* is a branch qualifier, and *formatID* is a number that identifies the format used by the *gtrid* and *bqual* values. As indicated by the syntax, *bqual* and *formatID* are optional. The default *bqual* value is '' if not given. The default *formatID* value is 1 if not given.

gtrid and *bqual* must be string literals, each up to 64 bytes (not characters) long. *gtrid* and *bqual* can be specified in several ways. You can use a quoted string ('ab'), hex string (X'6162', 0x6162), or bit value (b'*nnnn*').

formatID is an unsigned integer.

The *gtrid* and *bqual* values are interpreted in bytes by the MySQL server's underlying XA support routines. However, while an SQL statement containing an XA statement is being parsed, the server works with some specific character set. To be safe, write *gtrid* and *bqual* as hex strings.

xid values typically are generated by the Transaction Manager. Values generated by one TM must be different from values generated by other TMs. A given TM must be able to recognize its own *xid* values in a list of values returned by the XA RECOVER statement.

XA START *xid* starts an XA transaction with the given *xid* value. Each XA transaction must have a unique *xid* value, so the value must not currently be used by another XA transaction. Uniqueness is assessed using the *gtrid* and *bqual* values. All following XA statements for the XA transaction must be specified using the same *xid* value as that given in the XA START statement. If you use any of those statements but specify an *xid* value that does not correspond to some existing XA transaction, an error occurs.

One or more XA transactions can be part of the same global transaction. All XA transactions within a given global transaction must use the same *gtrid* value in the *xid* value. For this reason, *gtrid* values must be globally unique so that there is no ambiguity about which global transaction a given XA transaction is part of. The *bqual* part of the *xid* value must be different for each XA transaction within a global transaction. (The requirement that *bqual* values be different is a limitation of the current MySQL XA implementation. It is not part of the XA specification.)

The XA RECOVER statement returns information for those XA transactions on the MySQL server that are in the PREPARED state. (See Section 13.3.8.2, “XA Transaction States”.) The output includes a row for each such XA transaction on the server, regardless of which client started it.

XA RECOVER requires the XA_RECOVER_ADMIN privilege. This privilege requirement prevents users from discovering the XID values for outstanding prepared XA transactions other than their own. It does not affect normal commit or rollback of an XA transaction because the user who started it knows its XID.

XA RECOVER output rows look like this (for an example *xid* value consisting of the parts 'abc', 'def', and 7):

```
mysql> XA RECOVER;
+----------+--------------+--------------+--------+
| formatID | gtrid_length | bqual_length | data   |
+----------+--------------+--------------+--------+
|        7 |            3 |            3 | abcdef |
+----------+--------------+--------------+--------+
```

The output columns have the following meanings:

- formatID is the *formatID* part of the transaction *xid*
- gtrid_length is the length in bytes of the *gtrid* part of the *xid*

- `bqual_length` is the length in bytes of the *`bqual`* part of the *`xid`*
- `data` is the concatenation of the *`gtrid`* and *`bqual`* parts of the *`xid`*

XID values may contain nonprintable characters. `XA RECOVER` permits an optional `CONVERT XID` clause so that clients can request XID values in hexadecimal.

13.3.8.2 XA Transaction States

An XA transaction progresses through the following states:

1. Use `XA START` to start an XA transaction and put it in the `ACTIVE` state.
2. For an `ACTIVE` XA transaction, issue the SQL statements that make up the transaction, and then issue an `XA END` statement. `XA END` puts the transaction in the `IDLE` state.
3. For an `IDLE` XA transaction, you can issue either an `XA PREPARE` statement or an `XA COMMIT ... ONE PHASE` statement:
   - `XA PREPARE` puts the transaction in the `PREPARED` state. An `XA RECOVER` statement at this point will include the transaction's *`xid`* value in its output, because `XA RECOVER` lists all XA transactions that are in the `PREPARED` state.
   - `XA COMMIT ... ONE PHASE` prepares and commits the transaction. The *`xid`* value will not be listed by `XA RECOVER` because the transaction terminates.
4. For a `PREPARED` XA transaction, you can issue an `XA COMMIT` statement to commit and terminate the transaction, or `XA ROLLBACK` to roll back and terminate the transaction.

Here is a simple XA transaction that inserts a row into a table as part of a global transaction:

```
mysql> XA START 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mytable (i) VALUES(10);
Query OK, 1 row affected (0.04 sec)

mysql> XA END 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA PREPARE 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA COMMIT 'xatest';
Query OK, 0 rows affected (0.00 sec)
```

Within the context of a given client connection, XA transactions and local (non-XA) transactions are mutually exclusive. For example, if `XA START` has been issued to begin an XA transaction, a local transaction cannot be started until the XA transaction has been committed or rolled back. Conversely, if a local transaction has been started with `START TRANSACTION`, no XA statements can be used until the transaction has been committed or rolled back.

If an XA transaction is in the `ACTIVE` state, you cannot issue any statements that cause an implicit commit. That would violate the XA contract because you could not roll back the XA transaction. You will receive the following error if you try to execute such a statement:

```
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed
when global transaction is in the ACTIVE state
```

Statements to which the preceding remark applies are listed at Section 13.3.3, “Statements That Cause an Implicit Commit”.

13.3.8.3 Restrictions on XA Transactions

XA transaction support is limited to the `InnoDB` storage engine.

For “external XA,” a MySQL server acts as a Resource Manager and client programs act as Transaction Managers. For “Internal XA”, storage engines within a MySQL server act as RMs, and the server itself acts as a TM. Internal XA support is limited by the capabilities of individual storage engines. Internal XA is required for handling XA transactions that involve more than one storage engine. The implementation of internal XA requires that a storage engine support two-phase commit at the table handler level, and currently this is true only for `InnoDB`.

For `XA START`, the `JOIN` and `RESUME` clauses are recognized but have no effect.

For `XA END` the `SUSPEND [FOR MIGRATE]` clause is recognized but has no effect.

The requirement that the *`bqual`* part of the *`xid`* value be different for each XA transaction within a global transaction is a limitation of the current MySQL XA implementation. It is not part of the XA specification.

An XA transaction is written to the binary log in two parts. When `XA PREPARE` is issued, the first part of the transaction up to `XA PREPARE` is written using an initial GTID. A `XA_prepare_log_event` is used to identify such transactions in the binary log. When `XA COMMIT` or `XA ROLLBACK` is issued, a second part of the transaction containing only the `XA COMMIT` or `XA ROLLBACK` statement is written using a second GTID. Note that the initial part of the transaction, identified by `XA_prepare_log_event`, is not necessarily followed by its `XA COMMIT` or `XA ROLLBACK`, which can cause interleaved binary logging of any two XA transactions. The two parts of the XA transaction can even appear in different binary log files. This means that an XA transaction in `PREPARED` state is now persistent until an explicit `XA COMMIT` or `XA ROLLBACK` statement is issued, ensuring that XA transactions are compatible with replication.

On a replication slave, immediately after the XA transaction is prepared, it is detached from the slave applier thread, and can be committed or rolled back by any thread on the slave. This means that the same XA transaction can appear in the `events_transactions_current` table with different states on different threads. The `events_transactions_current` table displays the current status of the most recent monitored transaction event on the thread, and does not update this status when the thread is idle. So the XA transaction can still be displayed in the `PREPARED` state for the original applier thread, after it has been processed by another thread. To positively identify XA transactions that are still in the `PREPARED` state and need to be recovered, use the `XA RECOVER` statement rather than the Performance Schema transaction tables.

The following restrictions exist for using XA transactions:

- XA transactions are not fully resilient to an unexpected halt with respect to the binary log. If there is an unexpected halt while the server is in the middle of executing an `XA PREPARE`, `XA COMMIT`, `XA ROLLBACK`, or `XA COMMIT ... ONE PHASE` statement, the server might not be able to recover to a correct state, leaving the server and the binary log in an inconsistent state. In this situation, the binary log might either contain extra XA transactions that are not applied, or miss XA transactions that are applied. Also, if GTIDs are enabled, after recovery `@@GLOBAL.GTID_EXECUTED` might not correctly describe the transactions that have been applied. Note that if an unexpected halt occurs before `XA PREPARE`, between `XA PREPARE` and `XA COMMIT` (or `XA ROLLBACK`), or after `XA COMMIT` (or `XA ROLLBACK`), the server and binary log are correctly recovered and taken to a consistent state.

- The use of replication filters or binary log filters in combination with XA transactions is not supported. Filtering of tables could cause an XA transaction to be empty on a replication slave, and empty XA transactions are not supported. Also, with the settings `master_info_repository=TABLE` and

`relay_log_info_repository=TABLE` on a replication slave, which became the defaults in MySQL 8.0, the internal state of the data engine transaction is changed following a filtered XA transaction, and can become inconsistent with the replication transaction context state.

The error `ER_XA_REPLICATION_FILTERS` is logged whenever an XA transaction is impacted by a replication filter, whether or not the transaction was empty as a result. If the transaction is not empty, the replication slave is able to continue running, but you should take steps to discontinue the use of replication filters with XA transactions in order to avoid potential issues. If the transaction is empty, the replication slave stops. In that event, the replication slave might be in an undetermined state in which the consistency of the replication process might be compromised. In particular, the `gtid_executed` set on a slave of the slave might be inconsistent with that on the master. To resolve this situation, isolate the master and stop all replication, then check GTID consistency across the replication topology. Undo the XA transaction that generated the error message, then restart replication.

- XA transactions are considered unsafe for statement-based replication. If two XA transactions committed in parallel on the master are being prepared on the slave in the inverse order, locking dependencies can occur that cannot be safely resolved, and it is possible for replication to fail with deadlock on the slave. This situation can occur for a single-threaded or multithreaded replication slave. When `binlog_format=STATEMENT` is set, a warning is issued for DML statements inside XA transactions. When `binlog_format=MIXED` or `binlog_format=ROW` is set, DML statements inside XA transactions are logged using row-based replication, and the potential issue is not present.

Note

Prior to MySQL 5.7.7, XA transactions were not compatible with replication at all. This was because an XA transaction that was in `PREPARED` state would be rolled back on clean server shutdown or client disconnect. Similarly, an XA transaction that was in `PREPARED` state would still exist in `PREPARED` state in case the server was shutdown abnormally and then started again, but the contents of the transaction could not be written to the binary log. In both of these situations the XA transaction could not be replicated correctly.

13.4 Replication Statements

Replication can be controlled through the SQL interface using the statements described in this section. Statements are split into a group which controls master servers, a group which controls slave servers, and a group which can be applied to any replication servers.

13.4.1 SQL Statements for Controlling Master Servers

This section discusses statements for managing master replication servers. Section 13.4.2, “SQL Statements for Controlling Slave Servers”, discusses statements for managing slave servers.

In addition to the statements described here, the following `SHOW` statements are used with master servers in replication. For information about these statements, see Section 13.7.7, “SHOW Syntax”.

- `SHOW BINARY LOGS`
- `SHOW BINLOG EVENTS`
- `SHOW MASTER STATUS`
- `SHOW SLAVE HOSTS`

13.4.1.1 PURGE BINARY LOGS Syntax

```
PURGE { BINARY | MASTER } LOGS {
    TO 'log_name'
  | BEFORE datetime_expr
}
```

The binary log is a set of files that contain information about data modifications made by the MySQL server. The log consists of a set of binary log files, plus an index file (see Section 5.4.4, “The Binary Log”).

The `PURGE BINARY LOGS` statement deletes all the binary log files listed in the log index file prior to the specified log file name or date. `BINARY` and `MASTER` are synonyms. Deleted log files also are removed from the list recorded in the index file, so that the given log file becomes the first in the list.

`PURGE BINARY LOGS` requires the `BINLOG_ADMIN` privilege. This statement has no effect if the server was not started with the `--log-bin` option to enable binary logging.

Examples:

```
PURGE BINARY LOGS TO 'mysql-bin.010';
PURGE BINARY LOGS BEFORE '2019-04-02 22:46:26';
```

The `BEFORE` variant's *`datetime_expr`* argument should evaluate to a `DATETIME` value (a value in `'YYYY-MM-DD hh:mm:ss'` format).

This statement is safe to run while slaves are replicating. You need not stop them. If you have an active slave that currently is reading one of the log files you are trying to delete, this statement does not delete the log file that is in use or any log files later than that one, but it deletes any earlier log files. A warning message is issued in this situation. However, if a slave is not connected and you happen to purge one of the log files it has yet to read, the slave will be unable to replicate after it reconnects.

To safely purge binary log files, follow this procedure:

1. On each slave server, use `SHOW SLAVE STATUS` to check which log file it is reading.
2. Obtain a listing of the binary log files on the master server with `SHOW BINARY LOGS`.
3. Determine the earliest log file among all the slaves. This is the target file. If all the slaves are up to date, this is the last log file on the list.
4. Make a backup of all the log files you are about to delete. (This step is optional, but always advisable.)
5. Purge all log files up to but not including the target file.

`PURGE BINARY LOGS TO` and `PURGE BINARY LOGS BEFORE` both fail with an error when binary log files listed in the `.index` file had been removed from the system by some other means (such as using `rm` on Linux). (Bug #18199, Bug #18453) To handle such errors, edit the `.index` file (which is a simple text file) manually to ensure that it lists only the binary log files that are actually present, then run again the `PURGE BINARY LOGS` statement that failed.

Binary log files are automatically removed after the server's binary log expiration period. Removal of the files can take place at startup and when the binary log is flushed. The default binary log expiration period is 30 days. You can specify an alternative expiration period using the `binlog_expire_logs_seconds` system variable. If you are using replication, you should specify an expiration period that is no lower than the maximum amount of time your slaves might lag behind the master.

13.4.1.2 RESET MASTER Syntax

```
RESET MASTER [TO binary_log_file_index_number]
```

Warning

Use this statement with caution to ensure you do not lose any wanted binary log file data and GTID execution history.

`RESET MASTER` requires the `RELOAD` privilege.

For a server where binary logging is enabled (`log_bin` is `ON`), `RESET MASTER` deletes all existing binary log files and resets the binary log index file, resetting the server to its state before binary logging was started. A new empty binary log file is created so that binary logging can be restarted.

For a server where GTIDs are in use (`gtid_mode` is `ON`), issuing `RESET MASTER` resets the GTID execution history. The value of the `gtid_purged` system variable is set to an empty string (`''`), the global value (but not the session value) of the `gtid_executed` system variable is set to an empty string, and the `mysql.gtid_executed` table is cleared (see mysql.gtid_executed Table). If the GTID-enabled server has binary logging enabled, `RESET MASTER` also resets the binary log as described above. Note that `RESET MASTER` is the method to reset the GTID execution history even if the GTID-enabled server is a replication slave where binary logging is disabled; `RESET SLAVE` has no effect on the GTID execution history. For more information on resetting the GTID execution history, see Resetting the GTID Execution History.

Issuing `RESET MASTER` without the optional `TO` clause deletes all binary log files listed in the index file, resets the binary log index file to be empty, and creates a new binary log file starting at `1`. Use the optional `TO` clause to start the binary log file index from a number other than `1` after the reset.

Using `RESET MASTER` with the `TO` clause to specify a binary log file index number to start from simplifies failover by providing a single statement alternative to the `FLUSH BINARY LOGS` and `PURGE BINARY LOGS TO` statements. Check that you are using a reasonable value for the index number. If you enter an incorrect value, you can correct this by issuing another `RESET MASTER` statement with or without the `TO` clause. If you do not correct a value that is out of range, the server cannot be restarted.

The following example demonstrates `TO` clause usage:

```
RESET MASTER TO 1234;

SHOW BINARY LOGS;
+------------------+-----------+-----------+
| Log_name         | File_size | Encrypted |
+------------------+-----------+-----------+
| master-bin.001234 |       154 | No        |
+------------------+-----------+-----------+
```

Important

The effects of `RESET MASTER` without the `TO` clause differ from those of `PURGE BINARY LOGS` in 2 key ways:

1. `RESET MASTER` removes *all* binary log files that are listed in the index file, leaving only a single, empty binary log file with a numeric suffix of `.000001`, whereas the numbering is not reset by `PURGE BINARY LOGS`.

2. `RESET MASTER` is *not* intended to be used while any replication slaves are running. The behavior of `RESET MASTER` when used while slaves are running is undefined (and thus unsupported), whereas `PURGE BINARY LOGS` may be safely used while replication slaves are running.

> See also Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

`RESET MASTER` without the `TO` clause can prove useful when you first set up a master and slave, so that you can verify the setup as follows:

1. Start the master and slave, and start replication (see Section 17.1.2, “Setting Up Binary Log File Position Based Replication”).
2. Execute a few test queries on the master.
3. Check that the queries were replicated to the slave.
4. When replication is running correctly, issue `STOP SLAVE` followed by `RESET SLAVE` on the slave, then verify that no unwanted data from the test queries exists on the slave.
5. Issue `RESET MASTER` on the master to clean up the test queries.

After verifying the setup, resetting the master and slave and ensuring that no unwanted data or binary log files generated by testing remain on the master or slave, you can start the slave and begin replicating.

13.4.1.3 SET sql_log_bin Syntax

```
SET sql_log_bin = {OFF|ON}
```

The `sql_log_bin` variable controls whether logging to the binary log is enabled for the current session (assuming that the binary log itself is enabled). The default value is `ON`. To disable or enable binary logging for the current session, set the session `sql_log_bin` variable to `OFF` or `ON`.

Set this variable to `OFF` for a session to temporarily disable binary logging while making changes to the master you do not want replicated to the slave.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

It is not possible to set the session value of `sql_log_bin` within a transaction or subquery.

Setting this variable to `OFF` prevents new GTIDs from being assigned to transactions in the binary log. If you are using GTIDs for replication, this means that even when binary logging is later enabled again, the GTIDs written into the log from this point do not account for any transactions that occurred in the meantime, so in effect those transactions are lost.

`mysqldump` adds a `SET @@SESSION.sql_log_bin=0` statement to a dump file from a server where GTIDs are in use, which disables binary logging while the dump file is being reloaded. The statement prevents new GTIDs from being generated and assigned to the transactions in the dump file as they are executed, so that the original GTIDs for the transactions are used.

13.4.2 SQL Statements for Controlling Slave Servers

This section discusses statements for managing slave replication servers. Section 13.4.1, “SQL Statements for Controlling Master Servers”, discusses statements for managing master servers.

In addition to the statements described here, `SHOW SLAVE STATUS` and `SHOW RELAYLOG EVENTS` are also used with replication slaves. For information about these statements, see Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”, and Section 13.7.7.32, “SHOW RELAYLOG EVENTS Syntax”.

13.4.2.1 CHANGE MASTER TO Syntax

```
CHANGE MASTER TO option [, option] ... [ channel_option ]

option:
    MASTER_BIND = 'interface_name'
  | MASTER_HOST = 'host_name'
  | MASTER_USER = 'user_name'
  | MASTER_PASSWORD = 'password'
  | MASTER_PORT = port_num
  | PRIVILEGE_CHECKS_USER = {'account' | NULL}
  | MASTER_CONNECT_RETRY = interval
  | MASTER_RETRY_COUNT = count
  | MASTER_DELAY = interval
  | MASTER_HEARTBEAT_PERIOD = interval
  | MASTER_LOG_FILE = 'master_log_name'
  | MASTER_LOG_POS = master_log_pos
  | MASTER_AUTO_POSITION = {0|1}
  | RELAY_LOG_FILE = 'relay_log_name'
  | RELAY_LOG_POS = relay_log_pos
  | MASTER_COMPRESSION_ALGORITHMS = 'value'
  | MASTER_ZSTD_COMPRESSION_LEVEL = level
  | MASTER_SSL = {0|1}
  | MASTER_SSL_CA = 'ca_file_name'
  | MASTER_SSL_CAPATH = 'ca_directory_name'
  | MASTER_SSL_CERT = 'cert_file_name'
  | MASTER_SSL_CRL = 'crl_file_name'
  | MASTER_SSL_CRLPATH = 'crl_directory_name'
  | MASTER_SSL_KEY = 'key_file_name'
  | MASTER_SSL_CIPHER = 'cipher_list'
  | MASTER_SSL_VERIFY_SERVER_CERT = {0|1}
  | MASTER_TLS_VERSION = 'protocol_list'
  | MASTER_PUBLIC_KEY_PATH = 'key_file_name'
  | GET_MASTER_PUBLIC_KEY = {0|1}
  | IGNORE_SERVER_IDS = (server_id_list)

channel_option:
    FOR CHANNEL channel

server_id_list:
    [server_id [, server_id] ... ]
```

`CHANGE MASTER TO` changes the parameters that the slave server uses for connecting to the master server, for reading the master binary log, and reading the slave relay log. It also updates the contents of the master info and relay log info repositories (see Section 17.2.4, “Replication Relay and Status Logs”). `CHANGE MASTER TO` requires the `REPLICATION_SLAVE_ADMIN` or `SUPER` privilege.

You can issue `CHANGE MASTER TO` statements on a running slave without first stopping it, depending on the states of the slave SQL thread and slave I/O thread. The rules governing such use are provided later in this section.

When using a multithreaded slave (in other words `slave_parallel_workers` is greater than 0), stopping the slave can cause “gaps” in the sequence of transactions that have been executed from the relay log, regardless of whether the slave was stopped intentionally or otherwise. When such gaps exist, issuing `CHANGE MASTER TO` fails. The solution in this situation is to issue `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` which ensures that the gaps are closed.

The optional `FOR CHANNEL channel` clause enables you to name which replication channel the statement applies to. Providing a `FOR CHANNEL channel` clause applies the `CHANGE MASTER TO` statement to a specific replication channel, and is used to add a new channel or modify an existing channel. For example, to add a new channel called channel2:

```
CHANGE MASTER TO MASTER_HOST=host1, MASTER_PORT=3002 FOR CHANNEL 'channel2'
```

If no clause is named and no extra channels exist, the statement applies to the default channel.

When using multiple replication channels, if a `CHANGE MASTER TO` statement does not name a channel using a `FOR CHANNEL channel` clause, an error occurs. See Section 17.2.3, “Replication Channels” for more information.

Options not specified retain their value, except as indicated in the following discussion. Thus, in most cases, there is no need to specify options that do not change.

`MASTER_HOST`, `MASTER_USER`, `MASTER_PASSWORD`, and `MASTER_PORT` provide information to the slave about how to connect to its master:

- `MASTER_HOST` and `MASTER_PORT` are the host name (or IP address) of the master host and its TCP/IP port.

  Note

  Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

  If you specify the `MASTER_HOST` or `MASTER_PORT` option, the slave assumes that the master server is different from before (even if the option value is the same as its current value.) In this case, the old values for the master binary log file name and position are considered no longer applicable, so if you do not specify `MASTER_LOG_FILE` and `MASTER_LOG_POS` in the statement, `MASTER_LOG_FILE=''` and `MASTER_LOG_POS=4` are silently appended to it.

  Setting `MASTER_HOST=''` (that is, setting its value explicitly to an empty string) is *not* the same as not setting `MASTER_HOST` at all. Trying to set `MASTER_HOST` to an empty string fails with an error.

  Values used for `MASTER_HOST` and other `CHANGE MASTER TO` options are checked for linefeed (`\n` or `0x0A`) characters; the presence of such characters in these values causes the statement to fail with `ER_MASTER_INFO`. (Bug #11758581, Bug #50801)

- `MASTER_USER` and `MASTER_PASSWORD` are the user name and password of the account to use for connecting to the master.

  `MASTER_USER` cannot be made empty; setting `MASTER_USER = ''` or leaving it unset when setting a value for `MASTER_PASSWORD` causes an error (Bug #13427949).

  The password used for a MySQL Replication slave account in a `CHANGE MASTER TO` statement is limited to 32 characters in length; trying to use a password of more than 32 characters causes `CHANGE MASTER TO` to fail.

  The text of a running `CHANGE MASTER TO` statement, including values for `MASTER_USER` and `MASTER_PASSWORD`, can be seen in the output of a concurrent `SHOW PROCESSLIST` statement. (The complete text of a `START SLAVE` statement is also visible to `SHOW PROCESSLIST`.)

`PRIVILEGE_CHECKS_USER` (available as of MySQL 8.0.18) names a user account that supplies a security context for the specified channel. `NULL`, which is the default, means no security context is used. If you do not specify a channel and no other channels exist, the statement is applied to the default channel. For example, to start privilege checks on the channel `channel_1` on a running replication slave, issue the following statements:

```
mysql> STOP SLAVE FOR CHANNEL 'channel_1';
mysql> CHANGE MASTER TO PRIVILEGE_CHECKS_USER = 'priv_repl'@'%.example.com' FOR CHANNEL 'channel_1';
```

```
mysql> START SLAVE FOR CHANNEL 'channel_1';
```

The user name and host name for the user account must follow the syntax described in Section 6.2.4, “Specifying Account Names”, and the user must not be an anonymous user (with a blank user name) or the `CURRENT_USER`. The account must have the `REPLICATION_APPLIER` privilege, plus the required privileges to execute the transactions replicated on the channel. When you restart the replication channel, the privilege checks are applied from that point on. For more information, see Section 17.3.3, “Replication Privilege Checks”.

`MASTER_COMPRESSION_ALGORITHMS` and `MASTER_ZSTD_COMPRESSION_LEVEL` (available as of MySQL 8.0.18) enable control over the use of compression for connections to the master:

- `MASTER_COMPRESSION_ALGORITHMS` specifies the permitted compression algorithms. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. The default value is `uncompressed`.

  The value of `MASTER_COMPRESSION_ALGORITHMS` applies only if the `slave_compressed_protocol` system variable is disabled. If `slave_compressed_protocol` is enabled, it takes precedence over `MASTER_COMPRESSION_ALGORITHMS` and connections to the master use `zlib` compression if both master and slave support that algorithm.

- `MASTER_ZSTD_COMPRESSION_LEVEL` is the compression level to use for connections that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. The compression level setting has no effect on connections that do not use `zstd` compression.

For more information, see Section 4.2.6, “Connection Compression Control”.

The `MASTER_SSL_`*`xxx`* options and the `MASTER_TLS_VERSION` option specify how the slave uses encryption and ciphers to secure the replication connection. These options can be changed even on slaves that are compiled without SSL support. They are saved to the master info repository, but are ignored if the slave does not have SSL support enabled. The `MASTER_SSL_`*`xxx`* and `MASTER_TLS_VERSION` options perform the same functions as the `--ssl-`*`xxx`* and `--tls-version` client options described in Command Options for Encrypted Connections. The correspondence between the two sets of options, and the use of the `MASTER_SSL_`*`xxx`* and `MASTER_TLS_VERSION` options to set up a secure connection, is explained in Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”.

Important

To connect to the replication master using a user account that authenticates with the `caching_sha2_password` plugin, you must either set up a secure connection as described in Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”, or enable the unencrypted connection to support password exchange using an RSA key pair. The `caching_sha2_password` authentication plugin is the default for new users created from MySQL 8.0 (for details, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”). If the user account that you create or use for replication (as specified by the `MASTER_USER` option) uses this authentication plugin, and you are not using a secure connection, you must enable RSA key pair-based password exchange for a successful connection.

To enable RSA key pair-based password exchange, specify either the `MASTER_PUBLIC_KEY_PATH` or the `GET_MASTER_PUBLIC_KEY=1` option. Either of these options provides the RSA public key to the slave:

- `MASTER_PUBLIC_KEY_PATH` indicates the path name to a file containing a slave-side copy of the public key required by the master for RSA key pair-based password exchange. The file must be in PEM format. This option applies to slaves that authenticate with the

`sha256_password` or `caching_sha2_password` authentication plugin. (For `sha256_password`, `MASTER_PUBLIC_KEY_PATH` can be used only if MySQL was built using OpenSSL.)

- `GET_MASTER_PUBLIC_KEY` indicates whether to request from the master the public key required for RSA key pair-based password exchange. This option applies to slaves that authenticate with the `caching_sha2_password` authentication plugin. For connections by accounts that authenticate using this plugin, the master does not send the public key unless requested, so it must be requested or specified in the client. If `MASTER_PUBLIC_KEY_PATH` is given and specifies a valid public key file, it takes precedence over `GET_MASTER_PUBLIC_KEY`.

The `MASTER_HEARTBEAT_PERIOD`, `MASTER_CONNECT_RETRY`, and `MASTER_RETRY_COUNT` options control how the slave recognizes that the connection to the master has been lost and makes attempts to reconnect.

- The `slave_net_timeout` system variable specifies the number of seconds that the slave waits for either more data or a heartbeat signal from the master, before the slave considers the connection broken, aborts the read, and tries to reconnect. The default value is 60 seconds (one minute).

- The heartbeat interval, which stops the connection timeout occurring in the absence of data if the connection is still good, is controlled by the `MASTER_HEARTBEAT_PERIOD` option. A heartbeat signal is sent to the slave after that number of seconds, and the waiting period is reset whenever the master's binary log is updated with an event. Heartbeats are therefore sent by the master only if there are no unsent events in the binary log file for a period longer than this. The heartbeat interval *`interval`* is a decimal value having the range 0 to 4294967 seconds and a resolution in milliseconds; the smallest nonzero value is 0.001. Setting *`interval`* to 0 disables heartbeats altogether. The heartbeat interval defaults to half the value of the `slave_net_timeout` system variable. It is recorded in the master info log and shown in the `replication_connection_configuration` Performance Schema table. Issuing `RESET SLAVE` resets the heartbeat interval to the default value.

  Note that a change to the value or default setting of `slave_net_timeout` does not automatically change the heartbeat interval, whether that has been set explicitly or is using a previously calculated default. A warning is issued if you set `@@GLOBAL.slave_net_timeout` to a value less than that of the current heartbeat interval. If `slave_net_timeout` is changed, you must also issue `CHANGE MASTER TO` to adjust the heartbeat interval to an appropriate value so that the heartbeat signal occurs before the connection timeout. If you do not do this, the heartbeat signal has no effect, and if no data is received from the master, the slave can make repeated reconnection attempts, creating zombie dump threads.

- If the slave does need to reconnect, the first retry occurs immediately after the timeout. `MASTER_CONNECT_RETRY` specifies the interval between reconnection attempts, and `MASTER_RETRY_COUNT` limits the number of reconnection attempts. If both the default settings are used, the slave waits 60 seconds between reconnection attempts (`MASTER_CONNECT_RETRY=60`), and keeps attempting to reconnect at this rate for 60 days (`MASTER_RETRY_COUNT=86400`). These values are recorded in the master info log and shown in the `replication_connection_configuration` Performance Schema table. `MASTER_RETRY_COUNT` supersedes the `--master-retry-count` server startup option.

`MASTER_DELAY` specifies how many seconds behind the master the slave must lag. An event received from the master is not executed until at least *`interval`* seconds later than its execution on the master. The default is 0. An error occurs if *`interval`* is not a nonnegative integer in the range from 0 to $2^{31}-1$. For more information, see Section 17.4.10, “Delayed Replication”.

A `CHANGE MASTER TO` statement employing the `MASTER_DELAY` option can be executed on a running slave when the slave SQL thread is stopped.

`MASTER_BIND` is for use on replication slaves having multiple network interfaces, and determines which of the slave's network interfaces is chosen for connecting to the master.

The address configured with this option, if any, can be seen in the `Master_Bind` column of the output from `SHOW SLAVE STATUS`. In the master info repository table `mysql.slave_master_info`, the value can be seen as the `Master_bind` column.

The ability to bind a replication slave to a specific network interface is also supported by NDB Cluster.

`MASTER_LOG_FILE` and `MASTER_LOG_POS` are the coordinates at which the slave I/O thread should begin reading from the master the next time the thread starts. `RELAY_LOG_FILE` and `RELAY_LOG_POS` are the coordinates at which the slave SQL thread should begin reading from the relay log the next time the thread starts. If you specify either of `MASTER_LOG_FILE` or `MASTER_LOG_POS`, you cannot specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. If you specify either of `MASTER_LOG_FILE` or `MASTER_LOG_POS`, you also cannot specify `MASTER_AUTO_POSITION = 1` (described later in this section). If neither of `MASTER_LOG_FILE` or `MASTER_LOG_POS` is specified, the slave uses the last coordinates of the *slave SQL thread* before `CHANGE MASTER TO` was issued. This ensures that there is no discontinuity in replication, even if the slave SQL thread was late compared to the slave I/O thread, when you merely want to change, say, the password to use.

A `CHANGE MASTER TO` statement employing `RELAY_LOG_FILE`, `RELAY_LOG_POS`, or both options can be executed on a running slave when the slave SQL thread is stopped. Relay logs are preserved if at least one of the slave SQL thread and the slave I/O thread is running; if both threads are stopped, all relay log files are deleted unless at least one of `RELAY_LOG_FILE` or `RELAY_LOG_POS` is specified.

`RELAY_LOG_FILE` can use either an absolute or relative path, and uses the same base name as `MASTER_LOG_FILE`.

When `MASTER_AUTO_POSITION = 1` is used with `CHANGE MASTER TO`, the slave attempts to connect to the master using the GTID-based replication protocol. This option can be used with `CHANGE MASTER TO` only if both the slave SQL and slave I/O threads are stopped. Both the slave and the master must have GTIDs enabled (`GTID_MODE=ON`, `ON_PERMISSIVE`, or `OFF_PERMISSIVE` on the slave, and `GTID_MODE=ON` on the master). Auto-positioning is used for the connection, so the coordinates represented by `MASTER_LOG_FILE` and `MASTER_LOG_POS` are not used, and the use of either or both of these options together with `MASTER_AUTO_POSITION = 1` causes an error. If multi-source replication is enabled on the slave, you need to set the `MASTER_AUTO_POSITION = 1` option for each applicable replication channel.

With `MASTER_AUTO_POSITION = 1` set, in the initial connection handshake, the slave sends a GTID set containing the transactions that it has already received, committed, or both. The master responds by sending all transactions recorded in its binary log whose GTID is not included in the GTID set sent by the slave. This exchange ensures that the master only sends the transactions with a GTID that the slave has not already recorded or committed. If the slave receives transactions from more than one master, as in the case of a diamond topology, the auto-skip function ensures that the transactions are not applied twice. For details of how the GTID set sent by the slave is computed, see Section 17.1.3.3, “GTID Auto-Positioning”.

If any of the transactions that should be sent by the master have been purged from the master's binary log, or added to the set of GTIDs in the `gtid_purged` system variable by another method, the master sends the error **`ER_MASTER_HAS_PURGED_REQUIRED_GTIDS`** to the slave, and replication does not start. The GTIDs of the missing purged transactions are identified and listed in the master's error log in the warning message **`ER_FOUND_MISSING_GTIDS`**. Also, if during the exchange of transactions it is found that the slave has recorded or committed transactions with the master's UUID in the GTID, but the master itself has not committed them, the master sends the error **`ER_SLAVE_HAS_MORE_GTIDS_THAN_MASTER`** to the slave and replication does not start. For information on how to handle these situations, see Section 17.1.3.3, “GTID Auto-Positioning”.

You can see whether replication is running with auto-positioning enabled by checking the Performance Schema `replication_connection_status` table or the output of `SHOW SLAVE STATUS`. Disabling

the MASTER_AUTO_POSITION option again makes the slave revert to file-based replication, in which case you must also specify one or both of the MASTER_LOG_FILE or MASTER_LOG_POS options.

IGNORE_SERVER_IDS takes a comma-separated list of 0 or more server IDs. Events originating from the corresponding servers are ignored, with the exception of log rotation and deletion events, which are still recorded in the relay log.

In circular replication, the originating server normally acts as the terminator of its own events, so that they are not applied more than once. Thus, this option is useful in circular replication when one of the servers in the circle is removed. Suppose that you have a circular replication setup with 4 servers, having server IDs 1, 2, 3, and 4, and server 3 fails. When bridging the gap by starting replication from server 2 to server 4, you can include IGNORE_SERVER_IDS = (3) in the CHANGE MASTER TO statement that you issue on server 4 to tell it to use server 2 as its master instead of server 3. Doing so causes it to ignore and not to propagate any statements that originated with the server that is no longer in use.

If IGNORE_SERVER_IDS contains the server's own ID and the server was started with the --replicate-same-server-id option enabled, an error results.

Note

When global transaction identifiers (GTIDs) are used for replication, transactions that have already been applied are automatically ignored, so the IGNORE_SERVER_IDS function is not required and is deprecated. If gtid_mode=ON is set for the server, a deprecation warning is issued if you include the IGNORE_SERVER_IDS option in a CHANGE MASTER TO statement.

The master info repository and the output of SHOW SLAVE STATUS provide the list of servers that are currently ignored. For more information, see Section 17.2.4.2, “Slave Status Logs”, and Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”.

If a CHANGE MASTER TO statement is issued without any IGNORE_SERVER_IDS option, any existing list is preserved. To clear the list of ignored servers, it is necessary to use the option with an empty list:

```
CHANGE MASTER TO IGNORE_SERVER_IDS = ();
```

RESET SLAVE ALL clears IGNORE_SERVER_IDS.

Note

A deprecation warning is issued if SET GTID_MODE=ON is issued when any channel has existing server IDs set with IGNORE_SERVER_IDS. Before starting GTID-based replication, check for and clear all ignored server ID lists on the servers involved. The SHOW_SLAVE_STATUS statement displays the list of ignored IDs, if there is one. If you do receive the deprecation warning, you can still clear a list after gtid_mode=ON is set by issuing a CHANGE MASTER TO statement containing the IGNORE_SERVER_IDS option with an empty list.

Invoking CHANGE MASTER TO causes the previous values for MASTER_HOST, MASTER_PORT, MASTER_LOG_FILE, and MASTER_LOG_POS to be written to the error log, along with other information about the slave's state prior to execution.

CHANGE MASTER TO causes an implicit commit of an ongoing transaction. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

From MySQL 5.7, the strict requirement to execute STOP SLAVE prior to issuing any CHANGE MASTER TO statement (and START SLAVE afterward) is removed. Instead of depending on whether the slave is stopped, the behavior of CHANGE MASTER TO depends on the states of the slave SQL thread and

slave I/O threads; which of these threads is stopped or running now determines the options that can or cannot be used with a `CHANGE MASTER TO` statement at a given point in time. The rules for making this determination are listed here:

- If the SQL thread is stopped, you can execute `CHANGE MASTER TO` using any combination that is otherwise allowed of `RELAY_LOG_FILE`, `RELAY_LOG_POS`, and `MASTER_DELAY` options, even if the slave I/O thread is running. No other options may be used with this statement when the I/O thread is running.
- If the I/O thread is stopped, you can execute `CHANGE MASTER TO` using any of the options for this statement (in any allowed combination) *except* `RELAY_LOG_FILE`, `RELAY_LOG_POS`, `MASTER_DELAY`, or `MASTER_AUTO_POSITION = 1` even when the SQL thread is running.
- Both the SQL thread and the I/O thread must be stopped before issuing a `CHANGE MASTER TO` statement that employs `MASTER_AUTO_POSITION = 1`.

You can check the current state of the slave SQL and I/O threads using `SHOW SLAVE STATUS`.

For more information, see Section 17.4.8, “Switching Masters During Failover”.

If you are using statement-based replication and temporary tables, it is possible for a `CHANGE MASTER TO` statement following a `STOP SLAVE` statement to leave behind temporary tables on the slave. A warning (`ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO`) is now issued whenever this occurs. You can avoid this in such cases by making sure that the value of the `Slave_open_temp_tables` system status variable is equal to 0 prior to executing such a `CHANGE MASTER TO` statement.

`CHANGE MASTER TO` is useful for setting up a slave when you have the snapshot of the master and have recorded the master binary log coordinates corresponding to the time of the snapshot. After loading the snapshot into the slave to synchronize it with the master, you can run `CHANGE MASTER TO MASTER_LOG_FILE='log_name', MASTER_LOG_POS=log_pos` on the slave to specify the coordinates at which the slave should begin reading the master binary log.

The following example changes the master server the slave uses and establishes the master binary log coordinates from which the slave begins reading. This is used when you want to set up the slave to replicate the master:

```
CHANGE MASTER TO
  MASTER_HOST='master2.example.com',
  MASTER_USER='replication',
  MASTER_PASSWORD='password',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='master2-bin.001',
  MASTER_LOG_POS=4,
  MASTER_CONNECT_RETRY=10;
```

The next example shows an operation that is less frequently employed. It is used when the slave has relay log files that you want it to execute again for some reason. To do this, the master need not be reachable. You need only use `CHANGE MASTER TO` and start the SQL thread (`START SLAVE SQL_THREAD`):

```
CHANGE MASTER TO
  RELAY_LOG_FILE='slave-relay-bin.006',
  RELAY_LOG_POS=4025;
```

The following table shows the maximum permissible length for the string-valued options.

Option	Maximum Length
`MASTER_HOST`	255 (60 prior to MySQL 8.0.17)

Option	Maximum Length
MASTER_USER	96
MASTER_PASSWORD	32
MASTER_LOG_FILE	511
RELAY_LOG_FILE	511
MASTER_SSL_CA	511
MASTER_SSL_CAPATH	511
MASTER_SSL_CERT	511
MASTER_SSL_CRL	511
MASTER_SSL_CRLPATH	511
MASTER_SSL_KEY	511
MASTER_SSL_CIPHER	511
MASTER_TLS_VERSION	511
MASTER_PUBLIC_KEY_PATH	511
MASTER_COMPRESSION_ALGORITHMS	99

13.4.2.2 CHANGE REPLICATION FILTER Syntax

```
CHANGE REPLICATION FILTER filter[, filter]
 [, ...] [FOR CHANNEL channel]

filter:
    REPLICATE_DO_DB = (db_list)
  | REPLICATE_IGNORE_DB = (db_list)
  | REPLICATE_DO_TABLE = (tbl_list)
  | REPLICATE_IGNORE_TABLE = (tbl_list)
  | REPLICATE_WILD_DO_TABLE = (wild_tbl_list)
  | REPLICATE_WILD_IGNORE_TABLE = (wild_tbl_list)
  | REPLICATE_REWRITE_DB = (db_pair_list)

db_list:
    db_name[, db_name][, ...]

tbl_list:
    db_name.table_name[, db_name.table_name][, ...]
wild_tbl_list:
    'db_pattern.table_pattern'[, 'db_pattern.table_pattern'][, ...]

db_pair_list:
    (db_pair)[, (db_pair)][, ...]

db_pair:
    from_db, to_db
```

`CHANGE REPLICATION FILTER` sets one or more replication filtering rules on the slave in the same way as starting the slave `mysqld` with replication filtering options such as `--replicate-do-db` or `--replicate-wild-ignore-table`. Unlike the case with the server options, this statement does not require restarting the server to take effect, only that the slave SQL thread be stopped using `STOP SLAVE SQL_THREAD` first (and restarted with `START SLAVE SQL_THREAD` afterwards). `CHANGE REPLICATION FILTER` requires the `REPLICATION_SLAVE_ADMIN` or `SUPER` privilege. Use the `FOR CHANNEL channel` clause to make a replication filter specific to a replication channel, for example on a multi-source replication slave. Filters applied without a specific `FOR CHANNEL` clause are considered global filters, meaning that they are applied to all replication channels.

Note

Global replication filters cannot be set on a MySQL server instance that is configured for Group Replication, because filtering transactions on some servers would make the group unable to reach agreement on a consistent state. Channel specific replication filters can be set on replication channels that are not directly involved with Group Replication, such as where a group member also acts as a replication slave to a master that is outside the group. They cannot be set on the `group_replication_applier` or `group_replication_recovery` channels.

The following list shows the `CHANGE REPLICATION FILTER` options and how they relate to `--replicate-*` server options:

- `REPLICATE_DO_DB`: Include updates based on database name. Equivalent to `--replicate-do-db`.
- `REPLICATE_IGNORE_DB`: Exclude updates based on database name. Equivalent to `--replicate-ignore-db`.
- `REPLICATE_DO_TABLE`: Include updates based on table name. Equivalent to `--replicate-do-table`.
- `REPLICATE_IGNORE_TABLE`: Exclude updates based on table name. Equivalent to `--replicate-ignore-table`.
- `REPLICATE_WILD_DO_TABLE`: Include updates based on wildcard pattern matching table name. Equivalent to `--replicate-wild-do-table`.
- `REPLICATE_WILD_IGNORE_TABLE`: Exclude updates based on wildcard pattern matching table name. Equivalent to `--replicate-wild-ignore-table`.
- `REPLICATE_REWRITE_DB`: Perform updates on slave after substituting new name on slave for specified database on master. Equivalent to `--replicate-rewrite-db`.

The precise effects of `REPLICATE_DO_DB` and `REPLICATE_IGNORE_DB` filters are dependent on whether statement-based or row-based replication is in effect. See Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”, for more information.

Multiple replication filtering rules can be created in a single `CHANGE REPLICATION FILTER` statement by separating the rules with commas, as shown here:

```
CHANGE REPLICATION FILTER
    REPLICATE_DO_DB = (d1), REPLICATE_IGNORE_DB = (d2);
```

Issuing the statement just shown is equivalent to starting the slave `mysqld` with the options `--replicate-do-db=d1 --replicate-ignore-db=d2`.

On a multi-source replication slave, which uses multiple replication channels to process transaction from different sources, use the `FOR CHANNEL` *`channel`* clause to set a replication filter on a replication channel:

```
CHANGE REPLICATION FILTER REPLICATE_DO_DB = (d1) FOR CHANNEL channel_1;
```

This enables you to create a channel specific replication filter to filter out selected data from a source. When a `FOR CHANNEL` clause is provided, the replication filter statement acts on that slave replication channel removing any existing replication filter which has the same filter type as the specified replication filters, and replacing them with the specified filter. Filter types not explicitly listed in the statement are not modified. If issued against a slave replication channel which is not configured, the statement fails with an

ER_SLAVE_CONFIGURATION error. If issued against Group Replication channels, the statement fails with an ER_SLAVE_CHANNEL_OPERATION_NOT_ALLOWED error.

On a replication slave with multiple replication channels configured, issuing CHANGE REPLICATION FILTER with no FOR CHANNEL clause configures the replication filter for every configured slave replication channel, and for the global replication filters. For every filter type, if the filter type is listed in the statement, then any existing filter rules of that type are replaced by the filter rules specified in the most recently issued statement, otherwise the old value of the filter type is retained. For more information see Section 17.2.5.4, "Replication Channel Based Filters".

If the same filtering rule is specified multiple times, only the *last* such rule is actually used. For example, the two statements shown here have exactly the same effect, because the first REPLICATE_DO_DB rule in the first statement is ignored:

```
CHANGE REPLICATION FILTER
    REPLICATE_DO_DB = (db1, db2), REPLICATE_DO_DB = (db3, db4);

CHANGE REPLICATION FILTER
    REPLICATE_DO_DB = (db3, db4);
```

Caution

This behavior differs from that of the --replicate-* filter options where specifying the same option multiple times causes the creation of multiple filter rules.

Names of tables and database not containing any special characters need not be quoted. Values used with REPLICATION_WILD_TABLE and REPLICATION_WILD_IGNORE_TABLE are string expressions, possibly containing (special) wildcard characters, and so must be quoted. This is shown in the following example statements:

```
CHANGE REPLICATION FILTER
    REPLICATE_WILD_DO_TABLE = ('db1.old%');

CHANGE REPLICATION FILTER
    REPLICATE_WILD_IGNORE_TABLE = ('db1.new%', 'db2.new%');
```

Values used with REPLICATE_REWRITE_DB represent *pairs* of database names; each such value must be enclosed in parentheses. The following statement rewrites statements occurring on database db1 on the master to database db2 on the slave:

```
CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB = ((db1, db2));
```

The statement just shown contains two sets of parentheses, one enclosing the pair of database names, and the other enclosing the entire list. This is perhaps more easily seen in the following example, which creates two rewrite-db rules, one rewriting database dbA to dbB, and one rewriting database dbC to dbD:

```
CHANGE REPLICATION FILTER
  REPLICATE_REWRITE_DB = ((dbA, dbB), (dbC, dbD));
```

The CHANGE REPLICATION FILTER statement replaces replication filtering rules only for the filter types and replication channels affected by the statement, and leaves other rules and channels unchanged. If you want to unset all filters of a given type, set the filter's value to an explicitly empty list, as shown in this example, which removes all existing REPLICATE_DO_DB and REPLICATE_IGNORE_DB rules:

```
```

```
CHANGE REPLICATION FILTER
    REPLICATE_DO_DB = (), REPLICATE_IGNORE_DB = ();
```

Setting a filter to empty in this way removes all existing rules, does not create any new ones, and does not restore any rules set at mysqld startup using `--replicate-*` options on the command line or in the configuration file.

The `RESET SLAVE ALL` statement removes channel specific replication filters that were set on channels deleted by the statement. When the deleted channel or channels are recreated, any global replication filters specified for the slave are copied to them, and no channel specific replication filters are applied.

For more information, see Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”.

13.4.2.3 MASTER_POS_WAIT() Syntax

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos [, timeout][, channel])
```

This is actually a function, not a statement. It is used to ensure that the slave has read and executed events up to a given position in the master's binary log. See Section 12.24, “Miscellaneous Functions”, for a full description.

13.4.2.4 RESET SLAVE Syntax

```
RESET SLAVE [ALL] [channel_option]

channel_option:
    FOR CHANNEL channel
```

`RESET SLAVE` makes the slave forget its replication position in the master's binary log. This statement is meant to be used for a clean start: It clears the master info and relay log info repositories, deletes all the relay log files, and starts a new relay log file. It also resets to 0 the replication delay specified with the `MASTER_DELAY` option to `CHANGE MASTER TO`.

Note

All relay log files are deleted, even if they have not been completely executed by the slave SQL thread. (This is a condition likely to exist on a replication slave if you have issued a `STOP SLAVE` statement or if the slave is highly loaded.)

For a server where GTIDs are in use (`gtid_mode` is `ON`), issuing `RESET SLAVE` has no effect on the GTID execution history. The statement does not change the values of `gtid_executed` or `gtid_purged`, or the `mysql.gtid_executed` table. If you need to reset the GTID execution history, use `RESET MASTER`, even if the GTID-enabled server is a replication slave where binary logging is disabled.

`RESET SLAVE` requires the `RELOAD` privilege.

To use `RESET SLAVE`, the slave replication threads must be stopped, so on a running slave use `STOP SLAVE` before issuing `RESET SLAVE`. To use `RESET SLAVE` on a Group Replication group member, the member status must be `OFFLINE`, meaning that the plugin is loaded but the member does not currently belong to any group. A group member can be taken offline by using a `STOP GROUP REPLICATION` statement.

The optional `FOR CHANNEL channel` clause enables you to name which replication channel the statement applies to. Providing a `FOR CHANNEL channel` clause applies the `RESET SLAVE` statement to a specific replication channel. Combining a `FOR CHANNEL channel` clause with the `ALL` option deletes the specified channel. If no channel is named and no extra channels exist, the statement applies to the

default channel. Issuing a `RESET SLAVE ALL` statement without a `FOR CHANNEL channel` clause when multiple replication channels exist deletes *all* replication channels and recreates only the default channel. See Section 17.2.3, “Replication Channels” for more information.

`RESET SLAVE` does not change any replication connection parameters such as the master host, master port, master user, or master password. `RESET SLAVE` also does not affect the `PRIVILEGE_CHECKS_USER` account. If you want to reset the connection parameters intentionally, you need to use `RESET SLAVE ALL`, which clears the connection parameters, including the `MASTER_USER` and `PRIVILEGE_CHECKS_USER` accounts. `RESET SLAVE ALL` also clears the `IGNORE_SERVER_IDS` list set by `CHANGE MASTER TO`. When you have used `RESET SLAVE ALL`, you must issue a `CHANGE MASTER TO` statement after the server start to specify new connection parameters.

In the event of a server crash or deliberate restart after issuing `RESET SLAVE` but before issuing `START SLAVE`, retention of the replication connection parameters depends on the repository used for the master info log:

- From MySQL 8.0.13, when `master_info_repository=TABLE` is set on the server (which is the default from MySQL 8.0), replication connection parameters are preserved in the crash-safe `InnoDB` table `mysql.slave_master_info` as part of the `RESET SLAVE` operation. They are also retained in memory. In the event of a server crash or deliberate restart after issuing `RESET SLAVE` but before issuing `START SLAVE`, the replication connection parameters are retrieved from the table and reused for the new connection.
- When `master_info_repository=FILE` is set on the server, replication connection parameters are only retained in memory. If the slave `mysqld` is restarted immediately after issuing `RESET SLAVE` due to a server crash or deliberate restart, the connection parameters are lost. In that case, you must issue a `CHANGE MASTER TO` statement after the server start to respecify the connection parameters before issuing `START SLAVE`.

`RESET SLAVE` does not change any replication filter settings (such as `--replicate-ignore-table`) for channels affected by the statement. However, `RESET SLAVE ALL` removes the replication filters that were set on the channels deleted by the statement. When the deleted channel or channels are recreated, any global replication filters specified for the slave are copied to them, and no channel specific replication filters are applied. For more information see Section 17.2.5.4, “Replication Channel Based Filters”.

`RESET SLAVE` causes an implicit commit of an ongoing transaction. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

If the slave SQL thread was in the middle of replicating temporary tables when it was stopped, and `RESET SLAVE` is issued, these replicated temporary tables are deleted on the slave.

`RESET SLAVE` does not reset the heartbeat period (`Slave_heartbeat_period`) or `SSL_VERIFY_SERVER_CERT`.

Note

When used on an NDB Cluster replication slave SQL node, `RESET SLAVE` clears the `mysql.ndb_apply_status` table. You should keep in mind when using this statement that `ndb_apply_status` uses the `NDB` storage engine and so is shared by all SQL nodes attached to the slave cluster.

You can override this behavior by issuing `SET GLOBAL @@ndb_clear_apply_status=OFF` prior to executing `RESET SLAVE`, which keeps the slave from purging the `ndb_apply_status` table in such cases.

13.4.2.5 SET GLOBAL sql_slave_skip_counter Syntax

```
SET GLOBAL sql_slave_skip_counter = N
```

This statement skips the next *N* events from the master. This is useful for recovering from replication stops caused by a statement.

This statement is valid only when the slave threads are not running. Otherwise, it produces an error.

When using this statement, it is important to understand that the binary log is actually organized as a sequence of groups known as *event groups*. Each event group consists of a sequence of events.

- For transactional tables, an event group corresponds to a transaction.
- For nontransactional tables, an event group corresponds to a single SQL statement.

Note

A single transaction can contain changes to both transactional and nontransactional tables.

When you use `SET GLOBAL sql_slave_skip_counter` to skip events and the result is in the middle of a group, the slave continues to skip events until it reaches the end of the group. Execution then starts with the next event group.

13.4.2.6 START SLAVE Syntax

```
START SLAVE [thread_types] [until_option] [connection_options] [channel_option]

thread_types:
    [thread_type [, thread_type] ... ]

thread_type:
    IO_THREAD | SQL_THREAD

until_option:
    UNTIL {   {SQL_BEFORE_GTIDS | SQL_AFTER_GTIDS} = gtid_set
          |   MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
          |   RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos
          |   SQL_AFTER_MTS_GAPS  }

connection_options:
    [USER='user_name'] [PASSWORD='user_pass'] [DEFAULT_AUTH='plugin_name'] [PLUGIN_DIR='plugin_dir']


channel_option:
    FOR CHANNEL channel

gtid_set:
    uuid_set [, uuid_set] ...
    | ''

uuid_set:
    uuid:interval[:interval]...

uuid:
    hhhhhhhh-hhhh-hhhh-hhhh-hhhhhhhhhhhh

h:
    [0-9,A-F]

interval:
    n[-n]
```

```
    (n >= 1)
```

`START SLAVE` with no *`thread_type`* options starts both of the slave threads. The I/O thread reads events from the master server and stores them in the relay log. The SQL thread reads events from the relay log and executes them. `START SLAVE` requires the `REPLICATION_SLAVE_ADMIN` or `SUPER` privilege.

If `START SLAVE` succeeds in starting the slave threads, it returns without any error. However, even in that case, it might be that the slave threads start and then later stop (for example, because they do not manage to connect to the master or read its binary log, or some other problem). `START SLAVE` does not warn you about this. You must check the slave's error log for error messages generated by the slave threads, or check that they are running satisfactorily with `SHOW SLAVE STATUS`.

`START SLAVE` causes an implicit commit of an ongoing transaction. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

`gtid_next` must be set to `AUTOMATIC` before issuing this statement.

The optional `FOR CHANNEL` *`channel`* clause enables you to name which replication channel the statement applies to. Providing a `FOR CHANNEL` *`channel`* clause applies the `START SLAVE` statement to a specific replication channel. If no clause is named and no extra channels exist, the statement applies to the default channel. If a `START SLAVE` statement does not have a channel defined when using multiple channels, this statement starts the specified threads for all channels. This statement is disallowed for the `group_replication_recovery` channel. See Section 17.2.3, “Replication Channels” for more information.

MySQL supports pluggable user-password authentication with `START SLAVE` with the `USER`, `PASSWORD`, `DEFAULT_AUTH` and `PLUGIN_DIR` options, as described in the following list:

- `USER`: User name. Cannot be set to an empty or null string, or left unset if `PASSWORD` is used.
- `PASSWORD`: Password.
- `DEFAULT_AUTH`: Name of plugin; default is MySQL native authentication.
- `PLUGIN_DIR`: Location of plugin.

You cannot use the `SQL_THREAD` option when specifying any of `USER`, `PASSWORD`, `DEFAULT_AUTH`, or `PLUGIN_DIR`, unless the `IO_THREAD` option is also provided.

See Section 6.2.17, “Pluggable Authentication”, for more information.

If an insecure connection is used with any these options, the server issues the warning `Sending passwords in plain text without SSL/TLS is extremely insecure`.

`START SLAVE ... UNTIL` supports two additional options for use with global transaction identifiers (GTIDs) (see Section 17.1.3, “Replication with Global Transaction Identifiers”). Each of these takes a set of one or more global transaction identifiers *`gtid_set`* as an argument (see GTID Sets, for more information).

When no *`thread_type`* is specified, `START SLAVE UNTIL SQL_BEFORE_GTIDS` causes the slave SQL thread to process transactions until it has reached the *first* transaction whose GTID is listed in the *`gtid_set`*. `START SLAVE UNTIL SQL_AFTER_GTIDS` causes the slave threads to process all transactions until the *`last`* transaction in the *`gtid_set`* has been processed by both threads. In other words, `START SLAVE UNTIL SQL_BEFORE_GTIDS` causes the slave SQL thread to process all transactions occurring before the first GTID in the *`gtid_set`* is reached, and `START SLAVE UNTIL SQL_AFTER_GTIDS` causes the slave threads to handle all transactions, including those whose GTIDs are found in *`gtid_set`*, until each has encountered a transaction whose GTID is not part of the set.

`SQL_BEFORE_GTIDS` and `SQL_AFTER_GTIDS` each support the `SQL_THREAD` and `IO_THREAD` options, although using `IO_THREAD` with them currently has no effect.

For example, `START SLAVE SQL_THREAD UNTIL SQL_BEFORE_GTIDS = 3E11FA47-71CA-11E1-9E33-C80AA9429562:11-56` causes the slave SQL thread to process all transactions originating from the master whose `server_uuid` is `3E11FA47-71CA-11E1-9E33-C80AA9429562` until it encounters the transaction having sequence number 11; it then stops without processing this transaction. In other words, all transactions up to and including the transaction with sequence number 10 are processed. Executing `START SLAVE SQL_THREAD UNTIL SQL_AFTER_GTIDS = 3E11FA47-71CA-11E1-9E33-C80AA9429562:11-56`, on the other hand, would cause the slave SQL thread to obtain all transactions just mentioned from the master, including all of the transactions having the sequence numbers 11 through 56, and then to stop without processing any additional transactions; that is, the transaction having sequence number 56 would be the last transaction fetched by the slave SQL thread.

When using a multithreaded slave with `slave_preserve_commit_order=0` set, there is a chance of gaps in the sequence of transactions that have been executed from the relay log in the following cases:

- killing the coordinator thread
- after an error occurs in the applier threads
- `mysqld` shuts down unexpectedly

Use the `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` statement to cause a multithreaded slave's worker threads to only run until no more gaps are found in the relay log, and then to stop. This statement can take an `SQL_THREAD` option, but the effects of the statement remain unchanged. It has no effect on the slave I/O thread (and cannot be used with the `IO_THREAD` option).

Issuing `START SLAVE` on a multithreaded slave with gaps in the sequence of transactions executed from the relay log generates a warning. In such a situation, the solution is to use `START SLAVE UNTIL SQL_AFTER_MTS_GAPS`, then issue `RESET SLAVE` to remove any remaining relay logs. See Section 17.5.1.33, “Replication and Transaction Inconsistencies” for more information.

To change a failed multithreaded slave to single-threaded mode, you can issue the following series of statements, in the order shown:

```
START SLAVE UNTIL SQL_AFTER_MTS_GAPS;

SET @@GLOBAL.slave_parallel_workers = 0;

START SLAVE SQL_THREAD;
```

Note

It is possible to view the entire text of a running `START SLAVE ...` statement, including any `USER` or `PASSWORD` values used, in the output of `SHOW PROCESSLIST`. This is also true for the text of a running `CHANGE MASTER TO` statement, including any values it employs for `MASTER_USER` or `MASTER_PASSWORD`.

`START SLAVE` sends an acknowledgment to the user after both the I/O thread and the SQL thread have started. However, the I/O thread may not yet have connected. For this reason, a successful `START SLAVE` causes `SHOW SLAVE STATUS` to show `Slave_SQL_Running=Yes`, but this does not guarantee that `Slave_IO_Running=Yes` (because `Slave_IO_Running=Yes` only if the I/O thread is running *and connected*). For more information, see Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”, and Section 17.1.7.1, “Checking Replication Status”.

You can add IO_THREAD and SQL_THREAD options to the statement to name which of the threads to start. The SQL_THREAD option is disallowed when specifying any of USER, PASSWORD, DEFAULT_AUTH, or PLUGIN_DIR, unless the IO_THREAD option is also provided.

An UNTIL clause (*until_option*, in the preceding grammar) may be added to specify that the slave should start and run until the SQL thread reaches a given point in the master binary log, specified by the MASTER_LOG_POS and MASTER_LOG_FILE options, or a given point in the slave relay log, indicated with the RELAY_LOG_POS and RELAY_LOG_FILE options. When the SQL thread reaches the point specified, it stops. If the SQL_THREAD option is specified in the statement, it starts only the SQL thread. Otherwise, it starts both slave threads. If the SQL thread is running, the UNTIL clause is ignored and a warning is issued. You cannot use an UNTIL clause with the IO_THREAD option.

It is also possible with START SLAVE UNTIL to specify a stop point relative to a given GTID or set of GTIDs using one of the options SQL_BEFORE_GTIDS or SQL_AFTER_GTIDS, as explained previously in this section. When using one of these options, you can specify SQL_THREAD, IO_THREAD, both of these, or neither of them. If you specify only SQL_THREAD, then only the slave SQL thread is affected by the statement; if only IO_THREAD is used, then only the slave I/O is affected. If both SQL_THREAD and IO_THREAD are used, or if neither of them is used, then both the SQL and I/O threads are affected by the statement.

For an UNTIL clause, you must specify any one of the following:

- *Both* a log file name and a position in that file
- *Either* of SQL_BEFORE_GTIDS or SQL_AFTER_GTIDS
- SQL_AFTER_MTS_GAPS

Do not mix master and relay log options. Do not mix log file options with GTID options.

The UNTIL clause is not supported for multithreaded slaves except when also using SQL_AFTER_MTS_GAPS. If UNTIL is used on a multithreaded slave without SQL_AFTER_MTS_GAPS, the slave operates in single-threaded (sequential) mode for replication until the point specified by the UNTIL clause is reached.

Any UNTIL condition is reset by a subsequent STOP SLAVE statement, a START SLAVE statement that includes no UNTIL clause, or a server restart.

When specifying a log file and position, you can use the IO_THREAD option with START SLAVE ... UNTIL even though only the SQL thread is affected by this statement. The IO_THREAD option is ignored in such cases. The preceding restriction does not apply when using one of the GTID options (SQL_BEFORE_GTIDS and SQL_AFTER_GTIDS); the GTID options support both SQL_THREAD and IO_THREAD, as explained previously in this section.

The UNTIL clause can be useful for debugging replication, or to cause replication to proceed until just before the point where you want to avoid having the slave replicate an event. For example, if an unwise DROP TABLE statement was executed on the master, you can use UNTIL to tell the slave to execute up to that point but no farther. To find what the event is, use mysqlbinlog with the master binary log or slave relay log, or by using a SHOW BINLOG EVENTS statement.

If you are using UNTIL to have the slave process replicated queries in sections, it is recommended that you start the slave with the --skip-slave-start option to prevent the SQL thread from running when the slave server starts. It is probably best to use this option in an option file rather than on the command line, so that an unexpected server restart does not cause it to be forgotten.

The SHOW SLAVE STATUS statement includes output fields that display the current values of the UNTIL condition.

In very old versions of MySQL (before 4.0.5), this statement was called `SLAVE START`. That syntax now produces an error.

13.4.2.7 STOP SLAVE Syntax

```
STOP SLAVE [thread_types] [channel_option]

thread_types:
    [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD

channel_option:
    FOR CHANNEL channel
```

Stops the slave threads. `STOP SLAVE` requires the `REPLICATION_SLAVE_ADMIN` or `SUPER` privilege. Recommended best practice is to execute `STOP SLAVE` on the slave before stopping the slave server (see Section 5.1.17, “The Server Shutdown Process”, for more information).

When using the row-based logging format: You should execute `STOP SLAVE` or `STOP SLAVE SQL_THREAD` on the slave prior to shutting down the slave server if you are replicating any tables that use a nontransactional storage engine (see the *Note* later in this section).

Like `START SLAVE`, this statement may be used with the `IO_THREAD` and `SQL_THREAD` options to name the thread or threads to be stopped.

`STOP SLAVE` causes an implicit commit of an ongoing transaction. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

`gtid_next` must be set to `AUTOMATIC` before issuing this statement.

You can control how long `STOP SLAVE` waits before timing out by setting the `rpl_stop_slave_timeout` system variable. This can be used to avoid deadlocks between `STOP SLAVE` and other slave SQL statements using different client connections to the slave. When the timeout value is reached, the issuing client returns an error message and stops waiting, but the `STOP SLAVE` instruction remains in effect. Once the slave threads are no longer busy, the `STOP SLAVE` statement is executed and the slave stops.

Some `CHANGE MASTER TO` statements are allowed while the slave is running, depending on the states of the slave SQL and I/O threads. However, using `STOP SLAVE` prior to executing `CHANGE MASTER TO` in such cases is still supported. See Section 13.4.2.1, “CHANGE MASTER TO Syntax”, and Section 17.4.8, “Switching Masters During Failover”, for more information.

The optional `FOR CHANNEL channel` clause enables you to name which replication channel the statement applies to. Providing a `FOR CHANNEL channel` clause applies the `STOP SLAVE` statement to a specific replication channel. If no channel is named and no extra channels exist, the statement applies to the default channel. If a `STOP SLAVE` statement does not name a channel when using multiple channels, this statement stops the specified threads for all channels. This statement cannot be used with the `group_replication_recovery` channel. See Section 17.2.3, “Replication Channels” for more information.

When using statement-based replication: changing the master while it has open temporary tables is potentially unsafe. This is one of the reasons why statement-based replication of temporary tables is not recommended. You can find out whether there are any temporary tables on the slave by checking the value of `Slave_open_temp_tables`; when using statement-based replication, this value should be 0 before executing `CHANGE MASTER TO`. If there are any temporary tables open

on the slave, issuing a `CHANGE MASTER TO` statement after issuing a `STOP SLAVE` causes an `ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO` warning.

When using a multithreaded slave (`slave_parallel_workers` is a nonzero value), any gaps in the sequence of transactions executed from the relay log are closed as part of stopping the worker threads. If the slave is stopped unexpectedly (for example due to an error in a worker thread, or another thread issuing `KILL`) while a `STOP SLAVE` statement is executing, the sequence of executed transactions from the relay log may become inconsistent. See Section 17.5.1.33, "Replication and Transaction Inconsistencies" for more information.

If the current replication event group has modified one or more nontransactional tables, STOP SLAVE waits for up to 60 seconds for the event group to complete, unless you issue a `KILL QUERY` or `KILL CONNECTION` statement for the slave SQL thread. If the event group remains incomplete after the timeout, an error message is logged.

13.4.3 SQL Statements for Controlling Group Replication

This section provides information about the statements used for controlling group replication.

13.4.3.1 START GROUP_REPLICATION Syntax

```
START GROUP_REPLICATION
```

Starts group replication. This statement requires the `GROUP_REPLICATION_ADMIN` or `SUPER` privilege. If `super_read_only=ON` and the member should join as a primary, `super_read_only` is set to `OFF` once Group Replication successfully starts.

13.4.3.2 STOP GROUP_REPLICATION Syntax

```
STOP GROUP_REPLICATION
```

Stops Group Replication. This statement requires the `GROUP_REPLICATION_ADMIN` or `SUPER` privilege. As soon as you issue `STOP GROUP_REPLICATION` the member is set to `super_read_only=ON`, which ensures that no writes can be made to the member while Group Replication stops. Any other replication channels running on the member are also stopped.

Warning

Use this statement with extreme caution because it removes the server instance from the group, meaning it is no longer protected by Group Replication's consistency guarantee mechanisms. To be completely safe, ensure that your applications can no longer connect to the instance before issuing this statement to avoid any chance of stale reads.

13.4.3.3 Function which Configures Group Replication Primary

The following function enables you to configure which member of a single-primary replication group is the primary.

- `group_replication_set_as_primary()`

  Appoints a specific member of the group as the new primary, overriding any election process. Pass in *`member_uuid`* which is the `server_uuid` of the member that you want to become the new primary. Must be issued on a member of a replication group running in single-primary mode.

  Syntax:

```
STRING group_replication_set_as_primary(member_uuid)
```

Return value:

A string containing the result of the operation, for example whether it was successful or not.

Example:

```
SELECT group_replication_set_as_primary(member_uuid)
```

For more information, see Section 18.4.1.1, “Changing a Group's Primary Member”

13.4.3.4 Functions which Configure the Group Replication Mode

The following functions enable you to control the mode which a replication group is running in, either single-primary or multi-primary mode.

- `group_replication_switch_to_single_primary_mode()`

  Changes a group running in multi-primary mode to single-primary mode, without the need to stop Group Replication. Must be issued on a member of a replication group running in multi-primary mode. When you change to single-primary mode, strict consistency checks are also disabled on all group members, as required in single-primary mode (`group_replication_enforce_update_everywhere_checks=OFF`).

  Syntax:

  ```
  STRING group_replication_switch_to_single_primary_mode([str])
  ```

  Arguments:

  - *`str`*: A string containing the UUID of a member of the group which should become the new single primary. Other members of the group become secondaries.

  Return value:

  A string containing the result of the operation, for example whether it was successful or not.

  Example:

  ```
  SELECT group_replication_switch_to_single_primary_mode(member_uuid);
  ```

  For more information, see Section 18.4.1.2, “Changing a Group's Mode”

- `group_replication_switch_to_multi_primary_mode()`

  Changes a group running in single-primary mode to multi-primary mode. Must be issued on a member of a replication group running in single-primary mode.

  Syntax:

  ```
  STRING group_replication_switch_to_multi_primary_mode()
  ```

  This function has no parameters.

  Return value:

  A string containing the result of the operation, for example whether it was successful or not.

Example:

```
SELECT group_replication_switch_to_multi_primary_mode()
```

All members which belong to the group become primaries.

For more information, see Section 18.4.1.2, “Changing a Group's Mode”

13.4.3.5 Functions to Inspect and Configure the Maximum Consensus Instances of a Group

The following functions enable you to inspect and configure the maximum number of consensus instances that a group can execute in parallel.

- `group_replication_get_write_concurrency()`

  Check the maximum number of consensus instances that a group can execute in parallel.

  Syntax:

  ```
  STRING group_replication_get_write_concurrency()
  ```

  This function has no parameters.

  Return value:

  Any resulting error as a string.

  Example:

  ```
  SELECT group_replication_get_write_concurrency()
  ```

  For more information, see Section 18.4.1.3, “Using Group Replication Group Write Consensus”.

- `group_replication_set_write_concurrency()`

  Configures the maximum number of consensus instances that a group can execute in parallel. The `GROUP_REPLICATION_ADMIN` privilege is required to use this UDF.

  Syntax:

  ```
  STRING group_replication_set_write_concurrency(instances)
  ```

  Arguments:

  - *`members`*: Sets the maximum number of consensus instances that a group can execute in parallel. Default value is 10, valid values are integers in the range of 10 to 200.

  Return value:

  Any resulting error as a string.

  Example:

  ```
  SELECT group_replication_set_write_concurrency(instances);
  ```

  For more information, see Section 18.4.1.3, “Using Group Replication Group Write Consensus”.

13.4.3.6 Functions to Inspect and Set the Group Replication Communication Protocol Version

The following functions enable you to inspect and configure the Group Replication communication protocol version that is used by a replication group.

- `group_replication_get_communication_protocol()`

  Inspect the Group Replication communication protocol version that is currently in use for a group.

  Syntax:

```
STRING group_replication_get_communication_protocol()
```

  This function has no parameters.

  Return value:

  The oldest MySQL Server version that can join this group and use the group's communication protocol. Versions from MySQL 5.7.14 allow compression of messages, and versions from MySQL 8.0.16 also allow fragmentation of messages. Note that the `group_replication_get_communication_protocol()` UDF returns the minimum MySQL version that the group supports, which might differ from the version number that was passed to the `group_replication_set_communication_protocol()` UDF, and from the MySQL Server version that is installed on the member where you use the UDF.

  If the protocol cannot be inspected because this server instance does not belong to a replication group, an error is returned as a string.

  Example:

```
SELECT group_replication_get_communication_protocol();
+-----------------------------------------------+
| group_replication_get_communication_protocol() |
+-----------------------------------------------+
| 8.0.16                                        |
+-----------------------------------------------+
```

  For more information, see Section 18.4.1.4, “Setting a Group's Communication Protocol Version”.

- `group_replication_set_communication_protocol()`

  Downgrade the Group Replication communication protocol version of a group so that members at earlier releases can join, or upgrade the Group Replication communication protocol version of a group after upgrading MySQL Server on all members. The `GROUP_REPLICATION_ADMIN` privilege is required to use this UDF, and all existing group members must be online when you issue the statement, with no loss of majority.

  Note

  For MySQL InnoDB cluster, the communication protocol version is managed automatically whenever the cluster topology is changed using AdminAPI operations. You do not have to use these UDFs yourself for an InnoDB cluster.

  Syntax:

```
```

```
STRING group_replication_set_communication_protocol(version)
```

Arguments:

- *`version`*: For a downgrade, specify the MySQL Server version of the prospective group member that has the oldest installed server version. In this case, the command makes the group fall back to a communication protocol compatible with that server version if possible. The minimum server version that you can specify is MySQL 5.7.14. For an upgrade, specify the new MySQL Server version to which the existing group members have been upgraded.

Return value:

A string containing the result of the operation, for example whether it was successful or not.

Example:

```
SELECT group_replication_set_communication_protocol("5.7.25");
```

For more information, see Section 18.4.1.4, “Setting a Group's Communication Protocol Version”.

13.5 Prepared SQL Statement Syntax

MySQL 8.0 provides support for server-side prepared statements. This support takes advantage of the efficient client/server binary protocol. Using prepared statements with placeholders for parameter values has the following benefits:

- Less overhead for parsing the statement each time it is executed. Typically, database applications process large volumes of almost-identical statements, with only changes to literal or variable values in clauses such as `WHERE` for queries and deletes, `SET` for updates, and `VALUES` for inserts.
- Protection against SQL injection attacks. The parameter values can contain unescaped SQL quote and delimiter characters.

Prepared Statements in Application Programs

You can use server-side prepared statements through client programming interfaces, including the MySQL C API client library or MySQL Connector/C for C programs, MySQL Connector/J for Java programs, and MySQL Connector/NET for programs using .NET technologies. For example, the C API provides a set of function calls that make up its prepared statement API. See Section 28.7.8, “C API Prepared Statements”. Other language interfaces can provide support for prepared statements that use the binary protocol by linking in the C client library, one example being the `mysqli` extension, available in PHP 5.0 and later.

Prepared Statements in SQL Scripts

An alternative SQL interface to prepared statements is available. This interface is not as efficient as using the binary protocol through a prepared statement API, but requires no programming because it is available directly at the SQL level:

- You can use it when no programming interface is available to you.
- You can use it from any program that can send SQL statements to the server to be executed, such as the `mysql` client program.
- You can use it even if the client is using an old version of the client library, as long as you connect to a server running MySQL 4.1 or higher.

SQL syntax for prepared statements is intended to be used for situations such as these:

- To test how prepared statements work in your application before coding it.
- To use prepared statements when you do not have access to a programming API that supports them.
- To interactively troubleshoot application issues with prepared statements.
- To create a test case that reproduces a problem with prepared statements, so that you can file a bug report.

PREPARE, EXECUTE, and DEALLOCATE PREPARE Statements

SQL syntax for prepared statements is based on three SQL statements:

- `PREPARE` prepares a statement for execution (see Section 13.5.1, “PREPARE Syntax”).
- `EXECUTE` executes a prepared statement (see Section 13.5.2, “EXECUTE Syntax”).
- `DEALLOCATE PREPARE` releases a prepared statement (see Section 13.5.3, “DEALLOCATE PREPARE Syntax”).

The following examples show two equivalent ways of preparing a statement that computes the hypotenuse of a triangle given the lengths of the two sides.

The first example shows how to create a prepared statement by using a string literal to supply the text of the statement:

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|          5 |
+------------+
mysql> DEALLOCATE PREPARE stmt1;
```

The second example is similar, but supplies the text of the statement as a user variable:

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|         10 |
+------------+
mysql> DEALLOCATE PREPARE stmt2;
```

Here is an additional example that demonstrates how to choose the table on which to perform a query at runtime, by storing the name of the table as a user variable:

```
mysql> USE test;
mysql> CREATE TABLE t1 (a INT NOT NULL);
mysql> INSERT INTO t1 VALUES (4), (8), (11), (32), (80);
```

```
mysql> SET @table = 't1';
mysql> SET @s = CONCAT('SELECT * FROM ', @table);

mysql> PREPARE stmt3 FROM @s;
mysql> EXECUTE stmt3;
+----+
| a  |
+----+
|  4 |
|  8 |
| 11 |
| 32 |
| 80 |
+----+

mysql> DEALLOCATE PREPARE stmt3;
```

A prepared statement is specific to the session in which it was created. If you terminate a session without deallocating a previously prepared statement, the server deallocates it automatically.

A prepared statement is also global to the session. If you create a prepared statement within a stored routine, it is not deallocated when the stored routine ends.

To guard against too many prepared statements being created simultaneously, set the `max_prepared_stmt_count` system variable. To prevent the use of prepared statements, set the value to 0.

SQL Syntax Allowed in Prepared Statements

The following SQL statements can be used as prepared statements:

```
ALTER TABLE
ALTER USER
ANALYZE TABLE
CACHE INDEX
CALL
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
COMMIT
{CREATE | DROP} INDEX
{CREATE | RENAME | DROP} DATABASE
{CREATE | DROP} TABLE
{CREATE | RENAME | DROP} USER
{CREATE | DROP} VIEW
DELETE
DO
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
  | LOGS | STATUS | MASTER | SLAVE | USER_RESOURCES}
GRANT
INSERT
INSTALL PLUGIN
KILL
LOAD INDEX INTO CACHE
OPTIMIZE TABLE
RENAME TABLE
REPAIR TABLE
REPLACE
RESET {MASTER | SLAVE}
REVOKE
SELECT
SET
SHOW {WARNINGS | ERRORS}
```

```
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
TRUNCATE TABLE
UNINSTALL PLUGIN
UPDATE
```

For compliance with the SQL standard, which states that diagnostics statements are not preparable, MySQL does not support the following as prepared statements:

- `SHOW WARNINGS`, `SHOW COUNT(*) WARNINGS`
- `SHOW ERRORS`, `SHOW COUNT(*) ERRORS`
- Statements containing any reference to the `warning_count` or `error_count` system variable.

Other statements are not supported in MySQL 8.0.

Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. Exceptions are noted in Section 24.8, “Restrictions on Stored Programs”.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.10.3, “Caching of Prepared Statements and Stored Programs”.

Placeholders can be used for the arguments of the `LIMIT` clause when using prepared statements. See Section 13.2.10, “SELECT Syntax”.

In prepared `CALL` statements used with `PREPARE` and `EXECUTE`, placeholder support for `OUT` and `INOUT` parameters is available beginning with MySQL 8.0. See Section 13.2.1, “CALL Syntax”, for an example and a workaround for earlier versions. Placeholders can be used for `IN` parameters regardless of version.

SQL syntax for prepared statements cannot be used in nested fashion. That is, a statement passed to `PREPARE` cannot itself be a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

SQL syntax for prepared statements is distinct from using prepared statement API calls. For example, you cannot use the `mysql_stmt_prepare()` C API function to prepare a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

SQL syntax for prepared statements can be used within stored procedures, but not in stored functions or triggers. However, a cursor cannot be used for a dynamic statement that is prepared and executed with `PREPARE` and `EXECUTE`. The statement for a cursor is checked at cursor creation time, so the statement cannot be dynamic.

SQL syntax for prepared statements does not support multi-statements (that is, multiple statements within a single string separated by `;` characters).

To write C programs that use the `CALL` SQL statement to execute stored procedures that contain prepared statements, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). For additional information, see Section 13.2.1, “CALL Syntax”.

13.5.1 PREPARE Syntax

```
PREPARE stmt_name FROM preparable_stmt
```

The PREPARE statement prepares a SQL statement and assigns it a name, *stmt_name*, by which to refer to the statement later. The prepared statement is executed with EXECUTE and released with DEALLOCATE PREPARE. For examples, see Section 13.5, “Prepared SQL Statement Syntax”.

Statement names are not case-sensitive. *preparable_stmt* is either a string literal or a user variable that contains the text of the SQL statement. The text must represent a single statement, not multiple statements. Within the statement, ? characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The ? characters should not be enclosed within quotation marks, even if you intend to bind them to string values. Parameter markers can be used only where data values should appear, not for SQL keywords, identifiers, and so forth.

If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name exists.

The scope of a prepared statement is the session within which it is created, which as several implications:

- A prepared statement created in one session is not available to other sessions.
- When a session ends, whether normally or abnormally, its prepared statements no longer exist. If auto-reconnect is enabled, the client is not notified that the connection was lost. For this reason, clients may wish to disable auto-reconnect. See Section 28.7.28, “C API Automatic Reconnection Control”.
- A prepared statement created within a stored program continues to exist after the program finishes executing and can be executed outside the program later.
- A statement prepared in stored program context cannot refer to stored procedure or function parameters or local variables because they go out of scope when the program ends and would be unavailable were the statement to be executed later outside the program. As a workaround, refer instead to user-defined variables, which also have session scope; see Section 9.4, “User-Defined Variables”.

13.5.2 EXECUTE Syntax

```
EXECUTE stmt_name
    [USING @var_name [, @var_name] ...]
```

After preparing a statement with PREPARE, you execute it with an EXECUTE statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a USING clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the USING clause must name exactly as many variables as the number of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing different variables to it or setting the variables to different values before each execution.

For examples, see Section 13.5, “Prepared SQL Statement Syntax”.

13.5.3 DEALLOCATE PREPARE Syntax

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

To deallocate a prepared statement produced with `PREPARE`, use a `DEALLOCATE PREPARE` statement that refers to the prepared statement name. Attempting to execute a prepared statement after deallocating it results in an error. If too many prepared statements are created and not deallocated by either the `DEALLOCATE PREPARE` statement or the end of the session, you might encounter the upper limit enforced by the `max_prepared_stmt_count` system variable.

For examples, see Section 13.5, “Prepared SQL Statement Syntax”.

13.6 Compound-Statement Syntax

This section describes the syntax for the `BEGIN ... END` compound statement and other statements that can be used in the body of stored programs: Stored procedures and functions, triggers, and events. These objects are defined in terms of SQL code that is stored on the server for later invocation (see Chapter 24, *Stored Objects*).

A compound statement is a block that can contain other blocks; declarations for variables, condition handlers, and cursors; and flow control constructs such as loops and conditional tests.

13.6.1 BEGIN ... END Compound-Statement Syntax

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]
```

`BEGIN ... END` syntax is used for writing compound statements, which can appear within stored programs (stored procedures and functions, triggers, and events). A compound statement can contain multiple statements, enclosed by the `BEGIN` and `END` keywords. *`statement_list`* represents a list of one or more statements, each terminated by a semicolon (`;`) statement delimiter. The *`statement_list`* itself is optional, so the empty compound statement (`BEGIN END`) is legal.

`BEGIN ... END` blocks can be nested.

Use of multiple statements requires that a client is able to send statement strings containing the `;` statement delimiter. In the `mysql` command-line client, this is handled with the `delimiter` command. Changing the `;` end-of-statement delimiter (for example, to `//`) permit `;` to be used in a program body. For an example, see Section 24.1, “Defining Stored Programs”.

A `BEGIN ... END` block can be labeled. See Section 13.6.2, “Statement Label Syntax”.

The optional `[NOT] ATOMIC` clause is not supported. This means that no transactional savepoint is set at the start of the instruction block and the `BEGIN` clause used in this context has no effect on the current transaction.

Note

Within all stored programs, the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. To begin a transaction in this context, use `START TRANSACTION` instead.

13.6.2 Statement Label Syntax

```
[begin_label:] BEGIN
```

```
    [statement_list]
END [end_label]

[begin_label:] LOOP
    statement_list
END LOOP [end_label]

[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]

[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

Labels are permitted for `BEGIN ... END` blocks and for the `LOOP`, `REPEAT`, and `WHILE` statements. Label use for those statements follows these rules:

- *`begin_label`* must be followed by a colon.
- *`begin_label`* can be given without *`end_label`*. If *`end_label`* is present, it must be the same as *`begin_label`*.
- *`end_label`* cannot be given without *`begin_label`*.
- Labels at the same nesting level must be distinct.
- Labels can be up to 16 characters long.

To refer to a label within the labeled construct, use an `ITERATE` or `LEAVE` statement. The following example uses those statements to continue iterating or terminate the loop:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
END;
```

The scope of a block label does not include the code for handlers declared within the block. For details, see Section 13.6.7.2, “DECLARE ... HANDLER Syntax”.

13.6.3 DECLARE Syntax

The `DECLARE` statement is used to define various items local to a program:

- Local variables. See Section 13.6.4, “Variables in Stored Programs”.
- Conditions and handlers. See Section 13.6.7, “Condition Handling”.
- Cursors. See Section 13.6.6, “Cursors”.

`DECLARE` is permitted only inside a `BEGIN ... END` compound statement and must be at its start, before any other statements.

Declarations must follow a certain order. Cursor declarations must appear before handler declarations. Variable and condition declarations must appear before cursor or handler declarations.

13.6.4 Variables in Stored Programs

System variables and user-defined variables can be used in stored programs, just as they can be used outside stored-program context. In addition, stored programs can use `DECLARE` to define local variables, and stored routines (procedures and functions) can be declared to take parameters that communicate values between the routine and its caller.

- To declare local variables, use the `DECLARE` statement, as described in Section 13.6.4.1, “Local Variable DECLARE Syntax”.
- Variables can be set directly with the `SET` statement. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.
- Results from queries can be retrieved into local variables using `SELECT ... INTO var_list` or by opening a cursor and using `FETCH ... INTO var_list`. See Section 13.2.10.1, “SELECT ... INTO Syntax”, and Section 13.6.6, “Cursors”.

For information about the scope of local variables and how MySQL resolves ambiguous names, see Section 13.6.4.2, “Local Variable Scope and Resolution”.

It is not permitted to assign the value `DEFAULT` to stored procedure or function parameters or stored program local variables (for example with a `SET var_name = DEFAULT` statement). In MySQL 8.0, this results in a syntax error.

13.6.4.1 Local Variable DECLARE Syntax

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

This statement declares local variables within stored programs. To provide a default value for a variable, include a `DEFAULT` clause. The value can be specified as an expression; it need not be a constant. If the `DEFAULT` clause is missing, the initial value is `NULL`.

Local variables are treated like stored routine parameters with respect to data type and overflow checking. See Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.

Variable declarations must appear before cursor or handler declarations.

Local variable names are not case-sensitive. Permissible characters and quoting rules are the same as for other identifiers, as described in Section 9.2, “Schema Object Names”.

The scope of a local variable is the `BEGIN ... END` block within which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

For examples of variable declarations, see Section 13.6.4.2, “Local Variable Scope and Resolution”.

13.6.4.2 Local Variable Scope and Resolution

The scope of a local variable is the `BEGIN ... END` block within which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

Because local variables are in scope only during stored program execution, references to them are not permitted in prepared statements created within a stored program. Prepared statement scope is the current session, not the stored program, so the statement could be executed after the program ends, at which

point the variables would no longer be in scope. For example, `SELECT ... INTO local_var` cannot be used as a prepared statement. This restriction also applies to stored procedure and function parameters. See Section 13.5.1, “PREPARE Syntax”.

A local variable should not have the same name as a table column. If an SQL statement, such as a `SELECT ... INTO` statement, contains a reference to a column and a declared local variable with the same name, MySQL currently interprets the reference as the name of a variable. Consider the following procedure definition:

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;

  SELECT xname, id INTO newname, xid
    FROM table1 WHERE xname = xname;
  SELECT newname;
END;
```

MySQL interprets `xname` in the `SELECT` statement as a reference to the `xname` *variable* rather than the `xname` *column*. Consequently, when the procedure `sp1()` is called, the `newname` variable returns the value `'bob'` regardless of the value of the `table1.xname` column.

Similarly, the cursor definition in the following procedure contains a `SELECT` statement that refers to `xname`. MySQL interprets this as a reference to the variable of that name rather than a column reference.

```
CREATE PROCEDURE sp2 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;
  DECLARE done TINYINT DEFAULT 0;
  DECLARE cur1 CURSOR FOR SELECT xname, id FROM table1;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur1;
  read_loop: LOOP
    FETCH FROM cur1 INTO newname, xid;
    IF done THEN LEAVE read_loop; END IF;
    SELECT newname;
  END LOOP;
  CLOSE cur1;
END;
```

See also Section 24.8, “Restrictions on Stored Programs”.

13.6.5 Flow Control Statements

MySQL supports the `IF`, `CASE`, `ITERATE`, `LEAVE LOOP`, `WHILE`, and `REPEAT` constructs for flow control within stored programs. It also supports `RETURN` within stored functions.

Many of these constructs contain other statements, as indicated by the grammar specifications in the following sections. Such constructs may be nested. For example, an `IF` statement might contain a `WHILE` loop, which itself contains a `CASE` statement.

MySQL does not support `FOR` loops.

13.6.5.1 CASE Syntax

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Or:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

The `CASE` statement for stored programs implements a complex conditional construct.

Note

There is also a `CASE` *expr*, which differs from the `CASE` *statement* described here. See Section 12.4, “Control Flow Functions”. The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

For the first syntax, *`case_value`* is an expression. This value is compared to the *`when_value`* expression in each `WHEN` clause until one of them is equal. When an equal *`when_value`* is found, the corresponding `THEN` clause *`statement_list`* executes. If no *`when_value`* is equal, the `ELSE` clause *`statement_list`* executes, if there is one.

This syntax cannot be used to test for equality with `NULL` because `NULL = NULL` is false. See Section 3.3.4.6, “Working with NULL Values”.

For the second syntax, each `WHEN` clause *`search_condition`* expression is evaluated until one is true, at which point its corresponding `THEN` clause *`statement_list`* executes. If no *`search_condition`* is equal, the `ELSE` clause *`statement_list`* executes, if there is one.

If no *`when_value`* or *`search_condition`* matches the value tested and the `CASE` statement contains no `ELSE` clause, a `Case not found for CASE statement` error results.

Each *`statement_list`* consists of one or more SQL statements; an empty *`statement_list`* is not permitted.

To handle situations where no value is matched by any `WHEN` clause, use an `ELSE` containing an empty `BEGIN ... END` block, as shown in this example. (The indentation used here in the `ELSE` clause is for purposes of clarity only, and is not otherwise significant.)

```
DELIMITER |

CREATE PROCEDURE p()
  BEGIN
    DECLARE v INT DEFAULT 1;

    CASE v
      WHEN 2 THEN SELECT v;
      WHEN 3 THEN SELECT 0;
      ELSE
        BEGIN
        END;
    END CASE;
  END;
```

```
  |
```

13.6.5.2 IF Syntax

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

The `IF` statement for stored programs implements a basic conditional construct.

Note

There is also an `IF()` *function*, which differs from the `IF` *statement* described here. See Section 12.4, “Control Flow Functions”. The `IF` statement can have `THEN`, `ELSE`, and `ELSEIF` clauses, and it is terminated with `END IF`.

If a given *`search_condition`* evaluates to true, the corresponding `THEN` or `ELSEIF` clause *`statement_list`* executes. If no *`search_condition`* matches, the `ELSE` clause *`statement_list`* executes.

Each *`statement_list`* consists of one or more SQL statements; an empty *`statement_list`* is not permitted.

An `IF ... END IF` block, like all other flow-control blocks used within stored programs, must be terminated with a semicolon, as shown in this example:

```
DELIMITER //

CREATE FUNCTION SimpleCompare(n INT, m INT)
  RETURNS VARCHAR(20)

  BEGIN
    DECLARE s VARCHAR(20);

    IF n > m THEN SET s = '>';
    ELSEIF n = m THEN SET s = '=';
    ELSE SET s = '<';
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m);

    RETURN s;
  END //

DELIMITER ;
```

As with other flow-control constructs, `IF ... END IF` blocks may be nested within other flow-control constructs, including other `IF` statements. Each `IF` must be terminated by its own `END IF` followed by a semicolon. You can use indentation to make nested flow-control blocks more easily readable by humans (although this is not required by MySQL), as shown here:

```
DELIMITER //

CREATE FUNCTION VerboseCompare (n INT, m INT)
  RETURNS VARCHAR(50)

  BEGIN
    DECLARE s VARCHAR(50);
```

```
    IF n = m THEN SET s = 'equals';
    ELSE
      IF n > m THEN SET s = 'greater';
      ELSE SET s = 'less';
      END IF;

      SET s = CONCAT('is ', s, ' than');
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m, '.');

    RETURN s;
  END //

DELIMITER ;
```

In this example, the inner `IF` is evaluated only if `n` is not equal to `m`.

13.6.5.3 ITERATE Syntax

```
ITERATE label
```

`ITERATE` can appear only within `LOOP`, `REPEAT`, and `WHILE` statements. `ITERATE` means “start the loop again.”

For an example, see Section 13.6.5.5, “LOOP Syntax”.

13.6.5.4 LEAVE Syntax

```
LEAVE label
```

This statement is used to exit the flow control construct that has the given label. If the label is for the outermost stored program block, `LEAVE` exits the program.

`LEAVE` can be used within `BEGIN ... END` or loop constructs (`LOOP`, `REPEAT`, `WHILE`).

For an example, see Section 13.6.5.5, “LOOP Syntax”.

13.6.5.5 LOOP Syntax

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

`LOOP` implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements, each terminated by a semicolon (`;`) statement delimiter. The statements within the loop are repeated until the loop is terminated. Usually, this is accomplished with a `LEAVE` statement. Within a stored function, `RETURN` can also be used, which exits the function entirely.

Neglecting to include a loop-termination statement results in an infinite loop.

A `LOOP` statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Label Syntax”.

Example:

```
CREATE PROCEDURE doiterate(p1 INT)
```

```
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN
      ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END;
```

13.6.5.6 REPEAT Syntax

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

The statement list within a REPEAT statement is repeated until the *search_condition* expression is true. Thus, a REPEAT always enters the loop at least once. *statement_list* consists of one or more statements, each terminated by a semicolon (;) statement delimiter.

A REPEAT statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Label Syntax”.

Example:

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
       BEGIN
         SET @x = 0;
         REPEAT
           SET @x = @x + 1;
         UNTIL @x > p1 END REPEAT;
       END
       //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

13.6.5.7 RETURN Syntax

```
RETURN expr
```

The RETURN statement terminates execution of a stored function and returns the value *expr* to the function caller. There must be at least one RETURN statement in a stored function. There may be more than one if the function has multiple exit points.

This statement is not used in stored procedures, triggers, or events. The LEAVE statement can be used to exit a stored program of those types.

13.6.5.8 WHILE Syntax

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

The statement list within a `WHILE` statement is repeated as long as the *`search_condition`* expression is true. *`statement_list`* consists of one or more SQL statements, each terminated by a semicolon (`;`) statement delimiter.

A `WHILE` statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Label Syntax”.

Example:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END;
```

13.6.6 Cursors

MySQL supports cursors inside stored programs. The syntax is as in embedded SQL. Cursors have these properties:

- Asensitive: The server may or may not make a copy of its result table
- Read only: Not updatable
- Nonscrollable: Can be traversed only in one direction and cannot skip rows

Cursor declarations must appear before handler declarations and after variable and condition declarations.

Example:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE a CHAR(16);
  DECLARE b, c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur1;
  OPEN cur2;

  read_loop: LOOP
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF done THEN
      LEAVE read_loop;
    END IF;
    IF b < c THEN
```

```
      INSERT INTO test.t3 VALUES (a,b);
    ELSE
      INSERT INTO test.t3 VALUES (a,c);
    END IF;
  END LOOP;

  CLOSE cur1;
  CLOSE cur2;
END;
```

13.6.6.1 Cursor CLOSE Syntax

```
CLOSE cursor_name
```

This statement closes a previously opened cursor. For an example, see Section 13.6.6, “Cursors”.

An error occurs if the cursor is not open.

If not closed explicitly, a cursor is closed at the end of the `BEGIN ... END` block in which it was declared.

13.6.6.2 Cursor DECLARE Syntax

```
DECLARE cursor_name CURSOR FOR select_statement
```

This statement declares a cursor and associates it with a `SELECT` statement that retrieves the rows to be traversed by the cursor. To fetch the rows later, use a `FETCH` statement. The number of columns retrieved by the `SELECT` statement must match the number of output variables specified in the `FETCH` statement.

The `SELECT` statement cannot have an `INTO` clause.

Cursor declarations must appear before handler declarations and after variable and condition declarations.

A stored program may contain multiple cursor declarations, but each cursor declared in a given block must have a unique name. For an example, see Section 13.6.6, “Cursors”.

For information available through `SHOW` statements, it is possible in many cases to obtain equivalent information by using a cursor with an `INFORMATION_SCHEMA` table.

13.6.6.3 Cursor FETCH Syntax

```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
```

This statement fetches the next row for the `SELECT` statement associated with the specified cursor (which must be open), and advances the cursor pointer. If a row exists, the fetched columns are stored in the named variables. The number of columns retrieved by the `SELECT` statement must match the number of output variables specified in the `FETCH` statement.

If no more rows are available, a No Data condition occurs with SQLSTATE value `'02000'`. To detect this condition, you can set up a handler for it (or for a `NOT FOUND` condition). For an example, see Section 13.6.6, “Cursors”.

Be aware that another operation, such as a `SELECT` or another `FETCH`, may also cause the handler to execute by raising the same condition. If it is necessary to distinguish which operation raised the condition, place the operation within its own `BEGIN ... END` block so that it can be associated with its own handler.

13.6.6.4 Cursor OPEN Syntax

```
OPEN cursor_name
```

This statement opens a previously declared cursor. For an example, see Section 13.6.6, “Cursors”.

13.6.6.5 Restrictions on Server-Side Cursors

Server-side cursors are implemented in the C API using the `mysql_stmt_attr_set()` function. The same implementation is used for cursors in stored routines. A server-side cursor enables a result set to be generated on the server side, but not transferred to the client except for those rows that the client requests. For example, if a client executes a query but is only interested in the first row, the remaining rows are not transferred.

In MySQL, a server-side cursor is materialized into an internal temporary table. Initially, this is a `MEMORY` table, but is converted to a `MyISAM` table when its size exceeds the minimum value of the `max_heap_table_size` and `tmp_table_size` system variables. The same restrictions apply to internal temporary tables created to hold the result set for a cursor as for other uses of internal temporary tables. See Section 8.4.4, “Internal Temporary Table Use in MySQL”. One limitation of the implementation is that for a large result set, retrieving its rows through a cursor might be slow.

Cursors are read only; you cannot use a cursor to update rows.

`UPDATE WHERE CURRENT OF` and `DELETE WHERE CURRENT OF` are not implemented, because updatable cursors are not supported.

Cursors are nonholdable (not held open after a commit).

Cursors are asensitive.

Cursors are nonscrollable.

Cursors are not named. The statement handler acts as the cursor ID.

You can have open only a single cursor per prepared statement. If you need several cursors, you must prepare several statements.

You cannot use a cursor for a statement that generates a result set if the statement is not supported in prepared mode. This includes statements such as `CHECK TABLE`, `HANDLER READ`, and `SHOW BINLOG EVENTS`.

13.6.7 Condition Handling

Conditions may arise during stored program execution that require special handling, such as exiting the current program block or continuing execution. Handlers can be defined for general conditions such as warnings or exceptions, or for specific conditions such as a particular error code. Specific conditions can be assigned names and referred to that way in handlers.

To name a condition, use the `DECLARE ... CONDITION` statement. To declare a handler, use the `DECLARE ... HANDLER` statement. See Section 13.6.7.1, “DECLARE ... CONDITION Syntax”, and Section 13.6.7.2, “DECLARE ... HANDLER Syntax”. For information about how the server chooses handlers when a condition occurs, see Section 13.6.7.6, “Scope Rules for Handlers”.

To raise a condition, use the `SIGNAL` statement. To modify condition information within a condition handler, use `RESIGNAL`. See Section 13.6.7.1, “DECLARE ... CONDITION Syntax”, and Section 13.6.7.2, “DECLARE ... HANDLER Syntax”.

To retrieve information from the diagnostics area, use the `GET DIAGNOSTICS` statement (see Section 13.6.7.3, “GET DIAGNOSTICS Syntax”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

13.6.7.1 DECLARE ... CONDITION Syntax

```
DECLARE condition_name CONDITION FOR condition_value

condition_value: {
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
}
```

The `DECLARE ... CONDITION` statement declares a named error condition, associating a name with a condition that needs specific handling. The name can be referred to in a subsequent `DECLARE ... HANDLER` statement (see Section 13.6.7.2, “DECLARE ... HANDLER Syntax”).

Condition declarations must appear before cursor or handler declarations.

The *`condition_value`* for `DECLARE ... CONDITION` indicates the specific condition or class of conditions to associate with the condition name. It can take the following forms:

- *`mysql_error_code`*: An integer literal indicating a MySQL error code.

  Do not use MySQL error code 0 because that indicates success rather than an error condition. For a list of MySQL error codes, see Section B.3.1, “Server Error Message Reference”.

- SQLSTATE [VALUE] *`sqlstate_value`*: A 5-character string literal indicating an SQLSTATE value.

  Do not use SQLSTATE values that begin with `'00'` because those indicate success rather than an error condition. For a list of SQLSTATE values, see Section B.3.1, “Server Error Message Reference”.

Condition names referred to in `SIGNAL` or use `RESIGNAL` statements must be associated with SQLSTATE values, not MySQL error codes.

Using names for conditions can help make stored program code clearer. For example, this handler applies to attempts to drop a nonexistent table, but that is apparent only if you know that 1051 is the MySQL error code for “unknown table”:

```
DECLARE CONTINUE HANDLER FOR 1051
  BEGIN
    -- body of handler
  END;
```

By declaring a name for the condition, the purpose of the handler is more readily seen:

```
DECLARE no_such_table CONDITION FOR 1051;
DECLARE CONTINUE HANDLER FOR no_such_table
  BEGIN
    -- body of handler
  END;
```

Here is a named condition for the same condition, but based on the corresponding SQLSTATE value rather than the MySQL error code:

```
DECLARE no_such_table CONDITION FOR SQLSTATE '42S02';
DECLARE CONTINUE HANDLER FOR no_such_table
  BEGIN
```

```
    -- body of handler
  END;
```

13.6.7.2 DECLARE ... HANDLER Syntax

```
DECLARE handler_action HANDLER
    FOR condition_value [, condition_value] ...
    statement

handler_action: {
    CONTINUE
  | EXIT
  | UNDO
}

condition_value: {
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
}
```

The `DECLARE ... HANDLER` statement specifies a handler that deals with one or more conditions. If one of these conditions occurs, the specified *`statement`* executes. *`statement`* can be a simple statement such as `SET var_name = value`, or a compound statement written using `BEGIN` and `END` (see Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”).

Handler declarations must appear after variable or condition declarations.

The *`handler_action`* value indicates what action the handler takes after execution of the handler statement:

- `CONTINUE`: Execution of the current program continues.
- `EXIT`: Execution terminates for the `BEGIN ... END` compound statement in which the handler is declared. This is true even if the condition occurs in an inner block.
- `UNDO`: Not supported.

The *`condition_value`* for `DECLARE ... HANDLER` indicates the specific condition or class of conditions that activates the handler. It can take the following forms:

- *`mysql_error_code`*: An integer literal indicating a MySQL error code, such as 1051 to specify “unknown table”:

  ```
  DECLARE CONTINUE HANDLER FOR 1051
    BEGIN
      -- body of handler
    END;
  ```

  Do not use MySQL error code 0 because that indicates success rather than an error condition. For a list of MySQL error codes, see Section B.3.1, “Server Error Message Reference”.

- SQLSTATE [VALUE] *`sqlstate_value`*: A 5-character string literal indicating an SQLSTATE value, such as `'42S01'` to specify “unknown table”:

  ```
  DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
  ```

```
  BEGIN
    -- body of handler
  END;
```

Do not use SQLSTATE values that begin with `'00'` because those indicate success rather than an error condition. For a list of SQLSTATE values, see Section B.3.1, “Server Error Message Reference”.

- *`condition_name`*: A condition name previously specified with `DECLARE ... CONDITION`. A condition name can be associated with a MySQL error code or SQLSTATE value. See Section 13.6.7.1, “DECLARE ... CONDITION Syntax”.
- `SQLWARNING`: Shorthand for the class of SQLSTATE values that begin with `'01'`.

```
DECLARE CONTINUE HANDLER FOR SQLWARNING
  BEGIN
    -- body of handler
  END;
```

- `NOT FOUND`: Shorthand for the class of SQLSTATE values that begin with `'02'`. This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. If no more rows are available, a No Data condition occurs with SQLSTATE value `'02000'`. To detect this condition, you can set up a handler for it or for a `NOT FOUND` condition.

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
  BEGIN
    -- body of handler
  END;
```

For another example, see Section 13.6.6, “Cursors”. The `NOT FOUND` condition also occurs for `SELECT ... INTO` *`var_list`* statements that retrieve no rows.

- `SQLEXCEPTION`: Shorthand for the class of SQLSTATE values that do not begin with `'00'`, `'01'`, or `'02'`.

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
  BEGIN
    -- body of handler
  END;
```

For information about how the server chooses handlers when a condition occurs, see Section 13.6.7.6, “Scope Rules for Handlers”.

If a condition occurs for which no handler has been declared, the action taken depends on the condition class:

- For `SQLEXCEPTION` conditions, the stored program terminates at the statement that raised the condition, as if there were an `EXIT` handler. If the program was called by another stored program, the calling program handles the condition using the handler selection rules applied to its own handlers.
- For `SQLWARNING` conditions, the program continues executing, as if there were a `CONTINUE` handler.
- For `NOT FOUND` conditions, if the condition was raised normally, the action is `CONTINUE`. If it was raised by `SIGNAL` or `RESIGNAL`, the action is `EXIT`.

The following example uses a handler for `SQLSTATE '23000'`, which occurs for a duplicate-key error:

```
mysql> CREATE TABLE test.t (s1 INT, PRIMARY KEY (s1));
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
       BEGIN
         DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
         SET @x = 1;
         INSERT INTO test.t VALUES (1);
         SET @x = 2;
         INSERT INTO test.t VALUES (1);
         SET @x = 3;
       END;
       //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo()//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
    +------+
    | @x   |
    +------+
    | 3    |
    +------+
    1 row in set (0.00 sec)
```

Notice that `@x` is `3` after the procedure executes, which shows that execution continued to the end of the procedure after the error occurred. If the `DECLARE ... HANDLER` statement had not been present, MySQL would have taken the default action (`EXIT`) after the second `INSERT` failed due to the `PRIMARY KEY` constraint, and `SELECT @x` would have returned `2`.

To ignore a condition, declare a `CONTINUE` handler for it and associate it with an empty block. For example:

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

The scope of a block label does not include the code for handlers declared within the block. Therefore, the statement associated with a handler cannot use `ITERATE` or `LEAVE` to refer to labels for blocks that enclose the handler declaration. Consider the following example, where the `REPEAT` block has a label of `retry`:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  retry:
    REPEAT
      BEGIN
        DECLARE CONTINUE HANDLER FOR SQLWARNING
          BEGIN
            ITERATE retry;    # illegal
          END;
        IF i < 0 THEN
          LEAVE retry;        # legal
        END IF;
        SET i = i - 1;
      END;
    UNTIL FALSE END REPEAT;
END;
```

The `retry` label is in scope for the `IF` statement within the block. It is not in scope for the `CONTINUE` handler, so the reference there is invalid and results in an error:

```
ERROR 1308 (42000): LEAVE with no matching label: retry
```

To avoid references to outer labels in handlers, use one of these strategies:

- To leave the block, use an `EXIT` handler. If no block cleanup is required, the `BEGIN ... END` handler body can be empty:

  ```
  DECLARE EXIT HANDLER FOR SQLWARNING BEGIN END;
  ```

  Otherwise, put the cleanup statements in the handler body:

  ```
  DECLARE EXIT HANDLER FOR SQLWARNING
    BEGIN
      block cleanup statements
    END;
  ```

- To continue execution, set a status variable in a `CONTINUE` handler that can be checked in the enclosing block to determine whether the handler was invoked. The following example uses the variable `done` for this purpose:

  ```
  CREATE PROCEDURE p ()
  BEGIN
    DECLARE i INT DEFAULT 3;
    DECLARE done INT DEFAULT FALSE;
    retry:
      REPEAT
        BEGIN
          DECLARE CONTINUE HANDLER FOR SQLWARNING
            BEGIN
              SET done = TRUE;
            END;
          IF done OR i < 0 THEN
            LEAVE retry;
          END IF;
          SET i = i - 1;
        END;
      UNTIL FALSE END REPEAT;
  END;
  ```

13.6.7.3 GET DIAGNOSTICS Syntax

```
GET [CURRENT | STACKED] DIAGNOSTICS
{
    statement_information_item
    [, statement_information_item] ...
  | CONDITION condition_number
    condition_information_item
    [, condition_information_item] ...
}

statement_information_item:
    target = statement_information_item_name

condition_information_item:
    target = condition_information_item_name

statement_information_item_name:
    NUMBER
  | ROW_COUNT
```

```
condition_information_item_name: {
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | RETURNED_SQLSTATE
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
}

condition_number, target:
    (see following discussion)
```

SQL statements produce diagnostic information that populates the diagnostics area. The `GET DIAGNOSTICS` statement enables applications to inspect this information. (You can also use `SHOW WARNINGS` or `SHOW ERRORS` to see conditions or errors.)

No special privileges are required to execute `GET DIAGNOSTICS`.

The keyword `CURRENT` means to retrieve information from the current diagnostics area. The keyword `STACKED` means to retrieve information from the second diagnostics area, which is available only if the current context is a condition handler. If neither keyword is given, the default is to use the current diagnostics area.

The `GET DIAGNOSTICS` statement is typically used in a handler within a stored program. It is a MySQL extension that `GET [CURRENT] DIAGNOSTICS` is permitted outside handler context to check the execution of any SQL statement. For example, if you invoke the `mysql` client program, you can enter these statements at the prompt:

```
mysql> DROP TABLE test.no_such_table;
ERROR 1051 (42S02): Unknown table 'test.no_such_table'
mysql> GET DIAGNOSTICS CONDITION 1
        @p1 = RETURNED_SQLSTATE, @p2 = MESSAGE_TEXT;
mysql> SELECT @p1, @p2;
+-------+-------------------------------------+
| @p1   | @p2                                 |
+-------+-------------------------------------+
| 42S02 | Unknown table 'test.no_such_table' |
+-------+-------------------------------------+
```

This extension applies only to the current diagnostics area. It does not apply to the second diagnostics area because `GET STACKED DIAGNOSTICS` is permitted only if the current context is a condition handler. If that is not the case, a `GET STACKED DIAGNOSTICS when handler not active` error occurs.

For a description of the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”. Briefly, it contains two kinds of information:

- Statement information, such as the number of conditions that occurred or the affected-rows count.
- Condition information, such as the error code and message. If a statement raises multiple conditions, this part of the diagnostics area has a condition area for each one. If a statement raises no conditions, this part of the diagnostics area is empty.

For a statement that produces three conditions, the diagnostics area contains statement and condition information like this:

```
Statement information:
  row count
  ... other statement information items ...
Condition area list:
  Condition area 1:
    error code for condition 1
    error message for condition 1
    ... other condition information items ...
  Condition area 2:
    error code for condition 2:
    error message for condition 2
    ... other condition information items ...
  Condition area 3:
    error code for condition 3
    error message for condition 3
    ... other condition information items ...
```

`GET DIAGNOSTICS` can obtain either statement or condition information, but not both in the same statement:

- To obtain statement information, retrieve the desired statement items into target variables. This instance of `GET DIAGNOSTICS` assigns the number of available conditions and the rows-affected count to the user variables `@p1` and `@p2`:

  ```
  GET DIAGNOSTICS @p1 = NUMBER, @p2 = ROW_COUNT;
  ```

- To obtain condition information, specify the condition number and retrieve the desired condition items into target variables. This instance of `GET DIAGNOSTICS` assigns the SQLSTATE value and error message to the user variables `@p3` and `@p4`:

  ```
  GET DIAGNOSTICS CONDITION 1
    @p3 = RETURNED_SQLSTATE, @p4 = MESSAGE_TEXT;
  ```

The retrieval list specifies one or more *`target = item_name`* assignments, separated by commas. Each assignment names a target variable and either a *`statement_information_item_name`* or *`condition_information_item_name`* designator, depending on whether the statement retrieves statement or condition information.

Valid *`target`* designators for storing item information can be stored procedure or function parameters, stored program local variables declared with `DECLARE`, or user-defined variables.

Valid *`condition_number`* designators can be stored procedure or function parameters, stored program local variables declared with `DECLARE`, user-defined variables, system variables, or literals. A character literal may include a *`_charset`* introducer. A warning occurs if the condition number is not in the range from 1 to the number of condition areas that have information. In this case, the warning is added to the diagnostics area without clearing it.

When a condition occurs, MySQL does not populate all condition items recognized by `GET DIAGNOSTICS`. For example:

```
mysql> GET DIAGNOSTICS CONDITION 1
         @p5 = SCHEMA_NAME, @p6 = TABLE_NAME;
mysql> SELECT @p5, @p6;
+------+------+
| @p5  | @p6  |
+------+------+
|      |      |
+------+------+
```

In standard SQL, if there are multiple conditions, the first condition relates to the `SQLSTATE` value returned for the previous SQL statement. In MySQL, this is not guaranteed. To get the main error, you cannot do this:

```
GET DIAGNOSTICS CONDITION 1 @errno = MYSQL_ERRNO;
```

Instead, retrieve the condition count first, then use it to specify which condition number to inspect:

```
GET DIAGNOSTICS @cno = NUMBER;
GET DIAGNOSTICS CONDITION @cno @errno = MYSQL_ERRNO;
```

For information about permissible statement and condition information items, and which ones are populated when a condition occurs, see Diagnostics Area Information Items.

Here is an example that uses `GET DIAGNOSTICS` and an exception handler in stored procedure context to assess the outcome of an insert operation. If the insert was successful, the procedure uses `GET DIAGNOSTICS` to get the rows-affected count. This shows that you can use `GET DIAGNOSTICS` multiple times to retrieve information about a statement as long as the current diagnostics area has not been cleared.

```
CREATE PROCEDURE do_insert(value INT)
BEGIN
  -- Declare variables to hold diagnostics area information
  DECLARE code CHAR(5) DEFAULT '00000';
  DECLARE msg TEXT;
  DECLARE rows INT;
  DECLARE result TEXT;
  -- Declare exception handler for failed insert
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
      GET DIAGNOSTICS CONDITION 1
        code = RETURNED_SQLSTATE, msg = MESSAGE_TEXT;
    END;

  -- Perform the insert
  INSERT INTO t1 (int_col) VALUES(value);
  -- Check whether the insert was successful
  IF code = '00000' THEN
    GET DIAGNOSTICS rows = ROW_COUNT;
    SET result = CONCAT('insert succeeded, row count = ',rows);
  ELSE
    SET result = CONCAT('insert failed, error = ',code,', message = ',msg);
  END IF;
  -- Say what happened
  SELECT result;
END;
```

Suppose that `t1.int_col` is an integer column that is declared as `NOT NULL`. The procedure produces these results when invoked to insert non-`NULL` and `NULL` values, respectively:

```
mysql> CALL do_insert(1);
+---------------------------------+
| result                          |
+---------------------------------+
| insert succeeded, row count = 1 |
+---------------------------------+

mysql> CALL do_insert(NULL);
+-------------------------------------------------------------------------+
| result                                                                  |
```

```
+-------------------------------------------------------------------------+
| insert failed, error = 23000, message = Column 'int_col' cannot be null |
+-------------------------------------------------------------------------+
```

When a condition handler activates, a push to the diagnostics area stack occurs:

- The first (current) diagnostics area becomes the second (stacked) diagnostics area and a new current diagnostics area is created as a copy of it.
- `GET [CURRENT] DIAGNOSTICS` and `GET STACKED DIAGNOSTICS` can be used within the handler to access the contents of the current and stacked diagnostics areas.
- Initially, both diagnostics areas return the same result, so it is possible to get information from the current diagnostics area about the condition that activated the handler, *as long as* you execute no statements within the handler that change its current diagnostics area.
- However, statements executing within the handler can modify the current diagnostics area, clearing and setting its contents according to the normal rules (see How the Diagnostics Area is Cleared and Populated).

  A more reliable way to obtain information about the handler-activating condition is to use the stacked diagnostics area, which cannot be modified by statements executing within the handler except `RESIGNAL`. For information about when the current diagnostics area is set and cleared, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

The next example shows how `GET STACKED DIAGNOSTICS` can be used within a handler to obtain information about the handled exception, even after the current diagnostics area has been modified by handler statements.

Within a stored procedure `p()`, we attempt to insert two values into a table that contains a `TEXT NOT NULL` column. The first value is a non-`NULL` string and the second is `NULL`. The column prohibits `NULL` values, so the first insert succeeds but the second causes an exception. The procedure includes an exception handler that maps attempts to insert `NULL` into inserts of the empty string:

```
DROP TABLE IF EXISTS t1;
CREATE TABLE t1 (c1 TEXT NOT NULL);
DROP PROCEDURE IF EXISTS p;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  -- Declare variables to hold diagnostics area information
  DECLARE errcount INT;
  DECLARE errno INT;
  DECLARE msg TEXT;
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    -- Here the current DA is nonempty because no prior statements
    -- executing within the handler have cleared it
    GET CURRENT DIAGNOSTICS CONDITION 1
      errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
    SELECT 'current DA before mapped insert' AS op, errno, msg;
    GET STACKED DIAGNOSTICS CONDITION 1
      errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
    SELECT 'stacked DA before mapped insert' AS op, errno, msg;

    -- Map attempted NULL insert to empty string insert
    INSERT INTO t1 (c1) VALUES('');

    -- Here the current DA should be empty (if the INSERT succeeded),
    -- so check whether there are conditions before attempting to
    -- obtain condition information
```

```
    GET CURRENT DIAGNOSTICS errcount = NUMBER;
    IF errcount = 0
    THEN
      SELECT 'mapped insert succeeded, current DA is empty' AS op;
    ELSE
      GET CURRENT DIAGNOSTICS CONDITION 1
        errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
      SELECT 'current DA after mapped insert' AS op, errno, msg;
    END IF ;
    GET STACKED DIAGNOSTICS CONDITION 1
      errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
    SELECT 'stacked DA after mapped insert' AS op, errno, msg;
  END;
  INSERT INTO t1 (c1) VALUES('string 1');
  INSERT INTO t1 (c1) VALUES(NULL);
END;
//
delimiter ;
CALL p();
SELECT * FROM t1;
```

When the handler activates, a copy of the current diagnostics area is pushed to the diagnostics area stack. The handler first displays the contents of the current and stacked diagnostics areas, which are both the same initially:

```
+---------------------------------+-------+----------------------------+
| op                              | errno | msg                        |
+---------------------------------+-------+----------------------------+
| current DA before mapped insert |  1048 | Column 'c1' cannot be null |
+---------------------------------+-------+----------------------------+

+---------------------------------+-------+----------------------------+
| op                              | errno | msg                        |
+---------------------------------+-------+----------------------------+
| stacked DA before mapped insert |  1048 | Column 'c1' cannot be null |
+---------------------------------+-------+----------------------------+
```

Statements executing after the `GET DIAGNOSTICS` statements may reset the current diagnostics area. statements may reset the current diagnostics area. For example, the handler maps the `NULL` insert to an empty-string insert and displays the result. The new insert succeeds and clears the current diagnostics area, but the stacked diagnostics area remains unchanged and still contains information about the condition that activated the handler:

```
+-----------------------------------------------+
| op                                            |
+-----------------------------------------------+
| mapped insert succeeded, current DA is empty |
+-----------------------------------------------+

+---------------------------------+-------+----------------------------+
| op                              | errno | msg                        |
+---------------------------------+-------+----------------------------+
| stacked DA after mapped insert |  1048 | Column 'c1' cannot be null |
+---------------------------------+-------+----------------------------+
```

When the condition handler ends, its current diagnostics area is popped from the stack and the stacked diagnostics area becomes the current diagnostics area in the stored procedure.

After the procedure returns, the table contains two rows. The empty row results from the attempt to insert `NULL` that was mapped to an empty-string insert:

```
+----------+
| c1       |
+----------+
| string 1 |
|          |
+----------+
```

In the preceding example, the first two `GET DIAGNOSTICS` statements within the condition handler that retrieve information from the current and stacked diagnostics areas return the same values. This will not be the case if statements that reset the current diagnostics area execute earlier within the handler. Suppose that `p()` is rewritten to place the `DECLARE` statements within the handler definition rather than preceding it:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    -- Declare variables to hold diagnostics area information
    DECLARE errcount INT;
    DECLARE errno INT;
    DECLARE msg TEXT;
    GET CURRENT DIAGNOSTICS CONDITION 1
      errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
    SELECT 'current DA before mapped insert' AS op, errno, msg;
    GET STACKED DIAGNOSTICS CONDITION 1
      errno = MYSQL_ERRNO, msg = MESSAGE_TEXT;
    SELECT 'stacked DA before mapped insert' AS op, errno, msg;
...
```

In this case, the result is version dependent:

- Before MySQL 5.7.2, `DECLARE` does not change the current diagnostics area, so the first two `GET DIAGNOSTICS` statements return the same result, just as in the original version of `p()`.

  In MySQL 5.7.2, work was done to ensure that all nondiagnostic statements populate the diagnostics area, per the SQL standard. `DECLARE` is one of them, so in 5.7.2 and higher, `DECLARE` statements executing at the beginning of the handler clear the current diagnostics area and the `GET DIAGNOSTICS` statements produce different results:

  ```
  +---------------------------------+-------+------+
  | op                              | errno | msg  |
  +---------------------------------+-------+------+
  | current DA before mapped insert |  NULL | NULL |
  +---------------------------------+-------+------+

  +---------------------------------+-------+----------------------------+
  | op                              | errno | msg                        |
  +---------------------------------+-------+----------------------------+
  | stacked DA before mapped insert |  1048 | Column 'c1' cannot be null |
  +---------------------------------+-------+----------------------------+
  ```

To avoid this issue within a condition handler when seeking to obtain information about the condition that activated the handler, be sure to access the stacked diagnostics area, not the current diagnostics area.

13.6.7.4 RESIGNAL Syntax

```
RESIGNAL [condition_value]
    [SET signal_information_item
    [, signal_information_item] ...]

condition_value: {
```

```
    SQLSTATE [VALUE] sqlstate_value
  | condition_name
}

signal_information_item:
    condition_information_item_name = simple_value_specification

condition_information_item_name: {
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
}

condition_name, simple_value_specification:
    (see following discussion)
```

`RESIGNAL` passes on the error condition information that is available during execution of a condition handler within a compound statement inside a stored procedure or function, trigger, or event. `RESIGNAL` may change some or all information before passing it on. `RESIGNAL` is related to `SIGNAL`, but instead of originating a condition as `SIGNAL` does, `RESIGNAL` relays existing condition information, possibly after modifying it.

`RESIGNAL` makes it possible to both handle an error and return the error information. Otherwise, by executing an SQL statement within the handler, information that caused the handler's activation is destroyed. `RESIGNAL` also can make some procedures shorter if a given handler can handle part of a situation, then pass the condition “up the line” to another handler.

No privileges are required to execute the `RESIGNAL` statement.

All forms of `RESIGNAL` require that the current context be a condition handler. Otherwise, `RESIGNAL` is illegal and a `RESIGNAL when handler not active` error occurs.

To retrieve information from the diagnostics area, use the `GET DIAGNOSTICS` statement (see Section 13.6.7.3, “GET DIAGNOSTICS Syntax”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

- RESIGNAL Overview
- RESIGNAL Alone
- RESIGNAL with New Signal Information
- RESIGNAL with a Condition Value and Optional New Signal Information
- RESIGNAL Requires Condition Handler Context

RESIGNAL Overview

For *`condition_value`* and *`signal_information_item`*, the definitions and rules are the same for `RESIGNAL` as for `SIGNAL`. For example, the *`condition_value`* can be an `SQLSTATE` value, and the value can indicate errors, warnings, or “not found.” For additional information, see Section 13.6.7.5, “SIGNAL Syntax”.

The RESIGNAL statement takes *condition_value* and SET clauses, both of which are optional. This leads to several possible uses:

- RESIGNAL alone:

```
RESIGNAL;
```

- RESIGNAL with new signal information:

```
RESIGNAL SET signal_information_item [, signal_information_item] ...;
```

- RESIGNAL with a condition value and possibly new signal information:

```
RESIGNAL condition_value
    [SET signal_information_item [, signal_information_item] ...];
```

These use cases all cause changes to the diagnostics and condition areas:

- A diagnostics area contains one or more condition areas.
- A condition area contains condition information items, such as the SQLSTATE value, MYSQL_ERRNO, or MESSAGE_TEXT.

There is a stack of diagnostics areas. When a handler takes control, it pushes a diagnostics area to the top of the stack, so there are two diagnostics areas during handler execution:

- The first (current) diagnostics area, which starts as a copy of the last diagnostics area, but will be overwritten by the first statement in the handler that changes the current diagnostics area.
- The last (stacked) diagnostics area, which has the condition areas that were set up before the handler took control.

The maximum number of condition areas in a diagnostics area is determined by the value of the max_error_count system variable. See Diagnostics Area-Related System Variables.

RESIGNAL Alone

A simple RESIGNAL alone means “pass on the error with no change.” It restores the last diagnostics area and makes it the current diagnostics area. That is, it “pops” the diagnostics area stack.

Within a condition handler that catches a condition, one use for RESIGNAL alone is to perform some other actions, and then pass on without change the original condition information (the information that existed before entry into the handler).

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL; END IF;
  END;
  DROP TABLE xx;
```

```
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();
```

Suppose that the `DROP TABLE xx` statement fails. The diagnostics area stack looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

Then execution enters the `EXIT` handler. It starts by pushing a diagnostics area to the top of the stack, which now looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
DA 2. ERROR 1051 (42S02): Unknown table 'xx'
```

At this point, the contents of the first (current) and second (stacked) diagnostics areas are the same. The first diagnostics area may be modified by statements executing subsequently within the handler.

Usually a procedure statement clears the first diagnostics area. `BEGIN` is an exception, it does not clear, it does nothing. `SET` is not an exception, it clears, performs the operation, and produces a result of "success." The diagnostics area stack now looks like this:

```
DA 1. ERROR 0000 (00000): Successful operation
DA 2. ERROR 1051 (42S02): Unknown table 'xx'
```

At this point, if `@a = 0`, `RESIGNAL` pops the diagnostics area stack, which now looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

And that is what the caller sees.

If `@a` is not 0, the handler simply ends, which means that there is no more use for the current diagnostics area (it has been "handled"), so it can be thrown away, causing the stacked diagnostics area to become the current diagnostics area again. The diagnostics area stack looks like this:

```
DA 1. ERROR 0000 (00000): Successful operation
```

The details make it look complex, but the end result is quite useful: Handlers can execute without destroying information about the condition that caused activation of the handler.

RESIGNAL with New Signal Information

`RESIGNAL` with a `SET` clause provides new signal information, so the statement means "pass on the error with changes":

```
RESIGNAL SET signal_information_item [, signal_information_item] ...;
```

As with `RESIGNAL` alone, the idea is to pop the diagnostics area stack so that the original information will go out. Unlike `RESIGNAL` alone, anything specified in the `SET` clause changes.

Example:

```
```

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL SET MYSQL_ERRNO = 5; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();
```

Remember from the previous discussion that `RESIGNAL` alone results in a diagnostics area stack like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

The `RESIGNAL SET MYSQL_ERRNO = 5` statement results in this stack instead, which is what the caller sees:

```
DA 1. ERROR 5 (42S02): Unknown table 'xx'
```

In other words, it changes the error number, and nothing else.

The `RESIGNAL` statement can change any or all of the signal information items, making the first condition area of the diagnostics area look quite different.

RESIGNAL with a Condition Value and Optional New Signal Information

`RESIGNAL` with a condition value means “push a condition into the current diagnostics area.” If the `SET` clause is present, it also changes the error information.

```
RESIGNAL condition_value
    [SET signal_information_item [, signal_information_item] ...];
```

This form of `RESIGNAL` restores the last diagnostics area and makes it the current diagnostics area. That is, it “pops” the diagnostics area stack, which is the same as what a simple `RESIGNAL` alone would do. However, it also changes the diagnostics area depending on the condition value or signal information.

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL SQLSTATE '45000' SET MYSQL_ERRNO=5; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
```

```
SET @@max_error_count = 2;
CALL p();
SHOW ERRORS;
```

This is similar to the previous example, and the effects are the same, except that if `RESIGNAL` happens, the current condition area looks different at the end. (The reason the condition adds to rather than replaces the existing condition is the use of a condition value.)

The `RESIGNAL` statement includes a condition value (`SQLSTATE '45000'`), so it adds a new condition area, resulting in a diagnostics area stack that looks like this:

```
DA 1. (condition 2) ERROR 1051 (42S02): Unknown table 'xx'
      (condition 1) ERROR 5 (45000) Unknown table 'xx'
```

The result of `CALL p()` and `SHOW ERRORS` for this example is:

```
mysql> CALL p();
ERROR 5 (45000): Unknown table 'xx'
mysql> SHOW ERRORS;
+-------+------+---------------------------------+
| Level | Code | Message                         |
+-------+------+---------------------------------+
| Error | 1051 | Unknown table 'xx'              |
| Error |    5 | Unknown table 'xx'              |
+-------+------+---------------------------------+
```

RESIGNAL Requires Condition Handler Context

All forms of `RESIGNAL` require that the current context be a condition handler. Otherwise, `RESIGNAL` is illegal and a `RESIGNAL when handler not active` error occurs. For example:

```
mysql> CREATE PROCEDURE p () RESIGNAL;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL p();
ERROR 1645 (0K000): RESIGNAL when handler not active
```

Here is a more difficult example:

```
delimiter //
CREATE FUNCTION f () RETURNS INT
BEGIN
  RESIGNAL;
  RETURN 5;
END//
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION SET @a=f();
  SIGNAL SQLSTATE '55555';
END//
delimiter ;
CALL p();
```

`RESIGNAL` occurs within the stored function `f()`. Although `f()` itself is invoked within the context of the `EXIT` handler, execution within `f()` has its own context, which is not handler context. Thus, `RESIGNAL` within `f()` results in a “handler not active” error.

13.6.7.5 SIGNAL Syntax

```
SIGNAL condition_value
    [SET signal_information_item
    [, signal_information_item] ...]

condition_value: {
    SQLSTATE [VALUE] sqlstate_value
  | condition_name
}

signal_information_item:
    condition_information_item_name = simple_value_specification

condition_information_item_name: {
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
}

condition_name, simple_value_specification:
    (see following discussion)
```

`SIGNAL` is the way to “return” an error. `SIGNAL` provides error information to a handler, to an outer portion of the application, or to the client. Also, it provides control over the error's characteristics (error number, `SQLSTATE` value, message). Without `SIGNAL`, it is necessary to resort to workarounds such as deliberately referring to a nonexistent table to cause a routine to return an error.

No privileges are required to execute the `SIGNAL` statement.

To retrieve information from the diagnostics area, use the `GET DIAGNOSTICS` statement (see Section 13.6.7.3, “GET DIAGNOSTICS Syntax”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

- SIGNAL Overview
- Signal Condition Information Items
- Effect of Signals on Handlers, Cursors, and Statements

SIGNAL Overview

The *`condition_value`* in a `SIGNAL` statement indicates the error value to be returned. It can be an `SQLSTATE` value (a 5-character string literal) or a *`condition_name`* that refers to a named condition previously defined with `DECLARE ... CONDITION` (see Section 13.6.7.1, “DECLARE ... CONDITION Syntax”).

An `SQLSTATE` value can indicate errors, warnings, or “not found.” The first two characters of the value indicate its error class, as discussed in Signal Condition Information Items. Some signal values cause statement termination; see Effect of Signals on Handlers, Cursors, and Statements.

The `SQLSTATE` value for a `SIGNAL` statement should not start with `'00'` because such values indicate success and are not valid for signaling an error. This is true whether the `SQLSTATE` value is specified

directly in the SIGNAL statement or in a named condition referred to in the statement. If the value is invalid, a Bad SQLSTATE error occurs.

To signal a generic SQLSTATE value, use '45000', which means “unhandled user-defined exception.”

The SIGNAL statement optionally includes a SET clause that contains multiple signal items, in a list of *condition_information_item_name* = *simple_value_specification* assignments, separated by commas.

Each *condition_information_item_name* may be specified only once in the SET clause. Otherwise, a Duplicate condition information item error occurs.

Valid *simple_value_specification* designators can be specified using stored procedure or function parameters, stored program local variables declared with DECLARE, user-defined variables, system variables, or literals. A character literal may include a *_charset* introducer.

For information about permissible *condition_information_item_name* values, see Signal Condition Information Items.

The following procedure signals an error or warning depending on the value of pval, its input parameter:

```
CREATE PROCEDURE p (pval INT)
BEGIN
  DECLARE specialty CONDITION FOR SQLSTATE '45000';
  IF pval = 0 THEN
    SIGNAL SQLSTATE '01000';
  ELSEIF pval = 1 THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'An error occurred';
  ELSEIF pval = 2 THEN
    SIGNAL specialty
      SET MESSAGE_TEXT = 'An error occurred';
  ELSE
    SIGNAL SQLSTATE '01000'
      SET MESSAGE_TEXT = 'A warning occurred', MYSQL_ERRNO = 1000;
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'An error occurred', MYSQL_ERRNO = 1001;
  END IF;
END;
```

If pval is 0, p() signals a warning because SQLSTATE values that begin with '01' are signals in the warning class. The warning does not terminate the procedure, and can be seen with SHOW WARNINGS after the procedure returns.

If pval is 1, p() signals an error and sets the MESSAGE_TEXT condition information item. The error terminates the procedure, and the text is returned with the error information.

If pval is 2, the same error is signaled, although the SQLSTATE value is specified using a named condition in this case.

If pval is anything else, p() first signals a warning and sets the message text and error number condition information items. This warning does not terminate the procedure, so execution continues and p() then signals an error. The error does terminate the procedure. The message text and error number set by the warning are replaced by the values set by the error, which are returned with the error information.

SIGNAL is typically used within stored programs, but it is a MySQL extension that it is permitted outside handler context. For example, if you invoke the mysql client program, you can enter any of these statements at the prompt:

```
SIGNAL SQLSTATE '77777';

CREATE TRIGGER t_bi BEFORE INSERT ON t
  FOR EACH ROW SIGNAL SQLSTATE '77777';

CREATE EVENT e ON SCHEDULE EVERY 1 SECOND
  DO SIGNAL SQLSTATE '77777';
```

`SIGNAL` executes according to the following rules:

If the `SIGNAL` statement indicates a particular `SQLSTATE` value, that value is used to signal the condition specified. Example:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  IF divisor = 0 THEN
    SIGNAL SQLSTATE '22012';
  END IF;
END;
```

If the `SIGNAL` statement uses a named condition, the condition must be declared in some scope that applies to the `SIGNAL` statement, and must be defined using an `SQLSTATE` value, not a MySQL error number. Example:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE divide_by_zero CONDITION FOR SQLSTATE '22012';
  IF divisor = 0 THEN
    SIGNAL divide_by_zero;
  END IF;
END;
```

If the named condition does not exist in the scope of the `SIGNAL` statement, an `Undefined CONDITION` error occurs.

If `SIGNAL` refers to a named condition that is defined with a MySQL error number rather than an `SQLSTATE` value, a `SIGNAL/RESIGNAL can only use a CONDITION defined with SQLSTATE` error occurs. The following statements cause that error because the named condition is associated with a MySQL error number:

```
DECLARE no_such_table CONDITION FOR 1051;
SIGNAL no_such_table;
```

If a condition with a given name is declared multiple times in different scopes, the declaration with the most local scope applies. Consider the following procedure:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE my_error CONDITION FOR SQLSTATE '45000';
  IF divisor = 0 THEN
    BEGIN
      DECLARE my_error CONDITION FOR SQLSTATE '22012';
      SIGNAL my_error;
    END;
  END IF;
  SIGNAL my_error;
END;
```

If `divisor` is 0, the first `SIGNAL` statement executes. The innermost `my_error` condition declaration applies, raising `SQLSTATE '22012'`.

If `divisor` is not 0, the second `SIGNAL` statement executes. The outermost `my_error` condition declaration applies, raising `SQLSTATE '45000'`.

For information about how the server chooses handlers when a condition occurs, see Section 13.6.7.6, “Scope Rules for Handlers”.

Signals can be raised within exception handlers:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SIGNAL SQLSTATE VALUE '99999'
      SET MESSAGE_TEXT = 'An error occurred';
  END;
  DROP TABLE no_such_table;
END;
```

`CALL p()` reaches the `DROP TABLE` statement. There is no table named `no_such_table`, so the error handler is activated. The error handler destroys the original error (“no such table”) and makes a new error with `SQLSTATE '99999'` and message `An error occurred`.

Signal Condition Information Items

The following table lists the names of diagnostics area condition information items that can be set in a `SIGNAL` (or `RESIGNAL`) statement. All items are standard SQL except `MYSQL_ERRNO`, which is a MySQL extension. For more information about these items see Section 13.6.7.7, “The MySQL Diagnostics Area”.

```
Item Name             Definition
---------             ----------
CLASS_ORIGIN          VARCHAR(64)
SUBCLASS_ORIGIN       VARCHAR(64)
CONSTRAINT_CATALOG    VARCHAR(64)
CONSTRAINT_SCHEMA     VARCHAR(64)
CONSTRAINT_NAME       VARCHAR(64)
CATALOG_NAME          VARCHAR(64)
SCHEMA_NAME           VARCHAR(64)
TABLE_NAME            VARCHAR(64)
COLUMN_NAME           VARCHAR(64)
CURSOR_NAME           VARCHAR(64)
MESSAGE_TEXT          VARCHAR(128)
MYSQL_ERRNO           SMALLINT UNSIGNED
```

The character set for character items is UTF-8.

It is illegal to assign `NULL` to a condition information item in a `SIGNAL` statement.

A `SIGNAL` statement always specifies an `SQLSTATE` value, either directly, or indirectly by referring to a named condition defined with an `SQLSTATE` value. The first two characters of an `SQLSTATE` value are its class, and the class determines the default value for the condition information items:

- Class = `'00'` (success)

  Illegal. `SQLSTATE` values that begin with `'00'` indicate success and are not valid for `SIGNAL`.

- Class = `'01'` (warning)

```
MESSAGE_TEXT = 'Unhandled user-defined warning condition';
MYSQL_ERRNO = ER_SIGNAL_WARN
```

- Class = `'02'` (not found)

```
MESSAGE_TEXT = 'Unhandled user-defined not found condition';
MYSQL_ERRNO = ER_SIGNAL_NOT_FOUND
```

- Class > `'02'` (exception)

```
MESSAGE_TEXT = 'Unhandled user-defined exception condition';
MYSQL_ERRNO = ER_SIGNAL_EXCEPTION
```

For legal classes, the other condition information items are set as follows:

```
CLASS_ORIGIN = SUBCLASS_ORIGIN = '';
CONSTRAINT_CATALOG = CONSTRAINT_SCHEMA = CONSTRAINT_NAME = '';
CATALOG_NAME = SCHEMA_NAME = TABLE_NAME = COLUMN_NAME = '';
CURSOR_NAME = '';
```

The error values that are accessible after `SIGNAL` executes are the `SQLSTATE` value raised by the `SIGNAL` statement and the `MESSAGE_TEXT` and `MYSQL_ERRNO` items. These values are available from the C API:

- `mysql_sqlstate()` returns the `SQLSTATE` value.
- `mysql_errno()` returns the `MYSQL_ERRNO` value.
- `mysql_error()` returns the `MESSAGE_TEXT` value.

At the SQL level, the output from `SHOW WARNINGS` and `SHOW ERRORS` indicates the `MYSQL_ERRNO` and `MESSAGE_TEXT` values in the `Code` and `Message` columns.

To retrieve information from the diagnostics area, use the `GET DIAGNOSTICS` statement (see Section 13.6.7.3, “GET DIAGNOSTICS Syntax”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

Effect of Signals on Handlers, Cursors, and Statements

Signals have different effects on statement execution depending on the signal class. The class determines how severe an error is. MySQL ignores the value of the `sql_mode` system variable; in particular, strict SQL mode does not matter. MySQL also ignores `IGNORE`: The intent of `SIGNAL` is to raise a user-generated error explicitly, so a signal is never ignored.

In the following descriptions, “unhandled” means that no handler for the signaled `SQLSTATE` value has been defined with `DECLARE ... HANDLER`.

- Class = `'00'` (success)

  Illegal. `SQLSTATE` values that begin with `'00'` indicate success and are not valid for `SIGNAL`.

- Class = `'01'` (warning)

  The value of the `warning_count` system variable goes up. `SHOW WARNINGS` shows the signal. `SQLWARNING` handlers catch the signal.

Warnings cannot be returned from stored functions because the `RETURN` statement that causes the function to return clears the diagnostic area. The statement thus clears any warnings that may have been present there (and resets `warning_count` to 0).

- Class = `'02'` (not found)

  `NOT FOUND` handlers catch the signal. There is no effect on cursors. If the signal is unhandled in a stored function, statements end.

- Class > `'02'` (exception)

  `SQLEXCEPTION` handlers catch the signal. If the signal is unhandled in a stored function, statements end.

- Class = `'40'`

  Treated as an ordinary exception.

13.6.7.6 Scope Rules for Handlers

A stored program may include handlers to be invoked when certain conditions occur within the program. The applicability of each handler depends on its location within the program definition and on the condition or conditions that it handles:

- A handler declared in a `BEGIN ... END` block is in scope only for the SQL statements following the handler declarations in the block. If the handler itself raises a condition, it cannot handle that condition, nor can any other handlers declared in the block. In the following example, handlers `H1` and `H2` are in scope for conditions raised by statements *`stmt1`* and *`stmt2`*. But neither `H1` nor `H2` are in scope for conditions raised in the body of `H1` or `H2`.

```
BEGIN -- outer block
  DECLARE EXIT HANDLER FOR ...;  -- handler H1
  DECLARE EXIT HANDLER FOR ...;  -- handler H2
  stmt1;
  stmt2;
END;
```

- A handler is in scope only for the block in which it is declared, and cannot be activated for conditions occurring outside that block. In the following example, handler `H1` is in scope for *`stmt1`* in the inner block, but not for *`stmt2`* in the outer block:

```
BEGIN -- outer block
  BEGIN -- inner block
    DECLARE EXIT HANDLER FOR ...;  -- handler H1
    stmt1;
  END;
  stmt2;
END;
```

- A handler can be specific or general. A specific handler is for a MySQL error code, `SQLSTATE` value, or condition name. A general handler is for a condition in the `SQLWARNING`, `SQLEXCEPTION`, or `NOT FOUND` class. Condition specificity is related to condition precedence, as described later.

Multiple handlers can be declared in different scopes and with different specificities. For example, there might be a specific MySQL error code handler in an outer block, and a general `SQLWARNING` handler in an inner block. Or there might be handlers for a specific MySQL error code and the general `SQLWARNING` class in the same block.

Whether a handler is activated depends not only on its own scope and condition value, but on what other handlers are present. When a condition occurs in a stored program, the server searches for applicable handlers in the current scope (current `BEGIN ... END` block). If there are no applicable handlers, the search continues outward with the handlers in each successive containing scope (block). When the server finds one or more applicable handlers at a given scope, it chooses among them based on condition precedence:

- A MySQL error code handler takes precedence over an `SQLSTATE` value handler.
- An `SQLSTATE` value handler takes precedence over general `SQLWARNING`, `SQLEXCEPTION`, or `NOT FOUND` handlers.
- An `SQLEXCEPTION` handler takes precedence over an `SQLWARNING` handler.
- It is possible to have several applicable handlers with the same precedence. For example, a statement could generate multiple warnings with different error codes, for each of which an error-specific handler exists. In this case, the choice of which handler the server activates is nondeterministic, and may change depending on the circumstances under which the condition occurs.

One implication of the handler selection rules is that if multiple applicable handlers occur in different scopes, handlers with the most local scope take precedence over handlers in outer scopes, even over those for more specific conditions.

If there is no appropriate handler when a condition occurs, the action taken depends on the class of the condition:

- For `SQLEXCEPTION` conditions, the stored program terminates at the statement that raised the condition, as if there were an `EXIT` handler. If the program was called by another stored program, the calling program handles the condition using the handler selection rules applied to its own handlers.
- For `SQLWARNING` conditions, the program continues executing, as if there were a `CONTINUE` handler.
- For `NOT FOUND` conditions, if the condition was raised normally, the action is `CONTINUE`. If it was raised by `SIGNAL` or `RESIGNAL`, the action is `EXIT`.

The following examples demonstrate how MySQL applies the handler selection rules.

This procedure contains two handlers, one for the specific `SQLSTATE` value (`'42S02'`) that occurs for attempts to drop a nonexistent table, and one for the general `SQLEXCEPTION` class:

```
CREATE PROCEDURE p1()
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
    SELECT 'SQLSTATE handler was activated' AS msg;
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SELECT 'SQLEXCEPTION handler was activated' AS msg;

  DROP TABLE test.t;
END;
```

Both handlers are declared in the same block and have the same scope. However, `SQLSTATE` handlers take precedence over `SQLEXCEPTION` handlers, so if the table `t` is nonexistent, the `DROP TABLE` statement raises a condition that activates the `SQLSTATE` handler:

```
mysql> CALL p1();
+------------------------------+
| msg                          |
```

```
+--------------------------------+
| SQLSTATE handler was activated |
+--------------------------------+
```

This procedure contains the same two handlers. But this time, the `DROP TABLE` statement and `SQLEXCEPTION` handler are in an inner block relative to the `SQLSTATE` handler:

```
CREATE PROCEDURE p2()
BEGIN -- outer block
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      SELECT 'SQLEXCEPTION handler was activated' AS msg;

    DROP TABLE test.t; -- occurs within inner block
  END;
END;
```

In this case, the handler that is more local to where the condition occurs takes precedence. The `SQLEXCEPTION` handler activates, even though it is more general than the `SQLSTATE` handler:

```
mysql> CALL p2();
+-----------------------------------+
| msg                               |
+-----------------------------------+
| SQLEXCEPTION handler was activated |
+-----------------------------------+
```

In this procedure, one of the handlers is declared in a block inner to the scope of the `DROP TABLE` statement:

```
CREATE PROCEDURE p3()
BEGIN -- outer block
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SELECT 'SQLEXCEPTION handler was activated' AS msg;
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  END;

  DROP TABLE test.t; -- occurs within outer block
END;
```

Only the `SQLEXCEPTION` handler applies because the other one is not in scope for the condition raised by the `DROP TABLE`:

```
mysql> CALL p3();
+-----------------------------------+
| msg                               |
+-----------------------------------+
| SQLEXCEPTION handler was activated |
+-----------------------------------+
```

In this procedure, both handlers are declared in a block inner to the scope of the `DROP TABLE` statement:

```
CREATE PROCEDURE p4()
BEGIN -- outer block
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
```

```
      SELECT 'SQLEXCEPTION handler was activated' AS msg;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  END;

  DROP TABLE test.t; -- occurs within outer block
END;
```

Neither handler applies because they are not in scope for the `DROP TABLE`. The condition raised by the statement goes unhandled and terminates the procedure with an error:

```
mysql> CALL p4();
ERROR 1051 (42S02): Unknown table 'test.t'
```

13.6.7.7 The MySQL Diagnostics Area

SQL statements produce diagnostic information that populates the diagnostics area. Standard SQL has a diagnostics area stack, containing a diagnostics area for each nested execution context. Standard SQL also supports `GET STACKED DIAGNOSTICS` syntax for referring to the second diagnostics area during condition handler execution.

The following discussion describes the structure of the diagnostics area in MySQL, the information items recognized by MySQL, how statements clear and set the diagnostics area, and how diagnostics areas are pushed to and popped from the stack.

- Diagnostics Area Structure
- Diagnostics Area Information Items
- How the Diagnostics Area is Cleared and Populated
- How the Diagnostics Area Stack Works
- Diagnostics Area-Related System Variables

Diagnostics Area Structure

The diagnostics area contains two kinds of information:

- Statement information, such as the number of conditions that occurred or the affected-rows count.
- Condition information, such as the error code and message. If a statement raises multiple conditions, this part of the diagnostics area has a condition area for each one. If a statement raises no conditions, this part of the diagnostics area is empty.

For a statement that produces three conditions, the diagnostics area contains statement and condition information like this:

```
Statement information:
  row count
  ... other statement information items ...
Condition area list:
  Condition area 1:
    error code for condition 1
    error message for condition 1
    ... other condition information items ...
  Condition area 2:
    error code for condition 2:
```

```
    error message for condition 2
    ... other condition information items ...
  Condition area 3:
    error code for condition 3
    error message for condition 3
    ... other condition information items ...
```

Diagnostics Area Information Items

The diagnostics area contains statement and condition information items. Numeric items are integers. The character set for character items is UTF-8. No item can be `NULL`. If a statement or condition item is not set by a statement that populates the diagnostics area, its value is 0 or the empty string, depending on the item data type.

The statement information part of the diagnostics area contains these items:

- `NUMBER`: An integer indicating the number of condition areas that have information.
- `ROW_COUNT`: An integer indicating the number of rows affected by the statement. `ROW_COUNT` has the same value as the `ROW_COUNT()` function (see Section 12.15, “Information Functions”).

The condition information part of the diagnostics area contains a condition area for each condition. Condition areas are numbered from 1 to the value of the `NUMBER` statement condition item. If `NUMBER` is 0, there are no condition areas.

Each condition area contains the items in the following list. All items are standard SQL except `MYSQL_ERRNO`, which is a MySQL extension. The definitions apply for conditions generated other than by a signal (that is, by a `SIGNAL` or `RESIGNAL` statement). For nonsignal conditions, MySQL populates only those condition items not described as always empty. The effects of signals on the condition area are described later.

- `CLASS_ORIGIN`: A string containing the class of the `RETURNED_SQLSTATE` value. If the `RETURNED_SQLSTATE` value begins with a class value defined in SQL standards document ISO 9075-2 (section 24.1, SQLSTATE), `CLASS_ORIGIN` is `'ISO 9075'`. Otherwise, `CLASS_ORIGIN` is `'MySQL'`.
- `SUBCLASS_ORIGIN`: A string containing the subclass of the `RETURNED_SQLSTATE` value. If `CLASS_ORIGIN` is `'ISO 9075'` or `RETURNED_SQLSTATE` ends with `'000'`, `SUBCLASS_ORIGIN` is `'ISO 9075'`. Otherwise, `SUBCLASS_ORIGIN` is `'MySQL'`.
- `RETURNED_SQLSTATE`: A string that indicates the `SQLSTATE` value for the condition.
- `MESSAGE_TEXT`: A string that indicates the error message for the condition.
- `MYSQL_ERRNO`: An integer that indicates the MySQL error code for the condition.
- `CONSTRAINT_CATALOG`, `CONSTRAINT_SCHEMA`, `CONSTRAINT_NAME`: Strings that indicate the catalog, schema, and name for a violated constraint. They are always empty.
- `CATALOG_NAME`, `SCHEMA_NAME`, `TABLE_NAME`, `COLUMN_NAME`: Strings that indicate the catalog, schema, table, and column related to the condition. They are always empty.
- `CURSOR_NAME`: A string that indicates the cursor name. This is always empty.

For the `RETURNED_SQLSTATE`, `MESSAGE_TEXT`, and `MYSQL_ERRNO` values for particular errors, see Section B.3.1, “Server Error Message Reference”.

If a `SIGNAL` (or `RESIGNAL`) statement populates the diagnostics area, its `SET` clause can assign to any condition information item except `RETURNED_SQLSTATE` any value that is legal for the item data type.

`SIGNAL` also sets the `RETURNED_SQLSTATE` value, but not directly in its `SET` clause. That value comes from the `SIGNAL` statement `SQLSTATE` argument.

`SIGNAL` also sets statement information items. It sets `NUMBER` to 1. It sets `ROW_COUNT` to −1 for errors and 0 otherwise.

How the Diagnostics Area is Cleared and Populated

Nondiagnostic SQL statements populate the diagnostics area automatically, and its contents can be set explicitly with the `SIGNAL` and `RESIGNAL` statements. The diagnostics area can be examined with `GET DIAGNOSTICS` to extract specific items, or with `SHOW WARNINGS` or `SHOW ERRORS` to see conditions or errors.

SQL statements clear and set the diagnostics area as follows:

- When the server starts executing a statement after parsing it, it clears the diagnostics area for nondiagnostic statements. Diagnostic statements do not clear the diagnostics area. These statements are diagnostic:
  - `GET DIAGNOSTICS`
  - `SHOW ERRORS`
  - `SHOW WARNINGS`
- If a statement raises a condition, the diagnostics area is cleared of conditions that belong to earlier statements. The exception is that conditions raised by `GET DIAGNOSTICS` and `RESIGNAL` are added to the diagnostics area without clearing it.

Thus, even a statement that does not normally clear the diagnostics area when it begins executing clears it if the statement raises a condition.

The following example shows the effect of various statements on the diagnostics area, using `SHOW WARNINGS` to display information about conditions stored there.

This `DROP TABLE` statement clears the diagnostics area and populates it when the condition occurs:

```
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+-------+------+-------------------------------------+
| Level | Code | Message                             |
+-------+------+-------------------------------------+
| Note  | 1051 | Unknown table 'test.no_such_table' |
+-------+------+-------------------------------------+
1 row in set (0.00 sec)
```

This `SET` statement generates an error, so it clears and populates the diagnostics area:

```
mysql> SET @x = @@x;
ERROR 1193 (HY000): Unknown system variable 'x'

mysql> SHOW WARNINGS;
+-------+------+----------------------------+
| Level | Code | Message                    |
+-------+------+----------------------------+
| Error | 1193 | Unknown system variable 'x' |
```

```
+-------+------+-----------------------------+
1 row in set (0.00 sec)
```

The previous `SET` statement produced a single condition, so 1 is the only valid condition number for `GET DIAGNOSTICS` at this point. The following statement uses a condition number of 2, which produces a warning that is added to the diagnostics area without clearing it:

```
mysql> GET DIAGNOSTICS CONDITION 2 @p = MESSAGE_TEXT;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-------+------+-----------------------------+
| Level | Code | Message                     |
+-------+------+-----------------------------+
| Error | 1193 | Unknown system variable 'xx' |
| Error | 1753 | Invalid condition number     |
+-------+------+-----------------------------+
2 rows in set (0.00 sec)
```

Now there are two conditions in the diagnostics area, so the same `GET DIAGNOSTICS` statement succeeds:

```
mysql> GET DIAGNOSTICS CONDITION 2 @p = MESSAGE_TEXT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @p;
+--------------------------+
| @p                       |
+--------------------------+
| Invalid condition number |
+--------------------------+
1 row in set (0.01 sec)
```

How the Diagnostics Area Stack Works

When a push to the diagnostics area stack occurs, the first (current) diagnostics area becomes the second (stacked) diagnostics area and a new current diagnostics area is created as a copy of it. Diagnostics areas are pushed to and popped from the stack under the following circumstances:

- Execution of a stored program

  A push occurs before the program executes and a pop occurs afterward. If the stored program ends while handlers are executing, there can be more than one diagnostics area to pop; this occurs due to an exception for which there are no appropriate handlers or due to `RETURN` in the handler.

  Any warning or error conditions in the popped diagnostics areas then are added to the current diagnostics area, except that, for triggers, only errors are added. When the stored program ends, the caller sees these conditions in its current diagonstics area.

- Execution of a condition handler within a stored program

  When a push occurs as a result of condition handler activation, the stacked diagnostics area is the area that was current within the stored program prior to the push. The new now-current diagnostics area is the handler's current diagnostics area. `GET [CURRENT] DIAGNOSTICS` and `GET STACKED DIAGNOSTICS` can be used within the handler to access the contents of the current (handler) and stacked (stored program) diagnostics areas. Initially, they return the same result, but statements executing within the handler modify the current diagnostics area, clearing and setting its contents according to the normal rules (see How the Diagnostics Area is Cleared and Populated). The stacked diagnostics area cannot be modified by statements executing within the handler except `RESIGNAL`.

If the handler executes successfully, the current (handler) diagnostics area is popped and the stacked (stored program) diagnostics area again becomes the current diagnostics area. Conditions added to the handler diagnostics area during handler execution are added to the current diagnostics area.

- Execution of `RESIGNAL`

  The `RESIGNAL` statement passes on the error condition information that is available during execution of a condition handler within a compound statement inside a stored program. `RESIGNAL` may change some or all information before passing it on, modifying the diagnostics stack as described in Section 13.6.7.4, “RESIGNAL Syntax”.

Diagnostics Area-Related System Variables

Certain system variables control or are related to some aspects of the diagnostics area:

- `max_error_count` controls the number of condition areas in the diagnostics area. If more conditions than this occur, MySQL silently discards information for the excess conditions. (Conditions added by `RESIGNAL` are always added, with older conditions being discarded as necessary to make room.)
- `warning_count` indicates the number of conditions that occurred. This includes errors, warnings, and notes. Normally, `NUMBER` and `warning_count` are the same. However, as the number of conditions generated exceeds `max_error_count`, the value of `warning_count` continues to rise whereas `NUMBER` remains capped at `max_error_count` because no additional conditions are stored in the diagnostics area.
- `error_count` indicates the number of errors that occurred. This value includes “not found” and exception conditions, but excludes warnings and notes. Like `warning_count`, its value can exceed `max_error_count`.
- If the `sql_notes` system variable is set to 0, notes are not stored and do not increment `warning_count`.

Example: If `max_error_count` is 10, the diagnostics area can contain a maximum of 10 condition areas. Suppose that a statement raises 20 conditions, 12 of which are errors. In that case, the diagnostics area contains the first 10 conditions, `NUMBER` is 10, `warning_count` is 20, and `error_count` is 12.

Changes to the value of `max_error_count` have no effect until the next attempt to modify the diagnostics area. If the diagnostics area contains 10 condition areas and `max_error_count` is set to 5, that has no immediate effect on the size or content of the diagnostics area.

13.6.7.8 Condition Handling and OUT or INOUT Parameters

If a stored procedure exits with an unhandled exception, modified values of `OUT` and `INOUT` parameters are not propogated back to the caller.

If an exception is handled by a `CONTINUE` or `EXIT` handler that contains a `RESIGNAL` statement, execution of `RESIGNAL` pops the Diagnostics Area stack, thus signalling the exception (that is, the information that existed before entry into the handler). If the exception is an error, the values of `OUT` and `INOUT` parameters are not propogated back to the caller.

13.6.8 Restrictions on Condition Handling

`SIGNAL`, `RESIGNAL`, and `GET DIAGNOSTICS` are not permissible as prepared statements. For example, this statement is invalid:

```
PREPARE stmt1 FROM 'SIGNAL SQLSTATE "02000"';
```

`SQLSTATE` values in class `'04'` are not treated specially. They are handled the same as other exceptions.

In standard SQL, the first condition relates to the `SQLSTATE` value returned for the previous SQL statement. In MySQL, this is not guaranteed, so to get the main error, you cannot do this:

```
GET DIAGNOSTICS CONDITION 1 @errno = MYSQL_ERRNO;
```

Instead, do this:

```
GET DIAGNOSTICS @cno = NUMBER;
GET DIAGNOSTICS CONDITION @cno @errno = MYSQL_ERRNO;
```

13.7 Database Administration Statements

13.7.1 Account Management Statements

MySQL account information is stored in the tables of the `mysql` system database. This database and the access control system are discussed extensively in Chapter 5, *MySQL Server Administration*, which you should consult for additional details.

Important

Some MySQL releases introduce changes to the grant tables to add new privileges or features. To make sure that you can take advantage of any new capabilities, update your grant tables to the current structure whenever you upgrade MySQL. See Section 2.11, “Upgrading MySQL”.

When the `read_only` system variable is enabled, account-management statements require the `CONNECTION_ADMIN` or `SUPER` privilege, in addition to any other required privileges. This is because they modify tables in the `mysql` system database.

Account management statements are atomic and crash safe. For more information, see Section 13.1.1, “Atomic Data Definition Statement Support”.

13.7.1.1 ALTER USER Syntax

```
ALTER USER [IF EXISTS]
    user [auth_option] [, user [auth_option]] ...
    [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
    [WITH resource_option [resource_option] ...]
    [password_option | lock_option] ...

ALTER USER [IF EXISTS] USER() user_func_auth_option

ALTER USER [IF EXISTS]
    user DEFAULT ROLE
    {NONE | ALL | role [, role ] ...}

user:
    (see Section 6.2.4, “Specifying Account Names”)

auth_option: {
    IDENTIFIED BY 'auth_string'
```

```
        [REPLACE 'current_auth_string']
        [RETAIN CURRENT PASSWORD]
  | IDENTIFIED BY RANDOM PASSWORD
        [REPLACE 'current_auth_string']
        [RETAIN CURRENT PASSWORD]
  | IDENTIFIED WITH auth_plugin
  | IDENTIFIED WITH auth_plugin BY 'auth_string'
        [REPLACE 'current_auth_string']
        [RETAIN CURRENT PASSWORD]
  | IDENTIFIED WITH auth_plugin BY RANDOM PASSWORD
        [REPLACE 'current_auth_string']
        [RETAIN CURRENT PASSWORD]
  | IDENTIFIED WITH auth_plugin AS 'auth_string'
  | DISCARD OLD PASSWORD
}

user_func_auth_option: {
    IDENTIFIED BY 'auth_string'
        [REPLACE 'current_auth_string']
        [RETAIN CURRENT PASSWORD]
  | DISCARD OLD PASSWORD
}

tls_option: {
   SSL
 | X509
 | CIPHER 'cipher'
 | ISSUER 'issuer'
 | SUBJECT 'subject'
}

resource_option: {
    MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
}

password_option: {
    PASSWORD EXPIRE [DEFAULT | NEVER | INTERVAL N DAY]
  | PASSWORD HISTORY {DEFAULT | N}
  | PASSWORD REUSE INTERVAL {DEFAULT | N DAY}
  | PASSWORD REQUIRE CURRENT [DEFAULT | OPTIONAL]
}

lock_option: {
    ACCOUNT LOCK
  | ACCOUNT UNLOCK
}
```

The `ALTER USER` statement modifies MySQL accounts. It enables authentication, role, SSL/TLS, resource-limit, and password-management properties to be modified for existing accounts. It can also be used to lock and unlock accounts.

In most cases, `ALTER USER` requires the global `CREATE USER` privilege, or the `UPDATE` privilege for the `mysql` system database. The exceptions are:

- Any client who connects to the server using a nonanonymous account can change the password for that account. (In particular, you can change your own password.) To see which account the server authenticated you as, invoke the `CURRENT_USER()` function:

  ```
  SELECT CURRENT_USER();
  ```

- For `DEFAULT ROLE` syntax, `ALTER USER` requires these privileges:

  - Setting the default roles for another user requires the global `CREATE USER` privilege, or the `UPDATE` privilege for the `mysql.default_roles` system table.
  - Setting the default roles for yourself requires no special privileges, as long as the roles you want as the default have been granted to you.
- Statements that modify secondary passwords require these privileges:
  - The `APPLICATION_PASSWORD_ADMIN` privilege is required to use the `RETAIN CURRENT PASSWORD` or `DISCARD OLD PASSWORD` clause for `ALTER USER` statements that apply to your own account. The privilege is required to manipulate your own secondary password because most users require only one password.
  - If an account is to be permitted to manipulate secondary passwords for all accounts, it should be granted the `CREATE USER` privilege rather than `APPLICATION_PASSWORD_ADMIN`.

When the `read_only` system variable is enabled, `ALTER USER` additionally requires the `CONNECTION_ADMIN` or `SUPER` privilege.

By default, an error occurs if you try to modify a user that does not exist. If the `IF EXISTS` clause is given, the statement produces a warning for each named user that does not exist, rather than an error.

Important

Under some circumstances, `ALTER USER` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about the conditions under which this occurs for the server logs and how to control it, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, see Section 4.5.1.3, “mysql Client Logging”.

There are several aspects to the `ALTER USER` statement, described under the following topics:

- ALTER USER Overview
- ALTER USER Authentication Options
- ALTER USER Role Options
- ALTER USER SSL/TLS Options
- ALTER USER Resource-Limit Options
- ALTER USER Password-Management Options
- ALTER USER Account-Locking Options
- ALTER USER Binary Logging

ALTER USER Overview

For each affected account, `ALTER USER` modifies the corresponding row in the `mysql.user` system table to reflect the properties specified in the statement. Unspecified properties retain their current values.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. The host name part of the account name, if omitted, defaults to `'%'`. It is also possible to specify `CURRENT_USER` or `CURRENT_USER()` to refer to the account associated with the current session.

For one syntax only, the account may be specified with the `USER()` function:

```
ALTER USER USER() IDENTIFIED BY 'auth_string';
```

This syntax enables changing your own password without naming your account literally. (The syntax also supports the `REPLACE`, `RETAIN CURRENT PASSWORD`, and `DISCARD OLD PASSWORD` clauses described at ALTER USER Authentication Options.)

For `ALTER USER` syntax that permits an *`auth_option`* value to follow a *`user`* value, *`auth_option`* indicates how the account authenticates by specifying an account authentication plugin, credentials (for example, a password), or both. Each *`auth_option`* value applies *only* to the account named immediately preceding it.

Following the *`user`* specifications, the statement may include options for SSL/TLS, resource-limit, password-management, and locking properties. All such options are *global* to the statement and apply to *all* accounts named in the statement.

Example: Change an account's password and expire it. As a result, the user must connect with the named password and choose a new one at the next connection:

```
ALTER USER 'jeffrey'@'localhost'
  IDENTIFIED BY 'new_password' PASSWORD EXPIRE;
```

Example: Modify an account to use the `sha256_password` authentication plugin and the given password. Require that a new password be chosen every 180 days:

```
ALTER USER 'jeffrey'@'localhost'
  IDENTIFIED WITH sha256_password BY 'new_password'
  PASSWORD EXPIRE INTERVAL 180 DAY;
```

Example: Lock or unlock an account:

```
ALTER USER 'jeffrey'@'localhost' ACCOUNT LOCK;
ALTER USER 'jeffrey'@'localhost' ACCOUNT UNLOCK;
```

Example: Require an account to connect using SSL and establish a limit of 20 connections per hour:

```
ALTER USER 'jeffrey'@'localhost'
  REQUIRE SSL WITH MAX_CONNECTIONS_PER_HOUR 20;
```

Example: Alter multiple accounts, specifying some per-account properties and some global properties:

```
ALTER USER
  'jeffrey'@'localhost'
    IDENTIFIED BY 'jeffrey_new_password',
  'jeanne'@'localhost',
  'josh'@'localhost'
    IDENTIFIED BY 'josh_new_password'
    REPLACE 'josh_current_password'
    RETAIN CURRENT PASSWORD
  REQUIRE SSL WITH MAX_USER_CONNECTIONS 2
  PASSWORD HISTORY 5;
```

The `IDENTIFIED BY` value following `jeffrey` applies only to its immediately preceding account, so it changes the password to `'jeffrey_new_password'` only for `jeffrey`. For `jeanne`, there is no per-

account value (thus leaving the password unchanged). For `josh`, `IDENTIFIED BY` establishes a new password (`'josh_new_password'`), `REPLACE` is specified to verify that the user issuing the `ALTER USER` statement knows the current password (`'josh_current_password'`), and that current password is also retained as the account secondary password. (As a result, `josh` can connect with either the primary or secondary password.)

The remaining properties apply globally to all accounts named in the statement, so for both accounts:

- Connections are required to use SSL.
- The account can be used for a maximum of two simultaneous connections.
- Password changes cannot reuse any of the five most recent passwords.

Example: Discard the secondary password for `josh`, leaving the account with only its primary password:

```
ALTER USER 'josh'@'localhost' DISCARD OLD PASSWORD;
```

In the absence of a particular type of option, the account remains unchanged in that respect. For example, with no locking option, the locking state of the account is not changed.

ALTER USER Authentication Options

An account name may be followed by an *`auth_option`* authentication option that specifies the account authentication plugin, credentials, or both. It may also include a password-verification clause that specifies the account current password to be replaced, and clauses that manage whether an account has a secondary password.

Note

Clauses for random password generation, password verification, and secondary passwords apply only to accounts that use an authentication plugin that stores credentials internally to MySQL. For accounts that use plugins that perform authentication against a credentials system that is external to MySQL, password management must be handled externally against that system as well. For more information about internal credentials storage, see Section 6.2.15, “Password Management”.

- *`auth_plugin`* names an authentication plugin. The plugin name can be a quoted string literal or an unquoted name. Plugin names are stored in the `plugin` column of the `mysql.user` system table.

  For *`auth_option`* syntax that does not specify an authentication plugin, the default plugin is indicated by the value of the `default_authentication_plugin` system variable. For descriptions of each plugin, see Section 6.4.1, “Authentication Plugins”.

- Credentials that are stored internally are stored in the `mysql.user` system table. An `'auth_string'` value or `RANDOM PASSWORD` specifies account credentials, either as a cleartext (unencrypted) string or hashed in the format expected by the authentication plugin associated with the account, respectively:

  - For syntax that uses `BY 'auth_string'`, the string is cleartext and is passed to the authentication plugin for possible hashing. The result returned by the plugin is stored in the `mysql.user` table. A plugin may use the value as specified, in which case no hashing occurs.

  - For syntax that uses `BY RANDOM PASSWORD`, MySQL generates a random password and as cleartext and passes it to the authentication plugin for possible hashing. The result returned by the plugin is

stored in the `mysql.user` table. A plugin may use the value as specified, in which case no hashing occurs.

Randomly generated passwords are available as of MySQL 8.0.18 and have the characteristics described in Random Password Generation.

- For syntax that uses `AS 'auth_string'`, the string is assumed to be already in the format the authentication plugin requires, and is stored as is in the `mysql.user` table. If a plugin requires a hashed value, the value must be already hashed in a format appropriate for the plugin, or the value will not be usable by the plugin and correct authentication of client connections will not occur.

  As of MySQL 8.0.17, a hashed string can be either a string literal or a hexadecimal value. The latter corresponds to the type of value displayed by `SHOW CREATE USER` for password hashes containing unprintable characters when the `print_identified_with_as_hex` system variable is enabled.

- If an authentication plugin performs no hashing of the authentication string, the `BY 'auth_string'` and `AS 'auth_string'` clauses have the same effect: The authentication string is stored as is in the `mysql.user` system table.

- The `REPLACE 'current_auth_string'` clause performs password verification and is available as of MySQL 8.0.13. If given:

  - `REPLACE` specifies the account current password to be replaced, as a cleartext (unencrypted) string.
  - The clause must be given if password changes for the are required to specify the current password, as verification that the user attempting to make the change actually knows the current password.
  - The clause is optional if password changes for the account may but need not specify the current password.
  - The statement fails if the clause is given but does not match the current password, even if the clause is optional.
  - `REPLACE` can be specified only when changing the account password for the current user.

  For more information about password verification by specifying the current password, see Section 6.2.15, “Password Management”.

- The `RETAIN CURRENT PASSWORD` and `DISCARD OLD PASSWORD` clauses implement dual-password capability and are available as of MySQL 8.0.14. Both are optional, but if given, have the following effects:

  - `RETAIN CURRENT PASSWORD` retains an account current password as its secondary password, replacing any existing secondary password. The new password becomes the primary password, but clients can use the account to connect to the server using either the primary or secondary password. (Exception: If the new password specified by the `ALTER USER` statement is empty, the secondary password becomes empty as well, even if `RETAIN CURRENT PASSWORD` is given.)
  - If you specify `RETAIN CURRENT PASSWORD` for an account that has an empty primary password, the statement fails.
  - If an account has a secondary password and you change its primary password without specifying `RETAIN CURRENT PASSWORD`, the secondary password remains unchanged.
  - If you change the authentication plugin assigned to the account, the secondary password is discarded. If you change the authentication plugin and also specify `RETAIN CURRENT PASSWORD`, the statement fails.

- `DISCARD OLD PASSWORD` discards the secondary password, if one exists. The account retains only its primary password, and clients can use the account to connect to the server only with the primary password.

For more information about use of dual passwords, see Section 6.2.15, “Password Management”.

`ALTER USER` permits these *`auth_option`* syntaxes:

- `IDENTIFIED BY '`*`auth_string`*`' [REPLACE '`*`current_auth_string`*`'] [RETAIN CURRENT PASSWORD]`

  Sets the account authentication plugin to the default plugin, passes the cleartext `'`*`auth_string`*`'` value to the plugin for possible hashing, and stores the result in the account row in the `mysql.user` system table.

  The `REPLACE` clause, if given, specifies the account current password, as described previously in this section.

  The `RETAIN CURRENT PASSWORD` clause, if given, causes the account current password to be retained as its secondary password, as described previously in this section.

- `IDENTIFIED BY RANDOM PASSWORD [REPLACE '`*`current_auth_string`*`'] [RETAIN CURRENT PASSWORD]`

  Sets the account authentication plugin to the default plugin, generates a random password, passes the cleartext password value to the plugin for possible hashing, and stores the result in the account row in the `mysql.user` system table. The statement also returns the cleartext password in a result set to make it available to the user or application executing the statement. For details about the result set and characteristics of randomly generated passwords, see Random Password Generation.

  The `REPLACE` clause, if given, specifies the account current password, as described previously in this section.

  The `RETAIN CURRENT PASSWORD` clause, if given, causes the account current password to be retained as its secondary password, as described previously in this section.

- `IDENTIFIED WITH `*`auth_plugin`*

  Sets the account authentication plugin to *`auth_plugin`*, clears the credentials to the empty string (the credentials are associated with the old authentication plugin, not the new one), and stores the result in the account row in the `mysql.user` system table.

  In addition, the password is marked expired. The user must choose a new one when next connecting.

- `IDENTIFIED WITH `*`auth_plugin`*` BY '`*`auth_string`*`' [REPLACE '`*`current_auth_string`*`'] [RETAIN CURRENT PASSWORD]`

  Sets the account authentication plugin to *`auth_plugin`*, passes the cleartext `'`*`auth_string`*`'` value to the plugin for possible hashing, and stores the result in the account row in the `mysql.user` system table.

  The `REPLACE` clause, if given, specifies the account current password, as described previously in this section.

  The `RETAIN CURRENT PASSWORD` clause, if given, causes the account current password to be retained as its secondary password, as described previously in this section.

- `IDENTIFIED WITH auth_plugin BY RANDOM PASSWORD [REPLACE 'current_auth_string'] [RETAIN CURRENT PASSWORD]`

  Sets the account authentication plugin to *`auth_plugin`*, generates a random password, passes the cleartext password value to the plugin for possible hashing, and stores the result in the account row in the `mysql.user` system table. The statement also returns the cleartext password in a result set to make it available to the user or application executing the statement. For details about the result set and characteristics of randomly generated passwords, see Random Password Generation.

  The `REPLACE` clause, if given, specifies the account current password, as described previously in this section.

  The `RETAIN CURRENT PASSWORD` clause, if given, causes the account current password to be retained as its secondary password, as described previously in this section.

- `IDENTIFIED WITH auth_plugin AS 'auth_string'`

  Sets the account authentication plugin to *`auth_plugin`* and stores the *`'auth_string'`* value as is in the `mysql.user` account row. If the plugin requires a hashed string, the string is assumed to be already hashed in the format the plugin requires.

- `DISCARD OLD PASSWORD`

  Discards the account secondary password, if there is one, as described previously in this section.

Example: Specify the password as cleartext; the default plugin is used:

```
ALTER USER 'jeffrey'@'localhost'
  IDENTIFIED BY 'password';
```

Example: Specify the authentication plugin, along with a cleartext password value:

```
ALTER USER 'jeffrey'@'localhost'
  IDENTIFIED WITH mysql_native_password
             BY 'password';
```

Example: Like the preceding example, but in addition, specify the current password as a cleartext value to satisfy any account requirement that the user making the change knows that password:

```
ALTER USER 'jeffrey'@'localhost'
  IDENTIFIED WITH mysql_native_password
             BY 'password'
             REPLACE 'current_password';
```

The preceding statement fails unless the current user is `jeffrey` because `REPLACE` is permitted only for changes to the current user's password.

Example: Establish a new primary password and retain the existing password as the secondary password:

```
ALTER USER 'jeffrey'@'localhost'
  IDENTIFIED BY 'new_password'
  RETAIN CURRENT PASSWORD;
```

Example: Discard the secondary password, leaving the account with only its primary password:

```
ALTER USER 'jeffery'@'localhost' DISCARD OLD PASSWORD;
```

Example: Specify the authentication plugin, along with a hashed password value:

```
ALTER USER 'jeffrey'@'localhost'
  IDENTIFIED WITH mysql_native_password
             AS '*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4';
```

For additional information about setting passwords and authentication plugins, see Section 6.2.14, “Assigning Account Passwords”, and Section 6.2.17, “Pluggable Authentication”.

ALTER USER Role Options

`ALTER USER ... DEFAULT ROLE` defines which roles become active when the user connects to the server and authenticates, or when the user executes the `SET ROLE DEFAULT` statement during a session.

`ALTER USER ... DEFAULT ROLE` is alternative syntax for `SET DEFAULT ROLE` (see Section 13.7.1.9, “SET DEFAULT ROLE Syntax”). However, `ALTER USER` can set the default for only a single user, whereas `SET DEFAULT ROLE` can set the default for multiple users. On the other hand, you can specify `CURRENT_USER` as the user name for the `ALTER USER` statement, whereas you cannot for `SET DEFAULT ROLE`.

Each user account name uses the format described previously.

Each role name uses the format described in Section 6.2.5, “Specifying Role Names”. For example:

```
ALTER USER 'joe'@'10.0.0.1' DEFAULT ROLE administrator, developer;
```

The host name part of the role name, if omitted, defaults to `'%'`.

The clause following the `DEFAULT ROLE` keywords permits these values:

- `NONE`: Set the default to `NONE` (no roles).
- `ALL`: Set the default to all roles granted to the account.
- *`role [, role ] ...`*: Set the default to the named roles, which must exist and be granted to the account at the time `ALTER USER ... DEFAULT ROLE` is executed.

ALTER USER SSL/TLS Options

MySQL can check X.509 certificate attributes in addition to the usual authentication that is based on the user name and credentials. For background information on the use of SSL/TLS with MySQL, see Section 6.3, “Using Encrypted Connections”.

To specify SSL/TLS-related options for a MySQL account, use a `REQUIRE` clause that specifies one or more *`tls_option`* values.

Order of `REQUIRE` options does not matter, but no option can be specified twice. The `AND` keyword is optional between `REQUIRE` options.

`ALTER USER` permits these *`tls_option`* values:

- `NONE`

  Indicates that all accounts named by the statement have no SSL or X.509 requirements. Unencrypted connections are permitted if the user name and password are valid. Encrypted connections can be used, at the client's option, if the client has the proper certificate and key files.

```
ALTER USER 'jeffrey'@'localhost' REQUIRE NONE;
```

Clients attempt to establish a secure connection by default. For clients that have `REQUIRE NONE`, the connection attempt falls back to an unencrypted connection if a secure connection cannot be established. To require an encrypted connection, a client need specify only the `--ssl-mode=REQUIRED` option; the connection attempt fails if a secure connection cannot be established.

- `SSL`

  Tells the server to permit only encrypted connections for all accounts named by the statement.

  ```
  ALTER USER 'jeffrey'@'localhost' REQUIRE SSL;
  ```

  Clients attempt to establish a secure connection by default. For accounts that have `REQUIRE SSL`, the connection attempt fails if a secure connection cannot be established.

- `X509`

  For all accounts named by the statement, requires that clients present a valid certificate, but the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates. Use of X.509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

  ```
  ALTER USER 'jeffrey'@'localhost' REQUIRE X509;
  ```

  For accounts with `REQUIRE X509`, clients must specify the `--ssl-key` and `--ssl-cert` options to connect. (It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified.) This is true for `ISSUER` and `SUBJECT` as well because those `REQUIRE` options imply the requirements of `X509`.

- `ISSUER '`*`issuer`*`'`

  For all accounts named by the statement, requires that clients present a valid X.509 certificate issued by CA `'`*`issuer`*`'`. If a client presents a certificate that is valid but has a different issuer, the server rejects the connection. Use of X.509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

  ```
  ALTER USER 'jeffrey'@'localhost'
    REQUIRE ISSUER '/C=SE/ST=Stockholm/L=Stockholm/
      O=MySQL/CN=CA/emailAddress=ca@example.com';
  ```

  Because `ISSUER` implies the requirements of `X509`, clients must specify the `--ssl-key` and `--ssl-cert` options to connect. (It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified.)

- `SUBJECT '`*`subject`*`'`

  For all accounts named by the statement, requires that clients present a valid X.509 certificate containing the subject *`subject`*. If a client presents a certificate that is valid but has a different subject, the server rejects the connection. Use of X.509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

  ```
  ALTER USER 'jeffrey'@'localhost'
    REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/
      O=MySQL demo client certificate/
      CN=client/emailAddress=client@example.com';
  ```

MySQL does a simple string comparison of the *'subject'* value to the value in the certificate, so lettercase and component ordering must be given exactly as present in the certificate.

Because SUBJECT implies the requirements of X509, clients must specify the --ssl-key and --ssl-cert options to connect. (It is recommended but not required that --ssl-ca also be specified so that the public certificate provided by the server can be verified.)

- CIPHER *'cipher'*

  For all accounts named by the statement, requires a specific cipher method for encrypting connections. This option is needed to ensure that ciphers and key lengths of sufficient strength are used. Encryption can be weak if old algorithms using short encryption keys are used.

```
ALTER USER 'jeffrey'@'localhost'
  REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The SUBJECT, ISSUER, and CIPHER options can be combined in the REQUIRE clause:

```
ALTER USER 'jeffrey'@'localhost'
  REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/
    O=MySQL demo client certificate/
    CN=client/emailAddress=client@example.com'
  AND ISSUER '/C=SE/ST=Stockholm/L=Stockholm/
    O=MySQL/CN=CA/emailAddress=ca@example.com'
  AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

ALTER USER Resource-Limit Options

It is possible to place limits on use of server resources by an account, as discussed in Section 6.2.20, “Setting Account Resource Limits”. To do so, use a WITH clause that specifies one or more *resource_option* values.

Order of WITH options does not matter, except that if a given resource limit is specified multiple times, the last instance takes precedence.

ALTER USER permits these *resource_option* values:

- MAX_QUERIES_PER_HOUR *count*, MAX_UPDATES_PER_HOUR *count*, MAX_CONNECTIONS_PER_HOUR *count*

  For all accounts named by the statement, these options restrict how many queries, updates, and connections to the server are permitted to each account during any given one-hour period. If *count* is 0 (the default), this means that there is no limitation for the account.

- MAX_USER_CONNECTIONS *count*

  For all accounts named by the statement, restricts the maximum number of simultaneous connections to the server by each account. A nonzero *count* specifies the limit for the account explicitly. If *count* is 0 (the default), the server determines the number of simultaneous connections for the account from the global value of the max_user_connections system variable. If max_user_connections is also zero, there is no limit for the account.

Example:

```
ALTER USER 'jeffrey'@'localhost'
  WITH MAX_QUERIES_PER_HOUR 500 MAX_UPDATES_PER_HOUR 100;
```

ALTER USER Password-Management Options

`ALTER USER` supports several *`password_option`* values for password management:

- Password expiration options: You can expire an account password manually and establish its password expiration policy. Policy options do not expire the password. Instead, they determine how the server applies automatic expiration to the account based on password age, which is assessed from the date and time of the most recent account password change.
- Password reuse options: You can restrict password reuse based on number of password changes, time elapsed, or both.
- Password verification-required options: You can indicate whether attempts to change an account password must specify the current password, as verification that the user attempting to make the change actually knows the current password.

This section describes the syntax for password-management options. For information about establishing policy for password management, see Section 6.2.15, “Password Management”.

If multiple password-management options of a given type (`PASSWORD EXPIRE`, `PASSWORD HISTORY`, `PASSWORD REUSE INTERVAL`, `PASSWORD REQUIRE`) are specified, the last one takes precedence.

Note

Password-management options apply only to accounts that use an authentication plugin that stores credentials internally to MySQL. For accounts that use plugins that perform authentication against a credentials system that is external to MySQL, password management must be handled externally against that system as well. For more information about internal credentials storage, see Section 6.2.15, “Password Management”.

A client has an expired password if the account password was expired manually or the password age is considered greater than its permitted lifetime per the automatic expiration policy. In this case, the server either disconnects the client or restricts the operations permitted to it (see Section 6.2.16, “Server Handling of Expired Passwords”). Operations performed by a restricted client result in an error until the user establishes a new account password.

Note

Although it is possible to “reset” an expired password by setting it to its current value, it is preferable, as a matter of good policy, to choose a different password. DBAs can enforce non-reuse by establishing an appropriate password-reuse policy. See Password Reuse Policy.

`ALTER USER` permits these *`password_option`* values for controlling password expiration:

- `PASSWORD EXPIRE`

  Immediately marks the password expired for all accounts named by the statement.

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE;
```

- `PASSWORD EXPIRE DEFAULT`

  Sets all accounts named by the statement so that the global expiration policy applies, as specified by the `default_password_lifetime` system variable.

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
```

- `PASSWORD EXPIRE NEVER`

  This expiration option overrides the global policy for all accounts named by the statement. For each, it disables password expiration so that the password never expires.

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
```

- `PASSWORD EXPIRE INTERVAL N DAY`

  This expiration option overrides the global policy for all accounts named by the statement. For each, it sets the password lifetime to *`N`* days. The following statement requires the password to be changed every 180 days:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 180 DAY;
```

`ALTER USER` permits these *`password_option`* values for controlling reuse of previous passwords based on required minimum number of password changes:

- `PASSWORD HISTORY DEFAULT`

  Sets all accounts named by the statement so that the global policy about password history length applies, to prohibit reuse of passwords before the number of changes specified by the `password_history` system variable.

```
ALTER USER 'jeffrey'@'localhost' PASSWORD HISTORY DEFAULT;
```

- `PASSWORD HISTORY N`

  This history-length option overrides the global policy for all accounts named by the statement. For each, it sets the password history length to *`N`* passwords, to prohibit reusing any of the *`N`* most recently chosen passwords. The following statement prohibits reuse of any of the previous 6 passwords:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD HISTORY 6;
```

`ALTER USER` permits these *`password_option`* values for controlling reuse of previous passwords based on time elapsed:

- `PASSWORD REUSE INTERVAL DEFAULT`

  Sets all statements named by the account so that the global policy about time elapsed applies, to prohibit reuse of passwords newer than the number of days specified by the `password_reuse_interval` system variable.

```
ALTER USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL DEFAULT;
```

- `PASSWORD REUSE INTERVAL N DAY`

  This time-elapsed option overrides the global policy for all accounts named by the statement. For each, it sets the password reuse interval to *`N`* days, to prohibit reuse of passwords newer than that many days. The following statement prohibits password reuse for 360 days:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL 360 DAY;
```

`ALTER USER` permits these *`password_option`* values for controlling whether attempts to change an account password must specify the current password, as verification that the user attempting to make the change actually knows the current password:

- `PASSWORD REQUIRE CURRENT`

  This verification option overrides the global policy for all accounts named by the statement. For each, it requires that password changes specify the current password.

```
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT;
```

- `PASSWORD REQUIRE CURRENT OPTIONAL`

  This verification option overrides the global policy for all accounts named by the statement. For each, it does not require that password changes specify the current password. (The current password may but need not be given.)

```
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT OPTIONAL;
```

- `PASSWORD REQUIRE CURRENT DEFAULT`

  Sets all statements named by the account so that the global policy about password verification applies, as specified by the `password_require_current` system variable.

```
ALTER USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT DEFAULT;
```

ALTER USER Account-Locking Options

MySQL supports account locking and unlocking using the `ACCOUNT LOCK` and `ACCOUNT UNLOCK` options, which specify the locking state for an account. For additional discussion, see Section 6.2.19, “Account Locking”.

If multiple account-locking options are specified, the last one takes precedence.

ALTER USER Binary Logging

`ALTER USER` is written to the binary log if it succeeds, but not if it fails; in that case, rollback occurs and no changes are made. A statement written to the binary log includes all named users. If the `IF EXISTS` clause is given, this includes even users that do not exist and were not altered.

If the original statement changes the credentials for a user, the statement written to the binary log specifies the applicable authentication plugin for that user, determined as follows:

- The plugin named in the original statement, if one was specified.
- Otherwise, the plugin associated with the user account if the user exists, or the default authentication plugin if the user does not exist. (If the statement written to the binary log must specify a particular authentication plugin for a user, include it in the original statement.)

If the server adds the default authentication plugin for any users in the statement written to the binary log, it writes a warning to the error log naming those users.

13.7.1.2 CREATE ROLE Syntax

```
CREATE ROLE [IF NOT EXISTS] role [, role ] ...
```

`CREATE ROLE` creates one or more roles, which are named collections of privileges. To use this statement, you must have the global `CREATE ROLE` or `CREATE USER` privilege. When the `read_only` system variable is enabled, `CREATE ROLE` additionally requires the `CONNECTION_ADMIN` or `SUPER` privilege.

A role when created is locked, has no password, and is assigned the default authentication plugin. (These role attributes can be changed later with the `ALTER USER` statement, by users who have the global `CREATE USER` privilege.)

`CREATE ROLE` either succeeds for all named roles or rolls back and has no effect if any error occurs. By default, an error occurs if you try to create a role that already exists. If the `IF NOT EXISTS` clause is given, the statement produces a warning for each named role that already exists, rather than an error.

The statement is written to the binary log if it succeeds, but not if it fails; in that case, rollback occurs and no changes are made. A statement written to the binary log includes all named roles. If the `IF NOT EXISTS` clause is given, this includes even roles that already exist and were not created.

Each role name uses the format described in Section 6.2.5, “Specifying Role Names”. For example:

```
CREATE ROLE 'administrator', 'developer';
CREATE ROLE 'webapp'@'localhost';
```

The host name part of the role name, if omitted, defaults to `'%'`.

For role usage examples, see Section 6.2.10, “Using Roles”.

13.7.1.3 CREATE USER Syntax

```
CREATE USER [IF NOT EXISTS]
    user [auth_option] [, user [auth_option]] ...
    DEFAULT ROLE role [, role ] ...
    [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
    [WITH resource_option [resource_option] ...]
    [password_option | lock_option] ...

user:
    (see Section 6.2.4, “Specifying Account Names”)

auth_option: {
    IDENTIFIED BY 'auth_string'
  | IDENTIFIED BY RANDOM PASSWORD
  | IDENTIFIED WITH auth_plugin
  | IDENTIFIED WITH auth_plugin BY 'auth_string'
  | IDENTIFIED WITH auth_plugin BY RANDOM PASSWORD
  | IDENTIFIED WITH auth_plugin AS 'auth_string'
}

tls_option: {
   SSL
 | X509
 | CIPHER 'cipher'
 | ISSUER 'issuer'
 | SUBJECT 'subject'
}

resource_option: {
    MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
}

password_option: {
```

```
    PASSWORD EXPIRE [DEFAULT | NEVER | INTERVAL N DAY]
  | PASSWORD HISTORY {DEFAULT | N}
  | PASSWORD REUSE INTERVAL {DEFAULT | N DAY}
  | PASSWORD REQUIRE CURRENT [DEFAULT | OPTIONAL]
}

lock_option: {
    ACCOUNT LOCK
  | ACCOUNT UNLOCK
}
```

The `CREATE USER` statement creates new MySQL accounts. It enables authentication, role, SSL/TLS, resource-limit, and password-management properties to be established for new accounts. It also controls whether accounts are initially locked or unlocked.

To use `CREATE USER`, you must have the global `CREATE USER` privilege, or the `INSERT` privilege for the `mysql` system database. When the `read_only` system variable is enabled, `CREATE USER` additionally requires the `CONNECTION_ADMIN` or `SUPER` privilege.

`CREATE USER` either succeeds for all named users or rolls back and has no effect if any error occurs. By default, an error occurs if you try to create a user that already exists. If the `IF NOT EXISTS` clause is given, the statement produces a warning for each named user that already exists, rather than an error.

Important

Under some circumstances, `CREATE USER` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about the conditions under which this occurs for the server logs and how to control it, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, see Section 4.5.1.3, “mysql Client Logging”.

There are several aspects to the `CREATE USER` statement, described under the following topics:

- CREATE USER Overview
- CREATE USER Authentication Options
- CREATE USER Role Options
- CREATE USER SSL/TLS Options
- CREATE USER Resource-Limit Options
- CREATE USER Password-Management Options
- CREATE USER Account-Locking Options
- CREATE USER Binary Logging

CREATE USER Overview

For each account, `CREATE USER` creates a new row in the `mysql.user` system table. The account row reflects the properties specified in the statement. Unspecified properties are set to their default values:

- Authentication: The authentication plugin defined by the `default_authentication_plugin` system variable, and empty credentials
- Default role: `NONE`
- SSL/TLS: `NONE`

- Resource limits: Unlimited
- Password management: `PASSWORD EXPIRE DEFAULT PASSWORD HISTORY DEFAULT PASSWORD REUSE INTERVAL DEFAULT PASSWORD REQUIRE CURRENT DEFAULT`
- Account locking: `ACCOUNT UNLOCK`

An account when first created has no privileges and a default role of `NONE`. To assign privileges or roles, use the `GRANT` statement.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. For example:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

The host name part of the account name, if omitted, defaults to `'%'`.

Each *`user`* value naming an account may be followed by an optional *`auth_option`* value that indicates how the account authenticates. These values enable account authentication plugins and credentials (for example, a password) to be specified. Each *`auth_option`* value applies *only* to the account named immediately preceding it.

Following the *`user`* specifications, the statement may include options for SSL/TLS, resource-limit, password-management, and locking properties. All such options are *global* to the statement and apply to *all* accounts named in the statement.

Example: Create an account that uses the default authentication plugin and the given password. Mark the password expired so that the user must choose a new one at the first connection to the server:

```
CREATE USER 'jeffrey'@'localhost'
  IDENTIFIED BY 'new_password' PASSWORD EXPIRE;
```

Example: Create an account that uses the `sha256_password` authentication plugin and the given password. Require that a new password be chosen every 180 days:

```
CREATE USER 'jeffrey'@'localhost'
  IDENTIFIED WITH sha256_password BY 'new_password'
  PASSWORD EXPIRE INTERVAL 180 DAY;
```

Example: Create multiple accounts, specifying some per-account properties and some global properties:

```
CREATE USER
  'jeffrey'@'localhost' IDENTIFIED WITH mysql_native_password
                                    BY 'new_password1',
  'jeanne'@'localhost' IDENTIFIED WITH sha256_password
                                   BY 'new_password2'
  REQUIRE X509 WITH MAX_QUERIES_PER_HOUR 60
  PASSWORD HISTORY 5
  ACCOUNT LOCK;
```

Each *`auth_option`* value (`IDENTIFIED WITH ... BY` in this case) applies only to the account named immediately preceding it, so each account uses the immediately following authentication plugin and password.

The remaining properties apply globally to all accounts named in the statement, so for both accounts:

- Connections must be made using a valid X.509 certificate.
- Up to 60 queries per hour are permitted.

- Password changes cannot reuse any of the five most recent passwords.
- The account is locked initially, so effectively it is a placeholder and cannot be used until an administrator unlocks it.

CREATE USER Authentication Options

An account name may be followed by an *`auth_option`* authentication option that specifies the account authentication plugin, credentials, or both.

Note

Clauses for random password generation apply only to accounts that use an authentication plugin that stores credentials internally to MySQL. For accounts that use plugins that perform authentication against a credentials system that is external to MySQL, password management must be handled externally against that system as well. For more information about internal credentials storage, see Section 6.2.15, “Password Management”.

- *`auth_plugin`* names an authentication plugin. The plugin name can be a quoted string literal or an unquoted name. Plugin names are stored in the `plugin` column of the `mysql.user` system table.

  For *`auth_option`* syntax that does not specify an authentication plugin, the default plugin is indicated by the value of the `default_authentication_plugin` system variable. For descriptions of each plugin, see Section 6.4.1, “Authentication Plugins”.

- Credentials that are stored internally are stored in the `mysql.user` system table. An *`'auth_string'`* value or `RANDOM PASSWORD` specifies account credentials, either as a cleartext (unencrypted) string or hashed in the format expected by the authentication plugin associated with the account, respectively:

  - For syntax that uses `BY` *`'auth_string'`*, the string is cleartext and is passed to the authentication plugin for possible hashing. The result returned by the plugin is stored in the `mysql.user` table. A plugin may use the value as specified, in which case no hashing occurs.

  - For syntax that uses `BY RANDOM PASSWORD`, MySQL generates a random password and as cleartext and passes it to the authentication plugin for possible hashing. The result returned by the plugin is stored in the `mysql.user` table. A plugin may use the value as specified, in which case no hashing occurs.

    Randomly generated passwords are available as of MySQL 8.0.18 and have the characteristics described in Random Password Generation.

  - For syntax that uses `AS` *`'auth_string'`*, the string is assumed to be already in the format the authentication plugin requires, and is stored as is in the `mysql.user` table. If a plugin requires a hashed value, the value must be already hashed in a format appropriate for the plugin, or the value will not be usable by the plugin and correct authentication of client connections will not occur.

    As of MySQL 8.0.17, a hashed string can be either a string literal or a hexadecimal value. The latter corresponds to the type of value displayed by `SHOW CREATE USER` for password hashes containing unprintable characters when the `print_identified_with_as_hex` system variable is enabled.

  - If an authentication plugin performs no hashing of the authentication string, the `BY` *`'auth_string'`* and `AS` *`'auth_string'`* clauses have the same effect: The authentication string is stored as is in the `mysql.user` system table.

`CREATE USER` permits these *`auth_option`* syntaxes:

- `IDENTIFIED BY 'auth_string'`

  Sets the account authentication plugin to the default plugin, passes the cleartext `'auth_string'` value to the plugin for possible hashing, and stores the result in the account row in the `mysql.user` system table.

- `IDENTIFIED BY RANDOM PASSWORD`

  Sets the account authentication plugin to the default plugin, generates a random password, passes the cleartext password value to the plugin for possible hashing, and stores the result in the account row in the `mysql.user` system table. The statement also returns the cleartext password in a result set to make it available to the user or application executing the statement. For details about the result set and characteristics of randomly generated passwords, see Random Password Generation.

- `IDENTIFIED WITH auth_plugin`

  Sets the account authentication plugin to `auth_plugin`, clears the credentials to the empty string, and stores the result in the account row in the `mysql.user` system table.

- `IDENTIFIED WITH auth_plugin BY 'auth_string'`

  Sets the account authentication plugin to `auth_plugin`, passes the cleartext `'auth_string'` value to the plugin for possible hashing, and stores the result in the account row in the `mysql.user` system table.

- `IDENTIFIED WITH auth_plugin BY RANDOM PASSWORD`

  Sets the account authentication plugin to `auth_plugin`, generates a random password, passes the cleartext password value to the plugin for possible hashing, and stores the result in the account row in the `mysql.user` system table. The statement also returns the cleartext password in a result set to make it available to the user or application executing the statement. For details about the result set and characteristics of randomly generated passwords, see Random Password Generation.

- `IDENTIFIED WITH auth_plugin AS 'auth_string'`

  Sets the account authentication plugin to `auth_plugin` and stores the `'auth_string'` value as is in the `mysql.user` account row. If the plugin requires a hashed string, the string is assumed to be already hashed in the format the plugin requires.

Example: Specify the password as cleartext; the default plugin is used:

```
CREATE USER 'jeffrey'@'localhost'
  IDENTIFIED BY 'password';
```

Example: Specify the authentication plugin, along with a cleartext password value:

```
CREATE USER 'jeffrey'@'localhost'
  IDENTIFIED WITH mysql_native_password BY 'password';
```

In each case, the password value stored in the account row is the cleartext value `'password'` after it has been hashed by the authentication plugin associated with the account.

For additional information about setting passwords and authentication plugins, see Section 6.2.14, “Assigning Account Passwords”, and Section 6.2.17, “Pluggable Authentication”.

CREATE USER Role Options

The DEFAULT ROLE clause defines which roles become active when the user connects to the server and authenticates, or when the user executes the SET ROLE DEFAULT statement during a session.

Each role name uses the format described in Section 6.2.5, “Specifying Role Names”. For example:

```
CREATE USER 'joe'@'10.0.0.1' DEFAULT ROLE administrator, developer;
```

The host name part of the role name, if omitted, defaults to '%'.

The DEFAULT ROLE clause permits a list of one or more comma-separated role names. These roles need not exist at the time CREATE USER is executed.

CREATE USER SSL/TLS Options

MySQL can check X.509 certificate attributes in addition to the usual authentication that is based on the user name and credentials. For background information on the use of SSL/TLS with MySQL, see Section 6.3, “Using Encrypted Connections”.

To specify SSL/TLS-related options for a MySQL account, use a REQUIRE clause that specifies one or more *tls_option* values.

Order of REQUIRE options does not matter, but no option can be specified twice. The AND keyword is optional between REQUIRE options.

CREATE USER permits these *tls_option* values:

- NONE

  Indicates that all accounts named by the statement have no SSL or X.509 requirements. Unencrypted connections are permitted if the user name and password are valid. Encrypted connections can be used, at the client's option, if the client has the proper certificate and key files.

  ```
  CREATE USER 'jeffrey'@'localhost' REQUIRE NONE;
  ```

  Clients attempt to establish a secure connection by default. For clients that have REQUIRE NONE, the connection attempt falls back to an unencrypted connection if a secure connection cannot be established. To require an encrypted connection, a client need specify only the --ssl-mode=REQUIRED option; the connection attempt fails if a secure connection cannot be established.

  NONE is the default if no SSL-related REQUIRE options are specified.

- SSL

  Tells the server to permit only encrypted connections for all accounts named by the statement.

  ```
  CREATE USER 'jeffrey'@'localhost' REQUIRE SSL;
  ```

  Clients attempt to establish a secure connection by default. For accounts that have REQUIRE SSL, the connection attempt fails if a secure connection cannot be established.

- X509

  For all accounts named by the statement, requires that clients present a valid certificate, but the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates. Use of X.509 certificates always implies encryption, so the SSL option is unnecessary in this case.

```
CREATE USER 'jeffrey'@'localhost' REQUIRE X509;
```

For accounts with `REQUIRE X509`, clients must specify the `--ssl-key` and `--ssl-cert` options to connect. (It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified.) This is true for `ISSUER` and `SUBJECT` as well because those `REQUIRE` options imply the requirements of `X509`.

- `ISSUER 'issuer'`

  For all accounts named by the statement, requires that clients present a valid X.509 certificate issued by CA `'issuer'`. If a client presents a certificate that is valid but has a different issuer, the server rejects the connection. Use of X.509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

  ```
  CREATE USER 'jeffrey'@'localhost'
    REQUIRE ISSUER '/C=SE/ST=Stockholm/L=Stockholm/
      O=MySQL/CN=CA/emailAddress=ca@example.com';
  ```

  Because `ISSUER` implies the requirements of `X509`, clients must specify the `--ssl-key` and `--ssl-cert` options to connect. (It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified.)

- `SUBJECT 'subject'`

  For all accounts named by the statement, requires that clients present a valid X.509 certificate containing the subject `subject`. If a client presents a certificate that is valid but has a different subject, the server rejects the connection. Use of X.509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

  ```
  CREATE USER 'jeffrey'@'localhost'
    REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/
      O=MySQL demo client certificate/
      CN=client/emailAddress=client@example.com';
  ```

  MySQL does a simple string comparison of the `'subject'` value to the value in the certificate, so lettercase and component ordering must be given exactly as present in the certificate.

  Because `SUBJECT` implies the requirements of `X509`, clients must specify the `--ssl-key` and `--ssl-cert` options to connect. (It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified.)

- `CIPHER 'cipher'`

  For all accounts named by the statement, requires a specific cipher method for encrypting connections. This option is needed to ensure that ciphers and key lengths of sufficient strength are used. Encryption can be weak if old algorithms using short encryption keys are used.

  ```
  CREATE USER 'jeffrey'@'localhost'
    REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
  ```

The `SUBJECT`, `ISSUER`, and `CIPHER` options can be combined in the `REQUIRE` clause:

```
CREATE USER 'jeffrey'@'localhost'
  REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/
```

```
    O=MySQL demo client certificate/
    CN=client/emailAddress=client@example.com'
  AND ISSUER '/C=SE/ST=Stockholm/L=Stockholm/
    O=MySQL/CN=CA/emailAddress=ca@example.com'
  AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

CREATE USER Resource-Limit Options

It is possible to place limits on use of server resources by an account, as discussed in Section 6.2.20, “Setting Account Resource Limits”. To do so, use a `WITH` clause that specifies one or more *`resource_option`* values.

Order of `WITH` options does not matter, except that if a given resource limit is specified multiple times, the last instance takes precedence.

`CREATE USER` permits these *`resource_option`* values:

- `MAX_QUERIES_PER_HOUR` *`count`*, `MAX_UPDATES_PER_HOUR` *`count`*, `MAX_CONNECTIONS_PER_HOUR` *`count`*

  For all accounts named by the statement, these options restrict how many queries, updates, and connections to the server are permitted to each account during any given one-hour period. If *`count`* is `0` (the default), this means that there is no limitation for the account.

- `MAX_USER_CONNECTIONS` *`count`*

  For all accounts named by the statement, restricts the maximum number of simultaneous connections to the server by each account. A nonzero *`count`* specifies the limit for the account explicitly. If *`count`* is `0` (the default), the server determines the number of simultaneous connections for the account from the global value of the `max_user_connections` system variable. If `max_user_connections` is also zero, there is no limit for the account.

Example:

```
CREATE USER 'jeffrey'@'localhost'
  WITH MAX_QUERIES_PER_HOUR 500 MAX_UPDATES_PER_HOUR 100;
```

CREATE USER Password-Management Options

`CREATE USER` supports several *`password_option`* values for password management:

- Password expiration options: You can expire an account password manually and establish its password expiration policy. Policy options do not expire the password. Instead, they determine how the server applies automatic expiration to the account based on password age, which is assessed from the date and time of the most recent account password change.

- Password reuse options: You can restrict password reuse based on number of password changes, time elapsed, or both.

- Password verification-required options: You can indicate whether attempts to change an account password must specify the current password, as verification that the user attempting to make the change actually knows the current password.

This section describes the syntax for password-management options. For information about establishing policy for password management, see Section 6.2.15, “Password Management”.

If multiple password-management options of a given type (`PASSWORD EXPIRE`, `PASSWORD HISTORY`, `PASSWORD REUSE INTERVAL`, `PASSWORD REQUIRE`) are specified, the last one takes precedence.

Note

Password-management options apply only to accounts that use an authentication plugin that stores credentials internally to MySQL. For accounts that use plugins that perform authentication against a credentials system that is external to MySQL, password management must be handled externally against that system as well. For more information about internal credentials storage, see Section 6.2.15, “Password Management”.

A client has an expired password if the account password was expired manually or the password age is considered greater than its permitted lifetime per the automatic expiration policy. In this case, the server either disconnects the client or restricts the operations permitted to it (see Section 6.2.16, “Server Handling of Expired Passwords”). Operations performed by a restricted client result in an error until the user establishes a new account password.

`CREATE USER` permits these *`password_option`* values for controlling password expiration:

- `PASSWORD EXPIRE`

  Immediately marks the password expired for all accounts named by the statement.

  ```
  CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE;
  ```

- `PASSWORD EXPIRE DEFAULT`

  Sets all accounts named by the statement so that the global expiration policy applies, as specified by the `default_password_lifetime` system variable.

  ```
  CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE DEFAULT;
  ```

- `PASSWORD EXPIRE NEVER`

  This expiration option overrides the global policy for all accounts named by the statement. For each, it disables password expiration so that the password never expires.

  ```
  CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE NEVER;
  ```

- `PASSWORD EXPIRE INTERVAL` *`N`* `DAY`

  This expiration option overrides the global policy for all accounts named by the statement. For each, it sets the password lifetime to *`N`* days. The following statement requires the password to be changed every 180 days:

  ```
  CREATE USER 'jeffrey'@'localhost' PASSWORD EXPIRE INTERVAL 180 DAY;
  ```

`CREATE USER` permits these *`password_option`* values for controlling reuse of previous passwords based on required minimum number of password changes:

- `PASSWORD HISTORY DEFAULT`

  Sets all accounts named by the statement so that the global policy about password history length applies, to prohibit reuse of passwords before the number of changes specified by the `password_history` system variable.

  ```
  CREATE USER 'jeffrey'@'localhost' PASSWORD HISTORY DEFAULT;
  ```

- `PASSWORD HISTORY N`

  This history-length option overrides the global policy for all accounts named by the statement. For each, it sets the password history length to *`N`* passwords, to prohibit reusing any of the *`N`* most recently chosen passwords. The following statement prohibits reuse of any of the previous 6 passwords:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD HISTORY 6;
```

`CREATE USER` permits these *`password_option`* values for controlling reuse of previous passwords based on time elapsed:

- `PASSWORD REUSE INTERVAL DEFAULT`

  Sets all statements named by the account so that the global policy about time elapsed applies, to prohibit reuse of passwords newer than the number of days specified by the `password_reuse_interval` system variable.

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL DEFAULT;
```

- `PASSWORD REUSE INTERVAL N DAY`

  This time-elapsed option overrides the global policy for all accounts named by the statement. For each, it sets the password reuse interval to *`N`* days, to prohibit reuse of passwords newer than that many days. The following statement prohibits password reuse for 360 days:

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REUSE INTERVAL 360 DAY;
```

`CREATE USER` permits these *`password_option`* values for controlling whether attempts to change an account password must specify the current password, as verification that the user attempting to make the change actually knows the current password:

- `PASSWORD REQUIRE CURRENT`

  This verification option overrides the global policy for all accounts named by the statement. For each, it requires that password changes specify the current password.

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT;
```

- `PASSWORD REQUIRE CURRENT OPTIONAL`

  This verification option overrides the global policy for all accounts named by the statement. For each, it does not require that password changes specify the current password. (The current password may but need not be given.)

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT OPTIONAL;
```

- `PASSWORD REQUIRE CURRENT DEFAULT`

  Sets all statements named by the account so that the global policy about password verification applies, as specified by the `password_require_current` system variable.

```
CREATE USER 'jeffrey'@'localhost' PASSWORD REQUIRE CURRENT DEFAULT;
```

CREATE USER Account-Locking Options

MySQL supports account locking and unlocking using the `ACCOUNT LOCK` and `ACCOUNT UNLOCK` options, which specify the locking state for an account. For additional discussion, see Section 6.2.19, “Account Locking”.

If multiple account-locking options are specified, the last one takes precedence.

CREATE USER Binary Logging

`CREATE USER` is written to the binary log if it succeeds, but not if it fails; in that case, rollback occurs and no changes are made. A statement written to the binary log includes all named users. If the `IF NOT EXISTS` clause is given, this includes even users that already exist and were not created.

The statement written to the binary log specifies an authentication plugin for each user, determined as follows:

- The plugin named in the original statement, if one was specified.
- Otherwise, the default authentication plugin. In particular, if a user `u1` already exists and uses a nondefault authentication plugin, the statement written to the binary log for `CREATE USER IF NOT EXISTS u1` names the default authentication plugin. (If the statement written to the binary log must specify a nondefault authentication plugin for a user, include it in the original statement.)

If the server adds the default authentication plugin for any nonexisting users in the statement written to the binary log, it writes a warning to the error log naming those users.

13.7.1.4 DROP ROLE Syntax

```
DROP ROLE [IF EXISTS] role [, role ] ...
```

`DROP ROLE` removes one or more roles (named collections of privileges). To use this statement, you must have the global `DROP ROLE` or `CREATE USER` privilege. When the `read_only` system variable is enabled, `DROP ROLE` additionally requires the `CONNECTION_ADMIN` or `SUPER` privilege.

As of MySQL 8.0.16, users who have the `CREATE USER` privilege can use this statement to drop accounts that are locked or unlocked. Users who have the `DROP ROLE` privilege can use this statement only to drop accounts that are locked (unlocked accounts are presumably user accounts used to log in to the server and not just as roles).

Roles named in the `mandatory_roles` system variable value cannot be dropped.

`DROP ROLE` either succeeds for all named roles or rolls back and has no effect if any error occurs. By default, an error occurs if you try to drop a role that does not exist. If the `IF EXISTS` clause is given, the statement produces a warning for each named role that does not exist, rather than an error.

The statement is written to the binary log if it succeeds, but not if it fails; in that case, rollback occurs and no changes are made. A statement written to the binary log includes all named roles. If the `IF EXISTS` clause is given, this includes even roles that do not exist and were not dropped.

Each role name uses the format described in Section 6.2.5, “Specifying Role Names”. For example:

```
DROP ROLE 'administrator', 'developer';
DROP ROLE 'webapp'@'localhost';
```

The host name part of the role name, if omitted, defaults to `'%'`.

A dropped role is automatically revoked from any user account (or role) to which the role was granted. Within any current session for such an account, its adjusted privileges apply beginning with the next statement executed.

For role usage examples, see Section 6.2.10, “Using Roles”.

13.7.1.5 DROP USER Syntax

```
DROP USER [IF EXISTS] user [, user] ...
```

The `DROP USER` statement removes one or more MySQL accounts and their privileges. It removes privilege rows for the account from all grant tables.

Roles named in the `mandatory_roles` system variable value cannot be dropped.

To use `DROP USER`, you must have the global `CREATE USER` privilege, or the `DELETE` privilege for the `mysql` system database. When the `read_only` system variable is enabled, `DROP USER` additionally requires the `CONNECTION_ADMIN` or `SUPER` privilege.

`DROP USER` either succeeds for all named users or rolls back and has no effect if any error occurs. By default, an error occurs if you try to drop a user that does not exist. If the `IF EXISTS` clause is given, the statement produces a warning for each named user that does not exist, rather than an error.

The statement is written to the binary log if it succeeds, but not if it fails; in that case, rollback occurs and no changes are made. A statement written to the binary log includes all named users. If the `IF EXISTS` clause is given, this includes even users that do not exist and were not dropped.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. For example:

```
DROP USER 'jeffrey'@'localhost';
```

The host name part of the account name, if omitted, defaults to `'%'`.

Important

`DROP USER` does not automatically close any open user sessions. Rather, in the event that a user with an open session is dropped, the statement does not take effect until that user's session is closed. Once the session is closed, the user is dropped, and that user's next attempt to log in will fail. *This is by design*.

`DROP USER` does not automatically drop or invalidate databases or objects within them that the old user created. This includes stored programs or views for which the `DEFINER` attribute names the dropped user. Attempts to access such objects may produce an error if they execute in definer security context. (For information about security context, see Section 24.6, “Stored Object Access Control”.)

13.7.1.6 GRANT Syntax

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user_or_role [, user_or_role] ...
    [WITH GRANT OPTION]
    [AS user
        [WITH ROLE
            DEFAULT
          | NONE
          | ALL
          | ALL EXCEPT role [, role ] ...
          | role [, role ] ...
        ]
    ]
}
```

```
GRANT PROXY ON user_or_role
    TO user_or_role [, user_or_role] ...
    [WITH GRANT OPTION]

GRANT role [, role] ...
    TO user_or_role [, user_or_role] ...
    [WITH ADMIN OPTION]

object_type: {
    TABLE
  | FUNCTION
  | PROCEDURE
}

priv_level: {
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.routine_name
}

user_or_role: {
    user (see Section 6.2.4, “Specifying Account Names”)
  | role (see Section 6.2.5, “Specifying Role Names”)
}
```

The GRANT statement assigns privileges and roles to MySQL user accounts and roles. There are several aspects to the GRANT statement, described under the following topics:

- GRANT General Overview
- Object Quoting Guidelines
- Account Names
- Privileges Supported by MySQL
- Global Privileges
- Database Privileges
- Table Privileges
- Column Privileges
- Stored Routine Privileges
- Proxy User Privileges
- Granting Roles
- The AS Clause and Privilege Restrictions
- Other Account Characteristics
- MySQL and Standard SQL Versions of GRANT

GRANT General Overview

The GRANT statement enables system administrators to grant privileges and roles, which can be granted to user accounts and roles. These syntax restrictions apply:

- `GRANT` cannot mix granting both privileges and roles in the same statement. A given `GRANT` statement must grant either privileges or roles.
- The `ON` clause distinguishes whether the statement grants privileges or roles:
  - With `ON`, the statement grants privileges.
  - Without `ON`, the statement grants roles.
  - It is permitted to assign both privileges and roles to an account, but you must use separate `GRANT` statements, each with syntax appropriate to what is to be granted.

For more information about roles, see Section 6.2.10, “Using Roles”.

To use `GRANT`, you must have the `GRANT OPTION` privilege, and you must have the privileges that you are granting. When the `read_only` system variable is enabled, `GRANT` additionally requires the `CONNECTION_ADMIN` or `SUPER` privilege.

`GRANT` either succeeds for all named users and roles or rolls back and has no effect if any error occurs. The statement is written to the binary log only if it succeeds for all named users and roles.

The `REVOKE` statement is related to `GRANT` and enables administrators to remove account privileges. See Section 13.7.1.8, “REVOKE Syntax”.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. Each role name uses the format described in Section 6.2.5, “Specifying Role Names”. For example:

```
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
GRANT 'role1', 'role2' TO 'user1'@'localhost', 'user2'@'localhost';
GRANT SELECT ON world.* TO 'role3';
```

The host name part of the account or role name, if omitted, defaults to `'%'`.

Normally, a database administrator first uses `CREATE USER` to create an account and define its nonprivilege characteristics such as its password, whether it uses secure connections, and limits on access to server resources, then uses `GRANT` to define its privileges. `ALTER USER` may be used to change the nonprivilege characteristics of existing accounts. For example:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
GRANT SELECT ON db2.invoice TO 'jeffrey'@'localhost';
ALTER USER 'jeffrey'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;
```

From the `mysql` program, `GRANT` responds with `Query OK, 0 rows affected` when executed successfully. To determine what privileges result from the operation, use `SHOW GRANTS`. See Section 13.7.7.21, “SHOW GRANTS Syntax”.

Important

Under some circumstances, `GRANT` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about the conditions under which this occurs for the server logs and how to control it, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, see Section 4.5.1.3, “mysql Client Logging”.

`GRANT` supports host names up to 255 characters long (60 characters prior to MySQL 8.0.17). User names can be up to 32 characters. Database, table, column, and routine names can be up to 64 characters.

Warning

Do not attempt to change the permissible length for user names by altering the `mysql.user` system table. Doing so results in unpredictable behavior which may even make it impossible for users to log in to the MySQL server. Never alter the structure of tables in the `mysql` system database in any manner except by means of the procedure described in Section 2.11, “Upgrading MySQL”.

Object Quoting Guidelines

Several objects within `GRANT` statements are subject to quoting, although quoting is optional in many cases: Account, role, database, table, column, and routine names. For example, if a *`user_name`* or *`host_name`* value in an account name is legal as an unquoted identifier, you need not quote it. However, quotation marks are necessary to specify a *`user_name`* string containing special characters (such as `-`), or a *`host_name`* string containing special characters or wildcard characters such as `%` (for example, `'test-user'@'%.com'`). Quote the user name and host name separately.

To specify quoted values:

- Quote database, table, column, and routine names as identifiers.
- Quote user names and host names as identifiers or as strings.
- Quote passwords as strings.

For string-quoting and identifier-quoting guidelines, see Section 9.1.1, “String Literals”, and Section 9.2, “Schema Object Names”.

The `_` and `%` wildcards are permitted when specifying database names in `GRANT` statements that grant privileges at the database level (`GRANT ... ON db_name.*`). This means, for example, that to use a `_` character as part of a database name, specify it as `\_` in the `GRANT` statement, to prevent the user from being able to access additional databases matching the wildcard pattern (for example, `GRANT ... ON 'foo\_bar'.* TO ...`).

When a database name not is used to grant privileges at the database level, but as a qualifier for granting privileges to some other object such as a table or routine (for example, `GRANT ... ON db_name.tbl_name`), wildcard characters are treated as normal characters.

Account Names

A *`user`* value in a `GRANT` statement indicates a MySQL account to which the statement applies. To accommodate granting rights to users from arbitrary hosts, MySQL supports specifying the *`user`* value in the form `'user_name'@'host_name'`.

You can specify wildcards in the host name. For example, `'user_name'@'%.example.com'` applies to *`user_name`* for any host in the `example.com` domain, and `'user_name'@'198.51.100.%'` applies to *`user_name`* for any host in the `198.51.100` class C subnet.

The simple form `'user_name'` is a synonym for `'user_name'@'%'`.

MySQL does not support wildcards in user names. To refer to an anonymous user, specify an account with an empty user name with the `GRANT` statement:

```
GRANT ALL ON test.* TO ''@'localhost' ...;
```

In this case, any user who connects from the local host with the correct password for the anonymous user will be permitted access, with the privileges associated with the anonymous-user account.

For additional information about user name and host name values in account names, see Section 6.2.4, “Specifying Account Names”.

Warning

If you permit local anonymous users to connect to the MySQL server, you should also grant privileges to all local users as `'user_name'@'localhost'`. Otherwise, the anonymous user account for `localhost` in the `mysql.user` system table is used when named users try to log in to the MySQL server from the local machine. For details, see Section 6.2.6, “Access Control, Stage 1: Connection Verification”.

To determine whether this issue applies to you, execute the following query, which lists any anonymous users:

```
SELECT Host, User FROM mysql.user WHERE User='';
```

To avoid the problem just described, delete the local anonymous user account using this statement:

```
DROP USER ''@'localhost';
```

Privileges Supported by MySQL

The following tables summarize the permissible static and dynamic *`priv_type`* privilege types that can be specified for the `GRANT` and `REVOKE` statements, and the levels at which each privilege can be granted. For additional information about each privilege, see Section 6.2.2, “Privileges Provided by MySQL”. For information about the differences between static and dynamic privileges, see Static Versus Dynamic Privileges.

Table 13.11 Permissible Static Privileges for GRANT and REVOKE

Privilege	Meaning and Grantable Levels
`ALL [PRIVILEGES]`	Grant all privileges at specified access level except `GRANT OPTION` and `PROXY`.
`ALTER`	Enable use of `ALTER TABLE`. Levels: Global, database, table.
`ALTER ROUTINE`	Enable stored routines to be altered or dropped. Levels: Global, database, routine.
`CREATE`	Enable database and table creation. Levels: Global, database, table.
`CREATE ROLE`	Enable role creation. Level: Global.
`CREATE ROUTINE`	Enable stored routine creation. Levels: Global, database.
`CREATE TABLESPACE`	Enable tablespaces and log file groups to be created, altered, or dropped. Level: Global.
`CREATE TEMPORARY TABLES`	Enable use of `CREATE TEMPORARY TABLE`. Levels: Global, database.
`CREATE USER`	Enable use of `CREATE USER`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES`. Level: Global.
`CREATE VIEW`	Enable views to be created or altered. Levels: Global, database, table.

Privilege	Meaning and Grantable Levels
DELETE	Enable use of DELETE. Level: Global, database, table.
DROP	Enable databases, tables, and views to be dropped. Levels: Global, database, table.
DROP ROLE	Enable roles to be dropped. Level: Global.
EVENT	Enable use of events for the Event Scheduler. Levels: Global, database.
EXECUTE	Enable the user to execute stored routines. Levels: Global, database, routine.
FILE	Enable the user to cause the server to read or write files. Level: Global.
GRANT OPTION	Enable privileges to be granted to or removed from other accounts. Levels: Global, database, table, routine, proxy.
INDEX	Enable indexes to be created or dropped. Levels: Global, database, table.
INSERT	Enable use of INSERT. Levels: Global, database, table, column.
LOCK TABLES	Enable use of LOCK TABLES on tables for which you have the SELECT privilege. Levels: Global, database.
PROCESS	Enable the user to see all processes with SHOW PROCESSLIST. Level: Global.
PROXY	Enable user proxying. Level: From user to user.
REFERENCES	Enable foreign key creation. Levels: Global, database, table, column.
RELOAD	Enable use of FLUSH operations. Level: Global.
REPLICATION CLIENT	Enable the user to ask where master or slave servers are. Level: Global.
REPLICATION SLAVE	Enable replication slaves to read binary log events from the master. Level: Global.
SELECT	Enable use of SELECT. Levels: Global, database, table, column.
SHOW DATABASES	Enable SHOW DATABASES to show all databases. Level: Global.
SHOW VIEW	Enable use of SHOW CREATE VIEW. Levels: Global, database, table.
SHUTDOWN	Enable use of mysqladmin shutdown. Level: Global.
SUPER	Enable use of other administrative operations such as CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and mysqladmin debug command. Level: Global.
TRIGGER	Enable trigger operations. Levels: Global, database, table.
UPDATE	Enable use of UPDATE. Levels: Global, database, table, column.
USAGE	Synonym for “no privileges”

Table 13.12 Permissible Dynamic Privileges for GRANT and REVOKE

Privilege	Meaning and Grantable Levels
APPLICATION_PASSWORD_ADMIN	Enable dual password administration. Level: Global.
AUDIT_ADMIN	Enable audit log configuration. Level: Global.
BACKUP_ADMIN	Enable backup administration. Level: Global.
BINLOG_ADMIN	Enable binary log control. Level: Global.
BINLOG_ENCRYPTION_ADMIN	Enable activation and deactivation of binary log encryption. Level: Global.
CLONE_ADMIN	Enable clone administration. Level: Global.

Privilege	Meaning and Grantable Levels
CONNECTION_ADMIN	Enable connection limit/restriction control. Level: Global.
ENCRYPTION_KEY_ADMIN	Enable InnoDB key rotation. Level: Global.
FIREWALL_ADMIN	Enable firewall rule administration, any user. Level: Global.
FIREWALL_USER	Enable firewall rule administration, self. Level: Global.
GROUP_REPLICATION_ADMIN	Enable Group Replication control. Level: Global.
INNODB_REDO_LOG_ARCHIVE	Enable redo log archiving administration. Level: Global.
NDB_STORED_USER	Enable sharing of user or role between SQL nodes (NDB Cluster). Level: Global.
PERSIST_RO_VARIABLES_ADMIN	Enable persisting read-only system variables. Level: Global.
REPLICATION_APPLIER	Act as the PRIVILEGE_CHECKS_USER for a replication channel. Level: Global.
REPLICATION_SLAVE_ADMIN	Enable regular replication control. Level: Global.
RESOURCE_GROUP_ADMIN	Enable resource group administration. Level: Global.
RESOURCE_GROUP_USER	Enable resource group administration. Level: Global.
ROLE_ADMIN	Enable roles to be granted or revoked, use of WITH ADMIN OPTION. Level: Global.
SESSION_VARIABLES_ADMIN	Enable setting restricted session system variables. Level: Global.
SET_USER_ID	Enable setting non-self DEFINER values. Level: Global.
SYSTEM_USER	Designate account as system account. Level: Global.
SYSTEM_VARIABLES_ADMIN	Enable modifying or persisting global system variables. Level: Global.
TABLE_ENCRYPTION_ADMIN	Enable overriding default encryption settings. Level: Global.
VERSION_TOKEN_ADMIN	Enable use of Version Tokens UDFs. Level: Global.
XA_RECOVER_ADMIN	Enable XA RECOVER execution. Level: Global.

A trigger is associated with a table. To create or drop a trigger, you must have the TRIGGER privilege for the table, not the trigger.

In GRANT statements, the ALL [PRIVILEGES] or PROXY privilege must be named by itself and cannot be specified along with other privileges. ALL [PRIVILEGES] stands for all privileges available for the level at which privileges are to be granted except for the GRANT OPTION and PROXY privileges.

MySQL account information is stored in the tables of the mysql system database. For additional details, consult Section 6.2, “Access Control and Account Management”, which discusses the mysql system database and the access control system extensively.

If the grant tables hold privilege rows that contain mixed-case database or table names and the lower_case_table_names system variable is set to a nonzero value, REVOKE cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (GRANT will not create such rows when lower_case_table_names is set, but such rows might have been created prior to setting that variable. The lower_case_table_names setting can only be configured at server startup.)

Privileges can be granted at several levels, depending on the syntax used for the ON clause. For REVOKE, the same ON syntax specifies which privileges to remove.

For the global, database, table, and routine levels, GRANT ALL assigns only the privileges that exist at the level you are granting. For example, GRANT ALL ON *db_name*.* is a database-level statement, so it

does not grant any global-only privileges such as `FILE`. Granting `ALL` does not assign the `GRANT OPTION` or `PROXY` privilege.

The *`object_type`* clause, if present, should be specified as `TABLE`, `FUNCTION`, or `PROCEDURE` when the following object is a table, a stored function, or a stored procedure.

The privileges that a user holds for a database, table, column, or routine are formed additively as the logical `OR` of the account privileges at each of the privilege levels, including the global level. It is not possible to deny a privilege granted at a higher level by absence of that privilege at a lower level. For example, this statement grants the `SELECT` and `INSERT` privileges globally:

```
GRANT SELECT, INSERT ON *.* TO u1;
```

The globally granted privileges apply to all databases, tables, and columns, even though not granted at any of those lower levels.

As of MySQL 8.0.16, it is possible to explicitly deny a privilege granted at the global level by revoking it for particular databases, if the `partial_revokes` system variable is enabled:

```
GRANT SELECT, INSERT, UPDATE ON *.* TO u1;
REVOKE INSERT, UPDATE ON db1.* FROM u1;
```

The result of the preceding statements is that `SELECT` applies globally to all tables, whereas `INSERT` and `UPDATE` apply globally except to tables in `db1`. Account access to `db1` is read only.

Details of the privilege-checking procedure are presented in Section 6.2.7, “Access Control, Stage 2: Request Verification”.

If you are using table, column, or routine privileges for even one user, the server examines table, column, and routine privileges for all users and this slows down MySQL a bit. Similarly, if you limit the number of queries, updates, or connections for any users, the server must monitor these values.

MySQL enables you to grant privileges on databases or tables that do not exist. For tables, the privileges to be granted must include the `CREATE` privilege. *This behavior is by design*, and is intended to enable the database administrator to prepare user accounts and privileges for databases or tables that are to be created at a later time.

Important

MySQL does not automatically revoke any privileges when you drop a database or table. However, if you drop a routine, any routine-level privileges granted for that routine are revoked.

Global Privileges

Global privileges are administrative or apply to all databases on a given server. To assign global privileges, use `ON *.*` syntax:

```
GRANT ALL ON *.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON *.* TO 'someuser'@'somehost';
```

The `CREATE TABLESPACE`, `CREATE USER`, `FILE`, `PROCESS`, `RELOAD`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, `SHOW DATABASES`, `SHUTDOWN`, and `SUPER` static privileges are administrative and can only be granted globally.

Dynamic privileges are all global and can only be granted globally.

Other privileges can be granted globally or at more specific levels.

The effect of `GRANT OPTION` granted at the global level differs for static and dynamic privileges:

- `GRANT OPTION` granted for any static global privilege applies to all static global privileges.
- `GRANT OPTION` granted for any dynamic privilege applies only to that dynamic privilege.

`GRANT ALL` at the global level grants all static global privileges and all currently registered dynamic privileges. A dynamic privilege registered subsequent to execution of the `GRANT` statement is not granted retroactively to any account.

MySQL stores global privileges in the `mysql.user` system table.

Database Privileges

Database privileges apply to all objects in a given database. To assign database-level privileges, use `ON *db_name.** syntax:

```
GRANT ALL ON mydb.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.* TO 'someuser'@'somehost';
```

If you use `ON *` syntax (rather than `ON *.*`), privileges are assigned at the database level for the default database. An error occurs if there is no default database.

The `CREATE`, `DROP`, `EVENT`, `GRANT OPTION`, `LOCK TABLES`, and `REFERENCES` privileges can be specified at the database level. Table or routine privileges also can be specified at the database level, in which case they apply to all tables or routines in the database.

MySQL stores database privileges in the `mysql.db` system table.

Table Privileges

Table privileges apply to all columns in a given table. To assign table-level privileges, use `ON *db_name.tbl_name*` syntax:

```
GRANT ALL ON mydb.mytbl TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.mytbl TO 'someuser'@'somehost';
```

If you specify *`tbl_name`* rather than *`db_name.tbl_name`*, the statement applies to *`tbl_name`* in the default database. An error occurs if there is no default database.

The permissible *`priv_type`* values at the table level are `ALTER`, `CREATE VIEW`, `CREATE`, `DELETE`, `DROP`, `GRANT OPTION`, `INDEX`, `INSERT`, `REFERENCES`, `SELECT`, `SHOW VIEW`, `TRIGGER`, and `UPDATE`.

Table-level privileges apply to base tables and views. They do not apply to tables created with `CREATE TEMPORARY TABLE`, even if the table names match. For information about `TEMPORARY` table privileges, see Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”.

MySQL stores table privileges in the `mysql.tables_priv` system table.

Column Privileges

Column privileges apply to single columns in a given table. Each privilege to be granted at the column level must be followed by the column or columns, enclosed within parentheses.

```
GRANT SELECT (col1), INSERT (col1, col2) ON mydb.mytbl TO 'someuser'@'somehost';
```

The permissible *`priv_type`* values for a column (that is, when you use a *`column_list`* clause) are `INSERT`, `REFERENCES`, `SELECT`, and `UPDATE`.

MySQL stores column privileges in the `mysql.columns_priv` system table.

Stored Routine Privileges

The `ALTER ROUTINE`, `CREATE ROUTINE`, `EXECUTE`, and `GRANT OPTION` privileges apply to stored routines (procedures and functions). They can be granted at the global and database levels. Except for `CREATE ROUTINE`, these privileges can be granted at the routine level for individual routines.

```
GRANT CREATE ROUTINE ON mydb.* TO 'someuser'@'somehost';
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'someuser'@'somehost';
```

The permissible *`priv_type`* values at the routine level are `ALTER ROUTINE`, `EXECUTE`, and `GRANT OPTION`. `CREATE ROUTINE` is not a routine-level privilege because you must have the privilege at the global or database level to create a routine in the first place.

MySQL stores routine-level privileges in the `mysql.procs_priv` system table.

Proxy User Privileges

The `PROXY` privilege enables one user to be a proxy for another. The proxy user impersonates or takes the identity of the proxied user; that is, it assumes the privileges of the proxied user.

```
GRANT PROXY ON 'localuser'@'localhost' TO 'externaluser'@'somehost';
```

When `PROXY` is granted, it must be the only privilege named in the `GRANT` statement, and the only permitted `WITH` option is `WITH GRANT OPTION`.

Proxying requires that the proxy user authenticate through a plugin that returns the name of the proxied user to the server when the proxy user connects, and that the proxy user have the `PROXY` privilege for the proxied user. For details and examples, see Section 6.2.18, “Proxy Users”.

MySQL stores proxy privileges in the `mysql.proxies_priv` system table.

Granting Roles

`GRANT` syntax without an `ON` clause grants roles rather than individual privileges. A role is a named collection of privileges; see Section 6.2.10, “Using Roles”. For example:

```
GRANT 'role1', 'role2' TO 'user1'@'localhost', 'user2'@'localhost';
```

Each role to be granted must exist, as well as each user account or role to which it is to be granted. As of MySQL 8.0.16, roles cannot be granted to anonymous users.

Granting a role does not automatically cause the role to be active. For information about role activation and inactivation, see Activating Roles.

These privileges are required to grant roles:

- If you have the `ROLE_ADMIN` or `SUPER` privilege, you can grant or revoke any role to users or roles.

- If you were granted a role with a GRANT statement that includes the WITH ADMIN OPTION clause, you become able to grant that role to other users or roles, or revoke it from other users or roles, as long as the role is active at such time as you subsequently grant or revoke it. This includes the ability to use WITH ADMIN OPTION itself.
- To grant a role that has the SYSTEM_USER privilege, you must have the SYSTEM_USER privilege.

It is possible to create circular references with GRANT. For example:

```
CREATE USER 'u1', 'u2';
CREATE ROLE 'r1', 'r2';

GRANT 'u1' TO 'u1';   -- simple loop: u1 => u1
GRANT 'r1' TO 'r1';   -- simple loop: r1 => r1

GRANT 'r2' TO 'u2';
GRANT 'u2' TO 'r2';   -- mixed user/role loop: u2 => r2 => u2
```

Circular grant references are permitted but add no new privileges or roles to the grantee because a user or role already has its privileges and roles.

The AS Clause and Privilege Restrictions

As of MySQL 8.0.16, GRANT has an AS *user* [WITH ROLE] clause that specifies additional information about the privilege context to use for statement execution. This syntax is visible at the SQL level, although its primary purpose is to enable uniform replication across all nodes of grantor privilege restrictions imposed by partial revokes, by causing those restrictions to appear in the binary log. For information about partial revokes, see Section 6.2.12, “Privilege Restriction Using Partial Revokes”.

When the AS *user* clause is specified, statement execution takes into account any privilege restrictions associated with the named user, including all roles specified by WITH ROLE, if present. The result is that the privileges actually granted by the statement may be reduced relative to those specified.

These conditions apply to the AS *user* clause:

- AS has an effect only when the named *user* has privilege restrictions (which implies that the partial_revokes system variable is enabled).
- If WITH ROLE is given, all roles named must be granted to the named *user*.
- The named *user* should be a MySQL account specified as '*user_name*'@'*host_name*', CURRENT_USER, or CURRENT_USER(). The current user may be named together with WITH ROLE for the case that the executing user wants GRANT to execute with a set of roles applied that may differ from the roles active within the current session.
- AS cannot be used to gain privileges not possessed by the user who executes the GRANT statement. The executing user must have at least the privileges to be granted, but the AS clause can only restrict the privileges granted, not escalate them.
- With respect to the privileges to be granted, AS cannot specify a user/role combination that has more privileges (fewer restrictions) than the user who executes the GRANT statement. The AS user/role combination is permitted to have more privileges than the executing user, but only if the statement does not grant those additional privileges.
- AS is supported only for granting global privileges (ON *.*).
- AS is not supported for PROXY grants.

The following example illustrates the effect of the AS clause. Create a user u1 that has some global privileges, as well as restrictions on those privileges:

```
CREATE USER u1;
GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO u1;
REVOKE INSERT, UPDATE ON schema1.* FROM u1;
REVOKE SELECT ON schema2.* FROM u1;
```

Also create a role r1 that lifts some of the privilege restrictions and grant the role to u1:

```
CREATE ROLE r1;
GRANT INSERT ON schema1.* TO r1;
GRANT SELECT ON schema2.* TO r1;
GRANT r1 TO u1;
```

Now, using an account that has no privilege restrictions of its own, grant to multiple users the same set of global privileges, but each with different restrictions imposed by the AS clause, and check which privileges are actually granted.

- The GRANT statement here has no AS clause, so the privileges granted are exactly those specified:

```
mysql> CREATE USER u2;
mysql> GRANT SELECT, INSERT, UPDATE ON *.* TO u2;
mysql> SHOW GRANTS FOR u2;
+-----------------------------------------------+
| Grants for u2@%                               |
+-----------------------------------------------+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u2`@`%` |
+-----------------------------------------------+
```

- The GRANT statement here has an AS clause, so the privileges granted are those specified but with the restrictions from u1 applied:

```
mysql> CREATE USER u3;
mysql> GRANT SELECT, INSERT, UPDATE ON *.* TO u3 AS u1;
mysql> SHOW GRANTS FOR u3;
+-------------------------------------------------+
| Grants for u3@%                                 |
+-------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u3`@`%`    |
| REVOKE INSERT, UPDATE ON `schema1`.* FROM `u3`@`%` |
| REVOKE SELECT ON `schema2`.* FROM `u3`@`%`         |
+-------------------------------------------------+
```

  As mentioned previously, the AS clause can only add privilege restrictions; it cannot escalate privileges. Thus, although u1 has the DELETE privilege, that is not included in the privileges granted because the statement does not specify granting DELETE.

- The AS clause for the GRANT statement here makes the role r1 active for u1. That role lifts some of the restrictions on u1. Consequently, the privileges granted have some restrictions, but not so many as for the previous GRANT statement:

```
mysql> CREATE USER u4;
mysql> GRANT SELECT, INSERT, UPDATE ON *.* TO u4 AS u1 WITH ROLE r1;
mysql> SHOW GRANTS FOR u4;
+-----------------------------------------------+
| Grants for u4@%                               |
+-----------------------------------------------+
| GRANT SELECT, INSERT, UPDATE ON *.* TO `u4`@`%` |
```

```
| REVOKE UPDATE ON `schema1`.* FROM `u4`@`%`      |
+-----------------------------------------------+
```

If a `GRANT` statement includes an `AS user` clause, privilege restrictions on the user who executes the statement are ignored (rather than applied as they would be in the absence of an `AS` clause).

Other Account Characteristics

The optional `WITH` clause is used to enable a user to grant privileges to other users. The `WITH GRANT OPTION` clause gives the user the ability to give to other users any privileges the user has at the specified privilege level.

To grant the `GRANT OPTION` privilege to an account without otherwise changing its privileges, do this:

```
GRANT USAGE ON *.* TO 'someuser'@'somehost' WITH GRANT OPTION;
```

Be careful to whom you give the `GRANT OPTION` privilege because two users with different privileges may be able to combine privileges!

You cannot grant another user a privilege which you yourself do not have; the `GRANT OPTION` privilege enables you to assign only those privileges which you yourself possess.

Be aware that when you grant a user the `GRANT OPTION` privilege at a particular privilege level, any privileges the user possesses (or may be given in the future) at that level can also be granted by that user to other users. Suppose that you grant a user the `INSERT` privilege on a database. If you then grant the `SELECT` privilege on the database and specify `WITH GRANT OPTION`, that user can give to other users not only the `SELECT` privilege, but also `INSERT`. If you then grant the `UPDATE` privilege to the user on the database, the user can grant `INSERT`, `SELECT`, and `UPDATE`.

For a nonadministrative user, you should not grant the `ALTER` privilege globally or for the `mysql` system database. If you do that, the user can try to subvert the privilege system by renaming tables!

For additional information about security risks associated with particular privileges, see Section 6.2.2, “Privileges Provided by MySQL”.

MySQL and Standard SQL Versions of GRANT

The biggest differences between the MySQL and standard SQL versions of `GRANT` are:

- MySQL associates privileges with the combination of a host name and user name and not with only a user name.
- Standard SQL does not have global or database-level privileges, nor does it support all the privilege types that MySQL supports.
- MySQL does not support the standard SQL `UNDER` privilege.
- Standard SQL privileges are structured in a hierarchical manner. If you remove a user, all privileges the user has been granted are revoked. This is also true in MySQL if you use `DROP USER`. See Section 13.7.1.5, “DROP USER Syntax”.
- In standard SQL, when you drop a table, all privileges for the table are revoked. In standard SQL, when you revoke a privilege, all privileges that were granted based on that privilege are also revoked. In MySQL, privileges can be dropped with `DROP USER` or `REVOKE` statements.
- In MySQL, it is possible to have the `INSERT` privilege for only some of the columns in a table. In this case, you can still execute `INSERT` statements on the table, provided that you insert values only for those columns for which you have the `INSERT` privilege. The omitted columns are set to their implicit

default values if strict SQL mode is not enabled. In strict mode, the statement is rejected if any of the omitted columns have no default value. (Standard SQL requires you to have the `INSERT` privilege on all columns.) For information about strict SQL mode and implicit default values, see Section 5.1.11, “Server SQL Modes”, and Section 11.7, “Data Type Default Values”.

13.7.1.7 RENAME USER Syntax

```
RENAME USER old_user TO new_user
    [, old_user TO new_user] ...
```

The `RENAME USER` statement renames existing MySQL accounts. An error occurs for old accounts that do not exist or new accounts that already exist.

To use `RENAME USER`, you must have the global `CREATE USER` privilege, or the `UPDATE` privilege for the `mysql` system database. When the `read_only` system variable is enabled, `RENAME USER` additionally requires the `CONNECTION_ADMIN` or `SUPER` privilege.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. For example:

```
RENAME USER 'jeffrey'@'localhost' TO 'jeff'@'127.0.0.1';
```

The host name part of the account name, if omitted, defaults to `'%'`.

`RENAME USER` causes the privileges held by the old user to be those held by the new user. However, `RENAME USER` does not automatically drop or invalidate databases or objects within them that the old user created. This includes stored programs or views for which the `DEFINER` attribute names the old user. Attempts to access such objects may produce an error if they execute in definer security context. (For information about security context, see Section 24.6, “Stored Object Access Control”.)

The privilege changes take effect as indicated in Section 6.2.13, “When Privilege Changes Take Effect”.

13.7.1.8 REVOKE Syntax

```
REVOKE
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    FROM user_or_role [, user_or_role] ...

REVOKE ALL [PRIVILEGES], GRANT OPTION
    FROM user_or_role [, user_or_role] ...

REVOKE PROXY ON user_or_role
    FROM user_or_role [, user_or_role] ...

REVOKE role [, role ] ...
    FROM user_or_role [, user_or_role ] ...

user_or_role: {
    user (see Section 6.2.4, “Specifying Account Names”)
  | role (see Section 6.2.5, “Specifying Role Names”.
}
```

The `REVOKE` statement enables system administrators to revoke privileges and roles, which can be revoked from user accounts and roles.

For details on the levels at which privileges exist, the permissible *`priv_type`*, *`priv_level`*, and *`object_type`* values, and the syntax for specifying users and passwords, see Section 13.7.1.6, “GRANT Syntax”.

For information about roles, see Section 6.2.10, “Using Roles”.

When the `read_only` system variable is enabled, `REVOKE` requires the `CONNECTION_ADMIN` or `SUPER` privilege in addition to any other required privileges described in the following discussion.

`REVOKE` either succeeds for all named users and roles or rolls back and has no effect if any error occurs. The statement is written to the binary log only if it succeeds for all named users and roles.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. Each role name uses the format described in Section 6.2.5, “Specifying Role Names”. For example:

```
REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';
REVOKE 'role1', 'role2' FROM 'user1'@'localhost', 'user2'@'localhost';
REVOKE SELECT ON world.* FROM 'role3';
```

The host name part of the account or role name, if omitted, defaults to `'%'`.

To use the first `REVOKE` syntax, you must have the `GRANT OPTION` privilege, and you must have the privileges that you are revoking.

To revoke all privileges, use the second syntax, which drops all global, database, table, column, and routine privileges for the named users or roles:

```
REVOKE ALL PRIVILEGES, GRANT OPTION
  FROM user_or_role [, user_or_role] ...
```

`REVOKE ALL PRIVILEGES, GRANT OPTION` does not revoke any roles.

To use this `REVOKE` syntax, you must have the global `CREATE USER` privilege, or the `UPDATE` privilege for the `mysql` system database.

The syntax for which the `REVOKE` keyword is followed by one or more role names takes a `FROM` clause indicating one or more users or roles from which to revoke the roles.

Roles named in the `mandatory_roles` system variable value cannot be revoked.

A revoked role immediately affects any user account from which it was revoked, such that within any current session for the account, its privileges are adjusted for the next statement executed.

Revoking a role revokes the role itself, not the privileges that it represents. Suppose that an account is granted a role that includes a given privilege, and is also granted the privilege explicitly or another role that includes the privilege. In this case, the account still possesses that privilege if the first role is revoked. For example, if an account is granted two roles that each include `SELECT`, the account still can select after either role is revoked.

`REVOKE ALL ON *.*` (at the global level) revokes all granted static global privileges and all granted dynamic privileges.

User accounts and roles from which privileges and roles are to be revoked must exist, but the privileges and roles to be revoked need not be currently granted to them.

A revoked privilege that is granted but not known to the server is revoked with a warning. This situtation can occur for dynamic privileges. For example, a dynamic privilege can be granted while the component that registers it is installed, but if that component is subsequently uninstalled, the privilege becomes unregistered, although accounts that possess the privilege still possess it and it can be revoked from them.

`REVOKE` removes privileges, but does not remove rows from the `mysql.user` system table. To remove a user account entirely, use `DROP USER`. See Section 13.7.1.5, “DROP USER Syntax”.

If the grant tables hold privilege rows that contain mixed-case database or table names and the `lower_case_table_names` system variable is set to a nonzero value, `REVOKE` cannot be used to revoke these privileges. It will be necessary to manipulate the grant tables directly. (`GRANT` will not create such rows when `lower_case_table_names` is set, but such rows might have been created prior to setting the variable. The `lower_case_table_names` setting can only be configured when initializing the server.)

When successfully executed from the `mysql` program, `REVOKE` responds with `Query OK, 0 rows affected`. To determine what privileges remain after the operation, use `SHOW GRANTS`. See Section 13.7.7.21, “SHOW GRANTS Syntax”.

13.7.1.9 SET DEFAULT ROLE Syntax

```
SET DEFAULT ROLE
    {NONE | ALL | role [, role ] ...}
    TO user [, user ] ...
```

For each *`user`* named immediately after the `TO` keyword, this statement defines which roles become active when the user connects to the server and authenticates, or when the user executes the `SET ROLE DEFAULT` statement during a session.

`SET DEFAULT ROLE` is alternative syntax for `ALTER USER ... DEFAULT ROLE` (see Section 13.7.1.1, “ALTER USER Syntax”). However, `ALTER USER` can set the default for only a single user, whereas `SET DEFAULT ROLE` can set the default for multiple users. On the other hand, you can specify `CURRENT_USER` as the user name for the `ALTER USER` statement, whereas you cannot for `SET DEFAULT ROLE`.

`SET DEFAULT ROLE` requires these privileges:

- Setting the default roles for another user requires the global `CREATE USER` privilege, or the `UPDATE` privilege for the `mysql.default_roles` system table.
- Setting the default roles for yourself requires no special privileges, as long as the roles you want as the default have been granted to you.

Each role name uses the format described in Section 6.2.5, “Specifying Role Names”. For example:

```
SET DEFAULT ROLE administrator, developer TO 'joe'@'10.0.0.1';
```

The host name part of the role name, if omitted, defaults to `'%'`.

The clause following the `DEFAULT ROLE` keywords permits these values:

- `NONE`: Set the default to `NONE` (no roles).
- `ALL`: Set the default to all roles granted to the account.
- *`role [, role ] ...`*: Set the default to the named roles, which must exist and be granted to the account at the time `SET DEFAULT ROLE` is executed.

Note

`SET DEFAULT ROLE` and `SET ROLE DEFAULT` are different statements:

- `SET DEFAULT ROLE` defines which account roles to activate by default within account sessions.
- `SET ROLE DEFAULT` sets the active roles within the current session to the current account default roles.

For role usage examples, see Section 6.2.10, “Using Roles”.

13.7.1.10 SET PASSWORD Syntax

```
SET PASSWORD [FOR user] auth_option
    [REPLACE 'current_auth_string']
    [RETAIN CURRENT PASSWORD]

auth_option: {
    = 'auth_string'
  | TO RANDOM
}
```

The `SET PASSWORD` statement assigns a password to a MySQL user account. The password may be either explicitly specified in the statement or randomly generated by MySQL. The statement may also include a password-verification clause that specifies the account current password to be replaced, and a clause that manages whether an account has a secondary password. `'auth_string'` and `'current_auth_string'` each represent a cleartext (unencrypted) password.

Note

Rather than using `SET PASSWORD` to assign passwords, `ALTER USER` is the preferred statement for account alterations, including assigning passwords. For example:

```
ALTER USER user IDENTIFIED BY 'auth_string';
```

Note

Clauses for random password generation, password verification, and secondary passwords apply only to accounts that use an authentication plugin that stores credentials internally to MySQL. For accounts that use plugins that perform authentication against a credentials system that is external to MySQL, password management must be handled externally against that system as well. For more information about internal credentials storage, see Section 6.2.15, “Password Management”.

The `REPLACE 'current_auth_string'` clause performs password verification and is available as of MySQL 8.0.13. If given:

- `REPLACE` specifies the account current password to be replaced, as a cleartext (unencrypted) string.
- The clause must be given if password changes for the account are required to specify the current password, as verification that the user attempting to make the change actually knows the current password.
- The clause is optional if password changes for the account may but need not specify the current password.
- The statement fails if the clause is given but does not match the current password, even if the clause is optional.

- `REPLACE` can be specified only when changing the account password for the current user.

For more information about password verification by specifying the current password, see Section 6.2.15, “Password Management”.

The `RETAIN CURRENT PASSWORD` clause implements dual-password capability and is available as of MySQL 8.0.14. If given:

- `RETAIN CURRENT PASSWORD` retains an account current password as its secondary password, replacing any existing secondary password. The new password becomes the primary password, but clients can use the account to connect to the server using either the primary or secondary password. (Exception: If the new password specified by the `SET PASSWORD` statement is empty, the secondary password becomes empty as well, even if `RETAIN CURRENT PASSWORD` is given.)
- If you specify `RETAIN CURRENT PASSWORD` for an account that has an empty primary password, the statement fails.
- If an account has a secondary password and you change its primary password without specifying `RETAIN CURRENT PASSWORD`, the secondary password remains unchanged.

For more information about use of dual passwords, see Section 6.2.15, “Password Management”.

`SET PASSWORD` permits these *`auth_option`* syntaxes:

- `= '`*`auth_string`*`'`

  Assigns the account the given literal password.

- `TO RANDOM`

  Assigns the account a password randomly generated by MySQL. The statement also returns the cleartext password in a result set to make it available to the user or application executing the statement.

  For details about the result set and characteristics of randomly generated passwords, see Random Password Generation.

  Random password generation is available as of MySQL 8.0.18.

Important

Under some circumstances, `SET PASSWORD` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about the conditions under which this occurs for the server logs and how to control it, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, see Section 4.5.1.3, “mysql Client Logging”.

`SET PASSWORD` can be used with or without a `FOR` clause that explicitly names a user account:

- With a `FOR` *`user`* clause, the statement sets the password for the named account, which must exist:

```
SET PASSWORD FOR 'jeffrey'@'localhost' = 'auth_string';
```

- With no `FOR` *`user`* clause, the statement sets the password for the current user:

```
SET PASSWORD = 'auth_string';
```

Any client who connects to the server using a nonanonymous account can change the password for that account. (In particular, you can change your own password.) To see which account the server authenticated you as, invoke the `CURRENT_USER()` function:

```
SELECT CURRENT_USER();
```

If a `FOR user` clause is given, the account name uses the format described in Section 6.2.4, “Specifying Account Names”. For example:

```
SET PASSWORD FOR 'bob'@'%.example.org' = 'auth_string';
```

The host name part of the account name, if omitted, defaults to `'%'`.

`SET PASSWORD` interprets the string as a cleartext string, passes it to the authentication plugin associated with the account, and stores the result returned by the plugin in the account row in the `mysql.user` system table. (The plugin is given the opportunity to hash the value into the encryption format it expects. The plugin may use the value as specified, in which case no hashing occurs.)

Setting the password for a named account (with a `FOR` clause) requires the `UPDATE` privilege for the `mysql` system database. Setting the password for yourself (for a nonanonymous account with no `FOR` clause) requires no special privileges.

Statements that modify secondary passwords require these privileges:

- The `APPLICATION_PASSWORD_ADMIN` privilege is required to use the `RETAIN CURRENT PASSWORD` clause for `SET PASSWORD` statements that apply to your own account. The privilege is required to manipulate your own secondary password because most users require only one password.
- If an account is to be permitted to manipulate secondary passwords for all accounts, it should be granted the `CREATE USER` privilege rather than `APPLICATION_PASSWORD_ADMIN`.

When the `read_only` system variable is enabled, `SET PASSWORD` requires the `CONNECTION_ADMIN` or `SUPER` privilege in addition to any other required privileges.

For additional information about setting passwords and authentication plugins, see Section 6.2.14, “Assigning Account Passwords”, and Section 6.2.17, “Pluggable Authentication”.

13.7.1.11 SET ROLE Syntax

```
SET ROLE {
    DEFAULT
  | NONE
  | ALL
  | ALL EXCEPT role [, role ] ...
  | role [, role ] ...
}
```

`SET ROLE` modifies the current user's effective privileges within the current session by specifying which of its granted roles are active. Granted roles include those granted explicitly to the user and those named in the `mandatory_roles` system variable value.

Examples:

```
SET ROLE DEFAULT;
SET ROLE 'role1', 'role2';
SET ROLE ALL;
SET ROLE ALL EXCEPT 'role1', 'role2';
```

Each role name uses the format described in Section 6.2.5, “Specifying Role Names”. The host name part of the role name, if omitted, defaults to `'%'`.

Privileges that the user has been granted directly (rather than through roles) remain unaffected by changes to the active roles.

The statement permits these role specifiers:

- `DEFAULT`: Activate the account default roles. Default roles are those specified with `SET DEFAULT ROLE`.

  When a user connects to the server and authenticates successfully, the server determines which roles to activate as the default roles. If the `activate_all_roles_on_login` system variable is enabled, the server activates all granted roles. Otherwise, the server executes `SET ROLE DEFAULT` implicitly. The server activates only default roles that can be activated. The server writes warnings to its error log for default roles that cannot be activated, but the client receives no warnings.

  If a user executes `SET ROLE DEFAULT` during a session, an error occurs if any default role cannot be activated (for example, if it does not exist or is not granted to the user). In this case, the current active roles are not changed.

- `NONE`: Set the active roles to `NONE` (no active roles).

- `ALL`: Activate all roles granted to the account.

- `ALL EXCEPT` *`role`* `[,` *`role`* `] ...`: Activate all roles granted to the account except those named. The named roles need not exist or be granted to the account.

- *`role`* `[,` *`role`* `] ...`: Activate the named roles, which must be granted to the account.

Note

`SET DEFAULT ROLE` and `SET ROLE DEFAULT` are different statements:

- `SET DEFAULT ROLE` defines which account roles to activate by default within account sessions.
- `SET ROLE DEFAULT` sets the active roles within the current session to the current account default roles.

For role usage examples, see Section 6.2.10, “Using Roles”.

13.7.2 Resource Group Management Statements

MySQL supports creation and management of resource groups, and permits assigning threads running within the server to particular groups so that threads execute according to the resources available to the group. This section describes the SQL statements available for resource group management. For general discussion of the resource group capability, see Section 8.12.5, “Resource Groups”.

13.7.2.1 ALTER RESOURCE GROUP Syntax

```
ALTER RESOURCE GROUP group_name
    [VCPU [=] vcpu_spec [, vcpu_spec] ...]
    [THREAD_PRIORITY [=] N]
    [ENABLE|DISABLE [FORCE]]

vcpu_spec: {N | M - N}
```

`ALTER RESOURCE GROUP` is used for resource group management (see Section 8.12.5, "Resource Groups"). This statement alters modifiable attributes of an existing resource group. It requires the `RESOURCE_GROUP_ADMIN` privilege.

`group_name` identifies which resource group to alter. If the group does not exist, an error occurs.

The attributes for CPU affinity, priority, and whether the group is enabled can be modified with `ALTER RESOURCE GROUP`. These attributes are specified the same way as described for `CREATE RESOURCE GROUP` (see Section 13.7.2.2, "CREATE RESOURCE GROUP Syntax"). Only the attributes specified are altered. Unspecified attributes retain their current values.

The `FORCE` modifier is used with `DISABLE`. It determines statement behavior if the resource group has any threads assigned to it:

- If `FORCE` is not given, existing threads in the group continue to run until they terminate, but new threads cannot be assigned to the group.
- If `FORCE` is given, existing threads in the group are moved to their respective default group (system threads to `SYS_default`, user threads to `USR_default`).

The name and type attributes are set at group creation time and cannot be modified thereafter with `ALTER RESOURCE GROUP`.

Examples:

- Alter a group CPU affinity:

  ```
  ALTER RESOURCE GROUP rg1 VCPU = 0-63;
  ```

- Alter a group thread priority:

  ```
  ALTER RESOURCE GROUP rg2 THREAD_PRIORITY = 5;
  ```

- Disable a group, moving any threads assigned to it to the default groups:

  ```
  ALTER RESOURCE GROUP rg3 DISABLE FORCE;
  ```

Resource group management is local to the server on which it occurs. `ALTER RESOURCE GROUP` statements are not written to the binary log and are not replicated.

13.7.2.2 CREATE RESOURCE GROUP Syntax

```
CREATE RESOURCE GROUP group_name
    TYPE = {SYSTEM|USER}
    [VCPU [=] vcpu_spec [, vcpu_spec] ...]
    [THREAD_PRIORITY [=] N]
    [ENABLE|DISABLE]

vcpu_spec: {N | M - N}
```

`CREATE RESOURCE GROUP` is used for resource group management (see Section 8.12.5, “Resource Groups”). This statement creates a new resource group and assigns its initial attribute values. It requires the `RESOURCE_GROUP_ADMIN` privilege.

`group_name` identifies which resource group to create. If the group already exists, an error occurs.

The `TYPE` attribute is required. It should be `SYSTEM` for a system resource group, `USER` for a user resource group. The group type affects permitted `THREAD_PRIORITY` values, as described later.

The `VCPU` attribute indicates the CPU affinity; that is, the set of virtual CPUs the group can use:

- If `VCPU` is not given, the resource group has no CPU affinity and can use all available CPUs.
- If `VCPU` is given, the attribute value is a list of comma-separated CPU numbers or ranges:
  - Each number must be an integer in the range from 0 to the number of CPUs − 1. For example, on a system with 64 CPUs, the number can range from 0 to 63.
  - A range is given in the form *`M`* − *`N`*, where *`M`* is less than or equal to *`N`* and both numbers are in the CPU range.
  - If a CPU number is an integer outside the permitted range or is not an integer, an error occurs.

Example `VCPU` specifiers (these are all equivalent):

```
VCPU = 0,1,2,3,9,10
VCPU = 0-3,9-10
VCPU = 9,10,0-3
VCPU = 0,10,1,9,3,2
```

The `THREAD_PRIORITY` attribute indicates the priority for threads assigned to the group:

- If `THREAD_PRIORITY` is not given, the default priority is 0.
- If `THREAD_PRIORITY` is given, the attribute value must be in the range from -20 (highest priority) to 19 (lowest priority). The priority for system resource groups must be in the range from -20 to 0. The priority for user resource groups must be in the range from 0 to 19. Use of different ranges for system and user groups ensures that user threads never have a higher priority than system threads.

`ENABLE` and `DISABLE` specify that the resource group is initially enabled or disabled. If neither is specified, the group is enabled by default. A disabled group cannot have threads assigned to it.

Examples:

- Create an enabled user group that has a single CPU and the lowest priority:

```
CREATE RESOURCE GROUP rg1
  TYPE = USER
  VCPU = 0
  THREAD_PRIORITY = 19;
```

- Create a disabled system group that has no CPU affinity (can use all CPUs) and the highest priority:

```
CREATE RESOURCE GROUP rg2
  TYPE = SYSTEM
  THREAD_PRIORITY = -20
```

```
  DISABLE;
```

Resource group management is local to the server on which it occurs. `CREATE RESOURCE GROUP` statements are not written to the binary log and are not replicated.

13.7.2.3 DROP RESOURCE GROUP Syntax

```
DROP RESOURCE GROUP group_name [FORCE]
```

`DROP RESOURCE GROUP` is used for resource group management (see Section 8.12.5, “Resource Groups”). This statement drops a resource group. It requires the `RESOURCE_GROUP_ADMIN` privilege.

`group_name` identifies which resource group to drop. If the group does not exist, an error occurs.

The `FORCE` modifier determines statement behavior if the resource group has any threads assigned to it:

- If `FORCE` is not given and any threads are assigned to the group, an error occurs.
- If `FORCE` is given, existing threads in the group are moved to their respective default group (system threads to `SYS_default`, user threads to `USR_default`).

Examples:

- Drop a group, failing if the group contains any threads:

```
DROP RESOURCE GROUP rg1;
```

- Drop a group and move existing threads to the default groups:

```
DROP RESOURCE GROUP rg2 FORCE;
```

Resource group management is local to the server on which it occurs. `DROP RESOURCE GROUP` statements are not written to the binary log and are not replicated.

13.7.2.4 SET RESOURCE GROUP Syntax

```
SET RESOURCE GROUP group_name
    [FOR thread_id [, thread_id] ...]
```

`SET RESOURCE GROUP` is used for resource group management (see Section 8.12.5, “Resource Groups”). This statement assigns threads to a resource group. It requires the `RESOURCE_GROUP_ADMIN` or `RESOURCE_GROUP_USER` privilege.

`group_name` identifies which resource group to be assigned. Any *`thread_id`* values indicate threads to assign to the group. Thread IDs can be determined from the Performance Schema `threads` table. If the resource group or any named thread ID does not exist, an error occurs.

With no `FOR` clause, the statement assigns the current thread for the session to the resource group.

With a `FOR` clause that names thread IDs, the statement assigns those threads to the resource group.

For attempts to assign a system thread to a user resource group or a user thread to a system resource group, a warning occurs.

Examples:

- Assign the current session thread to a group:

```
SET RESOURCE GROUP rg1;
```

- Assign the named threads to a group:

```
SET RESOURCE GROUP rg2 FOR 14, 78, 4;
```

Resource group management is local to the server on which it occurs. `SET RESOURCE GROUP` statements are not written to the binary log and are not replicated.

An alternative to `SET RESOURCE GROUP` is the `RESOURCE_GROUP` optimizer hint, which assigns individual statements to a resource group. See Section 8.9.3, “Optimizer Hints”.

13.7.3 Table Maintenance Statements

13.7.3.1 ANALYZE TABLE Syntax

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] ...

ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name
    UPDATE HISTOGRAM ON col_name [, col_name] ...
        [WITH N BUCKETS]

ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name
    DROP HISTOGRAM ON col_name [, col_name] ...
```

`ANALYZE TABLE` generates table statistics:

- `ANALYZE TABLE` without either `HISTOGRAM` clause performs a key distribution analysis and stores the distribution for the named table or tables. For `MyISAM` tables, `ANALYZE TABLE` for key distribution analysis is equivalent to using `myisamchk --analyze`.
- `ANALYZE TABLE` with the `UPDATE HISTOGRAM` clause generates histogram statistics for the named table columns and stores them in the data dictionary. Only one table name is permitted for this syntax.
- `ANALYZE TABLE` with the `DROP HISTOGRAM` clause removes histogram statistics for the named table columns from the data dictionary. Only one table name is permitted for this syntax.

Note

If the `innodb_read_only` system variable is enabled, `ANALYZE TABLE` may fail because it cannot update statistics tables in the data dictionary, which use `InnoDB`. For `ANALYZE TABLE` operations that update the key distribution, failure may occur even if the operation updates the table itself (for example, if it is a `MyISAM` table). To obtain the updated distribution statistics, set `information_schema_stats_expiry=0`.

This statement requires `SELECT` and `INSERT` privileges for the table.

`ANALYZE TABLE` works with `InnoDB`, `NDB`, and `MyISAM` tables. It does not work with views.

`ANALYZE TABLE` is supported for partitioned tables, and you can use `ALTER TABLE ... ANALYZE PARTITION` to analyze one or more partitions; for more information, see Section 13.1.9, "ALTER TABLE Syntax", and Section 23.3.4, "Maintenance of Partitions".

During the analysis, the table is locked with a read lock for `InnoDB` and `MyISAM`.

By default, the server writes `ANALYZE TABLE` statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

- ANALYZE TABLE Output
- Key Distribution Analysis
- Histogram Statistics Analysis
- Other Considerations

ANALYZE TABLE Output

`ANALYZE TABLE` returns a result set with the columns shown in the following table.

Column	Value
`Table`	The table name
`Op`	`analyze` or `histogram`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

Key Distribution Analysis

`ANALYZE TABLE` without either `HISTOGRAM` clause performs a key distribution analysis and stores the distribution for the table or tables. Any existing histogram statistics remain unaffected.

If the table has not changed since the last key distribution analysis, the table is not analyzed again.

MySQL uses the stored key distribution to decide the order in which tables should be joined for joins on something other than a constant. In addition, key distributions can be used when deciding which indexes to use for a specific table within a query.

For more information on how key distribution analysis works within `InnoDB`, see Section 15.8.10.1, "Configuring Persistent Optimizer Statistics Parameters" and Section 15.8.10.3, "Estimating ANALYZE TABLE Complexity for InnoDB Tables". Also see Section 15.6.1.6, "Limits on InnoDB Tables". In particular, when you enable the `innodb_stats_persistent` option, you must run `ANALYZE TABLE` after loading substantial data into an `InnoDB` table, or creating a new index for one.

To check the stored key distribution cardinality, use the `SHOW INDEX` statement or the `INFORMATION_SCHEMA STATISTICS` table. See Section 13.7.7.22, "SHOW INDEX Syntax", and Section 25.26, "The INFORMATION_SCHEMA STATISTICS Table".

Histogram Statistics Analysis

`ANALYZE TABLE` with the `HISTOGRAM` clauses enables management of histogram statistics for table column values. For information about histogram statistics, see Section 8.9.6, "Optimizer Statistics".

These histogram operations are available:

- `ANALYZE TABLE` with an `UPDATE HISTOGRAM` clause generates histogram statistics for the named table columns and stores them in the data dictionary. Only one table name is permitted for this syntax.

  The optional `WITH N BUCKETS` clauses specifies the number of buckets for the histogram. The value of *`N`* must be an integer in the range from 1 to 1024. If this clause is omitted, the number of buckets is 100.

- `ANALYZE TABLE` with a `DROP HISTOGRAM` clause removes histogram statistics for the named table columns from the data dictionary. Only one table name is permitted for this syntax.

Stored histogram management statements affect only the named columns. Consider these statements:

```
ANALYZE TABLE t UPDATE HISTOGRAM ON c1, c2, c3 WITH 10 BUCKETS;
ANALYZE TABLE t UPDATE HISTOGRAM ON c1, c3 WITH 10 BUCKETS;
ANALYZE TABLE t DROP HISTOGRAM ON c2;
```

The first statement updates the histograms for columns `c1`, `c2`, and `c3`, replacing any existing histograms for those columns. The second statement updates the histograms for `c1` and `c3`, leaving the `c2` histogram unaffected. The third statement removes the histogram for `c2`, leaving those for `c1` and `c3` unaffected.

Histogram generation is not supported for encrypted tables (to avoid exposing data in the statistics) or `TEMPORARY` tables.

Histogram generation applies to columns of all data types except geometry types (spatial data) and `JSON`.

Histograms can be generated for stored and virtual generated columns.

Histograms cannot be generated for columns that are covered by single-column unique indexes.

Histogram management statements attempt to perform as much of the requested operation as possible, and report diagnostic messages for the remainder. For example, if an `UPDATE HISTOGRAM` statement names multiple columns, but some of them do not exist or have an unsupported data type, histograms are generated for the other columns, and messages are produced for the invalid columns.

The `histogram_generation_max_mem_size` system variable controls the maximum amount of memory available for histogram generation. The global and session values may be set at runtime.

Changing the global `histogram_generation_max_mem_size` value requires privileges sufficient to set global system variables. Changing the session `histogram_generation_max_mem_size` value requires privileges sufficient to set restricted session system variables. See Section 5.1.9.1, “System Variable Privileges”.

For information about memory allocations performed for histogram generation, monitor the Performance Schema `memory/sql/histograms` instrument. See Section 26.12.17.10, “Memory Summary Tables”.

Histograms are affected by these DDL statements:

- `DROP TABLE` removes histograms for columns in the dropped table.

- `DROP DATABASE` removes histograms for any table in the dropped database because the statement drops all tables in the database.

- `RENAME TABLE` does not remove histograms. Instead, it renames histograms for the renamed table to be associated with the new table name.

- `ALTER TABLE` statements that remove or modify a column remove histograms for that column.

- `ALTER TABLE ... CONVERT TO CHARACTER SET` removes histograms for character columns because they are affected by the change of character set. Histograms for noncharacter columns remain unaffected.

Other Considerations

`ANALYZE TABLE` clears table statistics from the `INFORMATION_SCHEMA.INNODB_TABLESTATS` table and sets the `STATS_INITIALIZED` column to `Uninitialized`. Statistics are collected again the next time the table is accessed.

13.7.3.2 CHECK TABLE Syntax

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option: {
    FOR UPGRADE
  | QUICK
  | FAST
  | MEDIUM
  | EXTENDED
  | CHANGED
}
```

`CHECK TABLE` checks a table or tables for errors. `CHECK TABLE` can also check views for problems, such as tables that are referenced in the view definition that no longer exist.

To check a table, you must have some privilege for it.

`CHECK TABLE` works for `InnoDB`, `MyISAM`, `ARCHIVE`, and `CSV` tables.

Before running `CHECK TABLE` on `InnoDB` tables, see CHECK TABLE Usage Notes for InnoDB Tables.

`CHECK TABLE` is supported for partitioned tables, and you can use `ALTER TABLE ... CHECK PARTITION` to check one or more partitions; for more information, see Section 13.1.9, “ALTER TABLE Syntax”, and Section 23.3.4, “Maintenance of Partitions”.

`CHECK TABLE` ignores virtual generated columns that are not indexed.

- CHECK TABLE Output
- Checking Version Compatibility
- Checking Data Consistency
- CHECK TABLE Usage Notes for InnoDB Tables
- CHECK TABLE Usage Notes for MyISAM Tables

CHECK TABLE Output

`CHECK TABLE` returns a result set with the columns shown in the following table.

Column	Value
`Table`	The table name
`Op`	Always `check`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

The statement might produce many rows of information for each checked table. The last row has a `Msg_type` value of `status` and the `Msg_text` normally should be `OK`. `Table is already up to date` means that the storage engine for the table indicated that there was no need to check the table.

Checking Version Compatibility

The `FOR UPGRADE` option checks whether the named tables are compatible with the current version of MySQL. With `FOR UPGRADE`, the server checks each table to determine whether there have been any incompatible changes in any of the table's data types or indexes since the table was created. If not, the check succeeds. Otherwise, if there is a possible incompatibility, the server runs a full check on the table (which might take some time).

Incompatibilities might occur because the storage format for a data type has changed or because its sort order has changed. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.

`FOR UPGRADE` discovers these incompatibilities:

- The indexing order for end-space in `TEXT` columns for `InnoDB` and `MyISAM` tables changed between MySQL 4.1 and 5.0.
- The storage method of the new `DECIMAL` data type changed between MySQL 5.0.3 and 5.0.5.
- Changes are sometimes made to character sets or collations that require table indexes to be rebuilt. For details about such changes, see Section 2.11.4, “Changes in MySQL 8.0”. For information about rebuilding tables, see Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”.
- MySQL 8.0 does not support the `YEAR(2)` data type permitted in older versions of MySQL. For tables containing `YEAR(2)` columns, `CHECK TABLE` recommends `REPAIR TABLE`, which converts `YEAR(2)` to `YEAR(4)`.
- Trigger creation time is maintained.
- A table is reported as needing a rebuild if it contains old temporal columns in pre-5.6.4 format (`TIME`, `DATETIME`, and `TIMESTAMP` columns without support for fractional seconds precision) and the `avoid_temporal_upgrade` system variable is disabled. This helps the MySQL upgrade procedure detect and upgrade tables containing old temporal columns. If `avoid_temporal_upgrade` is enabled, `FOR UPGRADE` ignores the old temporal columns present in the table; consequently, the upgrade procedure does not upgrade them.

  To check for tables that contain such temporal columns and need a rebuild, disable `avoid_temporal_upgrade` before executing `CHECK TABLE ... FOR UPGRADE`.
- Warnings are issued for tables that use nonnative partitioning because nonnative partitioning is removed in MySQL 8.0. See Chapter 23, *Partitioning*.

Checking Data Consistency

The following table shows the other check options that can be given. These options are passed to the storage engine, which may use or ignore them.

Type	Meaning
`QUICK`	Do not scan the rows to check for incorrect links. Applies to `InnoDB` and `MyISAM` tables and views.
`FAST`	Check only tables that have not been closed properly. Ignored for `InnoDB`; applies only to `MyISAM` tables and views.

Type	Meaning
CHANGED	Check only tables that have been changed since the last check or that have not been closed properly. Ignored for InnoDB; applies only to MyISAM tables and views.
MEDIUM	Scan rows to verify that deleted links are valid. This also calculates a key checksum for the rows and verifies this with a calculated checksum for the keys. Ignored for InnoDB; applies only to MyISAM tables and views.
EXTENDED	Do a full key lookup for all keys for each row. This ensures that the table is 100% consistent, but takes a long time. Ignored for InnoDB; applies only to MyISAM tables and views.

You can combine check options, as in the following example that does a quick check on the table to determine whether it was closed properly:

```
CHECK TABLE test_table FAST QUICK;
```

Note

If CHECK TABLE finds no problems with a table that is marked as “corrupted” or “not closed properly”, CHECK TABLE may remove the mark.

If a table is corrupted, the problem is most likely in the indexes and not in the data part. All of the preceding check types check the indexes thoroughly and should thus find most errors.

To check a table that you assume is okay, use no check options or the QUICK option. The latter should be used when you are in a hurry and can take the very small risk that QUICK does not find an error in the data file. (In most cases, under normal usage, MySQL should find any error in the data file. If this happens, the table is marked as “corrupted” and cannot be used until it is repaired.)

FAST and CHANGED are mostly intended to be used from a script (for example, to be executed from cron) to check tables periodically. In most cases, FAST is to be preferred over CHANGED. (The only case when it is not preferred is when you suspect that you have found a bug in the MyISAM code.)

EXTENDED is to be used only after you have run a normal check but still get errors from a table when MySQL tries to update a row or find a row by key. This is very unlikely if a normal check has succeeded.

Use of CHECK TABLE ... EXTENDED might influence execution plans generated by the query optimizer.

Some problems reported by CHECK TABLE cannot be corrected automatically:

- Found row where the auto_increment column has the value 0.

  This means that you have a row in the table where the AUTO_INCREMENT index column contains the value 0. (It is possible to create a row where the AUTO_INCREMENT column is 0 by explicitly setting the column to 0 with an UPDATE statement.)

  This is not an error in itself, but could cause trouble if you decide to dump the table and restore it or do an ALTER TABLE on the table. In this case, the AUTO_INCREMENT column changes value according to the rules of AUTO_INCREMENT columns, which could cause problems such as a duplicate-key error.

  To get rid of the warning, execute an UPDATE statement to set the column to some value other than 0.

CHECK TABLE Usage Notes for InnoDB Tables

The following notes apply to InnoDB tables:

- If `CHECK TABLE` encounters a corrupt page, the server exits to prevent error propagation (Bug #10132). If the corruption occurs in a secondary index but table data is readable, running `CHECK TABLE` can still cause a server exit.
- If `CHECK TABLE` encounters a corrupted `DB_TRX_ID` or `DB_ROLL_PTR` field in a clustered index, `CHECK TABLE` can cause `InnoDB` to access an invalid undo log record, resulting in an MVCC-related server exit.
- If `CHECK TABLE` encounters errors in `InnoDB` tables or indexes, it reports an error, and usually marks the index and sometimes marks the table as corrupted, preventing further use of the index or table. Such errors include an incorrect number of entries in a secondary index or incorrect links.
- If `CHECK TABLE` finds an incorrect number of entries in a secondary index, it reports an error but does not cause a server exit or prevent access to the file.
- `CHECK TABLE` surveys the index page structure, then surveys each key entry. It does not validate the key pointer to a clustered record or follow the path for `BLOB` pointers.
- When an `InnoDB` table is stored in its own `.ibd` file, the first 3 pages of the `.ibd` file contain header information rather than table or index data. The `CHECK TABLE` statement does not detect inconsistencies that affect only the header data. To verify the entire contents of an `InnoDB .ibd` file, use the `innochecksum` command.
- When running `CHECK TABLE` on large `InnoDB` tables, other threads may be blocked during `CHECK TABLE` execution. To avoid timeouts, the semaphore wait threshold (600 seconds) is extended by 2 hours (7200 seconds) for `CHECK TABLE` operations. If `InnoDB` detects semaphore waits of 240 seconds or more, it starts printing `InnoDB` monitor output to the error log. If a lock request extends beyond the semaphore wait threshold, `InnoDB` aborts the process. To avoid the possibility of a semaphore wait timeout entirely, run `CHECK TABLE QUICK` instead of `CHECK TABLE`.
- `CHECK TABLE` functionality for `InnoDB SPATIAL` indexes includes an R-tree validity check and a check to ensure that the R-tree row count matches the clustered index.
- `CHECK TABLE` supports secondary indexes on virtual generated columns, which are supported by `InnoDB`.
- As of MySQL 8.0.14, `InnoDB` supports parallel clustered index reads, which can improve `CHECK TABLE` performance. `InnoDB` reads the clustered index twice during a `CHECK TABLE` operation. The second read can be performed in parallel. The `innodb_parallel_read_threads` session variable must be set to a value greater than 1 for parallel clustered index reads to occur. The default value is 4. The actual number of threads used to perform a parallel clustered index read is determined by the `innodb_parallel_read_threads` setting or the number of index subtrees to scan, whichever is smaller.

CHECK TABLE Usage Notes for MyISAM Tables

The following notes apply to `MyISAM` tables:

- `CHECK TABLE` updates key statistics for `MyISAM` tables.
- If `CHECK TABLE` output does not return `OK` or `Table is already up to date`, you should normally run a repair of the table. See Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.
- If none of the `CHECK TABLE` options `QUICK`, `MEDIUM`, or `EXTENDED` are specified, the default check type for dynamic-format `MyISAM` tables is `MEDIUM`. This has the same result as running `myisamchk --medium-check tbl_name` on the table. The default check type also is `MEDIUM` for static-format

MyISAM tables, unless CHANGED or FAST is specified. In that case, the default is QUICK. The row scan is skipped for CHANGED and FAST because the rows are very seldom corrupted.

13.7.3.3 CHECKSUM TABLE Syntax

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [QUICK | EXTENDED]
```

CHECKSUM TABLE reports a checksum for the contents of a table. You can use this statement to verify that the contents are the same before and after a backup, rollback, or other operation that is intended to put the data back to a known state.

This statement requires the SELECT privilege for the table.

This statement is not supported for views. If you run CHECKSUM TABLE against a view, the Checksum value is always NULL, and a warning is returned.

For a nonexistent table, CHECKSUM TABLE returns NULL and generates a warning.

During the checksum operation, the table is locked with a read lock for InnoDB and MyISAM.

Performance Considerations

By default, the entire table is read row by row and the checksum is calculated. For large tables, this could take a long time, thus you would only perform this operation occasionally. This row-by-row calculation is what you get with the EXTENDED clause, with InnoDB and all other storage engines other than MyISAM, and with MyISAM tables not created with the CHECKSUM=1 clause.

For MyISAM tables created with the CHECKSUM=1 clause, CHECKSUM TABLE or CHECKSUM TABLE ... QUICK returns the “live” table checksum that can be returned very fast. If the table does not meet all these conditions, the QUICK method returns NULL. The QUICK method is not supported with InnoDB tables. See Section 13.1.20, “CREATE TABLE Syntax” for the syntax of the CHECKSUM clause.

The checksum value depends on the table row format. If the row format changes, the checksum also changes. For example, the storage format for temporal types such as TIME, DATETIME, and TIMESTAMP changed in MySQL 5.6 prior to MySQL 5.6.5, so if a 5.5 table is upgraded to MySQL 5.6, the checksum value may change.

Important

If the checksums for two tables are different, then it is almost certain that the tables are different in some way. However, because the hashing function used by CHECKSUM TABLE is not guaranteed to be collision-free, there is a slight chance that two tables which are not identical can produce the same checksum.

13.7.3.4 OPTIMIZE TABLE Syntax

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] ...
```

OPTIMIZE TABLE reorganizes the physical storage of table data and associated index data, to reduce storage space and improve I/O efficiency when accessing the table. The exact changes made to each table depend on the storage engine used by that table.

Use OPTIMIZE TABLE in these cases, depending on the type of table:

- After doing substantial insert, update, or delete operations on an `InnoDB` table that has its own .ibd file because it was created with the `innodb_file_per_table` option enabled. The table and indexes are reorganized, and disk space can be reclaimed for use by the operating system.
- After doing substantial insert, update, or delete operations on columns that are part of a `FULLTEXT` index in an `InnoDB` table. Set the configuration option `innodb_optimize_fulltext_only=1` first. To keep the index maintenance period to a reasonable time, set the `innodb_ft_num_word_optimize` option to specify how many words to update in the search index, and run a sequence of `OPTIMIZE TABLE` statements until the search index is fully updated.
- After deleting a large part of a `MyISAM` or `ARCHIVE` table, or making many changes to a `MyISAM` or `ARCHIVE` table with variable-length rows (tables that have `VARCHAR`, `VARBINARY`, `BLOB`, or `TEXT` columns). Deleted rows are maintained in a linked list and subsequent `INSERT` operations reuse old row positions. You can use `OPTIMIZE TABLE` to reclaim the unused space and to defragment the data file. After extensive changes to a table, this statement may also improve performance of statements that use the table, sometimes significantly.

This statement requires `SELECT` and `INSERT` privileges for the table.

`OPTIMIZE TABLE` works for `InnoDB`, `MyISAM`, and `ARCHIVE` tables. `OPTIMIZE TABLE` is also supported for dynamic columns of in-memory `NDB` tables. It does not work for fixed-width columns of in-memory tables, nor does it work for Disk Data tables. The performance of `OPTIMIZE` on NDB Cluster tables can be tuned using `--ndb-optimization-delay`, which controls the length of time to wait between processing batches of rows by `OPTIMIZE TABLE`. For more information, see Section 22.1.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 8.0”.

For NDB Cluster tables, `OPTIMIZE TABLE` can be interrupted by (for example) killing the SQL thread performing the `OPTIMIZE` operation.

By default, `OPTIMIZE TABLE` does *not* work for tables created using any other storage engine and returns a result indicating this lack of support. You can make `OPTIMIZE TABLE` work for other storage engines by starting `mysqld` with the `--skip-new` option. In this case, `OPTIMIZE TABLE` is just mapped to `ALTER TABLE`.

This statement does not work with views.

`OPTIMIZE TABLE` is supported for partitioned tables. For information about using this statement with partitioned tables and table partitions, see Section 23.3.4, “Maintenance of Partitions”.

By default, the server writes `OPTIMIZE TABLE` statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

- OPTIMIZE TABLE Output
- InnoDB Details
- MyISAM Details
- Other Considerations

OPTIMIZE TABLE Output

`OPTIMIZE TABLE` returns a result set with the columns shown in the following table.

Column	Value
`Table`	The table name

Column	Value
`Op`	Always `optimize`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

`OPTIMIZE TABLE` table catches and throws any errors that occur while copying table statistics from the old file to the newly created file. For example. if the user ID of the owner of the `.MYD` or `.MYI` file is different from the user ID of the `mysqld` process, `OPTIMIZE TABLE` generates a "cannot change ownership of the file" error unless `mysqld` is started by the `root` user.

InnoDB Details

For `InnoDB` tables, `OPTIMIZE TABLE` is mapped to `ALTER TABLE ... FORCE`, which rebuilds the table to update index statistics and free unused space in the clustered index. This is displayed in the output of `OPTIMIZE TABLE` when you run it on an `InnoDB` table, as shown here:

```
mysql> OPTIMIZE TABLE foo;
+----------+----------+----------+-------------------------------------------------------------------+
| Table    | Op       | Msg_type | Msg_text                                                          |
+----------+----------+----------+-------------------------------------------------------------------+
| test.foo | optimize | note     | Table does not support optimize, doing recreate + analyze instead |
| test.foo | optimize | status   | OK                                                                |
+----------+----------+----------+-------------------------------------------------------------------+
```

`OPTIMIZE TABLE` uses online DDL for regular and partitioned `InnoDB` tables, which reduces downtime for concurrent DML operations. The table rebuild triggered by `OPTIMIZE TABLE` and performed under the cover by `ALTER TABLE ... FORCE` is completed in place. An exclusive table lock is only taken briefly during the prepare phase and the commit phase of the operation. During the prepare phase, metadata is updated and an intermediate table is created. During the commit phase, table metadata changes are committed.

`OPTIMIZE TABLE` rebuilds the table using the table copy method under the following conditions:

- When the `old_alter_table` system variable is enabled.
- When the `mysqld --skip-new` option is enabled.

`OPTIMIZE TABLE` using online DDL is not supported for `InnoDB` tables that contain `FULLTEXT` indexes. The table copy method is used instead.

`InnoDB` stores data using a page-allocation method and does not suffer from fragmentation in the same way that legacy storage engines (such as `MyISAM`) will. When considering whether or not to run optimize, consider the workload of transactions that your server will process:

- Some level of fragmentation is expected. `InnoDB` only fills pages 93% full, to leave room for updates without having to split pages.
- Delete operations might leave gaps that leave pages less filled than desired, which could make it worthwhile to optimize the table.
- Updates to rows usually rewrite the data within the same page, depending on the data type and row format, when sufficient space is available. See Section 15.9.1.5, “How Compression Works for InnoDB Tables” and Section 15.10, “InnoDB Row Formats”.
- High-concurrency workloads might leave gaps in indexes over time, as `InnoDB` retains multiple versions of the same data due through its MVCC mechanism. See Section 15.3, “InnoDB Multi-Versioning”.

MyISAM Details

For `MyISAM` tables, `OPTIMIZE TABLE` works as follows:

1. If the table has deleted or split rows, repair the table.
2. If the index pages are not sorted, sort them.
3. If the table's statistics are not up to date (and the repair could not be accomplished by sorting the index), update them.

Other Considerations

`OPTIMIZE TABLE` is performed online for regular and partitioned `InnoDB` tables. Otherwise, MySQL locks the table during the time `OPTIMIZE TABLE` is running.

`OPTIMIZE TABLE` does not sort R-tree indexes, such as spatial indexes on `POINT` columns. (Bug #23578)

13.7.3.5 REPAIR TABLE Syntax

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] ...
    [QUICK] [EXTENDED] [USE_FRM]
```

`REPAIR TABLE` repairs a possibly corrupted table, for certain storage engines only.

This statement requires `SELECT` and `INSERT` privileges for the table.

Although normally you should never have to run `REPAIR TABLE`, if disaster strikes, this statement is very likely to get back all your data from a `MyISAM` table. If your tables become corrupted often, try to find the reason for it, to eliminate the need to use `REPAIR TABLE`. See Section B.4.3.3, “What to Do If MySQL Keeps Crashing”, and Section 16.2.4, “MyISAM Table Problems”.

`REPAIR TABLE` checks the table to see whether an upgrade is required. If so, it performs the upgrade, following the same rules as `CHECK TABLE ... FOR UPGRADE`. See Section 13.7.3.2, “CHECK TABLE Syntax”, for more information.

Important

- Make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors. See Chapter 7, *Backup and Recovery*.
- If the server crashes during a `REPAIR TABLE` operation, it is essential after restarting it that you immediately execute another `REPAIR TABLE` statement for the table before performing any other operations on it. In the worst case, you might have a new clean index file without information about the data file, and then the next operation you perform could overwrite the data file. This is an unlikely but possible scenario that underscores the value of making a backup first.
- In the event that a table on the master becomes corrupted and you run `REPAIR TABLE` on it, any resulting changes to the original table are *not* propagated to slaves.

- REPAIR TABLE Storage Engine and Partitioning Support
- REPAIR TABLE Options
- REPAIR TABLE Output
- Table Repair Considerations

REPAIR TABLE Storage Engine and Partitioning Support

`REPAIR TABLE` works for `MyISAM`, `ARCHIVE`, and `CSV` tables. For `MyISAM` tables, it has the same effect as `myisamchk --recover *tbl_name*` by default. This statement does not work with views.

`REPAIR TABLE` is supported for partitioned tables. However, the `USE_FRM` option cannot be used with this statement on a partitioned table.

You can use `ALTER TABLE ... REPAIR PARTITION` to repair one or more partitions; for more information, see Section 13.1.9, “ALTER TABLE Syntax”, and Section 23.3.4, “Maintenance of Partitions”.

REPAIR TABLE Options

- `NO_WRITE_TO_BINLOG` or `LOCAL`

  By default, the server writes `REPAIR TABLE` statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

- `QUICK`

  If you use the `QUICK` option, `REPAIR TABLE` tries to repair only the index file, and not the data file. This type of repair is like that done by `myisamchk --recover --quick`.

- `EXTENDED`

  If you use the `EXTENDED` option, MySQL creates the index row by row instead of creating one index at a time with sorting. This type of repair is like that done by `myisamchk --safe-recover`.

- `USE_FRM`

  The `USE_FRM` option is available for use if the `.MYI` index file is missing or if its header is corrupted. This option tells MySQL not to trust the information in the `.MYI` file header and to re-create it using information from the data dictionary. This kind of repair cannot be done with `myisamchk`.

Caution

Use the `USE_FRM` option *only* if you cannot use regular `REPAIR` modes. Telling the server to ignore the `.MYI` file makes important table metadata stored in the `.MYI` unavailable to the repair process, which can have deleterious consequences:

- The current `AUTO_INCREMENT` value is lost.
- The link to deleted records in the table is lost, which means that free space for deleted records will remain unoccupied thereafter.
- The `.MYI` header indicates whether the table is compressed. If the server ignores this information, it cannot tell that a table is compressed and repair

> can cause change or loss of table contents. This means that `USE_FRM` should not be used with compressed tables. That should not be necessary, anyway: Compressed tables are read only, so they should not become corrupt.
>
> If you use `USE_FRM` for a table that was created by a different version of the MySQL server than the one you are currently running, `REPAIR TABLE` does not attempt to repair the table. In this case, the result set returned by `REPAIR TABLE` contains a line with a `Msg_type` value of `error` and a `Msg_text` value of `Failed repairing incompatible .FRM file`.
>
> If `USE_FRM` is used, `REPAIR TABLE` does not check the table to see whether an upgrade is required.

REPAIR TABLE Output

`REPAIR TABLE` returns a result set with the columns shown in the following table.

Column	Value
`Table`	The table name
`Op`	Always `repair`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

The `REPAIR TABLE` statement might produce many rows of information for each repaired table. The last row has a `Msg_type` value of `status` and `Msg_test` normally should be `OK`. For a `MyISAM` table, if you do not get `OK`, you should try repairing it with `myisamchk --safe-recover`. (`REPAIR TABLE` does not implement all the options of `myisamchk`. With `myisamchk --safe-recover`, you can also use options that `REPAIR TABLE` does not support, such as `--max-record-length`.)

`REPAIR TABLE` table catches and throws any errors that occur while copying table statistics from the old corrupted file to the newly created file. For example. if the user ID of the owner of the `.MYD` or `.MYI` file is different from the user ID of the `mysqld` process, `REPAIR TABLE` generates a "cannot change ownership of the file" error unless `mysqld` is started by the `root` user.

Table Repair Considerations

`REPAIR TABLE` upgrades a table if it contains old temporal columns in pre-5.6.4 format (`TIME`, `DATETIME`, and `TIMESTAMP` columns without support for fractional seconds precision) and the `avoid_temporal_upgrade` system variable is disabled. If `avoid_temporal_upgrade` is enabled, `REPAIR TABLE` ignores the old temporal columns present in the table and does not upgrade them.

To upgrade tables that contain such temporal columns, disable `avoid_temporal_upgrade` before executing `REPAIR TABLE`.

You may be able to increase `REPAIR TABLE` performance by setting certain system variables. See Section 8.6.3, “Optimizing REPAIR TABLE Statements”.

13.7.4 Component, Plugin, and User-Defined Function Statements

13.7.4.1 CREATE FUNCTION Syntax for User-Defined Functions

```
CREATE [AGGREGATE] FUNCTION function_name
```

```
    RETURNS {STRING|INTEGER|REAL|DECIMAL}
    SONAME shared_library_name
```

A user-defined function (UDF) is a way to extend MySQL with a new function that works like a native (built-in) MySQL function such as `ABS()` or `CONCAT()`.

`function_name` is the name that should be used in SQL statements to invoke the function. The `RETURNS` clause indicates the type of the function's return value. `DECIMAL` is a legal value after `RETURNS`, but currently `DECIMAL` functions return string values and should be written like `STRING` functions.

`shared_library_name` is the base name of the shared library file that contains the code that implements the function. The file must be located in the plugin directory. This directory is given by the value of the `plugin_dir` system variable. For more information, see Section 5.7.1, “Installing and Uninstalling User-Defined Functions”.

To create a function, you must have the `INSERT` privilege for the `mysql` system database. This is necessary because `CREATE FUNCTION` adds a row to the `mysql.func` system table that records the function's name, type, and shared library name.

UDFs registered using `CREATE FUNCTION` are listed in the Performance Schema `user_defined_functions` table; see Section 26.12.18.6, “The user_defined_functions Table”.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable.

For instructions on writing user-defined functions, see Section 29.4.2, “Adding a New User-Defined Function”. For the UDF mechanism to work, functions must be written in C or C++ (or another language that can use C calling conventions), your operating system must support dynamic loading and you must have compiled `mysqld` dynamically (not statically).

An `AGGREGATE` function works exactly like a native MySQL aggregate (summary) function such as `SUM` or `COUNT()`.

Note

To upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may crash.

13.7.4.2 DROP FUNCTION Syntax

```
DROP FUNCTION function_name
```

This statement drops the user-defined function (UDF) named *`function_name`*.

To drop a function, you must have the `DELETE` privilege for the `mysql` system database. This is because `DROP FUNCTION` removes a row from the `mysql.func` system table that records the function's name, type, and shared library name.

Note

To upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION`

statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may crash.

`DROP FUNCTION` is also used to drop stored functions (see Section 13.1.29, “DROP PROCEDURE and DROP FUNCTION Syntax”).

13.7.4.3 INSTALL COMPONENT Syntax

```
INSTALL COMPONENT component_name [, component_name ] ...
```

This statement installs one or more server components, which become active immediately. A component provides services that are available to the server and other components; see Section 5.5, “MySQL Server Components”. `INSTALL COMPONENT` requires the `INSERT` privilege for the `mysql.component` system table.

Example:

```
INSTALL COMPONENT 'file://component1', 'file://component2';
```

Component names are URNs that begin with `file://` and indicate the base name of the file that implements the component, located in the directory named by the `plugin_dir` system variable. Component names do not include any platform-dependent file name suffix such as `.so` or `.dll`. (These naming details are subject to change because component name interpretation is itself performed by a service and the component infrastructure makes it possible to replace the default service implementation with alternative implementations.)

If any error occurs, the statement fails and has no effect. For example, this happens if a component name is erroneous, a named component does not exist or is already installed, or component initialization fails.

A loader service handles component loading, which includes adding installed components to the `mysql.component` system table that serves as a registry. For subsequent server restarts, any components listed in `mysql.component` are loaded by the loader service during the startup sequence. This occurs even if the server is started with the `--skip-grant-tables` option.

If a component depends on services not present in the registry and you attempt to install the component without also installing the component or components that provide the services on which it depends, an error occurs:

```
ERROR 3527 (HY000): Cannot satisfy dependency for service 'component_a'
required by component 'component_b'.
```

To avoid this problem, either install all components in the same statement, or install the dependent component after installing any components on which it depends.

13.7.4.4 INSTALL PLUGIN Syntax

```
INSTALL PLUGIN plugin_name SONAME 'shared_library_name'
```

This statement installs a server plugin. It requires the `INSERT` privilege for the `mysql.plugin` system table.

`plugin_name` is the name of the plugin as defined in the plugin descriptor structure contained in the library file (see Section 29.2.4.2, “Plugin Data Structures”). Plugin names are not case-sensitive. For maximal compatibility, plugin names should be limited to ASCII letters, digits, and underscore because they are used in C source files, shell command lines, M4 and Bourne shell scripts, and SQL environments.

shared_library_name is the name of the shared library that contains the plugin code. The name includes the file name extension (for example, libmyplugin.so, libmyplugin.dll, or libmyplugin.dylib).

The shared library must be located in the plugin directory (the directory named by the plugin_dir system variable). The library must be in the plugin directory itself, not in a subdirectory. By default, plugin_dir is the plugin directory under the directory named by the pkglibdir configuration variable, but it can be changed by setting the value of plugin_dir at server startup. For example, set its value in a my.cnf file:

```
[mysqld]
plugin_dir=/path/to/plugin/directory
```

If the value of plugin_dir is a relative path name, it is taken to be relative to the MySQL base directory (the value of the basedir system variable).

INSTALL PLUGIN loads and initializes the plugin code to make the plugin available for use. A plugin is initialized by executing its initialization function, which handles any setup that the plugin must perform before it can be used. When the server shuts down, it executes the deinitialization function for each plugin that is loaded so that the plugin has a chance to perform any final cleanup.

INSTALL PLUGIN also registers the plugin by adding a line that indicates the plugin name and library file name to the mysql.plugin system table. At server startup, the server loads and initializes any plugin that is listed in mysql.plugin. This means that a plugin is installed with INSTALL PLUGIN only once, not every time the server starts. Plugin loading at startup does not occur if the server is started with the --skip-grant-tables option.

A plugin library can contain multiple plugins. For each of them to be installed, use a separate INSTALL PLUGIN statement. Each statement names a different plugin, but all of them specify the same library name.

INSTALL PLUGIN causes the server to read option (my.cnf) files just as during server startup. This enables the plugin to pick up any relevant options from those files. It is possible to add plugin options to an option file even before loading a plugin (if the loose prefix is used). It is also possible to uninstall a plugin, edit my.cnf, and install the plugin again. Restarting the plugin this way enables it to the new option values without a server restart.

For options that control individual plugin loading at server startup, see Section 5.6.1, “Installing and Uninstalling Plugins”. If you need to load plugins for a single server startup when the --skip-grant-tables option is given (which tells the server not to read system tables), use the --plugin-load option. See Section 5.1.7, “Server Command Options”.

To remove a plugin, use the UNINSTALL PLUGIN statement.

For additional information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To see what plugins are installed, use the SHOW PLUGINS statement or query the INFORMATION_SCHEMA the PLUGINS table.

If you recompile a plugin library and need to reinstall it, you can use either of the following methods:

- Use UNINSTALL PLUGIN to uninstall all plugins in the library, install the new plugin library file in the plugin directory, and then use INSTALL PLUGIN to install all plugins in the library. This procedure has the advantage that it can be used without stopping the server. However, if the plugin library contains many plugins, you must issue many INSTALL PLUGIN and UNINSTALL PLUGIN statements.
- Stop the server, install the new plugin library file in the plugin directory, and restart the server.

13.7.4.5 UNINSTALL COMPONENT Syntax

```
UNINSTALL COMPONENT component_name [, component_name ] ...
```

This statement deactivates and uninstalls one or more server components. A component provides services that are available to the server and other components; see Section 5.5, “MySQL Server Components”. `UNINSTALL COMPONENT` is the complement of `INSTALL COMPONENT`. It requires the `DELETE` privilege for the `mysql.component` system table.

Example:

```
UNINSTALL COMPONENT 'file://component1', 'file://component2';
```

For information about component naming, see Section 13.7.4.3, “INSTALL COMPONENT Syntax”.

If any error occurs, the statement fails and has no effect. For example, this happens if a component name is erroneous, a named component is not installed, or cannot be uninstalled because other installed components depend on it.

A loader service handles component unloading, which includes removing uninstalled components from the `mysql.component` system table that serves as a registry. As a result, unloaded components are not loaded during the startup sequence for subsequent server restarts.

13.7.4.6 UNINSTALL PLUGIN Syntax

```
UNINSTALL PLUGIN plugin_name
```

This statement removes an installed server plugin. It requires the `DELETE` privilege for the `mysql.plugin` system table. `UNINSTALL PLUGIN` is the complement of `INSTALL PLUGIN`.

`plugin_name` must be the name of some plugin that is listed in the `mysql.plugin` table. The server executes the plugin's deinitialization function and removes the row for the plugin from the `mysql.plugin` system table, so that subsequent server restarts will not load and initialize the plugin. `UNINSTALL PLUGIN` does not remove the plugin's shared library file.

You cannot uninstall a plugin if any table that uses it is open.

Plugin removal has implications for the use of associated tables. For example, if a full-text parser plugin is associated with a `FULLTEXT` index on the table, uninstalling the plugin makes the table unusable. Any attempt to access the table results in an error. The table cannot even be opened, so you cannot drop an index for which the plugin is used. This means that uninstalling a plugin is something to do with care unless you do not care about the table contents. If you are uninstalling a plugin with no intention of reinstalling it later and you care about the table contents, you should dump the table with `mysqldump` and remove the `WITH PARSER` clause from the dumped `CREATE TABLE` statement so that you can reload the table later. If you do not care about the table, `DROP TABLE` can be used even if any plugins associated with the table are missing.

For additional information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

13.7.5 CLONE Syntax

```
CLONE clone_action

clone_action: {
    LOCAL DATA DIRECTORY [=] 'clone_dir';
```

```
  | INSTANCE FROM 'user'@'host':port
    IDENTIFIED BY 'password'
    [DATA DIRECTORY [=] 'clone_dir']
    [REQUIRE [NO] SSL]
}
```

The `CLONE` statement is used to clone data locally or from a remote MySQL server instance. To use `CLONE` syntax, the clone plugin must be installed. See Section 5.6.7, “The Clone Plugin”.

`CLONE LOCAL DATA DIRECTORY` syntax clones data from the local MySQL data directory to a directory on the same server or node where the MySQL server instance runs. The `'clone_dir'` directory is the full path of the local directory that data is cloned to. An absolute path is required. The specified directory must not exist, but the specified path must be an existent path. The MySQL server requires the necessary write access to create the specified directory. For more information, see Section 5.6.7.2, “Cloning Data Locally”.

`CLONE INSTANCE` syntax clones data from a remote MySQL server instance (the donor) and transfers it to the MySQL instance where the cloning operation was initiated (the recipient).

- *`user`* is the clone user on the donor MySQL server instance.
- *`host`* is the `hostname` address of the donor MySQL server instance. Internet Protocol version 6 (IPv6) address format is not supported. An alias to the IPv6 address can be used instead. An IPv4 address can be used as is.
- *`port`* is the `port` number of the donor MySQL server instance. (The X Protocol port specified by `mysqlx_port` is not supported. Connecting to the donor MySQL server instance through MySQL Router is also not supported.)
- `IDENTIFIED BY '`*`password`*`'` specifies the password of the clone user on the donor MySQL server instance.
- `DATA DIRECTORY [=] '`*`clone_dir`*`'` is an optional clause used to specify a directory on the recipient for the data you are cloning. Use this option if you do not want to remove existing data in the recipient data directory. An absolute path is required, and the directory must not exist. The MySQL server must have the necessary write access to create the directory.

  When the optional `DATA DIRECTORY [=] '`*`clone_dir`*`'` clause is not used, a cloning operation removes existing data in the recipient data directory, replaces it with the cloned data, and automatically restarts the server afterward.
- `[REQUIRE [NO] SSL]` explicitly specifies whether an encrypted connection is to be used or not when transferring cloned data over the network. An error is returned if the explicit specification cannot be satisfied. If an SSL clause is not specified, clone attempts to establish an encrypted connection by default, falling back to an insecure connection if the secure connection attempt fails. A secure connection is required when cloning encrypted data regardless of whether this clause is specified. For more information, see Configuring an Encrypted Connection for Cloning.

For additional information about cloning data from a remote MySQL server instance, see Section 5.6.7.3, “Cloning Remote Data”.

13.7.6 SET Syntax

The `SET` statement has several forms. Descriptions for those forms that are not associated with a specific server capability appear in subsections of this section:

- `SET` *`var_name = value`* enables you to assign values to variables that affect the operation of the server or clients. See Section 13.7.6.1, “SET Syntax for Variable Assignment”.

- `SET CHARACTER SET` and `SET NAMES` assign values to character set and collation variables associated with the current connection to the server. See Section 13.7.6.2, “SET CHARACTER SET Syntax”, and Section 13.7.6.3, “SET NAMES Syntax”.

Descriptions for the other forms appear elsewhere, grouped with other statements related to the capability they help implement:

- `SET DEFAULT ROLE` and `SET ROLE` set the default role and current role for user accounts. See Section 13.7.1.9, “SET DEFAULT ROLE Syntax”, and Section 13.7.1.11, “SET ROLE Syntax”.
- `SET PASSWORD` assigns account passwords. See Section 13.7.1.10, “SET PASSWORD Syntax”.
- `SET RESOURCE GROUP` assigns threads to a resource group. See Section 13.7.2.4, “SET RESOURCE GROUP Syntax”.
- `SET TRANSACTION ISOLATION LEVEL` sets the isolation level for transaction processing. See Section 13.3.7, “SET TRANSACTION Syntax”.

13.7.6.1 SET Syntax for Variable Assignment

```
SET variable = expr [, variable = expr] ...

variable: {
    user_var_name
  | param_name
  | local_var_name
  | {GLOBAL | @@GLOBAL.} system_var_name
  | {PERSIST | @@PERSIST.} system_var_name
  | {PERSIST_ONLY | @@PERSIST_ONLY.} system_var_name
  | [SESSION | @@SESSION. | @@] system_var_name
}
```

`SET` syntax for variable assignment enables you to assign values to different types of variables that affect the operation of the server or clients:

- User-defined variables. See Section 9.4, “User-Defined Variables”.
- Stored procedure and function parameters, and stored program local variables. See Section 13.6.4, “Variables in Stored Programs”.
- System variables. See Section 5.1.8, “Server System Variables”. System variables also can be set at server startup, as described in Section 5.1.9, “Using System Variables”.

A `SET` statement that assigns variable values is not written to the binary log, so in replication scenarios it affects only the host on which you execute it. To affect all replication hosts, execute the statement on each host.

The following sections describe `SET` syntax for setting variables. They use the `=` assignment operator, but the `:=` assignment operator is also permitted for this purpose.

- User-Defined Variable Assignment
- Parameter and Local Variable Assignment
- System Variable Assignment
- SET Error Handling
- Multiple Variable Assignment

- System Variable References in Expressions

User-Defined Variable Assignment

User-defined variables are created locally within a session and exist only within the context of that session; see Section 9.4, “User-Defined Variables”.

A user-defined variable is written as *`@var_name`* and is assigned an expression value as follows:

```
SET @var_name = expr;
```

Examples:

```
SET @name = 43;
SET @total_tax = (SELECT SUM(tax) FROM taxable_transactions);
```

As demonstrated by those statements, *`expr`* can range from simple (a literal value) to more complex (the value returned by a scalar subquery).

The Performance Schema `user_variables_by_thread` table contains information about user-defined variables. See Section 26.12.10, “Performance Schema User-Defined Variable Tables”.

Parameter and Local Variable Assignment

`SET` applies to parameters and local variables in the context of the stored object within which they are defined. The following procedure uses the `increment` procedure parameter and `counter` local variable:

```
CREATE PROCEDURE p(increment INT)
BEGIN
  DECLARE counter INT DEFAULT 0;
  WHILE counter < 10 DO
    -- ... do work ...
    SET counter = counter + increment;
  END WHILE;
END;
```

System Variable Assignment

The MySQL server maintains system variables that configure its operation. A system variable can have a global value that affects server operation as a whole, a session value that affects the current session, or both. Many system variables are dynamic and can be changed at runtime using the `SET` statement to affect operation of the current server instance. `SET` can also be used to persist certain system variables to the `mysqld-auto.cnf` file in the data directory, to affect server operation for subsequent startups.

If you change a session system variable, the value remains in effect within your session until you change the variable to a different value or the session ends. The change has no effect on other sessions.

If you change a global system variable, the value is remembered and used to initialize the session value for new sessions until you change the variable to a different value or the server exits. The change is visible to any client that accesses the global value. However, the change affects the corresponding session value only for clients that connect after the change. The global variable change does not affect the session value for any current client sessions (not even the session within which the global value change occurs).

To make a global system variable setting permanent so that it applies across server restarts, you can persist it to the `mysqld-auto.cnf` file in the data directory. It is also possible to make persistent configuration changes by manually modifying a `my.cnf` option file, but that is more cumbersome, and an error in a manually entered setting might not be discovered until much later. `SET` statements that persist

system variables are more convenient and avoid the possibility of malformed settings because settings with syntax errors do not succeed and do not change server configuration. For more information about persisting system variables and the `mysqld-auto.cnf` file, see Section 5.1.9.3, “Persisted System Variables”.

Note

Setting or persisting a global system variable value always requires special privileges. Setting a session system variable value normally requires no special privileges and can be done by any user, although there are exceptions. For more information, see Section 5.1.9.1, “System Variable Privileges”.

The following discussion describes the syntax options for setting and persisting system variables:

- To assign a value to a global system variable, precede the variable name by the `GLOBAL` keyword or the `@@GLOBAL.` qualifier:

```
SET GLOBAL max_connections = 1000;
SET @@GLOBAL.max_connections = 1000;
```

- To assign a value to a session system variable, precede the variable name by the `SESSION` or `LOCAL` keyword, by the `@@SESSION.`, `@@LOCAL.`, or `@@` qualifier, or by no keyword or no modifier at all:

```
SET SESSION sql_mode = 'TRADITIONAL';
SET LOCAL sql_mode = 'TRADITIONAL';
SET @@SESSION.sql_mode = 'TRADITIONAL';
SET @@LOCAL.sql_mode = 'TRADITIONAL';
SET @@sql_mode = 'TRADITIONAL';
SET sql_mode = 'TRADITIONAL';
```

  A client can change its own session variables, but not those of any other client.

- To persist a global system variable to the `mysqld-auto.cnf` option file in the data directory, precede the variable name by the `PERSIST` keyword or the `@@PERSIST.` qualifier:

```
SET PERSIST max_connections = 1000;
SET @@PERSIST.max_connections = 1000;
```

  This `SET` syntax enables you to make configuration changes at runtime that also persist across server restarts. Like `SET GLOBAL`, `SET PERSIST` sets the global variable runtime value, but also writes the variable setting to the `mysqld-auto.cnf` file (replacing any existing variable setting if there is one).

- To persist a global system variable to the `mysqld-auto.cnf` file without setting the global variable runtime value, precede the variable name by the `PERSIST_ONLY` keyword or the `@@PERSIST_ONLY.` qualifier:

```
SET PERSIST_ONLY back_log = 100;
SET @@PERSIST_ONLY.back_log = 100;
```

  Like `PERSIST`, `PERSIST_ONLY` writes the variable setting to `mysqld-auto.cnf`. However, unlike `PERSIST`, `PERSIST_ONLY` does not modify the global variable runtime value. This makes `PERSIST_ONLY` suitable for configuring read-only system variables that can be set only at server startup.

To set a global system variable value to the compiled-in MySQL default value or a session system variable to the current corresponding global value, set the variable to the value `DEFAULT`. For example, the

following two statements are identical in setting the session value of `max_join_size` to the current global value:

```
SET @@SESSION.max_join_size = DEFAULT;
SET @@SESSION.max_join_size = @@GLOBAL.max_join_size;
```

Using `SET` to persist a global system variable to a value of `DEFAULT` or to its literal default value assigns the variable its default value and adds a setting for the variable to `mysqld-auto.cnf`. To remove the variable from the file, use `RESET PERSIST`.

Some system variables cannot be persisted or are persist-restricted. See Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”.

A system variable implemented by a plugin can be persisted if the plugin is installed when the `SET` statement is executed. Assignment of the persisted plugin variable takes effect for subsequent server restarts if the plugin is still installed. If the plugin is no longer installed, the plugin variable will not exist when the server reads the `mysqld-auto.cnf` file. In this case, the server writes a warning to the error log and continues:

```
currently unknown variable 'var_name'
was read from the persisted config file
```

To display system variable names and values:

- Use the `SHOW VARIABLES` statement; see Section 13.7.7.39, “SHOW VARIABLES Syntax”.
- Several Performance Schema tables provide system variable information. See Section 26.12.13, “Performance Schema System Variable Tables”.
- The Performance Schema `variables_info` table contains information showing when and by which user each system variable was most recently set. See Section 26.12.13.2, “Performance Schema variables_info Table”.
- The Performance Schema `persisted_variables` table provides an SQL interface to the `mysqld-auto.cnf` file, enabling its contents to be inspected at runtime using `SELECT` statements. See Section 26.12.13.1, “Performance Schema persisted_variables Table”.

SET Error Handling

If any variable assignment in a `SET` statement fails, the entire statement fails and no variables are changed, nor is the `mysqld-auto.cnf` file changed.

`SET` produces an error under the circumstances described here. Most of the examples show `SET` statements that use keyword syntax (for example, `GLOBAL` or `SESSION`), but the principles are also true for statements that use the corresponding modifiers (for example, `@@GLOBAL.` or `@@SESSION.`).

- Use of `SET` (any variant) to set a read-only variable:

```
mysql> SET GLOBAL version = 'abc';
ERROR 1238 (HY000): Variable 'version' is a read only variable
```

- Use of `GLOBAL`, `PERSIST`, or `PERSIST_ONLY` to set a variable that has only a session value:

```
mysql> SET GLOBAL sql_log_bin = ON;
ERROR 1228 (HY000): Variable 'sql_log_bin' is a SESSION
variable and can't be used with SET GLOBAL
```

- Use of SESSION to set a variable that has only a global value:

```
mysql> SET SESSION max_connections = 1000;
ERROR 1229 (HY000): Variable 'max_connections' is a
GLOBAL variable and should be set with SET GLOBAL
```

- Omission of GLOBAL, PERSIST, or PERSIST_ONLY to set a variable that has only a global value:

```
mysql> SET max_connections = 1000;
ERROR 1229 (HY000): Variable 'max_connections' is a
GLOBAL variable and should be set with SET GLOBAL
```

- Use of PERSIST or PERSIST_ONLY to set a variable that cannot be persisted:

```
mysql> SET PERSIST port = 3307;
ERROR 1238 (HY000): Variable 'port' is a read only variable
mysql> SET PERSIST_ONLY port = 3307;
ERROR 1238 (HY000): Variable 'port' is a non persistent read only variable
```

- The @@GLOBAL., @@PERSIST., @@PERSIST_ONLY., @@SESSION., and @@ modifiers apply only to system variables. An error occurs for attempts to apply them to user-defined variables, stored procedure or function parameters, or stored program local variables.
- Not all system variables can be set to DEFAULT. In such cases, assigning DEFAULT results in an error.
- An error occurs for attempts to assign DEFAULT to user-defined variables, stored procedure or function parameters, or stored program local variables.

Multiple Variable Assignment

A SET statement can contain multiple variable assignments, separated by commas. This statement assigns values to a user-defined variable and a system variable:

```
SET @x = 1, SESSION sql_mode = '';
```

If you set multiple system variables in a single statement, the most recent GLOBAL, PERSIST, PERSIST_ONLY, or SESSION keyword in the statement is used for following assignments that have no keyword specified.

Examples of multiple-variable assignment:

```
SET GLOBAL sort_buffer_size = 1000000, SESSION sort_buffer_size = 1000000;
SET @@GLOBAL.sort_buffer_size = 1000000, @@LOCAL.sort_buffer_size = 1000000;
SET GLOBAL max_connections = 1000, sort_buffer_size = 1000000;
```

The @@GLOBAL., @@PERSIST., @@PERSIST_ONLY., @@SESSION., and @@ modifiers apply only to the immediately following system variable, not any remaining system variables. This statement sets the sort_buffer_size global value to 50000 and the session value to 1000000:

```
SET @@GLOBAL.sort_buffer_size = 50000, sort_buffer_size = 1000000;
```

System Variable References in Expressions

To refer to the value of a system variable in expressions, use one of the @@-modifiers (except @@PERSIST. and @@PERSIST_ONLY., which are not permitted in expressions). For example, you can retrieve system variable values in a SELECT statement like this:

```
SELECT @@GLOBAL.sql_mode, @@SESSION.sql_mode, @@sql_mode;
```

Note

A reference to a system variable in an expression as `@@var_name` (with `@@` rather than `@@GLOBAL.` or `@@SESSION.`) returns the session value if it exists and the global value otherwise. This differs from `SET @@var_name = expr`, which always refers to the session value.

13.7.6.2 SET CHARACTER SET Syntax

```
SET {CHARACTER SET | CHARSET}
    {'charset_name' | DEFAULT}
```

This statement maps all strings sent between the server and the current client with the given mapping. `SET CHARACTER SET` sets three session system variables: `character_set_client` and `character_set_results` are set to the given character set, and `character_set_connection` to the value of `character_set_database`. See Section 10.4, “Connection Character Sets and Collations”.

`charset_name` may be quoted or unquoted.

The default character set mapping can be restored by using the value `DEFAULT`. The default depends on the server configuration.

Some character sets cannot be used as the client character set. Attempting to use them with `SET CHARACTER SET` produces an error. See Impermissible Client Character Sets.

13.7.6.3 SET NAMES Syntax

```
SET NAMES {'charset_name'
    [COLLATE 'collation_name'] | DEFAULT}
```

This statement sets the three session system variables `character_set_client`, `character_set_connection`, and `character_set_results` to the given character set. Setting `character_set_connection` to `charset_name` also sets `collation_connection` to the default collation for `charset_name`. See Section 10.4, “Connection Character Sets and Collations”.

The optional `COLLATE` clause may be used to specify a collation explicitly. If given, the collation must one of the permitted collations for *`charset_name`*.

`charset_name` and *`collation_name`* may be quoted or unquoted.

The default mapping can be restored by using a value of `DEFAULT`. The default depends on the server configuration.

Some character sets cannot be used as the client character set. Attempting to use them with `SET NAMES` produces an error. See Impermissible Client Character Sets.

13.7.7 SHOW Syntax

`SHOW` has many forms that provide information about databases, tables, columns, or status information about the server. This section describes those following:

```
SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

```
SHOW CHARACTER SET [like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION func_name
SHOW CREATE PROCEDURE proc_name
SHOW CREATE TABLE tbl_name
SHOW CREATE TRIGGER trigger_name
SHOW CREATE VIEW view_name
SHOW DATABASES [like_or_where]
SHOW ENGINE engine_name {STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW EVENTS
SHOW FUNCTION CODE func_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW MASTER STATUS
SHOW OPEN TABLES [FROM db_name] [like_or_where]
SHOW PLUGINS
SHOW PROCEDURE CODE proc_name
SHOW PROCEDURE STATUS [like_or_where]
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW PROFILE [types] [FOR QUERY n] [OFFSET n] [LIMIT n]
SHOW PROFILES
SHOW RELAYLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW SLAVE HOSTS
SHOW SLAVE STATUS [FOR CHANNEL channel]
SHOW [GLOBAL | SESSION] STATUS [like_or_where]
SHOW TABLE STATUS [FROM db_name] [like_or_where]
SHOW [FULL] TABLES [FROM db_name] [like_or_where]
SHOW TRIGGERS [FROM db_name] [like_or_where]
SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
SHOW WARNINGS [LIMIT [offset,] row_count]

like_or_where:
    LIKE 'pattern'
  | WHERE expr
```

If the syntax for a given `SHOW` statement includes a `LIKE 'pattern'` part, `'pattern'` is a string that can contain the SQL `%` and `_` wildcard characters. The pattern is useful for restricting statement output to matching values.

Several `SHOW` statements also accept a `WHERE` clause that provides more flexibility in specifying which rows to display. See Section 25.42, “Extensions to SHOW Statements”.

Many MySQL APIs (such as PHP) enable you to treat the result returned from a `SHOW` statement as you would a result set from a `SELECT`; see Chapter 28, *Connectors and APIs*, or your API documentation for more information. In addition, you can work in SQL with results from queries on tables in the `INFORMATION_SCHEMA` database, which you cannot easily do with results from `SHOW` statements. See Chapter 25, *INFORMATION_SCHEMA Tables*.

13.7.7.1 SHOW BINARY LOGS Syntax

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Lists the binary log files on the server. This statement is used as part of the procedure described in Section 13.4.1.1, “PURGE BINARY LOGS Syntax”, that shows how to determine which logs can be purged. `SHOW BINARY LOGS` requires the `REPLICATION CLIENT` or `SUPER` privilege.

Encrypted binary log files have a 512-byte file header that stores information required for encryption and decryption of the file. This is included in the file size displayed by `SHOW BINARY LOGS`. The `Encrypted` column shows whether or not the binary log file is encrypted. Binary log encryption is active if `binlog_encryption=ON` is set for the server. Existing binary log files are not encrypted or decrypted if binary log encryption is activated or deactivated while the server is running.

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+-----------+
| Log_name      | File_size | Encrypted |
+---------------+-----------+-----------+
| binlog.000015 |    724935 |       Yes |
| binlog.000016 |    733481 |       Yes |
+---------------+-----------+-----------+
```

`SHOW MASTER LOGS` is equivalent to `SHOW BINARY LOGS`.

13.7.7.2 SHOW BINLOG EVENTS Syntax

```
SHOW BINLOG EVENTS
   [IN 'log_name']
   [FROM pos]
   [LIMIT [offset,] row_count]
```

Shows the events in the binary log. If you do not specify `'log_name'`, the first binary log is displayed. `SHOW BINLOG EVENTS` requires the `REPLICATION SLAVE` privilege.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.2.10, “SELECT Syntax”.

Note

Issuing a `SHOW BINLOG EVENTS` with no `LIMIT` clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the binary log (which includes all statements executed by the server that modify data). As an alternative to `SHOW BINLOG EVENTS`, use the `mysqlbinlog` utility to save the binary log to a text file for later examination and analysis. See Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

`SHOW BINLOG EVENTS` displays the following fields for each event in the binary log:

- `Log_name`

  The name of the file that is being listed.

- `Pos`

  The position at which the event occurs.

- `Event_type`

  An identifier that describes the event type.

- `Server_id`

  The server ID of the server on which the event originated.

- `End_log_pos`

The position at which the next event begins, which is equal to `Pos` plus the size of the event.

- `Info`

  More detailed information about the event type. The format of this information depends on the event type.

Note

Some events relating to the setting of user and system variables are not included in the output from `SHOW BINLOG EVENTS`. To get complete coverage of events within a binary log, use `mysqlbinlog`.

Note

`SHOW BINLOG EVENTS` does *not* work with relay log files. You can use `SHOW RELAYLOG EVENTS` for this purpose.

13.7.7.3 SHOW CHARACTER SET Syntax

```
SHOW CHARACTER SET
    [LIKE 'pattern' | WHERE expr]
```

The `SHOW CHARACTER SET` statement shows all available character sets. The `LIKE` clause, if present, indicates which character set names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, “Extensions to SHOW Statements”. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

`SHOW CHARACTER SET` output has these columns:

- `Charset`

  The character set name.

- `Description`

  A description of the character set.

- `Default collation`

  The default collation for the character set.

- `Maxlen`

  The maximum number of bytes required to store one character.

The `filename` character set is for internal use only; consequently, `SHOW CHARACTER SET` does not display it.

Character set information is also available from the `INFORMATION_SCHEMA CHARACTER_SETS` table. See Section 25.2, “The INFORMATION_SCHEMA CHARACTER_SETS Table”.

13.7.7.4 SHOW COLLATION Syntax

```
SHOW COLLATION
    [LIKE 'pattern' | WHERE expr]
```

This statement lists collations supported by the server. By default, the output from `SHOW COLLATION` includes all available collations. The `LIKE` clause, if present, indicates which collation names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, “Extensions to SHOW Statements”. For example:

```
mysql> SHOW COLLATION WHERE Charset = 'latin1';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         | Yes      |       1 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       1 |
| latin1_danish_ci  | latin1  | 15 |         | Yes      |       1 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci | latin1  | 48 |         | Yes      |       1 |
| latin1_general_cs | latin1  | 49 |         | Yes      |       1 |
| latin1_spanish_ci | latin1  | 94 |         | Yes      |       1 |
+-------------------+---------+----+---------+----------+---------+
```

`SHOW COLLATION` output has these columns:

- `Collation`

  The collation name.

- `Charset`

  The name of the character set with which the collation is associated.

- `Id`

  The collation ID.

- `Default`

  Whether the collation is the default for its character set.

- `Compiled`

  Whether the character set is compiled into the server.

- `Sortlen`

  This is related to the amount of memory required to sort strings expressed in the character set.

To see the default collation for each character set, use the following statement. `Default` is a reserved word, so to use it as an identifier, it must be quoted as such:

```
mysql> SHOW COLLATION WHERE `Default` = 'Yes';
+--------------------+----------+----+---------+----------+---------+
```

```
| Collation           | Charset  | Id | Default | Compiled | Sortlen |
+---------------------+----------+----+---------+----------+---------+
| big5_chinese_ci     | big5     |  1 | Yes     | Yes      |       1 |
| dec8_swedish_ci     | dec8     |  3 | Yes     | Yes      |       1 |
| cp850_general_ci    | cp850    |  4 | Yes     | Yes      |       1 |
| hp8_english_ci      | hp8      |  6 | Yes     | Yes      |       1 |
| koi8r_general_ci    | koi8r    |  7 | Yes     | Yes      |       1 |
| latin1_swedish_ci   | latin1   |  8 | Yes     | Yes      |       1 |
...
```

Collation information is also available from the `INFORMATION_SCHEMA COLLATIONS` table. See Section 25.4, “The INFORMATION_SCHEMA COLLATIONS Table”.

13.7.7.5 SHOW COLUMNS Syntax

```
SHOW [EXTENDED] [FULL] {COLUMNS | FIELDS}
    {FROM | IN} tbl_name
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW COLUMNS` displays information about the columns in a given table. It also works for views. `SHOW COLUMNS` displays information only for those columns for which you have some privilege.

```
mysql> SHOW COLUMNS FROM City;
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| ID          | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name        | char(35) | NO   |     |         |                |
| CountryCode | char(3)  | NO   | MUL |         |                |
| District    | char(20) | NO   |     |         |                |
| Population  | int(11)  | NO   |     | 0       |                |
+-------------+----------+------+-----+---------+----------------+
```

An alternative to *`tbl_name`* `FROM` *`db_name`* syntax is *`db_name.tbl_name`*. These two statements are equivalent:

```
SHOW COLUMNS FROM mytable FROM mydb;
SHOW COLUMNS FROM mydb.mytable;
```

The optional `EXTENDED` keyword causes the output to include information about hidden columns that MySQL uses internally and are not accessible by users.

The optional `FULL` keyword causes the output to include the column collation and comments, as well as the privileges you have for each column.

The `LIKE` clause, if present, indicates which column names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, “Extensions to SHOW Statements”.

The data types may differ from what you expect them to be based on a `CREATE TABLE` statement because MySQL sometimes changes data types when you create or alter a table. The conditions under which this occurs are described in Section 13.1.20.8, “Silent Column Specification Changes”.

`SHOW COLUMNS` displays the following values for each table column:

- `Field`

  The name of the column.

- `Type`

  The column data type.

- `Collation`

  The collation for nonbinary string columns, or `NULL` for other columns. This value is displayed only if you use the `FULL` keyword.

- `Null`

  The column nullability. The value is `YES` if `NULL` values can be stored in the column, `NO` if not.

- `Key`

  Whether the column is indexed:

  - If `Key` is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.
  - If `Key` is `PRI`, the column is a `PRIMARY KEY` or is one of the columns in a multiple-column `PRIMARY KEY`.
  - If `Key` is `UNI`, the column is the first column of a `UNIQUE` index. (A `UNIQUE` index permits multiple `NULL` values, but you can tell whether the column permits `NULL` by checking the `Null` field.)
  - If `Key` is `MUL`, the column is the first column of a nonunique index in which multiple occurrences of a given value are permitted within the column.

  If more than one of the `Key` values applies to a given column of a table, `Key` displays the one with the highest priority, in the order `PRI`, `UNI`, `MUL`.

  A `UNIQUE` index may be displayed as `PRI` if it cannot contain `NULL` values and there is no `PRIMARY KEY` in the table. A `UNIQUE` index may display as `MUL` if several columns form a composite `UNIQUE` index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

- `Default`

  The default value for the column. This is `NULL` if the column has an explicit default of `NULL`, or if the column definition includes no `DEFAULT` clause.

- `Extra`

  Any additional information that is available about a given column. The value is nonempty in these cases:

  - `auto_increment` for columns that have the `AUTO_INCREMENT` attribute.
  - `on update CURRENT_TIMESTAMP` for `TIMESTAMP` or `DATETIME` columns that have the `ON UPDATE CURRENT_TIMESTAMP` attribute.
  - `VIRTUAL GENERATED` or `VIRTUAL STORED` for generated columns.
  - `DEFAULT_GENERATED` for columns that have an expression default value.

- `Privileges`

  The privileges you have for the column. This value is displayed only if you use the `FULL` keyword.

- `Comment`

  Any comment included in the column definition. This value is displayed only if you use the `FULL` keyword.

Table column information is also available from the `INFORMATION_SCHEMA COLUMNS` table. See Section 25.6, “The INFORMATION_SCHEMA COLUMNS Table”. The extended information about hidden columns is available only using `SHOW EXTENDED COLUMNS`; it cannot be obtained from the `COLUMNS` table.

You can list a table's columns with the `mysqlshow *db_name tbl_name*` command.

The `DESCRIBE` statement provides information similar to `SHOW COLUMNS`. See Section 13.8.1, “DESCRIBE Syntax”.

The `SHOW CREATE TABLE`, `SHOW TABLE STATUS`, and `SHOW INDEX` statements also provide information about tables. See Section 13.7.7, “SHOW Syntax”.

13.7.7.6 SHOW CREATE DATABASE Syntax

```
SHOW CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
```

Shows the `CREATE DATABASE` statement that creates the named database. If the `SHOW` statement includes an `IF NOT EXISTS` clause, the output too includes such a clause. `SHOW CREATE SCHEMA` is a synonym for `SHOW CREATE DATABASE`.

```
mysql> SHOW CREATE DATABASE test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET utf8mb4
                 COLLATE utf8mb4_0900_ai_ci */ /*!80014 DEFAULT ENCRYPTION='N' */

mysql> SHOW CREATE SCHEMA test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test` /*!40100 DEFAULT CHARACTER SET utf8mb4
                 COLLATE utf8mb4_0900_ai_ci */ /*!80014 DEFAULT ENCRYPTION='N' */
```

`SHOW CREATE DATABASE` quotes table and column names according to the value of the `sql_quote_show_create` option. See Section 5.1.8, “Server System Variables”.

13.7.7.7 SHOW CREATE EVENT Syntax

```
SHOW CREATE EVENT event_name
```

This statement displays the `CREATE EVENT` statement needed to re-create a given event. It requires the `EVENT` privilege for the database from which the event is to be shown. For example (using the same event `e_daily` defined and then altered in Section 13.7.7.18, “SHOW EVENTS Syntax”):

```
mysql> SHOW CREATE EVENT myschema.e_daily\G
*************************** 1. row ***************************
               Event: e_daily
            sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
                      NO_ZERO_IN_DATE,NO_ZERO_DATE,
                      ERROR_FOR_DIVISION_BY_ZERO,
```

```
                      NO_ENGINE_SUBSTITUTION
           time_zone: SYSTEM
       Create Event: CREATE DEFINER=`jon`@`ghidora` EVENT `e_daily`
                       ON SCHEDULE EVERY 1 DAY
                       STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
                       ON COMPLETION NOT PRESERVE
                       ENABLE
                       COMMENT 'Saves total number of sessions then
                               clears the table each day'
                       DO BEGIN
                         INSERT INTO site_activity.totals (time, total)
                           SELECT CURRENT_TIMESTAMP, COUNT(*)
                             FROM site_activity.sessions;
                         DELETE FROM site_activity.sessions;
                       END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: utf8mb4_0900_ai_ci
```

`character_set_client` is the session value of the `character_set_client` system variable when the event was created. `collation_connection` is the session value of the `collation_connection` system variable when the event was created. `Database Collation` is the collation of the database with which the event is associated.

The output reflects the current status of the event (`ENABLE`) rather than the status with which it was created.

13.7.7.8 SHOW CREATE FUNCTION Syntax

```
SHOW CREATE FUNCTION func_name
```

This statement is similar to `SHOW CREATE PROCEDURE` but for stored functions. See Section 13.7.7.9, “SHOW CREATE PROCEDURE Syntax”.

13.7.7.9 SHOW CREATE PROCEDURE Syntax

```
SHOW CREATE PROCEDURE proc_name
```

This statement is a MySQL extension. It returns the exact string that can be used to re-create the named stored procedure. A similar statement, `SHOW CREATE FUNCTION`, displays information about stored functions (see Section 13.7.7.8, “SHOW CREATE FUNCTION Syntax”).

To use either statement, you must have the global `SELECT` privilege.

```
mysql> SHOW CREATE PROCEDURE test.simpleproc\G
*************************** 1. row ***************************
           Procedure: simpleproc
            sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
                      NO_ZERO_IN_DATE,NO_ZERO_DATE,
                      ERROR_FOR_DIVISION_BY_ZERO,
                      NO_ENGINE_SUBSTITUTION
    Create Procedure: CREATE PROCEDURE `simpleproc`(OUT param1 INT)
                      BEGIN
                      SELECT COUNT(*) INTO param1 FROM t;
                      END
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: utf8mb4_0900_ai_ci
```

```
mysql> SHOW CREATE FUNCTION test.hello\G
*************************** 1. row ***************************
            Function: hello
            sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
                      NO_ZERO_IN_DATE,NO_ZERO_DATE,
                      ERROR_FOR_DIVISION_BY_ZERO,
                      NO_ENGINE_SUBSTITUTION
     Create Function: CREATE FUNCTION `hello`(s CHAR(20))
                      RETURNS char(50) CHARSET utf8mb4
                      RETURN CONCAT('Hello, ',s,'!')
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: utf8mb4_0900_ai_ci
```

`character_set_client` is the session value of the `character_set_client` system variable when the routine was created. `collation_connection` is the session value of the `collation_connection` system variable when the routine was created. `Database Collation` is the collation of the database with which the routine is associated.

13.7.7.10 SHOW CREATE TABLE Syntax

```
SHOW CREATE TABLE tbl_name
```

Shows the `CREATE TABLE` statement that creates the named table. To use this statement, you must have some privilege for the table. This statement also works with views.

```
mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `s` char(60) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

As of MySQL 8.0.16, MySQL implements `CHECK` constraints and `SHOW CREATE TABLE` displays them. All `CHECK` constraints are displayed as table constraints. That is, a `CHECK` constraint originally specified as part of a column definition displays as a separate clause not part of the column definition. Example:

```
mysql> CREATE TABLE t1 (
         i1 INT CHECK (i1 <> 0),      -- column constraint
         i2 INT,
         CHECK (i2 > i1),             -- table constraint
         CHECK (i2 <> 0) NOT ENFORCED -- table constraint, not enforced
       );

mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `i1` int(11) DEFAULT NULL,
  `i2` int(11) DEFAULT NULL,
  CONSTRAINT `t1_chk_1` CHECK ((`i1` <> 0)),
  CONSTRAINT `t1_chk_2` CHECK ((`i2` > `i1`)),
  CONSTRAINT `t1_chk_3` CHECK ((`i2` <> 0)) /*!80016 NOT ENFORCED */
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

`SHOW CREATE TABLE` quotes table and column names according to the value of the `sql_quote_show_create` option. See Section 5.1.8, “Server System Variables”.

For information about how `CREATE TABLE` statements are stored by MySQL, see Section 13.1.20.1, “CREATE TABLE Statement Retention”.

13.7.7.11 SHOW CREATE TRIGGER Syntax

```
SHOW CREATE TRIGGER trigger_name
```

This statement shows the `CREATE TRIGGER` statement that creates the named trigger. This statement requires the `TRIGGER` privilege for the table associated with the trigger.

```
mysql> SHOW CREATE TRIGGER ins_sum\G
*************************** 1. row ***************************
               Trigger: ins_sum
              sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
                        NO_ZERO_IN_DATE,NO_ZERO_DATE,
                        ERROR_FOR_DIVISION_BY_ZERO,
                        NO_ENGINE_SUBSTITUTION
SQL Original Statement: CREATE DEFINER=`me`@`localhost` TRIGGER `ins_sum`
                        BEFORE INSERT ON `account`
                        FOR EACH ROW SET @sum = @sum + NEW.amount
  character_set_client: utf8mb4
  collation_connection: utf8mb4_0900_ai_ci
    Database Collation: utf8mb4_0900_ai_ci
               Created: 2018-08-08 10:10:12.61
```

`SHOW CREATE TRIGGER` output has these columns:

- `Trigger`: The trigger name.
- `sql_mode`: The SQL mode in effect when the trigger executes.
- `SQL Original Statement`: The `CREATE TRIGGER` statement that defines the trigger.
- `character_set_client`: The session value of the `character_set_client` system variable when the trigger was created.
- `collation_connection`: The session value of the `collation_connection` system variable when the trigger was created.
- `Database Collation`: The collation of the database with which the trigger is associated.
- `Created`: The date and time when the trigger was created. This is a `TIMESTAMP(2)` value (with a fractional part in hundredths of seconds) for triggers.

Trigger information is also available from the `INFORMATION_SCHEMA TRIGGERS` table. See Section 25.34, “The INFORMATION_SCHEMA TRIGGERS Table”.

13.7.7.12 SHOW CREATE USER Syntax

```
SHOW CREATE USER user
```

This statement shows the `CREATE USER` statement that creates the named user. An error occurs if the user does not exist. The statement requires the `SELECT` privilege for the `mysql` system database, except to see information for the current user. For the current user, the `SELECT` privilege for the `mysql.user` system table is required for display of the password hash in the `IDENTIFIED AS` clause; otherwise, the hash displays as `<secret>`.

To name the account, use the format described in Section 6.2.4, “Specifying Account Names”. The host name part of the account name, if omitted, defaults to `'%'`. It is also possible to specify `CURRENT_USER` or `CURRENT_USER()` to refer to the account associated with the current session.

Password hash values displayed in the `IDENTIFIED WITH` clause of output from `SHOW CREATE USER` may contain unprintable characters that have adverse effects on terminal displays and in other environments. Enabling the `print_identified_with_as_hex` system variable (available as of MySQL 8.0.17) causes `SHOW CREATE USER` to display such hash values as hexadecimal strings rather than as regular string literals. Hash values that do not contain unprintable characters still display as regular string literals, even with this variable enabled.

```
mysql> CREATE USER 'u1'@'localhost' IDENTIFIED BY 'secret';
mysql> SET print_identified_with_as_hex = ON;
mysql> SHOW CREATE USER 'u1'@'localhost'\G
*************************** 1. row ***************************
CREATE USER for u1@localhost: CREATE USER 'u1'@'localhost'
IDENTIFIED WITH 'caching_sha2_password'
AS 0x244124303035240C7745603626313D613C4C10633E0A104B1E14135A544A7871567245614F48723446435463365 46F624F6C78613
REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK
PASSWORD HISTORY DEFAULT PASSWORD REUSE INTERVAL DEFAULT
PASSWORD REQUIRE CURRENT DEFAULT
```

To display the privileges granted to an account, use the `SHOW GRANTS` statement. See Section 13.7.7.21, “SHOW GRANTS Syntax”.

13.7.7.13 SHOW CREATE VIEW Syntax

```
SHOW CREATE VIEW view_name
```

This statement shows the `CREATE VIEW` statement that creates the named view.

```
mysql> SHOW CREATE VIEW v\G
*************************** 1. row ***************************
                View: v
         Create View: CREATE ALGORITHM=UNDEFINED
                      DEFINER=`bob`@`localhost`
                      SQL SECURITY DEFINER VIEW
                      `v` AS select 1 AS `a`,2 AS `b`
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
```

`character_set_client` is the session value of the `character_set_client` system variable when the view was created. `collation_connection` is the session value of the `collation_connection` system variable when the view was created.

Use of `SHOW CREATE VIEW` requires the `SHOW VIEW` privilege, and the `SELECT` privilege for the view in question.

View information is also available from the `INFORMATION_SCHEMA VIEWS` table. See Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”.

MySQL lets you use different `sql_mode` settings to tell the server the type of SQL syntax to support. For example, you might use the `ANSI` SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (`||`), in your queries. If you then create a view that concatenates items, you might worry that changing the `sql_mode` setting to a value different from `ANSI` could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL

always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a `CONCAT()` function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW CREATE VIEW test.v\G
*************************** 1. row ***************************
                View: v
         Create View: CREATE VIEW "v" AS select concat('a','b') AS "col1"
...
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of `sql_mode` will not affect the results from the view. However an additional consequence is that comments prior to `SELECT` are stripped from the definition by the server.

13.7.7.14 SHOW DATABASES Syntax

```
SHOW {DATABASES | SCHEMAS}
    [LIKE 'pattern' | WHERE expr]
```

`SHOW DATABASES` lists the databases on the MySQL server host. `SHOW SCHEMAS` is a synonym for `SHOW DATABASES`. The `LIKE` clause, if present, indicates which database names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, “Extensions to SHOW Statements”.

You see only those databases for which you have some kind of privilege, unless you have the global `SHOW DATABASES` privilege. You can also get this list using the `mysqlshow` command.

If the server was started with the `--skip-show-database` option, you cannot use this statement at all unless you have the `SHOW DATABASES` privilege.

MySQL implements databases as directories in the data directory, so this statement simply lists directories in that location. However, the output may include names of directories that do not correspond to actual databases.

Database information is also available from the `INFORMATION_SCHEMA SCHEMATA` table. See Section 25.24, “The INFORMATION_SCHEMA SCHEMATA Table”.

Caution

Because any static global privilege is considered a privilege for all databases, any static global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`, except databases that have been restricted at the database level by partial revokes.

13.7.7.15 SHOW ENGINE Syntax

```
SHOW ENGINE engine_name {STATUS | MUTEX}
```

`SHOW ENGINE` displays operational information about a storage engine. It requires the `PROCESS` privilege. The statement has these variants:

```
SHOW ENGINE INNODB STATUS
SHOW ENGINE INNODB MUTEX
SHOW ENGINE PERFORMANCE_SCHEMA STATUS
```

`SHOW ENGINE INNODB STATUS` displays extensive information from the standard `InnoDB` Monitor about the state of the `InnoDB` storage engine. For information about the standard monitor and other `InnoDB` Monitors that provide information about `InnoDB` processing, see Section 15.16, “InnoDB Monitors”.

`SHOW ENGINE INNODB MUTEX` displays `InnoDB` mutex and rw-lock statistics.

Note

`InnoDB` mutexes and rwlocks can also be monitored using Performance Schema tables. See Section 15.15.2, “Monitoring InnoDB Mutex Waits Using Performance Schema”.

Mutex statistics collection is configured dynamically using the following options:

- To enable the collection of mutex statistics, run:

  ```
  SET GLOBAL innodb_monitor_enable='latch';
  ```

- To reset mutex statistics, run:

  ```
  SET GLOBAL innodb_monitor_reset='latch';
  ```

- To disable the collection of mutex statistics, run:

  ```
  SET GLOBAL innodb_monitor_disable='latch';
  ```

Collection of mutex statistics for `SHOW ENGINE INNODB MUTEX` can also be enabled by setting `innodb_monitor_enable='all'`, or disabled by setting `innodb_monitor_disable='all'`.

`SHOW ENGINE INNODB MUTEX` output has these columns:

- `Type`

  Always `InnoDB`.

- `Name`

  For mutexes, the `Name` field reports only the mutex name. For rwlocks, the `Name` field reports the source file where the rwlock is implemented, and the line number in the file where the rwlock is created. The line number is specific to your version of MySQL.

- `Status`

  The mutex status. This field reports the number of spins, waits, and calls. Statistics for low-level operating system mutexes, which are implemented outside of `InnoDB`, are not reported.

  - `spins` indicates the number of spins.
  - `waits` indicates the number of mutex waits.
  - `calls` indicates how many times the mutex was requested.

`SHOW ENGINE INNODB MUTEX` skips the mutexes and rw-locks of buffer pool blocks, as the amount of output can be overwhelming on systems with a large buffer pool. (There is one mutex and one rw-lock in

each 16K buffer pool block, and there are 65,536 blocks per gigabyte.) `SHOW ENGINE INNODB MUTEX` also does not list any mutexes or rw-locks that have never been waited on (`os_waits=0`). Thus, `SHOW ENGINE INNODB MUTEX` only displays information about mutexes and rw-locks outside of the buffer pool that have caused at least one OS-level wait.

Use `SHOW ENGINE PERFORMANCE_SCHEMA STATUS` to inspect the internal operation of the Performance Schema code:

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS\G
...
*************************** 3. row ***************************
  Type: performance_schema
  Name: events_waits_history.size
Status: 76
*************************** 4. row ***************************
  Type: performance_schema
  Name: events_waits_history.count
Status: 10000
*************************** 5. row ***************************
  Type: performance_schema
  Name: events_waits_history.memory
Status: 760000
...
*************************** 57. row ***************************
  Type: performance_schema
  Name: performance_schema.memory
Status: 26459600
...
```

This statement is intended to help the DBA understand the effects that different Performance Schema options have on memory requirements.

`Name` values consist of two parts, which name an internal buffer and a buffer attribute, respectively. Interpret buffer names as follows:

- An internal buffer that is not exposed as a table is named within parentheses. Examples: `(pfs_cond_class).size`, `(pfs_mutex_class).memory`.
- An internal buffer that is exposed as a table in the `performance_schema` database is named after the table, without parentheses. Examples: `events_waits_history.size`, `mutex_instances.count`.
- A value that applies to the Performance Schema as a whole begins with `performance_schema`. Example: `performance_schema.memory`.

Buffer attributes have these meanings:

- `size` is the size of the internal record used by the implementation, such as the size of a row in a table. `size` values cannot be changed.
- `count` is the number of internal records, such as the number of rows in a table. `count` values can be changed using Performance Schema configuration options.
- For a table, *`tbl_name`*`.memory` is the product of `size` and `count`. For the Performance Schema as a whole, `performance_schema.memory` is the sum of all the memory used (the sum of all other `memory` values).

In some cases, there is a direct relationship between a Performance Schema configuration parameter and a `SHOW ENGINE` value. For example, `events_waits_history_long.count` corresponds to `performance_schema_events_waits_history_long_size`. In other cases,

the relationship is more complex. For example, `events_waits_history.count` corresponds to `performance_schema_events_waits_history_size` (the number of rows per thread) multiplied by `performance_schema_max_thread_instances` (the number of threads).

SHOW ENGINE NDB STATUS. If the server has the `NDB` storage engine enabled, `SHOW ENGINE NDB STATUS` displays cluster status information such as the number of connected data nodes, the cluster connectstring, and cluster binary log epochs, as well as counts of various Cluster API objects created by the MySQL Server when connected to the cluster. Sample output from this statement is shown here:

```
mysql> SHOW ENGINE NDB STATUS;
+------------+----------------------+--------------------------------------------------+
| Type       | Name                 | Status                                           |
+------------+----------------------+--------------------------------------------------+
| ndbcluster | connection           | cluster_node_id=7,
  connected_host=198.51.100.103, connected_port=1186, number_of_data_nodes=4,
  number_of_ready_data_nodes=3, connect_count=0                                       |
| ndbcluster | NdbTransaction       | created=6, free=0, sizeof=212                    |
| ndbcluster | NdbOperation         | created=8, free=8, sizeof=660                    |
| ndbcluster | NdbIndexScanOperation | created=1, free=1, sizeof=744                   |
| ndbcluster | NdbIndexOperation    | created=0, free=0, sizeof=664                    |
| ndbcluster | NdbRecAttr           | created=1285, free=1285, sizeof=60               |
| ndbcluster | NdbApiSignal         | created=16, free=16, sizeof=136                  |
| ndbcluster | NdbLabel             | created=0, free=0, sizeof=196                    |
| ndbcluster | NdbBranch            | created=0, free=0, sizeof=24                     |
| ndbcluster | NdbSubroutine        | created=0, free=0, sizeof=68                     |
| ndbcluster | NdbCall              | created=0, free=0, sizeof=16                     |
| ndbcluster | NdbBlob              | created=1, free=1, sizeof=264                    |
| ndbcluster | NdbReceiver          | created=4, free=0, sizeof=68                     |
| ndbcluster | binlog               | latest_epoch=155467, latest_trans_epoch=148126,
  latest_received_binlog_epoch=0, latest_handled_binlog_epoch=0,
  latest_applied_binlog_epoch=0                                                       |
+------------+----------------------+--------------------------------------------------+
```

The `Status` column in each of these rows provides information about the MySQL server's connection to the cluster and about the cluster binary log's status, respectively. The `Status` information is in the form of comma-delimited set of name/value pairs.

The `connection` row's `Status` column contains the name/value pairs described in the following table.

Name	Value
`cluster_node_id`	The node ID of the MySQL server in the cluster
`connected_host`	The host name or IP address of the cluster management server to which the MySQL server is connected
`connected_port`	The port used by the MySQL server to connect to the management server (`connected_host`)
`number_of_data_nodes`	The number of data nodes configured for the cluster (that is, the number of `[ndbd]` sections in the cluster `config.ini` file)
`number_of_ready_data_nodes`	The number of data nodes in the cluster that are actually running
`connect_count`	The number of times this `mysqld` has connected or reconnected to cluster data nodes

The `binlog` row's `Status` column contains information relating to NDB Cluster Replication. The name/value pairs it contains are described in the following table.

Name	Value
latest_epoch	The most recent epoch most recently run on this MySQL server (that is, the sequence number of the most recent transaction run on the server)
latest_trans_epoch	The most recent epoch processed by the cluster's data nodes
latest_received_binlog_epoch	The most recent epoch received by the binary log thread
latest_handled_binlog_epoch	The most recent epoch processed by the binary log thread (for writing to the binary log)
latest_applied_binlog_epoch	The most recent epoch actually written to the binary log

See Section 22.6, “NDB Cluster Replication”, for more information.

The remaining rows from the output of SHOW ENGINE NDB STATUS which are most likely to prove useful in monitoring the cluster are listed here by Name:

- NdbTransaction: The number and size of NdbTransaction objects that have been created. An NdbTransaction is created each time a table schema operation (such as CREATE TABLE or ALTER TABLE) is performed on an NDB table.
- NdbOperation: The number and size of NdbOperation objects that have been created.
- NdbIndexScanOperation: The number and size of NdbIndexScanOperation objects that have been created.
- NdbIndexOperation: The number and size of NdbIndexOperation objects that have been created.
- NdbRecAttr: The number and size of NdbRecAttr objects that have been created. In general, one of these is created each time a data manipulation statement is performed by an SQL node.
- NdbBlob: The number and size of NdbBlob objects that have been created. An NdbBlob is created for each new operation involving a BLOB column in an NDB table.
- NdbReceiver: The number and size of any NdbReceiver object that have been created. The number in the created column is the same as the number of data nodes in the cluster to which the MySQL server has connected.

Note

SHOW ENGINE NDB STATUS returns an empty result if no operations involving NDB tables have been performed during the current session by the MySQL client accessing the SQL node on which this statement is run.

13.7.7.16 SHOW ENGINES Syntax

```
SHOW [STORAGE] ENGINES
```

SHOW ENGINES displays status information about the server's storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is.

For information about MySQL storage engines, see Chapter 15, *The InnoDB Storage Engine*, and Chapter 16, *Alternative Storage Engines*.

```
mysql> SHOW ENGINES\G
```

```
*************************** 1. row ***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: FEDERATED
     Support: NO
     Comment: Federated MySQL storage engine
Transactions: NULL
          XA: NULL
  Savepoints: NULL
*************************** 5. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 7. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 8. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 9. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
```

The output from `SHOW ENGINES` may vary according to the MySQL version used and other factors.

`SHOW ENGINES` output has these columns:

- `Engine`

  The name of the storage engine.

- `Support`

  The server's level of support for the storage engine, as shown in the following table.

Value	Meaning
`YES`	The engine is supported and is active
`DEFAULT`	Like `YES`, plus this is the default engine
`NO`	The engine is not supported
`DISABLED`	The engine is supported but has been disabled

  A value of `NO` means that the server was compiled without support for the engine, so it cannot be enabled at runtime.

  A value of `DISABLED` occurs either because the server was started with an option that disables the engine, or because not all options required to enable it were given. In the latter case, the error log should contain a reason indicating why the option is disabled. See Section 5.4.2, “The Error Log”.

  You might also see `DISABLED` for a storage engine if the server was compiled to support it, but was started with a `--skip-`*`engine_name`* option. For the `NDB` storage engine, `DISABLED` means the server was compiled with support for NDB Cluster, but was not started with the `--ndbcluster` option.

  All MySQL servers support `MyISAM` tables. It is not possible to disable `MyISAM`.

- `Comment`

  A brief description of the storage engine.

- `Transactions`

  Whether the storage engine supports transactions.

- `XA`

  Whether the storage engine supports XA transactions.

- `Savepoints`

  Whether the storage engine supports savepoints.

Storage engine information is also available from the `INFORMATION_SCHEMA ENGINES` table. See Section 25.9, “The INFORMATION_SCHEMA ENGINES Table”.

13.7.7.17 SHOW ERRORS Syntax

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

`SHOW ERRORS` is a diagnostic statement that is similar to `SHOW WARNINGS`, except that it displays information only for errors, rather than for errors, warnings, and notes.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.2.10, “SELECT Syntax”.

The `SHOW COUNT(*) ERRORS` statement displays the number of errors. You can also retrieve this number from the `error_count` variable:

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

`SHOW ERRORS` and `error_count` apply only to errors, not warnings or notes. In other respects, they are similar to `SHOW WARNINGS` and `warning_count`. In particular, `SHOW ERRORS` cannot display information for more than `max_error_count` messages, and `error_count` can exceed the value of `max_error_count` if the number of errors exceeds `max_error_count`.

For more information, see Section 13.7.7.40, “SHOW WARNINGS Syntax”.

13.7.7.18 SHOW EVENTS Syntax

```
SHOW EVENTS
    [{FROM | IN} schema_name]
    [LIKE 'pattern' | WHERE expr]
```

This statement displays information about Event Manager events, which are discussed in Section 24.4, “Using the Event Scheduler”. It requires the `EVENT` privilege for the database from which the events are to be shown.

In its simplest form, `SHOW EVENTS` lists all of the events in the current schema:

```
mysql> SELECT CURRENT_USER(), SCHEMA();
+----------------+----------+
| CURRENT_USER() | SCHEMA() |
+----------------+----------+
| jon@ghidora    | myschema |
+----------------+----------+
1 row in set (0.00 sec)

mysql> SHOW EVENTS\G
*************************** 1. row ***************************
                  Db: myschema
                Name: e_daily
             Definer: jon@ghidora
           Time zone: SYSTEM
                Type: RECURRING
          Execute at: NULL
      Interval value: 1
      Interval field: DAY
              Starts: 2018-08-08 11:06:34
                Ends: NULL
              Status: ENABLED
          Originator: 1
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: utf8mb4_0900_ai_ci
```

To see events for a specific schema, use the `FROM` clause. For example, to see events for the `test` schema, use the following statement:

```
SHOW EVENTS FROM test;
```

The `LIKE` clause, if present, indicates which event names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, “Extensions to SHOW Statements”.

`SHOW EVENTS` output has these columns:

- `Db`

  The name of the schema (database) to which the event belongs.

- `Name`

  The name of the event.

- `Definer`

  The account of the user who created the event, in `'user_name'@'host_name'` format.

- `Time zone`

  The event time zone, which is the time zone used for scheduling the event and that is in effect within the event as it executes. The default value is `SYSTEM`.

- `Type`

  The event repetition type, either `ONE TIME` (transient) or `RECURRING` (repeating).

- `Execute At`

  For a one-time event, this is the `DATETIME` value specified in the `AT` clause of the `CREATE EVENT` statement used to create the event, or of the last `ALTER EVENT` statement that modified the event. The value shown in this column reflects the addition or subtraction of any `INTERVAL` value included in the event's `AT` clause. For example, if an event is created using `ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR`, and the event was created at 2018-02-09 14:05:30, the value shown in this column would be `'2018-02-10 20:05:30'`. If the event's timing is determined by an `EVERY` clause instead of an `AT` clause (that is, if the event is recurring), the value of this column is `NULL`.

- `Interval Value`

  For a recurring event, the number of intervals to wait between event executions. For a transient event, the value of this column is always `NULL`.

- `Interval Field`

  The time units used for the interval which a recurring event waits before repeating. For a transient event, the value of this column is always `NULL`.

- `Starts`

  The start date and time for a recurring event. This is displayed as a `DATETIME` value, and is `NULL` if no start date and time are defined for the event. For a transient event, this column is always `NULL`. For a recurring event whose definition includes a `STARTS` clause, this column contains the corresponding `DATETIME` value. As with the `Execute At` column, this value resolves any expressions used. If there is no `STARTS` clause affecting the timing of the event, this column is `NULL`

- `Ends`

  For a recurring event whose definition includes a `ENDS` clause, this column contains the corresponding `DATETIME` value. As with the `Execute At` column, this value resolves any expressions used. If there is no `ENDS` clause affecting the timing of the event, this column is `NULL`.

- `Status`

The event status. One of `ENABLED`, `DISABLED`, or `SLAVESIDE_DISABLED`. `SLAVESIDE_DISABLED` indicates that the creation of the event occurred on another MySQL server acting as a replication master and replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave. For more information, see Section 17.5.1.16, “Replication of Invoked Features”. information.

- `Originator`

  The server ID of the MySQL server on which the event was created; used in replication. This value may be updated by `ALTER EVENT` to the server ID of the server on which that statement occurs, if executed on a master server. The default value is 0.

- `character_set_client`

  The session value of the `character_set_client` system variable when the event was created.

- `collation_connection`

  The session value of the `collation_connection` system variable when the event was created.

- `Database Collation`

  The collation of the database with which the event is associated.

For more information about `SLAVESIDE_DISABLED` and the `Originator` column, see Section 17.5.1.16, “Replication of Invoked Features”.

Times displayed by `SHOW EVENTS` are given in the event time zone, as discussed in Section 24.4.4, “Event Metadata”.

Event information is also available from the `INFORMATION_SCHEMA EVENTS` table. See Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”.

The event action statement is not shown in the output of `SHOW EVENTS`. Use `SHOW CREATE EVENT` or the `INFORMATION_SCHEMA EVENTS` table.

13.7.7.19 SHOW FUNCTION CODE Syntax

```
SHOW FUNCTION CODE func_name
```

This statement is similar to `SHOW PROCEDURE CODE` but for stored functions. See Section 13.7.7.27, “SHOW PROCEDURE CODE Syntax”.

13.7.7.20 SHOW FUNCTION STATUS Syntax

```
SHOW FUNCTION STATUS
    [LIKE 'pattern' | WHERE expr]
```

This statement is similar to `SHOW PROCEDURE STATUS` but for stored functions. See Section 13.7.7.28, “SHOW PROCEDURE STATUS Syntax”.

13.7.7.21 SHOW GRANTS Syntax

```
SHOW GRANTS
    [FOR user_or_role
        [USING role [, role] ...]]

user_or_role: {
    user (see Section 6.2.4, “Specifying Account Names”)
  | role (see Section 6.2.5, “Specifying Role Names”.
}
```

This statement displays the privileges and roles that are assigned to a MySQL user account or role, in the form of `GRANT` statements that must be executed to duplicate the privilege and role assignments.

Note

To display nonprivilege information for MySQL accounts, use the `SHOW CREATE USER` statement. See Section 13.7.7.12, “SHOW CREATE USER Syntax”.

`SHOW GRANTS` requires the `SELECT` privilege for the `mysql` system database, except to display privileges and roles for the current user.

To name the account or role for `SHOW GRANTS`, use the same format as for the `GRANT` statement (for example, `'jeffrey'@'localhost'`):

```
mysql> SHOW GRANTS FOR 'jeffrey'@'localhost';
+-----------------------------------------------------------------+
| Grants for jeffrey@localhost                                    |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO `jeffrey`@`localhost`                     |
| GRANT SELECT, INSERT, UPDATE ON `db1`.* TO `jeffrey`@`localhost` |
+-----------------------------------------------------------------+
```

The host part, if omitted, defaults to `'%'`. For additional information about specifying account and role names, see Section 6.2.4, “Specifying Account Names”, and Section 6.2.5, “Specifying Role Names”.

To display the privileges granted to the current user (the account you are using to connect to the server), you can use any of the following statements:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

If `SHOW GRANTS FOR CURRENT_USER` (or any equivalent syntax) is used in definer context, such as within a stored procedure that executes with definer rather than invoker privileges, the grants displayed are those of the definer and not the invoker.

In MySQL 8.0 compared to previous series, `SHOW GRANTS` no longer displays `ALL PRIVILEGES` in its global-privileges output because the meaning of `ALL PRIVILEGES` at the global level varies depending on which dynamic privileges are defined. Instead, `SHOW GRANTS` explictly lists each granted global privilege:

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
+--------------------------------------------------------------------+
| Grants for root@localhost                                          |
+--------------------------------------------------------------------+
| GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, RELOAD,        |
| SHUTDOWN, PROCESS, FILE, REFERENCES, INDEX, ALTER, SHOW DATABASES, |
| SUPER, CREATE TEMPORARY TABLES, LOCK TABLES, EXECUTE, REPLICATION  |
| SLAVE, REPLICATION CLIENT, CREATE VIEW, SHOW VIEW, CREATE ROUTINE, |
```

```
| ALTER ROUTINE, CREATE USER, EVENT, TRIGGER, CREATE TABLESPACE,       |
| CREATE ROLE, DROP ROLE ON *.* TO `root`@`localhost` WITH GRANT       |
| OPTION                                                               |
| GRANT PROXY ON ''@'' TO 'root'@'localhost' WITH GRANT OPTION         |
+----------------------------------------------------------------------+
```

Applications that process `SHOW GRANTS` output should be adjusted accordingly.

At the global level, `GRANT OPTION` applies to all granted static global privileges if granted for any of them, but applies individually to granted dynamic privileges. `SHOW GRANTS` displays global privileges this way:

- One line listing all granted static privileges, if there are any, including `WITH GRANT OPTION` if appropriate.
- One line listing all granted dynamic privileges for which `GRANT OPTION` is granted, if there are any, including `WITH GRANT OPTION`.
- One line listing all granted dynamic privileges for which `GRANT OPTION` is not granted, if there are any, without `WITH GRANT OPTION`.

With the optional `USING` clause, `SHOW GRANTS` enables you to examine the privileges associated with roles for the user. Each role named in the `USING` clause must be granted to the user.

Suppose that user `u1` is assigned roles `r1` and `r2`, as follows:

```
CREATE ROLE 'r1', 'r2';
GRANT SELECT ON db1.* TO 'r1';
GRANT INSERT, UPDATE, DELETE ON db1.* TO 'r2';
CREATE USER 'u1'@'localhost' IDENTIFIED BY 'u1pass';
GRANT 'r1', 'r2' TO 'u1'@'localhost';
```

`SHOW GRANTS` without `USING` shows the granted roles:

```
mysql> SHOW GRANTS FOR 'u1'@'localhost';
+-------------------------------------------+
| Grants for u1@localhost                   |
+-------------------------------------------+
| GRANT USAGE ON *.* TO `u1`@`localhost`    |
| GRANT `r1`@`%`,`r2`@`%` TO `u1`@`localhost` |
+-------------------------------------------+
```

Adding a `USING` clause causes the statement to also display the privileges associated with each role named in the clause:

```
mysql> SHOW GRANTS FOR 'u1'@'localhost' USING 'r1';
+-------------------------------------------+
| Grants for u1@localhost                   |
+-------------------------------------------+
| GRANT USAGE ON *.* TO `u1`@`localhost`    |
| GRANT SELECT ON `db1`.* TO `u1`@`localhost` |
| GRANT `r1`@`%`,`r2`@`%` TO `u1`@`localhost` |
+-------------------------------------------+
mysql> SHOW GRANTS FOR 'u1'@'localhost' USING 'r2';
+-----------------------------------------------------------+
| Grants for u1@localhost                                   |
+-----------------------------------------------------------+
| GRANT USAGE ON *.* TO `u1`@`localhost`                    |
| GRANT INSERT, UPDATE, DELETE ON `db1`.* TO `u1`@`localhost` |
| GRANT `r1`@`%`,`r2`@`%` TO `u1`@`localhost`               |
```

```
+-------------------------------------------------------------+
mysql> SHOW GRANTS FOR 'u1'@'localhost' USING 'r1', 'r2';
+---------------------------------------------------------------------+
| Grants for u1@localhost                                             |
+---------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `u1`@`localhost`                              |
| GRANT SELECT, INSERT, UPDATE, DELETE ON `db1`.* TO `u1`@`localhost` |
| GRANT `r1`@`%`,`r2`@`%` TO `u1`@`localhost`                         |
+---------------------------------------------------------------------+
```

Note

A privilege granted to an account is always in effect, but a role is not. The active roles for an account can differ across and within sessions, depending on the value of the `activate_all_roles_on_login` system variable, the account default roles, and whether `SET ROLE` has been executed within a session.

MySQL 8.0.16 and higher supports partial revokes of global privileges, such that a global privilege can be restricted from applying to particular schemas (see Section 6.2.12, “Privilege Restriction Using Partial Revokes”). To indicate which global schema privileges have been revoked for particular schemas, `SHOW GRANTS` output includes `REVOKE` statements:

```
mysql> SET PERSIST partial_revokes = ON;
mysql> CREATE USER u1;
mysql> GRANT SELECT, INSERT, DELETE ON *.* TO u1;
mysql> REVOKE SELECT, INSERT ON mysql.* FROM u1;
mysql> REVOKE DELETE ON world.* FROM u1;
mysql> SHOW GRANTS FOR u1;
+-------------------------------------------------+
| Grants for u1@%                                 |
+-------------------------------------------------+
| GRANT SELECT, INSERT, DELETE ON *.* TO `u1`@`%`  |
| REVOKE SELECT, INSERT ON `mysql`.* FROM `u1`@`%` |
| REVOKE DELETE ON `world`.* FROM `u1`@`%`         |
+-------------------------------------------------+
```

`SHOW GRANTS` does not display privileges that are available to the named account but are granted to a different account. For example, if an anonymous account exists, the named account might be able to use its privileges, but `SHOW GRANTS` does not display them.

`SHOW GRANTS` displays mandatory roles named in the `mandatory_roles` system variable value as follows:

- `SHOW GRANTS` without a `FOR` clause displays privileges for the current user, and includes mandatory roles.
- `SHOW GRANTS FOR` *`user`* displays privileges for the named user, and does not include mandatory roles.

This behavior is for the benefit of applications that use the output of `SHOW GRANTS FOR` *`user`* to determine which privileges are granted explicitly to the named user. Were that output to include mandatory roles, it would be difficult to distinguish roles granted explicitly to the user from mandatory roles.

For the current user, applications can determine privileges with or without mandatory roles by using `SHOW GRANTS` or `SHOW GRANTS FOR CURRENT_USER`, respectively.

13.7.7.22 SHOW INDEX Syntax

```
SHOW [EXTENDED] {INDEX | INDEXES | KEYS}
    {FROM | IN} tbl_name
    [{FROM | IN} db_name]
    [WHERE expr]
```

`SHOW INDEX` returns table index information. The format resembles that of the `SQLStatistics` call in ODBC. This statement requires some privilege for any column in the table.

```
mysql> SHOW INDEX FROM City\G
*************************** 1. row ***************************
        Table: city
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: ID
    Collation: A
  Cardinality: 4188
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
      Visible: YES
   Expression: NULL
*************************** 2. row ***************************
        Table: city
   Non_unique: 1
     Key_name: CountryCode
 Seq_in_index: 1
  Column_name: CountryCode
    Collation: A
  Cardinality: 232
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
      Visible: YES
   Expression: NULL
```

An alternative to *`tbl_name FROM db_name`* syntax is *`db_name.tbl_name`*. These two statements are equivalent:

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

The optional `EXTENDED` keyword causes the output to include information about hidden indexes that MySQL uses internally and are not accessible by users.

The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, “Extensions to SHOW Statements”.

`SHOW INDEX` returns the following fields:

- `Table`

  The name of the table.

- `Non_unique`

0 if the index cannot contain duplicates, 1 if it can.

- `Key_name`

  The name of the index. If the index is the primary key, the name is always `PRIMARY`.

- `Seq_in_index`

  The column sequence number in the index, starting with 1.

- `Column_name`

  The column name. See also the description for the `Expression` column.

- `Collation`

  How the column is sorted in the index. This can have values `A` (ascending), `D` (descending), or `NULL` (not sorted).

- `Cardinality`

  An estimate of the number of unique values in the index. To update this number, run `ANALYZE TABLE` or (for `MyISAM` tables) `myisamchk -a`.

  `Cardinality` is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

- `Sub_part`

  The index prefix. That is, the number of indexed characters if the column is only partly indexed, `NULL` if the entire column is indexed.

  Note

  Prefix *limits* are measured in bytes. However, prefix *lengths* for index specifications in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements are interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

  For additional information about index prefixes, see Section 8.3.5, “Column Indexes”, and Section 13.1.15, “CREATE INDEX Syntax”.

- `Packed`

  Indicates how the key is packed. `NULL` if it is not.

- `Null`

  Contains `YES` if the column may contain `NULL` values and `''` if not.

- `Index_type`

  The index method used (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`).

- `Comment`

Information about the index not described in its own column, such as `disabled` if the index is disabled.

- `Index_comment`

  Any comment provided for the index with a `COMMENT` attribute when the index was created.

- `Visible`

  Whether the index is visible to the optimizer. See Section 8.3.12, “Invisible Indexes”.

- `Expression`

  MySQL 8.0.13 and higher supports functional key parts (see Functional Key Parts), which affects both the `Column_name` and `Expression` columns:

  - For a nonfunctional key part, `Column_name` indicates the column indexed by the key part and `Expression` is `NULL`.
  - For a functional key part, `Column_name` column is `NULL` and `Expression` indicates the expression for the key part.

Information about table indexes is also available from the `INFORMATION_SCHEMA STATISTICS` table. See Section 25.26, “The INFORMATION_SCHEMA STATISTICS Table”. The extended information about hidden indexes is available only using `SHOW EXTENDED INDEX`; it cannot be obtained from the `STATISTICS` table.

You can list a table's indexes with the `mysqlshow -k db_name tbl_name` command.

13.7.7.23 SHOW MASTER STATUS Syntax

```
SHOW MASTER STATUS
```

This statement provides status information about the binary log files of the master. It requires either the `SUPER` or `REPLICATION CLIENT` privilege.

Example:

```
mysql> SHOW MASTER STATUS\G
*************************** 1. row ***************************
             File: master-bin.000002
         Position: 1307
     Binlog_Do_DB: test
 Binlog_Ignore_DB: manual, mysql
Executed_Gtid_Set: 3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5
1 row in set (0.00 sec)
```

When global transaction IDs are in use, `Executed_Gtid_Set` shows the set of GTIDs for transactions that have been executed on the master. This is the same as the value for the `gtid_executed` system variable on this server, as well as the value for `Executed_Gtid_Set` in the output of `SHOW SLAVE STATUS` on this server.

13.7.7.24 SHOW OPEN TABLES Syntax

```
SHOW OPEN TABLES
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW OPEN TABLES` lists the non-`TEMPORARY` tables that are currently open in the table cache. See Section 8.4.3.1, “How MySQL Opens and Closes Tables”. The `FROM` clause, if present, restricts the tables shown to those present in the *`db_name`* database. The `LIKE` clause, if present, indicates which table names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, “Extensions to SHOW Statements”.

`SHOW OPEN TABLES` output has these columns:

- `Database`

  The database containing the table.

- `Table`

  The table name.

- `In_use`

  The number of table locks or lock requests there are for the table. For example, if one client acquires a lock for a table using `LOCK TABLE t1 WRITE`, `In_use` will be 1. If another client issues `LOCK TABLE t1 WRITE` while the table remains locked, the client will block waiting for the lock, but the lock request causes `In_use` to be 2. If the count is zero, the table is open but not currently being used. `In_use` is also increased by the `HANDLER ... OPEN` statement and decreased by `HANDLER ... CLOSE`.

- `Name_locked`

  Whether the table name is locked. Name locking is used for operations such as dropping or renaming tables.

If you have no privileges for a table, it does not show up in the output from `SHOW OPEN TABLES`.

13.7.7.25 SHOW PLUGINS Syntax

```
SHOW PLUGINS
```

`SHOW PLUGINS` displays information about server plugins.

Example of `SHOW PLUGINS` output:

```
mysql> SHOW PLUGINS\G
*************************** 1. row ***************************
   Name: binlog
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
*************************** 2. row ***************************
   Name: CSV
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
*************************** 3. row ***************************
   Name: MEMORY
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
*************************** 4. row ***************************
```

```
   Name: MyISAM
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
...
```

`SHOW PLUGINS` output has these columns:

- `Name`

  The name used to refer to the plugin in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`.

- `Status`

  The plugin status, one of `ACTIVE`, `INACTIVE`, `DISABLED`, `DELETING`, or `DELETED`.

- `Type`

  The type of plugin, such as `STORAGE ENGINE`, `INFORMATION_SCHEMA`, or `AUTHENTICATION`.

- `Library`

  The name of the plugin shared library file. This is the name used to refer to the plugin file in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`. This file is located in the directory named by the `plugin_dir` system variable. If the library name is `NULL`, the plugin is compiled in and cannot be uninstalled with `UNINSTALL PLUGIN`.

- `License`

  How the plugin is licensed (for example, `GPL`).

For plugins installed with `INSTALL PLUGIN`, the `Name` and `Library` values are also registered in the `mysql.plugin` system table.

For information about plugin data structures that form the basis of the information displayed by `SHOW PLUGINS`, see Section 29.2, “The MySQL Plugin API”.

Plugin information is also available from the `INFORMATION_SCHEMA.PLUGINS` table. See Section 25.18, “The INFORMATION_SCHEMA PLUGINS Table”.

13.7.7.26 SHOW PRIVILEGES Syntax

```
SHOW PRIVILEGES
```

`SHOW PRIVILEGES` shows the list of system privileges that the MySQL server supports. The privileges displayed include all static privileges, and all currently registered dynamic privileges.

```
mysql> SHOW PRIVILEGES\G
*************************** 1. row ***************************
Privilege: Alter
  Context: Tables
  Comment: To alter the table
*************************** 2. row ***************************
Privilege: Alter routine
  Context: Functions,Procedures
  Comment: To alter or drop stored functions/procedures
```

```
*************************** 3. row ***************************
Privilege: Create
  Context: Databases,Tables,Indexes
  Comment: To create new databases and tables
*************************** 4. row ***************************
Privilege: Create routine
  Context: Databases
  Comment: To use CREATE FUNCTION/PROCEDURE
*************************** 5. row ***************************
Privilege: Create temporary tables
  Context: Databases
  Comment: To use CREATE TEMPORARY TABLE
...
```

Privileges belonging to a specific user are displayed by the `SHOW GRANTS` statement. See Section 13.7.7.21, “SHOW GRANTS Syntax”, for more information.

13.7.7.27 SHOW PROCEDURE CODE Syntax

```
SHOW PROCEDURE CODE proc_name
```

This statement is a MySQL extension that is available only for servers that have been built with debugging support. It displays a representation of the internal implementation of the named stored procedure. A similar statement, `SHOW FUNCTION CODE`, displays information about stored functions (see Section 13.7.7.19, “SHOW FUNCTION CODE Syntax”).

To use either statement, you must have the global `SELECT` privilege.

If the named routine is available, each statement produces a result set. Each row in the result set corresponds to one “instruction” in the routine. The first column is `Pos`, which is an ordinal number beginning with 0. The second column is `Instruction`, which contains an SQL statement (usually changed from the original source), or a directive which has meaning only to the stored-routine handler.

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE p1 ()
    -> BEGIN
    ->   DECLARE fanta INT DEFAULT 55;
    ->   DROP TABLE t2;
    ->   LOOP
    ->     INSERT INTO t3 VALUES (fanta);
    ->     END LOOP;
    ->   END//
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW PROCEDURE CODE p1//
+-----+----------------------------------------+
| Pos | Instruction                            |
+-----+----------------------------------------+
|   0 | set fanta@0 55                         |
|   1 | stmt 9 "DROP TABLE t2"                 |
|   2 | stmt 5 "INSERT INTO t3 VALUES (fanta)" |
|   3 | jump 2                                 |
+-----+----------------------------------------+
4 rows in set (0.00 sec)
```

In this example, the nonexecutable `BEGIN` and `END` statements have disappeared, and for the `DECLARE variable_name` statement, only the executable part appears (the part where the default is assigned). For each statement that is taken from source, there is a code word `stmt` followed by a type (9 means `DROP`, 5 means `INSERT`, and so on). The final row contains an instruction `jump 2`, meaning `GOTO instruction #2`.

13.7.7.28 SHOW PROCEDURE STATUS Syntax

```
SHOW PROCEDURE STATUS
    [LIKE 'pattern' | WHERE expr]
```

This statement is a MySQL extension. It returns characteristics of a stored procedure, such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, `SHOW FUNCTION STATUS`, displays information about stored functions (see Section 13.7.7.20, "SHOW FUNCTION STATUS Syntax").

The `LIKE` clause, if present, indicates which procedure or function names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, "Extensions to SHOW Statements".

```
mysql> SHOW PROCEDURE STATUS LIKE 'sp1'\G
*************************** 1. row ***************************
                  Db: test
                Name: sp1
                Type: PROCEDURE
             Definer: testuser@localhost
            Modified: 2018-08-08 13:54:11
             Created: 2018-08-08 13:54:11
       Security_type: DEFINER
             Comment:
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: utf8mb4_0900_ai_ci
```

`character_set_client` is the session value of the `character_set_client` system variable when the routine was created. `collation_connection` is the session value of the `collation_connection` system variable when the routine was created. `Database Collation` is the collation of the database with which the routine is associated.

Stored routine information is also available from the `INFORMATION_SCHEMA PARAMETERS` and `ROUTINES` tables. See Section 25.16, "The INFORMATION_SCHEMA PARAMETERS Table", and Section 25.23, "The INFORMATION_SCHEMA ROUTINES Table".

13.7.7.29 SHOW PROCESSLIST Syntax

```
SHOW [FULL] PROCESSLIST
```

`SHOW PROCESSLIST` shows which threads are running. If you have the `PROCESS` privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MySQL account that you are using). If you do not use the `FULL` keyword, only the first 100 characters of each statement are shown in the `Info` field.

The `SHOW PROCESSLIST` statement is very useful if you get the "too many connections" error message and want to find out what is going on. MySQL reserves one extra connection to be used by accounts that have the `CONNECTION_ADMIN` or `SUPER` privilege, to ensure that administrators should always be able to connect and check the system (assuming that you are not giving this privilege to all your users).

Threads can be killed with the `KILL` statement. See Section 13.7.8.4, "KILL Syntax".

Example of `SHOW PROCESSLIST` output:

```
mysql> SHOW FULL PROCESSLIST\G
```

```
*************************** 1. row ***************************
Id: 1
User: system user
Host:
db: NULL
Command: Connect
Time: 1030455
State: Waiting for master to send event
Info: NULL
*************************** 2. row ***************************
Id: 2
User: system user
Host:
db: NULL
Command: Connect
Time: 1004
State: Has read all relay log; waiting for the slave
       I/O thread to update it
Info: NULL
*************************** 3. row ***************************
Id: 3112
User: replikator
Host: artemis:2204
db: NULL
Command: Binlog Dump
Time: 2144
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
*************************** 4. row ***************************
Id: 3113
User: replikator
Host: iconnect2:45781
db: NULL
Command: Binlog Dump
Time: 2086
State: Has sent all binlog to slave; waiting for binlog to be updated
Info: NULL
*************************** 5. row ***************************
Id: 3123
User: stefan
Host: localhost
db: apollon
Command: Query
Time: 0
State: NULL
Info: SHOW FULL PROCESSLIST
5 rows in set (0.00 sec)
```

`SHOW PROCESSLIST` output has these columns:

- `Id`

  The connection identifier. This is the same type of value displayed in the `ID` column of the `INFORMATION_SCHEMA PROCESSLIST` table, the `PROCESSLIST_ID` column of the Performance Schema `threads` table, and returned by the `CONNECTION_ID()` function.

- `User`

  The MySQL user who issued the statement. A value of `system user` refers to a nonclient thread spawned by the server to handle tasks internally. This could be the I/O or SQL thread used on replication slaves or a delayed-row handler. For `system user`, there is no host specified in the `Host` column. `unauthenticated user` refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet been done. `event_scheduler` refers to the thread that monitors scheduled events (see Section 24.4, “Using the Event Scheduler”).

Note

A `User` value of `system user` is distinct from the `SYSTEM_USER` privilege. The former designates internal threads. The latter distinguishes the system user and regular user account categories (see Section 6.2.11, “Account Categories”).

- `Host`

  The host name of the client issuing the statement (except for `system user`, for which there is no host). The host name for TCP/IP connections is reported in *`host_name`*`:`*`client_port`* format to make it easier to determine which client is doing what.

- `db`

  The default database, if one is selected; otherwise `NULL`.

- `Command`

  The type of command the thread is executing. For descriptions for thread commands, see Section 8.14, “Examining Thread Information”. The value of this column corresponds to the `COM_`*`xxx`* commands of the client/server protocol and `Com_`*`xxx`* status variables. See Section 5.1.10, “Server Status Variables”.

- `Time`

  The time in seconds that the thread has been in its current state. For a slave SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. See Section 17.2.2, “Replication Implementation Details”.

- `State`

  An action, event, or state that indicates what the thread is doing. Descriptions for `State` values can be found at Section 8.14, “Examining Thread Information”.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

  For the `SHOW PROCESSLIST` statement, the value of `State` is `NULL`.

- `Info`

  The statement the thread is executing, or `NULL` if it is not executing any statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a `CALL` statement executes a stored procedure that is executing a `SELECT` statement, the `Info` value shows the `SELECT` statement.

Process information is also available from the `mysqladmin processlist` command, the `INFORMATION_SCHEMA PROCESSLIST` table, and the Performance Schema `threads` table (see Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”, Section 25.19, “The INFORMATION_SCHEMA PROCESSLIST Table”, and Section 26.12.18.5, “The threads Table”). In contrast to the `INFORMATION_SCHEMA PROCESSLIST` table and `SHOW PROCESSLIST` statement, which have negative performance consequences because they require a mutex, access to `threads` does not require a mutex and has minimal impact on server performance. The `threads` table also shows information about background threads, which the `PROCESSLIST` table and `SHOW PROCESSLIST` do not. This means that `threads` can be used to monitor activity the other thread information sources cannot.

13.7.7.30 SHOW PROFILE Syntax

```
SHOW PROFILE [type [, type] ... ]
    [FOR QUERY n]
    [LIMIT row_count [OFFSET offset]]

type: {
    ALL
  | BLOCK IO
  | CONTEXT SWITCHES
  | CPU
  | IPC
  | MEMORY
  | PAGE FAULTS
  | SOURCE
  | SWAPS
}
```

The `SHOW PROFILE` and `SHOW PROFILES` statements display profiling information that indicates resource usage for statements executed during the course of the current session.

Note

The `SHOW PROFILE` and `SHOW PROFILES` statements are deprecated and will be removed in a future MySQL release. Use the Performance Schema instead; see Section 26.19.1, “Query Profiling Using Performance Schema”.

To control profiling, use the `profiling` session variable, which has a default value of 0 (`OFF`). Enable profiling by setting `profiling` to 1 or `ON`:

```
mysql> SET profiling = 1;
```

`SHOW PROFILES` displays a list of the most recent statements sent to the server. The size of the list is controlled by the `profiling_history_size` session variable, which has a default value of 15. The maximum value is 100. Setting the value to 0 has the practical effect of disabling profiling.

All statements are profiled except `SHOW PROFILE` and `SHOW PROFILES`, so you will find neither of those statements in the profile list. Malformed statements are profiled. For example, `SHOW PROFILING` is an illegal statement, and a syntax error occurs if you try to execute it, but it will show up in the profiling list.

`SHOW PROFILE` displays detailed information about a single statement. Without the `FOR QUERY n` clause, the output pertains to the most recently executed statement. If `FOR QUERY n` is included, `SHOW PROFILE` displays information for statement *`n`*. The values of *`n`* correspond to the `Query_ID` values displayed by `SHOW PROFILES`.

The `LIMIT row_count` clause may be given to limit the output to *`row_count`* rows. If `LIMIT` is given, `OFFSET offset` may be added to begin the output *`offset`* rows into the full set of rows.

By default, `SHOW PROFILE` displays `Status` and `Duration` columns. The `Status` values are like the `State` values displayed by `SHOW PROCESSLIST`, although there might be some minor differences in interpretion for the two statements for some status values (see Section 8.14, “Examining Thread Information”).

Optional *`type`* values may be specified to display specific additional types of information:

- `ALL` displays all information
- `BLOCK IO` displays counts for block input and output operations
- `CONTEXT SWITCHES` displays counts for voluntary and involuntary context switches

- `CPU` displays user and system CPU usage times
- `IPC` displays counts for messages sent and received
- `MEMORY` is not currently implemented
- `PAGE FAULTS` displays counts for major and minor page faults
- `SOURCE` displays the names of functions from the source code, together with the name and line number of the file in which the function occurs
- `SWAPS` displays swap counts

Profiling is enabled per session. When a session ends, its profiling information is lost.

```
mysql> SELECT @@profiling;
+-------------+
| @@profiling |
+-------------+
|           0 |
+-------------+
1 row in set (0.00 sec)

mysql> SET profiling = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE T1 (id INT);
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW PROFILES;
+----------+----------+--------------------------+
| Query_ID | Duration | Query                    |
+----------+----------+--------------------------+
|        0 | 0.000088 | SET PROFILING = 1        |
|        1 | 0.000136 | DROP TABLE IF EXISTS t1  |
|        2 | 0.011947 | CREATE TABLE t1 (id INT) |
+----------+----------+--------------------------+
3 rows in set (0.00 sec)

mysql> SHOW PROFILE;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| checking permissions | 0.000040 |
| creating table       | 0.000056 |
| After create         | 0.011363 |
| query end            | 0.000375 |
| freeing items        | 0.000089 |
| logging slow query   | 0.000019 |
| cleaning up          | 0.000005 |
+----------------------+----------+
7 rows in set (0.00 sec)

mysql> SHOW PROFILE FOR QUERY 1;
+--------------------+----------+
| Status             | Duration |
+--------------------+----------+
| query end          | 0.000107 |
| freeing items      | 0.000008 |
| logging slow query | 0.000015 |
| cleaning up        | 0.000006 |
+--------------------+----------+
```

```
4 rows in set (0.00 sec)

mysql> SHOW PROFILE CPU FOR QUERY 2;
+----------------------+----------+----------+------------+
| Status               | Duration | CPU_user | CPU_system |
+----------------------+----------+----------+------------+
| checking permissions | 0.000040 | 0.000038 |   0.000002 |
| creating table       | 0.000056 | 0.000028 |   0.000028 |
| After create         | 0.011363 | 0.000217 |   0.001571 |
| query end            | 0.000375 | 0.000013 |   0.000028 |
| freeing items        | 0.000089 | 0.000010 |   0.000014 |
| logging slow query   | 0.000019 | 0.000009 |   0.000010 |
| cleaning up          | 0.000005 | 0.000003 |   0.000002 |
+----------------------+----------+----------+------------+
7 rows in set (0.00 sec)
```

Note

Profiling is only partially functional on some architectures. For values that depend on the `getrusage()` system call, `NULL` is returned on systems such as Windows that do not support the call. In addition, profiling is per process and not per thread. This means that activity on threads within the server other than your own may affect the timing information that you see.

Profiling information is also available from the `INFORMATION_SCHEMA PROFILING` table. See Section 25.20, “The INFORMATION_SCHEMA PROFILING Table”. For example, the following queries are equivalent:

```
SHOW PROFILE FOR QUERY 2;

SELECT STATE, FORMAT(DURATION, 6) AS DURATION
FROM INFORMATION_SCHEMA.PROFILING
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

13.7.7.31 SHOW PROFILES Syntax

```
SHOW PROFILES
```

The `SHOW PROFILES` statement, together with `SHOW PROFILE`, displays profiling information that indicates resource usage for statements executed during the course of the current session. For more information, see Section 13.7.7.30, “SHOW PROFILE Syntax”.

Note

The `SHOW PROFILE` and `SHOW PROFILES` statements are deprecated and will be removed in a future MySQL release. Use the Performance Schema instead; see Section 26.19.1, “Query Profiling Using Performance Schema”.

13.7.7.32 SHOW RELAYLOG EVENTS Syntax

```
SHOW RELAYLOG EVENTS
    [IN 'log_name']
    [FROM pos]
    [LIMIT [offset,] row_count]
    [channel_option]

channel_option:
    FOR CHANNEL channel
```

Shows the events in the relay log of a replication slave. If you do not specify `'log_name'`, the first relay log is displayed. This statement has no effect on the master. `SHOW RELAYLOG EVENTS` requires the `REPLICATION SLAVE` privilege.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.2.10, "SELECT Syntax".

Note

Issuing a `SHOW RELAYLOG EVENTS` with no `LIMIT` clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the relay log (including all statements modifying data that have been received by the slave).

The optional `FOR CHANNEL channel` clause enables you to name which replication channel the statement applies to. Providing a `FOR CHANNEL channel` clause applies the statement to a specific replication channel. If no channel is named and no extra channels exist, the statement applies to the default channel.

When using multiple replication channels, if a `SHOW RELAYLOG EVENTS` statement does not have a channel defined using a `FOR CHANNEL channel` clause an error is generated. See Section 17.2.3, "Replication Channels" for more information.

`SHOW RELAYLOG EVENTS` displays the following fields for each event in the relay log:

- `Log_name`

  The name of the file that is being listed.

- `Pos`

  The position at which the event occurs.

- `Event_type`

  An identifier that describes the event type.

- `Server_id`

  The server ID of the server on which the event originated.

- `End_log_pos`

  The value of `End_log_pos` for this event in the master's binary log.

- `Info`

  More detailed information about the event type. The format of this information depends on the event type.

Note

Some events relating to the setting of user and system variables are not included in the output from `SHOW RELAYLOG EVENTS`. To get complete coverage of events within a relay log, use `mysqlbinlog`.

13.7.7.33 SHOW SLAVE HOSTS Syntax

```
SHOW SLAVE HOSTS
```

Displays a list of replication slaves currently registered with the master. `SHOW SLAVE HOSTS` requires the `REPLICATION SLAVE` privilege.

`SHOW SLAVE HOSTS` should be executed on a server that acts as a replication master. The statement displays information about servers that are or have been connected as replication slaves, with each row of the result corresponding to one slave server, as shown here:

```
mysql> SHOW SLAVE HOSTS;
+------------+-----------+------+-----------+--------------------------------------+
| Server_id  | Host      | Port | Master_id | Slave_UUID                           |
+------------+-----------+------+-----------+--------------------------------------+
|  192168010 | iconnect2 | 3306 | 192168011 | 14cb6624-7f93-11e0-b2c0-c80aa9429562 |
| 1921680101 | athena    | 3306 | 192168011 | 07af4990-f41f-11df-a566-7ac56fdaf645 |
+------------+-----------+------+-----------+--------------------------------------+
```

- `Server_id`: The unique server ID of the slave server, as configured in the slave server's option file, or on the command line with `--server-id=value`.
- `Host`: The host name of the slave server as specified on the slave with the `--report-host` option. This can differ from the machine name as configured in the operating system.
- `User`: The slave server user name as, specified on the slave with the `--report-user` option. Statement output includes this column only if the master server is started with the `--show-slave-auth-info` option.
- `Password`: The slave server password as, specified on the slave with the `--report-password` option. Statement output includes this column only if the master server is started with the `--show-slave-auth-info` option.
- `Port`: The port on the master to which the slave server is listening, as specified on the slave with the `--report-port` option.

  A zero in this column means that the slave port (`--report-port`) was not set.
- `Master_id`: The unique server ID of the master server that the slave server is replicating from. This is the server ID of the server on which `SHOW SLAVE HOSTS` is executed, so this same value is listed for each row in the result.
- `Slave_UUID`: The globally unique ID of this slave, as generated on the slave and found in the slave's `auto.cnf` file.

13.7.7.34 SHOW SLAVE STATUS Syntax

```
SHOW SLAVE STATUS [FOR CHANNEL channel]
```

This statement provides status information on essential parameters of the slave threads. It requires either the `SUPER` or `REPLICATION CLIENT` privilege.

`SHOW SLAVE STATUS` is nonblocking. When run concurrently with `STOP SLAVE`, `SHOW SLAVE STATUS` returns without waiting for `STOP SLAVE` to finish shutting down the slave SQL thread or slave I/O thread (or both). This permits use in monitoring and other applications where getting an immediate response from `SHOW SLAVE STATUS` more important than ensuring that it returned the latest data.

If you issue this statement using the `mysql` client, you can use a `\G` statement terminator rather than a semicolon to obtain a more readable vertical layout:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
```

```
               Slave_IO_State: Waiting for master to send event
                  Master_Host: localhost
                  Master_User: repl
                  Master_Port: 13000
                Connect_Retry: 60
              Master_Log_File: master-bin.000002
          Read_Master_Log_Pos: 1307
               Relay_Log_File: slave-relay-bin.000003
                Relay_Log_Pos: 1508
        Relay_Master_Log_File: master-bin.000002
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 1307
              Relay_Log_Space: 1858
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: 3e11fa47-71ca-11e1-9e33-c80aa9429562
             Master_Info_File:
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Reading event from the relay log
           Master_Retry_Count: 10
                  Master_Bind:
      Last_IO_Error_Timestamp:
     Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set: 3e11fa47-71ca-11e1-9e33-c80aa9429562:1-5
            Executed_Gtid_Set: 3e11fa47-71ca-11e1-9e33-c80aa9429562:1-5
                Auto_Position: 1
         Replicate_Rewrite_DB:
                 Channel_name:
           Master_TLS_Version: TLSv1.2
       Master_public_key_path: public_key.pem
        Get_master_public_key: 0
```

The Performance Schema provides tables that expose replication information. This is similar to the information available from the `SHOW SLAVE STATUS` statement, but represented in table form. For details, see Section 26.12.11, “Performance Schema Replication Tables”.

The following list describes the fields returned by `SHOW SLAVE STATUS`. For additional information about interpreting their meanings, see Section 17.1.7.1, “Checking Replication Status”.

- `Slave_IO_State`

  A copy of the `State` field of the `SHOW PROCESSLIST` output for the slave I/O thread. This tells you what the thread is doing: trying to connect to the master, waiting for events from the master, reconnecting to the master, and so on. For a listing of possible states, see Section 8.14.4, “Replication Slave I/O Thread States”.

- `Master_Host`

  The master host that the slave is connected to.

- `Master_User`

  The user name of the account used to connect to the master.

- `Master_Port`

  The port used to connect to the master.

- `Connect_Retry`

  The number of seconds between connect retries (default 60). This can be set with the `CHANGE MASTER TO` statement.

- `Master_Log_File`

  The name of the master binary log file from which the I/O thread is currently reading.

- `Read_Master_Log_Pos`

  The position in the current master binary log file up to which the I/O thread has read.

- `Relay_Log_File`

  The name of the relay log file from which the SQL thread is currently reading and executing.

- `Relay_Log_Pos`

  The position in the current relay log file up to which the SQL thread has read and executed.

- `Relay_Master_Log_File`

  The name of the master binary log file containing the most recent event executed by the SQL thread.

- `Slave_IO_Running`

  Whether the I/O thread is started and has connected successfully to the master. Internally, the state of this thread is represented by one of the following three values:

  - **MYSQL_SLAVE_NOT_RUN.** The slave I/O thread is not running. For this state, `Slave_IO_Running` is `No`.
  - **MYSQL_SLAVE_RUN_NOT_CONNECT.** The slave I/O thread is running, but is not connected to a replication master. For this state, `Slave_IO_Running` is `Connecting`.
  - **MYSQL_SLAVE_RUN_CONNECT.** The slave I/O thread is running, and is connected to a replication master. For this state, `Slave_IO_Running` is `Yes`.

  The value of the `Slave_running` system status variable corresponds with this value.

- `Slave_SQL_Running`

  Whether the SQL thread is started.

- `Replicate_Do_DB`, `Replicate_Ignore_DB`

  The names of any databases that were specified with the `--replicate-do-db` and `--replicate-ignore-db` options, or the `CHANGE REPLICATION FILTER` statement. If the `FOR CHANNEL` clause was used, the channel specific replication filters are shown. Otherwise, the replication filters for every replication channel are shown.

- `Replicate_Do_Table`, `Replicate_Ignore_Table`, `Replicate_Wild_Do_Table`, `Replicate_Wild_Ignore_Table`

  The names of any tables that were specified with the `--replicate-do-table`, `--replicate-ignore-table`, `--replicate-wild-do-table`, and `--replicate-wild-ignore-table` options, or the `CHANGE REPLICATION FILTER` statement. If the `FOR CHANNEL` clause was used, the channel specific replication filters are shown. Otherwise, the replication filters for every replication channel are shown.

- `Last_Errno`, `Last_Error`

  These columns are aliases for `Last_SQL_Errno` and `Last_SQL_Error`.

  Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.

  Note

  When the slave SQL thread receives an error, it reports the error first, then stops the SQL thread. This means that there is a small window of time during which `SHOW SLAVE STATUS` shows a nonzero value for `Last_SQL_Errno` even though `Slave_SQL_Running` still displays `Yes`.

- `Skip_Counter`

  The current value of the `sql_slave_skip_counter` system variable. See Section 13.4.2.5, “SET GLOBAL sql_slave_skip_counter Syntax”.

- `Exec_Master_Log_Pos`

  The position in the current master binary log file to which the SQL thread has read and executed, marking the start of the next transaction or event to be processed. You can use this value with the `CHANGE MASTER TO` statement's `MASTER_LOG_POS` option when starting a new slave from an existing slave, so that the new slave reads from this point. The coordinates given by (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`) in the master's binary log correspond to the coordinates given by (`Relay_Log_File`, `Relay_Log_Pos`) in the relay log.

  Inconsistencies in the sequence of transactions from the relay log which have been executed can cause this value to be a “low-water mark”. In other words, transactions appearing before the position are guaranteed to have committed, but transactions after the position may have committed or not. If these gaps need to be corrected, use `START SLAVE UNTIL SQL_AFTER_MTS_GAPS`. See Section 17.5.1.33, “Replication and Transaction Inconsistencies” for more information.

- `Relay_Log_Space`

  The total combined size of all existing relay log files.

- `Until_Condition`, `Until_Log_File`, `Until_Log_Pos`

The values specified in the `UNTIL` clause of the `START SLAVE` statement.

`Until_Condition` has these values:

- `None` if no `UNTIL` clause was specified
- `Master` if the slave is reading until a given position in the master's binary log
- `Relay` if the slave is reading until a given position in its relay log
- `SQL_BEFORE_GTIDS` if the slave SQL thread is processing transactions until it has reached the first transaction whose GTID is listed in the `gtid_set`.
- `SQL_AFTER_GTIDS` if the slave threads are processing all transactions until the last transaction in the `gtid_set` has been processed by both threads.
- `SQL_AFTER_MTS_GAPS` if a multithreaded slave's SQL threads are running until no more gaps are found in the relay log.

`Until_Log_File` and `Until_Log_Pos` indicate the log file name and position that define the coordinates at which the SQL thread stops executing.

For more information on `UNTIL` clauses, see Section 13.4.2.6, “START SLAVE Syntax”.

- `Master_SSL_Allowed`, `Master_SSL_CA_File`, `Master_SSL_CA_Path`, `Master_SSL_Cert`, `Master_SSL_Cipher`, `Master_SSL_CRL_File`, `Master_SSL_CRL_Path`, `Master_SSL_Key`, `Master_SSL_Verify_Server_Cert`

  These fields show the SSL parameters used by the slave to connect to the master, if any.

  `Master_SSL_Allowed` has these values:

  - `Yes` if an SSL connection to the master is permitted
  - `No` if an SSL connection to the master is not permitted
  - `Ignored` if an SSL connection is permitted but the slave server does not have SSL support enabled

  The values of the other SSL-related fields correspond to the values of the `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_CIPHER`, `MASTER_SSL_CRL`, `MASTER_SSL_CRLPATH`, `MASTER_SSL_KEY`, and `MASTER_SSL_VERIFY_SERVER_CERT` options to the `CHANGE MASTER TO` statement. See Section 13.4.2.1, “CHANGE MASTER TO Syntax”.

- `Seconds_Behind_Master`

  This field is an indication of how “late” the slave is:

  - When the slave is actively processing updates, this field shows the difference between the current timestamp on the slave and the original timestamp logged on the master for the event currently being processed on the slave.
  - When no event is currently being processed on the slave, this value is 0.

  In essence, this field measures the time difference in seconds between the slave SQL thread and the slave I/O thread. If the network connection between master and slave is fast, the slave I/O thread is very close to the master, so this field is a good approximation of how late the slave SQL thread is compared to the master. If the network is slow, this is *not* a good approximation; the slave SQL thread may quite

often be caught up with the slow-reading slave I/O thread, so `Seconds_Behind_Master` often shows a value of 0, even if the I/O thread is late compared to the master. In other words, *this column is useful only for fast networks*.

This time difference computation works even if the master and slave do not have identical clock times, provided that the difference, computed when the slave I/O thread starts, remains constant from then on. Any changes—including NTP updates—can lead to clock skews that can make calculation of `Seconds_Behind_Master` less reliable.

In MySQL 8.0, this field is `NULL` (undefined or unknown) if the slave SQL thread is not running, or if the SQL thread has consumed all of the relay log and the slave I/O thread is not running. (In older versions of MySQL, this field was `NULL` if the slave SQL thread or the slave I/O thread was not running or was not connected to the master.) If the I/O thread is running but the relay log is exhausted, `Seconds_Behind_Master` is set to 0.

The value of `Seconds_Behind_Master` is based on the timestamps stored in events, which are preserved through replication. This means that if a master M1 is itself a slave of M0, any event from M1's binary log that originates from M0's binary log has M0's timestamp for that event. This enables MySQL to replicate `TIMESTAMP` successfully. However, the problem for `Seconds_Behind_Master` is that if M1 also receives direct updates from clients, the `Seconds_Behind_Master` value randomly fluctuates because sometimes the last event from M1 originates from M0 and sometimes is the result of a direct update on M1.

When using a multithreaded slave, you should keep in mind that this value is based on `Exec_Master_Log_Pos`, and so may not reflect the position of the most recently committed transaction.

- `Last_IO_Errno`, `Last_IO_Error`

  The error number and error message of the most recent error that caused the I/O thread to stop. An error number of 0 and message of the empty string mean “no error.” If the `Last_IO_Error` value is not empty, the error values also appear in the slave's error log.

  I/O error information includes a timestamp showing when the most recent I/O thread error occurred. This timestamp uses the format *`YYMMDD hh:mm:ss`*, and appears in the `Last_IO_Error_Timestamp` column.

  Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.

- `Last_SQL_Errno`, `Last_SQL_Error`

  The error number and error message of the most recent error that caused the SQL thread to stop. An error number of 0 and message of the empty string mean “no error.” If the `Last_SQL_Error` value is not empty, the error values also appear in the slave's error log.

  If the slave is multithreaded, the SQL thread is the coordinator for worker threads. In this case, the `Last_SQL_Error` field shows exactly what the `Last_Error_Message` column in the Performance Schema `replication_applier_status_by_coordinator` table shows. The field value is modified to suggest that there may be more failures in the other worker threads which can be seen in the `replication_applier_status_by_worker` table that shows each worker thread's status. If that table is not available, the slave error log can be used. The log or the `replication_applier_status_by_worker` table should also be used to learn more about the failure shown by `SHOW SLAVE STATUS` or the coordinator table.

SQL error information includes a timestamp showing when the most recent SQL thread error occurred. This timestamp uses the format *`YYMMDD hh:mm:ss`*, and appears in the `Last_SQL_Error_Timestamp` column.

Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.

In MySQL 8.0, all error codes and messages displayed in the `Last_SQL_Errno` and `Last_SQL_Error` columns correspond to error values listed in Section B.3.1, “Server Error Message Reference”. This was not always true in previous versions. (Bug #11760365, Bug #52768)

- `Replicate_Ignore_Server_Ids`

  Any server IDs that have been specified using the `IGNORE_SERVER_IDS` option of the `CHANGE MASTER TO` statement, so that the slave ignores events from these servers. This option is used in a circular or other multi-master replication setup when one of the servers is removed. If any server IDs have been set in this way, a comma-delimited list of one or more numbers is shown. If no server IDs have been set, the field is blank.

  Note

  The `Ignored_server_ids` value in the `slave_master_info` table also shows the server IDs to be ignored, but as a space-delimited list, preceded by the total number of server IDs to be ignored. For example, if a `CHANGE MASTER TO` statement containing the `IGNORE_SERVER_IDS = (2,6,9)` option has been issued to tell a slave to ignore masters having the server ID 2, 6, or 9, that information appears as shown here:

  ```
  Replicate_Ignore_Server_Ids: 2, 6, 9
  ```

  ```
  Ignored_server_ids: 3, 2, 6, 9
  ```

  `Replicate_Ignore_Server_Ids` filtering is performed by the I/O thread, rather than by the SQL thread, which means that events which are filtered out are not written to the relay log. This differs from the filtering actions taken by server options such `--replicate-do-table`, which apply to the SQL thread.

  Note

  From MySQL 8.0.3, a deprecation warning is issued if `SET GTID_MODE=ON` is issued when any channel has existing server IDs set with `IGNORE_SERVER_IDS`. Before starting GTID-based replication, use `SHOW_SLAVE_STATUS` to check for and clear all ignored server ID lists on the servers involved. You can clear a list by issuing a `CHANGE MASTER TO` statement containing the `IGNORE_SERVER_IDS` option with an empty list.

- `Master_Server_Id`

  The `server_id` value from the master.

- `Master_UUID`

  The `server_uuid` value from the master.

- `Master_Info_File`

The location of the `master.info` file, if a file rather than a table is used for the slave's master info repository. The use of a file for the master info log has been superseded by crash-safe slave tables, and the `master_info_repository=FILE` setting is deprecated.

- `SQL_Delay`

  The number of seconds that the slave must lag the master.

- `SQL_Remaining_Delay`

  When `Slave_SQL_Running_State` is `Waiting until MASTER_DELAY seconds after master executed event`, this field contains the number of delay seconds remaining. At other times, this field is `NULL`.

- `Slave_SQL_Running_State`

  The state of the SQL thread (analogous to `Slave_IO_State`). The value is identical to the `State` value of the SQL thread as displayed by `SHOW PROCESSLIST`. Section 8.14.5, "Replication Slave SQL Thread States", provides a listing of possible states

- `Master_Retry_Count`

  The number of times the slave can attempt to reconnect to the master in the event of a lost connection. This value can be set using the `MASTER_RETRY_COUNT` option of the `CHANGE MASTER TO` statement (preferred) or the older `--master-retry-count` server option (still supported for backward compatibility).

- `Master_Bind`

  The network interface that the slave is bound to, if any. This is set using the `MASTER_BIND` option for the `CHANGE MASTER TO` statement.

- `Last_IO_Error_Timestamp`

  A timestamp in *`YYMMDD hh:mm:ss`* format that shows when the most recent I/O error took place.

- `Last_SQL_Error_Timestamp`

  A timestamp in *`YYMMDD hh:mm:ss`* format that shows when the most recent SQL error occurred.

- `Retrieved_Gtid_Set`

  The set of global transaction IDs corresponding to all transactions received by this slave. Empty if GTIDs are not in use. See GTID Sets for more information.

  This is the set of all GTIDs that exist or have existed in the relay logs. Each GTID is added as soon as the `Gtid_log_event` is received. This can cause partially transmitted transactions to have their GTIDs included in the set.

  When all relay logs are lost due to executing `RESET SLAVE` or `CHANGE MASTER TO`, or due to the effects of the `--relay-log-recovery` option, the set is cleared. When `relay_log_purge = 1`, the newest relay log is always kept, and the set is not cleared.

- `Executed_Gtid_Set`

  The set of global transaction IDs written in the binary log. This is the same as the value for the global `gtid_executed` system variable on this server, as well as the value for `Executed_Gtid_Set` in the

output of `SHOW MASTER STATUS` on this server. Empty if GTIDs are not in use. See GTID Sets for more information.

- `Auto_Position`

  1 if autopositioning is in use; otherwise 0.

- `Replicate_Rewrite_DB`

  The `Replicate_Rewrite_DB` value displays any replication filtering rules that were specified. For example, if the following replication filter rule was set:

  ```
  CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB=((db1,db2), (db3,db4));
  ```

  the `Replicate_Rewrite_DB` value displays:

  ```
  Replicate_Rewrite_DB: (db1,db2),(db3,db4)
  ```

  For more information, see Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”.

- `Channel_name`

  The replication channel which is being displayed. There is always a default replication channel, and more replication channels can be added. See Section 17.2.3, “Replication Channels” for more information.

- `Master_TLS_Version`

  The TLS version used on the master. For TLS version information, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `Master_public_key_path`

  The path name to a file containing a slave-side copy of the public key required by the master for RSA key pair-based password exchange. The file must be in PEM format. This column applies to slaves that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin.

  If `Master_public_key_path` is given and specifies a valid public key file, it takes precedence over `Get_master_public_key`.

- `Get_master_public_key`

  Whether to request from the master the public key required for RSA key pair-based password exchange. This column applies to slaves that authenticate with the `caching_sha2_password` authentication plugin. For that plugin, the master does not send the public key unless requested.

  If `Master_public_key_path` is given and specifies a valid public key file, it takes precedence over `Get_master_public_key`.

13.7.7.35 SHOW STATUS Syntax

```
SHOW [GLOBAL | SESSION] STATUS
    [LIKE 'pattern' | WHERE expr]
```

`SHOW STATUS` provides server status information (see Section 5.1.10, “Server Status Variables”). This statement does not require any privilege. It requires only the ability to connect to the server.

Status variable information is also available from these sources:

- Performance Schema tables. See Section 26.12.14, “Performance Schema Status Variable Tables”.
- The `mysqladmin extended-status` command. See Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”.

For `SHOW STATUS`, a `LIKE` clause, if present, indicates which variable names to match. A `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, “Extensions to SHOW Statements”.

`SHOW STATUS` accepts an optional `GLOBAL` or `SESSION` variable scope modifier:

- With a `GLOBAL` modifier, the statement displays the global status values. A global status variable may represent status for some aspect of the server itself (for example, `Aborted_connects`), or the aggregated status over all connections to MySQL (for example, `Bytes_received` and `Bytes_sent`). If a variable has no global value, the session value is displayed.
- With a `SESSION` modifier, the statement displays the status variable values for the current connection. If a variable has no session value, the global value is displayed. `LOCAL` is a synonym for `SESSION`.
- If no modifier is present, the default is `SESSION`.

The scope for each status variable is listed at Section 5.1.10, “Server Status Variables”.

Each invocation of the `SHOW STATUS` statement uses an internal temporary table and increments the global `Created_tmp_tables` value.

Partial output is shown here. The list of names and values may differ for your server. The meaning of each variable is given in Section 5.1.10, “Server Status Variables”.

```
mysql> SHOW STATUS;
+--------------------------+------------+
| Variable_name            | Value      |
+--------------------------+------------+
| Aborted_clients          | 0          |
| Aborted_connects         | 0          |
| Bytes_received           | 155372598  |
| Bytes_sent               | 1176560426 |
| Connections              | 30023      |
| Created_tmp_disk_tables  | 0          |
| Created_tmp_tables       | 8340       |
| Created_tmp_files        | 60         |
...
| Open_tables              | 1          |
| Open_files               | 2          |
| Open_streams             | 0          |
| Opened_tables            | 44600      |
| Questions                | 2026873    |
...
| Table_locks_immediate    | 1920382    |
| Table_locks_waited       | 0          |
| Threads_cached           | 0          |
| Threads_created          | 30022      |
| Threads_connected        | 1          |
| Threads_running          | 1          |
| Uptime                   | 80380      |
+--------------------------+------------+
```

With a `LIKE` clause, the statement displays only rows for those variables with names that match the pattern:

```
mysql> SHOW STATUS LIKE 'Key%';
```

```
+--------------------+----------+
| Variable_name      | Value    |
+--------------------+----------+
| Key_blocks_used    | 14955    |
| Key_read_requests  | 96854827 |
| Key_reads          | 162040   |
| Key_write_requests | 7589728  |
| Key_writes         | 3813196  |
+--------------------+----------+
```

13.7.7.36 SHOW TABLE STATUS Syntax

```
SHOW TABLE STATUS
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW TABLE STATUS` works likes `SHOW TABLES`, but provides a lot of information about each non-`TEMPORARY` table. You can also get this list using the `mysqlshow --status db_name` command. The `LIKE` clause, if present, indicates which table names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, “Extensions to SHOW Statements”.

This statement also displays information about views.

`SHOW TABLE STATUS` output has these columns:

- `Name`

  The name of the table.

- `Engine`

  The storage engine for the table. See Chapter 15, *The InnoDB Storage Engine*, and Chapter 16, *Alternative Storage Engines*.

  For partitioned tables, `Engine` shows the name of the storage engine used by all partitions.

- `Version`

  This column is unused. With the removal of `.frm` files in MySQL 8.0, this column now reports a hardcoded value of `10`, which is the last `.frm` file version used in MySQL 5.7.

- `Row_format`

  The row-storage format (`Fixed`, `Dynamic`, `Compressed`, `Redundant`, `Compact`). For `MyISAM` tables, `Dynamic` corresponds to what `myisamchk -dvv` reports as `Packed`.

- `Rows`

  The number of rows. Some storage engines, such as `MyISAM`, store the exact count. For other storage engines, such as `InnoDB`, this value is an approximation, and may vary from the actual value by as much as 40% to 50%. In such cases, use `SELECT COUNT(*)` to obtain an accurate count.

  The `Rows` value is `NULL` for `INFORMATION_SCHEMA` tables.

  For `InnoDB` tables, the row count is only a rough estimate used in SQL optimization. (This is also true if the `InnoDB` table is partitioned.)

- `Avg_row_length`

The average row length.

- `Data_length`

  For `MyISAM`, `Data_length` is the length of the data file, in bytes.

  For `InnoDB`, `Data_length` is the approximate amount of memory allocated for the clustered index, in bytes. Specifically, it is the clustered index size, in pages, multiplied by the `InnoDB` page size.

  Refer to the notes at the end of this section for information regarding other storage engines.

- `Max_data_length`

  For `MyISAM`, `Max_data_length` is maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

  Unused for `InnoDB`.

  Refer to the notes at the end of this section for information regarding other storage engines.

- `Index_length`

  For `MyISAM`, `Index_length` is the length of the index file, in bytes.

  For `InnoDB`, `Index_length` is the approximate amount of memory allocated for non-clustered indexes, in bytes. Specifically, it is the sum of non-clustered index sizes, in pages, multiplied by the `InnoDB` page size.

  Refer to the notes at the end of this section for information regarding other storage engines.

- `Data_free`

  The number of allocated but unused bytes.

  `InnoDB` tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table. Free space means the number of bytes in completely free extents minus a safety margin. Even if free space displays as 0, it may be possible to insert rows as long as new extents need not be allocated.

  For NDB Cluster, `Data_free` shows the space allocated on disk for, but not used by, a Disk Data table or fragment on disk. (In-memory data resource usage is reported by the `Data_length` column.)

  For partitioned tables, this value is only an estimate and may not be absolutely correct. A more accurate method of obtaining this information in such cases is to query the `INFORMATION_SCHEMA PARTITIONS` table, as shown in this example:

  ```
  SELECT SUM(DATA_FREE)
      FROM  INFORMATION_SCHEMA.PARTITIONS
      WHERE TABLE_SCHEMA = 'mydb'
      AND   TABLE_NAME   = 'mytable';
  ```

  For more information, see Section 25.17, “The INFORMATION_SCHEMA PARTITIONS Table”.

- `Auto_increment`

  The next `AUTO_INCREMENT` value.

- `Create_time`

  When the table was created.

- `Update_time`

  When the data file was last updated. For some storage engines, this value is `NULL`. For example, `InnoDB` stores multiple tables in its system tablespace and the data file timestamp does not apply. Even with file-per-table mode with each `InnoDB` table in a separate `.ibd` file, change buffering can delay the write to the data file, so the file modification time is different from the time of the last insert, update, or delete. For `MyISAM`, the data file timestamp is used; however, on Windows the timestamp is not updated by updates, so the value is inaccurate.

  `Update_time` displays a timestamp value for the last `UPDATE`, `INSERT`, or `DELETE` performed on `InnoDB` tables that are not partitioned. For MVCC, the timestamp value reflects the `COMMIT` time, which is considered the last update time. Timestamps are not persisted when the server is restarted or when the table is evicted from the `InnoDB` data dictionary cache.

- `Check_time`

  When the table was last checked. Not all storage engines update this time, in which case, the value is always `NULL`.

  For partitioned `InnoDB` tables, `Check_time` is always `NULL`.

- `Collation`

  The table default collation. The output does not explicitly list the table default character set, but the collation name begins with the character set name.

- `Checksum`

  The live checksum value, if any.

- `Create_options`

  Extra options used with `CREATE TABLE`. The original options from when `CREATE TABLE` was executed are retained and the options reported here may differ from the active table settings and options.

  For `InnoDB` tables, the actual `ROW_FORMAT` and `KEY_BLOCK_SIZE` options are reported. Prior to MySQL 8.0, `Create_options` reports the originally supplied `ROW_FORMAT` and `KEY_BLOCK_SIZE`. For more information, see Section 13.1.20, “CREATE TABLE Syntax”.

  `Create_options` shows `partitioned` if the table is partitioned. It also shows the `ENCRYPTION` option specified when creating or altering a file-per-table tablespace. This column does not show the encryption option specified when creating or altering a general tablespace. The `ENCRYPTION` column of the `INNODB_TABLESPACES` table is applicable to both file-per-table and general tablespaces.

- `Comment`

  The comment used when creating the table (or information as to why MySQL could not access the table information).

Notes

- For `NDB` tables, the output of this statement shows appropriate values for the `Avg_row_length` and `Data_length` columns, with the exception that `BLOB` columns are not taken into account.

- For NDB tables, Data_length includes data stored in main memory only; the Max_data_length and Data_free columns apply to Disk Data.
- For NDB Cluster Disk Data tables, Max_data_length shows the space allocated for the disk part of a Disk Data table or fragment. (In-memory data resource usage is reported by the Data_length column.)
- For MEMORY tables, the Data_length, Max_data_length, and Index_length values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.
- For views, most columns displayed by SHOW TABLE STATUS are 0 or NULL except that Name indicates the view name, Create_time indicates the creation time, and Comment says VIEW.

Table information is also available from the INFORMATION_SCHEMA TABLES table. See Section 25.30, "The INFORMATION_SCHEMA TABLES Table".

13.7.7.37 SHOW TABLES Syntax

```
SHOW [EXTENDED] [FULL] TABLES
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW TABLES lists the non-TEMPORARY tables in a given database. You can also get this list using the mysqlshow *db_name* command. The LIKE clause, if present, indicates which table names to match. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 25.42, "Extensions to SHOW Statements".

Matching performed by the LIKE clause is dependent on the setting of the lower_case_table_names system variable.

The optional EXTENDED modifier causes SHOW TABLES to list hidden tables created by failed ALTER TABLE statements. These temporary tables have names beginning with #sql and can be dropped using DROP TABLE.

This statement also lists any views in the database. The optional FULL modifier causes SHOW TABLES to display a second output column with values of BASE TABLE for a table, VIEW for a view, or SYSTEM VIEW for an INFORMATION_SCHEMA table.

If you have no privileges for a base table or view, it does not show up in the output from SHOW TABLES or mysqlshow db_name.

Table information is also available from the INFORMATION_SCHEMA TABLES table. See Section 25.30, "The INFORMATION_SCHEMA TABLES Table".

13.7.7.38 SHOW TRIGGERS Syntax

```
SHOW TRIGGERS
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

SHOW TRIGGERS lists the triggers currently defined for tables in a database (the default database unless a FROM clause is given). This statement returns results only for databases and tables for which you have the TRIGGER privilege. The LIKE clause, if present, indicates which table names (not trigger names) to match and causes the statement to display triggers for those tables. The WHERE clause can be given to select rows using more general conditions, as discussed in Section 25.42, "Extensions to SHOW Statements".

For the ins_sum trigger defined in Section 24.3, "Using Triggers", the output of SHOW TRIGGERS is as shown here:

```
mysql> SHOW TRIGGERS LIKE 'acc%'\G
*************************** 1. row ***************************
             Trigger: ins_sum
               Event: INSERT
               Table: account
           Statement: SET @sum = @sum + NEW.amount
              Timing: BEFORE
             Created: 2018-08-08 10:10:12.61
            sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
                      NO_ZERO_IN_DATE,NO_ZERO_DATE,
                      ERROR_FOR_DIVISION_BY_ZERO,
                      NO_ENGINE_SUBSTITUTION
             Definer: me@localhost
character_set_client: utf8mb4
collation_connection: utf8mb4_0900_ai_ci
  Database Collation: utf8mb4_0900_ai_ci
```

`SHOW TRIGGERS` output has these columns:

- `Trigger`

  The name of the trigger.

- `Event`

  The trigger event. This is the type of operation on the associated table for which the trigger activates. The value is `INSERT` (a row was inserted), `DELETE` (a row was deleted), or `UPDATE` (a row was modified).

- `Table`

  The table for which the trigger is defined.

- `Statement`

  The trigger body; that is, the statement executed when the trigger activates.

- `Timing`

  Whether the trigger activates before or after the triggering event. The value is `BEFORE` or `AFTER`.

- `Created`

  The date and time when the trigger was created. This is a `TIMESTAMP(2)` value (with a fractional part in hundredths of seconds) for triggers.

- `sql_mode`

  The SQL mode in effect when the trigger was created, and under which the trigger executes. For the permitted values, see Section 5.1.11, “Server SQL Modes”.

- `Definer`

  The account of the user who created the trigger, in `'user_name'@'host_name'` format.

- `character_set_client`

  The session value of the `character_set_client` system variable when the trigger was created.

- `collation_connection`

The session value of the `collation_connection` system variable when the trigger was created.

- `Database Collation`

  The collation of the database with which the trigger is associated.

Trigger information is also available from the `INFORMATION_SCHEMA TRIGGERS` table. See Section 25.34, "The INFORMATION_SCHEMA TRIGGERS Table".

13.7.7.39 SHOW VARIABLES Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES
    [LIKE 'pattern' | WHERE expr]
```

`SHOW VARIABLES` shows the values of MySQL system variables (see Section 5.1.8, "Server System Variables"). This statement does not require any privilege. It requires only the ability to connect to the server.

System variable information is also available from these sources:

- Performance Schema tables. See Section 26.12.13, "Performance Schema System Variable Tables".
- The `mysqladmin variables` command. See Section 4.5.2, "`mysqladmin` — Client for Administering a MySQL Server".

For `SHOW VARIABLES`, a `LIKE` clause, if present, indicates which variable names to match. A `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 25.42, "Extensions to SHOW Statements".

`SHOW VARIABLES` accepts an optional `GLOBAL` or `SESSION` variable scope modifier:

- With a `GLOBAL` modifier, the statement displays global system variable values. These are the values used to initialize the corresponding session variables for new connections to MySQL. If a variable has no global value, no value is displayed.
- With a `SESSION` modifier, the statement displays the system variable values that are in effect for the current connection. If a variable has no session value, the global value is displayed. `LOCAL` is a synonym for `SESSION`.
- If no modifier is present, the default is `SESSION`.

The scope for each system variable is listed at Section 5.1.8, "Server System Variables".

`SHOW VARIABLES` is subject to a version-dependent display-width limit. For variables with very long values that are not completely displayed, use `SELECT` as a workaround. For example:

```
SELECT @@GLOBAL.innodb_data_file_path;
```

Most system variables can be set at server startup (read-only variables such as `version_comment` are exceptions). Many can be changed at runtime with the `SET` statement. See Section 5.1.9, "Using System Variables", and Section 13.7.6.1, "SET Syntax for Variable Assignment".

Partial output is shown here. The list of names and values may differ for your server. Section 5.1.8, "Server System Variables", describes the meaning of each variable, and Section 5.1.1, "Configuring the Server", provides information about tuning them.

```
mysql> SHOW VARIABLES;
```

```
+-----------------------------------------+------------------------------+
| Variable_name                           | Value                        |
+-----------------------------------------+------------------------------+
| activate_all_roles_on_login             | OFF                          |
| auto_generate_certs                     | ON                           |
| auto_increment_increment                | 1                            |
| auto_increment_offset                   | 1                            |
| autocommit                              | ON                           |
| automatic_sp_privileges                 | ON                           |
| avoid_temporal_upgrade                  | OFF                          |
| back_log                                | 151                          |
| basedir                                 | /usr/                        |
| big_tables                              | OFF                          |
| bind_address                            | *                            |
| binlog_cache_size                       | 32768                        |
| binlog_checksum                         | CRC32                        |
| binlog_direct_non_transactional_updates | OFF                          |
| binlog_error_action                     | ABORT_SERVER                 |
| binlog_expire_logs_seconds              | 2592000                      |
| binlog_format                           | ROW                          |
| binlog_group_commit_sync_delay          | 0                            |
| binlog_group_commit_sync_no_delay_count | 0                            |
| binlog_gtid_simple_recovery             | ON                           |
| binlog_max_flush_queue_time             | 0                            |
| binlog_order_commits                    | ON                           |
| binlog_row_image                        | FULL                         |
| binlog_row_metadata                     | MINIMAL                      |
| binlog_row_value_options                |                              |
| binlog_rows_query_log_events            | OFF                          |
| binlog_stmt_cache_size                  | 32768                        |
| binlog_transaction_dependency_history_size | 25000                        |
| binlog_transaction_dependency_tracking  | COMMIT_ORDER                 |
| block_encryption_mode                   | aes-128-ecb                  |
| bulk_insert_buffer_size                 | 8388608                      |

...

| max_allowed_packet                      | 67108864                     |
| max_binlog_cache_size                   | 18446744073709547520         |
| max_binlog_size                         | 1073741824                   |
| max_binlog_stmt_cache_size              | 18446744073709547520         |
| max_connect_errors                      | 100                          |
| max_connections                         | 151                          |
| max_delayed_threads                     | 20                           |
| max_digest_length                       | 1024                         |
| max_error_count                         | 1024                         |
| max_execution_time                      | 0                            |
| max_heap_table_size                     | 16777216                     |
| max_insert_delayed_threads              | 20                           |
| max_join_size                           | 18446744073709551615         |

...

| thread_handling                         | one-thread-per-connection    |
| thread_stack                            | 286720                       |
| time_zone                               | SYSTEM                       |
| timestamp                               | 1530906638.765316            |
| tls_version                             | TLSv1,TLSv1.1,TLSv1.2        |
| tmp_table_size                          | 16777216                     |
| tmpdir                                  | /tmp                         |
| transaction_alloc_block_size            | 8192                         |
| transaction_allow_batching              | OFF                          |
| transaction_isolation                   | REPEATABLE-READ              |
| transaction_prealloc_size               | 4096                         |
| transaction_read_only                   | OFF                          |
| transaction_write_set_extraction        | XXHASH64                     |
| unique_checks                           | ON                           |
```

```
| updatable_views_with_limit                | YES                           |
| version                                   | 8.0.12                        |
| version_comment                           | MySQL Community Server - GPL |
| version_compile_machine                   | x86_64                        |
| version_compile_os                        | Linux                         |
| version_compile_zlib                      | 1.2.11                        |
| wait_timeout                              | 28800                         |
| warning_count                             | 0                             |
| windowing_use_high_precision              | ON                            |
+-------------------------------------------+-------------------------------+
```

With a `LIKE` clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a `LIKE` clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the `%` wildcard character in a `LIKE` clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because `_` is a wildcard that matches any single character, you should escape it as `\_` to match it literally. In practice, this is rarely necessary.

13.7.7.40 SHOW WARNINGS Syntax

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

`SHOW WARNINGS` is a diagnostic statement that displays information about the conditions (errors, warnings, and notes) resulting from executing a statement in the current session. Warnings are generated for DML statements such as `INSERT`, `UPDATE`, and `LOAD DATA` as well as DDL statements such as `CREATE TABLE` and `ALTER TABLE`.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.2.10, “SELECT Syntax”.

`SHOW WARNINGS` is also used following `EXPLAIN`, to display the extended information generated by `EXPLAIN`. See Section 8.8.3, “Extended EXPLAIN Output Format”.

`SHOW WARNINGS` displays information about the conditions resulting from execution of the most recent nondiagnostic statement in the current session. If the most recent statement resulted in an error during parsing, `SHOW WARNINGS` shows the resulting conditions, regardless of statement type (diagnostic or nondiagnostic).

The `SHOW COUNT(*) WARNINGS` diagnostic statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the `warning_count` system variable:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

A difference in these statements is that the first is a diagnostic statement that does not clear the message list. The second, because it is a `SELECT` statement is considered nondiagnostic and does clear the message list.

A related diagnostic statement, `SHOW ERRORS`, shows only error conditions (it excludes warnings and notes), and `SHOW COUNT(*) ERRORS` statement displays the total number of errors. See Section 13.7.7.17, “SHOW ERRORS Syntax”. `GET DIAGNOSTICS` can be used to examine information for individual conditions. See Section 13.6.7.3, “GET DIAGNOSTICS Syntax”.

Here is a simple example that shows data-conversion warnings for `INSERT`. The example assumes that strict SQL mode is disabled. With strict mode enabled, the warnings would become errors and terminate the `INSERT`.

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4));
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'), (NULL,'test'), (300,'xyz');
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 1
*************************** 2. row ***************************
  Level: Warning
   Code: 1048
Message: Column 'a' cannot be null
*************************** 3. row ***************************
  Level: Warning
   Code: 1264
Message: Out of range value for column 'a' at row 3
3 rows in set (0.00 sec)
```

The `max_error_count` system variable controls the maximum number of error, warning, and note messages for which the server stores information, and thus the number of messages that `SHOW WARNINGS` displays. To change the number of messages the server can store, change the value of `max_error_count`.

`max_error_count` controls only how many messages are stored, not how many are counted. The value of `warning_count` is not limited by `max_error_count`, even if the number of messages generated exceeds `max_error_count`. The following example demonstrates this. The `ALTER TABLE` statement produces three warning messages (strict SQL mode is disabled for the example to prevent an error from occuring after a single conversion issue). Only one message is stored and displayed because `max_error_count` has been set to 1, but all three are counted (as shown by the value of `warning_count`):

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 1024  |
+-----------------+-------+
1 row in set (0.00 sec)

mysql> SET max_error_count=1, sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SHOW WARNINGS;
```

```
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT @@warning_count;
+-----------------+
| @@warning_count |
+-----------------+
|               3 |
+-----------------+
1 row in set (0.01 sec)
```

To disable message storage, set `max_error_count` to 0. In this case, `warning_count` still indicates how many warnings occurred, but messages are not stored and cannot be displayed.

The `sql_notes` system variable controls whether note messages increment `warning_count` and whether the server stores them. By default, `sql_notes` is 1, but if set to 0, notes do not increment `warning_count` and the server does not store them:

```
mysql> SET sql_notes = 1;
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS;
+-------+------+-----------------------------------+
| Level | Code | Message                           |
+-------+------+-----------------------------------+
| Note  | 1051 | Unknown table 'test.no_such_table' |
+-------+------+-----------------------------------+
1 row in set (0.00 sec)

mysql> SET sql_notes = 0;
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected (0.00 sec)
mysql> SHOW WARNINGS;
Empty set (0.00 sec)
```

The MySQL server sends to each client a count indicating the total number of errors, warnings, and notes resulting from the most recent statement executed by that client. From the C API, this value can be obtained by calling `mysql_warning_count()`. See Section 28.7.7.82, “mysql_warning_count()”.

In the `mysql` client, you can enable and disable automatic warnings display using the `warnings` and `nowarning` commands, respectively, or their shortcuts, `\W` and `\w` (see Section 4.5.1.2, “mysql Client Commands”). For example:

```
mysql> \W
Show warnings enabled.
mysql> SELECT 1/0;
+------+
| 1/0  |
+------+
| NULL |
+------+
1 row in set, 1 warning (0.03 sec)

Warning (Code 1365): Division by 0
mysql> \w
Show warnings disabled.
```

13.7.8 Other Administrative Statements

13.7.8.1 BINLOG Syntax

```
BINLOG 'str'
```

`BINLOG` is an internal-use statement. It is generated by the `mysqlbinlog` program as the printable representation of certain events in binary log files. (See Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.) The `'str'` value is a base 64-encoded string the that server decodes to determine the data change indicated by the corresponding event.

To execute `BINLOG` statements when applying `mysqlbinlog` output, a user account requires the `BINLOG_ADMIN` or `SUPER` privilege, or the `REPLICATION_APPLIER` privilege plus the appropriate privileges to execute each log event.

This statement can execute only format description events and row events.

13.7.8.2 CACHE INDEX Syntax

```
CACHE INDEX {
      tbl_index_list [, tbl_index_list] ...
    | tbl_name PARTITION (partition_list)
  }
  IN key_cache_name

tbl_index_list:
  tbl_name [{INDEX|KEY} (index_name[, index_name] ...)]

partition_list: {
    partition_name[, partition_name] ...
  | ALL
}
```

The `CACHE INDEX` statement assigns table indexes to a specific key cache. It applies only to `MyISAM` tables, including partitioned `MyISAM` tables. After the indexes have been assigned, they can be preloaded into the cache if desired with `LOAD INDEX INTO CACHE`.

The following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

The syntax of `CACHE INDEX` enables you to specify that only particular indexes from a table should be assigned to the cache. However, the implementation assigns all the table's indexes to the cache, so there is no reason to specify anything other than the table name.

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a parameter setting statement or in the server parameter settings. For example:

```
SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Key cache parameters are accessed as members of a structured system variable. See Section 5.1.9.5, “Structured System Variables”.

A key cache must exist before you assign indexes to it, or an error occurs:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

Index assignment affects the server globally: If one client assigns an index to a given cache, this cache is used for all queries involving the index, no matter which client issues the queries.

`CACHE INDEX` is supported for partitioned `MyISAM` tables. You can assign one or more indexes for one, several, or all partitions to a given key cache. For example, you can do the following:

```
CREATE TABLE pt (c1 INT, c2 VARCHAR(50), INDEX i(c1))
    ENGINE=MyISAM
    PARTITION BY HASH(c1)
    PARTITIONS 4;

SET GLOBAL kc_fast.key_buffer_size = 128 * 1024;
SET GLOBAL kc_slow.key_buffer_size = 128 * 1024;

CACHE INDEX pt PARTITION (p0) IN kc_fast;
CACHE INDEX pt PARTITION (p1, p3) IN kc_slow;
```

The previous set of statements performs the following actions:

- Creates a partitioned table with 4 partitions; these partitions are automatically named `p0`, ..., `p3`; this table has an index named `i` on column `c1`.
- Creates 2 key caches named `kc_fast` and `kc_slow`
- Assigns the index for partition `p0` to the `kc_fast` key cache and the index for partitions `p1` and `p3` to the `kc_slow` key cache; the index for the remaining partition (`p2`) uses the server's default key cache.

If you wish instead to assign the indexes for all partitions in table `pt` to a single key cache named `kc_all`, you can use either of the following two statements:

```
CACHE INDEX pt PARTITION (ALL) IN kc_all;

CACHE INDEX pt IN kc_all;
```

The two statements just shown are equivalent, and issuing either one has exactly the same effect. In other words, if you wish to assign indexes for all partitions of a partitioned table to the same key cache, the `PARTITION (ALL)` clause is optional.

When assigning indexes for multiple partitions to a key cache, the partitions need not be contiguous, and you need not list their names in any particular order. Indexes for any partitions not explicitly assigned to a key cache automatically use the server default key cache.

Index preloading is also supported for partitioned `MyISAM` tables. For more information, see Section 13.7.8.5, “LOAD INDEX INTO CACHE Syntax”.

13.7.8.3 FLUSH Syntax

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL] {
    flush_option [, flush_option] ...
  | tables_option
}

flush_option: {
    BINARY LOGS
  | ENGINE LOGS
  | ERROR LOGS
  | GENERAL LOGS
  | HOSTS
  | LOGS
  | PRIVILEGES
  | OPTIMIZER_COSTS
  | RELAY LOGS [FOR CHANNEL channel]
  | SLOW LOGS
  | STATUS
  | USER_RESOURCES
}

tables_option: {
    TABLES
  | TABLES tbl_name [, tbl_name] ...
  | TABLES WITH READ LOCK
  | TABLES tbl_name [, tbl_name] ... WITH READ LOCK
  | TABLES tbl_name [, tbl_name] ... FOR EXPORT
}
```

The `FLUSH` statement has several variant forms that clear or reload various internal caches, flush tables, or acquire locks. To execute `FLUSH`, you must have the `RELOAD` privilege. Specific flush options might require additional privileges, as described later.

Note

It is not possible to issue `FLUSH` statements within stored functions or triggers. However, you may use `FLUSH` in stored procedures, so long as these are not called from stored functions or triggers. See Section 24.8, “Restrictions on Stored Programs”.

By default, the server writes `FLUSH` statements to the binary log so that they replicate to replication slaves. To suppress logging, specify the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

Note

`FLUSH LOGS`, `FLUSH BINARY LOGS`, `FLUSH TABLES WITH READ LOCK` (with or without a table list), and `FLUSH TABLES tbl_name ... FOR EXPORT` are not written to the binary log in any case because they would cause problems if replicated to a slave.

The `FLUSH` statement causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

The `mysqladmin` utility provides a command-line interface to some flush operations, using commands such as `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, and `flush-tables`. See Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”.

Sending a `SIGHUP` signal to the server causes several flush operations to occur that are similar to various forms of the `FLUSH` statement. See Section 5.1.16, “Server Response to Signals”.

The RESET statement is similar to FLUSH. See Section 13.7.8.6, “RESET Syntax”, for information about using the RESET statement with replication.

The following list describes the permitted FLUSH statement *flush_option* values. For descriptions of FLUSH TABLES variants, see FLUSH TABLES Syntax.

- FLUSH BINARY LOGS

  Closes and reopens any binary log file to which the server is writing. If binary logging is enabled, the sequence number of the binary log file is incremented by one relative to the previous file.

- FLUSH ENGINE LOGS

  Closes and reopens any flushable logs for installed storage engines. This causes InnoDB to flush its logs to disk.

- FLUSH ERROR LOGS

  Closes and reopens any error log file to which the server is writing.

- FLUSH GENERAL LOGS

  Closes and reopens any general query log file to which the server is writing.

- FLUSH HOSTS

  Empties the host cache and the Performance Schema host_cache table that exposes the cache contents, and unblocks any blocked hosts. See Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

  Flush the host cache if some of your hosts change IP address or if the error message Host '*host_name*' is blocked occurs for connections from legitimate hosts. (See Section B.4.2.5, “Host 'host_name' is blocked”.) When more than max_connect_errors errors occur successively for a given host while connecting to the MySQL server, MySQL assumes that something is wrong and blocks the host from further connection requests. Flushing the host cache enables further connection attempts from the host. The default value of max_connect_errors is 100. To avoid this error message, start the server with max_connect_errors set to a large value.

- FLUSH LOGS

  Closes and reopens any log file to which the server is writing. If binary logging is enabled, the sequence number of the binary log file is incremented by one relative to the previous file. If relay logging is enabled, the sequence number of the relay log file is incremented by one relative to the previous file.

  FLUSH LOGS has no effect on tables used for the general query log or for the slow query log (see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”).

- FLUSH OPTIMIZER_COSTS

  Rereads the cost model tables so that the optimizer starts using the current cost estimates stored in them. The server writes a warning to the error log for any unrecognized entries. (For information about these tables, see Section 8.9.5, “The Optimizer Cost Model”.) This operation affects only sessions that begin subsequent to the flush. Existing sessions continue to use the cost estimates that were current when they began.

- FLUSH PRIVILEGES

Reloads the privileges from the grant tables in the `mysql` system database, and clears the in-memory cache used by the `caching_sha2_password` authentication plugin.

As part of this operation, the server reads the `global_grants` table containing dynamic privilege assignments and registers any unregistered privileges found there.

The server caches information in memory as a result of `GRANT`, `CREATE USER`, `CREATE SERVER`, and `INSTALL PLUGIN` statements. This memory is not released by the corresponding `REVOKE`, `DROP USER`, `DROP SERVER`, and `UNINSTALL PLUGIN` statements, so for a server that executes many instances of the statements that cause caching, there will be an increase in memory use. This cached memory can be freed with `FLUSH PRIVILEGES`.

- `FLUSH RELAY LOGS [FOR CHANNEL channel]`

  Closes and reopens any relay log file to which the server is writing. If relay logging is enabled, the sequence number of the relay log file is incremented by one relative to the previous file.

  The `FOR CHANNEL channel` clause enables you to name which replication channel the statement applies to. Execute `FLUSH RELAY LOGS FOR CHANNEL channel` to flush the relay log for a specific replication channel. If no channel is named and no extra replication channels exist, the statement applies to the default channel. If no channel is named and multiple replication channels exist, the statement applies to all replication channels. For more information, see Section 17.2.3, “Replication Channels”.

- `FLUSH SLOW LOGS`

  Closes and reopens any slow query log file to which the server is writing.

- `FLUSH STATUS`

  This option adds the session status from all active sessions to the global status variables, resets the status of all active sessions, and resets account, host, and user status values aggregated from disconnected sessions. See Section 26.12.14, “Performance Schema Status Variable Tables”. This information may be of use when debugging a query. See Section 1.7, “How to Report Bugs or Problems”.

- `FLUSH USER_RESOURCES`

  Resets all per-hour user resources to zero. This enables clients that have reached their hourly connection, query, or update limits to resume activity immediately. `FLUSH USER_RESOURCES` does not apply to the limit on maximum simultaneous connections that is controlled by the `max_user_connections` system variable. See Section 6.2.20, “Setting Account Resource Limits”.

FLUSH TABLES Syntax

`FLUSH TABLES` flushes tables, and, depending on the variant used, acquires locks. Any `TABLES` variant used in a `FLUSH` statement must be the only option used. `FLUSH TABLE` is a synonym for `FLUSH TABLES`.

Note

The descriptions here that indicate tables are flushed by closing them apply differently for `InnoDB`, which flushes table contents to disk but leaves them open. This still permits table files to be copied while the tables are open, as long as other activity does not modify them.

- `FLUSH TABLES`

Closes all open tables, forces all tables in use to be closed, and flushes the prepared statement cache. For information about prepared statement caching, see Section 8.10.3, “Caching of Prepared Statements and Stored Programs”.

`FLUSH TABLES` is not permitted when there is an active `LOCK TABLES ... READ`. To flush and lock tables, use `FLUSH TABLES tbl_name ... WITH READ LOCK` instead.

- `FLUSH TABLES tbl_name [, tbl_name] ...`

  With a list of one or more comma-separated table names, this statement is like `FLUSH TABLES` with no names except that the server flushes only the named tables. If a named table does not exist, no error occurs.

- `FLUSH TABLES WITH READ LOCK`

  Closes all open tables and locks all tables for all databases with a global read lock. This is a very convenient way to get backups if you have a file system such as Veritas or ZFS that can take snapshots in time. Use `UNLOCK TABLES` to release the lock.

  `FLUSH TABLES WITH READ LOCK` acquires a global read lock rather than table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits:

  - `UNLOCK TABLES` implicitly commits any active transaction only if any tables currently have been locked with `LOCK TABLES`. The commit does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table locks.
  - Beginning a transaction causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

  `FLUSH TABLES WITH READ LOCK` does not prevent the server from inserting rows into the log tables (see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”).

- `FLUSH TABLES tbl_name [, tbl_name] ... WITH READ LOCK`

  This statement flushes and acquires read locks for the named tables. The statement first acquires exclusive metadata locks for the tables, so it waits for transactions that have those tables open to complete. Then the statement flushes the tables from the table cache, reopens the tables, acquires table locks (like `LOCK TABLES ... READ`), and downgrades the metadata locks from exclusive to shared. After the statement acquires locks and downgrades the metadata locks, other sessions can read but not modify the tables.

  Because this statement acquires table locks, you must have the `LOCK TABLES` privilege for each table, in addition to the `RELOAD` privilege that is required to use any `FLUSH` statement.

  This statement applies only to existing base (non-`TEMPORARY`) tables. If a name refers to a base table, that table is used. If it refers to a `TEMPORARY` table, it is ignored. If a name applies to a view, an `ER_WRONG_OBJECT` error occurs. Otherwise, an `ER_NO_SUCH_TABLE` error occurs.

  Use `UNLOCK TABLES` to release the locks, `LOCK TABLES` to release the locks and acquire other locks, or `START TRANSACTION` to release the locks and begin a new transaction.

  This `FLUSH TABLES` variant enables tables to be flushed and locked in a single operation. It provides a workaround for the restriction that `FLUSH TABLES` is not permitted when there is an active `LOCK TABLES ... READ`.

This statement does not perform an implicit `UNLOCK TABLES`, so an error results if you use the statement while there is any active `LOCK TABLES` or use it a second time without first releasing the locks acquired.

If a flushed table was opened with `HANDLER`, the handler is implicitly flushed and loses its position.

- `FLUSH TABLES tbl_name [, tbl_name] ... FOR EXPORT`

  This `FLUSH TABLES` variant applies to `InnoDB` tables. It ensures that changes to the named tables have been flushed to disk so that binary table copies can be made while the server is running.

  The statement works like this:

  1. It acquires shared metadata locks for the named tables. The statement blocks as long as other sessions have active transactions that have modified those tables or hold table locks for them. When the locks have been acquired, the statement blocks transactions that attempt to update the tables, while permitting read-only operations to continue.

  2. It checks whether all storage engines for the tables support `FOR EXPORT`. If any do not, an `ER_ILLEGAL_HA` error occurs and the statement fails.

  3. The statement notifies the storage engine for each table to make the table ready for export. The storage engine must ensure that any pending changes are written to disk.

  4. The statement puts the session in lock-tables mode so that the metadata locks acquired earlier are not released when the `FOR EXPORT` statement completes.

  The `FLUSH TABLES ... FOR EXPORT` statement requires that you have the `SELECT` privilege for each table. Because this statement acquires table locks, you must also have the `LOCK TABLES` privilege for each table, in addition to the `RELOAD` privilege that is required to use any `FLUSH` statement.

  This statement applies only to existing base (non-`TEMPORARY`) tables. If a name refers to a base table, that table is used. If it refers to a `TEMPORARY` table, it is ignored. If a name applies to a view, an `ER_WRONG_OBJECT` error occurs. Otherwise, an `ER_NO_SUCH_TABLE` error occurs.

  `InnoDB` supports `FOR EXPORT` for tables that have their own `.ibd` file (that is, tables created with the `innodb_file_per_table` setting enabled). `InnoDB` ensures when notified by the `FOR EXPORT` statement that any changes have been flushed to disk. This permits a binary copy of table contents to be made while the `FOR EXPORT` statement is in effect because the `.ibd` file is transaction consistent and can be copied while the server is running. `FOR EXPORT` does not apply to `InnoDB` system tablespace files, or to `InnoDB` tables that have `FULLTEXT` indexes.

  `FLUSH TABLES ...FOR EXPORT` is supported for partitioned `InnoDB` tables.

  When notified by `FOR EXPORT`, `InnoDB` writes to disk certain kinds of data that is normally held in memory or in separate disk buffers outside the tablespace files. For each table, `InnoDB` also produces a file named *`table_name`*`.cfg` in the same database directory as the table. The `.cfg` file contains metadata needed to reimport the tablespace files later, into the same or different server.

  When the `FOR EXPORT` statement completes, `InnoDB` will have flushed all dirty pages to the table data files. Any change buffer entries are merged prior to flushing. At this point, the tables are locked and quiescent: The tables are in a transactionally consistent state on disk and you can copy the `.ibd` tablespace files along with the corresponding `.cfg` files to get a consistent snapshot of those tables.

For the procedure to reimport the copied table data into a MySQL instance, see Section 15.6.3.7, “Copying Tablespaces to Another Instance”.

After you are done with the tables, use `UNLOCK TABLES` to release the locks, `LOCK TABLES` to release the locks and acquire other locks, or `START TRANSACTION` to release the locks and begin a new transaction.

While any of these statements is in effect within the session, attempts to use `FLUSH TABLES ... FOR EXPORT` produce an error:

```
FLUSH TABLES ... WITH READ LOCK
FLUSH TABLES ... FOR EXPORT
LOCK TABLES ... READ
LOCK TABLES ... WRITE
```

While `FLUSH TABLES ... FOR EXPORT` is in effect within the session, attempts to use any of these statements produce an error:

```
FLUSH TABLES WITH READ LOCK
FLUSH TABLES ... WITH READ LOCK
FLUSH TABLES ... FOR EXPORT
```

13.7.8.4 KILL Syntax

```
KILL [CONNECTION | QUERY] processlist_id
```

Each connection to `mysqld` runs in a separate thread. You can kill a thread with the `KILL` *`processlist_id`* statement.

Thread processlist identifiers can be determined from the `ID` column of the `INFORMATION_SCHEMA PROCESSLIST` table, the `Id` column of `SHOW PROCESSLIST` output, and the `PROCESSLIST_ID` column of the Performance Schema `threads` table. The value for the current thread is returned by the `CONNECTION_ID()` function.

`KILL` permits an optional `CONNECTION` or `QUERY` modifier:

- `KILL CONNECTION` is the same as `KILL` with no modifier: It terminates the connection associated with the given *`processlist_id`*, after terminating any statement the connection is executing.
- `KILL QUERY` terminates the statement the connection is currently executing, but leaves the connection itself intact.

The ability to see which threads are available to be killed depends on the `PROCESS` privilege:

- Without `PROCESS`, you can see only your own threads.
- With `PROCESS`, you can see all threads.

The ability to kill threads and statements depends on the `CONNECTION_ADMIN` and `SUPER` privileges:

- Without `CONNECTION_ADMIN` or `SUPER`, you can kill only your own threads and statements.
- With `CONNECTION_ADMIN` or `SUPER`, you can kill all threads and statements, except that to affect a thread or statement that is executing with the `SYSTEM_USER` privilege, your own session must additionally have the `SYSTEM_USER` privilege.

You can also use the `mysqladmin processlist` and `mysqladmin kill` commands to examine and kill threads.

When you use `KILL`, a thread-specific kill flag is set for the thread. In most cases, it might take some time for the thread to die because the kill flag is checked only at specific intervals:

- During `SELECT` operations, for `ORDER BY` and `GROUP BY` loops, the flag is checked after reading a block of rows. If the kill flag is set, the statement is aborted.
- `ALTER TABLE` operations that make a table copy check the kill flag periodically for each few copied rows read from the original table. If the kill flag was set, the statement is aborted and the temporary table is deleted.

  The `KILL` statement returns without waiting for confirmation, but the kill flag check aborts the operation within a reasonably small amount of time. Aborting the operation to perform any necessary cleanup also takes some time.
- During `UPDATE` or `DELETE` operations, the kill flag is checked after each block read and after each updated or deleted row. If the kill flag is set, the statement is aborted. If you are not using transactions, the changes are not rolled back.
- `GET_LOCK()` aborts and returns `NULL`.
- If the thread is in the table lock handler (state: `Locked`), the table lock is quickly aborted.
- If the thread is waiting for free disk space in a write call, the write is aborted with a “disk full” error message.

Warning

Killing a `REPAIR TABLE` or `OPTIMIZE TABLE` operation on a `MyISAM` table results in a table that is corrupted and unusable. Any reads or writes to such a table fail until you optimize or repair it again (without interruption).

13.7.8.5 LOAD INDEX INTO CACHE Syntax

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...

tbl_index_list:
  tbl_name
    [PARTITION (partition_list)]
    [{INDEX|KEY} (index_name[, index_name] ...)]
    [IGNORE LEAVES]

partition_list: {
    partition_name[, partition_name] ...
  | ALL
}
```

The `LOAD INDEX INTO CACHE` statement preloads a table index into the key cache to which it has been assigned by an explicit `CACHE INDEX` statement, or into the default key cache otherwise.

`LOAD INDEX INTO CACHE` applies only to `MyISAM` tables, including partitioned `MyISAM` tables. In addition, indexes on partitioned tables can be preloaded for one, several, or all partitions.

The `IGNORE LEAVES` modifier causes only blocks for the nonleaf nodes of the index to be preloaded.

`IGNORE LEAVES` is also supported for partitioned `MyISAM` tables.

The following statement preloads nodes (index blocks) of indexes for the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

This statement preloads all index blocks from `t1`. It preloads only blocks for the nonleaf nodes from `t2`.

The syntax of `LOAD INDEX INTO CACHE` enables you to specify that only particular indexes from a table should be preloaded. However, the implementation preloads all the table's indexes into the cache, so there is no reason to specify anything other than the table name.

It is possible to preload indexes on specific partitions of partitioned `MyISAM` tables. For example, of the following 2 statements, the first preloads indexes for partition `p0` of a partitioned table `pt`, while the second preloads the indexes for partitions `p1` and `p3` of the same table:

```
LOAD INDEX INTO CACHE pt PARTITION (p0);
LOAD INDEX INTO CACHE pt PARTITION (p1, p3);
```

To preload the indexes for all partitions in table `pt`, you can use either of the following two statements:

```
LOAD INDEX INTO CACHE pt PARTITION (ALL);

LOAD INDEX INTO CACHE pt;
```

The two statements just shown are equivalent, and issuing either one has exactly the same effect. In other words, if you wish to preload indexes for all partitions of a partitioned table, the `PARTITION (ALL)` clause is optional.

When preloading indexes for multiple partitions, the partitions need not be contiguous, and you need not list their names in any particular order.

`LOAD INDEX INTO CACHE ... IGNORE LEAVES` fails unless all indexes in a table have the same block size. To determine index block sizes for a table, use `myisamchk -dv` and check the `Blocksize` column.

13.7.8.6 RESET Syntax

```
RESET reset_option [, reset_option] ...

reset_option: {
    MASTER
  | SLAVE
}
```

The `RESET` statement is used to clear the state of various server operations. You must have the `RELOAD` privilege to execute `RESET`.

For information about the `RESET PERSIST` statement that removes persisted global system variables, see Section 13.7.8.7, “RESET PERSIST Syntax”.

`RESET` acts as a stronger version of the `FLUSH` statement. See Section 13.7.8.3, “FLUSH Syntax”.

The RESET statement causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

The following list describes the permitted RESET statement *reset_option* values:

- RESET MASTER

  Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

- RESET SLAVE

  Makes the slave forget its replication position in the master binary logs. Also resets the relay log by deleting any existing relay log files and beginning a new one.

13.7.8.7 RESET PERSIST Syntax

```
RESET PERSIST [[IF EXISTS] system_var_name]
```

RESET PERSIST removes persisted global system variable settings from the mysqld-auto.cnf option file in the data directory. Removing a persisted system variable causes the variable no longer to be initialized from mysqld-auto.cnf at server startup. For more information about persisting system variables and the mysqld-auto.cnf file, see Section 5.1.9.3, “Persisted System Variables”.

The privileges required for RESET PERSIST depend on the type of system variable to be removed:

- For dynamic system variables, this statement requires the SYSTEM_VARIABLES_ADMIN or SUPER privilege.
- For read-only system variables, this statement requires the SYSTEM_VARIABLES_ADMIN and PERSIST_RO_VARIABLES_ADMIN privileges.

See Section 5.1.9.1, “System Variable Privileges”.

Depending on whether the variable name and IF EXISTS clauses are present, the RESET PERSIST statement has these forms:

- To remove all persisted variables from mysqld-auto.cnf, use RESET PERSIST without naming any system variable:

  ```
  RESET PERSIST;
  ```

  You must have privileges for removing both dynamic and read-only system variables if mysqld-auto.cnf contains both kinds of variables.

- To remove a specific persisted variable from mysqld-auto.cnf, name it in the statement:

  ```
  RESET PERSIST system_var_name;
  ```

  This includes plugin system variables, even if the plugin is not currently installed. If the variable is not present in the file, an error occurs.

- To remove a specific persisted variable from mysqld-auto.cnf, but produce a warning rather than an error if the variable is not present in the file, add an IF EXISTS clause to the previous syntax:

  ```
  ```

```
RESET PERSIST IF EXISTS system_var_name;
```

`RESET PERSIST` is not affected by the value of the `persisted_globals_load` system variable.

`RESET PERSIST` affects the contents of the Performance Schema `persisted_variables` table because the table contents correspond to the contents of the `mysqld-auto.cnf` file. On the other hand, because `RESET PERSIST` does not change variable values, it has no effect on the contents of the Performance Schema `variables_info` table until the server is restarted.

For information about `RESET` statement variants that clear the state of other server operations, see Section 13.7.8.6, “RESET Syntax”.

13.7.8.8 RESTART Syntax

```
RESTART
```

This statement stops and restarts the MySQL server. It requires the `SHUTDOWN` privilege.

One use for `RESTART` is when it is not possible or convenient to gain command-line access to the MySQL server on the server host to restart it. For example, `SET PERSIST_ONLY` can be used at runtime to make configuration changes to system variables that can be set only at server startup, but the server must still be restarted for those changes to take effect. The `RESTART` statement provides a way to do so from within client sessions, without requiring command-line access on the server host.

Note

After executing a `RESTART` statement, the client can expect the current connection to be lost. If auto-reconnect is enabled, the connection will be reestablished after the server restarts. Otherwise, the connection must be reestablished manually.

A successful `RESTART` operation requires `mysqld` to be running in an environment that has a monitoring process available to detect a server shutdown performed for restart purposes:

- In the presence of a monitoring process, `RESTART` causes `mysqld` to terminate such that the monitoring process can determine that it should start a new `mysqld` instance.
- If no monitoring process is present, `RESTART` fails with an error.

These platforms provide the necessary monitoring support for the `RESTART` statement:

- Windows, when `mysqld` is started as a Windows service or standalone. (`mysqld` forks, and one process acts as a monitor to the other, which acts as the server.)
- Unix and Unix-like systems that use systemd or `mysqld_safe` to manage `mysqld`.

To configure a monitoring environment such that `mysqld` enables the `RESTART` statement:

1. Set the `MYSQLD_PARENT_PID` environment variable to the value of the process ID of the process that starts `mysqld`, before starting `mysqld`.
2. When `mysqld` performs a shutdown due to use of the `RESTART` statement, it returns exit code 16.
3. When the monitoring process detects an exit code of 16, it starts `mysqld` again. Otherwise, it exits.

Here is a minimal example as implemented in the `bash` shell:

```
#!/bin/bash
```

```
export MYSQLD_PARENT_PID=$$

export MYSQLD_RESTART_EXIT=16

while true ; do
  bin/mysqld mysqld options here
  if [ $? -ne $MYSQLD_RESTART_EXIT ]; then
    break
  fi
done
```

On Windows, the forking used to implement `RESTART` makes determining the server process to attach to for debugging more difficult. To alleviate this, starting the server with `--gdb` suppresses forking, in addition to its other actions done to set up a debugging environment. In non-debug settings, `--no-monitor` may be used for the sole purpose of suppressing forking the monitor process. For a server started with either `--gdb` or `--no-monitor`, executing `RESTART` causes the server to simply exit without restarting.

13.7.8.9 SHUTDOWN Syntax

```
SHUTDOWN
```

This statement stops the MySQL server. It requires the `SHUTDOWN` privilege.

`SHUTDOWN` provides an SQL-level interface to the same functionality available using the `mysqladmin shutdown` command.

13.8 Utility Statements

13.8.1 DESCRIBE Syntax

The `DESCRIBE` and `EXPLAIN` statements are synonyms, used either to obtain information about table structure or query execution plans. For more information, see Section 13.7.7.5, “SHOW COLUMNS Syntax”, and Section 13.8.2, “EXPLAIN Syntax”.

13.8.2 EXPLAIN Syntax

```
{EXPLAIN | DESCRIBE | DESC}
    tbl_name [col_name | wild]

{EXPLAIN | DESCRIBE | DESC}
    [explain_type]
    {explainable_stmt | FOR CONNECTION connection_id}

{EXPLAIN | DESCRIBE | DESC} ANALYZE select_statement

explain_type: {
    FORMAT = format_name
}

format_name: {
    TRADITIONAL
  | JSON
  | TREE
}

explainable_stmt: {
    SELECT statement
  | DELETE statement
```

```
  | INSERT statement
  | REPLACE statement
  | UPDATE statement
}
```

The DESCRIBE and EXPLAIN statements are synonyms. In practice, the DESCRIBE keyword is more often used to obtain information about table structure, whereas EXPLAIN is used to obtain a query execution plan (that is, an explanation of how MySQL would execute a query).

The following discussion uses the DESCRIBE and EXPLAIN keywords in accordance with those uses, but the MySQL parser treats them as completely synonymous.

- Obtaining Table Structure Information
- Obtaining Execution Plan Information
- Obtaining Information with EXPLAIN ANALYZE

Obtaining Table Structure Information

DESCRIBE provides information about the columns in a table:

```
mysql> DESCRIBE City;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | NO   |     |         |                |
| Country    | char(3)  | NO   | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  | NO   |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
```

DESCRIBE is a shortcut for SHOW COLUMNS. These statements also display information for views. The description for SHOW COLUMNS provides more information about the output columns. See Section 13.7.7.5, “SHOW COLUMNS Syntax”.

By default, DESCRIBE displays information about all columns in the table. *col_name*, if given, is the name of a column in the table. In this case, the statement displays information only for the named column. *wild*, if given, is a pattern string. It can contain the SQL % and _ wildcard characters. In this case, the statement displays output only for the columns with names matching the string. There is no need to enclose the string within quotation marks unless it contains spaces or other special characters.

The DESCRIBE statement is provided for compatibility with Oracle.

The SHOW CREATE TABLE, SHOW TABLE STATUS, and SHOW INDEX statements also provide information about tables. See Section 13.7.7, “SHOW Syntax”.

Obtaining Execution Plan Information

The EXPLAIN statement provides information about how MySQL executes statements:

- EXPLAIN works with SELECT, DELETE, INSERT, REPLACE, and UPDATE statements.
- When EXPLAIN is used with an explainable statement, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. For information about using EXPLAIN to obtain execution plan information, see Section 8.8.2, “EXPLAIN Output Format”.

- When `EXPLAIN` is used with `FOR CONNECTION connection_id` rather than an explainable statement, it displays the execution plan for the statement executing in the named connection. See Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”.
- For explainable statements, `EXPLAIN` produces additional execution plan information that can be displayed using `SHOW WARNINGS`. See Section 8.8.3, “Extended EXPLAIN Output Format”.
- `EXPLAIN` is useful for examining queries involving partitioned tables. See Section 23.3.5, “Obtaining Information About Partitions”.
- The `FORMAT` option can be used to select the output format. `TRADITIONAL` presents the output in tabular format. This is the default if no `FORMAT` option is present. `JSON` format displays the information in JSON format. In MySQL 8.0.16 and later, `TREE` provides tree-like output with more precise descriptions of query handling than the `TRADITIONAL` format; it is the only format which shows hash join usage (see Section 8.2.1.4, “Hash Join Optimization”) and is always used for `EXPLAIN ANALYZE`. Currently, it is still under development and considered experimental only.

`EXPLAIN` requires the `SELECT` privilege for any tables or views accessed, including any underlying tables of views. For views, `EXPLAIN` also requires the `SHOW VIEW` privilege. `EXPLAIN ... FOR CONNECTION` additionally requires the `PROCESS` privilege if the specified connection belongs to a different user.

With the help of `EXPLAIN`, you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use `EXPLAIN` to check whether the optimizer joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in a `SELECT` statement, begin the statement with `SELECT STRAIGHT_JOIN` rather than just `SELECT`. (See Section 13.2.10, “SELECT Syntax”.)

The optimizer trace may sometimes provide information complementary to that of `EXPLAIN`. However, the optimizer trace format and content are subject to change between versions. For details, see MySQL Internals: Tracing the Optimizer.

If you have a problem with indexes not being used when you believe that they should be, run `ANALYZE TABLE` to update table statistics, such as cardinality of keys, that can affect the choices the optimizer makes. See Section 13.7.3.1, “ANALYZE TABLE Syntax”.

Note

MySQL Workbench has a Visual Explain capability that provides a visual representation of `EXPLAIN` output. See Tutorial: Using Explain to Improve Query Performance.

Obtaining Information with EXPLAIN ANALYZE

MySQL 8.0.18 introduces `EXPLAIN ANALYZE`, which runs a query and produces `EXPLAIN` output along with timing and additional, iterator-based information about how the optimizer's expectations matched the actual execution. For each iterator, the following information is provided:

- Estimated execution cost
- Estimated number of returned rows
- Time to return first row
- Time to return all rows (actual cost)
- Number of rows returned by the iterator

- Number of loops

The query execution information is displayed using the `TREE` output format, in which nodes represent iterators. `EXPLAIN ANALYZE` always uses the `TREE` output format and does not accept a format specifier. It also cannot be used with `FOR CONNECTION`.

`EXPLAIN ANALYZE` can be used only with `SELECT` statements.

Example output:

```
mysql> EXPLAIN ANALYZE SELECT * FROM t1 JOIN t2 ON (t1.c1 = t2.c2)\G
*************************** 1. row ***************************
EXPLAIN: -> Inner hash join (t2.c2 = t1.c1)  (cost=4.70 rows=6)
(actual time=0.032..0.035 rows=6 loops=1)
    -> Table scan on t2  (cost=0.06 rows=6)
(actual time=0.003..0.005 rows=6 loops=1)
    -> Hash
        -> Table scan on t1  (cost=0.85 rows=6)
(actual time=0.018..0.022 rows=6 loops=1)

mysql> EXPLAIN ANALYZE SELECT * FROM t3 WHERE i > 8\G
*************************** 1. row ***************************
EXPLAIN: -> Filter: (t3.i > 8)  (cost=1.75 rows=5)
(actual time=0.019..0.021 rows=6 loops=1)
    -> Table scan on t3  (cost=1.75 rows=15)
(actual time=0.017..0.019 rows=15 loops=1)

mysql> EXPLAIN ANALYZE SELECT * FROM t3 WHERE pk > 17\G
*************************** 1. row ***************************
EXPLAIN: -> Filter: (t3.pk > 17)  (cost=1.26 rows=5)
(actual time=0.013..0.016 rows=5 loops=1)
    -> Index range scan on t3 using PRIMARY  (cost=1.26 rows=5)
(actual time=0.012..0.014 rows=5 loops=1)
```

The tables used in the example output were created by the statements shown here:

```
CREATE TABLE t1 (
    c1 INT(11) DEFAULT NULL,
    c2 INT(11) DEFAULT NULL
);

CREATE TABLE t2 (
    c1 INT(11) DEFAULT NULL,
    c2 INT(11) DEFAULT NULL
);

CREATE TABLE t3 (
    pk INT(11) NOT NULL PRIMARY KEY,
    i INT(11) DEFAULT NULL
);
```

13.8.3 HELP Syntax

```
HELP 'search_string'
```

The `HELP` statement returns online information from the MySQL Reference Manual. Its proper operation requires that the help tables in the `mysql` database be initialized with help topic information (see Section 5.1.14, “Server-Side Help Support”).

The `HELP` statement searches the help tables for the given search string and displays the result of the search. The search string is not case-sensitive.

The search string can contain the wildcard characters `%` and `_`. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. For example, `HELP 'rep%'` returns a list of topics that begin with `rep`.

The HELP statement understands several types of search strings:

- At the most general level, use `contents` to retrieve a list of the top-level help categories:

```
HELP 'contents'
```

- For a list of topics in a given help category, such as `Data Types`, use the category name:

```
HELP 'data types'
```

- For help on a specific help topic, such as the `ASCII()` function or the `CREATE TABLE` statement, use the associated keyword or keywords:

```
HELP 'ascii'
HELP 'create table'
```

In other words, the search string matches a category, many topics, or a single topic. You cannot necessarily tell in advance whether a given search string will return a list of items or the help information for a single help topic. However, you can tell what kind of response `HELP` returned by examining the number of rows and columns in the result set.

The following descriptions indicate the forms that the result set can take. Output for the example statements is shown using the familiar “tabular” or “vertical” format that you see when using the `mysql` client, but note that `mysql` itself reformats `HELP` result sets in a different way.

- Empty result set

  No match could be found for the search string.

- Result set containing a single row with three columns

  This means that the search string yielded a hit for the help topic. The result has three columns:

  - `name`: The topic name.
  - `description`: Descriptive help text for the topic.
  - `example`: Usage example or examples. This column might be blank.

  Example: `HELP 'replace'`

  Yields:

```
name: REPLACE
description: Syntax:
REPLACE(str,from_str,to_str)

Returns the string str with all occurrences of the string from_str
replaced by the string to_str. REPLACE() performs a case-sensitive
match when searching for from_str.
example: mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

- Result set containing multiple rows with two columns

This means that the search string matched many help topics. The result set indicates the help topic names:

- `name`: The help topic name.
- `is_it_category`: `Y` if the name represents a help category, `N` if it does not. If it does not, the `name` value when specified as the argument to the `HELP` statement should yield a single-row result set containing a description for the named item.

Example: `HELP 'status'`

Yields:

```
+----------------------+----------------+
| name                 | is_it_category |
+----------------------+----------------+
| SHOW                 | N              |
| SHOW ENGINE          | N              |
| SHOW MASTER STATUS   | N              |
| SHOW PROCEDURE STATUS | N              |
| SHOW SLAVE STATUS    | N              |
| SHOW STATUS          | N              |
| SHOW TABLE STATUS    | N              |
+----------------------+----------------+
```

- Result set containing multiple rows with three columns

  This means the search string matches a category. The result set contains category entries:

  - `source_category_name`: The help category name.
  - `name`: The category or topic name
  - `is_it_category`: `Y` if the name represents a help category, `N` if it does not. If it does not, the `name` value when specified as the argument to the `HELP` statement should yield a single-row result set containing a description for the named item.

  Example: `HELP 'functions'`

  Yields:

```
+---------------------+-------------------------+----------------+
| source_category_name | name                    | is_it_category |
+---------------------+-------------------------+----------------+
| Functions           | CREATE FUNCTION         | N              |
| Functions           | DROP FUNCTION           | N              |
| Functions           | Bit Functions           | Y              |
| Functions           | Comparison operators    | Y              |
| Functions           | Control flow functions  | Y              |
| Functions           | Date and Time Functions | Y              |
| Functions           | Encryption Functions    | Y              |
| Functions           | Information Functions   | Y              |
| Functions           | Logical operators       | Y              |
| Functions           | Miscellaneous Functions | Y              |
| Functions           | Numeric Functions       | Y              |
| Functions           | String Functions        | Y              |
+---------------------+-------------------------+----------------+
```

13.8.4 USE Syntax

```
USE db_name
```

The USE statement tells MySQL to use the named database as the default (current) database for subsequent statements. This statement requires some privilege for the database or some object within it.

The named database remains the default until the end of the session or another USE statement is issued:

```
USE db1;
SELECT COUNT(*) FROM mytable;   # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

The database name must be specified on a single line. Newlines in database names are not supported.

Making a particular database the default by means of the USE statement does not preclude accessing tables in other databases. The following example accesses the `author` table from the `db1` database and the `editor` table from the `db2` database:

```
USE db1;
SELECT author_name,editor_name FROM author,db2.editor
  WHERE author.editor_id = db2.editor.editor_id;
```

Chapter 14 MySQL Data Dictionary

Table of Contents

14.1 Data Dictionary Schema 2709
14.2 Removal of File-based Metadata Storage 2711
14.3 Transactional Storage of Dictionary Data 2711
14.4 Dictionary Object Cache 2711
14.5 INFORMATION_SCHEMA and Data Dictionary Integration 2712
14.6 Serialized Dictionary Information (SDI) 2714
14.7 Data Dictionary Usage Differences 2715
14.8 Data Dictionary Limitations 2716

MySQL Server incorporates a transactional data dictionary that stores information about database objects. In previous MySQL releases, dictionary data was stored in metadata files, nontransactional tables, and storage engine-specific data dictionaries.

This chapter describes the main features, benefits, usage differences, and limitations of the data dictionary. For other implications of the data dictionary feature, refer to the “Data Dictionary Notes” section in the MySQL 8.0 Release Notes.

Benefits of the MySQL data dictionary include:

- Simplicity of a centralized data dictionary schema that uniformly stores dictionary data. See Section 14.1, “Data Dictionary Schema”.
- Removal of file-based metadata storage. See Section 14.2, “Removal of File-based Metadata Storage”.
- Transactional, crash-safe storage of dictionary data. See Section 14.3, “Transactional Storage of Dictionary Data”.
- Uniform and centralized caching for dictionary objects. See Section 14.4, “Dictionary Object Cache”.
- A simpler and improved implementation for some `INFORMATION_SCHEMA` tables. See Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”.
- Atomic DDL. See Section 13.1.1, “Atomic Data Definition Statement Support”.

Important

A data dictionary-enabled server entails some general operational differences compared to a server that does not have a data dictionary; see Section 14.7, “Data Dictionary Usage Differences”. Also, for upgrades to MySQL 8.0, the upgrade procedure differs somewhat from previous MySQL releases and requires that you verify the upgrade readiness of your installation by checking specific prerequisites. For more information, see Section 2.11, “Upgrading MySQL”, particularly Section 2.11.5, “Preparing Your Installation for Upgrade”.

14.1 Data Dictionary Schema

Data dictionary tables are protected and may only be accessed in debug builds of MySQL. However, MySQL supports access to data stored in data dictionary tables through `INFORMATION_SCHEMA` tables and `SHOW` statements. For an overview of the tables that comprise the data dictionary, see Data Dictionary Tables.

MySQL system tables still exist in MySQL 8.0 and can be viewed by issuing a `SHOW TABLES` statement on the `mysql` system database. Generally, the difference between MySQL data dictionary tables and system tables is that data dictionary tables contain metadata required to execute SQL queries, whereas system tables contain auxiliary data such as time zone and help information. MySQL system tables and data dictionary tables also differ in how they are upgraded. The MySQL server manages data dictionary upgrades. SQL server. See How the Data Dictionary is Upgraded. Upgrading MySQL system tables requires running the full MySQL upgrade procedure. See Section 2.11.3, “What the MySQL Upgrade Process Upgrades”.

How the Data Dictionary is Upgraded

New versions of MySQL may include changes to data dictionary table definitions. Such changes are present in newly installed versions of MySQL, but when performing an in-place upgrade of MySQL binaries, changes are applied when the MySQL server is restarted using the new binaries. At startup, the data dictionary version of the server is compared to the version information stored in the data dictionary to determine if data dictionary tables should be upgraded. If an upgrade is necessary and supported, the server creates data dictionary tables with updated definitions, copies persisted metadata to the new tables, atomically replaces the old tables with the new ones, and reinitializes the data dictionary. If an upgrade is not necessary, startup continues without updating the data dictionary tables.

Upgrade of data dictionary tables is an atomic operation, which means that all of the data dictionary tables are upgraded as necessary or the operation fails. If the upgrade operation fails, server startup fails with an error. In this case, the old server binaries can be used with the old data directory to start the server. When the new server binaries are used again to start the server, the data dictionary upgrade is reattempted.

Generally, after data dictionary tables are successfully upgraded, it is not possible to restart the server using the old server binaries. As a result, downgrading MySQL server binaries to a previous MySQL version is not supported after data dictionary tables are upgraded.

The `mysqld --no-dd-upgrade` option can be used to prevent automatic upgrade of data dictionary tables at startup. When `--no-dd-upgrade` is specified, and the server finds that the data dictionary version of the server is different from the version stored in the data dictionary, startup fails with an error stating that the data dictionary upgrade is prohibited.

Viewing Data Dictionary Tables Using a Debug Build of MySQL

Data dictionary tables are protected by default but can be accessed by compiling MySQL with debugging support (using the `-DWITH_DEBUG=1 CMake` option) and specifying the `+d,skip_dd_table_access_check debug` option and modifier. For information about compiling debug builds, see Section 29.5.1.1, “Compiling MySQL for Debugging”.

Warning

Modifying or writing to data dictionary tables directly is not recommended and may render your MySQL instance inoperable.

After compiling MySQL with debugging support, use this `SET` statement to make data dictionary tables visible to the `mysql` client session:

```
mysql> SET SESSION debug='+d,skip_dd_table_access_check';
```

Use this query to retrieve a list of data dictionary tables:

```
mysql> SELECT name, schema_id, hidden, type FROM mysql.tables where schema_id=1 AND hidden='System';
```

Use `SHOW CREATE TABLE` to view data dictionary table definitions. For example:

```
mysql> SHOW CREATE TABLE mysql.catalogs\G
```

14.2 Removal of File-based Metadata Storage

In previous MySQL releases, dictionary data was partially stored in metadata files. Issues with file-based metadata storage included expensive file scans, susceptibility to file system-related bugs, complex code for handling of replication and crash recovery failure states, and a lack of extensibility that made it difficult to add metadata for new features and relational objects.

The metadata files listed below are removed from MySQL. Unless otherwise noted, data previously stored in metadata files is now stored in data dictionary tables.

- `.frm` files: Table metadata files. With the removal of `.frm` files:
  - The 64KB table definition size limit imposed by the `.frm` file structure is removed.
  - The `INFORMATION_SCHEMA.TABLES VERSION` column reports a hardcoded value of `10`, which is the last `.frm` file version used in MySQL 5.7.
- `.par` files: Partition definition files. `InnoDB` stopped using partition definition files in MySQL 5.7 with the introduction of native partitioning support for `InnoDB` tables.
- `.TRN` files: Trigger namespace files.
- `.TRG` files: Trigger parameter files.
- `.isl` files: `InnoDB` Symbolic Link files containing the location of file-per-table tablespace files created outside of the data directory.
- `db.opt` files: Database configuration files. These files, one per database directory, contained database default character set attributes.

14.3 Transactional Storage of Dictionary Data

The data dictionary schema stores dictionary data in transactional (`InnoDB`) tables. Data dictionary tables are located in the `mysql` database together with non-data dictionary system tables.

Data dictionary tables are created in a single `InnoDB` tablespace named `mysql.ibd`, which resides in the MySQL data directory. The `mysql.ibd` tablespace file must reside in the MySQL data directory and its name cannot be modified or used by another tablespace.

Dictionary data is protected by the same commit, rollback, and crash-recovery capabilities that protect user data that is stored in `InnoDB` tables.

14.4 Dictionary Object Cache

The dictionary object cache is a shared global cache that stores previously accessed data dictionary objects in memory to enable object reuse and minimize disk I/O. Similar to other cache mechanisms used by MySQL, the dictionary object cache uses an LRU-based eviction strategy to evict least recently used objects from memory.

The dictionary object cache comprises cache partitions that store different object types. Some cache partition size limits are configurable, whereas others are hardcoded.

- **tablespace definition cache partition**: Stores tablespace definition objects. The `tablespace_definition_cache` option sets a limit for the number of tablespace definition objects that can be stored in the dictionary object cache. The default value is 256.
- **schema definition cache partition**: Stores schema definition objects. The `schema_definition_cache` option sets a limit for the number of schema definition objects that can be stored in the dictionary object cache. The default value is 256.
- **table definition cache partition**: Stores table definition objects. The object limit is set to the value of `max_connections`, which has a default value of 151.

  The table definition cache partition exists in parallel with the table definition cache that is configured using the `table_definition_cache` configuration option. Both caches store table definitions but serve different parts of the MySQL server. Objects in one cache have no dependence on the existence of objects in the other.
- **stored program definition cache partition**: Stores stored program definition objects. The `stored_program_definition_cache` option sets a limit for the number of stored program definition objects that can be stored in the dictionary object cache. The default value is 256.

  The stored program definition cache partition exists in parallel with the stored procedure and stored function caches that are configured using the `stored_program_cache` option.

  The `stored_program_cache` option sets a soft upper limit for the number of cached stored procedures or functions per connection, and the limit is checked each time a connection executes a stored procedure or function. The stored program definition cache partition, on the other hand, is a shared cache that stores stored program definition objects for other purposes. The existence of objects in the stored program definition cache partition has no dependence on the existence of objects in the stored procedure cache or stored function cache, and vice versa.
- **character set definition cache partition**: Stores character set definition objects and has a hardcoded object limit of 256.
- **collation definition cache partition**: Stores collation definition objects and has a hardcoded object limit of 256.

For information about valid values for dictionary object cache configuration options, refer to Section 5.1.8, “Server System Variables”.

14.5 INFORMATION_SCHEMA and Data Dictionary Integration

With the introduction of the data dictionary, the following `INFORMATION_SCHEMA` tables are implemented as views on data dictionary tables:

- `CHARACTER_SETS`
- `CHECK_CONSTRAINTS`
- `COLLATIONS`
- `COLLATION_CHARACTER_SET_APPLICABILITY`
- `COLUMNS`
- `COLUMN_STATISTICS`
- `EVENTS`

- FILES
- INNODB_COLUMNS
- INNODB_DATAFILES
- INNODB_FIELDS
- INNODB_FOREIGN
- INNODB_FOREIGN_COLS
- INNODB_INDEXES
- INNODB_TABLES
- INNODB_TABLESPACES
- INNODB_TABLESPACES_BRIEF
- INNODB_TABLESTATS
- KEY_COLUMN_USAGE
- KEYWORDS
- PARAMETERS
- PARTITIONS
- REFERENTIAL_CONSTRAINTS
- RESOURCE_GROUPS
- ROUTINES
- SCHEMATA
- STATISTICS
- ST_GEOMETRY_COLUMNS
- ST_SPATIAL_REFERENCE_SYSTEMS
- TABLES
- TABLE_CONSTRAINTS
- TRIGGERS
- VIEWS
- VIEW_ROUTINE_USAGE
- VIEW_TABLE_USAGE

Queries on those tables are now more efficient because they obtain information from data dictionary tables rather than by other, slower means. In particular, for each INFORMATION_SCHEMA table that is a view on data dictionary tables:

- The server no longer must create a temporary table for each query of the `INFORMATION_SCHEMA` table.
- When the underlying data dictionary tables store values previously obtained by directory scans (for example, to enumerate database names or table names within databases) or file-opening operations (for example, to read information from `.frm` files), `INFORMATION_SCHEMA` queries for those values now use table lookups instead. (Additionally, even for a non-view `INFORMATION_SCHEMA` table, values such as database and table names are retrieved by lookups from the data dictionary and do not require directory or file scans.)
- Indexes on the underlying data dictionary tables permit the optimizer to construct efficient query execution plans, something not true for the previous implementation that processed the `INFORMATION_SCHEMA` table using a temporary table per query.

The preceding improvements also apply to `SHOW` statements that display information corresponding to the `INFORMATION_SCHEMA` tables that are views on data dictionary tables. For example, `SHOW DATABASES` displays the same information as the `SCHEMATA` table.

In addition to the introduction of views on data dictionary tables, table statistics contained in the `STATISTICS` and `TABLES` tables is now cached to improve `INFORMATION_SCHEMA` query performance. The `information_schema_stats_expiry` system variable defines the period of time before cached table statistics expire. The default is 86400 seconds (24 hours). If there are no cached statistics or statistics have expired, statistics are retrieved from storage engine when querying table statistics columns. To update cached values at any time for a given table, use `ANALYZE TABLE`

`information_schema_stats_expiry` can be set to `0` to have `INFORMATION_SCHEMA` queries retrieve the latest statistics directly from the storage engine, which is not as fast as retrieving cached statistics.

For more information, see Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”.

`INFORMATION_SCHEMA` tables in MySQL 8.0 are closely tied to the data dictionary, resulting in several usage differences. See Section 14.7, “Data Dictionary Usage Differences”.

14.6 Serialized Dictionary Information (SDI)

In addition to storing metadata about database objects in the data dictionary, MySQL stores it in serialized form. This data is referred to as serialized dictionary information (SDI). `InnoDB` stores SDI data within its tablespace files. Other storage engines store SDI data in `.sdi` files that are created in the schema directory. SDI data is generated in a compact `JSON` format.

Serialized dictionary information (SDI) is present in all `InnoDB` tablespace files except for temporary tablespace and undo tablespace files. SDI records in an `InnoDB` tablespace file only describe table and tablespace objects contained within the tablespace.

SDI data in within an `InnoDB` tablespace file is only updated by DDL operations on tables within the tablespace.

The presence of SDI data provides metadata redundancy. For example, if the data dictionary becomes unavailable, object metadata can be extracted directly from `InnoDB` tablespace files using the `ibd2sdi` tool.

For `InnoDB`, an SDI record requires a single index page, which is 16KB in size by default. However, SDI data is compressed to reduce the storage footprint.

For partitioned `InnoDB` tables comprised of multiple tablespaces, SDI data is stored in the tablespace file of the first partition.

The MySQL server uses an internal API that is accessed during DDL operations to create and maintain SDI records.

The `IMPORT TABLE` statement imports `MyISAM` tables based on information contained in `.sdi` files. For more information, see Section 13.2.5, “IMPORT TABLE Syntax”.

14.7 Data Dictionary Usage Differences

Use of a data dictionary-enabled MySQL server entails some operational differences compared to a server that does not have a data dictionary:

- Previously, enabling the `innodb_read_only` system variable prevented creating and dropping tables only for the `InnoDB` storage. As of MySQL 8.0, enabling `innodb_read_only` prevents these operations for all storage engines. Table creation and drop operations for any storage engine modify data dictionary tables in the `mysql` system database, but those tables use the `InnoDB` storage engine and cannot be modified when `innodb_read_only` is enabled. The same principle applies to other table operations that require modifying data dictionary tables. Examples:

  - `ANALYZE TABLE` fails because it updates table statistics, which are stored in the data dictionary.
  - `ALTER TABLE tbl_name ENGINE=engine_name` fails because it updates the storage engine designation, which is stored in the data dictionary.

  Note

  Enabling `innodb_read_only` also has important implications for non-data dictionary tables in the `mysql` system database. For details, see the description of `innodb_read_only` in Section 15.13, “InnoDB Startup Options and System Variables”

- Previously, tables in the `mysql` system database were visible to DML and DDL statements. As of MySQL 8.0, data dictionary tables are invisible and cannot be modified or queried directly. However, in most cases there are corresponding `INFORMATION_SCHEMA` tables that can be queried instead. This enables the underlying data dictionary tables to be changed as server development proceeds, while maintaining a stable `INFORMATION_SCHEMA` interface for application use.

- `INFORMATION_SCHEMA` tables in MySQL 8.0 are closely tied to the data dictionary, resulting in several usage differences:

  - Previously, `INFORMATION_SCHEMA` queries for table statistics in the `STATISTICS` and `TABLES` tables retrieved statistics directly from storage engines. As of MySQL 8.0, cached table statistics are used by default. The `information_schema_stats_expiry` system variable defines the period of time before cached table statistics expire. The default is 86400 seconds (24 hours). (To update the cached values at any time for a given table, use `ANALYZE TABLE`.) If there are no cached statistics or statistics have expired, statistics are retrieved from storage engines when querying table statistics columns. To always retrieve the latest statistics directly from storage engines, set `information_schema_stats_expiry` to `0`. For more information, see Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”.

  - Several `INFORMATION_SCHEMA` tables are views on data dictionary tables, which enables the optimizer to use indexes on those underlying tables. Consequently, depending on optimizer choices, the row order of results for `INFORMATION_SCHEMA` queries might differ from previous results. If a query result must have specific row ordering characteristics, include an `ORDER BY` clause.

  - Queries on `INFORMATION_SCHEMA` tables may return column names in a different lettercase than in earlier MySQL series. Applications should test result set column names in case-insensitive fashion. If

that is not feasible, a workaround is to use column aliases in the select list that return column names in the required lettercase. For example:

```
SELECT TABLE_SCHEMA AS table_schema, TABLE_NAME AS table_name
FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME = 'users';
```

- `mysqldump` and `mysqlpump` no longer dump the `INFORMATION_SCHEMA` database, even if explicitly named on the command line.
- `CREATE TABLE dst_tbl LIKE src_tbl` requires that *`src_tbl`* be a base table and fails if it is an `INFORMATION_SCHEMA` table that is a view on data dictionary tables.
- Previously, result set headers of columns selected from `INFORMATION_SCHEMA` tables used the capitalization specified in the query. This query produces a result set with a header of `table_name`:

  ```
  SELECT table_name FROM INFORMATION_SCHEMA.TABLES;
  ```

  As of MySQL 8.0, these headers are capitalized; the preceding query produces a result set with a header of `TABLE_NAME`. If necessary, a column alias can be used to achieve a different lettercase. For example:

  ```
  SELECT table_name AS 'table_name' FROM INFORMATION_SCHEMA.TABLES;
  ```

- The data directory affects how `mysqldump` and `mysqlpump` dump information from the `mysql` system database:
  - Previously, it was possible to dump all tables in the `mysql` system database. As of MySQL 8.0, `mysqldump` and `mysqlpump` dump only non-data dictionary tables in that database.
  - Previously, the `--routines` and `--events` options were not required to include stored routines and events when using the `--all-databases` option: The dump included the `mysql` system database, and therefore also the `proc` and `event` tables containing stored routine and event definitions. As of MySQL 8.0, the `event` and `proc` tables are not used. Definitions for the corresponding objects are stored in data dictionary tables, but those tables are not dumped. To include stored routines and events in a dump made using `--all-databases`, use the `--routines` and `--events` options explicitly.
  - Previously, the `--routines` option required the `SELECT` privilege for the `proc` table. As of MySQL 8.0, that table is not used; `--routines` requires the global `SELECT` privilege instead.
  - Previously, it was possible to dump stored routine and event definitions together with their creation and modification timestamps, by dumping the `proc` and `event` tables. As of MySQL 8.0, those tables are not used, so it is not possible to dump timestamps.
- Previously, creating a stored routine that contains illegal characters produced a warning. As of MySQL 8.0, this is an error.

14.8 Data Dictionary Limitations

This section describes temporary limitations introduced with the MySQL data dictionary.

- Manual creation of database directories under the data directory (for example, with `mkdir`) is unsupported. Manually created database directories are not recognized by the MySQL Server.
- DDL operations take longer due to writing to storage, undo logs, and redo logs instead of `.frm` files.

Chapter 15 The InnoDB Storage Engine

Table of Contents

15.1 Introduction to InnoDB 2718
15.1.1 Benefits of Using InnoDB Tables 2720
15.1.2 Best Practices for InnoDB Tables 2721
15.1.3 Verifying that InnoDB is the Default Storage Engine 2722
15.1.4 Testing and Benchmarking with InnoDB 2722
15.2 InnoDB and the ACID Model 2722
15.3 InnoDB Multi-Versioning 2724
15.4 InnoDB Architecture 2725
15.5 InnoDB In-Memory Structures 2726
15.5.1 Buffer Pool 2726
15.5.2 Change Buffer 2731
15.5.3 Adaptive Hash Index 2735
15.5.4 Log Buffer 2735
15.6 InnoDB On-Disk Structures 2736
15.6.1 Tables 2736
15.6.2 Indexes 2757
15.6.3 Tablespaces 2765
15.6.4 Doublewrite Buffer 2804
15.6.5 Redo Log 2805
15.6.6 Undo Logs 2808
15.7 InnoDB Locking and Transaction Model 2810
15.7.1 InnoDB Locking 2810
15.7.2 InnoDB Transaction Model 2815
15.7.3 Locks Set by Different SQL Statements in InnoDB 2824
15.7.4 Phantom Rows 2827
15.7.5 Deadlocks in InnoDB 2828
15.8 InnoDB Configuration 2832
15.8.1 InnoDB Startup Configuration 2832
15.8.2 Configuring InnoDB for Read-Only Operation 2838
15.8.3 InnoDB Buffer Pool Configuration 2839
15.8.4 Configuring Thread Concurrency for InnoDB 2854
15.8.5 Configuring the Number of Background InnoDB I/O Threads 2855
15.8.6 Using Asynchronous I/O on Linux 2855
15.8.7 Configuring InnoDB I/O Capacity 2856
15.8.8 Configuring Spin Lock Polling 2856
15.8.9 Purge Configuration 2857
15.8.10 Configuring Optimizer Statistics for InnoDB 2859
15.8.11 Configuring the Merge Threshold for Index Pages 2870
15.8.12 Enabling Automatic Configuration for a Dedicated MySQL Server 2873
15.9 InnoDB Table and Page Compression 2875
15.9.1 InnoDB Table Compression 2875
15.9.2 InnoDB Page Compression 2889
15.10 InnoDB Row Formats 2893
15.11 InnoDB Disk I/O and File Space Management 2899
15.11.1 InnoDB Disk I/O 2900
15.11.2 File Space Management 2900
15.11.3 InnoDB Checkpoints 2902

15.11.4 Defragmenting a Table 2902
15.11.5 Reclaiming Disk Space with TRUNCATE TABLE 2903
15.12 InnoDB and Online DDL 2903
15.12.1 Online DDL Operations 2904
15.12.2 Online DDL Performance and Concurrency 2918
15.12.3 Online DDL Space Requirements 2922
15.12.4 Simplifying DDL Statements with Online DDL 2923
15.12.5 Online DDL Failure Conditions 2923
15.12.6 Online DDL Limitations 2924
15.13 InnoDB Startup Options and System Variables 2925
15.14 InnoDB INFORMATION_SCHEMA Tables 3019
15.14.1 InnoDB INFORMATION_SCHEMA Tables about Compression 3019
15.14.2 InnoDB INFORMATION_SCHEMA Transaction and Locking Information 3021
15.14.3 InnoDB INFORMATION_SCHEMA Schema Object Tables 3028
15.14.4 InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables 3033
15.14.5 InnoDB INFORMATION_SCHEMA Buffer Pool Tables 3037
15.14.6 InnoDB INFORMATION_SCHEMA Metrics Table 3041
15.14.7 InnoDB INFORMATION_SCHEMA Temporary Table Info Table 3050
15.14.8 Retrieving InnoDB Tablespace Metadata from INFORMATION_SCHEMA.FILES 3051
15.15 InnoDB Integration with MySQL Performance Schema 3053
15.15.1 Monitoring ALTER TABLE Progress for InnoDB Tables Using Performance Schema 3055
15.15.2 Monitoring InnoDB Mutex Waits Using Performance Schema 3057
15.16 InnoDB Monitors 3061
15.16.1 InnoDB Monitor Types 3061
15.16.2 Enabling InnoDB Monitors 3061
15.16.3 InnoDB Standard Monitor and Lock Monitor Output 3063
15.17 InnoDB Backup and Recovery 3067
15.17.1 InnoDB Backup 3068
15.17.2 InnoDB Recovery 3068
15.18 InnoDB and MySQL Replication 3071
15.19 InnoDB memcached Plugin 3073
15.19.1 Benefits of the InnoDB memcached Plugin 3073
15.19.2 InnoDB memcached Architecture 3075
15.19.3 Setting Up the InnoDB memcached Plugin 3078
15.19.4 InnoDB memcached Multiple get and Range Query Support 3084
15.19.5 Security Considerations for the InnoDB memcached Plugin 3087
15.19.6 Writing Applications for the InnoDB memcached Plugin 3088
15.19.7 The InnoDB memcached Plugin and Replication 3101
15.19.8 InnoDB memcached Plugin Internals 3105
15.19.9 Troubleshooting the InnoDB memcached Plugin 3110
15.20 InnoDB Troubleshooting 3112
15.20.1 Troubleshooting InnoDB I/O Problems 3112
15.20.2 Forcing InnoDB Recovery 3113
15.20.3 Troubleshooting InnoDB Data Dictionary Operations 3114
15.20.4 InnoDB Error Handling 3116

15.1 Introduction to InnoDB

`InnoDB` is a general-purpose storage engine that balances high reliability and high performance. In MySQL 8.0, `InnoDB` is the default MySQL storage engine. Unless you have configured a different default storage engine, issuing a `CREATE TABLE` statement without an `ENGINE=` clause creates an `InnoDB` table.

Key Advantages of InnoDB

- Its DML operations follow the ACID model, with transactions featuring commit, rollback, and crash-recovery capabilities to protect user data. See Section 15.2, “InnoDB and the ACID Model” for more information.
- Row-level locking and Oracle-style consistent reads increase multi-user concurrency and performance. See Section 15.7, “InnoDB Locking and Transaction Model” for more information.
- `InnoDB` tables arrange your data on disk to optimize queries based on primary keys. Each `InnoDB` table has a primary key index called the clustered index that organizes the data to minimize I/O for primary key lookups. See Section 15.6.2.1, “Clustered and Secondary Indexes” for more information.
- To maintain data integrity, `InnoDB` supports `FOREIGN KEY` constraints. With foreign keys, inserts, updates, and deletes are checked to ensure they do not result in inconsistencies across different tables. See Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints” for more information.

Table 15.1 InnoDB Storage Engine Features

Feature	Support
B-tree indexes	Yes
Backup/point-in-time recovery (Implemented in the server, rather than in the storage engine.)	Yes
Cluster database support	No
Clustered indexes	Yes
Compressed data	Yes
Data caches	Yes
Encrypted data	Yes (Implemented in the server via encryption functions; In MySQL 5.7 and later, data-at-rest tablespace encryption is supported.)
Foreign key support	Yes
Full-text search indexes	Yes (InnoDB support for FULLTEXT indexes is available in MySQL 5.6 and later.)
Geospatial data type support	Yes
Geospatial indexing support	Yes (InnoDB support for geospatial indexing is available in MySQL 5.7 and later.)
Hash indexes	No (InnoDB utilizes hash indexes internally for its Adaptive Hash Index feature.)
Index caches	Yes
Locking granularity	Row
MVCC	Yes
Replication support (Implemented in the server, rather than in the storage engine.)	Yes
Storage limits	64TB

Feature	Support
T-tree indexes	No
Transactions	Yes
Update statistics for data dictionary	Yes

To compare the features of InnoDB with other storage engines provided with MySQL, see the *Storage Engine Features* table in Chapter 16, *Alternative Storage Engines*.

InnoDB Enhancements and New Features

For information about InnoDB enhancements and new features, refer to:

- The InnoDB enhancements list in Section 1.4, “What Is New in MySQL 8.0”.
- The Release Notes.

Additional InnoDB Information and Resources

- For InnoDB-related terms and definitions, see the MySQL Glossary.
- For a forum dedicated to the InnoDB storage engine, see MySQL Forums::InnoDB.
- InnoDB is published under the same GNU GPL License Version 2 (of June 1991) as MySQL. For more information on MySQL licensing, see http://www.mysql.com/company/legal/licensing/.

15.1.1 Benefits of Using InnoDB Tables

You may find InnoDB tables beneficial for the following reasons:

- If your server crashes because of a hardware or software issue, regardless of what was happening in the database at the time, you don't need to do anything special after restarting the database. InnoDB crash recovery automatically finalizes any changes that were committed before the time of the crash, and undoes any changes that were in process but not committed. Just restart and continue where you left off.
- The InnoDB storage engine maintains its own buffer pool that caches table and index data in main memory as data is accessed. Frequently used data is processed directly from memory. This cache applies to many types of information and speeds up processing. On dedicated database servers, up to 80% of physical memory is often assigned to the buffer pool.
- If you split up related data into different tables, you can set up foreign keys that enforce referential integrity. Update or delete data, and the related data in other tables is updated or deleted automatically. Try to insert data into a secondary table without corresponding data in the primary table, and the bad data gets kicked out automatically.
- If data becomes corrupted on disk or in memory, a checksum mechanism alerts you to the bogus data before you use it.
- When you design your database with appropriate primary key columns for each table, operations involving those columns are automatically optimized. It is very fast to reference the primary key columns in WHERE clauses, ORDER BY clauses, GROUP BY clauses, and join operations.
- Inserts, updates, and deletes are optimized by an automatic mechanism called change buffering. InnoDB not only allows concurrent read and write access to the same table, it caches changed data to streamline disk I/O.

- Performance benefits are not limited to giant tables with long-running queries. When the same rows are accessed over and over from a table, a feature called the Adaptive Hash Index takes over to make these lookups even faster, as if they came out of a hash table.
- You can compress tables and associated indexes.
- You can create and drop indexes with much less impact on performance and availability.
- Truncating a file-per-table tablespace is very fast, and can free up disk space for the operating system to reuse, rather than freeing up space within the system tablespace that only `InnoDB` can reuse.
- The storage layout for table data is more efficient for `BLOB` and long text fields, with the DYNAMIC row format.
- You can monitor the internal workings of the storage engine by querying INFORMATION_SCHEMA tables.
- You can monitor the performance details of the storage engine by querying Performance Schema tables.
- You can freely mix `InnoDB` tables with tables from other MySQL storage engines, even within the same statement. For example, you can use a join operation to combine data from `InnoDB` and `MEMORY` tables in a single query.
- `InnoDB` has been designed for CPU efficiency and maximum performance when processing large data volumes.
- `InnoDB` tables can handle large quantities of data, even on operating systems where file size is limited to 2GB.

For `InnoDB`-specific tuning techniques you can apply in your application code, see Section 8.5, “Optimizing for InnoDB Tables”.

15.1.2 Best Practices for InnoDB Tables

This section describes best practices when using `InnoDB` tables.

- Specifying a primary key for every table using the most frequently queried column or columns, or an auto-increment value if there is no obvious primary key.
- Using joins wherever data is pulled from multiple tables based on identical ID values from those tables. For fast join performance, define foreign keys on the join columns, and declare those columns with the same data type in each table. Adding foreign keys ensures that referenced columns are indexed, which can improve performance. Foreign keys also propagate deletes or updates to all affected tables, and prevent insertion of data in a child table if the corresponding IDs are not present in the parent table.
- Turning off autocommit. Committing hundreds of times a second puts a cap on performance (limited by the write speed of your storage device).
- Grouping sets of related DML operations into transactions, by bracketing them with `START TRANSACTION` and `COMMIT` statements. While you don't want to commit too often, you also don't want to issue huge batches of `INSERT`, `UPDATE`, or `DELETE` statements that run for hours without committing.
- Not using `LOCK TABLES` statements. `InnoDB` can handle multiple sessions all reading and writing to the same table at once, without sacrificing reliability or high performance. To get exclusive write access to a set of rows, use the `SELECT ... FOR UPDATE` syntax to lock just the rows you intend to update.
- Enabling the `innodb_file_per_table` option or using general tablespaces to put the data and indexes for tables into separate files, instead of the system tablespace.

The `innodb_file_per_table` option is enabled by default.

- Evaluating whether your data and access patterns benefit from the `InnoDB` table or page compression features. You can compress `InnoDB` tables without sacrificing read/write capability.
- Running your server with the option `--sql_mode=NO_ENGINE_SUBSTITUTION` to prevent tables being created with a different storage engine if there is an issue with the engine specified in the `ENGINE=` clause of `CREATE TABLE`.

15.1.3 Verifying that InnoDB is the Default Storage Engine

Issue the `SHOW ENGINES` statement to view the available MySQL storage engines. Look for `DEFAULT` in the `InnoDB` line.

```
mysql> SHOW ENGINES;
```

Alternatively, query the `INFORMATION_SCHEMA.ENGINES` table.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.ENGINES;
```

15.1.4 Testing and Benchmarking with InnoDB

If `InnoDB` is not your default storage engine, you can determine if your database server or applications work correctly with `InnoDB` by restarting the server with `--default-storage-engine=InnoDB` defined on the command line or with `default-storage-engine=innodb` defined in the `[mysqld]` section of your MySQL server option file.

Since changing the default storage engine only affects new tables as they are created, run all your application installation and setup steps to confirm that everything installs properly. Then exercise all the application features to make sure all the data loading, editing, and querying features work. If a table relies on a feature that is specific to another storage engine, you will receive an error; add the `ENGINE=`*`other_engine_name`* clause to the `CREATE TABLE` statement to avoid the error.

If you did not make a deliberate decision about the storage engine, and you want to preview how certain tables work when created using `InnoDB`, issue the command `ALTER TABLE table_name ENGINE=InnoDB;` for each table. Or, to run test queries and other statements without disturbing the original table, make a copy:

```
CREATE TABLE InnoDB_Table (...) ENGINE=InnoDB AS SELECT * FROM other_engine_table;
```

To assess performance with a full application under a realistic workload, install the latest MySQL server and run benchmarks.

Test the full application lifecycle, from installation, through heavy usage, and server restart. Kill the server process while the database is busy to simulate a power failure, and verify that the data is recovered successfully when you restart the server.

Test any replication configurations, especially if you use different MySQL versions and options on the master and slaves.

15.2 InnoDB and the ACID Model

The ACID model is a set of database design principles that emphasize aspects of reliability that are important for business data and mission-critical applications. MySQL includes components such as the

InnoDB storage engine that adhere closely to the ACID model, so that data is not corrupted and results are not distorted by exceptional conditions such as software crashes and hardware malfunctions. When you rely on ACID-compliant features, you do not need to reinvent the wheel of consistency checking and crash recovery mechanisms. In cases where you have additional software safeguards, ultra-reliable hardware, or an application that can tolerate a small amount of data loss or inconsistency, you can adjust MySQL settings to trade some of the ACID reliability for greater performance or throughput.

The following sections discuss how MySQL features, in particular the InnoDB storage engine, interact with the categories of the ACID model:

- **A**: atomicity.
- **C**: consistency.
- **I:**: isolation.
- **D**: durability.

Atomicity

The **atomicity** aspect of the ACID model mainly involves InnoDB transactions. Related MySQL features include:

- Autocommit setting.
- COMMIT statement.
- ROLLBACK statement.
- Operational data from the INFORMATION_SCHEMA tables.

Consistency

The **consistency** aspect of the ACID model mainly involves internal InnoDB processing to protect data from crashes. Related MySQL features include:

- InnoDB doublewrite buffer.
- InnoDB crash recovery.

Isolation

The **isolation** aspect of the ACID model mainly involves InnoDB transactions, in particular the isolation level that applies to each transaction. Related MySQL features include:

- Autocommit setting.
- SET ISOLATION LEVEL statement.
- The low-level details of InnoDB locking. During performance tuning, you see these details through INFORMATION_SCHEMA tables.

Durability

The **durability** aspect of the ACID model involves MySQL software features interacting with your particular hardware configuration. Because of the many possibilities depending on the capabilities of your CPU,

network, and storage devices, this aspect is the most complicated to provide concrete guidelines for. (And those guidelines might take the form of buy “new hardware”.) Related MySQL features include:

- InnoDB doublewrite buffer, turned on and off by the innodb_doublewrite configuration option.
- Configuration option innodb_flush_log_at_trx_commit.
- Configuration option sync_binlog.
- Configuration option innodb_file_per_table.
- Write buffer in a storage device, such as a disk drive, SSD, or RAID array.
- Battery-backed cache in a storage device.
- The operating system used to run MySQL, in particular its support for the fsync() system call.
- Uninterruptible power supply (UPS) protecting the electrical power to all computer servers and storage devices that run MySQL servers and store MySQL data.
- Your backup strategy, such as frequency and types of backups, and backup retention periods.
- For distributed or hosted data applications, the particular characteristics of the data centers where the hardware for the MySQL servers is located, and network connections between the data centers.

15.3 InnoDB Multi-Versioning

InnoDB is a multi-versioned storage engine: it keeps information about old versions of changed rows, to support transactional features such as concurrency and rollback. This information is stored in the tablespace in a data structure called a rollback segment (after an analogous data structure in Oracle). InnoDB uses the information in the rollback segment to perform the undo operations needed in a transaction rollback. It also uses the information to build earlier versions of a row for a consistent read.

Internally, InnoDB adds three fields to each row stored in the database. A 6-byte DB_TRX_ID field indicates the transaction identifier for the last transaction that inserted or updated the row. Also, a deletion is treated internally as an update where a special bit in the row is set to mark it as deleted. Each row also contains a 7-byte DB_ROLL_PTR field called the roll pointer. The roll pointer points to an undo log record written to the rollback segment. If the row was updated, the undo log record contains the information necessary to rebuild the content of the row before it was updated. A 6-byte DB_ROW_ID field contains a row ID that increases monotonically as new rows are inserted. If InnoDB generates a clustered index automatically, the index contains row ID values. Otherwise, the DB_ROW_ID column does not appear in any index.

Undo logs in the rollback segment are divided into insert and update undo logs. Insert undo logs are needed only in transaction rollback and can be discarded as soon as the transaction commits. Update undo logs are used also in consistent reads, but they can be discarded only after there is no transaction present for which InnoDB has assigned a snapshot that in a consistent read could need the information in the update undo log to build an earlier version of a database row.

Commit your transactions regularly, including those transactions that issue only consistent reads. Otherwise, InnoDB cannot discard data from the update undo logs, and the rollback segment may grow too big, filling up your tablespace.

The physical size of an undo log record in the rollback segment is typically smaller than the corresponding inserted or updated row. You can use this information to calculate the space needed for your rollback segment.

In the InnoDB multi-versioning scheme, a row is not physically removed from the database immediately when you delete it with an SQL statement. InnoDB only physically removes the corresponding row and its index records when it discards the update undo log record written for the deletion. This removal operation is called a purge, and it is quite fast, usually taking the same order of time as the SQL statement that did the deletion.

If you insert and delete rows in smallish batches at about the same rate in the table, the purge thread can start to lag behind and the table can grow bigger and bigger because of all the “dead” rows, making everything disk-bound and very slow. In such a case, throttle new row operations, and allocate more resources to the purge thread by tuning the innodb_max_purge_lag system variable. See Section 15.13, “InnoDB Startup Options and System Variables” for more information.

Multi-Versioning and Secondary Indexes

InnoDB multiversion concurrency control (MVCC) treats secondary indexes differently than clustered indexes. Records in a clustered index are updated in-place, and their hidden system columns point undo log entries from which earlier versions of records can be reconstructed. Unlike clustered index records, secondary index records do not contain hidden system columns nor are they updated in-place.

When a secondary index column is updated, old secondary index records are delete-marked, new records are inserted, and delete-marked records are eventually purged. When a secondary index record is delete-marked or the secondary index page is updated by a newer transaction, InnoDB looks up the database record in the clustered index. In the clustered index, the record's DB_TRX_ID is checked, and the correct version of the record is retrieved from the undo log if the record was modified after the reading transaction was initiated.

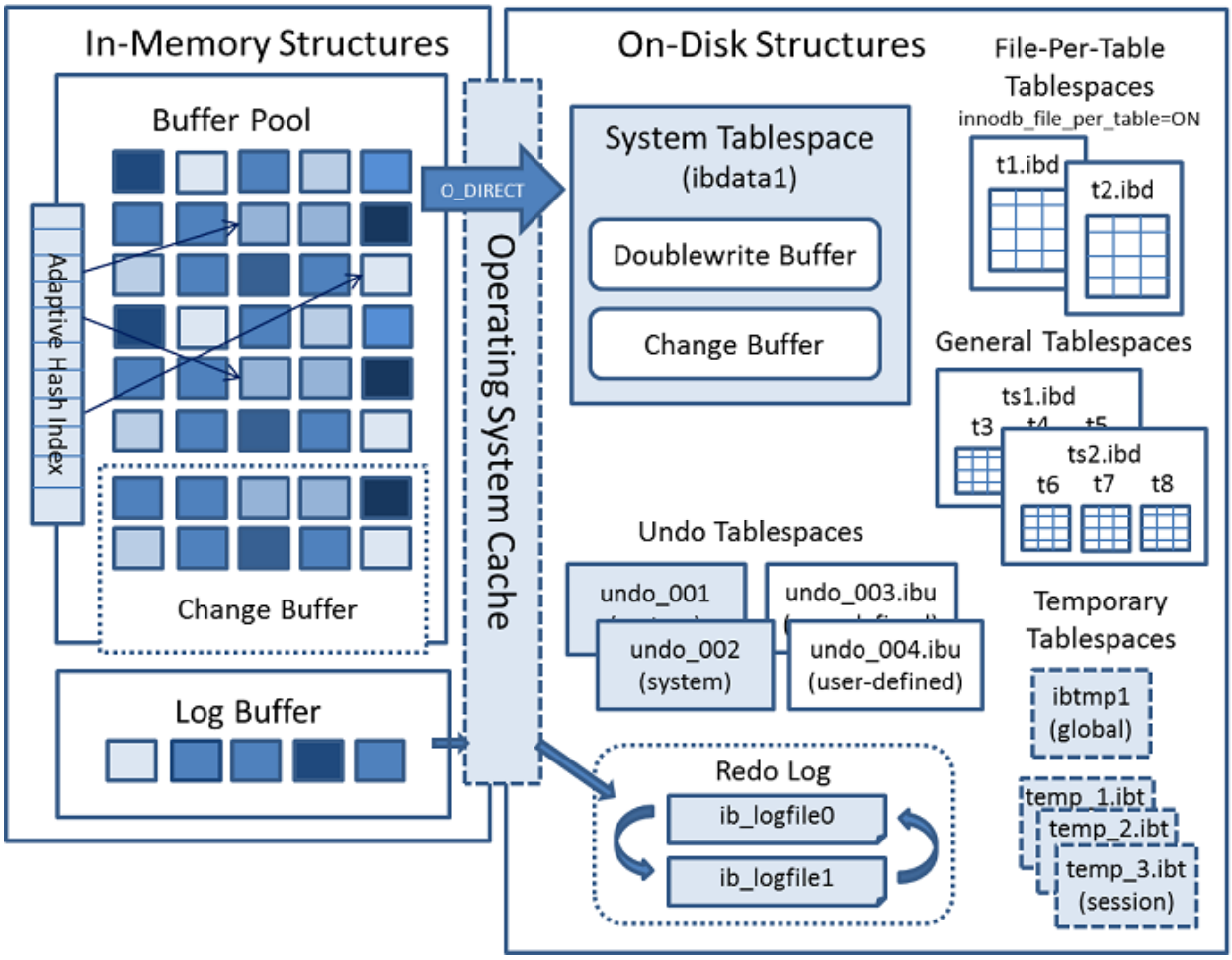
If a secondary index record is marked for deletion or the secondary index page is updated by a newer transaction, the covering index technique is not used. Instead of returning values from the index structure, InnoDB looks up the record in the clustered index.

However, if the index condition pushdown (ICP) optimization is enabled, and parts of the WHERE condition can be evaluated using only fields from the index, the MySQL server still pushes this part of the WHERE condition down to the storage engine where it is evaluated using the index. If no matching records are found, the clustered index lookup is avoided. If matching records are found, even among delete-marked records, InnoDB looks up the record in the clustered index.

15.4 InnoDB Architecture

The following diagram shows in-memory and on-disk structures that comprise the InnoDB storage engine architecture. For information about each structure, see Section 15.5, “InnoDB In-Memory Structures”, and Section 15.6, “InnoDB On-Disk Structures”.

Figure 15.1 InnoDB Architecture



15.5 InnoDB In-Memory Structures

This section describes `InnoDB` in-memory structures and related topics.

15.5.1 Buffer Pool

The buffer pool is an area in main memory where `InnoDB` caches table and index data as it is accessed. The buffer pool permits frequently used data to be processed directly from memory, which speeds up processing. On dedicated servers, up to 80% of physical memory is often assigned to the buffer pool.

For efficiency of high-volume read operations, the buffer pool is divided into pages that can potentially hold multiple rows. For efficiency of cache management, the buffer pool is implemented as a linked list of pages; data that is rarely used is aged out of the cache using a variation of the LRU algorithm.

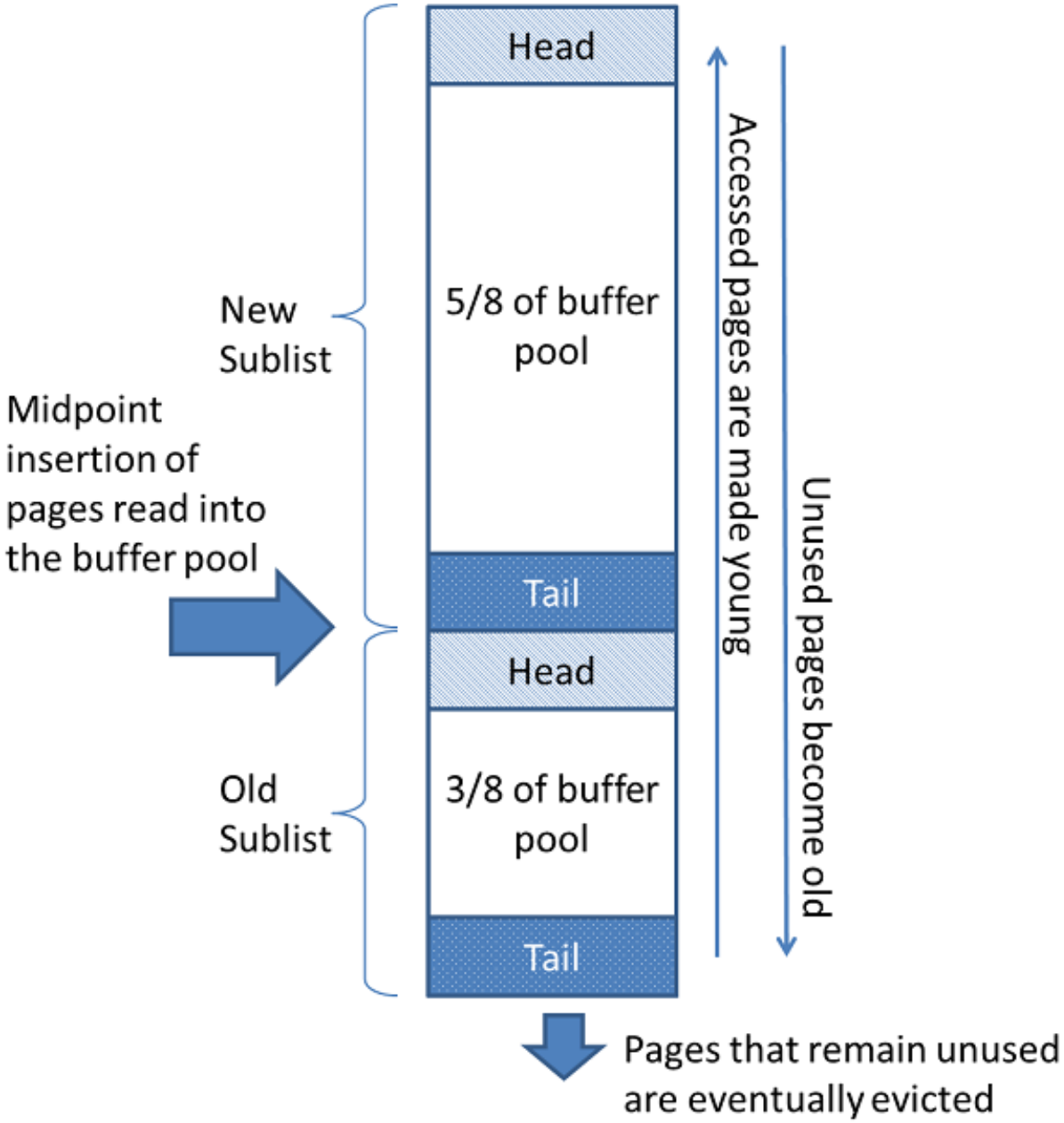
Knowing how to take advantage of the buffer pool to keep frequently accessed data in memory is an important aspect of MySQL tuning.

Buffer Pool LRU Algorithm

The buffer pool is managed as a list using a variation of the least recently used (LRU) algorithm. When room is needed to add a new page to the buffer pool, the least recently used page is evicted and a new page is added to the middle of the list. This midpoint insertion strategy treats the list as two sublists:

- At the head, a sublist of new (“young”) pages that were accessed recently
- At the tail, a sublist of old pages that were accessed less recently

Figure 15.2 Buffer Pool List



The algorithm keeps heavily pages in the new sublist. The old sublist contains less-used pages; these pages are candidates for eviction.

By default, the algorithm operates as follows:

- 3/8 of the buffer pool is devoted to the old sublist.
- The midpoint of the list is the boundary where the tail of the new sublist meets the head of the old sublist.

- When InnoDB reads a page into the buffer pool, it initially inserts it at the midpoint (the head of the old sublist). A page can be read because it is required for a user-initiated operation such as an SQL query, or as part of a read-ahead operation performed automatically by InnoDB.
- Accessing a page in the old sublist makes it “young”, moving it to the head of the new sublist. If the page was read because it was required by a user-initiated operation, the first access occurs immediately and the page is made young. If the page was read due to a read-ahead operation, the first access does not occur immediately, and might not occur at all before the page is evicted.
- As the database operates, pages in the buffer pool that are not accessed “age” by moving toward the tail of the list. Pages in both the new and old sublists age as other pages are made new. Pages in the old sublist also age as pages are inserted at the midpoint. Eventually, a page that remains unused reaches the tail of the old sublist and is evicted.

By default, pages read by queries are immediately moved into the new sublist, meaning they stay in the buffer pool longer. A table scan, performed for a mysqldump operation or a SELECT statement with no WHERE clause, for example, can bring a large amount of data into the buffer pool and evict an equivalent amount of older data, even if the new data is never used again. Similarly, pages that are loaded by the read-ahead background thread and accessed only once are moved to the head of the new list. These situations can push frequently used pages to the old sublist where they become subject to eviction. For information about optimizing this behavior, see Section 15.8.3.3, “Making the Buffer Pool Scan Resistant”, and Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”.

InnoDB Standard Monitor output contains several fields in the BUFFER POOL AND MEMORY section regarding operation of the buffer pool LRU algorithm. For details, see Monitoring the Buffer Pool Using the InnoDB Standard Monitor.

Buffer Pool Configuration

You can configure the various aspects of the buffer pool to improve performance.

- Ideally, you set the size of the buffer pool to as large a value as practical, leaving enough memory for other processes on the server to run without excessive paging. The larger the buffer pool, the more InnoDB acts like an in-memory database, reading data from disk once and then accessing the data from memory during subsequent reads. See Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”.
- On 64-bit systems with sufficient memory, you can split the buffer pool into multiple parts to minimize contention for memory structures among concurrent operations. For details, see Section 15.8.3.2, “Configuring Multiple Buffer Pool Instances”.
- You can keep frequently accessed data in memory regardless of sudden spikes of activity from operations that would bring large amounts of infrequently accessed data into the buffer pool. For details, see Section 15.8.3.3, “Making the Buffer Pool Scan Resistant”.
- You can control when and how to perform read-ahead requests to prefetch pages into the buffer pool asynchronously in anticipation that the pages will be needed soon. For details, see Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”.
- You can control when background flushing occurs and whether or not the rate of flushing is dynamically adjusted based on workload. For details, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”.
- You can configure how InnoDB preserves the current buffer pool state to avoid a lengthy warmup period after a server restart. For details, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

Monitoring the Buffer Pool Using the InnoDB Standard Monitor

InnoDB Standard Monitor output, which can be accessed using SHOW ENGINE INNODB STATUS, provides metrics regarding operation of the buffer pool. Buffer pool metrics are located in the BUFFER POOL AND MEMORY section of InnoDB Standard Monitor output and appear similar to the following:

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 2198863872
Dictionary memory allocated 776332
Buffer pool size   131072
Free buffers       124908
Database pages     5720
Old database pages 2071
Modified db pages  910
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 4, not young 0
0.10 youngs/s, 0.00 non-youngs/s
Pages read 197, created 5523, written 5060
0.00 reads/s, 190.89 creates/s, 244.94 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not
0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read
ahead 0.00/s
LRU len: 5720, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
```

The following table describes buffer pool metrics reported by the InnoDB Standard Monitor.

Note

Per second averages provided in InnoDB Standard Monitor output are based on the elapsed time since InnoDB Standard Monitor output was last printed.

Table 15.2 InnoDB Buffer Pool Metrics

Name	Description
Total memory allocated	The total memory allocated for the buffer pool in bytes.
Dictionary memory allocated	The total memory allocated for the InnoDB data dictionary in bytes.
Buffer pool size	The total size in pages allocated to the buffer pool.
Free buffers	The total size in pages of the buffer pool free list.
Database pages	The total size in pages of the buffer pool LRU list.
Old database pages	The total size in pages of the buffer pool old LRU sublist.
Modified db pages	The current number of pages modified in the buffer pool.
Pending reads	The number of buffer pool pages waiting to be read into the buffer pool.
Pending writes LRU	The number of old dirty pages within the buffer pool to be written from the bottom of the LRU list.
Pending writes flush list	The number of buffer pool pages to be flushed during checkpointing.
Pending writes single page	The number of pending independent page writes within the buffer pool.
Pages made young	The total number of pages made young in the buffer pool LRU list (moved to the head of sublist of “new” pages).

Name	Description
Pages made not young	The total number of pages not made young in the buffer pool LRU list (pages that have remained in the “old” sublist without being made young).
youngs/s	The per second average of accesses to old pages in the buffer pool LRU list that have resulted in making pages young. See the notes that follow this table for more information.
non-youngs/s	The per second average of accesses to old pages in the buffer pool LRU list that have resulted in not making pages young. See the notes that follow this table for more information.
Pages read	The total number of pages read from the buffer pool.
Pages created	The total number of pages created within the buffer pool.
Pages written	The total number of pages written from the buffer pool.
reads/s	The per second average number of buffer pool page reads per second.
creates/s	The per second average number of buffer pool pages created per second.
writes/s	The per second average number of buffer pool page writes per second.
Buffer pool hit rate	The buffer pool page hit rate for pages read from the buffer pool memory vs from disk storage.
young-making rate	The average hit rate at which page accesses have resulted in making pages young. See the notes that follow this table for more information.
not (young-making rate)	The average hit rate at which page accesses have not resulted in making pages young. See the notes that follow this table for more information.
Pages read ahead	The per second average of read ahead operations.
Pages evicted without access	The per second average of the pages evicted without being accessed from the buffer pool.
Random read ahead	The per second average of random read ahead operations.
LRU len	The total size in pages of the buffer pool LRU list.
unzip_LRU len	The total size in pages of the buffer pool unzip_LRU list.
I/O sum	The total number of buffer pool LRU list pages accessed, for the last 50 seconds.
I/O cur	The total number of buffer pool LRU list pages accessed.
I/O unzip sum	The total number of buffer pool unzip_LRU list pages accessed.
I/O unzip cur	The total number of buffer pool unzip_LRU list pages accessed.

Notes:

- The `youngs/s` metric is applicable only to old pages. It is based on the number of accesses to pages and not the number of pages. There can be multiple accesses to a given page, all of which are counted. If you see very low `youngs/s` values when there are no large scans occurring, you might need to reduce the delay time or increase the percentage of the buffer pool used for the old sublist. Increasing

the percentage makes the old sublist larger, so pages in that sublist take longer to move to the tail, which increases the likelihood that those pages will be accessed again and made young.

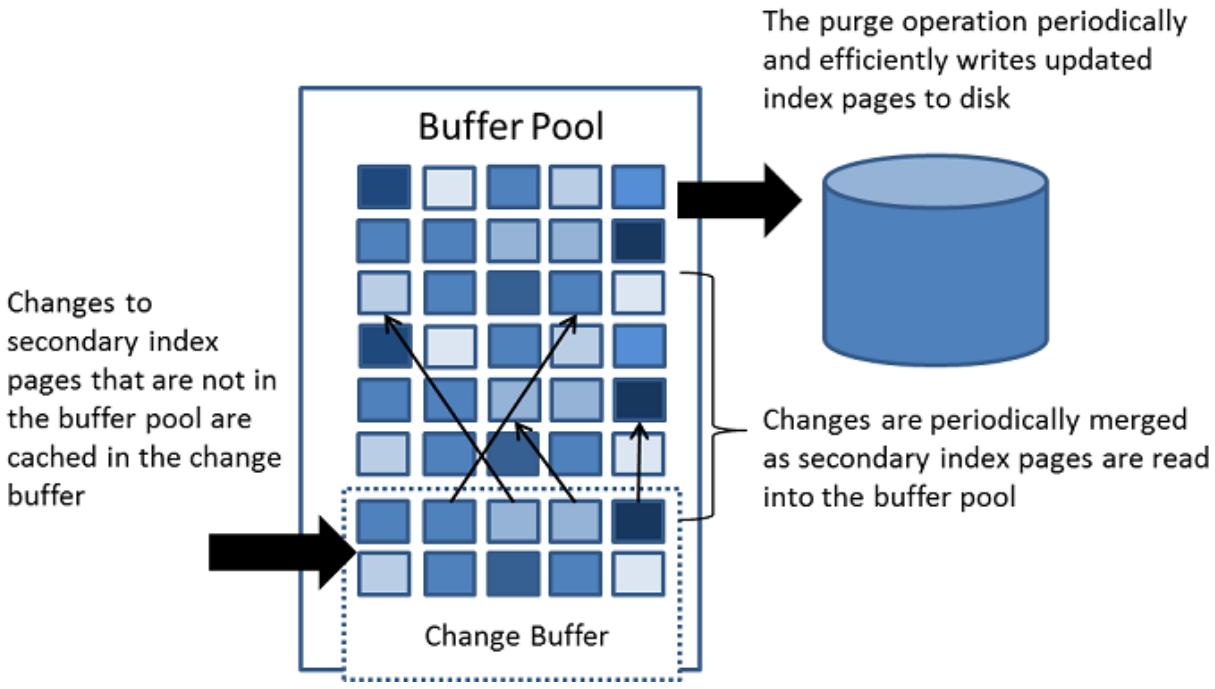
- The `non-youngs/s` metric is applicable only to old pages. It is based on the number of accesses to pages and not the number of pages. There can be multiple accesses to a given page, all of which are counted. If you do not see a higher `non-youngs/s` value when performing large table scans (and a higher `youngs/s` value), increase the delay value.
- The `young-making` rate accounts for accesses to all buffer pool pages, not just accesses to pages in the old sublist. The `young-making` rate and `not` rate do not normally add up to the overall buffer pool hit rate. Page hits in the old sublist cause pages to move to the new sublist, but page hits in the new sublist cause pages to move to the head of the list only if they are a certain distance from the head.
- `not (young-making rate)` is the average hit rate at which page accesses have not resulted in making pages young due to the delay defined by `innodb_old_blocks_time` not being met, or due to page hits in the new sublist that did not result in pages being moved to the head. This rate accounts for accesses to all buffer pool pages, not just accesses to pages in the old sublist.

Buffer pool server status variables and the `INNODB_BUFFER_POOL_STATS` table provide many of the same buffer pool metrics found in `InnoDB` Standard Monitor output. For more information, see Example 15.10, “Querying the INNODB_BUFFER_POOL_STATS Table”.

15.5.2 Change Buffer

The change buffer is a special data structure that caches changes to secondary index pages when those pages are not in the buffer pool. The buffered changes, which may result from `INSERT`, `UPDATE`, or `DELETE` operations (DML), are merged later when the pages are loaded into the buffer pool by other read operations.

Figure 15.3 Change Buffer



Unlike clustered indexes, secondary indexes are usually nonunique, and inserts into secondary indexes happen in a relatively random order. Similarly, deletes and updates may affect secondary index pages

that are not adjacently located in an index tree. Merging cached changes at a later time, when affected pages are read into the buffer pool by other operations, avoids substantial random access I/O that would be required to read secondary index pages into the buffer pool from disk.

Periodically, the purge operation that runs when the system is mostly idle, or during a slow shutdown, writes the updated index pages to disk. The purge operation can write disk blocks for a series of index values more efficiently than if each value were written to disk immediately.

Change buffer merging may take several hours when there are many affected rows and numerous secondary indexes to update. During this time, disk I/O is increased, which can cause a significant slowdown for disk-bound queries. Change buffer merging may also continue to occur after a transaction is committed, and even after a server shutdown and restart (see Section 15.20.2, “Forcing InnoDB Recovery” for more information).

In memory, the change buffer occupies part of the buffer pool. On disk, the change buffer is part of the system tablespace, where index changes are buffered when the database server is shut down.

The type of data cached in the change buffer is governed by the `innodb_change_buffering` variable. For more information, see Configuring Change Buffering. You can also configure the maximum change buffer size. For more information, see Configuring the Change Buffer Maximum Size.

Change buffering is not supported for a secondary index if the index contains a descending index column or if the primary key includes a descending index column.

For answers to frequently asked questions about the change buffer, see Section A.16, “MySQL 8.0 FAQ: InnoDB Change Buffer”.

Configuring Change Buffering

When `INSERT`, `UPDATE`, and `DELETE` operations are performed on a table, the values of indexed columns (particularly the values of secondary keys) are often in an unsorted order, requiring substantial I/O to bring secondary indexes up to date. The change buffer caches changes to secondary index entries when the relevant page is not in the buffer pool, thus avoiding expensive I/O operations by not immediately reading in the page from disk. The buffered changes are merged when the page is loaded into the buffer pool, and the updated page is later flushed to disk. The `InnoDB` main thread merges buffered changes when the server is nearly idle, and during a slow shutdown.

Because it can result in fewer disk reads and writes, the change buffer feature is most valuable for workloads that are I/O-bound, for example applications with a high volume of DML operations such as bulk inserts.

However, the change buffer occupies a part of the buffer pool, reducing the memory available to cache data pages. If the working set almost fits in the buffer pool, or if your tables have relatively few secondary indexes, it may be useful to disable change buffering. If the working data set fits entirely within the buffer pool, change buffering does not impose extra overhead, because it only applies to pages that are not in the buffer pool.

You can control the extent to which `InnoDB` performs change buffering using the `innodb_change_buffering` configuration parameter. You can enable or disable buffering for inserts, delete operations (when index records are initially marked for deletion) and purge operations (when index records are physically deleted). An update operation is a combination of an insert and a delete. The default `innodb_change_buffering` value is `all`.

Permitted `innodb_change_buffering` values include:

- **`all`**

The default value: buffer inserts, delete-marking operations, and purges.

- **none**

  Do not buffer any operations.

- **inserts**

  Buffer insert operations.

- **deletes**

  Buffer delete-marking operations.

- **changes**

  Buffer both inserts and delete-marking operations.

- **purges**

  Buffer the physical deletion operations that happen in the background.

You can set the innodb_change_buffering parameter in the MySQL option file (my.cnf or my.ini) or change it dynamically with the SET GLOBAL statement, which requires privileges sufficient to set global system variables. See Section 5.1.9.1, “System Variable Privileges”. Changing the setting affects the buffering of new operations; the merging of existing buffered entries is not affected.

Configuring the Change Buffer Maximum Size

The innodb_change_buffer_max_size variable permits configuring the maximum size of the change buffer as a percentage of the total size of the buffer pool. By default, innodb_change_buffer_max_size is set to 25. The maximum setting is 50.

Consider increasing innodb_change_buffer_max_size on a MySQL server with heavy insert, update, and delete activity, where change buffer merging does not keep pace with new change buffer entries, causing the change buffer to reach its maximum size limit.

Consider decreasing innodb_change_buffer_max_size on a MySQL server with static data used for reporting, or if the change buffer consumes too much of the memory space shared with the buffer pool, causing pages to age out of the buffer pool sooner than desired.

Test different settings with a representative workload to determine an optimal configuration. The innodb_change_buffer_max_size setting is dynamic, which permits modifying the setting without restarting the server.

Monitoring the Change Buffer

The following options are available for change buffer monitoring:

- InnoDB Standard Monitor output includes change buffer status information. To view monitor data, issue the SHOW ENGINE INNODB STATUS statement.

```
mysql> SHOW ENGINE INNODB STATUS\G
```

  Change buffer status information is located under the INSERT BUFFER AND ADAPTIVE HASH INDEX heading and appears similar to the following:

```
```

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2, 0 merges
merged operations:
 insert 0, delete mark 0, delete 0
discarded operations:
 insert 0, delete mark 0, delete 0
Hash table size 4425293, used cells 32, node heap has 1 buffer(s)
13577.57 hash searches/s, 202.47 non-hash searches/s
```

For more information, see Section 15.16.3, “InnoDB Standard Monitor and Lock Monitor Output”.

- The `INFORMATION_SCHEMA.INNODB_METRICS` table provides most of the data points found in `InnoDB` Standard Monitor output, plus other data points. To view change buffer metrics and a description of each, issue the following query:

```
mysql> SELECT NAME, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME LIKE '%ibuf%'\G
```

For `INNODB_METRICS` table usage information, see Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

- The `INFORMATION_SCHEMA.INNODB_BUFFER_PAGE` table provides metadata about each page in the buffer pool, including change buffer index and change buffer bitmap pages. Change buffer pages are identified by `PAGE_TYPE`. `IBUF_INDEX` is the page type for change buffer index pages, and `IBUF_BITMAP` is the page type for change buffer bitmap pages.

Warning

Querying the `INNODB_BUFFER_PAGE` table can introduce significant performance overhead. To avoid impacting performance, reproduce the issue you want to investigate on a test instance and run your queries on the test instance.

For example, you can query the `INNODB_BUFFER_PAGE` table to determine the approximate number of `IBUF_INDEX` and `IBUF_BITMAP` pages as a percentage of total buffer pool pages.

```
mysql> SELECT (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE PAGE_TYPE LIKE 'IBUF%') AS change_buffer_pages,
       (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE) AS total_pages,
       (SELECT ((change_buffer_pages/total_pages)*100))
       AS change_buffer_page_percentage;
+---------------------+-------------+-------------------------------+
| change_buffer_pages | total_pages | change_buffer_page_percentage |
+---------------------+-------------+-------------------------------+
|                  25 |        8192 |                        0.3052 |
+---------------------+-------------+-------------------------------+
```

For information about other data provided by the `INNODB_BUFFER_PAGE` table, see Section 25.39.1, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table”. For related usage information, see Section 15.14.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”.

- Performance Schema provides change buffer mutex wait instrumentation for advanced performance monitoring. To view change buffer instrumentation, issue the following query:

```
mysql> SELECT * FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%wait/synch/mutex/innodb/ibuf%';
+-----------------------------------------------------------+---------+-------+
| NAME                                                      | ENABLED | TIMED |
+-----------------------------------------------------------+---------+-------+
```

```
| wait/synch/mutex/innodb/ibuf_bitmap_mutex             | YES     | YES   |
| wait/synch/mutex/innodb/ibuf_mutex                    | YES     | YES   |
| wait/synch/mutex/innodb/ibuf_pessimistic_insert_mutex | YES     | YES   |
+-------------------------------------------------------+---------+-------+
```

For information about monitoring `InnoDB` mutex waits, see Section 15.15.2, “Monitoring InnoDB Mutex Waits Using Performance Schema”.

15.5.3 Adaptive Hash Index

The adaptive hash index feature enables `InnoDB` to perform more like an in-memory database on systems with appropriate combinations of workload and sufficient memory for the buffer pool without sacrificing transactional features or reliability. The adaptive hash index feature is enabled by the `innodb_adaptive_hash_index` variable, or turned off at server startup by `--skip-innodb-adaptive-hash-index`.

Based on the observed pattern of searches, a hash index is built using a prefix of the index key. The prefix can be any length, and it may be that only some values in the B-tree appear in the hash index. Hash indexes are built on demand for the pages of the index that are accessed often.

If a table fits almost entirely in main memory, a hash index can speed up queries by enabling direct lookup of any element, turning the index value into a sort of pointer. `InnoDB` has a mechanism that monitors index searches. If `InnoDB` notices that queries could benefit from building a hash index, it does so automatically.

With some workloads, the speedup from hash index lookups greatly outweighs the extra work to monitor index lookups and maintain the hash index structure. Access to the adaptive hash index can sometimes become a source of contention under heavy workloads, such as multiple concurrent joins. Queries with `LIKE` operators and `%` wildcards also tend not to benefit. For workloads that do not benefit from the adaptive hash index feature, turning it off reduces unnecessary performance overhead. Because it is difficult to predict in advance whether the adaptive hash index feature is appropriate for a particular system and workload, consider running benchmarks with it enabled and disabled. Architectural changes in MySQL 5.6 make it more suitable to disable the adaptive hash index feature than in earlier releases.

The adaptive hash index feature is partitioned. Each index is bound to a specific partition, and each partition is protected by a separate latch. Partitioning is controlled by the `innodb_adaptive_hash_index_parts` variable. The `innodb_adaptive_hash_index_parts` variable is set to 8 by default. The maximum setting is 512.

You can monitor adaptive hash index use and contention in the `SEMAPHORES` section of `SHOW ENGINE INNODB STATUS` output. If there are numerous threads waiting on RW-latches created in `btr0sea.c`, consider increasing the number of adaptive hash index partitions or disabling the adaptive hash index feature.

For information about the performance characteristics of hash indexes, see Section 8.3.9, “Comparison of B-Tree and Hash Indexes”.

15.5.4 Log Buffer

The log buffer is the memory area that holds data to be written to the log files on disk. Log buffer size is defined by the `innodb_log_buffer_size` variable. The default size is 16MB. The contents of the log buffer are periodically flushed to disk. A large log buffer enables large transactions to run without the need to write redo log data to disk before the transactions commit. Thus, if you have transactions that update, insert, or delete many rows, increasing the size of the log buffer saves disk I/O.

The `innodb_flush_log_at_trx_commit` variable controls how the contents of the log buffer are written and flushed to disk. The `innodb_flush_log_at_timeout` variable controls log flushing frequency.

For related information, see Memory Configuration, and Section 8.5.4, “Optimizing InnoDB Redo Logging”.

15.6 InnoDB On-Disk Structures

This section describes InnoDB on-disk structures and related topics.

15.6.1 Tables

This section covers topics related to InnoDB tables.

15.6.1.1 Creating InnoDB Tables

To create an InnoDB table, use the CREATE TABLE statement.

```
CREATE TABLE t1 (a INT, b CHAR (20), PRIMARY KEY (a)) ENGINE=InnoDB;
```

You do not need to specify the ENGINE=InnoDB clause if InnoDB is defined as the default storage engine, which it is by default. To check the default storage engine, issue the following statement:

```
mysql> SELECT @@default_storage_engine;
+--------------------------+
| @@default_storage_engine |
+--------------------------+
| InnoDB                   |
+--------------------------+
```

You might still use ENGINE=InnoDB clause if you plan to use mysqldump or replication to replay the CREATE TABLE statement on a server where the default storage engine is not InnoDB.

An InnoDB table and its indexes can be created in the system tablespace, in a file-per-table tablespace, or in a general tablespace. When innodb_file_per_table is enabled, which is the default, an InnoDB table is implicitly created in an individual file-per-table tablespace. Conversely, when innodb_file_per_table is disabled, an InnoDB table is implicitly created in the InnoDB system tablespace. To create a table in a general tablespace, use CREATE TABLE ... TABLESPACE syntax. For more information, see Section 15.6.3.3, “General Tablespaces”.

When you create a table in a file-per-table tablespace, MySQL creates an .ibd tablespace file in a database directory under the MySQL data directory, by default. A table created in the InnoDB system tablespace is created in an existing ibdata file, which resides in the MySQL data directory. A table created in a general tablespace is created in an existing general tablespace .ibd file. General tablespace files can be created inside or outside of the MySQL data directory. For more information, see Section 15.6.3.3, “General Tablespaces”.

Internally, InnoDB adds an entry for each table to the data dictionary. The entry includes the database name. For example, if table t1 is created in the test database, the data dictionary entry for the database name is 'test/t1'. This means you can create a table of the same name (t1) in a different database, and the table names do not collide inside InnoDB.

InnoDB Tables and Row Formats

The default row format for InnoDB tables is defined by the innodb_default_row_format configuration option, which has a default value of DYNAMIC. Dynamic and Compressed row format allow you to take advantage of InnoDB features such as table compression and efficient off-page storage of long column values. To use these row formats, innodb_file_per_table must be enabled (the default).

```
SET GLOBAL innodb_file_per_table=1;
```

```
CREATE TABLE t3 (a INT, b CHAR (20), PRIMARY KEY (a)) ROW_FORMAT=DYNAMIC;
CREATE TABLE t4 (a INT, b CHAR (20), PRIMARY KEY (a)) ROW_FORMAT=COMPRESSED;
```

Alternatively, you can use `CREATE TABLE ... TABLESPACE` syntax to create an `InnoDB` table in a general tablespace. General tablespaces support all row formats. For more information, see Section 15.6.3.3, “General Tablespaces”.

```
CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=DYNAMIC;
```

`CREATE TABLE ... TABLESPACE` syntax can also be used to create `InnoDB` tables with a `Dynamic` row format in the system tablespace, alongside tables with a `Compact` or `Redundant` row format.

```
CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE = innodb_system ROW_FORMAT=DYNAMIC;
```

For more information about `InnoDB` row formats, see Section 15.10, “InnoDB Row Formats”. For how to determine the row format of an `InnoDB` table and the physical characteristics of `InnoDB` row formats, see Section 15.10, “InnoDB Row Formats”.

InnoDB Tables and Primary Keys

Always define a primary key for an `InnoDB` table, specifying the column or columns that:

- Are referenced by the most important queries.
- Are never left blank.
- Never have duplicate values.
- Rarely if ever change value once inserted.

For example, in a table containing information about people, you would not create a primary key on `(firstname, lastname)` because more than one person can have the same name, some people have blank last names, and sometimes people change their names. With so many constraints, often there is not an obvious set of columns to use as a primary key, so you create a new column with a numeric ID to serve as all or part of the primary key. You can declare an auto-increment column so that ascending values are filled in automatically as rows are inserted:

```
# The value of ID can act like a pointer between related items in different tables.
CREATE TABLE t5 (id INT AUTO_INCREMENT, b CHAR (20), PRIMARY KEY (id));

# The primary key can consist of more than one column. Any autoinc column must come first.
CREATE TABLE t6 (id INT AUTO_INCREMENT, a INT, b CHAR (20), PRIMARY KEY (id,a));
```

Although the table works correctly without defining a primary key, the primary key is involved with many aspects of performance and is a crucial design aspect for any large or frequently used table. It is recommended that you always specify a primary key in the `CREATE TABLE` statement. If you create the table, load data, and then run `ALTER TABLE` to add a primary key later, that operation is much slower than defining the primary key when creating the table.

Viewing InnoDB Table Properties

To view the properties of an `InnoDB` table, issue a `SHOW TABLE STATUS` statement:

```
mysql> SHOW TABLE STATUS FROM test LIKE 't%' \G;
*************************** 1. row ***************************
           Name: t1
         Engine: InnoDB
        Version: 10
```

```
     Row_format: Compact
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2015-03-16 15:13:31
    Update_time: NULL
     Check_time: NULL
      Collation: utf8mb4_0900_ai_ci
       Checksum: NULL
 Create_options:
        Comment:
```

For information about `SHOW TABLE STATUS` output, see Section 13.7.7.36, “SHOW TABLE STATUS Syntax”.

`InnoDB` table properties may also be queried using the `InnoDB` Information Schema system tables:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME='test/t1' \G
*************************** 1. row ***************************
     TABLE_ID: 45
         NAME: test/t1
         FLAG: 1
       N_COLS: 5
        SPACE: 35
   ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
   SPACE_TYPE: Single
```

For more information, see Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”.

15.6.1.2 Moving or Copying InnoDB Tables

This section describes techniques for moving or copying some or all `InnoDB` tables to a different server or instance. For example, you might move an entire MySQL instance to a larger, faster server; you might clone an entire MySQL instance to a new replication slave server; you might copy individual tables to another instance to develop and test an application, or to a data warehouse server to produce reports.

On Windows, `InnoDB` always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, create all databases and tables using lowercase names. A convenient way to accomplish this is to add the following line to the `[mysqld]` section of your `my.cnf` or `my.ini` file before creating any databases or tables:

```
[mysqld]
lower_case_table_names=1
```

Note

It is prohibited to start the server with a `lower_case_table_names` setting that is different from the setting used when the server was initialized.

Techniques for moving or copying `InnoDB` tables include:

- Transportable Tablespaces
- MySQL Enterprise Backup
- Copying Data Files (Cold Backup Method)

- Export and Import (mysqldump)

Transportable Tablespaces

The transportable tablespaces feature uses `FLUSH TABLES ... FOR EXPORT` to ready `InnoDB` tables for copying from one server instance to another. To use this feature, `InnoDB` tables must be created with `innodb_file_per_table` set to `ON` so that each `InnoDB` table has its own tablespace. For usage information, see Section 15.6.3.7, “Copying Tablespaces to Another Instance”.

MySQL Enterprise Backup

The MySQL Enterprise Backup product lets you back up a running MySQL database with minimal disruption to operations while producing a consistent snapshot of the database. When MySQL Enterprise Backup is copying tables, reads and writes can continue. In addition, MySQL Enterprise Backup can create compressed backup files, and back up subsets of tables. In conjunction with the MySQL binary log, you can perform point-in-time recovery. MySQL Enterprise Backup is included as part of the MySQL Enterprise subscription.

For more details about MySQL Enterprise Backup, see Section 30.2, “MySQL Enterprise Backup Overview”.

Copying Data Files (Cold Backup Method)

You can move an `InnoDB` database simply by copying all the relevant files listed under "Cold Backups" in Section 15.17.1, “InnoDB Backup”.

`InnoDB` data and log files are binary-compatible on all platforms having the same floating-point number format. If the floating-point formats differ but you have not used `FLOAT` or `DOUBLE` data types in your tables, then the procedure is the same: simply copy the relevant files.

When you move or copy file-per-table `.ibd` files, the database directory name must be the same on the source and destination systems. The table definition stored in the `InnoDB` shared tablespace includes the database name. The transaction IDs and log sequence numbers stored in the tablespace files also differ between databases.

To move an `.ibd` file and the associated table from one database to another, use a `RENAME TABLE` statement:

```
RENAME TABLE db1.tbl_name TO db2.tbl_name;
```

If you have a “clean” backup of an `.ibd` file, you can restore it to the MySQL installation from which it originated as follows:

1. The table must not have been dropped or truncated since you copied the `.ibd` file, because doing so changes the table ID stored inside the tablespace.
2. Issue this `ALTER TABLE` statement to delete the current `.ibd` file:

   ```
   ALTER TABLE tbl_name DISCARD TABLESPACE;
   ```

3. Copy the backup `.ibd` file to the proper database directory.
4. Issue this `ALTER TABLE` statement to tell `InnoDB` to use the new `.ibd` file for the table:

   ```
   ALTER TABLE tbl_name IMPORT TABLESPACE;
   ```

Note

The `ALTER TABLE ... IMPORT TABLESPACE` feature does not enforce foreign key constraints on imported data.

In this context, a “clean” `.ibd` file backup is one for which the following requirements are satisfied:

- There are no uncommitted modifications by transactions in the `.ibd` file.
- There are no unmerged insert buffer entries in the `.ibd` file.
- Purge has removed all delete-marked index records from the `.ibd` file.
- `mysqld` has flushed all modified pages of the `.ibd` file from the buffer pool to the file.

You can make a clean backup `.ibd` file using the following method:

1. Stop all activity from the `mysqld` server and commit all transactions.
2. Wait until `SHOW ENGINE INNODB STATUS` shows that there are no active transactions in the database, and the main thread status of `InnoDB` is `Waiting for server activity`. Then you can make a copy of the `.ibd` file.

Another method for making a clean copy of an `.ibd` file is to use the MySQL Enterprise Backup product:

1. Use MySQL Enterprise Backup to back up the `InnoDB` installation.
2. Start a second `mysqld` server on the backup and let it clean up the `.ibd` files in the backup.

Export and Import (mysqldump)

You can use `mysqldump` to dump your tables on one machine and then import the dump files on the other machine. Using this method, it does not matter whether the formats differ or if your tables contain floating-point data.

One way to increase the performance of this method is to switch off autocommit mode when importing data, assuming that the tablespace has enough space for the big rollback segment that the import transactions generate. Do the commit only after importing a whole table or a segment of a table.

15.6.1.3 Converting Tables from MyISAM to InnoDB

If you have `MyISAM` tables that you want to convert to `InnoDB` for better reliability and scalability, review the following guidelines and tips before converting.

Note

Partitioned `MyISAM` tables created in previous versions of MySQL are not compatible with MySQL 8.0. Such tables must be prepared prior to upgrade, either by removing the partitioning, or by converting them to `InnoDB`. See Section 23.6.2, “Partitioning Limitations Relating to Storage Engines”, for more information.

- Adjusting Memory Usage for MyISAM and InnoDB
- Handling Too-Long Or Too-Short Transactions
- Handling Deadlocks
- Planning the Storage Layout
- Converting an Existing Table

- Cloning the Structure of a Table
- Transferring Existing Data
- Storage Requirements
- Defining a PRIMARY KEY for Each Table
- Application Performance Considerations
- Understanding Files Associated with InnoDB Tables

Adjusting Memory Usage for MyISAM and InnoDB

As you transition away from `MyISAM` tables, lower the value of the `key_buffer_size` configuration option to free memory no longer needed for caching results. Increase the value of the `innodb_buffer_pool_size` configuration option, which performs a similar role of allocating cache memory for `InnoDB` tables. The `InnoDB` buffer pool caches both table data and index data, speeding up lookups for queries and keeping query results in memory for reuse. For guidance regarding buffer pool size configuration, see Section 8.12.3.1, “How MySQL Uses Memory”.

Handling Too-Long Or Too-Short Transactions

Because `MyISAM` tables do not support transactions, you might not have paid much attention to the `autocommit` configuration option and the `COMMIT` and `ROLLBACK` statements. These keywords are important to allow multiple sessions to read and write `InnoDB` tables concurrently, providing substantial scalability benefits in write-heavy workloads.

While a transaction is open, the system keeps a snapshot of the data as seen at the beginning of the transaction, which can cause substantial overhead if the system inserts, updates, and deletes millions of rows while a stray transaction keeps running. Thus, take care to avoid transactions that run for too long:

- If you are using a `mysql` session for interactive experiments, always `COMMIT` (to finalize the changes) or `ROLLBACK` (to undo the changes) when finished. Close down interactive sessions rather than leave them open for long periods, to avoid keeping transactions open for long periods by accident.
- Make sure that any error handlers in your application also `ROLLBACK` incomplete changes or `COMMIT` completed changes.
- `ROLLBACK` is a relatively expensive operation, because `INSERT`, `UPDATE`, and `DELETE` operations are written to `InnoDB` tables prior to the `COMMIT`, with the expectation that most changes are committed successfully and rollbacks are rare. When experimenting with large volumes of data, avoid making changes to large numbers of rows and then rolling back those changes.
- When loading large volumes of data with a sequence of `INSERT` statements, periodically `COMMIT` the results to avoid having transactions that last for hours. In typical load operations for data warehousing, if something goes wrong, you truncate the table (using `TRUNCATE TABLE`) and start over from the beginning rather than doing a `ROLLBACK`.

The preceding tips save memory and disk space that can be wasted during too-long transactions. When transactions are shorter than they should be, the problem is excessive I/O. With each `COMMIT`, MySQL makes sure each change is safely recorded to disk, which involves some I/O.

- For most operations on `InnoDB` tables, you should use the setting `autocommit=0`. From an efficiency perspective, this avoids unnecessary I/O when you issue large numbers of consecutive `INSERT`, `UPDATE`, or `DELETE` statements. From a safety perspective, this allows you to issue a `ROLLBACK` statement to recover lost or garbled data if you make a mistake on the `mysql` command line, or in an exception handler in your application.

- The time when `autocommit=1` is suitable for `InnoDB` tables is when running a sequence of queries for generating reports or analyzing statistics. In this situation, there is no I/O penalty related to `COMMIT` or `ROLLBACK`, and `InnoDB` can automatically optimize the read-only workload.
- If you make a series of related changes, finalize all the changes at once with a single `COMMIT` at the end. For example, if you insert related pieces of information into several tables, do a single `COMMIT` after making all the changes. Or if you run many consecutive `INSERT` statements, do a single `COMMIT` after all the data is loaded; if you are doing millions of `INSERT` statements, perhaps split up the huge transaction by issuing a `COMMIT` every ten thousand or hundred thousand records, so the transaction does not grow too large.
- Remember that even a `SELECT` statement opens a transaction, so after running some report or debugging queries in an interactive `mysql` session, either issue a `COMMIT` or close the `mysql` session.

Handling Deadlocks

You might see warning messages referring to “deadlocks” in the MySQL error log, or the output of `SHOW ENGINE INNODB STATUS`. Despite the scary-sounding name, a deadlock is not a serious issue for `InnoDB` tables, and often does not require any corrective action. When two transactions start modifying multiple tables, accessing the tables in a different order, they can reach a state where each transaction is waiting for the other and neither can proceed. When deadlock detection is enabled (the default), MySQL immediately detects this condition and cancels (rolls back) the “smaller” transaction, allowing the other to proceed. If deadlock detection is disabled using the `innodb_deadlock_detect` configuration option, `InnoDB` relies on the `innodb_lock_wait_timeout` setting to roll back transactions in case of a deadlock.

Either way, your applications need error-handling logic to restart a transaction that is forcibly cancelled due to a deadlock. When you re-issue the same SQL statements as before, the original timing issue no longer applies. Either the other transaction has already finished and yours can proceed, or the other transaction is still in progress and your transaction waits until it finishes.

If deadlock warnings occur constantly, you might review the application code to reorder the SQL operations in a consistent way, or to shorten the transactions. You can test with the `innodb_print_all_deadlocks` option enabled to see all deadlock warnings in the MySQL error log, rather than only the last warning in the `SHOW ENGINE INNODB STATUS` output.

For more information, see Section 15.7.5, “Deadlocks in InnoDB”.

Planning the Storage Layout

To get the best performance from `InnoDB` tables, you can adjust a number of parameters related to storage layout.

When you convert `MyISAM` tables that are large, frequently accessed, and hold vital data, investigate and consider the `innodb_file_per_table` and `innodb_page_size` configuration options, and the `ROW_FORMAT` and `KEY_BLOCK_SIZE` clauses of the `CREATE TABLE` statement.

During your initial experiments, the most important setting is `innodb_file_per_table`. When this setting is enabled, which is the default, new `InnoDB` tables are implicitly created in file-per-table tablespaces. In contrast with the `InnoDB` system tablespace, file-per-table tablespaces allow disk space to be reclaimed by the operating system when a table is truncated or dropped. File-per-table tablespaces also support DYNAMIC and COMPRESSED row formats and associated features such as table compression, efficient off-page storage for long variable-length columns, and large index prefixes. For more information, see Section 15.6.3.2, “File-Per-Table Tablespaces”.

You can also store `InnoDB` tables in a shared general tablespace, which support multiple tables and all row formats. For more information, see Section 15.6.3.3, “General Tablespaces”.

Converting an Existing Table

To convert a non-InnoDB table to use InnoDB use ALTER TABLE:

```
ALTER TABLE table_name ENGINE=InnoDB;
```

Cloning the Structure of a Table

You might make an InnoDB table that is a clone of a MyISAM table, rather than using ALTER TABLE to perform conversion, to test the old and new table side-by-side before switching.

Create an empty InnoDB table with identical column and index definitions. Use SHOW CREATE TABLE *table_name*\G to see the full CREATE TABLE statement to use. Change the ENGINE clause to ENGINE=INNODB.

Transferring Existing Data

To transfer a large volume of data into an empty InnoDB table created as shown in the previous section, insert the rows with INSERT INTO *innodb_table* SELECT * FROM *myisam_table* ORDER BY *primary_key_columns*.

You can also create the indexes for the InnoDB table after inserting the data. Historically, creating new secondary indexes was a slow operation for InnoDB, but now you can create the indexes after the data is loaded with relatively little overhead from the index creation step.

If you have UNIQUE constraints on secondary keys, you can speed up a table import by turning off the uniqueness checks temporarily during the import operation:

```
SET unique_checks=0;
... import operation ...
SET unique_checks=1;
```

For big tables, this saves disk I/O because InnoDB can use its change buffer to write secondary index records as a batch. Be certain that the data contains no duplicate keys. unique_checks permits but does not require storage engines to ignore duplicate keys.

For better control over the insertion process, you can insert big tables in pieces:

```
INSERT INTO newtable SELECT * FROM oldtable
   WHERE yourkey > something AND yourkey <= somethingelse;
```

After all records are inserted, you can rename the tables.

During the conversion of big tables, increase the size of the InnoDB buffer pool to reduce disk I/O, to a maximum of 80% of physical memory. You can also increase the size of InnoDB log files.

Storage Requirements

If you intend to make several temporary copies of your data in InnoDB tables during the conversion process, it is recommended that you create the tables in file-per-table tablespaces so that you can reclaim the disk space when you drop the tables. When the innodb_file_per_table configuration option is enabled (the default), newly created InnoDB tables are implicitly created in file-per-table tablespaces.

Whether you convert the MyISAM table directly or create a cloned InnoDB table, make sure that you have sufficient disk space to hold both the old and new tables during the process. **InnoDB tables require more disk space than MyISAM tables.** If an ALTER TABLE operation runs out of space, it starts a rollback, and that can take hours if it is disk-bound. For inserts, InnoDB uses the insert buffer to merge secondary index

records to indexes in batches. That saves a lot of disk I/O. For rollback, no such mechanism is used, and the rollback can take 30 times longer than the insertion.

In the case of a runaway rollback, if you do not have valuable data in your database, it may be advisable to kill the database process rather than wait for millions of disk I/O operations to complete. For the complete procedure, see Section 15.20.2, “Forcing InnoDB Recovery”.

Defining a PRIMARY KEY for Each Table

The `PRIMARY KEY` clause is a critical factor affecting the performance of MySQL queries and the space usage for tables and indexes. The primary key uniquely identifies a row in a table. Every row in the table must have a primary key value, and no two rows can have the same primary key value.

These are guidelines for the primary key, followed by more detailed explanations.

- Declare a `PRIMARY KEY` for each table. Typically, it is the most important column that you refer to in `WHERE` clauses when looking up a single row.
- Declare the `PRIMARY KEY` clause in the original `CREATE TABLE` statement, rather than adding it later through an `ALTER TABLE` statement.
- Choose the column and its data type carefully. Prefer numeric columns over character or string ones.
- Consider using an auto-increment column if there is not another stable, unique, non-null, numeric column to use.
- An auto-increment column is also a good choice if there is any doubt whether the value of the primary key column could ever change. Changing the value of a primary key column is an expensive operation, possibly involving rearranging data within the table and within each secondary index.

Consider adding a primary key to any table that does not already have one. Use the smallest practical numeric type based on the maximum projected size of the table. This can make each row slightly more compact, which can yield substantial space savings for large tables. The space savings are multiplied if the table has any secondary indexes, because the primary key value is repeated in each secondary index entry. In addition to reducing data size on disk, a small primary key also lets more data fit into the buffer pool, speeding up all kinds of operations and improving concurrency.

If the table already has a primary key on some longer column, such as a `VARCHAR`, consider adding a new unsigned `AUTO_INCREMENT` column and switching the primary key to that, even if that column is not referenced in queries. This design change can produce substantial space savings in the secondary indexes. You can designate the former primary key columns as `UNIQUE NOT NULL` to enforce the same constraints as the `PRIMARY KEY` clause, that is, to prevent duplicate or null values across all those columns.

If you spread related information across multiple tables, typically each table uses the same column for its primary key. For example, a personnel database might have several tables, each with a primary key of employee number. A sales database might have some tables with a primary key of customer number, and other tables with a primary key of order number. Because lookups using the primary key are very fast, you can construct efficient join queries for such tables.

If you leave the `PRIMARY KEY` clause out entirely, MySQL creates an invisible one for you. It is a 6-byte value that might be longer than you need, thus wasting space. Because it is hidden, you cannot refer to it in queries.

Application Performance Considerations

The reliability and scalability features of `InnoDB` require more disk storage than equivalent `MyISAM` tables. You might change the column and index definitions slightly, for better space utilization, reduced I/O and

memory consumption when processing result sets, and better query optimization plans making efficient use of index lookups.

If you do set up a numeric ID column for the primary key, use that value to cross-reference with related values in any other tables, particularly for join queries. For example, rather than accepting a country name as input and doing queries searching for the same name, do one lookup to determine the country ID, then do other queries (or a single join query) to look up relevant information across several tables. Rather than storing a customer or catalog item number as a string of digits, potentially using up several bytes, convert it to a numeric ID for storing and querying. A 4-byte unsigned `INT` column can index over 4 billion items (with the US meaning of billion: 1000 million). For the ranges of the different integer types, see Section 11.2.1, "Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT".

Understanding Files Associated with InnoDB Tables

`InnoDB` files require more care and planning than `MyISAM` files do.

- You must not delete the ibdata files that represent the `InnoDB` system tablespace.
- Methods of moving or copying `InnoDB` tables to a different server are described in Section 15.6.1.2, "Moving or Copying InnoDB Tables".

15.6.1.4 AUTO_INCREMENT Handling in InnoDB

`InnoDB` provides a configurable locking mechanism that can significantly improve scalability and performance of SQL statements that add rows to tables with `AUTO_INCREMENT` columns. To use the `AUTO_INCREMENT` mechanism with an `InnoDB` table, an `AUTO_INCREMENT` column must be defined as part of an index such that it is possible to perform the equivalent of an indexed `SELECT MAX(ai_col)` lookup on the table to obtain the maximum column value. Typically, this is achieved by making the column the first column of some table index.

This section describes the behavior of `AUTO_INCREMENT` lock modes, usage implications for different `AUTO_INCREMENT` lock mode settings, and how `InnoDB` initializes the `AUTO_INCREMENT` counter.

- InnoDB AUTO_INCREMENT Lock Modes
- InnoDB AUTO_INCREMENT Lock Mode Usage Implications
- InnoDB AUTO_INCREMENT Counter Initialization

InnoDB AUTO_INCREMENT Lock Modes

This section describes the behavior of `AUTO_INCREMENT` lock modes used to generate auto-increment values, and how each lock mode affects replication. Auto-increment lock modes are configured at startup using the `innodb_autoinc_lock_mode` configuration parameter.

The following terms are used in describing `innodb_autoinc_lock_mode` settings:

- "`INSERT`-like" statements

  All statements that generate new rows in a table, including `INSERT`, `INSERT ... SELECT`, `REPLACE`, `REPLACE ... SELECT`, and `LOAD DATA`. Includes "simple-inserts", "bulk-inserts", and "mixed-mode" inserts.

- "Simple inserts"

  Statements for which the number of rows to be inserted can be determined in advance (when the statement is initially processed). This includes single-row and multiple-row `INSERT` and `REPLACE` statements that do not have a nested subquery, but not `INSERT ... ON DUPLICATE KEY UPDATE`.

- “Bulk inserts”

  Statements for which the number of rows to be inserted (and the number of required auto-increment values) is not known in advance. This includes `INSERT ... SELECT`, `REPLACE ... SELECT`, and `LOAD DATA` statements, but not plain `INSERT`. `InnoDB` assigns new values for the `AUTO_INCREMENT` column one at a time as each row is processed.

- “Mixed-mode inserts”

  These are “simple insert” statements that specify the auto-increment value for some (but not all) of the new rows. An example follows, where `c1` is an `AUTO_INCREMENT` column of table `t1`:

  ```
  INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
  ```

  Another type of “mixed-mode insert” is `INSERT ... ON DUPLICATE KEY UPDATE`, which in the worst case is in effect an `INSERT` followed by a `UPDATE`, where the allocated value for the `AUTO_INCREMENT` column may or may not be used during the update phase.

There are three possible settings for the `innodb_autoinc_lock_mode` configuration parameter. The settings are 0, 1, or 2, for “traditional”, “consecutive”, or “interleaved” lock mode, respectively. As of MySQL 8.0, interleaved lock mode (`innodb_autoinc_lock_mode=2`) is the default setting. Prior to MySQL 8.0, consecutive lock mode is the default (`innodb_autoinc_lock_mode=1`).

The default setting of interleaved lock mode in MySQL 8.0 reflects the change from statement-based replication to row based replication as the default replication type. Statement-based replication requires the consecutive auto-increment lock mode to ensure that auto-increment values are assigned in a predictable and repeatable order for a given sequence of SQL statements, whereas row-based replication is not sensitive to the execution order of SQL statements.

- `innodb_autoinc_lock_mode = 0` (“traditional” lock mode)

  The traditional lock mode provides the same behavior that existed before the `innodb_autoinc_lock_mode` configuration parameter was introduced in MySQL 5.1. The traditional lock mode option is provided for backward compatibility, performance testing, and working around issues with “mixed-mode inserts”, due to possible differences in semantics.

  In this lock mode, all “INSERT-like” statements obtain a special table-level `AUTO-INC` lock for inserts into tables with `AUTO_INCREMENT` columns. This lock is normally held to the end of the statement (not to the end of the transaction) to ensure that auto-increment values are assigned in a predictable and repeatable order for a given sequence of `INSERT` statements, and to ensure that auto-increment values assigned by any given statement are consecutive.

  In the case of statement-based replication, this means that when an SQL statement is replicated on a slave server, the same values are used for the auto-increment column as on the master server. The result of execution of multiple `INSERT` statements is deterministic, and the slave reproduces the same data as on the master. If auto-increment values generated by multiple `INSERT` statements were interleaved, the result of two concurrent `INSERT` statements would be nondeterministic, and could not reliably be propagated to a slave server using statement-based replication.

  To make this clear, consider an example that uses this table:

  ```
  CREATE TABLE t1 (
    c1 INT(11) NOT NULL AUTO_INCREMENT,
    c2 VARCHAR(10) DEFAULT NULL,
    PRIMARY KEY (c1)
  ) ENGINE=InnoDB;
  ```

Suppose that there are two transactions running, each inserting rows into a table with an `AUTO_INCREMENT` column. One transaction is using an `INSERT ... SELECT` statement that inserts 1000 rows, and another is using a simple `INSERT` statement that inserts one row:

```
Tx1: INSERT INTO t1 (c2) SELECT 1000 rows from another table ...
Tx2: INSERT INTO t1 (c2) VALUES ('xxx');
```

`InnoDB` cannot tell in advance how many rows are retrieved from the `SELECT` in the `INSERT` statement in Tx1, and it assigns the auto-increment values one at a time as the statement proceeds. With a table-level lock, held to the end of the statement, only one `INSERT` statement referring to table `t1` can execute at a time, and the generation of auto-increment numbers by different statements is not interleaved. The auto-increment value generated by the Tx1 `INSERT ... SELECT` statement are consecutive, and the (single) auto-increment value used by the `INSERT` statement in Tx2 are either smaller or larger than all those used for Tx1, depending on which statement executes first.

As long as the SQL statements execute in the same order when replayed from the binary log (when using statement-based replication, or in recovery scenarios), the results are the same as they were when Tx1 and Tx2 first ran. Thus, table-level locks held until the end of a statement make `INSERT` statements using auto-increment safe for use with statement-based replication. However, those table-level locks limit concurrency and scalability when multiple transactions are executing insert statements at the same time.

In the preceding example, if there were no table-level lock, the value of the auto-increment column used for the `INSERT` in Tx2 depends on precisely when the statement executes. If the `INSERT` of Tx2 executes while the `INSERT` of Tx1 is running (rather than before it starts or after it completes), the specific auto-increment values assigned by the two `INSERT` statements are nondeterministic, and may vary from run to run.

Under the consecutive lock mode, `InnoDB` can avoid using table-level `AUTO-INC` locks for “simple insert” statements where the number of rows is known in advance, and still preserve deterministic execution and safety for statement-based replication.

If you are not using the binary log to replay SQL statements as part of recovery or replication, the interleaved lock mode can be used to eliminate all use of table-level `AUTO-INC` locks for even greater concurrency and performance, at the cost of permitting gaps in auto-increment numbers assigned by a statement and potentially having the numbers assigned by concurrently executing statements interleaved.

- `innodb_autoinc_lock_mode = 1` (“consecutive” lock mode)

  In this mode, “bulk inserts” use the special `AUTO-INC` table-level lock and hold it until the end of the statement. This applies to all `INSERT ... SELECT`, `REPLACE ... SELECT`, and `LOAD DATA` statements. Only one statement holding the `AUTO-INC` lock can execute at a time. If the source table of the bulk insert operation is different from the target table, the `AUTO-INC` lock on the target table is taken after a shared lock is taken on the first row selected from the source table. If the source and target of the bulk insert operation are the same table, the `AUTO-INC` lock is taken after shared locks are taken on all selected rows.

  “Simple inserts” (for which the number of rows to be inserted is known in advance) avoid table-level `AUTO-INC` locks by obtaining the required number of auto-increment values under the control of a mutex (a light-weight lock) that is only held for the duration of the allocation process, *not* until the statement completes. No table-level `AUTO-INC` lock is used unless an `AUTO-INC` lock is held by another transaction. If another transaction holds an `AUTO-INC` lock, a “simple insert” waits for the `AUTO-INC` lock, as if it were a “bulk insert”.

This lock mode ensures that, in the presence of `INSERT` statements where the number of rows is not known in advance (and where auto-increment numbers are assigned as the statement progresses), all auto-increment values assigned by any "`INSERT`-like" statement are consecutive, and operations are safe for statement-based replication.

Simply put, this lock mode significantly improves scalability while being safe for use with statement-based replication. Further, as with "traditional" lock mode, auto-increment numbers assigned by any given statement are *consecutive*. There is *no change* in semantics compared to "traditional" mode for any statement that uses auto-increment, with one important exception.

The exception is for "mixed-mode inserts", where the user provides explicit values for an `AUTO_INCREMENT` column for some, but not all, rows in a multiple-row "simple insert". For such inserts, `InnoDB` allocates more auto-increment values than the number of rows to be inserted. However, all values automatically assigned are consecutively generated (and thus higher than) the auto-increment value generated by the most recently executed previous statement. "Excess" numbers are lost.

- `innodb_autoinc_lock_mode = 2` ("interleaved" lock mode)

  In this lock mode, no "`INSERT`-like" statements use the table-level `AUTO-INC` lock, and multiple statements can execute at the same time. This is the fastest and most scalable lock mode, but it is *not safe* when using statement-based replication or recovery scenarios when SQL statements are replayed from the binary log.

  In this lock mode, auto-increment values are guaranteed to be unique and monotonically increasing across all concurrently executing "`INSERT`-like" statements. However, because multiple statements can be generating numbers at the same time (that is, allocation of numbers is *interleaved* across statements), the values generated for the rows inserted by any given statement may not be consecutive.

  If the only statements executing are "simple inserts" where the number of rows to be inserted is known ahead of time, there are no gaps in the numbers generated for a single statement, except for "mixed-mode inserts". However, when "bulk inserts" are executed, there may be gaps in the auto-increment values assigned by any given statement.

InnoDB AUTO_INCREMENT Lock Mode Usage Implications

- Using auto-increment with replication

  If you are using statement-based replication, set `innodb_autoinc_lock_mode` to 0 or 1 and use the same value on the master and its slaves. Auto-increment values are not ensured to be the same on the slaves as on the master if you use `innodb_autoinc_lock_mode` = 2 ("interleaved") or configurations where the master and slaves do not use the same lock mode.

  If you are using row-based or mixed-format replication, all of the auto-increment lock modes are safe, since row-based replication is not sensitive to the order of execution of the SQL statements (and the mixed format uses row-based replication for any statements that are unsafe for statement-based replication).

- "Lost" auto-increment values and sequence gaps

  In all lock modes (0, 1, and 2), if a transaction that generated auto-increment values rolls back, those auto-increment values are "lost". Once a value is generated for an auto-increment column, it cannot be rolled back, whether or not the "`INSERT`-like" statement is completed, and whether or not the containing transaction is rolled back. Such lost values are not reused. Thus, there may be gaps in the values stored in an `AUTO_INCREMENT` column of a table.

- Specifying NULL or 0 for the `AUTO_INCREMENT` column

In all lock modes (0, 1, and 2), if a user specifies NULL or 0 for the AUTO_INCREMENT column in an INSERT, InnoDB treats the row as if the value was not specified and generates a new value for it.

- Assigning a negative value to the AUTO_INCREMENT column

  In all lock modes (0, 1, and 2), the behavior of the auto-increment mechanism is not defined if you assign a negative value to the AUTO_INCREMENT column.

- If the AUTO_INCREMENT value becomes larger than the maximum integer for the specified integer type

  In all lock modes (0, 1, and 2), the behavior of the auto-increment mechanism is not defined if the value becomes larger than the maximum integer that can be stored in the specified integer type.

- Gaps in auto-increment values for “bulk inserts”

  With innodb_autoinc_lock_mode set to 0 (“traditional”) or 1 (“consecutive”), the auto-increment values generated by any given statement are consecutive, without gaps, because the table-level AUTO-INC lock is held until the end of the statement, and only one such statement can execute at a time.

  With innodb_autoinc_lock_mode set to 2 (“interleaved”), there may be gaps in the auto-increment values generated by “bulk inserts,” but only if there are concurrently executing “INSERT-like” statements.

  For lock modes 1 or 2, gaps may occur between successive statements because for bulk inserts the exact number of auto-increment values required by each statement may not be known and overestimation is possible.

- Auto-increment values assigned by “mixed-mode inserts”

  Consider a “mixed-mode insert,” where a “simple insert” specifies the auto-increment value for some (but not all) resulting rows. Such a statement behaves differently in lock modes 0, 1, and 2. For example, assume c1 is an AUTO_INCREMENT column of table t1, and that the most recent automatically generated sequence number is 100.

```
mysql> CREATE TABLE t1 (
    -> c1 INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> c2 CHAR(1)
    -> ) ENGINE = INNODB;
```

  Now, consider the following “mixed-mode insert” statement:

```
mysql> INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
```

  With innodb_autoinc_lock_mode set to 0 (“traditional”), the four new rows are:

```
mysql> SELECT c1, c2 FROM t1 ORDER BY c2;
+-----+------+
| c1  | c2   |
+-----+------+
|   1 | a    |
| 101 | b    |
|   5 | c    |
| 102 | d    |
+-----+------+
```

The next available auto-increment value is 103 because the auto-increment values are allocated one at a time, not all at once at the beginning of statement execution. This result is true whether or not there are concurrently executing “INSERT-like” statements (of any type).

With innodb_autoinc_lock_mode set to 1 (“consecutive”), the four new rows are also:

```
mysql> SELECT c1, c2 FROM t1 ORDER BY c2;
+-----+------+
| c1  | c2   |
+-----+------+
|   1 | a    |
| 101 | b    |
|   5 | c    |
| 102 | d    |
+-----+------+
```

However, in this case, the next available auto-increment value is 105, not 103 because four auto-increment values are allocated at the time the statement is processed, but only two are used. This result is true whether or not there are concurrently executing “INSERT-like” statements (of any type).

With innodb_autoinc_lock_mode set to mode 2 (“interleaved”), the four new rows are:

```
mysql> SELECT c1, c2 FROM t1 ORDER BY c2;
+-----+------+
| c1  | c2   |
+-----+------+
|   1 | a    |
|   x | b    |
|   5 | c    |
|   y | d    |
+-----+------+
```

The values of *x* and *y* are unique and larger than any previously generated rows. However, the specific values of *x* and *y* depend on the number of auto-increment values generated by concurrently executing statements.

Finally, consider the following statement, issued when the most-recently generated sequence number is 100:

```
mysql> INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (101,'c'), (NULL,'d');
```

With any innodb_autoinc_lock_mode setting, this statement generates a duplicate-key error 23000 (Can't write; duplicate key in table) because 101 is allocated for the row (NULL, 'b') and insertion of the row (101, 'c') fails.

- Modifying AUTO_INCREMENT column values in the middle of a sequence of INSERT statements

  In MySQL 5.7 and earlier, modifying an AUTO_INCREMENT column value in the middle of a sequence of INSERT statements could lead to “Duplicate entry” errors. For example, if you performed an UPDATE operation that changed an AUTO_INCREMENT column value to a value larger than the current maximum auto-increment value, subsequent INSERT operations that did not specify an unused auto-increment value could encounter “Duplicate entry” errors. In MySQL 8.0 and later, if you modify an AUTO_INCREMENT column value to a value larger than the current maximum auto-increment value, the new value is persisted, and subsequent INSERT operations allocate auto-increment values starting from the new, larger value. This behavior is demonstrated in the following example.

```
mysql> CREATE TABLE t1 (
    -> c1 INT NOT NULL AUTO_INCREMENT,
    -> PRIMARY KEY (c1)
    ->  ) ENGINE = InnoDB;

mysql> INSERT INTO t1 VALUES(0), (0), (3);

mysql> SELECT c1 FROM t1;
+----+
| c1 |
+----+
|  1 |
|  2 |
|  3 |
+----+

mysql> UPDATE t1 SET c1 = 4 WHERE c1 = 1;

mysql> SELECT c1 FROM t1;
+----+
| c1 |
+----+
|  2 |
|  3 |
|  4 |
+----+

mysql> INSERT INTO t1 VALUES(0);

mysql> SELECT c1 FROM t1;
+----+
| c1 |
+----+
|  2 |
|  3 |
|  4 |
|  5 |
+----+
```

InnoDB AUTO_INCREMENT Counter Initialization

This section describes how InnoDB initializes AUTO_INCREMENT counters.

If you specify an AUTO_INCREMENT column for an InnoDB table, the in-memory table object contains a special counter called the auto-increment counter that is used when assigning new values for the column.

In MySQL 5.7 and earlier, the auto-increment counter is stored only in main memory, not on disk. To initialize an auto-increment counter after a server restart, InnoDB would execute the equivalent of the following statement on the first insert into a table containing an AUTO_INCREMENT column.

```
SELECT MAX(ai_col) FROM table_name FOR UPDATE;
```

In MySQL 8.0, this behavior is changed. The current maximum auto-increment counter value is written to the redo log each time it changes and is saved to an engine-private system table on each checkpoint. These changes make the current maximum auto-increment counter value persistent across server restarts.

On a server restart following a normal shutdown, InnoDB initializes the in-memory auto-increment counter using the current maximum auto-increment value stored in the data dictionary system table.

On a server restart during crash recovery, InnoDB initializes the in-memory auto-increment counter using the current maximum auto-increment value stored in the data dictionary system table and scans the redo log for auto-increment counter values written since the last checkpoint. If a redo-logged value is greater than the in-memory counter value, the redo-logged value is applied. However, in the case of a server

crash, reuse of a previously allocated auto-increment value cannot be guaranteed. Each time the current maximum auto-increment value is changed due to an `INSERT` or `UPDATE` operation, the new value is written to the redo log, but if the crash occurs before the redo log is flushed to disk, the previously allocated value could be reused when the auto-increment counter is initialized after the server is restarted.

The only circumstance in which `InnoDB` uses the equivalent of a `SELECT MAX(ai_col) FROM table_name FOR UPDATE` statement in MySQL 8.0 and later to initialize an auto-increment counter is when importing a tablespace without a `.cfg` metadata file. Otherwise, the current maximum auto-increment counter value is read from the `.cfg` metadata file.

In MySQL 5.7 and earlier, a server restart cancels the effect of the `AUTO_INCREMENT = N` table option, which may be used in a `CREATE TABLE` or `ALTER TABLE` statement to set an initial counter value or alter the existing counter value, respectively. In MySQL 8.0, a server restart does not cancel the effect of the `AUTO_INCREMENT = N` table option. If you initialize the auto-increment counter to a specific value, or if you alter the auto-increment counter value to a larger value, the new value is persisted across server restarts.

Note

`ALTER TABLE ... AUTO_INCREMENT = N` can only change the auto-increment counter value to a value larger than the current maximum.

In MySQL 5.7 and earlier, a server restart immediately following a `ROLLBACK` operation could result in the reuse of auto-increment values that were previously allocated to the rolled-back transaction, effectively rolling back the current maximum auto-increment value. In MySQL 8.0, the current maximum auto-increment value is persisted, preventing the reuse of previously allocated values.

If a `SHOW TABLE STATUS` statement examines a table before the auto-increment counter is initialized, `InnoDB` opens the table and initializes the counter value using the current maximum auto-increment value that is stored in the data dictionary system table. The value is stored in memory for use by later inserts or updates. Initialization of the counter value uses a normal exclusive-locking read on the table which lasts to the end of the transaction. `InnoDB` follows the same procedure when initializing the auto-increment counter for a newly created table that has a user-specified auto-increment value that is greater than 0.

After the auto-increment counter is initialized, if you do not explicitly specify an auto-increment value when inserting a row, `InnoDB` implicitly increments the counter and assigns the new value to the column. If you insert a row that explicitly specifies an auto-increment column value, and the value is greater than the current maximum counter value, the counter is set to the specified value.

`InnoDB` uses the in-memory auto-increment counter as long as the server runs. When the server is stopped and restarted, `InnoDB` reinitializes the auto-increment counter, as described earlier.

The `auto_increment_offset` configuration option determines the starting point for the `AUTO_INCREMENT` column value. The default setting is 1.

The `auto_increment_increment` configuration option controls the interval between successive column values. The default setting is 1.

15.6.1.5 InnoDB and FOREIGN KEY Constraints

How the `InnoDB` storage engine handles foreign key constraints is described under the following topics in this section:

- Foreign Key Definitions
- Referential Actions
- Foreign Key Restrictions for Generated Columns and Virtual Indexes

For foreign key usage information and examples, see Section 13.1.20.6, “Using FOREIGN KEY Constraints”.

Foreign Key Definitions

Foreign key definitions for `InnoDB` tables are subject to the following conditions:

- `InnoDB` permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are the *first* columns in the same order. Hidden columns that `InnoDB` adds to an index are also considered (see Section 15.6.2.1, “Clustered and Secondary Indexes”).
- `InnoDB` does not currently support foreign keys for tables with user-defined partitioning. This means that no user-partitioned `InnoDB` table may contain foreign key references or columns referenced by foreign keys.
- `InnoDB` allows a foreign key constraint to reference a nonunique key. *This is an `InnoDB` extension to standard SQL.*

Referential Actions

Referential actions for foreign keys of `InnoDB` tables are subject to the following conditions:

- While `SET DEFAULT` is allowed by the MySQL Server, it is rejected as invalid by `InnoDB`. `CREATE TABLE` and `ALTER TABLE` statements using this clause are not allowed for InnoDB tables.
- If there are several rows in the parent table that have the same referenced key value, `InnoDB` acts in foreign key checks as if the other parent rows with the same key value do not exist. For example, if you have defined a `NO ACTION` type constraint, and there is a child row with several parent rows, `InnoDB` does not permit the deletion of any of those parent rows.
- `InnoDB` performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.
- If `ON UPDATE CASCADE` or `ON UPDATE SET NULL` recurses to update the *same table* it has previously updated during the cascade, it acts like `NO ACTION`. This means that you cannot use self-referential `ON UPDATE CASCADE` or `ON UPDATE SET NULL` operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential `ON DELETE SET NULL`, on the other hand, is possible, as is a self-referential `ON DELETE CASCADE`. Cascading operations may not be nested more than 15 levels deep.
- Like MySQL in general, in an SQL statement that inserts, deletes, or updates many rows, `InnoDB` checks `UNIQUE` and `FOREIGN KEY` constraints row-by-row. When performing foreign key checks, `InnoDB` sets shared row-level locks on child or parent records it has to look at. `InnoDB` checks foreign key constraints immediately; the check is not deferred to transaction commit. According to the SQL standard, the default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. Until `InnoDB` implements deferred constraint checking, some things are impossible, such as deleting a record that refers to itself using a foreign key.

Foreign Key Restrictions for Generated Columns and Virtual Indexes

- A foreign key constraint on a stored generated column cannot use `CASCADE`, `SET NULL`, or `SET DEFAULT` as `ON UPDATE` referential actions, nor can it use `SET NULL` or `SET DEFAULT` as `ON DELETE` referential actions.
- A foreign key constraint on the base column of a stored generated column cannot use `CASCADE`, `SET NULL`, or `SET DEFAULT` as `ON UPDATE` or `ON DELETE` referential actions.
- A foreign key constraint cannot reference a virtual generated column.

- Prior to MySQL 8.0, a foreign key constraint cannot reference a secondary index defined on a virtual generated column.

15.6.1.6 Limits on InnoDB Tables

Limits on `InnoDB` tables are described under the following topics in this section:

- Maximums and Minimums
- Restrictions on InnoDB Tables
- Locking and Transactions

Warning

Before using NFS with `InnoDB`, review potential issues outlined in Using NFS with MySQL.

Maximums and Minimums

- A table can contain a maximum of 1017 columns. Virtual generated columns are included in this limit.
- A table can contain a maximum of 64 secondary indexes.
- The index key prefix length limit is 3072 bytes for `InnoDB` tables that use `DYNAMIC` or `COMPRESSED` row format.

  The index key prefix length limit is 767 bytes for `InnoDB` tables that use `REDUNDANT` or `COMPACT` row format. For example, you might hit this limit with a column prefix index of more than 191 characters on a `TEXT` or `VARCHAR` column, assuming a `utf8mb4` character set and the maximum of 4 bytes for each character.

  Attempting to use an index key prefix length that exceeds the limit returns an error.

  The limits that apply to index key prefixes also apply to full-column index keys.
- If you reduce the `InnoDB` page size to 8KB or 4KB by specifying the `innodb_page_size` option when creating the MySQL instance, the maximum length of the index key is lowered proportionally, based on the limit of 3072 bytes for a 16KB page size. That is, the maximum index key length is 1536 bytes when the page size is 8KB, and 768 bytes when the page size is 4KB.
- A maximum of 16 columns is permitted for multicolumn indexes. Exceeding the limit returns an error.

```
ERROR 1070 (42000): Too many key parts specified; max 16 parts allowed
```

- The maximum row length, except for variable-length columns (`VARBINARY`, `VARCHAR`, `BLOB` and `TEXT`), is slightly less than half of a page for 4KB, 8KB, 16KB, and 32KB page sizes. For example, the maximum row length for the default `innodb_page_size` of 16KB is about 8000 bytes. However, for an `InnoDB` page size of 64KB, the maximum row length is approximately 16000 bytes. `LONGBLOB` and `LONGTEXT` columns must be less than 4GB, and the total row length, including `BLOB` and `TEXT` columns, must be less than 4GB.

  If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page, as described in Section 15.11.2, “File Space Management”.
- Although `InnoDB` supports row sizes larger than 65,535 bytes internally, MySQL itself imposes a row-size limit of 65,535 for the combined size of all columns:

```
mysql> CREATE TABLE t (a VARCHAR(8000), b VARCHAR(10000),
    -> c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
    -> f VARCHAR(10000), g VARCHAR(10000)) ENGINE=InnoDB;
ERROR 1118 (42000): Row size too large. The maximum row size for the
used table type, not counting BLOBs, is 65535. You have to change some
columns to TEXT or BLOBs
```

See Section 8.4.7, “Limits on Table Column Count and Row Size”.

- On some older operating systems, files must be less than 2GB. This is not a limitation of `InnoDB` itself, but if you require a large tablespace, configure it using several smaller data files rather than one large data file.
- The combined size of the `InnoDB` log files can be up to 512GB.
- The minimum tablespace size is slightly larger than 10MB. The maximum tablespace size depends on the `InnoDB` page size.

  Table 15.3 InnoDB Maximum Tablespace Size

InnoDB Page Size	Maximum Tablespace Size
4KB	16TB
8KB	32TB
16KB	64TB
32KB	128TB
64KB	256TB

  The maximum tablespace size is also the maximum size for a table.
- The path of a tablespace file, including the file name, cannot exceed the `MAX_PATH` limit on Windows. Prior to Windows 10, the `MAX_PATH` limit is 260 characters. As of Windows 10, version 1607, `MAX_PATH` limitations are removed from common Win32 file and directory functions, but you must enable the new behavior.
- The default page size in `InnoDB` is 16KB. You can increase or decrease the page size by configuring the `innodb_page_size` option when creating the MySQL instance.

  32KB and 64KB page sizes are supported, but `ROW_FORMAT=COMPRESSED` is unsupported for page sizes greater than 16KB. For both 32KB and 64KB page sizes, the maximum record size is 16KB. For `innodb_page_size=32KB`, extent size is 2MB. For `innodb_page_size=64KB`, extent size is 4MB.

  A MySQL instance using a particular `InnoDB` page size cannot use data files or log files from an instance that uses a different page size.

Restrictions on InnoDB Tables

- `ANALYZE TABLE` determines index cardinality (as displayed in the `Cardinality` column of `SHOW INDEX` output) by performing random dives on each of the index trees and updating index cardinality estimates accordingly. Because these are only estimates, repeated runs of `ANALYZE TABLE` could produce different numbers. This makes `ANALYZE TABLE` fast on `InnoDB` tables but not 100% accurate because it does not take all rows into account.

  You can make the statistics collected by `ANALYZE TABLE` more precise and more stable by turning on the `innodb_stats_persistent` configuration option, as explained in Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”. When that setting is enabled, it is important to run `ANALYZE`

TABLE after major changes to indexed column data, because the statistics are not recalculated periodically (such as after a server restart).

If the persistent statistics setting is enabled, you can change the number of random dives by modifying the innodb_stats_persistent_sample_pages system variable. If the persistent statistics setting is disabled, modify the innodb_stats_transient_sample_pages system variable instead.

MySQL uses index cardinality estimates in join optimization. If a join is not optimized in the right way, try using ANALYZE TABLE. In the few cases that ANALYZE TABLE does not produce values good enough for your particular tables, you can use FORCE INDEX with your queries to force the use of a particular index, or set the max_seeks_for_key system variable to ensure that MySQL prefers index lookups over table scans. See Section B.4.5, “Optimizer-Related Issues”.

- If statements or transactions are running on a table, and ANALYZE TABLE is run on the same table followed by a second ANALYZE TABLE operation, the second ANALYZE TABLE operation is blocked until the statements or transactions are completed. This behavior occurs because ANALYZE TABLE marks the currently loaded table definition as obsolete when ANALYZE TABLE is finished running. New statements or transactions (including a second ANALYZE TABLE statement) must load the new table definition into the table cache, which cannot occur until currently running statements or transactions are completed and the old table definition is purged. Loading multiple concurrent table definitions is not supported.

- SHOW TABLE STATUS does not give accurate statistics on InnoDB tables except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimization.

- InnoDB does not keep an internal count of rows in a table because concurrent transactions might “see” different numbers of rows at the same time. Consequently, SELECT COUNT(*) statements only count rows visible to the current transaction.

  For information about how InnoDB processes SELECT COUNT(*) statements, refer to the COUNT() description in Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”.

- On Windows, InnoDB always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, create all databases and tables using lowercase names.

- An AUTO_INCREMENT column *ai_col* must be defined as part of an index such that it is possible to perform the equivalent of an indexed SELECT MAX(*ai_col*) lookup on the table to obtain the maximum column value. Typically, this is achieved by making the column the first column of some table index.

- InnoDB sets an exclusive lock on the end of the index associated with the AUTO_INCREMENT column while initializing a previously specified AUTO_INCREMENT column on a table.

  With innodb_autoinc_lock_mode=0, InnoDB uses a special AUTO-INC table lock mode where the lock is obtained and held to the end of the current SQL statement while accessing the auto-increment counter. Other clients cannot insert into the table while the AUTO-INC table lock is held. The same behavior occurs for “bulk inserts” with innodb_autoinc_lock_mode=1. Table-level AUTO-INC locks are not used with innodb_autoinc_lock_mode=2. For more information, See Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”.

- When an AUTO_INCREMENT integer column runs out of values, a subsequent INSERT operation returns a duplicate-key error. This is general MySQL behavior.

- DELETE FROM *tbl_name* does not regenerate the table but instead deletes all rows, one by one.

- Cascaded foreign key actions do not activate triggers.

- You cannot create a table with a column name that matches the name of an internal `InnoDB` column (including `DB_ROW_ID`, `DB_TRX_ID`, `DB_ROLL_PTR`, and `DB_MIX_ID`). This restriction applies to use of the names in any letter case.

```
mysql> CREATE TABLE t1 (c1 INT, db_row_id INT) ENGINE=INNODB;
ERROR 1166 (42000): Incorrect column name 'db_row_id'
```

Locking and Transactions

- `LOCK TABLES` acquires two locks on each table if `innodb_table_locks=1` (the default). In addition to a table lock on the MySQL layer, it also acquires an `InnoDB` table lock. Versions of MySQL before 4.1.2 did not acquire `InnoDB` table locks; the old behavior can be selected by setting `innodb_table_locks=0`. If no `InnoDB` table lock is acquired, `LOCK TABLES` completes even if some records of the tables are being locked by other transactions.

  In MySQL 8.0, `innodb_table_locks=0` has no effect for tables locked explicitly with `LOCK TABLES ... WRITE`. It does have an effect for tables locked for read or write by `LOCK TABLES ... WRITE` implicitly (for example, through triggers) or by `LOCK TABLES ... READ`.

- All `InnoDB` locks held by a transaction are released when the transaction is committed or aborted. Thus, it does not make much sense to invoke `LOCK TABLES` on `InnoDB` tables in `autocommit=1` mode because the acquired `InnoDB` table locks would be released immediately.

- You cannot lock additional tables in the middle of a transaction because `LOCK TABLES` performs an implicit `COMMIT` and `UNLOCK TABLES`.

- For limits associated with concurrent read-write transactions, see Section 15.6.6, “Undo Logs”.

15.6.2 Indexes

This section covers topics related to `InnoDB` indexes.

15.6.2.1 Clustered and Secondary Indexes

Every `InnoDB` table has a special index called the clustered index where the data for the rows is stored. Typically, the clustered index is synonymous with the primary key. To get the best performance from queries, inserts, and other database operations, you must understand how `InnoDB` uses the clustered index to optimize the most common lookup and DML operations for each table.

- When you define a `PRIMARY KEY` on your table, `InnoDB` uses it as the clustered index. Define a primary key for each table that you create. If there is no logical unique and non-null column or set of columns, add a new auto-increment column, whose values are filled in automatically.

- If you do not define a `PRIMARY KEY` for your table, MySQL locates the first `UNIQUE` index where all the key columns are `NOT NULL` and `InnoDB` uses it as the clustered index.

- If the table has no `PRIMARY KEY` or suitable `UNIQUE` index, `InnoDB` internally generates a hidden clustered index named `GEN_CLUST_INDEX` on a synthetic column containing row ID values. The rows are ordered by the ID that `InnoDB` assigns to the rows in such a table. The row ID is a 6-byte field that increases monotonically as new rows are inserted. Thus, the rows ordered by the row ID are physically in insertion order.

How the Clustered Index Speeds Up Queries

Accessing a row through the clustered index is fast because the index search leads directly to the page with all the row data. If a table is large, the clustered index architecture often saves a disk I/O operation when compared to storage organizations that store row data using a different page from the index record.

How Secondary Indexes Relate to the Clustered Index

All indexes other than the clustered index are known as secondary indexes. In InnoDB, each record in a secondary index contains the primary key columns for the row, as well as the columns specified for the secondary index. InnoDB uses this primary key value to search for the row in the clustered index.

If the primary key is long, the secondary indexes use more space, so it is advantageous to have a short primary key.

For guidelines to take advantage of InnoDB clustered and secondary indexes, see Section 8.3, “Optimization and Indexes”.

15.6.2.2 The Physical Structure of an InnoDB Index

With the exception of spatial indexes, InnoDB indexes are B-tree data structures. Spatial indexes use R-trees, which are specialized data structures for indexing multi-dimensional data. Index records are stored in the leaf pages of their B-tree or R-tree data structure. The default size of an index page is 16KB.

When new records are inserted into an InnoDB clustered index, InnoDB tries to leave 1/16 of the page free for future insertions and updates of the index records. If index records are inserted in a sequential order (ascending or descending), the resulting index pages are about 15/16 full. If records are inserted in a random order, the pages are from 1/2 to 15/16 full.

InnoDB performs a bulk load when creating or rebuilding B-tree indexes. This method of index creation is known as a sorted index build. The innodb_fill_factor configuration option defines the percentage of space on each B-tree page that is filled during a sorted index build, with the remaining space reserved for future index growth. Sorted index builds are not supported for spatial indexes. For more information, see Section 15.6.2.3, “Sorted Index Builds”. An innodb_fill_factor setting of 100 leaves 1/16 of the space in clustered index pages free for future index growth.

If the fill factor of an InnoDB index page drops below the MERGE_THRESHOLD, which is 50% by default if not specified, InnoDB tries to contract the index tree to free the page. The MERGE_THRESHOLD setting applies to both B-tree and R-tree indexes. For more information, see Section 15.8.11, “Configuring the Merge Threshold for Index Pages”.

You can define the page size for all InnoDB tablespaces in a MySQL instance by setting the innodb_page_size configuration option prior to initializing the MySQL instance. Once the page size for an instance is defined, you cannot change it without reinitializing the instance. Supported sizes are 64KB, 32KB, 16KB (default), 8KB, and 4KB.

A MySQL instance using a particular InnoDB page size cannot use data files or log files from an instance that uses a different page size.

15.6.2.3 Sorted Index Builds

InnoDB performs a bulk load instead of inserting one index record at a time when creating or rebuilding indexes. This method of index creation is also known as a sorted index build. Sorted index builds are not supported for spatial indexes.

There are three phases to an index build. In the first phase, the clustered index is scanned, and index entries are generated and added to the sort buffer. When the sort buffer becomes full, entries are sorted and written out to a temporary intermediate file. This process is also known as a “run”. In the second phase, with one or more runs written to the temporary intermediate file, a merge sort is performed on all entries in the file. In the third and final phase, the sorted entries are inserted into the B-tree.

Prior to the introduction of sorted index builds, index entries were inserted into the B-tree one record at a time using insert APIs. This method involved opening a B-tree cursor to find the insert position and then

inserting entries into a B-tree page using an optimistic insert. If an insert failed due to a page being full, a pessimistic insert would be performed, which involves opening a B-tree cursor and splitting and merging B-tree nodes as necessary to find space for the entry. The drawbacks of this “top-down” method of building an index are the cost of searching for an insert position and the constant splitting and merging of B-tree nodes.

Sorted index builds use a “bottom-up” approach to building an index. With this approach, a reference to the right-most leaf page is held at all levels of the B-tree. The right-most leaf page at the necessary B-tree depth is allocated and entries are inserted according to their sorted order. Once a leaf page is full, a node pointer is appended to the parent page and a sibling leaf page is allocated for the next insert. This process continues until all entries are inserted, which may result in inserts up to the root level. When a sibling page is allocated, the reference to the previously pinned leaf page is released, and the newly allocated leaf page becomes the right-most leaf page and new default insert location.

Reserving B-tree Page Space for Future Index Growth

To set aside space for future index growth, you can use the `innodb_fill_factor` configuration option to reserve a percentage of B-tree page space. For example, setting `innodb_fill_factor` to 80 reserves 20 percent of the space in B-tree pages during a sorted index build. This setting applies to both B-tree leaf and non-leaf pages. It does not apply to external pages used for `TEXT` or `BLOB` entries. The amount of space that is reserved may not be exactly as configured, as the `innodb_fill_factor` value is interpreted as a hint rather than a hard limit.

Sorted Index Builds and Full-Text Index Support

Sorted index builds are supported for fulltext indexes. Previously, SQL was used to insert entries into a fulltext index.

Sorted Index Builds and Compressed Tables

For compressed tables, the previous index creation method appended entries to both compressed and uncompressed pages. When the modification log (representing free space on the compressed page) became full, the compressed page would be recompressed. If compression failed due to a lack of space, the page would be split. With sorted index builds, entries are only appended to uncompressed pages. When an uncompressed page becomes full, it is compressed. Adaptive padding is used to ensure that compression succeeds in most cases, but if compression fails, the page is split and compression is attempted again. This process continues until compression is successful. For more information about compression of B-Tree pages, see Section 15.9.1.5, “How Compression Works for InnoDB Tables”.

Sorted Index Builds and Redo Logging

Redo logging is disabled during a sorted index build. Instead, there is a checkpoint to ensure that the index build can withstand a crash or failure. The checkpoint forces a write of all dirty pages to disk. During a sorted index build, the page cleaner thread is signaled periodically to flush dirty pages to ensure that the checkpoint operation can be processed quickly. Normally, the page cleaner thread flushes dirty pages when the number of clean pages falls below a set threshold. For sorted index builds, dirty pages are flushed promptly to reduce checkpoint overhead and to parallelize I/O and CPU activity.

Sorted Index Builds and Optimizer Statistics

Sorted index builds may result in optimizer statistics that differ from those generated by the previous method of index creation. The difference in statistics, which is not expected to affect workload performance, is due to the different algorithm used to populate the index.

15.6.2.4 InnoDB FULLTEXT Indexes

FULLTEXT indexes are created on text-based columns (CHAR, VARCHAR, or TEXT columns) to help speed up queries and DML operations on data contained within those columns, omitting any words that are defined as stopwords.

A FULLTEXT index is defined as part of a CREATE TABLE statement or added to an existing table using ALTER TABLE or CREATE INDEX.

Full-text search is performed using MATCH() ... AGAINST syntax. For usage information, see Section 12.9, “Full-Text Search Functions”.

InnoDB FULLTEXT indexes are described under the following topics in this section:

- InnoDB Full-Text Index Design
- InnoDB Full-Text Index Tables
- InnoDB Full-Text Index Cache
- InnoDB Full-Text Index Document ID and FTS_DOC_ID Column
- InnoDB Full-Text Index Deletion Handling
- InnoDB Full-Text Index Transaction Handling
- Monitoring InnoDB Full-Text Indexes

InnoDB Full-Text Index Design

InnoDB FULLTEXT indexes have an inverted index design. Inverted indexes store a list of words, and for each word, a list of documents that the word appears in. To support proximity search, position information for each word is also stored, as a byte offset.

InnoDB Full-Text Index Tables

When creating an InnoDB FULLTEXT index, a set of index tables is created, as shown in the following example:

```
mysql> CREATE TABLE opening_lines (
       id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
       opening_line TEXT(500),
       author VARCHAR(200),
       title VARCHAR(200),
       FULLTEXT idx (opening_line)
       ) ENGINE=InnoDB;

mysql> SELECT table_id, name, space from INFORMATION_SCHEMA.INNODB_TABLES
       WHERE name LIKE 'test/%';
+----------+--------------------------------------------------------+-------+
| table_id | name                                                   | space |
+----------+--------------------------------------------------------+-------+
|      333 | test/fts_0000000000000147_00000000000001c9_index_1 |   289 |
|      334 | test/fts_0000000000000147_00000000000001c9_index_2 |   290 |
|      335 | test/fts_0000000000000147_00000000000001c9_index_3 |   291 |
|      336 | test/fts_0000000000000147_00000000000001c9_index_4 |   292 |
|      337 | test/fts_0000000000000147_00000000000001c9_index_5 |   293 |
|      338 | test/fts_0000000000000147_00000000000001c9_index_6 |   294 |
|      330 | test/fts_0000000000000147_being_deleted            |   286 |
|      331 | test/fts_0000000000000147_being_deleted_cache      |   287 |
|      332 | test/fts_0000000000000147_config                   |   288 |
|      328 | test/fts_0000000000000147_deleted                  |   284 |
|      329 | test/fts_0000000000000147_deleted_cache            |   285 |
```

```
|      327 | test/opening_lines                                      |   283 |
+----------+---------------------------------------------------------+-------+
```

The first six tables represent the inverted index and are referred to as auxiliary index tables. When incoming documents are tokenized, the individual words (also referred to as “tokens”) are inserted into the index tables along with position information and the associated Document ID (`DOC_ID`). The words are fully sorted and partitioned among the six index tables based on the character set sort weight of the word's first character.

The inverted index is partitioned into six auxiliary index tables to support parallel index creation. By default, two threads tokenize, sort, and insert words and associated data into the index tables. The number of threads is configurable using the `innodb_ft_sort_pll_degree` option. Consider increasing the number of threads when creating `FULLTEXT` indexes on large tables.

Auxiliary index table names are prefixed with `fts_` and postfixed with `index_*`. Each index table is associated with the indexed table by a hex value in the index table name that matches the `table_id` of the indexed table. For example, the `table_id` of the `test/opening_lines` table is `327`, for which the hex value is 0x147. As shown in the preceding example, the “147” hex value appears in the names of index tables that are associated with the `test/opening_lines` table.

A hex value representing the `index_id` of the `FULLTEXT` index also appears in auxiliary index table names. For example, in the auxiliary table name `test/fts_0000000000000147_00000000000001c9_index_1`, the hex value `1c9` has a decimal value of 457. The index defined on the `opening_lines` table (`idx`) can be identified by querying the `INFORMATION_SCHEMA.INNODB_INDEXES` table for this value (457).

```
mysql> SELECT index_id, name, table_id, space from INFORMATION_SCHEMA.INNODB_INDEXES
       WHERE index_id=457;
+----------+------+----------+-------+
| index_id | name | table_id | space |
+----------+------+----------+-------+
|      457 | idx  |      327 |   283 |
+----------+------+----------+-------+
```

Index tables are stored in their own tablespace if the primary table is created in a file-per-table tablespace.

The other index tables shown in the preceding example are referred to as common index tables and are used for deletion handling and storing the internal state of `FULLTEXT` indexes. Unlike the inverted index tables, which are created for each full-text index, this set of tables is common to all full-text indexes created on a particular table.

Common auxiliary tables are retained even if full-text indexes are dropped. When a full-text index is dropped, the `FTS_DOC_ID` column that was created for the index is retained, as removing the `FTS_DOC_ID` column would require rebuilding the table. Common axillary tables are required to manage the `FTS_DOC_ID` column.

- `fts_*_deleted` and `fts_*_deleted_cache`

  Contain the document IDs (DOC_ID) for documents that are deleted but whose data is not yet removed from the full-text index. The `fts_*_deleted_cache` is the in-memory version of the `fts_*_deleted` table.

- `fts_*_being_deleted` and `fts_*_being_deleted_cache`

  Contain the document IDs (DOC_ID) for documents that are deleted and whose data is currently in the process of being removed from the full-text index. The `fts_*_being_deleted_cache` table is the in-memory version of the `fts_*_being_deleted` table.

- `fts_*_config`

  Stores information about the internal state of the `FULLTEXT` index. Most importantly, it stores the `FTS_SYNCED_DOC_ID`, which identifies documents that have been parsed and flushed to disk. In case of crash recovery, `FTS_SYNCED_DOC_ID` values are used to identify documents that have not been flushed to disk so that the documents can be re-parsed and added back to the `FULLTEXT` index cache. To view the data in this table, query the `INFORMATION_SCHEMA.INNODB_FT_CONFIG` table.

InnoDB Full-Text Index Cache

When a document is inserted, it is tokenized, and the individual words and associated data are inserted into the `FULLTEXT` index. This process, even for small documents, could result in numerous small insertions into the auxiliary index tables, making concurrent access to these tables a point of contention. To avoid this problem, `InnoDB` uses a `FULLTEXT` index cache to temporarily cache index table insertions for recently inserted rows. This in-memory cache structure holds insertions until the cache is full and then batch flushes them to disk (to the auxiliary index tables). You can query the `INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE` table to view tokenized data for recently inserted rows.

The caching and batch flushing behavior avoids frequent updates to auxiliary index tables, which could result in concurrent access issues during busy insert and update times. The batching technique also avoids multiple insertions for the same word, and minimizes duplicate entries. Instead of flushing each word individually, insertions for the same word are merged and flushed to disk as a single entry, improving insertion efficiency while keeping auxiliary index tables as small as possible.

The `innodb_ft_cache_size` variable is used to configure the full-text index cache size (on a per-table basis), which affects how often the full-text index cache is flushed. You can also define a global full-text index cache size limit for all tables in a given instance using the `innodb_ft_total_cache_size` option.

The full-text index cache stores the same information as auxiliary index tables. However, the full-text index cache only caches tokenized data for recently inserted rows. The data that is already flushed to disk (to the full-text auxiliary tables) is not brought back into the full-text index cache when queried. The data in auxiliary index tables is queried directly, and results from the auxiliary index tables are merged with results from the full-text index cache before being returned.

InnoDB Full-Text Index Document ID and FTS_DOC_ID Column

`InnoDB` uses a unique document identifier referred to as a Document ID (`DOC_ID`) to map words in the full-text index to document records where the word appears. The mapping requires an `FTS_DOC_ID` column on the indexed table. If an `FTS_DOC_ID` column is not defined, `InnoDB` automatically adds a hidden `FTS_DOC_ID` column when the full-text index is created. The following example demonstrates this behavior.

The following table definition does not include an `FTS_DOC_ID` column:

```
mysql> CREATE TABLE opening_lines (
       id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
       opening_line TEXT(500),
       author VARCHAR(200),
       title VARCHAR(200)
       ) ENGINE=InnoDB;
```

When you create a full-text index on the table using `CREATE FULLTEXT INDEX` syntax, a warning is returned which reports that `InnoDB` is rebuilding the table to add the `FTS_DOC_ID` column.

```
mysql> CREATE FULLTEXT INDEX idx ON opening_lines(opening_line);
```

```
Query OK, 0 rows affected, 1 warning (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------------+
| Level   | Code | Message                                       |
+---------+------+-----------------------------------------------+
| Warning |  124 | InnoDB rebuilding table to add column FTS_DOC_ID |
+---------+------+-----------------------------------------------+
```

The same warning is returned when using `ALTER TABLE` to add a full-text index to a table that does not have an `FTS_DOC_ID` column. If you create a full-text index at `CREATE TABLE` time and do not specify an `FTS_DOC_ID` column, `InnoDB` adds a hidden `FTS_DOC_ID` column, without warning.

Defining an `FTS_DOC_ID` column at `CREATE TABLE` time is less expensive than creating a full-text index on a table that is already loaded with data. If an `FTS_DOC_ID` column is defined on a table prior to loading data, the table and its indexes do not have to be rebuilt to add the new column. If you are not concerned with `CREATE FULLTEXT INDEX` performance, leave out the `FTS_DOC_ID` column to have `InnoDB` create it for you. `InnoDB` creates a hidden `FTS_DOC_ID` column along with a unique index (`FTS_DOC_ID_INDEX`) on the `FTS_DOC_ID` column. If you want to create your own `FTS_DOC_ID` column, the column must be defined as `BIGINT UNSIGNED NOT NULL` and named `FTS_DOC_ID` (all upper case), as in the following example:

Note

The `FTS_DOC_ID` column does not need to be defined as an `AUTO_INCREMENT` column, but `AUTO_INCREMENT` could make loading data easier.

```
mysql> CREATE TABLE opening_lines (
       FTS_DOC_ID BIGINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
       opening_line TEXT(500),
       author VARCHAR(200),
       title VARCHAR(200)
       ) ENGINE=InnoDB;
```

If you choose to define the `FTS_DOC_ID` column yourself, you are responsible for managing the column to avoid empty or duplicate values. `FTS_DOC_ID` values cannot be reused, which means `FTS_DOC_ID` values must be ever increasing.

Optionally, you can create the required unique `FTS_DOC_ID_INDEX` (all upper case) on the `FTS_DOC_ID` column.

```
mysql> CREATE UNIQUE INDEX FTS_DOC_ID_INDEX on opening_lines(FTS_DOC_ID);
```

If you do not create the `FTS_DOC_ID_INDEX`, `InnoDB` creates it automatically.

Note

`FTS_DOC_ID_INDEX` cannot be defined as a descending index because the `InnoDB` SQL parser does not use descending indexes.

The permitted gap between the largest used `FTS_DOC_ID` value and new `FTS_DOC_ID` value is 65535.

To avoid rebuilding the table, the `FTS_DOC_ID` column is retained when dropping a full-text index.

InnoDB Full-Text Index Deletion Handling

Deleting a record that has a full-text index column could result in numerous small deletions in the auxiliary index tables, making concurrent access to these tables a point of contention. To avoid this problem, the

Document ID (DOC_ID) of a deleted document is logged in a special FTS_*_DELETED table whenever a record is deleted from an indexed table, and the indexed record remains in the full-text index. Before returning query results, information in the FTS_*_DELETED table is used to filter out deleted Document IDs. The benefit of this design is that deletions are fast and inexpensive. The drawback is that the size of the index is not immediately reduced after deleting records. To remove full-text index entries for deleted records, run OPTIMIZE TABLE on the indexed table with innodb_optimize_fulltext_only=ON to rebuild the full-text index. For more information, see Optimizing InnoDB Full-Text Indexes.

InnoDB Full-Text Index Transaction Handling

InnoDB FULLTEXT indexes have special transaction handling characteristics due its caching and batch processing behavior. Specifically, updates and insertions on a FULLTEXT index are processed at transaction commit time, which means that a FULLTEXT search can only see committed data. The following example demonstrates this behavior. The FULLTEXT search only returns a result after the inserted lines are committed.

```
mysql> CREATE TABLE opening_lines (
       id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
       opening_line TEXT(500),
       author VARCHAR(200),
       title VARCHAR(200),
       FULLTEXT idx (opening_line)
       ) ENGINE=InnoDB;

mysql> BEGIN;

mysql> INSERT INTO opening_lines(opening_line,author,title) VALUES
       ('Call me Ishmael.','Herman Melville','Moby-Dick'),
       ('A screaming comes across the sky.','Thomas Pynchon','Gravity\'s Rainbow'),
       ('I am an invisible man.','Ralph Ellison','Invisible Man'),
       ('Where now? Who now? When now?','Samuel Beckett','The Unnamable'),
       ('It was love at first sight.','Joseph Heller','Catch-22'),
       ('All this happened, more or less.','Kurt Vonnegut','Slaughterhouse-Five'),
       ('Mrs. Dalloway said she would buy the flowers herself.','Virginia Woolf','Mrs. Dalloway'),
       ('It was a pleasure to burn.','Ray Bradbury','Fahrenheit 451');

mysql> SELECT COUNT(*) FROM opening_lines WHERE MATCH(opening_line) AGAINST('Ishmael');
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+

mysql> COMMIT;

mysql> SELECT COUNT(*) FROM opening_lines WHERE MATCH(opening_line) AGAINST('Ishmael');
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
```

Monitoring InnoDB Full-Text Indexes

You can monitor and examine the special text-processing aspects of InnoDB FULLTEXT indexes by querying the following INFORMATION_SCHEMA tables:

- INNODB_FT_CONFIG
- INNODB_FT_INDEX_TABLE
- INNODB_FT_INDEX_CACHE

- INNODB_FT_DEFAULT_STOPWORD
- INNODB_FT_DELETED
- INNODB_FT_BEING_DELETED

You can also view basic information for FULLTEXT indexes and tables by querying INNODB_INDEXES and INNODB_TABLES.

For more information, see Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

15.6.3 Tablespaces

This section covers topics related to InnoDB tablespaces.

15.6.3.1 The System Tablespace

The system tablespace is the storage area for the doublewrite buffer and the change buffer. It may also contain table and index data if tables are created in the system tablespace rather than file-per-table or general tablespaces. In previous MySQL versions, the system tablespace contained the InnoDB data dictionary. In MySQL 8.0, InnoDB stores metadata in the MySQL data dictionary. See Chapter 14, *MySQL Data Dictionary*.

The system tablespace can have one or more data files. By default, a single system tablespace data file, named ibdata1, is created in the data directory. The size and number of system tablespace data files is defined by the innodb_data_file_path startup option. For configuration information, see System Tablespace Data File Configuration.

Additional information about the system tablespace is provided under the following topics in the section:

- Resizing the System Tablespace
- Using Raw Disk Partitions for the System Tablespace

Resizing the System Tablespace

This section describes how to increase or decrease the size of the system tablespace.

Increasing the Size of the System Tablespace

The easiest way to increase the size of the system tablespace is to configure it to be auto-extending. To do so, specify the autoextend attribute for the last data file in the innodb_data_file_path setting, and restart the server. For example:

```
innodb_data_file_path=ibdata1:10MB:autoextend
```

When the autoextend attribute is specified, the data file automatically increases in size by 8MB increments as space is required. The innodb_autoextend_increment variable controls the increment size.

You can also increase system tablespace size by adding another data file. To do so:

1. Stop the MySQL server.
2. If the last data file in the innodb_data_file_path setting is defined with the autoextend attribute, remove it, and modify the size attribute to reflect the current data file size. To determine the appropriate data file size to specify, check your file system for the file size, and round that value down to the closest MB value, where a MB is equal to 1024 x 1024.

3. Append a new data file to the innodb_data_file_path setting, optionally specifying the autoextend attribute. The autoextend attribute can be specified only for the last data file in the innodb_data_file_path setting.

4. Start the MySQL server.

For example, this tablespace has one auto-extending data file:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

Suppose that the data file has grown to 988MB over time. This is the innodb_data_file_path setting after modifying the size attribute to reflect the current data file size, and after specifying a new 50MB auto-extending data file:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

When adding a new data file, do not specify an existing file name. InnoDB creates and initializes the new data file when you start the server.

Note

You cannot increase the size of an existing system tablespace data file by changing its size attribute. For example, changing the innodb_data_file_path setting from ibdata1:10M:autoextend to ibdata1:12M:autoextend produces the following error when starting the server:

```
[ERROR] [MY-012263] [InnoDB] The Auto-extending innodb_system
data file './ibdata1' is of a different size 640 pages (rounded down to MB) than
specified in the .cnf file: initial 768 pages, max 0 (relevant if non-zero) pages!
```

The error indicates that the existing data file size (expressed in InnoDB pages) is different from the size specified in the configuration file. If you encounter this error, restore the previous innodb_data_file_path setting, and refer to the system tablespace resizing instructions.

InnoDB page size is defined by the innodb_page_size variable. The default is 16384 bytes.

Decreasing the Size of the InnoDB System Tablespace

You cannot remove a data file from the system tablespace. To decrease the system tablespace size, use this procedure:

1. Use mysqldump to dump all your InnoDB tables, including InnoDB tables located in the MySQL database.

```
mysql> SELECT TABLE_NAME from INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA='mysql' and ENGINE='InnoDB';
+---------------------------+
| TABLE_NAME                |
+---------------------------+
| columns_priv              |
| component                 |
| db                        |
| default_roles             |
| engine_cost               |
| func                      |
```

```
| global_grants             |
| gtid_executed             |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| innodb_dynamic_metadata   |
| innodb_index_stats        |
| innodb_table_stats        |
| plugin                    |
| procs_priv                |
| proxies_priv              |
| role_edges                |
| server_cost               |
| servers                   |
| slave_master_info         |
| slave_relay_log_info      |
| slave_worker_info         |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

2. Stop the server.
3. Remove all the existing tablespace files (`*.ibd`), including the `ibdata` and `ib_log` files. Do not forget to remove `*.ibd` files for tables located in the MySQL database.
4. Configure a new tablespace.
5. Restart the server.
6. Import the dump files.

Note

If your databases only use the `InnoDB` engine, it may be simpler to dump **all** databases, stop the server, remove all databases and `InnoDB` log files, restart the server, and import the dump files.

Using Raw Disk Partitions for the System Tablespace

You can use raw disk partitions as data files in the `InnoDB` system tablespace. This technique enables nonbuffered I/O on Windows and on some Linux and Unix systems without file system overhead. Perform tests with and without raw partitions to verify whether this change actually improves performance on your system.

When you use a raw disk partition, ensure that the user ID that runs the MySQL server has read and write privileges for that partition. For example, if you run the server as the `mysql` user, the partition must be readable and writeable by `mysql`. If you run the server with the `--memlock` option, the server must be run as `root`, so the partition must be readable and writeable by `root`.

The procedures described below involve option file modification. For additional information, see Section 4.2.2.2, “Using Option Files”.

Allocating a Raw Disk Partition on Linux and Unix Systems

1. When you create a new data file, specify the keyword `newraw` immediately after the data file size for the `innodb_data_file_path` option. The partition must be at least as large as the size that you

specify. Note that 1MB in InnoDB is 1024 × 1024 bytes, whereas 1MB in disk specifications usually means 1,000,000 bytes.

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
```

2. Restart the server. InnoDB notices the newraw keyword and initializes the new partition. However, do not create or change any InnoDB tables yet. Otherwise, when you next restart the server, InnoDB reinitializes the partition and your changes are lost. (As a safety measure InnoDB prevents users from modifying data when any partition with newraw is specified.)

3. After InnoDB has initialized the new partition, stop the server, change newraw in the data file specification to raw:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Graw;/dev/hdd2:2Graw
```

4. Restart the server. InnoDB now permits changes to be made.

Allocating a Raw Disk Partition on Windows

On Windows systems, the same steps and accompanying guidelines described for Linux and Unix systems apply except that the innodb_data_file_path setting differs slightly on Windows.

1. When you create a new data file, specify the keyword newraw immediately after the data file size for the innodb_data_file_path option:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Gnewraw
```

The //./ corresponds to the Windows syntax of \\.\ for accessing physical drives. In the example above, D: is the drive letter of the partition.

2. Restart the server. InnoDB notices the newraw keyword and initializes the new partition.

3. After InnoDB has initialized the new partition, stop the server, change newraw in the data file specification to raw:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Graw
```

4. Restart the server. InnoDB now permits changes to be made.

15.6.3.2 File-Per-Table Tablespaces

Historically, InnoDB tables were stored in the system tablespace. This monolithic approach was targeted at machines dedicated to database processing, with carefully planned data growth, where any disk storage allocated to MySQL would never be needed for other purposes. The file-per-table tablespace feature provides a more flexible alternative, where each InnoDB table is stored in its own tablespace data file (.ibd file). This feature is controlled by the innodb_file_per_table configuration option, which is enabled by default.

Advantages

- You can reclaim disk space when truncating or dropping a table stored in a file-per-table tablespace. Truncating or dropping tables stored in the shared system tablespace creates free space internally in the system tablespace data files (ibdata files) which can only be used for new `InnoDB` data.

  Similarly, a table-copying `ALTER TABLE` operation on table that resides in a shared tablespace can increase the amount of space used by the tablespace. Such operations may require as much additional space as the data in the table plus indexes. The additional space required for the table-copying `ALTER TABLE` operation is not released back to the operating system as it is for file-per-table tablespaces.

- The `TRUNCATE TABLE` operation is faster when run on tables stored in file-per-table tablespaces.

- You can store specific tables on separate storage devices, for I/O optimization, space management, or backup purposes by specifying the location of each table using the syntax `CREATE TABLE ... DATA DIRECTORY = absolute_path_to_directory`, as explained in Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”.

- You can run `OPTIMIZE TABLE` to compact or recreate a file-per-table tablespace. When you run an `OPTIMIZE TABLE`, `InnoDB` creates a new `.ibd` file with a temporary name, using only the space required to store actual data. When the optimization is complete, `InnoDB` removes the old `.ibd` file and replaces it with the new one. If the previous `.ibd` file grew significantly but the actual data only accounted for a portion of its size, running `OPTIMIZE TABLE` can reclaim the unused space.

- You can move individual `InnoDB` tables rather than entire databases.

- You can copy individual `InnoDB` tables from one MySQL instance to another (known as the transportable tablespace feature).

- Tables created in file-per-table tablespaces support features associated with compressed and dynamic row formats.

- You can enable more efficient storage for tables with large `BLOB` or `TEXT` columns using the dynamic row format.

- File-per-table tablespaces may improve chances for a successful recovery and save time when a corruption occurs, when a server cannot be restarted, or when backup and binary logs are unavailable.

- You can back up or restore individual tables quickly using the MySQL Enterprise Backup product, without interrupting the use of other `InnoDB` tables. This is beneficial if you have tables that require backup less frequently or on a different backup schedule. See Making a Partial Backup for details.

- File-per-table tablespaces are convenient for per-table status reporting when copying or backing up tables.

- You can monitor table size at a file system level without accessing MySQL.

- Common Linux file systems do not permit concurrent writes to a single file when `innodb_flush_method` is set to `O_DIRECT`. As a result, there are possible performance improvements when using file-per-table tablespaces in conjunction with `innodb_flush_method`.

- The system tablespace stores the data dictionary and undo logs, and is limited in size by `InnoDB` tablespace size limits. See Section 15.6.1.6, “Limits on InnoDB Tables”. With file-per-table tablespaces, each table has its own tablespace, which provides room for growth.

Potential Disadvantages

- With file-per-table tablespaces, each table may have unused space, which can only be utilized by rows of the same table. This could lead to wasted space if not properly managed.

- `fsync` operations must run on each open table rather than on a single file. Because there is a separate `fsync` operation for each file, write operations on multiple tables cannot be combined into a single I/O operation. This may require `InnoDB` to perform a higher total number of `fsync` operations.
- `mysqld` must keep one open file handle per table, which may impact performance if you have numerous tables in file-per-table tablespaces.
- More file descriptors are used.
- `innodb_file_per_table` is enabled by default in MySQL 5.6 and higher. You may consider disabling it if backward compatibility with earlier versions of MySQL is a concern.
- If many tables are growing there is potential for more fragmentation which can impede `DROP TABLE` and table scan performance. However, when fragmentation is managed, having files in their own tablespace can improve performance.
- The buffer pool is scanned when dropping a file-per-table tablespace, which can take several seconds for buffer pools that are tens of gigabytes in size. The scan is performed with a broad internal lock, which may delay other operations. Tables in the system tablespace are not affected.
- The `innodb_autoextend_increment` variable, which defines increment size (in MB) for extending the size of an auto-extending shared tablespace file when it becomes full, does not apply to file-per-table tablespace files, which are auto-extending regardless of the `innodb_autoextend_increment` setting. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

Enabling File-Per-Table Tablespaces

The `innodb_file_per_table` option is enabled by default.

To set the `innodb_file_per_table` option at startup, start the server with the `--innodb_file_per_table` command-line option, or add this line to the `[mysqld]` section of `my.cnf`:

```
[mysqld]
innodb_file_per_table=1
```

You can also set `innodb_file_per_table` dynamically, while the server is running:

```
mysql> SET GLOBAL innodb_file_per_table=1;
```

With `innodb_file_per_table` enabled, you can store `InnoDB` tables in a *`tbl_name`*`.ibd` file. Unlike the `MyISAM` storage engine, with its separate *`tbl_name`*`.MYD` and *`tbl_name`*`.MYI` files for indexes and data, `InnoDB` stores the data and the indexes together in a single `.ibd` file.

If you disable `innodb_file_per_table` in your startup options and restart the server, or disable it with the `SET GLOBAL` command, `InnoDB` creates new tables inside the system tablespace unless you have explicitly placed the table in file-per-table tablespace or general tablespace using the `CREATE TABLE ... TABLESPACE` option.

You can always read and write any `InnoDB` tables, regardless of the file-per-table setting.

To move a table from the system tablespace to its own tablespace, change the `innodb_file_per_table` setting and rebuild the table:

```
mysql> SET GLOBAL innodb_file_per_table=1;
mysql> ALTER TABLE table_name ENGINE=InnoDB;
```

Tables added to the system tablespace using `CREATE TABLE ... TABLESPACE` or `ALTER TABLE ... TABLESPACE` syntax are not affected by the `innodb_file_per_table` setting. To move these tables

from the system tablespace to a file-per-table tablespace, they must be moved explicitly using `ALTER TABLE ... TABLESPACE` syntax.

Note

`InnoDB` always needs the system tablespace because it puts its internal data dictionary and undo logs there. The `.ibd` files are not sufficient for `InnoDB` to operate.

When a table is moved out of the system tablespace into its own `.ibd` file, the data files that make up the system tablespace remain the same size. The space formerly occupied by the table can be reused for new `InnoDB` data, but is not reclaimed for use by the operating system. When moving large `InnoDB` tables out of the system tablespace, where disk space is limited, you may prefer to enable `innodb_file_per_table` and recreate the entire instance using the `mysqldump` command. As mentioned above, tables added to the system tablespace using `CREATE TABLE ... TABLESPACE` or `ALTER TABLE ... TABLESPACE` syntax are not affected by the `innodb_file_per_table` setting. These tables must be moved individually.

15.6.3.3 General Tablespaces

A general tablespace is a shared `InnoDB` tablespace that is created using `CREATE TABLESPACE` syntax. General tablespace capabilities and features are described under the following topics in this section:

- General Tablespace Capabilities
- Creating a General Tablespace
- Adding Tables to a General Tablespace
- General Tablespace Row Format Support
- Moving Tables Between Tablespaces Using ALTER TABLE
- Renaming a General Tablespace
- Dropping a General Tablespace
- General Tablespace Limitations

General Tablespace Capabilities

The general tablespace feature provides the following capabilities:

- Similar to the system tablespace, general tablespaces are shared tablespaces that can store data for multiple tables.
- General tablespaces have a potential memory advantage over file-per-table tablespaces. The server keeps tablespace metadata in memory for the lifetime of a tablespace. Multiple tables in fewer general tablespaces consume less memory for tablespace metadata than the same number of tables in separate file-per-table tablespaces.
- General tablespace data files may be placed in a directory relative to or independent of the MySQL data directory, which provides you with many of the data file and storage management capabilities of file-per-table tablespaces. As with file-per-table tablespaces, the ability to place data files outside of the MySQL data directory allows you to manage performance of critical tables separately, setup RAID or DRBD for specific tables, or bind tables to particular disks, for example.

- General tablespaces support both Antelope and Barracuda file formats, and therefore support all table row formats and associated features. With support for both file formats, general tablespaces have no dependence on `innodb_file_format` or `innodb_file_per_table` settings, nor do these variables have any effect on general tablespaces.
- The `TABLESPACE` option can be used with `CREATE TABLE` to create tables in a general tablespaces, file-per-table tablespace, or in the system tablespace.
- The `TABLESPACE` option can be used with `ALTER TABLE` to move tables between general tablespaces, file-per-table tablespaces, and the system tablespace. Previously, it was not possible to move a table from a file-per-table tablespace to the system tablespace. With the general tablespace feature, you can now do so.

Creating a General Tablespace

General tablespaces are created using `CREATE TABLESPACE` syntax.

```
CREATE TABLESPACE tablespace_name
    [ADD DATAFILE 'file_name']
    [FILE_BLOCK_SIZE = value]
        [ENGINE [=] engine_name]
```

A general tablespace can be created in the data directory or outside of it. To avoid conflicts with implicitly created file-per-table tablespaces, creating a general tablespace in a subdirectory under the data directory is not supported. When creating a general tablespace outside of the data directory, the directory must exist and must be known to `InnoDB` prior to creating the tablespace. To make an unknown directory known to `InnoDB`, add the directory to the `innodb_directories` argument value. `innodb_directories` is a read-only startup option. Configuring it requires restarting the server.

Examples:

Creating a general tablespace in the data directory:

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' Engine=InnoDB;
```

or

```
mysql> CREATE TABLESPACE `ts1` Engine=InnoDB;
```

The `ADD DATAFILE` clause is optional as of MySQL 8.0.14 and required before that. If the `ADD DATAFILE` clause is not specified when creating a tablespace, a tablespace data file with a unique file name is created implicitly. The unique file name is a 128 bit UUID formatted into five groups of hexadecimal numbers separated by dashes (*aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee*). General tablespace data files include an `.ibd` file extension. In a replication environment, the data file name created on the master is not the same as the data file name created on the slave.

Creating a general tablespace in a directory outside of the data directory:

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE '/my/tablespace/directory/ts1.ibd' Engine=InnoDB;
```

You can specify a path that is relative to the data directory as long as the tablespace directory is not under the data directory. In this example, the `my_tablespace` directory is at the same level as the data directory:

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE '../my_tablespace/ts1.ibd' Engine=InnoDB;
```

Note

The `ENGINE = InnoDB` clause must be defined as part of the `CREATE TABLESPACE` statement, or `InnoDB` must be defined as the default storage engine (`default_storage_engine=InnoDB`).

Adding Tables to a General Tablespace

After creating an `InnoDB` general tablespace, you can use `CREATE TABLE tbl_name ... TABLESPACE [=] tablespace_name` or `ALTER TABLE tbl_name TABLESPACE [=] tablespace_name` to add tables to the tablespace, as shown in the following examples:

`CREATE TABLE`:

```
mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1;
```

`ALTER TABLE`:

```
mysql> ALTER TABLE t2 TABLESPACE ts1;
```

Note

Support for adding table partitions to shared tablespaces was deprecated in MySQL 5.7.24 and removed in MySQL 8.0.13. Shared tablespaces include the `InnoDB` system tablespace and general tablespaces.

For detailed syntax information, see `CREATE TABLE` and `ALTER TABLE`.

General Tablespace Row Format Support

General tablespaces support all table row formats (`REDUNDANT`, `COMPACT`, `DYNAMIC`, `COMPRESSED`) with the caveat that compressed and uncompressed tables cannot coexist in the same general tablespace due to different physical page sizes.

For a general tablespace to contain compressed tables (`ROW_FORMAT=COMPRESSED`), `FILE_BLOCK_SIZE` must be specified, and the `FILE_BLOCK_SIZE` value must be a valid compressed page size in relation to the `innodb_page_size` value. Also, the physical page size of the compressed table (`KEY_BLOCK_SIZE`) must be equal to `FILE_BLOCK_SIZE/1024`. For example, if `innodb_page_size=16KB` and `FILE_BLOCK_SIZE=8K`, the `KEY_BLOCK_SIZE` of the table must be 8.

The following table shows permitted `innodb_page_size`, `FILE_BLOCK_SIZE`, and `KEY_BLOCK_SIZE` combinations. `FILE_BLOCK_SIZE` values may also be specified in bytes. To determine a valid `KEY_BLOCK_SIZE` value for a given `FILE_BLOCK_SIZE`, divide the `FILE_BLOCK_SIZE` value by 1024. Table compression is not support for 32K and 64K `InnoDB` page sizes. For more information about `KEY_BLOCK_SIZE`, see `CREATE TABLE`, and Section 15.9.1.2, “Creating Compressed Tables”.

Table 15.4 Permitted Page Size, FILE_BLOCK_SIZE, and KEY_BLOCK_SIZE Combinations for Compressed Tables

InnoDB Page Size (innodb_page_size)	Permitted FILE_BLOCK_SIZE Value	Permitted KEY_BLOCK_SIZE Value
64KB	64K (65536)	Compression is not supported
32KB	32K (32768)	Compression is not supported
16KB	16K (16384)	N/A: If `innodb_page_size` is equal to `FILE_BLOCK_SIZE`,

InnoDB Page Size (innodb_page_size)	Permitted FILE_BLOCK_SIZE Value	Permitted KEY_BLOCK_SIZE Value
		the tablespace cannot contain a compressed table.
16KB	8K (8192)	8
16KB	4K (4096)	4
16KB	2K (2048)	2
16KB	1K (1024)	1
8KB	8K (8192)	N/A: If innodb_page_size is equal to FILE_BLOCK_SIZE, the tablespace cannot contain a compressed table.
8KB	4K (4096)	4
8KB	2K (2048)	2
8KB	1K (1024)	1
4KB	4K (4096)	N/A: If innodb_page_size is equal to FILE_BLOCK_SIZE, the tablespace cannot contain a compressed table.
4KB	2K (2048)	2
4KB	1K (1024)	1

This example demonstrates creating a general tablespace and adding a compressed table. The example assumes a default innodb_page_size of 16KB. The FILE_BLOCK_SIZE of 8192 requires that the compressed table have a KEY_BLOCK_SIZE of 8.

```
mysql> CREATE TABLESPACE `ts2` ADD DATAFILE 'ts2.ibd' FILE_BLOCK_SIZE = 8192 Engine=InnoDB;

mysql> CREATE TABLE t4 (c1 INT PRIMARY KEY) TABLESPACE ts2 ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8;
```

If you do not specify FILE_BLOCK_SIZE when creating a general tablespace, FILE_BLOCK_SIZE defaults to innodb_page_size. When FILE_BLOCK_SIZE is equal to innodb_page_size, the tablespace may only contain tables with an uncompressed row format (COMPACT, REDUNDANT, and DYNAMIC row formats).

Moving Tables Between Tablespaces Using ALTER TABLE

You can use ALTER TABLE with the TABLESPACE option to move a table to an existing general tablespace, to a new file-per-table tablespace, or to the system tablespace.

Note

Support for placing table partitions in shared tablespaces was deprecated in MySQL 5.7.24 and removed MySQL 8.0.13. Shared tablespaces include the InnoDB system tablespace and general tablespaces.

To move a table from a file-per-table tablespace or from the system tablespace to a general tablespace, specify the name of the general tablespace. The general tablespace must exist. See CREATE TABLESPACE for more information.

```
ALTER TABLE tbl_name TABLESPACE [=] tablespace_name;
```

To move a table from a general tablespace or file-per-table tablespace to the system tablespace, specify `innodb_system` as the tablespace name.

```
ALTER TABLE tbl_name TABLESPACE [=] innodb_system;
```

To move a table from the system tablespace or a general tablespace to a file-per-table tablespace, specify `innodb_file_per_table` as the tablespace name.

```
ALTER TABLE tbl_name TABLESPACE [=] innodb_file_per_table;
```

`ALTER TABLE ... TABLESPACE` operations always cause a full table rebuild, even if the `TABLESPACE` attribute has not changed from its previous value.

`ALTER TABLE ... TABLESPACE` syntax does not support moving a table from a temporary tablespace to a persistent tablespace.

The `DATA DIRECTORY` clause is permitted with `CREATE TABLE ... TABLESPACE=innodb_file_per_table` but is otherwise not supported for use in combination with the `TABLESPACE` option.

Restrictions apply when moving tables from encrypted tablespaces. See Encryption Limitations.

Renaming a General Tablespace

Renaming a general tablespace is supported using `ALTER TABLESPACE ... RENAME TO` syntax.

```
ALTER TABLESPACE s1 RENAME TO s2;
```

The `CREATE TABLESPACE` privilege is required to rename a general tablespace.

`RENAME TO` operations are implicitly performed in `autocommit` mode, regardless of the `autocommit` setting.

A `RENAME TO` operation cannot be performed while `LOCK TABLES` or `FLUSH TABLES WITH READ LOCK` is in effect for tables that reside in the tablespace.

Exclusive metadata locks are taken on tables within a general tablespace while the tablespace is renamed, which prevents concurrent DDL. Concurrent DML is supported.

Dropping a General Tablespace

The `DROP TABLESPACE` statement is used to drop an `InnoDB` general tablespace.

All tables must be dropped from the tablespace prior to a `DROP TABLESPACE` operation. If the tablespace is not empty, `DROP TABLESPACE` returns an error.

Use a query similar to the following to identify tables in a general tablespace.

```
mysql> SELECT a.NAME AS space_name, b.NAME AS table_name FROM INFORMATION_SCHEMA.INNODB_TABLESPACES a,
       INFORMATION_SCHEMA.INNODB_TABLES b WHERE a.SPACE=b.SPACE AND a.NAME LIKE 'ts1';
+------------+------------+
| space_name | table_name |
+------------+------------+
| ts1        | test/t1    |
| ts1        | test/t2    |
| ts1        | test/t3    |
+------------+------------+
```

A general `InnoDB` tablespace is not deleted automatically when the last table in the tablespace is dropped. The tablespace must be dropped explicitly using `DROP TABLESPACE` *`tablespace_name`*.

A general tablespace does not belong to any particular database. A `DROP DATABASE` operation can drop tables that belong to a general tablespace but it cannot drop the tablespace, even if the `DROP DATABASE` operation drops all tables that belong to the tablespace. A general tablespace must be dropped explicitly using `DROP TABLESPACE tablespace_name`.

Similar to the system tablespace, truncating or dropping tables stored in a general tablespace creates free space internally in the general tablespace .ibd data file which can only be used for new `InnoDB` data. Space is not released back to the operating system as it is when a file-per-table tablespace is deleted during a `DROP TABLE` operation.

This example demonstrates how to drop an `InnoDB` general tablespace. The general tablespace `ts1` is created with a single table. The table must be dropped before dropping the tablespace.

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' Engine=InnoDB;

mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts10 Engine=InnoDB;

mysql> DROP TABLE t1;

mysql> DROP TABLESPACE ts1;
```

Note

tablespace_name is a case-sensitive identifier in MySQL.

General Tablespace Limitations

- A generated or existing tablespace cannot be changed to a general tablespace.
- Creation of temporary general tablespaces is not supported.
- General tablespaces do not support temporary tables.
- Similar to the system tablespace, truncating or dropping tables stored in a general tablespace creates free space internally in the general tablespace .ibd data file which can only be used for new `InnoDB` data. Space is not released back to the operating system as it is for file-per-table tablespaces.

  Additionally, a table-copying `ALTER TABLE` operation on table that resides in a shared tablespace (a general tablespace or the system tablespace) can increase the amount of space used by the tablespace. Such operations require as much additional space as the data in the table plus indexes. The additional space required for the table-copying `ALTER TABLE` operation is not released back to the operating system as it is for file-per-table tablespaces.
- `ALTER TABLE ... DISCARD TABLESPACE` and `ALTER TABLE ...IMPORT TABLESPACE` are not supported for tables that belong to a general tablespace.
- Support for placing table partitions in general tablespaces was deprecated in MySQL 5.7.24 and removed in MySQL 8.0.13.

15.6.3.4 Undo Tablespaces

Undo tablespaces contain undo logs, which are collections of undo log records that contain information about how to undo the latest change by a transaction to a clustered index record. Undo logs exist within undo log segments, which are contained within rollback segments. The `innodb_rollback_segments` variable defines the number of rollback segments allocated to each undo tablespace.

Two default undo tablespaces are created when the MySQL instance is initialized. Default undo tablespaces are created at initialization time to provide a location for rollback segments that must exist

before SQL statements can be accepted. A minimum of two undo tablespaces is required to support automated truncation of undo tablespaces. See Truncating Undo Tablespaces.

Default undo tablespaces are created in the location defined by the `innodb_undo_directory` variable. If the `innodb_undo_directory` variable is undefined, default undo tablespaces are created in the data directory. Default undo tablespace data files are named `undo_001` and `undo_002`. The corresponding undo tablespace names defined in the data dictionary are `innodb_undo_001` and `innodb_undo_002`.

As of MySQL 8.0.14, additional undo tablespaces can be created at runtime using SQL. See Adding Undo Tablespaces.

The initial size of an undo tablespace data file depends on the `innodb_page_size` value. For the default 16KB page size, the initial undo tablespace file size is 10MiB. For 4KB, 8KB, 32KB, and 64KB page sizes, the initial undo tablespace files sizes are 7MiB, 8MiB, 20MiB, and 40MiB, respectively.

Adding Undo Tablespaces

Because undo logs can become large during long-running transactions, creating additional undo tablespaces can help prevent individual undo tablespaces from becoming too large. As of MySQL 8.0.14, additional undo tablespaces can be created at runtime using `CREATE UNDO TABLESPACE` syntax.

```
CREATE UNDO TABLESPACE tablespace_name ADD DATAFILE 'file_name.ibu'
```

The undo tablespace file name must have an `.ibu` extension. It is not permitted to specify a relative path when defining the undo tablespace file name. A fully qualified path is permitted, but the path must be known to `InnoDB`. Known paths are those defined by the `innodb_directories` variable. Unique undo tablespace file names are recommended to avoid potential file name conflicts when moving or cloning data.

At startup, directories defined by the `innodb_directories` variable are scanned for undo tablespace files. (The scan also traverses subdirectories.) Directories defined by the `innodb_data_home_dir`, `innodb_undo_directory`, and `datadir` variables are automatically appended to the `innodb_directories` value, regardless of whether the `innodb_directories` variable is defined explicitly. An undo tablespace can therefore reside in paths defined by any of those variables.

If the undo tablespace file name does not include a path, the undo tablespace is created in the directory defined by the `innodb_undo_directory` variable. If that variable is undefined, the undo tablespace is created in the data directory.

Note

The `InnoDB` recovery process requires that undo tablespace files reside in known directories. Undo tablespace files must be discovered and opened before redo recovery and before other data files are opened to permit uncommitted transactions and data dictionary changes to be rolled back. An undo tablespace not found before recovery cannot be used, which can cause database inconsistencies. An error message is reported at startup if an undo tablespace known to the data dictionary is not found. The known directory requirement also supports undo tablespace portability. See Moving Undo Tablespaces.

To create undo tablespaces in a path relative to the data directory, set the `innodb_undo_directory` variable to the relative path, and specify the file name only when creating an undo tablespace.

To view undo tablespace names and paths, query `INFORMATION_SCHEMA.FILES`:

```
SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES
  WHERE FILE_TYPE LIKE 'UNDO LOG';
```

A MySQL instance supports up to 127 undo tablespaces including the two default undo tablespaces created when the MySQL instance is initialized.

Note

Prior to MySQL 8.0.14, additional undo tablespaces are created by configuring the `innodb_undo_tablespaces` startup variable. This variable is deprecated and no longer configurable as of MySQL 8.0.14.

Prior to MySQL 8.0.14, increasing the `innodb_undo_tablespaces` setting creates the specified number of undo tablespaces and adds them to the list of active undo tablespaces. Decreasing the `innodb_undo_tablespaces` setting removes undo tablespaces from the list of active undo tablespaces. Undo tablespaces that are removed from the active list remain active until they are no longer used by existing transactions. The `innodb_undo_tablespaces` variable can be configured at runtime using a `SET` statement or defined in a configuration file.

Prior to MySQL 8.0.14, deactivated undo tablespaces cannot be removed. Manual removal of undo tablespace files is possible after a slow shutdown but is not recommended, as deactivated undo tablespaces may contain active undo logs for some time after the server is restarted if open transactions were present when shutting down the server. As of MySQL 8.0.14, undo tablespaces can be dropped using `DROP UNDO TABALESPACE` syntax. See Dropping Undo Tablespaces.

Dropping Undo Tablespaces

As of MySQL 8.0.14, undo tablespaces created using `CREATE UNDO TABLESPACE` syntax can be dropped at runtime using `DROP UNDO TABALESPACE` syntax.

An undo tablespace must be empty before it can be dropped. To empty an undo tablespace, the undo tablespace must first be marked as inactive using `ALTER UNDO TABLESPACE` syntax so that the tablespace is no longer used for assigning rollback segments to new transactions.

```
ALTER UNDO TABLESPACE tablespace_name SET INACTIVE;
```

After an undo tablespace is marked as inactive, transactions currently using rollback segments in the undo tablespace are permitted to finish, as are any transactions started before those transactions are completed. After transactions are completed, the purge system frees the rollback segments in the undo tablespace, and the undo tablespace is truncated to its initial size. (The same process is used when truncating undo tablespaces. See Truncating Undo Tablespaces.) When the undo tablespace is empty, it can be dropped.

```
DROP UNDO TABLESPACE tablespace_name;
```

Note

Alternatively, the undo tablespace can be left in an empty state and reactivated later, when needed, by issuing an `ALTER UNDO TABLESPACE tablespace_name SET ACTIVE` statement.

The state of an undo tablespace can be monitored by querying the `INFORMATION_SCHEMA.INNODB_TABLESPACES` table.

```
SELECT NAME, STATE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES
  WHERE NAME LIKE tablespace_name;
```

An `inactive` state indicates that rollback segments in an undo tablespace are no longer used by new transactions. An `empty` state indicates that an undo tablespace is empty and ready to be dropped, or made active again using an `ALTER UNDO TABLESPACE tablespace_name SET ACTIVE` statement. Attempting to drop an undo tablespace that is not empty returns an error.

The default undo tablespaces (`innodb_undo_001` and `innodb_undo_002`) created when the MySQL instance is initialized cannot be dropped. They can, however, be made inactive using an `ALTER UNDO TABLESPACE tablespace_name SET INACTIVE` statement. Before a default undo tablespace can be made inactive, there must be an undo tablespace to take its place. A minimum of two active undo tablespaces are required at all times to support automated truncation of undo tablespaces.

Moving Undo Tablespaces

Undo tablespaces created with `CREATE UNDO TABLESPACE` syntax can be moved while the server is offline to any known directory. Known directories are those defined by the `innodb_directories` variable. Directories defined by `innodb_data_home_dir`, `innodb_undo_directory`, and `datadir` are automatically appended to the `innodb_directories` value regardless of whether the `innodb_directories` variable is defined explicitly. Those directories and their subdirectories are scanned at startup for undo tablespaces files. An undo tablespace file moved to any of those directories is discovered at startup and assumed to be the undo tablespace that was moved.

The default undo tablespaces (`innodb_undo_001` and `innodb_undo_002`) created when the MySQL instance is initialized must always reside in the directory defined by the `innodb_undo_directory` variable. If the `innodb_undo_directory` variable is undefined, default undo tablespaces reside in the data directory. If default undo tablespaces are moved while the server is offline, the server must be started with the `innodb_undo_directory` variable configured to the new directory.

The I/O patterns for undo logs make undo tablespaces good candidates for SSD storage.

Configuring the Number of Rollback Segments

The `innodb_rollback_segments` variable defines the number of rollback segments allocated to each undo tablespace and to the global temporary tablespace. The `innodb_rollback_segments` variable can be configured at startup or while the server is running.

The default setting for `innodb_rollback_segments` is 128, which is also the maximum value. For information about the number of transactions that a rollback segment supports, see Section 15.6.6, “Undo Logs”.

Truncating Undo Tablespaces

There are two methods of truncating undo tablespaces, which can be used individually or in combination to manage undo tablespace size. One method is automated, enabled using configuration variables. The other method is manual, performed using SQL statements.

The automated method does not require monitoring undo tablespace size and, once enabled, it performs deactivation, truncation, and reactivation of undo tablespaces without manual intervention. The manual truncation method may be preferable if you want to control when undo tablespaces are taken offline for truncation. For example, you may want to avoid truncating undo tablespaces during peak workload times.

Automated Truncation

Automated truncation of undo tablespaces requires a minimum of two active undo tablespaces, which ensures that one undo tablespace remains active while the other is taken offline to be truncated. By default, two undo tablespaces are created when the MySQL instance is initialized.

To have undo tablespaces automatically truncated, enable the `innodb_undo_log_truncate` variable. For example:

```
mysql> SET GLOBAL innodb_undo_log_truncate=ON;
```

When the `innodb_undo_log_truncate` variable is enabled, undo tablespaces that exceed the size limit defined by the `innodb_max_undo_log_size` variable are subject to truncation. The `innodb_max_undo_log_size` variable is dynamic and has a default value of 1073741824 bytes (1024 MiB).

```
mysql> SELECT @@innodb_max_undo_log_size;
+----------------------------+
| @@innodb_max_undo_log_size |
+----------------------------+
|                 1073741824 |
+----------------------------+
```

When the `innodb_undo_log_truncate` variable is enabled:

1. Default and user-defined undo tablespaces that exceed the `innodb_max_undo_log_size` setting are marked for truncation. Selection of an undo tablespace for truncation is performed in a circular fashion to avoid truncating the same undo tablespace each time.

2. Rollback segments residing in the selected undo tablespace are made inactive so that they are not assigned to new transactions. Existing transactions that are currently using rollback segments are permitted to finish.

3. The purge system frees rollback segments that are no longer in use.

4. After all rollback segments in the undo tablespace are freed, the truncate operation runs and truncates the undo tablespace to its initial size. The initial size of an undo tablespace depends on the `innodb_page_size` value. For the default 16KB page size, the initial undo tablespace file size is 10MiB. For 4KB, 8KB, 32KB, and 64KB page sizes, the initial undo tablespace files sizes are 7MiB, 8MiB, 20MiB, and 40MiB, respectively.

   The size of an undo tablespace after a truncate operation may be larger than the initial size due to immediate use following the completion of the operation.

   The `innodb_undo_directory` variable defines the location of default undo tablespace files. If the `innodb_undo_directory` variable is undefined, default undo tablespaces reside in the data directory. The location of all undo tablespace files including user-defined undo tablespaces created using `CREATE UNDO TABLESPACE` syntax can be determined by querying the `INFORMATION_SCHEMA.FILES` table:

   ```
   SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES WHERE FILE_TYPE LIKE 'UNDO LOG';
   ```

5. Rollback segments are reactivated so that they can be assigned to new transactions.

Manual Truncation

Manual truncation of undo tablespaces requires a minimum of three active undo tablespaces. Two active undo tablespaces are required at all times to support the possibility that automated truncation is enabled. A minimum of three undo tablespaces satisfies this requirement while permitting an undo tablespace to be taken offline manually.

To manually initiate truncation of an undo tablespace, deactivate the undo tablespace by issuing the following statement:

```
ALTER UNDO TABLESPACE tablespace_name SET INACTIVE;
```

After the undo tablespace is marked as inactive, transactions currently using rollback segments in the undo tablespace are permitted to finish, as are any transactions started before those transactions are completed. After transactions are completed, the purge system frees the rollback segments in the undo tablespace, the undo tablespace is truncated to its initial size, and the undo tablespace state changes from `inactive` to `empty`.

Note

When an `ALTER UNDO TABLESPACE tablespace_name SET INACTIVE` statement deactivates an undo tablespace, the purge thread looks for that undo tablespaces at the next opportunity. Once the undo tablespace is found and marked for truncation, the purge thread returns with increased frequency to quickly empty and truncate the undo tablespace.

To check the state of an undo tablespace, query the `INFORMATION_SCHEMA.INNODB_TABLESPACES` table.

```
SELECT NAME, STATE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES
  WHERE NAME LIKE tablespace_name;
```

Once the undo tablespace is in an `empty` state, it can be reactivated by issuing the following statement:

```
ALTER UNDO TABLESPACE tablespace_name SET ACTIVE;
```

An undo tablespace in an `empty` state can also be dropped. See Dropping Undo Tablespaces.

Expediting Automated Truncation of Undo Tablespaces

The purge thread is responsible for emptying and truncating undo tablespaces. By default, the purge thread looks for undo tablespaces to truncate once every 128 times that purge is invoked. The frequency with which the purge thread looks for undo tablespaces to truncate is controlled by the `innodb_purge_rseg_truncate_frequency` variable, which has a default setting of 128.

```
mysql> SELECT @@innodb_purge_rseg_truncate_frequency;
+----------------------------------------+
| @@innodb_purge_rseg_truncate_frequency |
+----------------------------------------+
|                                    128 |
+----------------------------------------+
```

To increase that frequency, decrease the `innodb_purge_rseg_truncate_frequency` setting. For example, to have the purge thread look for undo tabespaces once every 32 timees that purge is invoked, set `innodb_purge_rseg_truncate_frequency` to 32.

```
mysql> SET GLOBAL innodb_purge_rseg_truncate_frequency=32;
```

When the purge thread finds an undo tablespace that requires truncation, the purge thread returns with increased frequency to quickly empty and truncate the undo tablespace.

Performance Impact of Truncating Undo Tablespace Files

When an undo tablespace is truncated, the rollback segments in the undo tablespace are deactivated. The active rollback segments in other undo tablespaces assume responsibility for the entire system load, which

may result in a slight performance degradation. The amount of performance degradation depends on a number of factors:

- Number of undo tablespaces
- Number of undo logs
- Undo tablespace size
- Speed of the I/O susbsystem
- Existing long running transactions
- System load

The easiest way to avoid impacting performance when truncating undo tablespaces is to increase the number of undo tablespaces.

Monitoring Undo Tablespace Truncation

As of MySQL 8.0.16, `undo` and `purge` susbsystem counters are provided for monitoring background activities associated with undo log truncation. For counter names and descriptions, query the `INFORMATION_SCHEMA.INNODB_METRICS` table.

```
SELECT NAME, SUBSYSTEM, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME LIKE '%truncate%';
```

For information about enabling counters and querying counter data, see Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

15.6.3.5 Temporary Tablespaces

`InnoDB` uses session temporary tablespaces and a global temporary tablespace.

Session Temporary Tablespaces

Session temporary tablespaces store user-created temporary tables and internal temporary tables created by the optimizer when `InnoDB` is configured as the storage engine for on-disk internal temporary tables. Beginning with MySQL 8.0.16, the storage engine used for on-disk internal temporary tables is always `InnoDB`. (Previously, the storage engine was determined by the value of `internal_tmp_disk_storage_engine`.)

Session temporary tablespaces are allocated to a session from a pool of temporary tablespaces on the first request to create an on-disk temporary table. A maximum of two tablespaces is allocated to a session, one for user-created temporary tables and the other for internal temporary tables created by the optimizer. The temporary tablespaces allocated to a session are used for all on-disk temporary tables created by the session. When a session disconnects, its temporary tablespaces are truncated and released back to the pool. A pool of 10 temporary tablespaces is created when the server is started. The size of the pool never shrinks and tablespaces are added to the pool automatically as necessary. The pool of temporary tablespaces is removed on normal shutdown or on an aborted initialization. Session temporary tablespace files are five pages in size when created and have an `.ibt` file name extension.

A range of 400 thousand space IDs is reserved for session temporary tablespaces. Because the pool of session temporary tablespaces is recreated each time the server is started, space IDs for session temporary tablespaces are not persisted when the server is shut down and may be reused.

The `innodb_temp_tablespaces_dir` variable defines the location where session temporary tablespaces are created. The default location is the `#innodb_temp` directory in the data directory. Startup is refused if the pool of temporary tablespaces cannot be created.

```
shell> cd BASEDIR/data/#innodb_temp
shell> ls
temp_10.ibt  temp_2.ibt  temp_4.ibt  temp_6.ibt  temp_8.ibt
temp_1.ibt   temp_3.ibt  temp_5.ibt  temp_7.ibt  temp_9.ibt
```

In statement based replication (SBR) mode, temporary tables created on a slave reside in a single session temporary tablespace that is truncated only when the MySQL server is shut down.

The `INNODB_SESSION_TEMP_TABLESPACES` table provides metadata about session temporary tablespaces.

The `INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO` table provides metadata about user-created temporary tables that are active in an `InnoDB` instance.

Global Temporary Tablespace

The global temporary tablespace (`ibtmp1`) stores rollback segments for changes made to user-created temporary tables.

The `innodb_temp_data_file_path` variable defines the relative path, name, size, and attributes for global temporary tablespace data files. If no value is specified for `innodb_temp_data_file_path`, the default behavior is to create a single auto-extending data file named `ibtmp1` in the `innodb_data_home_dir` directory. The initial file size is slightly larger than 12MB.

The global temporary tablespace is removed on normal shutdown or on an aborted initialization, and recreated each time the server is started. The global temporary tablespace receives a dynamically generated space ID when it is created. Startup is refused if the global temporary tablespace cannot be created. The global temporary tablespace is not removed if the server halts unexpectedly. In this case, a database administrator can remove the global temporary tablespace manually or restart the MySQL server. Restarting the MySQL server removes and recreates the global temporary tablespace automatically.

The global temporary tablespace cannot reside on a raw device.

`INFORMATION_SCHEMA.FILES` provides metadata about the global temporary tablespace. Issue a query similar to this one to view global temporary tablespace metadata:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.FILES WHERE TABLESPACE_NAME='innodb_temporary'\G
```

By default, the global temporary tablespace data file is autoextending and increases in size as necessary.

To determine if a global temporary tablespace data file is autoextending, check the `innodb_temp_data_file_path` setting:

```
mysql> SELECT @@innodb_temp_data_file_path;
+-----------------------------+
| @@innodb_temp_data_file_path |
+-----------------------------+
| ibtmp1:12M:autoextend        |
+-----------------------------+
```

To check the size of global temporary tablespace data files, query the `INFORMATION_SCHEMA.FILES` table using a query similar to this one:

```
mysql> SELECT FILE_NAME, TABLESPACE_NAME, ENGINE, INITIAL_SIZE, TOTAL_EXTENTS*EXTENT_SIZE
       AS TotalSizeBytes, DATA_FREE, MAXIMUM_SIZE FROM INFORMATION_SCHEMA.FILES
       WHERE TABLESPACE_NAME = 'innodb_temporary'\G
*************************** 1. row ***************************
```

```
      FILE_NAME: ./ibtmp1
TABLESPACE_NAME: innodb_temporary
         ENGINE: InnoDB
   INITIAL_SIZE: 12582912
 TotalSizeBytes: 12582912
      DATA_FREE: 6291456
   MAXIMUM_SIZE: NULL
```

`TotalSizeBytes` shows the current size of the global temporary tablespace data file. For information about other field values, see Section 25.11, “The INFORMATION_SCHEMA FILES Table”.

Alternatively, check the global temporary tablespace data file size on your operating system. The global temporary tablespace data file is located in the directory defined by the `innodb_temp_data_file_path` variable.

To reclaim disk space occupied by a global temporary tablespace data file, restart the MySQL server. Restarting the server removes and recreates the global temporary tablespace data file according to the attributes defined by `innodb_temp_data_file_path`.

To limit the size of the global temporary tablespace data file, configure `innodb_temp_data_file_path` to specify a maximum file size. For example:

```
[mysqld]
innodb_temp_data_file_path=ibtmp1:12M:autoextend:max:500M
```

Configuring `innodb_temp_data_file_path` requires restarting the server.

15.6.3.6 Creating a Tablespace Outside of the Data Directory

The `CREATE TABLE ... DATA DIRECTORY` clause permits creating a file-per-table tablespace outside of the data directory. For example, you can use the `DATA DIRECTORY` clause to create a tablespace on a separate storage device with particular performance or capacity characteristics, such as a fast SSD or a high-capacity HDD.

Be sure of the location that you choose. The `DATA DIRECTORY` clause cannot be used with `ALTER TABLE` to change the location later.

The tablespace data file is created in the specified directory, within in a subdirectory named for the schema to which the table belongs.

The following example demonstrates creating a file-per-table tablespace outside of the data directory. It is assumed that the `innodb_file_per_table` variable is enabled.

```
mysql> USE test;
Database changed

mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) DATA DIRECTORY = '/remote/directory';

# MySQL creates the tablespace file in a subdirectory that is named
# for the schema to which the table belongs

shell> cd /remote/directory/test
shell> ls
t1.ibd
```

When creating a tablespace outside of the data directory, ensure that the directory is known to `InnoDB`. Otherwise, if the server halts unexpectedly before tablespace data file pages are fully flushed, startup fails when the tablespace is not found during the pre-recovery discovery phase that searches known directories for tablespace data files (see Tablespace Discovery During Crash Recovery). To make a directory known,

add it to the `innodb_directories` argument value. `innodb_directories` is a read-only startup option that defines directories to scan at startup for tablespace data files. Configuring it requires restarting the server.

`CREATE TABLE ... TABLESPACE` syntax can also be used in combination with the `DATA DIRECTORY` clause to create a file-per-table tablespace outside of the data directory. To do so, specify `innodb_file_per_table` as the tablespace name.

```
mysql> CREATE TABLE t2 (c1 INT PRIMARY KEY) TABLESPACE = innodb_file_per_table
       DATA DIRECTORY = '/remote/directory';
```

The `innodb_file_per_table` variable does not need to be enabled when using this method.

Usage Notes:

- MySQL initially holds the tablespace data file open, preventing you from dismounting the device, but might eventually close the table if the server is busy. Be careful not to accidentally dismount an external device while MySQL is running, or start MySQL while the device is disconnected. Attempting to access a table when the associated tablespace data file is missing causes a serious error that requires a server restart.

  A server restart issues errors and warnings if the tablespace data file is not at the expected path. In this case, you can restore the tablespace data file from a backup or drop the table to remove the information about it from the data dictionary.

- Before placing a tablespace on an NFS-mounted volume, review potential issues outlined in Using NFS with MySQL.

- If using an LVM snapshot, file copy, or other file-based mechanism to back up the tablespace data file, always use the `FLUSH TABLES ... FOR EXPORT` statement first to ensure that all changes buffered in memory are flushed to disk before the backup occurs.

- Using the `DATA DIRECTORY` clause is an alternative to using symbolic links, which is not supported.

15.6.3.7 Copying Tablespaces to Another Instance

This section describes how to copy a file-per-table tablespaces from one MySQL instance to another, otherwise known as the Transportable Tablespaces feature. This feature also supports partitioned `InnoDB` tables and individual `InnoDB` table partitions and subpartitions.

For information about other `InnoDB` table copying methods, see Section 15.6.1.2, “Moving or Copying InnoDB Tables”.

There are many reasons why you might copy an `InnoDB` file-per-table tablespace to a different instance:

- To run reports without putting extra load on a production server.

- To set up identical data for a table on a new slave server.

- To restore a backed-up version of a table or partition after a problem or mistake.

- As a faster way of moving data around than importing the results of a `mysqldump` command. The data is available immediately, rather than having to be re-inserted and the indexes rebuilt.

- To move a file-per-table tablespace to a server with storage medium that better suits system requirements. For example, you may want to have busy tables on an SSD device, or large tables on a high-capacity HDD device.

Limitations and Usage Notes

- The tablespace copy procedure is only possible when `innodb_file_per_table` is enabled, which is the default setting. Tables residing in the shared system tablespace cannot be quiesced.
- When a table is quiesced, only read-only transactions are allowed on the affected table.
- When importing a tablespace, the page size must match the page size of the importing instance.
- `ALTER TABLE ... DISCARD TABLESPACE` is supported for partitioned `InnoDB` tables, and `ALTER TABLE ... DISCARD PARTITION ... TABLESPACE` is supported for `InnoDB` table partitions.
- `DISCARD TABLESPACE` is not supported for tablespaces with a parent-child (primary key-foreign key) relationship when `foreign_key_checks` is set to `1`. Before discarding a tablespace for parent-child tables, set `foreign_key_checks=0`. Partitioned `InnoDB` tables do not support foreign keys.
- `ALTER TABLE ... IMPORT TABLESPACE` does not enforce foreign key constraints on imported data. If there are foreign key constraints between tables, all tables should be exported at the same (logical) point in time. Partitioned `InnoDB` tables do not support foreign keys.
- `ALTER TABLE ... IMPORT TABLESPACE` and `ALTER TABLE ... IMPORT PARTITION ... TABLESPACE` do not require a `.cfg` metadata file to import a tablespace. However, metadata checks are not performed when importing without a `.cfg` file, and a warning similar to the following is issued:

  ```
  Message: InnoDB: IO Read error: (2, No such file or directory) Error opening '.\
  test\t.cfg', will attempt to import without schema verification
  1 row in set (0.00 sec)
  ```

  The ability to import without a `.cfg` file may be more convenient when no schema mismatches are expected. Additionally, the ability to import without a `.cfg` file could be useful in crash recovery scenarios in which metadata cannot be collected from an `.ibd` file.

  If no `.cfg` file is used, `InnoDB` uses the equivalent of a `SELECT MAX(ai_col) FROM table_name FOR UPDATE` statement to initialize the in-memory auto-increment counter that is used in assigning values for to an `AUTO_INCREMENT` column. Otherwise, the current maximum auto-increment counter value is read from the `.cfg` metadata file. For related information, see InnoDB AUTO_INCREMENT Counter Initialization.
- Due to a `.cfg` metadata file limitation, schema mismatches are not reported for partition type or partition definition differences when importing tablespace files for partitioned tables. Column differences are reported.
- When running `ALTER TABLE ... DISCARD PARTITION ... TABLESPACE` and `ALTER TABLE ... IMPORT PARTITION ... TABLESPACE` on subpartitioned tables, both partition and subpartition table names are allowed. When a partition name is specified, subpartitions of that partition are included in the operation.
- Importing a tablespace file from another MySQL server instance works if both instances have GA (General Availability) status and the server instance into which the file is imported is at the same or higher release level within the same release series. Importing a tablespace file into a server instance running an earlier release of MySQL is not supported.
- In replication scenarios, `innodb_file_per_table` must be set to `ON` on both the master and slave.
- On Windows, `InnoDB` stores database, tablespace, and table names internally in lowercase. To avoid import problems on case-sensitive operating systems such as Linux and UNIX, create all databases, tablespaces, and tables using lowercase names. A convenient way to accomplish this is to add the

following line to the [mysqld] section of your my.cnf or my.ini file before creating databases, tablespaces, or tables:

```
[mysqld]
lower_case_table_names=1
```

Note

It is prohibited to start the server with a lower_case_table_names setting that is different from the setting used when the server was initialized.

- ALTER TABLE ... DISCARD TABLESPACE and ALTER TABLE ...IMPORT TABLESPACE are not supported with tables that belong to an InnoDB general tablespace. For more information, see CREATE TABLESPACE.
- The default row format for InnoDB tables is configurable using the innodb_default_row_format configuration option. Attempting to import a table that does not explicitly define a row format (ROW_FORMAT), or that uses ROW_FORMAT=DEFAULT, could result in a schema mismatch error if the innodb_default_row_format setting on the source instance differs from the setting on the destination instance. For related information, see Defining the Row Format of a Table.
- When exporting an encrypted tablespace, InnoDB generates a .cfp file in addition to a .cfg metadata file. The .cfp file must be copied to the destination instance together with the .cfg file and tablespace file before performing the ALTER TABLE ... IMPORT TABLESPACE operation on the destination instance. The .cfp file contains a transfer key and an encrypted tablespace key. On import, InnoDB uses the transfer key to decrypt the tablespace key. For related information, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”.
- FLUSH TABLES ... FOR EXPORT is not supported on tables that have a FULLTEXT index. Full-text search auxiliary tables are not flushed. After importing a table with a FULLTEXT index, run OPTIMIZE TABLE to rebuild the FULLTEXT indexes. Alternatively, drop FULLTEXT indexes before the export operation and recreate them after importing the table on the destination instance.
- If the source table was created outside of the data directory by specifying the DATA DIRECTORY clause in the CREATE TABLE statement, the table that you replace on the destination instance must be defined with the same DATA DIRECTORY clause. A schema mismatch error is reported if the clauses do not match. To determine if the source table was defined with a DATA DIRECTORY clause, use SHOW CREATE TABLE to view the table definition. For information about using the DATA DIRECTORY clause, see Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”.

Transportable Tablespace Examples

Note

If you are transporting tables that are encrypted using the InnoDB tablespace encryption, see Limitations and Usage Notes before you begin for additional procedural information.

Example 1: Copying an InnoDB Table to Another Instance

This procedure demonstrates how to copy a regular InnoDB table from a running MySQL server instance to another running instance. The same procedure with minor adjustments can be used to perform a full table restore on the same instance.

1. On the source instance, create a table if one does not exist:

```
mysql> USE test;
mysql> CREATE TABLE t(c1 INT) ENGINE=InnoDB;
```

2. On the destination instance, create a table if one does not exist:

```
mysql> USE test;
mysql> CREATE TABLE t(c1 INT) ENGINE=InnoDB;
```

3. On the destination instance, discard the existing tablespace. (Before a tablespace can be imported, `InnoDB` must discard the tablespace that is attached to the receiving table.)

```
mysql> ALTER TABLE t DISCARD TABLESPACE;
```

4. On the source instance, run `FLUSH TABLES ... FOR EXPORT` to quiesce the table and create the `.cfg` metadata file:

```
mysql> USE test;
mysql> FLUSH TABLES t FOR EXPORT;
```

The metadata (`.cfg`) is created in the `InnoDB` data directory.

Note

The `FLUSH TABLES ... FOR EXPORT` statement ensures that changes to the named table have been flushed to disk so that a binary table copy can be made while the instance is running. When `FLUSH TABLES ... FOR EXPORT` is run, `InnoDB` produces a `.cfg` file in the same database directory as the table. The `.cfg` file contains metadata used for schema verification when importing the tablespace file.

5. Copy the `.ibd` file and `.cfg` metadata file from the source instance to the destination instance. For example:

```
shell> scp /path/to/datadir/test/t.{ibd,cfg} destination-server:/path/to/datadir/test
```

Note

The `.ibd` file and `.cfg` file must be copied before releasing the shared locks, as described in the next step.

6. On the source instance, use `UNLOCK TABLES` to release the locks acquired by `FLUSH TABLES ... FOR EXPORT`:

```
mysql> USE test;
mysql> UNLOCK TABLES;
```

7. On the destination instance, import the tablespace:

```
mysql> USE test;
mysql> ALTER TABLE t IMPORT TABLESPACE;
```

Note

The `ALTER TABLE ... IMPORT TABLESPACE` feature does not enforce foreign key constraints on imported data. If there are foreign key constraints

between tables, all tables should be exported at the same (logical) point in time. In this case you would stop updating the tables, commit all transactions, acquire shared locks on the tables, and then perform the export operation.

Example 2: Copying an InnoDB Partitioned Table to Another Instance

This procedure demonstrates how to copy a partitioned `InnoDB` table from a running MySQL server instance to another running instance. The same procedure with minor adjustments can be used to perform a full restore of a partitioned `InnoDB` table on the same instance.

1. On the source instance, create a partitioned table if one does not exist. In the following example, a table with three partitions (p0, p1, p2) is created:

```
mysql> USE test;
mysql> CREATE TABLE t1 (i int) ENGINE = InnoDB PARTITION BY KEY (i) PARTITIONS 3;
```

   In the */datadir*`/test` directory, there is a separate tablespace (`.ibd`) file for each of the three partitions.

```
mysql> \! ls /path/to/datadir/test/
t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd
```

2. On the destination instance, create the same partitioned table:

```
mysql> USE test;
mysql> CREATE TABLE t1 (i int) ENGINE = InnoDB PARTITION BY KEY (i) PARTITIONS 3;
```

   In the */datadir*`/test` directory, there is a separate tablespace (`.ibd`) file for each of the three partitions.

```
mysql> \! ls /path/to/datadir/test/
t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd
```

3. On the destination instance, discard the tablespace for the partitioned table. (Before the tablespace can be imported on the destination instance, the tablespace that is attached to the receiving table must be discarded.)

```
mysql> ALTER TABLE t1 DISCARD TABLESPACE;
```

   The three `.ibd` files that make up the tablespace for the partitioned table are discarded from the */datadir*`/test` directory.

4. On the source instance, run `FLUSH TABLES ... FOR EXPORT` to quiesce the partitioned table and create the `.cfg` metadata files:

```
mysql> USE test;
mysql> FLUSH TABLES t1 FOR EXPORT;
```

   Metadata (`.cfg`) files, one for each tablespace (`.ibd`) file, are created in the */datadir*`/test` directory on the source instance:

```
mysql> \! ls /path/to/datadir/test/
t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd
t1#P#p0.cfg  t1#P#p1.cfg  t1#P#p2.cfg
```

Note

`FLUSH TABLES ... FOR EXPORT` statement ensures that changes to the named table have been flushed to disk so that binary table copy can be made while the instance is running. When `FLUSH TABLES ... FOR EXPORT` is run, `InnoDB` produces a `.cfg` metadata file for the table's tablespace files in the same database directory as the table. The `.cfg` files contain metadata used for schema verification when importing tablespace files. `FLUSH TABLES ... FOR EXPORT` can only be run on the table, not on individual table partitions.

5. Copy the `.ibd` and `.cfg` files from the source instance database directory to the destination instance database directory. For example:

```
shell>scp /path/to/datadir/test/t1*.{ibd,cfg} destination-server:/path/to/datadir/test
```

Note

The `.ibd` and `.cfg` files must be copied before releasing the shared locks, as described in the next step.

6. On the source instance, use `UNLOCK TABLES` to release the locks acquired by `FLUSH TABLES ... FOR EXPORT`:

```
mysql> USE test;
mysql> UNLOCK TABLES;
```

7. On the destination instance, import the tablespace for the partitioned table:

```
mysql> USE test;
mysql> ALTER TABLE t1 IMPORT TABLESPACE;
```

Example 3: Copying InnoDB Table Partitions to Another Instance

This procedure demonstrates how to copy `InnoDB` table partitions from a running MySQL server instance to another running instance. The same procedure with minor adjustments can be used to perform a restore of `InnoDB` table partitions on the same instance. In the following example, a partitioned table with four partitions (p0, p1, p2, p3) is created on the source instance. Two of the partitions (p2 and p3) are copied to the destination instance.

1. On the source instance, create a partitioned table if one does not exist. In the following example, a table with four partitions (p0, p1, p2, p3) is created:

```
mysql> USE test;
mysql> CREATE TABLE t1 (i int) ENGINE = InnoDB PARTITION BY KEY (i) PARTITIONS 4;
```

In the *`/datadir/test`* directory, there is a separate tablespace (`.ibd`) file for each of the four partitions.

```
mysql> \! ls /path/to/datadir/test/
t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd t1#P#p3.ibd
```

2. On the destination instance, create the same partitioned table:

```
mysql> USE test;
```

```
mysql> CREATE TABLE t1 (i int) ENGINE = InnoDB PARTITION BY KEY (i) PARTITIONS 4;
```

In the */datadir/test* directory, there is a separate tablespace (.ibd) file for each of the four partitions.

```
mysql> \! ls /path/to/datadir/test/
t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd t1#P#p3.ibd
```

3. On the destination instance, discard the tablespace partitions that you plan to import from the source instance. (Before tablespace partitions can be imported on the destination instance, the corresponding partitions that are attached to the receiving table must be discarded.)

```
mysql> ALTER TABLE t1 DISCARD PARTITION p2, p3 TABLESPACE;
```

The .ibd files for the two discarded partitions are removed from the */datadir/test* directory on the destination instance, leaving the following files:

```
mysql> \! ls /path/to/datadir/test/
t1#P#p0.ibd  t1#P#p1.ibd
```

Note

When ALTER TABLE ... DISCARD PARTITION ... TABLESPACE is run on subpartitioned tables, both partition and subpartition table names are allowed. When a partition name is specified, subpartitions of that partition are included in the operation.

4. On the source instance, run FLUSH TABLES ... FOR EXPORT to quiesce the partitioned table and create the .cfg metadata files.

```
mysql> USE test;
mysql> FLUSH TABLES t1 FOR EXPORT;
```

The metadata files (.cfg files) are created in the */datadir/test* directory on the source instance. There is a .cfg file for each tablespace (.ibd) file.

```
mysql> \! ls /path/to/datadir/test/
t1#P#p0.ibd  t1#P#p1.ibd  t1#P#p2.ibd t1#P#p3.ibd
t1#P#p0.cfg  t1#P#p1.cfg  t1#P#p2.cfg t1#P#p3.cfg
```

Note

FLUSH TABLES ... FOR EXPORT statement ensures that changes to the named table have been flushed to disk so that binary table copy can be made while the instance is running. When FLUSH TABLES ... FOR EXPORT is run, InnoDB produces a .cfg metadata file for the table's tablespace files in the same database directory as the table. The .cfg files contain metadata used for schema verification when importing tablespace files. FLUSH TABLES ... FOR EXPORT can only be run on the table, not on individual table partitions.

5. Copy the .ibd and .cfg files from the source instance database directory to the destination instance database directory. In this example, only the .ibd and .cfg files for partition 2 (p2) and partition 3 (p3) are copied to the data directory on the destination instance. Partition 0 (p0) and partition 1 (p1) remain on the source instance.

```
shell> scp t1#P#p2.ibd  t1#P#p2.cfg t1#P#p3.ibd t1#P#p3.cfg destination-server:/path/to/datadir/test
```

Note

The `.ibd` files and `.cfg` files must be copied before releasing the shared locks, as described in the next step.

6. On the source instance, use `UNLOCK TABLES` to release the locks acquired by `FLUSH TABLES ... FOR EXPORT`:

```
mysql> USE test;
mysql> UNLOCK TABLES;
```

7. On the destination instance, import the tablespace partitions (p2 and p3):

```
mysql> USE test;
mysql> ALTER TABLE t1 IMPORT PARTITION p2, p3 TABLESPACE;
```

Note

When `ALTER TABLE ... IMPORT PARTITION ... TABLESPACE` is run on subpartitioned tables, both partition and subpartition table names are allowed. When a partition name is specified, subpartitions of that partition are included in the operation.

Transportable Tablespace Internals

The following information describes internals and error log messaging for the transportable tablespaces copy procedure for a regular `InnoDB` table.

When `ALTER TABLE ... DISCARD TABLESPACE` is run on the destination instance:

- The table is locked in X mode.
- The tablespace is detached from the table.

When `FLUSH TABLES ... FOR EXPORT` is run on the source instance:

- The table being flushed for export is locked in shared mode.
- The purge coordinator thread is stopped.
- Dirty pages are synchronized to disk.
- Table metadata is written to the binary `.cfg` file.

Expected error log messages for this operation:

```
2013-09-24T13:10:19.903526Z 2 [Note] InnoDB: Sync to disk of '"test"."t"' started.
2013-09-24T13:10:19.903586Z 2 [Note] InnoDB: Stopping purge
2013-09-24T13:10:19.903725Z 2 [Note] InnoDB: Writing table metadata to './test/t.cfg'
2013-09-24T13:10:19.904014Z 2 [Note] InnoDB: Table '"test"."t"' flushed to disk
```

When `UNLOCK TABLES` is run on the source instance:

- The binary .cfg file is deleted.
- The shared lock on the table or tables being imported is released and the purge coordinator thread is restarted.

Expected error log messages for this operation:

```
2013-09-24T13:10:21.181104Z 2 [Note] InnoDB: Deleting the meta-data file './test/t.cfg'
2013-09-24T13:10:21.181180Z 2 [Note] InnoDB: Resuming purge
```

When `ALTER TABLE ... IMPORT TABLESPACE` is run on the destination instance, the import algorithm performs the following operations for each tablespace being imported:

- Each tablespace page is checked for corruption.
- The space ID and log sequence numbers (LSNs) on each page are updated
- Flags are validated and LSN updated for the header page.
- Btree pages are updated.
- The page state is set to dirty so that it is written to disk.

Expected error log messages for this operation:

```
2013-07-18 15:15:01 34960 [Note] InnoDB: Importing tablespace for table 'test/t' that was exported from hos
2013-07-18 15:15:01 34960 [Note] InnoDB: Phase I - Update all pages
2013-07-18 15:15:01 34960 [Note] InnoDB: Sync to disk
2013-07-18 15:15:01 34960 [Note] InnoDB: Sync to disk - done!
2013-07-18 15:15:01 34960 [Note] InnoDB: Phase III - Flush changes to disk
2013-07-18 15:15:01 34960 [Note] InnoDB: Phase IV - Flush complete
```

Note

You may also receive a warning that a tablespace is discarded (if you discarded the tablespace for the destination table) and a message stating that statistics could not be calculated due to a missing `.ibd` file:

```
2013-07-18 15:14:38 34960 [Warning] InnoDB: Table "test"."t" tablespace is set as discard
2013-07-18 15:14:38 7f34d9a37700 InnoDB: cannot calculate statistics for table "test"."t"
because the .ibd file is missing. For help, please refer to
http://dev.mysql.com/doc/refman/8.0/en/innodb-troubleshooting.html
```

15.6.3.8 Moving Tablespace Files While the Server is Offline

The `innodb_directories` option, which defines directories to scan at startup for tablespace files, supports moving or restoring tablespace files to a new location while the server is offline. During startup, discovered tablespace files are used instead those referenced in the data dictionary, and the data dictionary is updated to reference the relocated files. If duplicate tablespace files are discovered by the scan, startup fails with an error indicating that multiple files were found for the same tablespace ID.

The directories defined by the `innodb_data_home_dir`, `innodb_undo_directory`, and `datadir` configuration options are automatically appended to the `innodb_directories` argument value. These directories are scanned at startup regardless of whether the `innodb_directories` option is specified explicitly. The implicit addition of these directories permits moving system tablespace files, the data directory, or undo tablespace files without configuring the `innodb_directories` setting. However,

settings must be updated when directories change. For example, after relocating the data directory, you must update the `--datadir` setting before restarting the server.

The `innodb_directories` option may be specified in a startup command or MySQL option file. Quotes are used around the argument value because otherwise a semicolon (;) is interpreted as a special character by some command interpreters. (Unix shells treat it as a command terminator, for example.)

Startup command:

```
mysqld --innodb-directories="directory_path_1;directory_path_2"
```

MySQL option file:

```
[mysqld]
innodb_directories="directory_path_1;directory_path_2"
```

The following procedure is applicable to moving individual file-per-table and general tablespace files, system tablespace files, undo tablespace files, or the data directory. Before moving files or directories, review the usage notes that follow.

1. Stop the server.
2. Move the tablespace files or directories.
3. Make the new directory known to `InnoDB`.
   - If moving individual file-per-table or general tablespace files, add unknown directories to the `innodb_directories` value.
     - The directories defined by the `innodb_data_home_dir`, `innodb_undo_directory`, and `datadir` configuration options are automatically appended to the `innodb_directories` argument value, so you need not specify these.
     - A file-per-table tablespace file can only be moved to a directory with same name as the schema. For example, if the `actor` table belongs to the `sakila` schema, then the `actor.ibd` data file can only be moved to a directory named `sakila`.
     - General tablespace files cannot be moved to the data directory or a subdirectory of the data directory.
   - If moving system tablespace files, undo tablespaces, or the data directory, update the `innodb_data_home_dir`, `innodb_undo_directory`, and `datadir` settings, as necessary.
4. Restart the server.

Usage Notes

- Wildcard expressions cannot be used in the `innodb_directories` argument value.
- The `innodb_directories` scan also traverses subdirectories of specified directories. Duplicate directories and subdirectories are discarded from the list of directories to be scanned.
- The `innodb_directories` option only supports moving `InnoDB` tablespace files. Moving files that belong to a storage engine other than `InnoDB` is not supported. This restriction also applies when moving the entire data directory.
- The `innodb_directories` option supports renaming of tablespace files when moving files to a scanned directory. It also supports moving tablespaces files to other supported operating systems.

- When moving tablespace files to a different operating system, ensure that tablespace file names do not include prohibited characters or characters with a special meaning on the destination system.
- When moving a data directory from a Windows operating system to a Linux operating system, modify the binary log file paths in the binary log index file to use backward slashes instead of forward slashes. By default, the binary log index file has the same base name as the binary log file, with the extension '`.index`'. The location of the binary log index file is defined by `--log-bin`. The default location is the data directory.
- If moving tablespace files to a different operating system introduces cross-platform replication, it is the responsibility of the Database Administrator to ensure proper replication of DDL statements that contain platform-specific directories. Statements that permit specifying directories include `CREATE TABLE ... DATA DIRECTORY` and `CREATE TABLESPACE`.
- The directory of file-per-table and general tablespace files created with an absolute path or in a location outside of the data directory should be added to the `innodb_directories` argument value. Otherwise, `InnoDB` is not able to locate these files during recovery. `CREATE TABLE ... DATA DIRECTORY` and `CREATE TABLESPACE` permit creation of tablespace files with absolute paths. `CREATE TABLESPACE` also permits tablespace file directories that are relative to the data directory. To view tablespace file locations, query the `INFORMATION_SCHEMA.FILES` table:

```
mysql> SELECT TABLESPACE_NAME, FILE_NAME FROM INFORMATION_SCHEMA.FILES \G
```

- `CREATE TABLESPACE` requires that the target directory exists and is known to `InnoDB`. Known directories include those implicitly and explicitly defined by the `innodb_directories` option.

15.6.3.9 InnoDB Data-at-Rest Encryption

`InnoDB` supports data-at-rest encryption for file-per-table tablespaces, general tablespaces, the `mysql` system tablespace, redo logs, and undo logs.

As of MySQL 8.0.16, setting an encryption default for schemas and general tablespaces is also supported, which permits DBAs to control whether tables created in those schemas and tablespaces are encrypted.

`InnoDB` data-at-rest encryption features and capabilities are described under the following topics in this section.

- About Data-at-Rest Encryption
- Encryption Prerequisites
- Defining an Encryption Default for Schemas and General Tablespaces
- File-Per-Table Tablespace Encryption
- General Tablespace Encryption
- mysql System Tablespace Encryption
- Redo Log Encryption
- Undo Log Encryption
- Master Key Rotation
- Encryption and Recovery

- Exporting Encrypted Tablespaces
- Encryption and Replication
- Identifying Encrypted Tablespaces and Schemas
- Monitoring Encryption Progress
- Encryption Usage Notes
- Encryption Limitations

About Data-at-Rest Encryption

`InnoDB` uses a two tier encryption key architecture, consisting of a master encryption key and tablespace keys. When a tablespace is encrypted, a tablespace key is encrypted and stored in the tablespace header. When an application or authenticated user wants to access encrypted tablespace data, `InnoDB` uses a master encryption key to decrypt the tablespace key. The decrypted version of a tablespace key never changes, but the master encryption key can be changed as required. This action is referred to as *master key rotation*.

The data-at-rest encryption feature relies on a keyring plugin for master encryption key management.

All MySQL editions provide a `keyring_file` plugin, which stores keyring data in a file local to the server host.

MySQL Enterprise Edition offers additional keyring plugins:

- The `keyring_encrypted_file` plugin, which stores keyring data in an encrypted file local to the server host.
- The `keyring_okv` plugin, which includes a KMIP client (KMIP 1.1) that uses a KMIP-compatible product as a back end for keyring storage. Supported KMIP-compatible products include centralized key management solutions such as Oracle Key Vault, Gemalto KeySecure, Thales Vormetric key management server, and Fornetix Key Orchestration.
- The `keyring_aws` plugin, which communicates with the Amazon Web Services Key Management Service (AWS KMS) as a back end for key generation and uses a local file for key storage.

Warning

The `keyring_file` and `keyring_encrypted file` plugins are not intended as regulatory compliance solutions. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

A secure and robust encryption key management solution is critical for security and for compliance with various security standards. When the data-at-rest encryption feature uses a centralized key management solution, the feature is referred to as “MySQL Enterprise Transparent Data Encryption (TDE)”.

The data-at-rest encryption feature supports the Advanced Encryption Standard (AES) block-based encryption algorithm. It uses Electronic Codebook (ECB) block encryption mode for tablespace key encryption and Cipher Block Chaining (CBC) block encryption mode for data encryption.

For frequently asked questions about the data-at-rest encryption feature, see Section A.17, “MySQL 8.0 FAQ: InnoDB Data-at-Rest Encryption”.

Encryption Prerequisites

- A keyring plugin must be installed and configured. Keyring plugin installation is performed at startup using the `early-plugin-load` option. Early loading ensures that the plugin is available prior to initialization of the `InnoDB` storage engine. For keyring plugin installation and configuration instructions, see Section 6.4.4, “The MySQL Keyring”.

  Only one keyring plugin can be enabled at a time. Enabling multiple keyring plugins is not supported.

  Important

  Once encrypted tablespaces are created in a MySQL instance, the keyring plugin that was loaded when creating the encrypted tablespace must continue to be loaded at startup using the `early-plugin-load` option. Failing to do so results in errors when starting the server and during `InnoDB` recovery.

  To verify that a keyring plugin is active, use the `SHOW PLUGINS` statement or query the `INFORMATION_SCHEMA.PLUGINS` table. For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'keyring%';
+--------------+---------------+
| PLUGIN_NAME  | PLUGIN_STATUS |
+--------------+---------------+
| keyring_file | ACTIVE        |
+--------------+---------------+
```

- When encrypting production data, ensure that you take steps to prevent loss of the master encryption key. *If the master encryption key is lost, data stored in encrypted tablespace files is unrecoverable.* If you use the `keyring_file` or `keyring_encrypted_file` plugin, create a backup of the keyring data file immediately after creating the first encrypted tablespace, before master key rotation, and after master key rotation. The `keyring_file_data` configuration option defines the keyring data file location for the `keyring_file` plugin. The `keyring_encrypted_file_data` configuration option defines the keyring data file location for the `keyring_encrypted_file` plugin. If you use the `keyring_okv` or `keyring_aws` plugin, ensure that you have performed the necessary configuration. For instructions, see Section 6.4.4, “The MySQL Keyring”.

Defining an Encryption Default for Schemas and General Tablespaces

As of MySQL 8.0.16, the `default_table_encryption` variable defines the default encryption setting for schemas and general tablespaces. `CREATE TABLESPACE` and `CREATE SCHEMA` operations apply the `default_table_encryption` setting when an `ENCRYPTION` clause is not specified explicitly.

`ALTER SCHEMA` and `ALTER TABLESPACE` operations do not apply the `default_table_encryption` setting. An `ENCRYPTION` clause must be specified explicitly to alter the encryption of an existing schema or general tablespace.

The `default_table_encryption` variable can be set for an individual client connection or globally using `SET` syntax. For example, the following statement enables default schema and tablespace encryption globally:

```
mysql> SET GLOBAL default_table_encryption=ON;
```

The default encryption setting for a schema can also be defined using the `DEFAULT ENCRYPTION` clause when creating or altering a schema, as in this example:

```
mysql> CREATE SCHEMA test DEFAULT ENCRYPTION = 'Y';
```

If the DEFAULT ENCRYPTION clause is not specified when creating a schema, the default_table_encryption setting is applied. The DEFAULT ENCRYPTION clause must be specified to alter the default encryption of an existing schema. Otherwise, the schema retains its current encryption setting.

By default, a table inherits the encryption setting of the schema or general tablespace it is created in. For example, a table created in an encryption-enabled schema is encrypted by default. This behavior enables a DBA to control table encryption usage by defining and enforcing schema and general tablespace encryption defaults.

Encryption defaults are enforced by enabling the table_encryption_privilege_check variable. When table_encryption_privilege_check is enabled, a privilege check occurs when creating or altering a schema or general tablespace with an encryption setting that differs from the default_table_encryption setting, or when creating or altering a table with an encryption setting that differs from the default schema encryption. When table_encryption_privilege_check is disabled (the default), the privilege check does not occur and the previously mentioned operations are permitted to proceed with a warning.

The TABLE_ENCRYPTION_ADMIN privilege is required to override default encryption settings when table_encryption_privilege_check is enabled. A DBA can grant this privilege to enable a user to deviate from the default_table_encryption setting when creating or altering a schema or general tablespace, or to deviate from the default schema encryption when creating or altering a table. This privilege does not permit deviating from the encryption of a general tablespace when creating or altering a table. A table must have the same encryption setting as the general tablespace it resides in.

File-Per-Table Tablespace Encryption

As of MySQL 8.0.16, a file-per-table tablespace inherits the default encryption of the schema in which the table is created unless an ENCRYPTION clause is specified explcitly in the CREATE TABLE statement. Prior to MySQL 8.0.16, the ENCRYPTION clause must be specified to enable encryption.

```
mysql> CREATE TABLE t1 (c1 INT) ENCRYPTION = 'Y';
```

To alter the encryption of an existing file-per-table tablespace, an ENCRYPTION clause must be specified.

```
mysql> ALTER TABLE t1 ENCRYPTION = 'Y';
```

As of MySQL 8.0.16, if the table_encryption_privilege_check variable is enabled, specifying an ENCRYPTION clause with a setting that differs from the default schema encryption requires the TABLE_ENCRYPTION_ADMIN privilege. See Defining an Encryption Default for Schemas and General Tablespaces.

General Tablespace Encryption

As of MySQL 8.0.16, the default_table_encryption variable determines the encryption of a newly created general tablespace unless an ENCRYPTION clause is specified explicitly in the CREATE TABLESPACE statement. Prior to MySQL 8.0.16, an ENCRYPTION clause must be specified to enable encryption.

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' ENCRYPTION = 'Y' Engine=InnoDB;
```

To alter the encryption of an existing general tablespace, an ENCRYPTION clause must be specified.

```
mysql> ALTER TABLESPACE ts1 ENCRYPTION = 'Y';
```

As of MySQL 8.0.16, if the `table_encryption_privilege_check` variable is enabled, specifying an `ENCRYPTION` clause with a setting that differs from the `default_table_encryption` setting requires the `TABLE_ENCRYPTION_ADMIN` privilege. See Defining an Encryption Default for Schemas and General Tablespaces.

mysql System Tablespace Encryption

Encryption support for the `mysql` system tablespace is available as of MySQL 8.0.16.

The `mysql` system tablespace contains the `mysql` system database and MySQL data dictionary tables. It is unencrypted by default. To enable encryption for the `mysql` system tablespace, specify the tablespace name and the `ENCRYPTION` option in an `ALTER TABLESPACE` statement.

```
mysql> ALTER TABLESPACE mysql ENCRYPTION = 'Y';
```

To disable encryption for the `mysql` system tablespace, set `ENCRYPTION = 'N'` using an `ALTER TABLESPACE` statement.

```
mysql> ALTER TABLESPACE mysql ENCRYPTION = 'N';
```

Enabling or disabling encryption for the `mysql` system tablespace requires the `CREATE TABLESPACE` privilege on all tables in the instance (`CREATE TABLESPACE on *.*`).

Redo Log Encryption

Redo log data encryption is enabled using the `innodb_redo_log_encrypt` configuration option. Redo log encryption is disabled by default.

As with tablespace data, redo log data encryption occurs when redo log data is written to disk, and decryption occurs when redo log data is read from disk. Once redo log data is read into memory, it is in unencrypted form. Redo log data is encrypted and decrypted using the tablespace encryption key.

When `innodb_redo_log_encrypt` is enabled, unencrypted redo log pages that are present on disk remain unencrypted, and new redo log pages are written to disk in encrypted form. Likewise, when `innodb_redo_log_encrypt` is disabled, encrypted redo log pages that are present on disk remain encrypted, and new redo log pages are written to disk in unencrypted form.

Redo log encryption metadata, including the tablespace encryption key, is stored in the header of the first redo log file (`ib_logfile0`). If this file is removed, redo log encryption is disabled.

Once redo log encryption is enabled, a normal restart without the keyring plugin or without the encryption key is not possible, as `InnoDB` must be able to scan redo pages during startup, which is not possible if redo log pages are encrypted. Without the keyring plugin or the encryption key, only a forced startup without the redo logs (`SRV_FORCE_NO_LOG_REDO`) is possible. See Section 15.20.2, “Forcing InnoDB Recovery”.

Undo Log Encryption

Undo log data encryption is enabled using the `innodb_undo_log_encrypt` configuration option. Undo log encryption applies to undo logs that reside in undo tablespaces. See Section 15.6.3.4, “Undo Tablespaces”. Undo log data encryption is disabled by default.

As with tablespace data, undo log data encryption occurs when undo log data is written to disk, and decryption occurs when undo log data is read from disk. Once undo log data is read into memory, it is in unencrypted form. Undo log data is encrypted and decrypted using the tablespace encryption key.

When `innodb_undo_log_encrypt` is enabled, unencrypted undo log pages that are present on disk remain unencrypted, and new undo log pages are written to disk in encrypted form. Likewise, when `innodb_undo_log_encrypt` is disabled, encrypted undo log pages that are present on disk remain encrypted, and new undo log pages are written to disk in unencrypted form.

Undo log encryption metadata, including the tablespace encryption key, is stored in the header of the undo log file.

Note

When undo log encryption is disabled, the server continues to require the keyring plugin that was used to encrypt undo log data until the undo tablespaces that contained the encrypted undo log data are truncated. (An encryption header is only removed from an undo tablespace when the undo tablespace is truncated.) For information about truncating undo tablespaces, see Truncating Undo Tablespaces.

Master Key Rotation

The master encryption key should be rotated periodically and whenever you suspect that the key has been compromised.

Master key rotation is an atomic, instance-level operation. Each time the master encryption key is rotated, all tablespace keys in the MySQL instance are re-encrypted and saved back to their respective tablespace headers. As an atomic operation, re-encryption must succeed for all tablespace keys once a rotation operation is initiated. If master key rotation is interrupted by a server failure, `InnoDB` rolls the operation forward on server restart. For more information, see Encryption and Recovery.

Rotating the master encryption key only changes the master encryption key and re-encrypts tablespace keys. It does not decrypt or re-encrypt associated tablespace data.

Rotating the master encryption key requires the `ENCRYPTION_KEY_ADMIN` or `SUPER` privilege.

To rotate the master encryption key, run:

```
mysql> ALTER INSTANCE ROTATE INNODB MASTER KEY;
```

`ALTER INSTANCE ROTATE INNODB MASTER KEY` supports concurrent DML. However, it cannot be run concurrently with tablespace encryption operations, and locks are taken to prevent conflicts that could arise from concurrent execution. If an `ALTER INSTANCE ROTATE INNODB MASTER KEY` operation is running, it must finish before a tablespace encryption operation can proceed, and vice versa.

Encryption and Recovery

If a server failure occurs during an encryption operation, the operation is rolled forward when the server is restarted. For general tablespaces, the encryption operation is resumed in a background thread from the last processed page.

If a server failure occurs during master key rotation, `InnoDB` continues the operation on server restart.

The keyring plugin must be loaded prior to storage engine initialization so that the information necessary to decrypt tablespace data pages can be retrieved from tablespace headers before `InnoDB` initialization and recovery activities access tablespace data. (See Encryption Prerequisites.)

When InnoDB initialization and recovery begin, the master key rotation operation resumes. Due to the server failure, some tablespace keys may already be encrypted using the new master encryption key. InnoDB reads the encryption data from each tablespace header, and if the data indicates that the tablespace key is encrypted using the old master encryption key, InnoDB retrieves the old key from the keyring and uses it to decrypt the tablespace key. InnoDB then re-encrypts the tablespace key using the new master encryption key and saves the re-encrypted tablespace key back to the tablespace header.

Exporting Encrypted Tablespaces

Tablespace export is only supported for file-per-table tablespaces.

When an encrypted tablespace is exported, InnoDB generates a *transfer key* that is used to encrypt the tablespace key. The encrypted tablespace key and transfer key are stored in a *tablespace_name*.cfp file. This file together with the encrypted tablespace file is required to perform an import operation. On import, InnoDB uses the transfer key to decrypt the tablespace key in the *tablespace_name*.cfp file. For related information, see Section 15.6.3.7, “Copying Tablespaces to Another Instance”.

Encryption and Replication

- The ALTER INSTANCE ROTATE INNODB MASTER KEY statement is only supported in replication environments where the master and slaves run a version of MySQL that supports tablespace encryption.
- Successful ALTER INSTANCE ROTATE INNODB MASTER KEY statements are written to the binary log for replication on slaves.
- If an ALTER INSTANCE ROTATE INNODB MASTER KEY statement fails, it is not logged to the binary log and is not replicated on slaves.
- Replication of an ALTER INSTANCE ROTATE INNODB MASTER KEY operation fails if the keyring plugin is installed on the master but not on the slave.
- If the keyring_file or keyring_encrypted_file plugin is installed on both the master and a slave but the slave does not have a keyring data file, the replicated ALTER INSTANCE ROTATE INNODB MASTER KEY statement creates the keyring data file on the slave, assuming the keyring file data is not cached in memory. ALTER INSTANCE ROTATE INNODB MASTER KEY uses keyring file data that is cached in memory, if available.

Identifying Encrypted Tablespaces and Schemas

The INFORMATION_SCHEMA.INNODB_TABLESPACES table, introduced in MySQL 8.0.13, includes an ENCRYPTION column that can be used to identify encrypted tablespaces.

```
mysql> SELECT SPACE, NAME, SPACE_TYPE, ENCRYPTION FROM INFORMATION_SCHEMA.INNODB_TABLESPACES
       WHERE ENCRYPTION='Y'\G
*************************** 1. row ***************************
     SPACE: 4294967294
      NAME: mysql
SPACE_TYPE: General
ENCRYPTION: Y
*************************** 2. row ***************************
     SPACE: 2
      NAME: test/t1
SPACE_TYPE: Single
ENCRYPTION: Y
*************************** 3. row ***************************
     SPACE: 3
      NAME: ts1
SPACE_TYPE: General
```

```
ENCRYPTION: Y
```

When the `ENCRYPTION` option is specified in a `CREATE TABLE` or `ALTER TABLE` statement, it is recorded in the `CREATE_OPTIONS` column of `INFORMATION_SCHEMA.TABLES`. This column can be queried to identify tables that reside in encrypted file-per-table tablespaces.

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, CREATE_OPTIONS FROM INFORMATION_SCHEMA.TABLES
       WHERE CREATE_OPTIONS LIKE '%ENCRYPTION%';
+--------------+------------+----------------+
| TABLE_SCHEMA | TABLE_NAME | CREATE_OPTIONS |
+--------------+------------+----------------+
| test         | t1         | ENCRYPTION="Y" |
+--------------+------------+----------------+
```

Query `INFORMATION_SCHEMA.INNODB_TABLESPACES` to retrieve information about the tablespace associated with a particular schema and table.

```
mysql> SELECT SPACE, NAME, SPACE_TYPE FROM INFORMATION_SCHEMA.INNODB_TABLESPACES WHERE NAME='test/t1';
+-------+---------+------------+
| SPACE | NAME    | SPACE_TYPE |
+-------+---------+------------+
|     3 | test/t1 | Single     |
+-------+---------+------------+
```

You can identify encryption-enabled schemas by querying the `INFORMATION_SCHEMA.SCHEMATA` table.

```
mysql> SELECT SCHEMA_NAME, DEFAULT_ENCRYPTION FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE DEFAULT_ENCRYPTION='YES';
+-------------+--------------------+
| SCHEMA_NAME | DEFAULT_ENCRYPTION |
+-------------+--------------------+
| test        | YES                |
+-------------+--------------------+
```

`SHOW CREATE SCHEMA` also shows the `DEFAULT ENCRYPTION` clause.

Monitoring Encryption Progress

You can monitor general tablespace and `mysql` system tablespace encryption progress using Performance Schema.

The `stage/innodb/alter tablespace (encryption)` stage event instrument reports `WORK_ESTIMATED` and `WORK_COMPLETED` information for general tablespace encryption operations.

The following example demonstrates how to enable the `stage/innodb/alter tablespace (encryption)` stage event instrument and related consumer tables to monitor general tablespace or `mysql` system tablespace encryption progress. For information about Performance Schema stage event instruments and related consumers, see Section 26.12.5, “Performance Schema Stage Event Tables”.

1. Enable the `stage/innodb/alter tablespace (encryption)` instrument:

   ```
   mysql> USE performance_schema;
   mysql> UPDATE setup_instruments SET ENABLED = 'YES'
          WHERE NAME LIKE 'stage/innodb/alter tablespace (encryption)';
   ```

2. Enable the stage event consumer tables, which include `events_stages_current`, `events_stages_history`, and `events_stages_history_long`.

```
mysql> UPDATE setup_consumers SET ENABLED = 'YES' WHERE NAME LIKE '%stages%';
```

3. Run a tablespace encryption operation. In this example, a general tablespace named `ts1` is encrypted.

```
mysql> ALTER TABLESPACE ts1 ENCRYPTION = 'Y';
```

4. Check the progress of the encryption operation by querying the Performance Schema `events_stages_current` table. `WORK_ESTIMATED` reports the total number of pages in the tablespace. `WORK_COMPLETED` reports the number of pages processed.

```
mysql> SELECT EVENT_NAME, WORK_ESTIMATED, WORK_COMPLETED FROM events_stages_current;
+----------------------------------------+----------------+----------------+
| EVENT_NAME                             | WORK_COMPLETED | WORK_ESTIMATED |
+----------------------------------------+----------------+----------------+
| stage/innodb/alter tablespace (encryption) |           1056 |           1407 |
+----------------------------------------+----------------+----------------+
```

The `events_stages_current` table returns an empty set if the encryption operation has completed. In this case, you can check the `events_stages_history` table to view event data for the completed operation. For example:

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED FROM events_stages_history;
+----------------------------------------+----------------+----------------+
| EVENT_NAME                             | WORK_COMPLETED | WORK_ESTIMATED |
+----------------------------------------+----------------+----------------+
| stage/innodb/alter tablespace (encryption) |           1407 |           1407 |
+----------------------------------------+----------------+----------------+
```

Encryption Usage Notes

- Plan appropriately when altering an existing file-per-table tablespace with the `ENCRYPTION` option. Tables residing in file-per-table tablespaces are rebuilt using the `COPY` algorithm. The `INPLACE` algorithm is used when altering the `ENCRYPTION` attribute of a general tablespace or the `mysql` system tablespace. The `INPLACE` algorithm permits concurrent DML on tables that reside in the general tablespace. Concurrent DDL is blocked.
- When a general tablespace or the `mysql` system tablespace is encrypted, all tables residing in the tablespace are encrypted. Likewise, a table created in an encrypted tablespace is encrypted.
- If the server exits or is stopped during normal operation, it is recommended to restart the server using the same encryption settings that were configured previously.
- The first master encryption key is generated when the first new or existing tablespace is encrypted.
- Master key rotation re-encrypts tablespaces keys but does not change the tablespace key itself. To change a tablespace key, you must disable and re-enable encryption. For file-per-table tablespaces, re-encrypting the tablespace is an `ALGORITHM=COPY` operation that rebuilds the table. For general tablespaces and the `mysql` system tablespace, it is an `ALGORITHM=INPLACE` operation, which does not require rebuilding tables that reside in the tablespace.
- If a table is created with both the `COMPRESSION` and `ENCRYPTION` options, compression is performed before tablespace data is encrypted.
- If a keyring data file (the file named by `keyring_file_data` or `keyring_encrypted_file_data`) is empty or missing, the first execution of `ALTER INSTANCE ROTATE INNODB MASTER KEY` creates a master encryption key.

- Uninstalling the `keyring_file` or `keyring_encrypted_file` plugin does not remove an existing keyring data file.
- It is recommended that you not place a keyring data file under the same directory as tablespace data files.
- Modifying the `keyring_file_data` or `keyring_encrypted_file_data` setting at runtime or when restarting the server can cause previously encrypted tablespaces to become inaccessible, resulting in lost data.

Encryption Limitations

- Advanced Encryption Standard (AES) is the only supported encryption algorithm. `InnoDB` tablespace encryption uses Electronic Codebook (ECB) block encryption mode for tablespace key encryption and Cipher Block Chaining (CBC) block encryption mode for data encryption.
- Encryption is only supported for file-per-table tablespaces, general tablespaces, and the `mysql` system tablespace. Encryption support for general tablespaces was introduced in MySQL 8.0.13. Encryption support for the `mysql` system tablespace is available as of MySQL 8.0.16. Encryption is not supported for other tablespace types including the `InnoDB` system tablespace.
- You cannot move or copy a table from an encrypted file-per-table tablespace, general tablespace, or the `mysql` system tablespace to a tablespace type that does not support encryption.
- You cannot move or copy a table from an encrypted tablespace to an unencrypted tablespace. However, moving a table from an unencrypted tablespace to an encrypted one is permitted. For example, you can move or copy a table from a unencrypted file-per-table or general tablespace to an encrypted general tablespace.
- By default, tablespace encryption only applies to data in the tablespace. Redo log and undo log data can be encrypted by enabling `innodb_redo_log_encrypt` and `innodb_undo_log_encrypt`. See Redo Log Encryption, and Undo Log Encryption. Binary log data is not encrypted.
- It is not permitted to change the storage engine of a table that resides in, or previously resided in, an encrypted tablespace.

15.6.4 Doublewrite Buffer

The doublewrite buffer is a storage area located in the system tablespace where `InnoDB` writes pages that are flushed from the `InnoDB` buffer pool, before the pages are written to their proper positions in the data file. Only after flushing and writing pages to the doublewrite buffer, does `InnoDB` write pages to their proper positions. If there is an operating system, storage subsystem, or `mysqld` process crash in the middle of a page write, `InnoDB` can later find a good copy of the page from the doublewrite buffer during crash recovery.

Although data is always written twice, the doublewrite buffer does not require twice as much I/O overhead or twice as many I/O operations. Data is written to the doublewrite buffer itself as a large sequential chunk, with a single `fsync()` call to the operating system.

The doublewrite buffer is enabled by default in most cases. To disable the doublewrite buffer, set `innodb_doublewrite` to 0.

If system tablespace files (“ibdata files”) are located on Fusion-io devices that support atomic writes, doublewrite buffering is automatically disabled and Fusion-io atomic writes are used for all data files. Because the doublewrite buffer setting is global, doublewrite buffering is also disabled for data files

residing on non-Fusion-io hardware. This feature is only supported on Fusion-io hardware and is only enabled for Fusion-io NVMFS on Linux. To take full advantage of this feature, an `innodb_flush_method` setting of `O_DIRECT` is recommended.

15.6.5 Redo Log

The redo log is a disk-based data structure used during crash recovery to correct data written by incomplete transactions. During normal operations, the redo log encodes requests to change table data that result from SQL statements or low-level API calls. Modifications that did not finish updating the data files before an unexpected shutdown are replayed automatically during initialization, and before the connections are accepted. For information about the role of the redo log in crash recovery, see Section 15.17.2, “InnoDB Recovery”.

By default, the redo log is physically represented on disk by two files named `ib_logfile0` and `ib_logfile1`. MySQL writes to the redo log files in a circular fashion. Data in the redo log is encoded in terms of records affected; this data is collectively referred to as redo. The passage of data through the redo log is represented by an ever-increasing LSN value.

For related information, see Redo Log File Configuration, and Section 8.5.4, “Optimizing InnoDB Redo Logging”.

For information about data-at-rest encryption for redo logs, see Redo Log Encryption.

Changing the Number or Size of Redo Log Files

To change the number or the size of redo log files, perform the following steps:

1. Stop the MySQL server and make sure that it shuts down without errors.
2. Edit `my.cnf` to change the log file configuration. To change the log file size, configure `innodb_log_file_size`. To increase the number of log files, configure `innodb_log_files_in_group`.
3. Start the MySQL server again.

If `InnoDB` detects that the `innodb_log_file_size` differs from the redo log file size, it writes a log checkpoint, closes and removes the old log files, creates new log files at the requested size, and opens the new log files.

Group Commit for Redo Log Flushing

`InnoDB`, like any other ACID-compliant database engine, flushes the redo log of a transaction before it is committed. `InnoDB` uses group commit functionality to group multiple such flush requests together to avoid one flush for each commit. With group commit, `InnoDB` issues a single write to the log file to perform the commit action for multiple user transactions that commit at about the same time, significantly improving throughput.

For more information about performance of `COMMIT` and other transactional operations, see Section 8.5.2, “Optimizing InnoDB Transaction Management”.

Redo Log Archiving

Backup utilities that copy redo log records may sometimes fail to keep pace with redo log generation while a backup operation is in progress, resulting in lost redo log records due to those records being overwritten.

This issue most often occurs when there is significant MySQL server activity during the backup operation, and the redo log file storage media operates at a faster speed than the backup storage media. The redo log archiving feature, introduced in MySQL 8.0.17, addresses this issue by sequentially writing redo log records to an archive file in addition to the redo log files. Backup utilities can copy redo log records from the archive file as necessary, thereby avoiding the potential loss of data.

If redo log archiving is configured on the server, MySQL Enterprise Backup, available with the MySQL Enterprise Edition, uses the redo log archiving feature when backing up a MySQL server.

Enabling redo log archiving on the server requires setting a value for the `innodb_redo_log_archive_dirs` system variable. The value is specified as a semicolon-separated list of labeled redo log archive directories. The *`label:directory`* pair is separated by a colon (`:`). For example:

```
mysql> SET GLOBAL innodb_redo_log_archive_dirs='label1:directory_path1[;label2:directory_path2;…]';
```

The *`label`* is an arbitrary identifier for the archive directory. It can be any string of characters, with the exception of colons (:), which are not permitted. An empty label is also permitted, but the colon (:) is still required in this case. A *`directory_path`* must be specified. The directory that is selected for the redo log archive file must exist when redo log archiving is activated, or an error is returned. The path can contain colons (':'), but semicolons (;) are not permitted.

The `innodb_redo_log_archive_dirs` variable must be configured before the redo log archiving can be activated. The default value is `NULL`, which does not permit activating redo log archiving.

Notes

The archive directories that you specify must satisfy the following requirements. (The requirements are enforced when redo log archiving is activated.):

- Directories must exist. Directories are not created by the redo log archive process. Otherwise, the following error is returned:

  ```
  ERROR 3844 (HY000): Redo log archive directory
  'directory_path1' does not exist or is not a directory
  ```

- Directories must not be world-accessible. This is to prevent the redo log data from being exposed to unauthorized users on the system. Otherwise, the following error is returned:

  ```
  ERROR 3846 (HY000): Redo log archive directory
  'directory_path1' is accessible to all OS users
  ```

- Directories cannot be those defined by `datadir`, `innodb_data_home_dir`, `innodb_directories`, `innodb_log_group_home_dir`, `innodb_temp_tablespaces_dir`, `innodb_tmpdir` `innodb_undo_directory`, or `secure_file_priv`, nor can they be parent directories or subdirectories of those directories. Otherwise, an error similar to the following is returned:

  ```
  ERROR 3845 (HY000): Redo log archive directory
  'directory_path1' is in, under, or over server directory
  'datadir' - '/path/to/data_directory'
  ```

When a backup utility that supports redo log archiving initiates a backup, the backup utility activates redo log archiving by invoking the `innodb_redo_log_archive_start()` user-defined function.

If you are not using a backup utility that supports redo log archiving, redo log archiving can also be activated manually, as shown:

```
mysql> SELECT innodb_redo_log_archive_start('label', 'subdir');
+-----------------------------------------+
| innodb_redo_log_archive_start('label') |
+-----------------------------------------+
| 0                                       |
+-----------------------------------------+
```

Or:

```
mysql> DO innodb_redo_log_archive_start('label', 'subdir');
Query OK, 0 rows affected (0.09 sec)
```

Note

The MySQL session that activates redo log archiving (using `innodb_redo_log_archive_start()`) must remain open for the duration of the archiving. The same session must deactivate redo log archiving (using `innodb_redo_log_archive_stop()`). If the session is terminated before the redo log archiving is explicitly deactivated, the server deactivates redo log archiving implicitly and removes the redo log archive file.

where *`label`* is a label defined by `innodb_redo_log_archive_dirs`; `subdir` is an optional argument for specifying a subdirectory of the directory identified by *`label`* for saving the archive file; it must be a simple directory name (no slash (/), backslash (\), or colon (:) is permitted). `subdir` can be empty, null, or it can be left out.

Only users with the `INNODB_REDO_LOG_ARCHIVE` privilege can activate redo log archiving by invoking `innodb_redo_log_archive_start()`, or deactivate it using `innodb_redo_log_archive_stop()`. The MySQL user running the backup utility or the MySQL user activating and deactivating redo log archiving manually must have this privilege.

The redo log archive file path is *`directory_identified_by_label/`*`[`*`subdir/`*`]archive.`*`serverUUID`*`.000001.log`, where *`directory_identified_by_label`* is the archive directory identified by the *`label`* argument for `innodb_redo_log_archive_start()`. *`subdir`* is the optional argument used for `innodb_redo_log_archive_start()`.

For example, the full path and name for a redo log archive file appears similar to the following:

```
/directory_path/subdirectory/archive.e71a47dc-61f8-11e9-a3cb-080027154b4d.000001.log
```

After the backup utility finishes copying `InnoDB` data files, it deactivates redo log archiving by calling the `innodb_redo_log_archive_stop()` user-defined function.

If you are not using a backup utility that supports redo log archiving, redo log archiving can also be deactivated manually, as shown:

```
mysql> SELECT innodb_redo_log_archive_stop();
+--------------------------------+
| innodb_redo_log_archive_stop() |
+--------------------------------+
| 0                              |
+--------------------------------+
```

Or:

```
mysql> DO innodb_redo_log_archive_stop();
Query OK, 0 rows affected (0.01 sec)
```

After the stop function completes successfully, the backup utility looks for the relevant section of redo log data from the archive file and copies it into the backup.

After the backup utility finishes copying the redo log data and no longer needs the redo log archive file, it deletes the archive file.

Removal of the archive file is the responsibility of the backup utility in normal situations. However, if the redo log archiving operation quits unexpectedly before `innodb_redo_log_archive_stop()` is called, the MySQL server removes the file.

Performance Considerations

Activating redo log archiving typically has a minor performance cost due to the additional write activity.

On Unix and Unix-like operating systems, the performance impact is typically minor, assuming there is not a sustained high rate of updates. On Windows, the performance impact is typically a bit higher, assuming the same.

If there is a sustained high rate of updates and the redo log archive file is on the same storage media as the redo log files, the performance impact may be more significant due to compounded write activity.

If there is a sustained high rate of updates and the redo log archive file is on slower storage media than the redo log files, performance is impacted arbitrarily.

Writing to the redo log archive file does not impede normal transactional logging except in the case that the redo log archive file storage media operates at a much slower rate than the redo log file storage media, and there is a large backlog of persisted redo log blocks waiting to be written to the redo log archive file. In this case, the transactional logging rate is reduced to a level that can be managed by the slower storage media where the redo log archive file resides.

15.6.6 Undo Logs

An undo log is a collection of undo log records associated with a single read-write transaction. An undo log record contains information about how to undo the latest change by a transaction to a clustered index record. If another transaction needs to see the original data as part of a consistent read operation, the unmodified data is retrieved from undo log records. Undo logs exist within undo log segments, which are contained within rollback segments. Rollback segments reside in undo tablespaces and in the global temporary tablespace.

Undo logs that reside in the global temporary tablespace are used for transactions that modify data in user-defined temporary tables. These undo logs are not redo-logged, as they are not required for crash recovery. They are used only for rollback while the server is running. This type of undo log benefits performance by avoiding redo logging I/O.

For information about data-at-rest encryption for undo logs, see Undo Log Encryption.

Each undo tablespace and the global temporary tablespace individually support a maximum of 128 rollback segments. The `innodb_rollback_segments` variable defines the number of rollback segments.

The number of transactions that a rollback segment supports depends on the number of undo slots in the rollback segment and the number of undo logs required by each transaction.

The number of undo slots in a rollback segment differs according to InnoDB page size.

InnoDB Page Size	Number of Undo Slots in a Rollback Segment (InnoDB Page Size / 16)
4096 (4KB)	256
8192 (8KB)	512
16384 (16KB)	1024
32768 (32KB)	2048
65536 (64KB)	4096

A transaction is assigned up to four undo logs, one for each of the following operation types:

1. INSERT operations on user-defined tables
2. UPDATE and DELETE operations on user-defined tables
3. INSERT operations on user-defined temporary tables
4. UPDATE and DELETE operations on user-defined temporary tables

Undo logs are assigned as needed. For example, a transaction that performs INSERT, UPDATE, and DELETE operations on regular and temporary tables requires a full assignment of four undo logs. A transaction that performs only INSERT operations on regular tables requires a single undo log.

A transaction that performs operations on regular tables is assigned undo logs from an assigned undo tablespace rollback segment. A transaction that performs operations on temporary tables is assigned undo logs from an assigned global temporary tablespace rollback segment.

An undo log assigned to a transaction remains tied to the transaction for its duration. For example, an undo log assigned to a transaction for an INSERT operation on a regular table is used for all INSERT operations on regular tables performed by that transaction.

Given the factors described above, the following formulas can be used to estimate the number of concurrent read-write transactions that InnoDB is capable of supporting.

Note

A transaction can encounter a concurrent transaction limit error before reaching the number of concurrent read-write transactions that InnoDB is capable of supporting. This occurs when a rollback segment assigned to a transaction runs out of undo slots. In such cases, try rerunning the transaction.

When transactions perform operations on temporary tables, the number of concurrent read-write transactions that InnoDB is capable of supporting is constrained by the number of rollback segments allocated to the global temporary tablespace, which is 128 by default.

- If each transaction performs either an INSERT **or** an UPDATE or DELETE operation, the number of concurrent read-write transactions that InnoDB is capable of supporting is:

```
(innodb_page_size / 16) * innodb_rollback_segments * number of undo tablespaces
```

- If each transaction performs an INSERT **and** an UPDATE or DELETE operation, the number of concurrent read-write transactions that InnoDB is capable of supporting is:

```
(innodb_page_size / 16 / 2) * innodb_rollback_segments * number of undo tablespaces
```

- If each transaction performs an `INSERT` operation on a temporary table, the number of concurrent read-write transactions that `InnoDB` is capable of supporting is:

```
(innodb_page_size / 16) * innodb_rollback_segments
```

- If each transaction performs an `INSERT` **and** an `UPDATE` or `DELETE` operation on a temporary table, the number of concurrent read-write transactions that `InnoDB` is capable of supporting is:

```
(innodb_page_size / 16 / 2) * innodb_rollback_segments
```

15.7 InnoDB Locking and Transaction Model

To implement a large-scale, busy, or highly reliable database application, to port substantial code from a different database system, or to tune MySQL performance, it is important to understand `InnoDB` locking and the `InnoDB` transaction model.

This section discusses several topics related to `InnoDB` locking and the `InnoDB` transaction model with which you should be familiar.

- Section 15.7.1, “InnoDB Locking” describes lock types used by `InnoDB`.
- Section 15.7.2, “InnoDB Transaction Model” describes transaction isolation levels and the locking strategies used by each. It also discusses the use of `autocommit`, consistent non-locking reads, and locking reads.
- Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB” discusses specific types of locks set in `InnoDB` for various statements.
- Section 15.7.4, “Phantom Rows” describes how `InnoDB` uses next-key locking to avoid phantom rows.
- Section 15.7.5, “Deadlocks in InnoDB” provides a deadlock example, discusses deadlock detection and rollback, and provides tips for minimizing and handling deadlocks in `InnoDB`.

15.7.1 InnoDB Locking

This section describes lock types used by `InnoDB`.

- Shared and Exclusive Locks
- Intention Locks
- Record Locks
- Gap Locks
- Next-Key Locks
- Insert Intention Locks
- AUTO-INC Locks
- Predicate Locks for Spatial Indexes

Shared and Exclusive Locks

InnoDB implements standard row-level locking where there are two types of locks, shared (S) locks and exclusive (X) locks.

- A shared (S) lock permits the transaction that holds the lock to read a row.
- An exclusive (X) lock permits the transaction that holds the lock to update or delete a row.

If transaction T1 holds a shared (S) lock on row r, then requests from some distinct transaction T2 for a lock on row r are handled as follows:

- A request by T2 for an S lock can be granted immediately. As a result, both T1 and T2 hold an S lock on r.
- A request by T2 for an X lock cannot be granted immediately.

If a transaction T1 holds an exclusive (X) lock on row r, a request from some distinct transaction T2 for a lock of either type on r cannot be granted immediately. Instead, transaction T2 has to wait for transaction T1 to release its lock on row r.

Intention Locks

InnoDB supports *multiple granularity locking* which permits coexistence of row locks and table locks. For example, a statement such as LOCK TABLES ... WRITE takes an exclusive lock (an X lock) on the specified table. To make locking at multiple granularity levels practical, InnoDB uses intention locks. Intention locks are table-level locks that indicate which type of lock (shared or exclusive) a transaction requires later for a row in a table. There are two types of intention locks:

- An intention shared lock (IS) indicates that a transaction intends to set a *shared* lock on individual rows in a table.
- An intention exclusive lock (IX) indicates that a transaction intends to set an exclusive lock on individual rows in a table.

For example, SELECT ... FOR SHARE sets an IS lock, and SELECT ... FOR UPDATE sets an IX lock.

The intention locking protocol is as follows:

- Before a transaction can acquire a shared lock on a row in a table, it must first acquire an IS lock or stronger on the table.
- Before a transaction can acquire an exclusive lock on a row in a table, it must first acquire an IX lock on the table.

Table-level lock type compatibility is summarized in the following matrix.

	X	IX	S	IS
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible
S	Conflict	Conflict	Compatible	Compatible
IS	Conflict	Compatible	Compatible	Compatible

A lock is granted to a requesting transaction if it is compatible with existing locks, but not if it conflicts with existing locks. A transaction waits until the conflicting existing lock is released. If a lock request conflicts with an existing lock and cannot be granted because it would cause deadlock, an error occurs.

Intention locks do not block anything except full table requests (for example, `LOCK TABLES ... WRITE`). The main purpose of intention locks is to show that someone is locking a row, or going to lock a row in the table.

Transaction data for an intention lock appears similar to the following in `SHOW ENGINE INNODB STATUS` and InnoDB monitor output:

```
TABLE LOCK table `test`.`t` trx id 10080 lock mode IX
```

Record Locks

A record lock is a lock on an index record. For example, `SELECT c1 FROM t WHERE c1 = 10 FOR UPDATE;` prevents any other transaction from inserting, updating, or deleting rows where the value of `t.c1` is `10`.

Record locks always lock index records, even if a table is defined with no indexes. For such cases, `InnoDB` creates a hidden clustered index and uses this index for record locking. See Section 15.6.2.1, “Clustered and Secondary Indexes”.

Transaction data for a record lock appears similar to the following in `SHOW ENGINE INNODB STATUS` and InnoDB monitor output:

```
RECORD LOCKS space id 58 page no 3 n bits 72 index `PRIMARY` of table `test`.`t`
trx id 10078 lock_mode X locks rec but not gap
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 00000000274f; asc     'O;;
 2: len 7; hex b60000019d0110; asc        ;;
```

Gap Locks

A gap lock is a lock on a gap between index records, or a lock on the gap before the first or after the last index record. For example, `SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE;` prevents other transactions from inserting a value of `15` into column `t.c1`, whether or not there was already any such value in the column, because the gaps between all existing values in the range are locked.

A gap might span a single index value, multiple index values, or even be empty.

Gap locks are part of the tradeoff between performance and concurrency, and are used in some transaction isolation levels and not others.

Gap locking is not needed for statements that lock rows using a unique index to search for a unique row. (This does not include the case that the search condition includes only some columns of a multiple-column unique index; in that case, gap locking does occur.) For example, if the `id` column has a unique index, the following statement uses only an index-record lock for the row having `id` value 100 and it does not matter whether other sessions insert rows in the preceding gap:

```
SELECT * FROM child WHERE id = 100;
```

If `id` is not indexed or has a nonunique index, the statement does lock the preceding gap.

It is also worth noting here that conflicting locks can be held on a gap by different transactions. For example, transaction A can hold a shared gap lock (gap S-lock) on a gap while transaction B holds an exclusive gap lock (gap X-lock) on the same gap. The reason conflicting gap locks are allowed is that if a record is purged from an index, the gap locks held on the record by different transactions must be merged.

Gap locks in `InnoDB` are “purely inhibitive”, which means that their only purpose is to prevent other transactions from inserting to the gap. Gap locks can co-exist. A gap lock taken by one transaction does not prevent another transaction from taking a gap lock on the same gap. There is no difference between shared and exclusive gap locks. They do not conflict with each other, and they perform the same function.

Gap locking can be disabled explicitly. This occurs if you change the transaction isolation level to `READ COMMITTED`. Under these circumstances, gap locking is disabled for searches and index scans and is used only for foreign-key constraint checking and duplicate-key checking.

There are also other effects of using the `READ COMMITTED` isolation level. Record locks for nonmatching rows are released after MySQL has evaluated the `WHERE` condition. For `UPDATE` statements, `InnoDB` does a “semi-consistent” read, such that it returns the latest committed version to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`.

Next-Key Locks

A next-key lock is a combination of a record lock on the index record and a gap lock on the gap before the index record.

`InnoDB` performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. A next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record `R` in an index, another session cannot insert a new index record in the gap immediately before `R` in the index order.

Suppose that an index contains the values 10, 11, 13, and 20. The possible next-key locks for this index cover the following intervals, where a round bracket denotes exclusion of the interval endpoint and a square bracket denotes inclusion of the endpoint:

```
(negative infinity, 10]
(10, 11]
(11, 13]
(13, 20]
(20, positive infinity)
```

For the last interval, the next-key lock locks the gap above the largest value in the index and the “supremum” pseudo-record having a value higher than any value actually in the index. The supremum is not a real index record, so, in effect, this next-key lock locks only the gap following the largest index value.

By default, `InnoDB` operates in `REPEATABLE READ` transaction isolation level. In this case, `InnoDB` uses next-key locks for searches and index scans, which prevents phantom rows (see Section 15.7.4, “Phantom Rows”).

Transaction data for a next-key lock appears similar to the following in `SHOW ENGINE INNODB STATUS` and InnoDB monitor output:

```
RECORD LOCKS space id 58 page no 3 n bits 72 index `PRIMARY` of table `test`.`t`
trx id 10080 lock_mode X
```

```
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info bits 0
 0: len 8; hex 73757072656d756d; asc supremum;;

Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 00000000274f; asc     'O;;
 2: len 7; hex b60000019d0110; asc        ;;
```

Insert Intention Locks

An insert intention lock is a type of gap lock set by `INSERT` operations prior to row insertion. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6, respectively, each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

The following example demonstrates a transaction taking an insert intention lock prior to obtaining an exclusive lock on the inserted record. The example involves two clients, A and B.

Client A creates a table containing two index records (90 and 102) and then starts a transaction that places an exclusive lock on index records with an ID greater than 100. The exclusive lock includes a gap lock before record 102:

```
mysql> CREATE TABLE child (id int(11) NOT NULL, PRIMARY KEY(id)) ENGINE=InnoDB;
mysql> INSERT INTO child (id) values (90),(102);

mysql> START TRANSACTION;
mysql> SELECT * FROM child WHERE id > 100 FOR UPDATE;
+-----+
| id  |
+-----+
| 102 |
+-----+
```

Client B begins a transaction to insert a record into the gap. The transaction takes an insert intention lock while it waits to obtain an exclusive lock.

```
mysql> START TRANSACTION;
mysql> INSERT INTO child (id) VALUES (101);
```

Transaction data for an insert intention lock appears similar to the following in `SHOW ENGINE INNODB STATUS` and InnoDB monitor output:

```
RECORD LOCKS space id 31 page no 3 n bits 72 index `PRIMARY` of table `test`.`child`
trx id 8731 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 80000066; asc    f;;
 1: len 6; hex 000000002215; asc     " ;;
 2: len 7; hex 9000000172011c; asc     r  ;;...
```

AUTO-INC Locks

An `AUTO-INC` lock is a special table-level lock taken by transactions inserting into tables with `AUTO_INCREMENT` columns. In the simplest case, if one transaction is inserting values into the table, any other transactions must wait to do their own inserts into that table, so that rows inserted by the first transaction receive consecutive primary key values.

The `innodb_autoinc_lock_mode` configuration option controls the algorithm used for auto-increment locking. It allows you to choose how to trade off between predictable sequences of auto-increment values and maximum concurrency for insert operations.

For more information, see Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”.

Predicate Locks for Spatial Indexes

`InnoDB` supports `SPATIAL` indexing of columns containing spatial columns (see Section 11.5.9, “Optimizing Spatial Analysis”).

To handle locking for operations involving `SPATIAL` indexes, next-key locking does not work well to support `REPEATABLE READ` or `SERIALIZABLE` transaction isolation levels. There is no absolute ordering concept in multidimensional data, so it is not clear which is the “next” key.

To enable support of isolation levels for tables with `SPATIAL` indexes, `InnoDB` uses predicate locks. A `SPATIAL` index contains minimum bounding rectangle (MBR) values, so `InnoDB` enforces consistent read on the index by setting a predicate lock on the MBR value used for a query. Other transactions cannot insert or modify a row that would match the query condition.

15.7.2 InnoDB Transaction Model

In the `InnoDB` transaction model, the goal is to combine the best properties of a multi-versioning database with traditional two-phase locking. `InnoDB` performs locking at the row level and runs queries as nonlocking consistent reads by default, in the style of Oracle. The lock information in `InnoDB` is stored space-efficiently so that lock escalation is not needed. Typically, several users are permitted to lock every row in `InnoDB` tables, or any random subset of the rows, without causing `InnoDB` memory exhaustion.

15.7.2.1 Transaction Isolation Levels

Transaction isolation is one of the foundations of database processing. Isolation is the I in the acronym ACID; the isolation level is the setting that fine-tunes the balance between performance and reliability, consistency, and reproducibility of results when multiple transactions are making changes and performing queries at the same time.

`InnoDB` offers all four transaction isolation levels described by the SQL:1992 standard: `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ`, and `SERIALIZABLE`. The default isolation level for `InnoDB` is `REPEATABLE READ`.

A user can change the isolation level for a single session or for all subsequent connections with the `SET TRANSACTION` statement. To set the server's default isolation level for all connections, use the `--transaction-isolation` option on the command line or in an option file. For detailed information about isolation levels and level-setting syntax, see Section 13.3.7, “SET TRANSACTION Syntax”.

`InnoDB` supports each of the transaction isolation levels described here using different locking strategies. You can enforce a high degree of consistency with the default `REPEATABLE READ` level, for operations on crucial data where ACID compliance is important. Or you can relax the consistency rules with `READ COMMITTED` or even `READ UNCOMMITTED`, in situations such as bulk reporting where precise consistency and repeatable results are less important than minimizing the amount of overhead for locking. `SERIALIZABLE` enforces even stricter rules than `REPEATABLE READ`, and is used mainly in specialized situations, such as with XA transactions and for troubleshooting issues with concurrency and deadlocks.

The following list describes how MySQL supports the different transaction levels. The list goes from the most commonly used level to the least used.

- `REPEATABLE READ`

  This is the default isolation level for `InnoDB`. Consistent reads within the same transaction read the snapshot established by the first read. This means that if you issue several plain (nonlocking) `SELECT` statements within the same transaction, these `SELECT` statements are consistent also with respect to each other. See Section 15.7.2.3, “Consistent Nonlocking Reads”.

  For locking reads (`SELECT` with `FOR UPDATE` or `FOR SHARE`), `UPDATE`, and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition, or a range-type search condition.

  - For a unique index with a unique search condition, `InnoDB` locks only the index record found, not the gap before it.
  - For other search conditions, `InnoDB` locks the index range scanned, using gap locks or next-key locks to block insertions by other sessions into the gaps covered by the range. For information about gap locks and next-key locks, see Section 15.7.1, “InnoDB Locking”.

- `READ COMMITTED`

  Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. For information about consistent reads, see Section 15.7.2.3, “Consistent Nonlocking Reads”.

  For locking reads (`SELECT` with `FOR UPDATE` or `FOR SHARE`), `UPDATE` statements, and `DELETE` statements, `InnoDB` locks only index records, not the gaps before them, and thus permits the free insertion of new records next to locked records. Gap locking is only used for foreign-key constraint checking and duplicate-key checking.

  Because gap locking is disabled, phantom problems may occur, as other sessions can insert new rows into the gaps. For information about phantoms, see Section 15.7.4, “Phantom Rows”.

  Only row-based binary logging is supported with the `READ COMMITTED` isolation level. If you use `READ COMMITTED` with `binlog_format=MIXED`, the server automatically uses row-based logging.

  Using `READ COMMITTED` has additional effects:

  - For `UPDATE` or `DELETE` statements, `InnoDB` holds locks only for rows that it updates or deletes. Record locks for nonmatching rows are released after MySQL has evaluated the `WHERE` condition. This greatly reduces the probability of deadlocks, but they can still happen.
  - For `UPDATE` statements, if a row is already locked, `InnoDB` performs a “semi-consistent” read, returning the latest committed version to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`. If the row matches (must be updated), MySQL reads the row again and this time `InnoDB` either locks it or waits for a lock on it.

  Consider the following example, beginning with this table:

```
CREATE TABLE t (a INT NOT NULL, b INT) ENGINE = InnoDB;
INSERT INTO t VALUES (1,2),(2,3),(3,2),(4,3),(5,2);
COMMIT;
```

  In this case, the table has no indexes, so searches and index scans use the hidden clustered index for record locking (see Section 15.6.2.1, “Clustered and Secondary Indexes”) rather than indexed columns.

  Suppose that one session performs an `UPDATE` using these statements:

```
```

```
# Session A
START TRANSACTION;
UPDATE t SET b = 5 WHERE b = 3;
```

Suppose also that a second session performs an UPDATE by executing these statements following those of the first session:

```
# Session B
UPDATE t SET b = 4 WHERE b = 2;
```

As InnoDB executes each UPDATE, it first acquires an exclusive lock for each row, and then determines whether to modify it. If InnoDB does not modify the row, it releases the lock. Otherwise, InnoDB retains the lock until the end of the transaction. This affects transaction processing as follows.

When using the default REPEATABLE READ isolation level, the first UPDATE acquires an x-lock on each row that it reads and does not release any of them:

```
x-lock(1,2); retain x-lock
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); retain x-lock
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); retain x-lock
```

The second UPDATE blocks as soon as it tries to acquire any locks (because first update has retained locks on all rows), and does not proceed until the first UPDATE commits or rolls back:

```
x-lock(1,2); block and wait for first UPDATE to commit or roll back
```

If READ COMMITTED is used instead, the first UPDATE acquires an x-lock on each row that it reads and releases those for rows that it does not modify:

```
x-lock(1,2); unlock(1,2)
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); unlock(3,2)
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); unlock(5,2)
```

For the second UPDATE, InnoDB does a “semi-consistent” read, returning the latest committed version of each row that it reads to MySQL so that MySQL can determine whether the row matches the WHERE condition of the UPDATE:

```
x-lock(1,2); update(1,2) to (1,4); retain x-lock
x-lock(2,3); unlock(2,3)
x-lock(3,2); update(3,2) to (3,4); retain x-lock
x-lock(4,3); unlock(4,3)
x-lock(5,2); update(5,2) to (5,4); retain x-lock
```

However, if the WHERE condition includes an indexed column, and InnoDB uses the index, only the indexed column is considered when taking and retaining record locks. In the following example, the first UPDATE takes and retains an x-lock on each row where b = 2. The second UPDATE blocks when it tries to acquire x-locks on the same records, as it also uses the index defined on column b.

```
CREATE TABLE t (a INT NOT NULL, b INT, c INT, INDEX (b)) ENGINE = InnoDB;
INSERT INTO t VALUES (1,2,3),(2,2,4);
COMMIT;
```

```
# Session A
START TRANSACTION;
UPDATE t SET b = 3 WHERE b = 2 AND c = 3;

# Session B
UPDATE t SET b = 4 WHERE b = 2 AND c = 4;
```

The effects of using the `READ COMMITTED` isolation level are the same as enabling the deprecated `innodb_locks_unsafe_for_binlog` configuration option, with these exceptions:

- Enabling `innodb_locks_unsafe_for_binlog` is a global setting and affects all sessions, whereas the isolation level can be set globally for all sessions, or individually per session.
- `innodb_locks_unsafe_for_binlog` can be set only at server startup, whereas the isolation level can be set at startup or changed at runtime.

`READ COMMITTED` therefore offers finer and more flexible control than `innodb_locks_unsafe_for_binlog`.

- `READ UNCOMMITTED`

  `SELECT` statements are performed in a nonlocking fashion, but a possible earlier version of a row might be used. Thus, using this isolation level, such reads are not consistent. This is also called a dirty read. Otherwise, this isolation level works like `READ COMMITTED`.

- `SERIALIZABLE`

  This level is like `REPEATABLE READ`, but `InnoDB` implicitly converts all plain `SELECT` statements to `SELECT ... FOR SHARE` if `autocommit` is disabled. If `autocommit` is enabled, the `SELECT` is its own transaction. It therefore is known to be read only and can be serialized if performed as a consistent (nonlocking) read and need not block for other transactions. (To force a plain `SELECT` to block if other transactions have modified the selected rows, disable `autocommit`.)

15.7.2.2 autocommit, Commit, and Rollback

In `InnoDB`, all user activity occurs inside a transaction. If `autocommit` mode is enabled, each SQL statement forms a single transaction on its own. By default, MySQL starts the session for each new connection with `autocommit` enabled, so MySQL does a commit after each SQL statement if that statement did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error. See Section 15.20.4, “InnoDB Error Handling”.

A session that has `autocommit` enabled can perform a multiple-statement transaction by starting it with an explicit `START TRANSACTION` or `BEGIN` statement and ending it with a `COMMIT` or `ROLLBACK` statement. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

If `autocommit` mode is disabled within a session with `SET autocommit = 0`, the session always has a transaction open. A `COMMIT` or `ROLLBACK` statement ends the current transaction and a new one starts.

If a session that has `autocommit` disabled ends without explicitly committing the final transaction, MySQL rolls back that transaction.

Some statements implicitly end a transaction, as if you had done a `COMMIT` before executing the statement. For details, see Section 13.3.3, “Statements That Cause an Implicit Commit”.

A `COMMIT` means that the changes made in the current transaction are made permanent and become visible to other sessions. A `ROLLBACK` statement, on the other hand, cancels all modifications made by the current transaction. Both `COMMIT` and `ROLLBACK` release all `InnoDB` locks that were set during the current transaction.

Grouping DML Operations with Transactions

By default, connection to the MySQL server begins with autocommit mode enabled, which automatically commits every SQL statement as you execute it. This mode of operation might be unfamiliar if you have experience with other database systems, where it is standard practice to issue a sequence of DML statements and commit them or roll them back all together.

To use multiple-statement transactions, switch autocommit off with the SQL statement `SET autocommit = 0` and end each transaction with `COMMIT` or `ROLLBACK` as appropriate. To leave autocommit on, begin each transaction with `START TRANSACTION` and end it with `COMMIT` or `ROLLBACK`. The following example shows two transactions. The first is committed; the second is rolled back.

```
shell> mysql test
```

```
mysql> CREATE TABLE customer (a INT, b CHAR (20), INDEX (a));
Query OK, 0 rows affected (0.00 sec)
mysql> -- Do a transaction with autocommit turned on.
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> -- Do another transaction with autocommit turned off.
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO customer VALUES (20, 'Paul');
Query OK, 1 row affected (0.00 sec)
mysql> DELETE FROM customer WHERE b = 'Heikki';
Query OK, 1 row affected (0.00 sec)
mysql> -- Now we undo those last 2 inserts and the delete.
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM customer;
+------+--------+
| a    | b      |
+------+--------+
|   10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
mysql>
```

Transactions in Client-Side Languages

In APIs such as PHP, Perl DBI, JDBC, ODBC, or the standard C call interface of MySQL, you can send transaction control statements such as `COMMIT` to the MySQL server as strings just like any other SQL statements such as `SELECT` or `INSERT`. Some APIs also offer separate special transaction commit and rollback functions or methods.

15.7.2.3 Consistent Nonlocking Reads

A consistent read means that `InnoDB` uses multi-versioning to present to a query a snapshot of the database at a point in time. The query sees the changes made by transactions that committed before that point of time, and no changes made by later or uncommitted transactions. The exception to this rule is that the query sees the changes made by earlier statements within the same transaction. This exception causes the following anomaly: If you update some rows in a table, a `SELECT` sees the latest version of the updated rows, but it might also see older versions of any rows. If other sessions simultaneously update the same table, the anomaly means that you might see the table in a state that never existed in the database.

If the transaction isolation level is `REPEATABLE READ` (the default level), all consistent reads within the same transaction read the snapshot established by the first such read in that transaction. You can get a fresher snapshot for your queries by committing the current transaction and after that issuing new queries.

With `READ COMMITTED` isolation level, each consistent read within a transaction sets and reads its own fresh snapshot.

Consistent read is the default mode in which `InnoDB` processes `SELECT` statements in `READ COMMITTED` and `REPEATABLE READ` isolation levels. A consistent read does not set any locks on the tables it accesses, and therefore other sessions are free to modify those tables at the same time a consistent read is being performed on the table.

Suppose that you are running in the default `REPEATABLE READ` isolation level. When you issue a consistent read (that is, an ordinary `SELECT` statement), `InnoDB` gives your transaction a timepoint according to which your query sees the database. If another transaction deletes a row and commits after your timepoint was assigned, you do not see the row as having been deleted. Inserts and updates are treated similarly.

Note

The snapshot of the database state applies to `SELECT` statements within a transaction, not necessarily to DML statements. If you insert or modify some rows and then commit that transaction, a `DELETE` or `UPDATE` statement issued from another concurrent `REPEATABLE READ` transaction could affect those just-committed rows, even though the session could not query them. If a transaction does update or delete rows committed by a different transaction, those changes do become visible to the current transaction. For example, you might encounter a situation like the following:

```
SELECT COUNT(c1) FROM t1 WHERE c1 = 'xyz';
-- Returns 0: no rows match.
DELETE FROM t1 WHERE c1 = 'xyz';
-- Deletes several rows recently committed by other transaction.

SELECT COUNT(c2) FROM t1 WHERE c2 = 'abc';
-- Returns 0: no rows match.
UPDATE t1 SET c2 = 'cba' WHERE c2 = 'abc';
-- Affects 10 rows: another txn just committed 10 rows with 'abc' values.
SELECT COUNT(c2) FROM t1 WHERE c2 = 'cba';
-- Returns 10: this txn can now see the rows it just updated.
```

You can advance your timepoint by committing your transaction and then doing another `SELECT` or `START TRANSACTION WITH CONSISTENT SNAPSHOT`.

This is called *multi-versioned concurrency control*.

In the following example, session A sees the row inserted by B only when B has committed the insert and A has committed as well, so that the timepoint is advanced past the commit of B.

```
             Session A              Session B

           SET autocommit=0;      SET autocommit=0;
time
|          SELECT * FROM t;
|          empty set
|                                 INSERT INTO t VALUES (1, 2);
|
v          SELECT * FROM t;
           empty set
```

```
                                COMMIT;

          SELECT * FROM t;
          empty set

          COMMIT;

          SELECT * FROM t;
          ---------------------
          |    1    |    2    |
          ---------------------
```

If you want to see the “freshest” state of the database, use either the `READ COMMITTED` isolation level or a locking read:

```
SELECT * FROM t FOR SHARE;
```

With `READ COMMITTED` isolation level, each consistent read within a transaction sets and reads its own fresh snapshot. With `FOR SHARE`, a locking read occurs instead: A `SELECT` blocks until the transaction containing the freshest rows ends (see Section 15.7.2.4, “Locking Reads”).

Consistent read does not work over certain DDL statements:

- Consistent read does not work over `DROP TABLE`, because MySQL cannot use a table that has been dropped and `InnoDB` destroys the table.
- Consistent read does not work over `ALTER TABLE`, because that statement makes a temporary copy of the original table and deletes the original table when the temporary copy is built. When you reissue a consistent read within a transaction, rows in the new table are not visible because those rows did not exist when the transaction's snapshot was taken. In this case, the transaction returns an error: `ER_TABLE_DEF_CHANGED`, “Table definition has changed, please retry transaction”.

The type of read varies for selects in clauses like `INSERT INTO ... SELECT`, `UPDATE ... (SELECT)`, and `CREATE TABLE ... SELECT` that do not specify `FOR UPDATE` or `FOR SHARE`:

- By default, `InnoDB` uses stronger locks and the `SELECT` part acts like `READ COMMITTED`, where each consistent read, even within the same transaction, sets and reads its own fresh snapshot.
- To use a consistent read in such cases, set the isolation level of the transaction to `READ UNCOMMITTED`, `READ COMMITTED`, or `REPEATABLE READ` (that is, anything other than `SERIALIZABLE`). In this case, no locks are set on rows read from the selected table.

15.7.2.4 Locking Reads

If you query data and then insert or update related data within the same transaction, the regular `SELECT` statement does not give enough protection. Other transactions can update or delete the same rows you just queried. `InnoDB` supports two types of locking reads that offer extra safety:

- `SELECT ... FOR SHARE`

  Sets a shared mode lock on any rows that are read. Other sessions can read the rows, but cannot modify them until your transaction commits. If any of these rows were changed by another transaction that has not yet committed, your query waits until that transaction ends and then uses the latest values.

Note

`SELECT ... FOR SHARE` is a replacement for `SELECT ... LOCK IN SHARE MODE`, but `LOCK IN SHARE MODE` remains available for backward

compatibility. The statements are equivalent. However, `FOR SHARE` supports `OF table_name`, `NOWAIT`, and `SKIP LOCKED` options. See Locking Read Concurrency with NOWAIT and SKIP LOCKED.

- `SELECT ... FOR UPDATE`

  For index records the search encounters, locks the rows and any associated index entries, the same as if you issued an `UPDATE` statement for those rows. Other transactions are blocked from updating those rows, from doing `SELECT ... FOR SHARE`, or from reading the data in certain transaction isolation levels. Consistent reads ignore any locks set on the records that exist in the read view. (Old versions of a record cannot be locked; they are reconstructed by applying undo logs on an in-memory copy of the record.)

These clauses are primarily useful when dealing with tree-structured or graph-structured data, either in a single table or split across multiple tables. You traverse edges or tree branches from one place to another, while reserving the right to come back and change any of these “pointer” values.

All locks set by `FOR SHARE` and `FOR UPDATE` queries are released when the transaction is committed or rolled back.

Note

Locking reads are only possible when autocommit is disabled (either by beginning transaction with `START TRANSACTION` or by setting `autocommit` to 0.

A locking read clause in an outer statement does not lock the rows of a table in a nested subquery unless a locking read clause is also specified in the subquery. For example, the following statement does not lock rows in table `t2`.

```
SELECT * FROM t1 WHERE c1 = (SELECT c1 FROM t2) FOR UPDATE;
```

To lock rows in table `t2`, add a locking read clause to the subquery:

```
SELECT * FROM t1 WHERE c1 = (SELECT c1 FROM t2 FOR UPDATE) FOR UPDATE;
```

Locking Read Examples

Suppose that you want to insert a new row into a table `child`, and make sure that the child row has a parent row in table `parent`. Your application code can ensure referential integrity throughout this sequence of operations.

First, use a consistent read to query the table `PARENT` and verify that the parent row exists. Can you safely insert the child row to table `CHILD`? No, because some other session could delete the parent row in the moment between your `SELECT` and your `INSERT`, without you being aware of it.

To avoid this potential issue, perform the `SELECT` using `FOR SHARE`:

```
SELECT * FROM parent WHERE NAME = 'Jones' FOR SHARE;
```

After the `FOR SHARE` query returns the parent `'Jones'`, you can safely add the child record to the `CHILD` table and commit the transaction. Any transaction that tries to acquire an exclusive lock in the applicable row in the `PARENT` table waits until you are finished, that is, until the data in all tables is in a consistent state.

For another example, consider an integer counter field in a table `CHILD_CODES`, used to assign a unique identifier to each child added to table `CHILD`. Do not use either consistent read or a shared mode read to

read the present value of the counter, because two users of the database could see the same value for the counter, and a duplicate-key error occurs if two transactions attempt to add rows with the same identifier to the `CHILD` table.

Here, `FOR SHARE` is not a good solution because if two users read the counter at the same time, at least one of them ends up in deadlock when it attempts to update the counter.

To implement reading and incrementing the counter, first perform a locking read of the counter using `FOR UPDATE`, and then increment the counter. For example:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

A `SELECT ... FOR UPDATE` reads the latest available data, setting exclusive locks on each row it reads. Thus, it sets the same locks a searched SQL `UPDATE` would set on the rows.

The preceding description is merely an example of how `SELECT ... FOR UPDATE` works. In MySQL, the specific task of generating a unique identifier actually can be accomplished using only a single access to the table:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

The `SELECT` statement merely retrieves the identifier information (specific to the current connection). It does not access any table.

Locking Read Concurrency with NOWAIT and SKIP LOCKED

If a row is locked by a transaction, a `SELECT ... FOR UPDATE` or `SELECT ... FOR SHARE` transaction that requests the same locked row must wait until the blocking transaction releases the row lock. This behavior prevents transactions from updating or deleting rows that are queried for updates by other transactions. However, waiting for a row lock to be released is not necessary if you want the query to return immediately when a requested row is locked, or if excluding locked rows from the result set is acceptable.

To avoid waiting for other transactions to release row locks, `NOWAIT` and `SKIP LOCKED` options may be used with `SELECT ... FOR UPDATE` or `SELECT ... FOR SHARE` locking read statements.

- `NOWAIT`

  A locking read that uses `NOWAIT` never waits to acquire a row lock. The query executes immediately, failing with an error if a requested row is locked.

- `SKIP LOCKED`

  A locking read that uses `SKIP LOCKED` never waits to acquire a row lock. The query executes immediately, removing locked rows from the result set.

Note

Queries that skip locked rows return an inconsistent view of the data. `SKIP LOCKED` is therefore not suitable for general transactional work. However, it may be used to avoid lock contention when multiple sessions access the same queue-like table.

`NOWAIT` and `SKIP LOCKED` only apply to row-level locks.

Statements that use `NOWAIT` or `SKIP LOCKED` are unsafe for statement based replication.

The following example demonstrates `NOWAIT` and `SKIP LOCKED`. Session 1 starts a transaction that takes a row lock on a single record. Session 2 attempts a locking read on the same record using the `NOWAIT` option. Because the requested row is locked by Session 1, the locking read returns immediately with an error. In Session 3, the locking read with `SKIP LOCKED` returns the requested rows except for the row that is locked by Session 1.

```
# Session 1:

mysql> CREATE TABLE t (i INT, PRIMARY KEY (i)) ENGINE = InnoDB;

mysql> INSERT INTO t (i) VALUES(1),(2),(3);

mysql> START TRANSACTION;

mysql> SELECT * FROM t WHERE i = 2 FOR UPDATE;
+---+
| i |
+---+
| 2 |
+---+

# Session 2:

mysql> START TRANSACTION;

mysql> SELECT * FROM t WHERE i = 2 FOR UPDATE NOWAIT;
ERROR 3572 (HY000): Do not wait for lock.

# Session 3:

mysql> START TRANSACTION;

mysql> SELECT * FROM t FOR UPDATE SKIP LOCKED;
+---+
| i |
+---+
| 1 |
| 3 |
+---+
```

15.7.3 Locks Set by Different SQL Statements in InnoDB

A locking read, an `UPDATE`, or a `DELETE` generally set record locks on every index record that is scanned in the processing of the SQL statement. It does not matter whether there are `WHERE` conditions in the statement that would exclude the row. `InnoDB` does not remember the exact `WHERE` condition, but only knows which index ranges were scanned. The locks are normally next-key locks that also block inserts into the “gap” immediately before the record. However, gap locking can be disabled explicitly, which causes next-key locking not to be used. For more information, see Section 15.7.1, “InnoDB Locking”. The transaction isolation level also can affect which locks are set; see Section 15.7.2.1, “Transaction Isolation Levels”.

If a secondary index is used in a search and index record locks to be set are exclusive, `InnoDB` also retrieves the corresponding clustered index records and sets locks on them.

If you have no indexes suitable for your statement and MySQL must scan the entire table to process the statement, every row of the table becomes locked, which in turn blocks all inserts by other users to the table. It is important to create good indexes so that your queries do not unnecessarily scan many rows.

`InnoDB` sets specific types of locks as follows.

- `SELECT ... FROM` is a consistent read, reading a snapshot of the database and setting no locks unless the transaction isolation level is set to `SERIALIZABLE`. For `SERIALIZABLE` level, the search sets shared next-key locks on the index records it encounters. However, only an index record lock is required for statements that lock rows using a unique index to search for a unique row.

- `SELECT ... FOR UPDATE` and `SELECT ... FOR SHARE` statements that use a unique index acquire locks for scanned rows, and release the locks for rows that do not qualify for inclusion in the result set (for example, if they do not meet the criteria given in the `WHERE` clause). However, in some cases, rows might not be unlocked immediately because the relationship between a result row and its original source is lost during query execution. For example, in a `UNION`, scanned (and locked) rows from a table might be inserted into a temporary table before evaluation whether they qualify for the result set. In this circumstance, the relationship of the rows in the temporary table to the rows in the original table is lost and the latter rows are not unlocked until the end of query execution.

- For locking reads (`SELECT` with `FOR UPDATE` or `FOR SHARE`), `UPDATE`, and `DELETE` statements, the locks that are taken depend on whether the statement uses a unique index with a unique search condition, or a range-type search condition.

  - For a unique index with a unique search condition, `InnoDB` locks only the index record found, not the gap before it.

  - For other search conditions, and for non-unique indexes, `InnoDB` locks the index range scanned, using gap locks or next-key locks to block insertions by other sessions into the gaps covered by the range. For information about gap locks and next-key locks, see Section 15.7.1, “InnoDB Locking”.

- For index records the search encounters, `SELECT ... FOR UPDATE` blocks other sessions from doing `SELECT ... FOR SHARE` or from reading in certain transaction isolation levels. Consistent reads ignore any locks set on the records that exist in the read view.

- `UPDATE ... WHERE ...` sets an exclusive next-key lock on every record the search encounters. However, only an index record lock is required for statements that lock rows using a unique index to search for a unique row.

- When `UPDATE` modifies a clustered index record, implicit locks are taken on affected secondary index records. The `UPDATE` operation also takes shared locks on affected secondary index records when performing duplicate check scans prior to inserting new secondary index records, and when inserting new secondary index records.

- `DELETE FROM ... WHERE ...` sets an exclusive next-key lock on every record the search encounters. However, only an index record lock is required for statements that lock rows using a unique index to search for a unique row.

- `INSERT` sets an exclusive lock on the inserted row. This lock is an index-record lock, not a next-key lock (that is, there is no gap lock) and does not prevent other sessions from inserting into the gap before the inserted row.

  Prior to inserting the row, a type of gap lock called an insert intention gap lock is set. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

  If a duplicate-key error occurs, a shared lock on the duplicate index record is set. This use of a shared lock can result in deadlock should there be multiple sessions trying to insert the same row if another session already has an exclusive lock. This can occur if another session deletes the row. Suppose that an `InnoDB` table `t1` has the following structure:

```
CREATE TABLE t1 (i INT, PRIMARY KEY (i)) ENGINE = InnoDB;
```

Now suppose that three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
ROLLBACK;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 rolls back, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

A similar situation occurs if the table already contains a row with key value 1 and three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;
DELETE FROM t1 WHERE i = 1;
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
COMMIT;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 commits, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

- `INSERT ... ON DUPLICATE KEY UPDATE` differs from a simple `INSERT` in that an exclusive lock rather than a shared lock is placed on the row to be updated when a duplicate-key error occurs. An exclusive index-record lock is taken for a duplicate primary key value. An exclusive next-key lock is taken for a duplicate unique key value.

- `REPLACE` is done like an `INSERT` if there is no collision on a unique key. Otherwise, an exclusive next-key lock is placed on the row to be replaced.

- `INSERT INTO T SELECT ... FROM S WHERE ...` sets an exclusive index record lock (without a gap lock) on each row inserted into `T`. If the transaction isolation level is `READ COMMITTED`, `InnoDB` does the search on `S` as a consistent read (no locks). Otherwise, `InnoDB` sets shared next-key locks on rows from `S`. `InnoDB` has to set locks in the latter case: During roll-forward recovery using a statement-based binary log, every SQL statement must be executed in exactly the same way it was done originally.

  `CREATE TABLE ... SELECT ...` performs the `SELECT` with shared next-key locks or as a consistent read, as for `INSERT ... SELECT`.

  When a `SELECT` is used in the constructs `REPLACE INTO t SELECT ... FROM s WHERE ...` or `UPDATE t ... WHERE col IN (SELECT ... FROM s ...)`, `InnoDB` sets shared next-key locks on rows from table `s`.

- While initializing a previously specified `AUTO_INCREMENT` column on a table, `InnoDB` sets an exclusive lock on the end of the index associated with the `AUTO_INCREMENT` column. In accessing the auto-increment counter, `InnoDB` uses a specific `AUTO-INC` table lock mode where the lock lasts only to the end of the current SQL statement, not to the end of the entire transaction. Other sessions cannot insert into the table while the `AUTO-INC` table lock is held; see Section 15.7.2, “InnoDB Transaction Model”.

  `InnoDB` fetches the value of a previously initialized `AUTO_INCREMENT` column without setting any locks.

- If a `FOREIGN KEY` constraint is defined on a table, any insert, update, or delete that requires the constraint condition to be checked sets shared record-level locks on the records that it looks at to check the constraint. `InnoDB` also sets these locks in the case where the constraint fails.

- `LOCK TABLES` sets table locks, but it is the higher MySQL layer above the `InnoDB` layer that sets these locks. `InnoDB` is aware of table locks if `innodb_table_locks = 1` (the default) and `autocommit = 0`, and the MySQL layer above `InnoDB` knows about row-level locks.

  Otherwise, `InnoDB`'s automatic deadlock detection cannot detect deadlocks where such table locks are involved. Also, because in this case the higher MySQL layer does not know about row-level locks, it is possible to get a table lock on a table where another session currently has row-level locks. However, this does not endanger transaction integrity, as discussed in Section 15.7.5.2, “Deadlock Detection and Rollback”. See also Section 15.6.1.6, “Limits on InnoDB Tables”.

15.7.4 Phantom Rows

The so-called *phantom* problem occurs within a transaction when the same query produces different sets of rows at different times. For example, if a `SELECT` is executed twice, but returns a row the second time that was not returned the first time, the row is a “phantom” row.

Suppose that there is an index on the `id` column of the `child` table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where `id` is bigger than 100. Let the table contain rows having `id` values of 90 and 102. If the locks set on the index records in the scanned range do not lock out inserts made in the gaps (in this case, the gap between 90 and 102), another session can insert a new row into the table with an `id` of 101. If you were to execute the same `SELECT` within the same transaction, you would see a new row with an `id` of 101 (a “phantom”) in the result set returned by the query. If we regard a set of rows as a data item, the new phantom child would violate the isolation principle of transactions that a transaction should be able to run so that the data it has read does not change during the transaction.

To prevent phantoms, `InnoDB` uses an algorithm called *next-key locking* that combines index-row locking with gap locking. `InnoDB` performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record `R` in an index, another session cannot insert a new index record in the gap immediately before `R` in the index order.

When `InnoDB` scans an index, it can also lock the gap after the last record in the index. Just that happens in the preceding example: To prevent any insert into the table where `id` would be bigger than 100, the locks set by `InnoDB` include a lock on the gap following `id` value 102.

You can use next-key locking to implement a uniqueness check in your application: If you read your data in share mode and do not see a duplicate for a row you are going to insert, then you can safely insert your row and know that the next-key lock set on the successor of your row during the read prevents anyone meanwhile inserting a duplicate for your row. Thus, the next-key locking enables you to “lock” the nonexistence of something in your table.

Gap locking can be disabled as discussed in Section 15.7.1, “InnoDB Locking”. This may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled.

15.7.5 Deadlocks in InnoDB

A deadlock is a situation where different transactions are unable to proceed because each holds a lock that the other needs. Because both transactions are waiting for a resource to become available, neither ever release the locks it holds.

A deadlock can occur when transactions lock rows in multiple tables (through statements such as `UPDATE` or `SELECT ... FOR UPDATE`), but in the opposite order. A deadlock can also occur when such statements lock ranges of index records and gaps, with each transaction acquiring some locks but not others due to a timing issue. For a deadlock example, see Section 15.7.5.1, “An InnoDB Deadlock Example”.

To reduce the possibility of deadlocks, use transactions rather than `LOCK TABLES` statements; keep transactions that insert or update data small enough that they do not stay open for long periods of time; when different transactions update multiple tables or large ranges of rows, use the same order of operations (such as `SELECT ... FOR UPDATE`) in each transaction; create indexes on the columns used in `SELECT ... FOR UPDATE` and `UPDATE ... WHERE` statements. The possibility of deadlocks is not affected by the isolation level, because the isolation level changes the behavior of read operations, while

deadlocks occur because of write operations. For more information about avoiding and recovering from deadlock conditions, see Section 15.7.5.3, “How to Minimize and Handle Deadlocks”.

When deadlock detection is enabled (the default) and a deadlock does occur, `InnoDB` detects the condition and rolls back one of the transactions (the victim). If deadlock detection is disabled using the `innodb_deadlock_detect` configuration option, `InnoDB` relies on the `innodb_lock_wait_timeout` setting to roll back transactions in case of a deadlock. Thus, even if your application logic is correct, you must still handle the case where a transaction must be retried. To see the last deadlock in an `InnoDB` user transaction, use the `SHOW ENGINE INNODB STATUS` command. If frequent deadlocks highlight a problem with transaction structure or application error handling, run with the `innodb_print_all_deadlocks` setting enabled to print information about all deadlocks to the `mysqld` error log. For more information about how deadlocks are automatically detected and handled, see Section 15.7.5.2, “Deadlock Detection and Rollback”.

15.7.5.1 An InnoDB Deadlock Example

The following example illustrates how an error can occur when a lock request would cause a deadlock. The example involves two clients, A and B.

First, client A creates a table containing one row, and then begins a transaction. Within the transaction, A obtains an `S` lock on the row by selecting it in share mode:

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);
Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE i = 1 FOR SHARE;
+------+
| i    |
+------+
|    1 |
+------+
```

Next, client B begins a transaction and attempts to delete the row from the table:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM t WHERE i = 1;
```

The delete operation requires an `X` lock. The lock cannot be granted because it is incompatible with the `S` lock that client A holds, so the request goes on the queue of lock requests for the row and client B blocks.

Finally, client A also attempts to delete the row from the table:

```
mysql> DELETE FROM t WHERE i = 1;
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

Deadlock occurs here because client A needs an `X` lock to delete the row. However, that lock request cannot be granted because client B already has a request for an `X` lock and is waiting for client A to release its `S` lock. Nor can the `S` lock held by A be upgraded to an `X` lock because of the prior request by B for an `X`

lock. As a result, InnoDB generates an error for one of the clients and releases its locks. The client returns this error:

```
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

At that point, the lock request for the other client can be granted and it deletes the row from the table.

15.7.5.2 Deadlock Detection and Rollback

When deadlock detection is enabled (the default), InnoDB automatically detects transaction deadlocks and rolls back a transaction or transactions to break the deadlock. InnoDB tries to pick small transactions to roll back, where the size of a transaction is determined by the number of rows inserted, updated, or deleted.

InnoDB is aware of table locks if innodb_table_locks = 1 (the default) and autocommit = 0, and the MySQL layer above it knows about row-level locks. Otherwise, InnoDB cannot detect deadlocks where a table lock set by a MySQL LOCK TABLES statement or a lock set by a storage engine other than InnoDB is involved. Resolve these situations by setting the value of the innodb_lock_wait_timeout system variable.

When InnoDB performs a complete rollback of a transaction, all locks set by the transaction are released. However, if just a single SQL statement is rolled back as a result of an error, some of the locks set by the statement may be preserved. This happens because InnoDB stores row locks in a format such that it cannot know afterward which lock was set by which statement.

If a SELECT calls a stored function in a transaction, and a statement within the function fails, that statement rolls back. Furthermore, if ROLLBACK is executed after that, the entire transaction rolls back.

If the LATEST DETECTED DEADLOCK section of InnoDB Monitor output includes a message stating, "TOO DEEP OR LONG SEARCH IN THE LOCK TABLE WAITS-FOR GRAPH, WE WILL ROLL BACK FOLLOWING TRANSACTION," this indicates that the number of transactions on the wait-for list has reached a limit of 200. A wait-for list that exceeds 200 transactions is treated as a deadlock and the transaction attempting to check the wait-for list is rolled back. The same error may also occur if the locking thread must look at more than 1,000,000 locks owned by transactions on the wait-for list.

For techniques to organize database operations to avoid deadlocks, see Section 15.7.5, "Deadlocks in InnoDB".

Disabling Deadlock Detection

On high concurrency systems, deadlock detection can cause a slowdown when numerous threads wait for the same lock. At times, it may be more efficient to disable deadlock detection and rely on the innodb_lock_wait_timeout setting for transaction rollback when a deadlock occurs. Deadlock detection can be disabled using the innodb_deadlock_detect configuration option.

15.7.5.3 How to Minimize and Handle Deadlocks

This section builds on the conceptual information about deadlocks in Section 15.7.5.2, "Deadlock Detection and Rollback". It explains how to organize database operations to minimize deadlocks and the subsequent error handling required in applications.

Deadlocks are a classic problem in transactional databases, but they are not dangerous unless they are so frequent that you cannot run certain transactions at all. Normally, you must write your applications so that they are always prepared to re-issue a transaction if it gets rolled back because of a deadlock.

InnoDB uses automatic row-level locking. You can get deadlocks even in the case of transactions that just insert or delete a single row. That is because these operations are not really “atomic”; they automatically set locks on the (possibly several) index records of the row inserted or deleted.

You can cope with deadlocks and reduce the likelihood of their occurrence with the following techniques:

- At any time, issue the SHOW ENGINE INNODB STATUS command to determine the cause of the most recent deadlock. That can help you to tune your application to avoid deadlocks.
- If frequent deadlock warnings cause concern, collect more extensive debugging information by enabling the innodb_print_all_deadlocks configuration option. Information about each deadlock, not just the latest one, is recorded in the MySQL error log. Disable this option when you are finished debugging.
- Always be prepared to re-issue a transaction if it fails due to deadlock. Deadlocks are not dangerous. Just try again.
- Keep transactions small and short in duration to make them less prone to collision.
- Commit transactions immediately after making a set of related changes to make them less prone to collision. In particular, do not leave an interactive mysql session open for a long time with an uncommitted transaction.
- If you use locking reads (SELECT ... FOR UPDATE or SELECT ... FOR SHARE), try using a lower isolation level such as READ COMMITTED.
- When modifying multiple tables within a transaction, or different sets of rows in the same table, do those operations in a consistent order each time. Then transactions form well-defined queues and do not deadlock. For example, organize database operations into functions within your application, or call stored routines, rather than coding multiple similar sequences of INSERT, UPDATE, and DELETE statements in different places.
- Add well-chosen indexes to your tables. Then your queries need to scan fewer index records and consequently set fewer locks. Use EXPLAIN SELECT to determine which indexes the MySQL server regards as the most appropriate for your queries.
- Use less locking. If you can afford to permit a SELECT to return data from an old snapshot, do not add the clause FOR UPDATE or FOR SHARE to it. Using the READ COMMITTED isolation level is good here, because each consistent read within the same transaction reads from its own fresh snapshot.
- If nothing else helps, serialize your transactions with table-level locks. The correct way to use LOCK TABLES with transactional tables, such as InnoDB tables, is to begin a transaction with SET autocommit = 0 (not START TRANSACTION) followed by LOCK TABLES, and to not call UNLOCK TABLES until you commit the transaction explicitly. For example, if you need to write to table t1 and read from table t2, you can do this:

  ```
  SET autocommit=0;
  LOCK TABLES t1 WRITE, t2 READ, ...;
  ... do something with tables t1 and t2 here ...
  COMMIT;
  UNLOCK TABLES;
  ```

  Table-level locks prevent concurrent updates to the table, avoiding deadlocks at the expense of less responsiveness for a busy system.
- Another way to serialize transactions is to create an auxiliary “semaphore” table that contains just a single row. Have each transaction update that row before accessing other tables. In that way, all transactions happen in a serial fashion. Note that the InnoDB instant deadlock detection algorithm also

works in this case, because the serializing lock is a row-level lock. With MySQL table-level locks, the timeout method must be used to resolve deadlocks.

15.8 InnoDB Configuration

This section provides configuration information and procedures for `InnoDB` initialization, startup, and various components and features of the `InnoDB` storage engine. For information about optimizing database operations for `InnoDB` tables, see Section 8.5, “Optimizing for InnoDB Tables”.

15.8.1 InnoDB Startup Configuration

The first decisions to make about `InnoDB` configuration involve the configuration of data files, log files, page size, and memory buffers. It is recommended that you define data file, log file, and page size configuration before creating the `InnoDB` instance. Modifying data file or log file configuration after the `InnoDB` instance is created may involve a non-trivial procedure, and page size can only be defined when the `InnoDB` instance is first initialized.

In addition to these topics, this section provides information about specifying `InnoDB` options in a configuration file, viewing `InnoDB` initialization information, and important storage considerations.

- Specifying Options in a MySQL Configuration File
- Viewing InnoDB Initialization Information
- Important Storage Considerations
- System Tablespace Data File Configuration
- Redo Log File Configuration
- Undo Tablespace Configuration
- Global Temporary Tablespace Configuration
- Session Temporary Tablespace Configuration
- Page Size Configuration
- Memory Configuration

Specifying Options in a MySQL Configuration File

Because MySQL uses data file, log file, and page size configuration settings to initialize the `InnoDB` instance, it is recommended that you define these settings in a configuration file that MySQL reads at startup, prior to initializing `InnoDB` for the first time. `InnoDB` is initialized when the MySQL server is started, and the first initialization of `InnoDB` normally occurs the first time you start the MySQL server.

You can place `InnoDB` options in the `[mysqld]` group of any option file that your server reads when it starts. The locations of MySQL option files are described in Section 4.2.2.2, “Using Option Files”.

To make sure that `mysqld` reads options only from a specific file (and `mysqld-auto.cnf`), use the `--defaults-file` option as the first option on the command line when starting the server:

```
mysqld --defaults-file=path_to_configuration_file
```

Viewing InnoDB Initialization Information

To view `InnoDB` initialization information during startup, start `mysqld` from a command prompt. When `mysqld` is started from a command prompt, initialization information is printed to the console.

For example, on Windows, if `mysqld` is located in `C:\Program Files\MySQL\MySQL Server 8.0\bin`, start the MySQL server like this:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld" --console
```

On Unix-like systems, `mysqld` is located in the `bin` directory of your MySQL installation:

```
shell> bin/mysqld --user=mysql &
```

If you do not send server output to the console, check the error log after startup to see the initialization information `InnoDB` printed during the startup process.

For information about starting MySQL using other methods, see Section 2.10.5, “Starting and Stopping MySQL Automatically”.

Note

`InnoDB` does not open all user tables and associated data files at startup. However, `InnoDB` does check for the existence of tablespace files (`*.ibd` files) that are referenced in the data dictionary. If a tablespace file is not found, `InnoDB` logs an error and continues the startup sequence. Tablespace files that are referenced in the redo log may be opened during crash recovery for redo application.

Important Storage Considerations

Review the following storage-related considerations before proceeding with your startup configuration.

- In some cases, database performance improves if the data is not all placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. For example, you can place system tablespace data files and log files on different disks. You can also use raw disk partitions (raw devices) for `InnoDB` data files, which may speed up I/O. See Using Raw Disk Partitions for the System Tablespace.

- `InnoDB` is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. **However, it cannot do so** if the underlying operating system or hardware does not work as advertised. Many operating systems or disk subsystems may delay or reorder write operations to improve performance. On some operating systems, the very `fsync()` system call that should wait until all unwritten data for a file has been flushed might actually return before the data has been flushed to stable storage. Because of this, an operating system crash or a power outage may destroy recently committed data, or in the worst case, even corrupt the database because of write operations having been reordered. If data integrity is important to you, perform some “pull-the-plug” tests before using anything in production. On OS X 10.3 and higher, `InnoDB` uses a special `fcntl()` file flush method. Under Linux, it is advisable to **disable the write-back cache**.

  On ATA/SATA disk drives, a command such `hdparm -W0 /dev/hda` may work to disable the write-back cache. **Beware that some drives or disk controllers may be unable to disable the write-back cache.**

- With regard to `InnoDB` recovery capabilities that protect user data, `InnoDB` uses a file flush technique involving a structure called the doublewrite buffer, which is enabled by default (`innodb_doublewrite=ON`). The doublewrite buffer adds safety to recovery following a crash or power outage, and improves performance on most varieties of Unix by reducing the need for `fsync()` operations. It is recommended that the `innodb_doublewrite` option remains enabled if you are concerned with data integrity or possible failures. For additional information about the doublewrite buffer, see Section 15.11.1, “InnoDB Disk I/O”.

- Before using NFS with InnoDB, review potential issues outlined in Using NFS with MySQL.

System Tablespace Data File Configuration

The innodb_data_file_path startup option defines the name, size, and attributes of InnoDB system tablespace data files. If you do not configure this option prior to initializing the MySQL server, the default behavior is to create a single auto-extending data file, slightly larger than 12MB, named ibdata1:

```
mysql> SHOW VARIABLES LIKE 'innodb_data_file_path';
+-----------------------+------------------------+
| Variable_name         | Value                  |
+-----------------------+------------------------+
| innodb_data_file_path | ibdata1:12M:autoextend |
+-----------------------+------------------------+
```

The full data file specification syntax includes the file name, file size, autoextend attribute, and max attribute:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

File sizes are specified in kilobytes, megabytes, or gigabytes by appending K, M or G to the size value. If specifying the data file size in kilobytes, do so in multiples of 1024. Otherwise, kilobyte values are rounded to nearest megabyte (MB) boundary. The sum of file sizes must be, at a minimum, slightly larger than 12MB.

You can specify more than one data file using a semicolon-separated list. For example:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

The autoextend and max attributes can be used only for the data file that is specified last.

When the autoextend attribute is specified, the data file automatically increases in size by 64MB increments as space is required. The innodb_autoextend_increment variable controls the increment size.

To specify a maximum size for an auto-extending data file, use the max attribute following the autoextend attribute. Use the max attribute only in cases where constraining disk usage is of critical importance. The following configuration permits ibdata1 to grow to a limit of 500MB:

```
[mysqld]
innodb_data_file_path=ibdata1:12M:autoextend:max:500M
```

A minimum file size is enforced for the *first* system tablespace data file to ensure that there is enough space for doublewrite buffer pages. The following table shows minimum file sizes for each InnoDB page size. The default InnoDB page size is 16384 (16KB).

Page Size (innodb_page_size)	**Minimum File Size**
16384 (16KB) or less	3MB
32768 (32KB)	6MB
65536 (64KB)	12MB

If your disk becomes full, you can add a data file on another disk. For instructions, see Resizing the System Tablespace.

The size limit for individual files is determined by your operating system. You can set the file size to more than 4GB on operating systems that support large files. You can also use raw disk partitions as data files. See Using Raw Disk Partitions for the System Tablespace.

InnoDB is not aware of the file system maximum file size, so be cautious on file systems where the maximum file size is a small value such as 2GB.

System tablespace files are created in the data directory by default (datadir). To specify an alternate location, use the innodb_data_home_dir option. For example, to create a system tablespace data file in a directory named myibdata, use this configuration:

```
[mysqld]
innodb_data_home_dir = /myibdata/
innodb_data_file_path=ibdata1:50M:autoextend
```

A trailing slash is required when specifying a value for innodb_data_home_dir. InnoDB does not create directories, so ensure that the specified directory exists before you start the server. Also, ensure sure that the MySQL server has the proper access rights to create files in the directory.

InnoDB forms the directory path for each data file by textually concatenating the value of innodb_data_home_dir to the data file name. If innodb_data_home_dir is not defined, the default value is "./", which is the data directory. (The MySQL server changes its current working directory to the data directory when it begins executing.)

If you specify innodb_data_home_dir as an empty string, you can specify absolute paths for data files listed in the innodb_data_file_path value. The following configuration is equivalent to the preceding one:

```
[mysqld]
innodb_data_home_dir =
innodb_data_file_path=/myibdata/ibdata1:50M:autoextend
```

Redo Log File Configuration

By default, InnoDB creates two 5MB redo log files in the data directory named ib_logfile0 and ib_logfile1.

The following options can be used to modify the default configuration:

- innodb_log_group_home_dir defines directory path to the InnoDB log files (the redo logs). If this option is not configured, InnoDB log files are created in the MySQL data directory (datadir).

  You might use this option to place InnoDB log files in a different physical storage location than InnoDB data files to avoid potential I/O resource conflicts. For example:

  ```
  [mysqld]
  innodb_log_group_home_dir = /dr3/iblogs
  ```

  Note

  InnoDB does not create directories, so make sure that the log directory exists before you start the server. Use the Unix or DOS mkdir command to create any necessary directories.

  Make sure that the MySQL server has the proper access rights to create files in the log directory. More generally, the server must have access rights in any directory where it needs to create log files.

- `innodb_log_files_in_group` defines the number of log files in the log group. The default and recommended value is 2.
- `innodb_log_file_size` defines the size in bytes of each log file in the log group. The combined size of log files (`innodb_log_file_size` * `innodb_log_files_in_group`) cannot exceed a maximum value that is slightly less than 512GB. A pair of 255 GB log files, for example, approaches the limit but does not exceed it. The default log file size is 48MB. Generally, the combined size of the log files should be large enough that the server can smooth out peaks and troughs in workload activity, which often means that there is enough redo log space to handle more than an hour of write activity. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. For additional information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

Undo Tablespace Configuration

By default, undo logs reside in two undo tablespaces that are created when the MySQL instance is initialized. The I/O patterns for undo logs make undo tablespaces good candidates for SSD storage.

The `innodb_undo_directory` variable defines the path where `InnoDB` creates default undo tablespaces. If that variable is undefined, default undo tablespaces are created in the data directory. The `innodb_undo_directory` variable is not dynamic. Configuring it requires restarting the server.

For information about configuring additional undo tablespaces, see Section 15.6.3.4, “Undo Tablespaces”.

Global Temporary Tablespace Configuration

The global temporary tablespace stores rollback segments for changes made to user-created temporary tables.

By default, `InnoDB` creates a single auto-extending global temporary tablespace data file named `ibtmp1` in the `innodb_data_home_dir` directory. The initial file size is slightly larger than 12MB.

The `innodb_temp_data_file_path` variable specifies the path, file name, and file size for global temporary tablespace data files. File size is specified in KB, MB, or GB by appending K, M, or G to the size value. The sum of the sizes of the files must be slightly larger than 12MB.

To specify an alternate location for global temporary tablespace data files, configure the `innodb_temp_data_file_path` variable at startup.

Session Temporary Tablespace Configuration

In MySQL 8.0.15 and earlier, session temporary tablespaces store user-created temporary tables and internal temporary tables created by the optimizer when `InnoDB` is configured as the on-disk storage engine for internal temporary tables (`internal_tmp_disk_storage_engine=InnoDB`). In MySQL 8.0.16 and later, the `InnoDB` storage engine is always used for internal temporary tables on disk.

The `innodb_temp_tablespaces_dir` variable defines the location where `InnoDB` creates session temporary tablespaces. The default location is the `#innodb_temp` directory in the data directory.

To specify an alternate location for session temporary tablespaces, configure the `innodb_temp_tablespaces_dir` variable at startup. A fully qualified path or path relative to the data directory is permitted.

Page Size Configuration

The `innodb_page_size` option specifies the page size for all `InnoDB` tablespaces in a MySQL instance. This value is set when the instance is created and remains constant afterward. Valid values are 64KB,

32KB, 16KB (the default), 8KB, and 4KB. Alternatively, you can specify page size in bytes (65536, 32768, 16384, 8192, 4096).

The default page size of 16KB is appropriate for a wide range of workloads, particularly for queries involving table scans and DML operations involving bulk updates. Smaller page sizes might be more efficient for OLTP workloads involving many small writes, where contention can be an issue when a single page contains many rows. Smaller pages might also be efficient with SSD storage devices, which typically use small block sizes. Keeping the `InnoDB` page size close to the storage device block size minimizes the amount of unchanged data that is rewritten to disk.

Memory Configuration

MySQL allocates memory to various caches and buffers to improve performance of database operations. When allocating memory for `InnoDB`, always consider memory required by the operating system, memory allocated to other applications, and memory allocated for other MySQL buffers and caches. For example, if you use `MyISAM` tables, consider the amount of memory allocated for the key buffer (`key_buffer_size`). For an overview of MySQL buffers and caches, see Section 8.12.3.1, “How MySQL Uses Memory”.

Buffers specific to `InnoDB` are configured using the following parameters:

- `innodb_buffer_pool_size` defines size of the buffer pool, which is the memory area that holds cached data for `InnoDB` tables, indexes, and other auxiliary buffers. The size of the buffer pool is important for system performance, and it is typically recommended that `innodb_buffer_pool_size` is configured to 50 to 75 percent of system memory. The default buffer pool size is 128MB. For additional guidance, see Section 8.12.3.1, “How MySQL Uses Memory”. For information about how to configure `InnoDB` buffer pool size, see Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”. Buffer pool size can be configured at startup or dynamically.

  On systems with a large amount of memory, you can improve concurrency by dividing the buffer pool into multiple buffer pool instances. The number of buffer pool instances is controlled by the by `innodb_buffer_pool_instances` option. By default, `InnoDB` creates one buffer pool instance. The number of buffer pool instances can be configured at startup. For more information, see Section 15.8.3.2, “Configuring Multiple Buffer Pool Instances”.

- `innodb_log_buffer_size` defines the size in bytes of the buffer that `InnoDB` uses to write to the log files on disk. The default size is 16MB. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit. If you have transactions that update, insert, or delete many rows, you might consider increasing the size of the log buffer to save disk I/O. `innodb_log_buffer_size` can be configured at startup. For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

Warning

On 32-bit GNU/Linux x86, be careful not to set memory usage too high. `glibc` may permit the process heap to grow over thread stacks, which crashes your server. It is a risk if the memory allocated to the `mysqld` process for global and per-thread buffers and caches is close to or exceeds 2GB.

A formula similar to the following that calculates global and per-thread memory allocation for MySQL can be used to estimate MySQL memory usage. You may need to modify the formula to account for buffers and caches in your MySQL version and configuration. For an overview of MySQL buffers and caches, see Section 8.12.3.1, “How MySQL Uses Memory”.

```
innodb_buffer_pool_size
```

```
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

Each thread uses a stack (often 2MB, but only 256KB in MySQL binaries provided by Oracle Corporation.) and in the worst case also uses `sort_buffer_size + read_buffer_size` additional memory.

On Linux, if the kernel is enabled for large page support, `InnoDB` can use large pages to allocate memory for its buffer pool. See Section 8.12.3.2, “Enabling Large Page Support”.

15.8.2 Configuring InnoDB for Read-Only Operation

You can now query `InnoDB` tables where the MySQL data directory is on read-only media, by enabling the `--innodb-read-only` configuration option at server startup.

How to Enable

To prepare an instance for read-only operation, make sure all the necessary information is flushed to the data files before storing it on the read-only medium. Run the server with change buffering disabled (`innodb_change_buffering=0`) and do a slow shutdown.

To enable read-only mode for an entire MySQL instance, specify the following configuration options at server startup:

- `--innodb-read-only=1`
- If the instance is on read-only media such as a DVD or CD, or the `/var` directory is not writeable by all: `--pid-file=`*`path_on_writeable_media`* and `--event-scheduler=disabled`
- `--innodb-temp-data-file-path`. This option specifies the path, file name, and file size for `InnoDB` temporary tablespace data files. The default setting is `ibtmp1:12M:autoextend`, which creates the `ibtmp1` temporary tablespace data file in the data directory. To prepare an instance for read-only operation, set `innodb_temp_data_file_path` to a location outside of the data directory. The path must be relative to the data directory. For example:

  ```
  --innodb-temp-data-file-path=../../../tmp/ibtmp1:12M:autoextend
  ```

As of MySQL 8.0, enabling `innodb_read_only` prevents table creation and drop operations for all storage engines. These operations modify data dictionary tables in the `mysql` system database, but those tables use the `InnoDB` storage engine and cannot be modified when `innodb_read_only` is enabled. The same restriction applies to any operation that modifies data dictionary tables, such as `ANALYZE TABLE` and `ALTER TABLE` *`tbl_name`* `ENGINE=`*`engine_name`*.

In addition, other tables in the `mysql` system database use the `InnoDB` storage engine in MySQL 8.0. Making those tables read only results in restrictions on operations that modify them. For example, `CREATE USER`, `GRANT`, `REVOKE`, and `INSTALL PLUGIN` operations are not permitted in read-only mode.

Usage Scenarios

This mode of operation is appropriate in situations such as:

- Distributing a MySQL application, or a set of MySQL data, on a read-only storage medium such as a DVD or CD.
- Multiple MySQL instances querying the same data directory simultaneously, typically in a data warehousing configuration. You might use this technique to avoid bottlenecks that can occur with

a heavily loaded MySQL instance, or you might use different configuration options for the various instances to tune each one for particular kinds of queries.

- Querying data that has been put into a read-only state for security or data integrity reasons, such as archived backup data.

Note

This feature is mainly intended for flexibility in distribution and deployment, rather than raw performance based on the read-only aspect. See Section 8.5.3, “Optimizing InnoDB Read-Only Transactions” for ways to tune the performance of read-only queries, which do not require making the entire server read-only.

How It Works

When the server is run in read-only mode through the `--innodb-read-only` option, certain `InnoDB` features and components are reduced or turned off entirely:

- No change buffering is done, in particular no merges from the change buffer. To make sure the change buffer is empty when you prepare the instance for read-only operation, disable change buffering (`innodb_change_buffering=0`) and do a slow shutdown first.
- There is no crash recovery phase at startup. The instance must have performed a slow shutdown before being put into the read-only state.
- Because the redo log is not used in read-only operation, you can set `innodb_log_file_size` to the smallest size possible (1 MB) before making the instance read-only.
- All background threads other than I/O read threads are turned off. As a consequence, a read-only instance cannot encounter any deadlock.
- Information about deadlocks, monitor output, and so on is not written to temporary files. As a consequence, `SHOW ENGINE INNODB STATUS` does not produce any output.
- Changes to configuration option settings that would normally change the behavior of write operations, have no effect when the server is in read-only mode.
- The MVCC processing to enforce isolation levels is turned off. All queries read the latest version of a record, because update and deletes are not possible.
- The undo log is not used. Disable any settings for the `innodb_undo_tablespaces` and `innodb_undo_directory` configuration options.

15.8.3 InnoDB Buffer Pool Configuration

This section provides configuration and tuning information for the `InnoDB` buffer pool.

15.8.3.1 Configuring InnoDB Buffer Pool Size

You can configure `InnoDB` buffer pool size offline (at startup) or online, while the server is running. Behavior described in this section applies to both methods. For additional information about configuring buffer pool size online, see Configuring InnoDB Buffer Pool Size Online.

When increasing or decreasing `innodb_buffer_pool_size`, the operation is performed in chunks. Chunk size is defined by the `innodb_buffer_pool_chunk_size` configuration option, which has a default of `128M`. For more information, see Configuring InnoDB Buffer Pool Chunk Size.

Buffer pool size must always be equal to or a multiple of `innodb_buffer_pool_chunk_size` * `innodb_buffer_pool_instances`. If you configure `innodb_buffer_pool_size` to a value that is not equal to or a multiple of `innodb_buffer_pool_chunk_size` * `innodb_buffer_pool_instances`, buffer pool size is automatically adjusted to a value that is equal to or a multiple of `innodb_buffer_pool_chunk_size` * `innodb_buffer_pool_instances`.

In the following example, `innodb_buffer_pool_size` is set to `8G`, and `innodb_buffer_pool_instances` is set to `16`. `innodb_buffer_pool_chunk_size` is `128M`, which is the default value.

`8G` is a valid `innodb_buffer_pool_size` value because `8G` is a multiple of `innodb_buffer_pool_instances=16` * `innodb_buffer_pool_chunk_size=128M`, which is `2G`.

```
shell> mysqld --innodb-buffer-pool-size=8G --innodb-buffer-pool-instances=16
```

```
mysql> SELECT @@innodb_buffer_pool_size/1024/1024/1024;
+----------------------------------------+
| @@innodb_buffer_pool_size/1024/1024/1024 |
+----------------------------------------+
|                          8.000000000000 |
+----------------------------------------+
```

In this example, `innodb_buffer_pool_size` is set to `9G`, and `innodb_buffer_pool_instances` is set to `16`. `innodb_buffer_pool_chunk_size` is `128M`, which is the default value. In this case, `9G` is not a multiple of `innodb_buffer_pool_instances=16` * `innodb_buffer_pool_chunk_size=128M`, so `innodb_buffer_pool_size` is adjusted to `10G`, which is a multiple of `innodb_buffer_pool_chunk_size` * `innodb_buffer_pool_instances`.

```
shell> mysqld --innodb-buffer-pool-size=9G --innodb-buffer-pool-instances=16
```

```
mysql> SELECT @@innodb_buffer_pool_size/1024/1024/1024;
+----------------------------------------+
| @@innodb_buffer_pool_size/1024/1024/1024 |
+----------------------------------------+
|                         10.000000000000 |
+----------------------------------------+
```

Configuring InnoDB Buffer Pool Chunk Size

`innodb_buffer_pool_chunk_size` can be increased or decreased in 1MB (1048576 byte) units but can only be modified at startup, in a command line string or in a MySQL configuration file.

Command line:

```
shell> mysqld --innodb-buffer-pool-chunk-size=134217728
```

Configuration file:

```
[mysqld]
innodb_buffer_pool_chunk_size=134217728
```

The following conditions apply when altering `innodb_buffer_pool_chunk_size`:

- If the new `innodb_buffer_pool_chunk_size` value * `innodb_buffer_pool_instances` is larger than the current buffer pool size when the buffer pool is initialized,

`innodb_buffer_pool_chunk_size` is truncated to `innodb_buffer_pool_size` / `innodb_buffer_pool_instances`.

For example, if the buffer pool is initialized with a size of `2GB` (2147483648 bytes), `4` buffer pool instances, and a chunk size of `1GB` (1073741824 bytes), chunk size is truncated to a value equal to `innodb_buffer_pool_size` / `innodb_buffer_pool_instances`, as shown below:

```
shell> mysqld --innodb-buffer-pool-size=2147483648 --innodb-buffer-pool-instances=4
--innodb-buffer-pool-chunk-size=1073741824;
```

```
mysql> SELECT @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|                2147483648 |
+---------------------------+

mysql> SELECT @@innodb_buffer_pool_instances;
+--------------------------------+
| @@innodb_buffer_pool_instances |
+--------------------------------+
|                              4 |
+--------------------------------+

# Chunk size was set to 1GB (1073741824 bytes) on startup but was
# truncated to innodb_buffer_pool_size / innodb_buffer_pool_instances

mysql> SELECT @@innodb_buffer_pool_chunk_size;
+---------------------------------+
| @@innodb_buffer_pool_chunk_size |
+---------------------------------+
|                       536870912 |
+---------------------------------+
```

- Buffer pool size must always be equal to or a multiple of `innodb_buffer_pool_chunk_size` * `innodb_buffer_pool_instances`. If you alter `innodb_buffer_pool_chunk_size`, `innodb_buffer_pool_size` is automatically adjusted to a value that is equal to or a multiple of `innodb_buffer_pool_chunk_size` * `innodb_buffer_pool_instances`. The adjustment occurs when the buffer pool is initialized. This behavior is demonstrated in the following example:

```
# The buffer pool has a default size of 128MB (134217728 bytes)

mysql> SELECT @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|                 134217728 |
+---------------------------+

# The chunk size is also 128MB (134217728 bytes)

mysql> SELECT @@innodb_buffer_pool_chunk_size;
+---------------------------------+
| @@innodb_buffer_pool_chunk_size |
+---------------------------------+
|                       134217728 |
+---------------------------------+

# There is a single buffer pool instance

mysql> SELECT @@innodb_buffer_pool_instances;
+--------------------------------+
```

```
| @@innodb_buffer_pool_instances |
+--------------------------------+
|                              1 |
+--------------------------------+

# Chunk size is decreased by 1MB (1048576 bytes) at startup
# (134217728 - 1048576 = 133169152):

shell> mysqld --innodb-buffer-pool-chunk-size=133169152

mysql> SELECT @@innodb_buffer_pool_chunk_size;
+---------------------------------+
| @@innodb_buffer_pool_chunk_size |
+---------------------------------+
|                       133169152 |
+---------------------------------+

# Buffer pool size increases from 134217728 to 266338304
# Buffer pool size is automatically adjusted to a value that is equal to
# or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances

mysql> SELECT @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|                 266338304 |
+---------------------------+
```

This example demonstrates the same behavior but with multiple buffer pool instances:

```

# The buffer pool has a default size of 2GB (2147483648 bytes)

mysql> SELECT @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|                2147483648 |
+---------------------------+

# The chunk size is .5 GB (536870912 bytes)

mysql> SELECT @@innodb_buffer_pool_chunk_size;
+---------------------------------+
| @@innodb_buffer_pool_chunk_size |
+---------------------------------+
|                       536870912 |
+---------------------------------+

# There are 4 buffer pool instances

mysql> SELECT @@innodb_buffer_pool_instances;
+--------------------------------+
| @@innodb_buffer_pool_instances |
+--------------------------------+
|                              4 |
+--------------------------------+

# Chunk size is decreased by 1MB (1048576 bytes) at startup
# (536870912 - 1048576 = 535822336):

shell> mysqld --innodb-buffer-pool-chunk-size=535822336

mysql> SELECT @@innodb_buffer_pool_chunk_size;
+---------------------------------+
| @@innodb_buffer_pool_chunk_size |
+---------------------------------+
```

```
|                       535822336 |
+---------------------------------+

# Buffer pool size increases from 2147483648 to 4286578688
# Buffer pool size is automatically adjusted to a value that is equal to
# or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances

mysql> SELECT @@innodb_buffer_pool_size;
+---------------------------+
| @@innodb_buffer_pool_size |
+---------------------------+
|                4286578688 |
+---------------------------+
```

Care should be taken when changing `innodb_buffer_pool_chunk_size`, as changing this value can increase the size of the buffer pool, as shown in the examples above. Before you change `innodb_buffer_pool_chunk_size`, calculate the effect on `innodb_buffer_pool_size` to ensure that the resulting buffer pool size is acceptable.

Note

To avoid potential performance issues, the number of chunks (`innodb_buffer_pool_size` / `innodb_buffer_pool_chunk_size`) should not exceed 1000.

Configuring InnoDB Buffer Pool Size Online

The `innodb_buffer_pool_size` configuration option can be set dynamically using a `SET` statement, allowing you to resize the buffer pool without restarting the server. For example:

```
mysql> SET GLOBAL innodb_buffer_pool_size=402653184;
```

Active transactions and operations performed through `InnoDB` APIs should be completed before resizing the buffer pool. When initiating a resizing operation, the operation does not start until all active transactions are completed. Once the resizing operation is in progress, new transactions and operations that require access to the buffer pool must wait until the resizing operation finishes. The exception to the rule is that concurrent access to the buffer pool is permitted while the buffer pool is defragmented and pages are withdrawn when buffer pool size is decreased. A drawback of allowing concurrent access is that it could result in a temporary shortage of available pages while pages are being withdrawn.

Note

Nested transactions could fail if initiated after the buffer pool resizing operation begins.

Monitoring Online Buffer Pool Resizing Progress

The `Innodb_buffer_pool_resize_status` reports buffer pool resizing progress. For example:

```
mysql> SHOW STATUS WHERE Variable_name='InnoDB_buffer_pool_resize_status';
+---------------------------------+---------------------------------+
| Variable_name                   | Value                           |
+---------------------------------+---------------------------------+
| Innodb_buffer_pool_resize_status | Resizing also other hash tables. |
+---------------------------------+---------------------------------+
```

Buffer pool resizing progress is also logged in the server error log. This example shows notes that are logged when increasing the size of the buffer pool:

```
[Note] InnoDB: Resizing buffer pool from 134217728 to 4294967296. (unit=134217728)
[Note] InnoDB: disabled adaptive hash index.
[Note] InnoDB: buffer pool 0 : 31 chunks (253952 blocks) was added.
[Note] InnoDB: buffer pool 0 : hash tables were resized.
[Note] InnoDB: Resized hash tables at lock_sys, adaptive hash index, dictionary.
[Note] InnoDB: completed to resize buffer pool from 134217728 to 4294967296.
[Note] InnoDB: re-enabled adaptive hash index.
```

This example shows notes that are logged when decreasing the size of the buffer pool:

```
[Note] InnoDB: Resizing buffer pool from 4294967296 to 134217728. (unit=134217728)
[Note] InnoDB: disabled adaptive hash index.
[Note] InnoDB: buffer pool 0 : start to withdraw the last 253952 blocks.
[Note] InnoDB: buffer pool 0 : withdrew 253952 blocks from free list. tried to relocate 0 pages.
(253952/253952)
[Note] InnoDB: buffer pool 0 : withdrawn target 253952 blocks.
[Note] InnoDB: buffer pool 0 : 31 chunks (253952 blocks) was freed.
[Note] InnoDB: buffer pool 0 : hash tables were resized.
[Note] InnoDB: Resized hash tables at lock_sys, adaptive hash index, dictionary.
[Note] InnoDB: completed to resize buffer pool from 4294967296 to 134217728.
[Note] InnoDB: re-enabled adaptive hash index.
```

Online Buffer Pool Resizing Internals

The resizing operation is performed by a background thread. When increasing the size of the buffer pool, the resizing operation:

- Adds pages in `chunks` (chunk size is defined by `innodb_buffer_pool_chunk_size`)
- Coverts hash tables, lists, and pointers to use new addresses in memory
- Adds new pages to the free list

While these operations are in progress, other threads are blocked from accessing the buffer pool.

When decreasing the size of the buffer pool, the resizing operation:

- Defragments the buffer pool and withdraws (frees) pages
- Removes pages in `chunks` (chunk size is defined by `innodb_buffer_pool_chunk_size`)
- Converts hash tables, lists, and pointers to use new addresses in memory

Of these operations, only defragmenting the buffer pool and withdrawing pages allow other threads to access to the buffer pool concurrently.

15.8.3.2 Configuring Multiple Buffer Pool Instances

For systems with buffer pools in the multi-gigabyte range, dividing the buffer pool into separate instances can improve concurrency, by reducing contention as different threads read and write to cached pages. This feature is typically intended for systems with a buffer pool size in the multi-gigabyte range. Multiple buffer pool instances are configured using the `innodb_buffer_pool_instances` configuration option, and you might also adjust the `innodb_buffer_pool_size` value.

When the `InnoDB` buffer pool is large, many data requests can be satisfied by retrieving from memory. You might encounter bottlenecks from multiple threads trying to access the buffer pool at once. You can enable multiple buffer pools to minimize this contention. Each page that is stored in or read from the buffer pool is assigned to one of the buffer pools randomly, using a hashing function. Each buffer pool manages its own free lists, flush lists, LRUs, and all other data structures connected to a buffer pool. Prior to MySQL

8.0, each buffer pool was protected by its own buffer pool mutex. In MySQL 8.0 and later, the buffer pool mutex was replaced by several list and hash protecting mutexes, to reduce contention.

To enable multiple buffer pool instances, set the `innodb_buffer_pool_instances` configuration option to a value greater than 1 (the default) up to 64 (the maximum). This option takes effect only when you set `innodb_buffer_pool_size` to a size of 1GB or more. The total size you specify is divided among all the buffer pools. For best efficiency, specify a combination of `innodb_buffer_pool_instances` and `innodb_buffer_pool_size` so that each buffer pool instance is at least 1GB.

For information about modifying `InnoDB` buffer pool size, see Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”.

15.8.3.3 Making the Buffer Pool Scan Resistant

Rather than using a strict LRU algorithm, `InnoDB` uses a technique to minimize the amount of data that is brought into the buffer pool and never accessed again. The goal is to make sure that frequently accessed (“hot”) pages remain in the buffer pool, even as read-ahead and full table scans bring in new blocks that might or might not be accessed afterward.

Newly read blocks are inserted into the middle of the LRU list. All newly read pages are inserted at a location that by default is `3/8` from the tail of the LRU list. The pages are moved to the front of the list (the most-recently used end) when they are accessed in the buffer pool for the first time. Thus, pages that are never accessed never make it to the front portion of the LRU list, and “age out” sooner than with a strict LRU approach. This arrangement divides the LRU list into two segments, where the pages downstream of the insertion point are considered “old” and are desirable victims for LRU eviction.

For an explanation of the inner workings of the `InnoDB` buffer pool and specifics about the LRU algorithm, see Section 15.5.1, “Buffer Pool”.

You can control the insertion point in the LRU list and choose whether `InnoDB` applies the same optimization to blocks brought into the buffer pool by table or index scans. The configuration parameter `innodb_old_blocks_pct` controls the percentage of “old” blocks in the LRU list. The default value of `innodb_old_blocks_pct` is `37`, corresponding to the original fixed ratio of 3/8. The value range is `5` (new pages in the buffer pool age out very quickly) to `95` (only 5% of the buffer pool is reserved for hot pages, making the algorithm close to the familiar LRU strategy).

The optimization that keeps the buffer pool from being churned by read-ahead can avoid similar problems due to table or index scans. In these scans, a data page is typically accessed a few times in quick succession and is never touched again. The configuration parameter `innodb_old_blocks_time` specifies the time window (in milliseconds) after the first access to a page during which it can be accessed without being moved to the front (most-recently used end) of the LRU list. The default value of `innodb_old_blocks_time` is `1000`. Increasing this value makes more and more blocks likely to age out faster from the buffer pool.

Both `innodb_old_blocks_pct` and `innodb_old_blocks_time` can be specified in the MySQL option file (`my.cnf` or `my.ini`) or changed at runtime with the `SET GLOBAL` statement. Changing the value at runtime requires privileges sufficient to set global system variables. See Section 5.1.9.1, “System Variable Privileges”.

To help you gauge the effect of setting these parameters, the `SHOW ENGINE INNODB STATUS` command reports buffer pool statistics. For details, see Monitoring the Buffer Pool Using the InnoDB Standard Monitor.

Because the effects of these parameters can vary widely based on your hardware configuration, your data, and the details of your workload, always benchmark to verify the effectiveness before changing these settings in any performance-critical or production environment.

In mixed workloads where most of the activity is OLTP type with periodic batch reporting queries which result in large scans, setting the value of `innodb_old_blocks_time` during the batch runs can help keep the working set of the normal workload in the buffer pool.

When scanning large tables that cannot fit entirely in the buffer pool, setting `innodb_old_blocks_pct` to a small value keeps the data that is only read once from consuming a significant portion of the buffer pool. For example, setting `innodb_old_blocks_pct=5` restricts this data that is only read once to 5% of the buffer pool.

When scanning small tables that do fit into memory, there is less overhead for moving pages around within the buffer pool, so you can leave `innodb_old_blocks_pct` at its default value, or even higher, such as `innodb_old_blocks_pct=50`.

The effect of the `innodb_old_blocks_time` parameter is harder to predict than the `innodb_old_blocks_pct` parameter, is relatively small, and varies more with the workload. To arrive at an optimal value, conduct your own benchmarks if the performance improvement from adjusting `innodb_old_blocks_pct` is not sufficient.

15.8.3.4 Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)

A read-ahead request is an I/O request to prefetch multiple pages in the buffer pool asynchronously, in anticipation that these pages will be needed soon. The requests bring in all the pages in one extent. `InnoDB` uses two read-ahead algorithms to improve I/O performance:

Linear read-ahead is a technique that predicts what pages might be needed soon based on pages in the buffer pool being accessed sequentially. You control when `InnoDB` performs a read-ahead operation by adjusting the number of sequential page accesses required to trigger an asynchronous read request, using the configuration parameter `innodb_read_ahead_threshold`. Before this parameter was added, `InnoDB` would only calculate whether to issue an asynchronous prefetch request for the entire next extent when it read the last page of the current extent.

The configuration parameter `innodb_read_ahead_threshold` controls how sensitive `InnoDB` is in detecting patterns of sequential page access. If the number of pages read sequentially from an extent is greater than or equal to `innodb_read_ahead_threshold`, `InnoDB` initiates an asynchronous read-ahead operation of the entire following extent. `innodb_read_ahead_threshold` can be set to any value from 0-64. The default value is 56. The higher the value, the more strict the access pattern check. For example, if you set the value to 48, `InnoDB` triggers a linear read-ahead request only when 48 pages in the current extent have been accessed sequentially. If the value is 8, `InnoDB` triggers an asynchronous read-ahead even if as few as 8 pages in the extent are accessed sequentially. You can set the value of this parameter in the MySQL configuration file, or change it dynamically with the `SET GLOBAL` statement, which requires privileges sufficient to set global system variables. See Section 5.1.9.1, “System Variable Privileges”.

Random read-ahead is a technique that predicts when pages might be needed soon based on pages already in the buffer pool, regardless of the order in which those pages were read. If 13 consecutive pages from the same extent are found in the buffer pool, `InnoDB` asynchronously issues a request to prefetch the remaining pages of the extent. To enable this feature, set the configuration variable `innodb_random_read_ahead` to `ON`.

The `SHOW ENGINE INNODB STATUS` command displays statistics to help you evaluate the effectiveness of the read-ahead algorithm. Statistics include counter information for the following global status variables:

- `Innodb_buffer_pool_read_ahead`
- `Innodb_buffer_pool_read_ahead_evicted`
- `Innodb_buffer_pool_read_ahead_rnd`

This information can be useful when fine-tuning the `innodb_random_read_ahead` setting.

For more information about I/O performance, see Section 8.5.8, “Optimizing InnoDB Disk I/O” and Section 8.12.1, “Optimizing Disk I/O”.

15.8.3.5 Configuring Buffer Pool Flushing

`InnoDB` performs certain tasks in the background, including flushing of dirty pages from the buffer pool. Dirty pages are those that have been modified but are not yet written to the data files on disk.

In MySQL 8.0, buffer pool flushing is performed by page cleaner threads. The number of page cleaner threads is controlled by the `innodb_page_cleaners` variable, which has a default value of 4.

Buffer pool flushing is initiated when the percentage of dirty pages reaches the low water mark value defined by the `innodb_max_dirty_pages_pct_lwm` variable. The default low water mark is 10% of buffer pool pages. A `innodb_max_dirty_pages_pct_lwm` value of 0 disables this early flushing behaviour.

The purpose of the `innodb_max_dirty_pages_pct_lwm` threshold is to control the percentage dirty pages in the buffer pool, and to prevent the amount of dirty pages from reaching the threshold defined by the `innodb_max_dirty_pages_pct` variable, which has a default value of 90. `InnoDB` aggressively flushes buffer pool pages if the percentage of dirty pages in the buffer pool reaches the `innodb_max_dirty_pages_pct` threshold.

Additional variables permit fine-tuning of buffer pool flushing behavior:

- The `innodb_flush_neighbors` variable defines whether flushing a page from the buffer pool also flushes other dirty pages in the same extent.
  - A setting of 0 disables `innodb_flush_neighbors`. Dirty pages in the same extent are not flushed.
  - The default setting of 1 flushes contiguous dirty pages in the same extent.
  - A setting of 2 flushes dirty pages in the same extent.

  When table data is stored on a traditional HDD storage device, flushing neighbor pages in one operation reduces I/O overhead (primarily for disk seek operations) compared to flushing individual pages at different times. For table data stored on SSD, seek time is not a significant factor and you can disable this setting to spread out write operations.
- The `innodb_lru_scan_depth` variable specifies, per buffer pool instance, how far down the buffer pool LRU list the page cleaner thread scans looking for dirty pages to flush. This is a background operation performed by a page cleaner thread once per second.

  A setting smaller than the default is generally suitable for most workloads. A value that is significantly higher than necessary may impact performance. Only consider increasing the value if you have spare I/O capacity under a typical workload. Conversely, if a write-intensive workload saturates your I/O capacity, decrease the value, especially in the case of a large buffer pool.

  When tuning `innodb_lru_scan_depth`, start with a low value and configure the setting upward with the goal of rarely seeing zero free pages. Also, consider adjusting `innodb_lru_scan_depth` when changing the number of buffer pool instances, since `innodb_lru_scan_depth` * `innodb_buffer_pool_instances` defines the amount of work performed by the page cleaner thread each second.

The `innodb_flush_neighbors` and `innodb_lru_scan_depth` variables are primarily intended for write-intensive workloads. With heavy DML activity, flushing can fall behind if it is not aggressive enough,

or disk writes can saturate I/O capacity if flushing is too aggressive. The ideal settings depend on your workload, data access patterns, and storage configuration (for example, whether data is stored on HDD or SSD devices).

Adaptive Flushing

`InnoDB` uses an adaptive flushing algorithm to dynamically adjust the rate of flushing based on the speed of redo log generation and the current rate of flushing. The intent is to smooth overall performance by ensuring that flushing activity keeps pace with the current workload. Automatically adjusting the flushing rate helps avoid sudden dips in throughput that can occur when bursts of I/O activity due to buffer pool flushing affects the I/O capacity available for ordinary read and write activity.

Sharp checkpoints, which are typically associated with write-intensive workloads that generate a lot of redo entries, can cause a sudden change in throughput, for example. A sharp checkpoint occurs when `InnoDB` wants to reuse a portion of a log file. Before doing so, all dirty pages with redo entries in that portion of the log file must be flushed. If log files become full, a sharp checkpoint occurs, causing a temporary reduction in throughput. This scenario can occur even if `innodb_max_dirty_pages_pct` threshold is not reached.

The adaptive flushing algorithm helps avoid such scenarios by tracking the number of dirty pages in the buffer pool and the rate at which redo log records are being generated. Based on this information, it decides how many dirty pages to flush from the buffer pool each second, which permits it to manage sudden changes in workload.

The `innodb_adaptive_flushing_lwm` variable defines a low water mark for redo log capacity. When that threshold is crossed, adaptive flushing is enabled, even if the `innodb_adaptive_flushing` variable is disabled.

Internal benchmarking has shown that the algorithm not only maintains throughput over time, but can also improve overall throughput significantly. However, adaptive flushing can affect the I/O pattern of a workload significantly and may not be appropriate in all cases. It gives the most benefit when the redo log is in danger of filling up. If adaptive flushing is not appropriate to the characteristics of your workload, you can disable it. Adaptive flushing controlled by the `innodb_adaptive_flushing` variable, which is enabled by default.

`innodb_flushing_avg_loops` defines the number of iterations that `InnoDB` keeps the previously calculated snapshot of the flushing state, controlling how quickly adaptive flushing responds to foreground workload changes. A high `innodb_flushing_avg_loops` value means that `InnoDB` keeps the previously calculated snapshot longer, so adaptive flushing responds more slowly. When setting a high value it is important to ensure that redo log utilization does not reach 75% (the hardcoded limit at which asynchronous flushing starts), and that the `innodb_max_dirty_pages_pct` threshold keeps the number of dirty pages to a level that is appropriate for the workload.

Systems with consistent workloads, a large log file size (`innodb_log_file_size`), and small spikes that do not reach 75% log space utilization should use a high `innodb_flushing_avg_loops` value to keep flushing as smooth as possible. For systems with extreme load spikes or log files that do not provide a lot of space, a smaller value allows flushing to closely track workload changes, and helps to avoid reaching 75% log space utilization.

Be aware that if flushing falls behind, the rate of buffer pool flushing can exceed the I/O capacity available to `InnoDB`, as defined by `innodb_io_capacity` setting. The `innodb_io_capacity_max` value defines an upper limit on I/O capacity in such situations, so that a spike in I/O activity does not consume the entire I/O capacity of the server.

The `innodb_io_capacity` setting is applicable to all buffer pool instances. When dirty pages are flushed, I/O capacity is divided equally among buffer pool instances.

Limiting Buffer Flushing During Idle Periods

As of MySQL 8.0.18, you can use the `innodb_idle_flush_pct` variable to limit the rate of buffer pool flushing during idle periods, which are periods of time that database pages are not modified. The `innodb_idle_flush_pct` value is a percentage of the `innodb_io_capacity` setting, which defines the number of I/O operations per second available to `InnoDB`. The default `innodb_idle_flush_pct` value is 100, which is 100 percent of the `innodb_io_capacity` setting. To limit flushing during idle periods, define an `innodb_idle_flush_pct` value less than 100.

Limiting page flushing during idle periods can help extend the life of solid state storage devices. Side effects of limiting page flushing during idle periods may include a longer shutdown time following a lengthy idle period, and a longer recovery period should a server failure occur.

15.8.3.6 Saving and Restoring the Buffer Pool State

To reduce the warmup period after restarting the server, `InnoDB` saves a percentage of the most recently used pages for each buffer pool at server shutdown and restores these pages at server startup. The percentage of recently used pages that is stored is defined by the `innodb_buffer_pool_dump_pct` configuration option.

After restarting a busy server, there is typically a warmup period with steadily increasing throughput, as disk pages that were in the buffer pool are brought back into memory (as the same data is queried, updated, and so on). The ability to restore the buffer pool at startup shortens the warmup period by reloading disk pages that were in the buffer pool before the restart rather than waiting for DML operations to access corresponding rows. Also, I/O requests can be performed in large batches, making the overall I/O faster. Page loading happens in the background, and does not delay database startup.

In addition to saving the buffer pool state at shutdown and restoring it at startup, you can save and restore the buffer pool state at any time, while the server is running. For example, you can save the state of the buffer pool after reaching a stable throughput under a steady workload. You could also restore the previous buffer pool state after running reports or maintenance jobs that bring data pages into the buffer pool that are only requited for those operations, or after running some other non-typical workload.

Even though a buffer pool can be many gigabytes in size, the buffer pool data that `InnoDB` saves to disk is tiny by comparison. Only tablespace IDs and page IDs necessary to locate the appropriate pages are saved to disk. This information is derived from the `INNODB_BUFFER_PAGE_LRU` `INFORMATION_SCHEMA` table. By default, tablespace ID and page ID data is saved in a file named `ib_buffer_pool`, which is saved to the `InnoDB` data directory. The file name and location can be modified using the `innodb_buffer_pool_filename` configuration parameter.

Because data is cached in and aged out of the buffer pool as it is with regular database operations, there is no problem if the disk pages are recently updated, or if a DML operation involves data that has not yet been loaded. The loading mechanism skips requested pages that no longer exist.

The underlying mechanism involves a background thread that is dispatched to perform the dump and load operations.

Disk pages from compressed tables are loaded into the buffer pool in their compressed form. Pages are uncompressed as usual when page contents are accessed during DML operations. Because uncompressing pages is a CPU-intensive process, it is more efficient for concurrency to perform the operation in a connection thread rather than in the single thread that performs the buffer pool restore operation.

Operations related to saving and restoring the buffer pool state are described in the following topics:

- Configuring the Dump Percentage for Buffer Pool Pages

- Saving the Buffer Pool State at Shutdown and Restoring it at Startup
- Saving and Restoring the Buffer Pool State Online
- Displaying Buffer Pool Dump Progress
- Displaying Buffer Pool Load Progress
- Aborting a Buffer Pool Load Operation
- Monitoring Buffer Pool Load Progress Using Performance Schema

Configuring the Dump Percentage for Buffer Pool Pages

Before dumping pages from the buffer pool, you can configure the percentage of most-recently-used buffer pool pages that you want to dump by setting the `innodb_buffer_pool_dump_pct` option. If you plan to dump buffer pool pages while the server is running, you can configure the option dynamically:

```
SET GLOBAL innodb_buffer_pool_dump_pct=40;
```

If you plan to dump buffer pool pages at server shutdown, set `innodb_buffer_pool_dump_pct` in your configuration file.

```
[mysqld]
innodb_buffer_pool_dump_pct=40
```

The `innodb_buffer_pool_dump_pct` default value is 25 (dump 25% of most-recently-used pages).

Saving the Buffer Pool State at Shutdown and Restoring it at Startup

To save the state of the buffer pool at server shutdown, issue the following statement prior to shutting down the server:

```
SET GLOBAL innodb_buffer_pool_dump_at_shutdown=ON;
```

`innodb_buffer_pool_dump_at_shutdown` is enabled by default.

To restore the buffer pool state at server startup, specify the `--innodb-buffer-pool-load-at-startup` option when starting the server:

```
mysqld --innodb-buffer-pool-load-at-startup=ON;
```

`innodb_buffer_pool_load_at_startup` is enabled by default.

Saving and Restoring the Buffer Pool State Online

To save the state of the buffer pool while MySQL server is running, issue the following statement:

```
SET GLOBAL innodb_buffer_pool_dump_now=ON;
```

To restore the buffer pool state while MySQL is running, issue the following statement:

```
```

```
SET GLOBAL innodb_buffer_pool_load_now=ON;
```

Displaying Buffer Pool Dump Progress

To display progress when saving the buffer pool state to disk, issue the following statement:

```
SHOW STATUS LIKE 'Innodb_buffer_pool_dump_status';
```

If the operation has not yet started, “not started” is returned. If the operation is complete, the completion time is printed (e.g. Finished at 110505 12:18:02). If the operation is in progress, status information is provided (e.g. Dumping buffer pool 5/7, page 237/2873).

Displaying Buffer Pool Load Progress

To display progress when loading the buffer pool, issue the following statement:

```
SHOW STATUS LIKE 'Innodb_buffer_pool_load_status';
```

If the operation has not yet started, “not started” is returned. If the operation is complete, the completion time is printed (e.g. Finished at 110505 12:23:24). If the operation is in progress, status information is provided (e.g. Loaded 123/22301 pages).

Aborting a Buffer Pool Load Operation

To abort a buffer pool load operation, issue the following statement:

```
SET GLOBAL innodb_buffer_pool_load_abort=ON;
```

Monitoring Buffer Pool Load Progress Using Performance Schema

You can monitor buffer pool load progress using Performance Schema.

The following example demonstrates how to enable the `stage/innodb/buffer pool load` stage event instrument and related consumer tables to monitor buffer pool load progress.

For information about buffer pool dump and load procedures used in this example, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”. For information about Performance Schema stage event instruments and related consumers, see Section 26.12.5, “Performance Schema Stage Event Tables”.

1. Enable the `stage/innodb/buffer pool load` instrument:

   ```
   mysql> UPDATE performance_schema.setup_instruments SET ENABLED = 'YES'
          WHERE NAME LIKE 'stage/innodb/buffer%';
   ```

2. Enable the stage event consumer tables, which include `events_stages_current`, `events_stages_history`, and `events_stages_history_long`.

   ```
   mysql> UPDATE performance_schema.setup_consumers SET ENABLED = 'YES'
          WHERE NAME LIKE '%stages%';
   ```

3. Dump the current buffer pool state by enabling `innodb_buffer_pool_dump_now`.

   ```
   mysql> SET GLOBAL innodb_buffer_pool_dump_now=ON;
   ```

4. Check the buffer pool dump status to ensure that the operation has completed.

```
mysql> SHOW STATUS LIKE 'Innodb_buffer_pool_dump_status'\G
*************************** 1. row ***************************
Variable_name: Innodb_buffer_pool_dump_status
        Value: Buffer pool(s) dump completed at 150202 16:38:58
```

5. Load the buffer pool by enabling `innodb_buffer_pool_load_now`:

```
mysql> SET GLOBAL innodb_buffer_pool_load_now=ON;
```

6. Check the current status of the buffer pool load operation by querying the Performance Schema `events_stages_current` table. The `WORK_COMPLETED` column shows the number of buffer pool pages loaded. The `WORK_ESTIMATED` column provides an estimate of the remaining work, in pages.

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED
       FROM performance_schema.events_stages_current;
+-------------------------------+----------------+----------------+
| EVENT_NAME                    | WORK_COMPLETED | WORK_ESTIMATED |
+-------------------------------+----------------+----------------+
| stage/innodb/buffer pool load |           5353 |           7167 |
+-------------------------------+----------------+----------------+
```

The `events_stages_current` table returns an empty set if the buffer pool load operation has completed. In this case, you can check the `events_stages_history` table to view data for the completed event. For example:

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED
       FROM performance_schema.events_stages_history;
+-------------------------------+----------------+----------------+
| EVENT_NAME                    | WORK_COMPLETED | WORK_ESTIMATED |
+-------------------------------+----------------+----------------+
| stage/innodb/buffer pool load |           7167 |           7167 |
+-------------------------------+----------------+----------------+
```

Note

You can also monitor buffer pool load progress using Performance Schema when loading the buffer pool at startup using `innodb_buffer_pool_load_at_startup`. In this case, the `stage/innodb/buffer pool load` instrument and related consumers must be enabled at startup. For more information, see Section 26.3, “Performance Schema Startup Configuration”.

15.8.3.7 Excluding Buffer Pool Pages from Core Files

A core file records the status and memory image of a running process. Because the buffer pool resides in main memory, and the memory image of a running process is dumped to the core file, systems with large buffer pools can produce large core files when the `mysqld` process dies.

Large core files can be problematic for a number of reasons including the time it takes to write them, the amount of disk space they consume, and the challenges associated with transferring large files.

To reduce core file size, you can disable the `innodb_buffer_pool_in_core_file` variable to omit buffer pool pages from core dumps. The `innodb_buffer_pool_in_core_file` variable was introduced in MySQL 8.0.14 and is enabled by default.

Excluding buffer pool pages may also be desirable from a security perspective if you have concerns about dumping database pages to core files that may be shared inside or outside of your organization for debugging purposes.

Note

Access to the data present in buffer pool pages at the time the `mysqld` process died may be beneficial in some debugging scenarios. If in doubt whether to include or exclude buffer pool pages, consult MySQL Support.

Disabling `innodb_buffer_pool_in_core_file` takes effect only if the `core_file` variable is enabled and the operating system supports the `MADV_DONTDUMP` non-POSIX extension to the madvise() system call, which is supported in Linux 3.4 and later. The `MADV_DONTDUMP` extension causes pages in a specified range to be excluded from core dumps.

Assuming the operating system supports the `MADV_DONTDUMP` extension, start the server with the `--core-file` and `--innodb-buffer-pool-in-core-file=OFF` options to generate core files without buffer pool pages.

```
shell> mysqld --core-file --innodb-buffer-pool-in-core-file=OFF
```

The `core_file` variable is read only and disabled by default. It is enabled by specifying the `--core-file` option at startup. The `innodb_buffer_pool_in_core_file` variable is dynamic. It can be specified at startup or configured at runtime using a `SET` statement.

```
mysql> SET GLOBAL innodb_buffer_pool_in_core_file=OFF;
```

If the `innodb_buffer_pool_in_core_file` variable is disabled but `MADV_DONTDUMP` is not supported by the operating system, or an `madvise()` failure occurs, a warning is written to the MySQL server error log and the `core_file` variable is disabled to prevent writing core files that unintentionally include buffer pool pages. If the read-only `core_file` variable becomes disabled, the server must be restarted to enable it again.

The following table shows configuration and `MADV_DONTDUMP` support scenarios that determine whether core files are generated and whether they include buffer pool pages.

Table 15.5 Core File Configuration Scenarios

`core_file` variable	`innodb_buffer_pool_in_core_file` variable	madvise() MADV_DONTDUMP Support	Outcome
OFF (default)	Not relevant to outcome	Not relevant to outcome	Core file is not generated
ON	ON (default)	Not relevant to outcome	Core file is generated with buffer pool pages
ON	OFF	Yes	Core file is generated without buffer pool pages
ON	OFF	No	Core file is not generated, `core_file` is disabled, and a warning is written to the server error log

The reduction in core file size achieved by disabling the `innodb_buffer_pool_in_core_file` variable depends on the size of the buffer pool, but it is also affected by the `InnoDB` page size. A smaller page size means more pages are required for the same amount of data, and more pages means more page metadata. The following table provides size reduction examples that you might see for a 1GB buffer pool with different pages sizes.

Table 15.6 Core File Size with Buffer Pool Pages Included and Excluded

innodb_page_size Setting	Buffer Pool Pages Included (innodb_buffer_pool_in_core_file=ON)	Buffer Pool Pages Excluded (innodb_buffer_pool_in_core_file=OFF)
4KB	2.1GB	0.9GB
64KB	1.7GB	0.7GB

15.8.4 Configuring Thread Concurrency for InnoDB

InnoDB uses operating system threads to process requests from user transactions. (Transactions may issue many requests to InnoDB before they commit or roll back.) On modern operating systems and servers with multi-core processors, where context switching is efficient, most workloads run well without any limit on the number of concurrent threads.

In situations where it is helpful to minimize context switching between threads, InnoDB can use a number of techniques to limit the number of concurrently executing operating system threads (and thus the number of requests that are processed at any one time). When InnoDB receives a new request from a user session, if the number of threads concurrently executing is at a pre-defined limit, the new request sleeps for a short time before it tries again. A request that cannot be rescheduled after the sleep is put in a first-in/first-out queue and eventually is processed. Threads waiting for locks are not counted in the number of concurrently executing threads.

You can limit the number of concurrent threads by setting the configuration parameter innodb_thread_concurrency. Once the number of executing threads reaches this limit, additional threads sleep for a number of microseconds, set by the configuration parameter innodb_thread_sleep_delay, before being placed into the queue.

You can set the configuration option innodb_adaptive_max_sleep_delay to the highest value you would allow for innodb_thread_sleep_delay, and InnoDB automatically adjusts innodb_thread_sleep_delay up or down depending on the current thread-scheduling activity. This dynamic adjustment helps the thread scheduling mechanism to work smoothly during times when the system is lightly loaded and when it is operating near full capacity.

The default value for innodb_thread_concurrency and the implied default limit on the number of concurrent threads has been changed in various releases of MySQL and InnoDB. The default value of innodb_thread_concurrency is 0, so that by default there is no limit on the number of concurrently executing threads.

InnoDB causes threads to sleep only when the number of concurrent threads is limited. When there is no limit on the number of threads, all contend equally to be scheduled. That is, if innodb_thread_concurrency is 0, the value of innodb_thread_sleep_delay is ignored.

When there is a limit on the number of threads (when innodb_thread_concurrency is > 0), InnoDB reduces context switching overhead by permitting multiple requests made during the execution of a *single SQL statement* to enter InnoDB without observing the limit set by innodb_thread_concurrency. Since an SQL statement (such as a join) may comprise multiple row operations within InnoDB, InnoDB assigns a specified number of “tickets” that allow a thread to be scheduled repeatedly with minimal overhead.

When a new SQL statement starts, a thread has no tickets, and it must observe innodb_thread_concurrency. Once the thread is entitled to enter InnoDB, it is assigned a number of tickets that it can use for subsequently entering InnoDB to perform row operations. If the tickets run out, the thread is evicted, and innodb_thread_concurrency is observed again which may place the thread back into the first-in/first-out queue of waiting threads. When the thread is once again entitled to enter InnoDB, tickets are assigned again. The number of tickets assigned is specified by the global option

`innodb_concurrency_tickets`, which is 5000 by default. A thread that is waiting for a lock is given one ticket once the lock becomes available.

The correct values of these variables depend on your environment and workload. Try a range of different values to determine what value works for your applications. Before limiting the number of concurrently executing threads, review configuration options that may improve the performance of `InnoDB` on multi-core and multi-processor computers, such as `innodb_adaptive_hash_index`.

For general performance information about MySQL thread handling, see Section 8.12.4.1, "How MySQL Handles Client Connections".

15.8.5 Configuring the Number of Background InnoDB I/O Threads

`InnoDB` uses background threads to service various types of I/O requests. You can configure the number of background threads that service read and write I/O on data pages using the `innodb_read_io_threads` and `innodb_write_io_threads` configuration parameters. These parameters signify the number of background threads used for read and write requests, respectively. They are effective on all supported platforms. You can set values for these parameters in the MySQL option file (`my.cnf` or `my.ini`); you cannot change values dynamically. The default value for these parameters is `4` and permissible values range from `1-64`.

The purpose of these configuration options to make `InnoDB` more scalable on high end systems. Each background thread can handle up to 256 pending I/O requests. A major source of background I/O is read-ahead requests. `InnoDB` tries to balance the load of incoming requests in such way that most background threads share work equally. `InnoDB` also attempts to allocate read requests from the same extent to the same thread, to increase the chances of coalescing the requests. If you have a high end I/O subsystem and you see more than 64 × `innodb_read_io_threads` pending read requests in `SHOW ENGINE INNODB STATUS` output, you might improve performance by increasing the value of `innodb_read_io_threads`.

On Linux systems, `InnoDB` uses the asynchronous I/O subsystem by default to perform read-ahead and write requests for data file pages, which changes the way that `InnoDB` background threads service these types of I/O requests. For more information, see Section 15.8.6, "Using Asynchronous I/O on Linux".

For more information about `InnoDB` I/O performance, see Section 8.5.8, "Optimizing InnoDB Disk I/O".

15.8.6 Using Asynchronous I/O on Linux

`InnoDB` uses the asynchronous I/O subsystem (native AIO) on Linux to perform read-ahead and write requests for data file pages. This behavior is controlled by the `innodb_use_native_aio` configuration option, which applies to Linux systems only and is enabled by default. On other Unix-like systems, `InnoDB` uses synchronous I/O only. Historically, `InnoDB` only used asynchronous I/O on Windows systems. Using the asynchronous I/O subsystem on Linux requires the `libaio` library.

With synchronous I/O, query threads queue I/O requests, and `InnoDB` background threads retrieve the queued requests one at a time, issuing a synchronous I/O call for each. When an I/O request is completed and the I/O call returns, the `InnoDB` background thread that is handling the request calls an I/O completion routine and returns to process the next request. The number of requests that can be processed in parallel is *n*, where *n* is the number of `InnoDB` background threads. The number of `InnoDB` background threads is controlled by `innodb_read_io_threads` and `innodb_write_io_threads`. See Section 15.8.5, "Configuring the Number of Background InnoDB I/O Threads".

With native AIO, query threads dispatch I/O requests directly to the operating system, thereby removing the limit imposed by the number of background threads. `InnoDB` background threads wait for I/O events to signal completed requests. When a request is completed, a background thread calls an I/O completion routine and resumes waiting for I/O events.

The advantage of native AIO is scalability for heavily I/O-bound systems that typically show many pending reads/writes in `SHOW ENGINE INNODB STATUS\G` output. The increase in parallel processing when using native AIO means that the type of I/O scheduler or properties of the disk array controller have a greater influence on I/O performance.

A potential disadvantage of native AIO for heavily I/O-bound systems is lack of control over the number of I/O write requests dispatched to the operating system at once. Too many I/O write requests dispatched to the operating system for parallel processing could, in some cases, result in I/O read starvation, depending on the amount of I/O activity and system capabilities.

If a problem with the asynchronous I/O subsystem in the OS prevents `InnoDB` from starting, you can start the server with `innodb_use_native_aio=0`. This option may also be disabled automatically during startup if `InnoDB` detects a potential problem such as a combination of `tmpdir` location, `tmpfs` file system, and Linux kernel that does not support asynchronous I/O on `tmpfs`.

15.8.7 Configuring InnoDB I/O Capacity

The `InnoDB` master thread and other background threads perform various tasks in the background, most of which are I/O related, such as flushing dirty pages from the buffer pool or writing changes from the insert buffer to the appropriate secondary indexes. `InnoDB` attempts to perform these tasks in a way that does not adversely affect the normal working of the server. It tries to estimate the free I/O bandwidth available and tune its activities to take advantage of this free capacity. Historically, `InnoDB` used a hard coded value of 100 IOPs (input/output operations per second) as the total I/O capacity of the server.

The `innodb_io_capacity` variable defines the overall I/O capacity available to `InnoDB`. It should be set to approximately the number of I/O operations that the system can perform per second. The value depends on your system configuration. When `innodb_io_capacity` is set, `InnoDB` estimates the I/O bandwidth available for background tasks based on the set value. Setting the value to `100` reverts to the old behavior.

You can set `innodb_io_capacity` to a value of 100 or greater. The default value is `200`, reflecting that the performance of typical modern I/O devices is higher than in the early days of MySQL. Typically, values around the previous default of 100 are appropriate for consumer-level storage devices, such as hard drives up to 7200 RPMs. Faster hard drives, RAID configurations, and SSDs benefit from higher values.

You can set the `innodb_io_capacity` value in the MySQL option file (`my.cnf` or `my.ini`) or change it dynamically with the `SET GLOBAL` statement, which requires privileges sufficient to set global system variables. See Section 5.1.9.1, “System Variable Privileges”.

The `innodb_flush_sync` configuration option causes the `innodb_io_capacity` setting to be ignored during bursts of I/O activity that occur at checkpoints. `innodb_flush_sync` is enabled by default.

For related information, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

15.8.8 Configuring Spin Lock Polling

`InnoDB` mutexes and rw-locks are typically reserved for short intervals. On a multi-core system, it can be more efficient for a thread to continuously check if it can acquire a mutex or rw-lock for a period of time before it sleeps. If the mutex or rw-lock becomes available during this period, the thread can continue immediately, in the same time slice. However, too-frequent polling of a shared object such as a mutex or rw-lock by multiple threads can cause “cache ping pong”, which results in processors invalidating portions of each other's cache. `InnoDB` minimizes this issue by forcing a random delay between polls to desychronize polling activity. The random delay is implemented as a spin-wait loop.

The duration of a spin-wait loop is determined by the number of PAUSE instructions that occur in the loop. That number is generated by randomly selecting an integer ranging from 0 up to but not including the `innodb_spin_wait_delay` value, and multiplying that value by 50. (The multiplier value, 50, is

hardcoded before MySQL 8.0.16, and configurable thereafter.) For example, an integer is randomly selected from the following range for an `innodb_spin_wait_delay` setting of 6:

```
{0,1,2,3,4,5}
```

The selected integer is multiplied by 50, resulting in one of six possible PAUSE instruction values:

```
{0,50,100,150,200,250}
```

For that set of values, 250 is the maximum number of PAUSE instructions that can occur in a spin-wait loop. An `innodb_spin_wait_delay` setting of 5 results in a set of five possible values `{0,50,100,150,200}`, where 200 is the maximum number of PAUSE instructions, and so on. In this way, the `innodb_spin_wait_delay` setting controls the maximum delay between spin lock polls.

On a system where all processor cores share a fast cache memory, you might reduce the maximum delay or disable the busy loop altogether by setting `innodb_spin_wait_delay=0`. On a system with multiple processor chips, the effect of cache invalidation can be more significant and you might increase the maximum delay.

In the 100MHz Pentium era, an `innodb_spin_wait_delay` unit was calibrated to be equivalent to one microsecond. That time equivalence did not hold, but PAUSE instruction duration remained fairly constant in terms of processor cycles relative to other CPU instructions until the introduction of the Skylake generation of processors, which have a comparatively longer PAUSE instruction. The `innodb_spin_wait_pause_multiplier` variable was introduced in MySQL 8.0.16 to provide a way to account for differences in PAUSE instruction duration.

The `innodb_spin_wait_pause_multiplier` variable controls the size of PAUSE instruction values. For example, assuming an `innodb_spin_wait_delay` setting of 6, decreasing the `innodb_spin_wait_pause_multiplier` value from 50 (the default and previously hardcoded value) to 5 generates a set of smaller PAUSE instruction values:

```
{0,5,10,15,20,25}
```

The ability to increase or decrease PAUSE instruction values permits fine tuning `InnoDB` for different processor architectures. Smaller PAUSE instruction values would be appropriate for processor architectures with a comparatively longer PAUSE instruction, for example.

The `innodb_spin_wait_delay` and `innodb_spin_wait_pause_multiplier` variables are dynamic. They can be specified in a MySQL option file or modified at runtime using a `SET GLOBAL` statement. Modifying the variables at runtime requires privileges sufficient to set global system variables. See Section 5.1.9.1, “System Variable Privileges”.

15.8.9 Purge Configuration

`InnoDB` does not physically remove a row from the database immediately when you delete it with an SQL statement. A row and its index records are only physically removed when `InnoDB` discards the undo log record written for the deletion of the row. This removal operation, which only occurs after the row is no longer required for multi-version concurrency control (MVCC) or rollback, is called a purge.

Purge runs on a periodic schedule. It parses and processes undo log pages from the history list, which is a list of undo log pages for committed transactions maintained by `InnoDB`. Purge frees the undo log pages from the history list after processing them.

Configuring Purge Threads

Purge operations are performed in the background by one or more dedicated purge threads. The number of purge threads is controlled by the `innodb_purge_threads` variable. The default value is 4. If DML action is concentrated on a single table or a few tables, keep the setting low so that the threads do not

contend with each other for access to the tables. If DML operations are spread across many tables, increase the setting. The maximum number of purge threads is 32.

Configuring Purge Batch Size

The `innodb_purge_batch_size` variable defines the number of undo log pages that purge parses and processes in one batch from the history list. The default value is 300. In a multithreaded purge configuration, the coordinator purge thread divides `innodb_purge_batch_size` by `innodb_purge_threads` and assigns that number of pages to each purge thread.

The `innodb_purge_batch_size` variable is intended for advanced performance tuning and experimentation. Most users need not change `innodb_purge_batch_size` from its default value.

Configuring the Maximum Purge Lag

The `innodb_max_purge_lag` variable defines the desired maximum purge lag. When the purge lag exceeds the `innodb_max_purge_lag` threshold, a delay is imposed on `INSERT`, `UPDATE`, and `DELETE` operations to allow time for purge operations to catch up. The default value is 0, which means there is no maximum purge lag and no delay.

Prior to MySQL 8.0.14, the purge lag delay is calculated by this formula, which results in a minimum delay of 5000 microseconds:

```
(purge lag/innodb_max_purge_lag - 0.5) * 10000
```

As of MySQL 8.0.14, the purge lag delay is calculated by this formula, which reduces the minimum delay to 5 microseconds. A delay of 5 microseconds is more appropriate for modern systems.

```
(purge_lag/innodb_max_purge_lag - 0.9995) * 10000
```

The purge lag delay is computed at the beginning of a purge batch.

A typical `innodb_max_purge_lag` setting for a problematic workload might be 1000000 (1 million), assuming that transactions are small, only 100 bytes in size, and it is permissible to have 100MB of unpurged table rows.

The purge lag is presented as the `History list length` value in the `TRANSACTIONS` section of `SHOW ENGINE INNODB STATUS` output.

```
mysql> SHOW ENGINE INNODB STATUS;
...
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
History list length 20
```

To prevent excessive delays in extreme situations where the purge lag becomes huge, you can limit the delay by setting the `innodb_max_purge_lag_delay` variable. The `innodb_max_purge_lag_delay` variable specifies the maximum delay in microseconds for the delay imposed when the `innodb_max_purge_lag` threshold is exceeded. The specified `innodb_max_purge_lag_delay` value is an upper limit on the delay period computed by the `innodb_max_purge_lag` formula.

Purge and Undo Tablespace Truncation

The purge system is also plays a roll in truncating undo tablespaces. You can configure the `innodb_purge_rseg_truncate_frequency` variable to control the frequency with which the purge system looks for undo tablespaces to truncate. For more information, see Truncating Undo Tablespaces.

15.8.10 Configuring Optimizer Statistics for InnoDB

This section describes how to configure persistent and non-persistent optimizer statistics for `InnoDB` tables.

Persistent optimizer statistics are persisted across server restarts, allowing for greater plan stability and more consistent query performance. Persistent optimizer statistics also provide control and flexibility with these additional benefits:

- You can use the `innodb_stats_auto_recalc` configuration option to control whether statistics are updated automatically after substantial changes to a table.
- You can use the `STATS_PERSISTENT`, `STATS_AUTO_RECALC`, and `STATS_SAMPLE_PAGES` clauses with `CREATE TABLE` and `ALTER TABLE` statements to configure optimizer statistics for individual tables.
- You can query optimizer statistics data in the `mysql.innodb_table_stats` and `mysql.innodb_index_stats` tables.
- You can view the `last_update` column of the `mysql.innodb_table_stats` and `mysql.innodb_index_stats` tables to see when statistics were last updated.
- You can manually modify the `mysql.innodb_table_stats` and `mysql.innodb_index_stats` tables to force a specific query optimization plan or to test alternative plans without modifying the database.

The persistent optimizer statistics feature is enabled by default (`innodb_stats_persistent=ON`).

Non-persistent optimizer statistics are cleared on each server restart and after some other operations, and recomputed on the next table access. As a result, different estimates could be produced when recomputing statistics, leading to different choices in execution plans and variations in query performance.

This section also provides information about estimating `ANALYZE TABLE` complexity, which may be useful when attempting to achieve a balance between accurate statistics and `ANALYZE TABLE` execution time.

15.8.10.1 Configuring Persistent Optimizer Statistics Parameters

The persistent optimizer statistics feature improves plan stability by storing statistics to disk and making them persistent across server restarts so that the optimizer is more likely to make consistent choices each time for a given query.

Optimizer statistics are persisted to disk when `innodb_stats_persistent=ON` or when individual tables are defined with `STATS_PERSISTENT=1`. `innodb_stats_persistent` is enabled by default.

Formerly, optimizer statistics were cleared when restarting the server and after some other types of operations, and recomputed on the next table access. Consequently, different estimates could be produced when recalculating statistics leading to different choices in query execution plans and variation in query performance.

Persistent statistics are stored in the `mysql.innodb_table_stats` and `mysql.innodb_index_stats` tables. See InnoDB Persistent Statistics Tables.

If you prefer not to persist optimizer statistics to disk, see Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”

Configuring Automatic Statistics Calculation for Persistent Optimizer Statistics

The `innodb_stats_auto_recalc` variable, which is enabled by default, controls whether statistics are calculated automatically when a table undergoes changes to more than 10% of its rows. You can also

configure automatic statistics recalculation for individual tables by specifying the STATS_AUTO_RECALC clause when creating or altering a table.

Because of the asynchronous nature of automatic statistics recalculation, which occurs in the background, statistics may not be recalculated instantly after running a DML operation that affects more than 10% of a table, even when innodb_stats_auto_recalc is enabled. Statistics recalculation can be delayed by few seconds in some cases. If up-to-date statistics are required immediately, run ANALYZE TABLE to initiate a synchronous (foreground) recalculation of statistics.

If innodb_stats_auto_recalc is disabled, you can ensure the accuracy of optimizer statistics by executing the ANALYZE TABLE statement after making substantial changes to indexed columns. You might also consider adding ANALYZE TABLE to setup scripts that you run after loading data, and running ANALYZE TABLE on a schedule at times of low activity.

When an index is added to an existing table, or when a column is added or dropped, index statistics are calculated and added to the innodb_index_stats table regardless of the value of innodb_stats_auto_recalc.

Configuring Optimizer Statistics Parameters for Individual Tables

innodb_stats_persistent, innodb_stats_auto_recalc, and innodb_stats_persistent_sample_pages are global variables. To override these system-wide settings and configure optimizer statistics parameters for individual tables, you can define STATS_PERSISTENT, STATS_AUTO_RECALC, and STATS_SAMPLE_PAGES clauses in CREATE TABLE or ALTER TABLE statements.

- STATS_PERSISTENT specifies whether to enable persistent statistics for an InnoDB table. The value DEFAULT causes the persistent statistics setting for the table to be determined by the innodb_stats_persistent setting. A value of 1 enables persistent statistics for the table, while a value of 0 disables the feature. After enabling persistent statistics for an individual table, use ANALYZE TABLE to calculate statistics after table data is loaded.
- STATS_AUTO_RECALC specifies whether to automatically recalculate persistent statistics. The value DEFAULT causes the persistent statistics setting for the table to be determined by the innodb_stats_auto_recalc setting. A value of 1 causes statistics to be recalculated when 10% of table data has changed. A value 0 prevents automatic recalculation for the table. When using a value of 0, use ANALYZE TABLE to recalculate statistics after making substantial changes to the table.
- STATS_SAMPLE_PAGES specifies the number of index pages to sample when cardinality and other statistics are calculated for an indexed column, by an ANALYZE TABLE operation, for example.

All three clauses are specified in the following CREATE TABLE example:

```
CREATE TABLE `t1` (
`id` int(8) NOT NULL auto_increment,
`data` varchar(255),
`date` datetime,
PRIMARY KEY  (`id`),
INDEX `DATE_IX` (`date`)
) ENGINE=InnoDB,
  STATS_PERSISTENT=1,
  STATS_AUTO_RECALC=1,
  STATS_SAMPLE_PAGES=25;
```

Configuring the Number of Sampled Pages for InnoDB Optimizer Statistics

The optimizer uses estimated statistics about key distributions to choose the indexes for an execution plan, based on the relative selectivity of the index. Operations such as ANALYZE TABLE cause InnoDB

to sample random pages from each index on a table to estimate the cardinality of the index. This sampling technique is known as a random dive.

The `innodb_stats_persistent_sample_pages` controls the number of sampled pages. You can adjust the setting at runtime to manage the quality of statistics estimates used by the optimizer. The default value is 20. Consider modifying the setting when encountering the following issues:

1. *Statistics are not accurate enough and the optimizer chooses suboptimal plans*, as shown in `EXPLAIN` output. You can check the accuracy of statistics by comparing the actual cardinality of an index (determined by running `SELECT DISTINCT` on the index columns) with the estimates in the `mysql.innodb_index_stats` table.

   If it is determined that statistics are not accurate enough, the value of `innodb_stats_persistent_sample_pages` should be increased until the statistics estimates are sufficiently accurate. Increasing `innodb_stats_persistent_sample_pages` too much, however, could cause `ANALYZE TABLE` to run slowly.

2. *`ANALYZE TABLE` is too slow*. In this case `innodb_stats_persistent_sample_pages` should be decreased until `ANALYZE TABLE` execution time is acceptable. Decreasing the value too much, however, could lead to the first problem of inaccurate statistics and suboptimal query execution plans.

   If a balance cannot be achieved between accurate statistics and `ANALYZE TABLE` execution time, consider decreasing the number of indexed columns in the table or limiting the number of partitions to reduce `ANALYZE TABLE` complexity. The number of columns in the table's primary key is also important to consider, as primary key columns are appended to each nonunique index.

   For related information, see Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

Including Delete-marked Records in Persistent Statistics Calculations

By default, `InnoDB` reads uncommitted data when calculating statistics. In the case of an uncommitted transaction that deletes rows from a table, delete-marked records are excluded when calculating row estimates and index statistics, which can lead to non-optimal execution plans for other transactions that are operating on the table concurrently using a transaction isolation level other than `READ UNCOMMITTED`. To avoid this scenario, `innodb_stats_include_delete_marked` can be enabled to ensure that delete-marked records are included when calculating persistent optimizer statistics.

When `innodb_stats_include_delete_marked` is enabled, `ANALYZE TABLE` considers delete-marked records when recalculating statistics.

`innodb_stats_include_delete_marked` is a global setting that affects all `InnoDB` tables, and it is only applicable to persistent optimizer statistics.

InnoDB Persistent Statistics Tables

The persistent statistics feature relies on the internally managed tables in the `mysql` database, named `innodb_table_stats` and `innodb_index_stats`. These tables are set up automatically in all install, upgrade, and build-from-source procedures.

Table 15.7 Columns of innodb_table_stats

Column name	Description
`database_name`	Database name
`table_name`	Table name, partition name, or subpartition name

Column name	Description
last_update	A timestamp indicating the last time that InnoDB updated this row
n_rows	The number of rows in the table
clustered_index_size	The size of the primary index, in pages
sum_of_other_index_sizes	The total size of other (non-primary) indexes, in pages

Table 15.8 Columns of innodb_index_stats

Column name	Description
database_name	Database name
table_name	Table name, partition name, or subpartition name
index_name	Index name
last_update	A timestamp indicating the last time the row was updated
stat_name	The name of the statistic, whose value is reported in the stat_value column
stat_value	The value of the statistic that is named in stat_name column
sample_size	The number of pages sampled for the estimate provided in the stat_value column
stat_description	Description of the statistic that is named in the stat_name column

The innodb_table_stats and innodb_index_stats tables include a last_update column that shows when index statistics were last updated:

```
mysql> SELECT * FROM innodb_table_stats \G
*************************** 1. row ***************************
           database_name: sakila
              table_name: actor
             last_update: 2014-05-28 16:16:44
                  n_rows: 200
    clustered_index_size: 1
sum_of_other_index_sizes: 1
...
```

```
mysql> SELECT * FROM innodb_index_stats \G
*************************** 1. row ***************************
   database_name: sakila
      table_name: actor
      index_name: PRIMARY
     last_update: 2014-05-28 16:16:44
       stat_name: n_diff_pfx01
      stat_value: 200
     sample_size: 1
     ...
```

The innodb_table_stats and innodb_index_stats tables can be updated manually, which makes it possible to force a specific query optimization plan or test alternative plans without modifying the database. If you manually update statistics, use the FLUSH TABLE *tbl_name* statement to load the updated statistics.

Persistent statistics are considered local information, because they relate to the server instance. The innodb_table_stats and innodb_index_stats tables are therefore not replicated when automatic statistics recalculation takes place. If you run ANALYZE TABLE to initiate a synchronous recalculation of statistics, the statement is replicated (unless you suppressed logging for it), and recalculation takes place on replication slaves.

InnoDB Persistent Statistics Tables Example

The `innodb_table_stats` table contains one row for each table. The following example demonstrates the type of data collected.

Table `t1` contains a primary index (columns `a`, `b`) secondary index (columns `c`, `d`), and unique index (columns `e`, `f`):

```
CREATE TABLE t1 (
a INT, b INT, c INT, d INT, e INT, f INT,
PRIMARY KEY (a, b), KEY i1 (c, d), UNIQUE KEY i2uniq (e, f)
) ENGINE=INNODB;
```

After inserting five rows of sample data, table `t1` appears as follows:

```
mysql> SELECT * FROM t1;
+---+---+------+------+------+------+
| a | b | c    | d    | e    | f    |
+---+---+------+------+------+------+
| 1 | 1 |   10 |   11 |  100 |  101 |
| 1 | 2 |   10 |   11 |  200 |  102 |
| 1 | 3 |   10 |   11 |  100 |  103 |
| 1 | 4 |   10 |   12 |  200 |  104 |
| 1 | 5 |   10 |   12 |  100 |  105 |
+---+---+------+------+------+------+
```

To immediately update statistics, run `ANALYZE TABLE` (if `innodb_stats_auto_recalc` is enabled, statistics are updated automatically within a few seconds assuming that the 10% threshold for changed table rows is reached):

```
mysql> ANALYZE TABLE t1;
+---------+---------+----------+----------+
| Table   | Op      | Msg_type | Msg_text |
+---------+---------+----------+----------+
| test.t1 | analyze | status   | OK       |
+---------+---------+----------+----------+
```

Table statistics for table `t1` show the last time `InnoDB` updated the table statistics (`2014-03-14 14:36:34`), the number of rows in the table (`5`), the clustered index size (`1` page), and the combined size of the other indexes (`2` pages).

```
mysql> SELECT * FROM mysql.innodb_table_stats WHERE table_name like 't1'\G
*************************** 1. row ***************************
           database_name: test
              table_name: t1
             last_update: 2014-03-14 14:36:34
                 n_rows: 5
    clustered_index_size: 1
sum_of_other_index_sizes: 2
```

The `innodb_index_stats` table contains multiple rows for each index. Each row in the `innodb_index_stats` table provides data related to a particular index statistic which is named in the `stat_name` column and described in the `stat_description` column. For example:

```
mysql> SELECT index_name, stat_name, stat_value, stat_description
       FROM mysql.innodb_index_stats WHERE table_name like 't1';
+------------+--------------+------------+---------------------------------+
| index_name | stat_name    | stat_value | stat_description                |
```

```
+------------+-------------+------------+----------------------------------+
| PRIMARY    | n_diff_pfx01 |          1 | a                                |
| PRIMARY    | n_diff_pfx02 |          5 | a,b                              |
| PRIMARY    | n_leaf_pages |          1 | Number of leaf pages in the index |
| PRIMARY    | size         |          1 | Number of pages in the index      |
| i1         | n_diff_pfx01 |          1 | c                                |
| i1         | n_diff_pfx02 |          2 | c,d                              |
| i1         | n_diff_pfx03 |          2 | c,d,a                            |
| i1         | n_diff_pfx04 |          5 | c,d,a,b                          |
| i1         | n_leaf_pages |          1 | Number of leaf pages in the index |
| i1         | size         |          1 | Number of pages in the index      |
| i2uniq     | n_diff_pfx01 |          2 | e                                |
| i2uniq     | n_diff_pfx02 |          5 | e,f                              |
| i2uniq     | n_leaf_pages |          1 | Number of leaf pages in the index |
| i2uniq     | size         |          1 | Number of pages in the index      |
+------------+-------------+------------+----------------------------------+
```

The `stat_name` column shows the following types of statistics:

- `size`: Where `stat_name=size`, the `stat_value` column displays the total number of pages in the index.
- `n_leaf_pages`: Where `stat_name=n_leaf_pages`, the `stat_value` column displays the number of leaf pages in the index.
- `n_diff_pfx`*`NN`*: Where `stat_name=n_diff_pfx01`, the `stat_value` column displays the number of distinct values in the first column of the index. Where `stat_name=n_diff_pfx02`, the `stat_value` column displays the number of distinct values in the first two columns of the index, and so on. Where `stat_name=n_diff_pfx`*`NN`*, the `stat_description` column shows a comma separated list of the index columns that are counted.

To further illustrate the `n_diff_pfx`*`NN`* statistic, which provides cardinality data, consider once again the `t1` table example that was introduced previously. As shown below, the `t1` table is created with a primary index (columns `a`, `b`), a secondary index (columns `c`, `d`), and a unique index (columns `e`, `f`):

```
CREATE TABLE t1 (
  a INT, b INT, c INT, d INT, e INT, f INT,
  PRIMARY KEY (a, b), KEY i1 (c, d), UNIQUE KEY i2uniq (e, f)
) ENGINE=INNODB;
```

After inserting five rows of sample data, table `t1` appears as follows:

```
mysql> SELECT * FROM t1;
+---+---+------+------+------+------+
| a | b | c    | d    | e    | f    |
+---+---+------+------+------+------+
| 1 | 1 |   10 |   11 |  100 |  101 |
| 1 | 2 |   10 |   11 |  200 |  102 |
| 1 | 3 |   10 |   11 |  100 |  103 |
| 1 | 4 |   10 |   12 |  200 |  104 |
| 1 | 5 |   10 |   12 |  100 |  105 |
+---+---+------+------+------+------+
```

When you query the `index_name`, `stat_name`, `stat_value`, and `stat_description`, where `stat_name LIKE 'n_diff%'`, the following result set is returned:

```
mysql> SELECT index_name, stat_name, stat_value, stat_description
       FROM mysql.innodb_index_stats
       WHERE table_name like 't1' AND stat_name LIKE 'n_diff%';
+------------+-------------+------------+------------------+
```

```
| index_name | stat_name    | stat_value | stat_description |
+------------+--------------+------------+------------------+
| PRIMARY    | n_diff_pfx01 |          1 | a                |
| PRIMARY    | n_diff_pfx02 |          5 | a,b              |
| i1         | n_diff_pfx01 |          1 | c                |
| i1         | n_diff_pfx02 |          2 | c,d              |
| i1         | n_diff_pfx03 |          2 | c,d,a            |
| i1         | n_diff_pfx04 |          5 | c,d,a,b          |
| i2uniq     | n_diff_pfx01 |          2 | e                |
| i2uniq     | n_diff_pfx02 |          5 | e,f              |
+------------+--------------+------------+------------------+
```

For the `PRIMARY` index, there are two `n_diff%` rows. The number of rows is equal to the number of columns in the index.

Note

For nonunique indexes, `InnoDB` appends the columns of the primary key.

- Where `index_name=PRIMARY` and `stat_name=n_diff_pfx01`, the `stat_value` is `1`, which indicates that there is a single distinct value in the first column of the index (column `a`). The number of distinct values in column `a` is confirmed by viewing the data in column `a` in table `t1`, in which there is a single distinct value (`1`). The counted column (`a`) is shown in the `stat_description` column of the result set.

- Where `index_name=PRIMARY` and `stat_name=n_diff_pfx02`, the `stat_value` is `5`, which indicates that there are five distinct values in the two columns of the index (`a,b`). The number of distinct values in columns `a` and `b` is confirmed by viewing the data in columns `a` and `b` in table `t1`, in which there are five distinct values: (`1,1`), (`1,2`), (`1,3`), (`1,4`) and (`1,5`). The counted columns (`a,b`) are shown in the `stat_description` column of the result set.

For the secondary index (`i1`), there are four `n_diff%` rows. Only two columns are defined for the secondary index (`c,d`) but there are four `n_diff%` rows for the secondary index because `InnoDB` suffixes all nonunique indexes with the primary key. As a result, there are four `n_diff%` rows instead of two to account for the both the secondary index columns (`c,d`) and the primary key columns (`a,b`).

- Where `index_name=i1` and `stat_name=n_diff_pfx01`, the `stat_value` is `1`, which indicates that there is a single distinct value in the first column of the index (column `c`). The number of distinct values in column `c` is confirmed by viewing the data in column `c` in table `t1`, in which there is a single distinct value: (`10`). The counted column (`c`) is shown in the `stat_description` column of the result set.

- Where `index_name=i1` and `stat_name=n_diff_pfx02`, the `stat_value` is `2`, which indicates that there are two distinct values in the first two columns of the index (`c,d`). The number of distinct values in columns `c` an `d` is confirmed by viewing the data in columns `c` and `d` in table `t1`, in which there are two distinct values: (`10,11`) and (`10,12`). The counted columns (`c,d`) are shown in the `stat_description` column of the result set.

- Where `index_name=i1` and `stat_name=n_diff_pfx03`, the `stat_value` is `2`, which indicates that there are two distinct values in the first three columns of the index (`c,d,a`). The number of distinct values in columns `c`, `d`, and `a` is confirmed by viewing the data in column `c`, `d`, and `a` in table `t1`, in which there are two distinct values: (`10,11,1`) and (`10,12,1`). The counted columns (`c,d,a`) are shown in the `stat_description` column of the result set.

- Where `index_name=i1` and `stat_name=n_diff_pfx04`, the `stat_value` is `5`, which indicates that there are five distinct values in the four columns of the index (`c,d,a,b`). The number of distinct values in columns `c`, `d`, `a` and `b` is confirmed by viewing the data in columns `c`, `d`, `a`, and `b` in table `t1`, in which there are five distinct values: (`10,11,1,1`), (`10,11,1,2`), (`10,11,1,3`), (`10,12,1,4`), and

(10,12,1,5). The counted columns (c,d,a,b) are shown in the stat_description column of the result set.

For the unique index (i2uniq), there are two n_diff% rows.

- Where index_name=i2uniq and stat_name=n_diff_pfx01, the stat_value is 2, which indicates that there are two distinct values in the first column of the index (column e). The number of distinct values in column e is confirmed by viewing the data in column e in table t1, in which there are two distinct values: (100) and (200). The counted column (e) is shown in the stat_description column of the result set.
- Where index_name=i2uniq and stat_name=n_diff_pfx02, the stat_value is 5, which indicates that there are five distinct values in the two columns of the index (e,f). The number of distinct values in columns e and f is confirmed by viewing the data in columns e and f in table t1, in which there are five distinct values: (100,101), (200,102), (100,103), (200,104), and (100,105). The counted columns (e,f) are shown in the stat_description column of the result set.

Retrieving Index Size Using the innodb_index_stats Table

You can retrieve the index size for tables, partitions, or subpartitions can using the innodb_index_stats table. In the following example, index sizes are retrieved for table t1. For a definition of table t1 and corresponding index statistics, see InnoDB Persistent Statistics Tables Example.

```
mysql> SELECT SUM(stat_value) pages, index_name,
       SUM(stat_value)*@@innodb_page_size size
       FROM mysql.innodb_index_stats WHERE table_name='t1'
       AND stat_name = 'size' GROUP BY index_name;
+-------+------------+-------+
| pages | index_name | size  |
+-------+------------+-------+
|     1 | PRIMARY    | 16384 |
|     1 | i1         | 16384 |
|     1 | i2uniq     | 16384 |
+-------+------------+-------+
```

For partitions or subpartitions, you can use the same query with a modified WHERE clause to retrieve index sizes. For example, the following query retrieves index sizes for partitions of table t1:

```
mysql> SELECT SUM(stat_value) pages, index_name,
       SUM(stat_value)*@@innodb_page_size size
       FROM mysql.innodb_index_stats WHERE table_name like 't1#P%'
       AND stat_name = 'size' GROUP BY index_name;
```

15.8.10.2 Configuring Non-Persistent Optimizer Statistics Parameters

This section describes how to configure non-persistent optimizer statistics. Optimizer statistics are not persisted to disk when innodb_stats_persistent=OFF or when individual tables are created or altered with STATS_PERSISTENT=0. Instead, statistics are stored in memory, and are lost when the server is shut down. Statistics are also updated periodically by certain operations and under certain conditions.

Optimizer statistics are persisted to disk by default, enabled by the innodb_stats_persistent configuration option. For information about persistent optimizer statistics, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

Optimizer Statistics Updates

Non-persistent optimizer statistics are updated when:

- Running `ANALYZE TABLE`.
- Running `SHOW TABLE STATUS`, `SHOW INDEX`, or querying the `INFORMATION_SCHEMA.TABLES` or `INFORMATION_SCHEMA.STATISTICS` tables with the `innodb_stats_on_metadata` option enabled.

  The default setting for `innodb_stats_on_metadata` is `OFF`. Enabling `innodb_stats_on_metadata` may reduce access speed for schemas that have a large number of tables or indexes, and reduce stability of execution plans for queries that involve `InnoDB` tables. `innodb_stats_on_metadata` is configured globally using a `SET` statement.

```
SET GLOBAL innodb_stats_on_metadata=ON
```

  Note

  `innodb_stats_on_metadata` only applies when optimizer statistics are configured to be non-persistent (when `innodb_stats_persistent` is disabled).

- Starting a `mysql` client with the `--auto-rehash` option enabled, which is the default. The `auto-rehash` option causes all `InnoDB` tables to be opened, and the open table operations cause statistics to be recalculated.

  To improve the start up time of the `mysql` client and to updating statistics, you can turn off `auto-rehash` using the `--disable-auto-rehash` option. The `auto-rehash` feature enables automatic name completion of database, table, and column names for interactive users.

- A table is first opened.
- `InnoDB` detects that 1 / 16 of table has been modified since the last time statistics were updated.

Configuring the Number of Sampled Pages

The MySQL query optimizer uses estimated statistics about key distributions to choose the indexes for an execution plan, based on the relative selectivity of the index. When `InnoDB` updates optimizer statistics, it samples random pages from each index on a table to estimate the cardinality of the index. (This technique is known as random dives.)

To give you control over the quality of the statistics estimate (and thus better information for the query optimizer), you can change the number of sampled pages using the parameter `innodb_stats_transient_sample_pages`. The default number of sampled pages is 8, which could be insufficient to produce an accurate estimate, leading to poor index choices by the query optimizer. This technique is especially important for large tables and tables used in joins. Unnecessary full table scans for such tables can be a substantial performance issue. See Section 8.2.1.23, “Avoiding Full Table Scans” for tips on tuning such queries. `innodb_stats_transient_sample_pages` is a global parameter that can be set at runtime.

The value of `innodb_stats_transient_sample_pages` affects the index sampling for all `InnoDB` tables and indexes when `innodb_stats_persistent=0`. Be aware of the following potentially significant impacts when you change the index sample size:

- Small values like 1 or 2 can result in inaccurate estimates of cardinality.
- Increasing the `innodb_stats_transient_sample_pages` value might require more disk reads. Values much larger than 8 (say, 100), can cause a significant slowdown in the time it takes to open a table or execute `SHOW TABLE STATUS`.
- The optimizer might choose very different query plans based on different estimates of index selectivity.

Whatever value of `innodb_stats_transient_sample_pages` works best for a system, set the option and leave it at that value. Choose a value that results in reasonably accurate estimates for all tables in your database without requiring excessive I/O. Because the statistics are automatically recalculated at various times other than on execution of `ANALYZE TABLE`, it does not make sense to increase the index sample size, run `ANALYZE TABLE`, then decrease sample size again.

Smaller tables generally require fewer index samples than larger tables. If your database has many large tables, consider using a higher value for `innodb_stats_transient_sample_pages` than if you have mostly smaller tables.

15.8.10.3 Estimating ANALYZE TABLE Complexity for InnoDB Tables

`ANALYZE TABLE` complexity for `InnoDB` tables is dependent on:

- The number of pages sampled, as defined by `innodb_stats_persistent_sample_pages`.
- The number of indexed columns in a table
- The number of partitions. If a table has no partitions, the number of partitions is considered to be 1.

Using these parameters, an approximate formula for estimating `ANALYZE TABLE` complexity would be:

The value of `innodb_stats_persistent_sample_pages` * number of indexed columns in a table * the number of partitions

Typically, the greater the resulting value, the greater the execution time for `ANALYZE TABLE`.

Note

`innodb_stats_persistent_sample_pages` defines the number of pages sampled at a global level. To set the number of pages sampled for an individual table, use the `STATS_SAMPLE_PAGES` option with `CREATE TABLE` or `ALTER TABLE`. For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

If `innodb_stats_persistent=OFF`, the number of pages sampled is defined by `innodb_stats_transient_sample_pages`. See Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters” for additional information.

For a more in-depth approach to estimating `ANALYZE TABLE` complexity, consider the following example.

In Big O notation, `ANALYZE TABLE` complexity is described as:

```
O(n_sample
  * (n_cols_in_uniq_i
     + n_cols_in_non_uniq_i
     + n_cols_in_pk * (1 + n_non_uniq_i))
  * n_part)
```

where:

- `n_sample` is the number of pages sampled (defined by `innodb_stats_persistent_sample_pages`)
- `n_cols_in_uniq_i` is total number of all columns in all unique indexes (not counting the primary key columns)
- `n_cols_in_non_uniq_i` is the total number of all columns in all nonunique indexes

- `n_cols_in_pk` is the number of columns in the primary key (if a primary key is not defined, `InnoDB` creates a single column primary key internally)
- `n_non_uniq_i` is the number of nonunique indexes in the table
- `n_part` is the number of partitions. If no partitions are defined, the table is considered to be a single partition.

Now, consider the following table (table `t`), which has a primary key (2 columns), a unique index (2 columns), and two nonunique indexes (two columns each):

```
CREATE TABLE t (
  a INT,
  b INT,
  c INT,
  d INT,
  e INT,
  f INT,
  g INT,
  h INT,
  PRIMARY KEY (a, b),
  UNIQUE KEY i1uniq (c, d),
  KEY i2nonuniq (e, f),
  KEY i3nonuniq (g, h)
);
```

For the column and index data required by the algorithm described above, query the `mysql.innodb_index_stats` persistent index statistics table for table `t`. The `n_diff_pfx%` statistics show the columns that are counted for each index. For example, columns `a` and `b` are counted for the primary key index. For the nonunique indexes, the primary key columns (a,b) are counted in addition to the user defined columns.

Note

For additional information about the `InnoDB` persistent statistics tables, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”

```
mysql> SELECT index_name, stat_name, stat_description
       FROM mysql.innodb_index_stats WHERE
       database_name='test' AND
       table_name='t' AND
       stat_name like 'n_diff_pfx%';
  +------------+--------------+------------------+
  | index_name | stat_name    | stat_description |
  +------------+--------------+------------------+
  | PRIMARY    | n_diff_pfx01 | a                |
  | PRIMARY    | n_diff_pfx02 | a,b              |
  | i1uniq     | n_diff_pfx01 | c                |
  | i1uniq     | n_diff_pfx02 | c,d              |
  | i2nonuniq  | n_diff_pfx01 | e                |
  | i2nonuniq  | n_diff_pfx02 | e,f              |
  | i2nonuniq  | n_diff_pfx03 | e,f,a            |
  | i2nonuniq  | n_diff_pfx04 | e,f,a,b          |
  | i3nonuniq  | n_diff_pfx01 | g                |
  | i3nonuniq  | n_diff_pfx02 | g,h              |
  | i3nonuniq  | n_diff_pfx03 | g,h,a            |
  | i3nonuniq  | n_diff_pfx04 | g,h,a,b          |
  +------------+--------------+------------------+
```

Based on the index statistics data shown above and the table definition, the following values can be determined:

- `n_cols_in_uniq_i`, the total number of all columns in all unique indexes not counting the primary key columns, is 2 (`c` and `d`)
- `n_cols_in_non_uniq_i`, the total number of all columns in all nonunique indexes, is 4 (`e`, `f`, `g` and `h`)
- `n_cols_in_pk`, the number of columns in the primary key, is 2 (`a` and `b`)
- `n_non_uniq_i`, the number of nonunique indexes in the table, is 2 (`i2nonuniq` and `i3nonuniq`))
- `n_part`, the number of partitions, is 1.

You can now calculate `innodb_stats_persistent_sample_pages` * (2 + 4 + 2 * (1 + 2)) * 1 to determine the number of leaf pages that are scanned. With `innodb_stats_persistent_sample_pages` set to the default value of `20`, and with a default page size of 16 `KiB` (`innodb_page_size`=16384), you can then estimate that 20 * 12 * 16384 `bytes` are read for table `t`, or about 4 `MiB`.

Note

All 4 `MiB` may not be read from disk, as some leaf pages may already be cached in the buffer pool.

15.8.11 Configuring the Merge Threshold for Index Pages

You can configure the `MERGE_THRESHOLD` value for index pages. If the “page-full” percentage for an index page falls below the `MERGE_THRESHOLD` value when a row is deleted or when a row is shortened by an `UPDATE` operation, `InnoDB` attempts to merge the index page with a neighboring index page. The default `MERGE_THRESHOLD` value is 50, which is the previously hardcoded value. The minimum `MERGE_THRESHOLD` value is 1 and the maximum value is 50.

When the “page-full” percentage for an index page falls below 50%, which is the default `MERGE_THRESHOLD` setting, `InnoDB` attempts to merge the index page with a neighboring page. If both pages are close to 50% full, a page split can occur soon after the pages are merged. If this merge-split behavior occurs frequently, it can have an adverse affect on performance. To avoid frequent merge-splits, you can lower the `MERGE_THRESHOLD` value so that `InnoDB` attempts page merges at a lower “page-full” percentage. Merging pages at a lower page-full percentage leaves more room in index pages and helps reduce merge-split behavior.

The `MERGE_THRESHOLD` for index pages can be defined for a table or for individual indexes. A `MERGE_THRESHOLD` value defined for an individual index takes priority over a `MERGE_THRESHOLD` value defined for the table. If undefined, the `MERGE_THRESHOLD` value defaults to 50.

Setting MERGE_THRESHOLD for a Table

You can set the `MERGE_THRESHOLD` value for a table using the *`table_option`* `COMMENT` clause of the `CREATE TABLE` statement. For example:

```
CREATE TABLE t1 (
   id INT,
  KEY id_index (id)
) COMMENT='MERGE_THRESHOLD=45';
```

You can also set the `MERGE_THRESHOLD` value for an existing table using the *`table_option`* `COMMENT` clause with `ALTER TABLE`:

```
CREATE TABLE t1 (
```

```
   id INT,
  KEY id_index (id)
);

ALTER TABLE t1 COMMENT='MERGE_THRESHOLD=40';
```

Setting MERGE_THRESHOLD for Individual Indexes

To set the MERGE_THRESHOLD value for an individual index, you can use the *index_option* COMMENT clause with CREATE TABLE, ALTER TABLE, or CREATE INDEX, as shown in the following examples:

- Setting MERGE_THRESHOLD for an individual index using CREATE TABLE:

```
CREATE TABLE t1 (
   id INT,
  KEY id_index (id) COMMENT 'MERGE_THRESHOLD=40'
);
```

- Setting MERGE_THRESHOLD for an individual index using ALTER TABLE:

```
CREATE TABLE t1 (
   id INT,
  KEY id_index (id)
);

ALTER TABLE t1 DROP KEY id_index;
ALTER TABLE t1 ADD KEY id_index (id) COMMENT 'MERGE_THRESHOLD=40';
```

- Setting MERGE_THRESHOLD for an individual index using CREATE INDEX:

```
CREATE TABLE t1 (id INT);
CREATE INDEX id_index ON t1 (id) COMMENT 'MERGE_THRESHOLD=40';
```

Note

You cannot modify the MERGE_THRESHOLD value at the index level for GEN_CLUST_INDEX, which is the clustered index created by InnoDB when an InnoDB table is created without a primary key or unique key index. You can only modify the MERGE_THRESHOLD value for GEN_CLUST_INDEX by setting MERGE_THRESHOLD for the table.

Querying the MERGE_THRESHOLD Value for an Index

The current MERGE_THRESHOLD value for an index can be obtained by querying the INNODB_INDEXES table. For example:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_INDEXES WHERE NAME='id_index' \G
*************************** 1. row ***************************
       INDEX_ID: 91
           NAME: id_index
       TABLE_ID: 68
           TYPE: 0
       N_FIELDS: 1
        PAGE_NO: 4
          SPACE: 57
MERGE_THRESHOLD: 40
```

You can use SHOW CREATE TABLE to view the MERGE_THRESHOLD value for a table, if explicitly defined using the *table_option* COMMENT clause:

```
mysql> SHOW CREATE TABLE t2 \G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE `t2` (
  `id` int(11) DEFAULT NULL,
  KEY `id_index` (`id`) COMMENT 'MERGE_THRESHOLD=40'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
```

Note

A `MERGE_THRESHOLD` value defined at the index level takes priority over a `MERGE_THRESHOLD` value defined for the table. If undefined, `MERGE_THRESHOLD` defaults to 50% (`MERGE_THRESHOLD=50`, which is the previously hardcoded value.

Likewise, you can use `SHOW INDEX` to view the `MERGE_THRESHOLD` value for an index, if explicitly defined using the *`index_option`* `COMMENT` clause:

```
mysql> SHOW INDEX FROM t2 \G
*************************** 1. row ***************************
        Table: t2
   Non_unique: 1
     Key_name: id_index
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 0
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment: MERGE_THRESHOLD=40
```

Measuring the Effect of MERGE_THRESHOLD Settings

The `INNODB_METRICS` table provides two counters that can be used to measure the effect of a `MERGE_THRESHOLD` setting on index page merges.

```
mysql> SELECT NAME, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS
       WHERE NAME like '%index_page_merge%';
+-----------------------------+---------------------------------------+
| NAME                        | COMMENT                               |
+-----------------------------+---------------------------------------+
| index_page_merge_attempts   | Number of index page merge attempts   |
| index_page_merge_successful | Number of successful index page merges |
+-----------------------------+---------------------------------------+
```

When lowering the `MERGE_THRESHOLD` value, the objectives are:

- A smaller number of page merge attempts and successful page merges
- A similar number of page merge attempts and successful page merges

A `MERGE_THRESHOLD` setting that is too small could result in large data files due to an excessive amount of empty page space.

For information about using `INNODB_METRICS` counters, see Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

15.8.12 Enabling Automatic Configuration for a Dedicated MySQL Server

When `innodb_dedicated_server` is enabled, `InnoDB` automatically configures the following variables:

- `innodb_buffer_pool_size`
- `innodb_log_file_size`
- `innodb_log_files_in_group` (as of MySQL 8.0.14)
- `innodb_flush_method`

Only consider enabling `innodb_dedicated_server` if the MySQL instance resides on a dedicated server where it can use all available system resources. For example, consider enabling if you run MySQL Server in a Docker container or dedicated VM. Enabling `innodb_dedicated_server` is not recommended if the MySQL instance shares system resources with other applications.

The information that follows describes how each variable is automatically configured.

- `innodb_buffer_pool_size`

  Buffer pool size is configured according to the amount of memory detected on the server.

  Table 15.9 Automatically Configured Buffer Pool Size

Detected Server Memory	Buffer Pool Size
Less than 1GB	128MiB (the default value)
1GB to 4GB	*`detected server memory`* * 0.5
Greater than 4GB	*`detected server memory`* * 0.75

- `innodb_log_file_size`

  As of MySQL 8.0.14, log file size is configured according to the automatically configured buffer pool size.

  Table 15.10 Automatically Configured Log File Size

Buffer Pool Size	Log File Size
Less than 8GB	512MiB
8GB to 128GB	1024MiB
Greater than 128GB	2048MiB

Note

Prior to MySQL 8.0.14, the `innodb_log_file_size` variable was automatically configured according to the amount of memory detected on the server, as shown below:

Table 15.11 Automatically Configured Log File Size (MySQL 8.0.13 and Earlier)

Detected Server Memory	Log File Size
< 1GB	48MiB (the default value)
<= 4GB	128MiB
<= 8GB	512MiB

Detected Server Memory	Log File Size
<= 16GB	1024MiB
> 16GB	2048MiB

- `innodb_log_files_in_group`

  The number of log files is configured according to the automatically configured buffer pool size (in gigabytes). Automatic configuration of the `innodb_log_files_in_group` variable was added in MySQL 8.0.14.

  Table 15.12 Automatically Configured Number of Log Files

Buffer Pool Size	Number of Log Files
Less than 8GB	ROUND(*`buffer pool size`*)
8GB to 128GB	ROUND(*`buffer pool size`* * 0.75)
Greater than 128GB	64

- `innodb_flush_method`

  The flush method is set to `O_DIRECT_NO_FSYNC` when `innodb_dedicated_server` is enabled. If the `O_DIRECT_NO_FSYNC` setting is not available, the default `innodb_flush_method` setting is used.

  `InnoDB` uses `O_DIRECT` during flushing I/O, but skips the `fsync()` system call after each write operation.

  Warning

  Prior to MySQL 8.0.14, this setting is not suitable for file systems such as XFS and EXT4, which require an `fsync()` system call to synchronize file system metadata changes.

  As of MySQL 8.0.14, `fsync()` is called after creating a new file, after increasing file size, and after closing a file, to ensure that file system metadata changes are synchronized. The `fsync()` system call is still skipped after each write operation.

  On storage devices with cache, data loss is possible if data files and redo log files reside on different storage devices, and a crash occurs before data file writes are flushed from the device cache. If you use or intend to use different storage devices for redo logs and data files, use `O_DIRECT` instead.

If an automatically configured option is configured explicitly in an option file or elsewhere, the explicitly specified setting is used, and a startup warning similar to this is printed to `stderr`:

```
[Warning] [000000] InnoDB: Option innodb_dedicated_server is ignored for
innodb_buffer_pool_size because innodb_buffer_pool_size=134217728 is specified
explicitly.
```

Explicit configuration of one option does not prevent the automatic configuration of other options. For example, if `innodb_dedicated_server` is enabled and `innodb_buffer_pool_size` is configured explicitly in an option file, `innodb_log_file_size` and `innodb_log_files_in_group` are automatically configured based on the implicit buffer pool size that is calculated according to the amount of memory detected on the server.

Automatically configured settings are evaluated and reconfigured if necessary each time the MySQL server is started.

15.9 InnoDB Table and Page Compression

This section provides information about the InnoDB table compression and InnoDB page compression features. The page compression feature is also referred to as transparent page compression.

Using the compression features of InnoDB, you can create tables where the data is stored in compressed form. Compression can help to improve both raw performance and scalability. The compression means less data is transferred between disk and memory, and takes up less space on disk and in memory. The benefits are amplified for tables with secondary indexes, because index data is compressed also. Compression can be especially important for SSD storage devices, because they tend to have lower capacity than HDD devices.

15.9.1 InnoDB Table Compression

This section describes InnoDB table compression, which is supported with InnoDB tables that reside in file_per_table tablespaces or general tablespaces. Table compression is enabled using the ROW_FORMAT=COMPRESSED attribute with CREATE TABLE or ALTER TABLE.

15.9.1.1 Overview of Table Compression

Because processors and cache memories have increased in speed more than disk storage devices, many workloads are disk-bound. Data compression enables smaller database size, reduced I/O, and improved throughput, at the small cost of increased CPU utilization. Compression is especially valuable for read-intensive applications, on systems with enough RAM to keep frequently used data in memory.

An InnoDB table created with ROW_FORMAT=COMPRESSED can use a smaller page size on disk than the configured innodb_page_size value. Smaller pages require less I/O to read from and write to disk, which is especially valuable for SSD devices.

The compressed page size is specified through the CREATE TABLE or ALTER TABLE KEY_BLOCK_SIZE parameter. The different page size requires that the table be placed in a file-per-table tablespace or general tablespace rather than in the system tablespace, as the system tablespace cannot store compressed tables. For more information, see Section 15.6.3.2, “File-Per-Table Tablespaces”, and Section 15.6.3.3, “General Tablespaces”.

The level of compression is the same regardless of the KEY_BLOCK_SIZE value. As you specify smaller values for KEY_BLOCK_SIZE, you get the I/O benefits of increasingly smaller pages. But if you specify a value that is too small, there is additional overhead to reorganize the pages when data values cannot be compressed enough to fit multiple rows in each page. There is a hard limit on how small KEY_BLOCK_SIZE can be for a table, based on the lengths of the key columns for each of its indexes. Specify a value that is too small, and the CREATE TABLE or ALTER TABLE statement fails.

In the buffer pool, the compressed data is held in small pages, with a page size based on the KEY_BLOCK_SIZE value. For extracting or updating the column values, MySQL also creates an uncompressed page in the buffer pool with the uncompressed data. Within the buffer pool, any updates to the uncompressed page are also re-written back to the equivalent compressed page. You might need to size your buffer pool to accommodate the additional data of both compressed and uncompressed pages, although the uncompressed pages are evicted from the buffer pool when space is needed, and then uncompressed again on the next access.

15.9.1.2 Creating Compressed Tables

Compressed tables can be created in file-per-table tablespaces or in general tablespaces. Table compression is not available for the InnoDB system tablespace. The system tablespace (space 0, the .ibdata files) can contain user-created tables, but it also contains internal system data, which is never

compressed. Thus, compression applies only to tables (and indexes) stored in file-per-table or general tablespaces.

Creating a Compressed Table in File-Per-Table Tablespace

To create a compressed table in a file-per-table tablespace, `innodb_file_per_table` must be enabled (the default). You can set this parameter in the MySQL configuration file (`my.cnf` or `my.ini`) or dynamically, using a `SET` statement.

After the `innodb_file_per_table` option is configured, specify the `ROW_FORMAT=COMPRESSED` clause or `KEY_BLOCK_SIZE` clause, or both, in a `CREATE TABLE` or `ALTER TABLE` statement to create a compressed table in a file-per-table tablespace.

For example, you might use the following statements:

```
SET GLOBAL innodb_file_per_table=1;
CREATE TABLE t1
 (c1 INT PRIMARY KEY)
 ROW_FORMAT=COMPRESSED
 KEY_BLOCK_SIZE=8;
```

Creating a Compressed Table in a General Tablespace

To create a compressed table in a general tablespace, `FILE_BLOCK_SIZE` must be defined for the general tablespace, which is specified when the tablespace is created. The `FILE_BLOCK_SIZE` value must be a valid compressed page size in relation to the `innodb_page_size` value, and the page size of the compressed table, defined by the `CREATE TABLE` or `ALTER TABLE KEY_BLOCK_SIZE` clause, must be equal to `FILE_BLOCK_SIZE/1024`. For example, if `innodb_page_size=16384` and `FILE_BLOCK_SIZE=8192`, the `KEY_BLOCK_SIZE` of the table must be 8. For more information, see Section 15.6.3.3, “General Tablespaces”.

The following example demonstrates creating a general tablespace and adding a compressed table. The example assumes a default `innodb_page_size` of 16K. The `FILE_BLOCK_SIZE` of 8192 requires that the compressed table have a `KEY_BLOCK_SIZE` of 8.

```
mysql> CREATE TABLESPACE `ts2` ADD DATAFILE 'ts2.ibd' FILE_BLOCK_SIZE = 8192 Engine=InnoDB;

mysql> CREATE TABLE t4 (c1 INT PRIMARY KEY) TABLESPACE ts2 ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8;
```

Notes

- As of MySQL 8.0, the tablespace file for a compressed table is created using the physical page size instead of the `InnoDB` page size, which makes the initial size of a tablespace file for an empty compressed table smaller than in previous MySQL releases.
- If you specify `ROW_FORMAT=COMPRESSED`, you can omit `KEY_BLOCK_SIZE`; the `KEY_BLOCK_SIZE` setting defaults to half the `innodb_page_size` value.
- If you specify a valid `KEY_BLOCK_SIZE` value, you can omit `ROW_FORMAT=COMPRESSED`; compression is enabled automatically.
- To determine the best value for `KEY_BLOCK_SIZE`, typically you create several copies of the same table with different values for this clause, then measure the size of the resulting `.ibd` files and see how well each performs with a realistic workload. For general tablespaces, keep in mind that dropping a table does not reduce the size of the general tablespace `.ibd` file, nor does it return disk space to the operating system. For more information, see Section 15.6.3.3, “General Tablespaces”.

- The KEY_BLOCK_SIZE value is treated as a hint; a different size could be used by InnoDB if necessary. For file-per-table tablespaces, the KEY_BLOCK_SIZE can only be less than or equal to the innodb_page_size value. If you specify a value greater than the innodb_page_size value, the specified value is ignored, a warning is issued, and KEY_BLOCK_SIZE is set to half of the innodb_page_size value. If innodb_strict_mode=ON, specifying an invalid KEY_BLOCK_SIZE value returns an error. For general tablespaces, valid KEY_BLOCK_SIZE values depend on the FILE_BLOCK_SIZE setting of the tablespace. For more information, see Section 15.6.3.3, “General Tablespaces”.
- InnoDB supports 32KB and 64KB page sizes but these page sizes do not support compression. For more information, refer to the innodb_page_size documentation.
- The default uncompressed size of InnoDB data pages is 16KB. Depending on the combination of option values, MySQL uses a page size of 1KB, 2KB, 4KB, 8KB, or 16KB for the tablespace data file (.ibd file). The actual compression algorithm is not affected by the KEY_BLOCK_SIZE value; the value determines how large each compressed chunk is, which in turn affects how many rows can be packed into each compressed page.
- When creating a compressed table in a file-per-table tablespace, setting KEY_BLOCK_SIZE equal to the InnoDB page size does not typically result in much compression. For example, setting KEY_BLOCK_SIZE=16 typically would not result in much compression, since the normal InnoDB page size is 16KB. This setting may still be useful for tables with many long BLOB, VARCHAR or TEXT columns, because such values often do compress well, and might therefore require fewer overflow pages as described in Section 15.9.1.5, “How Compression Works for InnoDB Tables”. For general tablespaces, a KEY_BLOCK_SIZE value equal to the InnoDB page size is not permitted. For more information, see Section 15.6.3.3, “General Tablespaces”.
- All indexes of a table (including the clustered index) are compressed using the same page size, as specified in the CREATE TABLE or ALTER TABLE statement. Table attributes such as ROW_FORMAT and KEY_BLOCK_SIZE are not part of the CREATE INDEX syntax for InnoDB tables, and are ignored if they are specified (although, if specified, they will appear in the output of the SHOW CREATE TABLE statement).
- For performance-related configuration options, see Section 15.9.1.3, “Tuning Compression for InnoDB Tables”.

Restrictions on Compressed Tables

- Compressed tables cannot be stored in the InnoDB system tablespace.
- General tablespaces can contain multiple tables, but compressed and uncompressed tables cannot coexist within the same general tablespace.
- Compression applies to an entire table and all its associated indexes, not to individual rows, despite the clause name ROW_FORMAT.
- InnoDB does not support compressed temporary tables. When innodb_strict_mode is enabled (the default), CREATE TEMPORARY TABLE returns errors if ROW_FORMAT=COMPRESSED or KEY_BLOCK_SIZE is specified. If innodb_strict_mode is disabled, warnings are issued and the temporary table is created using a non-compressed row format. The same restrictions apply to ALTER TABLE operations on temporary tables.

15.9.1.3 Tuning Compression for InnoDB Tables

Most often, the internal optimizations described in InnoDB Data Storage and Compression ensure that the system runs well with compressed data. However, because the efficiency of compression depends on the nature of your data, you can make decisions that affect the performance of compressed tables:

- Which tables to compress.
- What compressed page size to use.
- Whether to adjust the size of the buffer pool based on run-time performance characteristics, such as the amount of time the system spends compressing and uncompressing data. Whether the workload is more like a data warehouse (primarily queries) or an OLTP system (mix of queries and DML).
- If the system performs DML operations on compressed tables, and the way the data is distributed leads to expensive compression failures at runtime, you might adjust additional advanced configuration options.

Use the guidelines in this section to help make those architectural and configuration choices. When you are ready to conduct long-term testing and put compressed tables into production, see Section 15.9.1.4, "Monitoring InnoDB Table Compression at Runtime" for ways to verify the effectiveness of those choices under real-world conditions.

When to Use Compression

In general, compression works best on tables that include a reasonable number of character string columns and where the data is read far more often than it is written. Because there are no guaranteed ways to predict whether or not compression benefits a particular situation, always test with a specific workload and data set running on a representative configuration. Consider the following factors when deciding which tables to compress.

Data Characteristics and Compression

A key determinant of the efficiency of compression in reducing the size of data files is the nature of the data itself. Recall that compression works by identifying repeated strings of bytes in a block of data. Completely randomized data is the worst case. Typical data often has repeated values, and so compresses effectively. Character strings often compress well, whether defined in `CHAR`, `VARCHAR`, `TEXT` or `BLOB` columns. On the other hand, tables containing mostly binary data (integers or floating point numbers) or data that is previously compressed (for example JPEG or PNG images) may not generally compress well, significantly or at all.

You choose whether to turn on compression for each InnoDB table. A table and all of its indexes use the same (compressed) page size. It might be that the primary key (clustered) index, which contains the data for all columns of a table, compresses more effectively than the secondary indexes. For those cases where there are long rows, the use of compression might result in long column values being stored "off-page", as discussed in DYNAMIC Row Format. Those overflow pages may compress well. Given these considerations, for many applications, some tables compress more effectively than others, and you might find that your workload performs best only with a subset of tables compressed.

To determine whether or not to compress a particular table, conduct experiments. You can get a rough estimate of how efficiently your data can be compressed by using a utility that implements LZ77 compression (such as `gzip` or WinZip) on a copy of the .ibd file for an uncompressed table. You can expect less compression from a MySQL compressed table than from file-based compression tools, because MySQL compresses data in chunks based on the page size, 16KB by default. In addition to user data, the page format includes some internal system data that is not compressed. File-based compression utilities can examine much larger chunks of data, and so might find more repeated strings in a huge file than MySQL can find in an individual page.

Another way to test compression on a specific table is to copy some data from your uncompressed table to a similar, compressed table (having all the same indexes) in a file-per-table tablespace and look at the size of the resulting `.ibd` file. For example:

```
USE test;
SET GLOBAL innodb_file_per_table=1;
SET GLOBAL autocommit=0;

-- Create an uncompressed table with a million or two rows.
CREATE TABLE big_table AS SELECT * FROM information_schema.columns;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
COMMIT;
ALTER TABLE big_table ADD id int unsigned NOT NULL PRIMARY KEY auto_increment;

SHOW CREATE TABLE big_table\G

select count(id) from big_table;

-- Check how much space is needed for the uncompressed table.
\! ls -l data/test/big_table.ibd

CREATE TABLE key_block_size_4 LIKE big_table;
ALTER TABLE key_block_size_4 key_block_size=4 row_format=compressed;

INSERT INTO key_block_size_4 SELECT * FROM big_table;
commit;

-- Check how much space is needed for a compressed table
-- with particular compression settings.
\! ls -l data/test/key_block_size_4.ibd
```

This experiment produced the following numbers, which of course could vary considerably depending on your table structure and data:

```
-rw-rw----  1 cirrus  staff  310378496 Jan  9 13:44 data/test/big_table.ibd
-rw-rw----  1 cirrus  staff  83886080 Jan  9 15:10 data/test/key_block_size_4.ibd
```

To see whether compression is efficient for your particular workload:

- For simple tests, use a MySQL instance with no other compressed tables and run queries against the `INFORMATION_SCHEMA.INNODB_CMP` table.
- For more elaborate tests involving workloads with multiple compressed tables, run queries against the `INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX` table. Because the statistics in the `INNODB_CMP_PER_INDEX` table are expensive to collect, you must enable the configuration option `innodb_cmp_per_index_enabled` before querying that table, and you might restrict such testing to a development server or a non-critical slave server.
- Run some typical SQL statements against the compressed table you are testing.
- Examine the ratio of successful compression operations to overall compression operations by querying the `INFORMATION_SCHEMA.INNODB_CMP` or `INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX` table, and comparing `COMPRESS_OPS` to `COMPRESS_OPS_OK`.
- If a high percentage of compression operations complete successfully, the table might be a good candidate for compression.

- If you get a high proportion of compression failures, you can adjust `innodb_compression_level`, `innodb_compression_failure_threshold_pct`, and `innodb_compression_pad_pct_max` options as described in Section 15.9.1.6, “Compression for OLTP Workloads”, and try further tests.

Database Compression versus Application Compression

Decide whether to compress data in your application or in the table; do not use both types of compression for the same data. When you compress the data in the application and store the results in a compressed table, extra space savings are extremely unlikely, and the double compression just wastes CPU cycles.

Compressing in the Database

When enabled, MySQL table compression is automatic and applies to all columns and index values. The columns can still be tested with operators such as `LIKE`, and sort operations can still use indexes even when the index values are compressed. Because indexes are often a significant fraction of the total size of a database, compression could result in significant savings in storage, I/O or processor time. The compression and decompression operations happen on the database server, which likely is a powerful system that is sized to handle the expected load.

Compressing in the Application

If you compress data such as text in your application, before it is inserted into the database, You might save overhead for data that does not compress well by compressing some columns and not others. This approach uses CPU cycles for compression and uncompression on the client machine rather than the database server, which might be appropriate for a distributed application with many clients, or where the client machine has spare CPU cycles.

Hybrid Approach

Of course, it is possible to combine these approaches. For some applications, it may be appropriate to use some compressed tables and some uncompressed tables. It may be best to externally compress some data (and store it in uncompressed tables) and allow MySQL to compress (some of) the other tables in the application. As always, up-front design and real-life testing are valuable in reaching the right decision.

Workload Characteristics and Compression

In addition to choosing which tables to compress (and the page size), the workload is another key determinant of performance. If the application is dominated by reads, rather than updates, fewer pages need to be reorganized and recompressed after the index page runs out of room for the per-page “modification log” that MySQL maintains for compressed data. If the updates predominantly change non-indexed columns or those containing `BLOB`s or large strings that happen to be stored “off-page”, the overhead of compression may be acceptable. If the only changes to a table are `INSERT`s that use a monotonically increasing primary key, and there are few secondary indexes, there is little need to reorganize and recompress index pages. Since MySQL can “delete-mark” and delete rows on compressed pages “in place” by modifying uncompressed data, `DELETE` operations on a table are relatively efficient.

For some environments, the time it takes to load data can be as important as run-time retrieval. Especially in data warehouse environments, many tables may be read-only or read-mostly. In those cases, it might or might not be acceptable to pay the price of compression in terms of increased load time, unless the resulting savings in fewer disk reads or in storage cost is significant.

Fundamentally, compression works best when the CPU time is available for compressing and uncompressing data. Thus, if your workload is I/O bound, rather than CPU-bound, you might find that compression can improve overall performance. When you test your application performance with different compression configurations, test on a platform similar to the planned configuration of the production system.

Configuration Characteristics and Compression

Reading and writing database pages from and to disk is the slowest aspect of system performance. Compression attempts to reduce I/O by using CPU time to compress and uncompress data, and is most effective when I/O is a relatively scarce resource compared to processor cycles.

This is often especially the case when running in a multi-user environment with fast, multi-core CPUs. When a page of a compressed table is in memory, MySQL often uses additional memory, typically 16KB, in the buffer pool for an uncompressed copy of the page. The adaptive LRU algorithm attempts to balance the use of memory between compressed and uncompressed pages to take into account whether the workload is running in an I/O-bound or CPU-bound manner. Still, a configuration with more memory dedicated to the buffer pool tends to run better when using compressed tables than a configuration where memory is highly constrained.

Choosing the Compressed Page Size

The optimal setting of the compressed page size depends on the type and distribution of data that the table and its indexes contain. The compressed page size should always be bigger than the maximum record size, or operations may fail as noted in Compression of B-Tree Pages.

Setting the compressed page size too large wastes some space, but the pages do not have to be compressed as often. If the compressed page size is set too small, inserts or updates may require time-consuming recompression, and the B-tree nodes may have to be split more frequently, leading to bigger data files and less efficient indexing.

Typically, you set the compressed page size to 8K or 4K bytes. Given that the maximum row size for an InnoDB table is around 8K, `KEY_BLOCK_SIZE=8` is usually a safe choice.

15.9.1.4 Monitoring InnoDB Table Compression at Runtime

Overall application performance, CPU and I/O utilization and the size of disk files are good indicators of how effective compression is for your application. This section builds on the performance tuning advice from Section 15.9.1.3, “Tuning Compression for InnoDB Tables”, and shows how to find problems that might not turn up during initial testing.

To dig deeper into performance considerations for compressed tables, you can monitor compression performance at runtime using the Information Schema tables described in Example 15.1, “Using the Compression Information Schema Tables”. These tables reflect the internal use of memory and the rates of compression used overall.

The `INNODB_CMP` table reports information about compression activity for each compressed page size (`KEY_BLOCK_SIZE`) in use. The information in these tables is system-wide: it summarizes the compression statistics across all compressed tables in your database. You can use this data to help decide whether or not to compress a table by examining these tables when no other compressed tables are being accessed. It involves relatively low overhead on the server, so you might query it periodically on a production server to check the overall efficiency of the compression feature.

The `INNODB_CMP_PER_INDEX` table reports information about compression activity for individual tables and indexes. This information is more targeted and more useful for evaluating compression efficiency and diagnosing performance issues one table or index at a time. (Because that each `InnoDB` table is represented as a clustered index, MySQL does not make a big distinction between tables and indexes in this context.) The `INNODB_CMP_PER_INDEX` table does involve substantial overhead, so it is more suitable for development servers, where you can compare the effects of different workloads, data, and compression settings in isolation. To guard against imposing this monitoring overhead by accident, you must enable the `innodb_cmp_per_index_enabled` configuration option before you can query the `INNODB_CMP_PER_INDEX` table.

The key statistics to consider are the number of, and amount of time spent performing, compression and uncompression operations. Since MySQL splits B-tree nodes when they are too full to contain the compressed data following a modification, compare the number of "successful" compression operations with the number of such operations overall. Based on the information in the `INNODB_CMP` and `INNODB_CMP_PER_INDEX` tables and overall application performance and hardware resource utilization, you might make changes in your hardware configuration, adjust the size of the buffer pool, choose a different page size, or select a different set of tables to compress.

If the amount of CPU time required for compressing and uncompressing is high, changing to faster or multi-core CPUs can help improve performance with the same data, application workload and set of compressed tables. Increasing the size of the buffer pool might also help performance, so that more uncompressed pages can stay in memory, reducing the need to uncompress pages that exist in memory only in compressed form.

A large number of compression operations overall (compared to the number of `INSERT`, `UPDATE` and `DELETE` operations in your application and the size of the database) could indicate that some of your compressed tables are being updated too heavily for effective compression. If so, choose a larger page size, or be more selective about which tables you compress.

If the number of "successful" compression operations (`COMPRESS_OPS_OK`) is a high percentage of the total number of compression operations (`COMPRESS_OPS`), then the system is likely performing well. If the ratio is low, then MySQL is reorganizing, recompressing, and splitting B-tree nodes more often than is desirable. In this case, avoid compressing some tables, or increase `KEY_BLOCK_SIZE` for some of the compressed tables. You might turn off compression for tables that cause the number of "compression failures" in your application to be more than 1% or 2% of the total. (Such a failure ratio might be acceptable during a temporary operation such as a data load).

15.9.1.5 How Compression Works for InnoDB Tables

This section describes some internal implementation details about compression for InnoDB tables. The information presented here may be helpful in tuning for performance, but is not necessary to know for basic use of compression.

Compression Algorithms

Some operating systems implement compression at the file system level. Files are typically divided into fixed-size blocks that are compressed into variable-size blocks, which easily leads into fragmentation. Every time something inside a block is modified, the whole block is recompressed before it is written to disk. These properties make this compression technique unsuitable for use in an update-intensive database system.

MySQL implements compression with the help of the well-known zlib library, which implements the LZ77 compression algorithm. This compression algorithm is mature, robust, and efficient in both CPU utilization and in reduction of data size. The algorithm is "lossless", so that the original uncompressed data can always be reconstructed from the compressed form. LZ77 compression works by finding sequences of data that are repeated within the data to be compressed. The patterns of values in your data determine how well it compresses, but typical user data often compresses by 50% or more.

Note

`InnoDB` supports the `zlib` library up to version 1.2.11, which is the version bundled with MySQL 8.0.

Unlike compression performed by an application, or compression features of some other database management systems, InnoDB compression applies both to user data and to indexes. In many cases,

indexes can constitute 40-50% or more of the total database size, so this difference is significant. When compression is working well for a data set, the size of the InnoDB data files (the file-per-table tablespace or general tablespace `.ibd` files) is 25% to 50% of the uncompressed size or possibly smaller. Depending on the workload, this smaller database can in turn lead to a reduction in I/O, and an increase in throughput, at a modest cost in terms of increased CPU utilization. You can adjust the balance between compression level and CPU overhead by modifying the `innodb_compression_level` configuration option.

InnoDB Data Storage and Compression

All user data in InnoDB tables is stored in pages comprising a B-tree index (the clustered index). In some other database systems, this type of index is called an “index-organized table”. Each row in the index node contains the values of the (user-specified or system-generated) primary key and all the other columns of the table.

Secondary indexes in InnoDB tables are also B-trees, containing pairs of values: the index key and a pointer to a row in the clustered index. The pointer is in fact the value of the primary key of the table, which is used to access the clustered index if columns other than the index key and primary key are required. Secondary index records must always fit on a single B-tree page.

The compression of B-tree nodes (of both clustered and secondary indexes) is handled differently from compression of overflow pages used to store long `VARCHAR`, `BLOB`, or `TEXT` columns, as explained in the following sections.

Compression of B-Tree Pages

Because they are frequently updated, B-tree pages require special treatment. It is important to minimize the number of times B-tree nodes are split, as well as to minimize the need to uncompress and recompress their content.

One technique MySQL uses is to maintain some system information in the B-tree node in uncompressed form, thus facilitating certain in-place updates. For example, this allows rows to be delete-marked and deleted without any compression operation.

In addition, MySQL attempts to avoid unnecessary uncompression and recompression of index pages when they are changed. Within each B-tree page, the system keeps an uncompressed “modification log” to record changes made to the page. Updates and inserts of small records may be written to this modification log without requiring the entire page to be completely reconstructed.

When the space for the modification log runs out, InnoDB uncompresses the page, applies the changes and recompresses the page. If recompression fails (a situation known as a compression failure), the B-tree nodes are split and the process is repeated until the update or insert succeeds.

To avoid frequent compression failures in write-intensive workloads, such as for OLTP applications, MySQL sometimes reserves some empty space (padding) in the page, so that the modification log fills up sooner and the page is recompressed while there is still enough room to avoid splitting it. The amount of padding space left in each page varies as the system keeps track of the frequency of page splits. On a busy server doing frequent writes to compressed tables, you can adjust the `innodb_compression_failure_threshold_pct`, and `innodb_compression_pad_pct_max` configuration options to fine-tune this mechanism.

Generally, MySQL requires that each B-tree page in an InnoDB table can accommodate at least two records. For compressed tables, this requirement has been relaxed. Leaf pages of B-tree nodes (whether of the primary key or secondary indexes) only need to accommodate one record, but that record must fit, in uncompressed form, in the per-page modification log. If `innodb_strict_mode` is `ON`, MySQL checks the maximum row size during `CREATE TABLE` or `CREATE INDEX`. If the row does not fit, the following error message is issued: `ERROR HY000: Too big row`.

If you create a table when `innodb_strict_mode` is OFF, and a subsequent `INSERT` or `UPDATE` statement attempts to create an index entry that does not fit in the size of the compressed page, the operation fails with `ERROR 42000: Row size too large`. (This error message does not name the index for which the record is too large, or mention the length of the index record or the maximum record size on that particular index page.) To solve this problem, rebuild the table with `ALTER TABLE` and select a larger compressed page size (`KEY_BLOCK_SIZE`), shorten any column prefix indexes, or disable compression entirely with `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPACT`.

`innodb_strict_mode` is not applicable to general tablespaces, which also support compressed tables. Tablespace management rules for general tablespaces are strictly enforced independently of `innodb_strict_mode`. For more information, see Section 13.1.21, “CREATE TABLESPACE Syntax”.

Compressing BLOB, VARCHAR, and TEXT Columns

In an InnoDB table, `BLOB`, `VARCHAR`, and `TEXT` columns that are not part of the primary key may be stored on separately allocated overflow pages. We refer to these columns as off-page columns. Their values are stored on singly-linked lists of overflow pages.

For tables created in `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPRESSED`, the values of `BLOB`, `TEXT`, or `VARCHAR` columns may be stored fully off-page, depending on their length and the length of the entire row. For columns that are stored off-page, the clustered index record only contains 20-byte pointers to the overflow pages, one per column. Whether any columns are stored off-page depends on the page size and the total size of the row. When the row is too long to fit entirely within the page of the clustered index, MySQL chooses the longest columns for off-page storage until the row fits on the clustered index page. As noted above, if a row does not fit by itself on a compressed page, an error occurs.

Note

For tables created in `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPRESSED`, `TEXT` and `BLOB` columns that are less than or equal to 40 bytes are always stored in-line.

Tables that use `ROW_FORMAT=REDUNDANT` and `ROW_FORMAT=COMPACT` store the first 768 bytes of `BLOB`, `VARCHAR`, and `TEXT` columns in the clustered index record along with the primary key. The 768-byte prefix is followed by a 20-byte pointer to the overflow pages that contain the rest of the column value.

When a table is in `COMPRESSED` format, all data written to overflow pages is compressed “as is”; that is, MySQL applies the zlib compression algorithm to the entire data item. Other than the data, compressed overflow pages contain an uncompressed header and trailer comprising a page checksum and a link to the next overflow page, among other things. Therefore, very significant storage savings can be obtained for longer `BLOB`, `TEXT`, or `VARCHAR` columns if the data is highly compressible, as is often the case with text data. Image data, such as `JPEG`, is typically already compressed and so does not benefit much from being stored in a compressed table; the double compression can waste CPU cycles for little or no space savings.

The overflow pages are of the same size as other pages. A row containing ten columns stored off-page occupies ten overflow pages, even if the total length of the columns is only 8K bytes. In an uncompressed table, ten uncompressed overflow pages occupy 160K bytes. In a compressed table with an 8K page size, they occupy only 80K bytes. Thus, it is often more efficient to use compressed table format for tables with long column values.

For file-per-table tablespaces, using a 16K compressed page size can reduce storage and I/O costs for `BLOB`, `VARCHAR`, or `TEXT` columns, because such data often compress well, and might therefore require fewer overflow pages, even though the B-tree nodes themselves take as many pages as in the uncompressed form. General tablespaces do not support a 16K compressed page size (`KEY_BLOCK_SIZE`). For more information, see Section 15.6.3.3, “General Tablespaces”.

Compression and the InnoDB Buffer Pool

In a compressed InnoDB table, every compressed page (whether 1K, 2K, 4K or 8K) corresponds to an uncompressed page of 16K bytes (or a smaller size if innodb_page_size is set). To access the data in a page, MySQL reads the compressed page from disk if it is not already in the buffer pool, then uncompresses the page to its original form. This section describes how InnoDB manages the buffer pool with respect to pages of compressed tables.

To minimize I/O and to reduce the need to uncompress a page, at times the buffer pool contains both the compressed and uncompressed form of a database page. To make room for other required database pages, MySQL can evict from the buffer pool an uncompressed page, while leaving the compressed page in memory. Or, if a page has not been accessed in a while, the compressed form of the page might be written to disk, to free space for other data. Thus, at any given time, the buffer pool might contain both the compressed and uncompressed forms of the page, or only the compressed form of the page, or neither.

MySQL keeps track of which pages to keep in memory and which to evict using a least-recently-used (LRU) list, so that hot (frequently accessed) data tends to stay in memory. When compressed tables are accessed, MySQL uses an adaptive LRU algorithm to achieve an appropriate balance of compressed and uncompressed pages in memory. This adaptive algorithm is sensitive to whether the system is running in an I/O-bound or CPU-bound manner. The goal is to avoid spending too much processing time uncompressing pages when the CPU is busy, and to avoid doing excess I/O when the CPU has spare cycles that can be used for uncompressing compressed pages (that may already be in memory). When the system is I/O-bound, the algorithm prefers to evict the uncompressed copy of a page rather than both copies, to make more room for other disk pages to become memory resident. When the system is CPU-bound, MySQL prefers to evict both the compressed and uncompressed page, so that more memory can be used for “hot” pages and reducing the need to uncompress data in memory only in compressed form.

Compression and the InnoDB Redo Log Files

Before a compressed page is written to a data file, MySQL writes a copy of the page to the redo log (if it has been recompressed since the last time it was written to the database). This is done to ensure that redo logs are usable for crash recovery, even in the unlikely case that the zlib library is upgraded and that change introduces a compatibility problem with the compressed data. Therefore, some increase in the size of log files, or a need for more frequent checkpoints, can be expected when using compression. The amount of increase in the log file size or checkpoint frequency depends on the number of times compressed pages are modified in a way that requires reorganization and recompression.

To create a compressed table in a file-per-table tablespace, innodb_file_per_table must be enabled. There is no dependence on the innodb_file_per_table setting when creating a compressed table in a general tablespace. For more information, see Section 15.6.3.3, “General Tablespaces”.

15.9.1.6 Compression for OLTP Workloads

Traditionally, the InnoDB compression feature was recommended primarily for read-only or read-mostly workloads, such as in a data warehouse configuration. The rise of SSD storage devices, which are fast but relatively small and expensive, makes compression attractive also for OLTP workloads: high-traffic, interactive websites can reduce their storage requirements and their I/O operations per second (IOPS) by using compressed tables with applications that do frequent INSERT, UPDATE, and DELETE operations.

These configuration options let you adjust the way compression works for a particular MySQL instance, with an emphasis on performance and scalability for write-intensive operations:

- innodb_compression_level lets you turn the degree of compression up or down. A higher value lets you fit more data onto a storage device, at the expense of more CPU overhead during compression. A lower value lets you reduce CPU overhead when storage space is not critical, or you expect the data is not especially compressible.
- innodb_compression_failure_threshold_pct specifies a cutoff point for compression failures during updates to a compressed table. When this threshold is passed, MySQL begins to leave additional

free space within each new compressed page, dynamically adjusting the amount of free space up to the percentage of page size specified by `innodb_compression_pad_pct_max`

- `innodb_compression_pad_pct_max` lets you adjust the maximum amount of space reserved within each page to record changes to compressed rows, without needing to compress the entire page again. The higher the value, the more changes can be recorded without recompressing the page. MySQL uses a variable amount of free space for the pages within each compressed table, only when a designated percentage of compression operations "fail" at runtime, requiring an expensive operation to split the compressed page.
- `innodb_log_compressed_pages` lets you disable writing of images of re-compressed pages to the redo log. Re-compression may occur when changes are made to compressed data. This option is enabled by default to prevent corruption that could occur if a different version of the `zlib` compression algorithm is used during recovery. If you are certain that the `zlib` version will not change, disable `innodb_log_compressed_pages` to reduce redo log generation for workloads that modify compressed data.

Because working with compressed data sometimes involves keeping both compressed and uncompressed versions of a page in memory at the same time, when using compression with an OLTP-style workload, be prepared to increase the value of the `innodb_buffer_pool_size` configuration option.

15.9.1.7 SQL Compression Syntax Warnings and Errors

This section describes syntax warnings and errors that you may encounter when using the table compression feature with file-per-table tablespaces and general tablespaces.

SQL Compression Syntax Warnings and Errors for File-Per-Table Tablespaces

When `innodb_strict_mode` is enabled (the default), specifying `ROW_FORMAT=COMPRESSED` or `KEY_BLOCK_SIZE` in `CREATE TABLE` or `ALTER TABLE` statements produces the following error if `innodb_file_per_table` is disabled.

```
ERROR 1031 (HY000): Table storage engine for 't1' doesn't have this option
```

Note

The table is not created if the current configuration does not permit using compressed tables.

When `innodb_strict_mode` is disabled, specifying `ROW_FORMAT=COMPRESSED` or `KEY_BLOCK_SIZE` in `CREATE TABLE` or `ALTER TABLE` statements produces the following warnings if `innodb_file_per_table` is disabled.

```
mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------------------------------+
| Level   | Code | Message                                                         |
+---------+------+-----------------------------------------------------------------+
| Warning | 1478 | InnoDB: KEY_BLOCK_SIZE requires innodb_file_per_table.          |
| Warning | 1478 | InnoDB: ignoring KEY_BLOCK_SIZE=4.                              |
| Warning | 1478 | InnoDB: ROW_FORMAT=COMPRESSED requires innodb_file_per_table. |
| Warning | 1478 | InnoDB: assuming ROW_FORMAT=DYNAMIC.                            |
+---------+------+-----------------------------------------------------------------+
```

Note

These messages are only warnings, not errors, and the table is created without compression, as if the options were not specified.

The “non-strict” behavior lets you import a `mysqldump` file into a database that does not support compressed tables, even if the source database contained compressed tables. In that case, MySQL creates the table in `ROW_FORMAT=DYNAMIC` instead of preventing the operation.

To import the dump file into a new database, and have the tables re-created as they exist in the original database, ensure the server has the proper setting for the `innodb_file_per_table` configuration parameter.

The attribute `KEY_BLOCK_SIZE` is permitted only when `ROW_FORMAT` is specified as `COMPRESSED` or is omitted. Specifying a `KEY_BLOCK_SIZE` with any other `ROW_FORMAT` generates a warning that you can view with `SHOW WARNINGS`. However, the table is non-compressed; the specified `KEY_BLOCK_SIZE` is ignored).

Level	Code	Message
Warning	1478	`InnoDB: ignoring KEY_BLOCK_SIZE=n unless ROW_FORMAT=COMPRESSED.`

If you are running with `innodb_strict_mode` enabled, the combination of a `KEY_BLOCK_SIZE` with any `ROW_FORMAT` other than `COMPRESSED` generates an error, not a warning, and the table is not created.

Table 15.13, “ROW_FORMAT and KEY_BLOCK_SIZE Options” provides an overview the `ROW_FORMAT` and `KEY_BLOCK_SIZE` options that are used with `CREATE TABLE` or `ALTER TABLE`.

Table 15.13 ROW_FORMAT and KEY_BLOCK_SIZE Options

Option	Usage Notes	Description
`ROW_FORMAT=REDUNDANT`	Storage format used prior to MySQL 5.0.3	Less efficient than `ROW_FORMAT=COMPACT`; for backward compatibility
`ROW_FORMAT=COMPACT`	Default storage format since MySQL 5.0.3	Stores a prefix of 768 bytes of long column values in the clustered index page, with the remaining bytes stored in an overflow page
`ROW_FORMAT=DYNAMIC`		Store values within the clustered index page if they fit; if not, stores only a 20-byte pointer to an overflow page (no prefix)
`ROW_FORMAT=COMPRESSED`		Compresses the table and indexes using zlib
`KEY_BLOCK_SIZE=n`		Specifies compressed page size of 1, 2, 4, 8 or 16 kilobytes; implies `ROW_FORMAT=COMPRESSED`. For general tablespaces, a `KEY_BLOCK_SIZE` value equal to the `InnoDB` page size is not permitted.

Table 15.14, “CREATE/ALTER TABLE Warnings and Errors when InnoDB Strict Mode is OFF” summarizes error conditions that occur with certain combinations of configuration parameters and options on the `CREATE TABLE` or `ALTER TABLE` statements, and how the options appear in the output of `SHOW TABLE STATUS`.

When `innodb_strict_mode` is `OFF`, MySQL creates or alters the table, but ignores certain settings as shown below. You can see the warning messages in the MySQL error log. When `innodb_strict_mode`

is ON, these specified combinations of options generate errors, and the table is not created or altered. To see the full description of the error condition, issue the SHOW ERRORS statement: example:

```
mysql> CREATE TABLE x (id INT PRIMARY KEY, c INT)

-> ENGINE=INNODB KEY_BLOCK_SIZE=33333;

ERROR 1005 (HY000): Can't create table 'test.x' (errno: 1478)

mysql> SHOW ERRORS;
+-------+------+-----------------------------------------+
| Level | Code | Message                                 |
+-------+------+-----------------------------------------+
| Error | 1478 | InnoDB: invalid KEY_BLOCK_SIZE=33333.   |
| Error | 1005 | Can't create table 'test.x' (errno: 1478) |
+-------+------+-----------------------------------------+
```

Table 15.14 CREATE/ALTER TABLE Warnings and Errors when InnoDB Strict Mode is OFF

Syntax	Warning or Error Condition	Resulting ROW_FORMAT, as shown in SHOW TABLE STATUS
ROW_FORMAT=REDUNDANT	None	REDUNDANT
ROW_FORMAT=COMPACT	None	COMPACT
ROW_FORMAT=COMPRESSED or ROW_FORMAT=DYNAMIC or KEY_BLOCK_SIZE is specified	Ignored for file-per-table tablespaces unless innodb_file_per_table is enabled. General tablespaces support all row formats. See Section 15.6.3.3, "General Tablespaces".	the default row format for file-per-table tablespaces; the specified row format for general tablespaces
Invalid KEY_BLOCK_SIZE is specified (not 1, 2, 4, 8 or 16)	KEY_BLOCK_SIZE is ignored	the specified row format, or the default row format
ROW_FORMAT=COMPRESSED and valid KEY_BLOCK_SIZE are specified	None; KEY_BLOCK_SIZE specified is used	COMPRESSED
KEY_BLOCK_SIZE is specified with REDUNDANT, COMPACT or DYNAMIC row format	KEY_BLOCK_SIZE is ignored	REDUNDANT, COMPACT or DYNAMIC
ROW_FORMAT is not one of REDUNDANT, COMPACT, DYNAMIC or COMPRESSED	Ignored if recognized by the MySQL parser. Otherwise, an error is issued.	the default row format or N/A

When innodb_strict_mode is ON, MySQL rejects invalid ROW_FORMAT or KEY_BLOCK_SIZE parameters and issues errors. Strict mode is ON by default. When innodb_strict_mode is OFF, MySQL issues warnings instead of errors for ignored invalid parameters.

It is not possible to see the chosen KEY_BLOCK_SIZE using SHOW TABLE STATUS. The statement SHOW CREATE TABLE displays the KEY_BLOCK_SIZE (even if it was ignored when creating the table). The real compressed page size of the table cannot be displayed by MySQL.

SQL Compression Syntax Warnings and Errors for General Tablespaces

- If FILE_BLOCK_SIZE was not defined for the general tablespace when the tablespace was created, the tablespace cannot contain compressed tables. If you attempt to add a compressed table, an error is returned, as shown in the following example:

```
mysql> CREATE TABLESPACE `ts1` ADD DATAFILE 'ts1.ibd' Engine=InnoDB;

mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) TABLESPACE ts1 ROW_FORMAT=COMPRESSED
       KEY_BLOCK_SIZE=8;
ERROR 1478 (HY000): InnoDB: Tablespace `ts1` cannot contain a COMPRESSED table
```

- Attempting to add a table with an invalid `KEY_BLOCK_SIZE` to a general tablespace returns an error, as shown in the following example:

```
mysql> CREATE TABLESPACE `ts2` ADD DATAFILE 'ts2.ibd' FILE_BLOCK_SIZE = 8192 Engine=InnoDB;

mysql> CREATE TABLE t2 (c1 INT PRIMARY KEY) TABLESPACE ts2 ROW_FORMAT=COMPRESSED
       KEY_BLOCK_SIZE=4;
ERROR 1478 (HY000): InnoDB: Tablespace `ts2` uses block size 8192 and cannot
contain a table with physical page size 4096
```

  For general tablespaces, the `KEY_BLOCK_SIZE` of the table must be equal to the `FILE_BLOCK_SIZE` of the tablespace divided by 1024. For example, if the `FILE_BLOCK_SIZE` of the tablespace is 8192, the `KEY_BLOCK_SIZE` of the table must be 8.

- Attempting to add a table with an uncompressed row format to a general tablespace configured to store compressed tables returns an error, as shown in the following example:

```
mysql> CREATE TABLESPACE `ts3` ADD DATAFILE 'ts3.ibd' FILE_BLOCK_SIZE = 8192 Engine=InnoDB;

mysql> CREATE TABLE t3 (c1 INT PRIMARY KEY) TABLESPACE ts3 ROW_FORMAT=COMPACT;
ERROR 1478 (HY000): InnoDB: Tablespace `ts3` uses block size 8192 and cannot
contain a table with physical page size 16384
```

`innodb_strict_mode` is not applicable to general tablespaces. Tablespace management rules for general tablespaces are strictly enforced independently of `innodb_strict_mode`. For more information, see Section 13.1.21, “CREATE TABLESPACE Syntax”.

For more information about using compressed tables with general tablespaces, see Section 15.6.3.3, “General Tablespaces”.

15.9.2 InnoDB Page Compression

`InnoDB` supports page-level compression for tables that reside in file-per-table tablespaces. This feature is referred to as *Transparent Page Compression*. Page compression is enabled by specifying the `COMPRESSION` attribute with `CREATE TABLE` or `ALTER TABLE`. Supported compression algorithms include `Zlib` and `LZ4`.

Supported Platforms

Page compression requires sparse file and hole punching support. Page compression is supported on Windows with NTFS, and on the following subset of MySQL-supported Linux platforms where the kernel level provides hole punching support:

- RHEL 7 and derived distributions that use kernel version 3.10.0-123 or higher
- OEL 5.10 (UEK2) kernel version 2.6.39 or higher
- OEL 6.5 (UEK3) kernel version 3.8.13 or higher
- OEL 7.0 kernel version 3.8.13 or higher

- SLE11 kernel version 3.0-x
- SLE12 kernel version 3.12-x
- OES11 kernel version 3.0-x
- Ubuntu 14.0.4 LTS kernel version 3.13 or higher
- Ubuntu 12.0.4 LTS kernel version 3.2 or higher
- Debian 7 kernel version 3.2 or higher

Note

All of the available file systems for a given Linux distribution may not support hole punching.

How Page Compression Works

When a page is written, it is compressed using the specified compression algorithm. The compressed data is written to disk, where the hole punching mechanism releases empty blocks from the end of the page. If compression fails, data is written out as-is.

Hole Punch Size on Linux

On Linux systems, the file system block size is the unit size used for hole punching. Therefore, page compression only works if page data can be compressed to a size that is less than or equal to the `InnoDB` page size minus the file system block size. For example, if `innodb_page_size=16K` and the file system block size is 4K, page data must compress to less than or equal to 12K to make hole punching possible.

Hole Punch Size on Windows

On Windows systems, the underlying infrastructure for sparse files is based on NTFS compression. Hole punching size is the NTFS compression unit, which is 16 times the NTFS cluster size. Cluster sizes and their compression units are shown in the following table:

Table 15.15 Windows NTFS Cluster Size and Compression Units

Cluster Size	Compression Unit
512 Bytes	8 KB
1 KB	16 KB
2 KB	32 KB
4 KB	64 KB

Page compression on Windows systems only works if page data can be compressed to a size that is less than or equal to the `InnoDB` page size minus the compression unit size.

The default NTFS cluster size is 4KB, for which the compression unit size is 64KB. This means that page compression has no benefit for an out-of-the box Windows NTFS configuration, as the maximum `innodb_page_size` is also 64KB.

For page compression to work on Windows, the file system must be created with a cluster size smaller than 4K, and the `innodb_page_size` must be at least twice the size of the compression unit. For example, for page compression to work on Windows, you could build the file system with a cluster size of 512 Bytes (which has a compression unit of 8KB) and initialize `InnoDB` with an `innodb_page_size` value of 16K or greater.

Enabling Page Compression

To enable page compression, specify the `COMPRESSION` attribute in the `CREATE TABLE` statement. For example:

```
CREATE TABLE t1 (c1 INT) COMPRESSION="zlib";
```

You can also enable page compression in an `ALTER TABLE` statement. However, `ALTER TABLE ... COMPRESSION` only updates the tablespace compression attribute. Writes to the tablespace that occur after setting the new compression algorithm use the new setting, but to apply the new compression algorithm to existing pages, you must rebuild the table using `OPTIMIZE TABLE`.

```
ALTER TABLE t1 COMPRESSION="zlib";
OPTIMIZE TABLE t1;
```

Disabling Page Compression

To disable page compression, set `COMPRESSION=None` using `ALTER TABLE`. Writes to the tablespace that occur after setting `COMPRESSION=None` no longer use page compression. To uncompress existing pages, you must rebuild the table using `OPTIMIZE TABLE` after setting `COMPRESSION=None`.

```
ALTER TABLE t1 COMPRESSION="None";
OPTIMIZE TABLE t1;
```

Page Compression Metadata

Page compression metadata is found in the `INFORMATION_SCHEMA.INNODB_TABLESPACES` table, in the following columns:

- `FS_BLOCK_SIZE`: The file system block size, which is the unit size used for hole punching.
- `FILE_SIZE`: The apparent size of the file, which represents the maximum size of the file, uncompressed.
- `ALLOCATED_SIZE`: The actual size of the file, which is the amount of space allocated on disk.

Note

On Unix-like systems, `ls -l` *`tablespace_name.ibd`* shows the apparent file size (equivalent to `FILE_SIZE`) in bytes. To view the actual amount of space allocated on disk (equivalent to `ALLOCATED_SIZE`), use `du --block-size=1` *`tablespace_name.ibd`*. The `--block-size=1` option prints the allocated space in bytes instead of blocks, so that it can be compared to `ls -l` output.

Use `SHOW CREATE TABLE` to view the current page compression setting (`Zlib`, `Lz4`, or `None`). A table may contain a mix of pages with different compression settings.

In the following example, page compression metadata for the employees table is retrieved from the `INFORMATION_SCHEMA.INNODB_TABLESPACES` table.

```
# Create the employees table with Zlib page compression

CREATE TABLE employees (
    emp_no      INT             NOT NULL,
```

```
    birth_date  DATE            NOT NULL,
    first_name  VARCHAR(14)     NOT NULL,
    last_name   VARCHAR(16)     NOT NULL,
    gender      ENUM ('M','F')  NOT NULL,
    hire_date   DATE            NOT NULL,
    PRIMARY KEY (emp_no)
) COMPRESSION="zlib";

# Insert data (not shown)

# Query page compression metadata in INFORMATION_SCHEMA.INNODB_TABLESPACES

mysql> SELECT SPACE, NAME, FS_BLOCK_SIZE, FILE_SIZE, ALLOCATED_SIZE FROM
       INFORMATION_SCHEMA.INNODB_TABLESPACES WHERE NAME='employees/employees'\G
*************************** 1. row ***************************
SPACE: 45
NAME: employees/employees
FS_BLOCK_SIZE: 4096
FILE_SIZE: 23068672
ALLOCATED_SIZE: 19415040
```

Page compression metadata for the employees table shows that the apparent file size is 23068672 bytes while the actual file size (with page compression) is 19415040 bytes. The file system block size is 4096 bytes, which is the block size used for hole punching.

Identifying Tables Using Page Compression

To identify tables for which page compression is enabled, you can query the `INFORMATION_SCHEMA.TABLES CREATE_OPTIONS` column for tables defined with the `COMPRESSION` attribute:

```
mysql> SELECT TABLE_NAME, TABLE_SCHEMA, CREATE_OPTIONS FROM INFORMATION_SCHEMA.TABLES
       WHERE CREATE_OPTIONS LIKE '%COMPRESSION=%';
+------------+--------------+--------------------+
| TABLE_NAME | TABLE_SCHEMA | CREATE_OPTIONS     |
+------------+--------------+--------------------+
| employees  | test         | COMPRESSION="zlib" |
+------------+--------------+--------------------+
```

`SHOW CREATE TABLE` also shows the `COMPRESSION` attribute, if used.

Page Compression Limitations and Usage Notes

- Page compression is disabled if the file system block size (or compression unit size on Windows) * 2 > `innodb_page_size`.
- Page compression is not supported for tables that reside in shared tablespaces, which include the system tablespace, temporary tablespaces, and general tablespaces.
- Page compression is not supported for undo log tablespaces.
- Page compression is not supported for redo log pages.
- R-tree pages, which are used for spatial indexes, are not compressed.
- Pages that belong to compressed tables (`ROW_FORMAT=COMPRESSED`) are left as-is.
- During recovery, updated pages are written out in an uncompressed form.
- Loading a page-compressed tablespace on a server that does not support the compression algorithm that was used causes an I/O error.

- Before downgrading to an earlier version of MySQL that does not support page compression, uncompress the tables that use the page compression feature. To uncompress a table, run `ALTER TABLE ... COMPRESSION=None` and `OPTIMIZE TABLE`.
- Page-compressed tablespaces can be copied between Linux and Windows servers if the compression algorithm that was used is available on both servers.
- Preserving page compression when moving a page-compressed tablespace file from one host to another requires a utility that preserves sparse files.
- Better page compression may be achieved on Fusion-io hardware with NVMFS than on other platforms, as NVMFS is designed to take advantage of punch hole functionality.
- Using the page compression feature with a large `InnoDB` page size and relatively small file system block size could result in write amplification. For example, a maximum `InnoDB` page size of 64KB with a 4KB file system block size may improve compression but may also increase demand on the buffer pool, leading to increased I/O and potential write amplification.

15.10 InnoDB Row Formats

The row format of a table determines how its rows are physically stored, which in turn can affect the performance of queries and DML operations. As more rows fit into a single disk page, queries and index lookups can work faster, less cache memory is required in the buffer pool, and less I/O is required to write out updated values.

The data in each table is divided into pages. The pages that make up each table are arranged in a tree data structure called a B-tree index. Table data and secondary indexes both use this type of structure. The B-tree index that represents an entire table is known as the clustered index, which is organized according to the primary key columns. The nodes of a clustered index data structure contain the values of all columns in the row. The nodes of a secondary index structure contain the values of index columns and primary key columns.

Variable-length columns are an exception to the rule that column values are stored in B-tree index nodes. Variable-length columns that are too long to fit on a B-tree page are stored on separately allocated disk pages called overflow pages. Such columns are referred to as off-page columns. The values of off-page columns are stored in singly-linked lists of overflow pages, with each such column having its own list of one or more overflow pages. Depending on column length, all or a prefix of variable-length column values are stored in the B-tree to avoid wasting storage and having to read a separate page.

The `InnoDB` storage engine supports four row formats: `REDUNDANT`, `COMPACT`, `DYNAMIC`, and `COMPRESSED`.

Table 15.16 InnoDB Row Format Overview

Row Format	Compact Storage Characteristics	Enhanced Variable-Length Column Storage	Large Index Key Prefix Support	Compression Support	Supported Tablespace Types
`REDUNDANT`	No	No	No	No	system, file-per-table, general
`COMPACT`	Yes	No	No	No	system, file-per-table, general
`DYNAMIC`	Yes	Yes	Yes	No	system, file-per-table, general
`COMPRESSED`	Yes	Yes	Yes	Yes	file-per-table, general

The topics that follow describe row format storage characteristics and how to define and determine the row format of a table.

- REDUNDANT Row Format
- COMPACT Row Format
- DYNAMIC Row Format
- COMPRESSED Row Format
- Defining the Row Format of a Table
- Determining the Row Format of a Table

REDUNDANT Row Format

The `REDUNDANT` format provides compatibility with older versions of MySQL.

Tables that use the `REDUNDANT` row format store the first 768 bytes of variable-length column values (`VARCHAR`, `VARBINARY`, and `BLOB` and `TEXT` types) in the index record within the B-tree node, with the remainder stored on overflow pages. Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length columns, which can be stored off-page. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.

If the value of a column is 768 bytes or less, an overflow page is not used, and some savings in I/O may result, since the value is stored entirely in the B-tree node. This works well for relatively short `BLOB` column values, but may cause B-tree nodes to fill with data rather than key values, reducing their efficiency. Tables with many `BLOB` columns could cause B-tree nodes to become too full, and contain too few rows, making the entire index less efficient than if rows were shorter or column values were stored off-page.

REDUNDANT Row Format Storage Characteristics

The `REDUNDANT` row format has the following storage characteristics:

- Each index record contains a 6-byte header. The header is used to link together consecutive records, and for row-level locking.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte transaction ID field and a 7-byte roll pointer field.
- If no primary key is defined for a table, each clustered index record also contains a 6-byte row ID field.
- Each secondary index record contains all the primary key columns defined for the clustered index key that are not in the secondary index.
- A record contains a pointer to each field of the record. If the total length of the fields in a record is less than 128 bytes, the pointer is one byte; otherwise, two bytes. The array of pointers is called the record directory. The area where the pointers point is the data part of the record.
- Internally, fixed-length character columns such as `CHAR(10)` in stored in fixed-length format. Trailing spaces are not truncated from `VARCHAR` columns.
- Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length columns, which can be stored off-page. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.
- An SQL `NULL` value reserves one or two bytes in the record directory. An SQL `NULL` value reserves zero bytes in the data part of the record if stored in a variable-length column. For a fixed-length column,

the fixed length of the column is reserved in the data part of the record. Reserving fixed space for `NULL` values permits columns to be updated in place from `NULL` to non-`NULL` values without causing index page fragmentation.

COMPACT Row Format

The `COMPACT` row format reduces row storage space by about 20% compared to the `REDUNDANT` row format, at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed, `COMPACT` format is likely to be faster. If the workload is limited by CPU speed, compact format might be slower.

Tables that use the `COMPACT` row format store the first 768 bytes of variable-length column values (`VARCHAR`, `VARBINARY`, and `BLOB` and `TEXT` types) in the index record within the B-tree node, with the remainder stored on overflow pages. Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length columns, which can be stored off-page. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.

If the value of a column is 768 bytes or less, an overflow page is not used, and some savings in I/O may result, since the value is stored entirely in the B-tree node. This works well for relatively short `BLOB` column values, but may cause B-tree nodes to fill with data rather than key values, reducing their efficiency. Tables with many `BLOB` columns could cause B-tree nodes to become too full, and contain too few rows, making the entire index less efficient than if rows were shorter or column values were stored off-page.

COMPACT Row Format Storage Characteristics

The `COMPACT` row format has the following storage characteristics:

- Each index record contains a 5-byte header that may be preceded by a variable-length header. The header is used to link together consecutive records, and for row-level locking.
- The variable-length part of the record header contains a bit vector for indicating `NULL` columns. If the number of columns in the index that can be `NULL` is *`N`*, the bit vector occupies `CEILING(`*`N`*`/8)` bytes. (For example, if there are anywhere from 9 to 16 columns that can be `NULL`, the bit vector uses two bytes.) Columns that are `NULL` do not occupy space other than the bit in this vector. The variable-length part of the header also contains the lengths of variable-length columns. Each length takes one or two bytes, depending on the maximum length of the column. If all columns in the index are `NOT NULL` and have a fixed length, the record header has no variable-length part.
- For each non-`NULL` variable-length field, the record header contains the length of the column in one or two bytes. Two bytes are only needed if part of the column is stored externally in overflow pages or the maximum length exceeds 255 bytes and the actual length exceeds 127 bytes. For an externally stored column, the 2-byte length indicates the length of the internally stored part plus the 20-byte pointer to the externally stored part. The internal part is 768 bytes, so the length is 768+20. The 20-byte pointer stores the true length of the column.
- The record header is followed by the data contents of non-`NULL` columns.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte transaction ID field and a 7-byte roll pointer field.
- If no primary key is defined for a table, each clustered index record also contains a 6-byte row ID field.
- Each secondary index record contains all the primary key columns defined for the clustered index key that are not in the secondary index. If any of the primary key columns are variable length, the record header for each secondary index has a variable-length part to record their lengths, even if the secondary index is defined on fixed-length columns.

- Internally, for nonvariable-length character sets, fixed-length character columns such as `CHAR(10)` are stored in a fixed-length format.

  Trailing spaces are not truncated from `VARCHAR` columns.

- Internally, for variable-length character sets such as `utf8mb3` and `utf8mb4`, `InnoDB` attempts to store `CHAR(N)` in *`N`* bytes by trimming trailing spaces. If the byte length of a `CHAR(N)` column value exceeds *`N`* bytes, trailing spaces are trimmed to a minimum of the column value byte length. The maximum length of a `CHAR(N)` column is the maximum character byte length × *`N`*.

  A minimum of *`N`* bytes is reserved for `CHAR(N)`. Reserving the minimum space *`N`* in many cases enables column updates to be done in place without causing index page fragmentation. By comparison, `CHAR(N)` columns occupy the maximum character byte length × *`N`* when using the `REDUNDANT` row format.

  Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length fields, which can be stored off-page. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.

DYNAMIC Row Format

The `DYNAMIC` row format offers the same storage characteristics as the `COMPACT` row format but adds enhanced storage capabilities for long variable-length columns and supports large index key prefixes.

When a table is created with `ROW_FORMAT=DYNAMIC`, `InnoDB` can store long variable-length column values (for `VARCHAR`, `VARBINARY`, and `BLOB` and `TEXT` types) fully off-page, with the clustered index record containing only a 20-byte pointer to the overflow page. Fixed-length fields greater than or equal to 768 bytes are encoded as variable-length fields. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.

Whether columns are stored off-page depends on the page size and the total size of the row. When a row is too long, the longest columns are chosen for off-page storage until the clustered index record fits on the B-tree page. `TEXT` and `BLOB` columns that are less than or equal to 40 bytes are stored in line.

The `DYNAMIC` row format maintains the efficiency of storing the entire row in the index node if it fits (as do the `COMPACT` and `REDUNDANT` formats), but the `DYNAMIC` row format avoids the problem of filling B-tree nodes with a large number of data bytes of long columns. The `DYNAMIC` row format is based on the idea that if a portion of a long data value is stored off-page, it is usually most efficient to store the entire value off-page. With `DYNAMIC` format, shorter columns are likely to remain in the B-tree node, minimizing the number of overflow pages required for a given row.

The `DYNAMIC` row format supports index key prefixes up to 3072 bytes.

Tables that use the `DYNAMIC` row format can be stored in the system tablespace, file-per-table tablespaces, and general tablespaces. To store `DYNAMIC` tables in the system tablespace, either disable `innodb_file_per_table` and use a regular `CREATE TABLE` or `ALTER TABLE` statement, or use the `TABLESPACE [=] innodb_system` table option with `CREATE TABLE` or `ALTER TABLE`. The `innodb_file_per_table` variable is not applicable to general tablespaces, nor is it applicable when using the `TABLESPACE [=] innodb_system` table option to store `DYNAMIC` tables in the system tablespace.

DYNAMIC Row Format Storage Characteristics

The `DYNAMIC` row format is a variation of the `COMPACT` row format. For storage characteristics, see COMPACT Row Format Storage Characteristics.

COMPRESSED Row Format

The COMPRESSED row format offers the same storage characteristics and capabilities as the DYNAMIC row format but adds support for table and index data compression.

The COMPRESSED row format uses similar internal details for off-page storage as the DYNAMIC row format, with additional storage and performance considerations from the table and index data being compressed and using smaller page sizes. With the COMPRESSED row format, the KEY_BLOCK_SIZE option controls how much column data is stored in the clustered index, and how much is placed on overflow pages. For more information about the COMPRESSED row format, see Section 15.9, “InnoDB Table and Page Compression”.

The COMPRESSED row format supports index key prefixes up to 3072 bytes.

Tables that use the COMPRESSED row format can be created in file-per-table tablespaces or general tablespaces. The system tablespace does not support the COMPRESSED row format. To store a COMPRESSED table in a file-per-table tablespace, the innodb_file_per_table variable must be enabled. The innodb_file_per_table variable is not applicable to general tablespaces. General tablespaces support all row formats with the caveat that compressed and uncompressed tables cannot coexist in the same general tablespace due to different physical page sizes. For more information, see Section 15.6.3.3, “General Tablespaces”.

Compressed Row Format Storage Characteristics

The COMPRESSED row format is a variation of the COMPACT row format. For storage characteristics, see COMPACT Row Format Storage Characteristics.

Defining the Row Format of a Table

The default row format for InnoDB tables is defined by innodb_default_row_format variable, which has a default value of DYNAMIC. The default row format is used when the ROW_FORMAT table option is not defined explicitly or when ROW_FORMAT=DEFAULT is specified.

The row format of a table can be defined explicitly using the ROW_FORMAT table option in a CREATE TABLE or ALTER TABLE statement. For example:

```
CREATE TABLE t1 (c1 INT) ROW_FORMAT=DYNAMIC;
```

An explicitly defined ROW_FORMAT setting overrides the default row format. Specifying ROW_FORMAT=DEFAULT is equivalent to using the implicit default.

The innodb_default_row_format variable can be set dynamically:

```
mysql> SET GLOBAL innodb_default_row_format=DYNAMIC;
```

Valid innodb_default_row_format options include DYNAMIC, COMPACT, and REDUNDANT. The COMPRESSED row format, which is not supported for use in the system tablespace, cannot be defined as the default. It can only be specified explicitly in a CREATE TABLE or ALTER TABLE statement. Attempting to set the innodb_default_row_format variable to COMPRESSED returns an error:

```
mysql> SET GLOBAL innodb_default_row_format=COMPRESSED;
ERROR 1231 (42000): Variable 'innodb_default_row_format'
can't be set to the value of 'COMPRESSED'
```

Newly created tables use the row format defined by the innodb_default_row_format variable when a ROW_FORMAT option is not specified explicitly, or when ROW_FORMAT=DEFAULT is used. For example, the following CREATE TABLE statements use the row format defined by the innodb_default_row_format variable.

```
CREATE TABLE t1 (c1 INT);
```

```
CREATE TABLE t2 (c1 INT) ROW_FORMAT=DEFAULT;
```

When a `ROW_FORMAT` option is not specified explicitly, or when `ROW_FORMAT=DEFAULT` is used, an operation that rebuilds a table silently changes the row format of the table to the format defined by the `innodb_default_row_format` variable.

Table-rebuilding operations include `ALTER TABLE` operations that use `ALGORITHM=COPY` or `ALGORITHM=INPLACE` where table rebuilding is required. See Section 15.12.1, “Online DDL Operations” for more information. `OPTIMIZE TABLE` is also a table-rebuilding operation.

The following example demonstrates a table-rebuilding operation that silently changes the row format of a table created without an explicitly defined row format.

```
mysql> SELECT @@innodb_default_row_format;
+-----------------------------+
| @@innodb_default_row_format |
+-----------------------------+
| dynamic                     |
+-----------------------------+

mysql> CREATE TABLE t1 (c1 INT);

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME LIKE 'test/t1' \G
*************************** 1. row ***************************
     TABLE_ID: 54
         NAME: test/t1
         FLAG: 33
       N_COLS: 4
        SPACE: 35
   ROW_FORMAT: Dynamic
ZIP_PAGE_SIZE: 0
   SPACE_TYPE: Single

mysql> SET GLOBAL innodb_default_row_format=COMPACT;

mysql> ALTER TABLE t1 ADD COLUMN (c2 INT);

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME LIKE 'test/t1' \G
*************************** 1. row ***************************
     TABLE_ID: 55
         NAME: test/t1
         FLAG: 1
       N_COLS: 5
        SPACE: 36
   ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
   SPACE_TYPE: Single
```

Consider the following potential issues before changing the row format of existing tables from `REDUNDANT` or `COMPACT` to `DYNAMIC`.

- The `REDUNDANT` and `COMPACT` row formats support a maximum index key prefix length of 767 bytes whereas `DYNAMIC` and `COMPRESSED` row formats support an index key prefix length of 3072 bytes. In a replication environment, if the `innodb_default_row_format` variable is set to `DYNAMIC` on the master, and set to `COMPACT` on the slave, the following DDL statement, which does not explicitly define a row format, succeeds on the master but fails on the slave:

  ```
  CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 VARCHAR(5000), KEY i1(c2(3070)));
  ```

For related information, see Section 15.6.1.6, “Limits on InnoDB Tables”.

- Importing a table that does not explicitly define a row format results in a schema mismatch error if the `innodb_default_row_format` setting on the source server differs from the setting on the destination server. For more information, refer to the limitations outlined in Section 15.6.3.7, “Copying Tablespaces to Another Instance”.

Determining the Row Format of a Table

To determine the row format of a table, use `SHOW TABLE STATUS`:

```
mysql> SHOW TABLE STATUS IN test1\G
*************************** 1. row ***************************
           Name: t1
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 16384
      Data_free: 0
 Auto_increment: 1
    Create_time: 2016-09-14 16:29:38
    Update_time: NULL
     Check_time: NULL
      Collation: utf8mb4_0900_ai_ci
       Checksum: NULL
 Create_options:
        Comment:
```

Alternatively, query the `INFORMATION_SCHEMA.INNODB_TABLES` table:

```
mysql> SELECT NAME, ROW_FORMAT FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME='test1/t1';
+----------+------------+
| NAME     | ROW_FORMAT |
+----------+------------+
| test1/t1 | Dynamic    |
+----------+------------+
```

15.11 InnoDB Disk I/O and File Space Management

As a DBA, you must manage disk I/O to keep the I/O subsystem from becoming saturated, and manage disk space to avoid filling up storage devices. The ACID design model requires a certain amount of I/O that might seem redundant, but helps to ensure data reliability. Within these constraints, `InnoDB` tries to optimize the database work and the organization of disk files to minimize the amount of disk I/O. Sometimes, I/O is postponed until the database is not busy, or until everything needs to be brought to a consistent state, such as during a database restart after a fast shutdown.

This section discusses the main considerations for I/O and disk space with the default kind of MySQL tables (also known as `InnoDB` tables):

- Controlling the amount of background I/O used to improve query performance.
- Enabling or disabling features that provide extra durability at the expense of additional I/O.
- Organizing tables into many small files, a few larger files, or a combination of both.
- Balancing the size of redo log files against the I/O activity that occurs when the log files become full.

- How to reorganize a table for optimal query performance.

15.11.1 InnoDB Disk I/O

InnoDB uses asynchronous disk I/O where possible, by creating a number of threads to handle I/O operations, while permitting other database operations to proceed while the I/O is still in progress. On Linux and Windows platforms, InnoDB uses the available OS and library functions to perform "native" asynchronous I/O. On other platforms, InnoDB still uses I/O threads, but the threads may actually wait for I/O requests to complete; this technique is known as "simulated" asynchronous I/O.

Read-Ahead

If InnoDB can determine there is a high probability that data might be needed soon, it performs read-ahead operations to bring that data into the buffer pool so that it is available in memory. Making a few large read requests for contiguous data can be more efficient than making several small, spread-out requests. There are two read-ahead heuristics in InnoDB:

- In sequential read-ahead, if InnoDB notices that the access pattern to a segment in the tablespace is sequential, it posts in advance a batch of reads of database pages to the I/O system.
- In random read-ahead, if InnoDB notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool, it posts the remaining reads to the I/O system.

For information about configuring read-ahead heuristics, see Section 15.8.3.4, "Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)".

Doublewrite Buffer

InnoDB uses a novel file flush technique involving a structure called the doublewrite buffer, which is enabled by default in most cases (innodb_doublewrite=ON). It adds safety to recovery following a crash or power outage, and improves performance on most varieties of Unix by reducing the need for fsync() operations.

Before writing pages to a data file, InnoDB first writes them to a contiguous tablespace area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed does InnoDB write the pages to their proper positions in the data file. If there is an operating system, storage subsystem, or mysqld process crash in the middle of a page write (causing a torn page condition), InnoDB can later find a good copy of the page from the doublewrite buffer during recovery.

If system tablespace files ("ibdata files") are located on Fusion-io devices that support atomic writes, doublewrite buffering is automatically disabled and Fusion-io atomic writes are used for all data files. Because the doublewrite buffer setting is global, doublewrite buffering is also disabled for data files residing on non-Fusion-io hardware. This feature is only supported on Fusion-io hardware and is only enabled for Fusion-io NVMFS on Linux. To take full advantage of this feature, an innodb_flush_method setting of O_DIRECT is recommended.

15.11.2 File Space Management

The data files that you define in the configuration file using the innodb_data_file_path configuration option form the InnoDB system tablespace. The files are logically concatenated to form the system tablespace. There is no striping in use. You cannot define where within the system tablespace your tables are allocated. In a newly created system tablespace, InnoDB allocates space starting from the first data file.

To avoid the issues that come with storing all tables and indexes inside the system tablespace, you can enable the innodb_file_per_table configuration option (the default), which stores each newly created table in a separate tablespace file (with extension .ibd). For tables stored this way, there is less

fragmentation within the disk file, and when the table is truncated, the space is returned to the operating system rather than still being reserved by InnoDB within the system tablespace. For more information, see Section 15.6.3.2, “File-Per-Table Tablespaces”.

You can also store tables in general tablespaces. General tablespaces are shared tablespaces created using `CREATE TABLESPACE` syntax. They can be created outside of the MySQL data directory, are capable of holding multiple tables, and support tables of all row formats. For more information, see Section 15.6.3.3, “General Tablespaces”.

Pages, Extents, Segments, and Tablespaces

Each tablespace consists of database pages. Every tablespace in a MySQL instance has the same page size. By default, all tablespaces have a page size of 16KB; you can reduce the page size to 8KB or 4KB by specifying the `innodb_page_size` option when you create the MySQL instance. You can also increase the page size to 32KB or 64KB. For more information, refer to the `innodb_page_size` documentation.

The pages are grouped into extents of size 1MB for pages up to 16KB in size (64 consecutive 16KB pages, or 128 8KB pages, or 256 4KB pages). For a page size of 32KB, extent size is 2MB. For page size of 64KB, extent size is 4MB. The “files” inside a tablespace are called segments in `InnoDB`. (These segments are different from the rollback segment, which actually contains many tablespace segments.)

When a segment grows inside the tablespace, `InnoDB` allocates the first 32 pages to it one at a time. After that, `InnoDB` starts to allocate whole extents to the segment. `InnoDB` can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

Two segments are allocated for each index in `InnoDB`. One is for nonleaf nodes of the B-tree, the other is for the leaf nodes. Keeping the leaf nodes contiguous on disk enables better sequential I/O operations, because these leaf nodes contain the actual table data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an `InnoDB` tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you ask for available free space in the tablespace by issuing a `SHOW TABLE STATUS` statement, `InnoDB` reports the extents that are definitely free in the tablespace. `InnoDB` always reserves some extents for cleanup and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, `InnoDB` contracts the corresponding B-tree indexes. Whether the freed space becomes available for other users depends on whether the pattern of deletes frees individual pages or extents to the tablespace. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows are physically removed only by the purge operation, which happens automatically some time after they are no longer needed for transaction rollbacks or consistent reads. (See Section 15.3, “InnoDB Multi-Versioning”.)

How Pages Relate to Table Rows

The maximum row length is slightly less than half a database page for 4KB, 8KB, 16KB, and 32KB `innodb_page_size` settings. For example, the maximum row length is slightly less than 8KB for the default 16KB `InnoDB` page size. For 64KB pages, the maximum row length is slightly less than 16KB.

If a row does not exceed the maximum row length, all of it is stored locally within the page. If a row exceeds the maximum row length, variable-length columns are chosen for external off-page storage until the row fits within the maximum row length limit. External off-page storage for variable-length columns differs by row format:

- *COMPACT and REDUNDANT Row Formats*

  When a variable-length column is chosen for external off-page storage, `InnoDB` stores the first 768 bytes locally in the row, and the rest externally into overflow pages. Each such column has its own list of

overflow pages. The 768-byte prefix is accompanied by a 20-byte value that stores the true length of the column and points into the overflow list where the rest of the value is stored. See Section 15.10, “InnoDB Row Formats”.

- *DYNAMIC and COMPRESSED Row Formats*

  When a variable-length column is chosen for external off-page storage, `InnoDB` stores a 20-byte pointer locally in the row, and the rest externally into overflow pages. See Section 15.10, “InnoDB Row Formats”.

`LONGBLOB` and `LONGTEXT` columns must be less than 4GB, and the total row length, including `BLOB` and `TEXT` columns, must be less than 4GB.

15.11.3 InnoDB Checkpoints

Making your log files very large may reduce disk I/O during checkpointing. It often makes sense to set the total size of the log files as large as the buffer pool or even larger.

How Checkpoint Processing Works

`InnoDB` implements a checkpoint mechanism known as fuzzy checkpointing. `InnoDB` flushes modified database pages from the buffer pool in small batches. There is no need to flush the buffer pool in one single batch, which would disrupt processing of user SQL statements during the checkpointing process.

During crash recovery, `InnoDB` looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are present in the disk image of the database. Then `InnoDB` scans the log files forward from the checkpoint, applying the logged modifications to the database.

15.11.4 Defragmenting a Table

Random insertions into or deletions from a secondary index can cause the index to become fragmented. Fragmentation means that the physical ordering of the index pages on the disk is not close to the index ordering of the records on the pages, or that there are many unused pages in the 64-page blocks that were allocated to the index.

One symptom of fragmentation is that a table takes more space than it “should” take. How much that is exactly, is difficult to determine. All `InnoDB` data and indexes are stored in B-trees, and their fill factor may vary from 50% to 100%. Another symptom of fragmentation is that a table scan such as this takes more time than it “should” take:

```
SELECT COUNT(*) FROM t WHERE non_indexed_column <> 12345;
```

The preceding query requires MySQL to perform a full table scan, the slowest type of query for a large table.

To speed up index scans, you can periodically perform a “null” `ALTER TABLE` operation, which causes MySQL to rebuild the table:

```
ALTER TABLE tbl_name ENGINE=INNODB
```

You can also use `ALTER TABLE tbl_name FORCE` to perform a “null” alter operation that rebuilds the table.

Both `ALTER TABLE tbl_name ENGINE=INNODB` and `ALTER TABLE tbl_name FORCE` use online DDL. For more information, see Section 15.12, “InnoDB and Online DDL”.

Another way to perform a defragmentation operation is to use `mysqldump` to dump the table to a text file, drop the table, and reload it from the dump file.

If the insertions into an index are always ascending and records are deleted only from the end, the `InnoDB` filespace management algorithm guarantees that fragmentation in the index does not occur.

15.11.5 Reclaiming Disk Space with TRUNCATE TABLE

To reclaim operating system disk space when truncating an `InnoDB` table, the table must be stored in its own .ibd file. For a table to be stored in its own .ibd file, `innodb_file_per_table` must enabled when the table is created. Additionally, there cannot be a foreign key constraint between the table being truncated and other tables, otherwise the `TRUNCATE TABLE` operation fails. A foreign key constraint between two columns in the same table, however, is permitted.

When a table is truncated, it is dropped and re-created in a new `.ibd` file, and the freed space is returned to the operating system. This is in contrast to truncating `InnoDB` tables that are stored within the `InnoDB` system tablespace (tables created when `innodb_file_per_table=OFF`) and tables stored in shared general tablespaces, where only `InnoDB` can use the freed space after the table is truncated.

The ability to truncate tables and return disk space to the operating system also means that physical backups can be smaller. Truncating tables that are stored in the system tablespace (tables created when `innodb_file_per_table=OFF`) or in a general tablespace leaves blocks of unused space in the tablespace.

15.12 InnoDB and Online DDL

The online DDL feature provides support for instant and in-place table alterations and concurrent DML. Benefits of this feature include:

- Improved responsiveness and availability in busy production environments, where making a table unavailable for minutes or hours is not practical.
- For in-place operations, the ability to adjust the balance between performance and concurrency during DDL operations using the `LOCK` clause. See The LOCK clause.
- Less disk space usage and I/O overhead than the table-copy method.

Note

`ALGORITHM=INSTANT` support is available for `ADD COLUMN` and other operations in MySQL 8.0.12.

Typically, you do not need to do anything special to enable online DDL. By default, MySQL performs the operation instantly or in place, as permitted, with as little locking as possible.

You can control aspects of a DDL operation using the `ALGORITHM` and `LOCK` clauses of the `ALTER TABLE` statement. These clauses are placed at the end of the statement, separated from the table and column specifications by commas. For example:

```
ALTER TABLE tbl_name ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

The `LOCK` clause may be used for operations that are performed in place and is useful for fine-tuning the degree of concurrent access to the table during operations. Only `LOCK=DEFAULT` is supported for operations that are performed instantly. The `ALGORITHM` clause is primarily intended for performance comparisons and as a fallback to the older table-copying behavior in case you encounter any issues. For example:

- To avoid accidentally making the table unavailable for reads, writes, or both, during an in-place `ALTER TABLE` operation, specify a clause on the `ALTER TABLE` statement such as `LOCK=NONE` (permit reads and writes) or `LOCK=SHARED` (permit reads). The operation halts immediately if the requested level of concurrency is not available.
- To compare performance between algorithms, run a statement with `ALGORITHM=INSTANT`, `ALGORITHM=INPLACE` and `ALGORITHM=COPY`. You can also run a statement with the `old_alter_table` configuration option enabled to force the use of `ALGORITHM=COPY`.
- To avoid tying up the server with an `ALTER TABLE` operation that copies the table, include `ALGORITHM=INSTANT` or `ALGORITHM=INPLACE`. The statement halts immediately if it cannot use the specified algorithm.

15.12.1 Online DDL Operations

Online support details, syntax examples, and usage notes for DDL operations are provided under the following topics in this section.

- Index Operations
- Primary Key Operations
- Column Operations
- Generated Column Operations
- Foreign Key Operations
- Table Operations
- Tablespace Operations
- Partitioning Operations

Index Operations

The following table provides an overview of online DDL support for index operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 15.17 Online DDL Support for Index Operations

Operation	Instant	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Creating or adding a secondary index	No	Yes	No	Yes	No
Dropping an index	No	Yes	No	Yes	Yes
Renaming an index	No	Yes	No	Yes	Yes
Adding a `FULLTEXT` index	No	Yes*	No*	No	No

Operation	Instant	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a SPATIAL index	No	Yes	No	No	No
Changing the index type	Yes	Yes	No	Yes	Yes

Syntax and Usage Notes

- Creating or adding a secondary index

```
CREATE INDEX name ON table (col_list);
```

```
ALTER TABLE tbl_name ADD INDEX name (col_list);
```

The table remains available for read and write operations while the index is being created. The CREATE INDEX statement only finishes after all transactions that are accessing the table are completed, so that the initial state of the index reflects the most recent contents of the table.

Online DDL support for adding secondary indexes means that you can generally speed the overall process of creating and loading a table and associated indexes by creating the table without secondary indexes, then adding secondary indexes after the data is loaded.

A newly created secondary index contains only the committed data in the table at the time the CREATE INDEX or ALTER TABLE statement finishes executing. It does not contain any uncommitted values, old versions of values, or values marked for deletion but not yet removed from the old index.

Some factors affect the performance, space usage, and semantics of this operation. For details, see Section 15.12.6, “Online DDL Limitations”.

- Dropping an index

```
DROP INDEX name ON table;
```

```
ALTER TABLE tbl_name DROP INDEX name;
```

The table remains available for read and write operations while the index is being dropped. The DROP INDEX statement only finishes after all transactions that are accessing the table are completed, so that the initial state of the index reflects the most recent contents of the table.

- Renaming an index

```
ALTER TABLE tbl_name RENAME INDEX old_index_name TO new_index_name, ALGORITHM=INPLACE, LOCK=NONE;
```

- Adding a FULLTEXT index

```
CREATE FULLTEXT INDEX name ON table(column);
```

Adding the first FULLTEXT index rebuilds the table if there is no user-defined FTS_DOC_ID column. Additional FULLTEXT indexes may be added without rebuilding the table.

- Adding a SPATIAL index

```
CREATE TABLE geom (g GEOMETRY NOT NULL);
ALTER TABLE geom ADD SPATIAL INDEX(g), ALGORITHM=INPLACE, LOCK=SHARED;
```

Adding the first FULLTEXT index rebuilds the table if there is no user-defined FTS_DOC_ID column. Additional FULLTEXT indexes may be added without rebuilding the table.

- Changing the index type (USING {BTREE | HASH})

```
ALTER TABLE tbl_name DROP INDEX i1, ADD INDEX i1(key_part,...) USING BTREE, ALGORITHM=INSTANT;
```

Primary Key Operations

The following table provides an overview of online DDL support for primary key operations. An asterisk indicates additional information, an exception, or a dependency. See Syntax and Usage Notes.

Table 15.18 Online DDL Support for Primary Key Operations

Operation	Instant	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a primary key	No	Yes*	Yes*	Yes	No
Dropping a primary key	No	No	Yes	No	No
Dropping a primary key and adding another	No	Yes	Yes	Yes	No

Syntax and Usage Notes

- Adding a primary key

```
ALTER TABLE tbl_name ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

Rebuilds the table in place. Data is reorganized substantially, making it an expensive operation. ALGORITHM=INPLACE is not permitted under certain conditions if columns have to be converted to NOT NULL.

Restructuring the clustered index always requires copying of table data. Thus, it is best to define the primary key when you create a table, rather than issuing ALTER TABLE ... ADD PRIMARY KEY later.

When you create a UNIQUE or PRIMARY KEY index, MySQL must do some extra work. For UNIQUE indexes, MySQL checks that the table contains no duplicate values for the key. For a PRIMARY KEY index, MySQL also checks that none of the PRIMARY KEY columns contains a NULL.

When you add a primary key using the ALGORITHM=COPY clause, MySQL converts NULL values in the associated columns to default values: 0 for numbers, an empty string for character-based columns and BLOBs, and 0000-00-00 00:00:00 for DATETIME. This is a non-standard behavior that Oracle recommends you not rely on. Adding a primary key using ALGORITHM=INPLACE is only permitted when the SQL_MODE setting includes the strict_trans_tables or strict_all_tables flags; when the SQL_MODE setting is strict, ALGORITHM=INPLACE is permitted, but the statement can still fail if the requested primary key columns contain NULL values. The ALGORITHM=INPLACE behavior is more standard-compliant.

If you create a table without a primary key, `InnoDB` chooses one for you, which can be the first `UNIQUE` key defined on `NOT NULL` columns, or a system-generated key. To avoid uncertainty and the potential space requirement for an extra hidden column, specify the `PRIMARY KEY` clause as part of the `CREATE TABLE` statement.

MySQL creates a new clustered index by copying the existing data from the original table to a temporary table that has the desired index structure. Once the data is completely copied to the temporary table, the original table is renamed with a different temporary table name. The temporary table comprising the new clustered index is renamed with the name of the original table, and the original table is dropped from the database.

The online performance enhancements that apply to operations on secondary indexes do not apply to the primary key index. The rows of an InnoDB table are stored in a clustered index organized based on the primary key, forming what some database systems call an “index-organized table”. Because the table structure is closely tied to the primary key, redefining the primary key still requires copying the data.

When an operation on the primary key uses `ALGORITHM=INPLACE`, even though the data is still copied, it is more efficient than using `ALGORITHM=COPY` because:

- No undo logging or associated redo logging is required for `ALGORITHM=INPLACE`. These operations add overhead to DDL statements that use `ALGORITHM=COPY`.
- The secondary index entries are pre-sorted, and so can be loaded in order.
- The change buffer is not used, because there are no random-access inserts into the secondary indexes.

- Dropping a primary key

```
ALTER TABLE tbl_name DROP PRIMARY KEY, ALGORITHM=COPY;
```

Only `ALGORITHM=COPY` supports dropping a primary key without adding a new one in the same `ALTER TABLE` statement.

- Dropping a primary key and adding another

```
ALTER TABLE tbl_name DROP PRIMARY KEY, ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

Data is reorganized substantially, making it an expensive operation.

Column Operations

The following table provides an overview of online DDL support for column operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 15.19 Online DDL Support for Column Operations

Operation	Instant	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a column	Yes*	Yes	No*	Yes*	No
Dropping a column	No	Yes	Yes	Yes	No

Operation	Instant	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Renaming a column	No	Yes	No	Yes*	Yes
Reordering columns	No	Yes	Yes	Yes	No
Setting a column default value	Yes	Yes	No	Yes	Yes
Changing the column data type	No	No	Yes	No	No
Extending `VARCHAR` column size	No	Yes	No	Yes	Yes
Dropping the column default value	Yes	Yes	No	Yes	Yes
Changing the auto-increment value	No	Yes	No	Yes	No*
Making a column `NULL`	No	Yes	Yes*	Yes	No
Making a column `NOT NULL`	No	Yes*	Yes*	Yes	No
Modifying the definition of an `ENUM` or `SET` column	Yes	Yes	No	Yes	Yes

Syntax and Usage Notes

- Adding a column

```
ALTER TABLE tbl_name ADD COLUMN column_name column_definition, ALGORITHM=INSTANT;
```

The following limitations apply when the `INSTANT` algorithm is used to add a column:

- Adding a column cannot be combined in the same statement with other `ALTER TABLE` actions that do not support `ALGORITHM=INSTANT`.
- A column can only be added as the last column of the table. Adding a column to any other position among other columns is not supported.
- Columns cannot be added to tables that use `ROW_FORMAT=COMPRESSED`.
- Columns cannot be added to tables that include a `FULLTEXT` index.
- Columns cannot be added to temporary tables. Temporary tables only support `ALGORITHM=COPY`.

- Columns cannot be added to tables that reside in the data dictionary tablespace.
- Row size limits are not evaluated when adding a column. However, row size limits are checked during DML operations that insert and update rows in the table.

Multiple columns may be added in the same `ALTER TABLE` statement. For example:

```
ALTER TABLE t1 ADD COLUMN c2 INT, ADD COLUMN c3 INT, ALGORITHM=INSTANT;
```

`INFORMATION_SCHEMA.INNODB_TABLES` and `INFORMATION_SCHEMA.INNODB_COLUMNS` provide metadata for instantly added columns. `INFORMATION_SCHEMA.INNODB_TABLES.INSTANT_COLS` shows number of columns in the table prior to adding the first instant column. `INFORMATION_SCHEMA.INNODB_COLUMNS.HAS_DEFAULT` and `DEFAULT_VALUE` provide metadata about default values for instantly added columns.

Concurrent DML is not permitted when adding an auto-increment column. Data is reorganized substantially, making it an expensive operation. At a minimum, `ALGORITHM=INPLACE, LOCK=SHARED` is required.

The table is rebuilt if `ALGORITHM=INPLACE` is used to add a column.

- Dropping a column

```
ALTER TABLE tbl_name DROP COLUMN column_name, ALGORITHM=INPLACE, LOCK=NONE;
```

Data is reorganized substantially, making it an expensive operation.

- Renaming a column

```
ALTER TABLE tbl CHANGE old_col_name new_col_name data_type, ALGORITHM=INPLACE, LOCK=NONE;
```

To permit concurrent DML, keep the same data type and only change the column name.

When you keep the same data type and `[NOT] NULL` attribute, only changing the column name, the operation can always be performed online.

You can also rename a column that is part of a foreign key constraint. The foreign key definition is automatically updated to use the new column name. Renaming a column participating in a foreign key only works with `ALGORITHM=INPLACE`. If you use the `ALGORITHM=COPY` clause, or some other condition causes the command to use `ALGORITHM=COPY` behind the scenes, the `ALTER TABLE` statement fails.

`ALGORITHM=INPLACE` is not supported for renaming a generated column.

- Reordering columns

To reorder columns, use `FIRST` or `AFTER` in `CHANGE` or `MODIFY` operations.

```
ALTER TABLE tbl_name MODIFY COLUMN col_name column_definition FIRST, ALGORITHM=INPLACE, LOCK=NONE;
```

Data is reorganized substantially, making it an expensive operation.

- Changing the column data type

```
```

```
ALTER TABLE tbl_name CHANGE c1 c1 BIGINT, ALGORITHM=COPY;
```

Changing the column data type is only supported with ALGORITHM=COPY.

- Extending VARCHAR column size

```
ALTER TABLE tbl_name CHANGE COLUMN c1 c1 VARCHAR(255), ALGORITHM=INPLACE, LOCK=NONE;
```

The number of length bytes required by a VARCHAR column must remain the same. For VARCHAR columns of 0 to 255 bytes in size, one length byte is required to encode the value. For VARCHAR columns of 256 bytes in size or more, two length bytes are required. As a result, in-place ALTER TABLE only supports increasing VARCHAR column size from 0 to 255 bytes, or from 256 bytes to a greater size. In-place ALTER TABLE does not support increasing the size of a VARCHAR column from less than 256 bytes to a size equal to or greater than 256 bytes. In this case, the number of required length bytes changes from 1 to 2, which is only supported by a table copy (ALGORITHM=COPY). For example, attempting to change VARCHAR column size for a single byte character set from VARCHAR(255) to VARCHAR(256) using in-place ALTER TABLE returns this error:

```
ALTER TABLE tbl_name ALGORITHM=INPLACE, CHANGE COLUMN c1 c1 VARCHAR(256);
ERROR 0A000: ALGORITHM=INPLACE is not supported. Reason: Cannot change
column type INPLACE. Try ALGORITHM=COPY.
```

Note

The byte length of a VARCHAR column is dependant on the byte length of the character set.

Decreasing VARCHAR size using in-place ALTER TABLE is not supported. Decreasing VARCHAR size requires a table copy (ALGORITHM=COPY).

- Setting a column default value

```
ALTER TABLE tbl_name ALTER COLUMN col SET DEFAULT literal, ALGORITHM=INSTANT;
```

Only modifies table metadata. Default column values are stored in the data dictionary.

- Dropping a column default value

```
ALTER TABLE tbl ALTER COLUMN col DROP DEFAULT, ALGORITHM=INSTANT;
```

- Changing the auto-increment value

```
ALTER TABLE table AUTO_INCREMENT=next_value, ALGORITHM=INPLACE, LOCK=NONE;
```

Modifies a value stored in memory, not the data file.

In a distributed system using replication or sharding, you sometimes reset the auto-increment counter for a table to a specific value. The next row inserted into the table uses the specified value for its auto-increment column. You might also use this technique in a data warehousing environment where you periodically empty all the tables and reload them, and restart the auto-increment sequence from 1.

- Making a column NULL

```
ALTER TABLE tbl_name MODIFY COLUMN column_name data_type NULL, ALGORITHM=INPLACE, LOCK=NONE;
```

Rebuilds the table in place. Data is reorganized substantially, making it an expensive operation.

- Making a column `NOT NULL`

```
ALTER TABLE tbl_name MODIFY COLUMN column_name data_type NOT NULL, ALGORITHM=INPLACE, LOCK=NONE;
```

Rebuilds the table in place. `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` `SQL_MODE` is required for the operation to succeed. The operation fails if the column contains NULL values. The server prohibits changes to foreign key columns that have the potential to cause loss of referential integrity. See Section 13.1.9, “ALTER TABLE Syntax”. Data is reorganized substantially, making it an expensive operation.

- Modifying the definition of an `ENUM` or `SET` column

```
CREATE TABLE t1 (c1 ENUM('a', 'b', 'c'));
ALTER TABLE t1 MODIFY COLUMN c1 ENUM('a', 'b', 'c', 'd'), ALGORITHM=INSTANT;
```

Modifying the definition of an `ENUM` or `SET` column by adding new enumeration or set members to the *end* of the list of valid member values may be performed instantly or in place, as long as the storage size of the data type does not change. For example, adding a member to a `SET` column that has 8 members changes the required storage per value from 1 byte to 2 bytes; this requires a table copy. Adding members in the middle of the list causes renumbering of existing members, which requires a table copy.

Generated Column Operations

The following table provides an overview of online DDL support for generated column operations. For details, see Syntax and Usage Notes.

Table 15.20 Online DDL Support for Generated Column Operations

Operation	Instant	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a `STORED` column	No	No	Yes	No	No
Modifying `STORED` column order	No	No	Yes	No	No
Dropping a `STORED` column	No	Yes	Yes	Yes	No
Adding a `VIRTUAL` column	Yes	Yes	No	Yes	Yes
Modifying `VIRTUAL` column order	No	No	Yes	No	No
Dropping a `VIRTUAL` column	Yes	Yes	No	Yes	Yes

Syntax and Usage Notes

- Adding a `STORED` column

```
ALTER TABLE t1 ADD COLUMN (c2 INT GENERATED ALWAYS AS (c1 + 1) STORED), ALGORITHM=COPY;
```

  `ADD COLUMN` is not an in-place operation for stored columns (done without using a temporary table) because the expression must be evaluated by the server.

- Modifying `STORED` column order

```
ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED FIRST, ALGORITHM=COPY;
```

  Rebuilds the table in place.

- Dropping a `STORED` column

```
ALTER TABLE t1 DROP COLUMN c2, ALGORITHM=INPLACE, LOCK=NONE;
```

  Rebuilds the table in place.

- Adding a `VIRTUAL` column

```
ALTER TABLE t1 ADD COLUMN (c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL), ALGORITHM=INSTANT;
```

  Adding a virtual column can be performed instantly or in place for non-partitioned tables.

  Adding a `VIRTUAL` is not an in-place operation for partitioned tables.

- Modifying `VIRTUAL` column order

```
ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL FIRST, ALGORITHM=COPY;
```

- Dropping a `VIRTUAL` column

```
ALTER TABLE t1 DROP COLUMN c2, ALGORITHM=INSTANT;
```

  Dropping a `VIRTUAL` column can be performed instantly or in place for non-partitioned tables.

Foreign Key Operations

The following table provides an overview of online DDL support for foreign key operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 15.21 Online DDL Support for Foreign Key Operations

Operation	Instant	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a foreign key constraint	No	Yes*	No	Yes	Yes

Operation	Instant	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Dropping a foreign key constraint	No	Yes	No	Yes	Yes

Syntax and Usage Notes

- Adding a foreign key constraint

  The `INPLACE` algorithm is supported when `foreign_key_checks` is disabled. Otherwise, only the `COPY` algorithm is supported.

```
ALTER TABLE tbl1 ADD CONSTRAINT fk_name FOREIGN KEY index (col1)
  REFERENCES tbl2(col2) referential_actions;
```

- Dropping a foreign key constraint

```
ALTER TABLE tbl DROP FOREIGN KEY fk_name;
```

  Dropping a foreign key can be performed online with the `foreign_key_checks` option enabled or disabled.

  If you do not know the names of the foreign key constraints on a particular table, issue the following statement and find the constraint name in the `CONSTRAINT` clause for each foreign key:

```
SHOW CREATE TABLE table\G
```

  Or, query the `INFORMATION_SCHEMA.TABLE_CONSTRAINTS` table and use the `CONSTRAINT_NAME` and `CONSTRAINT_TYPE` columns to identify the foreign key names.

  You can also drop a foreign key and its associated index in a single statement:

```
ALTER TABLE table DROP FOREIGN KEY constraint, DROP INDEX index;
```

Note

If foreign keys are already present in the table being altered (that is, it is a child table containing a `FOREIGN KEY ... REFERENCE` clause), additional restrictions apply to online DDL operations, even those not directly involving the foreign key columns:

- An `ALTER TABLE` on the child table could wait for another transaction to commit, if a change to the parent table causes associated changes in the child table through an `ON UPDATE` or `ON DELETE` clause using the `CASCADE` or `SET NULL` parameters.
- In the same way, if a table is the parent table in a foreign key relationship, even though it does not contain any `FOREIGN KEY` clauses, it could wait for the `ALTER TABLE` to complete if an `INSERT`, `UPDATE`, or `DELETE` statement causes an `ON UPDATE` or `ON DELETE` action in the child table.

Table Operations

The following table provides an overview of online DDL support for table operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 15.22 Online DDL Support for Table Operations

Operation	Instant	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Changing the `ROW_FORMAT`	No	Yes	Yes	Yes	No
Changing the `KEY_BLOCK_SIZE`	No	Yes	Yes	Yes	No
Setting persistent table statistics	No	Yes	No	Yes	Yes
Specifying a character set	No	Yes	Yes*	No	No
Converting a character set	No	No	Yes*	No	No
Optimizing a table	No	Yes*	Yes	Yes	No
Rebuilding with the `FORCE` option	No	Yes*	Yes	Yes	No
Performing a null rebuild	No	Yes*	Yes	Yes	No
Renaming a table	Yes	Yes	No	Yes	Yes

Syntax and Usage Notes

- Changing the `ROW_FORMAT`

```
ALTER TABLE tbl_name ROW_FORMAT = row_format, ALGORITHM=INPLACE, LOCK=NONE;
```

  Data is reorganized substantially, making it an expensive operation.

  For additional information about the `ROW_FORMAT` option, see Table Options.

- Changing the `KEY_BLOCK_SIZE`

```
ALTER TABLE tbl_name KEY_BLOCK_SIZE = value, ALGORITHM=INPLACE, LOCK=NONE;
```

  Data is reorganized substantially, making it an expensive operation.

  For additional information about the `KEY_BLOCK_SIZE` option, see Table Options.

- Setting persistent table statistics options

```
ALTER TABLE tbl_name STATS_PERSISTENT=0, STATS_SAMPLE_PAGES=20, STATS_AUTO_RECALC=1, ALGORITHM=INPLACE,
```

Only modifies table metadata.

Persistent statistics include `STATS_PERSISTENT`, `STATS_AUTO_RECALC`, and `STATS_SAMPLE_PAGES`. For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

- Specifying a character set

```
ALTER TABLE tbl_name CHARACTER SET = charset_name, ALGORITHM=INPLACE, LOCK=NONE;
```

Rebuilds the table if the new character encoding is different.

- Converting a character set

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name, ALGORITHM=COPY;
```

Rebuilds the table if the new character encoding is different.

- Optimizing a table

```
OPTIMIZE TABLE tbl_name;
```

In-place operation is not supported for tables with `FULLTEXT` indexes. The operation uses the `INPLACE` algorithm, but `ALGORITHM` and `LOCK` syntax is not permitted.

- Rebuilding a table with the `FORCE` option

```
ALTER TABLE tbl_name FORCE, ALGORITHM=INPLACE, LOCK=NONE;
```

Uses `ALGORITHM=INPLACE` as of MySQL 5.6.17. `ALGORITHM=INPLACE` is not supported for tables with `FULLTEXT` indexes.

- Performing a "null" rebuild

```
ALTER TABLE tbl_name ENGINE=InnoDB, ALGORITHM=INPLACE, LOCK=NONE;
```

Uses `ALGORITHM=INPLACE` as of MySQL 5.6.17. `ALGORITHM=INPLACE` is not supported for tables with `FULLTEXT` indexes.

- Renaming a table

```
ALTER TABLE old_tbl_name RENAME TO new_tbl_name, ALGORITHM=INSTANT;
```

Renaming a table can be performed instantly or in place. MySQL renames files that correspond to the table *`tbl_name`* without making a copy. (You can also use the `RENAME TABLE` statement to rename tables. See Section 13.1.36, “RENAME TABLE Syntax”.) Privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.

Tablespace Operations

The following table provides an overview of online DDL support for tablespace operations. For details, see Syntax and Usage Notes.

Table 15.23 Online DDL Support for Tablespace Operations

Operation	Instant	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Renaming a general tablespace	No	Yes	No	Yes	Yes
Enabling or disabling general tablespace encryption	No	Yes	No	Yes	No
Enabling or disabling file-per-table tablespace encryption	No	No	Yes	No	No

Syntax and Usage Notes

- Renaming a general tablespace

```
ALTER TABLESPACE tablespace_name RENAME TO new_tablespace_name;
```

`ALTER TABLESPACE ... RENAME TO` uses the `INPLACE` algorithm but does not support the `ALGORITHM` clause.

- Enabling or disabling general tablespace encryption

```
ALTER TABLESPACE tablespace_name ENCRYPTION='Y';
```

`ALTER TABLESPACE ... ENCRYPTION` uses the `INPLACE` algorithm but does not support the `ALGORITHM` clause.

For related information, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”.

- Enabling or disabling file-per-table tablespace encryption

```
ALTER TABLE tbl_name ENCRYPTION='Y', ALGORITHM=COPY;
```

For related information, see Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”.

Partitioning Operations

With the exception of some `ALTER TABLE` partitioning clauses, online DDL operations for partitioned `InnoDB` tables follow the same rules that apply to regular `InnoDB` tables.

Some `ALTER TABLE` partitioning clauses do not go through the same internal online DDL API as regular non-partitioned `InnoDB` tables. As a result, online support for `ALTER TABLE` partitioning clauses varies.

The following table shows the online status for each `ALTER TABLE` partitioning statement. Regardless of the online DDL API that is used, MySQL attempts to minimize data copying and locking where possible.

`ALTER TABLE` partitioning options that use `ALGORITHM=COPY` or that only permit "`ALGORITHM=DEFAULT, LOCK=DEFAULT`", repartition the table using the `COPY` algorithm. In other words, a new partitioned table is created with the new partitioning scheme. The newly created table includes any changes applied by the `ALTER TABLE` statement, and table data is copied into the new table structure.

Table 15.24 Online DDL Support for Partitioning Operations

Partitioning Clause	Instant	In Place	Permits DML	Notes
`PARTITION BY`	No	No	No	Permits `ALGORITHM=COPY`, `LOCK={DEFAULT|SHARED|EXCLUSIVE}`
`ADD PARTITION`	No	Yes*	Yes*	`ALGORITHM=INPLACE, LOCK={DEFAULT|NONE|SHARED|EXCLUSISVE}` is supported for `RANGE` and `LIST` partitions, `ALGORITHM=INPLACE, LOCK={DEFAULT|SHARED|EXCLUSISVE}` for `HASH` and `KEY` partitions, and `ALGORITHM=COPY, LOCK={SHARED|EXCLUSIVE}` for all partition types. Does not copy existing data for tables partitioned by `RANGE` or `LIST`. Concurrent queries are permitted with `ALGORITHM=COPY` for tables partitioned by `HASH` or `LIST`, as MySQL copies the data while holding a shared lock.
`DROP PARTITION`	No	Yes*	Yes*	`ALGORITHM=INPLACE, LOCK={DEFAULT|NONE|SHARED|EXCLUSIVE}` is supported. Does not copy data for tables partitioned by `RANGE` or `LIST`. `DROP PARTITION` with `ALGORITHM=INPLACE` deletes data stored in the partition and drops the partition. However, `DROP PARTITION` with `ALGORITHM=COPY` or `old_alter_table=ON` rebuilds the partitioned table and attempts to move data from the dropped partition to another partition with a compatible `PARTITION ... VALUES` definition. Data that cannot be moved to another partition is deleted.

Partitioning Clause	Instant	In Place	Permits DML	Notes
`DISCARD PARTITION`	No	No	No	Only permits `ALGORITHM=DEFAULT`, `LOCK=DEFAULT`
`IMPORT PARTITION`	No	No	No	Only permits `ALGORITHM=DEFAULT`, `LOCK=DEFAULT`
`TRUNCATE PARTITION`	No	Yes	Yes	Does not copy existing data. It merely deletes rows; it does not alter the definition of the table itself, or of any of its partitions.
`COALESCE PARTITION`	No	Yes*	No	`ALGORITHM=INPLACE, LOCK={DEFAULT|SHARED|EXCLUSIVE}` is supported.
`REORGANIZE PARTITION`	No	Yes*	No	`ALGORITHM=INPLACE, LOCK={DEFAULT|SHARED|EXCLUSIVE}` is supported.
`EXCHANGE PARTITION`	No	Yes	Yes	
`ANALYZE PARTITION`	No	Yes	Yes	
`CHECK PARTITION`	No	Yes	Yes	
`OPTIMIZE PARTITION`	No	No	No	`ALGORITHM` and `LOCK` clauses are ignored. Rebuilds the entire table. See Section 23.3.4, “Maintenance of Partitions”.
`REBUILD PARTITION`	No	Yes*	No	`ALGORITHM=INPLACE, LOCK={DEFAULT|SHARED|EXCLUSIVE}` is supported.
`REPAIR PARTITION`	No	Yes	Yes	
`REMOVE PARTITIONING`	No	No	No	Permits `ALGORITHM=COPY`, `LOCK={DEFAULT|SHARED|EXCLUSIVE}`

Non-partitioning online `ALTER TABLE` operations on partitioned tables follow the same rules that apply to regular tables. However, `ALTER TABLE` performs online operations on each table partition, which causes increased demand on system resources due to operations being performed on multiple partitions.

For additional information about `ALTER TABLE` partitioning clauses, see Partitioning Options, and Section 13.1.9.1, “ALTER TABLE Partition Operations”. For information about partitioning in general, see Chapter 23, *Partitioning*.

15.12.2 Online DDL Performance and Concurrency

Online DDL improves several aspects of MySQL operation:

- Applications that access the table are more responsive because queries and DML operations on the table can proceed while the DDL operation is in progress. Reduced locking and waiting for MySQL server resources leads to greater scalability, even for operations that are not involved in the DDL operation.

- Instant operations only modify metadata in the data dictionary. No metadata locks are taken on the table, and table data is unaffected, making operations instantaneous. Concurrent DML is unaffected.
- Online operations avoid the disk I/O and CPU cycles associated with the table-copy method, which minimizes overall load on the database. Minimizing load helps maintain good performance and high throughput during the DDL operation.
- Online operations read less data into the buffer pool than table-copy operations, which reduces purging of frequently accessed data from memory. Purging of frequently accessed data can cause a temporary performance dip after a DDL operation.

The LOCK clause

By default, MySQL uses as little locking as possible during a DDL operation. The `LOCK` clause can be specified for in-place operations and some copy operations to enforce more restrictive locking, if required. If the `LOCK` clause specifies a less restrictive level of locking than is permitted for a particular DDL operation, the statement fails with an error. `LOCK` clauses are described below, in order of least to most restrictive:

- `LOCK=NONE`:

  Permits concurrent queries and DML.

  For example, use this clause for tables involving customer signups or purchases, to avoid making the tables unavailable during lengthy DDL operations.

- `LOCK=SHARED`:

  Permits concurrent queries but blocks DML.

  For example, use this clause on data warehouse tables, where you can delay data load operations until the DDL operation is finished, but queries cannot be delayed for long periods.

- `LOCK=DEFAULT`:

  Permits as much concurrency as possible (concurrent queries, DML, or both). Omitting the `LOCK` clause is the same as specifying `LOCK=DEFAULT`.

  Use this clause when you know that the default locking level of the DDL statement will not cause availability problems for the table.

- `LOCK=EXCLUSIVE`:

  Blocks concurrent queries and DML.

  Use this clause if the primary concern is finishing the DDL operation in the shortest amount of time possible, and concurrent query and DML access is not necessary. You might also use this clause if the server is supposed to be idle, to avoid unexpected table accesses.

Online DDL and Metadata Locks

Online DDL operations can be viewed as having three phases:

- *Phase 1: Initialization*

  In the initialization phase, the server determines how much concurrency is permitted during the operation, taking into account storage engine capabilities, operations specified in the statement, and

user-specified `ALGORITHM` and `LOCK` options. During this phase, a shared upgradeable metadata lock is taken to protect the current table definition.

- *Phase 2: Execution*

  In this phase, the statement is prepared and executed. Whether the metadata lock is upgraded to exclusive depends on the factors assessed in the initialization phase. If an exclusive metadata lock is required, it is only taken briefly during statement preparation.

- *Phase 3: Commit Table Definition*

  In the commit table definition phase, the metadata lock is upgraded to exclusive to evict the old table definition and commit the new one. Once granted, the duration of the exclusive metadata lock is brief.

Due to the exclusive metadata lock requirements outlined above, an online DDL operation may have to wait for concurrent transactions that hold metadata locks on the table to commit or rollback. Transactions started before or during the DDL operation can hold metadata locks on the table being altered. In the case of a long running or inactive transaction, an online DDL operation can time out waiting for an exclusive metadata lock. Additionally, a pending exclusive metadata lock requested by an online DDL operation blocks subsequent transactions on the table.

The following example demonstrates an online DDL operation waiting for an exclusive metadata lock, and how a pending metadata lock blocks subsequent transactions on the table.

Session 1:

```
mysql> CREATE TABLE t1 (c1 INT) ENGINE=InnoDB;
mysql> START TRANSACTION;
mysql> SELECT * FROM t1;
```

The session 1 `SELECT` statement takes a shared metadata lock on table t1.

Session 2:

```
mysql> ALTER TABLE t1 ADD COLUMN x INT, ALGORITHM=INPLACE, LOCK=NONE;
```

The online DDL operation in session 2, which requires an exclusive metadata lock on table t1 to commit table definition changes, must wait for the session 1 transaction to commit or roll back.

Session 3:

```
mysql> SELECT * FROM t1;
```

The `SELECT` statement issued in session 3 is blocked waiting for the exclusive metadata lock requested by the `ALTER TABLE` operation in session 2 to be granted.

You can use `SHOW FULL PROCESSLIST` to determine if transactions are waiting for a metadata lock.

```
mysql> SHOW FULL PROCESSLIST\G
...
*************************** 2. row ***************************
     Id: 5
   User: root
   Host: localhost
     db: test
Command: Query
   Time: 44
  State: Waiting for table metadata lock
   Info: ALTER TABLE t1 ADD COLUMN x INT, ALGORITHM=INPLACE, LOCK=NONE
```

```
...
*************************** 4. row ***************************
     Id: 7
   User: root
   Host: localhost
     db: test
Command: Query
   Time: 5
  State: Waiting for table metadata lock
   Info: SELECT * FROM t1
4 rows in set (0.00 sec)
```

Metadata lock information is also exposed through the Performance Schema `metadata_locks` table, which provides information about metadata lock dependencies between sessions, the metadata lock a session is waiting for, and the session that currently holds the metadata lock. For more information, see Section 26.12.12.3, “The metadata_locks Table”.

Online DDL Performance

The performance of a DDL operation is largely determined by whether the operation is performed instantly, in place, and whether it rebuilds the table.

To assess the relative performance of a DDL operation, you can compare results using `ALGORITHM=INSTANT`, `ALGORITHM=INPLACE`, and `ALGORITHM=COPY`. A statement can also be run with `old_alter_table` enabled to force the use of `ALGORITHM=COPY`.

For DDL operations that modify table data, you can determine whether a DDL operation performs changes in place or performs a table copy by looking at the “rows affected” value displayed after the command finishes. For example:

- Changing the default value of a column (fast, does not affect the table data):

  ```
  Query OK, 0 rows affected (0.07 sec)
  ```

- Adding an index (takes time, but `0 rows affected` shows that the table is not copied):

  ```
  Query OK, 0 rows affected (21.42 sec)
  ```

- Changing the data type of a column (takes substantial time and requires rebuilding all the rows of the table):

  ```
  Query OK, 1671168 rows affected (1 min 35.54 sec)
  ```

Before running a DDL operation on a large table, check whether the operation is fast or slow as follows:

1. Clone the table structure.
2. Populate the cloned table with a small amount of data.
3. Run the DDL operation on the cloned table.
4. Check whether the “rows affected” value is zero or not. A nonzero value means the operation copies table data, which might require special planning. For example, you might do the DDL operation during a period of scheduled downtime, or on each replication slave server one at a time.

Note

For a greater understanding of the MySQL processing associated with a DDL operation, examine Performance Schema and `INFORMATION_SCHEMA` tables

> related to `InnoDB` before and after DDL operations to see the number of physical reads, writes, memory allocations, and so on.
>
> Performance Schema stage events can be used to monitor `ALTER TABLE` progress. See Section 15.15.1, "Monitoring ALTER TABLE Progress for InnoDB Tables Using Performance Schema".

Because there is some processing work involved with recording the changes made by concurrent DML operations, then applying those changes at the end, an online DDL operation could take longer overall than the table-copy mechanism that blocks table access from other sessions. The reduction in raw performance is balanced against better responsiveness for applications that use the table. When evaluating the techniques for changing table structure, consider end-user perception of performance, based on factors such as load times for web pages.

15.12.3 Online DDL Space Requirements

Space requirements for in-place online DDL operations are outlined below. Space requirements do not apply to operations that are performed instantly.

- Space for temporary log files

  A temporary log file records concurrent DML when an online DDL operation creates an index or alters a table. The temporary log file is extended as required by the value of `innodb_sort_buffer_size` up to a maximum specified by `innodb_online_alter_log_max_size`. If a temporary log file exceeds the size limit, the online DDL operation fails, and uncommitted concurrent DML operations are rolled back. A large `innodb_online_alter_log_max_size` setting permits more DML during an online DDL operation, but it also extends the period of time at the end of the DDL operation when the table is locked to apply logged DML.

  If the operation takes a long time and concurrent DML modifies the table so much that the size of the temporary log file exceeds the value of `innodb_online_alter_log_max_size`, the online DDL operation fails with a `DB_ONLINE_LOG_TOO_BIG` error.

- Space for temporary sort files

  Online DDL operations that rebuild the table write temporary sort files to the MySQL temporary directory (`$TMPDIR` on Unix, `%TEMP%` on Windows, or the directory specified by `--tmpdir`) during index creation. Temporary sort files are not created in the directory that contains the original table. Each temporary sort file is large enough to hold all secondary index columns plus the primary key columns of the clustered index. Temporary sort files are removed as soon as their contents are merged into the final table or index. Temporary sort files may require space equal to the amount of data in the table plus indexes. An online DDL operation that rebuilds the table reports an error if it uses all of the available disk space on the file system where the data directory resides.

  If the MySQL temporary directory is not large enough to hold the sort files, set `tmpdir` to a different directory. Alternatively, define a separate temporary directory for online DDL operations using `innodb_tmpdir`. This option was introduced to help avoid temporary directory overflows that could occur as a result of large temporary sort files.

- Space for an intermediate table file

  Some online DDL operations that rebuild the table create a temporary intermediate table file in the same directory as the original table. An intermediate table file may require space equal to the size of the original table. Intermediate table file names begin with `#sql-ib` prefix and only appear briefly during the online DDL operation.

The `innodb_tmpdir` option is not applicable to intermediate table files.

15.12.4 Simplifying DDL Statements with Online DDL

Before the introduction of online DDL, it was common practice to combine many DDL operations into a single `ALTER TABLE` statement. Because each `ALTER TABLE` statement involved copying and rebuilding the table, it was more efficient to make several changes to the same table at once, since those changes could all be done with a single rebuild operation for the table. The downside was that SQL code involving DDL operations was harder to maintain and to reuse in different scripts. If the specific changes were different each time, you might have to construct a new complex `ALTER TABLE` for each slightly different scenario.

For DDL operations that can be done online, you can separate them into individual `ALTER TABLE` statements for easier scripting and maintenance, without sacrificing efficiency. For example, you might take a complicated statement such as:

```
ALTER TABLE t1 ADD INDEX i1(c1), ADD UNIQUE INDEX i2(c2),
  CHANGE c4_old_name c4_new_name INTEGER UNSIGNED;
```

and break it down into simpler parts that can be tested and performed independently, such as:

```
ALTER TABLE t1 ADD INDEX i1(c1);
ALTER TABLE t1 ADD UNIQUE INDEX i2(c2);
ALTER TABLE t1 CHANGE c4_old_name c4_new_name INTEGER UNSIGNED NOT NULL;
```

You might still use multi-part `ALTER TABLE` statements for:

- Operations that must be performed in a specific sequence, such as creating an index followed by a foreign key constraint that uses that index.
- Operations all using the same specific `LOCK` clause, that you want to either succeed or fail as a group.
- Operations that cannot be performed online, that is, that still use the table-copy method.
- Operations for which you specify `ALGORITHM=COPY` or `old_alter_table=1`, to force the table-copying behavior if needed for precise backward-compatibility in specialized scenarios.

15.12.5 Online DDL Failure Conditions

The failure of an online DDL operation is typically due to one of the following conditions:

- An `ALGORITHM` clause specifies an algorithm that is not compatible with the particular type of DDL operation or storage engine.
- A `LOCK` clause specifies a low degree of locking (`SHARED` or `NONE`) that is not compatible with the particular type of DDL operation.
- A timeout occurs while waiting for an exclusive lock on the table, which may be needed briefly during the initial and final phases of the DDL operation.
- The `tmpdir` or `innodb_tmpdir` file system runs out of disk space, while MySQL writes temporary sort files on disk during index creation. For more information, see Section 15.12.3, “Online DDL Space Requirements”.
- The operation takes a long time and concurrent DML modifies the table so much that the size of the temporary online log exceeds the value of the `innodb_online_alter_log_max_size` configuration option. This condition causes a `DB_ONLINE_LOG_TOO_BIG` error.

- Concurrent DML makes changes to the table that are allowed with the original table definition, but not with the new one. The operation only fails at the very end, when MySQL tries to apply all the changes from concurrent DML statements. For example, you might insert duplicate values into a column while a unique index is being created, or you might insert `NULL` values into a column while creating a primary key index on that column. The changes made by the concurrent DML take precedence, and the `ALTER TABLE` operation is effectively rolled back.

15.12.6 Online DDL Limitations

The following limitations apply to online DDL operations:

- The table is copied when creating an index on a `TEMPORARY TABLE`.
- The `ALTER TABLE` clause `LOCK=NONE` is not permitted if there are `ON...CASCADE` or `ON...SET NULL` constraints on the table.
- Before an in-place online DDL operation can finish, it must wait for transactions that hold metadata locks on the table to commit or roll back. An online DDL operation may briefly require an exclusive metadata lock on the table during its execution phase, and always requires one in the final phase of the operation when updating the table definition. Consequently, transactions holding metadata locks on the table can cause an online DDL operation to block. The transactions that hold metadata locks on the table may have been started before or during the online DDL operation. A long running or inactive transaction that holds a metadata lock on the table can cause an online DDL operation to timeout.
- When running an in-place online DDL operation, the thread that runs the `ALTER TABLE` statement applies an online log of DML operations that were run concurrently on the same table from other connection threads. When the DML operations are applied, it is possible to encounter a duplicate key entry error (`ERROR 1062 (23000): Duplicate entry`), even if the duplicate entry is only temporary and would be reverted by a later entry in the online log. This is similar to the idea of a foreign key constraint check in `InnoDB` in which constraints must hold during a transaction.
- `OPTIMIZE TABLE` for an `InnoDB` table is mapped to an `ALTER TABLE` operation to rebuild the table and update index statistics and free unused space in the clustered index. Secondary indexes are not created as efficiently because keys are inserted in the order they appeared in the primary key. `OPTIMIZE TABLE` is supported with the addition of online DDL support for rebuilding regular and partitioned `InnoDB` tables.
- Tables created before MySQL 5.6 that include temporal columns (`DATE`, `DATETIME` or `TIMESTAMP`) and have not been rebuilt using `ALGORITHM=COPY` do not support `ALGORITHM=INPLACE`. In this case, an `ALTER TABLE ... ALGORITHM=INPLACE` operation returns the following error:

```
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported.
Reason: Cannot change column type INPLACE. Try ALGORITHM=COPY.
```

- The following limitations are generally applicable to online DDL operations on large tables that involve rebuilding the table:
  - There is no mechanism to pause an online DDL operation or to throttle I/O or CPU usage for an online DDL operation.
  - Rollback of an online DDL operation can be expensive should the operation fail.
  - Long running online DDL operations can cause replication lag. An online DDL operation must finish running on the master before it is run on the slave. Also, DML that was processed concurrently on the master is only processed on the slave after the DDL operation on the slave is completed.

For additional information related to running online DDL operations on large tables, see Section 15.12.2, “Online DDL Performance and Concurrency”.

15.13 InnoDB Startup Options and System Variables

- System variables that are true or false can be enabled at server startup by naming them, or disabled by using a `--skip-` prefix. For example, to enable or disable the `InnoDB` adaptive hash index, you can use `--innodb-adaptive-hash-index` or `--skip-innodb-adaptive-hash-index` on the command line, or `innodb_adaptive_hash_index` or `skip_innodb_adaptive_hash_index` in an option file.
- System variables that take a numeric value can be specified as `--var_name=value` on the command line or as `var_name=value` in option files.
- Many system variables can be changed at runtime (see Section 5.1.9.2, “Dynamic System Variables”).
- For information about `GLOBAL` and `SESSION` variable scope modifiers, refer to the `SET` statement documentation.
- Certain options control the locations and layout of the `InnoDB` data files. Section 15.8.1, “InnoDB Startup Configuration” explains how to use these options.
- Some options, which you might not use initially, help tune `InnoDB` performance characteristics based on machine capacity and your database workload.
- For more information on specifying options and system variables, see Section 4.2.2, “Specifying Program Options”.

Table 15.25 InnoDB Option and Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
daemon_memcached_enable_binlog	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_path	Yes	Yes	Yes		Global	No
daemon_memcached_option	Yes	Yes	Yes		Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes		Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes		Global	No
foreign_key_checks			Yes		Both	Ye
ignore_builtin_innodb	Yes	Yes	Yes		Global	No
innodb	Yes	Yes				
innodb_adaptive_flushing	Yes	Yes	Yes		Global	Ye
innodb_adaptive_flushing_lwm	Yes	Yes	Yes		Global	Ye
innodb_adaptive_hash_index	Yes	Yes	Yes		Global	Ye
innodb_adaptive_hash_index_parts	Yes	Yes	Yes		Global	No
innodb_adaptive_max_sleep_delay	Yes	Yes	Yes		Global	Ye
innodb_api_bk_commit_interval	Yes	Yes	Yes		Global	Ye
innodb_api_disable_rowlock	Yes	Yes	Yes		Global	No
innodb_api_enable_binlog	Yes	Yes	Yes		Global	No
innodb_api_enable_mdl	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
innodb_api_trx_level	Yes	Yes	Yes		Global	Yes
innodb_autoextend_increment	Yes	Yes	Yes		Global	Yes
innodb_autoinc_lock_mode	Yes	Yes	Yes		Global	No
Innodb_available_undo_logs				Yes	Global	No
innodb_background_drop_list_empty	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_bytes_data				Yes	Global	No
Innodb_buffer_pool_bytes_dirty				Yes	Global	No
innodb_buffer_pool_chunk_size	Yes	Yes	Yes		Global	No
innodb_buffer_pool_debug	Yes	Yes	Yes		Global	No
innodb_buffer_pool_dump_at_shutdown	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_now	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_pct	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_dump_status				Yes	Global	No
innodb_buffer_pool_filename	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_in_core_file	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_instances	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_abort	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_load_at_startup	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_now	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_load_status				Yes	Global	No
Innodb_buffer_pool_pages_data				Yes	Global	No
Innodb_buffer_pool_pages_dirty				Yes	Global	No
Innodb_buffer_pool_pages_flushed				Yes	Global	No
Innodb_buffer_pool_pages_free				Yes	Global	No
Innodb_buffer_pool_pages_latched				Yes	Global	No
Innodb_buffer_pool_pages_misc				Yes	Global	No
Innodb_buffer_pool_pages_total				Yes	Global	No
Innodb_buffer_pool_read_ahead				Yes	Global	No
Innodb_buffer_pool_read_ahead_evicted				Yes	Global	No
Innodb_buffer_pool_read_ahead_rnd				Yes	Global	No
Innodb_buffer_pool_read_requests				Yes	Global	No
Innodb_buffer_pool_reads				Yes	Global	No
Innodb_buffer_pool_resize_status				Yes	Global	No
innodb_buffer_pool_size	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_wait_free				Yes	Global	No
Innodb_buffer_pool_write_requests				Yes	Global	No
innodb_change_buffer_max_size	Yes	Yes	Yes		Global	Yes
innodb_change_buffering	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
innodb_change_buffering_debug	Yes	Yes	Yes		Global	Ye
innodb_checkpoint_disabled	Yes	Yes	Yes		Global	Ye
innodb_checksum_algorithm	Yes	Yes	Yes		Global	Ye
innodb_cmp_per_index_enabled	Yes	Yes	Yes		Global	Ye
innodb_commit_concurrency	Yes	Yes	Yes		Global	Ye
innodb_compress_debug	Yes	Yes	Yes		Global	Ye
innodb_compression_failure_threshold_pct	Yes	Yes	Yes		Global	Ye
innodb_compression_level	Yes	Yes	Yes		Global	Ye
innodb_compression_pad_pct_max	Yes	Yes	Yes		Global	Ye
innodb_concurrency_tickets	Yes	Yes	Yes		Global	Ye
innodb_data_file_path	Yes	Yes	Yes		Global	No
Innodb_data_fsyncs				Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes		Global	No
Innodb_data_pending_fsyncs				Yes	Global	No
Innodb_data_pending_reads				Yes	Global	No
Innodb_data_pending_writes				Yes	Global	No
Innodb_data_read				Yes	Global	No
Innodb_data_reads				Yes	Global	No
Innodb_data_writes				Yes	Global	No
Innodb_data_written				Yes	Global	No
Innodb_dblwr_pages_written				Yes	Global	No
Innodb_dblwr_writes				Yes	Global	No
innodb_ddl_log_crash_reset_debug	Yes	Yes	Yes		Global	Ye
innodb_deadlock_detect	Yes	Yes	Yes		Global	Ye
innodb_dedicated_server	Yes	Yes	Yes		Global	No
innodb_default_row_format	Yes	Yes	Yes		Global	Ye
innodb_directories	Yes	Yes	Yes		Global	No
innodb_disable_sort_file_cache	Yes	Yes	Yes		Global	Ye
innodb_doublewrite	Yes	Yes	Yes		Global	No
innodb_fast_shutdown	Yes	Yes	Yes		Global	Ye
innodb_fil_make_page_dirty_debug	Yes	Yes	Yes		Global	Ye
innodb_file_per_table	Yes	Yes	Yes		Global	Ye
innodb_fill_factor	Yes	Yes	Yes		Global	Ye
innodb_flush_log_at_timeout	Yes	Yes	Yes		Global	Ye
innodb_flush_log_at_trx_commit	Yes	Yes	Yes		Global	Ye
innodb_flush_method	Yes	Yes	Yes		Global	No
innodb_flush_neighbors	Yes	Yes	Yes		Global	Ye
innodb_flush_sync	Yes	Yes	Yes		Global	Ye

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
innodb_flushing_avg_loops	Yes	Yes	Yes		Global	Yes
innodb_force_load_corrupted	Yes	Yes	Yes		Global	No
innodb_force_recovery	Yes	Yes	Yes		Global	No
innodb_fsync_threshold	Yes	Yes	Yes		Global	Yes
innodb_ft_aux_table			Yes		Global	Yes
innodb_ft_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_enable_diag_print	Yes	Yes	Yes		Global	Yes
innodb_ft_enable_stopword	Yes	Yes	Yes		Both	Yes
innodb_ft_max_token_size	Yes	Yes	Yes		Global	No
innodb_ft_min_token_size	Yes	Yes	Yes		Global	No
innodb_ft_num_word_optimize	Yes	Yes	Yes		Global	Yes
innodb_ft_result_cache_limit	Yes	Yes	Yes		Global	Yes
innodb_ft_server_stopword_table	Yes	Yes	Yes		Global	Yes
innodb_ft_sort_pll_degree	Yes	Yes	Yes		Global	No
innodb_ft_total_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_user_stopword_table	Yes	Yes	Yes		Both	Yes
Innodb_have_atomic_builtins				Yes	Global	No
innodb_idle_flush_pct	Yes	Yes	Yes		Global	Yes
innodb_io_capacity	Yes	Yes	Yes		Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes		Global	Yes
innodb_limit_optimistic_insert_debug	Yes	Yes	Yes		Global	Yes
innodb_lock_wait_timeout	Yes	Yes	Yes		Both	Yes
innodb_log_buffer_size	Yes	Yes	Yes		Global	Varies
innodb_log_checkpoint_fuzzy_now	Yes	Yes	Yes		Global	Yes
innodb_log_checkpoint_now	Yes	Yes	Yes		Global	Yes
innodb_log_checksums	Yes	Yes	Yes		Global	Yes
innodb_log_compressed_pages	Yes	Yes	Yes		Global	Yes
innodb_log_file_size	Yes	Yes	Yes		Global	No
innodb_log_files_in_group	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir	Yes	Yes	Yes		Global	No
innodb_log_spin_cpu_abs_lwm	Yes	Yes	Yes		Global	Yes
innodb_log_spin_cpu_pct_hwm	Yes	Yes	Yes		Global	Yes
innodb_log_wait_for_flush_spin_hwm	Yes	Yes	Yes		Global	Yes
Innodb_log_waits				Yes	Global	No
innodb_log_write_ahead_size	Yes	Yes	Yes		Global	Yes
Innodb_log_write_requests				Yes	Global	No
Innodb_log_writes				Yes	Global	No
innodb_lru_scan_depth	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
innodb_max_dirty_pages_pct	Yes	Yes	Yes		Global	Ye
innodb_max_dirty_pages_pct_lwm	Yes	Yes	Yes		Global	Ye
innodb_max_purge_lag	Yes	Yes	Yes		Global	Ye
innodb_max_purge_lag_delay	Yes	Yes	Yes		Global	Ye
innodb_max_undo_log_size	Yes	Yes	Yes		Global	Ye
innodb_merge_threshold_set_all_debug	Yes	Yes	Yes		Global	Ye
innodb_monitor_disable	Yes	Yes	Yes		Global	Ye
innodb_monitor_enable	Yes	Yes	Yes		Global	Ye
innodb_monitor_reset	Yes	Yes	Yes		Global	Ye
innodb_monitor_reset_all	Yes	Yes	Yes		Global	Ye
Innodb_num_open_files				Yes	Global	No
innodb_numa_interleave	Yes	Yes	Yes		Global	No
innodb_old_blocks_pct	Yes	Yes	Yes		Global	Ye
innodb_old_blocks_time	Yes	Yes	Yes		Global	Ye
innodb_online_alter_log_max_size	Yes	Yes	Yes		Global	Ye
innodb_open_files	Yes	Yes	Yes		Global	No
innodb_optimize_fulltext_only	Yes	Yes	Yes		Global	Ye
Innodb_os_log_fsyncs				Yes	Global	No
Innodb_os_log_pending_fsyncs				Yes	Global	No
Innodb_os_log_pending_writes				Yes	Global	No
Innodb_os_log_written				Yes	Global	No
innodb_page_cleaners	Yes	Yes	Yes		Global	No
Innodb_page_size				Yes	Global	No
innodb_page_size	Yes	Yes	Yes		Global	No
Innodb_pages_created				Yes	Global	No
Innodb_pages_read				Yes	Global	No
Innodb_pages_written				Yes	Global	No
innodb_parallel_read_threads	Yes	Yes	Yes		Session	Ye
innodb_print_all_deadlocks	Yes	Yes	Yes		Global	Ye
innodb_print_ddl_logs	Yes	Yes	Yes		Global	Ye
innodb_purge_batch_size	Yes	Yes	Yes		Global	Ye
innodb_purge_rseg_truncate_frequency	Yes	Yes	Yes		Global	Ye
innodb_purge_threads	Yes	Yes	Yes		Global	No
innodb_random_read_ahead	Yes	Yes	Yes		Global	Ye
innodb_read_ahead_threshold	Yes	Yes	Yes		Global	Ye
innodb_read_io_threads	Yes	Yes	Yes		Global	No
innodb_read_only	Yes	Yes	Yes		Global	No
innodb_redo_log_archive_dirs	Yes	Yes	Yes		Global	Ye

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
innodb_redo_log_encrypt	Yes	Yes	Yes		Global	Yes
innodb_replication_delay	Yes	Yes	Yes		Global	Yes
innodb_rollback_on_timeout	Yes	Yes	Yes		Global	No
innodb_rollback_segments	Yes	Yes	Yes		Global	Yes
Innodb_row_lock_current_waits				Yes	Global	No
Innodb_row_lock_time				Yes	Global	No
Innodb_row_lock_time_avg				Yes	Global	No
Innodb_row_lock_time_max				Yes	Global	No
Innodb_row_lock_waits				Yes	Global	No
Innodb_rows_deleted				Yes	Global	No
Innodb_rows_inserted				Yes	Global	No
Innodb_rows_read				Yes	Global	No
Innodb_rows_updated				Yes	Global	No
innodb_saved_page_number_debug	Yes	Yes	Yes		Global	Yes
innodb_scan_directories	Yes	Yes	Yes		Global	No
innodb_sort_buffer_size	Yes	Yes	Yes		Global	No
innodb_spin_wait_delay	Yes	Yes	Yes		Global	Yes
innodb_spin_wait_pause_multiplier	Yes	Yes	Yes		Global	Yes
innodb_stats_auto_recalc	Yes	Yes	Yes		Global	Yes
innodb_stats_include_delete_marked	Yes	Yes	Yes		Global	Yes
innodb_stats_method	Yes	Yes	Yes		Global	Yes
innodb_stats_on_metadata	Yes	Yes	Yes		Global	Yes
innodb_stats_persistent	Yes	Yes	Yes		Global	Yes
innodb_stats_persistent_sample_pages	Yes	Yes	Yes		Global	Yes
innodb_stats_transient_sample_pages	Yes	Yes	Yes		Global	Yes
innodb-status-file	Yes	Yes				
innodb_status_output	Yes	Yes	Yes		Global	Yes
innodb_status_output_locks	Yes	Yes	Yes		Global	Yes
innodb_strict_mode	Yes	Yes	Yes		Both	Yes
innodb_sync_array_size	Yes	Yes	Yes		Global	No
innodb_sync_debug	Yes	Yes	Yes		Global	No
innodb_sync_spin_loops	Yes	Yes	Yes		Global	Yes
Innodb_system_rows_deleted				Yes	Global	No
Innodb_system_rows_inserted				Yes	Global	No
Innodb_system_rows_read				Yes	Global	No
innodb_table_locks	Yes	Yes	Yes		Both	Yes
innodb_temp_data_file_path	Yes	Yes	Yes		Global	No
innodb_temp_tablespaces_dir	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
innodb_thread_concurrency	Yes	Yes	Yes		Global	Ye
innodb_thread_sleep_delay	Yes	Yes	Yes		Global	Ye
innodb_tmpdir	Yes	Yes	Yes		Both	Ye
Innodb_truncated_status_writes				Yes	Global	No
innodb_trx_purge_view_update_only_debug	Yes	Yes	Yes		Global	Ye
innodb_trx_rseg_n_slots_debug	Yes	Yes	Yes		Global	Ye
innodb_undo_directory	Yes	Yes	Yes		Global	No
innodb_undo_log_encrypt	Yes	Yes	Yes		Global	Ye
innodb_undo_log_truncate	Yes	Yes	Yes		Global	Ye
innodb_undo_logs	Yes	Yes	Yes		Global	Ye
innodb_undo_tablespaces	Yes	Yes	Yes		Global	Va
innodb_use_native_aio	Yes	Yes	Yes		Global	No
innodb_version			Yes		Global	No
innodb_write_io_threads	Yes	Yes	Yes		Global	No
unique_checks			Yes		Both	Ye

InnoDB Command Options

- `--innodb[=value]`

Property	Value
Command-Line Format	`--innodb[=value]`
Deprecated	Yes
Type	Enumeration
Default Value	`ON`

Valid Values	`OFF`
	`ON`
	`FORCE`

Controls loading of the `InnoDB` storage engine, if the server was compiled with `InnoDB` support. This option has a tristate format, with possible values of `OFF`, `ON`, or `FORCE`. See Section 5.6.1, “Installing and Uninstalling Plugins”.

To disable `InnoDB`, use `--innodb=OFF` or `--skip-innodb`. In this case, because the default storage engine is `InnoDB`, the server does not start unless you also use `--default-storage-engine` and `--default-tmp-storage-engine` to set the default to some other engine for both permanent and `TEMPORARY` tables.

The `InnoDB` storage engine can no longer be disabled, and the `--innodb=OFF` and `--skip-innodb` options are deprecated and have no effect. Their use results in a warning. These options will be removed in a future MySQL release.

- `--innodb-status-file`

Property	Value
Command-Line Format	`--innodb-status-file[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

The `--innodb-status-file` startup option controls whether `InnoDB` creates a file named `innodb_status.pid` in the data directory and writes `SHOW ENGINE INNODB STATUS` output to it every 15 seconds, approximately.

The `innodb_status.pid` file is not created by default. To create it, start `mysqld` with the `--innodb-status-file` option. `InnoDB` removes the file when the server is shut down normally. If an abnormal shutdown occurs, the status file may have to be removed manually.

The `--innodb-status-file` option is intended for temporary use, as `SHOW ENGINE INNODB STATUS` output generation can affect performance, and the `innodb_status.pid` file can become quite large over time.

For related information, see Section 15.16.2, “Enabling InnoDB Monitors”.

- `--skip-innodb`

  Disable the `InnoDB` storage engine. See the description of `--innodb`.

InnoDB System Variables

- `daemon_memcached_enable_binlog`

Property	Value
Command-Line Format	`--daemon-memcached-enable-binlog[={OFF|ON}]`
System Variable	`daemon_memcached_enable_binlog`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enable this option on the master server to use the `InnoDB memcached` plugin (`daemon_memcached`) with the MySQL binary log. This option can only be set at server startup. You must also enable the MySQL binary log on the master server using the `--log-bin` option.

For more information, see Section 15.19.7, “The InnoDB memcached Plugin and Replication”.

- `daemon_memcached_engine_lib_name`

Property	Value
Command-Line Format	`--daemon-memcached-engine-lib-name=file_name`
System Variable	`daemon_memcached_engine_lib_name`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	File name
Default Value	innodb_engine.so

Specifies the shared library that implements the InnoDB memcached plugin.

For more information, see Section 15.19.3, “Setting Up the InnoDB memcached Plugin”.

- daemon_memcached_engine_lib_path

Property	Value
Command-Line Format	--daemon-memcached-engine-lib-path=dir_name
System Variable	daemon_memcached_engine_lib_path
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name
Default Value	NULL

The path of the directory containing the shared library that implements the InnoDB memcached plugin. The default value is NULL, representing the MySQL plugin directory. You should not need to modify this parameter unless specifying a memcached plugin for a different storage engine that is located outside of the MySQL plugin directory.

For more information, see Section 15.19.3, “Setting Up the InnoDB memcached Plugin”.

- daemon_memcached_option

Property	Value
Command-Line Format	--daemon-memcached-option=options
System Variable	daemon_memcached_option
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	

Used to pass space-separated memcached options to the underlying memcached memory object caching daemon on startup. For example, you might change the port that memcached listens on, reduce the maximum number of simultaneous connections, change the maximum memory size for a key-value pair, or enable debugging messages for the error log.

See Section 15.19.3, “Setting Up the InnoDB memcached Plugin” for usage details. For information about memcached options, refer to the memcached man page.

- daemon_memcached_r_batch_size

Property	Value
Command-Line Format	`--daemon-memcached-r-batch-size=#`
System Variable	`daemon_memcached_r_batch_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`

Specifies how many `memcached` read operations (`get` operations) to perform before doing a `COMMIT` to start a new transaction. Counterpart of `daemon_memcached_w_batch_size`.

This value is set to 1 by default, so that any changes made to the table through SQL statements are immediately visible to `memcached` operations. You might increase it to reduce the overhead from frequent commits on a system where the underlying table is only being accessed through the `memcached` interface. If you set the value too large, the amount of undo or redo data could impose some storage overhead, as with any long-running transaction.

For more information, see Section 15.19.3, “Setting Up the InnoDB memcached Plugin”.

- `daemon_memcached_w_batch_size`

Property	Value
Command-Line Format	`--daemon-memcached-w-batch-size=#`
System Variable	`daemon_memcached_w_batch_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`

Specifies how many `memcached` write operations, such as `add`, `set`, and `incr`, to perform before doing a `COMMIT` to start a new transaction. Counterpart of `daemon_memcached_r_batch_size`.

This value is set to 1 by default, on the assumption that data being stored is important to preserve in case of an outage and should immediately be committed. When storing non-critical data, you might increase this value to reduce the overhead from frequent commits; but then the last *`N`*-1 uncommitted write operations could be lost if a crash occurs.

For more information, see Section 15.19.3, “Setting Up the InnoDB memcached Plugin”.

- `ignore_builtin_innodb`

Property	Value
Command-Line Format	`--ignore-builtin-innodb[={OFF|ON}]`
Deprecated	Yes (removed in 8.0.3)
System Variable	`ignore_builtin_innodb`

Property	Value
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean

In earlier versions of MySQL, enabling this variable caused the server to behave as if the built-in `InnoDB` were not present, which enabled the `InnoDB Plugin` to be used instead. In MySQL 8.0, `InnoDB` is the default storage engine and `InnoDB Plugin` is not used. This variable was removed in MySQL 8.0.

- `innodb_adaptive_flushing`

Property	Value
Command-Line Format	`--innodb-adaptive-flushing[={OFF|ON}]`
System Variable	`innodb_adaptive_flushing`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Specifies whether to dynamically adjust the rate of flushing dirty pages in the `InnoDB` buffer pool based on the workload. Adjusting the flush rate dynamically is intended to avoid bursts of I/O activity. This setting is enabled by default. See Section 15.8.3.5, “Configuring Buffer Pool Flushing” for more information. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_adaptive_flushing_lwm`

Property	Value
Command-Line Format	`--innodb-adaptive-flushing-lwm=#`
System Variable	`innodb_adaptive_flushing_lwm`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value	`70`

Defines the low water mark representing percentage of redo log capacity at which adaptive flushing is enabled. For more information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”.

- `innodb_adaptive_hash_index`

Property	Value
Command-Line Format	`--innodb-adaptive-hash-index[={OFF|ON}]`
System Variable	`innodb_adaptive_hash_index`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether the `InnoDB` adaptive hash index is enabled or disabled. It may be desirable, depending on your workload, to dynamically enable or disable adaptive hash indexing to improve query performance. Because the adaptive hash index may not be useful for all workloads, conduct benchmarks with it both enabled and disabled, using realistic workloads. See Section 15.5.3, “Adaptive Hash Index” for details.

This variable is enabled by default. You can modify this parameter using the `SET GLOBAL` statement, without restarting the server. Changing the setting at runtime requires privileges sufficient to set global system variables. See Section 5.1.9.1, “System Variable Privileges”. You can also use `--skip-innodb-adaptive-hash-index` at server startup to disable it.

Disabling the adaptive hash index empties the hash table immediately. Normal operations can continue while the hash table is emptied, and executing queries that were using the hash table access the index B-trees directly instead. When the adaptive hash index is re-enabled, the hash table is populated again during normal operation.

- `innodb_adaptive_hash_index_parts`

Property	Value
Command-Line Format	`--innodb-adaptive-hash-index-parts=#`
System Variable	`innodb_adaptive_hash_index_parts`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Numeric
Default Value	`8`
Minimum Value	`1`
Maximum Value	`512`

Partitions the adaptive hash index search system. Each index is bound to a specific partition, with each partition protected by a separate latch.

The adaptive hash index search system is partitioned into 8 parts by default. The maximum setting is 512.

For related information, see Section 15.5.3, “Adaptive Hash Index”.

- `innodb_adaptive_max_sleep_delay`

Property	Value
Command-Line Format	`--innodb-adaptive-max-sleep-delay=#`
System Variable	`innodb_adaptive_max_sleep_delay`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`150000`
Minimum Value	`0`
Maximum Value	`1000000`

Permits `InnoDB` to automatically adjust the value of `innodb_thread_sleep_delay` up or down according to the current workload. Any nonzero value enables automated, dynamic adjustment of the `innodb_thread_sleep_delay` value, up to the maximum value specified in the `innodb_adaptive_max_sleep_delay` option. The value represents the number of microseconds. This option can be useful in busy systems, with greater than 16 `InnoDB` threads. (In practice, it is most valuable for MySQL systems with hundreds or thousands of simultaneous connections.)

For more information, see Section 15.8.4, “Configuring Thread Concurrency for InnoDB”.

- `innodb_api_bk_commit_interval`

Property	Value
Command-Line Format	`--innodb-api-bk-commit-interval=#`
System Variable	`innodb_api_bk_commit_interval`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`5`
Minimum Value	`1`
Maximum Value	`1073741824`

How often to auto-commit idle connections that use the `InnoDB memcached` interface, in seconds. For more information, see Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- `innodb_api_disable_rowlock`

Property	Value
Command-Line Format	`--innodb-api-disable-rowlock[={OFF|ON}]`
System Variable	`innodb_api_disable_rowlock`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Use this option to disable row locks when InnoDB memcached performs DML operations. By default, innodb_api_disable_rowlock is disabled, which means that memcached requests row locks for get and set operations. When innodb_api_disable_rowlock is enabled, memcached requests a table lock instead of row locks.

innodb_api_disable_rowlock is not dynamic. It must be specified on the mysqld command line or entered in the MySQL configuration file. Configuration takes effect when the plugin is installed, which occurs when the MySQL server is started.

For more information, see Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- innodb_api_enable_binlog

Property	Value
Command-Line Format	--innodb-api-enable-binlog[={OFF|ON}]
System Variable	innodb_api_enable_binlog
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Lets you use the InnoDB memcached plugin with the MySQL binary log. For more information, see Enabling the InnoDB memcached Binary Log.

- innodb_api_enable_mdl

Property	Value
Command-Line Format	--innodb-api-enable-mdl[={OFF|ON}]
System Variable	innodb_api_enable_mdl
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Locks the table used by the InnoDB memcached plugin, so that it cannot be dropped or altered by DDL through the SQL interface. For more information, see Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- innodb_api_trx_level

Property	Value
Command-Line Format	`--innodb-api-trx-level=#`
System Variable	`innodb_api_trx_level`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`

Controls the transaction isolation level on queries processed by the `memcached` interface. The constants corresponding to the familiar names are:

- 0 = `READ UNCOMMITTED`
- 1 = `READ COMMITTED`
- 2 = `REPEATABLE READ`
- 3 = `SERIALIZABLE`

For more information, see Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- `innodb_autoextend_increment`

Property	Value
Command-Line Format	`--innodb-autoextend-increment=#`
System Variable	`innodb_autoextend_increment`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`64`
Minimum Value	`1`
Maximum Value	`1000`

The increment size (in megabytes) for extending the size of an auto-extending `InnoDB` system tablespace file when it becomes full. The default value is 64. For related information, see System Tablespace Data File Configuration, and Resizing the System Tablespace.

The `innodb_autoextend_increment` setting does not affect file-per-table tablespace files or general tablespace files. These files are auto-extending regardless of the `innodb_autoextend_increment` setting. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

- `innodb_autoinc_lock_mode`

Property	Value
Command-Line Format	`--innodb-autoinc-lock-mode=#`

Property	Value
System Variable	`innodb_autoinc_lock_mode`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.3)	`2`
Default Value (<= 8.0.2)	`1`

Valid Values	`0`
	`1`
	`2`

The lock mode to use for generating auto-increment values. Permissible values are 0, 1, or 2, for traditional, consecutive, or interleaved, respectively.

The default setting is 2 (interleaved) as of MySQL 8.0, and 1 (consecutive) before that. The change to interleaved lock mode as the default setting reflects the change from statement-based to row-based replication as the default replication type, which occurred in MySQL 5.7. Statement-based replication requires the consecutive auto-increment lock mode to ensure that auto-increment values are assigned in a predictable and repeatable order for a given sequence of SQL statements, whereas row-based replication is not sensitive to the execution order of SQL statements.

For the characteristics of each lock mode, see InnoDB AUTO_INCREMENT Lock Modes.

- `innodb_background_drop_list_empty`

Property	Value
Command-Line Format	`--innodb-background-drop-list-empty[={OFF|ON}]`
System Variable	`innodb_background_drop_list_empty`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enabling the `innodb_background_drop_list_empty` debug option helps avoid test case failures by delaying table creation until the background drop list is empty. For example, if test case A places table `t1` on the background drop list, test case B waits until the background drop list is empty before creating table `t1`.

- `innodb_buffer_pool_chunk_size`

Property	Value
Command-Line Format	`--innodb-buffer-pool-chunk-size=#`
System Variable	`innodb_buffer_pool_chunk_size`
Scope	Global

Property	Value
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	134217728
Minimum Value	1048576
Maximum Value	innodb_buffer_pool_size / innodb_buffer_pool_instances

innodb_buffer_pool_chunk_size defines the chunk size for InnoDB buffer pool resizing operations. The innodb_buffer_pool_size parameter is dynamic, which allows you to resize the buffer pool without restarting the server.

To avoid copying all buffer pool pages during resizing operations, the operation is performed in “chunks”. By default, innodb_buffer_pool_chunk_size is 128MB (134217728 bytes). The number of pages contained in a chunk depends on the value of innodb_page_size. innodb_buffer_pool_chunk_size can be increased or decreased in units of 1MB (1048576 bytes).

The following conditions apply when altering the innodb_buffer_pool_chunk_size value:

- If innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances is larger than the current buffer pool size when the buffer pool is initialized, innodb_buffer_pool_chunk_size is truncated to innodb_buffer_pool_size / innodb_buffer_pool_instances.
- Buffer pool size must always be equal to or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances. If you alter innodb_buffer_pool_chunk_size, innodb_buffer_pool_size is automatically rounded to a value that is equal to or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances. The adjustment occurs when the buffer pool is initialized.

Important

Care should be taken when changing innodb_buffer_pool_chunk_size, as changing this value can automatically increase the size of the buffer pool. Before changing innodb_buffer_pool_chunk_size, calculate the effect it will have on innodb_buffer_pool_size to ensure that the resulting buffer pool size is acceptable.

To avoid potential performance issues, the number of chunks (innodb_buffer_pool_size / innodb_buffer_pool_chunk_size) should not exceed 1000.

See Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size” for more information.

- innodb_buffer_pool_debug

Property	Value
Command-Line Format	--innodb-buffer-pool-debug[={OFF|ON}]
System Variable	innodb_buffer_pool_debug
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

Property	Value
Type	Boolean
Default Value	`OFF`

Enabling this option permits multiple buffer pool instances when the buffer pool is less than 1GB in size, ignoring the 1GB minimum buffer pool size constraint imposed on `innodb_buffer_pool_instances`. The `innodb_buffer_pool_debug` option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_buffer_pool_dump_at_shutdown`

Property	Value
Command-Line Format	`--innodb-buffer-pool-dump-at-shutdown[={OFF|ON}]`
System Variable	`innodb_buffer_pool_dump_at_shutdown`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Specifies whether to record the pages cached in the `InnoDB` buffer pool when the MySQL server is shut down, to shorten the warmup process at the next restart. Typically used in combination with `innodb_buffer_pool_load_at_startup`. The `innodb_buffer_pool_dump_pct` option defines the percentage of most recently used buffer pool pages to dump.

Both `innodb_buffer_pool_dump_at_shutdown` and `innodb_buffer_pool_load_at_startup` are enabled by default.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_dump_now`

Property	Value
Command-Line Format	`--innodb-buffer-pool-dump-now[={OFF|ON}]`
System Variable	`innodb_buffer_pool_dump_now`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Immediately records the pages cached in the `InnoDB` buffer pool. Typically used in combination with `innodb_buffer_pool_load_now`.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_dump_pct`

Property	Value
Command-Line Format	`--innodb-buffer-pool-dump-pct=#`
System Variable	`innodb_buffer_pool_dump_pct`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`25`
Minimum Value	`1`
Maximum Value	`100`

Specifies the percentage of the most recently used pages for each buffer pool to read out and dump. The range is 1 to 100. The default value is 25. For example, if there are 4 buffer pools with 100 pages each, and `innodb_buffer_pool_dump_pct` is set to 25, the 25 most recently used pages from each buffer pool are dumped.

- `innodb_buffer_pool_filename`

Property	Value
Command-Line Format	`--innodb-buffer-pool-filename=file_name`
System Variable	`innodb_buffer_pool_filename`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	`ib_buffer_pool`

Specifies the name of the file that holds the list of tablespace IDs and page IDs produced by `innodb_buffer_pool_dump_at_shutdown` or `innodb_buffer_pool_dump_now`. Tablespace IDs and page IDs are saved in the following format: `space, page_id`. By default, the file is named `ib_buffer_pool` and is located in the `InnoDB` data directory. A non-default location must be specified relative to the data directory.

A file name can be specified at runtime, using a `SET` statement:

```
SET GLOBAL innodb_buffer_pool_filename='file_name';
```

You can also specify a file name at startup, in a startup string or MySQL configuration file. When specifying a file name at startup, the file must exist or `InnoDB` will return a startup error indicating that there is no such file or directory.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_in_core_file`

Property	Value
Command-Line Format	`--innodb-buffer-pool-in-core-file[={OFF|ON}]`

Property	Value
Introduced	8.0.14
System Variable	`innodb_buffer_pool_in_core_file`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Disabling the `innodb_buffer_pool_in_core_file` variable reduces the size of core files by excluding `InnoDB` buffer pool pages. To use this variable, the `core_file` variable must be enabled and the operating system must support the `MADV_DONTDUMP` non-POSIX extension to `madvise()`, which is supported in Linux 3.4 and later. For more information, see Section 15.8.3.7, "Excluding Buffer Pool Pages from Core Files".

- `innodb_buffer_pool_instances`

Property	Value
Command-Line Format	`--innodb-buffer-pool-instances=#`
System Variable	`innodb_buffer_pool_instances`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value (Other)	`8 (or 1 if innodb_buffer_pool_size < 1GB`
Default Value (Windows, 32-bit platforms)	`(autosized)`
Minimum Value	`1`
Maximum Value	`64`

The number of regions that the `InnoDB` buffer pool is divided into. For systems with buffer pools in the multi-gigabyte range, dividing the buffer pool into separate instances can improve concurrency, by reducing contention as different threads read and write to cached pages. Each page that is stored in or read from the buffer pool is assigned to one of the buffer pool instances randomly, using a hashing function. Each buffer pool manages its own free lists, flush lists, LRUs, and all other data structures connected to a buffer pool, and is protected by its own buffer pool mutex.

This option only takes effect when setting `innodb_buffer_pool_size` to 1GB or more. The total buffer pool size is divided among all the buffer pools. For best efficiency, specify a combination of `innodb_buffer_pool_instances` and `innodb_buffer_pool_size` so that each buffer pool instance is at least 1GB.

The default value on 32-bit Windows systems depends on the value of `innodb_buffer_pool_size`, as described below:

- If `innodb_buffer_pool_size` is greater than 1.3GB, the default for `innodb_buffer_pool_instances` is `innodb_buffer_pool_size`/128MB, with individual memory allocation requests for each chunk. 1.3GB was chosen as the boundary at which there is

significant risk for 32-bit Windows to be unable to allocate the contiguous address space needed for a single buffer pool.

- Otherwise, the default is 1.

On all other platforms, the default value is 8 when `innodb_buffer_pool_size` is greater than or equal to 1GB. Otherwise, the default is 1.

For related information, see Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”.

- `innodb_buffer_pool_load_abort`

Property	Value
Command-Line Format	`--innodb-buffer-pool-load-abort[={OFF|ON}]`
System Variable	`innodb_buffer_pool_load_abort`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Interrupts the process of restoring `InnoDB` buffer pool contents triggered by `innodb_buffer_pool_load_at_startup` or `innodb_buffer_pool_load_now`.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_load_at_startup`

Property	Value
Command-Line Format	`--innodb-buffer-pool-load-at-startup[={OFF|ON}]`
System Variable	`innodb_buffer_pool_load_at_startup`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Specifies that, on MySQL server startup, the `InnoDB` buffer pool is automatically warmed up by loading the same pages it held at an earlier time. Typically used in combination with `innodb_buffer_pool_dump_at_shutdown`.

Both `innodb_buffer_pool_dump_at_shutdown` and `innodb_buffer_pool_load_at_startup` are enabled by default.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_load_now`

Property	Value
Command-Line Format	`--innodb-buffer-pool-load-now[={OFF|ON}]`

Property	Value
System Variable	`innodb_buffer_pool_load_now`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Immediately warms up the `InnoDB` buffer pool by loading a set of data pages, without waiting for a server restart. Can be useful to bring cache memory back to a known state during benchmarking, or to ready the MySQL server to resume its normal workload after running queries for reports or maintenance.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_size`

Property	Value
Command-Line Format	`--innodb-buffer-pool-size=#`
System Variable	`innodb_buffer_pool_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`134217728`
Minimum Value	`5242880`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`2**32-1`

The size in bytes of the buffer pool, the memory area where `InnoDB` caches table and index data. The default value is 134217728 bytes (128MB). The maximum value depends on the CPU architecture; the maximum is 4294967295 (2^{32}-1) on 32-bit systems and 18446744073709551615 (2^{64}-1) on 64-bit systems. On 32-bit systems, the CPU architecture and operating system may impose a lower practical maximum size than the stated maximum. When the size of the buffer pool is greater than 1GB, setting `innodb_buffer_pool_instances` to a value greater than 1 can improve the scalability on a busy server.

A larger buffer pool requires less disk I/O to access the same table data more than once. On a dedicated database server, you might set the buffer pool size to 80% of the machine's physical memory size. Be aware of the following potential issues when configuring buffer pool size, and be prepared to scale back the size of the buffer pool if necessary.

- Competition for physical memory can cause paging in the operating system.
- `InnoDB` reserves additional memory for buffers and control structures, so that the total allocated space is approximately 10% greater than the specified buffer pool size.

- Address space for the buffer pool must be contiguous, which can be an issue on Windows systems with DLLs that load at specific addresses.
- The time to initialize the buffer pool is roughly proportional to its size. On instances with large buffer pools, initialization time might be significant. To reduce the initialization period, you can save the buffer pool state at server shutdown and restore it at server startup. See Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

When you increase or decrease buffer pool size, the operation is performed in chunks. Chunk size is defined by the `innodb_buffer_pool_chunk_size` variable, which has a default of 128 MB.

Buffer pool size must always be equal to or a multiple of `innodb_buffer_pool_chunk_size` * `innodb_buffer_pool_instances`. If you alter the buffer pool size to a value that is not equal to or a multiple of `innodb_buffer_pool_chunk_size` * `innodb_buffer_pool_instances`, buffer pool size is automatically adjusted to a value that is equal to or a multiple of `innodb_buffer_pool_chunk_size` * `innodb_buffer_pool_instances`.

`innodb_buffer_pool_size` can be set dynamically, which allows you to resize the buffer pool without restarting the server. The `Innodb_buffer_pool_resize_status` status variable reports the status of online buffer pool resizing operations. See Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size” for more information.

If `innodb_dedicated_server` is enabled, the `innodb_buffer_pool_size` value is automatically configured if it is not explicitly defined. For more information, see Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

- `innodb_change_buffer_max_size`

Property	Value
Command-Line Format	`--innodb-change-buffer-max-size=#`
System Variable	`innodb_change_buffer_max_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`25`
Minimum Value	`0`
Maximum Value	`50`

Maximum size for the `InnoDB` change buffer, as a percentage of the total size of the buffer pool. You might increase this value for a MySQL server with heavy insert, update, and delete activity, or decrease it for a MySQL server with unchanging data used for reporting. For more information, see Section 15.5.2, “Change Buffer”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_change_buffering`

Property	Value
Command-Line Format	`--innodb-change-buffering=value`
System Variable	`innodb_change_buffering`

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`all`

Valid Values	`none`
	`inserts`
	`deletes`
	`changes`
	`purges`
	`all`

Whether `InnoDB` performs change buffering, an optimization that delays write operations to secondary indexes so that the I/O operations can be performed sequentially. Permitted values are described in the following table. Values may also be specified numerically.

Table 15.26 Permitted Values for innodb_change_buffering

Value	Numeric Value	Description
`none`	`0`	Do not buffer any operations.
`inserts`	`1`	Buffer insert operations.
`deletes`	`2`	Buffer delete marking operations; strictly speaking, the writes that mark index records for later deletion during a purge operation.
`changes`	`3`	Buffer inserts and delete-marking operations.
`purges`	`4`	Buffer the physical deletion operations that happen in the background.
`all`	`5`	The default. Buffer inserts, delete-marking operations, and purges.

For more information, see Section 15.5.2, “Change Buffer”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_change_buffering_debug`

Property	Value
Command-Line Format	`--innodb-change-buffering-debug=#`
System Variable	`innodb_change_buffering_debug`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value	`0`
Maximum Value	`2`

Sets a debug flag for `InnoDB` change buffering. A value of 1 forces all changes to the change buffer. A value of 2 causes a crash at merge. A default value of 0 indicates that the change buffering debug flag is not set. This option is only available when debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_checkpoint_disabled`

Property	Value
Command-Line Format	`--innodb-checkpoint-disabled[={OFF|ON}]`
Introduced	8.0.2
System Variable	`innodb_checkpoint_disabled`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`OFF`

This is a debug option that is only intended for expert debugging use. It disables checkpoints so that a deliberate server exit always initiates `InnoDB` recovery. It should only be enabled for a short interval, typically before running DML operations that write redo log entries that would require recovery following a server exit. This option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_checksum_algorithm`

Property	Value
Command-Line Format	`--innodb-checksum-algorithm=value`
System Variable	`innodb_checksum_algorithm`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`crc32`

Valid Values	`innodb`
	`crc32`
	`none`
	`strict_innodb`
	`strict_crc32`
	`strict_none`

Specifies how to generate and verify the checksum stored in the disk blocks of InnoDB tablespaces. The default value for innodb_checksum_algorithm is crc32.

Versions of MySQL Enterprise Backup up to 3.8.0 do not support backing up tablespaces that use CRC32 checksums. MySQL Enterprise Backup adds CRC32 checksum support in 3.8.1, with some limitations. Refer to the MySQL Enterprise Backup 3.8.1 Change History for more information.

The value innodb is backward-compatible with earlier versions of MySQL. The value crc32 uses an algorithm that is faster to compute the checksum for every modified block, and to check the checksums for each disk read. It scans blocks 32 bits at a time, which is faster than the innodb checksum algorithm, which scans blocks 8 bits at a time. The value none writes a constant value in the checksum field rather than computing a value based on the block data. The blocks in a tablespace can use a mix of old, new, and no checksum values, being updated gradually as the data is modified; once blocks in a tablespace are modified to use the crc32 algorithm, the associated tables cannot be read by earlier versions of MySQL.

The strict form of a checksum algorithm reports an error if it encounters a valid but non-matching checksum value in a tablespace. It is recommended that you only use strict settings in a new instance, to set up tablespaces for the first time. Strict settings are somewhat faster, because they do not need to compute all checksum values during disk reads.

The following table shows the difference between the none, innodb, and crc32 option values, and their strict counterparts. none, innodb, and crc32 write the specified type of checksum value into each data block, but for compatibility accept other checksum values when verifying a block during a read operation. Strict settings also accept valid checksum values but print an error message when a valid non-matching checksum value is encountered. Using the strict form can make verification faster if all InnoDB data files in an instance are created under an identical innodb_checksum_algorithm value.

Table 15.27 Permitted innodb_checksum_algorithm Values

Value	Generated checksum (when writing)	Permitted checksums (when reading)
none	A constant number.	Any of the checksums generated by none, innodb, or crc32.
innodb	A checksum calculated in software, using the original algorithm from InnoDB.	Any of the checksums generated by none, innodb, or crc32.
crc32	A checksum calculated using the crc32 algorithm, possibly done with a hardware assist.	Any of the checksums generated by none, innodb, or crc32.
strict_none	A constant number	Any of the checksums generated by none, innodb, or crc32. InnoDB prints an error message if a valid but non-matching checksum is encountered.
strict_innodb	A checksum calculated in software, using the original algorithm from InnoDB.	Any of the checksums generated by none, innodb, or crc32. InnoDB prints an error message if a valid but non-matching checksum is encountered.
strict_crc32	A checksum calculated using the crc32 algorithm, possibly done with a hardware assist.	Any of the checksums generated by none, innodb, or crc32. InnoDB prints an error message if a valid but non-matching checksum is encountered.

- `innodb_cmp_per_index_enabled`

Property	Value
Command-Line Format	`--innodb-cmp-per-index-enabled[={OFF|ON}]`
System Variable	`innodb_cmp_per_index_enabled`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enables per-index compression-related statistics in the `INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX` table. Because these statistics can be expensive to gather, only enable this option on development, test, or slave instances during performance tuning related to `InnoDB` compressed tables.

For more information, see Section 25.39.7, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”, and Section 15.9.1.4, “Monitoring InnoDB Table Compression at Runtime”.

- `innodb_commit_concurrency`

Property	Value
Command-Line Format	`--innodb-commit-concurrency=#`
System Variable	`innodb_commit_concurrency`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1000`

The number of threads that can commit at the same time. A value of 0 (the default) permits any number of transactions to commit simultaneously.

The value of `innodb_commit_concurrency` cannot be changed at runtime from zero to nonzero or vice versa. The value can be changed from one nonzero value to another.

- `innodb_compress_debug`

Property	Value
Command-Line Format	`--innodb-compress-debug=value`
System Variable	`innodb_compress_debug`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`none`

Valid Values	`none`
	`zlib`
	`lz4`
	`lz4hc`

Compresses all tables using a specified compression algorithm without having to define a `COMPRESSION` attribute for each table. This option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option.

For related information, see Section 15.9.2, “InnoDB Page Compression”.

- `innodb_compression_failure_threshold_pct`

Property	Value
Command-Line Format	`--innodb-compression-failure-threshold-pct=#`
System Variable	`innodb_compression_failure_threshold_pct`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`5`
Minimum Value	`0`
Maximum Value	`100`

Defines the compression failure rate threshold for a table, as a percentage, at which point MySQL begins adding padding within compressed pages to avoid expensive compression failures. When this threshold is passed, MySQL begins to leave additional free space within each new compressed page, dynamically adjusting the amount of free space up to the percentage of page size specified by `innodb_compression_pad_pct_max`. A value of zero disables the mechanism that monitors compression efficiency and dynamically adjusts the padding amount.

For more information, see Section 15.9.1.6, “Compression for OLTP Workloads”.

- `innodb_compression_level`

Property	Value
Command-Line Format	`--innodb-compression-level=#`
System Variable	`innodb_compression_level`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	6
Minimum Value	0
Maximum Value	9

Specifies the level of zlib compression to use for InnoDB compressed tables and indexes. A higher value lets you fit more data onto a storage device, at the expense of more CPU overhead during compression. A lower value lets you reduce CPU overhead when storage space is not critical, or you expect the data is not especially compressible.

For more information, see Section 15.9.1.6, “Compression for OLTP Workloads”.

- innodb_compression_pad_pct_max

Property	Value
Command-Line Format	--innodb-compression-pad-pct-max=#
System Variable	innodb_compression_pad_pct_max
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	50
Minimum Value	0
Maximum Value	75

Specifies the maximum percentage that can be reserved as free space within each compressed page, allowing room to reorganize the data and modification log within the page when a compressed table or index is updated and the data might be recompressed. Only applies when innodb_compression_failure_threshold_pct is set to a nonzero value, and the rate of compression failures passes the cutoff point.

For more information, see Section 15.9.1.6, “Compression for OLTP Workloads”.

- innodb_concurrency_tickets

Property	Value
Command-Line Format	--innodb-concurrency-tickets=#
System Variable	innodb_concurrency_tickets
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	5000
Minimum Value	1
Maximum Value	4294967295

Determines the number of threads that can enter `InnoDB` concurrently. A thread is placed in a queue when it tries to enter `InnoDB` if the number of threads has already reached the concurrency limit. When a thread is permitted to enter `InnoDB`, it is given a number of " tickets" equal to the value of `innodb_concurrency_tickets`, and the thread can enter and leave `InnoDB` freely until it has used up its tickets. After that point, the thread again becomes subject to the concurrency check (and possible queuing) the next time it tries to enter `InnoDB`. The default value is 5000.

With a small `innodb_concurrency_tickets` value, small transactions that only need to process a few rows compete fairly with larger transactions that process many rows. The disadvantage of a small `innodb_concurrency_tickets` value is that large transactions must loop through the queue many times before they can complete, which extends the amount of time required to complete their task.

With a large `innodb_concurrency_tickets` value, large transactions spend less time waiting for a position at the end of the queue (controlled by `innodb_thread_concurrency`) and more time retrieving rows. Large transactions also require fewer trips through the queue to complete their task. The disadvantage of a large `innodb_concurrency_tickets` value is that too many large transactions running at the same time can starve smaller transactions by making them wait a longer time before executing.

With a nonzero `innodb_thread_concurrency` value, you may need to adjust the `innodb_concurrency_tickets` value up or down to find the optimal balance between larger and smaller transactions. The `SHOW ENGINE INNODB STATUS` report shows the number of tickets remaining for an executing transaction in its current pass through the queue. This data may also be obtained from the `TRX_CONCURRENCY_TICKETS` column of the `INFORMATION_SCHEMA.INNODB_TRX` table.

For more information, see Section 15.8.4, "Configuring Thread Concurrency for InnoDB".

- `innodb_data_file_path`

Property	Value
Command-Line Format	`--innodb-data-file-path=file_name`
System Variable	`innodb_data_file_path`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`ibdata1:12M:autoextend`

Defines the name, size, and attributes of `InnoDB` system tablespace data files. If you do not specify a value for `innodb_data_file_path`, the default behavior is to create a single auto-extending data file, slightly larger than 12MB, named `ibdata1`.

The full syntax for a data file specification includes the file name, file size, `autoextend` attribute, and `max` attribute:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

File sizes are specified in kilobytes, megabytes, or gigabytes by appending `K`, `M` or `G` to the size value. If specifying the data file size in kilobytes, do so in multiples of 1024. Otherwise, KB values are rounded

to nearest megabyte (MB) boundary. The sum of file sizes must be, at a minimum, slightly larger than 12MB.

For additional configuration information, see System Tablespace Data File Configuration. For resizing instructions, see Resizing the System Tablespace.

- `innodb_data_home_dir`

Property	Value
Command-Line Format	`--innodb-data-home-dir=dir_name`
System Variable	`innodb_data_home_dir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

The common part of the directory path for `InnoDB` system tablespace data files. This setting does not affect the location of file-per-table tablespaces when `innodb_file_per_table` is enabled. The default value is the MySQL `data` directory. If you specify the value as an empty string, you can specify an absolute file paths for `innodb_data_file_path`.

A trailing slash is required when specifying a value for `innodb_data_home_dir`. For example:

```
[mysqld]
innodb_data_home_dir = /path/to/myibdata/
```

For related information, see Section 15.8.1, “InnoDB Startup Configuration”.

- `innodb_ddl_log_crash_reset_debug`

Property	Value
Command-Line Format	`--innodb-ddl-log-crash-reset-debug[={OFF|ON}]`
Introduced	8.0.3
System Variable	`innodb_ddl_log_crash_reset_debug`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enable this debug option to reset DDL log crash injection counters to 1. This option is only available when debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_deadlock_detect`

Property	Value
Command-Line Format	`--innodb-deadlock-detect[={OFF|ON}]`
System Variable	`innodb_deadlock_detect`

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

This option is used to disable deadlock detection. On high concurrency systems, deadlock detection can cause a slowdown when numerous threads wait for the same lock. At times, it may be more efficient to disable deadlock detection and rely on the innodb_lock_wait_timeout setting for transaction rollback when a deadlock occurs.

For related information, see Section 15.7.5.2, “Deadlock Detection and Rollback”.

- innodb_dedicated_server

Property	Value
Command-Line Format	--innodb-dedicated-server[={OFF|ON}]
Introduced	8.0.3
System Variable	innodb_dedicated_server
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

When innodb_dedicated_server is enabled, InnoDB automatically configures the following options according to the amount of memory detected on the server:

- innodb_buffer_pool_size
- innodb_log_file_size
- innodb_flush_method

Only consider enabling this option if your MySQL instance runs on a dedicated server where the MySQL server is able to consume all available system resources. Enabling this option is not recommended if your MySQL instance shares system resources with other applications.

For more information, see Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

- innodb_default_row_format

Property	Value
Command-Line Format	--innodb-default-row-format=value
System Variable	innodb_default_row_format
Scope	Global

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	DYNAMIC
Valid Values	DYNAMIC COMPACT REDUNDANT

The innodb_default_row_format option defines the default row format for InnoDB tables and user-created temporary tables. The default setting is DYNAMIC. Other permitted values are COMPACT and REDUNDANT. The COMPRESSED row format, which is not supported for use in the system tablespace, cannot be defined as the default.

Newly created tables use the row format defined by innodb_default_row_format when a ROW_FORMAT option is not specified explicitly or when ROW_FORMAT=DEFAULT is used.

When a ROW_FORMAT option is not specified explicitly or when ROW_FORMAT=DEFAULT is used, any operation that rebuilds a table also silently changes the row format of the table to the format defined by innodb_default_row_format. For more information, see Defining the Row Format of a Table.

Internal InnoDB temporary tables created by the server to process queries use the DYNAMIC row format, regardless of the innodb_default_row_format setting.

- innodb_directories

Property	Value
Command-Line Format	--innodb-directories=dir_name
Introduced	8.0.4
System Variable	innodb_directories
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

Defines directories to scan at startup for tablespace files. This option is used when moving or restoring tablespace files to a new location while the server is offline. It is also used to specify directories of tablespace files created using an absolute path or that reside outside of the data directory (see Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”).

Tablespace discovery during crash recovery relies on the innodb_directories setting to identify tablespaces referenced in the redo logs. For more information, see Tablespace Discovery During Crash Recovery.

Directories defined by innodb_data_home_dir, innodb_undo_directory, and datadir are automatically appended to the innodb_directories argument value, regardless of whether the innodb_directories option is specified explicitly.

`innodb_directories` may be specified as an option in a startup command or in a MySQL option file. Quotes are used around the argument value because otherwise a semicolon (;) is interpreted as a special character by some command interpreters. (Unix shells treat it as a command terminator, for example.)

Startup command:

```
mysqld --innodb-directories="directory_path_1;directory_path_2"
```

MySQL option file:

```
[mysqld]
innodb_directories="directory_path_1;directory_path_2"
```

Wildcard expressions cannot be used to specify directories.

The `innodb_directories` scan also traverses the subdirectories of specified directories. Duplicate directories and subdirectories are discarded from the list of directories to be scanned.

For more information, see Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”.

- `innodb_disable_sort_file_cache`

Property	Value
Command-Line Format	`--innodb-disable-sort-file-cache[={OFF|ON}]`
System Variable	`innodb_disable_sort_file_cache`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Disables the operating system file system cache for merge-sort temporary files. The effect is to open such files with the equivalent of `O_DIRECT`.

- `innodb_doublewrite`

Property	Value
Command-Line Format	`--innodb-doublewrite[={OFF|ON}]`
System Variable	`innodb_doublewrite`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

When enabled (the default), InnoDB stores all data twice, first to the doublewrite buffer, then to the actual data files. This variable can be turned off with --skip-innodb-doublewrite for benchmarks or cases when top performance is needed rather than concern for data integrity or possible failures.

If system tablespace data files (ibdata* files) are located on Fusion-io devices that support atomic writes, doublewrite buffering is automatically disabled and Fusion-io atomic writes are used for all data files. Because the doublewrite buffer setting is global, doublewrite buffering is also disabled for data files residing on non-Fusion-io hardware. This feature is only supported on Fusion-io hardware and only enabled for Fusion-io NVMFS on Linux. To take full advantage of this feature, an innodb_flush_method setting of O_DIRECT is recommended.

For related information, see Section 15.6.4, “Doublewrite Buffer”.

- innodb_fast_shutdown

Property	Value
Command-Line Format	--innodb-fast-shutdown=#
System Variable	innodb_fast_shutdown
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1
Valid Values	0 1 2

The InnoDB shutdown mode. If the value is 0, InnoDB does a slow shutdown, a full purge and a change buffer merge before shutting down. If the value is 1 (the default), InnoDB skips these operations at shutdown, a process known as a fast shutdown. If the value is 2, InnoDB flushes its logs and shuts down cold, as if MySQL had crashed; no committed transactions are lost, but the crash recovery operation makes the next startup take longer.

The slow shutdown can take minutes, or even hours in extreme cases where substantial amounts of data are still buffered. Use the slow shutdown technique before upgrading or downgrading between MySQL major releases, so that all data files are fully prepared in case the upgrade process updates the file format.

Use innodb_fast_shutdown=2 in emergency or troubleshooting situations, to get the absolute fastest shutdown if data is at risk of corruption.

- innodb_fil_make_page_dirty_debug

Property	Value
Command-Line Format	--innodb-fil-make-page-dirty-debug=#
System Variable	innodb_fil_make_page_dirty_debug
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Maximum Value	2**32-1

By default, setting innodb_fil_make_page_dirty_debug to the ID of a tablespace immediately dirties the first page of the tablespace. If innodb_saved_page_number_debug is set to a non-default value, setting innodb_fil_make_page_dirty_debug dirties the specified page. The innodb_fil_make_page_dirty_debug option is only available if debugging support is compiled in using the WITH_DEBUG CMake option.

- innodb_file_per_table

Property	Value
Command-Line Format	--innodb-file-per-table[={OFF|ON}]
System Variable	innodb_file_per_table
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

When innodb_file_per_table is enabled, tables are created in file-per-table tablespaces by default. When disabled, tables are created in the system tablespace by default. For information about file-per-table tablespaces, see Section 15.6.3.2, “File-Per-Table Tablespaces”. For information about the InnoDB system tablespace, see Section 15.6.3.1, “The System Tablespace”.

The innodb_file_per_table variable can be configured at runtime using a SET GLOBAL statement, specified on the command line at startup, or specified in an option file. Configuration at runtime requires privileges sufficient to set global system variables (see Section 5.1.9.1, “System Variable Privileges”) and immediately affects the operation of all connections.

When a table that resides in a file-per-table tablespace is truncated or dropped, the freed space is returned to the operating system. Truncating or dropping a table that resides in the system tablespace only frees space in the system tablespace. Freed space in the system tablespace can be used again for InnoDB data but is not returned to the operating system, as system tablespace data files never shrink.

The innodb_file_per-table setting does not affect the creation of temporary tables. As of MySQL 8.0.14, temporary tables are created in session temporary tablespaces, and in the global temporary tablespace before that. See Section 15.6.3.5, “Temporary Tablespaces”.

- innodb_fill_factor

Property	Value
Command-Line Format	--innodb-fill-factor=#
System Variable	innodb_fill_factor
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	100
Minimum Value	10
Maximum Value	100

InnoDB performs a bulk load when creating or rebuilding indexes. This method of index creation is known as a “sorted index build”.

innodb_fill_factor defines the percentage of space on each B-tree page that is filled during a sorted index build, with the remaining space reserved for future index growth. For example, setting innodb_fill_factor to 80 reserves 20 percent of the space on each B-tree page for future index growth. Actual percentages may vary. The innodb_fill_factor setting is interpreted as a hint rather than a hard limit.

An innodb_fill_factor setting of 100 leaves 1/16 of the space in clustered index pages free for future index growth.

innodb_fill_factor applies to both B-tree leaf and non-leaf pages. It does not apply to external pages used for TEXT or BLOB entries.

For more information, see Section 15.6.2.3, “Sorted Index Builds”.

- innodb_flush_log_at_timeout

Property	Value
Command-Line Format	--innodb-flush-log-at-timeout=#
System Variable	innodb_flush_log_at_timeout
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1
Minimum Value	1
Maximum Value	2700

Write and flush the logs every *N* seconds. innodb_flush_log_at_timeout allows the timeout period between flushes to be increased in order to reduce flushing and avoid impacting performance of binary log group commit. The default setting for innodb_flush_log_at_timeout is once per second.

- innodb_flush_log_at_trx_commit

Property	Value
Command-Line Format	--innodb-flush-log-at-trx-commit=#
System Variable	innodb_flush_log_at_trx_commit
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`1`

Valid Values	`0`
	`1`
	`2`

Controls the balance between strict ACID compliance for commit operations and higher performance that is possible when commit-related I/O operations are rearranged and done in batches. You can achieve better performance by changing the default value but then you can lose transactions in a crash.

- The default setting of 1 is required for full ACID compliance. Logs are written and flushed to disk at each transaction commit.
- With a setting of 0, logs are written and flushed to disk once per second. Transactions for which logs have not been flushed can be lost in a crash.
- With a setting of 2, logs are written after each transaction commit and flushed to disk once per second. Transactions for which logs have not been flushed can be lost in a crash.
- For settings 0 and 2, once-per-second flushing is not 100% guaranteed. Flushing may occur more frequently due to DDL changes and other internal `InnoDB` activities that cause logs to be flushed independently of the `innodb_flush_log_at_trx_commit` setting, and sometimes less frequently due to scheduling issues. If logs are flushed once per second, up to one second of transactions can be lost in a crash. If logs are flushed more or less frequently than once per second, the amount of transactions that can be lost varies accordingly.
- Log flushing frequency is controlled by `innodb_flush_log_at_timeout`, which allows you to set log flushing frequency to *`N`* seconds (where *`N`* is `1 ... 2700`, with a default value of 1). However, any `mysqld` process crash can erase up to *`N`* seconds of transactions.
- DDL changes and other internal `InnoDB` activities flush the log independently of the `innodb_flush_log_at_trx_commit` setting.
- `InnoDB` crash recovery works regardless of the `innodb_flush_log_at_trx_commit` setting. Transactions are either applied entirely or erased entirely.

For durability and consistency in a replication setup that uses `InnoDB` with transactions:

- If binary logging is enabled, set `sync_binlog=1`.
- Always set `innodb_flush_log_at_trx_commit=1`.

Caution

Many operating systems and some disk hardware fool the flush-to-disk operation. They may tell `mysqld` that the flush has taken place, even though it has not. In this case, the durability of transactions is not guaranteed even with the recommended settings, and in the worst case, a power outage can corrupt `InnoDB` data. Using a battery-backed disk cache in the SCSI disk controller or in

the disk itself speeds up file flushes, and makes the operation safer. You can also try to disable the caching of disk writes in hardware caches.

- `innodb_flush_method`

Property	Value
Command-Line Format	`--innodb-flush-method=value`
System Variable	`innodb_flush_method`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value (Windows)	`unbuffered`
Default Value (Unix)	`fsync`
Valid Values (Windows)	`unbuffered` `normal`

Valid Values (Unix)	`fsync`
	`O_DSYNC`
	`littlesync`
	`nosync`
	`O_DIRECT`
	`O_DIRECT_NO_FSYNC`

Defines the method used to flush data to `InnoDB` data files and log files, which can affect I/O throughput.

On Unix-like systems, the default value is `fsync`. On Windows, the default value is `unbuffered`.

Note

In MySQL 8.0, `innodb_flush_method` options may be specified numerically.

The `innodb_flush_method` options for Unix-like systems include:

- `fsync` or `0`: `InnoDB` uses the `fsync()` system call to flush both the data and log files. `fsync` is the default setting.
- `O_DSYNC` or `1`: `InnoDB` uses `O_SYNC` to open and flush the log files, and `fsync()` to flush the data files. `InnoDB` does not use `O_DSYNC` directly because there have been problems with it on many varieties of Unix.
- `littlesync` or `2`: This option is used for internal performance testing and is currently unsupported. Use at your own risk.
- `nosync` or `3`: This option is used for internal performance testing and is currently unsupported. Use at your own risk.

- `O_DIRECT` or `4`: `InnoDB` uses `O_DIRECT` (or `directio()` on Solaris) to open the data files, and uses `fsync()` to flush both the data and log files. This option is available on some GNU/Linux versions, FreeBSD, and Solaris.

- `O_DIRECT_NO_FSYNC`: `InnoDB` uses `O_DIRECT` during flushing I/O, but skips the `fsync()` system call after each write operation.

  Prior to MySQL 8.0.14, this setting is not suitable for file systems such as XFS and EXT4, which require an `fsync()` system call to synchronize file system metadata changes. If you are not sure whether your file system requires an `fsync()` system call to synchronize file system metadata changes, use `O_DIRECT` instead.

  As of MySQL 8.0.14, `fsync()` is called after creating a new file, after increasing file size, and after closing a file, to ensure that file system metadata changes are synchronized. The `fsync()` system call is still skipped after each write operation.

  On storage devices with cache, data loss is possible if data files and redo log files reside on different storage devices, and a crash occurs before data file writes are flushed from the device cache. If you use or intend to use different storage devices for redo logs and data files, use `O_DIRECT` instead.

The `innodb_flush_method` options for Windows systems include:

- `unbuffered` or `0`: `InnoDB` uses simulated asynchronous I/O and non-buffered I/O.

- `normal` or `1`: `InnoDB` uses simulated asynchronous I/O and buffered I/O.

How each setting affects performance depends on hardware configuration and workload. Benchmark your particular configuration to decide which setting to use, or whether to keep the default setting. Examine the `Innodb_data_fsyncs` status variable to see the overall number of `fsync()` calls for each setting. The mix of read and write operations in your workload can affect how a setting performs. For example, on a system with a hardware RAID controller and battery-backed write cache, `O_DIRECT` can help to avoid double buffering between the `InnoDB` buffer pool and the operating system file system cache. On some systems where `InnoDB` data and log files are located on a SAN, the default value or `O_DSYNC` might be faster for a read-heavy workload with mostly `SELECT` statements. Always test this parameter with hardware and workload that reflect your production environment. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

If `innodb_dedicated_server` is enabled, the `innodb_flush_method` value is automatically configured if it is not explicitly defined. For more information, see Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

- `innodb_flush_neighbors`

Property	Value
Command-Line Format	`--innodb-flush-neighbors=#`
System Variable	`innodb_flush_neighbors`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value (>= 8.0.3)	`0`

Property	Value
Default Value (<= 8.0.2)	`1`

Valid Values	`0`
	`1`
	`2`

Specifies whether flushing a page from the `InnoDB` buffer pool also flushes other dirty pages in the same extent.

- A setting of 0 disables `innodb_flush_neighbors`. Dirty pages in the same extent are not flushed.
- The default setting of 1 flushes contiguous dirty pages in the same extent.
- A setting of 2 flushes dirty pages in the same extent.

When the table data is stored on a traditional HDD storage device, flushing such neighbor pages in one operation reduces I/O overhead (primarily for disk seek operations) compared to flushing individual pages at different times. For table data stored on SSD, seek time is not a significant factor and you can set this option to 0 to spread out write operations. For related information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”.

- `innodb_flush_sync`

Property	Value
Command-Line Format	`--innodb-flush-sync[={OFF|ON}]`
System Variable	`innodb_flush_sync`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

The `innodb_flush_sync` parameter, which is enabled by default, causes the `innodb_io_capacity` setting to be ignored for bursts of I/O activity that occur at checkpoints. To adhere to the limit on `InnoDB` background I/O activity defined by the `innodb_io_capacity` setting, disable `innodb_flush_sync`.

For related information, see Section 15.8.7, “Configuring InnoDB I/O Capacity”.

- `innodb_flushing_avg_loops`

Property	Value
Command-Line Format	`--innodb-flushing-avg-loops=#`
System Variable	`innodb_flushing_avg_loops`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value	30
Minimum Value	1
Maximum Value	1000

Number of iterations for which InnoDB keeps the previously calculated snapshot of the flushing state, controlling how quickly adaptive flushing responds to changing workloads. Increasing the value makes the rate of flush operations change smoothly and gradually as the workload changes. Decreasing the value makes adaptive flushing adjust quickly to workload changes, which can cause spikes in flushing activity if the workload increases and decreases suddenly.

For related information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”.

- innodb_force_load_corrupted

Property	Value
Command-Line Format	--innodb-force-load-corrupted[={OFF|ON}]
System Variable	innodb_force_load_corrupted
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Permits InnoDB to load tables at startup that are marked as corrupted. Use only during troubleshooting, to recover data that is otherwise inaccessible. When troubleshooting is complete, disable this setting and restart the server.

- innodb_force_recovery

Property	Value
Command-Line Format	--innodb-force-recovery=#
System Variable	innodb_force_recovery
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	6

The crash recovery mode, typically only changed in serious troubleshooting situations. Possible values are from 0 to 6. For the meanings of these values and important information about innodb_force_recovery, see Section 15.20.2, “Forcing InnoDB Recovery”.

Warning

Only set this variable to a value greater than 0 in an emergency situation so that you can start `InnoDB` and dump your tables. As a safety measure, `InnoDB` prevents `INSERT`, `UPDATE`, or `DELETE` operations when `innodb_force_recovery` is greater than 0. An `innodb_force_recovery` setting of 4 or greater places `InnoDB` into read-only mode.

These restrictions may cause replication administration commands to fail with an error, as replication stores the slave status logs in `InnoDB` tables.

- `innodb_fsync_threshold`

Property	Value
Command-Line Format	`--innodb-fsync-threshold=#`
Introduced	8.0.13
System Variable	`innodb_fsync_threshold`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2**64-1`

By default, when `InnoDB` creates a new data file, such as a new log file or tablespace file, the file is fully written to the operating system cache before it is flushed to disk, which can cause a large amount of disk write activity to occur at once. To force smaller, periodic flushes of data from the operating system cache, you can use the `innodb_fsync_threshold` variable to define a threshold value, in bytes. When the byte threshold is reached, the contents of the operating system cache are flushed to disk. The default value of 0 forces the default behavior, which is to flush data to disk only after a file is fully written to the cache.

Specifying a threshold to force smaller, periodic flushes may be beneficial in cases where multiple MySQL instances use the same storage devices. For example, creating a new MySQL instance and its associated data files could cause large surges of disk write activity, impeding the performance of other MySQL instances that use the same storage devices. Configuring a threshold helps avoid such surges in write activity.

- `innodb_ft_aux_table`

Property	Value
System Variable	`innodb_ft_aux_table`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

Specifies the qualified name of an InnoDB table containing a FULLTEXT index. This variable is intended for diagnostic purposes and can only be set at runtime. For example:

```
SET GLOBAL innodb_ft_aux_table = 'test/t1';
```

After you set this variable to a name in the format *db_name/table_name*, the INFORMATION_SCHEMA tables INNODB_FT_INDEX_TABLE, INNODB_FT_INDEX_CACHE, INNODB_FT_CONFIG, INNODB_FT_DELETED, and INNODB_FT_BEING_DELETED show information about the search index for the specified table.

For more information, see Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

- innodb_ft_cache_size

Property	Value
Command-Line Format	--innodb-ft-cache-size=#
System Variable	innodb_ft_cache_size
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	8000000
Minimum Value	1600000
Maximum Value	80000000

The memory allocated, in bytes, for the InnoDB FULLTEXT search index cache, which holds a parsed document in memory while creating an InnoDB FULLTEXT index. Index inserts and updates are only committed to disk when the innodb_ft_cache_size size limit is reached. innodb_ft_cache_size defines the cache size on a per table basis. To set a global limit for all tables, see innodb_ft_total_cache_size.

For more information, see InnoDB Full-Text Index Cache.

- innodb_ft_enable_diag_print

Property	Value
Command-Line Format	--innodb-ft-enable-diag-print[={OFF|ON}]
System Variable	innodb_ft_enable_diag_print
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Whether to enable additional full-text search (FTS) diagnostic output. This option is primarily intended for advanced FTS debugging and will not be of interest to most users. Output is printed to the error log and includes information such as:

- FTS index sync progress (when the FTS cache limit is reached). For example:

```
FTS SYNC for table test, deleted count: 100 size: 10000 bytes
SYNC words: 100
```

- FTS optimize progress. For example:

```
FTS start optimize test
FTS_OPTIMIZE: optimize "mysql"
FTS_OPTIMIZE: processed "mysql"
```

- FTS index build progress. For example:

```
Number of doc processed: 1000
```

- For FTS queries, the query parsing tree, word weight, query processing time, and memory usage are printed. For example:

```
FTS Search Processing time: 1 secs: 100 millisec: row(s) 10000
Full Search Memory: 245666 (bytes),  Row: 10000
```

- `innodb_ft_enable_stopword`

Property	Value
Command-Line Format	`--innodb-ft-enable-stopword[={OFF|ON}]`
System Variable	`innodb_ft_enable_stopword`
Scope	Global, Session
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`ON`

Specifies that a set of stopwords is associated with an `InnoDB` `FULLTEXT` index at the time the index is created. If the `innodb_ft_user_stopword_table` option is set, the stopwords are taken from that table. Else, if the `innodb_ft_server_stopword_table` option is set, the stopwords are taken from that table. Otherwise, a built-in set of default stopwords is used.

For more information, see Section 12.9.4, “Full-Text Stopwords”.

- `innodb_ft_max_token_size`

Property	Value
Command-Line Format	`--innodb-ft-max-token-size=#`
System Variable	`innodb_ft_max_token_size`
Scope	Global

Property	Value
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	84
Minimum Value	10
Maximum Value	84

Maximum character length of words that are stored in an InnoDB FULLTEXT index. Setting a limit on this value reduces the size of the index, thus speeding up queries, by omitting long keywords or arbitrary collections of letters that are not real words and are not likely to be search terms.

For more information, see Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”.

- innodb_ft_min_token_size

Property	Value
Command-Line Format	--innodb-ft-min-token-size=#
System Variable	innodb_ft_min_token_size
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	3
Minimum Value	0
Maximum Value	16

Minimum length of words that are stored in an InnoDB FULLTEXT index. Increasing this value reduces the size of the index, thus speeding up queries, by omitting common words that are unlikely to be significant in a search context, such as the English words “a” and “to”. For content using a CJK (Chinese, Japanese, Korean) character set, specify a value of 1.

For more information, see Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”.

- innodb_ft_num_word_optimize

Property	Value
Command-Line Format	--innodb-ft-num-word-optimize=#
System Variable	innodb_ft_num_word_optimize
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	2000

Number of words to process during each `OPTIMIZE TABLE` operation on an `InnoDB` `FULLTEXT` index. Because a bulk insert or update operation to a table containing a full-text search index could require substantial index maintenance to incorporate all changes, you might do a series of `OPTIMIZE TABLE` statements, each picking up where the last left off.

For more information, see Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”.

- `innodb_ft_result_cache_limit`

Property	Value
Command-Line Format	`--innodb-ft-result-cache-limit=#`
System Variable	`innodb_ft_result_cache_limit`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`2000000000`
Minimum Value	`1000000`
Maximum Value	`2**32-1`

The `InnoDB` full-text search query result cache limit (defined in bytes) per full-text search query or per thread. Intermediate and final `InnoDB` full-text search query results are handled in memory. Use `innodb_ft_result_cache_limit` to place a size limit on the full-text search query result cache to avoid excessive memory consumption in case of very large `InnoDB` full-text search query results (millions or hundreds of millions of rows, for example). Memory is allocated as required when a full-text search query is processed. If the result cache size limit is reached, an error is returned indicating that the query exceeds the maximum allowed memory.

The maximum value of `innodb_ft_result_cache_limit` for all platform types and bit sizes is 2**32-1.

- `innodb_ft_server_stopword_table`

Property	Value
Command-Line Format	`--innodb-ft-server-stopword-table=db_name/table_name`
System Variable	`innodb_ft_server_stopword_table`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`NULL`

This option is used to specify your own `InnoDB` `FULLTEXT` index stopword list for all `InnoDB` tables. To configure your own stopword list for a specific `InnoDB` table, use `innodb_ft_user_stopword_table`.

Set `innodb_ft_server_stopword_table` to the name of the table containing a list of stopwords, in the format *`db_name/table_name`*.

The stopword table must exist before you configure `innodb_ft_server_stopword_table`. `innodb_ft_enable_stopword` must be enabled and `innodb_ft_server_stopword_table` option must be configured before you create the `FULLTEXT` index.

The stopword table must be an `InnoDB` table, containing a single `VARCHAR` column named `value`.

For more information, see Section 12.9.4, “Full-Text Stopwords”.

- `innodb_ft_sort_pll_degree`

Property	Value
Command-Line Format	`--innodb-ft-sort-pll-degree=#`
System Variable	`innodb_ft_sort_pll_degree`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`2`
Minimum Value	`1`
Maximum Value	`32`

Number of threads used in parallel to index and tokenize text in an `InnoDB FULLTEXT` index when building a search index.

For related information, see Section 15.6.2.4, “InnoDB FULLTEXT Indexes”, and `innodb_sort_buffer_size`.

- `innodb_ft_total_cache_size`

Property	Value
Command-Line Format	`--innodb-ft-total-cache-size=#`
System Variable	`innodb_ft_total_cache_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`640000000`
Minimum Value	`32000000`
Maximum Value	`1600000000`

The total memory allocated, in bytes, for the `InnoDB` full-text search index cache for all tables. Creating numerous tables, each with a `FULLTEXT` search index, could consume a significant portion of available memory. `innodb_ft_total_cache_size` defines a global memory limit for all full-text search indexes to help avoid excessive memory consumption. If the global limit is reached by an index operation, a forced sync is triggered.

For more information, see InnoDB Full-Text Index Cache.

- `innodb_ft_user_stopword_table`

Property	Value
Command-Line Format	`--innodb-ft-user-stopword-table=db_name/table_name`
System Variable	`innodb_ft_user_stopword_table`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`NULL`

This option is used to specify your own `InnoDB` `FULLTEXT` index stopword list on a specific table. To configure your own stopword list for all `InnoDB` tables, use `innodb_ft_server_stopword_table`.

Set `innodb_ft_user_stopword_table` to the name of the table containing a list of stopwords, in the format *`db_name/table_name`*.

The stopword table must exist before you configure `innodb_ft_user_stopword_table`. `innodb_ft_enable_stopword` must be enabled and `innodb_ft_user_stopword_table` must be configured before you create the `FULLTEXT` index.

The stopword table must be an `InnoDB` table, containing a single `VARCHAR` column named `value`.

For more information, see Section 12.9.4, “Full-Text Stopwords”.

- `innodb_idle_flush_pct`

Property	Value
Command-Line Format	`--innodb-idle-flush-pct=#`
Introduced	8.0.18
System Variable	`innodb_idle_flush_pct`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`100`
Minimum Value	`0`
Maximum Value	`100`

Limits page flushing when `InnoDB` is idle. The `innodb_idle_flush_pct` value is a percentage of the `innodb_io_capacity` setting, which defines the number of I/O operations per second available to `InnoDB`. For more information, see Limiting Buffer Flushing During Idle Periods.

- `innodb_io_capacity`

Property	Value
Command-Line Format	`--innodb-io-capacity=#`

Property	Value
System Variable	`innodb_io_capacity`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`200`
Minimum Value	`100`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`2**32-1`

The `innodb_io_capacity` parameter sets an upper limit on the number of I/O operations performed per second by `InnoDB` background tasks, such as flushing pages from the buffer pool and merging data from the change buffer.

The `innodb_io_capacity` limit is a total limit for all buffer pool instances. When dirty pages are flushed, the limit is divided equally among buffer pool instances.

`innodb_io_capacity` should be set to approximately the number of I/O operations that the system can perform per second. Ideally, keep the setting as low as practical, but not so low that background activities fall behind. If the value is too high, data is removed from the buffer pool and insert buffer too quickly for caching to provide a significant benefit.

The default value is 200. For busy systems capable of higher I/O rates, you can set a higher value to help the server handle the background maintenance work associated with a high rate of row changes.

In general, you can increase the value as a function of the number of drives used for `InnoDB` I/O. For example, you can increase the value on systems that use multiple disks or solid-state disks (SSD).

The default setting of 200 is generally sufficient for a lower-end SSD. For a higher-end, bus-attached SSD, consider a higher setting such as 1000, for example. For systems with individual 5400 RPM or 7200 RPM drives, you might lower the value to `100`, which represents an estimated proportion of the I/O operations per second (IOPS) available to older-generation disk drives that can perform about 100 IOPS.

Although you can specify a very high value such as one million, in practice such large values have little if any benefit. Generally, a value of 20000 or higher is not recommended unless you have proven that lower values are insufficient for your workload.

Consider write workload when tuning `innodb_io_capacity`. Systems with large write workloads are likely to benefit from a higher setting. A lower setting may be sufficient for systems with a small write workload.

You can set `innodb_io_capacity` to any number 100 or greater to a maximum defined by `innodb_io_capacity_max`. `innodb_io_capacity` can be set in the MySQL option file (`my.cnf` or `my.ini`) or changed dynamically using a `SET GLOBAL` statement, which requires privileges sufficient to set global system variables. See Section 5.1.9.1, “System Variable Privileges”.

The `innodb_flush_sync` variable causes the `innodb_io_capacity` setting to be ignored during bursts of I/O activity that occur at checkpoints. `innodb_flush_sync` is enabled by default.

See Section 15.8.7, “Configuring InnoDB I/O Capacity” for more information. For general information about InnoDB I/O performance, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- innodb_io_capacity_max

Property	Value
Command-Line Format	--innodb-io-capacity-max=#
System Variable	innodb_io_capacity_max
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	see description
Minimum Value	100
Maximum Value (Windows, 64-bit platforms)	2**32-1
Maximum Value (Unix, 64-bit platforms)	2**64-1
Maximum Value (32-bit platforms)	2**32-1

If flushing activity falls behind, InnoDB can flush more aggressively than the limit imposed by innodb_io_capacity. innodb_io_capacity_max defines an upper limit the number of I/O operations performed per second by InnoDB background tasks in such situations.

The innodb_io_capacity_max setting is a total limit for all buffer pool instances.

If you specify an innodb_io_capacity setting at startup but do not specify a value for innodb_io_capacity_max, innodb_io_capacity_max defaults to twice the value of innodb_io_capacity, with a minimum value of 2000.

When configuring innodb_io_capacity_max, twice the innodb_io_capacity is often a good starting point. The default value of 2000 is intended for workloads that use a solid-state disk (SSD) or more than one regular disk drive. A setting of 2000 is likely too high for workloads that do not use SSD or multiple disk drives, and could allow too much flushing. For a single regular disk drive, a setting between 200 and 400 is recommended. For a high-end, bus-attached SSD, consider a higher setting such as 2500. As with the innodb_io_capacity setting, keep the setting as low as practical, but not so low that InnoDB cannot sufficiently extend beyond the innodb_io_capacity limit, if necessary.

Consider write workload when tuning innodb_io_capacity_max. Systems with large write workloads may benefit from a higher setting. A lower setting may be sufficient for systems with a small write workload.

innodb_io_capacity_max cannot be set to a value lower than the innodb_io_capacity value.

Setting innodb_io_capacity_max to DEFAULT using a SET statement (SET GLOBAL innodb_io_capacity_max=DEFAULT) sets innodb_io_capacity_max to the maximum value.

For related information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”.

- `innodb_limit_optimistic_insert_debug`

Property	Value
Command-Line Format	`--innodb-limit-optimistic-insert-debug=#`
System Variable	`innodb_limit_optimistic_insert_debug`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2**32-1`

Limits the number of records per B-tree page. A default value of 0 means that no limit is imposed. This option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_lock_wait_timeout`

Property	Value
Command-Line Format	`--innodb-lock-wait-timeout=#`
System Variable	`innodb_lock_wait_timeout`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`50`
Minimum Value	`1`
Maximum Value	`1073741824`

The length of time in seconds an `InnoDB` transaction waits for a row lock before giving up. The default value is 50 seconds. A transaction that tries to access a row that is locked by another `InnoDB` transaction waits at most this many seconds for write access to the row before issuing the following error:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

When a lock wait timeout occurs, the current statement is rolled back (not the entire transaction). To have the entire transaction roll back, start the server with the `--innodb-rollback-on-timeout` option. See also Section 15.20.4, “InnoDB Error Handling”.

You might decrease this value for highly interactive applications or OLTP systems, to display user feedback quickly or put the update into a queue for processing later. You might increase this value for long-running back-end operations, such as a transform step in a data warehouse that waits for other large insert or update operations to finish.

`innodb_lock_wait_timeout` applies to `InnoDB` row locks. A MySQL table lock does not happen inside `InnoDB` and this timeout does not apply to waits for table locks.

The lock wait timeout value does not apply to deadlocks when `innodb_deadlock_detect` is enabled (the default) because `InnoDB` detects deadlocks immediately and rolls back one of the deadlocked transactions. When `innodb_deadlock_detect` is disabled, `InnoDB` relies on `innodb_lock_wait_timeout` for transaction rollback when a deadlock occurs. See Section 15.7.5.2, “Deadlock Detection and Rollback”.

`innodb_lock_wait_timeout` can be set at runtime with the `SET GLOBAL` or `SET SESSION` statement. Changing the `GLOBAL` setting requires privileges sufficient to set global system variables (see Section 5.1.9.1, “System Variable Privileges”) and affects the operation of all clients that subsequently connect. Any client can change the `SESSION` setting for `innodb_lock_wait_timeout`, which affects only that client.

- `innodb_log_buffer_size`

Property	Value
Command-Line Format	`--innodb-log-buffer-size=#`
System Variable (>= 8.0.11)	`innodb_log_buffer_size`
System Variable (<= 8.0.4)	`innodb_log_buffer_size`
Scope (>= 8.0.11)	Global
Scope (<= 8.0.4)	Global
Dynamic (>= 8.0.11)	Yes
Dynamic (<= 8.0.4)	No
SET_VAR Hint Applies (>= 8.0.11)	No
SET_VAR Hint Applies (<= 8.0.4)	No
Type	Integer
Default Value	`16777216`
Minimum Value	`1048576`
Maximum Value	`4294967295`

The size in bytes of the buffer that `InnoDB` uses to write to the log files on disk. The default is 16MB. A large log buffer enables large transactions to run without the need to write the log to disk before the transactions commit. Thus, if you have transactions that update, insert, or delete many rows, making the log buffer larger saves disk I/O. For related information, see Memory Configuration, and Section 8.5.4, “Optimizing InnoDB Redo Logging”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_log_checkpoint_fuzzy_now`

Property	Value
Command-Line Format	`--innodb-log-checkpoint-fuzzy-now[={OFF|ON}]`
Introduced	8.0.13
System Variable	`innodb_log_checkpoint_fuzzy_now`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Enable this debug option to force InnoDB to write a fuzzy checkpoint. This option is only available if debugging support is compiled in using the WITH_DEBUG CMake option.

- innodb_log_checkpoint_now

Property	Value
Command-Line Format	--innodb-log-checkpoint-now[={OFF|ON}]
System Variable	innodb_log_checkpoint_now
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Enable this debug option to force InnoDB to write a checkpoint. This option is only available if debugging support is compiled in using the WITH_DEBUG CMake option.

- innodb_log_checksums

Property	Value
Command-Line Format	--innodb-log-checksums[={OFF|ON}]
System Variable	innodb_log_checksums
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

Enables or disables checksums for redo log pages.

innodb_log_checksums=ON enables the CRC-32C checksum algorithm for redo log pages. When innodb_log_checksums is disabled, the contents of the redo log page checksum field are ignored.

Checksums on the redo log header page and redo log checkpoint pages are never disabled.

- innodb_log_compressed_pages

Property	Value
Command-Line Format	--innodb-log-compressed-pages[={OFF|ON}]
System Variable	innodb_log_compressed_pages
Scope	Global

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

Specifies whether images of re-compressed pages are written to the redo log. Re-compression may occur when changes are made to compressed data.

innodb_log_compressed_pages is enabled by default to prevent corruption that could occur if a different version of the zlib compression algorithm is used during recovery. If you are certain that the zlib version will not change, you can disable innodb_log_compressed_pages to reduce redo log generation for workloads that modify compressed data.

To measure the effect of enabling or disabling innodb_log_compressed_pages, compare redo log generation for both settings under the same workload. Options for measuring redo log generation include observing the Log sequence number (LSN) in the LOG section of SHOW ENGINE INNODB STATUS output, or monitoring Innodb_os_log_written status for the number of bytes written to the redo log files.

For related information, see Section 15.9.1.6, “Compression for OLTP Workloads”.

- innodb_log_file_size

Property	Value
Command-Line Format	--innodb-log-file-size=#
System Variable	innodb_log_file_size
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	50331648
Minimum Value	4194304
Maximum Value	512GB / innodb_log_files_in_group

The size in bytes of each log file in a log group. The combined size of log files (innodb_log_file_size * innodb_log_files_in_group) cannot exceed a maximum value that is slightly less than 512GB. A pair of 255 GB log files, for example, approaches the limit but does not exceed it. The default value is 48MB.

Generally, the combined size of the log files should be large enough that the server can smooth out peaks and troughs in workload activity, which often means that there is enough redo log space to handle more than an hour of write activity. The larger the value, the less checkpoint flush activity is required in the buffer pool, saving disk I/O. Larger log files also make crash recovery slower, although improvements to recovery performance make log file size less of a consideration than it was in earlier versions of MySQL.

The minimum innodb_log_file_size is 4MB.

For related information, see Redo Log File Configuration. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

If `innodb_dedicated_server` is enabled, the `innodb_log_file_size` value is automatically configured if it is not explicitly defined. For more information, see Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

- `innodb_log_files_in_group`

Property	Value
Command-Line Format	`--innodb-log-files-in-group=#`
System Variable	`innodb_log_files_in_group`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`2`
Minimum Value	`2`
Maximum Value	`100`

The number of log files in the log group. `InnoDB` writes to the files in a circular fashion. The default (and recommended) value is 2. The location of the files is specified by `innodb_log_group_home_dir`. The combined size of log files (`innodb_log_file_size` * `innodb_log_files_in_group`) can be up to 512GB.

For related information, see Redo Log File Configuration.

- `innodb_log_group_home_dir`

Property	Value
Command-Line Format	`--innodb-log-group-home-dir=dir_name`
System Variable	`innodb_log_group_home_dir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

The directory path to the `InnoDB` redo log files, whose number is specified by `innodb_log_files_in_group`. If you do not specify any `InnoDB` log variables, the default is to create two files named `ib_logfile0` and `ib_logfile1` in the MySQL data directory. Log file size is given by the `innodb_log_file_size` system variable.

For related information, see Redo Log File Configuration.

- `innodb_log_spin_cpu_abs_lwm`

Property	Value
Command-Line Format	`--innodb-log-spin-cpu-abs-lwm=#`

Property	Value
Introduced	8.0.11
System Variable	`innodb_log_spin_cpu_abs_lwm`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`80`
Minimum Value	`0`
Maximum Value	`4294967295`

Defines the minimum amount of CPU usage below which user threads no longer spin while waiting for flushed redo. The value is expressed as a sum of CPU core usage. For example, The default value of 80 is 80% of a single CPU core. On a system with a multi-core processor, a value of 150 represents 100% usage of one CPU core plus 50% usage of a second CPU core.

For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

- `innodb_log_spin_cpu_pct_hwm`

Property	Value
Command-Line Format	`--innodb-log-spin-cpu-pct-hwm=#`
Introduced	8.0.11
System Variable	`innodb_log_spin_cpu_pct_hwm`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`50`
Minimum Value	`0`
Maximum Value	`100`

Defines the maximum amount of CPU usage above which user threads no longer spin while waiting for flushed redo. The value is expressed as a percentage of the combined total processing power of all CPU cores. The default value is 50%. For example, 100% usage of two CPU cores is 50% of the combined CPU processing power on a server with four CPU cores.

The `innodb_log_spin_cpu_pct_hwm` variable respects processor affinity. For example, if a server has 48 cores but the `mysqld` process is pinned to only four CPU cores, the other 44 CPU cores are ignored.

For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

- `innodb_log_wait_for_flush_spin_hwm`

Property	Value
Command-Line Format	`--innodb-log-wait-for-flush-spin-hwm=#`

Property	Value
Introduced	8.0.11
System Variable	`innodb_log_wait_for_flush_spin_hwm`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`400`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`2**32-1`

Defines the maximum average log flush time beyond which user threads no longer spin while waiting for flushed redo. The default value is 400 microseconds.

For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

- `innodb_log_write_ahead_size`

Property	Value
Command-Line Format	`--innodb-log-write-ahead-size=#`
System Variable	`innodb_log_write_ahead_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`8192`
Minimum Value	`512 (log file block size)`
Maximum Value	`Equal to innodb_page_size`

Defines the write-ahead block size for the redo log, in bytes. To avoid “read-on-write”, set `innodb_log_write_ahead_size` to match the operating system or file system cache block size. The default setting is 8192 bytes. Read-on-write occurs when redo log blocks are not entirely cached to the operating system or file system due to a mismatch between write-ahead block size for the redo log and operating system or file system cache block size.

Valid values for `innodb_log_write_ahead_size` are multiples of the `InnoDB` log file block size (2^n). The minimum value is the `InnoDB` log file block size (512). Write-ahead does not occur when the minimum value is specified. The maximum value is equal to the `innodb_page_size` value. If you specify a value for `innodb_log_write_ahead_size` that is larger than the `innodb_page_size` value, the `innodb_log_write_ahead_size` setting is truncated to the `innodb_page_size` value.

Setting the `innodb_log_write_ahead_size` value too low in relation to the operating system or file system cache block size results in “read-on-write”. Setting the value too high may have a slight impact on `fsync` performance for log file writes due to several blocks being written at once.

For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

- `innodb_lru_scan_depth`

Property	Value
Command-Line Format	`--innodb-lru-scan-depth=#`
System Variable	`innodb_lru_scan_depth`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1024`
Minimum Value	`100`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`2**32-1`

A parameter that influences the algorithms and heuristics for the flush operation for the `InnoDB` buffer pool. Primarily of interest to performance experts tuning I/O-intensive workloads. It specifies, per buffer pool instance, how far down the buffer pool LRU page list the page cleaner thread scans looking for dirty pages to flush. This is a background operation performed once per second.

A setting smaller than the default is generally suitable for most workloads. A value that is much higher than necessary may impact performance. Only consider increasing the value if you have spare I/O capacity under a typical workload. Conversely, if a write-intensive workload saturates your I/O capacity, decrease the value, especially in the case of a large buffer pool.

When tuning `innodb_lru_scan_depth`, start with a low value and configure the setting upward with the goal of rarely seeing zero free pages. Also, consider adjusting `innodb_lru_scan_depth` when changing the number of buffer pool instances, since `innodb_lru_scan_depth` * `innodb_buffer_pool_instances` defines the amount of work performed by the page cleaner thread each second.

For related information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_max_dirty_pages_pct`

Property	Value
Command-Line Format	`--innodb-max-dirty-pages-pct=#`
System Variable	`innodb_max_dirty_pages_pct`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Numeric
Default Value (>= 8.0.3)	`90`

Property	Value
Default Value (<= 8.0.2)	75
Minimum Value	0
Maximum Value	99.99

InnoDB tries to flush data from the buffer pool so that the percentage of dirty pages does not exceed this value.

The innodb_max_dirty_pages_pct setting establishes a target for flushing activity. It does not affect the rate of flushing. For information about managing the rate of flushing, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”.

For related information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- innodb_max_dirty_pages_pct_lwm

Property	Value
Command-Line Format	--innodb-max-dirty-pages-pct-lwm=#
System Variable	innodb_max_dirty_pages_pct_lwm
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Numeric
Default Value (>= 8.0.3)	10
Default Value (<= 8.0.2)	0
Minimum Value	0
Maximum Value	99.99

Defines a low water mark representing the percentage of dirty pages at which preflushing is enabled to control the dirty page ratio. A value of 0 disables the pre-flushing behavior entirely. For more information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”.

- innodb_max_purge_lag

Property	Value
Command-Line Format	--innodb-max-purge-lag=#
System Variable	innodb_max_purge_lag
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	4294967295

Defines the maximum length of the purge queue. The default value of 0 indicates no limit (no delays).

Use this option to impose a delay for `INSERT`, `UPDATE`, and `DELETE` operations when purge operations are lagging (see Section 15.3, “InnoDB Multi-Versioning”).

The `InnoDB` transaction system maintains a list of transactions that have index records delete-marked by `UPDATE` or `DELETE` operations. The length of the list represents the *`purge_lag`* value. When *`purge_lag`* exceeds `innodb_max_purge_lag`, `INSERT`, `UPDATE`, and `DELETE` operations are delayed. Prior to MySQL 8.0.14, the delay calculation is `(purge_lag/innodb_max_purge_lag - 0.5) * 10000`, which results in a minimum delay of 5000 microseconds. As of MySQL 8.0.14, the delay calculation is `(purge_lag/innodb_max_purge_lag - 0.9995) * 10000`, which results in a minimum delay of 5 microseconds.

To prevent excessive delays in extreme situations where *`purge_lag`* becomes huge, you can limit the delay by setting the `innodb_max_purge_lag_delay` variable. The delay is computed at the beginning of a purge batch.

A typical setting for a problematic workload might be 1 million, assuming that transactions are small, only 100 bytes in size, and it is permissible to have 100MB of unpurged `InnoDB` table rows.

The lag value is displayed as the history list length in the `TRANSACTIONS` section of InnoDB Monitor output. The lag value is 20 in this example output:

```
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
History list length 20
```

For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_max_purge_lag_delay`

Property	Value
Command-Line Format	`--innodb-max-purge-lag-delay=#`
System Variable	`innodb_max_purge_lag_delay`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`

Specifies the maximum delay in microseconds for the delay imposed by the `innodb_max_purge_lag` variable. The specified value is the upper limit on the delay period computed from the formula based on the value of `innodb_max_purge_lag`.

For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_max_undo_log_size`

Property	Value
Command-Line Format	`--innodb-max-undo-log-size=#`
System Variable	`innodb_max_undo_log_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1073741824`
Minimum Value	`10485760`
Maximum Value	`2**64-1`

Defines a threshold size for undo tablespaces. If an undo tablespace exceeds the threshold, it can be marked for truncation when `innodb_undo_log_truncate` is enabled. The default value is 1073741824 bytes (1024 MiB).

For more information, see Truncating Undo Tablespaces.

- `innodb_merge_threshold_set_all_debug`

Property	Value
Command-Line Format	`--innodb-merge-threshold-set-all-debug=#`
System Variable	`innodb_merge_threshold_set_all_debug`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`50`
Minimum Value	`1`
Maximum Value	`50`

Defines a page-full percentage value for index pages that overrides the current `MERGE_THRESHOLD` setting for all indexes that are currently in the dictionary cache. This option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option. For related information, see Section 15.8.11, “Configuring the Merge Threshold for Index Pages”.

- `innodb_monitor_disable`

Property	Value
Command-Line Format	`--innodb-monitor-disable={counter|module|pattern|all}`
System Variable	`innodb_monitor_disable`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

Disables InnoDB metrics counters. Counter data may be queried using the INFORMATION_SCHEMA.INNODB_METRICS table. For usage information, see Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

innodb_monitor_disable='latch' disables statistics collection for SHOW ENGINE INNODB MUTEX. For more information, see Section 13.7.7.15, “SHOW ENGINE Syntax”.

- innodb_monitor_enable

Property	Value
Command-Line Format	--innodb-monitor-enable={counter|module|pattern|all}
System Variable	innodb_monitor_enable
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

Enables InnoDB metrics counters. Counter data may be queried using the INFORMATION_SCHEMA.INNODB_METRICS table. For usage information, see Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

innodb_monitor_enable='latch' enables statistics collection for SHOW ENGINE INNODB MUTEX. For more information, see Section 13.7.7.15, “SHOW ENGINE Syntax”.

- innodb_monitor_reset

Property	Value
Command-Line Format	--innodb-monitor-reset={counter|module|pattern|all}
System Variable	innodb_monitor_reset
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	empty string
Valid Values	counter module pattern all

Resets the count value for InnoDB metrics counters to zero. Counter data may be queried using the INFORMATION_SCHEMA.INNODB_METRICS table. For usage information, see Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

innodb_monitor_reset='latch' resets statistics reported by SHOW ENGINE INNODB MUTEX. For more information, see Section 13.7.7.15, “SHOW ENGINE Syntax”.

- innodb_monitor_reset_all

Property	Value
Command-Line Format	--innodb-monitor-reset-all={counter|module|pattern|all}
System Variable	innodb_monitor_reset_all
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	empty string
Valid Values	counter module pattern all

Resets all values (minimum, maximum, and so on) for InnoDB metrics counters. Counter data may be queried using the INFORMATION_SCHEMA.INNODB_METRICS table. For usage information, see Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

- innodb_numa_interleave

Property	Value
Command-Line Format	--innodb-numa-interleave[={OFF|ON}]
System Variable	innodb_numa_interleave
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Enables the NUMA interleave memory policy for allocation of the InnoDB buffer pool. When innodb_numa_interleave is enabled, the NUMA memory policy is set to MPOL_INTERLEAVE for the mysqld process. After the InnoDB buffer pool is allocated, the NUMA memory policy is set back to MPOL_DEFAULT. For the innodb_numa_interleave option to be available, MySQL must be compiled on a NUMA-enabled Linux system.

CMake sets the default WITH_NUMA value based on whether the current platform has NUMA support. For more information, see Section 2.9.7, “MySQL Source-Configuration Options”.

- innodb_old_blocks_pct

Property	Value
Command-Line Format	--innodb-old-blocks-pct=#
System Variable	innodb_old_blocks_pct

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	37
Minimum Value	5
Maximum Value	95

Specifies the approximate percentage of the InnoDB buffer pool used for the old block sublist. The range of values is 5 to 95. The default value is 37 (that is, 3/8 of the pool). Often used in combination with innodb_old_blocks_time.

For more information, see Section 15.8.3.3, “Making the Buffer Pool Scan Resistant”. For information about buffer pool management, the LRU algorithm, and eviction policies, see Section 15.5.1, “Buffer Pool”.

- innodb_old_blocks_time

Property	Value
Command-Line Format	--innodb-old-blocks-time=#
System Variable	innodb_old_blocks_time
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1000
Minimum Value	0
Maximum Value	2**32-1

Non-zero values protect against the buffer pool being filled by data that is referenced only for a brief period, such as during a full table scan. Increasing this value offers more protection against full table scans interfering with data cached in the buffer pool.

Specifies how long in milliseconds a block inserted into the old sublist must stay there after its first access before it can be moved to the new sublist. If the value is 0, a block inserted into the old sublist moves immediately to the new sublist the first time it is accessed, no matter how soon after insertion the access occurs. If the value is greater than 0, blocks remain in the old sublist until an access occurs at least that many milliseconds after the first access. For example, a value of 1000 causes blocks to stay in the old sublist for 1 second after the first access before they become eligible to move to the new sublist.

The default value is 1000.

This variable is often used in combination with innodb_old_blocks_pct. For more information, see Section 15.8.3.3, “Making the Buffer Pool Scan Resistant”. For information about buffer pool management, the LRU algorithm, and eviction policies, see Section 15.5.1, “Buffer Pool”.

- innodb_online_alter_log_max_size

Property	Value
Command-Line Format	`--innodb-online-alter-log-max-size=#`
System Variable	`innodb_online_alter_log_max_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`134217728`
Minimum Value	`65536`
Maximum Value	`2**64-1`

Specifies an upper limit in bytes on the size of the temporary log files used during online DDL operations for `InnoDB` tables. There is one such log file for each index being created or table being altered. This log file stores data inserted, updated, or deleted in the table during the DDL operation. The temporary log file is extended when needed by the value of `innodb_sort_buffer_size`, up to the maximum specified by `innodb_online_alter_log_max_size`. If a temporary log file exceeds the upper size limit, the `ALTER TABLE` operation fails and all uncommitted concurrent DML operations are rolled back. Thus, a large value for this option allows more DML to happen during an online DDL operation, but also extends the period of time at the end of the DDL operation when the table is locked to apply the data from the log.

- `innodb_open_files`

Property	Value
Command-Line Format	`--innodb-open-files=#`
System Variable	`innodb_open_files`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`10`
Maximum Value	`4294967295`

This variable is only relevant if you use multiple `InnoDB` tablespaces. It specifies the maximum number of `.ibd` files that MySQL can keep open at one time. The minimum value is 10. The default value is 300 if `innodb_file_per_table` is not enabled, and the higher of 300 and `table_open_cache` otherwise.

The file descriptors used for `.ibd` files are for `InnoDB` tables only. They are independent of those specified by the `open_files_limit` system variable, and do not affect the operation of the table cache. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_optimize_fulltext_only`

Property	Value
Command-Line Format	`--innodb-optimize-fulltext-only[={OFF|ON}]`
System Variable	`innodb_optimize_fulltext_only`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Changes the way `OPTIMIZE TABLE` operates on `InnoDB` tables. Intended to be enabled temporarily, during maintenance operations for `InnoDB` tables with `FULLTEXT` indexes.

By default, `OPTIMIZE TABLE` reorganizes data in the clustered index of the table. When this option is enabled, `OPTIMIZE TABLE` skips the reorganization of table data, and instead processes newly added, deleted, and updated token data for `InnoDB FULLTEXT` indexes. For more information, see Optimizing InnoDB Full-Text Indexes.

- `innodb_page_cleaners`

Property	Value
Command-Line Format	`--innodb-page-cleaners=#`
System Variable	`innodb_page_cleaners`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`4`
Minimum Value	`1`
Maximum Value	`64`

The number of page cleaner threads that flush dirty pages from buffer pool instances. Page cleaner threads perform flush list and LRU flushing. When there are multiple page cleaner threads, buffer pool flushing tasks for each buffer pool instance are dispatched to idle page cleaner threads. The `innodb_page_cleaners` default value is 4. If the number of page cleaner threads exceeds the number of buffer pool instances, `innodb_page_cleaners` is automatically set to the same value as `innodb_buffer_pool_instances`.

If your workload is write-IO bound when flushing dirty pages from buffer pool instances to data files, and if your system hardware has available capacity, increasing the number of page cleaner threads may help improve write-IO throughput.

Multithreaded page cleaner support extends to shutdown and recovery phases.

The `setpriority()` system call is used on Linux platforms where it is supported, and where the `mysqld` execution user is authorized to give `page_cleaner` threads priority over other MySQL and `InnoDB` threads to help page flushing keep pace with the current workload. `setpriority()` support is indicated by this `InnoDB` startup message:

```
[Note] InnoDB: If the mysqld execution user is authorized, page cleaner
thread priority can be changed. See the man page of setpriority().
```

For systems where server startup and shutdown is not managed by systemd, `mysqld` execution user authorization can be configured in `/etc/security/limits.conf`. For example, if `mysqld` is run under the `mysql` user, you can authorize the `mysql` user by adding these lines to `/etc/security/limits.conf`:

```
mysql             hard    nice       -20
mysql             soft    nice       -20
```

For systemd managed systems, the same can be achieved by specifying `LimitNICE=-20` in a localized systemd configuration file. For example, create a file named `override.conf` in `/etc/systemd/system/mysqld.service.d/override.conf` and add this entry:

```
[Service]
LimitNICE=-20
```

After creating or changing `override.conf`, reload the systemd configuration, then tell systemd to restart the MySQL service:

```
systemctl daemon-reload
systemctl restart mysqld  # RPM platforms
systemctl restart mysql   # Debian platforms
```

For more information about using a localized systemd configuration file, see Configuring systemd for MySQL.

After authorizing the `mysqld` execution user, use the `cat` command to verify the configured `Nice` limits for the `mysqld` process:

```
shell> cat /proc/mysqld_pid/limits | grep nice
Max nice priority         18446744073709551596 18446744073709551596
```

- `innodb_page_size`

Property	Value
Command-Line Format	`--innodb-page-size=#`
System Variable	`innodb_page_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`16384`

Valid Values	`4096`
	`8192`
	`16384`

Property	Value
	32768 65536

Specifies the page size for InnoDB tablespaces. Values can be specified in bytes or kilobytes. For example, a 16 kilobyte page size value can be specified as 16384, 16KB, or 16k.

innodb_page_size can only be configured prior to initializing the MySQL instance and cannot be changed afterward. If no value is specified, the instance is initialized using the default page size. See Section 15.8.1, “InnoDB Startup Configuration”.

For both 32KB and 64KB page sizes, the maximum row length is approximately 16000 bytes. ROW_FORMAT=COMPRESSED is not supported when innodb_page_size is set to 32KB or 64KB. For innodb_page_size=32KB, extent size is 2MB. For innodb_page_size=64KB, extent size is 4MB. innodb_log_buffer_size should be set to at least 16M (the default) when using 32KB or 64KB page sizes.

The default 16KB page size or larger is appropriate for a wide range of workloads, particularly for queries involving table scans and DML operations involving bulk updates. Smaller page sizes might be more efficient for OLTP workloads involving many small writes, where contention can be an issue when single pages contain many rows. Smaller pages might also be efficient with SSD storage devices, which typically use small block sizes. Keeping the InnoDB page size close to the storage device block size minimizes the amount of unchanged data that is rewritten to disk.

The minimum file size for the first system tablespace data file (ibdata1) differs depending on the innodb_page_size value. See the innodb_data_file_path option description for more information.

For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- innodb_parallel_read_threads

Property	Value
Command-Line Format	--innodb-parallel-read-threads=#
Introduced	8.0.14
System Variable	innodb_parallel_read_threads
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	4
Minimum Value	1
Maximum Value	256

Defines the number of threads that can be used for parallel clustered index reads. Parallel scanning of partitions is supported as of MySQL 8.0.17. Parallel read threads can improve CHECK TABLE performance. InnoDB reads the clustered index twice during a CHECK TABLE operation. The second read can be performed in parallel. This feature does not apply to secondary index scans. The innodb_parallel_read_threads session variable must be set to a value greater than 1 for parallel

clustered index reads to occur. The actual number of threads used to perform a parallel clustered index read is determined by the `innodb_parallel_read_threads` setting or the number of index subtrees to scan, whichever is smaller. The pages read into the buffer pool during the scan are kept at the tail of the buffer pool LRU list so that they can be discarded quickly when free buffer pool pages are required.

As of MySQL 8.0.17, the maximum number of parallel read threads (256) is the total number of threads for all client connections. If the thread limit is reached, connections fall back to using a single thread.

- `innodb_print_all_deadlocks`

Property	Value
Command-Line Format	`--innodb-print-all-deadlocks[={OFF|ON}]`
System Variable	`innodb_print_all_deadlocks`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

When this option is enabled, information about all deadlocks in `InnoDB` user transactions is recorded in the `mysqld` error log. Otherwise, you see information about only the last deadlock, using the `SHOW ENGINE INNODB STATUS` command. An occasional `InnoDB` deadlock is not necessarily an issue, because `InnoDB` detects the condition immediately and rolls back one of the transactions automatically. You might use this option to troubleshoot why deadlocks are occurring if an application does not have appropriate error-handling logic to detect the rollback and retry its operation. A large number of deadlocks might indicate the need to restructure transactions that issue DML or `SELECT ... FOR UPDATE` statements for multiple tables, so that each transaction accesses the tables in the same order, thus avoiding the deadlock condition.

For related information, see Section 15.7.5, “Deadlocks in InnoDB”.

- `innodb_print_ddl_logs`

Property	Value
Command-Line Format	`--innodb-print-ddl-logs[={OFF|ON}]`
Introduced	8.0.3
System Variable	`innodb_print_ddl_logs`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enabling this option causes MySQL to write DDL logs to `stderr`. For more information, see Viewing DDL Logs.

- `innodb_purge_batch_size`

Property	Value
Command-Line Format	`--innodb-purge-batch-size=#`
System Variable	`innodb_purge_batch_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`300`
Minimum Value	`1`
Maximum Value	`5000`

Defines the number of undo log pages that purge parses and processes in one batch from the history list. In a multithreaded purge configuration, the coordinator purge thread divides `innodb_purge_batch_size` by `innodb_purge_threads` and assigns that number of pages to each purge thread. The `innodb_purge_batch_size` option also defines the number of undo log pages that purge frees after every 128 iterations through the undo logs.

The `innodb_purge_batch_size` option is intended for advanced performance tuning in combination with the `innodb_purge_threads` setting. Most MySQL users need not change `innodb_purge_batch_size` from its default value.

For related information, see Section 15.8.9, “Purge Configuration”.

- `innodb_purge_threads`

Property	Value
Command-Line Format	`--innodb-purge-threads=#`
System Variable	`innodb_purge_threads`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`4`
Minimum Value	`1`
Maximum Value	`32`

The number of background threads devoted to the `InnoDB` purge operation. A minimum value of 1 signifies that the purge operation is always performed by a background thread, never as part of the master thread. Running the purge operation in one or more background threads helps reduce internal contention within `InnoDB`, improving scalability. Increasing the value to greater than 1 creates that many separate purge threads, which can improve efficiency on systems where DML operations are performed on multiple tables. The maximum is 32.

For related information, see Section 15.8.9, “Purge Configuration”.

- `innodb_purge_rseg_truncate_frequency`

Property	Value
Command-Line Format	`--innodb-purge-rseg-truncate-frequency=#`
System Variable	`innodb_purge_rseg_truncate_frequency`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`128`
Minimum Value	`1`
Maximum Value	`128`

Defines the frequency with which the purge system frees rollback segments in terms of the number of times that purge is invoked. An undo tablespace cannot be truncated until its rollback segments are freed. Normally, the purge system frees rollback segments once every 128 times that purge is invoked. The default value is 128. Reducing this value increases the frequency with which the purge thread frees rollback segments.

`innodb_purge_rseg_truncate_frequency` is intended for use with `innodb_undo_log_truncate`. For more information, see Truncating Undo Tablespaces.

- `innodb_random_read_ahead`

Property	Value
Command-Line Format	`--innodb-random-read-ahead[={OFF|ON}]`
System Variable	`innodb_random_read_ahead`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enables the random read-ahead technique for optimizing `InnoDB` I/O.

For details about performance considerations for different types of read-ahead requests, see Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_read_ahead_threshold`

Property	Value
Command-Line Format	`--innodb-read-ahead-threshold=#`
System Variable	`innodb_read_ahead_threshold`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	56
Minimum Value	0
Maximum Value	64

Controls the sensitivity of linear read-ahead that InnoDB uses to prefetch pages into the buffer pool. If InnoDB reads at least innodb_read_ahead_threshold pages sequentially from an extent (64 pages), it initiates an asynchronous read for the entire following extent. The permissible range of values is 0 to 64. A value of 0 disables read-ahead. For the default of 56, InnoDB must read at least 56 pages sequentially from an extent to initiate an asynchronous read for the following extent.

Knowing how many pages are read through the read-ahead mechanism, and how many of these pages are evicted from the buffer pool without ever being accessed, can be useful when fine-tuning the innodb_read_ahead_threshold setting. SHOW ENGINE INNODB STATUS output displays counter information from the Innodb_buffer_pool_read_ahead and Innodb_buffer_pool_read_ahead_evicted global status variables, which report the number of pages brought into the buffer pool by read-ahead requests, and the number of such pages evicted from the buffer pool without ever being accessed, respectively. The status variables report global values since the last server restart.

SHOW ENGINE INNODB STATUS also shows the rate at which the read-ahead pages are read and the rate at which such pages are evicted without being accessed. The per-second averages are based on the statistics collected since the last invocation of SHOW ENGINE INNODB STATUS and are displayed in the BUFFER POOL AND MEMORY section of the SHOW ENGINE INNODB STATUS output.

For more information, see Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- innodb_read_io_threads

Property	Value
Command-Line Format	--innodb-read-io-threads=#
System Variable	innodb_read_io_threads
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	4
Minimum Value	1
Maximum Value	64

The number of I/O threads for read operations in InnoDB. Its counterpart for write threads is innodb_write_io_threads. For more information, see Section 15.8.5, “Configuring the Number of Background InnoDB I/O Threads”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

Note

On Linux systems, running multiple MySQL servers (typically more than 12) with default settings for `innodb_read_io_threads`, `innodb_write_io_threads`, and the Linux `aio-max-nr` setting can exceed system limits. Ideally, increase the `aio-max-nr` setting; as a workaround, you might reduce the settings for one or both of the MySQL variables.

- `innodb_read_only`

Property	Value
Command-Line Format	`--innodb-read-only[={OFF|ON}]`
System Variable	`innodb_read_only`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Starts `InnoDB` in read-only mode. For distributing database applications or data sets on read-only media. Can also be used in data warehouses to share the same data directory between multiple instances. For more information, see Section 15.8.2, “Configuring InnoDB for Read-Only Operation”.

Previously, enabling the `innodb_read_only` system variable prevented creating and dropping tables only for the `InnoDB` storage engine. As of MySQL 8.0, enabling `innodb_read_only` prevents these operations for all storage engines. Table creation and drop operations for any storage engine modify data dictionary tables in the `mysql` system database, but those tables use the `InnoDB` storage engine and cannot be modified when `innodb_read_only` is enabled. The same principle applies to other table operations that require modifying data dictionary tables. Examples:

- If the `innodb_read_only` system variable is enabled, `ANALYZE TABLE` may fail because it cannot update statistics tables in the data dictionary, which use `InnoDB`. For `ANALYZE TABLE` operations that update the key distribution, failure may occur even if the operation updates the table itself (for example, if it is a `MyISAM` table). To obtain the updated distribution statistics, set `information_schema_stats_expiry=0`.
- `ALTER TABLE` *`tbl_name`* `ENGINE=`*`engine_name`* fails because it updates the storage engine designation, which is stored in the data dictionary.

In addition, other tables in the `mysql` system database use the `InnoDB` storage engine in MySQL 8.0. Making those tables read only results in restrictions on operations that modify them. Examples:

- Account-management statements such as `CREATE USER` and `GRANT` fail because the grant tables use `InnoDB`.
- The `INSTALL PLUGIN` and `UNINSTALL PLUGIN` plugin-management statements fail because the `plugin` table uses `InnoDB`.
- The `CREATE FUNCTION` and `DROP FUNCTION` UDF-management statements fail because the `func` table uses `InnoDB`.

- `innodb_redo_log_archive_dirs`

Property	Value
Command-Line Format	`--innodb-redo-log-archive-dirs`
Introduced	8.0.17
System Variable	`innodb_redo_log_archive_dirs`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`NULL`

Defines labeled directories where redo log archive files can be created. You can define multiple labeled directories in a semicolon-separated list. For example:

```
innodb_redo_log_archive_dirs='label1:/backups1;label2:/backups2'
```

A label can be any string of characters, with the exception of colons (:), which are not permitted. An empty label is also permitted, but the colon (:) is still required in this case.

A path must be specified, and the directory must exist. The path can contain colons (':'), but semicolons (;) are not permitted.

- `innodb_redo_log_encrypt`

Property	Value
Command-Line Format	`--innodb-redo-log-encrypt[={OFF|ON}]`
Introduced	8.0.1
System Variable	`innodb_redo_log_encrypt`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Controls encryption of redo log data for tables encrypted using the `InnoDB` data-at-rest encryption feature. Encryption of redo log data is disabled by default. For more information, see Redo Log Encryption.

- `innodb_replication_delay`

Property	Value
Command-Line Format	`--innodb-replication-delay=#`
System Variable	`innodb_replication_delay`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

The replication thread delay in milliseconds on a slave server if `innodb_thread_concurrency` is reached.

- `innodb_rollback_on_timeout`

Property	Value
Command-Line Format	`--innodb-rollback-on-timeout[={OFF|ON}]`
System Variable	`innodb_rollback_on_timeout`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

`InnoDB` rolls back only the last statement on a transaction timeout by default. If `--innodb-rollback-on-timeout` is specified, a transaction timeout causes `InnoDB` to abort and roll back the entire transaction.

Note

If the start-transaction statement was `START TRANSACTION` or `BEGIN` statement, rollback does not cancel that statement. Further SQL statements become part of the transaction until the occurrence of `COMMIT`, `ROLLBACK`, or some SQL statement that causes an implicit commit.

For more information, see Section 15.20.4, “InnoDB Error Handling”.

- `innodb_rollback_segments`

Property	Value
Command-Line Format	`--innodb-rollback-segments=#`
System Variable	`innodb_rollback_segments`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`128`
Minimum Value	`1`
Maximum Value	`128`

`innodb_rollback_segments` defines the number of rollback segments allocated to each undo tablespace and the global temporary tablespace for transactions that generate undo records. The number of transactions that each rollback segment supports depends on the `InnoDB` page size and the number of undo logs assigned to each transaction. For more information, see Section 15.6.6, “Undo Logs”.

For related information, see Section 15.3, “InnoDB Multi-Versioning”. For information about undo tablespaces, see Section 15.6.3.4, “Undo Tablespaces”.

- `innodb_scan_directories`

Property	Value
Command-Line Format	`--innodb-scan-directories=dir_name`
Introduced	8.0.2
Removed	8.0.4
System Variable	`innodb_scan_directories`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Directory name
Default Value	`NULL`

This variable was replaced by `innodb_directories` in MySQL 8.0.4.

- `innodb_saved_page_number_debug`

Property	Value
Command-Line Format	`--innodb-saved-page-number-debug=#`
System Variable	`innodb_saved_page_number_debug`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Maximum Value	`2**23-1`

Saves a page number. Setting the `innodb_fil_make_page_dirty_debug` option dirties the page defined by `innodb_saved_page_number_debug`. The `innodb_saved_page_number_debug` option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_sort_buffer_size`

Property	Value
Command-Line Format	`--innodb-sort-buffer-size=#`
System Variable	`innodb_sort_buffer_size`
Scope	Global

Property	Value
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	1048576
Minimum Value	65536
Maximum Value	67108864

Specifies the size of sort buffers used to sort data during creation of an InnoDB index. The specified size defines the amount of data that is read into memory for internal sorting and then written out to disk. This process is referred to as a “run”. During the merge phase, pairs of buffers of the specified size are read and merged. The larger the setting, the fewer runs and merges there are.

This sort area is only used for merge sorts during index creation, not during later index maintenance operations. Buffers are deallocated when index creation completes.

The value of this option also controls the amount by which the temporary log file is extended to record concurrent DML during online DDL operations.

Before this setting was made configurable, the size was hardcoded to 1048576 bytes (1MB), which remains the default.

During an ALTER TABLE or CREATE TABLE statement that creates an index, 3 buffers are allocated, each with a size defined by this option. Additionally, auxiliary pointers are allocated to rows in the sort buffer so that the sort can run on pointers (as opposed to moving rows during the sort operation).

For a typical sort operation, a formula such as this one can be used to estimate memory consumption:

```
(6 /*FTS_NUM_AUX_INDEX*/ * (3*@@GLOBAL.innodb_sort_buffer_size)
+ 2 * number_of_partitions * number_of_secondary_indexes_created
* (@@GLOBAL.innodb_sort_buffer_size/dict_index_get_min_size(index)*/)
* 8 /*64-bit sizeof *buf->tuples*/")
```

`@@GLOBAL.innodb_sort_buffer_size/dict_index_get_min_size(index)` indicates the maximum tuples held. `2 * (@@GLOBAL.innodb_sort_buffer_size/ *dict_index_get_min_size(index)*/) * 8 /*64-bit size of *buf->tuples*/` indicates auxiliary pointers allocated.

Note

For 32-bit, multiply by 4 instead of 8.

For parallel sorts on a full-text index, multiply by the innodb_ft_sort_pll_degree setting:

```
(6 /*FTS_NUM_AUX_INDEX*/ * @@GLOBAL.innodb_ft_sort_pll_degree)
```

- innodb_spin_wait_delay

Property	Value
Command-Line Format	--innodb-spin-wait-delay=#
System Variable	innodb_spin_wait_delay

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`6`
Minimum Value	`0`
Maximum Value (64-bit platforms, <= 8.0.13)	`2**64-1`
Maximum Value (32-bit platforms, <= 8.0.13)	`2**32-1`
Maximum Value (>= 8.0.14)	`1000`

The maximum delay between polls for a spin lock. The low-level implementation of this mechanism varies depending on the combination of hardware and operating system, so the delay does not correspond to a fixed time interval.

Can be used in combination with the `innodb_spin_wait_pause_multiplier` variable for greater control over the duration of spin-lock polling delays.

For more information, see Section 15.8.8, “Configuring Spin Lock Polling”.

- `innodb_spin_wait_pause_multiplier`

Property	Value
Command-Line Format	`--innodb-spin-wait-pause-multiplier=#`
Introduced	8.0.16
System Variable	`innodb_spin_wait_pause_multiplier`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`50`
Minimum Value	`1`
Maximum Value	`100`

Defines a multiplier value used to determine the number of PAUSE instructions in spin-wait loops that occur when a thread waits to acquire a mutex or rw-lock.

For more information, see Section 15.8.8, “Configuring Spin Lock Polling”.

- `innodb_stats_auto_recalc`

Property	Value
Command-Line Format	`--innodb-stats-auto-recalc[={OFF|ON}]`
System Variable	`innodb_stats_auto_recalc`

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

Causes InnoDB to automatically recalculate persistent statistics after the data in a table is changed substantially. The threshold value is 10% of the rows in the table. This setting applies to tables created when the innodb_stats_persistent option is enabled. Automatic statistics recalculation may also be configured by specifying STATS_PERSISTENT=1 in a CREATE TABLE or ALTER TABLE statement. The amount of data sampled to produce the statistics is controlled by the innodb_stats_persistent_sample_pages variable.

For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

- innodb_stats_include_delete_marked

Property	Value
Command-Line Format	--innodb-stats-include-delete-marked[={OFF|ON}]
Introduced	8.0.1
System Variable	innodb_stats_include_delete_marked
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

By default, InnoDB reads uncommitted data when calculating statistics. In the case of an uncommitted transaction that deletes rows from a table, InnoDB excludes records that are delete-marked when calculating row estimates and index statistics, which can lead to non-optimal execution plans for other transactions that are operating on the table concurrently using a transaction isolation level other than READ UNCOMMITTED. To avoid this scenario, innodb_stats_include_delete_marked can be enabled to ensure that InnoDB includes delete-marked records when calculating persistent optimizer statistics.

When innodb_stats_include_delete_marked is enabled, ANALYZE TABLE considers delete-marked records when recalculating statistics.

innodb_stats_include_delete_marked is a global setting that affects all InnoDB tables. It is only applicable to persistent optimizer statistics.

For related information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

- innodb_stats_method

Property	Value
Command-Line Format	--innodb-stats-method=value
System Variable	innodb_stats_method

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`nulls_equal`

Valid Values	`nulls_equal`
	`nulls_unequal`
	`nulls_ignored`

How the server treats `NULL` values when collecting statistics about the distribution of index values for `InnoDB` tables. Permitted values are `nulls_equal`, `nulls_unequal`, and `nulls_ignored`. For `nulls_equal`, all `NULL` index values are considered equal and form a single value group with a size equal to the number of `NULL` values. For `nulls_unequal`, `NULL` values are considered unequal, and each `NULL` forms a distinct value group of size 1. For `nulls_ignored`, `NULL` values are ignored.

The method used to generate table statistics influences how the optimizer chooses indexes for query execution, as described in Section 8.3.8, “InnoDB and MyISAM Index Statistics Collection”.

- `innodb_stats_on_metadata`

Property	Value
Command-Line Format	`--innodb-stats-on-metadata[={OFF|ON}]`
System Variable	`innodb_stats_on_metadata`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This option only applies when optimizer statistics are configured to be non-persistent. Optimizer statistics are not persisted to disk when `innodb_stats_persistent` is disabled or when individual tables are created or altered with `STATS_PERSISTENT=0`. For more information, see Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

When `innodb_stats_on_metadata` is enabled, `InnoDB` updates non-persistent statistics when metadata statements such as `SHOW TABLE STATUS` or when accessing the `INFORMATION_SCHEMA.TABLES` or `INFORMATION_SCHEMA.STATISTICS` tables. (These updates are similar to what happens for `ANALYZE TABLE`.) When disabled, `InnoDB` does not update statistics during these operations. Leaving the setting disabled can improve access speed for schemas that have a large number of tables or indexes. It can also improve the stability of execution plans for queries that involve `InnoDB` tables.

To change the setting, issue the statement `SET GLOBAL innodb_stats_on_metadata=`*`mode`*, where *`mode`* is either `ON` or `OFF` (or `1` or `0`). Changing the setting requires privileges sufficient to set global system variables (see Section 5.1.9.1, “System Variable Privileges”) and immediately affects the operation of all connections.

- `innodb_stats_persistent`

Property	Value
Command-Line Format	`--innodb-stats-persistent[={OFF|ON}]`
System Variable	`innodb_stats_persistent`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Specifies whether `InnoDB` index statistics are persisted to disk. Otherwise, statistics may be recalculated frequently which can lead to variations in query execution plans. This setting is stored with each table when the table is created. You can set `innodb_stats_persistent` at the global level before creating a table, or use the `STATS_PERSISTENT` clause of the `CREATE TABLE` and `ALTER TABLE` statements to override the system-wide setting and configure persistent statistics for individual tables.

For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

- `innodb_stats_persistent_sample_pages`

Property	Value
Command-Line Format	`--innodb-stats-persistent-sample-pages=#`
System Variable	`innodb_stats_persistent_sample_pages`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`20`

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by `ANALYZE TABLE`. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O during the execution of `ANALYZE TABLE` for an `InnoDB` table. For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

Note

Setting a high value for `innodb_stats_persistent_sample_pages` could result in lengthy `ANALYZE TABLE` execution time. To estimate the number of database pages accessed by `ANALYZE TABLE`, see Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

`innodb_stats_persistent_sample_pages` only applies when `innodb_stats_persistent` is enabled for a table; when `innodb_stats_persistent` is disabled, `innodb_stats_transient_sample_pages` applies instead.

- `innodb_stats_transient_sample_pages`

Property	Value
Command-Line Format	`--innodb-stats-transient-sample-pages=#`
System Variable	`innodb_stats_transient_sample_pages`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`8`

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by `ANALYZE TABLE`. The default value is 8. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O when opening an `InnoDB` table or recalculating statistics. For more information, see Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

Note

Setting a high value for `innodb_stats_transient_sample_pages` could result in lengthy `ANALYZE TABLE` execution time. To estimate the number of database pages accessed by `ANALYZE TABLE`, see Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

`innodb_stats_transient_sample_pages` only applies when `innodb_stats_persistent` is disabled for a table; when `innodb_stats_persistent` is enabled, `innodb_stats_persistent_sample_pages` applies instead. Takes the place of `innodb_stats_sample_pages`. For more information, see Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

- `innodb_status_output`

Property	Value
Command-Line Format	`--innodb-status-output[={OFF|ON}]`
System Variable	`innodb_status_output`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enables or disables periodic output for the standard `InnoDB` Monitor. Also used in combination with `innodb_status_output_locks` to enable or disable periodic output for the `InnoDB` Lock Monitor. For more information, see Section 15.16.2, “Enabling InnoDB Monitors”.

- `innodb_status_output_locks`

Property	Value
Command-Line Format	`--innodb-status-output-locks[={OFF|ON}]`

Property	Value
System Variable	`innodb_status_output_locks`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enables or disables the `InnoDB` Lock Monitor. When enabled, the `InnoDB` Lock Monitor prints additional information about locks in `SHOW ENGINE INNODB STATUS` output and in periodic output printed to the MySQL error log. Periodic output for the `InnoDB` Lock Monitor is printed as part of the standard `InnoDB` Monitor output. The standard `InnoDB` Monitor must therefore be enabled for the `InnoDB` Lock Monitor to print data to the MySQL error log periodically. For more information, see Section 15.16.2, “Enabling InnoDB Monitors”.

- `innodb_strict_mode`

Property	Value
Command-Line Format	`--innodb-strict-mode[={OFF|ON}]`
System Variable	`innodb_strict_mode`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

When `innodb_strict_mode` is enabled, `InnoDB` returns errors rather than warnings for certain conditions.

Strict mode helps guard against ignored typos and syntax errors in SQL, or other unintended consequences of various combinations of operational modes and SQL statements. When `innodb_strict_mode` is enabled, `InnoDB` raises error conditions in certain cases, rather than issuing a warning and processing the specified statement (perhaps with unintended behavior). This is analogous to `sql_mode` in MySQL, which controls what SQL syntax MySQL accepts, and determines whether it silently ignores errors, or validates input syntax and data values.

The `innodb_strict_mode` setting affects the handling of syntax errors for `CREATE TABLE`, `ALTER TABLE`, `CREATE INDEX`, and `OPTIMIZE TABLE` statements. `innodb_strict_mode` also enables a record size check, so that an `INSERT` or `UPDATE` never fails due to the record being too large for the selected page size.

Oracle recommends enabling `innodb_strict_mode` when using `ROW_FORMAT` and `KEY_BLOCK_SIZE` clauses in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements. When `innodb_strict_mode` is disabled, `InnoDB` ignores conflicting clauses and creates the table or index with only a warning in the message log. The resulting table might have different characteristics than intended, such as lack of compression support when attempting to create a compressed table. When `innodb_strict_mode` is enabled, such problems generate an immediate error and the table or index is not created.

You can enable or disable `innodb_strict_mode` on the command line when starting `mysqld`, or in a MySQL configuration file. You can also enable or disable `innodb_strict_mode` at runtime with the statement `SET [GLOBAL|SESSION] innodb_strict_mode=mode`, where *`mode`* is either `ON` or `OFF`. Changing the `GLOBAL` setting requires privileges sufficient to set global system variables (see Section 5.1.9.1, "System Variable Privileges") and affects the operation of all clients that subsequently connect. Any client can change the `SESSION` setting for `innodb_strict_mode`, and the setting affects only that client.

`innodb_strict_mode` is not applicable to general tablespaces. Tablespace management rules for general tablespaces are strictly enforced independently of `innodb_strict_mode`. For more information, see Section 13.1.21, "CREATE TABLESPACE Syntax".

- `innodb_sync_array_size`

Property	Value
Command-Line Format	`--innodb-sync-array-size=#`
System Variable	`innodb_sync_array_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`1024`

Defines the size of the mutex/lock wait array. Increasing the value splits the internal data structure used to coordinate threads, for higher concurrency in workloads with large numbers of waiting threads. This setting must be configured when the MySQL instance is starting up, and cannot be changed afterward. Increasing the value is recommended for workloads that frequently produce a large number of waiting threads, typically greater than 768.

- `innodb_sync_spin_loops`

Property	Value
Command-Line Format	`--innodb-sync-spin-loops=#`
System Variable	`innodb_sync_spin_loops`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`30`
Minimum Value	`0`
Maximum Value	`4294967295`

The number of times a thread waits for an `InnoDB` mutex to be freed before the thread is suspended.

- `innodb_sync_debug`

Property	Value
Command-Line Format	`--innodb-sync-debug[={OFF|ON}]`
System Variable	`innodb_sync_debug`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enables sync debug checking for the `InnoDB` storage engine. This option is only available if debugging support is compiled in using the `WITH_DEBUG` `CMake` option.

- `innodb_table_locks`

Property	Value
Command-Line Format	`--innodb-table-locks[={OFF|ON}]`
System Variable	`innodb_table_locks`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

If `autocommit = 0`, `InnoDB` honors `LOCK TABLES`; MySQL does not return from `LOCK TABLES ... WRITE` until all other threads have released all their locks to the table. The default value of `innodb_table_locks` is 1, which means that `LOCK TABLES` causes InnoDB to lock a table internally if `autocommit = 0`.

In MySQL 8.0, `innodb_table_locks = 0` has no effect for tables locked explicitly with `LOCK TABLES ... WRITE`. It does have an effect for tables locked for read or write by `LOCK TABLES ... WRITE` implicitly (for example, through triggers) or by `LOCK TABLES ... READ`.

For related information, see Section 15.7, “InnoDB Locking and Transaction Model”.

- `innodb_temp_data_file_path`

Property	Value
Command-Line Format	`--innodb-temp-data-file-path=file_name`
System Variable	`innodb_temp_data_file_path`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`ibtmp1:12M:autoextend`

Defines the relative path, name, size, and attributes of global temporary tablespace data files. The global temporary tablespace stores rollback segments for changes made to user-created temporary tables.

If no value is specified for `innodb_temp_data_file_path`, the default behavior is to create a single auto-extending data file named `ibtmp1` in the `innodb_data_home_dir` directory. The initial file size is slightly larger than 12MB.

The syntax for a global temporary tablespace data file specification includes the file name, file size, and `autoextend` and `max` attributes:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

The global temporary tablespace data file cannot have the same name as another `InnoDB` data file. Any inability or error creating the global temporary tablespace data file is treated as fatal and server startup is refused.

File sizes are specified in KB, MB, or GB by appending `K`, `M` or `G` to the size value. The sum of file sizes must be slightly larger than 12MB.

The size limit of individual files is determined by the operating system. File size can be more than 4GB on operating systems that support large files. Use of raw disk partitions for global temporary tablespace data files is not supported.

The `autoextend` and `max` attributes can be used only for the data file specified last in the `innodb_temp_data_file_path` setting. For example:

```
[mysqld]
innodb_temp_data_file_path=ibtmp1:50M;ibtmp2:12M:autoextend:max:500MB
```

The `autoextend` option causes the data file to automatically increase in size when it runs out of free space. The `autoextend` increment is 64MB by default. To modify the increment, change the `innodb_autoextend_increment` variable setting.

The directory path for global temporary tablespace data files is formed by concatenating the paths defined by `innodb_data_home_dir` and `innodb_temp_data_file_path`.

Before running `InnoDB` in read-only mode, set `innodb_temp_data_file_path` to a location outside of the data directory. The path must be relative to the data directory. For example:

```
--innodb-temp-data-file-path=../../../tmp/ibtmp1:12M:autoextend
```

For more information, see Global Temporary Tablespace.

- `innodb_temp_tablespaces_dir`

Property	Value
Command-Line Format	`--innodb-temp-tablespaces-dir=dir_name`
Introduced	8.0.13
System Variable	`innodb_temp_tablespaces_dir`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	Directory name
Default Value	`#innodb_temp`

Defines the location where `InnoDB` creates a pool of session temporary tablespaces at startup. The default location is the `#innodb_temp` directory in the data directory. A fully qualified path or path relative to the data directory is permitted.

As of MySQL 8.0.16, session temporary tablespaces always store user-created temporary tables and internal temporary tables created by the optimizer using `InnoDB`. (Previously, the on-disk storage engine for internal temporary tables was determined by the `internal_tmp_disk_storage_engine` system variable, which is no longer supported. See Storage Engine for On-Disk Internal Temporary Tables.)

For more information, see Session Temporary Tablespaces.

- `innodb_thread_concurrency`

Property	Value
Command-Line Format	`--innodb-thread-concurrency=#`
System Variable	`innodb_thread_concurrency`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1000`

`InnoDB` tries to keep the number of operating system threads concurrently inside `InnoDB` less than or equal to the limit given by this variable (`InnoDB` uses operating system threads to process user transactions). Once the number of threads reaches this limit, additional threads are placed into a wait state within a “First In, First Out” (FIFO) queue for execution. Threads waiting for locks are not counted in the number of concurrently executing threads.

The range of this variable is 0 to 1000. A value of 0 (the default) is interpreted as infinite concurrency (no concurrency checking). Disabling thread concurrency checking enables `InnoDB` to create as many threads as it needs. A value of 0 also disables the `queries inside InnoDB` and `queries in queue counters` in the `ROW OPERATIONS` section of `SHOW ENGINE INNODB STATUS` output.

Consider setting this variable if your MySQL instance shares CPU resources with other applications, or if your workload or number of concurrent users is growing. The correct setting depends on workload, computing environment, and the version of MySQL that you are running. You will need to test a range of values to determine the setting that provides the best performance. `innodb_thread_concurrency` is a dynamic variable, which allows you to experiment with different settings on a live test system. If a particular setting performs poorly, you can quickly set `innodb_thread_concurrency` back to 0.

Use the following guidelines to help find and maintain an appropriate setting:

- If the number of concurrent user threads for a workload is less than 64, set `innodb_thread_concurrency=0`.
- If your workload is consistently heavy or occasionally spikes, start by setting `innodb_thread_concurrency=128` and then lowering the value to 96, 80, 64, and so on, until you find the number of threads that provides the best performance. Suppose that your system typically has 40 to 50 users, but periodically the number increases to 60, 70, or even 200. You find that performance is stable at 80 concurrent users but starts to show a regression above this number. In this case, you would set `innodb_thread_concurrency=80` to avoid impacting performance.
- If you do not want `InnoDB` to use more than a certain number of virtual CPUs for user threads (20 virtual CPUs, for example), set `innodb_thread_concurrency` to this number (or possibly lower, depending on performance results). If your goal is to isolate MySQL from other applications, you may consider binding the `mysqld` process exclusively to the virtual CPUs. Be aware, however, that exclusive binding could result in non-optimal hardware usage if the `mysqld` process is not consistently busy. In this case, you might bind the `mysqld` process to the virtual CPUs but also allow other applications to use some or all of the virtual CPUs.

  Note

  From an operating system perspective, using a resource management solution to manage how CPU time is shared among applications may be preferable to binding the `mysqld` process. For example, you could assign 90% of virtual CPU time to a given application while other critical processes *are not* running, and scale that value back to 40% when other critical processes *are* running.

- `innodb_thread_concurrency` values that are too high can cause performance regression due to increased contention on system internals and resources.
- In some cases, the optimal `innodb_thread_concurrency` setting can be smaller than the number of virtual CPUs.
- Monitor and analyze your system regularly. Changes to workload, number of users, or computing environment may require that you adjust the `innodb_thread_concurrency` setting.

For related information, see Section 15.8.4, “Configuring Thread Concurrency for InnoDB”.

- `innodb_thread_sleep_delay`

Property	Value
Command-Line Format	`--innodb-thread-sleep-delay=#`
System Variable	`innodb_thread_sleep_delay`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10000`
Minimum Value	`0`
Maximum Value	`1000000`

How long InnoDB threads sleep before joining the InnoDB queue, in microseconds. The default value is 10000. A value of 0 disables sleep. You can set innodb_adaptive_max_sleep_delay to the highest value you would allow for innodb_thread_sleep_delay, and InnoDB automatically adjusts innodb_thread_sleep_delay up or down depending on current thread-scheduling activity. This dynamic adjustment helps the thread scheduling mechanism to work smoothly during times when the system is lightly loaded or when it is operating near full capacity.

For more information, see Section 15.8.4, “Configuring Thread Concurrency for InnoDB”.

- innodb_tmpdir

Property	Value
Command-Line Format	--innodb-tmpdir=dir_name
System Variable	innodb_tmpdir
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Directory name
Default Value	NULL

Used to define an alternate directory for temporary sort files created during online ALTER TABLE operations that rebuild the table.

Online ALTER TABLE operations that rebuild the table also create an *intermediate* table file in the same directory as the original table. The innodb_tmpdir option is not applicable to intermediate table files.

A valid value is any directory path other than the MySQL data directory path. If the value is NULL (the default), temporary files are created MySQL temporary directory ($TMPDIR on Unix, %TEMP% on Windows, or the directory specified by the --tmpdir configuration option). If a directory is specified, existence of the directory and permissions are only checked when innodb_tmpdir is configured using a SET statement. If a symlink is provided in a directory string, the symlink is resolved and stored as an absolute path. The path should not exceed 512 bytes. An online ALTER TABLE operation reports an error if innodb_tmpdir is set to an invalid directory. innodb_tmpdir overrides the MySQL tmpdir setting but only for online ALTER TABLE operations.

The FILE privilege is required to configure innodb_tmpdir.

The innodb_tmpdir option was introduced to help avoid overflowing a temporary file directory located on a tmpfs file system. Such overflows could occur as a result of large temporary sort files created during online ALTER TABLE operations that rebuild the table.

In replication environments, only consider replicating the innodb_tmpdir setting if all servers have the same operating system environment. Otherwise, replicating the innodb_tmpdir setting could result in a replication failure when running online ALTER TABLE operations that rebuild the table. If server operating environments differ, it is recommended that you configure innodb_tmpdir on each server individually.

For more information, see Section 15.12.3, “Online DDL Space Requirements”. For information about online ALTER TABLE operations, see Section 15.12, “InnoDB and Online DDL”.

- innodb_trx_purge_view_update_only_debug

Property	Value
Command-Line Format	`--innodb-trx-purge-view-update-only-debug[={OFF|ON}]`
System Variable	`innodb_trx_purge_view_update_only_debug`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Pauses purging of delete-marked records while allowing the purge view to be updated. This option artificially creates a situation in which the purge view is updated but purges have not yet been performed. This option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_trx_rseg_n_slots_debug`

Property	Value
Command-Line Format	`--innodb-trx-rseg-n-slots-debug=#`
System Variable	`innodb_trx_rseg_n_slots_debug`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Maximum Value	`1024`

Sets a debug flag that limits `TRX_RSEG_N_SLOTS` to a given value for the `trx_rsegf_undo_find_free` function that looks for free slots for undo log segments. This option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_undo_directory`

Property	Value
Command-Line Format	`--innodb-undo-directory=dir_name`
System Variable	`innodb_undo_directory`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

The path where `InnoDB` creates undo tablespaces. Typically used to place undo tablespaces on a different storage device.

There is no default value (it is NULL). If the `innodb_undo_directory` variable is undefined, undo tablespaces are created in the data directory.

The default undo tablespaces (`innodb_undo_001` and `innodb_undo_002`) created when the MySQL instance is initialized always reside in the directory defined by the `innodb_undo_directory` variable.

Undo tablespaces created using `CREATE UNDO TABLESPACE` syntax are created in the directory defined by the `innodb_undo_directory` variable if a different path is not specified.

For more information, see Section 15.6.3.4, “Undo Tablespaces”.

- `innodb_undo_log_encrypt`

Property	Value
Command-Line Format	`--innodb-undo-log-encrypt[={OFF|ON}]`
Introduced	8.0.1
System Variable	`innodb_undo_log_encrypt`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Controls encryption of undo log data for tables encrypted using the `InnoDB` data-at-rest encryption feature. Only applies to undo logs that reside in separate undo tablespaces. See Section 15.6.3.4, “Undo Tablespaces”. Encryption is not supported for undo log data that resides in the system tablespace. For more information, see Undo Log Encryption.

- `innodb_undo_log_truncate`

Property	Value
Command-Line Format	`--innodb-undo-log-truncate[={OFF|ON}]`
System Variable	`innodb_undo_log_truncate`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value (>= 8.0.2)	`ON`
Default Value (<= 8.0.1)	`OFF`

When enabled, undo tablespaces that exceed the threshold value defined by `innodb_max_undo_log_size` are marked for truncation. Only undo tablespaces can be truncated. Truncating undo logs that reside in the system tablespace is not supported. For truncation to occur, there must be at least two undo tablespaces.

The `innodb_purge_rseg_truncate_frequency` variable can be used to expedite truncation of undo tablespaces.

For more information, see Truncating Undo Tablespaces.

- `innodb_undo_logs`

Property	Value
Command-Line Format	`--innodb-undo-logs=#`
Deprecated	Yes (removed in 8.0.2)
System Variable	`innodb_undo_logs`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`128`
Minimum Value	`1`
Maximum Value	`128`

Note

`innodb_undo_logs` was removed in MySQL 8.0.2.

The `innodb_undo_logs` option is an alias for `innodb_rollback_segments`. For more information, see the description of `innodb_rollback_segments`.

- `innodb_undo_tablespaces`

Property	Value
Command-Line Format	`--innodb-undo-tablespaces=#`
Deprecated	8.0.4
System Variable	`innodb_undo_tablespaces`
Scope	Global
Dynamic (>= 8.0.2)	Yes
Dynamic (<= 8.0.1)	No
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.2)	`2`
Default Value (<= 8.0.1)	`0`
Minimum Value (>= 8.0.3)	`2`
Minimum Value (<= 8.0.2)	`0`
Maximum Value (>= 8.0.2)	`127`
Maximum Value (<= 8.0.1)	`95`

Defines the number of undo tablespaces used by `InnoDB`. The default and minimum value is 2.

Note

The `innodb_undo_tablespaces` variable is deprecated and is no longer configurable as of MySQL 8.0.14. It will be removed in a future release.

For more information, see Section 15.6.3.4, “Undo Tablespaces”.

- `innodb_use_native_aio`

Property	Value
Command-Line Format	`--innodb-use-native-aio[={OFF|ON}]`
System Variable	`innodb_use_native_aio`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Specifies whether to use the Linux asynchronous I/O subsystem. This variable applies to Linux systems only, and cannot be changed while the server is running. Normally, you do not need to configure this option, because it is enabled by default.

The asynchronous I/O capability that `InnoDB` has on Windows systems is available on Linux systems. (Other Unix-like systems continue to use synchronous I/O calls.) This feature improves the scalability of heavily I/O-bound systems, which typically show many pending reads/writes in `SHOW ENGINE INNODB STATUS\G` output.

Running with a large number of `InnoDB` I/O threads, and especially running multiple such instances on the same server machine, can exceed capacity limits on Linux systems. In this case, you may receive the following error:

```
EAGAIN: The specified maxevents exceeds the user's limit of available events.
```

You can typically address this error by writing a higher limit to `/proc/sys/fs/aio-max-nr`.

However, if a problem with the asynchronous I/O subsystem in the OS prevents `InnoDB` from starting, you can start the server with `innodb_use_native_aio=0`. This option may also be disabled automatically during startup if `InnoDB` detects a potential problem such as a combination of `tmpdir` location, `tmpfs` file system, and Linux kernel that does not support AIO on `tmpfs`.

For more information, see Section 15.8.6, “Using Asynchronous I/O on Linux”.

- `innodb_version`

The `InnoDB` version number. In MySQL 8.0, separate version numbering for `InnoDB` does not apply and this value is the same the `version` number of the server.

- `innodb_write_io_threads`

Property	Value
Command-Line Format	`--innodb-write-io-threads=#`
System Variable	`innodb_write_io_threads`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	4
Minimum Value	1
Maximum Value	64

The number of I/O threads for write operations in InnoDB. The default value is 4. Its counterpart for read threads is innodb_read_io_threads. For more information, see Section 15.8.5, “Configuring the Number of Background InnoDB I/O Threads”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

Note

On Linux systems, running multiple MySQL servers (typically more than 12) with default settings for innodb_read_io_threads, innodb_write_io_threads, and the Linux aio-max-nr setting can exceed system limits. Ideally, increase the aio-max-nr setting; as a workaround, you might reduce the settings for one or both of the MySQL variables.

Also take into consideration the value of sync_binlog, which controls synchronization of the binary log to disk.

For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

15.14 InnoDB INFORMATION_SCHEMA Tables

This section provides information and usage examples for InnoDB INFORMATION_SCHEMA tables.

InnoDB INFORMATION_SCHEMA tables provide metadata, status information, and statistics about various aspects of the InnoDB storage engine. You can view a list of InnoDB INFORMATION_SCHEMA tables by issuing a SHOW TABLES statement on the INFORMATION_SCHEMA database:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB%';
```

For table definitions, see Section 25.39, “INFORMATION_SCHEMA InnoDB Tables”. For general information regarding the MySQL INFORMATION_SCHEMA database, see Chapter 25, *INFORMATION_SCHEMA Tables*.

15.14.1 InnoDB INFORMATION_SCHEMA Tables about Compression

There are two pairs of InnoDB INFORMATION_SCHEMA tables about compression that can provide insight into how well compression is working overall:

- INNODB_CMP and INNODB_CMP_RESET provide information about the number of compression operations and the amount of time spent performing compression.
- INNODB_CMPMEM and INNODB_CMP_RESET provide information about the way memory is allocated for compression.

15.14.1.1 INNODB_CMP and INNODB_CMP_RESET

The INNODB_CMP and INNODB_CMP_RESET tables provide status information about operations related to compressed tables, which are described in Section 15.9, “InnoDB Table and Page Compression”. The PAGE_SIZE column reports the compressed page size.

These two tables have identical contents, but reading from INNODB_CMP_RESET resets the statistics on compression and uncompression operations. For example, if you archive the output of INNODB_CMP_RESET every 60 minutes, you see the statistics for each hourly period. If you monitor the output of INNODB_CMP (making sure never to read INNODB_CMP_RESET), you see the cumulative statistics since InnoDB was started.

For the table definition, see Section 25.39.5, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”.

15.14.1.2 INNODB_CMPMEM and INNODB_CMPMEM_RESET

The INNODB_CMPMEM and INNODB_CMPMEM_RESET tables provide status information about compressed pages that reside in the buffer pool. Please consult Section 15.9, “InnoDB Table and Page Compression” for further information on compressed tables and the use of the buffer pool. The INNODB_CMP and INNODB_CMP_RESET tables should provide more useful statistics on compression.

Internal Details

InnoDB uses a buddy allocator system to manage memory allocated to pages of various sizes, from 1KB to 16KB. Each row of the two tables described here corresponds to a single page size.

The INNODB_CMPMEM and INNODB_CMPMEM_RESET tables have identical contents, but reading from INNODB_CMPMEM_RESET resets the statistics on relocation operations. For example, if every 60 minutes you archived the output of INNODB_CMPMEM_RESET, it would show the hourly statistics. If you never read INNODB_CMPMEM_RESET and monitored the output of INNODB_CMPMEM instead, it would show the cumulative statistics since InnoDB was started.

For the table definition, see Section 25.39.6, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”.

15.14.1.3 Using the Compression Information Schema Tables

Example 15.1 Using the Compression Information Schema Tables

The following is sample output from a database that contains compressed tables (see Section 15.9, “InnoDB Table and Page Compression”, INNODB_CMP, INNODB_CMP_PER_INDEX, and INNODB_CMPMEM).

The following table shows the contents of INFORMATION_SCHEMA.INNODB_CMP under a light workload. The only compressed page size that the buffer pool contains is 8K. Compressing or uncompressing pages has consumed less than a second since the time the statistics were reset, because the columns COMPRESS_TIME and UNCOMPRESS_TIME are zero.

page size	compress ops	compress ops ok	compress time	uncompress ops	uncompress time
1024	0	0	0	0	0
2048	0	0	0	0	0
4096	0	0	0	0	0
8192	1048	921	0	61	0
16384	0	0	0	0	0

According to INNODB_CMPMEM, there are 6169 compressed 8KB pages in the buffer pool. The only other allocated block size is 64 bytes. The smallest PAGE_SIZE in INNODB_CMPMEM is used for block descriptors of those compressed pages for which no uncompressed page exists in the buffer pool. We see that there are 5910 such pages. Indirectly, we see that 259 (6169-5910) compressed pages also exist in the buffer pool in uncompressed form.

The following table shows the contents of INFORMATION_SCHEMA.INNODB_CMPMEM under a light workload. Some memory is unusable due to fragmentation of the memory allocator for compressed pages: SUM(PAGE_SIZE*PAGES_FREE)=6784. This is because small memory allocation requests are fulfilled by splitting bigger blocks, starting from the 16K blocks that are allocated from the main buffer pool, using the buddy allocation system. The fragmentation is this low because some allocated blocks have been relocated (copied) to form bigger adjacent free blocks. This copying of SUM(PAGE_SIZE*RELOCATION_OPS) bytes has consumed less than a second (SUM(RELOCATION_TIME)=0).

page size	pages used	pages free	relocation ops	relocation time
64	5910	0	2436	0
128	0	1	0	0
256	0	0	0	0
512	0	1	0	0
1024	0	0	0	0
2048	0	1	0	0
4096	0	1	0	0
8192	6169	0	5	0
16384	0	0	0	0

15.14.2 InnoDB INFORMATION_SCHEMA Transaction and Locking Information

Note

This section describes locking information as exposed by the Performance Schema data_locks and data_lock_waits tables, which supersede the INFORMATION_SCHEMA INNODB_LOCKS and INNODB_LOCK_WAITS tables in MySQL 8.0. For similar discussion written in terms of the older INFORMATION_SCHEMA tables, see InnoDB INFORMATION_SCHEMA Transaction and Locking Information in MySQL 5.7 Reference Manual.

One INFORMATION_SCHEMA table and two Performance Schema tables enable you to monitor InnoDB transactions and diagnose potential locking problems:

- INNODB_TRX: This INFORMATION_SCHEMA table provides information about every transaction currently executing inside InnoDB, including the transaction state (for example, whether it is running or waiting for a lock), when the transaction started, and the particular SQL statement the transaction is executing.
- data_locks: This Performance Schema table contains a row for each hold lock and each lock request that is blocked waiting for a held lock to be released:
  - There is one row for each held lock, whatever the state of the transaction that holds the lock (INNODB_TRX.TRX_STATE is RUNNING, LOCK WAIT, ROLLING BACK or COMMITTING).
  - Each transaction in InnoDB that is waiting for another transaction to release a lock (INNODB_TRX.TRX_STATE is LOCK WAIT) is blocked by exactly one blocking lock request. That

blocking lock request is for a row or table lock held by another transaction in an incompatible mode. A lock request always has a mode that is incompatible with the mode of the held lock that blocks the request (read vs. write, shared vs. exclusive).

The blocked transaction cannot proceed until the other transaction commits or rolls back, thereby releasing the requested lock. For every blocked transaction, `data_locks` contains one row that describes each lock the transaction has requested, and for which it is waiting.

- `data_lock_waits`: This Performance Schema table indicates which transactions are waiting for a given lock, or for which lock a given transaction is waiting. This table contains one or more rows for each blocked transaction, indicating the lock it has requested and any locks that are blocking that request. The `REQUESTING_ENGINE_LOCK_ID` value refers to the lock requested by a transaction, and the `BLOCKING_ENGINE_LOCK_ID` value refers to the lock (held by another transaction) that prevents the first transaction from proceeding. For any given blocked transaction, all rows in `data_lock_waits` have the same value for `REQUESTING_ENGINE_LOCK_ID` and different values for `BLOCKING_ENGINE_LOCK_ID`.

For more information about the preceding tables, see Section 25.39.29, “The INFORMATION_SCHEMA INNODB_TRX Table”, Section 26.12.12.1, “The data_locks Table”, and Section 26.12.12.2, “The data_lock_waits Table”.

15.14.2.1 Using InnoDB Transaction and Locking Information

Note

This section describes locking information as exposed by the Performance Schema `data_locks` and `data_lock_waits` tables, which supersede the `INFORMATION_SCHEMA` `INNODB_LOCKS` and `INNODB_LOCK_WAITS` tables in MySQL 8.0. For similar discussion written in terms of the older `INFORMATION_SCHEMA` tables, see Using InnoDB Transaction and Locking Information in MySQL 5.7 Reference Manual.

Identifying Blocking Transactions

It is sometimes helpful to identify which transaction blocks another. The tables that contain information about `InnoDB` transactions and data locks enable you to determine which transaction is waiting for another, and which resource is being requested. (For descriptions of these tables, see Section 15.14.2, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”.)

Suppose that three sessions are running concurrently. Each session corresponds to a MySQL thread, and executes one transaction after another. Consider the state of the system when these sessions have issued the following statements, but none has yet committed its transaction:

- Session A:

```
BEGIN;
SELECT a FROM t FOR UPDATE;
SELECT SLEEP(100);
```

- Session B:

```
SELECT b FROM t FOR UPDATE;
```

- Session C:

```
```

```
SELECT c FROM t FOR UPDATE;
```

In this scenario, use the following query to see which transactions are waiting and which transactions are blocking them:

```
SELECT
  r.trx_id waiting_trx_id,
  r.trx_mysql_thread_id waiting_thread,
  r.trx_query waiting_query,
  b.trx_id blocking_trx_id,
  b.trx_mysql_thread_id blocking_thread,
  b.trx_query blocking_query
FROM       performance_schema.data_lock_waits w
INNER JOIN information_schema.innodb_trx b
  ON b.trx_id = w.blocking_engine_transaction_id
INNER JOIN information_schema.innodb_trx r
  ON r.trx_id = w.requesting_engine_transaction_id;
```

Or, more simply, use the `sys` schema `innodb_lock_waits` view:

```
SELECT
  waiting_trx_id,
  waiting_pid,
  waiting_query,
  blocking_trx_id,
  blocking_pid,
  blocking_query
FROM sys.innodb_lock_waits;
```

If a NULL value is reported for the blocking query, see Identifying a Blocking Query After the Issuing Session Becomes Idle.

waiting trx id	waiting thread	waiting query	blocking trx id	blocking thread	blocking query
A4	6	SELECT b FROM t FOR UPDATE	A3	5	SELECT SLEEP(100)
A5	7	SELECT c FROM t FOR UPDATE	A3	5	SELECT SLEEP(100)
A5	7	SELECT c FROM t FOR UPDATE	A4	6	SELECT b FROM t FOR UPDATE

In the preceding table, you can identify sessions by the “waiting query” or “blocking query” columns. As you can see:

- Session B (trx id A4, thread 6) and Session C (trx id A5, thread 7) are both waiting for Session A (trx id A3, thread 5).
- Session C is waiting for Session B as well as Session A.

You can see the underlying data in the `INFORMATION_SCHEMA INNODB_TRX` table and Performance Schema `data_locks` and `data_lock_waits` tables.

The following table shows some sample contents of the `INNODB_TRX` table.

trx id	trx state	trx started
A3	RUNNING	2008-01-15 16:44:54
A4	LOCK WAIT	2008-01-15 16:45:09
A5	LOCK WAIT	2008-01-15 16:45:14

The following table shows some sample contents of the `data_locks` table.

lock id	lock trx id	lock mode	lock type
A3:1:3:2	A3	X	RECORD
A4:1:3:2	A4	X	RECORD
A5:1:3:2	A5	X	RECORD

The following table shows some sample contents of the `data_lock_waits` table.

requesting trx id	requested lock id	blocking trx id	blocking lock id
A4	A4:1:3:2	A3	A3:1:3:2
A5	A5:1:3:2	A3	A3:1:3:2
A5	A5:1:3:2	A4	A4:1:3:2

Identifying a Blocking Query After the Issuing Session Becomes Idle

When identifying blocking transactions, a NULL value is reported for the blocking query if the session that issued the query has become idle. In this case, use the following steps to determine the blocking query:

1. Identify the processlist ID of the blocking transaction. In the `sys.innodb_lock_waits` table, the processlist ID of the blocking transaction is the `blocking_pid` value.
2. Using the `blocking_pid`, query the MySQL Performance Schema `threads` table to determine the `THREAD_ID` of the blocking transaction. For example, if the `blocking_pid` is 6, issue this query:

```
SELECT THREAD_ID FROM performance_schema.threads WHERE PROCESSLIST_ID = 6;
```

3. Using the `THREAD_ID`, query the Performance Schema `events_statements_current` table to determine the last query executed by the thread. For example, if the `THREAD_ID` is 28, issue this query:

```
SELECT THREAD_ID, SQL_TEXT FROM performance_schema.events_statements_current
WHERE THREAD_ID = 28\G
```

4. If the last query executed by the thread is not enough information to determine why a lock is held, you can query the Performance Schema `events_statements_history` table to view the last 10 statements executed by the thread.

```
SELECT THREAD_ID, SQL_TEXT FROM performance_schema.events_statements_history
WHERE THREAD_ID = 28 ORDER BY EVENT_ID;
```

Correlating InnoDB Transactions with MySQL Sessions

Sometimes it is useful to correlate internal `InnoDB` locking information with the session-level information maintained by MySQL. For example, you might like to know, for a given `InnoDB` transaction ID, the corresponding MySQL session ID and name of the session that may be holding a lock, and thus blocking other transactions.

The following output from the `INFORMATION_SCHEMA INNODB_TRX` table and Performance Schema `data_locks` and `data_lock_waits` tables is taken from a somewhat loaded system. As can be seen, there are several transactions running.

The following `data_locks` and `data_lock_waits` tables show that:

- Transaction `77F` (executing an `INSERT`) is waiting for transactions `77E`, `77D`, and `77B` to commit.
- Transaction `77E` (executing an `INSERT`) is waiting for transactions `77D` and `77B` to commit.
- Transaction `77D` (executing an `INSERT`) is waiting for transaction `77B` to commit.
- Transaction `77B` (executing an `INSERT`) is waiting for transaction `77A` to commit.
- Transaction `77A` is running, currently executing `SELECT`.
- Transaction `E56` (executing an `INSERT`) is waiting for transaction `E55` to commit.
- Transaction `E55` (executing an `INSERT`) is waiting for transaction `19C` to commit.
- Transaction `19C` is running, currently executing an `INSERT`.

Note

There may be inconsistencies between queries shown in the `INFORMATION_SCHEMA PROCESSLIST` and `INNODB_TRX` tables. For an explanation, see Section 15.14.2.3, "Persistence and Consistency of InnoDB Transaction and Locking Information".

The following table shows the contents of the `PROCESSLIST` table for a system running a heavy workload.

ID	USER	HOST	DB	COMMAND	TIME	STATE
384	root	localhost	test	Query	10	update
257	root	localhost	test	Query	3	update
130	root	localhost	test	Query	0	update
61	root	localhost	test	Query	1	update
8	root	localhost	test	Query	1	update
4	root	localhost	test	Query	0	prepar
2	root	localhost	test	Sleep	566	

The following table shows the contents of the `INNODB_TRX` table for a system running a heavy workload.

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weigh
77F	LOCK WAIT	2008-01-15 13:10:16	77F	2008-01-15 13:10:16	1
77E	LOCK WAIT	2008-01-15 13:10:16	77E	2008-01-15 13:10:16	1
77D	LOCK WAIT	2008-01-15 13:10:16	77D	2008-01-15 13:10:16	1
77B	LOCK WAIT	2008-01-15 13:10:16	77B:733:12:1	2008-01-15 13:10:16	4
77A	RUNNING	2008-01-15 13:10:16	NULL	NULL	4

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weight
E56	LOCK WAIT	2008-01-15 13:10:06	E56:743:6:2	2008-01-15 13:10:06	5
E55	LOCK WAIT	2008-01-15 13:10:06	E55:743:38:2	2008-01-15 13:10:13	965
19C	RUNNING	2008-01-15 13:09:10	NULL	NULL	2900
E15	RUNNING	2008-01-15 13:08:59	NULL	NULL	5395
51D	RUNNING	2008-01-15 13:08:47	NULL	NULL	9807

The following table shows the contents of the `data_lock_waits` table for a system running a heavy workload.

requesting trx id	requested lock id	blocking trx id	blocking lock id
77F	77F:806	77E	77E:806
77F	77F:806	77D	77D:806
77F	77F:806	77B	77B:806
77E	77E:806	77D	77D:806
77E	77E:806	77B	77B:806
77D	77D:806	77B	77B:806
77B	77B:733:12:1	77A	77A:733:12:1
E56	E56:743:6:2	E55	E55:743:6:2
E55	E55:743:38:2	19C	19C:743:38:2

The following table shows the contents of the `data_locks` table for a system running a heavy workload.

lock id	lock trx id	lock mode	lock type	lock schema	lock table	lock index	lock data
77F:806	77F	AUTO_INC	TABLE	test	t09	NULL	NULL
77E:806	77E	AUTO_INC	TABLE	test	t09	NULL	NULL
77D:806	77D	AUTO_INC	TABLE	test	t09	NULL	NULL
77B:806	77B	AUTO_INC	TABLE	test	t09	NULL	NULL
77B:733:12:1	77B	X	RECORD	test	t09	PRIMARY	supremum pseudo-record
77A:733:12:1	77A	X	RECORD	test	t09	PRIMARY	supremum pseudo-record
E56:743:6:2	E56	S	RECORD	test	t2	PRIMARY	0, 0
E55:743:6:2	E55	X	RECORD	test	t2	PRIMARY	0, 0
E55:743:38:2	E55	S	RECORD	test	t2	PRIMARY	1922, 1922
19C:743:38:2	19C	X	RECORD	test	t2	PRIMARY	1922, 1922

15.14.2.2 InnoDB Lock and Lock-Wait Information

Note

This section describes locking information as exposed by the Performance Schema `data_locks` and `data_lock_waits` tables, which supersede the `INFORMATION_SCHEMA` `INNODB_LOCKS` and `INNODB_LOCK_WAITS` tables in MySQL 8.0. For similar discussion written in terms of the older `INFORMATION_SCHEMA` tables, see InnoDB Lock and Lock-Wait Information in MySQL 5.7 Reference Manual.

When a transaction updates a row in a table, or locks it with `SELECT FOR UPDATE`, `InnoDB` establishes a list or queue of locks on that row. Similarly, `InnoDB` maintains a list of locks on a table for table-level locks. If a second transaction wants to update a row or lock a table already locked by a prior transaction in an incompatible mode, `InnoDB` adds a lock request for the row to the corresponding queue. For a lock to be acquired by a transaction, all incompatible lock requests previously entered into the lock queue for that row or table must be removed (which occurs when the transactions holding or requesting those locks either commit or roll back).

A transaction may have any number of lock requests for different rows or tables. At any given time, a transaction may request a lock that is held by another transaction, in which case it is blocked by that other transaction. The requesting transaction must wait for the transaction that holds the blocking lock to commit or roll back. If a transaction is not waiting for a lock, it is in a `RUNNING` state. If a transaction is waiting for a lock, it is in a `LOCK WAIT` state. (The `INFORMATION_SCHEMA` `INNODB_TRX` table indicates transaction state values.)

The Performance Schema `data_locks` table holds one or more rows for each `LOCK WAIT` transaction, indicating any lock requests that prevent its progress. This table also contains one row describing each lock in a queue of locks pending for a given row or table. The Performance Schema `data_lock_waits` table shows which locks already held by a transaction are blocking locks requested by other transactions.

15.14.2.3 Persistence and Consistency of InnoDB Transaction and Locking Information

Note

This section describes locking information as exposed by the Performance Schema `data_locks` and `data_lock_waits` tables, which supersede the `INFORMATION_SCHEMA` `INNODB_LOCKS` and `INNODB_LOCK_WAITS` tables in MySQL 8.0. For similar discussion written in terms of the older `INFORMATION_SCHEMA` tables, see Persistence and Consistency of InnoDB Transaction and Locking Information in MySQL 5.7 Reference Manual.

The data exposed by the transaction and locking tables (`INFORMATION_SCHEMA` `INNODB_TRX` table, Performance Schema `data_locks` and `data_lock_waits` tables) represents a glimpse into fast-changing data. This is not like user tables, where the data changes only when application-initiated updates occur. The underlying data is internal system-managed data, and can change very quickly:

- Data might not be consistent between the `INNODB_TRX`, `data_locks`, and `data_lock_waits` tables.

  The `data_locks` and `data_lock_waits` tables expose live data from the `InnoDB` storage engine, to provide lock inormation about the transactions in the `INNODB_TRX` table. Data retrieved from the lock tables exists when the `SELECT` is executed, but might be gone or changed by the time the query result is consumed by the client.

  Joining `data_locks` with `data_lock_waits` can show rows in `data_lock_waits` that identify a parent row in `data_locks` that no longer exists or does not exist yet.

- Data in the transaction and locking tables might not be consistent with data in the INFORMATION_SCHEMA PROCESSLIST table or Performance Schema threads table.

  For example, you should be careful when comparing data in the InnoDB transaction and locking tables with data in the PROCESSLIST table. Even if you issue a single SELECT (joining INNODB_TRX and PROCESSLIST, for example), the content of those tables is generally not consistent. It is possible for INNODB_TRX to reference rows that are not present in PROCESSLIST or for the currently executing SQL query of a transaction shown in INNODB_TRX.TRX_QUERY to differ from the one in PROCESSLIST.INFO.

15.14.3 InnoDB INFORMATION_SCHEMA Schema Object Tables

You can extract metadata about schema objects managed by InnoDB using InnoDB INFORMATION_SCHEMA tables. This information comes from the data dictionary. Traditionally, you would get this type of information using the techniques from Section 15.16, “InnoDB Monitors”, setting up InnoDB monitors and parsing the output from the SHOW ENGINE INNODB STATUS statement. The InnoDB INFORMATION_SCHEMA table interface allows you to query this data using SQL.

InnoDB INFORMATION_SCHEMA schema object tables include the tables listed below.

```
INNODB_DATAFILES
INNODB_TABLESTATS
INNODB_FOREIGN
INNODB_COLUMNS
INNODB_INDEXES
INNODB_FIELDS
INNODB_TABLESPACES
INNODB_TABLESPACES_BRIEF
INNODB_FOREIGN_COLS
INNODB_TABLES
```

The table names are indicative of the type of data provided:

- INNODB_TABLES provides metadata about InnoDB tables.
- INNODB_COLUMNS provides metadata about InnoDB table columns.
- INNODB_INDEXES provides metadata about InnoDB indexes.
- INNODB_FIELDS provides metadata about the key columns (fields) of InnoDB indexes.
- INNODB_TABLESTATS provides a view of low-level status information about InnoDB tables that is derived from in-memory data structures.
- INNODB_DATAFILES provides data file path information for InnoDB file-per-table and general tablespaces.
- INNODB_TABLESPACES provides metadata about InnoDB file-per-table, general, and undo tablespaces.
- INNODB_TABLESPACES_BRIEF provides a subset of metadata about InnoDB tablespaces.
- INNODB_FOREIGN provides metadata about foreign keys defined on InnoDB tables.
- INNODB_FOREIGN_COLS provides metadata about the columns of foreign keys that are defined on InnoDB tables.

InnoDB INFORMATION_SCHEMA schema object tables can be joined together through fields such as TABLE_ID, INDEX_ID, and SPACE, allowing you to easily retrieve all available data for an object you want to study or monitor.

Refer to the InnoDB INFORMATION_SCHEMA documentation for information about the columns of each table.

Example 15.2 InnoDB INFORMATION_SCHEMA Schema Object Tables

This example uses a simple table (t1) with a single index (i1) to demonstrate the type of metadata found in the InnoDB INFORMATION_SCHEMA schema object tables.

1. Create a test database and table t1:

```
mysql> CREATE DATABASE test;

mysql> USE test;

mysql> CREATE TABLE t1 (
       col1 INT,
       col2 CHAR(10),
       col3 VARCHAR(10))
       ENGINE = InnoDB;

mysql> CREATE INDEX i1 ON t1(col1);
```

2. After creating the table t1, query INNODB_TABLES to locate the metadata for test/t1:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE NAME='test/t1' \G
*************************** 1. row ***************************
     TABLE_ID: 71
         NAME: test/t1
         FLAG: 1
       N_COLS: 6
        SPACE: 57
   ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
 INSTANT_COLS: 0
```

   Table t1 has a TABLE_ID of 71. The FLAG field provides bit level information about table format and storage characteristics. There are six columns, three of which are hidden columns created by InnoDB (DB_ROW_ID, DB_TRX_ID, and DB_ROLL_PTR). The ID of the table's SPACE is 57 (a value of 0 would indicate that the table resides in the system tablespace). The ROW_FORMAT is Compact. ZIP_PAGE_SIZE only applies to tables with a Compressed row format. INSTANT_COLS shows number of columns in the table prior to adding the first instant column using ALTER TABLE ... ADD COLUMN with ALGORITHM=INSTANT.

3. Using the TABLE_ID information from INNODB_TABLES, query the INNODB_COLUMNS table for information about the table's columns.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_COLUMNS where TABLE_ID = 71\G
*************************** 1. row ***************************
     TABLE_ID: 71
         NAME: col1
          POS: 0
        MTYPE: 6
      PRTYPE: 1027
          LEN: 4
  HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
*************************** 2. row ***************************
     TABLE_ID: 71
         NAME: col2
          POS: 1
```

```
       MTYPE: 2
      PRTYPE: 524542
         LEN: 10
  HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
*************************** 3. row ***************************
     TABLE_ID: 71
         NAME: col3
          POS: 2
       MTYPE: 1
      PRTYPE: 524303
         LEN: 10
  HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
```

In addition to the TABLE_ID and column NAME, INNODB_COLUMNS provides the ordinal position (POS) of each column (starting from 0 and incrementing sequentially), the column MTYPE or “main type” (6 = INT, 2 = CHAR, 1 = VARCHAR), the PRTYPE or “precise type” (a binary value with bits that represent the MySQL data type, character set code, and nullability), and the column length (LEN). The HAS_DEFAULT and DEFAULT_VALUE columns only apply to columns added instantly using ALTER TABLE ... ADD COLUMN with ALGORITHM=INSTANT.

4. Using the TABLE_ID information from INNODB_TABLES once again, query INNODB_INDEXES for information about the indexes associated with table t1.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_INDEXES WHERE TABLE_ID = 71 \G
*************************** 1. row ***************************
       INDEX_ID: 111
           NAME: GEN_CLUST_INDEX
       TABLE_ID: 71
           TYPE: 1
       N_FIELDS: 0
        PAGE_NO: 3
          SPACE: 57
MERGE_THRESHOLD: 50
*************************** 2. row ***************************
       INDEX_ID: 112
           NAME: i1
       TABLE_ID: 71
           TYPE: 0
       N_FIELDS: 1
        PAGE_NO: 4
          SPACE: 57
MERGE_THRESHOLD: 50
```

INNODB_INDEXES returns data for two indexes. The first index is GEN_CLUST_INDEX, which is a clustered index created by InnoDB if the table does not have a user-defined clustered index. The second index (i1) is the user-defined secondary index.

The INDEX_ID is an identifier for the index that is unique across all databases in an instance. The TABLE_ID identifies the table that the index is associated with. The index TYPE value indicates the type of index (1 = Clustered Index, 0 = Secondary index). The N_FILEDS value is the number of fields that comprise the index. PAGE_NO is the root page number of the index B-tree, and SPACE is the ID of the tablespace where the index resides. A nonzero value indicates that the index does not reside in the system tablespace. MERGE_THRESHOLD defines a percentage threshold value for the amount of data in an index page. If the amount of data in an index page falls below the this value (the default is 50%) when a row is deleted or when a row is shortened by an update operation, InnoDB attempts to merge the index page with a neighboring index page.

5. Using the INDEX_ID information from INNODB_INDEXES, query INNODB_FIELDS for information about the fields of index i1.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FIELDS where INDEX_ID = 112 \G
*************************** 1. row ***************************
INDEX_ID: 112
    NAME: col1
     POS: 0
```

`INNODB_FIELDS` provides the `NAME` of the indexed field and its ordinal position within the index. If the index (i1) had been defined on multiple fields, `INNODB_FIELDS` would provide metadata for each of the indexed fields.

6. Using the `SPACE` information from `INNODB_TABLES`, query `INNODB_TABLESPACES` table for information about the table's tablespace.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESPACES WHERE SPACE = 57 \G
*************************** 1. row ***************************
         SPACE: 57
         NAME: test/t1
         FLAG: 16417
    ROW_FORMAT: Dynamic
     PAGE_SIZE: 16384
 ZIP_PAGE_SIZE: 0
    SPACE_TYPE: Single
 FS_BLOCK_SIZE: 4096
     FILE_SIZE: 114688
ALLOCATED_SIZE: 98304
SERVER_VERSION: 8.0.4
 SPACE_VERSION: 1
    ENCRYPTION: N
```

In addition to the `SPACE` ID of the tablespace and the `NAME` of the associated table, `INNODB_TABLESPACES` provides tablespace `FLAG` data, which is bit level information about tablespace format and storage characteristics. Also provided are tablespace `ROW_FORMAT`, `PAGE_SIZE`, and several other tablespace metadata items.

7. Using the `SPACE` information from `INNODB_TABLES` once again, query `INNODB_DATAFILES` for the location of the tablespace data file.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_DATAFILES WHERE SPACE = 57 \G
*************************** 1. row ***************************
SPACE: 57
 PATH: ./test/t1.ibd
```

The datafile is located in the `test` directory under MySQL's `data` directory. If a file-per-table tablespace were created in a location outside the MySQL data directory using the `DATA DIRECTORY` clause of the `CREATE TABLE` statement, the tablespace `PATH` would be a fully qualified directory path.

8. As a final step, insert a row into table `t1` (`TABLE_ID = 71`) and view the data in the `INNODB_TABLESTATS` table. The data in this table is used by the MySQL optimizer to calculate which index to use when querying an `InnoDB` table. This information is derived from in-memory data structures.

```
mysql> INSERT INTO t1 VALUES(5, 'abc', 'def');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESTATS where TABLE_ID = 71 \G
*************************** 1. row ***************************
        TABLE_ID: 71
            NAME: test/t1
```

```
STATS_INITIALIZED: Initialized
         NUM_ROWS: 1
 CLUST_INDEX_SIZE: 1
 OTHER_INDEX_SIZE: 0
 MODIFIED_COUNTER: 1
          AUTOINC: 0
        REF_COUNT: 1
```

The STATS_INITIALIZED field indicates whether or not statistics have been collected for the table. NUM_ROWS is the current estimated number of rows in the table. The CLUST_INDEX_SIZE and OTHER_INDEX_SIZE fields report the number of pages on disk that store clustered and secondary indexes for the table, respectively. The MODIFIED_COUNTER value shows the number of rows modified by DML operations and cascade operations from foreign keys. The AUTOINC value is the next number to be issued for any autoincrement-based operation. There are no autoincrement columns defined on table t1, so the value is 0. The REF_COUNT value is a counter. When the counter reaches 0, it signifies that the table metadata can be evicted from the table cache.

Example 15.3 Foreign Key INFORMATION_SCHEMA Schema Object Tables

The INNODB_FOREIGN and INNODB_FOREIGN_COLS tables provide data about foreign key relationships. This example uses a parent table and child table with a foreign key relationship to demonstrate the data found in the INNODB_FOREIGN and INNODB_FOREIGN_COLS tables.

1. Create the test database with parent and child tables:

```
mysql> CREATE DATABASE test;

mysql> USE test;

mysql> CREATE TABLE parent (id INT NOT NULL,
       PRIMARY KEY (id)) ENGINE=INNODB;

mysql> CREATE TABLE child (id INT, parent_id INT,
       INDEX par_ind (parent_id),
       CONSTRAINT fk1
       FOREIGN KEY (parent_id) REFERENCES parent(id)
       ON DELETE CASCADE) ENGINE=INNODB;
```

2. After the parent and child tables are created, query INNODB_FOREIGN and locate the foreign key data for the test/child and test/parent foreign key relationship:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN \G
*************************** 1. row ***************************
      ID: test/fk1
FOR_NAME: test/child
REF_NAME: test/parent
  N_COLS: 1
    TYPE: 1
```

Metadata includes the foreign key ID (fk1), which is named for the CONSTRAINT that was defined on the child table. The FOR_NAME is the name of the child table where the foreign key is defined. REF_NAME is the name of the parent table (the “referenced” table). N_COLS is the number of columns in the foreign key index. TYPE is a numerical value representing bit flags that provide additional information about the foreign key column. In this case, the TYPE value is 1, which indicates that the ON DELETE CASCADE option was specified for the foreign key. See the INNODB_FOREIGN table definition for more information about TYPE values.

3. Using the foreign key ID, query INNODB_FOREIGN_COLS to view data about the columns of the foreign key.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN_COLS WHERE ID = 'test/fk1' \G
*************************** 1. row ***************************
          ID: test/fk1
FOR_COL_NAME: parent_id
REF_COL_NAME: id
         POS: 0
```

`FOR_COL_NAME` is the name of the foreign key column in the child table, and `REF_COL_NAME` is the name of the referenced column in the parent table. The `POS` value is the ordinal position of the key field within the foreign key index, starting at zero.

Example 15.4 Joining InnoDB INFORMATION_SCHEMA Schema Object Tables

This example demonstrates joining three `InnoDB INFORMATION_SCHEMA` schema object tables (`INNODB_TABLES`, `INNODB_TABLESPACES`, and `INNODB_TABLESTATS`) to gather file format, row format, page size, and index size information about tables in the employees sample database.

The following table name aliases are used to shorten the query string:

- `INFORMATION_SCHEMA.INNODB_TABLES`: a
- `INFORMATION_SCHEMA.INNODB_TABLESPACES`: b
- `INFORMATION_SCHEMA.INNODB_TABLESTATS`: c

An `IF()` control flow function is used to account for compressed tables. If a table is compressed, the index size is calculated using `ZIP_PAGE_SIZE` rather than `PAGE_SIZE`. `CLUST_INDEX_SIZE` and `OTHER_INDEX_SIZE`, which are reported in bytes, are divided by `1024*1024` to provide index sizes in megabytes (MBs). MB values are rounded to zero decimal spaces using the `ROUND()` function.

```
mysql> SELECT a.NAME, a.ROW_FORMAT,
        @page_size :=
         IF(a.ROW_FORMAT='Compressed',
          b.ZIP_PAGE_SIZE, b.PAGE_SIZE)
          AS page_size,
         ROUND((@page_size * c.CLUST_INDEX_SIZE)
          /(1024*1024)) AS pk_mb,
         ROUND((@page_size * c.OTHER_INDEX_SIZE)
          /(1024*1024)) AS secidx_mb
       FROM INFORMATION_SCHEMA.INNODB_TABLES a
       INNER JOIN INFORMATION_SCHEMA.INNODB_TABLESPACES b on a.NAME = b.NAME
       INNER JOIN INFORMATION_SCHEMA.INNODB_TABLESTATS c on b.NAME = c.NAME
       WHERE a.NAME LIKE 'employees/%'
       ORDER BY a.NAME DESC;
+-----------------------+------------+-----------+-------+-----------+
| NAME                  | ROW_FORMAT | page_size | pk_mb | secidx_mb |
+-----------------------+------------+-----------+-------+-----------+
| employees/titles      | Dynamic    |     16384 |    20 |        11 |
| employees/salaries    | Dynamic    |     16384 |    93 |        34 |
| employees/employees   | Dynamic    |     16384 |    15 |         0 |
| employees/dept_manager | Dynamic    |     16384 |     0 |         0 |
| employees/dept_emp    | Dynamic    |     16384 |    12 |        10 |
| employees/departments | Dynamic    |     16384 |     0 |         0 |
+-----------------------+------------+-----------+-------+-----------+
```

15.14.4 InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables

The following tables provide metadata for `FULLTEXT` indexes:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB_FT%';
```

```
+----------------------------------------+
| Tables_in_INFORMATION_SCHEMA (INNODB_FT%) |
+----------------------------------------+
| INNODB_FT_CONFIG                       |
| INNODB_FT_BEING_DELETED                |
| INNODB_FT_DELETED                      |
| INNODB_FT_DEFAULT_STOPWORD             |
| INNODB_FT_INDEX_TABLE                  |
| INNODB_FT_INDEX_CACHE                  |
+----------------------------------------+
```

Table Overview

- `INNODB_FT_CONFIG`: Provides metadata about the `FULLTEXT` index and associated processing for an `InnoDB` table.
- `INNODB_FT_BEING_DELETED`: Provides a snapshot of the `INNODB_FT_DELETED` table; it is used only during an `OPTIMIZE TABLE` maintenance operation. When `OPTIMIZE TABLE` is run, the `INNODB_FT_BEING_DELETED` table is emptied, and `DOC_ID` values are removed from the `INNODB_FT_DELETED` table. Because the contents of `INNODB_FT_BEING_DELETED` typically have a short lifetime, this table has limited utility for monitoring or debugging. For information about running `OPTIMIZE TABLE` on tables with `FULLTEXT` indexes, see Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”.
- `INNODB_FT_DELETED`: Stores rows that are deleted from the `FULLTEXT` index for an `InnoDB` table. To avoid expensive index reorganization during DML operations for an `InnoDB` `FULLTEXT` index, the information about newly deleted words is stored separately, filtered out of search results when you do a text search, and removed from the main search index only when you issue an `OPTIMIZE TABLE` statement for the `InnoDB` table.
- `INNODB_FT_DEFAULT_STOPWORD`: Holds a list of stopwords that are used by default when creating a `FULLTEXT` index on `InnoDB` tables.

  For information about the `INNODB_FT_DEFAULT_STOPWORD` table, see Section 12.9.4, “Full-Text Stopwords”.
- `INNODB_FT_INDEX_TABLE`: Provides information about the inverted index used to process text searches against the `FULLTEXT` index of an `InnoDB` table.
- `INNODB_FT_INDEX_CACHE`: Provides token information about newly inserted rows in a `FULLTEXT` index. To avoid expensive index reorganization during DML operations, the information about newly indexed words is stored separately, and combined with the main search index only when `OPTIMIZE TABLE` is run, when the server is shut down, or when the cache size exceeds a limit defined by the `innodb_ft_cache_size` or `innodb_ft_total_cache_size` system variable.

Note

With the exception of the `INNODB_FT_DEFAULT_STOPWORD` table, these tables are empty initially. Before querying any of them, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example, `test/articles`).

Example 15.5 InnoDB FULLTEXT Index INFORMATION_SCHEMA Tables

This example uses a table with a `FULLTEXT` index to demonstrate the data contained in the `FULLTEXT` index `INFORMATION_SCHEMA` tables.

1. Create a table with a `FULLTEXT` index and insert some data:

```
mysql> CREATE TABLE articles (
         id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
         title VARCHAR(200),
         body TEXT,
         FULLTEXT (title,body)
       ) ENGINE=InnoDB;

mysql> INSERT INTO articles (title,body) VALUES
       ('MySQL Tutorial','DBMS stands for DataBase ...'),
       ('How To Use MySQL Well','After you went through a ...'),
       ('Optimizing MySQL','In this tutorial we will show ...'),
       ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
       ('MySQL vs. YourSQL','In the following database comparison ...'),
       ('MySQL Security','When configured properly, MySQL ...');
```

2. Set the `innodb_ft_aux_table` variable to the name of the table with the `FULLTEXT` index. If this variable is not set, the `InnoDB` `FULLTEXT` `INFORMATION_SCHEMA` tables are empty, with the exception of `INNODB_FT_DEFAULT_STOPWORD`.

```
mysql> SET GLOBAL innodb_ft_aux_table = 'test/articles';
```

3. Query the `INNODB_FT_INDEX_CACHE` table, which shows information about newly inserted rows in a `FULLTEXT` index. To avoid expensive index reorganization during DML operations, data for newly inserted rows remains in the `FULLTEXT` index cache until `OPTIMIZE TABLE` is run (or until the server is shut down or cache limits are exceeded).

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE LIMIT 5;
+------------+--------------+-------------+-----------+--------+----------+
| WORD       | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+------------+--------------+-------------+-----------+--------+----------+
| 1001       |            5 |           5 |         1 |      5 |        0 |
| after      |            3 |           3 |         1 |      3 |       22 |
| comparison |            6 |           6 |         1 |      6 |       44 |
| configured |            7 |           7 |         1 |      7 |       20 |
| database   |            2 |           6 |         2 |      2 |       31 |
+------------+--------------+-------------+-----------+--------+----------+
```

4. Enable the `innodb_optimize_fulltext_only` system variable and run `OPTIMIZE TABLE` on the table that contains the `FULLTEXT` index. This operation flushes the contents of the `FULLTEXT` index cache to the main `FULLTEXT` index. `innodb_optimize_fulltext_only` changes the way the `OPTIMIZE TABLE` statement operates on `InnoDB` tables, and is intended to be enabled temporarily, during maintenance operations on `InnoDB` tables with `FULLTEXT` indexes.

```
mysql> SET GLOBAL innodb_optimize_fulltext_only=ON;

mysql> OPTIMIZE TABLE articles;
+---------------+----------+----------+----------+
| Table         | Op       | Msg_type | Msg_text |
+---------------+----------+----------+----------+
| test.articles | optimize | status   | OK       |
+---------------+----------+----------+----------+
```

5. Query the `INNODB_FT_INDEX_TABLE` table to view information about data in the main `FULLTEXT` index, including information about the data that was just flushed from the `FULLTEXT` index cache.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE LIMIT 5;
+------------+--------------+-------------+-----------+--------+----------+
| WORD       | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
```

```
+------------+--------------+-------------+-----------+--------+----------+
| 1001       |            5 |           5 |         1 |      5 |        0 |
| after      |            3 |           3 |         1 |      3 |       22 |
| comparison |            6 |           6 |         1 |      6 |       44 |
| configured |            7 |           7 |         1 |      7 |       20 |
| database   |            2 |           6 |         2 |      2 |       31 |
+------------+--------------+-------------+-----------+--------+----------+
```

The `INNODB_FT_INDEX_CACHE` table is now empty since the `OPTIMIZE TABLE` operation flushed the `FULLTEXT` index cache.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE LIMIT 5;
Empty set (0.00 sec)
```

6. Delete some records from the `test/articles` table.

```
mysql> DELETE FROM test.articles WHERE id < 4;
```

7. Query the `INNODB_FT_DELETED` table. This table records rows that are deleted from the `FULLTEXT` index. To avoid expensive index reorganization during DML operations, information about newly deleted records is stored separately, filtered out of search results when you do a text search, and removed from the main search index when you run `OPTIMIZE TABLE`.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
+--------+
| DOC_ID |
+--------+
|      2 |
|      3 |
|      4 |
+--------+
```

8. Run `OPTIMIZE TABLE` to remove the deleted records.

```
mysql> OPTIMIZE TABLE articles;
+---------------+----------+----------+----------+
| Table         | Op       | Msg_type | Msg_text |
+---------------+----------+----------+----------+
| test.articles | optimize | status   | OK       |
+---------------+----------+----------+----------+
```

The `INNODB_FT_DELETED` table should now be empty.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
Empty set (0.00 sec)
```

9. Query the `INNODB_FT_CONFIG` table. This table contains metadata about the `FULLTEXT` index and related processing:

   - `optimize_checkpoint_limit`: The number of seconds after which an `OPTIMIZE TABLE` run stops.
   - `synced_doc_id`: The next `DOC_ID` to be issued.
   - `stopword_table_name`: The *`database/table`* name for a user-defined stopword table. The `VALUE` column is empty if there is no user-defined stopword table.

- `use_stopword`: Indicates whether a stopword table is used, which is defined when the `FULLTEXT` index is created.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_CONFIG;
+---------------------------+-------+
| KEY                       | VALUE |
+---------------------------+-------+
| optimize_checkpoint_limit | 180   |
| synced_doc_id             | 8     |
| stopword_table_name       |       |
| use_stopword              | 1     |
+---------------------------+-------+
```

10. Disable `innodb_optimize_fulltext_only`, since it is intended to be enabled only temporarily:

```
mysql> SET GLOBAL innodb_optimize_fulltext_only=OFF;
```

15.14.5 InnoDB INFORMATION_SCHEMA Buffer Pool Tables

The `InnoDB` `INFORMATION_SCHEMA` buffer pool tables provide buffer pool status information and metadata about the pages within the `InnoDB` buffer pool.

The `InnoDB` `INFORMATION_SCHEMA` buffer pool tables include those listed below:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB_BUFFER%';
+-----------------------------------------------+
| Tables_in_INFORMATION_SCHEMA (INNODB_BUFFER%) |
+-----------------------------------------------+
| INNODB_BUFFER_PAGE_LRU                        |
| INNODB_BUFFER_PAGE                            |
| INNODB_BUFFER_POOL_STATS                      |
+-----------------------------------------------+
```

Table Overview

- `INNODB_BUFFER_PAGE`: Holds information about each page in the `InnoDB` buffer pool.
- `INNODB_BUFFER_PAGE_LRU`: Holds information about the pages in the `InnoDB` buffer pool, in particular how they are ordered in the LRU list that determines which pages to evict from the buffer pool when it becomes full. The `INNODB_BUFFER_PAGE_LRU` table has the same columns as the `INNODB_BUFFER_PAGE` table, except that the `INNODB_BUFFER_PAGE_LRU` table has an `LRU_POSITION` column instead of a `BLOCK_ID` column.
- `INNODB_BUFFER_POOL_STATS`: Provides buffer pool status information. Much of the same information is provided by `SHOW ENGINE INNODB STATUS` output, or may be obtained using `InnoDB` buffer pool server status variables.

Warning

Querying the `INNODB_BUFFER_PAGE` or `INNODB_BUFFER_PAGE_LRU` table can affect performance. Do not query these tables on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

Example 15.6 Querying System Data in the INNODB_BUFFER_PAGE Table

This query provides an approximate count of pages that contain system data by excluding pages where the `TABLE_NAME` value is either `NULL` or includes a slash `/` or period `.` in the table name, which indicates a user-defined table.

```
mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NULL OR (INSTR(TABLE_NAME, '/') = 0 AND INSTR(TABLE_NAME, '.') = 0);
+----------+
| COUNT(*) |
+----------+
|     1516 |
+----------+
```

This query returns the approximate number of pages that contain system data, the total number of buffer pool pages, and an approximate percentage of pages that contain system data.

```
mysql> SELECT
       (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NULL OR (INSTR(TABLE_NAME, '/') = 0 AND INSTR(TABLE_NAME, '.') = 0)
       ) AS system_pages,
       (
       SELECT COUNT(*)
       FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       ) AS total_pages,
       (
       SELECT ROUND((system_pages/total_pages) * 100)
       ) AS system_page_percentage;
+--------------+-------------+-----------------------+
| system_pages | total_pages | system_page_percentage |
+--------------+-------------+-----------------------+
|          295 |        8192 |                     4 |
+--------------+-------------+-----------------------+
```

The type of system data in the buffer pool can be determined by querying the `PAGE_TYPE` value. For example, the following query returns eight distinct `PAGE_TYPE` values among the pages that contain system data:

```
mysql> SELECT DISTINCT PAGE_TYPE FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NULL OR (INSTR(TABLE_NAME, '/') = 0 AND INSTR(TABLE_NAME, '.') = 0);
+-------------------+
| PAGE_TYPE         |
+-------------------+
| SYSTEM            |
| IBUF_BITMAP       |
| UNKNOWN           |
| FILE_SPACE_HEADER |
| INODE             |
| UNDO_LOG          |
| ALLOCATED         |
+-------------------+
```

Example 15.7 Querying User Data in the INNODB_BUFFER_PAGE Table

This query provides an approximate count of pages containing user data by counting pages where the `TABLE_NAME` value is `NOT NULL` and `NOT LIKE '%INNODB_TABLES%'`.

```
mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NOT NULL AND TABLE_NAME NOT LIKE '%INNODB_TABLES%';
+----------+
| COUNT(*) |
```

```
+----------+
|     7897 |
+----------+
```

This query returns the approximate number of pages that contain user data, the total number of buffer pool pages, and an approximate percentage of pages that contain user data.

```
mysql> SELECT
       (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NOT NULL AND (INSTR(TABLE_NAME, '/') > 0 OR INSTR(TABLE_NAME, '.') > 0)
       ) AS user_pages,
       (
       SELECT COUNT(*)
       FROM information_schema.INNODB_BUFFER_PAGE
       ) AS total_pages,
       (
       SELECT ROUND((user_pages/total_pages) * 100)
       ) AS user_page_percentage;
+------------+-------------+----------------------+
| user_pages | total_pages | user_page_percentage |
+------------+-------------+----------------------+
|       7897 |        8192 |                   96 |
+------------+-------------+----------------------+
```

This query identifies user-defined tables with pages in the buffer pool:

```
mysql> SELECT DISTINCT TABLE_NAME FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NOT NULL AND (INSTR(TABLE_NAME, '/') > 0 OR INSTR(TABLE_NAME, '.') > 0)
       AND TABLE_NAME NOT LIKE '`mysql`.`innodb_%';
+-------------------------+
| TABLE_NAME              |
+-------------------------+
| `employees`.`salaries`  |
| `employees`.`employees` |
+-------------------------+
```

Example 15.8 Querying Index Data in the INNODB_BUFFER_PAGE Table

For information about index pages, query the `INDEX_NAME` column using the name of the index. For example, the following query returns the number of pages and total data size of pages for the `emp_no` index that is defined on the `employees.salaries` table:

```
mysql> SELECT INDEX_NAME, COUNT(*) AS Pages,
ROUND(SUM(IF(COMPRESSED_SIZE = 0, @@GLOBAL.innodb_page_size, COMPRESSED_SIZE))/1024/1024)
AS 'Total Data (MB)'
FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
WHERE INDEX_NAME='emp_no' AND TABLE_NAME = '`employees`.`salaries`';
+------------+-------+-----------------+
| INDEX_NAME | Pages | Total Data (MB) |
+------------+-------+-----------------+
| emp_no     |  1609 |              25 |
+------------+-------+-----------------+
```

This query returns the number of pages and total data size of pages for all indexes defined on the `employees.salaries` table:

```
mysql> SELECT INDEX_NAME, COUNT(*) AS Pages,
       ROUND(SUM(IF(COMPRESSED_SIZE = 0, @@GLOBAL.innodb_page_size, COMPRESSED_SIZE))/1024/1024)
       AS 'Total Data (MB)'
       FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME = '`employees`.`salaries`'
       GROUP BY INDEX_NAME;
```

```
+------------+-------+-----------------+
| INDEX_NAME | Pages | Total Data (MB) |
+------------+-------+-----------------+
| emp_no     |  1608 |              25 |
| PRIMARY    |  6086 |              95 |
+------------+-------+-----------------+
```

Example 15.9 Querying LRU_POSITION Data in the INNODB_BUFFER_PAGE_LRU Table

The `INNODB_BUFFER_PAGE_LRU` table holds information about the pages in the `InnoDB` buffer pool, in particular how they are ordered that determines which pages to evict from the buffer pool when it becomes full. The definition for this page is the same as for `INNODB_BUFFER_PAGE`, except this table has an `LRU_POSITION` column instead of a `BLOCK_ID` column.

This query counts the number of positions at a specific location in the LRU list occupied by pages of the `employees.employees` table.

```
mysql> SELECT COUNT(LRU_POSITION) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU
       WHERE TABLE_NAME='`employees`.`employees`' AND LRU_POSITION < 3072;
+---------------------+
| COUNT(LRU_POSITION) |
+---------------------+
|                 548 |
+---------------------+
```

Example 15.10 Querying the INNODB_BUFFER_POOL_STATS Table

The `INNODB_BUFFER_POOL_STATS` table provides information similar to `SHOW ENGINE INNODB STATUS` and `InnoDB` buffer pool status variables.

```
mysql> SELECT * FROM information_schema.INNODB_BUFFER_POOL_STATS \G
*************************** 1. row ***************************
                         POOL_ID: 0
                       POOL_SIZE: 8192
                    FREE_BUFFERS: 1
                  DATABASE_PAGES: 8173
              OLD_DATABASE_PAGES: 3014
         MODIFIED_DATABASE_PAGES: 0
              PENDING_DECOMPRESS: 0
                   PENDING_READS: 0
               PENDING_FLUSH_LRU: 0
              PENDING_FLUSH_LIST: 0
                PAGES_MADE_YOUNG: 15907
            PAGES_NOT_MADE_YOUNG: 3803101
           PAGES_MADE_YOUNG_RATE: 0
       PAGES_MADE_NOT_YOUNG_RATE: 0
              NUMBER_PAGES_READ: 3270
           NUMBER_PAGES_CREATED: 13176
           NUMBER_PAGES_WRITTEN: 15109
                PAGES_READ_RATE: 0
              PAGES_CREATE_RATE: 0
             PAGES_WRITTEN_RATE: 0
               NUMBER_PAGES_GET: 33069332
                       HIT_RATE: 0
    YOUNG_MAKE_PER_THOUSAND_GETS: 0
NOT_YOUNG_MAKE_PER_THOUSAND_GETS: 0
         NUMBER_PAGES_READ_AHEAD: 2713
       NUMBER_READ_AHEAD_EVICTED: 0
                 READ_AHEAD_RATE: 0
         READ_AHEAD_EVICTED_RATE: 0
                    LRU_IO_TOTAL: 0
                  LRU_IO_CURRENT: 0
                UNCOMPRESS_TOTAL: 0
              UNCOMPRESS_CURRENT: 0
```

For comparison, `SHOW ENGINE INNODB STATUS` output and `InnoDB` buffer pool status variable output is shown below, based on the same data set.

For more information about `SHOW ENGINE INNODB STATUS` output, see Section 15.16.3, “InnoDB Standard Monitor and Lock Monitor Output”.

```
mysql> SHOW ENGINE INNODB STATUS \G
...
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 137428992
Dictionary memory allocated 579084
Buffer pool size   8192
Free buffers       1
Database pages     8173
Old database pages 3014
Modified db pages  0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 15907, not young 3803101
0.00 youngs/s, 0.00 non-youngs/s
Pages read 3270, created 13176, written 15109
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 8173, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
...
```

For status variable descriptions, see Section 5.1.10, “Server Status Variables”.

```
mysql> SHOW STATUS LIKE 'Innodb_buffer%';
+---------------------------------------+-------------+
| Variable_name                         | Value       |
+---------------------------------------+-------------+
| Innodb_buffer_pool_dump_status        | not started |
| Innodb_buffer_pool_load_status        | not started |
| Innodb_buffer_pool_resize_status      | not started |
| Innodb_buffer_pool_pages_data         | 8173        |
| Innodb_buffer_pool_bytes_data         | 133906432   |
| Innodb_buffer_pool_pages_dirty        | 0           |
| Innodb_buffer_pool_bytes_dirty        | 0           |
| Innodb_buffer_pool_pages_flushed      | 15109       |
| Innodb_buffer_pool_pages_free         | 1           |
| Innodb_buffer_pool_pages_misc         | 18          |
| Innodb_buffer_pool_pages_total        | 8192        |
| Innodb_buffer_pool_read_ahead_rnd     | 0           |
| Innodb_buffer_pool_read_ahead         | 2713        |
| Innodb_buffer_pool_read_ahead_evicted | 0           |
| Innodb_buffer_pool_read_requests      | 33069332    |
| Innodb_buffer_pool_reads              | 558         |
| Innodb_buffer_pool_wait_free          | 0           |
| Innodb_buffer_pool_write_requests     | 11985961    |
+---------------------------------------+-------------+
```

15.14.6 InnoDB INFORMATION_SCHEMA Metrics Table

The `INNODB_METRICS` table provides information about `InnoDB` performance and resource-related counters.

`INNODB_METRICS` table columns are shown below. For column descriptions, see Section 25.39.22, “The INFORMATION_SCHEMA INNODB_METRICS Table”.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts" \G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 46273
      MAX_COUNT: 46273
      MIN_COUNT: NULL
      AVG_COUNT: 492.2659574468085
    COUNT_RESET: 46273
MAX_COUNT_RESET: 46273
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: 2014-11-28 16:07:53
  TIME_DISABLED: NULL
   TIME_ELAPSED: 94
     TIME_RESET: NULL
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

Enabling, Disabling, and Resetting Counters

You can enable, disable, and reset counters using the following variables:

- `innodb_monitor_enable`: Enables counters.

  ```
  SET GLOBAL innodb_monitor_enable = [counter-name|module_name|pattern|all];
  ```

- `innodb_monitor_disable`: Disables counters.

  ```
  SET GLOBAL innodb_monitor_disable = [counter-name|module_name|pattern|all];
  ```

- `innodb_monitor_reset`: Resets counter values to zero.

  ```
  SET GLOBAL innodb_monitor_reset = [counter-name|module_name|pattern|all];
  ```

- `innodb_monitor_reset_all`: Resets all counter values. A counter must be disabled before using `innodb_monitor_reset_all`.

  ```
  SET GLOBAL innodb_monitor_reset_all = [counter-name|module_name|pattern|all];
  ```

Counters and counter modules can also be enabled at startup using the MySQL server configuration file. For example, to enable the `log` module, `metadata_table_handles_opened` and `metadata_table_handles_closed` counters, enter the following line in the `[mysqld]` section of the MySQL server configuration file.

```
[mysqld]
innodb_monitor_enable = module_recovery,metadata_table_handles_opened,metadata_table_handles_closed
```

When enabling multiple counters or modules in a configuration file, specify the `innodb_monitor_enable` variable followed by counter and module names separated by a comma, as shown above. Only the `innodb_monitor_enable` variable can be used in a configuration file. The `innodb_monitor_disable` and `innodb_monitor_reset` variables are supported on the command line only.

Note

Because each counter adds a degree of runtime overhead, use counters conservatively on production servers to diagnose specific issues or monitor specific

functionality. A test or development server is recommended for more extensive use of counters.

Counters

The list of available counters is subject to change. Query the `INFORMATION_SCHEMA.INNODB_METRICS` table for counters available in your MySQL server version.

The counters enabled by default correspond to those shown in `SHOW ENGINE INNODB STATUS` output. Counters shown in `SHOW ENGINE INNODB STATUS` output are always enabled at a system level but can be disable for the `INNODB_METRICS` table. Counter status is not persistent. Unless configured otherwise, counters revert to their default enabled or disabled status when the server is restarted.

If you run programs that would be affected by the addition or removal of counters, it is recommended that you review the releases notes and query the `INNODB_METRICS` table to identify those changes as part of your upgrade process.

```
mysql> SELECT name, subsystem, status FROM INFORMATION_SCHEMA.INNODB_METRICS ORDER BY NAME;
+----------------------------------------+--------------------+----------+
| name                                   | subsystem          | status   |
+----------------------------------------+--------------------+----------+
| adaptive_hash_pages_added              | adaptive_hash_index | disabled |
| adaptive_hash_pages_removed            | adaptive_hash_index | disabled |
| adaptive_hash_rows_added               | adaptive_hash_index | disabled |
| adaptive_hash_rows_deleted_no_hash_entry | adaptive_hash_index | disabled |
| adaptive_hash_rows_removed             | adaptive_hash_index | disabled |
| adaptive_hash_rows_updated             | adaptive_hash_index | disabled |
| adaptive_hash_searches                 | adaptive_hash_index | enabled  |
| adaptive_hash_searches_btree           | adaptive_hash_index | enabled  |
| buffer_data_reads                      | buffer             | enabled  |
| buffer_data_written                    | buffer             | enabled  |
| buffer_flush_adaptive                  | buffer             | disabled |
| buffer_flush_adaptive_avg_pass         | buffer             | disabled |
| buffer_flush_adaptive_avg_time_est     | buffer             | disabled |
| buffer_flush_adaptive_avg_time_slot    | buffer             | disabled |
| buffer_flush_adaptive_avg_time_thread  | buffer             | disabled |
| buffer_flush_adaptive_pages            | buffer             | disabled |
| buffer_flush_adaptive_total_pages      | buffer             | disabled |
| buffer_flush_avg_page_rate             | buffer             | disabled |
| buffer_flush_avg_pass                  | buffer             | disabled |
| buffer_flush_avg_time                  | buffer             | disabled |
| buffer_flush_background                | buffer             | disabled |
| buffer_flush_background_pages          | buffer             | disabled |
| buffer_flush_background_total_pages    | buffer             | disabled |
| buffer_flush_batches                   | buffer             | disabled |
| buffer_flush_batch_num_scan            | buffer             | disabled |
| buffer_flush_batch_pages               | buffer             | disabled |
| buffer_flush_batch_scanned             | buffer             | disabled |
| buffer_flush_batch_scanned_per_call    | buffer             | disabled |
| buffer_flush_batch_total_pages         | buffer             | disabled |
| buffer_flush_lsn_avg_rate              | buffer             | disabled |
| buffer_flush_neighbor                  | buffer             | disabled |
| buffer_flush_neighbor_pages            | buffer             | disabled |
| buffer_flush_neighbor_total_pages      | buffer             | disabled |
| buffer_flush_n_to_flush_by_age         | buffer             | disabled |
| buffer_flush_n_to_flush_requested      | buffer             | disabled |
| buffer_flush_pct_for_dirty             | buffer             | disabled |
| buffer_flush_pct_for_lsn               | buffer             | disabled |
| buffer_flush_sync                      | buffer             | disabled |
| buffer_flush_sync_pages                | buffer             | disabled |
| buffer_flush_sync_total_pages          | buffer             | disabled |
| buffer_flush_sync_waits                | buffer             | disabled |
| buffer_LRU_batches_evict               | buffer             | disabled |
| buffer_LRU_batches_flush               | buffer             | disabled |
```

```
| buffer_LRU_batch_evict_pages             | buffer             | disabled |
| buffer_LRU_batch_evict_total_pages       | buffer             | disabled |
| buffer_LRU_batch_flush_avg_pass          | buffer             | disabled |
| buffer_LRU_batch_flush_avg_time_est      | buffer             | disabled |
| buffer_LRU_batch_flush_avg_time_slot     | buffer             | disabled |
| buffer_LRU_batch_flush_avg_time_thread   | buffer             | disabled |
| buffer_LRU_batch_flush_pages             | buffer             | disabled |
| buffer_LRU_batch_flush_total_pages       | buffer             | disabled |
| buffer_LRU_batch_num_scan                | buffer             | disabled |
| buffer_LRU_batch_scanned                 | buffer             | disabled |
| buffer_LRU_batch_scanned_per_call        | buffer             | disabled |
| buffer_LRU_get_free_loops                | buffer             | disabled |
| buffer_LRU_get_free_search               | Buffer             | disabled |
| buffer_LRU_get_free_waits                | buffer             | disabled |
| buffer_LRU_search_num_scan               | buffer             | disabled |
| buffer_LRU_search_scanned                | buffer             | disabled |
| buffer_LRU_search_scanned_per_call       | buffer             | disabled |
| buffer_LRU_single_flush_failure_count    | Buffer             | disabled |
| buffer_LRU_single_flush_num_scan         | buffer             | disabled |
| buffer_LRU_single_flush_scanned          | buffer             | disabled |
| buffer_LRU_single_flush_scanned_per_call | buffer             | disabled |
| buffer_LRU_unzip_search_num_scan         | buffer             | disabled |
| buffer_LRU_unzip_search_scanned          | buffer             | disabled |
| buffer_LRU_unzip_search_scanned_per_call | buffer             | disabled |
| buffer_pages_created                     | buffer             | enabled  |
| buffer_pages_read                        | buffer             | enabled  |
| buffer_pages_written                     | buffer             | enabled  |
| buffer_page_read_blob                    | buffer_page_io     | disabled |
| buffer_page_read_fsp_hdr                 | buffer_page_io     | disabled |
| buffer_page_read_ibuf_bitmap             | buffer_page_io     | disabled |
| buffer_page_read_ibuf_free_list          | buffer_page_io     | disabled |
| buffer_page_read_index_ibuf_leaf         | buffer_page_io     | disabled |
| buffer_page_read_index_ibuf_non_leaf     | buffer_page_io     | disabled |
| buffer_page_read_index_inode             | buffer_page_io     | disabled |
| buffer_page_read_index_leaf              | buffer_page_io     | disabled |
| buffer_page_read_index_non_leaf          | buffer_page_io     | disabled |
| buffer_page_read_other                   | buffer_page_io     | disabled |
| buffer_page_read_system_page             | buffer_page_io     | disabled |
| buffer_page_read_trx_system              | buffer_page_io     | disabled |
| buffer_page_read_undo_log                | buffer_page_io     | disabled |
| buffer_page_read_xdes                    | buffer_page_io     | disabled |
| buffer_page_read_zblob                   | buffer_page_io     | disabled |
| buffer_page_read_zblob2                  | buffer_page_io     | disabled |
| buffer_page_written_blob                 | buffer_page_io     | disabled |
| buffer_page_written_fsp_hdr              | buffer_page_io     | disabled |
| buffer_page_written_ibuf_bitmap          | buffer_page_io     | disabled |
| buffer_page_written_ibuf_free_list       | buffer_page_io     | disabled |
| buffer_page_written_index_ibuf_leaf      | buffer_page_io     | disabled |
| buffer_page_written_index_ibuf_non_leaf  | buffer_page_io     | disabled |
| buffer_page_written_index_inode          | buffer_page_io     | disabled |
| buffer_page_written_index_leaf           | buffer_page_io     | disabled |
| buffer_page_written_index_non_leaf       | buffer_page_io     | disabled |
| buffer_page_written_other                | buffer_page_io     | disabled |
| buffer_page_written_system_page          | buffer_page_io     | disabled |
| buffer_page_written_trx_system           | buffer_page_io     | disabled |
| buffer_page_written_undo_log             | buffer_page_io     | disabled |
| buffer_page_written_xdes                 | buffer_page_io     | disabled |
| buffer_page_written_zblob                | buffer_page_io     | disabled |
| buffer_page_written_zblob2               | buffer_page_io     | disabled |
| buffer_pool_bytes_data                   | buffer             | enabled  |
| buffer_pool_bytes_dirty                  | buffer             | enabled  |
| buffer_pool_pages_data                   | buffer             | enabled  |
| buffer_pool_pages_dirty                  | buffer             | enabled  |
| buffer_pool_pages_free                   | buffer             | enabled  |
| buffer_pool_pages_misc                   | buffer             | enabled  |
| buffer_pool_pages_total                  | buffer             | enabled  |
| buffer_pool_reads                        | buffer             | enabled  |
```

```
| buffer_pool_read_ahead                | buffer             | enabled  |
| buffer_pool_read_ahead_evicted        | buffer             | enabled  |
| buffer_pool_read_requests             | buffer             | enabled  |
| buffer_pool_size                      | server             | enabled  |
| buffer_pool_wait_free                 | buffer             | enabled  |
| buffer_pool_write_requests            | buffer             | enabled  |
| compression_pad_decrements            | compression        | disabled |
| compression_pad_increments            | compression        | disabled |
| compress_pages_compressed             | compression        | disabled |
| compress_pages_decompressed           | compression        | disabled |
| ddl_background_drop_indexes           | ddl                | disabled |
| ddl_background_drop_tables            | ddl                | disabled |
| ddl_log_file_alter_table              | ddl                | disabled |
| ddl_online_create_index               | ddl                | disabled |
| ddl_pending_alter_table               | ddl                | disabled |
| ddl_sort_file_alter_table             | ddl                | disabled |
| dml_deletes                           | dml                | enabled  |
| dml_inserts                           | dml                | enabled  |
| dml_reads                             | dml                | disabled |
| dml_updates                           | dml                | enabled  |
| file_num_open_files                   | file_system        | enabled  |
| ibuf_merges                           | change_buffer      | enabled  |
| ibuf_merges_delete                    | change_buffer      | enabled  |
| ibuf_merges_delete_mark               | change_buffer      | enabled  |
| ibuf_merges_discard_delete            | change_buffer      | enabled  |
| ibuf_merges_discard_delete_mark       | change_buffer      | enabled  |
| ibuf_merges_discard_insert            | change_buffer      | enabled  |
| ibuf_merges_insert                    | change_buffer      | enabled  |
| ibuf_size                             | change_buffer      | enabled  |
| icp_attempts                          | icp                | disabled |
| icp_match                             | icp                | disabled |
| icp_no_match                          | icp                | disabled |
| icp_out_of_range                      | icp                | disabled |
| index_page_discards                   | index              | disabled |
| index_page_merge_attempts             | index              | disabled |
| index_page_merge_successful           | index              | disabled |
| index_page_reorg_attempts             | index              | disabled |
| index_page_reorg_successful           | index              | disabled |
| index_page_splits                     | index              | disabled |
| innodb_activity_count                 | server             | enabled  |
| innodb_background_drop_table_usec     | server             | disabled |
| innodb_checkpoint_usec                | server             | disabled |
| innodb_dblwr_pages_written            | server             | enabled  |
| innodb_dblwr_writes                   | server             | enabled  |
| innodb_dict_lru_count                 | server             | disabled |
| innodb_dict_lru_usec                  | server             | disabled |
| innodb_ibuf_merge_usec                | server             | disabled |
| innodb_log_flush_usec                 | server             | disabled |
| innodb_master_active_loops            | server             | disabled |
| innodb_master_idle_loops              | server             | disabled |
| innodb_master_purge_usec              | server             | disabled |
| innodb_master_thread_sleeps           | server             | disabled |
| innodb_mem_validate_usec              | server             | disabled |
| innodb_page_size                      | server             | enabled  |
| innodb_rwlock_sx_os_waits             | server             | enabled  |
| innodb_rwlock_sx_spin_rounds          | server             | enabled  |
| innodb_rwlock_sx_spin_waits           | server             | enabled  |
| innodb_rwlock_s_os_waits              | server             | enabled  |
| innodb_rwlock_s_spin_rounds           | server             | enabled  |
| innodb_rwlock_s_spin_waits            | server             | enabled  |
| innodb_rwlock_x_os_waits              | server             | enabled  |
| innodb_rwlock_x_spin_rounds           | server             | enabled  |
| innodb_rwlock_x_spin_waits            | server             | enabled  |
| lock_deadlocks                        | lock               | enabled  |
| lock_rec_locks                        | lock               | disabled |
| lock_rec_lock_created                 | lock               | disabled |
| lock_rec_lock_removed                 | lock               | disabled |
```

```
| lock_rec_lock_requests                    | lock                | disabled |
| lock_rec_lock_waits                       | lock                | disabled |
| lock_row_lock_current_waits               | lock                | enabled  |
| lock_row_lock_time                        | lock                | enabled  |
| lock_row_lock_time_avg                    | lock                | enabled  |
| lock_row_lock_time_max                    | lock                | enabled  |
| lock_row_lock_waits                       | lock                | enabled  |
| lock_table_locks                          | lock                | disabled |
| lock_table_lock_created                   | lock                | disabled |
| lock_table_lock_removed                   | lock                | disabled |
| lock_table_lock_waits                     | lock                | disabled |
| lock_timeouts                             | lock                | enabled  |
| log_checkpoints                           | recovery            | disabled |
| log_lsn_buf_pool_oldest                   | recovery            | disabled |
| log_lsn_checkpoint_age                    | recovery            | disabled |
| log_lsn_current                           | recovery            | disabled |
| log_lsn_last_checkpoint                   | recovery            | disabled |
| log_lsn_last_flush                        | recovery            | disabled |
| log_max_modified_age_async                | recovery            | disabled |
| log_max_modified_age_sync                 | recovery            | disabled |
| log_num_log_io                            | recovery            | disabled |
| log_padded                                | recovery            | enabled  |
| log_pending_checkpoint_writes             | recovery            | disabled |
| log_pending_log_flushes                   | recovery            | disabled |
| log_waits                                 | recovery            | enabled  |
| log_writes                                | recovery            | enabled  |
| log_write_requests                        | recovery            | enabled  |
| metadata_table_handles_closed             | metadata            | disabled |
| metadata_table_handles_opened             | metadata            | disabled |
| metadata_table_reference_count            | metadata            | disabled |
| os_data_fsyncs                            | os                  | enabled  |
| os_data_reads                             | os                  | enabled  |
| os_data_writes                            | os                  | enabled  |
| os_log_bytes_written                      | os                  | enabled  |
| os_log_fsyncs                             | os                  | enabled  |
| os_log_pending_fsyncs                     | os                  | enabled  |
| os_log_pending_writes                     | os                  | enabled  |
| os_pending_reads                          | os                  | disabled |
| os_pending_writes                         | os                  | disabled |
| purge_del_mark_records                    | purge               | disabled |
| purge_dml_delay_usec                      | purge               | disabled |
| purge_invoked                             | purge               | disabled |
| purge_resume_count                        | purge               | disabled |
| purge_stop_count                          | purge               | disabled |
| purge_undo_log_pages                      | purge               | disabled |
| purge_upd_exist_or_extern_records         | purge               | disabled |
| trx_active_transactions                   | transaction         | disabled |
| trx_commits_insert_update                 | transaction         | disabled |
| trx_nl_ro_commits                         | transaction         | disabled |
| trx_rollbacks                             | transaction         | disabled |
| trx_rollbacks_savepoint                   | transaction         | disabled |
| trx_rollback_active                       | transaction         | disabled |
| trx_ro_commits                            | transaction         | disabled |
| trx_rseg_current_size                     | transaction         | disabled |
| trx_rseg_history_len                      | transaction         | enabled  |
| trx_rw_commits                            | transaction         | disabled |
| trx_undo_slots_cached                     | transaction         | disabled |
| trx_undo_slots_used                       | transaction         | disabled |
+-------------------------------------------+---------------------+----------+
235 rows in set (0.01 sec)
```

Counter Modules

Each counter is associated with a particular module. Module names can be used to enable, disable, or reset all counters for a particular subsystem. For example, use `module_dml` to enable all counters associated with the `dml` subsystem.

```
mysql> SET GLOBAL innodb_monitor_enable = module_dml;

mysql> SELECT name, subsystem, status FROM INFORMATION_SCHEMA.INNODB_METRICS
       WHERE subsystem ='dml';
+-------------+-----------+---------+
| name        | subsystem | status  |
+-------------+-----------+---------+
| dml_reads   | dml       | enabled |
| dml_inserts | dml       | enabled |
| dml_deletes | dml       | enabled |
| dml_updates | dml       | enabled |
+-------------+-----------+---------+
```

Module names can be used with `innodb_monitor_enable` and related variables.

Module names and corresponding `SUBSYSTEM` names are listed below.

- `module_adaptive_hash` (subsystem = `adaptive_hash_index`)
- `module_buffer` (subsystem = `buffer`)
- `module_buffer_page` (subsystem = `buffer_page_io`)
- `module_compress` (subsystem = `compression`)
- `module_ddl` (subsystem = `ddl`)
- `module_dml` (subsystem = `dml`)
- `module_file` (subsystem = `file_system`)
- `module_ibuf_system` (subsystem = `change_buffer`)
- `module_icp` (subsystem = `icp`)
- `module_index` (subsystem = `index`)
- `module_innodb` (subsystem = `innodb`)
- `module_lock` (subsystem = `lock`)
- `module_log` (subsystem = `recovery`)
- `module_metadata` (subsystem = `metadata`)
- `module_os` (subsystem = `os`)
- `module_purge` (subsystem = `purge`)
- `module_trx` (subsystem = `transaction`)
- `module_undo` (subsystem = `undo`)

Example 15.11 Working with INNODB_METRICS Table Counters

This example demonstrates enabling, disabling, and resetting a counter, and querying counter data in the `INNODB_METRICS` table.

1. Create a simple `InnoDB` table:

```
mysql> USE test;
Database changed
```

```
mysql> CREATE TABLE t1 (c1 INT) ENGINE=INNODB;
Query OK, 0 rows affected (0.02 sec)
```

2. Enable the `dml_inserts` counter.

```
mysql> SET GLOBAL innodb_monitor_enable = dml_inserts;
Query OK, 0 rows affected (0.01 sec)
```

A description of the `dml_inserts` counter can be found in the `COMMENT` column of the `INNODB_METRICS` table:

```
mysql> SELECT NAME, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts";
+-------------+-------------------------+
| NAME        | COMMENT                 |
+-------------+-------------------------+
| dml_inserts | Number of rows inserted |
+-------------+-------------------------+
```

3. Query the `INNODB_METRICS` table for the `dml_inserts` counter data. Because no DML operations have been performed, the counter values are zero or NULL. The `TIME_ENABLED` and `TIME_ELAPSED` values indicate when the counter was last enabled and how many seconds have elapsed since that time.

```
mysql>  SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts" \G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 0
      MAX_COUNT: 0
      MIN_COUNT: NULL
      AVG_COUNT: 0
    COUNT_RESET: 0
MAX_COUNT_RESET: 0
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: NULL
   TIME_ELAPSED: 28
     TIME_RESET: NULL
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

4. Insert three rows of data into the table.

```
mysql> INSERT INTO t1 values(1);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t1 values(2);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t1 values(3);
Query OK, 1 row affected (0.00 sec)
```

5. Query the `INNODB_METRICS` table again for the `dml_inserts` counter data. A number of counter values have now incremented including `COUNT`, `MAX_COUNT`, `AVG_COUNT`, and `COUNT_RESET`. Refer to the `INNODB_METRICS` table definition for descriptions of these values.

```
mysql>  SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"\G
```

```
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.046153846153846156
    COUNT_RESET: 3
MAX_COUNT_RESET: 3
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: NULL
   TIME_ELAPSED: 65
     TIME_RESET: NULL
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

6. Reset the `dml_inserts` counter and query the `INNODB_METRICS` table again for the `dml_inserts` counter data. The `%_RESET` values that were reported previously, such as `COUNT_RESET` and `MAX_RESET`, are set back to zero. Values such as `COUNT`, `MAX_COUNT`, and `AVG_COUNT`, which cumulatively collect data from the time the counter is enabled, are unaffected by the reset.

```
mysql> SET GLOBAL innodb_monitor_reset = dml_inserts;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"\G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.03529411764705882
    COUNT_RESET: 0
MAX_COUNT_RESET: 0
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: 0
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: NULL
   TIME_ELAPSED: 85
     TIME_RESET: 2014-12-04 14:19:44
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

7. To reset all counter values, you must first disable the counter. Disabling the counter sets the `STATUS` value to `disabled`.

```
mysql> SET GLOBAL innodb_monitor_disable = dml_inserts;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"\G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.030612244897959183
    COUNT_RESET: 0
MAX_COUNT_RESET: 0
MIN_COUNT_RESET: NULL
```

```
AVG_COUNT_RESET: 0
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: 2014-12-04 14:20:06
   TIME_ELAPSED: 98
     TIME_RESET: NULL
         STATUS: disabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

Note

Wildcard match is supported for counter and module names. For example, instead of specifying the full `dml_inserts` counter name, you can specify `dml_i%`. You can also enable, disable, or reset multiple counters or modules at once using a wildcard match. For example, specify `dml_%` to enable, disable, or reset all counters that begin with `dml_`.

8. After the counter is disabled, you can reset all counter values using the `innodb_monitor_reset_all` option. All values are set to zero or NULL.

```
mysql> SET GLOBAL innodb_monitor_reset_all = dml_inserts;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"\G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 0
      MAX_COUNT: NULL
      MIN_COUNT: NULL
      AVG_COUNT: NULL
    COUNT_RESET: 0
MAX_COUNT_RESET: NULL
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: NULL
  TIME_DISABLED: NULL
   TIME_ELAPSED: NULL
     TIME_RESET: NULL
         STATUS: disabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

15.14.7 InnoDB INFORMATION_SCHEMA Temporary Table Info Table

`INNODB_TEMP_TABLE_INFO` provides information about user-created `InnoDB` temporary tables that are active in the `InnoDB` instance. It does not provide information about internal `InnoDB` temporary tables used by the optimizer.

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB_TEMP%';
+-------------------------------------------+
| Tables_in_INFORMATION_SCHEMA (INNODB_TEMP%) |
+-------------------------------------------+
| INNODB_TEMP_TABLE_INFO                    |
+-------------------------------------------+
```

For the table definition, see Section 25.39.28, “The INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO Table”.

Example 15.12 INNODB_TEMP_TABLE_INFO

This example demonstrates characteristics of the `INNODB_TEMP_TABLE_INFO` table.

1. Create a simple InnoDB temporary table:

```
mysql> CREATE TEMPORARY TABLE t1 (c1 INT PRIMARY KEY) ENGINE=INNODB;
```

2. Query INNODB_TEMP_TABLE_INFO to view the temporary table metadata.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO\G
*************************** 1. row ***************************
            TABLE_ID: 194
                NAME: #sql7a79_1_0
              N_COLS: 4
               SPACE: 182
```

The TABLE_ID is a unique identifier for the temporary table. The NAME column displays the system-generated name for the temporary table, which is prefixed with “#sql”. The number of columns (N_COLS) is 4 rather than 1 because InnoDB always creates three hidden table columns (DB_ROW_ID, DB_TRX_ID, and DB_ROLL_PTR).

3. Restart MySQL and query INNODB_TEMP_TABLE_INFO.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO\G
```

An empty set is returned because INNODB_TEMP_TABLE_INFO and its data are not persisted to disk when the server is shut down.

4. Create a new temporary table.

```
mysql> CREATE TEMPORARY TABLE t1 (c1 INT PRIMARY KEY) ENGINE=INNODB;
```

5. Query INNODB_TEMP_TABLE_INFO to view the temporary table metadata.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO\G
*************************** 1. row ***************************
            TABLE_ID: 196
                NAME: #sql7b0e_1_0
              N_COLS: 4
               SPACE: 184
```

The SPACE ID may be different because it is dynamically generated when the server is started.

15.14.8 Retrieving InnoDB Tablespace Metadata from INFORMATION_SCHEMA.FILES

The INFORMATION_SCHEMA.FILES table provides metadata about all InnoDB tablespace types including file-per-table tablespaces, general tablespaces, the system tablespace, temporary table tablespaces, and undo tablespaces (if present).

This section provides InnoDB-specific usage examples. For more information about data provided by the INFORMATION_SCHEMA.FILES table, see Section 25.11, “The INFORMATION_SCHEMA FILES Table”.

Note

The INNODB_TABLESPACES and INNODB_DATAFILES tables also provide metadata about InnoDB tablespaces, but data is limited to file-per-table, general, and undo tablespaces.

This query retrieves metadata about the InnoDB system tablespace from fields of the INFORMATION_SCHEMA.FILES table that are pertinent to InnoDB tablespaces. INFORMATION_SCHEMA.FILES fields that are not relevant to InnoDB always return NULL, and are excluded from the query.

```
mysql> SELECT FILE_ID, FILE_NAME, FILE_TYPE, TABLESPACE_NAME, FREE_EXTENTS,
       TOTAL_EXTENTS,  EXTENT_SIZE, INITIAL_SIZE, MAXIMUM_SIZE, AUTOEXTEND_SIZE, DATA_FREE, STATUS ENGINE
       FROM INFORMATION_SCHEMA.FILES WHERE TABLESPACE_NAME LIKE 'innodb_system' \G
*************************** 1. row ***************************
       FILE_ID: 0
     FILE_NAME: ./ibdata1
     FILE_TYPE: TABLESPACE
TABLESPACE_NAME: innodb_system
   FREE_EXTENTS: 0
  TOTAL_EXTENTS: 12
    EXTENT_SIZE: 1048576
   INITIAL_SIZE: 12582912
   MAXIMUM_SIZE: NULL
AUTOEXTEND_SIZE: 67108864
      DATA_FREE: 4194304
         ENGINE: NORMAL
```

This query retrieves the FILE_ID (equivalent to the space ID) and the FILE_NAME (which includes path information) for InnoDB file-per-table and general tablespaces. File-per-table and general tablespaces have a .ibd file extension.

```
mysql> SELECT FILE_ID, FILE_NAME FROM INFORMATION_SCHEMA.FILES
       WHERE FILE_NAME LIKE '%.ibd%' ORDER BY FILE_ID;
    +---------+---------------------------------------+
    | FILE_ID | FILE_NAME                             |
    +---------+---------------------------------------+
    |       2 | ./mysql/plugin.ibd                    |
    |       3 | ./mysql/servers.ibd                   |
    |       4 | ./mysql/help_topic.ibd                |
    |       5 | ./mysql/help_category.ibd             |
    |       6 | ./mysql/help_relation.ibd             |
    |       7 | ./mysql/help_keyword.ibd              |
    |       8 | ./mysql/time_zone_name.ibd            |
    |       9 | ./mysql/time_zone.ibd                 |
    |      10 | ./mysql/time_zone_transition.ibd      |
    |      11 | ./mysql/time_zone_transition_type.ibd |
    |      12 | ./mysql/time_zone_leap_second.ibd     |
    |      13 | ./mysql/innodb_table_stats.ibd        |
    |      14 | ./mysql/innodb_index_stats.ibd        |
    |      15 | ./mysql/slave_relay_log_info.ibd      |
    |      16 | ./mysql/slave_master_info.ibd         |
    |      17 | ./mysql/slave_worker_info.ibd         |
    |      18 | ./mysql/gtid_executed.ibd             |
    |      19 | ./mysql/server_cost.ibd               |
    |      20 | ./mysql/engine_cost.ibd               |
    |      21 | ./sys/sys_config.ibd                  |
    |      23 | ./test/t1.ibd                         |
    |      26 | /home/user/test/test/t2.ibd           |
    +---------+---------------------------------------+
```

This query retrieves the FILE_ID and FILE_NAME for the InnoDB global temporary tablespace. Global temporary tablespace file names are prefixed by ibtmp.

```
mysql> SELECT FILE_ID, FILE_NAME FROM INFORMATION_SCHEMA.FILES
       WHERE FILE_NAME LIKE '%ibtmp%';
+---------+-----------+
| FILE_ID | FILE_NAME |
+---------+-----------+
```

```
|      22 | ./ibtmp1  |
+---------+-----------+
```

Similarly, `InnoDB` undo tablespace file names are prefixed by `undo`. The following query returns the `FILE_ID` and `FILE_NAME` for `InnoDB` undo tablespaces.

```
mysql> SELECT FILE_ID, FILE_NAME FROM INFORMATION_SCHEMA.FILES
       WHERE FILE_NAME LIKE '%undo%';
```

15.15 InnoDB Integration with MySQL Performance Schema

This section provides a brief introduction to `InnoDB` integration with Performance Schema. For comprehensive Performance Schema documentation, see Chapter 26, *MySQL Performance Schema*.

You can profile certain internal `InnoDB` operations using the MySQL Performance Schema feature. This type of tuning is primarily for expert users who evaluate optimization strategies to overcome performance bottlenecks. DBAs can also use this feature for capacity planning, to see whether their typical workload encounters any performance bottlenecks with a particular combination of CPU, RAM, and disk storage; and if so, to judge whether performance can be improved by increasing the capacity of some part of the system.

To use this feature to examine `InnoDB` performance:

- You must be generally familiar with how to use the Performance Schema feature. For example, you should know how enable instruments and consumers, and how to query `performance_schema` tables to retrieve data. For an introductory overview, see Section 26.1, “Performance Schema Quick Start”.
- You should be familiar with Performance Schema instruments that are available for `InnoDB`. To view `InnoDB`-related instruments, you can query the `setup_instruments` table for instrument names that contain '`innodb`'.

```
mysql> SELECT *
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%innodb%';
+---------------------------------------------------------+---------+-------+
| NAME                                                    | ENABLED | TIMED |
+---------------------------------------------------------+---------+-------+
| wait/synch/mutex/innodb/commit_cond_mutex               | NO      | NO    |
| wait/synch/mutex/innodb/innobase_share_mutex            | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/cache_last_read_mutex           | NO      | NO    |
| wait/synch/mutex/innodb/dict_foreign_err_mutex          | NO      | NO    |
| wait/synch/mutex/innodb/dict_sys_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/recalc_pool_mutex               | NO      | NO    |
...
| wait/io/file/innodb/innodb_data_file                    | YES     | YES   |
| wait/io/file/innodb/innodb_log_file                     | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file                    | YES     | YES   |
| stage/innodb/alter table (end)                          | YES     | YES   |
| stage/innodb/alter table (flush)                        | YES     | YES   |
| stage/innodb/alter table (insert)                       | YES     | YES   |
| stage/innodb/alter table (log apply index)              | YES     | YES   |
| stage/innodb/alter table (log apply table)              | YES     | YES   |
| stage/innodb/alter table (merge sort)                   | YES     | YES   |
| stage/innodb/alter table (read PK and internal sort)    | YES     | YES   |
| stage/innodb/buffer pool load                           | YES     | YES   |
| memory/innodb/buf_buf_pool                              | NO      | NO    |
| memory/innodb/dict_stats_bg_recalc_pool_t               | NO      | NO    |
| memory/innodb/dict_stats_index_map_t                    | NO      | NO    |
```

```
| memory/innodb/dict_stats_n_diff_on_level              | NO      | NO    |
| memory/innodb/other                                   | NO      | NO    |
| memory/innodb/row_log_buf                             | NO      | NO    |
| memory/innodb/row_merge_sort                          | NO      | NO    |
| memory/innodb/std                                     | NO      | NO    |
| memory/innodb/sync_debug_latches                      | NO      | NO    |
| memory/innodb/trx_sys_t::rw_trx_ids                   | NO      | NO    |
...
+-------------------------------------------------------+---------+-------+
155 rows in set (0.00 sec)
```

For additional information about the instrumented `InnoDB` objects, you can query Performance Schema instances tables, which provide additional information about instrumented objects. Instance tables relevant to `InnoDB` include:

- The `mutex_instances` table
- The `rwlock_instances` table
- The `cond_instances` table
- The `file_instances` table

Note

Mutexes and RW-locks related to the `InnoDB` buffer pool are not included in this coverage; the same applies to the output of the `SHOW ENGINE INNODB MUTEX` command.

For example, to view information about instrumented `InnoDB` file objects seen by the Performance Schema when executing file I/O instrumentation, you might issue the following query:

```
mysql> SELECT *
       FROM performance_schema.file_instances
       WHERE EVENT_NAME LIKE '%innodb%'\G
*************************** 1. row ***************************
 FILE_NAME: /path/to/mysql-8.0/data/ibdata1
EVENT_NAME: wait/io/file/innodb/innodb_data_file
OPEN_COUNT: 3
*************************** 2. row ***************************
 FILE_NAME: /path/to/mysql-8.0/data/ib_logfile0
EVENT_NAME: wait/io/file/innodb/innodb_log_file
OPEN_COUNT: 2
*************************** 3. row ***************************
 FILE_NAME: /path/to/mysql-8.0/data/ib_logfile1
EVENT_NAME: wait/io/file/innodb/innodb_log_file
OPEN_COUNT: 2
*************************** 4. row ***************************
 FILE_NAME: /path/to/mysql-8.0/data/mysql/engine_cost.ibd
EVENT_NAME: wait/io/file/innodb/innodb_data_file
OPEN_COUNT: 3
...
```

- You should be familiar with `performance_schema` tables that store `InnoDB` event data. Tables relevant to `InnoDB`-related events include:
  - The Wait Event tables, which store wait events.
  - The Summary tables, which provide aggregated information for terminated events over time. Summary tables include file I/O summary tables, which aggregate information about I/O operations.

- Stage Event tables, which store event data for `InnoDB` `ALTER TABLE` and buffer pool load operations. For more information, see Section 15.15.1, "Monitoring ALTER TABLE Progress for InnoDB Tables Using Performance Schema", and Monitoring Buffer Pool Load Progress Using Performance Schema.

If you are only interested in `InnoDB`-related objects, use the clause `WHERE EVENT_NAME LIKE '%innodb%'` or `WHERE NAME LIKE '%innodb%'` (as required) when querying these tables.

15.15.1 Monitoring ALTER TABLE Progress for InnoDB Tables Using Performance Schema

You can monitor `ALTER TABLE` progress for `InnoDB` tables using Performance Schema.

There are seven stage events that represent different phases of `ALTER TABLE`. Each stage event reports a running total of `WORK_COMPLETED` and `WORK_ESTIMATED` for the overall `ALTER TABLE` operation as it progresses through its different phases. `WORK_ESTIMATED` is calculated using a formula that takes into account all of the work that `ALTER TABLE` performs, and may be revised during `ALTER TABLE` processing. `WORK_COMPLETED` and `WORK_ESTIMATED` values are an abstract representation of all of the work performed by `ALTER TABLE`.

In order of occurrence, `ALTER TABLE` stage events include:

- `stage/innodb/alter table (read PK and internal sort)`: This stage is active when `ALTER TABLE` is in the reading-primary-key phase. It starts with `WORK_COMPLETED=0` and `WORK_ESTIMATED` set to the estimated number of pages in the primary key. When the stage is completed, `WORK_ESTIMATED` is updated to the actual number of pages in the primary key.
- `stage/innodb/alter table (merge sort)`: This stage is repeated for each index added by the `ALTER TABLE` operation.
- `stage/innodb/alter table (insert)`: This stage is repeated for each index added by the `ALTER TABLE` operation.
- `stage/innodb/alter table (log apply index)`: This stage includes the application of DML log generated while `ALTER TABLE` was running.
- `stage/innodb/alter table (flush)`: Before this stage begins, `WORK_ESTIMATED` is updated with a more accurate estimate, based on the length of the flush list.
- `stage/innodb/alter table (log apply table)`: This stage includes the application of concurrent DML log generated while `ALTER TABLE` was running. The duration of this phase depends on the extent of table changes. This phase is instant if no concurrent DML was run on the table.
- `stage/innodb/alter table (end)`: Includes any remaining work that appeared after the flush phase, such as reapplying DML that was executed on the table while `ALTER TABLE` was running.

Note

`InnoDB` `ALTER TABLE` stage events do not currently account for the addition of spatial indexes.

ALTER TABLE Monitoring Example Using Performance Schema

The following example demonstrates how to enable the `stage/innodb/alter table%` stage event instruments and related consumer tables to monitor `ALTER TABLE` progress. For information about Performance Schema stage event instruments and related consumers, see Section 26.12.5, "Performance Schema Stage Event Tables".

1. Enable the `stage/innodb/alter%` instruments:

```
mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'YES'
       WHERE NAME LIKE 'stage/innodb/alter%';
Query OK, 7 rows affected (0.00 sec)
Rows matched: 7  Changed: 7  Warnings: 0
```

2. Enable the stage event consumer tables, which include `events_stages_current`, `events_stages_history`, and `events_stages_history_long`.

```
mysql> UPDATE performance_schema.setup_consumers
       SET ENABLED = 'YES'
       WHERE NAME LIKE '%stages%';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

3. Run an `ALTER TABLE` operation. In this example, a `middle_name` column is added to the employees table of the employees sample database.

```
mysql> ALTER TABLE employees.employees ADD COLUMN middle_name varchar(14) AFTER first_name;
Query OK, 0 rows affected (9.27 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

4. Check the progress of the `ALTER TABLE` operation by querying the Performance Schema `events_stages_current` table. The stage event shown differs depending on which `ALTER TABLE` phase is currently in progress. The `WORK_COMPLETED` column shows the work completed. The `WORK_ESTIMATED` column provides an estimate of the remaining work.

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED
       FROM performance_schema.events_stages_current;
+----------------------------------------------------+----------------+----------------+
| EVENT_NAME                                         | WORK_COMPLETED | WORK_ESTIMATED |
+----------------------------------------------------+----------------+----------------+
| stage/innodb/alter table (read PK and internal sort) |            280 |           1245 |
+----------------------------------------------------+----------------+----------------+
1 row in set (0.01 sec)
```

   The `events_stages_current` table returns an empty set if the `ALTER TABLE` operation has completed. In this case, you can check the `events_stages_history` table to view event data for the completed operation. For example:

```
mysql> SELECT EVENT_NAME, WORK_COMPLETED, WORK_ESTIMATED
       FROM performance_schema.events_stages_history;
+----------------------------------------------------+----------------+----------------+
| EVENT_NAME                                         | WORK_COMPLETED | WORK_ESTIMATED |
+----------------------------------------------------+----------------+----------------+
| stage/innodb/alter table (read PK and internal sort) |            886 |           1213 |
| stage/innodb/alter table (flush)                   |           1213 |           1213 |
| stage/innodb/alter table (log apply table)         |           1597 |           1597 |
| stage/innodb/alter table (end)                     |           1597 |           1597 |
| stage/innodb/alter table (log apply table)         |           1981 |           1981 |
+----------------------------------------------------+----------------+----------------+
5 rows in set (0.00 sec)
```

   As shown above, the `WORK_ESTIMATED` value was revised during `ALTER TABLE` processing. The estimated work after completion of the initial stage is 1213. When `ALTER TABLE` processing completed, `WORK_ESTIMATED` was set to the actual value, which is 1981.

15.15.2 Monitoring InnoDB Mutex Waits Using Performance Schema

A mutex is a synchronization mechanism used in the code to enforce that only one thread at a given time can have access to a common resource. When two or more threads executing in the server need to access the same resource, the threads compete against each other. The first thread to obtain a lock on the mutex causes the other threads to wait until the lock is released.

For `InnoDB` mutexes that are instrumented, mutex waits can be monitored using Performance Schema. Wait event data collected in Performance Schema tables can help identify mutexes with the most waits or the greatest total wait time, for example.

The following example demonstrates how to enable `InnoDB` mutex wait instruments, how to enable associated consumers, and how to query wait event data.

1. To view available `InnoDB` mutex wait instruments, query the Performance Schema `setup_instruments` table. All `InnoDB` mutex wait instruments are disabled by default.

```
mysql> SELECT *
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%wait/synch/mutex/innodb%';
+-----------------------------------------------------------+---------+-------+
| NAME                                                      | ENABLED | TIMED |
+-----------------------------------------------------------+---------+-------+
| wait/synch/mutex/innodb/commit_cond_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/innobase_share_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_persisted_mutex           | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_flush_state_mutex        | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_LRU_list_mutex           | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_free_list_mutex          | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_free_mutex           | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_hash_mutex           | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/cache_last_read_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/dict_foreign_err_mutex            | NO      | NO    |
| wait/synch/mutex/innodb/dict_persist_dirty_tables_mutex   | NO      | NO    |
| wait/synch/mutex/innodb/dict_sys_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/recalc_pool_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/fil_system_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/flush_list_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/fts_bg_threads_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/fts_delete_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/fts_optimize_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/fts_doc_id_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/log_flush_order_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/hash_table_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/ibuf_bitmap_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/ibuf_mutex                        | NO      | NO    |
| wait/synch/mutex/innodb/ibuf_pessimistic_insert_mutex     | NO      | NO    |
| wait/synch/mutex/innodb/log_sys_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/log_sys_write_mutex               | NO      | NO    |
| wait/synch/mutex/innodb/mutex_list_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/page_zip_stat_per_index_mutex     | NO      | NO    |
| wait/synch/mutex/innodb/purge_sys_pq_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/recv_sys_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/recv_writer_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/redo_rseg_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/noredo_rseg_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/rw_lock_list_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/rw_lock_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/srv_dict_tmpfile_mutex            | NO      | NO    |
| wait/synch/mutex/innodb/srv_innodb_monitor_mutex          | NO      | NO    |
| wait/synch/mutex/innodb/srv_misc_tmpfile_mutex            | NO      | NO    |
```

```
| wait/synch/mutex/innodb/srv_monitor_file_mutex          | NO      | NO    |
| wait/synch/mutex/innodb/buf_dblwr_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/trx_undo_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/trx_pool_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/trx_pool_manager_mutex          | NO      | NO    |
| wait/synch/mutex/innodb/srv_sys_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/lock_mutex                      | NO      | NO    |
| wait/synch/mutex/innodb/lock_wait_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/trx_mutex                       | NO      | NO    |
| wait/synch/mutex/innodb/srv_threads_mutex               | NO      | NO    |
| wait/synch/mutex/innodb/rtr_active_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/rtr_match_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/rtr_path_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/rtr_ssn_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/trx_sys_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/zip_pad_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/master_key_id_mutex             | NO      | NO    |
+---------------------------------------------------------+---------+-------+
```

2. Some `InnoDB` mutex instances are created at server startup and are only instrumented if the associated instrument is also enabled at server startup. To ensure that all `InnoDB` mutex instances are instrumented and enabled, add the following `performance-schema-instrument` rule to your MySQL configuration file:

```
performance-schema-instrument='wait/synch/mutex/innodb/%=ON'
```

If you do not require wait event data for all `InnoDB` mutexes, you can disable specific instruments by adding additional `performance-schema-instrument` rules to your MySQL configuration file. For example, to disable `InnoDB` mutex wait event instruments related to full-text search, add the following rule:

```
performance-schema-instrument='wait/synch/mutex/innodb/fts%=OFF'
```

Note

Rules with a longer prefix such as `wait/synch/mutex/innodb/fts%` take precedence over rules with shorter prefixes such as `wait/synch/mutex/innodb/%`.

After adding the `performance-schema-instrument` rules to your configuration file, restart the server. All the `InnoDB` mutexes except for those related to full text search are enabled. To verify, query the `setup_instruments` table. The `ENABLED` and `TIMED` columns should be set to `YES` for the instruments that you enabled.

```
mysql> SELECT *
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%wait/synch/mutex/innodb%';
+---------------------------------------------------------+---------+-------+
| NAME                                                    | ENABLED | TIMED |
+---------------------------------------------------------+---------+-------+
| wait/synch/mutex/innodb/commit_cond_mutex               | YES     | YES   |
| wait/synch/mutex/innodb/innobase_share_mutex            | YES     | YES   |
| wait/synch/mutex/innodb/autoinc_mutex                   | YES     | YES   |
...
| wait/synch/mutex/innodb/master_key_id_mutex             | YES     | YES   |
+---------------------------------------------------------+---------+-------+
49 rows in set (0.00 sec)
```

3. Enable wait event consumers by updating the `setup_consumers` table. Wait event consumers are disabled by default.

```
mysql> UPDATE performance_schema.setup_consumers
       SET enabled = 'YES'
       WHERE name like 'events_waits%';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

   You can verify that wait event consumers are enabled by querying the `setup_consumers` table. The `events_waits_current`, `events_waits_history`, and `events_waits_history_long` consumers should be enabled.

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | YES     |
| events_statements_history_long | NO      |
| events_transactions_current    | YES     |
| events_transactions_history    | YES     |
| events_transactions_history_long | NO    |
| events_waits_current           | YES     |
| events_waits_history           | YES     |
| events_waits_history_long      | YES     |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
15 rows in set (0.00 sec)
```

4. Once instruments and consumers are enabled, run the workload that you want to monitor. In this example, the `mysqlslap` load emulation client is used to simulate a workload.

```
shell> ./mysqlslap --auto-generate-sql --concurrency=100 --iterations=10
       --number-of-queries=1000 --number-char-cols=6 --number-int-cols=6;
```

5. Query the wait event data. In this example, wait event data is queried from the `events_waits_summary_global_by_event_name` table which aggregates data found in the `events_waits_current`, `events_waits_history`, and `events_waits_history_long` tables. Data is summarized by event name (`EVENT_NAME`), which is the name of the instrument that produced the event. Summarized data includes:

   - `COUNT_STAR`

     The number of summarized wait events.

   - `SUM_TIMER_WAIT`

     The total wait time of the summarized timed wait events.

   - `MIN_TIMER_WAIT`

     The minimum wait time of the summarized timed wait events.

- AVG_TIMER_WAIT

  The average wait time of the summarized timed wait events.

- MAX_TIMER_WAIT

  The maximum wait time of the summarized timed wait events.

The following query returns the instrument name (EVENT_NAME), the number of wait events (COUNT_STAR), and the total wait time for the events for that instrument (SUM_TIMER_WAIT). Because waits are timed in picoseconds (trillionths of a second) by default, wait times are divided by 1000000000 to show wait times in milliseconds. Data is presented in descending order, by the number of summarized wait events (COUNT_STAR). You can adjust the ORDER BY clause to order the data by total wait time.

```
mysql> SELECT EVENT_NAME, COUNT_STAR, SUM_TIMER_WAIT/1000000000 SUM_TIMER_WAIT_MS
       FROM performance_schema.events_waits_summary_global_by_event_name
       WHERE SUM_TIMER_WAIT > 0 AND EVENT_NAME LIKE 'wait/synch/mutex/innodb/%'
       ORDER BY COUNT_STAR DESC;
+-----------------------------------------------------------+------------+-------------------+
| EVENT_NAME                                                | COUNT_STAR | SUM_TIMER_WAIT_MS |
+-----------------------------------------------------------+------------+-------------------+
| wait/synch/mutex/innodb/trx_mutex                         |     201111 |           23.4719 |
| wait/synch/mutex/innodb/fil_system_mutex                  |      62244 |            9.6426 |
| wait/synch/mutex/innodb/redo_rseg_mutex                   |      48238 |            3.1135 |
| wait/synch/mutex/innodb/log_sys_mutex                     |      46113 |            2.0434 |
| wait/synch/mutex/innodb/trx_sys_mutex                     |      35134 |         1068.1588 |
| wait/synch/mutex/innodb/lock_mutex                        |      34872 |         1039.2589 |
| wait/synch/mutex/innodb/log_sys_write_mutex               |      17805 |         1526.0490 |
| wait/synch/mutex/innodb/dict_sys_mutex                    |      14912 |         1606.7348 |
| wait/synch/mutex/innodb/trx_undo_mutex                    |      10634 |            1.1424 |
| wait/synch/mutex/innodb/rw_lock_list_mutex                |       8538 |            0.1960 |
| wait/synch/mutex/innodb/buf_pool_free_list_mutex          |       5961 |            0.6473 |
| wait/synch/mutex/innodb/trx_pool_mutex                    |       4885 |         8821.7496 |
| wait/synch/mutex/innodb/buf_pool_LRU_list_mutex           |       4364 |            0.2077 |
| wait/synch/mutex/innodb/innobase_share_mutex              |       3212 |            0.2650 |
| wait/synch/mutex/innodb/flush_list_mutex                  |       3178 |            0.2349 |
| wait/synch/mutex/innodb/trx_pool_manager_mutex            |       2495 |            0.1310 |
| wait/synch/mutex/innodb/buf_pool_flush_state_mutex        |       1318 |            0.2161 |
| wait/synch/mutex/innodb/log_flush_order_mutex             |       1250 |            0.0893 |
| wait/synch/mutex/innodb/buf_dblwr_mutex                   |        951 |            0.0918 |
| wait/synch/mutex/innodb/recalc_pool_mutex                 |        670 |            0.0942 |
| wait/synch/mutex/innodb/dict_persist_dirty_tables_mutex   |        345 |            0.0414 |
| wait/synch/mutex/innodb/lock_wait_mutex                   |        303 |            0.1565 |
| wait/synch/mutex/innodb/autoinc_mutex                     |        196 |            0.0213 |
| wait/synch/mutex/innodb/autoinc_persisted_mutex           |        196 |            0.0175 |
| wait/synch/mutex/innodb/purge_sys_pq_mutex                |        117 |            0.0308 |
| wait/synch/mutex/innodb/srv_sys_mutex                     |         94 |            0.0077 |
| wait/synch/mutex/innodb/ibuf_mutex                        |         22 |            0.0086 |
| wait/synch/mutex/innodb/recv_sys_mutex                    |         12 |            0.0008 |
| wait/synch/mutex/innodb/srv_innodb_monitor_mutex          |          4 |            0.0009 |
| wait/synch/mutex/innodb/recv_writer_mutex                 |          1 |            0.0005 |
+-----------------------------------------------------------+------------+-------------------+
```

Note

The preceding result set includes wait event data produced during the startup process. To exclude this data, you can truncate the events_waits_summary_global_by_event_name table immediately after startup and before running your workload. However, the truncate operation itself may produce a negligible amount wait event data.

```
mysql> TRUNCATE performance_schema.events_waits_summary_global_by_event_name;
```

15.16 InnoDB Monitors

InnoDB monitors provide information about the InnoDB internal state. This information is useful for performance tuning.

15.16.1 InnoDB Monitor Types

There are two types of InnoDB monitor:

- The standard InnoDB Monitor displays the following types of information:
  - Work done by the main background thread
  - Semaphore waits
  - Data about the most recent foreign key and deadlock errors
  - Lock waits for transactions
  - Table and record locks held by active transactions
  - Pending I/O operations and related statistics
  - Insert buffer and adaptive hash index statistics
  - Redo log data
  - Buffer pool statistics
  - Row operation data
- The InnoDB Lock Monitor prints additional lock information as part of the standard InnoDB Monitor output.

15.16.2 Enabling InnoDB Monitors

When InnoDB monitors are enabled for periodic output, InnoDB writes the output to mysqld server standard error output (stderr) every 15 seconds, approximately.

InnoDB sends the monitor output to stderr rather than to stdout or fixed-size memory buffers to avoid potential buffer overflows.

On Windows, stderr is directed to the default log file unless configured otherwise. If you want to direct the output to the console window rather than to the error log, start the server from a command prompt in a console window with the --console option. For more information, see Error Logging on Windows.

On Unix and Unix-like systems, stderr is typically directed to the terminal unless configured otherwise. For more information, see Error Logging on Unix and Unix-Like Systems.

InnoDB monitors should only be enabled when you actually want to see monitor information because output generation causes some performance decrement. Also, if monitor output is directed to the error log, the log may become quite large if you forget to disable the monitor later.

Note

To assist with troubleshooting, InnoDB temporarily enables standard InnoDB Monitor output under certain conditions. For more information, see Section 15.20, "InnoDB Troubleshooting".

InnoDB monitor output begins with a header containing a timestamp and the monitor name. For example:

```
===================================
2014-10-16 18:37:29 0x7fc2a95c1700 INNODB MONITOR OUTPUT
===================================
```

The header for the standard InnoDB Monitor (INNODB MONITOR OUTPUT) is also used for the Lock Monitor because the latter produces the same output with the addition of extra lock information.

The innodb_status_output and innodb_status_output_locks system variables are used to enable the standard InnoDB Monitor and InnoDB Lock Monitor.

The PROCESS privilege is required to enable or disable InnoDB Monitors.

Enabling the Standard InnoDB Monitor

Enable the standard InnoDB Monitor by setting the innodb_status_output system variable to ON.

```
SET GLOBAL innodb_status_output=ON;
```

To disable the standard InnoDB Monitor, set innodb_status_output to OFF.

When you shut down the server, the innodb_status_output variable is set to the default OFF value.

Enabling the InnoDB Lock Monitor

InnoDB Lock Monitor data is printed with the InnoDB Standard Monitor output. Both the InnoDB Standard Monitor and InnoDB Lock Monitor must be enabled to have InnoDB Lock Monitor data printed periodically.

To enable the InnoDB Lock Monitor, set the innodb_status_output_locks system variable to ON. Both the InnoDB standard Monitor and InnoDB Lock Monitor must be enabled to have InnoDB Lock Monitor data printed periodically:

```
SET GLOBAL innodb_status_output=ON;
SET GLOBAL innodb_status_output_locks=ON;
```

To disable the InnoDB Lock Monitor, set innodb_status_output_locks to OFF. Set innodb_status_output to OFF to also disable the InnoDB Standard Monitor.

When you shut down the server, the innodb_status_output and innodb_status_output_locks variables are set to the default OFF value.

Note

To enable the InnoDB Lock Monitor for SHOW ENGINE INNODB STATUS output, you are only required to enable innodb_status_output_locks.

Obtaining Standard InnoDB Monitor Output On Demand

As an alternative to enabling the standard InnoDB Monitor for periodic output, you can obtain standard InnoDB Monitor output on demand using the SHOW ENGINE INNODB STATUS SQL statement, which

fetches the output to your client program. If you are using the `mysql` interactive client, the output is more readable if you replace the usual semicolon statement terminator with `\G`:

```
mysql> SHOW ENGINE INNODB STATUS\G
```

`SHOW ENGINE INNODB STATUS` output also includes `InnoDB` Lock Monitor data if the `InnoDB` Lock Monitor is enabled.

Directing Standard InnoDB Monitor Output to a Status File

Standard `InnoDB` Monitor output can be enabled and directed to a status file by specifying the `--innodb-status-file` option at startup. When this option is used, `InnoDB` creates a file named `innodb_status.pid` in the data directory and writes output to it every 15 seconds, approximately.

`InnoDB` removes the status file when the server is shut down normally. If an abnormal shutdown occurs, the status file may have to be removed manually.

The `--innodb-status-file` option is intended for temporary use, as output generation can affect performance, and the `innodb_status.pid` file can become quite large over time.

15.16.3 InnoDB Standard Monitor and Lock Monitor Output

The Lock Monitor is the same as the Standard Monitor except that it includes additional lock information. Enabling either monitor for periodic output turns on the same output stream, but the stream includes extra information if the Lock Monitor is enabled. For example, if you enable the Standard Monitor and Lock Monitor, that turns on a single output stream. The stream includes extra lock information until you disable the Lock Monitor.

Standard Monitor output is limited to 1MB when produced using the `SHOW ENGINE INNODB STATUS` statement. This limit does not apply to output written to server standard error output (`stderr`).

Example Standard Monitor output:

```
mysql> SHOW ENGINE INNODB STATUS\G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
2018-04-12 15:14:08 0x7f971c063700 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 4 seconds
-----------------
BACKGROUND THREAD
-----------------
srv_master_thread loops: 15 srv_active, 0 srv_shutdown, 1122 srv_idle
srv_master_thread log flush and writes: 0
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 24
OS WAIT ARRAY INFO: signal count 24
RW-shared spins 4, rounds 8, OS waits 4
RW-excl spins 2, rounds 60, OS waits 2
RW-sx spins 0, rounds 0, OS waits 0
Spin rounds per wait: 2.00 RW-shared, 30.00 RW-excl, 0.00 RW-sx
------------------------
LATEST FOREIGN KEY ERROR
------------------------
2018-04-12 14:57:24 0x7f97a9c91700 Transaction:
TRANSACTION 7717, ACTIVE 0 sec inserting
```

```
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1136, 3 row lock(s), undo log entries 3
MySQL thread id 8, OS thread handle 140289365317376, query id 14 localhost root update
INSERT INTO child VALUES (NULL, 1), (NULL, 2), (NULL, 3), (NULL, 4), (NULL, 5), (NULL, 6)
Foreign key constraint fails for table `test`.`child`:
,
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`) REFERENCES `parent` (`id`) ON DELETE
  CASCADE ON UPDATE CASCADE
Trying to add in child table, in index par_ind tuple:
DATA TUPLE: 2 fields;
 0: len 4; hex 80000003; asc     ;;
 1: len 4; hex 80000003; asc     ;;

But in parent table `test`.`parent`, in index PRIMARY,
the closest match we can find is record:
PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 80000004; asc     ;;
 1: len 6; hex 000000001e19; asc       ;;
 2: len 7; hex 81000001110137; asc       7;;

------------
TRANSACTIONS
------------
Trx id counter 7748
Purge done for trx's n:o < 7747 undo n:o < 0 state: running but idle
History list length 19
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 421764459790000, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 7747, ACTIVE 23 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 9, OS thread handle 140286987249408, query id 51 localhost root updating
DELETE FROM t WHERE i = 1
------- TRX HAS BEEN WAITING 23 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 4 page no 4 n bits 72 index GEN_CLUST_INDEX of table `test`.`t`
trx id 7747 lock_mode X waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000000202; asc       ;;
 1: len 6; hex 000000001e41; asc      A;;
 2: len 7; hex 820000008b0110; asc        ;;
 3: len 4; hex 80000001; asc     ;;

------------------
TABLE LOCK table `test`.`t` trx id 7747 lock mode IX
RECORD LOCKS space id 4 page no 4 n bits 72 index GEN_CLUST_INDEX of table `test`.`t`
trx id 7747 lock_mode X waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 4; compact format; info bits 0
 0: len 6; hex 000000000202; asc       ;;
 1: len 6; hex 000000001e41; asc      A;;
 2: len 7; hex 820000008b0110; asc        ;;
 3: len 4; hex 80000001; asc     ;;

--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (read thread)
I/O thread 4 state: waiting for i/o request (read thread)
I/O thread 5 state: waiting for i/o request (read thread)
I/O thread 6 state: waiting for i/o request (write thread)
I/O thread 7 state: waiting for i/o request (write thread)
I/O thread 8 state: waiting for i/o request (write thread)
I/O thread 9 state: waiting for i/o request (write thread)
Pending normal aio reads: [0, 0, 0, 0] , aio writes: [0, 0, 0, 0] ,
```

```
 ibuf aio reads:, log i/o's:, sync i/o's:
Pending flushes (fsync) log: 0; buffer pool: 0
833 OS file reads, 605 OS file writes, 208 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 0.00 writes/s, 0.00 fsyncs/s
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2, 0 merges
merged operations:
 insert 0, delete mark 0, delete 0
discarded operations:
 insert 0, delete mark 0, delete 0
Hash table size 553253, node heap has 0 buffer(s)
Hash table size 553253, node heap has 1 buffer(s)
Hash table size 553253, node heap has 3 buffer(s)
Hash table size 553253, node heap has 0 buffer(s)
Hash table size 553253, node heap has 0 buffer(s)
Hash table size 553253, node heap has 0 buffer(s)
Hash table size 553253, node heap has 0 buffer(s)
Hash table size 553253, node heap has 0 buffer(s)
0.00 hash searches/s, 0.00 non-hash searches/s
---
LOG
---
Log sequence number          19643450
Log buffer assigned up to    19643450
Log buffer completed up to   19643450
Log written up to            19643450
Log flushed up to            19643450
Added dirty pages up to      19643450
Pages flushed up to          19643450
Last checkpoint at           19643450
129 log i/o's done, 0.00 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total large memory allocated 2198863872
Dictionary memory allocated 409606
Buffer pool size   131072
Free buffers       130095
Database pages     973
Old database pages 0
Modified db pages  0
Pending reads      0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 0, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 810, created 163, written 404
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 973, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
----------------------
INDIVIDUAL BUFFER POOL INFO
----------------------
---BUFFER POOL 0
Buffer pool size   65536
Free buffers       65043
Database pages     491
Old database pages 0
Modified db pages  0
Pending reads      0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 0, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 411, created 80, written 210
```

```
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 491, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
---BUFFER POOL 1
Buffer pool size   65536
Free buffers       65052
Database pages     482
Old database pages 0
Modified db pages  0
Pending reads      0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 0, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 399, created 83, written 194
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 482, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
0 read views open inside InnoDB
Process ID=5772, Main thread ID=140286437054208 , state=sleeping
Number of rows inserted 57, updated 354, deleted 4, read 4421
0.00 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
```

Standard Monitor Output Sections

For a description of each metric reported by the Standard Monitor, refer to the Metrics chapter in the Oracle Enterprise Manager for MySQL Database User's Guide.

- `Status`

  This section shows the timestamp, the monitor name, and the number of seconds that per-second averages are based on. The number of seconds is the elapsed time between the current time and the last time `InnoDB` Monitor output was printed.

- `BACKGROUND THREAD`

  The `srv_master_thread` lines shows work done by the main background thread.

- `SEMAPHORES`

  This section reports threads waiting for a semaphore and statistics on how many times threads have needed a spin or a wait on a mutex or a rw-lock semaphore. A large number of threads waiting for semaphores may be a result of disk I/O, or contention problems inside `InnoDB`. Contention can be due to heavy parallelism of queries or problems in operating system thread scheduling. Setting the `innodb_thread_concurrency` system variable smaller than the default value might help in such situations. The `Spin rounds per wait` line shows the number of spinlock rounds per OS wait for a mutex.

  Mutex metrics are reported by `SHOW ENGINE INNODB MUTEX`.

- `LATEST FOREIGN KEY ERROR`

This section provides information about the most recent foreign key constraint error. It is not present if no such error has occurred. The contents include the statement that failed as well as information about the constraint that failed and the referenced and referencing tables.

- `LATEST DETECTED DEADLOCK`

  This section provides information about the most recent deadlock. It is not present if no deadlock has occurred. The contents show which transactions are involved, the statement each was attempting to execute, the locks they have and need, and which transaction `InnoDB` decided to roll back to break the deadlock. The lock modes reported in this section are explained in Section 15.7.1, “InnoDB Locking”.

- `TRANSACTIONS`

  If this section reports lock waits, your applications might have lock contention. The output can also help to trace the reasons for transaction deadlocks.

- `FILE I/O`

  This section provides information about threads that `InnoDB` uses to perform various types of I/O. The first few of these are dedicated to general `InnoDB` processing. The contents also display information for pending I/O operations and statistics for I/O performance.

  The number of these threads are controlled by the `innodb_read_io_threads` and `innodb_write_io_threads` parameters. See Section 15.13, “InnoDB Startup Options and System Variables”.

- `INSERT BUFFER AND ADAPTIVE HASH INDEX`

  This section shows the status of the `InnoDB` insert buffer (also referred to as the change buffer) and the adaptive hash index.

  For related information, see Section 15.5.2, “Change Buffer”, and Section 15.5.3, “Adaptive Hash Index”.

- `LOG`

  This section displays information about the `InnoDB` log. The contents include the current log sequence number, how far the log has been flushed to disk, and the position at which `InnoDB` last took a checkpoint. (See Section 15.11.3, “InnoDB Checkpoints”.) The section also displays information about pending writes and write performance statistics.

- `BUFFER POOL AND MEMORY`

  This section gives you statistics on pages read and written. You can calculate from these numbers how many data file I/O operations your queries currently are doing.

  For buffer pool statistics descriptions, see Monitoring the Buffer Pool Using the InnoDB Standard Monitor. For additional information about the operation of the buffer pool, see Section 15.5.1, “Buffer Pool”.

- `ROW OPERATIONS`

  This section shows what the main thread is doing, including the number and performance rate for each type of row operation.

15.17 InnoDB Backup and Recovery

This section covers topics related to `InnoDB` backup and recovery.

- For information about backup techniques applicable to `InnoDB`, see Section 15.17.1, “InnoDB Backup”.
- For information about point-in-time recovery, recovery from disk failure or corruption, and how `InnoDB` performs crash recovery, see Section 15.17.2, “InnoDB Recovery”.

15.17.1 InnoDB Backup

The key to safe database management is making regular backups. Depending on your data volume, number of MySQL servers, and database workload, you can use these backup techniques, alone or in combination: hot backup with *MySQL Enterprise Backup*; cold backup by copying files while the MySQL server is shut down; logical backup with `mysqldump` for smaller data volumes or to record the structure of schema objects. Hot and cold backups are physical backups that copy actual data files, which can be used directly by the `mysqld` server for faster restore.

Using *MySQL Enterprise Backup* is the recommended method for backing up `InnoDB` data.

Note

`InnoDB` does not support databases that are restored using third-party backup tools.

Hot Backups

The `mysqlbackup` command, part of the MySQL Enterprise Backup component, lets you back up a running MySQL instance, including `InnoDB` tables, with minimal disruption to operations while producing a consistent snapshot of the database. When `mysqlbackup` is copying `InnoDB` tables, reads and writes to `InnoDB` tables can continue. MySQL Enterprise Backup can also create compressed backup files, and back up subsets of tables and databases. In conjunction with the MySQL binary log, users can perform point-in-time recovery. MySQL Enterprise Backup is part of the MySQL Enterprise subscription. For more details, see Section 30.2, “MySQL Enterprise Backup Overview”.

Cold Backups

If you can shut down the MySQL server, you can make a physical backup that consists of all files used by `InnoDB` to manage its tables. Use the following procedure:

1. Perform a slow shutdown of the MySQL server and make sure that it stops without errors.
2. Copy all `InnoDB` data files (`ibdata` files and `.ibd` files) into a safe place.
3. Copy all `InnoDB` log files (`ib_logfile` files) to a safe place.
4. Copy your `my.cnf` configuration file or files to a safe place.

Logical Backups Using mysqldump

In addition to physical backups, it is recommended that you regularly create logical backups by dumping your tables using `mysqldump`. A binary file might be corrupted without you noticing it. Dumped tables are stored into text files that are human-readable, so spotting table corruption becomes easier. Also, because the format is simpler, the chance for serious data corruption is smaller. `mysqldump` also has a `--single-transaction` option for making a consistent snapshot without locking out other clients. See Section 7.3.1, “Establishing a Backup Policy”.

Replication works with `InnoDB` tables, so you can use MySQL replication capabilities to keep a copy of your database at database sites requiring high availability. See Section 15.18, “InnoDB and MySQL Replication”.

15.17.2 InnoDB Recovery

This section describes InnoDB recovery. Topics include:

- Point-in-Time Recovery
- Recovery from Data Corruption or Disk Failure
- InnoDB Crash Recovery
- Tablespace Discovery During Crash Recovery

Point-in-Time Recovery

To recover an InnoDB database to the present from the time at which the physical backup was made, you must run MySQL server with binary logging enabled, even before taking the backup. To achieve point-in-time recovery after restoring a backup, you can apply changes from the binary log that occurred after the backup was made. See Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

Recovery from Data Corruption or Disk Failure

If your database becomes corrupted or disk failure occurs, you must perform the recovery using a backup. In the case of corruption, first find a backup that is not corrupted. After restoring the base backup, do a point-in-time recovery from the binary log files using mysqlbinlog and mysql to restore the changes that occurred after the backup was made.

In some cases of database corruption, it is enough to dump, drop, and re-create one or a few corrupt tables. You can use the CHECK TABLE statement to check whether a table is corrupt, although CHECK TABLE naturally cannot detect every possible kind of corruption.

In some cases, apparent database page corruption is actually due to the operating system corrupting its own file cache, and the data on disk may be okay. It is best to try restarting the computer first. Doing so may eliminate errors that appeared to be database page corruption. If MySQL still has trouble starting because of InnoDB consistency problems, see Section 15.20.2, “Forcing InnoDB Recovery” for steps to start the instance in recovery mode, which permits you to dump the data.

InnoDB Crash Recovery

To recover from a MySQL server crash, the only requirement is to restart the MySQL server. InnoDB automatically checks the logs and performs a roll-forward of the database to the present. InnoDB automatically rolls back uncommitted transactions that were present at the time of the crash. During recovery, mysqld displays output similar to this:

```
InnoDB: The log sequence number 664050266 in the system tablespace does not match
the log sequence number 685111586 in the ib_logfiles!
InnoDB: Database was not shutdown normally!
InnoDB: Starting crash recovery.
InnoDB: Using 'tablespaces.open.2' max LSN: 664075228
InnoDB: Doing recovery: scanned up to log sequence number 690354176
InnoDB: Doing recovery: scanned up to log sequence number 695597056
InnoDB: Doing recovery: scanned up to log sequence number 700839936
InnoDB: Doing recovery: scanned up to log sequence number 706082816
InnoDB: Doing recovery: scanned up to log sequence number 711325696
InnoDB: Doing recovery: scanned up to log sequence number 713458156
InnoDB: Applying a batch of 1467 redo log records ...
InnoDB: 10%
InnoDB: 20%
InnoDB: 30%
InnoDB: 40%
InnoDB: 50%
InnoDB: 60%
InnoDB: 70%
```

```
InnoDB: 80%
InnoDB: 90%
InnoDB: 100%
InnoDB: Apply batch completed!
InnoDB: 1 transaction(s) which must be rolled back or cleaned up in total 561887 row
operations to undo
InnoDB: Trx id counter is 4096
...
InnoDB: 8.0.1 started; log sequence number 713458156
InnoDB: Waiting for purge to start
InnoDB: Starting in background the rollback of uncommitted transactions
InnoDB: Rolling back trx with id 3596, 561887 rows to undo
...
./mysqld: ready for connections....
```

InnoDB crash recovery consists of several steps:

- Tablespace discovery

  Tablespace discovery is the process that InnoDB uses to identify tablespaces that require redo log application. See Tablespace Discovery During Crash Recovery.

- Redo log application

  Redo log application is performed during initialization, before accepting any connections. If all changes are flushed from the buffer pool to the tablespaces (ibdata* and *.ibd files) at the time of the shutdown or crash, redo log application is skipped. InnoDB also skips redo log application if redo log files are missing at startup.

  - The current maximum auto-increment counter value is written to the redo log each time the value changes, which makes it crash-safe. During recovery, InnoDB scans the redo log to collect counter value changes and applies the changes to the in-memory table object.

    For more information about how InnoDB handles auto-increment values, see Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”, and InnoDB AUTO_INCREMENT Counter Initialization.

  - When encountering index tree corruption, InnoDB writes a corruption flag to the redo log, which makes the corruption flag crash-safe. InnoDB also writes in-memory corruption flag data to an engine-private system table on each checkpoint. During recovery, InnoDB reads corruption flags from both locations and merges results before marking in-memory table and index objects as corrupt.

  - Removing redo logs to speed up recovery is not recommended, even if some data loss is acceptable. Removing redo logs should only be considered after a clean shutdown, with innodb_fast_shutdown set to 0 or 1.

- Roll back of incomplete transactions

  Incomplete transactions are any transactions that were active at the time of crash or fast shutdown. The time it takes to roll back an incomplete transaction can be three or four times the amount of time a transaction is active before it is interrupted, depending on server load.

  You cannot cancel transactions that are being rolled back. In extreme cases, when rolling back transactions is expected to take an exceptionally long time, it may be faster to start InnoDB with an innodb_force_recovery setting of 3 or greater. See Section 15.20.2, “Forcing InnoDB Recovery”.

- Change buffer merge

  Applying changes from the change buffer (part of the system tablespace) to leaf pages of secondary indexes, as the index pages are read to the buffer pool.

- Purge

  Deleting delete-marked records that are no longer visible to active transactions.

The steps that follow redo log application do not depend on the redo log (other than for logging the writes) and are performed in parallel with normal processing. Of these, only rollback of incomplete transactions is special to crash recovery. The insert buffer merge and the purge are performed during normal processing.

After redo log application, `InnoDB` attempts to accept connections as early as possible, to reduce downtime. As part of crash recovery, `InnoDB` rolls back transactions that were not committed or in `XA PREPARE` state when the server crashed. The rollback is performed by a background thread, executed in parallel with transactions from new connections. Until the rollback operation is completed, new connections may encounter locking conflicts with recovered transactions.

In most situations, even if the MySQL server was killed unexpectedly in the middle of heavy activity, the recovery process happens automatically and no action is required of the DBA. If a hardware failure or severe system error corrupted `InnoDB` data, MySQL might refuse to start. In this case, see Section 15.20.2, “Forcing InnoDB Recovery”.

For information about the binary log and `InnoDB` crash recovery, see Section 5.4.4, “The Binary Log”.

Tablespace Discovery During Crash Recovery

If, during recovery, `InnoDB` encounters redo logs written since the last checkpoint, the redo logs must be applied to affected tablespaces. The process that identifies affected tablespaces during recovery is referred to as *tablespace discovery*.

Tablespace discovery relies on the `innodb_directories` setting, which defines the directories to scan at startup for tablespace files. Directories defined by `innodb_data_home_dir`, `innodb_undo_directory`, and `datadir` are automatically appended to the `innodb_directories` argument value, regardless of whether the `innodb_directories` option is configured explicitly. Tablespace files defined with an absolute path or that reside outside of the directories automatically appended to the `innodb_directories` setting should be added to the `innodb_directories` setting. Recovery is terminated if any tablespace file referenced in a redo log has not been discovered previously.

15.18 InnoDB and MySQL Replication

MySQL replication works for `InnoDB` tables as it does for `MyISAM` tables. It is also possible to use replication in a way where the storage engine on the slave is not the same as the original storage engine on the master. For example, you can replicate modifications to an `InnoDB` table on the master to a `MyISAM` table on the slave. For more information see, Section 17.4.4, “Using Replication with Different Master and Slave Storage Engines”.

For information about setting up a new slave for a master, see Section 17.1.2.6, “Setting Up Replication Slaves”, and Section 17.1.2.5, “Choosing a Method for Data Snapshots”. To make a new slave without taking down the master or an existing slave, use the MySQL Enterprise Backup product.

Transactions that fail on the master do not affect replication at all. MySQL replication is based on the binary log where MySQL writes SQL statements that modify data. A transaction that fails (for example, because of a foreign key violation, or because it is rolled back) is not written to the binary log, so it is not sent to slaves. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”.

Replication and CASCADE. Cascading actions for `InnoDB` tables on the master are replicated on the slave *only* if the tables sharing the foreign key relation use `InnoDB` on both the master and slave. This is true whether you are using statement-based or row-based replication. Suppose that you have started replication, and then create two tables on the master using the following `CREATE TABLE` statements:

```
CREATE TABLE fc1 (
    i INT PRIMARY KEY,
    j INT
) ENGINE = InnoDB;

CREATE TABLE fc2 (
    m INT PRIMARY KEY,
    n INT,
    FOREIGN KEY ni (n) REFERENCES fc1 (i)
        ON DELETE CASCADE
) ENGINE = InnoDB;
```

Suppose that the slave does not have `InnoDB` support enabled. If this is the case, then the tables on the slave are created, but they use the `MyISAM` storage engine, and the `FOREIGN KEY` option is ignored. Now we insert some rows into the tables on the master:

```
master> INSERT INTO fc1 VALUES (1, 1), (2, 2);
Query OK, 2 rows affected (0.09 sec)
Records: 2  Duplicates: 0  Warnings: 0

master> INSERT INTO fc2 VALUES (1, 1), (2, 2), (3, 1);
Query OK, 3 rows affected (0.19 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

At this point, on both the master and the slave, table `fc1` contains 2 rows, and table `fc2` contains 3 rows, as shown here:

```
master> SELECT * FROM fc1;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
2 rows in set (0.00 sec)

master> SELECT * FROM fc2;
+---+------+
| m | n    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 3 |    1 |
+---+------+
3 rows in set (0.00 sec)

slave> SELECT * FROM fc1;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
2 rows in set (0.00 sec)

slave> SELECT * FROM fc2;
+---+------+
| m | n    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 3 |    1 |
+---+------+
```

```
3 rows in set (0.00 sec)
```

Now suppose that you perform the following `DELETE` statement on the master:

```
master> DELETE FROM fc1 WHERE i=1;
Query OK, 1 row affected (0.09 sec)
```

Due to the cascade, table `fc2` on the master now contains only 1 row:

```
master> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 2 | 2 |
+---+---+
1 row in set (0.00 sec)
```

However, the cascade does not propagate on the slave because on the slave the `DELETE` for `fc1` deletes no rows from `fc2`. The slave's copy of `fc2` still contains all of the rows that were originally inserted:

```
slave> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 1 | 1 |
| 3 | 1 |
| 2 | 2 |
+---+---+
3 rows in set (0.00 sec)
```

This difference is due to the fact that the cascading deletes are handled internally by the `InnoDB` storage engine, which means that none of the changes are logged.

15.19 InnoDB memcached Plugin

The `InnoDB memcached` plugin (`daemon_memcached`) provides an integrated `memcached` daemon that automatically stores and retrieves data from `InnoDB` tables, turning the MySQL server into a fast "key-value store". Instead of formulating queries in SQL, you can use simple `get`, `set`, and `incr` operations that avoid the performance overhead associated with SQL parsing and constructing a query optimization plan. You can also access the same `InnoDB` tables through SQL for convenience, complex queries, bulk operations, and other strengths of traditional database software.

This "NoSQL-style" interface uses the `memcached` API to speed up database operations, letting `InnoDB` handle memory caching using its buffer pool mechanism. Data modified through `memcached` operations such as `add`, `set`, and `incr` are stored to disk, in `InnoDB` tables. The combination of `memcached` simplicity and `InnoDB` reliability and consistency provides users with the best of both worlds, as explained in Section 15.19.1, "Benefits of the InnoDB memcached Plugin". For an architectural overview, see Section 15.19.2, "InnoDB memcached Architecture".

15.19.1 Benefits of the InnoDB memcached Plugin

This section outlines advantages the `daemon_memcached` plugin. The combination of `InnoDB` tables and `memcached` offers advantages over using either by themselves.

- Direct access to the `InnoDB` storage engine avoids the parsing and planning overhead of SQL.
- Running `memcached` in the same process space as the MySQL server avoids the network overhead of passing requests back and forth.

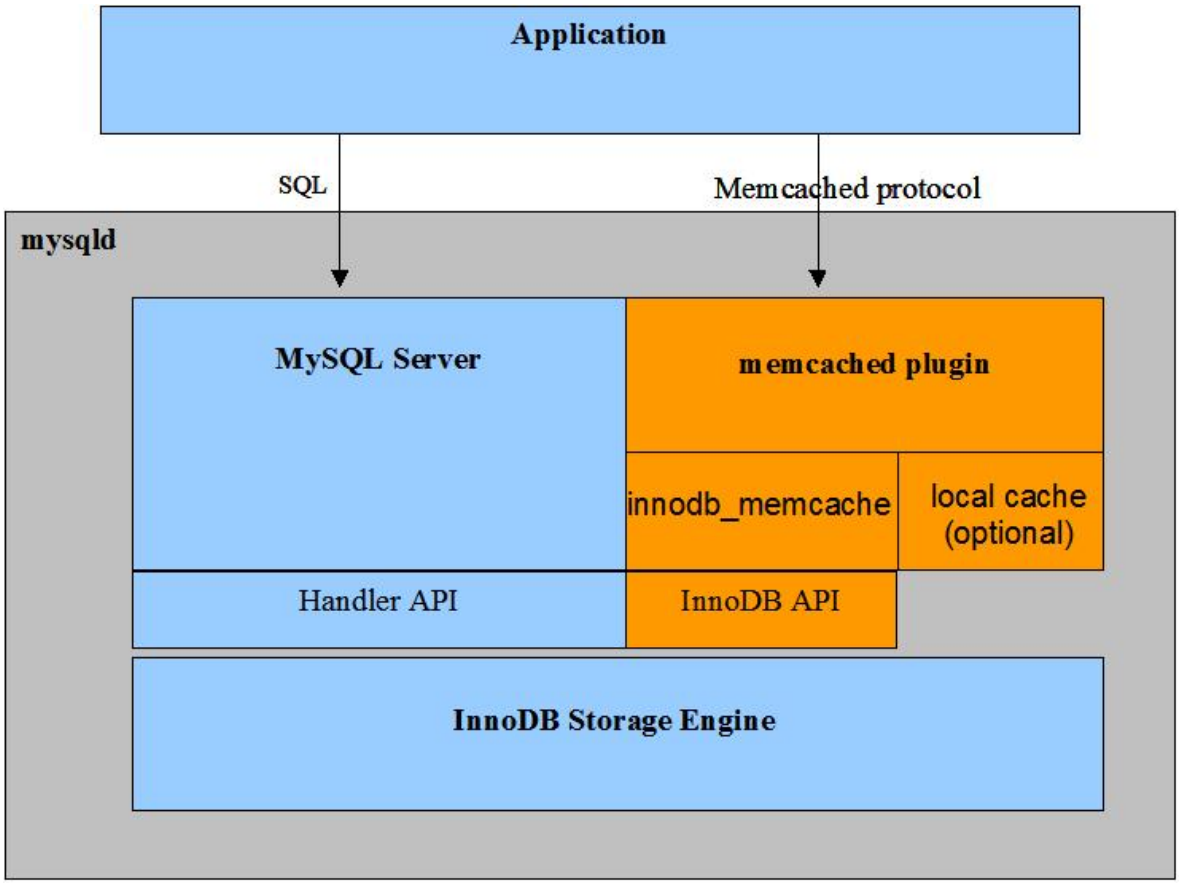
- Data written using the `memcached` protocol is transparently written to an `InnoDB` table, without going through the MySQL SQL layer. You can control frequency of writes to achieve higher raw performance when updating non-critical data.
- Data requested through the `memcached` protocol is transparently queried from an `InnoDB` table, without going through the MySQL SQL layer.
- Subsequent requests for the same data is served from the `InnoDB` buffer pool. The buffer pool handles the in-memory caching. You can tune performance of data-intensive operations using `InnoDB` configuration options.
- Data can be unstructured or structured, depending on the type of application. You can create a new table for data, or use existing tables.
- `InnoDB` can handle composing and decomposing multiple column values into a single `memcached` item value, reducing the amount of string parsing and concatenation required in your application. For example, you can store the string value `2|4|6|8` in the `memcached` cache, and have `InnoDB` split the value based on a separator character, then store the result in four numeric columns.
- The transfer between memory and disk is handled automatically, simplifying application logic.
- Data is stored in a MySQL database to protect against crashes, outages, and corruption.
- You can access the underlying `InnoDB` table through SQL for reporting, analysis, ad hoc queries, bulk loading, multi-step transactional computations, set operations such as union and intersection, and other operations suited to the expressiveness and flexibility of SQL.
- You can ensure high availability by using the `daemon_memcached` plugin on a master server in combination with MySQL replication.
- The integration of `memcached` with MySQL provides a way to make in-memory data persistent, so you can use it for more significant kinds of data. You can use more `add`, `incr`, and similar write operations in your application without concern that data could be lost. You can stop and start the `memcached` server without losing updates made to cached data. To guard against unexpected outages, you can take advantage of `InnoDB` crash recovery, replication, and backup capabilities.
- The way `InnoDB` does fast primary key lookups is a natural fit for `memcached` single-item queries. The direct, low-level database access path used by the `daemon_memcached` plugin is much more efficient for key-value lookups than equivalent SQL queries.
- The serialization features of `memcached`, which can turn complex data structures, binary files, or even code blocks into storeable strings, offer a simple way to get such objects into a database.
- Because you can access the underlying data through SQL, you can produce reports, search or update across multiple keys, and call functions such as `AVG()` and `MAX()` on `memcached` data. All of these operations are expensive or complicated using `memcached` by itself.
- You do not need to manually load data into `memcached` at startup. As particular keys are requested by an application, values are retrieved from the database automatically, and cached in memory using the `InnoDB` buffer pool.
- Because `memcached` consumes relatively little CPU, and its memory footprint is easy to control, it can run comfortably alongside a MySQL instance on the same system.
- Because data consistency is enforced by mechanisms used for regular `InnoDB` tables, you do not have to worry about stale `memcached` data or fallback logic to query the database in the case of a missing key.

15.19.2 InnoDB memcached Architecture

The `InnoDB` `memcached` plugin implements `memcached` as a MySQL plugin daemon that accesses the `InnoDB` storage engine directly, bypassing the MySQL SQL layer.

The following diagram illustrates how an application accesses data through the `daemon_memcached` plugin, compared with SQL.

Figure 15.4 MySQL Server with Integrated `memcached` Server



Features of the `daemon_memcached` plugin:

- `memcached` as a daemon plugin of `mysqld`. Both `mysqld` and `memcached` run in the same process space, with very low latency access to data.
- Direct access to `InnoDB` tables, bypassing the SQL parser, the optimizer, and even the Handler API layer.
- Standard `memcached` protocols, including the text-based protocol and the binary protocol. The `daemon_memcached` plugin passes all 55 compatibility tests of the `memcapable` command.
- Multi-column support. You can map multiple columns into the “value” part of the key-value store, with column values delimited by a user-specified separator character.
- By default, the `memcached` protocol is used to read and write data directly to `InnoDB`, letting MySQL manage in-memory caching using the `InnoDB` buffer pool. The default settings represent a combination of high reliability and the fewest surprises for database applications. For example, default settings avoid uncommitted data on the database side, or stale data returned for `memcached` `get` requests.

- Advanced users can configure the system as a traditional `memcached` server, with all data cached only in the `memcached` engine (memory caching), or use a combination of the "`memcached` engine" (memory caching) and the `InnoDB` `memcached` engine (`InnoDB` as back-end persistent storage).
- Control over how often data is passed back and forth between `InnoDB` and `memcached` operations through the `innodb_api_bk_commit_interval`, `daemon_memcached_r_batch_size`, and `daemon_memcached_w_batch_size` configuration options. Batch size options default to a value of 1 for maximum reliability.
- The ability to specify `memcached` options through the `daemon_memcached_option` configuration parameter. For example, you can change the port that `memcached` listens on, reduce the maximum number of simultaneous connections, change the maximum memory size for a key-value pair, or enable debugging messages for the error log.
- The `innodb_api_trx_level` configuration option controls the transaction isolation level on queries processed by `memcached`. Although `memcached` has no concept of transactions, you can use this option to control how soon `memcached` sees changes caused by SQL statements issued on the table used by the `daemon_memcached` plugin. By default, `innodb_api_trx_level` is set to `READ UNCOMMITTED`.
- The `innodb_api_enable_mdl` option can be used to lock the table at the MySQL level, so that the mapped table cannot be dropped or altered by DDL through the SQL interface. Without the lock, the table can be dropped from the MySQL layer, but kept in `InnoDB` storage until `memcached` or some other user stops using it. "MDL" stands for "metadata locking".

Differences Between InnoDB memcached and Traditional memcached

You may already be familiar with using `memcached` with MySQL, as described in Using MySQL with `memcached`. This section describes how features of the integrated `InnoDB` `memcached` plugin differ from traditional `memcached`.

- Installation: The `memcached` library comes with the MySQL server, making installation and setup relatively easy. Installation involves running the `innodb_memcached_config.sql` script to create a `demo_test` table for `memcached` to use, issuing an `INSTALL PLUGIN` statement to enable the `daemon_memcached` plugin, and adding desired `memcached` options to a MySQL configuration file or startup script. You might still install the traditional `memcached` distribution for additional utilities such as `memcp`, `memcat`, and `memcapable`.

  For comparison with traditional `memcached`, see Installing `memcached`.
- Deployment: With traditional `memcached`, it is typical to run large numbers of low-capacity `memcached` servers. A typical deployment of the `daemon_memcached` plugin, however, involves a smaller number of moderate or high-powered servers that are already running MySQL. The benefit of this configuration is in improving efficiency of individual database servers rather than exploiting unused memory or distributing lookups across large numbers of servers. In the default configuration, very little memory is used for `memcached`, and in-memory lookups are served from the `InnoDB` buffer pool, which automatically caches the most recently and frequently used data. As with a traditional MySQL server instance, keep the value of the `innodb_buffer_pool_size` configuration option as high as practical (without causing paging at the OS level), so that as much work as possible is performed in memory.

  For comparison with traditional `memcached`, see `memcached` Deployment.
- Expiry: By default (that is, using the `innodb_only` caching policy), the latest data from the `InnoDB` table is always returned, so the expiry options have no practical effect. If you change the caching policy to `caching` or `cache_only`, the expiry options work as usual, but requested data might be stale if it is updated in the underlying table before it expires from the memory cache.

For comparison with traditional `memcached`, see Data Expiry.

- Namespaces: `memcached` is like a large directory where you give files elaborate names with prefixes and suffixes to keep the files from conflicting. The `daemon_memcached` plugin lets you use similar naming conventions for keys, with one addition. Key names in the format *`@@table_id.key.table_id`* are decoded to reference a specific a table, using mapping data from the `innodb_memcache.containers` table. The *`key`* is looked up in or written to the specified table.

  The `@@` notation only works for individual calls to `get`, `add`, and `set` functions, but not others such as `incr` or `delete`. To designate a default table for subsequent `memcached` operations within a session, perform a `get` request using the `@@` notation with a *`table_id`*, but without the key portion. For example:

  ```
  get @@table_id
  ```

  Subsequent `get`, `set`, `incr`, `delete`, and other operations use the table designated by *`table_id`* in the `innodb_memcache.containers.name` column.

  For comparison with traditional `memcached`, see Using Namespaces.

- Hashing and distribution: The default configuration, which uses the `innodb_only` caching policy, is suitable for a traditional deployment configuration where all data is available on all servers, such as a set of replication slave servers.

  If you physically divide data, as in a sharded configuration, you can split data across several machines running the `daemon_memcached` plugin, and use the traditional `memcached` hashing mechanism to route requests to a particular machine. On the MySQL side, you would typically let all data be inserted by `add` requests to `memcached` so that appropriate values are stored in the database on the appropriate server.

  For comparison with traditional `memcached`, see `memcached` Hashing/Distribution Types.

- Memory usage: By default (with the `innodb_only` caching policy), the `memcached` protocol passes information back and forth with `InnoDB` tables, and the `InnoDB` buffer pool handles in-memory lookups instead of `memcached` memory usage growing and shrinking. Relatively little memory is used on the `memcached` side.

  If you switch the caching policy to `caching` or `cache_only`, the normal rules of `memcached` memory usage apply. Memory for `memcached` data values is allocated in terms of “slabs”. You can control slab size and maximum memory used for `memcached`.

  Either way, you can monitor and troubleshoot the `daemon_memcached` plugin using the familiar statistics system, accessed through the standard protocol, over a `telnet` session, for example. Extra utilities are not included with the `daemon_memcached` plugin. You can use the `memcached-tool` script to install a full `memcached` distribution.

  For comparison with traditional `memcached`, see Memory Allocation within `memcached`.

- Thread usage: MySQL threads and `memcached` threads co-exist on the same server. Limits imposed on threads by the operating system apply to the total number of threads.

  For comparison with traditional `memcached`, see `memcached` Thread Support.

- Log usage: Because the `memcached` daemon is run alongside the MySQL server and writes to `stderr`, the `-v`, `-vv`, and `-vvv` options for logging write output to the MySQL error log.

  For comparison with traditional `memcached`, see `memcached` Logs.

- `memcached` operations: Familiar `memcached` operations such as `get`, `set`, `add`, and `delete` are available. Serialization (that is, the exact string format representing complex data structures) depends on the language interface.

  For comparison with traditional `memcached`, see Basic `memcached` Operations.

- Using `memcached` as a MySQL front end: This is the primary purpose of the `InnoDB` `memcached` plugin. An integrated `memcached` daemon improves application performance, and having `InnoDB` handle data transfers between memory and disk simplifies application logic.

  For comparison with traditional `memcached`, see Using `memcached` as a MySQL Caching Layer.

- Utilities: The MySQL server includes the `libmemcached` library but not additional command-line utilities. To use commands such as `memcp`, `memcat`, and `memcapable` commands, install a full `memcached` distribution. When `memrm` and `memflush` remove items from the cache, the items are also removed from the underlying `InnoDB` table.

  For comparison with traditional `memcached`, see `libmemcached` Command-Line Utilities.

- Programming interfaces: You can access the MySQL server through the `daemon_memcached` plugin using all supported languages: C and C++, Java, Perl, Python, PHP, and Ruby. Specify the server hostname and port as with a traditional `memcached` server. By default, the `daemon_memcached` plugin listens on port `11211`. You can use both the text and binary protocols. You can customize the behavior of `memcached` functions at runtime. Serialization (that is, the exact string format representing complex data structures) depends on the language interface.

  For comparison with traditional `memcached`, see Developing a `memcached` Application.

- Frequently asked questions: MySQL has an extensive FAQ for traditional `memcached`. The FAQ is mostly applicable, except that using `InnoDB` tables as a storage medium for `memcached` data means that you can use `memcached` for more write-intensive applications than before, rather than as a read-only cache.

  See `memcached` FAQ.

15.19.3 Setting Up the InnoDB memcached Plugin

This section describes how to set up the `daemon_memcached` plugin on a MySQL server. Because the `memcached` daemon is tightly integrated with the MySQL server to avoid network traffic and minimize latency, you perform this process on each MySQL instance that uses this feature.

Note

Before setting up the `daemon_memcached` plugin, consult Section 15.19.5, "Security Considerations for the InnoDB memcached Plugin" to understand the security procedures required to prevent unauthorized access.

Prerequisites

- The `daemon_memcached` plugin is only supported on Linux, Solaris, and OS X platforms. Other operating systems are not supported.

- When building MySQL from source, you must build with `-DWITH_INNODB_MEMCACHED=ON`. This build option generates two shared libraries in the MySQL plugin directory (`plugin_dir`) that are required to run the `daemon_memcached` plugin:

  - `libmemcached.so`: the `memcached` daemon plugin to MySQL.

  - `innodb_engine.so`: an `InnoDB` API plugin to `memcached`.

- `libevent` must be installed.

  - If you did not build MySQL from source, the `libevent` library is not included in your installation. Use the installation method for your operating system to install `libevent` 1.4.12 or later. For example, depending on the operating system, you might use `apt-get`, `yum`, or `port install`. For example, on Ubuntu Linux, use:

```
sudo apt-get install libevent-dev
```

  - If you installed MySQL from a source code release, `libevent` 1.4.12 is bundled with the package and is located at the top level of the MySQL source code directory. If you use the bundled version of `libevent`, no action is required. If you want to use a local system version of `libevent`, you must build MySQL with the `-DWITH_LIBEVENT` build option set to `system` or `yes`.

Installing and Configuring the InnoDB memcached Plugin

1. Configure the `daemon_memcached` plugin so it can interact with `InnoDB` tables by running the `innodb_memcached_config.sql` configuration script, which is located in *`MYSQL_HOME`*`/share`. This script installs the `innodb_memcache` database with three required tables (`cache_policies`, `config_options`, and `containers`). It also installs the `demo_test` sample table in the `test` database.

```
mysql> source MYSQL_HOME/share/innodb_memcached_config.sql
```

   Running the `innodb_memcached_config.sql` script is a one-time operation. The tables remain in place if you later uninstall and re-install the `daemon_memcached` plugin.

```
mysql> USE innodb_memcache;
mysql> SHOW TABLES;
+----------------------------+
| Tables_in_innodb_memcache |
+----------------------------+
| cache_policies             |
| config_options             |
| containers                 |
+----------------------------+

mysql> USE test;
mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| demo_test      |
+----------------+
```

   Of these tables, the `innodb_memcache.containers` table is the most important. Entries in the `containers` table provide a mapping to `InnoDB` table columns. Each `InnoDB` table used with the `daemon_memcached` plugin requires an entry in the `containers` table.

   The `innodb_memcached_config.sql` script inserts a single entry in the `containers` table that provides a mapping for the `demo_test` table. It also inserts a single row of data into the `demo_test` table. This data allows you to immediately verify the installation after the setup is completed.

```
mysql> SELECT * FROM innodb_memcache.containers\G
```

```
*************************** 1. row ***************************
                  name: aaa
             db_schema: test
              db_table: demo_test
           key_columns: c1
         value_columns: c2
                 flags: c3
            cas_column: c4
    expire_time_column: c5
unique_idx_name_on_key: PRIMARY

mysql> SELECT * FROM test.demo_test;
+----+-----------------+------+------+------+
| c1 | c2              | c3   | c4   | c5   |
+----+-----------------+------+------+------+
| AA | HELLO, HELLO    |    8 |    0 |    0 |
+----+-----------------+------+------+------+
```

For more information about `innodb_memcache` tables and the `demo_test` sample table, see Section 15.19.8, “InnoDB memcached Plugin Internals”.

2. Activate the `daemon_memcached` plugin by running the `INSTALL PLUGIN` statement:

```
mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

Once the plugin is installed, it is automatically activated each time the MySQL server is restarted.

Verifying the InnoDB and memcached Setup

To verify the `daemon_memcached` plugin setup, use a `telnet` session to issue `memcached` commands. By default, the `memcached` daemon listens on port 11211.

1. Retrieve data from the `test.demo_test` table. The single row of data in the `demo_test` table has a key value of `AA`.

```
telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
get AA
VALUE AA 8 12
HELLO, HELLO
END
```

2. Insert data using a `set` command.

```
set BB 10 0 16
GOODBYE, GOODBYE
STORED
```

where:

- `set` is the command to store a value
- `BB` is the key
- `10` is a flag for the operation; ignored by `memcached` but may be used by the client to indicate any type of information; specify `0` if unused
- `0` is the expiration time (TTL); specify `0` if unused

- `16` is the length of the supplied value block in bytes
- `GOODBYE, GOODBYE` is the value that is stored

3. Verify that the data inserted is stored in MySQL by connecting to the MySQL server and querying the `test.demo_test` table.

```
mysql> SELECT * FROM test.demo_test;
+----+------------------+------+------+------+
| c1 | c2               | c3   | c4   | c5   |
+----+------------------+------+------+------+
| AA | HELLO, HELLO     |    8 |    0 |    0 |
| BB | GOODBYE, GOODBYE |   10 |    1 |    0 |
+----+------------------+------+------+------+
```

4. Return to the telnet session and retrieve the data that you inserted earlier using key `BB`.

```
get BB
VALUE BB 10 16
GOODBYE, GOODBYE
END
quit
```

If you shut down the MySQL server, which also shuts off the integrated `memcached` server, further attempts to access the `memcached` data fail with a connection error. Normally, the `memcached` data also disappears at this point, and you would require application logic to load the data back into memory when `memcached` is restarted. However, the `InnoDB memcached` plugin automates this process for you.

When you restart MySQL, `get` operations once again return the key-value pairs you stored in the earlier `memcached` session. When a key is requested and the associated value is not already in the memory cache, the value is automatically queried from the MySQL `test.demo_test` table.

Creating a New Table and Column Mapping

This example shows how to setup your own `InnoDB` table with the `daemon_memcached` plugin.

1. Create an `InnoDB` table. The table must have a key column with a unique index. The key column of the city table is `city_id`, which is defined as the primary key. The table must also include columns for `flags`, `cas`, and `expiry` values. There may be one or more value columns. The `city` table has three value columns (`name`, `state`, `country`).

Note

There is no special requirement with respect to column names as along as a valid mapping is added to the `innodb_memcache.containers` table.

```
mysql> CREATE TABLE city (
       city_id VARCHAR(32),
       name VARCHAR(1024),
       state VARCHAR(1024),
       country VARCHAR(1024),
       flags INT,
       cas BIGINT UNSIGNED,
       expiry INT,
       primary key(city_id)
       ) ENGINE=InnoDB;
```

2. Add an entry to the `innodb_memcache.containers` table so that the `daemon_memcached` plugin knows how to access the `InnoDB` table. The entry must satisfy the `innodb_memcache.containers` table definition. For a description of each field, see Section 15.19.8, “InnoDB memcached Plugin Internals”.

```
mysql> DESCRIBE innodb_memcache.containers;
+------------------------+-------------+------+-----+---------+-------+
| Field                  | Type        | Null | Key | Default | Extra |
+------------------------+-------------+------+-----+---------+-------+
| name                   | varchar(50)  | NO   | PRI | NULL    |       |
| db_schema              | varchar(250) | NO   |     | NULL    |       |
| db_table               | varchar(250) | NO   |     | NULL    |       |
| key_columns            | varchar(250) | NO   |     | NULL    |       |
| value_columns          | varchar(250) | YES  |     | NULL    |       |
| flags                  | varchar(250) | NO   |     | 0       |       |
| cas_column             | varchar(250) | YES  |     | NULL    |       |
| expire_time_column     | varchar(250) | YES  |     | NULL    |       |
| unique_idx_name_on_key | varchar(250) | NO   |     | NULL    |       |
+------------------------+-------------+------+-----+---------+-------+
```

   The `innodb_memcache.containers` table entry for the city table is defined as:

```
mysql> INSERT INTO `innodb_memcache`.`containers` (
       `name`, `db_schema`, `db_table`, `key_columns`, `value_columns`,
       `flags`, `cas_column`, `expire_time_column`, `unique_idx_name_on_key`)
       VALUES ('default', 'test', 'city', 'city_id', 'name|state|country',
       'flags','cas','expiry','PRIMARY');
```

   - `default` is specified for the `containers.name` column to configure the `city` table as the default `InnoDB` table to be used with the `daemon_memcached` plugin.
   - Multiple `InnoDB` table columns (`name`, `state`, `country`) are mapped to `containers.value_columns` using a “|” delimiter.
   - The `flags`, `cas_column`, and `expire_time_column` fields of the `innodb_memcache.containers` table are typically not significant in applications using the `daemon_memcached` plugin. However, a designated `InnoDB` table column is required for each. When inserting data, specify `0` for these columns if they are unused.

3. After updating the `innodb_memcache.containers` table, restart the `daemon_memcache` plugin to apply the changes.

```
mysql> UNINSTALL PLUGIN daemon_memcached;

mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

4. Using telnet, insert data into the `city` table using a `memcached set` command.

```
telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
set B 0 0 22
BANGALORE|BANGALORE|IN
STORED
```

5. Using MySQL, query the `test.city` table to verify that the data you inserted was stored.

```
mysql> SELECT * FROM test.city;
+---------+-----------+-----------+---------+-------+------+--------+
| city_id | name      | state     | country | flags | cas  | expiry |
+---------+-----------+-----------+---------+-------+------+--------+
| B       | BANGALORE | BANGALORE | IN      |     0 |    3 |      0 |
+---------+-----------+-----------+---------+-------+------+--------+
```

6. Using MySQL, insert additional data into the `test.city` table.

```
mysql> INSERT INTO city VALUES ('C','CHENNAI','TAMIL NADU','IN', 0, 0 ,0);
mysql> INSERT INTO city VALUES ('D','DELHI','DELHI','IN', 0, 0, 0);
mysql> INSERT INTO city VALUES ('H','HYDERABAD','TELANGANA','IN', 0, 0, 0);
mysql> INSERT INTO city VALUES ('M','MUMBAI','MAHARASHTRA','IN', 0, 0, 0);
```

Note

It is recommended that you specify a value of `0` for the `flags`, `cas_column`, and `expire_time_column` fields if they are unused.

7. Using telnet, issue a `memcached get` command to retrieve data you inserted using MySQL.

```
get H
VALUE H 0 22
HYDERABAD|TELANGANA|IN
END
```

Configuring the InnoDB memcached Plugin

Traditional `memcached` configuration options may be specified in a MySQL configuration file or a `mysqld` startup string, encoded in the argument of the `daemon_memcached_option` configuration parameter. `memcached` configuration options take effect when the plugin is loaded, which occurs each time the MySQL server is started.

For example, to make `memcached` listen on port 11222 instead of the default port 11211, specify `-p11222` as an argument of the `daemon_memcached_option` configuration option:

```
mysqld .... --daemon_memcached_option="-p11222"
```

Other `memcached` options can be encoded in the `daemon_memcached_option` string. For example, you can specify options to reduce the maximum number of simultaneous connections, change the maximum memory size for a key-value pair, or enable debugging messages for the error log, and so on.

There are also configuration options specific to the `daemon_memcached` plugin. These include:

- `daemon_memcached_engine_lib_name`: Specifies the shared library that implements the `InnoDB memcached` plugin. The default setting is `innodb_engine.so`.
- `daemon_memcached_engine_lib_path`: The path of the directory containing the shared library that implements the `InnoDB memcached` plugin. The default is NULL, representing the plugin directory.
- `daemon_memcached_r_batch_size`: Defines the batch commit size for read operations (`get`). It specifies the number of `memcached` read operations after which a commit occurs. `daemon_memcached_r_batch_size` is set to 1 by default so that every `get` request accesses the most recently committed data in the `InnoDB` table, whether the data was updated through `memcached` or by SQL. When the value is greater than 1, the counter for read operations is incremented with each `get` call. A `flush_all` call resets both read and write counters.

- `daemon_memcached_w_batch_size`: Defines the batch commit size for write operations (`set`, `replace`, `append`, `prepend`, `incr`, `decr`, and so on). `daemon_memcached_w_batch_size` is set to 1 by default so that no uncommitted data is lost in case of an outage, and so that SQL queries on the underlying table access the most recent data. When the value is greater than 1, the counter for write operations is incremented for each `add`, `set`, `incr`, `decr`, and `delete` call. A `flush_all` call resets both read and write counters.

By default, you do not need to modify `daemon_memcached_engine_lib_name` or `daemon_memcached_engine_lib_path`. You might configure these options if, for example, you want to use a different storage engine for `memcached` (such as the NDB `memcached` engine).

`daemon_memcached` plugin configuration parameters may be specified in the MySQL configuration file or in a `mysqld` startup string. They take effect when you load the `daemon_memcached` plugin.

When making changes to `daemon_memcached` plugin configuration, reload the plugin to apply the changes. To do so, issue the following statements:

```
mysql> UNINSTALL PLUGIN daemon_memcached;

mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

Configuration settings, required tables, and data are preserved when the plugin is restarted.

For additional information about enabling and disabling plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

15.19.4 InnoDB memcached Multiple get and Range Query Support

The `daemon_memcached` plugin supports multiple get operations (fetching multiple key-value pairs in a single `memcached` query) and range queries.

Multiple get Operations

The ability to fetch multiple key-value pairs in a single `memcached` query improves read performance by reducing communication traffic between the client and server. For `InnoDB`, it means fewer transactions and open-table operations.

The following example demonstrates multiple-get support. The example uses the `test.city` table described in Creating a New Table and Column Mapping.

```
mysql> USE test;
mysql> SELECT * FROM test.city;
+---------+-----------+-------------+---------+-------+------+--------+
| city_id | name      | state       | country | flags | cas  | expiry |
+---------+-----------+-------------+---------+-------+------+--------+
| B       | BANGALORE | BANGALORE   | IN      |     0 |    1 |      0 |
| C       | CHENNAI   | TAMIL NADU  | IN      |     0 |    0 |      0 |
| D       | DELHI     | DELHI       | IN      |     0 |    0 |      0 |
| H       | HYDERABAD | TELANGANA   | IN      |     0 |    0 |      0 |
| M       | MUMBAI    | MAHARASHTRA | IN      |     0 |    0 |      0 |
+---------+-----------+-------------+---------+-------+------+--------+
```

Run a `get` command to retrieve all values from the `city` table. The results are returned in a key-value pair sequence.

```
telnet 127.0.0.1 11211
```

```
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
get B C D H M
VALUE B 0 22
BANGALORE|BANGALORE|IN
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
VALUE M 0 21
MUMBAI|MAHARASHTRA|IN
END
```

When retrieving multiple values in a single `get` command, you can switch tables (using *`@@containers.name`* notation) to retrieve the value for the first key, but you cannot switch tables for subsequent keys. For example, the table switch in this example is valid:

```
get @@aaa.AA BB
VALUE @@aaa.AA 8 12
HELLO, HELLO
VALUE BB 10 16
GOODBYE, GOODBYE
END
```

Attempting to switch tables again in the same `get` command to retrieve a key value from a different table is not supported.

Range Queries

For range queries, the `daemon_memcached` plugin supports the following comparison operators: `<`, `>`, `<=`, `>=`. An operator must be preceded by an `@` symbol. When a range query finds multiple matching key-value pairs, results are returned in a key-value pair sequence.

The following examples demonstrate range query support. The examples use the `test.city` table described in Creating a New Table and Column Mapping.

```
mysql> SELECT * FROM test.city;
+---------+-----------+-------------+---------+-------+------+--------+
| city_id | name      | state       | country | flags | cas  | expiry |
+---------+-----------+-------------+---------+-------+------+--------+
| B       | BANGALORE | BANGALORE   | IN      |     0 |    1 |      0 |
| C       | CHENNAI   | TAMIL NADU  | IN      |     0 |    0 |      0 |
| D       | DELHI     | DELHI       | IN      |     0 |    0 |      0 |
| H       | HYDERABAD | TELANGANA   | IN      |     0 |    0 |      0 |
| M       | MUMBAI    | MAHARASHTRA | IN      |     0 |    0 |      0 |
+---------+-----------+-------------+---------+-------+------+--------+
```

Open a telnet session:

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
```

To get all values greater than `B`, enter `get @>B`:

```
get @>B
```

```
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
VALUE M 0 21
MUMBAI|MAHARASHTRA|IN
END
```

To get all values less than `M`, enter `get @<M`:

```
get @<M
VALUE B 0 22
BANGALORE|BANGALORE|IN
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
END
```

To get all values less than and including `M`, enter `get @<=M`:

```
get @<=M
VALUE B 0 22
BANGALORE|BANGALORE|IN
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
VALUE M 0 21
MUMBAI|MAHARASHTRA|IN
```

To get values greater than `B` but less than `M`, enter `get @>B@<M`:

```
get @>B@<M
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
END
```

A maximum of two comparison operators can be parsed, one being either a 'less than' (`@<`) or 'less than or equal to' (`@<=`) operator, and the other being either a 'greater than' (`@>`) or 'greater than or equal to' (`@>=`) operator. Any additional operators are assumed to be part of the key. For example, if you issue a `get` command with three operators, the third operator (`@>C`) is treated as part of the key, and the `get` command searches for values smaller than `M` and greater than `B@>C`.

```
get @<M@>B@>C
VALUE C 0 21
CHENNAI|TAMIL NADU|IN
VALUE D 0 14
DELHI|DELHI|IN
VALUE H 0 22
HYDERABAD|TELANGANA|IN
```

15.19.5 Security Considerations for the InnoDB memcached Plugin

Caution

Consult this section before deploying the `daemon_memcached` plugin on a production server, or even on a test server if the MySQL instance contains sensitive data.

Because `memcached` does not use an authentication mechanism by default, and the optional SASL authentication is not as strong as traditional DBMS security measures, only keep non-sensitive data in the MySQL instance that uses the `daemon_memcached` plugin, and wall off any servers that use this configuration from potential intruders. Do not allow `memcached` access to these servers from the Internet; only allow access from within a firewalled intranet, ideally from a subnet whose membership you can restrict.

Password-Protecting memcached Using SASL

SASL support provides the capability to protect your MySQL database from unauthenticated access through `memcached` clients. This section explains how to enable SASL with the `daemon_memcached` plugin. The steps are almost identical to those performed to enabled SASL for a traditional `memcached` server.

SASL stands for “Simple Authentication and Security Layer”, a standard for adding authentication support to connection-based protocols. `memcached` added SASL support in version 1.4.3.

SASL authentication is only supported with the binary protocol.

`memcached` clients are only able to access `InnoDB` tables that are registered in the `innodb_memcache.containers` table. Even though a DBA can place access restrictions on such tables, access through `memcached` applications cannot be controlled. For this reason, SASL support is provided to control access to `InnoDB` tables associated with the `daemon_memcached` plugin.

The following section shows how to build, enable, and test an SASL-enabled `daemon_memcached` plugin.

Building and Enabling SASL with the InnoDB memcached Plugin

By default, an SASL-enabled `daemon_memcached` plugin is not included in MySQL release packages, since an SASL-enabled `daemon_memcached` plugin requires building `memcached` with SASL libraries. To enable SASL support, download the MySQL source and rebuild the `daemon_memcached` plugin after downloading the SASL libraries:

1. Install the SASL development and utility libraries. For example, on Ubuntu, use `apt-get` to obtain the libraries:

   ```
   sudo apt-get -f install libsasl2-2 sasl2-bin libsasl2-2 libsasl2-dev libsasl2-modules
   ```

2. Build the `daemon_memcached` plugin shared libraries with SASL capability by adding `ENABLE_MEMCACHED_SASL=1` to your `cmake` options. `memcached` also provides *simple cleartext password support*, which facilitates testing. To enable simple cleartext password support, specify the `ENABLE_MEMCACHED_SASL_PWDB=1` `cmake` option.

   In summary, add following three `cmake` options:

   ```
   cmake ... -DWITH_INNODB_MEMCACHED=1 -DENABLE_MEMCACHED_SASL=1 -DENABLE_MEMCACHED_SASL_PWDB=1
   ```

3. Install the `daemon_memcached` plugin, as described in Section 15.19.3, “Setting Up the InnoDB memcached Plugin”.

4. Configure a user name and password file. (This example uses `memcached` simple cleartext password support.)

   a. In a file, create a user named `testname` and define the password as `testpasswd`:

   ```
   echo "testname:testpasswd:::::::" >/home/jy/memcached-sasl-db
   ```

   b. Configure the `MEMCACHED_SASL_PWDB` environment variable to inform `memcached` of the user name and password file:

   ```
   export MEMCACHED_SASL_PWDB=/home/jy/memcached-sasl-db
   ```

   c. Inform `memcached` that a cleartext password is used:

   ```
   echo "mech_list: plain" > /home/jy/work2/msasl/clients/memcached.conf
   export SASL_CONF_PATH=/home/jy/work2/msasl/clients
   ```

5. Enable SASL by restarting the MySQL server with the `memcached -S` option encoded in the `daemon_memcached_option` configuration parameter:

   ```
   mysqld ... --daemon_memcached_option="-S"
   ```

6. To test the setup, use an SASL-enabled client such as SASL-enabled libmemcached.

   ```
   memcp --servers=localhost:11211 --binary  --username=testname
     --password=password myfile.txt

   memcat --servers=localhost:11211 --binary --username=testname
     --password=password myfile.txt
   ```

   If you specify an incorrect user name or password, the operation is rejected with a `memcache error AUTHENTICATION FAILURE` message. In this case, examine the cleartext password set in the `memcached-sasl-db` file to verify that the credentials you supplied are correct.

There are other methods to test SASL authentication with `memcached`, but the method described above is the most straightforward.

15.19.6 Writing Applications for the InnoDB memcached Plugin

Typically, writing an application for the `InnoDB memcached` plugin involves some degree of rewriting or adapting existing code that uses MySQL or the `memcached` API.

- With the `daemon_memcached` plugin, instead of many traditional `memcached` servers running on low-powered machines, you have the same number of `memcached` servers as MySQL servers, running on relatively high-powered machines with substantial disk storage and memory. You might reuse some existing code that works with the `memcached` API, but adaptation is likely required due to the different server configuration.
- The data stored through the `daemon_memcached` plugin goes into `VARCHAR`, `TEXT`, or `BLOB` columns, and must be converted to do numeric operations. You can perform the conversion on the application side, or by using the `CAST()` function in queries.

- Coming from a database background, you might be used to general-purpose SQL tables with many columns. The tables accessed by `memcached` code likely have only a few or even a single column holding data values.
- You might adapt parts of your application that perform single-row queries, inserts, updates, or deletes, to improve performance in critical sections of code. Both queries (read) and DML (write) operations can be substantially faster when performed through the `InnoDB` `memcached` interface. The performance improvement for writes is typically greater than the performance improvement for reads, so you might focus on adapting code that performs logging or records interactive choices on a website.

The following sections explore these points in more detail.

15.19.6.1 Adapting an Existing MySQL Schema for the InnoDB memcached Plugin

Consider these aspects of `memcached` applications when adapting an existing MySQL schema or application to use the `daemon_memcached` plugin:

- `memcached` keys cannot contain spaces or newlines, because these characters are used as separators in the ASCII protocol. If you are using lookup values that contain spaces, transform or hash them into values without spaces before using them as keys in calls to `add()`, `set()`, `get()`, and so on. Although theoretically these characters are allowed in keys in programs that use the binary protocol, you should restrict the characters used in keys to ensure compatibility with a broad range of clients.
- If there is a short numeric primary key column in an `InnoDB` table, use it as the unique lookup key for `memcached` by converting the integer to a string value. If the `memcached` server is used for multiple applications, or with more than one `InnoDB` table, consider modifying the name to ensure that it is unique. For example, prepend the table name, or the database name and the table name, before the numeric value.

Note

The `daemon_memcached` plugin supports inserts and reads on mapped `InnoDB` tables that have an `INTEGER` defined as the primary key.

- You cannot use a partitioned table for data queried or stored using `memcached`.
- The `memcached` protocol passes numeric values around as strings. To store numeric values in the underlying `InnoDB` table, to implement counters that can be used in SQL functions such as `SUM()` or `AVG()`, for example:
  - Use `VARCHAR` columns with enough characters to hold all the digits of the largest expected number (and additional characters if appropriate for the negative sign, decimal point, or both).
  - In any query that performs arithmetic using column values, use the `CAST()` function to convert the values from string to integer, or to some other numeric type. For example:

```
# Alphabetic entries are returned as zero.

SELECT CAST(c2 as unsigned integer) FROM demo_test;

# Since there could be numeric values of 0, can't disqualify them.
# Test the string values to find the ones that are integers, and average only those.

SELECT AVG(cast(c2 as unsigned integer)) FROM demo_test
  WHERE c2 BETWEEN '0' and '9999999999';

# Views let you hide the complexity of queries. The results are already converted;
# no need to repeat conversion functions and WHERE clauses each time.
```

```
CREATE VIEW numbers AS SELECT c1 KEY, CAST(c2 AS UNSIGNED INTEGER) val
  FROM demo_test WHERE c2 BETWEEN '0' and '9999999999';
SELECT SUM(val) FROM numbers;
```

Note

Any alphabetic values in the result set are converted into 0 by the call to `CAST()`. When using functions such as `AVG()`, which depend on the number of rows in the result set, include `WHERE` clauses to filter out non-numeric values.

- If the `InnoDB` column used as a key could have values longer than 250 bytes, hash the value to less than 250 bytes.
- To use an existing table with the `daemon_memcached` plugin, define an entry for it in the `innodb_memcache.containers` table. To make that table the default for all `memcached` requests, specify a value of `default` in the `name` column, then restart the MySQL server to make the change take effect. If you use multiple tables for different classes of `memcached` data, set up multiple entries in the `innodb_memcache.containers` table with `name` values of your choice, then issue a `memcached` request in the form of `get @@name` or `set @@name` within the application to specify the table to be used for subsequent `memcached` requests.

  For an example of using a table other than the predefined `test.demo_test` table, see Example 15.13, "Using Your Own Table with an InnoDB memcached Application". For the required table layout, see Section 15.19.8, "InnoDB memcached Plugin Internals".
- To use multiple `InnoDB` table column values with `memcached` key-value pairs, specify column names separated by comma, semicolon, space, or pipe characters in the `value_columns` field of the `innodb_memcache.containers` entry for the `InnoDB` table. For example, specify `col1,col2,col3` or `col1|col2|col3` in the `value_columns` field.

  Concatenate the column values into a single string using the pipe character as a separator before passing the string to `memcached add` or `set` calls. The string is unpacked automatically into the correct column. Each `get` call returns a single string containing the column values that is also delimited by the pipe character. You can unpack the values using the appropriate application language syntax.

Example 15.13 Using Your Own Table with an InnoDB memcached Application

This example shows how to use your own table with a sample Python application that uses `memcached` for data manipulation.

The example assumes that the `daemon_memcached` plugin is installed as described in Section 15.19.3, "Setting Up the InnoDB memcached Plugin". It also assumes that your system is configured to run a Python script that uses the `python-memcache` module.

1. Create the `multicol` table which stores country information including population, area, and driver side data (`'R'` for right and `'L'` for left).

```
mysql> USE test;

mysql> CREATE TABLE `multicol` (
        `country` varchar(128) NOT NULL DEFAULT '',
        `population` varchar(10) DEFAULT NULL,
        `area_sq_km` varchar(9) DEFAULT NULL,
        `drive_side` varchar(1) DEFAULT NULL,
        `c3` int(11) DEFAULT NULL,
        `c4` bigint(20) unsigned DEFAULT NULL,
        `c5` int(11) DEFAULT NULL,
       PRIMARY KEY (`country`)
       ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

2. Insert a record into the `innodb_memcache.containers` table so that the `daemon_memcached` plugin can access the `multicol` table.

```
mysql> INSERT INTO innodb_memcache.containers
       (name,db_schema,db_table,key_columns,value_columns,flags,cas_column,
       expire_time_column,unique_idx_name_on_key)
       VALUES
       ('bbb','test','multicol','country','population,area_sq_km,drive_side',
       'c3','c4','c5','PRIMARY');

mysql> COMMIT;
```

   - The `innodb_memcache.containers` record for the `multicol` table specifies a `name` value of `'bbb'`, which is the table identifier.

     Note

     If a single `InnoDB` table is used for all `memcached` applications, the `name` value can be set to `default` to avoid using `@@` notation to switch tables.

   - The `db_schema` column is set to `test`, which is the name of the database where the `multicol` table resides.
   - The `db_table` column is set to `multicol`, which is the name of the `InnoDB` table.
   - `key_columns` is set to the unique `country` column. The `country` column is defined as the primary key in the `multicol` table definition.
   - Rather than a single `InnoDB` table column to hold a composite data value, data is divided among three table columns (`population`, `area_sq_km`, and `drive_side`). To accommodate multiple value columns, a comma-separated list of columns is specified in the `value_columns` field. The columns defined in the `value_columns` field are the columns used when storing or retrieving values.
   - Values for the `flags`, `expire_time`, and `cas_column` fields are based on values used in the `demo.test` sample table. These fields are typically not significant in applications that use the `daemon_memcached` plugin because MySQL keeps data synchronized, and there is no need to worry about data expiring or becoming stale.
   - The `unique_idx_name_on_key` field is set to `PRIMARY`, which refers to the primary index defined on the unique `country` column in the `multicol` table.

3. Copy the sample Python application into a file. In this example, the sample script is copied to a file named `multicol.py`.

   The sample Python application inserts data into the `multicol` table and retrieves data for all keys, demonstrating how to access an `InnoDB` table through the `daemon_memcached` plugin.

```
import sys, os
import memcache

def connect_to_memcached():
  memc = memcache.Client(['127.0.0.1:11211'], debug=0);
  print "Connected to memcached."
  return memc

def banner(message):
  print
```

```
  print "=" * len(message)
  print message
  print "=" * len(message)

country_data = [
("Canada","34820000","9984670","R"),
("USA","314242000","9826675","R"),
("Ireland","6399152","84421","L"),
("UK","62262000","243610","L"),
("Mexico","113910608","1972550","R"),
("Denmark","5543453","43094","R"),
("Norway","5002942","385252","R"),
("UAE","8264070","83600","R"),
("India","1210193422","3287263","L"),
("China","1347350000","9640821","R"),
]

def switch_table(memc,table):
  key = "@@" + table
  print "Switching default table to '" + table + "' by issuing GET for '" + key + "'."
  result = memc.get(key)

def insert_country_data(memc):
  banner("Inserting initial data via memcached interface")
  for item in country_data:
    country = item[0]
    population = item[1]
    area = item[2]
    drive_side = item[3]

    key = country
    value = "|".join([population,area,drive_side])
    print "Key = " + key
    print "Value = " + value

    if memc.add(key,value):
      print "Added new key, value pair."
    else:
      print "Updating value for existing key."
      memc.set(key,value)

def query_country_data(memc):
  banner("Retrieving data for all keys (country names)")
  for item in country_data:
    key = item[0]
    result = memc.get(key)
    print "Here is the result retrieved from the database for key " + key + ":"
    print result
    (m_population, m_area, m_drive_side) = result.split("|")
    print "Unpacked population value: " + m_population
    print "Unpacked area value      : " + m_area
    print "Unpacked drive side value: " + m_drive_side

if __name__ == '__main__':

  memc = connect_to_memcached()
  switch_table(memc,"bbb")
  insert_country_data(memc)
  query_country_data(memc)

  sys.exit(0)
```

Sample Python application notes:

- No database authorization is required to run the application, since data manipulation is performed through the `memcached` interface. The only required information is the port number on the local system where the `memcached` daemon listens.
- To make sure the application uses the `multicol` table, the `switch_table()` function is called, which performs a dummy `get` or `set` request using `@@` notation. The `name` value in the request is `bbb`, which is the `multicol` table identifier defined in the `innodb_memcache.containers.name` field.

  A more descriptive `name` value might be used in a real-world application. This example simply illustrates that a table identifier is specified rather than the table name in `get @@...` requests.
- The utility functions used to insert and query data demonstrate how to turn a Python data structure into pipe-separated values for sending data to MySQL with `add` or `set` requests, and how to unpack the pipe-separated values returned by `get` requests. This extra processing is only required when mapping a single `memcached` value to multiple MySQL table columns.

4. Run the sample Python application.

```
shell> python multicol.py
```

If successful, the sample application returns this output:

```
Connected to memcached.
Switching default table to 'bbb' by issuing GET for '@@bbb'.

==============================================
Inserting initial data via memcached interface
==============================================
Key = Canada
Value = 34820000|9984670|R
Added new key, value pair.
Key = USA
Value = 314242000|9826675|R
Added new key, value pair.
Key = Ireland
Value = 6399152|84421|L
Added new key, value pair.
Key = UK
Value = 62262000|243610|L
Added new key, value pair.
Key = Mexico
Value = 113910608|1972550|R
Added new key, value pair.
Key = Denmark
Value = 5543453|43094|R
Added new key, value pair.
Key = Norway
Value = 5002942|385252|R
Added new key, value pair.
Key = UAE
Value = 8264070|83600|R
Added new key, value pair.
Key = India
Value = 1210193422|3287263|L
Added new key, value pair.
Key = China
Value = 1347350000|9640821|R
Added new key, value pair.
```

```
==========================================
Retrieving data for all keys (country names)
==========================================
Here is the result retrieved from the database for key Canada:
34820000|9984670|R
Unpacked population value: 34820000
Unpacked area value      : 9984670
Unpacked drive side value: R
Here is the result retrieved from the database for key USA:
314242000|9826675|R
Unpacked population value: 314242000
Unpacked area value      : 9826675
Unpacked drive side value: R
Here is the result retrieved from the database for key Ireland:
6399152|84421|L
Unpacked population value: 6399152
Unpacked area value      : 84421
Unpacked drive side value: L
Here is the result retrieved from the database for key UK:
62262000|243610|L
Unpacked population value: 62262000
Unpacked area value      : 243610
Unpacked drive side value: L
Here is the result retrieved from the database for key Mexico:
113910608|1972550|R
Unpacked population value: 113910608
Unpacked area value      : 1972550
Unpacked drive side value: R
Here is the result retrieved from the database for key Denmark:
5543453|43094|R
Unpacked population value: 5543453
Unpacked area value      : 43094
Unpacked drive side value: R
Here is the result retrieved from the database for key Norway:
5002942|385252|R
Unpacked population value: 5002942
Unpacked area value      : 385252
Unpacked drive side value: R
Here is the result retrieved from the database for key UAE:
8264070|83600|R
Unpacked population value: 8264070
Unpacked area value      : 83600
Unpacked drive side value: R
Here is the result retrieved from the database for key India:
1210193422|3287263|L
Unpacked population value: 1210193422
Unpacked area value      : 3287263
Unpacked drive side value: L
Here is the result retrieved from the database for key China:
1347350000|9640821|R
Unpacked population value: 1347350000
Unpacked area value      : 9640821
Unpacked drive side value: R
```

5. Query the `innodb_memcache.containers` table to view the record you inserted earlier for the `multicol` table. The first record is the sample entry for the `demo_test` table that is created during the initial `daemon_memcached` plugin setup. The second record is the entry you inserted for the `multicol` table.

```
mysql> SELECT * FROM innodb_memcache.containers\G
*************************** 1. row ***************************
                name: aaa
           db_schema: test
            db_table: demo_test
         key_columns: c1
```

```
        value_columns: c2
                flags: c3
           cas_column: c4
    expire_time_column: c5
unique_idx_name_on_key: PRIMARY
*************************** 2. row ***************************
                 name: bbb
            db_schema: test
              db_table: multicol
          key_columns: country
        value_columns: population,area_sq_km,drive_side
                flags: c3
           cas_column: c4
    expire_time_column: c5
unique_idx_name_on_key: PRIMARY
```

6. Query the `multicol` table to view data inserted by the sample Python application. The data is available for MySQL queries, which demonstrates how the same data can be accessed using SQL or through applications (using the appropriate MySQL Connector or API).

```
mysql> SELECT * FROM test.multicol;
+---------+------------+------------+------------+------+------+------+
| country | population | area_sq_km | drive_side | c3   | c4   | c5   |
+---------+------------+------------+------------+------+------+------+
| Canada  | 34820000   | 9984670    | R          |    0 |   11 |    0 |
| China   | 1347350000 | 9640821    | R          |    0 |   20 |    0 |
| Denmark | 5543453    | 43094      | R          |    0 |   16 |    0 |
| India   | 1210193422 | 3287263    | L          |    0 |   19 |    0 |
| Ireland | 6399152    | 84421      | L          |    0 |   13 |    0 |
| Mexico  | 113910608  | 1972550    | R          |    0 |   15 |    0 |
| Norway  | 5002942    | 385252     | R          |    0 |   17 |    0 |
| UAE     | 8264070    | 83600      | R          |    0 |   18 |    0 |
| UK      | 62262000   | 243610     | L          |    0 |   14 |    0 |
| USA     | 314242000  | 9826675    | R          |    0 |   12 |    0 |
+---------+------------+------------+------------+------+------+------+
```

Note

Always allow sufficient size to hold necessary digits, decimal points, sign characters, leading zeros, and so on when defining the length for columns that are treated as numbers. Too-long values in a string column such as a `VARCHAR` are truncated by removing some characters, which could produce nonsensical numeric values.

7. Optionally, run report-type queries on the `InnoDB` table that stores the `memcached` data.

   You can produce reports through SQL queries, performing calculations and tests across any columns, not just the `country` key column. (Because the following examples use data from only a few countries, the numbers are for illustration purposes only.) The following queries return the average population of countries where people drive on the right, and the average size of countries whose names start with “U”:

```
mysql> SELECT AVG(population) FROM multicol WHERE drive_side = 'R';
+-------------------+
| avg(population)   |
+-------------------+
| 261304724.7142857 |
+-------------------+

mysql> SELECT SUM(area_sq_km) FROM multicol WHERE country LIKE 'U%';
+-----------------+
```

```
| sum(area_sq_km) |
+-----------------+
|        10153885 |
+-----------------+
```

Because the `population` and `area_sq_km` columns store character data rather than strongly typed numeric data, functions such as `AVG()` and `SUM()` work by converting each value to a number first. This approach *does not work* for operators such as `<` or `>`, for example, when comparing character-based values, `9 > 1000`, which is not expected from a clause such as `ORDER BY population DESC`. For the most accurate type treatment, perform queries against views that cast numeric columns to the appropriate types. This technique lets you issue simple `SELECT *` queries from database applications, while ensuring that casting, filtering, and ordering is correct. The following example shows a view that can be queried to find the top three countries in descending order of population, with the results reflecting the latest data in the `multicol` table, and with population and area figures treated as numbers:

```
mysql> CREATE VIEW populous_countries AS
       SELECT
       country,
       cast(population as unsigned integer) population,
       cast(area_sq_km as unsigned integer) area_sq_km,
       drive_side FROM multicol
       ORDER BY CAST(population as unsigned integer) DESC
       LIMIT 3;

mysql> SELECT * FROM populous_countries;
+---------+------------+------------+------------+
| country | population | area_sq_km | drive_side |
+---------+------------+------------+------------+
| China   | 1347350000 |    9640821 | R          |
| India   | 1210193422 |    3287263 | L          |
| USA     |  314242000 |    9826675 | R          |
+---------+------------+------------+------------+

mysql> DESC populous_countries;
+------------+---------------------+------+-----+---------+-------+
| Field      | Type                | Null | Key | Default | Extra |
+------------+---------------------+------+-----+---------+-------+
| country    | varchar(128)        | NO   |     |         |       |
| population | bigint(10) unsigned | YES  |     | NULL    |       |
| area_sq_km | int(9) unsigned     | YES  |     | NULL    |       |
| drive_side | varchar(1)          | YES  |     | NULL    |       |
+------------+---------------------+------+-----+---------+-------+
```

15.19.6.2 Adapting a memcached Application for the InnoDB memcached Plugin

Consider these aspects of MySQL and `InnoDB` tables when adapting existing `memcached` applications to use the `daemon_memcached` plugin:

- If there are key values longer than a few bytes, it may be more efficient to use a numeric auto-increment column as the primary key of the `InnoDB` table, and to create a unique secondary index on the column that contains the `memcached` key values. This is because `InnoDB` performs best for large-scale insertions if primary key values are added in sorted order (as they are with auto-increment values). Primary key values are included in secondary indexes, which takes up unnecessary space if the primary key is a long string value.

- If you store several different classes of information using `memcached`, consider setting up a separate `InnoDB` table for each type of data. Define additional table identifiers in the `innodb_memcache.containers` table, and use the `@@table_id.key` notation to store and retrieve items from different tables. Physically dividing different types of information allows you tune the

characteristics of each table for optimum space utilization, performance, and reliability. For example, you might enable compression for a table that holds blog posts, but not for a table that holds thumbnail images. You might back up one table more frequently than another because it holds critical data. You might create additional secondary indexes on tables that are frequently used to generate reports using SQL.

- Preferably, configure a stable set of table definitions for use with the `daemon_memcached` plugin, and leave the tables in place permanently. Changes to the `innodb_memcache.containers` table take effect the next time the `innodb_memcache.containers` table is queried. Entries in the containers table are processed at startup, and are consulted whenever an unrecognized table identifier (as defined by `containers.name`) is requested using `@@` notation. Thus, new entries are visible as soon as you use the associated table identifier, but changes to existing entries require a server restart before they take effect.
- When you use the default `innodb_only` caching policy, calls to `add()`, `set()`, `incr()`, and so on can succeed but still trigger debugging messages such as `while expecting 'STORED', got unexpected response 'NOT_STORED`. Debug messages occur because new and updated values are sent directly to the `InnoDB` table without being saved in the memory cache, due to the `innodb_only` caching policy.

15.19.6.3 Tuning InnoDB memcached Plugin Performance

Because using `InnoDB` in combination with `memcached` involves writing all data to disk, whether immediately or sometime later, raw performance is expected to be somewhat slower than using `memcached` by itself. When using the `InnoDB memcached` plugin, focus tuning goals for `memcached` operations on achieving better performance than equivalent SQL operations.

Benchmarks suggest that queries and DML operations (inserts, updates, and deletes) that use the `memcached` interface are faster than traditional SQL. DML operations typically see a larger improvements. Therefore, consider adapting write-intensive applications to use the `memcached` interface first. Also consider prioritizing adaptation of write-intensive applications that use fast, lightweight mechanisms that lack reliability.

Adapting SQL Queries

The types of queries that are most suited to simple `GET` requests are those with a single clause or a set of `AND` conditions in the `WHERE` clause:

```
SQL:
SELECT col FROM tbl WHERE key = 'key_value';

memcached:
get key_value

SQL:
SELECT col FROM tbl WHERE col1 = val1 and col2 = val2 and col3 = val3;

memcached:
# Since you must always know these 3 values to look up the key,
# combine them into a unique string and use that as the key
# for all ADD, SET, and GET operations.
key_value = val1 + ":" + val2 + ":" + val3
get key_value

SQL:
SELECT 'key exists!' FROM tbl
  WHERE EXISTS (SELECT col1 FROM tbl WHERE KEY = 'key_value') LIMIT 1;

memcached:
```

```
# Test for existence of key by asking for its value and checking if the call succeeds,
# ignoring the value itself. For existence checking, you typically only store a very
# short value such as "1".
get key_value
```

Using System Memory

For best performance, deploy the `daemon_memcached` plugin on machines that are configured as typical database servers, where the majority of system RAM is devoted to the `InnoDB` buffer pool, through the `innodb_buffer_pool_size` configuration option. For systems with multi-gigabyte buffer pools, consider raising the value of `innodb_buffer_pool_instances` for maximum throughput when most operations involve data that is already cached in memory.

Reducing Redundant I/O

`InnoDB` has a number of settings that let you choose the balance between high reliability, in case of a crash, and the amount of I/O overhead during high write workloads. For example, consider setting the `innodb_doublewrite` to `0` and `innodb_flush_log_at_trx_commit` to `2`. Measure performance with different `innodb_flush_method` settings.

For other ways to reduce or tune I/O for table operations, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

Reducing Transactional Overhead

A default value of 1 for `daemon_memcached_r_batch_size` and `daemon_memcached_w_batch_size` is intended for maximum reliability of results and safety of stored or updated data.

Depending on the type of application, you might increase one or both of these settings to reduce the overhead of frequent commit operations. On a busy system, you might increase `daemon_memcached_r_batch_size`, knowing that changes to data made through SQL may not become visible to `memcached` immediately (that is, until *`N`* more `get` operations are processed). When processing data where every write operation must be reliably stored, leave `daemon_memcached_w_batch_size` set to `1`. Increase the setting when processing large numbers of updates intended only for statistical analysis, where losing the last *`N`* updates in a crash is an acceptable risk.

For example, imagine a system that monitors traffic crossing a busy bridge, recording data for approximately 100,000 vehicles each day. If the application counts different types of vehicles to analyze traffic patterns, changing `daemon_memcached_w_batch_size` from `1` to `100` reduces I/O overhead for commit operations by 99%. In case of an outage, a maximum of 100 records are lost, which may be an acceptable margin of error. If instead the application performed automated toll collection for each car, you would set `daemon_memcached_w_batch_size` to `1` to ensure that each toll record is immediately saved to disk.

Because of the way `InnoDB` organizes `memcached` key values on disk, if you have a large number of keys to create, it may be faster to sort the data items by key value in the application and `add` them in sorted order, rather than create keys in arbitrary order.

The `memslap` command, which is part of the regular `memcached` distribution but not included with the `daemon_memcached` plugin, can be useful for benchmarking different configurations. It can also be used to generate sample key-value pairs to use in your own benchmarks. See `libmemcached` Command-Line Utilities for details.

15.19.6.4 Controlling Transactional Behavior of the InnoDB memcached Plugin

Unlike traditional `memcached`, the `daemon_memcached` plugin allows you to control durability of data values produced through calls to `add`, `set`, `incr`, and so on. By default, data written through the `memcached` interface is stored to disk, and calls to `get` return the most recent value from disk. Although

the default behavior does not offer the best possible raw performance, it is still fast compared to the SQL interface for `InnoDB` tables.

As you gain experience using the `daemon_memcached` plugin, you can consider relaxing durability settings for non-critical classes of data, at the risk of losing some updated values in the event of an outage, or returning data that is slightly out-of-date.

Frequency of Commits

One tradeoff between durability and raw performance is how frequently new and changed data is committed. If data is critical, is should be committed immediately so that it is safe in case of a crash or outage. If data is less critical, such as counters that are reset after a crash or logging data that you can afford to lose, you might prefer higher raw throughput that is available with less frequent commits.

When a `memcached` operation inserts, updates, or deletes data in the underlying `InnoDB` table, the change might be committed to the `InnoDB` table instantly (if `daemon_memcached_w_batch_size=1`) or some time later (if the `daemon_memcached_w_batch_size` value is greater than 1). In either case, the change cannot be rolled back. If you increase the value of `daemon_memcached_w_batch_size` to avoid high I/O overhead during busy times, commits could become infrequent when the workload decreases. As a safety measure, a background thread automatically commits changes made through the `memcached` API at regular intervals. The interval is controlled by the `innodb_api_bk_commit_interval` configuration option, which has a default setting of `5` seconds.

When a `memcached` operation inserts or updates data in the underlying `InnoDB` table, the changed data is immediately visible to other `memcached` requests because the new value remains in the memory cache, even if it is not yet committed on the MySQL side.

Transaction Isolation

When a `memcached` operation such as `get` or `incr` causes a query or DML operation on the underlying `InnoDB` table, you can control whether the operation sees the very latest data written to the table, only data that has been committed, or other variations of transaction isolation level. Use the `innodb_api_trx_level` configuration option to control this feature. The numeric values specified for this option correspond to isolation levels such as `REPEATABLE READ`. See the description of the `innodb_api_trx_level` option for information about other settings.

A strict isolation level ensures that data you retrieve is not rolled back or changed suddenly causing subsequent queries to return different values. However, strict isolation levels require greater locking overhead, which can cause waits. For a NoSQL-style application that does not use long-running transactions, you can typically use the default isolation level or switch to a less strict isolation level.

Disabling Row Locks for memcached DML Operations

The `innodb_api_disable_rowlock` option can be used to disable row locks when `memcached` requests through the `daemon_memcached` plugin cause DML operations. By default, `innodb_api_disable_rowlock` is set to `OFF` which means that `memcached` requests row locks for `get` and `set` operations. When `innodb_api_disable_rowlock` is set to `ON`, `memcached` requests a table lock instead of row locks.

The `innodb_api_disable_rowlock` option is not dynamic. It must be specified at startup on the `mysqld` command line or entered in a MySQL configuration file.

Allowing or Disallowing DDL

By default, you can perform DDL operations such as `ALTER TABLE` on tables used by the `daemon_memcached` plugin. To avoid potential slowdowns when these tables are used for high-throughput

applications, disable DDL operations on these tables by enabling `innodb_api_enable_mdl` at startup. This option is less appropriate when accessing the same tables through both `memcached` and SQL, because it blocks `CREATE INDEX` statements on the tables, which could be important for running reporting queries.

Storing Data on Disk, in Memory, or Both

The `innodb_memcache.cache_policies` table specifies whether to store data written through the `memcached` interface to disk (`innodb_only`, the default); in memory only, as with traditional `memcached` (`cache_only`); or both (`caching`).

With the `caching` setting, if `memcached` cannot find a key in memory, it searches for the value in an `InnoDB` table. Values returned from `get` calls under the `caching` setting could be out-of-date if the values were updated on disk in the `InnoDB` table but are not yet expired from the memory cache.

The caching policy can be set independently for `get`, `set` (including `incr` and `decr`), `delete`, and `flush` operations.

For example, you might allow `get` and `set` operations to query or update a table and the `memcached` memory cache at the same time (using the `caching` setting), while making `delete`, `flush`, or both operate only on the in-memory copy (using the `cache_only` setting). That way, deleting or flushing an item only expires the item from the cache, and the latest value is returned from the `InnoDB` table the next time the item is requested.

```
mysql> SELECT * FROM innodb_memcache.cache_policies;
+--------------+-------------+-------------+---------------+--------------+
| policy_name  | get_policy  | set_policy  | delete_policy | flush_policy |
+--------------+-------------+-------------+---------------+--------------+
| cache_policy | innodb_only | innodb_only | innodb_only   | innodb_only  |
+--------------+-------------+-------------+---------------+--------------+

mysql> UPDATE innodb_memcache.cache_policies SET set_policy = 'caching'
       WHERE policy_name = 'cache_policy';
```

`innodb_memcache.cache_policies` values are only read at startup. After changing values in this table, uninstall and reinstall the `daemon_memcached` plugin to ensure that changes take effect.

```
mysql> UNINSTALL PLUGIN daemon_memcached;

mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

15.19.6.5 Adapting DML Statements to memcached Operations

Benchmarks suggest that the `daemon_memcached` plugin speeds up DML operations (inserts, updates, and deletes) more than it speeds up queries. Therefore, consider focussing initial development efforts on write-intensive applications that are I/O-bound, and look for opportunities to use MySQL with the `daemon_memcached` plugin for new write-intensive applications.

Single-row DML statements are the easiest types of statements to turn into `memcached` operations. `INSERT` becomes `add`, `UPDATE` becomes `set`, `incr` or `decr`, and `DELETE` becomes `delete`. These operations are guaranteed to only affect one row when issued through the `memcached` interface, because the *`key`* is unique within the table.

In the following SQL examples, `t1` refers to the table used for `memcached` operations, based on the configuration in the `innodb_memcache.containers` table. `key` refers to the column listed under `key_columns`, and `val` refers to the column listed under `value_columns`.

```
INSERT INTO t1 (key,val) VALUES (some_key,some_value);
SELECT val FROM t1 WHERE key = some_key;
UPDATE t1 SET val = new_value WHERE key = some_key;
UPDATE t1 SET val = val + x WHERE key = some_key;
DELETE FROM t1 WHERE key = some_key;
```

The following `TRUNCATE TABLE` and `DELETE` statements, which remove all rows from the table, correspond to the `flush_all` operation, where `t1` is configured as the table for `memcached` operations, as in the previous example.

```
TRUNCATE TABLE t1;
DELETE FROM t1;
```

15.19.6.6 Performing DML and DDL Statements on the Underlying InnoDB Table

You can access the underlying `InnoDB` table (which is `test.demo_test` by default) through standard SQL interfaces. However, there are some restrictions:

- When querying a table that is also accessed through the `memcached` interface, remember that `memcached` operations can be configured to be committed periodically rather than after every write operation. This behavior is controlled by the `daemon_memcached_w_batch_size` option. If this option is set to a value greater than `1`, use `READ UNCOMMITTED` queries to find rows that were just inserted.

```
mysql> SET SESSSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

mysql> SELECT * FROM demo_test;
+------+------+------+------+-----------+------+------+------+------+------+------+
| cx   | cy   | c1   | cz   | c2        | ca   | CB   | c3   | cu   | c4   | C5   |
+------+------+------+------+-----------+------+------+------+------+------+------+
| NULL | NULL | a11  | NULL | 123456789 | NULL | NULL |   10 | NULL |    3 | NULL |
+------+------+------+------+-----------+------+------+------+------+------+------+
```

- When modifying a table using SQL that is also accessed through the `memcached` interface, you can configure `memcached` operations to start a new transaction periodically rather than for every read operation. This behavior is controlled by the `daemon_memcached_r_batch_size` option. If this option is set to a value greater than `1`, changes made to the table using SQL are not immediately visible to `memcached` operations.

- The `InnoDB` table is either IS (intention shared) or IX (intention exclusive) locked for all operations in a transaction. If you increase `daemon_memcached_r_batch_size` and `daemon_memcached_w_batch_size` substantially from their default value of `1`, the table is most likely locked between each operation, preventing DDL statements on the table.

15.19.7 The InnoDB memcached Plugin and Replication

Because the `daemon_memcached` plugin supports the MySQL binary log, updates made on a master server through the `memcached` interface can be replicated for backup, balancing intensive read workloads, and high availability. All `memcached` commands are supported with binary logging.

You do not need to set up the `daemon_memcached` plugin on slave servers. The primary advantage of this configuration is increased write throughput on the master. The speed of the replication mechanism is not affected.

The following sections show how to use the binary log capability when using the `daemon_memcached` plugin with MySQL replication. It is assumed that you have completed the setup described in Section 15.19.3, “Setting Up the InnoDB memcached Plugin”.

Enabling the InnoDB memcached Binary Log

1. To use the `daemon_memcached` plugin with the MySQL binary log, enable the `innodb_api_enable_binlog` configuration option on the master server. This option can only be set at server startup. You must also enable the MySQL binary log on the master server using the `--log-bin` option. You can add these options to the MySQL configuration file, or on the `mysqld` command line.

   ```
   mysqld ... --log-bin --innodb_api_enable_binlog=1
   ```

2. Configure the master and slave server, as described in Section 17.1.2, "Setting Up Binary Log File Position Based Replication".

3. Use `mysqldump` to create a master data snapshot, and sync the snapshot to the slave server.

   ```
   master shell> mysqldump --all-databases --lock-all-tables > dbdump.db
   slave shell> mysql < dbdump.db
   ```

4. On the master server, issue `SHOW MASTER STATUS` to obtain the master binary log coordinates.

   ```
   mysql> SHOW MASTER STATUS;
   ```

5. On the slave server, use a `CHANGE MASTER TO` statement to set up a slave server using the master binary log coordinates.

   ```
   mysql> CHANGE MASTER TO
          MASTER_HOST='localhost',
          MASTER_USER='root',
          MASTER_PASSWORD='',
          MASTER_PORT = 13000,
          MASTER_LOG_FILE='0.000001,
          MASTER_LOG_POS=114;
   ```

6. Start the slave.

   ```
   mysql> START SLAVE;
   ```

   If the error log prints output similar to the following, the slave is ready for replication.

   ```
   2013-09-24T13:04:38.639684Z 49 [Note] Slave I/O thread: connected to
   master 'root@localhost:13000', replication started in log '0.000001'
   at position 114
   ```

Testing the InnoDB memcached Replication Configuration

This example demonstrates how to test the `InnoDB memcached` replication configuration using the `memcached` and telnet to insert, update, and delete data. A MySQL client is used to verify results on the master and slave servers.

The example uses the `demo_test` table, which was created by the `innodb_memcached_config.sql` configuration script during the initial setup of the `daemon_memcached` plugin. The `demo_test` table contains a single example record.

1. Use the `set` command to insert a record with a key of `test1`, a flag value of `10`, an expiration value of `0`, a cas value of 1, and a value of `t1`.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set test1 10 0 1
t1
STORED
```

2. On the master server, check that the record was inserted into the `demo_test` table. Assuming the `demo_test` table was not previously modified, there should be two records. The example record with a key of `AA`, and the record you just inserted, with a key of `test1`. The `c1` column maps to the key, the `c2` column to the value, the `c3` column to the flag value, the `c4` column to the cas value, and the `c5` column to the expiration time. The expiration time was set to 0, since it is unused.

```
mysql> SELECT * FROM test.demo_test;
+-------+--------------+------+------+------+
| c1    | c2           | c3   | c4   | c5   |
+-------+--------------+------+------+------+
| AA    | HELLO, HELLO |    8 |    0 |    0 |
| test1 | t1           |   10 |    1 |    0 |
+-------+--------------+------+------+------+
```

3. Check to verify that the same record was replicated to the slave server.

```
mysql> SELECT * FROM test.demo_test;
+-------+--------------+------+------+------+
| c1    | c2           | c3   | c4   | c5   |
+-------+--------------+------+------+------+
| AA    | HELLO, HELLO |    8 |    0 |    0 |
| test1 | t1           |   10 |    1 |    0 |
+-------+--------------+------+------+------+
```

4. Use the `set` command to update the key to a value of `new`.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set test1 10 0 2
new
STORED
```

The update is replicated to the slave server (notice that the `cas` value is also updated).

```
mysql> SELECT * FROM test.demo_test;
+-------+--------------+------+------+------+
| c1    | c2           | c3   | c4   | c5   |
+-------+--------------+------+------+------+
| AA    | HELLO, HELLO |    8 |    0 |    0 |
| test1 | new          |   10 |    2 |    0 |
+-------+--------------+------+------+------+
```

5. Delete the `test1` record using a `delete` command.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
```

```
delete test1
DELETED
```

When the `delete` operation is replicated to the slave, the `test1` record on the slave is also deleted.

```
mysql> SELECT * FROM test.demo_test;
+----+--------------+------+------+------+
| c1 | c2           | c3   | c4   | c5   |
+----+--------------+------+------+------+
| AA | HELLO, HELLO |    8 |    0 |    0 |
+----+--------------+------+------+------+
```

6. Remove all rows from the table using the `flush_all` command.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
flush_all
OK
```

```
mysql> SELECT * FROM test.demo_test;
Empty set (0.00 sec)
```

7. Telnet to the master server and enter two new records.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'
set test2 10 0 4
again
STORED
set test3 10 0 5
again1
STORED
```

8. Confirm that the two records were replicated to the slave server.

```
mysql> SELECT * FROM test.demo_test;
+-------+--------------+------+------+------+
| c1    | c2           | c3   | c4   | c5   |
+-------+--------------+------+------+------+
| test2 | again        |   10 |    4 |    0 |
| test3 | again1       |   10 |    5 |    0 |
+-------+--------------+------+------+------+
```

9. Remove all rows from the table using the `flush_all` command.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
flush_all
OK
```

10. Check to ensure that the `flush_all` operation was replicated on the slave server.

```
mysql> SELECT * FROM test.demo_test;
```

```
Empty set (0.00 sec)
```

InnoDB memcached Binary Log Notes

Binary Log Format:

- Most `memcached` operations are mapped to DML statements (analogous to insert, delete, update). Since there is no actual SQL statement being processed by the MySQL server, all `memcached` commands (except for `flush_all`) use Row-Based Replication (RBR) logging, which is independent of any server `binlog_format` setting.
- The `memcached flush_all` command is mapped to the `TRUNCATE TABLE` command in MySQL 5.7 and earlier. Since DDL commands can only use statement-based logging, the `flush_all` command is replicated by sending a `TRUNCATE TABLE` statement. In MySQL 8.0 and later, `flush_all` is mapped to `DELETE` but is still replicated by sending a `TRUNCATE TABLE` statement.

Transactions:

- The concept of transactions has not typically been part of `memcached` applications. For performance considerations, `daemon_memcached_r_batch_size` and `daemon_memcached_w_batch_size` are used to control the batch size for read and write transactions. These settings do not affect replication. Each SQL operation on the underlying `InnoDB` table is replicated after successful completion.
- The default value of `daemon_memcached_w_batch_size` is `1`, which means that each `memcached` write operation is committed immediately. This default setting incurs a certain amount of performance overhead to avoid inconsistencies in the data that is visible on the master and slave servers. The replicated records are always available immediately on the slave server. If you set `daemon_memcached_w_batch_size` to a value greater than `1`, records inserted or updated through `memcached` are not immediately visible on the master server; to view the records on the master server before they are committed, issue `SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED`.

15.19.8 InnoDB memcached Plugin Internals

InnoDB API for the InnoDB memcached Plugin

The `InnoDB memcached` engine accesses `InnoDB` through `InnoDB` APIs, most of which are directly adopted from embedded `InnoDB`. `InnoDB` API functions are passed to the `InnoDB memcached` engine as callback functions. `InnoDB` API functions access the `InnoDB` tables directly, and are mostly DML operations with the exception of `TRUNCATE TABLE`.

`memcached` commands are implemented through the `InnoDB memcached` API. The following table outlines how `memcached` commands are mapped to DML or DDL operations.

Table 15.28 memcached Commands and Associated DML or DDL Operations

memcached Command	DML or DDL Operations
`get`	a read/fetch command
`set`	a search followed by an `INSERT` or `UPDATE` (depending on whether or not a key exists)
`add`	a search followed by an `INSERT` or `UPDATE`
`replace`	a search followed by an `UPDATE`
`append`	a search followed by an `UPDATE` (appends data to the result before `UPDATE`)

memcached Command	DML or DDL Operations
prepend	a search followed by an UPDATE (prepends data to the result before UPDATE)
incr	a search followed by an UPDATE
decr	a search followed by an UPDATE
delete	a search followed by a DELETE
flush_all	TRUNCATE TABLE (DDL)

InnoDB memcached Plugin Configuration Tables

This section describes configuration tables used by the daemon_memcached plugin. The cache_policies table, config_options table, and containers table are created by the innodb_memcached_config.sql configuration script in the innodb_memcache database.

```
mysql> USE innodb_memcache;
Database changed
mysql> SHOW TABLES;
+---------------------------+
| Tables_in_innodb_memcache |
+---------------------------+
| cache_policies            |
| config_options            |
| containers                |
+---------------------------+
```

cache_policies Table

The cache_policies table defines a cache policy for the InnoDB memcached installation. You can specify individual policies for get, set, delete, and flush operations, within a single cache policy. The default setting for all operations is innodb_only.

- innodb_only: Use InnoDB as the data store.
- cache_only: Use the memcached engine as the data store.
- caching: Use both InnoDB and the memcached engine as data stores. In this case, if memcached cannot find a key in memory, it searches for the value in an InnoDB table.
- disable: Disable caching.

Table 15.29 cache_policies Columns

Column	Description
policy_name	Name of the cache policy. The default cache policy name is cache_policy.
get_policy	The cache policy for get operations. Valid values are innodb_only, cache_only, caching, or disabled. The default setting is innodb_only.
set_policy	The cache policy for set operations. Valid values are innodb_only, cache_only, caching, or disabled. The default setting is innodb_only.
delete_policy	The cache policy for delete operations. Valid values are innodb_only, cache_only, caching, or disabled. The default setting is innodb_only.

Column	Description
flush_policy	The cache policy for flush operations. Valid values are innodb_only, cache_only, caching, or disabled. The default setting is innodb_only.

config_options Table

The config_options table stores memcached-related settings that can be changed at runtime using SQL. Supported configuration options are separator and table_map_delimiter.

Table 15.30 config_options Columns

Column	Description

Name	Name of the memcached-related configuration option. The following configuration options are supported by the config_options table:
	• separator: Used to separate values of a long string into separate values when there are multiple value_columns defined. By default, the separator is a | character. For example, if you define col1, col2 as value columns, and you define | as the separator, you can issue the following memcached command to insert values into col1 and col2, respectively:
	`set keyx 10 0 19`
	`valuecolx|valuecoly`
	valuecol1x is stored in col1 and valuecoly is stored in col2.
	• table_map_delimiter: The character separating the schema name and the table name when you use the @@ notation in a key name to access a key in a specific table. For example, @@t1.some_key and @@t2.some_key have the same key value, but are stored in different tables.

Value	The value assigned to the memcached-related configuration option.

containers Table

The containers table is the most important of the three configuration tables. Each InnoDB table that is used to store memcached values must have an entry in the containers table. The entry provides a mapping between InnoDB table columns and container table columns, which is required for memcached to work with InnoDB tables.

The containers table contains a default entry for the test.demo_test table, which is created by the innodb_memcached_config.sql configuration script. To use the daemon_memcached plugin with your own InnoDB table, you must create an entry in the containers table.

Table 15.31 containers Columns

Column	Description
name	The name given to the container. If an InnoDB table is not requested by name using @@ notation, the daemon_memcached plugin uses the InnoDB table with a containers.name value of default. If there is no such entry, the first entry in the containers table, ordered alphabetically by name (ascending), determines the default InnoDB table.

Column	Description
db_schema	The name of the database where the InnoDB table resides. This is a required value.
db_table	The name of the InnoDB table that stores memcached values. This is a required value.
key_columns	The column in the InnoDB table that contains lookup key values for memcached operations. This is a required value.
value_columns	The InnoDB table columns (one or more) that store memcached data. Multiple columns can be specified using the separator character specified in the innodb_memcached.config_options table. By default, the separator is a pipe character ("|"). To specify multiple columns, separate them with the defined separator character. For example: col1|col2|col3. This is a required value.
flags	The InnoDB table columns that are used as flags (a user-defined numeric value that is stored and retrieved along with the main value) for memcached. A flag value can be used as a column specifier for some operations (such as incr, prepend) if a memcached value is mapped to multiple columns, so that an operation is performed on a specified column. For example, if you have mapped a value_columns to three InnoDB table columns, and only want the increment operation performed on one columns, use the flags column to specify the column. If you do not use the flags column, set a value of 0 to indicate that it is unused.
cas_column	The InnoDB table column that stores compare-and-swap (cas) values. The cas_column value is related to the way memcached hashes requests to different servers and caches data in memory. Because the InnoDB memcached plugin is tightly integrated with a single memcached daemon, and the in-memory caching mechanism is handled by MySQL and the InnoDB buffer pool, this column is rarely needed. If you do not use this column, set a value of 0 to indicate that it is unused.
expire_time_column	The InnoDB table column that stores expiration values. The expire_time_column value is related to the way memcached hashes requests to different servers and caches data in memory. Because the InnoDB memcached plugin is tightly integrated with a single memcached daemon, and the in-memory caching mechanism is handled by MySQL and the InnoDB buffer pool, this column is rarely needed. If you do not use this column, set a value of 0 to indicate that the column is unused. The maximum expire time is defined as INT_MAX32 or 2147483647 seconds (approximately 68 years).
unique_idx_name_on_key	The name of the index on the key column. It must be a unique index. It can be the primary key or a secondary index. Preferably, use the primary key of the InnoDB table. Using the primary key avoids a lookup that is performed when using a secondary index. You cannot make a covering index for memcached lookups; InnoDB returns an error if you try to define a composite secondary index over both the key and value columns.

containers Table Column Constraints

- You must supply a value for db_schema, db_name, key_columns, value_columns and unique_idx_name_on_key. Specify 0 for flags, cas_column, and expire_time_column if they are unused. Failing to do so could cause your setup to fail.

- `key_columns`: The maximum limit for a `memcached` key is 250 characters, which is enforced by `memcached`. The mapped key must be a non-Null `CHAR` or `VARCHAR` type.
- `value_columns`: Must be mapped to a `CHAR`, `VARCHAR`, or `BLOB` column. There is no length restriction and the value can be NULL.
- `cas_column`: The `cas` value is a 64 bit integer. It must be mapped to a `BIGINT` of at least 8 bytes. If you do not use this column, set a value of `0` to indicate that it is unused.
- `expiration_time_column`: Must mapped to an `INTEGER` of at least 4 bytes. Expiration time is defined as a 32-bit integer for Unix time (the number of seconds since January 1, 1970, as a 32-bit value), or the number of seconds starting from the current time. For the latter, the number of seconds may not exceed 60*60*24*30 (the number of seconds in 30 days). If the number sent by a client is larger, the server considers it to be a real Unix time value rather than an offset from the current time. If you do not use this column, set a value of `0` to indicate that it is unused.
- `flags`: Must be mapped to an `INTEGER` of at least 32-bits and can be NULL. If you do not use this column, set a value of `0` to indicate that it is unused.

A pre-check is performed at plugin load time to enforce column constraints. If mismatches are found, the plugin is not loaded.

Multiple Value Column Mapping

- During plugin initialization, when `InnoDB` `memcached` is configured with information defined in the `containers` table, each mapped column defined in `containers.value_columns` is verified against the mapped `InnoDB` table. If multiple `InnoDB` table columns are mapped, there is a check to ensure that each column exists and is the right type.
- At run-time, for `memcached` insert operations, if there are more delimited values than the number of mapped columns, only the number of mapped values are taken. For example, if there are six mapped columns, and seven delimited values are provided, only the first six delimited values are taken. The seventh delimited value is ignored.
- If there are fewer delimited values than mapped columns, unfilled columns are set to NULL. If an unfilled column cannot be set to NULL, insert operations fail.
- If a table has more columns than mapped values, the extra columns do not affect results.

The demo_test Example Table

The `innodb_memcached_config.sql` configuration script creates a `demo_test` table in the `test` database, which can be used to verify `InnoDB` `memcached` plugin installation immediately after setup.

The `innodb_memcached_config.sql` configuration script also creates an entry for the `demo_test` table in the `innodb_memcache.containers` table.

```
mysql> SELECT * FROM innodb_memcache.containers\G
*************************** 1. row ***************************
                  name: aaa
             db_schema: test
              db_table: demo_test
           key_columns: c1
         value_columns: c2
                 flags: c3
            cas_column: c4
    expire_time_column: c5
unique_idx_name_on_key: PRIMARY

mysql> SELECT * FROM test.demo_test;
```

```
+----+------------------+------+------+------+
| c1 | c2               | c3   | c4   | c5   |
+----+------------------+------+------+------+
| AA | HELLO, HELLO     |    8 |    0 |    0 |
+----+------------------+------+------+------+
```

15.19.9 Troubleshooting the InnoDB memcached Plugin

This section describes issues that you may encounter when using the `InnoDB memcached` plugin.

- If you encounter the following error in the MySQL error log, the server might fail to start:

  ```
  failed to set rlimit for open files. Try running as root or requesting
  smaller maxconns value.
  ```

  The error message is from the `memcached` daemon. One solution is to raise the OS limit for the number of open files. The commands for checking and increasing the open file limit varies by operating system. This example shows commands for Linux and OS X:

  ```
  # Linux
  shell> ulimit -n
  1024
  shell> ulimit -n 4096
  shell> ulimit -n
  4096

  # OS X
  shell> ulimit -n
  256
  shell> ulimit -n 4096
  shell> ulimit -n
  4096
  ```

  The other solution is to reduce the number of concurrent connections permitted for the `memcached` daemon. To do so, encode the `-c memcached` option in the `daemon_memcached_option` configuration parameter in the MySQL configuration file. The `-c` option has a default value of 1024.

  ```
  [mysqld]
  ...
  loose-daemon_memcached_option='-c 64'
  ```

- To troubleshoot problems where the `memcached` daemon is unable to store or retrieve `InnoDB` table data, encode the `-vvv memcached` option in the `daemon_memcached_option` configuration parameter in the MySQL configuration file. Examine the MySQL error log for debug output related to `memcached` operations.

  ```
  [mysqld]
  ...
  loose-daemon_memcached_option='-vvv'
  ```

- If columns specified to hold `memcached` values are the wrong data type, such as a numeric type instead of a string type, attempts to store key-value pairs fail with no specific error code or message.

- If the `daemon_memcached` plugin causes MySQL server startup issues, you can temporarily disable the `daemon_memcached` plugin while troubleshooting by adding this line under the `[mysqld]` group in the MySQL configuration file:

  ```
  daemon_memcached=OFF
  ```

For example, if you run the `INSTALL PLUGIN` statement before running the `innodb_memcached_config.sql` configuration script to set up the necessary database and tables, the server might crash and fail to start. The server could also fail to start if you incorrectly configure an entry in the `innodb_memcache.containers` table.

To uninstall the `memcached` plugin for a MySQL instance, issue the following statement:

```
mysql> UNINSTALL PLUGIN daemon_memcached;
```

- If you run more than one instance of MySQL on the same machine with the `daemon_memcached` plugin enabled in each instance, use the `daemon_memcached_option` configuration parameter to specify a unique `memcached` port for each `daemon_memcached` plugin.
- If an SQL statement cannot find the `InnoDB` table or finds no data in the table, but `memcached` API calls retrieve the expected data, you may be missing an entry for the `InnoDB` table in the `innodb_memcache.containers` table, or you may have not switched to the correct `InnoDB` table by issuing a `get` or `set` request using `@@table_id` notation. This problem could also occur if you change an existing entry in the `innodb_memcache.containers` table without restarting the MySQL server afterward. The free-form storage mechanism is flexible enough that your requests to store or retrieve a multi-column value such as `col1|col2|col3` may still work, even if the daemon is using the `test.demo_test` table which stores values in a single column.
- When defining your own `InnoDB` table for use with the `daemon_memcached` plugin, and columns in the table are defined as `NOT NULL`, ensure that values are supplied for the `NOT NULL` columns when inserting a record for the table into the `innodb_memcache.containers` table. If the `INSERT` statement for the `innodb_memcache.containers` record contains fewer delimited values than there are mapped columns, unfilled columns are set to `NULL`. Attempting to insert a `NULL` value into a `NOT NULL` column causes the `INSERT` to fail, which may only become evident after you reinitialize the `daemon_memcached` plugin to apply changes to the `innodb_memcache.containers` table.
- If `cas_column` and `expire_time_column` fields of the `innodb_memcached.containers` table are set to `NULL`, the following error is returned when attempting to load the `memcached` plugin:

  ```
  InnoDB_Memcached: column 6 in the entry for config table 'containers' in
  database 'innodb_memcache' has an invalid NULL value.
  ```

  The `memcached` plugin rejects usage of `NULL` in the `cas_column` and `expire_time_column` columns. Set the value of these columns to `0` when the columns are unused.
- As the length of the `memcached` key and values increase, you might encounter size and length limits.
  - When the key exceeds 250 bytes, `memcached` operations return an error. This is currently a fixed limit within `memcached`.
  - `InnoDB` table limits may be encountered if values exceed 768 bytes in size, 3072 bytes in size, or half of the `innodb_page_size` value. These limits primarily apply if you intend to create an index on a value column to run report-generating queries on that column using SQL. See Section 15.6.1.6, “Limits on InnoDB Tables” for details.
  - The maximum size for the key-value combination is 1 MB.
- If you share configuration files across MySQL servers of different versions, using the latest configuration options for the `daemon_memcached` plugin could cause startup errors on older MySQL versions. To avoid compatibility problems, use the `loose` prefix with option names. For example, use `loose-daemon_memcached_option='-c 64'` instead of `daemon_memcached_option='-c 64'`.

- There is no restriction or check in place to validate character set settings. `memcached` stores and retrieves keys and values in bytes and is therefore not character set sensitive. However, you must ensure that the `memcached` client and the MySQL table use the same character set.
- `memcached` connections are blocked from accessing tables that contain an indexed virtual column. Accessing an indexed virtual column requires a callback to the server, but a `memcached` connection does not have access to the server code.

15.20 InnoDB Troubleshooting

The following general guidelines apply to troubleshooting `InnoDB` problems:

- When an operation fails or you suspect a bug, look at the MySQL server error log (see Section 5.4.2, “The Error Log”). Section B.3.1, “Server Error Message Reference” provides troubleshooting information for some of the common `InnoDB`-specific errors that you may encounter.
- If the failure is related to a deadlock, run with the `innodb_print_all_deadlocks` option enabled so that details about each deadlock are printed to the MySQL server error log. For information about deadlocks, see Section 15.7.5, “Deadlocks in InnoDB”.
- If the issue is related to the `InnoDB` data dictionary, see Section 15.20.3, “Troubleshooting InnoDB Data Dictionary Operations”.
- When troubleshooting, it is usually best to run the MySQL server from the command prompt, rather than through `mysqld_safe` or as a Windows service. You can then see what `mysqld` prints to the console, and so have a better grasp of what is going on. On Windows, start `mysqld` with the `--console` option to direct the output to the console window.
- Enable the `InnoDB` Monitors to obtain information about a problem (see Section 15.16, “InnoDB Monitors”). If the problem is performance-related, or your server appears to be hung, you should enable the standard Monitor to print information about the internal state of `InnoDB`. If the problem is with locks, enable the Lock Monitor. If the problem is with table creation, tablespaces, or data dictionary operations, refer to the InnoDB Information Schema system tables to examine contents of the `InnoDB` internal data dictionary.

  `InnoDB` temporarily enables standard `InnoDB` Monitor output under the following conditions:

  - A long semaphore wait
  - `InnoDB` cannot find free blocks in the buffer pool
  - Over 67% of the buffer pool is occupied by lock heaps or the adaptive hash index
- If you suspect that a table is corrupt, run `CHECK TABLE` on that table.

15.20.1 Troubleshooting InnoDB I/O Problems

The troubleshooting steps for `InnoDB` I/O problems depend on when the problem occurs: during startup of the MySQL server, or during normal operations when a DML or DDL statement fails due to problems at the file system level.

Initialization Problems

If something goes wrong when `InnoDB` attempts to initialize its tablespace or its log files, delete all files created by `InnoDB`: all `ibdata` files and all `ib_logfile` files. If you already created some `InnoDB` tables, also delete any `.ibd` files from the MySQL database directories. Then try the `InnoDB` database

creation again. For easiest troubleshooting, start the MySQL server from a command prompt so that you see what is happening.

Runtime Problems

If `InnoDB` prints an operating system error during a file operation, usually the problem has one of the following solutions:

- Make sure the `InnoDB` data file directory and the `InnoDB` log directory exist.
- Make sure `mysqld` has access rights to create files in those directories.
- Make sure `mysqld` can read the proper `my.cnf` or `my.ini` option file, so that it starts with the options that you specified.
- Make sure the disk is not full and you are not exceeding any disk quota.
- Make sure that the names you specify for subdirectories and data files do not clash.
- Doublecheck the syntax of the `innodb_data_home_dir` and `innodb_data_file_path` values. In particular, any `MAX` value in the `innodb_data_file_path` option is a hard limit, and exceeding that limit causes a fatal error.

15.20.2 Forcing InnoDB Recovery

To investigate database page corruption, you might dump your tables from the database with `SELECT ... INTO OUTFILE`. Usually, most of the data obtained in this way is intact. Serious corruption might cause `SELECT * FROM tbl_name` statements or `InnoDB` background operations to crash or assert, or even cause `InnoDB` roll-forward recovery to crash. In such cases, you can use the `innodb_force_recovery` option to force the `InnoDB` storage engine to start up while preventing background operations from running, so that you can dump your tables. For example, you can add the following line to the `[mysqld]` section of your option file before restarting the server:

```
[mysqld]
innodb_force_recovery = 1
```

For information about using option files, see Section 4.2.2.2, “Using Option Files”.

Warning

Only set `innodb_force_recovery` to a value greater than 0 in an emergency situation, so that you can start `InnoDB` and dump your tables. Before doing so, ensure that you have a backup copy of your database in case you need to recreate it. Values of 4 or greater can permanently corrupt data files. Only use an `innodb_force_recovery` setting of 4 or greater on a production server instance after you have successfully tested the setting on a separate physical copy of your database. When forcing `InnoDB` recovery, you should always start with `innodb_force_recovery=1` and only increase the value incrementally, as necessary.

`innodb_force_recovery` is 0 by default (normal startup without forced recovery). The permissible nonzero values for `innodb_force_recovery` are 1 to 6. A larger value includes the functionality of lesser values. For example, a value of 3 includes all of the functionality of values 1 and 2.

If you are able to dump your tables with an `innodb_force_recovery` value of 3 or less, then you are relatively safe that only some data on corrupt individual pages is lost. A value of 4 or greater is considered

dangerous because data files can be permanently corrupted. A value of 6 is considered drastic because database pages are left in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures.

As a safety measure, `InnoDB` prevents `INSERT`, `UPDATE`, or `DELETE` operations when `innodb_force_recovery` is greater than 0. An `innodb_force_recovery` setting of 4 or greater places `InnoDB` in read-only mode.

- `1` (`SRV_FORCE_IGNORE_CORRUPT`)

  Lets the server run even if it detects a corrupt page. Tries to make `SELECT * FROM tbl_name` jump over corrupt index records and pages, which helps in dumping tables.

- `2` (`SRV_FORCE_NO_BACKGROUND`)

  Prevents the master thread and any purge threads from running. If a crash would occur during the purge operation, this recovery value prevents it.

- `3` (`SRV_FORCE_NO_TRX_UNDO`)

  Does not run transaction rollbacks after crash recovery.

- `4` (`SRV_FORCE_NO_IBUF_MERGE`)

  Prevents insert buffer merge operations. If they would cause a crash, does not do them. Does not calculate table statistics. This value can permanently corrupt data files. After using this value, be prepared to drop and recreate all secondary indexes. Sets `InnoDB` to read-only.

- `5` (`SRV_FORCE_NO_UNDO_LOG_SCAN`)

  Does not look at undo logs when starting the database: `InnoDB` treats even incomplete transactions as committed. This value can permanently corrupt data files. Sets `InnoDB` to read-only.

- `6` (`SRV_FORCE_NO_LOG_REDO`)

  Does not do the redo log roll-forward in connection with recovery. This value can permanently corrupt data files. Leaves database pages in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures. Sets `InnoDB` to read-only.

You can `SELECT` from tables to dump them. With an `innodb_force_recovery` value of 3 or less you can `DROP` or `CREATE` tables. `DROP TABLE` is also supported with an `innodb_force_recovery` value greater than 3. `DROP TABLE` is not permitted with an `innodb_force_recovery` value greater than 4.

If you know that a given table is causing a crash on rollback, you can drop it. If you encounter a runaway rollback caused by a failing mass import or `ALTER TABLE`, you can kill the `mysqld` process and set `innodb_force_recovery` to `3` to bring the database up without the rollback, and then `DROP` the table that is causing the runaway rollback.

If corruption within the table data prevents you from dumping the entire table contents, a query with an `ORDER BY primary_key DESC` clause might be able to dump the portion of the table after the corrupted part.

If a high `innodb_force_recovery` value is required to start `InnoDB`, there may be corrupted data structures that could cause complex queries (queries containing `WHERE`, `ORDER BY`, or other clauses) to fail. In this case, you may only be able to run basic `SELECT * FROM t` queries.

15.20.3 Troubleshooting InnoDB Data Dictionary Operations

Information about table definitions is stored in the InnoDB data dictionary. If you move data files around, dictionary data can become inconsistent.

If a data dictionary corruption or consistency issue prevents you from starting `InnoDB`, see Section 15.20.2, “Forcing InnoDB Recovery” for information about manual recovery.

Cannot Open Datafile

With `innodb_file_per_table` enabled (the default), the following messages may appear at startup if a file-per-table tablespace file (`.ibd` file) is missing:

```
[ERROR] InnoDB: Operating system error number 2 in a file operation.
[ERROR] InnoDB: The error means the system cannot find the path specified.
[ERROR] InnoDB: Cannot open datafile for read-only: './test/t1.ibd' OS error: 71
[Warning] InnoDB: Ignoring tablespace `test/t1` because it could not be opened.
```

To address the these messages, issue `DROP TABLE` statement to remove data about the missing table from the data dictionary.

Restoring Orphan File-Per-Table ibd Files

This procedure describes how to restore orphan file-per-table `.ibd` files to another MySQL instance. You might use this procedure if the system tablespace is lost or unrecoverable and you want to restore `.ibd` file backups on a new MySQL instance.

The procedure is not supported for general tablespace `.ibd` files.

The procedure assumes that you only have `.ibd` file backups, you are recovering to the same version of MySQL that initially created the orphan `.ibd` files, and that `.ibd` file backups are clean. See Section 15.6.1.2, “Moving or Copying InnoDB Tables” for information about creating clean backups.

Tablespace copying limitations outlined in Section 15.6.3.7, “Copying Tablespaces to Another Instance” are applicable to this procedure.

1. On the new MySQL instance, recreate the table in a database of the same name.

```
mysql> CREATE DATABASE sakila;

mysql> USE sakila;

mysql> CREATE TABLE actor (
         actor_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
         first_name VARCHAR(45) NOT NULL,
         last_name VARCHAR(45) NOT NULL,
         last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
         PRIMARY KEY  (actor_id),
         KEY idx_actor_last_name (last_name)
       )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

2. Discard the tablespace of the newly created table.

```
mysql> ALTER TABLE sakila.actor DISCARD TABLESPACE;
```

3. Copy the orphan `.ibd` file from your backup directory to the new database directory.

```
shell> cp /backup_directory/actor.ibd path/to/mysql-5.7/data/sakila/
```

4. Ensure that the `.ibd` file has the necessary file permissions.

5. Import the orphan `.ibd` file. A warning is issued indicating that `InnoDB` will attempt to import the file without schema verification.

```
mysql> ALTER TABLE sakila.actor IMPORT TABLESPACE; SHOW WARNINGS;
Query OK, 0 rows affected, 1 warning (0.15 sec)

Warning | 1810 | InnoDB: IO Read error: (2, No such file or directory)
Error opening './sakila/actor.cfg', will attempt to import
without schema verification
```

6. Query the table to verify that the `.ibd` file was successfully restored.

```
mysql> SELECT COUNT(*) FROM sakila.actor;
+----------+
| count(*) |
+----------+
|      200 |
+----------+
```

15.20.4 InnoDB Error Handling

The following items describe how `InnoDB` performs error handling. `InnoDB` sometimes rolls back only the statement that failed, other times it rolls back the entire transaction.

- If you run out of file space in a tablespace, a MySQL `Table is full` error occurs and `InnoDB` rolls back the SQL statement.

- A transaction deadlock causes `InnoDB` to roll back the entire transaction. Retry the whole transaction when this happens.

  A lock wait timeout causes `InnoDB` to roll back only the single statement that was waiting for the lock and encountered the timeout. (To have the entire transaction roll back, start the server with the `--innodb-rollback-on-timeout` option.) Retry the statement if using the current behavior, or the entire transaction if using `--innodb-rollback-on-timeout`.

  Both deadlocks and lock wait timeouts are normal on busy servers and it is necessary for applications to be aware that they may happen and handle them by retrying. You can make them less likely by doing as little work as possible between the first change to data during a transaction and the commit, so the locks are held for the shortest possible time and for the smallest possible number of rows. Sometimes splitting work between different transactions may be practical and helpful.

  When a transaction rollback occurs due to a deadlock or lock wait timeout, it cancels the effect of the statements within the transaction. But if the start-transaction statement was `START TRANSACTION` or `BEGIN` statement, rollback does not cancel that statement. Further SQL statements become part of the transaction until the occurrence of `COMMIT`, `ROLLBACK`, or some SQL statement that causes an implicit commit.

- A duplicate-key error rolls back the SQL statement, if you have not specified the `IGNORE` option in your statement.

- A `row too long error` rolls back the SQL statement.

- Other errors are mostly detected by the MySQL layer of code (above the `InnoDB` storage engine level), and they roll back the corresponding SQL statement. Locks are not released in a rollback of a single SQL statement.

During implicit rollbacks, as well as during the execution of an explicit `ROLLBACK` SQL statement, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the relevant connection.

Chapter 16 Alternative Storage Engines

Table of Contents

16.1 Setting the Storage Engine 3123
16.2 The MyISAM Storage Engine 3124
16.2.1 MyISAM Startup Options 3126
16.2.2 Space Needed for Keys 3128
16.2.3 MyISAM Table Storage Formats 3128
16.2.4 MyISAM Table Problems 3131
16.3 The MEMORY Storage Engine 3132
16.4 The CSV Storage Engine 3137
16.4.1 Repairing and Checking CSV Tables 3138
16.4.2 CSV Limitations 3139
16.5 The ARCHIVE Storage Engine 3139
16.6 The BLACKHOLE Storage Engine 3141
16.7 The MERGE Storage Engine 3143
16.7.1 MERGE Table Advantages and Disadvantages 3146
16.7.2 MERGE Table Problems 3147
16.8 The FEDERATED Storage Engine 3149
16.8.1 FEDERATED Storage Engine Overview 3149
16.8.2 How to Create FEDERATED Tables 3150
16.8.3 FEDERATED Storage Engine Notes and Tips 3153
16.8.4 FEDERATED Storage Engine Resources 3154
16.9 The EXAMPLE Storage Engine 3154
16.10 Other Storage Engines 3155
16.11 Overview of MySQL Storage Engine Architecture 3155
16.11.1 Pluggable Storage Engine Architecture 3155
16.11.2 The Common Database Server Layer 3156

Storage engines are MySQL components that handle the SQL operations for different table types. `InnoDB` is the default and most general-purpose storage engine, and Oracle recommends using it for tables except for specialized use cases. (The `CREATE TABLE` statement in MySQL 8.0 creates `InnoDB` tables by default.)

MySQL Server uses a pluggable storage engine architecture that enables storage engines to be loaded into and unloaded from a running MySQL server.

To determine which storage engines your server supports, use the `SHOW ENGINES` statement. The value in the `Support` column indicates whether an engine can be used. A value of `YES`, `NO`, or `DEFAULT` indicates that an engine is available, not available, or available and currently set as the default storage engine.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: InnoDB
```

```
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 3. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
...
```

This chapter covers use cases for special-purpose MySQL storage engines. It does not cover the default `InnoDB` storage engine or the `NDB` storage engine which are covered in Chapter 15, *The InnoDB Storage Engine* and Chapter 22, *MySQL NDB Cluster 8.0*. For advanced users, it also contains a description of the pluggable storage engine architecture (see Section 16.11, “Overview of MySQL Storage Engine Architecture”).

For information about features offered in commercial MySQL Server binaries, see *MySQL Editions*, on the MySQL website. The storage engines available might depend on which edition of MySQL you are using.

For answers to commonly asked questions about MySQL storage engines, see Section A.2, “MySQL 8.0 FAQ: Storage Engines”.

MySQL 8.0 Supported Storage Engines

- `InnoDB`: The default storage engine in MySQL 8.0. `InnoDB` is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. `InnoDB` row-level locking (without escalation to coarser granularity locks) and Oracle-style consistent nonlocking reads increase multi-user concurrency and performance. `InnoDB` stores user data in clustered indexes to reduce I/O for common queries based on primary keys. To maintain data integrity, `InnoDB` also supports `FOREIGN KEY` referential-integrity constraints. For more information about `InnoDB`, see Chapter 15, *The InnoDB Storage Engine*.

- `MyISAM`: These tables have a small footprint. Table-level locking limits the performance in read/write workloads, so it is often used in read-only or read-mostly workloads in Web and data warehousing configurations.

- `Memory`: Stores all data in RAM, for fast access in environments that require quick lookups of non-critical data. This engine was formerly known as the `HEAP` engine. Its use cases are decreasing; `InnoDB` with its buffer pool memory area provides a general-purpose and durable way to keep most or all data in memory, and `NDBCLUSTER` provides fast key-value lookups for huge distributed data sets.

- `CSV`: Its tables are really text files with comma-separated values. CSV tables let you import or dump data in CSV format, to exchange data with scripts and applications that read and write that same format.

Because CSV tables are not indexed, you typically keep the data in `InnoDB` tables during normal operation, and only use CSV tables during the import or export stage.

- `Archive`: These compact, unindexed tables are intended for storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.
- `Blackhole`: The Blackhole storage engine accepts but does not store data, similar to the Unix `/dev/null` device. Queries always return an empty set. These tables can be used in replication configurations where DML statements are sent to slave servers, but the master server does not keep its own copy of the data.
- `NDB` (also known as `NDBCLUSTER`): This clustered database engine is particularly suited for applications that require the highest possible degree of uptime and availability.
- `Merge`: Enables a MySQL DBA or developer to logically group a series of identical `MyISAM` tables and reference them as one object. Good for VLDB environments such as data warehousing.
- `Federated`: Offers the ability to link separate MySQL servers to create one logical database from many physical servers. Very good for distributed or data mart environments.
- `Example`: This engine serves as an example in the MySQL source code that illustrates how to begin writing new storage engines. It is primarily of interest to developers. The storage engine is a “stub” that does nothing. You can create tables with this engine, but no data can be stored in them or retrieved from them.

You are not restricted to using the same storage engine for an entire server or schema. You can specify the storage engine for any table. For example, an application might use mostly `InnoDB` tables, with one `CSV` table for exporting data to a spreadsheet and a few `MEMORY` tables for temporary workspaces.

Choosing a Storage Engine

The various storage engines provided with MySQL are designed with different use cases in mind. The following table provides an overview of some storage engines provided with MySQL, with clarifying notes following the table.

Table 16.1 Storage Engines Feature Summary

Feature	MyISAM	Memory	InnoDB	Archive	NDB
B-tree indexes	Yes	Yes	Yes	No	No
Backup/point-in-time recovery (note 1)	Yes	Yes	Yes	Yes	Yes
Cluster database support	No	No	No	No	Yes
Clustered indexes	No	No	Yes	No	No
Compressed data	Yes (note 2)	No	Yes	Yes	No
Data caches	No	N/A	Yes	No	Yes

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Encrypted data	Yes (note 3)	Yes (note 3)	Yes (note 4)	Yes (note 3)	Yes (note 3)
Foreign key support	No	No	Yes	No	Yes (note 5)
Full-text search indexes	Yes	No	Yes (note 6)	No	No
Geospatial data type support	Yes	No	Yes	Yes	Yes
Geospatial indexing support	Yes	No	Yes (note 7)	No	No
Hash indexes	No	Yes	No (note 8)	No	Yes
Index caches	Yes	N/A	Yes	No	Yes
Locking granularity	Table	Table	Row	Row	Row
MVCC	No	No	Yes	No	No
Replication support (note 1)	Yes	Limited (note 9)	Yes	Yes	Yes
Storage limits	256TB	RAM	64TB	None	384EB
T-tree indexes	No	No	No	No	Yes
Transactions	No	No	Yes	No	Yes
Update statistics for data dictionary	Yes	Yes	Yes	Yes	Yes

Notes:

1. Implemented in the server, rather than in the storage engine.

2. Compressed MyISAM tables are supported only when using the compressed row format. Tables using the compressed row format with MyISAM are read only.

3. Implemented in the server via encryption functions.

4. Implemented in the server via encryption functions; In MySQL 5.7 and later, data-at-rest tablespace encryption is supported.

5. Support for foreign keys is available in MySQL Cluster NDB 7.3 and later.

6. InnoDB support for FULLTEXT indexes is available in MySQL 5.6 and later.

7. InnoDB support for geospatial indexing is available in MySQL 5.7 and later.

8. InnoDB utilizes hash indexes internally for its Adaptive Hash Index feature.

9. See the discussion later in this section.

16.1 Setting the Storage Engine

When you create a new table, you can specify which storage engine to use by adding an `ENGINE` table option to the `CREATE TABLE` statement:

```
-- ENGINE=INNODB not needed unless you have set a different
-- default storage engine.
CREATE TABLE t1 (i INT) ENGINE = INNODB;
-- Simple table definitions can be switched from one to another.
CREATE TABLE t2 (i INT) ENGINE = CSV;
CREATE TABLE t3 (i INT) ENGINE = MEMORY;
```

When you omit the `ENGINE` option, the default storage engine is used. The default engine is `InnoDB` in MySQL 8.0. You can specify the default engine by using the `--default-storage-engine` server startup option, or by setting the `default-storage-engine` option in the `my.cnf` configuration file.

You can set the default storage engine for the current session by setting the `default_storage_engine` variable:

```
SET default_storage_engine=NDBCLUSTER;
```

The storage engine for `TEMPORARY` tables created with `CREATE TEMPORARY TABLE` can be set separately from the engine for permanent tables by setting the `default_tmp_storage_engine`, either at startup or at runtime.

To convert a table from one storage engine to another, use an `ALTER TABLE` statement that indicates the new engine:

```
ALTER TABLE t ENGINE = InnoDB;
```

See Section 13.1.20, “CREATE TABLE Syntax”, and Section 13.1.9, “ALTER TABLE Syntax”.

If you try to use a storage engine that is not compiled in or that is compiled in but deactivated, MySQL instead creates a table using the default storage engine. For example, in a replication setup, perhaps your master server uses `InnoDB` tables for maximum safety, but the slave servers use other storage engines for speed at the expense of durability or concurrency.

By default, a warning is generated whenever `CREATE TABLE` or `ALTER TABLE` cannot use the default storage engine. To prevent confusing, unintended behavior if the desired engine is unavailable, enable the `NO_ENGINE_SUBSTITUTION` SQL mode. If the desired engine is unavailable, this setting produces an error instead of a warning, and the table is not created or altered. See Section 5.1.11, “Server SQL Modes”.

MySQL may store a table's index and data in one or more other files, depending on the storage engine. Table and column definitions are stored in the MySQL data dictionary. Individual storage engines create any additional files required for the tables that they manage. If a table name contains special characters,

the names for the table files contain encoded versions of those characters as described in Section 9.2.4, “Mapping of Identifiers to File Names”.

16.2 The MyISAM Storage Engine

`MyISAM` is based on the older (and no longer available) `ISAM` storage engine but has many useful extensions.

Table 16.2 MyISAM Storage Engine Features

Feature	Support
B-tree indexes	Yes
Backup/point-in-time recovery (Implemented in the server, rather than in the storage engine.)	Yes
Cluster database support	No
Clustered indexes	No
Compressed data	Yes (Compressed MyISAM tables are supported only when using the compressed row format. Tables using the compressed row format with MyISAM are read only.)
Data caches	No
Encrypted data	Yes (Implemented in the server via encryption functions.)
Foreign key support	No
Full-text search indexes	Yes
Geospatial data type support	Yes
Geospatial indexing support	Yes
Hash indexes	No
Index caches	Yes
Locking granularity	Table
MVCC	No
Replication support (Implemented in the server, rather than in the storage engine.)	Yes
Storage limits	256TB
T-tree indexes	No
Transactions	No
Update statistics for data dictionary	Yes

Each `MyISAM` table is stored on disk in two files. The files have names that begin with the table name and have an extension to indicate the file type. The data file has an `.MYD` (`MYData`) extension. The index file has an `.MYI` (`MYIndex`) extension. The table definition is stored in the MySQL data dictionary.

To specify explicitly that you want a `MyISAM` table, indicate that with an `ENGINE` table option:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

In MySQL 8.0, it is normally necessary to use `ENGINE` to specify the `MyISAM` storage engine because `InnoDB` is the default engine.

You can check or repair `MyISAM` tables with the `mysqlcheck` client or `myisamchk` utility. You can also compress `MyISAM` tables with `myisampack` to take up much less space. See Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”, Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”, and Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”.

In MySQL 8.0, the `MyISAM` storage engine provides no partitioning support. *Partitioned `MyISAM` tables created in previous versions of MySQL cannot be used in MySQL 8.0*. For more information, see Section 23.6.2, “Partitioning Limitations Relating to Storage Engines”. For help with upgrading such tables so that they can be used in MySQL 8.0, see Section 2.11.4, “Changes in MySQL 8.0”.

`MyISAM` tables have the following characteristics:

- All data values are stored with the low byte first. This makes the data machine and operating system independent. The only requirements for binary portability are that the machine uses two's-complement signed integers and IEEE floating-point format. These requirements are widely used among mainstream machines. Binary compatibility might not be applicable to embedded systems, which sometimes have peculiar processors.

  There is no significant speed penalty for storing data low byte first; the bytes in a table row normally are unaligned and it takes little more processing to read an unaligned byte in order than in reverse order. Also, the code in the server that fetches column values is not time critical compared to other code.

- All numeric key values are stored with the high byte first to permit better index compression.

- Large files (up to 63-bit file length) are supported on file systems and operating systems that support large files.

- There is a limit of $(2^{32})^2$ (1.844E+19) rows in a `MyISAM` table.

- The maximum number of indexes per `MyISAM` table is 64.

  The maximum number of columns per index is 16.

- The maximum key length is 1000 bytes. This can also be changed by changing the source and recompiling. For the case of a key longer than 250 bytes, a larger key block size than the default of 1024 bytes is used.

- When rows are inserted in sorted order (as when you are using an `AUTO_INCREMENT` column), the index tree is split so that the high node only contains one key. This improves space utilization in the index tree.

- Internal handling of one `AUTO_INCREMENT` column per table is supported. `MyISAM` automatically updates this column for `INSERT` and `UPDATE` operations. This makes `AUTO_INCREMENT` columns faster (at least 10%). Values at the top of the sequence are not reused after being deleted. (When an `AUTO_INCREMENT` column is defined as the last column of a multiple-column index, reuse of values deleted from the top of a sequence does occur.) The `AUTO_INCREMENT` value can be reset with `ALTER TABLE` or `myisamchk`.

- Dynamic-sized rows are much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.

- `MyISAM` supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. A free block

can occur as a result of deleting rows or an update of a dynamic length row with more data than its current contents. When all free blocks are used up (filled in), future inserts become concurrent again. See Section 8.11.3, “Concurrent Inserts”.

- You can put the data file and index file in different directories on different physical devices to get more speed with the `DATA DIRECTORY` and `INDEX DIRECTORY` table options to `CREATE TABLE`. See Section 13.1.20, “CREATE TABLE Syntax”.
- `BLOB` and `TEXT` columns can be indexed.
- `NULL` values are permitted in indexed columns. This takes 0 to 1 bytes per key.
- Each character column can have a different character set. See Chapter 10, *Character Sets, Collations, Unicode*.
- There is a flag in the `MyISAM` index file that indicates whether the table was closed correctly. If `mysqld` is started with the `myisam_recover_options` system variable set, `MyISAM` tables are automatically checked when opened, and are repaired if the table wasn't closed properly.
- `myisamchk` marks tables as checked if you run it with the `--update-state` option. `myisamchk --fast` checks only those tables that don't have this mark.
- `myisamchk --analyze` stores statistics for portions of keys, as well as for entire keys.
- `myisampack` can pack `BLOB` and `VARCHAR` columns.

`MyISAM` also supports the following features:

- Support for a true `VARCHAR` type; a `VARCHAR` column starts with a length stored in one or two bytes.
- Tables with `VARCHAR` columns may have fixed or dynamic row length.
- The sum of the lengths of the `VARCHAR` and `CHAR` columns in a table may be up to 64KB.
- Arbitrary length `UNIQUE` constraints.

Additional Resources

- A forum dedicated to the `MyISAM` storage engine is available at https://forums.mysql.com/list.php?21.

16.2.1 MyISAM Startup Options

The following options to `mysqld` can be used to change the behavior of `MyISAM` tables. For additional information, see Section 5.1.7, “Server Command Options”.

Table 16.3 MyISAM Option and Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
bulk_insert_buffer_size	Yes	Yes	Yes		Both	Yes
concurrent_insert	Yes	Yes	Yes		Global	Yes
delay_key_write	Yes	Yes	Yes		Global	Yes
have_rtree_keys			Yes		Global	No
key_buffer_size	Yes	Yes	Yes		Global	Yes
log-isam	Yes	Yes				
myisam-block-size	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
myisam_data_pointer_size	Yes	Yes	Yes		Global	Ye
myisam_max_sort_file_size	Yes	Yes	Yes		Global	Ye
myisam_mmap_size	Yes	Yes	Yes		Global	No
myisam_recover_options	Yes	Yes	Yes		Global	No
myisam_repair_threads	Yes	Yes	Yes		Both	Ye
myisam_sort_buffer_size	Yes	Yes	Yes		Both	Ye
myisam_stats_method	Yes	Yes	Yes		Both	Ye
myisam_use_mmap	Yes	Yes	Yes		Global	Ye
skip-concurrent-insert	Yes	Yes				
- *Variable*: concurrent_insert						
tmp_table_size	Yes	Yes	Yes		Both	Ye

The following system variables affect the behavior of `MyISAM` tables. For additional information, see Section 5.1.8, “Server System Variables”.

- `bulk_insert_buffer_size`

  The size of the tree cache used in bulk insert optimization.

  Note

  This is a limit *per thread*!

- `delay_key_write=ALL`

  Don't flush key buffers between writes for any `MyISAM` table.

  Note

  If you do this, you should not access `MyISAM` tables from another program (such as from another MySQL server or with `myisamchk`) when the tables are in use. Doing so risks index corruption. Using `--external-locking` does not eliminate this risk.

- `myisam_max_sort_file_size`

  The maximum size of the temporary file that MySQL is permitted to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

- `myisam_recover_options=mode`

  Set the mode for automatic recovery of crashed `MyISAM` tables.

- `myisam_sort_buffer_size`

  Set the size of the buffer used when recovering tables.

Automatic recovery is activated if you start `mysqld` with the `myisam_recover_options` system variable set. In this case, when the server opens a `MyISAM` table, it checks whether the table is marked as crashed

or whether the open count variable for the table is not 0 and you are running the server with external locking disabled. If either of these conditions is true, the following happens:

- The server checks the table for errors.
- If the server finds an error, it tries to do a fast table repair (with sorting and without re-creating the data file).
- If the repair fails because of an error in the data file (for example, a duplicate-key error), the server tries again, this time re-creating the data file.
- If the repair still fails, the server tries once more with the old repair option method (write row by row without sorting). This method should be able to repair any type of error and has low disk space requirements.

If the recovery wouldn't be able to recover all rows from previously completed statements and you didn't specify `FORCE` in the value of the `myisam_recover_options` system variable, automatic repair aborts with an error message in the error log:

```
Error: Couldn't repair table: test.g00pages
```

If you specify `FORCE`, a warning like this is written instead:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

If the automatic recovery value includes `BACKUP`, the recovery process creates files with names of the form *`tbl_name-datetime`*`.BAK`. You should have a `cron` script that automatically moves these files from the database directories to backup media.

16.2.2 Space Needed for Keys

`MyISAM` tables use B-tree indexes. You can roughly calculate the size for the index file as `(key_length+4)/0.67`, summed over all keys. This is for the worst case when all keys are inserted in sorted order and the table doesn't have any compressed keys.

String indexes are space compressed. If the first index part is a string, it is also prefix compressed. Space compression makes the index file smaller than the worst-case figure if a string column has a lot of trailing space or is a `VARCHAR` column that is not always used to the full length. Prefix compression is used on keys that start with a string. Prefix compression helps if there are many strings with an identical prefix.

In `MyISAM` tables, you can also prefix compress numbers by specifying the `PACK_KEYS=1` table option when you create the table. Numbers are stored with the high byte first, so this helps when you have many integer keys that have an identical prefix.

16.2.3 MyISAM Table Storage Formats

`MyISAM` supports three different storage formats. Two of them, fixed and dynamic format, are chosen automatically depending on the type of columns you are using. The third, compressed format, can be created only with the `myisampack` utility (see Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”).

When you use `CREATE TABLE` or `ALTER TABLE` for a table that has no `BLOB` or `TEXT` columns, you can force the table format to `FIXED` or `DYNAMIC` with the `ROW_FORMAT` table option.

See Section 13.1.20, “CREATE TABLE Syntax”, for information about `ROW_FORMAT`.

You can decompress (unpack) compressed `MyISAM` tables using `myisamchk --unpack`; see Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”, for more information.

16.2.3.1 Static (Fixed-Length) Table Characteristics

Static format is the default for `MyISAM` tables. It is used when the table contains no variable-length columns (`VARCHAR`, `VARBINARY`, `BLOB`, or `TEXT`). Each row is stored using a fixed number of bytes.

Of the three `MyISAM` storage formats, static format is the simplest and most secure (least subject to corruption). It is also the fastest of the on-disk formats due to the ease with which rows in the data file can be found on disk: To look up a row based on a row number in the index, multiply the row number by the row length to calculate the row position. Also, when scanning a table, it is very easy to read a constant number of rows with each disk read operation.

The security is evidenced if your computer crashes while the MySQL server is writing to a fixed-format `MyISAM` file. In this case, `myisamchk` can easily determine where each row starts and ends, so it can usually reclaim all rows except the partially written one. `MyISAM` table indexes can always be reconstructed based on the data rows.

Note

Fixed-length row format is only available for tables without `BLOB` or `TEXT` columns. Creating a table with these columns with an explicit `ROW_FORMAT` clause will not raise an error or warning; the format specification will be ignored.

Static-format tables have these characteristics:

- `CHAR` and `VARCHAR` columns are space-padded to the specified column width, although the column type is not altered. `BINARY` and `VARBINARY` columns are padded with `0x00` bytes to the column width.
- `NULL` columns require additional space in the row to record whether their values are `NULL`. Each `NULL` column takes one bit extra, rounded up to the nearest byte.
- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because rows are located in fixed positions.
- Reorganization is unnecessary unless you delete a huge number of rows and want to return free disk space to the operating system. To do this, use `OPTIMIZE TABLE` or `myisamchk -r`.
- Usually require more disk space than dynamic-format tables.
- The expected row length in bytes for static-sized rows is calculated using the following expression:

```
row length = 1
             + (sum of column lengths)
             + (number of NULL columns + delete_flag + 7)/8
             + (number of variable-length columns)
```

`delete_flag` is 1 for tables with static row format. Static tables use a bit in the row record for a flag that indicates whether the row has been deleted. *`delete_flag`* is 0 for dynamic tables because the flag is stored in the dynamic row header.

16.2.3.2 Dynamic Table Characteristics

Dynamic storage format is used if a MyISAM table contains any variable-length columns (VARCHAR, VARBINARY, BLOB, or TEXT), or if the table was created with the ROW_FORMAT=DYNAMIC table option.

Dynamic format is a little more complex than static format because each row has a header that indicates how long it is. A row can become fragmented (stored in noncontiguous pieces) when it is made longer as a result of an update.

You can use OPTIMIZE TABLE or myisamchk -r to defragment a table. If you have fixed-length columns that you access or change frequently in a table that also contains some variable-length columns, it might be a good idea to move the variable-length columns to other tables just to avoid fragmentation.

Dynamic-format tables have these characteristics:

- All string columns are dynamic except those with a length less than four.
- Each row is preceded by a bitmap that indicates which columns contain the empty string (for string columns) or zero (for numeric columns). This does not include columns that contain NULL values. If a string column has a length of zero after trailing space removal, or a numeric column has a value of zero, it is marked in the bitmap and not saved to disk. Nonempty strings are saved as a length byte plus the string contents.
- NULL columns require additional space in the row to record whether their values are NULL. Each NULL column takes one bit extra, rounded up to the nearest byte.
- Much less disk space usually is required than for fixed-length tables.
- Each row uses only as much space as is required. However, if a row becomes larger, it is split into as many pieces as are required, resulting in row fragmentation. For example, if you update a row with information that extends the row length, the row becomes fragmented. In this case, you may have to run OPTIMIZE TABLE or myisamchk -r from time to time to improve performance. Use myisamchk -ei to obtain table statistics.
- More difficult than static-format tables to reconstruct after a crash, because rows may be fragmented into many pieces and links (fragments) may be missing.
- The expected row length for dynamic-sized rows is calculated using the following expression:

```
3
+ (number of columns + 7) / 8
+ (number of char columns)
+ (packed size of numeric columns)
+ (length of strings)
+ (number of NULL columns + 7) / 8
```

  There is a penalty of 6 bytes for each link. A dynamic row is linked whenever an update causes an enlargement of the row. Each new link is at least 20 bytes, so the next enlargement probably goes in the same link. If not, another link is created. You can find the number of links using myisamchk -ed. All links may be removed with OPTIMIZE TABLE or myisamchk -r.

16.2.3.3 Compressed Table Characteristics

Compressed storage format is a read-only format that is generated with the myisampack tool. Compressed tables can be uncompressed with myisamchk.

Compressed tables have the following characteristics:

- Compressed tables take very little disk space. This minimizes disk usage, which is helpful when using slow disks (such as CD-ROMs).

- Each row is compressed separately, so there is very little access overhead. The header for a row takes up one to three bytes depending on the biggest row in the table. Each column is compressed differently. There is usually a different Huffman tree for each column. Some of the compression types are:
  - Suffix space compression.
  - Prefix space compression.
  - Numbers with a value of zero are stored using one bit.
  - If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from `-128` to `127`.
  - If a column has only a small set of possible values, the data type is converted to `ENUM`.
  - A column may use any combination of the preceding compression types.
- Can be used for fixed-length or dynamic-length rows.

Note

While a compressed table is read only, and you cannot therefore update or add rows in the table, DDL (Data Definition Language) operations are still valid. For example, you may still use `DROP` to drop the table, and `TRUNCATE TABLE` to empty the table.

16.2.4 MyISAM Table Problems

The file format that MySQL uses to store data has been extensively tested, but there are always circumstances that may cause database tables to become corrupted. The following discussion describes how this can happen and how to handle it.

16.2.4.1 Corrupted MyISAM Tables

Even though the `MyISAM` table format is very reliable (all changes to a table made by an SQL statement are written before the statement returns), you can still get corrupted tables if any of the following events occur:

- The `mysqld` process is killed in the middle of a write.
- An unexpected computer shutdown occurs (for example, the computer is turned off).
- Hardware failures.
- You are using an external program (such as `myisamchk`) to modify a table that is being modified by the server at the same time.
- A software bug in the MySQL or `MyISAM` code.

Typical symptoms of a corrupt table are:

- You get the following error while selecting data from the table:

```
Incorrect key file for table: '...'. Try to repair it
```

- Queries don't find rows in the table or return incomplete results.

You can check the health of a MyISAM table using the CHECK TABLE statement, and repair a corrupted MyISAM table with REPAIR TABLE. When mysqld is not running, you can also check or repair a table with the myisamchk command. See Section 13.7.3.2, “CHECK TABLE Syntax”, Section 13.7.3.5, “REPAIR TABLE Syntax”, and Section 4.6.4, “myisamchk — MyISAM Table-Maintenance Utility”.

If your tables become corrupted frequently, you should try to determine why this is happening. The most important thing to know is whether the table became corrupted as a result of a server crash. You can verify this easily by looking for a recent restarted mysqld message in the error log. If there is such a message, it is likely that table corruption is a result of the server dying. Otherwise, corruption may have occurred during normal operation. This is a bug. You should try to create a reproducible test case that demonstrates the problem. See Section B.4.3.3, “What to Do If MySQL Keeps Crashing”, and Section 29.5, “Debugging and Porting MySQL”.

16.2.4.2 Problems from Tables Not Being Closed Properly

Each MyISAM index file (.MYI file) has a counter in the header that can be used to check whether a table has been closed properly. If you get the following warning from CHECK TABLE or myisamchk, it means that this counter has gone out of sync:

```
clients are using or haven't closed the table properly
```

This warning doesn't necessarily mean that the table is corrupted, but you should at least check the table.

The counter works as follows:

- The first time a table is updated in MySQL, a counter in the header of the index files is incremented.
- The counter is not changed during further updates.
- When the last instance of a table is closed (because a FLUSH TABLES operation was performed or because there is no room in the table cache), the counter is decremented if the table has been updated at any point.
- When you repair the table or check the table and it is found to be okay, the counter is reset to zero.
- To avoid problems with interaction with other processes that might check the table, the counter is not decremented on close if it was zero.

In other words, the counter can become incorrect only under these conditions:

- A MyISAM table is copied without first issuing LOCK TABLES and FLUSH TABLES.
- MySQL has crashed between an update and the final close. (The table may still be okay because MySQL always issues writes for everything between each statement.)
- A table was modified by myisamchk --recover or myisamchk --update-state at the same time that it was in use by mysqld.
- Multiple mysqld servers are using the table and one server performed a REPAIR TABLE or CHECK TABLE on the table while it was in use by another server. In this setup, it is safe to use CHECK TABLE, although you might get the warning from other servers. However, REPAIR TABLE should be avoided because when one server replaces the data file with a new one, this is not known to the other servers.

  In general, it is a bad idea to share a data directory among multiple servers. See Section 5.8, “Running Multiple MySQL Instances on One Machine”, for additional discussion.

16.3 The MEMORY Storage Engine

The MEMORY storage engine (formerly known as HEAP) creates special-purpose tables with contents that are stored in memory. Because the data is vulnerable to crashes, hardware issues, or power outages, only use these tables as temporary work areas or read-only caches for data pulled from other tables.

Table 16.4 MEMORY Storage Engine Features

Feature	Support
B-tree indexes	Yes
Backup/point-in-time recovery (Implemented in the server, rather than in the storage engine.)	Yes
Cluster database support	No
Clustered indexes	No
Compressed data	No
Data caches	N/A
Encrypted data	Yes (Implemented in the server via encryption functions.)
Foreign key support	No
Full-text search indexes	No
Geospatial data type support	No
Geospatial indexing support	No
Hash indexes	Yes
Index caches	N/A
Locking granularity	Table
MVCC	No
Replication support (Implemented in the server, rather than in the storage engine.)	Limited (See the discussion later in this section.)
Storage limits	RAM
T-tree indexes	No
Transactions	No
Update statistics for data dictionary	Yes

- When to Use MEMORY or NDB Cluster
- Partitioning
- Performance Characteristics
- Characteristics of MEMORY Tables
- DDL Operations for MEMORY Tables
- Indexes
- User-Created and Temporary Tables
- Loading Data
- MEMORY Tables and Replication

- Managing Memory Use
- Additional Resources

When to Use MEMORY or NDB Cluster

Developers looking to deploy applications that use the `MEMORY` storage engine for important, highly available, or frequently updated data should consider whether NDB Cluster is a better choice. A typical use case for the `MEMORY` engine involves these characteristics:

- Operations involving transient, non-critical data such as session management or caching. When the MySQL server halts or restarts, the data in `MEMORY` tables is lost.
- In-memory storage for fast access and low latency. Data volume can fit entirely in memory without causing the operating system to swap out virtual memory pages.
- A read-only or read-mostly data access pattern (limited updates).

NDB Cluster offers the same features as the `MEMORY` engine with higher performance levels, and provides additional features not available with `MEMORY`:

- Row-level locking and multiple-thread operation for low contention between clients.
- Scalability even with statement mixes that include writes.
- Optional disk-backed operation for data durability.
- Shared-nothing architecture and multiple-host operation with no single point of failure, enabling 99.999% availability.
- Automatic data distribution across nodes; application developers need not craft custom sharding or partitioning solutions.
- Support for variable-length data types (including `BLOB` and `TEXT`) not supported by `MEMORY`.

Partitioning

`MEMORY` tables cannot be partitioned.

Performance Characteristics

`MEMORY` performance is constrained by contention resulting from single-thread execution and table lock overhead when processing updates. This limits scalability when load increases, particularly for statement mixes that include writes.

Despite the in-memory processing for `MEMORY` tables, they are not necessarily faster than `InnoDB` tables on a busy server, for general-purpose queries, or under a read/write workload. In particular, the table locking involved with performing updates can slow down concurrent usage of `MEMORY` tables from multiple sessions.

Depending on the kinds of queries performed on a `MEMORY` table, you might create indexes as either the default hash data structure (for looking up single values based on a unique key), or a general-purpose B-tree data structure (for all kinds of queries involving equality, inequality, or range operators such as less than or greater than). The following sections illustrate the syntax for creating both kinds of indexes. A common performance issue is using the default hash indexes in workloads where B-tree indexes are more efficient.

Characteristics of MEMORY Tables

The MEMORY storage engine does not create any files on disk. The table definition is stored in the MySQL data dictionary.

MEMORY tables have the following characteristics:

- Space for MEMORY tables is allocated in small blocks. Tables use 100% dynamic hashing for inserts. No overflow area or extra key space is needed. No extra space is needed for free lists. Deleted rows are put in a linked list and are reused when you insert new data into the table. MEMORY tables also have none of the problems commonly associated with deletes plus inserts in hashed tables.
- MEMORY tables use a fixed-length row-storage format. Variable-length types such as VARCHAR are stored using a fixed length.
- MEMORY tables cannot contain BLOB or TEXT columns.
- MEMORY includes support for AUTO_INCREMENT columns.
- Non-TEMPORARY MEMORY tables are shared among all clients, just like any other non-TEMPORARY table.

DDL Operations for MEMORY Tables

To create a MEMORY table, specify the clause ENGINE=MEMORY on the CREATE TABLE statement.

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

As indicated by the engine name, MEMORY tables are stored in memory. They use hash indexes by default, which makes them very fast for single-value lookups, and very useful for creating temporary tables. However, when the server shuts down, all rows stored in MEMORY tables are lost. The tables themselves continue to exist because their definitions are stored in the MySQL data dictionary, but they are empty when the server restarts.

This example shows how you might create, use, and remove a MEMORY table:

```
mysql> CREATE TABLE test ENGINE=MEMORY
           SELECT ip,SUM(downloads) AS down
           FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

The maximum size of MEMORY tables is limited by the max_heap_table_size system variable, which has a default value of 16MB. To enforce different size limits for MEMORY tables, change the value of this variable. The value in effect for CREATE TABLE, or a subsequent ALTER TABLE or TRUNCATE TABLE, is the value used for the life of the table. A server restart also sets the maximum size of existing MEMORY tables to the global max_heap_table_size value. You can set the size for individual tables as described later in this section.

Indexes

The MEMORY storage engine supports both HASH and BTREE indexes. You can specify one or the other for a given index by adding a USING clause as shown here:

```
CREATE TABLE lookup
    (id INT, INDEX USING HASH (id))
    ENGINE = MEMORY;
```

```
CREATE TABLE lookup
    (id INT, INDEX USING BTREE (id))
    ENGINE = MEMORY;
```

For general characteristics of B-tree and hash indexes, see Section 8.3.1, “How MySQL Uses Indexes”.

`MEMORY` tables can have up to 64 indexes per table, 16 columns per index and a maximum key length of 3072 bytes.

If a `MEMORY` table hash index has a high degree of key duplication (many index entries containing the same value), updates to the table that affect key values and all deletes are significantly slower. The degree of this slowdown is proportional to the degree of duplication (or, inversely proportional to the index cardinality). You can use a `BTREE` index to avoid this problem.

`MEMORY` tables can have nonunique keys. (This is an uncommon feature for implementations of hash indexes.)

Columns that are indexed can contain `NULL` values.

User-Created and Temporary Tables

`MEMORY` table contents are stored in memory, which is a property that `MEMORY` tables share with internal temporary tables that the server creates on the fly while processing queries. However, the two types of tables differ in that `MEMORY` tables are not subject to storage conversion, whereas internal temporary tables are:

- If an internal temporary table becomes too large, the server automatically converts it to on-disk storage, as described in Section 8.4.4, “Internal Temporary Table Use in MySQL”.
- User-created `MEMORY` tables are never converted to disk tables.

Loading Data

To populate a `MEMORY` table when the MySQL server starts, you can use the `init_file` system variable. For example, you can put statements such as `INSERT INTO ... SELECT` or `LOAD DATA` into a file to load the table from a persistent data source, and use `init_file` to name the file. See Section 5.1.8, “Server System Variables”, and Section 13.2.7, “LOAD DATA Syntax”.

MEMORY Tables and Replication

A server's `MEMORY` tables become empty when it is shut down and restarted. If the server is a replication master, its slaves are not aware that these tables have become empty, so you see out-of-date content if you select data from the tables on the slaves. To synchronize master and slave `MEMORY` tables, when a `MEMORY` table is used on a master for the first time since it was started, a `DELETE` statement is written to the master's binary log, to empty the table on the slaves also. The slave still has outdated data in the table during the interval between the master's restart and its first use of the table. To avoid this interval when a direct query to the slave could return stale data, use the `init_file` system variable set to name a file containing statements that populate the `MEMORY` table on the master at startup.

Managing Memory Use

The server needs sufficient memory to maintain all `MEMORY` tables that are in use at the same time.

Memory is not reclaimed if you delete individual rows from a `MEMORY` table. Memory is reclaimed only when the entire table is deleted. Memory that was previously used for deleted rows is re-used for new rows within the same table. To free all the memory used by a `MEMORY` table when you no longer require

its contents, execute `DELETE` or `TRUNCATE TABLE` to remove all rows, or remove the table altogether using `DROP TABLE`. To free up the memory used by deleted rows, use `ALTER TABLE ENGINE=MEMORY` to force a table rebuild.

The memory needed for one row in a `MEMORY` table is calculated using the following expression:

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) * 4)
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) * 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

`ALIGN()` represents a round-up factor to cause the row length to be an exact multiple of the `char` pointer size. `sizeof(char*)` is 4 on 32-bit machines and 8 on 64-bit machines.

As mentioned earlier, the `max_heap_table_size` system variable sets the limit on the maximum size of `MEMORY` tables. To control the maximum size for individual tables, set the session value of this variable before creating each table. (Do not change the global `max_heap_table_size` value unless you intend the value to be used for `MEMORY` tables created by all clients.) The following example creates two `MEMORY` tables, with a maximum size of 1MB and 2MB, respectively:

```
mysql> SET max_heap_table_size = 1024*1024;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.01 sec)

mysql> SET max_heap_table_size = 1024*1024*2;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t2 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.00 sec)
```

Both tables revert to the server's global `max_heap_table_size` value if the server restarts.

You can also specify a `MAX_ROWS` table option in `CREATE TABLE` statements for `MEMORY` tables to provide a hint about the number of rows you plan to store in them. This does not enable the table to grow beyond the `max_heap_table_size` value, which still acts as a constraint on maximum table size. For maximum flexibility in being able to use `MAX_ROWS`, set `max_heap_table_size` at least as high as the value to which you want each `MEMORY` table to be able to grow.

Additional Resources

A forum dedicated to the `MEMORY` storage engine is available at https://forums.mysql.com/list.php?92.

16.4 The CSV Storage Engine

The `CSV` storage engine stores data in text files using comma-separated values format.

The `CSV` storage engine is always compiled into the MySQL server.

To examine the source for the `CSV` engine, look in the `storage/csv` directory of a MySQL source distribution.

When you create a `CSV` table, the server creates a plain text data file having a name that begins with the table name and has a `.CSV` extension. When you store data into the table, the storage engine saves it into the data file in comma-separated values format.

```
mysql> CREATE TABLE test (i INT NOT NULL, c CHAR(10) NOT NULL)
       ENGINE = CSV;
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.05 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
+---+------------+
| i | c          |
+---+------------+
| 1 | record one |
| 2 | record two |
+---+------------+
2 rows in set (0.00 sec)
```

Creating a `CSV` table also creates a corresponding metafile that stores the state of the table and the number of rows that exist in the table. The name of this file is the same as the name of the table with the extension `CSM`.

If you examine the `test.CSV` file in the database directory created by executing the preceding statements, its contents should look like this:

```
"1","record one"
"2","record two"
```

This format can be read, and even written, by spreadsheet applications such as Microsoft Excel or StarOffice Calc.

16.4.1 Repairing and Checking CSV Tables

The `CSV` storage engine supports the `CHECK TABLE` and `REPAIR TABLE` statements to verify and, if possible, repair a damaged `CSV` table.

When running the `CHECK TABLE` statement, the `CSV` file will be checked for validity by looking for the correct field separators, escaped fields (matching or missing quotation marks), the correct number of fields compared to the table definition and the existence of a corresponding `CSV` metafile. The first invalid row discovered will report an error. Checking a valid table produces output like that shown below:

```
mysql> CHECK TABLE csvtest;
+--------------+-------+----------+----------+
| Table        | Op    | Msg_type | Msg_text |
+--------------+-------+----------+----------+
| test.csvtest | check | status   | OK       |
+--------------+-------+----------+----------+
```

A check on a corrupted table returns a fault:

```
mysql> CHECK TABLE csvtest;
+--------------+-------+----------+----------+
| Table        | Op    | Msg_type | Msg_text |
+--------------+-------+----------+----------+
| test.csvtest | check | error    | Corrupt  |
+--------------+-------+----------+----------+
```

If the check fails, the table is marked as crashed (corrupt). Once a table has been marked as corrupt, it is automatically repaired when you next run `CHECK TABLE` or execute a `SELECT` statement. The corresponding corrupt status and new status will be displayed when running `CHECK TABLE`:

```
mysql> CHECK TABLE csvtest;
+--------------+-------+----------+---------------------------+
| Table        | Op    | Msg_type | Msg_text                  |
+--------------+-------+----------+---------------------------+
| test.csvtest | check | warning  | Table is marked as crashed |
| test.csvtest | check | status   | OK                        |
+--------------+-------+----------+---------------------------+
```

To repair a table, use `REPAIR TABLE`, which copies as many valid rows from the existing `CSV` data as possible, and then replaces the existing `CSV` file with the recovered rows. Any rows beyond the corrupted data are lost.

```
mysql> REPAIR TABLE csvtest;
+--------------+--------+----------+----------+
| Table        | Op     | Msg_type | Msg_text |
+--------------+--------+----------+----------+
| test.csvtest | repair | status   | OK       |
+--------------+--------+----------+----------+
```

Warning

During repair, only the rows from the `CSV` file up to the first damaged row are copied to the new table. All other rows from the first damaged row to the end of the table are removed, even valid rows.

16.4.2 CSV Limitations

The `CSV` storage engine does not support indexing.

The `CSV` storage engine does not support partitioning.

All tables that you create using the `CSV` storage engine must have the `NOT NULL` attribute on all columns.

16.5 The ARCHIVE Storage Engine

The `ARCHIVE` storage engine produces special-purpose tables that store large amounts of unindexed data in a very small footprint.

Table 16.5 ARCHIVE Storage Engine Features

Feature	Support
B-tree indexes	No
Backup/point-in-time recovery (Implemented in the server, rather than in the storage engine.)	Yes
Cluster database support	No
Clustered indexes	No
Compressed data	Yes
Data caches	No
Encrypted data	Yes (Implemented in the server via encryption functions.)
Foreign key support	No

Feature	Support
Full-text search indexes	No
Geospatial data type support	Yes
Geospatial indexing support	No
Hash indexes	No
Index caches	No
Locking granularity	Row
MVCC	No
Replication support (Implemented in the server, rather than in the storage engine.)	Yes
Storage limits	None
T-tree indexes	No
Transactions	No
Update statistics for data dictionary	Yes

The `ARCHIVE` storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke `CMake` with the `-DWITH_ARCHIVE_STORAGE_ENGINE` option.

To examine the source for the `ARCHIVE` engine, look in the `storage/archive` directory of a MySQL source distribution.

You can check whether the `ARCHIVE` storage engine is available with the `SHOW ENGINES` statement.

When you create an `ARCHIVE` table, the storage engine creates files with names that begin with the table name. The data file has an extension of `.ARZ`. An `.ARN` file may appear during optimization operations.

The `ARCHIVE` engine supports `INSERT`, `REPLACE`, and `SELECT`, but not `DELETE` or `UPDATE`. It does support `ORDER BY` operations, `BLOB` columns, and spatial data types (see Section 11.5.1, “Spatial Data Types”). Geographic spatial reference systems are not supported. The `ARCHIVE` engine uses row-level locking.

The `ARCHIVE` engine supports the `AUTO_INCREMENT` column attribute. The `AUTO_INCREMENT` column can have either a unique or nonunique index. Attempting to create an index on any other column results in an error. The `ARCHIVE` engine also supports the `AUTO_INCREMENT` table option in `CREATE TABLE` statements to specify the initial sequence value for a new table or reset the sequence value for an existing table, respectively.

`ARCHIVE` does not support inserting a value into an `AUTO_INCREMENT` column less than the current maximum column value. Attempts to do so result in an `ER_DUP_KEY` error.

The `ARCHIVE` engine ignores `BLOB` columns if they are not requested and scans past them while reading.

The `ARCHIVE` storage engine does not support partitioning.

Storage: Rows are compressed as they are inserted. The `ARCHIVE` engine uses `zlib` lossless data compression (see http://www.zlib.net/). You can use `OPTIMIZE TABLE` to analyze the table and pack it into a smaller format (for a reason to use `OPTIMIZE TABLE`, see later in this section). The engine also supports `CHECK TABLE`. There are several types of insertions that are used:

- An `INSERT` statement just pushes rows into a compression buffer, and that buffer flushes as necessary. The insertion into the buffer is protected by a lock. A `SELECT` forces a flush to occur.

- A bulk insert is visible only after it completes, unless other inserts occur at the same time, in which case it can be seen partially. A `SELECT` never causes a flush of a bulk insert unless a normal insert occurs while it is loading.

Retrieval: On retrieval, rows are uncompressed on demand; there is no row cache. A `SELECT` operation performs a complete table scan: When a `SELECT` occurs, it finds out how many rows are currently available and reads that number of rows. `SELECT` is performed as a consistent read. Note that lots of `SELECT` statements during insertion can deteriorate the compression, unless only bulk inserts are used. To achieve better compression, you can use `OPTIMIZE TABLE` or `REPAIR TABLE`. The number of rows in `ARCHIVE` tables reported by `SHOW TABLE STATUS` is always accurate. See Section 13.7.3.4, “OPTIMIZE TABLE Syntax”, Section 13.7.3.5, “REPAIR TABLE Syntax”, and Section 13.7.7.36, “SHOW TABLE STATUS Syntax”.

Additional Resources

- A forum dedicated to the `ARCHIVE` storage engine is available at https://forums.mysql.com/list.php?112.

16.6 The BLACKHOLE Storage Engine

The `BLACKHOLE` storage engine acts as a “black hole” that accepts data but throws it away and does not store it. Retrievals always return an empty result:

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

To enable the `BLACKHOLE` storage engine if you build MySQL from source, invoke `CMake` with the `-DWITH_BLACKHOLE_STORAGE_ENGINE` option.

To examine the source for the `BLACKHOLE` engine, look in the `sql` directory of a MySQL source distribution.

When you create a `BLACKHOLE` table, the server creates the table definition in the global data dictionary. There are no files associated with the table.

The `BLACKHOLE` storage engine supports all kinds of indexes. That is, you can include index declarations in the table definition.

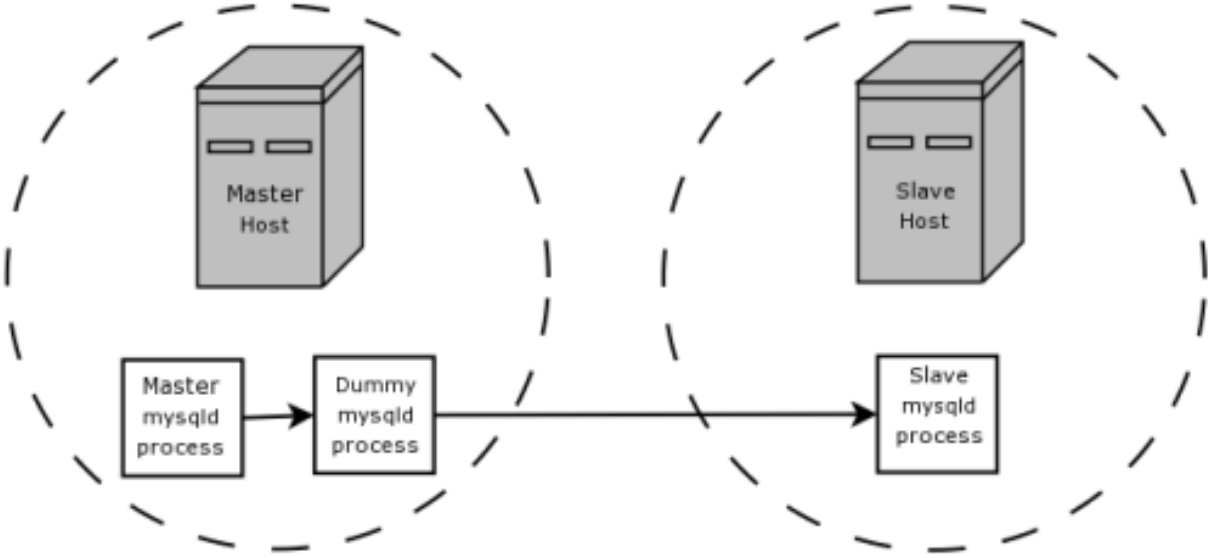
The `BLACKHOLE` storage engine does not support partitioning.

You can check whether the `BLACKHOLE` storage engine is available with the `SHOW ENGINES` statement.

Inserts into a `BLACKHOLE` table do not store any data, but if statement based binary logging is enabled, the SQL statements are logged and replicated to slave servers. This can be useful as a repeater or filter mechanism.

Suppose that your application requires slave-side filtering rules, but transferring all binary log data to the slave first results in too much traffic. In such a case, it is possible to set up on the master host a “dummy” slave process whose default storage engine is `BLACKHOLE`, depicted as follows:

Figure 16.1 Replication using BLACKHOLE for Filtering



The master writes to its binary log. The “dummy” `mysqld` process acts as a slave, applying the desired combination of `replicate-do-*` and `replicate-ignore-*` rules, and writes a new, filtered binary log of its own. (See Section 17.1.6, “Replication and Binary Logging Options and Variables”.) This filtered log is provided to the slave.

The dummy process does not actually store any data, so there is little processing overhead incurred by running the additional `mysqld` process on the replication master host. This type of setup can be repeated with additional replication slaves.

`INSERT` triggers for `BLACKHOLE` tables work as expected. However, because the `BLACKHOLE` table does not actually store any data, `UPDATE` and `DELETE` triggers are not activated: The `FOR EACH ROW` clause in the trigger definition does not apply because there are no rows.

Other possible uses for the `BLACKHOLE` storage engine include:

- Verification of dump file syntax.
- Measurement of the overhead from binary logging, by comparing performance using `BLACKHOLE` with and without binary logging enabled.
- `BLACKHOLE` is essentially a “no-op” storage engine, so it could be used for finding performance bottlenecks not related to the storage engine itself.

The `BLACKHOLE` engine is transaction-aware, in the sense that committed transactions are written to the binary log and rolled-back transactions are not.

Blackhole Engine and Auto Increment Columns

The Blackhole engine is a no-op engine. Any operations performed on a table using Blackhole will have no effect. This should be born in mind when considering the behavior of primary key columns that auto increment. The engine will not automatically increment field values, and does not retain auto increment field state. This has important implications in replication.

Consider the following replication scenario where all three of the following conditions apply:

1. On a master server there is a blackhole table with an auto increment field that is a primary key.
2. On a slave the same table exists but using the MyISAM engine.
3. Inserts are performed into the master's table without explicitly setting the auto increment value in the `INSERT` statement itself or through using a `SET INSERT_ID` statement.

In this scenario replication will fail with a duplicate entry error on the primary key column.

In statement based replication, the value of `INSERT_ID` in the context event will always be the same. Replication will therefore fail due to trying insert a row with a duplicate value for a primary key column.

In row based replication, the value that the engine returns for the row always be the same for each insert. This will result in the slave attempting to replay two insert log entries using the same value for the primary key column, and so replication will fail.

Column Filtering

When using row-based replication, (`binlog_format=ROW`), a slave where the last columns are missing from a table is supported, as described in the section Section 17.5.1.9, “Replication with Differing Table Definitions on Master and Slave”.

This filtering works on the slave side, that is, the columns are copied to the slave before they are filtered out. There are at least two cases where it is not desirable to copy the columns to the slave:

1. If the data is confidential, so the slave server should not have access to it.
2. If the master has many slaves, filtering before sending to the slaves may reduce network traffic.

Master column filtering can be achieved using the `BLACKHOLE` engine. This is carried out in a way similar to how master table filtering is achieved - by using the `BLACKHOLE` engine and the `--replicate-do-table` or `--replicate-ignore-table` option.

The setup for the master is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N,
                 secret_col_1, ..., secret_col_M) ENGINE=MyISAM;
```

The setup for the trusted slave is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N) ENGINE=BLACKHOLE;
```

The setup for the untrusted slave is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N) ENGINE=MyISAM;
```

16.7 The MERGE Storage Engine

The `MERGE` storage engine, also known as the `MRG_MyISAM` engine, is a collection of identical `MyISAM` tables that can be used as one. “Identical” means that all tables have identical column data types and index information. You cannot merge `MyISAM` tables in which the columns are listed in a different order, do not have exactly the same data types in corresponding columns, or have the indexes in different order. However, any or all of the `MyISAM` tables can be compressed with `myisampack`. See Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”. Differences between tables such as these do not matter:

- Names of corresponding columns and indexes can differ.
- Comments for tables, columns, and indexes can differ.
- Table options such as `AVG_ROW_LENGTH`, `MAX_ROWS`, or `PACK_KEYS` can differ.

An alternative to a `MERGE` table is a partitioned table, which stores partitions of a single table in separate files and enables some operations to be performed more efficiently. For more information, see Chapter 23, *Partitioning*.

When you create a `MERGE` table, MySQL creates a `.MRG` file on disk that contains the names of the underlying `MyISAM` tables that should be used as one. The table format of the `MERGE` table is stored in the MySQL data dictionary. The underlying tables do not have to be in the same database as the `MERGE` table.

You can use `SELECT`, `DELETE`, `UPDATE`, and `INSERT` on `MERGE` tables. You must have `SELECT`, `DELETE`, and `UPDATE` privileges on the `MyISAM` tables that you map to a `MERGE` table.

Note

The use of `MERGE` tables entails the following security issue: If a user has access to `MyISAM` table *`t`*, that user can create a `MERGE` table *`m`* that accesses *`t`*. However, if the user's privileges on *`t`* are subsequently revoked, the user can continue to access *`t`* by doing so through *`m`*.

Use of `DROP TABLE` with a `MERGE` table drops only the `MERGE` specification. The underlying tables are not affected.

To create a `MERGE` table, you must specify a `UNION=(`*`list-of-tables`*`)` option that indicates which `MyISAM` tables to use. You can optionally specify an `INSERT_METHOD` option to control how inserts into the `MERGE` table take place. Use a value of `FIRST` or `LAST` to cause inserts to be made in the first or last underlying table, respectively. If you specify no `INSERT_METHOD` option or if you specify it with a value of `NO`, inserts into the `MERGE` table are not permitted and attempts to do so result in an error.

The following example shows how to create a `MERGE` table:

```
mysql> CREATE TABLE t1 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20)) ENGINE=MyISAM;
mysql> CREATE TABLE t2 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20)) ENGINE=MyISAM;
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
    ->    a INT NOT NULL AUTO_INCREMENT,
    ->    message CHAR(20), INDEX(a))
    ->    ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

Column `a` is indexed as a `PRIMARY KEY` in the underlying `MyISAM` tables, but not in the `MERGE` table. There it is indexed but not as a `PRIMARY KEY` because a `MERGE` table cannot enforce uniqueness over the set of underlying tables. (Similarly, a column with a `UNIQUE` index in the underlying tables should be indexed in the `MERGE` table but not as a `UNIQUE` index.)

After creating the `MERGE` table, you can use it to issue queries that operate on the group of tables as a whole:

```
mysql> SELECT * FROM total;
```

```
+---+---------+
| a | message |
+---+---------+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

To remap a `MERGE` table to a different collection of `MyISAM` tables, you can use one of the following methods:

- `DROP` the `MERGE` table and re-create it.
- Use `ALTER TABLE tbl_name UNION=(...)` to change the list of underlying tables.

  It is also possible to use `ALTER TABLE ... UNION=()` (that is, with an empty `UNION` clause) to remove all of the underlying tables. However, in this case, the table is effectively empty and inserts fail because there is no underlying table to take new rows. Such a table might be useful as a template for creating new `MERGE` tables with `CREATE TABLE ... LIKE`.

The underlying table definitions and indexes must conform closely to the definition of the `MERGE` table. Conformance is checked when a table that is part of a `MERGE` table is opened, not when the `MERGE` table is created. If any table fails the conformance checks, the operation that triggered the opening of the table fails. This means that changes to the definitions of tables within a `MERGE` may cause a failure when the `MERGE` table is accessed. The conformance checks applied to each table are:

- The underlying table and the `MERGE` table must have the same number of columns.
- The column order in the underlying table and the `MERGE` table must match.
- Additionally, the specification for each corresponding column in the parent `MERGE` table and the underlying tables are compared and must satisfy these checks:
  - The column type in the underlying table and the `MERGE` table must be equal.
  - The column length in the underlying table and the `MERGE` table must be equal.
  - The column of the underlying table and the `MERGE` table can be `NULL`.
- The underlying table must have at least as many indexes as the `MERGE` table. The underlying table may have more indexes than the `MERGE` table, but cannot have fewer.

  Note

  A known issue exists where indexes on the same columns must be in identical order, in both the `MERGE` table and the underlying `MyISAM` table. See Bug #33653.

  Each index must satisfy these checks:

  - The index type of the underlying table and the `MERGE` table must be the same.
  - The number of index parts (that is, multiple columns within a compound index) in the index definition for the underlying table and the `MERGE` table must be the same.
  - For each index part:

- Index part lengths must be equal.
- Index part types must be equal.
- Index part languages must be equal.
- Check whether index parts can be `NULL`.

If a `MERGE` table cannot be opened or used because of a problem with an underlying table, `CHECK TABLE` displays information about which table caused the problem.

Additional Resources

- A forum dedicated to the `MERGE` storage engine is available at https://forums.mysql.com/list.php?93.

16.7.1 MERGE Table Advantages and Disadvantages

`MERGE` tables can help you solve the following problems:

- Easily manage a set of log tables. For example, you can put data from different months into separate tables, compress some of them with `myisampack`, and then create a `MERGE` table to use them as one.
- Obtain more speed. You can split a large read-only table based on some criteria, and then put individual tables on different disks. A `MERGE` table structured this way could be much faster than using a single large table.
- Perform more efficient searches. If you know exactly what you are looking for, you can search in just one of the underlying tables for some queries and use a `MERGE` table for others. You can even have many different `MERGE` tables that use overlapping sets of tables.
- Perform more efficient repairs. It is easier to repair individual smaller tables that are mapped to a `MERGE` table than to repair a single large table.
- Instantly map many tables as one. A `MERGE` table need not maintain an index of its own because it uses the indexes of the individual tables. As a result, `MERGE` table collections are *very* fast to create or remap. (You must still specify the index definitions when you create a `MERGE` table, even though no indexes are created.)
- If you have a set of tables from which you create a large table on demand, you can instead create a `MERGE` table from them on demand. This is much faster and saves a lot of disk space.
- Exceed the file size limit for the operating system. Each `MyISAM` table is bound by this limit, but a collection of `MyISAM` tables is not.
- You can create an alias or synonym for a `MyISAM` table by defining a `MERGE` table that maps to that single table. There should be no really notable performance impact from doing this (only a couple of indirect calls and `memcpy()` calls for each read).

The disadvantages of `MERGE` tables are:

- You can use only identical `MyISAM` tables for a `MERGE` table.
- Some `MyISAM` features are unavailable in `MERGE` tables. For example, you cannot create `FULLTEXT` indexes on `MERGE` tables. (You can create `FULLTEXT` indexes on the underlying `MyISAM` tables, but you cannot search the `MERGE` table with a full-text search.)

- If the MERGE table is nontemporary, all underlying MyISAM tables must be nontemporary. If the MERGE table is temporary, the MyISAM tables can be any mix of temporary and nontemporary.
- MERGE tables use more file descriptors than MyISAM tables. If 10 clients are using a MERGE table that maps to 10 tables, the server uses (10 × 10) + 10 file descriptors. (10 data file descriptors for each of the 10 clients, and 10 index file descriptors shared among the clients.)
- Index reads are slower. When you read an index, the MERGE storage engine needs to issue a read on all underlying tables to check which one most closely matches a given index value. To read the next index value, the MERGE storage engine needs to search the read buffers to find the next value. Only when one index buffer is used up does the storage engine need to read the next index block. This makes MERGE indexes much slower on eq_ref searches, but not much slower on ref searches. For more information about eq_ref and ref, see Section 13.8.2, “EXPLAIN Syntax”.

16.7.2 MERGE Table Problems

The following are known problems with MERGE tables:

- In versions of MySQL Server prior to 5.1.23, it was possible to create temporary merge tables with nontemporary child MyISAM tables.

  From versions 5.1.23, MERGE children were locked through the parent table. If the parent was temporary, it was not locked and so the children were not locked either. Parallel use of the MyISAM tables corrupted them.

- If you use ALTER TABLE to change a MERGE table to another storage engine, the mapping to the underlying tables is lost. Instead, the rows from the underlying MyISAM tables are copied into the altered table, which then uses the specified storage engine.
- The INSERT_METHOD table option for a MERGE table indicates which underlying MyISAM table to use for inserts into the MERGE table. However, use of the AUTO_INCREMENT table option for that MyISAM table has no effect for inserts into the MERGE table until at least one row has been inserted directly into the MyISAM table.
- A MERGE table cannot maintain uniqueness constraints over the entire table. When you perform an INSERT, the data goes into the first or last MyISAM table (as determined by the INSERT_METHOD option). MySQL ensures that unique key values remain unique within that MyISAM table, but not over all the underlying tables in the collection.
- Because the MERGE engine cannot enforce uniqueness over the set of underlying tables, REPLACE does not work as expected. The two key facts are:
  - REPLACE can detect unique key violations only in the underlying table to which it is going to write (which is determined by the INSERT_METHOD option). This differs from violations in the MERGE table itself.
  - If REPLACE detects a unique key violation, it will change only the corresponding row in the underlying table it is writing to; that is, the first or last table, as determined by the INSERT_METHOD option.

  Similar considerations apply for INSERT ... ON DUPLICATE KEY UPDATE.
- MERGE tables do not support partitioning. That is, you cannot partition a MERGE table, nor can any of a MERGE table's underlying MyISAM tables be partitioned.
- You should not use ANALYZE TABLE, REPAIR TABLE, OPTIMIZE TABLE, ALTER TABLE, DROP TABLE, DELETE without a WHERE clause, or TRUNCATE TABLE on any of the tables that are mapped into an open MERGE table. If you do so, the MERGE table may still refer to the original table and yield

unexpected results. To work around this problem, ensure that no MERGE tables remain open by issuing a FLUSH TABLES statement prior to performing any of the named operations.

The unexpected results include the possibility that the operation on the MERGE table will report table corruption. If this occurs after one of the named operations on the underlying MyISAM tables, the corruption message is spurious. To deal with this, issue a FLUSH TABLES statement after modifying the MyISAM tables.

- DROP TABLE on a table that is in use by a MERGE table does not work on Windows because the MERGE storage engine's table mapping is hidden from the upper layer of MySQL. Windows does not permit open files to be deleted, so you first must flush all MERGE tables (with FLUSH TABLES) or drop the MERGE table before dropping the table.
- The definition of the MyISAM tables and the MERGE table are checked when the tables are accessed (for example, as part of a SELECT or INSERT statement). The checks ensure that the definitions of the tables and the parent MERGE table definition match by comparing column order, types, sizes and associated indexes. If there is a difference between the tables, an error is returned and the statement fails. Because these checks take place when the tables are opened, any changes to the definition of a single table, including column changes, column ordering, and engine alterations will cause the statement to fail.
- The order of indexes in the MERGE table and its underlying tables should be the same. If you use ALTER TABLE to add a UNIQUE index to a table used in a MERGE table, and then use ALTER TABLE to add a nonunique index on the MERGE table, the index ordering is different for the tables if there was already a nonunique index in the underlying table. (This happens because ALTER TABLE puts UNIQUE indexes before nonunique indexes to facilitate rapid detection of duplicate keys.) Consequently, queries on tables with such indexes may return unexpected results.
- If you encounter an error message similar to ERROR 1017 (HY000): Can't find file: '*tbl_name*.MRG' (errno: 2), it generally indicates that some of the underlying tables do not use the MyISAM storage engine. Confirm that all of these tables are MyISAM.
- The maximum number of rows in a MERGE table is 2^{64} (~1.844E+19; the same as for a MyISAM table). It is not possible to merge multiple MyISAM tables into a single MERGE table that would have more than this number of rows.
- Use of underlying MyISAM tables of differing row formats with a parent MERGE table is currently known to fail. See Bug #32364.
- You cannot change the union list of a nontemporary MERGE table when LOCK TABLES is in effect. The following does *not* work:

```
CREATE TABLE m1 ... ENGINE=MRG_MYISAM ...;
LOCK TABLES t1 WRITE, t2 WRITE, m1 WRITE;
ALTER TABLE m1 ... UNION=(t1,t2) ...;
```

  However, you can do this with a temporary MERGE table.

- You cannot create a MERGE table with CREATE ... SELECT, neither as a temporary MERGE table, nor as a nontemporary MERGE table. For example:

```
CREATE TABLE m1 ... ENGINE=MRG_MYISAM ... SELECT ...;
```

  Attempts to do this result in an error: *tbl_name* is not BASE TABLE.

- In some cases, differing PACK_KEYS table option values among the MERGE and underlying tables cause unexpected results if the underlying tables contain CHAR or BINARY columns. As a workaround, use ALTER TABLE to ensure that all involved tables have the same PACK_KEYS value. (Bug #50646)

16.8 The FEDERATED Storage Engine

The FEDERATED storage engine lets you access data from a remote MySQL database without using replication or cluster technology. Querying a local FEDERATED table automatically pulls the data from the remote (federated) tables. No data is stored on the local tables.

To include the FEDERATED storage engine if you build MySQL from source, invoke CMake with the -DWITH_FEDERATED_STORAGE_ENGINE option.

The FEDERATED storage engine is not enabled by default in the running server; to enable FEDERATED, you must start the MySQL server binary using the --federated option.

To examine the source for the FEDERATED engine, look in the storage/federated directory of a MySQL source distribution.

16.8.1 FEDERATED Storage Engine Overview

When you create a table using one of the standard storage engines (such as MyISAM, CSV or InnoDB), the table consists of the table definition and the associated data. When you create a FEDERATED table, the table definition is the same, but the physical storage of the data is handled on a remote server.

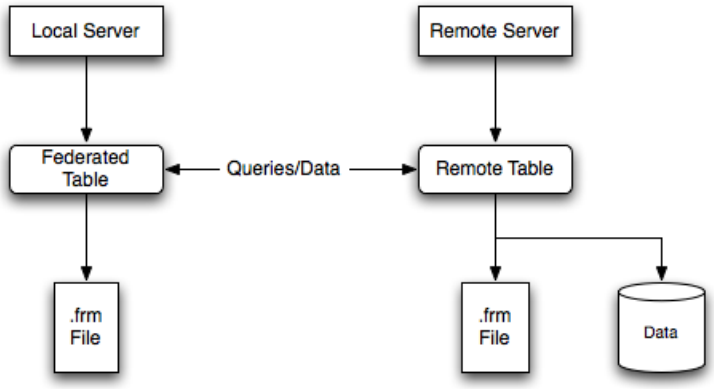
A FEDERATED table consists of two elements:

- A *remote server* with a database table, which in turn consists of the table definition (stored in the MySQL data dictionary) and the associated table. The table type of the remote table may be any type supported by the remote mysqld server, including MyISAM or InnoDB.
- A *local server* with a database table, where the table definition matches that of the corresponding table on the remote server. The table definition is stored in the data dictionary. There is no data file on the local server. Instead, the table definition includes a connection string that points to the remote table.

When executing queries and statements on a FEDERATED table on the local server, the operations that would normally insert, update or delete information from a local data file are instead sent to the remote server for execution, where they update the data file on the remote server or return matching rows from the remote server.

The basic structure of a FEDERATED table setup is shown in Figure 16.2, “FEDERATED Table Structure”.

Figure 16.2 FEDERATED Table Structure



When a client issues an SQL statement that refers to a `FEDERATED` table, the flow of information between the local server (where the SQL statement is executed) and the remote server (where the data is physically stored) is as follows:

1. The storage engine looks through each column that the `FEDERATED` table has and constructs an appropriate SQL statement that refers to the remote table.
2. The statement is sent to the remote server using the MySQL client API.
3. The remote server processes the statement and the local server retrieves any result that the statement produces (an affected-rows count or a result set).
4. If the statement produces a result set, each column is converted to internal storage engine format that the `FEDERATED` engine expects and can use to display the result to the client that issued the original statement.

The local server communicates with the remote server using MySQL client C API functions. It invokes `mysql_real_query()` to send the statement. To read a result set, it uses `mysql_store_result()` and fetches rows one at a time using `mysql_fetch_row()`.

16.8.2 How to Create FEDERATED Tables

To create a `FEDERATED` table you should follow these steps:

1. Create the table on the remote server. Alternatively, make a note of the table definition of an existing table, perhaps using the `SHOW CREATE TABLE` statement.
2. Create the table on the local server with an identical table definition, but adding the connection information that links the local table to the remote table.

For example, you could create the following table on the remote server:

```
CREATE TABLE test_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=MyISAM
DEFAULT CHARSET=utf8mb4;
```

To create the local table that will be federated to the remote table, there are two options available. You can either create the local table and specify the connection string (containing the server name, login, password) to be used to connect to the remote table using the `CONNECTION`, or you can use an existing connection that you have previously created using the `CREATE SERVER` statement.

Important

When you create the local table it *must* have an identical field definition to the remote table.

Note

You can improve the performance of a `FEDERATED` table by adding indexes to the table on the host. The optimization will occur because the query sent to the remote server will include the contents of the `WHERE` clause and will be sent to the remote

server and subsequently executed locally. This reduces the network traffic that would otherwise request the entire table from the server for local processing.

16.8.2.1 Creating a FEDERATED Table Using CONNECTION

To use the first method, you must specify the `CONNECTION` string after the engine type in a `CREATE TABLE` statement. For example:

```
CREATE TABLE federated_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=utf8mb4
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

Note

`CONNECTION` replaces the `COMMENT` used in some previous versions of MySQL.

The `CONNECTION` string contains the information required to connect to the remote server containing the table that will be used to physically store the data. The connection string specifies the server name, login credentials, port number and database/table information. In the example, the remote table is on the server `remote_host`, using port 9306. The name and port number should match the host name (or IP address) and port number of the remote MySQL server instance you want to use as your remote table.

The format of the connection string is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Where:

- *`scheme`*: A recognized connection protocol. Only `mysql` is supported as the *`scheme`* value at this point.
- *`user_name`*: The user name for the connection. This user must have been created on the remote server, and must have suitable privileges to perform the required actions (`SELECT`, `INSERT`, `UPDATE`, and so forth) on the remote table.
- *`password`*: (Optional) The corresponding password for *`user_name`*.
- *`host_name`*: The host name or IP address of the remote server.
- *`port_num`*: (Optional) The port number for the remote server. The default is 3306.
- *`db_name`*: The name of the database holding the remote table.
- *`tbl_name`*: The name of the remote table. The name of the local and the remote table do not have to match.

Sample connection strings:

```
CONNECTION='mysql://username:password@hostname:port/database/tablename'
CONNECTION='mysql://username@hostname/database/tablename'
CONNECTION='mysql://username:password@hostname/database/tablename'
```

16.8.2.2 Creating a FEDERATED Table Using CREATE SERVER

If you are creating a number of `FEDERATED` tables on the same server, or if you want to simplify the process of creating `FEDERATED` tables, you can use the `CREATE SERVER` statement to define the server connection parameters, just as you would with the `CONNECTION` string.

The format of the `CREATE SERVER` statement is:

```
CREATE SERVER
server_name
FOREIGN DATA WRAPPER wrapper_name
OPTIONS (option [, option] ...)
```

The *`server_name`* is used in the connection string when creating a new `FEDERATED` table.

For example, to create a server connection identical to the `CONNECTION` string:

```
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

You would use the following statement:

```
CREATE SERVER fedlink
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'fed_user', HOST 'remote_host', PORT 9306, DATABASE 'federated');
```

To create a `FEDERATED` table that uses this connection, you still use the `CONNECTION` keyword, but specify the name you used in the `CREATE SERVER` statement.

```
CREATE TABLE test_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=utf8mb4
CONNECTION='fedlink/test_table';
```

The connection name in this example contains the name of the connection (`fedlink`) and the name of the table (`test_table`) to link to, separated by a slash. If you specify only the connection name without a table name, the table name of the local table is used instead.

For more information on `CREATE SERVER`, see Section 13.1.18, “CREATE SERVER Syntax”.

The `CREATE SERVER` statement accepts the same arguments as the `CONNECTION` string. The `CREATE SERVER` statement updates the rows in the `mysql.servers` table. See the following table for information on the correspondence between parameters in a connection string, options in the `CREATE SERVER` statement, and the columns in the `mysql.servers` table. For reference, the format of the `CONNECTION` string is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Description	`CONNECTION` string	`CREATE SERVER` option	`mysql.servers` column
Connection scheme	*`scheme`*	`wrapper_name`	`Wrapper`
Remote user	*`user_name`*	`USER`	`Username`

Description	`CONNECTION` string	`CREATE SERVER` option	`mysql.servers` column
Remote password	*`password`*	`PASSWORD`	`Password`
Remote host	*`host_name`*	`HOST`	`Host`
Remote port	*`port_num`*	`PORT`	`Port`
Remote database	*`db_name`*	`DATABASE`	`Db`

16.8.3 FEDERATED Storage Engine Notes and Tips

You should be aware of the following points when using the `FEDERATED` storage engine:

- `FEDERATED` tables may be replicated to other slaves, but you must ensure that the slave servers are able to use the user/password combination that is defined in the `CONNECTION` string (or the row in the `mysql.servers` table) to connect to the remote server.

The following items indicate features that the `FEDERATED` storage engine does and does not support:

- The remote server must be a MySQL server.
- The remote table that a `FEDERATED` table points to *must* exist before you try to access the table through the `FEDERATED` table.
- It is possible for one `FEDERATED` table to point to another, but you must be careful not to create a loop.
- A `FEDERATED` table does not support indexes in the usual sense; because access to the table data is handled remotely, it is actually the remote table that makes use of indexes. This means that, for a query that cannot use any indexes and so requires a full table scan, the server fetches all rows from the remote table and filters them locally. This occurs regardless of any `WHERE` or `LIMIT` used with this `SELECT` statement; these clauses are applied locally to the returned rows.

  Queries that fail to use indexes can thus cause poor performance and network overload. In addition, since returned rows must be stored in memory, such a query can also lead to the local server swapping, or even hanging.

- Care should be taken when creating a `FEDERATED` table since the index definition from an equivalent `MyISAM` or other table may not be supported. For example, creating a `FEDERATED` table with an index prefix on `VARCHAR`, `TEXT` or `BLOB` columns will fail. The following definition in `MyISAM` is valid:

  ```
  CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A`(30))) ENGINE=MYISAM;
  ```

  The key prefix in this example is incompatible with the `FEDERATED` engine, and the equivalent statement will fail:

  ```
  CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A`(30))) ENGINE=FEDERATED
    CONNECTION='MYSQL://127.0.0.1:3306/TEST/T1';
  ```

  If possible, you should try to separate the column and index definition when creating tables on both the remote server and the local server to avoid these index issues.

- Internally, the implementation uses `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, but not `HANDLER`.
- The `FEDERATED` storage engine supports `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE TABLE`, and indexes. It does not support `ALTER TABLE`, or any Data Definition Language statements that directly affect the structure of the table, other than `DROP TABLE`. The current implementation does not use prepared statements.

- `FEDERATED` accepts `INSERT ... ON DUPLICATE KEY UPDATE` statements, but if a duplicate-key violation occurs, the statement fails with an error.
- Transactions are not supported.
- `FEDERATED` performs bulk-insert handling such that multiple rows are sent to the remote table in a batch, which improves performance. Also, if the remote table is transactional, it enables the remote storage engine to perform statement rollback properly should an error occur. This capability has the following limitations:
  - The size of the insert cannot exceed the maximum packet size between servers. If the insert exceeds this size, it is broken into multiple packets and the rollback problem can occur.
  - Bulk-insert handling does not occur for `INSERT ... ON DUPLICATE KEY UPDATE`.
- There is no way for the `FEDERATED` engine to know if the remote table has changed. The reason for this is that this table must work like a data file that would never be written to by anything other than the database system. The integrity of the data in the local table could be breached if there was any change to the remote database.
- When using a `CONNECTION` string, you cannot use an '@' character in the password. You can get round this limitation by using the `CREATE SERVER` statement to create a server connection.
- The `insert_id` and `timestamp` options are not propagated to the data provider.
- Any `DROP TABLE` statement issued against a `FEDERATED` table drops only the local table, not the remote table.
- `FEDERATED` tables do not work with the query cache.
- User-defined partitioning is not supported for `FEDERATED` tables.

16.8.4 FEDERATED Storage Engine Resources

The following additional resources are available for the `FEDERATED` storage engine:

- A forum dedicated to the `FEDERATED` storage engine is available at https://forums.mysql.com/list.php?105.

16.9 The EXAMPLE Storage Engine

The `EXAMPLE` storage engine is a stub engine that does nothing. Its purpose is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

To enable the `EXAMPLE` storage engine if you build MySQL from source, invoke `CMake` with the `-DWITH_EXAMPLE_STORAGE_ENGINE` option.

To examine the source for the `EXAMPLE` engine, look in the `storage/example` directory of a MySQL source distribution.

When you create an `EXAMPLE` table, no files are created. No data can be stored into the table. Retrievals return an empty result.

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
```

```
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't »
                    have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```

The EXAMPLE storage engine does not support indexing.

The EXAMPLE storage engine does not support partitioning.

16.10 Other Storage Engines

Other storage engines may be available from third parties and community members that have used the Custom Storage Engine interface.

Third party engines are not supported by MySQL. For further information, documentation, installation guides, bug reporting or for any help or assistance with these engines, please contact the developer of the engine directly.

For more information on developing a customer storage engine that can be used with the Pluggable Storage Engine Architecture, see MySQL Internals: Writing a Custom Storage Engine.

16.11 Overview of MySQL Storage Engine Architecture

The MySQL pluggable storage engine architecture enables a database professional to select a specialized storage engine for a particular application need while being completely shielded from the need to manage any specific application coding requirements. The MySQL server architecture isolates the application programmer and DBA from all of the low-level implementation details at the storage level, providing a consistent and easy application model and API. Thus, although there are different capabilities across different storage engines, the application is shielded from these differences.

The pluggable storage engine architecture provides a standard set of management and support services that are common among all underlying storage engines. The storage engines themselves are the components of the database server that actually perform actions on the underlying data that is maintained at the physical server level.

This efficient and modular architecture provides huge benefits for those wishing to specifically target a particular application need—such as data warehousing, transaction processing, or high availability situations—while enjoying the advantage of utilizing a set of interfaces and services that are independent of any one storage engine.

The application programmer and DBA interact with the MySQL database through Connector APIs and service layers that are above the storage engines. If application changes bring about requirements that demand the underlying storage engine change, or that one or more storage engines be added to support new needs, no significant coding or process changes are required to make things work. The MySQL server architecture shields the application from the underlying complexity of the storage engine by presenting a consistent and easy-to-use API that applies across storage engines.

16.11.1 Pluggable Storage Engine Architecture

MySQL Server uses a pluggable storage engine architecture that enables storage engines to be loaded into and unloaded from a running MySQL server.

Plugging in a Storage Engine

Before a storage engine can be used, the storage engine plugin shared library must be loaded into MySQL using the `INSTALL PLUGIN` statement. For example, if the `EXAMPLE` engine plugin is named `example` and the shared library is named `ha_example.so`, you load it with the following statement:

```
INSTALL PLUGIN example SONAME 'ha_example.so';
```

To install a pluggable storage engine, the plugin file must be located in the MySQL plugin directory, and the user issuing the `INSTALL PLUGIN` statement must have `INSERT` privilege for the `mysql.plugin` table.

The shared library must be located in the MySQL server plugin directory, the location of which is given by the `plugin_dir` system variable.

Unplugging a Storage Engine

To unplug a storage engine, use the `UNINSTALL PLUGIN` statement:

```
UNINSTALL PLUGIN example;
```

If you unplug a storage engine that is needed by existing tables, those tables become inaccessible, but will still be present on disk (where applicable). Ensure that there are no tables using a storage engine before you unplug the storage engine.

16.11.2 The Common Database Server Layer

A MySQL pluggable storage engine is the component in the MySQL database server that is responsible for performing the actual data I/O operations for a database as well as enabling and enforcing certain feature sets that target a specific application need. A major benefit of using specific storage engines is that you are only delivered the features needed for a particular application, and therefore you have less system overhead in the database, with the end result being more efficient and higher database performance. This is one of the reasons that MySQL has always been known to have such high performance, matching or beating proprietary monolithic databases in industry standard benchmarks.

From a technical perspective, what are some of the unique supporting infrastructure components that are in a storage engine? Some of the key feature differentiations include:

- *Concurrency*: Some applications have more granular lock requirements (such as row-level locks) than others. Choosing the right locking strategy can reduce overhead and therefore improve overall performance. This area also includes support for capabilities such as multi-version concurrency control or “snapshot” read.
- *Transaction Support*: Not every application needs transactions, but for those that do, there are very well defined requirements such as ACID compliance and more.
- *Referential Integrity*: The need to have the server enforce relational database referential integrity through DDL defined foreign keys.
- *Physical Storage*: This involves everything from the overall page size for tables and indexes as well as the format used for storing data to physical disk.
- *Index Support*: Different application scenarios tend to benefit from different index strategies. Each storage engine generally has its own indexing methods, although some (such as B-tree indexes) are common to nearly all engines.

- *Memory Caches*: Different applications respond better to some memory caching strategies than others, so although some memory caches are common to all storage engines (such as those used for user connections), others are uniquely defined only when a particular storage engine is put in play.
- *Performance Aids*: This includes multiple I/O threads for parallel operations, thread concurrency, database checkpointing, bulk insert handling, and more.
- *Miscellaneous Target Features*: This may include support for geospatial operations, security restrictions for certain data manipulation operations, and other similar features.

Each set of the pluggable storage engine infrastructure components are designed to offer a selective set of benefits for a particular application. Conversely, avoiding a set of component features helps reduce unnecessary overhead. It stands to reason that understanding a particular application's set of requirements and selecting the proper MySQL storage engine can have a dramatic impact on overall system efficiency and performance.

Chapter 17 Replication

Table of Contents

17.1 Configuring Replication 3161
17.1.1 Binary Log File Position Based Replication Configuration Overview 3161
17.1.2 Setting Up Binary Log File Position Based Replication 3162
17.1.3 Replication with Global Transaction Identifiers 3172
17.1.4 MySQL Multi-Source Replication 3194
17.1.5 Changing Replication Modes on Online Servers 3198
17.1.6 Replication and Binary Logging Options and Variables 3204
17.1.7 Common Replication Administration Tasks 3300
17.2 Replication Implementation 3304
17.2.1 Replication Formats 3304
17.2.2 Replication Implementation Details 3312
17.2.3 Replication Channels 3314
17.2.4 Replication Relay and Status Logs 3317
17.2.5 How Servers Evaluate Replication Filtering Rules 3323
17.3 Replication Security 3331
17.3.1 Setting Up Replication to Use Encrypted Connections 3331
17.3.2 Encrypting Binary Log Files and Relay Log Files 3333
17.3.3 Replication Privilege Checks 3337
17.4 Replication Solutions 3342
17.4.1 Using Replication for Backups 3342
17.4.2 Handling an Unexpected Halt of a Replication Slave 3346
17.4.3 Monitoring Row-based Replication 3348
17.4.4 Using Replication with Different Master and Slave Storage Engines 3349
17.4.5 Using Replication for Scale-Out 3350
17.4.6 Replicating Different Databases to Different Slaves 3351
17.4.7 Improving Replication Performance 3352
17.4.8 Switching Masters During Failover 3354
17.4.9 Semisynchronous Replication 3356
17.4.10 Delayed Replication 3361
17.5 Replication Notes and Tips 3364
17.5.1 Replication Features and Issues 3364
17.5.2 Replication Compatibility Between MySQL Versions 3390
17.5.3 Upgrading a Replication Setup 3391
17.5.4 Troubleshooting Replication 3393
17.5.5 How to Report Replication Bugs or Problems 3394

Replication enables data from one MySQL database server (the master) to be copied to one or more MySQL database servers (the slaves). Replication is asynchronous by default; slaves do not need to be connected permanently to receive updates from the master. Depending on the configuration, you can replicate all databases, selected databases, or even selected tables within a database.

Advantages of replication in MySQL include:

- Scale-out solutions - spreading the load among multiple slaves to improve performance. In this environment, all writes and updates must take place on the master server. Reads, however, may take place on one or more slaves. This model can improve the performance of writes (since the master is dedicated to updates), while dramatically increasing read speed across an increasing number of slaves.

- Data security - because data is replicated to the slave, and the slave can pause the replication process, it is possible to run backup services on the slave without corrupting the corresponding master data.
- Analytics - live data can be created on the master, while the analysis of the information can take place on the slave without affecting the performance of the master.
- Long-distance data distribution - you can use replication to create a local copy of data for a remote site to use, without permanent access to the master.

For information on how to use replication in such scenarios, see Section 17.4, “Replication Solutions”.

MySQL 8.0 supports different methods of replication. The traditional method is based on replicating events from the master's binary log, and requires the log files and positions in them to be synchronized between master and slave. The newer method based on *global transaction identifiers* (GTIDs) is transactional and therefore does not require working with log files or positions within these files, which greatly simplifies many common replication tasks. Replication using GTIDs guarantees consistency between master and slave as long as all transactions committed on the master have also been applied on the slave. For more information about GTIDs and GTID-based replication in MySQL, see Section 17.1.3, “Replication with Global Transaction Identifiers”. For information on using binary log file position based replication, see Section 17.1, “Configuring Replication”.

Replication in MySQL supports different types of synchronization. The original type of synchronization is one-way, asynchronous replication, in which one server acts as the master, while one or more other servers act as slaves. This is in contrast to the *synchronous* replication which is a characteristic of NDB Cluster (see Chapter 22, *MySQL NDB Cluster 8.0*). In MySQL 8.0, semisynchronous replication is supported in addition to the built-in asynchronous replication. With semisynchronous replication, a commit performed on the master blocks before returning to the session that performed the transaction until at least one slave acknowledges that it has received and logged the events for the transaction; see Section 17.4.9, “Semisynchronous Replication”. MySQL 8.0 also supports delayed replication such that a slave server deliberately lags behind the master by at least a specified amount of time; see Section 17.4.10, “Delayed Replication”. For scenarios where *synchronous* replication is required, use NDB Cluster (see Chapter 22, *MySQL NDB Cluster 8.0*).

There are a number of solutions available for setting up replication between servers, and the best method to use depends on the presence of data and the engine types you are using. For more information on the available options, see Section 17.1.2, “Setting Up Binary Log File Position Based Replication”.

There are two core types of replication format, Statement Based Replication (SBR), which replicates entire SQL statements, and Row Based Replication (RBR), which replicates only the changed rows. You can also use a third variety, Mixed Based Replication (MBR). For more information on the different replication formats, see Section 17.2.1, “Replication Formats”.

Replication is controlled through a number of different options and variables. For more information, see Section 17.1.6, “Replication and Binary Logging Options and Variables”. Additional security measures can be applied to a replication topology, as described in Section 17.3, “Replication Security”.

You can use replication to solve a number of different problems, including performance, supporting the backup of different databases, and as part of a larger solution to alleviate system failures. For information on how to address these issues, see Section 17.4, “Replication Solutions”.

For notes and tips on how different data types and statements are treated during replication, including details of replication features, version compatibility, upgrades, and potential problems and their resolution, see Section 17.5, “Replication Notes and Tips”. For answers to some questions often asked by those who are new to MySQL Replication, see Section A.14, “MySQL 8.0 FAQ: Replication”.

For detailed information on the implementation of replication, how replication works, the process and contents of the binary log, background threads and the rules used to decide how statements are recorded and replicated, see Section 17.2, “Replication Implementation”.

17.1 Configuring Replication

This section describes how to configure the different types of replication available in MySQL and includes the setup and configuration required for a replication environment, including step-by-step instructions for creating a new replication environment. The major components of this section are:

- For a guide to setting up two or more servers for replication using binary log file positions, Section 17.1.2, “Setting Up Binary Log File Position Based Replication”, deals with the configuration of the servers and provides methods for copying data between the master and slaves.
- For a guide to setting up two or more servers for replication using GTID transactions, Section 17.1.3, “Replication with Global Transaction Identifiers”, deals with the configuration of the servers.
- Events in the binary log are recorded using a number of formats. These are referred to as statement-based replication (SBR) or row-based replication (RBR). A third type, mixed-format replication (MIXED), uses SBR or RBR replication automatically to take advantage of the benefits of both SBR and RBR formats when appropriate. The different formats are discussed in Section 17.2.1, “Replication Formats”.
- Detailed information on the different configuration options and variables that apply to replication is provided in Section 17.1.6, “Replication and Binary Logging Options and Variables”.
- Once started, the replication process should require little administration or monitoring. However, for advice on common tasks that you may want to execute, see Section 17.1.7, “Common Replication Administration Tasks”.

17.1.1 Binary Log File Position Based Replication Configuration Overview

This section describes replication between MySQL servers based on the binary log file position method, where the MySQL instance operating as the master (the source of the database changes) writes updates and changes as “events” to the binary log. The information in the binary log is stored in different logging formats according to the database changes being recorded. Slaves are configured to read the binary log from the master and to execute the events in the binary log on the slave's local database.

Each slave receives a copy of the entire contents of the binary log. It is the responsibility of the slave to decide which statements in the binary log should be executed. Unless you specify otherwise, all events in the master binary log are executed on the slave. If required, you can configure the slave to process only events that apply to particular databases or tables.

Important

You cannot configure the master to log only certain events.

Each slave keeps a record of the binary log coordinates: the file name and position within the file that it has read and processed from the master. This means that multiple slaves can be connected to the master and executing different parts of the same binary log. Because the slaves control this process, individual slaves can be connected and disconnected from the server without affecting the master's operation. Also, because each slave records the current position within the binary log, it is possible for slaves to be disconnected, reconnect and then resume processing.

The master and each slave must be configured with a unique ID (using the `server-id` option). In addition, each slave must be configured with information about the master host name, log file name, and position within that file. These details can be controlled from within a MySQL session using the `CHANGE`

`MASTER TO` statement on the slave. The details are stored within the slave's master info repository (see Section 17.2.4, “Replication Relay and Status Logs”).

17.1.2 Setting Up Binary Log File Position Based Replication

This section describes how to set up a MySQL server to use binary log file position based replication. There are a number of different methods for setting up replication, and the exact method to use depends on how you are setting up replication, and whether you already have data within your master database.

There are some generic tasks that are common to all setups:

- On the master, you must ensure that binary logging is enabled, and configure a unique server ID. This might require a server restart. See Section 17.1.2.1, “Setting the Replication Master Configuration”.
- On each slave that you want to connect to the master, you must configure a unique server ID. This might require a server restart. See Section 17.1.2.2, “Setting the Replication Slave Configuration”.
- Optionally, create a separate user for your slaves to use during authentication with the master when reading the binary log for replication. See Section 17.1.2.3, “Creating a User for Replication”.
- Before creating a data snapshot or starting the replication process, on the master you should record the current position in the binary log. You need this information when configuring the slave so that the slave knows where within the binary log to start executing events. See Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”.
- If you already have data on the master and want to use it to synchronize the slave, you need to create a data snapshot to copy the data to the slave. The storage engine you are using has an impact on how you create the snapshot. When you are using `MyISAM`, you must stop processing statements on the master to obtain a read-lock, then obtain its current binary log coordinates and dump its data, before permitting the master to continue executing statements. If you do not stop the execution of statements, the data dump and the master status information will not match, resulting in inconsistent or corrupted databases on the slaves. For more information on replicating a `MyISAM` master, see Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”. If you are using `InnoDB`, you do not need a read-lock and a transaction that is long enough to transfer the data snapshot is sufficient. For more information, see Section 15.18, “InnoDB and MySQL Replication”.
- Configure the slave with settings for connecting to the master, such as the host name, login credentials, and binary log file name and position. See Section 17.1.2.7, “Setting the Master Configuration on the Slave”.
- Implement replication-specific security measures on the masters and slaves as appropriate for your system. See Section 17.3, “Replication Security”.

Note

Certain steps within the setup process require the `SUPER` privilege. If you do not have this privilege, it might not be possible to enable replication.

After configuring the basic options, select your scenario:

- To set up replication for a fresh installation of a master and slaves that contain no data, see Setting Up Replication with New Master and Slaves.
- To set up replication of a new master using the data from an existing MySQL server, see Setting Up Replication with Existing Data.
- To add replication slaves to an existing replication environment, see Section 17.1.2.8, “Adding Slaves to a Replication Environment”.

Before administering MySQL replication servers, read this entire chapter and try all statements mentioned in Section 13.4.1, “SQL Statements for Controlling Master Servers”, and Section 13.4.2, “SQL Statements for Controlling Slave Servers”. Also familiarize yourself with the replication startup options described in Section 17.1.6, “Replication and Binary Logging Options and Variables”.

17.1.2.1 Setting the Replication Master Configuration

To configure a master to use binary log file position based replication, you must ensure that binary logging is enabled, and establish a unique server ID. If this has not already been done, a server restart is required.

Binary logging is required on the master because the binary log is the basis for replicating changes from the master to its slaves. Binary logging is enabled by default (the `log_bin` system variable is set to ON). The `--log-bin` option tells the server what base name to use for binary log files. It is recommended that you specify this option to give the binary log files a non-default base name, so that if the host name changes, you can easily continue to use the same binary log file names (see Section B.4.7, “Known Issues in MySQL”).

Each server within a replication topology must be configured with a unique server ID, which you can specify using the `--server-id` option. This server ID is used to identify individual servers within the replication topology, and must be a positive integer between 1 and (2^{32})−1. If you set a server ID of 0 on a master, it refuses any connections from slaves, and if you set a server ID of 0 on a slave, it refuses to connect to a master. Other than that, how you organize and select the numbers is your choice, so long as each server ID is different from every other server ID in use by any other server in the replication topology. The `server_id` system variable is set to 1 by default. A server can be started with this default server ID, but an informational message is issued if you did not specify a server ID explicitly.

Note

The following options also have an impact on the replication master:

- For the greatest possible durability and consistency in a replication setup using `InnoDB` with transactions, you should use `innodb_flush_log_at_trx_commit=1` and `sync_binlog=1` in the replication master's `my.cnf` file.
- Ensure that the `skip-networking` option is not enabled on the replication master. If networking has been disabled, the slave cannot communicate with the master and replication fails.

17.1.2.2 Setting the Replication Slave Configuration

Each replication slave must have a unique server ID. If this has not already been done, this part of slave setup requires a server restart.

If the slave server ID is not already set, or the current value conflicts with the value that you have chosen for the master server, shut down the slave server and edit the `[mysqld]` section of the configuration file to specify a unique server ID. For example:

```
[mysqld]
server-id=2
```

After making the changes, restart the server.

If you are setting up multiple slaves, each one must have a unique nonzero `server-id` value that differs from that of the master and from any of the other slaves.

Binary logging is enabled by default on all servers. A slave is not required to have binary logging enabled for replication to take place. However, binary logging on a slave means that the slave's binary log can be used for data backups and crash recovery.

Slaves that have binary logging enabled can also be used as part of a more complex replication topology. For example, you might want to set up replication servers using this chained arrangement:

```
A -> B -> C
```

Here, `A` serves as the master for the slave `B`, and `B` serves as the master for the slave `C`. For this to work, `B` must be both a master *and* a slave. Updates received from `A` must be logged by `B` to its binary log, in order to be passed on to `C`. In addition to binary logging, this replication topology requires the `log_slave_updates` system variable to be enabled. With slave updates enabled, the slave writes updates that are received from a master server and performed by the slave's SQL thread to the slave's own binary log. `log_slave_updates` system variable is enabled by default.

If you need to disable binary logging or slave update logging on a slave server, you can do this by specifying the `--skip-log-bin` and `--log-slave-updates=OFF` options for the slave.

17.1.2.3 Creating a User for Replication

Each slave connects to the master using a MySQL user name and password, so there must be a user account on the master that the slave can use to connect. The user name is specified by the `MASTER_USER` option on the `CHANGE MASTER TO` command when you set up a replication slave. Any account can be used for this operation, providing it has been granted the `REPLICATION SLAVE` privilege. You can choose to create a different account for each slave, or connect to the master using the same account for each slave.

Although you do not have to create an account specifically for replication, you should be aware that the replication user name and password are stored in plain text in the master info repository table `mysql.slave_master_info` (see Section 17.2.4.2, “Slave Status Logs”). Therefore, you may want to create a separate account that has privileges only for the replication process, to minimize the possibility of compromise to other accounts.

To create a new account, use `CREATE USER`. To grant this account the privileges required for replication, use the `GRANT` statement. If you create an account solely for the purposes of replication, that account needs only the `REPLICATION SLAVE` privilege. For example, to set up a new user, `repl`, that can connect for replication from any host within the `example.com` domain, issue these statements on the master:

```
mysql> CREATE USER 'repl'@'%.example.com' IDENTIFIED BY 'password';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%.example.com';
```

See Section 13.7.1, “Account Management Statements”, for more information on statements for manipulation of user accounts.

Important

To connect to the replication master using a user account that authenticates with the `caching_sha2_password` plugin, you must either set up a secure connection as described in Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”, or enable the unencrypted connection to support password exchange using an RSA key pair. The `caching_sha2_password` authentication plugin is the default for new users created from MySQL 8.0 (for details, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”). If the user account that you create or use for replication (as specified by the `MASTER_USER`

option) uses this authentication plugin, and you are not using a secure connection, you must enable RSA key pair-based password exchange for a successful connection.

17.1.2.4 Obtaining the Replication Master Binary Log Coordinates

To configure the slave to start the replication process at the correct point, you need to note the master's current coordinates within its binary log.

Warning

This procedure uses `FLUSH TABLES WITH READ LOCK`, which blocks `COMMIT` operations for `InnoDB` tables.

If you are planning to shut down the master to create a data snapshot, you can optionally skip this procedure and instead store a copy of the binary log index file along with the data snapshot. In that situation, the master creates a new binary log file on restart. The master binary log coordinates where the slave must start the replication process are therefore the start of that new file, which is the next binary log file on the master following after the files that are listed in the copied binary log index file.

To obtain the master binary log coordinates, follow these steps:

1. Start a session on the master by connecting to it with the command-line client, and flush all tables and block write statements by executing the `FLUSH TABLES WITH READ LOCK` statement:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   ```

   Warning

   Leave the client from which you issued the `FLUSH TABLES` statement running so that the read lock remains in effect. If you exit the client, the lock is released.

2. In a different session on the master, use the `SHOW MASTER STATUS` statement to determine the current binary log file name and position:

   ```
   mysql > SHOW MASTER STATUS;
   +------------------+----------+--------------+------------------+
   | File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
   +------------------+----------+--------------+------------------+
   | mysql-bin.000003 | 73       | test         | manual,mysql     |
   +------------------+----------+--------------+------------------+
   ```

   The `File` column shows the name of the log file and the `Position` column shows the position within the file. In this example, the binary log file is `mysql-bin.000003` and the position is 73. Record these values. You need them later when you are setting up the slave. They represent the replication coordinates at which the slave should begin processing new updates from the master.

   If the master has been running previously with binary logging disabled, the log file name and position values displayed by `SHOW MASTER STATUS` or `mysqldump --master-data` will be empty. In that case, the values that you need to use later when specifying the slave's log file and position are the empty string (`''`) and `4`.

You now have the information you need to enable the slave to start reading from the binary log in the correct place to start replication.

The next step depends on whether you have existing data on the master. Choose one of the following options:

- If you have existing data that needs be to synchronized with the slave before you start replication, leave the client running so that the lock remains in place. This prevents any further changes being made, so that the data copied to the slave is in synchrony with the master. Proceed to Section 17.1.2.5, “Choosing a Method for Data Snapshots”.
- If you are setting up a new master and slave replication group, you can exit the first session to release the read lock. See Setting Up Replication with New Master and Slaves for how to proceed.

17.1.2.5 Choosing a Method for Data Snapshots

If the master database contains existing data it is necessary to copy this data to each slave. There are different ways to dump the data from the master database. The following sections describe possible options.

To select the appropriate method of dumping the database, choose between these options:

- Use the `mysqldump` tool to create a dump of all the databases you want to replicate. This is the recommended method, especially when using `InnoDB`.
- If your database is stored in binary portable files, you can copy the raw data files to a slave. This can be more efficient than using `mysqldump` and importing the file on each slave, because it skips the overhead of updating indexes as the `INSERT` statements are replayed. With storage engines such as `InnoDB` this is not recommended.

Creating a Data Snapshot Using mysqldump

To create a snapshot of the data in an existing master database, use the `mysqldump` tool. Once the data dump has been completed, import this data into the slave before starting the replication process.

The following example dumps all databases to a file named `dbdump.db`, and includes the `--master-data` option which automatically appends the `CHANGE MASTER TO` statement required on the slave to start the replication process:

```
shell> mysqldump --all-databases --master-data > dbdump.db
```

Note

If you do not use `--master-data`, then it is necessary to lock all tables in a separate session manually. See Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”.

It is possible to exclude certain databases from the dump using the `mysqldump` tool. If you want to choose which databases to include in the dump, do not use `--all-databases`. Choose one of these options:

- Exclude all the tables in the database using `--ignore-table` option.
- Name only those databases which you want dumped using the `--databases` option.

Note

By default, if GTIDs are in use on the master (`gtid_mode=ON`), `mysqldump` includes the GTIDs from the `gtid_executed` set on the master in the dump output to add them to the `gtid_purged` set on the slave. If you are dumping only specific databases or tables, it is important to note that the value that is included by `mysqldump` includes the GTIDs of all transactions in the `gtid_executed` set on the master, even those that changed suppressed parts of the database, or other databases on the server that were not included in the partial dump. Check

> the description for mysqldump's `--set-gtid-purged` option to find the outcome of the default behavior for the MySQL Server versions you are using, and how to change the behavior if this outcome is not suitable for your situation.

For more information, see Section 4.5.4, “`mysqldump` — A Database Backup Program”.

To import the data, either copy the dump file to the slave, or access the file from the master when connecting remotely to the slave.

Creating a Data Snapshot Using Raw Data Files

This section describes how to create a data snapshot using the raw files which make up the database. Employing this method with a table using a storage engine that has complex caching or logging algorithms requires extra steps to produce a perfect “point in time” snapshot: the initial copy command could leave out cache information and logging updates, even if you have acquired a global read lock. How the storage engine responds to this depends on its crash recovery abilities.

If you use `InnoDB` tables, you can use the `mysqlbackup` command from the MySQL Enterprise Backup component to produce a consistent snapshot. This command records the log name and offset corresponding to the snapshot to be used on the slave. MySQL Enterprise Backup is a commercial product that is included as part of a MySQL Enterprise subscription. See Section 30.2, “MySQL Enterprise Backup Overview” for detailed information.

This method also does not work reliably if the master and slave have different values for `ft_stopword_file`, `ft_min_word_len`, or `ft_max_word_len` and you are copying tables having full-text indexes.

Assuming the above exceptions do not apply to your database, use the cold backup technique to obtain a reliable binary snapshot of `InnoDB` tables: do a slow shutdown of the MySQL Server, then copy the data files manually.

To create a raw data snapshot of `MyISAM` tables when your MySQL data files exist on a single file system, you can use standard file copy tools such as `cp` or `copy`, a remote copy tool such as `scp` or `rsync`, an archiving tool such as `zip` or `tar`, or a file system snapshot tool such as `dump`. If you are replicating only certain databases, copy only those files that relate to those tables. For `InnoDB`, all tables in all databases are stored in the system tablespace files, unless you have the `innodb_file_per_table` option enabled.

The following files are not required for replication:

- Files relating to the `mysql` database.
- The master info repository file `master.info`, if used; the use of this file is now deprecated (see Section 17.2.4, “Replication Relay and Status Logs”).
- The master's binary log files, with the exception of the binary log index file if you are going to use this to locate the master binary log coordinates for the slave.
- Any relay log files.

Depending on whether you are using `InnoDB` tables or not, choose one of the following:

If you are using `InnoDB` tables, and also to get the most consistent results with a raw data snapshot, shut down the master server during the process, as follows:

1. Acquire a read lock and get the master's status. See Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”.
2. In a separate session, shut down the master server:

```
shell> mysqladmin shutdown
```

3. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

4. Restart the master server.

If you are not using `InnoDB` tables, you can get a snapshot of the system from a master without shutting down the server as described in the following steps:

1. Acquire a read lock and get the master's status. See Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”.

2. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

```
shell> tar cf /tmp/db.tar ./data
shell> zip -r /tmp/db.zip ./data
shell> rsync --recursive ./data /tmp/dbdata
```

3. In the client where you acquired the read lock, release the lock:

```
mysql> UNLOCK TABLES;
```

Once you have created the archive or copy of the database, copy the files to each slave before starting the slave replication process.

17.1.2.6 Setting Up Replication Slaves

The following sections describe how to set up slaves. Before you proceed, ensure that you have:

- Configured the MySQL master with the necessary configuration properties. See Section 17.1.2.1, “Setting the Replication Master Configuration”.
- Obtained the master status information, or a copy of the master's binary log index file made during a shutdown for the data snapshot. See Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”.
- On the master, released the read lock:

```
mysql> UNLOCK TABLES;
```

- On the slave, edited the MySQL configuration. See Section 17.1.2.2, “Setting the Replication Slave Configuration”.

The next steps depend on whether you have existing data to import to the slave or not. See Section 17.1.2.5, “Choosing a Method for Data Snapshots” for more information. Choose one of the following:

- If you do not have a snapshot of a database to import, see Setting Up Replication with New Master and Slaves.

- If you have a snapshot of a database to import, see Setting Up Replication with Existing Data.

Setting Up Replication with New Master and Slaves

When there is no snapshot of a previous database to import, configure the slave to start the replication from the new master.

To set up replication between a master and a new slave:

1. Start up the MySQL slave.
2. Execute a `CHANGE MASTER TO` statement to set the master replication server configuration. See Section 17.1.2.7, “Setting the Master Configuration on the Slave”.

Perform these slave setup steps on each slave.

This method can also be used if you are setting up new servers but have an existing dump of the databases from a different server that you want to load into your replication configuration. By loading the data into a new master, the data is automatically replicated to the slaves.

If you are setting up a new replication environment using the data from a different existing database server to create a new master, run the dump file generated from that server on the new master. The database updates are automatically propagated to the slaves:

```
shell> mysql -h master < fulldb.dump
```

Setting Up Replication with Existing Data

When setting up replication with existing data, transfer the snapshot from the master to the slave before starting replication. The process for importing data to the slave depends on how you created the snapshot of data on the master.

Choose one of the following:

If you used `mysqldump`:

1. Start the slave, using the `--skip-slave-start` option so that replication does not start.
2. Import the dump file:

   ```
   shell> mysql < fulldb.dump
   ```

If you created a snapshot using the raw data files:

1. Extract the data files into your slave data directory. For example:

   ```
   shell> tar xvf dbdump.tar
   ```

   You may need to set permissions and ownership on the files so that the slave server can access and modify them.
2. Start the slave, using the `--skip-slave-start` option so that replication does not start.
3. Configure the slave with the replication coordinates from the master. This tells the slave the binary log file and position within the file where replication needs to start. Also, configure the slave with the login credentials and host name of the master. For more information on the `CHANGE MASTER TO` statement required, see Section 17.1.2.7, “Setting the Master Configuration on the Slave”.

4. Start the slave threads:

```
mysql> START SLAVE;
```

After you have performed this procedure, the slave connects to the master and replicates any updates that have occurred on the master since the snapshot was taken. Error messages are issued to the slave's error log if it is not able to replicate for any reason.

The slave uses information logged in its master info log and relay log info log to keep track of how much of the master's binary log it has processed. From MySQL 8.0, by default, the repositories for these slave status logs are tables named `slave_master_info` and `slave_relay_log_info` in the `mysql` database. The alternative settings `--master-info-repository=FILE` and `--relay-log-info-repository=FILE`, where the repositories are files named `master.info` and `relay-log.info` in the data directory, are now deprecated and will be removed in a future release.

Do *not* remove or edit these tables (or files, if used) unless you know exactly what you are doing and fully understand the implications. Even in that case, it is preferred that you use the `CHANGE MASTER TO` statement to change replication parameters. The slave uses the values specified in the statement to update the slave status logs automatically. See Section 17.2.4, “Replication Relay and Status Logs”, for more information.

Note

The contents of the master info log override some of the server options specified on the command line or in `my.cnf`. See Section 17.1.6, “Replication and Binary Logging Options and Variables”, for more details.

A single snapshot of the master suffices for multiple slaves. To set up additional slaves, use the same master snapshot and follow the slave portion of the procedure just described.

17.1.2.7 Setting the Master Configuration on the Slave

To set up the slave to communicate with the master for replication, configure the slave with the necessary connection information. To do this, execute the following statement on the slave, replacing the option values with the actual values relevant to your system:

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master_host_name',
    ->     MASTER_USER='replication_user_name',
    ->     MASTER_PASSWORD='replication_password',
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;
```

Note

Replication cannot use Unix socket files. You must be able to connect to the master MySQL server using TCP/IP.

The `CHANGE MASTER TO` statement has other options as well. For example, it is possible to set up secure replication using SSL. For a full list of options, and information about the maximum permissible length for the string-valued options, see Section 13.4.2.1, “CHANGE MASTER TO Syntax”.

Important

As noted in Section 17.1.2.3, “Creating a User for Replication”, if you are not using a secure connection and the user account named in the `MASTER_USER` option authenticates with the `caching_sha2_password` plugin (the default

from MySQL 8.0), you must specify the `MASTER_PUBLIC_KEY_PATH` or `GET_MASTER_PUBLIC_KEY` option in the `CHANGE MASTER TO` statement to enable RSA key pair-based password exchange.

17.1.2.8 Adding Slaves to a Replication Environment

You can add another slave to an existing replication configuration without stopping the master. To do this, you can set up the new slave by copying the data directory of an existing slave, and giving the new slave a different server ID (which is user-specified) and server UUID (which is generated at startup).

To duplicate an existing slave:

1. Stop the existing slave and record the slave status information, particularly the master binary log file and relay log file positions. You can view the slave status either in the Performance Schema replication tables (see Section 26.12.11, “Performance Schema Replication Tables”), or by issuing `SHOW SLAVE STATUS` as follows:

```
mysql> STOP SLAVE;
mysql> SHOW SLAVE STATUS\G
```

2. Shut down the existing slave:

```
shell> mysqladmin shutdown
```

3. Copy the data directory from the existing slave to the new slave, including the log files and relay log files. You can do this by creating an archive using `tar` or `WinZip`, or by performing a direct copy using a tool such as `cp` or `rsync`.

Important

- Before copying, verify that all the files relating to the existing slave actually are stored in the data directory. For example, the `InnoDB` system tablespace, undo tablespace, and redo log might be stored in an alternative location. `InnoDB` tablespace files and file-per-table tablespaces might have been created in other directories. The binary logs and relay logs for the slave might be in their own directories outside the data directory. Check through the system variables that are set for the existing slave and look for any alternative paths that have been specified. If you find any, copy these directories over as well.
- During copying, if files have been used for the master info and relay log info repositories (see Section 17.2.4, “Replication Relay and Status Logs”), ensure that you also copy these files from the existing slave to the new slave. If tables have been used for the repositories, which is the default from MySQL 8.0, the tables are in the data directory.
- After copying, delete the `auto.cnf` file from the copy of the data directory on the new slave, so that the new slave is started with a different generated server UUID. The server UUID must be unique.

A common problem that is encountered when adding new replication slaves is that the new slave fails with a series of warning and error messages like these:

```
071118 16:44:10 [Warning] Neither --relay-log nor --relay-log-index were used; so
replication may break when this MySQL server acts as a slave and has his hostname
```

```
changed!! Please use '--relay-log=new_slave_hostname-relay-bin' to avoid this problem.
071118 16:44:10 [ERROR] Failed to open the relay log './old_slave_hostname-relay-bin.003525'
(relay_log_pos 22940879)
071118 16:44:10 [ERROR] Could not find target log during relay log initialization
071118 16:44:10 [ERROR] Failed to initialize the master info structure
```

This situation can occur if the `relay_log` system variable is not specified, as the relay log files contain the host name as part of their file names. This is also true of the relay log index file if the `relay_log_index` system variable is not used. For more information about these variables, see Section 17.1.6, “Replication and Binary Logging Options and Variables”.

To avoid this problem, use the same value for `relay_log` on the new slave that was used on the existing slave. If this option was not set explicitly on the existing slave, use *`existing_slave_hostname`*`-relay-bin`. If this is not possible, copy the existing slave's relay log index file to the new slave and set the `relay_log_index` system variable on the new slave to match what was used on the existing slave. If this option was not set explicitly on the existing slave, use *`existing_slave_hostname`*`-relay-bin.index`. Alternatively, if you have already tried to start the new slave after following the remaining steps in this section and have encountered errors like those described previously, then perform the following steps:

a. If you have not already done so, issue `STOP SLAVE` on the new slave.

   If you have already started the existing slave again, issue `STOP SLAVE` on the existing slave as well.

b. Copy the contents of the existing slave's relay log index file into the new slave's relay log index file, making sure to overwrite any content already in the file.

c. Proceed with the remaining steps in this section.

4. When copying is complete, restart the existing slave.

5. On the new slave, edit the configuration and give the new slave a unique server ID (using the `server-id` option) that is not used by the master or any of the existing slaves.

6. Start the new slave server, specifying the `--skip-slave-start` option so that replication does not start yet. Use the Performance Schema replication tables or issue `SHOW SLAVE STATUS` to confirm that the new slave has the correct settings when compared with the existing slave. Also display the server ID and server UUID and verify that these are correct and unique for the new slave.

7. Start the slave threads by issuing a `START SLAVE` statement:

```
mysql> START SLAVE;
```

   The new slave now uses the information in its master info repository to start the replication process.

17.1.3 Replication with Global Transaction Identifiers

This section explains transaction-based replication using *global transaction identifiers* (GTIDs). When using GTIDs, each transaction can be identified and tracked as it is committed on the originating server and applied by any slaves; this means that it is not necessary when using GTIDs to refer to log files or positions within those files when starting a new slave or failing over to a new master, which greatly simplifies these tasks. Because GTID-based replication is completely transaction-based, it is simple to determine whether masters and slaves are consistent; as long as all transactions committed on a master are also committed on a slave, consistency between the two is guaranteed. You can use either statement-based or row-based replication with GTIDs (see Section 17.2.1, “Replication Formats”); however, for best results, we recommend that you use the row-based format.

GTIDs are always preserved between master and slave. This means that you can always determine the source for any transaction applied on any slave by examining its binary log. In addition, once a transaction with a given GTID is committed on a given server, any subsequent transaction having the same GTID is ignored by that server. Thus, a transaction committed on the master can be applied no more than once on the slave, which helps to guarantee consistency.

This section discusses the following topics:

- How GTIDs are defined and created, and how they are represented in a MySQL server (see Section 17.1.3.1, “GTID Format and Storage”).
- The life cycle of a GTID (see Section 17.1.3.2, “GTID Life Cycle”).
- The auto-positioning function for synchronizing a slave and master that use GTIDs (see Section 17.1.3.3, “GTID Auto-Positioning”).
- A general procedure for setting up and starting GTID-based replication (see Section 17.1.3.4, “Setting Up Replication Using GTIDs”).
- Suggested methods for provisioning new replication servers when using GTIDs (see Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”).
- Restrictions and limitations that you should be aware of when using GTID-based replication (see Section 17.1.3.6, “Restrictions on Replication with GTIDs”).
- Stored functions that you can use to work with GTIDs (see Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”).

For information about MySQL Server options and variables relating to GTID-based replication, see Section 17.1.6.5, “Global Transaction ID Options and Variables”. See also Section 12.18, “Functions Used with Global Transaction Identifiers (GTIDs)”, which describes SQL functions supported by MySQL 8.0 for use with GTIDs.

17.1.3.1 GTID Format and Storage

A global transaction identifier (GTID) is a unique identifier created and associated with each transaction committed on the server of origin (the master). This identifier is unique not only to the server on which it originated, but is unique across all servers in a given replication topology.

GTID assignment distinguishes between client transactions, which are committed on the master, and replicated transactions, which are reproduced on a slave. When a client transaction is committed on the master, it is assigned a new GTID, provided that the transaction was written to the binary log. Client transactions are guaranteed to have monotonically increasing GTIDs without gaps between the generated numbers. If a client transaction is not written to the binary log (for example, because the transaction was filtered out, or the transaction was read-only), it is not assigned a GTID on the server of origin.

Replicated transactions retain the same GTID that was assigned to the transaction on the server of origin. The GTID is present before the replicated transaction begins to execute, and is persisted even if the replicated transaction is not written to the binary log on the slave, or is filtered out on the slave. The MySQL system table `mysql.gtid_executed` is used to preserve the assigned GTIDs of all the transactions applied on a MySQL server, except those that are stored in a currently active binary log file.

The auto-skip function for GTIDs means that a transaction committed on the master can be applied no more than once on the slave, which helps to guarantee consistency. Once a transaction with a given GTID has been committed on a given server, any attempt to execute a subsequent transaction with the same GTID is ignored by that server. No error is raised, and no statement in the transaction is executed.

If a transaction with a given GTID has started to execute on a server, but has not yet committed or rolled back, any attempt to start a concurrent transaction on the server with the same GTID will block. The server neither begins to execute the concurrent transaction nor returns control to the client. Once the first attempt at the transaction commits or rolls back, concurrent sessions that were blocking on the same GTID may proceed. If the first attempt rolled back, one concurrent session proceeds to attempt the transaction, and any other concurrent sessions that were blocking on the same GTID remain blocked. If the first attempt committed, all the concurrent sessions stop being blocked, and auto-skip all the statements of the transaction.

A GTID is represented as a pair of coordinates, separated by a colon character (`:`), as shown here:

```
GTID = source_id:transaction_id
```

The *`source_id`* identifies the originating server. Normally, the master's `server_uuid` is used for this purpose. The *`transaction_id`* is a sequence number determined by the order in which the transaction was committed on the master. For example, the first transaction to be committed has `1` as its *`transaction_id`*, and the tenth transaction to be committed on the same originating server is assigned a *`transaction_id`* of `10`. It is not possible for a transaction to have `0` as a sequence number in a GTID. For example, the twenty-third transaction to be committed originally on the server with the UUID `3E11FA47-71CA-11E1-9E33-C80AA9429562` has this GTID:

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:23
```

The GTID for a transaction is shown in the output from `mysqlbinlog`, and it is used to identify an individual transaction in the Performance Schema replication status tables, for example, `replication_applier_status_by_worker`. The value stored by the `gtid_next` system variable (`@@GLOBAL.gtid_next`) is a single GTID.

GTID Sets

A GTID set is a set comprising one or more single GTIDs or ranges of GTIDs. GTID sets are used in a MySQL server in several ways. For example, the values stored by the `gtid_executed` and `gtid_purged` system variables are GTID sets. The `START SLAVE` clauses `UNTIL SQL_BEFORE_GTIDS` and `UNTIL SQL_AFTER_GTIDS` can be used to make a slave process transactions only up to the first GTID in a GTID set, or stop after the last GTID in a GTID set. The built-in functions `GTID_SUBSET()` and `GTID_SUBTRACT()` require GTID sets as input.

A range of GTIDs originating from the same server can be collapsed into a single expression, as shown here:

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5
```

The above example represents the first through fifth transactions originating on the MySQL server whose `server_uuid` is `3E11FA47-71CA-11E1-9E33-C80AA9429562`. Multiple single GTIDs or ranges of GTIDs originating from the same server can also be included in a single expression, with the GTIDs or ranges separated by colons, as in the following example:

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:1-3:11:47-49
```

A GTID set can include any combination of single GTIDs and ranges of GTIDs, and it can include GTIDs originating from different servers. This example shows the GTID set stored in the `gtid_executed` system variable (`@@GLOBAL.gtid_executed`) of a slave that has applied transactions from more than one master:

```
2174B383-5441-11E8-B90A-C80AA9429562:1-3, 24DA167-0C0C-11E8-8442-00059A3C7B00:1-19
```

When GTID sets are returned from server variables, UUIDs are in alphabetical order, and numeric intervals are merged and in ascending order.

The syntax for a GTID set is as follows:

```
gtid_set:
    uuid_set [, uuid_set] ...
    | ''

uuid_set:
    uuid:interval[:interval]...

uuid:
    hhhhhhhh-hhhh-hhhh-hhhh-hhhhhhhhhhhh

h:
    [0-9|A-F]

interval:
    n[-n]

    (n >= 1)
```

mysql.gtid_executed Table

GTIDs are stored in a table named `gtid_executed`, in the `mysql` database. A row in this table contains, for each GTID or set of GTIDs that it represents, the UUID of the originating server, and the starting and ending transaction IDs of the set; for a row referencing only a single GTID, these last two values are the same.

The `mysql.gtid_executed` table is created (if it does not already exist) when MySQL Server is installed or upgraded, using a `CREATE TABLE` statement similar to that shown here:

```
CREATE TABLE gtid_executed (
    source_uuid CHAR(36) NOT NULL,
    interval_start BIGINT(20) NOT NULL,
    interval_end BIGINT(20) NOT NULL,
    PRIMARY KEY (source_uuid, interval_start)
)
```

Warning

As with other MySQL system tables, do not attempt to create or modify this table yourself.

The `mysql.gtid_executed` table is provided for internal use by the MySQL server. It enables a slave to use GTIDs when binary logging is disabled on the slave, and it enables retention of the GTID state when the binary logs have been lost. Note that the `mysql.gtid_executed` table is cleared if you issue `RESET MASTER`.

GTIDs are stored in the `mysql.gtid_executed` table only when `gtid_mode` is `ON` or `ON_PERMISSIVE`. The point at which GTIDs are stored depends on whether binary logging is enabled or disabled:

- If binary logging is disabled (`log_bin` is `OFF`), or if `log_slave_updates` is disabled, the server stores the GTID belonging to each transaction together with the transaction in the table. In addition, the table is compressed periodically at a user-configurable rate; see mysql.gtid_executed Table Compression,

for more information. This situation can only apply on a replication slave where binary logging or slave update logging is disabled. It does not apply on a replication master, because on a master, binary logging must be enabled for replication to take place.

- If binary logging is enabled (`log_bin` is `ON`), whenever the binary log is rotated or the server is shut down, the server writes GTIDs for all transactions that were written into the previous binary log into the `mysql.gtid_executed` table. This situation applies on a replication master, or a replication slave where binary logging is enabled.

  In the event of the server stopping unexpectedly, the set of GTIDs from the current binary log file is not saved in the `mysql.gtid_executed` table. These GTIDs are added to the table from the binary log file during recovery. The exception to this is if you disable binary logging when the server is restarted (using `--skip-log-bin` or `--disable-log-bin`). In this situation, the server cannot access the binary log file to recover the GTIDs, so replication cannot be started.

  When binary logging is enabled, the `mysql.gtid_executed` table does not hold a complete record of the GTIDs for all executed transactions. That information is provided by the global value of the `gtid_executed` system variable. Always use `@@GLOBAL.gtid_executed`, which is updated after every commit, to represent the GTID state for the MySQL server, and do not query the `mysql.gtid_executed` table.

The MySQL server can write to the `mysql.gtid_executed` table even when the server is in read only or super read only mode, so that the binary log file can still be rotated in these modes. If the `mysql.gtid_executed` table cannot be accessed for writes, and the binary log file is rotated for any reason other than reaching the maximum file size (`max_binlog_size`), the current binary log file continues to be used. An error message is returned to the client that requested the rotation, and a warning is logged on the server. If the `mysql.gtid_executed` table cannot be accessed for writes and `max_binlog_size` is reached, the server responds according to its `binlog_error_action` setting. If `IGNORE_ERROR` is set, an error is logged on the server and binary logging is halted, or if `ABORT_SERVER` is set, the server shuts down.

mysql.gtid_executed Table Compression

Over the course of time, the `mysql.gtid_executed` table can become filled with many rows referring to individual GTIDs that originate on the same server, and whose transaction IDs make up a range, similar to what is shown here:

```
+--------------------------------------+----------------+--------------+
| source_uuid                          | interval_start | interval_end |
|--------------------------------------+----------------+--------------|
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 37             | 37           |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 38             | 38           |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 39             | 39           |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 40             | 40           |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 41             | 41           |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 42             | 42           |
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 43             | 43           |
...
```

To save space, the MySQL server compresses the `mysql.gtid_executed` table periodically by replacing each such set of rows with a single row that spans the entire interval of transaction identifiers, like this:

```
+--------------------------------------+----------------+--------------+
| source_uuid                          | interval_start | interval_end |
|--------------------------------------+----------------+--------------|
| 3E11FA47-71CA-11E1-9E33-C80AA9429562 | 37             | 43           |
```

```
...
```

You can control the number of transactions that are allowed to elapse before the table is compressed, and thus the compression rate, by setting the `gtid_executed_compression_period` system variable. This variable's default value is 1000, meaning that by default, compression of the table is performed after each 1000 transactions. Setting `gtid_executed_compression_period` to 0 prevents the compression from being performed at all, and you should be prepared for a potentially large increase in the amount of disk space that may be required by the `gtid_executed` table if you do this.

Note

When binary logging is enabled, the value of `gtid_executed_compression_period` is *not* used and the `mysql.gtid_executed` table is compressed on each binary log rotation.

Compression of the `mysql.gtid_executed` table is performed by a dedicated foreground thread named `thread/sql/compress_gtid_table`. This thread is not listed in the output of `SHOW PROCESSLIST`, but it can be viewed as a row in the `threads` table, as shown here:

```
mysql> SELECT * FROM performance_schema.threads WHERE NAME LIKE '%gtid%'\G
*************************** 1. row ***************************
          THREAD_ID: 26
               NAME: thread/sql/compress_gtid_table
               TYPE: FOREGROUND
     PROCESSLIST_ID: 1
   PROCESSLIST_USER: NULL
   PROCESSLIST_HOST: NULL
     PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: Daemon
   PROCESSLIST_TIME: 1509
  PROCESSLIST_STATE: Suspending
   PROCESSLIST_INFO: NULL
   PARENT_THREAD_ID: 1
               ROLE: NULL
       INSTRUMENTED: YES
            HISTORY: YES
    CONNECTION_TYPE: NULL
       THREAD_OS_ID: 18677
```

The `thread/sql/compress_gtid_table` thread normally sleeps until `gtid_executed_compression_period` transactions have been executed, then wakes up to perform compression of the `mysql.gtid_executed` table as described previously. It then sleeps until another `gtid_executed_compression_period` transactions have taken place, then wakes up to perform the compression again, repeating this loop indefinitely. Setting this value to 0 when binary logging is disabled means that the thread always sleeps and never wakes up.

17.1.3.2 GTID Life Cycle

The life cycle of a GTID consists of the following steps:

1. A transaction is executed and committed on the master. This client transaction is assigned a GTID composed of the master's UUID and the smallest nonzero transaction sequence number not yet used on this server. The GTID is written to the master's binary log (immediately preceding the transaction itself in the log). If a client transaction is not written to the binary log (for example, because the transaction was filtered out, or the transaction was read-only), it is not assigned a GTID.

2. If a GTID was assigned for the transaction, the GTID is persisted atomically at commit time by writing it to the binary log at the beginning of the transaction (as a `Gtid_log_event`). Whenever the binary log

is rotated or the server is shut down, the server writes GTIDs for all transactions that were written into the previous binary log file into the `mysql.gtid_executed` table.

3. If a GTID was assigned for the transaction, the GTID is externalized non-atomically (very shortly after the transaction is committed) by adding it to the set of GTIDs in the `gtid_executed` system variable (`@@GLOBAL.gtid_executed`). This GTID set contains a representation of the set of all committed GTID transactions, and it is used in replication as a token that represents the server state. With binary logging enabled (as required for the master), the set of GTIDs in the `gtid_executed` system variable is a complete record of the transactions applied, but the `mysql.gtid_executed` table is not, because the most recent history is still in the current binary log file.

4. After the binary log data is transmitted to the slave and stored in the slave's relay log (using established mechanisms for this process, see Section 17.2, “Replication Implementation”, for details), the slave reads the GTID and sets the value of its `gtid_next` system variable as this GTID. This tells the slave that the next transaction must be logged using this GTID. It is important to note that the slave sets `gtid_next` in a session context.

5. The slave verifies that no thread has yet taken ownership of the GTID in `gtid_next` in order to process the transaction. By reading and checking the replicated transaction's GTID first, before processing the transaction itself, the slave guarantees not only that no previous transaction having this GTID has been applied on the slave, but also that no other session has already read this GTID but has not yet committed the associated transaction. So if multiple clients attempt to apply the same transaction concurrently, the server resolves this by letting only one of them execute. The `gtid_owned` system variable (`@@GLOBAL.gtid_owned`) for the slave shows each GTID that is currently in use and the ID of the thread that owns it. If the GTID has already been used, no error is raised, and the auto-skip function is used to ignore the transaction.

6. If the GTID has not been used, the slave applies the replicated transaction. Because `gtid_next` is set to the GTID already assigned by the master, the slave does not attempt to generate a new GTID for this transaction, but instead uses the GTID stored in `gtid_next`.

7. If binary logging is enabled on the slave, the GTID is persisted atomically at commit time by writing it to the binary log at the beginning of the transaction (as a `Gtid_log_event`). Whenever the binary log is rotated or the server is shut down, the server writes GTIDs for all transactions that were written into the previous binary log file into the `mysql.gtid_executed` table.

8. If binary logging is disabled on the slave, the GTID is persisted atomically by writing it directly into the `mysql.gtid_executed` table. MySQL appends a statement to the transaction to insert the GTID into the table. From MySQL 8.0, this operation is atomic for DDL statements as well as for DML statements. In this situation, the `mysql.gtid_executed` table is a complete record of the transactions applied on the slave.

9. Very shortly after the replicated transaction is committed on the slave, the GTID is externalized non-atomically by adding it to the set of GTIDs in the `gtid_executed` system variable (`@@GLOBAL.gtid_executed`) for the slave. As for the master, this GTID set contains a representation of the set of all committed GTID transactions. If binary logging is disabled on the slave, the `mysql.gtid_executed` table is also a complete record of the transactions applied on the slave. If binary logging is enabled on the slave, meaning that some GTIDs are only recorded in the binary log, the set of GTIDs in the `gtid_executed` system variable is the only complete record.

Client transactions that are completely filtered out on the master are not assigned a GTID, therefore they are not added to the set of transactions in the `gtid_executed` system variable, or added to the `mysql.gtid_executed` table. However, the GTIDs of replicated transactions that are completely filtered out on the slave are persisted. If binary logging is enabled on the slave, the filtered-out transaction is written to the binary log as a `Gtid_log_event` followed by an empty transaction containing only `BEGIN` and `COMMIT` statements. If binary logging is disabled, the GTID of the filtered-out transaction is written to

the `mysql.gtid_executed` table. Preserving the GTIDs for filtered-out transactions ensures that the `mysql.gtid_executed` table and the set of GTIDs in the `gtid_executed` system variable can be compressed. It also ensures that the filtered-out transactions are not retrieved again if the slave reconnects to the master, as explained in Section 17.1.3.3, "GTID Auto-Positioning".

On a multithreaded replication slave (with `slave_parallel_workers > 0`), transactions can be applied in parallel, so replicated transactions can commit out of order (unless `slave_preserve_commit_order=1` is set). When that happens, the set of GTIDs in the `gtid_executed` system variable will contain multiple GTID ranges with gaps between them. (On a master or a single-threaded replication slave, there will be monotonically increasing GTIDs without gaps between the numbers.) Gaps on multithreaded replication slaves only occur among the most recently applied transactions, and are filled in as replication progresses. When replication threads are stopped cleanly using the `STOP SLAVE` statement, ongoing transactions are applied so that the gaps are filled in. In the event of a shutdown such as a server failure or the use of the `KILL` statement to stop replication threads, the gaps might remain.

What changes are assigned a GTID?

The typical scenario is that the server generates a new GTID for a committed transaction. However, GTIDs can also be assigned to other changes besides transactions, and in some cases a single transaction can be assigned multiple GTIDs.

Every database change (DDL or DML) that is written to the binary log is assigned a GTID. This includes changes that are autocommitted, and changes that are committed using `BEGIN` and `COMMIT` or `START TRANSACTION` statements. A GTID is also assigned to the creation, alteration, or deletion of a database, and of a non-table database object such as a procedure, function, trigger, event, view, user, role, or grant.

Non-transactional updates as well as transactional updates are assigned GTIDs. In addition, for a non-transactional update, if a disk write failure occurs while attempting to write to the binary log cache and a gap is therefore created in the binary log, the resulting incident log event is assigned a GTID.

When a table is automatically dropped by a generated statement in the binary log, a GTID is assigned to the statement. Temporary tables are dropped automatically when a replication slave begins to apply events from a master that has just been started, and when statement-based replication is in use (`binlog_format=STATEMENT`) and a user session that has open temporary tables disconnects. Tables that use the `MEMORY` storage engine are deleted automatically the first time they are accessed after the server is started, because rows might have been lost during the shutdown.

When a transaction is not written to the binary log on the server of origin, the server does not assign a GTID to it. This includes transactions that are rolled back and transactions that are executed while binary logging is disabled on the server of origin, either globally (with `--skip-log-bin` specified in the server's configuration) or for the session (`SET @@SESSION.sql_log_bin = 0`). This also includes no-op transactions when row-based replication is in use (`binlog_format=ROW`).

XA transactions are assigned separate GTIDs for the `XA PREPARE` phase of the transaction and the `XA COMMIT` or `XA ROLLBACK` phase of the transaction. XA transactions are persistently prepared so that users can commit them or roll them back in the case of a failure (which in a replication topology might include a failover to another server). The two parts of the transaction are therefore replicated separately, so they must have their own GTIDs, even though a non-XA transaction that is rolled back would not have a GTID.

In the following special cases, a single statement can generate multiple transactions, and therefore be assigned multiple GTIDs:

- A stored procedure is invoked that commits multiple transactions. One GTID is generated for each transaction that the procedure commits.

- A multi-table `DROP TABLE` statement drops tables of different types. Multiple GTIDs can be generated if any of the tables use storage engines that do not support atomic DDL, or if any of the tables are temporary tables.
- A `CREATE TABLE ... SELECT` statement is issued when row-based replication is in use (`binlog_format=ROW`). One GTID is generated for the `CREATE TABLE` action and one GTID is generated for the row-insert actions.

The `gtid_next` System Variable

By default, for new transactions committed in user sessions, the server automatically generates and assigns a new GTID. When the transaction is applied on a replication slave, the GTID from the server of origin is preserved. You can change this behavior by setting the session value of the `gtid_next` system variable:

- When `gtid_next` is set to `AUTOMATIC`, which is the default, and a transaction is committed and written to the binary log, the server automatically generates and assigns a new GTID. If a transaction is rolled back or not written to the binary log for another reason, the server does not generate and assign a GTID.
- If you set `gtid_next` to a valid GTID (consisting of a UUID and a transaction sequence number, separated by a colon), the server assigns that GTID to your transaction. This GTID is assigned and added to `gtid_executed` even when the transaction is not written to the binary log, or when the transaction is empty.

Note that after you set `gtid_next` to a specific GTID, and the transaction has been committed or rolled back, an explicit `SET @@SESSION.gtid_next` statement must be issued before any other statement. You can use this to set the GTID value back to `AUTOMATIC` if you do not want to assign any more GTIDs explicitly.

When replication applier threads apply replicated transactions, they use this technique, setting `@@SESSION.gtid_next` explicitly to the GTID of the replicated transaction as assigned on the server of origin. This means the GTID from the server of origin is retained, rather than a new GTID being generated and assigned by the replication slave. It also means the GTID is added to `gtid_executed` on the replication slave even when binary logging or slave update logging is disabled on the slave, or when the transaction is a no-op or is filtered out on the slave.

It is possible for a client to simulate a replicated transaction by setting `@@SESSION.gtid_next` to a specific GTID before executing the transaction. This technique is used by `mysqlbinlog` to generate a dump of the binary log that the client can replay to preserve GTIDs. A simulated replicated transaction committed through a client is completely equivalent to a replicated transaction committed through a replication applier thread, and they cannot be distinguished after the fact.

The `gtid_purged` System Variable

The set of GTIDs in the `gtid_purged` system variable (`@@GLOBAL.gtid_purged`) contains the GTIDs of all the transactions that have been committed on the server, but do not exist in any binary log file on the server. `gtid_purged` is a subset of `gtid_executed`. The following categories of GTIDs are in `gtid_purged`:

- GTIDs of replicated transactions that were committed with binary logging disabled on the slave.
- GTIDs of transactions that were written to a binary log file that has now been purged.
- GTIDs that were added explicitly to the set by the statement `SET @@GLOBAL.gtid_purged`.

You can change the value of `gtid_purged` in order to record on the server that the transactions in a certain GTID set have been applied, although they do not exist in any binary log on the server. When you

add GTIDs to `gtid_purged`, they are also added to `gtid_executed`. An example use case for this action is when you are restoring a backup of one or more databases on a server, but you do not have the relevant binary logs containing the transactions on the server. Before MySQL 8.0, you could only change the value of `gtid_purged` when `gtid_executed` (and therefore `gtid_purged`) was empty. From MySQL 8.0, this restriction does not apply, and you can also choose whether to replace the whole GTID set in `gtid_purged` with a specified GTID set, or to add a specified GTID set to the GTIDs already in `gtid_purged`. For details of how to do this, see the description for `gtid_purged`.

The sets of GTIDs in the `gtid_executed` and `gtid_purged` system variables are initialized when the server starts. Every binary log file begins with the event `Previous_gtids_log_event`, which contains the set of GTIDs in all previous binary log files (composed from the GTIDs in the preceding file's `Previous_gtids_log_event`, and the GTIDs of every `Gtid_log_event` in the preceding file itself). The contents of `Previous_gtids_log_event` in the oldest and most recent binary log files are used to compute the `gtid_executed` and `gtid_purged` sets at server startup:

- `gtid_executed` is computed as the union of the GTIDs in `Previous_gtids_log_event` in the most recent binary log file, the GTIDs of transactions in that binary log file, and the GTIDs stored in the `mysql.gtid_executed` table. This GTID set contains all the GTIDs that have been used (or added explicitly to `gtid_purged`) on the server, whether or not they are currently in a binary log file on the server. It does not include the GTIDs for transactions that are currently being processed on the server (`@@GLOBAL.gtid_owned`).

- `gtid_purged` is computed by first adding the GTIDs in `Previous_gtids_log_event` in the most recent binary log file and the GTIDs of transactions in that binary log file. This step gives the set of GTIDs that are currently, or were once, recorded in a binary log on the server (`gtids_in_binlog`). Next, the GTIDs in `Previous_gtids_log_event` in the oldest binary log file are subtracted from `gtids_in_binlog`. This step gives the set of GTIDs that are currently recorded in a binary log on the server (`gtids_in_binlog_not_purged`). Finally, `gtids_in_binlog_not_purged` is subtracted from `gtid_executed`. The result is the set of GTIDs that have been used on the server, but are not currently recorded in a binary log file on the server, and this result is used to initialize `gtid_purged`.

If binary logs from MySQL 5.7.7 or older are involved in these computations, it is possible for incorrect GTID sets to be computed for `gtid_executed` and `gtid_purged`, and they remain incorrect even if the server is later restarted. For details, see the description for the `binlog_gtid_simple_recovery` system variable, which controls how the binary logs are iterated to compute the GTID sets. If one of the situations described there applies on a server, set `binlog_gtid_simple_recovery=FALSE` in the server's configuration file before starting it. That setting makes the server iterate all the binary log files (not just the newest and oldest) to find where GTID events start to appear. This process could take a long time if the server has a large number of binary log files without GTID events.

Resetting the GTID Execution History

If you need to reset the GTID execution history on a server, use the `RESET MASTER` statement. For example, you might need to do this after carrying out test queries to verify a replication setup on new GTID-enabled servers, or when you want to join a new server to a replication group but it contains some unwanted local transactions that are not accepted by Group Replication.

Warning

Use `RESET MASTER` with caution to avoid losing any wanted GTID execution history and binary log files.

Before issuing `RESET MASTER`, ensure that you have backups of the server's binary log files and binary log index file, if any, and obtain and save the GTID set held in the global value of the `gtid_executed` system variable (for example, by issuing a `SELECT @@GLOBAL.gtid_executed` statement and saving the results). If you are removing unwanted transactions from that GTID set, use `mysqlbinlog` to examine

the contents of the transactions to ensure that they have no value, contain no data that must be saved or replicated, and did not result in data changes on the server.

When you issue `RESET MASTER`, the following reset operations are carried out:

- The value of the `gtid_purged` system variable is set to an empty string (`''`).
- The global value (but not the session value) of the `gtid_executed` system variable is set to an empty string.
- The `mysql.gtid_executed` table is cleared (see mysql.gtid_executed Table).
- If the server has binary logging enabled, the existing binary log files are deleted and the binary log index file is cleared.

Note that `RESET MASTER` is the method to reset the GTID execution history even if the server is a replication slave where binary logging is disabled. `RESET SLAVE` has no effect on the GTID execution history.

17.1.3.3 GTID Auto-Positioning

GTIDs replace the file-offset pairs previously required to determine points for starting, stopping, or resuming the flow of data between master and slave. When GTIDs are in use, all the information that the slave needs for synchronizing with the master is obtained directly from the replication data stream.

To start a slave using GTID-based replication, you do not include `MASTER_LOG_FILE` or `MASTER_LOG_POS` options in the `CHANGE MASTER TO` statement used to direct the slave to replicate from a given master. These options specify the name of the log file and the starting position within the file, but with GTIDs the slave does not need this nonlocal data. Instead, you need to enable the `MASTER_AUTO_POSITION` option. For full instructions to configure and start masters and slaves using GTID-based replication, see Section 17.1.3.4, “Setting Up Replication Using GTIDs”.

The `MASTER_AUTO_POSITION` option is disabled by default. If multi-source replication is enabled on the slave, you need to set the option for each applicable replication channel. Disabling the `MASTER_AUTO_POSITION` option again makes the slave revert to file-based replication, in which case you must also specify one or both of the `MASTER_LOG_FILE` or `MASTER_LOG_POS` options.

When a replication slave has GTIDs enabled (`GTID_MODE=ON`, `ON_PERMISSIVE`, or `OFF_PERMISSIVE`) and the `MASTER_AUTO_POSITION` option enabled, auto-positioning is activated for connection to the master. The master must have `GTID_MODE=ON` set in order for the connection to succeed. In the initial handshake, the slave sends a GTID set containing the transactions that it has already received, committed, or both. This GTID set is equal to the union of the set of GTIDs in the `gtid_executed` system variable (`@@GLOBAL.gtid_executed`), and the set of GTIDs recorded in the Performance Schema `replication_connection_status` table as received transactions (the result of the statement `SELECT RECEIVED_TRANSACTION_SET FROM PERFORMANCE_SCHEMA.replication_connection_status`).

The master responds by sending all transactions recorded in its binary log whose GTID is not included in the GTID set sent by the slave. This exchange ensures that the master only sends the transactions with a GTID that the slave has not already received or committed. If the slave receives transactions from more than one master, as in the case of a diamond topology, the auto-skip function ensures that the transactions are not applied twice.

If any of the transactions that should be sent by the master have been purged from the master's binary log, or added to the set of GTIDs in the `gtid_purged` system variable by another method, the master sends the error **`ER_MASTER_HAS_PURGED_REQUIRED_GTIDS`** to the slave, and replication does not start. The

GTIDs of the missing purged transactions are identified and listed in the master's error log in the warning message **ER_FOUND_MISSING_GTIDS**. The slave cannot recover automatically from this error because parts of the transaction history that are needed to catch up with the master have been purged. Attempting to reconnect without the `MASTER_AUTO_POSITION` option enabled only results in the loss of the purged transactions on the slave. The correct approach to recover from this situation is for the slave to replicate the missing transactions listed in the **ER_FOUND_MISSING_GTIDS** message from another source, or for the slave to be replaced by a new slave created from a more recent backup. Consider revising the binary log expiration period (`binlog_expire_logs_seconds`) on the master to ensure that the situation does not occur again.

If during the exchange of transactions it is found that the slave has received or committed transactions with the master's UUID in the GTID, but the master itself does not have a record of them, the master sends the error **ER_SLAVE_HAS_MORE_GTIDS_THAN_MASTER** to the slave and replication does not start. This situation can occur if a master that does not have `sync_binlog=1` set experiences a power failure or operating system crash, and loses committed transactions that have not yet been synchronized to the binary log file, but have been received by the slave. The master and slave can diverge if any clients commit transactions on the master after it is restarted, which can lead to the situation where the master and slave are using the same GTID for different transactions. The correct approach to recover from this situation is to check manually whether the master and slave have diverged. If the same GTID is now in use for different transactions, you either need to perform manual conflict resolution for individual transactions as required, or remove either the master or the slave from the replication topology. If the issue is only missing transactions on the master, you can make the master into a slave instead, allow it to catch up with the other servers in the replication topology, and then make it a master again if needed.

17.1.3.4 Setting Up Replication Using GTIDs

This section describes a process for configuring and starting GTID-based replication in MySQL 8.0. This is a “cold start” procedure that assumes either that you are starting the replication master for the first time, or that it is possible to stop it; for information about provisioning replication slaves using GTIDs from a running master, see Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”. For information about changing GTID mode on servers online, see Section 17.1.5, “Changing Replication Modes on Online Servers”.

The key steps in this startup process for the simplest possible GTID replication topology, consisting of one master and one slave, are as follows:

1. If replication is already running, synchronize both servers by making them read-only.
2. Stop both servers.
3. Restart both servers with GTIDs enabled and the correct options configured.

   The `mysqld` options necessary to start the servers as described are discussed in the example that follows later in this section.
4. Instruct the slave to use the master as the replication data source and to use auto-positioning. The SQL statements needed to accomplish this step are described in the example that follows later in this section.
5. Take a new backup. Binary logs containing transactions without GTIDs cannot be used on servers where GTIDs are enabled, so backups taken before this point cannot be used with your new configuration.
6. Start the slave, then disable read-only mode on both servers, so that they can accept updates.

In the following example, two servers are already running as master and slave, using MySQL's binary log position-based replication protocol. If you are starting with new servers, see Section 17.1.2.3,

"Creating a User for Replication" for information about adding a specific user for replication connections and Section 17.1.2.1, "Setting the Replication Master Configuration" for information about setting the `server_id` variable. The following examples show how to store `mysqld` startup options in server's option file, see Section 4.2.2.2, "Using Option Files" for more information. Alternatively you can use startup options when running `mysqld`.

Most of the steps that follow require the use of the MySQL `root` account or another MySQL user account that has the `SUPER` privilege. `mysqladmin shutdown` requires either the `SUPER` privilege or the `SHUTDOWN` privilege.

Step 1: Synchronize the servers. This step is only required when working with servers which are already replicating without using GTIDs. For new servers proceed to Step 3. Make the servers read-only by setting the `read_only` system variable to `ON` on each server by issuing the following:

```
mysql> SET @@GLOBAL.read_only = ON;
```

Wait for all ongoing transactions to commit or roll back. Then, allow the slave to catch up with the master. *It is extremely important that you make sure the slave has processed all updates before continuing*.

If you use binary logs for anything other than replication, for example to do point in time backup and restore, wait until you do not need the old binary logs containing transactions without GTIDs. Ideally, wait for the server to purge all binary logs, and wait for any existing backup to expire.

Important

It is important to understand that logs containing transactions without GTIDs cannot be used on servers where GTIDs are enabled. Before proceeding, you must be sure that transactions without GTIDs do not exist anywhere in the topology.

Step 2: Stop both servers. Stop each server using `mysqladmin` as shown here, where *`username`* is the user name for a MySQL user having sufficient privileges to shut down the server:

```
shell> mysqladmin -uusername -p shutdown
```

Then supply this user's password at the prompt.

Step 3: Start both servers with GTIDs enabled. To enable GTID-based replication, each server must be started with GTID mode enabled by setting the `gtid_mode` variable to `ON`, and with the `enforce_gtid_consistency` variable enabled to ensure that only statements which are safe for GTID-based replication are logged. For example:

```
gtid_mode=ON
enforce-gtid-consistency=ON
```

In addition, you should start slaves with the `--skip-slave-start` option before configuring the slave settings. For more information on GTID related options and variables, see Section 17.1.6.5, "Global Transaction ID Options and Variables".

It is not mandatory to have binary logging enabled in order to use GTIDs when using the mysql.gtid_executed Table. Masters must always have binary logging enabled in order to be able to replicate. However, slave servers can use GTIDs but without binary logging. If you need to disable binary logging on a slave server, you can do this by specifying the `--skip-log-bin` and `--log-slave-updates=OFF` options for the slave.

Step 4: Configure the slave to use GTID-based auto-positioning. Tell the slave to use the master with GTID based transactions as the replication data source, and to use GTID-based auto-positioning rather than file-based positioning. Issue a `CHANGE MASTER TO` statement on the slave, including the `MASTER_AUTO_POSITION` option in the statement to tell the slave that the master's transactions are identified by GTIDs.

You may also need to supply appropriate values for the master's host name and port number as well as the user name and password for a replication user account which can be used by the slave to connect to the master; if these have already been set prior to Step 1 and no further changes need to be made, the corresponding options can safely be omitted from the statement shown here.

```
mysql> CHANGE MASTER TO
     >     MASTER_HOST = host,
     >     MASTER_PORT = port,
     >     MASTER_USER = user,
     >     MASTER_PASSWORD = password,
     >     MASTER_AUTO_POSITION = 1;
```

Neither the `MASTER_LOG_FILE` option nor the `MASTER_LOG_POS` option may be used with `MASTER_AUTO_POSITION` set equal to 1. Attempting to do so causes the `CHANGE MASTER TO` statement to fail with an error.

Step 5: Take a new backup. Existing backups that were made before you enabled GTIDs can no longer be used on these servers now that you have enabled GTIDs. Take a new backup at this point, so that you are not left without a usable backup.

For instance, you can execute `FLUSH LOGS` on the server where you are taking backups. Then either explicitly take a backup or wait for the next iteration of any periodic backup routine you may have set up.

Step 6: Start the slave and disable read-only mode. Start the slave like this:

```
mysql> START SLAVE;
```

The following step is only necessary if you configured a server to be read-only in Step 1. To allow the server to begin accepting updates again, issue the following statement:

```
mysql> SET @@GLOBAL.read_only = OFF;
```

GTID-based replication should now be running, and you can begin (or resume) activity on the master as before. Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”, discusses creation of new slaves when using GTIDs.

17.1.3.5 Using GTIDs for Failover and Scaleout

There are a number of techniques when using MySQL Replication with Global Transaction Identifiers (GTIDs) for provisioning a new slave which can then be used for scaleout, being promoted to master as necessary for failover. This section describes the following techniques:

- Simple replication
- Copying data and transactions to the slave
- Injecting empty transactions
- Excluding transactions with gtid_purged

- Restoring GTID mode slaves

Global transaction identifiers were added to MySQL Replication for the purpose of simplifying in general management of the replication data flow and of failover activities in particular. Each identifier uniquely identifies a set of binary log events that together make up a transaction. GTIDs play a key role in applying changes to the database: the server automatically skips any transaction having an identifier which the server recognizes as one that it has processed before. This behavior is critical for automatic replication positioning and correct failover.

The mapping between identifiers and sets of events comprising a given transaction is captured in the binary log. This poses some challenges when provisioning a new server with data from another existing server. To reproduce the identifier set on the new server, it is necessary to copy the identifiers from the old server to the new one, and to preserve the relationship between the identifiers and the actual events. This is neccessary for restoring a slave that is immediately available as a candidate to become a new master on failover or switchover.

Simple replication. The easiest way to reproduce all identifiers and transactions on a new server is to make the new server into the slave of a master that has the entire execution history, and enable global transaction identifiers on both servers. See Section 17.1.3.4, “Setting Up Replication Using GTIDs”, for more information.

Once replication is started, the new server copies the entire binary log from the master and thus obtains all information about all GTIDs.

This method is simple and effective, but requires the slave to read the binary log from the master; it can sometimes take a comparatively long time for the new slave to catch up with the master, so this method is not suitable for fast failover or restoring from backup. This section explains how to avoid fetching all of the execution history from the master by copying binary log files to the new server.

Copying data and transactions to the slave. Executing the entire transaction history can be time-consuming when the source server has processed a large number of transactions previously, and this can represent a major bottleneck when setting up a new replication slave. To eliminate this requirement, a snapshot of the data set, the binary logs and the global transaction information the source server contains can be imported to the new slave. The source server can be either the master or the slave, but you must ensure that the source has processed all required transactions before copying the data.

There are several variants of this method, the difference being in the manner in which data dumps and transactions from binary logs are transfered to the slave, as outlined here:

Data Set

1. Create a dump file using `mysqldump` on the source server. Set the `mysqldump` option `--master-data` (with the default value of 1) to include a `CHANGE MASTER TO` statement with binary logging information. Set the `--set-gtid-purged` option to `AUTO` (the default) or `ON`, to include information about executed transactions in the dump. Then use the `mysql` client to import the dump file on the target server.

2. Alternatively, create a data snapshot of the source server using raw data files, then copy these files to the target server, following the instructions in Section 17.1.2.5, “Choosing a Method for Data Snapshots”. If you use `InnoDB` tables, you can use the `mysqlbackup` command from the MySQL Enterprise Backup component to produce a consistent snapshot. This command records the log name and offset corresponding to the snapshot to be used on the slave. MySQL Enterprise Backup is a commercial

product that is included as part of a MySQL Enterprise subscription. See Section 30.2, “MySQL Enterprise Backup Overview” for detailed information.

3. Alternatively, stop both the source and target servers, copy the contents of the source's data directory to the new slave's data directory, then restart the slave. If you use this method, the slave must be configured for GTID-based replication, in other words with `gtid_mode=ON`. For instructions and important information for this method, see Section 17.1.2.8, “Adding Slaves to a Replication Environment”.

Transaction History

If the source server has a complete transaction history in its binary logs (that is, the GTID set `@@GLOBAL.gtid_purged` is empty), you can use these methods.

1. Import the binary logs from the source server to the new slave using `mysqlbinlog`, with the `--read-from-remote-server` and `--read-from-remote-master` options.

2. Alternatively, copy the source server's binary log files to the slave. You can make copies from the slave using `mysqlbinlog` with the `--read-from-remote-server` and `--raw` options. These can be read into the slave by using `mysqlbinlog > file` (without the `--raw` option) to export the binary log files to SQL files, then passing these files to the `mysql` client for processing. Ensure that all of the binary log files are processed using a single `mysql` process, rather than multiple connections. For example:

```
shell> mysqlbinlog copied-binlog.000001 copied-binlog.000002 | mysql -u
```

For more information, see Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”.

This method has the advantage that a new server is available almost immediately; only those transactions that were committed while the snapshot or dump file was being replayed still need to be obtained from the existing master. This means that the slave's availability is not instantaneous, but only a relatively short amount of time should be required for the slave to catch up with these few remaining transactions.

Copying over binary logs to the target server in advance is usually faster than reading the entire transaction execution history from the master in real time. However, it may not always be feasible to move these files to the target when required, due to size or other considerations. The two remaining methods for provisioning a new slave discussed in this section use other means to transfer information about transactions to the new slave.

Injecting empty transactions. The master's global `gtid_executed` variable contains the set of all transactions executed on the master. Rather than copy the binary logs when taking a snapshot to provision a new server, you can instead note the content of `gtid_executed` on the server from which the snapshot was taken. Before adding the new server to the replication chain, simply commit an empty transaction on the new server for each transaction identifier contained in the master's `gtid_executed`, like this:

```
SET GTID_NEXT='aaa-bbb-ccc-ddd:N';

BEGIN;
COMMIT;
```

```
SET GTID_NEXT='AUTOMATIC';
```

Once all transaction identifiers have been reinstated in this way using empty transactions, you must flush and purge the slave's binary logs, as shown here, where *N* is the nonzero suffix of the current binary log file name:

```
FLUSH LOGS;
PURGE BINARY LOGS TO 'master-bin.00000N';
```

You should do this to prevent this server from flooding the replication stream with false transactions in the event that it is later promoted to master. (The `FLUSH LOGS` statement forces the creation of a new binary log file; `PURGE BINARY LOGS` purges the empty transactions, but retains their identifiers.)

This method creates a server that is essentially a snapshot, but in time is able to become a master as its binary log history converges with that of the replication stream (that is, as it catches up with the master or masters). This outcome is similar in effect to that obtained using the remaining provisioning method, which we discuss in the next few paragraphs.

Excluding transactions with gtid_purged. The master's global `gtid_purged` variable contains the set of all transactions that have been purged from the master's binary log. As with the method discussed previously (see Injecting empty transactions), you can record the value of `gtid_executed` on the server from which the snapshot was taken (in place of copying the binary logs to the new server). Unlike the previous method, there is no need to commit empty transactions (or to issue `PURGE BINARY LOGS`); instead, you can set `gtid_purged` on the slave directly, based on the value of `gtid_executed` on the server from which the backup or snapshot was taken.

As with the method using empty transactions, this method creates a server that is functionally a snapshot, but in time is able to become a master as its binary log history converges with that of the replication master or group.

Restoring GTID mode slaves. When restoring a slave in a GTID based replication setup that has encountered an error, injecting an empty transaction may not solve the problem because an event does not have a GTID.

Use `mysqlbinlog` to find the next transaction, which is probably the first transaction in the next log file after the event. Copy everything up to the `COMMIT` for that transaction, being sure to include the `SET @@SESSION.gtid_next`. Even if you are not using row-based replication, you can still run binary log row events in the command line client.

Stop the slave and run the transaction you copied. The `mysqlbinlog` output sets the delimiter to `/*!*/;`, so set it back:

```
mysql> DELIMITER ;
```

Restart replication from the correct position automatically:

```
mysql> SET GTID_NEXT=automatic;
mysql> RESET SLAVE;
mysql> START SLAVE;
```

17.1.3.6 Restrictions on Replication with GTIDs

Because GTID-based replication is dependent on transactions, some features otherwise available in MySQL are not supported when using it. This section provides information about restrictions on and limitations of replication with GTIDs.

Updates involving nontransactional storage engines. When using GTIDs, updates to tables using nontransactional storage engines such as `MyISAM` cannot be made in the same statement or transaction as updates to tables using transactional storage engines such as `InnoDB`.

This restriction is due to the fact that updates to tables that use a nontransactional storage engine mixed with updates to tables that use a transactional storage engine within the same transaction can result in multiple GTIDs being assigned to the same transaction.

Such problems can also occur when the master and the slave use different storage engines for their respective versions of the same table, where one storage engine is transactional and the other is not. Also be aware that triggers that are defined to operate on nontransactional tables can be the cause of these problems.

In any of the cases just mentioned, the one-to-one correspondence between transactions and GTIDs is broken, with the result that GTID-based replication cannot function correctly.

CREATE TABLE ... SELECT statements. `CREATE TABLE ... SELECT` statements are not allowed when using GTID-based replication. When `binlog_format` is set to STATEMENT, a `CREATE TABLE ... SELECT` statement is recorded in the binary log as one transaction with one GTID, but if ROW format is used, the statement is recorded as two transactions with two GTIDs. If a master used STATEMENT format and a slave used ROW format, the slave would be unable to handle the transaction correctly, therefore the `CREATE TABLE ... SELECT` statement is disallowed with GTIDs to prevent this scenario.

Temporary tables. When `binlog_format` is set to `STATEMENT`, `CREATE TEMPORARY TABLE` and `DROP TEMPORARY TABLE` statements cannot be used inside transactions, procedures, functions, and triggers when GTIDs are in use on the server (that is, when the `enforce_gtid_consistency` system variable is set to `ON`). They can be used outside these contexts when GTIDs are in use, provided that `autocommit=1` is set. From MySQL 8.0.13, when `binlog_format` is set to `ROW` or `MIXED`, `CREATE TEMPORARY TABLE` and `DROP TEMPORARY TABLE` statements are allowed inside a transaction, procedure, function, or trigger when GTIDs are in use. The statements are not written to the binary log and are therefore not replicated to slaves. The use of row-based replication means that the slaves remain in sync without the need to replicate temporary tables. If the removal of these statements from a transaction results in an empty transaction, the transaction is not written to the binary log.

Preventing execution of unsupported statements. To prevent execution of statements that would cause GTID-based replication to fail, all servers must be started with the `--enforce-gtid-consistency` option when enabling GTIDs. This causes statements of any of the types discussed previously in this section to fail with an error.

Note that `--enforce-gtid-consistency` only takes effect if binary logging takes place for a statement. If binary logging is disabled on the server, or if statements are not written to the binary log because they are removed by a filter, GTID consistency is not checked or enforced for the statements that are not logged.

For information about other required startup options when enabling GTIDs, see Section 17.1.3.4, “Setting Up Replication Using GTIDs”.

Skipping transactions. `sql_slave_skip_counter` is not supported when using GTIDs. If you need to skip transactions, use the value of the master's `gtid_executed` variable instead; see Injecting empty transactions, for more information.

Ignoring servers. The IGNORE_SERVER_IDS option of the `CHANGE MASTER TO` statement is deprecated when using GTIDs, because transactions that have already been applied are automatically ignored. Before starting GTID-based replication, check for and clear all ignored server ID lists that have previously been set on the servers involved. The `SHOW SLAVE STATUS` statement, which can be

issued for individual channels, displays the list of ignored server IDs if there is one. If there is no list, the `Replicate_Ignore_Server_Ids` field is blank.

GTID mode and mysqldump. It is possible to import a dump made using `mysqldump` into a MySQL server running with GTID mode enabled, provided that there are no GTIDs in the target server's binary log.

GTID mode and mysql_upgrade. Prior to MySQL 8.0.16, when the server is running with global transaction identifiers (GTIDs) enabled (`gtid_mode=ON`), do not enable binary logging by `mysql_upgrade` (the `--write-binlog` option). As of MySQL 8.0.16, the server performs the entire MySQL upgrade procedure, but disables binary logging during the upgrade, so there is no issue.

17.1.3.7 Stored Function Examples to Manipulate GTIDs

MySQL includes some built-in (native) functions for use with GTID-based replication. These functions are as follows:

`GTID_SUBSET(`*`set1`*`,`*`set2`*`)`	Given two sets of global transaction identifiers *`set1`* and *`set2`*, returns true if all GTIDs in *`set1`* are also in *`set2`*. Returns false otherwise.
`GTID_SUBTRACT(`*`set1`*`,`*`set2`*`)`	Given two sets of global transaction identifiers *`set1`* and *`set2`*, returns only those GTIDs from *`set1`* that are not in *`set2`*.
`WAIT_FOR_EXECUTED_GTID_SET(`*`gtid_set`*`[, `*`timeout`*`])`	Wait until the server has applied all of the transactions whose global transaction identifiers are contained in *`gtid_set`*. The optional timeout stops the function from waiting after the specified number of seconds have elapsed.

For details of these functions, see Section 12.18, “Functions Used with Global Transaction Identifiers (GTIDs)”.

You can define your own stored functions to work with GTIDs. For information on defining stored functions, see Chapter 24, *Stored Objects*. The following examples show some useful stored functions that can be created based on the built-in `GTID_SUBSET()` and `GTID_SUBTRACT()` functions.

Note that in these stored functions, the delimiter command has been used to change the MySQL statement delimiter to a vertical bar, as follows:

```
mysql> delimiter |
```

All of these functions take string representations of GTID sets as arguments, so GTID sets must always be quoted when used with them.

This function returns nonzero (true) if two GTID sets are the same set, even if they are not formatted in the same way.

```
CREATE FUNCTION GTID_IS_EQUAL(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT)
RETURNS INT
  RETURN GTID_SUBSET(gtid_set_1, gtid_set_2) AND GTID_SUBSET(gtid_set_2, gtid_set_1)|
```

This function returns nonzero (true) if two GTID sets are disjoint.

```
CREATE FUNCTION GTID_IS_DISJOINT(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT)
RETURNS INT
  RETURN GTID_SUBSET(gtid_set_1, GTID_SUBTRACT(gtid_set_1, gtid_set_2))|
```

This function returns nonzero (true) if two GTID sets are disjoint, and `sum` is the union of the two sets.

```
CREATE FUNCTION GTID_IS_DISJOINT_UNION(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT, sum LONGTEXT)
RETURNS INT
```

```
  RETURN GTID_IS_EQUAL(GTID_SUBTRACT(sum, gtid_set_1), gtid_set_2) AND
         GTID_IS_EQUAL(GTID_SUBTRACT(sum, gtid_set_2), gtid_set_1)|
```

This function returns a normalized form of the GTID set, in all uppercase, with no whitespace and no duplicates. The UUIDs are arranged in alphabetic order and intervals are arranged in numeric order.

```
CREATE FUNCTION GTID_NORMALIZE(g LONGTEXT)
RETURNS LONGTEXT
RETURN GTID_SUBTRACT(g, '')|
```

This function returns the union of two GTID sets.

```
CREATE FUNCTION GTID_UNION(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT)
RETURNS LONGTEXT
  RETURN GTID_NORMALIZE(CONCAT(gtid_set_1, ',', gtid_set_2))|
```

This function returns the intersection of two GTID sets.

```
CREATE FUNCTION GTID_INTERSECTION(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT)
RETURNS LONGTEXT
  RETURN GTID_SUBTRACT(gtid_set_1, GTID_SUBTRACT(gtid_set_1, gtid_set_2))|
```

This function returns the symmetric difference between two GTID sets, that is, the GTIDs that exist in `gtid_set_1` but not in `gtid_set_2`, and also the GTIDs that exist in `gtid_set_2` but not in `gtid_set_1`.

```
CREATE FUNCTION GTID_SYMMETRIC_DIFFERENCE(gtid_set_1 LONGTEXT, gtid_set_2 LONGTEXT)
RETURNS LONGTEXT
  RETURN GTID_SUBTRACT(CONCAT(gtid_set_1, ',', gtid_set_2), GTID_INTERSECTION(gtid_set_1, gtid_set_2))|
```

This function removes from a GTID set all the GTIDs from a specified origin, and returns the remaining GTIDs, if any. The UUID is the identifier used by the server where the transaction originated, which is normally the `server_uuid` value.

```
CREATE FUNCTION GTID_SUBTRACT_UUID(gtid_set LONGTEXT, uuid TEXT)
RETURNS LONGTEXT
  RETURN GTID_SUBTRACT(gtid_set, CONCAT(UUID, ':1-', (1 << 63) - 2))|
```

This function reverses the previously listed function to return only those GTIDs from the GTID set that originate from the server with the specified identifier (UUID).

```
CREATE FUNCTION GTID_INTERSECTION_WITH_UUID(gtid_set LONGTEXT, uuid TEXT)
RETURNS LONGTEXT
  RETURN GTID_SUBTRACT(gtid_set, GTID_SUBTRACT_UUID(gtid_set, uuid))|
```

Example 17.1 Verifying that a replication slave is up to date

The built-in functions `GTID_SUBSET` and `GTID_SUBTRACT` can be used to check that a replication slave has applied at least every transaction that a master has applied.

To perform this check with `GTID_SUBSET`, execute the following statement on the slave:

```
SELECT GTID_SUBSET(master_gtid_executed, slave_gtid_executed)
```

If this returns 0 (false), some GTIDs in *`master_gtid_executed`* are not present in *`slave_gtid_executed`*, so the master has applied some transactions that the slave has not applied, and the slave is therefore not up to date.

To perform the check with `GTID_SUBTRACT`, execute the following statement on the slave:

```
SELECT GTID_SUBTRACT(master_gtid_executed, slave_gtid_executed)
```

This statement returns any GTIDs that are in *`master_gtid_executed`* but not in *`slave_gtid_executed`*. If any GTIDs are returned, the master has applied some transactions that the slave has not applied, and the slave is therefore not up to date.

Example 17.2 Backup and restore scenario

The stored functions `GTID_IS_EQUAL`, `GTID_IS_DISJOINT`, and `GTID_IS_DISJOINT_UNION` could be used to verify backup and restore operations involving multiple databases and servers. In this example scenario, `server1` contains database `db1`, and `server2` contains database `db2`. The goal is to copy database `db2` to `server1`, and the result on `server1` should be the union of the two databases. The procedure used is to back up `server2` using `mysqlpump` or `mysqldump`, then restore this backup on `server1`.

Provided the backup program's option `--set-gtid-purged` was set to `ON` or the default of `AUTO`, the program's output contains a `SET @@GLOBAL.gtid_purged` statement that will add the `gtid_executed` set from `server2` to the `gtid_purged` set on `server1`. The `gtid_purged` set contains the GTIDs of all the transactions that have been committed on a server but do not exist in any binary log file on the server. When database `db2` is copied to `server1`, the GTIDs of the transactions committed on `server2`, which are not in the binary log files on `server1`, must be added to `server1`'s `gtid_purged` set to make the set complete.

The stored functions can be used to assist with the following steps in this scenario:

- Use `GTID_IS_EQUAL` to verify that the backup operation computed the correct GTID set for the `SET @@GLOBAL.gtid_purged` statement. On `server2`, extract that statement from the `mysqlpump` or `mysqldump` output, and store the GTID set into a local variable, such as `$gtid_purged_set`. Then execute the following statement:

  ```
  server2> SELECT GTID_IS_EQUAL($gtid_purged_set, @@GLOBAL.gtid_executed);
  ```

  If the result is 1, the two GTID sets are equal, and the set has been computed correctly.

- Use `GTID_IS_DISJOINT` to verify that the GTID set in the `mysqlpump` or `mysqldump` output does not overlap with the `gtid_executed` set on `server1`. If there is any overlap, with identical GTIDs present on both servers for some reason, you will see errors when copying database `db2` to `server1`. To check, on `server1`, extract and store the `gtid_purged` set from the output into a local variable as above, then execute the following statement:

  ```
  server1> SELECT GTID_IS_DISJOINT($gtid_purged_set, @@GLOBAL.gtid_executed);
  ```

  If the result is 1, there is no overlap between the two GTID sets, so no duplicate GTIDs are present.

- Use `GTID_IS_DISJOINT_UNION` to verify that the restore operation resulted in the correct GTID state on `server1`. Before restoring the backup, on `server1`, obtain the existing `gtid_executed` set by executing the following statement:

  ```
  server1> SELECT @@GLOBAL.gtid_executed;
  ```

  Store the result in a local variable `$original_gtid_executed`. Also store the `gtid_purged` set in a local variable as described above. When the backup from `server2` has been restored onto `server1`, execute the following statement to verify the GTID state:

  ```
  server1> SELECT GTID_IS_DISJOINT_UNION($original_gtid_executed,
                                         $gtid_purged_set,
                                         @@GLOBAL.gtid_executed);
  ```

  If the result is 1, the stored function has verified that the original `gtid_executed` set from `server1` (`$original_gtid_executed`) and the `gtid_purged` set that was added from `server2`

(`$gtid_purged_set`) have no overlap, and also that the updated `gtid_executed` set on `server1` now consists of the previous `gtid_executed` set from `server1` plus the `gtid_purged` set from `server2`, which is the desired result. Ensure that this check is carried out before any further transactions take place on `server1`, otherwise the new transactions in the `gtid_executed` set will cause it to fail.

Example 17.3 Selecting the most up-to-date slave for manual failover

The stored function `GTID_UNION` could be used to identify the most up-to-date replication slave from a set of slaves, in order to perform a manual failover operation after a replication master has stopped unexpectedly. If some of the slaves are experiencing replication lag, this stored function can be used to compute the most up-to-date slave without waiting for all the slaves to apply their existing relay logs, and therefore to minimize the failover time. The function can return the union of the `gtid_executed` set on each slave with the set of transactions received by the slave, which is recorded in the Performance Schema table `replication_connection_status`. You can compare these results to find which slave's record of transactions is the most up-to-date, even if not all of the transactions have been committed yet.

On each replication slave, compute the complete record of transactions by issuing the following statement:

```
SELECT GTID_UNION(RECEIVED_TRANSACTION_SET, @@GLOBAL.gtid_executed)
    FROM performance_schema.replication_connection_status
    WHERE channel_name = 'name';
```

You can then compare the results from each slave to see which one has the most up-to-date record of transactions, and use this slave as the new replication master.

Example 17.4 Checking for extraneous transactions on a replication slave

The stored function `GTID_SUBTRACT_UUID` could be used to check whether a replication slave has received transactions that did not originate from its designated master or masters. If it has, there might be an issue with your replication setup, or with a proxy, router, or load balancer. This function works by removing from a GTID set all the GTIDs from a specified originating server, and returning the remaining GTIDs, if any.

For a replication slave with a single master, issue the following statement, giving the identifier of the originating replication master, which is normally the `server_uuid` value:

```
SELECT GTID_SUBTRACT_UUID(@@GLOBAL.gtid_executed, server_uuid_of_master);
```

If the result is not empty, the transactions returned are extra transactions that did not originate from the designated master.

For a slave in a multi-master replication topology, repeat the function, for example:

```
SELECT GTID_SUBTRACT_UUID(GTID_SUBTRACT_UUID(@@GLOBAL.gtid_executed,
                                             server_uuid_of_master_1),
                                             server_uuid_of_master_2);
```

If the result is not empty, the transactions returned are extra transactions that did not originate from any of the designated masters.

Example 17.5 Verifying that a server in a replication topology is read-only

The stored function `GTID_INTERSECTION_WITH_UUID` could be used to verify that a server has not originated any GTIDs and is in a read-only state. The function returns only those GTIDs from the GTID set that originate from the server with the specified identifier. If any of the transactions in the server's `gtid_executed` set have the server's own identifier, the server itself originated those transactions. You can issue the following statement on the server to check:

```
SELECT GTID_INTERSECTION_WITH_UUID(@@GLOBAL.gtid_executed, my_server_uuid);
```

Example 17.6 Validating an additional slave in a multi-master replication setup

The stored function `GTID_INTERSECTION_WITH_UUID` could be used to find out if a slave attached to a multi-master replication setup has applied all the transactions originating from one particular master. In this scenario, `master1` and `master2` are both masters and slaves and replicate to each other. `master2` also has its own replication slave. The replication slave will also receive and apply `master1`'s transactions if `master2` is configured with `log_slave_updates=ON`, but it will not do so if `master2` uses `log_slave_updates=OFF`. Whatever the case, we currently only want to find out if the replication slave is up to date with `master2`. In this situation, the stored function `GTID_INTERSECTION_WITH_UUID` can be used to identify the transactions that `master2` originated, discarding the transactions that `master2` has replicated from `master1`. The built-in function `GTID_SUBSET` can then be used to compare the result to the `gtid_executed` set on the slave. If the slave is up to date with `master2`, the `gtid_executed` set on the slave contains all the transactions in the intersection set (the transactions that originated from `master2`).

To carry out this check, store `master2`'s `gtid_executed` set, `master2`'s server UUID, and the slave's `gtid_executed` set, into client-side variables as follows:

```
    $master2_gtid_executed :=
      master2> SELECT @@GLOBAL.gtid_executed;
    $master2_server_uuid :=
      master2> SELECT @@GLOBAL.server_uuid;
    $slave_gtid_executed :=
      slave> SELECT @@GLOBAL.gtid_executed;
```

Then use `GTID_INTERSECTION_WITH_UUID` and `GTID_SUBSET` with these variables as input, as follows:

```
SELECT GTID_SUBSET(GTID_INTERSECTION_WITH_UUID($master2_gtid_executed,
                                               $master2_server_uuid),
                                               $slave_gtid_executed);
```

The server identifier from `master2` (`$master2_server_uuid`) is used with `GTID_INTERSECTION_WITH_UUID` to identify and return only those GTIDs from `master2`'s `gtid_executed` set that originated on `master2`, omitting those that originated on `master1`. The resulting GTID set is then compared with the set of all executed GTIDs on the slave, using `GTID_SUBSET`. If this statement returns nonzero (true), all the identified GTIDs from `master2` (the first set input) are also in the slave's `gtid_executed` set (the second set input), meaning that the slave has replicated all the transactions that originated from `master2`.

17.1.4 MySQL Multi-Source Replication

This section describes MySQL Multi-Source Replication, which enables you to replicate from multiple immediate masters in parallel. This section describes multi-source replication, and how to configure, monitor and troubleshoot it.

17.1.4.1 MySQL Multi-Source Replication Overview

MySQL Multi-Source Replication enables a replication slave to receive transactions from multiple sources simultaneously. Multi-source replication can be used to back up multiple servers to a single server, to merge table shards, and consolidate data from multiple servers to a single server. Multi-source replication does not implement any conflict detection or resolution when applying the transactions, and those tasks are left to the application if required. In a multi-source replication topology, a slave creates a replication channel for each master that it should receive transactions from. See Section 17.2.3, “Replication Channels”. The following sections describe how to set up multi-source replication.

17.1.4.2 Multi-Source Replication Tutorials

This section provides tutorials on how to configure masters and slaves for multi-source replication, and how to start, stop and reset multi-source slaves.

Configuring Multi-Source Replication

This section explains how to configure a multi-source replication topology, and provides details about configuring masters and slaves. Such a topology requires at least two masters and one slave configured.

Masters in a multi-source replication topology can be configured to use either global transaction identifier (GTID) based replication, or binary log position-based replication. See Section 17.1.3.4, "Setting Up Replication Using GTIDs" for how to configure a master using GTID based replication. See Section 17.1.2.1, "Setting the Replication Master Configuration" for how to configure a master using file position based replication.

Slaves in a multi-source replication topology require `TABLE` repositories for the master info log and relay log info log, which are the default in MySQL 8.0. Multi-source replication is not compatible with `FILE` based repositories, and the FILE setting for the `--master-info-repository` and `--relay-log-info-repository` options is now deprecated.

To modify an existing replication slave that is using a `FILE` repository for the slave status logs to use `TABLE` repositories, convert the existing replication repositories dynamically by running the following commands:

```
STOP SLAVE;
SET GLOBAL master_info_repository = 'TABLE';
SET GLOBAL relay_log_info_repository = 'TABLE';
```

Adding a GTID Based Master to a Multi-Source Replication Slave

This section assumes you have enabled GTID based transactions on the master using `gtid_mode=ON`, enabled a replication user, and ensured that the slave is using `TABLE` based replication repositories. Use the `CHANGE MASTER TO` statement to add a new master to a channel by using a `FOR CHANNEL` *`channel`* clause. For more information on replication channels, see Section 17.2.3, "Replication Channels"

For example, to add a new master with the host name *`master1`* using port *`3451`* to a channel called *`master-1`*:

```
CHANGE MASTER TO MASTER_HOST='master1', MASTER_USER='rpl', MASTER_PORT=3451, MASTER_PASSWORD='', \
MASTER_AUTO_POSITION = 1 FOR CHANNEL 'master-1';
```

Multi-source replication is compatible with auto-positioning. See Section 13.4.2.1, "CHANGE MASTER TO Syntax" for more information.

Repeat this process for each extra master that you want to add to a channel, changing the host name, port and channel as appropriate.

Adding a Binary Log Based Master to a Multi-Source Replication Slave

This section assumes that binary logging is enabled on the master (which is the default), the slave is using `TABLE` based replication repositories (which is the default in MySQL 8.0), and that you have enabled a replication user and noted the current binary log position. You need to know the current `MASTER_LOG_FILE` and `MASTER_LOG_POSITION`. Use the `CHANGE MASTER TO` statement to add a new master to a channel by specifying a `FOR CHANNEL` *`channel`* clause. For example, to add a new master with the host name *`master1`* using port *`3451`* to a channel called *`master-1`*:

```
CHANGE MASTER TO MASTER_HOST='master1', MASTER_USER='rpl', MASTER_PORT=3451, MASTER_PASSWORD='' \
MASTER_LOG_FILE='master1-bin.000006', MASTER_LOG_POS=628 FOR CHANNEL 'master-1';
```

Repeat this process for each extra master that you want to add to a channel, changing the host name, port and channel as appropriate.

Starting Multi-Source Replication Slaves

Once you have added all of the channels you want to use as replication masters, use a `START SLAVE thread_types` statement to start replication. When you have enabled multiple channels on a slave, you can choose to either start all channels, or select a specific channel to start.

- To start all currently configured replication channels:

```
START SLAVE thread_types;
```

- To start only a named channel, use a `FOR CHANNEL channel` clause:

```
START SLAVE thread_types FOR CHANNEL channel;
```

Use the `thread_types` option to choose specific threads you want the above statements to start on the slave. See Section 13.4.2.6, “START SLAVE Syntax” for more information.

Stopping Multi-Source Replication Slaves

The `STOP SLAVE` statement can be used to stop a multi-source replication slave. By default, if you use the `STOP SLAVE` statement on a multi-source replication slave all channels are stopped. Optionally, use the `FOR CHANNEL channel` clause to stop only a specific channel.

- To stop all currently configured replication channels:

```
STOP SLAVE thread_types;
```

- To stop only a named channel, use a `FOR CHANNEL channel` clause:

```
STOP SLAVE thread_types FOR CHANNEL channel;
```

Use the `thread_types` option to choose specific threads you want the above statements to stop on the slave. See Section 13.4.2.7, “STOP SLAVE Syntax” for more information.

Resetting Multi-Source Replication Slaves

The `RESET SLAVE` statement can be used to reset a multi-source replication slave. By default, if you use the `RESET SLAVE` statement on a multi-source replication slave all channels are reset. Optionally, use the `FOR CHANNEL channel` clause to reset only a specific channel.

- To reset all currently configured replication channels:

```
RESET SLAVE;
```

- To reset only a named channel, use a `FOR CHANNEL channel` clause:

```
RESET SLAVE FOR CHANNEL channel;
```

See Section 13.4.2.4, “RESET SLAVE Syntax” for more information.

17.1.4.3 Multi-Source Replication Monitoring

To monitor the status of replication channels the following options exist:

- Using the replication Performance Schema tables. The first column of these tables is `Channel_Name`. This enables you to write complex queries based on `Channel_Name` as a key. See Section 26.12.11, “Performance Schema Replication Tables”.
- Using `SHOW SLAVE STATUS FOR CHANNEL channel`. By default, if the `FOR CHANNEL channel` clause is not used, this statement shows the slave status for all channels with one row per channel. The identifier `Channel_name` is added as a column in the result set. If a `FOR CHANNEL channel` clause is provided, the results show the status of only the named replication channel.

Note

The `SHOW VARIABLES` statement does not work with multiple replication channels. The information that was available through these variables has been migrated to the replication performance tables. Using a `SHOW VARIABLES` statement in a topology with multiple channels shows the status of only the default channel.

Monitoring Channels Using Performance Schema Tables

This section explains how to use the replication Performance Schema tables to monitor channels. You can choose to monitor all channels, or a subset of the existing channels.

To monitor the connection status of all channels:

```
mysql> SELECT * FROM replication_connection_status\G;
*************************** 1. row ***************************
CHANNEL_NAME: master1
GROUP_NAME:
SOURCE_UUID: 046e41f8-a223-11e4-a975-0811960cc264
THREAD_ID: 24
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 0
LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
RECEIVED_TRANSACTION_SET: 046e41f8-a223-11e4-a975-0811960cc264:4-37
LAST_ERROR_NUMBER: 0
LAST_ERROR_MESSAGE:
LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
*************************** 2. row ***************************
CHANNEL_NAME: master2
GROUP_NAME:
SOURCE_UUID: 7475e474-a223-11e4-a978-0811960cc264
THREAD_ID: 26
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 0
LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
RECEIVED_TRANSACTION_SET: 7475e474-a223-11e4-a978-0811960cc264:4-6
LAST_ERROR_NUMBER: 0
LAST_ERROR_MESSAGE:
LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
2 rows in set (0.00 sec)
```

In the above output there are two channels enabled, and as shown by the `CHANNEL_NAME` field they are called `master1` and `master2`.

The addition of the `CHANNEL_NAME` field enables you to query the Performance Schema tables for a specific channel. To monitor the connection status of a named channel, use a `WHERE CHANNEL_NAME=channel` clause:

```
mysql> SELECT * FROM replication_connection_status WHERE CHANNEL_NAME='master1'\G
*************************** 1. row ***************************
CHANNEL_NAME: master1
GROUP_NAME:
SOURCE_UUID: 046e41f8-a223-11e4-a975-0811960cc264
THREAD_ID: 24
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 0
LAST_HEARTBEAT_TIMESTAMP: 0000-00-00 00:00:00
RECEIVED_TRANSACTION_SET: 046e41f8-a223-11e4-a975-0811960cc264:4-37
LAST_ERROR_NUMBER: 0
LAST_ERROR_MESSAGE:
LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
1 row in set (0.00 sec)
```

Similarly, the `WHERE CHANNEL_NAME=channel` clause can be used to monitor the other replication Performance Schema tables for a specific channel. For more information, see Section 26.12.11, "Performance Schema Replication Tables".

17.1.4.4 Multi-Source Replication Error Messages

Error codes and messages provide information about errors encountered in a multi-source replication topology. These error codes and messages are only emitted when multi-source replication is enabled, and provide information related to the channel which generated the error. For example:

`Slave is already running` and `Slave is already stopped` have been replaced with `Replication thread(s) for channel channel_name are already running` and `Replication threads(s) for channel channel_name are already stopped` respectively.

The server log messages have also been changed to indicate which channel the log messages relate to. This makes debugging and tracing easier.

17.1.5 Changing Replication Modes on Online Servers

This section describes how to change the mode of replication being used without having to take the server offline.

17.1.5.1 Replication Mode Concepts

To be able to safely configure the replication mode of an online server it is important to understand some key concepts of replication. This section explains these concepts and is essential reading before attempting to modify the replication mode of an online server.

The modes of replication available in MySQL rely on different techniques for identifying transactions which are logged. The types of transactions used by replication are as follows:

- GTID transactions are identified by a global transaction identifier (GTID) in the form `UUID:NUMBER`. Every GTID transaction in a log is always preceded by a `Gtid_log_event`. GTID transactions can be addressed using either the GTID or using the file name and position.
- Anonymous transactions do not have a GTID assigned, and MySQL ensures that every anonymous transaction in a log is preceded by an `Anonymous_gtid_log_event`. In previous versions, anonymous transactions were not preceded by any particular event. Anonymous transactions can only be addressed using file name and position.

When using GTIDs you can take advantage of auto-positioning and automatic fail-over, as well as use `WAIT_FOR_EXECUTED_GTID_SET()`, `session_track_gtids`, and monitor replicated transactions using Performance Schema tables. With GTIDs enabled you cannot use `sql_slave_skip_counter`, instead use empty transactions.

Transactions in a relay log that was received from a master running a previous version of MySQL may not be preceded by any particular event at all, but after being replayed and logged in the slave's binary log, they are preceded with an `Anonymous_gtid_log_event`.

The ability to configure the replication mode online means that the `gtid_mode` and `enforce_gtid_consistency` variables are now both dynamic and can be set from a top-level statement by an account that has privileges sufficient to set global system variables. See Section 5.1.9.1, "System Variable Privileges". In MySQL 5.6 and earlier, both of these variables could only be configured using the appropriate option at server start, meaning that changes to the replication mode required a server restart. In all versions `gtid_mode` could be set to `ON` or `OFF`, which corresponded to whether GTIDs were used to identify transactions or not. When `gtid_mode=ON` it is not possible to replicate anonymous transactions, and when `gtid_mode=OFF` only anonymous transactions can be replicated. When `gtid_mode=OFF_PERMISSIVE` then *new* transactions are anonymous while permitting replicated transactions to be either GTID or anonymous transactions. When `gtid_mode=ON_PERMISSIVE` then *new* transactions use GTIDs while permitting replicated transactions to be either GTID or anonymous transactions. This means it is possible to have a replication topology that has servers using both anonymous and GTID transactions. For example a master with `gtid_mode=ON` could be replicating to a slave with `gtid_mode=ON_PERMISSIVE`. The valid values for `gtid_mode` are as follows and in this order:

- `OFF`
- `OFF_PERMISSIVE`
- `ON_PERMISSIVE`
- `ON`

It is important to note that the state of `gtid_mode` can only be changed by one step at a time based on the above order. For example, if `gtid_mode` is currently set to `OFF_PERMISSIVE`, it is possible to change to `OFF` or `ON_PERMISSIVE` but not to `ON`. This is to ensure that the process of changing from anonymous transactions to GTID transactions online is correctly handled by the server. When you switch between `gtid_mode=ON` and `gtid_mode=OFF`, the GTID state (in other words the value of `gtid_executed`) is persistent. This ensures that the GTID set that has been applied by the server is always retained, regardless of changes between types of `gtid_mode`.

The fields related to GTIDs display the correct information regardless of the currently selected `gtid_mode`. This means that fields which display GTID sets, such as `gtid_executed`, `gtid_purged`, `RECEIVED_TRANSACTION_SET` in the `replication_connection_status` Performance Schema table, and the GTID related results of `SHOW SLAVE STATUS`, now return the empty string when there are no GTIDs present. Fields that display a single GTID, such as `CURRENT_TRANSACTION` in the Performance Schema `replication_applier_status_by_worker` table, now display `ANONYMOUS` when GTID transactions are not being used.

Replication from a master using `gtid_mode=ON` provides the ability to use auto-positioning, configured using the `CHANGE MASTER TO MASTER_AUTO_POSITION = 1;` statement. The replication topology being used impacts on whether it is possible to enable auto-positioning or not, as this feature relies on GTIDs and is not compatible with anonymous transactions. An error is generated if auto-positioning is enabled and an anonymous transaction is encountered. It is strongly recommended to ensure there are no anonymous transactions remaining in the topology before enabling auto-positioning, see Section 17.1.5.2, "Enabling GTID Transactions Online".

The valid combinations of `gtid_mode` and auto-positioning on master and slave are shown in the following table, where the master's `gtid_mode` is shown on the horizontal and the slave's `gtid_mode` is on the vertical. The meaning of each entry is as follows:

- `Y`: the `gtid_mode` of master and slave is compatible
- `N`: the `gtid_mode` of master and slave is not compatible
- `*`: auto-positioning can be used with this combination

Table 17.1 Valid Combinations of Master and Slave gtid_mode

`gtid_mode`	Master `OFF`	Master `OFF_PERMISSIVE`	Master `ON_PERMISSIVE`	Master `ON`
Slave `OFF`	Y	Y	N	N
Slave `OFF_PERMISSIVE`	Y	Y	Y	Y*
Slave `ON_PERMISSIVE`	Y	Y	Y	Y*
Slave `ON`	N	N	Y	Y*

The currently selected `gtid_mode` also impacts on the `gtid_next` variable. The following table shows the behavior of the server for the different values of `gtid_mode` and `gtid_next`. The meaning of each entry is as follows:

- `ANONYMOUS`: generate an anonymous transaction.
- `Error`: generate an error and fail to execute `SET GTID_NEXT`.
- `UUID:NUMBER`: generate a GTID with the specified UUID:NUMBER.
- `New GTID`: generate a GTID with an automatically generated number.

Table 17.2 Valid Combinations of gtid_mode and gtid_next

	`gtid_next` **AUTOMATIC** **binary log on**	`gtid_next` **AUTOMATIC** **binary log off**	`gtid_next` **ANONYMOUS**	`gtid_next` **UUID:NUMBER**
`gtid_mode OFF`	ANONYMOUS	ANONYMOUS	ANONYMOUS	Error
`gtid_mode OFF_PERMISSIVE`	ANONYMOUS	ANONYMOUS	ANONYMOUS	UUID:NUMBER
`gtid_mode ON_PERMISSIVE`	New GTID	ANONYMOUS	ANONYMOUS	UUID:NUMBER
`gtid_mode ON`	New GTID	ANONYMOUS	Error	UUID:NUMBER

When the binary log is off and `gtid_next` is set to `AUTOMATIC`, then no GTID is generated. This is consistent with the behavior of previous versions.

17.1.5.2 Enabling GTID Transactions Online

This section describes how to enable GTID transactions, and optionally auto-positioning, on servers that are already online and using anonymous transactions. This procedure does not require taking the server offline and is suited to use in production. However, if you have the possibility to take the servers offline when enabling GTID transactions that process is easier.

Before you start, ensure that the servers meet the following pre-conditions:

- *All* servers in your topology must use MySQL 5.7.6 or later. You cannot enable GTID transactions online on any single server unless *all* servers which are in the topology are using this version.

- All servers have `gtid_mode` set to the default value `OFF`.

The following procedure can be paused at any time and later resumed where it was, or reversed by jumping to the corresponding step of Section 17.1.5.3, “Disabling GTID Transactions Online”, the online procedure to disable GTIDs. This makes the procedure fault-tolerant because any unrelated issues that may appear in the middle of the procedure can be handled as usual, and then the procedure continued where it was left off.

Note

It is crucial that you complete every step before continuing to the next step.

To enable GTID transactions:

1. On each server, execute:

   ```
   SET @@GLOBAL.ENFORCE_GTID_CONSISTENCY = WARN;
   ```

   Let the server run for a while with your normal workload and monitor the logs. If this step causes any warnings in the log, adjust your application so that it only uses GTID-compatible features and does not generate any warnings.

   Important

   This is the first important step. You must ensure that no warnings are being generated in the error logs before going to the next step.

2. On each server, execute:

   ```
   SET @@GLOBAL.ENFORCE_GTID_CONSISTENCY = ON;
   ```

3. On each server, execute:

   ```
   SET @@GLOBAL.GTID_MODE = OFF_PERMISSIVE;
   ```

   It does not matter which server executes this statement first, but it is important that all servers complete this step before any server begins the next step.

4. On each server, execute:

   ```
   SET @@GLOBAL.GTID_MODE = ON_PERMISSIVE;
   ```

   It does not matter which server executes this statement first.

5. On each server, wait until the status variable `ONGOING_ANONYMOUS_TRANSACTION_COUNT` is zero. This can be checked using:

   ```
   SHOW STATUS LIKE 'ONGOING_ANONYMOUS_TRANSACTION_COUNT';
   ```

   Note

   On a replication slave, it is theoretically possible that this shows zero and then nonzero again. This is not a problem, it suffices that it shows zero once.

6. Wait for all transactions generated up to step 5 to replicate to all servers. You can do this without stopping updates: the only important thing is that all anonymous transactions get replicated.

   See Section 17.1.5.4, “Verifying Replication of Anonymous Transactions” for one method of checking that all anonymous transactions have replicated to all servers.

7. If you use binary logs for anything other than replication, for example point in time backup and restore, wait until you do not need the old binary logs having transactions without GTIDs.

   For instance, after step 6 has completed, you can execute `FLUSH LOGS` on the server where you are taking backups. Then either explicitly take a backup or wait for the next iteration of any periodic backup routine you may have set up.

   Ideally, wait for the server to purge all binary logs that existed when step 6 was completed. Also wait for any backup taken before step 6 to expire.

   Important

   This is the second important point. It is vital to understand that binary logs containing anonymous transactions, without GTIDs cannot be used after the next step. After this step, you must be sure that transactions without GTIDs do not exist anywhere in the topology.

8. On each server, execute:

   ```
   SET @@GLOBAL.GTID_MODE = ON;
   ```

9. On each server, add `gtid_mode=ON` and `enforce_gtid_consistency=ON` to `my.cnf`.

   You are now guaranteed that all transactions have a GTID (except transactions generated in step 5 or earlier, which have already been processed). To start using the GTID protocol so that you can later perform automatic fail-over, execute the following on each slave. Optionally, if you use multi-source replication, do this for each channel and include the `FOR CHANNEL` *`channel`* clause:

   ```
   STOP SLAVE [FOR CHANNEL 'channel'];
   CHANGE MASTER TO MASTER_AUTO_POSITION = 1 [FOR CHANNEL 'channel'];
   START SLAVE [FOR CHANNEL 'channel'];
   ```

17.1.5.3 Disabling GTID Transactions Online

This section describes how to disable GTID transactions on servers that are already online. This procedure does not require taking the server offline and is suited to use in production. However, if you have the possibility to take the servers offline when disabling GTIDs mode that process is easier.

The process is similar to enabling GTID transactions while the server is online, but reversing the steps. The only thing that differs is the point at which you wait for logged transactions to replicate.

Before you start, ensure that the servers meet the following pre-conditions:

- *All* servers in your topology must use MySQL 5.7.6 or later. You cannot disable GTID transactions online on any single server unless *all* servers which are in the topology are using this version.
- All servers have `gtid_mode` set to `ON`.
- The `--replicate-same-server-id` option is not set on any server. You cannot disable GTID transactions if this option is set together with the `--log-slave-updates` option (which is the default) and binary logging is enabled (which is also the default). Without GTIDs, this combination of options causes infinite loops in circular replication.

1. Execute the following on each slave, and if you using multi-source replication, do it for each channel and include the `FOR CHANNEL` channel clause:

   ```
   STOP SLAVE [FOR CHANNEL 'channel'];
   ```

```
CHANGE MASTER TO MASTER_AUTO_POSITION = 0, MASTER_LOG_FILE = file, \
MASTER_LOG_POS = position [FOR CHANNEL 'channel'];
START SLAVE [FOR CHANNEL 'channel'];
```

2. On each server, execute:

```
SET @@GLOBAL.GTID_MODE = ON_PERMISSIVE;
```

3. On each server, execute:

```
SET @@GLOBAL.GTID_MODE = OFF_PERMISSIVE;
```

4. On each server, wait until the variable @@GLOBAL.GTID_OWNED is equal to the empty string. This can be checked using:

```
SELECT @@GLOBAL.GTID_OWNED;
```

   On a replication slave, it is theoretically possible that this is empty and then nonempty again. This is not a problem, it suffices that it is empty once.

5. Wait for all transactions that currently exist in any binary log to replicate to all slaves. See Section 17.1.5.4, “Verifying Replication of Anonymous Transactions” for one method of checking that all anonymous transactions have replicated to all servers.

6. If you use binary logs for anything else than replication, for example to do point in time backup or restore: wait until you do not need the old binary logs having GTID transactions.

   For instance, after step 5 has completed, you can execute `FLUSH LOGS` on the server where you are taking the backup. Then either explicitly take a backup or wait for the next iteration of any periodic backup routine you may have set up.

   Ideally, wait for the server to purge all binary logs that existed when step 5 was completed. Also wait for any backup taken before step 5 to expire.

   Important

   This is the one important point during this procedure. It is important to understand that logs containing GTID transactions cannot be used after the next step. Before proceeding you must be sure that GTID transactions do not exist anywhere in the topology.

7. On each server, execute:

```
SET @@GLOBAL.GTID_MODE = OFF;
```

8. On each server, set `gtid_mode=OFF` in `my.cnf`.

   If you want to set `enforce_gtid_consistency=OFF`, you can do so now. After setting it, you should add `enforce_gtid_consistency=OFF` to your configuration file.

If you want to downgrade to an earlier version of MySQL, you can do so now, using the normal downgrade procedure.

17.1.5.4 Verifying Replication of Anonymous Transactions

This section explains how to monitor a replication topology and verify that all anonymous transactions have been replicated. This is helpful when changing the replication mode online as you can verify that it is safe to change to GTID transactions.

There are several possible ways to wait for transactions to replicate:

The simplest method, which works regardless of your topology but relies on timing is as follows: if you are sure that the slave never lags more than N seconds, just wait for a bit more than N seconds. Or wait for a day, or whatever time period you consider safe for your deployment.

A safer method in the sense that it does not depend on timing: if you only have a master with one or more slaves, do the following:

1. On the master, execute:

   ```
   SHOW MASTER STATUS;
   ```

   Note down the values in the `File` and `Position` column.

2. On every slave, use the file and position information from the master to execute:

   ```
   SELECT MASTER_POS_WAIT(file, position);
   ```

If you have a master and multiple levels of slaves, or in other words you have slaves of slaves, repeat step 2 on each level, starting from the master, then all the direct slaves, then all the slaves of slaves, and so on.

If you use a circular replication topology where multiple servers may have write clients, perform step 2 for each master-slave connection, until you have completed the full circle. Repeat the whole process so that you do the full circle *twice*.

For example, suppose you have three servers A, B, and C, replicating in a circle so that A -> B -> C -> A. The procedure is then:

- Do step 1 on A and step 2 on B.
- Do step 1 on B and step 2 on C.
- Do step 1 on C and step 2 on A.
- Do step 1 on A and step 2 on B.
- Do step 1 on B and step 2 on C.
- Do step 1 on C and step 2 on A.

17.1.6 Replication and Binary Logging Options and Variables

The following sections contain information about `mysqld` options and server variables that are used in replication and for controlling the binary log. Options and variables for use on replication masters and replication slaves are covered separately, as are options and variables relating to binary logging and global transaction identifiers (GTIDs). A set of quick-reference tables providing basic information about these options and variables is also included.

Of particular importance is the `--server-id` option.

Property	**Value**
Command-Line Format	`--server-id=#`
System Variable	`server_id`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value (>= 8.0.3)	`1`
Default Value (<= 8.0.2)	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

Specifies the server ID. The `server_id` system variable is set to 1 by default. The server can be started with this default ID, but when binary logging is enabled, an informational message is issued if you did not specify a server ID explicitly using the `--server-id` option.

For servers that are used in a replication topology, you must specify a unique server ID for each replication server, in the range from 1 to $2^{32} - 1$. “Unique” means that each ID must be different from every other ID in use by any other replication master or slave. For additional information, see Section 17.1.6.2, “Replication Master Options and Variables”, and Section 17.1.6.3, “Replication Slave Options and Variables”.

If the server ID is set to 0, binary logging takes place, but a master with a server ID of 0 refuses any connections from slaves, and a slave with a server ID of 0 refuses to connect to a master. Note that although you can change the server ID dynamically to a nonzero value, doing so does not enable replication to start immediately. You must change the server ID and then restart the server to initialize the replication slave.

For more information, see Section 17.1.2.2, “Setting the Replication Slave Configuration”.

`server_uuid`

The MySQL server generates a true UUID in addition to the default or user-supplied server ID set in the `server_id` system variable. This is available as the global, read-only variable `server_uuid`.

Note

The presence of the `server_uuid` system variable does not change the requirement for setting a unique `--server-id` for each MySQL server as part of preparing and running MySQL replication, as described earlier in this section.

Property	Value
System Variable	`server_uuid`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

When starting, the MySQL server automatically obtains a UUID as follows:

1. Attempt to read and use the UUID written in the file *`data_dir`*`/auto.cnf` (where *`data_dir`* is the server's data directory).

2. If *`data_dir`*`/auto.cnf` is not found, generate a new UUID and save it to this file, creating the file if necessary.

The `auto.cnf` file has a format similar to that used for `my.cnf` or `my.ini` files. `auto.cnf` has only a single `[auto]` section containing a single `server_uuid` setting and value; the file's contents appear similar to what is shown here:

```
[auto]
server_uuid=8a94f357-aab4-11df-86ab-c80aa9429562
```

Important

The `auto.cnf` file is automatically generated; do not attempt to write or modify this file.

When using MySQL replication, masters and slaves know each other's UUIDs. The value of a slave's UUID can be seen in the output of `SHOW SLAVE HOSTS`. Once `START SLAVE` has been executed, the value of the master's UUID is available on the slave in the output of `SHOW SLAVE STATUS`.

Note

Issuing a `STOP SLAVE` or `RESET SLAVE` statement does *not* reset the master's UUID as used on the slave.

A server's `server_uuid` is also used in GTIDs for transactions originating on that server. For more information, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

When starting, the slave I/O thread generates an error and aborts if its master's UUID is equal to its own unless the `--replicate-same-server-id` option has been set. In addition, the slave I/O thread generates a warning if either of the following is true:

- No master having the expected `server_uuid` exists.
- The master's `server_uuid` has changed, although no `CHANGE MASTER TO` statement has ever been executed.

17.1.6.1 Replication and Binary Logging Option and Variable Reference

The following two lists provide basic information about the MySQL command-line options and system variables applicable to replication and the binary log.

The command-line options and system variables in the following list relate to replication masters and replication slaves. Section 17.1.6.2, “Replication Master Options and Variables”, provides more detailed information about options and variables relating to replication master servers. For more information about options and variables relating to replication slaves, see Section 17.1.6.3, “Replication Slave Options and Variables”.

- `abort-slave-event-count`: Option used by mysql-test for debugging and testing of replication
- `auto_increment_increment`: AUTO_INCREMENT columns are incremented by this value
- `auto_increment_offset`: Offset added to AUTO_INCREMENT columns
- `binlog_expire_logs_seconds`: Purge binary logs after this many seconds
- `binlog_gtid_simple_recovery`: Controls how binary logs are iterated during GTID recovery
- `Com_change_master`: Count of CHANGE MASTER TO statements
- `Com_show_master_status`: Count of SHOW MASTER STATUS statements
- `Com_show_slave_hosts`: Count of SHOW SLAVE HOSTS statements
- `Com_show_slave_status`: Count of SHOW SLAVE STATUS statements
- `Com_slave_start`: Count of START SLAVE statements

- `Com_slave_stop`: Count of STOP SLAVE statements
- `disconnect-slave-event-count`: Option used by mysql-test for debugging and testing of replication
- `enforce_gtid_consistency`: Prevents execution of statements that cannot be logged in a transactionally safe manner
- `expire_logs_days`: Purge binary logs after this many days
- `gtid-executed-compression-period`: Compress gtid_executed table each time this many transactions have occurred. 0 means never compress this table. Applies only when binary logging is disabled.
- `gtid_executed`: Global: All GTIDs in the binary log (global) or current transaction (session). Read-only.
- `gtid_executed_compression_period`: Compress gtid_executed table each time this many transactions have occurred. 0 means never compress this table. Applies only when binary logging is disabled.
- `gtid_mode`: Controls whether GTID based logging is enabled and what type of transactions the logs can contain
- `gtid_next`: Specifies the GTID for the next statement to execute; see documentation for details
- `gtid_owned`: The set of GTIDs owned by this client (session), or by all clients, together with the thread ID of the owner (global). Read-only.
- `gtid_purged`: The set of all GTIDs that have been purged from the binary log
- `init_slave`: Statements that are executed when a slave connects to a master
- `log_bin_trust_function_creators`: If equal to 0 (the default), then when --log-bin is used, creation of a stored function is allowed only to users having the SUPER privilege and only if the function created does not break binary logging
- `log_builtin_as_identified_by_password`: Whether to log CREATE/ALTER USER, GRANT in backward-compatible fashion
- `log_statements_unsafe_for_binlog`: Disables error 1592 warnings being written to the error log
- `master-info-file`: The location and name of the file that remembers the master and where the I/O replication thread is in the master's binary logs
- `master-info-repository`: Whether to write master status information and replication I/O thread location in the master's binary logs to a file or table
- `master-retry-count`: Number of tries the slave makes to connect to the master before giving up
- `master_info_repository`: Whether to write master status information and replication I/O thread location in the master's binary logs to a file or table
- `max_relay_log_size`: If nonzero, relay log is rotated automatically when its size exceeds this value. If zero, size at which rotation occurs is determined by the value of max_binlog_size.
- `original_commit_timestamp`: The time when a transaction was committed on the original master
- `immediate_server_version`: The MySQL Server release number of the server that is the immediate master in a replication topology

- `original_server_version`: The MySQL Server release number of the server where a transaction was originally committed
- `relay-log-info-repository`: Whether to write the replication SQL thread's location in the relay logs to a file or a table
- `relay-log-recovery`: Enables automatic recovery of relay log files from master at startup
- `relay_log`: The location and base name to use for relay logs
- `relay_log_basename`: Complete path to relay log, including file name
- `relay_log_index`: The location and name to use for the file that keeps a list of the last relay logs
- `relay_log_info_file`: File in which the slave records information about the relay logs
- `relay_log_info_repository`: Whether to write the replication SQL thread's location in the relay logs to a file or a table
- `relay_log_purge`: Determines whether relay logs are purged
- `relay_log_recovery`: Whether automatic recovery of relay log files from master at startup is enabled; must be enabled for a crash-safe slave
- `relay_log_space_limit`: Maximum space to use for all relay logs
- `replicate-do-db`: Tells the slave SQL thread to restrict replication to the specified database
- `replicate-do-table`: Tells the slave SQL thread to restrict replication to the specified table
- `replicate-ignore-db`: Tells the slave SQL thread not to replicate to the specified database
- `replicate-ignore-table`: Tells the slave SQL thread not to replicate to the specified table
- `replicate-rewrite-db`: Updates to a database with a different name than the original
- `replicate-same-server-id`: In replication, if enabled, do not skip events having our server id
- `replicate-wild-do-table`: Tells the slave thread to restrict replication to the tables that match the specified wildcard pattern
- `replicate-wild-ignore-table`: Tells the slave thread not to replicate to the tables that match the given wildcard pattern
- `report_host`: Host name or IP of the slave to be reported to the master during slave registration
- `report_password`: An arbitrary password that the slave server should report to the master. Not the same as the password for the MySQL replication user account.
- `report_port`: Port for connecting to slave reported to the master during slave registration
- `report_user`: An arbitrary user name that a slave server should report to the master. Not the same as the name used with the MySQL replication user account.
- `Rpl_semi_sync_master_clients`: Number of semisynchronous slaves
- `rpl_semi_sync_master_enabled`: Whether semisynchronous replication is enabled on the master
- `Rpl_semi_sync_master_net_avg_wait_time`: The average time the master waited for a slave reply
- `Rpl_semi_sync_master_net_wait_time`: The total time the master waited for slave replies

- `Rpl_semi_sync_master_net_waits`: The total number of times the master waited for slave replies
- `Rpl_semi_sync_master_no_times`: Number of times the master turned off semisynchronous replication
- `Rpl_semi_sync_master_no_tx`: Number of commits not acknowledged successfully
- `Rpl_semi_sync_master_status`: Whether semisynchronous replication is operational on the master
- `Rpl_semi_sync_master_timefunc_failures`: Number of times the master failed when calling time functions
- `rpl_semi_sync_master_timeout`: Number of milliseconds to wait for slave acknowledgment
- `rpl_semi_sync_master_trace_level`: The semisynchronous replication debug trace level on the master
- `Rpl_semi_sync_master_tx_avg_wait_time`: The average time the master waited for each transaction
- `Rpl_semi_sync_master_tx_wait_time`: The total time the master waited for transactions
- `Rpl_semi_sync_master_tx_waits`: The total number of times the master waited for transactions
- `rpl_semi_sync_master_wait_for_slave_count`: How many slave acknowledgments the master must receive per transaction before proceeding
- `rpl_semi_sync_master_wait_no_slave`: Whether master waits for timeout even with no slaves
- `rpl_semi_sync_master_wait_point`: The wait point for slave transaction receipt acknowledgment
- `Rpl_semi_sync_master_wait_pos_backtraverse`: The total number of times the master waited for an event with binary coordinates lower than events waited for previously
- `Rpl_semi_sync_master_wait_sessions`: Number of sessions currently waiting for slave replies
- `Rpl_semi_sync_master_yes_tx`: Number of commits acknowledged successfully
- `rpl_semi_sync_slave_enabled`: Whether semisynchronous replication is enabled on slave
- `Rpl_semi_sync_slave_status`: Whether semisynchronous replication is operational on slave
- `rpl_semi_sync_slave_trace_level`: The semisynchronous replication debug trace level on the slave
- `rpl_read_size`: Set the minimum amount of data in bytes that is read from the binary log files and relay log files
- `rpl_stop_slave_timeout`: Set the number of seconds that STOP SLAVE waits before timing out
- `server_uuid`: The server's globally unique ID, automatically (re)generated at server start
- `show-slave-auth-info`: Show user name and password in SHOW SLAVE HOSTS on this master
- `skip-slave-start`: If set, slave is not autostarted
- `slave-checkpoint-group`: Maximum number of transactions processed by a multithreaded slave before a checkpoint operation is called to update progress status. Not supported by NDB Cluster.
- `slave-checkpoint-period`: Update progress status of multithreaded slave and flush relay log info to disk after this number of milliseconds. Not supported by NDB Cluster.

- `slave-load-tmpdir`: The location where the slave should put its temporary files when replicating a LOAD DATA statement
- `slave-max-allowed-packet`: Maximum size, in bytes, of a packet that can be sent from a replication master to a slave; overrides max_allowed_packet
- `slave-net-timeout`: Number of seconds to wait for more data from a master/slave connection before aborting the read
- `slave-parallel-type`: Tells the slave to use timestamp information (LOGICAL_CLOCK) or database partioning (DATABASE) to parallelize transactions. The default is LOGICAL_CLOCK.
- `slave-parallel-workers`: Number of applier threads for executing replication transactions in parallel. The default is 4 applier threads. Set to 0 to disable slave multithreading. Not supported by MySQL Cluster.
- `slave-pending-jobs-size-max`: Maximum size of slave worker queues holding events not yet applied
- `slave-rows-search-algorithms`: Determines search algorithms used for slave update batching. Any 2 or 3 from the list INDEX_SEARCH, TABLE_SCAN, HASH_SCAN
- `slave-skip-errors`: Tells the slave thread to continue replication when a query returns an error from the provided list
- `slave_checkpoint_group`: Maximum number of transactions processed by a multithreaded slave before a checkpoint operation is called to update progress status. Not supported by NDB Cluster.
- `slave_checkpoint_period`: Update progress status of multithreaded slave and flush relay log info to disk after this number of milliseconds. Not supported by NDB Cluster.
- `slave_compressed_protocol`: Use compression of master/slave protocol
- `slave_exec_mode`: Allows for switching the slave thread between IDEMPOTENT mode (key and some other errors suppressed) and STRICT mode; STRICT mode is the default, except for NDB Cluster, where IDEMPOTENT is always used
- `Slave_heartbeat_period`: The slave's replication heartbeat interval, in seconds
- `Slave_last_heartbeat`: Shows when the latest heartbeat signal was received, in TIMESTAMP format
- `slave_max_allowed_packet`: Maximum size, in bytes, of a packet that can be sent from a replication master to a slave; overrides max_allowed_packet
- `Slave_open_temp_tables`: Number of temporary tables that the slave SQL thread currently has open
- `slave_parallel_type`: Tells the slave to use timestamp information (LOGICAL_CLOCK) or database partioning (DATABASE) to parallelize transactions.
- `slave_parallel_workers`: Number of applier threads for executing replication transactions in parallel. A value of 0 disables slave multithreading. Not supported by MySQL Cluster.
- `slave_pending_jobs_size_max`: Maximum size of slave worker queues holding events not yet applied
- `slave_preserve_commit_order`: Ensures that all commits by slave workers happen in the same order as on the master to maintain consistency when using parallel applier threads.

- `Slave_received_heartbeats`: Number of heartbeats received by a replication slave since previous reset
- `Slave_retried_transactions`: The total number of times since startup that the replication slave SQL thread has retried transactions
- `slave_rows_search_algorithms`: Determines search algorithms used for slave update batching. Any 2 or 3 from the list INDEX_SEARCH, TABLE_SCAN, HASH_SCAN.
- `Slave_rows_last_search_algorithm_used`: Search algorithm most recently used by this slave to locate rows for row-based replication (index, table, or hash scan)
- `Slave_running`: The state of this server as a replication slave (slave I/O thread status)
- `slave_transaction_retries`: Number of times the slave SQL thread will retry a transaction in case it failed with a deadlock or elapsed lock wait timeout, before giving up and stopping
- `slave_type_conversions`: Controls type conversion mode on replication slave. Value is a list of zero or more elements from the list: ALL_LOSSY, ALL_NON_LOSSY. Set to an empty string to disallow type conversions between master and slave.
- `sql_log_bin`: Controls binary logging for the current session
- `sql_slave_skip_counter`: Number of events from the master that a slave server should skip. Not compatible with GTID replication.
- `sync_master_info`: Synchronize master.info to disk after every #th event
- `sync_relay_log`: Synchronize relay log to disk after every #th event
- `sync_relay_log_info`: Synchronize relay.info file to disk after every #th event
- `transaction_write_set_extraction`: Defines the algorithm used to hash the writes extracted during a transaction

The command-line options and system variables in the following list relate to the binary log. Section 17.1.6.4, “Binary Logging Options and Variables”, provides more detailed information about options and variables relating to binary logging. For additional general information about the binary log, see Section 5.4.4, “The Binary Log”.

- `binlog-checksum`: Enable/disable binary log checksums
- `binlog-do-db`: Limits binary logging to specific databases
- `binlog_format`: Specifies the format of the binary log
- `binlog-ignore-db`: Tells the master that updates to the given database should not be logged to the binary log
- `binlog-row-event-max-size`: Binary log max event size
- `binlog_encryption`: Enable encryption for binary log files and relay log files on this server
- `binlog_rotate_encryption_master_key_at_startup`: Rotate the binary log master key at server startup
- `Binlog_cache_disk_use`: Number of transactions that used a temporary file instead of the binary log cache

- `binlog_cache_size`: Size of the cache to hold the SQL statements for the binary log during a transaction
- `Binlog_cache_use`: Number of transactions that used the temporary binary log cache
- `binlog_checksum`: Enable/disable binary log checksums
- `binlog_direct_non_transactional_updates`: Causes updates using statement format to nontransactional engines to be written directly to binary log. See documentation before using.
- `binlog_error_action`: Controls what happens when the server cannot write to the binary log
- `binlog_group_commit_sync_delay`: Sets the number of microseconds to wait before synchronizing transactions to disk
- `binlog_group_commit_sync_no_delay_count`: Sets the maximum number of transactions to wait for before aborting the current delay specified by binlog_group_commit_sync_delay
- `binlog_max_flush_queue_time`: How long to read transactions before flushing to binary log
- `binlog_order_commits`: Whether to commit in same order as writes to binary log
- `binlog_row_image`: Use full or minimal images when logging row changes
- `binlog_row_metadata`: Configures the amount of table related metadata binary logged when using row-based logging.
- `binlog_row_value_options`: Enables binary logging of partial JSON updates for row-based replication.
- `binlog_rows_query_log_events`: When enabled, enables logging of rows query log events when using row-based logging. Disabled by default. Do not enable when producing logs for pre-5.6 slaves/ readers.
- `Binlog_stmt_cache_disk_use`: Number of nontransactional statements that used a temporary file instead of the binary log statement cache
- `binlog_stmt_cache_size`: Size of the cache to hold nontransactional statements for the binary log during a transaction
- `Binlog_stmt_cache_use`: Number of statements that used the temporary binary log statement cache
- `binlog_transaction_dependency_tracking`: Source of dependency information (commit timestamps or transaction write sets) from which to assess which transactions can be executed in parallel by slave's multithreaded applier.
- `binlog_transaction_dependency_history_size`: Number of row hashes kept for looking up transaction that last updated some row.
- `Com_show_binlog_events`: Count of SHOW BINLOG EVENTS statements
- `Com_show_binlogs`: Count of SHOW BINLOGS statements
- `log-bin`: Specifies the base name for binary log files
- `log-bin-index`: Specifies the name for the binary log index file
- `log_bin`: Whether the binary log is enabled
- `log_bin_basename`: Path and base name for binary log files

- `log_bin_use_v1_row_events`: Whether server is using version 1 binary log row events
- `log_slave_updates`: Whether the slave should log the updates performed by its SQL thread to its own binary log
- `master-verify-checksum`: Cause master to examine checksums when reading from the binary log
- `master_verify_checksum`: Cause master to read checksums from binary log
- `max-binlog-dump-events`: Option used by mysql-test for debugging and testing of replication
- `max_binlog_cache_size`: Can be used to restrict the total size used to cache a multi-statement transaction
- `max_binlog_size`: Binary log will be rotated automatically when size exceeds this value
- `max_binlog_stmt_cache_size`: Can be used to restrict the total size used to cache all nontransactional statements during a transaction
- `slave-sql-verify-checksum`: Cause slave to examine checksums when reading from the relay log
- `slave_sql_verify_checksum`: Cause slave to examine checksums when reading from relay log
- `sporadic-binlog-dump-fail`: Option used by mysql-test for debugging and testing of replication
- `sync_binlog`: Synchronously flush binary log to disk after every #th event

For a listing of *all* command-line options, system and status variables used with `mysqld`, see Section 5.1.4, “Server Option, System Variable, and Status Variable Reference”.

17.1.6.2 Replication Master Options and Variables

This section describes the server options and system variables that you can use on replication master servers. You can specify the options either on the command line or in an option file. You can specify system variable values using `SET`.

On the master and each slave, you must use the `server-id` option to establish a unique replication ID. For each server, you should pick a unique positive integer in the range from 1 to $2^{32} - 1$, and each ID must be different from every other ID in use by any other replication master or slave. Example: `server-id=3`.

For options used on the master for controlling binary logging, see Section 17.1.6.4, “Binary Logging Options and Variables”.

Startup Options for Replication Masters

The following list describes startup options for controlling replication master servers. Replication-related system variables are discussed later in this section.

- `--show-slave-auth-info`

Property	Value
Command-Line Format	`--show-slave-auth-info[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Display slave user names and passwords in the output of `SHOW SLAVE HOSTS` on the master server for slaves started with the `--report-user` and `--report-password` options.

System Variables Used on Replication Masters

The following system variables are used for or by replication masters:

- auto_increment_increment

Property	Value
Command-Line Format	--auto-increment-increment=#
System Variable	auto_increment_increment
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	1
Minimum Value	1
Maximum Value	65535

auto_increment_increment and auto_increment_offset are intended for use with master-to-master replication, and can be used to control the operation of AUTO_INCREMENT columns. Both variables have global and session values, and each can assume an integer value between 1 and 65,535 inclusive. Setting the value of either of these two variables to 0 causes its value to be set to 1 instead. Attempting to set the value of either of these two variables to an integer greater than 65,535 or less than 0 causes its value to be set to 65,535 instead. Attempting to set the value of auto_increment_increment or auto_increment_offset to a noninteger value produces an error, and the actual value of the variable remains unchanged.

Note

auto_increment_increment is also supported for use with NDB tables.

As of MySQL 8.0.18, setting the session value of this system variable is no longer a restricted operation.

When Group Replication is started on a server, the value of auto_increment_increment is changed to the value of group_replication_auto_increment_increment, which defaults to 7, and the value of auto_increment_offset is changed to the server ID. The changes are reverted when Group Replication is stopped. These changes are only made and reverted if auto_increment_increment and auto_increment_offset each have their default value of 1. If their values have already been modified from the default, Group Replication does not alter them. From MySQL 8.0, the system variables are also not modified when Group Replication is in single-primary mode, where only one server writes.

auto_increment_increment and auto_increment_offset affect AUTO_INCREMENT column behavior as follows:

- auto_increment_increment controls the interval between successive column values. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
```

```
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc1
    -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
  Query OK, 0 rows affected (0.04 sec)

mysql> SET @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.01 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
4 rows in set (0.00 sec)
```

- `auto_increment_offset` determines the starting point for the `AUTO_INCREMENT` column value. Consider the following, assuming that these statements are executed during the same session as the example given in the description for `auto_increment_increment`:

```
mysql> SET @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc2
    -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc2;
+-----+
| col |
+-----+
|   5 |
|  15 |
|  25 |
```

```
|  35 |
+-----+
4 rows in set (0.02 sec)
```

When the value of `auto_increment_offset` is greater than that of `auto_increment_increment`, the value of `auto_increment_offset` is ignored.

If either of these variables is changed, and then new rows inserted into a table containing an `AUTO_INCREMENT` column, the results may seem counterintuitive because the series of `AUTO_INCREMENT` values is calculated without regard to any values already present in the column, and the next value inserted is the least value in the series that is greater than the maximum existing value in the `AUTO_INCREMENT` column. The series is calculated like this:

`auto_increment_offset` + *N* × `auto_increment_increment`

where *N* is a positive integer value in the series [1, 2, 3, ...]. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
4 rows in set (0.00 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
|  35 |
|  45 |
|  55 |
|  65 |
+-----+
8 rows in set (0.00 sec)
```

The values shown for `auto_increment_increment` and `auto_increment_offset` generate the series 5 + *N* × 10, that is, [5, 15, 25, 35, 45, ...]. The highest value present in the `col` column prior to the `INSERT` is 31, and the next available value in the `AUTO_INCREMENT` series is 35, so the inserted values for `col` begin at that point and the results are as shown for the `SELECT` query.

It is not possible to restrict the effects of these two variables to a single table; these variables control the behavior of all `AUTO_INCREMENT` columns in *all* tables on the MySQL server. If the global value of either variable is set, its effects persist until the global value is changed or overridden by setting the session value, or until `mysqld` is restarted. If the local value is set, the new value affects `AUTO_INCREMENT` columns for all tables into which new rows are inserted by the current user for the duration of the session, unless the values are changed during that session.

The default value of `auto_increment_increment` is 1. See Section 17.5.1.1, “Replication and AUTO_INCREMENT”.

- `auto_increment_offset`

Property	Value
Command-Line Format	`--auto-increment-offset=#`
System Variable	`auto_increment_offset`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	Yes
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`65535`

This variable has a default value of 1. If it is left with its default value, and Group Replication is started on the server in multi-primary mode, it is changed to the server ID. For more information, see the description for `auto_increment_increment`.

Note

`auto_increment_offset` is also supported for use with `NDB` tables.

As of MySQL 8.0.18, setting the session value of this system variable is no longer a restricted operation.

- `immediate_server_version`

Property	Value
Introduced	8.0.14
System Variable	`immediate_server_version`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

For internal use by replication. This session system variable holds the MySQL Server release number of the server that is the immediate master in a replication topology (for example, `80014` for a MySQL 8.0.14 server instance). If this immediate server is at a release that does not support the session system variable, the value of the variable is set to 0 (`UNKNOWN_SERVER_VERSION`).

The value of the variable is replicated from a master to a slave. With this information the slave can correctly process data originating from a master at an older release, by recognizing where syntax changes or semantic changes have occurred between the releases involved and handling these appropriately. The information can also be used in a Group Replication environment where one or more members of the replication group is at a newer release than the others. The value of the variable can be viewed in the binary log for each transaction (as part of the `Gtid_log_event`, or `Anonymous_gtid_log_event` if GTIDs are not in use on the server), and could be helpful in debugging cross-version replication issues.

Setting the session value of this system variable is a restricted operation. The session user must have either the `REPLICATION_APPLIER` privilege (see Section 17.3.3, "Replication Privilege Checks"), or privileges sufficient to set restricted session variables (see Section 5.1.9.1, "System Variable Privileges"). However, note that the variable is not intended for users to set; it is set automatically by the replication infrastructure.

- `original_server_version`

Property	Value
Introduced	8.0.14
System Variable	`original_server_version`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

For internal use by replication. This session system variable holds the MySQL Server release number of the server where a transaction was originally committed (for example, `80014` for a MySQL 8.0.14 server instance). If this original server is at a release that does not support the session system variable, the value of the variable is set to 0 (`UNKNOWN_SERVER_VERSION`). Note that when a release number is set by the original server, the value of the variable is reset to 0 if the immediate server or any other intervening server in the replication topology does not support the session system variable, and so does not replicate its value.

The value of the variable is set and used in the same ways as for the `immediate_server_version` system variable. If the value of the variable is the same as that for the `immediate_server_version` system variable, only the latter is recorded in the binary log, with an indicator that the original server version is the same.

In a Group Replication environment, view change log events, which are special transactions queued by each group member when a new member joins the group, are tagged with the server version of the group member queuing the transaction. This ensures that the server version of the original donor is known to the joining member. Because the view change log events queued for a particular view change have the same GTID on all members, for this case only, instances of the same GTID might have a different original server version.

Setting the session value of this system variable is a restricted operation. The session user must have either the `REPLICATION_APPLIER` privilege (see Section 17.3.3, "Replication Privilege Checks"), or privileges sufficient to set restricted session variables (see Section 5.1.9.1, "System Variable Privileges"). However, note that the variable is not intended for users to set; it is set automatically by the replication infrastructure.

- `rpl_semi_sync_master_enabled`

Property	Value
Command-Line Format	`--rpl-semi-sync-master-enabled[={OFF|ON}]`
System Variable	`rpl_semi_sync_master_enabled`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Controls whether semisynchronous replication is enabled on the master. To enable or disable the plugin, set this variable to `ON` or `OFF` (or 1 or 0), respectively. The default is `OFF`.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_master_timeout`

Property	Value
Command-Line Format	`--rpl-semi-sync-master-timeout=#`
System Variable	`rpl_semi_sync_master_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10000`

A value in milliseconds that controls how long the master waits on a commit for acknowledgment from a slave before timing out and reverting to asynchronous replication. The default value is 10000 (10 seconds).

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_master_trace_level`

Property	Value
Command-Line Format	`--rpl-semi-sync-master-trace-level=#`
System Variable	`rpl_semi_sync_master_trace_level`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`32`

The semisynchronous replication debug trace level on the master. Four levels are defined:

- 1 = general level (for example, time function failures)

- 16 = detail level (more verbose information)
- 32 = net wait level (more information about network waits)
- 64 = function level (information about function entry and exit)

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_master_wait_for_slave_count`

Property	Value
Command-Line Format	`--rpl-semi-sync-master-wait-for-slave-count=#`
System Variable	`rpl_semi_sync_master_wait_for_slave_count`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`65535`

The number of slave acknowledgments the master must receive per transaction before proceeding. By default `rpl_semi_sync_master_wait_for_slave_count` is `1`, meaning that semisynchronous replication proceeds after receiving a single slave acknowledgment. Performance is best for small values of this variable.

For example, if `rpl_semi_sync_master_wait_for_slave_count` is `2`, then 2 slaves must acknowledge receipt of the transaction before the timeout period configured by `rpl_semi_sync_master_timeout` for semisynchronous replication to proceed. If less slaves acknowledge receipt of the transaction during the timeout period, the master reverts to normal replication.

Note

This behavior also depends on `rpl_semi_sync_master_wait_no_slave`

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_master_wait_no_slave`

Property	Value
Command-Line Format	`--rpl-semi-sync-master-wait-no-slave[={OFF|ON}]`
System Variable	`rpl_semi_sync_master_wait_no_slave`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Controls whether the master waits for the timeout period configured by `rpl_semi_sync_master_timeout` to expire, even if the slave count drops to less than the number of slaves configured by `rpl_semi_sync_master_wait_for_slave_count` during the timeout period.

When the value of `rpl_semi_sync_master_wait_no_slave` is `ON` (the default), it is permissible for the slave count to drop to less than `rpl_semi_sync_master_wait_for_slave_count` during the timeout period. As long as enough slaves acknowledge the transaction before the timeout period expires, semisynchronous replication continues.

When the value of `rpl_semi_sync_master_wait_no_slave` is `OFF`, if the slave count drops to less than the number configured in `rpl_semi_sync_master_wait_for_slave_count` at any time during the timeout period configured by `rpl_semi_sync_master_timeout`, the master reverts to normal replication.

This variable is available only if the master-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_master_wait_point`

Property	Value
Command-Line Format	`--rpl-semi-sync-master-wait-point=value`
System Variable	`rpl_semi_sync_master_wait_point`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`AFTER_SYNC`
Valid Values	`AFTER_SYNC` `AFTER_COMMIT`

This variable controls the point at which a semisynchronous replication master waits for slave acknowledgment of transaction receipt before returning a status to the client that committed the transaction. These values are permitted:

- `AFTER_SYNC` (the default): The master writes each transaction to its binary log and the slave, and syncs the binary log to disk. The master waits for slave acknowledgment of transaction receipt after the sync. Upon receiving acknowledgment, the master commits the transaction to the storage engine and returns a result to the client, which then can proceed.
- `AFTER_COMMIT`: The master writes each transaction to its binary log and the slave, syncs the binary log, and commits the transaction to the storage engine. The master waits for slave acknowledgment of transaction receipt after the commit. Upon receiving acknowledgment, the master returns a result to the client, which then can proceed.

The replication characteristics of these settings differ as follows:

- With `AFTER_SYNC`, all clients see the committed transaction at the same time: After it has been acknowledged by the slave and committed to the storage engine on the master. Thus, all clients see the same data on the master.

In the event of master failure, all transactions committed on the master have been replicated to the slave (saved to its relay log). A crash of the master and failover to the slave is lossless because the slave is up to date. Note, however, that the master cannot be restarted in this scenario and must be discarded, because its binary log might contain uncommitted transactions that would cause a conflict with the slave when externalized after binary log recovery.

- With `AFTER_COMMIT`, the client issuing the transaction gets a return status only after the server commits to the storage engine and receives slave acknowledgment. After the commit and before slave acknowledgment, other clients can see the committed transaction before the committing client.

  If something goes wrong such that the slave does not process the transaction, then in the event of a master crash and failover to the slave, it is possible that such clients will see a loss of data relative to what they saw on the master.

This variable is available only if the master-side semisynchronous replication plugin is installed.

With the addition of `rpl_semi_sync_master_wait_point` in MySQL 5.7, a version compatibility constraint was created because it increments the semisynchronous interface version: Servers for MySQL 5.7 and higher do not work with semisynchronous replication plugins from older versions, nor do servers from older versions work with semisynchronous replication plugins for MySQL 5.7 and higher.

17.1.6.3 Replication Slave Options and Variables

This section explains the server options and system variables that apply to slave replication servers and contains the following:

- Startup Options for Replication Slaves
- Options for Logging Slave Status to Tables
- System Variables Used on Replication Slaves

Specify the options either on the command line or in an option file. Many of the options can be set while the server is running by using the `CHANGE MASTER TO` statement. Specify system variable values using `SET`.

Server ID. On the master and each slave, you must use the `server-id` option to establish a unique replication ID in the range from 1 to $2^{32} - 1$. “Unique” means that each ID must be different from every other ID in use by any other replication master or slave. Example `my.cnf` file:

```
[mysqld]
server-id=3
```

Startup Options for Replication Slaves

This section explains startup options for controlling replication slave servers. Many of these options can be set while the server is running by using the `CHANGE MASTER TO` statement. Others, such as the `--replicate-*` options, can be set only when the slave server starts. Replication-related system variables are discussed later in this section.

- `--master-info-file=`*`file_name`*

Property	Value
Command-Line Format	`--master-info-file=file_name`
Deprecated	8.0.18
Type	File name

Property	Value
Default Value	`master.info`

The name for the master info log, if `--master-info-repository=FILE` is set. The default name is `master.info` in the data directory. `--master-info-file` and the setting `--master-info-repository=FILE` are deprecated, as the use of a file for the master info log has been superseded by crash-safe slave tables. For information about the master info log, see Section 17.2.4.2, “Slave Status Logs”.

- `--master-retry-count=`*`count`*

Property	Value
Command-Line Format	`--master-retry-count=#`
Deprecated	Yes
Type	Integer
Default Value	`86400`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The number of times that the slave tries to reconnect to the master before giving up. The default value is 86400 times. A value of 0 means “infinite”, and the slave attempts to connect forever. Reconnection attempts are triggered when the slave reaches its connection timeout (specified by the `--slave-net-timeout` option) without receiving data or a heartbeat signal from the master. Reconnection is attempted at intervals set by the `MASTER_CONNECT_RETRY` option of the `CHANGE MASTER TO` statement (which defaults to every 60 seconds).

This option is deprecated and will be removed in a future MySQL release. Use the `MASTER_RETRY_COUNT` option of the `CHANGE MASTER TO` statement instead.

- `--max-relay-log-size=`*`size`*

Property	Value
Command-Line Format	`--max-relay-log-size=#`
System Variable	`max_relay_log_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1073741824`

The size at which the server rotates relay log files automatically. If this value is nonzero, the relay log is rotated automatically when its size exceeds this value. If this value is zero (the default), the size at which

relay log rotation occurs is determined by the value of `max_binlog_size`. For more information, see Section 17.2.4.1, “The Slave Relay Log”.

- `--relay-log-purge={0|1}`

Property	Value
Command-Line Format	`--relay-log-purge[={OFF|ON}]`
System Variable	`relay_log_purge`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Disable or enable automatic purging of relay logs as soon as they are no longer needed. The default value is 1 (enabled). This is a global variable that can be changed dynamically with `SET GLOBAL relay_log_purge = N`. Disabling purging of relay logs when using the `--relay-log-recovery` option risks data consistency and is therefore not crash-safe.

- `--relay-log-recovery={0|1}`

Property	Value
Command-Line Format	`--relay-log-recovery[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Enables automatic relay log recovery immediately following server startup. The recovery process creates a new relay log file, initializes the SQL thread position to this new relay log, and initializes the I/O thread to the SQL thread position. Reading of the relay log from the master then continues. This should be used following a crash on the replication slave to ensure that no possibly corrupted relay logs are processed. The default value is 0 (disabled).

To provide a crash-safe slave, this option must be enabled (set to 1), `--relay-log-info-repository` must be set to `TABLE`, and `relay-log-purge` must be enabled. Enabling the `--relay-log-recovery` option when `relay-log-purge` is disabled risks reading the relay log from files that were not purged, leading to data inconsistency, and is therefore not crash-safe. See Making replication resilient to unexpected halts, for more information.

When using a multithreaded slave (in other words `slave_parallel_workers` is greater than 0), inconsistencies such as gaps can occur in the sequence of transactions that have been executed from the relay log. Enabling the `--relay-log-recovery` option when there are inconsistencies causes an error and the option has no effect. The solution in this situation is to issue `START SLAVE UNTIL SQL_AFTER_MTS_GAPS`, which brings the server to a more consistent state, then issue `RESET SLAVE` to remove the relay logs. See Section 17.5.1.33, “Replication and Transaction Inconsistencies” for more information.

Note

This option does not affect the following Group Replication channels:

- `group_replication_applier`

  - `group_replication_recovery`

  Any other channels running on a group are affected, such as a channel which is replicating from an outside master or another group.

- `--relay-log-space-limit=size`

Property	Value
Command-Line Format	`--relay-log-space-limit=#`
System Variable	`relay_log_space_limit`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

This option places an upper limit on the total size in bytes of all relay logs on the slave. A value of 0 means “no limit”. This is useful for a slave server host that has limited disk space. When the limit is reached, the I/O thread stops reading binary log events from the master server until the SQL thread has caught up and deleted some unused relay logs. Note that this limit is not absolute: There are cases where the SQL thread needs more events before it can delete relay logs. In that case, the I/O thread exceeds the limit until it becomes possible for the SQL thread to delete some relay logs because not doing so would cause a deadlock. You should not set `--relay-log-space-limit` to less than twice the value of `--max-relay-log-size` (or `--max-binlog-size` if `--max-relay-log-size` is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` is exceeded, but the SQL thread has no relay log to purge and is unable to satisfy the I/O thread. This forces the I/O thread to ignore `--relay-log-space-limit` temporarily.

- `--replicate-do-db=db_name`

Property	Value
Command-Line Format	`--replicate-do-db=name`
Type	String

Creates a replication filter using the name of a database. Such filters can also be created using `CHANGE REPLICATION FILTER REPLICATE_DO_DB`.

This option supports channel specific replication filters, enabling multi-source replication slaves to use specific filters for different sources. To configure a channel specific replication filter on a channel named *`channel_1`* use `--replicate-do-db:channel_1:db_name`. In this case, the first colon is interpreted as a separator and subsequent colons are literal colons. See Section 17.2.5.4, “Replication Channel Based Filters” for more information.

Note

Global replication filters cannot be used on a MySQL server instance that is configured for Group Replication, because filtering transactions on some servers would make the group unable to reach agreement on a consistent state. Channel specific replication filters can be used on replication channels that are not directly involved with Group Replication, such as where a group member also acts as a replication slave to a master that is outside the group. They cannot be used on the `group_replication_applier` or `group_replication_recovery` channels.

The precise effect of this replication filter depends on whether statement-based or row-based replication is in use.

Statement-based replication. Tell the slave SQL thread to restrict replication to statements where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database, use this option multiple times, once for each database; however, doing so does *not* replicate cross-database statements such as `UPDATE some_db.some_table SET foo='bar'` while a different database (or no database) is selected.

Warning

To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, if you supply a comma separated list then the list is treated as the name of a single database.

An example of what does not work as you might expect when using statement-based replication: If the slave is started with `--replicate-do-db=sales` and you issue the following statements on the master, the `UPDATE` statement is *not* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “check just the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

Row-based replication. Tells the slave SQL thread to restrict replication to database *`db_name`*. Only tables belonging to *`db_name`* are changed; the current database has no effect on this. Suppose that the slave is started with `--replicate-do-db=sales` and row-based replication is in effect, and then the following statements are run on the master:

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

The `february` table in the `sales` database on the slave is changed in accordance with the `UPDATE` statement; this occurs whether or not the `USE` statement was issued. However, issuing the following statements on the master has no effect on the slave when using row-based replication and `--replicate-do-db=sales`:

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

Even if the statement `USE prices` were changed to `USE sales`, the `UPDATE` statement's effects would still not be replicated.

Another important difference in how `--replicate-do-db` is handled in statement-based replication as opposed to row-based replication occurs with regard to statements that refer to multiple databases. Suppose that the slave is started with `--replicate-do-db=db1`, and the following statements are executed on the master:

```
USE db1;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

If you are using statement-based replication, then both tables are updated on the slave. However, when using row-based replication, only `table1` is affected on the slave; since `table2` is in a different database, `table2` on the slave is not changed by the `UPDATE`. Now suppose that, instead of the `USE db1` statement, a `USE db4` statement had been used:

```
USE db4;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

In this case, the `UPDATE` statement would have no effect on the slave when using statement-based replication. However, if you are using row-based replication, the `UPDATE` would change `table1` on the slave, but not `table2`—in other words, only tables in the database named by `--replicate-do-db` are changed, and the choice of default database has no effect on this behavior.

If you need cross-database updates to work, use `--replicate-wild-do-table=db_name.%` instead. See Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”.

Note

This option affects replication in the same manner that `--binlog-do-db` affects binary logging, and the effects of the replication format on how `--replicate-do-db` affects replication behavior are the same as those of the logging format on the behavior of `--binlog-do-db`.

This option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements.

- `--replicate-ignore-db=db_name`

Property	Value
Command-Line Format	`--replicate-ignore-db=name`
Type	String

Creates a replication filter using the name of a database. Such filters can also be created using `CHANGE REPLICATION FILTER REPLICATE_IGNORE_DB`.

This option supports channel specific replication filters, enabling multi-source replication slaves to use specific filters for different sources. To configure a channel specific replication filter on a channel named *`channel_1`* use `--replicate-ignore-db:channel_1:db_name`. In this case, the first colon is interpreted as a separator and subsequent colons are literal colons. See Section 17.2.5.4, “Replication Channel Based Filters” for more information.

Note

Global replication filters cannot be used on a MySQL server instance that is configured for Group Replication, because filtering transactions on some servers would make the group unable to reach agreement on a consistent state. Channel specific replication filters can be used on replication channels that are not directly involved with Group Replication, such as where a group member also acts as a replication slave to a master that is outside the group. They cannot be used on the `group_replication_applier` or `group_replication_recovery` channels.

To specify more than one database to ignore, use this option multiple times, once for each database. Because database names can contain commas, if you supply a comma separated list then the list will be treated as the name of a single database.

As with `--replicate-do-db`, the precise effect of this filtering depends on whether statement-based or row-based replication is in use, and are described in the next several paragraphs.

Statement-based replication. Tells the slave SQL thread not to replicate any statement where the default database (that is, the one selected by `USE`) is *`db_name`*.

Row-based replication. Tells the slave SQL thread not to update any tables in the database *`db_name`*. The default database has no effect.

When using statement-based replication, the following example does not work as you might expect. Suppose that the slave is started with `--replicate-ignore-db=sales` and you issue the following statements on the master:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The `UPDATE` statement *is* replicated in such a case because `--replicate-ignore-db` applies only to the default database (determined by the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered. However, when using row-based replication, the `UPDATE` statement's effects are *not* propagated to the slave, and the slave's copy of the `sales.january` table is unchanged; in this instance, `--replicate-ignore-db=sales` causes *all* changes made to tables in the master's copy of the `sales` database to be ignored by the slave.

You should not use this option if you are using cross-database updates and you do not want these updates to be replicated. See Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”.

If you need cross-database updates to work, use `--replicate-wild-ignore-table=`*`db_name`*`.%` instead. See Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”.

Note

This option affects replication in the same manner that `--binlog-ignore-db` affects binary logging, and the effects of the replication format on how `--replicate-ignore-db` affects replication behavior are the same as those of the logging format on the behavior of `--binlog-ignore-db`.

This option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements.

- `--replicate-do-table=`*`db_name.tbl_name`*

Property	Value
Command-Line Format	`--replicate-do-table=name`
Type	String

Creates a replication filter by telling the slave SQL thread to restrict replication to a given table. To specify more than one table, use this option multiple times, once for each table. This works for both cross-database updates and default database updates, in contrast to `--replicate-do-db`. See Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”. You can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_DO_TABLE` statement.

This option supports channel specific replication filters, enabling multi-source replication slaves to use specific filters for different sources. To configure a channel specific replication filter on a channel named *`channel_1`* use `--replicate-do-table=`*`channel_1:db_name.tbl_name`*. In this case, the first colon is interpreted as a separator and subsequent colons are literal colons. See Section 17.2.5.4, “Replication Channel Based Filters” for more information.

Note

Global replication filters cannot be used on a MySQL server instance that is configured for Group Replication, because filtering transactions on some servers would make the group unable to reach agreement on a consistent state. Channel specific replication filters can be used on replication channels that are not directly involved with Group Replication, such as where a group member also acts as a replication slave to a master that is outside the group. They cannot be used on the `group_replication_applier` or `group_replication_recovery` channels.

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the `--replicate-*-db` options.

- `--replicate-ignore-table=`*`db_name.tbl_name`*

Property	Value
Command-Line Format	`--replicate-ignore-table=name`
Type	String

Creates a replication filter by telling the slave SQL thread not to replicate any statement that updates the specified table, even if any other tables might be updated by the same statement. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-ignore-db`. See Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”. You can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_IGNORE_TABLE` statement.

This option supports channel specific replication filters, enabling multi-source replication slaves to use specific filters for different sources. To configure a channel specific replication filter on a channel named *`channel_1`* use `--replicate-ignore-table=`*`channel_1:db_name.tbl_name`*. In this case, the first colon is interpreted as a separator and subsequent colons are literal colons. See Section 17.2.5.4, “Replication Channel Based Filters” for more information.

Note

Global replication filters cannot be used on a MySQL server instance that is configured for Group Replication, because filtering transactions on some servers would make the group unable to reach agreement on a consistent state. Channel specific replication filters can be used on replication channels that are not directly involved with Group Replication, such as where a group member also acts as a replication slave to a master that is outside the group. They cannot be used on the `group_replication_applier` or `group_replication_recovery` channels.

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the `--replicate-*-db` options.

- `--replicate-rewrite-db=`*`from_name->to_name`*

Property	Value
Command-Line Format	`--replicate-rewrite-db=old_name->new_name`
Type	String

Tells the slave to create a replication filter that translates the default database (that is, the one selected by `USE`) to *`to_name`* if it was *`from_name`* on the master. Only statements involving tables are affected (not statements such as `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`), and only if *`from_name`* is the default database on the master. To specify multiple rewrites, use this option multiple times. The server uses the first one with a *`from_name`* value that matches. The database name translation is done *before* the `--replicate-*` rules are tested. You can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB` statement.

If you use this option on the command line and the `>` character is special to your command interpreter, quote the option value. For example:

```
shell> mysqld --replicate-rewrite-db="olddb->newdb"
```

This option supports channel specific replication filters, enabling multi-source replication slaves to use specific filters for different sources. Specify the channel name followed by a colon, followed by the filter specification. The first colon is interpreted as a separator, and any subsequent colons are interpreted as literal colons. For example, to configure a channel specific replication filter on a channel named *`channel_1`*, use:

```
shell> mysqld --replicate-rewrite-db=channel_1:db_name1->db_name2
```

If you use a colon but do not specify a channel name, the option configures the replication filter for the default replication channel. See Section 17.2.5.4, “Replication Channel Based Filters” for more information.

Note

Global replication filters cannot be used on a MySQL server instance that is configured for Group Replication, because filtering transactions on some servers would make the group unable to reach agreement on a consistent state. Channel specific replication filters can be used on replication channels that are not directly involved with Group Replication, such as where a group member also acts as

a replication slave to a master that is outside the group. They cannot be used on the `group_replication_applier` or `group_replication_recovery` channels.

Statements in which table names are qualified with database names when using this option do not work with table-level replication filtering options such as `--replicate-do-table`. Suppose we have a database named `a` on the master, one named `b` on the slave, each containing a table `t`, and have started the master with `--replicate-rewrite-db='a->b'`. At a later point in time, we execute `DELETE FROM a.t`. In this case, no relevant filtering rule works, for the reasons shown here:

1. `--replicate-do-table=a.t` does not work because the slave has table `t` in database `b`.
2. `--replicate-do-table=b.t` does not match the original statement and so is ignored.
3. `--replicate-do-table=*.t` is handled identically to `--replicate-do-table=a.t`, and thus does not work, either.

Similarly, the `--replication-rewrite-db` option does not work with cross-database updates.

- `--replicate-same-server-id`

Property	Value
Command-Line Format	`--replicate-same-server-id[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

This option is for use on replication slaves. The default is 0 (`FALSE`). With this option set to 1 (`TRUE`), the slave does not skip events that have its own server ID. This setting is normally useful only in rare configurations.

When binary logging is enabled on a replication slave, the combination of the `--replicate-same-server-id` and `--log-slave-updates` options on the slave can cause infinite loops in replication if the server is part of a circular replication topology. (In MySQL 8.0, binary logging is enabled by default, and slave update logging is the default when binary logging is enabled.) However, the use of global transaction identifiers (GTIDs) prevents this situation by skipping the execution of transactions that have already been applied. If `gtid_mode=ON` is set on the slave, you can start the server with this combination of options, but you cannot change to any other GTID mode while the server is running. If any other GTID mode is set, the server does not start with this combination of options.

By default, the slave I/O thread does not write binary log events to the relay log if they have the slave's server ID (this optimization helps save disk usage). If you want to use `--replicate-same-server-id`, be sure to start the slave with this option before you make the slave read its own events that you want the slave SQL thread to execute.

- `--replicate-wild-do-table=`*`db_name.tbl_name`*

Property	Value
Command-Line Format	`--replicate-wild-do-table=name`
Type	String

Creates a replication filter by telling the slave thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns. Patterns can contain the `%` and `_` wildcard characters, which have the same meaning as for the `LIKE` pattern-matching operator.

To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates. See Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”. You can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE` statement.

This option supports channel specific replication filters, enabling multi-source replication slaves to use specific filters for different sources. To configure a channel specific replication filter on a channel named *`channel_1`* use `--replicate-wild-do-table:`*`channel_1:db_name.tbl_name`*. In this case, the first colon is interpreted as a separator and subsequent colons are literal colons. See Section 17.2.5.4, “Replication Channel Based Filters” for more information.

Note

Global replication filters cannot be used on a MySQL server instance that is configured for Group Replication, because filtering transactions on some servers would make the group unable to reach agreement on a consistent state. Channel specific replication filters can be used on replication channels that are not directly involved with Group Replication, such as where a group member also acts as a replication slave to a master that is outside the group. They cannot be used on the `group_replication_applier` or `group_replication_recovery` channels.

This option applies to tables, views, and triggers. It does not apply to stored procedures and functions, or events. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

As an example, `--replicate-wild-do-table=foo%.bar%` replicates only updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

If the table name pattern is `%`, it matches any table name and the option also applies to database-level statements (`CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`). For example, if you use `--replicate-wild-do-table=foo%.%`, database-level statements are replicated if the database name matches the pattern `foo%`.

To include literal wildcard characters in the database or table name patterns, escape them with a backslash. For example, to replicate all tables of a database that is named `my_own%db`, but not replicate tables from the `my1ownAABCdb` database, you should escape the `_` and `%` characters like this: `--replicate-wild-do-table=my\_own\%db`. If you use the option on the command line, you might need to double the backslashes or quote the option value, depending on your command interpreter. For example, with the `bash` shell, you would need to type `--replicate-wild-do-table=my\\_own\\%db`.

- `--replicate-wild-ignore-table=`*`db_name.tbl_name`*

Property	Value
Command-Line Format	`--replicate-wild-ignore-table=name`
Type	String

Creates a replication filter which keeps the slave thread from replicating a statement in which any table matches the given wildcard pattern. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates. See Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”. You can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_WILD_IGNORE_TABLE` statement.

This option supports channel specific replication filters, enabling multi-source replication slaves to use specific filters for different sources. To configure a channel specific replication filter on a channel named *`channel_1`* use `--replicate-wild-ignore:`*`channel_1`*`:`*`db_name.tbl_name`*. In this case, the first colon is interpreted as a separator and subsequent colons are literal colons. See Section 17.2.5.4, "Replication Channel Based Filters" for more information.

Note

Global replication filters cannot be used on a MySQL server instance that is configured for Group Replication, because filtering transactions on some servers would make the group unable to reach agreement on a consistent state. Channel specific replication filters can be used on replication channels that are not directly involved with Group Replication, such as where a group member also acts as a replication slave to a master that is outside the group. They cannot be used on the `group_replication_applier` or `group_replication_recovery` channels.

As an example, `--replicate-wild-ignore-table=foo%.bar%` does not replicate updates that use a table where the database name starts with `foo` and the table name starts with `bar`. For information about how matching works, see the description of the `--replicate-wild-do-table` option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` as well.

- `--slave-checkpoint-group=#`

Property	Value
Command-Line Format	`--slave-checkpoint-group=#`
Type	Integer
Default Value	`512`
Minimum Value	`32`
Maximum Value	`524280`
Block Size	`8`

Sets the maximum number of transactions that can be processed by a multithreaded slave before a checkpoint operation is called to update its status as shown by `SHOW SLAVE STATUS`. Setting this option has no effect on slaves for which multithreading is not enabled.

Note

Multithreaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this option. See Section 22.6.3, "Known Issues in NDB Cluster Replication", for more information.

This option works in combination with the `--slave-checkpoint-period` option in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this option is 32, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 1. The effective value is always a multiple of 8; you can set it to a value that is not such a multiple, but the server rounds it down to the next lower multiple of 8 before storing the value. (*Exception*: No such rounding is performed by the debug server.) Regardless of how the server was built, the default value is 512, and the maximum allowed value is 524280.

- `--slave-checkpoint-period=#`

Property	Value
Command-Line Format	`--slave-checkpoint-period=#`
Type	Integer
Default Value	`300`
Minimum Value	`1`
Maximum Value	`4G`

Sets the maximum time (in milliseconds) that is allowed to pass before a checkpoint operation is called to update the status of a multithreaded slave as shown by `SHOW SLAVE STATUS`. Setting this option has no effect on slaves for which multithreading is not enabled.

Note

Multithreaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this option. See Section 22.6.3, "Known Issues in NDB Cluster Replication", for more information.

This option works in combination with the `--slave-checkpoint-group` option in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this option is 1, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 0. Regardless of how the server was built, the default value is 300, and the maximum possible value is 4294967296 (4GB).

- `--slave-parallel-workers`

Property	Value
Command-Line Format	`--slave-parallel-workers=#`
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1024`

Enables multithreading on the slave and sets the number of slave applier threads for executing replication transactions in parallel. When the value is a number greater than 0, the slave is a multithreaded slave with the specified number of applier threads, plus a coordinator thread to manage them. If you are using multiple replication channels, each channel has this number of threads.

Retrying of transactions is supported when multithreading is enabled on a slave. When `slave_preserve_commit_order=1`, transactions on a slave are externalized on the slave in the same order as they appear in the slave's relay log. The way in which transactions are distributed among applier threads is configured by `--slave-parallel-type`.

To disable parallel execution, set this option to 0, which gives the slave a single applier thread and no coordinator thread. With this setting, the `--slave-parallel-type` and `slave_preserve_commit_order` options have no effect and are ignored.

Note

Multithreaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this option. See Section 22.6.3, “Known Issues in NDB Cluster Replication”, for more information.

- `--slave-pending-jobs-size-max=#`

Property	Value
Command-Line Format	`--slave-pending-jobs-size-max=#`
Type	Integer
Default Value (>= 8.0.12)	`128M`
Default Value (<= 8.0.11)	`16M`
Minimum Value	`1024`
Maximum Value	`16EiB`
Block Size	`1024`

For multithreaded slaves, this option sets the maximum amount of memory (in bytes) available to slave worker queues holding events not yet applied. Setting this option has no effect on slaves for which multithreading is not enabled.

The minimum possible value for this option is 1024 bytes; the default is 128MB. The maximum possible value is 18446744073709551615 (16 exbibytes). Values that are not exact multiples of 1024 bytes are rounded down to the next lower multiple of 1024 bytes prior to being stored.

The value of this variable is a soft limit and can be set to match the normal workload. If an unusually large event exceeds this size, the transaction is held until all the slave workers have empty queues, and then processed. All subsequent transactions are held until the large transaction has been completed.

- `--skip-slave-start`

Property	Value
Command-Line Format	`--skip-slave-start[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Tells the slave server not to start the slave threads when the server starts. To start the threads later, use a `START SLAVE` statement.

- `--slave-load-tmpdir=`*`dir_name`*

Property	Value
Command-Line Format	`--slave-load-tmpdir=dir_name`
System Variable	`slave_load_tmpdir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

Property	Value
Default Value	`Value of --tmpdir`

The name of the directory where the slave creates temporary files. This option is by default equal to the value of the `tmpdir` system variable, or the default that applies when that system variable is not specified. When the slave SQL thread replicates a `LOAD DATA` statement, it extracts the file to be loaded from the relay log into temporary files, and then loads these into the table. If the file loaded on the master is huge, the temporary files on the slave are huge, too. Therefore, it might be advisable to use this option to tell the slave to put temporary files in a directory located in some file system that has a lot of available space. In that case, the relay logs are huge as well, so you might also want to use the `relay_log` system variable to place the relay logs in that file system.

The directory specified by this option should be located in a disk-based file system (not a memory-based file system) so that the temporary files used to replicate `LOAD DATA` can survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process. However, replication can now continue after a restart if the temporary files have been removed.

- `slave-max-allowed-packet=`*`bytes`*

Property	Value
Command-Line Format	`--slave-max-allowed-packet=#`
Type	Integer
Default Value	`1073741824`
Minimum Value	`1024`
Maximum Value	`1073741824`

This option sets the maximum packet size in bytes that the slave SQL and I/O threads can handle. It is possible for a replication master to write binary log events longer than its `max_allowed_packet` setting once the event header is added. The setting for `slave_max_allowed_packet` must be larger than the `max_allowed_packet` setting on the master, so that large updates using row-based replication do not cause replication to fail.

The corresponding server variable `slave_max_allowed_packet` always has a value that is a positive integer multiple of 1024; if you set it to some value that is not such a multiple, the value is automatically rounded down to the next highest multiple of 1024. (For example, if you start the server with `--slave-max-allowed-packet=10000`, the value used is 9216; setting 0 as the value causes 1024 to be used.) A truncation warning is issued in such cases.

The maximum (and default) value is 1073741824 (1 GB); the minimum is 1024.

- `--slave-net-timeout=`*`seconds`*

Property	Value
Command-Line Format	`--slave-net-timeout=#`
System Variable	`slave_net_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value	`60`
Minimum Value	`1`

The number of seconds to wait for more data or a heartbeat signal from the master before the slave considers the connection broken, aborts the read, and tries to reconnect. The default value is 60 seconds (one minute). The first retry occurs immediately after the timeout. The interval between retries is controlled by the `MASTER_CONNECT_RETRY` option for the `CHANGE MASTER TO` statement, and the number of reconnection attempts is limited by the `MASTER_RETRY_COUNT` option for the `CHANGE MASTER TO` statement.

The heartbeat interval, which stops the connection timeout occurring in the absence of data if the connection is still good, is controlled by the `MASTER_HEARTBEAT_PERIOD` option for the `CHANGE MASTER TO` statement. The heartbeat interval defaults to half the value of `--slave-net-timeout`, and it is recorded in the master info log and shown in the `replication_connection_configuration` Performance Schema table. Note that a change to the value or default setting of `--slave-net-timeout` does not automatically change the heartbeat interval, whether that has been set explicitly or is using a previously calculated default. If the connection timeout is changed, you must also issue `CHANGE MASTER TO` to adjust the heartbeat interval to an appropriate value so that it occurs before the connection timeout.

- `--slave-parallel-type=`*`type`*

Property	Value
Command-Line Format	`--slave-parallel-type=type`
Type	Enumeration
Default Value	`DATABASE`

Valid Values	`DATABASE`
	
	`LOGICAL_CLOCK`

When using a multithreaded slave (`slave_parallel_workers` is greater than 0), this option specifies the policy used to decide which transactions are allowed to execute in parallel on the slave. The option has no effect on slaves for which multithreading is not enabled. The possible values are:

- `LOGICAL_CLOCK`: Transactions that are part of the same binary log group commit on a master are applied in parallel on a slave. The dependencies between transactions are tracked based on their timestamps to provide additional parallelization where possible. When this value is set, the `binlog_transaction_dependency_tracking` system variable can be used on the master to specify that write sets are used for parallelization in place of timestamps, if a write set is available for the transaction and gives improved results compared to timestamps.

- `DATABASE`: Transactions that update different databases are applied in parallel. This value is only appropriate if data is partitioned into multiple databases which are being updated independently and concurrently on the master. There must be no cross-database constraints, as such constraints may be violated on the slave.

When `slave_preserve_commit_order=1` is set, you can only use `LOGICAL_CLOCK`.

If your replication topology uses multiple levels of slaves, `LOGICAL_CLOCK` may achieve less parallelization for each level the slave is away from the master. You can reduce this effect by using

`binlog_transaction_dependency_tracking` on the master to specify that write sets are used instead of timestamps for parallelization where possible.

- `slave-rows-search-algorithms=list`

Property	Value
Command-Line Format	`--slave-rows-search-algorithms=list`
Deprecated	8.0.18
Type	Set
Default Value (>= 8.0.2)	`INDEX_SCAN,HASH_SCAN`
Default Value (<= 8.0.1)	`TABLE_SCAN,INDEX_SCAN`

Valid Values	`TABLE_SCAN,INDEX_SCAN`
	
	`INDEX_SCAN,HASH_SCAN`
	
	`TABLE_SCAN,HASH_SCAN`
	
	`TABLE_SCAN,INDEX_SCAN,HASH_SCAN` (equivalent to INDEX_SCAN,HASH_SCAN)

When preparing batches of rows for row-based logging and replication, this option controls how the rows are searched for matches, in particular whether hash scans are used. The use of this option is now deprecated. The default setting `INDEX_SCAN,HASH_SCAN` is optimal for performance and works correctly in all scenarios.

- `--slave-skip-errors=[err_code1,err_code2,...|all|ddl_exist_errors]`

Property	Value
Command-Line Format	`--slave-skip-errors=name`
System Variable	`slave_skip_errors`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`OFF`

Valid Values	`OFF`
	
	`[list of error codes]`
	
	`all`
	
	`ddl_exist_errors`

Normally, replication stops when an error occurs on the slave, which gives you the opportunity to resolve the inconsistency in the data manually. This option causes the slave SQL thread to continue replication when a statement returns any of the errors listed in the option value.

Do not use this option unless you fully understand why you are getting errors. If there are no bugs in your replication setup and client programs, and no bugs in MySQL itself, an error that stops replication

should never occur. Indiscriminate use of this option results in slaves becoming hopelessly out of synchrony with the master, with you having no idea why this has occurred.

For error codes, you should use the numbers provided by the error message in your slave error log and in the output of `SHOW SLAVE STATUS`. Appendix B, *Errors, Error Codes, and Common Problems*, lists server error codes.

The shorthand value `ddl_exist_errors` is equivalent to the error code list `1007,1008,1050,1051,1054,1060,1061,1068,1094,1146`.

You can also (but should not) use the very nonrecommended value of `all` to cause the slave to ignore all error messages and keeps going regardless of what happens. Needless to say, if you use `all`, there are no guarantees regarding the integrity of your data. Please do not complain (or file bug reports) in this case if the slave's data is not anywhere close to what it is on the master. *You have been warned*.

Examples:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
--slave-skip-errors=ddl_exist_errors
```

- `--slave-sql-verify-checksum={0|1}`

Property	Value
Command-Line Format	`--slave-sql-verify-checksum[={OFF|ON}]`
Type	Boolean
Default Value	`ON`

When this option is enabled, the slave examines checksums read from the relay log, in the event of a mismatch, the slave stops with an error.

The following options are used internally by the MySQL test suite for replication testing and debugging. They are not intended for use in a production setting.

- `--abort-slave-event-count`

Property	Value
Command-Line Format	`--abort-slave-event-count=#`
Type	Integer
Default Value	`0`
Minimum Value	`0`

When this option is set to some positive integer *`value`* other than 0 (the default) it affects replication behavior as follows: After the slave SQL thread has started, *`value`* log events are permitted to be executed; after that, the slave SQL thread does not receive any more events, just as if the network connection from the master were cut. The slave thread continues to run, and the output from `SHOW SLAVE STATUS` displays `Yes` in both the `Slave_IO_Running` and the `Slave_SQL_Running` columns, but no further events are read from the relay log.

- `--disconnect-slave-event-count`

Property	Value
Command-Line Format	`--disconnect-slave-event-count=#`
Type	Integer
Default Value	`0`

Options for Logging Slave Status to Tables

Replication slave status information is logged to an InnoDB table in the `mysql` database. Before MySQL 8.0, this information could alternatively be logged to a file in the data directory, but the use of that format is now deprecated. Writing of the master info log and the relay log info log can be configured separately using the two server options listed here:

- `--master-info-repository={TABLE|FILE}`

Property	Value
Command-Line Format	`--master-info-repository={FILE|TABLE}`
Type	String
Default Value (>= 8.0.2)	`TABLE`
Default Value (<= 8.0.1)	`FILE`

Valid Values	`FILE`
	
	`TABLE`

This option determines whether the slave server logs master status and connection information to an InnoDB table in the `mysql` database, or to a file in the data directory.

The default setting is `TABLE`. As an InnoDB table, the master info log is named `mysql.slave_master_info`. The `TABLE` setting is required when multiple replication channels are configured.

The `FILE` setting is deprecated, and will be removed in a future release. As a file, the master info log is named `master.info` by default, and you can change this name using the `--master-info-file` option.

The setting for the location of this slave status log has a direct influence on the effect had by the setting of the `sync_master_info` system variable. You can only change the setting when no replication threads are executing.

- `--relay-log-info-repository={TABLE|FILE}`

Property	Value
Command-Line Format	`--relay-log-info-repository={FILE|TABLE}`
Type	String
Default Value (>= 8.0.2)	`TABLE`
Default Value (<= 8.0.1)	`FILE`

Valid Values	`FILE`
	
	`TABLE`

This option determines whether the slave server logs its position in the relay logs to an InnoDB table in the `mysql` database, or to a file in the data directory.

The default setting is `TABLE`. As an InnoDB table, the relay log info log is named `mysql.slave_relay_log_info`. The `TABLE` setting is required when multiple replication channels are configured. The `TABLE` setting for the relay log info log is also required to make replication resilient to unexpected halts, for which the `--relay-log-recovery` option must also be enabled. See Making replication resilient to unexpected halts for more information.

The `FILE` setting is deprecated, and will be removed in a future release. As a file, the relay log info log is named `relay-log.info` by default, and you can change this name using the `relay_log_info_file` system variable.

The setting for the location of this slave status log has a direct influence on the effect had by the setting of the `sync_relay_log_info` system variable. You can only change the setting when no replication threads are executing.

The slave status log tables and their contents are considered local to a given MySQL Server. They are not replicated, and changes to them are not written to the binary log.

For more information, see Section 17.2.4, “Replication Relay and Status Logs”.

System Variables Used on Replication Slaves

The following list describes system variables for controlling replication slave servers. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used with replication slaves are listed earlier in this section.

- `init_slave`

Property	Value
Command-Line Format	`--init-slave=name`
System Variable	`init_slave`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

This variable is similar to `init_connect`, but is a string to be executed by a slave server each time the SQL thread starts. The format of the string is the same as for the `init_connect` variable. The setting of this variable takes effect for subsequent `START SLAVE` statements.

Note

The SQL thread sends an acknowledgment to the client before it executes `init_slave`. Therefore, it is not guaranteed that `init_slave` has been executed when `START SLAVE` returns. See Section 13.4.2.6, “START SLAVE Syntax”, for more information.

- `log_slow_slave_statements`

Property	Value
Command-Line Format	`--log-slow-slave-statements[={OFF|ON}]`

Property	Value
System Variable	`log_slow_slave_statements`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

When the slow query log is enabled, this variable enables logging for queries that have taken more than `long_query_time` seconds to execute on the slave. Note that if row-based replication is in use (`binlog_format=ROW`), `log_slow_slave_statements` has no effect. Queries are only added to the slave's slow query log when they are logged in statement format in the binary log, that is, when `binlog_format=STATEMENT` is set, or when `binlog_format=MIXED` is set and the statement is logged in statement format. Slow queries that are logged in row format when `binlog_format=MIXED` is set, or that are logged when `binlog_format=ROW` is set, are not added to the slave's slow query log, even if `log_slow_slave_statements` is enabled.

Setting `log_slow_slave_statements` has no immediate effect. The state of the variable applies on all subsequent `START SLAVE` statements. Also note that the global setting for `long_query_time` applies for the lifetime of the SQL thread. If you change that setting, you must stop and restart the slave's SQL thread to implement the change there (for example, by issuing `STOP SLAVE` and `START SLAVE` statements with the `SQL_THREAD` option).

- `master_info_repository`

Property	Value
Command-Line Format	`--master-info-repository={FILE|TABLE}`
System Variable	`master_info_repository`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (>= 8.0.2)	`TABLE`
Default Value (<= 8.0.1)	`FILE`

Valid Values	`FILE`
	
	`TABLE`

The setting of this variable determines whether the slave server logs master status and connection information to an InnoDB table in the `mysql` database, or to a file in the data directory.

The default setting is `TABLE`. As an InnoDB table, the master info log is named `mysql.slave_master_info`. The `TABLE` setting is required when multiple replication channels are configured.

The `FILE` setting is deprecated, and will be removed in a future release. As a file, the master info log is named `master.info` by default, and you can change this name using the `--master-info-file` option.

The setting for the location of this slave status log has a direct influence on the effect had by the setting of the `sync_master_info` system variable. You can only change the setting when no replication threads are executing.

- `max_relay_log_size`

Property	Value
Command-Line Format	`--max-relay-log-size=#`
System Variable	`max_relay_log_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1073741824`

If a write by a replication slave to its relay log causes the current log file size to exceed the value of this variable, the slave rotates the relay logs (closes the current file and opens the next one). If `max_relay_log_size` is 0, the server uses `max_binlog_size` for both the binary log and the relay log. If `max_relay_log_size` is greater than 0, it constrains the size of the relay log, which enables you to have different sizes for the two logs. You must set `max_relay_log_size` to between 4096 bytes and 1GB (inclusive), or to 0. The default value is 0. See Section 17.2.2, “Replication Implementation Details”.

- `relay_log`

Property	Value
Command-Line Format	`--relay-log=file_name`
System Variable	`relay_log`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

The base name for relay log files. For the default replication channel, the default base name for relay logs is *`host_name`*`-relay-bin`. For non-default replication channels, the default base name for relay logs is *`host_name`*`-relay-bin-`*`channel`*, where *`channel`* is the name of the replication channel recorded in this relay log.

The server writes the file in the data directory unless the base name is given with a leading absolute path name to specify a different directory. The server creates relay log files in sequence by adding a numeric suffix to the base name.

The relay log and relay log index on a replication server cannot be given the same names as the binary log and binary log index, whose names are specified by the `--log-bin` and `--log-bin-index` options. The server issues an error message and does not start if the binary log and relay log file base names would be the same.

Due to the manner in which MySQL parses server options, if you specify this variable at server startup, you must supply a value; *the default base name is used only if the option is not actually specified*. If you specify the `relay_log` system variable at server startup without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see Section 4.2.2, "Specifying Program Options".

If you specify this variable, the value specified is also used as the base name for the relay log index file. You can override this behavior by specifying a different relay log index file base name using the `relay_log_index` system variable.

When the server reads an entry from the index file, it checks whether the entry contains a relative path. If it does, the relative part of the path is replaced with the absolute path set using the `relay_log` system variable. An absolute path remains unchanged; in such a case, the index must be edited manually to enable the new path or paths to be used.

You may find the `relay_log` system variable useful in performing the following tasks:

- Creating relay logs whose names are independent of host names.
- If you need to put the relay logs in some area other than the data directory because your relay logs tend to be very large and you do not want to decrease `max_relay_log_size`.
- To increase speed by using load-balancing between disks.

You can obtain the relay log file name (and path) from the `relay_log_basename` system variable.

- `relay_log_basename`

Property	Value
System Variable	`relay_log_basename`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	`datadir + '/' + hostname + '-relay-bin'`

Holds the name and complete path to the relay log file. This variable is set by the server and is read only.

- `relay_log_index`

Property	Value
Command-Line Format	`--relay-log-index=file_name`
System Variable	`relay_log_index`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

Property	Value
Default Value	*host_name*-relay-bin.index

The name for the relay log index file. If you do not specify this variable, but the relay_log system variable is specified, its value is used as the default base name for the relay log index file. If relay_log is also not specified, then for the default replication channel, the default name is *host_name*-relay-bin.index, using the name of the host machine. For non-default replication channels, the default name is *host_name*-relay-bin-*channel*.index, where *channel* is the name of the replication channel recorded in this relay log index.

The default location for relay log files is the data directory, or any other location that was specified using the relay_log system variable. You can use the relay_log_index system variable to specify an alternative location, by adding a leading absolute path name to the base name to specify a different directory.

The relay log and relay log index on a replication server cannot be given the same names as the binary log and binary log index, whose names are specified by the --log-bin and --log-bin-index options. The server issues an error message and does not start if the binary log and relay log file base names would be the same.

Due to the manner in which MySQL parses server options, if you specify this variable at server startup, you must supply a value; *the default base name is used only if the option is not actually specified*. If you specify the relay_log_index system variable at server startup without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see Section 4.2.2, “Specifying Program Options”.

- relay_log_info_file

Property	Value
Command-Line Format	--relay-log-info-file=file_name
Deprecated	8.0.18
System Variable	relay_log_info_file
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	relay-log.info

The name of the file in which the slave records information about the relay logs, when relay_log_info_repository=FILE. If relay_log_info_repository=TABLE, it is the file name that would be used in case the repository was changed to FILE). The default name is relay-log.info in the data directory. relay_log_info_file and the setting relay_log_info_repository=FILE are deprecated, as the use of a file for the relay log info log has been superseded by crash-safe slave tables. For information about the relay log info log, see Section 17.2.4.2, “Slave Status Logs”.

- relay_log_info_repository

Property	Value
Command-Line Format	`--relay-log-info-repository=value`
System Variable	`relay_log_info_repository`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value (>= 8.0.2)	`TABLE`
Default Value (<= 8.0.1)	`FILE`

Valid Values	`FILE`
	
	`TABLE`

The setting of this variable determines whether the slave server logs its position in the relay logs to an InnoDB table in the `mysql` database, or to a file in the data directory.

The default setting is `TABLE`. As an InnoDB table, the relay log info log is named `mysql.slave_relay_log_info`. The `TABLE` setting is required when multiple replication channels are configured. The `TABLE` setting for the relay log info log is also required to make replication resilient to unexpected halts, for which the `--relay-log-recovery` option must also be enabled. See Making replication resilient to unexpected halts for more information.

The `FILE` setting is deprecated, and will be removed in a future release. As a file, the relay log info log is named `relay-log.info` by default, and you can change this name using the `relay_log_info_file` system variable.

The setting for the location of this slave status log has a direct influence on the effect had by the setting of the `sync_relay_log_info` system variable. You can only change the setting when no replication threads are executing.

- `relay_log_purge`

Property	Value
Command-Line Format	`--relay-log-purge[={OFF|ON}]`
System Variable	`relay_log_purge`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Disables or enables automatic purging of relay log files as soon as they are not needed any more. The default value is 1 (`ON`).

- `relay_log_recovery`

Property	Value
Command-Line Format	`--relay-log-recovery[={OFF|ON}]`
System Variable	`relay_log_recovery`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enables automatic relay log recovery immediately following server startup. The recovery process creates a new relay log file, initializes the SQL thread position to this new relay log, and initializes the I/O thread to the SQL thread position. Reading of the relay log from the master then continues. This global variable is read-only; its value can be changed by starting the slave with the `--relay-log-recovery` option, which should be used following a crash on the replication slave to ensure that no possibly corrupted relay logs are processed, and must be used in order to guarantee a crash-safe slave. The default value is 0 (disabled).

This variable also interacts with `relay-log-purge`, which controls purging of logs when they are no longer needed. Enabling the `--relay-log-recovery` option when `relay-log-purge` is disabled risks reading the relay log from files that were not purged, leading to data inconsistency, and is therefore not crash-safe.

When `relay_log_recovery` is enabled and the slave has stopped due to errors encountered while running in multithreaded mode, you can use `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` to ensure that all gaps are processed before switching back to single-threaded mode or executing a `CHANGE MASTER TO` statement.

- `relay_log_space_limit`

Property	Value
Command-Line Format	`--relay-log-space-limit=#`
System Variable	`relay_log_space_limit`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The maximum amount of space to use for all relay logs.

- `report_host`

Property	Value
Command-Line Format	`--report-host=host_name`
System Variable	`report_host`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The host name or IP address of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server. Leave the value unset if you do not want the slave to register itself with the master.

Note

It is not sufficient for the master to simply read the IP address of the slave from the TCP/IP socket after the slave connects. Due to NAT and other routing issues, that IP may not be valid for connecting to the slave from the master or other hosts.

- `report_password`

Property	Value
Command-Line Format	`--report-password=name`
System Variable	`report_password`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The account password of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the master was started with `--show-slave-auth-info`.

Although the name of this variable might imply otherwise, `report_password` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the password for the MySQL replication user account.

- `report_port`

Property	Value
Command-Line Format	`--report-port=port_num`
System Variable	`report_port`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value	`[slave_port]`
Minimum Value	`0`
Maximum Value	`65535`

The TCP/IP port number for connecting to the slave, to be reported to the master during slave registration. Set this only if the slave is listening on a nondefault port or if you have a special tunnel from the master or other clients to the slave. If you are not sure, do not use this option.

The default value for this option is the port number actually used by the slave. This is also the default value displayed by `SHOW SLAVE HOSTS`.

- `report_user`

Property	Value
Command-Line Format	`--report-user=name`
System Variable	`report_user`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	String

The account user name of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server if the master was started with `--show-slave-auth-info`.

Although the name of this variable might imply otherwise, `report_user` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the name of the MySQL replication user account.

- `rpl_read_size`

Property	Value
Command-Line Format	`--rpl-read-size=#`
Introduced	8.0.11
System Variable	`rpl_read_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`8192`
Minimum Value	`8192`
Maximum Value	`4294967295`

The `rpl_read_size` system variable controls the minimum amount of data in bytes that is read from the binary log files and relay log files. If heavy disk I/O activity for these files is impeding performance

for the database, increasing the read size might reduce file reads and I/O stalls when the file data is not currently cached by the operating system.

The minimum and default value for `rpl_read_size` is 8192 bytes. The value must be a multiple of 4KB. Note that a buffer the size of this value is allocated for each thread that reads from the binary log and relay log files, including dump threads on masters and coordinator threads on slaves. Setting a large value might therefore have an impact on memory consumption for servers.

- `rpl_semi_sync_slave_enabled`

Property	Value
Command-Line Format	`--rpl-semi-sync-slave-enabled[={OFF|ON}]`
System Variable	`rpl_semi_sync_slave_enabled`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Controls whether semisynchronous replication is enabled on the slave. To enable or disable the plugin, set this variable to `ON` or `OFF` (or 1 or 0), respectively. The default is `OFF`.

This variable is available only if the slave-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_slave_trace_level`

Property	Value
Command-Line Format	`--rpl-semi-sync-slave-trace-level=#`
System Variable	`rpl_semi_sync_slave_trace_level`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`32`

The semisynchronous replication debug trace level on the slave. See `rpl_semi_sync_master_trace_level` for the permissible values.

This variable is available only if the slave-side semisynchronous replication plugin is installed.

- `rpl_stop_slave_timeout`

Property	Value
Command-Line Format	`--rpl-stop-slave-timeout=seconds`
System Variable	`rpl_stop_slave_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	`31536000`
Minimum Value	`2`
Maximum Value	`31536000`

You can control the length of time (in seconds) that `STOP SLAVE` waits before timing out by setting this variable. This can be used to avoid deadlocks between `STOP SLAVE` and other slave SQL statements using different client connections to the slave.

The maximum and default value of `rpl_stop_slave_timeout` is 31536000 seconds (1 year). The minimum is 2 seconds. Changes to this variable take effect for subsequent `STOP SLAVE` statements.

This variable affects only the client that issues a `STOP SLAVE` statement. When the timeout is reached, the issuing client returns an error message stating that the command execution is incomplete. The client then stops waiting for the slave threads to stop, but the slave threads continue to try to stop, and the `STOP SLAVE` instruction remains in effect. Once the slave threads are no longer busy, the `STOP SLAVE` statement is executed and the slave stops.

- `slave_checkpoint_group`

Property	Value
Command-Line Format	`--slave-checkpoint-group=#`
System Variable	`slave_checkpoint_group`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`512`
Minimum Value	`32`
Maximum Value	`524280`
Block Size	`8`

Sets the maximum number of transactions that can be processed by a multithreaded slave before a checkpoint operation is called to update its status as shown by `SHOW SLAVE STATUS`. Setting this variable has no effect on slaves for which multithreading is not enabled. Setting this variable has no immediate effect. The state of the variable applies on all subsequent `START SLAVE` commands.

Note

Multithreaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this variable. See Section 22.6.3, “Known Issues in NDB Cluster Replication”, for more information.

This variable works in combination with the `slave_checkpoint_period` system variable in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this variable is 32, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 1. The effective value is always a multiple of 8; you can set it to a value that is not such a multiple, but the server rounds it down to the next lower multiple of 8 before storing the value. (*Exception*: No such rounding is performed by the debug server.) Regardless of how the server was built, the default value is 512, and the maximum allowed value is 524280.

- `slave_checkpoint_period`

Property	Value
Command-Line Format	`--slave-checkpoint-period=#`
System Variable	`slave_checkpoint_period`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`300`
Minimum Value	`1`
Maximum Value	`4G`

Sets the maximum time (in milliseconds) that is allowed to pass before a checkpoint operation is called to update the status of a multithreaded slave as shown by `SHOW SLAVE STATUS`. Setting this variable has no effect on slaves for which multithreading is not enabled. Setting this variable takes effect for all replication channels immediately, including running channels.

Note

Multithreaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this variable. See Section 22.6.3, “Known Issues in NDB Cluster Replication”, for more information.

This variable works in combination with the `slave_checkpoint_group` system variable in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this variable is 1, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 0. Regardless of how the server was built, the default value is 300, and the maximum possible value is 4294967296 (4GB).

- `slave_compressed_protocol`

Property	Value
Command-Line Format	`--slave-compressed-protocol[={OFF|ON}]`
Deprecated	8.0.18
System Variable	`slave_compressed_protocol`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean

Property	Value
Default Value	`OFF`

Whether to use compression of the master/slave protocol if both master and slave support it. If this variable is disabled (the default), connections are uncompressed. Changes to this variable take effect on subsequent connection attempts; this includes after issuing a `START SLAVE` statement, as well as reconnections made by a running I/O thread (for example, after setting the `MASTER_RETRY_COUNT` option for the `CHANGE MASTER TO` statement).

As of MySQL 8.0.18, if `slave_compressed_protocol` is enabled, it takes precedence over any `MASTER_COMPRESSION_ALGORITHMS` option specified for the `CHANGE MASTER TO` statement. In this case, connections to the master use `zlib` compression if both the master and slave support that algorithm. If `slave_compressed_protocol` is disabled, the value of `MASTER_COMPRESSION_ALGORITHMS` applies. For more information, see Section 4.2.6, “Connection Compression Control”.

As of MySQL 8.0.18, this system variable is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `slave_exec_mode`

Property	Value
Command-Line Format	`--slave-exec-mode=mode`
System Variable	`slave_exec_mode`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`IDEMPOTENT` (NDB) `STRICT` (Other)

Valid Values	`IDEMPOTENT`
	
	`STRICT`

Controls how a slave thread resolves conflicts and errors during replication. `IDEMPOTENT` mode causes suppression of duplicate-key and no-key-found errors; `STRICT` means no such suppression takes place.

`IDEMPOTENT` mode is intended for use in multi-master replication, circular replication, and some other special replication scenarios for NDB Cluster Replication. (See Section 22.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”, and Section 22.6.11, “NDB Cluster Replication Conflict Resolution”, for more information.) NDB Cluster ignores any value explicitly set for `slave_exec_mode`, and always treats it as `IDEMPOTENT`.

In MySQL Server 8.0, `STRICT` mode is the default value.

Setting this variable takes immediate effect for all replication channels, including running channels.

For storage engines other than `NDB`, *`IDEMPOTENT` mode should be used only when you are absolutely sure that duplicate-key errors and key-not-found errors can safely be ignored*. It is meant to be used in fail-over scenarios for NDB Cluster where multi-master replication or circular replication is employed, and is not recommended for use in other cases.

- `slave_load_tmpdir`

Property	Value
Command-Line Format	`--slave-load-tmpdir=dir_name`
System Variable	`slave_load_tmpdir`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name
Default Value	`Value of --tmpdir`

The name of the directory where the slave creates temporary files. Setting this variable takes effect for all replication channels immediately, including running channels. This system variable is by default equal to the value of the `tmpdir` system variable, or the default that applies when that system variable is not specified.

When the slave SQL thread replicates a `LOAD DATA` statement, it extracts the file to be loaded from the relay log into temporary files, and then loads these into the table. If the file loaded on the master is huge, the temporary files on the slave are huge, too. Therefore, it might be advisable to use this option to tell the slave to put temporary files in a directory located in some file system that has a lot of available space. In that case, the relay logs are huge as well, so you might also want to use the `relay_log` system variable to place the relay logs in that file system.

The directory specified by this option should be located in a disk-based file system (not a memory-based file system) so that the temporary files used to replicate `LOAD DATA` can survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process. However, replication can now continue after a restart if the temporary files have been removed.

- `slave_max_allowed_packet`

Property	Value
Command-Line Format	`--slave-max-allowed-packet=#`
System Variable	`slave_max_allowed_packet`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1073741824`
Minimum Value	`1024`
Maximum Value	`1073741824`

This option sets the maximum packet size in bytes that the slave SQL and I/O threads can handle. Setting this variable takes effect for all replication channels immediately, including running channels. It is possible for a replication master to write binary log events longer than its `max_allowed_packet` setting once the event header is added. The setting for `slave_max_allowed_packet` must be larger than the `max_allowed_packet` setting on the master, so that large updates using row-based replication do not cause replication to fail.

This global variable always has a value that is a positive integer multiple of 1024; if you set it to some value that is not, the value is rounded down to the next highest multiple of 1024 for it is stored or used; setting `slave_max_allowed_packet` to 0 causes 1024 to be used. (A truncation warning is issued in all such cases.) The default and maximum value is 1073741824 (1 GB); the minimum is 1024.

`slave_max_allowed_packet` can also be set at startup, using the `--slave-max-allowed-packet` option.

- `slave_net_timeout`

Property	Value
Command-Line Format	`--slave-net-timeout=#`
System Variable	`slave_net_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`60`
Minimum Value	`1`

The number of seconds to wait for more data or a heartbeat signal from the master before the slave considers the connection broken, aborts the read, and tries to reconnect. Setting this variable has no immediate effect. The state of the variable applies on all subsequent `START SLAVE` commands.

The default value is 60 seconds (one minute). The first retry occurs immediately after the timeout. The interval between retries is controlled by the `MASTER_CONNECT_RETRY` option for the `CHANGE MASTER TO` statement, and the number of reconnection attempts is limited by the `MASTER_RETRY_COUNT` option for the `CHANGE MASTER TO` statement.

The heartbeat interval, which stops the connection timeout occurring in the absence of data if the connection is still good, is controlled by the `MASTER_HEARTBEAT_PERIOD` option for the `CHANGE MASTER TO` statement. The heartbeat interval defaults to half the value of `slave_net_timeout`, and it is recorded in the master info log and shown in the `replication_connection_configuration` Performance Schema table. Note that a change to the value or default setting of `slave_net_timeout` does not automatically change the heartbeat interval, whether that has been set explicitly or is using a previously calculated default. If the connection timeout is changed, you must also issue `CHANGE MASTER TO` to adjust the heartbeat interval to an appropriate value so that it occurs before the connection timeout.

- `slave_parallel_type=`*`type`*

Property	Value
Command-Line Format	`--slave-parallel-type=type`
Type	Enumeration
Default Value	`DATABASE`

Valid Values	`DATABASE`
	
	`LOGICAL_CLOCK`

When using a multithreaded slave (`slave_parallel_workers` is greater than 0), this variable specifies the policy used to decide which transactions are allowed to execute in parallel on the slave. The variable has no effect on slaves for which multithreading is not enabled. The possible values are:

- `LOGICAL_CLOCK`: Transactions that are part of the same binary log group commit on a master are applied in parallel on a slave. The dependencies between transactions are tracked based on their timestamps to provide additional parallelization where possible. When this value is set, the `binlog_transaction_dependency_tracking` system variable can be used on the master to specify that write sets are used for parallelization in place of timestamps, if a write set is available for the transaction and gives improved results compared to timestamps.
- `DATABASE`: Transactions that update different databases are applied in parallel. This value is only appropriate if data is partitioned into multiple databases which are being updated independently and concurrently on the master. There must be no cross-database constraints, as such constraints may be violated on the slave.

When `slave_preserve_commit_order=1` is set, you can only use `LOGICAL_CLOCK`.

If your replication topology uses multiple levels of slaves, `LOGICAL_CLOCK` may achieve less parallelization for each level the slave is away from the master. You can reduce this effect by using `binlog_transaction_dependency_tracking` on the master to specify that write sets are used instead of timestamps for parallelization where possible.

- `slave_parallel_workers`

Property	Value
Command-Line Format	`--slave-parallel-workers=#`
System Variable	`slave_parallel_workers`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1024`

Enables multithreading on the slave and sets the number of slave applier threads for executing replication transactions in parallel. When the value is a number greater than 0, the slave is a multithreaded slave with the specified number of applier threads, plus a coordinator thread to manage them. If you are using multiple replication channels, each channel has this number of threads.

Note

Multithreaded slaves are not currently supported by NDB Cluster, which silently ignores the setting for this variable. See Section 22.6.3, “Known Issues in NDB Cluster Replication”, for more information.

Retrying of transactions is supported when multithreading is enabled on a slave. When `slave_preserve_commit_order=1`, transactions on a slave are externalized on the slave in the same order as they appear in the slave's relay log. The way in which transactions are distributed among applier threads is configured by `--slave-parallel-type`.

To disable parallel execution, set this option to 0, which gives the slave a single applier thread and no coordinator thread. With this setting, the `--slave-parallel-type` and `slave_preserve_commit_order` options have no effect and are ignored.

Setting `slave_parallel_workers` has no immediate effect. The state of the variable applies on all subsequent `START SLAVE` statements.

- `slave_pending_jobs_size_max`

Property	Value
Command-Line Format	`--slave-pending-jobs-size-max=#`
System Variable	`slave_pending_jobs_size_max`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.12)	`128M`
Default Value (<= 8.0.11)	`16M`
Minimum Value	`1024`
Maximum Value	`16EiB`
Block Size	`1024`

For multithreaded slaves, this variable sets the maximum amount of memory (in bytes) available to slave worker queues holding events not yet applied. Setting this variable has no effect on slaves for which multithreading is not enabled. Setting this variable has no immediate effect. The state of the variable applies on all subsequent `START SLAVE` commands.

The minimum possible value for this variable is 1024 bytes; the default is 128MB. The maximum possible value is 18446744073709551615 (16 exbibytes). Values that are not exact multiples of 1024 bytes are rounded down to the next lower multiple of 1024 bytes prior to being stored.

The value of this variable is a soft limit and can be set to match the normal workload. If an unusually large event exceeds this size, the transaction is held until all the slave workers have empty queues, and then processed. All subsequent transactions are held until the large transaction has been completed.

- `slave_preserve_commit_order`

Property	Value
Command-Line Format	`--slave-preserve-commit-order[={OFF|ON}]`
System Variable	`slave_preserve_commit_order`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

For multithreaded slaves, the setting 1 for this variable ensures that transactions are externalized on the slave in the same order as they appear in the slave's relay log, and prevents gaps in the sequence of transactions that have been executed from the relay log. This variable has no effect on slaves for which multithreading is not enabled. Note that `slave_preserve_commit_order=1` does not preserve the order of non-transactional DML updates, so these might commit before transactions that precede them in the relay log, which might result in gaps.

`slave_preserve_commit_order=1` requires that `--log-bin` and `--log-slave-updates` are enabled on the slave, and `--slave-parallel-type` is set to `LOGICAL_CLOCK`. Before changing this variable, all replication threads (for all replication channels if you are using multiple replication channels) must be stopped.

With `slave_preserve_commit_order` enabled, the executing thread waits until all previous transactions are committed before committing. While the slave thread is waiting for other workers to commit their transactions it reports its status as `Waiting for preceding transaction to commit`. With this mode, a multithreaded slave never enters a state that the master was not in. This supports the use of replication for read scale-out. See Section 17.4.5, “Using Replication for Scale-Out”.

If `slave_preserve_commit_order=0` is set, the transactions that the slave applies in parallel may commit out of order. Therefore, checking for the most recently executed transaction does not guarantee that all previous transactions from the master have been executed on the slave. There is a chance of gaps in the sequence of transactions that have been executed from the slave's relay log. This has implications for logging and recovery when using a multithreaded slave. Note that the setting `slave_preserve_commit_order=1` prevents gaps, but does not prevent master log position lag (where `Exec_master_log_pos` is behind the position up to which transactions have been executed). See Section 17.5.1.33, “Replication and Transaction Inconsistencies” for more information.

- `slave_rows_search_algorithms`

Property	Value
Command-Line Format	`--slave-rows-search-algorithms=value`
Deprecated	8.0.18
System Variable	`slave_rows_search_algorithms`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Set
Default Value (>= 8.0.2)	`INDEX_SCAN,HASH_SCAN`
Default Value (<= 8.0.1)	`TABLE_SCAN,INDEX_SCAN`

Valid Values	`TABLE_SCAN,INDEX_SCAN`
	
	`INDEX_SCAN,HASH_SCAN`
	
	`TABLE_SCAN,HASH_SCAN`
	
	`TABLE_SCAN,INDEX_SCAN,HASH_SCAN` (equivalent to INDEX_SCAN,HASH_SCAN)

When preparing batches of rows for row-based logging and replication, this system variable controls how the rows are searched for matches, in particular whether hash scans are used. The use of this system

variable is now deprecated. The default setting `INDEX_SCAN,HASH_SCAN` is optimal for performance and works correctly in all scenarios.

- `slave_skip_errors`

Property	Value
Command-Line Format	`--slave-skip-errors=name`
System Variable	`slave_skip_errors`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`OFF`

Valid Values	`OFF`
	`[list of error codes]`
	`all`
	`ddl_exist_errors`

Normally, replication stops when an error occurs on the slave, which gives you the opportunity to resolve the inconsistency in the data manually. This variable causes the slave SQL thread to continue replication when a statement returns any of the errors listed in the variable value.

- `slave_sql_verify_checksum`

Property	Value
Command-Line Format	`--slave-sql-verify-checksum[={OFF|ON}]`
System Variable	`slave_sql_verify_checksum`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Cause the slave SQL thread to verify data using the checksums read from the relay log. In the event of a mismatch, the slave stops with an error. Setting this variable takes effect for all replication channels immediately, including running channels.

Note

The slave I/O thread always reads checksums if possible when accepting events from over the network.

- `slave_transaction_retries`

Property	Value
Command-Line Format	`--slave-transaction-retries=#`

Property	Value
System Variable	`slave_transaction_retries`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

Sets the maximum number of times for replication slave SQL threads on a single-threaded or multithreaded slave to automatically retry failed transactions before stopping. Setting this variable takes effect for all replication channels immediately, including running channels. The default value is 10. Setting the variable to 0 disables automatic retrying of transactions.

If a replication slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or because the transaction's execution time exceeded `InnoDB`'s `innodb_lock_wait_timeout` or `NDB`'s `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout`, it automatically retries `slave_transaction_retries` times before stopping with an error. Transactions with a non-temporary error are not retried.

The Performance Schema table `replication_applier_status` shows the number of retries that took place on each replication channel, in the `COUNT_TRANSACTIONS_RETRIES` column. The Performance Schema table `replication_applier_status_by_worker` shows detailed information on transaction retries by individual applier threads on a single-threaded or multithreaded replication slave, and identifies the errors that caused the last transaction and the transaction currently in progress to be reattempted.

- `slave_type_conversions`

Property	Value
Command-Line Format	`--slave-type-conversions=set`
System Variable	`slave_type_conversions`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Set
Default Value	

Valid Values	`ALL_LOSSY`
	`ALL_NON_LOSSY`
	`ALL_SIGNED`
	`ALL_UNSIGNED`

Controls the type conversion mode in effect on the slave when using row-based replication. Its value is a comma-delimited set of zero or more elements from the list: `ALL_LOSSY`, `ALL_NON_LOSSY`, `ALL_SIGNED`, `ALL_UNSIGNED`. Set this variable to an empty string to disallow type conversions between the master and the slave. Setting this variable takes effect for all replication channels immediately, including running channels.

For additional information on type conversion modes applicable to attribute promotion and demotion in row-based replication, see Row-based replication: attribute promotion and demotion.

- `sql_slave_skip_counter`

Property	Value
System Variable	`sql_slave_skip_counter`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer

The number of events from the master that a slave server should skip. Setting the option has no immediate effect. The variable applies to the next `START SLAVE` statement; the next `START SLAVE` statement also changes the value back to 0. When this variable is set to a nonzero value and there are multiple replication channels configured, the `START SLAVE` statement can only be used with the `FOR CHANNEL` *`channel`* clause.

This option is incompatible with GTID-based replication, and must not be set to a nonzero value when `gtid_mode=ON`. If you need to skip transactions when employing GTIDs, use `gtid_executed` from the master instead. See Injecting empty transactions, for information about how to do this.

Important

If skipping the number of events specified by setting this variable would cause the slave to begin in the middle of an event group, the slave continues to skip until it finds the beginning of the next event group and begins from that point. For more information, see Section 13.4.2.5, “SET GLOBAL sql_slave_skip_counter Syntax”.

- `sync_master_info`

Property	Value
Command-Line Format	`--sync-master-info=#`
System Variable	`sync_master_info`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10000`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`

Property	Value
Maximum Value (32-bit platforms)	4294967295

The effects of this variable on a replication slave depend on whether the slave's master_info_repository is set to FILE or TABLE, as explained in the following paragraphs.

master_info_repository = FILE. If the value of sync_master_info is greater than 0, the slave synchronizes its master.info file to disk (using fdatasync()) after every sync_master_info events. If it is 0, the MySQL server performs no synchronization of the master.info file to disk; instead, the server relies on the operating system to flush its contents periodically as with any other file.

master_info_repository = TABLE. If the value of sync_master_info is greater than 0, the slave updates its master info repository table after every sync_master_info events. If it is 0, the table is never updated.

The default value for sync_master_info is 10000. Setting this variable takes effect for all replication channels immediately, including running channels.

- sync_relay_log

Property	Value
Command-Line Format	--sync-relay-log=#
System Variable	sync_relay_log
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	10000
Minimum Value	0
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

If the value of this variable is greater than 0, the MySQL server synchronizes its relay log to disk (using fdatasync()) after every sync_relay_log events are written to the relay log. Setting this variable takes effect for all replication channels immediately, including running channels.

Setting sync_relay_log to 0 causes no synchronization to be done to disk; in this case, the server relies on the operating system to flush the relay log's contents from time to time as for any other file.

A value of 1 is the safest choice because in the event of a crash you lose at most one event from the relay log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

- sync_relay_log_info

Property	Value
Command-Line Format	--sync-relay-log-info=#

Property	Value
System Variable	`sync_relay_log_info`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10000`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The default value for `sync_relay_log_info` is 10000. Setting this variable takes effect for all replication channels immediately, including running channels.

The effects of this variable on the replication slave depend on the server's `relay_log_info_repository` setting (`FILE` or `TABLE`). If the setting is `TABLE`, the effects of the variable also depend on whether the storage engine used by the relay log info table is transactional (such as `InnoDB`) or not transactional (`MyISAM`). The effects of these factors on the behavior of the server for `sync_relay_log_info` values of zero and greater than zero are as follows:

`sync_relay_log_info = 0`

- If `relay_log_info_repository` is set to `FILE`, the MySQL server performs no synchronization of the `relay-log.info` file to disk; instead, the server relies on the operating system to flush its contents periodically as with any other file.
- If `relay_log_info_repository` is set to `TABLE`, and the storage engine for that table is transactional, the table is updated after each transaction. (The `sync_relay_log_info` setting is effectively ignored in this case.)
- If `relay_log_info_repository` is set to `TABLE`, and the storage engine for that table is not transactional, the table is never updated.

`sync_relay_log_info = N > 0`

- If `relay_log_info_repository` is set to `FILE`, the slave synchronizes its `relay-log.info` file to disk (using `fdatasync()`) after every *`N`* transactions.
- If `relay_log_info_repository` is set to `TABLE`, and the storage engine for that table is transactional, the table is updated after each transaction. (The `sync_relay_log_info` setting is effectively ignored in this case.)
- If `relay_log_info_repository` is set to `TABLE`, and the storage engine for that table is not transactional, the table is updated after every *`N`* events.

17.1.6.4 Binary Logging Options and Variables

- Startup Options Used with Binary Logging

- System Variables Used with Binary Logging

You can use the `mysqld` options and system variables that are described in this section to affect the operation of the binary log as well as to control which statements are written to the binary log. For additional information about the binary log, see Section 5.4.4, “The Binary Log”. For additional information about using MySQL server options and system variables, see Section 5.1.7, “Server Command Options”, and Section 5.1.8, “Server System Variables”.

Startup Options Used with Binary Logging

The following list describes startup options for enabling and configuring the binary log. System variables used with binary logging are discussed later in this section.

- `--binlog-row-event-max-size=N`

Property	Value
Command-Line Format	`--binlog-row-event-max-size=#`
System Variable (>= 8.0.14)	`binlog_row_event_max_size`
Scope (>= 8.0.14)	Global
Dynamic (>= 8.0.14)	No
SET_VAR Hint Applies (>= 8.0.14)	No
Type	Integer
Default Value	`8192`
Minimum Value	`256`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

When row-based binary logging is used, this setting is a soft limit on the maximum size of a row-based binary log event, in bytes. Where possible, rows stored in the binary log are grouped into events with a size not exceeding the value of this setting. If an event cannot be split, the maximum size can be exceeded. The value must be (or else gets rounded down to) a multiple of 256. The default is 8192 bytes.

- `--log-bin[=base_name]`

Property	Value
Command-Line Format	`--log-bin=file_name`
Type	File name

Specifies the base name to use for binary log files. With binary logging enabled, the server logs all statements that change data to the binary log, which is used for backup and replication. The binary log is a sequence of files with a base name and numeric extension. The `--log-bin` option value is the base name for the log sequence. The server creates binary log files in sequence by adding a numeric suffix to the base name.

If you do not supply the `--log-bin` option, MySQL uses `binlog` as the default base name for the binary log files. For compatibility with earlier releases, if you supply the `--log-bin` option with no

string or with an empty string, the base name defaults to *`host_name`*`-bin`, using the name of the host machine.

The default location for binary log files is the data directory. You can use the `--log-bin` option to specify an alternative location, by adding a leading absolute path name to the base name to specify a different directory. When the server reads an entry from the binary log index file, which tracks the binary log files that have been used, it checks whether the entry contains a relative path. If it does, the relative part of the path is replaced with the absolute path set using the `--log-bin` option. An absolute path recorded in the binary log index file remains unchanged; in such a case, the index file must be edited manually to enable a new path or paths to be used. The binary log file base name and any specified path are available as the `log_bin_basename` system variable.

In earlier MySQL versions, binary logging was disabled by default, and was enabled if you specified the `--log-bin` option. From MySQL 8.0, binary logging is enabled by default, whether or not you specify the `--log-bin` option. The exception is if you use `mysqld` to initialize the data directory manually by invoking it with the `--initialize` or `--initialize-insecure` option, when binary logging is disabled by default. It is possible to enable binary logging in this case by specifying the `--log-bin` option. When binary logging is enabled, the `log_bin` system variable, which shows the status of binary logging on the server, is set to ON.

To disable binary logging, you can specify the `--skip-log-bin` or `--disable-log-bin` option at startup. If either of these options is specified and `--log-bin` is also specified, the option specified later takes precedence. When binary logging is disabled, the `log_bin` system variable is set to OFF.

When GTIDs are in use on the server, if you disable binary logging when restarting the server after an abnormal shutdown, some GTIDs are likely to be lost, causing replication to fail. In a normal shutdown, the set of GTIDs from the current binary log file is saved in the `mysql.gtid_executed` table. Following an abnormal shutdown where this did not happen, during recovery the GTIDs are added to the table from the binary log file, provided that binary logging is still enabled. If binary logging is disabled for the server restart, the server cannot access the binary log file to recover the GTIDs, so replication cannot be started. Binary logging can be disabled safely after a normal shutdown.

The `--log-slave-updates` and `--slave-preserve-commit-order` options require binary logging. If you disable binary logging, either omit these options, or specify `--log-slave-updates=OFF` and `--skip-slave-preserve-commit-order`. MySQL disables these options by default when `--skip-log-bin` or `--disable-log-bin` is specified. If you specify `--log-slave-updates` or `--slave-preserve-commit-order` together with `--skip-log-bin` or `--disable-log-bin`, a warning or error message is issued.

In MySQL 5.7, a server ID had to be specified when binary logging was enabled, or the server would not start. In MySQL 8.0, the `server_id` system variable is set to 1 by default. The server can now be started with this default server ID when binary logging is enabled, but an informational message is issued if you do not specify a server ID explicitly using the `--server-id` option. For servers that are used in a replication topology, you must specify a unique nonzero server ID for each server.

For information on the format and management of the binary log, see Section 5.4.4, “The Binary Log”.

- `--log-bin-index[=`*`file_name`*`]`

Property	Value
Command-Line Format	`--log-bin-index=file_name`
System Variable	`log_bin_index`
Scope	Global

Property	Value
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

The name for the binary log index file, which contains the names of the binary log files. By default, it has the same location and base name as the value specified for the binary log files using the `--log-bin` option, plus the extension `.index`. If you do not specify `--log-bin`, the default binary log index file name is `binlog.index`. If you specify `--log-bin` option with no string or an empty string, the default binary log index file name is *`host_name`*`-bin.index`, using the name of the host machine.

For information on the format and management of the binary log, see Section 5.4.4, “The Binary Log”.

Statement selection options. The options in the following list affect which statements are written to the binary log, and thus sent by a replication master server to its slaves. There are also options for slave servers that control which statements received from the master should be executed or ignored. For details, see Section 17.1.6.3, “Replication Slave Options and Variables”.

- `--binlog-do-db=`*`db_name`*

Property	Value
Command-Line Format	`--binlog-do-db=name`
Type	String

This option affects binary logging in a manner similar to the way that `--replicate-do-db` affects replication.

The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of `--replicate-do-db` depend on whether statement-based or row-based replication is in use. You should keep in mind that the format used to log a given statement may not necessarily be the same as that indicated by the value of `binlog_format`. For example, DDL statements such as `CREATE TABLE` and `ALTER TABLE` are always logged as statements, without regard to the logging format in effect, so the following statement-based rules for `--binlog-do-db` always apply in determining whether or not the statement is logged.

Statement-based logging. Only those statements are written to the binary log where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database, use this option multiple times, once for each database; however, doing so does *not* cause cross-database statements such as `UPDATE` *`some_db.some_table`* `SET foo='bar'` to be logged while a different database (or no database) is selected.

Warning

To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

An example of what does not work as you might expect when using statement-based logging: If the server is started with `--binlog-do-db=sales` and you issue the following statements, the `UPDATE` statement is *not* logged:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “just check the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

Another case which may not be self-evident occurs when a given database is replicated even though it was not specified when setting the option. If the server is started with `--binlog-do-db=sales`, the following `UPDATE` statement is logged even though `prices` was not included when setting `--binlog-do-db`:

```
USE sales;
UPDATE prices.discounts SET percentage = percentage + 10;
```

Because `sales` is the default database when the `UPDATE` statement is issued, the `UPDATE` is logged.

Row-based logging. Logging is restricted to database *`db_name`*. Only changes to tables belonging to *`db_name`* are logged; the default database has no effect on this. Suppose that the server is started with `--binlog-do-db=sales` and row-based logging is in effect, and then the following statements are executed:

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

The changes to the `february` table in the `sales` database are logged in accordance with the `UPDATE` statement; this occurs whether or not the `USE` statement was issued. However, when using the row-based logging format and `--binlog-do-db=sales`, changes made by the following `UPDATE` are not logged:

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

Even if the `USE prices` statement were changed to `USE sales`, the `UPDATE` statement's effects would still not be written to the binary log.

Another important difference in `--binlog-do-db` handling for statement-based logging as opposed to the row-based logging occurs with regard to statements that refer to multiple databases. Suppose that the server is started with `--binlog-do-db=db1`, and the following statements are executed:

```
USE db1;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

If you are using statement-based logging, the updates to both tables are written to the binary log. However, when using the row-based format, only the changes to `table1` are logged; `table2` is in a different database, so it is not changed by the `UPDATE`. Now suppose that, instead of the `USE db1` statement, a `USE db4` statement had been used:

```
USE db4;
UPDATE db1.table1 SET col1 = 10, db2.table2 SET col2 = 20;
```

In this case, the `UPDATE` statement is not written to the binary log when using statement-based logging. However, when using row-based logging, the change to `table1` is logged, but not that to `table2`—in other words, only changes to tables in the database named by `--binlog-do-db` are logged, and the choice of default database has no effect on this behavior.

- `--binlog-ignore-db=db_name`

Property	Value
Command-Line Format	`--binlog-ignore-db=name`
Type	String

This option affects binary logging in a manner similar to the way that `--replicate-ignore-db` affects replication.

The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of `--replicate-ignore-db` depend on whether statement-based or row-based replication is in use. You should keep in mind that the format used to log a given statement may not necessarily be the same as that indicated by the value of `binlog_format`. For example, DDL statements such as `CREATE TABLE` and `ALTER TABLE` are always logged as statements, without regard to the logging format in effect, so the following statement-based rules for `--binlog-ignore-db` always apply in determining whether or not the statement is logged.

Statement-based logging. Tells the server to not log any statement where the default database (that is, the one selected by `USE`) is *`db_name`*.

When there is no default database, no `--binlog-ignore-db` options are applied, and such statements are always logged. (Bug #11829838, Bug #60188)

Row-based format. Tells the server not to log updates to any tables in the database *`db_name`*. The current database has no effect.

When using statement-based logging, the following example does not work as you might expect. Suppose that the server is started with `--binlog-ignore-db=sales` and you issue the following statements:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The `UPDATE` statement *is* logged in such a case because `--binlog-ignore-db` applies only to the default database (determined by the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered. However, when using row-based logging, the `UPDATE` statement's effects are *not* written to the binary log, which means that no changes to the `sales.january` table are logged; in this instance, `--binlog-ignore-db=sales` causes *all* changes made to tables in the master's copy of the `sales` database to be ignored for purposes of binary logging.

To specify more than one database to ignore, use this option multiple times, once for each database. Because database names can contain commas, the list will be treated as the name of a single database if you supply a comma-separated list.

You should not use this option if you are using cross-database updates and you do not want these updates to be logged.

Checksum options. MySQL supports reading and writing of binary log checksums. These are enabled using the two options listed here:

- `--binlog-checksum={NONE|CRC32}`

Property	Value
Command-Line Format	`--binlog-checksum=type`

Property	Value
Type	String
Default Value	`CRC32`
Valid Values	`NONE` `CRC32`

Enabling this option causes the master to write checksums for events written to the binary log. Set to `NONE` to disable, or the name of the algorithm to be used for generating checksums; currently, only CRC32 checksums are supported, and CRC32 is the default. You cannot change the setting for this option within a transaction.

- `--master-verify-checksum={0|1}`

Property	Value
Command-Line Format	`--master-verify-checksum[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Enabling this option causes the master to verify events from the binary log using checksums, and to stop with an error in the event of a mismatch. Disabled by default.

To control reading of checksums by the slave (from the relay) log, use the `--slave-sql-verify-checksum` option.

Testing and debugging options. The following binary log options are used in replication testing and debugging. They are not intended for use in normal operations.

- `--max-binlog-dump-events=N`

Property	Value
Command-Line Format	`--max-binlog-dump-events=#`
Type	Integer
Default Value	`0`

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--sporadic-binlog-dump-fail`

Property	Value
Command-Line Format	`--sporadic-binlog-dump-fail[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

This option is used internally by the MySQL test suite for replication testing and debugging.

System Variables Used with Binary Logging

The following list describes system variables for controlling binary logging. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used to control binary logging are listed earlier in this section.

- `binlog_cache_size`

Property	Value
Command-Line Format	`--binlog-cache-size=#`
System Variable	`binlog_cache_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`32768`
Minimum Value	`4096`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The size of the memory buffer to hold changes to the binary log during a transaction. When binary logging is enabled on the server (with the `log_bin` system variable set to ON), a binary log cache is allocated for each client if the server supports any transactional storage engines. If the data for the transaction exceeds the space in the memory buffer, the excess data is stored in a temporary file. When binary log encryption is active on the server, the memory buffer is not encrypted, but (from MySQL 8.0.17) any temporary file used to hold the binary log cache is encrypted. After each transaction is committed, the binary log cache is reset by clearing the memory buffer and truncating the temporary file if used.

If you often use large transactions, you can increase this cache size to get better performance by reducing or eliminating the need to write to temporary files. The `Binlog_cache_use` and `Binlog_cache_disk_use` status variables can be useful for tuning the size of this variable. See Section 5.4.4, “The Binary Log”.

`binlog_cache_size` sets the size for the transaction cache only; the size of the statement cache is governed by the `binlog_stmt_cache_size` system variable.

- `binlog_checksum`

Property	Value
Command-Line Format	`--binlog-checksum=name`
System Variable	`binlog_checksum`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`CRC32`

Valid Values	`NONE`
	
	`CRC32`

When enabled, this variable causes the master to write a checksum for each event in the binary log. `binlog_checksum` supports the values `NONE` (disabled) and `CRC32`. The default is `CRC32`. You cannot change the value of `binlog_checksum` within a transaction.

When `binlog_checksum` is disabled (value `NONE`), the server verifies that it is writing only complete events to the binary log by writing and checking the event length (rather than a checksum) for each event.

Changing the value of this variable causes the binary log to be rotated; checksums are always written to an entire binary log file, and never to only part of one.

Setting this variable on the master to a value unrecognized by the slave causes the slave to set its own `binlog_checksum` value to `NONE`, and to stop replication with an error. (Bug #13553750, Bug #61096) If backward compatibility with older slaves is a concern, you may want to set the value explicitly to `NONE`.

- `binlog_direct_non_transactional_updates`

Property	Value
Command-Line Format	`--binlog-direct-non-transactional-updates[={OFF|ON}]`
System Variable	`binlog_direct_non_transactional_updates`
Scope	Global, Session
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`OFF`

Due to concurrency issues, a slave can become inconsistent when a transaction contains updates to both transactional and nontransactional tables. MySQL tries to preserve causality among these statements by writing nontransactional statements to the transaction cache, which is flushed upon commit. However, problems arise when modifications done to nontransactional tables on behalf of a transaction become immediately visible to other connections because these changes may not be written immediately into the binary log.

The `binlog_direct_non_transactional_updates` variable offers one possible workaround to this issue. By default, this variable is disabled. Enabling `binlog_direct_non_transactional_updates` causes updates to nontransactional tables to be written directly to the binary log, rather than to the transaction cache.

As of MySQL 8.0.14, setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

`binlog_direct_non_transactional_updates` works only for statements that are replicated using the statement-based binary logging format; that is, it works only when the value of `binlog_format` is `STATEMENT`, or when `binlog_format` is `MIXED` and a given statement is being replicated using the statement-based format. This variable has no effect when the binary log format is `ROW`, or when `binlog_format` is set to `MIXED` and a given statement is replicated using the row-based format.

Important

Before enabling this variable, you must make certain that there are no dependencies between transactional and nontransactional tables; an example

> of such a dependency would be the statement `INSERT INTO myisam_table SELECT * FROM innodb_table`. Otherwise, such statements are likely to cause the slave to diverge from the master.

This variable has no effect when the binary log format is `ROW` or `MIXED`.

- `binlog_encryption`

Property	Value
Command-Line Format	`--binlog-encryption[={OFF|ON}]`
Introduced	8.0.14
System Variable	`binlog_encryption`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enables encryption for binary log files and relay log files on this server. `OFF` is the default. `ON` sets encryption on for binary log files and relay log files. Binary logging does not need to be enabled on the server to enable encryption, so you can encrypt the relay log files on a slave that has no binary log. To use encryption, a keyring plugin must be installed and configured to supply MySQL Server's keyring service. For instructions to do this, see Section 6.4.4, “The MySQL Keyring”. Any supported keyring plugin can be used to store binary log encryption keys.

When you first start the server with binary log encryption enabled, a new binary log encryption key is generated before the binary log and relay logs are initialized. This key is used to encrypt a file password for each binary log file (if the server has binary logging enabled) and relay log file (if the server has replication channels), and further keys generated from the file passwords are used to encrypt the data in the files. Relay log files are encrypted for all channels, including Group Replication applier channels and new channels that are created after encryption is activated. The binary log index file and relay log index file are never encrypted.

If you activate encryption while the server is running, a new binary log encryption key is generated at that time. The exception is if encryption was active previously on the server and was then disabled, in which case the binary log encryption key that was in use before is used again. The binary log file and relay log files are rotated immediately, and file passwords for the new files and all subsequent binary log files and relay log files are encrypted using this binary log encryption key. Existing binary log files and relay log files still present on the server are not automatically encrypted, but you can purge them if they are no longer needed.

If you deactivate encryption by changing the `binlog_encryption` system variable to `OFF`, the binary log file and relay log files are rotated immediately and all subsequent logging is unencrypted. Previously encrypted files are not automatically decrypted, but the server is still able to read them. `SUPER` privileges or the `BINLOG_ENCRYPTION_ADMIN` privilege are required to activate or deactivate encryption while the server is running. Group Replication applier channels are not included in the relay log rotation request, so unencrypted logging for these channels does not start until their logs are rotated in normal use.

For more information on binary log file and relay log file encryption, see Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”.

- `binlog_error_action`

Property	Value
Command-Line Format	`--binlog-error-action[=value]`
System Variable	`binlog_error_action`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`ABORT_SERVER`
Valid Values	`IGNORE_ERROR` `ABORT_SERVER`

Controls what happens when the server encounters an error such as not being able to write to, flush or synchronize the binary log, which can cause the master's binary log to become inconsistent and replication slaves to lose synchronization.

This variable defaults to `ABORT_SERVER`, which makes the server halt logging and shut down whenever it encounters such an error with the binary log. On restart, recovery proceeds as in the case of an unexpected server halt (see Section 17.4.2, “Handling an Unexpected Halt of a Replication Slave”).

When `binlog_error_action` is set to `IGNORE_ERROR`, if the server encounters such an error it continues the ongoing transaction, logs the error then halts logging, and continues performing updates. To resume binary logging `log_bin` must be enabled again, which requires a server restart. This setting provides backward compatibility with older versions of MySQL.

- `binlog_expire_logs_seconds`

Property	Value
Command-Line Format	`--binlog-expire-logs-seconds=#`
Introduced	8.0.1
System Variable	`binlog_expire_logs_seconds`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.11)	`2592000`
Default Value (<= 8.0.4)	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

Sets the binary log expiration period in seconds. After their expiration period ends, binary log files can be automatically removed. Possible removals happen at startup and when the binary log is flushed. Log flushing occurs as indicated in Section 5.4, “MySQL Server Logs”.

The default binary log expiration period is 2592000 seconds, which equals 30 days (30*24*60*60 seconds). The default applies if neither `binlog_expire_logs_seconds` nor the deprecated system

variable `expire_logs_days` has a value set at startup. If a non-zero value for one of the variables `binlog_expire_logs_seconds` or `expire_logs_days` is set at startup, this value is used as the binary log expiration period. If a non-zero value for both of those variables is set at startup, the value for `binlog_expire_logs_seconds` is used as the binary log expiration period, and the value for `expire_logs_days` is ignored with a warning message.

To disable automatic purging of the binary log, specify a value of 0 explicitly for `binlog_expire_logs_seconds`, and do not specify a value for `expire_logs_days`. For compatibility with earlier releases, automatic purging is also disabled if you specify a value of 0 explicitly for `expire_logs_days` and do not specify a value for `binlog_expire_logs_seconds`. In that case, the default for `binlog_expire_logs_seconds` is not applied.

To remove binary log files manually, use the `PURGE BINARY LOGS` statement. See Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

- `binlog_format`

Property	Value
Command-Line Format	`--binlog-format=format`
System Variable	`binlog_format`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`ROW`

Valid Values	`ROW`
	`STATEMENT`
	`MIXED`

This variable sets the binary logging format, and can be any one of `STATEMENT`, `ROW`, or `MIXED`. See Section 17.2.1, “Replication Formats”.

`binlog_format` can be set at startup or at runtime, except that under some conditions, changing this variable at runtime is not possible or causes replication to fail, as described later.

The default is `ROW`. *Exception*: In NDB Cluster, the default is `MIXED`; statement-based replication is not supported for NDB Cluster.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

The rules governing when changes to this variable take effect and how long the effect lasts are the same as for other MySQL server system variables. For more information, see Section 13.7.6.1, “SET Syntax for Variable Assignment”.

When `MIXED` is specified, statement-based replication is used, except for cases where only row-based replication is guaranteed to lead to proper results. For example, this happens when statements contain user-defined functions (UDF) or the `UUID()` function.

For details of how stored programs (stored procedures and functions, triggers, and events) are handled when each binary logging format is set, see Section 24.7, “Stored Program Binary Logging”.

There are exceptions when you cannot switch the replication format at runtime:

- The replication format cannot be changed from within a stored function or a trigger.
- If a session has open temporary tables, the replication format cannot be changed for the session (`SET @@SESSION.binlog_format`).
- If any replication channel has open temporary tables, the replication format cannot be changed globally (`SET @@GLOBAL.binlog_format` or `SET @@PERSIST.binlog_format`).
- If any replication channel applier thread is currently running, the replication format cannot be changed globally (`SET @@GLOBAL.binlog_format` or `SET @@PERSIST.binlog_format`).

Trying to switch the replication format in any of these cases (or attempting to set the current replication format) results in an error. You can, however, use `PERSIST_ONLY` (`SET @@PERSIST_ONLY.binlog_format`) to change the replication format at any time, because this action does not modify the runtime global system variable value, and takes effect only after a server restart.

Switching the replication format at runtime is not recommended when any temporary tables exist, because temporary tables are logged only when using statement-based replication, whereas with row-based replication and mixed replication, they are not logged.

Changing the logging format on a replication master does not cause a replication slave to change its logging format to match. Switching the replication format while replication is ongoing can cause issues if a replication slave has binary logging enabled, and the change results in the slave using `STATEMENT` format logging while the master is using `ROW` or `MIXED` format logging. A replication slave is not able to convert binary log entries received in `ROW` logging format to `STATEMENT` format for use in its own binary log, so this situation can cause replication to fail. For more information, see Section 5.4.4.2, “Setting The Binary Log Format”.

The binary log format affects the behavior of the following server options:

- `--replicate-do-db`
- `--replicate-ignore-db`
- `--binlog-do-db`
- `--binlog-ignore-db`

These effects are discussed in detail in the descriptions of the individual options.

- `binlog_group_commit_sync_delay`

Property	Value
Command-Line Format	`--binlog-group-commit-sync-delay=#`
System Variable	`binlog_group_commit_sync_delay`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1000000`

Controls how many microseconds the binary log commit waits before synchronizing the binary log file to disk. By default `binlog_group_commit_sync_delay` is set to 0, meaning that there is no delay. Setting `binlog_group_commit_sync_delay` to a microsecond delay enables more transactions to be synchronized together to disk at once, reducing the overall time to commit a group of transactions because the larger groups require fewer time units per group.

When `sync_binlog=0` or `sync_binlog=1` is set, the delay specified by `binlog_group_commit_sync_delay` is applied for every binary log commit group before synchronization (or in the case of `sync_binlog=0`, before proceeding). When `sync_binlog` is set to a value *n* greater than 1, the delay is applied after every *n* binary log commit groups.

Setting `binlog_group_commit_sync_delay` can increase the number of parallel committing transactions on any server that has (or might have after a failover) a replication slave, and therefore can increase parallel execution on the replication slaves. To benefit from this effect, the slave servers must have `slave_parallel_type=LOGICAL_CLOCK` set, and the effect is more significant when `binlog_transaction_dependency_tracking=COMMIT_ORDER` is also set. It is important to take into account both the master's throughput and the slaves' throughput when you are tuning the setting for `binlog_group_commit_sync_delay`.

Setting `binlog_group_commit_sync_delay` can also reduce the number of `fsync()` calls to the binary log on any server (master or slave) that has a binary log.

Note that setting `binlog_group_commit_sync_delay` increases the latency of transactions on the server, which might affect client applications. Also, on highly concurrent workloads, it is possible for the delay to increase contention and therefore reduce throughput. Typically, the benefits of setting a delay outweigh the drawbacks, but tuning should always be carried out to determine the optimal setting.

- `binlog_group_commit_sync_no_delay_count`

Property	Value
Command-Line Format	`--binlog-group-commit-sync-no-delay-count=#`
System Variable	`binlog_group_commit_sync_no_delay_count`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1000000`

The maximum number of transactions to wait for before aborting the current delay as specified by `binlog_group_commit_sync_delay`. If `binlog_group_commit_sync_delay` is set to 0, then this option has no effect.

- `binlog_max_flush_queue_time`

Property	Value
Command-Line Format	`--binlog-max-flush-queue-time=#`
Deprecated	Yes
System Variable	`binlog_max_flush_queue_time`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`100000`

`binlog_max_flush_queue_time` is deprecated, and is marked for eventual removal in a future MySQL release. Formerly, this system variable controlled the time in microseconds to continue reading transactions from the flush queue before proceeding with group commit. It no longer has any effect.

- `binlog_order_commits`

Property	Value
Command-Line Format	`--binlog-order-commits[={OFF|ON}]`
System Variable	`binlog_order_commits`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

When this variable is enabled on a replication master (which is the default), transaction commit instructions issued to storage engines are serialized on a single thread, so that transactions are always committed in the same order as they are written to the binary log. Disabling this variable permits transaction commit instructions to be issued using multiple threads. Used in combination with binary log group commit, this prevents the commit rate of a single transaction being a bottleneck to throughput, and might therefore produce a performance improvement.

Transactions are written to the binary log at the point when all the storage engines involved have confirmed that the transaction is prepared to commit. The binary log group commit logic then commits a group of transactions after their binary log write has taken place. When `binlog_order_commits` is disabled, because multiple threads are used for this process, transactions in a commit group might be committed in a different order from their order in the binary log. (Transactions from a single client always commit in chronological order.) In many cases this does not matter, as operations carried out in separate transactions should produce consistent results, and if that is not the case, a single transaction ought to be used instead.

If you want to ensure that the transaction history on the master and on a multithreaded replication slave remains identical, set `slave_preserve_commit_order=1` on the replication slave.

- `binlog_rotate_encryption_master_key_at_startup`

Property	Value
Command-Line Format	`--binlog-rotate-encryption-master-key-at-startup[={OFF|ON}]`
Introduced	8.0.14
System Variable	`binlog_rotate_encryption_master_key_at_startup`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Specifies whether or not the binary log master key is rotated at server startup. The binary log master key is the binary log encryption key that is used to encrypt file passwords for the binary log files and relay log files on the server. When a server is started for the first time with binary log encryption enabled (`binlog_encryption=ON`), a new binary log encryption key is generated and used as the binary log master key. If the `binlog_rotate_encryption_master_key_at_startup` system variable is also set to `ON`, whenever the server is restarted, a further binary log encryption key is generated and used as the binary log master key for all subsequent binary log files and relay log files. If the `binlog_rotate_encryption_master_key_at_startup` system variable is set to `OFF`, which is the default, the existing binary log master key is used again after the server restarts. For more information on binary log encryption keys and the binary log master key, see Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”.

- `binlog_row_event_max_size`

Property	Value
Command-Line Format	`--binlog-row-event-max-size=#`
System Variable (>= 8.0.14)	`binlog_row_event_max_size`
Scope (>= 8.0.14)	Global
Dynamic (>= 8.0.14)	No
SET_VAR Hint Applies (>= 8.0.14)	No
Type	Integer
Default Value	`8192`
Minimum Value	`256`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

When row-based binary logging is used, this setting is a soft limit on the maximum size of a row-based binary log event, in bytes. Where possible, rows stored in the binary log are grouped into events with a size not exceeding the value of this setting. If an event cannot be split, the maximum size can be exceeded. The value must be (or else gets rounded down to) a multiple of 256. The default is 8192 bytes.

This global system variable is read-only and can be set only at server startup. Its value can therefore only be modified by using the `PERSIST_ONLY` keyword or the `@@persist_only` qualifier with the `SET` statement.

- `binlog_row_image`

Property	Value
Command-Line Format	`--binlog-row-image=image_type`
System Variable	`binlog_row_image`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`full`
Valid Values	`full` (Log all columns) `minimal` (Log only changed columns, and columns needed to identify rows) `noblob` (Log all columns, except for unneeded BLOB and TEXT columns)

For MySQL row-based replication, this variable determines how row images are written to the binary log.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

In MySQL row-based replication, each row change event contains two images, a “before” image whose columns are matched against when searching for the row to be updated, and an “after” image containing the changes. Normally, MySQL logs full rows (that is, all columns) for both the before and after images. However, it is not strictly necessary to include every column in both images, and we can often save disk, memory, and network usage by logging only those columns which are actually required.

Note

When deleting a row, only the before image is logged, since there are no changed values to propagate following the deletion. When inserting a row, only the after image is logged, since there is no existing row to be matched. Only when updating a row are both the before and after images required, and both written to the binary log.

For the before image, it is necessary only that the minimum set of columns required to uniquely identify rows is logged. If the table containing the row has a primary key, then only the primary key column or columns are written to the binary log. Otherwise, if the table has a unique key all of whose columns are `NOT NULL`, then only the columns in the unique key need be logged. (If the table has neither a primary key nor a unique key without any `NULL` columns, then all columns must be used in the before image, and logged.) In the after image, it is necessary to log only the columns which have actually changed.

You can cause the server to log full or minimal rows using the `binlog_row_image` system variable. This variable actually takes one of three possible values, as shown in the following list:

- `full`: Log all columns in both the before image and the after image.
- `minimal`: Log only those columns in the before image that are required to identify the row to be changed; log only those columns in the after image where a value was specified by the SQL statement, or generated by auto-increment.
- `noblob`: Log all columns (same as `full`), except for `BLOB` and `TEXT` columns that are not required to identify rows, or that have not changed.

Note

This variable is not supported by NDB Cluster; setting it has no effect on the logging of `NDB` tables.

The default value is `full`.

When using `minimal` or `noblob`, deletes and updates are guaranteed to work correctly for a given table if and only if the following conditions are true for both the source and destination tables:

- All columns must be present and in the same order; each column must use the same data type as its counterpart in the other table.
- The tables must have identical primary key definitions.

(In other words, the tables must be identical with the possible exception of indexes that are not part of the tables' primary keys.)

If these conditions are not met, it is possible that the primary key column values in the destination table may prove insufficient to provide a unique match for a delete or update. In this event, no warning or error is issued; the master and slave silently diverge, thus breaking consistency.

Setting this variable has no effect when the binary logging format is `STATEMENT`. When `binlog_format` is `MIXED`, the setting for `binlog_row_image` is applied to changes that are logged using row-based format, but this setting no effect on changes logged as statements.

Setting `binlog_row_image` on either the global or session level does not cause an implicit commit; this means that this variable can be changed while a transaction is in progress without affecting the transaction.

- `binlog_row_metadata`

Property	Value
Command-Line Format	`--binlog-row-metadata=metadata_type`
Introduced	8.0.1
System Variable	`binlog_row_metadata`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`MINIMAL`
Valid Values	`FULL` (All metadata is included)

Property	Value
	`MINIMAL` (Limit included metadata)

Configures the amount of table metadata added to the binary log when using row-based logging. When set to `MINIMAL`, the default, only metadata related to `SIGNED` flags, column character set and geometry types are logged. When set to `FULL` complete metadata for tables is logged, such as column name, `ENUM` or `SET` string values, `PRIMARY KEY` information, and so on.

The extended metadata serves the following purposes:

- Slaves use the metadata to transfer data when its table structure is different from the master's.
- External software can use the metadata to decode row events and store the data into external databases, such as a data warehouse.

- `binlog_row_value_options`

Property	Value
Command-Line Format	`--binlog-row-value-options=#`
Introduced	8.0.3
System Variable	`binlog_row_value_options`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Set
Default Value	`''`
Valid Values	`PARTIAL_JSON`

When set to `PARTIAL_JSON`, this enables use of a space-efficient binary log format for updates that modify only a small portion of a JSON document, which causes row-based replication to write only the modified parts of the JSON document to the after-image for the update in the binary log (rather than writing the full document). This works for an `UPDATE` statement which modifies a JSON column using any sequence of `JSON_SET()`, `JSON_REPLACE()`, and `JSON_REMOVE()`. If the modification requires more space than the full document, or if the server is unable to generate a partial update, the full document is used instead.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

`PARTIAL_JSON` is the only supported value; to unset `binlog_row_value_options`, set its value to the empty string.

`binlog_row_value_options=PARTIAL_JSON` takes effect only when binary logging is enabled and `binlog_format` is set to `ROW` or `MIXED`. Statement-based replication *always* logs only the modified parts of the JSON document, regardless of any value set for `binlog_row_value_options`. To maximize the amount of space saved, use `binlog_row_image=NOBLOB` or `binlog_row_image=MINIMAL` together with this option. `binlog_row_image=FULL` saves less space than either of these, since the full JSON document is stored in the before-image, and the partial update is stored only in the after-image.

`mysqlbinlog` output includes partial JSON updates in the form of events encoded as base-64 strings using `BINLOG` statements. If the `--verbose` option is specified, `mysqlbinlog` displays the partial JSON updates as readable JSON using pseudo-SQL statements.

MySQL Replication generates an error if a modification cannot be applied to the JSON document on the slave. This includes a failure to find the path. Be aware that, even with this and other safety checks, if a JSON document on a slave has diverged from that on the master and a partial update is applied, it remains theoretically possible to produce a valid but unexpected JSON document on the slave.

- `binlog_rows_query_log_events`

Property	Value
Command-Line Format	`--binlog-rows-query-log-events[={OFF|ON}]`
System Variable	`binlog_rows_query_log_events`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This system variable affects row-based logging only. When enabled, it causes the server to write informational log events such as row query log events into its binary log. This information can be used for debugging and related purposes, such as obtaining the original query issued on the master when it cannot be reconstructed from the row updates.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

These informational events are normally ignored by MySQL programs reading the binary log and so cause no issues when replicating or restoring from backup. To view them, increase the verbosity level by using mysqlbinlog's `--verbose` option twice, either as `-vv` or `--verbose --verbose`.

- `binlog_stmt_cache_size`

Property	Value
Command-Line Format	`--binlog-stmt-cache-size=#`
System Variable	`binlog_stmt_cache_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`32768`
Minimum Value	`4096`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The size of the memory buffer for the binary log to hold nontransactional statements issued during a transaction. When binary logging is enabled on the server (with the `log_bin` system variable set to ON), separate binary log transaction and statement caches are allocated for each client if the server supports any transactional storage engines. If the data for the nontransactional statements used in the transaction exceeds the space in the memory buffer, the excess data is stored in a temporary file. When binary log encryption is active on the server, the memory buffer is not encrypted, but (from MySQL 8.0.17) any temporary file used to hold the binary log cache is encrypted. After each transaction is committed, the binary log statement cache is reset by clearing the memory buffer and truncating the temporary file if used.

If you often use large nontransactional statements during transactions, you can increase this cache size to get better performance by reducing or eliminating the need to write to temporary files. The `Binlog_stmt_cache_use` and `Binlog_stmt_cache_disk_use` status variables can be useful for tuning the size of this variable. See Section 5.4.4, “The Binary Log”.

The `binlog_cache_size` system variable sets the size for the transaction cache.

- `binlog_transaction_dependency_tracking`

Property	Value
Command-Line Format	`--binlog-transaction-dependency-tracking=value`
Introduced	8.0.1
System Variable	`binlog_transaction_dependency_tracking`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`COMMIT_ORDER`

Valid Values	`COMMIT_ORDER`
	`WRITESET`
	`WRITESET_SESSION`

The source of dependency information that the master uses to determine which transactions can be executed in parallel by the slave's multithreaded applier. This variable can take one of the three values described in the following list:

- `COMMIT_ORDER`: Dependency information is generated from the master's commit timestamps. This is the default. This mode is also used for any transactions without write sets, even if this variable's is `WRITESET` or `WRITESET_SESSION`; this is also the case for transactions updating tables without primary keys and transactions updating tables having foreign key constraints.
- `WRITESET`: Dependency information is generated from the master's write set, and any transactions which write different tuples can be parallelized.
- `WRITESET_SESSION`: Dependency information is generated from the master's write set, but no two updates from the same session can be reordered.

`WRITESET` and `WRITESET_SESSION` modes do not deliver any transaction dependencies that are newer than those that would have been returned in `COMMIT_ORDER` mode.

The value of this variable cannot be set to anything other than `COMMIT_ORDER` if `transaction_write_set_extraction` is `OFF`. You should also note that the value of `transaction_write_set_extraction` cannot be changed if the current value of `binlog_transaction_dependency_tracking` is `WRITESET` or `WRITESET_SESSION`.

The number of row hashes to be kept and checked for the latest transaction to have changed a given row is determined by the value of `binlog_transaction_dependency_history_size`.

- `binlog_transaction_dependency_history_size`

Property	Value
Command-Line Format	`--binlog-transaction-dependency-history-size=#`
Introduced	8.0.1
System Variable	`binlog_transaction_dependency_history_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`25000`
Minimum Value	`1`
Maximum Value	`1000000`

Sets an upper limit on the number of row hashes which are kept in memory and used for looking up the transaction that last modified a given row. Once this number of hashes has been reached, the history is purged.

- `expire_logs_days`

Property	Value
Command-Line Format	`--expire-logs-days=#`
Deprecated	8.0.3
System Variable	`expire_logs_days`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.11)	`0`
Default Value (>= 8.0.2, <= 8.0.4)	`30`
Default Value (<= 8.0.1)	`0`
Minimum Value	`0`
Maximum Value	`99`

Specifies the number of days before automatic removal of binary log files. `expire_logs_days` is deprecated, and will be removed in a future release. Instead, use `binlog_expire_logs_seconds`,

which sets the binary log expiration period in seconds. If you do not set a value for either system variable, the default expiration period is 30 days. Possible removals happen at startup and when the binary log is flushed. Log flushing occurs as indicated in Section 5.4, “MySQL Server Logs”.

Any non-zero value that you specify for `expire_logs_days` is ignored if `binlog_expire_logs_seconds` is also specified, and the value of `binlog_expire_logs_seconds` is used instead as the binary log expiration period. A warning message is issued in this situation. A non-zero value for `expire_logs_days` is only applied as the binary log expiration period if `binlog_expire_logs_seconds` is not specified or is specified as 0.

To disable automatic purging of the binary log, specify a value of 0 explicitly for `binlog_expire_logs_seconds`, and do not specify a value for `expire_logs_days`. For compatibility with earlier releases, automatic purging is also disabled if you specify a value of 0 explicitly for `expire_logs_days` and do not specify a value for `binlog_expire_logs_seconds`. In that case, the default for `binlog_expire_logs_seconds` is not applied.

To remove binary log files manually, use the `PURGE BINARY LOGS` statement. See Section 13.4.1.1, “PURGE BINARY LOGS Syntax”.

- `log_bin`

Property	Value
System Variable	`log_bin`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean

Shows the status of binary logging on the server, either enabled (`ON`) or disabled (`OFF`). With binary logging enabled, the server logs all statements that change data to the binary log, which is used for backup and replication. `ON` means that the binary log is available, `OFF` means that it is not in use. The `--log-bin` option can be used to specify a base name and location for the binary log.

In earlier MySQL versions, binary logging was disabled by default, and was enabled if you specified the `--log-bin` option. From MySQL 8.0, binary logging is enabled by default, with the `log_bin` system variable set to `ON`, whether or not you specify the `--log-bin` option. The exception is if you use `mysqld` to initialize the data directory manually by invoking it with the `--initialize` or `--initialize-insecure` option, when binary logging is disabled by default. It is possible to enable binary logging in this case by specifying the `--log-bin` option.

If the `--skip-log-bin` or `--disable-log-bin` option is specified at startup, binary logging is disabled, with the `log_bin` system variable set to `OFF`. If either of these options is specified and `--log-bin` is also specified, the option specified later takes precedence.

For information on the format and management of the binary log, see Section 5.4.4, “The Binary Log”.

- `log_bin_basename`

Property	Value
System Variable	`log_bin_basename`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	File name

Holds the base name and path for the binary log files, which can be set with the `--log-bin` server option. In MySQL 8.0, if the `--log-bin` option is not supplied, the default base name is `binlog`. For compatibility with MySQL 5.7, if the `--log-bin` option is supplied with no string or with an empty string, the default base name is *`host_name`*`-bin`, using the name of the host machine. The default location is the data directory.

- `log_bin_index`

Property	Value
Command-Line Format	`--log-bin-index=file_name`
System Variable	`log_bin_index`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

Holds the base name and path for the binary log index file, which can be set with the `--log-bin-index` server option.

- `log_bin_trust_function_creators`

Property	Value
Command-Line Format	`--log-bin-trust-function-creators[={OFF|ON}]`
System Variable	`log_bin_trust_function_creators`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This variable applies when binary logging is enabled. It controls whether stored function creators can be trusted not to create stored functions that will cause unsafe events to be written to the binary log. If set to 0 (the default), users are not permitted to create or alter stored functions unless they have the `SUPER` privilege in addition to the `CREATE ROUTINE` or `ALTER ROUTINE` privilege. A setting of 0 also enforces the restriction that a function must be declared with the `DETERMINISTIC` characteristic, or with the `READS SQL DATA` or `NO SQL` characteristic. If the variable is set to 1, MySQL does not enforce these restrictions on stored function creation. This variable also applies to trigger creation. See Section 24.7, “Stored Program Binary Logging”.

- `log_bin_use_v1_row_events`

Property	Value
Command-Line Format	`--log-bin-use-v1-row-events[={OFF|ON}]`
Deprecated	8.0.18

Property	Value
System Variable	`log_bin_use_v1_row_events`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This read-only system variable is deprecated. Setting the system variable to `ON` at server startup enabled row-based replication with slaves running MySQL Server 5.5 and earlier by writing the binary log using Version 1 binary log row events, instead of Version 2 binary log row events which are the default as of MySQL 5.6.

- `log_builtin_as_identified_by_password`

Property	Value
Command-Line Format	`--log-builtin-as-identified-by-password[={OFF|ON}]`
Removed	8.0.11
System Variable	`log_builtin_as_identified_by_password`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

This system variable was removed in MySQL 8.0.11.

- `log_slave_updates`

Property	Value
Command-Line Format	`--log-slave-updates[={OFF|ON}]`
System Variable	`log_slave_updates`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value (>= 8.0.3)	`ON`
Default Value (<= 8.0.2)	`OFF`

Whether updates received by a slave server from a master server should be logged to the slave's own binary log.

Enabling this variable causes the slave to write the updates that are received from a master server and performed by the slave's SQL thread to the slave's own binary log. Binary logging, which is controlled by the `--log-bin` option and is enabled by default, must also be enabled on the slave for updates to be logged. See Section 17.1.6, “Replication and Binary Logging Options and Variables”.

`log_slave_updates` is enabled by default, unless you specify `--skip-log-bin` to disable binary logging, in which case MySQL also disables slave update logging by default. If you need to disable slave update logging when binary logging is enabled, specify `--log-slave-updates=OFF` at slave server startup.

Enabling `log_slave_updates` enables replication servers to be chained. For example, you might want to set up replication servers using this arrangement:

```
A -> B -> C
```

Here, `A` serves as the master for the slave `B`, and `B` serves as the master for the slave `C`. For this to work, `B` must be both a master *and* a slave. With binary logging enabled and `log_slave_updates` enabled, which are the default settings, updates received from `A` are logged by `B` to its binary log, and can therefore be passed on to `C`.

- `log_statements_unsafe_for_binlog`

Property	Value
Command-Line Format	`--log-statements-unsafe-for-binlog[={OFF|ON}]`
System Variable	`log_statements_unsafe_for_binlog`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

If error 1592 is encountered, controls whether the generated warnings are added to the error log or not.

- `master_verify_checksum`

Property	Value
Command-Line Format	`--master-verify-checksum[={OFF|ON}]`
System Variable	`master_verify_checksum`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enabling this variable causes the master to examine checksums when reading from the binary log. `master_verify_checksum` is disabled by default; in this case, the master uses the event length from the binary log to verify events, so that only complete events are read from the binary log.

- `max_binlog_cache_size`

Property	Value
Command-Line Format	`--max-binlog-cache-size=#`
System Variable	`max_binlog_cache_size`

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	18446744073709551615
Minimum Value	4096
Maximum Value	18446744073709551615

If a transaction requires more than this many bytes of memory, the server generates a `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage` error. The minimum value is 4096. The maximum possible value is 16EiB (exbibytes). The maximum recommended value is 4GB; this is due to the fact that MySQL currently cannot work with binary log positions greater than 4GB.

`max_binlog_cache_size` sets the size for the transaction cache only; the upper limit for the statement cache is governed by the `max_binlog_stmt_cache_size` system variable.

The visibility to sessions of `max_binlog_cache_size` matches that of the `binlog_cache_size` system variable; in other words, changing its value affects only new sessions that are started after the value is changed.

- `max_binlog_size`

Property	Value
Command-Line Format	`--max-binlog-size=#`
System Variable	`max_binlog_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1073741824
Minimum Value	4096
Maximum Value	1073741824

If a write to the binary log causes the current log file size to exceed the value of this variable, the server rotates the binary logs (closes the current file and opens the next one). The minimum value is 4096 bytes. The maximum and default value is 1GB. Encrypted binary log files have an additional 512-byte header, which is included in `max_binlog_size`.

A transaction is written in one chunk to the binary log, so it is never split between several binary logs. Therefore, if you have big transactions, you might see binary log files larger than `max_binlog_size`.

If `max_relay_log_size` is 0, the value of `max_binlog_size` applies to relay logs as well.

With GTIDs in use on the server, when `max_binlog_size` is reached, if the system table `mysql.gtid_executed` cannot be accessed to write the GTIDs from the current binary log file, the binary log cannot be rotated. In this situation, the server responds according to its

`binlog_error_action` setting. If `IGNORE_ERROR` is set, an error is logged on the server and binary logging is halted, or if `ABORT_SERVER` is set, the server shuts down.

- `max_binlog_stmt_cache_size`

Property	Value
Command-Line Format	`--max-binlog-stmt-cache-size=#`
System Variable	`max_binlog_stmt_cache_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`18446744073709547520`
Minimum Value	`4096`
Maximum Value	`18446744073709547520`

If nontransactional statements within a transaction require more than this many bytes of memory, the server generates an error. The minimum value is 4096. The maximum and default values are 4GB on 32-bit platforms and 16EB (exabytes) on 64-bit platforms.

`max_binlog_stmt_cache_size` sets the size for the statement cache only; the upper limit for the transaction cache is governed exclusively by the `max_binlog_cache_size` system variable.

- `original_commit_timestamp`

Property	Value
Introduced	8.0.1
System Variable	`original_commit_timestamp`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Numeric

For internal use by replication. When re-executing a transaction on a slave, this is set to the time when the transaction was committed on the original master, measured in microseconds since the epoch. This allows the original commit timestamp to be propagated throughout a replication topology.

Setting the session value of this system variable is a restricted operation. The session user must have either the `REPLICATION_APPLIER` privilege (see Section 17.3.3, “Replication Privilege Checks”), or privileges sufficient to set restricted session variables (see Section 5.1.9.1, “System Variable Privileges”). However, note that the variable is not intended for users to set; it is set automatically by the replication infrastructure.

- `sql_log_bin`

Property	Value
System Variable	`sql_log_bin`
Scope	Session

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

This variable controls whether logging to the binary log is enabled for the current session (assuming that the binary log itself is enabled). The default value is ON. To disable or enable binary logging for the current session, set the session sql_log_bin variable to OFF or ON.

Set this variable to OFF for a session to temporarily disable binary logging while making changes to the master you do not want replicated to the slave.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

It is not possible to set the session value of sql_log_bin within a transaction or subquery.

Setting this variable to OFF prevents GTIDs from being assigned to transactions in the binary log. If you are using GTIDs for replication, this means that even when binary logging is later enabled again, the GTIDs written into the log from this point do not account for any transactions that occurred in the meantime, so in effect those transactions are lost.

- sync_binlog

Property	Value
Command-Line Format	--sync-binlog=#
System Variable	sync_binlog
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1
Minimum Value	0
Maximum Value	4294967295

Controls how often the MySQL server synchronizes the binary log to disk.

- sync_binlog=0: Disables synchronization of the binary log to disk by the MySQL server. Instead, the MySQL server relies on the operating system to flush the binary log to disk from time to time as it does for any other file. This setting provides the best performance, but in the event of a power failure or operating system crash, it is possible that the server has committed transactions that have not been synchronized to the binary log.

- sync_binlog=1: Enables synchronization of the binary log to disk before transactions are committed. This is the safest setting but can have a negative impact on performance due to the increased number of disk writes. In the event of a power failure or operating system crash, transactions that are missing from the binary log are only in a prepared state. This permits the automatic recovery routine to roll back the transactions, which guarantees that no transaction is lost from the binary log.

- `sync_binlog=N`, where `N` is a value other than 0 or 1: The binary log is synchronized to disk after `N` binary log commit groups have been collected. In the event of a power failure or operating system crash, it is possible that the server has committed transactions that have not been flushed to the binary log. This setting can have a negative impact on performance due to the increased number of disk writes. A higher value improves performance, but with an increased risk of data loss.

For the greatest possible durability and consistency in a replication setup that uses `InnoDB` with transactions, use these settings:

- `sync_binlog=1`.
- `innodb_flush_log_at_trx_commit=1`.

Caution

Many operating systems and some disk hardware fool the flush-to-disk operation. They may tell `mysqld` that the flush has taken place, even though it has not. In this case, the durability of transactions is not guaranteed even with the recommended settings, and in the worst case, a power outage can corrupt `InnoDB` data. Using a battery-backed disk cache in the SCSI disk controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try to disable the caching of disk writes in hardware caches.

- `transaction_write_set_extraction`

Property	Value
Command-Line Format	`--transaction-write-set-extraction[=value]`
System Variable	`transaction_write_set_extraction`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value (>= 8.0.2)	`XXHASH64`
Default Value	`OFF`

Valid Values	`OFF`
	`MURMUR32`
	`XXHASH64`

Defines the algorithm used to hash the writes extracted during a transaction. If you are using Group Replication, this variable must be set to `XXHASH64` because the process of extracting the writes from a transaction is required for conflict detection on all group members. See Section 18.9.1, “Group Replication Requirements”.

As of MySQL 8.0.14, setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Note

The value of this variable cannot be changed when `binlog_transaction_dependency_tracking` is set to either of `WRITESET` or `WRITESET_SESSION`.

17.1.6.5 Global Transaction ID Options and Variables

- Startup Options Used with GTID Replication
- System Variables Used with GTID Replication

The MySQL Server options and system variables described in this section are used to monitor and control Global Transaction Identifiers (GTIDs).

For additional information, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

Startup Options Used with GTID Replication

The following server startup options are used with GTID-based replication:

- `--gtid-executed-compression-period`

Property	Value
Command-Line Format	`--gtid-executed-compression-period=#`
Type	Integer
Default Value	`1000`
Minimum Value	`0`
Maximum Value	`4294967295`

Compress the `mysql.gtid_executed` table each time this many transactions have taken place. A setting of 0 means that this table is not compressed. No compression of the table occurs when binary logging is enabled, therefore the option has no effect unless `log_bin` is `OFF`.

See mysql.gtid_executed Table Compression, for more information.

System Variables Used with GTID Replication

The following system variables are used with GTID-based replication:

- `binlog_gtid_simple_recovery`

Property	Value
Command-Line Format	`--binlog-gtid-simple-recovery[={OFF|ON}]`
System Variable	`binlog_gtid_simple_recovery`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`ON`

This variable controls how binary log files are iterated during the search for GTIDs when MySQL starts or restarts.

When `binlog_gtid_simple_recovery=TRUE`, which is the default in MySQL 8.0, the values of `gtid_executed` and `gtid_purged` are computed at startup based on the values of `Previous_gtids_log_event` in the most recent and oldest binary log files. For a description of the computation, see The `gtid_purged` System Variable. This setting accesses only two binary log files during server restart. If all binary logs on the server were generated using MySQL 5.7.8 or later, `binlog_gtid_simple_recovery=TRUE` can always safely be used.

If any binary logs from MySQL 5.7.7 or older are present on the server (for example, following an upgrade of an older server to MySQL 8.0), with `binlog_gtid_simple_recovery=TRUE`, `gtid_executed` and `gtid_purged` might be initialized incorrectly in the following two situations:

- The newest binary log was generated by MySQL 5.7.5 or earlier, and `gtid_mode` was `ON` for some binary logs but `OFF` for the newest binary log.
- A `SET @@GLOBAL.gtid_purged` statement was issued on MySQL 5.7.7 or earlier, and the binary log that was active at the time of the `SET @@GLOBAL.gtid_purged` statement has not yet been purged.

If an incorrect GTID set is computed in either situation, it will remain incorrect even if the server is later restarted with `binlog_gtid_simple_recovery=FALSE`. If either of these situations apply or might apply on the server, set `binlog_gtid_simple_recovery=FALSE` before starting or restarting the server.

When `binlog_gtid_simple_recovery=FALSE` is set, the method of computing `gtid_executed` and `gtid_purged` as described in The `gtid_purged` System Variable is changed to iterate the binary log files as follows:

- Instead of using the value of `Previous_gtids_log_event` and GTID log events from the newest binary log file, the computation for `gtid_executed` iterates from the newest binary log file, and uses the value of `Previous_gtids_log_event` and any GTID log events from the first binary log file where it finds a `Previous_gtids_log_event` value. If the server's most recent binary log files do not have GTID log events, for example if `gtid_mode=ON` was used but the server was later changed to `gtid_mode=OFF`, this process can take a long time.
- Instead of using the value of `Previous_gtids_log_event` from the oldest binary log file, the computation for `gtid_purged` iterates from the oldest binary log file, and uses the value of `Previous_gtids_log_event` from the first binary log file where it finds either a nonempty `Previous_gtids_log_event` value, or at least one GTID log event (indicating that the use of GTIDs starts at that point). If the server's older binary log files do not have GTID log events, for example if `gtid_mode=ON` was only set recently on the server, this process can take a long time.

- `enforce_gtid_consistency`

Property	Value
Command-Line Format	`--enforce-gtid-consistency[=value]`
System Variable	`enforce_gtid_consistency`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Enumeration
Default Value	`OFF`

Valid Values	`OFF`
	`ON`
	`WARN`

Depending on the value of this variable, the server enforces GTID consistency by allowing execution of only statements that can be safely logged using a GTID. You *must* set this variable to `ON` before enabling GTID based replication.

The values that `enforce_gtid_consistency` can be configured to are:

- `OFF`: all transactions are allowed to violate GTID consistency.
- `ON`: no transaction is allowed to violate GTID consistency.
- `WARN`: all transactions are allowed to violate GTID consistency, but a warning is generated in this case.

Only statements that can be logged using GTID safe statements can be logged when `enforce_gtid_consistency` is set to `ON`, so the operations listed here cannot be used with this option:

- `CREATE TABLE ... SELECT` statements
- `CREATE TEMPORARY TABLE` or `DROP TEMPORARY TABLE` statements inside transactions
- Transactions or statements that update both transactional and nontransactional tables. There is an exception that nontransactional DML is allowed in the same transaction or in the same statement as transactional DML, if all *nontransactional* tables are temporary.

`--enforce-gtid-consistency` only takes effect if binary logging takes place for a statement. If binary logging is disabled on the server, or if statements are not written to the binary log because they are removed by a filter, GTID consistency is not checked or enforced for the statements that are not logged.

For more information, see Section 17.1.3.6, “Restrictions on Replication with GTIDs”.

Prior to MySQL 5.7 and in early releases in that release series, the boolean `enforce_gtid_consistency` defaulted to `OFF`. To maintain compatibility with these earlier releases, the enumeration defaults to `OFF`, and setting `--enforce-gtid-consistency` without a value is interpreted as setting the value to `ON`. The variable also has multiple textual aliases for the values: `0=OFF=FALSE`, `1=ON=TRUE`,`2=WARN`. This differs from other enumeration types but maintains compatibility with the boolean type used in previous releases. These changes impact on what is returned by the variable. Using `SELECT @@ENFORCE_GTID_CONSISTENCY`, `SHOW VARIABLES LIKE 'ENFORCE_GTID_CONSISTENCY'`, and `SELECT * FROM INFORMATION_SCHEMA.VARIABLES WHERE 'VARIABLE_NAME' = 'ENFORCE_GTID_CONSISTENCY'`, all return the textual form, not the numeric form. This is an incompatible change, since `@@ENFORCE_GTID_CONSISTENCY` returns the numeric form for booleans but returns the textual form for `SHOW` and the Information Schema.

- `gtid_executed`

Property	Value
System Variable	`gtid_executed`
System Variable	`gtid_executed`
Scope	Global
Scope	Global, Session
Dynamic	No
Dynamic	No
SET_VAR Hint Applies	No
SET_VAR Hint Applies	No
Type	String

When used with global scope, this variable contains a representation of the set of all transactions executed on the server and GTIDs that have been set by a `SET gtid_purged` statement. This is the same as the value of the `Executed_Gtid_Set` column in the output of `SHOW MASTER STATUS` and `SHOW SLAVE STATUS`. The value of this variable is a GTID set, see GTID Sets for more information.

When the server starts, `@@GLOBAL.gtid_executed` is initialized. See `binlog_gtid_simple_recovery` for more information on how binary logs are iterated to populate `gtid_executed`. GTIDs are then added to the set as transactions are executed, or if any `SET gtid_purged` statement is executed.

The set of transactions that can be found in the binary logs at any given time is equal to `GTID_SUBTRACT(@@GLOBAL.gtid_executed, @@GLOBAL.gtid_purged)`; that is, to all transactions in the binary log that have not yet been purged.

Issuing `RESET MASTER` causes the global value (but not the session value) of this variable to be reset to an empty string. GTIDs are not otherwise removed from this set other than when the set is cleared due to `RESET MASTER`.

In some older releases, this variable could also be used with session scope, where it contained a representation of the set of transactions that are written to the cache in the current session. The session scope is now deprecated.

- `gtid_executed_compression_period`

Property	Value
System Variable	`gtid_executed_compression_period`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1000`
Minimum Value	`0`
Maximum Value	`4294967295`

Compress the `mysql.gtid_executed` table each time this many transactions have been processed. A setting of 0 means that this table is not compressed. Since no compression of the table occurs when using the binary log, setting the value of the variable has no effect unless binary logging is disabled.

See mysql.gtid_executed Table Compression, for more information.

- `gtid_mode`

Property	Value
Command-Line Format	`--gtid-mode=MODE`
System Variable	`gtid_mode`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`OFF`

Valid Values	`OFF`
	`OFF_PERMISSIVE`
	`ON_PERMISSIVE`
	`ON`

Controls whether GTID based logging is enabled and what type of transactions the logs can contain. You must have privileges sufficient to set global system variables. See Section 5.1.9.1, “System Variable Privileges”. `enforce_gtid_consistency` must be true before you can set `gtid_mode=ON`. Before modifying this variable, see Section 17.1.5, “Changing Replication Modes on Online Servers”.

Logged transactions can be either anonymous or use GTIDs. Anonymous transactions rely on binary log file and position to identify specific transactions. GTID transactions have a unique identifier that is used to refer to transactions. The different modes are:

- `OFF`: Both new and replicated transactions must be anonymous.
- `OFF_PERMISSIVE`: New transactions are anonymous. Replicated transactions can be either anonymous or GTID transactions.
- `ON_PERMISSIVE`: New transactions are GTID transactions. Replicated transactions can be either anonymous or GTID transactions.
- `ON`: Both new and replicated transactions must be GTID transactions.

Changes from one value to another can only be one step at a time. For example, if `gtid_mode` is currently set to `OFF_PERMISSIVE`, it is possible to change to `OFF` or `ON_PERMISSIVE` but not to `ON`.

The values of `gtid_purged` and `gtid_executed` are persistent regardless of the value of `gtid_mode`. Therefore even after changing the value of `gtid_mode`, these variables contain the correct values.

- `gtid_next`

Property	Value
System Variable	`gtid_next`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`AUTOMATIC`

Valid Values	`AUTOMATIC`
	`ANONYMOUS`
	`UUID:NUMBER`

This variable is used to specify whether and how the next GTID is obtained.

Setting the session value of this system variable is a restricted operation. The session user must have either the `REPLICATION_APPLIER` privilege (see Section 17.3.3, “Replication Privilege Checks”), or privileges sufficient to set restricted session variables (see Section 5.1.9.1, “System Variable Privileges”).

`gtid_next` can take any of the following values:

- `AUTOMATIC`: Use the next automatically-generated global transaction ID.
- `ANONYMOUS`: Transactions do not have global identifiers, and are identified by file and position only.
- A global transaction ID in *`UUID`*:*`NUMBER`* format.

Exactly which of the above options are valid depends on the setting of `gtid_mode`, see Section 17.1.5.1, “Replication Mode Concepts” for more information. Setting this variable has no effect if `gtid_mode` is `OFF`.

After this variable has been set to *`UUID`*:*`NUMBER`*, and a transaction has been committed or rolled back, an explicit `SET GTID_NEXT` statement must again be issued before any other statement.

`DROP TABLE` or `DROP TEMPORARY TABLE` fails with an explicit error when used on a combination of nontemporary tables with temporary tables, or of temporary tables using transactional storage engines with temporary tables using nontransactional storage engines.

- `gtid_owned`

Property	Value
System Variable	`gtid_owned`
Scope	Global, Session
Dynamic	No
SET_VAR Hint Applies	No
Type	String

This read-only variable is primarily for internal use. Its contents depend on its scope.

- When used with global scope, `gtid_owned` holds a list of all the GTIDs that are currently in use on the server, with the IDs of the threads that own them. This variable is mainly useful for a multi-threaded replication slave to check whether a transaction is already being applied on another thread. An applier thread takes ownership of a transaction's GTID all the time it is processing the transaction, so `@@global.gtid_owned` shows the GTID and owner for the duration of processing. When a transaction has been committed (or rolled back), the applier thread releases ownership of the GTID.

- When used with session scope, `gtid_owned` holds a single GTID that is currently in use by and owned by this session. This variable is mainly useful for testing and debugging the use of GTIDs when the client has explicitly assigned a GTID for the transaction by setting `gtid_next`. In this case, `@@session.gtid_owned` displays the GTID all the time the client is processing the transaction, until the transaction has been committed (or rolled back). When the client has finished processing the transaction, the variable is cleared. If `gtid_next=AUTOMATIC` is used for the session, `gtid_owned` is only populated briefly during the execution of the commit statement for the transaction, so it cannot be observed from the session concerned, although it will be listed if `@@global.gtid_owned` is read at the right point. If you have a requirement to track the GTIDs that are handled by a client in a session, you can enable the session state tracker controlled by the `session_track_gtids` system variable.

- `gtid_purged`

Property	Value
System Variable	`gtid_purged`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The global value of the `gtid_purged` system variable (`@@GLOBAL.gtid_purged`) is a GTID set consisting of the GTIDs of all the transactions that have been committed on the server, but do not exist in any binary log file on the server. `gtid_purged` is a subset of `gtid_executed`. The following categories of GTIDs are in `gtid_purged`:

- GTIDs of replicated transactions that were committed with binary logging disabled on the slave.
- GTIDs of transactions that were written to a binary log file that has now been purged.
- GTIDs that were added explicitly to the set by the statement `SET @@GLOBAL.gtid_purged`.

When the server starts, the global value of `gtid_purged` is initialized to a set of GTIDs. For information on how this GTID set is computed, see The `gtid_purged` System Variable. If binary logs from MySQL 5.7.7 or older are present on the server, you might need to set `binlog_gtid_simple_recovery=FALSE` in the server's configuration file to produce the correct computation. See the description for `binlog_gtid_simple_recovery` for details of the situations in which this setting is needed.

Issuing `RESET MASTER` causes the value of `gtid_purged` to be reset to an empty string.

You can set the value of `gtid_purged` in order to record on the server that the transactions in a certain GTID set have been applied, although they do not exist in any binary log on the server. An example use case for this action is when you are restoring a backup of one or more databases on a server, but you do not have the relevant binary logs containing the transactions on the server.

From MySQL 8.0, there are two ways to set the value of `gtid_purged`. You can either replace the value of `gtid_purged` with your specified GTID set, or you can append your specified GTID set to the GTID set that is already held by `gtid_purged`. If the server has no existing GTIDs, for example an empty server that you are provisioning with a backup of an existing database, both methods have the same result. If you are restoring a backup that overlaps the transactions that are already on the server, for example replacing a corrupted table with a partial dump from the master made using `mysqldump` (which includes the GTIDs of all the transactions on the server, even though the dump is partial), use the first method of replacing the value of `gtid_purged`. If you are restoring a backup that is disjoint from the transactions that are already on the server, for example provisioning a multi-source replication slave using dumps from two different servers, use the second method of adding to the value of `gtid_purged`.

- To replace the value of `gtid_purged` with your specified GTID set, use the following statement:

  ```
  SET @@GLOBAL.gtid_purged = 'gtid_set'
  ```

  `gtid_set` must be a superset of the current value of `gtid_purged`, and must not intersect with `gtid_subtract(gtid_executed,gtid_purged)`. In other words, the new GTID set **must** include any GTIDs that were already in `gtid_purged`, and **must not** include any GTIDs in `gtid_executed` that have not yet been purged. `gtid_set` also cannot include any GTIDs that are in `@@global.gtid_owned`, that is, the GTIDs for transactions that are currently being processed on the server.

  The result is that the global value of `gtid_purged` is set equal to `gtid_set`, and the value of `gtid_executed` becomes the union of `gtid_set` and the previous value of `gtid_executed`.

- To append your specified GTID set to `gtid_purged`, use the following statement with a plus sign (+) before the GTID set:

  ```
  SET @@GLOBAL.gtid_purged = '+gtid_set'
  ```

  `gtid_set` **must not** intersect with the current value of `gtid_executed`. In other words, the new GTID set must not include any GTIDs in `gtid_executed`, including transactions that are already also in `gtid_purged`. `gtid_set` also cannot include any GTIDs that are in `@@global.gtid_owned`, that is, the GTIDs for transactions that are currently being processed on the server.

  The result is that `gtid_set` is added to both `gtid_executed` and `gtid_purged`.

Note

If any binary logs from MySQL 5.7.7 or older are present on the server (for example, following an upgrade of an older server to MySQL 8.0), after issuing a `SET @@GLOBAL.gtid_purged` statement, you might need to set `binlog_gtid_simple_recovery=FALSE` in the server's configuration file before restarting the server, otherwise `gtid_purged` can be computed incorrectly. See the description for `binlog_gtid_simple_recovery` for details of the situations in which this setting is needed.

17.1.7 Common Replication Administration Tasks

Once replication has been started it executes without requiring much regular administration. This section describes how to check the status of replication and how to pause a slave.

17.1.7.1 Checking Replication Status

The most common task when managing a replication process is to ensure that replication is taking place and that there have been no errors between the slave and the master.

The `SHOW SLAVE STATUS` statement, which you must execute on each slave, provides information about the configuration and status of the connection between the slave server and the master server. From MySQL 5.7, the Performance Schema has replication tables that provide this information in a more accessible form. See Section 26.12.11, “Performance Schema Replication Tables”.

The `SHOW STATUS` statement also provided some information relating specifically to replication slaves. From MySQL 5.7, the following status variables previously monitored using `SHOW STATUS` were deprecated and moved to the Performance Schema replication tables:

- `Slave_retried_transactions`
- `Slave_last_heartbeat`
- `Slave_received_heartbeats`
- `Slave_heartbeat_period`
- `Slave_running`

The replication heartbeat information shown in the Performance Schema replication tables lets you check that the replication connection is active even if the master has not sent events to the slave recently. The master sends a heartbeat signal to a slave if there are no updates to, and no unsent events in, the binary log for a longer period than the heartbeat interval. The `MASTER_HEARTBEAT_PERIOD` setting on the master (set by the `CHANGE MASTER TO` statement) specifies the frequency of the heartbeat, which defaults to half of the connection timeout interval for the slave (`slave_net_timeout`). The `replication_connection_status` Performance Schema table shows when the most recent heartbeat signal was received by a replication slave, and how many heartbeat signals it has received.

If you are using the `SHOW SLAVE STATUS` statement to check on the status of an individual slave, the statement provides the following information:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: master1
                  Master_User: root
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000004
          Read_Master_Log_Pos: 931
               Relay_Log_File: slave1-relay-bin.000056
                Relay_Log_Pos: 950
        Relay_Master_Log_File: mysql-bin.000004
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 931
              Relay_Log_Space: 1365
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
```

```
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids: 0
```

The key fields from the status report to examine are:

- `Slave_IO_State`: The current status of the slave. See Section 8.14.4, “Replication Slave I/O Thread States”, and Section 8.14.5, “Replication Slave SQL Thread States”, for more information.
- `Slave_IO_Running`: Whether the I/O thread for reading the master's binary log is running. Normally, you want this to be `Yes` unless you have not yet started replication or have explicitly stopped it with `STOP SLAVE`.
- `Slave_SQL_Running`: Whether the SQL thread for executing events in the relay log is running. As with the I/O thread, this should normally be `Yes`.
- `Last_IO_Error`, `Last_SQL_Error`: The last errors registered by the I/O and SQL threads when processing the relay log. Ideally these should be blank, indicating no errors.
- `Seconds_Behind_Master`: The number of seconds that the slave SQL thread is behind processing the master binary log. A high number (or an increasing one) can indicate that the slave is unable to handle events from the master in a timely fashion.

  A value of 0 for `Seconds_Behind_Master` can usually be interpreted as meaning that the slave has caught up with the master, but there are some cases where this is not strictly true. For example, this can occur if the network connection between master and slave is broken but the slave I/O thread has not yet noticed this—that is, `slave_net_timeout` has not yet elapsed.

  It is also possible that transient values for `Seconds_Behind_Master` may not reflect the situation accurately. When the slave SQL thread has caught up on I/O, `Seconds_Behind_Master` displays 0; but when the slave I/O thread is still queuing up a new event, `Seconds_Behind_Master` may show a large value until the SQL thread finishes executing the new event. This is especially likely when the events have old timestamps; in such cases, if you execute `SHOW SLAVE STATUS` several times in a relatively short period, you may see this value change back and forth repeatedly between 0 and a relatively large value.

Several pairs of fields provide information about the progress of the slave in reading events from the master binary log and processing them in the relay log:

- (`Master_Log_file`, `Read_Master_Log_Pos`): Coordinates in the master binary log indicating how far the slave I/O thread has read events from that log.
- (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`): Coordinates in the master binary log indicating how far the slave SQL thread has executed events received from that log.
- (`Relay_Log_File`, `Relay_Log_Pos`): Coordinates in the slave relay log indicating how far the slave SQL thread has executed the relay log. These correspond to the preceding coordinates, but are expressed in slave relay log coordinates rather than master binary log coordinates.

On the master, you can check the status of connected slaves using `SHOW PROCESSLIST` to examine the list of running processes. Slave connections have `Binlog Dump` in the `Command` field:

```
mysql> SHOW PROCESSLIST \G;
*************************** 4. row ***************************
     Id: 10
   User: root
   Host: slave1:58371
     db: NULL
Command: Binlog Dump
   Time: 777
  State: Has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL
```

Because it is the slave that drives the replication process, very little information is available in this report.

For slaves that were started with the `--report-host` option and are connected to the master, the `SHOW SLAVE HOSTS` statement on the master shows basic information about the slaves. The output includes the ID of the slave server, the value of the `--report-host` option, the connecting port, and master ID:

```
mysql> SHOW SLAVE HOSTS;
+-----------+--------+------+------------------+-----------+
| Server_id | Host   | Port | Rpl_recovery_rank | Master_id |
+-----------+--------+------+------------------+-----------+
|        10 | slave1 | 3306 |                0 |         1 |
+-----------+--------+------+------------------+-----------+
1 row in set (0.00 sec)
```

17.1.7.2 Pausing Replication on the Slave

You can stop and start replication on the slave using the `STOP SLAVE` and `START SLAVE` statements.

To stop processing of the binary log from the master, use `STOP SLAVE`:

```
mysql> STOP SLAVE;
```

When replication is stopped, the slave I/O thread stops reading events from the master binary log and writing them to the relay log, and the SQL thread stops reading events from the relay log and executing them. You can pause the I/O or SQL thread individually by specifying the thread type:

```
mysql> STOP SLAVE IO_THREAD;
mysql> STOP SLAVE SQL_THREAD;
```

To start execution again, use the `START SLAVE` statement:

```
mysql> START SLAVE;
```

To start a particular thread, specify the thread type:

```
mysql> START SLAVE IO_THREAD;
mysql> START SLAVE SQL_THREAD;
```

For a slave that performs updates only by processing events from the master, stopping only the SQL thread can be useful if you want to perform a backup or other task. The I/O thread will continue to read events from the master but they are not executed. This makes it easier for the slave to catch up when you restart the SQL thread.

Stopping only the I/O thread enables the events in the relay log to be executed by the SQL thread up to the point where the relay log ends. This can be useful when you want to pause execution to catch up with events already received from the master, when you want to perform administration on the slave but also

ensure that it has processed all updates to a specific point. This method can also be used to pause event receipt on the slave while you conduct administration on the master. Stopping the I/O thread but permitting the SQL thread to run helps ensure that there is not a massive backlog of events to be executed when replication is started again.

17.2 Replication Implementation

Replication is based on the master server keeping track of all changes to its databases (updates, deletes, and so on) in its binary log. The binary log serves as a written record of all events that modify database structure or content (data) from the moment the server was started. Typically, `SELECT` statements are not recorded because they modify neither database structure nor content.

Each slave that connects to the master requests a copy of the binary log. That is, it pulls the data from the master, rather than the master pushing the data to the slave. The slave also executes the events from the binary log that it receives. This has the effect of repeating the original changes just as they were made on the master. Tables are created or their structure modified, and data is inserted, deleted, and updated according to the changes that were originally made on the master.

Because each slave is independent, the replaying of the changes from the master's binary log occurs independently on each slave that is connected to the master. In addition, because each slave receives a copy of the binary log only by requesting it from the master, the slave is able to read and update the copy of the database at its own pace and can start and stop the replication process at will without affecting the ability to update to the latest database status on either the master or slave side.

For more information on the specifics of the replication implementation, see Section 17.2.2, “Replication Implementation Details”.

Masters and slaves report their status in respect of the replication process regularly so that you can monitor them. See Section 8.14, “Examining Thread Information”, for descriptions of all replicated-related states.

The master binary log is written to a local relay log on the slave before it is processed. The slave also records information about the current position with the master's binary log and the local relay log. See Section 17.2.4, “Replication Relay and Status Logs”.

Database changes are filtered on the slave according to a set of rules that are applied according to the various configuration options and variables that control event evaluation. For details on how these rules are applied, see Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”.

17.2.1 Replication Formats

Replication works because events written to the binary log are read from the master and then processed on the slave. The events are recorded within the binary log in different formats according to the type of event. The different replication formats used correspond to the binary logging format used when the events were recorded in the master's binary log. The correlation between binary logging formats and the terms used during replication are:

- When using statement-based binary logging, the master writes SQL statements to the binary log. Replication of the master to the slave works by executing the SQL statements on the slave. This is called *statement-based replication* (which can be abbreviated as *SBR*), which corresponds to the MySQL statement-based binary logging format.
- When using row-based logging, the master writes *events* to the binary log that indicate how individual table rows are changed. Replication of the master to the slave works by copying the events representing the changes to the table rows to the slave. This is called *row-based replication* (which can be abbreviated as *RBR*).

Row-based logging is the default method.

- You can also configure MySQL to use a mix of both statement-based and row-based logging, depending on which is most appropriate for the change to be logged. This is called *mixed-format logging*. When using mixed-format logging, a statement-based log is used by default. Depending on certain statements, and also the storage engine being used, the log is automatically switched to row-based in particular cases. Replication using the mixed format is referred to as *mixed-based replication* or *mixed-format replication*. For more information, see Section 5.4.4.3, “Mixed Binary Logging Format”.

NDB Cluster. The default binary logging format in MySQL NDB Cluster 8.0 is `MIXED`. You should note that NDB Cluster Replication always uses row-based replication, and that the `NDB` storage engine is incompatible with statement-based replication. See Section 22.6.2, “General Requirements for NDB Cluster Replication”, for more information.

When using `MIXED` format, the binary logging format is determined in part by the storage engine being used and the statement being executed. For more information on mixed-format logging and the rules governing the support of different logging formats, see Section 5.4.4.3, “Mixed Binary Logging Format”.

The logging format in a running MySQL server is controlled by setting the `binlog_format` server system variable. This variable can be set with session or global scope. The rules governing when and how the new setting takes effect are the same as for other MySQL server system variables. Setting the variable for the current session lasts only until the end of that session, and the change is not visible to other sessions. Setting the variable globally takes effect for clients that connect after the change, but not for any current client sessions, including the session where the variable setting was changed. To make the global system variable setting permanent so that it applies across server restarts, you must set it in an option file. For more information, see Section 13.7.6.1, “SET Syntax for Variable Assignment”.

There are conditions under which you cannot change the binary logging format at runtime or doing so causes replication to fail. See Section 5.4.4.2, “Setting The Binary Log Format”.

Changing the global `binlog_format` value requires privileges sufficient to set global system variables. Changing the session `binlog_format` value requires privileges sufficient to set restricted session system variables. See Section 5.1.9.1, “System Variable Privileges”.

The statement-based and row-based replication formats have different issues and limitations. For a comparison of their relative advantages and disadvantages, see Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

With statement-based replication, you may encounter issues with replicating stored routines or triggers. You can avoid these issues by using row-based replication instead. For more information, see Section 24.7, “Stored Program Binary Logging”.

17.2.1.1 Advantages and Disadvantages of Statement-Based and Row-Based Replication

Each binary logging format has advantages and disadvantages. For most users, the mixed replication format should provide the best combination of data integrity and performance. If, however, you want to take advantage of the features specific to the statement-based or row-based replication format when performing certain tasks, you can use the information in this section, which provides a summary of their relative advantages and disadvantages, to determine which is best for your needs.

- Advantages of statement-based replication
- Disadvantages of statement-based replication
- Advantages of row-based replication
- Disadvantages of row-based replication

Advantages of statement-based replication

- Proven technology.
- Less data written to log files. When updates or deletes affect many rows, this results in *much* less storage space required for log files. This also means that taking and restoring from backups can be accomplished more quickly.
- Log files contain all statements that made any changes, so they can be used to audit the database.

Disadvantages of statement-based replication

- **Statements that are unsafe for SBR.**
  Not all statements which modify data (such as `INSERT DELETE`, `UPDATE`, and `REPLACE` statements) can be replicated using statement-based replication. Any nondeterministic behavior is difficult to replicate when using statement-based replication. Examples of such Data Modification Language (DML) statements include the following:
  - A statement that depends on a UDF or stored program that is nondeterministic, since the value returned by such a UDF or stored program or depends on factors other than the parameters supplied to it. (Row-based replication, however, simply replicates the value returned by the UDF or stored program, so its effect on table rows and data is the same on both the master and slave.) See Section 17.5.1.16, “Replication of Invoked Features”, for more information.
  - `DELETE` and `UPDATE` statements that use a `LIMIT` clause without an `ORDER BY` are nondeterministic. See Section 17.5.1.18, “Replication and LIMIT”.
  - Locking read statements (`SELECT ... FOR UPDATE` and `SELECT ... FOR SHARE`) that use `NOWAIT` or `SKIP LOCKED` options. See Locking Read Concurrency with NOWAIT and SKIP LOCKED.
  - Deterministic UDFs must be applied on the slaves.
  - Statements using any of the following functions cannot be replicated properly using statement-based replication:
    - `LOAD_FILE()`
    - `UUID()`, `UUID_SHORT()`
    - `USER()`
    - `FOUND_ROWS()`
    - `SYSDATE()` (unless both the master and the slave are started with the `--sysdate-is-now` option)
    - `GET_LOCK()`
    - `IS_FREE_LOCK()`
    - `IS_USED_LOCK()`
    - `MASTER_POS_WAIT()`
    - `RAND()`
    - `RELEASE_LOCK()`
    - `SLEEP()`

- `VERSION()`

  However, all other functions are replicated correctly using statement-based replication, including `NOW()` and so forth.

  For more information, see Section 17.5.1.14, “Replication and System Functions”.

Statements that cannot be replicated correctly using statement-based replication are logged with a warning like the one shown here:

```
[Warning] Statement is not safe to log in statement format.
```

A similar warning is also issued to the client in such cases. The client can display it using `SHOW WARNINGS`.

- `INSERT ... SELECT` requires a greater number of row-level locks than with row-based replication.
- `UPDATE` statements that require a table scan (because no index is used in the `WHERE` clause) must lock a greater number of rows than with row-based replication.
- For `InnoDB`: An `INSERT` statement that uses `AUTO_INCREMENT` blocks other nonconflicting `INSERT` statements.
- For complex statements, the statement must be evaluated and executed on the slave before the rows are updated or inserted. With row-based replication, the slave only has to modify the affected rows, not execute the full statement.
- If there is an error in evaluation on the slave, particularly when executing complex statements, statement-based replication may slowly increase the margin of error across the affected rows over time. See Section 17.5.1.28, “Slave Errors During Replication”.
- Stored functions execute with the same `NOW()` value as the calling statement. However, this is not true of stored procedures.
- Deterministic UDFs must be applied on the slaves.
- Table definitions must be (nearly) identical on master and slave. See Section 17.5.1.9, “Replication with Differing Table Definitions on Master and Slave”, for more information.

Advantages of row-based replication

- All changes can be replicated. This is the safest form of replication.

Note

Statements that update the information in the `mysql` system schema—such as `GRANT`, `REVOKE` and the manipulation of triggers, stored routines (including stored procedures), and views—are all replicated to slaves using statement-based replication.

For statements such as `CREATE TABLE ... SELECT`, a `CREATE` statement is generated from the table definition and replicated using statement-based format, while the row insertions are replicated using row-based format.

- Fewer row locks are required on the master, which thus achieves higher concurrency, for the following types of statements:

- `INSERT ... SELECT`
- `INSERT` statements with `AUTO_INCREMENT`
- `UPDATE` or `DELETE` statements with `WHERE` clauses that do not use keys or do not change most of the examined rows.

- Fewer row locks are required on the slave for any `INSERT`, `UPDATE`, or `DELETE` statement.

Disadvantages of row-based replication

- RBR can generate more data that must be logged. To replicate a DML statement (such as an `UPDATE` or `DELETE` statement), statement-based replication writes only the statement to the binary log. By contrast, row-based replication writes each changed row to the binary log. If the statement changes many rows, row-based replication may write significantly more data to the binary log; this is true even for statements that are rolled back. This also means that making and restoring a backup can require more time. In addition, the binary log is locked for a longer time to write the data, which may cause concurrency problems. Use `binlog_row_image=minimal` to reduce the disadvantage considerably.
- Deterministic UDFs that generate large `BLOB` values take longer to replicate with row-based replication than with statement-based replication. This is because the `BLOB` column value is logged, rather than the statement generating the data.
- You cannot see on the slave what statements were received from the master and executed. However, you can see what data was changed using `mysqlbinlog` with the options `--base64-output=DECODE-ROWS` and `--verbose`.

  Alternatively, use the `binlog_rows_query_log_events` variable, which if enabled adds a `Rows_query` event with the statement to `mysqlbinlog` output when the `-vv` option is used.
- For tables using the `MyISAM` storage engine, a stronger lock is required on the slave for `INSERT` statements when applying them as row-based events to the binary log than when applying them as statements. This means that concurrent inserts on `MyISAM` tables are not supported when using row-based replication.

17.2.1.2 Usage of Row-Based Logging and Replication

MySQL uses statement-based logging (SBL), row-based logging (RBL) or mixed-format logging. The type of binary log used impacts the size and efficiency of logging. Therefore the choice between row-based replication (RBR) or statement-based replication (SBR) depends on your application and environment. This section describes known issues when using a row-based format log, and describes some best practices using it in replication.

For additional information, see Section 17.2.1, “Replication Formats”, and Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

For information about issues specific to NDB Cluster Replication (which depends on row-based replication), see Section 22.6.3, “Known Issues in NDB Cluster Replication”.

- **Row-based logging of temporary tables.** As noted in Section 17.5.1.30, “Replication and Temporary Tables”, temporary tables are not replicated when using row-based format or (from MySQL 8.0.4) mixed format. For more information, see Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

  Temporary tables are not replicated when using row-based or mixed format because there is no need. In addition, because temporary tables can be read only from the thread which created them, there is seldom if ever any benefit obtained from replicating them, even when using statement-based format.

You can switch from statement-based to row-based binary logging format at runtime even when temporary tables have been created. However, in MySQL 8.0, you cannot switch from row-based or mixed format for binary logging to statement-based format at runtime, because any `CREATE TEMPORARY TABLE` statements will have been omitted from the binary log in the previous mode.

The MySQL server tracks the logging mode that was in effect when each temporary table was created. When a given client session ends, the server logs a `DROP TEMPORARY TABLE IF EXISTS` statement for each temporary table that still exists and was created when statement-based binary logging was in use. If row-based or mixed format binary logging was in use when the table was created, the `DROP TEMPORARY TABLE IF EXISTS` statement is not logged. In releases before MySQL 8.0.4 and 5.7.25, the `DROP TEMPORARY TABLE IF EXISTS` statement was logged regardless of the logging mode that was in effect.

Nontransactional DML statements involving temporary tables are allowed when using `binlog_format=ROW`, as long as any nontransactional tables affected by the statements are temporary tables (Bug #14272672).

- **RBL and synchronization of nontransactional tables.** When many rows are affected, the set of changes is split into several events; when the statement commits, all of these events are written to the binary log. When executing on the slave, a table lock is taken on all tables involved, and then the rows are applied in batch mode. Depending on the engine used for the slave's copy of the table, this may or may not be effective.

- **Latency and binary log size.** RBL writes changes for each row to the binary log and so its size can increase quite rapidly. This can significantly increase the time required to make changes on the slave that match those on the master. You should be aware of the potential for this delay in your applications.

- **Reading the binary log.** `mysqlbinlog` displays row-based events in the binary log using the `BINLOG` statement (see Section 13.7.8.1, “BINLOG Syntax”). This statement displays an event as a base 64-encoded string, the meaning of which is not evident. When invoked with the `--base64-output=DECODE-ROWS` and `--verbose` options, `mysqlbinlog` formats the contents of the binary log to be human readable. When binary log events were written in row-based format and you want to read or recover from a replication or database failure you can use this command to read contents of the binary log. For more information, see Section 4.6.8.2, “mysqlbinlog Row Event Display”.

- **Binary log execution errors and slave_exec_mode.** Using `slave_exec_mode=IDEMPOTENT` is generally only useful with MySQL NDB Cluster replication, for which `IDEMPOTENT` is the default value. (See Section 22.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”). When `slave_exec_mode` is `IDEMPOTENT`, a failure to apply changes from RBL because the original row cannot be found does not trigger an error or cause replication to fail. This means that it is possible that updates are not applied on the slave, so that the master and slave are no longer synchronized. Latency issues and use of nontransactional tables with RBR when `slave_exec_mode` is `IDEMPOTENT` can cause the master and slave to diverge even further. For more information about `slave_exec_mode`, see Section 5.1.8, “Server System Variables”.

  For other scenarios, setting `slave_exec_mode` to `STRICT` is normally sufficient; this is the default value for storage engines other than `NDB`.

- **Filtering based on server ID not supported.** You can filter based on server ID by using the `IGNORE_SERVER_IDS` option for the `CHANGE MASTER TO` statement. This option works with statement-based and row-based logging formats, but is deprecated for use when `GTID_MODE=ON` is set. Another method to filter out changes on some slaves is to use a `WHERE` clause that includes the relation `@@server_id <> id_value` clause with `UPDATE` and `DELETE` statements. For example, `WHERE @@server_id <> 1`. However, this does not work correctly with row-based logging. To use the `server_id` system variable for statement filtering, use statement-based logging.

- **Database-level replication options.** The effects of the `--replicate-do-db`, `--replicate-ignore-db`, and `--replicate-rewrite-db` options differ considerably depending on whether row-based or statement-based logging is used. Therefore, it is recommended to avoid database-level options and instead use table-level options such as `--replicate-do-table` and `--replicate-ignore-table`. For more information about these options and the impact replication format has on how they operate, see Section 17.1.6, “Replication and Binary Logging Options and Variables”.

- **RBL, nontransactional tables, and stopped slaves.** When using row-based logging, if the slave server is stopped while a slave thread is updating a nontransactional table, the slave database can reach an inconsistent state. For this reason, it is recommended that you use a transactional storage engine such as `InnoDB` for all tables replicated using the row-based format. Use of `STOP SLAVE` or `STOP SLAVE SQL_THREAD` prior to shutting down the slave MySQL server helps prevent issues from occurring, and is always recommended regardless of the logging format or storage engine you use.

17.2.1.3 Determination of Safe and Unsafe Statements in Binary Logging

The “safeness” of a statement in MySQL replication refers to whether the statement and its effects can be replicated correctly using statement-based format. If this is true of the statement, we refer to the statement as *safe*; otherwise, we refer to it as *unsafe*.

In general, a statement is safe if it deterministic, and unsafe if it is not. However, certain nondeterministic functions are *not* considered unsafe (see Nondeterministic functions not considered unsafe, later in this section). In addition, statements using results from floating-point math functions—which are hardware-dependent—are always considered unsafe (see Section 17.5.1.12, “Replication and Floating-Point Values”).

Handling of safe and unsafe statements. A statement is treated differently depending on whether the statement is considered safe, and with respect to the binary logging format (that is, the current value of `binlog_format`).

- When using row-based logging, no distinction is made in the treatment of safe and unsafe statements.

- When using mixed-format logging, statements flagged as unsafe are logged using the row-based format; statements regarded as safe are logged using the statement-based format.

- When using statement-based logging, statements flagged as being unsafe generate a warning to this effect. Safe statements are logged normally.

Each statement flagged as unsafe generates a warning. If a large number of such statements were executed on the master, this could lead to excessively large error log files. To prevent this, MySQL has a warning suppression mechanism. Whenever the 50 most recent `ER_BINLOG_UNSAFE_STATEMENT` warnings have been generated more than 50 times in any 50-second period, warning suppression is enabled. When activated, this causes such warnings not to be written to the error log; instead, for each 50 warnings of this type, a note `The last warning was repeated N times in last S seconds` is written to the error log. This continues as long as the 50 most recent such warnings were issued in 50 seconds or less; once the rate has decreased below this threshold, the warnings are once again logged normally. Warning suppression has no effect on how the safety of statements for statement-based logging is determined, nor on how warnings are sent to the client. MySQL clients still receive one warning for each such statement.

For more information, see Section 17.2.1, “Replication Formats”.

Statements considered unsafe.
Statements with the following characteristics are considered unsafe:

- **Statements containing system functions that may return a different value on the slave.** These functions include `FOUND_ROWS()`, `GET_LOCK()`, `IS_FREE_LOCK()`, `IS_USED_LOCK()`,

`LOAD_FILE()`, `MASTER_POS_WAIT()`, `RAND()`, `RELEASE_LOCK()`, `ROW_COUNT()`, `SESSION_USER()`, `SLEEP()`, `SYSDATE()`, `SYSTEM_USER()`, `USER()`, `UUID()`, and `UUID_SHORT()`.

Nondeterministic functions not considered unsafe. Although these functions are not deterministic, they are treated as safe for purposes of logging and replication: `CONNECTION_ID()`, `CURDATE()`, `CURRENT_DATE()`, `CURRENT_TIME()`, `CURRENT_TIMESTAMP()`, `CURTIME()`,, `LAST_INSERT_ID()`, `LOCALTIME()`, `LOCALTIMESTAMP()`, `NOW()`, `UNIX_TIMESTAMP()`, `UTC_DATE()`, `UTC_TIME()`, and `UTC_TIMESTAMP()`.

For more information, see Section 17.5.1.14, “Replication and System Functions”.

- **References to system variables.** Most system variables are not replicated correctly using the statement-based format. See Section 17.5.1.38, “Replication and Variables”. For exceptions, see Section 5.4.4.3, “Mixed Binary Logging Format”.
- **UDFs.** Since we have no control over what a UDF does, we must assume that it is executing unsafe statements.
- **Fulltext plugin.** This plugin may behave differently on different MySQL servers; therefore, statements depending on it could have different results. For this reason, all statements relying on the fulltext plugin are treated as unsafe in MySQL.
- **Trigger or stored program updates a table having an AUTO_INCREMENT column.** This is unsafe because the order in which the rows are updated may differ on the master and the slave.

  In addition, an `INSERT` into a table that has a composite primary key containing an `AUTO_INCREMENT` column that is not the first column of this composite key is unsafe.

  For more information, see Section 17.5.1.1, “Replication and AUTO_INCREMENT”.
- **INSERT ... ON DUPLICATE KEY UPDATE statements on tables with multiple primary or unique keys.** When executed against a table that contains more than one primary or unique key, this statement is considered unsafe, being sensitive to the order in which the storage engine checks the keys, which is not deterministic, and on which the choice of rows updated by the MySQL Server depends.

  An `INSERT ... ON DUPLICATE KEY UPDATE` statement against a table having more than one unique or primary key is marked as unsafe for statement-based replication. (Bug #11765650, Bug #58637)
- **Updates using LIMIT.** The order in which rows are retrieved is not specified, and is therefore considered unsafe. See Section 17.5.1.18, “Replication and LIMIT”.
- **Accesses or references log tables.** The contents of the system log table may differ between master and slave.
- **Nontransactional operations after transactional operations.** Within a transaction, allowing any nontransactional reads or writes to execute after any transactional reads or writes is considered unsafe.

  For more information, see Section 17.5.1.34, “Replication and Transactions”.
- **Accesses or references self-logging tables.** All reads and writes to self-logging tables are considered unsafe. Within a transaction, any statement following a read or write to self-logging tables is also considered unsafe.
- **LOAD DATA statements.** `LOAD DATA` is treated as unsafe and when `binlog_format=MIXED` the statement is logged in row-based format. When `binlog_format=STATEMENT` `LOAD DATA` does not generate a warning, unlike other unsafe statements.

- **XA transactions.** If two XA transactions committed in parallel on the master are being prepared on the slave in the inverse order, locking dependencies can occur with statement-based replication that cannot be safely resolved, and it is possible for replication to fail with deadlock on the slave. When `binlog_format=STATEMENT` is set, DML statements inside XA transactions are flagged as being unsafe and generate a warning. When `binlog_format=MIXED` or `binlog_format=ROW` is set, DML statements inside XA transactions are logged using row-based replication, and the potential issue is not present.

- **`DEFAULT` clause that refers to a nondeterministic function.** If an expression default value refers to a nondeterministic function, any statement that causes the expression to be evaluated is unsafe for statement-based replication. This includes statements such as `INSERT`, `UPDATE`, and `ALTER TABLE`. Unlike most other unsafe statements, this category of statement cannot be replicated safely in row-based format. When `binlog_format` is set to `STATEMENT`, the statement is logged and executed but a warning message is written to the error log. When `binlog_format` is set to `MIXED` or `ROW`, the statement is not executed and an error message is written to the error log. For more information on the handling of explicit defaults, see Handling of Explicit Defaults as of MySQL 8.0.13.

For additional information, see Section 17.5.1, “Replication Features and Issues”.

17.2.2 Replication Implementation Details

MySQL replication capabilities are implemented using three threads, one on the master server and two on the slave:

- **Binlog dump thread.** The master creates a thread to send the binary log contents to a slave when the slave connects. This thread can be identified in the output of `SHOW PROCESSLIST` on the master as the `Binlog Dump` thread.

  The binary log dump thread acquires a lock on the master's binary log for reading each event that is to be sent to the slave. As soon as the event has been read, the lock is released, even before the event is sent to the slave.

- **Slave I/O thread.** When a `START SLAVE` statement is issued on a slave server, the slave creates an I/O thread, which connects to the master and asks it to send the updates recorded in its binary logs.

  The slave I/O thread reads the updates that the master's `Binlog Dump` thread sends (see previous item) and copies them to local files that comprise the slave's relay log.

  The state of this thread is shown as `Slave_IO_running` in the output of `SHOW SLAVE STATUS` or as `Slave_running` in the output of `SHOW STATUS`.

- **Slave SQL thread.** The slave creates an SQL thread to read the relay log that is written by the slave I/O thread and execute the events contained therein.

In the preceding description, there are three threads per master/slave connection. A master that has multiple slaves creates one binary log dump thread for each currently connected slave, and each slave has its own I/O and SQL threads.

A slave uses two threads to separate reading updates from the master and executing them into independent tasks. Thus, the task of reading statements is not slowed down if statement execution is slow. For example, if the slave server has not been running for a while, its I/O thread can quickly fetch all the binary log contents from the master when the slave starts, even if the SQL thread lags far behind. If the slave stops before the SQL thread has executed all the fetched statements, the I/O thread has at least fetched everything so that a safe copy of the statements is stored locally in the slave's relay logs, ready for execution the next time that the slave starts.

The `SHOW PROCESSLIST` statement provides information that tells you what is happening on the master and on the slave regarding replication. For information on master states, see Section 8.14.3, “Replication Master Thread States”. For slave states, see Section 8.14.4, “Replication Slave I/O Thread States”, and Section 8.14.5, “Replication Slave SQL Thread States”.

The following example illustrates how the three threads show up in the output from `SHOW PROCESSLIST`.

On the master server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 2
   User: root
   Host: localhost:32931
     db: NULL
Command: Binlog Dump
   Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
         be updated
   Info: NULL
```

Here, thread 2 is a `Binlog Dump` replication thread that services a connected slave. The `State` information indicates that all outstanding updates have been sent to the slave and that the master is waiting for more updates to occur. If you see no `Binlog Dump` threads on a master server, this means that replication is not running; that is, no slaves are currently connected.

On a slave server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 10
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 11
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Has read all relay log; waiting for the slave I/O
         thread to update it
   Info: NULL
```

The `State` information indicates that thread 10 is the I/O thread that is communicating with the master server, and thread 11 is the SQL thread that is processing the updates stored in the relay logs. At the time that `SHOW PROCESSLIST` was run, both threads were idle, waiting for further updates.

The value in the `Time` column can show how late the slave is compared to the master. See Section A.14, “MySQL 8.0 FAQ: Replication”. If sufficient time elapses on the master side without activity on the `Binlog Dump` thread, the master determines that the slave is no longer connected. As for any other client connection, the timeouts for this depend on the values of `net_write_timeout` and `net_retry_count`; for more information about these, see Section 5.1.8, “Server System Variables”.

The `SHOW SLAVE STATUS` statement provides additional information about replication processing on a slave server. See Section 17.1.7.1, “Checking Replication Status”.

17.2.3 Replication Channels

Replication channels represent the path of transactions flowing from a master to a slave. This section describes how channels can be used in a replication topology, and the impact they have on single-source replication.

To provide compatibility with previous versions, the MySQL server automatically creates on startup a default channel whose name is the empty string (`""`). This channel is always present; it cannot be created or destroyed by the user. If no other channels (having nonempty names) have been created, replication statements act on the default channel only, so that all replication statements from older slaves function as expected (see Section 17.2.3.2, “Compatibility with Previous Replication Statements”. Statements applying to replication channels as described in this section can be used only when there is at least one named channel.

A replication channel encompasses the path of transactions transmitted from a master to a slave. In multi-source replication a slave opens multiple channels, one per master, and each channel has its own relay log and applier (SQL) threads. Once transactions are received by a replication channel's receiver (I/O) thread, they are added to the channel's relay log file and passed through to an applier thread. This enables channels to function independently.

A replication channel is also associated with a host name and port. You can assign multiple channels to the same combination of host name and port. In MySQL 8.0, the maximum number of channels that can be added to one slave in a multi-source replication topology is 256. Each replication channel must have a unique (nonempty) name (see Section 17.2.3.4, “Replication Channel Naming Conventions”). Channels can be configured independently.

17.2.3.1 Commands for Operations on a Single Channel

To enable MySQL replication operations to act on individual replication channels, use the `FOR CHANNEL` *`channel`* clause with the following replication statements:

- `CHANGE MASTER TO`
- `START SLAVE`
- `STOP SLAVE`
- `SHOW RELAYLOG EVENTS`
- `FLUSH RELAY LOGS`
- `SHOW SLAVE STATUS`
- `RESET SLAVE`

An additional `channel` parameter is introduced for the following function:

- `MASTER_POS_WAIT()`

The following statements are disallowed for the `group_replication_recovery` channel:

- `START SLAVE`
- `STOP SLAVE`

The following statements are disallowed for the `group_replication_applier` channel:

- `START SLAVE`
- `STOP SLAVE`
- `SHOW SLAVE STATUS`

`FLUSH RELAY LOGS` is now permitted for the `group_replication_applier` channel, but if the request is received while a transaction is being applied, the request is performed after the transaction ends. The requester must wait while the transaction is completed and the rotation takes place. This behavior prevents transactions from being split, which is not permitted for Group Replication.

17.2.3.2 Compatibility with Previous Replication Statements

When a replication slave has multiple channels and a `FOR CHANNEL channel` option is not specified, a valid statement generally acts on all available channels, with some specific exceptions.

For example, the following statements behave as expected for all except certain Group Replication channels:

- `START SLAVE` starts replication threads for all channels, except the `group_replication_recovery` and `group_replication_applier` channels.
- `STOP SLAVE` stops replication threads for all channels, except the `group_replication_recovery` and `group_replication_applier` channels.
- `SHOW SLAVE STATUS` reports the status for all channels, except the `group_replication_applier` channel.
- `RESET SLAVE` resets all channels.

Warning

Use `RESET SLAVE` with caution as this statement deletes all existing channels, purges their relay log files, and recreates only the default channel.

Some replication statements cannot operate on all channels. In this case, error 1964 `Multiple channels exist on the slave. Please provide channel name as an argument.` is generated. The following statements and functions generate this error when used in a multi-source replication topology and a `FOR CHANNEL channel` option is not used to specify which channel to act on:

- `SHOW RELAYLOG EVENTS`
- `CHANGE MASTER TO`
- `MASTER_POS_WAIT()`

Note that a default channel always exists in a single source replication topology, where statements and functions behave as in previous versions of MySQL.

17.2.3.3 Startup Options and Replication Channels

This section describes startup options which are impacted by the addition of replication channels.

The following startup options *must* be configured correctly to use multi-source replication.

- `--relay-log-info-repository`

This must be set to `TABLE`. If this option is set to `FILE`, attempting to add more sources to a slave fails with `ER_SLAVE_NEW_CHANNEL_WRONG_REPOSITORY`. The `FILE` setting is now deprecated, and `TABLE` is the default.

- `--master-info-repository`

  This must be set to `TABLE`. If this option is set to `FILE`, attempting to add more sources to a slave fails with `ER_SLAVE_NEW_CHANNEL_WRONG_REPOSITORY`. The `FILE` setting is now deprecated, and `TABLE` is the default.

The following startup options now affect *all* channels in a replication topology.

- `--log-slave-updates`

  All transactions received by the slave (even from multiple sources) are written in the binary log.

- `--relay-log-purge`

  When set, each channel purges its own relay log automatically.

- `--slave_transaction_retries`

  The specified number of transaction retries can take place on all applier threads of all channels.

- `--skip-slave-start`

  No replication threads start on any channels.

- `--slave-skip-errors`

  Execution continues and errors are skipped for all channels.

The values set for the following startup options apply on each channel; since these are `mysqld` startup options, they are applied on every channel.

- `--max-relay-log-size=`*`size`*

  Maximum size of the individual relay log file for each channel; after reaching this limit, the file is rotated.

- `--relay-log-space-limit=`*`size`*

  Upper limit for the total size of all relay logs combined, for each individual channel. For *`N`* channels, the combined size of these logs is limited to `relay_log_space_limit * N`.

- `--slave-parallel-workers=`*`value`*

  Number of slave parallel workers per channel.

- `--slave-checkpoint-group`

  Waiting time by an I/O thread for each source.

- `--relay-log-index=filename`

  Base name for each channel's relay log index file. See Section 17.2.3.4, “Replication Channel Naming Conventions”.

- `--relay-log=filename`

Denotes the base name of each channel's relay log file. See Section 17.2.3.4, “Replication Channel Naming Conventions”.

- `--slave_net-timeout=N`

  This value is set per channel, so that each channel waits for *`N`* seconds to check for a broken connection.

- `--slave-skip-counter=N`

  This value is set per channel, so that each channel skips *`N`* events from its master.

17.2.3.4 Replication Channel Naming Conventions

This section describes how naming conventions are impacted by replication channels.

Each replication channel has a unique name which is a string with a maximum length of 64 characters and is case insensitive. Because channel names are used in slave tables, the character set used for these is always UTF-8. Although you are generally free to use any name for channels, the following names are reserved:

- `group_replication_applier`
- `group_replication_recovery`

The name you choose for a replication channel also influences the file names used by a multi-source replication slave. The relay log files and index files for each channel are named *`relay_log_basename-channel`*`.xxxxxx`, where *`relay_log_basename`* is a base name specified using the `relay_log` system variable, and *`channel`* is the name of the channel logged to this file. If you do not specify the `relay_log` system variable, a default file name is used that also includes the name of the channel.

17.2.4 Replication Relay and Status Logs

During replication, a slave server creates several logs that hold the binary log events relayed from the master to the slave, and record information about the current status and location within the relay log. There are three types of logs used in the process, listed here:

- The *relay log* consists of the events read from the binary log of the master and written by the slave I/O thread. Events in the relay log are executed on the slave as part of the SQL thread.
- The *master info log* contains status and current configuration information for the slave's connection to the master. This log holds information on the master host name, login credentials, and coordinates indicating how far the slave has read from the master's binary log. The master info log is written to the `mysql.slave_master_info` table.
- The *relay log info log* holds status information about the execution point within the slave's relay log. The relay log is written to the `mysql.slave_relay_log_info` table.

In MySQL 8.0, a warning is given when `mysqld` is unable to initialize the replication logging tables, but the slave is allowed to continue starting. This situation is most likely to occur when upgrading from a version of MySQL that does not support slave logging tables to one in which they are supported.

In MySQL 8.0, execution of any statement requiring a write lock on either or both of the `slave_master_info` and `slave_relay_log_info` tables is disallowed while replication is ongoing, while statements that perform only reads are permitted at any time.

Important

Do not attempt to update or insert rows in the `slave_master_info` or `slave_relay_log_info` tables manually. Doing so can cause undefined behavior, and is not supported.

Making replication resilient to unexpected halts. The `mysql.slave_master_info` and `mysql.slave_relay_log_info` tables are created using the transactional storage engine `InnoDB`. Updates to the relay log info log table are committed together with the transactions, meaning that the slave's progress information recorded in that log is always consistent with what has been applied to the database, even in the event of an unexpected server halt. The `--relay-log-recovery` option must be enabled on the slave to guarantee resilience. For more details, see Section 17.4.2, “Handling an Unexpected Halt of a Replication Slave”.

17.2.4.1 The Slave Relay Log

The relay log, like the binary log, consists of a set of numbered files containing events that describe database changes, and an index file that contains the names of all used relay log files. The default location for relay log files is the data directory.

The term “relay log file” generally denotes an individual numbered file containing database events. The term “relay log” collectively denotes the set of numbered relay log files plus the index file.

Relay log files have the same format as binary log files and can be read using `mysqlbinlog` (see Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”).

For the default replication channel, relay log file names have the default form *`host_name`*`-relay-bin.`*`nnnnnn`*, where *`host_name`* is the name of the slave server host and *`nnnnnn`* is a sequence number. Successive relay log files are created using successive sequence numbers, beginning with `000001`. For non-default replication channels, the default base name is *`host_name`*`-relay-bin-`*`channel`*, where *`channel`* is the name of the replication channel recorded in the relay log.

The slave uses an index file to track the relay log files currently in use. The default relay log index file name is *`host_name`*`-relay-bin.index` for the default channel, and *`host_name`*`-relay-bin-`*`channel`*`.index` for non-default replication channels.

The default relay log file and relay log index file names and locations can be overridden with, respectively, the `relay_log` and `relay_log_index` system variables (see Section 17.1.6, “Replication and Binary Logging Options and Variables”).

If a slave uses the default host-based relay log file names, changing a slave's host name after replication has been set up can cause replication to fail with the errors `Failed to open the relay log` and `Could not find target log during relay log initialization`. This is a known issue (see Bug #2122). If you anticipate that a slave's host name might change in the future (for example, if networking is set up on the slave such that its host name can be modified using DHCP), you can avoid this issue entirely by using the `relay_log` and `relay_log_index` system variables to specify relay log file names explicitly when you initially set up the slave. This will make the names independent of server host name changes.

If you encounter the issue after replication has already begun, one way to work around it is to stop the slave server, prepend the contents of the old relay log index file to the new one, and then restart the slave. On a Unix system, this can be done as shown here:

```
shell> cat new_relay_log_name.index >> old_relay_log_name.index
```

```
shell> mv old_relay_log_name.index new_relay_log_name.index
```

A slave server creates a new relay log file under the following conditions:

- Each time the I/O thread starts.
- When the logs are flushed (for example, with `FLUSH LOGS` or `mysqladmin flush-logs`).
- When the size of the current relay log file becomes too large, which is determined as follows:
  - If the value of `max_relay_log_size` is greater than 0, that is the maximum relay log file size.
  - If the value of `max_relay_log_size` is 0, `max_binlog_size` determines the maximum relay log file size.

The SQL thread automatically deletes each relay log file after it has executed all events in the file and no longer needs it. There is no explicit mechanism for deleting relay logs because the SQL thread takes care of doing so. However, `FLUSH LOGS` rotates relay logs, which influences when the SQL thread deletes them.

17.2.4.2 Slave Status Logs

A replication slave server creates two slave status logs in the form of `InnoDB` tables in the `mysql` system schema: the master info log `slave_master_info`, and the relay log info log `slave_relay_log_info`.

The two slave status logs contain information similar to that shown in the output of the `SHOW SLAVE STATUS` statement, which is discussed in Section 13.4.2, “SQL Statements for Controlling Slave Servers”. The slave status logs survive a slave server's shutdown. The next time the slave starts, it reads the two logs to determine how far it previously proceeded in reading binary logs from the master and in processing its own relay logs.

Access privileges for the master info log table should be restricted because it contains the password for connecting to the master. See Section 6.1.2.3, “Passwords and Logging”.

Before MySQL 8.0, to create the slave status logs as tables, it was necessary to specify the `--master-info-repository=TABLE` and `--relay-log-info-repository=TABLE` options at server startup. Otherwise, the logs were created as files in the data directory named `master.info` and `relay-log.info`, or with alternative names and locations specified by the `--master-info-file` option and `relay_log_info_file` system variable. From MySQL 8.0, creating the slave status logs as tables is the default, and creating the slave status logs as files is deprecated. For more information, see Section 17.1.6, “Replication and Binary Logging Options and Variables”.

The `mysql.slave_master_info` and `mysql.slave_relay_log_info` tables are created using the `InnoDB` transactional storage engine. Updates to the relay log info log table are committed together with the transactions, meaning that the slave's progress information recorded in that log is always consistent with what has been applied to the database, even in the event of an unexpected server halt. The `--relay-log-recovery` option must be enabled on the slave to guarantee resilience. For more details, see Section 17.4.2, “Handling an Unexpected Halt of a Replication Slave”.

One additional slave status log is created primarily for internal use, and holds status information about worker threads on a multithreaded replication slave. This slave worker log includes the names and positions for the relay log file and master binary log file for each worker thread. If the relay log info log for the slave is created as a table, which is the default, the slave worker log is written to the `mysql.slave_worker_info` table. If the relay log info log is written to a file, the slave worker log is written to the `worker-relay-log.info` file. For external use, status information for worker threads is presented in the Performance Schema `replication_applier_status_by_worker` table.

The slave I/O thread updates the master info log. The following table shows the correspondence between the columns in the `mysql.slave_master_info` table, the columns displayed by `SHOW SLAVE STATUS`, and the lines in the deprecated `master.info` file.

`slave_master_info` Table Column	`SHOW SLAVE STATUS` Column	`master.info` File Line	Description
`Number_of_lines`	[None]	1	Number of columns in the table (or lines in the file)
`Master_log_name`	`Master_Log_File`	2	The name of the master binary log currently being read from the master
`Master_log_pos`	`Read_Master_Log_Pos`	3	The current position within the master binary log that has been read from the master
`Host`	`Master_Host`	4	The host name of the master
`User_name`	`Master_User`	5	The user name used to connect to the master
`User_password`	Password (not shown by `SHOW SLAVE STATUS`)	6	The password used to connect to the master
`Port`	`Master_Port`	7	The network port used to connect to the master
`Connect_retry`	`Connect_Retry`	8	The period (in seconds) that the slave will wait before trying to reconnect to the master
`Enabled_ssl`	`Master_SSL_Allowed`	9	Indicates whether the server supports SSL connections
`Ssl_ca`	`Master_SSL_CA_File`	10	The file used for the Certificate Authority (CA) certificate
`Ssl_capath`	`Master_SSL_CA_Path`	11	The path to the Certificate Authority (CA) certificates

`slave_master_info` Table Column	`SHOW SLAVE STATUS` Column	`master.info` File Line	Description
`Ssl_cert`	`Master_SSL_Cert`	12	The name of the SSL certificate fil
`Ssl_cipher`	`Master_SSL_Cipher`	13	The list of possib ciphers used in th handshake for th SSL connection
`Ssl_key`	`Master_SSL_Key`	14	The name of the SSL key file
`Ssl_verify_server_cert`	`Master_SSL_Verify_Server_Cert`	15	Whether to verify the server certificate
`Heartbeat`	[None]	16	Interval between replication heartbeats, in seconds
`Bind`	`Master_Bind`	17	Which of the slave's network interfaces should be used for connecting to the master
`Ignored_server_ids`	`Replicate_Ignore_Server_Ids`	18	The list of server IDs to be ignored Note that for `Ignored_serve` the list of server IDs is preceded b the total number of server IDs to ignore.
`Uuid`	`Master_UUID`	19	The master's unique ID
`Retry_count`	`Master_Retry_Count`	20	Maximum numbe of reconnection attempts permitte
`Ssl_crl`	[None]	21	Path to an SSL certificate revocation-list file
`Ssl_crl_path`	[None]	22	Path to a director containing SSL certificate revocation-list file
`Enabled_auto_position`	`Auto_position`	23	If autopositioning is in use or not
`Channel_name`	`Channel_name`	24	The name of the replication chann

`slave_master_info` Table Column	`SHOW SLAVE STATUS` Column	`master.info` File Line	Description
`Tls_Version`	`Master_TLS_Version`	25	TLS version on master
`Master_public_key_path`	`Master_public_key_path`	26	Name of RSA public key file
`Get_master_public_key`	`Get_master_public_key`	27	Whether to request RSA public key from master
`Master_compression_algorithm`	[None]	28	Permitted compression algorithms
`Master_zstd_compression_level`	[None]	29	`zstd` compression level

The slave SQL thread updates the relay log info log. The following table shows the correspondence between the columns in the `mysql.slave_relay_log_info` table, the columns displayed by `SHOW SLAVE STATUS`, and the lines in the deprecated `relay-log.info` file.

`slave_relay_log_info` Table Column	`SHOW SLAVE STATUS` Column	Line in `relay-log.info` File	Description
`Number_of_lines`	[None]	1	Number of columns in the table or lines in the file
`Relay_log_name`	`Relay_Log_File`	2	The name of the current relay log file
`Relay_log_pos`	`Relay_Log_Pos`	3	The current position within the relay log file; events up to this position have been executed on the slave database
`Master_log_name`	`Relay_Master_Log_File`	4	The name of the master binary log file from which the events in the relay log file were read
`Master_log_pos`	`Exec_Master_Log_Pos`	5	The equivalent position within the master's binary log file of events that have already been executed
`Sql_delay`	`SQL_Delay`	6	The number of seconds that the slave must lag the master

slave_relay_log_info Table Column	SHOW SLAVE STATUS Column	Line in relay-log.info File	Description
Number_of_workers	[None]	7	The number of slav applier threads for executing replicatio events (transaction in parallel
Id	[None]	8	ID used for internal purposes; currently this is always 1
Channel_name	Channel_name	9	The name of the replication channel
Privilege_checks_username	[None]	10	The user name for the PRIVILEGE_CHEC account for the channel
Privilege_checks_hostname	[None]	11	The host name for the PRIVILEGE_CHEC account for the channel

When you back up the replication slave's data, ensure that you back up the `mysql.slave_master_info` and `mysql.slave_relay_log_info` tables containing the slave status logs, because they are needed to resume replication after you restore the data from the slave. If you lose the relay log files, but still have the relay log info log, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. Of course, this requires that the binary logs still exist on the master.

17.2.5 How Servers Evaluate Replication Filtering Rules

If a master server does not write a statement to its binary log, the statement is not replicated. If the server does log the statement, the statement is sent to all slaves and each slave determines whether to execute it or ignore it.

On the master, you can control which databases to log changes for by using the `--binlog-do-db` and `--binlog-ignore-db` options to control binary logging. For a description of the rules that servers use in evaluating these options, see Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”. You should not use these options to control which databases and tables are replicated. Instead, use filtering on the slave to control the events that are executed on the slave.

On the slave side, decisions about whether to execute or ignore statements received from the master are made according to the `--replicate-*` options that the slave was started with. (See Section 17.1.6, “Replication and Binary Logging Options and Variables”.) The filters governed by these options can also be set dynamically using the `CHANGE REPLICATION FILTER` statement. The rules governing such filters are the same whether they are created on startup using `--replicate-*` options or while the slave server is running by `CHANGE REPLICATION FILTER`. Note that replication filters cannot be used on Group Replication-specific channels on a MySQL server instance that is configured for Group Replication, because filtering transactions on some servers would make the group unable to reach agreement on a consistent state.

In the simplest case, when there are no `--replicate-*` options, the slave executes all statements that it receives from the master. Otherwise, the result depends on the particular options given.

Database-level options (`--replicate-do-db`, `--replicate-ignore-db`) are checked first; see Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”, for a description of this process. If no database-level options are used, option checking proceeds to any table-level options that may be in use (see Section 17.2.5.2, “Evaluation of Table-Level Replication Options”, for a discussion of these). If one or more database-level options are used but none are matched, the statement is not replicated.

For statements affecting databases only (that is, `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`), database-level options always take precedence over any `--replicate-wild-do-table` options. In other words, for such statements, `--replicate-wild-do-table` options are checked if and only if there are no database-level options that apply.

To make it easier to determine what effect an option set will have, it is recommended that you avoid mixing “do” and “ignore” options, or wildcard and nonwildcard options.

If any `--replicate-rewrite-db` options were specified, they are applied before the `--replicate-*` filtering rules are tested.

Note

All replication filtering options follow the same rules for case sensitivity that apply to names of databases and tables elsewhere in the MySQL server, including the effects of the `lower_case_table_names` system variable.

17.2.5.1 Evaluation of Database-Level Replication and Binary Logging Options

When evaluating replication options, the slave begins by checking to see whether there are any `--replicate-do-db` or `--replicate-ignore-db` options that apply. When using `--binlog-do-db` or `--binlog-ignore-db`, the process is similar, but the options are checked on the master.

The database that is checked for a match depends on the binary log format of the statement that is being handled. If the statement has been logged using the row format, the database where data is to be changed is the database that is checked. If the statement has been logged using the statement format, the default database (specified with a `USE` statement) is the database that is checked.

Note

Only DML statements can be logged using the row format. DDL statements are always logged as statements, even when `binlog_format=ROW`. All DDL statements are therefore always filtered according to the rules for statement-based replication. This means that you must select the default database explicitly with a `USE` statement in order for a DDL statement to be applied.

For replication, the steps involved are listed here:

1. Which logging format is used?
   - **STATEMENT.** Test the default database.
   - **ROW.** Test the database affected by the changes.
2. Are there any `--replicate-do-db` options?
   - **Yes.** Does the database match any of them?

- **Yes.** Continue to Step 4.
- **No.** Ignore the update and exit.

- **No.** Continue to step 3.

3. Are there any `--replicate-ignore-db` options?
   - **Yes.** Does the database match any of them?
     - **Yes.** Ignore the update and exit.
     - **No.** Continue to step 4.
   - **No.** Continue to step 4.
4. Proceed to checking the table-level replication options, if there are any. For a description of how these options are checked, see Section 17.2.5.2, “Evaluation of Table-Level Replication Options”.

Important

A statement that is still permitted at this stage is not yet actually executed. The statement is not executed until all table-level options (if any) have also been checked, and the outcome of that process permits execution of the statement.

For binary logging, the steps involved are listed here:

1. Are there any `--binlog-do-db` or `--binlog-ignore-db` options?
   - **Yes.** Continue to step 2.
   - **No.** Log the statement and exit.
2. Is there a default database (has any database been selected by `USE`)?
   - **Yes.** Continue to step 3.
   - **No.** Ignore the statement and exit.
3. There is a default database. Are there any `--binlog-do-db` options?
   - **Yes.** Do any of them match the database?
     - **Yes.** Log the statement and exit.
     - **No.** Ignore the statement and exit.
   - **No.** Continue to step 4.
4. Do any of the `--binlog-ignore-db` options match the database?
   - **Yes.** Ignore the statement and exit.
   - **No.** Log the statement and exit.

Important

For statement-based logging, an exception is made in the rules just given for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. In

those cases, the database being *created, altered, or dropped* replaces the default database when determining whether to log or ignore updates.

`--binlog-do-db` can sometimes mean “ignore other databases”. For example, when using statement-based logging, a server running with only `--binlog-do-db=sales` does not write to the binary log statements for which the default database differs from `sales`. When using row-based logging with the same option, the server logs only those updates that change data in `sales`.

17.2.5.2 Evaluation of Table-Level Replication Options

The slave checks for and evaluates table options only if either of the following two conditions is true:

- No matching database options were found.
- One or more database options were found, and were evaluated to arrive at an “execute” condition according to the rules described in the previous section (see Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”).

First, as a preliminary condition, the slave checks whether statement-based replication is enabled. If so, and the statement occurs within a stored function, the slave executes the statement and exits. If row-based replication is enabled, the slave does not know whether a statement occurred within a stored function on the master, so this condition does not apply.

Note

For statement-based replication, replication events represent statements (all changes making up a given event are associated with a single SQL statement); for row-based replication, each event represents a change in a single table row (thus a single statement such as `UPDATE mytable SET mycol = 1` may yield many row-based events). When viewed in terms of events, the process of checking table options is the same for both row-based and statement-based replication.

Having reached this point, if there are no table options, the slave simply executes all events. If there are any `--replicate-do-table` or `--replicate-wild-do-table` options, the event must match one of these if it is to be executed; otherwise, it is ignored. If there are any `--replicate-ignore-table` or `--replicate-wild-ignore-table` options, all events are executed except those that match any of these options.

The following steps describe this evaluation in more detail. The starting point is the end of the evaluation of the database-level options, as described in Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”.

1. Are there any table replication options?
   - **Yes.** Continue to step 2.
   - **No.** Execute the update and exit.
2. Which logging format is used?
   - **STATEMENT.** Carry out the remaining steps for each statement that performs an update.
   - **ROW.** Carry out the remaining steps for each update of a table row.
3. Are there any `--replicate-do-table` options?
   - **Yes.** Does the table match any of them?

    - **Yes.** Execute the update and exit.
    - **No.** Continue to step 4.
  - **No.** Continue to step 4.
4. Are there any `--replicate-ignore-table` options?
  - **Yes.** Does the table match any of them?
    - **Yes.** Ignore the update and exit.
    - **No.** Continue to step 5.
  - **No.** Continue to step 5.
5. Are there any `--replicate-wild-do-table` options?
  - **Yes.** Does the table match any of them?
    - **Yes.** Execute the update and exit.
    - **No.** Continue to step 6.
  - **No.** Continue to step 6.
6. Are there any `--replicate-wild-ignore-table` options?
  - **Yes.** Does the table match any of them?
    - **Yes.** Ignore the update and exit.
    - **No.** Continue to step 7.
  - **No.** Continue to step 7.
7. Is there another table to be tested?
  - **Yes.** Go back to step 3.
  - **No.** Continue to step 8.
8. Are there any `--replicate-do-table` or `--replicate-wild-do-table` options?
  - **Yes.** Ignore the update and exit.
  - **No.** Execute the update and exit.

Note

Statement-based replication stops if a single SQL statement operates on both a table that is included by a `--replicate-do-table` or `--replicate-wild-do-table` option, and another table that is ignored by a `--replicate-ignore-table` or `--replicate-wild-ignore-table` option. The slave must either execute or ignore the complete statement (which forms a replication event), and it cannot logically do this. This also applies to row-based replication for DDL statements, because DDL statements are always logged as statements, without regard to the logging format in effect. The only type of statement that can update

both an included and an ignored table and still be replicated successfully is a DML statement that has been logged with `binlog_format=ROW`.

17.2.5.3 Replication Rule Application

This section provides additional explanation and examples of usage for different combinations of replication filtering options.

Some typical combinations of replication filter rule types are given in the following table:

Condition (Types of Options)	Outcome
No `--replicate-*` options at all:	The slave executes all events that it receives from the master.
`--replicate-*-db` options, but no table options:	The slave accepts or ignores events using the database options. It executes all events permitted by those options because there are no table restrictions.
`--replicate-*-table` options, but no database options:	All events are accepted at the database-checking stage because there are no database conditions. The slave executes or ignores events based solely on the table options.
A combination of database and table options:	The slave accepts or ignores events using the database options. Then it evaluates all events permitted by those options according to the table options. This can sometimes lead to results that seem counterintuitive, and that may be different depending on whether you are using statement-based or row-based replication; see the text for an example.

A more complex example follows, in which we examine the outcomes for both statement-based and row-based settings.

Suppose that we have two tables `mytbl1` in database `db1` and `mytbl2` in database `db2` on the master, and the slave is running with the following options (and no other replication filtering options):

```
replicate-ignore-db = db1
replicate-do-table  = db2.tbl2
```

Now we execute the following statements on the master:

```
USE db1;
INSERT INTO db2.tbl2 VALUES (1);
```

The results on the slave vary considerably depending on the binary log format, and may not match initial expectations in either case.

Statement-based replication. The `USE` statement causes `db1` to be the default database. Thus the `--replicate-ignore-db` option matches, *and the `INSERT` statement is ignored*. The table options are not checked.

Row-based replication. The default database has no effect on how the slave reads database options when using row-based replication. Thus, the `USE` statement makes no difference in how the `--replicate-ignore-db` option is handled: the database specified by this option does not match the database where the `INSERT` statement changes data, so the slave proceeds to check the table options. The table specified by `--replicate-do-table` matches the table to be updated, *and the row is inserted*.

17.2.5.4 Replication Channel Based Filters

This section explains how to work with replication filters when multiple replication channels exist, for example in a multi-source replication topology. Before MySQL 8.0, replication filters were global - filters were applied to all replication channels. From MySQL 8.0, replication filters can be global or channel specific, enabling you to configure multi-source replication slaves with replication filters on specific replication channels. Channel specific replication filters are particularly useful in a multi-source replication topology when the same database or table is present on multiple masters, and the slave is only required to replicate it from one master.

For more background information, see Section 17.1.4, “MySQL Multi-Source Replication” and Section 17.2.3, “Replication Channels”.

Important

On a MySQL server instance that is configured for Group Replication, channel specific replication filters can be used on replication channels that are not directly involved with Group Replication, such as where a group member also acts as a replication slave to a master that is outside the group. They cannot be used on the `group_replication_applier` or `group_replication_recovery` channels. Filtering on these channels would make the group unable to reach agreement on a consistent state.

Overview of Replication Filters and Channels

When multiple replication channels exist, for example in a multi-source replication topology, replication filters are applied as follows:

- Any global replication filter specified is added to the global replication filters of the filter type (`do_db`, `do_ignore_table`, and so on).
- Any channel specific replication filter adds the filter to the specified channel’s replication filters for the specified filter type.
- Each slave replication channel copies global replication filters to its channel specific replication filters if no channel specific replication filter of this type is configured.
- Each channel uses its channel specific replication filters to filter the replication stream.

The syntax to create channel specific replication filters extends the existing SQL statements and command options. When a replication channel is not specified the global replication filter is configured to ensure backwards compatibility. The `CHANGE REPLICATION FILTER` statement supports the `FOR CHANNEL` clause to configure channel specific filters online. The `--replicate-*` command options to configure filters can specify a replication channel using the form `--replicate-`*`filter_type`*`=`*`channel_name`*`:`*`filter_details`*. For example, suppose channels `channel_1` and `channel_2` exist before the server starts, starting the slave with the command line options `--replicate-do-db=db1` `--replicate-do-db=channel_1:db2` `--replicate-do-db=db3` `--replicate-ignore-db=db4` `--replicate-ignore-db=channel_2:db5` would result in:

- *Global replication filters*: do_db=db1,db3, ignore_db=db4
- *Channel specific filters on channel_1*: do_db=db2 ignore_db=db4
- *Channel specific filters on channel_2*: do_db=db1,db3 ignore_db=db5

To monitor the replication filters in such a setup use the `replication_applier_global_filters` and `replication_applier_filters` tables.

Configuring Channel Specific Replication Filters at Startup

The replication filter related command options can take an optional *`channel`* followed by a colon, followed by the filter specification. The first colon is interpreted as a separator, subsequent colons are interpreted as literal colons. The following command options support channel specific replication filters using this format:

- `--replicate-do-db=`*`channel`*`:`*`database_id`*
- `--replicate-ignore-db=`*`channel`*`:`*`database_id`*
- `--replicate-do-table=`*`channel`*`:`*`table_id`*
- `--replicate-ignore-table=`*`channel`*`:`*`table_id`*
- `--replicate-rewrite-db=`*`channel`*`:`*`db1-db2`*
- `--replicate-wild-do-table=`*`channel`*`:`*`table regexid`*
- `--replicate-wild-ignore-table=`*`channel`*`:`*`table regexid`*

If you use a colon but do not specify a *`channel`* for the filter option, for example `--replicate-do-db=:`*`database_id`*, the option configures the replication filter for the default replication channel. The default replication channel is the replication channel which always exists once replication has been started, and differs from multi-source replication channels which you create manually. When neither the colon nor a *`channel`* is specified the option configures the global replication filters, for example `--replicate-do-db=`*`database_id`* configures the global `--replicate-do-db` filter.

If you configure multiple `rewrite-db=`*`from_name->to_name`* options with the same *`from_name`* database, all filters are added together (put into the `rewrite_do` list) and the first one takes effect.

Changing Channel Specific Replication Filters Online

In addition to the `--replicate-*` options, replication filters can be configured using the `CHANGE REPLICATION FILTER` statement. This removes the need to restart the server, but the slave applier thread must be stopped while making the change. To make this statement apply the filter to a specific channel, use the `FOR CHANNEL` *`channel`* clause. For example:

```
CHANGE REPLICATION FILTER REPLICATE_DO_DB=(db1) FOR CHANNEL channel_1;
```

When a `FOR CHANNEL` clause is provided, the statement acts on the specified channel's replication filters. If multiple types of filters (`do_db`, `do_ignore_table`, `wild_do_table`, and so on) are specified, only the specified filter types are replaced by the statement. In a replication topology with multiple channels, for example on a multi-source replication slave, when no `FOR CHANNEL` clause is provided, the statement acts on the global replication filters and all channels' replication filters, using a similar logic as the `FOR CHANNEL` case. For more information see Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”.

Removing Channel Specific Replication Filters

When channel specific replication filters have been configured, you can remove the filter by issuing an empty filter type statement. For example to remove all `REPLICATE_REWRITE_DB` filters from a replication channel named `channel_1` issue:

```
CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB=() FOR CHANNEL channel_1;
```

Any `REPLICATE_REWRITE_DB` filters previously configured, using either command options or `CHANGE REPLICATION FILTER`, are removed.

The `RESET SLAVE ALL` statement removes channel specific replication filters that were set on channels deleted by the statement. When the deleted channel or channels are recreated, any global replication filters specified for the slave are copied to them, and no channel specific replication filters are applied.

17.3 Replication Security

To protect against unauthorized access to data that is stored on and transferred between replication masters and slaves, set up all the servers involved using the security measures that you would choose for any MySQL instance in your installation, as described in Chapter 6, *Security*. In addition, for servers in a replication topology, consider implementing the following security measures:

- Set up masters and slaves to use encrypted connections to transfer the binary log, which protects this data in motion. Encryption for these connections must be activated using a `CHANGE MASTER TO` statement, in addition to setting up the servers to support encrypted network connections. See Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”.
- Encrypt the binary log files and relay log files on masters and slaves, which protects this data at rest, and also any data in use in the binary log cache. Binary log encryption is activated using the `binlog_encryption` system variable. See Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”.
- Apply privilege checks to replication appliers, which help to secure replication channels against the unauthorized or accidental use of privileged or unwanted operations. Privilege checks are implemented by setting up a `PRIVILEGE_CHECKS_USER` account, which MySQL uses to verify that you have authorized each specific transaction for that channel. See Section 17.3.3, “Replication Privilege Checks”.

For Group Replication, binary log encryption and privilege checks can be used as a security measure on replication group members. You should also consider encrypting the connections between group members, comprising group communication connections and distributed recovery connections, and applying IP address whitelisting to exclude untrusted hosts. For information on these security measures specific to Group Replication, see Section 18.5, “Group Replication Security”.

17.3.1 Setting Up Replication to Use Encrypted Connections

To use an encrypted connection for the transfer of the binary log required during replication, both the master and the slave servers must support encrypted network connections. If either server does not support encrypted connections (because it has not been compiled or configured for them), replication through an encrypted connection is not possible.

Setting up encrypted connections for replication is similar to doing so for client/server connections. You must obtain (or create) a suitable security certificate that you can use on the master, and a similar certificate (from the same certificate authority) on each slave. You must also obtain suitable key files.

For more information on setting up a server and client for encrypted connections, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

To enable encrypted connections on the master, you must create or obtain suitable certificate and key files, and then add the following configuration options to the master's configuration within the `[mysqld]` section of the master's `my.cnf` file, changing the file names as necessary:

```
[mysqld]
ssl-ca=cacert.pem
ssl-cert=server-cert.pem
ssl-key=server-key.pem
```

The paths to the files may be relative or absolute; we recommend that you always use complete paths for this purpose.

The options are as follows:

- `--ssl-ca`: The path name of the Certificate Authority (CA) certificate file. (`--ssl-capath` is similar but specifies the path name of a directory of CA certificate files.)
- `--ssl-cert`: The path name of the server public key certificate file. This certificate can be sent to the client and authenticated against the CA certificate that it has.
- `--ssl-key`: The path name of the server private key file.

To enable encrypted connections on the slave, use the `CHANGE MASTER TO` statement. You can either name the slave certificate and SSL private key files required for the encrypted connection in the `[client]` section of the slave's `my.cnf` file, or you can explicitly specify that information using the `CHANGE MASTER TO` statement. For more information on the `CHANGE MASTER TO` statement, see Section 13.4.2.1, "CHANGE MASTER TO Syntax".

- To name the slave certificate and key files using an option file, add the following lines to the `[client]` section of the slave's `my.cnf` file, changing the file names as necessary:

```
[client]
ssl-ca=cacert.pem
ssl-cert=client-cert.pem
ssl-key=client-key.pem
```

- Restart the slave server, using the `--skip-slave-start` option to prevent the slave from connecting to the master. Use `CHANGE MASTER TO` to specify the master configuration, and add the `MASTER_SSL` option to connect using encryption:

```
mysql> CHANGE MASTER TO
    -> MASTER_HOST='master_hostname',
    -> MASTER_USER='repl',
    -> MASTER_PASSWORD='password',
    -> MASTER_SSL=1;
```

  Setting `MASTER_SSL=1` for a replication connection and then setting no further `MASTER_SSL_xxx` options corresponds to setting `--ssl-mode=REQUIRED` for the client, as described in Command Options for Encrypted Connections. With `MASTER_SSL=1`, the connection attempt only succeeds if an encrypted connection can be established. A replication connection does not fall back to an unencrypted connection, so there is no setting corresponding to the `--ssl-mode=PREFERRED` setting for replication. If `MASTER_SSL=0` is set, this corresponds to `--ssl-mode=DISABLED`.

- To name the slave certificate and SSL private key files using the `CHANGE MASTER TO` statement, if you did not do this in the slave's `my.cnf` file, add the appropriate `MASTER_SSL_xxx` options:

```
    -> MASTER_SSL_CA = 'ca_file_name',
    -> MASTER_SSL_CAPATH = 'ca_directory_name',
    -> MASTER_SSL_CERT = 'cert_file_name',
    -> MASTER_SSL_KEY = 'key_file_name',
```

  These options correspond to the `--ssl-xxx` options with the same names, as described in Command Options for Encrypted Connections. For these options to take effect, `MASTER_SSL=1` must also be set. For a replication connection, specifying a value for either of `MASTER_SSL_CA` or `MASTER_SSL_CAPATH`, or specifying these options in the slave's `my.cnf` file, corresponds to setting `--ssl-mode=VERIFY_CA`. The connection attempt only succeeds if a valid matching Certificate Authority (CA) certificate is found using the specified information.

- To activate host name identity verification, add the `MASTER_SSL_VERIFY_SERVER_CERT` option:

```
    -> MASTER_SSL_VERIFY_SERVER_CERT=1,
```

This option corresponds to the `--ssl-verify-server-cert` option, which was deprecated from MySQL 5.7 and removed in MySQL 8.0. For a replication connection, specifying `MASTER_SSL_VERIFY_SERVER_CERT=1` corresponds to setting `--ssl-mode=VERIFY_IDENTITY`, as described in Command Options for Encrypted Connections. For this option to take effect, `MASTER_SSL=1` must also be set. Host name identity verification does not work with self-signed certificates.

- To activate certificate revocation list (CRL) checks, add the `MASTER_SSL_CRL` or `MASTER_SSL_CRLPATH` option:

```
    -> MASTER_SSL_CRL = 'crl_file_name',
    -> MASTER_SSL_CRLPATH = 'crl_directory_name',
```

These options correspond to the `--ssl-`*`xxx`* options with the same names, as described in Command Options for Encrypted Connections. If they are not specified, no CRL checking takes place.

- To specify lists of ciphers and encryption protocols permitted by the slave for the replication connection, add the `MASTER_SSL_CIPHER` and `MASTER_TLS_VERSION` options:

```
    -> MASTER_SSL_CIPHER = 'cipher_list',
    -> MASTER_TLS_VERSION = 'protocol_list',
```

The `MASTER_SSL_CIPHER` option specifies the list of ciphers permitted by the slave for the replication connection, with one or more cipher names separated by colons. The `MASTER_TLS_VERSION` option specifies the encryption protocols permitted by the slave for the replication connection. The format is like that for the `tls_version` system variable, with one or more comma-separated protocol versions. The protocols and ciphers that you can use in these lists depend on the SSL library used to compile MySQL. For information about the formats and permitted values, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- After the master information has been updated, start the slave replication process:

```
mysql> START SLAVE;
```

You can use the `SHOW SLAVE STATUS` statement to confirm that an encrypted connection was established successfully.

- Requiring encrypted connections on the slave does not ensure that the master requires encrypted connections from slaves. If you want to ensure that the master only accepts replication slaves that connect using encrypted connections, create a replication user account on the master using the `REQUIRE SSL` option, then grant that user the `REPLICATION SLAVE` privilege. For example:

```
mysql> CREATE USER 'repl'@'%.example.com' IDENTIFIED BY 'password'
    -> REQUIRE SSL;
mysql> GRANT REPLICATION SLAVE ON *.*
    -> TO 'repl'@'%.example.com';
```

If you have an existing replication user account on the master, you can add `REQUIRE SSL` to it with this statement:

```
mysql> ALTER USER 'repl'@'%.example.com' REQUIRE SSL;
```

17.3.2 Encrypting Binary Log Files and Relay Log Files

From MySQL 8.0.14, binary log files and relay log files can be encrypted, helping to protect these files and the potentially sensitive data contained in them from being misused by outside attackers, and also from unauthorized viewing by users of the operating system where they are stored. The encryption algorithm used for the files, the AES (Advanced Encryption Standard) cipher algorithm, is built in to MySQL Server and cannot be configured.

You enable this encryption on a MySQL server by setting the `binlog_encryption` system variable to `ON`. `OFF` is the default. The system variable sets encryption on for binary log files and relay log files. Binary logging does not need to be enabled on the server to enable encryption, so you can encrypt the relay log files on a slave that has no binary log. To use encryption, a keyring plugin must be installed and configured to supply MySQL Server's keyring service. For instructions to do this, see Section 6.4.4, “The MySQL Keyring”. Any supported keyring plugin can be used to store binary log encryption keys.

When you first start the server with encryption enabled, a new binary log encryption key is generated before the binary log and relay logs are initialized. This key is used to encrypt a file password for each binary log file (if the server has binary logging enabled) and relay log file (if the server has replication channels), and further keys generated from the file passwords are used to encrypt the data in the files. The binary log encryption key that is currently in use on the server is called the binary log master key. The two tier encryption key architecture means that the binary log master key can be rotated (replaced by a new master key) as required, and only the file password for each file needs to be re-encrypted with the new master key, not the whole file. Relay log files are encrypted for all channels, including new channels that are created after encryption is activated. The binary log index file and relay log index file are never encrypted.

If you activate encryption while the server is running, a new binary log encryption key is generated at that time. The exception is if encryption was active previously on the server and was then disabled, in which case the binary log encryption key that was in use before is used again. The binary log file and relay log files are rotated immediately, and file passwords for the new files and all subsequent binary log files and relay log files are encrypted using this binary log encryption key. Existing binary log files and relay log files still present on the server are not encrypted, but you can purge them if they are no longer needed.

If you deactivate encryption by changing the `binlog_encryption` system variable to `OFF`, the binary log file and relay log files are rotated immediately and all subsequent logging is unencrypted. Previously encrypted files are not automatically decrypted, but the server is still able to read them. The `BINLOG_ENCRYPTION_ADMIN` privilege is required to activate or deactivate encryption while the server is running.

Encrypted and unencrypted binary log files can be distinguished using the magic number at the start of the file header for encrypted log files (`0xFD62696E`), which differs from that used for unencrypted log files (`0xFE62696E`). The `SHOW BINARY LOGS` statement shows whether each binary log file is encrypted or unencrypted.

When binary log files have been encrypted, `mysqlbinlog` cannot read them directly, but can read them from the server using the `--read-from-remote-server` option. From MySQL 8.0.14, `mysqlbinlog` returns a suitable error if you attempt to read an encrypted binary log file directly, but older versions of `mysqlbinlog` do not recognise the file as a binary log file at all. If you back up encrypted binary log files using `mysqlbinlog`, note that the copies of the files that are generated using `mysqlbinlog` are stored in an unencrypted format.

17.3.2.1 Scope of Binary Log Encryption

When binary log encryption is active for a MySQL server instance, the encryption coverage is as follows:

- Data at rest that is written to the binary log files and relay log files is encrypted from the point in time where encryption is started, using the two tier encryption architecture described above. Existing

binary log files and relay log files that were present on the server when you started encryption are not encrypted. You can purge these files when they are no longer needed.

- Data in motion in the replication event stream, which is sent to MySQL clients including `mysqlbinlog`, is decrypted for transmission, and should therefore be protected in transit by the use of connection encryption (see Section 6.3, “Using Encrypted Connections” and Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”).

- Data in use that is held in the binary log transaction and statement caches during a transaction is in unencrypted format in the memory buffer that stores the cache. The data is written to a temporary file on disk if it exceeds the space available in the memory buffer. From MySQL 8.0.17, when binary log encryption is active on the server, temporary files used to hold the binary log cache are encrypted using AES-CTR (AES Counter mode) for stream encryption. Because the temporary files are volatile and tied to a single process, they are encrypted using single-tier encryption, using a randomly generated file password and initialization vector that exist only in memory and are never stored on disk or in the keyring. After each transaction is committed, the binary log cache is reset: the memory buffer is cleared, any temporary file used to hold the binary log cache is truncated, and a new file password and initialization vector are randomly generated for use with the next transaction. This reset also takes place when the server is restarted after a normal shutdown or an unexpected halt.

Note

If you use `LOAD DATA` when `binlog_format=STATEMENT` is set, which is not recommended as the statement is considered unsafe for statement-based replication, a temporary file containing the data is created on the replication slave where the changes are applied. These temporary files are not encrypted when binary log encryption is active on the server. Use row-based or mixed binary logging format instead, which do not create the temporary files.

17.3.2.2 Binary Log Encryption Keys

The binary log encryption keys used to encrypt the file passwords for the log files are 256-bit keys that are generated specifically for each MySQL server instance using MySQL Server's keyring service (see Section 6.4.4, “The MySQL Keyring”). The keyring service handles the creation, retrieval, and deletion of the binary log encryption keys. A server instance only creates and removes keys generated for itself, but it can read keys generated for other instances if they are stored in the keyring, as in the case of a server instance that has been cloned by file copying.

Important

The binary log encryption keys for a MySQL server instance must be included in your backup and recovery procedures, because if the keys required to decrypt the file passwords for current and retained binary log files or relay log files are lost, it might not be possible to start the server.

The format of binary log encryption keys in the keyring is as follows:

```
MySQLReplicationKey_{UUID}_{SEQ_NO}
```

For example:

```
MySQLReplicationKey_00508583-b5ce-11e8-a6a5-0010e0734796_1
```

`{UUID}` is the true UUID generated by the MySQL server (the value of the `server_uuid` system variable). `{SEQ_NO}` is the sequence number for the binary log encryption key, which is incremented by 1 for each new key that is generated on the server.

The binary log encryption key that is currently in use on the server is called the binary log master key. The sequence number for the current binary log master key is stored in the keyring. The binary log master key is used to encrypt each new log file's file password, which is a randomly generated 32-byte file password specific to the log file that is used to encrypt the file data. The file password is encrypted using AES-CBC (AES Cipher Block Chaining mode) with the 256-bit binary log encryption key and a random initialization vector (IV), and is stored in the log file's file header. The file data is encrypted using AES-CTR (AES Counter mode) with a 256-bit key generated from the file password and a nonce also generated from the file password. It is technically possible to decrypt an encrypted file offline, if the binary log encryption key used to encrypt the file password is known, by using tools available in the OpenSSL cryptography toolkit.

If you use file copying to clone a MySQL server instance that has encryption active so its binary log files and relay log files are encrypted, ensure that the keyring is also copied, so that the clone server can read the binary log encryption keys from the source server. When encryption is activated on the clone server (either at startup or subsequently), the clone server recognizes that the binary log encryption keys used with the copied files include the generated UUID of the source server. It automatically generates a new binary log encryption key using its own generated UUID, and uses this to encrypt the file passwords for subsequent binary log files and relay log files. The copied files continue to be read using the source server's keys.

17.3.2.3 Binary Log Master Key Rotation

When binary log encryption is enabled, you can rotate the binary log master key at any time while the server is running by issuing `ALTER INSTANCE ROTATE BINLOG MASTER KEY`. When the binary log master key is rotated manually using this statement, the passwords for the new and subsequent files are encrypted using the new binary log master key, and also the file passwords for existing encrypted binary log files and relay log files are re-encrypted using the new binary log master key, so the encryption is renewed completely. You can rotate the binary log master key on a regular basis to comply with your organization's security policy, and also if you suspect that the current or any of the previous binary log master keys might have been compromised.

When you rotate the binary log master key manually, MySQL Server takes the following actions in sequence:

1. A new binary log encryption key is generated with the next available sequence number, stored on the keyring, and used as the new binary log master key.
2. The binary log and relay log files are rotated on all channels.
3. The new binary log master key is used to encrypt the file passwords for the new binary log and relay log files, and subsequent files until the key is changed again.
4. The file passwords for existing encrypted binary log files and relay log files on the server are re-encrypted in turn using the new binary log master key, starting with the most recent files. Any unencrypted files are skipped.
5. Binary log encryption keys that are no longer in use for any files after the re-encryption process are removed from the keyring.

The `BINLOG_ENCRYPTION_ADMIN` privilege is required to issue `ALTER INSTANCE ROTATE BINLOG MASTER KEY`, and the statement cannot be used if the `binlog_encryption` system variable is set to `OFF`.

As the final step of the binary log master key rotation process, all binary log encryption keys that no longer apply to any retained binary log files or relay log files are cleaned up from the keyring. If a retained binary log file or relay log file cannot be initialized for re-encryption, the relevant binary log encryption keys are not deleted in case the files can be recovered in the future. For example, this might be the case if a file listed in

a binary log index file is currently unreadable, or if a channel fails to initialize. If the server UUID changes, for example because a backup created using MySQL Enterprise Backup is used to set up a new replication slave, issuing `ALTER INSTANCE ROTATE BINLOG MASTER KEY` on the new server does not delete any earlier binary log encryption keys that include the original server UUID.

If any of the first four steps of the binary log master key rotation process cannot be completed correctly, an error message is issued explaining the situation and the consequences for the encryption status of the binary log files and relay log files. Files that were previously encrypted are always left in an encrypted state, but their file passwords might still be encrypted using an old binary log master key. If you see these errors, first retry the process by issuing `ALTER INSTANCE ROTATE BINLOG MASTER KEY` again. Then investigate the status of individual files to see what is blocking the process, especially if you suspect that the current or any of the previous binary log master keys might have been compromised.

If the final step of the binary log master key rotation process cannot be completed correctly, a warning message is issued explaining the situation. The warning message identifies whether the process could not clean up the auxiliary keys in the keyring for rotating the binary log master key, or could not clean up unused binary log encryption keys. You can choose to ignore the message as the keys are auxiliary keys or no longer in use, or you can issue `ALTER INSTANCE ROTATE BINLOG MASTER KEY` again to retry the process.

If the server stops and is restarted with binary log encryption still set to `ON` during the binary log master key rotation process, new binary log files and relay log files after the restart are encrypted using the new binary log master key. However, the re-encryption of existing files is not continued, so files that did not get re-encrypted before the server stopped are left encrypted using the previous binary log master key. To complete re-encryption and clean up unused binary log encryption keys, issue `ALTER INSTANCE ROTATE BINLOG MASTER KEY` again after the restart.

`ALTER INSTANCE ROTATE BINLOG MASTER KEY` actions are not written to the binary log and are not executed on replication slaves. Binary log master key rotation can therefore be carried out in replication environments including a mix of MySQL versions. To schedule regular rotation of the binary log master key on all applicable master and slave servers, you can enable the MySQL Event Scheduler on each server and issue the `ALTER INSTANCE ROTATE BINLOG MASTER KEY` statement using a `CREATE EVENT` statement. If you rotate the binary log master key because you suspect that the current or any of the previous binary log master keys might have been compromised, issue the statement on every applicable master and slave server. Issuing the statement on individual servers ensures that you can verify immediate compliance, even in the case of slaves that are lagging, belong to multiple replication topologies, or are not currently active in the replication topology but have binary log and relay log files.

The `binlog_rotate_encryption_master_key_at_startup` system variable controls whether the binary log master key is automatically rotated when the server is restarted. If this system variable is set to `ON`, a new binary log encryption key is generated and used as the new binary log master key whenever the server is restarted. If it is set to `OFF`, which is the default, the existing binary log master key is used again after the restart. When the binary log master key is rotated at startup, the file passwords for the new binary log and relay log files are encrypted using the new key. The file passwords for the existing encrypted binary log files and relay log files are not re-encrypted, so they remain encrypted using the old key, which remains available on the keyring.

17.3.3 Replication Privilege Checks

By default, MySQL replication (including Group Replication) does not carry out privilege checks when transactions that were already accepted by another server are applied on a replication slave or group member. From MySQL 8.0.18, you can create a user account with the appropriate privileges to apply the transactions that are normally replicated on a channel, and specify this as the `PRIVILEGE_CHECKS_USER` account for the replication applier. MySQL then checks each transaction against the user account's privileges to verify that you have authorized the operation for that channel. The account can also be safely

used by an administrator to apply or reapply transactions from `mysqlbinlog` output, for example to recover from a replication error on the channel.

The use of a `PRIVILEGE_CHECKS_USER` account helps secure a replication channel against the unauthorized or accidental use of privileged or unwanted operations. The `PRIVILEGE_CHECKS_USER` account provides an additional layer of security in situations such as these:

- You are replicating between a server instance on your organization's network, and a server instance on another network, such as an instance supplied by a cloud service provider.
- You want to have multiple on-premise or off-site deployments administered as separate units, without giving one administrator account privileges on all the deployments.
- You want to have an administrator account that enables an administrator to perform only operations that are directly relevant to the replication channel and the databases it replicates, rather than having wide privileges on the server instance.

You grant the `REPLICATION_APPLIER` privilege to enable a user account to appear as the `PRIVILEGE_CHECKS_USER` for a replication applier thread, and to execute the internal-use `BINLOG` statements used by mysqlbinlog. The user name and host name for the `PRIVILEGE_CHECKS_USER` account must follow the syntax described in Section 6.2.4, “Specifying Account Names”, and the user must not be an anonymous user (with a blank user name) or the `CURRENT_USER`. To create a new account, use `CREATE USER`. To grant this account the `REPLICATION_APPLIER` privilege, use the `GRANT` statement. For example, to create a user account `priv_repl`, which can be used manually by an administrator from any host in the `example.com` domain, and requires an encrypted connection, issue the following statements:

```
mysql> SET sql_log_bin = 0;
mysql> CREATE USER 'priv_repl'@'%.example.com' IDENTIFIED BY 'password' REQUIRE SSL;
mysql> GRANT REPLICATION_APPLIER ON *.* TO 'priv_repl'@'%.example.com';
mysql> SET sql_log_bin = 1;
```

The `SET sql_log_bin` statements are used so that the account management statements are not added to the binary log and sent to the replication channels (see Section 13.4.1.3, “SET sql_log_bin Syntax”).

Important

The `caching_sha2_password` authentication plugin is the default for new users created from MySQL 8.0 (for details, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”). To connect to a server using a user account that authenticates with this plugin, you must either set up an encrypted connection as described in Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”, or enable the unencrypted connection to support password exchange using an RSA key pair.

After setting up the user account, use the `GRANT` statement to grant additional privileges to enable the user account to make the database changes that you expect the applier thread to carry out, such as updating specific tables held on the server. These same privileges enable an administrator to use the account if they need to execute any of those transactions manually on the replication channel. If an unexpected operation is attempted for which you did not grant the appropriate privileges, the operation is disallowed and the replication applier thread stops with an error. Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account” explains what additional privileges the account needs. For example, to grant the `priv_repl` user account the `INSERT` privilege to add rows to the `cust` table in `db1`, issue the following statement:

```
mysql> GRANT INSERT ON db1.cust TO 'priv_repl'@'%.example.com';
```

You assign the PRIVILEGE_CHECKS_USER account for a replication channel using a CHANGE MASTER TO statement. If replication is running, issue STOP SLAVE before the CHANGE MASTER TO statement, and START SLAVE after it. If you do not specify a channel and no other channels exist, the statement is applied to the default channel. For example, to start privilege checks on the channel channel_1 on a running replication slave, issue the following statements:

```
mysql> STOP SLAVE FOR CHANNEL 'channel_1';
mysql> CHANGE MASTER TO PRIVILEGE_CHECKS_USER = 'priv_repl'@'%.example.com' FOR CHANNEL 'channel_1';
mysql> START SLAVE FOR CHANNEL 'channel_1';
```

When you restart the replication channel, the privilege checks are applied from that point on. The user name and host name for the PRIVILEGE_CHECKS_USER account for a channel are shown in the Performance Schema replication_applier_configuration table, where they are properly escaped so they can be copied directly into SQL statements to execute individual transactions.

As well as securing asynchronous and semi-synchronous replication, you may choose to use a PRIVILEGE_CHECKS_USER account to secure the two replication applier threads used by Group Replication. The group_replication_applier thread on each group member is used for applying the group's transactions, and the group_replication_recovery thread on each group member is used for state transfer from the binary log as part of distributed recovery when the member joins or rejoins the group. To secure one of these threads, issue the CHANGE MASTER TO statement in the same way, specifying group_replication_applier or group_replication_recovery as the channel name.

By default, when a replication applier thread is started with a user account specified as the PRIVILEGE_CHECKS_USER, the security context is created using default roles, or with all roles if activate_all_roles_on_login is set to ON. You can use roles to supply a general privilege set to accounts that are used as PRIVILEGE_CHECKS_USER accounts, as in the following example, which grants the REPLICATION_APPLIER privilege together with the SESSION_VARIABLES_ADMIN privilege:

```
mysql> SET sql_log_bin = 0;
mysql> CREATE USER 'priv_repl'@'%.example.com' IDENTIFIED BY 'password' REQUIRE SSL;
mysql> CREATE ROLE 'priv_repl_role';
mysql> GRANT REPLICATION_APPLIER,SESSION_VARIABLES_ADMIN TO 'priv_repl_role';
mysql> GRANT 'priv_repl_role' TO 'priv_repl'@'%.example.com';
mysql> SET DEFAULT ROLE 'priv_repl_role' TO 'priv_repl'@'%.example.com';
mysql> SET sql_log_bin = 1;
```

If a remote cloning operation is used for distributed recovery in Group Replication (see Section 18.4.3.1, “Cloning for Distributed Recovery”), the PRIVILEGE_CHECKS_USER account is not automatically used on the joining member after cloning. You must assign the user account manually to the appropriate channels on the joining member.

17.3.3.1 Privileges For The Replication PRIVILEGE_CHECKS_USER Account

The user account that is specified as the PRIVILEGE_CHECKS_USER account for a replication channel must have the REPLICATION_APPLIER privilege, otherwise the replication applier thread does not start. As explained in Section 17.3.3, “Replication Privilege Checks”, the account requires further privileges that are sufficient to apply all the expected transactions expected on the replication channel. These privileges are checked only when relevant transactions are executed.

Important

The use of row-based binary logging (binlog_format=ROW) is strongly recommended for replication channels that are secured using a PRIVILEGE_CHECKS_USER account, and will be required in a future release. With statement-based binary logging, some administrator-level privileges are required to execute transactions successfully.

The REPLICATION_APPLIER privilege explicitly or implicitly allows the PRIVILEGE_CHECKS_USER account to carry out the following operations that a replication thread needs to perform:

- Setting the value of the system variables gtid_next, original_commit_timestamp, original_server_version, immediate_server_version, and pseudo_slave_mode, to apply appropriate metadata and behaviors when executing transactions.
- Executing internal-use BINLOG statements to apply mysqlbinlog output, provided that the account also has permission for the tables and operations in those statements.
- Updating the system tables mysql.gtid_executed, mysql.slave_relay_log_info, mysql.slave_worker_info, and mysql.slave_master_info, to update replication metadata. (If events access these tables explicitly for other purposes, you must grant the appropriate privileges on the tables.)
- Applying a binary log Table_map_log_event, which provides table metadata but does not make any database changes.

The PRIVILEGE_CHECKS_USER account needs the SESSION_VARIABLES_ADMIN privilege in order to change the value of certain system variables (pseudo_thread_id and sql_require_primary_key) for the duration of a session to carry out replication operations. This privilege also allows the account to apply mysqlbinlog output that was created using the --disable-log-bin option.

If table encryption is in use, the table_encryption_privilege_check system variable is set to ON, and the encryption setting for the tablespace involved in any event differs from the applying server's default encryption setting (specified by the default_table_encryption system variable), the PRIVILEGE_CHECKS_USER account needs the TABLE_ENCRYPTION_ADMIN privilege in order to override the default encryption setting. It is strongly recommended that you do not grant this privilege. Instead, ensure that the default encryption setting on a replication slave matches the encryption status of the tablespaces that it replicates, and that replication group members have the same default encryption setting, so that the privilege is not needed.

In order to execute specific replicated transactions from the relay log, or transactions from mysqlbinlog output as required, the PRIVILEGE_CHECKS_USER account must have the following privileges:

- For a row insertion logged in row format (which are logged as a Write_rows_log_event), the INSERT privilege on the relevant table.
- For a row update logged in row format (which are logged as an Update_rows_log_event), the UPDATE privilege on the relevant table.
- For a row deletion logged in row format (which are logged as a Delete_rows_log_event), the DELETE privilege on the relevant table.
- For a transaction control statement such as BEGIN or COMMIT or DML logged in statement format (which are logged as a Query_log_event), privileges to execute the statement contained in the event.

Note

If LOAD DATA INFILE operations need to be carried out on the replication channel, use row-based binary logging (binlog_format=ROW). LOAD DATA is considered unsafe for statement-based format, and the PRIVILEGE_CHECKS_USER account should not be given the required privilege to execute such a statement (the FILE privilege). For more information, see Section 17.5.1.19, “Replication and LOAD DATA”. The Format_description_log_event, which deletes any temporary files created by LOAD DATA events, is processed without privilege checks.

If the `init_slave` system variable is set to specify one or more SQL statements to be executed when the SQL thread starts, the `PRIVILEGE_CHECKS_USER` account must have the privileges needed to execute these statements.

It is recommended that you never give any ACL privileges to the `PRIVILEGE_CHECKS_USER` account, including `CREATE USER`, `CREATE ROLE`, `DROP ROLE`, and `GRANT OPTION`, and do not permit the account to update the `mysql.user` table. With these privileges, the account could be used to create or modify user accounts on the server. To avoid ACL statements issued on the master being replicated to the secured channel for execution (where they will fail in the absence of these privileges), you can issue `SET sql_log_bin = 0` before all ACL statements and `SET sql_log_bin = 1` after them, to omit the statements from the master's binary log. Alternatively, you can set a dedicated current database before executing all ACL statements, and use a replication filter (`--binlog-ignore-db`) to filter out this database on the slave.

17.3.3.2 Recovering From Failed Replication Privilege Checks

If a privilege check against the `PRIVILEGE_CHECKS_USER` account fails, the transaction is not executed and replication stops for the channel. Details of the error and the last applied transaction are recorded in the Performance Schema `replication_applier_status_by_worker` table. Follow this procedure to recover from the error:

1. Identify the replicated event that caused the error and verify whether or not the event is expected and from a trusted source. You can use `mysqlbinlog` to retrieve and display the events that were logged around the time of the error. For instructions to do this, see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

2. If the replicated event is not expected or is not from a known and trusted source, investigate the cause. If you can identify why the event took place and there are no security considerations, proceed to fix the error as described below.

3. If the `PRIVILEGE_CHECKS_USER` account should have been permitted to execute the transaction, but has been misconfigured, grant the missing privileges to the account and restart replication for the channel.

4. If the transaction needs to be executed and you have verified that it is trusted, but the `PRIVILEGE_CHECKS_USER` account should not have this privilege normally, you can grant the required privilege to the `PRIVILEGE_CHECKS_USER` account temporarily. After the replicated event has been applied, remove the privilege from the account, and take any necessary steps to ensure the event does not recur if it is avoidable.

5. If the transaction is an administrative action that should only have taken place on the master and not on the slave, or should only have taken place on a single replication group member, skip the transaction on the server or servers where it stopped replication, then issue `START SLAVE` to restart replication on the channel. To avoid the situation in future, you could issue such administrative statements with `SET sql_log_bin = 0` before them and `SET sql_log_bin = 1` after them, so that they are not logged on the master.

6. If the transaction is a DDL or DML statement that should not have taken place on either the master or the slave, skip the transaction on the server or servers where it stopped replication, undo the transaction manually on the server where it originally took place, then issue `START SLAVE` to restart replication.

To skip a transaction, if GTIDs are in use, commit an empty transaction that has the GTID of the failing transaction, for example:

```
SET GTID_NEXT='aaa-bbb-ccc-ddd:N';
BEGIN;
```

```
COMMIT;
SET GTID_NEXT='AUTOMATIC';
```

If GTIDs are not in use, issue a `SET GLOBAL sql_slave_skip_counter` statement to skip the event, as described in Section 13.4.2.5, “SET GLOBAL sql_slave_skip_counter Syntax”.

17.4 Replication Solutions

Replication can be used in many different environments for a range of purposes. This section provides general notes and advice on using replication for specific solution types.

For information on using replication in a backup environment, including notes on the setup, backup procedure, and files to back up, see Section 17.4.1, “Using Replication for Backups”.

For advice and tips on using different storage engines on the master and slaves, see Section 17.4.4, “Using Replication with Different Master and Slave Storage Engines”.

Using replication as a scale-out solution requires some changes in the logic and operation of applications that use the solution. See Section 17.4.5, “Using Replication for Scale-Out”.

For performance or data distribution reasons, you may want to replicate different databases to different replication slaves. See Section 17.4.6, “Replicating Different Databases to Different Slaves”

As the number of replication slaves increases, the load on the master can increase and lead to reduced performance (because of the need to replicate the binary log to each slave). For tips on improving your replication performance, including using a single secondary server as a replication master, see Section 17.4.7, “Improving Replication Performance”.

For guidance on switching masters, or converting slaves into masters as part of an emergency failover solution, see Section 17.4.8, “Switching Masters During Failover”.

For information on security measures specific to servers in a replication topology, see Section 17.3, “Replication Security”.

17.4.1 Using Replication for Backups

To use replication as a backup solution, replicate data from the master to a slave, and then back up the data slave. The slave can be paused and shut down without affecting the running operation of the master, so you can produce an effective snapshot of “live” data that would otherwise require the master to be shut down.

How you back up a database depends on its size and whether you are backing up only the data, or the data and the replication slave state so that you can rebuild the slave in the event of failure. There are therefore two choices:

- If you are using replication as a solution to enable you to back up the data on the master, and the size of your database is not too large, the `mysqldump` tool may be suitable. See Section 17.4.1.1, “Backing Up a Slave Using mysqldump”.
- For larger databases, where `mysqldump` would be impractical or inefficient, you can back up the raw data files instead. Using the raw data files option also means that you can back up the binary and relay logs that will enable you to recreate the slave in the event of a slave failure. For more information, see Section 17.4.1.2, “Backing Up Raw Data from a Slave”.

Another backup strategy, which can be used for either master or slave servers, is to put the server in a read-only state. The backup is performed against the read-only server, which then is changed back to its usual read/write operational status. See Section 17.4.1.3, “Backing Up a Master or Slave by Making It Read Only”.

17.4.1.1 Backing Up a Slave Using mysqldump

Using `mysqldump` to create a copy of a database enables you to capture all of the data in the database in a format that enables the information to be imported into another instance of MySQL Server (see Section 4.5.4, “`mysqldump` — A Database Backup Program”). Because the format of the information is SQL statements, the file can easily be distributed and applied to running servers in the event that you need access to the data in an emergency. However, if the size of your data set is very large, `mysqldump` may be impractical.

When using `mysqldump`, you should stop replication on the slave before starting the dump process to ensure that the dump contains a consistent set of data:

1. Stop the slave from processing requests. You can stop replication completely on the slave using `mysqladmin`:

   ```
   shell> mysqladmin stop-slave
   ```

   Alternatively, you can stop only the slave SQL thread to pause event execution:

   ```
   shell> mysql -e 'STOP SLAVE SQL_THREAD;'
   ```

   This enables the slave to continue to receive data change events from the master's binary log and store them in the relay logs using the I/O thread, but prevents the slave from executing these events and changing its data. Within busy replication environments, permitting the I/O thread to run during backup may speed up the catch-up process when you restart the slave SQL thread.

2. Run `mysqldump` to dump your databases. You may either dump all databases or select databases to be dumped. For example, to dump all databases:

   ```
   shell> mysqldump --all-databases > fulldb.dump
   ```

3. Once the dump has completed, start slave operations again:

   ```
   shell> mysqladmin start-slave
   ```

In the preceding example, you may want to add login credentials (user name, password) to the commands, and bundle the process up into a script that you can run automatically each day.

If you use this approach, make sure you monitor the slave replication process to ensure that the time taken to run the backup does not affect the slave's ability to keep up with events from the master. See Section 17.1.7.1, “Checking Replication Status”. If the slave is unable to keep up, you may want to add another slave and distribute the backup process. For an example of how to configure this scenario, see Section 17.4.6, “Replicating Different Databases to Different Slaves”.

17.4.1.2 Backing Up Raw Data from a Slave

To guarantee the integrity of the files that are copied, backing up the raw data files on your MySQL replication slave should take place while your slave server is shut down. If the MySQL server is still running, background tasks may still be updating the database files, particularly those involving storage engines with background processes such as `InnoDB`. With `InnoDB`, these problems should be resolved during crash recovery, but since the slave server can be shut down during the backup process without affecting the execution of the master it makes sense to take advantage of this capability.

To shut down the server and back up the files:

1. Shut down the slave MySQL server:

   ```
   shell> mysqladmin shutdown
   ```

2. Copy the data files. You can use any suitable copying or archive utility, including `cp`, `tar` or `WinZip`. For example, assuming that the data directory is located under the current directory, you can archive the entire directory as follows:

```
shell> tar cf /tmp/dbbackup.tar ./data
```

3. Start the MySQL server again. Under Unix:

```
shell> mysqld_safe &
```

Under Windows:

```
C:\> "C:\Program Files\MySQL\MySQL Server 8.0\bin\mysqld"
```

Normally you should back up the entire data directory for the slave MySQL server. If you want to be able to restore the data and operate as a slave (for example, in the event of failure of the slave), in addition to the data, you need to have the master info repository and relay log info repository, and the relay log files. These items are needed to resume replication after you restore the slave's data. If tables have been used for the master info and relay log info repositories (see Section 17.2.4, “Replication Relay and Status Logs”), which is the default in MySQL 8.0, these tables are backed up along with the data directory. If files have been used for the repositories, you must back these up separately. The relay log files must also be backed up separately if they have been placed in a different location to the data directory.

If you lose the relay logs but still have the `relay-log.info` file, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. This requires that the binary logs still exist on the master server.

If your slave is replicating `LOAD DATA` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication of any interrupted `LOAD DATA` operations. The location of this directory is the value of the `--slave-load-tmpdir` option. If the server was not started with that option, the directory location is the value of the `tmpdir` system variable.

17.4.1.3 Backing Up a Master or Slave by Making It Read Only

It is possible to back up either master or slave servers in a replication setup by acquiring a global read lock and manipulating the `read_only` system variable to change the read-only state of the server to be backed up:

1. Make the server read-only, so that it processes only retrievals and blocks updates.
2. Perform the backup.
3. Change the server back to its normal read/write state.

Note

The instructions in this section place the server to be backed up in a state that is safe for backup methods that get the data from the server, such as `mysqldump` (see Section 4.5.4, “`mysqldump` — A Database Backup Program”). You should not attempt to use these instructions to make a binary backup by copying files directly because the server may still have modified data cached in memory and not flushed to disk.

The following instructions describe how to do this for a master server and for a slave server. For both scenarios discussed here, suppose that you have the following replication setup:

- A master server M1
- A slave server S1 that has M1 as its master
- A client C1 connected to M1
- A client C2 connected to S1

In either scenario, the statements to acquire the global read lock and manipulate the `read_only` variable are performed on the server to be backed up and do not propagate to any slaves of that server.

Scenario 1: Backup with a Read-Only Master

Put the master M1 in a read-only state by executing these statements on it:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

While M1 is in a read-only state, the following properties are true:

- Requests for updates sent by C1 to M1 will block because the server is in read-only mode.
- Requests for query results sent by C1 to M1 will succeed.
- Making a backup on M1 is safe.
- Making a backup on S1 is not safe. This server is still running, and might be processing the binary log or update requests coming from client C2.

While M1 is read only, perform the backup. For example, you can use `mysqldump`.

After the backup operation on M1 completes, restore M1 to its normal operational state by executing these statements:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

Although performing the backup on M1 is safe (as far as the backup is concerned), it is not optimal for performance because clients of M1 are blocked from executing updates.

This strategy applies to backing up a master server in a replication setup, but can also be used for a single server in a nonreplication setting.

Scenario 2: Backup with a Read-Only Slave

Put the slave S1 in a read-only state by executing these statements on it:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

While S1 is in a read-only state, the following properties are true:

- The master M1 will continue to operate, so making a backup on the master is not safe.
- The slave S1 is stopped, so making a backup on the slave S1 is safe.

These properties provide the basis for a popular backup scenario: Having one slave busy performing a backup for a while is not a problem because it does not affect the entire network, and the system is still

running during the backup. In particular, clients can still perform updates on the master server, which remains unaffected by backup activity on the slave.

While S1 is read only, perform the backup. For example, you can use `mysqldump`.

After the backup operation on S1 completes, restore S1 to its normal operational state by executing these statements:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

After the slave is restored to normal operation, it again synchronizes to the master by catching up with any outstanding updates from the binary log of the master.

17.4.2 Handling an Unexpected Halt of a Replication Slave

In order for replication to be resilient to unexpected halts of the server (sometimes described as crash-safe) it must be possible for the slave to recover its state before halting. This section describes the impact of an unexpected halt of a slave during replication and how to configure a slave for the best chance of recovery to continue replication.

After an unexpected halt of a slave, upon restart the slave's SQL thread must recover which transactions have been executed already. The information required for recovery is stored in the slave's relay log info log. From MySQL 8.0, this log is created by default as an `InnoDB` table named `mysql.slave_relay_log_info` (with the system variable `relay_log_info_repository` set to the default of `TABLE`). By using this transactional storage engine the information is always recoverable upon restart.

Updates to the relay log info log table are committed together with the transactions, meaning that the slave's progress information recorded in that log is always consistent with what has been applied to the database, even in the event of an unexpected server halt. Previously, this information was stored by default in a file in the data directory that was updated after the transaction had been applied. This held the risk of losing synchrony with the master depending at which stage of processing a transaction the slave halted at, or even corruption of the file itself. The setting `relay_log_info_repository = FILE` is now deprecated, and will be removed in a future release. For further information on the slave logs, see Section 17.2.4, “Replication Relay and Status Logs”.

When the relay log info log is stored in the `mysql.slave_relay_log_info` table, DML transactions and also atomic DDL make the following three updates together, atomically:

- Apply the transaction on the database.
- Update the replication positions in the `mysql.slave_relay_log_info` table.
- Update the GTID in the mysql.gtid_executed table (when GTIDs are enabled and the binary log is disabled on the server).

In all other cases, including DDL statements that are not fully atomic, and exempted storage engines that do not support atomic DDL, the `mysql.slave_relay_log_info` table might be missing updates associated with replicated data if the server halts unexpectedly. Restoring updates in this case is a manual process. For details on atomic DDL support in MySQL 8.0, and the resulting behavior for the replication of certain statements, see Section 13.1.1, “Atomic Data Definition Statement Support”.

Exactly how a replication slave recovers from an unexpected halt is influenced by the chosen method of replication, whether the slave is single-threaded or multithreaded, the setting of variables such as `relay_log_recovery`, and whether features such as `MASTER_AUTO_POSITION` are being used.

The following table shows the impact of these different factors on how a single-threaded slave recovers from an unexpected halt.

Table 17.3 Factors Influencing Single-threaded Replication Slave Recovery

GTIDs and MASTER_AUTO_POSITION	relay_log_recovery	relay_log_info_repository	Transactions guaranteed to be recovered	Impact

As the table shows, when using a single-threaded slave the following configurations are most resilient to unexpected halts:

- When using GTIDs and `MASTER_AUTO_POSITION`, set `relay_log_recovery=1`. With this configuration the setting of `relay_log_info_repository` and other variables does not impact on recovery. Note that to guarantee recovery, `sync_binlog=1` (which is the default) must also be set on the slave, so that the slave's binary log is synchronized to disk at each write. Otherwise, committed transactions might not be present in the slave's binary log.
- When using file position based replication, set `relay_log_recovery=1` and `relay_log_info_repository=TABLE`.

Note

During recovery the relay log is lost.

The following table shows the impact of these different factors on how a multithreaded slave recovers from an unexpected halt.

Table 17.4 Factors Influencing Multithreaded Replication Slave Recovery

GTIDs and MASTER_AUTO_POSITION	relay_log_recovery	relay_log_info_repository	Transactions guaranteed to be recovered	Impact

As the table shows, when using a multithreaded slave the following configurations are most resilient to unexpected halts:

- When using GTIDs and `MASTER_AUTO_POSITION=ON`, set `relay_log_recovery=1`. With this configuration the setting of `relay_log_info_repository` and other variables does not impact on recovery. From MySQL 8.0.18, a multithreaded slave automatically skips relay log recovery when `MASTER_AUTO_POSITION` is set to `ON`, so the setting for `relay_log_recovery` makes no difference.
- When using file position based replication, set `relay_log_recovery=1`, `sync_relay_log=1`, and `relay_log_info_repository=TABLE`.

Note

During recovery the relay log is lost.

It is important to note the impact of `sync_relay_log=1`, which requires a write of to the relay log per transaction. Although this setting is the most resilient to an unexpected halt, with at most one unwritten transaction being lost, it also has the potential to greatly increase the load on storage. Without `sync_relay_log=1`, the effect of an unexpected halt depends on how the relay log is handled by the operating system. Also note that when `relay_log_recovery=0`, the next time the slave is started after an unexpected halt the relay log is processed as part of recovery. After this process completes, the relay log is deleted.

An unexpected halt of a multithreaded replication slave using the recommended file position based replication configuration above may result in a relay log with transaction inconsistencies (gaps in the sequence of transactions) caused by the unexpected halt. See Section 17.5.1.33, “Replication and Transaction Inconsistencies”. If the relay log recovery process encounters such transaction inconsistencies they are filled and the recovery process continues automatically.

When you are using multi-source replication and `relay_log_recovery=1`, after restarting due to an unexpected halt all replication channels go through the relay log recovery process. Any inconsistencies found in the relay log due to an unexpected halt of a multithreaded slave are filled.

17.4.3 Monitoring Row-based Replication

The current progress of the replication applier (SQL) thread when using row-based replication is monitored through Performance Schema instrument stages, enabling you to track the processing of operations and check the amount of work completed and work estimated. When these Performance Schema instrument stages are enabled the `events_stages_current` table shows stages for applier threads and their progress. For background information, see Section 26.12.5, “Performance Schema Stage Event Tables”.

To track progress of all three row-based replication event types (write, update, delete):

- Enable the three Performance Schema stages by issuing:

```
mysql> UPDATE performance_schema.setup_instruments SET ENABLED = 'YES'
    -> WHERE NAME LIKE 'stage/sql/Applying batch of row changes%';
```

- Wait for some events to be processed by the replication applier thread and then check progress by looking into the `events_stages_current` table. For example to get progress for `update` events issue:

```
mysql> SELECT WORK_COMPLETED, WORK_ESTIMATED FROM performance_schema.events_stages_current
    -> WHERE EVENT_NAME LIKE 'stage/sql/Applying batch of row changes (update)'
```

- If `binlog_rows_query_log_events` is enabled, information about queries is stored in the binary log and is exposed in the `processlist_info` field. To see the original query that triggered this event:

```
mysql> SELECT db, processlist_state, processlist_info FROM performance_schema.threads
    -> WHERE processlist_state LIKE 'stage/sql/Applying batch of row changes%' AND thread_id = N;
```

17.4.4 Using Replication with Different Master and Slave Storage Engines

It does not matter for the replication process whether the source table on the master and the replicated table on the slave use different engine types. In fact, the `default_storage_engine` system variable is not replicated.

This provides a number of benefits in the replication process in that you can take advantage of different engine types for different replication scenarios. For example, in a typical scale-out scenario (see Section 17.4.5, “Using Replication for Scale-Out”), you want to use `InnoDB` tables on the master to take advantage of the transactional functionality, but use `MyISAM` on the slaves where transaction support is not required because the data is only read. When using replication in a data-logging environment you may want to use the `Archive` storage engine on the slave.

Configuring different engines on the master and slave depends on how you set up the initial replication process:

- If you used `mysqldump` to create the database snapshot on your master, you could edit the dump file text to change the engine type used on each table.

  Another alternative for `mysqldump` is to disable engine types that you do not want to use on the slave before using the dump to build the data on the slave. For example, you can add the `--skip-federated` option on your slave to disable the `FEDERATED` engine. If a specific engine does not exist for a table to be created, MySQL will use the default engine type, usually `MyISAM`. (This requires that the `NO_ENGINE_SUBSTITUTION` SQL mode is not enabled.) If you want to disable additional engines in this way, you may want to consider building a special binary to be used on the slave that only supports the engines you want.

- If you are using raw data files (a binary backup) to set up the slave, you will be unable to change the initial table format. Instead, use `ALTER TABLE` to change the table types after the slave has been started.

- For new master/slave replication setups where there are currently no tables on the master, avoid specifying the engine type when creating new tables.

If you are already running a replication solution and want to convert your existing tables to another engine type, follow these steps:

1. Stop the slave from running replication updates:

   ```
   mysql> STOP SLAVE;
   ```

   This will enable you to change engine types without interruptions.

2. Execute an `ALTER TABLE ... ENGINE='engine_type'` for each table to be changed.

3. Start the slave replication process again:

   ```
   mysql> START SLAVE;
   ```

Although the `default_storage_engine` variable is not replicated, be aware that `CREATE TABLE` and `ALTER TABLE` statements that include the engine specification will be correctly replicated to the slave. For example, if you have a CSV table and you execute:

```
mysql> ALTER TABLE csvtable Engine='MyISAM';
```

The above statement will be replicated to the slave and the engine type on the slave will be converted to `MyISAM`, even if you have previously changed the table type on the slave to an engine other than CSV. If you want to retain engine differences on the master and slave, you should be careful to use the `default_storage_engine` variable on the master when creating a new table. For example, instead of:

```
mysql> CREATE TABLE tablea (columna int) Engine=MyISAM;
```

Use this format:

```
mysql> SET default_storage_engine=MyISAM;
mysql> CREATE TABLE tablea (columna int);
```

When replicated, the `default_storage_engine` variable will be ignored, and the `CREATE TABLE` statement will execute on the slave using the slave's default engine.

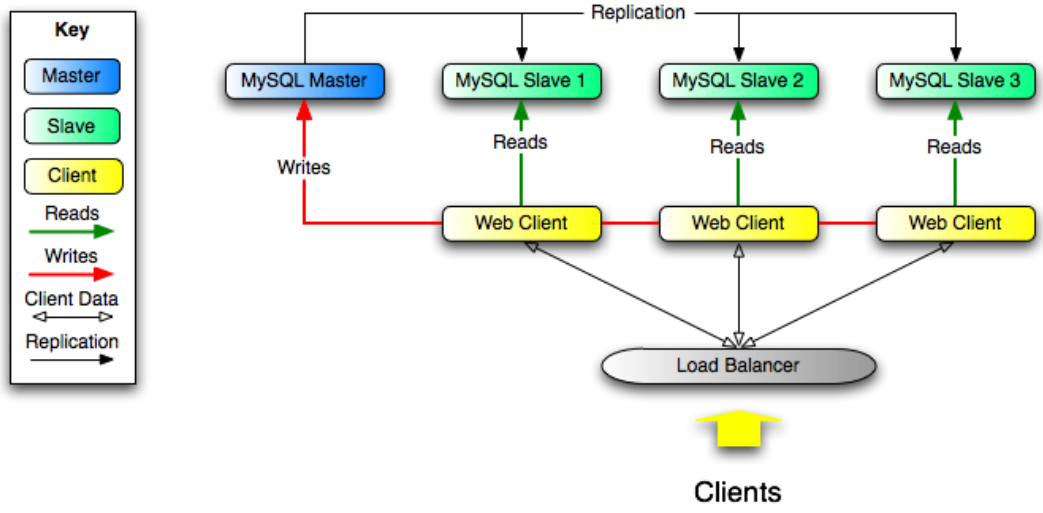
17.4.5 Using Replication for Scale-Out

You can use replication as a scale-out solution; that is, where you want to split up the load of database queries across multiple database servers, within some reasonable limitations.

Because replication works from the distribution of one master to one or more slaves, using replication for scale-out works best in an environment where you have a high number of reads and low number of writes/updates. Most websites fit into this category, where users are browsing the website, reading articles, posts, or viewing products. Updates only occur during session management, or when making a purchase or adding a comment/message to a forum.

Replication in this situation enables you to distribute the reads over the replication slaves, while still enabling your web servers to communicate with the replication master when a write is required. You can see a sample replication layout for this scenario in Figure 17.1, “Using Replication to Improve Performance During Scale-Out”.

Figure 17.1 Using Replication to Improve Performance During Scale-Out



If the part of your code that is responsible for database access has been properly abstracted/modularized, converting it to run with a replicated setup should be very smooth and easy. Change the implementation of

your database access to send all writes to the master, and to send reads to either the master or a slave. If your code does not have this level of abstraction, setting up a replicated system gives you the opportunity and motivation to clean it up. Start by creating a wrapper library or module that implements the following functions:

- `safe_writer_connect()`
- `safe_reader_connect()`
- `safe_reader_statement()`
- `safe_writer_statement()`

`safe_` in each function name means that the function takes care of handling all error conditions. You can use different names for the functions. The important thing is to have a unified interface for connecting for reads, connecting for writes, doing a read, and doing a write.

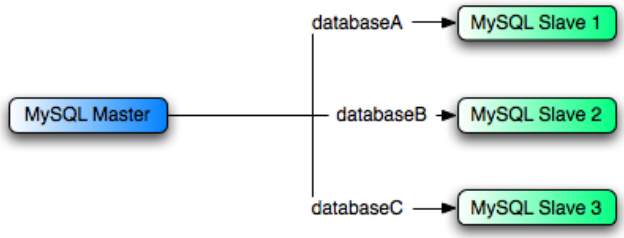
Then convert your client code to use the wrapper library. This may be a painful and scary process at first, but it pays off in the long run. All applications that use the approach just described are able to take advantage of a master/slave configuration, even one involving multiple slaves. The code is much easier to maintain, and adding troubleshooting options is trivial. You need modify only one or two functions (for example, to log how long each statement took, or which statement among those issued gave you an error).

If you have written a lot of code, you may want to automate the conversion task by writing a conversion script. Ideally, your code uses consistent programming style conventions. If not, then you are probably better off rewriting it anyway, or at least going through and manually regularizing it to use a consistent style.

17.4.6 Replicating Different Databases to Different Slaves

There may be situations where you have a single master and want to replicate different databases to different slaves. For example, you may want to distribute different sales data to different departments to help spread the load during data analysis. A sample of this layout is shown in Figure 17.2, “Using Replication to Replicate Databases to Separate Replication Slaves”.

Figure 17.2 Using Replication to Replicate Databases to Separate Replication Slaves



You can achieve this separation by configuring the master and slaves as normal, and then limiting the binary log statements that each slave processes by using the `--replicate-wild-do-table` configuration option on each slave.

Important

You should *not* use `--replicate-do-db` for this purpose when using statement-based replication, since statement-based replication causes this option's affects

to vary according to the database that is currently selected. This applies to mixed-format replication as well, since this enables some updates to be replicated using the statement-based format.

However, it should be safe to use `--replicate-do-db` for this purpose if you are using row-based replication only, since in this case the currently selected database has no effect on the option's operation.

For example, to support the separation as shown in Figure 17.2, “Using Replication to Replicate Databases to Separate Replication Slaves”, you should configure each replication slave as follows, before executing `START SLAVE`:

- Replication slave 1 should use `--replicate-wild-do-table=databaseA.%`.
- Replication slave 2 should use `--replicate-wild-do-table=databaseB.%`.
- Replication slave 3 should use `--replicate-wild-do-table=databaseC.%`.

Each slave in this configuration receives the entire binary log from the master, but executes only those events from the binary log that apply to the databases and tables included by the `--replicate-wild-do-table` option in effect on that slave.

If you have data that must be synchronized to the slaves before replication starts, you have a number of choices:

- Synchronize all the data to each slave, and delete the databases, tables, or both that you do not want to keep.
- Use `mysqldump` to create a separate dump file for each database and load the appropriate dump file on each slave.
- Use a raw data file dump and include only the specific files and databases that you need for each slave.

Note

This does not work with `InnoDB` databases unless you use `innodb_file_per_table`.

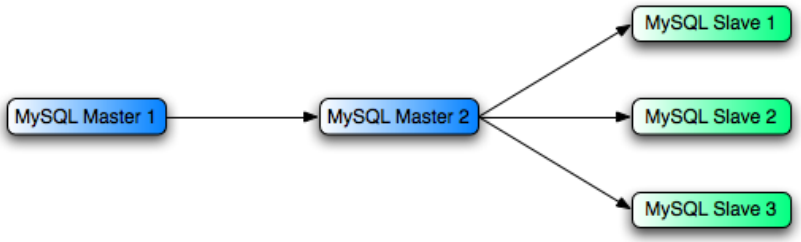
17.4.7 Improving Replication Performance

As the number of slaves connecting to a master increases, the load, although minimal, also increases, as each slave uses a client connection to the master. Also, as each slave must receive a full copy of the master binary log, the network load on the master may also increase and create a bottleneck.

If you are using a large number of slaves connected to one master, and that master is also busy processing requests (for example, as part of a scale-out solution), then you may want to improve the performance of the replication process.

One way to improve the performance of the replication process is to create a deeper replication structure that enables the master to replicate to only one slave, and for the remaining slaves to connect to this primary slave for their individual replication requirements. A sample of this structure is shown in Figure 17.3, “Using an Additional Replication Host to Improve Performance”.

Figure 17.3 Using an Additional Replication Host to Improve Performance



For this to work, you must configure the MySQL instances as follows:

- Master 1 is the primary master where all changes and updates are written to the database. Binary logging is enabled on both masters, which is the default.
- Master 2 is the slave to the Master 1 that provides the replication functionality to the remainder of the slaves in the replication structure. Master 2 is the only machine permitted to connect to Master 1. Master 2 has the `--log-slave-updates` option enabled (which is the default). With this option, replication instructions from Master 1 are also written to Master 2's binary log so that they can then be replicated to the true slaves.
- Slave 1, Slave 2, and Slave 3 act as slaves to Master 2, and replicate the information from Master 2, which actually consists of the upgrades logged on Master 1.

The above solution reduces the client load and the network interface load on the primary master, which should improve the overall performance of the primary master when used as a direct database solution.

If your slaves are having trouble keeping up with the replication process on the master, there are a number of options available:

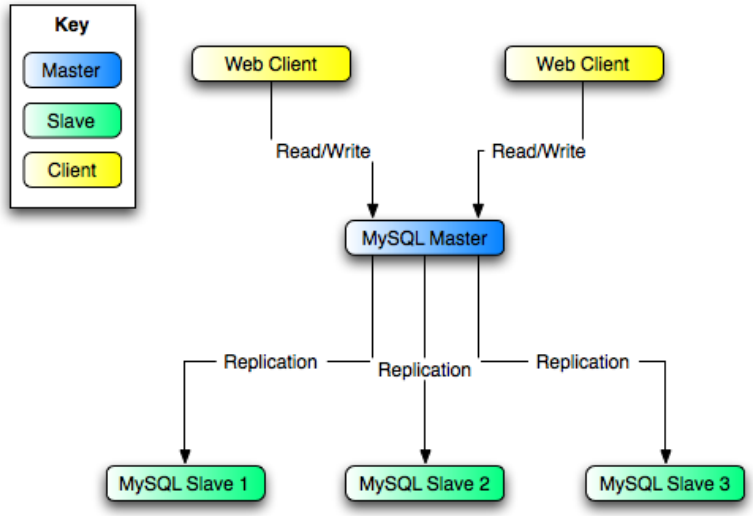
- If possible, put the relay logs and the data files on different physical drives. To do this, set the `relay_log` system variable to specify the location of the relay log.
- If heavy disk I/O activity for reads of the binary log file and relay log files is an issue, consider increasing the value of the `rpl_read_size` system variable. This system variable controls the minimum amount of data read from the log files, and increasing it might reduce file reads and I/O stalls when the file data is not currently cached by the operating system. Note that a buffer the size of this value is allocated for each thread that reads from the binary log and relay log files, including dump threads on masters and coordinator threads on slaves. Setting a large value might therefore have an impact on memory consumption for servers.
- If the slaves are significantly slower than the master, you may want to divide up the responsibility for replicating different databases to different slaves. See Section 17.4.6, “Replicating Different Databases to Different Slaves”.
- If your master makes use of transactions and you are not concerned about transaction support on your slaves, use `MyISAM` or another nontransactional engine on the slaves. See Section 17.4.4, “Using Replication with Different Master and Slave Storage Engines”.
- If your slaves are not acting as masters, and you have a potential solution in place to ensure that you can bring up a master in the event of failure, then you can disable the `log_slave_updates` system variable on the slaves. This prevents “dumb” slaves from also logging events they have executed into their own binary log.

17.4.8 Switching Masters During Failover

You can tell a slave to change to a new master using the `CHANGE MASTER TO` statement. The slave does not check whether the databases on the master are compatible with those on the slave; it simply begins reading and executing events from the specified coordinates in the new master's binary log. In a failover situation, all the servers in the group are typically executing the same events from the same binary log file, so changing the source of the events should not affect the structure or integrity of the database, provided that you exercise care in making the change.

Slaves should be run with binary logging enabled (the `--log-bin` option), which is the default. If you are not using GTIDs for replication, then the slaves should also be run with `--log-slave-updates=OFF` (logging slave updates is the default). In this way, the slave is ready to become a master without restarting the slave `mysqld`. Assume that you have the structure shown in Figure 17.4, “Redundancy Using Replication, Initial Structure”.

Figure 17.4 Redundancy Using Replication, Initial Structure



In this diagram, the `MySQL Master` holds the master database, the `MySQL Slave` hosts are replication slaves, and the `Web Client` machines are issuing database reads and writes. Web clients that issue only reads (and would normally be connected to the slaves) are not shown, as they do not need to switch to a new server in the event of failure. For a more detailed example of a read/write scale-out replication structure, see Section 17.4.5, “Using Replication for Scale-Out”.

Each MySQL Slave (`Slave 1`, `Slave 2`, and `Slave 3`) is a slave running with binary logging enabled, and with `--log-slave-updates=OFF`. Because updates received by a slave from the master are not logged in the binary log when `--log-slave-updates=OFF` is specified, the binary log on each slave is empty initially. If for some reason `MySQL Master` becomes unavailable, you can pick one of the slaves to become the new master. For example, if you pick `Slave 1`, all `Web Clients` should be redirected to `Slave 1`, which writes the updates to its binary log. `Slave 2` and `Slave 3` should then replicate from `Slave 1`.

The reason for running the slave with `--log-slave-updates=OFF` is to prevent slaves from receiving updates twice in case you cause one of the slaves to become the new master. If `Slave 1` has `--log-slave-updates` enabled, which is the default, it writes any updates that it receives from `Master` in its own binary log. This means that, when `Slave 2` changes from `Master` to `Slave 1` as its master, it may receive updates from `Slave 1` that it has already received from `Master`.

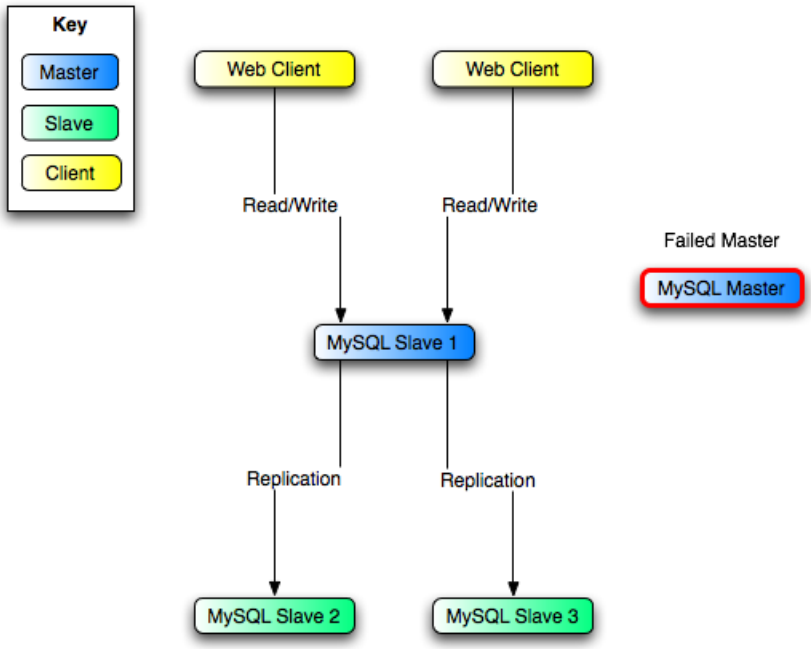
Make sure that all slaves have processed any statements in their relay log. On each slave, issue `STOP SLAVE IO_THREAD`, then check the output of `SHOW PROCESSLIST` until you see `Has read all relay log`. When this is true for all slaves, they can be reconfigured to the new setup. On the slave `Slave 1` being promoted to become the master, issue `STOP SLAVE` and `RESET MASTER`.

On the other slaves `Slave 2` and `Slave 3`, use `STOP SLAVE` and `CHANGE MASTER TO MASTER_HOST='Slave1'` (where `'Slave1'` represents the real host name of `Slave 1`). To use `CHANGE MASTER TO`, add all information about how to connect to `Slave 1` from `Slave 2` or `Slave 3` (*`user`*, *`password`*, *`port`*). When issuing the `CHANGE MASTER TO` statement in this, there is no need to specify the name of the `Slave 1` binary log file or log position to read from, since the first binary log file and position 4, are the defaults. Finally, execute `START SLAVE` on `Slave 2` and `Slave 3`.

Once the new replication setup is in place, you need to tell each `Web Client` to direct its statements to `Slave 1`. From that point on, all update statements sent by `Web Client` to `Slave 1` are written to the binary log of `Slave 1`, which then contains every update statement sent to `Slave 1` since `Master` died.

The resulting server structure is shown in Figure 17.5, “Redundancy Using Replication, After Master Failure”.

Figure 17.5 Redundancy Using Replication, After Master Failure



When `Master` becomes available again, you should make it a slave of `Slave 1`. To do this, issue on `Master` the same `CHANGE MASTER TO` statement as that issued on `Slave 2` and `Slave 3` previously. `Master` then becomes a slave of `S1ave 1` and picks up the `Web Client` writes that it missed while it was offline.

To make `Master` a master again, use the preceding procedure as if `Slave 1` was unavailable and `Master` was to be the new master. During this procedure, do not forget to run `RESET MASTER` on `Master` before making `Slave 1`, `Slave 2`, and `Slave 3` slaves of `Master`. If you fail to do this, the slaves may pick up stale writes from the `Web Client` applications dating from before the point at which `Master` became unavailable.

You should be aware that there is no synchronization between slaves, even when they share the same master, and thus some slaves might be considerably ahead of others. This means that in some cases the

procedure outlined in the previous example might not work as expected. In practice, however, relay logs on all slaves should be relatively close together.

One way to keep applications informed about the location of the master is to have a dynamic DNS entry for the master. With `bind` you can use `nsupdate` to update the DNS dynamically.

17.4.9 Semisynchronous Replication

In addition to the built-in asynchronous replication, MySQL 8.0 supports an interface to semisynchronous replication that is implemented by plugins. This section discusses what semisynchronous replication is and how it works. The following sections cover the administrative interface to semisynchronous replication and how to install, configure, and monitor it.

MySQL replication by default is asynchronous. The master writes events to its binary log but does not know whether or when a slave has retrieved and processed them. With asynchronous replication, if the master crashes, transactions that it has committed might not have been transmitted to any slave. Consequently, failover from master to slave in this case may result in failover to a server that is missing transactions relative to the master.

Semisynchronous replication can be used as an alternative to asynchronous replication:

- A slave indicates whether it is semisynchronous-capable when it connects to the master.
- If semisynchronous replication is enabled on the master side and there is at least one semisynchronous slave, a thread that performs a transaction commit on the master blocks and waits until at least one semisynchronous slave acknowledges that it has received all events for the transaction, or until a timeout occurs.
- The slave acknowledges receipt of a transaction's events only after the events have been written to its relay log and flushed to disk.
- If a timeout occurs without any slave having acknowledged the transaction, the master reverts to asynchronous replication. When at least one semisynchronous slave catches up, the master returns to semisynchronous replication.
- Semisynchronous replication must be enabled on both the master and slave sides. If semisynchronous replication is disabled on the master, or enabled on the master but on no slaves, the master uses asynchronous replication.

While the master is blocking (waiting for acknowledgment from a slave), it does not return to the session that performed the transaction. When the block ends, the master returns to the session, which then can proceed to execute other statements. At this point, the transaction has committed on the master side, and receipt of its events has been acknowledged by at least one slave.

The number of slave acknowledgments the master must receive per transaction before proceeding is configurable using the `rpl_semi_sync_master_wait_for_slave_count` system variable. The default value is 1.

Blocking also occurs after rollbacks that are written to the binary log, which occurs when a transaction that modifies nontransactional tables is rolled back. The rolled-back transaction is logged even though it has no effect for transactional tables because the modifications to the nontransactional tables cannot be rolled back and must be sent to slaves.

For statements that do not occur in transactional context (that is, when no transaction has been started with `START TRANSACTION` or `SET autocommit = 0`), autocommit is enabled and each statement commits implicitly. With semisynchronous replication, the master blocks for each such statement, just as it does for explicit transaction commits.

To understand what the “semi” in “semisynchronous replication” means, compare it with asynchronous and fully synchronous replication:

- With asynchronous replication, the master writes events to its binary log and slaves request them when they are ready. There is no guarantee that any event will ever reach any slave.
- With fully synchronous replication, when a master commits a transaction, all slaves also will have committed the transaction before the master returns to the session that performed the transaction. The drawback of this is that there might be a lot of delay to complete a transaction.
- Semisynchronous replication falls between asynchronous and fully synchronous replication. The master waits only until at least one slave has received and logged the events. It does not wait for all slaves to acknowledge receipt, and it requires only receipt, not that the events have been fully executed and committed on the slave side.

Compared to asynchronous replication, semisynchronous replication provides improved data integrity because when a commit returns successfully, it is known that the data exists in at least two places. Until a semisynchronous master receives acknowledgment from the number of slaves configured by `rpl_semi_sync_master_wait_for_slave_count`, the transaction is on hold and not committed.

Semisynchronous replication also places a rate limit on busy sessions by constraining the speed at which binary log events can be sent from master to slave. When one user is too busy, this will slow it down, which is useful in some deployment situations.

Semisynchronous replication does have some performance impact because commits are slower due to the need to wait for slaves. This is the tradeoff for increased data integrity. The amount of slowdown is at least the TCP/IP roundtrip time to send the commit to the slave and wait for the acknowledgment of receipt by the slave. This means that semisynchronous replication works best for close servers communicating over fast networks, and worst for distant servers communicating over slow networks.

The `rpl_semi_sync_master_wait_point` system variable controls the point at which a semisynchronous replication master waits for slave acknowledgment of transaction receipt before returning a status to the client that committed the transaction. These values are permitted:

- `AFTER_SYNC` (the default): The master writes each transaction to its binary log and the slave, and syncs the binary log to disk. The master waits for slave acknowledgment of transaction receipt after the sync. Upon receiving acknowledgment, the master commits the transaction to the storage engine and returns a result to the client, which then can proceed.
- `AFTER_COMMIT`: The master writes each transaction to its binary log and the slave, syncs the binary log, and commits the transaction to the storage engine. The master waits for slave acknowledgment of transaction receipt after the commit. Upon receiving acknowledgment, the master returns a result to the client, which then can proceed.

The replication characteristics of these settings differ as follows:

- With `AFTER_SYNC`, all clients see the committed transaction at the same time: After it has been acknowledged by the slave and committed to the storage engine on the master. Thus, all clients see the same data on the master.

  In the event of master failure, all transactions committed on the master have been replicated to the slave (saved to its relay log). A crash of the master and failover to the slave is lossless because the slave is up to date.
- With `AFTER_COMMIT`, the client issuing the transaction gets a return status only after the server commits to the storage engine and receives slave acknowledgment. After the commit and before slave acknowledgment, other clients can see the committed transaction before the committing client.

If something goes wrong such that the slave does not process the transaction, then in the event of a master crash and failover to the slave, it is possible that such clients will see a loss of data relative to what they saw on the master.

17.4.9.1 Semisynchronous Replication Administrative Interface

The administrative interface to semisynchronous replication has several components:

- Two plugins implement semisynchronous capability. There is one plugin for the master side and one for the slave side.
- System variables control plugin behavior. Some examples:
  - `rpl_semi_sync_master_enabled`

    Controls whether semisynchronous replication is enabled on the master. To enable or disable the plugin, set this variable to 1 or 0, respectively. The default is 0 (off).
  - `rpl_semi_sync_master_timeout`

    A value in milliseconds that controls how long the master waits on a commit for acknowledgment from a slave before timing out and reverting to asynchronous replication. The default value is 10000 (10 seconds).
  - `rpl_semi_sync_slave_enabled`

    Similar to `rpl_semi_sync_master_enabled`, but controls the slave plugin.

  All `rpl_semi_sync_`*`xxx`* system variables are described at Section 5.1.8, “Server System Variables”.
- Status variables enable semisynchronous replication monitoring. Some examples:
  - `Rpl_semi_sync_master_clients`

    The number of semisynchronous slaves.
  - `Rpl_semi_sync_master_status`

    Whether semisynchronous replication currently is operational on the master. The value is 1 if the plugin has been enabled and a commit acknowledgment has not occurred. It is 0 if the plugin is not enabled or the master has fallen back to asynchronous replication due to commit acknowledgment timeout.
  - `Rpl_semi_sync_master_no_tx`

    The number of commits that were not acknowledged successfully by a slave.
  - `Rpl_semi_sync_master_yes_tx`

    The number of commits that were acknowledged successfully by a slave.
  - `Rpl_semi_sync_slave_status`

    Whether semisynchronous replication currently is operational on the slave. This is 1 if the plugin has been enabled and the slave I/O thread is running, 0 otherwise.

  All `Rpl_semi_sync_`*`xxx`* status variables are described at Section 5.1.10, “Server Status Variables”.

The system and status variables are available only if the appropriate master or slave plugin has been installed with `INSTALL PLUGIN`.

17.4.9.2 Semisynchronous Replication Installation and Configuration

Semisynchronous replication is implemented using plugins, so the plugins must be installed into the server to make them available. After a plugin has been installed, you control it by means of the system variables associated with it. These system variables are unavailable until the associated plugin has been installed.

This section describes how to install the semisynchronous replication plugins. For general information about installing plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To use semisynchronous replication, the following requirements must be satisfied:

- The capability of installing plugins requires a MySQL server that supports dynamic loading. To verify this, check that the value of the `have_dynamic_loading` system variable is `YES`. Binary distributions should support dynamic loading.
- Replication must already be working, see Section 17.1, “Configuring Replication”.
- There must not be multiple replication channels configured. Semisynchronous replication is only compatible with the default replication channel. See Section 17.2.3, “Replication Channels”.

To set up semisynchronous replication, use the following instructions. The `INSTALL PLUGIN`, `SET GLOBAL`, `STOP SLAVE`, and `START SLAVE` statements mentioned here require the `REPLICATION_SLAVE_ADMIN` or `SUPER` privilege.

MySQL distributions include semisynchronous replication plugin files for the master side and the slave side.

To be usable by a master or slave server, the appropriate plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base names are `semisync_master` and `semisync_slave`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

The master plugin library file must be present in the plugin directory of the master server. The slave plugin library file must be present in the plugin directory of each slave server.

To load the plugins, use the `INSTALL PLUGIN` statement on the master and on each slave that is to be semisynchronous (adjust the `.so` suffix for your platform as necessary).

On the master:

```
INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
```

On each slave:

```
INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

If an attempt to install a plugin results in an error on Linux similar to that shown here, you must install `libimf`:

```
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
ERROR 1126 (HY000): Can't open shared library
'/usr/local/mysql/lib/plugin/semisync_master.so'
(errno: 22 libimf.so: cannot open shared object file:
```

```
No such file or directory)
```

You can obtain `libimf` from https://dev.mysql.com/downloads/os-linux.html.

To see which plugins are installed, use the `SHOW PLUGINS` statement, or query the `INFORMATION_SCHEMA.PLUGINS` table.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%semi%';
+----------------------+---------------+
| PLUGIN_NAME          | PLUGIN_STATUS |
+----------------------+---------------+
| rpl_semi_sync_master | ACTIVE        |
+----------------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

After a semisynchronous replication plugin has been installed, it is disabled by default. The plugins must be enabled both on the master side and the slave side to enable semisynchronous replication. If only one side is enabled, replication will be asynchronous.

To control whether an installed plugin is enabled, set the appropriate system variables. You can set these variables at runtime using `SET GLOBAL`, or at server startup on the command line or in an option file.

At runtime, these master-side system variables are available:

```
SET GLOBAL rpl_semi_sync_master_enabled = {0|1};
SET GLOBAL rpl_semi_sync_master_timeout = N;
```

On the slave side, this system variable is available:

```
SET GLOBAL rpl_semi_sync_slave_enabled = {0|1};
```

For `rpl_semi_sync_master_enabled` or `rpl_semi_sync_slave_enabled`, the value should be 1 to enable semisynchronous replication or 0 to disable it. By default, these variables are set to 0.

For `rpl_semi_sync_master_timeout`, the value *`N`* is given in milliseconds. The default value is 10000 (10 seconds).

If you enable semisynchronous replication on a slave at runtime, you must also start the slave I/O thread (stopping it first if it is already running) to cause the slave to connect to the master and register as a semisynchronous slave:

```
STOP SLAVE IO_THREAD;
START SLAVE IO_THREAD;
```

If the I/O thread is already running and you do not restart it, the slave continues to use asynchronous replication.

At server startup, the variables that control semisynchronous replication can be set as command-line options or in an option file. A setting listed in an option file takes effect each time the server starts. For example, you can set the variables in `my.cnf` files on the master and slave sides as follows.

On the master:

```
[mysqld]
rpl_semi_sync_master_enabled=1
rpl_semi_sync_master_timeout=1000 # 1 second
```

On each slave:

```
[mysqld]
rpl_semi_sync_slave_enabled=1
```

17.4.9.3 Semisynchronous Replication Monitoring

The plugins for the semisynchronous replication capability expose several system and status variables that you can examine to determine its configuration and operational state.

The system variable reflect how semisynchronous replication is configured. To check their values, use `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'rpl_semi_sync%';
```

The status variables enable you to monitor the operation of semisynchronous replication. To check their values, use `SHOW STATUS`:

```
mysql> SHOW STATUS LIKE 'Rpl_semi_sync%';
```

When the master switches between asynchronous or semisynchronous replication due to commit-blocking timeout or a slave catching up, it sets the value of the `Rpl_semi_sync_master_status` status variable appropriately. Automatic fallback from semisynchronous to asynchronous replication on the master means that it is possible for the `rpl_semi_sync_master_enabled` system variable to have a value of 1 on the master side even when semisynchronous replication is in fact not operational at the moment. You can monitor the `Rpl_semi_sync_master_status` status variable to determine whether the master currently is using asynchronous or semisynchronous replication.

To see how many semisynchronous slaves are connected, check `Rpl_semi_sync_master_clients`.

The number of commits that have been acknowledged successfully or unsuccessfully by slaves are indicated by the `Rpl_semi_sync_master_yes_tx` and `Rpl_semi_sync_master_no_tx` variables.

On the slave side, `Rpl_semi_sync_slave_status` indicates whether semisynchronous replication currently is operational.

17.4.10 Delayed Replication

MySQL supports delayed replication such that a slave server deliberately executes transactions later than the master by at least a specified amount of time. This section describes how to configure a replication delay on a slave, and how to monitor replication delay.

In MySQL 8.0, the method of delaying replication depends on two timestamps, `immediate_commit_timestamp` and `original_commit_timestamp` (see Replication Delay Timestamps). If all servers in the replication topology are running MySQL 8.0.1 or above, delayed replication is measured using these timestamps. If either the immediate master or slave is not using these timestamps, the implementation of delayed replication from MySQL 5.7 is used (see Delayed Replication). This section describes delayed replication between servers which are all using these timestamps.

The default replication delay is 0 seconds. Use the `CHANGE MASTER TO MASTER_DELAY=N` statement to set the delay to *`N`* seconds. A transaction received from the master is not executed until at least

N seconds later than its commit on the immediate master. The delay happens per transaction (not event as in previous MySQL versions) and the actual delay is imposed only on `gtid_log_event` or `anonymous_gtid_log_event`. The other events in the transaction always follow these events without any waiting time imposed on them.

Note

`START SLAVE` and `STOP SLAVE` take effect immediately and ignore any delay. `RESET SLAVE` resets the delay to 0.

The `replication_applier_configuration` Performance Schema table contains the `DESIRED_DELAY` column which shows the delay configured using the `MASTER_DELAY` option. The `replication_applier_status` Performance Schema table contains the `REMAINING_DELAY` column which shows the number of delay seconds remaining.

Delayed replication can be used for several purposes:

- To protect against user mistakes on the master. With a delay you can roll back a delayed slave to the time just before the mistake.
- To test how the system behaves when there is a lag. For example, in an application, a lag might be caused by a heavy load on the slave. However, it can be difficult to generate this load level. Delayed replication can simulate the lag without having to simulate the load. It can also be used to debug conditions related to a lagging slave.
- To inspect what the database looked like in the past, without having to reload a backup. For example, by configuring a slave with a delay of one week, if you then need to see what the database looked like before the last few days' worth of development, the delayed slave can be inspected.

Replication Delay Timestamps

MySQL 8.0 provides a new method for measuring delay (also referred to as replication lag) in replication topologies that depends on the following timestamps associated with the GTID of each transaction (instead of each event) written to the binary log.

- `original_commit_timestamp`: the number of microseconds since epoch when the transaction was written (committed) to the binary log of the original master.
- `immediate_commit_timestamp`: the number of microseconds since epoch when the transaction was written (committed) to the binary log of the immediate master.

The output of `mysqlbinlog` displays these timestamps in two formats, microseconds from epoch and also `TIMESTAMP` format, which is based on the user defined time zone for better readability. For example:

```
#170404 10:48:05 server id 1  end_log_pos 233 CRC32 0x016ce647     GTID    last_committed=0
\ sequence_number=1    original_committed_timestamp=1491299285661130    immediate_commit_timestamp=14912992858
# original_commit_timestamp=1491299285661130 (2017-04-04 10:48:05.661130 WEST)
# immediate_commit_timestamp=1491299285843771 (2017-04-04 10:48:05.843771 WEST)
 /*!80001 SET @@SESSION.original_commit_timestamp=1491299285661130*//*!*/;
   SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa:1'/*!*/;
# at 233
```

As a rule, the `original_commit_timestamp` is always the same on all replicas where the transaction is applied. In master-slave replication, the `original_commit_timestamp` of a transaction in the (original) master's binary log is always the same as its `immediate_commit_timestamp`. In the slave's relay log, the `original_commit_timestamp` and `immediate_commit_timestamp` of the transaction are the same as in the master's binary log; whereas in its own binary log, the transaction's `immediate_commit_timestamp` corresponds to when the slave committed the transaction.

In a Group Replication setup, when the original master is a member of a group, the `original_commit_timestamp` is generated when the transaction is ready to be committed. In other words, when it finished executing on the original master and its write set is ready to be sent to all members of the group for certification. Therefore, the same `original_commit_timestamp` is replicated to all servers (regardless of whether it is a group member or slave replicating from a member) applying the transaction and each stores in its binary log the local commit time using `immediate_commit_timestamp`.

View change events, which are exclusive to Group Replication, are a special case. Transactions containing these events are generated by each server but share the same GTID (so, they are not first executed in a master and then replicated to the group, but all members of the group execute and apply the same transaction). Since there is no original master, these transactions have their `original_commit_timestamp` set to zero.

Monitoring Replication Delay

One of the most common ways to monitor replication delay (lag) in previous MySQL versions was by relying on the `Seconds_Behind_Master` field in the output of `SHOW SLAVE STATUS`. However, this metric is not suitable when using replication topologies more complex than the traditional master-slave setup, such as Group Replication. The addition of `immediate_commit_timestamp` and `original_commit_timestamp` to MySQL 8 provides a much finer degree of information about replication delay. The recommended method to monitor replication delay in a topology that supports these timestamps is using the following Performance Schema tables.

- `replication_connection_status`: current status of the connection to the master, provides information on the last and current transaction the connection thread queued into the relay log.
- `replication_applier_status_by_coordinator`: current status of the coordinator thread that only displays information when using a multithreaded slave, provides information on the last transaction buffered by the coordinator thread to a worker's queue, as well as the transaction it is currently buffering.
- `replication_applier_status_by_worker`: current status of the thread(s) applying transactions received from the master, provides information about the transactions applied by the applier thread, or by each worker when using a multithreaded slave.

Using these tables you can monitor information about the last transaction the corresponding thread processed and the transaction that thread is currently processing. This information comprises:

- a transaction's GTID
- a transaction's `original_commit_timestamp` and `immediate_commit_timestamp`, retrieved from the slave's relay log
- the time a thread started processing a transaction
- for the last processed transaction, the time the thread finished processing it

In addition to the Performance Schema tables, the output of `SHOW SLAVE STATUS` has three fields that show:

- `SQL_Delay`: A nonnegative integer indicating the replication delay configured using `CHANGE MASTER TO MASTER_DELAY=N`, measured in seconds.
- `SQL_Remaining_Delay`: When `Slave_SQL_Running_State` is `Waiting until MASTER_DELAY seconds after master executed event`, this field contains an integer indicating the number of seconds left of the delay. At other times, this field is `NULL`.

- `Slave_SQL_Running_State`: A string indicating the state of the SQL thread (analogous to `Slave_IO_State`). The value is identical to the `State` value of the SQL thread as displayed by `SHOW PROCESSLIST`.

When the slave SQL thread is waiting for the delay to elapse before executing an event, `SHOW PROCESSLIST` displays its `State` value as `Waiting until MASTER_DELAY seconds after master executed event`.

17.5 Replication Notes and Tips

17.5.1 Replication Features and Issues

The following sections provide information about what is supported and what is not in MySQL replication, and about specific issues and situations that may occur when replicating certain statements.

Statement-based replication depends on compatibility at the SQL level between the master and slave. In other words, successful statement-based replication requires that any SQL features used be supported by both the master and the slave servers. If you use a feature on the master server that is available only in the current version of MySQL, you cannot replicate to a slave that uses an earlier version of MySQL. Such incompatibilities can also occur within a release series as well as between versions.

If you are planning to use statement-based replication between MySQL 8.0 and a previous MySQL release series, it is a good idea to consult the edition of the *MySQL Reference Manual* corresponding to the earlier release series for information regarding the replication characteristics of that series.

With MySQL's statement-based replication, there may be issues with replicating stored routines or triggers. You can avoid these issues by using MySQL's row-based replication instead. For a detailed list of issues, see Section 24.7, “Stored Program Binary Logging”. For more information about row-based logging and row-based replication, see Section 5.4.4.1, “Binary Logging Formats”, and Section 17.2.1, “Replication Formats”.

For additional information specific to replication and `InnoDB`, see Section 15.18, “InnoDB and MySQL Replication”. For information relating to replication with NDB Cluster, see Section 22.6, “NDB Cluster Replication”.

17.5.1.1 Replication and AUTO_INCREMENT

Statement-based replication of `AUTO_INCREMENT`, `LAST_INSERT_ID()`, and `TIMESTAMP` values is carried out subject to the following exceptions:

- A statement invoking a trigger or function that causes an update to an `AUTO_INCREMENT` column is not replicated correctly using statement-based replication. These statements are marked as unsafe. (Bug #45677)
- An `INSERT` into a table that has a composite primary key that includes an `AUTO_INCREMENT` column that is not the first column of this composite key is not safe for statement-based logging or replication. These statements are marked as unsafe. (Bug #11754117, Bug #45670)

  This issue does not affect tables using the `InnoDB` storage engine, since an `InnoDB` table with an AUTO_INCREMENT column requires at least one key where the auto-increment column is the only or leftmost column.
- Adding an `AUTO_INCREMENT` column to a table with `ALTER TABLE` might not produce the same ordering of the rows on the slave and the master. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the master and slave, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an

AUTO_INCREMENT column to a table t1 that has columns col1 and col2, the following statements produce a new table t2 identical to t1 but with an AUTO_INCREMENT column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important

To guarantee the same ordering on both master and slave, the ORDER BY clause must name *all* columns of t1.

The instructions just given are subject to the limitations of CREATE TABLE ... LIKE: Foreign key definitions are ignored, as are the DATA DIRECTORY and INDEX DIRECTORY table options. If a table definition includes any of those characteristics, create t2 using a CREATE TABLE statement that is identical to the one used to create t1, but with the addition of the AUTO_INCREMENT column.

Regardless of the method used to create and populate the copy having the AUTO_INCREMENT column, the final step is to drop the original table and then rename the copy:

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

See also Section B.4.6.1, “Problems with ALTER TABLE”.

17.5.1.2 Replication and BLACKHOLE Tables

The BLACKHOLE storage engine accepts data but discards it and does not store it. When performing binary logging, all inserts to such tables are always logged, regardless of the logging format in use. Updates and deletes are handled differently depending on whether statement based or row based logging is in use. With the statement based logging format, all statements affecting BLACKHOLE tables are logged, but their effects ignored. When using row-based logging, updates and deletes to such tables are simply skipped—they are not written to the binary log. A warning is logged whenever this occurs.

For this reason we recommend when you replicate to tables using the BLACKHOLE storage engine that you have the binlog_format server variable set to STATEMENT, and not to either ROW or MIXED.

17.5.1.3 Replication and Character Sets

The following applies to replication between MySQL servers that use different character sets:

- If the master has databases with a character set different from the global character_set_server value, you should design your CREATE TABLE statements so that they do not implicitly rely on the database default character set. A good workaround is to state the character set and collation explicitly in CREATE TABLE statements.

17.5.1.4 Replication and CHECKSUM TABLE

CHECKSUM TABLE returns a checksum that is calculated row by row, using a method that depends on the table row storage format. The storage format is not guaranteed to remain the same between MySQL versions, so the checksum value might change following an upgrade.

17.5.1.5 Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER

The statements CREATE SERVER, ALTER SERVER, and DROP SERVER are not written to the binary log, regardless of the binary logging format that is in use.

17.5.1.6 Replication of CREATE ... IF NOT EXISTS Statements

MySQL applies these rules when various `CREATE ... IF NOT EXISTS` statements are replicated:

- Every `CREATE DATABASE IF NOT EXISTS` statement is replicated, whether or not the database already exists on the master.
- Similarly, every `CREATE TABLE IF NOT EXISTS` statement without a `SELECT` is replicated, whether or not the table already exists on the master. This includes `CREATE TABLE IF NOT EXISTS ... LIKE`. Replication of `CREATE TABLE IF NOT EXISTS ... SELECT` follows somewhat different rules; see Section 17.5.1.7, "Replication of CREATE TABLE ... SELECT Statements", for more information.
- `CREATE EVENT IF NOT EXISTS` is always replicated, whether or not the event named in the statement already exists on the master.

17.5.1.7 Replication of CREATE TABLE ... SELECT Statements

MySQL applies these rules when `CREATE TABLE ... SELECT` statements are replicated:

- `CREATE TABLE ... SELECT` always performs an implicit commit (Section 13.3.3, "Statements That Cause an Implicit Commit").
- If the destination table does not exist, logging occurs as follows. It does not matter whether `IF NOT EXISTS` is present.
  - `STATEMENT` or `MIXED` format: The statement is logged as written.
  - `ROW` format: The statement is logged as a `CREATE TABLE` statement followed by a series of insert-row events.
- If the `CREATE TABLE ... SELECT` statement fails, nothing is logged. This includes the case that the destination table exists and `IF NOT EXISTS` is not given.
- If the destination table exists and `IF NOT EXISTS` is given, MySQL 8.0 ignores the statement completely; nothing is inserted or logged.

When statement-based replication is in use, MySQL 8.0 does not allow a `CREATE TABLE ... SELECT` statement to make any changes in tables other than the table that is created by the statement. This is not an issue when using row-based replication, because the statement is logged as a `CREATE TABLE` statement with any changes to table data logged as row-insert events, rather than as the entire `CREATE TABLE ... SELECT`.

17.5.1.8 Replication of CURRENT_USER()

The following statements support use of the `CURRENT_USER()` function to take the place of the name of, and possibly the host for, an affected user or a definer:

- `DROP USER`
- `RENAME USER`
- `GRANT`
- `REVOKE`
- `CREATE FUNCTION`
- `CREATE PROCEDURE`

- `CREATE TRIGGER`
- `CREATE EVENT`
- `CREATE VIEW`
- `ALTER EVENT`
- `ALTER VIEW`
- `SET PASSWORD`

When binary logging is enabled and `CURRENT_USER()` or `CURRENT_USER` is used as the definer in any of these statements, MySQL Server ensures that the statement is applied to the same user on both the master and the slave when the statement is replicated. In some cases, such as statements that change passwords, the function reference is expanded before it is written to the binary log, so that the statement includes the user name. For all other cases, the name of the current user on the master is replicated to the slave as metadata, and the slave applies the statement to the current user named in the metadata, rather than to the current user on the slave.

17.5.1.9 Replication with Differing Table Definitions on Master and Slave

Source and target tables for replication do not have to be identical. A table on the master can have more or fewer columns than the slave's copy of the table. In addition, corresponding table columns on the master and the slave can use different data types, subject to certain conditions.

Note

Replication between tables which are partitioned differently from one another is not supported. See Section 17.5.1.24, “Replication and Partitioning”.

In all cases where the source and target tables do not have identical definitions, the database and table names must be the same on both the master and the slave. Additional conditions are discussed, with examples, in the following two sections.

Replication with More Columns on Master or Slave

You can replicate a table from the master to the slave such that the master and slave copies of the table have differing numbers of columns, subject to the following conditions:

- Columns common to both versions of the table must be defined in the same order on the master and the slave.

  (This is true even if both tables have the same number of columns.)

- Columns common to both versions of the table must be defined before any additional columns.

  This means that executing an `ALTER TABLE` statement on the slave where a new column is inserted into the table within the range of columns common to both tables causes replication to fail, as shown in the following example:

  Suppose that a table `t`, existing on the master and the slave, is defined by the following `CREATE TABLE` statement:

```
CREATE TABLE t (
    c1 INT,
    c2 INT,
    c3 INT
);
```

Suppose that the `ALTER TABLE` statement shown here is executed on the slave:

```
ALTER TABLE t ADD COLUMN cnew1 INT AFTER c3;
```

The previous `ALTER TABLE` is permitted on the slave because the columns `c1`, `c2`, and `c3` that are common to both versions of table `t` remain grouped together in both versions of the table, before any columns that differ.

However, the following `ALTER TABLE` statement cannot be executed on the slave without causing replication to break:

```
ALTER TABLE t ADD COLUMN cnew2 INT AFTER c2;
```

Replication fails after execution on the slave of the `ALTER TABLE` statement just shown, because the new column `cnew2` comes between columns common to both versions of `t`.

- Each “extra” column in the version of the table having more columns must have a default value.

  A column's default value is determined by a number of factors, including its type, whether it is defined with a `DEFAULT` option, whether it is declared as `NULL`, and the server SQL mode in effect at the time of its creation; for more information, see Section 11.7, “Data Type Default Values”).

In addition, when the slave's copy of the table has more columns than the master's copy, each column common to the tables must use the same data type in both tables.

Examples. The following examples illustrate some valid and invalid table definitions:

More columns on the master. The following table definitions are valid and replicate correctly:

```
master> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
slave>  CREATE TABLE t1 (c1 INT, c2 INT);
```

The following table definitions would raise an error because the definitions of the columns common to both versions of the table are in a different order on the slave than they are on the master:

```
master> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
slave>  CREATE TABLE t1 (c2 INT, c1 INT);
```

The following table definitions would also raise an error because the definition of the extra column on the master appears before the definitions of the columns common to both versions of the table:

```
master> CREATE TABLE t1 (c3 INT, c1 INT, c2 INT);
slave>  CREATE TABLE t1 (c1 INT, c2 INT);
```

More columns on the slave. The following table definitions are valid and replicate correctly:

```
master> CREATE TABLE t1 (c1 INT, c2 INT);
slave>  CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
```

The following definitions raise an error because the columns common to both versions of the table are not defined in the same order on both the master and the slave:

```
master> CREATE TABLE t1 (c1 INT, c2 INT);
slave>  CREATE TABLE t1 (c2 INT, c1 INT, c3 INT);
```

The following table definitions also raise an error because the definition for the extra column in the slave's version of the table appears before the definitions for the columns which are common to both versions of the table:

```
master> CREATE TABLE t1 (c1 INT, c2 INT);
slave>  CREATE TABLE t1 (c3 INT, c1 INT, c2 INT);
```

The following table definitions fail because the slave's version of the table has additional columns compared to the master's version, and the two versions of the table use different data types for the common column `c2`:

```
master> CREATE TABLE t1 (c1 INT, c2 BIGINT);
slave>  CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
```

Replication of Columns Having Different Data Types

Corresponding columns on the master's and the slave's copies of the same table ideally should have the same data type. However, this is not always strictly enforced, as long as certain conditions are met.

It is usually possible to replicate from a column of a given data type to another column of the same type and same size or width, where applicable, or larger. For example, you can replicate from a `CHAR(10)` column to another `CHAR(10)`, or from a `CHAR(10)` column to a `CHAR(25)` column without any problems. In certain cases, it also possible to replicate from a column having one data type (on the master) to a column having a different data type (on the slave); when the data type of the master's version of the column is promoted to a type that is the same size or larger on the slave, this is known as *attribute promotion*.

Attribute promotion can be used with both statement-based and row-based replication, and is not dependent on the storage engine used by either the master or the slave. However, the choice of logging format does have an effect on the type conversions that are permitted; the particulars are discussed later in this section.

Important

Whether you use statement-based or row-based replication, the slave's copy of the table cannot contain more columns than the master's copy if you wish to employ attribute promotion.

Statement-based replication. When using statement-based replication, a simple rule of thumb to follow is, “If the statement run on the master would also execute successfully on the slave, it should also replicate successfully”. In other words, if the statement uses a value that is compatible with the type of a given column on the slave, the statement can be replicated. For example, you can insert any value that fits in a `TINYINT` column into a `BIGINT` column as well; it follows that, even if you change the type of a `TINYINT` column in the slave's copy of a table to `BIGINT`, any insert into that column on the master that succeeds should also succeed on the slave, since it is impossible to have a legal `TINYINT` value that is large enough to exceed a `BIGINT` column.

Row-based replication: attribute promotion and demotion. Row-based replication supports attribute promotion and demotion between smaller data types and larger types. It is also possible to specify whether or not to permit lossy (truncated) or non-lossy conversions of demoted column values, as explained later in this section.

Lossy and non-lossy conversions. In the event that the target type cannot represent the value being inserted, a decision must be made on how to handle the conversion. If we permit the conversion but truncate (or otherwise modify) the source value to achieve a “fit” in the target column, we make what is

known as a *lossy conversion*. A conversion which does not require truncation or similar modifications to fit the source column value in the target column is a *non-lossy* conversion.

Type conversion modes (slave_type_conversions variable). The setting of the `slave_type_conversions` global server variable controls the type conversion mode used on the slave. This variable takes a set of values from the following list, which describes the effects of each mode on the slave's type-conversion behavior:

ALL_LOSSY	In this mode, type conversions that would mean loss of information are permitted. This does not imply that non-lossy conversions are permitted, merely that only cases requiring either lossy conversions or no conversion at all are permitted; for example, enabling *only* this mode permits an `INT` column to be converted to `TINYINT` (a lossy conversion), but not a `TINYINT` column to an `INT` column (non-lossy). Attempting the latter conversion in this case would cause replication to stop with an error on the slave.
ALL_NON_LOSSY	This mode permits conversions that do not require truncation or other special handling of the source value; that is, it permits conversions where the target type has a wider range than the source type. Setting this mode has no bearing on whether lossy conversions are permitted; this is controlled with the `ALL_LOSSY` mode. If only `ALL_NON_LOSSY` is set, but not `ALL_LOSSY`, then attempting a conversion that would result in the loss of data (such as `INT` to `TINYINT`, or `CHAR(25)` to `VARCHAR(20)`) causes the slave to stop with an error.
ALL_LOSSY,ALL_NON_LOSSY	When this mode is set, all supported type conversions are permitted, whether or not they are lossy conversions.
ALL_SIGNED	Treat promoted integer types as signed values (the default behavior).
ALL_UNSIGNED	Treat promoted integer types as unsigned values.
ALL_SIGNED,ALL_UNSIGNED	Treat promoted integer types as signed if possible, otherwise as unsigned.
[*empty*]	When `slave_type_conversions` is not set, no attribute promotion or demotion is permitted; this means that all columns in the source and target tables must be of the same types. This mode is the default.

When an integer type is promoted, its signedness is not preserved. By default, the slave treats all such values as signed. You can control this behavior using `ALL_SIGNED`, `ALL_UNSIGNED`, or both. `ALL_SIGNED` tells the slave to treat all promoted integer types as signed; `ALL_UNSIGNED` instructs it to treat these as unsigned. Specifying both causes the slave to treat the value as signed if possible, otherwise to treat it as unsigned; the order in which they are listed is not significant. Neither `ALL_SIGNED` nor `ALL_UNSIGNED` has any effect if at least one of `ALL_LOSSY` or `ALL_NONLOSSY` is not also used.

Changing the type conversion mode requires restarting the slave with the new `slave_type_conversions` setting.

Supported conversions. Supported conversions between different but similar data types are shown in the following list:

- Between any of the integer types `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, and `BIGINT`.

  This includes conversions between the signed and unsigned versions of these types.

  Lossy conversions are made by truncating the source value to the maximum (or minimum) permitted by the target column. For ensuring non-lossy conversions when going from unsigned to signed types, the target column must be large enough to accommodate the range of values in the source column. For example, you can demote `TINYINT UNSIGNED` non-lossily to `SMALLINT`, but not to `TINYINT`.

- Between any of the decimal types `DECIMAL`, `FLOAT`, `DOUBLE`, and `NUMERIC`.

  `FLOAT` to `DOUBLE` is a non-lossy conversion; `DOUBLE` to `FLOAT` can only be handled lossily. A conversion from `DECIMAL(M,D)` to `DECIMAL(M',D')` where `D' >= D` and `(M'-D') >= (M-D)` is non-lossy; for any case where `M' < M`, `D' < D`, or both, only a lossy conversion can be made.

  For any of the decimal types, if a value to be stored cannot be fit in the target type, the value is rounded down according to the rounding rules defined for the server elsewhere in the documentation. See Section 12.25.4, “Rounding Behavior”, for information about how this is done for decimal types.

- Between any of the string types `CHAR`, `VARCHAR`, and `TEXT`, including conversions between different widths.

  Conversion of a `CHAR`, `VARCHAR`, or `TEXT` to a `CHAR`, `VARCHAR`, or `TEXT` column the same size or larger is never lossy. Lossy conversion is handled by inserting only the first `N` characters of the string on the slave, where `N` is the width of the target column.

  Important

  Replication between columns using different character sets is not supported.

- Between any of the binary data types `BINARY`, `VARBINARY`, and `BLOB`, including conversions between different widths.

  Conversion of a `BINARY`, `VARBINARY`, or `BLOB` to a `BINARY`, `VARBINARY`, or `BLOB` column the same size or larger is never lossy. Lossy conversion is handled by inserting only the first `N` bytes of the string on the slave, where `N` is the width of the target column.

- Between any 2 `BIT` columns of any 2 sizes.

  When inserting a value from a `BIT(M)` column into a `BIT(M')` column, where `M' > M`, the most significant bits of the `BIT(M')` columns are cleared (set to zero) and the `M` bits of the `BIT(M)` value are set as the least significant bits of the `BIT(M')` column.

  When inserting a value from a source `BIT(M)` column into a target `BIT(M')` column, where `M' < M`, the maximum possible value for the `BIT(M')` column is assigned; in other words, an “all-set” value is assigned to the target column.

Conversions between types not in the previous list are not permitted.

17.5.1.10 Replication and DIRECTORY Table Options

If a `DATA DIRECTORY` or `INDEX DIRECTORY` table option is used in a `CREATE TABLE` statement on the master server, the table option is also used on the slave. This can cause problems if no corresponding directory exists in the slave host file system or if it exists but is not accessible to the slave server. This can be overridden by using the `NO_DIR_IN_CREATE` server SQL mode on the slave, which causes the slave to ignore the `DATA DIRECTORY` and `INDEX DIRECTORY` table options when replicating `CREATE TABLE` statements. The result is that `MyISAM` data and index files are created in the table's database directory.

For more information, see Section 5.1.11, “Server SQL Modes”.

17.5.1.11 Replication of DROP ... IF EXISTS Statements

The `DROP DATABASE IF EXISTS`, `DROP TABLE IF EXISTS`, and `DROP VIEW IF EXISTS` statements are always replicated, even if the database, table, or view to be dropped does not exist on the master. This is to ensure that the object to be dropped no longer exists on either the master or the slave, once the slave has caught up with the master.

`DROP ... IF EXISTS` statements for stored programs (stored procedures and functions, triggers, and events) are also replicated, even if the stored program to be dropped does not exist on the master.

17.5.1.12 Replication and Floating-Point Values

With statement-based replication, values are converted from decimal to binary. Because conversions between decimal and binary representations of them may be approximate, comparisons involving floating-point values are inexact. This is true for operations that use floating-point values explicitly, or that use values that are converted to floating-point implicitly. Comparisons of floating-point values might yield different results on master and slave servers due to differences in computer architecture, the compiler used to build MySQL, and so forth. See Section 12.2, “Type Conversion in Expression Evaluation”, and Section B.4.4.8, “Problems with Floating-Point Values”.

17.5.1.13 Replication and FLUSH

Some forms of the `FLUSH` statement are not logged because they could cause problems if replicated to a slave: `FLUSH LOGS` and `FLUSH TABLES WITH READ LOCK`. For a syntax example, see Section 13.7.8.3, “FLUSH Syntax”. The `FLUSH TABLES`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements are written to the binary log and thus replicated to slaves. This is not normally a problem because these statements do not modify table data.

However, this behavior can cause difficulties under certain circumstances. If you replicate the privilege tables in the `mysql` database and update those tables directly without using `GRANT`, you must issue a `FLUSH PRIVILEGES` on the slaves to put the new privileges into effect. In addition, if you use `FLUSH TABLES` when renaming a `MyISAM` table that is part of a `MERGE` table, you must issue `FLUSH TABLES` manually on the slaves. These statements are written to the binary log unless you specify `NO_WRITE_TO_BINLOG` or its alias `LOCAL`.

17.5.1.14 Replication and System Functions

Certain functions do not replicate well under some conditions:

- The `USER()`, `CURRENT_USER()` (or `CURRENT_USER`), `UUID()`, `VERSION()`, and `LOAD_FILE()` functions are replicated without change and thus do not work reliably on the slave unless row-based replication is enabled. (See Section 17.2.1, “Replication Formats”.)

  `USER()` and `CURRENT_USER()` are automatically replicated using row-based replication when using `MIXED` mode, and generate a warning in `STATEMENT` mode. (See also Section 17.5.1.8, “Replication of CURRENT_USER()”.) This is also true for `VERSION()` and `RAND()`.

- For `NOW()`, the binary log includes the timestamp. This means that the value *as returned by the call to this function on the master* is replicated to the slave. To avoid unexpected results when replicating between MySQL servers in different time zones, set the time zone on both master and slave. See also Section 17.5.1.32, “Replication and Time Zones”

  To explain the potential problems when replicating between servers which are in different time zones, suppose that the master is located in New York, the slave is located in Stockholm, and both servers are

using local time. Suppose further that, on the master, you create a table `mytable`, perform an `INSERT` statement on this table, and then select from the table, as shown here:

```
mysql> CREATE TABLE mytable (mycol TEXT);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO mytable VALUES ( NOW() );
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM mytable;
+---------------------+
| mycol               |
+---------------------+
| 2009-09-01 12:00:00 |
+---------------------+
1 row in set (0.00 sec)
```

Local time in Stockholm is 6 hours later than in New York; so, if you issue `SELECT NOW()` on the slave at that exact same instant, the value `2009-09-01 18:00:00` is returned. For this reason, if you select from the slave's copy of `mytable` after the `CREATE TABLE` and `INSERT` statements just shown have been replicated, you might expect `mycol` to contain the value `2009-09-01 18:00:00`. However, this is not the case; when you select from the slave's copy of `mytable`, you obtain exactly the same result as on the master:

```
mysql> SELECT * FROM mytable;
+---------------------+
| mycol               |
+---------------------+
| 2009-09-01 12:00:00 |
+---------------------+
1 row in set (0.00 sec)
```

Unlike `NOW()`, the `SYSDATE()` function is not replication-safe because it is not affected by `SET TIMESTAMP` statements in the binary log and is nondeterministic if statement-based logging is used. This is not a problem if row-based logging is used.

An alternative is to use the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`. This must be done on the master and the slave to work correctly. In such cases, a warning is still issued by this function, but can safely be ignored as long as `--sysdate-is-now` is used on both the master and the slave.

`SYSDATE()` is automatically replicated using row-based replication when using `MIXED` mode, and generates a warning in `STATEMENT` mode.

See also Section 17.5.1.32, “Replication and Time Zones”.

- *The following restriction applies to statement-based replication only, not to row-based replication.* The `GET_LOCK()`, `RELEASE_LOCK()`, `IS_FREE_LOCK()`, and `IS_USED_LOCK()` functions that handle user-level locks are replicated without the slave knowing the concurrency context on the master. Therefore, these functions should not be used to insert into a master table because the content on the slave would differ. For example, do not issue a statement such as `INSERT INTO mytable VALUES(GET_LOCK(...))`.

  These functions are automatically replicated using row-based replication when using `MIXED` mode, and generate a warning in `STATEMENT` mode.

As a workaround for the preceding limitations when statement-based replication is in effect, you can use the strategy of saving the problematic function result in a user variable and referring to the variable in a

later statement. For example, the following single-row `INSERT` is problematic due to the reference to the `UUID()` function:

```
INSERT INTO t VALUES(UUID());
```

To work around the problem, do this instead:

```
SET @my_uuid = UUID();
INSERT INTO t VALUES(@my_uuid);
```

That sequence of statements replicates because the value of `@my_uuid` is stored in the binary log as a user-variable event prior to the `INSERT` statement and is available for use in the `INSERT`.

The same idea applies to multiple-row inserts, but is more cumbersome to use. For a two-row insert, you can do this:

```
SET @my_uuid1 = UUID(); @my_uuid2 = UUID();
INSERT INTO t VALUES(@my_uuid1),(@my_uuid2);
```

However, if the number of rows is large or unknown, the workaround is difficult or impracticable. For example, you cannot convert the following statement to one in which a given individual user variable is associated with each row:

```
INSERT INTO t2 SELECT UUID(), * FROM t1;
```

Within a stored function, `RAND()` replicates correctly as long as it is invoked only once during the execution of the function. (You can consider the function execution timestamp and random number seed as implicit inputs that are identical on the master and slave.)

The `FOUND_ROWS()` and `ROW_COUNT()` functions are not replicated reliably using statement-based replication. A workaround is to store the result of the function call in a user variable, and then use that in the `INSERT` statement. For example, if you wish to store the result in a table named `mytable`, you might normally do so like this:

```
SELECT SQL_CALC_FOUND_ROWS FROM mytable LIMIT 1;
INSERT INTO mytable VALUES( FOUND_ROWS() );
```

However, if you are replicating `mytable`, you should use `SELECT ... INTO`, and then store the variable in the table, like this:

```
SELECT SQL_CALC_FOUND_ROWS INTO @found_rows FROM mytable LIMIT 1;
INSERT INTO mytable VALUES(@found_rows);
```

In this way, the user variable is replicated as part of the context, and applied on the slave correctly.

These functions are automatically replicated using row-based replication when using `MIXED` mode, and generate a warning in `STATEMENT` mode. (Bug #12092, Bug #30244)

17.5.1.15 Replication and Fractional Seconds Support

MySQL 8.0 permits fractional seconds for `TIME`, `DATETIME`, and `TIMESTAMP` values, with up to microseconds (6 digits) precision. See Section 11.3.5, “Fractional Seconds in Time Values”.

17.5.1.16 Replication of Invoked Features

Replication of invoked features such as user-defined functions (UDFs) and stored programs (stored procedures and functions, triggers, and events) provides the following characteristics:

- The effects of the feature are always replicated.
- The following statements are replicated using statement-based replication:
  - `CREATE EVENT`
  - `ALTER EVENT`
  - `DROP EVENT`
  - `CREATE PROCEDURE`
  - `DROP PROCEDURE`
  - `CREATE FUNCTION`
  - `DROP FUNCTION`
  - `CREATE TRIGGER`
  - `DROP TRIGGER`

  However, the *effects* of features created, modified, or dropped using these statements are replicated using row-based replication.

Note

Attempting to replicate invoked features using statement-based replication produces the warning `Statement is not safe to log in statement format`. For example, trying to replicate a UDF with statement-based replication generates this warning because it currently cannot be determined by the MySQL server whether the UDF is deterministic. If you are absolutely certain that the invoked feature's effects are deterministic, you can safely disregard such warnings.

- In the case of `CREATE EVENT` and `ALTER EVENT`:
  - The status of the event is set to `SLAVESIDE_DISABLED` on the slave regardless of the state specified (this does not apply to `DROP EVENT`).
  - The master on which the event was created is identified on the slave by its server ID. The `ORIGINATOR` column in `INFORMATION_SCHEMA.EVENTS` stores this information. See Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”, and Section 13.7.7.18, “SHOW EVENTS Syntax”, for more information.
- The feature implementation resides on the slave in a renewable state so that if the master fails, the slave can be used as the master without loss of event processing.

To determine whether there are any scheduled events on a MySQL server that were created on a different server (that was acting as a replication master), query the `INFORMATION_SCHEMA.EVENTS` table in a manner similar to what is shown here:

```
SELECT EVENT_SCHEMA, EVENT_NAME
```

```
    FROM INFORMATION_SCHEMA.EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED';
```

Alternatively, you can use the `SHOW EVENTS` statement, like this:

```
SHOW EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED';
```

When promoting a replication slave having such events to a replication master, you must enable each event using `ALTER EVENT event_name ENABLE`, where *`event_name`* is the name of the event.

If more than one master was involved in creating events on this slave, and you wish to identify events that were created only on a given master having the server ID *`master_id`*, modify the previous query on the `EVENTS` table to include the `ORIGINATOR` column, as shown here:

```
SELECT EVENT_SCHEMA, EVENT_NAME, ORIGINATOR
    FROM INFORMATION_SCHEMA.EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED'
    AND   ORIGINATOR = 'master_id'
```

You can employ `ORIGINATOR` with the `SHOW EVENTS` statement in a similar fashion:

```
SHOW EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED'
    AND   ORIGINATOR = 'master_id'
```

Before enabling events that were replicated from the master, you should disable the MySQL Event Scheduler on the slave (using a statement such as `SET GLOBAL event_scheduler = OFF;`), run any necessary `ALTER EVENT` statements, restart the server, then re-enable the Event Scheduler on the slave afterward (using a statement such as `SET GLOBAL event_scheduler = ON;`)-

If you later demote the new master back to being a replication slave, you must disable manually all events enabled by the `ALTER EVENT` statements. You can do this by storing in a separate table the event names from the `SELECT` statement shown previously, or using `ALTER EVENT` statements to rename the events with a common prefix such as `replicated_` to identify them.

If you rename the events, then when demoting this server back to being a replication slave, you can identify the events by querying the `EVENTS` table, as shown here:

```
SELECT CONCAT(EVENT_SCHEMA, '.', EVENT_NAME) AS 'Db.Event'
      FROM INFORMATION_SCHEMA.EVENTS
      WHERE INSTR(EVENT_NAME, 'replicated_') = 1;
```

17.5.1.17 Replication of JSON Documents

Before MySQL 8.0, an update to a JSON column was always written to the binary log as the complete document. In MySQL 8.0, it is possible to log partial updates to JSON documents (see Partial Updates of JSON Values), which is more efficient. The logging behavior depends on the format used, as described here:

Statement-based replication. JSON partial updates are always logged as partial updates. This cannot be disabled when using statement-based logging.

Row-based replication. JSON partial updates are not logged as such by default, but instead are logged as complete documents. To enable logging of partial updates, set

`binlog_row_value_options=PARTIAL_JSON`. If a replication master has this variable set, partial updates received from that master are handled and applied by a replication slave regardless of the slave's own setting for the variable.

Servers running MySQL 8.0.2 or earlier do not recognize the log events used for JSON partial updates. For this reason, when replicating to such a server from a server running MySQL 8.0.3 or later, `binlog_row_value_options` must be disabled on the master by setting this variable to `''` (empty string). See the description of this variable for more information.

17.5.1.18 Replication and LIMIT

Statement-based replication of `LIMIT` clauses in `DELETE`, `UPDATE`, and `INSERT ... SELECT` statements is unsafe since the order of the rows affected is not defined. (Such statements can be replicated correctly with statement-based replication only if they also contain an `ORDER BY` clause.) When such a statement is encountered:

- When using `STATEMENT` mode, a warning that the statement is not safe for statement-based replication is now issued.

  When using `STATEMENT` mode, warnings are issued for DML statements containing `LIMIT` even when they also have an `ORDER BY` clause (and so are made deterministic). This is a known issue. (Bug #42851)

- When using `MIXED` mode, the statement is now automatically replicated using row-based mode.

17.5.1.19 Replication and LOAD DATA

`LOAD DATA` is considered unsafe for statement-based logging (see Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”). When `binlog_format=MIXED` is set, the statement is logged in row-based format. When `binlog_format=STATEMENT` is set, note that `LOAD DATA` does not generate a warning, unlike other unsafe statements.

If you do use `LOAD DATA` when `binlog_format=STATEMENT` is set, a temporary file containing the data is created on the replication slave where the changes are applied. The slave then uses a `LOAD DATA INFILE` statement to apply the changes. If binary log encryption is active on the server, note that this temporary file is not encrypted. When encryption is required, be sure to use row-based or mixed binary logging format instead, which do not create the temporary files.

If a `PRIVILEGE_CHECKS_USER` account has been used to help secure the replication channel (see Section 17.3.3, “Replication Privilege Checks”), for a `LOAD DATA INFILE` operation logged in statement format, the account would require the `INSERT` privilege on the relevant table and also the `FILE` privilege. However, it is strongly recommended that you do not give the `FILE` privilege to the account, and instead ensure that the event is logged in row-based format, so that this privilege is not needed. If you do need to recover from a replication error involving a `LOAD DATA INFILE` statement, and the replicated event is trusted, you could grant the `FILE` privilege to the `PRIVILEGE_CHECKS_USER` account temporarily, removing it after the replicated event has been applied.

When `mysqlbinlog` reads log events for `LOAD DATA` statements logged in statement-based format, a generated local file is created in a temporary directory. These temporary files are not automatically removed by `mysqlbinlog` or any other MySQL program. If you do use `LOAD DATA` statements with statement-based binary logging, you should delete the temporary files yourself after you no longer need the statement log. For more information, see Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

17.5.1.20 Replication and max_allowed_packet

`max_allowed_packet` sets an upper limit on the size of any single message between the MySQL server and clients, including replication slaves. If you are replicating large column values (such as might be found in `TEXT` or `BLOB` columns) and `max_allowed_packet` is too small on the master, the master fails with an error, and the slave shuts down the I/O thread. If `max_allowed_packet` is too small on the slave, this also causes the slave to stop the I/O thread.

Row-based replication currently sends all columns and column values for updated rows from the master to the slave, including values of columns that were not actually changed by the update. This means that, when you are replicating large column values using row-based replication, you must take care to set `max_allowed_packet` large enough to accommodate the largest row in any table to be replicated, even if you are replicating updates only, or you are inserting only relatively small values.

On a multi-threaded slave (with `slave_parallel_workers > 0`), ensure that the `slave_pending_jobs_size_max` system variable is set to a value equal to or greater than the setting for the `max_allowed_packet` system variable on the master. The default setting for `slave_pending_jobs_size_max`, 128M, is twice the default setting for `max_allowed_packet`, which is 64M. `max_allowed_packet` limits the packet size that the master will send, but the addition of an event header can produce a binary log event exceeding this size. Also, in row-based replication, a single event can be significantly larger than the `max_allowed_packet` size, because the value of `max_allowed_packet` only limits each column of the table.

The replication slave actually accepts packets up to the limit set by its `slave_max_allowed_packet` setting, which defaults to the maximum setting of 1GB, to prevent a replication failure due to a large packet. However, the value of `slave_pending_jobs_size_max` controls the memory that is made available on the slave to hold incoming packets. The specified memory is shared among all the slave worker queues.

The value of `slave_pending_jobs_size_max` is a soft limit, and if an unusually large event (consisting of one or multiple packets) exceeds this size, the transaction is held until all the slave workers have empty queues, and then processed. All subsequent transactions are held until the large transaction has been completed. So although unusual events larger than `slave_pending_jobs_size_max` can be processed, the delay to clear the queues of all the slave workers and the wait to queue subsequent transactions can cause lag on the replication slave and decreased concurrency of the slave workers. `slave_pending_jobs_size_max` should therefore be set high enough to accommodate most expected event sizes.

17.5.1.21 Replication and MEMORY Tables

When a master server shuts down and restarts, its `MEMORY` tables become empty. To replicate this effect to slaves, the first time that the master uses a given `MEMORY` table after startup, it logs an event that notifies slaves that the table must be emptied by writing a `DELETE` statement for that table to the binary log. This generated event is identifiable by a comment in the binary log, and if GTIDs are in use on the server, it has a GTID assigned.

When a slave server shuts down and restarts, its `MEMORY` tables become empty. This causes the slave to be out of synchrony with the master and may lead to other failures or cause the slave to stop:

- Row-format updates and deletes received from the master may fail with `Can't find record in 'memory_table'`.
- Statements such as `INSERT INTO ... SELECT FROM memory_table` may insert a different set of rows on the master and slave.

The safe way to restart a slave that is replicating `MEMORY` tables is to first drop or delete all rows from the `MEMORY` tables on the master and wait until those changes have replicated to the slave. Then it is safe to restart the slave.

An alternative restart method may apply in some cases. When `binlog_format=ROW`, you can prevent the slave from stopping if you set `slave_exec_mode=IDEMPOTENT` before you start the slave again. This allows the slave to continue to replicate, but its `MEMORY` tables will still be different from those on the master. This can be okay if the application logic is such that the contents of `MEMORY` tables can be safely lost (for example, if the `MEMORY` tables are used for caching). `slave_exec_mode=IDEMPOTENT` applies globally to all tables, so it may hide other replication errors in non-`MEMORY` tables.

(The method just described is not applicable in NDB Cluster, where `slave_exec_mode` is always `IDEMPOTENT`, and cannot be changed.)

The size of `MEMORY` tables is limited by the value of the `max_heap_table_size` system variable, which is not replicated (see Section 17.5.1.38, “Replication and Variables”). A change in `max_heap_table_size` takes effect for `MEMORY` tables that are created or updated using `ALTER TABLE ... ENGINE = MEMORY` or `TRUNCATE TABLE` following the change, or for all `MEMORY` tables following a server restart. If you increase the value of this variable on the master without doing so on the slave, it becomes possible for a table on the master to grow larger than its counterpart on the slave, leading to inserts that succeed on the master but fail on the slave with `Table is full` errors. This is a known issue (Bug #48666). In such cases, you must set the global value of `max_heap_table_size` on the slave as well as on the master, then restart replication. It is also recommended that you restart both the master and slave MySQL servers, to insure that the new value takes complete (global) effect on each of them.

See Section 16.3, “The MEMORY Storage Engine”, for more information about `MEMORY` tables.

17.5.1.22 Replication of the mysql System Schema

Data modification statements made to tables in the `mysql` schema are replicated according to the value of `binlog_format`; if this value is `MIXED`, these statements are replicated using row-based format. However, statements that would normally update this information indirectly—such `GRANT`, `REVOKE`, and statements manipulating triggers, stored routines, and views—are replicated to slaves using statement-based replication.

17.5.1.23 Replication and the Query Optimizer

It is possible for the data on the master and slave to become different if a statement is written in such a way that the data modification is nondeterministic; that is, left up the query optimizer. (In general, this is not a good practice, even outside of replication.) Examples of nondeterministic statements include `DELETE` or `UPDATE` statements that use `LIMIT` with no `ORDER BY` clause; see Section 17.5.1.18, “Replication and LIMIT”, for a detailed discussion of these.

17.5.1.24 Replication and Partitioning

Replication is supported between partitioned tables as long as they use the same partitioning scheme and otherwise have the same structure except where an exception is specifically allowed (see Section 17.5.1.9, “Replication with Differing Table Definitions on Master and Slave”).

Replication between tables having different partitioning is generally not supported. This because statements (such as `ALTER TABLE ... DROP PARTITION`) acting directly on partitions in such cases may produce different results on master and slave. In the case where a table is partitioned on the master but not on the slave, any statements operating on partitions on the master's copy of the slave fail on the slave. When the slave's copy of the table is partitioned but the master's copy is not, statements acting on partitions cannot be run on the master without causing errors there.

Due to these dangers of causing replication to fail entirely (on account of failed statements) and of inconsistencies (when the result of a partition-level SQL statement produces different results on master

and slave), we recommend that insure that the partitioning of any tables to be replicated from the master is matched by the slave's versions of these tables.

17.5.1.25 Replication and REPAIR TABLE

When used on a corrupted or otherwise damaged table, it is possible for the `REPAIR TABLE` statement to delete rows that cannot be recovered. However, any such modifications of table data performed by this statement are not replicated, which can cause master and slave to lose synchronization. For this reason, in the event that a table on the master becomes damaged and you use `REPAIR TABLE` to repair it, you should first stop replication (if it is still running) before using `REPAIR TABLE`, then afterward compare the master's and slave's copies of the table and be prepared to correct any discrepancies manually, before restarting replication.

17.5.1.26 Replication and Reserved Words

You can encounter problems when you attempt to replicate from an older master to a newer slave and you make use of identifiers on the master that are reserved words in the newer MySQL version running on the slave. For example, a table column named `rank` on a MySQL 5.7 master that is replicating to a MySQL 8.0 slave could cause a problem because `RANK` is a reserved word beginning in MySQL 8.0.

Replication can fail in such cases with Error 1064 `You have an error in your SQL syntax...`, *even if a database or table named using the reserved word or a table having a column named using the reserved word is excluded from replication*. This is due to the fact that each SQL event must be parsed by the slave prior to execution, so that the slave knows which database object or objects would be affected. Only after the event is parsed can the slave apply any filtering rules defined by `--replicate-do-db`, `--replicate-do-table`, `--replicate-ignore-db`, and `--replicate-ignore-table`.

To work around the problem of database, table, or column names on the master which would be regarded as reserved words by the slave, do one of the following:

- Use one or more `ALTER TABLE` statements on the master to change the names of any database objects where these names would be considered reserved words on the slave, and change any SQL statements that use the old names to use the new names instead.
- In any SQL statements using these database object names, write the names as quoted identifiers using backtick characters (`` ` ``).

For listings of reserved words by MySQL version, see Reserved Words, in the *MySQL Server Version Reference*. For identifier quoting rules, see Section 9.2, “Schema Object Names”.

17.5.1.27 Replication and Master or Slave Shutdowns

It is safe to shut down a master server and restart it later. When a slave loses its connection to the master, the slave tries to reconnect immediately and retries periodically if that fails. The default is to retry every 60 seconds. This may be changed with the `CHANGE MASTER TO` statement. A slave also is able to deal with network connectivity outages. However, the slave notices the network outage only after receiving no data from the master for `slave_net_timeout` seconds. If your outages are short, you may want to decrease `slave_net_timeout`. See Section 17.4.2, “Handling an Unexpected Halt of a Replication Slave”.

An unclean shutdown (for example, a crash) on the master side can result in the master binary log having a final position less than the most recent position read by the slave, due to the master binary log file not being flushed. This can cause the slave not to be able to replicate when the master comes back up. Setting `sync_binlog=1` in the master `my.cnf` file helps to minimize this problem because it causes the master to flush its binary log more frequently. For the greatest possible durability and consistency in a replication setup using `InnoDB` with transactions, you should also set `innodb_flush_log_at_trx_commit=1`.

With this setting, the contents of the `InnoDB` redo log buffer are written out to the log file at each transaction commit and the log file is flushed to disk. Note that the durability of transactions is still not guaranteed with this setting, because operating systems or disk hardware may tell `mysqld` that the flush-to-disk operation has taken place, even though it has not.

Shutting down a slave cleanly is safe because it keeps track of where it left off. However, be careful that the slave does not have temporary tables open; see Section 17.5.1.30, “Replication and Temporary Tables”. Unclean shutdowns might produce problems, especially if the disk cache was not flushed to disk before the problem occurred:

- For transactions, the slave commits and then updates `relay-log.info`. If a crash occurs between these two operations, relay log processing will have proceeded further than the information file indicates and the slave will re-execute the events from the last transaction in the relay log after it has been restarted.
- A similar problem can occur if the slave updates `relay-log.info` but the server host crashes before the write has been flushed to disk. To minimize the chance of this occurring, set `sync_relay_log_info=1` in the slave `my.cnf` file. Setting `sync_relay_log_info` to 0 causes no writes to be forced to disk and the server relies on the operating system to flush the file from time to time.

The fault tolerance of your system for these types of problems is greatly increased if you have a good uninterruptible power supply.

17.5.1.28 Slave Errors During Replication

If a statement produces the same error (identical error code) on both the master and the slave, the error is logged, but replication continues.

If a statement produces different errors on the master and the slave, the slave SQL thread terminates, and the slave writes a message to its error log and waits for the database administrator to decide what to do about the error. This includes the case that a statement produces an error on the master or the slave, but not both. To address the issue, connect to the slave manually and determine the cause of the problem. `SHOW SLAVE STATUS` is useful for this. Then fix the problem and run `START SLAVE`. For example, you might need to create a nonexistent table before you can start the slave again.

Note

If a temporary error is recorded in the slave's error log, you do not necessarily have to take any action suggested in the quoted error message. Temporary errors should be handled by the client retrying the transaction. For example, if the slave SQL thread records a temporary error relating to a deadlock, you do not need to restart the transaction manually on the slave, unless the slave SQL thread subsequently terminates with a nontemporary error message.

If this error code validation behavior is not desirable, some or all errors can be masked out (ignored) with the `--slave-skip-errors` option.

For nontransactional storage engines such as `MyISAM`, it is possible to have a statement that only partially updates a table and returns an error code. This can happen, for example, on a multiple-row insert that has one row violating a key constraint, or if a long update statement is killed after updating some of the rows. If that happens on the master, the slave expects execution of the statement to result in the same error code. If it does not, the slave SQL thread stops as described previously.

If you are replicating between tables that use different storage engines on the master and slave, keep in mind that the same statement might produce a different error when run against one version of the table, but not the other, or might cause an error for one version of the table, but not the other. For example, since

MyISAM ignores foreign key constraints, an INSERT or UPDATE statement accessing an InnoDB table on the master might cause a foreign key violation but the same statement performed on a MyISAM version of the same table on the slave would produce no such error, causing replication to stop.

17.5.1.29 Replication and Server SQL Mode

Using different server SQL mode settings on the master and the slave may cause the same INSERT statements to be handled differently on the master and the slave, leading the master and slave to diverge. For best results, you should always use the same server SQL mode on the master and on the slave. This advice applies whether you are using statement-based or row-based replication.

If you are replicating partitioned tables, using different SQL modes on the master and the slave is likely to cause issues. At a minimum, this is likely to cause the distribution of data among partitions to be different in the master's and slave's copies of a given table. It may also cause inserts into partitioned tables that succeed on the master to fail on the slave.

For more information, see Section 5.1.11, “Server SQL Modes”.

17.5.1.30 Replication and Temporary Tables

In MySQL 8.0, when binlog_format is set to ROW or MIXED, statements that exclusively use temporary tables are not logged on the master, and therefore the temporary tables are not replicated. Statements that involve a mix of temporary and nontemporary tables are logged on the master only for the operations on nontemporary tables, and the operations on temporary tables are not logged. This means that there are never any temporary tables on the slave to be lost in the event of an unplanned shutdown by the slave. For more information about row-based replication and temporary tables, see Row-based logging of temporary tables.

When binlog_format is set to STATEMENT, operations on temporary tables are logged on the master and replicated on the slave, provided that the statements involving temporary tables can be logged safely using statement-based format. In this situation, loss of replicated temporary tables on the slave can be an issue. In statement-based replication mode, CREATE TEMPORARY TABLE and DROP TEMPORARY TABLE statements cannot be used inside a transaction, procedure, function, or trigger when GTIDs are in use on the server (that is, when the enforce_gtid_consistency system variable is set to ON). They can be used outside these contexts when GTIDs are in use, provided that autocommit=1 is set.

Because of the differences in behavior between row-based or mixed replication mode and statement-based replication mode regarding temporary tables, you cannot switch the replication format at runtime, if the change applies to a context (global or session) that contains any open temporary tables. For more details, see the description of the binlog_format option.

Safe slave shutdown when using temporary tables. In statement-based replication mode, temporary tables are replicated except in the case where you stop the slave server (not just the slave threads) and you have replicated temporary tables that are open for use in updates that have not yet been executed on the slave. If you stop the slave server, the temporary tables needed by those updates are no longer available when the slave is restarted. To avoid this problem, do not shut down the slave while it has temporary tables open. Instead, use the following procedure:

1. Issue a STOP SLAVE SQL_THREAD statement.
2. Use SHOW STATUS to check the value of the Slave_open_temp_tables variable.
3. If the value is not 0, restart the slave SQL thread with START SLAVE SQL_THREAD and repeat the procedure later.
4. When the value is 0, issue a mysqladmin shutdown command to stop the slave.

Temporary tables and replication options. By default, with statement-based replication, all temporary tables are replicated; this happens whether or not there are any matching `--replicate-do-db`, `--replicate-do-table`, or `--replicate-wild-do-table` options in effect. However, the `--replicate-ignore-table` and `--replicate-wild-ignore-table` options are honored for temporary tables. The exception is that to enable correct removal of temporary tables at the end of a session, a replication slave always replicates a `DROP TEMPORARY TABLE IF EXISTS` statement, regardless of any exclusion rules that would normally apply for the specified table.

A recommended practice when using statement-based replication is to designate a prefix for exclusive use in naming temporary tables that you do not want replicated, then employ a `--replicate-wild-ignore-table` option to match that prefix. For example, you might give all such tables names beginning with `norep` (such as `norepmytable`, `norepyourtable`, and so on), then use `--replicate-wild-ignore-table=norep%` to prevent them from being replicated.

17.5.1.31 Replication Retries and Timeouts

The global system variable `slave_transaction_retries` sets the maximum number of times for applier threads on a single-threaded or multithreaded replication slave to automatically retry failed transactions before stopping. Transactions are automatically retried when the SQL thread fails to execute them because of an `InnoDB` deadlock, or when the transaction's execution time exceeds the `InnoDB` `innodb_lock_wait_timeout` value. If a transaction has a non-temporary error that will prevent it from ever succeeding, it is not retried.

The default setting for `slave_transaction_retries` is 10, meaning that a failing transaction with an apparently temporary error is retried 10 times before the applier thread stops. Setting the variable to 0 disables automatic retrying of transactions. On a multithreaded slave, the specified number of transaction retries can take place on all applier threads of all channels. The Performance Schema table `replication_applier_status` shows the total number of transaction retries that took place on each replication channel, in the `COUNT_TRANSACTIONS_RETRIES` column.

The process of retrying transactions can cause lag on a replication slave or on a Group Replication group member, which can be configured as a single-threaded or multithreaded slave. The Performance Schema table `replication_applier_status_by_worker` shows detailed information on transaction retries by the applier threads on a single-threaded or multithreaded slave. This data includes timestamps showing how long it took the applier thread to apply the last transaction from start to finish (and when the transaction currently in progress was started), and how long this was after the commit on the original master and the immediate master. The data also shows the number of retries for the last transaction and the transaction currently in progress, and enables you to identify the transient errors that caused the transactions to be retried. You can use this information to see whether transaction retries are the cause of replication lag, and investigate the root cause of the failures that led to the retries.

17.5.1.32 Replication and Time Zones

By default, master and slave servers assume that they are in the same time zone. If you are replicating between servers in different time zones, the time zone must be set on both master and slave. Otherwise, statements depending on the local time on the master are not replicated properly, such as statements that use the `NOW()` or `FROM_UNIXTIME()` functions. Set the time zone in which MySQL server runs by using the `--timezone=`*`timezone_name`* option of the `mysqld_safe` script or by setting the `TZ` environment variable. See also Section 17.5.1.14, “Replication and System Functions”.

17.5.1.33 Replication and Transaction Inconsistencies

Inconsistencies in the sequence of transactions that have been executed from the relay log can occur depending on your replication configuration. This section explains how to avoid inconsistencies and solve any problems they cause.

The following types of inconsistencies can exist:

- *Half-applied transactions*. A transaction which updates non-transactional tables has applied some but not all of its changes.
- *Gaps*. A gap is a transaction that has not been fully applied, even though some transaction later in the sequence has been applied. Gaps can only appear when using a multithreaded slave. To avoid gaps occurring, set `slave_preserve_commit_order=1`, which requires `slave_parallel_type=LOGICAL_CLOCK`, and that binary logging (the `log_bin` system variable) and slave update logging (the `--log-slave-updates` option) are also enabled. Note that `slave_preserve_commit_order=1` does not preserve the order of non-transactional DML updates, so these might commit before transactions that precede them in the relay log, which might result in gaps.
- *Master log position lag*. Even in the absence of gaps, it is possible that transactions after `Exec_master_log_pos` have been applied. That is, all transactions up to point `N` have been applied, and no transactions after `N` have been applied, but `Exec_master_log_pos` has a value smaller than `N`. In this situation, `Exec_master_log_pos` is a “low-water mark” of the transactions applied, and lags behind the position of the most recently applied transaction. This can only happen on multithreaded slaves. Enabling `slave_preserve_commit_order` does not prevent master log position lag.

The following scenarios are relevant to the existence of half-applied transactions, gaps, and master log position lag:

1. While slave threads are running, there may be gaps and half-applied transactions.
2. `mysqld` shuts down. Both clean and unclean shutdown abort ongoing transactions and may leave gaps and half-applied transactions.
3. `KILL` of replication threads (the SQL thread when using a single-threaded slave, the coordinator thread when using a multithreaded slave). This aborts ongoing transactions and may leave gaps and half-applied transactions.
4. Error in applier threads. This may leave gaps. If the error is in a mixed transaction, that transaction is half-applied. When using a multithreaded slave, workers which have not received an error complete their queues, so it may take time to stop all threads.
5. `STOP SLAVE` when using a multithreaded slave. After issuing `STOP SLAVE`, the slave waits for any gaps to be filled and then updates `Exec_master_log_pos`. This ensures it never leaves gaps or master log position lag, unless any of the cases above applies, in other words, before `STOP SLAVE` completes, either an error happens, or another thread issues `KILL`, or the server restarts. In these cases, `STOP SLAVE` returns successfully.
6. If the last transaction in the relay log is only half-received and the multithreaded slave coordinator has started to schedule the transaction to a worker, then `STOP SLAVE` waits up to 60 seconds for the transaction to be received. After this timeout, the coordinator gives up and aborts the transaction. If the transaction is mixed, it may be left half-completed.
7. `STOP SLAVE` when using a single-threaded slave. If the ongoing transaction only updates transactional tables, it is rolled back and `STOP SLAVE` stops immediately. If the ongoing transaction is mixed, `STOP SLAVE` waits up to 60 seconds for the transaction to complete. After this timeout, it aborts the transaction, so it may be left half-completed.

The global variable `rpl_stop_slave_timeout` is unrelated to the process of stopping the replication threads. It only makes the client that issues `STOP SLAVE` return to the client, but the replication threads continue to try to stop.

If a replication channel has gaps, it has the following consequences:

1. The slave database is in a state that may never have existed on the master.

2. The field `Exec_master_log_pos` in `SHOW SLAVE STATUS` is only a “low-water mark”. In other words, transactions appearing before the position are guaranteed to have committed, but transactions after the position may have committed or not.

3. `CHANGE MASTER TO` statements for that channel fail with an error, unless the applier threads are running and the `CHANGE MASTER TO` statement only sets receiver options.

4. If `mysqld` is started with `--relay-log-recovery`, no recovery is done for that channel, and a warning is printed.

5. If `mysqldump` is used with `--dump-slave`, it does not record the existence of gaps; thus it prints `CHANGE MASTER TO` with `RELAY_LOG_POS` set to the “low-water mark” position in `Exec_master_log_pos`.

   After applying the dump on another server, and starting the replication threads, transactions appearing after the position are replicated again. Note that this is harmless if GTIDs are enabled (however, in that case it is not recommended to use `--dump-slave`).

If a replication channel has master log position lag but no gaps, cases 2 to 5 above apply, but case 1 does not.

The master log position information is persisted in binary format in the internal table `mysql.slave_worker_info`. `START SLAVE [SQL_THREAD]` always consults this information so that it applies only the correct transactions. This remains true even if `slave_parallel_workers` has been changed to 0 before `START SLAVE`, and even if `START SLAVE` is used with `UNTIL` clauses. `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` only applies as many transactions as needed in order to fill in the gaps. If `START SLAVE` is used with `UNTIL` clauses that tell it to stop before it has consumed all the gaps, then it leaves remaining gaps.

Warning

`RESET SLAVE` removes the relay logs and resets the replication position. Thus issuing `RESET SLAVE` on a slave with gaps means the slave loses any information about the gaps, without correcting the gaps.

17.5.1.34 Replication and Transactions

Mixing transactional and nontransactional statements within the same transaction. In general, you should avoid transactions that update both transactional and nontransactional tables in a replication environment. You should also avoid using any statement that accesses both transactional (or temporary) and nontransactional tables and writes to any of them.

The server uses these rules for binary logging:

- If the initial statements in a transaction are nontransactional, they are written to the binary log immediately. The remaining statements in the transaction are cached and not written to the binary log until the transaction is committed. (If the transaction is rolled back, the cached statements are written to the binary log only if they make nontransactional changes that cannot be rolled back. Otherwise, they are discarded.)

- For statement-based logging, logging of nontransactional statements is affected by the `binlog_direct_non_transactional_updates` system variable. When this variable is `OFF` (the default), logging is as just described. When this variable is `ON`, logging occurs immediately for nontransactional statements occurring anywhere in the transaction (not just initial nontransactional statements). Other statements are kept in the transaction cache and logged when the transaction

commits. `binlog_direct_non_transactional_updates` has no effect for row-format or mixed-format binary logging.

Transactional, nontransactional, and mixed statements.
To apply those rules, the server considers a statement nontransactional if it changes only nontransactional tables, and transactional if it changes only transactional tables. A statement that references both nontransactional and transactional tables and updates *any* of the tables involved is considered a “mixed” statement. Mixed statements, like transactional statements, are cached and logged when the transaction commits.

A mixed statement that updates a transactional table is considered unsafe if the statement also performs either of the following actions:

- Updates or reads a temporary table
- Reads a nontransactional table and the transaction isolation level is less than REPEATABLE_READ

A mixed statement following the update of a transactional table within a transaction is considered unsafe if it performs either of the following actions:

- Updates any table and reads from any temporary table
- Updates a nontransactional table and `binlog_direct_non_transactional_updates` is OFF

For more information, see Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”.

Note

A mixed statement is unrelated to mixed binary logging format.

In situations where transactions mix updates to transactional and nontransactional tables, the order of statements in the binary log is correct, and all needed statements are written to the binary log even in case of a `ROLLBACK`. However, when a second connection updates the nontransactional table before the first connection transaction is complete, statements can be logged out of order because the second connection update is written immediately after it is performed, regardless of the state of the transaction being performed by the first connection.

Using different storage engines on master and slave. It is possible to replicate transactional tables on the master using nontransactional tables on the slave. For example, you can replicate an `InnoDB` master table as a `MyISAM` slave table. However, if you do this, there are problems if the slave is stopped in the middle of a `BEGIN` ... `COMMIT` block because the slave restarts at the beginning of the `BEGIN` block.

It is also safe to replicate transactions from `MyISAM` tables on the master to transactional tables—such as tables that use the `InnoDB` storage engine—on the slave. In such cases, an `AUTOCOMMIT=1` statement issued on the master is replicated, thus enforcing `AUTOCOMMIT` mode on the slave.

When the storage engine type of the slave is nontransactional, transactions on the master that mix updates of transactional and nontransactional tables should be avoided because they can cause inconsistency of the data between the master transactional table and the slave nontransactional table. That is, such transactions can lead to master storage engine-specific behavior with the possible effect of replication going out of synchrony. MySQL does not issue a warning about this, so extra care should be taken when replicating transactional tables from the master to nontransactional tables on the slaves.

Changing the binary logging format within transactions. The `binlog_format` and `binlog_checksum` system variables are read-only as long as a transaction is in progress.

Every transaction (including `autocommit` transactions) is recorded in the binary log as though it starts with a `BEGIN` statement, and ends with either a `COMMIT` or a `ROLLBACK` statement. This is even true for statements affecting tables that use a nontransactional storage engine (such as `MyISAM`).

Note

For restrictions that apply specifically to XA transactions, see Section 13.3.8.3, “Restrictions on XA Transactions”.

17.5.1.35 Replication and Triggers

With statement-based replication, triggers executed on the master also execute on the slave. With row-based replication, triggers executed on the master do not execute on the slave. Instead, the row changes on the master resulting from trigger execution are replicated and applied on the slave.

This behavior is by design. If under row-based replication the slave applied the triggers as well as the row changes caused by them, the changes would in effect be applied twice on the slave, leading to different data on the master and the slave.

If you want triggers to execute on both the master and the slave—perhaps because you have different triggers on the master and slave—you must use statement-based replication. However, to enable slave-side triggers, it is not necessary to use statement-based replication exclusively. It is sufficient to switch to statement-based replication only for those statements where you want this effect, and to use row-based replication the rest of the time.

A statement invoking a trigger (or function) that causes an update to an `AUTO_INCREMENT` column is not replicated correctly using statement-based replication. MySQL 8.0 marks such statements as unsafe. (Bug #45677)

A trigger can have triggers for different combinations of trigger event (`INSERT`, `UPDATE`, `DELETE`) and action time (`BEFORE`, `AFTER`), and multiple triggers are permitted.

For brevity, “multiple triggers” here is shorthand for “multiple triggers that have the same trigger event and action time.”

Upgrades. Multiple triggers are not supported in versions earlier than MySQL 5.7. If you upgrade servers in a replication topology that use a version earlier than MySQL 5.7, upgrade the replication slaves first and then upgrade the master. If an upgraded replication master still has old slaves using MySQL versions that do not support multiple triggers, an error occurs on those slaves if a trigger is created on the master for a table that already has a trigger with the same trigger event and action time.

Downgrades. If you downgrade a server that supports multiple triggers to an older version that does not, the downgrade has these effects:

- For each table that has triggers, all trigger definitions are in the `.TRG` file for the table. However, if there are multiple triggers with the same trigger event and action time, the server executes only one of them when the trigger event occurs. For information about `.TRG` files, see the Table Trigger Storage section of the MySQL Server Doxygen documentation, available at https://dev.mysql.com/doc/index-other.html.
- If triggers for the table are added or dropped subsequent to the downgrade, the server rewrites the table's `.TRG` file. The rewritten file retains only one trigger per combination of trigger event and action time; the others are lost.

To avoid these problems, modify your triggers before downgrading. For each table that has multiple triggers per combination of trigger event and action time, convert each such set of triggers to a single trigger as follows:

1. For each trigger, create a stored routine that contains all the code in the trigger. Values accessed using `NEW` and `OLD` can be passed to the routine using parameters. If the trigger needs a single result value from the code, you can put the code in a stored function and have the function return the value. If the trigger needs multiple result values from the code, you can put the code in a stored procedure and return the values using `OUT` parameters.

2. Drop all triggers for the table.

3. Create one new trigger for the table that invokes the stored routines just created. The effect for this trigger is thus the same as the multiple triggers it replaces.

17.5.1.36 Replication and TRUNCATE TABLE

`TRUNCATE TABLE` is normally regarded as a DML statement, and so would be expected to be logged and replicated using row-based format when the binary logging mode is `ROW` or `MIXED`. However this caused issues when logging or replicating, in `STATEMENT` or `MIXED` mode, tables that used transactional storage engines such as `InnoDB` when the transaction isolation level was `READ COMMITTED` or `READ UNCOMMITTED`, which precludes statement-based logging.

`TRUNCATE TABLE` is treated for purposes of logging and replication as DDL rather than DML so that it can be logged and replicated as a statement. However, the effects of the statement as applicable to `InnoDB` and other transactional tables on replication slaves still follow the rules described in Section 13.1.37, “TRUNCATE TABLE Syntax” governing such tables. (Bug #36763)

17.5.1.37 Replication and User Name Length

The maximum length of MySQL user names is 32 characters. Replication of user names longer than 16 characters to a slave earlier than MySQL 5.7 that supports only shorter user names will fail. However, this should occur only when replicating from a newer master to an older slave, which is not a recommended configuration.

17.5.1.38 Replication and Variables

System variables are not replicated correctly when using `STATEMENT` mode, except for the following variables when they are used with session scope:

- `auto_increment_increment`
- `auto_increment_offset`
- `character_set_client`
- `character_set_connection`
- `character_set_database`
- `character_set_server`
- `collation_connection`
- `collation_database`
- `collation_server`
- `foreign_key_checks`
- `identity`

- last_insert_id
- lc_time_names
- pseudo_thread_id
- sql_auto_is_null
- time_zone
- timestamp
- unique_checks

When MIXED mode is used, the variables in the preceding list, when used with session scope, cause a switch from statement-based to row-based logging. See Section 5.4.4.3, “Mixed Binary Logging Format”.

sql_mode is also replicated except for the NO_DIR_IN_CREATE mode; the slave always preserves its own value for NO_DIR_IN_CREATE, regardless of changes to it on the master. This is true for all replication formats.

However, when mysqlbinlog parses a SET @@sql_mode = *mode* statement, the full *mode* value, including NO_DIR_IN_CREATE, is passed to the receiving server. For this reason, replication of such a statement may not be safe when STATEMENT mode is in use.

The default_storage_engine system variable is not replicated, regardless of the logging mode; this is intended to facilitate replication between different storage engines.

The read_only system variable is not replicated. In addition, the enabling this variable has different effects with regard to temporary tables, table locking, and the SET PASSWORD statement in different MySQL versions.

The max_heap_table_size system variable is not replicated. Increasing the value of this variable on the master without doing so on the slave can lead eventually to Table is full errors on the slave when trying to execute INSERT statements on a MEMORY table on the master that is thus permitted to grow larger than its counterpart on the slave. For more information, see Section 17.5.1.21, “Replication and MEMORY Tables”.

In statement-based replication, session variables are not replicated properly when used in statements that update tables. For example, the following sequence of statements will not insert the same data on the master and the slave:

```
SET max_join_size=1000;
INSERT INTO mytable VALUES(@@max_join_size);
```

This does not apply to the common sequence:

```
SET time_zone=...;
INSERT INTO mytable VALUES(CONVERT_TZ(..., ..., @@time_zone));
```

Replication of session variables is not a problem when row-based replication is being used, in which case, session variables are always replicated safely. See Section 17.2.1, “Replication Formats”.

The following session variables are written to the binary log and honored by the replication slave when parsing the binary log, regardless of the logging format:

- `sql_mode`
- `foreign_key_checks`
- `unique_checks`
- `character_set_client`
- `collation_connection`
- `collation_database`
- `collation_server`
- `sql_auto_is_null`

Important

Even though session variables relating to character sets and collations are written to the binary log, replication between different character sets is not supported.

To help reduce possible confusion, we recommend that you always use the same setting for the `lower_case_table_names` system variable on both master and slave, especially when you are running MySQL on platforms with case-sensitive file systems. The `lower_case_table_names` setting can only be configured when initializing the server.

17.5.1.39 Replication and Views

Views are always replicated to slaves. Views are filtered by their own name, not by the tables they refer to. This means that a view can be replicated to the slave even if the view contains a table that would normally be filtered out by `replication-ignore-table` rules. Care should therefore be taken to ensure that views do not replicate table data that would normally be filtered for security reasons.

Replication from a table to a same-named view is supported using statement-based logging, but not when using row-based logging. Trying to do so when row-based logging is in effect causes an error.

17.5.2 Replication Compatibility Between MySQL Versions

MySQL supports replication from one release series to the next higher release series. For example, you can replicate from a master running MySQL 5.6 to a slave running MySQL 5.7, from a master running MySQL 5.7 to a slave running MySQL 8.0, and so on. However, you might encounter difficulties when replicating from an older master to a newer slave if the master uses statements or relies on behavior no longer supported in the version of MySQL used on the slave. For example, foreign key names longer than 64 characters are no longer supported from MySQL 8.0.

The use of more than two MySQL Server versions is not supported in replication setups involving multiple masters, regardless of the number of master or slave MySQL servers. This restriction applies not only to release series, but to version numbers within the same release series as well. For example, if you are using a chained or circular replication setup, you cannot use MySQL 8.0.1, MySQL 8.0.2, and MySQL 8.0.4 concurrently, although you could use any two of these releases together.

Important

It is strongly recommended to use the most recent release available within a given MySQL release series because replication (and other) capabilities are continually being improved. It is also recommended to upgrade masters and slaves that use

early releases of a release series of MySQL to GA (production) releases when the latter become available for that release series.

From MySQL 8.0.14, the server version is recorded in the binary log for each transaction for the server that originally committed the transaction (`original_server_version`), and for the server that is the immediate master of the current server in the replication topology (`immediate_server_version`).

Replication from newer masters to older slaves might be possible, but is generally not supported. This is due to a number of factors:

- **Binary log format changes.** The binary log format can change between major releases. While we attempt to maintain backward compatibility, this is not always possible.

  This also has significant implications for upgrading replication servers; see Section 17.5.3, “Upgrading a Replication Setup”, for more information.

- For more information about row-based replication, see Section 17.2.1, “Replication Formats”.

- **SQL incompatibilities.** You cannot replicate from a newer master to an older slave using statement-based replication if the statements to be replicated use SQL features available on the master but not on the slave.

  However, if both the master and the slave support row-based replication, and there are no data definition statements to be replicated that depend on SQL features found on the master but not on the slave, you can use row-based replication to replicate the effects of data modification statements even if the DDL run on the master is not supported on the slave.

For more information on potential replication issues, see Section 17.5.1, “Replication Features and Issues”.

17.5.3 Upgrading a Replication Setup

When you upgrade servers that participate in a replication setup, the procedure for upgrading depends on the current server versions and the version to which you are upgrading. This section provides information about how upgrading affects replication. For general information about upgrading MySQL, see Section 2.11, “Upgrading MySQL”

When you upgrade a master to 8.0 from an earlier MySQL release series, you should first ensure that all the slaves of this master are using the same 8.0.x release. If this is not the case, you should first upgrade the slaves. To upgrade each slave, shut it down, upgrade it to the appropriate 8.0.x version, restart it, and restart replication. Relay logs created by the slave after the upgrade are in 8.0 format.

Changes affecting operations in strict SQL mode (`STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES`) may result in replication failure on an upgraded slave. If you use statement-based logging (`binlog_format=STATEMENT`), if a slave is upgraded before the master, the nonupgraded master will execute statements without error that may fail on the slave and replication will stop. To deal with this, stop all new statements on the master and wait until the slaves catch up. Then upgrade the slaves. Alternatively, if you cannot stop new statements, temporarily change to row-based logging on the master (`binlog_format=ROW`) and wait until all slaves have processed all binary logs produced up to the point of this change. Then upgrade the slaves.

The default character set has changed from `latin1` to `utf8mb4` in MySQL 8.0. In a replicated setting, when upgrading from MySQL 5.7 to 8.0, it is advisable to change the default character set back to the character set used in MySQL 5.7 before upgrading. After the upgrade is completed, the default character set can be changed to `utf8mb4`. Assuming that the previous defaults were used, one way to preserve them is to start the server with these lines in the `my.cnf` file:

```
[mysqld]
character_set_server=latin1
collation_server=latin1_swedish_ci
```

After the slaves have been upgraded, shut down the master, upgrade it to the same 8.0.x release as the slaves, and restart it. If you had temporarily changed the master to row-based logging, change it back to statement-based logging. The 8.0 master is able to read the old binary logs written prior to the upgrade and to send them to the 8.0 slaves. The slaves recognize the old format and handle it properly. Binary logs created by the master subsequent to the upgrade are in 8.0 format. These too are recognized by the 8.0 slaves.

In other words, when upgrading to MySQL 8.0, the slaves must be MySQL 8.0 before you can upgrade the master to 8.0. Note that downgrading from 8.0 to older versions does not work so simply: You must ensure that any 8.0 binary log or relay log has been fully processed, so that you can remove it before proceeding with the downgrade.

Some upgrades may require that you drop and re-create database objects when you move from one MySQL series to the next. For example, collation changes might require that table indexes be rebuilt. Such operations, if necessary, are detailed at Section 2.11.4, “Changes in MySQL 8.0”. It is safest to perform these operations separately on the slaves and the master, and to disable replication of these operations from the master to the slave. To achieve this, use the following procedure:

1. Stop all the slaves and upgrade them. Restart them with the `--skip-slave-start` option so that they do not connect to the master. Perform any table repair or rebuilding operations needed to re-create database objects, such as use of `REPAIR TABLE` or `ALTER TABLE`, or dumping and reloading tables or triggers.

2. Disable the binary log on the master. To do this without restarting the master, execute a `SET sql_log_bin = OFF` statement. Alternatively, stop the master and restart it with the `--skip-log-bin` option. If you restart the master, you might also want to disallow client connections. For example, if all clients connect using TCP/IP, use the `--skip-networking` option when you restart the master.

3. With the binary log disabled, perform any table repair or rebuilding operations needed to re-create database objects. The binary log must be disabled during this step to prevent these operations from being logged and sent to the slaves later.

4. Re-enable the binary log on the master. If you set `sql_log_bin` to `OFF` earlier, execute a `SET sql_log_bin = ON` statement. If you restarted the master to disable the binary log, restart it without `--skip-log-bin`, and without `--skip-networking` so that clients and slaves can connect.

5. Restart the slaves, this time without the `--skip-slave-start` option.

If you are upgrading an existing replication setup from a version of MySQL that does not support global transaction identifiers to a version that does, you should not enable GTIDs on either the master or the slave before making sure that the setup meets all the requirements for GTID-based replication. See Section 17.1.3.4, “Setting Up Replication Using GTIDs”, which contains information about converting existing replication setups to use GTID-based replication.

Prior to MySQL 8.0.16, when the server is running with global transaction identifiers (GTIDs) enabled (`gtid_mode=ON`), do not enable binary logging by `mysql_upgrade` (the `--write-binlog` option). As of MySQL 8.0.16, the server performs the entire MySQL upgrade procedure, but disables binary logging during the upgrade, so there is no issue.

It is not recommended to load a dump file when GTIDs are enabled on the server (`gtid_mode=ON`), if your dump file includes system tables. `mysqldump` issues DML instructions for the system tables which use the non-transactional MyISAM storage engine, and this combination is not permitted when GTIDs are enabled.

Also be aware that loading a dump file from a server with GTIDs enabled, into another server with GTIDs enabled, causes different transaction identifiers to be generated.

17.5.4 Troubleshooting Replication

If you have followed the instructions but your replication setup is not working, the first thing to do is *check the error log for messages*. Many users have lost time by not doing this soon enough after encountering problems.

If you cannot tell from the error log what the problem was, try the following techniques:

- Verify that the master has binary logging enabled by issuing a `SHOW MASTER STATUS` statement. Binary logging is enabled by default. If binary logging is enabled, `Position` is nonzero. If binary logging is not enabled, verify that you are not running the master with any settings that disable binary logging, such as the `--skip-log-bin` option.
- Verify that the master and slave both were started with the `--server-id` option and that the ID value is unique on each server.
- Verify that the slave is running. Use `SHOW SLAVE STATUS` to check whether the `Slave_IO_Running` and `Slave_SQL_Running` values are both `Yes`. If not, verify the options that were used when starting the slave server. For example, `--skip-slave-start` prevents the slave threads from starting until you issue a `START SLAVE` statement.
- If the slave is running, check whether it established a connection to the master. Use `SHOW PROCESSLIST`, find the I/O and SQL threads and check their `State` column to see what they display. See Section 17.2.2, “Replication Implementation Details”. If the I/O thread state says `Connecting to master`, check the following:
  - Verify the privileges for the user being used for replication on the master.
  - Check that the host name of the master is correct and that you are using the correct port to connect to the master. The port used for replication is the same as used for client network communication (the default is `3306`). For the host name, ensure that the name resolves to the correct IP address.
  - Check that networking has not been disabled on the master or slave. Look for the `skip-networking` option in the configuration file. If present, comment it out or remove it.
  - If the master has a firewall or IP filtering configuration, ensure that the network port being used for MySQL is not being filtered.
  - Check that you can reach the master by using `ping` or `traceroute`/`tracert` to reach the host.
- If the slave was running previously but has stopped, the reason usually is that some statement that succeeded on the master failed on the slave. This should never happen if you have taken a proper snapshot of the master, and never modified the data on the slave outside of the slave thread. If the slave stops unexpectedly, it is a bug or you have encountered one of the known replication limitations described in Section 17.5.1, “Replication Features and Issues”. If it is a bug, see Section 17.5.5, “How to Report Replication Bugs or Problems”, for instructions on how to report it.
- If a statement that succeeded on the master refuses to run on the slave, try the following procedure if it is not feasible to do a full database resynchronization by deleting the slave's databases and copying a new snapshot from the master:
  1. Determine whether the affected table on the slave is different from the master table. Try to understand how this happened. Then make the slave's table identical to the master's and run `START SLAVE`.

2. If the preceding step does not work or does not apply, try to understand whether it would be safe to make the update manually (if needed) and then ignore the next statement from the master.

3. If you decide that the slave can skip the next statement from the master, issue the following statements:

```
mysql> SET GLOBAL sql_slave_skip_counter = N;
mysql> START SLAVE;
```

   The value of *N* should be 1 if the next statement from the master does not use `AUTO_INCREMENT` or `LAST_INSERT_ID()`. Otherwise, the value should be 2. The reason for using a value of 2 for statements that use `AUTO_INCREMENT` or `LAST_INSERT_ID()` is that they take two events in the binary log of the master.

   See also Section 13.4.2.5, “SET GLOBAL sql_slave_skip_counter Syntax”.

4. If you are sure that the slave started out perfectly synchronized with the master, and that no one has updated the tables involved outside of the slave thread, then presumably the discrepancy is the result of a bug. If you are running the most recent version of MySQL, please report the problem. If you are running an older version, try upgrading to the latest production release to determine whether the problem persists.

17.5.5 How to Report Replication Bugs or Problems

When you have determined that there is no user error involved, and replication still either does not work at all or is unstable, it is time to send us a bug report. We need to obtain as much information as possible from you to be able to track down the bug. Please spend some time and effort in preparing a good bug report.

If you have a repeatable test case that demonstrates the bug, please enter it into our bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”. If you have a “phantom” problem (one that you cannot duplicate at will), use the following procedure:

1. Verify that no user error is involved. For example, if you update the slave outside of the slave thread, the data goes out of synchrony, and you can have unique key violations on updates. In this case, the slave thread stops and waits for you to clean up the tables manually to bring them into synchrony. *This is not a replication problem. It is a problem of outside interference causing replication to fail.*

2. Ensure that the slave is running with binary logging enabled (the `log_bin` system variable), and with the `--log-slave-updates` option enabled, which causes the slave to log the updates that it receives from the master into its own binary logs. These settings are the defaults.

3. Save all evidence before resetting the replication state. If we have no information or only sketchy information, it becomes difficult or impossible for us to track down the problem. The evidence you should collect is:

   - All binary log files from the master
   - All binary log files from the slave
   - The output of `SHOW MASTER STATUS` from the master at the time you discovered the problem
   - The output of `SHOW SLAVE STATUS` from the slave at the time you discovered the problem
   - Error logs from the master and the slave

4. Use `mysqlbinlog` to examine the binary logs. The following should be helpful to find the problem statement. *`log_file`* and *`log_pos`* are the `Master_Log_File` and `Read_Master_Log_Pos` values from `SHOW SLAVE STATUS`.

```
shell> mysqlbinlog --start-position=log_pos log_file | head
```

After you have collected the evidence for the problem, try to isolate it as a separate test case first. Then enter the problem with as much information as possible into our bugs database using the instructions at Section 1.7, “How to Report Bugs or Problems”.

Chapter 18 Group Replication

Table of Contents

18.1 Group Replication Background 3398
18.1.1 Replication Technologies 3399
18.1.2 Group Replication Use Cases 3401
18.1.3 Multi-Primary and Single-Primary Modes 3402
18.1.4 Group Replication Services 3406
18.1.5 Group Replication Plugin Architecture 3409
18.2 Getting Started 3410
18.2.1 Deploying Group Replication in Single-Primary Mode 3410
18.2.2 Deploying Group Replication Locally 3422
18.3 Monitoring Group Replication 3424
18.3.1 Group Replication Server States 3424
18.3.2 The replication_group_members Table 3425
18.3.3 The replication_group_member_stats Table 3426
18.4 Group Replication Operations 3426
18.4.1 Configuring an Online Group 3426
18.4.2 Transaction Consistency Guarantees 3431
18.4.3 Distributed Recovery 3437
18.4.4 Network Partitioning 3449
18.4.5 Support For IPv6 And For Mixed IPv6 And IPv4 Groups 3454
18.4.6 Using MySQL Enterprise Backup with Group Replication 3456
18.5 Group Replication Security 3462
18.5.1 Group Replication IP Address Whitelisting 3462
18.5.2 Group Replication Secure Socket Layer (SSL) Support 3464
18.6 Group Replication Performance 3468
18.6.1 Fine Tuning the Group Communication Thread 3468
18.6.2 Flow Control 3468
18.6.3 Message Compression 3470
18.6.4 Message Fragmentation 3472
18.6.5 XCom Cache Management 3472
18.6.6 Responses to Failure Detection and Network Partitioning 3473
18.7 Upgrading Group Replication 3478
18.7.1 Combining Different Member Versions in a Group 3479
18.7.2 Group Replication Offline Upgrade 3481
18.7.3 Group Replication Online Upgrade 3482
18.8 Group Replication System Variables 3485
18.9 Requirements and Limitations 3518
18.9.1 Group Replication Requirements 3518
18.9.2 Group Replication Limitations 3520
18.10 Frequently Asked Questions 3522

This chapter explains MySQL Group Replication and how to install, configure and monitor groups. MySQL Group Replication is a MySQL Server plugin that enables you to create elastic, highly-available, fault-tolerant replication topologies.

Groups can operate in a single-primary mode with automatic primary election, where only one server accepts updates at a time. Alternatively, for more advanced users, groups can be deployed in multi-primary mode, where all servers can accept updates, even if they are issued concurrently.

There is a built-in group membership service that keeps the view of the group consistent and available for all servers at any given point in time. Servers can leave and join the group and the view is updated accordingly. Sometimes servers can leave the group unexpectedly, in which case the failure detection mechanism detects this and notifies the group that the view has changed. This is all automatic.

The chapter is structured as follows:

- Section 18.1, “Group Replication Background” provides an introduction to groups and how Group Replication works.
- Section 18.2, “Getting Started” explains how to configure multiple MySQL Server instances to create a group.
- Section 18.3, “Monitoring Group Replication” explains how to monitor a group.
- Section 18.4, “Group Replication Operations” explains how to work with a group.
- Section 18.5, “Group Replication Security” explains how to secure a group.
- Section 18.6, “Group Replication Performance” explains how to fine tune performance for a group.
- Section 18.7, “Upgrading Group Replication” explains how to upgrade a group.
- Section 18.8, “Group Replication System Variables” is a reference for the system variables specific to Group Replication.
- Section 18.9, “Requirements and Limitations” explains technical requirements and limitations for Group Replication.
- Section 18.10, “Frequently Asked Questions” provides answers to some technical questions about deploying and operating Group Replication.

18.1 Group Replication Background

This section provides background information on MySQL Group Replication.

The most common way to create a fault-tolerant system is to resort to making components redundant, in other words the component can be removed and the system should continue to operate as expected. This creates a set of challenges that raise complexity of such systems to a whole different level. Specifically, replicated databases have to deal with the fact that they require maintenance and administration of several servers instead of just one. Moreover, as servers are cooperating together to create the group several other classic distributed systems problems have to be dealt with, such as network partitioning or split brain scenarios.

Therefore, the ultimate challenge is to fuse the logic of the database and data replication with the logic of having several servers coordinated in a consistent and simple way. In other words, to have multiple servers agreeing on the state of the system and the data on each and every change that the system goes through. This can be summarized as having servers reaching agreement on each database state transition, so that they all progress as one single database or alternatively that they eventually converge to the same state. Meaning that they need to operate as a (distributed) state machine.

MySQL Group Replication provides distributed state machine replication with strong coordination between servers. Servers coordinate themselves automatically when they are part of the same group. The group can operate in a single-primary mode with automatic primary election, where only one server accepts updates at a time. Alternatively, for more advanced users the group can be deployed in multi-primary mode, where all servers can accept updates, even if they are issued concurrently. This power comes at the expense of applications having to work around the limitations imposed by such deployments.

There is a built-in group membership service that keeps the view of the group consistent and available for all servers at any given point in time. Servers can leave and join the group and the view is updated accordingly. Sometimes servers can leave the group unexpectedly, in which case the failure detection mechanism detects this and notifies the group that the view has changed. This is all automatic.

For a transaction to commit, the majority of the group have to agree on the order of a given transaction in the global sequence of transactions. Deciding to commit or abort a transaction is done by each server individually, but all servers make the same decision. If there is a network partition, resulting in a split where members are unable to reach agreement, then the system does not progress until this issue is resolved. Hence there is also a built-in, automatic, split-brain protection mechanism.

All of this is powered by the provided Group Communication System (GCS) protocols. These provide a failure detection mechanism, a group membership service, and safe and completely ordered message delivery. All these properties are key to creating a system which ensures that data is consistently replicated across the group of servers. At the very core of this technology lies an implementation of the Paxos algorithm. It acts as the group communication engine.

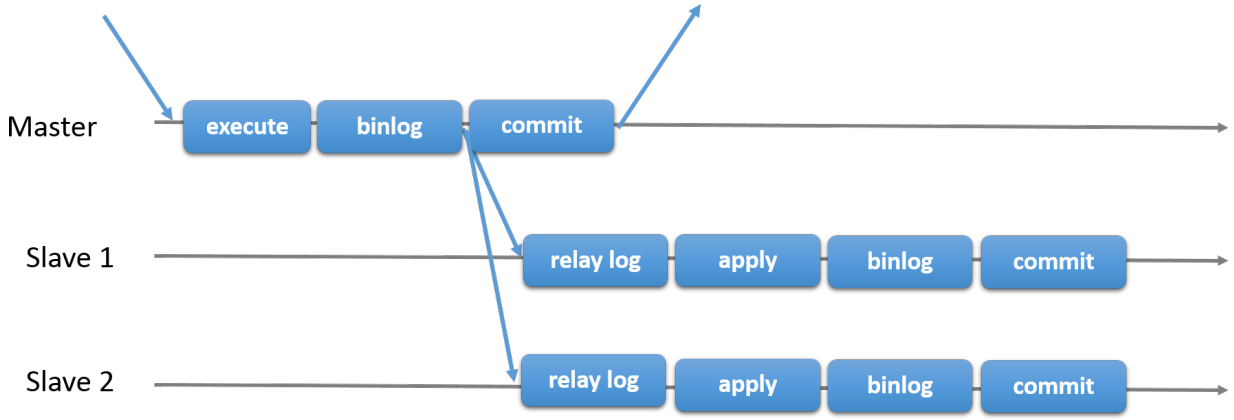
18.1.1 Replication Technologies

Before getting into the details of MySQL Group Replication, this section introduces some background concepts and an overview of how things work. This provides some context to help understand what is required for Group Replication and what the differences are between classic asynchronous MySQL Replication and Group Replication.

18.1.1.1 Primary-Secondary Replication

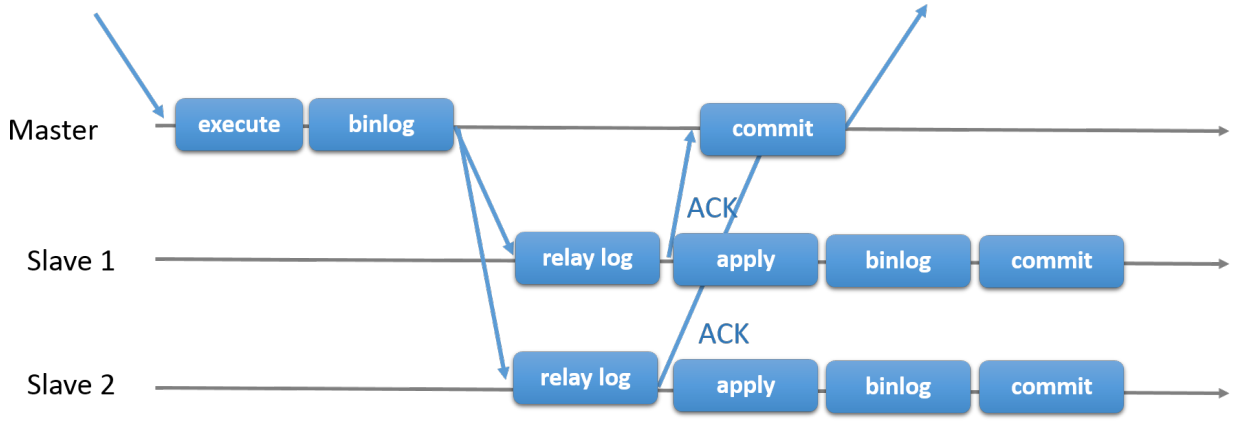
Traditional MySQL Replication provides a simple Primary-Secondary approach to replication. There is a primary (master) and there is one or more secondaries (slaves). The primary executes transactions, commits them and then they are later (thus asynchronously) sent to the secondaries to be either re-executed (in statement-based replication) or applied (in row-based replication). It is a shared-nothing system, where all servers have a full copy of the data by default.

Figure 18.1 MySQL Asynchronous Replication



There is also semisynchronous replication, which adds one synchronization step to the protocol. This means that the Primary waits, at commit time, for the secondary to acknowledge that it has *received* the transaction. Only then does the Primary resume the commit operation.

Figure 18.2 MySQL Semisynchronous Replication



In the two pictures above, you can see a diagram of the classic asynchronous MySQL Replication protocol (and its semisynchronous variant as well). Diagonal arrows represent messages exchanged between servers or messages exchanged between servers and the client application.

18.1.1.2 Group Replication

Group Replication is a technique that can be used to implement fault-tolerant systems. The replication group is a set of servers that each have their own entire copy of the data (a shared-nothing replication scheme), and interact with each other through message passing. The communication layer provides a set of guarantees such as atomic message and total order message delivery. These are very powerful properties that translate into very useful abstractions that one can resort to build more advanced database replication solutions.

MySQL Group Replication builds on top of such properties and abstractions and implements a multi-master update everywhere replication protocol. A replication group is formed by multiple servers and each server in the group may execute transactions independently at any time. However, all read-write transactions commit only after they have been approved by the group. In other words, for any read-write transaction the group needs to decide whether it commits or not, so the commit operation is not a unilateral decision from the originating server. Read-only transactions need no coordination within the group and commit immediately.

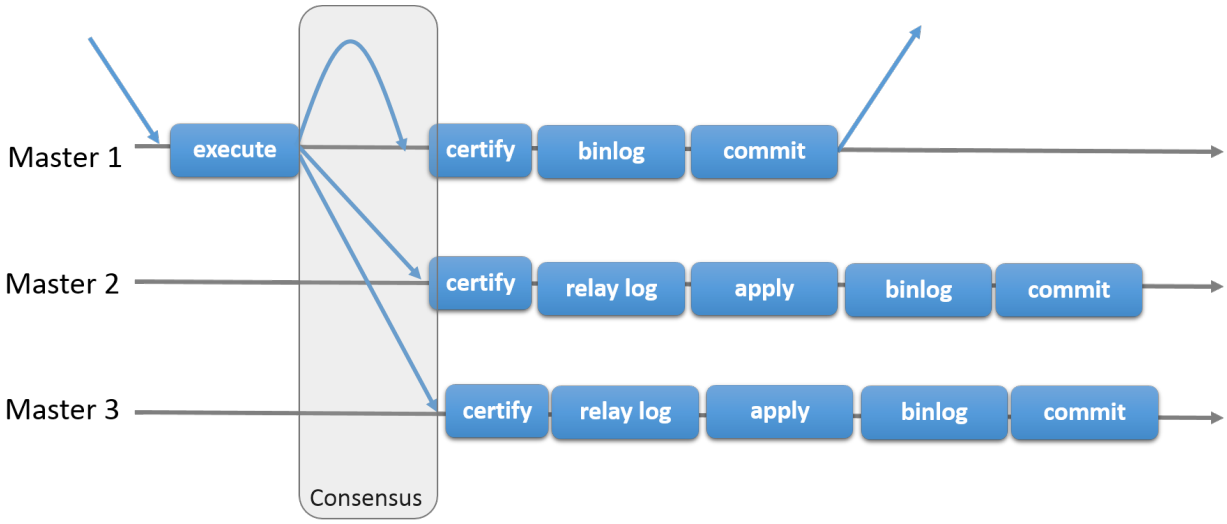
When a read-write transaction is ready to commit at the originating server, the server atomically broadcasts the write values (the rows that were changed) and the corresponding write set (the unique identifiers of the rows that were updated). Because the transaction is sent through an atomic broadcast, either all servers in the group receive the transaction or none do. If they receive it, then they all receive it in the same order with respect to other transactions that were sent before. All servers therefore receive the same set of transactions in the same order, and a global total order is established for the transactions.

However, there may be conflicts between transactions that execute concurrently on different servers. Such conflicts are detected by inspecting and comparing the write sets of two different and concurrent transactions, in a process called *certification*. During certification, conflict detection is carried out at row level: if two concurrent transactions, that executed on different servers, update the same row, then there is a conflict. The conflict resolution procedure states that the transaction that was ordered first commits on all servers, and the transaction ordered second aborts, and is therefore rolled back on the originating server and dropped by the other servers in the group. For example, if t1 and t2 execute concurrently at different sites, both changing the same row, and t2 is ordered before t1, then t2 wins the conflict and t1 is rolled back. This is in fact a distributed first commit wins rule. Note that if two transactions are bound to conflict more often than not, then it is a good practice to start them on the same server, where they have a chance to synchronize on the local lock manager instead of being rolled back as a result of certification.

For applying and externalizing the certified transactions, Group Replication permits servers to deviate from the agreed order of the transactions if this does not break consistency and validity. Group Replication is an eventual consistency system, meaning that as soon as the incoming traffic slows down or stops, all group members have the same data content. While traffic is flowing, transactions can be externalized in a slightly different order, or externalized on some members before the others. For example, in multi-primary mode, a local transaction might be externalized immediately following certification, although a remote transaction that is earlier in the global order has not yet been applied. This is permitted when the certification process has established that there is no conflict between the transactions. In single-primary mode, on the primary server, there is a small chance that concurrent, non-conflicting local transactions might be committed and externalized in a different order from the global order agreed by Group Replication. On the secondaries, which do not accept writes from clients, transactions are always committed and externalized in the agreed order.

The following figure depicts the MySQL Group Replication protocol and by comparing it to MySQL Replication (or even MySQL semisynchronous replication) you can see some differences. Some underlying consensus and Paxos related messages are missing from this picture for the sake of clarity.

Figure 18.3 MySQL Group Replication Protocol



18.1.2 Group Replication Use Cases

Group Replication enables you to create fault-tolerant systems with redundancy by replicating the system state to a set of servers. Even if some of the servers subsequently fail, as long it is not all or a majority, the system is still available. Depending on the number of servers which fail the group might have degraded performance or scalability, but it is still available. Server failures are isolated and independent. They are tracked by a group membership service which relies on a distributed failure detector that is able to signal when any servers leave the group, either voluntarily or due to an unexpected halt. There is a distributed recovery procedure to ensure that when servers join the group they are brought up to date automatically. There is no need for server failover, and the multi-master update everywhere nature ensures that even updates are not blocked in the event of a single server failure. To summarize, MySQL Group Replication guarantees that the database service is continuously available.

It is important to understand that although the database service is available, in the event of a server crash, those clients connected to it must be redirected, or failed over, to a different server. This is not something Group Replication attempts to resolve. A connector, load balancer, router, or some form of middleware are more suitable to deal with this issue. For example see MySQL Router 8.0.

To summarize, MySQL Group Replication provides a highly available, highly elastic, dependable MySQL service.

18.1.2.1 Examples of Use Case Scenarios

The following examples are typical use cases for Group Replication.

- *Elastic Replication* - Environments that require a very fluid replication infrastructure, where the number of servers has to grow or shrink dynamically and with as few side-effects as possible. For instance, database services for the cloud.
- *Highly Available Shards* - Sharding is a popular approach to achieve write scale-out. Use MySQL Group Replication to implement highly available shards, where each shard maps to a replication group.
- *Alternative to Master-Slave replication* - In certain situations, using a single master server makes it a single point of contention. Writing to an entire group may prove more scalable under certain circumstances.
- *Autonomic Systems* - Additionally, you can deploy MySQL Group Replication purely for the automation that is built into the replication protocol (described already in this and previous chapters).

18.1.3 Multi-Primary and Single-Primary Modes

Group Replication operates either in single-primary mode or in multi-primary mode. The group's mode is a group-wide configuration setting, specified by the `group_replication_single_primary_mode` system variable, which must be the same on all members. `ON` means single-primary mode, which is the default mode, and `OFF` means multi-primary mode. It is not possible to have members of the group deployed in different modes, for example one member configured in multi-primary mode while another member is in single-primary mode.

You cannot change the value of `group_replication_single_primary_mode` manually while Group Replication is running. From MySQL 8.0.13, you can use the `group_replication_switch_to_single_primary_mode()` and `group_replication_switch_to_multi_primary_mode()` UDFs to move a group from one mode to another while Group Replication is still running. These UDFs manage the process of changing the group's mode and ensure the safety and consistency of your data. In earlier releases, to change the group's mode you must stop Group Replication and change the value of `group_replication_single_primary_mode` on all members. Then carry out a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) to implement the change to the new operating configuration. You do not need to restart the servers.

Regardless of the deployed mode, Group Replication does not handle client-side failover. That must be handled by a middleware framework such as MySQL Router 8.0, a proxy, a connector, or the application itself.

18.1.3.1 Single-Primary Mode

In single-primary mode (`group_replication_single_primary_mode=ON`) the group has a single primary server that is set to read-write mode. All the other members in the group are set to read-only mode (with `super-read-only=ON`). The primary is typically the first server to bootstrap the group. All other servers that join the group learn about the primary server and are automatically set to read-only mode.

In single-primary mode, Group Replication enforces that only a single server writes to the group, so compared to multi-primary mode, consistency checking can be less strict and DDL statements do not need to be handled with any extra care. The option `group_replication_enforce_update_everywhere_checks` enables or disables strict consistency

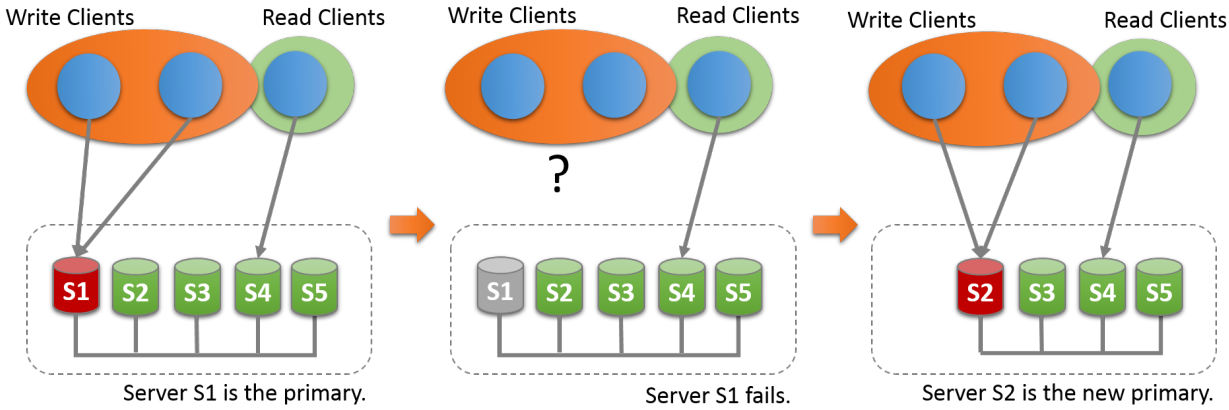
checks for a group. When deploying in single-primary mode, or changing the group to single-primary mode, this system variable must be set to `OFF`.

The member that is designated as the primary server can change in the following ways:

- If the existing primary leaves the group, whether voluntarily or unexpectedly, a new primary is elected automatically.
- You can appoint a specific member as the new primary using the `group_replication_set_as_primary()` UDF.
- If you use the `group_replication_switch_to_single_primary_mode()` UDF to change a group that was running in multi-primary mode to run in single-primary mode, a new primary is elected automatically, or you can appoint the new primary by specifying it with the UDF.

The UDFs can only be used when all group members are running MySQL 8.0.13 or higher. When a new primary server is elected automatically or appointed manually, it is automatically set to read-write, and the other group members remain as secondaries, and as such, read-only. Figure 18.4, “New Primary Election” shows this process.

Figure 18.4 New Primary Election



When a new primary is elected or appointed, it might have a backlog of changes that had been applied on the old primary but have not yet been applied on this server. In this situation, until the new primary catches up with the old primary, read-write transactions might result in conflicts and be rolled back, and read-only transactions might result in stale reads. Group Replication's flow control mechanism, which minimizes the difference between fast and slow members, reduces the chances of this happening if it is activated and properly tuned. For more information on flow control, see Section 18.6.2, “Flow Control”. From MySQL 8.0.14, you can also use the `group_replication_consistency` system variable to configure the group's level of transaction consistency to prevent this issue. The setting `BEFORE_ON_PRIMARY_FAILOVER` (or any higher consistency level) holds new transactions on a newly elected primary until the backlog has been applied. For more information on transaction consistency, see Section 18.4.2, “Transaction Consistency Guarantees”. If flow control and transaction consistency guarantees are not used for a group, it is a good practice to wait for the new primary to apply its replication-related relay log before re-routing client applications to it.

Primary Election Algorithm

The automatic primary member election process involves each member looking at the new view of the group, ordering the potential new primary members, and choosing the member that qualifies as the most suitable. Each member makes its own decision locally, following the primary election algorithm in its

MySQL Server release. Because all members must reach the same decision, members adapt their primary election algorithm if other group members are running lower MySQL Server versions, so that they have the same behavior as the member with the lowest MySQL Server version in the group.

The factors considered by members when electing a primary, in order, are as follows:

1. The first factor considered is which member or members are running the lowest MySQL Server version. If all group members are running MySQL 8.0.17 or higher, members are first ordered by the patch version of their release. If any members are running MySQL Server 5.7 or MySQL 8.0.16 or lower, members are first ordered by the major version of their release, and the patch version is ignored.

2. If more than one member is running the lowest MySQL Server version, the second factor considered is the member weight of each of those members, as specified by the `group_replication_member_weight` system variable on the member. If any member of the group is running MySQL Server 5.7, where this system variable was not available, this factor is ignored.

   The `group_replication_member_weight` system variable specifies a number in the range 0-100. All members default to a weight of 50, so set a weight below this to lower their ordering, and a weight above it to increase their ordering. You can use this weighting function to prioritize the use of better hardware or to ensure failover to a specific member during scheduled maintenance of the primary.

3. If more than one member is running the lowest MySQL Server version, and more than one of those members has the highest member weight (or member weighting is being ignored), the third factor considered is the lexographical order of the generated server UUIDs of each member, as specified by the `server_uuid` system variable. The member with the lowest server UUID is chosen as the primary. This factor acts as a guaranteed and predictable tie-breaker so that all group members reach the same decision if it cannot be determined by any important factors.

Finding the Primary

To find out which server is currently the primary when deployed in single-primary mode, use the `MEMBER_ROLE` column in the `performance_schema.replication_group_members` table. For example:

```
mysql> SELECT MEMBER_HOST, MEMBER_ROLE FROM performance_schema.replication_group_members;
+-------------------------+-------------+
| MEMBER_HOST             | MEMBER_ROLE |
+-------------------------+-------------+
| remote1.example.com     | PRIMARY     |
| remote2.example.com     | SECONDARY   |
| remote3.example.com     | SECONDARY   |
+-------------------------+-------------+
```

Warning

The `group_replication_primary_member` status variable has been deprecated and is scheduled to be removed in a future version.

Alternatively use the `group_replication_primary_member` status variable.

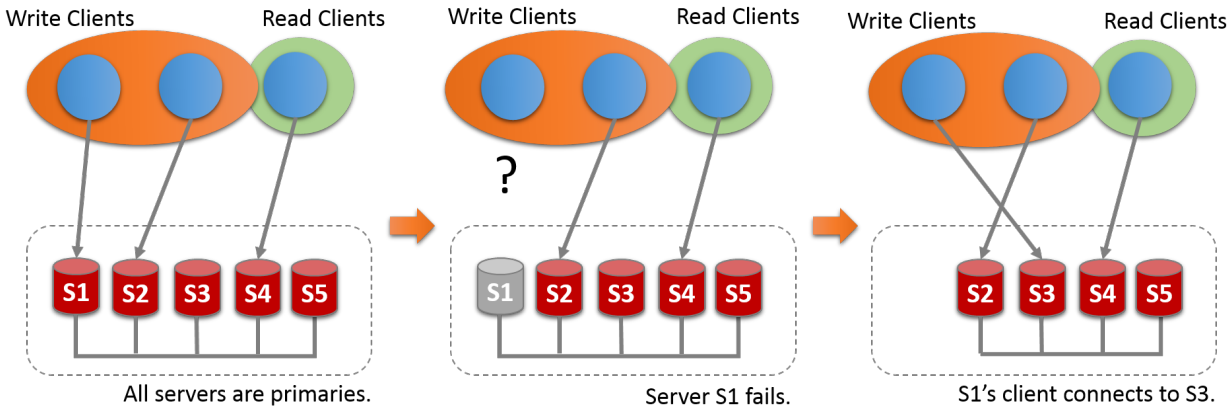
```
mysql> SHOW STATUS LIKE 'group_replication_primary_member'
```

18.1.3.2 Multi-Primary Mode

In multi-primary mode (`group_replication_single_primary_mode=OFF`) no member has a special role. Any member that is compatible with the other group members is set to read-write mode when joining the group, and can process write transactions, even if they are issued concurrently.

If a member stops accepting write transactions, for example in the event of a server crash, clients connected to it can be redirected, or failed over, to any other member that is in read-write mode. Group Replication does not handle client-side failover itself, so you need to arrange this using a middleware framework such as MySQL Router 8.0, a proxy, a connector, or the application itself. Figure 18.5, “Client Failover” shows how clients can reconnect to an alternative group member if a member leaves the group.

Figure 18.5 Client Failover



Group Replication is an eventual consistency system. This means that as soon as the incoming traffic slows down or stops, all group members have the same data content. While traffic is flowing, transactions can be externalized on some members before the others, especially if some members have less write throughput than others, creating the possibility of stale reads. In multi-primary mode, slower members can also build up an excessive backlog of transactions to certify and apply, leading to a greater risk of conflicts and certification failure. To limit these issues, you can activate and tune Group Replication's flow control mechanism to minimize the difference between fast and slow members. For more information on flow control, see Section 18.6.2, “Flow Control”.

From MySQL 8.0.14, if you want to have a transaction consistency guarantee for every transaction in the group, you can do this using the `group_replication_consistency` system variable. You can choose a setting that suits the workload of your group and your priorities for data reads and writes, taking into account the performance impact of the synchronization required to increase consistency. You can also set the system variable for individual sessions to protect particularly concurrency-sensitive transactions. For more information on transaction consistency, see Section 18.4.2, “Transaction Consistency Guarantees”.

Transaction Checks

When a group is deployed in multi-primary mode, transactions are checked to ensure they are compatible with the mode. The following strict consistency checks are made when Group Replication is deployed in multi-primary mode:

- If a transaction is executed under the SERIALIZABLE isolation level, then its commit fails when synchronizing itself with the group.
- If a transaction executes against a table that has foreign keys with cascading constraints, then its commit fails when synchronizing itself with the group.

The checks are controlled by the `group_replication_enforce_update_everywhere_checks` system variable. In multi-primary mode, the system variable should normally be set to `ON`, but the checks can optionally be deactivated by setting the system variable to `OFF`. When deploying in single-primary mode, the system variable must be set to `OFF`.

Data Definition Statements

In a Group Replication topology in multi-primary mode, care needs to be taken when executing data definition statements, also commonly known as data definition language (DDL).

MySQL 8.0 introduces support for atomic Data Definition Language (DDL) statements, where the complete DDL statement is either committed or rolled back as a single atomic transaction. However, DDL statements, atomic or otherwise, implicitly end any transaction that is active in the current session, as if you had done a `COMMIT` before executing the statement. This means that DDL statements cannot be performed within another transaction, within transaction control statements such as `START TRANSACTION ... COMMIT`, or combined with other statements within the same transaction.

Group Replication is based on an optimistic replication paradigm, where statements are optimistically executed and rolled back later if necessary. Each server executes without securing group agreement first. Therefore, more care needs to be taken when replicating DDL statements in multi-primary mode. If you make schema changes (using DDL) and changes to the data that an object contains (using DML) for the same object, the changes need to be handled through the same server while the schema operation has not yet completed and replicated everywhere. Failure to do so can result in data inconsistency when operations are interrupted or only partially completed. If the group is deployed in single-primary mode this issue does not occur, because all changes are performed through the same server, the primary.

For details on atomic DDL support in MySQL 8.0, and the resulting changes in behavior for the replication of certain statements, see Section 13.1.1, “Atomic Data Definition Statement Support”.

Version Compatibility

For optimal compatibility and performance, all members of a group should run the same version of MySQL Server and therefore of Group Replication. In multi-primary mode, this is more significant because all members would normally join the group in read-write mode. If a group includes members running more than one MySQL Server version, there is a potential for some members to be incompatible with others, because they support functions others do not, or lack functions others have. To guard against this, when a new member joins (including a former member that has been upgraded and restarted), the member carries out compatibility checks against the rest of the group.

One result of these compatibility checks is particularly important in multi-primary mode. If a joining member is running a higher MySQL Server version than the lowest version that the existing group members are running, it joins the group but remains in read-only mode. (In a group that is running in single-primary mode, newly added members default to being read-only in any case.) Members running MySQL 8.0.17 or higher take into account the patch version of the release when checking their compatibility. Members running MySQL 8.0.16 or lower, or MySQL 5.7, only take into account the major version.

In a group running in multi-primary mode with members that use different MySQL Server versions, Group Replication automatically manages the read-write and read-only status of members running MySQL 8.0.17 or higher. If a member leaves the group, the members running the version that is now the lowest are automatically set to read-write mode. When you change a group that was running in single-primary mode to run in multi-primary mode, using the `group_replication_switch_to_multi_primary_mode()` UDF, Group Replication automatically sets members to the correct mode. Members are automatically placed in read-only mode if they are running a higher MySQL server version than the lowest version present in the group, and members running the lowest version are placed in read-write mode.

For full information on version compatibility in a group and how this influences the behavior of a group during an upgrade process, see Section 18.7.1, “Combining Different Member Versions in a Group” .

18.1.4 Group Replication Services

This section introduces some of the services that Group Replication builds on.

18.1.4.1 Group Membership

In MySQL Group Replication, a set of servers forms a replication group. A group has a name, which takes the form of a UUID. The group is dynamic and servers can leave (either voluntarily or involuntarily) and join it at any time. The group adjusts itself whenever servers join or leave.

If a server joins the group, it automatically brings itself up to date by fetching the missing state from an existing server. If a server leaves the group, for instance it was taken down for maintenance, the remaining servers notice that it has left and reconfigure the group automatically.

Group Replication has a group membership service that defines which servers are online and participating in the group. The list of online servers is referred to as a *view*. Every server in the group has a consistent view of which servers are the members participating actively in the group at a given moment in time.

Group members must agree not only on transaction commits, but also on which is the current view. If existing members agree that a new server should become part of the group, the group is reconfigured to integrate that server in it, which triggers a view change. If a server leaves the group, either voluntarily or not, the group dynamically rearranges its configuration and a view change is triggered.

In the case where a member leaves the group voluntarily, it first initiates a dynamic group reconfiguration, during which all members have to agree on a new view without the leaving server. However, if a member leaves the group involuntarily, for example because it has stopped unexpectedly or the network connection is down, it cannot initiate the reconfiguration. In this situation, Group Replication's failure detection mechanism recognizes after a short period of time that the member has left, and a reconfiguration of the group without the failed member is proposed. As with a member that leaves voluntarily, the reconfiguration requires agreement from the majority of servers in the group. However, if the group is not able to reach agreement, for example because it partitioned in such a way that there is no majority of servers online, the system is not able to dynamically change the configuration, and blocks to prevent a split-brain situation. This situation requires intervention from an administrator.

It is possible for a member to go offline for a short time, then attempt to rejoin the group again before the failure detection mechanism has detected its failure, and before the group has been reconfigured to remove the member. In this situation, the rejoining member forgets its previous state, but if other members send it messages that are intended for its pre-crash state, this can cause issues including possible data inconsistency. If a member in this situation participates in XCom's consensus protocol, it could potentially cause XCom to deliver different values for the same consensus round, by making a different decision before and after failure.

To counter this possibility, from MySQL 5.7.22 and in MySQL 8.0, servers are given a unique identifier when they join a group. This enables Group Replication to be aware of the situation where a new incarnation of the same server (with the same address but a new identifier) is trying to join the group while its old incarnation is still listed as a member. The new incarnation is blocked from joining the group until the old incarnation can be removed by a reconfiguration. If the `group_replication_member_expel_timeout` system variable has been set to allow additional time for members to return to the group before they are expelled, a member under suspicion can rejoin as long as it has not actually failed. If Group Replication is stopped and restarted on the server, the member becomes a new incarnation and cannot rejoin until the suspicion times out.

18.1.4.2 Failure Detection

Group Replication includes a failure detection mechanism that is able to find and report which servers are silent and as such assumed to be dead. At a high level, the failure detector is a distributed service

that provides information about which servers may be dead (suspicions). Suspicions are triggered when servers go mute. When server A does not receive messages from server B during a given period, a timeout occurs and a suspicion is raised. Later if the group agrees that the suspicions are probably true, then the group decides that a given server has indeed failed. This means that the remaining members in the group take a coordinated decision to exclude a given member.

If a server gets isolated from the rest of the group, then it suspects that all others have failed. Being unable to secure agreement with the group (as it cannot secure a quorum), its suspicion does not have consequences. When a server is isolated from the group in this way, it is unable to execute any local transactions.

For information on the Group Replication system variables that you can configure to specify the responses of working group members, and group members that are suspected of having failed, to these situations, see Section 18.6.6, “Responses to Failure Detection and Network Partitioning”.

18.1.4.3 Fault-tolerance

MySQL Group Replication builds on an implementation of the Paxos distributed algorithm to provide distributed coordination between servers. As such, it requires a majority of servers to be active to reach quorum and thus make a decision. This has direct impact on the number of failures the system can tolerate without compromising itself and its overall functionality. The number of servers (n) needed to tolerate `f` failures is then `n = 2 x f + 1`.

In practice this means that to tolerate one failure the group must have three servers in it. As such if one server fails, there are still two servers to form a majority (two out of three) and allow the system to continue to make decisions automatically and progress. However, if a second server fails *involuntarily*, then the group (with one server left) blocks, because there is no majority to reach a decision.

The following is a small table illustrating the formula above.

Group Size	Majority	Failures Tolerated
1	1	0
2	2	0
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

18.1.4.4 Observability

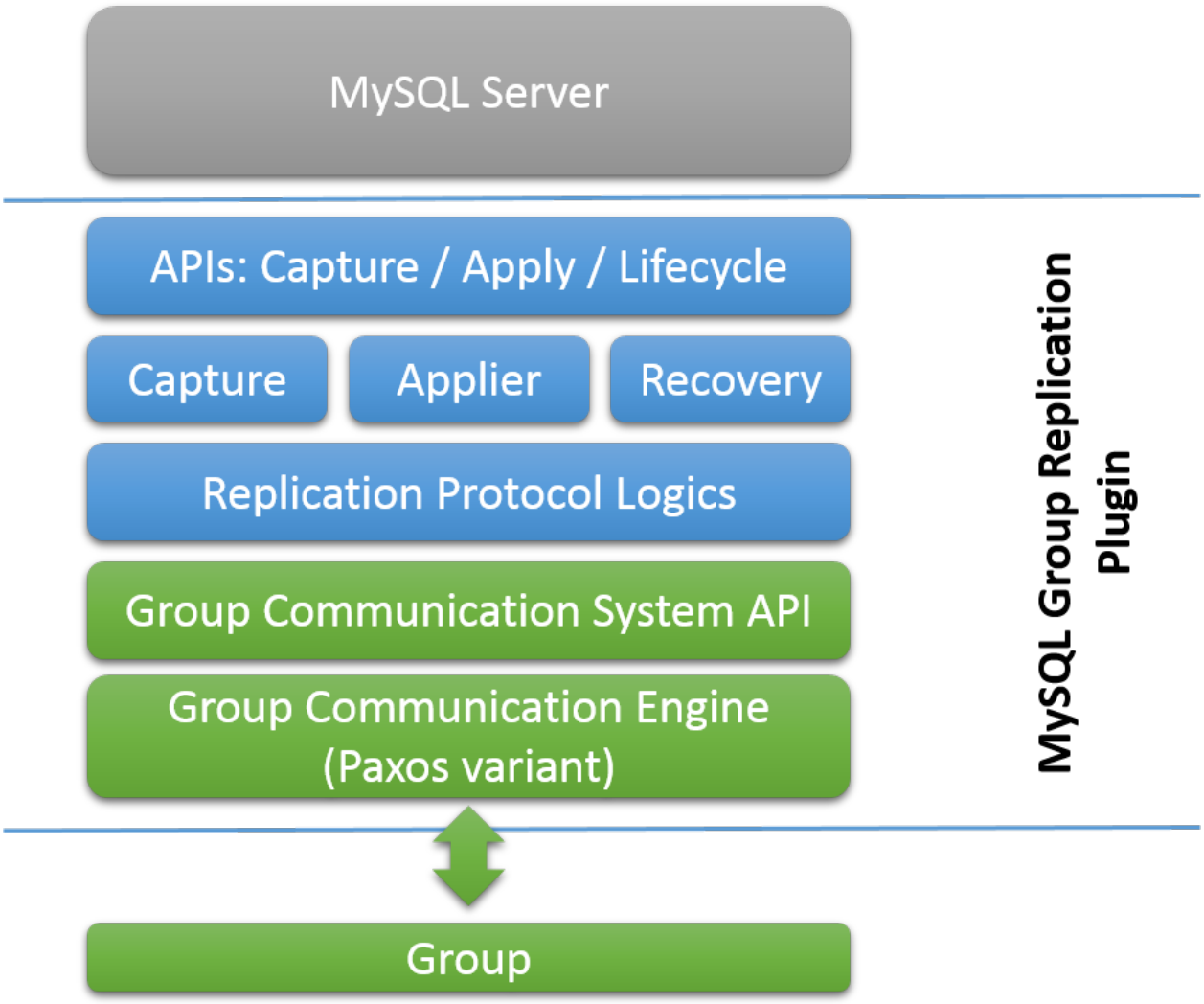
There is a lot of automation built into the Group Replication plugin. Nonetheless, you might sometimes need to understand what is happening behind the scenes. This is where the instrumentation of Group Replication and Performance Schema becomes important. The entire state of the system (including the view, conflict statistics and service states) can be queried through Performance Schema tables. The distributed nature of the replication protocol and the fact that server instances agree and thus synchronize on transactions and metadata makes it simpler to inspect the state of the group. For example, you can connect to a single server in the group and obtain both local and global information by issuing select

statements on the Group Replication related Performance Schema tables. For more information, see Section 18.3, “Monitoring Group Replication”.

18.1.5 Group Replication Plugin Architecture

MySQL Group Replication is a MySQL plugin and it builds on the existing MySQL replication infrastructure, taking advantage of features such as the binary log, row-based logging, and global transaction identifiers. It integrates with current MySQL frameworks, such as the performance schema or plugin and service infrastructures. The following figure presents a block diagram depicting the overall architecture of MySQL Group Replication.

Figure 18.6 Group Replication Plugin Block Diagram



The MySQL Group Replication plugin includes a set of APIs for capture, apply, and lifecycle, which control how the plugin interacts with MySQL Server. There are interfaces to make information flow from the server to the plugin and vice versa. These interfaces isolate the MySQL Server core from the Group Replication plugin, and are mostly hooks placed in the transaction execution pipeline. In one direction, from server to the plugin, there are notifications for events such as the server starting, the server recovering, the server being ready to accept connections, and the server being about to commit a transaction. In the

other direction, the plugin instructs the server to perform actions such as committing or aborting ongoing transactions, or queuing transactions in the relay log.

The next layer of the Group Replication plugin architecture is a set of components that react when a notification is routed to them. The capture component is responsible for keeping track of context related to transactions that are executing. The applier component is responsible for executing remote transactions on the database. The recovery component manages distributed recovery, and is responsible for getting a server that is joining the group up to date by selecting the donor, managing the catch up procedure and reacting to donor failures.

Continuing down the stack, the replication protocol module contains the specific logic of the replication protocol. It handles conflict detection, and receives and propagates transactions to the group.

The final two layers of the Group Replication plugin architecture are the Group Communication System (GCS) API, and an implementation of a Paxos-based group communication engine (XCom). The GCS API is a high level API that abstracts the properties required to build a replicated state machine (see Section 18.1, “Group Replication Background”). It therefore decouples the implementation of the messaging layer from the remaining upper layers of the plugin. The group communication engine handles communications with the members of the replication group.

18.2 Getting Started

MySQL Group Replication is provided as a plugin to MySQL server, and each server in a group requires configuration and installation of the plugin. This section provides a detailed tutorial with the steps required to create a replication group with at least three members.

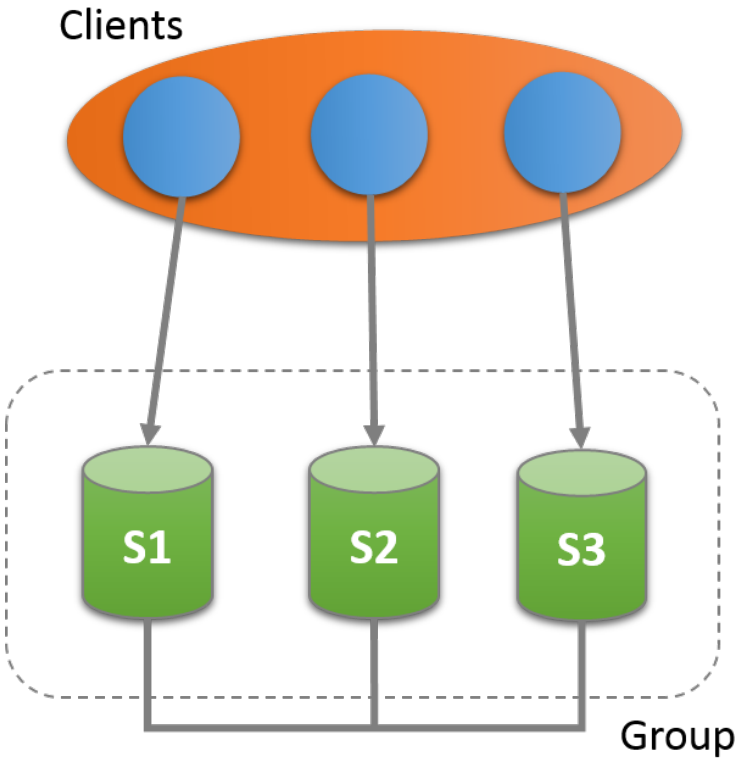
Tip

An alternative way to deploy multiple instances of MySQL is by using InnoDB cluster, which uses Group Replication and wraps it in a programmatic environment that enables you to easily work with groups of MySQL server instances in the MySQL Shell 8.0 (part of MySQL 8.0). In addition, InnoDB cluster interfaces seamlessly with MySQL Router and simplifies deploying MySQL with high availability. See Chapter 21, *InnoDB Cluster*.

18.2.1 Deploying Group Replication in Single-Primary Mode

Each of the MySQL server instances in a group can run on an independent physical host machine, which is the recommended way to deploy Group Replication. This section explains how to create a replication group with three MySQL Server instances, each running on a different host machine. See Section 18.2.2, “Deploying Group Replication Locally” for information about deploying multiple MySQL server instances running Group Replication on the same host machine, for example for testing purposes.

Figure 18.7 Group Architecture



This tutorial explains how to get and deploy MySQL Server with the Group Replication plugin, how to configure each server instance before creating a group, and how to use Performance Schema monitoring to verify that everything is working correctly.

18.2.1.1 Deploying Instances for Group Replication

The first step is to deploy at least three instances of MySQL Server, this procedure demonstrates using multiple hosts for the instances, named s1, s2 and s3. It is assumed that MySQL Server was installed on each of the hosts, see Chapter 2, *Installing and Upgrading MySQL*. Group Replication is a built-in MySQL plugin provided with MySQL Server 8.0, therefore no additional installation is required. For more background information on MySQL plugins, see Section 5.6, “MySQL Server Plugins”.

In this example, three instances are used for the group, which is the minimum number of instances to create a group. Adding more instances increases the fault tolerance of the group. For example if the group consists of three members, in event of failure of one instance the group can continue. But in the event of another failure the group can no longer continue processing write transactions. By adding more instances, the number of servers which can fail while the group continues to process transactions also increases. The maximum number of instances which can be used in a group is nine. For more information see Section 18.1.4.2, “Failure Detection”.

18.2.1.2 Configuring an Instance for Group Replication

This section explains the configuration settings required for MySQL Server instances that you want to use for Group Replication. For background information, see Section 18.9, “Requirements and Limitations”.

- Storage Engines

- Replication Framework
- Group Replication Settings

Storage Engines

For Group Replication, data must be stored in the InnoDB transactional storage engine (for details of why, see Section 18.9.1, “Group Replication Requirements”). The use of other storage engines, including the temporary `MEMORY` storage engine, might cause errors in Group Replication. Set the `disabled_storage_engines` system variable as follows to prevent their use:

```
disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"
```

Note that with the `MyISAM` storage engine disabled, when you are upgrading a MySQL instance to a release where `mysql_upgrade` is still used (before MySQL 8.0.16), `mysql_upgrade` might fail with an error. To handle this, you can re-enable that storage engine while you run `mysql_upgrade`, then disable it again when you restart the server. For more information, see Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

Replication Framework

The following settings configure replication according to the MySQL Group Replication requirements.

```
server_id=1
gtid_mode=ON
enforce_gtid_consistency=ON
binlog_checksum=NONE
```

These settings configure the server to use the unique identifier number 1, to enable Section 17.1.3, “Replication with Global Transaction Identifiers”, to allow execution of only statements that can be safely logged using a GTID, and to disable writing checksums for events written to the binary log.

If you are using a version of MySQL earlier than 8.0.3, where the defaults were improved for replication, you need to add these lines to the member's option file.

```
log_bin=binlog
log_slave_updates=ON
binlog_format=ROW
master_info_repository=TABLE
relay_log_info_repository=TABLE
```

These settings instruct the server to turn on binary logging, use row-based format, to store replication metadata in system tables instead of files, and disable binary log event checksums. For more details see Section 18.9.1, “Group Replication Requirements”.

Group Replication Settings

At this point the option file ensures that the server is configured and is instructed to instantiate the replication infrastructure under a given configuration. The following section configures the Group Replication settings for the server.

```
plugin_load_add='group_replication.so
group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
group_replication_start_on_boot=off
```

```
group_replication_local_address= "s1:33061"
group_replication_group_seeds= "s1:33061,s2:33061,s3:33061"
group_replication_bootstrap_group=off
```

- `plugin-load-add` adds the Group Replication plugin to the list of plugins which the server loads at startup. This is preferable in a production deployment to installing the plugin manually.
- Configuring `group_replication_group_name` tells the plugin that the group that it is joining, or creating, is named "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa".

  The value of `group_replication_group_name` must be a valid UUID. This UUID is used internally when setting GTIDs for Group Replication events in the binary log. You can use `SELECT UUID()` to generate a UUID.
- Configuring the `group_replication_start_on_boot` variable to `off` instructs the plugin to not start operations automatically when the server starts. This is important when setting up Group Replication as it ensures you can configure the server before manually starting the plugin. Once the member is configured you can set `group_replication_start_on_boot` to `on` so that Group Replication starts automatically upon server boot.
- Configuring `group_replication_local_address` sets the network address and port which the member uses for internal communication with other members in the group. Group Replication uses this address for internal member-to-member connections involving remote instances of the group communication engine (XCom, a Paxos variant).

  Important

  The group replication local address must be different to the `hostname` and `port` used for SQL and it must not be used for client applications. It must be only be used for internal communication between the members of the group while running Group Replication.

  The network address configured by `group_replication_local_address` must be resolvable by all group members. For example, if each server instance is on a different machine with a fixed network address, you could use the IP address of the machine, such as 10.0.0.1. If you use a host name, you must use a fully qualified name, and ensure it is resolvable through DNS, correctly configured `/etc/hosts` files, or other name resolution processes. From MySQL 8.0.14, IPv6 addresses (or host names that resolve to them) can be used as well as IPv4 addresses. A group can contain a mix of members using IPv6 and members using IPv4. For more information on Group Replication support for IPv6 networks and on mixed IPv4 and IPv6 groups, see Section 18.4.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”.

  The recommended port for `group_replication_local_address` is 33061. `group_replication_local_address` is used by Group Replication as the unique identifier for a group member within the replication group. You can use the same port for all members of a replication group as long as the host names or IP addresses are all different, as demonstrated in this tutorial. Alternatively you can use the same host name or IP address for all members as long as the ports are all different, for example as shown in Section 18.2.2, “Deploying Group Replication Locally”.

  Important

  Although the group replication local address is different to the `hostname` and `port` used for SQL, Group Replication's distributed recovery process for joining members can fail if the server cannot correctly identify the other members using the server's SQL `hostname`. It is recommended that operating systems running MySQL have a properly configured unique `hostname`, either using DNS or local settings. The `hostname` can be verified in the `Member_host` column of the

> `performance_schema.replication_group_members` table. If multiple group members externalize a default `hostname` set by the operating system, there is a chance of the member not resolving to the correct member address and not being able to join the group. In such a situation use `report_host` to configure a unique `hostname` to be externalized by each of the servers.

- Configuring `group_replication_group_seeds` sets the hostname and port of the group members which are used by the new member to establish its connection to the group. These members are called the seed members. Once the connection is established, the group membership information is listed at `performance_schema.replication_group_members`. Usually the `group_replication_group_seeds` list contains the `hostname:port` of each of the group member's `group_replication_local_address`, but this is not obligatory and a subset of the group members can be chosen as seeds.

  > **Important**
  >
  > The `hostname:port` listed in `group_replication_group_seeds` is the seed member's internal network address, configured by `group_replication_local_address` and not the SQL `hostname:port` used for client connections, and shown for example in `performance_schema.replication_group_members` table.

  The server that starts the group does not make use of this option, since it is the initial server and as such, it is in charge of bootstrapping the group. In other words, any existing data which is on the server bootstrapping the group is what is used as the data for the next joining member. The second server joining asks the one and only member in the group to join, any missing data on the second server is replicated from the donor data on the bootstrapping member, and then the group expands. The third server joining can ask any of these two to join, data is synchronized to the new member, and then the group expands again. Subsequent servers repeat this procedure when joining.

  > **Warning**
  >
  > When joining multiple servers at the same time, make sure that they point to seed members that are already in the group. Do not use members that are also joining the group as seeds, because they might not yet be in the group when contacted.
  >
  > It is good practice to start the bootstrap member first, and let it create the group. Then make it the seed member for the rest of the members that are joining. This ensures that there is a group formed when joining the rest of the members.
  >
  > Creating a group and joining multiple members at the same time is not supported. It might work, but chances are that the operations race and then the act of joining the group ends up in an error or a time out.

  A joining member must communicate with a seed member using the same protocol (IPv4 or IPv6) that the seed member advertises in the `group_replication_group_seeds` option. For the purpose of IP address whitelisting for Group Replication, the whitelist on the seed member must include an IP address for the joining member for the protocol offered by the seed member, or a host name that resolves to an address for that protocol. This address or host name must be set up and whitelisted in addition to the joining member's `group_replication_local_address` if the protocol for that address does not match the seed member's advertised protocol. If a joining member does not have a whitelisted address for the appropriate protocol, its connection attempt is refused. For more information, see Section 18.5.1, “Group Replication IP Address Whitelisting”.

- Configuring `group_replication_bootstrap_group` instructs the plugin whether to bootstrap the group or not. In this case, even though s1 is the first member of the group we set this variable to off in

the option file. Instead we configure `group_replication_bootstrap_group` when the instance is running, to ensure that only one member actually bootstraps the group.

Important

The `group_replication_bootstrap_group` variable must only be enabled on one server instance belonging to a group at any time, usually the first time you bootstrap the group (or in case the entire group is brought down and back up again). If you bootstrap the group multiple times, for example when multiple server instances have this option set, then they could create an artificial split brain scenario, in which two distinct groups with the same name exist. Always set `group_replication_bootstrap_group=off` after the first server instance comes online.

If you are using an instance running a version of MySQL earlier than 8.0.2, you need to also configure the `transaction_write_set_extraction` variable to XXHASH64. This variable instructs the server that for each transaction it has to collect the write set and encode it as a hash using the XXHASH64 hashing algorithm. From MySQL 8.0.2, this setting is the default, otherwise add the following to the option file:

```
transaction_write_set_extraction=XXHASH64
```

Configuration for all servers in the group is quite similar. You need to change the specifics about each server (for example `server_id`, `datadir`, `group_replication_local_address`). This is illustrated later in this tutorial.

18.2.1.3 User Credentials

Group Replication uses a distributed recovery process to synchronize group members when joining them to the group. Distributed recovery involves transferring transactions from a donor's binary log to a joining member using a replication channel named `group_replication_recovery`. You must therefore set up a replication user with the correct permissions so that Group Replication can establish direct member-to-member replication channels. If group members have been set up to support the use of a remote cloning operation as part of distributed recovery, which is available from MySQL 8.0.17, this replication user is also used as the clone user on the donor , and requires the correct permissions for this role too. For a complete description of distributed recovery, see Section 18.4.3, “Distributed Recovery”.

The process of setting up the replication user for distributed recovery can be captured in the binary log, and then you can rely on distributed recovery to replicate the statements used to create the user. Alternatively, you can disable binary logging before creating the replication user, and then create the user manually on each member, for example if you want to avoid the changes being propagated to other server instances. If you do this, ensure you re-enable binary logging once you have configured the user.

If you have set up cloning for your replication group, the replication user and password used by the donor for the `group_replication_recovery` replication channel are transferred to the joining member after cloning, and used by the joining member afterwards, so they must be valid there. All group members that received state transfer by a remote cloning operation therefore use the same replication user and password for distributed recovery.

Important

If distributed recovery connections for your group use SSL, the replication user must be created on each server *before* the joining member connects to the donor. For instructions to set up SSL for distributed recovery connections, see Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”

To create the replication user for distributed recovery, follow these steps:

1. Start the MySQL server instance, then connect a client to it.

2. If you want to disable binary logging in order to create the replication user separately on each instance, do so by issuing the following statement:

```
mysql> SET SQL_LOG_BIN=0;
```

3. Create a MySQL user with the `REPLICATION SLAVE` privilege to use for distributed recovery, and if the server is set up to support cloning, the `BACKUP_ADMIN` privilege to use as the donor in a cloning operation. In this example the user *`rpl_user`* with the password *`password`* is shown. When configuring your servers use a suitable user name and password:

```
mysql> CREATE USER rpl_user@'%' IDENTIFIED BY 'password';
mysql> GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%';
mysql> GRANT BACKUP_ADMIN ON *.* TO rpl_user@'%';
mysql> FLUSH PRIVILEGES;
```

4. If you disabled binary logging, enable it again as soon as you have created the user, by issuing the following statement:

```
mysql> SET SQL_LOG_BIN=1;
```

5. When the user has been configured, use a `CHANGE MASTER TO` statement to configure the server to use the given credentials for state transfer by distributed recovery or a remote cloning operation. Issue the following statement, replacing *`rpl_user`* and *`password`* with the values used when creating the user:

```
mysql> CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='password' \\
        FOR CHANNEL 'group_replication_recovery';
```

   If these credentials are not set correctly for the `group_replication_recovery` replication channel and the `rpl_user` as shown, the server cannot connect to a donor to carry out state transfer and therefore cannot join the group.

Using Group Replication and the Caching SHA-2 User Credentials Plugin

By default, users created in MySQL 8 use Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”. If the *`rpl_user`* you configure for distributed recovery uses the caching SHA-2 authentication plugin and you are not using Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for the `group_replication_recovery` replication channel, RSA key-pairs are used for password exchange, see Section 6.3.3, “Creating SSL and RSA Certificates and Keys”. You can either copy the public key of the `rpl_user` to the joining member, or configure the donors to provide the public key when requested.

The more secure approach is to copy the public key of the `rpl_user` to the joining member. Then you need to configure the `group_replication_recovery_public_key_path` system variable on the joining member with the path to the public key for the `rpl_user`.

Optionally, a less secure approach is to set `group_replication_recovery_get_public_key=ON` on donors so that they provide the public key of the `rpl_user` to joining members. There is no way to verify the identity of a server, therefore only set `group_replication_recovery_get_public_key=ON` when you are sure there is no risk of server identity being compromised, for example by a man-in-the-middle attack.

18.2.1.4 Launching Group Replication

Once server s1 has been configured and started, install the Group Replication plugin. If you used `plugin_load_add='group_replication.so'` in the option file then the Group Replication plugin is installed and you can proceed to the next step. In the event that you decide to install the plugin manually, connect to the server and issue the following:

```
INSTALL PLUGIN group_replication SONAME 'group_replication.so';
```

Important

The `mysql.session` user must exist before you can load Group Replication. `mysql.session` was added in MySQL version 8.0.2. If your data dictionary was initialized using an earlier version you must perform the MySQL upgrade procedure (see Section 2.11, “Upgrading MySQL”). If the upgrade is not run, Group Replication fails to start with the error message `There was an error when trying to access the server with user: mysql.session@localhost. Make sure the user is present in the server and that mysql_upgrade was ran after a server update.`.

To check that the plugin was installed successfully, issue `SHOW PLUGINS;` and check the output. It should show something like this:

```
mysql> SHOW PLUGINS;
+----------------------------+----------+--------------------+----------------------+-------------+
| Name                       | Status   | Type               | Library              | License     |
+----------------------------+----------+--------------------+----------------------+-------------+
| binlog                     | ACTIVE   | STORAGE ENGINE     | NULL                 | PROPRIETARY |

(...)

| group_replication          | ACTIVE   | GROUP REPLICATION  | group_replication.so | PROPRIETARY |
+----------------------------+----------+--------------------+----------------------+-------------+
```

18.2.1.5 Bootstrapping the Group

The process of starting a group for the first time is called bootstrapping. You use the `group_replication_bootstrap_group` system variable to bootstrap a group. The bootstrap should only be done by a single server, the one that starts the group and only once. This is why the value of the `group_replication_bootstrap_group` option was not stored in the instance's option file. If it is saved in the option file, upon restart the server automatically bootstraps a second group with the same name. This would result in two distinct groups with the same name. The same reasoning applies to stopping and restarting the plugin with this option set to `ON`. Therefore to safely bootstrap the group, connect to s1 and issue:

```
mysql> SET GLOBAL group_replication_bootstrap_group=ON;
mysql> START GROUP_REPLICATION;
mysql> SET GLOBAL group_replication_bootstrap_group=OFF;
```

Once the `START GROUP_REPLICATION` statement returns, the group has been started. You can check that the group is now created and that there is one member in it:

```
mysql> SELECT * FROM performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------+-------------+----------
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST | MEMBER_PORT | MEMBER_STA
+---------------------------+--------------------------------------+-------------+-------------+----------
| group_replication_applier | ce9be252-2b71-11e6-b8f4-00212844f856 |   s1        |       3306  | ONLINE
```

```
+--------------------------+---------------------------------------+-------------+-------------+-------------
```

The information in this table confirms that there is a member in the group with the unique identifier `ce9be252-2b71-11e6-b8f4-00212844f856`, that it is `ONLINE` and is at `s1` listening for client connections on port `3306`.

For the purpose of demonstrating that the server is indeed in a group and that it is able to handle load, create a table and add some content to it.

```
mysql> CREATE DATABASE test;
mysql> USE test;
mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 TEXT NOT NULL);
mysql> INSERT INTO t1 VALUES (1, 'Luis');
```

Check the content of table `t1` and the binary log.

```
mysql> SELECT * FROM t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 | Luis |
+----+------+

mysql> SHOW BINLOG EVENTS;
+---------------+-----+----------------+-----------+-------------+---------------------------------------
| Log_name      | Pos | Event_type     | Server_id | End_log_pos | Info
+---------------+-----+----------------+-----------+-------------+---------------------------------------
| binlog.000001 |   4 | Format_desc    |         1 |         123 | Server ver: 8.0.19-log, Binlog ver: 4
| binlog.000001 | 123 | Previous_gtids |         1 |         150 |
| binlog.000001 | 150 | Gtid           |         1 |         211 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaa
| binlog.000001 | 211 | Query          |         1 |         270 | BEGIN
| binlog.000001 | 270 | View_change    |         1 |         369 | view_id=14724817264259180:1
| binlog.000001 | 369 | Query          |         1 |         434 | COMMIT
| binlog.000001 | 434 | Gtid           |         1 |         495 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaa
| binlog.000001 | 495 | Query          |         1 |         585 | CREATE DATABASE test
| binlog.000001 | 585 | Gtid           |         1 |         646 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaa
| binlog.000001 | 646 | Query          |         1 |         770 | use `test`; CREATE TABLE t1 (c1 INT PRIMARY
| binlog.000001 | 770 | Gtid           |         1 |         831 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aaa
| binlog.000001 | 831 | Query          |         1 |         899 | BEGIN
| binlog.000001 | 899 | Table_map      |         1 |         942 | table_id: 108 (test.t1)
| binlog.000001 | 942 | Write_rows     |         1 |         984 | table_id: 108 flags: STMT_END_F
| binlog.000001 | 984 | Xid            |         1 |        1011 | COMMIT /* xid=38 */
+---------------+-----+----------------+-----------+-------------+---------------------------------------
```

As seen above, the database and the table objects were created and their corresponding DDL statements were written to the binary log. Also, the data was inserted into the table and written to the binary log, so it can be used for distributed recovery by state transfer from a donor's binary log.

18.2.1.6 Adding Instances to the Group

At this point, the group has one member in it, server s1, which has some data in it. It is now time to expand the group by adding the other two servers configured previously.

Adding a Second Instance

In order to add a second instance, server s2, first create the configuration file for it. The configuration is similar to the one used for server s1, except for things such as the `server_id`. These different lines are highlighted in the listing below.

```
[mysqld]

#
# Disable other storage engines
#
disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"

#
# Replication configuration parameters
#
server_id=2
gtid_mode=ON
enforce_gtid_consistency=ON
binlog_checksum=NONE

#
# Group Replication configuration
#
transaction_write_set_extraction=XXHASH64
group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
group_replication_start_on_boot=off
group_replication_local_address= "s2:33061"
group_replication_group_seeds= "s1:33061,s2:33061,s3:33061"
group_replication_bootstrap_group= off
```

Similar to the procedure for server s1, with the option file in place you launch the server. Then configure the distributed recovery credentials as follows. The commands are the same as used when setting up server s1 as the user is shared within the group. This member needs to have the same replication user configured in Section 18.2.1.3, “User Credentials”. If you are relying on distributed recovery to configure the user on all members, when s2 connects to the seed s1 the replication user is replicated or cloned to s1. If you did not have binary logging enabled when you configured the user credentials on s1, and a remote cloning operation is not used for state transfer, you must create the replication user on s2. In this case, connect to s2 and issue:

```
SET SQL_LOG_BIN=0;
CREATE USER rpl_user@'%' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%';
GRANT BACKUP_ADMIN ON *.* TO rpl_user@'%';
SET SQL_LOG_BIN=1;
CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='password' \\
 FOR CHANNEL 'group_replication_recovery';
```

Tip

If you are using the caching SHA-2 authentication plugin, the default in MySQL 8, see Using Group Replication and the Caching SHA-2 User Credentials Plugin.

If necessary, install the Group Replication plugin, see Section 18.2.1.4, “Launching Group Replication”.

Start Group Replication and s2 starts the process of joining the group.

```
mysql> START GROUP_REPLICATION;
```

Unlike the previous steps that were the same as those executed on s1, here there is a difference in that you do *not* need to boostrap the group because the group already exiists. In other words on s2 `group_replication_bootstrap_group` is set to off, and you do not issue `SET GLOBAL group_replication_bootstrap_group=ON;` before starting Group Replication, because the group has already been created and bootstrapped by server s1. At this point server s2 only needs to be added to the already existing group.

Tip

When Group Replication starts successfully and the server joins the group it checks the `super_read_only` variable. By setting `super_read_only` to ON in the member's configuration file, you can ensure that servers which fail when starting Group Replication for any reason do not accept transactions. If the server should join the group as read-write instance, for example as the primary in a single-primary group or as a member of a multi-primary group, when the `super_read_only` variable is set to ON then it is set to OFF upon joining the group.

Checking the `performance_schema.replication_group_members` table again shows that there are now two *ONLINE* servers in the group.

```
mysql> SELECT * FROM performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------+-------------+--------------
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE
+---------------------------+--------------------------------------+-------------+-------------+--------------
| group_replication_applier | 395409e1-6dfa-11e6-970b-00212844f856 |   s1        |        3306 | ONLINE
| group_replication_applier | ac39f1e6-6dfa-11e6-a69d-00212844f856 |   s2        |        3306 | ONLINE
+---------------------------+--------------------------------------+-------------+-------------+--------------
```

When s2 attempted to join the group, Section 18.4.3, “Distributed Recovery” ensured that s2 applied the same transactions which s1 had applied. Once this process completed, s2 could join the group as a member, and at this point it is marked as ONLINE. In other words it must have already caught up with server s1 automatically. Once s2 is ONLINE, it then begins to process transactions with the group. Verify that s2 has indeed synchronized with server s1 as follows.

```
mysql> SHOW DATABASES LIKE 'test';
+-----------------+
| Database (test) |
+-----------------+
| test            |
+-----------------+

mysql> SELECT * FROM test.t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 | Luis |
+----+------+

mysql> SHOW BINLOG EVENTS;
+---------------+------+----------------+-----------+-------------+-----------------------------------------
| Log_name      | Pos  | Event_type     | Server_id | End_log_pos | Info
+---------------+------+----------------+-----------+-------------+-----------------------------------------
| binlog.000001 |    4 | Format_desc    |         2 |         123 | Server ver: 8.0.19-log, Binlog ver: 4
| binlog.000001 |  123 | Previous_gtids |         2 |         150 |
| binlog.000001 |  150 | Gtid           |         1 |         211 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 |  211 | Query          |         1 |         270 | BEGIN
| binlog.000001 |  270 | View_change    |         1 |         369 | view_id=14724832985483517:1
| binlog.000001 |  369 | Query          |         1 |         434 | COMMIT
| binlog.000001 |  434 | Gtid           |         1 |         495 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 |  495 | Query          |         1 |         585 | CREATE DATABASE test
| binlog.000001 |  585 | Gtid           |         1 |         646 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 |  646 | Query          |         1 |         770 | use `test`; CREATE TABLE t1 (c1 INT PRIMAR
| binlog.000001 |  770 | Gtid           |         1 |         831 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 |  831 | Query          |         1 |         890 | BEGIN
| binlog.000001 |  890 | Table_map      |         1 |         933 | table_id: 108 (test.t1)
| binlog.000001 |  933 | Write_rows     |         1 |         975 | table_id: 108 flags: STMT_END_F
| binlog.000001 |  975 | Xid            |         1 |        1002 | COMMIT /* xid=30 */
| binlog.000001 | 1002 | Gtid           |         1 |        1063 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 | 1063 | Query          |         1 |        1122 | BEGIN
```

```
| binlog.000001 | 1122 | View_change    |         1 |        1261 | view_id=14724832985483517:2
| binlog.000001 | 1261 | Query          |         1 |        1326 | COMMIT
+---------------+------+----------------+-----------+-------------+-----------------------------------
```

As seen above, the second server has been added to the group and it has replicated the changes from server s1 automatically. In other words, the transactions applied on s1 up to the point in time that s2 joined the group have been replicated to s2.

Adding Additional Instances

Adding additional instances to the group is essentially the same sequence of steps as adding the second server, except that the configuration has to be changed as it had to be for server s2. To summarise the required commands:

1. Create the configuration file

```
[mysqld]

#
# Disable other storage engines
#
disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"

#
# Replication configuration parameters
#
server_id=3
gtid_mode=ON
enforce_gtid_consistency=ON
binlog_checksum=NONE

#
# Group Replication configuration
#
group_replication_group_name="aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa"
group_replication_start_on_boot=off
group_replication_local_address= "s3:33061"
group_replication_group_seeds= "s1:33061,s2:33061,s3:33061"
group_replication_bootstrap_group= off
```

2. Start the server and connect to it. Configure the distributed recovery credentials for the group_replication_recovery channel.

```
SET SQL_LOG_BIN=0;
CREATE USER rpl_user@'%' IDENTIFIED BY 'password';
GRANT REPLICATION SLAVE ON *.* TO rpl_user@'%';
GRANT BACKUP_ADMIN ON *.* TO rpl_user@'%';
FLUSH PRIVILEGES;
SET SQL_LOG_BIN=1;
CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='password'  \\
FOR CHANNEL 'group_replication_recovery';
```

4. Install the Group Replication plugin and start it.

```
INSTALL PLUGIN group_replication SONAME 'group_replication.so';
START GROUP_REPLICATION;
```

At this point server s3 is booted and running, has joined the group and caught up with the other servers in the group. Consulting the `performance_schema.replication_group_members` table again confirms this is the case.

```
mysql> SELECT * FROM performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------+-------------+--------------
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST | MEMBER_PORT | MEMBER_STATE
+---------------------------+--------------------------------------+-------------+-------------+--------------
| group_replication_applier | 395409e1-6dfa-11e6-970b-00212844f856 |   s1        |       3306  | ONLINE
| group_replication_applier | 7eb217ff-6df3-11e6-966c-00212844f856 |   s3        |       3306  | ONLINE
| group_replication_applier | ac39f1e6-6dfa-11e6-a69d-00212844f856 |   s2        |       3306  | ONLINE
+---------------------------+--------------------------------------+-------------+-------------+--------------
```

Issuing this same query on server s2 or server s1 yields the same result. Also, you can verify that server s3 has caught up:

```
mysql> SHOW DATABASES LIKE 'test';
+-----------------+
| Database (test) |
+-----------------+
| test            |
+-----------------+

mysql> SELECT * FROM test.t1;
+----+------+
| c1 | c2   |
+----+------+
|  1 | Luis |
+----+------+

mysql> SHOW BINLOG EVENTS;
+---------------+------+----------------+-----------+-------------+-----------------------------------------
| Log_name      | Pos  | Event_type     | Server_id | End_log_pos | Info
+---------------+------+----------------+-----------+-------------+-----------------------------------------
| binlog.000001 |    4 | Format_desc    |         3 |         123 | Server ver: 8.0.19-log, Binlog ver: 4
| binlog.000001 |  123 | Previous_gtids |         3 |         150 |
| binlog.000001 |  150 | Gtid           |         1 |         211 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 |  211 | Query          |         1 |         270 | BEGIN
| binlog.000001 |  270 | View_change    |         1 |         369 | view_id=14724832985483517:1
| binlog.000001 |  369 | Query          |         1 |         434 | COMMIT
| binlog.000001 |  434 | Gtid           |         1 |         495 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 |  495 | Query          |         1 |         585 | CREATE DATABASE test
| binlog.000001 |  585 | Gtid           |         1 |         646 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 |  646 | Query          |         1 |         770 | use `test`; CREATE TABLE t1 (c1 INT PRIMAR
| binlog.000001 |  770 | Gtid           |         1 |         831 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 |  831 | Query          |         1 |         890 | BEGIN
| binlog.000001 |  890 | Table_map      |         1 |         933 | table_id: 108 (test.t1)
| binlog.000001 |  933 | Write_rows     |         1 |         975 | table_id: 108 flags: STMT_END_F
| binlog.000001 |  975 | Xid            |         1 |        1002 | COMMIT /* xid=29 */
| binlog.000001 | 1002 | Gtid           |         1 |        1063 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 | 1063 | Query          |         1 |        1122 | BEGIN
| binlog.000001 | 1122 | View_change    |         1 |        1261 | view_id=14724832985483517:2
| binlog.000001 | 1261 | Query          |         1 |        1326 | COMMIT
| binlog.000001 | 1326 | Gtid           |         1 |        1387 | SET @@SESSION.GTID_NEXT= 'aaaaaaaa-aaaa-aa
| binlog.000001 | 1387 | Query          |         1 |        1446 | BEGIN
| binlog.000001 | 1446 | View_change    |         1 |        1585 | view_id=14724832985483517:3
| binlog.000001 | 1585 | Query          |         1 |        1650 | COMMIT
+---------------+------+----------------+-----------+-------------+-----------------------------------------
```

18.2.2 Deploying Group Replication Locally

The most common way to deploy Group Replication is using multiple server instances, to provide high availability. It is also possible to deploy Group Replication locally, for example for testing purposes. This section explains how you can deploy Group Replication locally.

Important

Group Replication is usually deployed on multiple hosts because this ensures that high-availability is provided. The instructions in this section are not suitable for production deployments because all MySQL server instances are running on the same single host. In the event of failure of this host, the whole group fails. Therefore this information should be used for testing purposes and it should not be used in a production environments.

This section explains how to create a replication group with three MySQL Server instances on one physical machine. This means that three data directories are needed, one per server instance, and that you need to configure each instance independently. This - procedure assumes that MySQL Server was downloaded and unpacked - into the directory named `mysql-8.0`. Each MySQL server instance requires a specific data directory. Create a directory named `data`, then in that directory create a subdirectory for each server instance, for example s1, s2 and s3, and initialize each one.

```
mysql-8.0/bin/mysqld --initialize-insecure --basedir=$PWD/mysql-8.0 --datadir=$PWD/data/s1
mysql-8.0/bin/mysqld --initialize-insecure --basedir=$PWD/mysql-8.0 --datadir=$PWD/data/s2
mysql-8.0/bin/mysqld --initialize-insecure --basedir=$PWD/mysql-8.0 --datadir=$PWD/data/s3
```

Inside `data/s1`, `data/s2`, `data/s3` is an initialized data directory, containing the mysql system database and related tables and much more. To learn more about the initialization procedure, see Section 2.10.1, “Initializing the Data Directory”.

Warning

Do not use `-initialize-insecure` in production environments, it is only used here to simplify the tutorial. For more information on security settings, see Section 18.5, “Group Replication Security”.

Configuration of Local Group Replication Members

When you are following Section 18.2.1.2, “Configuring an Instance for Group Replication”, you need to add configuration for the data directories added in the previous section. For example:

```
[mysqld]

# server configuration
datadir=<full_path_to_data>/data/s1
basedir=<full_path_to_bin>/mysql-8.0/

port=24801
socket=<full_path_to_sock_dir>/s1.sock
```

These settings configure MySQL server to use the data directory created earlier and which port the server should open and start listening for incoming connections.

Note

The non-default port of 24801 is used because in this tutorial the three server instances use the same hostname. In a setup with three different machines this would not be required.

Group Replication requires a network connection between the members, which means that each member must be able to resolve the network address of all of the other members. For example in this tutorial all three instances run on one machine, so to ensure that the members can contact each other you could add a line to the option file such as `report_host=127.0.0.1`.

Then each member needs to be able to connect to the other members on their `group_replication_local_address`. For example in the option file of member s1 add:

```
group_replication_local_address= "127.0.0.1:24901"
group_replication_group_seeds= "127.0.0.1:24901,127.0.0.1:24902,127.0.0.1:24903"
```

This configures s1 to use port 24901 for internal group communication with seed members. For each server instance you want to add to the group, make these changes in the option file of the member. For each member you must ensure a unique address is specified, so use a unique port per instance for `group_replication_local_address`. Usually you want all members to be able to serve as seeds for members that are joining the group and have not got the transactions processed by the group. In this case, add all of the ports to `group_replication_group_seeds` as shown above.

The remaining steps of Section 18.2.1, “Deploying Group Replication in Single-Primary Mode” apply equally to a group which you have deployed locally in this way.

18.3 Monitoring Group Replication

Use the Perfomance Schema tables to monitor Group Replication, assuming that the Performance Schema is enabled. Group Replication adds the following tables:

- `performance_schema.replication_group_member_stats`
- `performance_schema.replication_group_members`

These Perfomance Schema replication tables also show information about Group Replication:

- `performance_schema.replication_connection_status` shows information regarding Group Replication, for example the transactions that have been received from the group and queued in the applier queue (the relay log).
- `performance_schema.replication_applier_status` shows the state of the Group Replication related channels and threads If there are many different worker threads applying transactions, then the worker tables can also be used to monitor what each worker thread is doing.

The replication channels created by the Group Replication plugin are named:

- `group_replication_recovery` - This channel is used for the replication changes that are related to the distributed recovery phase.
- `group_replication_applier` - This channel is used for the incoming changes from the group. This is the channel used to apply transactions coming directly from the group.

The following sections describe how to interpret the information available.

18.3.1 Group Replication Server States

There are various states that a server instance can be in. If servers are communicating properly, all report the same states for all servers. However, if there is a network partition, or a server leaves the group, then different information could be reported, depending on which server is queried. If the server has left the group then it cannot report updated information about the other servers' states. If there is a partition, such that quorum is lost, servers are not able to coordinate between themselves. As a consequence, they cannot guess what the status of different servers is. Therefore, instead of guessing their state they report that some servers are unreachable.

Table 18.1 Server State

Field	Description	Group Synchronized
ONLINE	The member is ready to serve as a fully functional group member, meaning that the client can connect and start executing transactions.	Yes
RECOVERING	The member is in the process of becoming an active member of the group and is currently going through the recovery process, receiving state transfer from a donor.	No
OFFLINE	The Group Replication plugin is loaded but the member does not belong to any group.	No
ERROR	The member is in an error state and is not functioning correctly as a group member. Depending on the exit action set by group_replication_exit_state_action, the member is in read-only mode (super_read_only=ON) and could also be in offline mode (offline_mode=ON). Note that a server in offline mode following the OFFLINE_MODE exit action is displayed with ERROR status, not OFFLINE. A server with the exit action ABORT_SERVER shuts down and is removed from the view of the group.	No
UNREACHABLE	Whenever the local failure detector suspects that a given server is not reachable, because for example it was disconnected involuntarily, it shows that server's state as UNREACHABLE.	No

18.3.2 The replication_group_members Table

The performance_schema.replication_group_members table is used for monitoring the status of the different server instances that are members of the group. The information in the table is updated whenever there is a view change, for example when the configuration of the group is dynamically changed when a new member joins. At that point, servers exchange some of their metadata to synchronize themselves and continue to cooperate together. The information is shared between all the server instances that are members of the replication group, so information on all the group members can be queried from any member. This table can be used to get a high level view of the state of a replication group, for example by issuing:

```
SELECT * FROM performance_schema.replication_group_members;
+---------------------------+--------------------------------------+-------------+-------------+---------
| CHANNEL_NAME              | MEMBER_ID                            | MEMBER_HOST  | MEMBER_PORT | MEMBER_S
+---------------------------+--------------------------------------+-------------+-------------+---------
| group_replication_applier | 041f26d8-f3f3-11e8-adff-080027337932 | example1     |      3306   | ONLINE
| group_replication_applier | f60a3e10-f3f2-11e8-8258-080027337932 | example2     |      3306   | ONLINE
| group_replication_applier | fc890014-f3f2-11e8-a9fd-080027337932 | example3     |      3306   | ONLINE
+---------------------------+--------------------------------------+-------------+-------------+---------
```

Based on this result we can see that the group consists of three members, each member's host and port number which clients use to connect to the member, and the server_uuid of the member. The MEMBER_STATE column shows one of the Section 18.3.1, “Group Replication Server States”, in this case

it shows that all three members in this group are `ONLINE`, and the `MEMBER_ROLE` column shows that there are two secondaries, and a single primary. Therefore this group must be running in single-primary mode. The `MEMBER_VERSION` column can be useful when you are upgrading a group and are combining members running different MySQL versions. See Section 18.3.1, “Group Replication Server States” for more information.

For more information about the `Member_host` value and its impact on the distributed recovery process, see Section 18.2.1.3, “User Credentials”.

18.3.3 The replication_group_member_stats Table

Each member in a replication group certifies and applies transactions received by the group. Statistics regarding the certifier and applier procedures are useful to understand how the applier queue is growing, how many conflicts have been found, how many transactions were checked, which transactions are committed everywhere, and so on.

The `performance_schema.replication_group_member_stats` table provides group-level information related to the certification process, and also statistics for the transactions received and originated by each individual member of the replication group. The information is shared between all the server instances that are members of the replication group, so information on all the group members can be queried from any member. Note that refreshing of statistics for remote members is controlled by the message period specified in the `group_replication_flow_control_period` option, so these can differ slightly from the locally collected statistics for the member where the query is made. To use this table to monitor a Group Replication member, issue:

```
mysql> SELECT * FROM performance_schema.replication_group_member_stats\G
```

These fields are important for monitoring the performance of the members connected in the group. For example, suppose that one of the group’s members always reports a large number of transactions in its queue compared to other members. This means that the member is delayed and is not able to keep up to date with the other members of the group. Based on this information, you could decide to either remove the member from the group, or delay the processing of transactions on the other members of the group in order to reduce the number of queued transactions. This information can also help you to decide how to adjust the flow control of the Group Replication plugin, see Section 18.6.2, “Flow Control”.

18.4 Group Replication Operations

This section explains common operations for managing groups.

18.4.1 Configuring an Online Group

You can configure an online group while Group Replication is running by using a set of UDFs, which rely on a group action coordinator. These UDFs are installed by the Group Replication plugin in version 8.0.13 and higher. This section describes how changes are made to a running group, and the available UDFs.

Important

For the coordinator to be able to configure group wide actions on a running group, all members must be running MySQL 8.0.13 or higher and have the UDFs installed.

To use the UDFs, connect to a member of the running group and issue the UDF with the `SELECT` statement. The Group Replication plugin processes the action and its parameters and the coordinator sends it to all members which are visible to the member where you issued the UDF. If the action is accepted, all members execute the action and send a termination message when completed. Once all members declare the action as finished, the invoking member returns the result to the client.

When configuring a whole group, the distributed nature of the operations means that they interact with many processes of the Group Replication plugin, and therefore you should observe the following:

You can issue configuration operations everywhere. If you want to make member A the new primary you do not need to invoke the operation on member A. All operations are sent and executed in a coordinated way on all group members. Also, this distributed execution of an operation has a different ramification: if the invoking member dies, any already running configuration process continues to run on other members. In the unlikely event that the invoking member dies, you can still use the monitoring features to ensure other members complete the operation successfully.

All members must be online. To simplify the migration or election processes and guarantee they are as fast as possible, the group must not contain any member currently in the distributed recovery process, otherwise the configuration action is rejected by the member where you issue the statement.

No members can join a group during a configuration change. Any member that attempts to join the group during a coordinated configuration change leaves the group and cancels its join process.

Only one configuration at once. A group which is executing a configuration change cannot accept any other group configuration change, because concurrent configuration operations could lead to member divergence.

All members must be running MySQL 8.0.13 or higher. Due to the distributed nature of the configuration actions, all members must recognize them in order to execute them. The operation is therefore rejected if any server running MySQL Server version 8.0.12 or lower is present in the group.

18.4.1.1 Changing a Group's Primary Member

This section explains how to change which member of a single-primary group is the primary. The function used to change a group's mode can be run on any member.

Changing which Member is Primary

Use the `group_replication_set_as_primary()` UDF to change which member is the primary in a single-primary group. This function has no effect if issued on a member of a multi-primary group. Only a primary member can write to the group, so if an asynchronous channel is running on that member, no switch is allowed until the asynchronous channel is stopped.

If you issue the UDF on a member running a MySQL Server version from 8.0.17, and all members are running MySQL Server version 8.0.17 or higher, you can only specify a new primary member that is running the lowest MySQL Server version in the group, based on the patch version. This safeguard is applied to ensure the group maintains compatibility with new functions. If any member is running a MySQL Server version between MySQL 8.0.13 and MySQL 8.0.16, this safeguard is not enforced for the group and you can specify any new primary member, but it is recommended to select a primary that is running the lowest MySQL Server version in the group.

Pass in the `server_uuid` of the member which you want to become the new primary of the group by issuing:

```
SELECT group_replication_set_as_primary(member_uuid);
```

While the action runs, you can check its progress by issuing:

```
SELECT event_name, work_completed, work_estimated FROM performance_schema.events_stages_current WHERE event
+--------------------------------------------------------------------------------+----------------+-----
| event_name                                                                     | work_completed | work_
```

```
+----------------------------------------------------------------------------+---------------+---------
| stage/group_rpl/Primary Election: Waiting for members to turn on super_read_only |             3 |
+----------------------------------------------------------------------------+---------------+---------
```

18.4.1.2 Changing a Group's Mode

This section explains how to change the mode which a group is running in, either single or multi-primary. The functions used to change a group's mode can be run on any member.

Changing to Single-Primary Mode

Use the `group_replication_switch_to_single_primary_mode()` UDF to change a group running in multi-primary mode to single-primary mode by issuing:

```
SELECT group_replication_switch_to_single_primary_mode()
```

When you change to single-primary mode, strict consistency checks are also disabled on all group members, as required in single-primary mode (`group_replication_enforce_update_everywhere_checks=OFF`).

If no string is passed in, the election of the new primary in the resulting single-primary group follows the election policies described in Section 18.1.3.1, “Single-Primary Mode”. To override the election process and configure a specific member of the multi-primary group as the new primary in the process, get the `server_uuid` of the member and pass it to `group_replication_switch_to_single_primary_mode()`. For example issue:

```
SELECT group_replication_switch_to_single_primary_mode(member_uuid);
```

If you issue the UDF on a member running a MySQL Server version from 8.0.17, and all members are running MySQL Server version 8.0.17 or higher, you can only specify a new primary member that is running the lowest MySQL Server version in the group, based on the patch version. This safeguard is applied to ensure the group maintains compatibility with new functions. If you do not specify a new primary member, the election process considers the patch version of the group members.

If any member is running a MySQL Server version between MySQL 8.0.13 and MySQL 8.0.16, this safeguard is not enforced for the group and you can specify any new primary member, but it is recommended to select a primary that is running the lowest MySQL Server version in the group. If you do not specify a new primary member, the election process considers only the major version of the group members.

While the action runs, you can check its progress by issuing:

```
SELECT event_name, work_completed, work_estimated FROM performance_schema.events_stages_current WHERE event_na
+----------------------------------------------------------------------------+----------------+---------------
| event_name                                                                 | work_completed | work_estimated
+----------------------------------------------------------------------------+----------------+---------------
| stage/group_rpl/Primary Switch: waiting for pending transactions to finish |              4 |             20
+----------------------------------------------------------------------------+----------------+---------------
```

Changing to Multi-Primary Mode

Use the `group_replication_switch_to_multi_primary_mode()` UDF to change a group running in single-primary mode to multi-primary mode by issuing:

```
SELECT group_replication_switch_to_multi_primary_mode()
```

After some coordinated group operations to ensure the safety and consistency of your data, all members which belong to the group become primaries.

When you change a group that was running in single-primary mode to run in multi-primary mode, members running MySQL 8.0.17 or higher are automatically placed in read-only mode if they are running a higher MySQL server version than the lowest version present in the group. Members running MySQL 8.0.16 or lower do not carry out this check, and are always placed in read-write mode.

While the action runs, you can check its progress by issuing:

```
SELECT event_name, work_completed, work_estimated FROM performance_schema.events_stages_current WHERE even
+-----------------------------------------------------------------------+----------------+----------------+
| event_name                                                            | work_completed | work_estimated |
+-----------------------------------------------------------------------+----------------+----------------+
| stage/group_rpl/Multi-primary Switch: applying buffered transactions |              0 |              1 |
+-----------------------------------------------------------------------+----------------+----------------+
```

18.4.1.3 Using Group Replication Group Write Consensus

This section explains how to inspect and configure the maximum number of consensus instances at any time for a group. This maximum is referred to as the event horizon for a group, and is the maximum number of consensus instances that the group can execute in parallel. This enables you to fine tune the performance of your Group Replication deployment. For example, the default value of 10 is suitable for a group running on a LAN, but for groups operating over a slower network such as a WAN, increase this number to improve performance.

Inspecting a Group's Write Concurrency

Use the `group_replication_get_write_concurrency()` UDF to inspect a group's event horizon value at runtime by issuing:

```
SELECT group_replication_get_write_concurrency();
```

Configuring a Group's Write Concurrency

Use the `group_replication_set_write_concurrency()` UDF to set the maximum number of consensus instances that the system can execute in parallel by issuing:

```
SELECT group_replication_set_write_concurrency(instances);
```

where *`instances`* is the new maximum number of consensus instances. The `GROUP_REPLICATION_ADMIN` privilege is required to use this UDF.

18.4.1.4 Setting a Group's Communication Protocol Version

From MySQL 8.0.16, Group Replication has the concept of a communication protocol for the group. The Group Replication communication protocol version can be managed explicitly, and set to accommodate the oldest MySQL Server version that you want the group to support. This enables groups to be formed from members at different MySQL Server versions while ensuring backward compatibility. Versions from MySQL 5.7.14 allow compression of messages, and versions from MySQL 8.0.16 also allow fragmentation of messages. All members of the group must use the same communication protocol version, so that group members can be at different MySQL Server releases but only send messages that can be understood by all group members.

A MySQL server at version X can only join and reach `ONLINE` status in a replication group if the group's communication protocol version is less than or equal to X. When a new member joins a replication group,

it checks the communication protocol version that is announced by the existing members of the group. If the joining member supports that version, it joins the group and uses the communication protocol that the group has announced, even if the member supports additional communication capabilities. If the joining member does not support the communication protocol version, it is expelled from the group.

If two members attempt to join in the same membership change event, they can only join if the communication protocol version for both members is already compatible with the group's communication protocol version. Members with different communication protocol versions from the group must join in isolation. For example:

- One MySQL Server 8.0.16 instance can successfully join a group that uses the communication protocol version 5.7.24.
- One MySQL Server 5.7.24 instance cannot successfully join a group that uses the communication protocol version 8.0.16.
- Two MySQL Server 8.0.16 instances cannot simultaneously join a group that uses the communication protocol version 5.7.24.
- Two MySQL Server 8.0.16 instances can simultaneously join a group that uses the communication protocol version 8.0.16.

You can inspect the communication protocol in use by a group by using the `group_replication_get_communication_protocol()` UDF, which returns the oldest MySQL Server version that the group supports. All existing members of the group return the same communication protocol version. For example:

```
SELECT group_replication_get_communication_protocol();
+-----------------------------------------------+
| group_replication_get_communication_protocol() |
+-----------------------------------------------+
| 8.0.16                                        |
+-----------------------------------------------+
```

Note that the `group_replication_get_communication_protocol()` UDF returns the minimum MySQL version that the group supports, which might differ from the version number that was passed to the `group_replication_set_communication_protocol()` UDF, and from the MySQL Server version that is installed on the member where you use the UDF.

If you need to change the communication protocol version of a group so that members at earlier releases can join, use the `group_replication_set_communication_protocol()` UDF to specify the MySQL Server version of the oldest member that you want to allow. This makes the group fall back to a compatible communication protocol version if possible. The `GROUP_REPLICATION_ADMIN` privilege is required to use this UDF, and all existing group members must be online when you issue the statement, with no loss of majority. For example:

```
SELECT group_replication_set_communication_protocol("5.7.25");
```

If you upgrade all the members of a replication group to a new MySQL Server release, the group's communication protocol version is not automatically upgraded to match. If you no longer need to support members at earlier releases, you can use the `group_replication_set_communication_protocol()` UDF to set the communication protocol version to the new MySQL Server version to which you have upgraded the members. For example:

```
SELECT group_replication_set_communication_protocol("8.0.16");
```

The `group_replication_set_communication_protocol()` UDF is implemented as a group action, so it is executed at the same time on all members of the group. The group action starts buffering messages and waits for delivery of any outgoing messages that were already in progress to complete, then changes the communication protocol version and sends the buffered messages. If a member attempts to join the group at any time after you change the communication protocol version, the group members announce the new protocol version.

MySQL InnoDB cluster automatically and transparently manages the communication protocol versions of its members, whenever the cluster topology is changed using AdminAPI operations. An InnoDB cluster always uses the most recent communication protocol version that is supported by all the instances that are currently part of the cluster or joining it. For details, see InnoDB cluster and Group Replication Protocol.

18.4.2 Transaction Consistency Guarantees

One of the major implications of a distributed system such as Group Replication is the consistency guarantees that it provides as a group. In other words, the consistency of the global synchronization of transactions distributed across the members of the group. This section describes how Group Replication handles consistency guarantees depending on the events that occur in a group, and how to best configure your group's consistency guarantees.

18.4.2.1 Understanding Transaction Consistency Guarantees

In terms of distributed consistency guarantees, either in normal or failure repair operations, Group Replication has always been an eventual consistency system. This means that as soon as the incoming traffic slows down or stops, all group members have the same data content. The events that relate to the consistency of a system can be split into control operations, either manual or automatically triggered by failures; and data flow operations.

For Group Replication, the control operations that can be evaluated in terms of consistency are:

- a member joining or leaving, which is covered by Group Replication's Section 18.4.3, “Distributed Recovery” and write protection.
- network failures, which are covered by the fencing modes.
- in single-primary groups, primary failover, which can also be an operation triggered by `group_replication_set_as_primary()`.

Consistency Guarantees and Primary Failover

In a single-primary group, in the event of a primary failover when a secondary is promoted to primary, the new primary can either be made available to application traffic immediately, regardless of how large the replication backlog is, or alternatively access to it can be restricted until the backlog has been applied.

With the first approach, the group takes the minimum time possible to secure a stable group membership after a primary failure by electing a new primary and then allowing data access immediately while it is still applying any possible backlog from the old primary. Write consistency is ensured, but reads can temporarily retrieve stale data while the new primary applies the backlog. For example, if client C1 wrote `A=2 WHERE A=1` on the old primary just before its failure, when client C1 is reconnected to the new primary it could potentially read `A=1` until the new primary applies its backlog and catches up with the state of the old primary before it left the group.

With the second alternative, the system secures a stable group membership after the primary failure and elects a new primary in the same way as the first alternative, but in this case the group then waits until the new primary applies all backlog and only then does it permit data access. This ensures that in a situation

as described previously, when client C1 is reconnected to the new primary it reads `A=2`. However, the trade-off is that the time required to failover is then proportional to the size of the backlog, which on a correctly configured group should be small .

Prior to MySQL 8.0.14 there was no way to configure the failover policy, by default availability was maximized as described in the first approach. In a group with members running MySQL 8.0.14 and higher, you can configure the level of transaction consistency guarantees provided by members during primary failover using the `group_replication_consistency` variable. See Impact of Consistency on Primary Election.

Data Flow Operations

Data flow is relevant to group consistency guarantees due to the reads and writes executed against a group, especially when these operations are distributed across all members. Data flow operations apply to both modes of Group Replication: single-primary and multi-primary, however to make this explanation clearer it is restricted to single-primary mode. The usual way to split incoming read or write transactions across a single-primary group's members is to route writes to the primary and evenly distribute reads to the secondaries. Since the group should behave as a single entity, it is reasonable to expect that writes on the primary are instantaneously available on the secondaries. Although Group Replication is written using Group Communication System (GCS) protocols that implement the Paxos algorithm, some parts of Group Replication are asynchronous, which implies that data is asynchronously applied to secondaries. This means that a client C2 can write `B=2 WHERE B=1` on the primary, immediately connect to a secondary and read `B=1`. This is because the secondary is still applying backlog, and has not applied the transaction which was applied by the primary.

Transaction Synchronization Points

You configure a group's consistency guarantee based on the point at which you want to synchronize transactions across the group. To help you understand the concept, this section simplifies the points of synchronizing transactions across a group to be at the time of a read operation or at the time of a write operation. If data is synchronized at the time of a read, the current client session waits until a given point, which is the point in time that all preceding update transactions have been applied, before it can start executing. With this approach, only this session is affected, all other concurrent data operations are not affected.

If data is synchronized at the time of write, the writing session waits until all secondaries have written their data. Group Replication uses a total order on writes, and therefore this implies waiting for this and all preceding writes that are in secondaries' queues to be applied. Therefore when using this synchronization point, the writing session waits for all secondaries queues to be applied.

Any alternative ensures that in the situation described for client C2 would always read `B=2` even if immediately connected to a secondary. Each alternative has its advantages and disadvantages, which are directly related to your system workload. The following examples describe different types of workloads and advise which point of synchronization is appropriate.

Imagine the following situations:

- you want to load balance your reads without deploying additional restrictions on which server you read from to avoid reading stale data, group writes are much less common than group reads.
- you have a group that has a predominantly read-only data, you want read-write transactions to be applied everywhere once they commit, so that subsequent reads are done on up-to-date data that includes the latest write. This ensures that you do not pay the synchronization cost for every RO transaction, but only on RW ones.

In these cases, you should choose to synchronize on writes.

Imagine the following situations:

- you want to load balance your reads without deploying additional restrictions on which server you read from to avoid reading stale data, group writes are much more common than group reads.
- you want specific transactions in your workload to always read up-to-date data from the group, for example whenever sensitive data is updated (such as credentials for a file or similar data) and you want to enforce that reads retrieve the most up to date value.

In these cases, you should choose to synchronize on reads.

18.4.2.2 Configuring Transaction Consistency Guarantees

Although the Transaction Synchronization Points section explains that conceptually there are two synchronization points from which you can choose: on read or on write, these terms were a simplification and the terms used in Group Replication are: *before* and *after* transaction execution. The consistency level can have a different impact on read-only (RO) and read-write (RW) transactions processed by the group as demonstrated in this section.

- How to Choose a Consistency Level
- Impacts of Consistency Levels
- Impact of Consistency on Primary Election

The following list shows the possible consistency levels that you can configure in Group Replication using the `group_replication_consistency` variable, in order of increasing transaction consistency guarantee:

- `EVENTUAL`

  Both RO and RW transactions do not wait for preceding transactions to be applied before executing. This was the behavior of Group Replication before the `group_replication_consistency` variable was added. A RW transaction does not wait for other members to apply a transaction. This means that a transaction could be externalized on one member before the others. This also means that in the event of a primary failover, the new primary can accept new RO and RW transactions before the previous primary transactions are all applied. RO transactions could result in outdated values, RW transactions could result in a rollback due to conflicts.

- `BEFORE_ON_PRIMARY_FAILOVER`

  New RO or RW transactions with a newly elected primary that is applying backlog from the old primary are held (not applied) until any backlog has been applied. This ensures that when a primary failover happens, intentionally or not, clients always see the latest value on the primary. This guarantees consistency, but means that clients must be able to handle the delay in the event that a backlog is being applied. Usually this delay should be minimal, but it does depend on the size of the backlog.

- `BEFORE`

  A RW transaction waits for all preceding transactions to complete before being applied. A RO transaction waits for all preceding transactions to complete before being executed. This ensures that this transaction reads the latest value by only affecting the latency of the transaction. This reduces the overhead of synchronization on every RW transaction, by ensuring synchronization is used only on RO transactions. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

- `AFTER`

A RW transaction waits until its changes have been applied to all of the other members. This value has no effect on RO transactions. This mode ensures that when a transaction is committed on the local member, any subsequent transaction reads the written value or a more recent value on any group member. Use this mode with a group that is used for predominantly RO operations to ensure that applied RW transactions are applied everywhere once they commit. This could be used by your application to ensure that subsequent reads fetch the latest data which includes the latest writes. This reduces the overhead of synchronization on every RO transaction, by ensuring synchronization is used only on RW transactions. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

- `BEFORE_AND_AFTER`

  A RW transaction waits for 1) all preceding transactions to complete before being applied and 2) until its changes have been applied on other members. A RO transaction waits for all preceding transactions to complete before execution takes place. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

The `BEFORE` and `BEFORE_AND_AFTER` consistency levels can be both used on RO and RW transactions. The `AFTER` consistency level has no impact on RO transactions, because they do not generate changes.

How to Choose a Consistency Level

The different consistency levels provide flexibility to both DBAs, who can use them to set up their infrastructure; and to developers who can use the consistency level that best suits their application's requirements. The following scenarios show how to choose a consistency guarantee level based on how you use your group:

- *Scenario 1* you want to load balance your reads without worrying about stale reads, your group write operations are considerably fewer than your group read operations. In this case, you should choose `AFTER`.
- *Scenario 2* you have a data set that applies a lot of writes and you want to do occasional reads without having to worry about reading stale data. In this case, you should choose `BEFORE`.
- *Scenario 3* you want specific transactions in your workload to always read up-to-date data from the group, so that whenever that sensitive data is updated (such as credentials for a file or similar data) you want to enforce that reads always read the most up to date value. In this case, you should choose `BEFORE`.
- *Scenario 4* you have a group that has predominantly read-only (RO) data, you want your read-write (RW) transactions to be applied everywhere once they commit, so that subsequent reads are done on up-to-date data that includes your latest writes and you do not pay the synchronization on every RO transaction, but only on RW ones. In this case, you should choose `AFTER`.
- *Scenario 5* you have a group that has predominantly read-only data, you want your read-write (RW) transactions to always read up-to-date data from the group and to be applied everywhere once they commit, so that subsequent reads are done on up-to-date data that includes your latest write and you do not pay the synchronization on every read-only (RO) transaction, but only on RW ones. In this case, you should choose `BEFORE_AND_AFTER`.

You have the freedom to choose the scope at which the consistency level is enforced. This is important because consistency levels could have a negative impact on group performance if you set them at a global scope. Therefore you can configure the consistency level of a group by using the `group_replication_consistency` system variable at different scopes.

To enforce the consistency level on the current session, use the session scope:

```
> SET @@SESSION.group_replication_consistency= 'BEFORE';
```

To enforce the consistency level on all sessions, use the global scope:

```
> SET @@GLOBAL.group_replication_consistency= 'BEFORE';
```

The possibility of setting the consistency level on specific sessions enables you to take advantage of scenarios such as:

- *Scenario 6* A given system handles several instructions that do not require a strong consistency level, but one kind of instruction does require strong consistency: managing access permissions to documents;. In this scenario, the system changes access permissions and it wants to be sure that all clients see the correct permission. You only need to `SET @@SESSION.group_replication_consistency= 'AFTER'`, on those instructions and leave the other instructions to run with `EVENTUAL` set at the global scope.
- *Scenario 7* On the same system as described in Scenario 6, every day an instruction needs to do some analytical processing, and as such it requires to always read the most up-to-date data. To achieve this, you only need to `SET @@SESSION.group_replication_consistency= 'BEFORE'` on that specific instruction.

To summarize, you do not need to run all transactions with a specific consistency level, especially if only some transactions actually require it.

Note that all read-write transactions are totally ordered in Group Replication, so even when you set the consistency level to `AFTER` for the current session this transaction waits until its changes are applied on all members, which means waiting for this and all preceding transactions that could be in the secondaries' queues. In practice, the consistency level `AFTER` waits for everything until and including this transaction.

Impacts of Consistency Levels

Another way to classify the consistency levels is in terms of impact on the group, that is, the repercussions that the consistency levels have on the other members.

The `BEFORE` consistency level, apart from being ordered on the transaction stream, only impacts on the local member. That is, it does not require coordination with the other members and does not have repercussions on their transactions. In other words, `BEFORE` only impacts the transactions on which it is used.

The `AFTER` and `BEFORE_AND_AFTER` consistency levels do have side-effects on concurrent transactions executed on other members. These consistency levels make the other members transactions wait if transactions with the `EVENTUAL` consistency level start while a transaction with `AFTER` or `BEFORE_AND_AFTER` is executing. The other members wait until the `AFTER` transaction is committed on that member, even if the other member's transactions have the `EVENTUAL` consistency level. In other words, `AFTER` and `BEFORE_AND_AFTER` impact *all* `ONLINE` group members.

To illustrate this further, imagine a group with 3 members, M1, M2 and M3. On member M1 a client issues:

```
> SET @@SESSION.group_replication_consistency= AFTER;
> BEGIN;
> INSERT INTO t1 VALUES (1);
> COMMIT;
```

Then, while the above transaction is being applied, on member M2 a client issues:

```
> SET SESSION group_replication_consistency= EVENTUAL;
```

In this situation, even though the second transaction's consistency level is `EVENTUAL`, because it starts executing while the first transaction is already in the commit phase on M2, the second transaction has to wait for the first transaction to finish the commit and only then can it execute.

You can only use the consistency levels `BEFORE`, `AFTER` and `BEFORE_AND_AFTER` on `ONLINE` members, attempting to use them on members in other states causes a session error.

Transactions whose consistency level is not `EVENTUAL` hold execution until a timeout, configured by `wait_timeout` value is reached, which defaults to 8 hours. If the timeout is reached an `ER_GR_HOLD_WAIT_TIMEOUT` error is thrown.

Impact of Consistency on Primary Election

This section describes how a group's consistency level impacts on a single-primary group that has elected a new primary. Such a group automatically detects failures and adjusts the view of the members that are active, in other words the membership configuration. Furthermore, if a group is deployed in single-primary mode, whenever the group's membership changes there is a check performed to detect if there is still a primary member in the group. If there is none, a new one is selected from the list of secondary members. Typically, this is known as the secondary promotion.

Given the fact that the system detects failures and reconfigures itself automatically, the user may also expect that once the promotion takes place, the new primary is in the exact state, data-wise, as that of the old one. In other words, the user may expect that there is no backlog of replicated transactions to be applied on the new primary once he is able to read from and write to it. In practical terms, the user may expect that once his application fails-over to the new primary, there would be no chance, even if temporarily, to read old data or write into old data records.

When flow control is activated and properly tuned on a group, there is only a small chance of transiently reading stale data from a newly elected primary immediately after the promotion, as there should not be a backlog, or if there is one it should be small. Moreover, you might have a proxy or middleware layers that govern application accesses to the primary after a promotion and enforce the consistency criteria at that level. If your group members are using MySQL 8.0.14 or higher, you can specify the behavior of the new primary once it is promoted using the `group_replication_consistency` variable, which controls whether a newly elected primary blocks both reads and writes until after the backlog is fully applied or if it behaves in the manner of members running MySQL 8.0.13 or earlier. If the `group_replication_consistency` option was set to `BEFORE_ON_PRIMARY_FAILOVER` on a newly elected primary which has backlog to apply, and transactions are issued against the new primary while it is still applying the backlog, incoming transactions are blocked until the backlog is fully applied. Thus, the following anomalies are prevented:

- No stale reads for read-only and read-write transactions. This prevents stale reads from being externalized to the application by the new primary.
- No spurious roll backs for read-write transactions, due to write-write conflicts with replicated read-write transactions still in the backlog waiting to be applied.
- No read skew on read-write transactions, such as:

```
> BEGIN;
> SELECT x FROM t1; -- x=1 because x=2 is in the backlog;
> INSERT x INTO t2;
> COMMIT;
```

  This query should not cause a conflict but writes outdated values.

To summarize, when `group_replication_consistency` is set to `BEFORE_ON_PRIMARY_FAILOVER` you are choosing to prioritize consistency over availability, because reads and writes are held whenever

a new primary is elected. This is the trade-off you have to consider when configuring your group. It should also be remembered that if flow control is working correctly, backlog should be minimal. Note that the higher consistency levels `BEFORE`, `AFTER`, and `BEFORE_AND_AFTER` also include the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

To guarantee that the group provides the same consistency level regardless of which member is promoted to primary, all members of the group should have `BEFORE_ON_PRIMARY_FAILOVER` (or a higher consistency level) persisted to their configuration. For example on each member issue:

```
> SET PERSIST group_replication_consistency='BEFORE_ON_PRIMARY_FAILOVER';
```

This ensures that the members all behave in the same way, and that the configuration is persisted after a restart of the member.

Although all writes are held when using `BEFORE_ON_PRIMARY_FAILOVER` consistency level, not all reads are blocked to ensure that you can still inspect the server while it is applying backlog after a promotion took place. This is useful for debugging, monitoring, observability and troubleshooting. Some queries that do not modify data are allowed, such as the following:

- `SHOW` statements
- `SET` statements
- `DO` statements
- `EMPTY` statements
- `USE` statements
- using `SELECT` statements against the `performance_schema` and `sys` databases
- using `SELECT` statements against the `PROCESSLIST` table from the `infoschema` database
- `SELECT` statements that do not use tables or user defined functions
- `STOP GROUP_REPLICATION` statements
- `SHUTDOWN` statements
- `RESET PERSIST` statements

A transaction cannot be on-hold forever, and if the time held exceeds `wait_timeout` it returns an `ER_GR_HOLD_WAIT_TIMEOUT` error.

18.4.3 Distributed Recovery

Whenever a member joins or rejoins a replication group, it must catch up with the transactions that were applied by the group members before it joined, or while it was away. This process is called distributed recovery.

The joining member begins by checking the relay log for its `group_replication_applier` channel for any transactions that it already received from the group but did not yet apply. If the joining member was in the group previously, it might find unapplied transactions from before it left, in which case it applies these as a first step. A member that is new to the group does not have anything to apply.

After this, the joining member connects to an online existing member to carry out state transfer. The joining member transfers all the transactions that took place in the group before it joined or while it was away, which are provided by the existing member (called the *donor*). Next, the joining member applies

the transactions that took place in the group while this state transfer was in progress. When this process is complete, the joining member has caught up with the remaining servers in the group, and it begins to participate normally in the group.

Group Replication uses a combination of these methods for state transfer during distributed recovery:

- A remote cloning operation using the clone plugin's function, which is available from MySQL 8.0.17. To enable this method of state transfer, you must install the clone plugin on the group members and the joining member. Group Replication automatically configures the required clone plugin settings and manages the remote cloning operation.
- Replicating from a donor's binary log and applying the transactions on the joining member. This method uses a standard asynchronous replication channel named `group_replication_recovery` that is established between the donor and the joining member.

Group Replication automatically selects the best combination of these methods for state transfer after you issue `START GROUP_REPLICATION` on the joining member. To do this, Group Replication checks which existing members are suitable as donors, how many transactions the joining member needs from a donor, and whether any required transactions are no longer present in the binary log files on any group member. If the transaction gap between the joining member and a suitable donor is large, or if some required transactions are not in any donor's binary log files, Group Replication begins distributed recovery with a remote cloning operation. If there is not a large transaction gap, or if the clone plugin is not installed, Group Replication proceeds directly to state transfer from a donor's binary log.

- During a remote cloning operation, the existing data on the joining member is removed, and replaced with a copy of the donor's data. When the remote cloning operation is complete and the joining member has restarted, state transfer from a donor's binary log is carried out to get the transactions that the group applied while the remote cloning operation was in progress.
- During state transfer from a donor's binary log, the joining member replicates and applies the required transactions from the donor's binary log, applying the transactions as they are received, up to the point where the binary log records that the joining member joined the group (a view change event). While this is in progress, the joining member buffers the new transactions that the group applies. When state transfer from the binary log is complete, the joining member applies the buffered transactions.

When the joining member is up to date with all the group's transactions, it is declared online and can participate in the group as a normal member, and distributed recovery is complete.

18.4.3.1 Cloning for Distributed Recovery

MySQL Server's clone plugin is available from MySQL 8.0.17. If you want to use remote cloning operations for distributed recovery in a group, you must set up existing members and joining members beforehand to support this function. If you do not want to use this function in a group, do not set it up, in which case Group Replication only uses state transfer from the binary log.

To use cloning, at least one existing group member and the joining member must be set up beforehand to support remote cloning operations. As a minimum, you must install the clone plugin on the donor and joining member, grant the `BACKUP_ADMIN` permission to the replication user for distributed recovery, and set the `group_replication_clone_threshold` system variable to an appropriate level. To ensure the maximum availability of donors, it is advisable to set up all current and future group members to support remote cloning operations.

Be aware that a remote cloning operation removes user-created tablespaces and data from the joining member before transferring the data from the donor. If the operation is stopped while in progress, the joining member might be left with partial data or no data. This can be repaired by retrying the remote cloning operation, which Group Replication does automatically.

Prerequisites for Cloning

For full instructions to set up and configure the clone plugin, see Section 5.6.7, “The Clone Plugin” . Detailed prerequisites for a remote cloning operation are covered in Section 5.6.7.3, “Cloning Remote Data” . For Group Replication, note the following key points and differences:

- The donor (an existing group member) and the recipient (the joining member) must have the clone plugin installed and active. For instructions to do this, see Section 5.6.7.1, “Installing the Clone Plugin” .
- The donor and the recipient must run on the same operating system, and must have the same MySQL Server version (which must be MySQL 8.0.17 or above to support the clone plugin). Cloning is therefore not suitable for groups where members run different MySQL Server versions.
- If distributed recovery is configured to use SSL (`group_replication_recovery_use_ssl=ON`), Group Replication applies this setting for remote cloning operations. Group Replication automatically configures the settings for the clone SSL options (`clone_ssl_ca`, `clone_ssl_cert`, and `clone_ssl_key`) to match your settings for the corresponding Group Replication distributed recovery options (`group_replication_recovery_ssl_ca`, `group_replication_recovery_ssl_cert`, and `group_replication_recovery_ssl_key`).
- You do not need to set up a list of valid donors in the `clone_valid_donor_list` system variable for the purpose of joining a replication group. Group Replication configures this setting automatically for you after it selects a donor from the existing group members. Note that remote cloning operations use the server's SQL protocol hostname and port.
- The clone plugin has a number of system variables to manage the network load and performance impact of the remote cloning operation. Group Replication does not configure these settings, so you can review them and set them if you want to, or allow them to default. Note that when a remote cloning operation is used for distributed recovery, the clone plugin's `clone_enable_compression` setting applies to the operation, rather than the Group Replication compression setting.
- To invoke the remote cloning operation on the recipient, Group Replication uses the internal `mysql.session` user, which already has the `CLONE_ADMIN` privilege, so you do not need to set this up.
- As the clone user on the donor for the remote cloning operation, Group Replication uses the replication user that you set up for distributed recovery (which is covered in Section 18.2.1.3, “User Credentials”). You must therefore give the `BACKUP_ADMIN` privilege to this replication user on all group members that support cloning. Also give the privilege to the replication user on joining members when you are configuring them for Group Replication, because they can act as donors after they join the group. To give this privilege to the replication user on existing members, you can issue this statement on each group member individually with binary logging disabled, or on one group member with binary logging enabled:

```
GRANT BACKUP_ADMIN ON *.* TO rpl_user@'%';
```

Threshold for Cloning

When group members have been set up to support cloning, the `group_replication_clone_threshold` system variable specifies a threshold, expressed as a number of transactions, for the use of a remote cloning operation in distributed recovery. If the gap between the transactions on the donor and the transactions on the joining member is larger than this number, a remote cloning operation is used for state transfer to the joining member when this is technically possible. Group Replication calculates whether the threshold has been exceeded based on the `gtid_executed` sets of the existing group members. Using a remote cloning operation in the event

of a large transaction gap lets you add new members to the group without transferring the group's data to the server manually beforehand, and also enables a member that is very out of date to catch up more efficiently.

The default setting for the `group_replication_clone_threshold` Group Replication system variable is extremely high (the maximum permitted sequence number for a transaction in a GTID), so it effectively deactivates cloning wherever state transfer from the binary log is possible. To enable Group Replication to select a remote cloning operation for state transfer where this is more appropriate, set the system variable to specify a number of transactions as the transaction gap above which you want cloning to take place.

Warning

Do not use a low setting for `group_replication_clone_threshold` in an active group. If a number of transactions above the threshold takes place in the group while the remote cloning operation is in progress, the joining member triggers a remote cloning operation again after restarting, and could continue this indefinitely. To avoid this situation, ensure that you set the threshold to a number higher than the number of transactions that you would expect to occur in the group during the time taken for the remote cloning operation.

Group Replication attempts to execute a remote cloning operation regardless of your threshold when state transfer from a donor's binary log is impossible, for example because the transactions needed by the joining member are not available in the binary log on any existing group member. Group Replication identifies this based on the `gtid_purged` sets of the existing group members. You cannot use the `group_replication_clone_threshold` system variable to deactivate cloning when the required transactions are not available in any member's binary log files, because in that situation cloning is the only alternative to transferring data to the joining member manually.

Cloning Operations

When group members and joining members are set up for cloning, Group Replication manages remote cloning operations for you. A remote cloning operation might take some time to complete, depending on the size of the data. See Section 5.6.7.9, “Monitoring Cloning Operations” for information on monitoring the process.

Note

When state transfer is complete, Group Replication restarts the joining member to complete the process. If `group_replication_start_on_boot=OFF` is set on the joining member, you must issue `START GROUP_REPLICATION` manually again following this restart. If `group_replication_start_on_boot=ON` and other settings required to start Group Replication were set in a configuration file or using a `SET PERSIST` statement, you do not need to intervene and the process continues automatically to bring the joining member online.

A remote cloning operation clones settings that are persisted in tables from the donor to the recipient, as well as the data. Group Replication manages the settings that relate specifically to Group Replication channels. Group Replication member settings that are persisted in configuration files, such as the group replication local address, are not cloned and are not changed on the joining member.

The credentials used by the donor for the `group_replication_recovery` replication channel (the replication user and password) are transferred to and used by the joining member after cloning, and they must be valid there. All group members that received state transfer by a remote cloning operation therefore use the same replication user and password for distributed recovery. However, Group Replication preserves the channel settings that relate to the use of SSL, so these are unique to the individual member. If a `PRIVILEGE_CHECKS_USER` account has been used to help secure the replication appliers, this

user account is not used by the joining member after cloning. You must set up the user manually for the applicable replication channels, as described in Section 17.3.3, “Replication Privilege Checks”.

18.4.3.2 Configuring Distributed Recovery

Several aspects of Group Replication's distributed recovery process can be configured to suit your system.

Replication User for Distributed Recovery

Distributed recovery requires a replication user that has the correct permissions so that Group Replication can establish direct member-to-member replication channels. The replication user must also have the correct permissions to act as the clone user on the donor for a remote cloning operation. For instructions to set up this replication user, see Section 18.2.1.3, “User Credentials”.

Manual State Transfer

State transfer from the binary log is Group Replication's base mechanism for distributed recovery, and if the donors and joining members in your replication group are not set up to support cloning, this is the only available option. As state transfer from the binary log is based on classic asynchronous replication, it might take a very long time if the server joining the group does not have the group's data at all, or has data taken from a very old backup image. In this situation, it is therefore recommended that before adding a server to the group, you should set it up with the group's data by transferring a fairly recent snapshot of a server already in the group. This minimizes the time taken for distributed recovery, and reduces the impact on donor servers, since they have to retain and transfer fewer binary log files.

Number of Connection Attempts

For state transfer from the binary log, Group Replication limits the number of attempts a joining member makes when trying to connect to a donor from the pool of donors. If the connection retry limit is reached without a successful connection, the distributed recovery procedure terminates with an error. Note that this limit specifies the total number of attempts that the joining member makes to connect to a donor. For example, if 2 group members are suitable donors, and the connection retry limit is set to 4, the joining member makes 2 attempts to connect to each of the donors before reaching the limit.

The default connection retry limit is 10. You can configure this setting using the `group_replication_recovery_retry_count` system variable. The following command sets the maximum number of attempts to connect to a donor to 5:

```
mysql> SET GLOBAL group_replication_recovery_retry_count= 5;
```

For remote cloning operations, this limit does not apply. Group Replication makes only one connection attempt to each suitable donor for cloning, before starting to attempt state transfer from the binary log.

Sleep Interval for Connection Attempts

For state transfer from the binary log, the `group_replication_recovery_reconnect_interval` system variable defines how much time the distributed recovery process should sleep between donor connection attempts. Note that distributed recovery does not sleep after every donor connection attempt. As the joining member is connecting to different servers and not to the same one repeatedly, it can assume that the problem that affects server A does not affect server B. Distributed recovery therefore suspends only when it has gone through all the possible donors. Once the server joining the group has made one attempt to connect to each of the suitable donors in the group, the distributed recovery process sleeps for the number of seconds configured by the `group_replication_recovery_reconnect_interval` system variable. For example, if 2 group members are suitable donors, and the connection retry limit is set to 4, the joining member makes one attempt to connect to each of the donors, then sleeps for the connection retry interval, then makes one further attempt to connect to each of the donors before reaching the limit.

The default connection retry interval is 60 seconds, and you can change this value dynamically. The following command sets the distributed recovery donor connection retry interval to 120 seconds:

```
mysql> SET GLOBAL group_replication_recovery_reconnect_interval= 120;
```

For remote cloning operations, this interval does not apply. Group Replication makes only one connection attempt to each suitable donor for cloning, before starting to attempt state transfer from the binary log.

Marking the Joining Member Online

When distributed recovery has successfully completed state transfer from the donor to the joining member, the joining member can be marked as online in the group and ready to participate. By default, this is done after the joining member has received and applied all the transactions that it was missing. Optionally, you can allow a joining member to be marked as online when it has received and certified (that is, completed conflict detection for) all the transactions that it was missing, but before it has applied them. If you want to do this, use the `group_replication_recovery_complete_at` system variable to specify the alternative setting `TRANSACTIONS_CERTIFIED`.

SSL and Authentication for Distributed Recovery

You can optionally use SSL for distributed recovery connections between group members. SSL for distributed recovery is configured separately from SSL for normal group communications, which is determined by the server's SSL settings and the `group_replication_ssl_mode` system variable. For distributed recovery connections, dedicated Group Replication distributed recovery SSL system variables are available to configure the use of certificates and ciphers specifically for distributed recovery.

By default, SSL is not used for distributed recovery connections. To activate this, set `group_replication_recovery_use_ssl=ON`, and configure the Group Replication distributed recovery SSL system variables as described in Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”. You need a replication user that is set up to use SSL.

When distributed recovery is configured to use SSL, Group Replication applies this setting for remote cloning operations, as well as for state transfer from a donor's binary log. Group Replication automatically configures the settings for the clone SSL options (`clone_ssl_ca`, `clone_ssl_cert`, and `clone_ssl_key`) to match your settings for the corresponding Group Replication distributed recovery options (`group_replication_recovery_ssl_ca`, `group_replication_recovery_ssl_cert`, and `group_replication_recovery_ssl_key`).

If you are not using SSL for distributed recovery (so `group_replication_recovery_use_ssl` is set to `OFF`), and the replication user account for Group Replication authenticates with the `caching_sha2_password` plugin (which is the default in MySQL 8.0) or the `sha256_password` plugin, RSA key-pairs are used for password exchange. In this case, either use the `group_replication_recovery_public_key_path` system variable to specify the RSA public key file, or use the `group_replication_recovery_get_public_key` system variable to request the public key from the master, as described in Using Group Replication and the Caching SHA-2 User Credentials Plugin.

Compression for Distributed Recovery

From MySQL 8.0.18, you can optionally configure compression for distributed recovery by the method of state transfer from a donor's binary log. Compression can benefit distributed recovery where network bandwidth is limited and the donor has to transfer many transactions to the joining member. The `group_replication_recovery_compression_algorithm` and `group_replication_recovery_zstd_compression_level` system variables configure permitted compression algorithms, and the `zstd` compression level, used when carrying out state transfer from a donor's binary log. For more information, see Section 4.2.6, “Connection Compression Control”.

Note that these compression settings do not apply for remote cloning operations. When a remote cloning operation is used for distributed recovery, the clone plugin's `clone_enable_compression` setting applies.

18.4.3.3 Fault Tolerance for Distributed Recovery

Group Replication's distributed recovery process has a number of built-in measures to ensure fault tolerance in the event of any problems during the process.

The donor for distributed recovery is selected randomly from the existing list of suitable online group members in the current view. Selecting a random donor means that there is a good chance that the same server is not selected more than once when multiple members enter the group. From MySQL 8.0.17, for state transfer from the binary log, the joiner only selects a donor that is running a lower or equal patch version of MySQL Server compared to itself. For earlier releases, all of the online members are allowed to be a donor. For a remote cloning operation, the joiner only selects a donor that is running the same patch version as itself. Note that when the joining member has restarted at the end of the operation, it establishes a connection with a new donor for state transfer from the binary log, which might be a different member from the original donor used for the remote cloning operation.

In the following situations, Group Replication detects an error in distributed recovery, automatically switches over to a new donor, and retries the state transfer:

- *Connection error* - There is an authentication issue or another problem with making the connection to a candidate donor.
- *Replication errors* - One of the replication threads (the receiver or applier threads) being used for state transfer from the binary log fails. Because this method of state transfer uses the existing MySQL replication framework, it is possible that some transient errors could cause errors in the receiver or applier threads.
- *Remote cloning operation errors* - A remote cloning operation fails or is stopped before it completes.
- *Donor leaves the group* - The donor leaves the group, or Group Replication is stopped on the donor, while state transfer is in progress.

In these situations, the new connection following the error is attempted with a new candidate donor. Selecting a different donor in the event of an error means that there is a chance the new candidate donor does not have the same error. If the clone plugin is installed, Group Replication attempts a remote cloning operation with each of the suitable online clone-supporting donors first. If all those attempts fail, Group Replication attempts state transfer from the binary log with all the suitable donors in turn, if that is possible.

Warning

For a remote cloning operation, user-created tablespaces and data on the recipient (the joining member) are dropped before the remote cloning operation begins to transfer the data from the donor. If the remote cloning operation starts but does not complete, the joining member might be left with a partial set of its original data files, or with no user data. Data transferred by the donor is removed from the recipient if the cloning operation is stopped before the data is fully cloned. This situation can be repaired by retrying the cloning operation, which Group Replication does automatically.

In the following situations, the distributed recovery process cannot be completed, and the joining member leaves the group:

- *Purged transactions* - Transactions that are required by the joining member are not present in any online group member's binary log files, and the data cannot be obtained by a remote cloning operation

(because the clone plugin is not installed, or because cloning was attempted with all possible donors but failed). The joining member is therefore unable to catch up with the group.

- *Conflicting transactions* - The joining member already contains some transactions that are not present in the group. If a remote cloning operation was carried out, these transactions would be deleted and lost, because the data directory on the joining member is erased. If state transfer from a donor's binary log was carried out, these transactions could conflict with the group's transactions.
- *Connection retry limit reached* - The joining member has made all the connection attempts allowed by the connection retry limit. You can configure this using the `group_replication_recovery_retry_count` system variable (see Section 18.4.3.2, “Configuring Distributed Recovery”).
- *No more donors* - The joining member has unsuccessfully attempted a remote cloning operation with each of the online clone-supporting donors in turn (if the clone plugin is installed), then has unsuccessfully attempted state transfer from the binary log with each of the suitable online donors in turn, if possible.
- *Joining member leaves the group* - The joining member leaves the group or Group Replication is stopped on the joining member while state transfer is in progress.

If the joining member left the group unintentionally, so in any situation listed above except the last, it proceeds to take the action specified by the `group_replication_exit_state_action` system variable.

18.4.3.4 How Distributed Recovery Works

When Group Replication's distributed recovery process is carrying out state transfer from the binary log, to synchronize the joining member with the donor up to a specific point in time, the joining member and donor make use of GTIDs (see Section 17.1.3, “Replication with Global Transaction Identifiers”). However, GTIDs only provide a means to realize which transactions the joining member is missing. They do not help marking a specific point in time to which the server joining the group must catch up, nor do they convey certification information. This is the job of binary log view markers, which mark view changes in the binary log stream, and also contain additional metadata information, supplying the joining member with missing certification-related data.

This topic explains the role of view changes and the view change identifier, and the steps to carry out state transfer from the binary log.

View and View Changes

A *view* corresponds to a group of members participating actively in the current configuration, in other words at a specific point in time. They are functioning correctly and online in the group.

A *view change* occurs when a modification to the group configuration happens, such as a member joining or leaving. Any group membership change results in an independent view change communicated to all members at the same logical point in time.

A *view identifier* uniquely identifies a view. It is generated whenever a view change happens.

At the group communication layer, view changes with their associated view identifiers mark boundaries between the data exchanged before and after a member joins. This concept is implemented through a binary log event: the"view change log event". The view identifier is recorded to demarcate transactions transmitted before and after changes happen in the group membership.

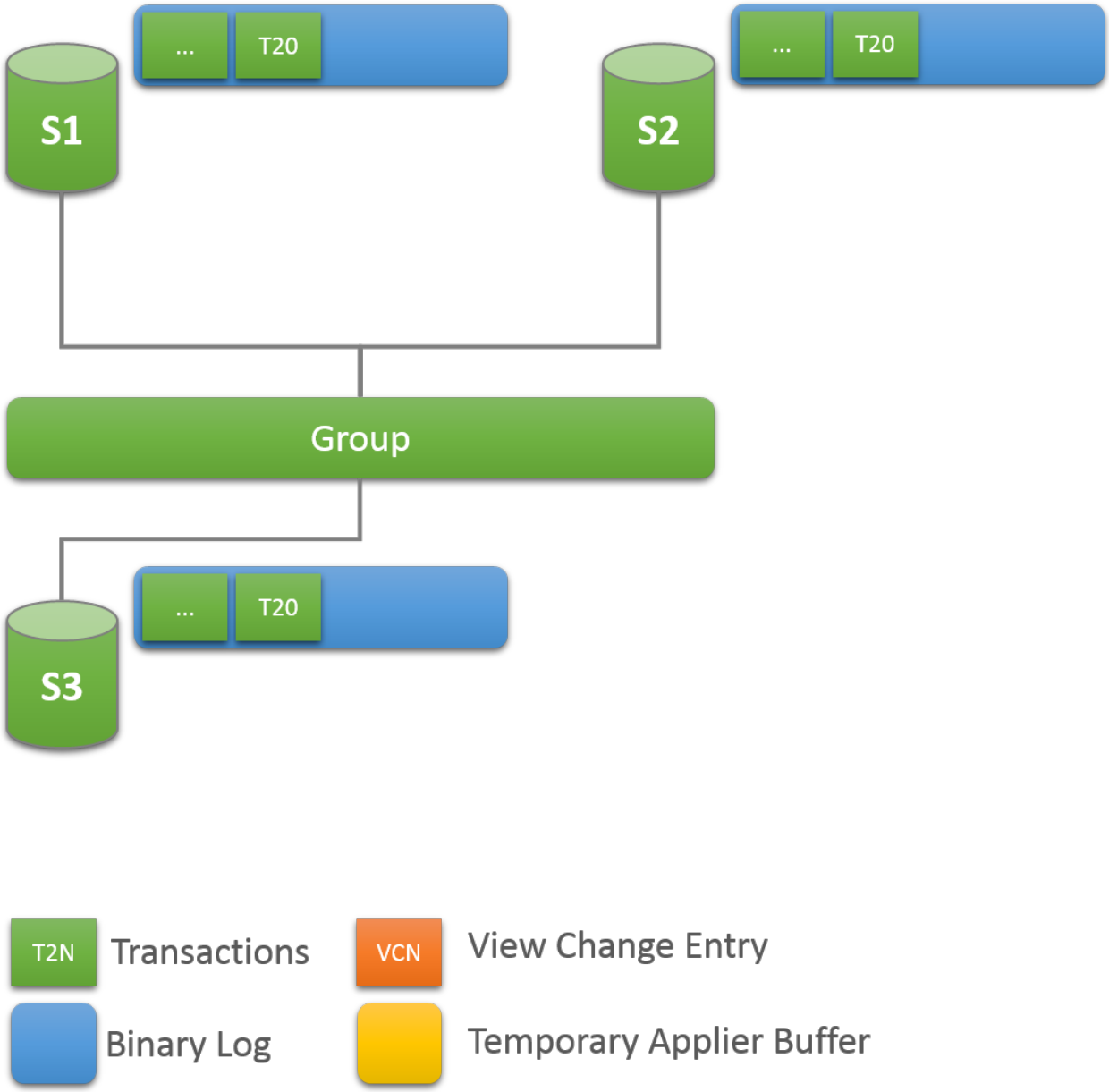
The view identifier itself is built from two parts: a randomly generated part, and a monotonically increasing integer. The randomly generated part is generated when the group is created, and remains unchanged

while there is at least one member in the group. The integer is incremented every time a view change happens. Using these two different parts enables the view identifier to identify incremental group changes caused by members joining or leaving, and also to identify the situation where all members leave the group in a full group shutdown, so no information remains of what view the group was in. Randomly generating part of the identifier when the group is started from the beginning ensures that the data markers in the binary log remain unique, and an identical identifier is not reused after a full group shutdown, as this would cause issues with distributed recovery in the future.

Begin: Stable Group

All servers are online and processing incoming transactions from the group. Some servers may be a little behind in terms of transactions replicated, but eventually they converge. The group acts as one distributed and replicated database.

Figure 18.8 Stable Group

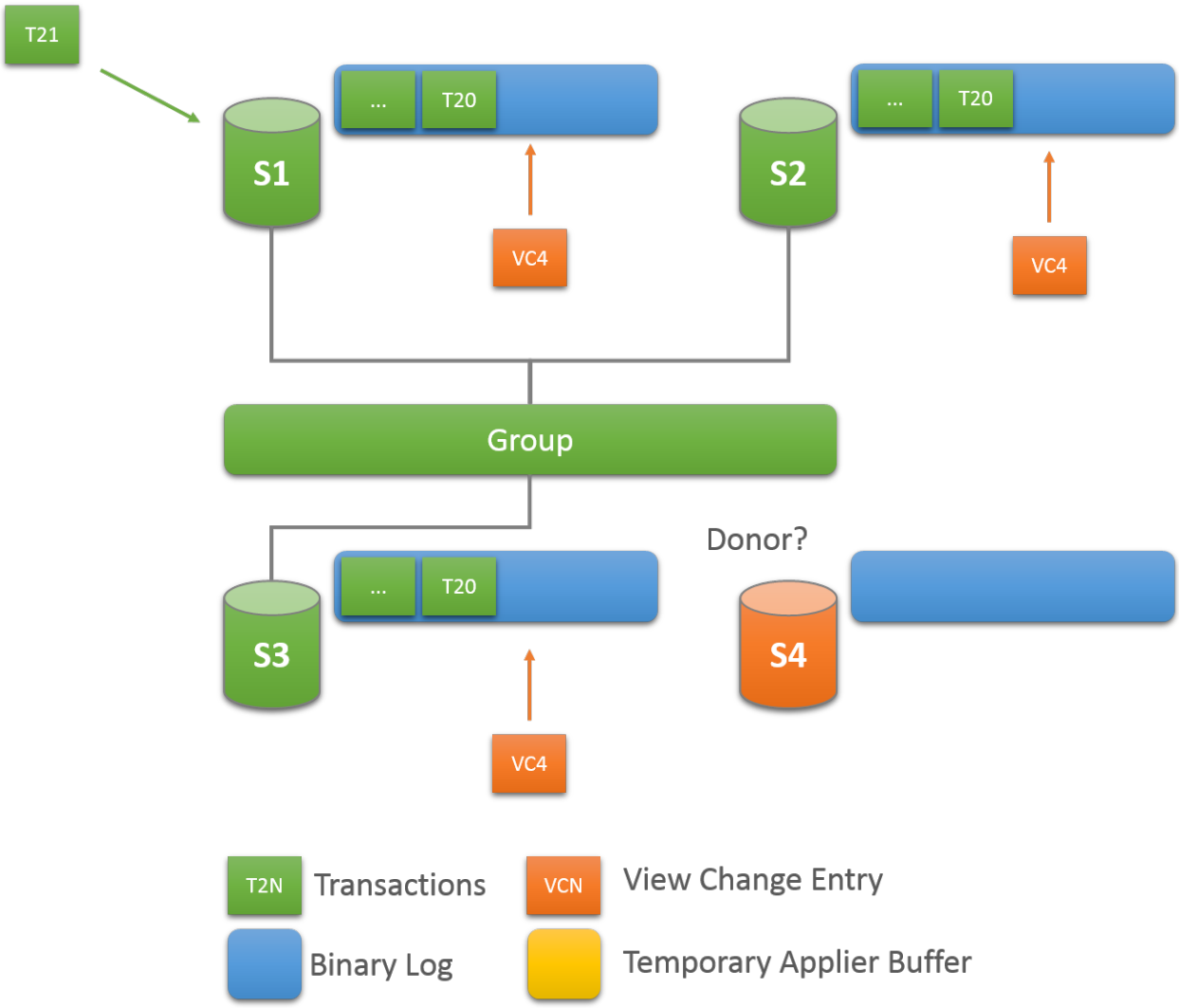


View Change: a Member Joins

Whenever a new member joins the group and therefore a view change is performed, every online server queues a view change log event for execution. This is queued because before the view change, several transactions can be queued on the server to be applied and as such, these belong to the old view. Queuing the view change event after them guarantees a correct marking of when this happened.

Meanwhile, the joining member selects a suitable donor the donor from the list of online servers as stated by the membership service through the view abstraction. A member joins on view 4 and the online members write a view change event to the binary log.

Figure 18.9 A Member Joins



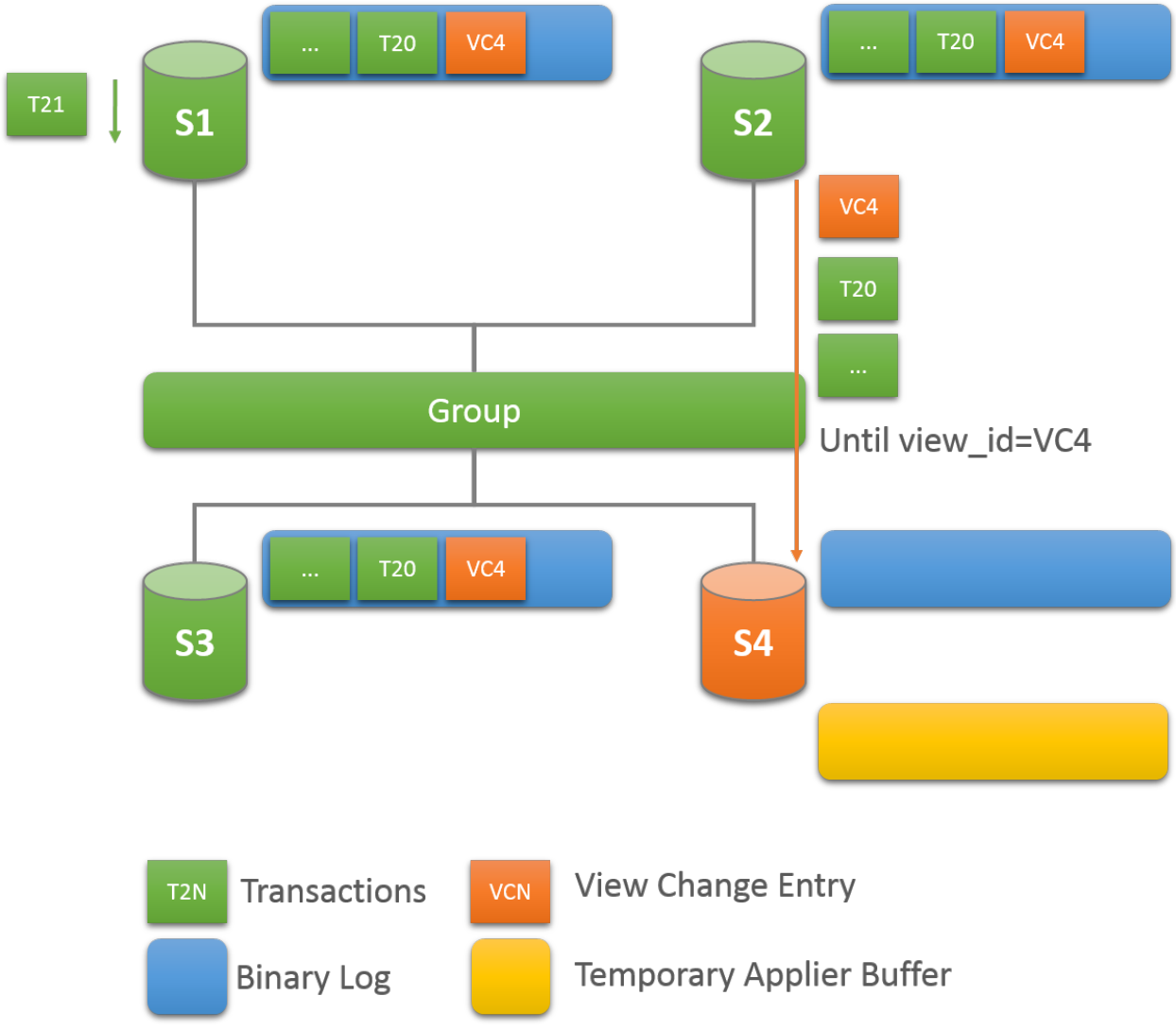
State Transfer: Catching Up

If group members and the joining member are set up with the clone plugin (see Section 18.4.3.1, “Cloning for Distributed Recovery”), and the difference in transactions between the joining member and the group exceeds the threshold set for a remote cloning operation (`group_replication_clone_threshold`), Group Replication begins distributed recovery with a remote cloning operation. A remote cloning operation

is also carried out if required transactions are no longer present in any group member's binary log files. During a remote cloning operation, the existing data on the joining member is removed, and replaced with a copy of the donor's data. When the remote cloning operation is complete and the joining member has restarted, state transfer from a donor's binary log is carried out to get the transactions that the group applied while the remote cloning operation was in progress. If there is not a large transaction gap, or if the clone plugin is not installed, Group Replication proceeds directly to state transfer from a donor's binary log.

For state transfer from a donor's binary log, a connection is established between the joining member and the donor and state transfer begins. This interaction with the donor continues until the server joining the group's applier thread processes the view change log event that corresponds to the view change triggered when the server joining the group came into the group. In other words, the server joining the group replicates from the donor, until it gets to the marker with the view identifier which matches the view marker it is already in.

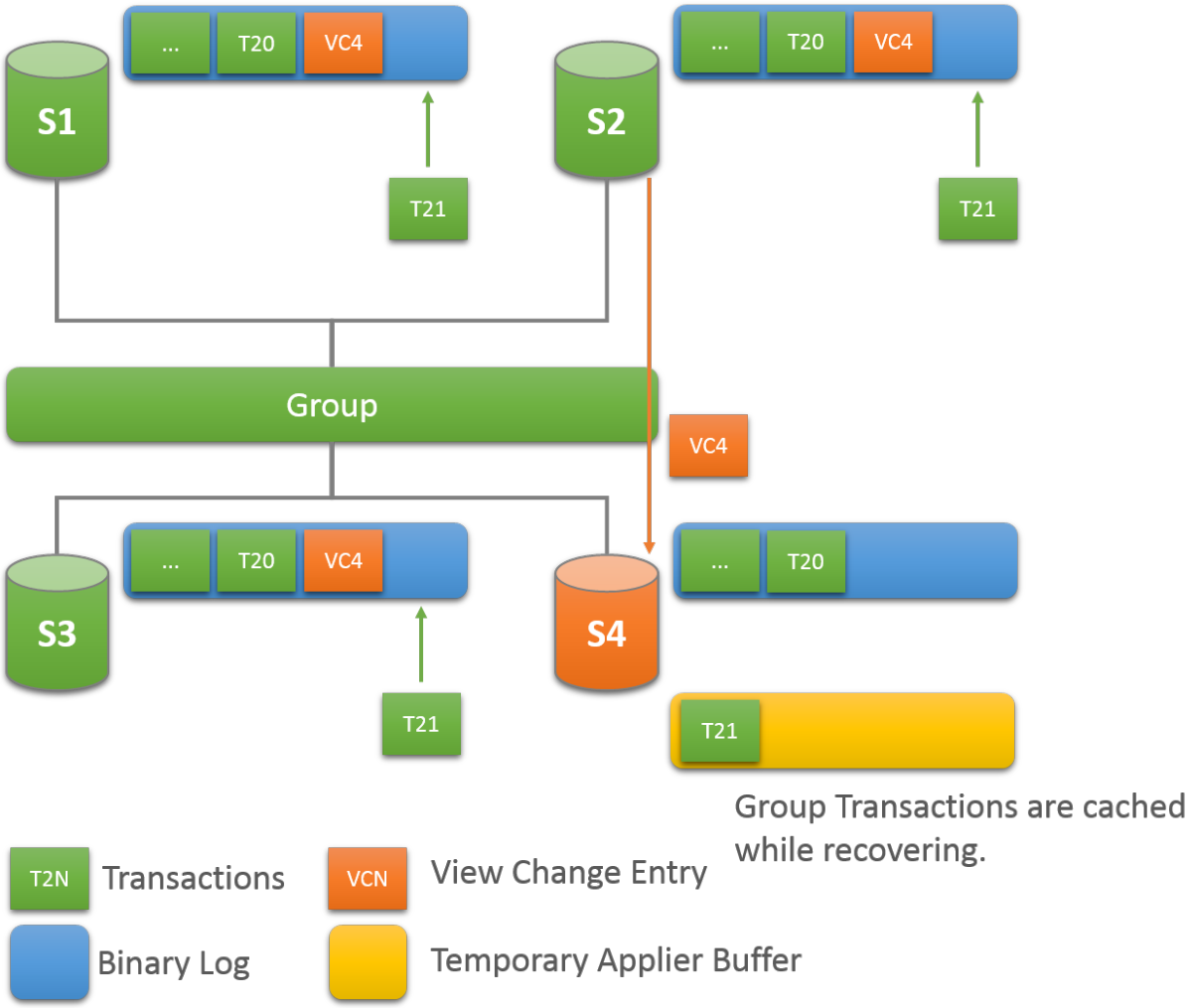
Figure 18.10 State Transfer: Catching Up



As view identifiers are transmitted to all members in the group at the same logical time, the server joining the group knows at which view identifier it should stop replicating. This avoids complex GTID set calculations because the view identifier clearly marks which data belongs to each group view.

While the server joining the group is replicating from the donor, it is also caching incoming transactions from the group. Eventually, it stops replicating from the donor and switches to applying those that are cached.

Figure 18.11 Queued Transactions



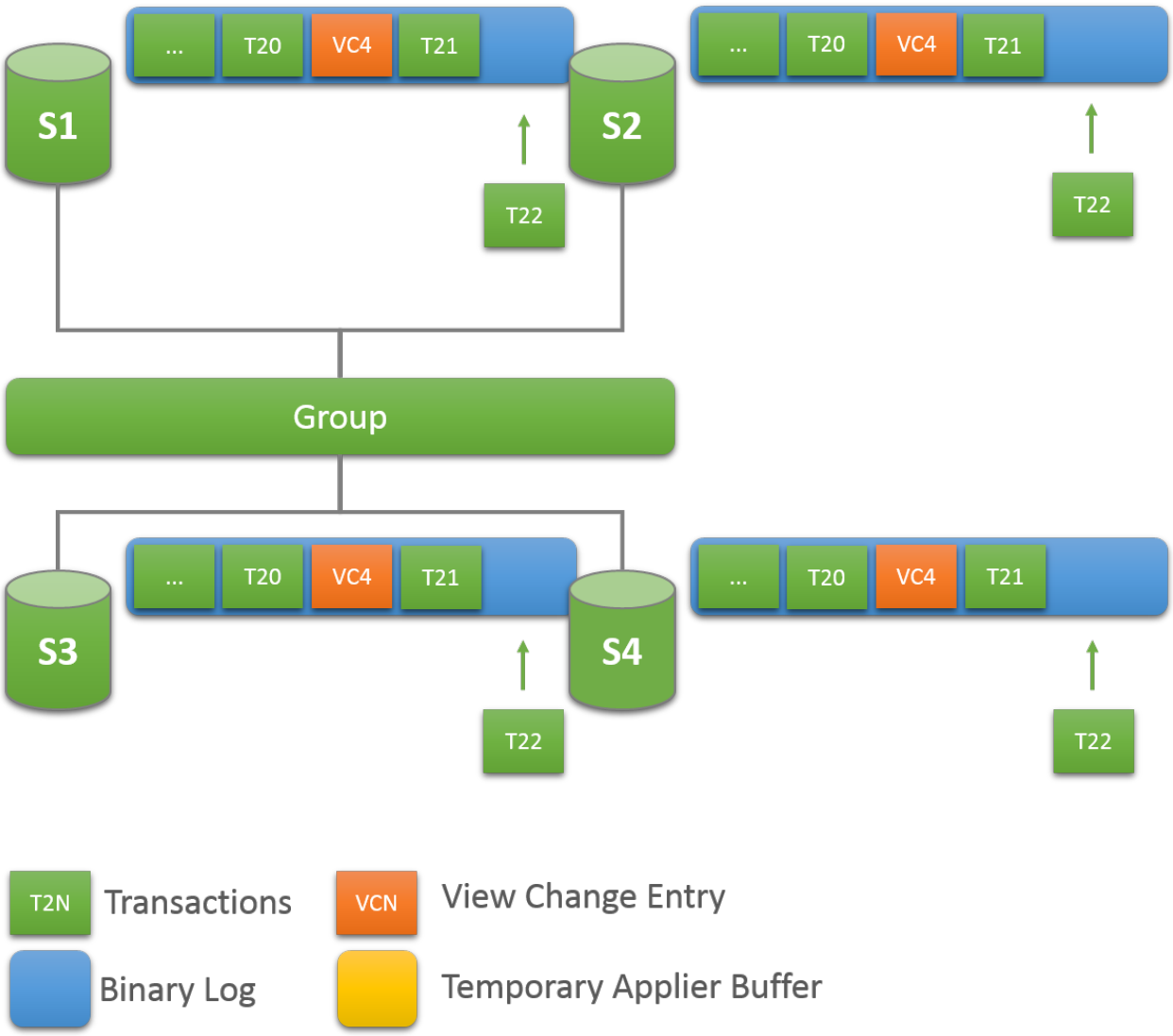
Finish: Caught Up

When the server joining the group recognizes a view change log event with the expected view identifier, the connection to the donor is terminated and it starts applying the cached transactions. Although it acts as a marker in the binary log, delimiting view changes, the view change log event also plays another role. It conveys the certification information as perceived by all servers when the server joining the group entered the group, in other words the last view change. Without it, the server joining the group would not have the necessary information to be able to certify (detect conflicts) subsequent transactions.

The duration of the catch up is not deterministic, because it depends on the workload and the rate of incoming transactions to the group. This process is completely online and the server joining the group does not block any other server in the group while it is catching up. Therefore the number of transactions the server joining the group is behind when it moves to this stage can, for this reason, vary and thus increase or decrease according to the workload.

When the server joining the group reaches zero queued transactions and its stored data is equal to the other members, its public state changes to online.

Figure 18.12 Instance Online



18.4.4 Network Partitioning

The group needs to achieve consensus whenever a change that needs to be replicated happens. This is the case for regular transactions but is also required for group membership changes and some internal messaging that keeps the group consistent. Consensus requires a majority of group members to agree on a given decision. When a majority of group members is lost, the group is unable to progress and blocks because it cannot secure majority or quorum.

Quorum may be lost when there are multiple involuntary failures, causing a majority of servers to be removed abruptly from the group. For example in a group of 5 servers, if 3 of them become silent at once, the majority is compromised and thus no quorum can be achieved. In fact, the remaining two are not able to tell if the other 3 servers have crashed or whether a network partition has isolated these 2 alone and therefore the group cannot be reconfigured automatically.

On the other hand, if servers exit the group voluntarily, they instruct the group that it should reconfigure itself. In practice, this means that a server that is leaving tells others that it is going away. This means that other members can reconfigure the group properly, the consistency of the membership is maintained and

the majority is recalculated. For example, in the above scenario of 5 servers where 3 leave at once, if the 3 leaving servers warn the group that they are leaving, one by one, then the membership is able to adjust itself from 5 to 2, and at the same time, securing quorum while that happens.

Note

Loss of quorum is by itself a side-effect of bad planning. Plan the group size for the number of expected failures (regardless whether they are consecutive, happen all at once or are sporadic).

The following sections explain what to do if the system partitions in such a way that no quorum is automatically achieved by the servers in the group.

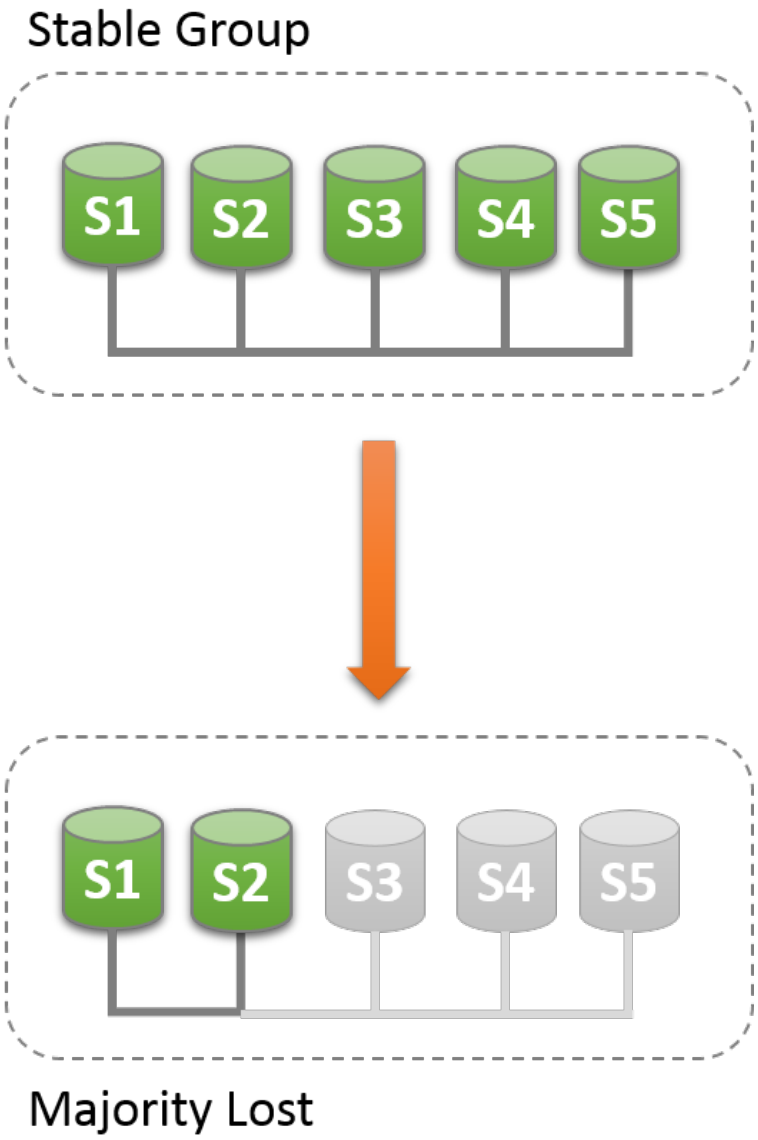
Tip

A primary that has been excluded from a group after a majority loss followed by a reconfiguration can contain extra transactions that are not included in the new group. If this happens, the attempt to add back the excluded member from the group results in an error with the message `This member has more executed transactions than those present in the group.`

Detecting Partitions

The `replication_group_members` performance schema table presents the status of each server in the current view from the perspective of this server. The majority of the time the system does not run into partitioning, and therefore the table shows information that is consistent across all servers in the group. In other words, the status of each server on this table is agreed by all in the current view. However, if there is network partitioning, and quorum is lost, then the table shows the status `UNREACHABLE` for those servers that it cannot contact. This information is exported by the local failure detector built into Group Replication.

Figure 18.13 Losing Quorum



To understand this type of network partition the following section describes a scenario where there are initially 5 servers working together correctly, and the changes that then happen to the group once only 2 servers are online. The scenario is depicted in the figure.

As such, lets assume that there is a group with these 5 servers in it:

- Server s1 with member identifier `199b2df7-4aaf-11e6-bb16-28b2bd168d07`
- Server s2 with member identifier `199bb88e-4aaf-11e6-babe-28b2bd168d07`
- Server s3 with member identifier `1999b9fb-4aaf-11e6-bb54-28b2bd168d07`
- Server s4 with member identifier `19ab72fc-4aaf-11e6-bb51-28b2bd168d07`
- Server s5 with member identifier `19b33846-4aaf-11e6-ba81-28b2bd168d07`

Initially the group is running fine and the servers are happily communicating with each other. You can verify this by logging into s1 and looking at its `replication_group_members` performance schema table. For example:

```
mysql> SELECT MEMBER_ID,MEMBER_STATE, MEMBER_ROLE FROM performance_schema.replication_group_members;
+--------------------------------------+--------------+-------------+
| MEMBER_ID                            | MEMBER_STATE |-MEMBER_ROLE |
+--------------------------------------+--------------+-------------+
| 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 | ONLINE       | SECONDARY   |
| 199b2df7-4aaf-11e6-bb16-28b2bd168d07 | ONLINE       | PRIMARY     |
| 199bb88e-4aaf-11e6-babe-28b2bd168d07 | ONLINE       | SECONDARY   |
| 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 | ONLINE       | SECONDARY   |
| 19b33846-4aaf-11e6-ba81-28b2bd168d07 | ONLINE       | SECONDARY   |
+--------------------------------------+--------------+-------------+
```

However, moments later there is a catastrophic failure and servers s3, s4 and s5 stop unexpectedly. A few seconds after this, looking again at the `replication_group_members` table on s1 shows that it is still online, but several others members are not. In fact, as seen below they are marked as `UNREACHABLE`. Moreover, the system could not reconfigure itself to change the membership, because the majority has been lost.

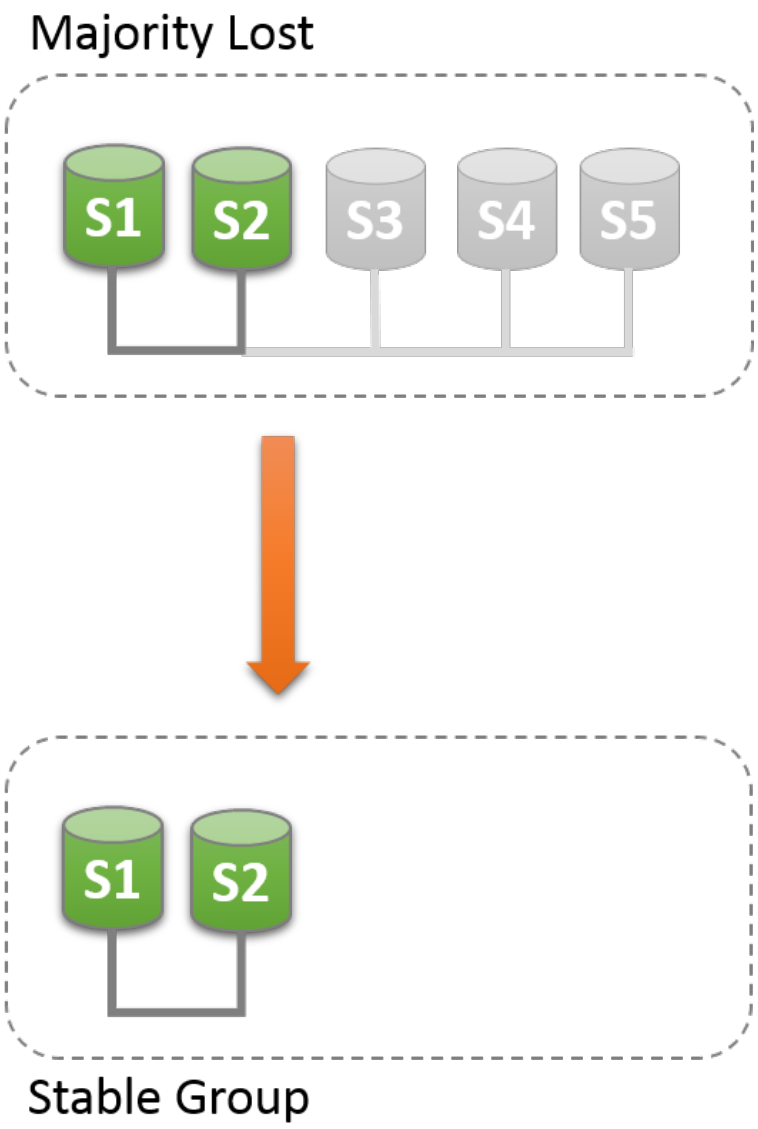
```
mysql> SELECT MEMBER_ID,MEMBER_STATE FROM performance_schema.replication_group_members;
+--------------------------------------+--------------+
| MEMBER_ID                            | MEMBER_STATE |
+--------------------------------------+--------------+
| 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 | UNREACHABLE  |
| 199b2df7-4aaf-11e6-bb16-28b2bd168d07 | ONLINE       |
| 199bb88e-4aaf-11e6-babe-28b2bd168d07 | ONLINE       |
| 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 | UNREACHABLE  |
| 19b33846-4aaf-11e6-ba81-28b2bd168d07 | UNREACHABLE  |
+--------------------------------------+--------------+
```

The table shows that s1 is now in a group that has no means of progressing without external intervention, because a majority of the servers are unreachable. In this particular case, the group membership list needs to be reset to allow the system to proceed, which is explained in this section. Alternatively, you could also choose to stop Group Replication on s1 and s2 (or stop completely s1 and s2), figure out what happened with s3, s4 and s5 and then restart Group Replication (or the servers).

Unblocking a Partition

Group replication enables you to reset the group membership list by forcing a specific configuration. For instance in the case above, where s1 and s2 are the only servers online, you could chose to force a membership configuration consisting of only s1 and s2. This requires checking some information about s1 and s2 and then using the `group_replication_force_members` variable.

Figure 18.14 Forcing a New Membership



Suppose that you are back in the situation where s1 and s2 are the only servers left in the group. Servers s3, s4 and s5 have left the group unexpectedly. To make servers s1 and s2 continue, you want to force a membership configuration that contains only s1 and s2.

Warning

This procedure uses `group_replication_force_members` and should be considered a last resort remedy. It *must* be used with extreme care and only for overriding loss of quorum. If misused, it could create an artificial split-brain scenario or block the entire system altogether.

Recall that the system is blocked and the current configuration is the following (as perceived by the local failure detector on s1):

```
mysql> SELECT MEMBER_ID,MEMBER_STATE FROM performance_schema.replication_group_members;
```

```
+--------------------------------------+--------------+
| MEMBER_ID                            | MEMBER_STATE |
+--------------------------------------+--------------+
| 1999b9fb-4aaf-11e6-bb54-28b2bd168d07 | UNREACHABLE  |
| 199b2df7-4aaf-11e6-bb16-28b2bd168d07 | ONLINE       |
| 199bb88e-4aaf-11e6-babe-28b2bd168d07 | ONLINE       |
| 19ab72fc-4aaf-11e6-bb51-28b2bd168d07 | UNREACHABLE  |
| 19b33846-4aaf-11e6-ba81-28b2bd168d07 | UNREACHABLE  |
+--------------------------------------+--------------+
```

The first thing to do is to check what is the local address (group communication identifier) for s1 and s2. Log in to s1 and s2 and get that information as follows.

```
mysql> SELECT @@group_replication_local_address;
```

Once you know the group communication addresses of s1 (`127.0.0.1:10000`) and s2 (`127.0.0.1:10001`), you can use that on one of the two servers to inject a new membership configuration, thus overriding the existing one that has lost quorum. To do that on s1:

```
mysql> SET GLOBAL group_replication_force_members="127.0.0.1:10000,127.0.0.1:10001";
```

This unblocks the group by forcing a different configuration. Check `replication_group_members` on both s1 and s2 to verify the group membership after this change. First on s1.

```
mysql> SELECT MEMBER_ID,MEMBER_STATE FROM performance_schema.replication_group_members;
+--------------------------------------+--------------+
| MEMBER_ID                            | MEMBER_STATE |
+--------------------------------------+--------------+
| b5ffe505-4ab6-11e6-b04b-28b2bd168d07 | ONLINE       |
| b60907e7-4ab6-11e6-afb7-28b2bd168d07 | ONLINE       |
+--------------------------------------+--------------+
```

And then on s2.

```
mysql> SELECT * FROM performance_schema.replication_group_members;
+--------------------------------------+--------------+
| MEMBER_ID                            | MEMBER_STATE |
+--------------------------------------+--------------+
| b5ffe505-4ab6-11e6-b04b-28b2bd168d07 | ONLINE       |
| b60907e7-4ab6-11e6-afb7-28b2bd168d07 | ONLINE       |
+--------------------------------------+--------------+
```

When forcing a new membership configuration, make sure that any servers are going to be forced out of the group are indeed stopped. In the scenario depicted above, if s3, s4 and s5 are not really unreachable but instead are online, they may have formed their own functional partition (they are 3 out of 5, hence they have the majority). In that case, forcing a group membership list with s1 and s2 could create an artificial split-brain situation. Therefore it is important before forcing a new membership configuration to ensure that the servers to be excluded are indeed shutdown and if they are not, shut them down before proceeding.

After you have used the `group_replication_force_members` system variable to successfully force a new group membership and unblock the group, ensure that you clear the system variable. `group_replication_force_members` must be empty in order to issue a `START GROUP_REPLICATION` statement.

18.4.5 Support For IPv6 And For Mixed IPv6 And IPv4 Groups

From MySQL 8.0.14, Group Replication group members can use IPv6 addresses as an alternative to IPv4 addresses for communications within the group. To use IPv6 addresses, the operating system on the

server host and the MySQL Server instance must both be configured to support IPv6. For instructions to set up IPv6 support for a server instance, see Section 5.1.12, “IPv6 Support”.

IPv6 addresses, or host names that resolve to them, can be specified as the network address that the member provides in the `group_replication_local_address` option for connections from other members. When specified with a port number, an IPv6 address must be specified in square brackets, for example:

```
group_replication_local_address= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

The network address or host name specified in `group_replication_local_address` is used by Group Replication as the unique identifier for a group member within the replication group. If a host name specified as the Group Replication local address for a server instance resolves to both an IPv4 and an IPv6 address, the IPv4 address is always used for Group Replication connections. The address or host name specified as the Group Replication local address is not the same as the MySQL server SQL protocol host and port, and is not specified in the `bind_address` system variable for the server instance. For the purpose of IP address whitelisting for Group Replication (see Section 18.5.1, “Group Replication IP Address Whitelisting”), the address that you specify for each group member in `group_replication_local_address` must be added to the list for the `group_replication_ip_whitelist` option on the other servers in the replication group.

A replication group can contain a combination of members that present an IPv6 address as their Group Replication local address, and members that present an IPv4 address. When a server joins such a mixed group, it must make the initial contact with the seed member using the protocol that the seed member advertises in the `group_replication_group_seeds` option, whether that is IPv4 or IPv6. If any of the seed members for the group are listed in the `group_replication_group_seeds` option with an IPv6 address when a joining member has an IPv4 Group Replication local address, or the reverse, you must also set up and whitelist an alternative address for the joining member for the required protocol (or a host name that resolves to an address for that protocol). If a joining member does not have a whitelisted address for the appropriate protocol, its connection attempt is refused. The alternative address or host name only needs to be added to the `group_replication_ip_whitelist` option on the other servers in the replication group, not to the `group_replication_local_address` value for the joining member (which can only contain a single address).

For example, server A is a seed member for a group, and has the following configuration settings for Group Replication, so that it is advertising an IPv6 address in the `group_replication_group_seeds` option:

```
group_replication_bootstrap_group=on
group_replication_local_address= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
group_replication_group_seeds= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

Server B is a joining member for the group, and has the following configuration settings for Group Replication, so that it has an IPv4 Group Replication local address:

```
group_replication_bootstrap_group=off
group_replication_local_address= "203.0.113.21:33061"
group_replication_group_seeds= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

Server B also has an alternative IPv6 address `2001:db8:8b0:40:3d9c:cc43:e006:19e8`. For Server B to join the group successfully, both its IPv4 Group Replication local address, and its alternative IPv6 address, must be listed in Server A's whitelist, as in the following example:

```
group_replication_ip_whitelist=
"203.0.113.0/24,2001:db8:85a3:8d3:1319:8a2e:370:7348,
2001:db8:8b0:40:3d9c:cc43:e006:19e8"
```

As a best practice for Group Replication IP whitelisting, Server B (and all other group members) should have the same whitelist as Server A, unless security requirements demand otherwise.

If any or all members of a replication group are using an older MySQL Server version that does not support the use of IPv6 addresses for Group Replication, a member cannot participate in the group using an IPv6 address (or a host name that resolves to one) as its Group Replication local address. This applies both in the case where at least one existing member uses an IPv6 address and a new member that does not support this attempts to join, and in the case where a new member attempts to join using an IPv6 address but the group includes at least one member that does not support this. In each situation, the new member cannot join. To make a joining member present an IPv4 address for group communications, you can either change the value of `group_replication_local_address` to an IPv4 address, or configure your DNS to resolve the joining member's existing host name to an IPv4 address. After you have upgraded every group member to a MySQL Server version that supports IPv6 for Group Replication, you can change the `group_replication_local_address` value for each member to an IPv6 address, or configure your DNS to present an IPv6 address. Changing the value of `group_replication_local_address` takes effect only when you stop and restart Group Replication.

18.4.6 Using MySQL Enterprise Backup with Group Replication

MySQL Enterprise Backup is a commercially-licensed backup utility for MySQL Server, available with MySQL Enterprise Edition. This section explains how to back up and subsequently restore a Group Replication member using MySQL Enterprise Backup. The same technique can be used to quickly add a new member to a group.

Backing up a Group Replication Member Using MySQL Enterprise Backup

Backing up a Group Replication member is similar to backing up a stand-alone MySQL instance. The following instructions assume that you are already familiar with how to use MySQL Enterprise Backup to perform a backup; if that is not the case, please review the MySQL Enterprise Backup 8.0 User's Guide, especially Backing Up a Database Server. Also note the requirements described in Grant MySQL Privileges to Backup Administrator and Using MySQL Enterprise Backup with Group Replication.

Consider the following group with three members, `s1`, `s2`, and `s3`, running on hosts with the same names:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-------------+-------------+--------------+
| member_host | member_port | member_state |
+-------------+-------------+--------------+
| s1          |        3306 | ONLINE       |
| s2          |        3306 | ONLINE       |
| s3          |        3306 | ONLINE       |
+-------------+-------------+--------------+
```

Using MySQL Enterprise Backup, create a backup of `s2` by issuing on its host, for example, the following command:

```
s2> mysqlbackup --defaults-file=/etc/my.cnf --backup-image=/backups/my.mbi_`date +%d%m_%H%M` \
        --backup-dir=/backups/backup_`date +%d%m_%H%M` --user=root -p \
        --host=127.0.0.1 backup-to-image
```

Notes

- If the system variable `sql_require_primary_key` is set to `ON` for the group, MySQL Enterprise Backup will not be able to log the backup progress on the servers. This is because the `backup_progress` table on the server is a CSV table, for which primary keys are not supported. In that case, `mysqlbackup` issues the following warnings during the backup operation:

  ```
  181011 11:17:06 MAIN WARNING: MySQL query 'CREATE TABLE IF NOT EXISTS
  ```

```
mysql.backup_progress( `backup_id` BIGINT NOT NULL, `tool_name` VARCHAR(4096)
NOT NULL, `error_code` INT NOT NULL, `error_message` VARCHAR(4096) NOT NULL,
`current_time` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP              ON
UPDATE CURRENT_TIMESTAMP,`current_state` VARCHAR(200) NOT NULL ) ENGINE=CSV
DEFAULT CHARSET=utf8 COLLATE=utf8_bin': 3750, Unable to create a table
without PK, when system variable 'sql_require_primary_key' is set. Add a PK
to the table or unset this variable to avoid this message. Note that tables
without PK can cause performance problems in row-based replication, so please
consult your DBA before changing this setting.
181011 11:17:06 MAIN WARNING: This backup operation's progress info cannot be
logged.
```

This does not prevent `mysqlbackup` from finishing the backup though.

- *For MySQL Enterprise Backup 8.0.11*, when backing up a secondary member, as MySQL Enterprise Backup cannot write backup status and metadata to a read-only server instance, it issues the following warnings during the backup operation:

```
181113 21:31:08 MAIN WARNING: This backup operation cannot write to backup
progress. The MySQL server is running with the --super-read-only option.
```

You can avoid the warning by using the `--no-history-logging` option with your backup command. This is not an issue for MySQL Enterprise Backup 8.0.12 and higher—see Using MySQL Enterprise Backup with Group Replication for details.

Restoring a Failed Member

Assume one of the members (`s3` in the following example) is irreconcilably corrupted. The most recent backup of group member `s2` can be used to restore `s3`. Here are the steps for performing the restore:

1. *Copy the backup of s2 onto the host for s3.* The exact way to copy the backup depends on the operating system and tools available to you. In this example, we assume the hosts are both Linux servers and use SCP to copy the files between them:

```
s2/backups> scp my.mbi_2206_1429 s3:/backups
```

2. *Restore the backup.* Connect to the target host (the host for `s3` in this case), and restore the backup using MySQL Enterprise Backup. Here are the steps:

   a. Stop the corrupted server, if it is still running. For example, on Linux distributions that use systemd:

```
s3> systemctl stop mysqld
```

   b. Preserve the two configuration files in the corrupted server's data directory, `auto.cnf` and `mysqld-auto.cnf` (if it exists), by copying them to a safe location outside of the data directory. This is for preserving the server's UUID and Section 5.1.9.3, “Persisted System Variables” (if used), which are needed in the steps below.

   c. Delete all contents in the data directory of `s3`. For example:

```
s3> rm -rf /var/lib/mysql/*
```

   If the system variables `innodb_data_home_dir`, `innodb_log_group_home_dir`, and `innodb_undo_directory` point to any directories other than the data directory, they should also be made empty; otherwise, the restore operation fails.

   d. Restore backup of `s2` onto the host for `s3`:

```
s3> mysqlbackup --defaults-file=/etc/my.cnf \
```

```
  --datadir=/var/lib/mysql \
  --backup-image=/backups/my.mbi_2206_1429  \
  --backup-dir=/tmp/restore_`date +%d%m_%H%M` copy-back-and-apply-log
```

Note

The command above assumes that the binary logs and relay logs on `s2` and `s3` have the same base name and are at the same location on the two servers. If these conditions are not met, you should use the `--log-bin` and `--relay-log` options to restore the binary log and relay log to their original file paths on `s3`. For example, if you know that on `s3` the binary log's base name is `s3-bin` and the relay-log's base name is `s3-relay-bin`, your restore command should look like:

```
mysqlbackup --defaults-file=/etc/my.cnf \
  --datadir=/var/lib/mysql \
  --backup-image=/backups/my.mbi_2206_1429  \
  --log-bin=s3-bin --relay-log=s3-relay-bin \
  --backup-dir=/tmp/restore_`date +%d%m_%H%M` copy-back-and-apply-log
```

Being able to restore the binary log and relay log to the right file paths makes the restore process easier; if that is impossible for some reason, see Rebuild the Failed Member to Rejoin as a New Member.

3. *Restore the `auto.cnf` file for s3.* To rejoin the replication group, the restored member *must* have the same `server_uuid` it used to join the group before. Supply the old server UUID by copying the `auto.cnf` file preserved in step 2 above into the data directory of the restored member.

Note

If you cannot supply the failed member's original `server_uuid` to the restored member by restoring its old `auto.cnf` file, you will have to let the restored member join the group as a new member; see instructions in Rebuild the Failed Member to Rejoin as a New Member below on how to do that.

4. *Restore the `mysqld-auto.cnf` file for s3 (only required if s3 used persistent system variables).* The settings for the Section 5.1.9.3, “Persisted System Variables” that were used to configure the failed member must be provided to the restored member. These settings are to be found in the `mysqld-auto.cnf` file of the failed server, which you should have preserved in step 2 above. Restore the file to the data directory of the restored server. See Restoring Persisted System Variables on what to do if you do not have a copy of the file.

5. *Start the restored server.* For example, on Linux distributions that use systemd:

```
systemctl start mysqld
```

Note

If the server you are restoring is a primary member, perform the steps described in Restoring a Primary Member *before starting the restored server*.

6. *Restart Group Replication.* Connect to the restarted `s3` using, for example, a `mysql` client, and issue the following command:

```
mysql> START GROUP_REPLICATION;
```

Before the restored instance can become an online member of the group, it needs to apply any transactions that have happened to the group after the backup was taken; this is achieved using Group Replication's distributed recovery mechanism, and the process starts after the START

GROUP_REPLICATION statement has been issued. To check the member status of the restored instance, issue:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-------------+-------------+--------------+
| member_host | member_port | member_state |
+-------------+-------------+--------------+
| s1          |        3306 | ONLINE       |
| s2          |        3306 | ONLINE       |
| s3          |        3306 | RECOVERING   |
+-------------+-------------+--------------+
```

This shows that `s3` is applying transactions to catch up with the group. Once it has caught up with the rest of the group, its `member_state` changes to `ONLINE`:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-------------+-------------+--------------+
| member_host | member_port | member_state |
+-------------+-------------+--------------+
| s1          |        3306 | ONLINE       |
| s2          |        3306 | ONLINE       |
| s3          |        3306 | ONLINE       |
+-------------+-------------+--------------+
```

Note

If the server you are restoring is a primary member, once it has gained synchrony with the group and become `ONLINE`, perform the steps described at the end of Restoring a Primary Member to revert the configuration changes you had made to the server before you started it.

The member has now been fully restored from the backup and functions as a regular member of the group.

Rebuild the Failed Member to Rejoin as a New Member

Sometimes, the steps outlined above in Restoring a Failed Member cannot be carried out because, for example, the binary log or relay log is corrupted, or it is just missing from the backup. In such a situation, use the backup to rebuild the member, and then add it to the group as a new member. In the steps below, we assume the rebuilt member will be named `s3`, like the failed member, and it will be run on the same host as `s3` was:

1. *Copy the backup of s2 onto the host for s3 .* The exact way to copy the backup depends on the operating system and tools available to you. In this example we assume the hosts are both Linux servers and use SCP to copy the files between them:

   ```
   s2/backups> scp my.mbi_2206_1429 s3:/backups
   ```

2. *Restore the backup.* Connect to the target host (the host for `s3` in this case), and restore the backup using MySQL Enterprise Backup. Here are the steps:

   a. Stop the corrupted server, if it is still running. For example, on Linux distributions that use systemd:

      ```
      s3> systemctl stop mysqld
      ```

   b. Preserve the configuration file `mysqld-auto.cnf`, if it is found in the corrupted server's data directory, by copying it to a safe location outside of the data directory. This is for preserving the server's Section 5.1.9.3, “Persisted System Variables”, which are needed later.

   c. Delete all contents in the data directory of `s3`. For example:

```
s3> rm -rf /var/lib/mysql/*
```

If the system variables `innodb_data_home_dir`, `innodb_log_group_home_dir`, and `innodb_undo_directory` point to any directories other than the data directory, they should also be made empty; otherwise, the restore operation will fail.

d. Restore the backup of `s2` onto the host of `s3`. With this approach, we are rebuilding `s3` as a new member, for which we do not need or do not want to use the old binary and relay logs in the backup; therefore, if these logs have been included in your backup, exclude them using the `--skip-binlog` and `--skip-relaylog` options:

```
s3> mysqlbackup --defaults-file=/etc/my.cnf \
  --datadir=/var/lib/mysql \
  --backup-image=/backups/my.mbi_2206_1429  \
  --backup-dir=/tmp/restore_`date +%d%m_%H%M` \
  --skip-binlog --skip-relaylog \
  copy-back-and-apply-log
```

Note

If you have healthy binary log and relay logs in the backup that you can transfer onto the target host with no issues, you are recommended to follow the easier procedure as described in Restoring a Failed Member above.

3. *Restore the `mysqld-auto.cnf` file for s3 (only required if s3 used persistent system variables).* The settings for the Section 5.1.9.3, “Persisted System Variables” that were used to configure the failed member must be provided to the restored server. These settings are to be found in the `mysqld-auto.cnf` file of the failed server, which you should have preserved in step 2 above. Restore the file to the data directory of the restored server. See Restoring Persisted System Variables on what to do if you do not have a copy of the file.

Note

Do NOT restore the corrupted server's `auto.cnf` file to the data directory of the new member—when the rebuilt `s3` joins the group as a new member, it is going to be assigned a new server UUID.

4. *Start the restored server.* For example, on Linux distributions that use systemd:

```
systemctl start mysqld
```

Note

If the server you are restoring is a primary member, perform the steps described in Restoring a Primary Member *before starting the restored server*.

5. *Reconfigure the restored member to join Group Replication.* Connect to the restored server with a `mysql` client and reset the master and slave information with the following commands:

```
mysql> RESET MASTER;
```

```
mysql> RESET SLAVE ALL;
```

For the restored server to be able to recover automatically using Group Replication's built-in mechanism for distributed recovery, configure the server's `gtid_executed` variable. To do this, use the `backup_gtid_executed.sql` file included in the backup of `s2`, which is usually restored under the restored member's data directory. Disable binary logging, use the `backup_gtid_executed.sql`

file to configure `gtid_executed`, and then re-enable binary logging by issuing the following statements with your `mysql` client:

```
mysql> SET SQL_LOG_BIN=OFF;
mysql> SOURCE datadir/backup_gtid_executed.sql
mysql> SET SQL_LOG_BIN=ON;
```

Then, configure the Group Replication user credentials on the member:

```
mysql> CHANGE MASTER TO MASTER_USER='rpl_user', MASTER_PASSWORD='password' /
  FOR CHANNEL 'group_replication_recovery';
```

6. *Restart Group Replication.* Issue the following command to the restored server with your `mysql` client:

```
mysql> START GROUP_REPLICATION;
```

Before the restored instance can become an online member of the group, it needs to apply any transactions that have happened to the group after the backup was taken; this is achieved using Group Replication's distributed recovery mechanism, and the process starts after the START GROUP_REPLICATION statement has been issued. To check the member status of the restored instance, issue:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-------------+-------------+--------------+
| member_host | member_port | member_state |
+-------------+-------------+--------------+
| s3          |        3306 | RECOVERING   |
| s2          |        3306 | ONLINE       |
| s1          |        3306 | ONLINE       |
+-------------+-------------+--------------+
```

This shows that `s3` is applying transactions to catch up with the group. Once it has caught up with the rest of the group, its `member_state` changes to `ONLINE`:

```
mysql> SELECT member_host, member_port, member_state FROM performance_schema.replication_group_members;
+-------------+-------------+--------------+
| member_host | member_port | member_state |
+-------------+-------------+--------------+
| s3          |        3306 | ONLINE       |
| s2          |        3306 | ONLINE       |
| s1          |        3306 | ONLINE       |
+-------------+-------------+--------------+
```

Note

If the server you are restoring is a primary member, once it has gained synchrony with the group and become `ONLINE`, perform the steps described at the end of Restoring a Primary Member to revert the configuration changes you had made to the server before you started it.

The member has now been restored to the group as a new member.

Restoring Persisted System Variables. `mysqlbackup` does not provide support for backing up or preserving Section 5.1.9.3, “Persisted System Variables”—the file `mysqld-auto.cnf` is not included in a backup. To start the restored member with its persisted variable settings, you need to do one of the following:

- Preserve a copy of the `mysqld-auto.cnf` file from the corrupted server, and copy it to the restored server's data directory.
- Copy the `mysqld-auto.cnf` file from another member of the group into the restored server's data directory, if that member has the same persisted system variable settings as the corrupted member.
- After the restored server is started and before you restart Group Replication, set all the system variables manually to their persisted values through a `mysql` client.

Restoring a Primary Member. If the restored member is a primary in the group, care must be taken to prevent writes to the restored database during the Group Replication distributed recovery process. Depending on how the group is accessed by clients, there is a possibility of DML statements being executed on the restored member once it becomes accessible on the network, prior to the member finishing its catch-up on the activities it has missed while off the group. To avoid this, *before starting the restored server*, configure the following system variables in the server option file:

```
group_replication_start_on_boot=OFF
super_read_only=ON
event_scheduler=OFF
```

These settings ensure that the member becomes read-only at startup and that the event scheduler is turned off while the member is catching up with the group during the distributed recovery process. Adequate error handling must also be configured on the clients, as they will be prevented temporarily from performing DML operations during this period on the restored member. Once the restore process is fully completed and the restored member is in-sync with the rest of the group, revert those changes; restart the event scheduler:

```
mysql> SET global event_scheduler=ON;
```

Edit the following system variables in the member's option file, so things are correctly configured for the next startup:

```
group_replication_start_on_boot=ON
super_read_only=OFF
event_scheduler=ON
```

18.5 Group Replication Security

This section explains how to secure a group, securing the connections between members of a group, or by establishing a security perimeter using IP address whitelisting.

18.5.1 Group Replication IP Address Whitelisting

The Group Replication plugin has a configuration option to determine from which hosts an incoming Group Communication System connection can be accepted. This option is called `group_replication_ip_whitelist`. If you set this option on a server s1, then when server s2 is establishing a connection to s1 for the purpose of engaging group communication, s1 first checks the whitelist before accepting the connection from s2. If s2 is in the whitelist, then s1 accepts the connection, otherwise s1 rejects the connection attempt by s2.

If you do not specify a whitelist explicitly, the group communication engine (XCom) automatically scans active interfaces on the host, and identifies those with addresses on private subnetworks. These addresses and the `localhost` IP address for IPv4 and (from MySQL 8.0.14) IPv6 are used to create an automatic Group Replication whitelist. The automatic whitelist therefore includes any IP addresses found for the host in the following ranges:

```
IPv4 (as defined in RFC 1918)
10/8 prefix       (10.0.0.0 - 10.255.255.255) - Class A
172.16/12 prefix  (172.16.0.0 - 172.31.255.255) - Class B
192.168/16 prefix (192.168.0.0 - 192.168.255.255) - Class C

IPv6 (as defined in RFC 4193 and RFC 5156)
fc00:/7 prefix    - unique-local addresses
fe80::/10 prefix  - link-local unicast addresses

127.0.0.1 - localhost for IPv4
::1       - localhost for IPv6
```

An entry is added to the error log stating the addresses that have been whitelisted automatically for the host.

The automatic whitelist of private addresses cannot be used for connections from servers outside the private network, so a server, even if it has interfaces on public IPs, does not by default allow Group Replication connections from external hosts. For Group Replication connections between server instances that are on different machines, you must provide public IP addresses and specify these as an explicit whitelist. If you specify any entries for the whitelist, the private and `localhost` addresses are not added automatically, so if you use any of these, you must specify them explicitly.

To specify a whitelist manually, use the `group_replication_ip_whitelist` option. You cannot change the whitelist on a server while it is an active member of a replication group. If the member is active, you must issue a `STOP GROUP_REPLICATION` statement before changing the whitelist, and a `START GROUP_REPLICATION` statement afterwards.

In the whitelist, you can specify any combination of the following:

- IPv4 addresses (for example, `198.51.100.44`)
- IPv4 addresses with CIDR notation (for example, `192.0.2.21/24`)
- IPv6 addresses, from MySQL 8.0.14 (for example, `2001:db8:85a3:8d3:1319:8a2e:370:7348`)
- IPv6 addresses with CIDR notation, from MySQL 8.0.14 (for example, `2001:db8:85a3:8d3::/64`)
- Host names (for example, `example.org`)
- Host names with CIDR notation (for example, `www.example.com/24`)

Before MySQL 8.0.14, host names could only resolve to IPv4 addresses. From MySQL 8.0.14, host names can resolve to IPv4 addresses, IPv6 addresses, or both. If a host name resolves to both an IPv4 and an IPv6 address, the IPv4 address is always used for Group Replication connections. You can use CIDR notation in combination with host names or IP addresses to whitelist a block of IP addresses with a particular network prefix, but do ensure that all the IP addresses in the specified subnet are under your control.

You must stop and restart Group Replication on a member in order to change its whitelist. A comma must separate each entry in the whitelist. For example:

```
mysql> STOP GROUP_REPLICATION;
mysql> SET GLOBAL group_replication_ip_whitelist="192.0.2.21/24,198.51.100.44,203.0.113.0/24,2001:db8:85a3
mysql> START GROUP_REPLICATION;
```

The whitelist must contain the IP address or host name that is specified in each member's `group_replication_local_address` system variable. This address is not the same as the MySQL server SQL protocol host and port, and is not specified in the `bind_address` system variable for the

server instance. If a host name used as the Group Replication local address for a server instance resolves to both an IPv4 and an IPv6 address, the IPv4 address is preferred for Group Replication connections.

To join a replication group, a server needs to be whitelisted on the seed member to which it makes the request to join the group. Typically, this would be the bootstrap member for the replication group, but it can be any of the servers listed by the `group_replication_group_seeds` option in the configuration for the server joining the group. If any of the seed members for the group are listed in the `group_replication_group_seeds` option with an IPv6 address when a joining member has an IPv4 `group_replication_local_address`, or the reverse, you must also set up and whitelist an alternative address for the joining member for the protocol offered by the seed member (or a host name that resolves to an address for that protocol). This is because when a server joins a replication group, it must make the initial contact with the seed member using the protocol that the seed member advertises in the `group_replication_group_seeds` option, whether that is IPv4 or IPv6. If a joining member does not have a whitelisted address for the appropriate protocol, its connection attempt is refused. For more information on managing mixed IPv4 and IPv6 replication groups, see Section 18.4.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”.

When a replication group is reconfigured (for example, when a new primary is elected or a member joins or leaves), the group members re-establish connections between themselves. If a group member is only whitelisted by servers that are no longer part of the replication group after the reconfiguration, it is unable to reconnect to the remaining servers in the replication group that do not whitelist it. To avoid this scenario entirely, specify the same whitelist for all servers that are members of the replication group.

Note

It is possible to configure different whitelists on different group members according to your security requirements, for example, in order to keep different subnets separate. If you need to configure different whitelists to meet your security requirements, ensure that there is sufficient overlap between the whitelists in the replication group to maximize the possibility of servers being able to reconnect in the absence of their original seed member.

For host names, name resolution takes place only when a connection request is made by another server. A host name that cannot be resolved is not considered for whitelist validation, and a warning message is written to the error log. Forward-confirmed reverse DNS (FCrDNS) verification is carried out for resolved host names.

Warning

Host names are inherently less secure than IP addresses in a whitelist. FCrDNS verification provides a good level of protection, but can be compromised by certain types of attack. Specify host names in your whitelist only when strictly necessary, and ensure that all components used for name resolution, such as DNS servers, are maintained under your control. You can also implement name resolution locally using the hosts file, to avoid the use of external components.

18.5.2 Group Replication Secure Socket Layer (SSL) Support

Group communication connections and distributed recovery connections can be secured using SSL. The following sections explain how to configure connections.

Configuring SSL for Group Communication

Secure sockets can be used for group communication connections between members of a group. The Group Replication system variable `group_replication_ssl_mode` is used to activate the use of SSL

for group communication connections and specify the security mode for the connections. The default setting means that SSL is not used. The option has the following possible values:

Table 18.2 group_replication_ssl_mode configuration values

Value	Description
DISABLED	Establish an unencrypted connection (the default).
REQUIRED	Establish a secure connection if the server supports secure connections.
VERIFY_CA	Like REQUIRED, but additionally verify the server TLS certificate against the configured Certificate Authority (CA) certificates.
VERIFY_IDENTITY	Like VERIFY_CA, but additionally verify that the server certificate matches the host to which the connection is attempted.

The remainder of the configuration for Group Replication's group communication connections is taken from the server's SSL configuration. For more information on the options for configuring the server SSL, see Command Options for Encrypted Connections. The server SSL options that are applied to Group Replication's group communication connections are as follows:

Table 18.3 SSL Options

Server Configuration	Description
`ssl_key`	The path name of the SSL private key file in PEM format. On the client side, this is the client private key. On the server side, this is the server private key.
`ssl_cert`	The path name of the SSL public key certificate file in PEM format. On the client side, this is the client public key certificate. On the server side, this is the server public key certificate.
`ssl_ca`	The path name of the Certificate Authority (CA) certificate file in PEM format.
`ssl_capath`	The path name of the directory that contains trusted SSL certificate authority (CA) certificate files in PEM format.
`ssl_crl`	The path name of the file containing certificate revocation lists in PEM format.
`ssl_crlpath`	The path name of the directory that contains certificate revocation list files in PEM format.
`ssl_cipher`	A list of permissible ciphers for encrypted connections.
`tls_version`	A list of the TLS protocols the server permits for encrypted connections.
`tls_ciphersuites`	Which TLSv1.3 ciphersuites the server permits for encrypted connections.

Important

- Support for the TLSv1.3 protocol is available in MySQL Server as of MySQL 8.0.16, provided that MySQL was compiled using OpenSSL 1.1.1 or higher. Group Replication supports TLSv1.3 from MySQL 8.0.18. In MySQL 8.0.16 and MySQL 8.0.17, if the server supports TLSv1.3, the protocol is not supported in the group communication engine and cannot be used by Group Replication.

- In the list of TLS protocols specified in the `tls_version` system variable, ensure the specified versions are contiguous (for example, `TLSv1,TLSv1.1,TLSv1.2`). If there are any gaps in the list of protocols (for example, if you specified `TLSv1,TLSv1.2`, omitting TLS 1.1) Group Replication might be unable to make group communication connections.
- If TLSv1.3 is used in Group Replication for distributed recovery, the Group Replication group members must permit the use of at least one TLSv1.3 ciphersuite that is enabled by default.

In a replication group, OpenSSL negotiates the use of the highest TLS protocol that is supported by all members. A joining member that is configured to use only TLSv1.3 (`tls_version=TLSv1.3`) cannot join a replication group where any existing member does not support TLSv1.3, because the group members in that case are using a lower TLS protocol version. To join the member to the group, you must configure the joining member to also permit the use of lower TLS protocol versions supported by the existing group members. Conversely, if a joining member does not support TLSv1.3, but the existing group members all do and are using that version, the member can join if the existing group members already permit the use of a suitable lower TLS protocol version, or if you configure them to do so. In that situation, OpenSSL renegotiates to a lower TLS protocol version to accept the joining member, so the group no longer uses TLSv1.3.

From MySQL 8.0.16, you can change the `tls_version` system variable at runtime to alter the list of permitted TLS protocol versions for the server. Note that for Group Replication, the `ALTER INSTANCE RELOAD TLS` statement, which reconfigures the server's SSL context from the current values of the system variables that define the context, does not change the SSL context for Group Replication's group communication connection while Group Replication is running. To apply the reconfiguration to these connections, you must issue `STOP GROUP_REPLICATION` followed by `START GROUP_REPLICATION` to restart Group Replication on the member or members where you changed the `tls_version` system variable. Similarly, if you want to make all members of a group change to using a higher or lower TLS protocol version, you must carry out a rolling restart of Group Replication on the members after changing the list of permitted TLS protocol versions, so that OpenSSL negotiates the use of the higher TLS protocol version when the rolling restart is completed. For instructions to change the list of permitted TLS protocol versions at runtime, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers” and Server-Side Runtime Configuration for Encrypted Connections.

The following example shows a section from a `my.cnf` file that configures SSL on a server, and activates SSL for Group Replication group communication connections:

```
[mysqld]
ssl_ca = "cacert.pem"
ssl_capath = "/.../ca_directory"
ssl_cert = "server-cert.pem"
ssl_cipher = "DHE-RSA-AEs256-SHA"
ssl_crl = "crl-server-revoked.crl"
ssl_crlpath = "/.../crl_directory"
ssl_key = "server-key.pem"
group_replication_ssl_mode= REQUIRED
```

Important

The `ALTER INSTANCE RELOAD TLS` statement, which reconfigures the server's SSL context from the current values of the system variables that define the context, does not change the SSL context for Group Replication's group communication connections while Group Replication is running. To apply the reconfiguration to these connections, you must issue `STOP GROUP_REPLICATION` followed by `START GROUP_REPLICATION` to restart Group Replication.

Configuring SSL for Distributed Recovery

When a member joins the group, distributed recovery is carried out using a combination of a remote cloning operation, if available and appropriate, and an asynchronous replication connection. Both methods of state transfer require a replication user that has been set up for distributed recovery, as described in Section 18.2.1.3, “User Credentials”. A replication user that requires an SSL connection must be created *before* the server joining the group (the joining member) connects to the donor. Typically, this is set up at the time you are provisioning a server to join the group.

To create a replication user for distributed recovery that requires an SSL connection, issue these statements:

```
donor> SET SQL_LOG_BIN=0;
donor> CREATE USER 'rec_ssl_user'@'%' REQUIRE SSL;
donor> GRANT replication slave ON *.* TO 'rec_ssl_user'@'%';
donor> GRANT BACKUP_ADMIN ON *.* TO 'rec_ssl_user'@'%';
donor> SET SQL_LOG_BIN=1;
```

Assuming that all servers already in the group have a replication user set up to use SSL, you configure joining members to use those credentials when connecting to the donor by configuring the Group Replication recovery channel to use the credentials, as follows:

```
new_member> CHANGE MASTER TO MASTER_USER="rec_ssl_user" FOR CHANNEL "group_replication_recovery";
```

To configure secure distributed recovery connections, use Group Replication's dedicated distributed recovery SSL options. These options correspond to the server SSL options that are used for group communication connections, but they are only applied for distributed recovery connections. By default, distributed recovery connections do not use SSL, even if you activated SSL for group communication connections, and the server SSL options are not applied for distributed recovery connections. You must configure these connections separately.

If a remote cloning operation is used as part of distributed recovery, Group Replication automatically configures the clone plugin's SSL options to match your settings for the distributed recovery SSL options. (For details of how the clone plugin uses SSL, see Configuring an Encrypted Connection for Cloning.)

The distributed recovery SSL options are as follows:

- `group_replication_recovery_use_ssl`: Set to `ON` to make Group Replication use SSL for distributed recovery connections, including remote cloning operations and state transfer from a donor's binary log.
- `group_replication_recovery_ssl_ca`: The path name of the Certificate Authority (CA) file to use for distributed recovery connections. Group Replication automatically configures the clone SSL option `clone_ssl_ca` to match this.

  `group_replication_recovery_ssl_capath`: The path name of a directory that contains trusted SSL certificate authority (CA) certificate files.
- `group_replication_recovery_ssl_cert`: The path name of the SSL public key certificate file to use for distributed recovery connections. Group Replication automatically configures the clone SSL option `clone_ssl_cert` to match this.
- `group_replication_recovery_ssl_key`: The path name of the SSL private key file to use for distributed recovery connections. Group Replication automatically configures the clone SSL option `clone_ssl_cert` to match this.

- `group_replication_recovery_ssl_verify_server_cert`: Makes the distributed recovery connection check the server's Common Name value in the donor sent certificate. Setting this option to `ON` is the equivalent for distributed recovery connections of setting `VERIFY_IDENTITY` for the `group_replication_ssl_mode` option for group communication connections.
- `group_replication_recovery_ssl_crl`: The path name of a file containing certificate revocation lists.
- `group_replication_recovery_ssl_crlpath`: The path name of a directory containing certificate revocation lists.
- `group_replication_recovery_ssl_cipher`: A list of permissible ciphers for connection encryption for the distributed recovery connection. Specify a list of one or more cipher names, separated by colons. For information about which encryption ciphers MySQL supports, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

For example, issuing the following statements enables the use of SSL for distributed recovery connections, and identifies the paths to the certificate authority (CA) file, the public key certificate file, and the private key file that must be used for those connections:

```
new_member> SET GLOBAL group_replication_recovery_use_ssl=1;
new_member> SET GLOBAL group_replication_recovery_ssl_ca= '.../cacert.pem';
new_member> SET GLOBAL group_replication_recovery_ssl_cert= '.../client-cert.pem';
new_member> SET GLOBAL group_replication_recovery_ssl_key= '.../client-key.pem';
```

18.6 Group Replication Performance

This section explains how to use the available configuration options to gain the best performance from your replication group.

18.6.1 Fine Tuning the Group Communication Thread

The group communication thread (GCT) runs in a loop while the Group Replication plugin is loaded. The GCT receives messages from the group and from the plugin, handles quorum and failure detection related tasks, sends out some keep alive messages and also handles the incoming and outgoing transactions from/to the server/group. The GCT waits for incoming messages in a queue. When there are no messages, the GCT waits. By configuring this wait to be a little longer (doing an active wait) before actually going to sleep can prove to be beneficial in some cases. This is because the alternative is for the operating system to switch out the GCT from the processor and do a context switch.

To force the GCT to do an active wait, use the `group_replication_poll_spin_loops` option, which makes the GCT loop, doing nothing relevant for the configured number of loops, before actually polling the queue for the next message.

For example:

```
mysql> SET GLOBAL group_replication_poll_spin_loops= 10000;
```

18.6.2 Flow Control

Group Replication ensures that a transaction only commits after a majority of the members in a group have received it and agreed on the relative order between all transactions that were sent concurrently. This approach works well if the total number of writes to the group does not exceed the write capacity of

any member in the group. If it does and some of the members have less write throughput than others, particularly less than the writer members, those members can start lagging behind of the writers.

Having some members lagging behind the group brings some problematic consequences, particularly, the reads on such members may externalize very old data. Depending on why the member is lagging behind, other members in the group may have to save more or less replication context to be able to fulfil potential data transfer requests from the slow member.

There is however a mechanism in the replication protocol to avoid having too much distance, in terms of transactions applied, between fast and slow members. This is known as the flow control mechanism. It tries to address several goals:

1. to keep the members close enough to make buffering and de-synchronization between members a small problem;
2. to adapt quickly to changing conditions like different workloads or more writers in the group;
3. to give each member a fair share of the available write capacity;
4. to not reduce throughput more than strictly necessary to avoid wasting resources.

Given the design of Group Replication, the decision whether to throttle or not may be decided taking into account two work queues: *(i)* the *certification* queue; *(ii)* and on the binary log *applier* queue. Whenever the size of one of these queues exceeds the user-defined threshold, the throttling mechanism is triggered. Only configure: *(i)* whether to do flow control at the certifier or at the applier level, or both; and *(ii)* what is the threshold for each queue.

The flow control depends on two basic mechanisms:

1. the monitoring of members to collect some statistics on throughput and queue sizes of all group members to make educated guesses on what is the maximum write pressure each member should be subjected to;
2. the throttling of members that are trying to write beyond their fair-share of the available capacity at each moment in time.

18.6.2.1 Probes and Statistics

The monitoring mechanism works by having each member deploying a set of probes to collect information about its work queues and throughput. It then propagates that information to the group periodically to share that data with the other members.

Such probes are scattered throughout the plugin stack and allow one to establish metrics, such as:

- the certifier queue size;
- the replication applier queue size;
- the total number of transactions certified;
- the total number of remote transactions applied in the member;
- the total number of local transactions.

Once a member receives a message with statistics from another member, it calculates additional metrics regarding how many transactions were certified, applied and locally executed in the last monitoring period.

Monitoring data is shared with others in the group periodically. The monitoring period must be high enough to allow the other members to decide on the current write requests, but low enough that it has minimal impact on group bandwidth. The information is shared every second, and this period is sufficient to address both concerns.

18.6.2.2 Group Replication Throttling

Based on the metrics gathered across all servers in the group, a throttling mechanism kicks in and decides whether to limit the rate a member is able to execute/commit new transactions.

Therefore, metrics acquired from all members are the basis for calculating the capacity of each member: if a member has a large queue (for certification or the applier thread), then the capacity to execute new transactions should be close to ones certified or applied in the last period.

The lowest capacity of all the members in the group determines the real capacity of the group, while the number of local transactions determines how many members are writing to it, and, consequently, how many members should that available capacity be shared with.

This means that every member has an established write quota based on the available capacity, in other words a number of transactions it can safely issue for the next period. The writer-quota will be enforced by the throttling mechanism if the queue size of the certifier or the binary log applier exceeds a user-defined threshold.

The quota is reduced by the number of transactions that were delayed in the last period, and then also further reduced by 10% to allow the queue that triggered the problem to reduce its size. In order to avoid large jumps in throughput once the queue size goes beyond the threshold, the throughput is only allowed to grow by the same 10% per period after that.

The current throttling mechanism does not penalize transactions below quota, but delays finishing those transactions that exceed it until the end of the monitoring period. As a consequence, if the quota is very small for the write requests issued some transactions may have latencies close to the monitoring period.

18.6.3 Message Compression

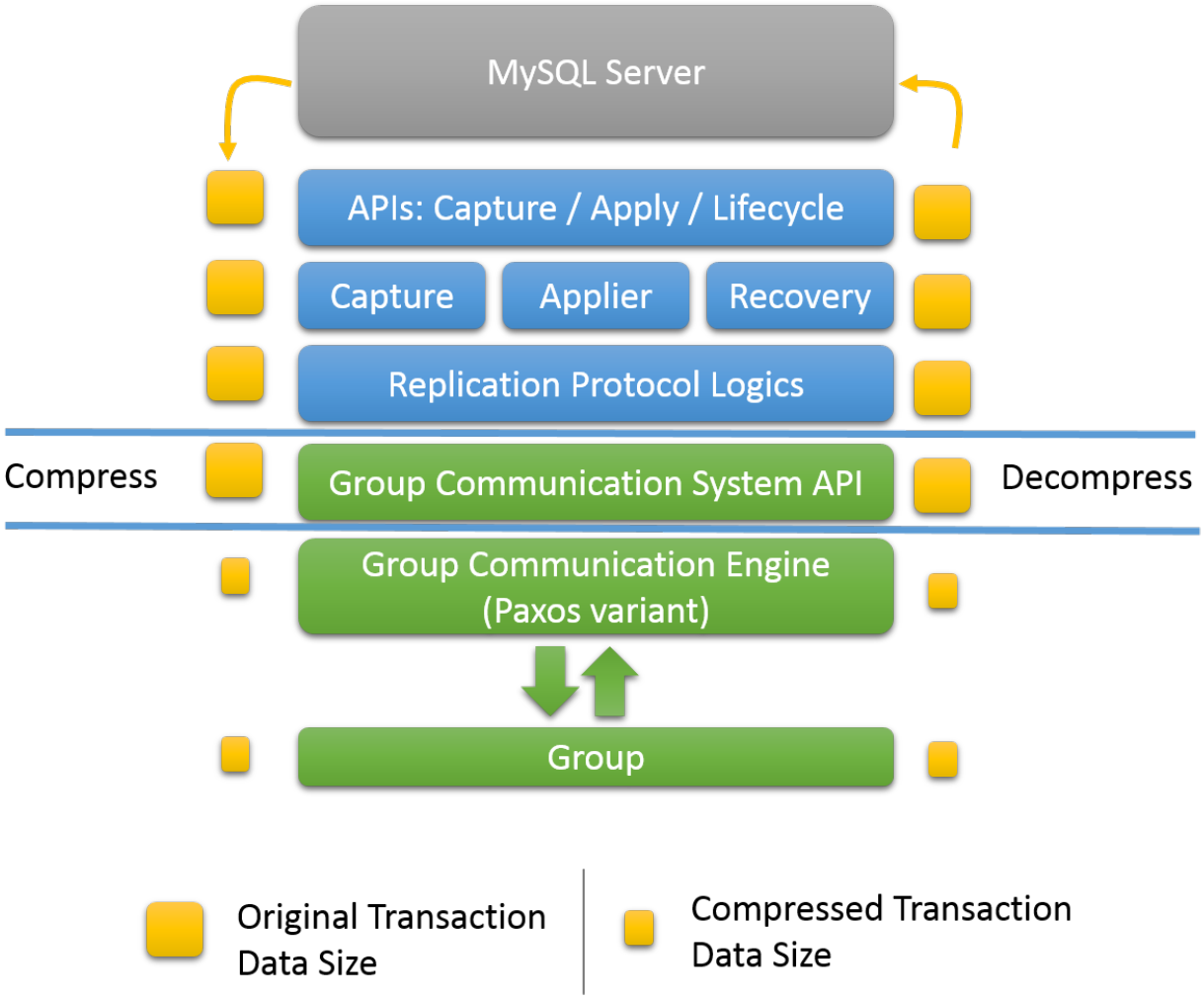
When network bandwidth is a bottleneck, message compression can provide up to 30-40% throughput improvement at the group communication level. This is especially important within the context of large groups of servers under load.

Table 18.4 LZ4 Compression Ratios for Different Binary Log Formats

Workload	Ratio for STMT	Ratio for ROW
mysqlslapd	4.5	2.1
sysbench	3.1	2.1

The TCP peer-to-peer nature of the interconnections between *N* participants on the group makes the sender send the same amount of data *N* times. Furthermore, binary logs are likely to exhibit a high compression ratio (see table above). This makes compression a compelling feature for workloads that contain large transaction.

Figure 18.15 Compression Support



Compression happens at the group communication engine level, before the data is handed over to the group communication thread, so it happens within the context of the mysql user session thread. Transaction payloads may be compressed before being sent out to the group and decompressed when received. Compression is conditional and depends on a configured threshold. By default compression is enabled.

In addition, there is no requirement that all servers in the group have compression enabled to be able to work together. Upon receiving a message, the member checks the message envelope to verify whether it is compressed or not. If needed, then the member decompresses the transaction, before delivering it to the upper layer.

The compression algorithm used is LZ4. Compression is enabled by default with threshold of 1000000 bytes. The compression threshold, in bytes, may be set to something larger than default. In that case, only transactions that have a payload larger than the threshold are compressed. Below is an example of how to set a compression threshold.

```
STOP GROUP_REPLICATION;
SET GLOBAL group_replication_compression_threshold= 2097152;
START GROUP_REPLICATION;
```

This sets the compression threshold to 2MB. If a transaction generates a replication message with a payload larger than 2MB, for example a binary log transaction entry larger than 2MB, then it is compressed. To disable compression set threshold to 0.

18.6.4 Message Fragmentation

When an abnormally large message is sent between Group Replication group members, it can result in some group members being reported as failed and expelled from the group. This is because the single thread used by Group Replication's group communication engine (XCom, a Paxos variant) is occupied processing the message for too long, so some of the group members might report the receiver as failed. From MySQL 8.0.16, by default, large messages are automatically split into fragments that are sent separately and reassembled by the recipients.

The system variable `group_replication_communication_max_message_size` specifies a maximum message size for Group Replication communications, above which messages are fragmented. The default maximum message size is 10485760 bytes (10 MiB). The greatest permitted value is the same as the maximum value of the `slave_max_allowed_packet` system variable, which is 1073741824 bytes (1 GB). The setting for `group_replication_communication_max_message_size` must be less than the `slave_max_allowed_packet` setting, because the applier thread cannot handle message fragments larger than `slave_max_allowed_packet`. To switch off fragmentation, specify a zero value for `group_replication_communication_max_message_size`.

As with most other Group Replication system variables, you must restart the Group Replication plugin for the change to take effect. For example:

```
STOP GROUP_REPLICATION;
SET GLOBAL group_replication_communication_max_message_size= 5242880;
START GROUP_REPLICATION;
```

Message delivery for a fragmented message is considered complete when all the fragments of the message have been received and reassembled by all the group members. Fragmented messages include information in their headers that enables a member joining during message transmission to recover the earlier fragments that were sent before it joined. If the joining member fails to recover the fragments, it expels itself from the group.

In order for a replication group to use fragmentation, all group members must be at MySQL 8.0.16 or above, and the Group Replication communication protocol version in use by the group must allow fragmentation. You can inspect the communication protocol in use by a group by using the `group_replication_get_communication_protocol()` UDF, which returns the oldest MySQL Server version that the group supports. Versions from MySQL 5.7.14 allow compression of messages, and versions from MySQL 8.0.16 also allow fragmentation of messages. If all group members are at MySQL 8.0.16 or above and there is no requirement to allow members at earlier releases to join, you can use the `group_replication_set_communication_protocol()` UDF to set the communication protocol version to MySQL 8.0.16 or above in order to allow fragmentation. For more information, see Section 18.4.1.4, “Setting a Group's Communication Protocol Version”.

If a replication group cannot use fragmentation because some members do not support it, the system variable `group_replication_transaction_size_limit` can be used to limit the maximum size of transactions the group accepts. In MySQL 8.0, the default setting is approximately 143 MB. Transactions above this size are rolled back. You can also use the system variable `group_replication_member_expel_timeout` to allow additional time (up to an hour) before a member under suspicion of having failed is expelled from the group.

18.6.5 XCom Cache Management

The group communication engine for Group Replication (XCom, a Paxos variant) includes a cache for messages (and their metadata) exchanged between the group members as a part of the consensus protocol. Among other functions, the message cache is used for recovery by members that return to the group after a period where they were unable to communicate with the other group members.

From MySQL 8.0.16, a cache size limit can be set for XCom's message cache using the `group_replication_message_cache_size` system variable. This system variable has a default and minimum setting of 1 GB, which is the size of the message cache in MySQL Server versions prior to MySQL 8.0.16. If the cache size limit is reached, XCom removes the oldest entries that have been decided and delivered. Ensure that sufficient memory is available on your system for your chosen cache size limit, considering the size of MySQL Server's other caches and object pools.

If an unreachable member that is attempting to reconnect requires a message for recovery, but the message has already been removed from the message cache, the member cannot reconnect. This situation is more likely to occur if you have used the `group_replication_member_expel_timeout` system variable (introduced in MySQL 8.0.13) to specify an additional delay time before suspect members are expelled from a group. Group Replication's Group Communication System (GCS) alerts you, by a warning message, when a message that is likely to be needed for recovery by a member that is currently unreachable is removed from the message cache. This warning message is logged on all the active group members (only once for each unreachable member). Although the group members cannot know for sure what message was the last message seen by the unreachable member, the warning message indicates that the cache size might not be sufficient to support your chosen waiting period before a member is expelled. In this situation, consider increasing the cache size limit with reference to the expected volume of messages in the time period specified by the `group_replication_member_expel_timeout` system variable, so that the cache contains all the missed messages required for members to return successfully. You can also consider increasing the cache size limit temporarily if you expect a member to become unreachable for an unusual period of time.

If you are considering reducing the cache size limit, you can query the Performance Schema table `memory_summary_global_by_event_name` using the following statement:

```
SELECT * FROM performance_schema.memory_summary_global_by_event_name
  WHERE EVENT_NAME LIKE 'memory/group_rpl/GCS_XCom::xcom_cache';
```

This returns memory usage statistics for the message cache, including the current number of cached entries and current size of the cache. If you reduce the cache size limit, XCom removes the oldest entries that have been decided and delivered until the current size is below the limit. XCom might temporarily exceed the cache size limit while this removal process is ongoing.

18.6.6 Responses to Failure Detection and Network Partitioning

Group Replication's failure detection mechanism is designed to identify group members that are no longer communicating with the group, and expel them as and when it seems likely that they have failed. Having a failure detection mechanism increases the chance that the group contains a majority of correctly working members, and that requests from clients are therefore processed correctly.

Normally, all group members regularly exchange messages with all other group members. If a group member does not receive any messages from a particular fellow member for 5 seconds, when this detection period ends, it creates a suspicion of the fellow member. When a suspicion times out, the suspected member is assumed to have failed, and is expelled from the group. An expelled member is removed from the membership list seen by the other members, but it does not know that it has been expelled from the group, so it sees itself as online and the other members as unreachable. If the member has not in fact failed (for example, because it was just disconnected due to a temporary network issue) and it is able to resume communication with the other members, it receives a view containing the information that it has been expelled from the group.

The responses of group members, including the failed member itself, to these situations can be configured at a number of points in the process. By default, the following behaviors happen if a member is suspected of having failed:

1. When a suspicion is created, it times out immediately (its lifetime is set to 0), so the suspected member is expelled as soon as the expired suspicion is identified. The member could potentially survive for a further few seconds after the timeout because the check for expired suspicions is carried out periodically.
2. If an expelled member resumes communication and realises that it was expelled, it does not try to rejoin the group and accepts its expulsion.
3. When an expelled member accepts its expulsion, it switches to super read only mode and awaits operator attention. (The exception is in releases from MySQL 8.0.12 to 8.0.15, where the default was for the member to shut itself down. From MySQL 8.0.16, the behavior was changed to match the behavior in MySQL 5.7.)

These defaults are set to prioritize the correct operation of the group and the correct handling of requests. However, they might be inconvenient in the case of slower networks or networks with a high rate of transient failures, because in these situations there could be a frequent requirement for operator intervention to fix expelled members. They also do not allow for continued operation of the group to be planned in the case of expected network failures or machine slowdowns. You can use the Group Replication configuration options described in this section to change these behaviors either permanently or temporarily, to suit your system's requirements and your priorities.

Members that have not failed might lose contact with part, but not all, of the replication group due to a network partition. For example, in a group of 5 servers (S1,S2,S3,S4,S5), if there is a disconnection between (S1,S2) and (S3,S4,S5) there is a network partition. The first group (S1,S2) is now in a minority because it cannot contact more than half of the group. Any transactions that are processed by the members in the minority group are blocked, because the majority of the group is unreachable, therefore the group cannot achieve quorum. For a detailed description of this scenario, see Section 18.4.4, “Network Partitioning”. In this situation, the default behavior is for the members in both the minority and the majority to remain in the group, continue to accept transactions (although they are blocked on the members in the minority), and wait for operator intervention. This behavior is also configurable.

Note that where group members are at an older MySQL Server release that does not support a relevant setting, or at a release with a different default, they act towards themselves and other group members according to the default behaviors stated above. For example, a member that does not support the `group_replication_member_expel_timeout` system variable expels other members as soon as an expired suspicion is detected, and this expulsion is accepted by other members even if they support the system variable and have a longer timeout set.

18.6.6.1 Expel Timeout

You can use the `group_replication_member_expel_timeout` system variable, which is available from MySQL 8.0.13, to allow additional time between the creation of a suspicion and the expulsion of the suspect member.

A group member is expelled when another member's suspicion of it (or its own suspicion of itself) times out. By default, `group_replication_member_expel_timeout` is set to 0, meaning that there is no waiting period and a suspected member is liable for expulsion immediately after the 5-second detection period ends. An additional short period of time might elapse before the expelling mechanism detects and implements the expulsion. If a group member is at an older MySQL Server version that does not support this setting, this is its behavior towards other members or itself.

To avoid unnecessary expulsions on slower networks, or in the case of expected transient network failures or machine slowdowns, you can specify a timeout value greater than zero, up to a maximum of 3600

seconds (1 hour). Suspect members in this state are listed as `UNREACHABLE`, but are not removed from the group's membership list. If a suspect member becomes active again before the suspicion times out, it rejoins the group, applies all the messages that were buffered by the remaining group members, and enters `ONLINE` state.

If the timeout is exceeded, the suspect member is liable for expulsion immediately after the suspicion times out. If the member is able to resume communications and receives a view where it is expelled, it realises it was expelled. By default, the expelled member then follows the exit action specified by `group_replication_exit_state_action`. Alternatively you can use the `group_replication_autorejoin_tries` system variable, which is available from MySQL 8.0.16, to make the member automatically try to rejoin the group.

The waiting period before expelling a member only applies to members that have previously been active in the group. Non-members that were never active in the group do not get this waiting period and are removed after the initial detection period because they took too long to join.

If any members in a group are currently under suspicion, the group membership cannot be reconfigured (by adding or removing members or electing a new leader). If group membership changes need to be implemented while one or more members are under suspicion, and you want the suspect members to remain in the group, take any actions required to make the members active again, if that is possible. If you cannot make the members active again and you want them to be expelled from the group, you can force the suspicions to time out immediately. Do this by changing the value of `group_replication_member_expel_timeout` on any active members to a value lower than the time that has already elapsed since the suspicions were created. The suspect members then become liable for expulsion immediately.

For alternative mitigation strategies to avoid unnecessary expulsions where the `group_replication_member_expel_timeout` system variable is not available, see Section 18.9.2, “Group Replication Limitations”.

18.6.6.2 Unreachable Majority Timeout

By default, members that find themselves in a minority due to a network partition do not automatically leave the group. You can use the system variable `group_replication_unreachable_majority_timeout` to set a number of seconds for a member to wait after losing contact with the majority of group members, and then exit the group. Setting a timeout means you do not need to pro-actively monitor for servers that are in a minority group after a network partition, and you can avoid the possibility of creating a split-brain situation (with two versions of the group membership) due to inappropriate intervention.

When the timeout specified by `group_replication_unreachable_majority_timeout` elapses, all pending transactions that have been processed by the member and the others in the minority group are rolled back, and the servers in that group move to the `ERROR` state. By default, a minority member then follows the exit action specified by `group_replication_exit_state_action`. Alternatively you can use the `group_replication_autorejoin_tries` system variable, which is available from MySQL 8.0.16, to make the member automatically try to rejoin the group.

Consider the following points when deciding whether or not to set an unreachable majority timeout:

- In a symmetric group, for example a group with two or four servers, if both partitions contain an equal number of servers, both groups consider themselves to be in a minority and enter the `ERROR` state. In this situation, the group has no functional partition.
- While a minority group exists, any transactions processed by the minority group are accepted, but blocked because the minority servers cannot reach quorum, until either `STOP GROUP_REPLICATION` is issued on those servers or the unreachable majority timeout is reached.

- If you do not set an unreachable majority timeout, the servers in the minority group will never enter the `ERROR` state automatically, and you must stop them manually.
- Setting an unreachable majority timeout has no effect if it is set on the servers in the minority group after the loss of majority has been detected.

If you do not use the `group_replication_unreachable_majority_timeout`system variable, the process for operator invention in the event of a network partition is described in Section 18.4.4, “Network Partitioning”. The process involves checking which servers are functioning and forcing a new group membership if necessary.

18.6.6.3 Auto-Rejoin

The `group_replication_autorejoin_tries` system variable, which is available from MySQL 8.0.16, makes a member that has been expelled or reached its unreachable majority timeout try to rejoin the group automatically. You can specify a number of attempts that the member makes to rejoin the group, instead of just accepting its expulsion as soon as it resumes communication. Activate auto-rejoin if you can tolerate the possibility of stale reads and want to minimize the need for manual intervention, especially where transient network issues fairly often result in the expulsion of members.

When the member's expulsion or unreachable majority timeout is reached, it makes an attempt to rejoin (using the current plugin option values), then continues to make further auto-rejoin attempts up to the specified number of tries. After an unsuccessful auto-rejoin attempt, the member waits 5 minutes before the next try. If the specified number of tries is exhausted without the member rejoining or being stopped, the member proceeds to the action specified by the `group_replication_exit_state_action` system variable.

During the auto-rejoin procedure, the expelled member remains in super read only mode and displays an `ERROR` state on its view of the replication group. Bear in mind that while a member remains in super read only mode, although writes cannot be made on the member, reads can, with an increasing likelihood of stale reads over time. If you do want to intervene to take the member offline, the member can be stopped manually at any time by using a `STOP GROUP_REPLICATION` statement or shutting down the server.

You can monitor the auto-rejoin procedure using the Performance Schema. While an auto-rejoin procedure is taking place, the Performance Schema table `events_stages_current` shows the event “Undergoing auto-rejoin procedure”, with the number of retries that have been attempted so far during this instance of the procedure (in the `WORK_COMPLETED` field). The `events_stages_summary_global_by_event_name` table shows the number of times the server instance has initiated the auto-rejoin procedure (in the `COUNT_STAR` field). The `events_stages_history_long` table shows the time each of these auto-rejoin procedures was completed (in the `TIMER_END` field).

18.6.6.4 Exit Action

The `group_replication_exit_state_action` system variable, which is available from MySQL 8.0.12 and MySQL 5.7.24, specifies what Group Replication does when the member leaves the group unintentionally due to an error or problem, and either fails to auto-rejoin or does not try. Note that in the case of an expelled member, the member does not know that it was expelled until it reconnects to the group, so the specified action is only taken if the member manages to reconnect, or if the member raises a suspicion on itself and expels itself.

In order of impact, the exit actions are as follows:

1. If `READ_ONLY` is the exit action, the instance switches MySQL to super read only mode by setting the system variable `super_read_only` to `ON`. When the member is in super read only mode, clients

cannot make any updates, even if they have the SUPER privilege. However, clients can still read data, and because updates are no longer being made, there is a probability of stale reads which increases over time. With this setting, you therefore need to pro-actively monitor the servers for failures. This exit action is the default from MySQL 8.0.15. After this exit action is taken, the member's status is displayed as ERROR in the view of the group.

2. If OFFLINE_MODE is the exit action, the instance switches MySQL to offline mode by setting the system variable offline_mode to ON. When the member is in offline mode, connected client users are disconnected on their next request and connections are no longer accepted, with the exception of client users that have the CONNECTION_ADMIN or SUPER privilege. Group Replication also sets the system variable super_read_only to ON, so clients cannot make any updates, even if they have connected with the SUPER privilege. This exit action prevents both updates and stale reads (with the exception of reads by client users with the stated privileges), and enables proxy tools such as MySQL Router to recognize that the server is unavailable and redirect client connections. It also leaves the instance running so that an administrator can attempt to resolve the issue without shutting down MySQL. This exit action is available from MySQL 8.0.18. After this exit action is taken, the member's status is displayed as ERROR in the view of the group (not OFFLINE, which means a member has Group Replication functionality available but does not currently belong to a group).

3. If ABORT_SERVER is the exit action, the instance shuts down MySQL. Instructing the member to shut itself down prevents all stale reads and client updates, but it means that the MySQL Server instance is unavailable and must be restarted, even if the issue could have been resolved without that step. This exit action was the default from MySQL 8.0.12, when the system variable was added, to MySQL 8.0.15 inclusive. After this exit action is taken, the member is removed from the listing of servers in the view of the group.

Bear in mind that operator intervention is required whatever exit action is set, as an ex-member that has exhausted its auto-rejoin attempts (or never had any) and has been expelled from the group is not allowed to rejoin without a restart of Group Replication. The exit action only influences whether or not clients can still read data on the server that was unable to rejoin the group, and whether or not the server stays running.

Important

If a failure occurs before the member has successfully joined the group, the exit action specified by group_replication_exit_state_action *is not taken*. This is the case if there is a failure during the local configuration check, or a mismatch between the configuration of the joining member and the configuration of the group. In these situations, the super_read_only system variable is left with its original value, and the server does not shut down MySQL. To ensure that the server cannot accept updates when Group Replication did not start, we therefore recommend that super_read_only=ON is set in the server's configuration file at startup, which Group Replication will change to OFF on primary members after it has been started successfully. This safeguard is particularly important when the server is configured to start Group Replication on server boot (group_replication_start_on_boot=ON), but it is also useful when Group Replication is started manually using a START GROUP_REPLICATION command.

If a failure occurs after the member has successfully joined the group, the specified exit action is taken. This is the case in the following situations:

1. *Applier error* - There is an error in the replication applier. This issue is not recoverable.

2. *Distributed recovery not possible* - There is an issue that means Group Replication's distributed recovery process (which uses remote cloning operations and state transfer from the binary log) cannot be completed. Group Replication retries distributed recovery automatically where this makes sense,

but stops if there are no more options to complete the process. For details, see Section 18.4.3.3, “Fault Tolerance for Distributed Recovery”.

3. *Group configuration change error* - An error occurred during a group-wide configuration change carried out using a UDF, as described in Section 18.4.1, “Configuring an Online Group”.
4. *Primary election error* - An error occurred during election of a new primary member for a group in single-primary mode, as described in Section 18.1.3.1, “Single-Primary Mode”.
5. *Unreachable majority timeout* - The member has lost contact with a majority of the group members so is in a minority, and a timeout that was set by the `group_replication_unreachable_majority_timeout` system variable has expired.
6. *Member expelled from group* - A suspicion has been raised on the member, and any timeout set by the `group_replication_member_expel_timeout` system variable has expired, and the member has resumed communication with the group and found that it has been expelled.
7. *Out of auto-rejoin attempts* - The `group_replication_autorejoin_tries` system variable was set to specify a number of auto-rejoin attempts after a loss of majority or expulsion, and the member completed this number of attempts without success.

The following table summarizes the failure scenarios and actions in each case:

Table 18.5 Exit actions in Group Replication failure situations

Failure situation	Group Replication started with `START GROUP_REPLICATION`	Group Replication started with `group_replication_start_on_boot=ON`
Member fails local configuration check Mismatch between joining member and group configuration	`super_read_only` and `offline_mode` unchanged MySQL continues running Set `super_read_only=ON` at startup to prevent updates	`super_read_only` and `offline_mode` unchanged MySQL continues running Set `super_read_only=ON` at startup to prevent updates (Important)

Applier error on member
 	
Distributed recovery not possible
 	
Group configuration change error
 	
Primary election error
 	
Unreachable majority timeout
 	
Member expelled from group
 	
Out of auto-rejoin attempts
	`super_read_only` set to `ON`
	
	OR
	
	`offline_mode` and `super_read_only` set to `ON`
	
	OR
	
	MySQL shuts down
		`super_read_only` set to `ON`
		
		OR
		
		`offline_mode` and `super_read_only` set to `ON`
		
		OR
		
		MySQL shuts down

18.7 Upgrading Group Replication

This section explains how to upgrade a Group Replication setup. The basic process of upgrading members of a group is the same as upgrading stand-alone instances, see Section 2.11, “Upgrading MySQL” for the actual process of doing upgrade and types available. Choosing between an in-place or logical upgrade depends on the amount of data stored in the group. Usually an in-place upgrade is faster, and therefore is recommended. You should also consult Section 17.5.3, “Upgrading a Replication Setup”.

While you are in the process of upgrading an online group, in order to maximize availability, you might need to have members with different MySQL Server versions running at the same time. Group Replication includes compatibility policies that enable you to safely combine members running different versions of MySQL in the same group during the upgrade procedure. Depending on your group, the effects of these policies might affect the order in which you should upgrade group members. For details, see Section 18.7.1, “Combining Different Member Versions in a Group”.

If your group can be taken fully offline see Section 18.7.2, “Group Replication Offline Upgrade”. If your group needs to remain online, as is common with production deployments, see Section 18.7.3, “Group Replication Online Upgrade” for the different approaches available for upgrading a group with minimal downtime.

18.7.1 Combining Different Member Versions in a Group

Group Replication is versioned according to the MySQL Server version that the Group Replication plugin was bundled with. For example, if a member is running MySQL 5.7.26 then that is the version of the Group Replication plugin. To check the version of MySQL Server on a group member issue:

```
SELECT MEMBER_HOST,MEMBER_PORT,MEMBER_VERSION FROM performance_schema.replication_group_members;
+-------------+-------------+----------------+
| member_host | member_port | member_version |
+-------------+-------------+----------------+
| example.com |    3306     |   8.0.13       |
+-------------+-------------+----------------+
```

For guidance on understanding the MySQL Server version and selecting a version, see Section 2.1.1, “Which MySQL Version and Distribution to Install”.

For optimal compatibility and performance, all members of a group should run the same version of MySQL Server and therefore of Group Replication. However, while you are in the process of upgrading an online group, in order to maximize availability, you might need to have members with different MySQL Server versions running at the same time. Depending on the changes made between the versions of MySQL, you could encounter incompatibilities in this situation. For example, if a feature has been deprecated between major versions, then combining the versions in a group might cause members that rely on the deprecated feature to fail. Conversely, writing to a member running a newer MySQL version while there are read-write members in the group running an older MySQL version might cause issues on members that lack functions introduced in the newer release.

To prevent these issues, Group Replication includes compatibility policies that enable you to safely combine members running different versions of MySQL in the same group. A member applies these policies to decide whether to join the group normally, or join in read-only mode, or not join the group, depending on which choice results in the safe operation of the joining member and of the existing members of the group. In an upgrade scenario, each server must leave the group, be upgraded, and rejoin the group with its new server version. At this point the member applies the policies for its new server version, which might have changed from the policies it applied when it originally joined the group.

As the administrator, you can instruct any server to attempt to join any group by configuring the server appropriately and issuing a `START GROUP_REPLICATION` statement. A decision to join or not join the group, or to join the group in read-only mode, is made and implemented by the joining member itself after you attempt to add it to the group. The joining member receives information on the MySQL Server versions of the current group members, assesses its own compatibility with those members, and applies the policies used in its own MySQL Server version (*not* the policies used by the existing members) to decide whether it is compatible.

The compatibility policies that a joining member applies when attempting to join a group are as follows:

- A member does not join a group if it is running a lower MySQL Server version than the lowest version that the existing group members are running.
- A member joins a group normally if it is running the same MySQL Server version as the lowest version that the existing group members are running.
- A member joins a group but remains in read-only mode if it is running a higher MySQL Server version than the lowest version that the existing group members are running. This behavior only makes a difference when the group is running in multi-primary mode, because in a group that is running in single-primary mode, newly added members default to being read-only in any case.

Members running MySQL 8.0.17 or higher take into account the patch version of the release when checking their compatibility. Members running MySQL 8.0.16 or lower, or MySQL 5.7, only take into account the major version. For example, if you have a group with members all running MySQL version 8.0.13:

- A member that is running MySQL version 5.7 does not join.
- A member running MySQL 8.0.16 joins normally (because it considers the major version).
- A member running MySQL 8.0.17 joins but remains in read-only mode (because it considers the patch version).

Note that joining members running releases before MySQL 5.7.27 check against all group members to find whether their own MySQL Server major version is lower. They therefore fail this check for a group where any members are running MySQL 8.0 releases, and cannot join the group even if it already has other members running MySQL 5.7. From MySQL 5.7.27, joining members only check against the group members that are running the lowest major version, so they can join a mixed version group where other MySQL 5.7 servers are present.

In a multi-primary mode group with members that use different MySQL Server versions, Group Replication automatically manages the read-write and read-only status of members running MySQL 8.0.17 or higher. If a member leaves the group, the members running the version that is now the lowest are automatically set to read-write mode. When you change a group that was running in single-primary mode to run in multi-primary mode, using the `group_replication_switch_to_multi_primary_mode()` UDF, Group Replication automatically sets members to the correct mode. Members are automatically placed in read-only mode if they are running a higher MySQL server version than the lowest version present in the group, and members running the lowest version are placed in read-write mode.

18.7.1.1 Member Versions During Upgrades

During an online upgrade procedure, if the group is in single-primary mode, all the servers that are not currently offline for upgrading function as they did before. The group elects a new primary whenever necessary, following the election policies described in Section 18.1.3.1, “Single-Primary Mode”. Note that if you require the primary to remain the same throughout (except when it is being upgraded itself), you must first upgrade all of the secondaries to a version higher than or equal to the target primary member version, then upgrade the primary last. The primary cannot remain as the primary unless it is running the lowest MySQL Server version in the group. After the primary has been upgraded, you can use the `group_replication_set_as_primary()` UDF to reappoint it as the primary.

If the group is in multi-primary mode, fewer online members are available to perform writes during the upgrade procedure, because upgraded members join in read-only mode after their upgrade. From MySQL 8.0.17, this applies to upgrades between patch versions, and for lower releases, this only applies to upgrades between major versions. When all members have been upgraded to the same release, from MySQL 8.0.17, they all change back to read-write mode automatically. For earlier releases, you must set

`super_read_only` to `OFF` manually on each member that should function as a primary following the upgrade.

To deal with a problem situation, for example if you have to roll back an upgrade or add extra capacity to a group in an emergency, it is possible to allow a member to join an online group although it is running a lower MySQL Server version than the lowest version in use by other group members. The Group Replication system variable `group_replication_allow_local_lower_version_join` can be used in such situations to override the normal compatibility policies. It is important to note that setting the option to `ON` does not make the new member compatible with the group, and allows it to join the group without any safeguards against incompatible behaviors by the existing members. The option must therefore only be used carefully in specific situations, and you must take additional precautions to avoid the new member failing due to normal group activity. For details of these precautions, see the description for `group_replication_allow_local_lower_version_join`.

18.7.1.2 Group Replication Communication Protocol Version

A replication group uses a Group Replication communication protocol version that can differ from the MySQL Server version of the members. To check the group's communication protocol version, issue the following statement on any member:

```
SELECT group_replication_get_communication_protocol();
```

The return value shows the oldest MySQL Server version that can join this group and use the group's communication protocol. Versions from MySQL 5.7.14 allow compression of messages, and versions from MySQL 8.0.16 also allow fragmentation of messages. Note that the `group_replication_get_communication_protocol()` UDF returns the minimum MySQL version that the group supports, which might differ from the version number that was passed to the `group_replication_set_communication_protocol()` UDF, and from the MySQL Server version that is installed on the member where you use the UDF.

When you upgrade all the members of a replication group to a new MySQL Server release, the Group Replication communication protocol version is not automatically upgraded, in case there is still a requirement to allow members at earlier releases to join. If you do not need to support older members and want to allow the upgraded members to use any added communication capabilities, after the upgrade use the `group_replication_set_communication_protocol()` UDF to upgrade the communication protocol, specifying the new MySQL Server version to which you have upgraded the members. For more information, see Section 18.4.1.4, “Setting a Group's Communication Protocol Version”.

18.7.2 Group Replication Offline Upgrade

To perform an offline upgrade of a Group Replication group, you remove each member from the group, perform an upgrade of the member and then restart the group as usual. In a multi-primary group you can shutdown the members in any order. In a single-primary group, shutdown each secondary first and then finally the primary. See Section 18.7.3.2, “Upgrading a Group Replication Member” for how to remove members from a group and shutdown MySQL.

Once the group is offline, upgrade all of the members. See Section 2.11, “Upgrading MySQL” for how to perform an upgrade. When all members have been upgraded, restart the members.

If you upgrade all the members of a replication group when they are offline and then restart the group, the members join using the new release's Group Replication communication protocol version, so that becomes the group's communication protocol version. If you have a requirement to allow members at earlier releases to join, you can use the `group_replication_set_communication_protocol()` UDF to downgrade the communication protocol version, specifying the MySQL Server version of the prospective group member that has the oldest installed server version.

18.7.3 Group Replication Online Upgrade

When you have a group running which you want to upgrade but you need to keep the group online to serve your application, you need to consider your approach to the upgrade. This section describes the different elements involved in an online upgrade, and various methods of how to upgrade your group.

18.7.3.1 Online Upgrade Considerations

When upgrading an online group you should consider the following points:

- Regardless of the way which you upgrade your group, it is important to disable any writes to group members until they are ready to rejoin the group.
- When a member is stopped, the `super_read_only` variable is set to on automatically, but this change is not persisted.
- When MySQL 5.7.22 or MySQL 8.0.11 tries to join a group running MySQL 5.7.21 or lower it fails to join the group because MySQL 5.7.21 does not send its value of `lower_case_table_names`.

18.7.3.2 Upgrading a Group Replication Member

This section explains the steps required for upgrading a member of a group. This procedure is part of the methods described at Section 18.7.3.3, “Group Replication Online Upgrade Methods”. The process of upgrading a member of a group is common to all methods and is explained first. The way which you join upgraded members can depend on which method you are following, and other factors such as whether the group is operating in single-primary or multi-primary mode. How you upgrade the server instance, using either the in-place or provision approach, does not impact on the methods described here.

The process of upgrading a member consists of removing it from the group, following your chosen method of upgrading the member, and then rejoining the upgraded member to a group. The recommended order of upgrading members in a single-primary group is to upgrade all secondaries, and then upgrade the primary last. If the primary is upgraded before a secondary, a new primary using the older MySQL version is chosen, but there is no need for this step.

To upgrade a member of a group:

- Connect a client to the group member and issue `STOP GROUP_REPLICATION`. Before proceeding, ensure that the member's status is `OFFLINE` by monitoring the `replication_group_members` table.
- Disable Group Replication from starting up automatically so that you can safely connect to the member after upgrading and configure it without it rejoining the group by setting `group_replication_start_on_boot=0`.

Important

If an upgraded member has `group_replication_start_on_boot=1` then it could rejoin the group before you can perform the MySQL upgrade procedure and could result in issues. For example, if the upgrade fails and the server restarts again, then a possibly broken server could try to join the group.

- Stop the member, for example using `mysqladmin shutdown` or the `SHUTDOWN` statement. Any other members in the group continue running.
- Upgrade the member, using the in-place or provisioning approach. See Section 2.11, “Upgrading MySQL” for details. When restarting the upgraded member, because `group_replication_start_on_boot` is set to 0, Group Replication does not start on the instance, and therefore it does not rejoin the group.

- Once the MySQL upgrade procedure has been performed on the member, `group_replication_start_on_boot` must be set to 1 to ensure Group Replication starts correctly after restart. Restart the member.
- Connect to the upgraded member and issue `START GROUP_REPLICATION`. This rejoins the member to the group. The Group Replication metadata is in place on the upgraded server, therefore there is usually no need to reconfigure Group Replication. The server has to catch up with any transactions processed by the group while the server was offline. Once it has caught up with the group, it becomes an online member of the group.

Note

The longer it takes to upgrade a server, the more time that member is offline and therefore the more time it takes for the server to catch up when added back to the group.

When an upgraded member joins a group which has any member running an earlier MySQL Server version, the upgraded member joins with `super_read_only=on`. This ensures that no writes are made to upgraded members until all members are running the newer version. In a multi-primary mode group, when the upgrade has been completed successfully and the group is ready to process transactions, members that are intended as writeable primaries must be set to read-write mode. From MySQL 8.0.17, when all members of a group have been upgraded to the same release, they all change back to read-write mode automatically. For earlier releases you must set each member manually to read-write mode. Connect to each member and issue:

```
SET GLOBAL super_read_only=OFF;
```

18.7.3.3 Group Replication Online Upgrade Methods

Choose one of the following methods of upgrading a Group Replication group:

Rolling In-Group Upgrade

This method is supported provided that servers running a newer version are not generating workload to the group while there are still servers with an older version in it. In other words servers with a newer version can join the group only as secondaries. In this method there is only ever one group, and each server instance is removed from the group, upgraded and then rejoined to the group.

This method is well suited to single-primary groups. When the group is operating in single-primary mode, if you require the primary to remain the same throughout (except when it is being upgraded itself), it should be the last member to be upgraded. The primary cannot remain as the primary unless it is running the lowest MySQL Server version in the group. After the primary has been upgraded, you can use the `group_replication_set_as_primary()` UDF to reappoint it as the primary. If you do not mind which member is the primary, the members can be upgraded in any order. The group elects a new primary whenever necessary from among the members running the lowest MySQL Server version, following the election policies described in Section 18.1.3.1, “Single-Primary Mode”.

For groups operating in multi-primary mode, during a rolling in-group upgrade the number of primaries is decreased, causing a reduction in write availability. This is because if a member joins a group when it is running a higher MySQL Server version than the lowest version that the existing group members are running, it automatically remains in read-only mode (`super_read_only=ON`). Note that members running MySQL 8.0.17 or higher take into account the patch version of the release when checking this, but members running MySQL 8.0.16 or lower, or MySQL 5.7, only take into account the major version. When all members have been upgraded to the same release, from MySQL 8.0.17, they all change back to read-write mode automatically. For earlier releases, you must set `super_read_only=OFF` manually on each member that should function as a primary following the upgrade.

For full information on version compatibility in a group and how this influences the behavior of a group during an upgrade process, see Section 18.7.1, “Combining Different Member Versions in a Group” .

Rolling Migration Upgrade

In this method you remove members from the group, upgrade them and then create a second group using the upgraded members. For groups operating in multi-primary mode, during this process the number of primaries is decreased, causing a reduction in write availability. This does not impact groups operating in single-primary mode.

Because the group running the older version is online while you are upgrading the members, you need the group running the newer version to catch up with any transactions executed while the members were being upgraded. Therefore one of the servers in the new group is configured as a replication slave of a primary from the older group. This ensures that the new group catches up with the older group. Because this method relies on an asynchronous replication channel which is used to replicate data from one group to another, it is supported under the same assumptions and requirements of master-slave replication, see Chapter 17, *Replication*. For groups operating in single-primary mode, the asynchronous replication connection to the old group must send data to the primary in the new group, for a multi-primary group the asynchronous replication channel can connect to any primary.

The process is to:

- remove members from the original group running the older server version one by one, see Section 18.7.3.2, “Upgrading a Group Replication Member”
- upgrade the server version running on the member, see Section 2.11, “Upgrading MySQL”. You can either follow an in-place or provision approach to upgrading.
- create a new group with the upgraded members, see Chapter 18, *Group Replication*. In this case you need to configure a new group name on each member (because the old group is still running and using the old name), bootstrap an initial upgraded member, and then add the remaining upgraded members.
- set up an asynchronous replication channel between the old group and the new group, see Section 17.1.3.4, “Setting Up Replication Using GTIDs”. Configure the older primary to function as the asynchronous replication master and the new group member as a GTID-based replication slave.

Before you can redirect your application to the new group, you must ensure that the new group has a suitable number of members, for example so that the group can handle the failure of a member. Issue `SELECT * FROM performance_schema.replication_group_members` and compare the initial group size and the new group size. Wait until all data from the old group is propagated to the new group and then drop the asynchronous replication connection and upgrade any missing members.

Rolling Duplication Upgrade

In this method you create a second group consisting of members which are running the newer version, and the data missing from the older group is replicated to the newer group. This assumes that you have enough servers to run both groups simultaneously. Due to the fact that during this process the number of primaries is *not* decreased, for groups operating in multi-primary mode there is no reduction in write availability. This makes rolling duplication upgrade well suited to groups operating in multi-primary mode. This does not impact groups operating in single-primary mode.

Because the group running the older version is online while you are provisioning the members in the new group, you need the group running the newer version to catch up with any transactions executed while the members were being provisioned. Therefore one of the servers in the new group is configured as a replication slave of a primary from the older group. This ensures that the new group catches up with the older group. Because this method relies on an asynchronous replication channel which is used to

replicate data from one group to another, it is supported under the same assumptions and requirements of master-slave replication, see Chapter 17, *Replication*. For groups operating in single-primary mode, the asynchronous replication connection to the old group must send data to the primary in the new group, for a multi-primary group the asynchronous replication channel can connect to any primary.

The process is to:

- deploy a suitable number of members so that the group running the newer version can handle failure of a member
- take a backup of the existing data from a member of the group
- use the backup from the older member to provision the members of the new group, see Section 18.7.3.4, "Group Replication Upgrade with `mysqlbackup`" for one method.

Note

You must restore the backup to the same version of MySQL which the backup was taken from, and then perform an in-place upgrade. For instructions, see Section 2.11, "Upgrading MySQL".

- create a new group with the upgraded members, see Chapter 18, *Group Replication*. In this case you need to configure a new group name on each member (because the old group is still running and using the old name), bootstrap an initial upgraded member, and then add the remaining upgraded members.
- set up an asynchronous replication channel between the old group and the new group, see Section 17.1.3.4, "Setting Up Replication Using GTIDs". Configure the older primary to function as the asynchronous replication master and the new group member as a GTID-based replication slave.

Once the ongoing data missing from the newer group is small enough to be quickly transferred, you must redirect write operations to the new group. Wait until all data from the old group is propagated to the new group and then drop the asynchronous replication connection.

18.7.3.4 Group Replication Upgrade with `mysqlbackup`

As part of a provisioning approach you can use MySQL Enterprise Backup to copy and restore the data from a group member to new members. However you cannot use this technique to directly restore a backup taken from a member running an older version of MySQL to a member running a newer version of MySQL. The solution is to restore the backup to a new server instance which is running the same version of MySQL as the member which the backup was taken from, and then upgrade the instance. This process consists of:

- Take a backup from a member of the older group using `mysqlbackup`. See Section 18.4.6, "Using MySQL Enterprise Backup with Group Replication".
- Deploy a new server instance, which must be running the same version of MySQL as the older member where the backup was taken.
- Restore the backup from the older member to the new instance using `mysqlbackup`.
- Upgrade MySQL on the new instance, see Section 2.11, "Upgrading MySQL".

Repeat this process to create a suitable number of new instances, for example to be able to handle a failover. Then join the instances to a group based on the Section 18.7.3.3, "Group Replication Online Upgrade Methods".`

18.8 Group Replication System Variables

This section lists the system variables that are specific to the Group Replication plugin. Every configuration option is prefixed with "`group_replication`".

Most system variables for Group Replication are described as dynamic, and their values can be changed while the server is running. However, in most cases, the change only takes effect after you stop and restart Group Replication on the group member using a `STOP GROUP_REPLICATION` statement followed by a `START GROUP_REPLICATION` statement. Changes to the following system variables take effect without stopping and restarting Group Replication:

- `group_replication_flow_control_applier_threshold`
- `group_replication_flow_control_certifier_threshold`
- `group_replication_flow_control_mode`

Most system variables for Group Replication can have different values on different group members. For the following system variables, it is advisable to set the same value on all members of a group in order to avoid unnecessary rollback of transactions, failure of message delivery, or failure of message recovery:

- `group_replication_auto_increment_increment`
- `group_replication_communication_max_message_size`
- `group_replication_compression_threshold`
- `group_replication_message_cache_size`
- `group_replication_transaction_size_limit`

Some system variables on a Group Replication group member, including some Group Replication-specific system variables and some general system variables, are group-wide configuration settings. These system variables must have the same value on all group members, cannot be changed while Group Replication is running, and require a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) in order for the value change to take effect. These conditions apply to the following system variables:

- `group_replication_single_primary_mode`
- `group_replication_enforce_update_everywhere_checks`
- `group_replication_gtid_assignment_block_size`
- `default_table_encryption`
- `lower_case_table_names`
- `transaction_write_set_extraction`

From MySQL 8.0.16, you can use the `group_replication_switch_to_single_primary_mode()` and `group_replication_switch_to_multi_primary_mode()` UDFs to change the values of `group_replication_single_primary_mode` and `group_replication_enforce_update_everywhere_checks` while the group is still running. For more information, see Section 18.4.1.2, “Changing a Group's Mode”.

Important

- A number of system variables for Group Replication are not completely validated during server startup if they are passed as command line arguments to the

server. These system variables include `group_replication_group_name`, `group_replication_single_primary_mode`, `group_replication_force_members`, the SSL variables, and the flow control system variables. They are only fully validated after the server has started.

- System variables for Group Replication that specify IP addresses or host names for group members are not validated until a `START GROUP_REPLICATION` statement is issued. Group Replication's Group Communication System (GCS) is not available to validate the values until that point.

The system variables that are specific to the Group Replication plugin are as follows:

- `group_replication_allow_local_disjoint_gtids_join`

Property	Value
Command-Line Format	`--group-replication-allow-local-disjoint-gtids-join[={OFF|ON}]`
Deprecated	Yes (removed in 8.0.4)
System Variable	`group_replication_allow_local_disjoint_gtids_join`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Removed in version 8.0.4. Allows the server to join the group even if it has local transactions that are not present in the group.

Warning

Use caution when enabling this option as incorrect usage can lead to conflicts in the group and rollback of transactions. The option should only be enabled as a last resort method to allow a server that has local transactions to join an existing group, and then only if the local transactions do not affect the data that is handled by the group (for example, an administrative action that was written to the binary log). The option should not be left enabled on all group members.

- `group_replication_allow_local_lower_version_join`

Property	Value
Command-Line Format	`--group-replication-allow-local-lower-version-join[={OFF|ON}]`
System Variable	`group_replication_allow_local_lower_version_join`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Allows the current server to join the group even if it is running a lower MySQL Server version than the group. With the default setting `OFF`, servers are not permitted to join a replication group if they are running a lower version than the existing group members. This standard policy ensures that all members of a group are able to exchange messages and apply transactions. Note that members running MySQL 8.0.17 or higher take into account the patch version of the release when checking their compatibility. Members running MySQL 8.0.16 or lower, or MySQL 5.7, only take into account the major version.

Set `group_replication_allow_local_lower_version_join` to `ON` only in the following scenarios:

- A server must be added to the group in an emergency in order to improve the group's fault tolerance, and only older versions are available.
- You want to roll back an upgrade for one or more replication group members without shutting down the whole group and bootstrapping it again.

Warning

Setting this option to `ON` does not make the new member compatible with the group, and allows it to join the group without any safeguards against incompatible behaviors by the existing members. To ensure the new member's correct operation, take *both* of the following precautions:

1. Before the server running the lower version joins the group, stop all writes on that server.
2. From the point where the server running the lower version joins the group, stop all writes on the other servers in the group.

Without these precautions, the server running the lower version is likely to experience difficulties and terminate with an error.

- `group_replication_auto_increment_increment`

Property	Value
Command-Line Format	`--group-replication-auto-increment-increment=#`
System Variable	`group_replication_auto_increment_increment`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`7`
Minimum Value	`1`
Maximum Value	`65535`

Determines the interval between successive column values for transactions that execute on this server instance. This system variable should have the same value on all group members. When Group Replication is started on a server, the value of the server system variable `auto_increment_increment` is changed to this value, and the value of the server system variable `auto_increment_offset` is changed to the server ID. These settings avoid the selection of duplicate auto-increment values for writes on group members, which causes rollback of transactions. The

changes are reverted when Group Replication is stopped. These changes are only made and reverted if `auto_increment_increment` and `auto_increment_offset` each have their default value of 1. If their values have already been modified from the default, Group Replication does not alter them. From MySQL 8.0, the system variables are also not modified when Group Replication is in single-primary mode, where only one server writes.

The default value of 7 represents a balance between the number of usable values and the permitted maximum size of a replication group (9 members). If your group has more or fewer members, you can set this system variable to match the expected number of group members before Group Replication is started. You cannot change the setting while Group Replication is running.

- `group_replication_autorejoin_tries`

Property	Value
Command-Line Format	`--group-replication-autorejoin-tries=#`
Introduced	8.0.16
System Variable	`group_replication_autorejoin_tries`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2016`

Specifies the number of tries that a member makes to automatically rejoin the group if it is expelled, or if it is unable to contact a majority of the group before the `group_replication_unreachable_majority_timeout` setting is reached. The default setting, 0, means that the member does not try to rejoin, and proceeds to the action specified by the `group_replication_exit_state_action` system variable. You can specify a maximum of 2016 tries.

If you specify a number of tries, when the member's expulsion or unreachable majority timeout is reached, it makes an attempt to rejoin (using the current plugin option values), then continues to make further auto-rejoin attempts up to the specified number of tries. After an unsuccessful auto-rejoin attempt, the member waits 5 minutes before the next try. During the auto-rejoin procedure, the member remains in super read only mode and displays an `ERROR` state on its view of the replication group. The member can be stopped manually at any time by using a `STOP GROUP_REPLICATION` statement or shutting down the server. If the specified number of tries is exhausted without the member rejoining or being stopped, the member proceeds to the action specified by the `group_replication_exit_state_action` system variable.

For more information on using this option, see Section 18.6.6.3, “Auto-Rejoin”.

- `group_replication_bootstrap_group`

Property	Value
Command-Line Format	`--group-replication-bootstrap-group[={OFF|ON}]`
System Variable	`group_replication_bootstrap_group`
Scope	Global

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Configure this server to bootstrap the group. This option must only be set on one server and only when starting the group for the first time or restarting the entire group. After the group has been bootstrapped, set this option to `OFF`. It should be set to `OFF` both dynamically and in the configuration files. Starting two servers or restarting one server with this option set while the group is running may lead to an artificial split brain situation, where two independent groups with the same name are bootstrapped.

- `group_replication_clone_threshold`

Property	Value
Command-Line Format	`--group-replication-clone-threshold=#`
Introduced	8.0.17
System Variable	`group_replication_clone_threshold`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`9223372036854775807`
Minimum Value	`1`
Maximum Value	`9223372036854775807`

The transaction gap, as a number of transactions, between the existing member (donor) and the joining member (recipient) that triggers the use of a remote cloning operation for state transfer to the joining member during the distributed recovery process. If the transaction gap between the joining member and a suitable donor exceeds the threshold, Group Replication begins distributed recovery with a remote cloning operation. If the transaction gap is below the threshold, or if the remote cloning operation is not technically possible, Group Replication proceeds directly to state transfer from a donor's binary log.

Warning

Do not use a low setting for `group_replication_clone_threshold` in an active group. If a number of transactions above the threshold takes place in the group while the remote cloning operation is in progress, the joining member triggers a remote cloning operation again after restarting, and could continue this indefinitely. To avoid this situation, ensure that you set the threshold to a number higher than the number of transactions that you would expect to occur in the group during the time taken for the remote cloning operation.

To use this function, both the donor and the joining member must be set up beforehand to support cloning. For instructions, see Section 18.4.3.1, “Cloning for Distributed Recovery”. When a remote cloning operation is carried out, Group Replication manages it for you, including the required server restart, provided that `group_replication_start_on_boot=ON` is set. If not, you must restart the server manually. The remote cloning operation replaces the existing data dictionary on the joining member, but Group Replication checks and does not proceed if the joining member has additional

transactions that are not present on the other group members, because these transactions would be erased by the cloning operation.

The default setting (which is the maximum permitted sequence number for a transaction in a GTID) means that state transfer from a donor's binary log will virtually always be attempted rather than cloning. However, note that Group Replication always attempts to execute a cloning operation, regardless of your threshold, if state transfer from a donor's binary log is impossible, for example because the transactions needed by the joining member are not available in the binary logs on any existing group member. If you do not want to use cloning at all in your replication group, do not install the clone plugin on the members.

- `group_replication_communication_debug_options`

Property	Value
Command-Line Format	`--group-replication-communication-debug-options=value`
Introduced	8.0.3
System Variable	`group_replication_communication_debug_options`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`GCS_DEBUG_NONE`

Valid Values	`GCS_DEBUG_NONE`
	`GCS_DEBUG_BASIC`
	`GCS_DEBUG_TRACE`
	`XCOM_DEBUG_BASIC`
	`XCOM_DEBUG_TRACE`
	`GCS_DEBUG_ALL`

Configures the level of debugging messages to provide for the different Group Replication components, such as Group Communication System (GCS) and the group communication engine (XCom, a Paxos variant). The debug information is stored in the `GCS_DEBUG_TRACE` file in the data directory.

The set of available options, specified as strings, can be combined. The following options are available:

- `GCS_DEBUG_NONE` disables all debugging levels for both GCS and XCOM
- `GCS_DEBUG_BASIC` enables basic debugging information in GCS
- `GCS_DEBUG_TRACE` enables trace information in GCS
- `XCOM_DEBUG_BASIC` enables basic debugging information in XCOM
- `XCOM_DEBUG_TRACE` enables trace information in XCOM
- `GCS_DEBUG_ALL` enables all debugging levels for both GCS and XCOM

Setting the debug level to `GCS_DEBUG_NONE` only has an effect when provided without any other option. Setting the debug level to `GCS_DEBUG_ALL` overrides all other options.

- `group_replication_communication_max_message_size`

Property	Value
Command-Line Format	`--group-replication-communication-max-message-size=#`
Introduced	8.0.16
System Variable	`group_replication_communication_max_message_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10485760`
Minimum Value	`0`
Maximum Value	`1073741824`

Specifies a maximum message size for Group Replication communications. Messages greater than this size are automatically split into fragments that are sent separately and reassembled by the recipients. For more information, see Section 18.6.4, “Message Fragmentation”.

A maximum message size of 10485760 bytes (10 MiB) is set by default, which means that fragmentation is used by default in releases from MySQL 8.0.16. The greatest permitted value is the same as the maximum value of the `slave_max_allowed_packet` system variable, which is 1073741824 bytes (1 GB). The setting for `group_replication_communication_max_message_size` must be less than the `slave_max_allowed_packet` setting, because the applier thread cannot handle message fragments larger than `slave_max_allowed_packet`. To switch off fragmentation, specify a zero value for `group_replication_communication_max_message_size`. The value of `group_replication_communication_max_message_size` should be the same on all group members.

In order for members of a replication group to use fragmentation, the group's communication protocol version must be MySQL 8.0.16 or above. Use the `group_replication_get_communication_protocol()` UDF to view the group's communication protocol version. If a lower version is in use, group members do not fragment messages. You can use the `group_replication_set_communication_protocol()` UDF to set the group's communication protocol to a higher version if all group members support it. For more information, see Section 18.4.1.4, “Setting a Group's Communication Protocol Version”.

- `group_replication_components_stop_timeout`

Property	Value
Command-Line Format	`--group-replication-components-stop-timeout=#`
System Variable	`group_replication_components_stop_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`31536000`
Minimum Value	`2`

Property	Value
Maximum Value	`31536000`

Timeout, in seconds, that Group Replication waits for each of the components when shutting down.

- `group_replication_compression_threshold`

Property	Value
Command-Line Format	`--group-replication-compression-threshold=#`
System Variable	`group_replication_compression_threshold`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1000000`
Minimum Value	`0`
Maximum Value	`4294967295`

The value in bytes above which (LZ4) compression is enforced. When set to zero, deactivates compression. The value of `group_replication_compression_threshold` should be the same on all group members.

- `group_replication_consistency`

Property	Value
Command-Line Format	`--group-replication-consistency=value`
Introduced	8.0.14
System Variable	`group_replication_consistency`
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`EVENTUAL`

Valid Values	`EVENTUAL`
	`BEFORE_ON_PRIMARY_FAILOVER`
	`BEFORE`
	`AFTER`
	`BEFORE_AND_AFTER`

Controls the transaction consistency guarantee which a group provides. You can configure the consistency globally or per transaction. Also configures the fencing mechanism used by newly elected primaries in single primary groups. The effect of the variable must be considered for both read only (RO)

and read write (RW) transactions. The following list shows the possible values of this variable, in order of increasing transaction consistency guarantee:

- `EVENTUAL`

  Both RO and RW transactions do not wait for preceding transactions to be applied before executing. This was the behavior of Group Replication before this variable was added. A RW transaction does not wait for other members to apply a transaction. This means that a transaction could be externalized on one member before the others. This also means that in the event of a primary failover, the new primary can accept new RO and RW transactions before the previous primary transactions are all applied. RO transactions could result in outdated values, RW transactions could result in a rollback due to conflicts.

- `BEFORE_ON_PRIMARY_FAILOVER`

  New RO or RW transactions with a newly elected primary that is applying backlog from the old primary are held (not applied) until any backlog has been applied. This ensures that when a primary failover happens, intentionally or not, clients always see the latest value on the primary. This guarantees consistency, but means that clients must be able to handle the delay in the event that a backlog is being applied. Usually this delay should be minimal, but does depend on the size of the backlog.

- `BEFORE`

  A RW transaction waits for all preceding transactions to complete before being applied. A RO transaction waits for all preceding transactions to complete before being executed. This ensures that this transaction reads the latest value by only affecting the latency of the transaction. This reduces the overhead of synchronization on every RW transaction, by ensuring synchronization is used only on RO transactions. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

- `AFTER`

  A RW transaction waits until its changes have been applied to all of the other members. This value has no effect on RO transactions. This mode ensures that when a transaction is committed on the local member, any subsequent transaction reads the written value or a more recent value on any group member. Use this mode with a group that is used for predominantly RO operations to ensure that applied RW transactions are applied everywhere once they commit. This could be used by your application to ensure that subsequent reads fetch the latest data which includes the latest writes. This reduces the overhead of synchronization on every RO transaction, by ensuring synchronization is used only on RW transactions. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

- `BEFORE_AND_AFTER`

  A RW transaction waits for 1) all preceding transactions to complete before being applied and 2) until its changes have been applied on other members. A RO transaction waits for all preceding transactions to complete before execution takes place. This consistency level also includes the consistency guarantees provided by `BEFORE_ON_PRIMARY_FAILOVER`.

The `GROUP_REPLICATION_ADMIN` privilege is required to change the global setting for this system variable. For more information, see Section 18.4.2, “Transaction Consistency Guarantees”.

- `group_replication_enforce_update_everywhere_checks`

Property	Value
Command-Line Format	`--group-replication-enforce-update-everywhere-checks[={OFF|ON}]`
System Variable	`group_replication_enforce_update_everywhere_checks`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Enable or disable strict consistency checks for multi-primary update everywhere. The default is that checks are disabled. In single-primary mode, this option must be disabled on all group members. In multi-primary mode, when this option is enabled, statements are checked as follows to ensure they are compatible with multi-primary mode:

- If a transaction is executed under the `SERIALIZABLE` isolation level, then its commit fails when synchronizing itself with the group.
- If a transaction executes against a table that has foreign keys with cascading constraints, then the transaction fails to commit when synchronizing itself with the group.

This system variable is a group-wide configuration setting. It must have the same value on all group members, cannot be changed while Group Replication is running, and requires a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) in order for the value change to take effect. From MySQL 8.0.16, you can use the `group_replication_switch_to_single_primary_mode()` and `group_replication_switch_to_multi_primary_mode()` UDFs to change the value of this system variable while the group is still running. For more information, see Section 18.4.1.2, “Changing a Group's Mode”.

- `group_replication_exit_state_action`

Property	Value
Command-Line Format	`--group-replication-exit-state-action=value`
Introduced	8.0.12
System Variable	`group_replication_exit_state_action`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value (>= 8.0.16)	`READ_ONLY`
Default Value (>= 8.0.12, <= 8.0.15)	`ABORT_SERVER`
Valid Values (>= 8.0.18)	`ABORT_SERVER` `OFFLINE_MODE`

Property	Value
	READ_ONLY
Valid Values (>= 8.0.12, <= 8.0.17)	ABORT_SERVER READ_ONLY

Configures how Group Replication behaves when a server instance leaves the group unintentionally, for example after encountering an applier error, or in the case of a loss of majority, or when another member of the group expels it due to a suspicion timing out. The timeout period for a member to leave the group in the case of a loss of majority is set by the group_replication_unreachable_majority_timeout system variable, and the timeout period for suspicions is set by the group_replication_member_expel_timeout system variable. Note that an expelled group member does not know that it was expelled until it reconnects to the group, so the specified action is only taken if the member manages to reconnect, or if the member raises a suspicion on itself and expels itself.

When a member is expelled due to a suspicion timing out or a loss of majority, if the member has the group_replication_autorejoin_tries system variable set to specify a number of auto-rejoin attempts, it first makes the specified number of attempts while in super read only mode, and then follows the action specified by group_replication_exit_state_action. Auto-rejoin attempts are not made in case of an applier error, because these are not recoverable.

When group_replication_exit_state_action is set to READ_ONLY, if the member exits the group unintentionally or exhausts its auto-rejoin attempts, the instance switches MySQL to super read only mode (by setting the system variable super_read_only to ON). The READ_ONLY exit action was the behavior for MySQL 8.0 releases before the system variable was introduced, and became the default again from MySQL 8.0.16.

When group_replication_exit_state_action is set to OFFLINE_MODE, if the member exits the group unintentionally or exhausts its auto-rejoin attempts, the instance switches MySQL to offline mode (by setting the system variable offline_mode to ON). In this mode, connected client users are disconnected on their next request and connections are no longer accepted, with the exception of client users that have the CONNECTION_ADMIN or SUPER privilege. Group Replication also sets the system variable super_read_only to ON, so clients cannot make any updates, even if they have connected with the SUPER privilege. The OFFLINE_MODE exit action is available from MySQL 8.0.18.

When group_replication_exit_state_action is set to ABORT_SERVER, if the member exits the group unintentionally or exhausts its auto-rejoin attempts, the instance shuts down MySQL. This setting was the default from MySQL 8.0.12, when the system variable was added, to MySQL 8.0.15 inclusive.

Important

If a failure occurs before the member has successfully joined the group, the specified exit action *is not taken*. This is the case if there is a failure during the local configuration check, or a mismatch between the configuration of the joining member and the configuration of the group. In these situations, the super_read_only system variable is left with its original value, connections continue to be accepted, and the server does not shut down MySQL. To ensure that the server cannot accept updates when Group Replication did not start, we therefore recommend that super_read_only=ON is set in the server's configuration file at startup, which Group Replication will change to OFF on primary members after it has been started successfully. This safeguard is particularly important when the server is configured to start Group

> Replication on server boot (`group_replication_start_on_boot=ON`), but it is also useful when Group Replication is started manually using a `START GROUP_REPLICATION` command.

For more information on using this option, and the full list of situations in which the exit action is taken, see Section 18.6.6.4, “Exit Action”.

- `group_replication_flow_control_applier_threshold`

Property	Value
Command-Line Format	`--group-replication-flow-control-applier-threshold=#`
System Variable	`group_replication_flow_control_applier_threshold`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`25000`
Minimum Value	`0`
Maximum Value	`2147483647`

Specifies the number of waiting transactions in the applier queue that trigger flow control. This variable can be changed without resetting Group Replication.

- `group_replication_flow_control_certifier_threshold`

Property	Value
Command-Line Format	`--group-replication-flow-control-certifier-threshold=#`
System Variable	`group_replication_flow_control_certifier_threshold`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`25000`
Minimum Value	`0`
Maximum Value	`2147483647`

Specifies the number of waiting transactions in the certifier queue that trigger flow control. This variable can be changed without resetting Group Replication.

- `group_replication_flow_control_hold_percent`

Property	Value
Command-Line Format	`--group-replication-flow-control-hold-percent=#`
Introduced	8.0.2
System Variable	`group_replication_flow_control_hold_percent`

Property	Value
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value	`100`

Defines what percentage of the group quota remains unused to allow a cluster under flow control to catch up on backlog. A value of 0 implies that no part of the quota is reserved for catching up on the work backlog.

- `group_replication_flow_control_max_commit_quota`

Property	Value
Command-Line Format	`--group-replication-flow-control-max-commit-quota=#`
Introduced	8.0.2
System Variable	`group_replication_flow_control_max_commit_quota`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2147483647`

Defines the maximum flow control quota of the group, or the maximum available quota for any period while flow control is enabled. A value of 0 implies that there is no maximum quota set. Cannot be smaller than `group_replication_flow_control_min_quota` and `group_replication_flow_control_min_recovery_quota`.

- `group_replication_flow_control_member_quota_percent`

Property	Value
Command-Line Format	`--group-replication-flow-control-member-quota-percent=#`
Introduced	8.0.2
System Variable	`group_replication_flow_control_member_quota_percent`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`

Property	Value
Minimum Value	`0`
Maximum Value	`100`

Defines the percentage of the quota that a member should assume is available for itself when calculating the quotas. A value of 0 implies that the quota should be split equally between members that were writers in the last period.

- `group_replication_flow_control_min_quota`

Property	Value
Command-Line Format	`--group-replication-flow-control-min-quota=#`
Introduced	8.0.2
System Variable	`group_replication_flow_control_min_quota`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2147483647`

Controls the lowest flow control quota that can be assigned to a member, independently of the calculated minimum quota executed in the last period. A value of 0 implies that there is no minimum quota. Cannot be larger than `group_replication_flow_control_max_commit_quota`.

- `group_replication_flow_control_min_recovery_quota`

Property	Value
Command-Line Format	`--group-replication-flow-control-min-recovery-quota=#`
Introduced	8.0.2
System Variable	`group_replication_flow_control_min_recovery_quota`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2147483647`

Controls the lowest quota that can be assigned to a member because of another recovering member in the group, independently of the calculated minimum quota executed in the last period. A value of 0 implies that there is no minimum quota. Cannot be larger than `group_replication_flow_control_max_commit_quota`.

- `group_replication_flow_control_mode`

Property	Value
Command-Line Format	`--group-replication-flow-control-mode=value`
System Variable	`group_replication_flow_control_mode`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`QUOTA`
Valid Values	`DISABLED` `QUOTA`

Specifies the mode used for flow control. This variable can be changed without resetting Group Replication.

- `group_replication_flow_control_period`

Property	Value
Command-Line Format	`--group-replication-flow-control-period=#`
Introduced	8.0.2
System Variable	`group_replication_flow_control_period`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`60`

Defines how many seconds to wait between flow control iterations, in which flow control messages are sent and flow control management tasks are run.

- `group_replication_flow_control_release_percent`

Property	Value
Command-Line Format	`--group-replication-flow-control-release-percent=#`
Introduced	8.0.2
System Variable	`group_replication_flow_control_release_percent`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`50`

Property	Value
Minimum Value	`0`
Maximum Value	`1000`

Defines how the group quota should be released when flow control no longer needs to throttle the writer members, with this percentage being the quota increase per flow control period. A value of 0 implies that once the flow control thresholds are within limits the quota is released in a single flow control iteration. The range allows the quota to be released at up to 10 times current quota, as that allows a greater degree of adaptation, mainly when the flow control period is large and the quotas are very small.

- `group_replication_force_members`

Property	Value
Command-Line Format	`--group-replication-force-members=value`
System Variable	`group_replication_force_members`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

A list of peer addresses as a comma separated list such as `host1:port1`,`host2:port2`. This option is used to force a new group membership, in which the excluded members do not receive a new view and are blocked.

You must specify the address or host name and port as they are given in the `group_replication_local_address` option for each member. An IPv6 address must be specified in square brackets. For example:

```
"198.51.100.44:33061,[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061,example.org:33061"
```

The group communication engine for Group Replication (XCom) checks that the supplied IP addresses are in a valid format, and checks that you have not included any group members that are currently unreachable. Otherwise, the new configuration is not validated, so you must be careful to include only online servers that are reachable members of the group. Any incorrect values or invalid host names in the list could cause the group to be blocked with an invalid configuration.

It is important before forcing a new membership configuration to ensure that the servers to be excluded have been shut down. If they are not, shut them down before proceeding. Group members that are still online can automatically form new configurations, and if this has already taken place, forcing a further new configuration could create an artificial split-brain situation for the group.

After you have used the `group_replication_force_members` system variable to successfully force a new group membership and unblock the group, ensure that you clear the system variable. `group_replication_force_members` must be empty in order to issue a `START GROUP_REPLICATION` statement.

For details of the procedure to follow, see Section 18.4.4, “Network Partitioning”.

- `group_replication_group_name`

Property	Value
Command-Line Format	`--group-replication-group-name=value`

Property	Value
System Variable	`group_replication_group_name`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The name of the group which this server instance belongs to. Must be a valid UUID. This UUID is used internally when setting GTIDs for Group Replication events in the binary log.

Important

A unique UUID must be used.

- `group_replication_group_seeds`

Property	Value
Command-Line Format	`--group-replication-group-seeds=value`
System Variable	`group_replication_group_seeds`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

A list of group members that provide a member which joins the group with the data required for the joining member to gain synchrony with the group. The list consists of a single internal network address or host name for each included seed member, as configured in the seed member's `group_replication_local_address` system variable (not the seed member's SQL hostname and port). The addresses of the seed members are specified as a comma separated list, such as `host1:port1`,`host2:port2`. An IPv6 address must be specified in square brackets. For example:

```
group_replication_group_seeds= "198.51.100.44:33061,[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061, example.or
```

Note that the value you specify for this variable is not validated until a `START GROUP_REPLICATION` statement is issued and the Group Communication System (GCS) is available.

Usually this list consists of all members of the group, but you can choose a subset of the group members to be seeds. The list must contain at least one valid member address. Each address is validated when starting Group Replication. If the list does not contain any valid member addresses, issuing `START GROUP_REPLICATION` fails.

When a server is joining a replication group, it attempts to connect to the first seed member listed in its `group_replication_group_seeds` system variable. If the connection is refused, the joining member tries to connect to each of the other seed members in the list in order. If the joining member connects to a seed member but does not get added to the replication group as a result (for example, because the seed member does not have the joining member's address in its whitelist and closes the connection), the joining member continues to try the remaining seed members in the list in order.

A joining member must communicate with the seed member using the same protocol (IPv4 or IPv6) that the seed member advertises in the `group_replication_group_seeds` option. For the purpose of IP address whitelisting for Group Replication, the whitelist on the seed member must include an IP address for the joining member for the protocol offered by the seed member, or a host name that resolves to an address for that protocol. This address or host name must be set up and whitelisted in addition to the joining member's `group_replication_local_address` if the protocol for that address does not match the seed member's advertised protocol. If a joining member does not have a whitelisted address for the appropriate protocol, its connection attempt is refused. For more information, see Section 18.5.1, "Group Replication IP Address Whitelisting".

- `group_replication_gtid_assignment_block_size`

Property	Value
Command-Line Format	`--group-replication-gtid-assignment-block-size=#`
System Variable	`group_replication_gtid_assignment_block_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`1000000`
Minimum Value	`1`
Maximum Value (64-bit platforms)	`9223372036854775807`
Maximum Value (32-bit platforms)	`4294967295`

The number of consecutive GTIDs that are reserved for each member. Each member consumes its blocks and reserves more when needed.

This system variable is a group-wide configuration setting. It must have the same value on all group members, cannot be changed while Group Replication is running, and requires a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) in order for the value change to take effect.

- `group_replication_ip_whitelist`

Property	Value
Command-Line Format	`--group-replication-ip-whitelist=value`
System Variable	`group_replication_ip_whitelist`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String
Default Value	`AUTOMATIC`

Specifies which hosts are permitted to connect to the group. The address that you specify for each group member in `group_replication_local_address` must be whitelisted on the other servers

in the replication group. Note that the value you specify for this variable is not validated until a `START GROUP_REPLICATION` statement is issued and the Group Communication System (GCS) is available.

By default, this system variable is set to `AUTOMATIC`, which permits connections from private subnetworks active on the host. The group communication engine for Group Replication (XCom) automatically scans active interfaces on the host, and identifies those with addresses on private subnetworks. These addresses and the `localhost` IP address for IPv4 and (from MySQL 8.0.14) IPv6 are used to create the Group Replication whitelist. For a list of the ranges from which addresses are automatically whitelisted, see Section 18.5.1, “Group Replication IP Address Whitelisting”.

The automatic whitelist of private addresses cannot be used for connections from servers outside the private network. For Group Replication connections between server instances that are on different machines, you must provide public IP addresses and specify these as an explicit whitelist. If you specify any entries for the whitelist, the private and `localhost` addresses are not added automatically, so if you use any of these, you must specify them explicitly.

As the value of the `group_replication_ip_whitelist` option, you can specify any combination of the following:

- IPv4 addresses (for example, `198.51.100.44`)
- IPv4 addresses with CIDR notation (for example, `192.0.2.21/24`)
- IPv6 addresses, from MySQL 8.0.14 (for example, `2001:db8:85a3:8d3:1319:8a2e:370:7348`)
- IPv6 addresses with CIDR notation, from MySQL 8.0.14 (for example, `2001:db8:85a3:8d3::/64`)
- Host names (for example, `example.org`)
- Host names with CIDR notation (for example, `www.example.com/24`)

Before MySQL 8.0.14, host names could only resolve to IPv4 addresses. From MySQL 8.0.14, host names can resolve to IPv4 addresses, IPv6 addresses, or both. If a host name resolves to both an IPv4 and an IPv6 address, the IPv4 address is always used for Group Replication connections. You can use CIDR notation in combination with host names or IP addresses to whitelist a block of IP addresses with a particular network prefix, but do ensure that all the IP addresses in the specified subnet are under your control.

A comma must separate each entry in the whitelist. For example:

```
"192.0.2.21/24,198.51.100.44,203.0.113.0/24,2001:db8:85a3:8d3:1319:8a2e:370:7348,example.org,www.example.com
```

If any of the seed members for the group are listed in the `group_replication_group_seeds` option with an IPv6 address when a joining member has an IPv4 `group_replication_local_address`, or the reverse, you must also set up and whitelist an alternative address for the joining member for the protocol offered by the seed member (or a host name that resolves to an address for that protocol). For more information, see Section 18.5.1, “Group Replication IP Address Whitelisting”.

It is possible to configure different whitelists on different group members according to your security requirements, for example, in order to keep different subnets separate. However, this can cause issues when a group is reconfigured. If you do not have a specific security requirement to do otherwise, use the same whitelist on all members of a group. For more details, see Section 18.5.1, “Group Replication IP Address Whitelisting”.

For host names, name resolution takes place only when a connection request is made by another server. A host name that cannot be resolved is not considered for whitelist validation, and a warning

message is written to the error log. Forward-confirmed reverse DNS (FCrDNS) verification is carried out for resolved host names.

Warning

Host names are inherently less secure than IP addresses in a whitelist. FCrDNS verification provides a good level of protection, but can be compromised by certain types of attack. Specify host names in your whitelist only when strictly necessary, and ensure that all components used for name resolution, such as DNS servers, are maintained under your control. You can also implement name resolution locally using the hosts file, to avoid the use of external components.

- `group_replication_local_address`

Property	Value
Command-Line Format	`--group-replication-local-address=value`
System Variable	`group_replication_local_address`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The network address which the member provides for connections from other members, specified as a `host:port` formatted string. This address must be reachable by all members of the group because it is used by the group communication engine for Group Replication (XCom, a Paxos variant) for TCP communication between remote XCom instances. Communication with the local instance is over an input channel using shared memory.

Warning

Do not use this address for communication with the member. This is not the MySQL server SQL protocol host and port.

The address or host name that you specify in `group_replication_local_address` is used by Group Replication as the unique identifier for a group member within the replication group. You can use the same port for all members of a replication group as long as the host names or IP addresses are all different, and you can use the same host name or IP address for all members as long as the ports are all different. The recommended port for `group_replication_local_address` is 33061. Note that the value you specify for this variable is not validated until the `START GROUP_REPLICATION` statement is issued and the Group Communication System (GCS) is available.

The network address configured by `group_replication_local_address` must be resolvable by all group members. For example, if each server instance is on a different machine with a fixed network address, you could use the IP address of the machine, such as 10.0.0.1. If you use a host name, you must use a fully qualified name, and ensure it is resolvable through DNS, correctly configured `/etc/hosts` files, or other name resolution processes. From MySQL 8.0.14, IPv6 addresses (or host names that resolve to them) can be used as well as IPv4 addresses. An IPv6 address must be specified in square brackets in order to distinguish the port number, for example:

```
group_replication_local_address= "[2001:db8:85a3:8d3:1319:8a2e:370:7348]:33061"
```

If a host name specified as the Group Replication local address for a server instance resolves to both an IPv4 and an IPv6 address, the IPv4 address is always used for Group Replication connections. For

more information on Group Replication support for IPv6 networks and on replication groups with a mix of members using IPv4 and members using IPv6, see Section 18.4.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”.

For the purpose of IP address whitelisting for Group Replication, the address that you specify for each group member in `group_replication_local_address` must be added to the list for the `group_replication_ip_whitelist` option on the other servers in the replication group. If any of the seed members for the group are listed in the `group_replication_group_seeds` option with an IPv6 address when this member has an IPv4 `group_replication_local_address`, or the reverse, you must also set up and whitelist an alternative address for this member for the required protocol (or a host name that resolves to an address for that protocol). For more information, see Section 18.5.1, “Group Replication IP Address Whitelisting”.

- `group_replication_member_weight`

Property	Value
Command-Line Format	`--group-replication-member-weight=#`
Introduced	8.0.2
System Variable	`group_replication_member_weight`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`50`
Minimum Value	`0`
Maximum Value	`100`

A percentage weight that can be assigned to members to influence the chance of the member being elected as primary in the event of failover, for example when the existing primary leaves a single-primary group. Assign numeric weights to members to ensure that specific members are elected, for example during scheduled maintenance of the primary or to ensure certain hardware is prioritized in the event of failover.

For a group with members configured as follows:

- `member-1`: group_replication_member_weight=30, server_uuid=aaaa
- `member-2`: group_replication_member_weight=40, server_uuid=bbbb
- `member-3`: group_replication_member_weight=40, server_uuid=cccc
- `member-4`: group_replication_member_weight=40, server_uuid=dddd

during election of a new primary the members above would be sorted as `member-2`, `member-3`, `member-4`, and `member-1`. This results in `member`-2 being chosen as the new primary in the event of failover. For more information, see Section 18.1.3.1, “Single-Primary Mode”.

- `group_replication_member_expel_timeout`

Property	Value
Command-Line Format	`--group-replication-member-expel-timeout=#`

Property	Value
Introduced	8.0.13
System Variable	`group_replication_member_expel_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value (>= 8.0.14)	`3600`
Maximum Value (<= 8.0.13)	`31536000`

The period of time in seconds that a Group Replication group member waits after creating a suspicion before expelling from the group the member suspected of having failed. The initial 5-second detection period before a suspicion is created does not count as part of this time. Changing the value of `group_replication_member_expel_timeout` on a group member takes effect immediately for existing as well as future suspicions on that group member. It is not mandatory for all members of a group to have the same setting, but it is recommended in order to avoid unexpected expulsions.

By default, `group_replication_member_expel_timeout` is set to 0, meaning that there is no waiting period and a suspected member is liable for expulsion immediately after the 5-second detection period ends. To avoid unnecessary expulsions on slower networks, or in the case of expected transient network failures or machine slowdowns, you can specify a timeout value greater than zero, up to a maximum of 3600 seconds (1 hour). If a suspect member becomes active again before the suspicion times out, it rejoins the group, applies all the messages that were buffered by the remaining group members, and enters `ONLINE` state.

If the timeout is exceeded, the suspect member is liable for expulsion immediately after the suspicion times out. If the member is able to resume communications and receives a view where it is expelled, and the member has the `group_replication_autorejoin_tries` system variable set to specify a number of auto-rejoin attempts, it proceeds to make the specified number of attempts to rejoin the group while in super read only mode. If the member does not have any auto-rejoin attempts specified, or if it has exhausted the specified number of attempts, it follows the action specified by the system variable `group_replication_exit_state_action`.

For more information on using this option, see Section 18.6.6.1, “Expel Timeout”. For alternative mitigation strategies to avoid unnecessary expulsions where this system variable is not available, see Section 18.9.2, “Group Replication Limitations”.

- `group_replication_message_cache_size`

Property	Value
Command-Line Format	`--group-replication-message-cache-size=#`
Introduced	8.0.16
System Variable	`group_replication_message_cache_size`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	`1073741824 (1 GB)`
Minimum Value	`1073741824 (1 GB)`
Maximum Value (64-bit platforms)	`18446744073709551615 (16 EiB)`
Maximum Value (32-bit platforms)	`315360004294967295 (4 GB)`

The maximum amount of memory that is available for the message cache in the group communication engine for Group Replication (XCom), which holds messages (and their metadata) that are exchanged between the group members as a part of the consensus protocol. Among other functions, the message cache is used for recovery by members that return to the group after a period where they were unable to communicate with the other group members. The `group_replication_member_expel_timeout` system variable determines the time (up to an hour) allowed for members to return to the group rather than being expelled. The size of the message cache should be set with reference to the expected volume of messages in this time period, so that it contains all the missed messages required for members to return successfully.

`group_replication_message_cache_size` has a default and minimum setting of 1073741824 bytes (1 GB). The same cache size limit should be set on all group members, because an unreachable member that is attempting to reconnect selects any other member at random for recovery of missed messages. Ensure that sufficient memory is available on your system for your chosen cache size limit, considering the size of MySQL Server's other caches and object pools.

The cache size limit can be increased or reduced dynamically at runtime. If you reduce the cache size limit, XCom removes the oldest entries that have been decided and delivered until the current size is below the limit. Group Replication's Group Communication System (GCS) alerts you, by a warning message, when a message that is likely to be needed for recovery by a member that is currently unreachable is removed from the message cache. For more information on tuning the message cache size, see Section 18.6.5, “XCom Cache Management”.

- `group_replication_poll_spin_loops`

Property	Value
Command-Line Format	`--group-replication-poll-spin-loops=#`
System Variable	`group_replication_poll_spin_loops`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The number of times the group communication thread waits for the communication engine mutex to be released before the thread waits for more incoming network messages.

- `group_replication_recovery_complete_at`

Property	Value
Command-Line Format	`--group-replication-recovery-complete-at=value`
System Variable	`group_replication_recovery_complete_at`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`TRANSACTIONS_APPLIED`

Valid Values	`TRANSACTIONS_CERTIFIED`
	
	`TRANSACTIONS_APPLIED`

The policy applied during the distributed recovery process when handling cached transactions after state transfer. This option specifies whether a member is marked online after it has received all transactions that it missed before it joined the group (`TRANSACTIONS_CERTIFIED`) or after it has received and applied them (`TRANSACTIONS_APPLIED`).

- `group_replication_recovery_compression_algorithm`

Property	Value
Command-Line Format	`--group-replication-recovery-compression-algorithm=value`
Introduced	8.0.18
System Variable	`group_replication_recovery_compression_algorithm`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Set
Default Value	`uncompressed`

Valid Values	`zlib`
	
	`zstd`
	
	`uncompressed`

The compression algorithms permitted for Group Replication distributed recovery connections for state transfer from a donor's binary log. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. For more information, see Section 4.2.6, “Connection Compression Control”.

This setting does not apply if the server has been set up to support cloning (see Section 18.4.3.1, “Cloning for Distributed Recovery”) and a remote cloning operation is used during distributed recovery. For this method of state transfer, the clone plugin's `clone_enable_compression` setting applies.

- `group_replication_recovery_get_public_key`

Property	Value
Command-Line Format	`--group-replication-recovery-get-public-key[={OFF|ON}]`
Introduced	8.0.4
System Variable	`group_replication_recovery_get_public_key`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether to request from the master the public key required for RSA key pair-based password exchange. If `group_replication_recovery_public_key_path` is set to a valid public key file, it takes precedence over `group_replication_recovery_get_public_key`. This variable applies if you are not using SSL for distributed recovery over the `group_replication_recovery` channel (`group_replication_recovery_use_ssl=ON`), and the replication user account for Group Replication authenticates with the `caching_sha2_password` plugin (which is the default in MySQL 8.0). For more details, see Using Group Replication and the Caching SHA-2 User Credentials Plugin.

- `group_replication_recovery_public_key_path`

Property	Value
Command-Line Format	`--group-replication-recovery-public-key-path=file_name`
Introduced	8.0.4
System Variable	`group_replication_recovery_public_key_path`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name
Default Value	`NULL`

The path name to a file containing a slave-side copy of the public key required by the master for RSA key pair-based password exchange. The file must be in PEM format. If `group_replication_recovery_public_key_path` is set to a valid public key file, it takes precedence over `group_replication_recovery_get_public_key`. This variable applies if you are not using SSL for distributed recovery over the `group_replication_recovery` channel (so `group_replication_recovery_use_ssl` is set to `OFF`), and the replication user account for Group Replication authenticates with the `caching_sha2_password` plugin (which is the default in MySQL 8.0) or the `sha256_password` plugin. (For `sha256_password`, setting `group_replication_recovery_public_key_path` applies only if MySQL was built using OpenSSL.) For more details, see Using Group Replication and the Caching SHA-2 User Credentials Plugin.

- `group_replication_recovery_reconnect_interval`

Property	Value
Command-Line Format	`--group-replication-recovery-reconnect-interval=#`
System Variable	`group_replication_recovery_reconnect_interval`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`60`
Minimum Value	`0`
Maximum Value	`31536000`

The sleep time, in seconds, between reconnection attempts when no suitable donor was found in the group for distributed recovery.

- `group_replication_recovery_retry_count`

Property	Value
Command-Line Format	`--group-replication-recovery-retry-count=#`
System Variable	`group_replication_recovery_retry_count`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value	`31536000`

The number of times that the member that is joining tries to connect to the available donors for distributed recovery before giving up.

- `group_replication_recovery_ssl_ca`

Property	Value
Command-Line Format	`--group-replication-recovery-ssl-ca=value`
System Variable	`group_replication_recovery_ssl_ca`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The path to a file that contains a list of trusted SSL certificate authorities for distributed recovery connections. See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for information on configuring SSL for recovery.

If this server has been set up to support cloning (see Section 18.4.3.1, “Cloning for Distributed Recovery”), and you have set `group_replication_recovery_use_ssl` to `ON`, Group Replication automatically configures the setting for the clone SSL option `clone_ssl_ca` to match your setting for `group_replication_recovery_ssl_ca`.

- `group_replication_recovery_ssl_capath`

Property	Value
Command-Line Format	`--group-replication-recovery-ssl-capath=value`
System Variable	`group_replication_recovery_ssl_capath`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The path to a directory that contains trusted SSL certificate authority certificates for distributed recovery connections. See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for information on configuring SSL for distributed recovery.

- `group_replication_recovery_ssl_cert`

Property	Value
Command-Line Format	`--group-replication-recovery-ssl-cert=value`
System Variable	`group_replication_recovery_ssl_cert`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The name of the SSL certificate file to use for establishing a secure connection for distributed recovery. See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for information on configuring SSL for distributed recovery.

If this server has been set up to support cloning (see Section 18.4.3.1, “Cloning for Distributed Recovery”), and you have set `group_replication_recovery_use_ssl` to `ON`, Group Replication automatically configures the setting for the clone SSL option `clone_ssl_cert` to match your setting for `group_replication_recovery_ssl_cert`.

- `group_replication_recovery_ssl_cipher`

Property	Value
Command-Line Format	`--group-replication-recovery-ssl-cipher=value`
System Variable	`group_replication_recovery_ssl_cipher`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The list of permissible ciphers for SSL encryption. See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for information on configuring SSL for distributed recovery.

- `group_replication_recovery_ssl_crl`

Property	Value
Command-Line Format	`--group-replication-recovery-ssl-crl=value`
System Variable	`group_replication_recovery_ssl_crl`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	File name

The path to a directory that contains files containing certificate revocation lists. See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for information on configuring SSL for distributed recovery.

- `group_replication_recovery_ssl_crlpath`

Property	Value
Command-Line Format	`--group-replication-recovery-ssl-crlpath=value`
System Variable	`group_replication_recovery_ssl_crlpath`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Directory name

The path to a directory that contains files containing certificate revocation lists. See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for information on configuring SSL for distributed recovery.

- `group_replication_recovery_ssl_key`

Property	Value
Command-Line Format	`--group-replication-recovery-ssl-key=value`
System Variable	`group_replication_recovery_ssl_key`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

The name of the SSL key file to use for establishing a secure connection. See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for information on configuring SSL for distributed recovery.

If this server has been set up to support cloning (see Section 18.4.3.1, “Cloning for Distributed Recovery”), and you have set `group_replication_recovery_use_ssl` to `ON`, Group Replication

automatically configures the setting for the clone SSL option `clone_ssl_key` to match your setting for `group_replication_recovery_ssl_key`.

- `group_replication_recovery_ssl_verify_server_cert`

Property	Value
Command-Line Format	`--group-replication-recovery-ssl-verify-server-cert[={OFF|ON}]`
System Variable	`group_replication_recovery_ssl_verify_server_cert`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Make the distributed recovery connection check the server's Common Name value in the donor sent certificate. See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for information on configuring SSL for distributed recovery.

- `group_replication_recovery_use_ssl`

Property	Value
Command-Line Format	`--group-replication-recovery-use-ssl[={OFF|ON}]`
System Variable	`group_replication_recovery_use_ssl`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether Group Replication distributed recovery connections between group members should use SSL or not. See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for information on configuring SSL for distributed recovery.

If this server has been set up to support cloning (see Section 18.4.3.1, “Cloning for Distributed Recovery”), and you set this option to `ON`, Group Replication uses SSL for remote cloning operations as well as for state transfer from a donor's binary log. If you set this option to `OFF`, Group Replication does not use SSL for remote cloning operations.

- `group_replication_recovery_zstd_compression_level`

Property	Value
Command-Line Format	`--group-replication-recovery-zstd-compression-level=#`
Introduced	8.0.18
System Variable	`group_replication_recovery_zstd_compression_level`
Scope	Global
Dynamic	Yes

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	3
Minimum Value	1
Maximum Value	22

The compression level to use for Group Replication distributed recovery connections that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. The default `zstd` compression level is 3. For distributed recovery connections that do not use `zstd` compression, this variable has no effect.

For more information, see Section 4.2.6, “Connection Compression Control”.

- `group_replication_single_primary_mode`

Property	Value
Command-Line Format	`--group-replication-single-primary-mode[={OFF|ON}]`
System Variable	`group_replication_single_primary_mode`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Instructs the group to automatically pick a single server to be the one that handles read/write workload. This server is the PRIMARY and all others are SECONDARIES.

This system variable is a group-wide configuration setting. It must have the same value on all group members, cannot be changed while Group Replication is running, and requires a full reboot of the group (a bootstrap by a server with `group_replication_bootstrap_group=ON`) in order for the value change to take effect. From MySQL 8.0.16, you can use the `group_replication_switch_to_single_primary_mode()` and `group_replication_switch_to_multi_primary_mode()` UDFs to change the value of this system variable while the group is still running. For more information, see Section 18.4.1.2, “Changing a Group's Mode”.

- `group_replication_ssl_mode`

Property	Value
Command-Line Format	`--group-replication-ssl-mode=value`
System Variable	`group_replication_ssl_mode`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	`DISABLED`

Property	Value

Valid Values	`DISABLED`
	`REQUIRED`
	`VERIFY_CA`
	`VERIFY_IDENTITY`

Sets the security state of group communication connections between Group Replication members. The possible values are as follows:

DISABLED	Establish an unencrypted connection (the default).
REQUIRED	Establish a secure connection if the server supports secure connections.
VERIFY_CA	Like `REQUIRED`, but additionally verify the server TLS certificate against the configured Certificate Authority (CA) certificates.
VERIFY_IDENTITY	Like `VERIFY_CA`, but additionally verify that the server certificate matches the host to which the connection is attempted.

See Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support” for information on configuring SSL for group communication.

- `group_replication_start_on_boot`

Property	Value
Command-Line Format	`--group-replication-start-on-boot[={OFF|ON}]`
System Variable	`group_replication_start_on_boot`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether the server should start Group Replication or not during server start.

- `group_replication_transaction_size_limit`

Property	Value
Command-Line Format	`--group-replication-transaction-size-limit=#`
System Variable	`group_replication_transaction_size_limit`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.2)	`150000000`

Property	Value
Default Value (<= 8.0.1)	`0`
Minimum Value	`0`
Maximum Value	`2147483647`

Configures the maximum transaction size in bytes which the replication group accepts. Transactions larger than this size are rolled back by the receiving member and are not broadcast to the group. Large transactions can cause problems for a replication group in terms of memory allocation, which can cause the system to slow down, or in terms of network bandwidth consumption, which can cause a member to be suspected of having failed because it is busy processing the large transaction.

When this system variable is set to 0 there is no limit to the size of transactions the group accepts. From MySQL 8.0, the default setting for this system variable is 150000000 bytes (approximately 143 MB). Adjust the value of this system variable depending on the maximum message size that you need the group to tolerate, bearing in mind that the time taken to process a transaction is proportional to its size. The value of `group_replication_transaction_size_limit` should be the same on all group members. For further mitigation strategies for large transactions, see Section 18.9.2, “Group Replication Limitations”.

- `group_replication_unreachable_majority_timeout`

Property	Value
Command-Line Format	`--group-replication-unreachable-majority-timeout=#`
Introduced	8.0.2
System Variable	`group_replication_unreachable_majority_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`31536000`

Specifies a number of seconds for which members that suffer a network partition and cannot connect to the majority wait before leaving the group. In a group of 5 servers (S1,S2,S3,S4,S5), if there is a disconnection between (S1,S2) and (S3,S4,S5) there is a network partition. The first group (S1,S2) is now in a minority because it cannot contact more than half of the group. While the majority group (S3,S4,S5) remains running, the minority group waits for the specified time for a network reconnection. For a detailed description of this scenario, see Section 18.4.4, “Network Partitioning”.

By default, `group_replication_unreachable_majority_timeout` is set to 0, which means that members that find themselves in a minority due to a network partition wait forever to leave the group. If you set a timeout, when the specified time elapses, all pending transactions processed by the minority are rolled back, and the servers in the minority partition move to the `ERROR` state. If a member has the `group_replication_autorejoin_tries` system variable set to specify a number of auto-rejoin attempts, it proceeds to make the specified number of attempts to rejoin the group while in super read only mode. If the member does not have any auto-rejoin attempts specified, or if it has exhausted the specified number of attempts, it follows the action specified by the system variable `group_replication_exit_state_action`.

Warning

When you have a symmetric group, with just two members for example (S0,S2), if there is a network partition and there is no majority, after the configured timeout all members enter the `ERROR` state.

For more information on using this option, see Section 18.6.6.2, “Unreachable Majority Timeout”.

Group Replication Status Variable

This section describes the status variables which provide information about Group Replication. The variable has the following meaning:

- `group_replication_primary_member`

  Shows the primary member's UUID when the group is operating in single-primary mode. If the group is operating in multi-primary mode, shows an empty string.

Warning

The `group_replication_primary_member` status variable has been deprecated and is scheduled to be removed in a future version.

See Finding the Primary.

18.9 Requirements and Limitations

This section lists and explains the requirements and limitations of Group Replication.

18.9.1 Group Replication Requirements

Server instances that you want to use for Group Replication must satisfy the following requirements.

Infrastructure

- **InnoDB Storage Engine.** Data must be stored in the `InnoDB` transactional storage engine. Transactions are executed optimistically and then, at commit time, are checked for conflicts. If there are conflicts, in order to maintain consistency across the group, some transactions are rolled back. This means that a transactional storage engine is required. Moreover, `InnoDB` provides some additional functionality that enables better management and handling of conflicts when operating together with Group Replication. The use of other storage engines, including the temporary `MEMORY` storage engine, might cause errors in Group Replication. You can prevent the use of other storage engines by setting the `disabled_storage_engines` system variable on group members, for example:

  ```
  disabled_storage_engines="MyISAM,BLACKHOLE,FEDERATED,ARCHIVE,MEMORY"
  ```

- **Primary Keys.** Every table that is to be replicated by the group must have a defined primary key, or primary key equivalent where the equivalent is a non-null unique key. Such keys are required as a unique identifier for every row within a table, enabling the system to determine which transactions conflict by identifying exactly which rows each transaction has modified.

- **Network Performance.** MySQL Group Replication is designed to be deployed in a cluster environment where server instances are very close to each other. The performance and stabiity of a

group can be impacted by both network latency and network bandwidth. Bi-directional communication must be maintained at all times between all group members. If either inbound or outbound communication is blocked for a server instance (for example, by a firewall, or by connectivity issues), the member cannot function in the group, and the group members (including the member with issues) might not be able to report the correct member status for the affected server instance.

From MySQL 8.0.14, you can use an IPv4 or IPv6 network infrastructure, or a mix of the two, for TCP communication between remote Group Replication servers. There is also nothing preventing Group Replication from operating over a virtual private network (VPN).

Also from MySQL 8.0.14, where Group Replication server instances are co-located and share a local group communication engine (XCom) instance, a dedicated input channel with lower overhead is used for communication where possible instead of the TCP socket. For certain Group Replication tasks that require communication between remote XCom instances, such as joining a group, the TCP network is still used, so network performance influences the group's performance.

Server Instance Configuration

The following options must be configured as shown on server instances that are members of a group.

- **Binary Log Active.** Set `--log-bin[=log_file_name]`. MySQL Group Replication replicates binary log contents, therefore the binary log needs to be on for it to operate. This option is enabled by default. See Section 5.4.4, “The Binary Log”.
- **Slave Updates Logged.** Set `--log-slave-updates`. This option is enabled by default. Group members need to log transactions that are received from their donors at joining time and applied through the replication applier, and to log all transactions that they receive and apply from the group. This enables Group Replication to carry out distributed recovery by state transfer from an existing group member's binary log.
- **Binary Log Row Format.** Set `--binlog-format=row`. Group Replication relies on row-based replication format to propagate changes consistently among the servers in the group. It relies on row-based infrastructure to be able to extract the necessary information to detect conflicts among transactions that execute concurrently in different servers in the group. See Section 17.2.1, “Replication Formats”.
- **Binary Log Checksums Off.** Set `--binlog-checksum=NONE`. Due to a design limitation of replication event checksums, Group Replication cannot make use of them, and they must be disabled.
- **Global Transaction Identifiers On.** Set `gtid_mode=ON`. Group Replication uses global transaction identifiers to track exactly which transactions have been committed on every server instance and thus be able to infer which servers have executed transactions that could conflict with already committed transactions elsewhere. In other words, explicit transaction identifiers are a fundamental part of the framework to be able to determine which transactions may conflict. See Section 17.1.3, “Replication with Global Transaction Identifiers”.
- **Replication Information Repositories.** Set `--master-info-repository=TABLE` and `--relay-log-info-repository=TABLE`. The replication applier needs to have the master information and relay log metadata written to the `mysql.slave_master_info` and `mysql.slave_relay_log_info` system tables. This ensures the Group Replication plugin has consistent recoverability and transactional management of the replication metadata. From MySQL 8.0.2, these options are set to `TABLE` by default, and from MySQL 8.0.3, the `FILE` setting is deprecated. See Section 17.2.4.2, “Slave Status Logs”.
- **Transaction Write Set Extraction.** Set `--transaction-write-set-extraction=XXHASH64` so that while collecting rows to log them to the binary log, the server collects the write set as well. The write set is based on the primary keys of each row and is a simplified and compact view of a tag that uniquely

identifies the row that was changed. This tag is then used for detecting conflicts. This option is enabled by default.

- **Default Table Encryption.** Set `--default-table-encryption` to the same value on all group members. Default schema and tablespace encryption can be either enabled (`ON`) or disabled (`OFF`, the default) as long as the setting is the same on all members.
- **Lower Case Table Names.** Set `--lower-case-table-names` to the same value on all group members. A setting of 1 is correct for the use of the `InnoDB` storage engine, which is required for Group Replication. Note that this setting is not the default on all platforms.
- **Multithreaded Appliers.** Group Replication members can be configured as multithreaded slaves, enabling transactions to be applied in parallel. A nonzero value for `slave_parallel_workers` enables the multithreaded applier on the member, and up to 1024 parallel applier threads can be specified. Setting `slave_preserve_commit_order=1` ensures that the final commit of parallel transactions is in the same order as the original transactions, as required for Group Replication, which relies on consistency mechanisms built around the guarantee that all participating members receive and apply committed transaction in the same order. Finally, the setting `slave_parallel_type=LOGICAL_CLOCK`, which specifies the policy used to decide which transactions are allowed to execute in parallel on the slave, is required with `slave_preserve_commit_order=1`. Setting `slave_parallel_workers=0` disables parallel execution and gives the slave a single applier thread and no coordinator thread. With that setting, the `slave_parallel_type` and `slave_preserve_commit_order` options have no effect and are ignored.

18.9.2 Group Replication Limitations

The following known limitations exist for Group Replication. Note that the limitations and issues described for multi-primary mode groups can also apply in single-primary mode clusters during a failover event, while the newly elected primary flushes out its applier queue from the old primary.

Tip

Group Replication is built on GTID based replication, therefore you should also be aware of Section 17.1.3.6, “Restrictions on Replication with GTIDs”.

- **`--upgrade=MINIMAL` option.** Group Replication cannot be started following a MySQL Server upgrade that uses the MINIMAL option (`--upgrade=MINIMAL`), which does not upgrade system tables on which the replication internals depend.
- **Gap Locks.** The certification process does not take into account gap locks, as information about gap locks is not available outside of `InnoDB`. See Gap Locks for more information.

Note

Unless you rely on `REPEATABLE READ` semantics in your applications, we recommend using the `READ COMMITTED` isolation level with Group Replication. InnoDB does not use gap locks in `READ COMMITTED`, which aligns the local conflict detection within InnoDB with the distributed conflict detection performed by Group Replication.

- **Table Locks and Named Locks.** The certification process does not take into account table locks (see Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”) or named locks (see `GET_LOCK()`).
- **Replication Event Checksums.** Due to a design limitation of replication event checksums, Group Replication cannot currently make use of them. Therefore set `--binlog-checksum=NONE`.

- **SERIALIZABLE Isolation Level.** `SERIALIZABLE` isolation level is not supported in multi-primary groups by default. Setting a transaction isolation level to `SERIALIZABLE` configures Group Replication to refuse to commit the transaction.

- **Concurrent DDL versus DML Operations.** Concurrent data definition statements and data manipulation statements executing against the same object but on different servers is not supported when using multi-primary mode. During execution of Data Definition Language (DDL) statements on an object, executing concurrent Data Manipulation Language (DML) on the same object but on a different server instance has the risk of conflicting DDL executing on different instances not being detected.

- **Foreign Keys with Cascading Constraints.** Multi-primary mode groups (members all configured with `group_replication_single_primary_mode=OFF`) do not support tables with multi-level foreign key dependencies, specifically tables that have defined `CASCADING` foreign key constraints. This is because foreign key constraints that result in cascading operations executed by a multi-primary mode group can result in undetected conflicts and lead to inconsistent data across the members of the group. Therefore we recommend setting `group_replication_enforce_update_everywhere_checks=ON` on server instances used in multi-primary mode groups to avoid undetected conflicts.

  In single-primary mode this is not a problem as it does not allow concurrent writes to multiple members of the group and thus there is no risk of undetected conflicts.

- **Multi-primary Mode Deadlock.** When a group is operating in multi-primary mode, `SELECT .. FOR UPDATE` statements can result in a deadlock. This is because the lock is not shared across the members of the group, therefore the expectation for such a statement might not be reached.

- **Replication Filters.** Global replication filters cannot be used on a MySQL server instance that is configured for Group Replication, because filtering transactions on some servers would make the group unable to reach agreement on a consistent state. Channel specific replication filters can be used on replication channels that are not directly involved with Group Replication, such as where a group member also acts as a replication slave to a master that is outside the group. They cannot be used on the `group_replication_applier` or `group_replication_recovery` channels.

- **Encrypted Connections.** Support for the TLSv1.3 protocol is available in MySQL Server as of MySQL 8.0.16, provided that MySQL was compiled using OpenSSL 1.1.1 or higher. Group Replication supports TLSv1.3 from MySQL 8.0.18. In MySQL 8.0.16 and MySQL 8.0.17, if the server supports TLSv1.3, the protocol is not supported in the group communication engine and cannot be used by Group Replication.

  If TLSv1.3 is used in Group Replication for distributed recovery, the Group Replication group members must permit the use of at least one TLSv1.3 ciphersuite that is enabled by default.

Limit on Group Size

The maximum number of MySQL servers that can be members of a single replication group is 9. If further members attempt to join the group, their request is refused. This limit has been identified from testing and benchmarking as a safe boundary where the group performs reliably on a stable local area network.

Limits on Transaction Size

If an individual transaction results in message contents which are large enough that the message cannot be copied between group members over the network within a 5-second window, members can be suspected of having failed, and then expelled, just because they are busy processing the transaction. Large transactions can also cause the system to slow due to problems with memory allocation. To avoid these issues use the following mitigations:

- If unnecessary expulsions occur due to large messages, use the system variable `group_replication_member_expel_timeout` to allow additional time before a member under suspicion of having failed is expelled. You can allow up to an hour after the initial 5-second detection period before a suspect member is expelled from the group.
- Where possible, try and limit the size of your transactions before they are handled by Group Replication. For example, split up files used with `LOAD DATA` into smaller chunks.
- Use the system variable `group_replication_transaction_size_limit` to specify a maximum transaction size that the group will accept. In MySQL 8.0, this system variable defaults to a maximum transaction size of 150000000 bytes (approximately 143 MB). Transactions above this size are rolled back and are not sent to Group Replication's Group Communication System (GCS) for distribution to the group. Adjust the value of this variable depending on the maximum message size that you need the group to tolerate, bearing in mind that the time taken to process a transaction is proportional to its size.
- Use the system variable `group_replication_compression_threshold` to specify a message size above which compression is applied. This system variable defaults to 1000000 bytes (1 MB), so large messages are automatically compressed. Compression is carried out by Group Replication's Group Communication System (GCS) when it receives a message that was permitted by the `group_replication_transaction_size_limit` setting but exceeds the `group_replication_compression_threshold` setting. For more information, see Section 18.6.3, “Message Compression”.
- Use the system variable `group_replication_communication_max_message_size` to specify a message size above which messages are fragmented. This system variable defaults to 10485760 bytes (10 MiB), so large messages are automatically fragmented. GCS carries out fragmentation after compression if the compressed message still exceeds the `group_replication_communication_max_message_size` limit. In order for a replication group to use fragmentation, all group members must be at MySQL 8.0.16 or above, and the Group Replication communication protocol version in use by the group must allow fragmentation. For more information, see Section 18.6.4, “Message Fragmentation”.

The maximum transaction size, message compression, and message fragmentation can all be deactivated by specifying a zero value for the relevant system variable. If you have deactivated all these safeguards, the upper size limit for a message that can be handled by the applier thread on a member of a replication group is the value of the member's `slave_max_allowed_packet` system variable, which has a default and maximum value of 1073741824 bytes (1 GB). A message that exceeds this limit fails when the receiving member attempts to handle it. The upper size limit for a message that a group member can originate and attempt to transmit to the group is 4294967295 bytes (approximately 4 GB). This is a hard limit on the packet size that is accepted by the group communication engine for Group Replication (XCom, a Paxos variant), which receives messages after GCS has handled them. A message that exceeds this limit fails when the originating member attempts to broadcast it.

18.10 Frequently Asked Questions

This section provides answers to frequently asked questions.

What is the maximum number of MySQL servers in a group?

A group can consist of maximum 9 servers. Attempting to add another server to a group with 9 members causes the request to join to be refused. This limit has been identified from testing and benchmarking as a safe boundary where the group performs reliably on a stable local area network.

How are servers in a group connected?

Servers in a group connect to the other servers in the group by opening a peer-to-peer TCP connection. These connections are only used for internal communication and message passing between servers in the group. This address is configured by the `group_replication_local_address` variable.

What is the group_replication_bootstrap_group option used for?

The bootstrap flag instructs a member to *create* a group and act as the initial seed server. The second member joining the group needs to ask the member that bootstrapped the group to dynamically change the configuration in order for it to be added to the group.

A member needs to bootstrap the group in two scenarios. When the group is originally created, or when shutting down and restarting the entire group.

How do I set credentials for the distributed recovery process?

You pre-configure the Group Replication recovery channel credentials using the `CHANGE MASTER TO` statement.

Can I scale-out my write-load using Group Replication?

Not directly, but MySQL Group replication is a shared nothing full replication solution, where all servers in the group replicate the same amount of data. Therefore if one member in the group writes N bytes to storage as the result of a transaction commit operation, then roughly N bytes are written to storage on other members as well, because the transaction is replicated everywhere.

However, given that other members do not have to do the same amount of processing that the original member had to do when it originally executed the transaction, they apply the changes faster. Transactions are replicated in a format that is used to apply row transformations only, without having to re-execute transactions again (row-based format).

Furthermore, given that changes are propagated and applied in row-based format, this means that they are received in an optimized and compact format, and likely reducing the number of IO operations required when compared to the originating member.

To summarize, you can scale-out processing, by spreading conflict free transactions throughout different members in the group. And you can likely scale-out a small fraction of your IO operations, since remote servers receive only the necessary changes to read-modify-write changes to stable storage.

Does Group Replication require more network bandwidth and CPU, when compared to simple replication and under the same workload?

Some additional load is expected because servers need to be constantly interacting with each other for synchronization purposes. It is difficult to quantify how much more data. It also depends on the size of the group (three servers puts less stress on the bandwidth requirements than nine servers in the group).

Also the memory and CPU footprint are larger, because more complex work is done for the server synchronization part and for the group messaging.

Can I deploy Group Replication across wide-area networks?

Yes, but the network connection between each member *must* be reliable and have suitable perfomance. Low latency, high bandwidth network connections are a requirement for optimal performance.

If network bandwidth alone is an issue, then Section 18.6.3, “Message Compression” can be used to lower the bandwidth required. However, if the network drops packets, leading to re-transmissions and higher end-to-end latency, throughput and latency are both negatively affected.

Warning

When the network round-trip time (RTT) between any group members is 5 seconds or more you could encounter problems as the built-in failure detection mechanism could be incorrectly triggered.

Do members automatically rejoin a group in case of temporary connectivity problems?

This depends on the reason for the connectivity problem. If the connectivity problem is transient and the reconnection is quick enough that the failure detector is not aware of it, then the server may not be removed from the group. If it is a "long" connectivity problem, then the failure detector eventually suspects a problem and the server is removed from the group.

From MySQL 8.0, you can activate two settings to increase the chances of a member remaining in or rejoining a group:

- `group_replication_member_expel_timeout` increases the time between the creation of a suspicion (which happens after an initial 5-second detection period) and the expulsion of the member. You can set a waiting period of up to 1 hour.
- `group_replication_autorejoin_tries` makes a member try to rejoin the group after an expulsion or unreachable majority timeout. The member makes the specified number of auto-rejoin attempts five minutes apart.

If a server is expelled from the group and any auto-rejoin attempts do not succeed, you need to join it back again. In other words, after a server is removed explicitly from the group you need to rejoin it manually (or have a script doing it automatically).

When is a member excluded from a group?

If the member becomes silent, the other members remove it from the group configuration. In practice this may happen when the member has crashed or there is a network disconnection.

The failure is detected after a given timeout elapses for a given member and a new configuration without the silent member in it is created.

What happens when one node is significantly lagging behind?

There is no method for defining policies for when to expel members automatically from the group. You need to find out why a member is lagging behind and fix that or remove the member from the group. Otherwise, if the server is so slow that it triggers the flow control, then the entire group slows down as well. The flow control can be configured according to the your needs.

Upon suspicion of a problem in the group, is there a special member responsible for triggering a reconfiguration?

No, there is no special member in the group in charge of triggering a reconfiguration.

Any member can suspect that there is a problem. All members need to (automatically) agree that a given member has failed. One member is in charge of expelling it from the group, by triggering a reconfiguration. Which member is responsible for expelling the member is not something you can control or set.

Can I use Group Replication for sharding?

Group Replication is designed to provide highly available replica sets; data and writes are duplicated on each member in the group. For scaling beyond what a single system can provide, you need an orchestration and sharding framework built around a number of Group Replication sets, where each replica set maintains and manages a given shard or partition of your total dataset. This type of setup, often called a “sharded cluster”, allows you to scale reads and writes linearly and without limit.

How do I use Group Replication with SELinux?

If SELinux is enabled, which you can verify using `sestatus -v`, then you need to enable the use of the Group Replication communication port, configured by `group_replication_local_address`, for `mysqld` so that it can bind to it and listen there. To see which ports MySQL is currently allowed to use, issue `semanage port -l | grep mysqld`. Assuming the port configured is 33061, add the necessary port to those permitted by SELinux by issuing `semanage port -a -t mysqld_port_t -p tcp 33061`.

How do I use Group Replication with iptables?

If `iptables` is enabled, then you need to open up the Group Replication port for communication between the machines. To see the current rules in place on each machine, issue `iptables -L`. Assuming the port configured is 33061, enable communication over the necessary port by issuing `iptables -A INPUT -p tcp --dport 33061 -j ACCEPT`.

How do I recover the relay log for a replication channel used by a group member?

The replication channels used by Group Replication behave in the same way as replication channels used in master to slave replication, and as such rely on the relay log. In the event of a change of the `relay_log` variable, or when the option is not set and the host name changes, there is a chance of errors. See Section 17.2.4.1, “The Slave Relay Log” for a recovery procedure in this situation. Alternatively, another way of fixing the issue specifically in Group Replication is to issue a `STOP GROUP_REPLICATION` statement and then a `START GROUP_REPLICATION` statement to restart the instance. The Group Replication plugin creates the `group_replication_applier` channel again.

Why does Group Replication use two bind addresses?

Group Replication uses two bind addresses in order to split network traffic between the SQL address, used by clients to communicate with the member, and the `group_replication_local_address`, used internally by the group members to communicate. For example, assume a server with two network interfaces assigned to the network addresses `203.0.113.1` and `198.51.100.179`. In such a situation you could use `203.0.113.1:33061` for the internal group network address by setting `group_replication_local_address=203.0.113.1:33061`. Then you could use `198.51.100.179` for `hostname` and `3306` for the `port`. Client SQL applications would then connect to the member at `198.51.100.179:3306`. This enables you to configure different rules on the different networks. Similarly, the internal group communication can be separated from the network connection used for client applications, for increased security.

How does Group Replication use network addresses and hostnames?

Group Replication uses network connections between members and therefore its functionality is directly impacted by how you configure hostnames and ports. For example, Group Replication's distributed recovery process creates a connection to an existing group member using the server's hostname and

port. When a member joins a group it receives the group membership information, using the network address information that is listed at `performance_schema.replication_group_members`. One of the members listed in that table is selected as the donor of the missing data from the group to the joining member.

This means that any value you configure using a hostname, such as the SQL network address or the group seeds address, must be a fully qualified name and resolvable by each member of the group. You can ensure this for example through DNS, or correctly configured `/etc/hosts` files, or other local processes. If a you want to configure the `MEMBER_HOST` value on a server, specify it using the `--report-host` option on the server before joining it to the group.

Important

The assigned value is used directly and is not affected by the `--skip-name-resolve` option.

To configure `MEMBER_PORT` on a server, specify it using the `--report-port` option.

Why did the auto increment setting on the server change?

When Group Replication is started on a server, the value of `auto_increment_increment` is changed to the value of `group_replication_auto_increment_increment`, which defaults to 7, and the value of `auto_increment_offset` is changed to the server ID. The changes are reverted when Group Replication is stopped. These settings avoid the selection of duplicate auto-increment values for writes on group members, which causes rollback of transactions. The default auto increment value of 7 for Group Replication represents a balance between the number of usable values and the permitted maximum size of a replication group (9 members).

The changes are only made and reverted if `auto_increment_increment` and `auto_increment_offset` each have their default value of 1. If their values have already been modified from the default, Group Replication does not alter them. From MySQL 8.0, the system variables are also not modified when Group Replication is in single-primary mode, where only one server writes.

How do I find the primary?

If the group is operating in single-primary mode, it can be useful to find out which member is the primary. See Finding the Primary

Chapter 19 MySQL Shell

MySQL Shell is an advanced client and code editor for MySQL Server. In addition to the provided SQL functionality, similar to `mysql`, MySQL Shell provides scripting capabilities for JavaScript and Python and includes APIs for working with MySQL.

The following discussion briefly describes MySQL Shell's capabilities. For more information, see the MySQL Shell manual, available at https://dev.mysql.com/doc/mysql-shell/en/.

MySQL Shell includes the following APIs implemented in JavaScript and Python which you can use to develop code that interacts with MySQL.

- The X DevAPI enables you to work with both relational and document data when MySQL Shell is connected to a MySQL server using the X Protocol. For more information, see Chapter 20, *Using MySQL as a Document Store*. For documentation on the concepts and usage of X DevAPI, see X DevAPI User Guide.
- The AdminAPI enables you to work with InnoDB cluster, which provides an integrated solution for high availability and scalability using InnoDB based MySQL databases, without requiring advanced MySQL expertise. See Chapter 21, *InnoDB Cluster*.

MySQL Shell is available in two editions, the Community Edition and the Commercial Edition. The Community Edition is available free of charge. The Commercial Edition provides additional Enterprise features at low cost.

Chapter 20 Using MySQL as a Document Store

Table of Contents

20.1 Interfaces to a MySQL Document Store 3530
20.2 Document Store Concepts 3530
20.3 JavaScript Quick-Start Guide: MySQL Shell for Document Store 3531
20.3.1 MySQL Shell 3532
20.3.2 Download and Import world_x Database 3533
20.3.3 Documents and Collections 3534
20.3.4 Relational Tables 3545
20.3.5 Documents in Tables 3551
20.4 Python Quick-Start Guide: MySQL Shell for Document Store 3552
20.4.1 MySQL Shell 3553
20.4.2 Download and Import world_x Database 3554
20.4.3 Documents and Collections 3555
20.4.4 Relational Tables 3567
20.4.5 Documents in Tables 3573
20.5 X Plugin 3574
20.5.1 Checking X Plugin Installation 3574
20.5.2 Disabling X Plugin 3574
20.5.3 Using Secure Connections with X Plugin 3574
20.5.4 Using X Plugin with the Caching SHA-2 Authentication Plugin 3575
20.5.5 X Plugin Options and Variables 3575
20.5.6 Monitoring X Plugin 3587

This chapter introduces an alternative way of working with MySQL as a document store, sometimes referred to as “using NoSQL”. If your intention is to use MySQL in a traditional (SQL) way, this chapter is probably not relevant to you.

Traditionally, relational databases such as MySQL have usually required a schema to be defined before documents can be stored. The features described in this section enable you to use MySQL as a document store, which is a schema-less, and therefore schema-flexible, storage system for documents. For example, when you create documents describing products, you do not need to know and define all possible attributes of any products before storing and operating with the documents. This differs from working with a relational database and storing products in a table, when all columns of the table must be known and defined before adding any products to the database. The features described in this chapter enable you to choose how you configure MySQL, using only the document store model, or combining the flexibility of the document store model with the power of the relational model.

To use MySQL as a document store, you use the following server features:

- X Plugin enables MySQL Server to communicate with clients using X Protocol, which is a prerequisite for using MySQL as a document store. X Plugin is enabled by default in MySQL Server as of MySQL 8.0. For instructions to verify X Plugin installation and to configure and monitor X Plugin, see Section 20.5, “X Plugin”.
- X Protocol supports both CRUD and SQL operations, authentication via SASL, allows streaming (pipelining) of commands and is extensible on the protocol and the message layer. Clients compatible with X Protocol include MySQL Shell and MySQL 8.0 Connectors.
- Clients that communicate with a MySQL Server using X Protocol can use X DevAPI to develop applications. X DevAPI offers a modern programming interface with a simple yet powerful design which

provides support for established industry standard concepts. This chapter explains how to get started using either the JavaScript or Python implementation of X DevAPI in MySQL Shell as a client. See X DevAPI User Guide for in-depth tutorials on using X DevAPI.

20.1 Interfaces to a MySQL Document Store

To work with MySQL as a document store, you use dedicated components and a choice of clients that support communicating with the MySQL server to develop document based applications.

- The following MySQL products support X Protocol and enable you to use X DevAPI in your chosen language to develop applications that communicate with a MySQL Server functioning as a document store:
  - MySQL Shell (which provides implementations of X DevAPI in JavaScript and Python)
  - Connector/C++
  - Connector/J
  - Connector/Node.js
  - Connector/NET
  - Connector/Python
- MySQL Shell is an interactive interface to MySQL supporting JavaScript, Python, or SQL modes. You can use MySQL Shell to prototype applications, execute queries and update data. Installing MySQL Shell has instructions to download and install MySQL Shell.
- The quick-start guides (tutorials) in this chapter help you to get started using MySQL Shell with MySQL as a document store.

  The quick-start guide for JavaScript is here: Section 20.3, “JavaScript Quick-Start Guide: MySQL Shell for Document Store”.

  The quick-start guide for Python is here: Section 20.4, “Python Quick-Start Guide: MySQL Shell for Document Store”.
- The *MySQL Shell User Guide* at MySQL Shell 8.0 (part of MySQL 8.0) provides detailed information about configuring and using MySQL Shell.

20.2 Document Store Concepts

This section explains the concepts introduced as part of using MySQL as a document store.

- JSON Document
- Collection
- CRUD Operations

JSON Document

A JSON document is a data structure composed of key-value pairs and is the fundamental structure for using MySQL as document store. For example, the world_x schema (installed later in this chapter) contains this document:

```
{
    "GNP": .6,
    "IndepYear": 1967,
    "Name": "Sealand",
    "_id": "SEA",
    "demographics": {
        "LifeExpectancy": 79,
        "Population": 27
    },
    "geography": {
        "Continent": "Europe",
        "Region": "British Islands",
        "SurfaceArea": 193
    },
    "government": {
        "GovernmentForm": "Monarchy",
        "HeadOfState": "Michael Bates"
    }
}
```

This document shows that the values of keys can be simple data types, such as integers or strings, but can also contain other documents, arrays, and lists of documents. For example, the `geography` key's value consists of multiple key-value pairs. A JSON document is represented internally using the MySQL binary JSON object, through the `JSON` MySQL datatype.

The most important differences between a document and the tables known from traditional relational databases are that the structure of a document does not have to be defined in advance, and a collection can contain multiple documents with different structures. Relational tables on the other hand require that their structure be defined, and all rows in the table must contain the same columns.

Collection

A collection is a container that is used to store JSON documents in a MySQL database. Applications usually run operations against a collection of documents, for example to find a specific document.

CRUD Operations

The four basic operations that can be issued against a collection are Create, Read, Update and Delete (CRUD). In terms of MySQL this means:

- Create a new document (insertion or addition)
- Read one or more documents (queries)
- Update one or more documents
- Delete one or more documents

20.3 JavaScript Quick-Start Guide: MySQL Shell for Document Store

This quick-start guide provides instructions to begin prototyping document store applications interactively with MySQL Shell. The guide includes the following topics:

- Introduction to MySQL functionality, MySQL Shell, and the `world_x` example schema.
- Operations to manage collections and documents.
- Operations to manage relational tables.
- Operations that apply to documents within tables.

To follow this quick-start guide you need a MySQL server with X Plugin installed, the default in 8.0, and MySQL Shell to use as the client. MySQL Shell 8.0 (part of MySQL 8.0) provides more in-depth information about MySQL Shell. The Document Store is accessed using X DevAPI, and MySQL Shell provides this API in both JavaScript and Python.

Related Information

- MySQL Shell 8.0 (part of MySQL 8.0) provides more in-depth information about MySQL Shell.
- See Installing MySQL Shell and Section 20.5, “X Plugin” for more information about the tools used in this quick-start guide.
- X DevAPI User Guide provides more examples of using X DevAPI to develop applications which use Document Store.

20.3.1 MySQL Shell

This quick-start guide assumes a certain level of familiarity with MySQL Shell. The following section is a high level overview, see the xshe documentation for more information. MySQL Shell is a unified scripting interface to MySQL Server. It supports scripting in JavaScript and Python. JavaScript is the default processing mode. In most cases, you need an account to connect to the MySQL server instance.

Start MySQL Shell

After you have installed and started MySQL server, connect MySQL Shell to the server instance. You need to know the address of the MySQL server instance you plan to connect to. To be able to use the instance as a Document Store, the server instance must have X Plugin installed and you should connect to the server using X Protocol. For example to connect to the instance `ds1.example.com` on the default X Protocol port of 33060 use the network string *`user`*`@ds1.example.com:33060`.

Tip

If you connect to the instance using MySQL classic protocol, for example by using the default port of 3306 rather than the `mysqlx_port`, you cannot use the Document Store functionality shown in this tutorial. For example the `db` global object is not populated. To use the Document Store, always connect using X Protocol.

If MySQL Shell is not already running, open a terminal window and issue:

```
mysqlsh user@ds1.example.com:33060/world_x
```

Alternatively, if MySQL Shell is already running use the `\connect` command by issuing:

```
\connect user@ds1.example.com:33060/world_x
```

You need to specify the address of the MySQL server instance which you want to connect MySQL Shell to. For example in the previous example:

- *`user`* represents the user name of your MySQL account.
- `ds1.example.com` is the hostname of the server instance running MySQL. Replace this with the hostname of the MySQL server instance you are using as a Document Store.
- The default schema for this session is `world_x`. For instructions on setting up the `world_x` schema, see Section 20.3.2, “Download and Import world_x Database”.

For more information, see Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”.

Once MySQL Shell opens, the `mysql-js>` prompt indicates that the active language for this session is JavaScript.

```
mysql-js>
```

MySQL Shell supports input-line editing as follows:

- **left-arrow** and **right-arrow** keys move horizontally within the current input line.
- **up-arrow** and **down-arrow** keys move up and down through the set of previously entered lines.
- **Backspace** deletes the character before the cursor and typing new characters enters them at the cursor position.
- **Enter** sends the current input line to the server.

Get Help for MySQL Shell

Type `mysqlsh --help` at the prompt of your command interpreter for a list of command-line options.

```
mysqlsh --help
```

Type `\help` at the MySQL Shell prompt for a list of available commands and their descriptions.

```
mysql-js> \help
```

Type `\help` followed by a command name for detailed help about an individual MySQL Shell command. For example, to view help on the `\connect` command, issue:

```
mysql-js> \help \connect
```

Quit MySQL Shell

To quit MySQL Shell, issue the following command:

```
mysql-js> \quit
```

Related Information

- See Interactive Code Execution for an explanation of how interactive code execution works in MySQL Shell.
- See Getting Started with MySQL Shell to learn about session and connection alternatives.

20.3.2 Download and Import world_x Database

As part of this quick-start guide, an example schema is provided which is referred to as the `world_x` schema. Many of the examples demonstrate Document Store functionality using this schema. Start your MySQL server so that you can load the `world_x` schema, then follow these steps:

1. Download world_x-db.zip.

2. Extract the installation archive to a temporary location such as `/tmp/`. Unpacking the archive results in a single file named `world_x.sql`.

3. Import the `world_x.sql` file to your server. You can either:

   - Start MySQL Shell in SQL mode and import the file by issuing:

     ```
     mysqlsh -u root --sql --file /tmp/world_x-db/world_x.sql
     Enter password: ****
     ```

   - Set MySQL Shell to SQL mode while it is running and source the schema file by issuing:

     ```
     \sql
     Switching to SQL mode... Commands end with ;
     \source /tmp/world_x-db/world_x.sql
     ```

   Replace `/tmp/` with the path to the `world_x.sql` file on your system. Enter your password if prompted. A non-root account can be used as long as the account has privileges to create new schemas.

The world_x Schema

The `world_x` example schema contains the following JSON collection and relational tables:

- Collection
  - `countryinfo`: Information about countries in the world.
- Tables
  - `country`: Minimal information about countries of the world.
  - `city`: Information about some of the cities in those countries.
  - `countrylanguage`: Languages spoken in each country.

Related Information

- MySQL Shell Sessions explains session types.

20.3.3 Documents and Collections

When you are using MySQL as a Document Store, collections are containers within a schema that you can create, list, and drop. Collections contain JSON documents that you can add, find, update, and remove.

The examples in this section use the `countryinfo` collection in the `world_x` schema. For instructions on setting up the `world_x` schema, see Section 20.3.2, “Download and Import world_x Database”.

Documents

In MySQL, documents are represented as JSON objects. Internally, they are stored in an efficient binary format that enables fast lookups and updates.

- Simple document format for JavaScript:

  ```
  {field1: "value", field2 : 10, "field 3": null}
  ```

An array of documents consists of a set of documents separated by commas and enclosed within [and] characters.

- Simple array of documents for JavaScript:

```
[{Name: "Aruba", _id: "ABW"}, {Name: "Angola", _id: "AGO"}]
```

MySQL supports the following JavaScript value types in JSON documents:

- numbers (integer and floating point)
- strings
- boolean (False and True)
- null
- arrays of more JSON values
- nested (or embedded) objects of more JSON values

Collections

Collections are containers for documents that share a purpose and possibly share one or more indexes. Each collection has a unique name and exists within a single schema.

The term schema is equivalent to a database, which means a group of database objects as opposed to a relational schema, used to enforce structure and constraints over data. A schema does not enforce conformity on the documents in a collection.

In this quick-start guide:

- Basic objects include:

Object form	Description
`db`	`db` is a global variable assigned to the current active schema. When you want to run operations against the schema, for example to retrieve a collection, you use methods available for the `db` variable.
`db.getCollections()`	db.getCollections() holds a list of collections in the schema. Use the list to get references to collection objects, iterate over them, and so on.

- Basic operations scoped by collections include:

Operation form	Description
`db.`*`name`*`.add()`	The add() method inserts one document or a list of documents into the named collection.
`db.`*`name`*`.find()`	The find() method returns some or all documents in the named collection.
`db.`*`name`*`.modify()`	The modify() method updates documents in the named collection.
`db.`*`name`*`.remove()`	The remove() method deletes one document or a list of documents from the named collection.

Related Information

- See Working with Collections for a general overview.
- CRUD EBNF Definitions provides a complete list of operations.

20.3.3.1 Create, List, and Drop Collections

In MySQL Shell, you can create new collections, get a list of the existing collections in a schema, and remove an existing collection from a schema. Collection names are case-sensitive and each collection name must be unique.

Confirm the Schema

To show the value that is assigned to the schema variable, issue:

```
mysql-js> db
```

If the schema value is not `Schema:world_x`, then set the `db` variable by issuing:

```
mysql-js> \use world_x
```

Create a Collection

To create a new collection in an existing schema, use the `db` object's `createCollection()` method. The following example creates a collection called `flags` in the `world_x` schema.

```
mysql-js> db.createCollection("flags")
```

The method returns a collection object.

```
<Collection:flags>
```

List Collections

To display all collections in the `world_x` schema, use the `db` object's `getCollections()` method. Collections returned by the server you are currently connected to appear between brackets.

```
mysql-js> db.getCollections()
[
    <Collection:countryinfo>,
    <Collection:flags>
]
```

Drop a Collection

To drop an existing collection from a schema, use the `db` object's `dropCollection()` method. For example, to drop the flags collection from the `world_x` schema, issue:

```
mysql-js> db.dropCollection("flags")
```

The `dropCollection()` method is also used in MySQL Shell to drop a relational table from a schema.

Related Information

- See Collection Objects for more examples.

20.3.3.2 Working with Collections

To work with the collections in a schema, use the `db` global object to access the current schema. In this example we are using the `world_x` schema imported previously, and the `countryinfo` collection. Therefore, the format of the operations you issue is `db.collection_name.operation`, where *`collection_name`* is the name of the collection which the operation is executed against. In the following examples, the operations are executed against the `countryinfo` collection.

Add a Document

Use the `add()` method to insert one document or a list of documents into an existing collection. Insert the following document into the `countryinfo` collection. As this is multi-line content, press **Enter** twice to insert the document.

```
mysql-js> db.countryinfo.add(
 {
    GNP: .6,
    IndepYear: 1967,
    Name: "Sealand",
    demographics: {
        LifeExpectancy: 79,
        Population: 27
    },
    geography: {
        Continent: "Europe",
        Region: "British Islands",
        SurfaceArea: 193
    },
    government: {
        GovernmentForm: "Monarchy",
        HeadOfState: "Michael Bates"
    }
  }
)
```

The method returns the status of the operation.

Each document requires an `_id` identifier field, and the value must be unique among all documents in the same collection. In MySQL 8.0.11 and higher, document IDs are generated by the server, not the client, so MySQL Shell does not automatically set an `_id` value. A MySQL server at 8.0.11 or higher sets an `_id` value if the document does not contain the `_id` field. A MySQL server at an earlier 8.0 release or at 5.7 does not set an `_id` value in this situation, so you must specify it explicitly. If you do not, MySQL Shell returns error 5115 `Document is missing a required field`.

Related Information

- See CollectionAddFunction for the full syntax definition.
- See Understanding Document IDs.

20.3.3.3 Find Documents

You can use the `find()` method to query for and return documents from a collection in a schema. MySQL Shell provides additional methods to use with the `find()` method to filter and sort the returned documents.

MySQL provides the following operators to specify search conditions: `OR` (`||`), `AND` (`&&`), `XOR`, `IS`, `NOT`, `BETWEEN`, `IN`, `LIKE`, `!=`, `<>`, `>`, `>=`, `<`, `<=`, `&`, `|`, `<<`, `>>`, `+`, `-`, `*`, `/`, `~`, and `%`.

Find All Documents in a Collection

To return all documents in a collection, use the `find()` method without specifying search conditions. For example, the following operation returns all documents in the `countryinfo` collection.

```
mysql-js> db.countryinfo.find()
[
     {
          "GNP": 828,
          "IndepYear": null,
          "Name": "Aruba",
          "_id": "ABW",
          "demographics": {
              "LifeExpectancy": 78.4000015258789,
              "Population": 103000
          },
          "geography": {
              "Continent": "North America",
              "Region": "Caribbean",
              "SurfaceArea": 193
          },
          "government": {
              "GovernmentForm": "Nonmetropolitan Territory of The Netherlands",
              "HeadOfState": "Beatrix"
          }
          ...
      }
 ]
240 documents in set (0.00 sec)
```

The method produces results that contain operational information in addition to all documents in the collection.

An empty set (no matching documents) returns the following information:

```
Empty set (0.00 sec)
```

Filter Searches

You can include search conditions with the `find()` method. The syntax for expressions that form a search condition is the same as that of traditional MySQL Chapter 12, *Functions and Operators*. You must enclose all expressions in quotes. For the sake of brevity, some of the examples do not display output.

A simple search condition could consist of the `Name` field and a value we know is in a document. The following example returns a single document:

```
mysql-js> db.countryinfo.find("Name = 'Australia'")
[
    {
        "GNP": 351182,
        "IndepYear": 1901,
        "Name": "Australia",
        "_id": "AUS",
        "demographics": {
            "LifeExpectancy": 79.80000305175781,
            "Population": 18886000
        },
        "geography": {
            "Continent": "Oceania",
            "Region": "Australia and New Zealand",
            "SurfaceArea": 7741220
        },
```

```
        "government": {
            "GovernmentForm": "Constitutional Monarchy, Federation",
            "HeadOfState": "Elisabeth II"
        }
    }
]
```

The following example searches for all countries that have a GNP higher than $500 billion. The `countryinfo` collection measures GNP in units of million.

```
mysql-js> db.countryinfo.find("GNP > 500000")
...[output removed]
10 documents in set (0.00 sec)
```

The Population field in the following query is embedded within the demographics object. To access the embedded field, use a period between demographics and Population to identify the relationship. Document and field names are case-sensitive.

```
mysql-js> db.countryinfo.find("GNP > 500000 and demographics.Population < 100000000")
...[output removed]
6 documents in set (0.00 sec)
```

Arithmetic operators in the following expression are used to query for countries with a GNP per capita higher than $30000. Search conditions can include arithmetic operators and most MySQL functions.

Note

Seven documents in the `countryinfo` collection have a population value of zero. Therefore warning messages appear at the end of the output.

```
mysql-js> db.countryinfo.find("GNP*1000000/demographics.Population > 30000")
...[output removed]
9 documents in set, 7 warnings (0.00 sec)
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
```

You can separate a value from the search condition by using the `bind()` method. For example, instead of specifying a hard-coded country name as the condition, substitute a named placeholder consisting of a colon followed by a name that begins with a letter, such as *country*. Then use the `bind(placeholder, value)` method as follows:

```
mysql-js> db.countryinfo.find("Name = :country").bind("country", "Italy")
[
    {
        "GNP": 1161755,
        "IndepYear": 1861,
        "Name": "Italy",
        "_id": "ITA",
        "demographics": {
            "LifeExpectancy": 79,
            "Population": 57680000
        },
        "geography": {
            "Continent": "Europe",
            "Region": "Southern Europe",
```

```
            "SurfaceArea": 301316
        },
        "government": {
            "GovernmentForm": "Republic",
            "HeadOfState": "Carlo Azeglio Ciampi"
        }
    }
]
1 document in set (0.01 sec)
```

Tip

Within a program, binding enables you to specify placeholders in your expressions, which are filled in with values before execution and can benefit from automatic escaping, as appropriate.

Always use binding to sanitize input. Avoid introducing values in queries using string concatenation, which can produce invalid input and, in some cases, can cause security issues.

You can use placeholders and the `bind()` method to create saved searches which you can then call with different values. For example to create a saved search for a country:

```
mysql-js> var myFind = db.countryinfo.find("Name = :country")
mysql-js> myFind.bind('country', 'France')
[
    {
        "GNP": 1424285,
        "IndepYear": 843,
        "Name": "France",
        "_id": "FRA",
        "demographics": {
            "LifeExpectancy": 78.80000305175781,
            "Population": 59225700
        },
        "geography": {
            "Continent": "Europe",
            "Region": "Western Europe",
            "SurfaceArea": 551500
        },
        "government": {
            "GovernmentForm": "Republic",
            "HeadOfState": "Jacques Chirac"
        }
    }
]
1 document in set (0.0028 sec)

mysql-js> myFind.bind('country', 'Germany')
[
    {
        "GNP": 2133367,
        "IndepYear": 1955,
        "Name": "Germany",
        "_id": "DEU",
        "demographics": {
            "LifeExpectancy": 77.4000015258789,
            "Population": 82164700
        },
        "geography": {
            "Continent": "Europe",
            "Region": "Western Europe",
            "SurfaceArea": 357022
        },
```

```
        "government": {
            "GovernmentForm": "Federal Republic",
            "HeadOfState": "Johannes Rau"
        }
    }
]
1 document in set (0.0026 sec)
```

Project Results

You can return specific fields of a document, instead of returning all the fields. The following example returns the GNP and Name fields of all documents in the `countryinfo` collection matching the search conditions.

Use the `fields()` method to pass the list of fields to return.

```
mysql-js> db.countryinfo.find("GNP > 5000000").fields(["GNP", "Name"])
[
    {
        "GNP": 8510700,
        "Name": "United States"
    }
]
1 document in set (0.00 sec)
```

In addition, you can alter the returned documents—adding, renaming, nesting and even computing new field values—with an expression that describes the document to return. For example, alter the names of the fields with the following expression to return only two documents.

```
mysql-js> db.countryinfo.find().
fields(mysqlx.expr('{"Name": upper(Name), "GNPPerCapita": GNP*1000000/demographics.Population}')).
limit(2)
[
    {
        "GNPPerCapita": 8038.834951456311,
        "Name": "ARUBA"
    },
    {
        "GNPPerCapita": 263.0281690140845,
        "Name": "AFGHANISTAN"
    }
]
2 documents in set (0.00 sec)
```

Limit, Sort, and Skip Results

You can apply the `limit()`, `sort()`, and `skip()` methods to manage the number and order of documents returned by the `find()` method.

To specify the number of documents included in a result set, append the `limit()` method with a value to the `find()` method. The following query returns the first five documents in the `countryinfo` collection.

```
mysql-js> db.countryinfo.find().limit(5)
... [output removed]
5 documents in set (0.00 sec)
```

To specify an order for the results, append the `sort()` method to the `find()` method. Pass to the `sort()` method a list of one or more fields to sort by and, optionally, the descending (`desc`) or ascending (`asc`) attribute as appropriate. Ascending order is the default order type.

For example, the following query sorts all documents by the IndepYear field and then returns the first eight documents in descending order.

```
mysql-js> db.countryinfo.find().sort(["IndepYear desc"]).limit(8)
... [output removed]
8 documents in set (0.00 sec)
```

By default, the `limit()` method starts from the first document in the collection. You can use the `skip()` method to change the starting document. For example, to ignore the first document and return the next eight documents matching the condition, pass to the `skip()` method a value of 1.

```
mysql-js> db.countryinfo.find().sort(["IndepYear desc"]).limit(8).skip(1)
... [output removed]
8 documents in set (0.00 sec)
```

Related Information

- The MySQL Reference Manual provides detailed documentation on functions and operators.
- See CollectionFindFunction for the full syntax definition.

20.3.3.4 Modify Documents

You can use the `modify()` method to update one or more documents in a collection. The X DevAPI provides additional methods for use with the `modify()` method to:

- Set and unset fields within documents.
- Append, insert, and delete arrays.
- Bind, limit, and sort the documents to be modified.

Set and Unset Fields

The `modify()` method works by filtering a collection to include only the documents to be modified and then applying the operations that you specify to those documents.

In the following example, the `modify()` method uses the search condition to identify the document to change and then the `set()` method replaces two values within the nested demographics object.

```
mysql-js> db.countryinfo.modify("_id = 'SEA'").\
set("demographics", {LifeExpectancy: 78, Population: 28})
Query OK, 1 item affected (0.04 sec)
```

After you modify a document, use the `find()` method to verify the change.

To remove content from a document, use the `modify()` and `unset()` methods. For example, the following query removes the GNP from a document that matches the search condition.

```
mysql-js> db.countryinfo.modify("Name = 'Sealand'").unset("GNP")
Query OK, 1 item affected (0.01 sec)
```

Use the `find()` method to verify the change.

```
mysql-js> db.countryinfo.find("Name = 'Sealand'")
[
    {
        "IndepYear": 1967,
```

```
        "Name": "Sealand",
        "_id": "SEA",
        "demographics": {
            "LifeExpectancy": 78,
            "Population": 28
        },
        "geography": {
            "Continent": "Europe",
            "Region": "British Islands",
            "SurfaceArea": 193
        },
        "government": {
            "GovernmentForm": "Monarchy",
            "HeadOfState": "Michael Bates"
        }
    }
]
1 document in set (0.00 sec)
```

Append, Insert, and Delete Arrays

To append an element to an array field, or insert, or delete elements in an array, use the `arrayAppend()`, `arrayInsert()`, or `arrayDelete()` methods. The following examples modify the `countryinfo` collection to enable tracking of international airports.

The first example uses the `modify()` and `set()` methods to create a new Airports field in all documents.

Caution

Use care when you modify documents without specifying a search condition. This action will modify all documents in the collection.

```
mysql-js> db.countryinfo.modify("true").set("Airports", [])
Query OK, 240 items affected (0.07 sec)
```

With the Airports field added, the next example uses the `arrayAppend()` method to add a new airport to one of the documents. *$.Airports* in the following example represents the Airports field of the current document.

```
mysql-js> db.countryinfo.modify("Name = 'France'").arrayAppend("$.Airports", "ORY")
Query OK, 1 item affected (0.02 sec)
```

Use **`db.countryinfo.find("Name = 'France'")`** to see the change.

To insert an element at a different position in the array, use the `arrayInsert()` method to specify which index to insert in the path expression. In this case, the index is 0, or the first element in the array.

```
mysql-js> db.countryinfo.modify("Name = 'France'").arrayInsert("$.Airports[0]", "CDG")
Query OK, 1 item affected (0.04 sec)
```

To delete an element from the array, you must pass to the `arrayDelete()` method the index of the element to be deleted.

```
mysql-js> db.countryinfo.modify("Name = 'France'").arrayDelete("$.Airports[1]")
Query OK, 1 item affected (0.03 sec)
```

Related Information

- The MySQL Reference Manual provides instructions to help you search for and modify JSON values.

- See CollectionModifyFunction for the full syntax definition.

20.3.3.5 Remove Documents

You can use the `remove()` method to delete some or all documents from a collection in a schema. The X DevAPI provides additional methods for use with the `remove()` method to filter and sort the documents to be removed.

Remove Documents Using Conditions

The example that follows passes a search condition to the `remove()` method. All documents matching the condition are removed from the `countryinfo` collection. In this example, one document matches the condition.

```
mysql-js> db.countryinfo.remove("_id = 'SEA'")
Query OK, 1 item affected (0.02 sec)
```

Remove the First Document

To remove the first document in the `countryinfo` collection, use the `limit()` method with a value of 1.

```
mysql-js> db.countryinfo.remove("true").limit(1)
Query OK, 1 item affected (0.03 sec)
```

Remove the Last Document in an Order

The following example removes the last document in the `countryinfo` collection by country name.

```
mysql-js> db.countryinfo.remove("true").sort(["Name desc"]).limit(1)
Query OK, 1 item affected (0.02 sec)
```

Remove All Documents in a Collection

You can remove all documents in a collection. To do so, use the `remove("true")` method without specifying any search condition.

Caution

Use care when you remove documents without specifying a search condition. This action will delete all documents from the collection.

Related Information

- See CollectionRemoveFunction for the full syntax definition.
- See Section 20.3.2, “Download and Import world_x Database” for instructions to recreate the `world_x` schema.

20.3.3.6 Create and Drop Indexes

Indexes are used to find documents with specific field values quickly. Without an index, MySQL must begin with the first document and then read through the entire collection to find the relevant fields. The larger the collection, the more this costs. If a collection is large and queries on a specific field are common, then consider creating an index on a specific field inside a document.

For example, the following query will perform better with an index:

```
mysql-js> db.countryinfo.find("demographics.Population < 100")
...[output removed]
8 documents in set (0.00 sec)
```

The `createIndex()` method creates an index that you can define as nonunique or unique. Use the `field()` method to chain the fields that should be indexed. The `execute()` method is required to create or drop an index.

In MySQL, the `_id` field is equivalent to a primary key by default.

Add a Nonunique Index

To create a nonunique index, pass to the `createIndex()` method an index name. Duplicate index names are prohibited.

In the following example, the first parameter of the `field()` method specifies the Population field inside the demographics object and the next parameter indicates that the field should be indexed as an Integer numeric value. The last parameter indicates whether the field should require the NOT NULL constraint. If the value is `false`, the field can contain `NULL` values.

```
mysql-js> db.countryinfo.createIndex("pop").
field("demographics.Population", "INTEGER", false).execute()
Query OK (0.04 sec)
```

Add a Unique Index

To create a unique index, pass to the `createIndex()` method an index name and the `mysqlx.IndexType.UNIQUE` type. Country `"Name"` is another common field in the `countryinfo` collection to index. In the following example, `"Text(40)"` represents the number of characters to index and `true` indicates that the field cannot contain any `NULL` values.

```
mysql-js> db.countryinfo.createIndex("name", mysqlx.IndexType.UNIQUE).
field("Name", "TEXT(40)", true).execute()
Query OK (0.04 sec)
```

Drop an Index

To drop an index, pass to the `dropIndex()` method the name of the index to drop. For example, you can drop the “pop” index as follows:

```
mysql-js> db.countryinfo.dropIndex("pop").execute()
Query OK (0.58 sec)
```

Related Information

- See Collection Index Management Functions for the full syntax definition.

20.3.4 Relational Tables

You can also use X DevAPI to work with relational tables. In MySQL, each relational table is associated with a particular storage engine. The examples in this section use `InnoDB` tables in the `world_x` schema.

Confirm the Schema

To show the schema that is assigned to the `db` global variable, issue `db`.

```
mysql-js> db
```

```
<Schema:world_x>
```

If the returned value is not `Schema:world_x`, set the `db` variable as follows:

```
mysql-js> \use world_x
Schema `world_x` accessible through db.
```

Show All Tables

To display all relational tables in the `world_x` schema, use the `getTables()` method on the `db` object.

```
mysql-js> db.getTables()
{
    "city": <Table:city>,
    "country": <Table:country>,
    "countrylanguage": <Table:countrylanguage>
}
```

Basic Table Operations

Basic operations scoped by tables include:

Operation form	Description
`db.name.insert()`	The insert() method inserts one or more records into the named table.
`db.name.select()`	The select() method returns some or all records in the named table.
`db.name.update()`	The update() method updates records in the named table.
`db.name.delete()`	The delete() method deletes one or more records from the named table.

Related Information

- See Working with Relational Tables for more information.
- CRUD EBNF Definitions provides a complete list of operations.
- See Section 20.3.2, “Download and Import world_x Database” for instructions on setting up the `world_x` schema sample.

20.3.4.1 Insert Records into Tables

You can use the `insert()` method with the `values()` method to insert records into an existing relational table. The `insert()` method accepts individual columns or all columns in the table. Use one or more `values()` methods to specify the values to be inserted.

Insert a Complete Record

To insert a complete record, pass to the `insert()` method all columns in the table. Then pass to the `values()` method one value for each column in the table. For example, to add a new record to the city table in the `world_x` schema, insert the following record and press **Enter** twice.

```
mysql-js> db.city.insert("ID", "Name", "CountryCode", "District", "Info").
values(null, "Olympia", "USA", "Washington", '{"Population": 5000}')
Query OK, 1 item affected (0.01 sec)
```

The city table has five columns: ID, Name, CountryCode, District, and Info. Each value must match the data type of the column it represents.

Insert a Partial Record

The following example inserts values into the ID, Name, and CountryCode columns of the city table.

```
mysql-js> db.city.insert("ID", "Name", "CountryCode").
values(null, "Little Falls", "USA").values(null, "Happy Valley", "USA")
Query OK, 2 item affected (0.03 sec)
```

When you specify columns using the `insert()` method, the number of values must match the number of columns. In the previous example, you must supply three values to match the three columns specified.

Related Information

- See TableInsertFunction for the full syntax definition.

20.3.4.2 Select Tables

You can use the `select()` method to query for and return records from a table in a database. The X DevAPI provides additional methods to use with the `select()` method to filter and sort the returned records.

MySQL provides the following operators to specify search conditions: `OR` (`||`), `AND` (`&&`), `XOR`, `IS`, `NOT`, `BETWEEN`, `IN`, `LIKE`, `!=`, `<>`, `>`, `>=`, `<`, `<=`, `&`, `|`, `<<`, `>>`, `+`, `-`, `*`, `/`, `~`, and `%`.

Select All Records

To issue a query that returns all records from an existing table, use the `select()` method without specifying search conditions. The following example selects all records from the city table in the `world_x` database.

Note

Limit the use of the empty `select()` method to interactive statements. Always use explicit column-name selections in your application code.

```
mysql-js> db.city.select()
+------+------------+-------------+------------+-------------------------+
| ID   | Name       | CountryCode | District   | Info                    |
+------+------------+-------------+------------+-------------------------+
|    1 | Kabul      | AFG         | Kabol      |{"Population": 1780000}  |
|    2 | Qandahar   | AFG         | Qandahar   |{"Population": 237500}   |
|    3 | Herat      | AFG         | Herat      |{"Population": 186800}   |
...    ...          ...           ...          ...
| 4079 | Rafah      | PSE         | Rafah      |{"Population": 92020}    |
+------+------- ----+-------------+------------+-------------------------+
4082 rows in set (0.01 sec)
```

An empty set (no matching records) returns the following information:

```
Empty set (0.00 sec)
```

Filter Searches

To issue a query that returns a set of table columns, use the `select()` method and specify the columns to return between square brackets. This query returns the Name and CountryCode columns from the city table.

```
mysql-js> db.city.select(["Name", "CountryCode"])
+-------------------+-------------+
| Name              | CountryCode |
+-------------------+-------------+
| Kabul             | AFG         |
| Qandahar          | AFG         |
| Herat             | AFG         |
| Mazar-e-Sharif    | AFG         |
| Amsterdam         | NLD         |
...                 ...
| Rafah             | PSE         |
| Olympia           | USA         |
| Little Falls      | USA         |
| Happy Valley      | USA         |
+-------------------+-------------+
4082 rows in set (0.00 sec)
```

To issue a query that returns rows matching specific search conditions, use the `where()` method to include those conditions. For example, the following example returns the names and country codes of the cities that start with the letter Z.

```
mysql-js> db.city.select(["Name", "CountryCode"]).where("Name like 'Z%'")
+-------------------+-------------+
| Name              | CountryCode |
+-------------------+-------------+
| Zaanstad          | NLD         |
| Zoetermeer        | NLD         |
| Zwolle            | NLD         |
| Zenica            | BIH         |
| Zagazig           | EGY         |
| Zaragoza          | ESP         |
| Zamboanga         | PHL         |
| Zahedan           | IRN         |
| Zanjan            | IRN         |
| Zabol             | IRN         |
| Zama              | JPN         |
| Zhezqazghan       | KAZ         |
| Zhengzhou         | CHN         |
...                 ...
| Zeleznogorsk      | RUS         |
+-------------------+-------------+
59 rows in set (0.00 sec)
```

You can separate a value from the search condition by using the `bind()` method. For example, instead of using "Name = 'Z%' " as the condition, substitute a named placeholder consisting of a colon followed by a name that begins with a letter, such as *name*. Then include the placeholder and value in the `bind()` method as follows:

```
mysql-js> db.city.select(["Name", "CountryCode"]).
             where("Name like :name").bind("name", "Z%")
```

Tip

Within a program, binding enables you to specify placeholders in your expressions, which are filled in with values before execution and can benefit from automatic escaping, as appropriate.

Always use binding to sanitize input. Avoid introducing values in queries using string concatenation, which can produce invalid input and, in some cases, can cause security issues.

Project Results

To issue a query using the `AND` operator, add the operator between search conditions in the `where()` method.

```
mysql-js> db.city.select(["Name", "CountryCode"]).
              where("Name like 'Z%' and CountryCode = 'CHN'")
+----------------+-------------+
| Name           | CountryCode |
+----------------+-------------+
| Zhengzhou      | CHN         |
| Zibo           | CHN         |
| Zhangjiakou    | CHN         |
| Zhuzhou        | CHN         |
| Zhangjiang     | CHN         |
| Zigong         | CHN         |
| Zaozhuang      | CHN         |
...              ...
| Zhangjiagang   | CHN         |
+----------------+-------------+
22 rows in set (0.01 sec)
```

To specify multiple conditional operators, you can enclose the search conditions in parenthesis to change the operator precedence. The following example demonstrates the placement of `AND` and `OR` operators.

```
mysql-js> db.city.select(["Name", "CountryCode"]).
where("Name like 'Z%' and (CountryCode = 'CHN' or CountryCode = 'RUS')")
+------------------+-------------+
| Name             | CountryCode |
+------------------+-------------+
| Zhengzhou        | CHN         |
| Zibo             | CHN         |
| Zhangjiakou      | CHN         |
| Zhuzhou          | CHN         |
...                ...
| Zeleznogorsk     | RUS         |
+------------------+-------------+
29 rows in set (0.01 sec)
```

Limit, Order, and Offset Results

You can apply the `limit()`, `orderBy()`, and `offSet()` methods to manage the number and order of records returned by the `select()` method.

To specify the number of records included in a result set, append the `limit()` method with a value to the `select()` method. For example, the following query returns the first five records in the country table.

```
mysql-js> db.country.select(["Code", "Name"]).limit(5)
+------+-------------+
| Code | Name        |
+------+-------------+
| ABW  | Aruba       |
| AFG  | Afghanistan |
| AGO  | Angola      |
| AIA  | Anguilla    |
| ALB  | Albania     |
+------+-------------+
5 rows in set (0.00 sec)
```

To specify an order for the results, append the `orderBy()` method to the `select()` method. Pass to the `orderBy()` method a list of one or more columns to sort by and, optionally, the descending (`desc`) or ascending (`asc`) attribute as appropriate. Ascending order is the default order type.

For example, the following query sorts all records by the Name column and then returns the first three records in descending order .

```
mysql-js> db.country.select(["Code", "Name"]).orderBy(["Name desc"]).limit(3)
+------+------------+
| Code | Name       |
+------+------------+
| ZWE  | Zimbabwe   |
| ZMB  | Zambia     |
| YUG  | Yugoslavia |
+------+------------+
3 rows in set (0.00 sec)
```

By default, the `limit()` method starts from the first record in the table. You can use the `offset()` method to change the starting record. For example, to ignore the first record and return the next three records matching the condition, pass to the `offset()` method a value of 1.

```
mysql-js> db.country.select(["Code", "Name"]).orderBy(["Name desc"]).limit(3).offset(1)
+------+------------+
| Code | Name       |
+------+------------+
| ZMB  | Zambia     |
| YUG  | Yugoslavia |
| YEM  | Yemen      |
+------+------------+
3 rows in set (0.00 sec)
```

Related Information

- The MySQL Reference Manual provides detailed documentation on functions and operators.
- See TableSelectFunction for the full syntax definition.

20.3.4.3 Update Tables

You can use the `update()` method to modify one or more records in a table. The `update()` method works by filtering a query to include only the records to be updated and then applying the operations you specify to those records.

To replace a city name in the city table, pass to the `set()` method the new city name. Then, pass to the `where()` method the city name to locate and replace. The following example replaces the city Peking with Beijing.

```
mysql-js> db.city.update().set("Name", "Beijing").where("Name = 'Peking'")
Query OK, 1 item affected (0.04 sec)
```

Use the `select()` method to verify the change.

```
mysql-js> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).where("Name = 'Beijing'")
+------+-----------+-------------+----------+----------------------------+
| ID   | Name      | CountryCode | District | Info                       |
+------+-----------+-------------+----------+----------------------------+
| 1891 | Beijing   | CHN         | Peking   | {"Population": 7472000}    |
+------+-----------+-------------+----------+----------------------------+
1 row in set (0.00 sec)
```

Related Information

- See TableUpdateFunction for the full syntax definition.

20.3.4.4 Delete Tables

You can use the `delete()` method to remove some or all records from a table in a database. The X DevAPI provides additional methods to use with the `delete()` method to filter and order the records to be deleted.

Delete Records Using Conditions

The example that follows passes search conditions to the `delete()` method. All records matching the condition will be deleted from the city table. In this example, one record matches the condition.

```
mysql-js> db.city.delete().where("Name = 'Olympia'")
Query OK, 1 item affected (0.01 sec)
```

Delete the First Record

To delete the first record in the city table, use the `limit()` method with a value of 1.

```
mysql-js> db.city.delete().limit(1)
Query OK, 1 item affected (0.02 sec)
```

Delete All Records in a Table

You can delete all records in a table. To do so, use the `delete()` method without specifying a search condition.

Caution

Use care when you delete records without specifying a search condition. This action will delete all records from the table.

Drop a Table

The `dropCollection()` method is also used in MySQL Shell to drop a relational table from a database. For example, to drop the citytest table from the `world_x` database, issue:

```
mysql-js> session.dropCollection("world_x", "citytest")
Query OK (0.04 sec)
```

Related Information

- See TableDeleteFunction for the full syntax definition.
- See Section 20.3.2, “Download and Import world_x Database” for instructions to recreate the `world_x` database.

20.3.5 Documents in Tables

In MySQL, a table may contain traditional relational data, JSON values, or both. You can combine traditional data with JSON documents by storing the documents in columns having a native `JSON` data type.

Examples in this section use the city table in the `world_x` database.

city Table Description

The city table has five columns (or fields).

```
+---------------+------------+-------+-------+---------+------------------+
| Field         | Type       | Null  | Key   | Default | Extra            |
+---------------+------------+-------+-------+---------+------------------+
| ID            | int(11)    | NO    | PRI   | null    | auto_increment   |
| Name          | char(35)   | NO    |       |         |                  |
| CountryCode   | char(3)    | NO    |       |         |                  |
| District      | char(20)   | NO    |       |         |                  |
| Info          | json       | YES   |       | null    |                  |
+---------------+------------+-------+-------+---------+------------------+
```

Insert a Record

To insert a document into the column of a table, pass to the `values()` method a well-formed JSON document in the correct order. In the following example, a document is passed as the final value to be inserted into the Info column.

```
mysql-js> db.city.insert().
values(null, "San Francisco", "USA", "California", '{"Population":830000}')
Query OK, 1 item affected (0.01 sec)
```

Select a Record

You can issue a query with a search condition that evaluates document values in the expression.

```
mysql-js> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).
  where("CountryCode = :country and Info->'$.Population' > 1000000").
  bind('country', 'USA')
+------+---------------+-------------+----------------+-----------------------------+
| ID   | Name          | CountryCode | District       | Info                        |
+------+---------------+-------------+----------------+-----------------------------+
| 3793 | New York      | USA         | New York       | {"Population": 8008278}     |
| 3794 | Los Angeles   | USA         | California     | {"Population": 3694820}     |
| 3795 | Chicago       | USA         | Illinois       | {"Population": 2896016}     |
| 3796 | Houston       | USA         | Texas          | {"Population": 1953631}     |
| 3797 | Philadelphia  | USA         | Pennsylvania   | {"Population": 1517550}     |
| 3798 | Phoenix       | USA         | Arizona        | {"Population": 1321045}     |
| 3799 | San Diego     | USA         | California     | {"Population": 1223400}     |
| 3800 | Dallas        | USA         | Texas          | {"Population": 1188580}     |
| 3801 | San Antonio   | USA         | Texas          | {"Population": 1144646}     |
+------+---------------+-------------+----------------+-----------------------------+
9 rows in set (0.01 sec)
```

Related Information

- See Working with Relational Tables and Documents for more information.
- See Section 11.6, “The JSON Data Type” for a detailed description of the data type.

20.4 Python Quick-Start Guide: MySQL Shell for Document Store

This quick-start guide provides instructions to begin prototyping document store applications interactively with MySQL Shell. The guide includes the following topics:

- Introduction to MySQL functionality, MySQL Shell, and the `world_x` example schema.
- Operations to manage collections and documents.
- Operations to manage relational tables.
- Operations that apply to documents within tables.

To follow this quick-start guide you need a MySQL server with X Plugin installed, the default in 8.0, and MySQL Shell to use as the client. MySQL Shell 8.0 (part of MySQL 8.0) provides more in-depth information about MySQL Shell. The Document Store is accessed using X DevAPI, and MySQL Shell provides this API in both JavaScript and Python.

Related Information

- MySQL Shell 8.0 (part of MySQL 8.0) provides more in-depth information about MySQL Shell.
- See Installing MySQL Shell and Section 20.5, “X Plugin” for more information about the tools used in this quick-start guide.
- X DevAPI User Guide provides more examples of using X DevAPI to develop applications which use Document Store.
- A JavaScript quick-start guide is also available.

20.4.1 MySQL Shell

This quick-start guide assumes a certain level of familiarity with MySQL Shell. The following section is a high level overview, see the MySQL Shell documentation for more information. MySQL Shell is a unified scripting interface to MySQL Server. It supports scripting in JavaScript and Python. JavaScript is the default processing mode.

Start MySQL Shell

After you have installed and started MySQL server, connect MySQL Shell to the server instance. You need to know the address of the MySQL server instance you plan to connect to. To be able to use the instance as a Document Store, the server instance must have X Plugin installed and you should connect to the server using X Protocol. For example to connect to the instance `ds1.example.com` on the default `mysqlx_port` of 33060 use the network string *`user`*`@ds1.example.com:33060`.

Tip

If you connect to the instance using MySQL classic protocol, for example by using the default `port` of 3306 instead of the `mysqlx_port`, you *cannot* use the Document Store functionality shown in this tutorial. For example the `db` global object is not populated. To use the Document Store, always connect using X Protocol.

If MySQL Shell is not already running, open a terminal window and issue:

```
mysqlsh user@ds1.example.com:33060/world_x
```

Alternatively, if MySQL Shell is already running use the `\connect` command by issuing:

```
\connect user@ds1.example.com:33060/world_x
```

You need to specify the address of the MySQL server instance which you want to connect MySQL Shell to. For example in the previous example:

- *`user`* represents the user name of your MySQL account.
- `ds1.example.com` is the hostname of the server instance running MySQL. Replace this with the hostname of the MySQL server instance you are using as a Document Store.

- The default schema for this session is `world_x`. For instructions on setting up the `world_x` schema, see Section 20.4.2, “Download and Import world_x Database”.

For more information, see Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”.

Once MySQL Shell opens, the `mysql-js>` prompt indicates that the active language for this session is JavaScript. To switch MySQL Shell to Python mode, use the \py command.

```
mysql-js> \py
Switching to Python mode...
mysql-py>
```

MySQL Shell supports input-line editing as follows:

- **left-arrow** and **right-arrow** keys move horizontally within the current input line.
- **up-arrow** and **down-arrow** keys move up and down through the set of previously entered lines.
- **Backspace** deletes the character before the cursor and typing new characters enters them at the cursor position.
- **Enter** sends the current input line to the server.

Get Help for MySQL Shell

Type `mysqlsh --help` at the prompt of your command interpreter for a list of command-line options.

```
mysqlsh --help
```

Type `\help` at the MySQL Shell prompt for a list of available commands and their descriptions.

```
mysql-py> \help
```

Type `\help` followed by a command name for detailed help about an individual MySQL Shell command. For example, to view help on the `\connect` command, issue:

```
mysql-py> \help \connect
```

Quit MySQL Shell

To quit MySQL Shell, issue the following command:

```
mysql-py> \quit
```

Related Information

- See Interactive Code Execution for an explanation of how interactive code execution works in MySQL Shell.
- See Getting Started with MySQL Shell to learn about session and connection alternatives.

20.4.2 Download and Import world_x Database

As part of this quick-start guide, an example schema is provided which is referred to as the `world_x` schema. Many of the examples demonstrate Document Store functionality using this schema. Start your MySQL server so that you can load the `world_x` schema, then follow these steps:

1. Download world_x-db.zip.
2. Extract the installation archive to a temporary location such as `/tmp/`. Unpacking the archive results in a single file named `world_x.sql`.
3. Import the `world_x.sql` file to your server. You can either:
   - Start MySQL Shell in SQL mode and import the file by issuing:

     ```
     mysqlsh -u root --sql --file /tmp/world_x-db/world_x.sql
     Enter password: ****
     ```

   - Set MySQL Shell to SQL mode while it is running and source the schema file by issuing:

     ```
     \sql
     Switching to SQL mode... Commands end with ;
     \source /tmp/world_x-db/world_x.sql
     ```

   Replace `/tmp/` with the path to the `world_x.sql` file on your system. Enter your password if prompted. A non-root account can be used as long as the account has privileges to create new schemas.

The world_x Schema

The `world_x` example schema contains the following JSON collection and relational tables:

- Collection
  - `countryinfo`: Information about countries in the world.
- Tables
  - `country`: Minimal information about countries of the world.
  - `city`: Information about some of the cities in those countries.
  - `countrylanguage`: Languages spoken in each country.

Related Information

- MySQL Shell Sessions explains session types.

20.4.3 Documents and Collections

When you are using MySQL as a Document Store, collections are containers within a schema that you can create, list, and drop. Collections contain JSON documents that you can add, find, update, and remove.

The examples in this section use the `countryinfo` collection in the `world_x` schema. For instructions on setting up the `world_x` schema, see Section 20.4.2, “Download and Import world_x Database”.

Documents

In MySQL, documents are represented as JSON objects. Internally, they are stored in an efficient binary format that enables fast lookups and updates.

- Simple document format for Python:

```
{"field1": "value", "field2" : 10, "field 3": null}
```

An array of documents consists of a set of documents separated by commas and enclosed within `[` and `]` characters.

- Simple array of documents for Python:

```
[{"Name": "Aruba", "_id": "ABW"}, {"Name": "Angola", "_id": "AGO"}]
```

MySQL supports the following Python value types in JSON documents:

- numbers (integer and floating point)
- strings
- boolean (False and True)
- None
- arrays of more JSON values
- nested (or embedded) objects of more JSON values

Collections

Collections are containers for documents that share a purpose and possibly share one or more indexes. Each collection has a unique name and exists within a single schema.

The term schema is equivalent to a database, which means a group of database objects as opposed to a relational schema, used to enforce structure and constraints over data. A schema does not enforce conformity on the documents in a collection.

In this quick-start guide:

- Basic objects include:

Object form	Description
`db`	`db` is a global variable assigned to the current active schema. When you want to run operations against the schema, for example to retrieve a collection, you use methods available for the `db` variable.
`db.get_collections()`	db.get_collections() holds a list of collections in the schema. Use the list to get references to collection objects, iterate over them, and so on.

- Basic operations scoped by collections include:

Operation form	Description
`db.`*`name`*`.add()`	The add() method inserts one document or a list of documents into the named collection.
`db.`*`name`*`.find()`	The find() method returns some or all documents in the named collection.
`db.`*`name`*`.modify()`	The modify() method updates documents in the named collection.

Operation form	Description
`db.name.remove()`	The remove() method deletes one document or a list of documents from the named collection.

Related Information

- See Working with Collections for a general overview.
- CRUD EBNF Definitions provides a complete list of operations.

20.4.3.1 Create, List, and Drop Collections

In MySQL Shell, you can create new collections, get a list of the existing collections in a schema, and remove an existing collection from a schema. Collection names are case-sensitive and each collection name must be unique.

Confirm the Schema

To show the value that is assigned to the schema variable, issue:

```
mysql-py> db
```

If the schema value is not `Schema:world_x`, then set the `db` variable by issuing:

```
mysql-py> \use world_x
```

Create a Collection

To create a new collection in an existing schema, use the `db` object's `createCollection()` method. The following example creates a collection called `flags` in the `world_x` schema.

```
mysql-py> db.create_collection("flags")
```

The method returns a collection object.

```
<Collection:flags>
```

List Collections

To display all collections in the `world_x` schema, use the `db` object's `get_collections()` method. Collections returned by the server you are currently connected to appear between brackets.

```
mysql-py> db.get_collections()
[
    <Collection:countryinfo>,
    <Collection:flags>
]
```

Drop a Collection

To drop an existing collection from a schema, use the `db` object's `drop_collection()` method. For example, to drop the `flags` collection from the current schema, issue:

```
mysql-py> db.drop_collection("flags")
```

The `drop_collection()` method is also used in MySQL Shell to drop a relational table from a schema.

Related Information

- See Collection Objects for more examples.

20.4.3.2 Working with Collections

To work with the collections in a schema, use the `db` global object to access the current schema. In this example we are using the `world_x` schema imported previously, and the `countryinfo` collection. Therefore, the format of the operations you issue is `db.collection_name.operation`, where *`collection_name`* is the name of the collection which the operation is executed against. In the following examples, the operations are executed against the `countryinfo` collection.

Add a Document

Use the `add()` method to insert one document or a list of documents into an existing collection. Insert the following document into the `countryinfo` collection. As this is multi-line content, press **Enter** twice to insert the document.

```
mysql-py> db.countryinfo.add(
 {
    GNP: .6,
    IndepYear: 1967,
    Name: "Sealand",
    demographics: {
        LifeExpectancy: 79,
        Population: 27
    },
    geography: {
        Continent: "Europe",
        Region: "British Islands",
        SurfaceArea: 193
    },
    government: {
        GovernmentForm: "Monarchy",
        HeadOfState: "Michael Bates"
    }
  }
)
```

The method returns the status of the operation.

Each document requires an identifier field called `_id`. The value of the `_id` field must be unique among all documents in the same collection. In MySQL 8.0.11 and higher, document IDs are generated by the server, not the client, so MySQL Shell does not automatically set an `_id` value. A MySQL server at 8.0.11 or higher sets an `_id` value if the document does not contain the `_id` field. A MySQL server at an earlier 8.0 release or at 5.7 does not set an `_id` value in this situation, so you must specify it explicitly. If you do not, MySQL Shell returns error 5115 `Document is missing a required field`.

Related Information

- See CollectionAddFunction for the full syntax definition.
- See Understanding Document IDs.

20.4.3.3 Find Documents

You can use the `find()` method to query for and return documents from a collection in a schema. MySQL Shell provides additional methods to use with the `find()` method to filter and sort the returned documents.

MySQL provides the following operators to specify search conditions: `OR` (`||`), `AND` (`&&`), `XOR`, `IS`, `NOT`, `BETWEEN`, `IN`, `LIKE`, `!=`, `<>`, `>`, `>=`, `<`, `<=`, `&`, `|`, `<<`, `>>`, `+`, `-`, `*`, `/`, `~`, and `%`.

Find All Documents in a Collection

To return all documents in a collection, use the `find()` method without specifying search conditions. For example, the following operation returns all documents in the `countryinfo` collection.

```
mysql-py> db.countryinfo.find()
[
     {
          "GNP": 828,
          "IndepYear": null,
          "Name": "Aruba",
          "_id": "ABW",
          "demographics": {
              "LifeExpectancy": 78.4000015258789,
              "Population": 103000
          },
          "geography": {
              "Continent": "North America",
              "Region": "Caribbean",
              "SurfaceArea": 193
          },
          "government": {
              "GovernmentForm": "Nonmetropolitan Territory of The Netherlands",
              "HeadOfState": "Beatrix"
          }
          ...
      }
 ]
240 documents in set (0.00 sec)
```

The method produces results that contain operational information in addition to all documents in the collection.

An empty set (no matching documents) returns the following information:

```
Empty set (0.00 sec)
```

Filter Searches

You can include search conditions with the `find()` method. The syntax for expressions that form a search condition is the same as that of traditional MySQL Chapter 12, *Functions and Operators*. You must enclose all expressions in quotes. For the sake of brevity, some of the examples do not display output.

A simple search condition could consist of the `Name` field and a value we know is in a document. The following example returns a single document:

```
mysql-py> db.countryinfo.find("Name = 'Australia'")
[
    {
        "GNP": 351182,
        "IndepYear": 1901,
        "Name": "Australia",
        "_id": "AUS",
        "demographics": {
            "LifeExpectancy": 79.80000305175781,
            "Population": 18886000
        },
```

```
        "geography": {
            "Continent": "Oceania",
            "Region": "Australia and New Zealand",
            "SurfaceArea": 7741220
        },
        "government": {
            "GovernmentForm": "Constitutional Monarchy, Federation",
            "HeadOfState": "Elisabeth II"
        }
    }
]
```

The following example searches for all countries that have a GNP higher than $500 billion. The `countryinfo` collection measures GNP in units of million.

```
mysql-py> db.countryinfo.find("GNP > 500000")
...[output removed]
10 documents in set (0.00 sec)
```

The Population field in the following query is embedded within the demographics object. To access the embedded field, use a period between demographics and Population to identify the relationship. Document and field names are case-sensitive.

```
mysql-py> db.countryinfo.find("GNP > 500000 and demographics.Population < 100000000")
...[output removed]
6 documents in set (0.00 sec)
```

Arithmetic operators in the following expression are used to query for countries with a GNP per capita higher than $30000. Search conditions can include arithmetic operators and most MySQL functions.

Note

Seven documents in the `countryinfo` collection have a population value of zero. Therefore warning messages appear at the end of the output.

```
mysql-py> db.countryinfo.find("GNP*1000000/demographics.Population > 30000")
...[output removed]
9 documents in set, 7 warnings (0.00 sec)
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
Warning (Code 1365): Division by 0
```

You can separate a value from the search condition by using the `bind()` method. For example, instead of specifying a hard-coded country name as the condition, substitute a named placeholder consisting of a colon followed by a name that begins with a letter, such as *country*. Then use the `bind(placeholder, value)` method as follows:

```
mysql-py> db.countryinfo.find("Name = :country").bind("country", "Italy")
[
    {
        "GNP": 1161755,
        "IndepYear": 1861,
        "Name": "Italy",
        "_id": "ITA",
        "demographics": {
```

```
            "LifeExpectancy": 79,
            "Population": 57680000
        },
        "geography": {
            "Continent": "Europe",
            "Region": "Southern Europe",
            "SurfaceArea": 301316
        },
        "government": {
            "GovernmentForm": "Republic",
            "HeadOfState": "Carlo Azeglio Ciampi"
        }
    }
]
1 document in set (0.01 sec)
```

Tip

Within a program, binding enables you to specify placeholders in your expressions, which are filled in with values before execution and can benefit from automatic escaping, as appropriate.

Always use binding to sanitize input. Avoid introducing values in queries using string concatenation, which can produce invalid input and, in some cases, can cause security issues.

You can use placeholders and the `bind()` method to create saved searches which you can then call with different values. For example to create a saved search for a country:

```
mysql-py> myFind = db.countryinfo.find("Name = :country")
mysql-py> myFind.bind('country', 'France')
[
    {
        "GNP": 1424285,
        "IndepYear": 843,
        "Name": "France",
        "_id": "FRA",
        "demographics": {
            "LifeExpectancy": 78.80000305175781,
            "Population": 59225700
        },
        "geography": {
            "Continent": "Europe",
            "Region": "Western Europe",
            "SurfaceArea": 551500
        },
        "government": {
            "GovernmentForm": "Republic",
            "HeadOfState": "Jacques Chirac"
        }
    }
]
1 document in set (0.0028 sec)

mysql-py> myFind.bind('country', 'Germany')
[
    {
        "GNP": 2133367,
        "IndepYear": 1955,
        "Name": "Germany",
        "_id": "DEU",
        "demographics": {
            "LifeExpectancy": 77.4000015258789,
            "Population": 82164700
```

```
        },
        "geography": {
            "Continent": "Europe",
            "Region": "Western Europe",
            "SurfaceArea": 357022
        },
        "government": {
            "GovernmentForm": "Federal Republic",
            "HeadOfState": "Johannes Rau"
        }
    }
]
1 document in set (0.0026 sec)
```

Project Results

You can return specific fields of a document, instead of returning all the fields. The following example returns the GNP and Name fields of all documents in the `countryinfo` collection matching the search conditions.

Use the `fields()` method to pass the list of fields to return.

```
mysql-py> db.countryinfo.find("GNP > 5000000").fields(["GNP", "Name"])
[
    {
        "GNP": 8510700,
        "Name": "United States"
    }
]
1 document in set (0.00 sec)
```

In addition, you can alter the returned documents—adding, renaming, nesting and even computing new field values—with an expression that describes the document to return. For example, alter the names of the fields with the following expression to return only two documents.

```
mysql-py> db.countryinfo.find().\
fields(mysqlx.expr('{"Name": upper(Name), "GNPPerCapita": GNP*1000000/demographics.Population}')).\
limit(2)
[
    {
        "GNPPerCapita": 8038.834951456311,
        "Name": "ARUBA"
    },
    {
        "GNPPerCapita": 263.0281690140845,
        "Name": "AFGHANISTAN"
    }
]
2 documents in set (0.00 sec)
```

Limit, Sort, and Skip Results

You can apply the `limit()`, `sort()`, and `skip()` methods to manage the number and order of documents returned by the `find()` method.

To specify the number of documents included in a result set, append the `limit()` method with a value to the `find()` method. The following query returns the first five documents in the `countryinfo` collection.

```
mysql-py> db.countryinfo.find().limit(5)
... [output removed]
```

```
5 documents in set (0.00 sec)
```

To specify an order for the results, append the `sort()` method to the `find()` method. Pass to the `sort()` method a list of one or more fields to sort by and, optionally, the descending (`desc`) or ascending (`asc`) attribute as appropriate. Ascending order is the default order type.

For example, the following query sorts all documents by the IndepYear field and then returns the first eight documents in descending order.

```
mysql-py> db.countryinfo.find().sort(["IndepYear desc"]).limit(8)
... [output removed]
8 documents in set (0.00 sec)
```

By default, the `limit()` method starts from the first document in the collection. You can use the `skip()` method to change the starting document. For example, to ignore the first document and return the next eight documents matching the condition, pass to the `skip()` method a value of 1.

```
mysql-py> db.countryinfo.find().sort(["IndepYear desc"]).limit(8).skip(1)
... [output removed]
8 documents in set (0.00 sec)
```

Related Information

- The MySQL Reference Manual provides detailed documentation on functions and operators.
- See CollectionFindFunction for the full syntax definition.

20.4.3.4 Modify Documents

You can use the `modify()` method to update one or more documents in a collection. The X DevAPI provides additional methods for use with the `modify()` method to:

- Set and unset fields within documents.
- Append, insert, and delete arrays.
- Bind, limit, and sort the documents to be modified.

Set and Unset Fields

The `modify()` method works by filtering a collection to include only the documents to be modified and then applying the operations that you specify to those documents.

In the following example, the `modify()` method uses the search condition to identify the document to change and then the `set()` method replaces two values within the nested demographics object.

```
mysql-py> db.countryinfo.modify("_id = 'SEA'").\
set("demographics", {"LifeExpectancy": 78, "Population": 28})
Query OK, 1 item affected (0.04 sec)
```

After you modify a document, use the `find()` method to verify the change.

To remove content from a document, use the `modify()` and `unset()` methods. For example, the following query removes the GNP from a document that matches the search condition.

```
mysql-py> db.countryinfo.modify("Name = 'Sealand'").unset("GNP")
Query OK, 1 item affected (0.01 sec)
```

Use the `find()` method to verify the change.

```
mysql-py> db.countryinfo.find("Name = 'Sealand'")
[
    {
        "IndepYear": 1967,
        "Name": "Sealand",
        "_id": "SEA",
        "demographics": {
            "LifeExpectancy": 78,
            "Population": 28
        },
        "geography": {
            "Continent": "Europe",
            "Region": "British Islands",
            "SurfaceArea": 193
        },
        "government": {
            "GovernmentForm": "Monarchy",
            "HeadOfState": "Michael Bates"
        }
    }
]
1 document in set (0.00 sec)
```

Append, Insert, and Delete Arrays

To append an element to an array field, or insert, or delete elements in an array, use the `array_append()`, `array_insert()`, or `array_delete()` methods. The following examples modify the `countryinfo` collection to enable tracking of international airports.

The first example uses the `modify()` and `set()` methods to create a new Airports field in all documents.

Caution

Use care when you modify documents without specifying a search condition. This action will modify all documents in the collection.

```
mysql-py> db.countryinfo.modify(True).set("Airports", [])
Query OK, 240 items affected (0.07 sec)
```

With the Airports field added, the next example uses the `array_append()` method to add a new airport to one of the documents. *$.Airports* in the following example represents the Airports field of the current document.

```
mysql-py> db.countryinfo.modify("Name = 'France'").array_append("$.Airports", "ORY")
Query OK, 1 item affected (0.02 sec)
```

Use **`db.countryinfo.find("Name = 'France'")`** to see the change.

To insert an element at a different position in the array, use the `array_insert()` method to specify which index to insert in the path expression. In this case, the index is 0, or the first element in the array.

```
mysql-py> db.countryinfo.modify("Name = 'France'").array_insert("$.Airports[0]", "CDG")
Query OK, 1 item affected (0.04 sec)
```

To delete an element from the array, you must pass to the `array_delete()` method the index of the element to be deleted.

```
mysql-py> db.countryinfo.modify("Name = 'France'").array_delete("$.Airports[1]")
Query OK, 1 item affected (0.03 sec)
```

Related Information

- The MySQL Reference Manual provides instructions to help you search for and modify JSON values.
- See CollectionModifyFunction for the full syntax definition.

20.4.3.5 Remove Documents

You can use the `remove()` method to delete some or all documents from a collection in a schema. The X DevAPI provides additional methods for use with the `remove()` method to filter and sort the documents to be removed.

Remove Documents Using Conditions

The example that follows passes a search condition to the `remove()` method. All documents matching the condition are removed from the `countryinfo` collection. In this example, one document matches the condition.

```
mysql-py> db.countryinfo.remove("_id = 'SEA'")
Query OK, 1 item affected (0.02 sec)
```

Remove the First Document

To remove the first document in the `countryinfo` collection, use the `limit()` method with a value of 1.

```
mysql-py> db.countryinfo.remove(True).limit(1)
Query OK, 1 item affected (0.03 sec)
```

Remove the Last Document in an Order

The following example removes the last document in the `countryinfo` collection by country name.

```
mysql-py> db.countryinfo.remove(True).sort(["Name desc"]).limit(1)
Query OK, 1 item affected (0.02 sec)
```

Remove All Documents in a Collection

You can remove all documents in a collection. To do so, use the `remove(True)` method without specifying a search condition.

Caution

Use care when you remove documents without specifying a search condition. This action will delete all documents from the collection.

Related Information

- See CollectionRemoveFunction for the full syntax definition.
- See Section 20.4.2, “Download and Import world_x Database” for instructions to recreate the `world_x` schema.

20.4.3.6 Create and Drop Indexes

Indexes are used to find documents with specific field values quickly. Without an index, MySQL must begin with the first document and then read through the entire collection to find the relevant fields. The larger the collection, the more this costs. If a collection is large and queries on a specific field are common, then consider creating an index on a specific field inside a document.

For example, the following query will perform better with an index:

```
mysql-js> db.countryinfo.find("demographics.Population < 100")
...[output removed]
8 documents in set (0.00 sec)
```

The `create_index()` method creates an index that you can define as nonunique or unique. Use the `field()` method to chain the fields that should be indexed. The `execute()` method is required to create or drop an index.

In MySQL, the `_id` field is equivalent to a primary key by default.

Add a Nonunique Index

To create a nonunique index, pass to the `create_index()` method an index name. Duplicate index names are prohibited.

In the following example, the first parameter of the `field()` method specifies the Population field inside the demographics object and the next parameter indicates that the field should be indexed as an Integer numeric value. The last parameter indicates whether the field should require the NOT NULL constraint. If the value is `False`, the field can contain `NULL` values.

```
mysql-js> db.countryinfo.create_index("pop").\
field("demographics.Population", "INTEGER", False).execute()
Query OK (0.04 sec)
```

Add a Unique Index

To create a unique index, pass to the `create_index()` method an index name and the `mysqlx.IndexType.UNIQUE` type. Country `"Name"` is another common field in the `countryinfo` collection to index. In the following example, `"Text(40)"` represents the number of characters to index and `True` indicates that the field cannot contain any `NULL` values.

```
mysql-js> db.countryinfo.create_index("name", mysqlx.IndexType.UNIQUE).\
field("Name", "TEXT(40)", True).execute()
Query OK (0.04 sec)
```

Drop an Index

To drop an index, pass to the `drop_index()` method the name of the index to drop. For example, you can drop the “pop” index as follows:

```
mysql-js> db.countryinfo.drop_index("pop").execute()
Query OK (0.58 sec)
```

Related Information

- See Indexing Collections for more information.

- See Collection Index Management Functions for the full syntax definition.

20.4.4 Relational Tables

You can also use X DevAPI to work with relational tables. In MySQL, each relational table is associated with a particular storage engine. The examples in this section use `InnoDB` tables in the `world_x` schema.

Confirm the Schema

To show the schema that is assigned to the `db` global variable, issue `db`.

```
mysql-py> db
<Schema:world_x>
```

If the returned value is not `Schema:world_x`, set the `db` variable as follows:

```
mysql-py> \use world_x
Schema `world_x` accessible through db.
```

Show All Tables

To display all relational tables in the `world_x` schema, use the `get_tables()` method on the `db` object.

```
mysql-py> db.get_tables()
{
    "city": <Table:city>,
    "country": <Table:country>,
    "countrylanguage": <Table:countrylanguage>
}
```

Basic Table Operations

Basic operations scoped by tables include:

Operation form	Description
`db.`*`name`*`.insert()`	The insert() method inserts one or more records into the named table.
`db.`*`name`*`.select()`	The select() method returns some or all records in the named table.
`db.`*`name`*`.update()`	The update() method updates records in the named table.
`db.`*`name`*`.delete()`	The delete() method deletes one or more records from the named table.

Related Information

- See Working with Relational Tables for more information.
- CRUD EBNF Definitions provides a complete list of operations.
- See Section 20.4.2, “Download and Import world_x Database” for instructions on setting up the `world_x` schema sample.

20.4.4.1 Insert Records into Tables

You can use the `insert()` method with the `values()` method to insert records into an existing relational table. The `insert()` method accepts individual columns or all columns in the table. Use one or more `values()` methods to specify the values to be inserted.

Insert a Complete Record

To insert a complete record, pass to the `insert()` method all columns in the table. Then pass to the `values()` method one value for each column. For example, to add a new record to the city table in the `world_x` database, insert the following record and press **Enter** twice.

```
mysql-py> db.city.insert("ID", "Name", "CountryCode", "District", "Info").\
values(None, "Olympia", "USA", "Washington", '{"Population": 5000}')
Query OK, 1 item affected (0.01 sec)
```

The city table has five columns: ID, Name, CountryCode, District, and Info. Each value must match the data type of the column it represents.

Insert a Partial Record

The following example inserts values into the ID, Name, and CountryCode columns of the city table.

```
mysql-py> db.city.insert("ID", "Name", "CountryCode").\
values(None, "Little Falls", "USA").values(None, "Happy Valley", "USA")
Query OK, 2 item affected (0.03 sec)
```

When you specify columns using the `insert()` method, the number of values must match the number of columns. In the previous example, you must supply three values to match the three columns specified.

Related Information

- See TableInsertFunction for the full syntax definition.

20.4.4.2 Select Tables

You can use the `select()` method to query for and return records from a table in a database. The X DevAPI provides additional methods to use with the `select()` method to filter and sort the returned records.

MySQL provides the following operators to specify search conditions: `OR` (`||`), `AND` (`&&`), `XOR`, `IS`, `NOT`, `BETWEEN`, `IN`, `LIKE`, `!=`, `<>`, `>`, `>=`, `<`, `<=`, `&`, `|`, `<<`, `>>`, `+`, `-`, `*`, `/`, `~`, and `%`.

Select All Records

To issue a query that returns all records from an existing table, use the `select()` method without specifying search conditions. The following example selects all records from the city table in the `world_x` database.

Note

Limit the use of the empty `select()` method to interactive statements. Always use explicit column-name selections in your application code.

```
mysql-py> db.city.select()
+------+------------+-------------+------------+-------------------------+
| ID   | Name       | CountryCode | District   | Info                    |
+------+------------+-------------+------------+-------------------------+
|    1 | Kabul      | AFG         | Kabol      |{"Population": 1780000}  |
|    2 | Qandahar   | AFG         | Qandahar   |{"Population": 237500}   |
|    3 | Herat      | AFG         | Herat      |{"Population": 186800}   |
...    ...          ...           ...          ...
| 4079 | Rafah      | PSE         | Rafah      |{"Population": 92020}    |
+------+------- ----+-------------+------------+-------------------------+
```

```
4082 rows in set (0.01 sec)
```

An empty set (no matching records) returns the following information:

```
Empty set (0.00 sec)
```

Filter Searches

To issue a query that returns a set of table columns, use the `select()` method and specify the columns to return between square brackets. This query returns the Name and CountryCode columns from the city table.

```
mysql-py> db.city.select(["Name", "CountryCode"])
+------------------+-------------+
| Name             | CountryCode |
+------------------+-------------+
| Kabul            | AFG         |
| Qandahar         | AFG         |
| Herat            | AFG         |
| Mazar-e-Sharif   | AFG         |
| Amsterdam        | NLD         |
...                ...
| Rafah            | PSE         |
| Olympia          | USA         |
| Little Falls     | USA         |
| Happy Valley     | USA         |
+------------------+-------------+
4082 rows in set (0.00 sec)
```

To issue a query that returns rows matching specific search conditions, use the `where()` method to include those conditions. For example, the following example returns the names and country codes of the cities that start with the letter Z.

```
mysql-py> db.city.select(["Name", "CountryCode"]).where("Name like 'Z%'")
+------------------+-------------+
| Name             | CountryCode |
+------------------+-------------+
| Zaanstad         | NLD         |
| Zoetermeer       | NLD         |
| Zwolle           | NLD         |
| Zenica           | BIH         |
| Zagazig          | EGY         |
| Zaragoza         | ESP         |
| Zamboanga        | PHL         |
| Zahedan          | IRN         |
| Zanjan           | IRN         |
| Zabol            | IRN         |
| Zama             | JPN         |
| Zhezqazghan      | KAZ         |
| Zhengzhou        | CHN         |
...                ...
| Zeleznogorsk     | RUS         |
+------------------+-------------+
59 rows in set (0.00 sec)
```

You can separate a value from the search condition by using the `bind()` method. For example, instead of using "Name = 'Z%' " as the condition, substitute a named placeholder consisting of a colon followed by a name that begins with a letter, such as *name*. Then include the placeholder and value in the `bind()` method as follows:

```
```

```
mysql-py> db.city.select(["Name", "CountryCode"]).\
              where("Name like :name").bind("name", "Z%")
```

Tip

Within a program, binding enables you to specify placeholders in your expressions, which are filled in with values before execution and can benefit from automatic escaping, as appropriate.

Always use binding to sanitize input. Avoid introducing values in queries using string concatenation, which can produce invalid input and, in some cases, can cause security issues.

Project Results

To issue a query using the `AND` operator, add the operator between search conditions in the `where()` method.

```
mysql-py> db.city.select(["Name", "CountryCode"]).\
              where("Name like 'Z%' and CountryCode = 'CHN'")
+---------------+-------------+
| Name          | CountryCode |
+---------------+-------------+
| Zhengzhou     | CHN         |
| Zibo          | CHN         |
| Zhangjiakou   | CHN         |
| Zhuzhou       | CHN         |
| Zhangjiang    | CHN         |
| Zigong        | CHN         |
| Zaozhuang     | CHN         |
...             ...
| Zhangjiagang  | CHN         |
+---------------+-------------+
22 rows in set (0.01 sec)
```

To specify multiple conditional operators, you can enclose the search conditions in parenthesis to change the operator precedence. The following example demonstrates the placement of `AND` and `OR` operators.

```
mysql-py> db.city.select(["Name", "CountryCode"]).\
where("Name like 'Z%' and (CountryCode = 'CHN' or CountryCode = 'RUS')")
+------------------+-------------+
| Name             | CountryCode |
+------------------+-------------+
| Zhengzhou        | CHN         |
| Zibo             | CHN         |
| Zhangjiakou      | CHN         |
| Zhuzhou          | CHN         |
...                ...
| Zeleznogorsk     | RUS         |
+------------------+-------------+
29 rows in set (0.01 sec)
```

Limit, Order, and Offset Results

You can apply the `limit()`, `order_by()`, and `offset()` methods to manage the number and order of records returned by the `select()` method.

To specify the number of records included in a result set, append the `limit()` method with a value to the `select()` method. For example, the following query returns the first five records in the country table.

```
mysql-py> db.country.select(["Code", "Name"]).limit(5)
+------+-------------+
| Code | Name        |
+------+-------------+
| ABW  | Aruba       |
| AFG  | Afghanistan |
| AGO  | Angola      |
| AIA  | Anguilla    |
| ALB  | Albania     |
+------+-------------+
5 rows in set (0.00 sec)
```

To specify an order for the results, append the `order_by()` method to the `select()` method. Pass to the `order_by()` method a list of one or more columns to sort by and, optionally, the descending (`desc`) or ascending (`asc`) attribute as appropriate. Ascending order is the default order type.

For example, the following query sorts all records by the Name column and then returns the first three records in descending order .

```
mysql-py> db.country.select(["Code", "Name"]).order_by(["Name desc"]).limit(3)
+------+------------+
| Code | Name       |
+------+------------+
| ZWE  | Zimbabwe   |
| ZMB  | Zambia     |
| YUG  | Yugoslavia |
+------+------------+
3 rows in set (0.00 sec)
```

By default, the `limit()` method starts from the first record in the table. You can use the `offset()` method to change the starting record. For example, to ignore the first record and return the next three records matching the condition, pass to the `offset()` method a value of 1.

```
mysql-py> db.country.select(["Code", "Name"]).order_by(["Name desc"]).limit(3).offset(1)
+------+------------+
| Code | Name       |
+------+------------+
| ZMB  | Zambia     |
| YUG  | Yugoslavia |
| YEM  | Yemen      |
+------+------------+
3 rows in set (0.00 sec)
```

Related Information

- The MySQL Reference Manual provides detailed documentation on functions and operators.
- See TableSelectFunction for the full syntax definition.

20.4.4.3 Update Tables

You can use the `update()` method to modify one or more records in a table. The `update()` method works by filtering a query to include only the records to be updated and then applying the operations you specify to those records.

To replace a city name in the city table, pass to the `set()` method the new city name. Then, pass to the `where()` method the city name to locate and replace. The following example replaces the city Peking with Beijing.

```
mysql-py> db.city.update().set("Name", "Beijing").where("Name = 'Peking'")
Query OK, 1 item affected (0.04 sec)
```

Use the `select()` method to verify the change.

```
mysql-py> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).where("Name = 'Beijing'")
+------+-----------+-------------+----------+----------------------------+
| ID   | Name      | CountryCode | District | Info                       |
+------+-----------+-------------+----------+----------------------------+
| 1891 | Beijing   | CHN         | Peking   | {"Population": 7472000}    |
+------+-----------+-------------+----------+----------------------------+
1 row in set (0.00 sec)
```

Related Information

- See TableUpdateFunction for the full syntax definition.

20.4.4.4 Delete Tables

You can use the `delete()` method to remove some or all records from a table in a database. The X DevAPI provides additional methods to use with the `delete()` method to filter and order the records to be deleted.

Delete Records Using Conditions

The example that follows passes search conditions to the `delete()` method. All records matching the condition will be deleted from the city table. In this example, one record matches the condition.

```
mysql-py> db.city.delete().where("Name = 'Olympia'")
Query OK, 1 item affected (0.01 sec)
```

Delete the First Record

To delete the first record in the city table, use the `limit()` method with a value of 1.

```
mysql-py> db.city.delete().limit(1)
Query OK, 1 item affected (0.02 sec)
```

Delete All Records in a Table

You can delete all records in a table. To do so, use the `delete()` method without specifying a search condition.

Caution

Use care when you delete records without specifying a search condition. This action will delete all records from the table.

Drop a Table

The `drop_collection()` method is also used in MySQL Shell to drop a relational table from a database. For example, to drop the citytest table from the `world_x` database, issue:

```
mysql-py> session.drop_collection("world_x", "citytest")
Query OK (0.04 sec)
```

Related Information

- See TableDeleteFunction for the full syntax definition.
- See Section 20.4.2, “Download and Import world_x Database” for instructions to recreate the `world_x` database.

20.4.5 Documents in Tables

In MySQL, a table may contain traditional relational data, JSON values, or both. You can combine traditional data with JSON documents by storing the documents in columns having a native `JSON` data type.

Examples in this section use the city table in the `world_x` schema.

city Table Description

The city table has five columns (or fields).

```
+---------------+------------+-------+-------+---------+------------------+
| Field         | Type       | Null  | Key   | Default | Extra            |
+---------------+------------+-------+-------+---------+------------------+
| ID            | int(11)    | NO    | PRI   | null    | auto_increment   |
| Name          | char(35)   | NO    |       |         |                  |
| CountryCode   | char(3)    | NO    |       |         |                  |
| District      | char(20)   | NO    |       |         |                  |
| Info          | json       | YES   |       | null    |                  |
+---------------+------------+-------+-------+---------+------------------+
```

Insert a Record

To insert a document into the column of a table, pass to the `values()` method a well-formed JSON document in the correct order. In the following example, a document is passed as the final value to be inserted into the Info column.

```
mysql-py> db.city.insert().\
values(None, "San Francisco", "USA", "California", '{"Population":830000}')
Query OK, 1 item affected (0.01 sec)
```

Select a Record

You can issue a query with a search condition that evaluates document values in the expression.

```
mysql-py> db.city.select(["ID", "Name", "CountryCode", "District", "Info"]).\
  where("CountryCode = :country and Info->'$.Population' > 1000000").\
  bind('country', 'USA')
+------+----------------+-------------+----------------+-----------------------------+
| ID   | Name           | CountryCode | District       | Info                        |
+------+----------------+-------------+----------------+-----------------------------+
| 3793 | New York       | USA         | New York       | {"Population": 8008278}     |
| 3794 | Los Angeles    | USA         | California     | {"Population": 3694820}     |
| 3795 | Chicago        | USA         | Illinois       | {"Population": 2896016}     |
| 3796 | Houston        | USA         | Texas          | {"Population": 1953631}     |
| 3797 | Philadelphia   | USA         | Pennsylvania   | {"Population": 1517550}     |
| 3798 | Phoenix        | USA         | Arizona        | {"Population": 1321045}     |
| 3799 | San Diego      | USA         | California     | {"Population": 1223400}     |
| 3800 | Dallas         | USA         | Texas          | {"Population": 1188580}     |
| 3801 | San Antonio    | USA         | Texas          | {"Population": 1144646}     |
+------+----------------+-------------+----------------+-----------------------------+
9 rows in set (0.01 sec)
```

Related Information

- See Working with Relational Tables and Documents for more information.
- See Section 11.6, “The JSON Data Type” for a detailed description of the data type.

20.5 X Plugin

This section explains how to use, configure and monitor X Plugin.

20.5.1 Checking X Plugin Installation

X Plugin is enabled by default in MySQL 8, therefore installing or upgrading to MySQL 8 makes the plugin available. You can verify X Plugin is installed on an instance of MySQL server by using the `SHOW plugins` statement to view the plugins list.

To use MySQL Shell to verify X Plugin is installed, issue:

```
shell> mysqlsh -u user --sqlc -P 3306 -e "SHOW plugins"
```

To use MySQL Client to verify X Plugin is installed, issue:

```
shell> mysql -u user -p -e "SHOW plugins"
```

An example result if X Plugin is installed is highlighted here:

```
+----------------------------+----------+--------------------+---------+---------+
| Name                       | Status   | Type               | Library | License |
+----------------------------+----------+--------------------+---------+---------+

...

| mysqlx                     | ACTIVE   | DAEMON             | NULL    | GPL     |

...

+----------------------------+----------+--------------------+---------+---------+
```

20.5.2 Disabling X Plugin

The X Plugin can be disabled at startup by either setting `mysqlx=0` in your MySQL configuration file, or by passing in either `--mysqlx=0` or `--skip-mysqlx` when starting the MySQL server.

Alternatively, use the `-DWITH_MYSQLX=OFF` CMake option to compile MySQL Server without X Plugin.

20.5.3 Using Secure Connections with X Plugin

This section explains how to configure X Plugin to use secure connections. For more background information, see Section 6.3, “Using Encrypted Connections”.

X Plugin has its own SSL settings which can differ from those used with MySQL Server. This means that X Plugin can be configured with a different SSL key, certificate, and certificate authorities file than MySQL Server. Similarly, X Plugin has its own SSL status variables calculated independently from the MySQL Server SSL related variables. By default the X Plugin SSL configuration is taken from the `mysqlx_ssl_*` variables, described at Section 20.5.5.2, “X Plugin Options and System Variables”. If no configuration is

provided using the `mysqlx_ssl_*` variables, X Plugin falls back to using the MySQL Server SSL system variables. This means you can choose to either have separate SSL configurations for MySQL Protocol and X Protocol connections by configuring each separately, or share the SSL configuration between MySQL Protocol and X Protocol connections by only configuring the `ssl-*` variables.

On a server with X Plugin installed, to configure MySQL Protocol and X Protocol connections with separate SSL configurations use both the `ssl-*` and `mysqlx-ssl-*` variables in `my.cnf`:

```
[mysqld]
ssl-ca=ca1.pem
ssl-cert=server-cert1.pem
ssl-key=server-key1.pem

mysqlx-ssl-ca=ca2.pem
mysqlx-ssl-cert=server-cert2.pem
mysqlx-ssl-key=server-key2.pem
```

The available `mysqlx_ssl_*` variables mirror the SSL variables in MySQL Server, so the files and techniques described for configuring MySQL Server to use SSL at Section 6.3.1, “Configuring MySQL to Use Encrypted Connections” are relevant to configuring X Plugin to use secure connections.

You can configure the TLS versions used by X Protocol SSL connections using the `tls_version` system variable. The TLS version used by MySQL Protocol and X Protocol connections is therefore the same TLS version.

Encryption per connection is optional, but a specific user can be forced to use encryption for X Protocol and MySQL Protocol connections. You configure such a user by issuing a `GRANT` statement with the `REQUIRE` option. For more details see Section 13.7.1.6, “GRANT Syntax”. Alternatively all X Protocol and MySQL Protocol connections can be forced to use encryption by setting `require_secure_transport`.

20.5.4 Using X Plugin with the Caching SHA-2 Authentication Plugin

X Plugin supports MySQL accounts created with the Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication” plugin. You can use X Plugin to authenticate against such accounts using non-SSL connections with `SHA256_MEMORY` authentication and SSL connections with `PLAIN` authentication.

Although the caching SHA-2 authentication plugin holds an authentication cache, to use such accounts with X Plugin the X Plugin authentication cache plugin named `mysqlx_cache_cleaner` is used. Like X Plugin, it is enabled by default.

Before you can use non-SSL connections to authenticate an account which uses the `caching_sha2_password` plugin, the account must have authenticated at least once over an SSL connection to store the password in the X Plugin authentication cache. This means that the first use of an account must be done using an SSL connection with the X Plugin authentication cache enabled. Once this initial authentication over SSL has succeeded non-SSL connections can be used.

20.5.5 X Plugin Options and Variables

This section describes the command options and system variables that configure X Plugin. If values specified at startup time are incorrect, X Plugin could fail to initialize properly and the server does not load it. In this case, the server could also produce error messages for other X Plugin settings because it cannot recognize them.

20.5.5.1 X Plugin Option and Variable Reference

This table provides an overview of the command options, and system and status variables provided by X Plugin.

Table 20.1 X Plugin Option and Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
mysqlx	Yes	Yes				
Mysqlx_aborted_clients				Yes	Global	No
Mysqlx_address				Yes	Global	No
mysqlx_bind_address	Yes	Yes	Yes		Global	No
Mysqlx_bytes_received				Yes	Both	No
Mysqlx_bytes_sent				Yes	Both	No
mysqlx_connect_timeout	Yes	Yes	Yes		Global	Yes
Mysqlx_connection_accept_errors				Yes	Both	No
Mysqlx_connection_errors				Yes	Both	No
Mysqlx_connections_accepted				Yes	Global	No
Mysqlx_connections_closed				Yes	Global	No
Mysqlx_connections_rejected				Yes	Global	No
Mysqlx_crud_create_view				Yes	Both	No
Mysqlx_crud_delete				Yes	Both	No
Mysqlx_crud_drop_view				Yes	Both	No
Mysqlx_crud_find				Yes	Both	No
Mysqlx_crud_insert				Yes	Both	No
Mysqlx_crud_modify_view				Yes	Both	No
Mysqlx_crud_update				Yes	Both	No
mysqlx_document_id_unique_prefix	Yes	Yes	Yes		Global	Yes
mysqlx_enable_hello_notice	Yes	Yes	Yes		Global	Yes
Mysqlx_errors_sent				Yes	Both	No
Mysqlx_errors_unknown_message_type				Yes	Both	No
Mysqlx_expect_close				Yes	Both	No
Mysqlx_expect_open				Yes	Both	No
mysqlx_idle_worker_thread_timeout	Yes	Yes	Yes		Global	Yes
Mysqlx_init_error				Yes	Both	No
mysqlx_interactive_timeout	Yes	Yes	Yes		Global	Yes
mysqlx_max_allowed_packet	Yes	Yes	Yes		Global	Yes
mysqlx_max_connections	Yes	Yes	Yes		Global	Yes
mysqlx_min_worker_threads	Yes	Yes	Yes		Global	Yes
Mysqlx_notice_global_sent				Yes	Both	No
Mysqlx_notice_other_sent				Yes	Both	No
Mysqlx_notice_warning_sent				Yes	Both	No
Mysqlx_notified_by_group_replication				Yes	Both	No
Mysqlx_port				Yes	Global	No
mysqlx_port	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
mysqlx_port_open_timeout	Yes	Yes	Yes		Global	No
mysqlx_read_timeout	Yes	Yes	Yes		Session	Ye
Mysqlx_rows_sent				Yes	Both	No
Mysqlx_sessions				Yes	Global	No
Mysqlx_sessions_accepted				Yes	Global	No
Mysqlx_sessions_closed				Yes	Global	No
Mysqlx_sessions_fatal_error				Yes	Global	No
Mysqlx_sessions_killed				Yes	Global	No
Mysqlx_sessions_rejected				Yes	Global	No
Mysqlx_socket				Yes	Global	No
mysqlx_socket	Yes	Yes	Yes		Global	No
Mysqlx_ssl_accept_renegotiates				Yes	Global	No
Mysqlx_ssl_accepts				Yes	Global	No
Mysqlx_ssl_active				Yes	Both	No
mysqlx_ssl_ca	Yes	Yes	Yes		Global	No
mysqlx_ssl_capath	Yes	Yes	Yes		Global	No
mysqlx_ssl_cert	Yes	Yes	Yes		Global	No
Mysqlx_ssl_cipher				Yes	Both	No
mysqlx_ssl_cipher	Yes	Yes	Yes		Global	No
Mysqlx_ssl_cipher_list				Yes	Both	No
mysqlx_ssl_crl	Yes	Yes	Yes		Global	No
mysqlx_ssl_crlpath	Yes	Yes	Yes		Global	No
Mysqlx_ssl_ctx_verify_depth				Yes	Both	No
Mysqlx_ssl_ctx_verify_mode				Yes	Both	No
Mysqlx_ssl_finished_accepts				Yes	Global	No
mysqlx_ssl_key	Yes	Yes	Yes		Global	No
Mysqlx_ssl_server_not_after				Yes	Global	No
Mysqlx_ssl_server_not_before				Yes	Global	No
Mysqlx_ssl_verify_depth				Yes	Global	No
Mysqlx_ssl_verify_mode				Yes	Global	No
Mysqlx_ssl_version				Yes	Both	No
Mysqlx_stmt_create_collection				Yes	Both	No
Mysqlx_stmt_create_collection_index				Yes	Both	No
Mysqlx_stmt_disable_notices				Yes	Both	No
Mysqlx_stmt_drop_collection				Yes	Both	No
Mysqlx_stmt_drop_collection_index				Yes	Both	No
Mysqlx_stmt_enable_notices				Yes	Both	No
Mysqlx_stmt_ensure_collection				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Mysqlx_stmt_execute_mysqlx				Yes	Both	No
Mysqlx_stmt_execute_sql				Yes	Both	No
Mysqlx_stmt_execute_xplugin				Yes	Both	No
Mysqlx_stmt_kill_client				Yes	Both	No
Mysqlx_stmt_list_clients				Yes	Both	No
Mysqlx_stmt_list_notices				Yes	Both	No
Mysqlx_stmt_list_objects				Yes	Both	No
Mysqlx_stmt_ping				Yes	Both	No
mysqlx_wait_timeout	Yes	Yes	Yes		Session	Yes
Mysqlx_worker_threads				Yes	Global	No
Mysqlx_worker_threads_active				Yes	Global	No
mysqlx_write_timeout	Yes	Yes	Yes		Session	Yes

20.5.5.2 X Plugin Options and System Variables

To control activation of X Plugin, use this option:

- `--mysqlx[=value]`

Property	Value
Command-Line Format	`--mysqlx[=value]`
Introduced	8.0.11
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`FORCE`
	`FORCE_PLUS_PERMANENT`

This option controls how the server loads X Plugin at startup. In MySQL 8.0, X Plugin is enabled by default, but this option may be used to control its activation state.

The option value should be one of those available for plugin-loading options, as described in Section 5.6.1, “Installing and Uninstalling Plugins”.

If X Plugin is enabled, it exposes several system variables that permit control over its operation:

- `mysqlx_bind_address`

Property	Value
Command-Line Format	`--mysqlx-bind-address=addr`
System Variable	`mysqlx_bind_address`
Scope	Global

Property	Value
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	*

The network address on which X Plugin listens for TCP/IP connections. This variable is not dynamic and can be configured only at startup. This is the X Plugin equivalent of the `bind_address` system variable; see that variable description for more information.

If `mysqlx_bind_address` is specified, its value must be a single non-wildcard IP address or host name, or one of the wildcard address formats that permit listening on multiple network interfaces (`*`, `0.0.0.0`, or `::`).

An IP address can be specified as an IPv4 or IPv6 address. If the value is a host name, X Plugin resolves the name to an IP address and binds to that address. If a host name resolves to multiple IP addresses, X Plugin uses the first IPv4 address if there are any, or the first IPv6 address otherwise.

X Plugin treats different types of addresses as follows:

- If the address is `*`, X Plugin accepts TCP/IP connections on all server host IPv4 interfaces, and, if the server host supports IPv6, on all IPv6 interfaces. Use this address to permit both IPv4 and IPv6 connections for X Plugin. This value is the default.
- If the address is `0.0.0.0`, X Plugin accepts TCP/IP connections on all server host IPv4 interfaces.
- If the address is `::`, X Plugin accepts TCP/IP connections on all server host IPv4 and IPv6 interfaces.
- If the address is an IPv4-mapped address, X Plugin accepts TCP/IP connections for that address, in either IPv4 or IPv6 format. For example, if X Plugin is bound to `::ffff:127.0.0.1`, a client such as MySQL Shell can connect using `--host=127.0.0.1` or `--host=::ffff:127.0.0.1`.
- If the address is a “regular” IPv4 or IPv6 address (such as `127.0.0.1` or `::1`), X Plugin accepts TCP/IP connections only for that IPv4 or IPv6 address.

If binding to the address fails, X Plugin produces an error and the server does not load it.

- `mysqlx_connect_timeout`

Property	Value
Command-Line Format	`--mysqlx-connect-timeout=#`
System Variable	`mysqlx_connect_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`30`
Minimum Value	`1`
Maximum Value	`1000000000`

The number of seconds X Plugin waits for the first packet to be received from newly connected clients. This is the X Plugin equivalent of `connect_timeout`; see that variable for more information.

- `mysqlx_document_id_unique_prefix`

Property	Value
Command-Line Format	`--mysqlx-document-id-unique-prefix=#`
System Variable	`mysqlx_document_id_unique_prefix`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`65535`

Sets the first 4 bytes of document IDs generated by the server when documents are added to a collection. By setting this variable to a unique value per instance, you can ensure document IDs are unique across instances. See Understanding Document IDs.

- `mysqlx_enable_hello_notice`

Property	Value
Command-Line Format	`--mysqlx-enable-hello-notice[={OFF|ON}]`
System Variable	`mysqlx_enable_hello_notice`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Controls messages sent to classic MySQL protocol clients that try to connect over X Protocol. When enabled, clients which do not support X Protocol that attempt to connect to the server X Protocol port receive an error explaining they are using the wrong protocol.

- `mysqlx_idle_worker_thread_timeout`

Property	Value
Command-Line Format	`--mysqlx-idle-worker-thread-timeout=#`
System Variable	`mysqlx_idle_worker_thread_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`60`

Property	Value
Minimum Value	`0`
Maximum Value	`3600`

The number of seconds after which idle worker threads are terminated.

- `mysqlx_interactive_timeout`

Property	Value
Command-Line Format	`--mysqlx-interactive-timeout=#`
Introduced	8.0.4
System Variable	`mysqlx_interactive_timeout`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`28800`
Minimum Value	`1`
Maximum Value	`2147483`

The default value of the `mysqlx_wait_timeout` session variable for interactive clients. (The number of seconds to wait for interactive clients to timeout.)

- `mysqlx_max_allowed_packet`

Property	Value
Command-Line Format	`--mysqlx-max-allowed-packet=#`
System Variable	`mysqlx_max_allowed_packet`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`67108864`
Minimum Value	`512`
Maximum Value	`1073741824`

The maximum size of network packets that can be received by X Plugin. This is the X Plugin equivalent of `max_allowed_packet`; see that variable for more information.

- `mysqlx_max_connections`

Property	Value
Command-Line Format	`--mysqlx-max-connections=#`
System Variable	`mysqlx_max_connections`
Scope	Global

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	100
Minimum Value	1
Maximum Value	65535

The maximum number of concurrent client connections X Plugin can accept. This is the X Plugin equivalent of `max_connections`; see that variable for more information.

For modifications to this variable, if the new value is smaller than the current number of connections, the new limit is taken into account only for new connections.

- `mysqlx_min_worker_threads`

Property	Value
Command-Line Format	`--mysqlx-min-worker-threads=#`
System Variable	`mysqlx_min_worker_threads`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	2
Minimum Value	1
Maximum Value	100

The minimum number of worker threads used by X Plugin for handling client requests.

- `mysqlx_port`

Property	Value
Command-Line Format	`--mysqlx-port=port_num`
System Variable	`mysqlx_port`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	33060
Minimum Value	1
Maximum Value	65535

The network port on which X Plugin listens for TCP/IP connections. This is the X Plugin equivalent of `port`; see that variable for more information.

- `mysqlx_port_open_timeout`

Property	Value
Command-Line Format	`--mysqlx-port-open-timeout=#`
System Variable	`mysqlx_port_open_timeout`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`120`

The number of seconds X Plugin waits for a TCP/IP port to become free.

- `mysqlx_read_timeout`

Property	Value
Command-Line Format	`--mysqlx-read-timeout=#`
Introduced	8.0.4
System Variable	`mysqlx_read_timeout`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`28800`
Minimum Value	`30`
Maximum Value	`2147483`

The number of seconds that X Plugin waits for blocking read operations to complete. After this time, if the read operation is not successful, the connection is aborted.

- `mysqlx_socket`

Property	Value
Command-Line Format	`--mysqlx-socket=file_name`
System Variable	`mysqlx_socket`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	`/tmp/mysqlx.sock`

The path to a Unix socket file which X Plugin uses for connections. This setting is only used by MySQL Server when running on Unix operating systems. Clients can use this socket to connect to MySQL Server using X Plugin.

The default `mysqlx_socket` path and file name is based on the default path and file name for the main socket file for MySQL Server, with the addition of an `x` appended to the file name. The default path and file name for the main socket file is `/tmp/mysql.sock`, therefore the default path and file name for the X Plugin socket file is `/tmp/mysqlx.sock`.

If you specify an alternative path and file name for the main socket file at server startup using the `socket` system variable, this does not affect the default for the X Plugin socket file. In this situation, if you want to store both sockets at a single path, you must set the `mysqlx_socket` system variable as well. For example in a configuration file:

```
socket=/home/sockets/mysqld/mysql.sock
mysqlx_socket=/home/sockets/xplugin/xplugin.sock
```

If you change the default path and file name for the main socket file at compile time using the `MYSQL_UNIX_ADDR` compile option, this does affect the default for the X Plugin socket file, which is formed by appending an `x` to the `MYSQL_UNIX_ADDR` file name. If you want to set a different default for the X Plugin socket file at compile time, use the `MYSQLX_UNIX_ADDR` compile option.

The `MYSQLX_UNIX_PORT` environment variable can also be used to set a default for the X Plugin socket file at server startup (see Section 4.9, “MySQL Environment Variables”). If you set this environment variable, it overrides the compiled `MYSQLX_UNIX_ADDR` value, but is overridden by the `mysqlx_socket` value.

- `mysqlx_ssl_ca`

Property	Value
Command-Line Format	`--mysqlx-ssl-ca=file_name`
System Variable	`mysqlx_ssl_ca`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

This is the X Plugin equivalent of `ssl_ca`; see that variable for more information.

- `mysqlx_ssl_capath`

Property	Value
Command-Line Format	`--mysqlx-ssl-capath=dir_name`
System Variable	`mysqlx_ssl_capath`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

This is the X Plugin equivalent of `ssl_capath`; see that variable for more information.

- `mysqlx_ssl_cert`

Property	Value
Command-Line Format	`--mysqlx-ssl-cert=name`
System Variable	`mysqlx_ssl_cert`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

This is the X Plugin equivalent of `ssl_cert`; see that variable for more information.

- `mysqlx_ssl_cipher`

Property	Value
Command-Line Format	`--mysqlx-ssl-cipher=name`
System Variable	`mysqlx_ssl_cipher`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String

The SSL cipher to use for X Protocol connections. This is the X Plugin equivalent of `ssl_cipher`; see that variable for more information.

- `mysqlx_ssl_crl`

Property	Value
Command-Line Format	`--mysqlx-ssl-crl=file_name`
System Variable	`mysqlx_ssl_crl`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

This is the X Plugin equivalent of `ssl_crl`; see that variable for more information.

- `mysqlx_ssl_crlpath`

Property	Value
Command-Line Format	`--mysqlx-ssl-crlpath=dir_name`
System Variable	`mysqlx_ssl_crlpath`
Scope	Global

Property	Value
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

This is the X Plugin equivalent of `ssl_crlpath`; see that variable for more information.

- `mysqlx_ssl_key`

Property	Value
Command-Line Format	`--mysqlx-ssl-key=file_name`
System Variable	`mysqlx_ssl_key`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name

This is the X Plugin equivalent of `ssl_key`; see that variable for more information.

- `mysqlx_wait_timeout`

Property	Value
Command-Line Format	`--mysqlx-wait-timeout=#`
Introduced	8.0.4
System Variable	`mysqlx_wait_timeout`
Scope	Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`28800`
Minimum Value	`1`
Maximum Value	`2147483`

The number of seconds that X Plugin waits for activity on a connection. After this time, if the read operation is not successful, the connection is aborted. If the client is noninteractive, the initial value of the session variable is copied from the global `mysqlx_wait_timeout` variable. For interactive clients, the initial value is copied from the session `mysqlx_interactive_timeout`.

- `mysqlx_write_timeout`

Property	Value
Command-Line Format	`--mysqlx-write-timeout=#`
Introduced	8.0.4
System Variable	`mysqlx_write_timeout`
Scope	Session

Property	Value
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	60
Minimum Value	1
Maximum Value	2147483

The number of seconds that X Plugin waits for blocking write operations to complete. After this time, if the write operation is not successful, the connection is aborted.

20.5.6 Monitoring X Plugin

This section describes how to monitor X Plugin. There are two available methods of monitoring, using Performance Schema tables or status variables.

20.5.6.1 Status Variables for X Plugin

The status variables have the following meanings.

- `Mysqlx_aborted_clients`

  The number of clients that were disconnected because of an input or output error.

- `Mysqlx_address`

  The network address which X Plugin is bound to. If the bind has failed, or if the `skip_networking` option has been used, the value shows `UNDEFINED`.

- `Mysqlx_bytes_received`

  The number of bytes received through the network.

- `Mysqlx_bytes_sent`

  The number of bytes sent through the network.

- `Mysqlx_connection_accept_errors`

  The number of connections which have caused accept errors.

- `Mysqlx_connection_errors`

  The number of connections which have caused errors.

- `Mysqlx_connections_accepted`

  The number of connections which have been accepted.

- `Mysqlx_connections_closed`

  The number of connections which have been closed.

- `Mysqlx_connections_rejected`

The number of connections which have been rejected.

- `Mysqlx_crud_create_view`

  The number of create view requests received.

- `Mysqlx_crud_delete`

  The number of delete requests received.

- `Mysqlx_crud_drop_view`

  The number of drop view requests received.

- `Mysqlx_crud_find`

  The number of find requests received.

- `Mysqlx_crud_insert`

  The number of insert requests received.

- `Mysqlx_crud_modify_view`

  The number of modify view requests received.

- `Mysqlx_crud_update`

  The number of update requests received.

- `Mysqlx_cursor_close`

  The number of cursor-close messages received

- `Mysqlx_cursor_fetch`

  The number of cursor-fetch messages received

- `Mysqlx_cursor_open`

  The number of cursor-open messages received

- `Mysqlx_errors_sent`

  The number of errors sent to clients.

- `Mysqlx_expect_close`

  The number of expectation blocks closed.

- `Mysqlx_expect_open`

  The number of expectation blocks opened.

- `Mysqlx_init_error`

  The number of errors during initialisation.

- `Mysqlx_notice_global_sent`

The number of global notifications sent to clients.

- `Mysqlx_notice_other_sent`

  The number of other types of notices sent back to clients.

- `Mysqlx_notice_warning_sent`

  The number of warning notices sent back to clients.

- `Mysqlx_notified_by_group_replication`

  Number of Group Replication notifications sent to clients

- `Mysqlx_port`

  The TCP port which X Plugin is listening to. If a network bind has failed, or if the `--skip-networking` option has been used, the value shows `UNDEFINED`.

- `Mysqlx_prep_deallocate`

  The number of prepared-statement-deallocate messages received

- `Mysqlx_prep_execute`

  The number of prepared-statement-execute messages received

- `Mysqlx_prep_prepare`

  The number of prepared-statement messages received

- `Mysqlx_rows_sent`

  The number of rows sent back to clients.

- `Mysqlx_sessions`

  The number of sessions that have been opened.

- `Mysqlx_sessions_accepted`

  The number of session attempts which have been accepted.

- `Mysqlx_sessions_closed`

  The number of sessions that have been closed.

- `Mysqlx_sessions_fatal_error`

  The number of sessions that have closed with a fatal error.

- `Mysqlx_sessions_killed`

  The number of sessions which have been killed.

- `Mysqlx_sessions_rejected`

  The number of session attempts which have been rejected.

- `Mysqlx_socket`

  The Unix socket which X Plugin is listening to.

- `Mysqlx_ssl_accept_renegotiates`

  The number of negotiations needed to establish the connection.

- `Mysqlx_ssl_accepts`

  The number of accepted SSL connections.

- `Mysqlx_ssl_active`

  If SSL is active.

- `Mysqlx_ssl_cipher`

  The current SSL cipher (empty for non-SSL connections).

- `Mysqlx_ssl_cipher_list`

  A list of possible SSL ciphers (empty for non-SSL connections).

- `Mysqlx_ssl_ctx_verify_depth`

  The certificate verification depth limit currently set in ctx.

- `Mysqlx_ssl_ctx_verify_mode`

  The certificate verification mode currently set in ctx.

- `Mysqlx_ssl_finished_accepts`

  The number of successful SSL connections to the server.

- `Mysqlx_ssl_server_not_after`

  The last date for which the SSL certificate is valid.

- `Mysqlx_ssl_server_not_before`

  The first date for which the SSL certificate is valid.

- `Mysqlx_ssl_verify_depth`

  The certificate verification depth for SSL connections.

- `Mysqlx_ssl_verify_mode`

  The certificate verification mode for SSL connections.

- `Mysqlx_ssl_version`

  The name of the protocol used for SSL connections.

- `Mysqlx_stmt_create_collection`

  The number of create collection statements received.

- `Mysqlx_stmt_create_collection_index`

  The number of create collection index statements received.

- `Mysqlx_stmt_disable_notices`

  The number of disable notice statements received.

- `Mysqlx_stmt_drop_collection`

  The number of drop collection statements received.

- `Mysqlx_stmt_drop_collection_index`

  The number of drop collection index statements received.

- `Mysqlx_stmt_enable_notices`

  The number of enable notice statements received.

- `Mysqlx_stmt_ensure_collection`

  The number of ensure collection statements received.

- `Mysqlx_stmt_execute_mysqlx`

  The number of StmtExecute messages received with namespace set to `mysqlx`.

- `Mysqlx_stmt_execute_sql`

  The number of StmtExecute requests received for the SQL namespace.

- `Mysqlx_stmt_execute_xplugin`

  The number of StmtExecute requests received for the X Plugin namespace.

- `Mysqlx_stmt_kill_client`

  The number of kill client statements received.

- `Mysqlx_stmt_list_clients`

  The number of list client statements received.

- `Mysqlx_stmt_list_notices`

  The number of list notice statements received.

- `Mysqlx_stmt_list_objects`

  The number of list object statements received.

- `Mysqlx_stmt_ping`

  The number of ping statements received.

- `Mysqlx_worker_threads`

  The number of worker threads available.

- `Mysqlx_worker_threads_active`

  The number of worker threads currently used.

Chapter 21 InnoDB Cluster

Table of Contents

21.1 Introducing InnoDB Cluster 3593
21.2 Creating an InnoDB Cluster 3596
21.2.1 Deployment Scenarios 3596
21.2.2 InnoDB Cluster Requirements 3596
21.2.3 Methods of Installing 3597
21.2.4 Production Deployment of InnoDB Cluster 3597
21.2.5 Using MySQL Clone with InnoDB cluster 3608
21.2.6 Sandbox Deployment of InnoDB Cluster 3613
21.2.7 Adopting a Group Replication Deployment 3616
21.3 Using MySQL Router with InnoDB Cluster 3616
21.4 Working with InnoDB Cluster 3620
21.5 Known Limitations 3647

This chapter covers MySQL InnoDB cluster, which combines MySQL technologies to enable you to create highly available clusters of MySQL server instances.

21.1 Introducing InnoDB Cluster

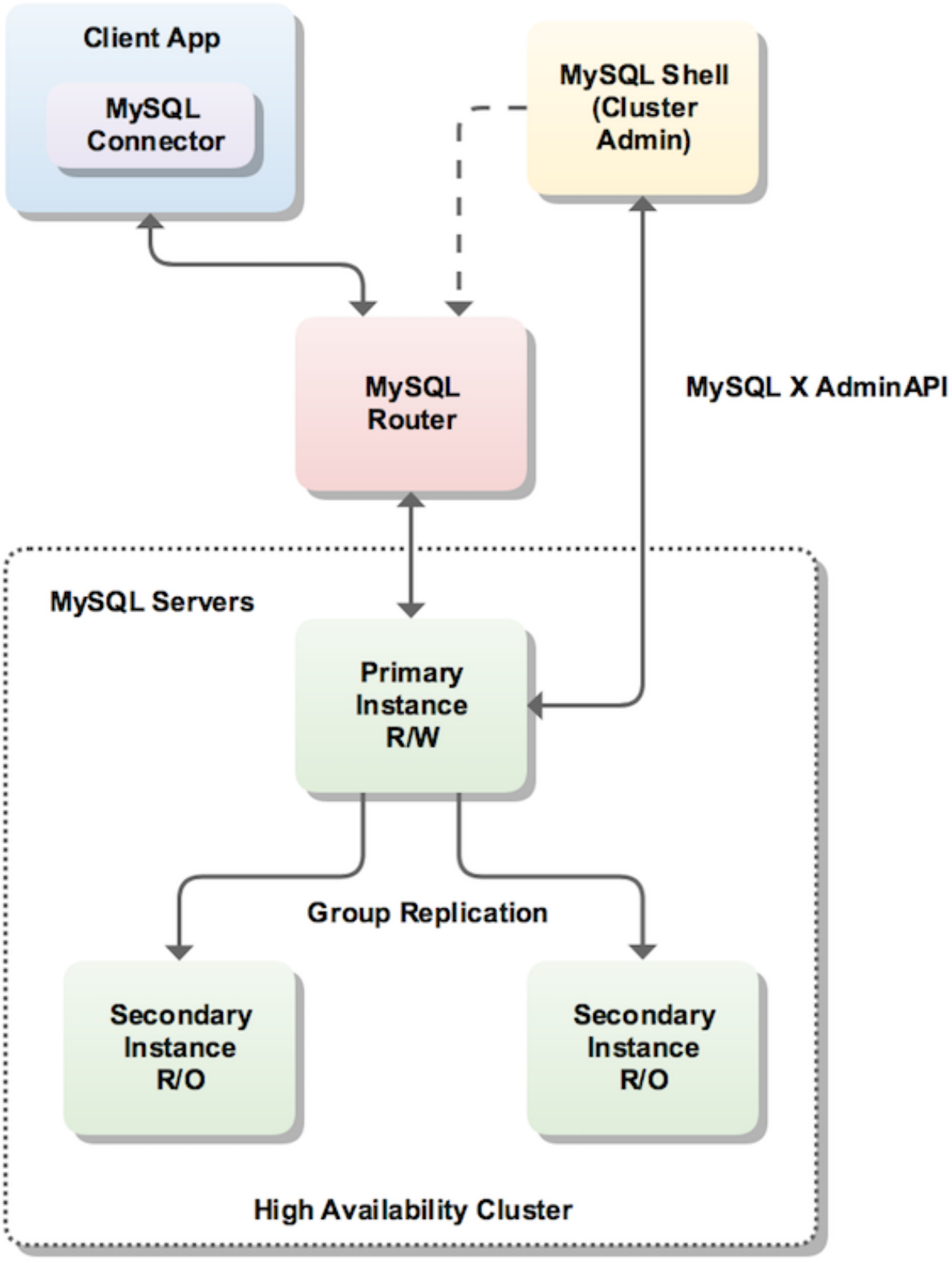
MySQL InnoDB cluster provides a complete high availability solution for MySQL. MySQL Shell includes AdminAPI which enables you to easily configure and administer a group of at least three MySQL server instances to function as an InnoDB cluster. Each MySQL server instance runs MySQL Group Replication, which provides the mechanism to replicate data within InnoDB clusters, with built-in failover. AdminAPI removes the need to work directly with Group Replication in InnoDB clusters, but for more information see Chapter 18, *Group Replication* which explains the details. MySQL Router can automatically configure itself based on the cluster you deploy, connecting client applications transparently to the server instances. In the event of an unexpected failure of a server instance the cluster reconfigures automatically. In the default single-primary mode, an InnoDB cluster has a single read-write server instance - the primary. Multiple secondary server instances are replicas of the primary. If the primary fails, a secondary is automatically promoted to the role of primary. MySQL Router detects this and forwards client applications to the new primary. Advanced users can also configure a cluster to have multiple-primaries.

Important

InnoDB cluster does not provide support for MySQL NDB Cluster. NDB Cluster depends on the `NDB` storage engine as well as a number of programs specific to NDB Cluster which are not furnished with MySQL Server 8.0; `NDB` is available only as part of the MySQL NDB Cluster distribution. In addition, the MySQL server binary (`mysqld`) that is supplied with MySQL Server 8.0 cannot be used with NDB Cluster. For more information about MySQL NDB Cluster, see Chapter 22, *MySQL NDB Cluster 8.0*. Section 22.1.6, “MySQL Server Using InnoDB Compared with NDB Cluster”, provides information about the differences between the `InnoDB` and `NDB` storage engines.

The following diagram shows an overview of how these technologies work together:

Figure 21.1 InnoDB cluster overview



Using AdminAPI

MySQL Shell includes the AdminAPI, which is accessed through the `dba` global variable and its associated methods. The `dba` variable's methods enable you to deploy, configure, and administer InnoDB clusters. For example, use the `dba.createCluster()` method to create an InnoDB cluster.

Important

MySQL Shell enables you to connect to servers over a socket connection, but AdminAPI requires TCP connections to a server instance. Socket based connections are not supported in AdminAPI.

MySQL Shell provides online help for the AdminAPI. To list all available `dba` commands, use the `dba.help()` method. For online help on a specific method, use the general format `object.help('methodname')`. For example:

```
mysql-js> dba.help('getCluster')

Retrieves a cluster from the Metadata Store.

SYNTAX

  dba.getCluster([name][, options])

WHERE

  name: Parameter to specify the name of the cluster to be returned.
  options: Dictionary with additional options.

RETURNS

  The cluster object identified by the given name or the default cluster.

DESCRIPTION

If name is not specified or is null, the default cluster will be returned.

If name is specified, and no cluster with the indicated name is found, an error
will be raised.

The options dictionary accepts the connectToPrimary option,which defaults to
true and indicates the shell to automatically connect to the primary member of
the cluster.

EXCEPTIONS

  MetadataError in the following scenarios:

   - If the Metadata is inaccessible.
   - If the Metadata update operation failed.

  ArgumentError in the following scenarios:

   - If the Cluster name is empty.
   - If the Cluster name is invalid.
   - If the Cluster does not exist.

  RuntimeError in the following scenarios:

   - If the current connection cannot be used for Group Replication.
```

MySQL Shell can optionally log the SQL statements used by AdminAPI operations (with the exception of sandbox operations), and can also display them in the terminal as they are executed. To configure MySQL Shell to do this, see Logging AdminAPI Operations.

21.2 Creating an InnoDB Cluster

This section explains the different ways you can create an InnoDB cluster, the requirements for server instances and the software you need to install to deploy a cluster.

21.2.1 Deployment Scenarios

InnoDB cluster supports the following deployment scenarios:

- *Production deployment:* if you want to use InnoDB cluster in a full production environment you need to configure the required number of machines and then deploy your server instances to the machines. A production deployment enables you to exploit the high availability features of InnoDB cluster to their full potential. See Section 21.2.4, “Production Deployment of InnoDB Cluster” for instructions.
- *Sandbox deployment:* if you want to test out InnoDB cluster before committing to a full production deployment, the provided sandbox feature enables you to quickly set up a cluster on your local machine. Sandbox server instances are created with the required configuration and you can experiment with InnoDB cluster to become familiar with the technologies employed. See Section 21.2.6, “Sandbox Deployment of InnoDB Cluster” for instructions.

Important

A sandbox deployment is not suitable for use in a full production environment.

21.2.2 InnoDB Cluster Requirements

Before installing a production deployment of InnoDB cluster, ensure that the server instances you intend to use meet the following requirements.

- InnoDB cluster uses Group Replication and therefore your server instances must meet the same requirements. See Section 18.9.1, “Group Replication Requirements”. AdminAPI provides the `dba.checkInstanceConfiguration()` method to verify that an instance meets the Group Replication requirements, and the `dba.configureInstance()` method to configure an instance to meet the requirements.

Note

When using a sandbox deployment the instances are configured to meet these requirements automatically.

- Group Replication members can contain tables using a storage engine other than `InnoDB`, for example `MyISAM`. Such tables cannot be written to by Group Replication, and therefore when using InnoDB cluster. To be able to write to such tables with InnoDB cluster, convert all such tables to `InnoDB` before using the instance in a InnoDB cluster.
- The Performance Schema must be enabled on any instance which you want to use with InnoDB cluster.
- The provisioning scripts that MySQL Shell uses to configure servers for use in InnoDB cluster require access to Python version 2.7. For a sandbox deployment Python is required on the single machine used for the deployment, production deployments require Python on each server instance which should run MySQL Shell locally, see Persisting Settings.

On Windows MySQL Shell includes Python and no user configuration is required. On Unix Python must be found as part of the shell environment. To check that your system has Python configured correctly issue:

```
$ /usr/bin/env python
```

If a Python interpreter starts, no further action is required. If the previous command fails, create a soft link between `/usr/bin/python` and your chosen Python binary.

- From version 8.0.17, instances must use a unique `server_id` within a InnoDB cluster. When you use the *`Cluster`*`.addInstance(`*`instance`*`)` operation, if the `server_id` of *`instance`* is already used by an instance in the cluster then the operation fails with an error.

21.2.3 Methods of Installing

The method you use to install InnoDB cluster depends on the type of deployment you intend to use. For a sandbox deployment install the components of InnoDB cluster to a single machine. A sandbox deployment is local to a single machine, therefore the install needs to only be done once on the local machine. For a production deployment install the components to each machine that you intend to add to your cluster. A production deployment uses multiple remote host machines running MySQL server instances, so you need to connect to each machine using a tool such as SSH or Windows remote desktop to carry out tasks such as installing components. The following methods of installing InnoDB cluster are available:

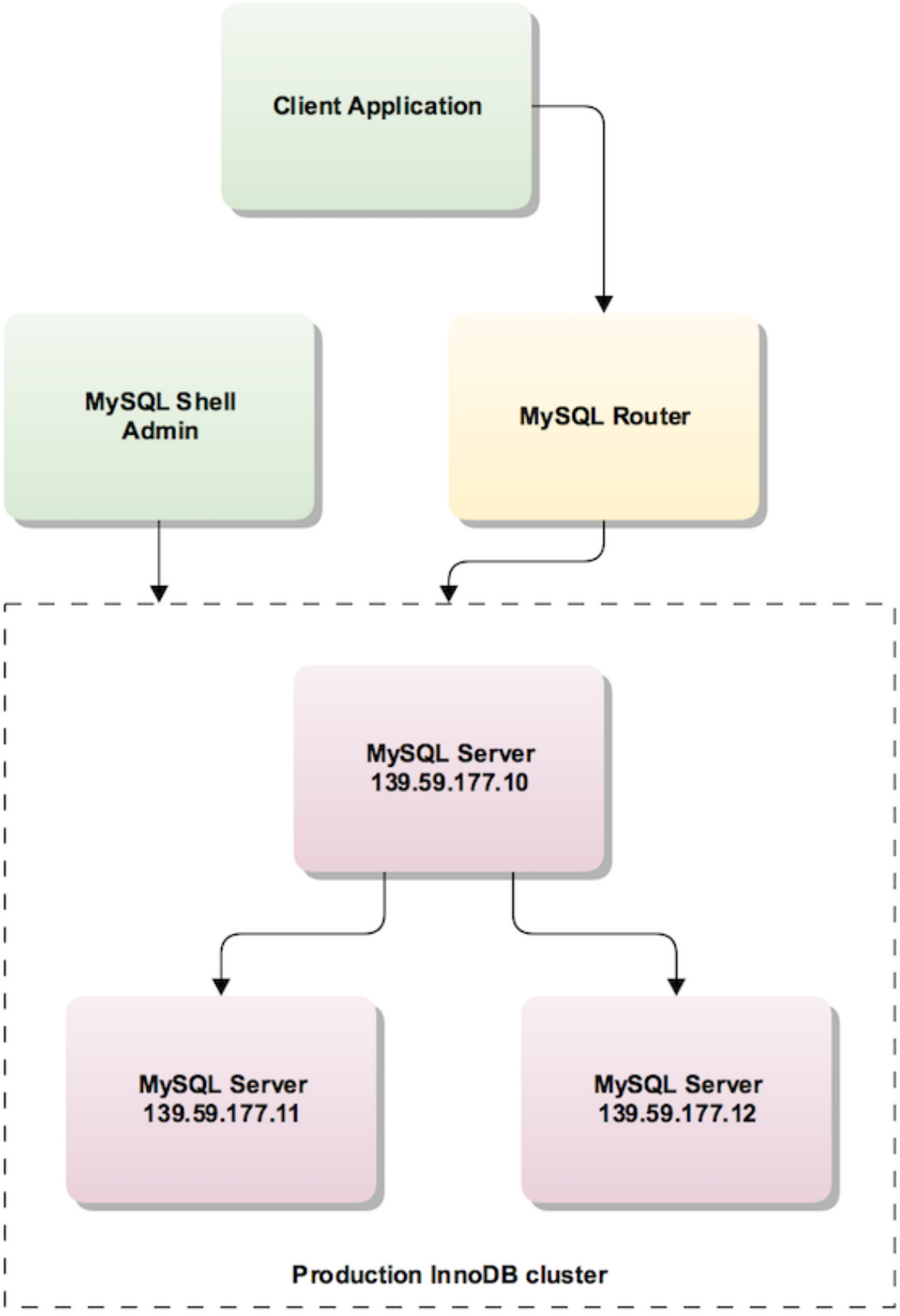
- Downloading and installing the components using the following documentation:
  - MySQL Server - see Chapter 2, *Installing and Upgrading MySQL*.
  - MySQL Shell - see Installing MySQL Shell.
  - MySQL Router - see Installing MySQL Router.
- On Windows you can use the MySQL Installer for Windows for a sandbox deployment. For details, see High Availability.

Once you have installed the software required by InnoDB cluster choose to follow either Section 21.2.6, “Sandbox Deployment of InnoDB Cluster” or Section 21.2.4, “Production Deployment of InnoDB Cluster”.

21.2.4 Production Deployment of InnoDB Cluster

When working in a production environment, the MySQL server instances which make up an InnoDB cluster run on multiple host machines as part of a network rather than on single machine as described in Section 21.2.6, “Sandbox Deployment of InnoDB Cluster”. Before proceeding with these instructions you must install the required software to each machine that you intend to add as a server instance to your cluster, see Section 21.2.3, “Methods of Installing”.

The following diagram illustrates the scenario you work with in this section:


Client Application
MySQL Shell
Admin
MySQL Router
MySQL Server
139.59.177.10
MySQL Server
139.59.177.11
MySQL Server
139.59.177.12
Production InnoDB cluster

Important

Unlike a sandbox deployment, where all instances are deployed locally to one machine which AdminAPI has local file access to and can persist configuration changes, for a production deployment you must persist any configuration changes on the instance. How you do this depends on the version of MySQL running on the instance, see Persisting Settings.

To pass a server's connection information to AdminAPI, use URI-like connection strings or a data dictionary; see Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”. In this documentation, URI-like strings are shown.

The following sections describe how to deploy a production InnoDB cluster.

- User Privileges
- User Accounts Created by InnoDB Cluster
- Configuring Hostname
- Persisting Settings
- Verbose Logging
- Configuring Production Instances
- Creating the Cluster
- Adding Instances to a Cluster

User Privileges

The user account used to administer an instance does not have to be the root account, however the user needs to be assigned full read and write privileges on the InnoDB cluster metadata tables in addition to full MySQL administrator privileges (`SUPER`, `GRANT OPTION`, `CREATE`, `DROP` and so on). The preferred method to create users to administer the cluster is using the `clusterAdmin` option with the `dba.configureInstance()`, and *`Cluster`*`.addInstance()` operations. In this procedure the user `ic` is shown in examples.

If only read operations are needed (such as for monitoring purposes), an account with more restricted privileges can be used. See Configuring Users for InnoDB Cluster.

User Accounts Created by InnoDB Cluster

As part of using Group Replication, InnoDB cluster creates internal users which enable replication between the servers in the cluster. These users are internal to the cluster, and the user name of the generated users follows a naming scheme of `mysql_innodb_cluster_r[`*`10_numbers`*`]`. The hostname used for the internal users depends on whether the `ipWhitelist` option has been configured. If `ipWhitelist` is not configured, it defaults to `AUTOMATIC` and the internal users are created using both the wildcard `%` character and `localhost` for the hostname value. When `ipWhitelist` has been configured, for each address in the `ipWhitelist` list an internal user is created. For more information, see Creating a Whitelist of Servers.

Each internal user has a randomly generated password. The randomly generated users are given the following grants:

```
GRANT REPLICATION SLAVE ON *.* to internal_user;
```

The internal user accounts are created on the seed instance and then replicated to the other instances in the cluster. The internal users are:

- generated when creating a new cluster by issuing `dba.createCluster()`
- generated when adding a new instance to the cluster by issuing *`Cluster`*`.addInstance()`.

In addition, the *`Cluster`*`.rejoinInstance()` operation can also result in a new internal user being generated when the `ipWhitelist` option is used to specify a hostname. For example by issuing:

```
Cluster.rejoinInstance({ipWhitelist: "192.168.1.1/22"});
```

all previously existing internal users are removed and a new internal user is created, taking into account the `ipWhitelist` value used.

For more information on the internal users required by Group Replication, see Section 18.2.1.3, “User Credentials”.

Configuring Hostname

The production instances which make up a cluster run on separate machines, therefore each machine must have a unique host name and be able to resolve the host names of the other machines which run server instances in the cluster. If this is not the case, you can:

- configure each machine to map the IP of each other machine to a hostname. See your operating system documentation for details. This is the recommended solution.
- set up a DNS service
- configure the `report_host` variable in the MySQL configuration of each instance to a suitable externally reachable address

InnoDB cluster supports using IP addresses instead of host names, and addresses must be specified in IPv4 format.

In this procedure the host name `ic-`*`number`* is used in examples.

To verify whether the hostname of a MySQL server is correctly configured, execute the following query to see how the instance reports its own address to other servers and try to connect to that MySQL server from other hosts using the returned address:

```
SELECT coalesce(@@report_host, @@hostname);
```

Persisting Settings

The AdminAPI commands you use to work with a cluster and it's server instances modify the configuration of the instance. Depending on the way MySQL Shell is connected to the instance and the version of MySQL installed on the instance, these configuration changes can be persisted to the instance automatically. Persisting settings to the instance ensures that configuration changes are retained after the instance restarts, for background information see `SET PERSIST`. This is essential for reliable cluster usage, for example if settings are not persisted then an instance which has been added to a cluster does not rejoin the cluster after a restart because configuration changes are lost. Persisting changes is required after the following operations:

- `dba.configureInstance()`
- `dba.createCluster()`
- *`Cluster`*`.addInstance()`

- *Cluster*.removeInstance()
- *Cluster*.rejoinInstance()

Instances which meet the following requirements support persisting configuration changes automatically:

- the instance is running MySQL version 8.0.11 or later
- persisted_globals_load is set to ON

Instances which do not meet these requirements do not support persisting configuration changes automatically, when AdminAPI operations result in changes to the instance's settings to be persisted you receive warnings such as:

```
WARNING: On instance 'localhost:3320' membership change cannot be persisted since MySQL version 5.7.21
does not support the SET PERSIST command (MySQL version >= 8.0.5 required). Please use the
<Dba>.configureLocalInstance command locally to persist the changes.
```

When AdminAPI commands are issued against the MySQL instance which MySQL Shell is currently running on, in other words the local instance, MySQL Shell persists configuration changes directly to the instance. On local instances which support persisting configuration changes automatically, configuration changes are persisted to the instance's mysqld-auto.cnf file and the configuration change does not require any further steps. On local instances which do not support persisting configuration changes automatically, you need to make the changes locally, see Configuring Instances with dba.configureLocalInstance().

When run against a remote instance, in other words an instance other than the one which MySQL Shell is currently running on, if the instance supports persisting configuration changes automatically, the AdminAPI commands persist configuration changes to the instance's mysql-auto.conf option file. If a remote instance does not support persisting configuration changes automatically, the AdminAPI commands can not automatically configure the instance's option file. This means that AdminAPI commands can read information from the instance, for example to display the current configuration, but changes to the configuration cannot be persisted to the instance's option file. In this case, you need to persist the changes locally, see Configuring Instances with dba.configureLocalInstance().

Verbose Logging

When working with a production deployment it can be useful to configure verbose logging for MySQL Shell, the information in the log can help you to find and resolve any issues that might occur when you are preparing server instances to work as part of InnoDB cluster. To start MySQL Shell with a verbose logging level use the --log-level option:

```
shell> mysqlsh --log-level=DEBUG3
```

The DEBUG3 is recommended, see --log-level for more information. When DEBUG3 is set the MySQL Shell log file contains lines such as Debug: execute_sql(...) which contain the SQL queries that are executed as part of each AdminAPI call. The log file generated by MySQL Shell is located in ~/.mysqlsh/mysqlsh.log for Unix-based systems; on Microsoft Windows systems it is located in %APPDATA%\MySQL\mysqlsh\mysqlsh.log. See MySQL Shell Logging and Debug for more information.

In addition to enabling the MySQL Shell log level, you can configure the amount of output AdminAPI provides in MySQL Shell after issuing each command. To enable the amount of AdminAPI output, in MySQL Shell issue:

```
mysql-js> dba.verbose=2
```

This enables the maximum output from AdminAPI calls. The available levels of output are:

- 0 or OFF is the default. This provides minimal output and is the recommended level when not troubleshooting.
- 1 or ON adds verbose output from each call to the AdminAPI.
- 2 adds debug output to the verbose output providing full information about what each call to AdminAPI executes.

Configuring Production Instances

AdminAPI provides the `dba.configureInstance()` function that checks if an instance is suitably configured for InnoDB cluster usage, and configures the instance if it finds any settings which are not compatible with InnoDB cluster. You run the `dba.configureInstance()` command against an instance and it checks all of the settings required to enable the instance to be used for InnoDB cluster usage. If the instance does not require configuration changes, there is no need to modify the configuration of the instance, and the `dba.configureInstance()` command output confirms that the instance is ready for InnoDB cluster usage. If any changes are required to make the instance compatible with InnoDB cluster, a report of the incompatible settings is displayed, and you can choose to let the command make the changes to the instance's option file. Depending on the way MySQL Shell is connected to the instance, and the version of MySQL running on the instance, you can make these changes permanent by persisting them to a remote instance's option file, see Persisting Settings. Instances which do not support persisting configuration changes automatically require that you configure the instance locally, see Configuring Instances with `dba.configureLocalInstance()`. Alternatively you can make the changes to the instance's option file manually, see Section 4.2.2.2, “Using Option Files” for more information. Regardless of the way you make the configuration changes, you might have to restart MySQL to ensure the configuration changes are detected.

The syntax of the `dba.configureInstance()` command is:

```
dba.configureInstance([instance][, options])
```

where *`instance`* is an instance definition, and *`options`* is a data dictionary with additional options to configure the operation. The command returns a descriptive text message about the operation's result.

The *`instance`* definition is the connection data for the instance, see Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”. If the target instance already belongs to an InnoDB cluster an error is generated and the process fails.

The options dictionary can contain the following:

- `mycnfPath` - the path to the MySQL option file of the instance.
- `outputMycnfPath` - alternative output path to write the MySQL option file of the instance.
- `password` - the password to be used by the connection.
- `clusterAdmin` - the name of an InnoDB cluster administrator user to be created. The supported format is the standard MySQL account name format. Supports identifiers or strings for the user name and host name. By default if unquoted it assumes input is a string.
- `clusterAdminPassword` - the password for the InnoDB cluster administrator account being created using `clusterAdmin`.

- `clearReadOnly` - a boolean value used to confirm that `super_read_only` should be set to off, see Super Read-only and Instances.
- `interactive` - a boolean value used to disable the interactive wizards in the command execution, so that prompts are not provided to the user and confirmation prompts are not shown.
- `restart` - a boolean value used to indicate that a remote restart of the target instance should be performed to finalize the operation.

Although the connection password can be contained in the instance definition, this is insecure and not recommended. Use the MySQL Shell Pluggable Password Store to store instace passwords securely.

Once `dba.configureInstance()` is issued against an instance, the command checks if the instance's settings are suitable for InnoDB cluster usage. A report is displayed which shows the settings required by InnoDB cluster . If the instance does not require any changes to its settings you can use it in an InnoDB cluster, and can proceed to Creating the Cluster. If the instance's settings are not valid for InnoDB cluster usage the `dba.configureInstance()` command displays the settings which require modification. Before configuring the instance you are prompted to confirm the changes shown in a table with the following information:

- `Variable` - the invalid configuration variable.
- `Current Value` - the current value for the invalid configuration variable.
- `Required Value` - the required value for the configuration variable.

How you proceed depends on whether the instance supports persisting settings, see Persisting Settings. When `dba.configureInstance()` is issued against the MySQL instance which MySQL Shell is currently running on, in other words the local instance, it attempts to automatically configure the instance. When `dba.configureInstance()` is issued against a remote instance, if the instance supports persisting configuration changes automatically, you can choose to do this. If a remote instance does not support persisting the changes to configure it for InnoDB cluster usage, you have to configure the instance locally. See Configuring Instances with `dba.configureLocalInstance()`.

In general, a restart of the instance is not required after `dba.configureInstance()` configures the option file, but for some specific settings a restart might be required. This information is shown in the report generated after issuing `dba.configureInstance()`. If the instance supports the `RESTART` statement, MySQL Shell can shutdown and then start the instance. This ensures that the changes made to the instance's option file are detected by mysqld. For more information see `RESTART`.

Note

After executing a `RESTART` statement, the current connection to the instance is lost. If auto-reconnect is enabled, the connection is reestablished after the server restarts. Otherwise, the connection must be reestablished manually.

The `dba.configureInstance()` method verifies that a suitable user is available for cluster usage, which is used for connections between members of the cluster, see User Privileges. The recommended way to add a suitable user is to use the `clusterAdmin` and `clusterAdminPassword` options, which enable you to configure the cluster user and password when calling the function. For example:

```
mysql-js> dba.configureInstance('ic@ic-1:3306', \
{clusterAdmin: "'icadmin'@'ic-1%'", clusterAdminPassword: 'password'});
```

This user is granted the privileges to be able to administer the cluster. The format of the user names accepted follows the standard MySQL account name format, see Section 6.2.4, “Specifying Account Names”.

If you do not specify a user to administer the cluster, in interactive mode a wizard enables you to choose one of the following options:

- enable remote connections for the root user
- create a new user, the equivalent of specifying the `clusterAdmin` and `clusterAdminPassword` options
- no automatic configuration, in which case you need to manually create the user

The following example demonstrates the option to create a new user for cluster usage.

```
mysql-js> dba.configureLocalInstance('root@localhost:3306')

Please provide the password for 'root@localhost:3306':

Please specify the path to the MySQL configuration file: /etc/mysql/mysql.conf.d/mysqld.cnf
Validating instance...

The configuration has been updated but it is required to restart the server.
{
  "config_errors": [
    {
      "action": "restart",
      "current": "OFF",
      "option": "enforce_gtid_consistency",
      "required": "ON"
    },
    {
      "action": "restart",
      "current": "OFF",
      "option": "gtid_mode",
      "required": "ON"
      },
    {
      "action": "restart",
      "current": "0",
      "option": "log_bin",
      "required": "1"
    },
    {
      "action": "restart",
      "current": "0",
      "option": "log_slave_updates",
      "required": "ON"
    },
    {
      "action": "restart",
      "current": "FILE",
      "option": "master_info_repository",
      "required": "TABLE"
    },
    {
      "action": "restart",
      "current": "FILE",
      "option": "relay_log_info_repository",
      "required": "TABLE"
    },
    {
      "action": "restart",
      "current": "OFF",
      "option": "transaction_write_set_extraction",
      "required": "XXHASH64"
    }
```

```
  ],
  "errors": [],
  "restart_required": true,
  "status": "error"
}
mysql-js>
```

Tip

If the instance has `super_read_only=ON` then you might need to confirm that AdminAPI can set `super_read_only=OFF`. See Super Read-only and Instances for more information.

Creating the Cluster

Once you have prepared your instances, use the `dba.createCluster()` function to create the cluster. The machine which you are running MySQL Shell on is used as the seed instance for the cluster. The seed instance is replicated to the other instances which you add to the cluster, making them replicas of the seed instance.

MySQL Shell must be connected to an instance before you can create a cluster because when you issue `dba.createCluster(name)` MySQL Shell creates a MySQL protocol session to the server instance connected to the MySQL Shell's current global session. Use the `dba.createCluster(name)` function to create the cluster and assign the returned cluster to a variable called `cluster`:

```
mysql-js> var cluster = dba.createCluster('testCluster')

Validating instance at ic@ic-1:3306...

This instance reports its own address as ic-1

Instance configuration is suitable.
Creating InnoDB cluster 'testCluster' on 'ic@ic-1:3306'...

Adding Seed Instance...
Cluster successfully created. Use Cluster.addInstance() to add MySQL instances.
At least 3 instances are needed for the cluster to be able to withstand up to
one server failure.
```

The returned Cluster object uses a new session, independent from the MySQL Shell's main session. This ensures that if you change the MySQL Shell global session, the Cluster object maintains its session to the instance.

The `dba.createCluster()` operation supports MySQL Shell's `interactive` option. When `interactive` is on, prompts appear in the following situations:

- when run on an instance that belongs to a cluster and the `adoptFromGr` option is false, you are asked if you want to adopt an existing cluster
- when the `force` option is not used (not set to `true`), you are asked to confirm the creation of a multi-primary cluster

Note

If you encounter an error related to metadata being inaccessible you might have the loopback network interface configured. For correct InnoDB cluster usage disable the loopback interface.

To check the cluster has been created, use the cluster instance's `status()` function. See Checking a cluster's Status with `Cluster.status()`.

Tip

Once server instances belong to a cluster it is important to only administer them using MySQL Shell and AdminAPI. Attempting to manually change the configuration of Group Replication on an instance once it has been added to a cluster is not supported. Similarly, modifying server variables critical to InnoDB cluster, such as `server_uuid` after an instance is configured using AdminAPI is not supported.

When you create a cluster using MySQL Shell 8.0.14 and later, you can set the timeout before instances are expelled from the cluster, for example when they become unreachable. Pass the `expelTimeout` option to the `dba.createCluster()` operation, which configures the `group_replication_member_expel_timeout` variable on the seed instance. The `expelTimeout` option can take an integer value in the range of 0 to 3600. All instances running MySQL server 8.0.13 and later which are added to a cluster with `expelTimeout` configured are automatically configured to have the same `expelTimeout` value as configured on the seed instance.

For information on the other options which you can pass to `dba.createCluster()`, see Section 21.4, "Working with InnoDB Cluster".

Adding Instances to a Cluster

Use the *`Cluster`*`.addInstance(`*`instance`*`)` function to add more instances to the cluster, where *`instance`* is connection information to a configured instance, see Configuring Production Instances. From version 8.0.17, Group Replication implements compatibility policies which consider the patch version of the instances, and the *`Cluster`*`.addInstance()` operation detects this and in the event of an incompatibility the operation terminates with an error. See Checking the MySQL Version on Instances and Section 18.7.1, "Combining Different Member Versions in a Group"

You need a minimum of three instances in the cluster to make it tolerant to the failure of one instance. Adding further instances increases the tolerance to failure of an instance. To add an instance to the cluster issue:

```
mysql-js> cluster.addInstance('ic@ic-2:3306')
A new instance will be added to the InnoDB cluster. Depending on the amount of
data on the cluster this might take from a few seconds to several hours.

Please provide the password for 'ic@ic-2:3306': ********
Adding instance to the cluster ...

Validating instance at ic-2:3306...

This instance reports its own address as ic-2

Instance configuration is suitable.

The instance 'ic@ic-2:3306' was successfully added to the cluster.
```

If you are using MySQL 8.0.17 or later you can choose how the instance recovers the transactions it requires to synchronize with the cluster. Only when the joining instance has recovered all of the transactions previously processed by the cluster can it then join as an online instance and begin processing transactions. For more information, see Section 21.2.5, "Using MySQL Clone with InnoDB cluster".

Also in 8.0.17 and later, you can configure how *`Cluster`*`.addInstance()` behaves, letting recovery operations proceed in the background or monitoring different levels of progress in MySQL Shell.

Depending on which option you chose to recover the instance from the cluster, you see different output in MySQL Shell. Suppose that you are adding the instance ic-2 to the cluster, and ic-1 is the seed or donor.

- When you use MySQL Clone to recover an instance from the cluster, the output looks like:

```
Validating instance at ic-2:3306...

This instance reports its own address as ic-2:3306

Instance configuration is suitable.
A new instance will be added to the InnoDB cluster. Depending on the amount of
data on the cluster this might take from a few seconds to several hours.

Adding instance to the cluster...

Monitoring recovery process of the new cluster member. Press ^C to stop monitoring and let it continue i
Clone based state recovery is now in progress.

NOTE: A server restart is expected to happen as part of the clone process. If the
server does not support the RESTART command or does not come back after a
while, you may need to manually start it back.

* Waiting for clone to finish...
NOTE: ic-2:3306 is being cloned from ic-1:3306
** Stage DROP DATA: Completed
** Clone Transfer
    FILE COPY  ############################################################  100%  Completed
    PAGE COPY  ############################################################  100%  Completed
    REDO COPY  ############################################################  100%  Completed

NOTE: ic-2:3306 is shutting down...

* Waiting for server restart... ready
* ic-2:3306 has restarted, waiting for clone to finish...
** Stage RESTART: Completed
* Clone process has finished: 2.18 GB transferred in 7 sec (311.26 MB/s)

State recovery already finished for 'ic-2:3306'

The instance 'ic-2:3306' was successfully added to the cluster.
```

  The warnings about server restart should be observed, you might have to manually restart an instance. See Section 13.7.8.8, “RESTART Syntax”.

- When you use incremental recovery to recover an instance from the cluster, the output looks like:

```
Incremental distributed state recovery is now in progress.

* Waiting for incremental recovery to finish...
NOTE: 'ic-2:3306' is being recovered from 'ic-1:3306'
* Distributed recovery has finished
```

To cancel the monitoring of the recovery phase, issue **CONTROL+C**. This stops the monitoring but the recovery process continues in the background. The `waitRecovery` integer option can be used with the *`Cluster`*`.addInstance()` operation to control the behavior of the command regarding the recovery phase. The following values are accepted:

- 0: do not wait and let the recovery process finish in the background;
- 1: wait for the recovery process to finish;
- 2: wait for the recovery process to finish; and show detailed static progress information;
- 3: wait for the recovery process to finish; and show detailed dynamic progress information (progress bars);

By default, if the standard output which MySQL Shell is running on refers to a terminal, the `waitRecovery` option defaults to 3. Otherwise, it defaults to 2. See Monitoring Recovery Operations.

To verify the instance has been added, use the cluster instance's `status()` function. For example this is the status output of a sandbox cluster after adding a second instance:

```
mysql-js> cluster.status()
{
    "clusterName": "testCluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "ic-1:3306",
        "ssl": "REQUIRED",
        "status": "OK_NO_TOLERANCE",
        "statusText": "Cluster is NOT tolerant to any failures.",
        "topology": {
            "ic-1:3306": {
                "address": "ic-1:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "ic-2:3306": {
                "address": "ic-2:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
    },
    "groupInformationSourceMember": "mysql://ic@ic-1:3306"
}
```

How you proceed depends on whether the instance is local or remote to the instance MySQL Shell is running on, and whether the instance supports persisting configuration changes automatically, see Persisting Settings. If the instance supports persisting configuration changes automatically, you do not need to persist the settings manually and can either add more instances or continue to the next step. If the instance does not support persisting configuration changes automatically, you have to configure the instance locally. See Configuring Instances with `dba.configureLocalInstance()`. This is essential to ensure that instances rejoin the cluster in the event of leaving the cluster.

Tip

If the instance has `super_read_only=ON` then you might need to confirm that AdminAPI can set `super_read_only=OFF`. See Super Read-only and Instances for more information.

Once you have your cluster deployed you can configure MySQL Router to provide high availability, see Section 21.3, “Using MySQL Router with InnoDB Cluster”.

21.2.5 Using MySQL Clone with InnoDB cluster

In MySQL 8.0.17, InnoDB cluster integrates the MySQL Clone plugin to provide automatic provisioning of joining instances. The process of retrieving the cluster's data so that the instance can synchronize with the cluster is called distributed recovery. When an instance needs to recover a cluster's transactions we distinguish between the *donor*, which is the cluster instance that provides the data, and the *receiver*, which is the instance that receives the data from the donor. In previous versions, Group Replication provided

only asynchronous replication to recover the transactions required for the joining instance to synchronize with the cluster so that it could join the cluster. For a cluster with a large amount of previously processed transactions it could take a long time for the new instance to recover all of the transactions before being able to join the cluster. Or a cluster which had purged GTIDs, for example as part of regular maintenance, could be missing some of the transactions required to recover the new instance. In such cases the only alternative was to manually provision the instance using tools such as MySQL Enterprise Backup, as shown in Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”.

MySQL Clone provides an alternative way for an instance to recover the transactions required to synchronize with a cluster. Instead of relying on asynchronous replication to recover the transactions, MySQL Clone takes a snapshot of the data on the donor instance and then transfers the snapshot to the receiver.

Warning

All previous data in the receiver is destroyed during a clone operation. All MySQL settings not stored in tables are however maintained.

Once a clone operation has transferred the snapshot to the receiver, if the cluster has processed transactions while the snapshot was being transferred, asynchronous replication is used to recover any required data for the receiver to be synchronized with the cluster. This can be much more efficient than the instance recovering all of the transactions using asynchronous replication, and avoids any issues caused by purged GTIDs, enabling you to quickly provision new instances for InnoDB cluster. For more information, see Section 5.6.7, “The Clone Plugin” and Section 18.4.3.1, “Cloning for Distributed Recovery”

In contrast to using MySQL Clone, incremental recovery is the process where an instance joining a cluster uses only asynchronous replication to recover an instance from the cluster. When an InnoDB cluster is configured to use MySQL Clone, instances which join the cluster use either MySQL Clone or incremental recovery to recover the cluster's transactions. By default, the cluster automatically chooses the most suitable method, but you can optionally configure this behavior, for example to force cloning, which replaces any transactions already processed by the joining instance. When you are using MySQL Shell in interactive mode, the default, if the cluster is not sure it can proceed with recovery it provides an interactive prompt. This section describes the different options you are offered, and the different scenarios which influence which of the options you can choose.

In addition, the output of `Cluster.status()` for members in `RECOVERING` state includes recovery progress information to enable you to easily monitor recovery operations, whether they are using MySQL Clone or incremental recovery. InnoDB cluster provides additional information about instances using MySQL Clone in the output of `Cluster.status()`.

21.2.5.1 Working with a Cluster that uses MySQL Clone

A InnoDB cluster that uses MySQL Clone provides the following additional behavior.

`dba.createCluster()` and MySQL Clone

From version 8.0.17, by default when a new cluster is created on an instance where the MySQL Clone plugin is available then it is automatically installed and the cluster is configured to support cloning. The InnoDB cluster recovery accounts are created with the required `BACKUP_ADMIN` privilege.

Set the `disableClone` Boolean option to `true` to disable MySQL Clone for the cluster. In this case a metadata entry is added for this configuration and the MySQL Clone plugin is uninstalled if it is installed. You can set the `disableClone` option when you issue `dba.createCluster()`, or at any time when the cluster is running using `Cluster.setOption()`.

Cluster.addInstance(*instance*) and MySQL Clone

MySQL Clone can be used for a joining *instance* if the new instance is running MySQL 8.0.17 or later, and there is at least one donor in the cluster (included in the group_replication_group_seeds list) running MySQL 8.0.17 or later. A cluster using MySQL Clone follows the behavior documented at Adding Instances to a Cluster, with the addition of a possible choice of how to transfer the data required to recover the instance from the cluster. How *Cluster*.addInstance(*instance*) behaves depends on the following factors:

- Whether MySQL Clone is supported.
- Whether incremental recovery is possible or not, which depends on the availability of binary logs. For example, if a donor instance has all binary logs required (GTID_PURGED is empty) then incremental recovery is possible. If no cluster instance has all binary logs required then incremental recovery is not possible.
- Whether incremental recovery is appropriate or not. Even though incremental recovery might be possible, because it has the potential to clash with data already on the instance, the GTID sets on the donor and receiver are checked to make sure that incremental recovery is appropriate. The following are possible results of the comparison:
  - New: the receiver has an empty GTID_EXECUTED GTID set
  - Identical: the receiver has a GTID set identical to the donor's GTID set
  - Recoverable: the receiver has a GTID set that is missing transactions but these can be recovered from the donor
  - Irrecoverable: the donor has a GTID set that is missing transactions, possibly they have been purged
  - Diverged: the GTID sets of the donor and receiver have diverged

  When the result of the comparison is determined to be Identical or Recoverable, incremental recovery is considered appropriate. When the result of the comparison is determined to be Irrecoverable or Diverged, incremental recovery is not considered appropriate.

  For an instance considered New, incremental recovery cannot be considered appropriate because it is impossible to determine if the binary logs have been purged, or even if the GTID_PURGED and GTID_EXECUTED variables were reset. Alternatively, it could be that the server had already processed transactions before binary logs and GTIDs were enabled. Therefore in interactive mode, you have to confirm that you want to use incremental recovery.
- The state of the gtidSetIsComplete option. If you are sure a cluster has been created with a complete GTID set, and therefore instances with empty GTID sets can be added to it without extra confirmations, set the cluster level gtidSetIsComplete Boolean option to true.

Warning

Setting the gtidSetIsComplete option to true means that joining servers are recovered regardless of any data they contain, use with caution. If you try to add an instance which has applied transactions you risk data corruption.

The combination of these factors influence how instances join the cluster when you issue *Cluster*.addInstance(). The recoveryMethod option is set to auto by default, which means that in MySQL Shell's interactive mode, the cluster selects the best way to recover the instance from the cluster, and the prompts advise you how to proceed. In other words the cluster recommends using MySQL Clone or incremental recovery based on the best approach and what the server supports. If you are not using

interactive mode and are scripting MySQL Shell, you must set `recoveryMethod` to the type of recovery you want to use - either `clone` or `incremental`. This section explains the different possible scenarios.

When you are using MySQL Shell in interactive mode, the main prompt with all of the possible options for adding the instance is:

```
Please select a recovery method [C]lone/[I]ncremental recovery/[A]bort (default Clone):
```

Depending on the factors mentioned, you might not be offered all of these options. The scenarios described later in this section explain which options you are offered. The options offered by this prompt are:

- *Clone*: choose this option to clone the donor to the instance which you are adding to the cluster, deleting any transactions the instance contains. The MySQL Clone plugin is automatically installed. The InnoDB cluster recovery accounts are created with the required `BACKUP_ADMIN` privilege. Assuming you are adding an instance which is either empty (has not processed any transactions) or which contains transactions you do not want to retain, select the Clone option. The cluster then uses MySQL Clone to completely overwrite the joining instance with a snapshot from an donor cluster member. To use this method by default and disable this prompt, set the cluster's `recoveryMethod` option to `clone`.
- *Incremental recovery* choose this option to use incremental recovery to recover all transactions processed by the cluster to the joining instance using asynchronous replication. Incremental recovery is appropriate if you are sure all updates ever processed by the cluster were done with GTIDs enabled, there are no purged transactions and the new instance contains the same GTID set as the cluster or a subset of it. To use this method by default, set the `recoveryMethod` option to `incremental`.

The combination of factors mentioned influences which of these options is available at the prompt as follows:

Note

If the `group_replication_clone_threshold` system variable has been manually changed outside of AdminAPI, then the cluster might decide to use Clone recovery instead of following these scenarios.

- In a scenario where
  - incremental recovery is possible
  - incremental recovery is not appropriate
  - Clone is supported

  you can choose between any of the options. It is recommended that you use MySQL Clone, the default.
- In a scenario where
  - incremental recovery is possible
  - incremental recovery is appropriate

  you are not provided with the prompt, and incremental recovery is used.
- In a scenario where
  - incremental recovery is possible

  - incremental recovery is not appropriate
  - Clone is not supported or is disabled

  you cannot use MySQL Clone to add the instance to the cluster. You are provided with the prompt, and the recommended option is to proceed with incremental recovery.

- In a scenario where
  - incremental recovery is not possible
  - Clone is not supported or is disabled

  you cannot add the instance to the cluster and an `ERROR: The target instance must be either cloned or fully provisioned before it can be added to the target cluster. Cluster.addInstance: Instance provisioning required (RuntimeError)` is shown. This could be the result of binary logs being purged from all cluster instances. It is recommended to use MySQL Clone, by either upgrading the cluster or setting the `disableClone` option to `false`.

- In a scenario where
  - incremental recovery is not possible
  - Clone is supported

  you can only use MySQL Clone to add the instance to the cluster. This could be the result of the cluster missing binary logs, for example when they have been purged.

Once you select an option from the prompt, by default the progress of the instance recovering the transactions from the cluster is displayed. This monitoring enables you to check the recovery phase is working and also how long it should take for the instance to join the cluster and come online. To cancel the monitoring of the recovery phase, issue **CONTROL+C**.

`Cluster``.checkInstanceState()` and MySQL Clone

When the *`Cluster`*`.checkInstanceState()` operation is run to verify an instance against a cluster that is using MySQL Clone, if the instance does not have the binary logs, for example because they were purged but Clone is available and not disabled (`disableClone` is `false`) the operation provides a warning that the Clone can be used. For example:

```
The cluster transactions cannot be recovered on the instance, however,
Clone is available and can be used when adding it to a cluster.

{
"reason": "all_purged",
"state": "warning"
}
```

Similarly, on an instance where Clone is either not available or has been disabled and the binary logs are not available, for example because they were purged, then the output includes:

```
The cluster transactions cannot be recovered on the instance.

{
"reason": "all_purged",
"state": "warning"
}
```

`dba.checkInstanceConfiguration()` and MySQL Clone

When the `dba.checkInstanceConfiguration()` operation is run against an instance that has MySQL Clone available but it is disabled, a warning is displayed.

21.2.6 Sandbox Deployment of InnoDB Cluster

This section explains how to set up a sandbox InnoDB cluster deployment. You create and administer your InnoDB clusters using MySQL Shell with the included AdminAPI. This section assumes familiarity with MySQL Shell, see MySQL Shell 8.0 (part of MySQL 8.0) for further information.

Initially deploying and using local sandbox instances of MySQL is a good way to start your exploration of InnoDB cluster. You can fully test out InnoDB cluster locally, prior to deployment on your production servers. MySQL Shell has built-in functionality for creating sandbox instances that are correctly configured to work with Group Replication in a locally deployed scenario.

Important

Sandbox instances are only suitable for deploying and running on your local machine for testing purposes. In a production environment the MySQL Server instances are deployed to various host machines on the network. See Section 21.2.4, “Production Deployment of InnoDB Cluster” for more information.

This tutorial shows how to use MySQL Shell to create an InnoDB cluster consisting of three MySQL server instances.

- Deploying Sandbox Instances
- Creating the Sandbox InnoDB Cluster
- Adding Instances to an InnoDB Cluster
- Persisting the Sandbox Configuration

Deploying Sandbox Instances

MySQL Shell includes the AdminAPI that adds the `dba` global variable, which provides functions for administration of sandbox instances. In this example setup, you create three sandbox instances using `dba.deploySandboxInstance()`.

Start MySQL Shell from a command prompt by issuing the command:

```
shell> mysqlsh
```

MySQL Shell provides two scripting language modes, JavaScript and Python, in addition to a native SQL mode. Throughout this guide MySQL Shell is used primarily in JavaScript mode . When MySQL Shell starts it is in JavaScript mode by default. Switch modes by issuing `\js` for JavaScript mode, `\py` for Python mode, and `\sql` for SQL mode. Ensure you are in JavaScript mode by issuing the `\js` command, then execute:

```
mysql-js> dba.deploySandboxInstance(3310)
```

Note

Terminating commands with a semi-colon is not required in JavaScript and Python modes.

The argument passed to `deploySandboxInstance()` is the TCP port number where the MySQL Server instance listens for connections. By default the sandbox is created in a directory named `$HOME/mysql-sandboxes/port` on Unix systems. For Microsoft Windows systems the directory is `%userprofile%\MySQL\mysql-sandboxes\port`.

The root user's password for the instance is prompted for.

Important

Each instance has its own password. Defining the same password for all sandboxes in this tutorial makes it easier, but remember to use different passwords for each instance in production deployments.

To deploy further sandbox server instances, repeat the steps followed for the sandbox instance at port 3310, choosing different port numbers. For each additional sandbox instance issue:

```
mysql-js> dba.deploySandboxInstance(port_number)
```

To follow this tutorial, use port numbers 3310, 3320 and 3330 for the three sandbox server instances. Issue:

```
mysql-js> dba.deploySandboxInstance(3320)
mysql-js> dba.deploySandboxInstance(3330)
```

Creating the Sandbox InnoDB Cluster

The next step is to create the InnoDB cluster while connected to the seed MySQL Server instance. The seed instance contains the data that you want to replicate to the other instances. In this example the sandbox instances are blank, therefore we can choose any instance.

Connect MySQL Shell to the seed instance, in this case the one at port 3310:

```
mysql-js> \connect root@localhost:3310
```

The `\connect` MySQL Shell command is a shortcut for the `shell.connect()` method:

```
mysql-js> shell.connect('root@localhost:3310')
```

Once you have connected, AdminAPI can write to the local instance's option file. This is different to working with a production deployment, where you would need to connect to the remote instance and run the MySQL Shell application locally on the instance before AdminAPI can write to the instance's option file.

Use the `dba.createCluster()` method to create the InnoDB cluster with the currently connected instance as the seed:

```
mysql-js> var cluster = dba.createCluster('testCluster')
```

The `createCluster()` method deploys the InnoDB cluster metadata to the selected instance, and adds the instance you are currently connected to as the seed instance. The `createCluster()` method returns the created cluster, in the example above this is assigned to the `cluster` variable. The parameter passed to the `createCluster()` method is a symbolic name given to this InnoDB cluster, in this case `testCluster`.

Tip

If the instance has `super_read_only=ON` then you might need to confirm that AdminAPI can set `super_read_only=OFF`. See Super Read-only and Instances for more information.

Adding Instances to an InnoDB Cluster

The next step is to add more instances to the InnoDB cluster. Any transactions that were executed by the seed instance are re-executed by each secondary instance as it is added. This tutorial uses the sandbox instances that were created earlier at ports 3320 and 3330.

The seed instance in this example was recently created, so it is nearly empty. Therefore, there is little data that needs to be replicated from the seed instance to the secondary instances. In a production environment, where you have an existing database on the seed instance, you could use a tool such as MySQL Enterprise Backup to ensure that the secondaries have matching data before replication starts. This avoids the possibility of lengthy delays while data replicates from the primary to the secondaries. See Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”.

Add the second instance to the InnoDB cluster:

```
mysql-js> cluster.addInstance('root@localhost:3320')
```

The root user's password is prompted for.

Add the third instance:

```
mysql-js> cluster.addInstance('root@localhost:3330')
```

The root user's password is prompted for.

At this point you have created a cluster with three instances: a primary, and two secondaries.

Tip

You can only specify `localhost` in `addInstance()` if the instance is a sandbox instance. This also applies to the implicit `addInstance()` after issuing `createCluster()`.

Persisting the Sandbox Configuration

Once the sandbox instances have been added to the cluster, the configuration required for InnoDB cluster must be persisted to each of the instance's option files. How you proceed depends on whether the instance supports persisting configuration changes automatically, see Persisting Settings. When the MySQL instance which you are using supports persisting configuration changes automatically, adding the instance automatically configures the instance. When the MySQL instance which you are using does not support persisting configuration changes automatically, you have to configure the instance locally. See Configuring Instances with `dba.configureLocalInstance()`.

To check the cluster has been created, use the cluster instance's `status()` function. See Checking a cluster's Status with `Cluster.status()`.

Once you have your cluster deployed you can configure MySQL Router to provide high availability, see Section 21.3, “Using MySQL Router with InnoDB Cluster”.

21.2.7 Adopting a Group Replication Deployment

If you have an existing deployment of Group Replication and you want to use it to create a cluster, pass the `adoptFromGR` option to the `dba.createCluster()` function. The created InnoDB cluster matches whether the replication group is running as single-primary or multi-primary.

To adopt an existing Group Replication group, connect to a group member using MySQL Shell. In the following example a single-primary group is adopted. We connect to `gr-member-2`, a secondary instance, while `gr-member-1` is functioning as the group's primary. Create a cluster using `dba.createCluster()`, passing in the `adoptFromGR` option. For example:

```
mysql-js> var cluster = dba.createCluster('prodCluster', {adoptFromGR: true});

A new InnoDB cluster will be created on instance 'root@gr-member-2:3306'.

Creating InnoDB cluster 'prodCluster' on 'root@gr-member-2:3306'...
Adding Seed Instance...

Cluster successfully created. Use cluster.addInstance() to add MySQL instances.
At least 3 instances are needed for the cluster to be able to withstand up to
one server failure.
```

Tip

If the instance has `super_read_only=ON` then you might need to confirm that AdminAPI can set `super_read_only=OFF`. See Super Read-only and Instances for more information.

The new cluster matches the mode of the group. If the adopted group was running in single-primary mode then a single-primary cluster is created. If the adopted group was running in multi-primary mode then a multi-primary cluster is created.

21.3 Using MySQL Router with InnoDB Cluster

This section describes how to use MySQL Router with InnoDB cluster to achieve high availability. Regardless of whether you have deployed a sandbox or production cluster, MySQL Router can configure itself based on the InnoDB cluster's metadata using the `--bootstrap` option. This configures MySQL Router automatically to route connections to the cluster's server instances. Client applications connect to the ports MySQL Router provides, without any need to be aware of the InnoDB cluster topology. In the event of a unexpected failure, the InnoDB cluster adjusts itself automatically and MySQL Router detects the change. This removes the need for your client application to handle failover. For more information, see Routing for MySQL InnoDB cluster.

Note

Do not attempt to configure MySQL Router manually to redirect to the ports of an InnoDB cluster. Always use the `--bootstrap` option as this ensures that MySQL Router takes its configuration from the InnoDB cluster's metadata. See Cluster Metadata and State.

The recommended deployment of MySQL Router is on the same host as the application. When using a sandbox deployment, everything is running on a single host, therefore you deploy MySQL Router to the same host. When using a production deployment, we recommend deploying one MySQL Router instance to each machine used to host one of your client applications. It is also possible to deploy MySQL Router to a common machine through which your application instances connect.

Assuming MySQL Router is already installed (see Installing MySQL Router), use the `--bootstrap` option to provide the location of a server instance that belongs to the InnoDB cluster. MySQL Router uses the included metadata cache plugin to retrieve the InnoDB cluster's metadata, consisting of a list of server instance addresses which make up the InnoDB cluster and their role in the cluster. You pass the URI-like connection string of the server that MySQL Router should retrieve the InnoDB cluster metadata from. For example:

```
shell> mysqlrouter --bootstrap ic@ic-1:3306 --user=mysqlrouter
```

You are prompted for the instance password and encryption key for MySQL Router to use. This encryption key is used to encrypt the instance password used by MySQL Router to connect to the cluster. The ports you can use to connect to the InnoDB cluster are also displayed. The MySQL Router bootstrap process creates a `mysqlrouter.conf` file, with the settings based on the cluster metadata retrieved from the address passed to the `--bootstrap` option, in the above example `ic@ic-1:3306`. Based on the InnoDB cluster metadata retrieved, MySQL Router automatically configures the `mysqlrouter.conf` file, including a `metadata_cache` section. If you are using MySQL Router 8.0.14 and later, the `--bootstrap` option automatically configures MySQL Router to track and store active MySQL InnoDB cluster Metadata server addresses at the path configured by `dynamic_state`. This ensures that when MySQL Router is restarted it knows which MySQL InnoDB cluster Metadata server addresses are current. For more information see the `dynamic_state` documentation.

In earlier MySQL Router versions, metadata server information was defined during Router's initial bootstrap operation and stored statically as `bootstrap_server_addresses` in the configuration file, which contained the addresses for all server instances in the cluster. For example:

```
[metadata_cache:prodCluster]
router_id=1
bootstrap_server_addresses=mysql://ic@ic-1:3306,mysql://ic@ic-2:3306,mysql://ic@ic-3:3306
user=mysql_router1_jy95yozko3k2
metadata_cluster=prodCluster
ttl=300
```

Tip

If using MySQL Router 8.0.13 or earlier, when you change the topology of a cluster by adding another server instance after you have bootstrapped MySQL Router, you need to update `bootstrap_server_addresses` based on the updated metadata. Either restart MySQL Router using the `--bootstrap` option, or manually edit the `bootstrap_server_addresses` section of the `mysqlrouter.conf` file and restart MySQL Router.

The generated MySQL Router configuration creates TCP ports which you use to connect to the cluster. Ports for communicating with the cluster using both Classic MySQL protocol and X Protocol are created. To use X Protocol the server instances must have X Plugin installed and configured. For a sandbox deployment, instances have X Plugin set up automatically. For a production deployment, if you want to use X Protocol you need to install and configure X Plugin on each instance, see Setting Up MySQL as a Document Store. The default available TCP ports are:

- `6446` - for Classic MySQL protocol read-write sessions, which MySQL Router redirects incoming connections to primary server instances.
- `6447` - for Classic MySQL protocol read-only sessions, which MySQL Router redirects incoming connections to one of the secondary server instances.
- `64460` - for X Protocol read-write sessions, which MySQL Router redirects incoming connections to primary server instances.

- `64470` - for X Protocol read-only sessions, which MySQL Router redirects incoming connections to one of the secondary server instances.

Depending on your MySQL Router configuration the port numbers might be different to the above. For example if you use the `--conf-base-port` option, or the `group_replication_single_primary_mode` variable. The exact ports are listed when you start MySQL Router.

The way incoming connections are redirected depends on the type of cluster being used. When using a single-primary cluster, by default MySQL Router publishes a X Protocol and a classic protocol port, which clients connect to for read-write sessions and which are redirected to the cluster's single primary. With a multi-primary cluster read-write sessions are redirected to one of the primary instances in a round-robin fashion. For example, this means that the first connection to port 6446 would be redirected to the ic-1 instance, the second connection to port 6446 would be redirected to the ic-2 instance, and so on. For incoming read-only connections MySQL Router redirects connections to one of the secondary instances, also in a round-robin fashion. To modify this behavior see the `routing_strategy` option.

Once bootstrapped and configured, start MySQL Router. If you used a system wide install with the `--bootstrap` option then issue:

```
shell> mysqlrouter &
```

If you installed MySQL Router to a directory using the `--directory` option, use the `start.sh` script found in the directory you installed to. Alternatively set up a service to start MySQL Router automatically when the system boots, see Starting MySQL Router. You can now connect a MySQL client, such as MySQL Shell to one of the incoming MySQL Router ports as described above and see how the client gets transparently connected to one of the InnoDB cluster instances.

```
shell> mysqlsh --uri root@localhost:6442
```

To verify which instance you are actually connected to, simply issue an SQL query against the `port` status variable.

```
mysql-js> \sql
Switching to SQL mode... Commands end with ;
mysql-sql> select @@port;
+--------+
| @@port |
+--------+
|   3310 |
+--------+
```

Testing High Availability

To test if high availability works, simulate an unexpected halt by killing an instance. The cluster detects the fact that the instance left the cluster and reconfigures itself. Exactly how the cluster reconfigures itself depends on whether you are using a single-primary or multi-primary cluster, and the role the instance serves within the cluster.

In single-primary mode:

- If the current primary leaves the cluster, one of the secondary instances is elected as the new primary, with instances prioritized by the lowest `server_uuid`. MySQL Router redirects read-write connections to the newly elected primary.

- If a current secondary leaves the cluster, MySQL Router stops redirecting read-only connections to the instance.

For more information see Section 18.1.3.1, “Single-Primary Mode”.

In multi-primary mode:

- If a current "R/W" instance leaves the cluster, MySQL Router redirects read-write connections to other primaries. If the instance which left was the last primary in the cluster then the cluster is completely gone and you cannot connect to any MySQL Router port.

For more information see Section 18.1.3.2, “Multi-Primary Mode”.

There are various ways to simulate an instance leaving a cluster, for example you can forcibly stop the MySQL server on an instance, or use the AdminAPI `dba.killSandboxInstance()` if testing a sandbox deployment. In this example assume there is a single-primary sandbox cluster deployment with three server instances and the instance listening at port 3310 is the current primary. Simulate the instance leaving the cluster unexpectedly:

```
mysql-js> dba.killSandboxInstance(3310)
```

The cluster detects the change and elects a new primary automatically. Assuming your session is connected to port 6446, the default read-write classic MySQL protocol port, MySQL Router should detect the change to the cluster's topology and redirect your session to the newly elected primary. To verify this, switch to SQL mode in MySQL Shell using the `\sql` command and select the instance's `port` variable to check which instance your session has been redirected to. Notice that the first `SELECT` statement fails as the connection to the original primary was lost. This means the current session has been closed, MySQL Shell automatically reconnects for you and when you issue the command again the new port is confirmed.

```
mysql-js> \sql
Switching to SQL mode... Commands end with ;
mysql-sql> SELECT @@port;
ERROR: 2013 (HY000): Lost connection to MySQL server during query
The global session got disconnected.
Attempting to reconnect to 'root@localhost:6446'...
The global session was successfully reconnected.
mysql-sql> SELECT @@port;
+--------+
| @@port |
+--------+
|   3330 |
+--------+
1 row in set (0.00 sec)
```

In this example, the instance at port 3330 has been elected as the new primary. This shows that the InnoDB cluster provided us with automatic failover, that MySQL Router has automatically reconnected us to the new primary instance, and that we have high availability.

MySQL Router and Metadata Servers

When MySQL Router is bootstrapped against a cluster, it records the server instance's addresses in its configuration file. If any additional instances are added to the cluster after bootstrapping the MySQL Router, they are not automatically detected and therefore are *not* used for connection routing.

To ensure that newly added instances are routed to correctly you must bootstrap MySQL Router against the cluster to read the updated metadata. This means that you must restart MySQL Router and include the `--bootstrap` option.

21.4 Working with InnoDB Cluster

This section explains how to work with InnoDB cluster, and how to handle common administration tasks.

- Using `dba.checkInstanceConfiguration()`
- Configuring Instances with `dba.configureLocalInstance()`
- Retrieving an InnoDB cluster with `dba.getCluster()`
- Using `cluster.describe()`
- Checking a cluster's Status with *`Cluster`*`.status()`
- Monitoring Recovery Operations
- InnoDB cluster and Group Replication Protocol
- Checking the MySQL Version on Instances
- Super Read-only and Instances
- Configuring Users for InnoDB Cluster
- Configuring Automatic Rejoin of Instances
- Managing Sandbox Instances
- Removing Instances from the InnoDB Cluster
- Customizing InnoDB clusters
- Rejoining a Cluster
- Restoring a Cluster from Quorum Loss
- Rebooting a Cluster from a Major Outage
- Rescanning a Cluster
- Checking Instance State
- Dissolving an InnoDB Cluster
- Securing your Cluster
- Creating a Whitelist of Servers
- Scripting AdminAPI
- Configuring the Election Process
- Configuring Failover Consistency
- Changing a Cluster's Topology
- Setting Options for InnoDB cluster
- InnoDB cluster and Auto-increment

Using dba.checkInstanceConfiguration()

Before creating a production deployment from server instances you need to check that MySQL on each instance is correctly configured. In addition to dba.configureInstance(), which checks the configuration as part of configuring an instance, you can use the dba.checkInstanceConfiguration() function. This ensures that the instance satisfies the Section 21.2.2, “InnoDB Cluster Requirements” without changing any configuration on the instance. This does not check any data that is on the instance, see Checking Instance State for more information. The following demonstrates issuing this in a running MySQL Shell:

```
mysql-js> dba.checkInstanceConfiguration('ic@ic-1:3306')
Please provide the password for 'ic@ic-1:3306': ***
Validating MySQL instance at ic-1:3306 for use in an InnoDB cluster...

This instance reports its own address as ic-1
Clients and other cluster members will communicate with it through this address by default.
If this is not correct, the report_host MySQL system variable should be changed.

Checking whether existing tables comply with Group Replication requirements...
No incompatible tables detected

Checking instance configuration...

Some configuration options need to be fixed:
+--------------------------+---------------+----------------+-----------------------------------------
| Variable                 | Current Value | Required Value | Note
+--------------------------+---------------+----------------+-----------------------------------------
| binlog_checksum          | CRC32         | NONE           | Update the server variable
| enforce_gtid_consistency | OFF           | ON             | Update read-only variable and restart the se
| gtid_mode                | OFF           | ON             | Update read-only variable and restart the se
| server_id                | 1             |                | Update read-only variable and restart the se
+--------------------------+---------------+----------------+-----------------------------------------

Please use the dba.configureInstance() command to repair these issues.

{
    "config_errors": [
        {
            "action": "server_update",
            "current": "CRC32",
            "option": "binlog_checksum",
            "required": "NONE"
        },
        {
            "action": "restart",
            "current": "OFF",
            "option": "enforce_gtid_consistency",
            "required": "ON"
        },
        {
            "action": "restart",
            "current": "OFF",
            "option": "gtid_mode",
            "required": "ON"
        },
        {
            "action": "restart",
            "current": "1",
            "option": "server_id",
            "required": ""
        }
    ],
    "status": "error"
}
```

Repeat this process for each server instance that you plan to use as part of your cluster. The report generated after running dba.checkInstanceConfiguration() provides information about any configuration changes required before you can proceed. The action field in the config_error section of the report tells you whether MySQL on the instance requires a restart to detect any change made to the configuration file.

Configuring Instances with dba.configureLocalInstance()

Instances which do not support persisting configuration changes automatically (see Persisting Settings) require you to connect to the server, run MySQL Shell, connect to the instance locally and issue dba.configureLocalInstance(). This enables MySQL Shell to modify the instance's option file after running the following commands against a remote instance:

- dba.configureInstance()
- dba.createCluster()
- *Cluster*.addInstance()
- *Cluster*.removeInstance()
- *Cluster*.rejoinInstance()

Important

Failing to persist configuration changes to an instance's option file can result in the instance not rejoining the cluster after the next restart.

The recommended method is to log in to the remote machine, for example using SSH, run MySQL Shell as the root user and then connect to the local MySQL server. For example, use the --uri option to connect to the local *instance*:

```
shell> sudo -i mysqlsh --uri=instance
```

Alternatively use the \connect command to log in to the local instance. Then issue dba.configureInstance(*instance*), where *instance* is the connection information to the local instance, to persist any changes made to the local instance's option file.

```
mysql-js> dba.configureLocalInstance('ic@ic-2:3306')
```

Repeat this process for each instance in the cluster which does not support persisting configuration changes automatically. For example if you add 2 instances to a cluster which do not support persisting configuration changes automatically, you must connect to each server and persist the configuration changes required for InnoDB cluster before the instance restarts. Similarly if you modify the cluster structure, for example changing the number of instances, you need to repeat this process for each server instance to update the InnoDB cluster metadata accordingly for each instance in the cluster.

Retrieving an InnoDB cluster with dba.getCluster()

When you create a cluster using dba.createCluster(), the operation returns a Cluster object which can be assigned to a variable. You use this object to work with the cluster, for example to add instances or check the cluster's status. If you want to retrieve a cluster again at a later date, for example after restarting MySQL Shell, use the dba.getCluster([*name*],[*options*]) function. For example:

```
mysql-js> var cluster1 = dba.getCluster()
```

If you do not specify a cluster *name* then the default cluster is returned. By default MySQL Shell attempts to connect to the primary instance of the cluster when you use dba.getCluster(). Set the connectToPrimary option to configure this behavior. If connectToPrimary is true and the active global MySQL Shell session is not to a primary instance, the cluster is queried for the primary member and the cluster object connects to it. If there is no quorum in the cluster, the operation fails. If connectToPrimary is false, the cluster object uses the active session, in other words the same instance as the MySQL Shell's current global session. If connectToPrimary is not specified, MySQL Shell treats connectToPrimary as true, and falls back to connectToPrimary being false.

To force connecting to a secondary when getting a cluster, establish a connection to the secondary member of the cluster and use the connectToPrimary option by issuing:

```
mysql-js> shell.connect(secondary_member)
mysql-js> var cluster1 = dba.getCluster(testCluster, {connectToPrimary:false})
```

Tip

Remember that secondary instances have super_read_only=ON, so you cannot write changes to them.

Using cluster.describe()

To get information about the structure of the InnoDB cluster itself, use the *Cluster*.describe() function:

```
mysql-js> cluster.describe();
{
    "clusterName": "testCluster",
    "defaultReplicaSet": {
        "name": "default",
        "topology": [
            {
                "address": "ic-1:3306",
                "label": "ic-1:3306",
                "role": "HA"
            },
            {
                "address": "ic-2:3306",
                "label": "ic-2:3306",
                "role": "HA"
            },
            {
                "address": "ic-3:3306",
                "label": "ic-3:3306",
                "role": "HA"
            }
        ]
    }
}
```

The output from this function shows the structure of the InnoDB cluster including all of its configuration information, and so on. The address, label and role values match those described at Checking a cluster's Status with *Cluster*.status() .

Checking a cluster's Status with *Cluster*.status()

Cluster objects provide the status() method that enables you to check how a cluster is running. Before you can check the status of the InnoDB cluster, you need to get a reference to the InnoDB cluster object

by connecting to any of its instances. However, if you want to make changes to the configuration of the cluster, you must connect to a "R/W" instance. Issuing status() retrieves the status of the cluster based on the view of the cluster which the server instance you are connected to is aware of and outputs a status report.

Important

The instance's state in the cluster directly influences the information provided in the status report. Therefore ensure the instance you are connected to has a status of ONLINE.

For information about how the InnoDB cluster is running, use the cluster's status() method:

```
mysql-js> var cluster = dba.getCluster()
mysql-js> cluster.status()
{
    "clusterName": "testcluster",
    "defaultReplicaSet": {
        "name": "default",
        "primary": "ic-1:3306",
        "ssl": "REQUIRED",
        "status": "OK",
        "statusText": "Cluster is ONLINE and can tolerate up to ONE failure.",
        "topology": {
            "ic-1:3306": {
                "address": "ic-1:3306",
                "mode": "R/W",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "ic-2:3306": {
                "address": "ic-2:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            },
            "ic-3:3306": {
                "address": "ic-3:3306",
                "mode": "R/O",
                "readReplicas": {},
                "role": "HA",
                "status": "ONLINE"
            }
        }
    },
    "groupInformationSourceMember": "mysql://ic@ic-1:3306"
}
```

The output of *Cluster*.status() provides the following information:

- clusterName: name assigned to this cluster during dba.createCluster().
- defaultReplicaSet: the server instances which belong to an InnoDB cluster and contain the data set.
- primary: displayed when the cluster is operating in single-primary mode only. Shows the address of the current primary instance. If this field is not displayed, the cluster is operating in multi-primary mode.
- ssl: whether secure connections are used by the cluster or not. Shows values of REQUIRED or DISABLED, depending on how the memberSslMode option was configured during either createCluster() or addInstance(). The value returned by this parameter corresponds to the

value of the group_replication_ssl_mode server variable on the instance. See Securing your Cluster.

- status: The status of this element of the cluster. For the overall cluster this describes the high availability provided by this cluster. The status is one of the following:
  - ONLINE: The instance is online and participating in the cluster.
  - OFFLINE: The instance has lost connection to the other instances.
  - RECOVERING: The instance is attempting to synchronize with the cluster by retrieving transactions it needs before it can become an ONLINE member.
  - UNREACHABLE: The instance has lost communication with the cluster.
  - ERROR: The instance has encountered an error during the recovery phase or while applying a transaction.

    Important

    Once an instance enters ERROR state, the super_read_only option is set to ON. To leave the ERROR state you must manually configure the instance with super_read_only=OFF.

  - (MISSING): The state of an instance which is part of the configured cluster, but is currently unavailable.

    Note

    The MISSING state is specific to InnoDB cluster, it is not a state generated by Group Replication. MySQL Shell uses this state to indicate instances that are registered in the metadata, but cannot be found in the live cluster view.

- topology: The instances which have been added to the cluster.
- Host name of instance: The host name of an instance, for example localhost:3310.
- role: what function this instance provides in the cluster. Currently only HA, for high availability.
- mode: whether the server is read-write ("R/W") or read-only ("R/O"). From version 8.0.17, this is derived from the current state of the super_read_only variable on the instance, and whether the cluster has quorum. In previous versions the value of mode was derived from whether the instance was serving as a primary or secondary instance. Usually if the instance is a primary, then the mode is "R/W", and if the instance is a secondary the mode is "R/O". Any instances in a cluster that have no visible quorum are marked as "R/O", regardless of the state of the super_read_only variable.
- groupInformationSourceMember: the internal connection used to get information about the cluster, shown as a URI-like connection string. Usually the connection initially used to create the cluster.

To display more information about the cluster use the extended option. From version 8.0.17, the extended option supports integer or Boolean values. To configure the additional information that *Cluster*.status({'extended':*value*}) provides, use the following values:

- 0: disables the additional information, the default
- 1: includes information about the Group Replication Protocol Version, Group name, cluster member UUIDs, cluster member roles and states as reported by Group Replication, and the list of fenced system variables

- 2: includes information about transactions processed by connection and applier
- 3: includes more detailed statistics about the replication performed by each cluster member.

Setting extended using Boolean values is the equivalent of setting the integer values 0 and 1. In versions prior to 8.0.17, the extended option was only Boolean. Similarly prior versions used the queryMembers Boolean option to provide more information about the instances in the cluster, which is the equivalent of setting extended to 3. The queryMembers option is deprecated and scheduled to be removed in a future release.

When you issue *Cluster*.status({'extended':1}), or the extended option is set to true, the output includes:

- the following additional attributes for the defaultReplicaSet object:
  - GRProtocolVersion is the Group Replication Protocol Version being used in the cluster.

Tip

InnoDB cluster manages the Group Replication Protocol version being used automatically, see InnoDB cluster and Group Replication Protocol for more information.

  - groupName is the group's name, a UUID.
- the following additional attributes for each object of the topology object:
  - fenceSysVars a list containing the name of the fenced system variables which are enabled. Currently the fenced system variables considered are read_only, super_read_only and offline_mode.
  - memberId Each cluster member UUID.
  - memberRole the Member Role as reported by the Group Replication plugin, see the MEMBER_ROLE column of the replication_group_members table.
  - memberState the Member State as reported by the Group Replication plugin, see the MEMBER_STATE column of the replication_group_members table.

To see information about recovery and regular transaction I/O, applier worker thread statistics and any lags; applier coordinator statistics, if parallel apply is enabled; error, and other information from I/O and applier threads issue use the values 2 and 3. A value of 3 is the equivalent of setting the deprecated queryMembers option to true. When you use these values, a connection to each instance in the cluster is opened so that additional instance specific statistics can be queried. The exact statistics that are included in the output depend on the state and configuration of the instance and the server version. This information matches that shown in the replication_group_member_stats table, see the descriptions of the matching columns for more information. Instances which are ONLINE have a transactions section included in the output. Instances which are RECOVERING have a recovery section included in the output. When you set extended to 2, in either case, these sections can contain the following:

- appliedCount: see COUNT_TRANSACTIONS_REMOTE_APPLIED
- checkedCount: see COUNT_TRANSACTIONS_CHECKED
- committedAllMembers: see TRANSACTIONS_COMMITTED_ALL_MEMBERS
- conflictsDetectedCount: see COUNT_CONFLICTS_DETECTED

- inApplierQueueCount: see COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE
- inQueueCount: see COUNT_TRANSACTIONS_IN_QUEUE
- lastConflictFree: see LAST_CONFLICT_FREE_TRANSACTION
- proposedCount: see COUNT_TRANSACTIONS_LOCAL_PROPOSED
- rollbackCount: see COUNT_TRANSACTIONS_LOCAL_ROLLBACK

When you set extended to 3, the connection section shows information from the replication_connection_status table. The connection section can contain the following:

The currentlyQueueing section has information about the transactions currently queued:

- immediateCommitTimestamp: see QUEUEING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP
- immediateCommitToNowTime: see QUEUEING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP minus NOW()
- originalCommitTimestamp: see QUEUEING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
- originalCommitToNowTime: see QUEUEING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP minus NOW()
- startTimestamp: see QUEUEING_TRANSACTION_START_QUEUE_TIMESTAMP
- transaction: see QUEUEING_TRANSACTION
- lastHeartbeatTimestamp: see LAST_HEARTBEAT_TIMESTAMP

The lastQueued section has information about the most recently queued transaction:

- endTimestamp: see LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP
- immediateCommitTimestamp: see LAST_QUEUED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP
- immediateCommitToEndTime: LAST_QUEUED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP minus NOW()
- originalCommitTimestamp: see LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
- originalCommitToEndTime: LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP minus NOW()
- queueTime: LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP minus LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP
- startTimestamp: see LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP
- transaction: see LAST_QUEUED_TRANSACTION
- receivedHeartbeats: see COUNT_RECEIVED_HEARTBEATS
- receivedTransactionSet: see RECEIVED_TRANSACTION_SET
- threadId: see THREAD_ID

Instances which are using a multithreaded slave have a workers section which contains information about the worker threads, and matches the information shown by the replication_applier_status_by_worker table.

The lastApplied section shows the following information about the last transaction applied by the worker:

- applyTime: see LAST_APPLIED_TRANSACTION_END_APPLY_TIMESTAMP minus LAST_APPLIED_TRANSACTION_START_APPLY_TIMESTAMP
- endTimestamp: see LAST_APPLIED_TRANSACTION_END_APPLY_TIMESTAMP
- immediateCommitTimestamp: see LAST_APPLIED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP
- immediateCommitToEndTime: see LAST_APPLIED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP minus NOW()
- originalCommitTimestamp: see LAST_APPLIED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
- originalCommitToEndTime: see LAST_APPLIED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP minus NOW()
- startTimestamp: see LAST_APPLIED_TRANSACTION_START_APPLY_TIMESTAMP
- transaction: see LAST_APPLIED_TRANSACTION

The currentlyApplying section shows the following information about the transaction currently being applied by the worker:

- immediateCommitTimestamp: see APPLYING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP
- immediateCommitToNowTime: see APPLYING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP minus NOW()
- originalCommitTimestamp: see APPLYING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
- originalCommitToNowTime: see APPLYING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP minus NOW()
- startTimestamp: see APPLYING_TRANSACTION_START_APPLY_TIMESTAMP
- transaction: see APPLYING_TRANSACTION

The lastProcessed section has the following information about the last transaction processed by the worker:

- bufferTime: LAST_PROCESSED_TRANSACTION_END_BUFFER_TIMESTAMP minus LAST_PROCESSED_TRANSACTION_START_BUFFER_TIMESTAMP
- endTimestamp: see LAST_PROCESSED_TRANSACTION_END_BUFFER_TIMESTAMP
- immediateCommitTimestamp: see LAST_PROCESSED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP
- immediateCommitToEndTime: LAST_PROCESSED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP minus LAST_PROCESSED_TRANSACTION_END_BUFFER_TIMESTAMP

- originalCommitTimestamp: see LAST_PROCESSED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
- originalCommitToEndTime: LAST_PROCESSED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP minus LAST_PROCESSED_TRANSACTION_END_BUFFER_TIMESTAMP
- startTimestamp: see LAST_PROCESSED_TRANSACTION_START_BUFFER_TIMESTAMP
- transaction: see LAST_PROCESSED_TRANSACTION

If parallel applier workers are enabled, then the number of objects in the workers array in transactions or recovery matches the number of configured workers and an additional coordinator object is included. The information shown matches the information in the replication_applier_status_by_coordinator table. The object can contain:

The currentlyProcessing section has the following information about the transaction being processed by the worker:

- immediateCommitTimestamp: see PROCESSING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP
- immediateCommitToNowTime: PROCESSING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP minus NOW()
- originalCommitTimestamp: see PROCESSING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP
- originalCommitToNowTime: PROCESSING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP minus NOW()
- startTimestamp: see PROCESSING_TRANSACTION_START_BUFFER_TIMESTAMP
- transaction: see PROCESSING_TRANSACTION

worker objects have the following information if an error was detected in the replication_applier_status_by_worker table:

- lastErrno: see LAST_ERROR_NUMBER
- lastError: see LAST_ERROR_MESSAGE
- lastErrorTimestamp: see LAST_ERROR_TIMESTAMP

connection objects have the following information if an error was detected in the replication_connection_status table:

- lastErrno: see LAST_ERROR_NUMBER
- lastError: see LAST_ERROR_MESSAGE
- lastErrorTimestamp: see LAST_ERROR_TIMESTAMP

coordinator objects have the following information if an error was detected in the replication_applier_status_by_coordinator table:

- lastErrno: see LAST_ERROR_NUMBER
- lastError: see LAST_ERROR_MESSAGE
- lastErrorTimestamp: see LAST_ERROR_TIMESTAMP

Monitoring Recovery Operations

The output of `Cluster.status()` shows information about the progress of recovery operations for instances in `RECOVERING` state. Information is shown for instances recovering using either MySQL Clone, or incremental recovery. Monitor these fields:

- The `recoveryStatusText` field includes information about the type of recovery being used. When MySQL Clone is working the field shows “Cloning in progress”. When incremental recovery is working the field shows “Distributed recovery in progress”.
- When MySQL Clone is being used, the `recovery` field includes a dictionary with the following fields:
  - `cloneStartTime`: The timestamp of the start of the clone process
  - `cloneState`: The state of the clone progress
  - `currentStage`: The current stage which the clone process has reached
  - `currentStageProgress`: The current stage progress as a percentage of completion
  - `currentStageState`: The current stage state

  Example `Cluster.status()` output, trimmed for brevity:

```
...
"recovery": {
"cloneStartTime": "2019-07-15 12:50:22.730",
"cloneState": "In Progress",
"currentStage": "FILE COPY",
"currentStageProgress": 61.726837675213865,
"currentStageState": "In Progress"
},
"recoveryStatusText": "Cloning in progress",
...
```

- When incremental recovery is being used, the `recovery` field includes a dictionary with the following field:
  - `state`: The state of the `group_replication_recovery` channel

  Example output `Cluster.status()`, trimmed for brevity:

```
...
"recovery": {
"state": "ON"
},
...
```

InnoDB cluster and Group Replication Protocol

From MySQL 8.0.16, Group Replication has the concept of a communication protocol for the group, see Section 18.4.1.4, “Setting a Group's Communication Protocol Version” for background information. The Group Replication communication protocol version usually has to be managed explicitly, and set to accommodate the oldest MySQL Server version that you want the group to support. However, InnoDB cluster automatically and transparently manages the communication protocol versions of its members, whenever the cluster topology is changed using AdminAPI operations. A cluster always uses the most

recent communication protocol version that is supported by all the instances that are currently part of the cluster or joining it.

- When an instance is added to, removed from, or rejoins the cluster, or a rescan or reboot operation is carried out on the cluster, the communication protocol version is automatically set to a version supported by the instance that is now at the earliest MySQL Server version.
- When you carry out a rolling upgrade by removing instances from the cluster, upgrading them, and adding them back into the cluster, the communication protocol version is automatically upgraded when the last remaining instance at the old MySQL Server version is removed from the cluster prior to its upgrade.

To see the communication protocol version being used in a cluster, use the *Cluster*.status() function with the extended option enabled. The communication protocol version is returned in the GRProtocolVersion field, provided that the cluster has quorum and no cluster members are unreachable.

Checking the MySQL Version on Instances

The following operations can report information about the MySQL Server version running on the instance:

- *Cluster*.status()
- *Cluster*.describe()
- *Cluster*.rescan()

The behavior varies depending on the MySQL Server version of the *Cluster* object session.

- *Cluster*.status()

  If either of the following requirements are met, a version string attribute is returned for each instance JSON object of the topology object:

  - The *Cluster* object's current session is version 8.0.11 or later.
  - The *Cluster* object's current session is running a version earlier than version 8.0.11 but the extended option is set to 3 (or the deprecated queryMembers is true).

  For example on an instance running version 8.0.16:

```
"topology": {
    "ic-1:3306": {
        "address": "ic-1:3306",
        "mode": "R/W",
        "readReplicas": {},
        "role": "HA",
        "status": "ONLINE",
        "version": "8.0.16"
}
```

  For example on an instance running version 5.7.24:

```
"topology": {
    "ic-1:3306": {
        "address": "ic-1:3306",
        "mode": "R/W",
        "readReplicas": {},
```

```
        "role": "HA",
        "status": "ONLINE",
        "version": "5.7.24"
}
```

- `Cluster.describe()`

  If the `Cluster` object's current session is version 8.0.11 or later, a `version` string attribute is returned for each instance JSON object of the `topology` object

  For example on an instance running version 8.0.16:

```
"topology": [
    {
        "address": "ic-1:3306",
        "label": "ic-1:3306",
        "role": "HA",
        "version": "8.0.16"
    }
]
```

- `Cluster.rescan()`

  If the `Cluster` object's current session is version 8.0.11 or later, and the `Cluster.rescan()` operation detects instances which do not belong to the cluster, a `version` string attribute is returned for each instance JSON object of the `newlyDiscoveredInstance` object.

  For example on an instance running version 8.0.16:

```
"newlyDiscoveredInstances": [
    {
        "host": "ic-4:3306",
        "member_id": "82a67a06-2ba3-11e9-8cfc-3c6aa7197deb",
        "name": null,
        "version": "8.0.16"
    }
]
```

Super Read-only and Instances

Whenever Group Replication stops, the `super_read_only` variable is set to `ON` to ensure no writes are made to the instance. When you try to use such an instance with the following AdminAPI commands you are given the choice to set `super_read_only=OFF` on the instance:

- `dba.configureInstance()`
- `dba.configureLocalInstance()`
- `dba.dropMetadataSchema()`

When AdminAPI encounters an instance which has `super_read_only=ON`, in interactive mode you are given the choice to set `super_read_only=OFF`. For example:

```
mysql-js> var myCluster = dba.dropMetadataSchema()
Are you sure you want to remove the Metadata? [y/N]: y
The MySQL instance at 'localhost:3310' currently has the super_read_only system
variable set to protect it from inadvertent updates from applications. You must
first unset it to be able to perform any changes to this instance.
For more information see:
```

```
https://dev.mysql.com/doc/refman/en/server-system-variables.html#sysvar_super_read_only.

Do you want to disable super_read_only and continue? [y/N]: y

Metadata Schema successfully removed.
```

The number of current active sessions to the instance is shown. You must ensure that no applications can write to the instance inadvertently. By answering `y` you confirm that AdminAPI can write to the instance. If there is more than one open session to the instance listed, exercise caution before permitting AdminAPI to set `super_read_only=OFF`.

To force the function to set `super_read_only=OFF` in a script, pass the `clearReadOnly` option set to `true`. For example `dba.configureInstance(`*`instance`*`, {clearReadOnly: true}).`

Configuring Users for InnoDB Cluster

The recommended way to create a user which can administer an InnoDB cluster is to use the `clusterAdmin` and `clusterAdminPassword` options with the `dba.configureInstance()` or `dba.configureLocalInstance()` operations. If you want to manually configure a user which can administer an InnoDB cluster that user requires the following privileges, all with `GRANT OPTION`:

- Global privileges on *.* for `RELOAD`, `SHUTDOWN`, `PROCESS`, `FILE`, `SELECT`, `SUPER`, `REPLICATION SLAVE`, `REPLICATION CLIENT`, `CREATE USER`.
- Schema specific privileges for `mysql_innodb_cluster_metadata.*` are `ALTER`, `ALTER ROUTINE`, `CREATE`, `CREATE ROUTINE`, `CREATE TEMPORARY TABLES`, `CREATE VIEW`, `DELETE`, `DROP`, `EVENT`, `EXECUTE`, `INDEX`, `INSERT`, LOCK TABLES, `REFERENCES`, SHOW VIEW, `TRIGGER`, `UPDATE`; and for `mysql.*` are `INSERT`, `UPDATE`, `DELETE`;

If only read operations are needed, for example to create a user for monitoring purposes, an account with more restricted privileges can be used. To give the user *`your_user`* the privileges needed to monitor InnoDB cluster issue:

```
GRANT SELECT ON mysql_innodb_cluster_metadata.* TO your_user@'%';
GRANT SELECT ON performance_schema.global_status TO your_user@'%';
GRANT SELECT ON performance_schema.replication_applier_configuration TO your_user@'%';
GRANT SELECT ON performance_schema.replication_applier_status TO your_user@'%';
GRANT SELECT ON performance_schema.replication_applier_status_by_coordinator TO your_user@'%';
GRANT SELECT ON performance_schema.replication_applier_status_by_worker TO your_user@'%';
GRANT SELECT ON performance_schema.replication_connection_configuration TO your_user@'%';
GRANT SELECT ON performance_schema.replication_connection_status TO your_user@'%';
GRANT SELECT ON performance_schema.replication_group_member_stats TO your_user@'%';
GRANT SELECT ON performance_schema.replication_group_members TO your_user@'%';
GRANT SELECT ON performance_schema.threads TO your_user@'%' WITH GRANT OPTION;
```

For more information, see Section 13.7.1, “Account Management Statements”.

Configuring Automatic Rejoin of Instances

Instances running MySQL 8.0.16 and later support the Group Replication automatic rejoin functionality, which enables you to configure instances to automatically rejoin the cluster after being expelled. See Section 18.6.6, “Responses to Failure Detection and Network Partitioning” for background information. AdminAPI provides the `autoRejoinTries` option to configure the number of tries instances make to rejoin the cluster after being expelled. By default instances do not automatically rejoin the cluster. You can configure the `autoRejoinTries` option at either the cluster level or for an individual instance using the following commands:

- `dba.createCluster()`

- `Cluster.addInstance()`
- `Cluster.setOption()`
- `Cluster.setInstanceOption()`

The `autoRejoinTries` option accepts positive integer values between 0 and 2016 and the default value is 0, which means that instances do not try to automatically rejoin. When you are using the automatic rejoin functionality, your cluster is more tolerant to faults, especially temporary ones such as unreliable networks. But if quorum has been lost, you should not expect members to automatically rejoin the cluster, because majority is required to rejoin instances.

Instances running MySQL version 8.0.12 and later have the `group_replication_exit_state_action` variable, which you can configure using the AdminAPI `exitStateAction` option. This controls what instances do in the event of leaving the cluster unexpectedly. By default the `exitStateAction` option is `READ_ONLY`, which means that instances which leave the cluster unexpectedly become read-only. If `exitStateAction` is set to `OFFLINE_MODE` (available from MySQL 8.0.18), instances which leave the cluster unexpectedly become read-only and also enter offline mode, where they disconnect existing clients and do not accept new connections (except from clients with administrator privileges). If `exitStateAction` is set to `ABORT_SERVER` then in the event of leaving the cluster unexpectedly, the instance shuts down MySQL, and it has to be started again before it can rejoin the cluster. Note that when you are using the automatic rejoin functionality, the action configured by the `exitStateAction` option only happens in the event that all attempts to rejoin the cluster fail.

There is a chance you might connect to an instance and try to configure it using the AdminAPI, but at that moment the instance could be rejoining the cluster. This could happen whenever you use any of these operations:

- `Cluster.status()`
- `dba.getCluster()`
- `Cluster.rejoinInstance()`
- `Cluster.addInstance()`
- `Cluster.removeInstance()`
- `Cluster.rescan()`
- `Cluster.checkInstanceState()`

These operations might provide extra information while the instance is automatically rejoining the cluster. In addition, when you are using `*Cluster*.removeInstance()`, if the target instance is automatically rejoining the cluster the operation aborts unless you pass in `force:true`.

Managing Sandbox Instances

Once a sandbox instance is running, it is possible to change its status at any time using the following:

- To stop a sandbox instance use `dba.stopSandboxInstance(*instance*)`. This stops the instance gracefully, unlike `dba.killSandboxInstance(*instance*)`.
- To start a sandbox instance use `dba.startSandboxInstance(*instance*)`.
- To kill a sandbox instance use `dba.killSandboxInstance(*instance*)`. This stops the instance without gracefully stopping it and is useful in simulating unexpected halts.

- To delete a sandbox instance use `dba.deleteSandboxInstance(instance)`. This completely removes the sandbox instance from your file system.

Removing Instances from the InnoDB Cluster

You can remove an instance from a cluster at any time should you wish to do so. This can be done with the `Cluster.removeInstance(instance)` method, as in the following example:

```
mysql-js> cluster.removeInstance('root@localhost:3310')

The instance will be removed from the InnoDB cluster. Depending on the instance
being the Seed or not, the Metadata session might become invalid. If so, please
start a new session to the Metadata Storage R/W instance.

Attempting to leave from the Group Replication group...

The instance 'localhost:3310' was successfully removed from the cluster.
```

You can optionally pass in the `interactive` option to control whether you are prompted to confirm the removal of the instance from the cluster. In interactive mode, you are prompted to continue with the removal of the instance (or not) in case it is not reachable. The `cluster.removeInstance()` operation ensures that the instance is removed from the metadata of all the cluster members which are `ONLINE`, and the instance itself.

When the instance being removed has transactions which still need to be applied, AdminAPI waits for up to the number of seconds configured by the MySQL Shell `dba.gtidWaitTimeout` option for transactions (GTIDs) to be applied. The MySQL Shell `dba.gtidWaitTimeout` option has a default value of 60 seconds, see Configuring MySQL Shell Options for information on changing the default. If the timeout value defined by `dba.gtidWaitTimeout` is reached when waiting for transactions to be applied and the `force` option is `false` (or not defined) then an error is issued and the remove operation is aborted. If the timeout value defined by `dba.gtidWaitTimeout` is reached when waiting for transactions to be applied and the `force` option is set to `true` then the operation continues without an error and removes the instance from the cluster.

Important

The `force` option should only be used with `Cluster.removeInstance(instance)` when you want to ignore any errors, for example unprocessed transactions or an instance being `UNREACHABLE`, and do not plan to reuse the instance with the cluster. Ignoring errors when removing an instance from the cluster could result in an instance which is not in synchrony with the cluster, preventing it from rejoining the cluster at a later time. Only use the `force` option when you plan to no longer use the instance with the cluster, in all other cases you should always try to recover the instance and only remove it when it is available and healthy, in other words with the status `ONLINE`.

Customizing InnoDB clusters

When you create a cluster and add instances to it, values such as the group name, the local address, and the seed instances are configured automatically by AdminAPI. These default values are recommended for most deployments, but advanced users can override the defaults by passing the following options to the `dba.createCluster()` and `Cluster.addInstance()`.

To customize the name of the replication group created by InnoDB cluster, pass the `groupName` option to the `dba.createCluster()` command. This sets the `group_replication_group_name` system variable. The name must be a valid UUID.

To customize the address which an instance provides for connections from other instances, pass the `localAddress` option to the `dba.createCluster()` and `cluster.addInstance()` commands. Specify the address in the format *`host:port`*. This sets the `group_replication_local_address` system variable on the instance. The address must be accessible to all instances in the cluster, and must be reserved for internal cluster communication only. In other words do not use this address for communication with the instance.

To customize the instances used as seeds when an instance joins the cluster, pass the `groupSeeds` option to the `dba.createCluster()` and `cluster.addInstance()` commands. Seed instances are contacted when a new instance joins a cluster and used to provide data to the new instance. The addresses are specified as a comma separated list such as `host1:port1`,`host2:port2`. This configures the `group_replication_group_seeds` system variable.

For more information see the documentation of the system variables configured by these AdminAPI options.

Rejoining a Cluster

If an instance leaves the cluster, for example because it lost connection, and for some reason it could not automatically rejoin the cluster, it might be necessary to rejoin it to the cluster at a later stage. To rejoin an instance to a cluster issue *`Cluster`*`.rejoinInstance(`*`instance`*`)`.

Tip

If the instance has `super_read_only=ON` then you might need to confirm that AdminAPI can set `super_read_only=OFF`. See Super Read-only and Instances for more information.

In the case where an instance has not had it's configuration persisted (see Persisting Settings), upon restart the instance does not rejoin the cluster automatically. The solution is to issue `cluster.rejoinInstance()` so that the instance is added to the cluster again and ensure the changes are persisted. Once the InnoDB cluster configuration is persisted to the instance's option file it rejoins the cluster automatically.

If you are rejoining an instance which has changed in some way then you might have to modify the instance to make the rejoin process work correctly. For example, when you restore a MySQL Enterprise Backup backup, the `server_uuid` changes. Attempting to rejoin such an instance fails because InnoDB cluster instances are identified by the `server_uuid` variable. In such a situation, information about the instance's old `server_uuid` must be removed from the InnoDB cluster metadata and then a *`Cluster`*`.rescan()` must be executed to add the instance to the metadata using it's new `server_uuid`. For example:

```
cluster.removeInstance("root@instanceWithOldUUID:3306", {force: true})

cluster.rescan()
```

In this case you must pass the `force` option to the *`Cluster`*`.removeInstance()` method because the instance is unreachable from the cluster's perspective and we want to remove it from the InnoDB cluster metadata anyway.

Restoring a Cluster from Quorum Loss

If an instance (or instances) fail, then a cluster can lose its quorum, which is the ability to vote in a new primary. This can happen when a there is a failure of enough instances that there is no longer a majority of the instances which make up the cluster to vote on Group Replication operations. When a cluster loses

quorum you can no longer process write transactions with the cluster, or change the cluster's topology, for example by adding, rejoining, or removing instances. However if you have an instance online which contains the InnoDB cluster metadata, it is possible to restore a cluster with quorum. This assumes you can connect to an instance that contains the InnoDB cluster metadata, and that instance can contact the other instances you want to use to restore the cluster.

Important

This operation is potentially dangerous because it can create a split-brain scenario if incorrectly used and should be considered a last resort. Make absolutely sure that there are no partitions of this group that are still operating somewhere in the network, but not accessible from your location.

Connect to an instance which contains the cluster's metadata, then use the `Cluster.forceQuorumUsingPartitionOf(instance)` operation, which restores the cluster based on the metadata on *instance*, and then all the instances that are `ONLINE` from the point of view of the given instance definition are added to the restored cluster.

```
mysql-js> cluster.forceQuorumUsingPartitionOf("ic@ic-1:3306")

  Restoring replicaset 'default' from loss of quorum, by using the partition composed of [ic@ic-1:3306]

  Please provide the password for 'ic@ic-1:3306': ******
  Restoring the InnoDB cluster ...

  The InnoDB cluster was successfully restored using the partition from the instance 'ic@ic-1:3306'.

  WARNING: To avoid a split-brain scenario, ensure that all other members of the replicaset
  are removed or joined back to the group that was restored.
```

In the event that an instance is not automatically added to the cluster, for example if its settings were not persisted, use `Cluster.rejoinInstance()` to manually add the instance back to the cluster.

The restored cluster might not, and does not have to, consist of all of the original instances which made up the cluster. For example, if the original cluster consisted of the following five instances:

- `ic-1`
- `ic-2`
- `ic-3`
- `ic-4`
- `ic-5`

and the cluster experiences a split-brain scenario, with `ic-1`, `ic-2`, and `ic-3` forming one partition while `ic-4` and `ic-5` form another partition. If you connect to `ic-1` and issue `Cluster.forceQuorumUsingPartitionOf('ic@ic-1:3306')` to restore the cluster the reulting cluster would consist of these three instances:

- `ic-1`
- `ic-2`
- `ic-3`

because `ic-1` sees `ic-2` and `ic-3` as `ONLINE` and does not see `ic-4` and `ic-5`.

Rebooting a Cluster from a Major Outage

If your cluster suffers from a complete outage, you can ensure it is reconfigured correctly using `dba.rebootClusterFromCompleteOutage()`. This operation takes the instance which MySQL Shell is currently connected to and uses its metadata to recover the cluster. In the event that a cluster's instances have completely stopped, the instances must be started and only then can the cluster be started. For example if the machine a sandbox cluster was running on has been restarted, and the instances were at ports 3310, 3320 and 3330, issue:

```
mysql-js> dba.startSandboxInstance(3310)
mysql-js> dba.startSandboxInstance(3320)
mysql-js> dba.startSandboxInstance(3330)
```

This ensures the sandbox instances are running. In the case of a production deployment you would have to start the instances outside of MySQL Shell. Once the instances have started, you need to connect to an instance with the GTID superset, which means the instance which had applied the most transaction before the outage. If you are unsure which instance contains the GTID superset, connect to any instance and follow the interactive messages from the `dba.rebootClusterFromCompleteOutage()`, which detects if the instance you are connected to contains the GTID superset. Reboot the cluster by issuing:

```
mysql-js> var cluster = dba.rebootClusterFromCompleteOutage();
```

The `dba.rebootClusterFromCompleteOutage()` operation then follows these steps to ensure the cluster is correctly reconfigured:

- The InnoDB cluster metadata found on the instance which MySQL Shell is currently connected to is checked to see if it contains the GTID superset, in other words the transactions applied by the cluster. If the currently connected instance does not contain the GTID superset, the operation aborts with that information. See the subsequent paragraphs for more information.
- If the instance contains the GTID superset, the cluster is recovered based on the metadata of the instance.
- Assuming you are running MySQL Shell in interactive mode, a wizard is run that checks which instances of the cluster are currently reachable and asks if you want to rejoin any discovered instances to the rebooted cluster.
- Similarly, in interactive mode the wizard also detects instances which are currently not reachable and asks if you would like to remove such instances from the rebooted cluster.

If you are not using MySQL Shell's interactive mode, you can use the `rejoinInstances` and `removeInstances` options to manually configure instances which should be joined or removed during the reboot of the cluster.

If you encounter an error such as `The active session instance isn't the most updated in comparison with the ONLINE instances of the Cluster's metadata.` then the instance you are connected to does not have the GTID superset of transactions applied by the cluster. In this situation, connect MySQL Shell to the instance suggested in the error message and issue `dba.rebootClusterFromCompleteOutage()` from that instance.

Tip

To manually detect which instance has the GTID superset rather than using the interactive wizard, check the `gtid_executed` variable on each instance. For example issue:

```
mysql-sql> SHOW VARIABLES LIKE 'gtid_executed';
```

The instance which has applied the largest GTID set of transactions contains the GTID superset.

If this process fails, and the cluster metadata has become badly corrupted, you might need to drop the metadata and create the cluster again from scratch. You can drop the cluster metadata using `dba.dropMetadataSchema()`.

Warning

The `dba.dropMetadataSchema()` method should only be used as a last resort, when it is not possible to restore the cluster. It cannot be undone.

Rescanning a Cluster

If you make configuration changes to a cluster outside of the AdminAPI commands, for example by changing an instance's configuration manually to resolve configuration issues or after the loss of an instance, you need to update the InnoDB cluster metadata so that it matches the current configuration of instances. In these cases, use the *`Cluster`*`.rescan()` operation, which enables you to update the InnoDB cluster metadata either manually or using an interactive wizard. The *`Cluster`*`.rescan()` operation can detect new active instances that are not registered in the metadata and add them, or obsolete instances (no longer active) still registered in the metadata, and remove them. You can automatically update the metadata depending on the instances found by the command, or you can specify a list of instance addresses to either add to the metadata or remove from the metadata. You can also update the topology mode stored in the metadata, for example after changing from single-primary mode to multi-primary mode outside of AdminAPI.

The syntax of the command is *`Cluster`*`.rescan([options])`. The `options` dictionary supports the following:

- `interactive`: boolean value used to disable or enable the wizards in the command execution. Controls whether prompts and confirmations are provided. The default value is equal to MySQL Shell wizard mode, specified by `shell.options.useWizards`.
- `addInstances`: list with the connection data of the new active instances to add to the metadata, or “auto” to automatically add missing instances to the metadata. The value “auto” is case-insensitive.
  - Instances specified in the list are added to the metadata, without prompting for confirmation
  - In interactive mode, you are prompted to confirm the addition of newly discovered instances that are not included in the `addInstances` option
  - In non-interactive mode, newly discovered instances that are not included in the `addInstances` option are reported in the output, but you are not prompted to add them
- `removeInstances`: list with the connection data of the obsolete instances to remove from the metadata, or “auto” to automatically remove obsolete instances from the metadata.
  - Instances specified in the list are removed from the metadata, without prompting for confirmation
  - In interactive mode, you are prompted to confirm the removal of obsolete instances that are not included in the `removeInstances` option
  - In non-interactive mode, obsolete instances that are not included in the `removeInstances` option are reported in the output but you are not prompted to remove them

- `updateTopologyMode`: boolean value used to indicate if the topology mode (single-primary or multi-primary) in the metadata should be updated (true) or not (false) to match the one being used by the cluster. By default, the metadata is not updated (false).
  - If the value is `true` then the InnoDB cluster metadata is compared to the current mode being used by Group Replication, and the metadata is updated if necessary. Use this option to update the metadata after making changes to the topology mode of your cluster outside of AdminAPI.
  - If the value is `false` then InnoDB cluster metadata about the cluster's topology mode is not updated even if it is different from the topology used by the cluster's Group Replication group
  - If the option is not specified and the topology mode in the metadata is different from the topology used by the cluster's Group Replication group, then:
    - In interactive mode, you are prompted to confirm the update of the topology mode in the metadata
    - In non-interactive mode, if there is a difference between the topology used by the cluster's Group Replication group and the InnoDB cluster metadata, it is reported and no changes are made to the metadata
  - When the metadata topology mode is updated to match the Group Replication mode, the auto-increment settings on all instances are updated as described at InnoDB cluster and Auto-increment.

Checking Instance State

The `cluster.checkInstanceState()` function can be used to verify the existing data on an instance does not prevent it from joining a cluster. This process works by validating the instance's global transaction identifier (GTID) state compared to the GTIDs already processed by the cluster. For more information on GTIDs see Section 17.1.3.1, “GTID Format and Storage”. This check enables you to determine if an instance which has processed transactions can be added to the cluster.

The following demonstrates issuing this in a running MySQL Shell:

```
mysql-js> cluster.checkInstanceState('ic@ic-4:3306')
```

The output of this function can be one of the following:

- OK new: the instance has not executed any GTID transactions, therefore it cannot conflict with the GTIDs executed by the cluster
- OK recoverable: the instance has executed GTIDs which do not conflict with the executed GTIDs of the cluster seed instances
- ERROR diverged: the instance has executed GTIDs which diverge with the executed GTIDs of the cluster seed instances
- ERROR lost_transactions: the instance has more executed GTIDs than the executed GTIDs of the cluster seed instances

Instances with an OK status can be added to the cluster because any data on the instance is consistent with the cluster. In other words the instance being checked has not executed any transactions which conflict with the GTIDs executed by the cluster, and can be recovered to the same state as the rest of the cluster instances.

Dissolving an InnoDB Cluster

To dissolve an InnoDB cluster you connect to a read-write instance, for example the primary in a single-primary cluster, and use the *Cluster*.dissolve() command. This removes all metadata and configuration associated with the cluster, and disables Group Replication on the instances. Any data that was replicated between the instances is not removed.

Important

There is no way to undo the dissolving of a cluster. To create it again use dba.createCluster().

The *Cluster*.dissolve() operation can only configure instances which are ONLINE or reachable. If members of a cluster cannot be reached by the member where you issued the *Cluster*.dissolve() command you have to decide how the dissolve operation should proceed. If there is any chance you want to rejoin any instances that are identified as missing from the cluster, it is strongly recommended to cancel the dissolve operation and first bring the missing instances back online, before proceeding with a dissolve operation. This ensures that all instances can have their metadata updated correctly, and that there is no chance of a split-brain situation. However, if the instances from the cluster which cannot be reached have permanently left the cluster there could be no choice but to force the dissolve operation, which means that the missing instances are ignored and only online instances are affected by the operation.

Warning

Forcing the dissolve operation to ignore cluster instances can result in instances which could not be reached during the dissolve operation continuing to operate, creating the risk of a split-brain situation. Only ever force a dissolve operation to ignore missing instances if you are sure there is no chance of the instance coming online again.

In interactive mode, if members of a cluster are not reachable during a dissolve operation then an interactive prompt is displayed, for example:

```
mysql-js> Cluster.dissolve()
The cluster still has the following registered ReplicaSets:
{
    "clusterName": "testCluster",
    "defaultReplicaSet": {
        "name": "default",
        "topology": [
            {
                "address": "ic-1:3306",
                "label": "ic-1:3306",
                "role": "HA"
            },
            {
                "address": "ic-2:3306",
                "label": "ic-2:3306",
                "role": "HA"
            },
            {
                "address": "ic-3:3306",
                "label": "ic-3:3306",
                "role": "HA"
            }
        ]
    }
}
WARNING: You are about to dissolve the whole cluster and lose the high
availability features provided by it. This operation cannot be reverted. All
members will be removed from their ReplicaSet and replication will be stopped,
```

```
internal recovery user accounts and the cluster metadata will be dropped. User
data will be maintained intact in all instances.

Are you sure you want to dissolve the cluster? [y/N]: y

ERROR: The instance 'ic-2:3306' cannot be removed because it is on a '(MISSING)'
state. Please bring the instance back ONLINE and try to dissolve the cluster
again. If the instance is permanently not reachable, then you can choose to
proceed with the operation and only remove the instance from the Cluster
Metadata.

Do you want to continue anyway (only the instance metadata will be removed)?
[y/N]: y

Instance 'ic-3:3306' is attempting to leave the cluster...  Instance 'ic-1:3306'
is attempting to leave the cluster...

WARNING: The cluster was successfully dissolved, but the following instance was
skipped: 'ic-2:3306'. Please make sure this instance is permanently unavailable
or take any necessary manual action to ensure the cluster is fully dissolved.
```

In this example, the cluster consisted of three instances, one of which was offline when dissolve was issued. The error is caught, and you are given the choice how to proceed. In this case the missing `ic-2` instance is ignored and the reachable members have their metadata updated.

When MySQL Shell is running in non-interactive mode, for example when running a batch file, you can configure the behavior of the *`Cluster`*`.dissolve()` operation using the `force` option. To force the dissolve operation to ignore any instances which are unreachable, issue:

```
mysql-js> Cluster.dissolve({force: true})
```

Any instances which can be reached are removed from the cluster, and any unreachable instances are ignored. The warnings in this section about forcing the removal of missing instances from a cluster apply equally to this technique of forcing the dissolve operation.

You can also use the `interactive` option with the *`Cluster`*`.dissolve()` operation to override the mode which MySQL Shell is running in, for example to make the interactive prompt appear when running a batch script. For example:

```
mysql-js> Cluster.dissolve({interactive: true})
```

The `dba.gtidWaitTimeout` MySQL Shell option configures how long the *`Cluster`*`.dissolve()` operation waits for cluster transactions to be applied before removing a target instance from the cluster, but only if the target instance is `ONLINE`. An error is issued if the timeout is reached when waiting for cluster transactions to be applied on any of the instances being removed, except if force: true is used, which skips the error in that case.

Note

After issuing `cluster.dissolve()`, any variable assigned to the *`Cluster`* object is no longer valid.

Securing your Cluster

Server instances can be configured to use secure connections. For general information on using SSL with MySQL see Section 6.3, “Using Encrypted Connections”. This section explains how to configure a cluster to use SSL. An additional security possibility is to configure which servers can access the cluster, see Creating a Whitelist of Servers.

Important

Once you have configured a cluster to use SSL you must add the servers to the `ipWhitelist`.

When using `dba.createCluster()` to set up a cluster, if the server instance provides SSL encryption then it is automatically enabled on the seed instance. Pass the `memberSslMode` option to the `dba.createCluster()` method to specify a different SSL mode. The SSL mode of a cluster can only be set at the time of creation. The `memberSslMode` option is a string that configures the SSL mode to be used, it defaults to `AUTO`. The permitted values are `DISABLED`, `REQUIRED`, and `AUTO`. These modes are defined as:

- Setting `createCluster({memberSslMode:'DISABLED'})` ensures SSL encryption is disabled for the seed instance in the cluster.
- Setting `createCluster({memberSslMode:'REQUIRED'})` then SSL encryption is enabled for the seed instance in the cluster. If it cannot be enabled an error is raised.
- Setting `createCluster({memberSslMode:'AUTO'})` (the default) then SSL encryption is automatically enabled if the server instance supports it, or disabled if the server does not support it.

Note

When using the commercial version of MySQL, SSL is enabled by default and you might need to configure the whitelist for all instances. See Creating a Whitelist of Servers.

When you issue the `cluster.addInstance()` and `cluster.rejoinInstance()` commands, SSL encryption on the instance is enabled or disabled based on the setting found for the seed instance.

When using `createCluster()` with the `adoptFromGR` option to adopt an existing Group Replication group, no SSL settings are changed on the adopted cluster:

- `memberSslMode` cannot be used with `adoptFromGR`.
- If the SSL settings of the adopted cluster are different from the ones supported by the MySQL Shell, in other words SSL for Group Replication recovery and Group Communication, both settings are not modified. This means you are not be able to add new instances to the cluster, unless you change the settings manually for the adopted cluster.

MySQL Shell always enables or disables SSL for the cluster for both Group Replication recovery and Group Communication, see Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”. A verification is performed and an error issued in case those settings are different for the seed instance (for example as the result of a `dba.createCluster()` using `adoptFromGR`) when adding a new instance to the cluster. SSL encryption must be enabled or disabled for all instances in the cluster. Verifications are performed to ensure that this invariant holds when adding a new instance to the cluster.

The `deploySandboxInstance()` command attempts to deploy sandbox instances with SSL encryption support by default. If it is not possible, the server instance is deployed without SSL support. Use the `ignoreSslError` option set to false to ensure that sandbox instances are deployed with SSL support, issuing an error if SSL support cannot be provided. When `ignoreSslError` is true, which is the default, no error is issued during the operation if the SSL support cannot be provided and the server instance is deployed without SSL support.

Creating a Whitelist of Servers

When using a cluster's `createCluster()`, `addInstance()`, and `rejoinInstance()` methods you can optionally specify a list of approved servers that belong to the cluster, referred to as a whitelist.

By specifying the whitelist explicitly in this way you can increase the security of your cluster because only servers in the whitelist can connect to the cluster. Using the `ipWhitelist` option configures the `group_replication_ip_whitelist` system variable on the instance. By default, if not specified explicitly, the whitelist is automatically set to the private network addresses that the server has network interfaces on. To configure the whitelist, specify the servers to add with the `ipWhitelist` option when using the method. IP addresses must be specified in IPv4 format. Pass the servers as a comma separated list, surrounded by quotes. For example:

```
mysql-js> cluster.addInstance("ic@ic-3:3306", {ipWhitelist: "203.0.113.0/24, 198.51.100.110"})
```

This configures the instance to only accept connections from servers at addresses `203.0.113.0/24` and `198.51.100.110`. The whitelist can also include host names, which are resolved only when a connection request is made by another server.

Warning

Host names are inherently less secure than IP addresses in a whitelist. MySQL carries out FCrDNS verification, which provides a good level of protection, but can be compromised by certain types of attack. Specify host names in your whitelist only when strictly necessary, and ensure that all components used for name resolution, such as DNS servers, are maintained under your control. You can also implement name resolution locally using the hosts file, to avoid the use of external components.

Scripting AdminAPI

You can automate cluster configuration with scripts, which can be run using MySQL Shell. For example:

```
shell> mysqlsh -f setup-innodb-cluster.js
```

Note

Any command line options specified after the script file name are passed to the script and *not* to MySQL Shell. You can access those options using the `os.argv` array in JavaScript, or the `sys.argv` array in Python. In both cases, the first option picked up in the array is the script name.

The contents of an example script file is shown here:

```
print('InnoDB cluster sandbox set up\n');
print('================================\n');
print('Setting up a MySQL InnoDB cluster with 3 MySQL Server sandbox instances.\n');
print('The instances will be installed in ~/mysql-sandboxes.\n');
print('They will run on ports 3310, 3320 and 3330.\n\n');

var dbPass = shell.prompt('Please enter a password for the MySQL root account: ', {type:"password"});

try {
   print('\nDeploying the sandbox instances.');
   dba.deploySandboxInstance(3310, {password: dbPass});
   print('.');
   dba.deploySandboxInstance(3320, {password: dbPass});
   print('.');
   dba.deploySandboxInstance(3330, {password: dbPass});
   print('.\nSandbox instances deployed successfully.\n\n');

   print('Setting up InnoDB cluster...\n');
```

```
   shell.connect('root@localhost:3310', dbPass);

   var cluster = dba.createCluster("prodCluster");

   print('Adding instances to the cluster.');
   cluster.addInstance({user: "root", host: "localhost", port: 3320, password: dbPass});
   print('.');
   cluster.addInstance({user: "root", host: "localhost", port: 3330, password: dbPass});
   print('.\nInstances successfully added to the cluster.');

   print('\nInnoDB cluster deployed successfully.\n');
} catch(e) {
   print('\nThe InnoDB cluster could not be created.\n\nError: ' +
   + e.message + '\n');
}
```

Configuring the Election Process

You can optionally configure how a single-primary cluster elects a new primary, for example to prefer one instance as the new primary to fail over to. Use the `memberWeight` option and pass it to the `dba.createCluster()` and `Cluster.addInstance()` methods when creating your cluster. The `memberWeight` option accepts an integer value between 0 and 100, which is a percentage weight for automatic primary election on failover. When an instance has a higher precentage number set by `memberWeight`, it is more likely to be elected as primary in a single-primary cluster. When a primary election takes place, if multiple instances have the same `memberWeight` value, the instances are then prioritized based on their server UUID in lexicographical order (the lowest) and by picking the first one.

Setting the value of `memberWeight` configures the `group_replication_member_weight` system variable on the instance. Group Replication limits the value range from 0 to 100, automatically adjusting it if a higher or lower value is provided. Group Replication uses a default value of 50 if no value is provided. See Section 18.1.3.1, “Single-Primary Mode” for more information.

For example to configure a cluster where `ic-3` is the preferred instance to fail over to in the event that `ic-1`, the current primary, leaves the cluster unexpectedly use `memberWeight` as follows:

```
dba.createCluster('cluster1', {memberWeight:35})
var mycluster = dba.getCluster()
mycluster.addInstance('ic@ic2', {memberWeight:25})
mycluster.addInstance('ic@ic3', {memberWeight:50})
```

Configuring Failover Consistency

Group Replication provides the ability to specify the failover guarantees (eventual or “read your writes”) if a primary failover happens in single-primary mode (see Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”). You can configure the failover guarantees of an InnoDB cluster at creation by passing the `consistency` option (prior to version 8.0.16 this option was the `failoverConsistency` option, which is now deprecated) to the `dba.createCluster()` operation, which configures the `group_replication_consistency` system variable on the seed instance. This option defines the behavior of a new fencing mechanism used when a new primary is elected in a single-primary group. The fencing restricts connections from writing and reading from the new primary until it has applied any pending backlog of changes that came from the old primary (sometimes referred to as “read your writes”). While the fencing mechanism is in place, applications effectively do not see time going backward for a short period of time while any backlog is applied. This ensures that applications do not read stale information from the newly elected primary.

The `consistency` option is only supported if the target MySQL server version is 8.0.14 or later, and instances added to a cluster which has been configured with the `consistency` option are automatically

configured to have `group_replication_consistency` the same on all cluster members that have support for the option. The variable default value is controlled by Group Replication and is `EVENTUAL`, change the `consistency` option to `BEFORE_ON_PRIMARY_FAILOVER` to enable the fencing mechanism. Alternatively use `consistency=0` for `EVENTUAL` and `consistency=1` for `BEFORE_ON_PRIMARY_FAILOVER`.

Note

Using the `consistency` option on a multi-primary InnoDB cluster has no effect but is allowed because the cluster can later be changed into single-primary mode with the *`Cluster`*`.switchToSinglePrimaryMode()` operation.

Changing a Cluster's Topology

By default, an InnoDB cluster runs in single-primary mode, where the cluster has one primary server that accepts read and write queries (R/W), and all of the remaining instances in the cluster accept only read queries (R/O). When you configure a cluster to run in multi-primary mode, all of the instances in the cluster are primaries, which means that they accept both read and write queries (R/W). If a cluster has all of its instances running MySQL server version 8.0.15 or later, you can make changes to the topology of the cluster while the cluster is online. In previous versions it was necessary to completely dissolve and re-create the cluster to make the configuration changes. This uses the group action coordinator exposed through the UDFs described at Section 18.4.1, “Configuring an Online Group”, and as such you should observe the rules for configuring online groups.

Note

multi-primary mode is considered an advanced mode

Usually a single-primary cluster elects a new primary when the current primary leaves the cluster unexpectedly, for example due to an unexpected halt. The election process is normally used to choose which of the current secondaries becomes the new primary. To override the election process and force a specific server to become the new primary, use the *`Cluster`*`.setPrimaryInstance(`*`instance`*`)` function, where *`instance`* specifies the connection to the instance which should become the new primary. This enables you to configure the underlying Group Replication group to choose a specific instance as the new primary, bypassing the election process.

You can change the mode (sometimes described as the topology) which a cluster is running in between single-primary and multi-primary using the following operations:

- *`Cluster`*`.switchToMultiPrimaryMode()`, which switches the cluster to multi-primary mode. All instances become primaries.
- *`Cluster`*`.switchToSinglePrimaryMode([`*`instance`*`])`, which switches the cluster to single-primary mode. If *`instance`* is specified, it becomes the primary and all the other instances become secondaries. If *`instance`* is not specified, the new primary is the instance with the highest member weight (and the lowest UUID in case of a tie on member weight).

Setting Options for InnoDB cluster

You can check and modify the settings in place for an InnoDB cluster while the instances are online. To check the current settings of a cluster, use the following operation:

- *`Cluster`*`.options()`, which lists the cluster configuration options for its ReplicaSets and instances. A boolean option `all` can also be specified to include information about all Group Replication system variables in the output.

You can configure the options of an InnoDB cluster at a cluster level or instance level, while instances remain online. This avoids the need to remove, reconfigure and then again add the instance to change InnoDB cluster options. Use the following operations:

- `Cluster.setOption(option, value)` to change the settings of all cluster instances globally or cluster global settings such as `clusterName`.
- `Cluster.setInstanceOption(instance, option, value)` to change the settings of individual cluster instances

The way which you use InnoDB cluster options with the operations listed depends on whether the option can be changed to be the same on all instances or not. These options are changeable at both the cluster (all instances) and per instance level:

- `exitStateAction`
- `memberWeight`

This option is changeable at the per instance level only:

- `label`

These options are changeable at the cluster level only:

- `consistency`
- `expelTimeout`
- `clusterName`

InnoDB cluster and Auto-increment

When you are using an instance as part of an InnoDB cluster, the `auto_increment_increment` and `auto_increment_offset` variables are configured to avoid the possibility of auto increment collisions for multi-primary clusters up to a size of 9 (the maximum supported size of a Group Replication group). The logic used to configure these variables can be summarized as:

- If the group is running in single-primary mode, then set `auto_increment_increment` to 1 and `auto_increment_offset` to 2.
- If the group is running in multi-primary mode, then when the cluster has 7 instances or less set `auto_increment_increment` to 7 and `auto_increment_offset` to 1 + `server_id` % 7. If a multi-primary cluster has 8 or more instances set `auto_increment_increment` to the number of instances and `auto_increment_offset` to 1 + `server_id` % the number of instances.

21.5 Known Limitations

This section describes the known limitations of InnoDB cluster. As InnoDB cluster uses Group Replication, you should also be aware of its limitations, see Section 18.9.2, “Group Replication Limitations”.

- If a session type is not specified when creating the global session, MySQL Shell provides automatic protocol detection which attempts to first create a NodeSession and if that fails it tries to create a ClassicSession. With an InnoDB cluster that consists of three server instances, where there is one read-write port and two read-only ports, this can cause MySQL Shell to only connect to one of the read-only instances. Therefore it is recommended to always specify the session type when creating the global session.

- When adding non-sandbox server instances (instances which you have configured manually rather than using `dba.deploySandboxInstance()`) to a cluster, MySQL Shell is not able to persist any configuration changes in the instance's configuration file. This leads to one or both of the following scenarios:

  1. The Group Replication configuration is not persisted in the instance's configuration file and upon restart the instance does not rejoin the cluster.

  2. The instance is not valid for cluster usage. Although the instance can be verified with `dba.checkInstanceConfiguration()`, and MySQL Shell makes the required configuration changes in order to make the instance ready for cluster usage, those changes are not persisted in the configuration file and so are lost once a restart happens.

  If only `a` happens, the instance does not rejoin the cluster after a restart.

  If `b` also happens, and you observe that the instance did not rejoin the cluster after a restart, you cannot use the recommended `dba.rebootClusterFromCompleteOutage()` in this situation to get the cluster back online. This is because the instance loses any configuration changes made by MySQL Shell, and because they were not persisted, the instance reverts to the previous state before being configured for the cluster. This causes Group Replication to stop responding, and eventually the command times out.

  To avoid this problem it is strongly recommended to use `dba.configureInstance()` before adding instances to a cluster in order to persist the configuration changes.

- The use of the `--defaults-extra-file` option to specify an option file is not supported by InnoDB cluster server instances. InnoDB cluster only supports a single option file on instances and no extra option files are supported. Therefore for any operation working with the instance's option file the main one should be specified. If you want to use multiple option files you have to configure the files manually and make sure they are updated correctly considering the precedence rules of the use of multiple option files and ensuring that the desired settings are not incorrectly overwritten by options in an extra unrecognized option file.

- Attempting to use instances with a host name that resolves to an IP address which does not match a real network interface fails with an error that `This instance reports its own address as` *`the hostname`*. This is not supported by the Group Replication communication layer. On Debian based instances this means instances cannot use addresses such as `user@localhost` because localhost resolves to a non-existent IP (such as 127.0.1.1). This impacts on using a sandbox deployment, which usually uses local instances on a single machine.

  A workaround is to configure the `report_host` system variable on each instance to use the actual IP address of your machine. Retrieve the IP of your machine and add `report_host=`*`IP of your machine`* to the `my.cnf` file of each instance. You need to ensure the instances are then restarted to make the change.

Chapter 22 MySQL NDB Cluster 8.0

Table of Contents

22.1 NDB Cluster Overview 3653
22.1.1 NDB Cluster Core Concepts 3654
22.1.2 NDB Cluster Nodes, Node Groups, Replicas, and Partitions 3657
22.1.3 NDB Cluster Hardware, Software, and Networking Requirements 3660
22.1.4 What is New in NDB Cluster 3661
22.1.5 Options, Variables, and Parameters Added, Deprecated or Removed in NDB 8.0 3670
22.1.6 MySQL Server Using InnoDB Compared with NDB Cluster 3671
22.1.7 Known Limitations of NDB Cluster 3674
22.2 NDB Cluster Installation 3685
22.2.1 The NDB Cluster Auto-Installer 3688
22.2.2 Installation of NDB Cluster on Linux 3710
22.2.3 Installing NDB Cluster on Windows 3719
22.2.4 Initial Configuration of NDB Cluster 3728
22.2.5 Initial Startup of NDB Cluster 3730
22.2.6 NDB Cluster Example with Tables and Data 3731
22.2.7 Safe Shutdown and Restart of NDB Cluster 3735
22.2.8 Upgrading and Downgrading NDB Cluster 3736
22.3 Configuration of NDB Cluster 3737
22.3.1 Quick Test Setup of NDB Cluster 3737
22.3.2 Overview of NDB Cluster Configuration Parameters, Options, and Variables 3740
22.3.3 NDB Cluster Configuration Files 3759
22.3.4 Using High-Speed Interconnects with NDB Cluster 3946
22.4 NDB Cluster Programs 3946
22.4.1 `ndbd` — The NDB Cluster Data Node Daemon 3947
22.4.2 `ndbinfo_select_all` — Select From ndbinfo Tables 3955
22.4.3 `ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded) 3956
22.4.4 `ndb_mgmd` — The NDB Cluster Management Server Daemon 3958
22.4.5 `ndb_mgm` — The NDB Cluster Management Client 3966
22.4.6 `ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables 3968
22.4.7 `ndb_config` — Extract NDB Cluster Configuration Information 3971
22.4.8 `ndb_delete_all` — Delete All Rows from an NDB Table 3980
22.4.9 `ndb_desc` — Describe NDB Tables 3981
22.4.10 `ndb_drop_index` — Drop Index from an NDB Table 3987
22.4.11 `ndb_drop_table` — Drop an NDB Table 3988
22.4.12 `ndb_error_reporter` — NDB Error-Reporting Utility 3989
22.4.13 `ndb_import` — Import CSV Data Into NDB 3990
22.4.14 `ndb_index_stat` — NDB Index Statistics Utility 4004
22.4.15 `ndb_move_data` — NDB Data Copy Utility 4010
22.4.16 `ndb_perror` — Obtain NDB Error Message Information 4013
22.4.17 `ndb_print_backup_file` — Print NDB Backup File Contents 4015
22.4.18 `ndb_print_file` — Print NDB Disk Data File Contents 4015
22.4.19 `ndb_print_frag_file` — Print NDB Fragment List File Contents 4016
22.4.20 `ndb_print_schema_file` — Print NDB Schema File Contents 4017
22.4.21 `ndb_print_sys_file` — Print NDB System File Contents 4017
22.4.22 `ndb_redo_log_reader` — Check and Print Content of Cluster Redo Log 4018
22.4.23 `ndb_restore` — Restore an NDB Cluster Backup 4021
22.4.24 `ndb_select_all` — Print Rows from an NDB Table 4044
22.4.25 `ndb_select_count` — Print Row Counts for NDB Tables 4047

22.4.26 ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster 4048
22.4.27 ndb_show_tables — Display List of NDB Tables 4052
22.4.28 ndb_size.pl — NDBCLUSTER Size Requirement Estimator 4053
22.4.29 ndb_top — View CPU usage information for NDB threads 4056
22.4.30 ndb_waiter — Wait for NDB Cluster to Reach a Given Status 4062
22.4.31 Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs 4065
22.5 Management of NDB Cluster 4070
22.5.1 Summary of NDB Cluster Start Phases 4071
22.5.2 Commands in the NDB Cluster Management Client 4073
22.5.3 Online Backup of NDB Cluster 4078
22.5.4 MySQL Server Usage for NDB Cluster 4083
22.5.5 Performing a Rolling Restart of an NDB Cluster 4085
22.5.6 Event Reports Generated in NDB Cluster 4087
22.5.7 NDB Cluster Log Messages 4098
22.5.8 NDB Cluster Single User Mode 4114
22.5.9 Quick Reference: NDB Cluster SQL Statements 4114
22.5.10 ndbinfo: The NDB Cluster Information Database 4117
22.5.11 INFORMATION_SCHEMA Tables for NDB Cluster 4168
22.5.12 NDB Cluster Security Issues 4169
22.5.13 NDB Cluster Disk Data Tables 4176
22.5.14 Online Operations with ALTER TABLE in NDB Cluster 4184
22.5.15 Adding NDB Cluster Data Nodes Online 4187
22.5.16 Distributed MySQL Privileges with NDB_STORED_USER 4199
22.5.17 NDB API Statistics Counters and Variables 4200
22.6 NDB Cluster Replication 4211
22.6.1 NDB Cluster Replication: Abbreviations and Symbols 4212
22.6.2 General Requirements for NDB Cluster Replication 4213
22.6.3 Known Issues in NDB Cluster Replication 4214
22.6.4 NDB Cluster Replication Schema and Tables 4221
22.6.5 Preparing the NDB Cluster for Replication 4224
22.6.6 Starting NDB Cluster Replication (Single Replication Channel) 4226
22.6.7 Using Two Replication Channels for NDB Cluster Replication 4228
22.6.8 Implementing Failover with NDB Cluster Replication 4229
22.6.9 NDB Cluster Backups With NDB Cluster Replication 4230
22.6.10 NDB Cluster Replication: Multi-Master and Circular Replication 4237
22.6.11 NDB Cluster Replication Conflict Resolution 4241
22.7 NDB Cluster Release Notes 4255

MySQL *NDB Cluster* is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. Recent NDB Cluster release series use version 8 of the NDB storage engine (also known as NDBCLUSTER) to enable running several computers with MySQL servers and other software in a cluster. NDB Cluster 8.0, now available as a Release Candidate (RC) beginning with version 8.0.17, incorporates version 8.0 of the NDB storage engine. NDB Cluster 7.6, is the current GA release, and uses version 7.6 of NDB. Previous GA releases still available for use in production, NDB Cluster 7.5 and NDB Cluster 7.4, incorporate NDB versions 7.5 and 7.4, respectively. NDB Cluster 7.2, which uses version 7.2 of the NDB storage engine, is a previous GA release that is currently still maintained; 7.2 users are encouraged to upgrade to NDB 7.5 or NDB 7.6. *NDB 7.1 and previous release series are no longer supported or maintained*.

Support for the NDB storage engine is not included in standard MySQL Server 8.0 binaries built by Oracle. Instead, users of NDB Cluster binaries from Oracle should upgrade to the most recent binary release of NDB Cluster for supported platforms—these include RPMs that should work with most Linux distributions. NDB Cluster 8.0 users who build from source should use the sources provided for MySQL 8.0 and build

with the options required to provide NDB support. (Locations where the sources can be obtained are listed later in this section.)

Important

MySQL NDB Cluster does not support InnoDB cluster, which must be deployed using MySQL Server 8.0 with the `InnoDB` storage engine as well as additional applications that are not included in the NDB Cluster distribution. MySQL Server 8.0 binaries cannot be used with MySQL NDB Cluster. For more information about deploying and using InnoDB cluster, see Chapter 21, *InnoDB Cluster*. Section 22.1.6, "MySQL Server Using InnoDB Compared with NDB Cluster", discusses differences between the `NDB` and `InnoDB` storage engines.

This chapter contains information about NDB Cluster 8.0 releases through 8.0.19-ndb-8.0.19, currently available as a Release Candidate. NDB Cluster 7.6 is the latest General Availability release, and is recommended for new deployments; for information about NDB Cluster 7.6, see What is New in NDB Cluster 7.6. For similar information about NDB Cluster 7.5, see What is New in NDB Cluster 7.5. NDB Cluster 7.4 and 7.3 are previous GA releases still supported in production; see MySQL NDB Cluster 7.3 and NDB Cluster 7.4. NDB Cluster 7.2 is a previous GA release series which is still maintained, although we recommend that new deployments for production use NDB Cluster 7.6. For more information about NDB Cluster 7.2, see MySQL NDB Cluster 7.2.

Supported Platforms. NDB Cluster is currently available and supported on a number of platforms. For exact levels of support available for on specific combinations of operating system versions, operating system distributions, and hardware platforms, please refer to https://www.mysql.com/support/supportedplatforms/cluster.html.

Availability. NDB Cluster binary and source packages are available for supported platforms from https://dev.mysql.com/downloads/cluster/.

NDB Cluster release numbers. NDB 8.0 follows the same release pattern as the MySQL Server 8.0 series of releases, beginning with MySQL 8.0.13 and MySQL NDB Cluster 8.0.13. In this *Manual* and other MySQL documentation, we identify these and later NDB Cluster releases employing a version number that begins with "NDB". This version number is that of the `NDBCLUSTER` storage engine used in the NDB 8.0 release, and is the same as the MySQL 8.0 server version on which the NDB Cluster 8.0 release is based.

Version strings used in NDB Cluster software. The version string displayed by the `mysql` client supplied with the MySQL NDB Cluster distribution uses this format:

```
mysql-mysql_server_version-cluster
```

`mysql_server_version` represents the version of the MySQL Server on which the NDB Cluster release is based. For all NDB Cluster 8.0 releases, this is `8.0.n`, where *`n`* is the release number. Building from source using `-DWITH_NDBCLUSTER` or the equivalent adds the `-cluster` suffix to the version string. (See Section 22.2.2.4, "Building NDB Cluster from Source on Linux", and Section 22.2.3.2, "Compiling and Installing NDB Cluster from Source on Windows".) You can see this format used in the `mysql` client, as shown here:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 8.0.19-cluster Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT VERSION()\G
```

```
*************************** 1. row ***************************
VERSION(): 8.0.19-cluster
1 row in set (0.00 sec)
```

The first release of NDB Cluster using MySQL 8.0 was NDB 8.0.13, which used MySQL 8.0.13.

The version string displayed by other NDB Cluster programs not normally included with the MySQL 8.0 distribution uses this format:

```
mysql-mysql_server_version ndb-ndb_engine_version
```

`mysql_server_version` represents the version of the MySQL Server on which the NDB Cluster release is based. For all NDB Cluster 8.0 releases, this is `8.0.n`, where *`n`* is the release number. *`ndb_engine_version`* is the version of the `NDB` storage engine used by this release of the NDB Cluster software. For all NDB 8.0 releases, this number is the same as the MySQL Server version. You can see this format used in the output of the `SHOW` command in the `ndb_mgm` client, like this:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @10.0.10.6  (mysql-8.0.19 ndb-8.0.19-ndb-8.0.19, Nodegroup: 0, *)
id=2    @10.0.10.8  (mysql-8.0.19 ndb-8.0.19-ndb-8.0.19, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=3    @10.0.10.2  (mysql-8.0.19 ndb-8.0.19-ndb-8.0.19)

[mysqld(API)]   2 node(s)
id=4    @10.0.10.10  (mysql-8.0.19 ndb-8.0.19-ndb-8.0.19)
id=5 (not connected, accepting connect from any host)
```

Compatibility with standard MySQL 8.0 releases. While many standard MySQL schemas and applications can work using NDB Cluster, it is also true that unmodified applications and database schemas may be slightly incompatible or have suboptimal performance when run using NDB Cluster (see Section 22.1.7, “Known Limitations of NDB Cluster”). Most of these issues can be overcome, but this also means that you are very unlikely to be able to switch an existing application datastore—that currently uses, for example, `MyISAM` or `InnoDB`—to use the `NDB` storage engine without allowing for the possibility of changes in schemas, queries, and applications. A `mysqld` compiled without `NDB` support (that is, built without `-DWITH_NDBCLUSTER_STORAGE_ENGINE` or its alias `-DWITH_NDBCLUSTER`) cannot function as a drop-in replacement for a `mysqld` that is built with it.

NDB Cluster development source trees. NDB Cluster development trees can also be accessed from https://github.com/mysql/mysql-server.

The NDB Cluster development sources maintained at https://github.com/mysql/mysql-server are licensed under the GPL. For information about obtaining MySQL sources using Git and building them yourself, see Section 2.9.5, “Installing MySQL Using a Development Source Tree”.

Note

As with MySQL Server 8.0, NDB Cluster 8.0 releases are built using `CMake`.

NDB Cluster 7.5 and NDB Cluster 7.6 are available as General Availability (GA) releases; NDB 7.6 is recommended for new deployments. NDB Cluster 7.4 and NDB Cluster 7.3 are previous GA releases which are still supported in production. NDB 7.2 is a previous GA release series which is still maintained; it is no longer recommended for new deployments. For an overview of major features added in NDB 7.6, see What is New in NDB Cluster 7.6. For similar information about NDB Cluster 7.5, see What is New in NDB

Cluster 7.5. For information about previous NDB Cluster releases, see MySQL NDB Cluster 7.3 and NDB Cluster 7.4, and MySQL NDB Cluster 7.2.

The contents of this chapter are subject to revision as NDB Cluster continues to evolve. Additional information regarding NDB Cluster can be found on the MySQL website at http://www.mysql.com/products/cluster/.

Additional Resources. More information about NDB Cluster can be found in the following places:

- For answers to some commonly asked questions about NDB Cluster, see Section A.10, “MySQL 8.0 FAQ: NDB Cluster”.
- The NDB Cluster Forum: https://forums.mysql.com/list.php?25.
- Many NDB Cluster users and developers blog about their experiences with NDB Cluster, and make feeds of these available through PlanetMySQL.

22.1 NDB Cluster Overview

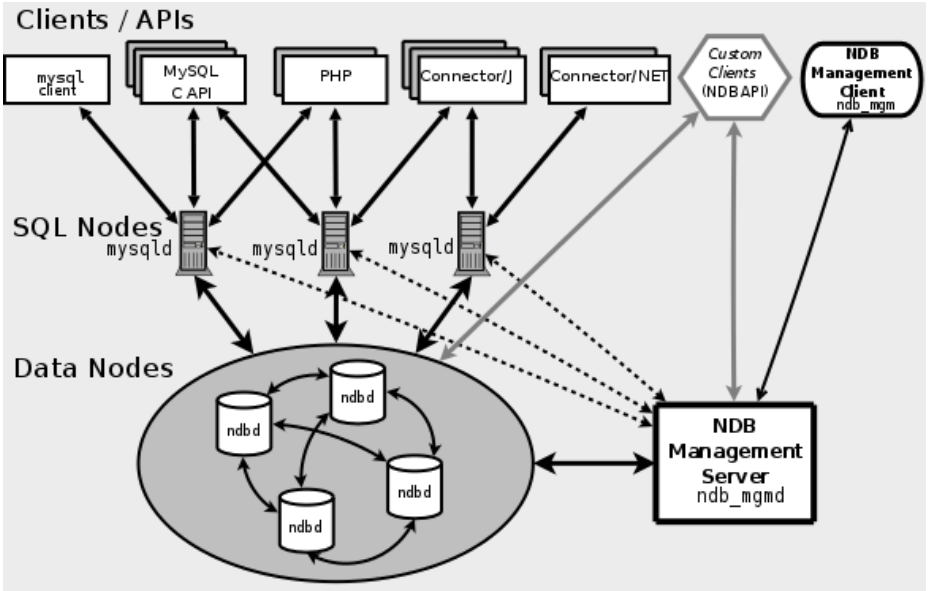
NDB Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. The shared-nothing architecture enables the system to work with very inexpensive hardware, and with a minimum of specific requirements for hardware or software.

NDB Cluster is designed not to have any single point of failure. In a shared-nothing system, each component is expected to have its own memory and disk, and the use of shared storage mechanisms such as network shares, network file systems, and SANs is not recommended or supported.

NDB Cluster integrates the standard MySQL server with an in-memory clustered storage engine called `NDB` (which stands for “*N*etwork *D*ata*B*ase”). In our documentation, the term `NDB` refers to the part of the setup that is specific to the storage engine, whereas “MySQL NDB Cluster” refers to the combination of one or more MySQL servers with the `NDB` storage engine.

An NDB Cluster consists of a set of computers, known as *hosts*, each running one or more processes. These processes, known as *nodes*, may include MySQL servers (for access to NDB data), data nodes (for storage of the data), one or more management servers, and possibly other specialized data access programs. The relationship of these components in an NDB Cluster is shown here:

Figure 22.1 NDB Cluster Components



All these programs work together to form an NDB Cluster (see Section 22.4, “NDB Cluster Programs”. When data is stored by the NDB storage engine, the tables (and table data) are stored in the data nodes. Such tables are directly accessible from all other MySQL servers (SQL nodes) in the cluster. Thus, in a payroll application storing data in a cluster, if one application updates the salary of an employee, all other MySQL servers that query this data can see this change immediately.

Although an NDB Cluster SQL node uses the mysqld server daemon, it differs in a number of critical respects from the mysqld binary supplied with the MySQL 8.0 distributions, and the two versions of mysqld are not interchangeable.

In addition, a MySQL server that is not connected to an NDB Cluster cannot use the NDB storage engine and cannot access any NDB Cluster data.

The data stored in the data nodes for NDB Cluster can be mirrored; the cluster can handle failures of individual data nodes with no other impact than that a small number of transactions are aborted due to losing the transaction state. Because transactional applications are expected to handle transaction failure, this should not be a source of problems.

Individual nodes can be stopped and restarted, and can then rejoin the system (cluster). Rolling restarts (in which all nodes are restarted in turn) are used in making configuration changes and software upgrades (see Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”). Rolling restarts are also used as part of the process of adding new data nodes online (see Section 22.5.15, “Adding NDB Cluster Data Nodes Online”). For more information about data nodes, how they are organized in an NDB Cluster, and how they handle and store NDB Cluster data, see Section 22.1.2, “NDB Cluster Nodes, Node Groups, Replicas, and Partitions”.

Backing up and restoring NDB Cluster databases can be done using the NDB-native functionality found in the NDB Cluster management client and the ndb_restore program included in the NDB Cluster distribution. For more information, see Section 22.5.3, “Online Backup of NDB Cluster”, and Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”. You can also use the standard MySQL functionality provided for this purpose in mysqldump and the MySQL server. See Section 4.5.4, “mysqldump — A Database Backup Program”, for more information.

NDB Cluster nodes can employ different transport mechanisms for inter-node communications; TCP/IP over standard 100 Mbps or faster Ethernet hardware is used in most real-world deployments.

22.1.1 NDB Cluster Core Concepts

NDBCLUSTER (also known as NDB) is an in-memory storage engine offering high-availability and data-persistence features.

The NDBCLUSTER storage engine can be configured with a range of failover and load-balancing options, but it is easiest to start with the storage engine at the cluster level. NDB Cluster's NDB storage engine contains a complete set of data, dependent only on other data within the cluster itself.

The “Cluster” portion of NDB Cluster is configured independently of the MySQL servers. In an NDB Cluster, each part of the cluster is considered to be a *node*.

Note

In many contexts, the term “node” is used to indicate a computer, but when discussing NDB Cluster it means a *process*. It is possible to run multiple nodes on a single computer; for a computer on which one or more cluster nodes are being run we use the term *cluster host*.

There are three types of cluster nodes, and in a minimal NDB Cluster configuration, there will be at least three nodes, one of each of these types:

- *Management node*: The role of this type of node is to manage the other nodes within the NDB Cluster, performing such functions as providing configuration data, starting and stopping nodes, and running backups. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node. An MGM node is started with the command `ndb_mgmd`.

- *Data node*: This type of node stores cluster data. There are as many data nodes as there are replicas, times the number of fragments (see Section 22.1.2, “NDB Cluster Nodes, Node Groups, Replicas, and Partitions”). For example, with two replicas, each having two fragments, you need four data nodes. One replica is sufficient for data storage, but provides no redundancy; therefore, it is recommended to have 2 (or more) replicas to provide redundancy, and thus high availability. A data node is started with the command `ndbd` (see Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”) or `ndbmtd` (see Section 22.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”).

  NDB Cluster tables are normally stored completely in memory rather than on disk (this is why we refer to NDB Cluster as an *in-memory* database). However, some NDB Cluster data can be stored on disk; see Section 22.5.13, “NDB Cluster Disk Data Tables”, for more information.

- *SQL node*: This is a node that accesses the cluster data. In the case of NDB Cluster, an SQL node is a traditional MySQL server that uses the `NDBCLUSTER` storage engine. An SQL node is a `mysqld` process started with the `--ndbcluster` and `--ndb-connectstring` options, which are explained elsewhere in this chapter, possibly with additional MySQL server options as well.

  An SQL node is actually just a specialized type of *API node*, which designates any application which accesses NDB Cluster data. Another example of an API node is the `ndb_restore` utility that is used to restore a cluster backup. It is possible to write such applications using the NDB API. For basic information about the NDB API, see Getting Started with the NDB API.

Important

It is not realistic to expect to employ a three-node setup in a production environment. Such a configuration provides no redundancy; to benefit from NDB Cluster's high-availability features, you must use multiple data and SQL nodes. The use of multiple management nodes is also highly recommended.

For a brief introduction to the relationships between nodes, node groups, replicas, and partitions in NDB Cluster, see Section 22.1.2, “NDB Cluster Nodes, Node Groups, Replicas, and Partitions”.

Configuration of a cluster involves configuring each individual node in the cluster and setting up individual communication links between nodes. NDB Cluster is currently designed with the intention that data nodes are homogeneous in terms of processor power, memory space, and bandwidth. In addition, to provide a single point of configuration, all configuration data for the cluster as a whole is located in one configuration file.

The management server manages the cluster configuration file and the cluster log. Each node in the cluster retrieves the configuration data from the management server, and so requires a way to determine where the management server resides. When interesting events occur in the data nodes, the nodes transfer information about these events to the management server, which then writes the information to the cluster log.

In addition, there can be any number of cluster client processes or applications. These include standard MySQL clients, `NDB`-specific API programs, and management clients. These are described in the next few paragraphs.

Standard MySQL clients. NDB Cluster can be used with existing MySQL applications written in PHP, Perl, C, C++, Java, Python, Ruby, and so on. Such client applications send SQL statements to and receive

responses from MySQL servers acting as NDB Cluster SQL nodes in much the same way that they interact with standalone MySQL servers.

MySQL clients using an NDB Cluster as a data source can be modified to take advantage of the ability to connect with multiple MySQL servers to achieve load balancing and failover. For example, Java clients using Connector/J 5.0.6 and later can use `jdbc:mysql:loadbalance://` URLs (improved in Connector/J 5.1.7) to achieve load balancing transparently; for more information about using Connector/J with NDB Cluster, see Using Connector/J with NDB Cluster.

NDB client programs. Client programs can be written that access NDB Cluster data directly from the `NDBCLUSTER` storage engine, bypassing any MySQL Servers that may be connected to the cluster, using the *NDB API*, a high-level C++ API. Such applications may be useful for specialized purposes where an SQL interface to the data is not needed. For more information, see The NDB API.

`NDB`-specific Java applications can also be written for NDB Cluster using the *NDB Cluster Connector for Java*. This NDB Cluster Connector includes *ClusterJ*, a high-level database API similar to object-relational mapping persistence frameworks such as Hibernate and JPA that connect directly to `NDBCLUSTER`, and so does not require access to a MySQL Server. Support is also provided in NDB Cluster for *ClusterJPA*, an OpenJPA implementation for NDB Cluster that leverages the strengths of ClusterJ and JDBC; ID lookups and other fast operations are performed using ClusterJ (bypassing the MySQL Server), while more complex queries that can benefit from MySQL's query optimizer are sent through the MySQL Server, using JDBC. See Java and NDB Cluster, and The ClusterJ API and Data Object Model, for more information.

NDB Cluster also supports applications written in JavaScript using Node.js. The MySQL Connector for JavaScript includes adapters for direct access to the `NDB` storage engine and as well as for the MySQL Server. Applications using this Connector are typically event-driven and use a domain object model similar in many ways to that employed by ClusterJ. For more information, see MySQL NoSQL Connector for JavaScript.

The Memcache API for NDB Cluster, implemented as the loadable *ndbmemcache* storage engine for memcached version 1.6 and later, can be used to provide a persistent NDB Cluster data store, accessed using the memcache protocol.

The standard `memcached` caching engine is included in the NDB Cluster 8.0 distribution. Each `memcached` server has direct access to data stored in NDB Cluster, but is also able to cache data locally and to serve (some) requests from this local cache.

For more information, see ndbmemcache—Memcache API for NDB Cluster.

Management clients. These clients connect to the management server and provide commands for starting and stopping nodes gracefully, starting and stopping message tracing (debug versions only), showing node versions and status, starting and stopping backups, and so on. An example of this type of program is the `ndb_mgm` management client supplied with NDB Cluster (see Section 22.4.5, “`ndb_mgm` — The NDB Cluster Management Client”). Such applications can be written using the *MGM API*, a C-language API that communicates directly with one or more NDB Cluster management servers. For more information, see The MGM API.

Oracle also makes available MySQL Cluster Manager, which provides an advanced command-line interface simplifying many complex NDB Cluster management tasks, such restarting an NDB Cluster with a large number of nodes. The MySQL Cluster Manager client also supports commands for getting and setting the values of most node configuration parameters as well as `mysqld` server options and variables relating to NDB Cluster. See MySQL™ Cluster Manager 1.4.7 User Manual, for more information.

Event logs. NDB Cluster logs events by category (startup, shutdown, errors, checkpoints, and so on), priority, and severity. A complete listing of all reportable events may be found in Section 22.5.6, “Event Reports Generated in NDB Cluster”. Event logs are of the two types listed here:

- *Cluster log*: Keeps a record of all desired reportable events for the cluster as a whole.
- *Node log*: A separate log which is also kept for each individual node.

Note

Under normal circumstances, it is necessary and sufficient to keep and examine only the cluster log. The node logs need be consulted only for application development and debugging purposes.

Checkpoint. Generally speaking, when data is saved to disk, it is said that a *checkpoint* has been reached. More specific to NDB Cluster, a checkpoint is a point in time where all committed transactions are stored on disk. With regard to the `NDB` storage engine, there are two types of checkpoints which work together to ensure that a consistent view of the cluster's data is maintained. These are shown in the following list:

- *Local Checkpoint (LCP)*: This is a checkpoint that is specific to a single node; however, LCPs take place for all nodes in the cluster more or less concurrently. An LCP usually occurs every few minutes; the precise interval varies, and depends upon the amount of data stored by the node, the level of cluster activity, and other factors.

  NDB 8.0 supports partial LCPs, which can significantly improve performance under some conditions. See the descriptions of the `EnablePartialLcp` and `RecoveryWork` configuration parameters which enable partial LCPs and control the amount of storage they use.

- *Global Checkpoint (GCP)*: A GCP occurs every few seconds, when transactions for all nodes are synchronized and the redo-log is flushed to disk.

For more information about the files and directories created by local checkpoints and global checkpoints, see NDB Cluster Data Node File System Directory Files.

22.1.2 NDB Cluster Nodes, Node Groups, Replicas, and Partitions

This section discusses the manner in which NDB Cluster divides and duplicates data for storage.

A number of concepts central to an understanding of this topic are discussed in the next few paragraphs.

Data node. An `ndbd` or `ndbmtd` process, which stores one or more *replicas*—that is, copies of the *partitions* (discussed later in this section) assigned to the node group of which the node is a member.

Each data node should be located on a separate computer. While it is also possible to host multiple data node processes on a single computer, such a configuration is not usually recommended.

It is common for the terms “node” and “data node” to be used interchangeably when referring to an `ndbd` or `ndbmtd` process; where mentioned, management nodes (`ndb_mgmd` processes) and SQL nodes (`mysqld` processes) are specified as such in this discussion.

Node group. A node group consists of one or more nodes, and stores partitions, or sets of *replicas* (see next item).

The number of node groups in an NDB Cluster is not directly configurable; it is a function of the number of data nodes and of the number of replicas (`NoOfReplicas` configuration parameter), as shown here:

```
[# of node groups] = [# of data nodes] / NoOfReplicas
```

Thus, an NDB Cluster with 4 data nodes has 4 node groups if `NoOfReplicas` is set to 1 in the `config.ini` file, 2 node groups if `NoOfReplicas` is set to 2, and 1 node group if `NoOfReplicas` is set to 4. Replicas are discussed later in this section; for more information about `NoOfReplicas`, see Section 22.3.3.6, "Defining NDB Cluster Data Nodes".

Note

All node groups in an NDB Cluster must have the same number of data nodes.

You can add new node groups (and thus new data nodes) online, to a running NDB Cluster; see Section 22.5.15, "Adding NDB Cluster Data Nodes Online", for more information.

Partition. This is a portion of the data stored by the cluster. Each node is responsible for keeping at least one copy of any partitions assigned to it (that is, at least one replica) available to the cluster.

The number of partitions used by default by NDB Cluster depends on the number of data nodes and the number of LDM threads in use by the data nodes, as shown here:

```
[# of partitions] = [# of data nodes] * [# of LDM threads]
```

When using data nodes running `ndbmtd`, the number of LDM threads is controlled by the setting for `MaxNoOfExecutionThreads`. When using `ndbd` there is a single LDM thread, which means that there are as many cluster partitions as nodes participating in the cluster. This is also the case when using `ndbmtd` with `MaxNoOfExecutionThreads` set to 3 or less. (You should be aware that the number of LDM threads increases with the value of this parameter, but not in a strictly linear fashion, and that there are additional constraints on setting it; see the description of `MaxNoOfExecutionThreads` for more information.)

NDB and user-defined partitioning. NDB Cluster normally partitions `NDBCLUSTER` tables automatically. However, it is also possible to employ user-defined partitioning with `NDBCLUSTER` tables. This is subject to the following limitations:

1. Only the `KEY` and `LINEAR KEY` partitioning schemes are supported in production with `NDB` tables.

2. The maximum number of partitions that may be defined explicitly for any `NDB` table is `8 * MaxNoOfExecutionThreads * [number of node groups]`, the number of node groups in an NDB Cluster being determined as discussed previously in this section. When using `ndbd` for data node processes, setting `MaxNoOfExecutionThreads` has no effect; in such a case, it can be treated as though it were equal to 1 for purposes of performing this calculation.

   See Section 22.4.3, "`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)", for more information.

For more information relating to NDB Cluster and user-defined partitioning, see Section 22.1.7, "Known Limitations of NDB Cluster", and Section 23.6.2, "Partitioning Limitations Relating to Storage Engines".

Replica. This is a copy of a cluster partition. Each node in a node group stores a replica. Also sometimes known as a *partition replica*. The number of replicas is equal to the number of nodes per node group.

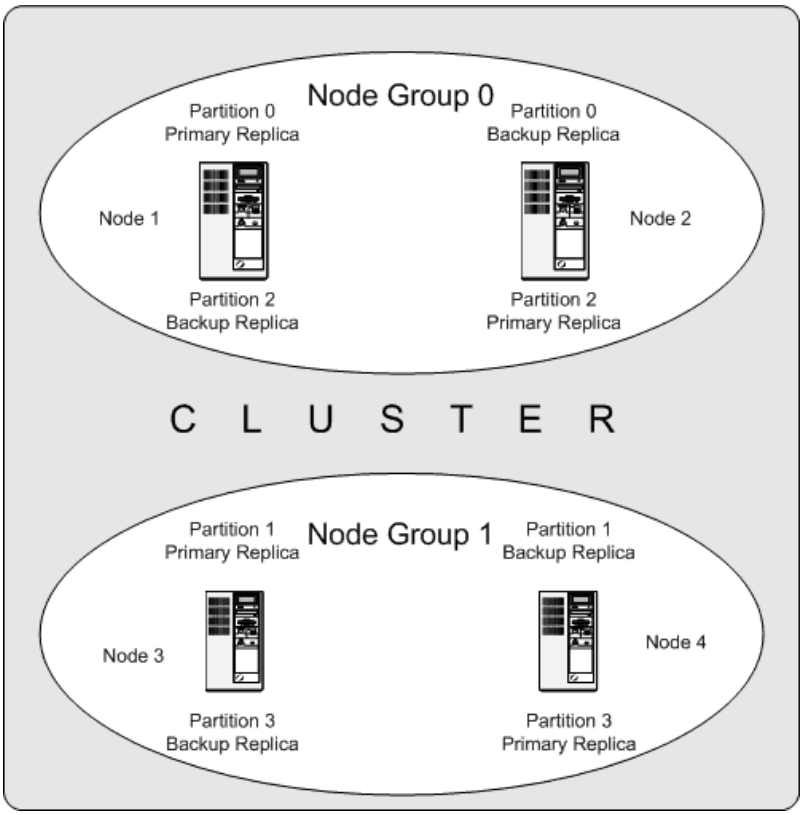
A replica belongs entirely to a single node; a node can (and usually does) store several replicas.

The following diagram illustrates an NDB Cluster with four data nodes running `ndbd`, arranged in two node groups of two nodes each; nodes 1 and 2 belong to node group 0, and nodes 3 and 4 belong to node group 1.

Note

Only data nodes are shown here; although a working NDB Cluster requires an `ndb_mgmd` process for cluster management and at least one SQL node to access the data stored by the cluster, these have been omitted from the figure for clarity.

Figure 22.2 NDB Cluster with Two Node Groups

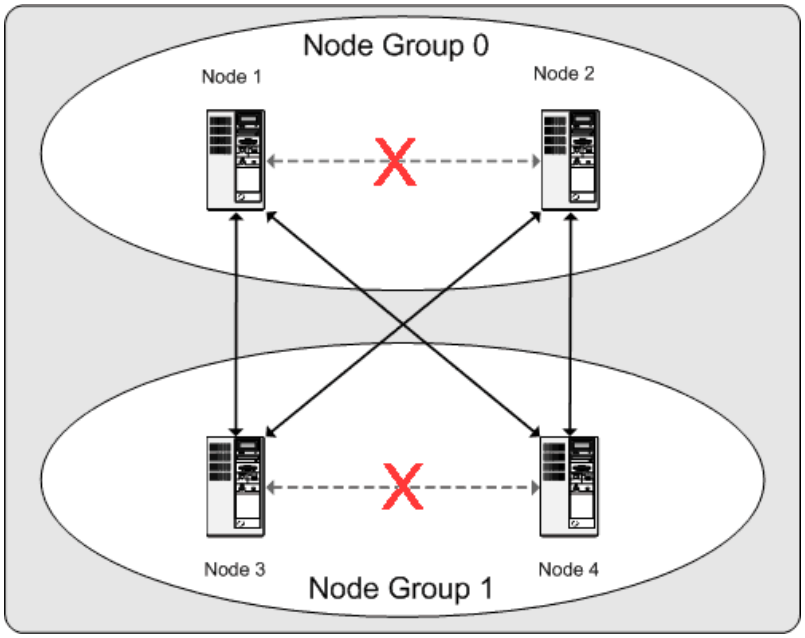


The data stored by the cluster is divided into four partitions, numbered 0, 1, 2, and 3. Each partition is stored—in multiple copies—on the same node group. Partitions are stored on alternate node groups as follows:

- Partition 0 is stored on node group 0; a *primary replica* (primary copy) is stored on node 1, and a *backup replica* (backup copy of the partition) is stored on node 2.
- Partition 1 is stored on the other node group (node group 1); this partition's primary replica is on node 3, and its backup replica is on node 4.
- Partition 2 is stored on node group 0. However, the placing of its two replicas is reversed from that of Partition 0; for Partition 2, the primary replica is stored on node 2, and the backup on node 1.
- Partition 3 is stored on node group 1, and the placement of its two replicas are reversed from those of partition 1. That is, its primary replica is located on node 4, with the backup on node 3.

What this means regarding the continued operation of an NDB Cluster is this: so long as each node group participating in the cluster has at least one node operating, the cluster has a complete copy of all data and remains viable. This is illustrated in the next diagram.

Figure 22.3 Nodes Required for a 2x2 NDB Cluster



In this example, the cluster consists of two node groups each consisting of two data nodes. Each data node is running an instance of ndbd. Any combination of at least one node from node group 0 and at least one node from node group 1 is sufficient to keep the cluster “alive”. However, if both nodes from a single node group fail, the combination consisting of the remaining two nodes in the other node group is not sufficient. In this situation, the cluster has lost an entire partition and so can no longer provide access to a complete set of all NDB Cluster data.

The maximum number of node groups supported for a single NDB Cluster instance is 48.

22.1.3 NDB Cluster Hardware, Software, and Networking Requirements

One of the strengths of NDB Cluster is that it can be run on commodity hardware and has no unusual requirements in this regard, other than for large amounts of RAM, due to the fact that all live data storage is done in memory. (It is possible to reduce this requirement using Disk Data tables—see Section 22.5.13, “NDB Cluster Disk Data Tables”, for more information about these.) Naturally, multiple and faster CPUs can enhance performance. Memory requirements for other NDB Cluster processes are relatively small.

The software requirements for NDB Cluster are also modest. Host operating systems do not require any unusual modules, services, applications, or configuration to support NDB Cluster. For supported operating systems, a standard installation should be sufficient. The MySQL software requirements are simple: all that is needed is a production release of NDB Cluster. It is not strictly necessary to compile MySQL yourself merely to be able to use NDB Cluster. We assume that you are using the binaries appropriate to your platform, available from the NDB Cluster software downloads page at https://dev.mysql.com/downloads/cluster/.

For communication between nodes, NDB Cluster supports TCP/IP networking in any standard topology, and the minimum expected for each host is a standard 100 Mbps Ethernet card, plus a switch, hub, or router to provide network connectivity for the cluster as a whole. We strongly recommend that an NDB Cluster be run on its own subnet which is not shared with machines not forming part of the cluster for the following reasons:

- **Security.** Communications between NDB Cluster nodes are not encrypted or shielded in any way. The only means of protecting transmissions within an NDB Cluster is to run your NDB Cluster on a

protected network. If you intend to use NDB Cluster for Web applications, the cluster should definitely reside behind your firewall and not in your network's De-Militarized Zone (DMZ) or elsewhere.

See Section 22.5.12.1, “NDB Cluster Security and Networking Issues”, for more information.

- **Efficiency.** Setting up an NDB Cluster on a private or protected network enables the cluster to make exclusive use of bandwidth between cluster hosts. Using a separate switch for your NDB Cluster not only helps protect against unauthorized access to NDB Cluster data, it also ensures that NDB Cluster nodes are shielded from interference caused by transmissions between other computers on the network. For enhanced reliability, you can use dual switches and dual cards to remove the network as a single point of failure; many device drivers support failover for such communication links.

Network communication and latency. NDB Cluster requires communication between data nodes and API nodes (including SQL nodes), as well as between data nodes and other data nodes, to execute queries and updates. Communication latency between these processes can directly affect the observed performance and latency of user queries. In addition, to maintain consistency and service despite the silent failure of nodes, NDB Cluster uses heartbeating and timeout mechanisms which treat an extended loss of communication from a node as node failure. This can lead to reduced redundancy. Recall that, to maintain data consistency, an NDB Cluster shuts down when the last node in a node group fails. Thus, to avoid increasing the risk of a forced shutdown, breaks in communication between nodes should be avoided wherever possible.

The failure of a data or API node results in the abort of all uncommitted transactions involving the failed node. Data node recovery requires synchronization of the failed node's data from a surviving data node, and re-establishment of disk-based redo and checkpoint logs, before the data node returns to service. This recovery can take some time, during which the Cluster operates with reduced redundancy.

Heartbeating relies on timely generation of heartbeat signals by all nodes. This may not be possible if the node is overloaded, has insufficient machine CPU due to sharing with other programs, or is experiencing delays due to swapping. If heartbeat generation is sufficiently delayed, other nodes treat the node that is slow to respond as failed.

This treatment of a slow node as a failed one may or may not be desirable in some circumstances, depending on the impact of the node's slowed operation on the rest of the cluster. When setting timeout values such as `HeartbeatIntervalDbDb` and `HeartbeatIntervalDbApi` for NDB Cluster, care must be taken care to achieve quick detection, failover, and return to service, while avoiding potentially expensive false positives.

Where communication latencies between data nodes are expected to be higher than would be expected in a LAN environment (on the order of 100 µs), timeout parameters must be increased to ensure that any allowed periods of latency periods are well within configured timeouts. Increasing timeouts in this way has a corresponding effect on the worst-case time to detect failure and therefore time to service recovery.

LAN environments can typically be configured with stable low latency, and such that they can provide redundancy with fast failover. Individual link failures can be recovered from with minimal and controlled latency visible at the TCP level (where NDB Cluster normally operates). WAN environments may offer a range of latencies, as well as redundancy with slower failover times. Individual link failures may require route changes to propagate before end-to-end connectivity is restored. At the TCP level this can appear as large latencies on individual channels. The worst-case observed TCP latency in these scenarios is related to the worst-case time for the IP layer to reroute around the failures.

22.1.4 What is New in NDB Cluster

The following sections describe changes in the implementation of NDB Cluster in MySQL NDB Cluster 8.0 through 8.0.19, as compared to earlier release series. NDB Cluster 8.0 is currently available as a Release Candidate. NDB Cluster 7.6 is available as a General Availability release, as is NDB Cluster 7.5

For information about additions and other changes in NDB Cluster 7.6, see What is New in NDB Cluster 7.5; for information about new features and other changes in NDB Cluster 7.5, see What is New in NDB Cluster 7.6.

NDB Cluster 7.4 and 7.3 are recent General Availability releases, and are still supported. NDB Cluster 7.2 is a previous GA release, still supported in production for existing deployments. *NDB 7.1 and earlier releases series are no longer maintained or supported in production*. We recommend that new deployments use NDB Cluster 7.6 or NDB Cluster 7.5. For information about NDB 7.4 and NDB 7.3, see MySQL NDB Cluster 7.3 and NDB Cluster 7.4. For information about NDB 7.2 and previous NDB releases, see MySQL NDB Cluster 7.2.

What is New in NDB Cluster 8.0

Major changes and new features in NDB Cluster 8.0 which are likely to be of interest are shown in the following list:

- **Compatibility enhancements.** The following changes reduce longstanding nonessential differences in `NDB` behavior as compared to that of other MySQL storage engines:

  - **Development in parallel with MySQL server.** Beginning with this release, MySQL NDB Cluster is being developed in parallel with the standard MySQL 8.0 server under a new unified release model with the following features:

    - NDB 8.0 is developed in, built from, and released with the MySQL 8.0 source code tree.
    - The numbering scheme for NDB Cluster 8.0 releases follows the scheme for MySQL 8.0, starting with version 8.0.13.
    - Building the source with NDB support appends `-cluster` to the version string returned by `mysql -V`, as shown here:

      ```
      shell≫ mysql -V
      mysql  Ver 8.0.19-cluster for Linux on x86_64 (Source distribution)
      ```

      NDB binaries continue to display both the MySQL Server version and the NDB engine version, like this:

      ```
      shell> ndb_mgm -V
      MySQL distrib mysql-8.0.19 ndb-8.0.19, for Linux (x86_64)
      ```

      In MySQL Cluster NDB 8.0, these two version numbers are always the same.

    To build the MySQL 8.0.13 (or later) source with NDB Cluster support, use the CMake option `-DWITH_NDBCLUSTER`.

  - **Database and table names.** As of NDB 8.0.18, the 63-byte limit on identifiers for databases and tables is removed. These identifiers can now use up to 64 bytes, as for such objects using other MySQL storage engines. See Section 22.1.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 8.0”.

  - **Generated names for foreign keys.** `NDB` (version 8.0.18 and later) now uses the pattern *`tbl_name`*`_fk_`*`N`* for naming internally generated foreign keys. This is similar to the pattern used by `InnoDB`.

- **Schema and metadata distribution and synchronization.** NDB 8.0 makes use of the MySQL data dictionary to distribute schema information to SQL nodes joining a cluster and to synchronize new

schema changes between existing SQL nodes. The following list describes individual enhancements relating to this integration work:

- **Schema distribution enhancements.** The `NDB` schema distribution coordinator, which handles schema operations and tracks their progress, has been extended in NDB 8.0.17 to ensure that resources used during a schema operation are released at its conclusion. Previously, some of this work was done by the schema distribution client; this has been changed due to the fact that the client did not always have all needed state information, which could lead to resource leaks when the client decided to abandon the schema operation prior to completion and without informing the coordinator.

  To help fix this issue, schema operation timeout detection has been moved from the schema distribution client to the coordinator, providing the coordinator with an opportunity to clean up any resources used during the schema operation. The coordinator now checks ongoing schema operations for timeout at regular intervals, and marks participants that have not yet completed a given schema operation as failed when detecting timeout. It also provides suitable warnings whenever a schema operation timeout occurs. (It should be noted that, after such a timeout is detected, the schema operation itself continues.) Additional reporting is done by printing a list of active schema operations at regular intervals whenever one or more of these operations is ongoing.

  As an additional part of this work, a new `mysqld` option `--ndb-schema-dist-timeout` makes it possible to set the length of time to wait until a schema operation is marked as having timed out.

- **Disk data file distribution.** Beginning with NDB Cluster 8.0.14, `NDB` uses the MySQL data dictionary to make sure that disk data files and related constructs such as tablespaces and log file groups are correctly distributed between all connected SQL nodes.

- **Schema distribution of tablespace objects.** When a MySQL Server connects as an SQL node to an NDB cluster, it checks its data dictionary against the information found in `NDB` dictionary.

  Previously, the only `NDB` objects synchronized on connection of a new SQL node were databases and tables; MySQL NDB Cluster 8.0.14 and later also implement schema synchronization of disk data objects including tablespaces and log file groups. Among other benefits, this eliminates the possibility of a mismatch between the MySQL data dictionary and the `NDB` dictionary following a native backup and restore, in which tablespaces and log file groups were restored to the `NDB` dictionary, but not to the MySQL Server's data dictionary.

  It is also no longer possible to issue a `CREATE TABLE` statement that refers to a nonexistent tablespace. Such a statement now fails with an error.

- **Database DDL synchronization enhancements.** Work done in NDB 8.0.17 insures that synchronization of databases by newly joined (or rejoined) SQL nodes with those on existing SQL nodes now makes proper use of the data dictionary so that any database-level operations (`CREATE DATABASE`, `ALTER DATABASE`, or `DROP DATABASE`) that may have been misssed by this SQL node are now correctly duplicated on it when it connects (or reconnects) to the cluster.

  As part of the schema synchronization procedure performed when starting, an SQL node now compares all databases on the cluster's data nodes with those in its own data dictionary, and if any of these is found to be missing from the SQL node's data dictionary, the SQL Node installs it locally by executing a `CREATE DATABASE` statement. A database thus created uses the default MySQL Server database properties (such as those as determined by `character_set_database` and `collation_database`) that are in effect in effect on this SQL node at the time the statement is executed.

- **NDB metadata change detection.** NDB 8.0.16 implements a new mechanism for detection of updates to metadata for data objects such as tables, tablespaces, and log file groups with the MySQL

data dictionary. This is done using a thread, the NDB metadata change monitor thread, which runs in the background and checks periodically for inconsistencies between the NDB dictionary and the MySQL data dictionary.

The monitor performs metadata checks every 60 seconds by default. The polling interval can be adjusted by setting the value of the ndb_metadata_check_interval system variable; polling can be disabled altogether by setting the ndb_metadata_check system variable to OFF. A status variable, Ndb_metadata_detected_count shows the number of times since mysqld was last started that inconsistencies have been detected.

- **Changes in NDB table extra metadata.** In NDB 8.0.14 and later, the extra metadata property of an NDB table is used for storing serialized metadata from the MySQL data dictionary, rather than storing the binary representation of the table as in previous versions. (This was a .frm file, no longer used by the MySQL Server—see Chapter 14, *MySQL Data Dictionary*.) As part of the work to support this change, the available size of the table's extra metadata has been increased. This means that NDB tables created in NDB Cluster 8.0.14 and later are not compatible with previous NDB Cluster releases. Tables created in previous releases can be used with NDB 8.0.14 and later, but cannot be opened afterwards by an earlier version.

  This metadata is accessible using the NDB API methods getExtraMetadata() and setExtraMetadata() that were implemented in NDB 8.0.13.

  For more information, see Section 22.2.8, “Upgrading and Downgrading NDB Cluster”.

- **On-the-fly upgrades of tables using .frm files.** A table created in NDB 7.6 and earlier contains metadata in the form of a compressed .frm file, which is no longer supported in MySQL 8.0. To facilitate online upgrades to NDB 8.0, NDB performs on-the-fly translation of this metadata and writes it into the MySQL Server's data dictionary, which enables the mysqld in NDB Cluster 8.0 to work with the table without preventing subsequent use of the table by a previous version of the NDB software.

  Important

  Once a table's structure has been modified in NDB 8.0, its metadata is stored using the data dictionary, and it can no longer be accessed by NDB 7.6 and earlier.

  This enhancement also makes it possible to restore an NDB backup made using an earlier version to a cluster running NDB 8.0 (or later).

- **Metadata change detection and automatic synchronization.** Beginning in version 8.0.18, NDB ensures that NDB table objects submitted by the metadata change monitor thread during operations following startup are automatically checked for mismatches and synchronized by the NDB binlog thread.

  NDB 8.0.18 also adds two status variables relating to automatic synchronization: Ndb_metadata_synced_count shows the number of objects synchronized automatically; Ndb_metadata_blacklist_size indicates the number of objects for which synchronization has failed. In addition, you can see which objects have been synchronized by inspecting the cluster log.

- **Synchronization of user privileges with NDB_STORED_USER.** A new mechanism for sharing and synchronizing users, roles, and privileges between SQL nodes is available beginning with NDB 8.0.18, using the NDB_STORED_USER privilege. Distributed privileges as implemented in NDB 7.6 and earlier (see Distributed Privileges Using Shared Grant Tables) are no longer supported.

  Once a user account is created on an SQL node, the user and its privileges can be stored in NDB and thus shared between all SQL nodes in the cluster by issuing a GRANT statement such as this one:

```
GRANT NDB_STORED_USER ON *.* TO 'jon'@'localhost';
```

`NDB_STORED_USER` always has global scope and must be granted using `ON *.*`. System reserved accounts such as `mysql.session@localhost` or `mysql.infoschema@localhost` cannot be assigned this privilege.

Roles can also be shared between SQL nodes by issuing the appropriate `GRANT NDB_STORED_USER` statement. Assigning such a role to a user does not cause the user to be shared; the `NDB_STORED_USER` privilege must be granted to each user explicitly.

A user or role having `NDB_STORED_USER`, along with its privileges, is shared with all SQL nodes as soon as they join a given NDB Cluster. Changes to the privileges of the user or role are synchronized immediately with all connected SQL nodes. It is possible to make such changes from any connected SQL node, but recommended practice is to do so from a designated SQL node only, since the order of execution of statements affecting privileges from different SQL nodes cannot be guaranteed to be the same on all SQL nodes.

Implications for upgrades. Due to changes in the MySQL server's privilege system (see Section 6.2.3, “Grant Tables”), privilege tables using the `NDB` storage engine do not function correctly in NDB 8.0. It is safe but not necessary to retain such privilege tables created in NDB 7.6 or earlier, but they are no longer used for access control. Beginning with NDB 8.0.16, a `mysqld` acting as an SQL node and detecting such tables in `NDB` writes a warning to the MySQL server log, and creates `InnoDB` shadow tables local to itself; such shadow tables are created on each MySQL server connected to the cluster. When performing an upgrade from NDB 7.6 or earlier, the privilege tables using `NDB` can be removed safely using `ndb_drop_table` once all MySQL servers acting as SQL nodes have been upgraded (see Section 22.2.8, “Upgrading and Downgrading NDB Cluster”).

The `ndb_restore` utility's `--restore-privilege-tables` option continues to be honored in NDB 8.0, and can still be used to restore distributed privilege tables present in a backup taken from a previous release of NDB Cluster to a cluster running NDB 8.0. These tables are handled as described in the preceeding paragraph.

- **INFORMATION_SCHEMA changes.** The following changes are made in the display of information regarding Disk Data files in the `INFORMATION_SCHEMA.FILES` table:

  - Tablespaces and log file groups are no longer represented in the `FILES` table. (These constructs are not actually files.)

  - Each data file is now represented by a single row in the `FILES` table. Each undo log file is also now represented in this table by one row only. (Previously, a row was displayed for each copy of each of these files on each data node.)

  In addition, `INFORMATION_SCHEMA` tables are now populated with tablespace statistics for MySQL Cluster tables. (Bug #27167728)

- **Error information with ndb_perror.** The deprecated `--ndb` option for `perror` has been removed. Instead, use `ndb_perror` to obtain error message information from `NDB` error codes. (Bug #81704, Bug #81705, Bug #23523926, Bug #23523957)

- **Condition pushdown enhancements.** Previously, condition pushdown was limited to predicate terms referring to column values from the same table to which the condition was being pushed. In NDB 8.0.16, this restriction is removed such that column values from tables earlier in the query plan can also be referred to from pushed conditions. As of NDB 8.0.18, joins comparing column expressions are supported, as are comparisons between columns in the same table. Columns and column expressions to

be compared must be of exactly the same type; this means they must also be of the same signedness, length, character set, precision, and scale, whenever these attributes apply.

Pushing down larger parts of a condition allows more rows to be filtered out by the data nodes, thereby reducing the number of rows which `mysqld` must handle during join processing. Another benefit of these enhancements is that filtering can be performed in parallel in the LDM threads, rather than in a single mysqld process on an SQL node; this has the potential to improve query performance significantly.

Existing rules for type compatibility between column values being compared continue to apply (see Section 8.2.1.5, “Engine Condition Pushdown Optimization”).

- **Increase in maximum row size.** NDB 8.0.18 increases the maximum number of bytes that can be stored in an `NDBCLUSTER` table from 14000 to 3000 bytes.

  A `BLOB` or `TEXT` column continues to use 264 bytes of this total, as before.

  The maximum offset for a fixed-width column of an `NDB` table is 8188 bytes; this is also unchanged from releases previous to 8.0.18.

  See Section 22.1.7.5, “Limits Associated with Database Objects in NDB Cluster”, for more information.

- **ndb_mgm SHOW command and single user mode.** Beginning with NDB 8.0.17, when the cluster in single user mode, the output of the management client `SHOW` command indicates which API or SQL node has exclusive access while this mode is in effect.

- **Online column renames.** Beginning with NDB 8.0.18, columns of `NDB` tables can be renamed online, using `ALGORITHM=INPLACE`. See Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”, for more information.

- **Improved ndb_mgmd startup times.** Start times for management nodes daemon have been significantly improved in NDB 8.0.18 and later, in the following ways:

  - Due to replacing the list data structure formerly used by `ndb_mgmd` for handling node properties from configuration data with a hash table, overall startup times for the management server have been decreased by a factor of 6 or more.

  - In addition, in cases where data and SQL node host names not present in the management server's `hosts` file are used in the cluster configuration file, `ndb_mgmd` start times can be up to 20 times shorter than was previously the case.

- **NDB API enhancements.** Beginning with NDB 8.0.18, `NdbScanFilter::cmp()` and several comparison methods of `NdbInterpretedCode` can be used to compare table column values with each other. The affected `NdbInterpretedCode` methods are listed here:

  - `branch_col_eq()`
  - `branch_col_ge()`
  - `branch_col_gt()`
  - `branch_col_le()`
  - `branch_col_lt()`
  - `branch_col_ne()`

For all of the methods just listed, table column values to be compared much be of exactly matching types, including with respect to length, precision, signedness, scale, character set, and collation, as applicable.

See the descriptions of the individual API methods for more information.

- **Offline multithreaded index builds.** It is now possible to specify a set of cores to be used for I/O threads performing offline multithreaded builds of ordered indexes, as opposed to normal I/O duties such as file I/O , compression , or decompression. “Offline” in this context refers to building of ordered indexes performed when the parent table is not being written to; such building takes place when an `NDB` cluster performs a node or system restart, or as part of restoring a cluster from backup using `ndb_restore --rebuild-indexes`.

  In addition, the default behaviour for offline index build work is modified to use all cores available to `ndbmtd`, rather limiting itself to the core reserved for the I/O thread. Doing so can improve restart and restore times and performance, availability, and the user experience.

  This enhancement is implemented as follows:

  1. The default value for `BuildIndexThreads` is changed from 0 to 128. This means that offline ordered index builds are now multithreaded by default.

  2. The default value for `TwoPassInitialNodeRestartCopy` is changed from `false` to `true`. This means that an initial node restart first copies all data from a “live” node to one that is starting —without creating any indexes—builds ordered indexes offline, and then again synchronizes its data with the live node, that is, synchronizing twice and building indexes offline between the two synchonizations. This causes an initial node restart to behave more like the normal restart of a node, and reduces the time required for building indexes.

  3. A new thread type (`idxbld`) is defined for the `ThreadConfig` configuration parameter, to allow locking of offline index build threads to specific CPUs.

  In addition, `NDB` now distinguishes the thread types that are accessible to `ThreadConfig` by these two criteria:

  1. Whether the thread is an execution thread. Threads of types `main`, `ldm`, `recv`, `rep`, `tc`, and `send` are execution threads; thread types `io`, `watchdog`, and `idxbld` are not.

  2. Whether the allocation of the thread to a given task is permanent or temporary. Currently all thread types except `idxbld` are permanent.

  For additonal information, see the descriptions of the indicated parameters in the Manual. (Bug #25835748, Bug #26928111)

- **logbuffers table backup process information.** When performing an NDB backup, the `ndbinfo.logbuffers` table now displays information regarding buffer usage by the backup process on each data node. This is implemented as rows reflecting two new log types in addition to `REDO` and `DD-UNDO`. One of these rows has the log type `BACKUP-DATA`, which shows the amount of data buffer used during backup to copy fragments to backup files. The other row has the log type `BACKUP-LOG`, which displays the amount of log buffer used during the backup to record changes made after the backup has started. One each of these `log_type` rows is shown in the `logbuffers` table for each data node in the cluster. Rows having these two log types are present in the table only while an NDB backup is currently in progress. (Bug #25822988)

- **ndbinfo.processes table on Windows.** The process ID of the monitor process used on Windows platforms by `RESTART` to spawn and restart a `mysqld` is now shown in the `processes` table as an `angel_pid`.

- **String hashing improvements.** Prior to NDB 8.0, all string hashing was based on first transforming the string into a normalized form, then MD5-hashing the resulting binary image. This could give rise to some performance problems, for the following reasons:

  - The normalized string is always space padded to its full length. For a `VARCHAR`, this often involved adding more spaces than there were characters in the original string.
  - The string libraries were not optimized for this space padding, which added considerable overhead in some use cases.
  - The padding semantics varied between character sets, some of which were not padded to their full length.
  - The transformed string could become quite large, even without space padding; some Unicode 9.0 collations can transform a single code point into 100 bytes or more of character data.
  - Subsequent MD5 hashing consisted mainly of padding with spaces, and was not particularly efficient, possibly causing additional performance penalties by flushing significant portions of the L1 cache.

  A collation provides its own hash function, which hashes the string directly without first creating a normalized string. In addition, for a Unicode 9.0 collation, the hash is computed without padding. `NDB` now takes advantage of this built-in function whenever hashing a string identified as using a Unicode 9.0 collation.

  Since, for other collations, there are existing databases which are hash partitioned on the transformed string, `NDB` continues to employ the previous method for hashing strings that use these, to maintain compatibility. (Bug #89590, Bug #89604, Bug #89609, Bug #27515000, Bug #27523758, Bug #27522732)

- **RESET MASTER changes.** Because the MySQL Server now executes `RESET MASTER` with a global read lock, the behavior of this statement when used with NDB Cluster has changed in the following two respects:

  - It is no longer guaranteed to be synonchrous; that is, it is now possible that a read coming immediately before `RESET MASTER` is issued may not be logged until after the binary log has been rotated.
  - It now behaves in exactly the same fashion, whether the statement is issued on the same SQL node that is writing the binary log, or on a different SQL node in the same cluster.

  Note

  `SHOW BINLOG EVENTS`, `FLUSH LOGS`, and most data definition statements continue, as they did in previous `NDB` versions, to operate in a synchronous fashion.

- **ndb_restore option usage.** Beginning with NDB 8.0.16, the `--nodeid` and `--backupid` options are both required when invoking `ndb_restore`.

- **ndb_log_bin default.** Beginning with NDB 8.0.16, the default value of the `ndb_log_bin` system variable has changed from `TRUE` to `FALSE`.

- **Dynamic transactional resource allocation.** Allocation of resources in the transaction corrdinator (see The DBTC Block) is now performed using dynamic

memory pools. This means that resource allocation determined by data node configuration parameters such as `MaxDMLOperationsPerTransaction`, `MaxNoOfConcurrentIndexOperations`, `MaxNoOfConcurrentOperations`, `MaxNoOfConcurrentScans`, `MaxNoOfConcurrentTransactions`, `MaxNoOfFiredTriggers`, `MaxNoOfLocalScans`, and `TransactionBufferMemory` is now done in such a way that, if the load represented by each of these parameters is within the target load for all such resources, others of these resources can be limited so as not to exceed the total resources available.

As part of this work, several new data node parameters controlling transactional resources in `DBTC`, listed here, have been added:

- `ReservedConcurrentIndexOperations`
- `ReservedConcurrentOperations`
- `ReservedConcurrentScans`
- `ReservedConcurrentTransactions`
- `ReservedFiredTriggers`
- `ReservedLocalScans`
- `ReservedTransactionBufferMemory`.

See the descriptions of the parameters just listed for further information.

- **Backups using multiple LDMs per data node.** `NDB` backups can now be performed in a parallel fashion on individual data nodes using multiple local data managers (LDMs). (Previously, backups were done in parallel across data nodes, but were always serial within data node processes.) No special syntax is required for the `START BACKUP` command in the `ndb_mgm` client to enable this feature, but all data nodes must be using multiple LDMs. This means that data nodes must be running `ndbmtd` (`ndbd` is single-threaded and thus always has only one LDM) and they must be configured to use multiple LDMs before taking the backup; you can do this by choosing an appropriate setting for one of the multi-threaded data node configuration parameters `MaxNoOfExecutionThreads` or `ThreadConfig`.

  Backups using multiple LDMs create subdirectories, one per LDM, under the `BACKUP/BACKUP-`*`backup_id/`* directory. `ndb_restore` now detects these subdirectories automatically, and if they exist, attempts to restore the backup in parallel; see Section 22.4.23.2, “Restoring from a backup taken in parallel”, for details. (Single-threaded backups are restored as in previous versions of `NDB`.) It is also possible to restore backups taken in parallel using an `ndb_restore` binary from a previous version of NDB Cluster by modifying the usual restore procedure; Restoring a parallel backup serially, provides information on how to do this.

- **ARM support (source only).** Beginning with NDB 8.0.18, it is possible to build `NDB` from source for 64-bit `ARM` CPUs. Currently, this support is source-only, and we do not provide any precompiled binaries for this platform.

- **Binary configuration file enhancements.** Beginning with NDB 8.0.18, a new format is used for the management server's binary configuration file. Previously, a maximum of 16381 sections could appear in the cluster configuration file; now the maximum number of sections is 4G. This is intended to support larger numbers of nodes in a cluster than was possible before this change.

  Upgrades to the new format are relatively seamless, and should seldom if ever require manual intervention, as the management server continues to be able to read the old format without issue. A downgrade from NDB 8.0.18 (or later) to an older version of the NDB Cluster software requires manual

removal of any binary configuration files or, alternatively, starting the older management server binary with the `--initial` option.

For more information, see Section 22.2.8, “Upgrading and Downgrading NDB Cluster”.

- **Increased number of data nodes and replicas.** NDB 8.0.18 increases the maximum number of data nodes supported per cluster to 144 (previously, this was 48). Data nodes can now use node IDs in the range 1 to 144, inclusive.

  Previously, the recommended node IDs for management nodes were 49 and 50. These are still supported for management nodes, but using them as such limits the maximum number of data nodes to 142; for this reason, it is now recommended that node IDs 145 and 146 are used for management nodes.

22.1.5 Options, Variables, and Parameters Added, Deprecated or Removed in NDB 8.0

- Node Configuration Parameters Introduced in NDB 8.0
- Node Configuration Parameters Deprecated in NDB 8.0
- Node Configuration Parameters Removed in NDB 8.0
- MySQL Server Options and Variables Introduced in NDB 8.0
- MySQL Server Options and Variables Deprecated in NDB 8.0
- MySQL Server Options and Variables Removed in NDB 8.0

The next few sections contain information about `NDB` configuration parameters and NDB-specific `mysqld` options and variables that have been added to, deprecated in, or removed from NDB 8.0.

Node Configuration Parameters Introduced in NDB 8.0

The following node configuration parameters have been added in NDB 8.0.

- `ReservedConcurrentIndexOperations`: Number of simultaneous index operations having dedicated resources on one data node. Added in NDB 8.0.16.
- `ReservedConcurrentOperations`: Number of simultaneous operations having dedicated resources in transaction coordinators on one data node. Added in NDB 8.0.16.
- `ReservedConcurrentScans`: Number of simultaneous scans having dedicated resources on one data node. Added in NDB 8.0.16.
- `ReservedConcurrentTransactions`: Number of simultaneous transactions having dedicated resources on one data node. Added in NDB 8.0.16.
- `ReservedFiredTriggers`: Number of triggers having dedicated resources on one data node. Added in NDB 8.0.16.
- `ReservedLocalScans`: Number of simultaneous fragment scans having dedicated resources on one data node. Added in NDB 8.0.16.
- `ReservedTransactionBufferMemory`: Dynamic buffer space (in bytes) for key and attribute data allocated to each data node. Added in NDB 8.0.16.

Node Configuration Parameters Deprecated in NDB 8.0

No node configuration parameters have been deprecated in NDB 8.0.

Node Configuration Parameters Removed in NDB 8.0

No node configuration parameters have been removed from NDB 8.0.

MySQL Server Options and Variables Introduced in NDB 8.0

The following `mysqld` system variables, status variables, and options have been added in NDB 8.0.

- `Ndb_metadata_blacklist_size`: Number of NDB metadata objects that NDB binlog thread has failed to synchronize. Added in NDB 8.0.18.
- `Ndb_metadata_detected_count`: Number of times NDB metadata change monitor thread has detected changes. Added in NDB 8.0.16.
- `Ndb_metadata_synced_count`: Number of NDB metadata objects which have been synchronized. Added in NDB 8.0.18.
- `Ndb_trans_hint_count_session`: Number of transactions using hints that have been started in this session. Added in NDB 8.0.17.
- `ndb-schema-dist-timeout`: How long to wait before detecting timeout during schema distribution. Added in NDB 8.0.17.
- `ndb_dbg_check_shares`: Check for any lingering shares (debug builds only). Added in NDB 8.0.13.
- `ndb_metadata_check`: Enable auto-detection of NDB metadata changes with respect to MySQL data dictionary; enabled by default. Added in NDB 8.0.16.
- `ndb_metadata_check_interval`: Interval in seconds to perform check for NDB metadata changes with respect to MySQL data dictionary. Added in NDB 8.0.16.
- `ndb_schema_dist_lock_wait_timeout`: Time during schema distribution to wait for lock before returning error. Added in NDB 8.0.18.
- `ndb_schema_dist_timeout`: Time to wait before detecting timeout during schema distribution. Added in NDB 8.0.16.
- `ndb_schema_dist_upgrade_allowed`: Allow schema distribution table upgrade when connecting to NDB. Added in NDB 8.0.17.
- `ndbinfo`: Enable ndbinfo plugin, if supported. Added in NDB 8.0.13.

MySQL Server Options and Variables Deprecated in NDB 8.0

No system variables, status variables, or options have been deprecated in NDB 8.0.

MySQL Server Options and Variables Removed in NDB 8.0

No system variables, status variables, or options have been removed from NDB 8.0.

22.1.6 MySQL Server Using InnoDB Compared with NDB Cluster

MySQL Server offers a number of choices in storage engines. Since both `NDB` and `InnoDB` can serve as transactional MySQL storage engines, users of MySQL Server sometimes become interested in NDB Cluster. They see `NDB` as a possible alternative or upgrade to the default `InnoDB` storage engine in MySQL 8.0. While `NDB` and `InnoDB` share common characteristics, there are differences in architecture

and implementation, so that some existing MySQL Server applications and usage scenarios can be a good fit for NDB Cluster, but not all of them.

In this section, we discuss and compare some characteristics of the NDB storage engine used by NDB 8.0 with InnoDB used in MySQL 8.0. The next few sections provide a technical comparison. In many instances, decisions about when and where to use NDB Cluster must be made on a case-by-case basis, taking all factors into consideration. While it is beyond the scope of this documentation to provide specifics for every conceivable usage scenario, we also attempt to offer some very general guidance on the relative suitability of some common types of applications for NDB as opposed to InnoDB back ends.

NDB Cluster 8.0 uses a mysqld based on MySQL 8.0, including support for InnoDB 1.1. While it is possible to use InnoDB tables with NDB Cluster, such tables are not clustered. It is also not possible to use programs or libraries from an NDB Cluster 8.0 distribution with MySQL Server 8.0, or the reverse.

While it is also true that some types of common business applications can be run either on NDB Cluster or on MySQL Server (most likely using the InnoDB storage engine), there are some important architectural and implementation differences. Section 22.1.6.1, “Differences Between the NDB and InnoDB Storage Engines”, provides a summary of the these differences. Due to the differences, some usage scenarios are clearly more suitable for one engine or the other; see Section 22.1.6.2, “NDB and InnoDB Workloads”. This in turn has an impact on the types of applications that better suited for use with NDB or InnoDB. See Section 22.1.6.3, “NDB and InnoDB Feature Usage Summary”, for a comparison of the relative suitability of each for use in common types of database applications.

For information about the relative characteristics of the NDB and MEMORY storage engines, see When to Use MEMORY or NDB Cluster.

See Chapter 16, *Alternative Storage Engines*, for additional information about MySQL storage engines.

22.1.6.1 Differences Between the NDB and InnoDB Storage Engines

The NDB storage engine is implemented using a distributed, shared-nothing architecture, which causes it to behave differently from InnoDB in a number of ways. For those unaccustomed to working with NDB, unexpected behaviors can arise due to its distributed nature with regard to transactions, foreign keys, table limits, and other characteristics. These are shown in the following table:

Table 22.1 Differences between InnoDB and NDB storage engines

Feature	InnoDB (MySQL 8.0)	NDB 8.0
MySQL Server Version	8.0	8.0
InnoDB Version	InnoDB 8.0.19	InnoDB 8.0.19
NDB Cluster Version	N/A	NDB 8.0.19/8.0.19
Storage Limits	64TB	128TB
Foreign Keys	Yes	Yes
Transactions	All standard types	READ COMMITTED
MVCC	Yes	No
Data Compression	Yes	No (NDB checkpoint and backup files can be compressed)
Large Row Support (> 14K)	Supported for VARBINARY, VARCHAR, BLOB, and TEXT columns	Supported for BLOB and TEXT columns only (Using these types to store very large amounts of data can lower NDB performance)
Replication Support	Asynchronous and semisynchronous replication using	Automatic synchronous replication within an NDB Cluster;

Feature	InnoDB (MySQL 8.0)	NDB 8.0
	MySQL Replication; MySQL Group Replication	asynchronous replication between NDB Clusters, using MySQL Replication (Semisynchronous replication is not supported)
Scaleout for Read Operations	Yes (MySQL Replication)	Yes (Automatic partitioning in NDB Cluster; NDB Cluster Replication)
Scaleout for Write Operations	Requires application-level partitioning (sharding)	Yes (Automatic partitioning in NDB Cluster is transparent to applications)
High Availability (HA)	Built-in, from InnoDB cluster	Yes (Designed for 99.999% uptime)
Node Failure Recovery and Failover	From MySQL Group Replication	Automatic (Key element in NDB architecture)
Time for Node Failure Recovery	30 seconds or longer	Typically < 1 second
Real-Time Performance	No	Yes
In-Memory Tables	No	Yes (Some data can optionally be stored on disk; both in-memory and disk data storage are durable)
NoSQL Access to Storage Engine	Yes	Yes (Multiple APIs, including Memcached, Node.js/JavaScript, Java, JPA, C++, and HTTP/REST)
Concurrent and Parallel Writes	Yes	Up to 48 writers, optimized for concurrent writes
Conflict Detection and Resolution (Multiple Replication Masters)	Yes (MySQL Group Replication)	Yes
Hash Indexes	No	Yes
Online Addition of Nodes	Read/write replicas using MySQL Group Replication	Yes (all node types)
Online Upgrades	Yes (using replication)	Yes
Online Schema Modifications	Yes, as part of MySQL 8.0	Yes

22.1.6.2 NDB and InnoDB Workloads

NDB Cluster has a range of unique attributes that make it ideal to serve applications requiring high availability, fast failover, high throughput, and low latency. Due to its distributed architecture and multi-node implementation, NDB Cluster also has specific constraints that may keep some workloads from performing well. A number of major differences in behavior between the NDB and InnoDB storage engines with regard to some common types of database-driven application workloads are shown in the following table::

Table 22.2 Differences between InnoDB and NDB storage engines, common types of data-driven application workloads.

Workload	InnoDB	NDB Cluster (NDB)
High-Volume OLTP Applications	Yes	Yes

Workload	InnoDB	NDB Cluster (NDB)
DSS Applications (data marts, analytics)	Yes	Limited (Join operations across OLTP datasets not exceeding 3TB in size)
Custom Applications	Yes	Yes
Packaged Applications	Yes	Limited (should be mostly primary key access); NDB Cluster 8.0 supports foreign keys
In-Network Telecoms Applications (HLR, HSS, SDP)	No	Yes
Session Management and Caching	Yes	Yes
E-Commerce Applications	Yes	Yes
User Profile Management, AAA Protocol	Yes	Yes

22.1.6.3 NDB and InnoDB Feature Usage Summary

When comparing application feature requirements to the capabilities of InnoDB with NDB, some are clearly more compatible with one storage engine than the other.

The following table lists supported application features according to the storage engine to which each feature is typically better suited.

Table 22.3 Supported application features according to the storage engine to which each feature is typically better suited

Preferred application requirements for InnoDB	Preferred application requirements for NDB
• Foreign keys **Note** NDB Cluster 8.0 supports foreign keys • Full table scans • Very large databases, rows, or transactions • Transactions other than READ COMMITTED	• Write scaling • 99.999% uptime • Online addition of nodes and online schema operations • Multiple SQL and NoSQL APIs (see NDB Cluster APIs: Overview and Concepts) • Real-time performance • Limited use of BLOB columns • Foreign keys are supported, although their use may have an impact on performance at high throughput

22.1.7 Known Limitations of NDB Cluster

In the sections that follow, we discuss known limitations in current releases of NDB Cluster as compared with the features available when using the MyISAM and InnoDB storage engines. If you check the “Cluster” category in the MySQL bugs database at http://bugs.mysql.com, you can find known bugs in the following categories under “MySQL Server:” in the MySQL bugs database at http://bugs.mysql.com, which we intend to correct in upcoming releases of NDB Cluster:

- NDB Cluster
- Cluster Direct API (NDBAPI)
- Cluster Disk Data
- Cluster Replication
- ClusterJ

This information is intended to be complete with respect to the conditions just set forth. You can report any discrepancies that you encounter to the MySQL bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”. If we do not plan to fix the problem in NDB Cluster 8.0, we will add it to the list.

See Section 22.1.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 8.0” for a list of issues in earlier releases that have been resolved in NDB Cluster 8.0.

Note

Limitations and other issues specific to NDB Cluster Replication are described in Section 22.6.3, “Known Issues in NDB Cluster Replication”.

22.1.7.1 Noncompliance with SQL Syntax in NDB Cluster

Some SQL statements relating to certain MySQL features produce errors when used with `NDB` tables, as described in the following list:

- **Temporary tables.** Temporary tables are not supported. Trying either to create a temporary table that uses the `NDB` storage engine or to alter an existing temporary table to use `NDB` fails with the error `Table storage engine 'ndbcluster' does not support the create option 'TEMPORARY'`.
- **Indexes and keys in NDB tables.** Keys and indexes on NDB Cluster tables are subject to the following limitations:
  - **Column width.** Attempting to create an index on an `NDB` table column whose width is greater than 3072 bytes succeeds, but only the first 3072 bytes are actually used for the index. In such cases, a warning `Specified key was too long; max key length is 3072 bytes` is issued, and a `SHOW CREATE TABLE` statement shows the length of the index as 3072.
  - **TEXT and BLOB columns.** You cannot create indexes on `NDB` table columns that use any of the `TEXT` or `BLOB` data types.
  - **FULLTEXT indexes.** The `NDB` storage engine does not support `FULLTEXT` indexes, which are possible for `MyISAM` and `InnoDB` tables only.

    However, you can create indexes on `VARCHAR` columns of `NDB` tables.
  - **USING HASH keys and NULL.** Using nullable columns in unique keys and primary keys means that queries using these columns are handled as full table scans. To work around this issue, make the column `NOT NULL`, or re-create the index without the `USING HASH` option.
  - **Prefixes.** There are no prefix indexes; only entire columns can be indexed. (The size of an `NDB` column index is always the same as the width of the column in bytes, up to and including 3072 bytes, as described earlier in this section. Also see Section 22.1.7.6, “Unsupported or Missing Features in NDB Cluster”, for additional information.)

- **BIT columns.** A `BIT` column cannot be a primary key, unique key, or index, nor can it be part of a composite primary key, unique key, or index.

- **AUTO_INCREMENT columns.** Like other MySQL storage engines, the `NDB` storage engine can handle a maximum of one `AUTO_INCREMENT` column per table. However, in the case of an NDB table with no explicit primary key, an `AUTO_INCREMENT` column is automatically defined and used as a "hidden" primary key. For this reason, you cannot define a table that has an explicit `AUTO_INCREMENT` column unless that column is also declared using the `PRIMARY KEY` option. Attempting to create a table with an `AUTO_INCREMENT` column that is not the table's primary key, and using the `NDB` storage engine, fails with an error.

- **Restrictions on foreign keys.** Support for foreign key constraints in NDB 8.0 is comparable to that provided by `InnoDB`, subject to the following restrictions:

  - Every column referenced as a foreign key requires an explicit unique key, if it is not the table's primary key.

  - `ON UPDATE CASCADE` is not supported when the reference is to the parent table's primary key.

    This is because an update of a primary key is implemented as a delete of the old row (containing the old primary key) plus an insert of the new row (with a new primary key). This is not visible to the `NDB` kernel, which views these two rows as being the same, and thus has no way of knowing that this update should be cascaded.

  - As of NDB 8.0.16: `ON DELETE CASCADE` is not supported where the child table contains one or more columns of any of the `TEXT` or `BLOB` types. (Bug #89511, Bug #27484882)

  - `SET DEFAULT` is not supported. (Also not supported by `InnoDB`.)

  - The `NO ACTION` keyword is accepted but treated as `RESTRICT`. `NO ACTION`, which is a standard SQL keyword, is the default in MySQL 8.0. (Also the same as with `InnoDB`.)

  - In earlier versions of NDB Cluster, when creating a table with foreign key referencing an index in another table, it sometimes appeared possible to create the foreign key even if the order of the columns in the indexes did not match, due to the fact that an appropriate error was not always returned internally. A partial fix for this issue improved the error used internally to work in most cases; however, it remains possible for this situation to occur in the event that the parent index is a unique index. (Bug #18094360)

  For more information, see Section 13.1.20.6, "Using FOREIGN KEY Constraints", and Section 1.8.3.2, "FOREIGN KEY Constraints".

- **NDB Cluster and geometry data types.**
  Geometry data types (`WKT` and `WKB`) are supported for `NDB` tables. However, spatial indexes are not supported.

- **Character sets and binary log files.** Currently, the `ndb_apply_status` and `ndb_binlog_index` tables are created using the `latin1` (ASCII) character set. Because names of binary logs are recorded in this table, binary log files named using non-Latin characters are not referenced correctly in these tables. This is a known issue, which we are working to fix. (Bug #50226)

  To work around this problem, use only Latin-1 characters when naming binary log files or setting any the `--basedir`, `--log-bin`, or `--log-bin-index` options.

- **Creating NDB tables with user-defined partitioning.** Support for user-defined partitioning in NDB Cluster is restricted to [LINEAR] KEY partitioning. Using any other partitioning type with ENGINE=NDB or ENGINE=NDBCLUSTER in a CREATE TABLE statement results in an error.

  It is possible to override this restriction, but doing so is not supported for use in production settings. For details, see User-defined partitioning and the NDB storage engine (NDB Cluster).

  Default partitioning scheme. All NDB Cluster tables are by default partitioned by KEY using the table's primary key as the partitioning key. If no primary key is explicitly set for the table, the "hidden" primary key automatically created by the NDB storage engine is used instead. For additional discussion of these and related issues, see Section 23.2.5, "KEY Partitioning".

  CREATE TABLE and ALTER TABLE statements that would cause a user-partitioned NDBCLUSTER table not to meet either or both of the following two requirements are not permitted, and fail with an error:

  1. The table must have an explicit primary key.

  2. All columns listed in the table's partitioning expression must be part of the primary key.

  Exception. If a user-partitioned NDBCLUSTER table is created using an empty column-list (that is, using PARTITION BY [LINEAR] KEY()), then no explicit primary key is required.

  Maximum number of partitions for NDBCLUSTER tables. The maximum number of partitions that can defined for a NDBCLUSTER table when employing user-defined partitioning is 8 per node group. (See Section 22.1.2, "NDB Cluster Nodes, Node Groups, Replicas, and Partitions", for more information about NDB Cluster node groups.

  DROP PARTITION not supported. It is not possible to drop partitions from NDB tables using ALTER TABLE ... DROP PARTITION. The other partitioning extensions to ALTER TABLE—ADD PARTITION, REORGANIZE PARTITION, and COALESCE PARTITION—are supported for NDB tables, but use copying and so are not optimized. See Section 23.3.1, "Management of RANGE and LIST Partitions" and Section 13.1.9, "ALTER TABLE Syntax".

- **Row-based replication.**
  When using row-based replication with NDB Cluster, binary logging cannot be disabled. That is, the NDB storage engine ignores the value of sql_log_bin.

- **JSON data type.** The MySQL JSON data type is supported for NDB tables in the mysqld supplied with NDB 8.0.

  An NDB table can have a maximum of 3 JSON columns.

  The NDB API has no special provision for working with JSON data, which it views simply as BLOB data. Handling data as JSON must be performed by the application.

22.1.7.2 Limits and Differences of NDB Cluster from Standard MySQL Limits

In this section, we list limits found in NDB Cluster that either differ from limits found in, or that are not found in, standard MySQL.

Memory usage and recovery. Memory consumed when data is inserted into an NDB table is not automatically recovered when deleted, as it is with other storage engines. Instead, the following rules hold true:

- A DELETE statement on an NDB table makes the memory formerly used by the deleted rows available for re-use by inserts on the same table only. However, this memory can be made available for general re-use by performing OPTIMIZE TABLE.

A rolling restart of the cluster also frees any memory used by deleted rows. See Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”.

- A `DROP TABLE` or `TRUNCATE TABLE` operation on an `NDB` table frees the memory that was used by this table for re-use by any `NDB` table, either by the same table or by another `NDB` table.

  Note

  Recall that `TRUNCATE TABLE` drops and re-creates the table. See Section 13.1.37, “TRUNCATE TABLE Syntax”.

- **Limits imposed by the cluster's configuration.**
  A number of hard limits exist which are configurable, but available main memory in the cluster sets limits. See the complete list of configuration parameters in Section 22.3.3, “NDB Cluster Configuration Files”. Most configuration parameters can be upgraded online. These hard limits include:

  - Database memory size and index memory size (`DataMemory` and `IndexMemory`, respectively).

    `DataMemory` is allocated as 32KB pages. As each `DataMemory` page is used, it is assigned to a specific table; once allocated, this memory cannot be freed except by dropping the table.

    See Section 22.3.3.6, “Defining NDB Cluster Data Nodes”, for more information.

  - The maximum number of operations that can be performed per transaction is set using the configuration parameters `MaxNoOfConcurrentOperations` and `MaxNoOfLocalOperations`.

    Note

    Bulk loading, `TRUNCATE TABLE`, and `ALTER TABLE` are handled as special cases by running multiple transactions, and so are not subject to this limitation.

  - Different limits related to tables and indexes. For example, the maximum number of ordered indexes in the cluster is determined by `MaxNoOfOrderedIndexes`, and the maximum number of ordered indexes per table is 16.

- **Node and data object maximums.** The following limits apply to numbers of cluster nodes and metadata objects:

  - As of NDB 8.0.18, the maximum number of data nodes is 145. (Previously, this was 48.)

    A data node must have a node ID in the range of 1 to 144, inclusive. (In NDB 8.0.17 and earlier releases, this was 1 to 48, inclusive.)

    Management and API nodes may use node IDs in the range 1 to 255, inclusive.

  - The total maximum number of nodes in an NDB Cluster is 255. This number includes all SQL nodes (MySQL Servers), API nodes (applications accessing the cluster other than MySQL servers), data nodes, and management servers.

  - The maximum number of metadata objects in current versions of NDB Cluster is 20320. This limit is hard-coded.

  See Section 22.1.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 8.0”, for more information.

22.1.7.3 Limits Relating to Transaction Handling in NDB Cluster

A number of limitations exist in NDB Cluster with regard to the handling of transactions. These include the following:

- **Transaction isolation level.** The NDBCLUSTER storage engine supports only the READ COMMITTED transaction isolation level. (InnoDB, for example, supports READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, and SERIALIZABLE.) You should keep in mind that NDB implements READ COMMITTED on a per-row basis; when a read request arrives at the data node storing the row, what is returned is the last committed version of the row at that time.

  Uncommitted data is never returned, but when a transaction modifying a number of rows commits concurrently with a transaction reading the same rows, the transaction performing the read can observe "before" values, "after" values, or both, for different rows among these, due to the fact that a given row read request can be processed either before or after the commit of the other transaction.

  To ensure that a given transaction reads only before or after values, you can impose row locks using SELECT ... LOCK IN SHARE MODE. In such cases, the lock is held until the owning transaction is committed. Using row locks can also cause the following issues:

  - Increased frequency of lock wait timeout errors, and reduced concurrency
  - Increased transaction processing overhead due to reads requiring a commit phase
  - Possibility of exhausting the available number of concurrent locks, which is limited by MaxNoOfConcurrentOperations

  NDB uses READ COMMITTED for all reads unless a modifier such as LOCK IN SHARE MODE or FOR UPDATE is used. LOCK IN SHARE MODE causes shared row locks to be used; FOR UPDATE causes exclusive row locks to be used. Unique key reads have their locks upgraded automatically by NDB to ensure a self-consistent read; BLOB reads also employ extra locking for consistency.

  See Section 22.5.3.4, "NDB Cluster Backup Troubleshooting", for information on how NDB Cluster's implementation of transaction isolation level can affect backup and restoration of NDB databases.

- **Transactions and BLOB or TEXT columns.** NDBCLUSTER stores only part of a column value that uses any of MySQL's BLOB or TEXT data types in the table visible to MySQL; the remainder of the BLOB or TEXT is stored in a separate internal table that is not accessible to MySQL. This gives rise to two related issues of which you should be aware whenever executing SELECT statements on tables that contain columns of these types:

  1. For any SELECT from an NDB Cluster table: If the SELECT includes a BLOB or TEXT column, the READ COMMITTED transaction isolation level is converted to a read with read lock. This is done to guarantee consistency.
  2. For any SELECT which uses a unique key lookup to retrieve any columns that use any of the BLOB or TEXT data types and that is executed within a transaction, a shared read lock is held on the table for the duration of the transaction—that is, until the transaction is either committed or aborted.

     This issue does not occur for queries that use index or table scans, even against NDB tables having BLOB or TEXT columns.

     For example, consider the table t defined by the following CREATE TABLE statement:

```
CREATE TABLE t (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    b INT NOT NULL,
    c INT NOT NULL,
```

```
    d TEXT,
    INDEX i(b),
    UNIQUE KEY u(c)
) ENGINE = NDB,
```

Either of the following queries on `t` causes a shared read lock, because the first query uses a primary key lookup and the second uses a unique key lookup:

```
SELECT * FROM t WHERE a = 1;

SELECT * FROM t WHERE c = 1;
```

However, none of the four queries shown here causes a shared read lock:

```
SELECT * FROM t WHERE b = 1;

SELECT * FROM t WHERE d = '1';

SELECT * FROM t;

SELECT b,c WHERE a = 1;
```

This is because, of these four queries, the first uses an index scan, the second and third use table scans, and the fourth, while using a primary key lookup, does not retrieve the value of any `BLOB` or `TEXT` columns.

You can help minimize issues with shared read locks by avoiding queries that use unique key lookups that retrieve `BLOB` or `TEXT` columns, or, in cases where such queries are not avoidable, by committing transactions as soon as possible afterward.

- **Rollbacks.** There are no partial transactions, and no partial rollbacks of transactions. A duplicate key or similar error causes the entire transaction to be rolled back.

  This behavior differs from that of other transactional storage engines such as `InnoDB` that may roll back individual statements.

- **Transactions and memory usage.**
  As noted elsewhere in this chapter, NDB Cluster does not handle large transactions well; it is better to perform a number of small transactions with a few operations each than to attempt a single large transaction containing a great many operations. Among other considerations, large transactions require very large amounts of memory. Because of this, the transactional behavior of a number of MySQL statements is affected as described in the following list:

  - `TRUNCATE TABLE` is not transactional when used on `NDB` tables. If a `TRUNCATE TABLE` fails to empty the table, then it must be re-run until it is successful.

  - `DELETE FROM` (even with no `WHERE` clause) *is* transactional. For tables containing a great many rows, you may find that performance is improved by using several `DELETE FROM ... LIMIT ...` statements to “chunk” the delete operation. If your objective is to empty the table, then you may wish to use `TRUNCATE TABLE` instead.

  - **LOAD DATA statements.** `LOAD DATA` is not transactional when used on `NDB` tables.

Important

When executing a `LOAD DATA` statement, the `NDB` engine performs commits at irregular intervals that enable better utilization of the communication network. It is not possible to know ahead of time when such commits take place.

- **ALTER TABLE and transactions.** When copying an `NDB` table as part of an `ALTER TABLE`, the creation of the copy is nontransactional. (In any case, this operation is rolled back when the copy is deleted.)

- **Transactions and the COUNT() function.** When using NDB Cluster Replication, it is not possible to guarantee the transactional consistency of the `COUNT()` function on the slave. In other words, when performing on the master a series of statements (`INSERT`, `DELETE`, or both) that changes the number of rows in a table within a single transaction, executing `SELECT COUNT(*) FROM table` queries on the slave may yield intermediate results. This is due to the fact that `SELECT COUNT(...)` may perform dirty reads, and is not a bug in the `NDB` storage engine. (See Bug #31321 for more information.)

22.1.7.4 NDB Cluster Error Handling

Starting, stopping, or restarting a node may give rise to temporary errors causing some transactions to fail. These include the following cases:

- **Temporary errors.** When first starting a node, it is possible that you may see Error 1204 `Temporary failure, distribution changed` and similar temporary errors.

- **Errors due to node failure.** The stopping or failure of any data node can result in a number of different node failure errors. (However, there should be no aborted transactions when performing a planned shutdown of the cluster.)

In either of these cases, any errors that are generated must be handled within the application. This should be done by retrying the transaction.

See also Section 22.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”.

22.1.7.5 Limits Associated with Database Objects in NDB Cluster

Some database objects such as tables and indexes have different limitations when using the `NDBCLUSTER` storage engine:

- **Number of database objects.** The maximum number of *all* `NDB` database objects in a single NDB Cluster—including databases, tables, and indexes—is limited to 20320.

- **Attributes per table.** The maximum number of attributes (that is, columns and indexes) that can belong to a given table is 512.

- **Attributes per key.** The maximum number of attributes per key is 32.

- **Row size.** Beginning with NDB 8.0.18, the maximum permitted size of any one row is 30000 bytes, in NDB 8.0.17 and earlier releases, this limit is 14000 bytes.

  Each `BLOB` or `TEXT` column contributes 256 + 8 = 264 bytes to this total; see String Type Storage Requirements, for more information relating to these types.

  In addition, the maximum offset for a fixed-width column of an `NDB` table is 8188 bytes; attempting to create a table that violates this limitation fails with NDB error 851 `Maximum offset for fixed-`

`size columns exceeded`. For memory-based columns, you can work around this limitation by using a variable-width column type such as `VARCHAR` or defining the column as `COLUMN_FORMAT=DYNAMIC`; this does not work with columns stored on disk. For disk-based columns, you may be able to do so by reordering one or more of the table's disk-based columns such that the combined width of all but the disk-based column defined last in the `CREATE TABLE` statement used to create the table does not exceed 8188 bytes, less any possible rounding performed for some data types such as `CHAR` or `VARCHAR`; otherwise it is necessary to use memory-based storage for one or more of the offending column or columns instead.

- **BIT column storage per table.** The maximum combined width for all `BIT` columns used in a given `NDB` table is 4096.
- **FIXED column storage.** NDB Cluster 8.0 supports a maximum of 128 TB per fragment of data in `FIXED` columns.

22.1.7.6 Unsupported or Missing Features in NDB Cluster

A number of features supported by other storage engines are not supported for `NDB` tables. Trying to use any of these features in NDB Cluster does not cause errors in or of itself; however, errors may occur in applications that expects the features to be supported or enforced. Statements referencing such features, even if effectively ignored by `NDB`, must be syntactically and otherwise valid.

- **Index prefixes.** Prefixes on indexes are not supported for `NDB` tables. If a prefix is used as part of an index specification in a statement such as `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX`, the prefix is not created by `NDB`.

  A statement containing an index prefix, and creating or modifying an `NDB` table, must still be syntactically valid. For example, the following statement always fails with Error 1089 `Incorrect prefix key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique prefix keys`, regardless of storage engine:

  ```
  CREATE TABLE t1 (
      c1 INT NOT NULL,
      c2 VARCHAR(100),
      INDEX i1 (c2(500))
  );
  ```

  This happens on account of the SQL syntax rule that no index may have a prefix larger than itself.

- **Savepoints and rollbacks.** Savepoints and rollbacks to savepoints are ignored as in `MyISAM`.
- **Durability of commits.** There are no durable commits on disk. Commits are replicated, but there is no guarantee that logs are flushed to disk on commit.
- **Replication.** Statement-based replication is not supported. Use `--binlog-format=ROW` (or `--binlog-format=MIXED`) when setting up cluster replication. See Section 22.6, “NDB Cluster Replication”, for more information.

  Replication using global transaction identifiers (GTIDs) is not compatible with NDB Cluster, and is not supported in NDB Cluster 8.0. Do not enable GTIDs when using the `NDB` storage engine, as this is very likely to cause problems up to and including failure of NDB Cluster Replication.

  Semisynchronous replication is not supported in NDB Cluster.

- **Generated columns.** The `NDB` storage engine does not support indexes on virtual generated columns.

As with other storage engines, you can create an index on a stored generated column, but you should bear in mind that `NDB` uses `DataMemory` for storage of the generated column as well as `IndexMemory` for the index. See JSON columns and indirect indexing in NDB Cluster, for an example.

NDB Cluster writes changes in stored generated columns to the binary log, but does log not those made to virtual columns. This should not effect NDB Cluster Replication or replication between `NDB` and other MySQL storage engines.

Note

See Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”, for more information relating to limitations on transaction handling in `NDB`.

22.1.7.7 Limitations Relating to Performance in NDB Cluster

The following performance issues are specific to or especially pronounced in NDB Cluster:

- **Range scans.** There are query performance issues due to sequential access to the `NDB` storage engine; it is also relatively more expensive to do many range scans than it is with either `MyISAM` or `InnoDB`.
- **Reliability of Records in range.** The `Records in range` statistic is available but is not completely tested or officially supported. This may result in nonoptimal query plans in some cases. If necessary, you can employ `USE INDEX` or `FORCE INDEX` to alter the execution plan. See Section 8.9.4, “Index Hints”, for more information on how to do this.
- **Unique hash indexes.** Unique hash indexes created with `USING HASH` cannot be used for accessing a table if `NULL` is given as part of the key.

22.1.7.8 Issues Exclusive to NDB Cluster

The following are limitations specific to the `NDB` storage engine:

- **Machine architecture.** All machines used in the cluster must have the same architecture. That is, all machines hosting nodes must be either big-endian or little-endian, and you cannot use a mixture of both. For example, you cannot have a management node running on a PowerPC which directs a data node that is running on an x86 machine. This restriction does not apply to machines simply running `mysql` or other clients that may be accessing the cluster's SQL nodes.
- **Binary logging.**
  NDB Cluster has the following limitations or restrictions with regard to binary logging:
  - `sql_log_bin` has no effect on data operations; however, it is supported for schema operations.
  - NDB Cluster cannot produce a binary log for tables having `BLOB` columns but no primary key.
  - Only the following schema operations are logged in a cluster binary log which is *not* on the `mysqld` executing the statement:
    - `CREATE TABLE`
    - `ALTER TABLE`
    - `DROP TABLE`
    - `CREATE DATABASE` / `CREATE SCHEMA`

- `DROP DATABASE` / `DROP SCHEMA`
- `CREATE TABLESPACE`
- `ALTER TABLESPACE`
- `DROP TABLESPACE`
- `CREATE LOGFILE GROUP`
- `ALTER LOGFILE GROUP`
- `DROP LOGFILE GROUP`

- Schema operations (DDL statements) are rejected while any data node restarts.
- **Number of replicas.** The number of replicas, as determined by the `NoOfReplicas` data node configuration parameter, is the number of copies of all data stored by NDB Cluster. Setting this parameter to 1 means there is only a single copy; in this case, no redundancy is provided, and the loss of a data node entails loss of data. To guarantee redundancy, and thus preservation of data even if a data node fails, set this parameter to 2, which is the default and recommended value in production.

  Setting `NoOfReplicas` to a value greater than 2 is possible (to a maximum of 4) but unnecessary to guard against loss of data. In addition, *values greater than 2 for this parameter are not supported in production*.

See also Section 22.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”.

22.1.7.9 Limitations Relating to NDB Cluster Disk Data Storage

Disk Data object maximums and minimums. Disk data objects are subject to the following maximums and minimums:

- Maximum number of tablespaces: 2^{32} (4294967296)
- Maximum number of data files per tablespace: 2^{16} (65536)
- The minimum and maximum possible sizes of extents for tablespace data files are 32K and 2G, respectively. See Section 13.1.21, “CREATE TABLESPACE Syntax”, for more information.

In addition, when working with NDB Disk Data tables, you should be aware of the following issues regarding data files and extents:

- Data files use `DataMemory`. Usage is the same as for in-memory data.
- Data files use file descriptors. It is important to keep in mind that data files are always open, which means the file descriptors are always in use and cannot be re-used for other system tasks.
- Extents require sufficient `DiskPageBufferMemory`; you must reserve enough for this parameter to account for all memory used by all extents (number of extents times size of extents).

Disk Data tables and diskless mode. Use of Disk Data tables is not supported when running the cluster in diskless mode.

22.1.7.10 Limitations Relating to Multiple NDB Cluster Nodes

Multiple SQL nodes.

The following are issues relating to the use of multiple MySQL servers as NDB Cluster SQL nodes, and are specific to the `NDBCLUSTER` storage engine:

- **No distributed table locks.** A `LOCK TABLES` works only for the SQL node on which the lock is issued; no other SQL node in the cluster "sees" this lock. This is also true for a lock issued by any statement that locks tables as part of its operations. (See next item for an example.)
- **ALTER TABLE operations.** `ALTER TABLE` is not fully locking when running multiple MySQL servers (SQL nodes). (As discussed in the previous item, NDB Cluster does not support distributed table locks.)

Multiple management nodes.
When using multiple management servers:

- If any of the management servers are running on the same host, you must give nodes explicit IDs in connection strings because automatic allocation of node IDs does not work across multiple management servers on the same host. This is not required if every management server resides on a different host.
- When a management server starts, it first checks for any other management server in the same NDB Cluster, and upon successful connection to the other management server uses its configuration data. This means that the management server `--reload` and `--initial` startup options are ignored unless the management server is the only one running. It also means that, when performing a rolling restart of an NDB Cluster with multiple management nodes, the management server reads its own configuration file if (and only if) it is the only management server running in this NDB Cluster. See Section 22.5.5, "Performing a Rolling Restart of an NDB Cluster", for more information.

Multiple network addresses. Multiple network addresses per data node are not supported. Use of these is liable to cause problems: In the event of a data node failure, an SQL node waits for confirmation that the data node went down but never receives it because another route to that data node remains open. This can effectively make the cluster inoperable.

Note

It is possible to use multiple network hardware *interfaces* (such as Ethernet cards) for a single data node, but these must be bound to the same address. This also means that it not possible to use more than one `[tcp]` section per connection in the `config.ini` file. See Section 22.3.3.10, "NDB Cluster TCP/IP Connections", for more information.

22.1.7.11 Previous NDB Cluster Issues Resolved in NDB Cluster 8.0

A number of limitations and related issues that existed in earlier versions of NDB Cluster have been resolved in NDB 8.0. These are described briefly in the following list:

Database and table names. Prior to NDB 8.0.18, when using the `NDB` storage engine, the maximum allowed length both for database names and for table names is 63 characters, and a statement using a database name or table name longer than this limit failed with an appropriate error. As of NDB 8.0.18, this restriction is lifted and identifiers for `NDB` databases and tables may now use up to 64 bytes, as with other MySQL database and table names.

22.2 NDB Cluster Installation

This section describes the basics for planning, installing, configuring, and running an NDB Cluster. Whereas the examples in Section 22.3, "Configuration of NDB Cluster" provide more in-depth information on a variety of clustering options and configuration, the result of following the guidelines and procedures outlined here should be a usable NDB Cluster which meets the *minimum* requirements for availability and safeguarding of data.

This section covers hardware and software requirements; networking issues; installation of NDB Cluster; basic configuration issues; starting, stopping, and restarting the cluster; loading of a sample database; and performing queries.

NDB Cluster also provides the NDB Cluster Auto-Installer, a web-based graphical installer, as part of the NDB Cluster distribution. The Auto-Installer can be used to perform basic installation and setup of an NDB Cluster on one (for testing) or more host computers. See Section 22.2.1, “The NDB Cluster Auto-Installer”, for more information.

Assumptions. The following sections make a number of assumptions regarding the cluster's physical and network configuration. These assumptions are discussed in the next few paragraphs.

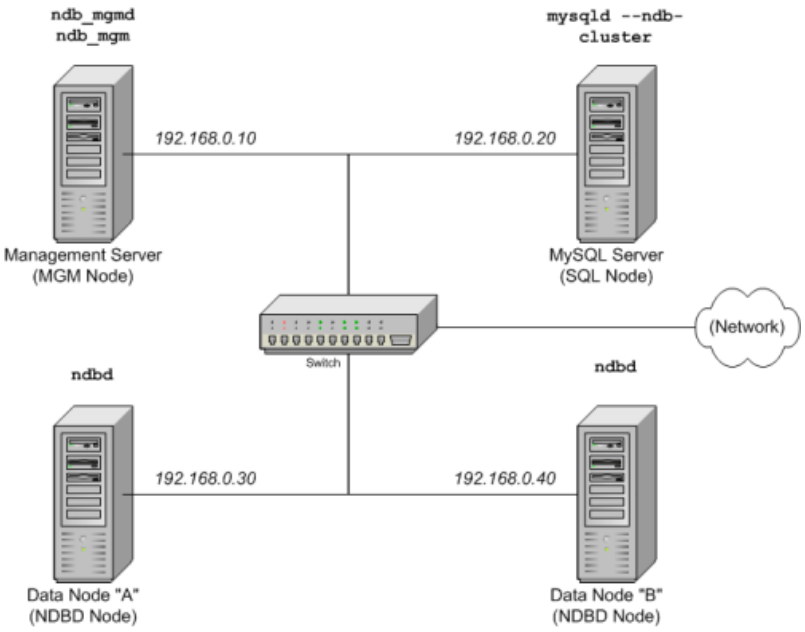
Cluster nodes and host computers. The cluster consists of four nodes, each on a separate host computer, and each with a fixed network address on a typical Ethernet network as shown here:

Table 22.4 Network addresses of nodes in example cluster

Node	IP Address
Management node (`mgmd`)	198.51.100.10
SQL node (`mysqld`)	198.51.100.20
Data node "A" (`ndbd`)	198.51.100.30
Data node "B" (`ndbd`)	198.51.100.40

This setup is also shown in the following diagram:

Figure 22.4 NDB Cluster Multi-Computer Setup



Network addressing. In the interest of simplicity (and reliability), this *How-To* uses only numeric IP addresses. However, if DNS resolution is available on your network, it is possible to use host names in lieu of IP addresses in configuring Cluster. Alternatively, you can use the `hosts` file (typically `/etc/hosts` for Linux and other Unix-like operating systems, `C:\WINDOWS\system32\drivers\etc\hosts` on Windows, or your operating system's equivalent) for providing a means to do host lookup if such is available.

Potential hosts file issues. A common problem when trying to use host names for Cluster nodes arises because of the way in which some operating systems (including some Linux distributions) set up the system's own host name in the `/etc/hosts` during installation. Consider two machines with the host names `ndb1` and `ndb2`, both in the `cluster` network domain. Red Hat Linux (including some derivatives such as CentOS and Fedora) places the following entries in these machines' `/etc/hosts` files:

```
#  ndb1 /etc/hosts:
127.0.0.1   ndb1.cluster ndb1 localhost.localdomain localhost
```

```
#  ndb2 /etc/hosts:
127.0.0.1   ndb2.cluster ndb2 localhost.localdomain localhost
```

SUSE Linux (including OpenSUSE) places these entries in the machines' `/etc/hosts` files:

```
#  ndb1 /etc/hosts:
127.0.0.1       localhost
127.0.0.2       ndb1.cluster ndb1
```

```
#  ndb2 /etc/hosts:
127.0.0.1       localhost
127.0.0.2       ndb2.cluster ndb2
```

In both instances, `ndb1` routes `ndb1.cluster` to a loopback IP address, but gets a public IP address from DNS for `ndb2.cluster`, while `ndb2` routes `ndb2.cluster` to a loopback address and obtains a public address for `ndb1.cluster`. The result is that each data node connects to the management server, but cannot tell when any other data nodes have connected, and so the data nodes appear to hang while starting.

Caution

You cannot mix `localhost` and other host names or IP addresses in `config.ini`. For these reasons, the solution in such cases (other than to use IP addresses for *all* `config.ini` `HostName` entries) is to remove the fully qualified host names from `/etc/hosts` and use these in `config.ini` for all cluster hosts.

Host computer type. Each host computer in our installation scenario is an Intel-based desktop PC running a supported operating system installed to disk in a standard configuration, and running no unnecessary services. The core operating system with standard TCP/IP networking capabilities should be sufficient. Also for the sake of simplicity, we also assume that the file systems on all hosts are set up identically. In the event that they are not, you should adapt these instructions accordingly.

Network hardware. Standard 100 Mbps or 1 gigabit Ethernet cards are installed on each machine, along with the proper drivers for the cards, and that all four hosts are connected through a standard-issue Ethernet networking appliance such as a switch. (All machines should use network cards with the same throughput. That is, all four machines in the cluster should have 100 Mbps cards *or* all four machines should have 1 Gbps cards.) NDB Cluster works in a 100 Mbps network; however, gigabit Ethernet provides better performance.

Important

NDB Cluster is *not* intended for use in a network for which throughput is less than 100 Mbps or which experiences a high degree of latency. For this reason (among others), attempting to run an NDB Cluster over a wide area network such as the Internet is not likely to be successful, and is not supported in production.

Sample data. We use the `world` database which is available for download from the MySQL website (see https://dev.mysql.com/doc/index-other.html). We assume that each machine has sufficient memory for running the operating system, required NDB Cluster processes, and (on the data nodes) storing the database.

For general information about installing MySQL, see Chapter 2, *Installing and Upgrading MySQL*. For information about installation of NDB Cluster on Linux and other Unix-like operating systems, see Section 22.2.2, “Installation of NDB Cluster on Linux”. For information about installation of NDB Cluster on Windows operating systems, see Section 22.2.3, “Installing NDB Cluster on Windows”.

For general information about NDB Cluster hardware, software, and networking requirements, see Section 22.1.3, “NDB Cluster Hardware, Software, and Networking Requirements”.

22.2.1 The NDB Cluster Auto-Installer

This section describes the web-based graphical configuration installer included as part of the NDB Cluster distribution. Topics discussed include an overview of the installer and its parts, software and other requirements for running the installer, navigating the GUI, and using the installer to set up and start or stop an NDB Cluster on one or more host computers.

The NDB Cluster Auto-Installer is made up of two components. The front end is a GUI client implemented as a Web page that loads and runs in a standard Web browser such as Firefox or Microsoft Internet Explorer. The back end is a server process (`ndb_setup.py`) that runs on the local machine or on another host to which you have access.

These two components (client and server) communicate with each other using standard HTTP requests and responses. The back end can manage NDB Cluster software programs on any host where the back end user has granted access. If the NDB Cluster software is on a different host, the back end relies on SSH for access, using the Paramiko library for executing commands remotely (see Section 22.2.1.1, “NDB Cluster Auto-Installer Requirements”).

22.2.1.1 NDB Cluster Auto-Installer Requirements

This section provides information on supported operating platforms and software, required software, and other prerequisites for running the NDB Cluster Auto-Installer.

Supported platforms. The NDB Cluster Auto-Installer is available with NDB 8.0 distributions for recent versions of Linux, Windows, Solaris, and MacOS X. For more detailed information about platform support for NDB Cluster and the NDB Cluster Auto-Installer, see https://www.mysql.com/support/supportedplatforms/cluster.html.

Supported web browsers. The web-based installer is supported with recent versions of Firefox and Microsoft Internet Explorer. It should also work with recent versions of Opera, Safari, and Chrome, although we have not thoroughly tested for compability with these browsers.

Required software—setup host. The following software must be installed on the host where the Auto-Installer is run:

- **Python 2.6 or higher.** The Auto-Installer requires the Python interpreter and standard libraries. If these are not already installed on the system, you may be able to add them using the system's package manager. Otherwise, you can download them from http://python.org/download/.
- **Paramiko 2 or higher.** This is required to communicate with remote hosts using SSH. You can download it from http://www.lag.net/paramiko/. Paramiko may also be available from your system's package manager.

- **Pycrypto version 1.9 or higher.** This cryptography module is required by Paramiko, and can be iunstalled using `pip install cryptography`. If `pip` is not installed, and the module is not available using your system's package manage, you can download it from https://www.dlitz.net/software/pycrypto/.

All of the software in the preceding list is included in the Windows version of the configuration tool, and does not need to be installed separately.

The Paramiko and Pycrypto libraries are required only if you intend to deploy NDB Cluster nodes on remote hosts, and are not needed if all nodes are on the same host where the installer is run.

Required software—remote hosts. The only software required for remote hosts where you wish to deploy NDB Cluster nodes is the SSH server, which is usually installed by default on Linux and Solaris systems. Several alternatives are available for Windows; for an overview of these, see http://en.wikipedia.org/wiki/Comparison_of_SSH_servers.

An additional requirement when using multiple hosts is that it is possible to authenticate to any of the remote hosts using SSH and the proper keys or user credentials, as discussed in the next few paragraphs:

Authentication and security. Three basic security or authentication mechanisms for remote access are available to the Auto-Installer, which we list and describe here:

- **SSH.** A secure shell connection is used to enable the back end to perform actions on remote hosts. For this reason, an SSH server must be running on the remote host. In addition, the operating system user running the installer must have access to the remote server, either with a user name and password, or by using public and private keys.

  Important

  You should never use the system `root` account for remote access, as this is extremely insecure. In addition, `mysqld` cannot normally be started by system `root`. For these and other reasons, you should provide SSH credentials for a regular user account on the target system, and not for system `root`. For more information about this issue, see Section 6.1.5, “How to Run MySQL as a Normal User”.

- **HTTPS.** Remote communication between the Web browser front end and the back end is not encrypted by default, which means that information such as the user's SSH password is transmitted as cleartext that is readable to anyone. For communication from a remote client to be encrypted, the back end must have a certificate, and the front end must communicate with the back end using HTTPS rather than HTTP. Enabling HTTPS is accomplished most easily through issuing a self-signed certificate. Once the certificate is issued, you must make sure that it is used. You can do this by starting `ndb_setup.py` from the command line with the `--use-https` (`-S`) and `--cert-file` (`-c`) options.

  A sample certificate file `cfg.pem` is included and is used by default. This file is located in the `mcc` directory under the installation share directory; on Linux, the full path to the file is normally `/usr/share/mysql/mcc/cfg.pem`. On Windows systems, this is usually `C:\Program Files\MySQL\MySQL Server 8.0\share\mcc\cfg.pem`. Letting the default be used means that, for testing purposes, you can simply start the installer with the `-S` option to use an HTTPS connection between the browser and the back end.

  The Auto-Installer saves the configuration file for a given cluster `mycluster01` as `mycluster01.mcc` in the home directory of the user invoking the `ndb_setup.py` executable. This file is encrypted with a passphrase supplied by the user (using Fernet); because HTTP transmits the passphrase in the clear, *it is strongly recommended that you always use an HTTPS connection to access the Auto-Installer on a remote host*.

- **Certificate-based authentication.** The back end `ndb_setup.py` process can execute commands on the local host as well as remote hosts. This means that anyone connecting to the back end can take charge of how commands are executed. To reject unwanted connections to the back end, a certificate may be required for authentication of the client. In this case, a certificate must be issued by the user, installed in the browser, and made available to the back end for authentication purposes. You can enact this requirement (together with or in place of password or key authentication) by starting `ndb_setup.py` with the `--ca-certs-file` (`-a`) option.

There is no need or requirement for secure authentication when the client browser is running on the same host as the Auto-Installer back end.

See also Section 22.5.12, “NDB Cluster Security Issues”, which discusses security considerations to take into account when deploying NDB Cluster, as well as Chapter 6, *Security*, for more general MySQL security information.

22.2.1.2 Using the NDB Cluster Auto-Installer

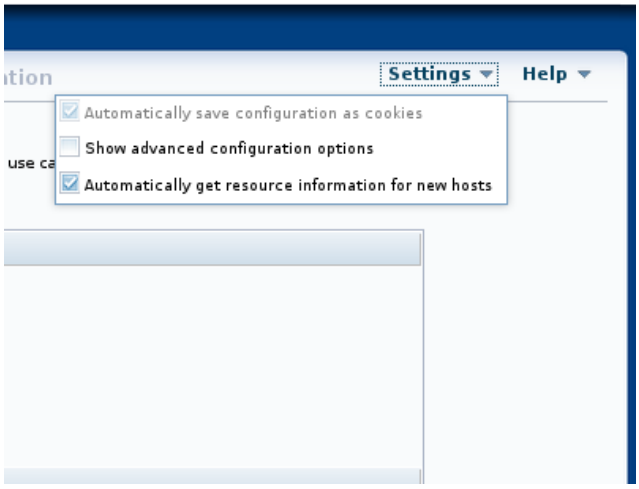
The NDB Cluster Auto-Installer interface is made up of several pages, each corresponding to a step in the process used to configure and deploy an NDB Cluster. These pages are listed here, in order:

- **Welcome**: Begin using the Auto-Installer by choosing either to configure a new NDB Cluster, or to continue configuring an existing one.
- **Define Cluster**: Set basic information about the cluster as a whole, such as name, hosts, and load type. Here you can also set the SSH authentication type for accessing remote hosts, if needed.
- **Define Hosts**: Identify the hosts where you intend to run NDB Cluster processes.
- **Define Processes**: Assign one or more processes of a given type or types to each cluster host.
- **Define Parameters**: Set configuration attributes for processes or types of processes.
- **Deploy Configuration**: Deploy the cluster with the configuration set previously; start and stop the deployed cluster.

NDB Cluster Installer Settings and Help Menus

These menus are shown on all screens except for the **Welcome** screen. They provide access to installer settings and information. The **Settings** menu is shown here in more detail:

Figure 22.5 NDB Cluster Auto-Installer Settings menu



The **Settings** menu has the following entries:

- **Automatically save configuration as cookies**: Save your configuration information—such as host names, process data, and parameter values—as a cookie in the browser. When this option is chosen, all information except any SSH password is saved. This means that you can quit and restart the browser, and continue working on the same configuration from where you left off at the end of the previous session. This option is enabled by default.

  The SSH password is never saved; if you use one, you must supply it at the beginning of each new session.

- **Show advanced configuration options**: Shows by default advanced configuration parameters where available.

  Once set, the advanced parameters continue to be used in the configuration file until they are explicitly changed or reset. This is regardless of whether the advanced parameters are currently visible in the installer; in other words, disabling the menu item does not reset the values of any of these parameters.

  You can also toggle the display of advanced parameters for individual processes on the **Define Parameters** screen.

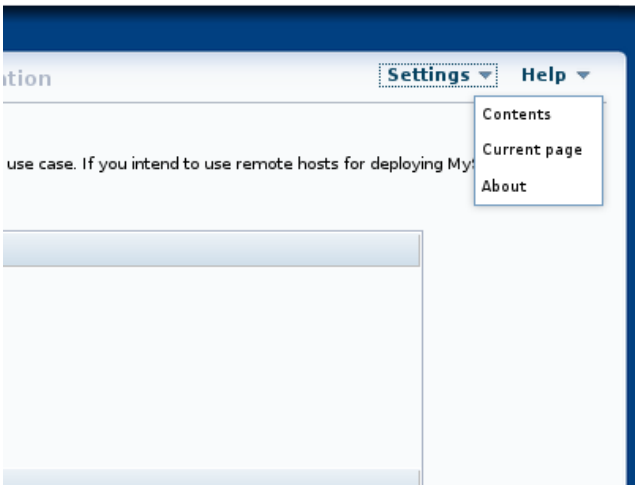
  This option is disabled by default.

- **Automatically get resource information for new hosts**: Query new hosts automatically for hardware resource information to pre-populate a number of configuration options and values. In this case, the suggested values are not mandatory, but they are used unless explicitly changed using the appropriate editing options in the installer.

  This option is enabled by default.

The installer **Help** menu is shown here:

Figure 22.6 NDB Cluster Auto-Installer Help menu



The **Help** menu provides several options, described in the following list:

- **Contents**: Show the built-in user guide. This is opened in a separate browser window, so that it can be used simultaneously with the installer without interrupting workflow.

- **Current page**: Open the built-in user guide to the section describing the page currently displayed in the installer.

- **About**: open a dialog displaying the installer name and the version number of the NDB Cluster distribution with which it was supplied.

The Auto-Installer also provides context-sensitive help in the form of tooltips for most input widgets.

In addition, the names of most NDB configuration parameters are linked to their descriptions in the online documentation. The documentation is displayed in a separate browser window.

The next section discusses starting the Auto-Installer. The sections immediately following it describe in greater detail the purpose and function of each of these pages in the order listed previously.

Starting the NDB Cluster Auto-Installer

The Auto-Installer is provided together with the NDB Cluster software. Separate RPM and `.deb` packages containing only the Auto-Installer are also available for many Linux distributions. (See Section 22.2, “NDB Cluster Installation”.)

The present section explains how to start the installer. You can do by invoking the `ndb_setup.py` executable.

User and privileges

You should run the `ndb_setup.py` as a normal user; no special privileges are needed to do so. You should *not* run this program as the `mysql` user, or using the system `root` or Administrator account; doing so may cause the installation to fail.

`ndb_setup.py` is found in the `bin` within the NDB Cluster installation directory; a typical location might be `/usr/local/mysql/bin` on a Linux system or `C:\Program Files\MySQL\MySQL Server 8.0\bin` on a Windows system. This can vary according to where the NDB Cluster software is installed on your system, and the installation method.

On Windows, you can also start the installer by running `setup.bat` in the NDB Cluster installation directory. When invoked from the command line, this batch file accepts the same options as `ndb_setup.py`.

`ndb_setup.py` can be started with any of several options that affect its operation, but it is usually sufficient to allow the default settings be used, in which case you can start `ndb_setup.py` by either of the following two methods:

1. Navigate to the NDB Cluster `bin` directory in a terminal and invoke it from the command line, without any additional arguments or options, like this:

   ```
   shell> ndb_setup.py
   Running out of install dir: /usr/local/mysql/bin
   Starting web server on port 8081
   URL is https://localhost:8081/welcome.html
   deathkey=627876
   Press CTRL+C to stop web server.
   The application should now be running in your browser.
   (Alternatively you can navigate to https://localhost:8081/welcome.html to start it)
   ```

   This works regardless of operating platform.

2. Navigate to the NDB Cluster `bin` directory in a file browser (such as Windows Explorer on Windows, or Konqueror, Dolphin, or Nautilus on Linux) and activate (usually by double-clicking) the **ndb_setup.py** file icon. This works on Windows, and should work with most common Linux desktops as well.

   On Windows, you can also navigate to the NDB Cluster installation directory and activate the **setup.bat** file icon.

In either case, once `ndb_setup.py` is invoked, the Auto-Installer's **Welcome** screen should open in the system's default web browser. If not, you should be able to open the page `http://localhost:8081/welcome.html` or `https://localhost:8081/welcome.html` manually in the browser.

In some cases, you may wish to use non-default settings for the installer, such as specifying HTTPS for connections, or a different port for the Auto-Installer's included web server to run on, in which case you must invoke `ndb_setup.py` with one or more startup options with values overriding the necessary defaults. The same startup options can be used on Windows systems with the `setup.bat` file supplied for such platforms in the NDB Cluster software distribution. This can be done using the command line, but if you want or need to start the installer from a desktop or file browser while employing one or more of these options, it is also possible to create a script or batch file containing the proper invocation, then to double-click its file icon in the file browser to start the installer. (On Linux systems, you might also need to make the script file executable first.) If you plan to use the Auto-Installer from a remote host, you should start using the `-S` option. For information about this and other advanced startup options for the NDB Cluster Auto-Installer, see Section 22.4.26, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster”.

NDB Cluster Auto-Installer Welcome Screen

The **Welcome** screen is loaded in the default browser when `ndb_setup.py` is invoked. The first time the Auto-Installer is run (or if for some other reason there are no existing configurations), this screen appears as shown here:

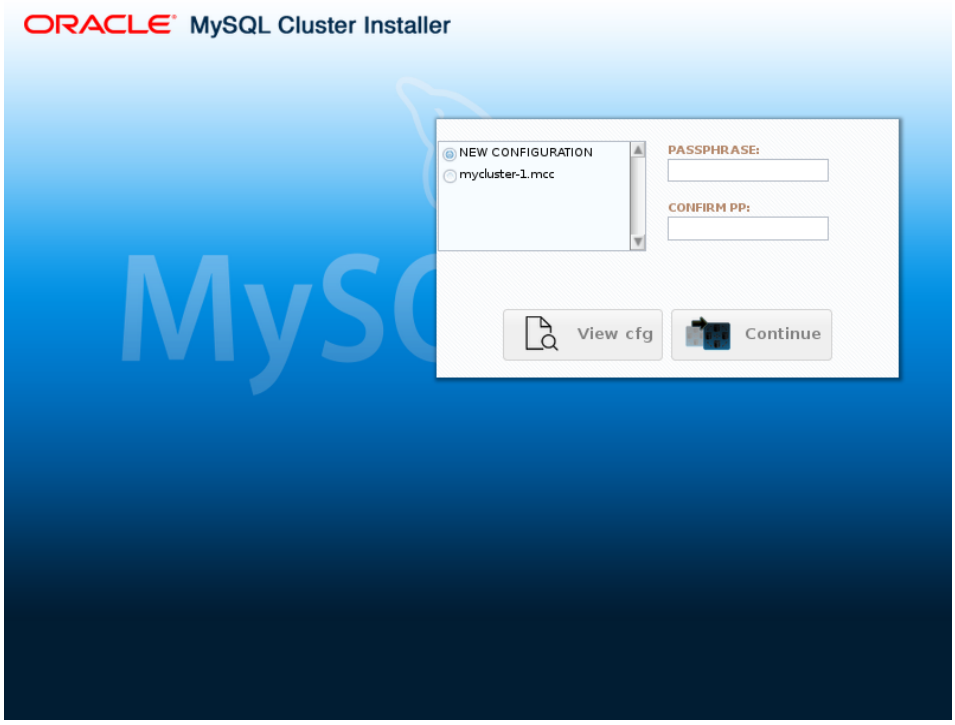
Figure 22.7 The NDB Cluster Auto-Installer Welcome screen, first run

In this case, the only choice of cluster listed is for configuration of a new cluster, and both the **View Cfg** and **Continue** buttons are inactive.

To create a new configuration, enter and confirm a passphrase in the text boxes provided. When this has been done, you can click **Continue** to proceed to the **Define Cluster** screen where you can assign a name to the new cluster.

If you have previously created one or more clusters with the Auto-Installer, they are listed by name. This example shows an existing cluster named `mycluster-1`:

Figure 22.8 The NDB Cluster Auto-Installer Welcome screen, with previously created cluster mycluster-1

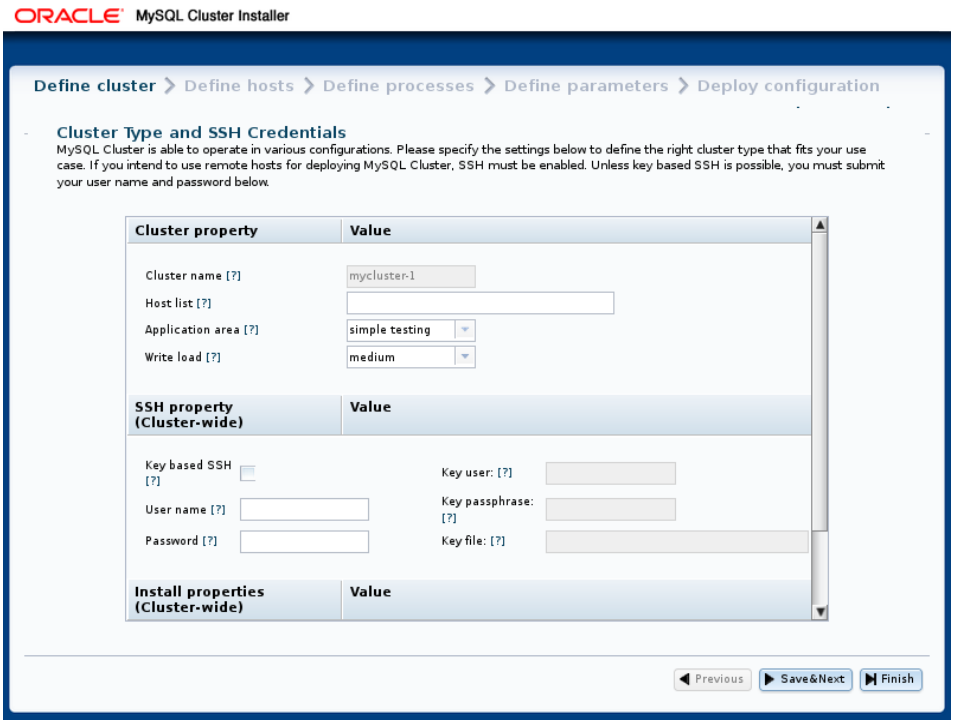


To view the configuration for and work with a given cluster, select the radiobutton next to its name in the list, then enter and confirm the passphrase that was used to create it. When you have done this correctly, you can click **View Cfg** to view and edit this cluster's configuration.

NDB Cluster Auto-Installer Define Cluster Screen

The **Define Cluster** screen is appears following the **Welcome** screen, and is used for setting general properties of the cluster. The layout of the **Define Cluster** screen is shown here:

Figure 22.9 The NDB Cluster Auto-Installer Define Cluster screen



This screen and subsequent screens also include **Settings** and **Help** menus which are described later in this section; see NDB Cluster Installer Settings and Help Menus.

The **Define Cluster** screen allows you to set three sorts of properties for the cluster: cluster properties, SSH properties, and installation properties.

Cluster properties that can be set on this screen are listed here:

- **Cluster name**: A name that identifies the cluster; in this example, this is `mycluster-1`. The name is set on the previous screen and cannot be changed here.
- **Host list**: A comma-delimited list of one or more hosts where cluster processes should run. By default, this is `127.0.0.1`. If you add remote hosts to the list, you must be able to connect to them using the credentials supplied as SSH properties.
- **Application type**: Choose one of the following:

  1. **Simple testing**: Minimal resource usage for small-scale testing. This the default. *Not intended for production environments*.
  2. **Web**: Maximize performance for the given hardware.
  3. **Real-time**: Maximize performance while maximizing sensitivity to timeouts in order to minimize the time needed to detect failed cluster processes.

- **Write load**: Choose a level for the anticipated number of writes for the cluster as a whole. You can choose any one of the following levels:

  1. **Low**: The expected load includes fewer than 100 write transactions for second.
  2. **Medium**: The expected load includes 100 to 1000 write transactions per second; this is the default.

3. **High**: The expected load includes more than 1000 write transactions per second.

SSH properties are described in the following list:

- **Key-Based SSH**: Check this box to use key-enabled login to the remote host. If checked, the key user and passphrase must also be supplied; otherwise, a user and password for a remote login account are needed.
- **User**: Name of user with remote login access.
- **Password**: Password for remote user.
- **Key user**: Name of the user for whom the key is valid, if not the same as the operating system user.
- **Key passphrase**: Passphrase for the key, if required.
- **Key file**: Path to the key file. The default is `~/.ssh/id_rsa`.

The SSH properties set on this page apply to all hosts in the cluster. They can be overridden for a given host by editing that hosts's properties on the **Define Hosts** screen.

Two installation properties can also be set on this screen:

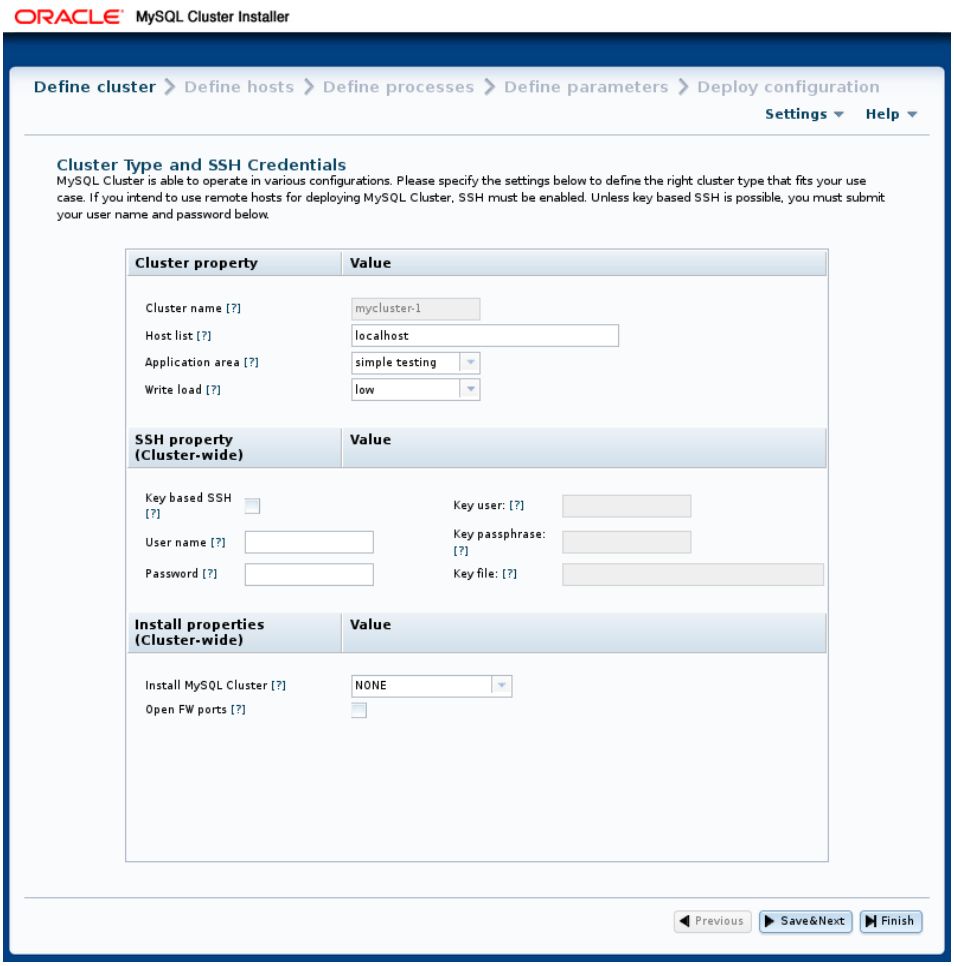
- **Install MySQL Cluster**: This setting determines the source from which the Auto-Installer installs NDB Cluster software, if any, on the cluster hosts. Possible values and their effects are listed here:

  1. `DOCKER`: Try to install the MySQL Cluster Docker image from `https://hub.docker.com/r/mysql/mysql-cluster/` on each host
  2. `REPO`: Try to install the NDB Cluster software from the MySQL Repositories on each host
  3. `BOTH`: Try to install either the Docker image or the software from the repository on each host, giving preference to the repository
  4. `NONE`: Do not install the NDB Cluster software on the hosts; this is the default

- **Open FW Ports**: Check this checkbox to have the installer attempt to open ports required by NDB CLuster processes on all hosts.

The next figure shows the **Define Cluster** page with settings for a small test cluster with all nodes running on `localhost`:

Figure 22.10 The NDB Cluster Auto-Installer Define Cluster screen, with settings for a test cluster



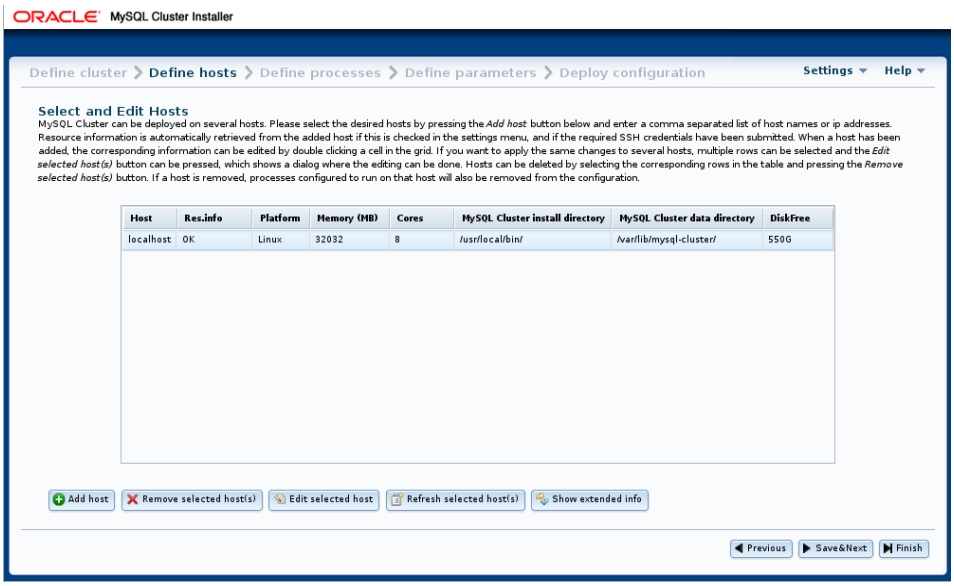
After making the desired settings, you can save them to the configuration file and proceed to the **Define Hosts** screen by clicking the **Save & Next** button.

If you exit the installer without saving, no changes are made to the configuration file.

NDB Cluster Auto-Installer Define Hosts Screen

The **Define Hosts** screen, shown here, provides a means of viewing and specifying several key properties of each cluster host:

Figure 22.11 NDB Cluster Define Hosts screen, start



Properties shown include the following:

- **Host**: Name or IP address of this host
- **Res.info**: Shows `OK` if the installer was able to retrieve requested resource information from this host
- **Platform**: Operating system or platform
- **Memory (MB)**: Amount of RAM on this host
- **Cores**: Number of CPU cores available on this host
- **MySQL Cluster install directory**: Path to directory where the NDB Cluster software is installed on this host; defaults to `/usr/local/bin`
- **MySQL Cluster data directory**: Path to directory used for data by NDB Cluster processes on this host; defaults to `/var/lib/mysql-cluster`.
- **DiskFree**: Free disk space in bytes

  For hosts with multiple disks, only the space available on the disk used for the data directory is shown.

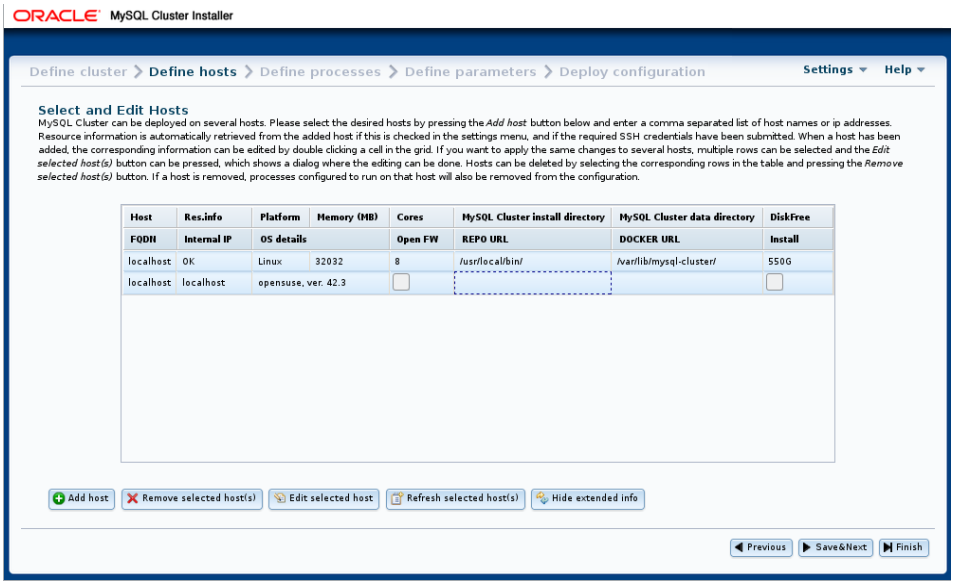
This screen also provides an extended view for each host that includes the following properties:

- **FDQN**: This host's fully qualified domain name, used by the installer to connect with it, distribute configuration information to it, and start and stop cluster processes on it.
- **Internal IP**: The IP address used for communication with cluster processes running on this host by processes running elsewhere.
- **OS Details**: Detailed operating system name and version information.
- **Open FW**: If this checkbox is enabled, the installer attempts to open ports in the host's firewall needed by cluster processes.
- **REPO URL**: URL for MySQL NDB Cluster repository

- **DOCKER URL**: URL for MySQL NDB CLuster Docker images; for NDB 8.0, this is `mysql/mysql-cluster:8.0`.
- **Install**: If this checkbox is enabled, the Auto-Installer attempts to install the NDB Cluster software on this host

The extended view is shown here:

Figure 22.12 NDB Cluster Define Hosts screen, extended host info view



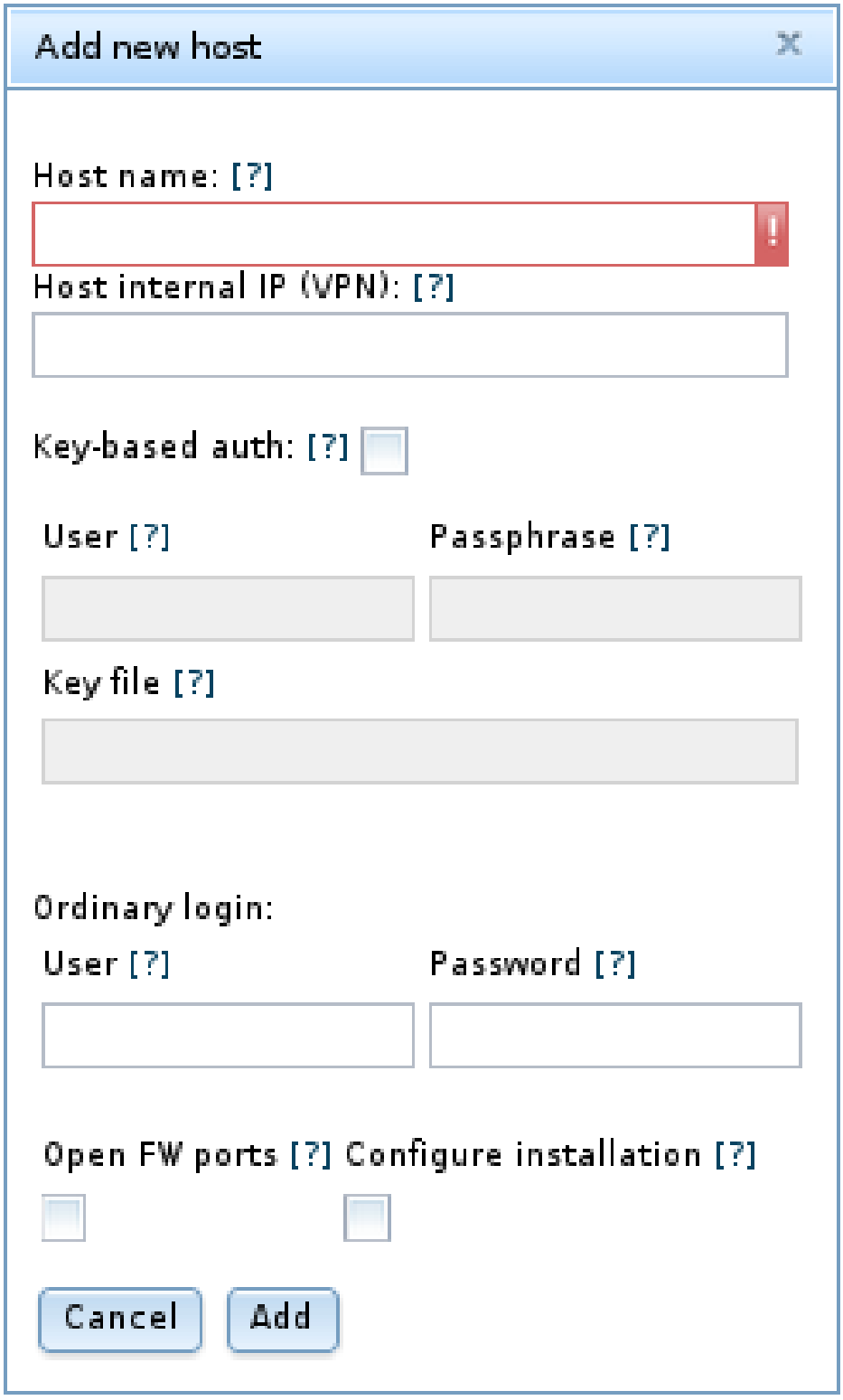
All cells in the display are editable, with the exceptions of those in the **Host**, **Res.info**, and **FQDN** columns.

Be aware that it may take some time for information to be retrieved from remote hosts. Fields for which no value could be retrieved are indicated with an ellipsis (...). You can retry the fetching of resource information from one or more hosts by selecting the hosts in the list and then clicking the **Refresh selected host(s)** button.

Adding and Removing Hosts

You can add one or more hosts by clicking the **Add Host** button and entering the required properties where indicated in the **Add new host** dialog, shown here:

Figure 22.13 NDB Cluster Add Host dialog



This dialog includes the following fields:

- **Host name**: A comma-separated list of one or more host names, IP addresses, or both. These must be accessible from the host where the Auto-Installer is running.
- **Host internal IP (VPN)**: If you are setting up the cluster to run on a VPN or other internal network, enter the IP address or addresses used for contact by cluster nodes on other hosts.
- **Key-based auth**: If checked, enables key-based authentication. You can enter any additional needed information in the **User**, **Passphrase**, and **Key file** fields.
- **Ordinary login**: If accessing this host using a password-based login, enter the appropriate information in the **User** and **Password** fields.
- **Open FW ports**: Selecting this checkbox allows the installer try opening any ports needed by cluster processes in this host's firewall.
- **Configure installation**: Checking this allows the Auto-Install to attempt to set up the NDB Cluster software on this host.

To save the new host and its properties, click **Add**. If you wish to cancel without saving any changes, click **Cancel** instead.

Similarly, you can remove one or more hosts using the button labelled **Remove selected host(s)**. *When you remove a host, any process which was configured for that host is also removed*.

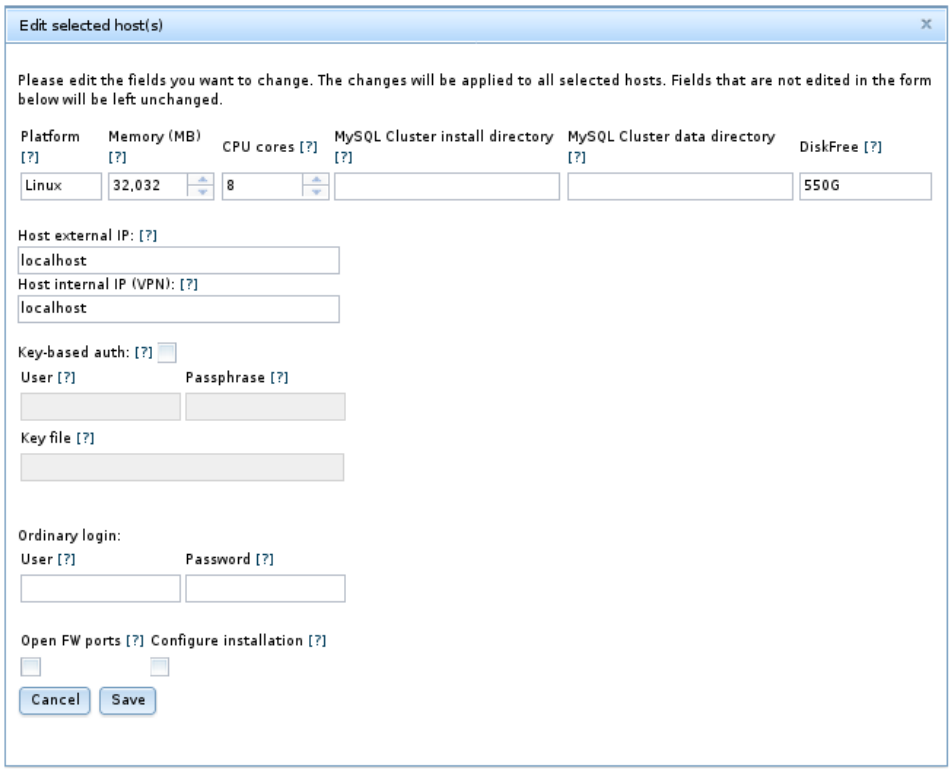
Warning

Remove selected host(s) acts immediately. There is no confirmation dialog. If you remove a host in error, you must re-enter its name and properties manually using **Add host**.

If the SSH user credentials on the *Define Cluster* screen are changed, the Auto-Installer attempts to refresh the resource information from any hosts for which information is missing.

You can edit the host's platform name, hardware resource information, installation directory, and data directory by clicking the corresponding cell in the grid, by selecting one or more hosts and clicking the button labelled **Edit selected host(s)**. This causes a dialog box to appear, in which these fields can be edited, as shown here:

Figure 22.14 NDB Cluster Auto-Installer Edit Hosts dialog



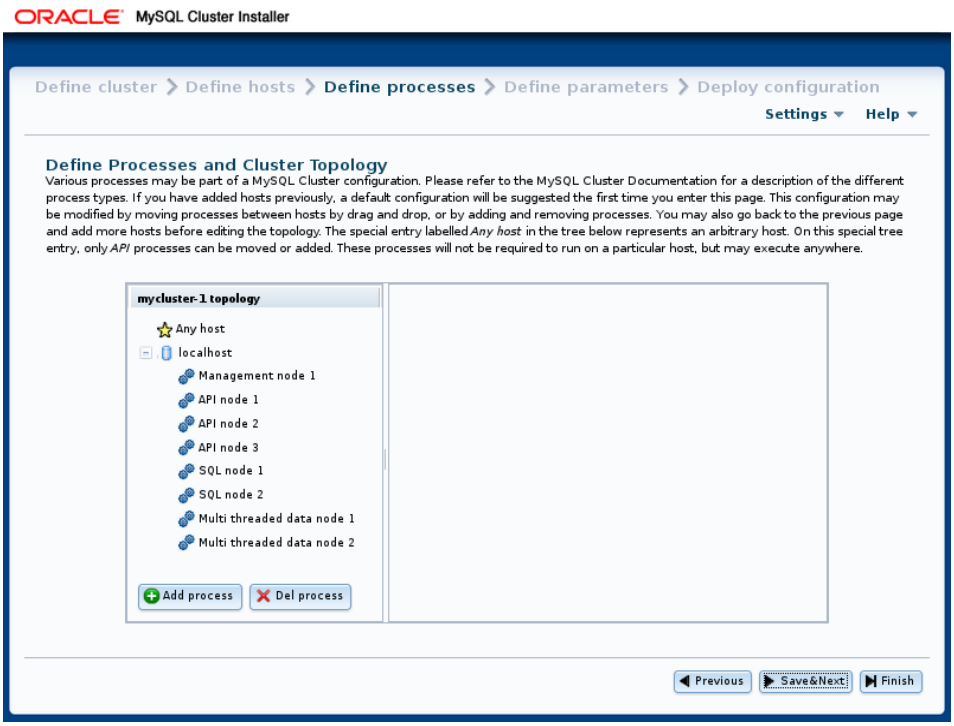
When more than one host is selected, any edited values are applied to all selected hosts.

Once you have entered all desired host information, you can use the **Save & Next** button to save the information to the cluster's configuration file and proceed to the **Define Processes** screen, where you can set up NDB Cluster processes on one or more hosts.

NDB Cluster Auto-Installer Define Processes Screen

The **Define Processes** screen, shown here, provides a way to assign NDB Cluster processes (nodes) to cluster hosts:

Figure 22.15 NDB Cluster Auto-Installer Define Processes dialog



This screen contains a process tree showing cluster hosts and processes set up to run on each one, as well as a panel which displays information about the item currently selected in the tree.

When this screen is accessed for the first time for a given cluster, a default set of processes is defined for you, based on the number of hosts. If you later return to the **Define Hosts** screen, remove all hosts, and add new hosts, this also causes a new default set of processes to be defined.

NDB Cluster processes are of the types described in this list:

- **Management node.** Performs administrative tasks such as stopping individual data nodes, querying node and cluster status, and making backups. Executable: `ndb_mgmd`.
- **Single-threaded data node.** Stores data and executes queries. Executable: `ndbd`.
- **Multi threaded data node.** Stores data and executes queries with multiple worker threads executing in parallel. Executable: `ndbmtd`.
- **SQL node.** MySQL server for executing SQL queries against `NDB`. Executable: `mysqld`.
- **API node.** A client accessing data in `NDB` by means of the NDB API or other low-level client API, rather than by using SQL. See MySQL NDB Cluster API Developer Guide, for more information.

For more information about process (node) types, see Section 22.1.1, “NDB Cluster Core Concepts”.

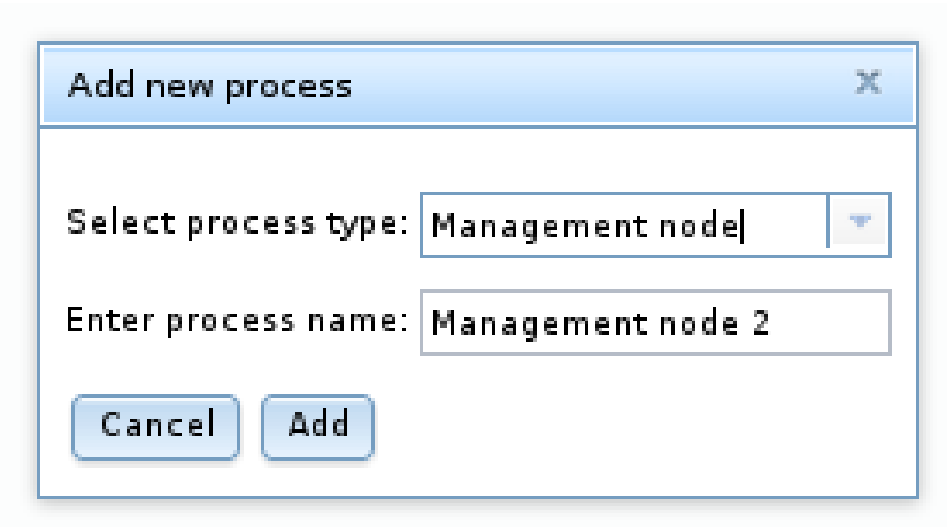
Processes shown in the tree are numbered sequentially by type, for each host—for example, `SQL node 1`, `SQL node 2`, and so on—to simplify identification.

Each management node, data node, or SQL process must be assigned to a specific host, and is not allowed to run on any other host. An API node *may* be assigned to a single host, but this is not required. Instead, you can assign it to the special **Any host** entry which the tree also contains in addition to any

other hosts, and which acts as a placeholder for processes that are allowed to run on any host. *Only API processes may use this **Any host** entry*.

Adding processes. To add a new process to a given host, either right-click that host's entry in the tree, then select the **Add process** popup when it appears, or select a host in the process tree, and press the **Add process** button below the process tree. Performing either of these actions opens the add process dialog, as shown here:

Figure 22.16 NDB Cluster Auto-Installer Add Process Dialog



Here you can select from among the available process types described earlier this section; you can also enter an arbitrary process name to take the place of the suggested value, if desired.

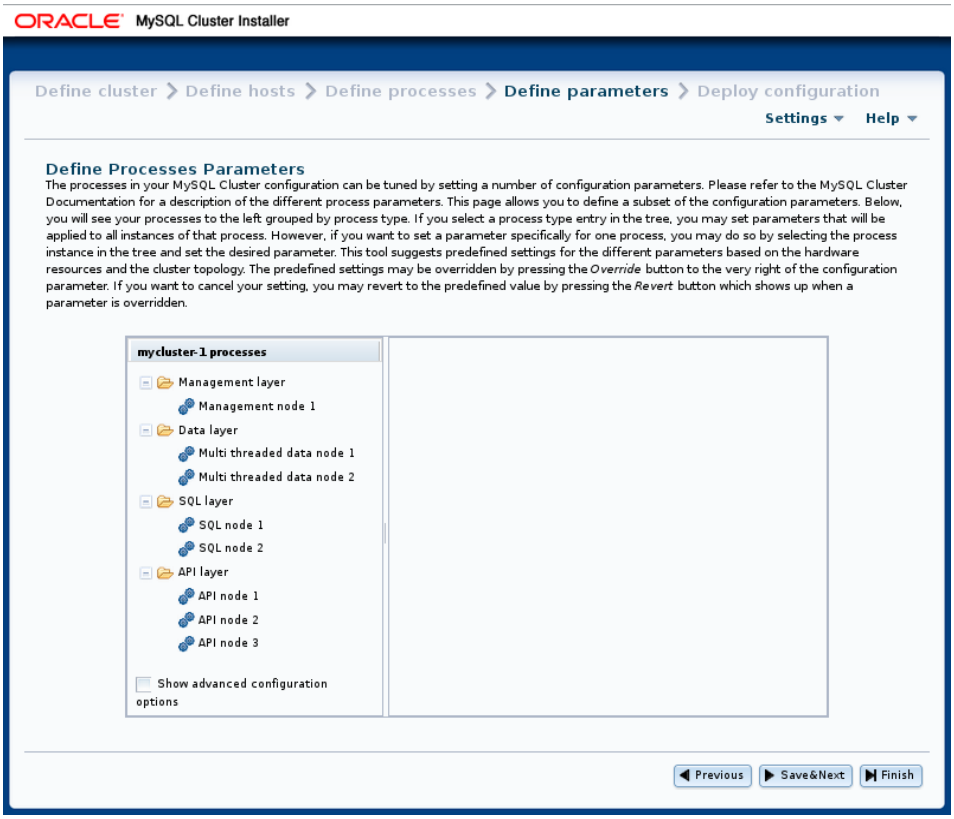
Removing processes. To delete a process, select that process in the tree and use the **Del process** button.

When you select a process in the process tree, information about that process is displayed in the information panel, where you can change the process name and possibly its type. You can change a multi-threaded data node (`ndbmtd`) to a single-threaded data node (`ndbd`), or the reverse, only; no other process type changes are allowed. *If you want to make a change between any other process types, you must delete the original process first, then add a new process of the desired type*.

NDB Cluster Auto-Installer Define Parameters Screen

Like the **Define Processes** screen, this screen includes a process tree; the **Define Parameters** process tree is organized by process or node type, in groups labelled **Management Layer**, **Data Layer**, **SQL Layer**, and **API Layer**. An information panel displays information regarding the item currently selected. The **Define Attributes** screen is shown here:

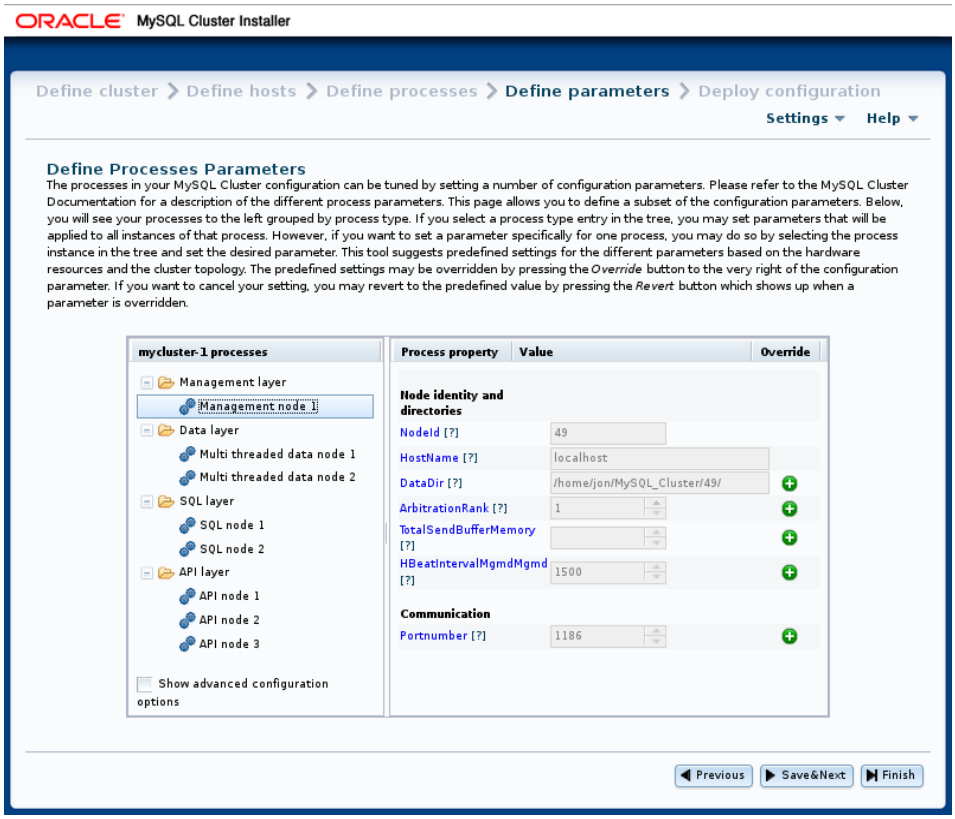
Figure 22.17 NDB Cluster Auto-Installer Define Parameters screen



The checkbox labelled **Show advanced configuration**, when checked, makes advanced options for data node and SQL node processes visible in the information pane. These options are set and used whether or not they are visible. You can also enable this behavior globally by checking **Show advanced configuration options** under **Settings** (see NDB Cluster Installer Settings and Help Menus).

You can edit attributes for a single process by selecting that process from the tree, or for all processes of the same type in the cluster by selecting one of the **Layer** folders. A per-process value set for a given attribute overrides any per-group setting for that attribute that would otherwise apply to the process in question. An example of such an information panel (for an SQL process) is shown here:

Figure 22.18 Define Parameters—Process Attributes



Attributes whose values can be overridden are shown in the information panel with a button bearing a plus sign. This **+** button activates an input widget for the attribute, enabling you to change its value. When the value has been overridden, this button changes into a button showing an **X**. The **X** button undoes any changes made to a given attribute, which immediately reverts to the predefined value.

All configuration attributes have predefined values calculated by the installer, based such factors as host name, node ID, node type, and so on. In most cases, these values may be left as they are. If you are not familiar with it already, it is highly recommended that you read the applicable documentation before making changes to any of the attribute values. To make finding this information easier, each attribute name shown in the information panel is linked to its description in the online NDB Cluster documentation.

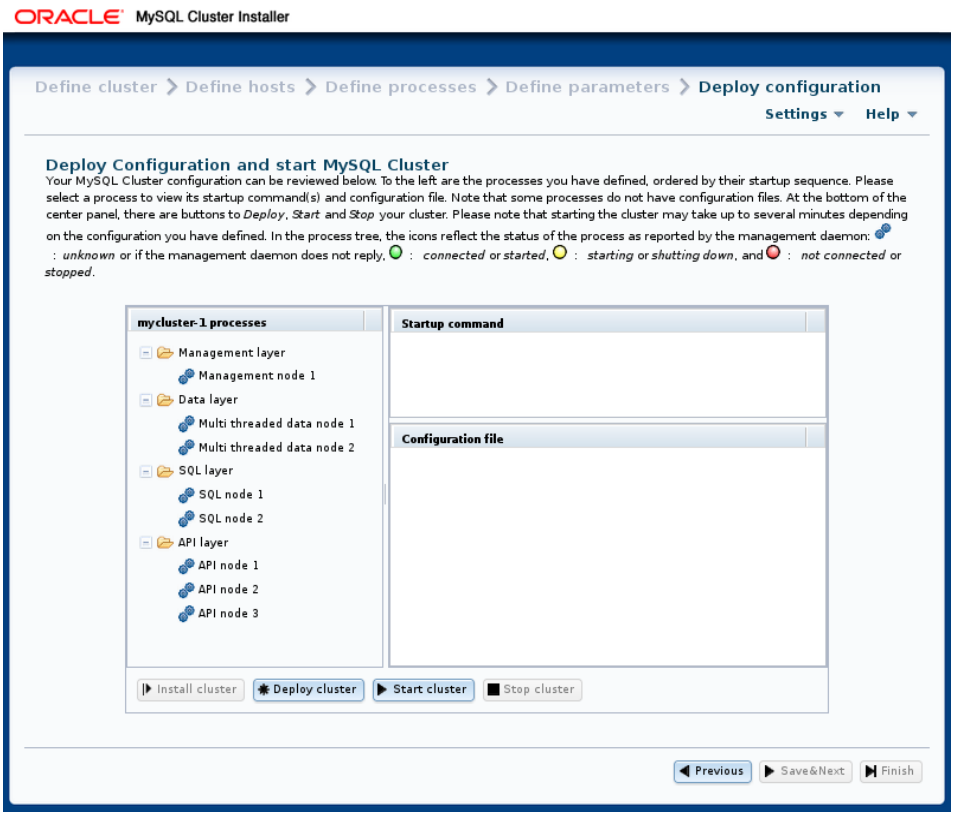
NDB Cluster Auto-Installer Deploy Configuration Screen

This screen allows you to perform the following tasks:

- Review process startup commands and configuration files to be applied
- Distribute configuration files by creating any necessary files and directories on all cluster hosts—that is, *deploy* the cluster as presently configured
- Start and stop the cluster

The **Deploy Configuration** screen is shown here:

Figure 22.19 NDB Cluster Auto-Installer Deploy Configuration screen



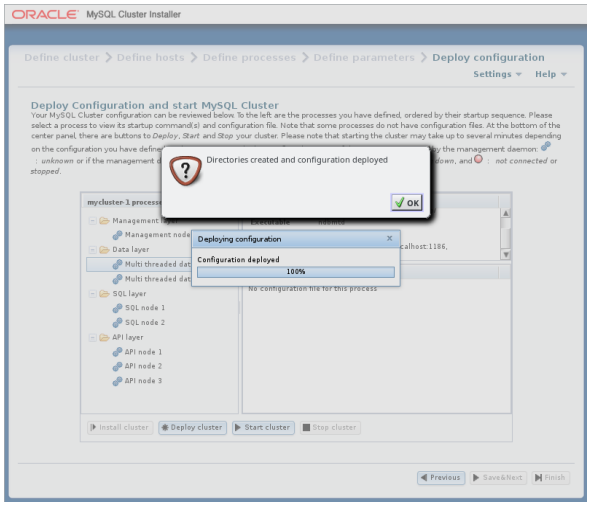
Like the **Define Parameters** screen, this screen features a process tree which is organized by process type. Next to each process in the tree is a status icon indicating the current status of the process: connected (`CONNECTED`), starting (`STARTING`), running (`STARTED`), stopping (`STOPPING`), or disconnected (`NO_CONTACT`). The icon shows green if the process is connected or running; yellow if it is starting or stopping; red if the process is stopped or cannot be contacted by the management server.

This screen also contains two information panels, one showing the startup command or commands needed to start the selected process. (For some processes, more than one command may be required—for example, if initialization is necessary.) The other panel shows the contents of the configuration file, if any, for the given process.

This screen also contains four buttons, labelled as and performing the functions described in the following list:

- **Install cluster**: Nonfunctional in this release; implementation intended for a future release.
- **Deploy cluster**: Verify that the configuration is valid. Create any directories required on the cluster hosts, and distribute the configuration files onto the hosts. A progress bar shows how far the deployment has proceeded, as shown here, and a dialog is pisplayed when the deployment has completed, as shown here:

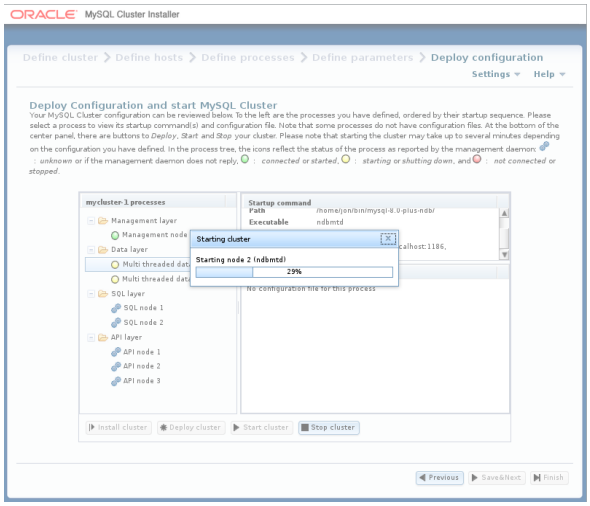
Figure 22.20 Cluster Deployment Process



- **Start cluster**: The cluster is deployed as with **Deploy cluster**, after which all cluster processes are started in the correct order.

  Starting these processes may take some time. If the estimated time to completion is too large, the installer provides an opportunity to cancel or to continue of the startup procedure. A progress bar indicates the current status of the startup procedure, as shown here:

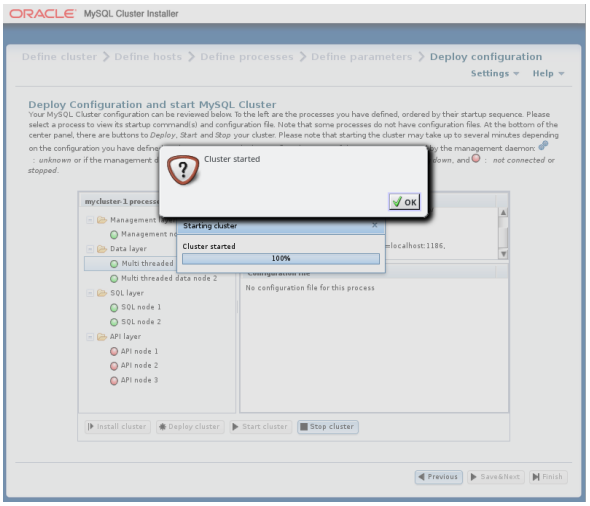
Figure 22.21 Cluster Startup Process with Progress Bar



The process status icons next to the items shown in the process tree also update with the status of each process.

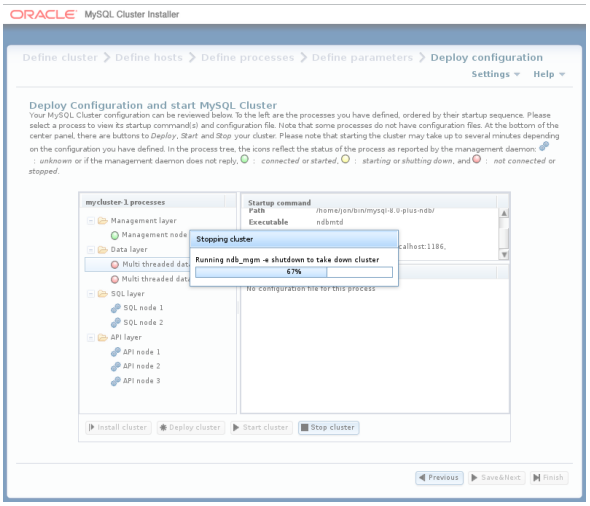
A confirmation dialog is shown when the startup process has completed, as shown here:

Figure 22.22 Cluster Startup, Process Completed Dialog



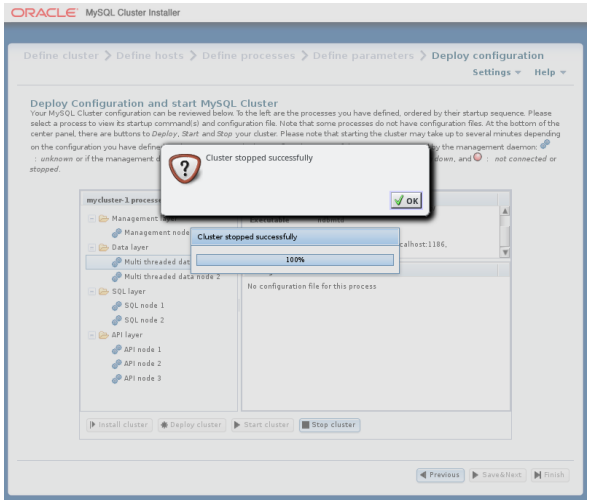
- **Stop cluster**: After the cluster has been started, you can stop it using this. As with starting the cluster, cluster shutdown is not instantaneous, and may require some time complete. A progress bar, similar to that displayed during cluster startup, shows the approximate current status of the cluster shutdown procedure, as do the process status icons adjoining the process tree. The progress bar is shown here:

Figure 22.23 Cluster Shutdown Process, with Progress Bar



A confirmation dialog indicates when the shutdown process is complete:

Figure 22.24 Cluster Shutdown, Process Completed Dialog



The Auto-Installer generates a `config.ini` file containing NDB node parameters for each management node, as well as a `my.cnf` file containing the appropriate options for each `mysqld` process in the cluster. No configuration files are created for data nodes or API nodes.

22.2.2 Installation of NDB Cluster on Linux

This section covers installation methods for NDB Cluster on Linux and other Unix-like operating systems. While the next few sections refer to a Linux operating system, the instructions and procedures given there should be easily adaptable to other supported Unix-like platforms. For manual installation and setup instructions specific to Windows systems, see Section 22.2.3, “Installing NDB Cluster on Windows”.

Each NDB Cluster host computer must have the correct executable programs installed. A host running an SQL node must have installed on it a MySQL Server binary (`mysqld`). Management nodes require the management server daemon (`ndb_mgmd`); data nodes require the data node daemon (`ndbd` or `ndbmtd`). It is not necessary to install the MySQL Server binary on management node hosts and data node hosts. It is recommended that you also install the management client (`ndb_mgm`) on the management server host.

Installation of NDB Cluster on Linux can be done using precompiled binaries from Oracle (downloaded as a .tar.gz archive), with RPM packages (also available from Oracle), or from source code. All three of these installation methods are described in the section that follow.

Regardless of the method used, it is still necessary following installation of the NDB Cluster binaries to create configuration files for all cluster nodes, before you can start the cluster. See Section 22.2.4, “Initial Configuration of NDB Cluster”.

22.2.2.1 Installing an NDB Cluster Binary Release on Linux

This section covers the steps necessary to install the correct executables for each type of Cluster node from precompiled binaries supplied by Oracle.

For setting up a cluster using precompiled binaries, the first step in the installation process for each cluster host is to download the binary archive from the NDB Cluster downloads page. (For the most recent 64-bit NDB 8.0 release, this is `mysql-cluster-gpl-8.0.19-linux-glibc2.12-x86_64.tar.gz`.) We assume that you have placed this file in each machine's `/var/tmp` directory.

If you require a custom binary, see Section 2.9.5, “Installing MySQL Using a Development Source Tree”.

Note

After completing the installation, do not yet start any of the binaries. We show you how to do so following the configuration of the nodes (see Section 22.2.4, “Initial Configuration of NDB Cluster”).

SQL nodes. On each of the machines designated to host SQL nodes, perform the following steps as the system `root` user:

1. Check your `/etc/passwd` and `/etc/group` files (or use whatever tools are provided by your operating system for managing users and groups) to see whether there is already a `mysql` group and `mysql` user on the system. Some OS distributions create these as part of the operating system installation process. If they are not already present, create a new `mysql` user group, and then add a `mysql` user to this group:

   ```
   shell> groupadd mysql
   shell> useradd -g mysql -s /bin/false mysql
   ```

   The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

2. Change location to the directory containing the downloaded file, unpack the archive, and create a symbolic link named `mysql` to the `mysql` directory.

   Note

   The actual file and directory names vary according to the NDB Cluster version number.

   ```
   shell> cd /var/tmp
   shell> tar -C /usr/local -xzvf mysql-cluster-gpl-8.0.19-linux-glibc2.12-x86_64.tar.gz
   shell> ln -s /usr/local/mysql-cluster-gpl-8.0.19-linux-glibc2.12-x86_64 /usr/local/mysql
   ```

3. Change location to the `mysql` directory and set up the system databases using `mysqld --initialize` as shown here:

   ```
   shell> cd mysql
   shell> mysqld --initialize
   ```

   This generates a random password for the MySQL `root` account. If you do *not* want the random password to be generated, you can substitute the `--initialize-insecure` option for `--initialize`. In either case, you should review Section 2.10.1, “Initializing the Data Directory”, for additional information before performing this step. See also Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”.

4. Set the necessary permissions for the MySQL server and data directories:

   ```
   shell> chown -R root .
   shell> chown -R mysql data
   shell> chgrp -R mysql .
   ```

5. Copy the MySQL startup script to the appropriate directory, make it executable, and set it to start when the operating system is booted up:

   ```
   shell> cp support-files/mysql.server /etc/rc.d/init.d/
   ```

```
shell> chmod +x /etc/rc.d/init.d/mysql.server
shell> chkconfig --add mysql.server
```

(The startup scripts directory may vary depending on your operating system and version—for example, in some Linux distributions, it is `/etc/init.d`.)

Here we use Red Hat's `chkconfig` for creating links to the startup scripts; use whatever means is appropriate for this purpose on your platform, such as `update-rc.d` on Debian.

Remember that the preceding steps must be repeated on each machine where an SQL node is to reside.

Data nodes. Installation of the data nodes does not require the `mysqld` binary. Only the NDB Cluster data node executable `ndbd` (single-threaded) or `ndbmtd` (multithreaded) is required. These binaries can also be found in the `.tar.gz` archive. Again, we assume that you have placed this archive in `/var/tmp`.

As system `root` (that is, after using `sudo`, `su root`, or your system's equivalent for temporarily assuming the system administrator account's privileges), perform the following steps to install the data node binaries on the data node hosts:

1. Change location to the `/var/tmp` directory, and extract the `ndbd` and `ndbmtd` binaries from the archive into a suitable directory such as `/usr/local/bin`:

   ```
   shell> cd /var/tmp
   shell> tar -zxvf mysql-cluster-gpl-8.0.19-linux-glibc2.12-x86_64.tar.gz
   shell> cd mysql-cluster-gpl-8.0.19-linux-glibc2.12-x86_64
   shell> cp bin/ndbd /usr/local/bin/ndbd
   shell> cp bin/ndbmtd /usr/local/bin/ndbmtd
   ```

   (You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once `ndb_mgm` and `ndb_mgmd` have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

   ```
   shell> cd /usr/local/bin
   shell> chmod +x ndb*
   ```

The preceding steps should be repeated on each data node host.

Although only one of the data node executables is required to run an NDB Cluster data node, we have shown you how to install both `ndbd` and `ndbmtd` in the preceding instructions. We recommend that you do this when installing or upgrading NDB Cluster, even if you plan to use only one of them, since this will save time and trouble in the event that you later decide to change from one to the other.

Note

The data directory on each machine hosting a data node is `/usr/local/mysql/data`. This piece of information is essential when configuring the management node. (See Section 22.2.4, “Initial Configuration of NDB Cluster”.)

Management nodes. Installation of the management node does not require the `mysqld` binary. Only the NDB Cluster management server (`ndb_mgmd`) is required; you most likely want to install the management client (`ndb_mgm`) as well. Both of these binaries also be found in the `.tar.gz` archive. Again, we assume that you have placed this archive in `/var/tmp`.

As system `root`, perform the following steps to install `ndb_mgmd` and `ndb_mgm` on the management node host:

1. Change location to the `/var/tmp` directory, and extract the `ndb_mgm` and `ndb_mgmd` from the archive into a suitable directory such as `/usr/local/bin`:

```
shell> cd /var/tmp
shell> tar -zxvf mysql-cluster-gpl-8.0.19-linux-glibc2.12-x86_64.tar.gz
shell> cd mysql-cluster-gpl-8.0.19-linux-glibc2.12-x86_64
shell> cp bin/ndb_mgm* /usr/local/bin
```

   (You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once `ndb_mgm` and `ndb_mgmd` have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

```
shell> cd /usr/local/bin
shell> chmod +x ndb_mgm*
```

In Section 22.2.4, “Initial Configuration of NDB Cluster”, we create configuration files for all of the nodes in our example NDB Cluster.

22.2.2.2 Installing NDB Cluster from RPM

This section covers the steps necessary to install the correct executables for each type of NDB Cluster 8.0 node using RPM packages supplied by Oracle. For information about RPMs for previous versions of NDB Cluster, see Installation using old-style RPMs (NDB 7.5.3 and earlier).

As an alternative to the method described in this section, Oracle provides MySQL Repositories for NDB Cluster that are compatible with many common Linux distributions. Two repostories, listed here, are available for RPM-based distributions:

- For distributions using `yum` or `dnf`, you can use the MySQL Yum Repository for NDB Cluster. See *Installing MySQL NDB Cluster Using the Yum Repository*, for instructions and additional information.
- For SLES, you can use the MySQL SLES Repository for NDB Cluster. See *Installing MySQL NDB Cluster Using the SLES Repository*, for instructions and additional information.

RPMs are available for both 32-bit and 64-bit Linux platforms. The filenames for these RPMs use the following pattern:

```
mysql-cluster-community-data-node-8.0.19-1.el7.x86_64.rpm

mysql-cluster-license-component-ver-rev.distro.arch.rpm

    license:= {commercial | community}

    component: {management-server | data-node | server | client | other—see text}

    ver: major.minor.release

    rev: major[.minor]

    distro: {el6 | el7 | sles12}

    arch: {i686 | x86_64}
```

`license` indicates whether the RPM is part of a Commercial or Community release of NDB Cluster. In the remainder of this section, we assume for the examples that you are installing a Community release.

Possible values for *component*, with descriptions, can be found in the following table:

Table 22.5 Components of the NDB Cluster RPM distribution

Component	Description
auto-installer	NDB Cluster Auto Installer program; see Section 22.2.1, “The NDB Cluster Auto-Installer”, for usage
client	MySQL and NDB client programs; includes mysql client, ndb_mgm client, and other client tools
common	Character set and error message information needed by the MySQL server
data-node	ndbd and ndbmtd data node binaries
devel	Headers and library files needed for MySQL client development
embedded	Embedded MySQL server
embedded-compat	Backwards-compatible embedded MySQL server
embedded-devel	Header and library files for developing applications for embedded MySQL
java	JAR files needed for support of ClusterJ applications
libs	MySQL client libraries
libs-compat	Backwards-compatible MySQL client libraries
management-server	The NDB Cluster management server (ndb_mgmd)
memcached	Files needed to support ndbmemcache
minimal-debuginfo	Debug information for package server-minimal; useful when developing applications that use this package or when debugging this package
ndbclient	NDB client library for running NDB API and MGM API applications (libndbclient)
ndbclient-devel	Header and other files needed for developing NDB API and MGM API applications
nodejs	Files needed to set up Node.JS support for NDB Cluster
server	The MySQL server (mysqld) with NDB storage engine support included, and associated MySQL server programs
server-minimal	Minimal installation of the MySQL server for NDB and related tools
test	mysqltest, other MySQL test programs, and support files

A single bundle (.tar file) of all NDB Cluster RPMs for a given platform and architecture is also available. The name of this file follows the pattern shown here:

```
mysql-cluster-license-ver-rev.distro.arch.rpm-bundle.tar
```

You can extract the individual RPM files from this file using `tar` or your preferred tool for extracting archives.

The components required to install the three major types of NDB Cluster nodes are given in the following list:

- *Management node*: `management-server`
- *Data node*: `data-node`
- *SQL node*: `server` and `common`

In addition, the `client` RPM should be installed to provide the `ndb_mgm` management client on at least one management node. You may also wish to install it on SQL nodes, to have `mysql` and other MySQL client programs available on these. We discuss installation of nodes by type later in this section.

`ver` represents the three-part `NDB` storage engine version number in 8.0.*`x`* format, shown as `8.0.19` in the examples. `rev` provides the RPM revision number in *`major`*.*`minor`* format. In the examples shown in this section, we use `1.1` for this value.

The *`distro`* (Linux distribution) is one of `rhel5` (Oracle Linux 5, Red Hat Enterprise Linux 4 and 5), `el6` (Oracle Linux 6, Red Hat Enterprise Linux 6), `el7` (Oracle Linux 7, Red Hat Enterprise Linux 7), or `sles12` (SUSE Enterprise Linux 12). For the examples in this section, we assume that the host runs Oracle Linux 7, Red Hat Enterprise Linux 7, or the equivalent (`el7`).

`arch` is `i686` for 32-bit RPMs and `x86_64` for 64-bit versions. In the examples shown here, we assume a 64-bit platform.

The NDB Cluster version number in the RPM file names (shown here as `8.0.19`) can vary according to the version which you are actually using. *It is very important that all of the Cluster RPMs to be installed have the same version number*. The architecture should also be appropriate to the machine on which the RPM is to be installed; in particular, you should keep in mind that 64-bit RPMs (`x86_64`) cannot be used with 32-bit operating systems (use `i686` for the latter).

Data nodes. On a computer that is to host an NDB Cluster data node it is necessary to install only the `data-node` RPM. To do so, copy this RPM to the data node host, and run the following command as the system root user, replacing the name shown for the RPM as necessary to match that of the RPM downloaded from the MySQL website:

```
shell> rpm -Uhv mysql-cluster-community-data-node-8.0.19-1.el7.x86_64.rpm
```

This installs the `ndbd` and `ndbmtd` data node binaries in `/usr/sbin`. Either of these can be used to run a data node process on this host.

SQL nodes. Copy the `server` and `common` RPMs to each machine to be used for hosting an NDB Cluster SQL node (`server` requires `common`). Install the `server` RPM by executing the following command as the system root user, replacing the name shown for the RPM as necessary to match the name of the RPM downloaded from the MySQL website:

```
shell> rpm -Uhv mysql-cluster-community-server-8.0.19-1.el7.x86_64.rpm
```

This installs the MySQL server binary (`mysqld`), with `NDB` storage engine support, in the `/usr/sbin` directory. It also installs all needed MySQL Server support files and useful MySQL server programs, including the `mysql.server` and `mysqld_safe` startup scripts (in `/usr/share/mysql` and `/usr/bin`, respectively). The RPM installer should take care of general configuration issues (such as creating the `mysql` user and group, if needed) automatically.

Important

You must use the versions of these RPMs released for NDB Cluster ; those released for the standard MySQL server do not provide support for the `NDB` storage engine.

To administer the SQL node (MySQL server), you should also install the `client` RPM, as shown here:

```
shell> rpm -Uhv mysql-cluster-community-client-8.0.19-1.el7.x86_64.rpm
```

This installs the `mysql` client and other MySQL client programs, such as `mysqladmin` and `mysqldump`, to `/usr/bin`.

Management nodes. To install the NDB Cluster management server, it is necessary only to use the `management-server` RPM. Copy this RPM to the computer intended to host the management node, and then install it by running the following command as the system root user (replace the name shown for the RPM as necessary to match that of the `management-server` RPM downloaded from the MySQL website):

```
shell> rpm -Uhv mysql-cluster-commercial-management-server-8.0.19-1.el7.x86_64.rpm
```

This RPM installs the management server binary `ndb_mgmd` in the `/usr/sbin` directory. While this is the only program actually required for running a management node, it is also a good idea to have the `ndb_mgm` NDB Cluster management client available as well. You can obtain this program, as well as other `NDB` client programs such as `ndb_desc` and `ndb_config`, by installing the `client` RPM as described previously.

See Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”, for general information about installing MySQL using RPMs supplied by Oracle.

After installing from RPM, you still need to configure the cluster; see Section 22.2.4, “Initial Configuration of NDB Cluster”, for the relevant information.

It is very important that all of the Cluster RPMs to be installed have the same version number. The *`architecture`* designation should also be appropriate to the machine on which the RPM is to be installed; in particular, you should keep in mind that 64-bit RPMs cannot be used with 32-bit operating systems.

Data nodes. On a computer that is to host a cluster data node it is necessary to install only the `server` RPM. To do so, copy this RPM to the data node host, and run the following command as the system root user, replacing the name shown for the RPM as necessary to match that of the RPM downloaded from the MySQL website:

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-8.0.19-1.sles11.i386.rpm
```

Although this installs all NDB Cluster binaries, only the program `ndbd` or `ndbmtd` (both in `/usr/sbin`) is actually needed to run an NDB Cluster data node.

SQL nodes. On each machine to be used for hosting a cluster SQL node, install the `server` RPM by executing the following command as the system root user, replacing the name shown for the RPM as necessary to match the name of the RPM downloaded from the MySQL website:

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-8.0.19-1.sles11.i386.rpm
```

This installs the MySQL server binary (`mysqld`) with `NDB` storage engine support in the `/usr/sbin` directory, as well as all needed MySQL Server support files. It also installs the `mysql.server` and

`mysqld_safe` startup scripts (in `/usr/share/mysql` and `/usr/bin`, respectively). The RPM installer should take care of general configuration issues (such as creating the `mysql` user and group, if needed) automatically.

To administer the SQL node (MySQL server), you should also install the `client` RPM, as shown here:

```
shell> rpm -Uhv MySQL-Cluster-client-gpl-8.0.19-1.sles11.i386.rpm
```

This installs the `mysql` client program.

Management nodes. To install the NDB Cluster management server, it is necessary only to use the `server` RPM. Copy this RPM to the computer intended to host the management node, and then install it by running the following command as the system root user (replace the name shown for the RPM as necessary to match that of the `server` RPM downloaded from the MySQL website):

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-8.0.19-1.sles11.i386.rpm
```

Although this RPM installs many other files, only the management server binary `ndb_mgmd` (in the `/usr/sbin` directory) is actually required for running a management node. The `server` RPM also installs `ndb_mgm`, the `NDB` management client.

See Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”, for general information about installing MySQL using RPMs supplied by Oracle. See Section 22.2.4, “Initial Configuration of NDB Cluster”, for information about required post-installation configuration.

22.2.2.3 Installing NDB Cluster Using .deb Files

The section provides information about installing NDB Cluster on Debian and related Linux distributions such Ubuntu using the `.deb` files supplied by Oracle for this purpose.

Oracle also provides an NDB Cluster APT repository for Debian and other distributions. See *Installing MySQL NDB Cluster Using the APT Repository*, for instructions and additional information.

Oracle provides `.deb` installer files for NDB Cluster for 32-bit and 64-bit platforms. For a Debian-based system, only a single installer file is necessary. This file is named using the pattern shown here, according to the applicable NDB Cluster version, Debian version, and architecture:

```
mysql-cluster-gpl-ndbver-debiandebianver-arch.deb
```

Here, *`ndbver`* is the 3-part `NDB` engine version number, *`debianver`* is the major version of Debian (`8` or `9`), and *`arch`* is one of `i686` or `x86_64`. In the examples that follow, we assume you wish to install NDB 8.0.19 on a 64-bit Debian 9 system; in this case, the installer file is named `mysql-cluster-gpl-8.0.19-debian9-x86_64.deb-bundle.tar`.

Once you have downloaded the appropriate `.deb` file, you can untar it, and then install it from the command line using `dpkg`, like this:

```
shell> dpkg -i mysql-cluster-gpl-8.0.19-debian9-i686.deb
```

You can also remove it using `dpkg` as shown here:

```
shell> dpkg -r mysql
```

The installer file should also be compatible with most graphical package managers that work with `.deb` files, such as `GDebi` for the Gnome desktop.

The `.deb` file installs NDB Cluster under `/opt/mysql/server-version/`, where *`version`* is the 2-part release series version for the included MySQL server. For NDB 8.0, this is always `5.7`. The directory layout is the same as that for the generic Linux binary distribution (see Table 2.3, "MySQL Installation Layout for Generic Unix/Linux Binary Package"), with the exception that startup scripts and configuration files are found in `support-files` instead of `share`. All NDB Cluster executables, such as `ndb_mgm`, `ndbd`, and `ndb_mgmd`, are placed in the `bin` directory.

22.2.2.4 Building NDB Cluster from Source on Linux

This section provides information about compiling NDB Cluster on Linux and other Unix-like platforms. Building NDB Cluster from source is similar to building the standard MySQL Server, although it differs in a few key respects discussed here. For general information about building MySQL from source, see Section 2.9, "Installing MySQL from Source". For information about compiling NDB Cluster on Windows platforms, see Section 22.2.3.2, "Compiling and Installing NDB Cluster from Source on Windows".

Building MySQL NDB Cluster 8.0 requires using the MySQL Server 8.0 sources. These are available from the MySQL downloads page at https://dev.mysql.com/downloads/. The archived source file should have a name similar to `mysql-8.0.19.tar.gz`. You can also obtain the sources from GitHub at https://github.com/mysql/mysql-server.

Note

In previous versions, building of NDB Cluster from standard MySQL Server sources was not supported. In MySQL 8.0 and NDB Cluster 8.0, this is no longer the case —*both products are now built from the same sources*.

The `WITH_NDBCLUSTER` option for `CMake` causes the binaries for the management nodes, data nodes, and other NDB Cluster programs to be built; it also causes `mysqld` to be compiled with `NDB` storage engine support. This option (or one of its aliases `WITH_NDBCLUSTER_STORAGE_ENGINE` and `WITH_PLUGIN_NDBCLUSTER`) is required when building NDB Cluster.

Important

The `WITH_NDB_JAVA` option is enabled by default. This means that, by default, if `CMake` cannot find the location of Java on your system, the configuration process fails; if you do not wish to enable Java and ClusterJ support, you must indicate this explicitly by configuring the build using `-DWITH_NDB_JAVA=OFF`. Use `WITH_CLASSPATH` to provide the Java classpath if needed.

For more information about `CMake` options specific to building NDB Cluster, see Options for Compiling NDB Cluster.

After you have run `make && make install` (or your system's equivalent), the result is similar to what is obtained by unpacking a precompiled binary to the same location.

Management nodes. When building from source and running the default `make install`, the management server and management client binaries (`ndb_mgmd` and `ndb_mgm`) can be found in `/usr/local/mysql/bin`. Only `ndb_mgmd` is required to be present on a management node host; however, it is also a good idea to have `ndb_mgm` present on the same host machine. Neither of these executables requires a specific location on the host machine's file system.

Data nodes. The only executable required on a data node host is the data node binary `ndbd` or `ndbmtd`. (`mysqld`, for example, does not have to be present on the host machine.) By default, when building from source, this file is placed in the directory `/usr/local/mysql/bin`. For installing on multiple data node hosts, only `ndbd` or `ndbmtd` need be copied to the other host machine or machines. (This

assumes that all data node hosts use the same architecture and operating system; otherwise you may need to compile separately for each different platform.) The data node binary need not be in any particular location on the host's file system, as long as the location is known.

When compiling NDB Cluster from source, no special options are required for building multithreaded data node binaries. Configuring the build with `NDB` storage engine support causes `ndbmtd` to be built automatically; `make install` places the `ndbmtd` binary in the installation `bin` directory along with `mysqld`, `ndbd`, and `ndb_mgm`.

SQL nodes. If you compile MySQL with clustering support, and perform the default installation (using `make install` as the system `root` user), `mysqld` is placed in `/usr/local/mysql/bin`. Follow the steps given in Section 2.9, “Installing MySQL from Source” to make `mysqld` ready for use. If you want to run multiple SQL nodes, you can use a copy of the same `mysqld` executable and its associated support files on several machines. The easiest way to do this is to copy the entire `/usr/local/mysql` directory and all directories and files contained within it to the other SQL node host or hosts, then repeat the steps from Section 2.9, “Installing MySQL from Source” on each machine. If you configure the build with a nondefault `PREFIX` option, you must adjust the directory accordingly.

In Section 22.2.4, “Initial Configuration of NDB Cluster”, we create configuration files for all of the nodes in our example NDB Cluster.

22.2.3 Installing NDB Cluster on Windows

This section describes installation procedures for NDB Cluster on Windows hosts. NDB Cluster 8.0 binaries for Windows can be obtained from https://dev.mysql.com/downloads/cluster/. For information about installing NDB Cluster on Windows from a binary release provided by Oracle, see Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”.

It is also possible to compile and install NDB Cluster from source on Windows using Microsoft Visual Studio. For more information, see Section 22.2.3.2, “Compiling and Installing NDB Cluster from Source on Windows”.

22.2.3.1 Installing NDB Cluster on Windows from a Binary Release

This section describes a basic installation of NDB Cluster on Windows using a binary “no-install” NDB Cluster release provided by Oracle, using the same 4-node setup outlined in the beginning of this section (see Section 22.2, “NDB Cluster Installation”), as shown in the following table:

Table 22.6 Network addresses of nodes in example cluster

Node	IP Address
Management node (`mgmd`)	198.51.100.10
SQL node (`mysqld`)	198.51.100.20
Data node "A" (`ndbd`)	198.51.100.30
Data node "B" (`ndbd`)	198.51.100.40

As on other platforms, the NDB Cluster host computer running an SQL node must have installed on it a MySQL Server binary (`mysqld.exe`). You should also have the MySQL client (`mysql.exe`) on this host. For management nodes and data nodes, it is not necessary to install the MySQL Server binary; however, each management node requires the management server daemon (`ndb_mgmd.exe`); each data node requires the data node daemon (`ndbd.exe` or `ndbmtd.exe`). For this example, we refer to `ndbd.exe` as the data node executable, but you can install `ndbmtd.exe`, the multithreaded version of this program, instead, in exactly the same way. You should also install the management client (`ndb_mgm.exe`) on the

management server host. This section covers the steps necessary to install the correct Windows binaries for each type of NDB Cluster node.

Note

As with other Windows programs, NDB Cluster executables are named with the `.exe` file extension. However, it is not necessary to include the `.exe` extension when invoking these programs from the command line. Therefore, we often simply refer to these programs in this documentation as `mysqld`, `mysql`, `ndb_mgmd`, and so on. You should understand that, whether we refer (for example) to `mysqld` or `mysqld.exe`, either name means the same thing (the MySQL Server program).

For setting up an NDB Cluster using Oracles's `no-install` binaries, the first step in the installation process is to download the latest NDB Cluster Windows ZIP binary archive from https://dev.mysql.com/downloads/cluster/. This archive has a filename of the `mysql-cluster-gpl-`*`ver`*`-win`*`arch`*`.zip`, where *`ver`* is the `NDB` storage engine version (such as `8.0.19`), and *`arch`* is the architecture (`32` for 32-bit binaries, and `64` for 64-bit binaries). For example, the NDB Cluster 8.0.19 archive for 64-bit Windows systems is named `mysql-cluster-gpl-8.0.19-win64.zip`.

You can run 32-bit NDB Cluster binaries on both 32-bit and 64-bit versions of Windows; however, 64-bit NDB Cluster binaries can be used only on 64-bit versions of Windows. If you are using a 32-bit version of Windows on a computer that has a 64-bit CPU, then you must use the 32-bit NDB Cluster binaries.

To minimize the number of files that need to be downloaded from the Internet or copied between machines, we start with the computer where you intend to run the SQL node.

SQL node. We assume that you have placed a copy of the archive in the directory `C:\Documents and Settings\`*`username`*`\My Documents\Downloads` on the computer having the IP address 198.51.100.20, where *`username`* is the name of the current user. (You can obtain this name using `ECHO %USERNAME%` on the command line.) To install and run NDB Cluster executables as Windows services, this user should be a member of the `Administrators` group.

Extract all the files from the archive. The Extraction Wizard integrated with Windows Explorer is adequate for this task. (If you use a different archive program, be sure that it extracts all files and directories from the archive, and that it preserves the archive's directory structure.) When you are asked for a destination directory, enter `C:\`, which causes the Extraction Wizard to extract the archive to the directory `C:\mysql-cluster-gpl-`*`ver`*`-win`*`arch`*. Rename this directory to `C:\mysql`.

It is possible to install the NDB Cluster binaries to directories other than `C:\mysql\bin`; however, if you do so, you must modify the paths shown in this procedure accordingly. In particular, if the MySQL Server (SQL node) binary is installed to a location other than `C:\mysql` or `C:\Program Files\MySQL\MySQL Server 8.0`, or if the SQL node's data directory is in a location other than `C:\mysql\data` or `C:\Program Files\MySQL\MySQL Server 8.0\data`, extra configuration options must be used on the command line or added to the `my.ini` or `my.cnf` file when starting the SQL node. For more information about configuring a MySQL Server to run in a nonstandard location, see Section 2.3.4, “Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive”.

For a MySQL Server with NDB Cluster support to run as part of an NDB Cluster, it must be started with the options `--ndbcluster` and `--ndb-connectstring`. While you can specify these options on the command line, it is usually more convenient to place them in an option file. To do this, create a new text file in Notepad or another text editor. Enter the following configuration information into this file:

```
[mysqld]
# Options for mysqld process:
ndbcluster                      # run NDB storage engine
ndb-connectstring=198.51.100.10  # location of management server
```

You can add other options used by this MySQL Server if desired (see Section 2.3.4.2, “Creating an Option File”), but the file must contain the options shown, at a minimum. Save this file as `C:\mysql\my.ini`. This completes the installation and setup for the SQL node.

Data nodes. An NDB Cluster data node on a Windows host requires only a single executable, one of either `ndbd.exe` or `ndbmtd.exe`. For this example, we assume that you are using `ndbd.exe`, but the same instructions apply when using `ndbmtd.exe`. On each computer where you wish to run a data node (the computers having the IP addresses 198.51.100.30 and 198.51.100.40), create the directories `C:\mysql`, `C:\mysql\bin`, and `C:\mysql\cluster-data`; then, on the computer where you downloaded and extracted the `no-install` archive, locate `ndbd.exe` in the `C:\mysql\bin` directory. Copy this file to the `C:\mysql\bin` directory on each of the two data node hosts.

To function as part of an NDB Cluster, each data node must be given the address or hostname of the management server. You can supply this information on the command line using the `--ndb-connectstring` or `-c` option when starting each data node process. However, it is usually preferable to put this information in an option file. To do this, create a new text file in Notepad or another text editor and enter the following text:

```
[mysql_cluster]
# Options for data node process:
ndb-connectstring=198.51.100.10  # location of management server
```

Save this file as `C:\mysql\my.ini` on the data node host. Create another text file containing the same information and save it on as `C:mysql\my.ini` on the other data node host, or copy the my.ini file from the first data node host to the second one, making sure to place the copy in the second data node's `C:\mysql` directory. Both data node hosts are now ready to be used in the NDB Cluster, which leaves only the management node to be installed and configured.

Management node. The only executable program required on a computer used for hosting an NDB Cluster management node is the management server program `ndb_mgmd.exe`. However, in order to administer the NDB Cluster once it has been started, you should also install the NDB Cluster management client program `ndb_mgm.exe` on the same machine as the management server. Locate these two programs on the machine where you downloaded and extracted the `no-install` archive; this should be the directory `C:\mysql\bin` on the SQL node host. Create the directory `C:\mysql\bin` on the computer having the IP address 198.51.100.10, then copy both programs to this directory.

You should now create two configuration files for use by `ndb_mgmd.exe`:

1. A local configuration file to supply configuration data specific to the management node itself. Typically, this file needs only to supply the location of the NDB Cluster global configuration file (see item 2).

   To create this file, start a new text file in Notepad or another text editor, and enter the following information:

   ```
   [mysql_cluster]
   # Options for management node process
   config-file=C:/mysql/bin/config.ini
   ```

   Save this file as the text file `C:\mysql\bin\my.ini`.

2. A global configuration file from which the management node can obtain configuration information governing the NDB Cluster as a whole. At a minimum, this file must contain a section for each node in the NDB Cluster, and the IP addresses or hostnames for the management node and all data nodes (`HostName` configuration parameter). It is also advisable to include the following additional information:

   - The IP address or hostname of any SQL nodes

- The data memory and index memory allocated to each data node (`DataMemory` and `IndexMemory` configuration parameters)
- The number of replicas, using the `NoOfReplicas` configuration parameter (see Section 22.1.2, “NDB Cluster Nodes, Node Groups, Replicas, and Partitions”)
- The directory where each data node stores it data and log file, and the directory where the management node keeps its log files (in both cases, the `DataDir` configuration parameter)

Create a new text file using a text editor such as Notepad, and input the following information:

```
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2                      # Number of replicas
DataDir=C:/mysql/cluster-data       # Directory for each data node's data files
                                    # Forward slashes used in directory path,
                                    # rather than backslashes. This is correct;
                                    # see Important note in text
DataMemory=80M    # Memory allocated to data storage
IndexMemory=18M   # Memory allocated to index storage
                  # For DataMemory and IndexMemory, we have used the
                  # default values. Since the "world" database takes up
                  # only about 500KB, this should be more than enough for
                  # this example Cluster setup.

[ndb_mgmd]
# Management process options:
HostName=198.51.100.10              # Hostname or IP address of management node
DataDir=C:/mysql/bin/cluster-logs   # Directory for management node log files

[ndbd]
# Options for data node "A":
                                # (one [ndbd] section per data node)
HostName=198.51.100.30          # Hostname or IP address

[ndbd]
# Options for data node "B":
HostName=198.51.100.40          # Hostname or IP address

[mysqld]
# SQL node options:
HostName=198.51.100.20          # Hostname or IP address
```

Save this file as the text file `C:\mysql\bin\config.ini`.

Important

A single backslash character (`\`) cannot be used when specifying directory paths in program options or configuration files used by NDB Cluster on Windows. Instead, you must either escape each backslash character with a second backslash (`\\`), or replace the backslash with a forward slash character (`/`). For example, the following line from the `[ndb_mgmd]` section of an NDB Cluster `config.ini` file does not work:

```
DataDir=C:\mysql\bin\cluster-logs
```

Instead, you may use either of the following:

```
DataDir=C:\\mysql\\bin\\cluster-logs  # Escaped backslashes
```

```
DataDir=C:/mysql/bin/cluster-logs     # Forward slashes
```

For reasons of brevity and legibility, we recommend that you use forward slashes in directory paths used in NDB Cluster program options and configuration files on Windows.

22.2.3.2 Compiling and Installing NDB Cluster from Source on Windows

Oracle provides precompiled NDB Cluster binaries for Windows which should be adequate for most users. However, if you wish, it is also possible to compile NDB Cluster for Windows from source code. The procedure for doing this is almost identical to the procedure used to compile the standard MySQL Server binaries for Windows, and uses the same tools. However, there are two major differences:

- Building MySQL NDB Cluster 8.0 requires using the MySQL Server 8.0 sources. These are available from the MySQL downloads page at https://dev.mysql.com/downloads/. The archived source file should have a name similar to `mysql-8.0.19.tar.gz`. You can also obtain the sources from GitHub at https://github.com/mysql/mysql-server.

- You must configure the build using the `WITH_NDBCLUSTER` option in addition to any other build options you wish to use with `CMake`. `WITH_NDBCLUSTER_STORAGE_ENGINE` and `WITH_PLUGIN_NDBCLUSTER` are supported as aliases for `WITH_NDBCLUSTER`, and work in exactly the same way.

Important

The `WITH_NDB_JAVA` option is enabled by default. This means that, by default, if `CMake` cannot find the location of Java on your system, the configuration process fails; if you do not wish to enable Java and ClusterJ support, you must indicate this explicitly by configuring the build using `-DWITH_NDB_JAVA=OFF`. (Bug #12379735) Use `WITH_CLASSPATH` to provide the Java classpath if needed.

For more information about `CMake` options specific to building NDB Cluster, see Options for Compiling NDB Cluster.

Once the build process is complete, you can create a Zip archive containing the compiled binaries; Section 2.9.4, “Installing MySQL Using a Standard Source Distribution” provides the commands needed to perform this task on Windows systems. The NDB Cluster binaries can be found in the `bin` directory of the resulting archive, which is equivalent to the `no-install` archive, and which can be installed and configured in the same manner. For more information, see Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”.

22.2.3.3 Initial Startup of NDB Cluster on Windows

Once the NDB Cluster executables and needed configuration files are in place, performing an initial start of the cluster is simply a matter of starting the NDB Cluster executables for all nodes in the cluster. Each cluster node process must be started separately, and on the host computer where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes.

1. On the management node host, issue the following command from the command line to start the management node process. The output should appear similar to what is shown here:

```
C:\mysql\bin> ndb_mgmd
2010-06-23 07:53:34 [MgmtSrvr] INFO -- NDB Cluster Management Server. mysql-8.0.19-ndb-8.0.19
2010-06-23 07:53:34 [MgmtSrvr] INFO -- Reading cluster configuration from 'config.ini'
```

The management node process continues to print logging output to the console. This is normal, because the management node is not running as a Windows service. (If you have used NDB Cluster on a Unix-like platform such as Linux, you may notice that the management node's default behavior in this regard on Windows is effectively the opposite of its behavior on Unix systems, where it runs by default as a Unix daemon process. This behavior is also true of NDB Cluster data node processes running on Windows.) For this reason, do not close the window in which `ndb_mgmd.exe` is running; doing so kills the management node process. (See Section 22.2.3.4, "Installing NDB Cluster Processes as Windows Services", where we show how to install and run NDB Cluster processes as Windows services.)

The required `-f` option tells the management node where to find the global configuration file (`config.ini`). The long form of this option is `--config-file`.

Important

An NDB Cluster management node caches the configuration data that it reads from `config.ini`; once it has created a configuration cache, it ignores the `config.ini` file on subsequent starts unless forced to do otherwise. This means that, if the management node fails to start due to an error in this file, you must make the management node re-read `config.ini` after you have corrected any errors in it. You can do this by starting `ndb_mgmd.exe` with the `--reload` or `--initial` option on the command line. Either of these options works to refresh the configuration cache.

It is not necessary or advisable to use either of these options in the management node's `my.ini` file.

For additional information about options which can be used with `ndb_mgmd`, see Section 22.4.4, "`ndb_mgmd` — The NDB Cluster Management Server Daemon", as well as Section 22.4.31, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

2. On each of the data node hosts, run the command shown here to start the data node processes:

```
C:\mysql\bin> ndbd
2010-06-23 07:53:46 [ndbd] INFO -- Configuration fetched from 'localhost:1186', generation: 1
```

In each case, the first line of output from the data node process should resemble what is shown in the preceding example, and is followed by additional lines of logging output. As with the management node process, this is normal, because the data node is not running as a Windows service. For this reason, do not close the console window in which the data node process is running; doing so kills `ndbd.exe`. (For more information, see Section 22.2.3.4, "Installing NDB Cluster Processes as Windows Services".)

3. Do not start the SQL node yet; it cannot connect to the cluster until the data nodes have finished starting, which may take some time. Instead, in a new console window on the management node host, start the NDB Cluster management client `ndb_mgm.exe`, which should be in `C:\mysql\bin` on the management node host. (Do not try to re-use the console window where `ndb_mgmd.exe` is running by typing **CTRL**+**C**, as this kills the management node.) The resulting output should look like this:

```
C:\mysql\bin> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm>
```

When the prompt `ndb_mgm>` appears, this indicates that the management client is ready to receive NDB Cluster management commands. You can observe the status of the data nodes as they start by

entering `ALL STATUS` at the management client prompt. This command causes a running report of the data nodes's startup sequence, which should look something like this:

```
ndb_mgm> ALL STATUS
Connected to Management Server at: localhost:1186
Node 2: starting (Last completed phase 3) (mysql-8.0.19-ndb-8.0.19)
Node 3: starting (Last completed phase 3) (mysql-8.0.19-ndb-8.0.19)

Node 2: starting (Last completed phase 4) (mysql-8.0.19-ndb-8.0.19)
Node 3: starting (Last completed phase 4) (mysql-8.0.19-ndb-8.0.19)

Node 2: Started (version 8.0.19)
Node 3: Started (version 8.0.19)

ndb_mgm>
```

Note

Commands issued in the management client are not case-sensitive; we use uppercase as the canonical form of these commands, but you are not required to observe this convention when inputting them into the `ndb_mgm` client. For more information, see Section 22.5.2, “Commands in the NDB Cluster Management Client”.

The output produced by `ALL STATUS` is likely to vary from what is shown here, according to the speed at which the data nodes are able to start, the release version number of the NDB Cluster software you are using, and other factors. What is significant is that, when you see that both data nodes have started, you are ready to start the SQL node.

You can leave `ndb_mgm.exe` running; it has no negative impact on the performance of the NDB Cluster, and we use it in the next step to verify that the SQL node is connected to the cluster after you have started it.

4. On the computer designated as the SQL node host, open a console window and navigate to the directory where you unpacked the NDB Cluster binaries (if you are following our example, this is `C:\mysql\bin`).

   Start the SQL node by invoking `mysqld.exe` from the command line, as shown here:

   ```
   C:\mysql\bin> mysqld --console
   ```

   The `--console` option causes logging information to be written to the console, which can be helpful in the event of problems. (Once you are satisfied that the SQL node is running in a satisfactory manner, you can stop it and restart it out without the `--console` option, so that logging is performed normally.)

   In the console window where the management client (`ndb_mgm.exe`) is running on the management node host, enter the `SHOW` command, which should produce output similar to what is shown here:

   ```
   ndb_mgm> SHOW
   Connected to Management Server at: localhost:1186
   Cluster Configuration
   ---------------------
   [ndbd(NDB)]     2 node(s)
   id=2    @198.51.100.30  (Version: 8.0.19-ndb-8.0.19, Nodegroup: 0, *)
   id=3    @198.51.100.40  (Version: 8.0.19-ndb-8.0.19, Nodegroup: 0)

   [ndb_mgmd(MGM)] 1 node(s)
   id=1    @198.51.100.10  (Version: 8.0.19-ndb-8.0.19)
   ```

```
[mysqld(API)]   1 node(s)
id=4    @198.51.100.20  (Version: 8.0.19-ndb-8.0.19)
```

You can also verify that the SQL node is connected to the NDB Cluster in the `mysql` client (`mysql.exe`) using the `SHOW ENGINE NDB STATUS` statement.

You should now be ready to work with database objects and data using NDB Cluster 's `NDBCLUSTER` storage engine. See Section 22.2.6, “NDB Cluster Example with Tables and Data”, for more information and examples.

You can also install `ndb_mgmd.exe`, `ndbd.exe`, and `ndbmtd.exe` as Windows services. For information on how to do this, see Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”).

22.2.3.4 Installing NDB Cluster Processes as Windows Services

Once you are satisfied that NDB Cluster is running as desired, you can install the management nodes and data nodes as Windows services, so that these processes are started and stopped automatically whenever Windows is started or stopped. This also makes it possible to control these processes from the command line with the appropriate `SC START` and `SC STOP` commands, or using the Windows graphical `Services` utility. `NET START` and `NET STOP` commands can also be used.

Installing programs as Windows services usually must be done using an account that has Administrator rights on the system.

To install the management node as a service on Windows, invoke `ndb_mgmd.exe` from the command line on the machine hosting the management node, using the `--install` option, as shown here:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --install
Installing service 'NDB Cluster Management Server'
  as '"C:\mysql\bin\ndbd.exe" "--service=ndb_mgmd"'
Service successfully installed.
```

Important

When installing an NDB Cluster program as a Windows service, you should always specify the complete path; otherwise the service installation may fail with the error `The system cannot find the file specified`.

The `--install` option must be used first, ahead of any other options that might be specified for `ndb_mgmd.exe`. However, it is preferable to specify such options in an options file instead. If your options file is not in one of the default locations as shown in the output of `ndb_mgmd.exe --help`, you can specify the location using the `--config-file` option.

Now you should be able to start and stop the management server like this:

```
C:\> SC START ndb_mgmd

C:\> SC STOP ndb_mgmd
```

Note

If using `NET` commands, you can also start or stop the management server as a Windows service using the descriptive name, as shown here:

```
C:\> NET START 'NDB Cluster Management Server'
```

```
The NDB Cluster Management Server service is starting.
The NDB Cluster Management Server service was started successfully.

C:\> NET STOP  'NDB Cluster Management Server'
The NDB Cluster Management Server service is stopping..
The NDB Cluster Management Server service was stopped successfully.
```

It is usually simpler to specify a short service name or to permit the default service name to be used when installing the service, and then reference that name when starting or stopping the service. To specify a service name other than `ndb_mgmd`, append it to the `--install` option, as shown in this example:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --install=mgmd1
Installing service 'NDB Cluster Management Server'
  as '"C:\mysql\bin\ndb_mgmd.exe" "--service=mgmd1"'
Service successfully installed.
```

Now you should be able to start or stop the service using the name you have specified, like this:

```
C:\> SC START mgmd1

C:\> SC STOP mgmd1
```

To remove the management node service, use `SC DELETE service_name`:

```
C:\> SC DELETE mgmd1
```

Alternatively, invoke `ndb_mgmd.exe` with the `--remove` option, as shown here:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --remove
Removing service 'NDB Cluster Management Server'
Service successfully removed.
```

If you installed the service using a service name other than the default, pass the service name as the value of the `ndb_mgmd.exe --remove` option, like this:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --remove=mgmd1
Removing service 'mgmd1'
Service successfully removed.
```

Installation of an NDB Cluster data node process as a Windows service can be done in a similar fashion, using the `--install` option for `ndbd.exe` (or `ndbmtd.exe`), as shown here:

```
C:\> C:\mysql\bin\ndbd.exe --install
Installing service 'NDB Cluster Data Node Daemon' as '"C:\mysql\bin\ndbd.exe" "--service=ndbd"'
Service successfully installed.
```

Now you can start or stop the data node as shown in the following example:

```
C:\> SC START ndbd

C:\> SC STOP ndbd
```

To remove the data node service, use `SC DELETE service_name`:

```
C:\> SC DELETE ndbd
```

Alternatively, invoke `ndbd.exe` with the `--remove` option, as shown here:

```
C:\> C:\mysql\bin\ndbd.exe --remove
Removing service 'NDB Cluster Data Node Daemon'
Service successfully removed.
```

As with `ndb_mgmd.exe` (and `mysqld.exe`), when installing `ndbd.exe` as a Windows service, you can also specify a name for the service as the value of `--install`, and then use it when starting or stopping the service, like this:

```
C:\> C:\mysql\bin\ndbd.exe --install=dnode1
Installing service 'dnode1' as '"C:\mysql\bin\ndbd.exe" "--service=dnode1"'
Service successfully installed.

C:\> SC START dnode1

C:\> SC STOP dnode1
```

If you specified a service name when installing the data node service, you can use this name when removing it as well, as shown here:

```
C:\> SC DELETE dnode1
```

Alternatively, you can pass the service name as the value of the `ndbd.exe --remove` option, as shown here:

```
C:\> C:\mysql\bin\ndbd.exe --remove=dnode1
Removing service 'dnode1'
Service successfully removed.
```

Installation of the SQL node as a Windows service, starting the service, stopping the service, and removing the service are done in a similar fashion, using `mysqld --install`, `SC START`, `SC STOP`, and `SC DELETE` (or `mysqld --remove`). `NET` commands can also be used to start or stop a service. For additional information, see Section 2.3.4.8, “Starting MySQL as a Windows Service”.

22.2.4 Initial Configuration of NDB Cluster

In this section, we discuss manual configuration of an installed NDB Cluster by creating and editing configuration files.

NDB Cluster also provides a GUI installer which can be used to perform the configuration without the need to edit text files in a separate application. For more information, see Section 22.2.1, “The NDB Cluster Auto-Installer”.

For our four-node, four-host NDB Cluster (see Cluster nodes and host computers), it is necessary to write four configuration files, one per node host.

- Each data node or SQL node requires a `my.cnf` file that provides two pieces of information: a *connection string* that tells the node where to find the management node, and a line telling the MySQL server on this host (the machine hosting the data node) to enable the `NDBCLUSTER` storage engine.

  For more information on connection strings, see Section 22.3.3.3, “NDB Cluster Connection Strings”.

- The management node needs a `config.ini` file telling it how many replicas to maintain, how much memory to allocate for data and indexes on each data node, where to find the data nodes, where to save data to disk on each data node, and where to find any SQL nodes.

Configuring the data nodes and SQL nodes. The `my.cnf` file needed for the data nodes is fairly simple. The configuration file should be located in the `/etc` directory and can be edited using any text editor. (Create the file if it does not exist.) For example:

```
shell> vi /etc/my.cnf
```

Note

We show `vi` being used here to create the file, but any text editor should work just as well.

For each data node and SQL node in our example setup, `my.cnf` should look like this:

```
[mysqld]
# Options for mysqld process:
ndbcluster                      # run NDB storage engine

[mysql_cluster]
# Options for NDB Cluster processes:
ndb-connectstring=198.51.100.10  # location of management server
```

After entering the preceding information, save this file and exit the text editor. Do this for the machines hosting data node “A”, data node “B”, and the SQL node.

Important

Once you have started a `mysqld` process with the `ndbcluster` and `ndb-connectstring` parameters in the `[mysqld]` and `[mysql_cluster]` sections of the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. This is by design.

Configuring the management node. The first step in configuring the management node is to create the directory in which the configuration file can be found and then to create the file itself. For example (running as `root`):

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini
```

For our representative setup, the `config.ini` file should read as follows:

```
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2    # Number of replicas
DataMemory=98M    # How much memory to allocate for data storage

[ndb_mgmd]
# Management process options:
HostName=198.51.100.10          # Hostname or IP address of MGM node
DataDir=/var/lib/mysql-cluster  # Directory for MGM node log files

[ndbd]
# Options for data node "A":
                                # (one [ndbd] section per data node)
HostName=198.51.100.30          # Hostname or IP address
NodeId=2                        # Node ID for this data node
DataDir=/usr/local/mysql/data   # Directory for this data node's data files
```

```
[ndbd]
# Options for data node "B":
HostName=198.51.100.40          # Hostname or IP address
NodeId=3                        # Node ID for this data node
DataDir=/usr/local/mysql/data   # Directory for this data node's data files

[mysqld]
# SQL node options:
HostName=198.51.100.20          # Hostname or IP address
                                # (additional mysqld connections can be
                                # specified for this node for various
                                # purposes such as running ndb_restore)
```

Note

The `world` database can be downloaded from https://dev.mysql.com/doc/index-other.html.

After all the configuration files have been created and these minimal options have been specified, you are ready to proceed with starting the cluster and verifying that all processes are running. We discuss how this is done in Section 22.2.5, “Initial Startup of NDB Cluster”.

For more detailed information about the available NDB Cluster configuration parameters and their uses, see Section 22.3.3, “NDB Cluster Configuration Files”, and Section 22.3, “Configuration of NDB Cluster”. For configuration of NDB Cluster as relates to making backups, see Section 22.5.3.3, “Configuration for NDB Cluster Backups”.

Note

The default port for Cluster management nodes is 1186; the default port for data nodes is 2202. However, the cluster can automatically allocate ports for data nodes from those that are already free.

22.2.5 Initial Startup of NDB Cluster

Starting the cluster is not very difficult after it has been configured. Each cluster node process must be started separately, and on the host where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes:

1. On the management host, issue the following command from the system shell to start the management node process:

   ```
   shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
   ```

   The first time that it is started, `ndb_mgmd` must be told where to find its configuration file, using the `-f` or `--config-file` option. (See Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”, for details.)

   For additional options which can be used with `ndb_mgmd`, see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

2. On each of the data node hosts, run this command to start the `ndbd` process:

   ```
   shell> ndbd
   ```

3. If you used RPM files to install MySQL on the cluster host where the SQL node is to reside, you can (and should) use the supplied startup script to start the MySQL server process on the SQL node.

If all has gone well, and the cluster has been set up correctly, the cluster should now be operational. You can test this by invoking the `ndb_mgm` management node client. The output should look like that shown here, although you might see some slight differences in the output depending upon the exact version of MySQL that you are using:

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @198.51.100.30  (Version: 8.0.19-ndb-8.0.19, Nodegroup: 0, *)
id=3    @198.51.100.40  (Version: 8.0.19-ndb-8.0.19, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @198.51.100.10  (Version: 8.0.19-ndb-8.0.19)

[mysqld(API)]   1 node(s)
id=4    @198.51.100.20  (Version: 8.0.19-ndb-8.0.19)
```

The SQL node is referenced here as `[mysqld(API)]`, which reflects the fact that the `mysqld` process is acting as an NDB Cluster API node.

Note

The IP address shown for a given NDB Cluster SQL or other API node in the output of `SHOW` is the address used by the SQL or API node to connect to the cluster data nodes, and not to any management node.

You should now be ready to work with databases, tables, and data in NDB Cluster. See Section 22.2.6, “NDB Cluster Example with Tables and Data”, for a brief discussion.

22.2.6 NDB Cluster Example with Tables and Data

Note

The information in this section applies to NDB Cluster running on both Unix and Windows platforms.

Working with database tables and data in NDB Cluster is not much different from doing so in standard MySQL. There are two key points to keep in mind:

- For a table to be replicated in the cluster, it must use the `NDBCLUSTER` storage engine. To specify this, use the `ENGINE=NDBCLUSTER` or `ENGINE=NDB` option when creating the table:

  ```
  CREATE TABLE tbl_name (col_name column_definitions) ENGINE=NDBCLUSTER;
  ```

  Alternatively, for an existing table that uses a different storage engine, use `ALTER TABLE` to change the table to use `NDBCLUSTER`:

  ```
  ALTER TABLE tbl_name ENGINE=NDBCLUSTER;
  ```

- Every `NDBCLUSTER` table has a primary key. If no primary key is defined by the user when a table is created, the `NDBCLUSTER` storage engine automatically generates a hidden one. Such a key takes up space just as does any other table index. (It is not uncommon to encounter problems due to insufficient memory for accommodating these automatically created indexes.)

If you are importing tables from an existing database using the output of `mysqldump`, you can open the SQL script in a text editor and add the `ENGINE` option to any table creation statements, or replace any existing `ENGINE` options. Suppose that you have the `world` sample database on another MySQL server that does not support NDB Cluster, and you want to export the `City` table:

```
shell> mysqldump --add-drop-table world City > city_table.sql
```

The resulting `city_table.sql` file will contain this table creation statement (and the `INSERT` statements necessary to import the table data):

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

You need to make sure that MySQL uses the `NDBCLUSTER` storage engine for this table. There are two ways that this can be accomplished. One of these is to modify the table definition *before* importing it into the Cluster database. Using the `City` table as an example, modify the `ENGINE` option of the definition as follows:

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

This must be done for the definition of each table that is to be part of the clustered database. The easiest way to accomplish this is to do a search-and-replace on the file that contains the definitions and replace all instances of `TYPE=engine_name` or `ENGINE=engine_name` with `ENGINE=NDBCLUSTER`. If you do not want to modify the file, you can use the unmodified file to create the tables, and then use `ALTER TABLE` to change their storage engine. The particulars are given later in this section.

Assuming that you have already created a database named `world` on the SQL node of the cluster, you can then use the `mysql` command-line client to read `city_table.sql`, and create and populate the corresponding table in the usual manner:

```
shell> mysql world < city_table.sql
```

It is very important to keep in mind that the preceding command must be executed on the host where the SQL node is running (in this case, on the machine with the IP address `198.51.100.20`).

To create a copy of the entire `world` database on the SQL node, use `mysqldump` on the noncluster server to export the database to a file named `world.sql` (for example, in the `/tmp` directory). Then modify the table definitions as just described and import the file into the SQL node of the cluster like this:

```
shell> mysql world < /tmp/world.sql
```

If you save the file to a different location, adjust the preceding instructions accordingly.

Running `SELECT` queries on the SQL node is no different from running them on any other instance of a MySQL server. To run queries from the command line, you first need to log in to the MySQL Monitor in the usual way (specify the `root` password at the `Enter password:` prompt):

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 8.0.19-ndb-8.0.19

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

We simply use the MySQL server's `root` account and assume that you have followed the standard security precautions for installing a MySQL server, including setting a strong `root` password. For more information, see Section 2.10.4, “Securing the Initial MySQL Account”.

It is worth taking into account that NDB Cluster nodes do *not* make use of the MySQL privilege system when accessing one another. Setting or changing MySQL user accounts (including the `root` account) effects only applications that access the SQL node, not interaction between nodes. See Section 22.5.12.2, “NDB Cluster and MySQL Privileges”, for more information.

If you did not modify the `ENGINE` clauses in the table definitions prior to importing the SQL script, you should run the following statements at this point:

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

Selecting a database and running a `SELECT` query against a table in that database is also accomplished in the usual manner, as is exiting the MySQL Monitor:

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
+-----------+------------+
| Name      | Population |
+-----------+------------+
| Bombay    |   10500000 |
| Seoul     |    9981619 |
| São Paulo |    9968485 |
| Shanghai  |    9696300 |
| Jakarta   |    9604900 |
+-----------+------------+
5 rows in set (0.34 sec)

mysql> \q
Bye

shell>
```

Applications that use MySQL can employ standard APIs to access NDB tables. It is important to remember that your application must access the SQL node, and not the management or data nodes. This brief example shows how we might execute the SELECT statement just shown by using the PHP 5.X mysqli extension running on a Web server elsewhere on the network:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
          content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
  # connect to SQL node:
  $link = new mysqli('198.51.100.20', 'root', 'root_password', 'world');
  # parameters for mysqli constructor are:
  #   host, user, password, database

  if( mysqli_connect_errno() )
    die("Connect failed: " . mysqli_connect_error());

  $query = "SELECT Name, Population
            FROM City
            ORDER BY Population DESC
            LIMIT 5";

  # if no errors...
  if( $result = $link->query($query) )
  {
?>
<table border="1" width="40%" cellpadding="4" cellspacing ="1">
  <tbody>
  <tr>
    <th width="10%">City</th>
    <th>Population</th>
  </tr>
<?
    # then display the results...
    while($row = $result->fetch_object())
      printf("<tr>\n  <td align=\"center\">%s</td><td>%d</td>\n</tr>\n",
              $row->Name, $row->Population);
?>
  </tbody
</table>
<?
  # ...and verify the number of rows that were retrieved
    printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
  }
  else
    # otherwise, tell us what went wrong
    echo mysqli_error();

  # free the result set and the mysqli connection object
  $result->close();
  $link->close();
?>
</body>
</html>
```

We assume that the process running on the Web server can reach the IP address of the SQL node.

In a similar fashion, you can use the MySQL C API, Perl-DBI, Python-mysql, or MySQL Connectors to perform the tasks of data definition and manipulation just as you would normally with MySQL.

22.2.7 Safe Shutdown and Restart of NDB Cluster

To shut down the cluster, enter the following command in a shell on the machine hosting the management node:

```
shell> ndb_mgm -e shutdown
```

The `-e` option here is used to pass a command to the `ndb_mgm` client from the shell. (See Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”, for more information about this option.) The command causes the `ndb_mgm`, `ndb_mgmd`, and any `ndbd` or `ndbmtd` processes to terminate gracefully. Any SQL nodes can be terminated using `mysqladmin shutdown` and other means. On Windows platforms, assuming that you have installed the SQL node as a Windows service, you can use `SC STOP service_name` or `NET STOP service_name`.

To restart the cluster on Unix platforms, run these commands:

- On the management host (`198.51.100.10` in our example setup):

  ```
  shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
  ```

- On each of the data node hosts (`198.51.100.30` and `198.51.100.40`):

  ```
  shell> ndbd
  ```

- Use the `ndb_mgm` client to verify that both data nodes have started successfully.

- On the SQL host (`198.51.100.20`):

  ```
  shell> mysqld_safe &
  ```

On Windows platforms, assuming that you have installed all NDB Cluster processes as Windows services using the default service names (see Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”), you can restart the cluster as follows:

- On the management host (`198.51.100.10` in our example setup), execute the following command:

  ```
  C:\> SC START ndb_mgmd
  ```

- On each of the data node hosts (`198.51.100.30` and `198.51.100.40`), execute the following command:

  ```
  C:\> SC START ndbd
  ```

- On the management node host, use the `ndb_mgm` client to verify that the management node and both data nodes have started successfully (see Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”).

- On the SQL node host (`198.51.100.20`), execute the following command:

  ```
  C:\> SC START mysql
  ```

In a production setting, it is usually not desirable to shut down the cluster completely. In many cases, even when making configuration changes, or performing upgrades to the cluster hardware or software (or both), which require shutting down individual host machines, it is possible to do so without shutting down the

cluster as a whole by performing a *rolling restart* of the cluster. For more information about doing this, see Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”.

22.2.8 Upgrading and Downgrading NDB Cluster

This section provides information about NDB Cluster software and table file compatibility between different NDB Cluster 8.0 releases with regard to performing upgrades and downgrades as well as compatibility matrices and notes. You are expected already to be familiar with installing and configuring an NDB Cluster prior to attempting an upgrade or downgrade. See Section 22.3, “Configuration of NDB Cluster”.

Important

Only compatibility between MySQL versions with regard to `NDBCLUSTER` is taken into account in this section, and there are likely other issues to be considered. *As with any other MySQL software upgrade or downgrade, you are strongly encouraged to review the relevant portions of the MySQL Manual for the MySQL versions from which and to which you intend to migrate, before attempting an upgrade or downgrade of the NDB Cluster software*. See Section 2.11, “Upgrading MySQL”.

Known Issues. The following issues are known to occur when upgrading to or between NDB 8.0 releases:

- Online downgrades from NDB 8.0.14 to previous releases are not supported. Tables created in NDB 8.0.14 are not backwards compatible with previous releases. This is due to a change in usage of the extra metadata property implemented by `NDB` tables to provide full support for the MySQL data dictionary.

  For more information, see Changes in NDB table extra metadata. See also Chapter 14, *MySQL Data Dictionary*.

- Distributed privileges shared between MySQL servers as implemented in prior release series (see Distributed Privileges Using Shared Grant Tables) are not supported in NDB Cluster 8.0. When started, the `mysqld` supplied with NDB 8.0.16 and later checks for the existence of any grant tables which use the `NDB` storage engine; if it finds any, it creates local copies (“shadow tables”) of these using `InnoDB`. This is true for each MySQL server connected to NDB Cluster. After this has been performed on all MySQL servers acting as NDB Cluster SQL nodes, the `NDB` grant tables may be safely removed using the `ndb_drop_table` utility supplied with the NDB Cluster distribution, like this:

  ```
  ndb_drop_table -d mysql user db columns_priv tables_priv proxies_priv procs_priv
  ```

  It is safe to retain the `NDB` grant tables, but they are not used for access control and are effectively ignored.

  For more information about the MySQL privileges system used in NDB 8.0, see Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”, as well as Section 6.2.3, “Grant Tables”.

- In NDB 8.0.18, the binary configuration file format has been enhanced to provide support for great numbers of nodes. The new format is not accessible to 8.0.17 and older nodes, although newer management servers can detect older nodes and communicate with them using the appropriate format.

  Upgrades to NDB 8.0.18 or later from 8.0.17 and earlier should not be problematic in this regard. In the case of downgrades from NDB 8.0.18 or later to 8.0.17 or earlier, because older management servers cannot read the newer binary configuration file format, some manual intervention is required. When performing such a downgrade, it is necessary to remove any cached binary configuration files prior to

starting the management using the older NDB software version, and to have the plaintext configuration file available for the management server to read. Alternatively, you can start the older management server using the --initial option (again, it is necessary to have the config.ini available). If the cluster uses multiple management servers, one of these two things must be done for each management server binary.

- Direct downgrades of clusters running more than 48 data nodes, or with data nodes using node IDs greater than 48, to NDB versions 8.0.17 and earlier from NDB 8.0.18 or later are not supported. It is necessary in such cases to reduce the number of data nodes, change the configurations for all data nodes such that they use node IDs less than or equal to 48, or both, as required not to exceed the old maximums.

22.3 Configuration of NDB Cluster

A MySQL server that is part of an NDB Cluster differs in one chief respect from a normal (nonclustered) MySQL server, in that it employs the NDB storage engine. This engine is also referred to sometimes as NDBCLUSTER, although NDB is preferred.

To avoid unnecessary allocation of resources, the server is configured by default with the NDB storage engine disabled. To enable NDB, you must modify the server's my.cnf configuration file, or start the server with the --ndbcluster option.

This MySQL server is a part of the cluster, so it also must know how to access a management node to obtain the cluster configuration data. The default behavior is to look for the management node on localhost. However, should you need to specify that its location is elsewhere, this can be done in my.cnf, or with the mysql client. Before the NDB storage engine can be used, at least one management node must be operational, as well as any desired data nodes.

For more information about --ndbcluster and other mysqld options specific to NDB Cluster, see MySQL Server Options for NDB Cluster.

You can use also the NDB Cluster Auto-Installer to set up and deploy an NDB Cluster on one or more hosts using a browser-based GUI. For more information, see The NDB Cluster Auto-Installer (NDB 7.5).

For general information about installing NDB Cluster, see Section 22.2, “NDB Cluster Installation”.

22.3.1 Quick Test Setup of NDB Cluster

To familiarize you with the basics, we will describe the simplest possible configuration for a functional NDB Cluster. After this, you should be able to design your desired setup from the information provided in the other relevant sections of this chapter.

First, you need to create a configuration directory such as /var/lib/mysql-cluster, by executing the following command as the system root user:

```
shell> mkdir /var/lib/mysql-cluster
```

In this directory, create a file named config.ini that contains the following information. Substitute appropriate values for HostName and DataDir as necessary for your system.

```
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not required
# to do so.
```

```
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: [db], [api], and [mgm] are aliases for [ndbd], [mysqld], and [ndb_mgmd],
# respectively. [db] is deprecated and should not be used in new installations.

[ndbd default]
NoOfReplicas= 1

[mysqld  default]
[ndb_mgmd default]
[tcp default]

[ndb_mgmd]
HostName= myhost.example.com

[ndbd]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster

[mysqld]
[mysqld]
[mysqld]
```

You can now start the `ndb_mgmd` management server. By default, it attempts to read the `config.ini` file in its current working directory, so change location into the directory where the file is located and then invoke `ndb_mgmd`:

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

Then start a single data node by running `ndbd`:

```
shell> ndbd
```

For command-line options which can be used when starting `ndbd`, see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

By default, `ndbd` looks for the management server at `localhost` on port 1186.

Note

If you have installed MySQL from a binary tarball, you will need to specify the path of the `ndb_mgmd` and `ndbd` servers explicitly. (Normally, these will be found in `/usr/local/mysql/bin`.)

Finally, change location to the MySQL data directory (usually `/var/lib/mysql` or `/usr/local/mysql/data`), and make sure that the `my.cnf` file contains the option necessary to enable the NDB storage engine:

```
[mysqld]
ndbcluster
```

You can now start the MySQL server as usual:

```
shell> mysqld_safe --user=mysql &
```

Wait a moment to make sure the MySQL server is running properly. If you see the notice `mysql ended`, check the server's `.err` file to find out what went wrong.

If all has gone well so far, you now can start using the cluster. Connect to the server and verify that the NDBCLUSTER storage engine is enabled:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 8.0.19

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW ENGINES\G
...
*************************** 12. row ***************************
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
*************************** 13. row ***************************
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
...
```

The row numbers shown in the preceding example output may be different from those shown on your system, depending upon how your server is configured.

Try to create an NDBCLUSTER table:

```
shell> mysql
mysql> USE test;
Database changed

mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW CREATE TABLE ctest \G
*************************** 1. row ***************************
       Table: ctest
Create Table: CREATE TABLE `ctest` (
  `i` int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

To check that your nodes were set up properly, start the management client:

```
shell> ndb_mgm
```

Use the SHOW command from within the management client to obtain a report on the cluster's status:

```
ndb_mgm> SHOW
Cluster Configuration
---------------------
[ndbd(NDB)]     1 node(s)
id=2    @127.0.0.1  (Version: 8.0.19-ndb-8.0.19, Nodegroup: 0, *)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @127.0.0.1  (Version: 8.0.19-ndb-8.0.19)

[mysqld(API)]   3 node(s)
id=3    @127.0.0.1  (Version: 8.0.19-ndb-8.0.19)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)
```

At this point, you have successfully set up a working NDB Cluster . You can now store data in the cluster by using any table created with `ENGINE=NDBCLUSTER` or its alias `ENGINE=NDB`.

22.3.2 Overview of NDB Cluster Configuration Parameters, Options, and Variables

The next several sections provide summary tables of NDB Cluster node configuration parameters used in the `config.ini` file to govern various aspects of node behavior, as well as of options and variables read by `mysqld` from a `my.cnf` file or from the command line when run as an NDB Cluster process. Each of the node parameter tables lists the parameters for a given type (`ndbd`, `ndb_mgmd`, `mysqld`, `computer`, `tcp`, or `shm`). All tables include the data type for the parameter, option, or variable, as well as its default, mimimum, and maximum values as applicable.

Considerations when restarting nodes. For node parameters, these tables also indicate what type of restart is required (node restart or system restart)—and whether the restart must be done with `--initial`—to change the value of a given configuration parameter. When performing a node restart or an initial node restart, all of the cluster's data nodes must be restarted in turn (also referred to as a *rolling restart*). It is possible to update cluster configuration parameters marked as `node` online—that is, without shutting down the cluster—in this fashion. An initial node restart requires restarting each `ndbd` process with the `--initial` option.

A system restart requires a complete shutdown and restart of the entire cluster. An initial system restart requires taking a backup of the cluster, wiping the cluster file system after shutdown, and then restoring from the backup following the restart.

In any cluster restart, all of the cluster's management servers must be restarted for them to read the updated configuration parameter values.

Important

Values for numeric cluster parameters can generally be increased without any problems, although it is advisable to do so progressively, making such adjustments in relatively small increments. Many of these can be increased online, using a rolling restart.

However, decreasing the values of such parameters—whether this is done using a node restart, node initial restart, or even a complete system restart of the cluster—is not to be undertaken lightly; it is recommended that you do so only after careful planning and testing. This is especially true with regard to those parameters that relate to memory usage and disk space, such as `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes`. In addition, it is the generally the case that configuration parameters relating to memory and disk usage can be raised using a simple node restart, but they require an initial node restart to be lowered.

Because some of these parameters can be used for configuring more than one type of cluster node, they may appear in more than one of the tables.

Note

`4294967039` often appears as a maximum value in these tables. This value is defined in the `NDBCLUSTER` sources as `MAX_INT_RNIL` and is equal to `0xFFFFFEFF`, or $2^{32} - 2^8 - 1$.

22.3.2.1 NDB Cluster Data Node Configuration Parameters

The listings in this section provide information about parameters used in the `[ndbd]` or `[ndbd default]` sections of a `config.ini` file for configuring NDB Cluster data nodes. For detailed descriptions and other additional information about each of these parameters, see Section 22.3.3.6, “Defining NDB Cluster Data Nodes”.

These parameters also apply to `ndbmtd`, the multithreaded version of `ndbd`. For more information, see Section 22.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”.

- `Arbitration`: How arbitration should be performed to avoid split-brain issues in the event of node failure.
- `ArbitrationTimeout`: Maximum time (milliseconds) database partition waits for arbitration signal
- `BackupDataBufferSize`: Default size of databuffer for a backup (in bytes)
- `BackupDataDir`: Path to where to store backups. Note that the string '/BACKUP' is always appended to this setting, so that the *effective* default is FileSystemPath/BACKUP.
- `BackupDiskWriteSpeedPct`: Sets the percentage of the data node's allocated maximum write speed (MaxDiskWriteSpeed) to reserve for LCPs when starting abackup.
- `BackupLogBufferSize`: Default size of log buffer for a backup (in bytes)
- `BackupMaxWriteSize`: Maximum size of file system writes made by backup (in bytes)
- `BackupMemory`: Total memory allocated for backups per node (in bytes)
- `BackupReportFrequency`: Frequency of backup status reports during backup in seconds
- `BackupWriteSize`: Default size of file system writes made by backup (in bytes)
- `BatchSizePerLocalScan`: Used to calculate the number of lock records for scan with hold lock
- `BuildIndexThreads`: Number of threads to use for building ordered indexes during a system or node restart. Also applies when running ndb_restore --rebuild-indexes. Setting this parameter to 0 disables multithreaded building of ordered indexes.
- `CompressedBackup`: Use zlib to compress backups as they are written
- `CompressedLCP`: Write compressed LCPs using zlib
- `ConnectCheckIntervalDelay`: Time between data node connectivity check stages. Data node is considered suspect after 1 interval and dead after 2 intervals with no response.
- `CrashOnCorruptedTuple`: When enabled, forces node to shut down whenever it detects a corrupted tuple.
- `DataDir`: Data directory for this node
- `DataMemory`: Number of bytes on each data node allocated for storing data; subject to available system RAM.
- `DefaultHashMapSize`: Set size (in buckets) to use for table hash maps. Three values are supported: 0, 240, and 3840. Intended primarily for upgrades and downgrades within NDB 7.2.
- `DictTrace`: Enable DBDICT debugging; for NDB development
- `DiskIOThreadPool`: Number of unbound threads for file access (currently only for Disk Data); known as IOThreadPool before MySQL Cluster NDB 6.4.3.

- `Diskless`: Run without using the disk
- `DiskPageBufferEntries`: Number of 32K page entries to allocate in DiskPageBufferMemory. Very large disk transactions may require increasing this value.
- `DiskPageBufferMemory`: Number of bytes on each data node allocated for the disk page buffer cache
- `DiskSyncSize`: Amount of data written to file before a synch is forced
- `EnablePartialLcp`: Enable partial LCP (true); if this disabled (false), all LCPs write full checkpoints.
- `EnableRedoControl`: Enable adaptive checkpointing speed for controlling redo log usage
- `EventLogBufferSize`: Size of circular buffer for NDB log events within data nodes.
- `ExecuteOnComputer`: String referencing an earlier defined COMPUTER
- `ExtraSendBufferMemory`: Memory to use for send buffers in addition to any allocated by TotalSendBufferMemory or SendBufferMemory. Default (0) allows up to 16MB.
- `FileSystemPath`: Path to directory where the data node stores its data (directory must exist)
- `FileSystemPathDataFiles`: Path to directory where the data node stores its Disk Data files. The default value is FilesystemPathDD, if set; otherwise, FilesystemPath is used if it is set; otherwise, the value of DataDir is used.
- `FileSystemPathDD`: Path to directory where the data node stores its Disk Data and undo files. The default value is FileSystemPath, if set; otherwise, the value of DataDir is used.
- `FileSystemPathUndoFiles`: Path to directory where the data node stores its undo files for Disk Data. The default value is FilesystemPathDD, if set; otherwise, FilesystemPath is used if it is set; otherwise, the value of DataDir is used.
- `FragmentLogFileSize`: Size of each redo log file
- `HeartbeatIntervalDbApi`: Time between API node-data node heartbeats. (API connection closed after 3 missed heartbeats)
- `HeartbeatIntervalDbDb`: Time between data node-to-data node heartbeats; data node considered dead after 3 missed heartbeats
- `HeartbeatOrder`: Sets the order in which data nodes check each others' heartbeats for determining whether a given node is still active and connected to the cluster. Must be zero for all data nodes or distinct nonzero values for all data nodes; see documentation for further guidance.
- `HostName`: Host name or IP address for this data node.
- `IndexMemory`: Deprecated.
- `IndexStatAutoCreate`: Enable/disable automatic statistics collection when indexes are created.
- `IndexStatAutoUpdate`: Monitor indexes for changes and trigger automatic statistics updates
- `IndexStatSaveScale`: Scaling factor used in determining size of stored index statistics.
- `IndexStatSaveSize`: Maximum size in bytes for saved statistics per index.
- `IndexStatTriggerPct`: Threshold percent change in DML operations for index statistics updates. The value is scaled down by IndexStatTriggerScale.

- `IndexStatTriggerScale`: Scale down IndexStatTriggerPct by this amount, multiplied by the base 2 logarithm of the index size, for a large index. Set to 0 to disable scaling.
- `IndexStatUpdateDelay`: Minimum delay between automatic index statistics updates for a given index. 0 means no delay.
- `InitFragmentLogFiles`: Initialize fragment logfiles (sparse/full)
- `InitialLogFileGroup`: Describes a log file group that is created during an initial start. See documentation for format.
- `InitialNoOfOpenFiles`: Initial number of files open per data node. (One thread is created per file)
- `InitialTablespace`: Describes a tablespace that is created during an initial start. See documentation for format.
- `InsertRecoveryWork`: Percentage of RecoveryWork used for inserted rows; has no effect unless partial local checkpoints are in use.
- `LateAlloc`: Allocate memory after the connection to the management server has been established.
- `LcpScanProgressTimeout`: Maximum time that local checkpoint fragment scan can be stalled before node is shut down to ensure systemwide LCP progress. Use 0 to disable.
- `LockExecuteThreadToCPU`: A comma-delimited list of CPU IDs
- `LockMaintThreadsToCPU`: CPU ID indicating which CPU runs the maintenance threads
- `LockPagesInMainMemory`: 0=disable locking, 1=lock after memory allocation, 2=lock before memory allocation
- `LogLevelCheckpoint`: Log level of local and global checkpoint information printed to stdout
- `LogLevelCongestion`: Level of congestion information printed to stdout
- `LogLevelConnection`: Level of node connect/disconnect information printed to stdout
- `LogLevelError`: Transporter, heartbeat errors printed to stdout
- `LogLevelInfo`: Heartbeat and log information printed to stdout
- `LogLevelNodeRestart`: Level of node restart and node failure information printed to stdout
- `LogLevelShutdown`: Level of node shutdown information printed to stdout
- `LogLevelStartup`: Level of node startup information printed to stdout
- `LogLevelStatistic`: Level of transaction, operation, and transporter information printed to stdout
- `LongMessageBuffer`: Number of bytes allocated on each data node for internal long messages
- `MaxAllocate`: Maximum size of allocation to use when allocating memory for tables
- `MaxBufferedEpochs`: Allowed numbered of epochs that a subscribing node can lag behind (unprocessed epochs). Exceeding will cause lagging subscribers to be disconnected.
- `MaxBufferedEpochBytes`: Total number of bytes allocated for buffering epochs.
- `MaxDiskWriteSpeed`: Maximum number of bytes per second that can be written by LCP and backup when no restarts are ongoing.

- MaxDiskWriteSpeedOtherNodeRestart: Maximum number of bytes per second that can be written by LCP and backup when another node is restarting.
- MaxDiskWriteSpeedOwnRestart: Maximum number of bytes per second that can be written by LCP and backup when this node is restarting.
- MaxFKBuildBatchSize: Maximum scan batch size to use for building foreign keys. Increasing this value may speed up builds of foreign keys but impacts ongoing traffic as well.
- MaxDMLOperationsPerTransaction: Limit size of a transaction; aborts the transaction if it requires more than this many DML operations. Set to 0 to disable.
- MaxLCPStartDelay: Time in seconds that LCP polls for checkpoint mutex (to allow other data nodes to complete metadata synchronization), before putting itself in lock queue for parallel recovery of table data.
- MaxNoOfAttributes: Suggests a total number of attributes stored in database (sum over all tables)
- MaxNoOfConcurrentIndexOperations: Total number of index operations that can execute simultaneously on one data node
- MaxNoOfConcurrentOperations: Maximum number of operation records in transaction coordinator
- MaxNoOfConcurrentScans: Maximum number of scans executing concurrently on the data node
- MaxNoOfConcurrentSubOperations: Maximum number of concurrent subscriber operations
- MaxNoOfConcurrentTransactions: Maximum number of transactions executing concurrently on this data node, the total number of transactions that can be executed concurrently is this value times the number of data nodes in the cluster.
- MaxNoOfFiredTriggers: Total number of triggers that can fire simultaneously on one data node
- MaxNoOfLocalOperations: Maximum number of operation records defined on this data node
- MaxNoOfLocalScans: Maximum number of fragment scans in parallel on this data node
- MaxNoOfOpenFiles: Maximum number of files open per data node.(One thread is created per file)
- MaxNoOfOrderedIndexes: Total number of ordered indexes that can be defined in the system
- MaxNoOfSavedMessages: Maximum number of error messages to write in error log and maximum number of trace files to retain
- MaxNoOfSubscribers: Maximum number of subscribers (default 0 = MaxNoOfTables * 2)
- MaxNoOfSubscriptions: Maximum number of subscriptions (default 0 = MaxNoOfTables)
- MaxNoOfTables: Suggests a total number of NDB tables stored in the database
- MaxNoOfTriggers: Total number of triggers that can be defined in the system
- MaxNoOfUniqueHashIndexes: Total number of unique hash indexes that can be defined in the system
- MaxParallelCopyInstances: Number of parallel copies during node restarts. Default is 0, which uses number of LDMs on both nodes, to a maximum of 16.
- MaxParallelScansPerFragment: Maximum number of parallel scans per fragment. Once this limit is reached, scans are serialized.

- `MaxReorgBuildBatchSize`: Maximum scan batch size to use for reorganization of table partitions. Increasing this value may speed up table partition reorganization but impacts ongoing traffic as well.
- `MaxStartFailRetries`: Maximum retries when data node fails on startup, requires StopOnError = 0. Setting to 0 causes start attempts to continue indefinitely.
- `MaxUIBuildBatchSize`: Maximum scan batch size to use for building unique keys. Increasing this value may speed up builds of unique keys but impacts ongoing traffic as well.
- `MemReportFrequency`: Frequency of memory reports in seconds; 0 = report only when exceeding percentage limits
- `MinDiskWriteSpeed`: Minimum number of bytes per second that can be written by LCP and backup.
- `MinFreePct`: The percentage of memory resources to keep in reserve for restarts.
- `NodeGroup`: Node group to which the data node belongs; used only during initial start of cluster.
- `NodeId`: Number uniquely identifying the data node among all nodes in the cluster.
- `NoOfFragmentLogFiles`: Number of 16 MB redo log files in each of 4 file sets belonging to the data node
- `NoOfReplicas`: Number of copies of all data in database; recommended value is 2 (default). Values greater than 2 are not supported in production.
- `Numa`: (Linux only; requires libnuma) Controls NUMA support. Setting to 0 permits system to determine use of interleaving by data node process; 1 means that it is determined by data node.
- `ODirect`: Use O_DIRECT file reads and writes when possible.
- `ODirectSyncFlag`: O_DIRECT writes are treated as synchronized writes; ignored when ODirect is not enabled, InitFragmentLogFiles is set to SPARSE, or both.
- `RealtimeScheduler`: When true, data node threads are scheduled as real-time threads. Default is false.
- `RecoveryWork`: Percentage of storage overhead for LCP files: greater value means less work in normal operations, more work during recovery
- `RedoBuffer`: Number of bytes on each data node allocated for writing redo logs
- `RedoOverCommitCounter`: When RedoOverCommitLimit has been exceeded this many times, transactions are aborted, and operations are handled as specified by DefaultOperationRedoProblemAction
- `RedoOverCommitLimit`: Each time that flushing the current redo buffer takes longer than this many seconds, the number of times that this has happened is compared to RedoOverCommitCounter.
- `ReservedConcurrentIndexOperations`: Number of simultaneous index operations having dedicated resources on one data node
- `ReservedConcurrentOperations`: Number of simultaneous operations having dedicated resources in transaction coordinators on one data node
- `ReservedConcurrentScans`: Number of simultaneous scans having dedicated resources on one data node
- `ReservedConcurrentTransactions`: Number of simultaneous transactions having dedicated resources on one data node

- ReservedFiredTriggers: Number of triggers having dedicated resources on one data node
- ReservedLocalScans: Number of simultaneous fragment scans having dedicated resources on one data node
- ReservedTransactionBufferMemory: Dynamic buffer space (in bytes) for key and attribute data allocated to each data node
- RestartOnErrorInsert: Control the type of restart caused by inserting an error (when StopOnError is enabled)
- SchedulerExecutionTimer: Number of microseconds to execute in scheduler before sending
- SchedulerResponsiveness: Set NDB scheduler response optimization 0-10; higher values provide better response time but lower throughput
- SchedulerSpinTimer: Number of microseconds to execute in scheduler before sleeping
- ServerPort: Port used to set up transporter for incoming connections from API nodes
- SharedGlobalMemory: Total number of bytes on each data node allocated for any use
- StartFailRetryDelay: Delay in seconds after start failure prior to retry; requires StopOnError = 0.
- StartFailureTimeout: Milliseconds to wait before terminating. (0=Wait forever)
- StartNoNodeGroupTimeout: Time to wait for nodes without a nodegroup before trying to start (0=forever)
- StartPartialTimeout: Milliseconds to wait before trying to start without all nodes. (0=Wait forever)
- StartPartitionedTimeout: Milliseconds to wait before trying to start partitioned. (0=Wait forever)
- StartupStatusReportFrequency: Frequency of status reports during startup
- StopOnError: When set to 0, the data node automatically restarts and recovers following node failures
- StringMemory: Default size of string memory (0 to 100 = % of maximum, 101+ = actual bytes)
- TcpBind_INADDR_ANY: Bind IP_ADDR_ANY so that connections can be made from anywhere (for autogenerated connections)
- TimeBetweenEpochs: Time between epochs (synchronization used for replication)
- TimeBetweenEpochsTimeout: Timeout for time between epochs. Exceeding will cause node shutdown.
- TimeBetweenGlobalCheckpoints: Time between group commits of transactions to disk
- TimeBetweenGlobalCheckpointsTimeout: Minimum timeout for group commit of transactions to disk
- TimeBetweenInactiveTransactionAbortCheck: Time between checks for inactive transactions
- TimeBetweenLocalCheckpoints: Time between taking snapshots of the database (expressed in base-2 logarithm of bytes)
- TimeBetweenWatchDogCheck: Time between execution checks inside a data node

- `TimeBetweenWatchDogCheckInitial`: Time between execution checks inside a data node (early start phases when memory is allocated)
- `TotalSendBufferMemory`: Total memory to use for all transporter send buffers.
- `TransactionBufferMemory`: Dynamic buffer space (in bytes) for key and attribute data allocated for each data node
- `TransactionDeadlockDetectionTimeout`: Time transaction can spend executing within a data node. This is the time that the transaction coordinator waits for each data node participating in the transaction to execute a request. If the data node takes more than this amount of time, the transaction is aborted.
- `TransactionInactiveTimeout`: Milliseconds that the application waits before executing another part of the transaction. This is the time the transaction coordinator waits for the application to execute or send another part (query, statement) of the transaction. If the application takes too much time, then the transaction is aborted. Timeout = 0 means that the application never times out.
- `TwoPassInitialNodeRestartCopy`: Copy data in 2 passes during initial node restart, which enables multithreaded building of ordered indexes for such restarts.
- `UndoDataBuffer`: Number of bytes on each data node allocated for writing data undo logs
- `UndoIndexBuffer`: Number of bytes on each data node allocated for writing index undo logs
- `UseShm`: Use shared memory connections between this data node and API node also running on this host

The following parameters are specific to `ndbmtd`:

- `MaxNoOfExecutionThreads`: For ndbmtd only, specify maximum number of execution threads
- `NoOfFragmentLogParts`: Number of redo log file groups belonging to this data node; value must be an even multiple of 4.
- `ThreadConfig`: Used for configuration of multithreaded data nodes (ndbmtd). Default is an empty string; see documentation for syntax and other information.

22.3.2.2 NDB Cluster Management Node Configuration Parameters

The listing in this section provides information about parameters used in the `[ndb_mgmd]` or `[mgm]` section of a `config.ini` file for configuring NDB Cluster management nodes. For detailed descriptions and other additional information about each of these parameters, see Section 22.3.3.5, “Defining an NDB Cluster Management Server”.

- `ArbitrationDelay`: When asked to arbitrate, arbitrator waits this long before voting (milliseconds)
- `ArbitrationRank`: If 0, then management node is not arbitrator. Kernel selects arbitrators in order 1, 2
- `DataDir`: Data directory for this node
- `ExecuteOnComputer`: String referencing an earlier defined COMPUTER
- `ExtraSendBufferMemory`: Memory to use for send buffers in addition to any allocated by TotalSendBufferMemory or SendBufferMemory. Default (0) allows up to 16MB.
- `HeartbeatIntervalMgmdMgmd`: Time between management node-to-management node heartbeats; the connection between management node is considered lost after 3 missed heartbeats.

- `HeartbeatThreadPriority`: Set heartbeat thread policy and priority for management nodes; see manual for allowed values
- `HostName`: Host name or IP address for this management node.
- `Id`: Number identifying the management node (Id). Now deprecated; use NodeId instead.
- `LogDestination`: Where to send log messages: console, system log, or specified log file
- `NodeId`: Number uniquely identifying the management node among all nodes in the cluster.
- `PortNumber`: Port number to send commands to and fetch configuration from management server
- `PortNumberStats`: Port number used to get statistical information from a management server
- `TotalSendBufferMemory`: Total memory to use for all transporter send buffers
- `wan`: Use WAN TCP setting as default

Note

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect. See Section 22.3.3.5, “Defining an NDB Cluster Management Server”, for more information.

To add new management servers to a running NDB Cluster, it is also necessary perform a rolling restart of all cluster nodes after modifying any existing `config.ini` files. For more information about issues arising when using multiple management nodes, see Section 22.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”.

22.3.2.3 NDB Cluster SQL Node and API Node Configuration Parameters

The listing in this section provides information about parameters used in the `[mysqld]` and `[api]` sections of a `config.ini` file for configuring NDB Cluster SQL nodes and API nodes. For detailed descriptions and other additional information about each of these parameters, see Section 22.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

- `ApiVerbose`: Enable NDB API debugging; for NDB development
- `ArbitrationDelay`: When asked to arbitrate, arbitrator waits this many milliseconds before voting
- `ArbitrationRank`: If 0, then API node is not arbitrator. Kernel selects arbitrators in order 1, 2
- `AutoReconnect`: Specifies whether an API node should reconnect fully when disconnected from the cluster
- `BatchByteSize`: The default batch size in bytes
- `BatchSize`: The default batch size in number of records
- `ConnectBackoffMaxTime`: Specifies longest time in milliseconds (~100ms resolution) to allow between connection attempts to any given data node by this API node. Excludes time elapsed while connection attempts are ongoing, which in worst case can take several seconds. Disable by setting to 0. If no data nodes are currently connected to this API node, StartConnectBackoffMaxTime is used instead.
- `ConnectionMap`: Specifies which data nodes to connect

- `DefaultHashMapSize`: Set size (in buckets) to use for table hash maps. Three values are supported: 0, 240, and 3840. Intended primarily for upgrades and downgrades within NDB 7.2.
- `DefaultOperationRedoProblemAction`: How operations are handled in the event that RedoOverCommitCounter is exceeded
- `ExecuteOnComputer`: String referencing an earlier defined COMPUTER
- `ExtraSendBufferMemory`: Memory to use for send buffers in addition to any allocated by TotalSendBufferMemory or SendBufferMemory. Default (0) allows up to 16MB.
- `HeartbeatThreadPriority`: Set heartbeat thread policy and priority for API nodes; see manual for allowed values
- `HostName`: Host name or IP address for this SQL or API node.
- `Id`: Number identifying MySQL server or API node (Id). Now deprecated; use NodeId instead.
- `MaxScanBatchSize`: The maximum collective batch size for one scan
- `NodeId`: Number uniquely identifying the SQL node or API node among all nodes in the cluster.
- `StartConnectBackoffMaxTime`: Same as ConnectBackoffMaxTime except that this parameter is used in its place if no data nodes are connected to this API node.
- `TotalSendBufferMemory`: Total memory to use for all transporter send buffers
- `wan`: Use WAN TCP setting as default

For a discussion of MySQL server options for NDB Cluster, see MySQL Server Options for NDB Cluster. For information about MySQL server system variables relating to NDB Cluster, see NDB Cluster System Variables.

Note

To add new SQL or API nodes to the configuration of a running NDB Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new `[mysqld]` or `[api]` sections to the `config.ini` file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is *not* necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

22.3.2.4 Other NDB Cluster Configuration Parameters

The listings in this section provide information about parameters used in the `[computer]`, `[tcp]`, and `[shm]` sections of a `config.ini` file for configuring NDB Cluster. For detailed descriptions and additional information about individual parameters, see Section 22.3.3.10, “NDB Cluster TCP/IP Connections”, or Section 22.3.3.12, “NDB Cluster Shared-Memory Connections”, as appropriate.

The following parameters apply to the `config.ini` file's `[computer]` section:

- `HostName`: Host name or IP address of this computer.
- `Id`: A unique identifier for this computer.

The following parameters apply to the `config.ini` file's `[tcp]` section:

- `Checksum`: If checksum is enabled, all signals between nodes are checked for errors
- `Group`: Used for group proximity; smaller value is interpreted as being closer
- `NodeId1`: ID of node (data node, API node, or management node) on one side of the connection
- `NodeId2`: ID of node (data node, API node, or management node) on one side of the connection
- `NodeIdServer`: Set server side of TCP connection
- `OverloadLimit`: When more than this many unsent bytes are in the send buffer, the connection is considered overloaded.
- `PreSendChecksum`: If this parameter and Checksum are both enabled, perform pre-send checksum checks, and check all TCP signals between nodes for errors
- `Proxy`:
- `ReceiveBufferMemory`: Bytes of buffer for signals received by this node
- `SendBufferMemory`: Bytes of TCP buffer for signals sent from this node
- `SendSignalId`: Sends ID in each signal. Used in trace files. Defaults to true in debug builds.
- `TCP_MAXSEG_SIZE`: Value used for TCP_MAXSEG
- `TCP_RCV_BUF_SIZE`: Value used for SO_RCVBUF
- `TCP_SND_BUF_SIZE`: Value used for SO_SNDBUF
- `TcpBind_INADDR_ANY`: Bind InAddrAny instead of host name for server part of connection

The following parameters apply to the `config.ini` file's `[shm]` section:

- `Checksum`: If checksum is enabled, all signals between nodes are checked for errors
- `Group`:
- `NodeId1`: ID of node (data node, API node, or management node) on one side of the connection
- `NodeId2`: ID of node (data node, API node, or management node) on one side of the connection
- `NodeIdServer`: Set server side of SHM connection
- `OverloadLimit`: When more than this many unsent bytes are in the send buffer, the connection is considered overloaded.
- `PreSendChecksum`: If this parameter and Checksum are both enabled, perform pre-send checksum checks, and check all SHM signals between nodes for errors
- `SendBufferMemory`: Bytes in shared memory buffer for signals sent from this node
- `SendSignalId`: Sends ID in each signal. Used in trace files.
- `ShmKey`: A shared memory key; when set to 1, this is calculated by NDB
- `ShmSpinTime`: When receiving, number of microseconds to spin before sleeping
- `ShmSize`: Size of shared memory segment

- `Signum`: Signal number to be used for signalling

22.3.2.5 NDB Cluster mysqld Option and Variable Reference

The following table provides a list of the command-line options, server and status variables applicable within `mysqld` when it is running as an SQL node in an NDB Cluster. For a table showing *all* command-line options, server and status variables available for use with `mysqld`, see Section 5.1.4, "Server Option, System Variable, and Status Variable Reference".

- `Com_show_ndb_status`: Count of SHOW NDB STATUS statements
- `Handler_discover`: Number of times that tables have been discovered
- `ndb-batch-size`: Size (in bytes) to use for NDB transaction batches
- `ndb-blob-read-batch-bytes`: Specifies size in bytes that large BLOB reads should be batched into. 0 = no limit.
- `ndb-blob-write-batch-bytes`: Specifies size in bytes that large BLOB writes should be batched into. 0 = no limit.
- `ndb-cluster-connection-pool`: Number of connections to the cluster used by MySQL
- `ndb-cluster-connection-pool-nodeids`: Comma-separated list of node IDs for connections to the cluster used by MySQL; the number of nodes in the list must be the same as the value set for --ndb-cluster-connection-pool
- `ndb-connectstring`: Point to the management server that distributes the cluster configuration
- `ndb-default-column-format`: Use this value (FIXED or DYNAMIC) by default for COLUMN_FORMAT and ROW_FORMAT options when creating or adding columns to a table.
- `ndb-deferred-constraints`: Specifies that constraint checks on unique indexes (where these are supported) should be deferred until commit time. Not normally needed or used; for testing purposes only.
- `ndb-distribution`: Default distribution for new tables in NDBCLUSTER (KEYHASH or LINHASH, default is KEYHASH)
- `ndb-log-apply-status`: Cause a MySQL server acting as a slave to log mysql.ndb_apply_status updates received from its immediate master in its own binary log, using its own server ID. Effective only if the server is started with the --ndbcluster option.
- `ndb-log-empty-epochs`: When enabled, causes epochs in which there were no changes to be written to the ndb_apply_status and ndb_binlog_index tables, even when `log_slave_updates` is enabled.
- `ndb-log-empty-update`: When enabled, causes updates that produced no changes to be written to the ndb_apply_status and ndb_binlog_index tables, even when `log_slave_updates` is enabled.
- `ndb-log-exclusive-reads`: Log primary key reads with exclusive locks; allow conflict resolution based on read conflicts
- `ndb-log-orig`: Log originating server id and epoch in mysql.ndb_binlog_index table
- `ndb-log-transaction-id`: Write NDB transaction IDs in the binary log. Requires --log-bin-v1-events=OFF.
- `ndb-log-update-as-write`: Toggles logging of updates on the master between updates (OFF) and writes (ON)

- `ndb-mgmd-host`: Set the host (and port, if desired) for connecting to management server
- `ndb-nodeid`: NDB Cluster node ID for this MySQL server
- `ndb-recv-thread-activation-threshold`: Activation threshold when receive thread takes over the polling of the cluster connection (measured in concurrently active threads)
- `ndb-recv-thread-cpu-mask`: CPU mask for locking receiver threads to specific CPUs; specified as hexadecimal. See documentation for details.
- `ndb-transid-mysql-connection-map`: Enable or disable the ndb_transid_mysql_connection_map plugin; that is, enable or disable the INFORMATION_SCHEMA table having that name
- `ndb-wait-connected`: Time (in seconds) for the MySQL server to wait for connection to cluster management and data nodes before accepting MySQL client connections
- `ndb-wait-setup`: Time (in seconds) for the MySQL server to wait for NDB engine setup to complete
- `ndb-allow-copying-alter-table`: Set to OFF to keep ALTER TABLE from using copying operations on NDB tables
- `Ndb_api_bytes_received_count`: Amount of data (in bytes) received from the data nodes by this MySQL Server (SQL node)
- `Ndb_api_bytes_received_count_session`: Amount of data (in bytes) received from the data nodes in this client session
- `Ndb_api_bytes_received_count_slave`: Amount of data (in bytes) received from the data nodes by this slave
- `Ndb_api_bytes_sent_count`: Amount of data (in bytes) sent to the data nodes by this MySQL Server (SQL node)
- `Ndb_api_bytes_sent_count_session`: Amount of data (in bytes) sent to the data nodes in this client session
- `Ndb_api_bytes_sent_count_slave`: Amount of data (in bytes) sent to the data nodes by this slave
- `Ndb_api_event_bytes_count`: Number of bytes of events received by this MySQL Server (SQL node)
- `Ndb_api_event_bytes_count_injector`: Number of bytes of events received by the NDB binary log injector thread
- `Ndb_api_event_data_count`: Number of row change events received by this MySQL Server (SQL node)
- `Ndb_api_event_data_count_injector`: Number of row change events received by the NDB binary log injector thread
- `Ndb_api_event_nondata_count`: Number of events received, other than row change events, by this MySQL Server (SQL node)
- `Ndb_api_event_nondata_count_injector`: Number of events received, other than row change events, by the NDB binary log injector thread
- `Ndb_api_pk_op_count`: Number of operations based on or using primary keys by this MySQL Server (SQL node)

- Ndb_api_pk_op_count_session: Number of operations based on or using primary keys in this client session
- Ndb_api_pk_op_count_slave: Number of operations based on or using primary keys by this slave
- Ndb_api_pruned_scan_count: Number of scans that have been pruned to a single partition by this MySQL Server (SQL node)
- Ndb_api_pruned_scan_count_session: Number of scans that have been pruned to a single partition in this client session
- Ndb_api_pruned_scan_count_slave: Number of scans that have been pruned to a single partition by this slave
- Ndb_api_range_scan_count: Number of range scans that have been started by this MySQL Server (SQL node)
- Ndb_api_range_scan_count_session: Number of range scans that have been started in this client session
- Ndb_api_range_scan_count_slave: Number of range scans that have been started by this slave
- Ndb_api_read_row_count: Total number of rows that have been read by this MySQL Server (SQL node)
- Ndb_api_read_row_count_session: Total number of rows that have been read in this client session
- Ndb_api_read_row_count_slave: Total number of rows that have been read by this slave
- Ndb_api_scan_batch_count: Number of batches of rows received by this MySQL Server (SQL node)
- Ndb_api_scan_batch_count_session: Number of batches of rows received in this client session
- Ndb_api_scan_batch_count_slave: Number of batches of rows received by this slave
- Ndb_api_table_scan_count: Number of table scans that have been started, including scans of internal tables, by this MySQL Server (SQL node)
- Ndb_api_table_scan_count_session: Number of table scans that have been started, including scans of internal tables, in this client session
- Ndb_api_table_scan_count_slave: Number of table scans that have been started, including scans of internal tables, by this slave
- Ndb_api_trans_abort_count: Number of transactions aborted by this MySQL Server (SQL node)
- Ndb_api_trans_abort_count_session: Number of transactions aborted in this client session
- Ndb_api_trans_abort_count_slave: Number of transactions aborted by this slave
- Ndb_api_trans_close_count: Number of transactions aborted (may be greater than the sum of TransCommitCount and TransAbortCount) by this MySQL Server (SQL node)
- Ndb_api_trans_close_count_session: Number of transactions aborted (may be greater than the sum of TransCommitCount and TransAbortCount) in this client session
- Ndb_api_trans_close_count_slave: Number of transactions aborted (may be greater than the sum of TransCommitCount and TransAbortCount) by this slave

- Ndb_api_trans_commit_count: Number of transactions committed by this MySQL Server (SQL node)
- Ndb_api_trans_commit_count_session: Number of transactions committed in this client session
- Ndb_api_trans_commit_count_slave: Number of transactions committed by this slave
- Ndb_api_trans_local_read_row_count: Total number of rows that have been read by this MySQL Server (SQL node)
- Ndb_api_trans_local_read_row_count_session: Total number of rows that have been read in this client session
- Ndb_api_trans_local_read_row_count_slave: Total number of rows that have been read by this slave
- Ndb_api_trans_start_count: Number of transactions started by this MySQL Server (SQL node)
- Ndb_api_trans_start_count_session: Number of transactions started in this client session
- Ndb_api_trans_start_count_slave: Number of transactions started by this slave
- Ndb_api_uk_op_count: Number of operations based on or using unique keys by this MySQL Server (SQL node)
- Ndb_api_uk_op_count_session: Number of operations based on or using unique keys in this client session
- Ndb_api_uk_op_count_slave: Number of operations based on or using unique keys by this slave
- Ndb_api_wait_exec_complete_count: Number of times thread has been blocked while waiting for execution of an operation to complete by this MySQL Server (SQL node)
- Ndb_api_wait_exec_complete_count_session: Number of times thread has been blocked while waiting for execution of an operation to complete in this client session
- Ndb_api_wait_exec_complete_count_slave: Number of times thread has been blocked while waiting for execution of an operation to complete by this slave
- Ndb_api_wait_meta_request_count: Number of times thread has been blocked waiting for a metadata-based signal by this MySQL Server (SQL node)
- Ndb_api_wait_meta_request_count_session: Number of times thread has been blocked waiting for a metadata-based signal in this client session
- Ndb_api_wait_meta_request_count_slave: Number of times thread has been blocked waiting for a metadata-based signal by this slave
- Ndb_api_wait_nanos_count: Total time (in nanoseconds) spent waiting for some type of signal from the data nodes by this MySQL Server (SQL node)
- Ndb_api_wait_nanos_count_session: Total time (in nanoseconds) spent waiting for some type of signal from the data nodes in this client session
- Ndb_api_wait_nanos_count_slave: Total time (in nanoseconds) spent waiting for some type of signal from the data nodes by this slave
- Ndb_api_wait_scan_result_count: Number of times thread has been blocked while waiting for a scan-based signal by this MySQL Server (SQL node)

- Ndb_api_wait_scan_result_count_session: Number of times thread has been blocked while waiting for a scan-based signal in this client session
- Ndb_api_wait_scan_result_count_slave: Number of times thread has been blocked while waiting for a scan-based signal by this slave
- ndb_autoincrement_prefetch_sz: NDB auto-increment prefetch size
- ndb_cache_check_time: Number of milliseconds between checks of cluster SQL nodes made by the MySQL query cache
- ndb_clear_apply_status: Causes RESET SLAVE to clear all rows from the ndb_apply_status table; ON by default
- Ndb_cluster_node_id: If the server is acting as an NDB Cluster node, then the value of this variable its node ID in the cluster
- Ndb_config_from_host: The host name or IP address of the Cluster management server Formerly Ndb_connected_host
- Ndb_config_from_port: The port for connecting to Cluster management server. Formerly Ndb_connected_port
- Ndb_conflict_fn_epoch: Number of rows that have been found in conflict by the NDB$EPOCH() conflict detection function
- Ndb_conflict_fn_epoch2: Number of rows that have been found in conflict by the NDB$EPOCH2() conflict detection function
- Ndb_conflict_fn_epoch2_trans: Number of rows that have been found in conflict by the NDB $EPOCH2_TRANS() conflict detection function
- Ndb_conflict_fn_epoch_trans: Number of rows that have been found in conflict by the NDB $EPOCH_TRANS() conflict detection function
- Ndb_conflict_fn_max: If the server is part of an NDB Cluster involved in cluster replication, the value of this variable indicates the number of times that conflict resolution based on "greater timestamp wins" has been applied
- Ndb_conflict_fn_old: If the server is part of an NDB Cluster involved in cluster replication, the value of this variable indicates the number of times that "same timestamp wins" conflict resolution has been applied
- Ndb_conflict_last_conflict_epoch: Most recent NDB epoch on this slave in which a conflict was detected
- Ndb_conflict_last_stable_epoch: Number of rows found to be in conflict by a transactional conflict function
- Ndb_conflict_reflected_op_discard_count: Number of reflected operations that were not applied due an error during execution
- Ndb_conflict_reflected_op_prepare_count: Number of reflected operations received that have been prepared for execution
- Ndb_conflict_refresh_op_count: Number of refresh operations that have been prepared
- Ndb_conflict_trans_conflict_commit_count: Number of epoch transactions committed after requiring transactional conflict handling

- `Ndb_conflict_trans_detect_iter_count`: Number of internal iterations required to commit an epoch transaction. Should be (slightly) greater than or equal to Ndb_conflict_trans_conflict_commit_count
- `Ndb_conflict_trans_reject_count`: Number of transactions rejected after being found in conflict by a transactional conflict function
- `Ndb_conflict_trans_row_conflict_count`: Number of rows found in conflict by a transactional conflict function. Includes any rows included in or dependent on conflicting transactions.
- `Ndb_conflict_trans_row_reject_count`: Total number of rows realigned after being found in conflict by a transactional conflict function. Includes Ndb_conflict_trans_row_conflict_count and any rows included in or dependent on conflicting transactions.
- `ndb_data_node_neighbour`: Specifies cluster data node "closest" to this MySQL Server, for transaction hinting and fully replicated tables
- `ndb_default_column_format`: Sets default row format and column format (FIXED or DYNAMIC) used for new NDB tables
- `ndb_deferred_constraints`: Specifies that constraint checks should be deferred (where these are supported). Not normally needed or used; for testing purposes only.
- `ndb-schema-dist-timeout`: How long to wait before detecting timeout during schema distribution
- `ndb_distribution`: Default distribution for new tables in NDBCLUSTER (KEYHASH or LINHASH, default is KEYHASH)
- `Ndb_epoch_delete_delete_count`: Number of delete-delete conflicts detected (delete operation is applied, but row does not exist)
- `ndb_eventbuffer_free_percent`: Percentage of free memory that should be available in event buffer before resumption of buffering, after reaching limit set by ndb_eventbuffer_max_alloc
- `ndb_eventbuffer_max_alloc`: Maximum memory that can be allocated for buffering events by the NDB API. Defaults to 0 (no limit).
- `Ndb_execute_count`: Provides the number of round trips to the NDB kernel made by operations
- `ndb_extra_logging`: Controls logging of NDB Cluster schema, connection, and data distribution events in the MySQL error log
- `ndb_force_send`: Forces sending of buffers to NDB immediately, without waiting for other threads
- `ndb_fully_replicated`: Whether new NDB tables are fully replicated
- `ndb_index_stat_enable`: Use NDB index statistics in query optimization
- `ndb_index_stat_option`: Comma-separated list of tunable options for NDB index statistics; the list should contain no spaces
- `ndb_join_pushdown`: Enables pushing down of joins to data nodes
- `Ndb_last_commit_epoch_server`: Epoch most recently committed by NDB
- `Ndb_last_commit_epoch_session`: Epoch most recently committed by this NDB client
- `ndb_log_apply_status`: Whether or not a MySQL server acting as a slave logs mysql.ndb_apply_status updates received from its immediate master in its own binary log, using its own server ID

- ndb_log_bin: Write updates to NDB tables in the binary log. Effective only if binary logging is enabled with --log-bin.
- ndb_log_binlog_index: Insert mapping between epochs and binary log positions into the ndb_binlog_index table. Defaults to ON. Effective only if binary logging is enabled on the server.
- ndb_log_empty_epochs: When enabled, epochs in which there were no changes are written to the ndb_apply_status and ndb_binlog_index tables, even when log_slave_updates is enabled.
- ndb_log_empty_update: When enabled, updates which produce no changes are written to the ndb_apply_status and ndb_binlog_index tables, even when log_slave_updates is enabled.
- ndb_log_exclusive_reads: Log primary key reads with exclusive locks; allow conflict resolution based on read conflicts
- ndb_log_orig: Whether the id and epoch of the originating server are recorded in the mysql.ndb_binlog_index table. Set using the --ndb-log-orig option when starting mysqld.
- ndb_log_transaction_id: Whether NDB transaction IDs are written into the binary log (Read-only.)
- ndb-log-update-minimal: Log updates in a minimal format.
- ndb_log_updated_only: Log complete rows (ON) or updates only (OFF)
- ndb_metadata_check: Enable auto-detection of NDB metadata changes with respect to MySQL data dictionary; enabled by default
- ndb_metadata_check_interval: Interval in seconds to perform check for NDB metadata changes with respect to MySQL data dictionary
- Ndb_metadata_blacklist_size: Number of NDB metadata objects that NDB binlog thread has failed to synchronize
- Ndb_metadata_detected_count: Number of times NDB metadata change monitor thread has detected changes
- Ndb_metadata_synced_count: Number of NDB metadata objects which have been synchronized
- Ndb_number_of_data_nodes: If the server is part of an NDB Cluster, the value of this variable is the number of data nodes in the cluster
- ndb-optimization-delay: Sets the number of milliseconds to wait between processing sets of rows by OPTIMIZE TABLE on NDB tables
- ndb_optimized_node_selection: Determines how an SQL node chooses a cluster data node to use as transaction coordinator
- Ndb_pruned_scan_count: Number of scans executed by NDB since the cluster was last started where partition pruning could be used
- Ndb_pushed_queries_defined: Number of joins that API nodes have attempted to push down to the data nodes
- Ndb_pushed_queries_dropped: Number of joins that API nodes have tried to push down, but failed
- Ndb_pushed_queries_executed: Number of joins successfully pushed down and executed on the data nodes
- Ndb_pushed_reads: Number of reads executed on the data nodes by pushed-down joins

- `ndb_read_backup`: Enable read from any replica
- `ndb_recv_thread_activation_threshold`: Activation threshold when receive thread takes over the polling of the cluster connection (measured in concurrently active threads)
- `ndb_recv_thread_cpu_mask`: CPU mask for locking receiver threads to specific CPUs; specified as hexadecimal. See documentation for details.
- `ndb_report_thresh_binlog_epoch_slip`: NDB 7.5.4 and later: Threshold for number of epochs completely buffered, but not yet consumed by binlog injector thread which when exceeded generates BUFFERED_EPOCHS_OVER_THRESHOLD event buffer status message; prior to NDB 7.5.4: Threshold for number of epochs to lag behind before reporting binary log status
- `ndb_report_thresh_binlog_mem_usage`: This is a threshold on the percentage of free memory remaining before reporting binary log status
- `ndb_row_checksum`: When enabled, set row checksums; enabled by default
- `Ndb_scan_count`: The total number of scans executed by NDB since the cluster was last started
- `ndb_schema_dist_lock_wait_timeout`: Time during schema distribution to wait for lock before returning error
- `ndb_schema_dist_timeout`: Time to wait before detecting timeout during schema distribution
- `ndb_schema_dist_upgrade_allowed`: Allow schema distribution table upgrade when connecting to NDB
- `ndb_show_foreign_key_mock_tables`: Show the mock tables used to support foreign_key_checks=0
- `ndb_slave_conflict_role`: Role for slave to play in conflict detection and resolution. Value is one of PRIMARY, SECONDARY, PASS, or NONE (default). Can be changed only when slave SQL thread is stopped. See documentation for further information.
- `Ndb_slave_max_replicated_epoch`: The most recently committed NDB epoch on this slave. When this value is greater than or equal to Ndb_conflict_last_conflict_epoch, no conflicts have yet been detected.
- `Ndb_system_name`: Configured cluster system name; empty if server not connected to NDB
- `ndb_table_no_logging`: NDB tables created when this setting is enabled are not checkpointed to disk (although table schema files are created). The setting in effect when the table is created with or altered to use NDBCLUSTER persists for the lifetime of the table.
- `ndb_table_temporary`: NDB tables are not persistent on disk: no schema files are created and the tables are not logged
- `Ndb_trans_hint_count_session`: Number of transactions using hints that have been started in this session
- `ndb_use_exact_count`: Use exact row count when planning queries
- `ndb_use_transactions`: Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query
- `ndb_version`: Shows build and NDB engine version as an integer
- `ndb_version_string`: Shows build information including NDB engine version in ndb-x.y.z format

- `ndbcluster`: Enable NDB Cluster (if this version of MySQL supports it) Disabled by `--skip-ndbcluster`
- `ndbinfo`: Enable ndbinfo plugin, if supported
- `ndbinfo_database`: The name used for the NDB information database; read only
- `ndbinfo_max_bytes`: Used for debugging only
- `ndbinfo_max_rows`: Used for debugging only
- `ndbinfo_offline`: Put the ndbinfo database into offline mode, in which no rows are returned from tables or views
- `ndbinfo_show_hidden`: Whether to show ndbinfo internal base tables in the mysql client. The default is OFF.
- `ndbinfo_table_prefix`: The prefix to use for naming ndbinfo internal base tables
- `ndbinfo_version`: The version of the ndbinfo engine; read only
- `server-id-bits`: Sets the number of least significant bits in the server_id actually used for identifying the server, permitting NDB API applications to store application data in the most significant bits. server_id must be less than 2 to the power of this value.
- `slave_allow_batching`: Turns update batching on and off for a replication slave
- `transaction_allow_batching`: Allows batching of statements within a transaction. Disable AUTOCOMMIT to use.

22.3.3 NDB Cluster Configuration Files

Configuring NDB Cluster requires working with two files:

- `my.cnf`: Specifies options for all NDB Cluster executables. This file, with which you should be familiar with from previous work with MySQL, must be accessible by each executable running in the cluster.
- `config.ini`: This file, sometimes known as the *global configuration file*, is read only by the NDB Cluster management server, which then distributes the information contained therein to all processes participating in the cluster. `config.ini` contains a description of each node involved in the cluster. This includes configuration parameters for data nodes and configuration parameters for connections between all nodes in the cluster. For a quick reference to the sections that can appear in this file, and what sorts of configuration parameters may be placed in each section, see Sections of the `config.ini` File.

Caching of configuration data. `NDB` uses *stateful configuration*. Rather than reading the global configuration file every time the management server is restarted, the management server caches the configuration the first time it is started, and thereafter, the global configuration file is read only when one of the following conditions is true:

- **The management server is started using the --initial option.** When `--initial` is used, the global configuration file is re-read, any existing cache files are deleted, and the management server creates a new configuration cache.
- **The management server is started using the --reload option.** The `--reload` option causes the management server to compare its cache with the global configuration file. If they differ, the management server creates a new configuration cache; any existing configuration cache is preserved, but not used. If the management server's cache and the global configuration file contain the same configuration data, then the existing cache is used, and no new cache is created.

- **The management server is started using --config-cache=FALSE.** This disables `--config-cache` (enabled by default), and can be used to force the management server to bypass configuration caching altogether. In this case, the management server ignores any configuration files that may be present, always reading its configuration data from the `config.ini` file instead.
- **No configuration cache is found.** In this case, the management server reads the global configuration file and creates a cache containing the same configuration data as found in the file.

Configuration cache files. The management server by default creates configuration cache files in a directory named `mysql-cluster` in the MySQL installation directory. (If you build NDB Cluster from source on a Unix system, the default location is `/usr/local/mysql-cluster`.) This can be overridden at runtime by starting the management server with the `--configdir` option. Configuration cache files are binary files named according to the pattern `ndb_node_id_config.bin.seq_id`, where *`node_id`* is the management server's node ID in the cluster, and *`seq_id`* is a cache idenitifer. Cache files are numbered sequentially using *`seq_id`*, in the order in which they are created. The management server uses the latest cache file as determined by the *`seq_id`*.

Note

It is possible to roll back to a previous configuration by deleting later configuration cache files, or by renaming an earlier cache file so that it has a higher *`seq_id`*. However, since configuration cache files are written in a binary format, you should not attempt to edit their contents by hand.

For more information about the `--configdir`, `--config-cache`, `--initial`, and `--reload` options for the NDB Cluster management server, see Section 22.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”.

We are continuously making improvements in Cluster configuration and attempting to simplify this process. Although we strive to maintain backward compatibility, there may be times when introduce an incompatible change. In such cases we will try to let Cluster users know in advance if a change is not backward compatible. If you find such a change and we have not documented it, please report it in the MySQL bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”.

22.3.3.1 NDB Cluster Configuration: Basic Example

To support NDB Cluster, you will need to update `my.cnf` as shown in the following example. You may also specify these parameters on the command line when invoking the executables.

Note

The options shown here should not be confused with those that are used in `config.ini` global configuration files. Global configuration options are discussed later in this section.

```
# my.cnf
# example additions to my.cnf for NDB Cluster
# (valid in MySQL 8.0)

# enable ndbcluster storage engine, and provide connection string for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com


# provide connection string for management server host (default port: 1186)
```

```
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connection string for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com

# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
```

(For more information on connection strings, see Section 22.3.3.3, “NDB Cluster Connection Strings”.)

```
# my.cnf
# example additions to my.cnf for NDB Cluster
# (will work on all versions)

# enable ndbcluster storage engine, and provide connection string for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Important

Once you have started a `mysqld` process with the `NDBCLUSTER` and `ndb-connectstring` parameters in the `[mysqld]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements will fail with an error. *This is by design*.

You may also use a separate `[mysql_cluster]` section in the cluster `my.cnf` file for settings to be read and used by all executables:

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

For additional `NDB` variables that can be set in the `my.cnf` file, see NDB Cluster System Variables.

The NDB Cluster global configuration file is by convention named `config.ini` (but this is not required). If needed, it is read by `ndb_mgmd` at startup and can be placed in any location that can be read by it. The location and name of the configuration are specified using `--config-file=`*`path_name`* with `ndb_mgmd` on the command line. This option has no default value, and is ignored if `ndb_mgmd` uses the configuration cache.

The global configuration file for NDB Cluster uses INI format, which consists of sections preceded by section headings (surrounded by square brackets), followed by the appropriate parameter names and values. One deviation from the standard INI format is that the parameter name and value can be separated by a colon (`:`) as well as the equal sign (`=`); however, the equal sign is preferred. Another deviation is that sections are not uniquely identified by section name. Instead, unique sections (such as two different nodes of the same type) are identified by a unique ID specified as a parameter within the section.

Default values are defined for most parameters, and can also be specified in `config.ini`. To create a default value section, simply add the word `default` to the section name. For example, an `[ndbd]` section contains parameters that apply to a particular data node, whereas an `[ndbd default]` section contains parameters that apply to all data nodes. Suppose that all data nodes should use the same data memory size. To configure them all, create an `[ndbd default]` section that contains a `DataMemory` line to specify the data memory size.

If used, the `[ndbd default]` section must precede any `[ndbd]` sections in the configuration file. This is also true for `default` sections of any other type.

Note

In some older releases of NDB Cluster, there was no default value for `NoOfReplicas`, which always had to be specified explicitly in the `[ndbd default]` section. Although this parameter now has a default value of 2, which is the recommended setting in most common usage scenarios, it is still recommended practice to set this parameter explicitly.

The global configuration file must define the computers and nodes involved in the cluster and on which computers these nodes are located. An example of a simple configuration file for a cluster consisting of one management server, two data nodes and two MySQL servers is shown here:

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the
# management server)
# The first MySQL Server can be started from any host. The second
# can be started only on the host mysqld_5.mysql.com

[ndbd default]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster

[ndb_mgmd]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster

[ndbd]
HostName= ndbd_2.mysql.com

[ndbd]
HostName= ndbd_3.mysql.com

[mysqld]
[mysqld]
HostName= mysqld_5.mysql.com
```

Note

The preceding example is intended as a minimal starting configuration for purposes of familiarization with NDB Cluster , and is almost certain not to be sufficient for production settings. See Section 22.3.3.2, “Recommended Starting Configuration for NDB Cluster”, which provides a more complete example starting configuration.

Each node has its own section in the `config.ini` file. For example, this cluster has two data nodes, so the preceding configuration file contains two `[ndbd]` sections defining these nodes.

Note

Do not place comments on the same line as a section heading in the `config.ini` file; this causes the management server not to start because it cannot parse the configuration file in such cases.

Sections of the config.ini File

There are six different sections that you can use in the `config.ini` configuration file, as described in the following list:

- `[computer]`: Defines cluster hosts. This is not required to configure a viable NDB Cluster, but be may used as a convenience when setting up a large cluster. See Section 22.3.3.4, “Defining Computers in an NDB Cluster”, for more information.
- `[ndbd]`: Defines a cluster data node (`ndbd` process). See Section 22.3.3.6, “Defining NDB Cluster Data Nodes”, for details.
- `[mysqld]`: Defines the cluster's MySQL server nodes (also called SQL or API nodes). For a discussion of SQL node configuration, see Section 22.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.
- `[mgm]` or `[ndb_mgmd]`: Defines a cluster management server (MGM) node. For information concerning the configuration of management nodes, see Section 22.3.3.5, “Defining an NDB Cluster Management Server”.
- `[tcp]`: Defines a TCP/IP connection between cluster nodes, with TCP/IP being the default connection protocol. Normally, `[tcp]` or `[tcp default]` sections are not required to set up an NDB Cluster, as the cluster handles this automatically; however, it may be necessary in some situations to override the defaults provided by the cluster. See Section 22.3.3.10, “NDB Cluster TCP/IP Connections”, for information about available TCP/IP configuration parameters and how to use them. (You may also find Section 22.3.3.11, “NDB Cluster TCP/IP Connections Using Direct Connections” to be of interest in some cases.)
- `[shm]`: Defines shared-memory connections between nodes. In MySQL 8.0, it is enabled by default, but should still be considered experimental. For a discussion of SHM interconnects, see Section 22.3.3.12, “NDB Cluster Shared-Memory Connections”.
- `[sci]`: Defines Scalable Coherent Interface connections between cluster data nodes. Not supported in NDB 8.0.

You can define `default` values for each section. If used, a `default` section should come before any other sections of that type. For example, an `[ndbd default]` section should appear in the configuration file before any `[ndbd]` sections.

NDB Cluster parameter names are case-insensitive, unless specified in MySQL Server `my.cnf` or `my.ini` files.

22.3.3.2 Recommended Starting Configuration for NDB Cluster

Achieving the best performance from an NDB Cluster depends on a number of factors including the following:

- NDB Cluster software version
- Numbers of data nodes and SQL nodes
- Hardware
- Operating system
- Amount of data to be stored
- Size and type of load under which the cluster is to operate

Therefore, obtaining an optimum configuration is likely to be an iterative process, the outcome of which can vary widely with the specifics of each NDB Cluster deployment. Changes in configuration are also likely to be indicated when changes are made in the platform on which the cluster is run, or in applications that use the NDB Cluster 's data. For these reasons, it is not possible to offer a single configuration that is ideal for all usage scenarios. However, in this section, we provide a recommended base configuration.

Starting config.ini file. The following `config.ini` file is a recommended starting point for configuring a cluster running NDB Cluster 8.0:

```
# TCP PARAMETERS

[tcp default]
SendBufferMemory=2M
ReceiveBufferMemory=2M

# Increasing the sizes of these 2 buffers beyond the default values
# helps prevent bottlenecks due to slow disk I/O.

# MANAGEMENT NODE PARAMETERS

[ndb_mgmd default]
DataDir=path/to/management/server/data/directory

# It is possible to use a different data directory for each management
# server, but for ease of administration it is preferable to be
# consistent.

[ndb_mgmd]
HostName=management-server-A-hostname
# NodeId=management-server-A-nodeid

[ndb_mgmd]
HostName=management-server-B-hostname
# NodeId=management-server-B-nodeid

# Using 2 management servers helps guarantee that there is always an
# arbitrator in the event of network partitioning, and so is
# recommended for high availability. Each management server must be
# identified by a HostName. You may for the sake of convenience specify
# a NodeId for any management server, although one will be allocated
# for it automatically; if you do so, it must be in the range 1-255
# inclusive and must be unique among all IDs specified for cluster
# nodes.

# DATA NODE PARAMETERS

[ndbd default]
NoOfReplicas=2

# Using 2 replicas is recommended to guarantee availability of data;
# using only 1 replica does not provide any redundancy, which means
# that the failure of a single data node causes the entire cluster to
# shut down. We do not recommend using more than 2 replicas, since 2 is
# sufficient to provide high availability, and we do not currently test
# with greater values for this parameter.

LockPagesInMainMemory=1

# On Linux and Solaris systems, setting this parameter locks data node
# processes into memory. Doing so prevents them from swapping to disk,
# which can severely degrade cluster performance.

DataMemory=3456M

# The value provided for DataMemory assumes 4 GB RAM
# per data node. However, for best results, you should first calculate
# the memory that would be used based on the data you actually plan to
# store (you may find the ndb_size.pl utility helpful in estimating
# this), then allow an extra 20% over the calculated values. Naturally,
# you should ensure that each data node host has at least as much
# physical memory as the sum of these two values.
```

```
# ODirect=1

# Enabling this parameter causes NDBCLUSTER to try using O_DIRECT
# writes for local checkpoints and redo logs; this can reduce load on
# CPUs. We recommend doing so when using NDB Cluster on systems running
# Linux kernel 2.6 or later.

NoOfFragmentLogFiles=300
DataDir=path/to/data/node/data/directory
MaxNoOfConcurrentOperations=100000

SchedulerSpinTimer=400
SchedulerExecutionTimer=100
RealTimeScheduler=1
# Setting these parameters allows you to take advantage of real-time scheduling
# of NDB threads to achieve increased throughput when using ndbd. They
# are not needed when using ndbmtd; in particular, you should not set
# RealTimeScheduler for ndbmtd data nodes.

TimeBetweenGlobalCheckpoints=1000
TimeBetweenEpochs=200
RedoBuffer=32M

# CompressedLCP=1
# CompressedBackup=1
# Enabling CompressedLCP and CompressedBackup causes, respectively, local
checkpoint files and backup files to be compressed, which can result in a space
savings of up to 50% over noncompressed LCPs and backups.

# MaxNoOfLocalScans=64
MaxNoOfTables=1024
MaxNoOfOrderedIndexes=256

[ndbd]
HostName=data-node-A-hostname
# NodeId=data-node-A-nodeid

LockExecuteThreadToCPU=1
LockMaintThreadsToCPU=0
# On systems with multiple CPUs, these parameters can be used to lock NDBCLUSTER
# threads to specific CPUs

[ndbd]
HostName=data-node-B-hostname
# NodeId=data-node-B-nodeid

LockExecuteThreadToCPU=1
LockMaintThreadsToCPU=0

# You must have an [ndbd] section for every data node in the cluster;
# each of these sections must include a HostName. Each section may
# optionally include a NodeId for convenience, but in most cases, it is
# sufficient to allow the cluster to allocate node IDs dynamically. If
# you do specify the node ID for a data node, it must be in the range 1
# to 144 inclusive and must be unique among all IDs specified for
# cluster nodes. (Previous to NDB 8.0.18, this range was 1 to 48 inclusive.)

# SQL NODE / API NODE PARAMETERS

[mysqld]
# HostName=sql-node-A-hostname
# NodeId=sql-node-A-nodeid

[mysqld]

[mysqld]
```

```
# Each API or SQL node that connects to the cluster requires a [mysqld]
# or [api] section of its own. Each such section defines a connection
# “slot”; you should have at least as many of these sections in the
# config.ini file as the total number of API nodes and SQL nodes that
# you wish to have connected to the cluster at any given time. There is
# no performance or other penalty for having extra slots available in
# case you find later that you want or need more API or SQL nodes to
# connect to the cluster at the same time.
# If no HostName is specified for a given [mysqld] or [api] section,
# then any API or SQL node may use that slot to connect to the
# cluster. You may wish to use an explicit HostName for one connection slot
# to guarantee that an API or SQL node from that host can always
# connect to the cluster. If you wish to prevent API or SQL nodes from
# connecting from other than a desired host or hosts, then use a
# HostName for every [mysqld] or [api] section in the config.ini file.
# You can if you wish define a node ID (NodeId parameter) for any API or
# SQL node, but this is not necessary; if you do so, it must be in the
# range 1 to 255 inclusive and must be unique among all IDs specified
# for cluster nodes.
```

Recommended my.cnf options for SQL nodes. MySQL Servers acting as NDB Cluster SQL nodes must always be started with the `--ndbcluster` and `--ndb-connectstring` options, either on the command line or in `my.cnf`. In addition, set the following options for all `mysqld` processes in the cluster, unless your setup requires otherwise:

- `--ndb-use-exact-count=0`
- `--ndb-index-stat-enable=0`
- `--ndb-force-send=1`
- `--optimizer-switch=engine_condition_pushdown=on`

22.3.3.3 NDB Cluster Connection Strings

With the exception of the NDB Cluster management server (`ndb_mgmd`), each node that is part of an NDB Cluster requires a *connection string* that points to the management server's location. This connection string is used in establishing a connection to the management server as well as in performing other tasks depending on the node's role in the cluster. The syntax for a connection string is as follows:

```
[nodeid=node_id, ]host-definition[, host-definition[, ...]]

host-definition:
    host_name[:port_number]
```

`node_id` is an integer greater than or equal to 1 which identifies a node in `config.ini`. *`host_name`* is a string representing a valid Internet host name or IP address. *`port_number`* is an integer referring to a TCP/IP port number.

```
example 1 (long):    "nodeid=2,myhost1:1100,myhost2:1100,198.51.100.3:1200"
example 2 (short):   "myhost1"
```

`localhost:1186` is used as the default connection string value if none is provided. If *`port_num`* is omitted from the connection string, the default port is 1186. This port should always be available on the network because it has been assigned by IANA for this purpose (see http://www.iana.org/assignments/port-numbers for details).

By listing multiple host definitions, it is possible to designate several redundant management servers. An NDB Cluster data or API node attempts to contact successive management servers on each host in the order specified, until a successful connection has been established.

It is also possible to specify in a connection string one or more bind addresses to be used by nodes having multiple network interfaces for connecting to management servers. A bind address consists of a hostname or network address and an optional port number. This enhanced syntax for connection strings is shown here:

```
[nodeid=node_id, ]
    [bind-address=host-definition, ]
    host-definition[; bind-address=host-definition]
    host-definition[; bind-address=host-definition]
    [, ...]]

host-definition:
    host_name[:port_number]
```

If a single bind address is used in the connection string *prior* to specifying any management hosts, then this address is used as the default for connecting to any of them (unless overridden for a given management server; see later in this section for an example). For example, the following connection string causes the node to use `198.51.100.242` regardless of the management server to which it connects:

```
bind-address=198.51.100.242, poseidon:1186, perch:1186
```

If a bind address is specified *following* a management host definition, then it is used only for connecting to that management node. Consider the following connection string:

```
poseidon:1186;bind-address=localhost, perch:1186;bind-address=198.51.100.242
```

In this case, the node uses `localhost` to connect to the management server running on the host named `poseidon` and `198.51.100.242` to connect to the management server running on the host named `perch`.

You can specify a default bind address and then override this default for one or more specific management hosts. In the following example, `localhost` is used for connecting to the management server running on host `poseidon`; since `198.51.100.242` is specified first (before any management server definitions), it is the default bind address and so is used for connecting to the management servers on hosts `perch` and `orca`:

```
bind-address=198.51.100.242,poseidon:1186;bind-address=localhost,perch:1186,orca:2200
```

There are a number of different ways to specify the connection string:

- Each executable has its own command-line option which enables specifying the management server at startup. (See the documentation for the respective executable.)
- It is also possible to set the connection string for all nodes in the cluster at once by placing it in a `[mysql_cluster]` section in the management server's `my.cnf` file.
- For backward compatibility, two other options are available, using the same syntax:

  1. Set the `NDB_CONNECTSTRING` environment variable to contain the connection string.
  2. Write the connection string for each executable into a text file named `Ndb.cfg` and place this file in the executable's startup directory.

  However, these are now deprecated and should not be used for new installations.

The recommended method for specifying the connection string is to set it on the command line or in the `my.cnf` file for each executable.

22.3.3.4 Defining Computers in an NDB Cluster

The `[computer]` section has no real significance other than serving as a way to avoid the need of defining host names for each node in the system. All parameters mentioned here are required.

- `Id`

Table 22.7 This table provides type and value information for the Id computer configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	string
Default	[none]
Range	...
Restart Type	IS

This is a unique identifier, used to refer to the host computer elsewhere in the configuration file.

Important

The computer ID is *not* the same as the node ID used for a management, API, or data node. Unlike the case with node IDs, you cannot use `NodeId` in place of `Id` in the `[computer]` section of the `config.ini` file.

- `HostName`

Table 22.8 This table provides type and value information for the HostName computer configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

This is the computer's hostname or IP address.

22.3.3.5 Defining an NDB Cluster Management Server

The `[ndb_mgmd]` section is used to configure the behavior of the management server. If multiple management servers are employed, you can specify parameters common to all of them in an `[ndb_mgmd default]` section. `[mgm]` and `[mgm default]` are older aliases for these, supported for backward compatibility.

All parameters in the following list are optional and assume their default values if omitted.

Note

If neither the `ExecuteOnComputer` nor the `HostName` parameter is present, the default value `localhost` will be assumed for both.

- `Id`

Table 22.9 This table provides type and value information for the Id management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	[none]
Range	1 - 255
Restart Type	IS

Each node in the cluster has a unique identity. For a management node, this is represented by an integer value in the range 1 to 255, inclusive. This ID is used by all internal cluster messages for addressing the node, and so must be unique for each NDB Cluster node, regardless of the type of node.

Note

Data node IDs must be less than 145. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for management nodes (and API nodes) to values greater than 144. (In NDB 8.0.17 and earlier, the maximum value for a data node ID was 48.)

The use of the `Id` parameter for identifying management nodes is deprecated in favor of `NodeId`. Although `Id` continues to be supported for backward compatibility, it now generates a warning and is subject to removal in a future version of NDB Cluster.

- `NodeId`

Table 22.10 This table provides type and value information for the NodeId management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	[none]
Range	1 - 255
Restart Type	IS

Each node in the cluster has a unique identity. For a management node, this is represented by an integer value in the range 1 to 255 inclusive. This ID is used by all internal cluster messages for addressing the node, and so must be unique for each NDB Cluster node, regardless of the type of node.

Note

As of NDB 8.0.18, data node IDs must be less than 145. (Previously, this was less than 49.) If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for management nodes (and API nodes) to values greater than 144.

`NodeId` is the preferred parameter name to use when identifying management nodes. Although the older `Id` continues to be supported for backward compatibility, it is now deprecated and generates a warning when used; it is also subject to removal in a future NDB Cluster release.

- `ExecuteOnComputer`

Table 22.11 This table provides type and value information for the ExecuteOnComputer management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name
Default	[none]
Range	...
Restart Type	S

This refers to the `Id` set for one of the computers defined in a `[computer]` section of the `config.ini` file.

Important

This parameter is deprecated, and is subject to removal in a future release. Use the `HostName` parameter instead.

- `PortNumber`

Table 22.12 This table provides type and value information for the PortNumber management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	1186
Range	0 - 64K
Restart Type	S

This is the port number on which the management server listens for configuration requests and management commands.

- `HostName`

Table 22.13 This table provides type and value information for the HostName management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

Specifying this parameter defines the hostname of the computer on which the management node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `LocationDomainId`

Table 22.14 This table provides type and value information for the LocationDomainId management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 16
Restart Type	S

Assigns a management node to a specific availability domain (also known as an availability zone) within a cloud. By informing `NDB` which nodes are in which availability domains, performance can be improved in a cloud environment in the following ways:

- If requested data is not found on the same node, reads can be directed to another node in the same availability domain.
- Communication between nodes in different availability domains are guaranteed to use `NDB` transporters' WAN support without any further manual intervention.
- The transporter's group number can be based on which availability domain is used, such that also SQL and other API nodes communicate with local data nodes in the same availability domain whenever possible.
- The arbitrator can be selected from an availability domain in which no data nodes are present, or, if no such availability domain can be found, from a third availability domain.

`LocationDomainId` takes an integer value between 0 and 16 inclusive, with 0 being the default; using 0 is the same as leaving the parameter unset.

- `LogDestination`

Table 22.15 This table provides type and value information for the LogDestination management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	{CONSOLE|SYSLOG|FILE}
Default	[see text]
Range	...
Restart Type	N

This parameter specifies where to send cluster logging information. There are three options in this regard—`CONSOLE`, `SYSLOG`, and `FILE`—with `FILE` being the default:

- `CONSOLE` outputs the log to `stdout`:

```
CONSOLE
```

- `SYSLOG` sends the log to a `syslog` facility, possible values being one of `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `news`, `syslog`, `user`, `uucp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, or `local7`.

  Note

  Not every facility is necessarily supported by every operating system.

```
SYSLOG:facility=syslog
```

- `FILE` pipes the cluster log output to a regular file on the same machine. The following values can be specified:

  - `filename`: The name of the log file.

    The default log file name used in such cases is `ndb_nodeid_cluster.log`.

  - `maxsize`: The maximum size (in bytes) to which the file can grow before logging rolls over to a new file. When this occurs, the old log file is renamed by appending `.N` to the file name, where `N` is the next number not yet used with this name.

  - `maxfiles`: The maximum number of log files.

```
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
```

  The default value for the `FILE` parameter is `FILE:filename=ndb_node_id_cluster.log,maxsize=1000000,maxfiles=6`, where `node_id` is the ID of the node.

It is possible to specify multiple log destinations separated by semicolons as shown here:

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

- `ArbitrationRank`

Table 22.16 This table provides type and value information for the ArbitrationRank management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	0-2
Default	1
Range	0 - 2
Restart Type	N

This parameter is used to define which nodes can act as arbitrators. Only management nodes and SQL nodes can be arbitrators. `ArbitrationRank` can take one of the following values:

- `0`: The node will never be used as an arbitrator.

- 1: The node has high priority; that is, it will be preferred as an arbitrator over low-priority nodes.
- 2: Indicates a low-priority node which be used as an arbitrator only if a node with a higher priority is not available for that purpose.

Normally, the management server should be configured as an arbitrator by setting its ArbitrationRank to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

You can disable arbitration completely either by setting ArbitrationRank to 0 on all management and SQL nodes, or by setting the Arbitration parameter in the [ndbd default] section of the config.ini global configuration file. Setting Arbitration causes any settings for ArbitrationRank to be disregarded.

- ArbitrationDelay

Table 22.17 This table provides type and value information for the ArbitrationDelay management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

An integer value which causes the management server's responses to arbitration requests to be delayed by that number of milliseconds. By default, this value is 0; it is normally not necessary to change it.

- DataDir

Table 22.18 This table provides type and value information for the DataDir management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	path
Default	.
Range	...
Restart Type	N

This specifies the directory where output files from the management server will be placed. These files include cluster log files, process output files, and the daemon's process ID (PID) file. (For log files, this location can be overridden by setting the FILE parameter for LogDestination as discussed previously in this section.)

The default value for this parameter is the directory in which ndb_mgmd is located.

- PortNumberStats

Table 22.19 This table provides type and value information for the PortNumberStats management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	[none]
Range	0 - 64K
Restart Type	N

This parameter specifies the port number used to obtain statistical information from an NDB Cluster management server. It has no default value.

- `Wan`

Table 22.20 This table provides type and value information for the wan management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Use WAN TCP setting as default.

- `HeartbeatThreadPriority`

Table 22.21 This table provides type and value information for the HeartbeatThreadPriority management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	string
Default	[none]
Range	...
Restart Type	S

Set the scheduling policy and priority of heartbeat threads for management and API nodes.

The syntax for setting this parameter is shown here:

```
HeartbeatThreadPriority = policy[, priority]

policy:
  {FIFO | RR}
```

When setting this parameter, you must specify a policy. This is one of `FIFO` (first in, first out) or `RR` (round robin). The policy value is followed optionally by the priority (an integer).

- `TotalSendBufferMemory`

Table 22.22 This table provides type and value information for the TotalSendBufferMemory management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	0
Range	256K - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

If this parameter is set, its minimum permitted value is 256KB; 0 indicates that the parameter has not been set. For more detailed information, see Section 22.3.3.13, “Configuring NDB Cluster Send Buffer Parameters”.

- `HeartbeatIntervalMgmdMgmd`

Table 22.23 This table provides type and value information for the HeartbeatIntervalMgmdMgmd management node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	1500
Range	100 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Specify the interval between heartbeat messages used to determine whether another management node is on contact with this one. The management node waits after 3 of these intervals to declare the connection dead; thus, the default setting of 1500 milliseconds causes the management node to wait for approximately 1600 ms before timing out.

Note

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect.

To add new management servers to a running NDB Cluster, it is also necessary to perform a rolling restart of all cluster nodes after modifying any existing `config.ini` files. For more information about issues arising when using multiple management nodes, see Section 22.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”.

22.3.3.6 Defining NDB Cluster Data Nodes

The [ndbd] and [ndbd default] sections are used to configure the behavior of the cluster's data nodes.

[ndbd] and [ndbd default] are always used as the section names whether you are using ndbd or ndbmtd binaries for the data node processes.

There are many parameters which control buffer sizes, pool sizes, timeouts, and so forth. The only mandatory parameter is either one of ExecuteOnComputer or HostName; this must be defined in the local [ndbd] section.

The parameter NoOfReplicas should be defined in the [ndbd default] section, as it is common to all Cluster data nodes. It is not strictly necessary to set NoOfReplicas, but it is good practice to set it explicitly.

Most data node parameters are set in the [ndbd default] section. Only those parameters explicitly stated as being able to set local values are permitted to be changed in the [ndbd] section. Where present, HostName, NodeId and ExecuteOnComputer *must* be defined in the local [ndbd] section, and not in any other section of config.ini. In other words, settings for these parameters are specific to one data node.

For those parameters affecting memory usage or buffer sizes, it is possible to use K, M, or G as a suffix to indicate units of 1024, 1024×1024, or 1024×1024×1024. (For example, 100K means 100 × 1024 = 102400.)

Parameter names and values are case-insensitive, unless used in a MySQL Server my.cnf or my.ini file, in which case they are case sensitive.

Information about configuration parameters specific to NDB Cluster Disk Data tables can be found later in this section (see Disk Data Configuration Parameters).

All of these parameters also apply to ndbmtd (the multithreaded version of ndbd). Three additional data node configuration parameters—MaxNoOfExecutionThreads, ThreadConfig, and NoOfFragmentLogParts—apply to ndbmtd only; these have no effect when used with ndbd. For more information, see Multi-Threading Configuration Parameters (ndbmtd). See also Section 22.4.3, “ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded)”.

Identifying data nodes. The NodeId or Id value (that is, the data node identifier) can be allocated on the command line when the node is started or in the configuration file.

- NodeId

Table 22.24 This table provides type and value information for the NodeId data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	[none]
Range	1 - 48
Restart Type	IS
Version (or later)	NDB 8.0.18
Type or units	unsigned
Default	[none]
Range	1 - 144

Property	Value
Restart Type	IS

A unique node ID is used as the node's address for all cluster internal messages. For data nodes, this is an integer in the range 1 to 144 inclusive. (In NDB 8.0.17 and earlier, this was 1 to 48 inclusive.) Each node in the cluster must have a unique identifier.

`NodeId` is the only supported parameter name to use when identifying data nodes.

- `ExecuteOnComputer`

Table 22.25 This table provides type and value information for the ExecuteOnComputer data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name
Default	[none]
Range	...
Restart Type	S

This refers to the `Id` set for one of the computers defined in a `[computer]` section.

Important

This parameter is deprecated, and is subject to removal in a future release. Use the `HostName` parameter instead.

- `HostName`

Table 22.26 This table provides type and value information for the HostName data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name or IP address
Default	localhost
Range	...
Restart Type	N

Specifying this parameter defines the hostname of the computer on which the data node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `ServerPort`

Table 22.27 This table provides type and value information for the ServerPort data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned

Property	Value
Default	[none]
Range	1 - 64K
Restart Type	S

Each node in the cluster uses a port to connect to other nodes. By default, this port is allocated dynamically in such a way as to ensure that no two nodes on the same host computer receive the same port number, so it should normally not be necessary to specify a value for this parameter.

However, if you need to be able to open specific ports in a firewall to permit communication between data nodes and API nodes (including SQL nodes), you can set this parameter to the number of the desired port in an `[ndbd]` section or (if you need to do this for multiple data nodes) the `[ndbd default]` section of the `config.ini` file, and then open the port having that number for incoming connections from SQL nodes, API nodes, or both.

Note

Connections from data nodes to management nodes is done using the `ndb_mgmd` management port (the management server's `PortNumber`) so outgoing connections to that port from any data nodes should always be permitted.

- `TcpBind_INADDR_ANY`

  Setting this parameter to `TRUE` or `1` binds `IP_ADDR_ANY` so that connections can be made from anywhere (for autogenerated connections). The default is `FALSE` (`0`).

- `NodeGroup`

Table 22.28 This table provides type and value information for the NodeGroup data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	
Default	[none]
Range	0 - 65536
Restart Type	IS

This parameter can be used to assign a data node to a specific node group. It is read only when the cluster is started for the first time, and cannot be used to reassign a data node to a different node group online. It is generally not desirable to use this parameter in the `[ndbd default]` section of the `config.ini` file, and care must be taken not to assign nodes to node groups in such a way that an invalid numbers of nodes are assigned to any node groups.

The `NodeGroup` parameter is chiefly intended for use in adding a new node group to a running NDB Cluster without having to perform a rolling restart. For this purpose, you should set it to 65536 (the maximum value). You are not required to set a `NodeGroup` value for all cluster data nodes, only for those nodes which are to be started and added to the cluster as a new node group at a later time. For more information, see Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”.

- `LocationDomainId`

Table 22.29 This table provides type and value information for the LocationDomainId data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 16
Restart Type	S

Assigns a data node to a specific availability domain (also known as an availability zone) within a cloud. By informing `NDB` which nodes are in which availability domains, performance can be improved in a cloud environment in the following ways:

- If requested data is not found on the same node, reads can be directed to another node in the same availability domain.
- Communication between nodes in different availability domains are guaranteed to use `NDB` transporters' WAN support without any further manual intervention.
- The transporter's group number can be based on which availability domain is used, such that also SQL and other API nodes communicate with local data nodes in the same availability domain whenever possible.
- The arbitrator can be selected from an availability domain in which no data nodes are present, or, if no such availability domain can be found, from a third availability domain.

`LocationDomainId` takes an integer value between 0 and 16 inclusive, with 0 being the default; using 0 is the same as leaving the parameter unset.

- `NoOfReplicas`

Table 22.30 This table provides type and value information for the NoOfReplicas data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	2
Range	1 - 4
Restart Type	IS

This global parameter can be set only in the `[ndbd default]` section, and defines the number of replicas for each table stored in the cluster. This parameter also specifies the size of node groups. A node group is a set of nodes all storing the same information.

Node groups are formed implicitly. The first node group is formed by the set of data nodes with the lowest node IDs, the next node group by the set of the next lowest node identities, and so on. By way of example, assume that we have 4 data nodes and that `NoOfReplicas` is set to 2. The four data nodes have node IDs 2, 3, 4 and 5. Then the first node group is formed from nodes 2 and 3, and the second node group by nodes 4 and 5. It is important to configure the cluster in such a manner that nodes in the

same node groups are not placed on the same computer because a single hardware failure would cause the entire cluster to fail.

If no node IDs are provided, the order of the data nodes will be the determining factor for the node group. Whether or not explicit assignments are made, they can be viewed in the output of the management client's `SHOW` command.

The default value for `NoOfReplicas` is 2. *This is the recommended value for most production environments*.

Important

While the maximum *possible* value for this parameter is 4, **setting `NoOfReplicas` to a value greater than 2 is not supported in production**.

Warning

Setting `NoOfReplicas` to 1 means that there is only a single copy of all Cluster data; in this case, the loss of a single data node causes the cluster to fail because there are no additional copies of the data stored by that node.

The value for this parameter must divide evenly into the number of data nodes in the cluster. For example, if there are two data nodes, then `NoOfReplicas` must be equal to either 1 or 2, since 2/3 and 2/4 both yield fractional values; if there are four data nodes, then `NoOfReplicas` must be equal to 1, 2, or 4.

- `DataDir`

Table 22.31 This table provides type and value information for the DataDir data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	path
Default	.
Range	...
Restart Type	IN

This parameter specifies the directory where trace files, log files, pid files and error logs are placed.

The default is the data node process working directory.

- `FileSystemPath`

Table 22.32 This table provides type and value information for the FileSystemPath data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	path
Default	DataDir
Range	...
Restart Type	IN

This parameter specifies the directory where all files created for metadata, REDO logs, UNDO logs (for Disk Data tables), and data files are placed. The default is the directory specified by `DataDir`.

Note

This directory must exist before the `ndbd` process is initiated.

The recommended directory hierarchy for NDB Cluster includes `/var/lib/mysql-cluster`, under which a directory for the node's file system is created. The name of this subdirectory contains the node ID. For example, if the node ID is 2, this subdirectory is named `ndb_2_fs`.

- `BackupDataDir`

Table 22.33 This table provides type and value information for the BackupDataDir data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	path
Default	[see text]
Range	...
Restart Type	IN

This parameter specifies the directory in which backups are placed.

Important

The string '`/BACKUP`' is always appended to this value. For example, if you set the value of `BackupDataDir` to `/var/lib/cluster-data`, then all backups are stored under `/var/lib/cluster-data/BACKUP`. This also means that the *effective* default backup location is the directory named `BACKUP` under the location specified by the `FileSystemPath` parameter.

Data Memory, Index Memory, and String Memory

`DataMemory` and `IndexMemory` are `[ndbd]` parameters specifying the size of memory segments used to store the actual records and their indexes. In setting values for these, it is important to understand how `DataMemory` is used, as it usually needs to be updated to reflect actual usage by the cluster.

Note

`IndexMemory` is deprecated, and subject to removal in a future version of NDB Cluster. See the descriptions that follow for further information.

- `DataMemory`

Table 22.34 This table provides type and value information for the DataMemory data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes

Property	Value
Default	98M
Range	1M - 1T
Restart Type	N

This parameter defines the amount of space (in bytes) available for storing database records. The entire amount specified by this value is allocated in memory, so it is extremely important that the machine has sufficient physical memory to accommodate it.

The memory allocated by `DataMemory` is used to store both the actual records and indexes. There is a 16-byte overhead on each record; an additional amount for each record is incurred because it is stored in a 32KB page with 128 byte page overhead (see below). There is also a small amount wasted per page due to the fact that each record is stored in only one page.

For variable-size table attributes, the data is stored on separate data pages, allocated from `DataMemory`. Variable-length records use a fixed-size part with an extra overhead of 4 bytes to reference the variable-size part. The variable-size part has 2 bytes overhead plus 2 bytes per attribute.

As of NDB 8.0.18, the maximum record size is 30000 bytes. Previously, this was 14000 bytes.

Resources assigned to `DataMemory` are used for storing all data and indexes. (Any memory configured as `IndexMemory` is automatically added to that used by `DataMemory` to form a common resource pool.)

Currently, NDB Cluster can use a maximum of 512 MB for hash indexes per partition, which means in some cases it is possible to get `Table is full` errors in MySQL client applications even when `ndb_mgm -e "ALL REPORT MEMORYUSAGE"` shows significant free `DataMemory`. This can also pose a problem with data node restarts on nodes that are heavily loaded with data.

You can control the number of partitions per local data manager for a given table by setting the `NDB_TABLE` option `PARTITION_BALANCE` to one of the values `FOR_RA_BY_LDM`, `FOR_RA_BY_LDM_X_2`, `FOR_RA_BY_LDM_X_3`, or `FOR_RA_BY_LDM_X_4`, for 1, 2, 3, or 4 partitions per LDM, respectively, when creating the table (see Section 13.1.20.11, “Setting NDB_TABLE Options”).

Note

In previous versions of NDB Cluster it was possible to create extra partitions for NDB Cluster tables and thus have more memory available for hash indexes by using the `MAX_ROWS` option for `CREATE TABLE`. While still supported for backward compatibility, using `MAX_ROWS` for this purpose is deprecated; you should use `PARTITION_BALANCE` instead.

You can also use the `MinFreePct` configuration parameter to help avoid problems with node restarts.

The memory space allocated by `DataMemory` consists of 32KB pages, which are allocated to table fragments. Each table is normally partitioned into the same number of fragments as there are data nodes in the cluster. Thus, for each node, there are the same number of fragments as are set in `NoOfReplicas`.

Once a page has been allocated, it is currently not possible to return it to the pool of free pages, except by deleting the table. (This also means that `DataMemory` pages, once allocated to a given table, cannot be used by other tables.) Performing a data node recovery also compresses the partition because all records are inserted into empty partitions from other live nodes.

The DataMemory memory space also contains UNDO information: For each update, a copy of the unaltered record is allocated in the DataMemory. There is also a reference to each copy in the ordered table indexes. Unique hash indexes are updated only when the unique index columns are updated, in which case a new entry in the index table is inserted and the old entry is deleted upon commit. For this reason, it is also necessary to allocate enough memory to handle the largest transactions performed by applications using the cluster. In any case, performing a few large transactions holds no advantage over using many smaller ones, for the following reasons:

- Large transactions are not any faster than smaller ones
- Large transactions increase the number of operations that are lost and must be repeated in event of transaction failure
- Large transactions use more memory

The default value for DataMemory in NDB 8.0 is 98MB. The minimum value is 1MB. There is no maximum size, but in reality the maximum size has to be adapted so that the process does not start swapping when the limit is reached. This limit is determined by the amount of physical RAM available on the machine and by the amount of memory that the operating system may commit to any one process. 32-bit operating systems are generally limited to 2−4GB per process; 64-bit operating systems can use more. For large databases, it may be preferable to use a 64-bit operating system for this reason.

- IndexMemory

Table 22.35 This table provides type and value information for the IndexMemory data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	0
Range	1M - 1T
Restart Type	N

The IndexMemory parameter is deprecated (and subject to future removal); any memory assigned to IndexMemory is allocated instead to the same pool as DataMemory, which is solely responsible for all resources needed for storing data and indexes in memory. In NDB 8.0, the use of IndexMemory in the cluster configuration file triggers a warning from the management server.

You can estimate the size of a hash index using this formula:

```
size  = ( (fragments * 32K) + (rows * 18) )
        * replicas
```

fragments is the number of fragments, *replicas* is the number of replicas (normally 2), and *rows* is the number of rows. If a table has one million rows, 8 fragments, and 2 replicas, the expected index memory usage is calculated as shown here:

```
((8 * 32K) + (1000000 * 18)) * 2 = ((8 * 32768) + (1000000 * 18)) * 2
= (262144 + 18000000) * 2
= 18262144 * 2 = 36524288 bytes = ~35MB
```

Index statistics for ordered indexes (when these are enabled) are stored in the `mysql.ndb_index_stat_sample` table. Since this table has a hash index, this adds to index memory usage. An upper bound to the number of rows for a given ordered index can be calculated as follows:

```
sample_size= key_size + ((key_attributes + 1) * 4)

sample_rows = IndexStatSaveSize
              * ((0.01 * IndexStatSaveScale * log2(rows * sample_size)) + 1)
              / sample_size
```

In the preceding formula, *`key_size`* is the size of the ordered index key in bytes, *`key_attributes`* is the number ot attributes in the ordered index key, and *`rows`* is the number of rows in the base table.

Assume that table `t1` has 1 million rows and an ordered index named `ix1` on two four-byte integers. Assume in addition that `IndexStatSaveSize` and `IndexStatSaveScale` are set to their default values (32K and 100, respectively). Using the previous 2 formulas, we can calculate as follows:

```
sample_size = 8  + ((1 + 2) * 4) = 20 bytes

sample_rows = 32K
              * ((0.01 * 100 * log2(1000000*20)) + 1)
              / 20
              = 32768 * ( (1 * ~16.811) +1) / 20
              = 32768 * ~17.811 / 20
              = ~29182 rows
```

The expected index memory usage is thus 2 * 18 * 29182 = ~1050550 bytes.

In NDB 8.0, the minimum and default vaue for this parameter is 0 (zero).

- `StringMemory`

Table 22.36 This table provides type and value information for the StringMemory data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	% or bytes
Default	25
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	S

This parameter determines how much memory is allocated for strings such as table names, and is specified in an `[ndbd]` or `[ndbd default]` section of the `config.ini` file. A value between `0` and `100` inclusive is interpreted as a percent of the maximum default value, which is calculated based on a number of factors including the number of tables, maximum table name size, maximum size of `.FRM` files, `MaxNoOfTriggers`, maximum column name size, and maximum default column value.

A value greater than `100` is interpreted as a number of bytes.

The default value is 25—that is, 25 percent of the default maximum.

Under most circumstances, the default value should be sufficient, but when you have a great many `NDB` tables (1000 or more), it is possible to get Error 773 `Out of string memory, please modify`

`StringMemory config parameter: Permanent error: Schema error`, in which case you should increase this value. `25` (25 percent) is not excessive, and should prevent this error from recurring in all but the most extreme conditions.

The following example illustrates how memory is used for a table. Consider this table definition:

```
CREATE TABLE example (
  a INT NOT NULL,
  b INT NOT NULL,
  c INT NOT NULL,
  PRIMARY KEY(a),
  UNIQUE(b)
) ENGINE=NDBCLUSTER;
```

For each record, there are 12 bytes of data plus 12 bytes overhead. Having no nullable columns saves 4 bytes of overhead. In addition, we have two ordered indexes on columns `a` and `b` consuming roughly 10 bytes each per record. There is a primary key hash index on the base table using roughly 29 bytes per record. The unique constraint is implemented by a separate table with `b` as primary key and `a` as a column. This other table consumes an additional 29 bytes of index memory per record in the `example` table as well 8 bytes of record data plus 12 bytes of overhead.

Thus, for one million records, we need 58MB for index memory to handle the hash indexes for the primary key and the unique constraint. We also need 64MB for the records of the base table and the unique index table, plus the two ordered index tables.

You can see that hash indexes takes up a fair amount of memory space; however, they provide very fast access to the data in return. They are also used in NDB Cluster to handle uniqueness constraints.

Currently, the only partitioning algorithm is hashing and ordered indexes are local to each node. Thus, ordered indexes cannot be used to handle uniqueness constraints in the general case.

An important point for both `IndexMemory` and `DataMemory` is that the total database size is the sum of all data memory and all index memory for each node group. Each node group is used to store replicated information, so if there are four nodes with two replicas, there will be two node groups. Thus, the total data memory available is 2 × `DataMemory` for each data node.

It is highly recommended that `DataMemory` and `IndexMemory` be set to the same values for all nodes. Data distribution is even over all nodes in the cluster, so the maximum amount of space available for any node can be no greater than that of the smallest node in the cluster.

`DataMemory` can be changed, but decreasing it can be risky; doing so can easily lead to a node or even an entire NDB Cluster that is unable to restart due to there being insufficient memory space. Increasing these values should be acceptable, but it is recommended that such upgrades are performed in the same manner as a software upgrade, beginning with an update of the configuration file, and then restarting the management server followed by restarting each data node in turn.

MinFreePct. A proportion (5% by default) of data node resources including `DataMemory` is kept in reserve to insure that the data node does not exhaust its memory when performing a restart. This can be adjusted using the `MinFreePct` data node configuration parameter (default 5).

Table 22.37 This table provides type and value information for the MinFreePct data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13

Property	Value
Type or units	unsigned
Default	5
Range	0 - 100
Restart Type	N

Updates do not increase the amount of index memory used. Inserts take effect immediately; however, rows are not actually deleted until the transaction is committed.

Transaction parameters. The next few `[ndbd]` parameters that we discuss are important because they affect the number of parallel transactions and the sizes of transactions that can be handled by the system. `MaxNoOfConcurrentTransactions` sets the number of parallel transactions possible in a node. `MaxNoOfConcurrentOperations` sets the number of records that can be in update phase or locked simultaneously.

Both of these parameters (especially `MaxNoOfConcurrentOperations`) are likely targets for users setting specific values and not using the default value. The default value is set for systems using small transactions, to ensure that these do not use excessive memory.

`MaxDMLOperationsPerTransaction` sets the maximum number of DML operations that can be performed in a given transaction.

- `MaxNoOfConcurrentTransactions`

Table 22.38 This table provides type and value information for the MaxNoOfConcurrentTransactions data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	4096
Range	32 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Each cluster data node requires a transaction record for each active transaction in the cluster. The task of coordinating transactions is distributed among all of the data nodes. The total number of transaction records in the cluster is the number of transactions in any given node times the number of nodes in the cluster.

Transaction records are allocated to individual MySQL servers. Each connection to a MySQL server requires at least one transaction record, plus an additional transaction object per table accessed by that connection. This means that a reasonable minimum for the total number of transactions in the cluster can be expressed as

```
TotalNoOfConcurrentTransactions =
    (maximum number of tables accessed in any single transaction + 1)
    * number of SQL nodes
```

Suppose that there are 10 SQL nodes using the cluster. A single join involving 10 tables requires 11 transaction records; if there are 10 such joins in a transaction, then 10 * 11 = 110 transaction records are required for this transaction, per MySQL server, or 110 * 10 = 1100 transaction records total. Each data node can be expected to handle TotalNoOfConcurrentTransactions / number of data nodes. For

an NDB Cluster having 4 data nodes, this would mean setting `MaxNoOfConcurrentTransactions` on each data node to 1100 / 4 = 275. In addition, you should provide for failure recovery by ensuring that a single node group can accommodate all concurrent transactions; in other words, that each data node's MaxNoOfConcurrentTransactions is sufficient to cover a number of transactions equal to TotalNoOfConcurrentTransactions / number of node groups. If this cluster has a single node group, then `MaxNoOfConcurrentTransactions` should be set to 1100 (the same as the total number of concurrent transactions for the entire cluster).

In addition, each transaction involves at least one operation; for this reason, the value set for `MaxNoOfConcurrentTransactions` should always be no more than the value of `MaxNoOfConcurrentOperations`.

This parameter must be set to the same value for all cluster data nodes. This is due to the fact that, when a data node fails, the oldest surviving node re-creates the transaction state of all transactions that were ongoing in the failed node.

It is possible to change this value using a rolling restart, but the amount of traffic on the cluster must be such that no more transactions occur than the lower of the old and new levels while this is taking place.

The default value is 4096.

- `MaxNoOfConcurrentOperations`

Table 22.39 This table provides type and value information for the MaxNoOfConcurrentOperations data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	32K
Range	32 - 4294967039 (0xFFFFFEFF)
Restart Type	N

It is a good idea to adjust the value of this parameter according to the size and number of transactions. When performing transactions which involve only a few operations and records, the default value for this parameter is usually sufficient. Performing large transactions involving many records usually requires that you increase its value.

Records are kept for each transaction updating cluster data, both in the transaction coordinator and in the nodes where the actual updates are performed. These records contain state information needed to find UNDO records for rollback, lock queues, and other purposes.

This parameter should be set at a minimum to the number of records to be updated simultaneously in transactions, divided by the number of cluster data nodes. For example, in a cluster which has four data nodes and which is expected to handle one million concurrent updates using transactions, you should set this value to 1000000 / 4 = 250000. To help provide resiliency against failures, it is suggested that you set this parameter to a value that is high enough to permit an individual data node to handle the load for its node group. In other words, you should set the value equal to `total number of concurrent operations / number of node groups`. (In the case where there is a single node group, this is the same as the total number of concurrent operations for the entire cluster.)

Because each transaction always involves at least one operation, the value of `MaxNoOfConcurrentOperations` should always be greater than or equal to the value of `MaxNoOfConcurrentTransactions`.

Read queries which set locks also cause operation records to be created. Some extra space is allocated within individual nodes to accommodate cases where the distribution is not perfect over the nodes.

When queries make use of the unique hash index, there are actually two operation records used per record in the transaction. The first record represents the read in the index table and the second handles the operation on the base table.

The default value is 32768.

This parameter actually handles two values that can be configured separately. The first of these specifies how many operation records are to be placed with the transaction coordinator. The second part specifies how many operation records are to be local to the database.

A very large transaction performed on an eight-node cluster requires as many operation records in the transaction coordinator as there are reads, updates, and deletes involved in the transaction. However, the operation records of the are spread over all eight nodes. Thus, if it is necessary to configure the system for one very large transaction, it is a good idea to configure the two parts separately. `MaxNoOfConcurrentOperations` will always be used to calculate the number of operation records in the transaction coordinator portion of the node.

It is also important to have an idea of the memory requirements for operation records. These consume about 1KB per record.

- `MaxNoOfLocalOperations`

Table 22.40 This table provides type and value information for the MaxNoOfLocalOperations data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	UNDEFINED
Range	32 - 4294967039 (0xFFFFFEFF)
Restart Type	N

By default, this parameter is calculated as 1.1 × `MaxNoOfConcurrentOperations`. This fits systems with many simultaneous transactions, none of them being very large. If there is a need to handle one very large transaction at a time and there are many nodes, it is a good idea to override the default value by explicitly specifying this parameter.

- `MaxDMLOperationsPerTransaction`

Table 22.41 This table provides type and value information for the MaxDMLOperationsPerTransaction data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	operations (DML)
Default	4294967295
Range	32 - 4294967295
Restart Type	N

This parameter limits the size of a transaction. The transaction is aborted if it requires more than this many DML operations. The minimum number of operations per transaction is 32; however, you can set `MaxDMLOperationsPerTransaction` to 0 to disable any limitation on the number of DML operations per transaction. The maximum (and default) is 4294967295.

The value of this parameter cannot exceed that set for `MaxNoOfConcurrentOperations`.

Transaction temporary storage. The next set of `[ndbd]` parameters is used to determine temporary storage when executing a statement that is part of a Cluster transaction. All records are released when the statement is completed and the cluster is waiting for the commit or rollback.

The default values for these parameters are adequate for most situations. However, users with a need to support transactions involving large numbers of rows or operations may need to increase these values to enable better parallelism in the system, whereas users whose applications require relatively small transactions can decrease the values to save memory.

- `MaxNoOfConcurrentIndexOperations`

  Table 22.42 This table provides type and value information for the MaxNoOfConcurrentIndexOperations data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	8K
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

  For queries using a unique hash index, another temporary set of operation records is used during a query's execution phase. This parameter sets the size of that pool of records. Thus, this record is allocated only while executing a part of a query. As soon as this part has been executed, the record is released. The state needed to handle aborts and commits is handled by the normal operation records, where the pool size is set by the parameter `MaxNoOfConcurrentOperations`.

  The default value of this parameter is 8192. Only in rare cases of extremely high parallelism using unique hash indexes should it be necessary to increase this value. Using a smaller value is possible and can save memory if the DBA is certain that a high degree of parallelism is not required for the cluster.

- `MaxNoOfFiredTriggers`

  Table 22.43 This table provides type and value information for the MaxNoOfFiredTriggers data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	4000
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

  The default value of `MaxNoOfFiredTriggers` is 4000, which is sufficient for most situations. In some cases it can even be decreased if the DBA feels certain the need for parallelism in the cluster is not high.

A record is created when an operation is performed that affects a unique hash index. Inserting or deleting a record in a table with unique hash indexes or updating a column that is part of a unique hash index fires an insert or a delete in the index table. The resulting record is used to represent this index table operation while waiting for the original operation that fired it to complete. This operation is short-lived but can still require a large number of records in its pool for situations with many parallel write operations on a base table containing a set of unique hash indexes.

- `TransactionBufferMemory`

Table 22.44 This table provides type and value information for the TransactionBufferMemory data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	1M
Range	1K - 4294967039 (0xFFFFFEFF)
Restart Type	N

The memory affected by this parameter is used for tracking operations fired when updating index tables and reading unique indexes. This memory is used to store the key and column information for these operations. It is only very rarely that the value for this parameter needs to be altered from the default.

The default value for `TransactionBufferMemory` is 1MB.

Normal read and write operations use a similar buffer, whose usage is even more short-lived. The compile-time parameter `ZATTRBUF_FILESIZE` (found in `ndb/src/kernel/blocks/Dbtc/Dbtc.hpp`) set to 4000 × 128 bytes (500KB). A similar buffer for key information, `ZDATABUF_FILESIZE` (also in `Dbtc.hpp`) contains 4000 × 16 = 62.5KB of buffer space. `Dbtc` is the module that handles transaction coordination.

Transaction resource allocation parameters. The parameters in the following list are used to allocate transaction resources in the transaction coordinator (`DBTC`; see The DBTC Block). Leaving any one of these set to the default (0) dedicates transaction memory for 25% of estimated total data node usage for the corresponding resource. The actual maximum possible values for these parameters are typically limited by the amount of memory available to the data node; setting them has no impact on the total amount of memory allocated to the data node. In addition, you should keep in mind that they control numbers of reserved internal records for the data node independent of any settings for `MaxDMLOperationsPerTransaction`, `MaxNoOfConcurrentIndexOperations`, `MaxNoOfConcurrentOperations`, `MaxNoOfConcurrentScans`, `MaxNoOfConcurrentTransactions`, `MaxNoOfFiredTriggers`, `MaxNoOfLocalScans`, or `TransactionBufferMemory` (see Transaction parameters and Transaction temporary storage).

- `ReservedConcurrentIndexOperations`

Table 22.45 This table provides type and value information for the ReservedConcurrentIndexOperations data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.16
Type or units	numeric

Property	Value
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Number of simultaneous index operations having dedicated resources on one data node.

- `ReservedConcurrentOperations`

Table 22.46 This table provides type and value information for the ReservedConcurrentOperations data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.16
Type or units	numeric
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Number of simultaneous operations having dedicated resources in transaction coordinators on one data node.

- `ReservedConcurrentScans`

Table 22.47 This table provides type and value information for the ReservedConcurrentScans data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.16
Type or units	numeric
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Number of simultaneous scans having dedicated resources on one data node.

- `ReservedConcurrentTransactions`

Table 22.48 This table provides type and value information for the ReservedConcurrentTransactions data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.16
Type or units	numeric
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Number of simultaneous transactions having dedicated resources on one data node.

- `ReservedFiredTriggers`

Table 22.49 This table provides type and value information for the ReservedFiredTriggers data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.16
Type or units	numeric
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Number of triggers that have dedicated resources on one ndbd(DB) node.

- `ReservedLocalScans`

Table 22.50 This table provides type and value information for the ReservedLocalScans data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.16
Type or units	numeric
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Number of simultaneous fragment scans having dedicated resources on one data node.

- `ReservedTransactionBufferMemory`

Table 22.51 This table provides type and value information for the ReservedTransactionBufferMemory data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.16
Type or units	numeric
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Dynamic buffer space (in bytes) for key and attribute data allocated to each data node.

Scans and buffering. There are additional `[ndbd]` parameters in the `Dblqh` module (in `ndb/src/kernel/blocks/Dblqh/Dblqh.hpp`) that affect reads and updates. These include `ZATTRINBUF_FILESIZE`, set by default to 10000 × 128 bytes (1250KB) and `ZDATABUF_FILE_SIZE`, set by default to 10000*16 bytes (roughly 156KB) of buffer space. To date, there have been neither any reports from users nor any results from our own extensive tests suggesting that either of these compile-time limits should be increased.

- `BatchSizePerLocalScan`

Table 22.52 This table provides type and value information for the BatchSizePerLocalScan data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	256
Range	1 - 992
Restart Type	N

This parameter is used to calculate the number of lock records used to handle concurrent scan operations.

`BatchSizePerLocalScan` has a strong connection to the `BatchSize` defined in the SQL nodes.

- `LongMessageBuffer`

Table 22.53 This table provides type and value information for the LongMessageBuffer data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	64M
Range	512K - 4294967039 (0xFFFFFEFF)
Restart Type	N

This is an internal buffer used for passing messages within individual nodes and between nodes. The default is 64MB.

This parameter seldom needs to be changed from the default.

- `MaxFKBuildBatchSize`

Table 22.54 This table provides type and value information for the MaxFKBuildBatchSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	64
Range	16 - 512
Restart Type	S

Maximum scan batch size used for building foreign keys. Increasing the value set for this parameter may speed up building of foreign key builds at the expense of greater impact to ongoing traffic.

- `MaxNoOfConcurrentScans`

Table 22.55 This table provides type and value information for the MaxNoOfConcurrentScans data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	256
Range	2 - 500
Restart Type	N

This parameter is used to control the number of parallel scans that can be performed in the cluster. Each transaction coordinator can handle the number of parallel scans defined for this parameter. Each scan query is performed by scanning all partitions in parallel. Each partition scan uses a scan record in the node where the partition is located, the number of records being the value of this parameter times the number of nodes. The cluster should be able to sustain `MaxNoOfConcurrentScans` scans concurrently from all nodes in the cluster.

Scans are actually performed in two cases. The first of these cases occurs when no hash or ordered indexes exists to handle the query, in which case the query is executed by performing a full table scan. The second case is encountered when there is no hash index to support the query but there is an ordered index. Using the ordered index means executing a parallel range scan. The order is kept on the local partitions only, so it is necessary to perform the index scan on all partitions.

The default value of `MaxNoOfConcurrentScans` is 256. The maximum value is 500.

- `MaxNoOfLocalScans`

Table 22.56 This table provides type and value information for the MaxNoOfLocalScans data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	[see text]
Range	32 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Specifies the number of local scan records if many scans are not fully parallelized. When the number of local scan records is not provided, it is calculated as shown here:

```
4 * MaxNoOfConcurrentScans * [# data nodes] + 2
```

The minimum value is 32.

- `MaxParallelCopyInstances`

Table 22.57 This table provides type and value information for the MaxParallelCopyInstances data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 64
Restart Type	S

This parameter sets the parallelization used in the copy phase of a node restart or system restart, when a node that is currently just starting is synchronised with a node that already has current data by copying over any changed records from the node that is up to date. Because full parallelism in such cases can lead to overload situations, `MaxParallelCopyInstances` provides a means to decrease it. This parameter's default value 0. This value means that the effective parallelism is equal to the number of LDM instances in the node just starting as well as the node updating it.

- `MaxParallelScansPerFragment`

Table 22.58 This table provides type and value information for the MaxParallelScansPerFragment data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	256
Range	1 - 4294967039 (0xFFFFFEFF)
Restart Type	N

It is possible to configure the maximum number of parallel scans (`TUP` scans and `TUX` scans) allowed before they begin queuing for serial handling. You can increase this to take advantage of any unused CPU when performing large number of scans in parallel and improve their performance.

The default value for this parameter is 256.

- `MaxReorgBuildBatchSize`

Table 22.59 This table provides type and value information for the MaxReorgBuildBatchSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	64
Range	16 - 512
Restart Type	S

Maximum scan batch size used for reorganization of table partitions. Increasing the value set for this parameter may speed up reorganization at the expense of greater impact to ongoing traffic.

- `MaxUIBuildBatchSize`

Table 22.60 This table provides type and value information for the MaxUIBuildBatchSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	64
Range	16 - 512
Restart Type	S

Maximum scan batch size used for building unique keys. Increasing the value set for this parameter may speed up such builds at the expense of greater impact to ongoing traffic.

Memory Allocation

`MaxAllocate`

Table 22.61 This table provides type and value information for the MaxAllocate data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	32M
Range	1M - 1G
Restart Type	N

This is the maximum size of the memory unit to use when allocating memory for tables. In cases where `NDB` gives `Out of memory` errors, but it is evident by examining the cluster logs or the output of `DUMP 1000` that all available memory has not yet been used, you can increase the value of this parameter (or `MaxNoOfTables`, or both) to cause `NDB` to make sufficient memory available.

Hash Map Size

`DefaultHashMapSize`

Table 22.62 This table provides type and value information for the DefaultHashMapSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	LDM threads
Default	3840
Range	0 - 3840
Restart Type	N

The size of the table hash maps used by `NDB` is configurable using this parameter. `DefaultHashMapSize` can take any of three possible values (0, 240, 3840). These values and their effects are described in the following table:

Table 22.63 DefaultHashMapSize parameters

Value	Description / Effect
0	Use the lowest value set, if any, for this parameter among all data nodes and API nodes in the cluster; if it is not set on any data or API node, use the default value.
240	Original hash map size (used in older NDB Cluster releases)
3840	Larger hash map size (used by default in NDB 8.0)

The original intended use for this parameter was to facilitate upgrades and especially downgrades to and from very old releases with differing default hash map sizes. This is not an issue when upgrading from NDB Cluster 7.6 to NDB Cluster 8.0.

Logging and checkpointing. The following [ndbd] parameters control log and checkpoint behavior.

- FragmentLogFileSize

  Table 22.64 This table provides type and value information for the FragmentLogFileSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	16M
Range	4M - 1G
Restart Type	IN

  Setting this parameter enables you to control directly the size of redo log files. This can be useful in situations when NDB Cluster is operating under a high load and it is unable to close fragment log files quickly enough before attempting to open new ones (only 2 fragment log files can be open at one time); increasing the size of the fragment log files gives the cluster more time before having to open each new fragment log file. The default value for this parameter is 16M.

  For more information about fragment log files, see the description for NoOfFragmentLogFiles.

- InitialNoOfOpenFiles

  Table 22.65 This table provides type and value information for the InitialNoOfOpenFiles data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	files
Default	27
Range	20 - 4294967039 (0xFFFFFEFF)
Restart Type	N

  This parameter sets the initial number of internal threads to allocate for open files.

  The default value is 27.

- InitFragmentLogFiles

Table 22.66 This table provides type and value information for the InitFragmentLogFiles data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	[see values]
Default	SPARSE
Range	SPARSE, FULL
Restart Type	IN

By default, fragment log files are created sparsely when performing an initial start of a data node—that is, depending on the operating system and file system in use, not all bytes are necessarily written to disk. However, it is possible to override this behavior and force all bytes to be written, regardless of the platform and file system type being used, by means of this parameter. `InitFragmentLogFiles` takes either of two values:

- `SPARSE`. Fragment log files are created sparsely. This is the default value.
- `FULL`. Force all bytes of the fragment log file to be written to disk.

Depending on your operating system and file system, setting `InitFragmentLogFiles=FULL` may help eliminate I/O errors on writes to the REDO log.

- `EnablePartialLcp`

Table 22.67 This table provides type and value information for the EnablePartialLcp data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	true
Range	...
Restart Type	N

When `true`, enable partial local checkpoints: This means that each LCP records only part of the full database, plus any records containing rows changed since the last LCP; if no rows have changed, the LCP updates only the LCP control file and does not update any data files.

If `EnablePartialLcp` is disabled (`false`), each LCP uses only a single file and writes a full checkpoint; this requires the least amount of disk space for LCPs, but increases the write load for each LCP. The default value is enabled (`true`). The proportion of space used by partiaL LCPS can be modified by the setting for the `RecoveryWork` configuration parameter.

Setting this parameter to `false` also disables the calculation of disk write speed used by the adaptive LCP control mechanism.

- `LcpScanProgressTimeout`

Table 22.68 This table provides type and value information for the LcpScanProgressTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	second
Default	60
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

A local checkpoint fragment scan watchdog checks periodically for no progress in each fragment scan performed as part of a local checkpoint, and shuts down the node if there is no progress after a given amount of time has elapsed. This interval can be set using the `LcpScanProgressTimeout` data node configuration parameter, which sets the maximum time for which the local checkpoint can be stalled before the LCP fragment scan watchdog shuts down the node.

The default value is 60 seconds (providing compatibility with previous releases). Setting this parameter to 0 disables the LCP fragment scan watchdog altogether.

- `MaxNoOfOpenFiles`

Table 22.69 This table provides type and value information for the MaxNoOfOpenFiles data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0
Range	20 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets a ceiling on how many internal threads to allocate for open files. *Any situation requiring a change in this parameter should be reported as a bug*.

The default value is 0. However, the minimum value to which this parameter can be set is 20.

- `MaxNoOfSavedMessages`

Table 22.70 This table provides type and value information for the MaxNoOfSavedMessages data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	25
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets the maximum number of errors written in the error log as well as the maximum number of trace files that are kept before overwriting the existing ones. Trace files are generated when, for whatever reason, the node crashes.

The default is 25, which sets these maximums to 25 error messages and 25 trace files.

- `MaxLCPStartDelay`

Table 22.71 This table provides type and value information for the MaxLCPStartDelay data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	seconds
Default	0
Range	0 - 600
Restart Type	N

In parallel data node recovery, only table data is actually copied and synchronized in parallel; synchronization of metadata such as dictionary and checkpoint information is done in a serial fashion. In addition, recovery of dictionary and checkpoint information cannot be executed in parallel with performing of local checkpoints. This means that, when starting or restarting many data nodes concurrently, data nodes may be forced to wait while a local checkpoint is performed, which can result in longer node recovery times.

It is possible to force a delay in the local checkpoint to permit more (and possibly all) data nodes to complete metadata synchronization; once each data node's metadata synchronization is complete, all of the data nodes can recover table data in parallel, even while the local checkpoint is being executed. To force such a delay, set `MaxLCPStartDelay`, which determines the number of seconds the cluster can wait to begin a local checkpoint while data nodes continue to synchronize metadata. This parameter should be set in the `[ndbd default]` section of the `config.ini` file, so that it is the same for all data nodes. The maximum value is 600; the default is 0.

- `NoOfFragmentLogFiles`

Table 22.72 This table provides type and value information for the NoOfFragmentLogFiles data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	16
Range	3 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

This parameter sets the number of REDO log files for the node, and thus the amount of space allocated to REDO logging. Because the REDO log files are organized in a ring, it is extremely important that the first and last log files in the set (sometimes referred to as the “head” and “tail” log files, respectively) do not meet. When these approach one another too closely, the node begins aborting all transactions encompassing updates due to a lack of room for new log records.

A REDO log record is not removed until both required local checkpoints have been completed since that log record was inserted. Checkpointing frequency is determined by its own set of configuration parameters discussed elsewhere in this chapter.

The default parameter value is 16, which by default means 16 sets of 4 16MB files for a total of 1024MB. The size of the individual log files is configurable using the FragmentLogFileSize parameter. In scenarios requiring a great many updates, the value for NoOfFragmentLogFiles may need to be set as high as 300 or even higher to provide sufficient space for REDO logs.

If the checkpointing is slow and there are so many writes to the database that the log files are full and the log tail cannot be cut without jeopardizing recovery, all updating transactions are aborted with internal error code 410 (Out of log file space temporarily). This condition prevails until a checkpoint has completed and the log tail can be moved forward.

Important

This parameter cannot be changed “on the fly”; you must restart the node using --initial. If you wish to change this value for all data nodes in a running cluster, you can do so using a rolling node restart (using --initial when starting each data node).

- RecoveryWork

Table 22.73 This table provides type and value information for the RecoveryWork data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	60
Range	25 - 100
Restart Type	N

Percentage of storage overhead for LCP files. This parameter has an effect only when EnablePartialLcp is true, that is, only when partial local checkpoints are enabled. A higher value means:

- Fewer records are written for each LCP, LCPs use more space
- More work is needed during restarts

A lower value for RecoveryWork means:

- More records are written during each LCP, but LCPs require less space on disk.
- Less work during restart and thus faster restarts, at the expense of more work during normal operations

For example, setting RecoveryWork to 60 means that the total size of an LCP is roughly 1 + 0.6 = 1.6 times the size of the data to be checkpointed. This means that 60% more work is required during the restore phase of a restart compared to the work done during a restart that uses full checkpoints. (This is more than compensated for during other phases of the restart such that the restart as a whole is still faster when using partial LCPs than when using full LCPs.) In order not to fill up the redo log, it

is necessary to write at 1 + (1 / RecoveryWork) times the rate of data changes during checkpoints—thus, when RecoveryWork = 60, it is necessary to write at approximately 1 + (1 / 0.6) = 2.67 times the change rate. In other words, if changes are being written at 10 MByte per second, the checkpoint needs to be written at roughly 26.7 MByte per second.

Setting RecoveryWork = 40 means that only 1.4 times the total LCP size is needed (and thus the restore phase takes 10 to 15 percent less time. In this case, the checkpoint write rate is 3.5 times the rate of change.

The NDB source distribution includes a test program for simulating LCPs. lcp_simulator.cc can be found in storage/ndb/src/kernel/blocks/backup/. To compile and run it on Unix platforms, execute the commands shown here:

```
shell> gcc lcp_simulator.cc
shell> ./a.out
```

This program has no dependencies other than stdio.h, and does not require a connection to an NDB cluster or a MySQL server. By default, it simulates 300 LCPs (three sets of 100 LCPs, each consisting of inserts, updates, and deletes, in turn), reporting the size of the LCP after each one. You can alter the simulation by changing the values of recovery_work, insert_work, and delete_work in the source and recompiling. For more information, see the source of the program.

- InsertRecoveryWork

Table 22.74 This table provides type and value information for the InsertRecoveryWork data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	40
Range	0 - 70
Restart Type	N

Percentage of RecoveryWork used for inserted rows. A higher value increases the number of writes during a local checkpoint, and decreases the total size of the LCP. A lower value decreases the number of writes during an LCP, but results in more space being used for the LCP, which means that recovery takes longer. This parameter has an effect only when EnablePartialLcp is true, that is, only when partial local checkpoints are enabled.

- EnableRedoControl

Table 22.75 This table provides type and value information for the EnableRedoControl data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	...
Restart Type	N

Enable adaptive checkpointing speed for controlling redo log usage. Set to `false` to disable (the default). Setting `EnablePartialLcp` to `false` also disables the adaptive calculation.

When enabled, `EnableRedoControl` allows the data nodes greater flexibility with regard to the rate at which they write LCPs to disk. More specifically, enabling this parameter means that higher write rates can be employed, so that LCPs can complete and Redo logs be trimmed more quickly, thereby reducing recovery time and disk space requirements. This functionality allows data nodes to make better use of the higher rate of I/O and greater bandwidth available from modern solid-state storage devices and protocols, such as solid-state drives (SSDs) using Non-Volatile Memory Express (NVMe).

The parameter currently defaults to `false` (disabled) due to the fact that `NDB` is still deployed widely on systems whose I/O or bandwidth is constrained relative to those employing solid-state technology, such as those using conventional hard disks (HDDs). In settings such as these, the `EnableRedoControl` mechanism can easily cause the I/O subsystem to become saturated, increasing wait times for data node input and output. In particular, this can cause issues with NDB Disk Data tables which have tablespaces or log file groups sharing a constrained IO subsystem with data node LCP and redo log files; such problems potentially include node or cluster failure due to GCP stop errors.

Metadata objects. The next set of `[ndbd]` parameters defines pool sizes for metadata objects, used to define the maximum number of attributes, tables, indexes, and trigger objects used by indexes, events, and replication between clusters.

Note

These act merely as “suggestions” to the cluster, and any that are not specified revert to the default values shown.

- `MaxNoOfAttributes`

Table 22.76 This table provides type and value information for the MaxNoOfAttributes data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	1000
Range	32 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets a suggested maximum number of attributes that can be defined in the cluster; like `MaxNoOfTables`, it is not intended to function as a hard upper limit.

(In older NDB Cluster releases, this parameter was sometimes treated as a hard limit for certain operations. This caused problems with NDB Cluster Replication, when it was possible to create more tables than could be replicated, and sometimes led to confusion when it was possible [or not possible, depending on the circumstances] to create more than `MaxNoOfAttributes` attributes.)

The default value is 1000, with the minimum possible value being 32. The maximum is 4294967039. Each attribute consumes around 200 bytes of storage per node due to the fact that all metadata is fully replicated on the servers.

When setting `MaxNoOfAttributes`, it is important to prepare in advance for any `ALTER TABLE` statements that you might want to perform in the future. This is due to the fact, during the

execution of `ALTER TABLE` on a Cluster table, 3 times the number of attributes as in the original table are used, and a good practice is to permit double this amount. For example, if the NDB Cluster table having the greatest number of attributes (*`greatest_number_of_attributes`*) has 100 attributes, a good starting point for the value of `MaxNoOfAttributes` would be `6 * greatest_number_of_attributes = 600`.

You should also estimate the average number of attributes per table and multiply this by `MaxNoOfTables`. If this value is larger than the value obtained in the previous paragraph, you should use the larger value instead.

Assuming that you can create all desired tables without any problems, you should also verify that this number is sufficient by trying an actual `ALTER TABLE` after configuring the parameter. If this is not successful, increase `MaxNoOfAttributes` by another multiple of `MaxNoOfTables` and test it again.

- `MaxNoOfTables`

Table 22.77 This table provides type and value information for the MaxNoOfTables data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	128
Range	8 - 20320
Restart Type	N

A table object is allocated for each table and for each unique hash index in the cluster. This parameter sets a suggested maximum number of table objects for the cluster as a whole; like `MaxNoOfAttributes`, it is not intended to function as a hard upper limit.

(In older NDB Cluster releases, this parameter was sometimes treated as a hard limit for certain operations. This caused problems with NDB Cluster Replication, when it was possible to create more tables than could be replicated, and sometimes led to confusion when it was possible [or not possible, depending on the circumstances] to create more than `MaxNoOfTables` tables.)

For each attribute that has a `BLOB` data type an extra table is used to store most of the `BLOB` data. These tables also must be taken into account when defining the total number of tables.

The default value of this parameter is 128. The minimum is 8 and the maximum is 20320. Each table object consumes approximately 20KB per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfOrderedIndexes`

Table 22.78 This table provides type and value information for the MaxNoOfOrderedIndexes data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13

Property	Value
Type or units	integer
Default	128
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

For each ordered index in the cluster, an object is allocated describing what is being indexed and its storage segments. By default, each index so defined also defines an ordered index. Each unique index and primary key has both an ordered index and a hash index. `MaxNoOfOrderedIndexes` sets the total number of ordered indexes that can be in use in the system at any one time.

The default value of this parameter is 128. Each index object consumes approximately 10KB of data per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfUniqueHashIndexes`

Table 22.79 This table provides type and value information for the MaxNoOfUniqueHashIndexes data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	64
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

For each unique index that is not a primary key, a special table is allocated that maps the unique key to the primary key of the indexed table. By default, an ordered index is also defined for each unique index. To prevent this, you must specify the `USING HASH` option when defining the unique index.

The default value is 64. Each index consumes approximately 15KB per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfTriggers`

Table 22.80 This table provides type and value information for the MaxNoOfTriggers data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	768

Property	Value
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Internal update, insert, and delete triggers are allocated for each unique hash index. (This means that three triggers are created for each unique hash index.) However, an *ordered* index requires only a single trigger object. Backups also use three trigger objects for each normal table in the cluster.

Replication between clusters also makes use of internal triggers.

This parameter sets the maximum number of trigger objects in the cluster.

The default value is 768.

- `MaxNoOfSubscriptions`

Table 22.81 This table provides type and value information for the MaxNoOfSubscriptions data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Each `NDB` table in an NDB Cluster requires a subscription in the NDB kernel. For some NDB API applications, it may be necessary or desirable to change this parameter. However, for normal usage with MySQL servers acting as SQL nodes, there is not any need to do so.

The default value for `MaxNoOfSubscriptions` is 0, which is treated as equal to `MaxNoOfTables`. Each subscription consumes 108 bytes.

- `MaxNoOfSubscribers`

Table 22.82 This table provides type and value information for the MaxNoOfSubscribers data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter is of interest only when using NDB Cluster Replication. The default value is 0, which is treated as `2 * MaxNoOfTables`; that is, there is one subscription per `NDB` table for each of two MySQL servers (one acting as the replication master and the other as the slave). Each subscriber uses 16 bytes of memory.

When using circular replication, multi-master replication, and other replication setups involving more than 2 MySQL servers, you should increase this parameter to the number of `mysqld` processes included in replication (this is often, but not always, the same as the number of clusters). For example, if you have a circular replication setup using three NDB Cluster s, with one `mysqld` attached to each cluster, and each of these `mysqld` processes acts as a master and as a slave, you should set `MaxNoOfSubscribers` equal to `3 * MaxNoOfTables`.

For more information, see Section 22.6, “NDB Cluster Replication”.

- `MaxNoOfConcurrentSubOperations`

Table 22.83 This table provides type and value information for the MaxNoOfConcurrentSubOperations data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	256
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets a ceiling on the number of operations that can be performed by all API nodes in the cluster at one time. The default value (256) is sufficient for normal operations, and might need to be adjusted only in scenarios where there are a great many API nodes each performing a high volume of operations concurrently.

Boolean parameters. The behavior of data nodes is also affected by a set of `[ndbd]` parameters taking on boolean values. These parameters can each be specified as `TRUE` by setting them equal to `1` or `Y`, and as `FALSE` by setting them equal to `0` or `N`.

- `CompressedBackup`

Table 22.84 This table provides type and value information for the CompressedBackup data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Enabling this parameter causes backup files to be compressed. The compression used is equivalent to `gzip --fast`, and can save 50% or more of the space required on the data node to store uncompressed backup files. Compressed backups can be enabled for individual data nodes, or for all data nodes (by setting this parameter in the `[ndbd default]` section of the `config.ini` file).

Important

You cannot restore a compressed backup to a cluster running a MySQL version that does not support this feature.

The default value is `0` (disabled).

- `CompressedLCP`

Table 22.85 This table provides type and value information for the CompressedLCP data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Setting this parameter to `1` causes local checkpoint files to be compressed. The compression used is equivalent to `gzip --fast`, and can save 50% or more of the space required on the data node to store uncompressed checkpoint files. Compressed LCPs can be enabled for individual data nodes, or for all data nodes (by setting this parameter in the `[ndbd default]` section of the `config.ini` file).

Important

You cannot restore a compressed local checkpoint to a cluster running a MySQL version that does not support this feature.

The default value is `0` (disabled).

- `CrashOnCorruptedTuple`

Table 22.86 This table provides type and value information for the CrashOnCorruptedTuple data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	true
Range	true, false
Restart Type	S

When this parameter is enabled (the default), it forces a data node to shut down whenever it encounters a corrupted tuple.

- `Diskless`

Table 22.87 This table provides type and value information for the Diskless data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	true|false (1|0)
Default	false

Property	Value
Range	true, false
Restart Type	IS

It is possible to specify NDB Cluster tables as *diskless*, meaning that tables are not checkpointed to disk and that no logging occurs. Such tables exist only in main memory. A consequence of using diskless tables is that neither the tables nor the records in those tables survive a crash. However, when operating in diskless mode, it is possible to run `ndbd` on a diskless computer.

Important

This feature causes the *entire* cluster to operate in diskless mode.

When this feature is enabled, Cluster online backup is disabled. In addition, a partial start of the cluster is not possible.

`Diskless` is disabled by default.

- `LateAlloc`

Table 22.88 This table provides type and value information for the LateAlloc data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	1
Range	0 - 1
Restart Type	N

Allocate memory for this data node after a connection to the management server has been established. Enabled by default.

- `LockPagesInMainMemory`

Table 22.89 This table provides type and value information for the LockPagesInMainMemory data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	0
Range	0 - 2
Restart Type	N

For a number of operating systems, including Solaris and Linux, it is possible to lock a process into memory and so avoid any swapping to disk. This can be used to help guarantee the cluster's real-time characteristics.

This parameter takes one of the integer values `0`, `1`, or `2`, which act as shown in the following list:

- `0`: Disables locking. This is the default value.
- `1`: Performs the lock after allocating memory for the process.
- `2`: Performs the lock before memory for the process is allocated.

If the operating system is not configured to permit unprivileged users to lock pages, then the data node process making use of this parameter may have to be run as system root. (`LockPagesInMainMemory` uses the `mlockall` function. From Linux kernel 2.6.9, unprivileged users can lock memory as limited by `max locked memory`. For more information, see `ulimit -l` and http://linux.die.net/man/2/mlock).

Note

In older NDB Cluster releases, this parameter was a Boolean. `0` or `false` was the default setting, and disabled locking. `1` or `true` enabled locking of the process after its memory was allocated. NDB Cluster 8.0 treats `true` or `false` for the value of this parameter as an error.

Important

Beginning with `glibc` 2.10, `glibc` uses per-thread arenas to reduce lock contention on a shared pool, which consumes real memory. In general, a data node process does not need per-thread arenas, since it does not perform any memory allocation after startup. (This difference in allocators does not appear to affect performance significantly.)

The `glibc` behavior is intended to be configurable via the `MALLOC_ARENA_MAX` environment variable, but a bug in this mechanism prior to `glibc` 2.16 meant that this variable could not be set to less than 8, so that the wasted memory could not be reclaimed. (Bug #15907219; see also http://sourceware.org/bugzilla/show_bug.cgi?id=13137 for more information concerning this issue.)

One possible workaround for this problem is to use the `LD_PRELOAD` environment variable to preload a `jemalloc` memory allocation library to take the place of that supplied with `glibc`.

- `ODirect`

Table 22.90 This table provides type and value information for the ODirect data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Enabling this parameter causes `NDB` to attempt using `O_DIRECT` writes for LCP, backups, and redo logs, often lowering `kswapd` and CPU usage. When using NDB Cluster on Linux, enable `ODirect` if you are using a 2.6 or later kernel.

`ODirect` is disabled by default.

- `ODirectSyncFlag`

Table 22.91 This table provides type and value information for the ODirectSyncFlag data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

When this parameter is enabled, redo log writes are performed such that each completed file system write is handled as a call to `fsync`. The setting for this parameter is ignored if at least one of the following conditions is true:

- `ODirect` is not enabled.
- `InitFragmentLogFiles` is set to `SPARSE`.

Disabled by default.

- `RestartOnErrorInsert`

Table 22.92 This table provides type and value information for the RestartOnErrorInsert data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	error code
Default	2
Range	0 - 4
Restart Type	N

This feature is accessible only when building the debug version where it is possible to insert errors in the execution of individual blocks of code as part of testing.

This feature is disabled by default.

- `StopOnError`

Table 22.93 This table provides type and value information for the StopOnError data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	1
Range	0, 1
Restart Type	N

This parameter specifies whether a data node process should exit or perform an automatic restart when an error condition is encountered.

This parameter's default value is 1; this means that, by default, an error causes the data node process to halt.

When an error is encountered and `StopOnError` is 0, the data node process is restarted.

Users of MySQL Cluster Manager should note that, when `StopOnError` equals 1, this prevents the MySQL Cluster Manager agent from restarting any data nodes after it has performed its own restart and recovery. See Starting and Stopping the Agent on Linux, for more information.

- `UseShm`

Table 22.94 This table provides type and value information for the UseShm shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	S

Enable a shared memory connection between this data node and the API node also running on this host. Set to 1 to enable.

Controlling Timeouts, Intervals, and Disk Paging

There are a number of `[ndbd]` parameters specifying timeouts and intervals between various actions in Cluster data nodes. Most of the timeout values are specified in milliseconds. Any exceptions to this are mentioned where applicable.

- `TimeBetweenWatchDogCheck`

Table 22.95 This table provides type and value information for the TimeBetweenWatchDogCheck data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	6000
Range	70 - 4294967039 (0xFFFFFEFF)
Restart Type	N

To prevent the main thread from getting stuck in an endless loop at some point, a “watchdog” thread checks the main thread. This parameter specifies the number of milliseconds between checks. If the process remains in the same state after three checks, the watchdog thread terminates it.

This parameter can easily be changed for purposes of experimentation or to adapt to local conditions. It can be specified on a per-node basis although there seems to be little reason for doing so.

The default timeout is 6000 milliseconds (6 seconds).

- `TimeBetweenWatchDogCheckInitial`

Table 22.96 This table provides type and value information for the TimeBetweenWatchDogCheckInitial data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	6000
Range	70 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This is similar to the `TimeBetweenWatchDogCheck` parameter, except that `TimeBetweenWatchDogCheckInitial` controls the amount of time that passes between execution checks inside a storage node in the early start phases during which memory is allocated.

The default timeout is 6000 milliseconds (6 seconds).

- `StartPartialTimeout`

Table 22.97 This table provides type and value information for the StartPartialTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	30000
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter specifies how long the Cluster waits for all data nodes to come up before the cluster initialization routine is invoked. This timeout is used to avoid a partial Cluster startup whenever possible.

This parameter is overridden when performing an initial start or initial restart of the cluster.

The default value is 30000 milliseconds (30 seconds). 0 disables the timeout, in which case the cluster may start only if all nodes are available.

- `StartPartitionedTimeout`

Table 22.98 This table provides type and value information for the StartPartitionedTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	60000
Range	0 - 4294967039 (0xFFFFFEFF)

Property	Value
Restart Type	N

If the cluster is ready to start after waiting for `StartPartialTimeout` milliseconds but is still possibly in a partitioned state, the cluster waits until this timeout has also passed. If `StartPartitionedTimeout` is set to 0, the cluster waits indefinitely.

This parameter is overridden when performing an initial start or initial restart of the cluster.

The default timeout is 60000 milliseconds (60 seconds).

- `StartFailureTimeout`

Table 22.99 This table provides type and value information for the StartFailureTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

If a data node has not completed its startup sequence within the time specified by this parameter, the node startup fails. Setting this parameter to 0 (the default value) means that no data node timeout is applied.

For nonzero values, this parameter is measured in milliseconds. For data nodes containing extremely large amounts of data, this parameter should be increased. For example, in the case of a data node containing several gigabytes of data, a period as long as 10−15 minutes (that is, 600000 to 1000000 milliseconds) might be required to perform a node restart.

- `StartNoNodeGroupTimeout`

Table 22.100 This table provides type and value information for the StartNoNodeGroupTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	15000
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When a data node is configured with `Nodegroup = 65536`, is regarded as not being assigned to any node group. When that is done, the cluster waits `StartNoNodegroupTimeout` milliseconds, then treats such nodes as though they had been added to the list passed to the `--nowait-nodes` option, and starts. The default value is `15000` (that is, the management server waits 15 seconds). Setting this parameter equal to `0` means that the cluster waits indefinitely.

`StartNoNodegroupTimeout` must be the same for all data nodes in the cluster; for this reason, you should always set it in the `[ndbd default]` section of the `config.ini` file, rather than for individual data nodes.

See Section 22.5.15, “Adding NDB Cluster Data Nodes Online”, for more information.

- `HeartbeatIntervalDbDb`

Table 22.101 This table provides type and value information for the HeartbeatIntervalDbDb data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	5000
Range	10 - 4294967039 (0xFFFFFEFF)
Restart Type	N

One of the primary methods of discovering failed nodes is by the use of heartbeats. This parameter states how often heartbeat signals are sent and how often to expect to receive them. Heartbeats cannot be disabled.

After missing four heartbeat intervals in a row, the node is declared dead. Thus, the maximum time for discovering a failure through the heartbeat mechanism is five times the heartbeat interval.

The default heartbeat interval is 5000 milliseconds (5 seconds). This parameter must not be changed drastically and should not vary widely between nodes. If one node uses 5000 milliseconds and the node watching it uses 1000 milliseconds, obviously the node will be declared dead very quickly. This parameter can be changed during an online software upgrade, but only in small increments.

See also Network communication and latency, as well as the description of the `ConnectCheckIntervalDelay` configuration parameter.

- `HeartbeatIntervalDbApi`

Table 22.102 This table provides type and value information for the HeartbeatIntervalDbApi data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	1500
Range	100 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Each data node sends heartbeat signals to each MySQL server (SQL node) to ensure that it remains in contact. If a MySQL server fails to send a heartbeat in time it is declared “dead,” in which case all ongoing transactions are completed and all resources released. The SQL node cannot reconnect until all activities initiated by the previous MySQL instance have been completed. The three-heartbeat criteria for this determination are the same as described for `HeartbeatIntervalDbDb`.

The default interval is 1500 milliseconds (1.5 seconds). This interval can vary between individual data nodes because each data node watches the MySQL servers connected to it, independently of all other data nodes.

For more information, see Network communication and latency.

- `HeartbeatOrder`

Table 22.103 This table provides type and value information for the HeartbeatOrder data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	0
Range	0 - 65535
Restart Type	S

Data nodes send heartbeats to one another in a circular fashion whereby each data node monitors the previous one. If a heartbeat is not detected by a given data node, this node declares the previous data node in the circle “dead” (that is, no longer accessible by the cluster). The determination that a data node is dead is done globally; in other words; once a data node is declared dead, it is regarded as such by all nodes in the cluster.

It is possible for heartbeats between data nodes residing on different hosts to be too slow compared to heartbeats between other pairs of nodes (for example, due to a very low heartbeat interval or temporary connection problem), such that a data node is declared dead, even though the node can still function as part of the cluster. .

In this type of situation, it may be that the order in which heartbeats are transmitted between data nodes makes a difference as to whether or not a particular data node is declared dead. If this declaration occurs unnecessarily, this can in turn lead to the unnecessary loss of a node group and as thus to a failure of the cluster.

Consider a setup where there are 4 data nodes A, B, C, and D running on 2 host computers `host1` and `host2`, and that these data nodes make up 2 node groups, as shown in the following table:

Table 22.104 Four data nodes A, B, C, D running on two host computers host1, host2; each data node belongs to one of two node groups.

Node Group	Nodes Running on `host1`	Nodes Running on `host2`
Node Group 0:	Node A	Node B
Node Group 1:	Node C	Node D

Suppose the heartbeats are transmitted in the order A->B->C->D->A. In this case, the loss of the heartbeat between the hosts causes node B to declare node A dead and node C to declare node B dead. This results in loss of Node Group 0, and so the cluster fails. On the other hand, if the order of transmission is A->B->D->C->A (and all other conditions remain as previously stated), the loss of the heartbeat causes nodes A and D to be declared dead; in this case, each node group has one surviving node, and the cluster survives.

The `HeartbeatOrder` configuration parameter makes the order of heartbeat transmission user-configurable. The default value for `HeartbeatOrder` is zero; allowing the default value to be used on all data nodes causes the order of heartbeat transmission to be determined by `NDB`. If this parameter is used, it must be set to a nonzero value (maximum 65535) for every data node in the cluster, and this value must be unique for each data node; this causes the heartbeat transmission to proceed from data node to data node in the order of their `HeartbeatOrder` values from lowest to highest (and then directly from the data node having the highest `HeartbeatOrder` to the data node having the lowest value, to complete the circle). The values need not be consecutive. For example, to force the heartbeat transmission order A->B->D->C->A in the scenario outlined previously, you could set the `HeartbeatOrder` values as shown here:

Table 22.105 HeartbeatOrder values to force a heartbeat transition order of A->B->D->C->A.

Node	`HeartbeatOrder` Value
A	10
B	20
C	30
D	25

To use this parameter to change the heartbeat transmission order in a running NDB Cluster, you must first set `HeartbeatOrder` for each data node in the cluster in the global configuration (`config.ini`) file (or files). To cause the change to take effect, you must perform either of the following:

- A complete shutdown and restart of the entire cluster.
- 2 rolling restarts of the cluster in succession. *All nodes must be restarted in the same order in both rolling restarts*.

You can use `DUMP 908` to observe the effect of this parameter in the data node logs.

- `ConnectCheckIntervalDelay`

Table 22.106 This table provides type and value information for the ConnectCheckIntervalDelay data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter enables connection checking between data nodes after one of them has failed heartbeat checks for 5 intervals of up to `HeartbeatIntervalDbDb` milliseconds.

Such a data node that further fails to respond within an interval of `ConnectCheckIntervalDelay` milliseconds is considered suspect, and is considered dead after two such intervals. This can be useful in setups with known latency issues.

The default value for this parameter is 0 (disabled).

- `TimeBetweenLocalCheckpoints`

Table 22.107 This table provides type and value information for the TimeBetweenLocalCheckpoints data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	number of 4-byte words, as a base-2 logarithm
Default	20
Range	0 - 31
Restart Type	N

This parameter is an exception in that it does not specify a time to wait before starting a new local checkpoint; rather, it is used to ensure that local checkpoints are not performed in a cluster where relatively few updates are taking place. In most clusters with high update rates, it is likely that a new local checkpoint is started immediately after the previous one has been completed.

The size of all write operations executed since the start of the previous local checkpoints is added. This parameter is also exceptional in that it is specified as the base-2 logarithm of the number of 4-byte words, so that the default value 20 means 4MB (4×2^{20}) of write operations, 21 would mean 8MB, and so on up to a maximum value of 31, which equates to 8GB of write operations.

All the write operations in the cluster are added together. Setting `TimeBetweenLocalCheckpoints` to 6 or less means that local checkpoints will be executed continuously without pause, independent of the cluster's workload.

- `TimeBetweenGlobalCheckpoints`

Table 22.108 This table provides type and value information for the TimeBetweenGlobalCheckpoints data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	2000
Range	20 - 32000
Restart Type	N

When a transaction is committed, it is committed in main memory in all nodes on which the data is mirrored. However, transaction log records are not flushed to disk as part of the commit. The reasoning behind this behavior is that having the transaction safely committed on at least two autonomous host machines should meet reasonable standards for durability.

It is also important to ensure that even the worst of cases—a complete crash of the cluster—is handled properly. To guarantee that this happens, all transactions taking place within a given interval are put into a global checkpoint, which can be thought of as a set of committed transactions that has been flushed to disk. In other words, as part of the commit process, a transaction is placed in a global checkpoint group. Later, this group's log records are flushed to disk, and then the entire group of transactions is safely committed to disk on all computers in the cluster.

This parameter defines the interval between global checkpoints. The default is 2000 milliseconds.

- `TimeBetweenGlobalCheckpointsTimeout`

Table 22.109 This table provides type and value information for the TimeBetweenGlobalCheckpointsTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	120000
Range	10 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter defines the minimum timeout between global checkpoints. The default is 120000 milliseconds.

- `TimeBetweenEpochs`

Table 22.110 This table provides type and value information for the TimeBetweenEpochs data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	100
Range	0 - 32000
Restart Type	N

This parameter defines the interval between synchronization epochs for NDB Cluster Replication. The default value is 100 milliseconds.

`TimeBetweenEpochs` is part of the implementation of “micro-GCPs”, which can be used to improve the performance of NDB Cluster Replication.

- `TimeBetweenEpochsTimeout`

Table 22.111 This table provides type and value information for the TimeBetweenEpochsTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	0
Range	0 - 256000
Restart Type	N

This parameter defines a timeout for synchronization epochs for NDB Cluster Replication. If a node fails to participate in a global checkpoint within the time determined by this parameter, the node is shut down. The default value is 0; in other words, the timeout is disabled.

`TimeBetweenEpochsTimeout` is part of the implementation of “micro-GCPs”, which can be used to improve the performance of NDB Cluster Replication.

The current value of this parameter and a warning are written to the cluster log whenever a GCP save takes longer than 1 minute or a GCP commit takes longer than 10 seconds.

Setting this parameter to zero has the effect of disabling GCP stops caused by save timeouts, commit timeouts, or both. The maximum possible value for this parameter is 256000 milliseconds.

- `MaxBufferedEpochs`

Table 22.112 This table provides type and value information for the MaxBufferedEpochs data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	epochs
Default	100
Range	0 - 100000
Restart Type	N

The number of unprocessed epochs by which a subscribing node can lag behind. Exceeding this number causes a lagging subscriber to be disconnected.

The default value of 100 is sufficient for most normal operations. If a subscribing node does lag enough to cause disconnections, it is usually due to network or scheduling issues with regard to processes or threads. (In rare circumstances, the problem may be due to a bug in the `NDB` client.) It may be desirable to set the value lower than the default when epochs are longer.

Disconnection prevents client issues from affecting the data node service, running out of memory to buffer data, and eventually shutting down. Instead, only the client is affected as a result of the disconnect (by, for example gap events in the binary log), forcing the client to reconnect or restart the process.

- `MaxBufferedEpochBytes`

Table 22.113 This table provides type and value information for the MaxBufferedEpochBytes data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	26214400
Range	26214400 (0x01900000) - 4294967039 (0xFFFFFEFF)
Restart Type	N

The total number of bytes allocated for buffering epochs by this node.

- `TimeBetweenInactiveTransactionAbortCheck`

Table 22.114 This table provides type and value information for the TimeBetweenInactiveTransactionAbortCheck data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	1000
Range	1000 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Timeout handling is performed by checking a timer on each transaction once for every interval specified by this parameter. Thus, if this parameter is set to 1000 milliseconds, every transaction will be checked for timing out once per second.

The default value is 1000 milliseconds (1 second).

- `TransactionInactiveTimeout`

Table 22.115 This table provides type and value information for the TransactionInactiveTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	[see text]
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter states the maximum time that is permitted to lapse between operations in the same transaction before the transaction is aborted.

The default for this parameter is `4G` (also the maximum). For a real-time database that needs to ensure that no transaction keeps locks for too long, this parameter should be set to a relatively small value. Setting it to 0 means that the application never times out. The unit is milliseconds.

- `TransactionDeadlockDetectionTimeout`

Table 22.116 This table provides type and value information for the TransactionDeadlockDetectionTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	1200
Range	50 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When a node executes a query involving a transaction, the node waits for the other nodes in the cluster to respond before continuing. This parameter sets the amount of time that the transaction can spend

executing within a data node, that is, the time that the transaction coordinator waits for each data node participating in the transaction to execute a request.

A failure to respond can occur for any of the following reasons:

- The node is “dead”
- The operation has entered a lock queue
- The node requested to perform the action could be heavily overloaded.

This timeout parameter states how long the transaction coordinator waits for query execution by another node before aborting the transaction, and is important for both node failure handling and deadlock detection.

The default timeout value is 1200 milliseconds (1.2 seconds).

The minimum for this parameter is 50 milliseconds.

- `DiskSyncSize`

Table 22.117 This table provides type and value information for the DiskSyncSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	4M
Range	32K - 4294967039 (0xFFFFFEFF)
Restart Type	N

This is the maximum number of bytes to store before flushing data to a local checkpoint file. This is done to prevent write buffering, which can impede performance significantly. This parameter is *not* intended to take the place of `TimeBetweenLocalCheckpoints`.

Note

When `ODirect` is enabled, it is not necessary to set `DiskSyncSize`; in fact, in such cases its value is simply ignored.

The default value is 4M (4 megabytes).

- `MaxDiskWriteSpeed`

Table 22.118 This table provides type and value information for the MaxDiskWriteSpeed data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	20M
Range	1M - 1024G
Restart Type	S

Set the maximum rate for writing to disk, in bytes per second, by local checkpoints and backup operations when no restarts (by this data node or any other data node) are taking place in this NDB Cluster.

For setting the maximum rate of disk writes allowed while this data node is restarting, use `MaxDiskWriteSpeedOwnRestart`. For setting the maximum rate of disk writes allowed while other data nodes are restarting, use `MaxDiskWriteSpeedOtherNodeRestart`. The minimum speed for disk writes by all LCPs and backup operations can be adjusted by setting `MinDiskWriteSpeed`.

- `MaxDiskWriteSpeedOtherNodeRestart`

Table 22.119 This table provides type and value information for the MaxDiskWriteSpeedOtherNodeRestart data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	50M
Range	1M - 1024G
Restart Type	S

Set the maximum rate for writing to disk, in bytes per second, by local checkpoints and backup operations when one or more data nodes in this NDB Cluster are restarting, other than this node.

For setting the maximum rate of disk writes allowed while this data node is restarting, use `MaxDiskWriteSpeedOwnRestart`. For setting the maximum rate of disk writes allowed when no data nodes are restarting anywhere in the cluster, use `MaxDiskWriteSpeed`. The minimum speed for disk writes by all LCPs and backup operations can be adjusted by setting `MinDiskWriteSpeed`.

- `MaxDiskWriteSpeedOwnRestart`

Table 22.120 This table provides type and value information for the MaxDiskWriteSpeedOwnRestart data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	200M
Range	1M - 1024G
Restart Type	S

Set the maximum rate for writing to disk, in bytes per second, by local checkpoints and backup operations while this data node is restarting.

For setting the maximum rate of disk writes allowed while other data nodes are restarting, use `MaxDiskWriteSpeedOtherNodeRestart`. For setting the maximum rate of disk writes allowed when no data nodes are restarting anywhere in the cluster, use `MaxDiskWriteSpeed`. The minimum speed for disk writes by all LCPs and backup operations can be adjusted by setting `MinDiskWriteSpeed`.

- `MinDiskWriteSpeed`

Table 22.121 This table provides type and value information for the MinDiskWriteSpeed data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	10M
Range	1M - 1024G
Restart Type	S

Set the minimum rate for writing to disk, in bytes per second, by local checkpoints and backup operations.

The maximum rates of disk writes allowed for LCPs and backups under various conditions are adjustable using the parameters `MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOwnRestart`, and `MaxDiskWriteSpeedOtherNodeRestart`. See the descriptions of these parameters for more information.

- `ArbitrationTimeout`

Table 22.122 This table provides type and value information for the ArbitrationTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	7500
Range	10 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter specifies how long data nodes wait for a response from the arbitrator to an arbitration message. If this is exceeded, the network is assumed to have split.

The default value is 7500 milliseconds (7.5 seconds).

- `Arbitration`

Table 22.123 This table provides type and value information for the Arbitration data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	enumeration
Default	Default
Range	Default, Disabled, WaitExternal
Restart Type	N

The `Arbitration` parameter enables a choice of arbitration schemes, corresponding to one of 3 possible values for this parameter:

- **Default.** This enables arbitration to proceed normally, as determined by the `ArbitrationRank` settings for the management and API nodes. This is the default value.
- **Disabled.** Setting `Arbitration = Disabled` in the `[ndbd default]` section of the `config.ini` file to accomplishes the same task as setting `ArbitrationRank` to 0 on all management and API nodes. When `Arbitration` is set in this way, any `ArbitrationRank` settings are ignored.
- **WaitExternal.** The `Arbitration` parameter also makes it possible to configure arbitration in such a way that the cluster waits until after the time determined by `ArbitrationTimeout` has passed for an external cluster manager application to perform arbitration instead of handling arbitration internally. This can be done by setting `Arbitration = WaitExternal` in the `[ndbd default]` section of the `config.ini` file. For best results with the `WaitExternal` setting, it is recommended that `ArbitrationTimeout` be 2 times as long as the interval required by the external cluster manager to perform arbitration.

Important

This parameter should be used only in the `[ndbd default]` section of the cluster configuration file. The behavior of the cluster is unspecified when `Arbitration` is set to different values for individual data nodes.

- `RestartSubscriberConnectTimeout`

Table 22.124 This table provides type and value information for the RestartSubscriberConnectTimeout data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	ms
Default	12000
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	S

This parameter determines the time that a data node waits for subscribing API nodes to connect. Once this timeout expires, any “missing” API nodes are disconnected from the cluster. To disable this timeout, set `RestartSubscriberConnectTimeout` to 0.

While this parameter is specified in milliseconds, the timeout itself is resolved to the next-greatest whole second.

Buffering and logging. Several `[ndbd]` configuration parameters enable the advanced user to have more control over the resources used by node processes and to adjust various buffer sizes at need.

These buffers are used as front ends to the file system when writing log records to disk. If the node is running in diskless mode, these parameters can be set to their minimum values without penalty due to the fact that disk writes are “faked” by the `NDB` storage engine's file system abstraction layer.

- `UndoIndexBuffer`

Table 22.125 This table provides type and value information for the UndoIndexBuffer data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	2M
Range	1M - 4294967039 (0xFFFFFEFF)
Restart Type	N

The UNDO index buffer, whose size is set by this parameter, is used during local checkpoints. The `NDB` storage engine uses a recovery scheme based on checkpoint consistency in conjunction with an operational REDO log. To produce a consistent checkpoint without blocking the entire system for writes, UNDO logging is done while performing the local checkpoint. UNDO logging is activated on a single table fragment at a time. This optimization is possible because tables are stored entirely in main memory.

The UNDO index buffer is used for the updates on the primary key hash index. Inserts and deletes rearrange the hash index; the NDB storage engine writes UNDO log records that map all physical changes to an index page so that they can be undone at system restart. It also logs all active insert operations for each fragment at the start of a local checkpoint.

Reads and updates set lock bits and update a header in the hash index entry. These changes are handled by the page-writing algorithm to ensure that these operations need no UNDO logging.

This buffer is 2MB by default. The minimum value is 1MB, which is sufficient for most applications. For applications doing extremely large or numerous inserts and deletes together with large transactions and large primary keys, it may be necessary to increase the size of this buffer. If this buffer is too small, the NDB storage engine issues internal error code 677 (`Index UNDO buffers overloaded`).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

- `UndoDataBuffer`

Table 22.126 This table provides type and value information for the UndoDataBuffer data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	16M
Range	1M - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets the size of the UNDO data buffer, which performs a function similar to that of the UNDO index buffer, except the UNDO data buffer is used with regard to data memory rather than index memory. This buffer is used during the local checkpoint phase of a fragment for inserts, deletes, and updates.

Because UNDO log entries tend to grow larger as more operations are logged, this buffer is also larger than its index memory counterpart, with a default value of 16MB.

This amount of memory may be unnecessarily large for some applications. In such cases, it is possible to decrease this size to a minimum of 1MB.

It is rarely necessary to increase the size of this buffer. If there is such a need, it is a good idea to check whether the disks can actually handle the load caused by database update activity. A lack of sufficient disk space cannot be overcome by increasing the size of this buffer.

If this buffer is too small and gets congested, the NDB storage engine issues internal error code 891 (`Data UNDO buffers overloaded`).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

- `RedoBuffer`

Table 22.127 This table provides type and value information for the RedoBuffer data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	32M
Range	1M - 4294967039 (0xFFFFFEFF)
Restart Type	N

All update activities also need to be logged. The REDO log makes it possible to replay these updates whenever the system is restarted. The NDB recovery algorithm uses a “fuzzy” checkpoint of the data together with the UNDO log, and then applies the REDO log to play back all changes up to the restoration point.

`RedoBuffer` sets the size of the buffer in which the REDO log is written. The default value is 32MB; the minimum value is 1MB.

If this buffer is too small, the `NDB` storage engine issues error code 1221 (`REDO log buffers overloaded`). For this reason, you should exercise care if you attempt to decrease the value of `RedoBuffer` as part of an online change in the cluster's configuration.

`ndbmtd` allocates a separate buffer for each LDM thread (see `ThreadConfig`). For example, with 4 LDM threads, an `ndbmtd` data node actually has 4 buffers and allocates `RedoBuffer` bytes to each one, for a total of `4 * RedoBuffer` bytes.

- `EventLogBufferSize`

Table 22.128 This table provides type and value information for the EventLogBufferSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13

Property	Value
Type or units	bytes
Default	8192
Range	0 - 64K
Restart Type	S

Controls the size of the circular buffer used for NDB log events within data nodes.

Controlling log messages. In managing the cluster, it is very important to be able to control the number of log messages sent for various event types to `stdout`. For each event category, there are 16 possible event levels (numbered 0 through 15). Setting event reporting for a given event category to level 15 means all event reports in that category are sent to `stdout`; setting it to 0 means that there will be no event reports made in that category.

By default, only the startup message is sent to `stdout`, with the remaining event reporting level defaults being set to 0. The reason for this is that these messages are also sent to the management server's cluster log.

An analogous set of levels can be set for the management client to determine which event levels to record in the cluster log.

- `LogLevelStartup`

Table 22.129 This table provides type and value information for the LogLevelStartup data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	1
Range	0 - 15
Restart Type	N

The reporting level for events generated during startup of the process.

The default level is 1.

- `LogLevelShutdown`

Table 22.130 This table provides type and value information for the LogLevelShutdown data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

The reporting level for events generated as part of graceful shutdown of a node.

The default level is 0.

- `LogLevelStatistic`

Table 22.131 This table provides type and value information for the LogLevelStatistic data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

The reporting level for statistical events such as number of primary key reads, number of updates, number of inserts, information relating to buffer usage, and so on.

The default level is 0.

- `LogLevelCheckpoint`

Table 22.132 This table provides type and value information for the LogLevelCheckpoint data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	log level
Default	0
Range	0 - 15
Restart Type	N

The reporting level for events generated by local and global checkpoints.

The default level is 0.

- `LogLevelNodeRestart`

Table 22.133 This table provides type and value information for the LogLevelNodeRestart data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

The reporting level for events generated during node restart.

The default level is 0.

- `LogLevelConnection`

  Table 22.134 This table provides type and value information for the LogLevelConnection data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

  The reporting level for events generated by connections between cluster nodes.

  The default level is 0.

- `LogLevelError`

  Table 22.135 This table provides type and value information for the LogLevelError data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

  The reporting level for events generated by errors and warnings by the cluster as a whole. These errors do not cause any node failure but are still considered worth reporting.

  The default level is 0.

- `LogLevelCongestion`

  Table 22.136 This table provides type and value information for the LogLevelCongestion data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	level
Default	0
Range	0 - 15
Restart Type	N

  The reporting level for events generated by congestion. These errors do not cause node failure but are still considered worth reporting.

  The default level is 0.

- `LogLevelInfo`

Table 22.137 This table provides type and value information for the LogLevelInfo data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

The reporting level for events generated for information about the general state of the cluster.

The default level is 0.

- `MemReportFrequency`

Table 22.138 This table provides type and value information for the MemReportFrequency data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter controls how often data node memory usage reports are recorded in the cluster log; it is an integer value representing the number of seconds between reports.

Each data node's data memory and index memory usage is logged as both a percentage and a number of 32 KB pages of `DataMemory`, as set in the `config.ini` file. For example, if `DataMemory` is equal to 100 MB, and a given data node is using 50 MB for data memory storage, the corresponding line in the cluster log might look like this:

```
2006-12-24 01:18:16 [MgmSrvr] INFO -- Node 2: Data usage is 50%(1280 32K pages of total 2560)
```

`MemReportFrequency` is not a required parameter. If used, it can be set for all cluster data nodes in the `[ndbd default]` section of `config.ini`, and can also be set or overridden for individual data nodes in the corresponding `[ndbd]` sections of the configuration file. The minimum value—which is also the default value—is 0, in which case memory reports are logged only when memory usage reaches certain percentages (80%, 90%, and 100%), as mentioned in the discussion of statistics events in Section 22.5.6.2, “NDB Cluster Log Events”.

- `StartupStatusReportFrequency`

Table 22.139 This table provides type and value information for the StartupStatusReportFrequency data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13

Property	Value
Type or units	seconds
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When a data node is started with the `--initial`, it initializes the redo log file during Start Phase 4 (see Section 22.5.1, “Summary of NDB Cluster Start Phases”). When very large values are set for `NoOfFragmentLogFiles`, `FragmentLogFileSize`, or both, this initialization can take a long time.You can force reports on the progress of this process to be logged periodically, by means of the `StartupStatusReportFrequency` configuration parameter. In this case, progress is reported in the cluster log, in terms of both the number of files and the amount of space that have been initialized, as shown here:

```
2009-06-20 16:39:23 [MgmSrvr] INFO -- Node 1: Local redo log file initialization status:
#Total files: 80, Completed: 60
#Total MBytes: 20480, Completed: 15557
2009-06-20 16:39:23 [MgmSrvr] INFO -- Node 2: Local redo log file initialization status:
#Total files: 80, Completed: 60
#Total MBytes: 20480, Completed: 15570
```

These reports are logged each `StartupStatusReportFrequency` seconds during Start Phase 4. If `StartupStatusReportFrequency` is 0 (the default), then reports are written to the cluster log only when at the beginning and at the completion of the redo log file initialization process.

Data Node Debugging Parameters

The following parameters are intended for use during testing or debugging of data nodes, and not for use in production.

- `DictTrace`

Table 22.140 This table provides type and value information for the DictTrace data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	undefined
Range	0 - 100
Restart Type	N

It is possible to cause logging of traces for events generated by creating and dropping tables using `DictTrace`. This parameter is useful only in debugging NDB kernel code. `DictTrace` takes an integer value. 0 is the default, and means no logging is performed; 1 enables trace logging, and 2 enables logging of additional `DBDICT` debugging output.

- `WatchdogImmediateKill`

Table 22.141 This table provides type and value information for the WatchDogImmediateKill data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	S

You can cause threads to be killed immediately whenever watchdog issues occur by enabling the `WatchdogImmediateKill` data node configuration parameter. This parameter should be used only when debugging or troubleshooting, to obtain trace files reporting exactly what was occurring the instant that execution ceased.

Backup parameters. The `[ndbd]` parameters discussed in this section define memory buffers set aside for execution of online backups.

- `BackupDataBufferSize`

Table 22.142 This table provides type and value information for the BackupDataBufferSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	16M
Range	512K - 4294967039 (0xFFFFFEFF)
Restart Type	N

In creating a backup, there are two buffers used for sending data to the disk. The backup data buffer is used to fill in data recorded by scanning a node's tables. Once this buffer has been filled to the level specified as `BackupWriteSize`, the pages are sent to disk. While flushing data to disk, the backup process can continue filling this buffer until it runs out of space. When this happens, the backup process pauses the scan and waits until some disk writes have completed freeing up memory so that scanning may continue.

The default value for this parameter is 16MB. The minimum is 512K.

- `BackupDiskWriteSpeedPct`

Table 22.143 This table provides type and value information for the BackupDiskWriteSpeedPct data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	percent
Default	50
Range	0 - 90

Property	Value
Restart Type	N

During normal operation, data nodes attempt to maximize the disk write speed used for local checkpoints and backups while remaining within the bounds set by `MinDiskWriteSpeed` and `MaxDiskWriteSpeed`. Disk write throttling gives each LDM thread an equal share of the total budget. This allows parallel LCPs to take place without exceeding the disk I/O budget. Because a backup is executed by only one LDM thread, this effectively caused a budget cut, resulting in longer backup completion times, and—if the rate of change is sufficiently high—in failure to complete the backup when the backup log buffer fill rate is higher than the achievable write rate.

This problem can be addressed by using the `BackupDiskWriteSpeedPct` configuration parameter, which takes a value in the range 0-90 (inclusive) which is interpreted as the percentage of the node's maximum write rate budget that is reserved prior to sharing out the remainder of the budget among LDM threads for LCPs. The LDM thread running the backup receives the whole write rate budget for the backup, plus its (reduced) share of the write rate budget for local checkpoints.

The default value for this parameter is 50 (interpreted as 50%).

- `BackupLogBufferSize`

Table 22.144 This table provides type and value information for the BackupLogBufferSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	16M
Range	2M - 4294967039 (0xFFFFFEFF)
Restart Type	N

The backup log buffer fulfills a role similar to that played by the backup data buffer, except that it is used for generating a log of all table writes made during execution of the backup. The same principles apply for writing these pages as with the backup data buffer, except that when there is no more space in the backup log buffer, the backup fails. For that reason, the size of the backup log buffer must be large enough to handle the load caused by write activities while the backup is being made. See Section 22.5.3.3, “Configuration for NDB Cluster Backups”.

The default value for this parameter should be sufficient for most applications. In fact, it is more likely for a backup failure to be caused by insufficient disk write speed than it is for the backup log buffer to become full. If the disk subsystem is not configured for the write load caused by applications, the cluster is unlikely to be able to perform the desired operations.

It is preferable to configure cluster nodes in such a manner that the processor becomes the bottleneck rather than the disks or the network connections.

The default value for this parameter is 16MB.

- `BackupMemory`

Table 22.145 This table provides type and value information for the BackupMemory data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	32M
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter is deprecated, and subject to removal in a future version of NDB Cluster. Any setting made for it is ignored.

- `BackupReportFrequency`

Table 22.146 This table provides type and value information for the BackupReportFrequency data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	seconds
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter controls how often backup status reports are issued in the management client during a backup, as well as how often such reports are written to the cluster log (provided cluster event logging is configured to permit it—see Logging and checkpointing). `BackupReportFrequency` represents the time in seconds between backup status reports.

The default value is 0.

- `BackupWriteSize`

Table 22.147 This table provides type and value information for the BackupWriteSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	256K
Range	32K - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter specifies the default size of messages written to disk by the backup log and backup data buffers.

The default value for this parameter is 256KB.

- `BackupMaxWriteSize`

Table 22.148 This table provides type and value information for the BackupMaxWriteSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	1M
Range	256K - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter specifies the maximum size of messages written to disk by the backup log and backup data buffers.

The default value for this parameter is 1MB.

Note

The location of the backup files is determined by the `BackupDataDir` data node configuration parameter.

Additional requirements. When specifying these parameters, the following relationships must hold true. Otherwise, the data node will be unable to start.

- `BackupDataBufferSize >= BackupWriteSize + 188KB`
- `BackupLogBufferSize >= BackupWriteSize + 16KB`
- `BackupMaxWriteSize >= BackupWriteSize`

NDB Cluster Realtime Performance Parameters

The `[ndbd]` parameters discussed in this section are used in scheduling and locking of threads to specific CPUs on multiprocessor data node hosts.

Note

To make use of these parameters, the data node process must be run as system root.

- `LockExecuteThreadToCPU`

Table 22.149 This table provides type and value information for the LockExecuteThreadToCPU data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	set of CPU IDs
Default	0
Range	...
Restart Type	N

When used with `ndbd`, this parameter (now a string) specifies the ID of the CPU assigned to handle the `NDBCLUSTER` execution thread. When used with `ndbmtd`, the value of this parameter is a comma-

separated list of CPU IDs assigned to handle execution threads. Each CPU ID in the list should be an integer in the range 0 to 65535 (inclusive).

The number of IDs specified should match the number of execution threads determined by `MaxNoOfExecutionThreads`. However, there is no guarantee that threads are assigned to CPUs in any given order when using this parameter. You can obtain more finely-grained control of this type using `ThreadConfig`.

`LockExecuteThreadToCPU` has no default value.

- `LockMaintThreadsToCPU`

Table 22.150 This table provides type and value information for the LockMaintThreadsToCPU data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	CPU ID
Default	0
Range	0 - 64K
Restart Type	N

This parameter specifies the ID of the CPU assigned to handle `NDBCLUSTER` maintenance threads.

The value of this parameter is an integer in the range 0 to 65535 (inclusive). *There is no default value*.

- `RealtimeScheduler`

Table 22.151 This table provides type and value information for the RealtimeScheduler data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Setting this parameter to 1 enables real-time scheduling of data node threads.

The default is 0 (scheduling disabled).

- `SchedulerExecutionTimer`

Table 22.152 This table provides type and value information for the SchedulerExecutionTimer data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	µs
Default	50

Property	Value
Range	0 - 11000
Restart Type	N

This parameter specifies the time in microseconds for threads to be executed in the scheduler before being sent. Setting it to 0 minimizes the response time; to achieve higher throughput, you can increase the value at the expense of longer response times.

The default is 50 µsec, which our testing shows to increase throughput slightly in high-load cases without materially delaying requests.

- `SchedulerResponsiveness`

Table 22.153 This table provides type and value information for the SchedulerResponsiveness data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	5
Range	0 - 10
Restart Type	S

Set the balance in the `NDB` scheduler between speed and throughput. This parameter takes an integer whose value is in the range 0-10 inclusive, with 5 as the default. Higher values provide better response times relative to throughput. Lower values provide increased throughput at the expense of longer response times.

- `SchedulerSpinTimer`

Table 22.154 This table provides type and value information for the SchedulerSpinTimer data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	µs
Default	0
Range	0 - 500
Restart Type	N

This parameter specifies the time in microseconds for threads to be executed in the scheduler before sleeping.

The default value is 0.

- `BuildIndexThreads`

Table 22.155 This table provides type and value information for the BuildIndexThreads data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	128
Range	0 - 128
Restart Type	S

This parameter determines the number of threads to create when rebuilding ordered indexes during a system or node start, as well as when running `ndb_restore --rebuild-indexes`. It is supported only when there is more than one fragment for the table per data node (for example, when `COMMENT="NDB_TABLE=PARTITION_BALANCE=FOR_RA_BY_LDM_X_2"` is used with `CREATE TABLE`).

Setting this parameter to 0 (the default) disables multithreaded building of ordered indexes.

This parameter is supported when using `ndbd` or `ndbmtd`.

You can enable multithreaded builds during data node initial restarts by setting the `TwoPassInitialNodeRestartCopy` data node configuration parameter to `TRUE`.

- `TwoPassInitialNodeRestartCopy`

Table 22.156 This table provides type and value information for the TwoPassInitialNodeRestartCopy data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	true
Range	true, false
Restart Type	N

Multithreaded building of ordered indexes can be enabled for initial restarts of data nodes by setting this configuration parameter to `true` (the default value), which enables two-pass copying of data during initial node restarts.

You must also set `BuildIndexThreads` to a nonzero value.

- `Numa`

Table 22.157 This table provides type and value information for the Numa data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	1

Property	Value
Range	...
Restart Type	N

This parameter determines whether Non-Uniform Memory Access (NUMA) is controlled by the operating system or by the data node process, whether the data node uses `ndbd` or `ndbmtd`. By default, `NDB` attempts to use an interleaved NUMA memory allocation policy on any data node where the host operating system provides NUMA support.

Setting `Numa = 0` means that the datanode process does not itself attempt to set a policy for memory allocation, and permits this behavior to be determined by the operating system, which may be further guided by the separate `numactl` tool. That is, `Numa = 0` yields the system default behavior, which can be customised by `numactl`. For many Linux systems, the system default behavior is to allocate socket-local memory to any given process at allocation time. This can be problematic when using `ndbmtd`; this is because `nbdmtd` allocates all memory at startup, leading to an imbalance, giving different access speeds for different sockets, especially when locking pages in main memory.

Setting `Numa = 1` means that the data node process uses `libnuma` to request interleaved memory allocation. (This can also be accomplished manually, on the operating system level, using `numactl`.) Using interleaved allocation in effect tells the data node process to ignore non-uniform memory access but does not attempt to take any advantage of fast local memory; instead, the data node process tries to avoid imbalances due to slow remote memory. If interleaved allocation is not desired, set `Numa` to 0 so that the desired behavior can be determined on the operating system level.

The `Numa` configuration parameter is supported only on Linux systems where `libnuma.so` is available.

Multi-Threading Configuration Parameters (ndbmtd). `ndbmtd` runs by default as a single-threaded process and must be configured to use multiple threads, using either of two methods, both of which require setting configuration parameters in the `config.ini` file. The first method is simply to set an appropriate value for the `MaxNoOfExecutionThreads` configuration parameter. A second method, makes it possible to set up more complex rules for `ndbmtd` multithreading using `ThreadConfig`. The next few paragraphs provide information about these parameters and their use with multithreaded data nodes.

Note

A backup using parallelism on the data nodes requires that multiple LDMs are in use on all data nodes in the cluster prior to taking the backup. For more information, see Section 22.5.3.5, “Taking an NDB with Parallel Data Nodes”, as well as Section 22.4.23.2, “Restoring from a backup taken in parallel”.

- `MaxNoOfExecutionThreads`

Table 22.158 This table provides type and value information for the MaxNoOfExecutionThreads multi-threaded data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	2
Range	2 - 72
Restart Type	S

This parameter directly controls the number of execution threads used by `ndbmtd`, up to a maximum of 72. Although this parameter is set in `[ndbd]` or `[ndbd default]` sections of the `config.ini` file, it is exclusive to `ndbmtd` and does not apply to `ndbd`.

Setting `MaxNoOfExecutionThreads` sets the number of threads for each type as determined by a matrix in the file `storage/ndb/src/kernel/vm/mt_thr_config.cpp`. This table shows these numbers of threads for possible values of `MaxNoOfExecutionThreads`.

Table 22.159 MaxNoOfExecutionThreads values and the corresponding number of threads by thread type (LQH, TC, Send, Receive).

`MaxNoOfExecutionThreads` Value	LDM Threads	TC Threads	Send Threads	Receive Threads
0 .. 3	1	0	0	1
4 .. 6	2	0	0	1
7 .. 8	4	0	0	1
9	4	2	0	1
10	4	2	1	1
11	4	3	1	1
12	6	2	1	1
13	6	3	1	1
14	6	3	1	2
15	6	3	2	2
16	8	3	1	2
17	8	4	1	2
18	8	4	2	2
19	8	5	2	2
20	10	4	2	2
21	10	5	2	2
22	10	5	2	3
23	10	6	2	3
24	12	5	2	3
25	12	6	2	3
26	12	6	3	3
27	12	7	3	3
28	12	7	3	4
29	12	8	3	4
30	12	8	4	4
31	12	9	4	4
32	16	8	3	3
33	16	8	3	4
34	16	8	4	4

MaxNoOfExecutionThreads Value	LDM Threads	TC Threads	Send Threads	Receive Threads
35	16	9	4	4
36	16	10	4	4
37	16	10	4	5
38	16	11	4	5
39	16	11	5	5
40	20	10	4	4
41	20	10	4	5
42	20	11	4	5
43	20	11	5	5
44	20	12	5	5
45	20	12	5	6
46	20	13	5	6
47	20	13	6	6
48	24	12	5	5
49	24	12	5	6
50	24	13	5	6
51	24	13	6	6
52	24	14	6	6
53	24	14	6	7
54	24	15	6	7
55	24	15	7	7
56	24	16	7	7
57	24	16	7	8
58	24	17	7	8
59	24	17	8	8
60	24	18	8	8
61	24	18	8	9
62	24	19	8	9
63	24	19	9	9
64	32	16	7	7
65	32	16	7	8
66	32	17	7	8
67	32	17	8	8
68	32	18	8	8
69	32	18	8	9
70	32	19	8	9
71	32	20	8	9

MaxNoOfExecutionThreads Value	LDM Threads	TC Threads	Send Threads	Receive Threads
72	32	20	8	10

There is always one SUMA (replication) thread.

NoOfFragmentLogParts should be set equal to the number of LDM threads used by ndbmtd, as determined by the setting for this parameter. This ratio should not be any greater than 4:1; a configuration in which this is the case is specifically disallowed.

The number of LDM threads also determines the number of partitions used by an NDB table that is not explicitly partitioned; this is the number of LDM threads times the number of data nodes in the cluster. (If ndbd is used on the data nodes rather than ndbmtd, then there is always a single LDM thread; in this case, the number of partitions created automatically is simply equal to the number of data nodes. See Section 22.1.2, “NDB Cluster Nodes, Node Groups, Replicas, and Partitions”, for more information.

Adding large tablespaces for Disk Data tables when using more than the default number of LDM threads may cause issues with resource and CPU usage if the disk page buffer is insufficiently large; see the description of the DiskPageBufferMemory configuration parameter, for more information.

The thread types are described later in this section (see ThreadConfig).

Setting this parameter outside the permitted range of values causes the management server to abort on startup with the error `Error line number: Illegal value value for parameter MaxNoOfExecutionThreads`.

For MaxNoOfExecutionThreads, a value of 0 or 1 is rounded up internally by NDB to 2, so that 2 is considered this parameter's default and minimum value.

MaxNoOfExecutionThreads is generally intended to be set equal to the number of CPU threads available, and to allocate a number of threads of each type suitable to typical workloads. It does not assign particular threads to specified CPUs. For cases where it is desirable to vary from the settings provided, or to bind threads to CPUs, you should use ThreadConfig instead, which allows you to allocate each thread directly to a desired type, CPU, or both.

The multithreaded data node process always spawns, at a minimum, the threads listed here:

- 1 local query handler (LDM) thread
- 1 receive thread
- 1 subscription manager (SUMA or replication) thread

For a MaxNoOfExecutionThreads value of 8 or less, no TC threads are created, and TC handling is instead performed by the main thread.

Changing the number of LDM threads normally requires a system restart, whether it is changed using this parameter or ThreadConfig, but it is possible to effect the change using a node initial restart (*NI*) provided the following two conditions are met:

- Each LDM thread handles a maximum of 8 fragments, and
- The total number of table fragments is an integer multiple of the number of LDM threads.

In NDB 8.0, an initial restart is *not* required to effect a change in this parameter, as it was in some older versions of NDB Cluster.

- `NoOfFragmentLogParts`

Table 22.160 This table provides type and value information for the NoOfFragmentLogParts multi-threaded data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	4
Range	4, 8, 12, 16, 24, 32
Restart Type	IN

Set the number of log file groups for redo logs belonging to this `ndbmtd`. The maximum value is 32; the value set must be an even multiple of 4.

`NoOfFragmentLogParts` should be set equal to the number of LDM threads used by `ndbmtd` as determined by the setting for `MaxNoOfExecutionThreads`. A configuration using more than 4 redo log parts per LDM is disallowed.

See the description of `MaxNoOfExecutionThreads` for more information.

- `ThreadConfig`

Table 22.161 This table provides type and value information for the ThreadConfig multi-threaded data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	string
Default	''
Range	...
Restart Type	SI

This parameter is used with `ndbmtd` to assign threads of different types to different CPUs. Its value is a string whose format has the following syntax:

```
ThreadConfig := entry[,entry[,...]]

entry := type={param[,param[,...]]}

type := ldm | main | recv | send | rep | io | tc | watchdog | idxbld

param := count=number
  | cpubind=cpu_list
  | cpuset=cpu_list
  | spintime=number
  | realtime={0|1}
  | nosend={0|1}
  | thread_prio={0..10}
  | cpubind_exclusive=cpu_list
  | cpuset_exclusive=cpu_list
```

The curly braces (`{...}`) surrounding the list of parameters are required, even if there is only one parameter in the list.

A *param* (parameter) specifies any or all of the following information:

- The number of threads of the given type (count).
- The set of CPUs to which the threads of the given type are to be nonexclusively bound. This is determined by either one of cpubind or cpuset). cpubind causes each thread to be bound (nonexclusively) to a CPU in the set; cpuset means that each thread is bound (nonexclusively) to the set of CPUs specified.

  On Solaris, you can instead specify a set of CPUs to which the threads of the given type are to be bound exclusively. cpubind_exclusive causes each thread to be bound exclusively to a CPU in the set; cpuset_exclsuive means that each thread is bound exclusively to the set of CPUs specified.

  Only one of cpubind, cpuset, cpubind_exclusive, or cpuset_exclusive can be provided in a single configuration.
- spintime determines the wait time in microseconds the thread spins before going to sleep.

  The default value for spintime is the value of the SchedulerSpinTimer data node configuration parameter.

  spintime does not apply to I/O threads, watchdog, or offline index build threads, and so cannot be set for these thread types.
- realtime can be set to 0 or 1. If it is set to 1, the threads run with real-time priority. This also means that thread_prio cannot be set.

  The realtime parameter is set by default to the value of the RealtimeScheduler data node configuration parameter.

  realtime cannot be set for offline index build threads.
- By setting nosend to 1, you can prevent a main, ldm, rep, or tc thread from assisting the send threads. This parameter is 0 by default, and cannot be used with other types of threads.
- thread_prio is a thread priority level that can be set from 0 to 10, with 10 representing the greatest priority. The default is 5. The precise effects of this parameter are platform-specific, and are described later in this section.

  The thread priority level cannot be set for offline index build threads.

thread_prio settings and effects by platform. The implementation of thread_prio differs between Linux/FreeBSD, Solaris, and Windows. In the following list, we discuss its effects on each of these platforms in turn:

- *Linux and FreeBSD*: We map thread_prio to a value to be supplied to the nice system call. Since a lower niceness value for a process indicates a higher process priority, increasing thread_prio has the effect of lowering the nice value.

Table 22.162 Mapping of thread_prio to nice values on Linux and FreeBSD

thread_prio value	nice value
0	19
1	16

thread_prio value	nice value
2	12
3	8
4	4
5	0
6	-4
7	-8
8	-12
9	-16
10	-20

Some operating systems may provide for a maximum process niceness level of 20, but this is not supported by all targeted versions; for this reason, we choose 19 as the maximum `nice` value that can be set.

- *Solaris*: Setting `thread_prio` on Solaris sets the Solaris FX priority, with mappings as shown in the following table:

Table 22.163 Mapping of thread_prio to FX priority on Solaris

thread_prio value	Solaris FX priority
0	15
1	20
2	25
3	30
4	35
5	40
6	45
7	50
8	55
9	59
10	60

A `thread_prio` setting of 9 is mapped on Solaris to the special FX priority value 59, which means that the operating system also attempts to force the thread to run alone on its own CPU core.

- *Windows*: We map `thread_prio` to a Windows thread priority value passed to the Windows API `SetThreadPriority()` function. This mapping is shown in the following table:

Table 22.164 Mapping of thread_prio to Windows thread priority

thread_prio value	Windows thread priority
0 - 1	THREAD_PRIORITY_LOWEST
2 - 3	THREAD_PRIORITY_BELOW_NORMAL
4 - 5	THREAD_PRIORITY_NORMAL

`thread_prio` value	Windows thread priority
6 - 7	`THREAD_PRIORITY_ABOVE_NORMAL`
8 - 10	`THREAD_PRIORITY_HIGHEST`

The *`type`* attribute represents an NDB thread type. The thread types supported, and the range of permitted `count` values for each, are provided in the following list:

- `ldm`: Local query handler (`DBLQH` kernel block) that handles data. The more LDM threads that are used, the more highly partitioned the data becomes. Each LDM thread maintains its own sets of data and index partitions, as well as its own redo log. The value set for `ldm` must be one of the values 1, 2, 4, 6, 8, 12, 16, 24, or 32.

  Changing the number of LDM threads normally requires a system restart to be effective and safe for cluster operations, this requirement is relaxed in certain cases, as explained later in this section. This is also true when this is done using `MaxNoOfExecutionThreads`.

  Adding large tablespaces (hundreds of gigabytes or more) for Disk Data tables when using more than the default number of LDMs may cause issues with resource and CPU usage if `DiskPageBufferMemory` is not sufficiently large.

- `tc`: Transaction coordinator thread (`DBTC` kernel block) containing the state of an ongoing transaction. The maximum number of TC threads is 32.

  Optimally, every new transaction can be assigned to a new TC thread. In most cases 1 TC thread per 2 LDM threads is sufficient to guarantee that this can happen. In cases where the number of writes is relatively small when compared to the number of reads, it is possible that only 1 TC thread per 4 LQH threads is required to maintain transaction states. Conversely, in applications that perform a great many updates, it may be necessary for the ratio of TC threads to LDM threads to approach 1 (for example, 3 TC threads to 4 LDM threads).

  Setting `tc` to 0 causes TC handling to be done by the main thread. In most cases, this is effectively the same as setting it to 1.

  Range: 0 - 32

- `main`: Data dictionary and transaction coordinator (`DBDIH` and `DBTC` kernel blocks), providing schema management. This is always handled by a single dedicated thread.

  Range: 1 only.

- `recv`: Receive thread (`CMVMI` kernel block). Each receive thread handles one or more sockets for communicating with other nodes in an NDB Cluster, with one socket per node. NDB Cluster supports multiple receive threads; the maximum is 16 such threads.

  Range: 1 - 16

- `send`: Send thread (`CMVMI` kernel block). To increase throughput, it is possible to perform sends from one or more separate, dedicated threads (maximum 8).

  Previously, all threads handled their own sending directly; this can still be made to happen by setting the number of send threads to 0 (this also happens when `MaxNoOfExecutionThreads` is set less than 10). While doing so can have an adeverse impact on throughput, it can also in some cases provide decreased latency.

  Range: 0 - 16

- `rep`: Replication thread (`SUMA` kernel block). Asynchronous replication operations are always handled by a single, dedicated thread.

  Range: 1 only.

- `io`: File system and other miscellaneous operations. These are not demanding tasks, and are always handled as a group by a single, dedicated I/O thread.

  Range: 1 only.

- `watchdog`: Parameters settings associated with this type are actually applied to several threads, each having a specific use. These threads include the `SocketServer` thread, which receives connection setups from other nodes; the `SocketClient` thread, which attempts to set up connections to other nodes; and the thread watchdog thread that checks that threads are progressing.

  Range: 1 only.

- `idxbld`: Offline index build threads. Unlike the other thread types listed previously, which are permanent, these are temporary threads which are created and used only during node or system restarts, or when running `ndb_restore --rebuild-indexes`. They may be bound to CPU sets which overlap with CPU sets bound to permanent thread types.

  `thread_prio`, `realtime`, and `spintime` values cannot be set for offline index build threads. In addition, `count` is ignored for this type of thread.

  If `idxbld` is not specified, the default behavior is as follows:

  - Offline index build threads are not bound if the I/O thread is also not bound, and these threads use any available cores.
  - If the I/O thread is bound, then the offline index build threads are bound to the entire set of bound threads, due to the fact that there should be no other tasks for these threads to perform.

  Range: 0 - 1.

Changing `ThreadCOnfig` normally requires a system initial restart, but this requirement can be relaxed under certain cirumstances:

- If, following the change, the number of LDM threads remains the same as before, nothing more than a simple node restart (rolling restart, or *N*) is required to implement the change.
- Otherwise (that is, if the number of LDM threads changes), it is still possible to effect the change using a node initial restart (*NI*) provided the following two conditions are met:

  a. Each LDM thread handles a maximum of 8 fragments, and

  b. The total number of table fragments is an integer multiple of the number of LDM threads.

In any other case, a system initial restart is needed to change this parameter.

`NDB` can distinguish between thread types by both of the following criteria:

- Whether the thread is an execution thread. Threads of type `main`, `ldm`, `recv`, `rep`, `tc`, and `send` are execution threads; `io`, `watchdog`, and `idxbld` threads are not considered execution threads.
- Whether the allocation of threads to a given task is permanent or temporary. Currently all thread types except `idxbld` are considered permanent; `idxbld` threads are regarded as temporary threads.

Simple examples:

```
# Example 1.

ThreadConfig=ldm={count=2,cpubind=1,2},main={cpubind=12},rep={cpubind=11}

# Example 2.

Threadconfig=main={cpubind=0},ldm={count=4,cpubind=1,2,5,6},io={cpubind=3}
```

It is usually desirable when configuring thread usage for a data node host to reserve one or more number of CPUs for operating system and other tasks. Thus, for a host machine with 24 CPUs, you might want to use 20 CPU threads (leaving 4 for other uses), with 8 LDM threads, 4 TC threads (half the number of LDM threads), 3 send threads, 3 receive threads, and 1 thread each for schema management, asynchronous replication, and I/O operations. (This is almost the same distribution of threads used when `MaxNoOfExecutionThreads` is set equal to 20.) The following `ThreadConfig` setting performs these assignments, additionally binding all of these threads to specific CPUs:

```
ThreadConfig=ldm{count=8,cpubind=1,2,3,4,5,6,7,8},main={cpubind=9},io={cpubind=9}, \
rep={cpubind=10},tc{count=4,cpubind=11,12,13,14},recv={count=3,cpubind=15,16,17}, \
send{count=3,cpubind=18,19,20}
```

It should be possible in most cases to bind the main (schema management) thread and the I/O thread to the same CPU, as we have done in the example just shown.

The following example incorporates groups of CPUs defined using both `cpuset` and `cpubind`, as well as use of thread prioritization.

```
ThreadConfig=ldm={count=4,cpuset=0-3,thread_prio=8,spintime=200}, \
ldm={count=4,cpubind=4-7,thread_prio=8,spintime=200}, \
tc={count=4,cpuset=8-9,thread_prio=6},send={count=2,thread_prio=10,cpubind=10-11}, \
main={count=1,cpubind=10},rep={count=1,cpubind=11}
```

In this case we create two LDM groups; the first uses `cpubind` and the second uses `cpuset`. `thread_prio` and `spintime` are set to the same values for each group. This means there are eight LDM threads in total. (You should ensure that `NoOfFragmentLogParts` is also set to 8.) The four TC threads use only two CPUs; it is possible when using `cpuset` to specify fewer CPUs than threads in the group. (This is not true for `cpubind`.) The send threads use two threads using `cpubind` to bind these threads to CPUs 10 and 11. The main and rep threads can reuse these CPUs.

This example shows how `ThreadConfig` and `NoOfFragmentLogParts` might be set up for a 24-CPU host with hyperthreading, leaving CPUs 10, 11, 22, and 23 available for operating system functions and interrupts:

```
NoOfFragmentLogParts=10
ThreadConfig=ldm={count=10,cpubind=0-4,12-16,thread_prio=9,spintime=200}, \
tc={count=4,cpuset=6-7,18-19,thread_prio=8},send={count=1,cpuset=8}, \
recv={count=1,cpuset=20},main={count=1,cpuset=9,21},rep={count=1,cpuset=9,21}, \
io={count=1,cpuset=9,21,thread_prio=8},watchdog={count=1,cpuset=9,21,thread_prio=9}
```

The next few examples include settings for `idxbld`. The first two of these demonstrate how a CPU set defined for `idxbld` can overlap those specified for other (permanent) thread types, the first using `cpuset` and the second using `cpubind`:

```
ThreadConfig=main,ldm={count=4,cpuset=1-4},tc={count=4,cpuset=5,6,7}, \
io={cpubind=8},idxbld={cpuset=1-8}
```

```
ThreadConfig=main,ldm={count=1,cpubind=1},idxbld={count=1,cpubind=1}
```

The next example specifies a CPU for the I/O thread, but not for the index build threads:

```
ThreadConfig=main,ldm={count=4,cpuset=1-4},tc={count=4,cpuset=5,6,7}, \
io={cpubind=8}
```

Since the `ThreadConfig` setting just shown locks threads to eight cores numbered 1 through 8, it is equivalent to the setting shown here:

```
ThreadConfig=main,ldm={count=4,cpuset=1-4},tc={count=4,cpuset=5,6,7}, \
io={cpubind=8},idxbld={cpuset=1,2,3,4,5,6,7,8}
```

In order to take advantage of the enhanced stability that the use of `ThreadConfig` offers, it is necessary to insure that CPUs are isolated, and that they not subject to interrupts, or to being scheduled for other tasks by the operating system. On many Linux systems, you can do this by setting `IRQBALANCE_BANNED_CPUS` in `/etc/sysconfig/irqbalance` to `0xFFFFF0`, and by using the `isolcpus` boot option in `grub.conf`. For specific information, see your operating system or platform documentation.

Disk Data Configuration Parameters. Configuration parameters affecting Disk Data behavior include the following:

- `DiskPageBufferEntries`

  Table 22.165 This table provides type and value information for the DiskPageBufferEntries data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	32K pages
Default	10
Range	1 - 1000
Restart Type	N

  This is the number of page entries (page references) to allocate. It is specified as a number of 32K pages in `DiskPageBufferMemory`. The default is sufficient for most cases but you may need to increase the value of this parameter if you encounter problems with very large transactions on Disk Data tables. Each page entry requires approximately 100 bytes.

- `DiskPageBufferMemory`

  Table 22.166 This table provides type and value information for the DiskPageBufferMemory data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	64M
Range	4M - 1T
Restart Type	N

This determines the amount of space used for caching pages on disk, and is set in the [ndbd] or [ndbd default] section of the config.ini file. It is measured in bytes. Each page takes up 32 KB. This means that NDB Cluster Disk Data storage always uses *N* * 32 KB memory where *N* is some nonnegative integer.

The default value for this parameter is 64M (2000 pages of 32 KB each).

If the value for DiskPageBufferMemory is set too low in conjunction with using more than the default number of LDM threads in ThreadConfig (for example {ldm=6...}), problems can arise when trying to add a large (for example 500G) data file to a disk-based NDB table, wherein the process takes indefinitely long while occupying one of the CPU cores.

This is due to the fact that, as part of adding a data file to a tablespace, extent pages are locked into memory in an extra PGMAN worker thread, for quick metadata access. When adding a large file, this worker has insufficient memory for all of the data file metadata. In such cases, you should either increase DiskPageBufferMemory, or add smaller tablespace files. You may also need to adjust DiskPageBufferEntries.

You can query the ndbinfo.diskpagebuffer table to help determine whether the value for this parameter should be increased to minimize unnecessary disk seeks. See Section 22.5.10.20, "The ndbinfo diskpagebuffer Table", for more information.

- SharedGlobalMemory

Table 22.167 This table provides type and value information for the SharedGlobalMemory data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	128M
Range	0 - 64T
Restart Type	N

This parameter determines the amount of memory that is used for log buffers, disk operations (such as page requests and wait queues), and metadata for tablespaces, log file groups, UNDO files, and data files. The shared global memory pool also provides memory used for satisfying the memory requirements of the UNDO_BUFFER_SIZE option used with CREATE LOGFILE GROUP and ALTER LOGFILE GROUP statements, including any default value implied for this options by the setting of the InitialLogFileGroup data node configuration parameter. SharedGlobalMemory can be set in the [ndbd] or [ndbd default] section of the config.ini configuration file, and is measured in bytes.

The default value is 128M.

- DiskIOThreadPool

Table 22.168 This table provides type and value information for the DiskIOThreadPool data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	threads

Property	Value
Default	2
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter determines the number of unbound threads used for Disk Data file access. Before `DiskIOThreadPool` was introduced, exactly one thread was spawned for each Disk Data file, which could lead to performance issues, particularly when using very large data files. With `DiskIOThreadPool`, you can—for example—access a single large data file using several threads working in parallel.

This parameter applies to Disk Data I/O threads only.

The optimum value for this parameter depends on your hardware and configuration, and includes these factors:

- **Physical distribution of Disk Data files.** You can obtain better performance by placing data files, undo log files, and the data node file system on separate physical disks. If you do this with some or all of these sets of files, then you can set `DiskIOThreadPool` higher to enable separate threads to handle the files on each disk.
- **Disk performance and types.** The number of threads that can be accommodated for Disk Data file handling is also dependent on the speed and throughput of the disks. Faster disks and higher throughput allow for more disk I/O threads. Our test results indicate that solid-state disk drives can handle many more disk I/O threads than conventional disks, and thus higher values for `DiskIOThreadPool`.

The default value for this parameter is 2.

- **Disk Data file system parameters.** The parameters in the following list make it possible to place NDB Cluster Disk Data files in specific directories without the need for using symbolic links.

  - `FileSystemPathDD`

  Table 22.169 This table provides type and value information for the FileSystemPathDD data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	filename
Default	[see text]
Range	...
Restart Type	IN

  If this parameter is specified, then NDB Cluster Disk Data data files and undo log files are placed in the indicated directory. This can be overridden for data files, undo log files, or both, by specifying values for `FileSystemPathDataFiles`, `FileSystemPathUndoFiles`, or both, as explained for these parameters. It can also be overridden for data files by specifying a path in the `ADD DATAFILE` clause of a `CREATE TABLESPACE` or `ALTER TABLESPACE` statement, and for undo log files by specifying a path in the `ADD UNDOFILE` clause of a `CREATE LOGFILE GROUP` or `ALTER LOGFILE GROUP` statement. If `FileSystemPathDD` is not specified, then `FileSystemPath` is used.

If a FileSystemPathDD directory is specified for a given data node (including the case where the parameter is specified in the [ndbd default] section of the config.ini file), then starting that data node with --initial causes all files in the directory to be deleted.

- FileSystemPathDataFiles

Table 22.170 This table provides type and value information for the FileSystemPathDataFiles data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	filename
Default	[see text]
Range	...
Restart Type	IN

If this parameter is specified, then NDB Cluster Disk Data data files are placed in the indicated directory. This overrides any value set for FileSystemPathDD. This parameter can be overridden for a given data file by specifying a path in the ADD DATAFILE clause of a CREATE TABLESPACE or ALTER TABLESPACE statement used to create that data file. If FileSystemPathDataFiles is not specified, then FileSystemPathDD is used (or FileSystemPath, if FileSystemPathDD has also not been set).

If a FileSystemPathDataFiles directory is specified for a given data node (including the case where the parameter is specified in the [ndbd default] section of the config.ini file), then starting that data node with --initial causes all files in the directory to be deleted.

- FileSystemPathUndoFiles

Table 22.171 This table provides type and value information for the FileSystemPathUndoFiles data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	filename
Default	[see text]
Range	...
Restart Type	IN

If this parameter is specified, then NDB Cluster Disk Data undo log files are placed in the indicated directory. This overrides any value set for FileSystemPathDD. This parameter can be overridden for a given data file by specifying a path in the ADD UNDO clause of a CREATE LOGFILE GROUP or ALTER LOGFILE GROUP statement used to create that data file. If FileSystemPathUndoFiles is not specified, then FileSystemPathDD is used (or FileSystemPath, if FileSystemPathDD has also not been set).

If a FileSystemPathUndoFiles directory is specified for a given data node (including the case where the parameter is specified in the [ndbd default] section of the config.ini file), then starting that data node with --initial causes all files in the directory to be deleted.

For more information, see Section 22.5.13.1, “NDB Cluster Disk Data Objects”.

- **Disk Data object creation parameters.** The next two parameters enable you—when starting the cluster for the first time—to cause a Disk Data log file group, tablespace, or both, to be created without the use of SQL statements.

  - `InitialLogFileGroup`

Table 22.172 This table provides type and value information for the InitialLogFileGroup data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	string
Default	[see text]
Range	...
Restart Type	S

This parameter can be used to specify a log file group that is created when performing an initial start of the cluster. `InitialLogFileGroup` is specified as shown here:

```
InitialLogFileGroup = [name=name;] [undo_buffer_size=size;] file-specification-list

file-specification-list:
    file-specification[; file-specification[; ...]]

file-specification:
    filename:size
```

The `name` of the log file group is optional and defaults to `DEFAULT-LG`. The `undo_buffer_size` is also optional; if omitted, it defaults to `64M`. Each *`file-specification`* corresponds to an undo log file, and at least one must be specified in the *`file-specification-list`*. Undo log files are placed according to any values that have been set for `FileSystemPath`, `FileSystemPathDD`, and `FileSystemPathUndoFiles`, just as if they had been created as the result of a `CREATE LOGFILE GROUP` or `ALTER LOGFILE GROUP` statement.

Consider the following:

```
InitialLogFileGroup = name=LG1; undo_buffer_size=128M; undo1.log:250M; undo2.log:150M
```

This is equivalent to the following SQL statements:

```
CREATE LOGFILE GROUP LG1
    ADD UNDOFILE 'undo1.log'
    INITIAL_SIZE 250M
    UNDO_BUFFER_SIZE 128M
    ENGINE NDBCLUSTER;

ALTER LOGFILE GROUP LG1
    ADD UNDOFILE 'undo2.log'
    INITIAL_SIZE 150M
    ENGINE NDBCLUSTER;
```

This logfile group is created when the data nodes are started with `--initial`.

Resources for the initial log file group are added to the global memory pool along with those indicated by the value of `SharedGlobalMemory`.

This parameter, if used, should always be set in the `[ndbd default]` section of the `config.ini` file. The behavior of an NDB Cluster when different values are set on different data nodes is not defined.

- `InitialTablespace`

Table 22.173 This table provides type and value information for the InitialTablespace data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	string
Default	[see text]
Range	...
Restart Type	S

This parameter can be used to specify an NDB Cluster Disk Data tablespace that is created when performing an initial start of the cluster. `InitialTablespace` is specified as shown here:

```
InitialTablespace = [name=name;] [extent_size=size;] file-specification-list
```

The `name` of the tablespace is optional and defaults to `DEFAULT-TS`. The `extent_size` is also optional; it defaults to `1M`. The *`file-specification-list`* uses the same syntax as shown with the `InitialLogfileGroup` parameter, the only difference being that each *`file-specification`* used with `InitialTablespace` corresponds to a data file. At least one must be specified in the *`file-specification-list`*. Data files are placed according to any values that have been set for `FileSystemPath`, `FileSystemPathDD`, and `FileSystemPathDataFiles`, just as if they had been created as the result of a `CREATE TABLESPACE` or `ALTER TABLESPACE` statement.

For example, consider the following line specifying `InitialTablespace` in the `[ndbd default]` section of the `config.ini` file (as with `InitialLogfileGroup`, this parameter should always be set in the `[ndbd default]` section, as the behavior of an NDB Cluster when different values are set on different data nodes is not defined):

```
InitialTablespace = name=TS1; extent_size=8M; data1.dat:2G; data2.dat:4G
```

This is equivalent to the following SQL statements:

```
CREATE TABLESPACE TS1
    ADD DATAFILE 'data1.dat'
    EXTENT_SIZE 8M
    INITIAL_SIZE 2G
    ENGINE NDBCLUSTER;

ALTER TABLESPACE TS1
    ADD DATAFILE 'data2.dat'
    INITIAL_SIZE 4G
    ENGINE NDBCLUSTER;
```

This tablespace is created when the data nodes are started with `--initial`, and can be used whenever creating NDB Cluster Disk Data tables thereafter.

Disk Data and GCP Stop errors. Errors encountered when using Disk Data tables such as `Node nodeid killed this node because GCP stop was detected` (error 2303) are often referred to as “GCP stop errors”. Such errors occur when the redo log is not flushed to disk quickly enough; this is usually due to slow disks and insufficient disk throughput.

You can help prevent these errors from occurring by using faster disks, and by placing Disk Data files on a separate disk from the data node file system. Reducing the value of `TimeBetweenGlobalCheckpoints` tends to decrease the amount of data to be written for each global checkpoint, and so may provide some protection against redo log buffer overflows when trying to write a global checkpoint; however, reducing this value also permits less time in which to write the GCP, so this must be done with caution.

In addition to the considerations given for `DiskPageBufferMemory` as explained previously, it is also very important that the `DiskIOThreadPool` configuration parameter be set correctly; having `DiskIOThreadPool` set too high is very likely to cause GCP stop errors (Bug #37227).

GCP stops can be caused by save or commit timeouts; the `TimeBetweenEpochsTimeout` data node configuration parameter determines the timeout for commits. However, it is possible to disable both types of timeouts by setting this parameter to 0.

Parameters for configuring send buffer memory allocation. Send buffer memory is allocated dynamically from a memory pool shared between all transporters, which means that the size of the send buffer can be adjusted as necessary. (Previously, the NDB kernel used a fixed-size send buffer for every node in the cluster, which was allocated when the node started and could not be changed while the node was running.) The `TotalSendBufferMemory` and `OverLoadLimit` data node configuration parameters permit the setting of limits on this memory allocation. For more information about the use of these parameters (as well as `SendBufferMemory`), see Section 22.3.3.13, “Configuring NDB Cluster Send Buffer Parameters”.

- `ExtraSendBufferMemory`

  This parameter specifies the amount of transporter send buffer memory to allocate in addition to any set using `TotalSendBufferMemory`, `SendBufferMemory`, or both.

- `TotalSendBufferMemory`

  This parameter is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

  If this parameter is set, its minimum permitted value is 256KB; 0 indicates that the parameter has not been set. For more detailed information, see Section 22.3.3.13, “Configuring NDB Cluster Send Buffer Parameters”.

See also Section 22.5.15, “Adding NDB Cluster Data Nodes Online”.

Redo log over-commit handling. It is possible to control a data node's handling of operations when too much time is taken flushing redo logs to disk. This occurs when a given redo log flush takes longer than `RedoOverCommitLimit` seconds, more than `RedoOverCommitCounter` times, causing any pending transactions to be aborted. When this happens, the API node that sent the transaction can handle the operations that should have been committed either by queuing the operations and re-trying them, or by aborting them, as determined by `DefaultOperationRedoProblemAction`. The data node configuration parameters for setting the timeout and number of times it may be exceeded before the API node takes this action are described in the following list:

- `RedoOverCommitCounter`

Table 22.174 This table provides type and value information for the RedoOverCommitCounter data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	3
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When `RedoOverCommitLimit` is exceeded when trying to write a given redo log to disk this many times or more, any transactions that were not committed as a result are aborted, and an API node where any of these transactions originated handles the operations making up those transactions according to its value for `DefaultOperationRedoProblemAction` (by either queuing the operations to be re-tried, or aborting them).

`RedoOverCommitCounter` defaults to 3. Set it to 0 to disable the limit.

- `RedoOverCommitLimit`

Table 22.175 This table provides type and value information for the RedoOverCommitLimit data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	seconds
Default	20
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets an upper limit in seconds for trying to write a given redo log to disk before timing out. The number of times the data node tries to flush this redo log, but takes longer than `RedoOverCommitLimit`, is kept and compared with `RedoOverCommitCounter`, and when flushing takes too long more times than the value of that parameter, any transactions that were not committed as a result of the flush timeout are aborted. When this occurs, the API node where any of these transactions originated handles the operations making up those transactions according to its `DefaultOperationRedoProblemAction` setting (it either queues the operations to be re-tried, or aborts them).

By default, `RedoOverCommitLimit` is 20 seconds. Set to 0 to disable checking for redo log flush timeouts.

Controlling restart attempts. It is possible to exercise finely-grained control over restart attempts by data nodes when they fail to start using the `MaxStartFailRetries` and `StartFailRetryDelay` data node configuration parameters.

`MaxStartFailRetries` limits the total number of retries made before giving up on starting the data node, `StartFailRetryDelay` sets the number of seconds between retry attempts. These parameters are listed here:

- `StartFailRetryDelay`

Table 22.176 This table provides type and value information for the StartFailRetryDelay data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Use this parameter to set the number of seconds between restart attempts by the data node in the event on failure on startup. The default is 0 (no delay).

Both this parameter and `MaxStartFailRetries` are ignored unless `StopOnError` is equal to 0.

- `MaxStartFailRetries`

Table 22.177 This table provides type and value information for the MaxStartFailRetries data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	3
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Use this parameter to limit the number restart attempts made by the data node in the event that it fails on startup. The default is 3 attempts.

Both this parameter and `StartFailRetryDelay` are ignored unless `StopOnError` is equal to 0.

NDB index statistics parameters. The parameters in the following list relate to NDB index statistics generation.

- `IndexStatAutoCreate`

Table 22.178 This table provides type and value information for the IndexStatAutoCreate data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0, 1
Restart Type	S

Enable (set equal to 1) or disable (set equal to 0) automatic statistics collection when indexes are created. Disabled by default.

- IndexStatAutoUpdate

Table 22.179 This table provides type and value information for the IndexStatAutoUpdate data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0, 1
Restart Type	S

Enable (set equal to 1) or disable (set equal to 0) monitoring of indexes for changes and trigger automatic statistics updates these are detected. The amount and degree of change needed to trigger the updates are determined by the settings for the IndexStatTriggerPct and IndexStatTriggerScale options.

- IndexStatSaveSize

Table 22.180 This table provides type and value information for the IndexStatSaveSize data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	32768
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

Maximum space in bytes allowed for the saved statistics of any given index in the NDB system tables and in the mysqld memory cache.

At least one sample is always produced, regardless of any size limit. This size is scaled by IndexStatSaveScale.

The size specified by IndexStatSaveSize is scaled by the value of IndexStatTriggerPct for a large index, times 0.01. This is further multiplied by the logarithm to the base 2 of the index size. Setting IndexStatTriggerPct equal to 0 disables the scaling effect.

- IndexStatSaveScale

Table 22.181 This table provides type and value information for the IndexStatSaveScale data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	percentage
Default	100
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

The size specified by `IndexStatSaveSize` is scaled by the value of `IndexStatTriggerPct` for a large index, times 0.01. This is further multiplied by the logarithm to the base 2 of the index size. Setting `IndexStatTriggerPct` equal to 0 disables the scaling effect.

- `IndexStatTriggerPct`

Table 22.182 This table provides type and value information for the IndexStatTriggerPct data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	percentage
Default	100
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

Percentage change in updates that triggers an index statistics update. The value is scaled by `IndexStatTriggerScale`. You can disable this trigger altogether by setting `IndexStatTriggerPct` to 0.

- `IndexStatTriggerScale`

Table 22.183 This table provides type and value information for the IndexStatTriggerScale data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	percentage
Default	100
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

Scale `IndexStatTriggerPct` by this amount times 0.01 for a large index. A value of 0 disables scaling.

- `IndexStatUpdateDelay`

Table 22.184 This table provides type and value information for the IndexStatUpdateDelay data node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	seconds
Default	60
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

Minimum delay in seconds between automatic index statistics updates for a given index. Setting this variable to 0 disables any delay. The default is 60 seconds.

22.3.3.7 Defining SQL and Other API Nodes in an NDB Cluster

The `[mysqld]` and `[api]` sections in the `config.ini` file define the behavior of the MySQL servers (SQL nodes) and other applications (API nodes) used to access cluster data. None of the parameters shown is required. If no computer or host name is provided, any host can use this SQL or API node.

Generally speaking, a `[mysqld]` section is used to indicate a MySQL server providing an SQL interface to the cluster, and an `[api]` section is used for applications other than `mysqld` processes accessing cluster data, but the two designations are actually synonymous; you can, for instance, list parameters for a MySQL server acting as an SQL node in an `[api]` section.

Note

For a discussion of MySQL server options for NDB Cluster, see MySQL Server Options for NDB Cluster. For information about MySQL server system variables relating to NDB Cluster, see NDB Cluster System Variables.

- `Id`

Table 22.185 This table provides type and value information for the Id API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	[none]
Range	1 - 255
Restart Type	IS

The `Id` is an integer value used to identify the node in all cluster internal messages. The permitted range of values is 1 to 255 inclusive. This value must be unique for each node in the cluster, regardless of the type of node.

Note

In NDB 8.0.18 and later, data node IDs must be less than 145. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for API nodes (and management nodes) to values greater than 144. (Previous to NDB 8.0.18, the maximum supported value for a data node ID was 48.)

`NodeId` is the preferred parameter name to use when identifying API nodes. (`Id` continues to be supported for backward compatibility, but is now deprecated and generates a warning when used. It is also subject to future removal.)

- `ConnectionMap`

Table 22.186 This table provides type and value information for the ConnectionMap API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	string
Default	[none]

Property	Value
Range	...
Restart Type	N

Specifies which data nodes to connect.

- `NodeId`

Table 22.187 This table provides type and value information for the NodeId API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	[none]
Range	1 - 255
Restart Type	IS

The `NodeId` is an integer value used to identify the node in all cluster internal messages. The permitted range of values is 1 to 255 inclusive. This value must be unique for each node in the cluster, regardless of the type of node.

Note

In NDB 8.0.18 and later, data node IDs must be less than 145. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for API nodes (and management nodes) to values greater than 144. (Previous to NDB 8.0.18, the maximum supported value for a data node ID was 48.)

`NodeId` is the preferred parameter name to use when identifying management nodes. An alias, `Id`, was used for this purpose in very old versions of NDB Cluster, and continues to be supported for backward compatibility; it is now deprecated and generates a warning when used, and is subject to removal in a future release of NDB Cluster.

- `ExecuteOnComputer`

Table 22.188 This table provides type and value information for the ExecuteOnComputer API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name
Default	[none]
Range	...
Restart Type	S

This refers to the `Id` set for one of the computers (hosts) defined in a `[computer]` section of the configuration file.

Important

This parameter is deprecated, and is subject to removal in a future release. Use the `HostName` parameter instead.

- `HostName`

Table 22.189 This table provides type and value information for the HostName API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

Specifying this parameter defines the hostname of the computer on which the SQL node (API node) is to reside. To specify a hostname, either this parameter or `ExecuteOnComputer` is required.

If no `HostName` or `ExecuteOnComputer` is specified in a given `[mysql]` or `[api]` section of the `config.ini` file, then an SQL or API node may connect using the corresponding "slot" from any host which can establish a network connection to the management server host machine. *This differs from the default behavior for data nodes, where `localhost` is assumed for `HostName` unless otherwise specified*.

- `LocationDomainId`

Table 22.190 This table provides type and value information for the LocationDomainId API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 16
Restart Type	S

Assigns an SQL or other API node to a specific availability domain (also known as an availability zone) within a cloud. By informing `NDB` which nodes are in which availability domains, performance can be improved in a cloud environment in the following ways:

- If requested data is not found on the same node, reads can be directed to another node in the same availability domain.
- Communication between nodes in different availability domains are guaranteed to use `NDB` transporters' WAN support without any further manual intervention.
- The transporter's group number can be based on which availability domain is used, such that also SQL and other API nodes communicate with local data nodes in the same availability domain whenever possible.

- The arbitrator can be selected from an availability domain in which no data nodes are present, or, if no such availability domain can be found, from a third availability domain.

`LocationDomainId` takes an integer value between 0 and 16 inclusive, with 0 being the default; using 0 is the same as leaving the parameter unset.

- `ArbitrationRank`

Table 22.191 This table provides type and value information for the ArbitrationRank API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	0-2
Default	0
Range	0 - 2
Restart Type	N

This parameter defines which nodes can act as arbitrators. Both management nodes and SQL nodes can be arbitrators. A value of 0 means that the given node is never used as an arbitrator, a value of 1 gives the node high priority as an arbitrator, and a value of 2 gives it low priority. A normal configuration uses the management server as arbitrator, setting its `ArbitrationRank` to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

By setting `ArbitrationRank` to 0 on all management and SQL nodes, you can disable arbitration completely. You can also control arbitration by overriding this parameter; to do so, set the `Arbitration` parameter in the `[ndbd default]` section of the `config.ini` global configuration file.

- `ArbitrationDelay`

Table 22.192 This table provides type and value information for the ArbitrationDelay API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	milliseconds
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Setting this parameter to any other value than 0 (the default) means that responses by the arbitrator to arbitration requests will be delayed by the stated number of milliseconds. It is usually not necessary to change this value.

- `BatchByteSize`

Table 22.193 This table provides type and value information for the BatchByteSize API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13

Property	Value
Type or units	bytes
Default	16K
Range	1K - 1M
Restart Type	N

For queries that are translated into full table scans or range scans on indexes, it is important for best performance to fetch records in properly sized batches. It is possible to set the proper size both in terms of number of records (`BatchSize`) and in terms of bytes (`BatchByteSize`). The actual batch size is limited by both parameters.

The speed at which queries are performed can vary by more than 40% depending upon how this parameter is set.

This parameter is measured in bytes. The default value is 16K.

- `BatchSize`

Table 22.194 This table provides type and value information for the BatchSize API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	records
Default	256
Range	1 - 992
Restart Type	N

This parameter is measured in number of records and is by default set to 256. The maximum size is 992.

- `ExtraSendBufferMemory`

Table 22.195 This table provides type and value information for the ExtraSendBufferMemory API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter specifies the amount of transporter send buffer memory to allocate in addition to any that has been set using `TotalSendBufferMemory`, `SendBufferMemory`, or both.

- `HeartbeatThreadPriority`

Table 22.196 This table provides type and value information for the HeartbeatThreadPriority API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	string
Default	[none]
Range	...
Restart Type	S

Use this parameter to set the scheduling policy and priority of heartbeat threads for management and API nodes. The syntax for setting this parameter is shown here:

```
HeartbeatThreadPriority = policy[, priority]

policy:
  {FIFO | RR}
```

When setting this parameter, you must specify a policy. This is one of `FIFO` (first in, first in) or `RR` (round robin). This followed optionally by the priority (an integer).

- `MaxScanBatchSize`

Table 22.197 This table provides type and value information for the MaxScanBatchSize API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	256K
Range	32K - 16M
Restart Type	N

The batch size is the size of each batch sent from each data node. Most scans are performed in parallel to protect the MySQL Server from receiving too much data from many nodes in parallel; this parameter sets a limit to the total batch size over all nodes.

The default value of this parameter is set to 256KB. Its maximum size is 16MB.

- `TotalSendBufferMemory`

Table 22.198 This table provides type and value information for the TotalSendBufferMemory API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	0
Range	256K - 4294967039 (0xFFFFFEFF)

Property	Value
Restart Type	N

This parameter is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

If this parameter is set, its minimum permitted value is 256KB; 0 indicates that the parameter has not been set. For more detailed information, see Section 22.3.3.13, “Configuring NDB Cluster Send Buffer Parameters”.

- `AutoReconnect`

Table 22.199 This table provides type and value information for the AutoReconnect API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

This parameter is `false` by default. This forces disconnected API nodes (including MySQL Servers acting as SQL nodes) to use a new connection to the cluster rather than attempting to re-use an existing one, as re-use of connections can cause problems when using dynamically-allocated node IDs. (Bug #45921)

Note

This parameter can be overridden using the NDB API. For more information, see Ndb_cluster_connection::set_auto_reconnect(), and Ndb_cluster_connection::get_auto_reconnect().

- `DefaultOperationRedoProblemAction`

Table 22.200 This table provides type and value information for the DefaultOperationRedoProblemAction API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	enumeration
Default	QUEUE
Range	ABORT, QUEUE
Restart Type	S

This parameter (along with `RedoOverCommitLimit` and `RedoOverCommitCounter`) controls the data node's handling of operations when too much time is taken flushing redo logs to disk. This occurs when a given redo log flush takes longer than `RedoOverCommitLimit` seconds, more than `RedoOverCommitCounter` times, causing any pending transactions to be aborted.

When this happens, the node can respond in either of two ways, according to the value of `DefaultOperationRedoProblemAction`, listed here:

- `ABORT`: Any pending operations from aborted transactions are also aborted.
- `QUEUE`: Pending operations from transactions that were aborted are queued up to be re-tried. This the default. Pending operations are still aborted when the redo log runs out of space—that is, when **`P_TAIL_PROBLEM`** errors occur.

- `DefaultHashMapSize`

Table 22.201 This table provides type and value information for the DefaultHashMapSize API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	buckets
Default	3840
Range	0 - 3840
Restart Type	N

The size of the table hash maps used by `NDB` is configurable using this parameter. `DefaultHashMapSize` can take any of three possible values (0, 240, 3840). These values and their effects are described in the following table.

Table 22.202 DefaultHashMapSize parameter values

Value	Description / Effect
`0`	Use the lowest value set, if any, for this parameter among all data nodes and API nodes in the cluster; if it is not set on any data or API node, use the default value.
`240`	Old default hash map size
`3840`	Hash map size used by default in NDB 8.0

The original intended use for this parameter was to facilitate upgrades and downgrades to and from older NDB Cluster versions, in which the hash map size differed, due to the fact that this change was not otherwise backward compatible. This is not an issue when upgrading to or downgrading from NDB Cluster 8.0.

- `Wan`

Table 22.203 This table provides type and value information for the wan API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Use WAN TCP setting as default.

- `ConnectBackoffMaxTime`

Table 22.204 This table provides type and value information for the ConnectBackoffMaxTime API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

In an NDB Cluster with many unstarted data nodes, the value of this parameter can be raised to circumvent connection attempts to data nodes which have not yet begun to function in the cluster, as well as moderate high traffic to management nodes. As long as the API node is not connected to any new data nodes, the value of the `StartConnectBackoffMaxTime` parameter is applied; otherwise, `ConnectBackoffMaxTime` is used to determine the length of time in milliseconds to wait between connection attempts.

Time elapsed *during* node connection attempts is not taken into account when calculating elapsed time for this parameter. The timeout is applied with approximately 100 ms resolution, starting with a 100 ms delay; for each subsequent attempt, the length of this period is doubled until it reaches `ConnectBackoffMaxTime` milliseconds, up to a maximum of 100000 ms (100s).

Once the API node is connected to a data node and that node reports (in a heartbeat message) that it has connected to other data nodes, connection attempts to those data nodes are no longer affected by this parameter, and are made every 100 ms thereafter until connected. Once a data node has started, it can take up `HeartbeatIntervalDbApi` for the API node to be notified that this has occurred.

- `StartConnectBackoffMaxTime`

Table 22.205 This table provides type and value information for the StartConnectBackoffMaxTime API node configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

In an NDB Cluster with many unstarted data nodes, the value of this parameter can be raised to circumvent connection attempts to data nodes which have not yet begun to function in the cluster, as well as moderate high traffic to management nodes. As long as the API node is not connected to any new data nodes, the value of the `StartConnectBackoffMaxTime` parameter is applied; otherwise, `ConnectBackoffMaxTime` is used to determine the length of time in milliseconds to wait between connection attempts.

Time elapsed *during* node connection attempts is not taken into account when calculating elapsed time for this parameter. The timeout is applied with approximately 100 ms resolution, starting with a 100 ms delay; for each subsequent attempt, the length of this period is doubled until it reaches `StartConnectBackoffMaxTime` milliseconds, up to a maximum of 100000 ms (100s).

Once the API node is connected to a data node and that node reports (in a heartbeat message) that it has connected to other data nodes, connection attempts to those data nodes are no longer affected by this parameter, and are made every 100 ms thereafter until connected. Once a data node has started, it can take up `HeartbeatIntervalDbApi` for the API node to be notified that this has occurred.

API Node Debugging Parameters. You can use the `ApiVerbose` configuration parameter to enable debugging output from a given API node. This parameter takes an integer value. 0 is the default, and disables such debugging; 1 enables debugging output to the cluster log; 2 adds `DBDICT` debugging output as well. (Bug #20638450) See also DUMP 1229.

You can also obtain information from a MySQL server running as an NDB Cluster SQL node using `SHOW STATUS` in the `mysql` client, as shown here:

```
mysql> SHOW STATUS LIKE 'ndb%';
+-----------------------------+----------------+
| Variable_name               | Value          |
+-----------------------------+----------------+
| Ndb_cluster_node_id         | 5              |
| Ndb_config_from_host        | 198.51.100.112 |
| Ndb_config_from_port        | 1186           |
| Ndb_number_of_storage_nodes | 4              |
+-----------------------------+----------------+
4 rows in set (0.02 sec)
```

For information about the status variables appearing in the output from this statement, see NDB Cluster Status Variables.

Note

To add new SQL or API nodes to the configuration of a running NDB Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new `[mysqld]` or `[api]` sections to the `config.ini` file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is *not* necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

22.3.3.8 Defining the System

The `[system]` section is used for parameters applying to the cluster as a whole. The `Name` system parameter is used with MySQL Enterprise Monitor; `ConfigGenerationNumber` and `PrimaryMGMNode` are not used in production environments. Except when using NDB Cluster with MySQL Enterprise Monitor, is not necessary to have a `[system]` section in the `config.ini` file.

More information about these parameters can be found in the following list:

- `ConfigGenerationNumber`

Table 22.206 This table provides type and value information for the ConfigGenerationNumber system configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Configuration generation number. This parameter is currently unused.

- `Name`

Table 22.207 This table provides type and value information for the Name system configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	string
Default	[none]
Range	...
Restart Type	N

Set a name for the cluster. This parameter is required for deployments with MySQL Enterprise Monitor; it is otherwise unused.

You can obtain the value of this parameter by checking the `Ndb_system_name` status variable. In NDB API applications, you can also retrieve it using `get_system_name()`.

- `PrimaryMGMNode`

Table 22.208 This table provides type and value information for the PrimaryMGMNode system configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Node ID of the primary management node. This parameter is currently unused.

22.3.3.9 MySQL Server Options and Variables for NDB Cluster

This section provides information about MySQL server options, server and status variables that are specific to NDB Cluster. For general information on using these, and for other options and variables not specific to NDB Cluster, see Section 5.1, “The MySQL Server”.

For NDB Cluster configuration parameters used in the cluster configuration file (usually named `config.ini`), see Section 22.3, “Configuration of NDB Cluster”.

MySQL Server Options for NDB Cluster

This section provides descriptions of `mysqld` server options relating to NDB Cluster. For information about `mysqld` options not specific to NDB Cluster, and for general information about the use of options with `mysqld`, see Section 5.1.7, “Server Command Options”.

For information about command-line options used with other NDB Cluster processes (`ndbd`, `ndb_mgmd`, and `ndb_mgm`), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”. For information about command-line options used with `NDB` utility programs (such as `ndb_desc`, `ndb_size.pl`, and `ndb_show_tables`), see Section 22.4, “NDB Cluster Programs”.

- `--ndbcluster`

Table 22.209 Type and value information for ndbcluster

Property	Value
Name	`ndbcluster`
Command Line	Yes
System Variable	No
Status Variable	No
Option File	Yes
Scope	
Dynamic	No
Type	
Default, Range	FALSE (Version: 8.0)
Notes	DESCRIPTION: Enable NDB Cluster (if this version of MySQL supports it) Disabled by `--skip-ndbcluster`.

The `NDBCLUSTER` storage engine is necessary for using NDB Cluster. If a `mysqld` binary includes support for the `NDBCLUSTER` storage engine, the engine is disabled by default. Use the `--ndbcluster` option to enable it. Use `--skip-ndbcluster` to explicitly disable the engine.

The `--ndbcluster` option is ignored (and the `NDB` storage engine is *not* enabled) if `--initialize` is also used. (It is neither necessary nor desirable to use this option together with `--initialize`.)

- `--ndb-allow-copying-alter-table=[ON|OFF]`

Table 22.210 Type and value information for ndb-allow-copying-alter-table

Property	Value
Name	`ndb-allow-copying-alter-table`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes

Property	Value
Scope	Both
Dynamic	Yes
Type	
Default, Range	ON (Version: 8.0)
Notes	DESCRIPTION: Set to OFF to keep ALTER TABLE from using copying operations on NDB tables

Let `ALTER TABLE` and other DDL statements use copying operations on `NDB` tables. Set to `OFF` to keep this from happening; doing so may improve performance of critical applications.

- `--ndb-batch-size=#`

Table 22.211 Type and value information for ndb-batch-size

Property	Value
Name	`ndb-batch-size`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	32768 / 0 - 31536000 (Version: 8.0)
Notes	DESCRIPTION: Size (in bytes) to use for NDB transaction batches

This sets the size in bytes that is used for NDB transaction batches.

- `--ndb-cluster-connection-pool=#`

Table 22.212 Type and value information for ndb-cluster-connection-pool

Property	Value
Name	`ndb-cluster-connection-pool`
Command Line	Yes
System Variable	Yes
Status Variable	Yes
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	1 / 1 - 63 (Version: 8.0)
Notes	DESCRIPTION: Number of connections to the cluster used by MySQL

By setting this option to a value greater than 1 (the default), a `mysqld` process can use multiple connections to the cluster, effectively mimicking several SQL nodes. Each connection requires its own `[api]` or `[mysqld]` section in the cluster configuration (`config.ini`) file, and counts against the maximum number of API connections supported by the cluster.

Suppose that you have 2 cluster host computers, each running an SQL node whose `mysqld` process was started with `--ndb-cluster-connection-pool=4`; this means that the cluster must have 8 API slots available for these connections (instead of 2). All of these connections are set up when the SQL node connects to the cluster, and are allocated to threads in a round-robin fashion.

This option is useful only when running `mysqld` on host machines having multiple CPUs, multiple cores, or both. For best results, the value should be smaller than the total number of cores available on the host machine. Setting it to a value greater than this is likely to degrade performance severely.

Important

Because each SQL node using connection pooling occupies multiple API node slots—each slot having its own node ID in the cluster—you must *not* use a node ID as part of the cluster connection string when starting any `mysqld` process that employs connection pooling.

Setting a node ID in the connection string when using the `--ndb-cluster-connection-pool` option causes node ID allocation errors when the SQL node attempts to connect to the cluster.

- `--ndb-cluster-connection-pool-nodeids=`*`list`*

Table 22.213 Type and value information for ndb-cluster-connection-pool-nodeids

Property	Value
Name	`ndb-cluster-connection-pool-nodeids`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	/ (Version: 8.0)
Notes	DESCRIPTION: Comma-separated list of node IDs for connections to the cluster used by MySQL; the number of nodes in the list must be the same as the value set for --ndb-cluster-connection-pool

Specifies a comma-separated list of node IDs for connections to the cluster used by an SQL node. The number of nodes in this list must be the same as the value set for the `--ndb-cluster-connection-pool` option.

- `--ndb-blob-read-batch-bytes=`*`bytes`*

Table 22.214 Type and value information for ndb-blob-read-batch-bytes

Property	Value
Name	ndb-blob-read-batch-bytes
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	65536 / 0 - 4294967295 (Version: NDB 8.0)
Notes	DESCRIPTION: Specifies size in bytes that large BLOB reads should be batched into. 0 = no limit.

This option can be used to set the size (in bytes) for batching of BLOB data reads in NDB Cluster applications. When this batch size is exceeded by the amount of BLOB data to be read within the current transaction, any pending BLOB read operations are immediately executed.

The maximum value for this option is 4294967295; the default is 65536. Setting it to 0 has the effect of disabling BLOB read batching.

Note

In NDB API applications, you can control BLOB write batching with the setMaxPendingBlobReadBytes() and getMaxPendingBlobReadBytes() methods.

- --ndb-blob-write-batch-bytes=*bytes*

Table 22.215 Type and value information for ndb-blob-write-batch-bytes

Property	Value
Name	ndb-blob-write-batch-bytes
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	65536 / 0 - 4294967295 (Version: 8.0)
Notes	DESCRIPTION: Specifies size in bytes that large BLOB writes should be batched into. 0 = no limit.

This option can be used to set the size (in bytes) for batching of `BLOB` data writes in NDB Cluster applications. When this batch size is exceeded by the amount of `BLOB` data to be written within the current transaction, any pending `BLOB` write operations are immediately executed.

The maximum value for this option is 4294967295; the default is 65536. Setting it to 0 has the effect of disabling `BLOB` write batching.

Note

In NDB API applications, you can control `BLOB` write batching with the `setMaxPendingBlobWriteBytes()` and `getMaxPendingBlobWriteBytes()` methods.

- `--ndb-connectstring=`*`connection_string`*

Table 22.216 Type and value information for ndb-connectstring

Property	Value
Name	`ndb-connectstring`
Command Line	Yes
System Variable	No
Status Variable	No
Option File	Yes
Scope	
Dynamic	No
Type	
Default, Range	(Version: 8.0)
Notes	DESCRIPTION: Point to the management server that distributes the cluster configuration

When using the `NDBCLUSTER` storage engine, this option specifies the management server that distributes cluster configuration data. See Section 22.3.3.3, “NDB Cluster Connection Strings”, for syntax.

- `--ndb-default-column-format=[FIXED|DYNAMIC]`

Table 22.217 Type and value information for ndb-default-column-format

Property	Value
Name	`ndb-default-column-format`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	

Property	Value
Default, Range	DYNAMIC / FIXED, DYNAMIC (Version: NDB 8.0)
Default, Range	FIXED / FIXED, DYNAMIC (Version: NDB 8.0)
Notes	DESCRIPTION: Use this value (FIXED or DYNAMIC) by default for COLUMN_FORMAT and ROW_FORMAT options when creating or adding columns to a table.

Sets the default `COLUMN_FORMAT` and `ROW_FORMAT` for new tables (see Section 13.1.20, “CREATE TABLE Syntax”). The default is `FIXED`.

- `--ndb-deferred-constraints=[0|1]`

Table 22.218 Type and value information for ndb-deferred-constraints

Property	Value
Name	`ndb-deferred-constraints`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	0 / 0 - 1 (Version: 8.0)
Notes	DESCRIPTION: Specifies that constraint checks on unique indexes (where these are supported) should be deferred until commit time. Not normally needed or used; for testing purposes only.

Controls whether or not constraint checks on unique indexes are deferred until commit time, where such checks are supported. `0` is the default.

This option is not normally needed for operation of NDB Cluster or NDB Cluster Replication, and is intended primarily for use in testing.

- `--ndb-schema-dist-timeout=#`

Table 22.219 Type and value information for ndb-schema-dist-timeout

Property	Value
Name	`ndb-schema-dist-timeout`
Command Line	Yes
System Variable	Yes
Option File	Yes
Scope	Global
Dynamic	No
Type: Default, Range	integer: 120 / 5 - 1200 (Version: 8.0.17)

Property	Value
Notes	DESCRIPTION: How long to wait before detecting timeout during schema distribution

Specifies the maximum time in seconds that this `mysqld` waits for a schema operation to complete before marking it as having timed out.

- `--ndb-distribution=[KEYHASH|LINHASH]`

Table 22.220 Type and value information for ndb-distribution

Property	Value
Name	`ndb-distribution`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	KEYHASH / LINHASH, KEYHASH (Version: 8.0)
Notes	DESCRIPTION: Default distribution for new tables in NDBCLUSTER (KEYHASH or LINHASH, default is KEYHASH)

Controls the default distribution method for `NDB` tables. Can be set to either of `KEYHASH` (key hashing) or `LINHASH` (linear hashing). `KEYHASH` is the default.

- `--ndb-log-apply-status`

Table 22.221 Type and value information for ndb-log-apply-status

Property	Value
Name	`ndb-log-apply-status`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: Cause a MySQL server acting as a slave to log mysql.ndb_apply_status updates received from its immediate master in its own binary log, using its own server ID. Effective only if the server is started with the --ndbcluster option.

Causes a slave `mysqld` to log any updates received from its immediate master to the `mysql.ndb_apply_status` table in its own binary log using its own server ID rather than the server ID of the master. In a circular or chain replication setting, this allows such updates to propagate to the `mysql.ndb_apply_status` tables of any MySQL servers configured as slaves of the current `mysqld`.

In a chain replication setup, using this option allows downstream (slave) clusters to be aware of their positions relative to all of their upstream contributors (masters).

In a circular replication setup, this option causes changes to `ndb_apply_status` tables to complete the entire circuit, eventually propagating back to the originating NDB Cluster. This also allows a cluster acting as a master to see when its changes (epochs) have been applied to the other clusters in the circle.

This option has no effect unless the MySQL server is started with the `--ndbcluster` option.

- `--ndb-log-empty-epochs=[ON|OFF]`

Table 22.222 Type and value information for ndb-log-empty-epochs

Property	Value
Name	`ndb-log-empty-epochs`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: When enabled, causes epochs in which there were no changes to be written to the ndb_apply_status and ndb_binlog_index tables, even when `log_slave_updates` is enabled.

Causes epochs during which there were no changes to be written to the `ndb_apply_status` and `ndb_binlog_index` tables, even when `log_slave_updates` is enabled.

By default this option is disabled. Disabling `--ndb-log-empty-epochs` causes epoch transactions with no changes not to be written to the binary log, although a row is still written even for an empty epoch in `ndb_binlog_index`.

Because `--ndb-log-empty-epochs=1` causes the size of the `ndb_binlog_index` table to increase independently of the size of the binary log, users should be prepared to manage the growth of this table, even if they expect the cluster to be idle a large part of the time.

- `--ndb-log-empty-update=[ON|OFF]`

Table 22.223 Type and value information for ndb-log-empty-update

Property	Value
Name	`ndb-log-empty-update`

Property	Value
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: When enabled, causes updates that produced no changes to be written to the ndb_apply_status and ndb_binlog_index tables, even when `log_slave_updates` is enabled.

Causes updates that produced no changes to be written to the `ndb_apply_status` and `ndb_binlog_index` tables, when when `log_slave_updates` is enabled.

By default this option is disabled (`OFF`). Disabling `--ndb-log-empty-update` causes updates with no changes not to be written to the binary log.

- `--ndb-log-exclusive-reads=[0|1]`

Table 22.224 Type and value information for ndb-log-exclusive-reads

Property	Value
Name	`ndb-log-exclusive-reads`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	0 (Version: 8.0)
Notes	DESCRIPTION: Log primary key reads with exclusive locks; allow conflict resolution based on read conflicts

Starting the server with this option causes primary key reads to be logged with exclusive locks, which allows for NDB Cluster Replication conflict detection and resolution based on read conflicts. You can also enable and disable these locks at runtime by setting the value of the `ndb_log_exclusive_reads` system variable to 1 or 0, respectively. 0 (disable locking) is the default.

For more information, see Read conflict detection and resolution.

- `--ndb-log-orig`

Table 22.225 Type and value information for ndb-log-orig

Property	Value
Name	`ndb-log-orig`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: Log originating server id and epoch in mysql.ndb_binlog_index table

Log the originating server ID and epoch in the `ndb_binlog_index` table.

Note

This makes it possible for a given epoch to have multiple rows in `ndb_binlog_index`, one for each originating epoch.

For more information, see Section 22.6.4, “NDB Cluster Replication Schema and Tables”.

- `--ndb-log-transaction-id`

Table 22.226 Type and value information for ndb-log-transaction-id

Property	Value
Name	`ndb-log-transaction-id`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: Write NDB transaction IDs in the binary log.

Causes a slave `mysqld` to write the NDB transaction ID in each row of the binary log. The default value is `FALSE`.

This option is not supported in mainline MySQL Server 8.0. It is required to enable NDB Cluster Replication conflict detection and resolution using the `NDB$EPOCH_TRANS()` function (see NDB $EPOCH_TRANS()). For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

The deprecated `log_bin_use_v1_row_events` system variable, which defaults to `OFF`, must not be set to `ON` when you use `--ndb-log-transaction-id`.

- `--ndb-log-update-minimal`

Table 22.227 Type and value information for ndb-log-update-minimal

Property	Value
Name	`ndb-log-update-minimal`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	OFF (Version: NDB 8.0)
Default, Range	OFF (Version: NDB 8.0)
Default, Range	OFF (Version: NDB 8.0)
Notes	DESCRIPTION: Log updates in a minimal format.

Log updates in a minimal fashion, by writing only the primary key values in the before image, and only the changed columns in the after image. This may cause compatibility problems if replicating to storage engines other than `NDB`.

- `--ndb-mgmd-host=host[:port]`

Table 22.228 Type and value information for ndb-mgmd-host

Property	Value
Name	`ndb-mgmd-host`
Command Line	Yes
System Variable	No
Status Variable	No
Option File	Yes
Scope	
Dynamic	No
Type	
Default, Range	localhost:1186 (Version: 8.0)
Notes	DESCRIPTION: Set the host (and port, if desired) for connecting to management server

Can be used to set the host and port number of a single management server for the program to connect to. If the program requires node IDs or references to multiple management servers (or both) in its connection information, use the `--ndb-connectstring` option instead.

- `--ndb-nodeid=#`

Table 22.229 Type and value information for ndb-nodeid

Property	Value
Name	`ndb-nodeid`
Command Line	Yes
System Variable	No
Status Variable	Yes
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	/ 1 - 63 (Version: NDB 8.0)
Default, Range	/ 1 - 255 (Version: NDB 8.0)
Notes	DESCRIPTION: NDB Cluster node ID for this MySQL server

Set this MySQL server's node ID in an NDB Cluster.

The `--ndb-nodeid` option overrides any node ID set with `--ndb-connectstring`, regardless of the order in which the two options are used.

In addition, if `--ndb-nodeid` is used, then either a matching node ID must be found in a `[mysqld]` or `[api]` section of `config.ini`, or there must be an “open” `[mysqld]` or `[api]` section in the file (that is, a section without a `NodeId` or `Id` parameter specified). This is also true if the node ID is specified as part of the connection string.

Regardless of how the node ID is determined, its is shown as the value of the global status variable `Ndb_cluster_node_id` in the output of `SHOW STATUS`, and as `cluster_node_id` in the `connection` row of the output of `SHOW ENGINE NDBCLUSTER STATUS`.

For more information about node IDs for NDB Cluster SQL nodes, see Section 22.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

- `--ndbinfo={ON|OFF|FORCE}`

Table 22.230 Type and value information for ndbinfo

Property	Value
Name	`ndbinfo`
Command Line	Yes
System Variable	No
Option File	No
Scope	
Dynamic	No
Type: Default, Range	enumeration: ON, OFF, FORCE (Version: 8.0.13)
Notes	DESCRIPTION: Enable ndbinfo plugin, if supported

Enables the plugin for the `ndbinfo` information database. By default this is ON whenever `NDBCLUSTER` is enabled.

- `--ndb-optimization-delay=milliseconds`

Table 22.231 Type and value information for ndb-optimization-delay

Property	Value
Name	`ndb-optimization-delay`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	No
Scope	Global
Dynamic	Yes
Type	
Default, Range	10 / 0 - 100000 (Version: 8.0)
Notes	DESCRIPTION: Sets the number of milliseconds to wait between processing sets of rows by OPTIMIZE TABLE on NDB tables

Set the number of milliseconds to wait between sets of rows by `OPTIMIZE TABLE` statements on `NDB` tables. The default is 10.

- `--ndb-recv-thread-activation-threshold=threshold`

Table 22.232 Type and value information for ndb-recv-thread-activation-threshold

Property	Value
Name	`ndb-recv-thread-activation-threshold`
Command Line	Yes
System Variable	No
Status Variable	No
Option File	Yes
Scope	
Dynamic	No
Type	
Default, Range	8 / 0 (MIN_ACTIVATION_THRESHOLD) - 16 (MAX_ACTIVATION_THRESHOLD) (Version: NDB 8.0)
Notes	DESCRIPTION: Activation threshold when receive thread takes over the polling of the cluster connection (measured in concurrently active threads)

When this number of concurrently active threads is reached, the receive thread takes over polling of the cluster connection.

- `--ndb-recv-thread-cpu-mask=bitmask`

Table 22.233 Type and value information for ndb-recv-thread-cpu-mask

Property	Value
Name	ndb-recv-thread-cpu-mask
Command Line	Yes
System Variable	No
Status Variable	No
Option File	Yes
Scope	
Dynamic	No
Type	
Default, Range	[empty] (Version: 8.0)
Notes	DESCRIPTION: CPU mask for locking receiver threads to specific CPUs; specified as hexadecimal. See documentation for details.

Set a CPU mask for locking receiver threads to specific CPUs. This is specified as a hexadecimal bitmask; for example, `0x33` means that one CPU is used per receiver thread. An empty string (no locking of receiver threads) is the default.

- `ndb-transid-mysql-connection-map=`*`state`*

Table 22.234 Type and value information for ndb-transid-mysql-connection-map

Property	Value
Name	ndb-transid-mysql-connection-map
Command Line	Yes
System Variable	No
Status Variable	No
Option File	No
Scope	
Dynamic	No
Type	
Default, Range	ON / ON, OFF, FORCE (Version: 8.0)
Notes	DESCRIPTION: Enable or disable the ndb_transid_mysql_connection_map plugin; that is, enable or disable the INFORMATION_SCHEMA table having that name

Enables or disables the plugin that handles the `ndb_transid_mysql_connection_map` table in the `INFORMATION_SCHEMA` database. Takes one of the values `ON`, `OFF`, or `FORCE`. `ON` (the default) enables the plugin. `OFF` disables the plugin, which makes `ndb_transid_mysql_connection_map` inaccessible. `FORCE` keeps the MySQL Server from starting if the plugin fails to load and start.

You can see whether the `ndb_transid_mysql_connection_map` table plugin is running by checking the output of `SHOW PLUGINS`.

- `--ndb-wait-connected=`*`seconds`*

Table 22.235 Type and value information for ndb-wait-connected

Property	Value
Name	`ndb-wait-connected`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	0 / 0 - 31536000 (Version: 8.0)
Default, Range	30 / 0 - 31536000 (Version: NDB 8.0)
Default, Range	0 / 0 - 31536000 (Version: 8.0)
Default, Range	30 / 0 - 31536000 (Version: NDB 8.0)
Notes	DESCRIPTION: Time (in seconds) for the MySQL server to wait for connection to cluster management and data nodes before accepting MySQL client connections

This option sets the period of time that the MySQL server waits for connections to NDB Cluster management and data nodes to be established before accepting MySQL client connections. The time is specified in seconds. The default value is `30`.

- `--ndb-wait-setup=`*`seconds`*

Table 22.236 Type and value information for ndb-wait-setup

Property	Value
Name	`ndb-wait-setup`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	15 / 0 - 31536000 (Version: NDB 8.0)
Default, Range	15 / 0 - 31536000 (Version: NDB 8.0)
Default, Range	15 / 0 - 31536000 (Version: NDB 8.0)
Default, Range	30 / 0 - 31536000 (Version: NDB 8.0)
Default, Range	15 / 0 - 31536000 (Version: NDB 8.0)
Default, Range	30 / 0 - 31536000 (Version: NDB 8.0)

Property	Value
Notes	DESCRIPTION: Time (in seconds) for the MySQL server to wait for NDB engine setup to complete

This variable shows the period of time that the MySQL server waits for the `NDB` storage engine to complete setup before timing out and treating `NDB` as unavailable. The time is specified in seconds. The default value is `30`.

- `--server-id-bits=#`

Table 22.237 Type and value information for server-id-bits

Property	Value
Name	`server-id-bits`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	32 / 7 - 32 (Version: 8.0)
Notes	DESCRIPTION: Sets the number of least significant bits in the server_id actually used for identifying the server, permitting NDB API applications to store application data in the most significant bits. server_id must be less than 2 to the power of this value.

This option indicates the number of least significant bits within the 32-bit `server_id` which actually identify the server. Indicating that the server is actually identified by fewer than 32 bits makes it possible for some of the remaining bits to be used for other purposes, such as storing user data generated by applications using the NDB API's Event API within the `AnyValue` of an `OperationOptions` structure (NDB Cluster uses the `AnyValue` to store the server ID).

When extracting the effective server ID from `server_id` for purposes such as detection of replication loops, the server ignores the remaining bits. The `--server-id-bits` option is used to mask out any irrelevant bits of `server_id` in the IO and SQL threads when deciding whether an event should be ignored based on the server ID.

This data can be read from the binary log by `mysqlbinlog`, provided that it is run with its own `--server-id-bits` option set to 32 (the default).

The value of `server_id` must be less than 2 ^ `server_id_bits`; otherwise, `mysqld` refuses to start.

This system variable is supported only by NDB Cluster. It is not supported in the standard MySQL 8.0 Server.

- `--skip-ndbcluster`

Table 22.238 Type and value information for skip-ndbcluster

Property	Value
Name	skip-ndbcluster
Command Line	Yes
System Variable	No
Status Variable	No
Option File	Yes
Scope	
Dynamic	No
Type	
Notes	DESCRIPTION: Disable the NDB Cluster storage engine

Disable the NDBCLUSTER storage engine. This is the default for binaries that were built with NDBCLUSTER storage engine support; the server allocates memory and other resources for this storage engine only if the --ndbcluster option is given explicitly. See Section 22.3.1, “Quick Test Setup of NDB Cluster”, for an example.

NDB Cluster System Variables

This section provides detailed information about MySQL server system variables that are specific to NDB Cluster and the NDB storage engine. For system variables not specific to NDB Cluster, see Section 5.1.8, “Server System Variables”. For general information on using system variables, see Section 5.1.9, “Using System Variables”.

- ndb_autoincrement_prefetch_sz

Table 22.239 Type and value information for ndb_autoincrement_prefetch_sz

Property	Value
Name	ndb_autoincrement_prefetch_sz
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	32 / 1 - 256 (Version: 8.0)
Default, Range	1 / 1 - 256 (Version: NDB 8.0)
Default, Range	32 / 1 - 256 (Version: NDB 8.0)
Default, Range	1 / 1 - 256 (Version: NDB 8.0)
Default, Range	32 / 1 - 256 (Version: NDB 8.0)
Default, Range	1 / 1 - 256 (Version: NDB 8.0)
Default, Range	32 / 1 - 256 (Version: NDB 8.0)

Property	Value
Default, Range	1 / 1 - 256 (Version: NDB 8.0)
Default, Range	1 / 1 - 65536 (Version: NDB 8.0)
Default, Range	32 / 1 - 256 (Version: NDB 8.0)
Default, Range	1 / 1 - 65536 (Version: NDB 8.0)
Default, Range	1 / 1 - 65536 (Version: NDB 8.0)
Notes	DESCRIPTION: NDB auto-increment prefetch size

Determines the probability of gaps in an autoincremented column. Set it to `1` to minimize this. Setting it to a high value for optimization makes inserts faster, but decreases the likelihood that consecutive autoincrement numbers will be used in a batch of inserts. The mininum and default value is 1. The maximum value for `ndb_autoincrement_prefetch_sz` is 65536.

This variable affects only the number of `AUTO_INCREMENT` IDs that are fetched between statements; within a given statement, at least 32 IDs are obtained at a time. The default value is 1.

Important

This variable does not affect inserts performed using `INSERT ... SELECT`.

- `ndb_cache_check_time`

Table 22.240 Type and value information for ndb_cache_check_time

Property	Value
Name	`ndb_cache_check_time`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	0 / - (Version: 8.0)
Notes	DESCRIPTION: Number of milliseconds between checks of cluster SQL nodes made by the MySQL query cache

The number of milliseconds that elapse between checks of NDB Cluster SQL nodes by the MySQL query cache. Setting this to 0 (the default and minimum value) means that the query cache checks for validation on every query.

The recommended maximum value for this variable is 1000, which means that the check is performed once per second. A larger value means that the check is performed and possibly invalidated due to updates on different SQL nodes less often. It is generally not desirable to set this to a value greater than 2000.

Note

The query cache `ndb_cache_check_time` are deprecated in MySQL 5.7; the query cache was removed in MySQL 8.0.

- `ndb_clear_apply_status`

Table 22.241 Type and value information for ndb_clear_apply_status

Property	Value
Name	`ndb_clear_apply_status`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	No
Scope	Global
Dynamic	Yes
Type	
Default, Range	ON (Version: 8.0)
Notes	DESCRIPTION: Causes RESET SLAVE to clear all rows from the ndb_apply_status table; ON by default

By the default, executing `RESET SLAVE` causes an NDB Cluster replication slave to purge all rows from its `ndb_apply_status` table. You can disable this by setting `ndb_clear_apply_status=OFF`.

- `ndb_data_node_neighbour`

Table 22.242 Type and value information for ndb_data_node_neighbour

Property	Value
Name	`ndb_data_node_neighbour`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	0 / 0 - 255 (Version: NDB 8.0)
Notes	DESCRIPTION: Specifies cluster data node "closest" to this MySQL Server, for transaction hinting and fully replicated tables

Sets the ID of a “nearest” data node—that is, a preferred nonlocal data node is chosen to execute the transaction, rather than one running on the same host as the SQL or API node. This used to ensure that when a fully replicated table is accessed, we access it on this data node, to ensure that the local copy of the table is always used whenever possible. This can also be used for providing hints for transactions.

This can improve data access times in the case of a node that is physically closer than and thus has higher network throughput than others on the same host.

See Section 13.1.20.11, “Setting NDB_TABLE Options”, for further information.

Note

An equivalent method `set_data_node_neighbour()` is provided for use in NDB API applications.

- `ndb_dbg_check_shares`

Table 22.243 Type and value information for ndb_dbg_check_shares

Property	Value
Name	`ndb_dbg_check_shares`
Command Line	Yes
System Variable	Yes
Option File	Yes
Scope	Both
Dynamic	Yes
Type: Default, Range	integer: 0 / 0 - 1 (Version: 8.0)
Notes	DESCRIPTION: Check for any lingering shares (debug builds only)

When set to 1, check that no shares are lingering. Available in debug builds only.

Added in NDB 8.0.13.

- `ndb_default_column_format`

Table 22.244 Type and value information for ndb_default_column_format

Property	Value
Name	`ndb_default_column_format`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	DYNAMIC / FIXED, DYNAMIC (Version: NDB 8.0)
Default, Range	FIXED / FIXED, DYNAMIC (Version: NDB 8.0)
Notes	DESCRIPTION: Sets default row format and column format (FIXED or DYNAMIC) used for new NDB tables

Sets the default `COLUMN_FORMAT` and `ROW_FORMAT` for new tables (see Section 13.1.20, “CREATE TABLE Syntax”). The default is `FIXED`.

- ndb_deferred_constraints

Table 22.245 Type and value information for ndb_deferred_constraints

Property	Value
Name	ndb_deferred_constraints
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	0 / 0 - 1 (Version: 8.0)
Notes	DESCRIPTION: Specifies that constraint checks should be deferred (where these are supported). Not normally needed or used; for testing purposes only.

Controls whether or not constraint checks are deferred, where these are supported. 0 is the default.

This variable is not normally needed for operation of NDB Cluster or NDB Cluster Replication, and is intended primarily for use in testing.

- ndb_distribution

Table 22.246 Type and value information for ndb_distribution

Property	Value
Name	ndb_distribution
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	KEYHASH / LINHASH, KEYHASH (Version: 8.0)
Notes	DESCRIPTION: Default distribution for new tables in NDBCLUSTER (KEYHASH or LINHASH, default is KEYHASH)

Controls the default distribution method for NDB tables. Can be set to either of KEYHASH (key hashing) or LINHASH (linear hashing). KEYHASH is the default.

- ndb_eventbuffer_free_percent

Table 22.247 Type and value information for ndb_eventbuffer_free_percent

Property	Value
Name	ndb_eventbuffer_free_percent
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	20 / 1 - 99 (Version: 8.0)
Notes	DESCRIPTION: Percentage of free memory that should be available in event buffer before resumption of buffering, after reaching limit set by ndb_eventbuffer_max_alloc

Sets the percentage of the maximum memory allocated to the event buffer (ndb_eventbuffer_max_alloc) that should be available in event buffer after reaching the maximum, before starting to buffer again.

- ndb_eventbuffer_max_alloc

Table 22.248 Type and value information for ndb_eventbuffer_max_alloc

Property	Value
Name	ndb_eventbuffer_max_alloc
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	0 / 0 - 4294967295 (Version: 8.0)
Notes	DESCRIPTION: Maximum memory that can be allocated for buffering events by the NDB API. Defaults to 0 (no limit).

Sets the maximum amount memory (in bytes) that can be allocated for buffering events by the NDB API. 0 means that no limit is imposed, and is the default.

- ndb_extra_logging

Table 22.249 Type and value information for ndb_extra_logging

Property	Value
Name	ndb_extra_logging
Command Line	Yes

Property	Value
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	0 / - (Version: 8.0)
Default, Range	1 / - (Version: NDB 8.0)
Notes	DESCRIPTION: Controls logging of NDB Cluster schema, connection, and data distribution events in the MySQL error log

This variable enables recording in the MySQL error log of information specific to the `NDB` storage engine.

When this variable is set to 0, the only information specific to `NDB` that is written to the MySQL error log relates to transaction handling. If it set to a value greater than 0 but less than 10, `NDB` table schema and connection events are also logged, as well as whether or not conflict resolution is in use, and other `NDB` errors and information. If the value is set to 10 or more, information about `NDB` internals, such as the progress of data distribution among cluster nodes, is also written to the MySQL error log. The default is 1.

- `ndb_force_send`

Table 22.250 Type and value information for ndb_force_send

Property	Value
Name	`ndb_force_send`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	TRUE (Version: 8.0)
Notes	DESCRIPTION: Forces sending of buffers to NDB immediately, without waiting for other threads

Forces sending of buffers to `NDB` immediately, without waiting for other threads. Defaults to `ON`.

- `ndb_fully_replicated`

Table 22.251 Type and value information for ndb_fully_replicated

Property	Value
Name	`ndb_fully_replicated`
Command Line	Yes

Property	Value
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	OFF (Version: NDB 8.0)
Notes	DESCRIPTION: Whether new NDB tables are fully replicated

Determines whether new `NDB` tables are fully replicated. This setting can be overridden for an individual table using `COMMENT="NDB_TABLE=FULLY_REPLICATED=..."` in a `CREATE TABLE` or `ALTER TABLE` statement; see Section 13.1.20.11, “Setting NDB_TABLE Options”, for syntax and other information.

- `ndb_index_stat_enable`

Table 22.252 Type and value information for ndb_index_stat_enable

Property	Value
Name	`ndb_index_stat_enable`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	OFF (Version: 8.0)
Default, Range	ON (Version: NDB 8.0)
Notes	DESCRIPTION: Use NDB index statistics in query optimization

Use `NDB` index statistics in query optimization. The default is `ON`.

- `ndb_index_stat_option`

Table 22.253 Type and value information for ndb_index_stat_option

Property	Value
Name	`ndb_index_stat_option`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both

Property	Value
Dynamic	Yes
Type	
Default, Range	loop_enable=1000ms,loop_idle=1000ms,loop_busy=100ms, update_batch=1,read_batch=4,idle_batch=32,check_batch=8, check_delay=10m,delete_batch=8, clean_delay=1m,error_batch=4, error_delay=1m,evict_batch=8,evict_delay=1m,cache_limit=32M, cache_lowpct=90,zero_total=0 (Version: 8.0)
Default, Range	loop_checkon=1000ms,loop_idle=1000ms,loop_busy=100ms, update_batch=1,read_batch=4,idle_batch=32,check_batch=32, check_delay=1m,delete_batch=8,clean_delay=0,error_batch=4, error_delay=1m,evict_batch=8,evict_delay=1m,cache_limit=32M, cache_lowpct=90 (Version: NDB 8.0)
Notes	DESCRIPTION: Comma-separated list of tunable options for NDB index statistics; the list should contain no spaces

This variable is used for providing tuning options for NDB index statistics generation. The list consist of comma-separated name-value pairs of option names and values, and this list must not contain any space characters.

Options not used when setting `ndb_index_stat_option` are not changed from their default values. For example, you can set `ndb_index_stat_option = 'loop_idle=1000ms,cache_limit=32M'`.

Time values can be optionally suffixed with `h` (hours), `m` (minutes), or `s` (seconds). Millisecond values can optionally be specified using `ms`; millisecond values cannot be specified using `h`, `m`, or `s`.) Integer values can be suffixed with `K`, `M`, or `G`.

The names of the options that can be set using this variable are shown in the table that follows. The table also provides brief descriptions of the options, their default values, and (where applicable) their minimum and maximum values.

Table 22.254 ndb_index_stat_option options and values

Name	Description	Default/Units	Minimum/Maximum
`loop_enable`		1000 ms	0/4G
`loop_idle`	Time to sleep when idle	1000 ms	0/4G
`loop_busy`	Time to sleep when more work is waiting	100 ms	0/4G
`update_batch`		1	0/4G
`read_batch`		4	1/4G
`idle_batch`		32	1/4G
`check_batch`		8	1/4G
`check_delay`	How often to check for new statistics	10 m	1/4G
`delete_batch`		8	0/4G
`clean_delay`		1 m	0/4G

Name	Description	Default/Units	Minimum/Maximum
error_batch		4	1/4G
error_delay		1 m	1/4G
evict_batch		8	1/4G
evict_delay	Clean LRU cache, from read time	1 m	0/4G
cache_limit	Maximum amount of memory in bytes used for cached index statistics by this mysqld; clean up the cache when this is exceeded.	32 M	0/4G
cache_lowpct		90	0/100
zero_total	Setting this to 1 resets all accumulating counters in ndb_index_stat_status to 0. This option value is also reset to 0 when this is done.	0	0/1

-
- ndb_log_apply_status

Table 22.256 Type and value information for ndb_log_apply_status

Property	Value
Name	ndb_log_apply_status
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: Whether or not a MySQL server acting as a slave logs mysql.ndb_apply_status updates received from its immediate master in its own binary log, using its own server ID

A read-only variable which shows whether the server was started with the --ndb-log-apply-status option.

- ndb_log_bin

Table 22.257 Type and value information for ndb_log_bin

Property	Value
Name	ndb_log_bin
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	No
Scope	Both
Dynamic	Yes
Type	
Default, Range	ON (Version: 8.0)
Notes	DESCRIPTION: Write updates to NDB tables in the binary log. Effective only if binary logging is enabled with --log-bin.

Causes updates to NDB tables to be written to the binary log. The setting for this variable has no effect if binary logging is not already enabled on the server using log_bin. In NDB 8.0.16 and later, ndb_log_bin defaults to 0 (FALSE).

- ndb_log_binlog_index

Table 22.258 Type and value information for ndb_log_binlog_index

Property	Value
Name	ndb_log_binlog_index
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	No
Scope	Global
Dynamic	Yes
Type	
Default, Range	ON (Version: 8.0)
Notes	DESCRIPTION: Insert mapping between epochs and binary log positions into the ndb_binlog_index table. Defaults to ON. Effective only if binary logging is enabled on the server.

Causes a mapping of epochs to positions in the binary log to be inserted into the ndb_binlog_index table. Setting this variable has no effect if binary logging is not already enabled for the server using log_bin. (In addition, ndb_log_bin must not be disabled.) ndb_log_binlog_index defaults to 1 (ON); normally, there is never any need to change this value in a production environment.

- ndb_log_empty_epochs

Table 22.259 Type and value information for ndb_log_empty_epochs

Property	Value
Name	ndb_log_empty_epochs
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: When enabled, epochs in which there were no changes are written to the ndb_apply_status and ndb_binlog_index tables, even when log_slave_updates is enabled.

When this variable is set to 0, epoch transactions with no changes are not written to the binary log, although a row is still written even for an empty epoch in ndb_binlog_index.

- ndb_log_empty_update

Table 22.260 Type and value information for ndb_log_empty_update

Property	Value
Name	ndb_log_empty_update
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: When enabled, updates which produce no changes are written to the ndb_apply_status and ndb_binlog_index tables, even when log_slave_updates is enabled.

When this variable is set to ON (1), update transactions with no changes are written to the binary log, even when log_slave_updates is enabled.

- ndb_log_exclusive_reads

Table 22.261 Type and value information for ndb_log_exclusive_reads

Property	Value
Name	ndb_log_exclusive_reads

Property	Value
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	0 (Version: 8.0)
Notes	DESCRIPTION: Log primary key reads with exclusive locks; allow conflict resolution based on read conflicts

This variable determines whether primary key reads are logged with exclusive locks, which allows for NDB Cluster Replication conflict detection and resolution based on read conflicts. To enable these locks, set the value of `ndb_log_exclusive_reads` to 1. 0, which disables such locking, is the default.

For more information, see Read conflict detection and resolution.

- `ndb_log_orig`

Table 22.262 Type and value information for ndb_log_orig

Property	Value
Name	`ndb_log_orig`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: Whether the id and epoch of the originating server are recorded in the mysql.ndb_binlog_index table. Set using the --ndb-log-orig option when starting mysqld.

Shows whether the originating server ID and epoch are logged in the `ndb_binlog_index` table. Set using the `--ndb-log-orig` server option.

- `ndb_log_transaction_id`

Table 22.263 Type and value information for ndb_log_transaction_id

Property	Value
Name	`ndb_log_transaction_id`
Command Line	No
System Variable	Yes

Property	Value
Status Variable	No
Option File	No
Scope	Global
Dynamic	No
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: Whether NDB transaction IDs are written into the binary log (Read-only.)

This read-only, Boolean system variable shows whether a slave `mysqld` writes NDB transaction IDs in the binary log (required to use “active-active” NDB Cluster Replication with `NDB$EPOCH_TRANS()` conflict detection). To change the setting, use the `--ndb-log-transaction-id` option.

`ndb_log_transaction_id` is not supported in mainline MySQL Server 8.0.

For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `ndb_metadata_check`

Table 22.264 Type and value information for ndb_metadata_check

Property	Value
Name	`ndb_metadata_check`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type: Default, Range	boolean: ON (Version: 8.0.16)
Notes	DESCRIPTION: Enable auto-detection of NDB metadata changes with respect to MySQL data dictionary; enabled by default

Beginning with NDB 8.0.16, `NDB` uses a background thread to check for metadata changes each `ndb_metadata_check_interval` seconds as compared with the MySQL data dictionary. This metadata change detection thread can be disabled by setting `ndb_metadata_check` to `OFF`. The thread is enabled by default.

- `ndb_metadata_check_interval`

Table 22.265 Type and value information for ndb_metadata_check_interval

Property	Value
Name	`ndb_metadata_check_interval`
Command Line	Yes
System Variable	Yes
Status Variable	No

Property	Value
Option File	Yes
Scope	Global
Dynamic	Yes
Type: Default, Range	integer: 60 / 0 - 31536000 (Version: 8.0.16)
Notes	DESCRIPTION: Interval in seconds to perform check for NDB metadata changes with respect to MySQL data dictionary

In NDB 8.0.16 and later, NDB runs a metadata change detection thread in the background to determine when the NDB dictionary has changed with respect to the MySQL data dictionary. By default,the interval between such checks is 60 seconds; this can be adjusted by setting the value of `ndb_metadata_check_interval`. To enable or disable the thread, use `ndb_metadata_check`.

- `ndb_optimized_node_selection`

Table 22.266 Type and value information for ndb_optimized_node_selection

Property	Value
Name	`ndb_optimized_node_selection`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	
Default, Range	ON (Version: 8.0)
Default, Range	3 / 0 - 3 (Version: NDB 8.0)
Notes	DESCRIPTION: Determines how an SQL node chooses a cluster data node to use as transaction coordinator

There are two forms of optimized node selection, described here:

1. The SQL node uses *promixity* to determine the transaction coordinator; that is, the “closest” data node to the SQL node is chosen as the transaction coordinator. For this purpose, a data node having a shared memory connection with the SQL node is considered to be “closest” to the SQL node; the next closest (in order of decreasing proximity) are: TCP connection to `localhost`, followed by TCP connection from a host other than `localhost`.

2. The SQL thread uses *distribution awareness* to select the data node. That is, the data node housing the cluster partition accessed by the first statement of a given transaction is used as the transaction coordinator for the entire transaction. (This is effective only if the first statement of the transaction accesses no more than one cluster partition.)

This option takes one of the integer values `0`, `1`, `2`, or `3`. `3` is the default. These values affect node selection as follows:

- `0`: Node selection is not optimized. Each data node is employed as the transaction coordinator 8 times before the SQL thread proceeds to the next data node.

- 1: Proximity to the SQL node is used to determine the transaction coordinator.
- 2: Distribution awareness is used to select the transaction coordinator. However, if the first statement of the transaction accesses more than one cluster partition, the SQL node reverts to the round-robin behavior seen when this option is set to 0.
- 3: If distribution awareness can be employed to determine the transaction coordinator, then it is used; otherwise proximity is used to select the transaction coordinator. (This is the default behavior.)

Proximity is determined as follows:

1. Start with the value set for the Group parameter (default 55).
2. For an API node sharing the same host with other API nodes, decrement the value by 1. Assuming the default value for Group, the effective value for data nodes on same host as the API node is 54, and for remote data nodes 55.
3. Setting ndb_data_node_neighbour further decreases the effective Group value by 50, causing this node to be regarded as the nearest node. This is needed only when all data nodes are on hosts other than that hosts the API node and it is desirable to dedicate one of them to the API node. In normal cases, the default adjustment described previously is sufficient.

Frequent changes in ndb_data_node_neighbour are not advisable, since this changes the state of the cluster connection and thus may disrupt the selection algorithm for new transactions from each thread until it stablilizes.

- ndb_read_backup

Table 22.267 Type and value information for ndb_read_backup

Property	Value
Name	ndb_read_backup
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	OFF (Version: NDB 8.0)
Notes	DESCRIPTION: Enable read from any replica

Enable read from any replica for any NDB table subsequently created.

- ndb_recv_thread_activation_threshold

Table 22.268 Type and value information for ndb_recv_thread_activation_threshold

Property	Value
Name	ndb_recv_thread_activation_threshold
Command Line	No

Property	Value
System Variable	No
Status Variable	No
Option File	No
Scope	
Dynamic	No
Type	
Default, Range	8 / 0 (MIN_ACTIVATION_THRESHOLD) - 16 (MAX_ACTIVATION_THRESHOLD) (Version: NDB 8.0)
Notes	DESCRIPTION: Activation threshold when receive thread takes over the polling of the cluster connection (measured in concurrently active threads)

When this number of concurrently active threads is reached, the receive thread takes over polling of the cluster connection.

This variable is global in scope. It can also be set on startup using the `--ndb-recv-thread-activation-threshold` option.

- `ndb_recv_thread_cpu_mask`

Table 22.269 Type and value information for ndb_recv_thread_cpu_mask

Property	Value
Name	`ndb_recv_thread_cpu_mask`
Command Line	No
System Variable	Yes
Status Variable	No
Option File	No
Scope	Global
Dynamic	Yes
Type	
Default, Range	[empty] (Version: 8.0)
Notes	DESCRIPTION: CPU mask for locking receiver threads to specific CPUs; specified as hexadecimal. See documentation for details.

CPU mask for locking receiver threads to specific CPUs. This is specified as a hexadecimal bitmask. For example, `0x33` means that one CPU is used per receiver thread. An empty string is the default; setting `ndb_recv_thread_cpu_mask` to this value removes any receiver thread locks previously set.

This variable is global in scope. It can also be set on startup using the `--ndb-recv-thread-cpu-mask` option.

- `ndb_report_thresh_binlog_epoch_slip`

Table 22.270 Type and value information for ndb_report_thresh_binlog_epoch_slip

Property	Value
Name	ndb_report_thresh_binlog_epoch_slip
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	3 / 0 - 256 (Version: 8.0)
Default, Range	10 / 0 - 256 (Version: NDB 8.0)
Notes	DESCRIPTION: NDB 7.5.4 and later: Threshold for number of epochs completely buffered, but not yet consumed by binlog injector thread which when exceeded generates BUFFERED_EPOCHS_OVER_THRESHOLD event buffer status message; prior to NDB 7.5.4: Threshold for number of epochs to lag behind before reporting binary log status

This represents the threshold for the number of epochs completely buffered in the event buffer, but not yet consumed by the binlog injector thread. When this degree of slippage (lag) is exceeded, an event buffer status message is reported, with `BUFFERED_EPOCHS_OVER_THRESHOLD` supplied as the reason (see Section 22.5.7.3, “Event Buffer Reporting in the Cluster Log”). Slip is increased when an epoch is received from data nodes and buffered completely in the event buffer; it is decreased when an epoch is consumed by the binlog injector thread, it is reduced. Empty epochs are buffered and queued, and so included in this calculation only when this is enabled using the `Ndb::setEventBufferQueueEmptyEpoch()` method from the NDB API.

- `ndb_report_thresh_binlog_mem_usage`

Table 22.271 Type and value information for ndb_report_thresh_binlog_mem_usage

Property	Value
Name	ndb_report_thresh_binlog_mem_usage
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	10 / 0 - 10 (Version: 8.0)
Notes	DESCRIPTION: This is a threshold on the percentage of free memory remaining before reporting binary log status

This is a threshold on the percentage of free memory remaining before reporting binary log status. For example, a value of `10` (the default) means that if the amount of available memory for receiving binary log data from the data nodes falls below 10%, a status message is sent to the cluster log.

- `ndb_row_checksum`

Table 22.272 Type and value information for ndb_row_checksum

Property	Value
Name	`ndb_row_checksum`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	integer
Default, Range	1 / 0 - 1 (Version: NDB 8.0)
Notes	DESCRIPTION: When enabled, set row checksums; enabled by default

Traditionally, `NDB` has created tables with row checksums, which checks for hardware issues at the expense of performance. Setting `ndb_row_checksum` to 0 means that row checksums are *not* used for new or altered tables, which has a significant impact on performance for all types of queries. This variable is set to 1 by default, to provide backward-compatible behavior.

- `ndb_schema_dist_lock_wait_timeout`

Table 22.273 Type and value information for ndb_show_foreign_key_mock_tables

Property	Value
Name	`ndb_show_foreign_key_mock_tables`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: Show the mock tables used to support foreign_key_checks=0

Number of seconds to wait during schema distribution for a lock. After this time has elapsed, an error is returned. This can be used to avoid having the binlog injector thread wait an excessive length of time while handling schema operations.

- ndb_schema_dist_timeout

Table 22.274 Type and value information for ndb_schema_dist_timeout

Property	Value
Name	ndb_schema_dist_timeout
Command Line	Yes
System Variable	Yes
Option File	Yes
Scope	Global
Dynamic	No
Type: Default, Range	integer: 120 / 5 - 1200 (Version: 8.0)
Notes	DESCRIPTION: Time to wait before detecting timeout during schema distribution

Number of seconds to wait before detecting a timeout during schema distribution. This can indicate that other SQL nodes are experiencing excessive activity, or that they are somehow being prevented from acquiring necessary resources at this time.

- ndb_schema_dist_upgrade_allowed

Table 22.275 Type and value information for ndb_schema_dist_upgrade_allowed

Property	Value
Name	ndb_schema_dist_upgrade_allowed
Command Line	Yes
System Variable	Yes
Option File	Yes
Scope	Global
Dynamic	No
Type: Default, Range	boolean: true (Version: 8.0)
Notes	DESCRIPTION: Allow schema distribution table upgrade when connecting to NDB

Allow upgrading of the schema distribution table when connecting to NDB. When true (the default), this change is deferred until all SQL nodes have been upgraded to the same version of the NDB Cluster software.

Note

The performance of the schema distribution may be somewhat degraded until the upgrade has been performed.

- ndb_show_foreign_key_mock_tables

Table 22.276 Type and value information for ndb_show_foreign_key_mock_tables

Property	Value
Name	ndb_show_foreign_key_mock_tables

Property	Value
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: Show the mock tables used to support foreign_key_checks=0

Show the mock tables used by `NDB` to support `foreign_key_checks=0`. When this is enabled, extra warnings are shown when creating and dropping the tables. The real (internal) name of the table can be seen in the output of `SHOW CREATE TABLE`.

- `ndb_slave_conflict_role`

Table 22.277 Type and value information for ndb_slave_conflict_role

Property	Value
Name	`ndb_slave_conflict_role`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	NONE / NONE, PRIMARY, SECONDARY, PASS (Version: 8.0)
Notes	DESCRIPTION: Role for slave to play in conflict detection and resolution. Value is one of PRIMARY, SECONDARY, PASS, or NONE (default). Can be changed only when slave SQL thread is stopped. See documentation for further information.

Determine the role of this SQL node (and NDB Cluster) in a circular (“active-active”) replication setup. `ndb_slave_conflict_role` can take any one of the values `PRIMARY`, `SECONDARY`, `PASS`, or `NULL` (the default). The slave SQL thread must be stopped before you can change `ndb_slave_conflict_role`. In addition, it is not possible to change directly between `PASS` and either of `PRIMARY` or `SECONDARY` directly; in such cases, you must ensure that the SQL thread is stopped, then execute `SET @@GLOBAL.ndb_slave_conflict_role = 'NONE'` first.

For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `ndb_table_no_logging`

Table 22.278 Type and value information for ndb_table_no_logging

Property	Value
Name	ndb_table_no_logging
Command Line	No
System Variable	Yes
Status Variable	No
Option File	No
Scope	Session
Dynamic	Yes
Type	
Default, Range	FALSE (Version: 8.0)
Notes	DESCRIPTION: NDB tables created when this setting is enabled are not checkpointed to disk (although table schema files are created). The setting in effect when the table is created with or altered to use NDBCLUSTER persists for the lifetime of the table.

When this variable is set to ON or 1, it causes NDB tables not to be checkpointed to disk. More specifically, this setting applies to tables which are created or altered using ENGINE NDB when ndb_table_no_logging is enabled, and continues to apply for the lifetime of the table, even if ndb_table_no_logging is later changed. Suppose that A, B, C, and D are tables that we create (and perhaps also alter), and that we also change the setting for ndb_table_no_logging as shown here:

```
SET @@ndb_table_no_logging = 1;

CREATE TABLE A ... ENGINE NDB;

CREATE TABLE B ... ENGINE MYISAM;
CREATE TABLE C ... ENGINE MYISAM;

ALTER TABLE B ENGINE NDB;

SET @@ndb_table_no_logging = 0;

CREATE TABLE D ... ENGINE NDB;
ALTER TABLE C ENGINE NDB;

SET @@ndb_table_no_logging = 1;
```

After the previous sequence of events, tables A and B are not checkpointed; A was created with ENGINE NDB and B was altered to use NDB, both while ndb_table_no_logging was enabled. However, tables C and D are logged; C was altered to use NDB and D was created using ENGINE NDB, both while ndb_table_no_logging was disabled. Setting ndb_table_no_logging back to 1 or ON does *not* cause table C or D to be checkpointed.

Note

ndb_table_no_logging has no effect on the creation of NDB table schema files; to suppress these, use ndb_table_temporary instead.

- ndb_table_temporary

Table 22.279 Type and value information for ndb_table_temporary

Property	Value
Name	`ndb_table_temporary`
Command Line	No
System Variable	Yes
Status Variable	No
Option File	No
Scope	Session
Dynamic	Yes
Type	
Default, Range	FALSE (Version: 8.0)
Notes	DESCRIPTION: NDB tables are not persistent on disk: no schema files are created and the tables are not logged

When set to `ON` or `1`, this variable causes `NDB` tables not to be written to disk: This means that no table schema files are created, and that the tables are not logged.

Note

Setting this variable currently has no effect. This is a known issue; see Bug #34036.

- `ndb_use_copying_alter_table`

Table 22.280 Type and value information for ndb_use_copying_alter_table

Property	Value
Name	`ndb_use_copying_alter_table`
Command Line	No
System Variable	Yes
Status Variable	No
Option File	No
Scope	Both
Dynamic	No
Type	
Notes	DESCRIPTION: Use copying ALTER TABLE operations in NDB Cluster

Forces `NDB` to use copying of tables in the event of problems with online `ALTER TABLE` operations. The default value is `OFF`.

- `ndb_use_exact_count`

Table 22.281 Type and value information for ndb_use_exact_count

Property	Value
Name	ndb_use_exact_count
Command Line	No
System Variable	Yes
Status Variable	No
Option File	No
Scope	Both
Dynamic	Yes
Type	
Default, Range	ON (Version: 8.0)
Default, Range	OFF (Version: NDB 8.0)
Notes	DESCRIPTION: Use exact row count when planning queries

Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query. The default value is OFF, which allows for faster queries overall.

- ndb_use_transactions

Table 22.282 Type and value information for ndb_use_transactions

Property	Value
Name	ndb_use_transactions
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Both
Dynamic	Yes
Type	
Default, Range	ON (Version: 8.0)
Notes	DESCRIPTION: Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query

You can disable NDB transaction support by setting this variable's values to OFF (not recommended). The default is ON.

- ndb_version

Table 22.283 Type and value information for ndb_version

Property	Value
Name	ndb_version
Command Line	No

Property	Value
System Variable	Yes
Status Variable	No
Option File	No
Scope	Global
Dynamic	No
Type	
Default, Range	(Version: 8.0)
Notes	DESCRIPTION: Shows build and NDB engine version as an integer

`NDB` engine version, as a composite integer.

- `ndb_version_string`

Table 22.284 Type and value information for ndb_version_string

Property	Value
Name	`ndb_version_string`
Command Line	No
System Variable	Yes
Status Variable	No
Option File	No
Scope	Global
Dynamic	No
Type	
Default, Range	(Version: 8.0)
Notes	DESCRIPTION: Shows build information including NDB engine version in ndb-x.y.z format

`NDB` engine version in `ndb-x.y.z` format.

- `server_id_bits`

Table 22.285 Type and value information for server_id_bits

Property	Value
Name	`server_id_bits`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	No
Type	

Property	Value
Default, Range	32 / 7 - 32 (Version: 8.0)
Notes	DESCRIPTION: The effective value of server_id if the server was started with the --server-id-bits option set to a nondefault value

The effective value of `server_id` if the server was started with the `--server-id-bits` option set to a nondefault value.

If the value of `server_id` greater than or equal to 2 to the power of `server_id_bits`, `mysqld` refuses to start.

This system variable is supported only by NDB Cluster. `server_id_bits` is not supported by the standard MySQL Server.

- `slave_allow_batching`

Table 22.286 Type and value information for slave_allow_batching

Property	Value
Name	`slave_allow_batching`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	Yes
Scope	Global
Dynamic	Yes
Type	
Default, Range	off (Version: 8.0)
Notes	DESCRIPTION: Turns update batching on and off for a replication slave

Whether or not batched updates are enabled on NDB Cluster replication slaves.

Setting this variable has an effect only when using replication with the `NDB` storage engine; in MySQL Server 8.0, it is present but does nothing. For more information, see Section 22.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”.

- `transaction_allow_batching`

Table 22.287 Type and value information for transaction_allow_batching

Property	Value
Name	`transaction_allow_batching`
Command Line	No
System Variable	Yes
Status Variable	No
Option File	No
Scope	Session

Property	Value
Dynamic	Yes
Type	
Default, Range	FALSE (Version: 8.0)
Notes	DESCRIPTION: Allows batching of statements within a transaction. Disable AUTOCOMMIT to use.

When set to `1` or `ON`, this variable enables batching of statements within the same transaction. To use this variable, `autocommit` must first be disabled by setting it to `0` or `OFF`; otherwise, setting `transaction_allow_batching` has no effect.

It is safe to use this variable with transactions that performs writes only, as having it enabled can lead to reads from the “before” image. You should ensure that any pending transactions are committed (using an explicit `COMMIT` if desired) before issuing a `SELECT`.

Important

`transaction_allow_batching` should not be used whenever there is the possibility that the effects of a given statement depend on the outcome of a previous statement within the same transaction.

This variable is currently supported for NDB Cluster only.

The system variables in the following list all relate to the `ndbinfo` information database.

- `ndbinfo_database`

Table 22.288 Type and value information for ndbinfo_database

Property	Value
Name	`ndbinfo_database`
Command Line	No
System Variable	Yes
Status Variable	No
Option File	No
Scope	Global
Dynamic	No
Type	
Default, Range	ndbinfo (Version: 8.0)
Notes	DESCRIPTION: The name used for the NDB information database; read only

Shows the name used for the `NDB` information database; the default is `ndbinfo`. This is a read-only variable whose value is determined at compile time; you can set it by starting the server using `--ndbinfo-database=`*`name`*, which sets the value shown for this variable but does not actually change the name used for the NDB information database.

- `ndbinfo_max_bytes`

Table 22.289 Type and value information for ndbinfo_max_bytes

Property	Value
Name	ndbinfo_max_bytes
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	No
Scope	Both
Dynamic	Yes
Type	
Default, Range	0 / - (Version: 8.0)
Notes	DESCRIPTION: Used for debugging only

Used in testing and debugging only.

- ndbinfo_max_rows

Table 22.290 Type and value information for ndbinfo_max_rows

Property	Value
Name	ndbinfo_max_rows
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	No
Scope	Both
Dynamic	Yes
Type	
Default, Range	10 / - (Version: 8.0)
Notes	DESCRIPTION: Used for debugging only

Used in testing and debugging only.

- ndbinfo_offline

Table 22.291 Type and value information for ndbinfo_offline

Property	Value
Name	ndbinfo_offline
Command Line	No
System Variable	Yes
Status Variable	No
Option File	No
Scope	Global

Property	Value
Dynamic	Yes
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: Put the ndbinfo database into offline mode, in which no rows are returned from tables or views

Place the `ndbinfo` database into offline mode, in which tables and views can be opened even when they do not actually exist, or when they exist but have different definitions in `NDB`. No rows are returned from such tables (or views).

- `ndbinfo_show_hidden`

Table 22.292 Type and value information for ndbinfo_show_hidden

Property	Value
Name	`ndbinfo_show_hidden`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	No
Scope	Both
Dynamic	Yes
Type	
Default, Range	OFF (Version: 8.0)
Notes	DESCRIPTION: Whether to show ndbinfo internal base tables in the mysql client. The default is OFF.

Whether or not the `ndbinfo` database's underlying internal tables are shown in the `mysql` client. The default is `OFF`.

- `ndbinfo_table_prefix`

Table 22.293 Type and value information for ndbinfo_table_prefix

Property	Value
Name	`ndbinfo_table_prefix`
Command Line	Yes
System Variable	Yes
Status Variable	No
Option File	No
Scope	Both
Dynamic	Yes
Type	
Default, Range	ndb$ (Version: 8.0)

Property	Value
Notes	DESCRIPTION: The prefix to use for naming ndbinfo internal base tables

The prefix used in naming the ndbinfo database's base tables (normally hidden, unless exposed by setting `ndbinfo_show_hidden`). This is a read-only variable whose default value is `ndb$`. You can start the server with the `--ndbinfo-table-prefix` option, but this merely sets the variable and does not change the actual prefix used to name the hidden base tables; the prefix itself is determined at compile time.

- `ndbinfo_version`

Table 22.294 Type and value information for ndbinfo_version

Property	Value
Name	`ndbinfo_version`
Command Line	No
System Variable	Yes
Status Variable	No
Option File	No
Scope	Global
Dynamic	No
Type	
Default, Range	(Version: 8.0)
Notes	DESCRIPTION: The version of the ndbinfo engine; read only

Shows the version of the `ndbinfo` engine in use; read-only.

NDB Cluster Status Variables

This section provides detailed information about MySQL server status variables that relate to NDB Cluster and the `NDB` storage engine. For status variables not specific to NDB Cluster, and for general information on using status variables, see Section 5.1.10, “Server Status Variables”.

- `Handler_discover`

  The MySQL server can ask the `NDBCLUSTER` storage engine if it knows about a table with a given name. This is called discovery. `Handler_discover` indicates the number of times that tables have been discovered using this mechanism.

- `Ndb_api_bytes_sent_count_session`

  Amount of data (in bytes) sent to the data nodes in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_sent_count_slave`

  Amount of data (in bytes) sent to the data nodes by this slave.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_sent_count`

  Amount of data (in bytes) sent to the data nodes by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_received_count_session`

  Amount of data (in bytes) received from the data nodes in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_received_count_slave`

  Amount of data (in bytes) received from the data nodes by this slave.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_received_count`

  Amount of data (in bytes) received from the data nodes by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_data_count_injector`

  The number of row change events received by the NDB binlog injector thread.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_data_count`

  The number of row change events received by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_nondata_count_injector`

  The number of events received, other than row change events, by the NDB binary log injector thread.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_nondata_count`

  The number of events received, other than row change events, by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_bytes_count_injector`

  The number of bytes of events received by the NDB binlog injector thread.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_bytes_count`

  The number of bytes of events received by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pk_op_count_session`

  The number of operations in this client session based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pk_op_count_slave`

  The number of operations by this slave based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pk_op_count`

  The number of operations by this MySQL Server (SQL node) based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pruned_scan_count_session`

  The number of scans in this client session that have been pruned to a single partition.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pruned_scan_count_slave`

  The number of scans by this slave that have been pruned to a single partition.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pruned_scan_count`

  The number of scans by this MySQL Server (SQL node) that have been pruned to a single partition.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_range_scan_count_session`

  The number of range scans that have been started in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_range_scan_count_slave`

  The number of range scans that have been started by this slave.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_range_scan_count`

  The number of range scans that have been started by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_read_row_count_session`

  The total number of rows that have been read in this client session. This includes all rows read by any primary key, unique key, or scan operation made in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_read_row_count_slave`

  The total number of rows that have been read by this slave. This includes all rows read by any primary key, unique key, or scan operation made by this slave.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_read_row_count`

  The total number of rows that have been read by this MySQL Server (SQL node). This includes all rows read by any primary key, unique key, or scan operation made by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_scan_batch_count_session`

  The number of batches of rows received in this client session. 1 batch is defined as 1 set of scan results from a single fragment.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_scan_batch_count_slave`

  The number of batches of rows received by this slave. 1 batch is defined as 1 set of scan results from a single fragment.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_scan_batch_count`

  The number of batches of rows received by this MySQL Server (SQL node). 1 batch is defined as 1 set of scan results from a single fragment.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_table_scan_count_session`

  The number of table scans that have been started in this client session, including scans of internal tables,.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_table_scan_count_slave`

  The number of table scans that have been started by this slave, including scans of internal tables,.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_table_scan_count`

  The number of table scans that have been started by this MySQL Server (SQL node), including scans of internal tables,.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_abort_count_session`

  The number of transactions aborted in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_abort_count_slave`

  The number of transactions aborted by this slave.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_abort_count`

  The number of transactions aborted by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_close_count_session`

  The number of transactions closed in this client session. This value may be greater than the sum of `Ndb_api_trans_commit_count_session` and `Ndb_api_trans_abort_count_session`, since some transactions may have been rolled back.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_close_count_slave`

  The number of transactions closed by this slave. This value may be greater than the sum of `Ndb_api_trans_commit_count_slave` and `Ndb_api_trans_abort_count_slave`, since some transactions may have been rolled back.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_close_count`

  The number of transactions closed by this MySQL Server (SQL node). This value may be greater than the sum of `Ndb_api_trans_commit_count` and `Ndb_api_trans_abort_count`, since some transactions may have been rolled back.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_commit_count_session`

  The number of transactions committed in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_commit_count_slave`

  The number of transactions committed by this slave.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_commit_count`

  The number of transactions committed by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_local_read_row_count_session`

  The total number of rows that have been read in this client session. This includes all rows read by any primary key, unique key, or scan operation made in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_local_read_row_count_slave`

  The total number of rows that have been read by this slave. This includes all rows read by any primary key, unique key, or scan operation made by this slave.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_local_read_row_count`

  The total number of rows that have been read by this MySQL Server (SQL node). This includes all rows read by any primary key, unique key, or scan operation made by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_start_count_session`

  The number of transactions started in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_start_count_slave`

  The number of transactions started by this slave.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_start_count`

  The number of transactions started by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_uk_op_count_session`

  The number of operations in this client session based on or using unique keys.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_uk_op_count_slave`

  The number of operations by this slave based on or using unique keys.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_uk_op_count`

  The number of operations by this MySQL Server (SQL node) based on or using unique keys.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_exec_complete_count_session`

  The number of times a thread has been blocked in this client session while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit executes for blob and auto-increment operations not visible to clients.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_exec_complete_count_slave`

The number of times a thread has been blocked by this slave while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit executes for blob and auto-increment operations not visible to clients.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_exec_complete_count`

  The number of times a thread has been blocked by this MySQL Server (SQL node) while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit executes for blob and auto-increment operations not visible to clients.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_meta_request_count_session`

  The number of times a thread has been blocked in this client session waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_meta_request_count_slave`

  The number of times a thread has been blocked by this slave waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_meta_request_count`

  The number of times a thread has been blocked by this MySQL Server (SQL node) waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_nanos_count_session`

  Total time (in nanoseconds) spent in this client session waiting for any type of signal from the data nodes.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_nanos_count_slave`

  Total time (in nanoseconds) spent by this slave waiting for any type of signal from the data nodes.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_nanos_count`

  Total time (in nanoseconds) spent by this MySQL Server (SQL node) waiting for any type of signal from the data nodes.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_scan_result_count_session`

  The number of times a thread has been blocked in this client session while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_scan_result_count_slave`

  The number of times a thread has been blocked by this slave while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replication slave, or does not use NDB tables, this value is always 0.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_scan_result_count`

  The number of times a thread has been blocked by this MySQL Server (SQL node) while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 22.5.17, “NDB API Statistics Counters and Variables”.

- `Ndb_cluster_node_id`

  If the server is acting as an NDB Cluster node, then the value of this variable its node ID in the cluster.

  If the server is not part of an NDB Cluster, then the value of this variable is 0.

- `Ndb_config_from_host`

  If the server is part of an NDB Cluster, the value of this variable is the host name or IP address of the Cluster management server from which it gets its configuration data.

  If the server is not part of an NDB Cluster, then the value of this variable is an empty string.

- `Ndb_config_from_port`

  If the server is part of an NDB Cluster, the value of this variable is the number of the port through which it is connected to the Cluster management server from which it gets its configuration data.

  If the server is not part of an NDB Cluster, then the value of this variable is 0.

- `Ndb_conflict_fn_max_del_win`

  Shows the number of times that a row was rejected on the current SQL node due to NDB Cluster Replication conflict resolution using `NDB$MAX_DELETE_WIN()`, since the last time that this `mysqld` was started.

  For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_fn_max`

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of times that a row was not applied on the current SQL node due to “greatest timestamp wins” conflict resolution since the last time that this `mysqld` was started.

  For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_fn_old`

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of times that a row was not applied as the result of “same timestamp wins” conflict resolution on a given `mysqld` since the last time it was restarted.

  For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_fn_epoch`

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH()` conflict resolution on a given `mysqld` since the last time it was restarted.

  For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_fn_epoch2`

  Shows the number of rows found to be in conflict in NDB Cluster Replication conflict resolution, when using `NDB$EPOCH2()`, on the master designated as the primary since the last time it was restarted.

  For more information, see NDB$EPOCH2().

- `Ndb_conflict_fn_epoch_trans`

Used in NDB Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH_TRANS()` conflict resolution on a given `mysqld` since the last time it was restarted.

For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_fn_epoch2_trans`

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH_TRANS2()` conflict resolution on a given `mysqld` since the last time it was restarted.

  For more information, see NDB$EPOCH2_TRANS().

- `Ndb_conflict_last_conflict_epoch`

  The most recent epoch in which a conflict was detected on this slave. You can compare this value with `Ndb_slave_max_replicated_epoch`; if `Ndb_slave_max_replicated_epoch` is greater than `Ndb_conflict_last_conflict_epoch`, no conflicts have yet been detected.

  See Section 22.6.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- `Ndb_conflict_reflected_op_discard_count`

  When using NDB Cluster Replication conflict resolution, this is the number of reflected operations that were not applied on the secondary, due to encountering an error during execution.

  See Section 22.6.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- `Ndb_conflict_reflected_op_prepare_count`

  When using conflict resolution with NDB Cluster Replication, this status variable contains the number of reflected operations that have been defined (that is, prepared for execution on the secondary).

  See Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_refresh_op_count`

  When using conflict resolution with NDB Cluster Replication, this gives the number of refresh operations that have been prepared for execution on the secondary.

  See Section 22.6.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- `Ndb_conflict_last_stable_epoch`

  Number of rows found to be in conflict by a transactional conflict function

  See Section 22.6.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- `Ndb_conflict_trans_row_conflict_count`

  Used in NDB Cluster Replication conflict resolution, this status variable shows the number of rows found to be directly in-conflict by a transactional conflict function on a given `mysqld` since the last time it was restarted.

  Currently, the only transactional conflict detection function supported by NDB Cluster is NDB$EPOCH_TRANS(), so this status variable is effectively the same as `Ndb_conflict_fn_epoch_trans`.

For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_trans_row_reject_count`

  Used in NDB Cluster Replication conflict resolution, this status variable shows the total number of rows realigned due to being determined as conflicting by a transactional conflict detection function. This includes not only `Ndb_conflict_trans_row_conflict_count`, but any rows in or dependent on conflicting transactions.

  For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_trans_reject_count`

  Used in NDB Cluster Replication conflict resolution, this status variable shows the number of transactions found to be in conflict by a transactional conflict detection function.

  For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_trans_detect_iter_count`

  Used in NDB Cluster Replication conflict resolution, this shows the number of internal iterations required to commit an epoch transaction. Should be (slightly) greater than or equal to `Ndb_conflict_trans_conflict_commit_count`.

  For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_trans_conflict_commit_count`

  Used in NDB Cluster Replication conflict resolution, this shows the number of epoch transactions committed after they required transactional conflict handling.

  For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_epoch_delete_delete_count`

  When using delete-delete conflict detection, this is the number of delete-delete conflicts detected, where a delete operation is applied, but the indicated row does not exist.

- `Ndb_execute_count`

  Provides the number of round trips to the `NDB` kernel made by operations.

- `Ndb_last_commit_epoch_server`

  The epoch most recently committed by `NDB`.

- `Ndb_last_commit_epoch_session`

  The epoch most recently committed by this `NDB` client.

- `Ndb_metadata_blacklist_size`

  The number of metadata objects that the NDB binlog thread has been unable to synchronize on this SQL node since it was last restarted.

  Should an object be blacklisted, it is not again considered for automatic synchronization until the user corrects the mismatch manually. This can be done by attempting to use the table with a statement such

as `SHOW CREATE TABLE table`, `SELECT * FROM table`, or any other statement that would trigger table discovery.

Added in NDB 8.0.18.

- `Ndb_metadata_detected_count`

  The number of times since this server was last started that the NDB metadata change detection thread has discovered changes with respect to the MySQL data dictionary.

  Added in NDB 8.0.16.

- `Ndb_metadata_synced_count`

  The number of NDB metadata objects which have been synchronized on this SQL node since it was last restarted.

  Added in NDB 8.0.18.

- `Ndb_number_of_data_nodes`

  If the server is part of an NDB Cluster, the value of this variable is the number of data nodes in the cluster.

  If the server is not part of an NDB Cluster, then the value of this variable is 0.

- `Ndb_pushed_queries_defined`

  The total number of joins pushed down to the NDB kernel for distributed handling on the data nodes.

  Note

  Joins tested using `EXPLAIN` that can be pushed down contribute to this number.

- `Ndb_pushed_queries_dropped`

  The number of joins that were pushed down to the NDB kernel but that could not be handled there.

- `Ndb_pushed_queries_executed`

  The number of joins successfully pushed down to `NDB` and executed there.

- `Ndb_pushed_reads`

  The number of rows returned to `mysqld` from the NDB kernel by joins that were pushed down.

  Note

  Executing `EXPLAIN` on joins that can be pushed down to `NDB` does not add to this number.

- `Ndb_pruned_scan_count`

  This variable holds a count of the number of scans executed by `NDBCLUSTER` since the NDB Cluster was last started where `NDBCLUSTER` was able to use partition pruning.

  Using this variable together with `Ndb_scan_count` can be helpful in schema design to maximize the ability of the server to prune scans to a single table partition, thereby involving only a single data node.

- `Ndb_scan_count`

  This variable holds a count of the total number of scans executed by `NDBCLUSTER` since the NDB Cluster was last started.

- `Ndb_slave_max_replicated_epoch`

  The most recently committed epoch on this slave. You can compare this value with `Ndb_conflict_last_conflict_epoch`; if `Ndb_slave_max_replicated_epoch` is the greater of the two, no conflicts have yet been detected.

  For more information, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_system_name`

  If this MySQL Server is connected to an NDB cluster, this read-only variable shows the cluster system name. Otherwise, the value is an empty string.

- `Ndb_trans_hint_count_session`

  The number of transactions using hints that have been started in the current session. Compare with `Ndb_api_trans_start_count_session` to obtain the proportion of all NDB transactions able to use hints. Added in NDB 8.0.17.

22.3.3.10 NDB Cluster TCP/IP Connections

TCP/IP is the default transport mechanism for all connections between nodes in an NDB Cluster. Normally it is not necessary to define TCP/IP connections; NDB Cluster automatically sets up such connections for all data nodes, management nodes, and SQL or API nodes.

Note

For an exception to this rule, see Section 22.3.3.11, “NDB Cluster TCP/IP Connections Using Direct Connections”.

To override the default connection parameters, it is necessary to define a connection using one or more `[tcp]` sections in the `config.ini` file. Each `[tcp]` section explicitly defines a TCP/IP connection between two NDB Cluster nodes, and must contain at a minimum the parameters `NodeId1` and `NodeId2`, as well as any connection parameters to override.

It is also possible to change the default values for these parameters by setting them in the `[tcp default]` section.

Important

Any `[tcp]` sections in the `config.ini` file should be listed *last*, following all other sections in the file. However, this is not required for a `[tcp default]` section. This requirement is a known issue with the way in which the `config.ini` file is read by the NDB Cluster management server.

Connection parameters which can be set in `[tcp]` and `[tcp default]` sections of the `config.ini` file are listed here:

- `NodeId1`

Table 22.295 This table provides type and value information for the NodeId1 TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	[none]
Range	1 - 255
Restart Type	N

To identify a connection between two nodes it is necessary to provide their node IDs in the `[tcp]` section of the configuration file as the values of `NodeId1` and `NodeId2`. These are the same unique `Id` values for each of these nodes as described in Section 22.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

- `NodeId2`

Table 22.296 This table provides type and value information for the NodeId2 TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	[none]
Range	1 - 255
Restart Type	N

To identify a connection between two nodes it is necessary to provide their node IDs in the `[tcp]` section of the configuration file as the values of `NodeId1` and `NodeId2`. These are the same unique `Id` values for each of these nodes as described in Section 22.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

- `HostName1`

Table 22.297 This table provides type and value information for the HostName1 TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given TCP connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `HostName2`

Table 22.298 This table provides type and value information for the HostName1 TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given TCP connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `OverloadLimit`

Table 22.299 This table provides type and value information for the OverloadLimit TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When more than this many unsent bytes are in the send buffer, the connection is considered overloaded.

This parameter can be used to determine the amount of unsent data that must be present in the send buffer before the connection is considered overloaded. See Section 22.3.3.13, “Configuring NDB Cluster Send Buffer Parameters”, for more information.

- `SendBufferMemory`

Table 22.300 This table provides type and value information for the SendBufferMemory TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	2M
Range	256K - 4294967039 (0xFFFFFEFF)
Restart Type	N

TCP transporters use a buffer to store all messages before performing the send call to the operating system. When this buffer reaches 64KB its contents are sent; these are also sent when a round of messages have been executed. To handle temporary overload situations it is also possible to define a bigger send buffer.

If this parameter is set explicitly, then the memory is not dedicated to each transporter; instead, the value used denotes the hard limit for how much memory (out of the total available memory—that is, `TotalSendBufferMemory`) that may be used by a single transporter. For more information about configuring dynamic transporter send buffer memory allocation in NDB Cluster, see Section 22.3.3.13, “Configuring NDB Cluster Send Buffer Parameters”.

The default size of the send buffer is 2MB, which is the size recommended in most situations. The minimum size is 64 KB; the theoretical maximum is 4 GB.

- `SendSignalId`

Table 22.301 This table provides type and value information for the SendSignalId TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	[see text]
Range	true, false
Restart Type	N

To be able to retrace a distributed message datagram, it is necessary to identify each message. When this parameter is set to `Y`, message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- `Checksum`

Table 22.302 This table provides type and value information for the Checksum TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

This parameter is a boolean parameter (enabled by setting it to `Y` or `1`, disabled by setting it to `N` or `0`). It is disabled by default. When it is enabled, checksums for all messages are calculated before they placed in the send buffer. This feature ensures that messages are not corrupted while waiting in the send buffer, or by the transport mechanism.

- `PreSendChecksum`

Table 22.303 This table provides type and value information for the PreSendChecksum TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean

Property	Value
Default	false
Range	true, false
Restart Type	S

If this parameter and `Checksum` are both enabled, perform pre-send checksum checks, and check all TCP signals between nodes for errors. Has no effect if `Checksum` is not also enabled.

- `ReceiveBufferMemory`

Table 22.304 This table provides type and value information for the ReceiveBufferMemory TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	2M
Range	16K - 4294967039 (0xFFFFFEFF)
Restart Type	N

Specifies the size of the buffer used when receiving data from the TCP/IP socket.

The default value of this parameter is 2MB. The minimum possible value is 16KB; the theoretical maximum is 4GB.

- `TCP_RCV_BUF_SIZE`

Table 22.305 This table provides type and value information for the TCP_RCV_BUF_SIZE TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0
Range	0 - 2G
Restart Type	N

Determines the size of the receive buffer set during TCP transporter initialization. The default and minimum value is 0, which allows the operating system or platform to set this value. The default is recommended for most common usage cases.

- `TCP_SND_BUF_SIZE`

Table 22.306 This table provides type and value information for the TCP_SND_BUF_SIZE TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0

Property	Value
Range	0 - 2G
Restart Type	N

Determines the size of the send buffer set during TCP transporter initialization. The default and minimum value is 0, which allows the operating system or platform to set this value. The default is recommended for most common usage cases.

- `TCP_MAXSEG_SIZE`

Table 22.307 This table provides type and value information for the TCP_MAXSEG_SIZE TCP configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0
Range	0 - 2G
Restart Type	N

Determines the size of the memory set during TCP transporter initialization. The default is recommended for most common usage cases.

- `TcpBind_INADDR_ANY`

  Setting this parameter to `TRUE` or `1` binds `IP_ADDR_ANY` so that connections can be made from anywhere (for autogenerated connections). The default is `FALSE` (`0`).

- `Group`

  When `ndb_optimized_node_selection` is enabled, node proximity is used in some cases to select which node to connect to. This parameter can be used to influence proximity by setting it to a lower value, which is interpreted as “closer”. See the description of the system variable for more information.

22.3.3.11 NDB Cluster TCP/IP Connections Using Direct Connections

Setting up a cluster using direct connections between data nodes requires specifying explicitly the crossover IP addresses of the data nodes so connected in the `[tcp]` section of the cluster `config.ini` file.

In the following example, we envision a cluster with at least four hosts, one each for a management server, an SQL node, and two data nodes. The cluster as a whole resides on the `172.23.72.*` subnet of a LAN. In addition to the usual network connections, the two data nodes are connected directly using a standard crossover cable, and communicate with one another directly using IP addresses in the `1.1.0.*` address range as shown:

```
# Management Server
[ndb_mgmd]
Id=1
HostName=172.23.72.20

# SQL Node
[mysqld]
```

```
Id=2
HostName=172.23.72.21

# Data Nodes
[ndbd]
Id=3
HostName=172.23.72.22

[ndbd]
Id=4
HostName=172.23.72.23

# TCP/IP Connections
[tcp]
NodeId1=3
NodeId2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

The `HostName1` and `HostName2` parameters are used only when specifying direct connections.

The use of direct TCP connections between data nodes can improve the cluster's overall efficiency by enabling the data nodes to bypass an Ethernet device such as a switch, hub, or router, thus cutting down on the cluster's latency.

Note

To take the best advantage of direct connections in this fashion with more than two data nodes, you must have a direct connection between each data node and every other data node in the same node group.

22.3.3.12 NDB Cluster Shared-Memory Connections

Communications between NDB cluster nodes are normally handled using TCP/IP. The shared memory (SHM) transporter is distinguished by the fact that signals are transmitted by writing in memory rather than on a socket. The shared-memory transporter (SHM) can improve performance by negating up to 20% of the overhead required by a TCP connection when running an API node (usually an SQL node) and a data node together on the same host. NDB Cluster attempts to use the shared memory transporter and configure it automatically between data nodes and API nodes on the same host. You can enable a shared memory connection explicitly, by setting the `UseShm` data node configuration parameter to `1`. When explicitly defining shared memory as the connection method, it is also necessary to set `HostName` for the data node and `HostName` for the API node, and that these be the same. It is also possible to employ multiple SHM connections in the same NDB cluster, on different hosts, each having one API node and one data node; see later in this section for an exaample of how to do this.

Suppose a cluster is running a data node which has node ID 1 and an SQL node having node ID 51 on the same host computer at 10.0.0.1. To enable an SHM connection between these two nodes, all that is necessary is to insure that the following entries are included in the cluster configuration file:

```
[ndbd]
NodeId=1
HostName=10.0.0.1
UseShm=1

[mysqld]
NodeId=51
HostName=10.0.0.1
```

Important

The two entries just shown are in addition to any other entries and parameter settings needed by the cluster. A more comlete example is shown later in this section.

Before starting data nodes that use SHM connections, it is also necessary to make sure that the operating system on each computer hosting such a data node has sufficient memory allocated to shared memory segments. See the documentation for your operating platform for information regarding this. In setups where multiple hosts are each running a data node and an API node, it is possible to enable shared memory on all such hosts by setting `UseShm` in the `[ndbd default]` section of the configuration file. This is shown in the example later in this section.

While not strictly required, tuning for all SHM connections in the cluster can be done by setting one or more of the following parameters in the `[shm default]` section of the cluster configuration (`config.ini`) file:

- `ShmSize`: Shared memory size
- `ShmSpinTime`: Time in µs to spin before sleeping
- `SendBufferMemory`: Size of buffer for signals sent from this node, in bytes.
- `SendSignalId`: Indicates that a signal ID is included in each signal sent through the transporter.
- `Checksum`: Indicates that a checksum is included in each signal sent through the transporter.
- `PreSendChecksum`: Checks of the checksum are made prior to sending the signal; Checksum must also be enabled for this to work

This example shows a simple setup with SHM connections definied on multiple hosts, in an NDB Cluster using 3 computers listed here by host name, hosting the node types shown:

1. `10.0.0.0`: The management server
2. `10.0.0.1`: A data node and an SQL node
3. `10.0.0.2`: A data node and an SQL node

In this scenario, each data node communicates with both the management server and the other data node using TCP transporters; each SQL node uses a shared memory transporter to communicate with the data nodes that is local to it, and a TCP transporter to communicate with the remote data node. A basic configuration reflecting this setup is enabled by the config.ini file whose contents are shown here:

```
[ndbd default]
DataDir=/path/to/datadir
UseShm=1

[shm default]
ShmSize=8M
ShmSpintime=200
SendBufferMemory=4M

[tcp default]
SendBufferMemory=8M

[ndb_mgmd]
NodeId=49
Hostname=10.0.0.0
DataDir=/path/to/datadir

[ndbd]
```

```
NodeId=1
Hostname=10.0.0.1
DataDir=/path/to/datadir

[ndbd]
NodeId=2
Hostname=10.0.0.2
DataDir=/path/to/datadir

[mysqld]
NodeId=51
Hostname=10.0.0.1

[mysqld]
NodeId=52
Hostname=10.0.0.2

[api]
[api]
```

Parameters affecting all shared memory transporters are set in the `[shm default]` section; these can be overridden on a per-connection basis in one or more `[shm]` sections. Each such section must be associated with a given SHM connection using `NodeId1` and `NodeId2`; the values required for these parameters are the node IDs of the two nodes connected by the transporter. You can also identify the nodes by host name using `HostName1` and `HostName2`, but these parameters are not required.

The API nodes for which no host names are set use the TCP transporter to communicate with data nodes independent of the hosts on which they are started; the parameters and values set in the `[tcp default]` section of the configuration file apply to all TCP transporters in the cluster.

For optimum performance, you can define a spin time for the SHM transporter (`ShmSpinTime` parameter); this affects both the data node receiver thread and the poll owner (receive thread or user thread) in `NDB`.

Restart types. Information about the restart types used by the parameter descriptions in this section is shown in the following table:

Table 22.308 NDB Cluster restart types

Symbol	Restart Type	Description
N	Node	The parameter can be updated using a rolling restart (see Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”)
S	System	All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter
I	Initial	Data nodes must be restarted using the `--initial` option

- `Checksum`

  Table 22.309 This table provides type and value information for the Checksum shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	true
Range	true, false
Restart Type	N

This parameter is a boolean (`Y`/`N`) parameter which is disabled by default. When it is enabled, checksums for all messages are calculated before being placed in the send buffer.

This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

- `Group`

Table 22.310 This table provides type and value information for the Group shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	35
Range	0-200
Restart Type	N

Determines the group proximity; a smaller value is interpreted as being closer. The default value is sufficient for most conditions.

- `HostName1`

Table 22.311 This table provides type and value information for the HostName1 shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SHM connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `HostName2`

Table 22.312 This table provides type and value information for the HostName1 shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SHM connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `NodeId1`

Table 22.313 This table provides type and value information for the NodeId1 shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	[none]
Range	1 - 255
Restart Type	N

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `NodeId2`

Table 22.314 This table provides type and value information for the NodeId2 shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	[none]
Range	1 - 255
Restart Type	N

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `NodeIdServer`

Table 22.315 This table provides type and value information for the NodeIdServer shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	numeric
Default	[none]
Range	1 - 63
Restart Type	N

Identify the server end of a shared memory connection. By default, this is the node ID of the data node.

- `OverloadLimit`

Table 22.316 This table provides type and value information for the OverloadLimit shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When more than this many unsent bytes are in the send buffer, the connection is considered overloaded. See Section 22.3.3.13, “Configuring NDB Cluster Send Buffer Parameters”, and Section 22.5.10.44, “The ndbinfo transporters Table”, for more information.

- `PreSendChecksum`

Table 22.317 This table provides type and value information for the PreSendChecksum shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	S

If this parameter and `Checksum` are both enabled, perform pre-send checksum checks, and check all SHM signals between nodes for errors. Has no effect if `Checksum` is not also enabled.

- `SendBufferMemory`

Table 22.318 This table provides type and value information for the SendBufferMemory shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	2M
Range	256K - 4294967039 (0xFFFFFEFF)
Restart Type	N

Size (in bytes) of the shared memory buffer for signals sent from this node using a shared memory connection.

- `SendSignalId`

Table 22.319 This table provides type and value information for the SendSignalId shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

To retrace the path of a distributed message, it is necessary to provide each message with a unique identifier. Setting this parameter to `Y` causes these message IDs to be transported over the network as well. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- `ShmKey`

Table 22.320 This table provides type and value information for the ShmKey shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When setting up shared memory segments, a node ID, expressed as an integer, is used to identify uniquely the shared memory segment to use for the communication. There is no default value. If `UseShm` is enabled, the shared memory key is calculated automatically by `NDB`.

- `ShmSize`

Table 22.321 This table provides type and value information for the ShmSize shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	bytes
Default	4M
Range	64K - 4294967039 (0xFFFFFEFF)
Restart Type	N

Each SHM connection has a shared memory segment where messages between nodes are placed by the sender and read by the reader. The size of this segment is defined by `ShmSize`. The default value is 4MB.

- `ShmSpinTime`

Table 22.322 This table provides type and value information for the ShmSpinTime shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	integer
Default	0
Range	0 - 2000
Restart Type	S

When receiving, the time to wait before sleeping, in microseconds.

- `SigNum`

Table 22.323 This table provides type and value information for the Signum shared memory configuration parameter

Property	Value
Version (or later)	NDB 8.0.13
Type or units	unsigned
Default	[none]
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter was used formerly to override operating system signal numbers; in NDB 8.0, it is no longer used, and any setting for it is ignored.

22.3.3.13 Configuring NDB Cluster Send Buffer Parameters

The `NDB` kernel employs a unified send buffer whose memory is allocated dynamically from a pool shared by all transporters. This means that the size of the send buffer can be adjusted as necessary. Configuration of the unified send buffer can accomplished by setting the following parameters:

- **TotalSendBufferMemory.** This parameter can be set for all types of NDB Cluster nodes—that is, it can be set in the `[ndbd]`, `[mgm]`, and `[api]` (or `[mysql]`) sections of the `config.ini` file. It represents the total amount of memory (in bytes) to be allocated by each node for which it is set for use among all configured transporters. If set, its minimum is 256KB; the maximum is 4294967039.

  To be backward-compatible with existing configurations, this parameter takes as its default value the sum of the maximum send buffer sizes of all configured transporters, plus an additional 32KB (one page) per transporter. The maximum depends on the type of transporter, as shown in the following table:

  Table 22.324 Transporter types with maximum send buffer sizes

Transporter	Maximum Send Buffer Size (bytes)
TCP	`SendBufferMemory` (default = 2M)
SHM	20K

  This enables existing configurations to function in close to the same way as they did with NDB Cluster 6.3 and earlier, with the same amount of memory and send buffer space available to each transporter. However, memory that is unused by one transporter is not available to other transporters.

- **OverloadLimit.** This parameter is used in the `config.ini` file `[tcp]` section, and denotes the amount of unsent data (in bytes) that must be present in the send buffer before the connection is considered overloaded. When such an overload condition occurs, transactions that affect the overloaded connection fail with NDB API Error 1218 (`Send Buffers overloaded in NDB kernel`) until the overload status passes. The default value is 0, in which case the effective overload limit is calculated as `SendBufferMemory * 0.8` for a given connection. The maximum value for this parameter is 4G.

- **SendBufferMemory.** This value denotes a hard limit for the amount of memory that may be used by a single transporter out of the entire pool specified by `TotalSendBufferMemory`. However, the sum of `SendBufferMemory` for all configured transporters may be greater than the `TotalSendBufferMemory` that is set for a given node. This is a way to save memory when many nodes are in use, as long as the maximum amount of memory is never required by all transporters at the same time.

You can use the `ndbinfo.transporters` table to monitor send buffer memory usage, and to detect slowdown and overload conditions that can adversely affect performance.

22.3.4 Using High-Speed Interconnects with NDB Cluster

Even before design of `NDBCLUSTER` began in 1996, it was evident that one of the major problems to be encountered in building parallel databases would be communication between the nodes in the network. For this reason, `NDBCLUSTER` was designed from the very beginning to permit the use of a number of different data transport mechanisms. In this Manual, we use the term *transporter* for these.

The NDB Cluster codebase provides for four different transporters:

- *TCP/IP using 100 Mbps or gigabit Ethernet*, as discussed in Section 22.3.3.10, “NDB Cluster TCP/IP Connections”.

- *Direct (machine-to-machine) TCP/IP*; although this transporter uses the same TCP/IP protocol as mentioned in the previous item, it requires setting up the hardware differently and is configured differently as well. For this reason, it is considered a separate transport mechanism for NDB Cluster. See Section 22.3.3.11, “NDB Cluster TCP/IP Connections Using Direct Connections”, for details.

- *Shared memory (SHM)*. For more information about SHM, see Section 22.3.3.12, “NDB Cluster Shared-Memory Connections”.

- *Scalable Coherent Interface (SCI)*.

Note

Using SCI transporters in NDB Cluster requires specialized hardware, software, and MySQL binaries not available with NDB 8.0. See SCI Transport Connections in NDB Cluster.

Most users today employ TCP/IP over Ethernet because it is ubiquitous. TCP/IP is also by far the best-tested transporter for use with NDB Cluster.

We are working to make sure that communication with the `ndbd` process is made in “chunks” that are as large as possible because this benefits all types of data transmission.

22.4 NDB Cluster Programs

Using and managing an NDB Cluster requires several specialized programs, which we describe in this chapter. We discuss the purposes of these programs in an NDB Cluster, how to use the programs, and what startup options are available for each of them.

These programs include the NDB Cluster data, management, and SQL node processes (ndbd, ndbmtd, ndb_mgmd, and mysqld) and the management client (ndb_mgm).

Information about the program ndb_setup.py, used to start the NDB Cluster Auto-Installer, is also included in this section. You should be aware that Section 22.4.26, "ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster", contains information about the command-line client only; for information about using the GUI installer spawned by this program to configure and deploy an NDB Cluster, see The NDB Cluster Auto-Installer (NDB 7.5).

For information about using mysqld as an NDB Cluster process, see Section 22.5.4, "MySQL Server Usage for NDB Cluster".

Other NDB utility, diagnostic, and example programs are included with the NDB Cluster distribution. These include ndb_restore, ndb_show_tables, and ndb_config. These programs are also covered in this section.

The final portion of this section contains tables of options that are common to all the various NDB Cluster programs.

22.4.1 ndbd — The NDB Cluster Data Node Daemon

ndbd is the process that is used to handle all the data in tables using the NDB Cluster storage engine. This is the process that empowers a data node to accomplish distributed transaction handling, node recovery, checkpointing to disk, online backup, and related tasks.

In an NDB Cluster, a set of ndbd processes cooperate in handling data. These processes can execute on the same computer (host) or on different computers. The correspondences between data nodes and Cluster hosts is completely configurable.

The following table includes command options specific to the NDB Cluster data node program ndbd. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndbd), see Section 22.4.31, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 22.325 Command-line options for the ndbd program

Format	Description	Added, Deprecated, or Removed
--bind-address=name	Local bind address	All NDB 8.0 releases
--connect-delay=#	Time to wait between attempts to contact a management server, in seconds; 0 means do not wait between attempts	All NDB 8.0 releases
--connect-retries=#	Set the number of times to retry a connection before giving up; 0 means 1 attempt only (and no retries)	All NDB 8.0 releases
--connect-retry-delay=#	Time to wait between attempts to contact a management server, in seconds; 0 means do not wait between attempts	All NDB 8.0 releases
--daemon, -d	Start ndbd as daemon (default); override with --nodaemon	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
--foreground	Run ndbd in foreground, provided for debugging purposes (implies --nodaemon)	All NDB 8.0 releases
--initial	Perform initial start of ndbd, including cleaning the file system. Consult the documentation before using this option	All NDB 8.0 releases
--initial-start	Perform partial initial start (requires --nowait-nodes)	All NDB 8.0 releases
--install[=name]	Used to install the data node process as a Windows service. Does not apply on non-Windows platforms.	All NDB 8.0 releases
--logbuffer-size=#	Control size of log buffer. For use when debugging with many log messages being generated; default is sufficient for normal operations.	All NDB 8.0 releases
--nostart, -n	Don't start ndbd immediately; ndbd waits for command to start from ndb_mgmd	All NDB 8.0 releases
--nodaemon	Do not start ndbd as daemon; provided for testing purposes	All NDB 8.0 releases
--nowait-nodes=list	Do not wait for these data nodes to start (takes comma-separated list of node IDs). Also requires --ndb-nodeid to be used.	All NDB 8.0 releases
--remove[=name]	Used to remove a data node process that was previously installed as a Windows service. Does not apply on non-Windows platforms.	All NDB 8.0 releases
--verbose, -v	Causes the data log to write extra debugging information to the node log.	All NDB 8.0 releases

Note

All of these options also apply to the multithreaded version of this program (ndbmtd) and you may substitute “ndbmtd” for “ndbd” wherever the latter occurs in this section.

- --bind-address

Property	Value
Command-Line Format	--bind-address=name
Type	String
Default Value	

Causes ndbd to bind to a specific network interface (host name or IP address). This option has no default value.

- --connect-delay=#

Property	Value
Command-Line Format	--connect-delay=#
Deprecated	Yes
Type	Numeric
Default Value	5
Minimum Value	0
Maximum Value	3600

Determines the time to wait between attempts to contact a management server when starting (the number of attempts is controlled by the --connect-retries option). The default is 5 seconds.

This option is deprecated, and is subject to removal in a future release of NDB Cluster. Use --connect-retry-delay instead.

- --connect-retries=#

Property	Value
Command-Line Format	--connect-retries=#
Type	Numeric
Default Value	12
Minimum Value	0
Maximum Value	65535

Set the number of times to retry a connection before giving up; 0 means 1 attempt only (and no retries). The default is 12 attempts. The time to wait between attempts is controlled by the --connect-retry-delay option.

- --connect-retry-delay=#

Property	Value
Command-Line Format	--connect-retry-delay=#
Type	Numeric
Default Value	5
Minimum Value	0
Maximum Value	4294967295

Determines the time to wait between attempts to contact a management server when starting (the time between attempts is controlled by the --connect-retries option). The default is 5 seconds.

This option takes the place of the --connect-delay option, which is now deprecated and subject to removal in a future release of NDB Cluster.

- --daemon, -d

Property	Value
Command-Line Format	`--daemon`
Type	Boolean
Default Value	`TRUE`

Instructs `ndbd` or `ndbmtd` to execute as a daemon process. This is the default behavior. `--nodaemon` can be used to prevent the process from running as a daemon.

This option has no effect when running `ndbd` or `ndbmtd` on Windows platforms.

- `--foreground`

Property	Value
Command-Line Format	`--foreground`
Type	Boolean
Default Value	`FALSE`

Causes `ndbd` or `ndbmtd` to execute as a foreground process, primarily for debugging purposes. This option implies the `--nodaemon` option.

This option has no effect when running `ndbd` or `ndbmtd` on Windows platforms.

- `--initial`

Property	Value
Command-Line Format	`--initial`
Type	Boolean
Default Value	`FALSE`

Instructs `ndbd` to perform an initial start. An initial start erases any files created for recovery purposes by earlier instances of `ndbd`. It also re-creates recovery log files. On some operating systems, this process can take a substantial amount of time.

An `--initial` start is to be used *only* when starting the `ndbd` process under very special circumstances; this is because this option causes all files to be removed from the NDB Cluster file system and all redo log files to be re-created. These circumstances are listed here:

- When performing a software upgrade which has changed the contents of any files.
- When restarting the node with a new version of `ndbd`.
- As a measure of last resort when for some reason the node restart or system restart repeatedly fails. In this case, be aware that this node can no longer be used to restore data due to the destruction of the data files.

Warning

To avoid the possibility of eventual data loss, it is recommended that you *not* use the `--initial` option together with `StopOnError = 0`. Instead, set `StopOnError` to 0 in `config.ini` only after the cluster has been started, then restart the data nodes normally—that is, without the `--initial` option. See

the description of the StopOnError parameter for a detailed explanation of this issue. (Bug #24945638)

Use of this option prevents the StartPartialTimeout and StartPartitionedTimeout configuration parameters from having any effect.

Important

This option does *not* affect either of the following types of files:

- Backup files that have already been created by the affected node
- NDB Cluster Disk Data files (see Section 22.5.13, "NDB Cluster Disk Data Tables").

This option also has no effect on recovery of data by a data node that is just starting (or restarting) from data nodes that are already running. This recovery of data occurs automatically, and requires no user intervention in an NDB Cluster that is running normally.

It is permissible to use this option when starting the cluster for the very first time (that is, before any data node files have been created); however, it is *not* necessary to do so.

- `--initial-start`

Property	Value
Command-Line Format	`--initial-start`
Type	Boolean
Default Value	`FALSE`

This option is used when performing a partial initial start of the cluster. Each node should be started with this option, as well as `--nowait-nodes`.

Suppose that you have a 4-node cluster whose data nodes have the IDs 2, 3, 4, and 5, and you wish to perform a partial initial start using only nodes 2, 4, and 5—that is, omitting node 3:

```
shell> ndbd --ndb-nodeid=2 --nowait-nodes=3 --initial-start
shell> ndbd --ndb-nodeid=4 --nowait-nodes=3 --initial-start
shell> ndbd --ndb-nodeid=5 --nowait-nodes=3 --initial-start
```

When using this option, you must also specify the node ID for the data node being started with the `--ndb-nodeid` option.

Important

Do not confuse this option with the `--nowait-nodes` option for `ndb_mgmd`, which can be used to enable a cluster configured with multiple management servers to be started without all management servers being online.

- `--install[=name]`

Property	Value
Command-Line Format	`--install[=name]`
Platform Specific	Windows

Property	Value
Type	String
Default Value	ndbd

Causes ndbd to be installed as a Windows service. Optionally, you can specify a name for the service; if not set, the service name defaults to ndbd. Although it is preferable to specify other ndbd program options in a my.ini or my.cnf configuration file, it is possible to use together with --install. However, in such cases, the --install option must be specified first, before any other options are given, for the Windows service installation to succeed.

It is generally not advisable to use this option together with the --initial option, since this causes the data node file system to be wiped and rebuilt every time the service is stopped and started. Extreme care should also be taken if you intend to use any of the other ndbd options that affect the starting of data nodes—including --initial-start, --nostart, and --nowait-nodes—together with --install, and you should make absolutely certain you fully understand and allow for any possible consequences of doing so.

The --install option has no effect on non-Windows platforms.

- --logbuffer-size=#

Property	Value
Command-Line Format	--logbuffer-size=#
Type	Integer
Default Value	32768
Minimum Value	2048
Maximum Value	4294967295

Sets the size of the data node log buffer. When debugging with high amounts of extra logging, it is possible for the log buffer to run out of space if there are too many log messages, in which case some log messages can be lost. This should not occur during normal operations.

- --nodaemon

Property	Value
Command-Line Format	--nodaemon
Type	Boolean
Default Value	FALSE

Prevents ndbd or ndbmtd from executing as a daemon process. This option overrides the --daemon option. This is useful for redirecting output to the screen when debugging the binary.

The default behavior for ndbd and ndbmtd on Windows is to run in the foreground, making this option unnecessary on Windows platforms, where it has no effect.

- --nostart, -n

Property	Value
Command-Line Format	--nostart
Type	Boolean

Property	Value
Default Value	FALSE

Instructs ndbd not to start automatically. When this option is used, ndbd connects to the management server, obtains configuration data from it, and initializes communication objects. However, it does not actually start the execution engine until specifically requested to do so by the management server. This can be accomplished by issuing the proper START command in the management client (see Section 22.5.2, “Commands in the NDB Cluster Management Client”).

- --nowait-nodes=*node_id_1*[, *node_id_2*[, ...]]

Property	Value
Command-Line Format	--nowait-nodes=list
Type	String
Default Value	

This option takes a list of data nodes which for which the cluster will not wait for before starting.

This can be used to start the cluster in a partitioned state. For example, to start the cluster with only half of the data nodes (nodes 2, 3, 4, and 5) running in a 4-node cluster, you can start each ndbd process with --nowait-nodes=3,5. In this case, the cluster starts as soon as nodes 2 and 4 connect, and does *not* wait StartPartitionedTimeout milliseconds for nodes 3 and 5 to connect as it would otherwise.

If you wanted to start up the same cluster as in the previous example without one ndbd (say, for example, that the host machine for node 3 has suffered a hardware failure) then start nodes 2, 4, and 5 with --nowait-nodes=3. Then the cluster will start as soon as nodes 2, 4, and 5 connect and will not wait for node 3 to start.

- --remove[=*name*]

Property	Value
Command-Line Format	--remove[=name]
Platform Specific	Windows
Type	String
Default Value	ndbd

Causes an ndbd process that was previously installed as a Windows service to be removed. Optionally, you can specify a name for the service to be uninstalled; if not set, the service name defaults to ndbd.

The --remove option has no effect on non-Windows platforms.

- --verbose, -v

Causes extra debug output to be written to the node log.

You can also use NODELOG DEBUG ON and NODELOG DEBUG OFF to enable and disable this extra logging while the data node is running.

ndbd generates a set of log files which are placed in the directory specified by DataDir in the config.ini configuration file.

These log files are listed below. *node_id* is and represents the node's unique identifier. For example, ndb_2_error.log is the error log generated by the data node whose node ID is 2.

- ndb_*node_id*_error.log is a file containing records of all crashes which the referenced ndbd process has encountered. Each record in this file contains a brief error string and a reference to a trace file for this crash. A typical entry in this file might appear as shown here:

```
Date/Time: Saturday 30 July 2004 - 00:20:01
Type of error: error
Message: Internal program error (failed ndbrequire)
Fault ID: 2341
Problem data: DbtupFixAlloc.cpp
Object of reference: DBTUP (Line: 173)
ProgramName: NDB Kernel
ProcessID: 14909
TraceFile: ndb_2_trace.log.2
***EOM***
```

  Listings of possible ndbd exit codes and messages generated when a data node process shuts down prematurely can be found in Data Node Error Messages.

Important

The last entry in the error log file is not necessarily the newest one (nor is it likely to be). Entries in the error log are *not* listed in chronological order; rather, they correspond to the order of the trace files as determined in the ndb_*node_id*_trace.log.next file (see below). Error log entries are thus overwritten in a cyclical and not sequential fashion.

- ndb_*node_id*_trace.log.*trace_id* is a trace file describing exactly what happened just before the error occurred. This information is useful for analysis by the NDB Cluster development team.

  It is possible to configure the number of these trace files that will be created before old files are overwritten. *trace_id* is a number which is incremented for each successive trace file.

- ndb_*node_id*_trace.log.next is the file that keeps track of the next trace file number to be assigned.

- ndb_*node_id*_out.log is a file containing any data output by the ndbd process. This file is created only if ndbd is started as a daemon, which is the default behavior.

- ndb_*node_id*.pid is a file containing the process ID of the ndbd process when started as a daemon. It also functions as a lock file to avoid the starting of nodes with the same identifier.

- ndb_*node_id*_signal.log is a file used only in debug versions of ndbd, where it is possible to trace all incoming, outgoing, and internal messages with their data in the ndbd process.

It is recommended not to use a directory mounted through NFS because in some environments this can cause problems whereby the lock on the .pid file remains in effect even after the process has terminated.

To start ndbd, it may also be necessary to specify the host name of the management server and the port on which it is listening. Optionally, one may also specify the node ID that the process is to use.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

See Section 22.3.3.3, “NDB Cluster Connection Strings”, for additional information about this issue. Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster

Programs”, describes other command-line options which can be used with ndbd. For information about data node configuration parameters, see Section 22.3.3.6, “Defining NDB Cluster Data Nodes”.

When ndbd starts, it actually initiates two processes. The first of these is called the “angel process”; its only job is to discover when the execution process has been completed, and then to restart the ndbd process if it is configured to do so. Thus, if you attempt to kill ndbd using the Unix kill command, it is necessary to kill both processes, beginning with the angel process. The preferred method of terminating an ndbd process is to use the management client and stop the process from there.

The execution process uses one thread for reading, writing, and scanning data, as well as all other activities. This thread is implemented asynchronously so that it can easily handle thousands of concurrent actions. In addition, a watch-dog thread supervises the execution thread to make sure that it does not hang in an endless loop. A pool of threads handles file I/O, with each thread able to handle one open file. Threads can also be used for transporter connections by the transporters in the ndbd process. In a multi-processor system performing a large number of operations (including updates), the ndbd process can consume up to 2 CPUs if permitted to do so.

For a machine with many CPUs it is possible to use several ndbd processes which belong to different node groups; however, such a configuration is still considered experimental and is not supported for MySQL 8.0 in a production setting. See Section 22.1.7, “Known Limitations of NDB Cluster”.

22.4.2 ndbinfo_select_all — Select From ndbinfo Tables

ndbinfo_select_all is a client program that selects all rows and columns from one or more tables in the ndbinfo database

Not all ndbinfo tables available in the mysql client can be read by this program. In addition, ndbinfo_select_all can show information about some tables internal to ndbinfo which cannot be accessed using SQL, including the tables and columns metadata tables.

To select from one or more ndbinfo tables using ndbinfo_select_all, it is necessary to supply the names of the tables when invoking the program as shown here:

```
shell> ndbinfo_select_all table_name1  [table_name2] [...]
```

For example:

```
shell> ndbinfo_select_all logbuffers logspaces
== logbuffers ==
node_id log_type        log_id  log_part        total   used    high
5       0       0       0       33554432        262144  0
6       0       0       0       33554432        262144  0
7       0       0       0       33554432        262144  0
8       0       0       0       33554432        262144  0
== logspaces ==
node_id log_type        log_id  log_part        total   used    high
5       0       0       0       268435456       0       0
5       0       0       1       268435456       0       0
5       0       0       2       268435456       0       0
5       0       0       3       268435456       0       0
6       0       0       0       268435456       0       0
6       0       0       1       268435456       0       0
6       0       0       2       268435456       0       0
6       0       0       3       268435456       0       0
7       0       0       0       268435456       0       0
7       0       0       1       268435456       0       0
7       0       0       2       268435456       0       0
7       0       0       3       268435456       0       0
8       0       0       0       268435456       0       0
```

```
8       0       0       1       268435456       0       0
8       0       0       2       268435456       0       0
8       0       0       3       268435456       0       0
shell>
```

The following table includes options that are specific to ndbinfo_select_all. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndbinfo_select_all), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.326 Command-line options for the ndbinfo_select_all program

Format	Description	Added, Deprecated, or Removed
--delay=#	Set the delay in seconds between loops. Default is 5.	All NDB 8.0 releases
--loops=#, -l	Set the number of times to perform the select. Default is 1.	All NDB 8.0 releases
--database=db_name, -d	Name of the database where the table located.	All NDB 8.0 releases
--parallelism=#, -p	Set the degree of parallelism.	All NDB 8.0 releases

- --delay=seconds

Property	Value
Command-Line Format	--delay=#
Type	Numeric
Default Value	5
Minimum Value	0
Maximum Value	MAX_INT

This option sets the number of seconds to wait between executing loops. Has no effect if --loops is set to 0 or 1.

- --loops=number, -l *number*

Property	Value
Command-Line Format	--loops=#
Type	Numeric
Default Value	1
Minimum Value	0
Maximum Value	MAX_INT

This option sets the number of times to execute the select. Use --delay to set the time between loops.

22.4.3 ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded)

ndbmtd is a multithreaded version of ndbd, the process that is used to handle all the data in tables using the NDBCLUSTER storage engine. ndbmtd is intended for use on host computers having multiple CPU cores. Except where otherwise noted, ndbmtd functions in the same way as ndbd; therefore, in this section, we concentrate on the ways in which ndbmtd differs from ndbd, and you should consult Section 22.4.1, “ndbd — The NDB Cluster Data Node Daemon”, for additional information about running NDB Cluster data nodes that apply to both the single-threaded and multithreaded versions of the data node process.

Command-line options and configuration parameters used with ndbd also apply to ndbmtd. For more information about these options and parameters, see Section 22.4.1, “ndbd — The NDB Cluster Data Node Daemon”, and Section 22.3.3.6, “Defining NDB Cluster Data Nodes”, respectively.

ndbmtd is also file system-compatible with ndbd. In other words, a data node running ndbd can be stopped, the binary replaced with ndbmtd, and then restarted without any loss of data. (However, when doing this, you must make sure that MaxNoOfExecutionThreads is set to an apppriate value before restarting the node if you wish for ndbmtd to run in multithreaded fashion.) Similarly, an ndbmtd binary can be replaced with ndbd simply by stopping the node and then starting ndbd in place of the multithreaded binary. It is not necessary when switching between the two to start the data node binary using --initial.

Using ndbmtd differs from using ndbd in two key respects:

1. Because ndbmtd runs by default in single-threaded mode (that is, it behaves like ndbd), you must configure it to use multiple threads. This can be done by setting an appropriate value in the config.ini file for the MaxNoOfExecutionThreads configuration parameter or the ThreadConfig configuration parameter. Using MaxNoOfExecutionThreads is simpler, but ThreadConfig offers more flexibility. For more information about these configuration parameters and their use, see Multi-Threading Configuration Parameters (ndbmtd).

2. Trace files are generated by critical errors in ndbmtd processes in a somewhat different fashion from how these are generated by ndbd failures. These differences are discussed in more detail in the next few paragraphs.

Like ndbd, ndbmtd generates a set of log files which are placed in the directory specified by DataDir in the config.ini configuration file. Except for trace files, these are generated in the same way and have the same names as those generated by ndbd.

In the event of a critical error, ndbmtd generates trace files describing what happened just prior to the error' occurrence. These files, which can be found in the data node's DataDir, are useful for analysis of problems by the NDB Cluster Development and Support teams. One trace file is generated for each ndbmtd thread. The names of these files have the following pattern:

```
ndb_node_id_trace.log.trace_id_tthread_id,
```

In this pattern, *node_id* stands for the data node's unique node ID in the cluster, *trace_id* is a trace sequence number, and *thread_id* is the thread ID. For example, in the event of the failure of an ndbmtd process running as an NDB Cluster data node having the node ID 3 and with MaxNoOfExecutionThreads equal to 4, four trace files are generated in the data node's data directory. If the is the first time this node has failed, then these files are named ndb_3_trace.log.1_t1, ndb_3_trace.log.1_t2, ndb_3_trace.log.1_t3, and ndb_3_trace.log.1_t4. Internally, these trace files follow the same format as ndbd trace files.

The ndbd exit codes and messages that are generated when a data node process shuts down prematurely are also used by ndbmtd. See Data Node Error Messages, for a listing of these.

Note

It is possible to use ndbd and ndbmtd concurrently on different data nodes in the same NDB Cluster. However, such configurations have not been tested extensively; thus, we cannot recommend doing so in a production setting at this time.

22.4.4 ndb_mgmd — The NDB Cluster Management Server Daemon

The management server is the process that reads the cluster configuration file and distributes this information to all nodes in the cluster that request it. It also maintains a log of cluster activities. Management clients can connect to the management server and check the cluster's status.

The following table includes options that are specific to the NDB Cluster management server program ndb_mgmd. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_mgmd), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.327 Command-line options for the ndb_mgmd program

Format	Description	Added, Deprecated, or Removed
--bind-address=host	Local bind address	All NDB 8.0 releases
--config-cache[=TRUE|FALSE]	Enable the management server configuration cache; TRUE by default.	All NDB 8.0 releases
--config-file=file (>=), -f (>=)	Specify the cluster configuration file; in NDB-6.4.0 and later, needs --reload or --initial to override configuration cache if present	All NDB 8.0 releases
--configdir=directory, --config-dir=directory (>=7.0.8)	Specify the cluster management server's configuration cache directory	All NDB 8.0 releases
--daemon, -d	Run ndb_mgmd in daemon mode (default)	All NDB 8.0 releases
--initial	Causes the management server reload its configuration data from the configuration file, bypassing the configuration cache	All NDB 8.0 releases
--install[=name]	Used to install the management server process as a Windows service. Does not apply on non-Windows platforms.	All NDB 8.0 releases
--interactive	Run ndb_mgmd in interactive mode (not officially supported in production; for testing purposes only)	All NDB 8.0 releases
--log-name=name	A name to use when writing messages applying to this node in the cluster log.	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
--mycnf	Read cluster configuration data from the my.cnf file	All NDB 8.0 releases
--no-nodeid-checks	Do not provide any node id checks	All NDB 8.0 releases
--nodaemon	Do not run ndb_mgmd as a daemon	All NDB 8.0 releases
--nowait-nodes=list	Do not wait for these management nodes when starting this management server. Also requires --ndb-nodeid to be used.	All NDB 8.0 releases
--print-full-config, -P	Print full configuration and exit	All NDB 8.0 releases
--reload	Causes the management server to compare the configuration file with its configuration cache	All NDB 8.0 releases
--remove[=name]	Used to remove a management server process that was previously installed as a Windows service, optionally specifying the name of the service to be removed. Does not apply on non-Windows platforms.	All NDB 8.0 releases
--verbose, -v	Write additional information to the log.	All NDB 8.0 releases

- `--bind-address=host`

Property	Value
Command-Line Format	`--bind-address=host`
Type	String
Default Value	`[none]`

Causes the management server to bind to a specific network interface (host name or IP address). This option has no default value.

- `--config-cache`

Property	Value
Command-Line Format	`--config-cache[=TRUE|FALSE]`
Type	Boolean
Default Value	`TRUE`

This option, whose default value is `1` (or `TRUE`, or `ON`), can be used to disable the management server's configuration cache, so that it reads its configuration from `config.ini` every time it starts (see Section 22.3.3, “NDB Cluster Configuration Files”). You can do this by starting the `ndb_mgmd` process with any one of the following options:

- `--config-cache=0`
- `--config-cache=FALSE`
- `--config-cache=OFF`
- `--skip-config-cache`

Using one of the options just listed is effective only if the management server has no stored configuration at the time it is started. If the management server finds any configuration cache files, then the `--config-cache` option or the `--skip-config-cache` option is ignored. Therefore, to disable configuration caching, the option should be used the *first* time that the management server is started. Otherwise—that is, if you wish to disable configuration caching for a management server that has *already* created a configuration cache—you must stop the management server, delete any existing configuration cache files manually, then restart the management server with `--skip-config-cache` (or with `--config-cache` set equal to 0, `OFF`, or `FALSE`).

Configuration cache files are normally created in a directory named `mysql-cluster` under the installation directory (unless this location has been overridden using the `--configdir` option). Each time the management server updates its configuration data, it writes a new cache file. The files are named sequentially in order of creation using the following format:

```
ndb_node-id_config.bin.seq-number
```

`node-id` is the management server's node ID; *`seq-number`* is a sequence number, beginning with 1. For example, if the management server's node ID is 5, then the first three configuration cache files would, when they are created, be named `ndb_5_config.bin.1`, `ndb_5_config.bin.2`, and `ndb_5_config.bin.3`.

If your intent is to purge or reload the configuration cache without actually disabling caching, you should start `ndb_mgmd` with one of the options `--reload` or `--initial` instead of `--skip-config-cache`.

To re-enable the configuration cache, simply restart the management server, but without the `--config-cache` or `--skip-config-cache` option that was used previously to disable the configuration cache.

`ndb_mgmd` does not check for the configuration directory (`--configdir`) or attempts to create one when `--skip-config-cache` is used. (Bug #13428853)

- `--config-file=filename`, `-f filename`

Property	Value
Command-Line Format	`--config-file=file`
Type	File name
Default Value	`[none]`

Instructs the management server as to which file it should use for its configuration file. By default, the management server looks for a file named `config.ini` in the same directory as the `ndb_mgmd` executable; otherwise the file name and location must be specified explicitly.

This option has no default value, and is ignored unless the management server is forced to read the configuration file, either because `ndb_mgmd` was started with the `--reload` or `--initial` option, or because the management server could not find any configuration cache. This option is also read if

ndb_mgmd was started with --config-cache=OFF. See Section 22.3.3, “NDB Cluster Configuration Files”, for more information.

- --configdir=*dir_name*

Property	Value
Command-Line Format	--configdir=directory --config-dir=directory
Type	File name
Default Value	$INSTALLDIR/mysql-cluster

Specifies the cluster management server's configuration cache directory. --config-dir is an alias for this option.

- --daemon, -d

Property	Value
Command-Line Format	--daemon
Type	Boolean
Default Value	TRUE

Instructs ndb_mgmd to start as a daemon process. This is the default behavior.

This option has no effect when running ndb_mgmd on Windows platforms.

- --initial

Property	Value
Command-Line Format	--initial
Type	Boolean
Default Value	FALSE

Configuration data is cached internally, rather than being read from the cluster global configuration file each time the management server is started (see Section 22.3.3, “NDB Cluster Configuration Files”). Using the --initial option overrides this behavior, by forcing the management server to delete any existing cache files, and then to re-read the configuration data from the cluster configuration file and to build a new cache.

This differs in two ways from the --reload option. First, --reload forces the server to check the configuration file against the cache and reload its data only if the contents of the file are different from the cache. Second, --reload does not delete any existing cache files.

If ndb_mgmd is invoked with --initial but cannot find a global configuration file, the management server cannot start.

When a management server starts, it checks for another management server in the same NDB Cluster and tries to use the other management server's configuration data. This behavior has implications when performing a rolling restart of an NDB Cluster with multiple management nodes. See Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”, for more information.

When used together with the `--config-file` option, the cache is cleared only if the configuration file is actually found.

- `--install[=name]`

Property	Value
Command-Line Format	`--install[=name]`
Platform Specific	Windows
Type	String
Default Value	`ndb_mgmd`

Causes `ndb_mgmd` to be installed as a Windows service. Optionally, you can specify a name for the service; if not set, the service name defaults to `ndb_mgmd`. Although it is preferable to specify other `ndb_mgmd` program options in a `my.ini` or `my.cnf` configuration file, it is possible to use them together with `--install`. However, in such cases, the `--install` option must be specified first, before any other options are given, for the Windows service installation to succeed.

It is generally not advisable to use this option together with the `--initial` option, since this causes the configuration cache to be wiped and rebuilt every time the service is stopped and started. Care should also be taken if you intend to use any other `ndb_mgmd` options that affect the starting of the management server, and you should make absolutely certain you fully understand and allow for any possible consequences of doing so.

The `--install` option has no effect on non-Windows platforms.

- `--interactive`

Property	Value
Command-Line Format	`--interactive`
Type	Boolean
Default Value	`FALSE`

Starts `ndb_mgmd` in interactive mode; that is, an `ndb_mgm` client session is started as soon as the management server is running. This option does not start any other NDB Cluster nodes.

- `--log-name=name`

Property	Value
Command-Line Format	`--log-name=name`
Type	String
Default Value	`MgmtSrvr`

Provides a name to be used for this node in the cluster log.

- `--mycnf`

Property	Value
Command-Line Format	`--mycnf`
Type	Boolean

Property	Value
Default Value	FALSE

Read configuration data from the my.cnf file.

- --no-nodeid-checks

Property	Value
Command-Line Format	--no-nodeid-checks
Type	Boolean
Default Value	FALSE

Do not perform any checks of node IDs.

- --nodaemon

Property	Value
Command-Line Format	--nodaemon
Type	Boolean
Default Value	FALSE

Instructs ndb_mgmd not to start as a daemon process.

The default behavior for ndb_mgmd on Windows is to run in the foreground, making this option unnecessary on Windows platforms.

- --nowait-nodes

Property	Value
Command-Line Format	--nowait-nodes=list
Type	Numeric
Default Value	
Minimum Value	1
Maximum Value	255

When starting an NDB Cluster is configured with two management nodes, each management server normally checks to see whether the other ndb_mgmd is also operational and whether the other management server's configuration is identical to its own. However, it is sometimes desirable to start the cluster with only one management node (and perhaps to allow the other ndb_mgmd to be started later). This option causes the management node to bypass any checks for any other management nodes whose node IDs are passed to this option, permitting the cluster to start as though configured to use only the management node that was started.

For purposes of illustration, consider the following portion of a config.ini file (where we have omitted most of the configuration parameters that are not relevant to this example):

```
[ndbd]
NodeId = 1
HostName = 198.51.100.101
```

```
[ndbd]
NodeId = 2
HostName = 198.51.100.102

[ndbd]
NodeId = 3
HostName = 198.51.100.103

[ndbd]
NodeId = 4
HostName = 198.51.100.104

[ndb_mgmd]
NodeId = 10
HostName = 198.51.100.150

[ndb_mgmd]
NodeId = 11
HostName = 198.51.100.151

[api]
NodeId = 20
HostName = 198.51.100.200

[api]
NodeId = 21
HostName = 198.51.100.201
```

Assume that you wish to start this cluster using only the management server having node ID `10` and running on the host having the IP address 198.51.100.150. (Suppose, for example, that the host computer on which you intend to the other management server is temporarily unavailable due to a hardware failure, and you are waiting for it to be repaired.) To start the cluster in this way, use a command line on the machine at 198.51.100.150 to enter the following command:

```
shell> ndb_mgmd --ndb-nodeid=10 --nowait-nodes=11
```

As shown in the preceding example, when using `--nowait-nodes`, you must also use the `--ndb-nodeid` option to specify the node ID of this `ndb_mgmd` process.

You can then start each of the cluster's data nodes in the usual way. If you wish to start and use the second management server in addition to the first management server at a later time without restarting the data nodes, you must start each data node with a connection string that references both management servers, like this:

```
shell> ndbd -c 198.51.100.150,198.51.100.151
```

The same is true with regard to the connection string used with any `mysqld` processes that you wish to start as NDB Cluster SQL nodes connected to this cluster. See Section 22.3.3.3, “NDB Cluster Connection Strings”, for more information.

When used with `ndb_mgmd`, this option affects the behavior of the management node with regard to other management nodes only. Do not confuse it with the `--nowait-nodes` option used with `ndbd` or `ndbmtd` to permit a cluster to start with fewer than its full complement of data nodes; when used with data nodes, this option affects their behavior only with regard to other data nodes.

Multiple management node IDs may be passed to this option as a comma-separated list. Each node ID must be no less than 1 and no greater than 255. In practice, it is quite rare to use more than two management servers for the same NDB Cluster (or to have any need for doing so); in most cases you

need to pass to this option only the single node ID for the one management server that you do not wish to use when starting the cluster.

Note

When you later start the “missing” management server, its configuration must match that of the management server that is already in use by the cluster. Otherwise, it fails the configuration check performed by the existing management server, and does not start.

- `--print-full-config`, `-P`

Property	Value
Command-Line Format	`--print-full-config`
Type	Boolean
Default Value	`FALSE`

Shows extended information regarding the configuration of the cluster. With this option on the command line the `ndb_mgmd` process prints information about the cluster setup including an extensive list of the cluster configuration sections as well as parameters and their values. Normally used together with the `--config-file` (`-f`) option.

- `--reload`

Property	Value
Command-Line Format	`--reload`
Type	Boolean
Default Value	`FALSE`

NDB Cluster configuration data is stored internally rather than being read from the cluster global configuration file each time the management server is started (see Section 22.3.3, “NDB Cluster Configuration Files”). Using this option forces the management server to check its internal data store against the cluster configuration file and to reload the configuration if it finds that the configuration file does not match the cache. Existing configuration cache files are preserved, but not used.

This differs in two ways from the `--initial` option. First, `--initial` causes all cache files to be deleted. Second, `--initial` forces the management server to re-read the global configuration file and construct a new cache.

If the management server cannot find a global configuration file, then the `--reload` option is ignored.

When `--reload` is used, the management server must be able to communicate with data nodes and any other management servers in the cluster before it attempts to read the global configuration file; otherwise, the management server fails to start. This can happen due to changes in the networking environment, such as new IP addresses for nodes or an altered firewall configuration. In such cases, you must use `--initial` instead to force the exsiting cached configuration to be discarded and reloaded from the file. See Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”, for additional information.

- `--remove{=name]`

Property	Value
Command-Line Format	`--remove[=name]`
Platform Specific	Windows
Type	String
Default Value	`ndb_mgmd`

Remove a management server process that has been installed as a Windows service, optionally specifying the name of the service to be removed. Applies only to Windows platforms.

- `--verbose`, `-v`

Property	Value
Command-Line Format	`--verbose`
Type	Boolean
Default Value	`FALSE`

Remove a management server process that has been installed as a Windows service, optionally specifying the name of the service to be removed. Applies only to Windows platforms.

It is not strictly necessary to specify a connection string when starting the management server. However, if you are using more than one management server, a connection string should be provided and each node in the cluster should specify its node ID explicitly.

See Section 22.3.3.3, “NDB Cluster Connection Strings”, for information about using connection strings. Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”, describes other options for `ndb_mgmd`.

The following files are created or used by `ndb_mgmd` in its starting directory, and are placed in the `DataDir` as specified in the `config.ini` configuration file. In the list that follows, *`node_id`* is the unique node identifier.

- `config.ini` is the configuration file for the cluster as a whole. This file is created by the user and read by the management server. Section 22.3, “Configuration of NDB Cluster”, discusses how to set up this file.
- `ndb_`*`node_id`*`_cluster.log` is the cluster events log file. Examples of such events include checkpoint startup and completion, node startup events, node failures, and levels of memory usage. A complete listing of cluster events with descriptions may be found in Section 22.5, “Management of NDB Cluster”.

  By default, when the size of the cluster log reaches one million bytes, the file is renamed to `ndb_`*`node_id`*`_cluster.log.`*`seq_id`*, where *`seq_id`* is the sequence number of the cluster log file. (For example: If files with the sequence numbers 1, 2, and 3 already exist, the next log file is named using the number 4.) You can change the size and number of files, and other characteristics of the cluster log, using the `LogDestination` configuration parameter.
- `ndb_`*`node_id`*`_out.log` is the file used for `stdout` and `stderr` when running the management server as a daemon.
- `ndb_`*`node_id`*`.pid` is the process ID file used when running the management server as a daemon.

22.4.5 `ndb_mgm` — The NDB Cluster Management Client

The ndb_mgm management client process is actually not needed to run the cluster. Its value lies in providing a set of commands for checking the cluster's status, starting backups, and performing other administrative functions. The management client accesses the management server using a C API. Advanced users can also employ this API for programming dedicated management processes to perform tasks similar to those performed by ndb_mgm.

To start the management client, it is necessary to supply the host name and port number of the management server:

```
shell> ndb_mgm [host_name [port_num]]
```

For example:

```
shell> ndb_mgm ndb_mgmd.mysql.com 1186
```

The default host name and port number are localhost and 1186, respectively.

The following table includes options that are specific to the NDB Cluster management client program ndb_mgm. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_mgm), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.328 Command-line options for the ndb_mgm program

Format	Description	Added, Deprecated, or Removed
--try-reconnect=#, -t	Set the number of times to retry a connection before giving up; synonym for --connect-retries	All NDB 8.0 releases
--execute=name, -e	Execute command and exit	All NDB 8.0 releases

- --connect-retries=#

Property	Value
Command-Line Format	--connect-retries=#
Type	Numeric
Default Value	3
Minimum Value	0
Maximum Value	4294967295

This option specifies the number of times following the first attempt to retry a connection before giving up (the client always tries the connection at least once). The length of time to wait per attempt is set using --connect-retry-delay.

This option is synonymous with the --try-reconnect option, which is now deprecated.

The default for this option this option differs from its default when used with other NDB programs. See Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”, for more information.

- --execute=command, -e command

Property	Value
Command-Line Format	--execute=name

This option can be used to send a command to the NDB Cluster management client from the system shell. For example, either of the following is equivalent to executing SHOW in the management client:

```
shell> ndb_mgm -e "SHOW"

shell> ndb_mgm --execute="SHOW"
```

This is analogous to how the --execute or -e option works with the mysql command-line client. See Section 4.2.2.1, “Using Options on the Command Line”.

Note

If the management client command to be passed using this option contains any space characters, then the command *must* be enclosed in quotation marks. Either single or double quotation marks may be used. If the management client command contains no space characters, the quotation marks are optional.

- --try-reconnect=*number*

Property	Value
Command-Line Format	--try-reconnect=#
Deprecated	Yes
Type	Numeric
Type	Integer
Default Value	12
Default Value	3
Minimum Value	0
Maximum Value	4294967295

If the connection to the management server is broken, the node tries to reconnect to it every 5 seconds until it succeeds. By using this option, it is possible to limit the number of attempts to *number* before giving up and reporting an error instead.

This option is deprecated and subject to removal in a future release. Use --connect-retries, instead.

Additional information about using ndb_mgm can be found in Section 22.5.2, “Commands in the NDB Cluster Management Client”.

22.4.6 ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables

This tool can be used to check for and remove orphaned BLOB column parts from NDB tables, as well as to generate a file listing any orphaned parts. It is sometimes useful in diagnosing and repairing corrupted or damaged NDB tables containing BLOB or TEXT columns.

The basic syntax for ndb_blob_tool is shown here:

```
ndb_blob_tool [options] table [column, ...]
```

Unless you use the `--help` option, you must specify an action to be performed by including one or more of the options `--check-orphans`, `--delete-orphans`, or `--dump-file`. These options cause `ndb_blob_tool` to check for orphaned BLOB parts, remove any orphaned BLOB parts, and generate a dump file listing orphaned BLOB parts, respectively, and are described in more detail later in this section.

You must also specify the name of a table when invoking `ndb_blob_tool`. In addition, you can optionally follow the table name with the (comma-separated) names of one or more `BLOB` or `TEXT` columns from that table. If no columns are listed, the tool works on all of the table's `BLOB` and `TEXT` columns. If you need to specify a database, use the `--database` (`-d`) option.

The `--verbose` option provides additional information in the output about the tool's progress.

The following table includes options that are specific to `ndb_blob_tool`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_blob_tool`), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.329 Command-line options for the ndb_blob_tool program

Format	Description	Added, Deprecated, or Removed
`--check-orphans`	Check for orphan blob parts	All NDB 8.0 releases
`--database=db_name`, `-d`	Database to find the table in.	All NDB 8.0 releases
`--delete-orphans`	Delete orphan blob parts	All NDB 8.0 releases
`--dump-file=file`	Write orphan keys to specified file	All NDB 8.0 releases
`--verbose`, `-v`	Verbose output	All NDB 8.0 releases

- `--check-orphans`

Property	Value
Command-Line Format	`--check-orphans`
Type	Boolean
Default Value	`FALSE`

Check for orphaned BLOB parts in NDB Cluster tables.

- `--database=db_name`, `-d`

Property	Value
Command-Line Format	`--database=db_name`
Type	String
Default Value	`[none]`

Specify the database to find the table in.

- `--delete-orphans`

Property	Value
Command-Line Format	`--delete-orphans`
Type	Boolean
Default Value	`FALSE`

Remove orphaned BLOB parts from NDB Cluster tables.

- `--dump-file=file`

Property	Value
Command-Line Format	`--dump-file=file`
Type	File name
Default Value	`[none]`

Writes a list of orphaned BLOB column parts to *`file`*. The information written to the file includes the table key and BLOB part number for each orphaned BLOB part.

- `--verbose`

Property	Value
Command-Line Format	`--verbose`
Type	Boolean
Default Value	`FALSE`

Provide extra information in the tool's output regarding its progress.

Example

First we create an `NDB` table in the `test` database, using the `CREATE TABLE` statement shown here:

```
USE test;

CREATE TABLE btest (
    c0 BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    c1 TEXT,
    c2 BLOB
)   ENGINE=NDB;
```

Then we insert a few rows into this table, using a series of statements similar to this one:

```
INSERT INTO btest VALUES (NULL, 'x', REPEAT('x', 1000));
```

When run with `--check-orphans` against this table, `ndb_blob_tool` generates the following output:

```
shell> ndb_blob_tool --check-orphans --verbose -d test btest
connected
processing 2 blobs
processing blob #0 c1 NDB$BLOB_19_1
NDB$BLOB_19_1: nextResult: res=1
total parts: 0
orphan parts: 0
processing blob #1 c2 NDB$BLOB_19_2
NDB$BLOB_19_2: nextResult: res=0
```

```
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=1
total parts: 10
orphan parts: 0
disconnected

NDBT_ProgramExit: 0 - OK
```

The tool reports that there are no NDB BLOB column parts associated with column c1, even though c1 is a TEXT column. This is due to the fact that, in an NDB table, only the first 256 bytes of a BLOB or TEXT column value are stored inline, and only the excess, if any, is stored separately; thus, if there are no values using more than 256 bytes in a given column of one of these types, no BLOB column parts are created by NDB for this column. See Section 11.8, “Data Type Storage Requirements”, for more information.

22.4.7 ndb_config — Extract NDB Cluster Configuration Information

This tool extracts current configuration information for data nodes, SQL nodes, and API nodes from one of a number of sources: an NDB Cluster management node, or its config.ini or my.cnf file. By default, the management node is the source for the configuration data; to override the default, execute ndb_config with the --config-file or --mycnf option. It is also possible to use a data node as the source by specifying its node ID with --config_from_node=*node_id*.

ndb_config can also provide an offline dump of all configuration parameters which can be used, along with their default, maximum, and minimum values and other information. The dump can be produced in either text or XML format; for more information, see the discussion of the --configinfo and --xml options later in this section).

You can filter the results by section (DB, SYSTEM, or CONNECTIONS) using one of the options --nodes, --system, or --connections.

The following table includes options that are specific to ndb_config. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_config), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.330 Command-line options for the ndb_config program

Format	Description	Added, Deprecated, or Removed
--config-file=file_name	Set the path to config.ini file	All NDB 8.0 releases
--config_from_node=#	Obtain configuration data from the node having this ID (must be a data node).	All NDB 8.0 releases
--configinfo	Dumps information about all NDB configuration parameters in text format with default, maximum, and minimum values. Use with --xml to obtain XML output.	All NDB 8.0 releases
--connections	Print connections information ([tcp], [tcp default], [shm], or [shm default] sections of cluster	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
	configuration file) only. Cannot be used with --system or --nodes.	
--diff-default	Print only configuration parameters that have non-default values	All NDB 8.0 releases
--fields=string, -f	Field separator	All NDB 8.0 releases
--host=name	Specify host	All NDB 8.0 releases
--mycnf	Read configuration data from my.cnf file	All NDB 8.0 releases
--nodeid, --id	Get configuration of node with this ID	All NDB 8.0 releases
--nodes	Print node information ([ndbd] or [ndbd default] section of cluster configuration file) only. Cannot be used with --system or --connections.	All NDB 8.0 releases
-c	Short form for --ndb-connectstring	All NDB 8.0 releases
--query=string, -q	One or more query options (attributes)	All NDB 8.0 releases
--query-all, -a	Dumps all parameters and values to a single comma-delimited string.	All NDB 8.0 releases
--rows=string, -r	Row separator	All NDB 8.0 releases
--system	Print SYSTEM section information only (see ndb_config --configinfo output). Cannot be used with --nodes or --connections.	All NDB 8.0 releases
--type=name	Specify node type	All NDB 8.0 releases
--configinfo --xml	Use --xml with --configinfo to obtain a dump of all NDB configuration parameters in XML format with default, maximum, and minimum values.	All NDB 8.0 releases

- `--configinfo`

  The `--configinfo` option causes `ndb_config` to dump a list of each NDB Cluster configuration parameter supported by the NDB Cluster distribution of which `ndb_config` is a part, including the following information:

  - A brief description of each parameter's purpose, effects, and usage
  - The section of the `config.ini` file where the parameter may be used

- The parameter's data type or unit of measurement
- Where applicable, the parameter's default, minimum, and maximum values
- NDB Cluster release version and build information

By default, this output is in text format. Part of this output is shown here:

```
shell> ndb_config --configinfo

****** SYSTEM ******

Name (String)
Name of system (NDB Cluster)
MANDATORY

PrimaryMGMNode (Non-negative Integer)
Node id of Primary ndb_mgmd(MGM) node
Default: 0 (Min: 0, Max: 4294967039)

ConfigGenerationNumber (Non-negative Integer)
Configuration generation number
Default: 0 (Min: 0, Max: 4294967039)

****** DB ******

MaxNoOfSubscriptions (Non-negative Integer)
Max no of subscriptions (default 0 == MaxNoOfTables)
Default: 0 (Min: 0, Max: 4294967039)

MaxNoOfSubscribers (Non-negative Integer)
Max no of subscribers (default 0 == 2 * MaxNoOfTables)
Default: 0 (Min: 0, Max: 4294967039)

…
```

Use this option together with the `--xml` option to obtain output in XML format.

- `--config-file=path-to-file`

Property	Value
Command-Line Format	`--config-file=file_name`
Type	File name
Default Value	

Gives the path to the management server's configuration file (`config.ini`). This may be a relative or absolute path. If the management node resides on a different host from the one on which `ndb_config` is invoked, then an absolute path must be used.

- `--config_from_node=#`

Property	Value
Command-Line Format	`--config-from-node=#`
Type	Numeric
Default Value	`none`
Minimum Value	`1`

Property	Value
Maximum Value	48

Obtain the cluster's configuration data from the data node that has this ID.

If the node having this ID is not a data node, ndb_config fails with an error. (To obtain configuration data from the management node instead, simply omit this option.)

- --connections

Property	Value
Command-Line Format	--connections
Type	Boolean
Default Value	FALSE

Tells ndb_config to print CONNECTIONS information only—that is, information about parameters found in the [tcp], [tcp default], [shm], or [shm default] sections of the cluster configuration file (see Section 22.3.3.10, “NDB Cluster TCP/IP Connections”, and Section 22.3.3.12, “NDB Cluster Shared-Memory Connections”, for more information).

This option is mutually exclusive with --nodes and --system; only one of these 3 options can be used.

- --diff-default

Property	Value
Command-Line Format	--diff-default
Type	Boolean
Default Value	FALSE

Print only configuration parameters that have non-default values.

- --fields=*delimiter*, -f *delimiter*

Property	Value
Command-Line Format	--fields=string
Type	String
Default Value	

Specifies a *delimiter* string used to separate the fields in the result. The default is , (the comma character).

Note

If the *delimiter* contains spaces or escapes (such as \n for the linefeed character), then it must be quoted.

- --host=*hostname*

Property	Value
Command-Line Format	--host=name

Property	Value
Type	String
Default Value	

Specifies the host name of the node for which configuration information is to be obtained.

Note

While the hostname `localhost` usually resolves to the IP address `127.0.0.1`, this may not necessarily be true for all operating platforms and configurations. This means that it is possible, when `localhost` is used in `config.ini`, for `ndb_config --host=localhost` to fail if `ndb_config` is run on a different host where `localhost` resolves to a different address (for example, on some versions of SUSE Linux, this is `127.0.0.2`). In general, for best results, you should use numeric IP addresses for all NDB Cluster configuration values relating to hosts, or verify that all NDB Cluster hosts handle `localhost` in the same fashion.

- `--mycnf`

Property	Value
Command-Line Format	`--mycnf`
Type	Boolean
Default Value	`FALSE`

Read configuration data from the `my.cnf` file.

- `--ndb-connectstring=connection_string`, `-c connection_string`

Property	Value

Command-Line Format	`--ndb-connectstring=connectstring`
	
	`--connect-string=connectstring`

Type	String
Default Value	`localhost:1186`

Specifies the connection string to use in connecting to the management server. The format for the connection string is the same as described in Section 22.3.3.3, “NDB Cluster Connection Strings”, and defaults to `localhost:1186`.

- `--nodeid=node_id`

Property	Value
Command-Line Format	`--ndb-nodeid=#`
Type	Numeric
Default Value	`0`

Specify the node ID of the node for which configuration information is to be obtained.

- `--nodes`

Property	Value
Command-Line Format	--nodes
Type	Boolean
Default Value	FALSE

Tells ndb_config to print information relating only to parameters defined in an [ndbd] or [ndbd default] section of the cluster configuration file (see Section 22.3.3.6, "Defining NDB Cluster Data Nodes").

This option is mutually exclusive with --connections and --system; only one of these 3 options can be used.

- --query=*query-options*, -q *query-options*

Property	Value
Command-Line Format	--query=string
Type	String
Default Value	

This is a comma-delimited list of *query options*—that is, a list of one or more node attributes to be returned. These include nodeid (node ID), type (node type—that is, ndbd, mysqld, or ndb_mgmd), and any configuration parameters whose values are to be obtained.

For example, --query=nodeid,type,datamemory,datadir returns the node ID, node type, DataMemory, and DataDir for each node.

Note

If a given parameter is not applicable to a certain type of node, than an empty string is returned for the corresponding value. See the examples later in this section for more information.

- --query-all, -a

Property	Value
Command-Line Format	--query-all
Type	String
Default Value	

Returns a comma-delimited list of all query options (node attributes; note that this list is a single string.

- --rows=*separator*, -r *separator*

Property	Value
Command-Line Format	--rows=string
Type	String
Default Value	

Specifies a *separator* string used to separate the rows in the result. The default is a space character.

Note

If the *separator* contains spaces or escapes (such as \n for the linefeed character), then it must be quoted.

- --system

Property	Value
Command-Line Format	--system
Type	Boolean
Default Value	FALSE

Tells ndb_config to print SYSTEM information only. This consists of system variables that cannot be changed at run time; thus, there is no corresponding section of the cluster configuration file for them. They can be seen (prefixed with ****** SYSTEM ******) in the output of ndb_config --configinfo.

This option is mutually exclusive with --nodes and --connections; only one of these 3 options can be used.

- --type=*node_type*

Property	Value
Command-Line Format	--type=name
Type	Enumeration
Default Value	[none]
Valid Values	ndbd mysqld ndb_mgmd

Filters results so that only configuration values applying to nodes of the specified *node_type* (ndbd, mysqld, or ndb_mgmd) are returned.

- --usage, --help, or -?

Property	Value
Command-Line Format	--help --usage

Causes ndb_config to print a list of available options, and then exit.

- --version, -V

Property	Value
Command-Line Format	--version

Causes ndb_config to print a version information string, and then exit.

- --configinfo --xml

Property	Value
Command-Line Format	`--configinfo --xml`
Type	Boolean
Default Value	`false`

Cause `ndb_config --configinfo` to provide output as XML by adding this option. A portion of such output is shown in this example:

```
shell> ndb_config --configinfo --xml

<configvariables protocolversion="1" ndbversionstring="5.7.28-ndb-7.5.16"
                    ndbversion="460032" ndbversionmajor="7" ndbversionminor="5"
                    ndbversionbuild="0">
  <section name="SYSTEM">
    <param name="Name" comment="Name of system (NDB Cluster)" type="string"
              mandatory="true"/>
    <param name="PrimaryMGMNode" comment="Node id of Primary ndb_mgmd(MGM) node"
              type="unsigned" default="0" min="0" max="4294967039"/>
    <param name="ConfigGenerationNumber" comment="Configuration generation number"
              type="unsigned" default="0" min="0" max="4294967039"/>
  </section>
  <section name="MYSQLD" primarykeys="NodeId">
    <param name="wan" comment="Use WAN TCP setting as default" type="bool"
              default="false"/>
    <param name="HostName" comment="Name of computer for this node"
              type="string" default=""/>
    <param name="Id" comment="NodeId" type="unsigned" mandatory="true"
              min="1" max="255" deprecated="true"/>
    <param name="NodeId" comment="Number identifying application node (mysqld(API))"
              type="unsigned" mandatory="true" min="1" max="255"/>
    <param name="ExecuteOnComputer" comment="HostName" type="string"
              deprecated="true"/>

    …

  </section>

  …

</configvariables>
```

Note

Normally, the XML output produced by `ndb_config --configinfo --xml` is formatted using one line per element; we have added extra whitespace in the previous example, as well as the next one, for reasons of legibility. This should not make any difference to applications using this output, since most XML processors either ignore nonessential whitespace as a matter of course, or can be instructed to do so.

The XML output also indicates when changing a given parameter requires that data nodes be restarted using the `--initial` option. This is shown by the presence of an `initial="true"` attribute in the corresponding `<param>` element. In addition, the restart type (`system` or `node`) is also shown; if a given parameter requires a system restart, this is indicated by the presence of a `restart="system"` attribute in the corresponding `<param>` element. For example, changing the value set for the `Diskless` parameter requires a system initial restart, as shown here (with the `restart` and `initial` attributes highlighted for visibility):

```
<param name="Diskless" comment="Run wo/ disk" type="bool" default="false"
         restart="system" initial="true"/>
```

Currently, no `initial` attribute is included in the XML output for `<param>` elements corresponding to parameters which do not require initial restarts; in other words, `initial="false"` is the default, and the value `false` should be assumed if the attribute is not present. Similarly, the default restart type is `node` (that is, an online or “rolling” restart of the cluster), but the `restart` attribute is included only if the restart type is `system` (meaning that all cluster nodes must be shut down at the same time, then restarted).

Deprecated parameters are indicated in the XML output by the `deprecated` attribute, as shown here:

```
<param name="NoOfDiskPagesToDiskAfterRestartACC" comment="DiskCheckpointSpeed"
       type="unsigned" default="20" min="1" max="4294967039" deprecated="true"/>
```

In such cases, the `comment` refers to one or more parameters that supersede the deprecated parameter. Similarly to `initial`, the `deprecated` attribute is indicated only when the parameter is deprecated, with `deprecated="true"`, and does not appear at all for parameters which are not deprecated. (Bug #21127135)

Beginning with NDB 7.5.0, parameters that are required are indicated with `mandatory="true"`, as shown here:

```
<param name="NodeId"
         comment="Number identifying application node (mysqld(API))"
         type="unsigned" mandatory="true" min="1" max="255"/>
```

In much the same way that the `initial` or `deprecated` attribute is displayed only for a parameter that requires an intial restart or that is deprecated, the `mandatory` attribute is included only if the given parameter is actually required.

Important

The `--xml` option can be used only with the `--configinfo` option. Using `--xml` without `--configinfo` fails with an error.

Unlike the options used with this program to obtain current configuration data, `--configinfo` and `--xml` use information obtained from the NDB Cluster sources when `ndb_config` was compiled. For this reason, no connection to a running NDB Cluster or access to a `config.ini` or `my.cnf` file is required for these two options.

Combining other `ndb_config` options (such as `--query` or `--type`) with `--configinfo` (with or without the `--xml` option is not supported. Currently, if you attempt to do so, the usual result is that all other options besides `--configinfo` or `--xml` are simply ignored. *However, this behavior is not guaranteed and is subject to change at any time*. In addition, since `ndb_config`, when used with the `--configinfo` option, does not access the NDB Cluster or read any files, trying to specify additional options such as `--ndb-connectstring` or `--config-file` with `--configinfo` serves no purpose.

Examples

1. To obtain the node ID and type of each node in the cluster:

```
shell> ./ndb_config --query=nodeid,type --fields=':' --rows='\n'
1:ndbd
```

```
2:ndbd
3:ndbd
4:ndbd
5:ndb_mgmd
6:mysqld
7:mysqld
8:mysqld
9:mysqld
```

In this example, we used the `--fields` options to separate the ID and type of each node with a colon character (`:`), and the `--rows` options to place the values for each node on a new line in the output.

2. To produce a connection string that can be used by data, SQL, and API nodes to connect to the management server:

```
shell> ./ndb_config --config-file=usr/local/mysql/cluster-data/config.ini \
--query=hostname,portnumber --fields=: --rows=, --type=ndb_mgmd
198.51.100.179:1186
```

3. This invocation of `ndb_config` checks only data nodes (using the `--type` option), and shows the values for each node's ID and host name, as well as the values set for its `DataMemory` and `DataDir` parameters:

```
shell> ./ndb_config --type=ndbd --query=nodeid,host,datamemory,datadir -f ' : ' -r '\n'
1 : 198.51.100.193 : 83886080 : /usr/local/mysql/cluster-data
2 : 198.51.100.112 : 83886080 : /usr/local/mysql/cluster-data
3 : 198.51.100.176 : 83886080 : /usr/local/mysql/cluster-data
4 : 198.51.100.119 : 83886080 : /usr/local/mysql/cluster-data
```

In this example, we used the short options `-f` and `-r` for setting the field delimiter and row separator, respectively, as well as the short option `-q` to pass a list of parameters to be obtained.

4. To exclude results from any host except one in particular, use the `--host` option:

```
shell> ./ndb_config --host=198.51.100.176 -f : -r '\n' -q id,type
3:ndbd
5:ndb_mgmd
```

In this example, we also used the short form `-q` to determine the attributes to be queried.

Similarly, you can limit results to a node with a specific ID using the `--nodeid` option.

22.4.8 ndb_delete_all — Delete All Rows from an NDB Table

`ndb_delete_all` deletes all rows from the given `NDB` table. In some cases, this can be much faster than `DELETE` or even `TRUNCATE TABLE`.

Usage

```
ndb_delete_all -c connection_string tbl_name -d db_name
```

This deletes all rows from the table named *`tbl_name`* in the database named *`db_name`*. It is exactly equivalent to executing `TRUNCATE db_name.tbl_name` in MySQL.

The following table includes options that are specific to `ndb_delete_all`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_delete_all`), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.331 Command-line options for the ndb_delete_all program

Format	Description	Added, Deprecated, or Removed
`--database=dbname`, `-d`	Name of the database in which the table is found	All NDB 8.0 releases
`--transactional`, `-t`	Perform the delete in a single transaction (may run out of operations)	All NDB 8.0 releases
`--tupscan`	Run tup scan	All NDB 8.0 releases
`--diskscan`	Run disk scan	All NDB 8.0 releases

- `--transactional`, `-t`

  Use of this option causes the delete operation to be performed as a single transaction.

  Warning

  With very large tables, using this option may cause the number of operations available to the cluster to be exceeded.

Prior to NDB 8.0.18, this program printed `NDBT_ProgramExit - status` upon completion of its run, due to an unnecessary dependency on the `NDBT` testing library. This dependency is removed as of NDB 8.0.18, eliminating the extraneous output.

22.4.9 ndb_desc — Describe NDB Tables

`ndb_desc` provides a detailed description of one or more `NDB` tables.

Usage

```
ndb_desc -c connection_string tbl_name -d db_name [options]

ndb_desc -c connection_string index_name -d db_name -t tbl_name
```

Additional options that can be used with `ndb_desc` are listed later in this section.

Sample Output

MySQL table creation and population statements:

```
USE test;

CREATE TABLE fish (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL,
    length_mm INT(11) NOT NULL,
    weight_gm INT(11) NOT NULL,

    PRIMARY KEY pk (id),
    UNIQUE KEY uk (name)
) ENGINE=NDB;

INSERT INTO fish VALUES
    (NULL, 'guppy', 35, 2), (NULL, 'tuna', 2500, 150000),
    (NULL, 'shark', 3000, 110000), (NULL, 'manta ray', 1500, 50000),
    (NULL, 'grouper', 900, 125000), (NULL ,'puffer', 250, 2500);
```

Output from ndb_desc:

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 2
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 337
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 2
FragmentCount: 2
PartitionBalance: FOR_RP_BY_LDM
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options:
HashMap: DEFAULT-HASHMAP-3840-2
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY DYNAMIC
length_mm Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC
weight_gm Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC
-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
uk(name) - OrderedIndex
uk$unique(name) - UniqueHashIndex
-- Per partition info --
Partition       Row count       Commit count    Frag fixed memory       Frag varsized memory    Extent_space
0               2               2               32768                   32768                   0
1               4               4               32768                   32768                   0


NDBT_ProgramExit: 0 - OK
```

Information about multiple tables can be obtained in a single invocation of ndb_desc by using their names, separated by spaces. All of the tables must be in the same database.

You can obtain additional information about a specific index using the --table (short form: -t) option and supplying the name of the index as the first argument to ndb_desc, as shown here:

```
shell> ./ndb_desc uk -d test -t fish
-- uk --
Version: 2
Base table: fish
Number of attributes: 1
Logging: 0
Index type: OrderedIndex
Index status: Retrieved
-- Attributes --
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
-- IndexTable 10/uk --
Version: 2
Fragment type: FragUndefined
K Value: 6
```

```
Min load factor: 78
Max load factor: 80
Temporary table: yes
Number of attributes: 2
Number of primary keys: 1
Length of frm data: 0
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 2
ForceVarPart: 0
PartitionCount: 2
FragmentCount: 2
FragmentCountType: ONE_PER_LDM_PER_NODE
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options:
-- Attributes --
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
NDB$TNODE Unsigned [64] PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
-- Indexes --
PRIMARY KEY(NDB$TNODE) - UniqueHashIndex

NDBT_ProgramExit: 0 - OK
```

When an index is specified in this way, the `--extra-partition-info` and `--extra-node-info` options have no effect.

The `Version` column in the output contains the table's schema object version. For information about interpreting this value, see NDB Schema Object Versions.

Three of the table properties that can be set using `NDB_TABLE` comments embedded in `CREATE TABLE` and `ALTER TABLE` statements are also visible in `ndb_desc` output. The table's `FRAGMENT_COUNT_TYPE` is always shown in the `FragmentCountType` column. `READ_ONLY` and `FULLY_REPLICATED`, if set to 1, are shown in the `Table options` column. You can see this after executing the following `ALTER TABLE` statement in the `mysql` client:

```
mysql> ALTER TABLE fish COMMENT='NDB_TABLE=READ_ONLY=1,FULLY_REPLICATED=1';
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
+---------+------+-----------------------------------------------------------------------------------------------
| Level   | Code | Message
+---------+------+-----------------------------------------------------------------------------------------------
| Warning | 1296 | Got error 4503 'Table property is FRAGMENT_COUNT_TYPE=ONE_PER_LDM_PER_NODE but not in c
+---------+------+-----------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

The warning is issued because `READ_ONLY=1` requires that the table's fragment count type is (or be set to) `ONE_PER_LDM_PER_NODE_GROUP`; `NDB` sets this automatically in such cases. You can check that the `ALTER TABLE` statement has the desired effect using `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE fish\G
*************************** 1. row ***************************
       Table: fish
Create Table: CREATE TABLE `fish` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL,
  `length_mm` int(11) NOT NULL,
  `weight_gm` int(11) NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `uk` (`name`)
```

```
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
COMMENT='NDB_TABLE=READ_BACKUP=1,FULLY_REPLICATED=1'
1 row in set (0.01 sec)
```

Because FRAGMENT_COUNT_TYPE was not set explicitly, its value is not shown in the comment text printed by SHOW CREATE TABLE. ndb_desc, however, displays the updated value for this attribute. The Table options column shows the binary properties just enabled. You can see this in the output shown here (emphasized text):

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 4
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 380
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 1
FragmentCount: 1
FragmentCountType: ONE_PER_LDM_PER_NODE_GROUP
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options: readbackup, fullyreplicated
HashMap: DEFAULT-HASHMAP-3840-1
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY DYNAMIC
length_mm Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC
weight_gm Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC
-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
uk(name) - OrderedIndex
uk$unique(name) - UniqueHashIndex
-- Per partition info --
Partition       Row count       Commit count    Frag fixed memory       Frag varsized memory    Extent_space

NDBT_ProgramExit: 0 - OK
```

For more information about these table properties, see Section 13.1.20.11, “Setting NDB_TABLE Options”.

The Extent_space and Free extent_space columns are applicable only to NDB tables having columns on disk; for tables having only in-memory columns, these columns always contain the value 0.

To illustrate their use, we modify the previous example. First, we must create the necessary Disk Data objects, as shown here:

```
CREATE LOGFILE GROUP lg_1
    ADD UNDOFILE 'undo_1.log'
    INITIAL_SIZE 16M
    UNDO_BUFFER_SIZE 2M
    ENGINE NDB;

ALTER LOGFILE GROUP lg_1
```

```
    ADD UNDOFILE 'undo_2.log'
    INITIAL_SIZE 12M
    ENGINE NDB;

CREATE TABLESPACE ts_1
    ADD DATAFILE 'data_1.dat'
    USE LOGFILE GROUP lg_1
    INITIAL_SIZE 32M
    ENGINE NDB;

ALTER TABLESPACE ts_1
    ADD DATAFILE 'data_2.dat'
    INITIAL_SIZE 48M
    ENGINE NDB;
```

(For more information on the statements just shown and the objects created by them, see Section 22.5.13.1, “NDB Cluster Disk Data Objects”, as well as Section 13.1.16, “CREATE LOGFILE GROUP Syntax”, and Section 13.1.21, “CREATE TABLESPACE Syntax”.)

Now we can create and populate a version of the `fish` table that stores 2 of its columns on disk (deleting the previous version of the table first, if it already exists):

```
CREATE TABLE fish (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL,
    length_mm INT(11) NOT NULL,
    weight_gm INT(11) NOT NULL,

    PRIMARY KEY pk (id),
    UNIQUE KEY uk (name)
) TABLESPACE ts_1 STORAGE DISK
ENGINE=NDB;

INSERT INTO fish VALUES
    (NULL, 'guppy', 35, 2), (NULL, 'tuna', 2500, 150000),
    (NULL, 'shark', 3000, 110000), (NULL, 'manta ray', 1500, 50000),
    (NULL, 'grouper', 900, 125000), (NULL ,'puffer', 250, 2500);
```

When run against this version of the table, `ndb_desc` displays the following output:

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 1
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 346
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 2
FragmentCount: 2
FragmentCountType: ONE_PER_LDM_PER_NODE
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options:
HashMap: DEFAULT-HASHMAP-3840-2
```

```
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
length_mm Int NOT NULL AT=FIXED ST=DISK
weight_gm Int NOT NULL AT=FIXED ST=DISK
-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
uk(name) - OrderedIndex
uk$unique(name) - UniqueHashIndex
-- Per partition info --
Partition       Row count       Commit count    Frag fixed memory       Frag varsized memory    Extent_space
0               2               2               32768                   32768                   1048576
1               4               4               32768                   32768                   1048576


NDBT_ProgramExit: 0 - OK
```

This means that 1048576 bytes are allocated from the tablespace for this table on each partition, of which 1044440 bytes remain free for additional storage. In other words, 1048576 - 1044440 = 4136 bytes per partition is currently being used to store the data from this table's disk-based columns. The number of bytes shown as `Free extent_space` is available for storing on-disk column data from the `fish` table only; for this reason, it is not visible when selecting from the `INFORMATION_SCHEMA.FILES` table.

For fully replicated tables, `ndb_desc` shows only the nodes holding primary partition fragment replicas; nodes with copy fragment replicas (only) are ignored. Beginning with NDB 7.5.4, you can obtain such information, using the `mysql` client, from the `table_distribution_status`, `table_fragments`, `table_info`, and `table_replicas` tables in the `ndbinfo` database.

The following table includes options that are specific to `ndb_desc`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_desc`), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.332 Command-line options for the ndb_desc program

Format	Description	Added, Deprecated, or Removed
`--blob-info`, `-b`	Include partition information for BLOB tables in output. Requires that the -p option also be used	All NDB 8.0 releases
`--database=dbname`, `-d`	Name of database containing table	All NDB 8.0 releases
`--extra-node-info`, `-n`	Include partition-to-data-node mappings in output. Requires that the -p option also be used	All NDB 8.0 releases
`--extra-partition-info`, `-p`	Display information about partitions	All NDB 8.0 releases
`--retries=#`, `-r`	Number of times to retry the connection (once per second)	All NDB 8.0 releases
`--table=tbl_name`, `-t`	Specify the table in which to find an index. When this option is used, -p and -n have no effect and are ignored.	All NDB 8.0 releases
`--unqualified`,	Use unqualified table names	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
-u		

- --blob-info, -b

  Include information about subordinate BLOB and TEXT columns.

  Use of this option also requires the use of the --extra-partition-info (-p) option.

- --database=*db_name*, -d

  Specify the database in which the table should be found.

- --extra-node-info, -n

  Include information about the mappings between table partitions and the data nodes upon which they reside. This information can be useful for verifying distribution awareness mechanisms and supporting more efficient application access to the data stored in NDB Cluster.

  Use of this option also requires the use of the --extra-partition-info (-p) option.

- --extra-partition-info, -p

  Print additional information about the table's partitions.

- --retries=#, -r

  Try to connect this many times before giving up. One connect attempt is made per second.

- --table=*tbl_name*, -t

  Specify the table in which to look for an index.

- --unqualified, -u

  Use unqualified table names.

In NDB 7.5.3 and later, table indexes listed in the output are ordered by ID. Previously, this was not deterministic and could vary between platforms. (Bug #81763, Bug #23547742)

22.4.10 ndb_drop_index — Drop Index from an NDB Table

ndb_drop_index drops the specified index from an NDB table. *It is recommended that you use this utility only as an example for writing NDB API applications*—see the Warning later in this section for details.

Usage

```
ndb_drop_index -c connection_string table_name index -d db_name
```

The statement shown above drops the index named *index* from the *table* in the *database*.

The following table includes options that are specific to ndb_drop_index. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_drop_index), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.333 Command-line options for the ndb_drop_index program

Format	Description	Added, Deprecated, or Removed
`--database=dbname`, `-d`	Name of the database in which the table is found	All NDB 8.0 releases

Warning

Operations performed on Cluster table indexes using the NDB API are not visible to MySQL and make the table unusable by a MySQL server. If you use this program to drop an index, then try to access the table from an SQL node, an error results, as shown here:

```
shell> ./ndb_drop_index -c localhost dogs ix -d ctest1
Dropping index dogs/idx...OK

NDBT_ProgramExit: 0 - OK

shell> ./mysql -u jon -p ctest1
Enter password: *******
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 5.7.28-ndb-7.5.16

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW TABLES;
+------------------+
| Tables_in_ctest1 |
+------------------+
| a                |
| bt1              |
| bt2              |
| dogs             |
| employees        |
| fish             |
+------------------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM dogs;
ERROR 1296 (HY000): Got error 4243 'Index not found' from NDBCLUSTER
```

In such a case, your *only* option for making the table available to MySQL again is to drop the table and re-create it. You can use either the SQL statement`DROP TABLE` or the `ndb_drop_table` utility (see Section 22.4.11, “`ndb_drop_table` — Drop an NDB Table”) to drop the table.

22.4.11 `ndb_drop_table` — Drop an NDB Table

`ndb_drop_table` drops the specified `NDB` table. (If you try to use this on a table created with a storage engine other than `NDB`, the attempt fails with the error `723: No such table exists`.) This operation is extremely fast; in some cases, it can be an order of magnitude faster than using a MySQL `DROP TABLE` statement on an `NDB` table.

Usage

```
ndb_drop_table -c connection_string tbl_name -d db_name
```

The following table includes options that are specific to ndb_drop_table. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_drop_table), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.334 Command-line options for the ndb_drop_table program

Format	Description	Added, Deprecated, or Removed
--database=dbname, -d	Name of the database in which the table is found	All NDB 8.0 releases

Prior to NDB 8.0.17, an NDB table dropped using this utility persisted in the MySQL data dictionary but could not be dropped using DROP TABLE in the mysql client. In NDB 8.0.17 and later, such “orphan” tables can be dropped using DROP TABLE. (Bug #29125206, Bug #93672)

22.4.12 ndb_error_reporter — NDB Error-Reporting Utility

ndb_error_reporter creates an archive from data node and management node log files that can be used to help diagnose bugs or other problems with a cluster. *It is highly recommended that you make use of this utility when filing reports of bugs in NDB Cluster*.

The following table includes command options specific to the NDB Cluster program ndb_error_reporter. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_error_reporter), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.335 Command-line options for the ndb_error_reporter program

Format	Description	Added, Deprecated, or Removed
--connection-timeout=timeout	Number of seconds to wait when connecting to nodes before timing out.	All NDB 8.0 releases
--dry-scp	Disable scp with remote hosts; used only for testing.	All NDB 8.0 releases
--fs	Include file system data in error report; can use a large amount of disk space	All NDB 8.0 releases
--skip-nodegroup=nodegroup_id	Skip all nodes in the node group having this ID.	All NDB 8.0 releases

Usage

```
ndb_error_reporter path/to/config-file [username] [options]
```

This utility is intended for use on a management node host, and requires the path to the management host configuration file (usually named config.ini). Optionally, you can supply the name of a user that is able to access the cluster's data nodes using SSH, to copy the data node log files. ndb_error_reporter then includes all of these files in archive that is created in the same directory in which it is run. The archive is named ndb_error_report_*YYYYMMDDhhmmss*.tar.bz2, where *YYYYMMDDhhmmss* is a datetime string.

ndb_error_reporter also accepts the options listed here:

- --connection-timeout=*timeout*

Property	Value
Command-Line Format	--connection-timeout=timeout
Type	Integer
Default Value	0

Wait this many seconds when trying to connect to nodes before timing out.

- --dry-scp

Property	Value
Command-Line Format	--dry-scp
Type	Boolean
Default Value	TRUE

Run ndb_error_reporter without using scp from remote hosts. Used for testing only.

- --fs

Property	Value
Command-Line Format	--fs
Type	Boolean
Default Value	FALSE

Copy the data node file systems to the management host and include them in the archive.

Because data node file systems can be extremely large, even after being compressed, we ask that you please do *not* send archives created using this option to Oracle unless you are specifically requested to do so.

- --skip-nodegroup=*nodegroup_id*

Property	Value
Command-Line Format	--connection-timeout=timeout
Type	Integer
Default Value	0

Skip all nodes belong to the node group having the supplied node group ID.

22.4.13 ndb_import — Import CSV Data Into NDB

ndb_import imports CSV-formatted data, such as that produced by mysqldump --tab, directly into NDB using the NDB API. ndb_import requires a connection to an NDB management server (ndb_mgmd) to function; it does not require a connection to a MySQL Server.

Usage

```
ndb_import db_name file_name options
```

ndb_import requires two arguments. *db_name* is the name of the database where the table into which to import the data is found; *file_name* is the name of the CSV file from which to read the data; this must include the path to this file if it is not in the current directory. The name of the file must match that of the table; the file's extension, if any, is not taken into consideration. Options supported by ndb_import include those for specifying field separators, escapes, and line terminators, and are described later in this section. ndb_import must be able to connect to an NDB Cluster management server; for this reason, there must be an unused [api] slot in the cluster config.ini file.

To duplicate an existing table that uses a different storage engine, such as InnoDB, as an NDB table, use the mysql client to perform a SELECT INTO OUTFILE statement to export the existing table to a CSV file, then to execute a CREATE TABLE LIKE statement to create a new table having the same structure as the existing table, then perform ALTER TABLE ... ENGINE=NDB on the new table; after this, from the system shell, invoke ndb_import to load the data into the new NDB table. For example, an existing InnoDB table named myinnodb_table in a database named myinnodb can be exported into an NDB table named myndb_table in a database named myndb as shown here, assuming that you are already logged in as a MySQL user with the appropriate privileges:

1. In the mysql client:

   ```
   mysql> USE myinnodb;

   mysql> SELECT * INTO OUTFILE '/tmp/myndb_table.csv'
        >  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
        >  LINES TERMINATED BY '\n'
        >  FROM myinnodbtable;

   mysql> CREATE DATABASE myndb;

   mysql> USE myndb;

   mysql> CREATE TABLE myndb_table LIKE myinnodb.myinnodb_table;

   mysql> ALTER TABLE myndb_table ENGINE=NDB;

   mysql> EXIT;
   Bye
   shell>
   ```

   Once the target database and table have been created, a running mysqld is no longer required. You can stop it using mysqladmin shutdown or another method before proceeding, if you wish.

2. In the system shell:

   ```
   # if you are not already in the MySQL bin directory:
   shell> cd path-to-mysql-bin-dir

   shell> ndb_import myndb /tmp/myndb_table.csv --fields-optionally-enclosed-by='"' \
       --fields-terminated-by="," --fields-escaped-by='\\'
   ```

   The output should resemble what is shown here:

   ```
   job-1 import myndb.myndb_table from /tmp/myndb_table.csv
   job-1 [running] import myndb.myndb_table from /tmp/myndb_table.csv
   job-1 [success] import myndb.myndb_table from /tmp/myndb_table.csv
   job-1 imported 19984 rows in 0h0m9s at 2277 rows/s
   jobs summary: defined: 1 run: 1 with success: 1 with failure: 0
   shell>
   ```

The following table includes options that are specific to ndb_import. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_import), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.336 Command-line options for the ndb_import program

Format	Description	Added, Deprecated, or Removed
--abort-on-error	Dump core on any fatal error; used for debugging	All NDB 8.0 releases
--ai-increment=#	For table with hidden PK, specify autoincrement increment. See mysqld	All NDB 8.0 releases
--ai-offset=#	For table with hidden PK, specify autoincrement offset. See mysqld	All NDB 8.0 releases
--ai-prefetch-sz=#	For table with hidden PK, specify number of autoincrement values that are prefetched. See mysqld	All NDB 8.0 releases
--connections=#	Number of cluster connections to create	All NDB 8.0 releases
--continue	When job fails, continue to next job	All NDB 8.0 releases
--db-workers=#	Number of threads, per data node, executing database operations	All NDB 8.0 releases
--errins-type=name	Error insert type, for testing purposes; use "list" to obtain all possible values	All NDB 8.0 releases
--errins-delay=#	Error insert delay in milliseconds; random variation is added	All NDB 8.0 releases
--fields-enclosed-by=char	Same as FIELDS ENCLOSED BY option for LOAD DATA statements. For CSV input this is same as using --fields-optionally-enclosed-by	All NDB 8.0 releases
--fields-escaped-by=name	Same as FIELDS ESCAPED BY option for LOAD DATA statements	All NDB 8.0 releases
--fields-optionally-enclosed-by=char	Same as FIELDS OPTIONALLY ENCLOSED BY option for LOAD DATA statements	All NDB 8.0 releases
--fields-terminated-by=char	Same as FIELDS TERMINATED BY option for LOAD DATA statements.	All NDB 8.0 releases
--idlesleep=#	Number of milliseconds to sleep waiting for more to do	All NDB 8.0 releases
--idlespin=#	Number of times to re-try before idlesleep	All NDB 8.0 releases
--ignore-lines=#	Ignore first # lines in input file. Used to skip a non-data header.	All NDB 8.0 releases
--input-type=name	Input type: random or csv	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
--input-workers=#	Number of threads processing input. Must be 2 or more if --input-type is csv.	All NDB 8.0 releases
--keep-state	Preserve state files	All NDB 8.0 releases
--lines-terminated-by=name	Same as LINES TERMINATED BY option for LOAD DATA statements	All NDB 8.0 releases
--log-level=#	Set internal logging level; for debugging and development	All NDB 8.0 releases
--max-rows=#	Import only this number of input data rows; default is 0, which imports all rows	All NDB 8.0 releases
--monitor=#	Periodically print status of running job if something has changed (status, rejected rows, temporary errors). Value 0 disables. Value 1 prints any change seen. Higher values reduce status printing exponentially up to some pre-defined limit.	All NDB 8.0 releases
--no-asynch	Run database operations as batches, in single transactions	All NDB 8.0 releases
--no-hint	Do not use distribution key hint to select data node (TC)	All NDB 8.0 releases
--opbatch=#	A db execution batch is a set of transactions and operations sent to NDB kernel. This option limits NDB operations (including blob operations) in a db execution batch. Therefore it also limits number of asynch transactions. Value 0 is not valid	All NDB 8.0 releases
--opbytes=#	Limit bytes in execution batch (default 0 = no limit)	All NDB 8.0 releases
--output-type=name	Output type: ndb is default, null used for testing	All NDB 8.0 releases
--output-workers=#	Number of threads processing output or relaying database operations	All NDB 8.0 releases
--pagesize=#	Align I/O buffers to given size	All NDB 8.0 releases
--pagecnt=#	Size of I/O buffers as multiple of page size. CSV input worker allocates a double-sized buffer	All NDB 8.0 releases
--polltimeout=#	Timeout per poll for completed asynchonous transactions; polling continues until all polls are completed, or error occurs	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
--rejects=#	Limit number of rejected rows (rows with permanent error) in data load. Default is 0 which means that any rejected row causes a fatal error. The row exceeding the limit is also added to *.rej	All NDB 8.0 releases
--resume	If job aborted (temporary error, user interrupt), resume with rows not yet processed	All NDB 8.0 releases
--rowbatch=#	Limit rows in row queues (default 0 = no limit); must be 1 or more if --input-type is random	All NDB 8.0 releases
--rowbytes=#	Limit bytes in row queues (0 = no limit)	All NDB 8.0 releases
--state-dir=name	Where to write state files; currect directory is default	All NDB 8.0 releases
--stats	Save performance and statistics information in *.sto and *.stt files	All NDB 8.0 releases
--tempdelay=#	Number of milliseconds to sleep between temporary errors	All NDB 8.0 releases
--temperrors=#	Number of times a transaction can fail due to a temporary error, per execution batch; 0 means any temporary error is fatal. Such errors do not cause any rows to be written to .rej file	All NDB 8.0 releases
--verbose=#, -v	Enable verbose output	All NDB 8.0 releases

- --abort-on-error

Property	Value
Command-Line Format	--abort-on-error
Type	Boolean
Default Value	FALSE

Dump core on any fatal error; used for debugging only.

- --ai-increment=#

Property	Value
Command-Line Format	--ai-increment=#
Type	Integer
Default Value	1

Property	Value
Minimum Value	1
Maximum Value	4294967295

For a table with a hidden primary key, specify the autoincrement increment, like the auto_increment_increment system variable does in the MySQL Server.

- --ai-offset=#

Property	Value
Command-Line Format	--ai-offset=#
Type	Integer
Default Value	1
Minimum Value	1
Maximum Value	4294967295

For a table with hidden primary key, specify the autoincrement offset. Similar to the auto_increment_offset system variable.

- --ai-prefetch-sz=#

Property	Value
Command-Line Format	--ai-prefetch-sz=#
Type	Integer
Default Value	1024
Minimum Value	1
Maximum Value	4294967295

For a table with a hidden primary key, specify the number of autoincrement values that are prefetched. Behaves like the ndb_autoincrement_prefetch_sz system variable does in the MySQL Server.

- --connections=#

Property	Value
Command-Line Format	--connections=#
Type	Integer
Default Value	1
Minimum Value	1
Maximum Value	4294967295

Number of cluster connections to create.

- --continue

Property	Value
Command-Line Format	--continue
Type	Boolean

Property	Value
Default Value	FALSE

When a job fails, continue to the next job.

- --db-workers=#

Property	Value
Command-Line Format	--db-workers=#
Type	Integer
Default Value	4
Minimum Value	1
Maximum Value	4294967295

Number of threads, per data node, executing database operations.

- --errins-type=*name*

Property	Value
Command-Line Format	--errins-type=name
Type	Enumeration
Default Value	[none]
Valid Values	stopjob stopall sighup sigint list

Error insert type; use list as the *name* value to obtain all possible values. This option is used for testing purposes only.

- --errins-delay=#

Property	Value
Command-Line Format	--errins-delay=#
Type	Integer
Default Value	1000
Minimum Value	0
Maximum Value	4294967295

Error insert delay in milliseconds; random variation is added. This option is used for testing purposes only.

- --fields-enclosed-by=*char*

Property	Value
Command-Line Format	`--fields-enclosed-by=char`
Type	String
Default Value	`[none]`

This works in the same way as the `FIELDS ENCLOSED BY` option does for the `LOAD DATA` statement, specifying a character to be interpeted as quoting field values. For CSV input, this is the same as `--fields-optionally-enclosed-by`.

- `--fields-escaped-by=`*`name`*

Property	Value
Command-Line Format	`--fields-escaped-by=name`
Type	String
Default Value	`\`

Specify an escape character in the same way as the `FIELDS ESCAPED BY` option does for the SQL `LOAD DATA` statement.

- `--fields-optionally-enclosed-by=`*`char`*

Property	Value
Command-Line Format	`--fields-optionally-enclosed-by=char`
Type	String
Default Value	`[none]`

This works in the same way as the `FIELDS OPTIONALLY ENCLOSED BY` option does for the `LOAD DATA` statement, specifying a character to be interpeted as optionally quoting field values. For CSV input, this is the same as `--fields-enclosed-by`.

- `--fields-terminated-by=`*`char`*

Property	Value
Command-Line Format	`--fields-terminated-by=char`
Type	String
Default Value	`\t`

This works in the same way as the `FIELDS TERMINATED BY` option does for the `LOAD DATA` statement, specifying a character to be interpeted as the field separator.

- `--idlesleep=`*`#`*

Property	Value
Command-Line Format	`--idlesleep=#`
Type	Integer
Default Value	`1`
Minimum Value	`1`

Property	Value
Maximum Value	4294967295

Number of milliseconds to sleep waiting for more work to perform.

- --idlespin=#

Property	Value
Command-Line Format	--idlespin=#
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	4294967295

Number of times to retry before sleeping.

- --ignore-lines=#

Property	Value
Command-Line Format	--ignore-lines=#
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	4294967295

Cause ndb_import to ignore the first # lines of the input file. This can be employed to skip a file header that does not contain any data.

- --input-type=*name*

Property	Value
Command-Line Format	--input-type=name
Type	Enumeration
Default Value	csv
Valid Values	random csv

Set the type of input type. The default is csv; random is intended for testing purposes only. .

- --input-workers=#

Property	Value
Command-Line Format	--input-workers=#
Type	Integer
Default Value	4
Minimum Value	1

Property	Value
Maximum Value	4294967295

Set the number of threads processing input.

- `--keep-state`

Property	Value
Command-Line Format	`--keep-state`
Type	Boolean
Default Value	`false`

By default, ndb_import removes all state files (except non-empty `*.rej` files) when it completes a job. Specify this option (nor argument is required) to force the program to retain all state files instead.

- `--lines-terminated-by=`*`name`*

Property	Value
Command-Line Format	`--lines-terminated-by=name`
Type	String
Default Value	`\n`

This works in the same way as the `LINES TERMINATED BY` option does for the `LOAD DATA` statement, specifying a character to be interpeted as end-of-line.

- `--log-level=#`

Property	Value
Command-Line Format	`--log-level=#`
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2`

Performs internal logging at the given level. This option is intended primarily for internal and development use.

In debug builds of NDB only, the logging level can be set using this option to a maximum of 4.

- `--max-rows=#`

Property	Value
Command-Line Format	`--max-rows=#`
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

Import only this number of input data rows; the default is 0, which imports all rows.

- `--monitor=#`

Property	Value
Command-Line Format	`--monitor=#`
Type	Integer
Default Value	`2`
Minimum Value	`0`
Maximum Value	`4294967295`

Periodically print the status of a running job if something has changed (status, rejected rows, temporary errors). Set to 0 to disable this reporting. Setting to 1 prints any change that is seen. Higher values reduce the frequency of this status reporting.

- `--no-asynch`

Property	Value
Command-Line Format	`--no-asynch`
Type	Boolean
Default Value	`FALSE`

Run database operations as batches, in single transactions.

- `--no-hint`

Property	Value
Command-Line Format	`--no-hint`
Type	Boolean
Default Value	`FALSE`

Do not use distribution key hinting to select a data node.

- `--opbatch=#`

Property	Value
Command-Line Format	`--opbatch=#`
Type	Integer
Default Value	`256`
Minimum Value	`1`
Maximum Value	`4294967295`

Set a limit on the number of operations (including blob operations), and thus the number of asynchronous transactions, per execution batch.

- `--opbytes=#`

Property	Value
Command-Line Format	`--opbytes=#`
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

Set a limit on the number of bytes per execution batch. Use 0 for no limit.

- `--output-type=`*`name`*

Property	Value
Command-Line Format	`--output-type=name`
Type	Enumeration
Default Value	`ndb`
Valid Values	`null`

Set the output type. `ndb` is the default. `null` is used only for testing.

- `--output-workers=#`

Property	Value
Command-Line Format	`--output-workers=#`
Type	Integer
Default Value	`2`
Minimum Value	`1`
Maximum Value	`4294967295`

Set the number of threads processing output or relaying database operations.

- `--pagesize=#`

Property	Value
Command-Line Format	`--pagesize=#`
Type	Integer
Default Value	`4096`
Minimum Value	`1`
Maximum Value	`4294967295`

Align I/O buffers to the given size.

- `--pagecnt=#`

Property	Value
Command-Line Format	`--pagecnt=#`
Type	Integer

Property	Value
Default Value	64
Minimum Value	1
Maximum Value	4294967295

Set the size of I/O buffers as multiple of page size. The CSV input worker allocates buffer that is doubled in size.

- --polltimeout=#

Property	Value
Command-Line Format	--polltimeout=#
Type	Integer
Default Value	1000
Minimum Value	1
Maximum Value	4294967295

Set a timeout per poll for completed asynchonous transactions; polling continues until all polls are completed, or until an error occurs.

- --rejects=#

Property	Value
Command-Line Format	--rejects=#
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	4294967295

Limit the number of rejected rows (rows with permanent errors) in the data load. The default is 0, which means that any rejected row causes a fatal error. Any rows causing the limit to be exceeded are added to the .rej file.

The limit imposed by this option is effective for the duration of the current run. A run restarted using --resume is considered a “new” run for this purpose.

- --resume

Property	Value
Command-Line Format	--resume
Type	Boolean
Default Value	FALSE

If a job is aborted (due to a temporary db error or when interrupted by the user), resume with any rows not yet processed.

- --rowbatch=#

Property	Value
Command-Line Format	`--rowbatch=#`
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

Set a limit on the number of rows per row queue. Use 0 for no limit.

- `--rowbytes=#`

Property	Value
Command-Line Format	`--rowbytes=#`
Type	Integer
Default Value	`262144`
Minimum Value	`0`
Maximum Value	`4294967295`

Set a limit on the number of bytes per row queue. Use 0 for no limit.

- `--stats`

Property	Value
Command-Line Format	`--stats`
Type	Boolean
Default Value	`false`

Save information about options related to performance and other internal statistics in files named `*.sto` and `*.stt`. These files are always kept on successful completion (even if `--keep-state` is not also specified).

- `--state-dir=`*`name`*

Property	Value
Command-Line Format	`--state-dir=name`
Type	String
Default Value	`.`

Where to write the state files (*`tbl_name`*`.map`, *`tbl_name`*`.rej`, *`tbl_name`*`.res`, and *`tbl_name`*`.stt`) produced by a run of the program; the default is the current directory.

- `--tempdelay=#`

Property	Value
Command-Line Format	`--tempdelay=#`
Type	Integer

Property	Value
Default Value	10
Minimum Value	0
Maximum Value	4294967295

Number of milliseconds to sleep between temporary errors.

- `--temperrors=#`

Property	Value
Command-Line Format	--temperrors=#
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	4294967295

Number of times a transaction can fail due to a temporary error, per execution batch. The default is 0, which means that any temporary error is fatal. Temporary errors do not cause any rows to be added to the `.rej` file.

- `--verbose`, `-v`

Property	Value
Command-Line Format	--verbose
Type	Boolean
Default Value	false

Enable verbose output.

As with `LOAD DATA`, options for field and line formatting much match those used to create the CSV file, whether this was done using `SELECT INTO ... OUTFILE`, or by some other means. There is no equivalent to the `LOAD DATA` statement `STARTING WITH` option.

`ndb_import` was added in NDB 7.6.2.

22.4.14 ndb_index_stat — NDB Index Statistics Utility

`ndb_index_stat` provides per-fragment statistical information about indexes on `NDB` tables. This includes cache version and age, number of index entries per partition, and memory consumption by indexes.

Usage

To obtain basic index statistics about a given `NDB` table, invoke `ndb_index_stat` as shown here, with the name of the table as the first argument and the name of the database containing this table specified immediately following it, using the `--database` (`-d`) option:

```
ndb_index_stat table -d database
```

In this example, we use `ndb_index_stat` to obtain such information about an `NDB` table named `mytable` in the `test` database:

```
shell> ndb_index_stat -d test mytable
table:City index:PRIMARY fragCount:2
sampleVersion:3 loadTime:1399585986 sampleCount:1994 keyBytes:7976
query cache: valid:1 sampleCount:1994 totalBytes:27916
times in ms: save: 7.133 sort: 1.974 sort per sample: 0.000

NDBT_ProgramExit: 0 - OK
```

`sampleVersion` is the version number of the cache from which the statistics data is taken. Running `ndb_index_stat` with the `--update` option causes sampleVersion to be incremented.

`loadTime` shows when the cache was last updated. This is expressed as seconds since the Unix Epoch.

`sampleCount` is the number of index entries found per partition. You can estimate the total number of entries by multiplying this by the number of fragments (shown as `fragCount`).

`sampleCount` can be compared with the cardinality of `SHOW INDEX` or `INFORMATION_SCHEMA.STATISTICS`, although the latter two provide a view of the table as a whole, while `ndb_index_stat` provides a per-fragment average.

`keyBytes` is the number of bytes used by the index. In this example, the primary key is an integer, which requires four bytes for each index, so `keyBytes` can be calculated in this case as shown here:

```
    keyBytes = sampleCount * (4 bytes per index) = 1994 * 4 = 7976
```

This information can also be obtained using the corresponding column definitions from `INFORMATION_SCHEMA.COLUMNS` (this requires a MySQL Server and a MySQL client application).

`totalBytes` is the total memory consumed by all indexes on the table, in bytes.

Timings shown in the preceding examples are specific to each invocation of `ndb_index_stat`.

The `--verbose` option provides some additional output, as shown here:

```
shell> ndb_index_stat -d test mytable --verbose
random seed 1337010518
connected
loop 1 of 1
table:mytable index:PRIMARY fragCount:4
sampleVersion:2 loadTime:1336751773 sampleCount:0 keyBytes:0
read stats
query cache created
query cache: valid:1 sampleCount:0 totalBytes:0
times in ms: save: 20.766 sort: 0.001
disconnected

NDBT_ProgramExit: 0 - OK

shell>
```

If the only output from the program is `NDBT_ProgramExit: 0 - OK`, this may indicate that no statistics yet exist. To force them to be created (or updated if they already exist), invoke `ndb_index_stat` with the `--update` option, or execute `ANALYZE TABLE` on the table in the `mysql` client.

Options

The following table includes options that are specific to the NDB Cluster `ndb_index_stat` utility. Additional descriptions are listed following the table. For options common to most NDB Cluster programs (including `ndb_index_stat`), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.337 Command-line options for the ndb_index_stat program

Format	Description	Added, Deprecated, or Removed
`--database=name`, `-d`	Name of the database containing the table.	All NDB 8.0 releases
`--delete`	Delete index statistics for the given table, stopping any auto-update previously configured.	All NDB 8.0 releases
`--update`	Update index statistics for the given table, restarting any auto-update previously configured.	All NDB 8.0 releases
`--dump`	Print the query cache.	All NDB 8.0 releases
`--query=#`	Perform a number of random range queries on first key attr (must be int unsigned).	All NDB 8.0 releases
`--sys-drop`	Drop any statistics tables and events in NDB kernel (all statistics are lost)	All NDB 8.0 releases
`--sys-create`	Create all statistics tables and events in NDB kernel, if none of them already exist	All NDB 8.0 releases
`--sys-create-if-not-exist`	Create any statistics tables and events in NDB kernel that do not already exist.	All NDB 8.0 releases
`--sys-create-if-not-valid`	Create any statistics tables or events that do not already exist in the NDB kernel. after dropping any that are invalid.	All NDB 8.0 releases
`--sys-check`	Verify that NDB system index statistics and event tables exist.	All NDB 8.0 releases
`--sys-skip-tables`	Do not apply sys-* options to tables.	All NDB 8.0 releases
`--sys-skip-events`	Do not apply sys-* options to events.	All NDB 8.0 releases
`--verbose`, `-v`	Turn on verbose output	All NDB 8.0 releases
`--loops=#`	Set the number of times to perform a given command. Default is 0.	All NDB 8.0 releases

ndb_index_stat statistics options. The following options are used to generate index statistics. They work with a given table and database. They cannot be mixed with system options (see ndb_index_stat system options).

- `--database=`*`name`*, `-d` *`name`*

Property	Value
Command-Line Format	`--database=name`

Property	Value
Type	String
Default Value	[none]
Minimum Value	
Maximum Value	

The name of the database that contains the table being queried.

- --delete

Property	Value
Command-Line Format	--delete
Type	Boolean
Default Value	false
Minimum Value	
Maximum Value	

Delete the index statistics for the given table, stopping any auto-update that was previously configured.

- --update

Property	Value
Command-Line Format	--update
Type	Boolean
Default Value	false
Minimum Value	
Maximum Value	

Update the index statistics for the given table, and restart any auto-update that was previously configured.

- --dump

Property	Value
Command-Line Format	--dump
Type	Boolean
Default Value	false
Minimum Value	
Maximum Value	

Dump the contents of the query cache.

- --query=#

Property	Value
Command-Line Format	--query=#

Property	Value
Type	Numeric
Default Value	`0`
Minimum Value	`0`
Maximum Value	`MAX_INT`

Perform random range queries on first key attribute (must be int unsigned).

ndb_index_stat system options. The following options are used to generate and update the statistics tables in the NDB kernel. None of these options can be mixed with statistics options (see ndb_index_stat statistics options).

- `--sys-drop`

Property	Value
Command-Line Format	`--sys-drop`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Drop all statistics tables and events in the NDB kernel. *This causes all statistics to be lost*.

- `--sys-create`

Property	Value
Command-Line Format	`--sys-create`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Create all statistics tables and events in the NDB kernel. This works only if none of them exist previously.

- `sys-create-if-not-exist`

Property	Value
Command-Line Format	`--sys-create-if-not-exist`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Create any NDB system statistics tables or events (or both) that do not already exist when the program is invoked.

- `--sys-create-if-not-valid`

Property	Value
Command-Line Format	`--sys-create-if-not-valid`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Create any NDB system statistics tables or events that do not already exist, after dropping any that are invalid.

- `--sys-check`

Property	Value
Command-Line Format	`--sys-check`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Verify that all required system statistics tables and events exist in the NDB kernel.

- `--sys-skip-tables`

Property	Value
Command-Line Format	`--sys-skip-tables`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Do not apply any `--sys-*` options to any statistics tables.

- `--sys-skip-events`

Property	Value
Command-Line Format	`--sys-skip-events`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Do not apply any `--sys-*` options to any events.

- `--verbose`

Property	Value
Command-Line Format	`--verbose`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Turn on verbose output.

- `--loops=#`

Property	Value
Command-Line Format	`--loops=#`
Type	Numeric
Default Value	`0`
Minimum Value	`0`
Maximum Value	`MAX_INT`

Repeat commands this number of times (for use in testing).

22.4.15 ndb_move_data — NDB Data Copy Utility

`ndb_move_data` copies data from one NDB table to another.

Usage

The program is invoked with the names of the source and target tables; either or both of these may be qualified optionally with the database name. Both tables must use the NDB storage engine.

```
ndb_move_data options source target
```

The following table includes options that are specific to `ndb_move_data`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_move_data`), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.338 Command-line options for the ndb_move_data program

Format	Description	Added, Deprecated, or Removed
`--abort-on-error`	Dump core on permanent error (debug option)	All NDB 8.0 releases
`--character-sets-dir=name`	Directory where character sets are	All NDB 8.0 releases
`--database=dbname`, `-d`	Name of the database in which the table is found	All NDB 8.0 releases
`--drop-source`	Drop source table after all rows have been moved	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
--error-insert	Insert random temporary errors (testing option)	All NDB 8.0 releases
--exclude-missing-columns	Ignore extra columns in source or target table	All NDB 8.0 releases
--lossy-conversions, -l	Allow attribute data to be truncated when converted to a smaller type	All NDB 8.0 releases
--promote-attributes, -A	Allow attribute data to be converted to a larger type	All NDB 8.0 releases
--staging-tries=x[,y[,z]]	Specify tries on temporary errors. Format is x[,y[,z]] where x=max tries (0=no limit), y=min delay (ms), z=max delay (ms)	All NDB 8.0 releases
--verbose	Enable verbose messages	All NDB 8.0 releases

- --abort-on-error

Property	Value
Command-Line Format	--abort-on-error
Type	Boolean
Default Value	FALSE

Dump core on permanent error (debug option).

- --character-sets-dir=*name*

Property	Value
Command-Line Format	--character-sets-dir=name
Type	String
Default Value	[none]

Directory where character sets are.

- --database=*dbname*, -d

Property	Value
Command-Line Format	--database=dbname
Type	String
Default Value	TEST_DB

Name of the database in which the table is found.

- --drop-source

Property	Value
Command-Line Format	--drop-source

Property	Value
Type	Boolean
Default Value	FALSE

Drop source table after all rows have been moved.

- --error-insert

Property	Value
Command-Line Format	--error-insert
Type	Boolean
Default Value	FALSE

Insert random temporary errors (testing option).

- --exclude-missing-columns

Property	Value
Command-Line Format	--exclude-missing-columns
Type	Boolean
Default Value	FALSE

Ignore extra columns in source or target table.

- --lossy-conversions, -l

Property	Value
Command-Line Format	--lossy-conversions
Type	Boolean
Default Value	FALSE

Allow attribute data to be truncated when converted to a smaller type.

- --promote-attributes, -A

Property	Value
Command-Line Format	--promote-attributes
Type	Boolean
Default Value	FALSE

Allow attribute data to be converted to a larger type.

- --staging-tries=*x[,y[,z]]*

Property	Value
Command-Line Format	--staging-tries=x[,y[,z]]
Type	String
Default Value	0,1000,60000

Specify tries on temporary errors. Format is x[,y[,z]] where x=max tries (0=no limit), y=min delay (ms), z=max delay (ms).

- `--verbose`

Property	Value
Command-Line Format	`--verbose`
Type	Boolean
Default Value	FALSE

Enable verbose messages.

22.4.16 ndb_perror — Obtain NDB Error Message Information

`ndb_perror` shows information about an NDB error, given its error code. This includes the error message, the type of error, and whether the error is permanent or temporary. Added to the MySQL NDB Cluster distribution in NDB 7.6.4, it is intended as a drop-in replacement for `perror --ndb`.

Usage

```
ndb_perror [options] error_code
```

`ndb_perror` does not need to access a running NDB Cluster, or any nodes (including SQL nodes). To view information about a given NDB error, invoke the program, using the error code as an argument, like this:

```
shell> ndb_perror 323
NDB error code 323: Invalid nodegroup id, nodegroup already existing: Permanent error: Application error
```

To display only the error message, invoke `ndb_perror` with the `--silent` option (short form `-s`), as shown here:

```
shell> ndb_perror -s 323
Invalid nodegroup id, nodegroup already existing: Permanent error: Application error
```

Like `perror`, `ndb_perror` accepts multiple error codes:

```
shell> ndb_perror 321 1001
NDB error code 321: Invalid nodegroup id: Permanent error: Application error
NDB error code 1001: Illegal connect string
```

Additional program options for `ndb_perror` are described later in this section.

`ndb_perror` replaces `perror --ndb`, which is deprecated as of NDB 7.6.4 and subject to removal in a future release of MySQL NDB Cluster. To make substitution easier in scripts and other applications that might depend on `perror` for obtaining NDB error information, `ndb_perror` supports its own “dummy” `--ndb` option, which does nothing.

The following table includes all options that are specific to the NDB Cluster program `ndb_perror`. Additional descriptions follow the table.

Table 22.339 Command-line options for the ndb_perror program

Format	Description	Added, Deprecated, or Removed
`--help`, `-?`	Display help text	All NDB 8.0 releases
`--ndb`	For compatibility with applications depending on old versions of perror; does nothing	All NDB 8.0 releases
`--silent`, `-s`	Show error message only	All NDB 8.0 releases
`--version`, `-V`	Print program version information and exit	All NDB 8.0 releases
`--verbose`, `-v`	Verbose output; disable with --silent	All NDB 8.0 releases

Additional Options

- `--help`, `-?`

Property	Value
Command-Line Format	`--help`
Type	Boolean
Default Value	`TRUE`

Display program help text and exit.

- `--ndb`

Property	Value
Command-Line Format	`--ndb`
Type	Boolean
Default Value	`TRUE`

For compatibility with applications depending on old versions of `perror` that use that program's `--ndb` option. The option when used with `ndb_perror` does nothing, and is ignored by it.

- `--silent`, `-s`

Property	Value
Command-Line Format	`--silent`
Type	Boolean
Default Value	`TRUE`

Show error message only.

- `--version`, `-V`

Property	Value
Command-Line Format	`--version`
Type	Boolean
Default Value	`TRUE`

Print program version information and exit.

- `--verbose`, `-v`

Property	Value
Command-Line Format	`--verbose`
Type	Boolean
Default Value	`TRUE`

Verbose output; disable with `--silent`.

22.4.17 ndb_print_backup_file — Print NDB Backup File Contents

`ndb_print_backup_file` obtains diagnostic information from a cluster backup file.

Usage

```
ndb_print_backup_file file_name
```

`file_name` is the name of a cluster backup file. This can be any of the files (`.Data`, `.ctl`, or `.log` file) found in a cluster backup directory. These files are found in the data node's backup directory under the subdirectory `BACKUP-#`, where `#` is the sequence number for the backup. For more information about cluster backup files and their contents, see Section 22.5.3.1, “NDB Cluster Backup Concepts”.

Like `ndb_print_schema_file` and `ndb_print_sys_file` (and unlike most of the other `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_backup_file` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

In NDB 8.0.17 and later, this program can also be used to read undo log files.

Additional Options

None.

22.4.18 ndb_print_file — Print NDB Disk Data File Contents

`ndb_print_file` obtains information from an NDB Cluster Disk Data file.

Usage

```
ndb_print_file [-v] [-q] file_name+
```

`file_name` is the name of an NDB Cluster Disk Data file. Multiple filenames are accepted, separated by spaces.

Like ndb_print_schema_file and ndb_print_sys_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_print_file must be run on an NDB Cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

ndb_print_file supports the following options:

- -v: Make output verbose.
- -q: Suppress output (quiet mode).
- --help, -h, -?: Print help message.

For more information, see Section 22.5.13, “NDB Cluster Disk Data Tables”.

22.4.19 ndb_print_frag_file — Print NDB Fragment List File Contents

ndb_print_frag_file obtains information from a cluster fragment list file. It is intended for use in helping to diagnose issues with data node restarts.

Usage

```
ndb_print_frag_file file_name
```

file_name is the name of a cluster fragment list file, which matches the pattern S*X*.FragList, where *X* is a digit in the range 2-9 inclusive, and are found in the data node file system of the data node having the node ID *nodeid*, in directories named ndb_*nodeid*_fs/D*N*/DBDIH/, where *N* is 1 or 2. Each fragment file contains records of the fragments belonging to each NDB table. For more information about cluster fragment files, see NDB Cluster Data Node File System Directory Files.

Like ndb_print_backup_file, ndb_print_sys_file, and ndb_print_schema_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server), ndb_print_frag_file must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

Sample Output

```
shell> ndb_print_frag_file /usr/local/mysqld/data/ndb_3_fs/D1/DBDIH/S2.FragList
Filename: /usr/local/mysqld/data/ndb_3_fs/D1/DBDIH/S2.FragList with size 8192
noOfPages = 1 noOfWords = 182
Table Data
----------
Num Frags: 2 NoOfReplicas: 2 hashpointer: 4294967040
kvalue: 6 mask: 0x00000000 method: HashMap
Storage is on Logged and checkpointed, survives SR
------ Fragment with FragId: 0 --------
```

```
Preferred Primary: 2 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 0
-------Stored Replica----------
Replica node is: 2 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
-------Stored Replica----------
Replica node is: 3 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
------ Fragment with FragId: 1 --------
Preferred Primary: 3 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 1
-------Stored Replica----------
Replica node is: 3 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
-------Stored Replica----------
Replica node is: 2 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
```

22.4.20 ndb_print_schema_file — Print NDB Schema File Contents

ndb_print_schema_file obtains diagnostic information from a cluster schema file.

Usage

```
ndb_print_schema_file file_name
```

file_name is the name of a cluster schema file. For more information about cluster schema files, see NDB Cluster Data Node File System Directory Files.

Like ndb_print_backup_file and ndb_print_sys_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_print_schema_file must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

22.4.21 ndb_print_sys_file — Print NDB System File Contents

ndb_print_sys_file obtains diagnostic information from an NDB Cluster system file.

Usage

```
ndb_print_sys_file file_name
```

file_name is the name of a cluster system file (sysfile). Cluster system files are located in a data node's data directory (DataDir); the path under this directory to system files matches the pattern ndb_#_fs/D#/DBDIH/P#.sysfile. In each case, the # represents a number (not necessarily the same number). For more information, see NDB Cluster Data Node File System Directory Files.

Like ndb_print_backup_file and ndb_print_schema_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_print_backup_file must be run on a cluster data node, since it accesses the data node file

system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

22.4.22 ndb_redo_log_reader — Check and Print Content of Cluster Redo Log

Reads a redo log file, checking it for errors, printing its contents in a human-readable format, or both. ndb_redo_log_reader is intended for use primarily by NDB Cluster developers and Support personnel in debugging and diagnosing problems.

This utility remains under development, and its syntax and behavior are subject to change in future NDB Cluster releases.

Note

Prior to NDB 7.2, this utility was named ndbd_redo_log_reader.

The C++ source files for ndb_redo_log_reader can be found in the directory /storage/ndb/src/kernel/blocks/dblqh/redoLogReader.

The following table includes options that are specific to the NDB Cluster program ndb_redo_log_reader. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_redo_log_reader), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.340 Command-line options for the ndb_redo_log_reader program

Format	Description	Added, Deprecated, or Removed
-dump	Print dump info	All NDB 8.0 releases
-filedescriptors	Print file descriptors only	All NDB 8.0 releases
--help	Print usage information	All NDB 8.0 releases
-lap	Provide lap info, with max GCI started and completed	All NDB 8.0 releases
-mbyte #	Starting megabyte	All NDB 8.0 releases
-mbyteheaders	Show only the first page header of every megabyte in the file	All NDB 8.0 releases
-nocheck	Do not check records for errors	All NDB 8.0 releases
-noprint	Do not print records	All NDB 8.0 releases
-page #	Start with this page	All NDB 8.0 releases
-pageheaders	Show page headers only	All NDB 8.0 releases
-pageindex #	Start with this page index	All NDB 8.0 releases
-twiddle	Bit-shifted dump	All NDB 8.0 releases

Usage

```
ndb_redo_log_reader file_name [options]
```

file_name is the name of a cluster redo log file. redo log files are located in the numbered directories under the data node's data directory (DataDir); the path under this directory to the redo log files matches the pattern ndb_*nodeid*_fs/D#/DBLQH/S#.FragLog. *nodeid* is the data node's node ID. The two instances of # each represent a number (not necessarily the same number); the number following D is in the range 8-39 inclusive; the range of the number following S varies according to the value of the NoOfFragmentLogFiles configuration parameter, whose default value is 16; thus, the default range of the number in the file name is 0-15 inclusive. For more information, see NDB Cluster Data Node File System Directory Files.

The name of the file to be read may be followed by one or more of the options listed here:

- -dump

Property	Value
Command-Line Format	-dump
Type	Boolean
Default Value	FALSE

Print dump info.

-

Property	Value
Command-Line Format	-filedescriptors
Type	Boolean
Default Value	FALSE

-filedescriptors: Print file descriptors only.

-

Property	Value
Command-Line Format	--help

--help: Print usage information.

- -lap

Property	Value
Command-Line Format	-lap
Type	Boolean
Default Value	FALSE

Provide lap info, with max GCI started and completed.

-

Property	Value
Command-Line Format	-mbyte #
Type	Numeric
Default Value	0
Minimum Value	0
Maximum Value	15

`-mbyte #`: Starting megabyte.

`#` is an integer in the range 0 to 15, inclusive.

Property	Value
Command-Line Format	`-mbyteheaders`
Type	Boolean
Default Value	`FALSE`

  `-mbyteheaders`: Show only the first page header of every megabyte in the file.

Property	Value
Command-Line Format	`-noprint`
Type	Boolean
Default Value	`FALSE`

  `-noprint`: Do not print the contents of the log file.

Property	Value
Command-Line Format	`-nocheck`
Type	Boolean
Default Value	`FALSE`

  `-nocheck`: Do not check the log file for errors.

Property	Value
Command-Line Format	`-page #`
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`31`

  `-page #`: Start at this page.

  `#` is an integer in the range 0 to 31, inclusive.

Property	Value
Command-Line Format	`-pageheaders`
Type	Boolean
Default Value	`FALSE`

  `-pageheaders`: Show page headers only.

Property	Value
Command-Line Format	`-pageindex #`
Type	Integer

Property	Value
Default Value	12
Minimum Value	12
Maximum Value	8191

`-pageindex #`: Start at this page index.

`#` is an integer between 12 and 8191, inclusive.

- `-twiddle`

Property	Value
Command-Line Format	`-twiddle`
Type	Boolean
Default Value	FALSE

Bit-shifted dump.

Like `ndb_print_backup_file` and `ndb_print_schema_file` (and unlike most of the `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_redo_log_reader` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

22.4.23 `ndb_restore` — Restore an NDB Cluster Backup

The NDB Cluster restoration program is implemented as a separate command-line utility `ndb_restore`, which can normally be found in the MySQL `bin` directory. This program reads the files created as a result of the backup and inserts the stored information into the database.

Note

Beginning with NDB 8.0.17, this program no longer prints `NDBT_ProgramExit: ...` when it finishes its run. Applications depending on this behavior should be modified accordingly when upgrading from NDB 8.0.16 or earlier to a NDB 8.0 later release.

`ndb_restore` must be executed once for each of the backup files that were created by the `START BACKUP` command used to create the backup (see Section 22.5.3.2, “Using The NDB Cluster Management Client to Create a Backup”). This is equal to the number of data nodes in the cluster at the time that the backup was created.

Note

Before using `ndb_restore`, it is recommended that the cluster be running in single user mode, unless you are restoring multiple data nodes in parallel. See Section 22.5.8, “NDB Cluster Single User Mode”, for more information.

The following table includes options that are specific to the NDB Cluster native backup restoration program `ndb_restore`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_restore`), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.341 Command-line options for the ndb_restore program

Format	Description	Added, Deprecated, or Removed
--append	Append data to a tab-delimited file	All NDB 8.0 releases
--backup_path=dir_name	Path to backup files directory	All NDB 8.0 releases
--backupid=#, -b	Restore from the backup with the given ID	All NDB 8.0 releases
--connect, -c	Alias for --connectstring.	All NDB 8.0 releases
--disable-indexes	Causes indexes from a backup to be ignored; may decrease time needed to restore data.	All NDB 8.0 releases
--dont-ignore-systab-0, -f	Do not ignore system table during restore. Experimental only; not for production use	All NDB 8.0 releases
--exclude-databases=db-list	List of one or more databases to exclude (includes those not named)	All NDB 8.0 releases
--exclude-intermediate-sql-tables[=TRUE|FALSE]	If TRUE (the default), do not restore any intermediate tables (having names prefixed with '#sql-') that were left over from copying ALTER TABLE operations.	All NDB 8.0 releases
--exclude-missing-columns	Causes columns from the backup version of a table that are missing from the version of the table in the database to be ignored.	All NDB 8.0 releases
--exclude-missing-tables	Causes tables from the backup that are missing from the database to be ignored.	All NDB 8.0 releases
--exclude-tables=table-list	List of one or more tables to exclude (includes those in the same database that are not named); each table reference must include the database name	All NDB 8.0 releases
--fields-enclosed-by=char	Fields are enclosed with the indicated character	All NDB 8.0 releases
--fields-optionally-enclosed-by	Fields are optionally enclosed with the indicated character	All NDB 8.0 releases
--fields-terminated-by=char	Fields are terminated by the indicated character	All NDB 8.0 releases
--hex	Print binary types in hexadecimal format	All NDB 8.0 releases
--include-databases=db-list	List of one or more databases to restore (excludes those not named)	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
--include-tables=table-list	List of one or more tables to restore (excludes those in same database that are not named); each table reference must include the database name	All NDB 8.0 releases
--lines-terminated-by=char	Lines are terminated by the indicated character	All NDB 8.0 releases
--lossy-conversions, -L	Allow lossy conversions of column values (type demotions or changes in sign) when restoring data from backup	All NDB 8.0 releases
--no-binlog	If a mysqld is connected and using binary logging, do not log the restored data	All NDB 8.0 releases
--no-restore-disk-objects, -d	Do not restore objects relating to Disk Data	All NDB 8.0 releases
--no-upgrade, -u	Do not upgrade array type for varsize attributes which do not already resize VAR data, and do not change column attributes	All NDB 8.0 releases
--ndb-nodegroup-map=map, -z	Nodegroup map for NDBCLUSTER storage engine. Syntax: list of (source_nodegroup, destination_nodegroup)	All NDB 8.0 releases
--nodeid=#, -n	ID of node where backup was taken	All NDB 8.0 releases
--parallelism=#, -p	Number of parallel transactions to use while restoring data	All NDB 8.0 releases
--preserve-trailing-spaces, -P	Allow preservation of trailing spaces (including padding) when promoting fixed-width string types to variable-width types	All NDB 8.0 releases
--print	Print metadata, data and log to stdout (equivalent to --print-meta --print-data --print-log)	All NDB 8.0 releases
--print-data	Print data to stdout	All NDB 8.0 releases
--print-log	Print to stdout	All NDB 8.0 releases
--print-meta	Print metadata to stdout	All NDB 8.0 releases
print-sql-log	Write SQL log to stdout; default is FALSE	All NDB 8.0 releases
--progress-frequency=#	Print status of restoration each given number of seconds	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
--promote-attributes, -A	Allow attributes to be promoted when restoring data from backup	All NDB 8.0 releases
--rebuild-indexes	Causes multithreaded rebuilding of ordered indexes found in the backup. Number of threads used is determined by setting BuildIndexThreads parameter.	All NDB 8.0 releases
--restore-data, -r	Restore table data and logs into NDB Cluster using the NDB API	All NDB 8.0 releases
--restore-epoch, -e	Restore epoch info into the status table. Convenient on a MySQL Cluster replication slave for starting replication. The row in mysql.ndb_apply_status with id 0 will be updated/inserted.	All NDB 8.0 releases
--restore-meta, -m	Restore metadata to NDB Cluster using the NDB API	All NDB 8.0 releases
--restore-privilege-tables	Restore MySQL privilege tables that were previously moved to NDB.	All NDB 8.0 releases
--rewrite-database=olddb,newdb	Restores to a database with a different name than the original	All NDB 8.0 releases
--skip-broken-objects	Causes missing blob tables in the backup file to be ignored.	All NDB 8.0 releases
--skip-table-check, -s	Skip table structure check during restoring of data	All NDB 8.0 releases
--skip-unknown-objects	Causes schema objects not recognized by ndb_restore to be ignored when restoring a backup made from a newer MySQL Cluster version to an older version.	All NDB 8.0 releases
--tab=dir_name, -T dir_name	Creates a tab-separated .txt file for each table in the given path	All NDB 8.0 releases
--verbose=#	Level of verbosity in output	All NDB 8.0 releases

Typical options for this utility are shown here:

```
ndb_restore [-c connection_string] -n node_id -b backup_id \
      [-m] -r --backup_path=/path/to/backup/files
```

Normally, when restoring from an NDB Cluster backup, ndb_restore requires at a minimum the --nodeid (short form: -n), --backupid (short form: -b), and --backup_path options. In addition, when

ndb_restore is used to restore any tables containing unique indexes, you must include --disable-indexes or --rebuild-indexes. (Bug #57782, Bug #11764893)

The -c option is used to specify a connection string which tells ndb_restore where to locate the cluster management server (see Section 22.3.3.3, “NDB Cluster Connection Strings”). If this option is not used, then ndb_restore attempts to connect to a management server on localhost:1186. This utility acts as a cluster API node, and so requires a free connection “slot” to connect to the cluster management server. This means that there must be at least one [api] or [mysqld] section that can be used by it in the cluster config.ini file. It is a good idea to keep at least one empty [api] or [mysqld] section in config.ini that is not being used for a MySQL server or other application for this reason (see Section 22.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”).

You can verify that ndb_restore is connected to the cluster by using the SHOW command in the ndb_mgm management client. You can also accomplish this from a system shell, as shown here:

```
shell> ndb_mgm -e "SHOW"
```

More detailed information about all options used by ndb_restore can be found in the following list:

- --append

Property	Value
Command-Line Format	--append

When used with the --tab and --print-data options, this causes the data to be appended to any existing files having the same names.

- --backup_path=*dir_name*

Property	Value
Command-Line Format	--backup-path=dir_name
Type	Directory name
Default Value	./

The path to the backup directory is required; this is supplied to ndb_restore using the --backup_path option, and must include the subdirectory corresponding to the ID backup of the backup to be restored. For example, if the data node's DataDir is /var/lib/mysql-cluster, then the backup directory is /var/lib/mysql-cluster/BACKUP, and the backup files for the backup with the ID 3 can be found in /var/lib/mysql-cluster/BACKUP/BACKUP-3. The path may be absolute or relative to the directory in which the ndb_restore executable is located, and may be optionally prefixed with backup_path=.

It is possible to restore a backup to a database with a different configuration than it was created from. For example, suppose that a backup with backup ID 12, created in a cluster with two storage nodes having the node IDs 2 and 3, is to be restored to a cluster with four nodes. Then ndb_restore must be run twice—once for each storage node in the cluster where the backup was taken. However, ndb_restore cannot always restore backups made from a cluster running one version of MySQL to a cluster running a different MySQL version.

Important

It is not possible to restore a backup made from a newer version of NDB Cluster using an older version of ndb_restore. You can restore a backup made from

> a newer version of MySQL to an older cluster, but you must use a copy of ndb_restore from the newer NDB Cluster version to do so.
>
> For example, to restore a cluster backup taken from a cluster running NDB Cluster 7.5.16 to a cluster running NDB Cluster 7.4.26, you must use the ndb_restore that comes with the NDB Cluster 7.5.16 distribution.

For more rapid restoration, the data may be restored in parallel, provided that there is a sufficient number of cluster connections available. That is, when restoring to multiple nodes in parallel, you must have an [api] or [mysqld] section in the cluster config.ini file available for each concurrent ndb_restore process. However, the data files must always be applied before the logs.

- --backupid=#, -b

Property	Value
Command-Line Format	--backupid=#
Type	Numeric
Default Value	none

This option is used to specify the ID or sequence number of the backup, and is the same number shown by the management client in the Backup *backup_id* completed message displayed upon completion of a backup. (See Section 22.5.3.2, “Using The NDB Cluster Management Client to Create a Backup”.)

> **Important**
>
> When restoring cluster backups, you must be sure to restore all data nodes from backups having the same backup ID. Using files from different backups will at best result in restoring the cluster to an inconsistent state, and may fail altogether.

In NDB 8.0.15 and later, this option is required.

- --connect, -c

Property	Value
Command-Line Format	--connect
Type	String
Default Value	localhost:1186

Alias for --ndb-connectstring.

- --disable-indexes

Property	Value
Command-Line Format	--disable-indexes

Disable restoration of indexes during restoration of the data from a native NDB backup. Afterwards, you can restore indexes for all tables at once with multithreaded building of indexes using --rebuild-indexes, which should be faster than rebuilding indexes concurrently for very large tables.

- --dont-ignore-systab-0, -f

Property	Value
Command-Line Format	--dont-ignore-systab-0

Normally, when restoring table data and metadata, ndb_restore ignores the copy of the NDB system table that is present in the backup. --dont-ignore-systab-0 causes the system table to be restored. *This option is intended for experimental and development use only, and is not recommended in a production environment*.

- --exclude-databases=*db-list*

Property	Value
Command-Line Format	--exclude-databases=db-list
Type	String
Default Value	

Comma-delimited list of one or more databases which should not be restored.

This option is often used in combination with --exclude-tables; see that option's description for further information and examples.

- --exclude-intermediate-sql-tables[=*TRUE|FALSE*]

Property	Value
Command-Line Format	--exclude-intermediate-sql-tables[=TRUE|FALSE]
Type	Boolean
Default Value	TRUE

When performing copying ALTER TABLE operations, mysqld creates intermediate tables (whose names are prefixed with #sql-). When TRUE, the --exclude-intermediate-sql-tables option keeps ndb_restore from restoring such tables that may have been left over from these operations. This option is TRUE by default.

- --exclude-missing-columns

Property	Value
Command-Line Format	--exclude-missing-columns

It is possible to restore only selected table columns using this option, which causes ndb_restore to ignore any columns missing from tables being restored as compared to the versions of those tables found in the backup. This option applies to all tables being restored. If you wish to apply this option only to selected tables or databases, you can use it in combination with one or more of the --include-* or --exclude-* options described elsewhere in this section to do so, then restore data to the remaining tables using a complementary set of these options.

- --exclude-missing-tables

Property	Value
Command-Line Format	--exclude-missing-tables

It is possible to restore only selected tables using this option, which causes ndb_restore to ignore any tables from the backup that are not found in the target database.

- --exclude-tables=*table-list*

Property	Value
Command-Line Format	--exclude-tables=table-list
Type	String
Default Value	

List of one or more tables to exclude; each table reference must include the database name. Often used together with --exclude-databases.

When --exclude-databases or --exclude-tables is used, only those databases or tables named by the option are excluded; all other databases and tables are restored by ndb_restore.

This table shows several invocations of ndb_restore usng --exclude-* options (other options possibly required have been omitted for clarity), and the effects these options have on restoring from an NDB Cluster backup:

Table 22.342 Several invocations of ndb_restore using --exclude-* options, and the effects these options have on restoring from an NDB Cluster backup.

Option	Result
--exclude-databases=db1	All tables in all databases except db1 are restored; no tables in db1 are restored
--exclude-databases=db1,db2 (or --exclude-databases=db1 --exclude-databases=db2)	All tables in all databases except db1 and db2 are restored; no tables in db1 or db2 are restored
--exclude-tables=db1.t1	All tables except t1 in database db1 are restored; all other tables in db1 are restored; all tables in all other databases are restored
--exclude-tables=db1.t2,db2.t1 (or --exclude-tables=db1.t2 --exclude-tables=db2.t1)	All tables in database db1 except for t2 and all tables in database db2 except for table t1 are restored; no other tables in db1 or db2 are restored; all tables in all other databases are restored

You can use these two options together. For example, the following causes all tables in all databases *except for* databases db1 and db2, and tables t1 and t2 in database db3, to be restored:

```
shell> ndb_restore [...] --exclude-databases=db1,db2 --exclude-tables=db3.t1,db3.t2
```

(Again, we have omitted other possibly necessary options in the interest of clarity and brevity from the example just shown.)

You can use --include-* and --exclude-* options together, subject to the following rules:

- The actions of all --include-* and --exclude-* options are cumulative.

- All --include-* and --exclude-* options are evaluated in the order passed to ndb_restore, from right to left.
- In the event of conflicting options, the first (rightmost) option takes precedence. In other words, the first option (going from right to left) that matches against a given database or table “wins”.

For example, the following set of options causes ndb_restore to restore all tables from database db1 except db1.t1, while restoring no other tables from any other databases:

```
--include-databases=db1 --exclude-tables=db1.t1
```

However, reversing the order of the options just given simply causes all tables from database db1 to be restored (including db1.t1, but no tables from any other database), because the --include-databases option, being farthest to the right, is the first match against database db1 and thus takes precedence over any other option that matches db1 or any tables in db1:

```
--exclude-tables=db1.t1 --include-databases=db1
```

- --fields-enclosed-by=*char*

Property	Value
Command-Line Format	--fields-enclosed-by=char
Type	String
Default Value	

Each column value is enclosed by the string passed to this option (regardless of data type; see the description of --fields-optionally-enclosed-by).

- --fields-optionally-enclosed-by

Property	Value
Command-Line Format	--fields-optionally-enclosed-by
Type	String
Default Value	

The string passed to this option is used to enclose column values containing character data (such as CHAR, VARCHAR, BINARY, TEXT, or ENUM).

- --fields-terminated-by=*char*

Property	Value
Command-Line Format	--fields-terminated-by=char
Type	String
Default Value	\t (tab)

The string passed to this option is used to separate column values. The default value is a tab character (\t).

- --hex

Property	Value
Command-Line Format	`--hex`

If this option is used, all binary values are output in hexadecimal format.

- `--include-databases=db-list`

Property	Value
Command-Line Format	`--include-databases=db-list`
Type	String
Default Value	

Comma-delimited list of one or more databases to restore. Often used together with `--include-tables`; see the description of that option for further information and examples.

- `--include-tables=table-list`

Property	Value
Command-Line Format	`--include-tables=table-list`
Type	String
Default Value	

Comma-delimited list of tables to restore; each table reference must include the database name.

When `--include-databases` or `--include-tables` is used, only those databases or tables named by the option are restored; all other databases and tables are excluded by `ndb_restore`, and are not restored.

The following table shows several invocations of `ndb_restore` using `--include-*` options (other options possibly required have been omitted for clarity), and the effects these have on restoring from an NDB Cluster backup:

Table 22.343 Several invocations of ndb_restore using --include-* options, and their effects on restoring from an NDB Cluster backup.

Option	Result
`--include-databases=db1`	Only tables in database `db1` are restored; all tables in all other databases are ignored
`--include-databases=db1,db2` (or `--include-databases=db1 --include-databases=db2`)	Only tables in databases `db1` and `db2` are restored; all tables in all other databases are ignored
`--include-tables=db1.t1`	Only table `t1` in database `db1` is restored; no other tables in `db1` or in any other database are restored
`--include-tables=db1.t2,db2.t1` (or `--include-tables=db1.t2 --include-tables=db2.t1`)	Only the table `t2` in database `db1` and the table `t1` in database `db2` are restored; no other tables in `db1`, `db2`, or any other database are restored

You can also use these two options together. For example, the following causes all tables in databases `db1` and `db2`, together with the tables `t1` and `t2` in database `db3`, to be restored (and no other databases or tables):

```
shell> ndb_restore [...] --include-databases=db1,db2 --include-tables=db3.t1,db3.t2
```

(Again we have omitted other, possibly required, options in the example just shown.)

It also possible to restore only selected databases, or selected tables from a single database, without any `--include-*` (or `--exclude-*`) options, using the syntax shown here:

```
ndb_restore other_options db_name,[db_name[,...] | tbl_name[,tbl_name][,...]]
```

In other words, you can specify either of the following to be restored:

- All tables from one or more databases
- One or more tables from a single database

- `--lines-terminated-by=char`

Property	Value
Command-Line Format	`--lines-terminated-by=char`
Type	String
Default Value	`\n (linebreak)`

Specifies the string used to end each line of output. The default is a linefeed character (`\n`).

- `--lossy-conversions`, `-L`

Property	Value
Command-Line Format	`--lossy-conversions`
Type	Boolean
Default Value	`FALSE` (If option is not used)

This option is intended to complement the `--promote-attributes` option. Using `--lossy-conversions` allows lossy conversions of column values (type demotions or changes in sign) when restoring data from backup. With some exceptions, the rules governing demotion are the same as for MySQL replication; see Replication of Columns Having Different Data Types, for information about specific type conversions currently supported by attribute demotion.

`ndb_restore` reports any truncation of data that it performs during lossy conversions once per attribute and column.

- `--no-binlog`

Property	Value
Command-Line Format	`--no-binlog`

This option prevents any connected SQL nodes from writing data restored by `ndb_restore` to their binary logs.

- `--no-restore-disk-objects`, `-d`

Property	Value
Command-Line Format	--no-restore-disk-objects
Type	Boolean
Default Value	FALSE

This option stops ndb_restore from restoring any NDB Cluster Disk Data objects, such as tablespaces and log file groups; see Section 22.5.13, “NDB Cluster Disk Data Tables”, for more information about these.

- --no-upgrade, -u

Property	Value
Command-Line Format	--no-upgrade

When using ndb_restore to restore a backup, VARCHAR columns created using the old fixed format are resized and recreated using the variable-width format now employed. This behavior can be overridden by specifying --no-upgrade.

- --ndb-nodegroup-map=*map*, -z

Property	Value
Command-Line Format	--ndb-nodegroup-map=map

This option can be used to restore a backup taken from one node group to a different node group. Its argument is a list of the form *source_node_group, target_node_group*.

- --nodeid=#, -n

Property	Value
Command-Line Format	--nodeid=#
Type	Numeric
Default Value	none

Specify the node ID of the data node on which the backup was taken.

When restoring to a cluster with different number of data nodes from that where the backup was taken, this information helps identify the correct set or sets of files to be restored to a given node. (In such cases, multiple files usually need to be restored to a single data node.) See Section 22.4.23.1, “Restoring to a different number of data nodes”, for additional information and examples.

In NDB 8.0.15 and later, this option is required.

- --parallelism=#, -p

Property	Value
Command-Line Format	--parallelism=#
Type	Numeric
Default Value	128
Minimum Value	1

Property	Value
Maximum Value	1024

ndb_restore uses single-row transactions to apply many rows concurrently. This parameter determines the number of parallel transactions (concurrent rows) that an instance of ndb_restore tries to use. By default, this is 128; the minimum is 1, and the maximum is 1024.

The work of performing the inserts is parallelized across the threads in the data nodes involved. This mechanism is employed for restoring bulk data from the .Data file—that is, the fuzzy snapshot of the data; it is not used for building or rebuilding indexes. The change log is applied serially; index drops and builds are DDL operations and handled separately. There is no thread-level parallelism on the client side of the restore.

- --preserve-trailing-spaces, -P

Property	Value
Command-Line Format	--preserve-trailing-spaces

Cause trailing spaces to be preserved when promoting a fixed-width character data type to its variable-width equivalent—that is, when promoting a CHAR column value to VARCHAR, or a BINARY column value to VARBINARY. Otherwise, any trailing spaces are dropped from such column values when they are inserted into the new columns.

Note

Although you can promote CHAR columns to VARCHAR and BINARY columns to VARBINARY, you cannot promote VARCHAR columns to CHAR or VARBINARY columns to BINARY.

- --print

Property	Value
Command-Line Format	--print
Type	Boolean
Default Value	FALSE

Causes ndb_restore to print all data, metadata, and logs to stdout. Equivalent to using the --print-data, --print-meta, and --print-log options together.

Note

Use of --print or any of the --print_* options is in effect performing a dry run. Including one or more of these options causes any output to be redirected to stdout; in such cases, ndb_restore makes no attempt to restore data or metadata to an NDB Cluster.

- --print-data

Property	Value
Command-Line Format	--print-data
Type	Boolean
Default Value	FALSE

Cause ndb_restore to direct its output to stdout. Often used together with one or more of --tab, --fields-enclosed-by, --fields-optionally-enclosed-by, --fields-terminated-by, --hex, and --append.

TEXT and BLOB column values are always truncated. Such values are truncated to the first 256 bytes in the output. This cannot currently be overridden when using --print-data.

- --print-log

Property	Value
Command-Line Format	--print-log
Type	Boolean
Default Value	FALSE

Cause ndb_restore to output its log to stdout.

- --print-meta

Property	Value
Command-Line Format	--print-meta
Type	Boolean
Default Value	FALSE

Print all metadata to stdout.

- print-sql-log

Property	Value
Command-Line Format	--print-sql-log
Type	Boolean
Default Value	FALSE

Log SQL statements to stdout. Use the option to enable; normally this behavior is disabled. The option checks before attempting to log whether all the tables being restored have explicitly defined primary keys; queries on a table having only the hidden primary key implemented by NDB cannot be converted to valid SQL.

This option does not work with tables having BLOB columns.

- --progress-frequency=*N*

Property	Value
Command-Line Format	--progress-frequency=#
Type	Numeric
Default Value	0
Minimum Value	0
Maximum Value	65535

Print a status report each *N* seconds while the backup is in progress. 0 (the default) causes no status reports to be printed. The maximum is 65535.

- `--promote-attributes`, `-A`

Property	Value
Command-Line Format	`--promote-attributes`

`ndb_restore` supports limited *attribute promotion* in much the same way that it is supported by MySQL replication; that is, data backed up from a column of a given type can generally be restored to a column using a “larger, similar” type. For example, data from a `CHAR(20)` column can be restored to a column declared as `VARCHAR(20)`, `VARCHAR(30)`, or `CHAR(30)`; data from a `MEDIUMINT` column can be restored to a column of type `INT` or `BIGINT`. See Replication of Columns Having Different Data Types, for a table of type conversions currently supported by attribute promotion.

Attribute promotion by `ndb_restore` must be enabled explicitly, as follows:

1. Prepare the table to which the backup is to be restored. `ndb_restore` cannot be used to re-create the table with a different definition from the original; this means that you must either create the table manually, or alter the columns which you wish to promote using `ALTER TABLE` after restoring the table metadata but before restoring the data.

2. Invoke `ndb_restore` with the `--promote-attributes` option (short form `-A`) when restoring the table data. Attribute promotion does not occur if this option is not used; instead, the restore operation fails with an error.

When converting between character data types and `TEXT` or `BLOB`, only conversions between character types (`CHAR` and `VARCHAR`) and binary types (`BINARY` and `VARBINARY`) can be performed at the same time. For example, you cannot promote an `INT` column to `BIGINT` while promoting a `VARCHAR` column to `TEXT` in the same invocation of `ndb_restore`.

Converting between `TEXT` columns using different character sets is not supported, and is expressly disallowed.

When performing conversions of character or binary types to `TEXT` or `BLOB` with `ndb_restore`, you may notice that it creates and uses one or more staging tables named *`table_name`*`$ST`*`node_id`*. These tables are not needed afterwards, and are normally deleted by `ndb_restore` following a successful restoration.

- `--rebuild-indexes`

Property	Value
Command-Line Format	`--rebuild-indexes`

Enable multithreaded rebuilding of the ordered indexes while restoring a native `NDB` backup. The number of threads used for building ordered indexes by `ndb_restore` with this option is controlled by the `BuildIndexThreads` data node configuration parameter and the number of LDMs.

It is necessary to use this option only for the first run of `ndb_restore`; this causes all ordered indexes to be rebuilt without using `--rebuild-indexes` again when restoring subsequent nodes. You should use this option prior to inserting new rows into the database; otherwise, it is possible for a row to be inserted that later causes a unique constraint violation when trying to rebuild the indexes.

Building of ordered indices is parallelized with the number of LDMs by default. Offline index builds performed during node and system restarts can be made faster using the BuildIndexThreads data node configuration parameter; this parameter has no effect on dropping and rebuilding of indexes by ndb_restore, which is performed online.

Rebuilding of unique indexes uses disk write bandwidth for redo logging and local checkpointing. An insufficient amount of this bandwith can lead to redo buffer overload or log overload errors. In such cases you can run ndb_restore --rebuild-indexes again; the process resumes at the point where the error occurred. You can also do this when you have encountered temporary errors. You can repeat execution of ndb_restore --rebuild-indexes indefinitely; you may be able to stop such errors by reducing the value of --parallelism. If the problem is insufficient space, you can increase the size of the redo log (FragmentLogFileSize node configuration parameter), or you can increase the speed at which LCPs are performed (MaxDiskWriteSpeed and related parameters), in order to free space more quickly.

- --restore-data, -r

Property	Value
Command-Line Format	--restore-data
Type	Boolean
Default Value	FALSE

Output NDB table data and logs.

- --restore-epoch, -e

Property	Value
Command-Line Format	--restore-epoch

Add (or restore) epoch information to the cluster replication status table. This is useful for starting replication on an NDB Cluster replication slave. When this option is used, the row in the mysql.ndb_apply_status having 0 in the id column is updated if it already exists; such a row is inserted if it does not already exist. (See Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”.)

- --restore-meta, -m

Property	Value
Command-Line Format	--restore-meta
Type	Boolean
Default Value	FALSE

This option causes ndb_restore to print NDB table metadata.

The first time you run the ndb_restore restoration program, you also need to restore the metadata. In other words, you must re-create the database tables—this can be done by running it with the --restore-meta (-m) option. Restoring the metadata need be done only on a single data node; this is sufficient to restore it to the entire cluster.

In older versions of NDB Cluster, tables whose schemas were restored using this option used the same number of partitions as they did on the original cluster, even if it had a differing number of data nodes

from the new cluster. In NDB 8.0, when restoring metadata, this is no longer an issue; ndb_restore now uses the default number of partitions for the target cluster, unless the number of local data manager threads is also changed from what it was for data nodes in the original cluster.

When using this option in NDB 8.0.16 or later, it is recommended that auto synchronization be disabled by setting ndb_metadata_check=OFF until ndb_restore has completed restoring the metadata, after which it can it turned on again to synchronize objects newly created in the NDB dictionary.

Note

The cluster should have an empty database when starting to restore a backup. (In other words, you should start the data nodes with --initial prior to performing the restore.)

- --restore-privilege-tables

Property	Value
Command-Line Format	--restore-privilege-tables
Type	Boolean
Default Value	FALSE (If option is not used)

ndb_restore does not by default restore distributed MySQL privilege tables created in releases of NDB Cluster prior to version 8.0, which does not support distrubuted privileges as implemented in NDB 7.6 and earlier. This option causes ndb_restore to restore them.

In NDB 8.0.16 and later, such tables are not used for access control; as part of the MySQL server's upgrade process, the server creates InnoDB copies of these tables local to itself. For more information, see Section 22.2.8, “Upgrading and Downgrading NDB Cluster”, as well as Section 6.2.3, “Grant Tables”.

- --rewrite-database=*olddb,newdb*

Property	Value
Command-Line Format	--rewrite-database=olddb,newdb
Type	String
Default Value	none

This option makes it possible to restore to a database having a different name from that used in the backup. For example, if a backup is made of a database named products, you can restore the data it contains to a database named inventory, use this option as shown here (omitting any other options that might be required):

```
shell> ndb_restore --rewrite-database=product,inventory
```

The option can be employed multiple times in a single invocation of ndb_restore. Thus it is possible to restore simultaneously from a database named db1 to a database named db2 and from a database named db3 to one named db4 using --rewrite-database=db1,db2 --rewrite-database=db3,db4. Other ndb_restore options may be used between multiple occurrences of --rewrite-database.

In the event of conflicts between multiple --rewrite-database options, the last --rewrite-database option used, reading from left to right, is the one that takes effect. For example, if --

rewrite-database=db1,db2 --rewrite-database=db1,db3 is used, only --rewrite-database=db1,db3 is honored, and --rewrite-database=db1,db2 is ignored. It is also possible to restore from multiple databases to a single database, so that --rewrite-database=db1,db3 --rewrite-database=db2,db3 restores all tables and data from databases db1 and db2 into database db3.

Important

When restoring from multiple backup databases into a single target database using --rewrite-database, no check is made for collisions between table or other object names, and the order in which rows are restored is not guaranteed. This means that it is possible in such cases for rows to be overwritten and updates to be lost.

- --skip-broken-objects

Property	Value
Command-Line Format	--skip-broken-objects

This option causes ndb_restore to ignore corrupt tables while reading a native NDB backup, and to continue restoring any remaining tables (that are not also corrupted). Currently, the --skip-broken-objects option works only in the case of missing blob parts tables.

- --skip-table-check, -s

Property	Value
Command-Line Format	--skip-table-check

It is possible to restore data without restoring table metadata. By default when doing this, ndb_restore fails with an error if a mismatch is found between the table data and the table schema; this option overrides that behavior.

Some of the restrictions on mismatches in column definitions when restoring data using ndb_restore are relaxed; when one of these types of mismatches is encountered, ndb_restore does not stop with an error as it did previously, but rather accepts the data and inserts it into the target table while issuing a warning to the user that this is being done. This behavior occurs whether or not either of the options --skip-table-check or --promote-attributes is in use. These differences in column definitions are of the following types:

- Different COLUMN_FORMAT settings (FIXED, DYNAMIC, DEFAULT)
- Different STORAGE settings (MEMORY, DISK)
- Different default values
- Different distribution key settings

- --skip-unknown-objects

Property	Value
Command-Line Format	--skip-unknown-objects

This option causes ndb_restore to ignore any schema objects it does not recognize while reading a native NDB backup. This can be used for restoring a backup made from a cluster running (for example) NDB 7.6 to a cluster running NDB Cluster 7.5.

- --tab=*dir_name*, -T *dir_name*

Property	Value
Command-Line Format	--tab=dir_name
Type	Directory name

Causes --print-data to create dump files, one per table, each named *tbl_name*.txt. It requires as its argument the path to the directory where the files should be saved; use . for the current directory.

- --verbose=#

Property	Value
Command-Line Format	--verbose=#
Type	Numeric
Default Value	1
Minimum Value	0
Maximum Value	255

Sets the level for the verbosity of the output. The minimum is 0; the maximum is 255. The default value is 1.

Error reporting.
ndb_restore reports both temporary and permanent errors. In the case of temporary errors, it may able to recover from them, and reports Restore successful, but encountered temporary error, please look at configuration in such cases.

Important

After using ndb_restore to initialize an NDB Cluster for use in circular replication, binary logs on the SQL node acting as the replication slave are not automatically created, and you must cause them to be created manually. To cause the binary logs to be created, issue a SHOW TABLES statement on that SQL node before running START SLAVE. This is a known issue in NDB Cluster.

Restoring a backup to a previous version of NDB Cluster. You may encounter issues when restoring a backup taken from a later version of NDB Cluster to a previous one, due to the use of features which do not exist in the earlier version. For example, tables created in NDB 8.0 by default use the utf8mb4_ai_ci character set, which is not available in NDB 7.6 and earlier, and so cannot be read by an ndb_restore binary from one of these earlier versions.

22.4.23.1 Restoring to a different number of data nodes

It is possible to restore from an NDB backup to a cluster having a different number of data nodes than the original from which the backup was taken. The following two sections discuss, respectively, the cases where the target cluster has a lesser or greater number of data nodes than the source of the backup.

Restoring to Fewer Nodes Than the Original

You can restore to a cluster having fewer data nodes than the original provided that the larger number of nodes is an even multiple of the smaller number. In the following example, we use a backup taken on a cluster having four data nodes to a cluster having two data nodes.

1. The management server for the original cluster is on host `host10`. The original cluster has four data nodes, with the node IDs and host names shown in the following extract from the management server's `config.ini` file:

   ```
   [ndbd]
   NodeId=2
   HostName=host2

   [ndbd]
   NodeId=4
   HostName=host4

   [ndbd]
   NodeId=6
   HostName=host6

   [ndbd]
   NodeId=8
   HostName=host8
   ```

   We assume that each data node was originally started with `ndbmtd --ndb-connectstring=host10` or the equivalent.

2. Perform a backup in the normal manner. See Section 22.5.3.2, “Using The NDB Cluster Management Client to Create a Backup”, for information about how to do this.

3. The files created by the backup on each data node are listed here, where *`N`* is the node ID and *`B`* is the backup ID.

   - `BACKUP-`*`B`*`-0.`*`N`*`.Data`
   - `BACKUP-`*`B`*`.`*`N`*`.ctl`
   - `BACKUP-`*`B`*`.`*`N`*`.log`

   These files are found under `BackupDataDir/BACKUP/BACKUP-`*`B`*, on each data node. For the rest of this example, we assume that the backup ID is 1.

   Have all of these files available for later copying to the new data nodes (where they can be accessed on the data node's local file system by `ndb_restore`). It is simplest to copy them all to a single location; we assume that this is what you have done.

4. The management server for the target cluster is on host `host20`, and the target has two data nodes, with the node IDs and host names shown, from the management server `config.ini` file on `host20`:

   ```
   [ndbd]
   NodeId=3
   hostname=host3

   [ndbd]
   NodeId=5
   hostname=host5
   ```

Each of the data node processes on host3 and host5 should be started with ndbmtd -c host20 --initial or the equivalent, so that the new (target) cluster starts with clean data node file systems.

5. Copy two different sets of two backup files to each of the target data nodes. For this example, copy the backup files from nodes 2 and 4 from the original cluster to node 3 in the target cluster. These files are listed here:

   - BACKUP-1-0.2.Data
   - BACKUP-1.2.ctl
   - BACKUP-1.2.log
   - BACKUP-1-0.6.Data
   - BACKUP-1.6.ctl
   - BACKUP-1.6.log

   Then copy the backup files from nodes 6 and 8 to node 5; these files are shown in the following list:

   - BACKUP-1-0.4.Data
   - BACKUP-1.4.ctl
   - BACKUP-1.4.log
   - BACKUP-1-0.8.Data
   - BACKUP-1.8.ctl
   - BACKUP-1.8.log

   For the remainder of this example, we assume that the respective backup files have been saved to the directory /BACKUP-1 on each of nodes 3 and 5.

6. On each of the two target data nodes, you must restore from both sets of backups. First, restore the backups from nodes 2 and 4 to node 3 by invoking ndb_restore on host3 as shown here:

```
shell> ndb_restore -c host20 --nodeid=2 --backupid=1 --restore-data --backup_path=/BACKUP-1
shell> ndb_restore -c host20 --nodeid=4 --backupid=1 --restore-data --backup_path=/BACKUP-1
```

   Then restore the backups from nodes 6 and 8 to node 5 by invoking ndb_restore on host5, like this:

```
shell> ndb_restore -c host20 --nodeid=6 --backupid=1 --restore-data --backup_path=/BACKUP-1
shell> ndb_restore -c host20 --nodeid=8 --backupid=1 --restore-data --backup_path=/BACKUP-1
```

Restoring to More Nodes Than the Original

The node ID specified for a given ndb_restore command is that of the node in the original backup and not that of the data node to restore it to. When performing a backup using the method described in this section, ndb_restore connects to the management server and obtains a list of data nodes in the cluster the backup is being restored to. The restored data is distributed accordingly, so that the number of nodes in the target cluster does not need to be to be known or calculated when performing the backup.

Note

When changing the total number of LCP threads or LQH threads per node group, you should recreate the schema from backup created using `mysqldump`.

1. *Create the backup of the data*. You can do this by invoking the `ndb_mgm` client `START BACKUP` command from the system shell, like this:

   ```
   shell> ndb_mgm -e "START BACKUP 1"
   ```

   This assumes that the desired backup ID is 1.

2. Create a backup of the schema. This step is necessary only if the total number of LCP threads or LQH threads per node group is changed.

   ```
   shell> mysqldump --no-data --routines --events --triggers --databases > myschema.sql
   ```

   Important

   Once you have created the `NDB` native backup using `ndb_mgm`, you must not make any schema changes before creating the backup of the schema, if you do so.

3. Copy the backup directory to the new cluster. For example if the backup you want to restore has ID 1 and `BackupDataDir` = `/backups/node_nodeid`, then the path to the backup on this node is `/backups/node_1/BACKUP/BACKUP-1`. Inside this directory there are three files, listed here:

   - `BACKUP-1-0.1.Data`
   - `BACKUP-1.1.ctl`
   - `BACKUP-1.1.log`

   You should copy the entire directory to the new node.

   If you needed to create a schema file, copy this to a location on an SQL node where it can be read by `mysqld`.

There is no requirement for the backup to be restored from a specific node or nodes.

To restore from the backup just created, perform the following steps:

1. *Restore the schema*.

   - If you created a separate schema backup file using `mysqldump`, import this file using the `mysql` client, similar to what is shown here:

     ```
     shell> mysql < myschema.sql
     ```

     When importing the schema file, you may need to specify the `--user` and `--password` options (and possibly others) in addition to what is shown, in order for the `mysql` client to be able to connect to the MySQL server.

   - If you did *not* need to create a schema file, you can re-create the schema using `ndb_restore --restore-meta` (short form `-m`), similar to what is shown here:

     ```
     ```

```
shell> ndb_restore --nodeid=1 --backupid=1 --restore-meta --backup_path=/backups/node_1/BACKUP/BACKUP
```

ndb_restore must be able to contact the management server; add the --ndb-connectstring option if and as needed to make this possible.

2. *Restore the data*. This needs to be done once for each data node in the original cluster, each time using that data node's node ID. Assuming that there were 4 data nodes originally, the set of commands required would look something like this:

```
ndb_restore --nodeid=1 --backupid=1 --restore-data --backup_path=/backups/node_1/BACKUP/BACKUP-1 --disa
ndb_restore --nodeid=2 --backupid=1 --restore-data --backup_path=/backups/node_2/BACKUP/BACKUP-1 --disa
ndb_restore --nodeid=3 --backupid=1 --restore-data --backup_path=/backups/node_3/BACKUP/BACKUP-1 --disa
ndb_restore --nodeid=4 --backupid=1 --restore-data --backup_path=/backups/node_4/BACKUP/BACKUP-1 --disa
```

These can be run in parallel.

Be sure to add the --ndb-connectstring option as needed.

3. *Rebuild the indexes*. These were disabled by the --disable-indexes option used in the commands just shown. Recreating the indexes avoids errors due to the restore not being consistent at all points. Rebuilding the indexes can also improve performance in some cases. To rebuild the indexes, execute the following command once, on a single node:

```
shell> ndb_restore --nodeid=1 --backupid=1 --backup_path=/backups/node_1/BACKUP/BACKUP-1 --rebuild-inde
```

As mentioned previously, you may need to add the --ndb-connectstring option, so that ndb_restore can contact the management server.

22.4.23.2 Restoring from a backup taken in parallel

Beginning with NDB Cluster 8.0.16, it is possible to take parallel backups on each data node using ndbmtd with multiple LDMs (see Section 22.5.3.5, “Taking an NDB with Parallel Data Nodes”). The next two sections describe how to restore backups that were taken in this fashion.

Restoring a parallel backup in parallel

Restoring a parallel backup in parallel requires an ndb_restore binary from an NDB Cluster distribution version 8.0.16 or later. The process is not substantially different from that outlined in the general usage section under the description of the ndb_restore program, and consists of executing ndb_restore twice, similarly to what is shown here:

```
shell> ndb_restore -n 1 -b 1 -m --backup_path=path/to/backup_dir/BACKUP/BACKUP-backup_id
shell> ndb_restore -n 1 -b 1 -r --backup_path=path/to/backup_dir/BACKUP/BACKUP-backup_id
```

backup_id is the ID of the backup to be restored. In the general case, no additional special arguments are required; ndb_restore always checks for the existence of parallel subdirectories under the directory indicated by the --backup_path option and restores the metadata (serially) and then the table data (in parallel).

Restoring a parallel backup serially

It is possible to restore a backup that was made using parallelism on the data nodes in serial fashion. To do this, invoke ndb_restore with --backup_path pointing to the subdirectories created by each LDM under the main backup directory, once to any one of the subdirectories to restore the metadata (it does not matter which one, since each subdirectory contains a complete copy of the metadata), then to each of the

subdirectories in turn to restore the data. Suppose that we want to restore the backup having backup ID 100 that was taken with four LDMs, and that the BackupDataDir is /opt. To restore the metadata in this case, we can invoke ndb_restore like this:

```
shell> ndb_restore -n 1 -b 1 -m --backup_path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-1-OF-4
```

To restore the table data, execute ndb_restore four times, each time using one of the subdirectories in turn, as shown here:

```
shell> ndb_restore -n 1 -b 1 -r --backup_path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-1-OF-4
shell> ndb_restore -n 1 -b 1 -r --backup_path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-2-OF-4
shell> ndb_restore -n 1 -b 1 -r --backup_path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-3-OF-4
shell> ndb_restore -n 1 -b 1 -r --backup_path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-4-OF-4
```

You can employ the same technique to restore a parallel backup to an older version of NDB Cluster (prior to NDB 8.0.16) that does not support parallel backups, using the ndb_restore binary supplied with the older version of the NDB Cluster software.

22.4.24 ndb_select_all — Print Rows from an NDB Table

ndb_select_all prints all rows from an NDB table to stdout.

Usage

```
ndb_select_all -c connection_string tbl_name -d db_name [> file_name]
```

The following table includes options that are specific to the NDB Cluster native backup restoration program ndb_select_all. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_select_all), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.344 Command-line options for the ndb_select_all program

Format	Description	Added, Deprecated, or Removed
--database=dbname, -d	Name of the database in which the table is found	All NDB 8.0 releases
--parallelism=#, -p	Degree of parallelism	All NDB 8.0 releases
--lock=#, -l	Lock type	All NDB 8.0 releases
--order=index, -o	Sort resultset according to index whose name is supplied	All NDB 8.0 releases
--descending, -z	Sort resultset in descending order (requires order flag)	All NDB 8.0 releases
--header,	Print header (set to 0|FALSE to disable headers in output)	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
-h --useHexFormat, -x	Output numbers in hexadecimal format	All NDB 8.0 releases
--delimiter=char, -D	Set a column delimiter	All NDB 8.0 releases
--disk	Print disk references (useful only for Disk Data tables having nonindexed columns)	All NDB 8.0 releases
--rowid	Print rowid	All NDB 8.0 releases
--gci	Include GCI in output	All NDB 8.0 releases
--gci64	Include GCI and row epoch in output	All NDB 8.0 releases
--tupscan, -t	Scan in tup order	All NDB 8.0 releases
--nodata	Do not print table column data	All NDB 8.0 releases

- `--database=dbname`, `-d dbname`

  Name of the database in which the table is found. The default value is `TEST_DB`.

- `parallelism=#`, `-p #`

  Specifies the degree of parallelism.

- `--lock=lock_type`, `-l lock_type`

  Employs a lock when reading the table. Possible values for *`lock_type`* are:

  - `0`: Read lock
  - `1`: Read lock with hold
  - `2`: Exclusive read lock

  There is no default value for this option.

- `--order=index_name`, `-o index_name`

  Orders the output according to the index named *`index_name`*.

  Note

  This is the name of an index, not of a column; the index must have been explicitly named when created.

- `--descending`, `-z`

  Sorts the output in descending order. This option can be used only in conjunction with the `-o` (`--order`) option.

- `--header=FALSE`

  Excludes column headers from the output.

- `--useHexFormat -x`

  Causes all numeric values to be displayed in hexadecimal format. This does not affect the output of numerals contained in strings or datetime values.

- `--delimiter=character`, `-D character`

  Causes the *`character`* to be used as a column delimiter. Only table data columns are separated by this delimiter.

  The default delimiter is the tab character.

- `--disk`

  Adds a disk reference column to the output. The column is nonempty only for Disk Data tables having nonindexed columns.

- `--rowid`

  Adds a `ROWID` column providing information about the fragments in which rows are stored.

- `--gci`

  Adds a `GCI` column to the output showing the global checkpoint at which each row was last updated. See Section 22.1, “NDB Cluster Overview”, and Section 22.5.6.2, “NDB Cluster Log Events”, for more information about checkpoints.

- `--gci64`

  Adds a `ROW$GCI64` column to the output showing the global checkpoint at which each row was last updated, as well as the number of the epoch in which this update occurred.

- `--tupscan`, `-t`

  Scan the table in the order of the tuples.

- `--nodata`

  Causes any table data to be omitted.

Sample Output

Output from a MySQL `SELECT` statement:

```
mysql> SELECT * FROM ctest1.fish;
+----+-----------+
| id | name      |
+----+-----------+
|  3 | shark     |
|  6 | puffer    |
|  2 | tuna      |
|  4 | manta ray |
|  5 | grouper   |
|  1 | guppy     |
+----+-----------+
```

```
6 rows in set (0.04 sec)
```

Output from the equivalent invocation of ndb_select_all:

```
shell> ./ndb_select_all -c localhost fish -d ctest1
id      name
3       [shark]
6       [puffer]
2       [tuna]
4       [manta ray]
5       [grouper]
1       [guppy]
6 rows returned

NDBT_ProgramExit: 0 - OK
```

All string values are enclosed by square brackets ([...]) in the output of ndb_select_all. For another example, consider the table created and populated as shown here:

```
CREATE TABLE dogs (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(25) NOT NULL,
    breed VARCHAR(50) NOT NULL,
    PRIMARY KEY pk (id),
    KEY ix (name)
)
TABLESPACE ts STORAGE DISK
ENGINE=NDBCLUSTER;

INSERT INTO dogs VALUES
    ('', 'Lassie', 'collie'),
    ('', 'Scooby-Doo', 'Great Dane'),
    ('', 'Rin-Tin-Tin', 'Alsatian'),
    ('', 'Rosscoe', 'Mutt');
```

This demonstrates the use of several additional ndb_select_all options:

```
shell> ./ndb_select_all -d ctest1 dogs -o ix -z --gci --disk
GCI     id name          breed        DISK_REF
834461  2  [Scooby-Doo]  [Great Dane] [ m_file_no: 0 m_page: 98 m_page_idx: 0 ]
834878  4  [Rosscoe]     [Mutt]       [ m_file_no: 0 m_page: 98 m_page_idx: 16 ]
834463  3  [Rin-Tin-Tin] [Alsatian]   [ m_file_no: 0 m_page: 34 m_page_idx: 0 ]
835657  1  [Lassie]      [Collie]     [ m_file_no: 0 m_page: 66 m_page_idx: 0 ]
4 rows returned

NDBT_ProgramExit: 0 - OK
```

22.4.25 ndb_select_count — Print Row Counts for NDB Tables

ndb_select_count prints the number of rows in one or more NDB tables. With a single table, the result is equivalent to that obtained by using the MySQL statement SELECT COUNT(*) FROM *tbl_name*.

Usage

```
ndb_select_count [-c connection_string] -ddb_name tbl_name[, tbl_name2[, ...]]
```

The following table includes options that are specific to the NDB Cluster native backup restoration program ndb_select_count. Additional descriptions follow the table. For options common to most NDB Cluster

programs (including ndb_select_count), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.345 Command-line options for the ndb_select_count program

Format	Description	Added, Deprecated, or Removed
--database=dbname, -d	Name of the database in which the table is found	All NDB 8.0 releases
--parallelism=#, -p	Degree of parallelism	All NDB 8.0 releases
--lock=#, -l	Lock type	All NDB 8.0 releases

You can obtain row counts from multiple tables in the same database by listing the table names separated by spaces when invoking this command, as shown under **Sample Output**.

Sample Output

```
shell> ./ndb_select_count -c localhost -d ctest1 fish dogs
6 records in table fish
4 records in table dogs

NDBT_ProgramExit: 0 - OK
```

22.4.26 ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster

ndb_setup.py starts the NDB Cluster Auto-Installer and opens the installer's Start page in the default Web browser.

Important

This program is intended to be invoked as a normal user, and not with the mysql, system root or other administrative account.

This section describes usage of and program options for the command-line tool only. For information about using the Auto-Installer GUI that is spawned when ndb_setup.py is invoked, see The NDB Cluster Auto-Installer (NDB 7.5).

Usage

All platforms:

```
ndb_setup.py [options]
```

Additionally, on Windows platforms only:

```
setup.bat [options]
```

The following table includes all options that are supported by the NDB Cluster installation and configuration program ndb_setup.py. Additional descriptions follow the table.

Table 22.346 Command-line options for the ndb_setup.py program

Format	Description	Added, Deprecated, or Removed
--browser-start-page=filename, -s	Page that the web browser opens when starting.	All NDB 8.0 releases
--ca-certs-file=filename, -a	File containing list of client certificates allowed to connect to the server	All NDB 8.0 releases
--cert-file=filename, -c	File containing X509 certificate that identifies the server. (Default: cfg.pem)	All NDB 8.0 releases
--debug-level=level, -d	Python logging module debug level. One of DEBUG, INFO, WARNING (default), ERROR, or CRITICAL.	All NDB 8.0 releases
--help, -h	Print help message	All NDB 8.0 releases
--key-file=file, -k	Specify file containing private key (if not included in --cert-file)	All NDB 8.0 releases
--no-browser, -n	Do not open the start page in a browser, merely start the tool	All NDB 8.0 releases
--port=#, -p	Specify the port used by the web server	All NDB 8.0 releases
--server-log-file=file, o	Log requests to this file. Use '-' to force logging to stderr instead.	All NDB 8.0 releases
--server-name=name, -N	The name of the server to connect with	All NDB 8.0 releases
--use-http, -H	Use unencrypted (HTTP) client/server connection	All NDB 8.0 releases
--use-https, -S	Use encrypted (HTTPS) client/server connection	All NDB 8.0 releases

- --browser-start-page=*file*, -s

Property	Value
Command-Line Format	--browser-start-page=filename
Type	String

Property	Value
Default Value	index.html

Specify the file to open in the browser as the installation and configuration Start page. The default is index.html.

- --ca-certs-file=*file*, -a

Property	Value
Command-Line Format	--ca-certs-file=filename
Type	File name
Default Value	[none]

Specify a file containing a list of client certificates which are allowed to connect to the server. The default is an empty string, which means that no client authentication is used.

- --cert-file=*file*, -c

Property	Value
Command-Line Format	--cert-file=filename
Type	File name
Default Value	/usr/share/mysql/mcc/cfg.pem

Specify a file containing an X509 certificate which identifies the server. It is possible for the certificate to be self-signed. The default is cfg.pem.

- --debug-level=*level*, -d

Property	Value
Command-Line Format	--debug-level=level
Type	Enumeration
Default Value	WARNING
Valid Values	WARNING DEBUG INFO ERROR CRITICAL

Set the Python logging module debug level. This is one of DEBUG, INFO, WARNING, ERROR, or CRITICAL. WARNING is the default.

- --help, -h

Property	Value
Command-Line Format	--help

Print a help message.

- `--key-file=file`, `-d`

Property	Value
Command-Line Format	`--key-file=file`
Type	File name
Default Value	`[none]`

Specify a file containing the private key if this is not included in the X509 certificate file (`--cert-file`). The default is an empty string, which means that no such file is used.

- `--no-browser`, `-n`

Property	Value
Command-Line Format	`--no-browser`

Start the installation and configuration tool, but do not open the Start page in a browser.

- `--port=#`, `-p`

Property	Value
Command-Line Format	`--port=#`
Type	Numeric
Default Value	`8081`
Minimum Value	`1`
Maximum Value	`65535`

Set the port used by the web server. The default is 8081.

- `--server-log-file=file`, `-o`

Property	Value
Command-Line Format	`--server-log-file=file`
Type	File name
Default Value	`ndb_setup.log`
Valid Values	`ndb_setup.log` `-` (Log to stderr)

Log requests to this file. The default is `ndb_setup.log`. To specify logging to `stderr`, rather than to a file, use a `-` (dash character) for the file name.

- `--server-name=host`, `-N`

Property	Value
Command-Line Format	`--server-name=name`
Type	String
Default Value	`localhost`

Specify the host name or IP address for the browser to use when connecting. The default is `localhost`.

- `--use-http`, `-H`

Property	Value
Command-Line Format	`--use-http`

Make the browser use HTTP to connect with the server. This means that the connection is unencrypted and not secured in any way.

- `--use-https`, `-S`

Property	Value
Command-Line Format	`--use-https`

Make the browser use a secure (HTTPS) connection with the server.

22.4.27 ndb_show_tables — Display List of NDB Tables

`ndb_show_tables` displays a list of all `NDB` database objects in the cluster. By default, this includes not only both user-created tables and `NDB` system tables, but `NDB`-specific indexes, internal triggers, and NDB Cluster Disk Data objects as well.

The following table includes options that are specific to the NDB Cluster native backup restoration program `ndb_show_tables`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_show_tables`), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.347 Command-line options for the ndb_show_tables program

Format	Description	Added, Deprecated, or Removed
`--database=string`, `-d`	Specifies the database in which the table is found	All NDB 8.0 releases
`--loops=#`, `-l`	Number of times to repeat output	All NDB 8.0 releases
`--parsable`, `-p`	Return output suitable for MySQL LOAD DATA INFILE statement	All NDB 8.0 releases
`--show-temp-status`	Show table temporary flag	All NDB 8.0 releases
`--type=#`, `-t`	Limit output to objects of this type	All NDB 8.0 releases
`--unqualified`, `-u`	Do not qualify table names	All NDB 8.0 releases

Usage

```
ndb_show_tables [-c connection_string]
```

- `--database`, `-d`

Specifies the name of the database in which the tables are found. If this option has not been specified, and no tables are found in the TEST_DB database, ndb_show_tables issues a warning.

- --loops, -l

  Specifies the number of times the utility should execute. This is 1 when this option is not specified, but if you do use the option, you must supply an integer argument for it.

- --parsable, -p

  Using this option causes the output to be in a format suitable for use with LOAD DATA.

- --show-temp-status

  If specified, this causes temporary tables to be displayed.

- --type, -t

  Can be used to restrict the output to one type of object, specified by an integer type code as shown here:

  - 1: System table
  - 2: User-created table
  - 3: Unique hash index

  Any other value causes all NDB database objects to be listed (the default).

- --unqualified, -u

  If specified, this causes unqualified object names to be displayed.

Note

Only user-created NDB Cluster tables may be accessed from MySQL; system tables such as SYSTAB_0 are not visible to mysqld. However, you can examine the contents of system tables using NDB API applications such as ndb_select_all (see Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”).

22.4.28 ndb_size.pl — NDBCLUSTER Size Requirement Estimator

This is a Perl script that can be used to estimate the amount of space that would be required by a MySQL database if it were converted to use the NDBCLUSTER storage engine. Unlike the other utilities discussed in this section, it does not require access to an NDB Cluster (in fact, there is no reason for it to do so). However, it does need to access the MySQL server on which the database to be tested resides.

Requirements

- A running MySQL server. The server instance does not have to provide support for NDB Cluster.
- A working installation of Perl.
- The DBI module, which can be obtained from CPAN if it is not already part of your Perl installation. (Many Linux and other operating system distributions provide their own packages for this library.)

- A MySQL user account having the necessary privileges. If you do not wish to use an existing account, then creating one using `GRANT USAGE ON db_name.*`—where *db_name* is the name of the database to be examined—is sufficient for this purpose.

`ndb_size.pl` can also be found in the MySQL sources in `storage/ndb/tools`.

The following table includes options that are specific to the NDB Cluster program `ndb_size.pl`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_size.pl`), see Section 22.4.31, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 22.348 Command-line options for the ndb_size.pl program

Format	Description	Added, Deprecated, or Removed
`--database=dbname`	The database or databases to examine; accepts a comma-delimited list; the default is ALL (use all databases found on the server)	All NDB 8.0 releases
`--hostname[:port]`	Specify host and optional port as host[:port]	All NDB 8.0 releases
`--socket=file_name`	Specify a socket to connect to	All NDB 8.0 releases
`--user=string`	Specify a MySQL user name	All NDB 8.0 releases
`--password=string`	Specify a MySQL user password	All NDB 8.0 releases
`--format=string`	Set output format (text or HTML)	All NDB 8.0 releases
`--excludetables=tbl_list`	Skip any tables in a comma-separated list of tables	All NDB 8.0 releases
`--excludedbs=db_list`	Skip any databases in a comma-separated list of databases	All NDB 8.0 releases
`--savequeries=file`	Saves all queries to the database into the file specified	All NDB 8.0 releases
`--loadqueries=file`	Loads all queries from the file specified; does not connect to a database	All NDB 8.0 releases
`--real_table_name=table`	Designates a table to handle unique index size calculations	All NDB 8.0 releases

Usage

```
perl ndb_size.pl [--database={db_name|ALL}] [--hostname=host[:port]] [--socket=socket] \
      [--user=user] [--password=password]  \
      [--help|-h] [--format={html|text}] \
      [--loadqueries=file_name] [--savequeries=file_name]
```

By default, this utility attempts to analyze all databases on the server. You can specify a single database using the `--database` option; the default behavior can be made explicit by using `ALL` for the name of the database. You can also exclude one or more databases by using the `--excludedbs` option with a comma-separated list of the names of the databases to be skipped. Similarly, you can cause specific tables to be skipped by listing their names, separated by commas, following the optional `--`

excludetables option. A host name can be specified using --hostname; the default is localhost. You can specify a port in addition to the host using *host*:*port* format for the value of --hostname. The default port number is 3306. If necessary, you can also specify a socket; the default is /var/lib/mysql.sock. A MySQL user name and password can be specified the corresponding options shown. It also possible to control the format of the output using the --format option; this can take either of the values html or text, with text being the default. An example of the text output is shown here:

```
shell> ndb_size.pl --database=test --socket=/tmp/mysql.sock
ndb_size.pl report for database: 'test' (1 tables)
----------------------------------------------------
Connected to: DBI:mysql:host=localhost;mysql_socket=/tmp/mysql.sock

Including information for versions: 4.1, 5.0, 5.1

test.t1
-------

DataMemory for Columns (* means varsized DataMemory):
         Column Name            Type  Varsized   Key  4.1  5.0   5.1
     HIDDEN_NDB_PKEY          bigint             PRI    8    8     8
                  c2     varchar(50)         Y         52   52    4*
                  c1         int(11)                    4    4     4
                                                       --   --    --
Fixed Size Columns DM/Row                              64   64    12
   Varsize Columns DM/Row                               0    0     4

DataMemory for Indexes:
   Index Name                 Type        4.1        5.0        5.1
      PRIMARY                BTREE         16         16         16
                                           --         --         --
       Total Index DM/Row                  16         16         16

IndexMemory for Indexes:
               Index Name        4.1        5.0        5.1
                  PRIMARY         33         16         16
                                  --         --         --
           Indexes IM/Row         33         16         16

Summary (for THIS table):
                                 4.1        5.0        5.1
    Fixed Overhead DM/Row         12         12         16
           NULL Bytes/Row          4          4          4
           DataMemory/Row         96         96         48
                    (Includes overhead, bitmap and indexes)

  Varsize Overhead DM/Row          0          0          8
   Varsize NULL Bytes/Row          0          0          4
       Avg Varside DM/Row          0          0         16

                 No. Rows          0          0          0

        Rows/32kb DM Page        340        340        680
Fixedsize DataMemory (KB)          0          0          0

Rows/32kb Varsize DM Page          0          0       2040
  Varsize DataMemory (KB)          0          0          0

         Rows/8kb IM Page        248        512        512
         IndexMemory (KB)          0          0          0

Parameter Minimum Requirements
------------------------------
* indicates greater than default

                Parameter     Default        4.1         5.0         5.1
```

```
          DataMemory (KB)       81920          0          0          0
     NoOfOrderedIndexes           128          1          1          1
              NoOfTables           128          1          1          1
       IndexMemory (KB)       18432          0          0          0
NoOfUniqueHashIndexes             64          0          0          0
       NoOfAttributes           1000          3          3          3
         NoOfTriggers            768          5          5          5
```

For debugging purposes, the Perl arrays containing the queries run by this script can be read from the file specified using can be saved to a file using `--savequeries`; a file containing such arrays to be read during script execution can be specified using `--loadqueries`. Neither of these options has a default value.

To produce output in HTML format, use the `--format` option and redirect the output to a file, as shown here:

```
shell> ndb_size.pl --database=test --socket=/tmp/mysql.sock --format=html > ndb_size.html
```

(Without the redirection, the output is sent to `stdout`.)

The output from this script includes the following information:

- Minimum values for the `DataMemory`, `IndexMemory`, `MaxNoOfTables`, `MaxNoOfAttributes`, `MaxNoOfOrderedIndexes`, and `MaxNoOfTriggers` configuration parameters required to accommodate the tables analyzed.
- Memory requirements for all of the tables, attributes, ordered indexes, and unique hash indexes defined in the database.
- The `IndexMemory` and `DataMemory` required per table and table row.

22.4.29 ndb_top — View CPU usage information for NDB threads

`ndb_top` displays running information in the terminal about CPU usage by NDB threads on an NDB Cluster data node. Each thread is represented by two rows in the output, the first showing system statistics, the second showing the measured statistics for the thread.

`ndb_top` is available beginning with MySQL NDB Cluster 7.6.3.

Usage

```
ndb_top [-h hostname] [-t port] [-u user] [-p pass] [-n node_id]
```

`ndb_top` connects to a MySQL Server running as an SQL node of the cluster. By default, it attempts to connect to a `mysqld` running on `localhost` and port 3306, as the MySQL `root` user with no password specified. You can override the default host and port using, respectively, `--host` (`-h`) and `--port` (`-t`). To specify a MySQL user and password, use the `--user` (`-u`) and `--passwd` (`-p`) options. This user must be able to read tables in the `ndbinfo` database (`ndb_top` uses information from `ndbinfo.cpustat` and related tables).

For more information about MySQL user accounts and passwords, see Section 6.2, “Access Control and Account Management”.

Output is available as plain text or an ASCII graph; you can specify this using the `--text` (`-x`) and `--graph` (`-g`) options, respectively. These two display modes provide the same information; they can be used concurrently. At least one display mode must be in use.

Color display of the graph is supported and enabled by default (`--color` or `-c` option). With color support enabled, the graph display shows OS user time in blue, OS system time in green, and idle time as blank. For measured load, blue is used for execution time, yellow for send time, red for time spent in send buffer full waits, and blank spaces for idle time. The percentage shown in the graph display is the sum of percentages for all threads which are not idle. Colors are not currently configurable; you can use grayscale instead by using `--skip-color`.

The sorted view (`--sort`, `-r`) is based on the maximum of the measured load and the load reported by the OS. Display of these can be enabled and disabled using the `--measured-load` (`-m`) and `--os-load` (`-o`) options. Display of at least one of these loads must be enabled.

The program tries to obtain statistics from a data node having the node ID given by the `--node-id` (`-n`) option; if unspecified, this is 1. `ndb_top` cannot provide information about other types of nodes.

The view adjusts itself to the height and width of the terminal window; the minimum supported width is 76 characters.

Once started, `ndb_top` runs continuously until forced to exit; you can quit the program using `Ctrl-C`. The display updates once per second; to set a different delay interval, use `--sleep-time` (`-s`).

Note

`ndb_top` is available on Mac OS X, Linux, and Solaris. It is not currently supported on Windows platforms.

The following table includes all options that are specific to the NDB Cluster program `ndb_top`. Additional descriptions follow the table.

Table 22.349 Command-line options for the ndb_top program

Format	Description	Added, Deprecated, or Removed
`--color`, `-c`	Show ASCII graphs in color; use --skip-colors to disable	All NDB 8.0 releases
`--graph`, `-g`	Display data using graphs; use --skip-graphs to disable	All NDB 8.0 releases
`--help`, `-?`	Show program usage information	All NDB 8.0 releases
`--host[=name]`, `-h`	Host name or IP address of MySQL Server to connect to	All NDB 8.0 releases
`--measured-load`, `-m`	Show measured load by thread	All NDB 8.0 releases
`--node-id[=#]`, `-n`	Watch node having this node ID	All NDB 8.0 releases
`--os-load`, `-o`	Show load measured by operating system	All NDB 8.0 releases
`--passwd[=password]`,	Connect using this password	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
-p		REMOVED: NDB 7.6.4
--password[=password], -p	Connect using this password	All NDB 8.0 releases
--port[=#], -t (<=7.6.5), -P (>=7.6.6)	Port number to use when connecting to MySQL Server	All NDB 8.0 releases
--sleep-time[=seconds], -s	Time to wait between display refreshes, in seconds	All NDB 8.0 releases
--socket, -S	Socket file to use for connection.	All NDB 8.0 releases
--sort, -r	Sort threads by usage; use --skip-sort to disable	All NDB 8.0 releases
--text, -x (<=7.6.5), -t (>=7.6.6)	Display data using text	All NDB 8.0 releases
--user[=name], -u	Connect as this MySQL user	All NDB 8.0 releases

In NDB 7.6.6 and later, ndb_top also supports the common NDB program options --defaults-file, --defaults-extra-file, --print-defaults, --no-defaults, and --defaults-group-suffix. (Bug #86614, Bug #26236298)

Additional Options

- --color, -c

Property	Value
Command-Line Format	--color
Type	Boolean
Default Value	TRUE

Show ASCII graphs in color; use --skip-colors to disable.

- --graph, -g

Property	Value
Command-Line Format	--graph
Type	Boolean
Default Value	TRUE

Display data using graphs; use `--skip-graphs` to disable. This option or `--text` must be true; both options may be true.

- `--help`, `-?`

Property	Value
Command-Line Format	`--help`
Type	Boolean
Default Value	`TRUE`

Show program usage information.

- `--host[=name]`, `-h`

Property	Value
Command-Line Format	`--host[=name]`
Type	String
Default Value	`localhost`

Host name or IP address of MySQL Server to connect to.

- `--measured-load`, `-m`

Property	Value
Command-Line Format	`--measured-load`
Type	Boolean
Default Value	`FALSE`

Show measured load by thread. This option or `--os-load` must be true; both options may be true.

- `--node-id[=#]`, `-n`

Property	Value
Command-Line Format	`--node-id[=#]`
Type	Integer
Default Value	`1`

Watch the data node having this node ID.

- `--os-load`, `-o`

Property	Value
Command-Line Format	`--os-load`
Type	Boolean
Default Value	`TRUE`

Show load measured by operating system. This option or `--measured-load` must be true; both options may be true.

- `--passwd[=password]`, `-p`

Property	Value
Command-Line Format	`--passwd[=password]`
Type	Boolean
Default Value	`NULL`

Connect using this password.

This option is deprecated in NDB 7.6.4. It is removed in NDB 7.6.6, where it is replaced by the `--password` option. (Bug #26907833)

- `--password[=password]`, `-p`

Property	Value
Command-Line Format	`--password[=password]`
Type	Boolean
Default Value	`NULL`

Connect using this password.

This option was added in NDB 7.6.6 as a replacement for the `--passwd` option used previously. (Bug #26907833)

- `--port[=#]`, `-t` (NDB 7.6.6 and later: `-P`)

Property	Value
Command-Line Format	`--port[=#]`
Type	Integer
Default Value	`3306`

Port number to use when connecting to MySQL Server.

Beginning with NDB 7.6.6, the short form for this option is `-P`, and `-t` is repurposed as the short form for the `--text` option. (Bug #26907833)

- `--sleep-time[=seconds]`, `-s`

Property	Value
Command-Line Format	`--sleep-time[=seconds]`
Type	Integer
Default Value	`1`

Time to wait between display refreshes, in seconds.

- `--socket=path/to/file`, `-S`

Property	Value
Command-Line Format	`--socket`
Type	Path name

Property	Value
Default Value	[none]

Use the specified socket file for the connection.

Added in NDB 7.6.6. (Bug #86614, Bug #26236298)

- --sort, -r

Property	Value
Command-Line Format	--sort
Type	Boolean
Default Value	TRUE

Sort threads by usage; use --skip-sort to disable.

- --text, -x (NDB 7.6.6 and later: -t)

Property	Value
Command-Line Format	--text
Type	Boolean
Default Value	FALSE

Display data using text. This option or --graph must be true; both options may be true.

Beginning with NDB 7.6.6, the short form for this option is -t and support for -x is removed. (Bug #26907833)

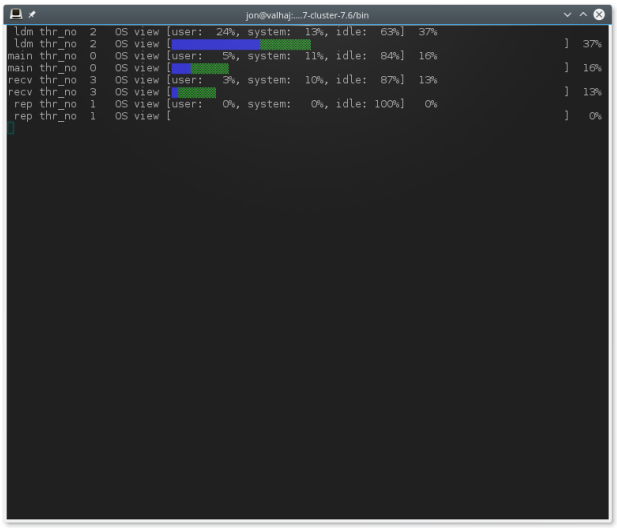
- --user[=*name*], -u

Property	Value
Command-Line Format	--user[=name]
Type	String
Default Value	root

Connect as this MySQL user.

Sample Output. The next figure shows ndb_top running in a terminal window on a Linux system with an ndbmtd data node under a moderate load. Here, the program has been invoked using ndb_top -n8 -x to provide both text and graph output:

Figure 22.25 ndb_top Running in Terminal



22.4.30 ndb_waiter — Wait for NDB Cluster to Reach a Given Status

ndb_waiter repeatedly (each 100 milliseconds) prints out the status of all cluster data nodes until either the cluster reaches a given status or the --timeout limit is exceeded, then exits. By default, it waits for the cluster to achieve STARTED status, in which all nodes have started and connected to the cluster. This can be overridden using the --no-contact and --not-started options.

The node states reported by this utility are as follows:

- NO_CONTACT: The node cannot be contacted.
- UNKNOWN: The node can be contacted, but its status is not yet known. Usually, this means that the node has received a START or RESTART command from the management server, but has not yet acted on it.
- NOT_STARTED: The node has stopped, but remains in contact with the cluster. This is seen when restarting the node using the management client's RESTART command.
- STARTING: The node's ndbd process has started, but the node has not yet joined the cluster.
- STARTED: The node is operational, and has joined the cluster.
- SHUTTING_DOWN: The node is shutting down.
- SINGLE USER MODE: This is shown for all cluster data nodes when the cluster is in single user mode.

The following table includes options that are specific to the NDB Cluster native backup restoration program ndb_waiter. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_waiter), see Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 22.350 Command-line options for the ndb_waiter program

Format	Description	Added, Deprecated, or Removed
--no-contact,	Wait for cluster to reach NO CONTACT state	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
-n		
--not-started	Wait for cluster to reach NOT STARTED state	All NDB 8.0 releases
--single-user	Wait for cluster to enter single user mode	All NDB 8.0 releases
--timeout=#, -t	Wait this many seconds, then exit whether or not cluster has reached desired state; default is 2 minutes (120 seconds)	All NDB 8.0 releases
--nowait-nodes=list	List of nodes not to be waited for.	All NDB 8.0 releases
--wait-nodes=list, -w	List of nodes to be waited for.	All NDB 8.0 releases

Usage

```
ndb_waiter [-c connection_string]
```

Additional Options

- --no-contact, -n

  Instead of waiting for the STARTED state, ndb_waiter continues running until the cluster reaches NO_CONTACT status before exiting.

- --not-started

  Instead of waiting for the STARTED state, ndb_waiter continues running until the cluster reaches NOT_STARTED status before exiting.

- --timeout=*seconds*, -t *seconds*

  Time to wait. The program exits if the desired state is not achieved within this number of seconds. The default is 120 seconds (1200 reporting cycles).

- --single-user

  The program waits for the cluster to enter single user mode.

- --nowait-nodes=*list*

  When this option is used, ndb_waiter does not wait for the nodes whose IDs are listed. The list is comma-delimited; ranges can be indicated by dashes, as shown here:

  ```
  shell> ndb_waiter --nowait-nodes=1,3,7-9
  ```

  Important

  Do *not* use this option together with the --wait-nodes option.

- --wait-nodes=*list*, -w *list*

When this option is used, ndb_waiter waits only for the nodes whose IDs are listed. The list is comma-delimited; ranges can be indicated by dashes, as shown here:

```
shell> ndb_waiter --wait-nodes=2,4-6,10
```

Important

Do *not* use this option together with the --nowait-nodes option.

Sample Output. Shown here is the output from ndb_waiter when run against a 4-node cluster in which two nodes have been shut down and then started again manually. Duplicate reports (indicated by ...) are omitted.

```
shell> ./ndb_waiter -c localhost

Connecting to mgmsrv at (localhost)
State node 1 STARTED
State node 2 NO_CONTACT
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 UNKNOWN
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 UNKNOWN
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED
```

```
...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTED
Waiting for cluster enter state STARTED

NDBT_ProgramExit: 0 - OK
```

Note

If no connection string is specified, then `ndb_waiter` tries to connect to a management on `localhost`, and reports `Connecting to mgmsrv at (null)`.

22.4.31 Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs

All NDB Cluster programs accept the options described in this section, with the following exceptions:

- `mysqld`
- `ndb_print_backup_file`
- `ndb_print_schema_file`
- `ndb_print_sys_file`

Note

Users of earlier NDB Cluster versions should note that some of these options have been changed to make them consistent with one another, and also with `mysqld`. You can use the `--help` option with any NDB Cluster program—with the exception of `ndb_print_backup_file`, `ndb_print_schema_file`, and `ndb_print_sys_file`—to view a list of the options which the program supports.

The options in the following table are common to all NDB Cluster executables (except those noted previously in this section).

Table 22.351 Command-line options common to all MySQL NDB Cluster programs

Format	Description	Added, Deprecated, or Removed
`--character-sets-dir=dir_name`	Directory where character sets are installed	All NDB 8.0 releases
`--connect-retries=#`	Set the number of times to retry a connection before giving up	All NDB 8.0 releases
`--connect-retry-delay=#`	Time to wait between attempts to contact a management server, in seconds	All NDB 8.0 releases
`--core-file`	Write core on errors (defaults to TRUE in debug builds)	All NDB 8.0 releases
`--debug=options`	Enable output from debug calls. Can be used only for versions compiled with debugging enabled	All NDB 8.0 releases
`--defaults-extra-file=filename`	Read this file after global option files are read	All NDB 8.0 releases

Format	Description	Added, Deprecated, or Removed
`--defaults-file=filename`	Read default options from this file	All NDB 8.0 releases
`--defaults-group-suffix`	Also read groups with names ending in this suffix	All NDB 8.0 releases
`--help`, `--usage`, `-?`	Display help message and exit	All NDB 8.0 releases
`--login-path=path`	Read this path from the login file	All NDB 8.0 releases
`--ndb-connectstring=connectstring`, `--connect-string=connectstring`, `-c`	Set connection string for connecting to ndb_mgmd. Syntax: [nodeid=<id>;] [host=]<hostname>[:<port>]. Overrides entries specified in NDB_CONNECTSTRING or my.cnf.	All NDB 8.0 releases
`--ndb-mgmd-host=host[:port]`	Set the host (and port, if desired) for connecting to management server	All NDB 8.0 releases
`--ndb-nodeid=#`	Set node id for this node	All NDB 8.0 releases
`--ndb-optimized-node-selection`	Select nodes for transactions in a more optimal way	All NDB 8.0 releases
`--no-defaults`	Do not read default options from any option file other than login file	All NDB 8.0 releases
`--print-defaults`	Print the program argument list and exit	All NDB 8.0 releases
`--version`, `-V`	Output version information and exit	All NDB 8.0 releases

For options specific to individual NDB Cluster programs, see Section 22.4, “NDB Cluster Programs”.

See MySQL Server Options for NDB Cluster, for `mysqld` options relating to NDB Cluster.

- `--character-sets-dir=`*`name`*

Property	Value
Command-Line Format	`--character-sets-dir=dir_name`
Type	Directory name
Default Value	

Tells the program where to find character set information.

- `--connect-retries=#`

Property	Value
Command-Line Format	`--connect-retries=#`

Property	Value
Type	Numeric
Default Value	`12`
Minimum Value	`0`
Maximum Value	`4294967295`

This option specifies the number of times following the first attempt to retry a connection before giving up (the client always tries the connection at least once). The length of time to wait per attempt is set using `--connect-retry-delay`.

Note

When used with `ndb_mgm`, this option has 3 as its default. See Section 22.4.5, "`ndb_mgm` — The NDB Cluster Management Client", for more information.

- `--connect-retry-delay=#`

Property	Value
Command-Line Format	`--connect-retry-delay=#`
Type	Numeric
Default Value	`5`
Minimum Value	`1`
Minimum Value	`0`
Maximum Value	`4294967295`

This option specifies the length of time to wait per attempt a connection before giving up. The number of times to try connecting is set by `--connect-retries`.

- `--core-file`

Property	Value
Command-Line Format	`--core-file`
Type	Boolean
Default Value	`FALSE`

Write a core file if the program dies. The name and location of the core file are system-dependent. (For NDB Cluster programs nodes running on Linux, the default location is the program's working directory—for a data node, this is the node's `DataDir`.) For some systems, there may be restrictions or limitations. For example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation for detailed information.

If NDB Cluster was built using the `--debug` option for `configure`, then `--core-file` is enabled by default. For regular builds, `--core-file` is disabled by default.

- `--debug[=options]`

Property	Value
Command-Line Format	`--debug=options`
Type	String

Property	Value
Default Value	`d:t:O,/tmp/ndb_restore.trace`

This option can be used only for versions compiled with debugging enabled. It is used to enable output from debug calls in the same manner as for the `mysqld` process.

- `--defaults-extra-file=`*`filename`*

Property	Value
Command-Line Format	`--defaults-extra-file=filename`
Type	String
Default Value	`[none]`

Read this file after global option files are read.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=`*`filename`*

Property	Value
Command-Line Format	`--defaults-file=filename`
Type	String
Default Value	`[none]`

Read default options from this file.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix`

Property	Value
Command-Line Format	`--defaults-group-suffix`
Type	String
Default Value	`[none]`

Also read groups with names ending in this suffix.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--help`, `--usage`, `-?`

Property	Value

Command-Line Format	`--help`
	
	`--usage`

Prints a short list with descriptions of the available command options.

- `--login-path=path`

Property	Value
Command-Line Format	`--login-path=path`
Type	String
Default Value	`[none]`

Read this path from the login file.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--ndb-connectstring=connection_string`, `--connect-string=connection_string`, `-c connection_string`

Property	Value

Command-Line Format	`--ndb-connectstring=connectstring`
	
	`--connect-string=connectstring`

Type	String
Default Value	`localhost:1186`

This option takes an NDB Cluster connection string that specifies the management server for the application to connect to, as shown here:

```
shell> ndbd --ndb-connectstring="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

For more information, see Section 22.3.3.3, “NDB Cluster Connection Strings”.

- `--ndb-mgmd-host=host[:port]`

Property	Value
Command-Line Format	`--ndb-mgmd-host=host[:port]`
Type	String
Default Value	`localhost:1186`

Can be used to set the host and port number of a single management server for the program to connect to. If the program requires node IDs or references to multiple management servers (or both) in its connection information, use the `--ndb-connectstring` option instead.

- `--ndb-nodeid=#`

Property	Value
Command-Line Format	`--ndb-nodeid=#`
Type	Numeric
Default Value	`0`

Sets this node's NDB Cluster node ID. *The range of permitted values depends on the node's type (data, management, or API) and the NDB Cluster software version*. See Section 22.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”, for more information.

- `--no-defaults`

Property	Value
Command-Line Format	`--no-defaults`
Type	Boolean
Default Value	`TRUE`

Do not read default options from any option file other than login file.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--ndb-optimized-node-selection`

Property	Value
Command-Line Format	`--ndb-optimized-node-selection`
Type	Boolean
Default Value	`TRUE`

Optimize selection of nodes for transactions. Enabled by default.

- `--print-defaults`

Property	Value
Command-Line Format	`--print-defaults`
Type	Boolean
Default Value	`TRUE`

Print the program argument list and exit.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--version`, `-V`

Property	Value
Command-Line Format	`--version`

Prints the NDB Cluster version number of the executable. The version number is relevant because not all versions can be used together, and the NDB Cluster startup process verifies that the versions of the binaries being used can co-exist in the same cluster. This is also important when performing an online (rolling) software upgrade or downgrade of NDB Cluster.

See Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”), for more information.

22.5 Management of NDB Cluster

Managing an NDB Cluster involves a number of tasks, the first of which is to configure and start NDB Cluster. This is covered in Section 22.3, “Configuration of NDB Cluster”, and Section 22.4, “NDB Cluster Programs”.

The next few sections cover the management of a running NDB Cluster.

For information about security issues relating to management and deployment of an NDB Cluster, see Section 22.5.12, “NDB Cluster Security Issues”.

There are essentially two methods of actively managing a running NDB Cluster. The first of these is through the use of commands entered into the management client whereby cluster status can be checked, log levels changed, backups started and stopped, and nodes stopped and started. The second method involves studying the contents of the cluster log `ndb_node_id_cluster.log`; this is usually found in the management server's `DataDir` directory, but this location can be overridden using the `LogDestination` option. (Recall that *`node_id`* represents the unique identifier of the node whose activity is being logged.) The cluster log contains event reports generated by `ndbd`. It is also possible to send cluster log entries to a Unix system log.

Some aspects of the cluster's operation can be also be monitored from an SQL node using the `SHOW ENGINE NDB STATUS` statement.

More detailed information about NDB Cluster operations is available in real time through an SQL interface using the `ndbinfo` database. For more information, see Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”.

NDB statistics counters provide improved monitoring using the `mysql` client. These counters, implemented in the NDB kernel, relate to operations performed by or affecting `Ndb` objects, such as starting, closing, and aborting transactions; primary key and unique key operations; table, range, and pruned scans; blocked threads waiting for various operations to complete; and data and events sent and received by NDB Cluster. The counters are incremented by the NDB kernel whenever NDB API calls are made or data is sent to or received by the data nodes.

`mysqld` exposes the NDB API statistics counters as system status variables, which can be identified from the prefix common to all of their names (`Ndb_api_`). The values of these variables can be read in the `mysql` client from the output of a `SHOW STATUS` statement, or by querying either the `SESSION_STATUS` table or the `GLOBAL_STATUS` table (in the `INFORMATION_SCHEMA` database). By comparing the values of the status variables before and after the execution of an SQL statement that acts on `NDB` tables, you can observe the actions taken on the NDB API level that correspond to this statement, which can be beneficial for monitoring and performance tuning of NDB Cluster.

MySQL Cluster Manager provides an advanced command-line interface that simplifies many otherwise complex NDB Cluster management tasks, such as starting, stopping, or restarting an NDB Cluster with a large number of nodes. The MySQL Cluster Manager client also supports commands for getting and setting the values of most node configuration parameters as well as `mysqld` server options and variables relating to NDB Cluster. See MySQL™ Cluster Manager 1.4.7 User Manual, for more information.

22.5.1 Summary of NDB Cluster Start Phases

This section provides a simplified outline of the steps involved when NDB Cluster data nodes are started. More complete information can be found in NDB Cluster Start Phases, in the *`NDB` Internals Guide*.

These phases are the same as those reported in the output from the *`node_id`* `STATUS` command in the management client (see Section 22.5.2, “Commands in the NDB Cluster Management Client”). These start phases are also reported in the `start_phase` column of the `ndbinfo.nodes` table.

Start types. There are several different startup types and modes, as shown in the following list:

- **Initial start.** The cluster starts with a clean file system on all data nodes. This occurs either when the cluster started for the very first time, or when all data nodes are restarted using the `--initial` option.

Note

Disk Data files are not removed when restarting a node using `--initial`.

- **System restart.** The cluster starts and reads data stored in the data nodes. This occurs when the cluster has been shut down after having been in use, when it is desired for the cluster to resume operations from the point where it left off.
- **Node restart.** This is the online restart of a cluster node while the cluster itself is running.
- **Initial node restart.** This is the same as a node restart, except that the node is reinitialized and started with a clean file system.

Setup and initialization (phase -1). Prior to startup, each data node (`ndbd` process) must be initialized. Initialization consists of the following steps:

1. Obtain a node ID
2. Fetch configuration data
3. Allocate ports to be used for inter-node communications
4. Allocate memory according to settings obtained from the configuration file

When a data node or SQL node first connects to the management node, it reserves a cluster node ID. To make sure that no other node allocates the same node ID, this ID is retained until the node has managed to connect to the cluster and at least one `ndbd` reports that this node is connected. This retention of the node ID is guarded by the connection between the node in question and `ndb_mgmd`.

After each data node has been initialized, the cluster startup process can proceed. The stages which the cluster goes through during this process are listed here:

- **Phase 0.** The `NDBFS` and `NDBCNTR` blocks start (see NDB Kernel Blocks). Data node file systems are cleared on those data nodes that were started with `--initial` option.
- **Phase 1.** In this stage, all remaining `NDB` kernel blocks are started. NDB Cluster connections are set up, inter-block communications are established, and heartbeats are started. In the case of a node restart, API node connections are also checked.

Note

When one or more nodes hang in Phase 1 while the remaining node or nodes hang in Phase 2, this often indicates network problems. One possible cause of such issues is one or more cluster hosts having multiple network interfaces. Another common source of problems causing this condition is the blocking of TCP/IP ports needed for communications between cluster nodes. In the latter case, this is often due to a misconfigured firewall.

- **Phase 2.** The `NDBCNTR` kernel block checks the states of all existing nodes. The master node is chosen, and the cluster schema file is initialized.
- **Phase 3.** The `DBLQH` and `DBTC` kernel blocks set up communications between them. The startup type is determined; if this is a restart, the `DBDIH` block obtains permission to perform the restart.
- **Phase 4.** For an initial start or initial node restart, the redo log files are created. The number of these files is equal to `NoOfFragmentLogFiles`.

For a system restart:

- Read schema or schemas.
- Read data from the local checkpoint.
- Apply all redo information until the latest restorable global checkpoint has been reached.

For a node restart, find the tail of the redo log.

- **Phase 5.** Most of the database-related portion of a data node start is performed during this phase. For an initial start or system restart, a local checkpoint is executed, followed by a global checkpoint. Periodic checks of memory usage begin during this phase, and any required node takeovers are performed.
- **Phase 6.** In this phase, node groups are defined and set up.
- **Phase 7.** The arbitrator node is selected and begins to function. The next backup ID is set, as is the backup disk write speed. Nodes reaching this start phase are marked as `Started`. It is now possible for API nodes (including SQL nodes) to connect to the cluster.
- **Phase 8.** If this is a system restart, all indexes are rebuilt (by `DBDIH`).
- **Phase 9.** The node internal startup variables are reset.
- **Phase 100 (OBSOLETE).** Formerly, it was at this point during a node restart or initial node restart that API nodes could connect to the node and begin to receive events. Currently, this phase is empty.
- **Phase 101.** At this point in a node restart or initial node restart, event delivery is handed over to the node joining the cluster. The newly-joined node takes over responsibility for delivering its primary data to subscribers. This phase is also referred to as *`SUMA` handover phase*.

After this process is completed for an initial start or system restart, transaction handling is enabled. For a node restart or initial node restart, completion of the startup process means that the node may now act as a transaction coordinator.

22.5.2 Commands in the NDB Cluster Management Client

In addition to the central configuration file, a cluster may also be controlled through a command-line interface available through the management client `ndb_mgm`. This is the primary administrative interface to a running cluster.

Commands for the event logs are given in Section 22.5.6, “Event Reports Generated in NDB Cluster”; commands for creating backups and restoring from them are provided in Section 22.5.3, “Online Backup of NDB Cluster”.

Using ndb_mgm with MySQL Cluster Manager. MySQL Cluster Manager handles starting and stopping processes and tracks their states internally, so it is not necessary to use `ndb_mgm` for these tasks for an NDB Cluster that is under MySQL Cluster Manager control. it is recommended *not* to use the `ndb_mgm` command-line client that comes with the NDB Cluster distribution to perform operations that involve starting or stopping nodes. These include but are not limited to the `START`, `STOP`, `RESTART`, and `SHUTDOWN` commands. For more information, see MySQL Cluster Manager Process Commands.

The management client has the following basic commands. In the listing that follows, *`node_id`* denotes either a data node ID or the keyword `ALL`, which indicates that the command should be applied to all of the cluster's data nodes.

- `HELP`

Displays information on all available commands.

- `CONNECT connection-string`

  Connects to the management server indicated by the connection string. If the client is already connected to this server, the client reconnects.

- `SHOW`

  Displays information on the cluster's status. Possible node status values include `UNKNOWN`, `NO_CONTACT`, `NOT_STARTED`, `STARTING`, `STARTED`, `SHUTTING_DOWN`, and `RESTARTING`. The output from this command also indicates when the cluster is in single user mode (status `SINGLE USER MODE`), and, in NDB 8.0.17 and later, indicates which API or SQL node has exclusive access when this mode is in effect.

- `node_id START`

  Brings online the data node identified by *`node_id`* (or all data nodes).

  `ALL START` works on all data nodes only, and does not affect management nodes.

  Important

  To use this command to bring a data node online, the data node must have been started using `--nostart` or `-n`.

- `node_id STOP [-a] [-f]`

  Stops the data or management node identified by *`node_id`*.

  Note

  `ALL STOP` works to stop all data nodes only, and does not affect management nodes.

  A node affected by this command disconnects from the cluster, and its associated `ndbd` or `ndb_mgmd` process terminates.

  The `-a` option causes the node to be stopped immediately, without waiting for the completion of any pending transactions.

  Normally, `STOP` fails if the result would cause an incomplete cluster. The `-f` option forces the node to shut down without checking for this. If this option is used and the result is an incomplete cluster, the cluster immediately shuts down.

  Warning

  Use of the `-a` option also disables the safety check otherwise performed when `STOP` is invoked to insure that stopping the node does not cause an incomplete cluster. In other words, you should exercise extreme care when using the `-a` option with the `STOP` command, due to the fact that this option makes it possible for the cluster to undergo a forced shutdown because it no longer has a complete copy of all data stored in `NDB`.

- `node_id RESTART [-n] [-i] [-a] [-f]`

  Restarts the data node identified by *`node_id`* (or all data nodes).

Using the `-i` option with `RESTART` causes the data node to perform an initial restart; that is, the node's file system is deleted and recreated. The effect is the same as that obtained from stopping the data node process and then starting it again using `ndbd --initial` from the system shell.

Note

Backup files and Disk Data files are not removed when this option is used.

Using the `-n` option causes the data node process to be restarted, but the data node is not actually brought online until the appropriate `START` command is issued. The effect of this option is the same as that obtained from stopping the data node and then starting it again using `ndbd --nostart` or `ndbd -n` from the system shell.

Using the `-a` causes all current transactions relying on this node to be aborted. No GCP check is done when the node rejoins the cluster.

Normally, `RESTART` fails if taking the node offline would result in an incomplete cluster. The `-f` option forces the node to restart without checking for this. If this option is used and the result is an incomplete cluster, the entire cluster is restarted.

- *`node_id`* `STATUS`

  Displays status information for the data node identified by *`node_id`* (or for all data nodes).

  The output from this command also indicates when the cluster is in single user mode.

- *`node_id`* `REPORT` *`report-type`*

  Displays a report of type *`report-type`* for the data node identified by *`node_id`*, or for all data nodes using `ALL`.

  Currently, there are three accepted values for *`report-type`*:

  - `BackupStatus` provides a status report on a cluster backup in progress
  - `MemoryUsage` displays how much data memory and index memory is being used by each data node as shown in this example:

    ```
    ndb_mgm> ALL REPORT MEMORY

    Node 1: Data usage is 5%(177 32K pages of total 3200)
    Node 1: Index usage is 0%(108 8K pages of total 12832)
    Node 2: Data usage is 5%(177 32K pages of total 3200)
    Node 2: Index usage is 0%(108 8K pages of total 12832)
    ```

    This information is also available from the `ndbinfo.memoryusage` table.

  - `EventLog` reports events from the event log buffers of one or more data nodes.

  `report-type` is case-insensitive and “fuzzy”; for `MemoryUsage`, you can use `MEMORY` (as shown in the prior example), `memory`, or even simply `MEM` (or `mem`). You can abbreviate `BackupStatus` in a similar fashion.

- `ENTER SINGLE USER MODE` *`node_id`*

Enters single user mode, whereby only the MySQL server identified by the node ID *`node_id`* is permitted to access the database.

Beginning with NDB 8.0.17, the `ndb_mgm` client provides a clear acknowledgement that this command has been issued and has taken effect, as shown here:

```
ndb_mgm> ENTER SINGLE USER MODE 100
Single user mode entered
Access is granted for API node 100 only.
```

Also in NDB 8.0.17 and later, the API or SQL node having exclusive access when in single user mode is indicated in the output of the `SHOW` command, like this:

```
ndb_mgm> SHOW
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=5    @127.0.0.1  (mysql-8.0.17 ndb-8.0.17, single user mode, Nodegroup: 0, *)
id=6    @127.0.0.1  (mysql-8.0.17 ndb-8.0.17, single user mode, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=50   @127.0.0.1  (mysql-8.0.17 ndb-8.0.17)

[mysqld(API)]   2 node(s)
id=100  @127.0.0.1  (mysql-8.0.17 ndb-8.0.17, allowed single user)
id=101 (not connected, accepting connect from any host)
```

Note

All data and management nodes must be running version 8.0.17 of the NDB Cluster software for this feature to be enabled.

- `EXIT SINGLE USER MODE`

  Exits single user mode, enabling all SQL nodes (that is, all running `mysqld` processes) to access the database.

  Note

  It is possible to use `EXIT SINGLE USER MODE` even when not in single user mode, although the command has no effect in this case.

- `QUIT`, `EXIT`

  Terminates the management client.

  This command does not affect any nodes connected to the cluster.

- `SHUTDOWN`

  Shuts down all cluster data nodes and management nodes. To exit the management client after this has been done, use `EXIT` or `QUIT`.

  This command does *not* shut down any SQL nodes or API nodes that are connected to the cluster.

- `CREATE NODEGROUP` *`nodeid`*`[, nodeid, ...]`

  Creates a new NDB Cluster node group and causes data nodes to join it.

This command is used after adding new data nodes online to an NDB Cluster, and causes them to join a new node group and thus to begin participating fully in the cluster. The command takes as its sole parameter a comma-separated list of node IDs—these are the IDs of the nodes just added and started that are to join the new node group. The number of nodes must be the same as the number of nodes in each node group that is already part of the cluster (each NDB Cluster node group must have the same number of nodes). In other words, if the NDB Cluster has 2 node groups of 2 data nodes each, then the new node group must also have 2 data nodes.

The node group ID of the new node group created by this command is determined automatically, and always the next highest unused node group ID in the cluster; it is not possible to set it manually.

For more information, see Section 22.5.15, “Adding NDB Cluster Data Nodes Online”.

- `DROP NODEGROUP nodegroup_id`

  Drops the NDB Cluster node group with the given *`nodegroup_id`*.

  This command can be used to drop a node group from an NDB Cluster. `DROP NODEGROUP` takes as its sole argument the node group ID of the node group to be dropped.

  `DROP NODEGROUP` acts only to remove the data nodes in the effected node group from that node group. It does not stop data nodes, assign them to a different node group, or remove them from the cluster's configuration. A data node that does not belong to a node group is indicated in the output of the management client `SHOW` command with `no nodegroup` in place of the node group ID, like this (indicated using bold text):

  ```
  id=3    @10.100.2.67  (8.0.19-ndb-8.0.19, no nodegroup)
  ```

  `DROP NODEGROUP` works only when all data nodes in the node group to be dropped are completely empty of any table data and table definitions. Since there is currently no way using `ndb_mgm` or the `mysql` client to remove all data from a specific data node or node group, this means that the command succeeds only in the two following cases:

  1. After issuing `CREATE NODEGROUP` in the `ndb_mgm` client, but before issuing any `ALTER TABLE ... REORGANIZE PARTITION` statements in the `mysql` client.
  2. After dropping all `NDBCLUSTER` tables using `DROP TABLE`.

     `TRUNCATE TABLE` does not work for this purpose because this removes only the table data; the data nodes continue to store an `NDBCLUSTER` table's definition until a `DROP TABLE` statement is issued that causes the table metadata to be dropped.

  For more information about `DROP NODEGROUP`, see Section 22.5.15, “Adding NDB Cluster Data Nodes Online”.

- `PROMPT [prompt]`

  Changes the prompt shown by `ndb_mgm` to the string literal *`prompt`*.

  `prompt` should not be quoted (unless you want the prompt to include the quotation marks). Unlike the case with the `mysql` client, special character sequences and escapes are not recognized. If called without an argument, the command resets the prompt to the default value (`ndb_mgm>`).

  Some examples are shown here:

  ```
  ```

```
ndb_mgm> PROMPT mgm#1:
mgm#1: SHOW
Cluster Configuration
...
mgm#1: PROMPT mymgm >
mymgm > PROMPT 'mymgm:'
'mymgm:' PROMPT  mymgm:
mymgm: PROMPT
ndb_mgm> EXIT
jon@valhaj:~/bin>
```

Note that leading spaces and spaces within the *`prompt`* string are not trimmed. Trailing spaces are removed.

- *`node_id`* `NODELOG DEBUG {ON|OFF}`

  Toggles debug logging in the node log, as though the effected data node or nodes had been started with the `--verbose` option. `NODELOG DEBUG ON` starts debug logging; `NODELOG DEBUG OFF` switches debug logging off.

Additional commands. A number of other commands available in the `ndb_mgm` client are described elsewhere, as shown in the following list:

- `START BACKUP` is used to perform an online backup in the `ndb_mgm` client; the `ABORT BACKUP` command is used to cancel a backup already in progress. For more information, see Section 22.5.3, “Online Backup of NDB Cluster”.
- The `CLUSTERLOG` command is used to perform various logging functions. See Section 22.5.6, “Event Reports Generated in NDB Cluster”, for more information and examples. `NODELOG DEBUG` activates or deactivates debug printouts in node logs, as described previously in this section.
- For testing and diagnostics work, the client supports a `DUMP` command which can be used to execute internal commands on the cluster. It should never be used in a production setting unless directed to do so by MySQL Support. For more information, see MySQL NDB Cluster Internals Manual.

22.5.3 Online Backup of NDB Cluster

The next few sections describe how to prepare for and then to create an NDB Cluster backup using the functionality for this purpose found in the `ndb_mgm` management client. To distinguish this type of backup from a backup made using `mysqldump`, we sometimes refer to it as a “native” NDB Cluster backup. (For information about the creation of backups with `mysqldump`, see Section 4.5.4, “`mysqldump` — A Database Backup Program”.) Restoration of NDB Cluster backups is done using the `ndb_restore` utility provided with the NDB Cluster distribution; for information about `ndb_restore` and its use in restoring NDB Cluster backups, see Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”.

Starting with NDB 8.0.16, it is possible to create backups using multiple LDMs to achieve parallelism on the data nodes. See Section 22.5.3.5, “Taking an NDB with Parallel Data Nodes”.

22.5.3.1 NDB Cluster Backup Concepts

A backup is a snapshot of the database at a given time. The backup consists of three main parts:

- **Metadata.** The names and definitions of all database tables
- **Table records.** The data actually stored in the database tables at the time that the backup was made
- **Transaction log.** A sequential record telling how and when data was stored in the database

Each of these parts is saved on all nodes participating in the backup. During backup, each node saves these three parts into three files on disk:

- BACKUP-*backup_id*.*node_id*.ctl

  A control file containing control information and metadata. Each node saves the same table definitions (for all tables in the cluster) to its own version of this file.

- BACKUP-*backup_id*-0.*node_id*.data

  A data file containing the table records, which are saved on a per-fragment basis. That is, different nodes save different fragments during the backup. The file saved by each node starts with a header that states the tables to which the records belong. Following the list of records there is a footer containing a checksum for all records.

- BACKUP-*backup_id*.*node_id*.log

  A log file containing records of committed transactions. Only transactions on tables stored in the backup are stored in the log. Nodes involved in the backup save different records because different nodes host different database fragments.

In the listing just shown, *backup_id* stands for the backup identifier and *node_id* is the unique identifier for the node creating the file.

The location of the backup files is determined by the BackupDataDir parameter.

22.5.3.2 Using The NDB Cluster Management Client to Create a Backup

Before starting a backup, make sure that the cluster is properly configured for performing one. (See Section 22.5.3.3, “Configuration for NDB Cluster Backups”.)

The START BACKUP command is used to create a backup:

```
START BACKUP [backup_id] [wait_option] [snapshot_option]

wait_option:
WAIT {STARTED | COMPLETED} | NOWAIT

snapshot_option:
SNAPSHOTSTART | SNAPSHOTEND
```

Successive backups are automatically identified sequentially, so the *backup_id*, an integer greater than or equal to 1, is optional; if it is omitted, the next available value is used. If an existing *backup_id* value is used, the backup fails with the error Backup failed: file already exists. If used, the *backup_id* must follow START BACKUP immediately, before any other options are used.

The *wait_option* can be used to determine when control is returned to the management client after a START BACKUP command is issued, as shown in the following list:

- If NOWAIT is specified, the management client displays a prompt immediately, as seen here:

  ```
  ndb_mgm> START BACKUP NOWAIT
  ndb_mgm>
  ```

  In this case, the management client can be used even while it prints progress information from the backup process.

- With WAIT STARTED the management client waits until the backup has started before returning control to the user, as shown here:

```
ndb_mgm> START BACKUP WAIT STARTED
Waiting for started, this may take several minutes
Node 2: Backup 3 started from node 1
ndb_mgm>
```

- **WAIT COMPLETED** causes the management client to wait until the backup process is complete before returning control to the user.

`WAIT COMPLETED` is the default.

A *`snapshot_option`* can be used to determine whether the backup matches the state of the cluster when `START BACKUP` was issued, or when it was completed. `SNAPSHOTSTART` causes the backup to match the state of the cluster when the backup began; `SNAPSHOTEND` causes the backup to reflect the state of the cluster when the backup was finished. `SNAPSHOTEND` is the default, and matches the behavior found in previous NDB Cluster releases.

Note

If you use the `SNAPSHOTSTART` option with `START BACKUP`, and the `CompressedBackup` parameter is enabled, only the data and control files are compressed—the log file is not compressed.

If both a *`wait_option`* and a *`snapshot_option`* are used, they may be specified in either order. For example, all of the following commands are valid, assuming that there is no existing backup having 4 as its ID:

```
START BACKUP WAIT STARTED SNAPSHOTSTART
START BACKUP SNAPSHOTSTART WAIT STARTED
START BACKUP 4 WAIT COMPLETED SNAPSHOTSTART
START BACKUP SNAPSHOTEND WAIT COMPLETED
START BACKUP 4 NOWAIT SNAPSHOTSTART
```

The procedure for creating a backup consists of the following steps:

1. Start the management client (`ndb_mgm`), if it not running already.
2. Execute the **START BACKUP** command. This produces several lines of output indicating the progress of the backup, as shown here:

   ```
   ndb_mgm> START BACKUP
   Waiting for completed, this may take several minutes
   Node 2: Backup 1 started from node 1
   Node 2: Backup 1 started from node 1 completed
    StartGCP: 177 StopGCP: 180
    #Records: 7362 #LogRecords: 0
    Data: 453648 bytes Log: 0 bytes
   ndb_mgm>
   ```

3. When the backup has started the management client displays this message:

   ```
   Backup backup_id started from node node_id
   ```

   `backup_id` is the unique identifier for this particular backup. This identifier is saved in the cluster log, if it has not been configured otherwise. *`node_id`* is the identifier of the management server that is coordinating the backup with the data nodes. At this point in the backup process the cluster has received and processed the backup request. It does not mean that the backup has finished. An example of this statement is shown here:

```
Node 2: Backup 1 started from node 1
```

4. The management client indicates with a message like this one that the backup has started:

   ```
   Backup backup_id started from node node_id completed
   ```

   As is the case for the notification that the backup has started, *`backup_id`* is the unique identifier for this particular backup, and *`node_id`* is the node ID of the management server that is coordinating the backup with the data nodes. This output is accompanied by additional information including relevant global checkpoints, the number of records backed up, and the size of the data, as shown here:

   ```
   Node 2: Backup 1 started from node 1 completed
    StartGCP: 177 StopGCP: 180
    #Records: 7362 #LogRecords: 0
    Data: 453648 bytes Log: 0 bytes
   ```

It is also possible to perform a backup from the system shell by invoking `ndb_mgm` with the `-e` or `--execute` option, as shown in this example:

```
shell> ndb_mgm -e "START BACKUP 6 WAIT COMPLETED SNAPSHOTSTART"
```

When using `START BACKUP` in this way, you must specify the backup ID.

Cluster backups are created by default in the `BACKUP` subdirectory of the `DataDir` on each data node. This can be overridden for one or more data nodes individually, or for all cluster data nodes in the `config.ini` file using the `BackupDataDir` configuration parameter. The backup files created for a backup with a given *`backup_id`* are stored in a subdirectory named `BACKUP-`*`backup_id`* in the backup directory.

Cancelling backups. To cancel or abort a backup that is already in progress, perform the following steps:

1. Start the management client.
2. Execute this command:

   ```
   ndb_mgm> ABORT BACKUP backup_id
   ```

   The number *`backup_id`* is the identifier of the backup that was included in the response of the management client when the backup was started (in the message `Backup` *`backup_id`* `started from node` *`management_node_id`*).

3. The management client will acknowledge the abort request with `Abort of backup` *`backup_id`* `ordered`.

   Note

   At this point, the management client has not yet received a response from the cluster data nodes to this request, and the backup has not yet actually been aborted.

4. After the backup has been aborted, the management client will report this fact in a manner similar to what is shown here:

   ```
   Node 1: Backup 3 started from 5 has been aborted.
     Error: 1321 - Backup aborted by user request: Permanent error: User defined error
   ```

```
Node 3: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
Node 2: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
Node 4: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
```

In this example, we have shown sample output for a cluster with 4 data nodes, where the sequence number of the backup to be aborted is 3, and the management node to which the cluster management client is connected has the node ID 5. The first node to complete its part in aborting the backup reports that the reason for the abort was due to a request by the user. (The remaining nodes report that the backup was aborted due to an unspecified internal error.)

Note

There is no guarantee that the cluster nodes respond to an `ABORT BACKUP` command in any particular order.

The `Backup backup_id started from node management_node_id has been aborted` messages mean that the backup has been terminated and that all files relating to this backup have been removed from the cluster file system.

It is also possible to abort a backup in progress from a system shell using this command:

```
shell> ndb_mgm -e "ABORT BACKUP backup_id"
```

Note

If there is no backup having the ID *`backup_id`* running when an `ABORT BACKUP` is issued, the management client makes no response, nor is it indicated in the cluster log that an invalid abort command was sent.

22.5.3.3 Configuration for NDB Cluster Backups

Five configuration parameters are essential for backup:

- `BackupDataBufferSize`

  The amount of memory used to buffer data before it is written to disk.

- `BackupLogBufferSize`

  The amount of memory used to buffer log records before these are written to disk.

- `BackupMemory`

  The total memory allocated in a data node for backups. This should be the sum of the memory allocated for the backup data buffer and the backup log buffer.

- `BackupWriteSize`

  The default size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

- `BackupMaxWriteSize`

  The maximum size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

More detailed information about these parameters can be found in Backup Parameters.

You can also set a location for the backup files using the `BackupDataDir` configuration parameter. The default is `FileSystemPath/BACKUP/BACKUP-backup_id`.

22.5.3.4 NDB Cluster Backup Troubleshooting

If an error code is returned when issuing a backup request, the most likely cause is insufficient memory or disk space. You should check that there is enough memory allocated for the backup.

Important

If you have set `BackupDataBufferSize` and `BackupLogBufferSize` and their sum is greater than 4MB, then you must also set `BackupMemory` as well.

You should also make sure that there is sufficient space on the hard drive partition of the backup target.

`NDB` does not support repeatable reads, which can cause problems with the restoration process. Although the backup process is “hot”, restoring an NDB Cluster from backup is not a 100% “hot” process. This is due to the fact that, for the duration of the restore process, running transactions get nonrepeatable reads from the restored data. This means that the state of the data is inconsistent while the restore is in progress.

22.5.3.5 Taking an NDB with Parallel Data Nodes

Beginning with NDB 8.0.16, it is possible to take a backup with multiple local data managers (LDMs) acting in parallel on the data nodes. For this to work, all data nodes in the cluster must use multiple LDMs, and each data node must use the same number of LDMs. This means that all data nodes must run `ndbmtd` (`ndbd` is single-threaded and thus always has only one LDM) and they must be configured to use multiple LDMs before taking the backup; ndbmtd by default runs in single-threaded mode. You can cause them to use multiple LDMs this by choosing an appropriate setting for one of the multi-threaded data node configuration parameters `MaxNoOfExecutionThreads` or `ThreadConfig`. Keep in mind that changing these parameters requires a restart of the cluster; this can be a rolling restart.

Depending on the number of LDMs and other factors, you may also need to increase `NoOfFragmentLogParts`. If you are using large Disk Data tables, you may also need to increase `DiskPageBufferMemory`. As with single-threaded backups, you also want or need to make adjustments to settings for `BackupDataBufferSize`, `BackupMemory`, and other configuration parameters relating to backups (see Backup parameters).

Once all data nodes are using multiple LDMs, you can take the parallel backup using the `START BACKUP` command in the NDB management client just as you would if the data nodes were running `ndbd` (or `ndbmtd` in single-threaded mode); no additional or special syntax is required, and you can specify a backup ID, wait option, or snapshot option in any combination as needed or desired.

Backups using multiple LDMs create subdirectories, one per LDM, under the directory `BACKUP/BACKUP-backup_id/` (which in turn resides under the `BackupDataDir`) on each data node; these subdirectories are named `BACKUP-backup_id-PART-1-OF-N/`, `BACKUP-backup_id-PART-2-OF-N/`, and so on, up to `BACKUP-backup_id-PART-N-OF-N/`, where *`backup_id`* is the backup ID used for this backup and *`N`* is the number of LDMs per data node. Each of these subdirectories contains the usual backup files `BACKUP-backup_id-0.node_id.Data`, `BACKUP-backup_id.node_id.ctl`, and `BACKUP-backup_id.node_id.log`, where *`node_id`* is the node ID of this data node.

In NDB 8.0.16 and later, `ndb_restore` automatically checks for the presence of the subdirectories just described; if it finds them, it attempts to restore the backup in parallel. For information about restoring backups taken with multiple LDMs, see Section 22.4.23.2, “Restoring from a backup taken in parallel”.

22.5.4 MySQL Server Usage for NDB Cluster

`mysqld` is the traditional MySQL server process. To be used with NDB Cluster, `mysqld` needs to be built with support for the `NDB` storage engine, as it is in the precompiled binaries available from https://dev.mysql.com/downloads/. If you build MySQL from source, you must invoke `CMake` with the `-DWITH_NDBCLUSTER=1` option to include support for `NDB`.

For more information about compiling NDB Cluster from source, see Section 22.2.2.4, “Building NDB Cluster from Source on Linux”, and Section 22.2.3.2, “Compiling and Installing NDB Cluster from Source on Windows”.

(For information about `mysqld` options and variables, in addition to those discussed in this section, which are relevant to NDB Cluster, see Section 22.3.3.9, “MySQL Server Options and Variables for NDB Cluster”.)

If the `mysqld` binary has been built with Cluster support, the `NDBCLUSTER` storage engine is still disabled by default. You can use either of two possible options to enable this engine:

- Use `--ndbcluster` as a startup option on the command line when starting `mysqld`.
- Insert a line containing `ndbcluster` in the `[mysqld]` section of your `my.cnf` file.

An easy way to verify that your server is running with the `NDBCLUSTER` storage engine enabled is to issue the `SHOW ENGINES` statement in the MySQL Monitor (`mysql`). You should see the value `YES` as the `Support` value in the row for `NDBCLUSTER`. If you see `NO` in this row or if there is no such row displayed in the output, you are not running an `NDB`-enabled version of MySQL. If you see `DISABLED` in this row, you need to enable it in either one of the two ways just described.

To read cluster configuration data, the MySQL server requires at a minimum three pieces of information:

- The MySQL server's own cluster node ID
- The host name or IP address for the management server (MGM node)
- The number of the TCP/IP port on which it can connect to the management server

Node IDs can be allocated dynamically, so it is not strictly necessary to specify them explicitly.

The `mysqld` parameter `ndb-connectstring` is used to specify the connection string either on the command line when starting `mysqld` or in `my.cnf`. The connection string contains the host name or IP address where the management server can be found, as well as the TCP/IP port it uses.

In the following example, `ndb_mgmd.mysql.com` is the host where the management server resides, and the management server listens for cluster messages on port 1186:

```
shell> mysqld --ndbcluster --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

See Section 22.3.3.3, “NDB Cluster Connection Strings”, for more information on connection strings.

Given this information, the MySQL server will be a full participant in the cluster. (We often refer to a `mysqld` process running in this manner as an SQL node.) It will be fully aware of all cluster data nodes as well as their status, and will establish connections to all data nodes. In this case, it is able to use any data node as a transaction coordinator and to read and update node data.

You can see in the `mysql` client whether a MySQL server is connected to the cluster using `SHOW PROCESSLIST`. If the MySQL server is connected to the cluster, and you have the `PROCESS` privilege, then the first row of the output is as shown here:

```
```

```
mysql> SHOW PROCESSLIST \G
*************************** 1. row ***************************
     Id: 1
   User: system user
   Host:
     db:
Command: Daemon
   Time: 1
  State: Waiting for event from ndbcluster
   Info: NULL
```

Important

To participate in an NDB Cluster, the `mysqld` process must be started with *both* the options `--ndbcluster` and `--ndb-connectstring` (or their equivalents in `my.cnf`). If `mysqld` is started with only the `--ndbcluster` option, or if it is unable to contact the cluster, it is not possible to work with `NDB` tables, *nor is it possible to create any new tables regardless of storage engine*. The latter restriction is a safety measure intended to prevent the creation of tables having the same names as `NDB` tables while the SQL node is not connected to the cluster. If you wish to create tables using a different storage engine while the `mysqld` process is not participating in an NDB Cluster, you must restart the server *without* the `--ndbcluster` option.

22.5.5 Performing a Rolling Restart of an NDB Cluster

This section discusses how to perform a *rolling restart* of an NDB Cluster installation, so called because it involves stopping and starting (or restarting) each node in turn, so that the cluster itself remains operational. This is often done as part of a *rolling upgrade* or *rolling downgrade*, where high availability of the cluster is mandatory and no downtime of the cluster as a whole is permissible. Where we refer to upgrades, the information provided here also generally applies to downgrades as well.

There are a number of reasons why a rolling restart might be desirable. These are described in the next few paragraphs.

Configuration change.
To make a change in the cluster's configuration, such as adding an SQL node to the cluster, or setting a configuration parameter to a new value.

NDB Cluster software upgrade or downgrade. To upgrade the cluster to a newer version of the NDB Cluster software (or to downgrade it to an older version). This is usually referred to as a “rolling upgrade” (or “rolling downgrade”, when reverting to an older version of NDB Cluster).

Change on node host. To make changes in the hardware or operating system on which one or more NDB Cluster node processes are running.

System reset (cluster reset).
To reset the cluster because it has reached an undesirable state. In such cases it is often desirable to reload the data and metadata of one or more data nodes. This can be done in any of three ways:

- Start each data node process (`ndbd` or possibly `ndbmtd`) with the `--initial` option, which forces the data node to clear its file system and to reload all NDB Cluster data and metadata from the other data nodes.
- Create a backup using the `ndb_mgm` client `START BACKUP` command prior to performing the restart. Following the upgrade, restore the node or nodes using `ndb_restore`.

  See Section 22.5.3, “Online Backup of NDB Cluster”, and Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”, for more information.

- Use `mysqldump` to create a backup prior to the upgrade; afterward, restore the dump using `LOAD DATA`.

Resource Recovery.
To free memory previously allocated to a table by successive `INSERT` and `DELETE` operations, for re-use by other NDB Cluster tables.

The process for performing a rolling restart may be generalized as follows:

1. Stop all cluster management nodes (`ndb_mgmd` processes), reconfigure them, then restart them. (See Rolling restarts with multiple management servers.)

2. Stop, reconfigure, then restart each cluster data node (`ndbd` process) in turn.

   Some node configuration parameters can be updated by issuing `RESTART` for each of the data nodes in the `ndb_mgm` client following the previous step; others require that the data node be stopped completely using a shell command (such as `kill` on most Unix systems) or the management client `STOP` command, then started again from a system shell by invoking the `ndbd` or `ndbmtd` executable as appropriate.

   Note

   On Windows, you can also use `SC STOP` and `SC START` commands, `NET STOP` and `NET START` commands, or the Windows Service Manager to stop and start nodes which have been installed as Windows services (see Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”).

   The type of restart required is indicated in the documentation for each node configuration parameter. See Section 22.3.3, “NDB Cluster Configuration Files”.

3. Stop, reconfigure, then restart each cluster SQL node (`mysqld` process) in turn.

NDB Cluster supports a somewhat flexible order for upgrading nodes. When upgrading an NDB Cluster, you may upgrade API nodes (including SQL nodes) before upgrading the management nodes, data nodes, or both. In other words, you are permitted to upgrade the API and SQL nodes in any order. This is subject to the following provisions:

- This functionality is intended for use as part of an online upgrade only. A mix of node binaries from different NDB Cluster releases is neither intended nor supported for continuous, long-term use in a production setting.

- All management nodes must be upgraded before any data nodes are upgraded. This remains true regardless of the order in which you upgrade the cluster's API and SQL nodes.

- Features specific to the “new” version must not be used until all management nodes and data nodes have been upgraded.

  This also applies to any MySQL Server version change that may apply, in addition to the NDB engine version change, so do not forget to take this into account when planning the upgrade. (This is true for online upgrades of NDB Cluster in general.)

See also Bug #48528 and Bug #49163.

Note

It is not possible for any API node to perform schema operations (such as data definition statements) during a node restart.

Rolling restarts with multiple management servers. When performing a rolling restart of an NDB Cluster with multiple management nodes, you should keep in mind that `ndb_mgmd` checks to see if any other management node is running, and, if so, tries to use that node's configuration data. To keep this from occurring, and to force `ndb_mgmd` to reread its configuration file, perform the following steps:

1. Stop all NDB Cluster `ndb_mgmd` processes.
2. Update all `config.ini` files.
3. Start a single `ndb_mgmd` with `--reload`, `--initial`, or both options as desired.
4. If you started the first `ndb_mgmd` with the `--initial` option, you must also start any remaining `ndb_mgmd` processes using `--initial`.

   Regardless of any other options used when starting the first `ndb_mgmd`, you should not start any remaining `ndb_mgmd` processes after the first one using `--reload`.
5. Complete the rolling restarts of the data nodes and API nodes as normal.

When performing a rolling restart to update the cluster's configuration, you can use the `config_generation` column of the `ndbinfo.nodes` table to keep track of which data nodes have been successfully restarted with the new configuration. See Section 22.5.10.28, “The ndbinfo nodes Table”.

22.5.6 Event Reports Generated in NDB Cluster

In this section, we discuss the types of event logs provided by NDB Cluster, and the types of events that are logged.

NDB Cluster provides two types of event log:

- The *cluster log*, which includes events generated by all cluster nodes. The cluster log is the log recommended for most uses because it provides logging information for an entire cluster in a single location.

  By default, the cluster log is saved to a file named `ndb_`*`node_id`*`_cluster.log`, (where *`node_id`* is the node ID of the management server) in the management server's `DataDir`.

  Cluster logging information can also be sent to `stdout` or a `syslog` facility in addition to or instead of being saved to a file, as determined by the values set for the `DataDir` and `LogDestination` configuration parameters. See Section 22.3.3.5, “Defining an NDB Cluster Management Server”, for more information about these parameters.

- *Node logs* are local to each node.

  Output generated by node event logging is written to the file `ndb_`*`node_id`*`_out.log` (where *`node_id`* is the node's node ID) in the node's `DataDir`. Node event logs are generated for both management nodes and data nodes.

  Node logs are intended to be used only during application development, or for debugging application code.

Both types of event logs can be set to log different subsets of events.

Each reportable event can be distinguished according to three different criteria:

- *Category*: This can be any one of the following values: `STARTUP`, `SHUTDOWN`, `STATISTICS`, `CHECKPOINT`, `NODERESTART`, `CONNECTION`, `ERROR`, or `INFO`.

- *Priority*: This is represented by one of the numbers from 0 to 15 inclusive, where 0 indicates “most important” and 15 “least important.”
- *Severity Level*: This can be any one of the following values: `ALERT`, `CRITICAL`, `ERROR`, `WARNING`, `INFO`, or `DEBUG`.

Both the cluster log and the node log can be filtered on these properties.

The format used in the cluster log is as shown here:

```
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Data usage is 2%(60 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Data usage is 2%(76 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Data usage is 2%(58 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Data usage is 2%(74 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 4: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 1: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 1: Node 9: API 8.0.19-ndb-8.0.19
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 2: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 2: Node 9: API 8.0.19-ndb-8.0.19
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 3: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 3: Node 9: API 8.0.19-ndb-8.0.19
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 4: Node 9: API 8.0.19-ndb-8.0.19
2007-01-26 19:59:22 [MgmSrvr] ALERT    -- Node 2: Node 7 Disconnected
2007-01-26 19:59:22 [MgmSrvr] ALERT    -- Node 2: Node 7 Disconnected
```

Each line in the cluster log contains the following information:

- A timestamp in *`YYYY-MM-DD HH:MM:SS`* format.
- The type of node which is performing the logging. In the cluster log, this is always `[MgmSrvr]`.
- The severity of the event.
- The ID of the node reporting the event.
- A description of the event. The most common types of events to appear in the log are connections and disconnections between different nodes in the cluster, and when checkpoints occur. In some cases, the description may contain status information.

22.5.6.1 NDB Cluster Logging Management Commands

`ndb_mgm` supports a number of management commands related to the cluster log and node logs. In the listing that follows, *`node_id`* denotes either a storage node ID or the keyword `ALL`, which indicates that the command should be applied to all of the cluster's data nodes.

- `CLUSTERLOG ON`

  Turns the cluster log on.

- `CLUSTERLOG OFF`

  Turns the cluster log off.

- `CLUSTERLOG INFO`

Provides information about cluster log settings.

- *node_id* CLUSTERLOG *category=threshold*

  Logs *category* events with priority less than or equal to *threshold* in the cluster log.

- CLUSTERLOG FILTER *severity_level*

  Toggles cluster logging of events of the specified *severity_level*.

The following table describes the default setting (for all data nodes) of the cluster log category threshold. If an event has a priority with a value lower than or equal to the priority threshold, it is reported in the cluster log.

Note

Events are reported per data node, and that the threshold can be set to different values on different nodes.

Table 22.352 Cluster log categories, with default threshold setting

Category	Default threshold (All data nodes)
STARTUP	7
SHUTDOWN	7
STATISTICS	7
CHECKPOINT	7
NODERESTART	7
CONNECTION	7
ERROR	15
INFO	7

The STATISTICS category can provide a great deal of useful data. See Section 22.5.6.3, “Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client”, for more information.

Thresholds are used to filter events within each category. For example, a STARTUP event with a priority of 3 is not logged unless the threshold for STARTUP is set to 3 or higher. Only events with priority 3 or lower are sent if the threshold is 3.

The following table shows the event severity levels.

Note

These correspond to Unix syslog levels, except for LOG_EMERG and LOG_NOTICE, which are not used or mapped.

Table 22.353 Event severity levels

Severity Level Value	Severity	Description
1	ALERT	A condition that should be corrected immediately, such as a corrupted system database
2	CRITICAL	Critical conditions, such as device errors or insufficient resources

Severity Level Value	Severity	Description
3	ERROR	Conditions that should be corrected, such as configuration errors
4	WARNING	Conditions that are not errors, but that might require special handling
5	INFO	Informational messages
6	DEBUG	Debugging messages used for NDBCLUSTER development

Event severity levels can be turned on or off (using CLUSTERLOG FILTER—see above). If a severity level is turned on, then all events with a priority less than or equal to the category thresholds are logged. If the severity level is turned off then no events belonging to that severity level are logged.

Important

Cluster log levels are set on a per ndb_mgmd, per subscriber basis. This means that, in an NDB Cluster with multiple management servers, using a CLUSTERLOG command in an instance of ndb_mgm connected to one management server affects only logs generated by that management server but not by any of the others. This also means that, should one of the management servers be restarted, only logs generated by that management server are affected by the resetting of log levels caused by the restart.

22.5.6.2 NDB Cluster Log Events

An event report reported in the event logs has the following format:

```
datetime [string] severity -- message
```

For example:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

This section discusses all reportable events, ordered by category and severity level within each category.

In the event descriptions, GCP and LCP mean “Global Checkpoint” and “Local Checkpoint”, respectively.

CONNECTION Events

These events are associated with connections between Cluster nodes.

Table 22.354 Events associated with connections between cluster nodes

Event	Priority	Severity Level	Description
Connected	8	INFO	Data nodes connected
Disconnected	8	ALERT	Data nodes disconnected
CommunicationClosed	8	INFO	SQL node or data node connection closed
CommunicationOpened	8	INFO	SQL node or data node connection open
ConnectedApiVersion	8	INFO	Connection using API version

CHECKPOINT Events

The logging messages shown here are associated with checkpoints.

Table 22.355 Events associated with checkpoints

Event	Priority	Severity Level	Description
GlobalCheckpointStarted	9	INFO	Start of GCP: REDO log is written to disk
GlobalCheckpointCompleted	10	INFO	GCP finished
LocalCheckpointStarted	7	INFO	Start of LCP: data written to disk
LocalCheckpointCompleted	7	INFO	LCP completed normally
LCPStoppedInCalcKeepGci	0	ALERT	LCP stopped
LCPFragmentCompleted	11	INFO	LCP on a fragment has been completed
UndoLogBlocked	7	INFO	UNDO logging blocked; buffer near overflow
RedoStatus	7	INFO	Redo status

STARTUP Events

The following events are generated in response to the startup of a node or of the cluster and of its success or failure. They also provide information relating to the progress of the startup process, including information concerning logging activities.

Table 22.356 Events relating to the startup of a node or cluster

Event	Priority	Severity Level	Description
NDBStartStarted	1	INFO	Data node start phases initiated (all nodes starting)
NDBStartCompleted	1	INFO	Start phases completed, all data nodes
STTORRYRecieved	15	INFO	Blocks received after completion of restart
StartPhaseCompleted	4	INFO	Data node start phase *X* completed
CM_REGCONF	3	INFO	Node has been successfully included into the cluster; shows the node, managing node, and dynamic ID
CM_REGREF	8	INFO	Node has been refused for inclusion in the cluster; cannot be included in cluster due to misconfiguration, inability to establish communication, or other problem
FIND_NEIGHBOURS	8	INFO	Shows neighboring data nodes
NDBStopStarted	1	INFO	Data node shutdown initiated
NDBStopCompleted	1	INFO	Data node shutdown complete
NDBStopForced	1	ALERT	Forced shutdown of data node
NDBStopAborted	1	INFO	Unable to shut down data node normally
StartREDOLog	4	INFO	New redo log started; GCI keep *X*, newest restorable GCI *Y*
StartLog	10	INFO	New log started; log part *X*, start MB *Y*, stop MB *Z*
UNDORecordsExecuted	15	INFO	Undo records executed

Event	Priority	Severity Level	Description
StartReport	4	INFO	Report started
LogFileInitStatus	7	INFO	Log file initialization status
LogFileInitCompStatus	7	INFO	Log file completion status
StartReadLCP	10	INFO	Start read for local checkpoint
ReadLCPComplete	10	INFO	Read for local checkpoint completed
RunRedo	8	INFO	Running the redo log
RebuildIndex	10	INFO	Rebuilding indexes

NODERESTART Events

The following events are generated when restarting a node and relate to the success or failure of the node restart process.

Table 22.357 Events relating to restarting a node

Event	Priority	Severity Level	Description
NR_CopyDict	7	INFO	Completed copying of dictionary information
NR_CopyDistr	7	INFO	Completed copying distribution information
NR_CopyFragsStarted	7	INFO	Starting to copy fragments
NR_CopyFragDone	10	INFO	Completed copying a fragment
NR_CopyFragsCompleted	7	INFO	Completed copying all fragments
NodeFailCompleted	8	ALERT	Node failure phase completed
NODE_FAILREP	8	ALERT	Reports that a node has failed
ArbitState	6	INFO	Report whether an arbitrator is found or not; there are seven different possible outcomes when seeking an arbitrator, listed here: • Management server restarts arbitration thread [state=*X*] • Prepare arbitrator node *X* [ticket=*Y*] • Receive arbitrator node *X* [ticket=*Y*] • Started arbitrator node *X* [ticket=*Y*] • Lost arbitrator node *X* - process failure [state=*Y*] • Lost arbitrator node *X* - process exit [state=*Y*] • Lost arbitrator node *X* <error msg> [state=*Y*]
ArbitResult	2	ALERT	Report arbitrator results; there are eight different possible results for arbitration attempts, listed here:

Event	Priority	Severity Level	Description
			• Arbitration check failed: less than 1/2 nodes left • Arbitration check succeeded: node group majority • Arbitration check failed: missing node group • Network partitioning: arbitration required • Arbitration succeeded: affirmative response from node *X* • Arbitration failed: negative response from node *X* • Network partitioning: no arbitrator available • Network partitioning: no arbitrator configured
GCP_TakeoverStarted	7	INFO	GCP takeover started
GCP_TakeoverCompleted	7	INFO	GCP takeover complete
LCP_TakeoverStarted	7	INFO	LCP takeover started
LCP_TakeoverCompleted	7	INFO	LCP takeover complete (state = *X*)
ConnectCheckStarted	6	INFO	Connection check started
ConnectCheckCompleted	6	INFO	Connection check completed
NodeFailRejected	6	ALERT	Node failure phase failed

STATISTICS Events

The following events are of a statistical nature. They provide information such as numbers of transactions and other operations, amount of data sent or received by individual nodes, and memory usage.

Table 22.358 Events of a statistical nature

Event	Priority	Severity Level	Description
TransReportCounters	8	INFO	Report transaction statistics, including numbers of transactions, commits, reads, simple reads, writes, concurrent operations, attribute information, and aborts
OperationReportCounters	8	INFO	Number of operations
TableCreated	7	INFO	Report number of tables created
JobStatistic	9	INFO	Mean internal job scheduling statistics
ThreadConfigLoop	9	INFO	Number of thread configuration loops
SendBytesStatistic	9	INFO	Mean number of bytes sent to node *X*
ReceiveBytesStatistic	9	INFO	Mean number of bytes received from node *X*
MemoryUsage	5	INFO	Data and index memory usage (80%, 90%, and 100%)

Event	Priority	Severity Level	Description
MTSignalStatistics	9	INFO	Multithreaded signals

SCHEMA Events

These events relate to NDB Cluster schema operations.

Table 22.359 Events relating to NDB Cluster schema operations

Event	Priority	Severity Level	Description
CreateSchemaObject	8	INFO	Schema objected created
AlterSchemaObject	8	INFO	Schema object updated
DropSchemaObject	8	INFO	Schema object dropped

ERROR Events

These events relate to Cluster errors and warnings. The presence of one or more of these generally indicates that a major malfunction or failure has occurred.

Table 22.360 Events relating to cluster errors and warnings

Event	Priority	Severity Level	Description
TransporterError	2	ERROR	Transporter error
TransporterWarning	8	WARNING	Transporter warning
MissedHeartbeat	8	WARNING	Node *X* missed heartbeat number *Y*
DeadDueToHeartbeat	8	ALERT	Node *X* declared “dead” due to missed heartbeat
WarningEvent	2	WARNING	General warning event
SubscriptionStatus	4	WARNING	Change in subscription status

INFO Events

These events provide general information about the state of the cluster and activities associated with Cluster maintenance, such as logging and heartbeat transmission.

Table 22.361 Information events

Event	Priority	Severity Level	Description
SentHeartbeat	12	INFO	Sent heartbeat
CreateLogBytes	11	INFO	Create log: Log part, log file, size in MB
InfoEvent	2	INFO	General informational event
EventBufferStatus	7	INFO	Event buffer status
EventBufferStatus2	7	INFO	Improved event buffer status information

Note

SentHeartbeat events are available only if NDB Cluster was compiled with VM_TRACE enabled.

SINGLEUSER Events

These events are associated with entering and exiting single user mode.

Table 22.362 Events relating to single user mode

Event	Priority	Severity Level	Description
SingleUser	7	INFO	Entering or exiting single user mode

BACKUP Events

These events provide information about backups being created or restored.

Table 22.363 Backup events

Event	Priority	Severity Level	Description
BackupStarted	7	INFO	Backup started
BackupStatus	7	INFO	Backup status
BackupCompleted	7	INFO	Backup completed
BackupFailedToStart	7	ALERT	Backup failed to start
BackupAborted	7	ALERT	Backup aborted by user
RestoreStarted	7	INFO	Started restoring from backup
RestoreMetaData	7	INFO	Restoring metadata
RestoreData	7	INFO	Restoring data
RestoreLog	7	INFO	Restoring log files
RestoreCompleted	7	INFO	Completed restoring from backup
SavedEvent	7	INFO	Event saved

22.5.6.3 Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client

The NDB management client's CLUSTERLOG STATISTICS command can provide a number of useful statistics in its output. Counters providing information about the state of the cluster are updated at 5-second reporting intervals by the transaction coordinator (TC) and the local query handler (LQH), and written to the cluster log.

Transaction coordinator statistics. Each transaction has one transaction coordinator, which is chosen by one of the following methods:

- In a round-robin fashion
- By communication proximity
- By supplying a data placement hint when the transaction is started

Note

You can determine which TC selection method is used for transactions started from a given SQL node using the ndb_optimized_node_selection system variable.

All operations within the same transaction use the same transaction coordinator, which reports the following statistics:

- **Trans count.** This is the number transactions started in the last interval using this TC as the transaction coordinator. Any of these transactions may have committed, have been aborted, or remain uncommitted at the end of the reporting interval.

  Note

  Transactions do not migrate between TCs.

- **Commit count.** This is the number of transactions using this TC as the transaction coordinator that were committed in the last reporting interval. Because some transactions committed in this reporting interval may have started in a previous reporting interval, it is possible for `Commit count` to be greater than `Trans count`.

- **Read count.** This is the number of primary key read operations using this TC as the transaction coordinator that were started in the last reporting interval, including simple reads. This count also includes reads performed as part of unique index operations. A unique index read operation generates 2 primary key read operations—1 for the hidden unique index table, and 1 for the table on which the read takes place.

- **Simple read count.** This is the number of simple read operations using this TC as the transaction coordinator that were started in the last reporting interval.

- **Write count.** This is the number of primary key write operations using this TC as the transaction coordinator that were started in the last reporting interval. This includes all inserts, updates, writes and deletes, as well as writes performed as part of unique index operations.

  Note

  A unique index update operation can generate multiple PK read and write operations on the index table and on the base table.

- **AttrInfoCount.** This is the number of 32-bit data words received in the last reporting interval for primary key operations using this TC as the transaction coordinator. For reads, this is proportional to the number of columns requested. For inserts and updates, this is proportional to the number of columns written, and the size of their data. For delete operations, this is usually zero.

  Unique index operations generate multiple PK operations and so increase this count. However, data words sent to describe the PK operation itself, and the key information sent, are *not* counted here. Attribute information sent to describe columns to read for scans, or to describe ScanFilters, is also not counted in `AttrInfoCount`.

- **Concurrent Operations.** This is the number of primary key or scan operations using this TC as the transaction coordinator that were started during the last reporting interval but that were not completed. Operations increment this counter when they are started and decrement it when they are completed; this occurs after the transaction commits. Dirty reads and writes—as well as failed operations—decrement this counter.

  The maximum value that `Concurrent Operations` can have is the maximum number of operations that a TC block can support; currently, this is `(2 * MaxNoOfConcurrentOperations) + 16 + MaxNoOfConcurrentTransactions`. (For more information about these configuration parameters, see the *Transaction Parameters* section of Section 22.3.3.6, “Defining NDB Cluster Data Nodes”.)

- **Abort count.** This is the number of transactions using this TC as the transaction coordinator that were aborted during the last reporting interval. Because some transactions that were aborted in the last reporting interval may have started in a previous reporting interval, `Abort count` can sometimes be greater than `Trans count`.

- **Scans.** This is the number of table scans using this TC as the transaction coordinator that were started during the last reporting interval. This does not include range scans (that is, ordered index scans).
- **Range scans.** This is the number of ordered index scans using this TC as the transaction coordinator that were started in the last reporting interval.
- **Local reads.** This is the number of primary-key read operations performed using a transaction coordinator on a node that also holds the primary replica of the record. This count can also be obtained from the `LOCAL_READS` counter in the `ndbinfo.counters` table.
- **Local writes.** This contains the number of primary-key read operations that were performed using a transaction coordinator on a node that also holds the primary replica of the record. This count can also be obtained from the `LOCAL_WRITES` counter in the `ndbinfo.counters` table.

Local query handler statistics (Operations). There is 1 cluster event per local query handler block (that is, 1 per data node process). Operations are recorded in the LQH where the data they are operating on resides.

Note

A single transaction may operate on data stored in multiple LQH blocks.

The `Operations` statistic provides the number of local operations performed by this LQH block in the last reporting interval, and includes all types of read and write operations (insert, update, write, and delete operations). This also includes operations used to replicate writes. For example, in a 2-replica cluster, the write to the primary replica is recorded in the primary LQH, and the write to the backup will be recorded in the backup LQH. Unique key operations may result in multiple local operations; however, this does *not* include local operations generated as a result of a table scan or ordered index scan, which are not counted.

Process scheduler statistics. In addition to the statistics reported by the transaction coordinator and local query handler, each `ndbd` process has a scheduler which also provides useful metrics relating to the performance of an NDB Cluster. This scheduler runs in an infinite loop; during each loop the scheduler performs the following tasks:

1. Read any incoming messages from sockets into a job buffer.
2. Check whether there are any timed messages to be executed; if so, put these into the job buffer as well.
3. Execute (in a loop) any messages in the job buffer.
4. Send any distributed messages that were generated by executing the messages in the job buffer.
5. Wait for any new incoming messages.

Process scheduler statistics include the following:

- **Mean Loop Counter.** This is the number of loops executed in the third step from the preceding list. This statistic increases in size as the utilization of the TCP/IP buffer improves. You can use this to monitor changes in performance as you add new data node processes.
- **Mean send size and Mean receive size.** These statistics enable you to gauge the efficiency of, respectively writes and reads between nodes. The values are given in bytes. Higher values mean a lower cost per byte sent or received; the maximum value is 64K.

To cause all cluster log statistics to be logged, you can use the following command in the NDB management client:

```
ndb_mgm> ALL CLUSTERLOG STATISTICS=15
```

Note

Setting the threshold for STATISTICS to 15 causes the cluster log to become very verbose, and to grow quite rapidly in size, in direct proportion to the number of cluster nodes and the amount of activity in the NDB Cluster.

For more information about NDB Cluster management client commands relating to logging and reporting, see Section 22.5.6.1, “NDB Cluster Logging Management Commands”.

22.5.7 NDB Cluster Log Messages

This section contains information about the messages written to the cluster log in response to different cluster log events. It provides additional, more specific information on NDB transporter errors.

22.5.7.1 NDB Cluster: Messages in the Cluster Log

The following table lists the most common NDB cluster log messages. For information about the cluster log, log events, and event types, see Section 22.5.6, “Event Reports Generated in NDB Cluster”. These log messages also correspond to log event types in the MGM API; see The Ndb_logevent_type Type, for related information of interest to Cluster API developers.

Table 22.364 Common NDB cluster log messages

Log Message	Description	Event Name	Event Type	Priority	Severity
Node *mgm_node_id*: Node *data_node_id* Connected	The data node having node ID *node_id* has connected to the management server (node *mgm_node_id*).	Connected	Connection	8	INFO
Node *mgm_node_id*: Node *data_node_id* Disconnected	The data node having node ID *data_node_id* has disconnected from the management server (node *mgm_node_id*).	Disconnected	Connection	8	ALERT
Node *data_node_id*: Communication to Node *api_node_id* closed	The API node or SQL node having node ID *api_node_id* is no longer communicating with data node *data_node_id*.	CommunicationClosed	Connection	8	INFO
Node *data_node_id*:	The API node or SQL node	CommunicationOpened	Connection	8	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
Communication to Node *api_node_id* opened	having node ID *api_node_id* is now communicating with data node *data_node_id*.				
Node *mgm_node_id*: Node *api_node_id*: API *version*	The API node having node ID *api_node_id* has connected to management node *mgm_node_id* using NDB API version *version* (generally the same as the MySQL version number).	ConnectedApiVersion	Connection	8	INFO
Node *node_id*: Global checkpoint *gci* started	A global checkpoint with the ID *gci* has been started; node *node_id* is the master responsible for this global checkpoint.	GlobalCheckpointStarted	Checkpoint	9	INFO
Node *node_id*: Global checkpoint *gci* completed	The global checkpoint having the ID *gci* has been completed; node *node_id* was the master responsible for this global checkpoint.	GlobalCheckpointCompleted	Checkpoint	10	INFO
Node *node_id*: Local checkpoint *lcp* started. Keep GCI = *current_gci* oldest restorable GCI = *old_gci*	The local checkpoint having sequence ID *lcp* has been started on node *node_id*. The most recent GCI that can be used has the index *current_gci*, and the oldest GCI from which the cluster can be restored has the index *old_gci*.	LocalCheckpointStarted	Checkpoint	7	INFO
Node *node_id*: Local checkpoint *lcp* completed	The local checkpoint having sequence ID *lcp* on node *node_id* has been completed.	LocalCheckpointCompleted	Checkpoint	8	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
Node *node_id*: Local Checkpoint stopped in CALCULATED_KEEP_GCI	The node was unable to determine the most recent usable GCI.	LCPStoppedInCalcKeepGci	Checkpoint	0	ALERT
Node *node_id*: Table ID = *table_id*, fragment *ID* = *fragment_id* has completed LCP on Node *node_id* maxGciStarted: *started_gci* maxGciCompleted: *completed_gci*	A table fragment has been checkpointed to disk on node *node_id*. The GCI in progress has the index *started_gci*, and the most recent GCI to have been completed has the index *completed_gci*.	LCPFragmentCompleted	Checkpoint	11	INFO
Node *node_id*: ACC Blocked *num_1* and TUP Blocked *num_2* times last second	Undo logging is blocked because the log buffer is close to overflowing.	UndoLogBlocked	Checkpoint	7	INFO
Node *node_id*: Start initiated *version*	Data node *node_id*, running NDB version *version*, is beginning its startup process.	NDBStartStarted	StartUp	1	INFO
Node *node_id*: Started *version*	Data node *node_id*, running NDB version *version*, has started successfully.	NDBStartCompleted	StartUp	1	INFO
Node *node_id*: STTORRY received after restart finished	The node has received a signal indicating that a cluster restart has completed.	STTORRYRecieved	StartUp	15	INFO
Node *node_id*: Start phase *phase* completed (*type*)	The node has completed start phase *phase* of a *type* start. For a listing of start phases, see Section 22.5.1, “Summary of NDB Cluster Start Phases”. (*type* is	StartPhaseCompleted	StartUp	4	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
	one of initial, system, node, initial node, or <Unknown>.)				
Node *node_id*: CM_REGCONF president = *president_id*, own Node = *own_id*, our dynamic id = *dynamic_id*	Node *president_id* has been selected as "president". *own_id* and *dynamic_id* should always be the same as the ID (*node_id*) of the reporting node.	CM_REGCONF	StartUp	3	INFO
Node *node_id*: CM_REGREF from Node *president_id* to our Node *node_id*. Cause = *cause*	The reporting node (ID *node_id*) was unable to accept node *president_id* as president. The *cause* of the problem is given as one of Busy, Election with wait = false, Not president, Election without selecting new candidate, or No such cause.	CM_REGREF	StartUp	8	INFO
Node *node_id*: We are Node *own_id* with dynamic ID *dynamic_id*, our left neighbor is Node *id_1*, our right is Node *id_2*	The node has discovered its neighboring nodes in the cluster (node *id_1* and node *id_2*). *node_id*, *own_id*, and *dynamic_id* should always be the same; if they are not, this indicates a serious misconfiguration of the cluster nodes.	FIND_NEIGHBOURS	StartUp	8	INFO
Node *node_id*: *type* shutdown initiated	The node has received a shutdown signal. The *type* of	NDBStopStarted	StartUp	1	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
	shutdown is either Cluster or Node.				
Node *node_id*: Node shutdown completed [, *action*] [Initiated by signal *signal*.]	The node has been shut down. This report may include an *action*, which if present is one of restarting, no start, or initial. The report may also include a reference to an NDB Protocol *signal*; for possible signals, refer to Operations and Signals.	NDBStopCompleted	StartUp	1	INFO
Node *node_id*: Forced node shutdown completed [, action]. [Occurred during startphase *start_phase*.] [Initiated by *signal*.] [Caused by error *error_code*: '*error_message*(*error_classification*): *error_status*'. [(extra info *extra_code*)]]	The node has been forcibly shut down. The *action* (one of restarting, no start, or initial) subsequently being taken, if any, is also reported. If the shutdown occurred while the node was starting, the report includes the *start_phase* during which the node failed. If this was a result of a *signal* sent to the node, this information is also provided (see Operations and Signals, for more information). If the error causing the failure is known, this is also included; for more information about NDB error messages and classifications, see NDB Cluster API Errors.	NDBStopForced	StartUp	1	ALERT

Log Message	Description	Event Name	Event Type	Priority	Severity
`Node node_id: Node shutdown aborted`	The node shutdown process was aborted by the user.	`NDBStopAborted`	`StartUp`	1	`INFO`
`Node node_id: StartLog: [GCI Keep: keep_pos LastCompleted: last_pos NewestRestorable restore_pos]`	This reports global checkpoints referenced during a node start. The redo log prior to *`keep_pos`* is dropped. *`last_pos`* is the last global checkpoint in which data node the participated; *`restore_pos`* is the global checkpoint which is actually used to restore all data nodes.	`StartREDOLog`	`StartUp`	4	`INFO`
`startup_message` [*Listed separately; see below.*]	There are a number of possible startup messages that can be logged under different circumstances. These are listed separately; see Section 22.5.7.2, “NDB Cluster Log Startup Messages”.	`StartReport`	`StartUp`	4	`INFO`
`Node node_id: Node restart completed copy of dictionary information`	Copying of data dictionary information to the restarted node has been completed.	`NR_CopyDict`	`NodeRestart`	8	`INFO`
`Node node_id: Node restart completed copy of distribution information`	Copying of data distribution information to the restarted node has been completed.	`NR_CopyDistr`	`NodeRestart`	8	`INFO`
`Node node_id: Node restart starting to copy the fragments to Node node_id`	Copy of fragments to starting data node *`node_id`* has begun	`NR_CopyFragsStarted`	`NodeRestart`	8	`INFO`
`Node node_id: Table ID = table_id,`	Fragment *`fragment_id`* from table *`table_id`*	`NR_CopyFragDone`	`NodeRestart`	10	`INFO`

Log Message	Description	Event Name	Event Type	Priority	Severity
`fragment ID = fragment_id have been copied to Node node_id`	has been copied to data node *node_id*				
`Node node_id: Node restart completed copying the fragments to Node node_id`	Copying of all table fragments to restarting data node *node_id* has been completed	`NR_CopyFragsCompleted`	`NodeRestart`	8	`INFO`
`Node node_id: Node node1_id completed failure of Node node2_id`	Data node *node1_id* has detected the failure of data node *node2_id*	`NodeFailCompleted`	`NodeRestart`	8	`ALERT`
`All nodes completed failure of Node node_id`	All (remaining) data nodes have detected the failure of data node *node_id*	`NodeFailCompleted`	`NodeRestart`	8	`ALERT`
`Node failure of node_idblock completed`	The failure of data node *node_id* has been detected in the *block*`NDB` kernel block, where block is 1 of `DBTC`, `DBDICT`, `DBDIH`, or `DBLQH`; for more information, see NDB Kernel Blocks	`NodeFailCompleted`	`NodeRestart`	8	`ALERT`
`Node mgm_node_id: Node data_node_id has failed. The Node state at failure was state_code`	A data node has failed. Its state at the time of failure is described by an arbitration state code *state_code*: possible state code values can be found in the file `include/kernel/signaldata/ArbitSignalData.hpp`.	`NODE_FAILREP`	`NodeRestart`	8	`ALERT`
`President restarts arbitration thread [state=state_code]` or `Prepare arbitrator`	This is a report on the current state and progress of arbitration in the cluster. *node_id* is the node ID of the management	`ArbitState`	`NodeRestart`	6	`INFO`

Log Message	Description	Event Name	Event Type	Priority	Severity
`node node_id [ticket=ticket_id]` or `Receive arbitrator node node_id [ticket=ticket_id]` or `Started arbitrator node node_id [ticket=ticket_id]` or `Lost arbitrator node node_id - process failure [state=state_code]` or `Lost arbitrator node node_id - process exit [state=state_code]` or `Lost arbitrator node node_id - error_message [state=state_code]`	node or SQL node selected as the arbitrator. *state_code* is an arbitration state code, as found in `include/kernel/signaldata/ArbitSignalData.hpp`. When an error has occurred, an *error_message*, also defined in `ArbitSignalData.hpp`, is provided. *ticket_id* is a unique identifier handed out by the arbitrator when it is selected to all the nodes that participated in its selection; this is used to ensure that each node requesting arbitration was one of the nodes that took part in the selection process.				
`Arbitration check lost - less than 1/2 nodes left` or `Arbitration check won - all node groups and more than 1/2 nodes left` or `Arbitration check won - node group majority` or `Arbitration check lost - missing node group` or `Network partitioning - arbitration required` or `Arbitration won - positive`	This message reports on the result of arbitration. In the event of arbitration failure, an *error_message* and an arbitration *state_code* are provided; definitions for both of these are found in `include/kernel/signaldata/ArbitSignalData.hpp`.	`ArbitResult`	`NodeRestart`	2	`ALERT`

Log Message	Description	Event Name	Event Type	Priority	Severity
`reply from node node_id` or `Arbitration lost - negative reply from node node_id` or `Network partitioning - no arbitrator available` or `Network partitioning - no arbitrator configured` or `Arbitration failure - error_message [state=state_code]`					
`Node node_id: GCP Take over started`	This node is attempting to assume responsibility for the next global checkpoint (that is, it is becoming the master node)	`GCP_TakeoverStarted`	`NodeRestart`	7	`INFO`
`Node node_id: GCP Take over completed`	This node has become the master, and has assumed responsibility for the next global checkpoint	`GCP_TakeoverCompleted`	`NodeRestart`	7	`INFO`
`Node node_id: LCP Take over started`	This node is attempting to assume responsibility for the next set of local checkpoints (that is, it is becoming the master node)	`LCP_TakeoverStarted`	`NodeRestart`	7	`INFO`
`Node node_id: LCP Take over completed`	This node has become the master, and has assumed responsibility for the next set of local checkpoints	`LCP_TakeoverCompleted`	`NodeRestart`	7	`INFO`
`Node node_id: Trans. Count = transactions, Commit Count`	This report of transaction activity is given	`TransReportCounters`	`Statistic`	8	`INFO`

Log Message	Description	Event Name	Event Type	Priority	Severity
= *commits*, Read Count = *reads*, Simple Read Count = *simple_reads*, Write Count = *writes*, AttrInfo Count = *AttrInfo_objects*, Concurrent Operations = *concurrent_operations*, Abort Count = *aborts*, Scans = *scans*, Range scans = *range_scans*	approximately once every 10 seconds				
Node *node_id*: Operations=*operations*	Number of operations performed by this node, provided approximately once every 10 seconds	OperationReportCounters	Statistic	8	INFO
Node *node_id*: Table with ID = *table_id* created	A table having the table ID shown has been created	TableCreated	Statistic	7	INFO
Node *node_id*: Mean loop Counter in doJob last 8192 times = *count*		JobStatistic	Statistic	9	INFO
Mean send size to Node = *node_id* last 4096 sends = *bytes* bytes	This node is sending an average of *bytes* bytes per send to node *node_id*	SendBytesStatistic	Statistic	9	INFO
Mean receive size to Node = *node_id* last 4096 sends = *bytes* bytes	This node is receiving an average of *bytes* of data each time it receives data from node *node_id*	ReceiveBytesStatistics	Statistic	9	INFO
Node *node_id*: Data usage is *data_memory_percentage*% (*data_pages_used* 32K pages of total	This report is generated when a DUMP 1000 command is issued in the cluster management client	MemoryUsage	Statistic	5	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
data_pages_total) / Node *node_id*: Index usage is *index_memory_percentage*% (*index_pages_used* 8K pages of total *index_pages_total*)					
Node *node1_id*: Transporter to node *node2_id* reported error *error_code*: *error_message*	A transporter error occurred while communicating with node *node2_id*; for a listing of transporter error codes and messages, see NDB Transporter Errors, in MySQL NDB Cluster Internals Manual	TransporterError	Error	2	ERROR
Node *node1_id*: Transporter to node *node2_id* reported error *error_code*: *error_message*	A warning of a potential transporter problem while communicating with node *node2_id*; for a listing of transporter error codes and messages, see NDB Transporter Errors, for more information	TransporterWarning	Error	8	WARNING
Node *node1_id*: Node *node2_id* missed heartbeat *heartbeat_id*	This node missed a heartbeat from node *node2_id*	MissedHeartbeat	Error	8	WARNING
Node *node1_id*: Node *node2_id* declared dead due to missed heartbeat	This node has missed at least 3 heartbeats from node *node2_id*, and so has declared that node “dead”	DeadDueToHeartbeat	Error	8	ALERT
Node *node1_id*: Node Sent Heartbeat to node = *node2_id*	This node has sent a heartbeat to node *node2_id*	SentHeartbeat	Info	12	INFO
Node *node_id*: Event buffer status	This report is seen during heavy event buffer usage, for	EventBufferStatus2	Info	7	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
(object_id): used=*bytes_used* (*percent_used*% of alloc) alloc=*bytes_allocated* max=*bytes_available* latest_consumed_epoch=*latest_consumed_epoch* latest_buffered_epoch=*latest_buffered_epoch* report_reason=*report_reason*	example, when many updates are being applied in a relatively short period of time; the report shows the number of bytes and the percentage of event buffer memory used, the bytes allocated and percentage still available, and the latest buffered and consumed epochs; for more information, see Section 22.5.7.3, “Event Buffer Reporting in the Cluster Log”				
Node *node_id*: Entering single user mode, Node *node_id*: Entered single user mode Node *API_node_id* has exclusive access, Node *node_id*: Entering single user mode	These reports are written to the cluster log when entering and exiting single user mode; *API_node_id* is the node ID of the API or SQL having exclusive access to the cluster (for more information, see Section 22.5.8, “NDB Cluster Single User Mode”); the message Unknown single user report *API_node_id* indicates an error has taken place and should never be seen in normal operation	SingleUser	Info	7	INFO
Node *node_id*: Backup *backup_id* started from node *mgm_node_id*	A backup has been started using the management node having *mgm_node_id*; this message is also displayed	BackupStarted	Backup	7	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
	in the cluster management client when the START BACKUP command is issued; for more information, see Section 22.5.3.2, "Using The NDB Cluster Management Client to Create a Backup"				
Node *node_id*: Backup *backup_id* started from node *mgm_node_id* completed. StartGCP: *start_gcp* StopGCP: *stop_gcp* #Records: *records* #LogRecords: *log_records* Data: *data_bytes* bytes Log: *log_bytes* bytes	The backup having the ID *backup_id* has been completed; for more information, see Section 22.5.3.2, "Using The NDB Cluster Management Client to Create a Backup"	BackupCompleted	Backup	7	INFO
Node *node_id*: Backup request from *mgm_node_id* failed to start. Error: *error_code*	The backup failed to start; for error codes, see MGM API Errors	BackupFailedToStart	Backup	7	ALERT
Node *node_id*: Backup *backup_id* started from *mgm_node_id* has been aborted. Error: *error_code*	The backup was terminated after starting, possibly due to user intervention	BackupAborted	Backup	7	ALERT

22.5.7.2 NDB Cluster Log Startup Messages

Possible startup messages with descriptions are provided in the following list:

- Initial start, waiting for %s to connect, nodes [all: %s connected: %s no-wait: %s]

- `Waiting until nodes: %s connects, nodes [ all: %s connected: %s no-wait: %s ]`
- `Waiting %u sec for nodes %s to connect, nodes [ all: %s connected: %s no-wait: %s ]`
- `Waiting for non partitioned start, nodes [ all: %s connected: %s missing: %s no-wait: %s ]`
- `Waiting %u sec for non partitioned start, nodes [ all: %s connected: %s missing: %s no-wait: %s ]`
- `Initial start with nodes %s [ missing: %s no-wait: %s ]`
- `Start with all nodes %s`
- `Start with nodes %s [ missing: %s no-wait: %s ]`
- `Start potentially partitioned with nodes %s [ missing: %s no-wait: %s ]`
- `Unknown startreport: 0x%x [ %s %s %s %s ]`

22.5.7.3 Event Buffer Reporting in the Cluster Log

`NDB` uses one or more memory buffers for events received from the data nodes. There is one such buffer for each `Ndb` object subscribing to table events, which means that there are usually two buffers for each `mysqld` performing binary logging (one buffer for schema events, and one for data events). Each buffer contains epochs made up of events. These events consist of operation types (insert, update, delete) and row data (before and after images plus metadata).

`NDB` generates messages in the cluster log to describe the state of these buffers. Although these reports appear in the cluster log, they refer to buffers on API nodes (unlike most other cluster log messages, which are generated by data nodes).

Event buffer logging reports in the cluster log use the format shown here:

```
Node node_id: Event buffer status (object_id):
used=bytes_used (percent_used% of alloc)
alloc=bytes_allocated (percent_alloc% of max) max=bytes_available
latest_consumed_epoch=latest_consumed_epoch
latest_buffered_epoch=latest_buffered_epoch
report_reason=report_reason
```

The fields making up this report are listed here, with descriptions:

- *`node_id`*: ID of the node where the report originated.
- *`object_id`*: ID of the `Ndb` object where the report originated.
- *`bytes_used`*: Number of bytes used by the buffer.
- *`percent_used`*: Percentage of allocated bytes used.
- *`bytes_allocated`*: Number of bytes allocated to this buffer.
- *`percent_alloc`*: Percentage of available bytes used; not printed if `ndb_eventbuffer_max_alloc` is equal to 0 (unlimited).
- *`bytes_available`*: Number of bytes available; this is 0 if `ndb_eventbuffer_max_alloc` is 0 (unlimited).

- *latest_consumed_epoch*: The epoch most recently consumed to completion. (In NDB API applications, this is done by calling nextEvent().)
- *latest_buffered_epoch*: The epoch most recently buffered (completely) in the event buffer.
- *report_reason*: The reason for making the report. Possible reasons are shown later in this section.

Possible reasons for reporting are described in the following list:

- ENOUGH_FREE_EVENTBUFFER: The event buffer has sufficient space.

  LOW_FREE_EVENTBUFFER: The event buffer is running low on free space.

  The threshold free percentage level triggering these reports can be adjusted by setting the ndb_report_thresh_binlog_mem_usage server variable.

- BUFFERED_EPOCHS_OVER_THRESHOLD: Whether the number of buffered epochs has exceeded the configured threshold. This number is the difference between the latest epoch that has been received in its entirety and the epoch that has most recently been consumed (in NDB API applications, this is done by calling nextEvent() or nextEvent2()). The report is generated every second until the number of buffered epochs goes below the threshold, which can be adjusted by setting the ndb_report_thresh_binlog_epoch_slip server variable. You can also adjust the threshold in NDB API applications by calling setEventBufferQueueEmptyEpoch().
- PARTIALLY_DISCARDING: Event buffer memory is exhausted—that is, 100% of ndb_eventbuffer_max_alloc has been used. Any partially buffered epoch is buffered to completion even is usage exceeds 100%, but any new epochs received are discarded. This means that a gap has occurred in the event stream.
- COMPLETELY_DISCARDING: No epochs are buffered.
- PARTIALLY_BUFFERING: The buffer free percentage following the gap has risen to the threshold, which can be set in the mysql client using the ndb_eventbuffer_free_percent server system variable or in NDB API applications by calling set_eventbuffer_free_percent(). New epochs are buffered. Epochs that could not be completed due to the gap are discarded.
- COMPLETELY_BUFFERING: All epochs received are being buffered, which means that there is sufficient event buffer memory. The gap in the event stream has been closed.

22.5.7.4 NDB Cluster: NDB Transporter Errors

This section lists error codes, names, and messages that are written to the cluster log in the event of transporter errors.

Table 22.365 Error codes generated by transporter errors

Error Code	Error Name	Error Text
0x00	**TE_NO_ERROR**	No error
0x01	**TE_ERROR_CLOSING_SOCKET**	Error found during closing of socket
0x02	**TE_ERROR_IN_SELECT_BEFORE_ACCEPT**	Error found before accept. The transporter will retry
0x03	**TE_INVALID_MESSAGE_LENGTH**	Error found in message (invalid message length)

Error Code	Error Name	Error Text
0x04	TE_INVALID_CHECKSUM	Error found in message (checksum)
0x05	TE_COULD_NOT_CREATE_SOCKET	Error found while creating socket(can't create socket)
0x06	TE_COULD_NOT_BIND_SOCKET	Error found while binding server socket
0x07	TE_LISTEN_FAILED	Error found while listening to server socket
0x08	TE_ACCEPT_RETURN_ERROR	Error found during accept(accept return error)
0x0b	TE_SHM_DISCONNECT	The remote node has disconnected
0x0c	TE_SHM_IPC_STAT	Unable to check shm segment
0x0d	TE_SHM_UNABLE_TO_CREATE_SEGMENT	Unable to create shm segment
0x0e	TE_SHM_UNABLE_TO_ATTACH_SEGMENT	Unable to attach shm segment
0x0f	TE_SHM_UNABLE_TO_REMOVE_SEGMENT	Unable to remove shm segment
0x10	TE_TOO_SMALL_SIGID	Sig ID too small
0x11	TE_TOO_LARGE_SIGID	Sig ID too large
0x12	TE_WAIT_STACK_FULL	Wait stack was full
0x13	TE_RECEIVE_BUFFER_FULL	Receive buffer was full
0x14	TE_SIGNAL_LOST_SEND_BUFFER_FULL	Send buffer was full,and trying to force send fails
0x15	TE_SIGNAL_LOST	Send failed for unknown reason(signal lost)
0x16	TE_SEND_BUFFER_FULL	The send buffer was full, but sleeping for a while solved
0x21	TE_SHM_IPC_PERMANENT	Shm ipc Permanent error

Note

Transporter error codes 0x17 through 0x20 and 0x22 are reserved for SCI connections, which are not supported in this version of NDB Cluster, and so are not included here.

22.5.8 NDB Cluster Single User Mode

Single user mode enables the database administrator to restrict access to the database system to a single API node, such as a MySQL server (SQL node) or an instance of `ndb_restore`. When entering single user mode, connections to all other API nodes are closed gracefully and all running transactions are aborted. No new transactions are permitted to start.

Once the cluster has entered single user mode, only the designated API node is granted access to the database.

You can use the `ALL STATUS` command in the `ndb_mgm` client to see when the cluster has entered single user mode. You can also check the `status` column of the `ndbinfo.nodes` table (see Section 22.5.10.28, “The ndbinfo nodes Table”, for more information).

Example:

```
ndb_mgm> ENTER SINGLE USER MODE 5
```

After this command has executed and the cluster has entered single user mode, the API node whose node ID is `5` becomes the cluster's only permitted user.

The node specified in the preceding command must be an API node; attempting to specify any other type of node will be rejected.

Note

When the preceding command is invoked, all transactions running on the designated node are aborted, the connection is closed, and the server must be restarted.

The command `EXIT SINGLE USER MODE` changes the state of the cluster's data nodes from single user mode to normal mode. API nodes—such as MySQL Servers—waiting for a connection (that is, waiting for the cluster to become ready and available), are again permitted to connect. The API node denoted as the single-user node continues to run (if still connected) during and after the state change.

Example:

```
ndb_mgm> EXIT SINGLE USER MODE
```

There are two recommended ways to handle a node failure when running in single user mode:

- Method 1:

  1. Finish all single user mode transactions
  2. Issue the `EXIT SINGLE USER MODE` command
  3. Restart the cluster's data nodes

- Method 2:

  Restart storage nodes prior to entering single user mode.

22.5.9 Quick Reference: NDB Cluster SQL Statements

This section discusses several SQL statements that can prove useful in managing and monitoring a MySQL server that is connected to an NDB Cluster, and in some cases provide information about the cluster itself.

- `SHOW ENGINE NDB STATUS`, `SHOW ENGINE NDBCLUSTER STATUS`

  The output of this statement contains information about the server's connection to the cluster, creation and usage of NDB Cluster objects, and binary logging for NDB Cluster replication.

  See Section 13.7.7.15, “SHOW ENGINE Syntax”, for a usage example and more detailed information.

- `SHOW ENGINES`

  This statement can be used to determine whether or not clustering support is enabled in the MySQL server, and if so, whether it is active.

  See Section 13.7.7.16, “SHOW ENGINES Syntax”, for more detailed information.

  Note

  This statement does not support a `LIKE` clause. However, you can use `LIKE` to filter queries against the `INFORMATION_SCHEMA.ENGINES` table, as discussed in the next item.

- `SELECT * FROM INFORMATION_SCHEMA.ENGINES [WHERE ENGINE LIKE 'NDB%']`

  This is the equivalent of `SHOW ENGINES`, but uses the `ENGINES` table of the `INFORMATION_SCHEMA` database. Unlike the case with the `SHOW ENGINES` statement, it is possible to filter the results using a `LIKE` clause, and to select specific columns to obtain information that may be of use in scripts. For example, the following query shows whether the server was built with `NDB` support and, if so, whether it is enabled:

```
mysql> SELECT SUPPORT FROM INFORMATION_SCHEMA.ENGINES
    ->   WHERE ENGINE LIKE 'NDB%';
+---------+
| support |
+---------+
| ENABLED |
+---------+
```

  See Section 25.9, “The INFORMATION_SCHEMA ENGINES Table”, for more information.

- `SHOW VARIABLES LIKE 'NDB%'`

  This statement provides a list of most server system variables relating to the `NDB` storage engine, and their values, as shown here:

```
mysql> SHOW VARIABLES LIKE 'NDB%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| ndb_autoincrement_prefetch_sz     | 32    |
| ndb_cache_check_time              | 0     |
| ndb_extra_logging                 | 0     |
| ndb_force_send                    | ON    |
| ndb_index_stat_cache_entries      | 32    |
| ndb_index_stat_enable             | OFF   |
| ndb_index_stat_update_freq        | 20    |
| ndb_report_thresh_binlog_epoch_slip | 3     |
```

```
| ndb_report_thresh_binlog_mem_usage  | 10    |
| ndb_use_copying_alter_table         | OFF   |
| ndb_use_exact_count                 | ON    |
| ndb_use_transactions                | ON    |
+-------------------------------------+-------+
```

See Section 5.1.8, “Server System Variables”, for more information.

- `SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES WHERE VARIABLE_NAME LIKE 'NDB%';`

  This statement is the equivalent of the `SHOW` command described in the previous item, and provides almost identical output, as shown here:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES
    ->   WHERE VARIABLE_NAME LIKE 'NDB%';
+-------------------------------------+----------------+
| VARIABLE_NAME                       | VARIABLE_VALUE |
+-------------------------------------+----------------+
| NDB_AUTOINCREMENT_PREFETCH_SZ       | 32             |
| NDB_CACHE_CHECK_TIME                | 0              |
| NDB_EXTRA_LOGGING                   | 0              |
| NDB_FORCE_SEND                      | ON             |
| NDB_INDEX_STAT_CACHE_ENTRIES        | 32             |
| NDB_INDEX_STAT_ENABLE               | OFF            |
| NDB_INDEX_STAT_UPDATE_FREQ          | 20             |
| NDB_REPORT_THRESH_BINLOG_EPOCH_SLIP | 3              |
| NDB_REPORT_THRESH_BINLOG_MEM_USAGE  | 10             |
| NDB_USE_COPYING_ALTER_TABLE         | OFF            |
| NDB_USE_EXACT_COUNT                 | ON             |
| NDB_USE_TRANSACTIONS                | ON             |
+-------------------------------------+----------------+
```

  Unlike the case with the `SHOW` command, it is possible to select individual columns. For example:

```
mysql> SELECT VARIABLE_VALUE
    ->   FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES
    ->   WHERE VARIABLE_NAME = 'ndb_force_send';
+----------------+
| VARIABLE_VALUE |
+----------------+
| ON             |
+----------------+
```

  See The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables, and Section 5.1.8, “Server System Variables”, for more information.

- `SHOW STATUS LIKE 'NDB%'`

  This statement shows at a glance whether or not the MySQL server is acting as a cluster SQL node, and if so, it provides the MySQL server's cluster node ID, the host name and port for the cluster management server to which it is connected, and the number of data nodes in the cluster, as shown here:

```
mysql> SHOW STATUS LIKE 'NDB%';
+--------------------------+----------------+
| Variable_name            | Value          |
+--------------------------+----------------+
| Ndb_cluster_node_id      | 10             |
| Ndb_config_from_host     | 198.51.100.103 |
| Ndb_config_from_port     | 1186           |
| Ndb_number_of_data_nodes | 4              |
```

```
+-------------------------+---------------+
```

If the MySQL server was built with clustering support, but it is not connected to a cluster, all rows in the output of this statement contain a zero or an empty string:

```
mysql> SHOW STATUS LIKE 'NDB%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Ndb_cluster_node_id      | 0     |
| Ndb_config_from_host     |       |
| Ndb_config_from_port     | 0     |
| Ndb_number_of_data_nodes | 0     |
+--------------------------+-------+
```

See also Section 13.7.7.35, “SHOW STATUS Syntax”.

- `SELECT * FROM INFORMATION_SCHEMA.GLOBAL_STATUS WHERE VARIABLE_NAME LIKE 'NDB%';`

  This statement provides similar output to the `SHOW` command discussed in the previous item. However, unlike the case with `SHOW STATUS`, it is possible using the `SELECT` to extract values in SQL for use in scripts for monitoring and automation purposes.

  See The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables, for more information.

You can also query the tables in the `ndbinfo` information database for real-time data about many NDB Cluster operations. See Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”.

22.5.10 ndbinfo: The NDB Cluster Information Database

`ndbinfo` is a database containing information specific to NDB Cluster.

This database contains a number of tables, each providing a different sort of data about NDB Cluster node status, resource usage, and operations. You can find more detailed information about each of these tables in the next several sections.

`ndbinfo` is included with NDB Cluster support in the MySQL Server; no special compilation or configuration steps are required; the tables are created by the MySQL Server when it connects to the cluster. You can verify that `ndbinfo` support is active in a given MySQL Server instance using `SHOW PLUGINS`; if `ndbinfo` support is enabled, you should see a row containing `ndbinfo` in the `Name` column and `ACTIVE` in the `Status` column, as shown here (emphasized text):

```
mysql> SHOW PLUGINS;
+---------------------------------+--------+--------------------+---------+---------+
| Name                            | Status | Type               | Library | License |
+---------------------------------+--------+--------------------+---------+---------+
| binlog                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| mysql_native_password           | ACTIVE | AUTHENTICATION     | NULL    | GPL     |
| sha256_password                 | ACTIVE | AUTHENTICATION     | NULL    | GPL     |
| MRG_MYISAM                      | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MEMORY                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| CSV                             | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MyISAM                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| InnoDB                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| INNODB_TRX                      | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCKS                    | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCK_WAITS               | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP                      | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
```

```
| INNODB_CMP_RESET                | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM                   | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM_RESET             | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_PER_INDEX            | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_PER_INDEX_RESET      | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_BUFFER_PAGE              | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_BUFFER_PAGE_LRU          | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_BUFFER_POOL_STATS        | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_TEMP_TABLE_INFO          | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_METRICS                  | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_DEFAULT_STOPWORD      | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_DELETED               | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_BEING_DELETED         | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_CONFIG                | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_INDEX_CACHE           | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_FT_INDEX_TABLE           | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_TABLES               | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_TABLESTATS           | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_INDEXES              | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_COLUMNS              | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_FIELDS               | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_FOREIGN              | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_FOREIGN_COLS         | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_TABLESPACES          | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_DATAFILES            | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_SYS_VIRTUAL              | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| PERFORMANCE_SCHEMA              | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndbCluster                      | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndbinfo                         | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndb_transid_mysql_connection_map | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| BLACKHOLE                       | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ARCHIVE                         | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| partition                       | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ngram                           | ACTIVE | FTPARSER           | NULL    | GPL     |
+---------------------------------+--------+--------------------+---------+---------+
46 rows in set (0.00 sec)
```

You can also do this by checking the output of `SHOW ENGINES` for a line including `ndbinfo` in the `Engine` column and `YES` in the `Support` column, as shown here (emphasized text):

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: ndbcluster
     Support: YES
     Comment: Clustered, fault-tolerant tables
Transactions: YES
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 4. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
```

```
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 7. row ***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row ***************************
      Engine: ndbinfo
     Support: YES
     Comment: NDB Cluster system information storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 9. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 10. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
10 rows in set (0.00 sec)
```

If `ndbinfo` support is enabled, then you can access `ndbinfo` using SQL statements in `mysql` or another MySQL client. For example, you can see `ndbinfo` listed in the output of `SHOW DATABASES`, as shown here (emphasized text):

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| ndbinfo            |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.04 sec)
```

If the `mysqld` process was not started with the `--ndbcluster` option, `ndbinfo` is not available and is not displayed by `SHOW DATABASES`. If `mysqld` was formerly connected to an NDB Cluster but the

cluster becomes unavailable (due to events such as cluster shutdown, loss of network connectivity, and so forth), `ndbinfo` and its tables remain visible, but an attempt to access any tables (other than `blocks` or `config_params`) fails with `Got error 157 'Connection to NDB failed' from NDBINFO`.

With the exception of the `blocks` and `config_params` tables, what we refer to as `ndbinfo` "tables" are actually views generated from internal `NDB` tables not normally visible to the MySQL Server.

All `ndbinfo` tables are read-only, and are generated on demand when queried. Because many of them are generated in parallel by the data nodes while other are specific to a given SQL node, they are not guaranteed to provide a consistent snapshot.

In addition, pushing down of joins is not supported on `ndbinfo` tables; so joining large `ndbinfo` tables can require transfer of a large amount of data to the requesting API node, even when the query makes use of a `WHERE` clause.

`ndbinfo` tables are not included in the query cache. (Bug #59831)

You can select the `ndbinfo` database with a `USE` statement, and then issue a `SHOW TABLES` statement to obtain a list of tables, just as for any other database, like this:

```
mysql> USE ndbinfo;
Database changed

mysql> SHOW TABLES;
+---------------------------------+
| Tables_in_ndbinfo               |
+---------------------------------+
| arbitrator_validity_detail      |
| arbitrator_validity_summary     |
| blocks                          |
| cluster_locks                   |
| cluster_operations              |
| cluster_transactions            |
| config_nodes                    |
| config_params                   |
| config_values                   |
| counters                        |
| cpustat                         |
| cpustat_1sec                    |
| cpustat_20sec                   |
| cpustat_50ms                    |
| dict_obj_info                   |
| dict_obj_types                  |
| disk_write_speed_aggregate      |
| disk_write_speed_aggregate_node |
| disk_write_speed_base           |
| diskpagebuffer                  |
| error_messages                  |
| locks_per_fragment              |
| logbuffers                      |
| logspaces                       |
| membership                      |
| memory_per_fragment             |
| memoryusage                     |
| nodes                           |
| operations_per_fragment         |
| processes                       |
| resources                       |
| restart_info                    |
| server_locks                    |
| server_operations               |
| server_transactions             |
| table_distribution_status       |
```

```
| table_fragments                 |
| table_info                      |
| table_replicas                  |
| tc_time_track_stats             |
| threadblocks                    |
| threads                         |
| threadstat                      |
| transporters                    |
+---------------------------------+
44 rows in set (0.00 sec)
```

In NDB 8.0, all `ndbinfo` tables use the `NDB` storage engine; however, an `ndbinfo` entry still appears in the output of `SHOW ENGINES` and `SHOW PLUGINS` as described previously.

You can execute `SELECT` statements against these tables, just as you would normally expect:

```
mysql> SELECT * FROM memoryusage;
+---------+--------------------+--------+------------+------------+-------------+
| node_id | memory_type        | used   | used_pages | total      | total_pages |
+---------+--------------------+--------+------------+------------+-------------+
|       5 | Data memory        | 753664 |         23 | 1073741824 |       32768 |
|       5 | Index memory       | 163840 |         20 | 1074003968 |      131104 |
|       5 | Long message buffer |   2304 |          9 |   67108864 |      262144 |
|       6 | Data memory        | 753664 |         23 | 1073741824 |       32768 |
|       6 | Index memory       | 163840 |         20 | 1074003968 |      131104 |
|       6 | Long message buffer |   2304 |          9 |   67108864 |      262144 |
+---------+--------------------+--------+------------+------------+-------------+
6 rows in set (0.02 sec)
```

More complex queries, such as the two following `SELECT` statements using the `memoryusage` table, are possible:

```
mysql> SELECT SUM(used) as 'Data Memory Used, All Nodes'
     >     FROM memoryusage
     >     WHERE memory_type = 'Data memory';
+-----------------------------+
| Data Memory Used, All Nodes |
+-----------------------------+
|                        6460 |
+-----------------------------+
1 row in set (0.37 sec)

mysql> SELECT SUM(max) as 'Total IndexMemory Available'
     >     FROM memoryusage
     >     WHERE memory_type = 'Index memory';
+-----------------------------+
| Total IndexMemory Available |
+-----------------------------+
|                       25664 |
+-----------------------------+
1 row in set (0.33 sec)
```

`ndbinfo` table and column names are case sensitive (as is the name of the `ndbinfo` database itself). These identifiers are in lowercase. Trying to use the wrong lettercase results in an error, as shown in this example:

```
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+
| node_id | uptime | status  | start_phase |
+---------+--------+---------+-------------+
|       1 |  13602 | STARTED |           0 |
|       2 |     16 | STARTED |           0 |
```

```
+---------+--------+---------+-------------+
2 rows in set (0.04 sec)

mysql> SELECT * FROM Nodes;
ERROR 1146 (42S02): Table 'ndbinfo.Nodes' doesn't exist
```

`mysqldump` ignores the `ndbinfo` database entirely, and excludes it from any output. This is true even when using the `--databases` or `--all-databases` option.

NDB Cluster also maintains tables in the `INFORMATION_SCHEMA` information database, including the `FILES` table which contains information about files used for NDB Cluster Disk Data storage, and the `ndb_transid_mysql_connection_map` table, which shows the relationships between transactions, transaction coordinators, and NDB Cluster API nodes. For more information, see the descriptions of the tables or Section 22.5.11, “INFORMATION_SCHEMA Tables for NDB Cluster”.

22.5.10.1 The ndbinfo arbitrator_validity_detail Table

The `arbitrator_validity_detail` table shows the view that each data node in the cluster has of the arbitrator. It is a subset of the `membership` table.

The following table provides information about the columns in the `arbitrator_validity_detail` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.366 Columns of the arbitrator_validity_detail table

Column Name	Type	Description
`node_id`	integer	This node's node ID
`arbitrator`	integer	Node ID of arbitrator
`arb_ticket`	string	Internal identifier used to track arbitration
`arb_connected`	`Yes` or `No`	Whether this node is connected to the arbitrator
`arb_state`	Enumeration (see text)	Arbitration state

The node ID is the same as that reported by `ndb_mgm -e "SHOW"`.

All nodes should show the same `arbitrator` and `arb_ticket` values as well as the same `arb_state` value. Possible `arb_state` values are `ARBIT_NULL`, `ARBIT_INIT`, `ARBIT_FIND`, `ARBIT_PREP1`, `ARBIT_PREP2`, `ARBIT_START`, `ARBIT_RUN`, `ARBIT_CHOOSE`, `ARBIT_CRASH`, and `UNKNOWN`.

`arb_connected` shows whether the current node is connected to the `arbitrator`.

22.5.10.2 The ndbinfo arbitrator_validity_summary Table

The `arbitrator_validity_summary` table provides a composite view of the arbitrator with regard to the cluster's data nodes.

The following table provides information about the columns in the `arbitrator_validity_summary` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.367 Columns of the arbitrator_validity_summary table

Column Name	Type	Description
`arbitrator`	integer	Node ID of arbitrator

Column Name	Type	Description
arb_ticket	string	Internal identifier used to track arbitration
arb_connected	Yes or No	Whether this arbitrator is connected to the cluster
consensus_count	integer	Number of data nodes that see this node as arbitrator

In normal operations, this table should have only 1 row for any appreciable length of time. If it has more than 1 row for longer than a few moments, then either not all nodes are connected to the arbitrator, or all nodes are connected, but do not agree on the same arbitrator.

The arbitrator column shows the arbitrator's node ID.

arb_ticket is the internal identifier used by this arbitrator.

arb_connected shows whether this node is connected to the cluster as an arbitrator.

22.5.10.3 The ndbinfo blocks Table

The blocks table is a static table which simply contains the names and internal IDs of all NDB kernel blocks (see NDB Kernel Blocks). It is for use by the other ndbinfo tables (most of which are actually views) in mapping block numbers to block names for producing human-readable output.

The following table provides information about the columns in the blocks table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.368 Columns of the blocks table

Column Name	Type	Description
block_number	integer	Block number
block_name	string	Block name

To obtain a list of all block names, simply execute SELECT block_name FROM ndbinfo.blocks. Although this is a static table, its content can vary between different NDB Cluster releases.

22.5.10.4 The ndbinfo cluster_locks Table

The cluster_locks table provides information about current lock requests holding and waiting for locks on NDB tables in an NDB Cluster, and is intended as a companion table to cluster_operations. Information obtain from the cluster_locks table may be useful in investigating stalls and deadlocks.

The following table provides information about the columns in the cluster_locks table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.369 Columns of the cluster_locks table

Column Name	Type	Description
node_id	integer	ID of reporting node
block_instance	integer	ID of reporting LDM instance
tableid	integer	ID of table containing this row

Column Name	Type	Description
fragmentid	integer	ID of fragment containing locked row
rowid	integer	ID of locked row
transid	integer	Transaction ID
mode	string	Lock request mode
state	string	Lock state
detail	string	Whether this is first holding lock in row lock queue
op	string	Operation type
duration_millis	integer	Milliseconds spent waiting or holding lock
lock_num	integer	ID of lock object
waiting_for	integer	Waiting for lock with this ID

The table ID (tableid column) is assigned internally, and is the same as that used in other ndbinfo tables. It is also shown in the output of ndb_show_tables.

The transaction ID (transid column) is the identifier generated by the NDB API for the transaction requesting or holding the current lock.

The mode column shows the lock mode; this is always one of S (indicating a shared lock) or X (an exclusive lock). If a transaction holds an exclusive lock on a given row, all other locks on that row have the same transaction ID.

The state column shows the lock state. Its value is always one of H (holding) or W (waiting). A waiting lock request waits for a lock held by a different transaction.

When the detail column contains a * (asterisk character), this means that this lock is the first holding lock in the affected row's lock queue; otherwise, this column is empty. This information can be used to help identify the unique entries in a list of lock requests.

The op column shows the type of operation requesting the lock. This is always one of the values READ, INSERT, UPDATE, DELETE, SCAN, or REFRESH.

The duration_millis column shows the number of milliseconds for which this lock request has been waiting or holding the lock. This is reset to 0 when a lock is granted for a waiting request.

The lock ID (lockid column) is unique to this node and block instance.

The lock state is shown in the lock_state column; if this is W, the lock is waiting to be granted, and the waiting_for column shows the lock ID of the lock object this request is waiting for. Otherwise, the waiting_for column is empty. waiting_for can refer only to locks on the same row, as identified by node_id, block_instance, tableid, fragmentid, and rowid.

22.5.10.5 The ndbinfo cluster_operations Table

The cluster_operations table provides a per-operation (stateful primary key op) view of all activity in the NDB Cluster from the point of view of the local data management (LQH) blocks (see The DBLQH Block).

The following table provides information about the columns in the cluster_operations table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.370 Columns of the cluster_operations table

Column Name	Type	Description
node_id	integer	Node ID of reporting LQH block
block_instance	integer	LQH block instance
transid	integer	Transaction ID
operation_type	string	Operation type (see text for possible values)
state	string	Operation state (see text for possible values)
tableid	integer	Table ID
fragmentid	integer	Fragment ID
client_node_id	integer	Client node ID
client_block_ref	integer	Client block reference
tc_node_id	integer	Transaction coordinator node ID
tc_block_no	integer	Transaction coordinator block number
tc_block_instance	integer	Transaction coordinator block instance

The transaction ID is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The `operation_type` column can take any one of the values `READ`, `READ-SH`, `READ-EX`, `INSERT`, `UPDATE`, `DELETE`, `WRITE`, `UNLOCK`, `REFRESH`, `SCAN`, `SCAN-SH`, `SCAN-EX`, or `<unknown>`.

The `state` column can have any one of the values `ABORT_QUEUED`, `ABORT_STOPPED`, `COMMITTED`, `COMMIT_QUEUED`, `COMMIT_STOPPED`, `COPY_CLOSE_STOPPED`, `COPY_FIRST_STOPPED`, `COPY_STOPPED`, `COPY_TUPKEY`, `IDLE`, `LOG_ABORT_QUEUED`, `LOG_COMMIT_QUEUED`, `LOG_COMMIT_QUEUED_WAIT_SIGNAL`, `LOG_COMMIT_WRITTEN`, `LOG_COMMIT_WRITTEN_WAIT_SIGNAL`, `LOG_QUEUED`, `PREPARED`, `PREPARED_RECEIVED_COMMIT`, `SCAN_CHECK_STOPPED`, `SCAN_CLOSE_STOPPED`, `SCAN_FIRST_STOPPED`, `SCAN_RELEASE_STOPPED`, `SCAN_STATE_USED`, `SCAN_STOPPED`, `SCAN_TUPKEY`, `STOPPED`, `TC_NOT_CONNECTED`, `WAIT_ACC`, `WAIT_ACC_ABORT`, `WAIT_AI_AFTER_ABORT`, `WAIT_ATTR`, `WAIT_SCAN_AI`, `WAIT_TUP`, `WAIT_TUPKEYINFO`, `WAIT_TUP_COMMIT`, or `WAIT_TUP_TO_ABORT`. (If the MySQL Server is running with `ndbinfo_show_hidden` enabled, you can view this list of states by selecting from the `ndb$dblqh_tcconnect_state` table, which is normally hidden.)

You can obtain the name of an `NDB` table from its table ID by checking the output of `ndb_show_tables`.

The `fragid` is the same as the partition number seen in the output of `ndb_desc --extra-partition-info` (short form `-p`).

In `client_node_id` and `client_block_ref`, `client` refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The `block_instance` and `tc_block_instance` column provide, respectively, the `DBLQH` and `DBTC` block instance numbers. You can use these along with the block names to obtain information about specific threads from the `threadblocks` table.

22.5.10.6 The ndbinfo cluster_transactions Table

The `cluster_transactions` table shows information about all ongoing transactions in an NDB Cluster.

The following table provides information about the columns in the `cluster_transactions` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.371 Columns of the cluster_transactions table

Column Name	Type	Description
`node_id`	integer	Node ID of transaction coordinator
`block_instance`	integer	TC block instance
`transid`	integer	Transaction ID
`state`	string	Operation state (see text for possible values)
`count_operations`	integer	Number of stateful primary key operations in transaction (includes reads with locks, as well as DML operations)
`outstanding_operations`	integer	Operations still being executed in local data management blocks
`inactive_seconds`	integer	Time spent waiting for API
`client_node_id`	integer	Client node ID
`client_block_ref`	integer	Client block reference

The transaction ID is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

`block_instance` refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the `threadblocks` table.

The `state` column can have any one of the values `CS_ABORTING`, `CS_COMMITTING`, `CS_COMMIT_SENT`, `CS_COMPLETE_SENT`, `CS_COMPLETING`, `CS_CONNECTED`, `CS_DISCONNECTED`, `CS_FAIL_ABORTED`, `CS_FAIL_ABORTING`, `CS_FAIL_COMMITTED`, `CS_FAIL_COMMITTING`, `CS_FAIL_COMPLETED`, `CS_FAIL_PREPARED`, `CS_PREPARE_TO_COMMIT`, `CS_RECEIVING`, `CS_REC_COMMITTING`, `CS_RESTART`, `CS_SEND_FIRE_TRIG_REQ`, `CS_STARTED`, `CS_START_COMMITTING`, `CS_START_SCAN`, `CS_WAIT_ABORT_CONF`, `CS_WAIT_COMMIT_CONF`, `CS_WAIT_COMPLETE_CONF`, `CS_WAIT_FIRE_TRIG_REQ`. (If the MySQL Server is running with `ndbinfo_show_hidden` enabled, you can view this list of states by selecting from the `ndb$dbtc_apiconnect_state` table, which is normally hidden.)

In `client_node_id` and `client_block_ref`, `client` refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The `tc_block_instance` column provides the `DBTC` block instance number. You can use this along with the block name to obtain information about specific threads from the `threadblocks` table.

22.5.10.7 The ndbinfo config_nodes Table

The `config_nodes` table shows nodes configured in an NDB Cluster `config.ini` file. For each node, the table displays a row containing the node ID, the type of node (management node, data node, or API node), and the name or IP address of the host on which the node is configured to run.

This table does not indicate whether a given node is actually running, or whether it is currently connected to the cluster. Information about nodes connected to an NDB Cluster can be obtained from the `nodes` and `processes` table.

The following table provides information about the columns in the config_nodes table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.372 Columns of the config_params table

Column Name	Type	Description
node_id	integer	The node's ID
node_type	string	The type of node
node_hostname	string	The name or IP address of the host on which the node resides

The node_id column shows the node ID used in the config.ini file for this node; if none is specified, the node ID that would be assigned automatically to this node is displayed.

The node_type column displays one of the following three values:

- MGM: Management node.
- NDB: Data node.
- API: API node; this includes SQL nodes.

The node_hostname column shows the node host as specified in the config.ini file. This can be empty for an API node, if HostName has not been set in the cluster configuration file. If HostName has not been set for a data node in the configuration file, localhost is used here. localhost is also used if HostName has not been specified for a management node.

22.5.10.8 The ndbinfo config_params Table

The config_params table is a static table which provides the names and internal ID numbers of and other information about NDB Cluster configuration parameters.

The following table provides information about the columns in the config_params table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table. This table can also be used in conjunction with the config_values table for obtaining realtime information about node configuration parameters.

Table 22.373 Columns of the config_params table

Column Name	Type	Description
param_number	integer	The parameter's internal ID number
param_name	string	The name of the parameter
param_description	string	A brief description of the parameter
param_type	string	The parameter's data type
param_default	string	The parameter's default value, if any
param_min	string	The parameter's maximum value, if any
param_max	string	The parameter's minimum value, if any
param_mandatory	integer	This is 1 if the parameter is required, otherwise 0

Column Name	Type	Description
param_status	string	Currently unused

This table is read-only.

Although this is a static table, its content can vary between NDB Cluster installations, since supported parameters can vary due to differences between software releases, cluster hardware configurations, and other factors.

22.5.10.9 The ndbinfo config_values Table

The config_values table provides information about the current state of node configuration parameter values. Each row in the table corresponds to the current value of a parameter on a given node.

Table 22.374 Columns of the config_params table

Column Name	Type	Description
node_id	integer	ID of the node in the cluster
config_param	integer	The parameter's internal ID number
config_value	string	Current value of the parameter

This table's config_param column and the config_params table's param_number column use the same parameter identifiers. By joining the two tables on these columns, you can obtain detailed information about desired node configuration parameters. The query shown here provides the current values for all parameters on each data node in the cluster, ordered by node ID and parameter name:

```
SELECT    v.node_id AS 'Node Id',
          p.param_name AS 'Parameter',
          v.config_value AS 'Value'
FROM      config_values v
JOIN      config_params p
ON        v.config_param=p.param_number
WHERE     p.param_name NOT LIKE '\_\_%'
ORDER BY  v.node_id, p.param_name;
```

Partial output from the previous query when run on a small example cluster used for simple testing:

```
+---------+-----------------------------------------+----------------+
| Node Id | Parameter                               | Value          |
+---------+-----------------------------------------+----------------+
|       2 | Arbitration                             | 1              |
|       2 | ArbitrationTimeout                      | 7500           |
|       2 | BackupDataBufferSize                    | 16777216       |
|       2 | BackupDataDir                           | /home/jon/data |
|       2 | BackupDiskWriteSpeedPct                 | 50             |
|       2 | BackupLogBufferSize                     | 16777216       |

...

|       3 | TotalSendBufferMemory                   | 0              |
|       3 | TransactionBufferMemory                 | 1048576        |
|       3 | TransactionDeadlockDetectionTimeout     | 1200           |
|       3 | TransactionInactiveTimeout              | 4294967039     |
|       3 | TwoPassInitialNodeRestartCopy           | 0              |
|       3 | UndoDataBuffer                          | 16777216       |
|       3 | UndoIndexBuffer                         | 2097152        |
+---------+-----------------------------------------+----------------+
248 rows in set (0.02 sec)
```

The `WHERE` clause filters out parameters whose names begin with a double underscore (`__`); these parameters are reserved for testing and other internal uses by the NDB developers, and are not intended for use in a production NDB Cluster.

You can obtain output that is more specific, more detailed, or both by issuing the proper queries. This example provides all types of available information about the `NodeId`, `NoOfReplicas`, `HostName`, `DataMemory`, `IndexMemory`, and `TotalSendBufferMemory` parameters as currently set for all data nodes in the cluster:

```
SELECT  p.param_name AS Name,
        v.node_id AS Node,
        p.param_type AS Type,
        p.param_default AS 'Default',
        p.param_min AS Minimum,
        p.param_max AS Maximum,
        CASE p.param_mandatory WHEN 1 THEN 'Y' ELSE 'N' END AS 'Required',
        v.config_value AS Current
FROM    config_params p
JOIN    config_values v
ON      p.param_number = v.config_param
WHERE   p. param_name
  IN ('NodeId', 'NoOfReplicas', 'HostName',
      'DataMemory', 'IndexMemory', 'TotalSendBufferMemory')\G
```

The output from this query when run on a small NDB Cluster with 2 data nodes used for simple testing is shown here (NDB 8.0.18 and later):

```
*************************** 1. row ***************************
    Name: NodeId
    Node: 2
    Type: unsigned
 Default:
 Minimum: 1
 Maximum: 144
Required: Y
 Current: 2
*************************** 2. row ***************************
    Name: HostName
    Node: 2
    Type: string
 Default: localhost
 Minimum:
 Maximum:
Required: N
 Current: 127.0.0.1
*************************** 3. row ***************************
    Name: TotalSendBufferMemory
    Node: 2
    Type: unsigned
 Default: 0
 Minimum: 262144
 Maximum: 4294967039
Required: N
 Current: 0
*************************** 4. row ***************************
    Name: NoOfReplicas
    Node: 2
    Type: unsigned
 Default: 2
 Minimum: 1
 Maximum: 4
Required: N
 Current: 2
```

```
*************************** 5. row ***************************
    Name: DataMemory
    Node: 2
    Type: unsigned
 Default: 102760448
 Minimum: 1048576
 Maximum: 1099511627776
Required: N
 Current: 524288000
*************************** 6. row ***************************
    Name: NodeId
    Node: 3
    Type: unsigned
 Default:
 Minimum: 1
 Maximum: 144
Required: Y
 Current: 3
*************************** 7. row ***************************
    Name: HostName
    Node: 3
    Type: string
 Default: localhost
 Minimum:
 Maximum:
Required: N
 Current: 127.0.0.1
*************************** 8. row ***************************
    Name: TotalSendBufferMemory
    Node: 3
    Type: unsigned
 Default: 0
 Minimum: 262144
 Maximum: 4294967039
Required: N
 Current: 0
*************************** 9. row ***************************
    Name: NoOfReplicas
    Node: 3
    Type: unsigned
 Default: 2
 Minimum: 1
 Maximum: 4
Required: N
 Current: 2
*************************** 10. row ***************************
    Name: DataMemory
    Node: 3
    Type: unsigned
 Default: 102760448
 Minimum: 1048576
 Maximum: 1099511627776
Required: N
 Current: 524288000
10 rows in set (0.01 sec)
```

22.5.10.10 The ndbinfo counters Table

The `counters` table provides running totals of events such as reads and writes for specific kernel blocks and data nodes. Counts are kept from the most recent node start or restart; a node start or restart resets all counters on that node. Not all kernel blocks have all types of counters.

The following table provides information about the columns in the `counters` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.375 Columns of the counters table

Column Name	Type	Description
node_id	integer	The data node ID
block_name	string	Name of the associated NDB kernel block (see NDB Kernel Blocks).
block_instance	integer	Block instance
counter_id	integer	The counter's internal ID number; normally an integer between 1 and 10, inclusive.
counter_name	string	The name of the counter. See text for names of individual counters and the NDB kernel block with which each counter is associated.
val	integer	The counter's value

Each counter is associated with a particular NDB kernel block.

The OPERATIONS counter is associated with the DBLQH (local query handler) kernel block (see The DBLQH Block). A primary-key read counts as one operation, as does a primary-key update. For reads, there is one operation in DBLQH per operation in DBTC. For writes, there is one operation counted per replica.

The ATTRINFO, TRANSACTIONS, COMMITS, READS, LOCAL_READS, SIMPLE_READS, WRITES, LOCAL_WRITES, ABORTS, TABLE_SCANS, and RANGE_SCANS counters are associated with the DBTC (transaction co-ordinator) kernel block (see The DBTC Block).

LOCAL_WRITES and LOCAL_READS are primary-key operations using a transaction coordinator in a node that also holds the primary replica of the record.

The READS counter includes all reads. LOCAL_READS includes only those reads of the primary replica on the same node as this transaction coordinator. SIMPLE_READS includes only those reads in which the read operation is the beginning and ending operation for a given transaction. Simple reads do not hold locks but are part of a transaction, in that they observe uncommitted changes made by the transaction containing them but not of any other uncommitted transactions. Such reads are “simple” from the point of view of the TC block; since they hold no locks they are not durable, and once DBTC has routed them to the relevant LQH block, it holds no state for them.

ATTRINFO keeps a count of the number of times an interpreted program is sent to the data node. See NDB Protocol Messages, for more information about ATTRINFO messages in the NDB kernel.

The LOCAL_TABLE_SCANS_SENT, READS_RECEIVED, PRUNED_RANGE_SCANS_RECEIVED, RANGE_SCANS_RECEIVED, LOCAL_READS_SENT, CONST_PRUNED_RANGE_SCANS_RECEIVED, LOCAL_RANGE_SCANS_SENT, REMOTE_READS_SENT, REMOTE_RANGE_SCANS_SENT, READS_NOT_FOUND, SCAN_BATCHES_RETURNED, TABLE_SCANS_RECEIVED, and SCAN_ROWS_RETURNED counters are associated with the DBSPJ (select push-down join) kernel block (see The DBSPJ Block).

The block_name and block_instance columns provide, respectively, the applicable NDB kernel block name and instance number. You can use these to obtain information about specific threads from the threadblocks table.

A number of counters provide information about transporter overload and send buffer sizing when troubleshooting such issues. For each LQH instance, there is one instance of each counter in the following list:

- LQHKEY_OVERLOAD: Number of primary key requests rejected at the LQH block instance due to transporter overload
- LQHKEY_OVERLOAD_TC: Count of instances of LQHKEY_OVERLOAD where the TC node transporter was overloaded
- LQHKEY_OVERLOAD_READER: Count of instances of LQHKEY_OVERLOAD where the API reader (reads only) node was overloaded.
- LQHKEY_OVERLOAD_NODE_PEER: Count of instances of LQHKEY_OVERLOAD where the next backup data node (writes only) was overloaded
- LQHKEY_OVERLOAD_SUBSCRIBER: Count of instances of LQHKEY_OVERLOAD where a event subscriber (writes only) was overloaded.
- LQHSCAN_SLOWDOWNS: Count of instances where a fragment scan batch size was reduced due to scanning API transporter overload.

22.5.10.11 The ndbinfo cpustat Table

The cpustat table provides per-thread CPU statistics gathered each second, for each thread running in the NDB kernel.

The following table provides information about the columns in the cpustat table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.376 Columns of the cpustat table

Column Name	Type	Description
node_id	integer	ID of the node where the thread is running
thr_no	integer	Thread ID (specific to this node)
OS_user	integer	OS user time
OS_system	integer	OS system time
OS_idle	integer	OS idle time
thread_exec	integer	Thread execution time
thread_sleeping	integer	Thread sleep time
thread_send	integer	Thread send time
thread_buffer_full	integer	Thread buffer full time
elapsed_time	integer	Elapsed time

22.5.10.12 The ndbinfo cpustat_50ms Table

The cpustat_50ms table provides raw, per-thread CPU data obtained each 50 milliseconds for each thread running in the NDB kernel.

Like cpustat_1sec and cpustat_20sec, this table shows 20 measurement sets per thread, each referencing a period of the named duration. Thus, cpsustat_50ms provides 1 second of history.

The following table provides information about the columns in the cpustat_50ms table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.377 Columns of the cpustat_50ms table

Column Name	Type	Description
node_id	integer	ID of the node where the thread is running
thr_no	integer	Thread ID (specific to this node)
OS_user_time	integer	OS user time
OS_system_time	integer	OS system time
OS_idle_time	integer	OS idle time
exec_time	integer	Thread execution time
sleep_time	integer	Thread sleep time
send_time	integer	Thread send time
buffer_full_time	integer	Thread buffer full time
elapsed_time	integer	Elapsed time

22.5.10.13 The ndbinfo cpustat_1sec Table

The cpustat-1sec table provides raw, per-thread CPU data obtained each second for each thread running in the NDB kernel.

Like cpustat_50ms and cpustat_20sec, this table shows 20 measurement sets per thread, each referencing a period of the named duration. Thus, cpsustat_1sec provides 20 seconds of history.

The following table provides information about the columns in the cpustat_1sec table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.378 Columns of the cpustat_1sec table

Column Name	Type	Description
node_id	integer	ID of the node where the thread is running
thr_no	integer	Thread ID (specific to this node)
OS_user_time	integer	OS user time
OS_system_time	integer	OS system time
OS_idle_time	integer	OS idle time
exec_time	integer	Thread execution time
sleep_time	integer	Thread sleep time
send_time	integer	Thread send time
buffer_full_time	integer	Thread buffer full time
elapsed_time	integer	Elapsed time

22.5.10.14 The ndbinfo cpustat_20sec Table

The cpustat_20sec table provides raw, per-thread CPU data obtained each 20 seconds, for each thread running in the NDB kernel.

Like cpustat_50ms and cpustat_1sec, this table shows 20 measurement sets per thread, each referencing a period of the named duration. Thus, cpsustat_20sec provides 400 seconds of history.

The following table provides information about the columns in the `cpustat_20sec` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.379 Columns of the cpustat_20sec table

Column Name	Type	Description
`node_id`	integer	ID of the node where the thread is running
`thr_no`	integer	Thread ID (specific to this node)
`OS_user_time`	integer	OS user time
`OS_system_time`	integer	OS system time
`OS_idle_time`	integer	OS idle time
`exec_time`	integer	Thread execution time
`sleep_time`	integer	Thread sleep time
`send_time`	integer	Thread send time
`buffer_full_time`	integer	Thread buffer full time
`elapsed_time`	integer	Elapsed time

22.5.10.15 The ndbinfo dict_obj_info Table

The `dict_obj_info` table provides information about `NDB` data dictionary (`DICT`) objects such as tables and indexes. (The `dict_obj_types` table can be queried for a list of all the types.) This information includes the object's type, state, parent object (if any), and fully qualified name.

The following table provides information about the columns in the `dict_obj_info` table. For each column, the table shows the name, data type, and a brief description.

Table 22.380 Columns of the dict_obj_info table

Column Name	Type	Description
`type`	integer	Type of `DICT` object; join on `dict_obj_types` to obtain the name
`id`	integer	Object identifier
`version`	integer	Object version
`state`	integer	Object state
`parent_obj_type`	integer	Parent object's type (a `dict_obj_types` type ID); 0 indicates that the object has no parent
`parent_obj_id`	integer	Parent object ID (such as a base table); 0 indicates that the object has no parent
`fq_name`	string	Fully qualified object name; for a table, this has the form *`database_name`*`/def/`*`table_name`*, for a primary key, the form is `sys/def/`*`table_id`*`/PRIMARY`, and for a unique key it is `sys/def/`*`table_id`*`/`*`uk_name`*`$unique`

22.5.10.16 The ndbinfo dict_obj_types Table

The `dict_obj_types` table is a static table listing possible dictionary object types used in the NDB kernel. These are the same types defined by `Object::Type` in the NDB API.

The following table provides information about the columns in the `dict_obj_types` table. For each column, the table shows the name, data type, and a brief description.

Table 22.381 Columns of the dict_obj_types table

Column Name	Type	Description
`type_id`	integer	The type ID for this type
`type_name`	string	The name of this type

22.5.10.17 The ndbinfo disk_write_speed_base Table

The `disk_write_speed_base` table provides base information about the speed of disk writes during LCP, backup, and restore operations.

The following table provides information about the columns in the `disk_write_speed_base` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.382 Columns of the disk_write_speed_base table

Column Name	Type	Description
node_id	integer	Node ID of this node
thr_no	integer	Thread ID of this LDM thread
millis_ago	integer	Milliseconds since this reporting period ended
millis_passed	integer	Milliseconds elapsed in this reporting period
backup_lcp_bytes_written	integer	Number of bytes written to disk by local checkpoints and backup processes during this period
redo_bytes_written	integer	Number of bytes written to REDO log during this period
target_disk_write_speed	integer	Actual speed of disk writes per LDM thread (base data)

22.5.10.18 The ndbinfo disk_write_speed_aggregate Table

The `disk_write_speed_aggregate` table provides aggregated information about the speed of disk writes during LCP, backup, and restore operations.

The following table provides information about the columns in the `disk_write_speed_aggregate` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.383 Columns in the disk_write_speed_aggregate table

Column Name	Type	Description
node_id	integer	Node ID of this node
thr_no	integer	Thread ID of this LDM thread
backup_lcp_speed_last_sec	integer	Number of bytes written to disk by backup and LCP processes in the last second
redo_speed_last_sec	integer	Number of bytes written to REDO log in the last second

Column Name	Type	Description
backup_lcp_speed_last_10sec	integer	Number of bytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds
redo_speed_last_10sec	integer	Number of bytes written to REDO log per second, averaged over the last 10 seconds
std_dev_backup_lcp_speed_last_10sec	integer	Standard deviation in number of bytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds
std_dev_redo_speed_last_10sec	integer	Standard deviation in number of bytes written to REDO log per second, averaged over the last 10 seconds
backup_lcp_speed_last_60sec	integer	Number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds
redo_speed_last_60sec	integer	Number of bytes written to REDO log per second, averaged over the last 10 seconds
std_dev_backup_lcp_speed_last_60sec	integer	Standard deviation in number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds
std_dev_redo_speed_last_60sec	integer	Standard deviation in number of bytes written to REDO log per second, averaged over the last 60 seconds
slowdowns_due_to_io_lag	integer	Number of seconds since last node start that disk writes were slowed due to REDO log I/O lag
slowdowns_due_to_high_cpu	integer	Number of seconds since last node start that disk writes were slowed due to high CPU usage
disk_write_speed_set_to_min	integer	Number of seconds since last node start that disk write speed was set to minimum
current_target_disk_write_speed	integer	Actual speed of disk writes per LDM thread (aggregated)

22.5.10.19 The ndbinfo disk_write_speed_aggregate_node Table

The `disk_write_speed_aggregate_node` table provides aggregated information per node about the speed of disk writes during LCP, backup, and restore operations.

The following table provides information about the columns in the `disk_write_speed_aggregate_node` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.384 Columns of the disk_write_speed_aggregate_node table

Column Name	Type	Description
node_id	integer	Node ID of this node
backup_lcp_speed_last_sec	integer	Number of bytes written to disk by backup and LCP processes in the last second

Column Name	Type	Description
redo_speed_last_sec	integer	Number of bytes written to REDO log in the last second
backup_lcp_speed_last_10sec	integer	Number of bytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds
redo_speed_last_10sec	integer	Number of bytes written to REDO log per second, averaged over the last 10 seconds
backup_lcp_speed_last_60sec	integer	Number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds
redo_speed_last_60sec	integer	Number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

22.5.10.20 The ndbinfo diskpagebuffer Table

The `diskpagebuffer` table provides statistics about disk page buffer usage by NDB Cluster Disk Data tables.

The following table provides information about the columns in the `diskpagebuffer` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.385 Columns of the diskpagebuffer table

Column Name	Type	Description
`node_id`	integer	The data node ID
`block_instance`	integer	Block instance
`pages_written`	integer	Number of pages written to disk.
`pages_written_lcp`	integer	Number of pages written by local checkpoints.
`pages_read`	integer	Number of pages read from disk
`log_waits`	integer	Number of page writes waiting for log to be written to disk
`page_requests_direct_return`	integer	Number of requests for pages that were available in buffer
`page_requests_wait_queue`	integer	Number of requests that had to wait for pages to become available in buffer
`page_requests_wait_io`	integer	Number of requests that had to be read from pages on disk (pages were unavailable in buffer)

You can use this table with NDB Cluster Disk Data tables to determine whether `DiskPageBufferMemory` is sufficiently large to allow data to be read from the buffer rather from disk; minimizing disk seeks can help improve performance of such tables.

You can determine the proportion of reads from `DiskPageBufferMemory` to the total number of reads using a query such as this one, which obtains this ratio as a percentage:

```
SELECT
  node_id,
  100 * page_requests_direct_return /
    (page_requests_direct_return + page_requests_wait_io)
      AS hit_ratio
FROM ndbinfo.diskpagebuffer;
```

The result from this query should be similar to what is shown here, with one row for each data node in the cluster (in this example, the cluster has 4 data nodes):

```
+---------+-----------+
| node_id | hit_ratio |
+---------+-----------+
|       5 |   97.6744 |
|       6 |   97.6879 |
|       7 |   98.1776 |
|       8 |   98.1343 |
+---------+-----------+
4 rows in set (0.00 sec)
```

`hit_ratio` values approaching 100% indicate that only a very small number of reads are being made from disk rather than from the buffer, which means that Disk Data read performance is approaching an optimum level. If any of these values are less than 95%, this is a strong indicator that the setting for `DiskPageBufferMemory` needs to be increased in the `config.ini` file.

Note

A change in `DiskPageBufferMemory` requires a rolling restart of all of the cluster's data nodes before it takes effect.

`block_instance` refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the `threadblocks` table. Using this information, you can obtain information about disk page buffer metrics relating to individual threads; an example query using `LIMIT 1` to limit the output to a single thread is shown here:

```
mysql> SELECT
     >   node_id, thr_no, block_name, thread_name, pages_written,
     >   pages_written_lcp, pages_read, log_waits,
     >   page_requests_direct_return, page_requests_wait_queue,
     >   page_requests_wait_io
     > FROM ndbinfo.diskpagebuffer
     >   INNER JOIN ndbinfo.threadblocks USING (node_id, block_instance)
     >   INNER JOIN ndbinfo.threads USING (node_id, thr_no)
     > WHERE block_name = 'PGMAN' LIMIT 1\G
*************************** 1. row ***************************
                   node_id: 1
                    thr_no: 1
                block_name: PGMAN
               thread_name: rep
             pages_written: 0
         pages_written_lcp: 0
                pages_read: 1
                 log_waits: 0
page_requests_direct_return: 4
   page_requests_wait_queue: 0
      page_requests_wait_io: 1
1 row in set (0.01 sec)
```

22.5.10.21 The ndbinfo error_messages Table

The `error_messages` table provides information about

The following table provides information about the columns in the `error_messages` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.386 Columns of the error_messages table

Column Name	Type	Description
`error_code`	integer	Numeric error code
`error_description`	string	Description of error
`error_status`	string	Error status code
`error_classification`	integer	Error classification code

`error_code` is a numeric NDB error code. This is the same error code that can be supplied to `ndb_perror` or `perror --ndb`.

`error_description` provides a basic description of the condition causing the error.

The `error_status` column provides status information relating to the error. Possible values for this column are listed here:

- `No error`
- `Illegal connect string`
- `Illegal server handle`
- `Illegal reply from server`
- `Illegal number of nodes`
- `Illegal node status`
- `Out of memory`
- `Management server not connected`
- `Could not connect to socket`
- `Start failed`
- `Stop failed`
- `Restart failed`
- `Could not start backup`
- `Could not abort backup`
- `Could not enter single user mode`
- `Could not exit single user mode`
- `Failed to complete configuration change`
- `Failed to get configuration`
- `Usage error`

- `Success`
- `Permanent error`
- `Temporary error`
- `Unknown result`
- `Temporary error, restart node`
- `Permanent error, external action needed`
- `Ndbd file system error, restart node initial`
- `Unknown`

The error_classification column shows the error classification. See NDB Error Classifications, for information about classification codes and their meanings.

22.5.10.22 The ndbinfo locks_per_fragment Table

The `locks_per_fragment` table provides information about counts of lock claim requests, and the outcomes of these requests on a per-fragment basis, serving as a companion table to `operations_per_fragment` and `memory_per_fragment`. This table also shows the total time spent waiting for locks successfully and unsuccessfully since fragment or table creation, or since the most recent restart.

The following table provides information about the columns in the `locks_per_fragment` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.387 Columns of the locks_per_fragment table

Column Name	Type	Description
`fq_name`	string	Fully qualified table name
`parent_fq_name`	string	Fully qualified name of parent object
`type`	string	Table type; see text for possible values
`table_id`	integer	Table ID
`node_id`	integer	Reporting node ID
`block_instance`	integer	LDM instance ID
`fragment_num`	integer	Fragment identifier
`ex_req`	integer	Exclusive lock requests started
`ex_imm_ok`	integer	Exclusive lock requests immediately granted
`ex_wait_ok`	integer	Exclusive lock requests granted following wait
`ex_wait_fail`	integer	Exclusive lock requests not granted
`sh_req`	integer	Shared lock requests started
`sh_imm_ok`	integer	Shared lock requests immediately granted
`sh_wait_ok`	integer	Shared lock requests granted following wait
`sh_wait_fail`	integer	Shared lock requests not granted
`wait_ok_millis`	integer	Time spent waiting for lock requests that were granted, in milliseconds

Column Name	Type	Description
wait_fail_millis	integer	Time spent waiting for lock requests that failed, in milliseconds

block_instance refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the threadblocks table.

fq_name is a fully qualified database object name in *database*/*schema*/*name* format, such as test/def/t1 or sys/def/10/b$unique.

parent_fq_name is the fully qualified name of this object's parent object (table).

table_id is the table's internal ID generated by NDB. This is the same internal table ID shown in other ndbinfo tables; it is also visible in the output of ndb_show_tables.

The type column shows the type of table. This is always one of System table, User table, Unique hash index, Hash index, Unique ordered index, Ordered index, Hash index trigger, Subscription trigger, Read only constraint, Index trigger, Reorganize trigger, Tablespace, Log file group, Data file, Undo file, Hash map, Foreign key definition, Foreign key parent trigger, Foreign key child trigger, or Schema transaction.

The values shown in all of the columns ex_req, ex_req_imm_ok, ex_wait_ok, ex_wait_fail, sh_req, sh_req_imm_ok, sh_wait_ok, and sh_wait_fail represent cumulative numbers of requests since the table or fragment was created, or since the last restart of this node, whichever of these occurred later. This is also true for the time values shown in the wait_ok_millis and wait_fail_millis columns.

Every lock request is considered either to be in progress, or to have completed in some way (that is, to have succeeded or failed). This means that the following relationships are true:

```
ex_req >= (ex_req_imm_ok + ex_wait_ok + ex_wait_fail)

sh_req >= (sh_req_imm_ok + sh_wait_ok + sh_wait_fail)
```

The number of requests currently in progress is the current number of incomplete requests, which can be found as shown here:

```
[exclusive lock requests in progress] =
    ex_req - (ex_req_imm_ok + ex_wait_ok + ex_wait_fail)

[shared lock requests in progress] =
    sh_req - (sh_req_imm_ok + sh_wait_ok + sh_wait_fail)
```

A failed wait indicates an aborted transaction, but the abort may or may not be caused by a lock wait timeout. You can obtain the total number of aborts while waiting for locks as shown here:

```
[aborts while waiting for locks] = ex_wait_fail + sh_wait_fail
```

22.5.10.23 The ndbinfo logbuffers Table

The logbuffer table provides information on NDB Cluster log buffer usage.

The following table provides information about the columns in the logbuffers table. For each column, the table shows the name, data type, and a brief description.

Table 22.388 Columns in the logbuffers table

Column Name	Type	Description
node_id	integer	The ID of this data node.
log_type	string	Type of log. One of: REDO, DD-UNDO, BACKUP-DATA, or BACKUP-LOG.
log_id	integer	The log ID.
log_part	integer	The log part number.
total	integer	Total space available for this log.
used	integer	Space used by this log.

logbuffers table rows reflecting two additional log types are available when performing an NDB backup. One of these rows has the log type BACKUP-DATA, which shows the amount of data buffer used during backup to copy fragments to backup files. The other row has the log type BACKUP-LOG, which displays the amount of log buffer used during the backup to record changes made after the backup has started. One each of these log_type rows is shown in the logbuffers table for each data node in the cluster. These rows are not present unless an NDB backup is currently being performed.

22.5.10.24 The ndbinfo logspaces Table

This table provides information about NDB Cluster log space usage.

The following table provides information about the columns in the logspaces table. For each column, the table shows the name, data type, and a brief description.

Table 22.389 Columns in the logspaces table

Column Name	Type	Description
node_id	integer	The ID of this data node.
log_type	string	Type of log; one of: REDO or DD-UNDO.
log_id	integer	The log ID.
log_part	integer	The log part number.
total	integer	Total space available for this log.
used	integer	Space used by this log.

22.5.10.25 The ndbinfo membership Table

The membership table describes the view that each data node has of all the others in the cluster, including node group membership, president node, arbitrator, arbitrator successor, arbitrator connection states, and other information.

The following table provides information about the columns in the membership table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.390 Columns of the membership table

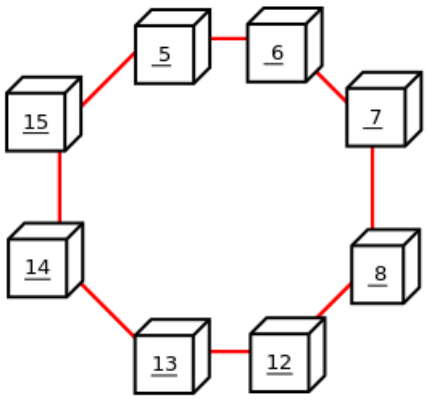
Column Name	Type	Description
node_id	integer	This node's node ID
group_id	integer	Node group to which this node belongs

Column Name	Type	Description
left_node	integer	Node ID of the previous node
right_node	integer	Node ID of the next node
president	integer	President's node ID
successor	integer	Node ID of successor to president
succession_order	integer	Order in which this node succeeds to presidency
Conf_HB_order	integer	-
arbitrator	integer	Node ID of arbitrator
arb_ticket	string	Internal identifier used to track arbitration
arb_state	Enumeration (see text)	Arbitration state
arb_connected	Yes or No	Whether this node is connected to the arbitrator
connected_rank1_arbs	List of node IDs	Connected arbitrators of rank 1
connected_rank2_arbs	List of node IDs	Connected arbitrators of rank 1

The node ID and node group ID are the same as reported by `ndb_mgm -e "SHOW"`.

`left_node` and `right_node` are defined in terms of a model that connects all data nodes in a circle, in order of their node IDs, similar to the ordering of the numbers on a clock dial, as shown here:

Figure 22.26 Circular Arrangement of NDB Cluster Nodes



In this example, we have 8 data nodes, numbered 5, 6, 7, 8, 12, 13, 14, and 15, ordered clockwise in a circle. We determine “left” and “right” from the interior of the circle. The node to the left of node 5 is node 15, and the node to the right of node 5 is node 6. You can see all these relationships by running the following query and observing the output:

```
mysql> SELECT node_id,left_node,right_node
    -> FROM ndbinfo.membership;
+---------+-----------+------------+
| node_id | left_node | right_node |
+---------+-----------+------------+
|       5 |        15 |          6 |
|       6 |         5 |          7 |
|       7 |         6 |          8 |
|       8 |         7 |         12 |
```

```
|      12 |         8 |         13 |
|      13 |        12 |         14 |
|      14 |        13 |         15 |
|      15 |        14 |          5 |
+---------+-----------+------------+
8 rows in set (0.00 sec)
```

The designations “left” and “right” are used in the event log in the same way.

The `president` node is the node viewed by the current node as responsible for setting an arbitrator (see NDB Cluster Start Phases). If the president fails or becomes disconnected, the current node expects the node whose ID is shown in the `successor` column to become the new president. The `succession_order` column shows the place in the succession queue that the current node views itself as having.

In a normal NDB Cluster, all data nodes should see the same node as `president`, and the same node (other than the president) as its `successor`. In addition, the current president should see itself as `1` in the order of succession, the `successor` node should see itself as `2`, and so on.

All nodes should show the same `arb_ticket` values as well as the same `arb_state` values. Possible `arb_state` values are `ARBIT_NULL`, `ARBIT_INIT`, `ARBIT_FIND`, `ARBIT_PREP1`, `ARBIT_PREP2`, `ARBIT_START`, `ARBIT_RUN`, `ARBIT_CHOOSE`, `ARBIT_CRASH`, and `UNKNOWN`.

`arb_connected` shows whether this node is connected to the node shown as this node's `arbitrator`.

The `connected_rank1_arbs` and `connected_rank2_arbs` columns each display a list of 0 or more arbitrators having an `ArbitrationRank` equal to 1, or to 2, respectively.

Note

Both management nodes and API nodes are eligible to become arbitrators.

22.5.10.26 The ndbinfo memoryusage Table

Querying this table provides information similar to that provided by the `ALL REPORT MemoryUsage` command in the `ndb_mgm` client, or logged by `ALL DUMP 1000`.

The following table provides information about the columns in the `memoryusage` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.391 Columns of the memoryusage table

Column Name	Type	Description
`node_id`	integer	The node ID of this data node.
`memory_type`	string	One of `Data memory`, `Index memory`, or `Long message buffer`.
`used`	integer	Number of bytes currently used for data memory or index memory by this data node.
`used_pages`	integer	Number of pages currently used for data memory or index memory by this data node; see text.
`total`	integer	Total number of bytes of data memory or index memory available for this data node; see text.

Column Name	Type	Description
total_pages	integer	Total number of memory pages available for data memory or index memory on this data node; see text.

The total column represents the total amount of memory in bytes available for the given resource (data memory or index memory) on a particular data node. This number should be approximately equal to the setting of the corresponding configuration parameter in the config.ini file.

Suppose that the cluster has 2 data nodes having node IDs 5 and 6, and the config.ini file contains the following:

```
[ndbd default]
DataMemory = 1G
IndexMemory = 1G
```

Suppose also that the value of the LongMessageBuffer configuration parameter is allowed to assume its default (64 MB).

The following query shows approximately the same values:

```
mysql> SELECT node_id, memory_type, total
     > FROM ndbinfo.memoryusage;
+---------+--------------------+------------+
| node_id | memory_type        | total      |
+---------+--------------------+------------+
|       5 | Data memory        | 1073741824 |
|       5 | Index memory       | 1074003968 |
|       5 | Long message buffer |   67108864 |
|       6 | Data memory        | 1073741824 |
|       6 | Index memory       | 1074003968 |
|       6 | Long message buffer |   67108864 |
+---------+--------------------+------------+
6 rows in set (0.00 sec)
```

In this case, the total column values for index memory are slightly higher than the value set of IndexMemory due to internal rounding.

For the used_pages and total_pages columns, resources are measured in pages, which are 32K in size for DataMemory and 8K for IndexMemory. For long message buffer memory, the page size is 256 bytes.

22.5.10.27 The ndbinfo memory_per_fragment Table

The memory_per_fragment table provides information about the usage of memory by individual fragments.

The following table provides information about the columns in the memory_per_fragment table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.392 Columns of the memory_per_fragment table

Column Name	Type	Description
fq_name	string	Name of this fragment
parent_fq_name	string	Name of this fragment's parent

Column Name	Type	Description
type	string	Type of object; see text for possible values
table_id	integer	Table ID for this table
node_id	integer	Node ID for this node
block_instance	integer	Kernel block instance ID
fragment_num	integer	Fragment ID (number)
fixed_elem_alloc_bytes	integer	Number of bytes allocated for fixed-sized elements
fixed_elem_free_bytes	integer	Free bytes remaining in pages allocated to fixed-size elements
fixed_elem_size_bytes	integer	Length of each fixed-size element in bytes
fixed_elem_count	integer	Number of fixed-size elements
fixed_elem_free_count	decimal	Number of free rows for fixed-size elements
var_elem_alloc_bytes	integer	Number of bytes allocated for variable-size elements
var_elem_free_bytes	integer	Free bytes remaining in pages allocated to variable-size elements
var_elem_count	integer	Number of variable-size elements
hash_index_alloc_bytes	integer	Number of bytes allocated to hash indexes

The `type` column from this table shows the dictionary object type used for this fragment (`Object::Type`, in the NDB API), and can take any one of the values shown in the following list:

- System table
- User table
- Unique hash index
- Hash index
- Unique ordered index
- Ordered index
- Hash index trigger
- Subscription trigger
- Read only constraint
- Index trigger
- Reorganize trigger
- Tablespace
- Log file group
- Data file
- Undo file

- Hash map
- Foreign key definition
- Foreign key parent trigger
- Foreign key child trigger
- Schema transaction

You can also obtain this list by executing `SELECT * FROM ndbinfo.dict_obj_types` in the `mysql` client.

The `block_instance` column provides the NDB kernel block instance number. You can use this to obtain information about specific threads from the `threadblocks` table.

22.5.10.28 The ndbinfo nodes Table

This table contains information on the status of data nodes. For each data node that is running in the cluster, a corresponding row in this table provides the node's node ID, status, and uptime. For nodes that are starting, it also shows the current start phase.

The following table provides information about the columns in the `nodes` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.393 Columns of the nodes table

Column Name	Type	Description
`node_id`	integer	The data node's unique node ID in the cluster.
`uptime`	integer	Time since the node was last started, in seconds.
`status`	string	Current status of the data node; see text for possible values.
`start_phase`	integer	If the data node is starting, the current start phase.
`config_generation`	integer	The version of the cluster configuration file in use on this data node.

The `uptime` column shows the time in seconds that this node has been running since it was last started or restarted. This is a `BIGINT` value. This figure includes the time actually needed to start the node; in other words, this counter starts running the moment that `ndbd` or `ndbmtd` is first invoked; thus, even for a node that has not yet finished starting, `uptime` may show a nonzero value.

The `status` column shows the node's current status. This is one of: `NOTHING`, `CMVMI`, `STARTING`, `STARTED`, `SINGLEUSER`, `STOPPING_1`, `STOPPING_2`, `STOPPING_3`, or `STOPPING_4`. When the status is `STARTING`, you can see the current start phase in the `start_phase` column (see later in this section). `SINGLEUSER` is displayed in the `status` column for all data nodes when the cluster is in single user mode (see Section 22.5.8, “NDB Cluster Single User Mode”). Seeing one of the `STOPPING` states does not necessarily mean that the node is shutting down but can mean rather that it is entering a new state. For example, if you put the cluster in single user mode, you can sometimes see data nodes report their state briefly as `STOPPING_2` before the status changes to `SINGLEUSER`.

The `start_phase` column uses the same range of values as those used in the output of the `ndb_mgm` client *`node_id`* `STATUS` command (see Section 22.5.2, “Commands in the NDB Cluster Management

Client"). If the node is not currently starting, then this column shows `0`. For a listing of NDB Cluster start phases with descriptions, see Section 22.5.1, "Summary of NDB Cluster Start Phases".

The `config_generation` column shows which version of the cluster configuration is in effect on each data node. This can be useful when performing a rolling restart of the cluster in order to make changes in configuration parameters. For example, from the output of the following `SELECT` statement, you can see that node 3 is not yet using the latest version of the cluster configuration (`6`) although nodes 1, 2, and 4 are doing so:

```
mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+-------------------+
| node_id | uptime | status  | start_phase | config_generation |
+---------+--------+---------+-------------+-------------------+
|       1 |  10462 | STARTED |           0 |                 6 |
|       2 |  10460 | STARTED |           0 |                 6 |
|       3 |  10457 | STARTED |           0 |                 5 |
|       4 |  10455 | STARTED |           0 |                 6 |
+---------+--------+---------+-------------+-------------------+
2 rows in set (0.04 sec)
```

Therefore, for the case just shown, you should restart node 3 to complete the rolling restart of the cluster.

Nodes that are stopped are not accounted for in this table. Suppose that you have an NDB Cluster with 4 data nodes (node IDs 1, 2, 3 and 4), and all nodes are running normally, then this table contains 4 rows, 1 for each data node:

```
mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+-------------------+
| node_id | uptime | status  | start_phase | config_generation |
+---------+--------+---------+-------------+-------------------+
|       1 |  11776 | STARTED |           0 |                 6 |
|       2 |  11774 | STARTED |           0 |                 6 |
|       3 |  11771 | STARTED |           0 |                 6 |
|       4 |  11769 | STARTED |           0 |                 6 |
+---------+--------+---------+-------------+-------------------+
4 rows in set (0.04 sec)
```

If you shut down one of the nodes, only the nodes that are still running are represented in the output of this `SELECT` statement, as shown here:

```
ndb_mgm> 2 STOP
Node 2: Node shutdown initiated
Node 2: Node shutdown completed.
Node 2 has shutdown.
```

```
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+-------------------+
| node_id | uptime | status  | start_phase | config_generation |
+---------+--------+---------+-------------+-------------------+
|       1 |  11807 | STARTED |           0 |                 6 |
|       3 |  11802 | STARTED |           0 |                 6 |
|       4 |  11800 | STARTED |           0 |                 6 |
+---------+--------+---------+-------------+-------------------+
3 rows in set (0.02 sec)
```

22.5.10.29 The ndbinfo operations_per_fragment Table

The `operations_per_fragment` table provides information about the operations performed on individual fragments and fragment replicas, as well as about some of the results from these operations.

The following table provides information about the columns in the `operations_per_fragment` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.394 Columns of the operations_per_fragment table

Column Name	Type	Description
fq_name	string	Name of this fragment
parent_fq_name	string	Name of this fragment's parent
type	string	Type of object; see text for possible values
table_id	integer	Table ID for this table
node_id	integer	Node ID for this node
block_instance	integer	Kernel block instance ID
fragment_num	integer	Fragment ID (number)
tot_key_reads	integer	Total number of key reads for this fragment replica
tot_key_inserts	integer	Total number of key inserts for this fragment replica
tot_key_updates	integer	total number of key updates for this fragment replica
tot_key_writes	integer	Total number of key writes for this fragment replica
tot_key_deletes	integer	Total number of key deletes for this fragment replica
tot_key_refs	integer	Number of key operations refused
tot_key_attrinfo_bytes	integer	Total size of all `attrinfo` attributes
tot_key_keyinfo_bytes	integer	Total size of all `keyinfo` attributes
tot_key_prog_bytes	integer	Total size of all interpreted programs carried by `attrinfo` attributes
tot_key_inst_exec	integer	Total number of instructions executed by interpreted programs for key operations
tot_key_bytes_returned	integer	Total size of all data and metadata returned from key read operations
tot_frag_scans	integer	Total number of scans performed on this fragment replica
tot_scan_rows_examined	integer	Total number of rows examined by scans
tot_scan_rows_returned	integer	Total number of rows returned to client
tot_scan_bytes_returned	integer	Total size of data and metadata returned to the client
tot_scan_prog_bytes	integer	Total size of interpreted programs for scan operations
tot_scan_bound_bytes	integer	Total size of all bounds used in ordered index scans

Column Name	Type	Description
tot_scan_inst_exec	integer	Total number of instructions executed for scans
tot_qd_frag_scans	integer	Number of times that scans of this fragment replica have been queued
conc_frag_scans	integer	Number of scans currently active on this fragment replica (excluding queued scans)
conc_qd_frag_scans	integer	Number of scans currently queued for this fragment replica
tot_commits	integer	Total number of row changes committed to this fragment replica

The `fq_name` contains the fully qualified name of the schema object to which this fragment replica belongs. This currently has the following formats:

- Base table: - *`DbName`*`/def/`*`TblName`*
- `BLOB` table: - *`DbName`*`/def/NDB$BLOB_`*`BaseTblId_ColNo`*
- Ordered index: - `sys/def/`*`BaseTblId/IndexName`*
- Unique index: - `sys/def/`*`BaseTblId/IndexName`*`$unique`

The `$unique` suffix shown for unique indexes is added by `mysqld`; for an index created by a different NDB API client application, this may differ, or not be present.

The syntax just shown for fully qualified object names is an internal interface which is subject to change in future releases.

Consider a table `t1` created and modified by the following SQL statements:

```
CREATE DATABASE mydb;

USE mydb;

CREATE TABLE t1 (
  a INT NOT NULL,
  b INT NOT NULL,
  t TEXT NOT NULL,
  PRIMARY KEY (b)
) ENGINE=ndbcluster;

CREATE UNIQUE INDEX ix1 ON t1(b) USING HASH;
```

If `t1` is assigned table ID 11, this yields the `fq_name` values shown here:

- Base table: `mydb/def/t1`
- `BLOB` table: `mydb/def/NDB$BLOB_11_2`
- Ordered index (primary key): `sys/def/11/PRIMARY`
- Unique index: `sys/def/11/ix1$unique`

For indexes or `BLOB` tables, the `parent_fq_name` column contains the `fq_name` of the corresponding base table. For base tables, this column is always `NULL`.

The `type` column shows the schema object type used for this fragment, which can take any one of the values `System table`, `User table`, `Unique hash index`, or `Ordered index`. `BLOB` tables are shown as `User table`.

The `table_id` column value is unique at any given time, but can be reused if the corresponding object has been deleted. The same ID can be seen using the `ndb_show_tables` utility.

The `block_instance` column shows which LDM instance this fragment replica belongs to. You can use this to obtain information about specific threads from the `threadblocks` table. The first such instance is always numbered 0.

Since there are typically two replicas, and assuming that this is so, each `fragment_num` value should appear twice in the table, on two different data nodes from the same node group.

Since `NDB` does not use single-key access for ordered indexes, the counts for `tot_key_reads`, `tot_key_inserts`, `tot_key_updates`, `tot_key_writes`, and `tot_key_deletes` are not incremented by ordered index operations.

Note

When using `tot_key_writes`, you should keep in mind that a write operation in this context updates the row if the key exists, and inserts a new row otherwise. (One use of this is in the `NDB` implementation of the `REPLACE` SQL statement.)

The `tot_key_refs` column shows the number of key operations refused by the LDM. Generally, such a refusal is due to duplicate keys (inserts), `Key not found` errors (updates, deletes, and reads), or the operation was rejected by an interpreted program used as a predicate on the row matching the key.

The `attrinfo` and `keyinfo` attributes counted by the `tot_key_attrinfo_bytes` and `tot_key_keyinfo_bytes` columns are attributes of an `LQHKEYREQ` signal (see The NDB Communication Protocol) used to initiate a key operation by the LDM. An `attrinfo` typically contains tuple field values (inserts and updates) or projection specifications (for reads); `keyinfo` contains the primary or unique key needed to locate a given tuple in this schema object.

The value shown by `tot_frag_scans` includes both full scans (that examine every row) and scans of subsets. Unique indexes and `BLOB` tables are never scanned, so this value, like other scan-related counts, is 0 for fragment replicas of these.

`tot_scan_rows_examined` may display less than the total number of rows in a given fragment replica, since ordered index scans can limited by bounds. In addition, a client may choose to end a scan before all potentially matching rows have been examined; this occurs when using an SQL statement containing a `LIMIT` or `EXISTS` clause, for example. `tot_scan_rows_returned` is always less than or equal to `tot_scan_rows_examined`.

`tot_scan_bytes_returned` includes, in the case of pushed joins, projections returned to the `DBSPJ` block in the NDB kernel.

`tot_qd_frag_scans` can be effected by the setting for the `MaxParallelScansPerFragment` data node configuration parameter, which limits the number of scans that may execute concurrently on a single fragment replica.

22.5.10.30 The ndbinfo processes Table

This table contains information about NDB Cluster node processes; each node is represented by the row in the table. Only nodes that are connected to the cluster are shown in this table. You can obtain information about nodes that are configured but not connected to the cluster from the `nodes` and `config_nodes` tables.

The following table provides information about the columns in the processes table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.395 Columns of the nodes table

Column Name	Type	Description
node_id	integer	The node's unique node ID in the cluster
node_type	string	Type of node (management, data, or API node; see text)
node_version	string	Version of the NDB software program running on this node.
process_id	integer	This node's process ID
angel_process_id	integer	Process ID of this node's angel process
process_name	string	Name of the executable
service_URI	string	Service URI of this node (see text)

node_id is the ID assigned to this node in the cluster.

The node_type column displays one of the following three values:

- MGM: Management node.
- NDB: Data node.
- API: API or SQL node.

For an executable shipped with the NDB Cluster distribution, node_version shows the software Cluster version string, such as 8.0.19-ndb-8.0.19.

process_id is the node executable's process ID as shown by the host operating system using a process display application such as top on Linux, or the Task Manager on Windows platforms.

angel_process_id is the system process ID for the node's angel process, which ensures that a data node or SQL is automatically restarted in cases of failures. For management nodes and API nodes other than SQL nodes, the value of this column is NULL.

The process_name column shows the name of the running executable. For management nodes, this is ndb_mgmd. For data nodes, this is ndbd (single-threaded) or ndbmtd (multithreaded). For SQL nodes, this is mysqld. For other types of API nodes, it is the name of the executable program connected to the cluster; NDB API applications can set a custom value for this using Ndb_cluster_connection::set_name().

service_URI shows the service network address. For management nodes and data nodes, the scheme used is ndb://. For SQL nodes, this is mysql://. By default, API nodes other than SQL nodes use ndb:// for the scheme; NDB API applications can set this to a custom value using Ndb_cluster_connection::set_service_uri(). regardless of the node type, the scheme is followed by the IP address used by the NDB transporter for the node in question. For management nodes and SQL nodes, this address includes the port number (usually 1186 for management nodes and 3306 for SQL nodes). If the SQL node was started with the bind_address system variable set, this address is used instead of the transporter address, unless the bind address is set to *, 0.0.0.0, or ::.

Additional path information may be included in the service_URI value for an SQL node reflecting various configuration options. For example, mysql://198.51.100.3/tmp/mysql.sock indicates that the SQL

node was started with `--skip-networking`, and `mysql://198.51.100.3:3306/?server-id=1` shows that replication is enabled for this SQL node.

22.5.10.31 The ndbinfo resources Table

This table provides information about data node resource availability and usage.

These resources are sometimes known as *super-pools*.

The following table provides information about the columns in the `resources` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.396 Columns of the resources table

Column Name	Type	Description
`node_id`	integer	The unique node ID of this data node.
`resource_name`	string	Name of the resource; see text.
`reserved`	integer	The amount reserved for this resource.
`used`	integer	The amount actually used by this resource.
`max`	integer	The maximum amount of this resource used, since the node was last started.

The `resource_name` can be one of the names shown in the following table:

Table 22.397 ndbinfo.resources table resource names and descriptions

Resource name	Description
`RESERVED`	Reserved by the system; cannot be overridden.
`DISK_OPERATIONS`	If a log file group is allocated, the size of the undo log buffer is used to set the size of this resource. This resource is used only to allocate the undo log buffer for an undo log file group; there can only be one such group. Overallocation occurs as needed by `CREATE LOGFILE GROUP`.
`DISK_RECORDS`	Records allocated for Disk Data operations.
`DATA_MEMORY`	Used for main memory tuples, indexes, and hash indexes. Sum of DataMemory and IndexMemory, plus 8 pages of 32 KB each if IndexMemory has been set. Cannot be overallocated.
`JOBBUFFER`	Used for allocating job buffers by the NDB scheduler; cannot be overallocated. This is approximately 2 MB per thread plus a 1 MB buffer in both directions for all threads that can communicate. For large configurations this consume several GB.
`FILE_BUFFERS`	Used by the redo log handler in the `DBLQH` kernel block; cannot be overallocated. Size is `NoOfFragmentLogParts` * `RedoBuffer`, plus 1 MB per log file part.
`TRANSPORTER_BUFFERS`	Used for send buffers by `ndbmtd`; the sum of `TotalSendBufferMemory` and `ExtraSendBufferMemory`. This resource that can be overallocated by up to 25 percent. `TotalSendBufferMemory` is calculated by summing the send buffer memory per node, the default value of which is 2 MB. Thus, in a system having four data nodes and eight API nodes, the data nodes have 12 * 2 MB send buffer memory. `ExtraSendBufferMemory` is used by `ndbmtd` and amounts to 2 MB extra memory per thread. Thus, with 4 LDM threads, 2 TC threads, 1 main thread, 1 replication thread, and 2 receive threads,

Resource name	Description
	ExtraSendBufferMemory is 10 * 2 MB. Overallocation of this resource can be performed by setting the SharedGlobalMemory data node configuration parameter.
DISK_PAGE_BUFFER	Used for the disk page buffer; determined by the DiskPageBufferMemory configuration parameter. Cannot be overallocated.
QUERY_MEMORY	Used by the DBSPJ kernel block.
SCHEMA_TRANS_MEMORY	Minimum is 2 MB; can be overallocated to use any remaining available memory.

22.5.10.32 The ndbinfo restart_info Table

The restart_info table contains information about node restart operations. Each entry in the table corresponds to a node restart status report in real time from a data node with the given node ID. Only the most recent report for any given node is shown.

The following table provides information about the columns in the restart_info table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.398 Columns of the restart_info table

Column Name	Type	Description
node_id	integer	Node ID in the cluster
node_restart_status	VARCHAR(256)	Node status; see text for values. Each of these corresponds to a possible value of node_restart_status_int.
node_restart_status_int	integer	Node status code; see text for values.
secs_to_complete_node_failure	integer	Time in seconds to complete node failure handling
secs_to_allocate_node_id	integer	Time in seconds from node failure completion to allocation of node ID
secs_to_include_in_heartbeat_protocol	integer	Time in seconds from allocation of node ID to inclusion in heartbeat protocol
secs_until_wait_for_ndbcntr_master	integer	Time in seconds from being included in heartbeat protocol until waiting for NDBCNTR master began
secs_wait_for_ndbcntr_master	integer	Time in seconds spent waiting to be accepted by NDBCNTR master for starting
secs_to_get_start_permitted	integer	Time in seconds elapsed from receiving of permission for start from master until all nodes have accepted start of this node
secs_to_wait_for_lcp_for_copy_meta_data	integer	Time in seconds spent waiting for LCP completion before copying meta data
secs_to_copy_meta_data	integer	Time in seconds required to copy metadata from master to newly starting node
secs_to_include_node	integer	Time in seconds waited for GCP and inclusion of all nodes into protocols

Column Name	Type	Description
secs_starting_node_to_request_local_recovery	integer	Time in seconds that the node just starting spent waiting to request local recovery
secs_for_local_recovery	integer	Time in seconds required for local recovery by node just starting
secs_restore_fragments	integer	Time in seconds required to restore fragments from LCP files
secs_undo_disk_data	integer	Time in seconds required to execute undo log on disk data part of records
secs_exec_redo_log	integer	Time in seconds required to execute redo log on all restored fragments
secs_index_rebuild	integer	Time in seconds required to rebuild indexes on restored fragments
secs_to_synchronize_starting_node	integer	Time in seconds required to synchronize starting node from live nodes
secs_wait_lcp_for_restart	integer	Time in seconds required for LCP start and completion before restart was completed
secs_wait_subscription_handover	integer	Time in seconds spent waiting for handover of replication subscriptions
total_restart_secs	integer	Total number of seconds from node failure until node is started again

Defined values for node_restart_status_int and corresponding status names and messages (node_restart_status) are shown in the following table:

Table 22.399 Status codes and messages used in the restart_info table

Column Name	Type	Description
0	ALLOCATED_NODE_ID	Allocated node id
1	INCLUDED_IN_HB_PROTOCOL	Included in heartbeat protocol
2	NDBCNTR_START_WAIT	Wait for NDBCNTR master to permit us to start
3	NDBCNTR_STARTED	NDBCNTR master permitted us to start
4	START_PERMITTED	All nodes permitted us to start
5	WAIT_LCP_TO_COPY_DICT	Wait for LCP completion to start copying metadata
6	COPY_DICT_TO_STARTING_NODE	Copying metadata to starting node
7	INCLUDE_NODE_IN_LCP_AND_GCP	Include node in LCP and GCP protocols
8	LOCAL_RECOVERY_STARTED	Restore fragments ongoing
9	COPY_FRAGMENTS_STARTED	Synchronizing starting node with live nodes
10	WAIT_LCP_FOR_RESTART	Wait for LCP to ensure durability
11	WAIT_SUMA_HANDOVER	Wait for handover of subscriptions
12	RESTART_COMPLETED	Restart completed

Column Name	Type	Description
13	NODE_FAILED	Node failed, failure handling in progress
14	NODE_FAILURE_COMPLETED	Node failure handling completed
15	NODE_GETTING_PERMIT	All nodes permitted us to start
16	NODE_GETTING_INCLUDED	Include node in LCP and GCP protocols
17	NODE_GETTING_SYNCHED	Synchronizing starting node with live nodes
18	NODE_GETTING_LCP_WAITED	[none]
19	NODE_ACTIVE	Restart completed
20	NOT_DEFINED_IN_CLUSTER	[none]
21	NODE_NOT_RESTARTED_YET	Initial state

Status numbers 0 through 12 apply on master nodes only; the remainder of those shown in the table apply to all restarting data nodes. Status numbers 13 and 14 define node failure states; 20 and 21 occur when no information about the restart of a given node is available.

See also Section 22.5.1, “Summary of NDB Cluster Start Phases”.

22.5.10.33 The ndbinfo server_locks Table

The `server_locks` table is similar in structure to the `cluster_locks` table, and provides a subset of the information found in the latter table, but which is specific to the SQL node (MySQL server) where it resides. (The `cluster_locks` table provides information about all locks in the cluster.) More precisely, `server_locks` contains information about locks requested by threads belonging to the current `mysqld` instance, and serves as a companion table to `server_operations`. This may be useful for correlating locking patterns with specific MySQL user sessions, queries, or use cases.

The following table provides information about the columns in the `server_locks` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.400 Columns of the server_locks table

Column Name	Type	Description
mysql_connection_id	integer	MySQL connection ID
node_id	integer	ID of reporting node
block_instance	integer	ID of reporting LDM instance
tableid	integer	ID of table containing this row
fragmentid	integer	ID of fragment containing locked row
rowid	integer	ID of locked row
transid	integer	Transaction ID
mode	string	Lock request mode
state	string	Lock state
detail	string	Whether this is first holding lock in row lock queue

Column Name	Type	Description
op	string	Operation type
duration_millis	integer	Milliseconds spent waiting or holding lock
lock_num	integer	ID of lock object
waiting_for	integer	Waiting for lock with this ID

The mysql_connection_id column shows the MySQL connection or thread ID as shown by SHOW PROCESSLIST.

block_instance refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the threadblocks table.

The tableid is assigned to the table by NDB; the same ID is used for this table in other ndbinfo tables, as well as in the output of ndb_show_tables.

The transaction ID shown in the transid column is the identifier generated by the NDB API for the transaction requesting or holding the current lock.

The mode column shows the lock mode, which is always one of S (shared lock) or X (exclusive lock). If a transaction has an exclusive lock on a given row, all other locks on that row have the same transaction ID.

The state column shows the lock state. Its value is always one of H (holding) or W (waiting). A waiting lock request waits for a lock held by a different transaction.

The detail column indicates whether this lock is the first holding lock in the affected row's lock queue, in which case it contains a * (asterisk character); otherwise, this column is empty. This information can be used to help identify the unique entries in a list of lock requests.

The op column shows the type of operation requesting the lock. This is always one of the values READ, INSERT, UPDATE, DELETE, SCAN, or REFRESH.

The duration_millis column shows the number of milliseconds for which this lock request has been waiting or holding the lock. This is reset to 0 when a lock is granted for a waiting request.

The lock ID (lockid column) is unique to this node and block instance.

If the lock_state column's value is W, this lock is waiting to be granted, and the waiting_for column shows the lock ID of the lock object this request is waiting for. Otherwise, waiting_for is empty. waiting_for can refer only to locks on the same row (as identified by node_id, block_instance, tableid, fragmentid, and rowid).

22.5.10.34 The ndbinfo server_operations Table

The server_operations table contains entries for all ongoing NDB operations that the current SQL node (MySQL Server) is currently involved in. It effectively is a subset of the cluster_operations table, in which operations for other SQL and API nodes are not shown.

The following table provides information about the columns in the server_operations table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.401 Columns of the server_operations table

Column Name	Type	Description
mysql_connection_id	integer	MySQL Server connection ID

Column Name	Type	Description
node_id	integer	Node ID
block_instance	integer	Block instance
transid	integer	Transaction ID
operation_type	string	Operation type (see text for possible values)
state	string	Operation state (see text for possible values)
tableid	integer	Table ID
fragmentid	integer	Fragment ID
client_node_id	integer	Client node ID
client_block_ref	integer	Client block reference
tc_node_id	integer	Transaction coordinator node ID
tc_block_no	integer	Transaction coordinator block number
tc_block_instance	integer	Transaction coordinator block instance

The mysql_connection_id is the same as the connection or session ID shown in the output of SHOW PROCESSLIST. It is obtained from the INFORMATION_SCHEMA table NDB_TRANSID_MYSQL_CONNECTION_MAP.

block_instance refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the threadblocks table.

The transaction ID (transid) is a unique 64-bit number which can be obtained using the NDB API's getTransactionId() method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The operation_type column can take any one of the values READ, READ-SH, READ-EX, INSERT, UPDATE, DELETE, WRITE, UNLOCK, REFRESH, SCAN, SCAN-SH, SCAN-EX, or <unknown>.

The state column can have any one of the values ABORT_QUEUED, ABORT_STOPPED, COMMITTED, COMMIT_QUEUED, COMMIT_STOPPED, COPY_CLOSE_STOPPED, COPY_FIRST_STOPPED, COPY_STOPPED, COPY_TUPKEY, IDLE, LOG_ABORT_QUEUED, LOG_COMMIT_QUEUED, LOG_COMMIT_QUEUED_WAIT_SIGNAL, LOG_COMMIT_WRITTEN, LOG_COMMIT_WRITTEN_WAIT_SIGNAL, LOG_QUEUED, PREPARED, PREPARED_RECEIVED_COMMIT, SCAN_CHECK_STOPPED, SCAN_CLOSE_STOPPED, SCAN_FIRST_STOPPED, SCAN_RELEASE_STOPPED, SCAN_STATE_USED, SCAN_STOPPED, SCAN_TUPKEY, STOPPED, TC_NOT_CONNECTED, WAIT_ACC, WAIT_ACC_ABORT, WAIT_AI_AFTER_ABORT, WAIT_ATTR, WAIT_SCAN_AI, WAIT_TUP, WAIT_TUPKEYINFO, WAIT_TUP_COMMIT, or WAIT_TUP_TO_ABORT. (If the MySQL Server is running with ndbinfo_show_hidden enabled, you can view this list of states by selecting from the ndb$dblqh_tcconnect_state table, which is normally hidden.)

You can obtain the name of an NDB table from its table ID by checking the output of ndb_show_tables.

The fragid is the same as the partition number seen in the output of ndb_desc --extra-partition-info (short form -p).

In client_node_id and client_block_ref, client refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The block_instance and tc_block_instance column provide NDB kernel block instance numbers. You can use these to obtain information about specific threads from the threadblocks table.

22.5.10.35 The ndbinfo server_transactions Table

The `server_transactions` table is subset of the `cluster_transactions` table, but includes only those transactions in which the current SQL node (MySQL Server) is a participant, while including the relevant connection IDs.

The following table provides information about the columns in the `server_transactions` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.402 Columns of the server_transactions table

Column Name	Type	Description
`mysql_connection_id`	integer	MySQL Server connection ID
`node_id`	integer	Transaction coordinator node ID
`block_instance`	integer	Transaction coordinator block instance
`transid`	integer	Transaction ID
`state`	string	Operation state (see text for possible values)
`count_operations`	integer	Number of stateful operations in the transaction
`outstanding_operations`	integer	Operations still being executed by local data management layer (LQH blocks)
`inactive_seconds`	integer	Time spent waiting for API
`client_node_id`	integer	Client node ID
`client_block_ref`	integer	Client block reference

The `mysql_connection_id` is the same as the connection or session ID shown in the output of `SHOW PROCESSLIST`. It is obtained from the `INFORMATION_SCHEMA` table `NDB_TRANSID_MYSQL_CONNECTION_MAP`.

`block_instance` refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the `threadblocks` table.

The transaction ID (`transid`) is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The `state` column can have any one of the values `CS_ABORTING`, `CS_COMMITTING`, `CS_COMMIT_SENT`, `CS_COMPLETE_SENT`, `CS_COMPLETING`, `CS_CONNECTED`, `CS_DISCONNECTED`, `CS_FAIL_ABORTED`, `CS_FAIL_ABORTING`, `CS_FAIL_COMMITTED`, `CS_FAIL_COMMITTING`, `CS_FAIL_COMPLETED`, `CS_FAIL_PREPARED`, `CS_PREPARE_TO_COMMIT`, `CS_RECEIVING`, `CS_REC_COMMITTING`, `CS_RESTART`, `CS_SEND_FIRE_TRIG_REQ`, `CS_STARTED`, `CS_START_COMMITTING`, `CS_START_SCAN`, `CS_WAIT_ABORT_CONF`, `CS_WAIT_COMMIT_CONF`, `CS_WAIT_COMPLETE_CONF`, `CS_WAIT_FIRE_TRIG_REQ`. (If the MySQL Server is running with `ndbinfo_show_hidden` enabled, you can view this list of states by selecting from the `ndb$dbtc_apiconnect_state` table, which is normally hidden.)

In `client_node_id` and `client_block_ref`, `client` refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The `block_instance` column provides the `DBTC` kernel block instance number. You can use this to obtain information about specific threads from the `threadblocks` table.

22.5.10.36 The ndbinfo table_distribution_status Table

The `table_distribution_status` table provides information about the progress of table distribution for `NDB` tables.

The following table provides information about the columns in `table_distribution_status`. For each column, the table shows the name, data type, and a brief description.

Table 22.403 Columns of the table_distribution_status table

Column Name	Type	Description
`node_id`	integer	Node id
`table_id`	integer	Table ID
`tab_copy_status`	string	Status of copying of table distribution data to disk; one of `IDLE`, `SR_PHASE1_READ_PAGES`, `SR_PHASE2_READ_TABLE`, `SR_PHASE3_COPY_TABLE`, `REMOVE_NODE`, `LCP_READ_TABLE`, `COPY_TAB_REQ`, `COPY_NODE_STATE`, `ADD_TABLE_MASTER`, `ADD_TABLE_SLAVE`, `INVALIDATE_NODE_LCP`, `ALTER_TABLE`, `COPY_TO_SAVE`, or `GET_TABINFO`
`tab_update_status`	string	Status of updating of table distribution data; one of `IDLE`, `LOCAL_CHECKPOINT`, `LOCAL_CHECKPOINT_QUEUED`, `REMOVE_NODE`, `COPY_TAB_REQ`, `ADD_TABLE_MASTER`, `ADD_TABLE_SLAVE`, `INVALIDATE_NODE_LCP`, or `CALLBACK`
`tab_lcp_status`	string	Status of table LCP; one of `ACTIVE` (waiting for local checkpoint to be performed), `WRITING_TO_FILE` (checkpoint performed but not yet written to disk), or `COMPLETED` (checkpoint performed and persisted to disk)
`tab_status`	string	Table internal status; one of `ACTIVE` (table exists), `CREATING` (table is being created), or `DROPPING` (table is being dropped)
`tab_storage`	string	Table recoverability; one of `NORMAL` (fully recoverable with redo logging and checkpointing), `NOLOGGING` (recoverable from node crash, empty following cluster crash), or `TEMPORARY` (not recoverable)
`tab_partitions`	integer	Number of partitions in table
`tab_fragments`	integer	Number of fragments in table; normally same as `tab_partitions`; for fully replicated tables equal to `tab_partitions * [number of node groups]`
`current_scan_count`	integer	Current number of active scans
`scan_count_wait`	integer	Current number of scans waiting to be performed before `ALTER TABLE` can complete.

Column Name	Type	Description
is_reorg_ongoing	integer	Whether table is currently being reorganized (1 if true)

22.5.10.37 The ndbinfo table_fragments Table

The table_fragments table provides information about the fragmentation, partitioning, distribution, and (internal) replication of NDB tables.

The following table provides information about the columns in table_fragments. For each column, the table shows the name, data type, and a brief description.

Table 22.404 Columns of the table_fragments table

Column Name	Type	Description
node_id	integer	Node ID (DIH master)
table_id	integer	Table ID
partition_id	integer	Partition ID
fragment_id	integer	Fragment ID (same as partition ID unless table is fully replicated)
partition_order	integer	Order of fragment in partition
log_part_id	integer	Log part ID of fragment
no_of_replicas	integer	Number of replicas
current_primary	integer	Current primary node ID
preferred_primary	integer	Preferred primary node ID
current_first_backup	integer	Current first backup node ID
current_second_backup	integer	Current second backup node ID
current_third_backup	integer	Current third backup node ID
num_alive_replicas	integer	Current number of live replicas
num_dead_replicas	integer	Current number of dead replicas
num_lcp_replicas	integer	Number of replicas remaining to be checkpointed

22.5.10.38 The ndbinfo table_info Table

The table_info table provides information about logging, checkpointing, distribution, and storage options in effect for individual NDB tables.

The following table provides information about the columns in table_info. For each column, the table shows the name, data type, and a brief description.

Table 22.405 Columns of the table_info table

Column Name	Type	Description
table_id	integer	Table ID
logged_table	integer	Whether table is logged (1) or not (0)

Column Name	Type	Description
row_contains_gci	integer	Whether table rows contain GCI (1 true, 0 false)
row_contains_checksum	integer	Whether table rows contain checksum (1 true, 0 false)
read_backup	integer	If backup replicas are read this is 1, otherwise 0
fully_replicated	integer	If table is fully replicated this is 1, otherwise 0
storage_type	string	Table storage type; one of MEMORY or DISK
hashmap_id	integer	Hashmap ID
partition_balance	string	Partition balance (fragment count type) used for table; one of FOR_RP_BY_NODE, FOR_RA_BY_NODE, FOR_RP_BY_LDM, or FOR_RA_BY_LDM
create_gci	integer	GCI in which table was created

22.5.10.39 The ndbinfo table_replicas Table

The table_replicas table provides information about the copying, distribution, and checkpointing of NDB table fragments and fragment replicas.

The following table provides information about the columns in table_replicas. For each column, the table shows the name, data type, and a brief description.

Table 22.406 Columns of the table_replicas table

Column Name	Type	Description
node_id	integer	ID of the node from which data is fetched (DIH master)
table_id	integer	Table ID
fragment_id	integer	Fragment ID
initial_gci	integer	Initial GCI for table
replica_node_id	integer	ID of node where replica is stored
is_lcp_ongoing	integer	Is 1 if LCP is ongoing on this fragment, 0 otherwise
num_crashed_replicas	integer	Number of crashed replica instances
last_max_gci_started	integer	Highest GCI started in most recent LCP
last_max_gci_completed	integer	Highest GCI completed in most recent LCP
last_lcp_id	integer	ID of most recent LCP
prev_lcp_id	integer	ID of previous LCP
prev_max_gci_started	integer	Highest GCI started in previous LCP
prev_max_gci_completed	integer	Highest GCI completed in previous LCP
last_create_gci	integer	Last Create GCI of last crashed replica instance
last_replica_gci	integer	Last GCI of last crashed replica instance

Column Name	Type	Description
is_replica_alive	integer	1 if this replica is alive, 0 otherwise

22.5.10.40 The ndbinfo tc_time_track_stats Table

The tc_time_track_stats table provides time-tracking information obtained from the DBTC block (TC) instances in the data nodes, through API nodes access NDB. Each TC instance tracks latencies for a set of activities it undertakes on behalf of API nodes or other data nodes; these activities include transactions, transaction errors, key reads, key writes, unique index operations, failed key operations of any type, scans, failed scans, fragment scans, and failed fragment scans.

A set of counters is maintained for each activity, each counter covering a range of latencies less than or equal to an upper bound. At the conclusion of each activity, its latency is determined and the appropriate counter incremented. tc_time_track_stats presents this information as rows, with a row for each instance of the following:

- Data node, using its ID
- TC block instance
- Other communicating data node or API node, using its ID
- Upper bound value

Each row contains a value for each activity type. This is the number of times that this activity occurred with a latency within the range specified by the row (that is, where the latency does not exceed the upper bound).

The following table provides information about the columns in tc_time_track_stats. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.407 Columns of the tc_time_track_stats table

Column Name	Type	Description
node_id	integer	Requesting node ID
block_number	integer	TC block number
block_instance	integer	TC block instance number
comm_node_id	integer	Node ID of communicating API or data node
upper_bound	integer	Upper bound of interval (in microseconds)
scans	integer	Based on duration of successful scans from opening to closing, tracked against the API or data nodes requesting them.
scan_errors	integer	Based on duration of failed scans from opening to closing, tracked against the API or data nodes requesting them.
scan_fragments	integer	Based on duration of successful fragment scans from opening to closing, tracked against the data nodes executing them
scan_fragment_errors	integer	Based on duration of failed fragment scans from opening to closing, tracked against the data nodes executing them

Column Name	Type	Description
transactions	integer	Based on duration of successful transactions from beginning until sending of commit ACK, tracked against the API or data nodes requesting them. Stateless transactions are not included.
transaction_errors	integer	Based on duration of failing transactions from start to point of failure, tracked against the API or data nodes requesting them.
read_key_ops	integer	Based on duration of successful primary key reads with locks. Tracked against both the API or data node requesting them and the data node executing them.
write_key_ops	integer	Based on duration of successful primary key writes, tracked against both the API or data node requesting them and the data node executing them.
index_key_ops	integer	Based on duration of successful unique index key operations, tracked against both the API or data node requesting them and the data node executing reads of base tables.
key_op_errors	integer	Based on duration of all unsuccessful key read or write operations, tracked against both the API or data node requesting them and the data node executing them.

The block_instance column provides the DBTC kernel block instance number. You can use this together with the block name to obtain information about specific threads from the threadblocks table.

22.5.10.41 The ndbinfo threadblocks Table

The threadblocks table associates data nodes, threads, and instances of NDB kernel blocks.

The following table provides information about the columns in the threadblocks table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.408 Columns of the threadblocks table

Column Name	Type	Description
node_id	integer	Node ID
thr_no	integer	Thread ID
block_name	string	Block name
block_instance	integer	Block instance number

The value of the block_name in this table is one of the values found in the block_name column when selecting from the ndbinfo.blocks table. Although the list of possible values is static for a given NDB Cluster release, the list may vary between releases.

The block_instance column provides the kernel block instance number.

22.5.10.42 The ndbinfo threads Table

The `threads` table provides information about threads running in the `NDB` kernel.

The following table provides information about the columns in the `threads` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.409 Columns of the threads table

Column Name	Type	Description
`node_id`	integer	ID of the node where the thread is running
`thr_no`	integer	Thread ID (specific to this node)
`thread_name`	string	Thread name (type of thread)
`thread_description`	string	Thread (type) description

Sample output from a 2-node example cluster, including thread descriptions, is shown here:

```
mysql> SELECT * FROM threads;
+---------+--------+-------------+-------------------------------------------------------------------+
| node_id | thr_no | thread_name | thread_description                                                |
+---------+--------+-------------+-------------------------------------------------------------------+
|       5 |      0 | main        | main thread, schema and distribution handling                     |
|       5 |      1 | rep         | rep thread, asynch replication and proxy block handling           |
|       5 |      2 | ldm         | ldm thread, handling a set of data partitions                     |
|       5 |      3 | recv        | receive thread, performing receive and polling for new receives |
|       6 |      0 | main        | main thread, schema and distribution handling                     |
|       6 |      1 | rep         | rep thread, asynch replication and proxy block handling           |
|       6 |      2 | ldm         | ldm thread, handling a set of data partitions                     |
|       6 |      3 | recv        | receive thread, performing receive and polling for new receives |
+---------+--------+-------------+-------------------------------------------------------------------+
8 rows in set (0.01 sec)
```

22.5.10.43 The ndbinfo threadstat Table

The `threadstat` table provides a rough snapshot of statistics for threads running in the `NDB` kernel.

The following table provides information about the columns in the `threadstat` table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.410 Columns of the threadstat table

Column Name	Type	Description
`node_id`	integer	Node ID
`thr_no`	integer	Thread ID
`thr_nm`	string	Thread name
`c_loop`	string	Number of loops in main loop
`c_exec`	string	Number of signals executed
`c_wait`	string	Number of times waiting for additional input
`c_l_sent_prioa`	integer	Number of priority A signals sent to own node
`c_l_sent_priob`	integer	Number of priority B signals sent to own node

Column Name	Type	Description
c_r_sent_prioa	integer	Number of priority A signals sent to remote node
c_r_sent_priob	integer	Number of priority B signals sent to remote node
os_tid	integer	OS thread ID
os_now	integer	OS time (ms)
os_ru_utime	integer	OS user CPU time (µs)
os_ru_stime	integer	OS system CPU time (µs)
os_ru_minflt	integer	OS page reclaims (soft page faults)
os_ru_majflt	integer	OS page faults (hard page faults)
os_ru_nvcsw	integer	OS voluntary context switches
os_ru_nivcsw	integer	OS involuntary context switches

os_time uses the system gettimeofday() call.

The values of the os_ru_utime, os_ru_stime, os_ru_minflt, os_ru_majflt, os_ru_nvcsw, and os_ru_nivcsw columns are obtained using the system getrusage() call, or the equivalent.

Since this table contains counts taken at a given point in time, for best results it is necessary to query this table periodically and store the results in an intermediate table or tables. The MySQL Server's Event Scheduler can be employed to automate such monitoring. For more information, see Section 24.4, “Using the Event Scheduler”.

22.5.10.44 The ndbinfo transporters Table

This table contains information about NDB transporters.

The following table provides information about the columns in the transporters table. For each column, the table shows the name, data type, and a brief description. Additional information can be found in the notes following the table.

Table 22.411 Columns of the transporters table

Column Name	Type	Description
node_id	integer	This data node's unique node ID in the cluster
remote_node_id	integer	The remote data node's node ID
status	string	Status of the connection
remote_address	string	Name or IP address of the remote host
bytes_sent	integer	Number of bytes sent using this connection
bytes_received	integer	Number of bytes received using this connection
connect_count	integer	Number of times connection established on this transporter
overloaded	boolean (0 or 1)	1 if this transporter is currently overloaded, otherwise 0
overload_count	integer	Number of times this transporter has entered overload state since connecting

Column Name	Type	Description
slowdown	boolean (0 or 1)	1 if this transporter is in slowdown state, otherwise 0
slowdown_count	integer	Number of times this transporter has entered slowdown state since connecting

For each running data node in the cluster, the transporters table displays a row showing the status of each of that node's connections with all nodes in the cluster, *including itself*. This information is shown in the table's *status* column, which can have any one of the following values: CONNECTING, CONNECTED, DISCONNECTING, or DISCONNECTED.

Connections to API and management nodes which are configured but not currently connected to the cluster are shown with status DISCONNECTED. Rows where the node_id is that of a data node which is not currently connected are not shown in this table. (This is similar omission of disconnected nodes in the ndbinfo.nodes table.

The remote_address is the host name or address for the node whose ID is shown in the remote_node_id column. The bytes_sent from this node and bytes_received by this node are the numbers, respectively, of bytes sent and received by the node using this connection since it was established. For nodes whose status is CONNECTING or DISCONNECTED, these columns always display 0.

Assume you have a 5-node cluster consisting of 2 data nodes, 2 SQL nodes, and 1 management node, as shown in the output of the SHOW command in the ndb_mgm client:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @10.100.10.1  (8.0.19-ndb-8.0.19, Nodegroup: 0, *)
id=2    @10.100.10.2  (8.0.19-ndb-8.0.19, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @10.100.10.10  (8.0.19-ndb-8.0.19)

[mysqld(API)]   2 node(s)
id=20   @10.100.10.20  (8.0.19-ndb-8.0.19)
id=21   @10.100.10.21  (8.0.19-ndb-8.0.19)
```

There are 10 rows in the transporters table—5 for the first data node, and 5 for the second—assuming that all data nodes are running, as shown here:

```
mysql> SELECT node_id, remote_node_id, status
    ->   FROM ndbinfo.transporters;
+---------+----------------+---------------+
| node_id | remote_node_id | status        |
+---------+----------------+---------------+
|       1 |              1 | DISCONNECTED  |
|       1 |              2 | CONNECTED     |
|       1 |             10 | CONNECTED     |
|       1 |             20 | CONNECTED     |
|       1 |             21 | CONNECTED     |
|       2 |              1 | CONNECTED     |
|       2 |              2 | DISCONNECTED  |
|       2 |             10 | CONNECTED     |
|       2 |             20 | CONNECTED     |
|       2 |             21 | CONNECTED     |
+---------+----------------+---------------+
10 rows in set (0.04 sec)
```

If you shut down one of the data nodes in this cluster using the command `2 STOP` in the `ndb_mgm` client, then repeat the previous query (again using the `mysql` client), this table now shows only 5 rows—1 row for each connection from the remaining management node to another node, including both itself and the data node that is currently offline—and displays `CONNECTING` for the status of each remaining connection to the data node that is currently offline, as shown here:

```
mysql> SELECT node_id, remote_node_id, status
    ->   FROM ndbinfo.transporters;
+---------+----------------+--------------+
| node_id | remote_node_id | status       |
+---------+----------------+--------------+
|       1 |              1 | DISCONNECTED |
|       1 |              2 | CONNECTING   |
|       1 |             10 | CONNECTED    |
|       1 |             20 | CONNECTED    |
|       1 |             21 | CONNECTED    |
+---------+----------------+--------------+
5 rows in set (0.02 sec)
```

The `connect_count`, `overloaded`, `overload_count`, `slowdown`, and `slowdown_count` counters are reset on connection, and retain their values after the remote node disconnects. The `bytes_sent` and `bytes_received` counters are also reset on connection, and so retain their values following disconnection (until the next connection resets them).

The *overload* state referred to by the `overloaded` and `overload_count` columns occurs when this transporter's send buffer contains more than `OVerloadLimit` bytes (default is 80% of `SendBufferMemory`, that is, 0.8 * 2097152 = 1677721 bytes). When a given transporter is in a state of overload, any new transaction that tries to use this transporter fails with Error 1218 (`Send Buffers overloaded in NDB kernel`). This affects both scans and primary key operations.

The *slowdown* state referenced by the `slowdown` and `slowdown_count` columns of this table occurs when the transporter's send buffer contains more than 60% of the overload limit (equal to 0.6 * 2097152 = 1258291 bytes by default). In this state, any new scan using this transporter has its batch size reduced to minimize the load on the transporter.

Common causes of send buffer slowdown or overloading include the following:

- Data size, in particular the quantity of data stored in `TEXT` columns or `BLOB` columns (or both types of columns)
- Having a data node (ndbd or ndbmtd) on the same host as an SQL node that is engaged in binary logging
- Large number of rows per transaction or transaction batch
- Configuration issues such as insufficient `SendBufferMemory`
- Hardware issues such as insufficient RAM or poor network connectivity

See also Section 22.3.3.13, “Configuring NDB Cluster Send Buffer Parameters”.

22.5.11 INFORMATION_SCHEMA Tables for NDB Cluster

Two `INFORMATION_SCHEMA` tables provide information that is of particular use when managing an NDB Cluster . The `FILES` table provides information about NDB Cluster Disk Data files. The `ndb_transid_mysql_connection_map` table provides a mapping between transactions, transaction coordinators, and API nodes.

Additional statistical and other data about NDB Cluster transactions, operations, threads, blocks, and other aspects of performance can be obtained from the tables in the `ndbinfo` database. For information about these tables, see Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”.

22.5.12 NDB Cluster Security Issues

This section discusses security considerations to take into account when setting up and running NDB Cluster.

Topics covered in this section include the following:

- NDB Cluster and network security issues
- Configuration issues relating to running NDB Cluster securely
- NDB Cluster and the MySQL privilege system
- MySQL standard security procedures as applicable to NDB Cluster

22.5.12.1 NDB Cluster Security and Networking Issues

In this section, we discuss basic network security issues as they relate to NDB Cluster. It is extremely important to remember that NDB Cluster “out of the box” is not secure; you or your network administrator must take the proper steps to ensure that your cluster cannot be compromised over the network.

Cluster communication protocols are inherently insecure, and no encryption or similar security measures are used in communications between nodes in the cluster. Because network speed and latency have a direct impact on the cluster's efficiency, it is also not advisable to employ SSL or other encryption to network connections between nodes, as such schemes will effectively slow communications.

It is also true that no authentication is used for controlling API node access to an NDB Cluster. As with encryption, the overhead of imposing authentication requirements would have an adverse impact on Cluster performance.

In addition, there is no checking of the source IP address for either of the following when accessing the cluster:

- SQL or API nodes using “free slots” created by empty `[mysqld]` or `[api]` sections in the `config.ini` file

  This means that, if there are any empty `[mysqld]` or `[api]` sections in the `config.ini` file, then any API nodes (including SQL nodes) that know the management server's host name (or IP address) and port can connect to the cluster and access its data without restriction. (See Section 22.5.12.2, “NDB Cluster and MySQL Privileges”, for more information about this and related issues.)

Note

You can exercise some control over SQL and API node access to the cluster by specifying a `HostName` parameter for all `[mysqld]` and `[api]` sections in the `config.ini` file. However, this also means that, should you wish to connect an API node to the cluster from a previously unused host, you need to add an `[api]` section containing its host name to the `config.ini` file.

More information is available elsewhere in this chapter about the `HostName` parameter. Also see Section 22.3.1, “Quick Test Setup of NDB Cluster”, for configuration examples using `HostName` with API nodes.

- Any `ndb_mgm` client

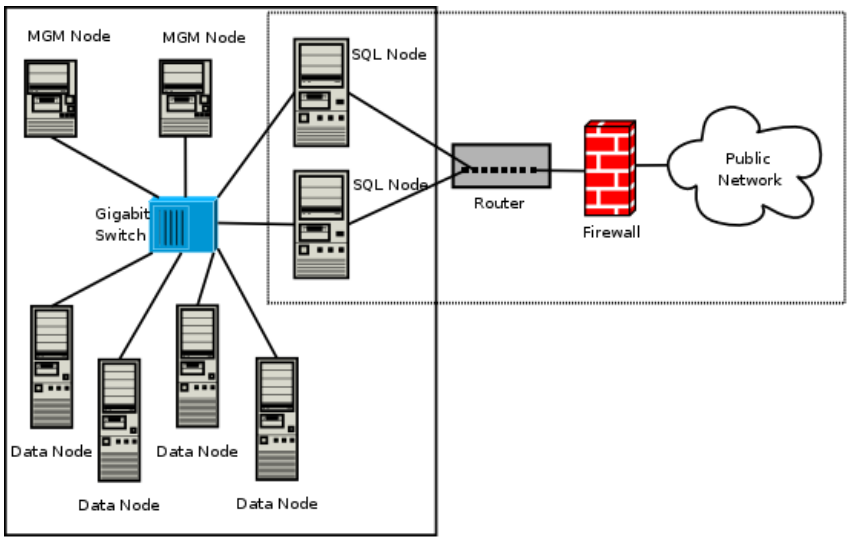
  This means that any cluster management client that is given the management server's host name (or IP address) and port (if not the standard port) can connect to the cluster and execute any management client command. This includes commands such as `ALL STOP` and `SHUTDOWN`.

For these reasons, it is necessary to protect the cluster on the network level. The safest network configuration for Cluster is one which isolates connections between Cluster nodes from any other network communications. This can be accomplished by any of the following methods:

1. Keeping Cluster nodes on a network that is physically separate from any public networks. This option is the most dependable; however, it is the most expensive to implement.

   We show an example of an NDB Cluster setup using such a physically segregated network here:

   Figure 22.27 NDB Cluster with Hardware Firewall

   This setup has two networks, one private (solid box) for the Cluster management servers and data nodes, and one public (dotted box) where the SQL nodes reside. (We show the management and data nodes connected using a gigabit switch since this provides the best performance.) Both networks are protected from the outside by a hardware firewall, sometimes also known as a *network-based firewall*.

   This network setup is safest because no packets can reach the cluster's management or data nodes from outside the network—and none of the cluster's internal communications can reach the outside—without going through the SQL nodes, as long as the SQL nodes do not permit any packets to be forwarded. This means, of course, that all SQL nodes must be secured against hacking attempts.

   Important

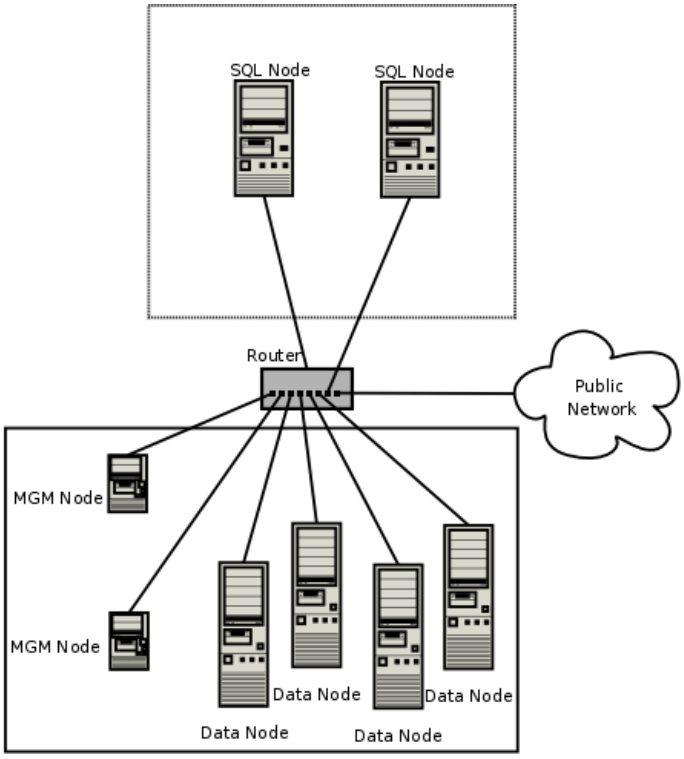
   With regard to potential security vulnerabilities, an SQL node is no different from any other MySQL server. See Section 6.1.3, “Making MySQL Secure Against Attackers”, for a description of techniques you can use to secure MySQL servers.

2. Using one or more software firewalls (also known as *host-based firewalls*) to control which packets pass through to the cluster from portions of the network that do not require access to it. In this type of setup, a software firewall must be installed on every host in the cluster which might otherwise be accessible from outside the local network.

The host-based option is the least expensive to implement, but relies purely on software to provide protection and so is the most difficult to keep secure.

This type of network setup for NDB Cluster is illustrated here:

Figure 22.28 NDB Cluster with Software Firewalls



Using this type of network setup means that there are two zones of NDB Cluster hosts. Each cluster host must be able to communicate with all of the other machines in the cluster, but only those hosting SQL nodes (dotted box) can be permitted to have any contact with the outside, while those in the zone containing the data nodes and management nodes (solid box) must be isolated from any machines that are not part of the cluster. Applications using the cluster and user of those applications must *not* be permitted to have direct access to the management and data node hosts.

To accomplish this, you must set up software firewalls that limit the traffic to the type or types shown in the following table, according to the type of node that is running on each cluster host computer:

Table 22.412 Node types in a host-based firewall cluster configuration

Node Type	Permitted Traffic
SQL or API node	• It originates from the IP address of a management or data node (using any TCP or UDP port). • It originates from within the network in which the cluster resides and is on the port that your application is using.
Data node or Management node	• It originates from the IP address of a management or data node (using any TCP or UDP port). • It originates from the IP address of an SQL or API node.

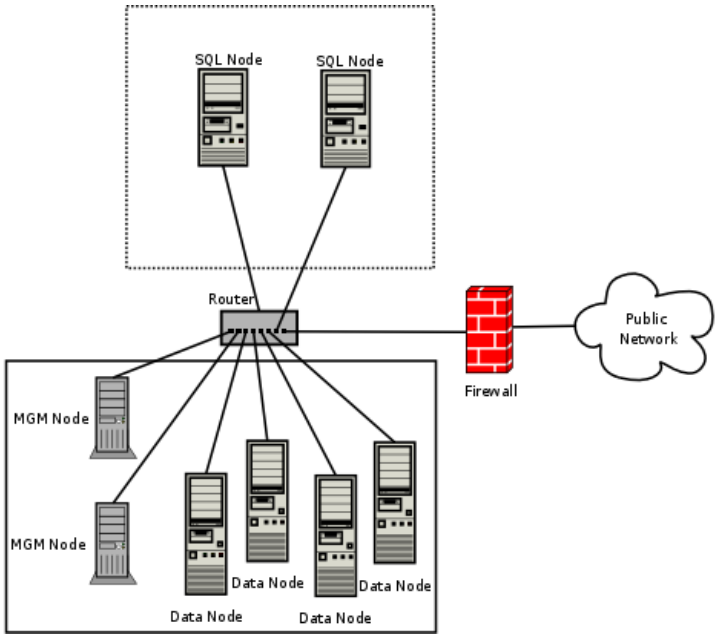
Any traffic other than that shown in the table for a given node type should be denied.

The specifics of configuring a firewall vary from firewall application to firewall application, and are beyond the scope of this Manual. `iptables` is a very common and reliable firewall application, which is often used with `APF` as a front end to make configuration easier. You can (and should) consult the documentation for the software firewall that you employ, should you choose to implement an NDB Cluster network setup of this type, or of a “mixed” type as discussed under the next item.

3. It is also possible to employ a combination of the first two methods, using both hardware and software to secure the cluster—that is, using both network-based and host-based firewalls. This is between the first two schemes in terms of both security level and cost. This type of network setup keeps the cluster behind the hardware firewall, but permits incoming packets to travel beyond the router connecting all cluster hosts to reach the SQL nodes.

   One possible network deployment of an NDB Cluster using hardware and software firewalls in combination is shown here:

   Figure 22.29 NDB Cluster with a Combination of Hardware and Software Firewalls



   In this case, you can set the rules in the hardware firewall to deny any external traffic except to SQL nodes and API nodes, and then permit traffic to them only on the ports required by your application.

Whatever network configuration you use, remember that your objective from the viewpoint of keeping the cluster secure remains the same—to prevent any unessential traffic from reaching the cluster while ensuring the most efficient communication between the nodes in the cluster.

Because NDB Cluster requires large numbers of ports to be open for communications between nodes, the recommended option is to use a segregated network. This represents the simplest way to prevent unwanted traffic from reaching the cluster.

Note

If you wish to administer an NDB Cluster remotely (that is, from outside the local network), the recommended way to do this is to use `ssh` or another secure

login shell to access an SQL node host. From this host, you can then run the management client to access the management server safely, from within the Cluster's own local network.

Even though it is possible to do so in theory, it is *not* recommended to use `ndb_mgm` to manage a Cluster directly from outside the local network on which the Cluster is running. Since neither authentication nor encryption takes place between the management client and the management server, this represents an extremely insecure means of managing the cluster, and is almost certain to be compromised sooner or later.

22.5.12.2 NDB Cluster and MySQL Privileges

In this section, we discuss how the MySQL privilege system works in relation to NDB Cluster and the implications of this for keeping an NDB Cluster secure.

Standard MySQL privileges apply to NDB Cluster tables. This includes all MySQL privilege types (`SELECT` privilege, `UPDATE` privilege, `DELETE` privilege, and so on) granted on the database, table, and column level. As with any other MySQL Server, user and privilege information is stored in the `mysql` system database. The SQL statements used to grant and revoke privileges on `NDB` tables, databases containing such tables, and columns within such tables are identical in all respects with the `GRANT` and `REVOKE` statements used in connection with database objects involving any (other) MySQL storage engine. The same thing is true with respect to the `CREATE USER` and `DROP USER` statements.

It is important to keep in mind that, by default, the MySQL grant tables use the `MyISAM` storage engine. Because of this, those tables are not normally duplicated or shared among MySQL servers acting as SQL nodes in an NDB Cluster. In other words, changes in users and their privileges do not automatically propagate between SQL nodes by default. If you wish, you can enable automatic distribution of MySQL users and privileges across NDB Cluster SQL nodes; see Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”, for details.

Conversely, because there is no way in MySQL to deny privileges (privileges can either be revoked or not granted in the first place, but not denied as such), there is no special protection for `NDB` tables on one SQL node from users that have privileges on another SQL node; (This is true even if you are not using automatic distribution of user privileges. The definitive example of this is the MySQL `root` account, which can perform any action on any database object. In combination with empty `[mysqld]` or `[api]` sections of the `config.ini` file, this account can be especially dangerous. To understand why, consider the following scenario:

- The `config.ini` file contains at least one empty `[mysqld]` or `[api]` section. This means that the NDB Cluster management server performs no checking of the host from which a MySQL Server (or other API node) accesses the NDB Cluster.
- There is no firewall, or the firewall fails to protect against access to the NDB Cluster from hosts external to the network.
- The host name or IP address of the NDB Cluster management server is known or can be determined from outside the network.

If these conditions are true, then anyone, anywhere can start a MySQL Server with `--ndbcluster --ndb-connectstring=`*`management_host`* and access this NDB Cluster. Using the MySQL `root` account, this person can then perform the following actions:

- Execute metadata statements such as `SHOW DATABASES` statement (to obtain a list of all `NDB` databases on the server) or `SHOW TABLES FROM` *`some_ndb_database`* statement to obtain a list of all `NDB` tables in a given database

- Run any legal MySQL statements on any of the discovered tables, such as:
  - `SELECT * FROM some_table` to read all the data from any table
  - `DELETE FROM some_table` to delete all the data from a table
  - `DESCRIBE some_table` or `SHOW CREATE TABLE some_table` to determine the table schema
  - `UPDATE some_table SET column1 = some_value` to fill a table column with “garbage” data; this could actually cause much greater damage than simply deleting all the data

    More insidious variations might include statements like these:

    ```
    UPDATE some_table SET an_int_column = an_int_column + 1
    ```

    or

    ```
    UPDATE some_table SET a_varchar_column = REVERSE(a_varchar_column)
    ```

    Such malicious statements are limited only by the imagination of the attacker.

  The only tables that would be safe from this sort of mayhem would be those tables that were created using storage engines other than `NDB`, and so not visible to a “rogue” SQL node.

  A user who can log in as `root` can also access the `INFORMATION_SCHEMA` database and its tables, and so obtain information about databases, tables, stored routines, scheduled events, and any other database objects for which metadata is stored in `INFORMATION_SCHEMA`.

  It is also a very good idea to use different passwords for the `root` accounts on different NDB Cluster SQL nodes unless you are using distributed privileges.

In sum, you cannot have a safe NDB Cluster if it is directly accessible from outside your local network.

Important

Never leave the MySQL root account password empty. This is just as true when running MySQL as an NDB Cluster SQL node as it is when running it as a standalone (non-Cluster) MySQL Server, and should be done as part of the MySQL installation process before configuring the MySQL Server as an SQL node in an NDB Cluster.

If you wish to employ NDB Cluster's distributed privilege capabilities, you should not simply convert the system tables in the `mysql` database to use the `NDB` storage engine manually. Use the stored procedure provided for this purpose instead; see Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”.

Otherwise, if you need to synchronize `mysql` system tables between SQL nodes, you can use standard MySQL replication to do so, or employ a script to copy table entries between the MySQL servers.

Summary. The most important points to remember regarding the MySQL privilege system with regard to NDB Cluster are listed here:

1. Users and privileges established on one SQL node do not automatically exist or take effect on other SQL nodes in the cluster. Conversely, removing a user or privilege on one SQL node in the cluster does not remove the user or privilege from any other SQL nodes.

2. You can distribute MySQL users and privileges among SQL nodes using the SQL script, and the stored procedures it contains, that are supplied for this purpose in the NDB Cluster distribution.

3. Once a MySQL user is granted privileges on an `NDB` table from one SQL node in an NDB Cluster, that user can “see” any data in that table regardless of the SQL node from which the data originated, even if you are not using privilege distribution.

22.5.12.3 NDB Cluster and MySQL Security Procedures

In this section, we discuss MySQL standard security procedures as they apply to running NDB Cluster.

In general, any standard procedure for running MySQL securely also applies to running a MySQL Server as part of an NDB Cluster. First and foremost, you should always run a MySQL Server as the `mysql` operating system user; this is no different from running MySQL in a standard (non-Cluster) environment. The `mysql` system account should be uniquely and clearly defined. Fortunately, this is the default behavior for a new MySQL installation. You can verify that the `mysqld` process is running as the `mysql` operating system user by using the system command such as the one shown here:

```
shell> ps aux | grep mysql
root     10467  0.0  0.1   3616  1380 pts/3    S    11:53   0:00 \
  /bin/sh ./mysqld_safe --ndbcluster --ndb-connectstring=localhost:1186
mysql    10512  0.2  2.5  58528 26636 pts/3    Sl   11:53   0:00 \
  /usr/local/mysql/libexec/mysqld --basedir=/usr/local/mysql \
  --datadir=/usr/local/mysql/var --user=mysql --ndbcluster \
  --ndb-connectstring=localhost:1186 --pid-file=/usr/local/mysql/var/mothra.pid \
  --log-error=/usr/local/mysql/var/mothra.err
jon      10579  0.0  0.0   2736   688 pts/0    S+   11:54   0:00 grep mysql
```

If the `mysqld` process is running as any other user than `mysql`, you should immediately shut it down and restart it as the `mysql` user. If this user does not exist on the system, the `mysql` user account should be created, and this user should be part of the `mysql` user group; in this case, you should also make sure that the MySQL data directory on this system (as set using the `--datadir` option for `mysqld`) is owned by the `mysql` user, and that the SQL node's `my.cnf` file includes `user=mysql` in the `[mysqld]` section. Alternatively, you can start the MySQL server process with `--user=mysql` on the command line, but it is preferable to use the `my.cnf` option, since you might forget to use the command-line option and so have `mysqld` running as another user unintentionally. The `mysqld_safe` startup script forces MySQL to run as the `mysql` user.

Important

Never run `mysqld` as the system root user. Doing so means that potentially any file on the system can be read by MySQL, and thus—should MySQL be compromised—by an attacker.

As mentioned in the previous section (see Section 22.5.12.2, “NDB Cluster and MySQL Privileges”), you should always set a root password for the MySQL Server as soon as you have it running. You should also delete the anonymous user account that is installed by default. You can accomplish these tasks using the following statements:

```
shell> mysql -u root

mysql> UPDATE mysql.user
    ->     SET Password=PASSWORD('secure_password')
    ->     WHERE User='root';

mysql> DELETE FROM mysql.user
    ->     WHERE User='';
```

```
mysql> FLUSH PRIVILEGES;
```

Be very careful when executing the `DELETE` statement not to omit the `WHERE` clause, or you risk deleting *all* MySQL users. Be sure to run the `FLUSH PRIVILEGES` statement as soon as you have modified the `mysql.user` table, so that the changes take immediate effect. Without `FLUSH PRIVILEGES`, the changes do not take effect until the next time that the server is restarted.

Note

Many of the NDB Cluster utilities such as `ndb_show_tables`, `ndb_desc`, and `ndb_select_all` also work without authentication and can reveal table names, schemas, and data. By default these are installed on Unix-style systems with the permissions `wxr-xr-x` (755), which means they can be executed by any user that can access the `mysql/bin` directory.

See Section 22.4, “NDB Cluster Programs”, for more information about these utilities.

22.5.13 NDB Cluster Disk Data Tables

NDB Cluster supports storing nonindexed columns of `NDB` tables on disk, rather than in RAM. Column data and logging metadata are kept in data files and undo log files, conceptualized as tablespaces and log file groups, as described in the next section—see Section 22.5.13.1, “NDB Cluster Disk Data Objects”.

NDB Cluster Disk Data performance can be influenced by a number of configuration parameters. For information about these parameters and their effects, see Disk Data Configuration Parameters, and Disk Data and GCP Stop errors.

The performance of an NDB Cluster that uses Disk Data storage can also be greatly improved by separating data node file systems from undo log files and tablespace data files, which can be done using symbolic links. For more information, see Section 22.5.13.2, “Using Symbolic Links with Disk Data Objects”.

22.5.13.1 NDB Cluster Disk Data Objects

NDB Cluster Disk Data storage is implemented using the following objects:

- *Tablespace*: Acts as containers for other Disk Data objects. A tablespace contains one or more data files and one or more undo log file groups.
- *Data file*: Stores column data. A data file is assigned directly to a tablespace.
- *Undo log file*: Contains undo information required for rolling back transactions. Assigned to an undo log file group.
- *log file group*: Contains one or more undo log files. Assigned to a tablespace.

Undo log files and data files are actual files in the file system of each data node; by default they are placed in `ndb_node_id_fs` in the `DataDir` specified in the NDB Cluster `config.ini` file, and where `node_id` is the data node's node ID. It is possible to place these elsewhere by specifying either an absolute or relative path as part of the filename when creating the undo log or data file. Statements that create these files are shown later in this section.

Undo log files are used only by Disk Data tables, and are not needed or used by `NDB` tables that are stored in memory only.

NDB Cluster tablespaces and log file groups are not implemented as files.

Although not all Disk Data objects are implemented as files, they all share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a log file group both named `dd1`.

Assuming that you have already set up an NDB Cluster with all nodes (including management and SQL nodes), the basic steps for creating an NDB Cluster table on disk are as follows:

1. Create a log file group, and assign one or more undo log files to it (an undo log file is also sometimes referred to as an *undofile*).
2. Create a tablespace; assign the log file group, as well as one or more data files, to the tablespace.
3. Create a Disk Data table that uses this tablespace for data storage.

Each of these tasks can be accomplished using SQL statements in the `mysql` client or other MySQL client application, as shown in the example that follows.

1. We create a log file group named `lg_1` using `CREATE LOGFILE GROUP`. This log file group is to be made up of two undo log files, which we name `undo_1.log` and `undo_2.log`, whose initial sizes are 16 MB and 12 MB, respectively. (The default initial size for an undo log file is 128 MB.) Optionally, you can also specify a size for the log file group's undo buffer, or permit it to assume the default value of 8 MB. In this example, we set the UNDO buffer's size at 2 MB. A log file group must be created with an undo log file; so we add `undo_1.log` to `lg_1` in this `CREATE LOGFILE GROUP` statement:

   ```
   CREATE LOGFILE GROUP lg_1
       ADD UNDOFILE 'undo_1.log'
       INITIAL_SIZE 16M
       UNDO_BUFFER_SIZE 2M
       ENGINE NDBCLUSTER;
   ```

   To add `undo_2.log` to the log file group, use the following `ALTER LOGFILE GROUP` statement:

   ```
   ALTER LOGFILE GROUP lg_1
       ADD UNDOFILE 'undo_2.log'
       INITIAL_SIZE 12M
       ENGINE NDBCLUSTER;
   ```

   Some items of note:

   - The `.log` file extension used here is not required. We employ it merely to make the log files easily recognizable.
   - Every `CREATE LOGFILE GROUP` and `ALTER LOGFILE GROUP` statement must include an `ENGINE` option. The only permitted values for this option are `NDBCLUSTER` and `NDB`.

     Important

     There can exist at most one log file group in the same NDB Cluster at any given time.

   - When you add an undo log file to a log file group using `ADD UNDOFILE 'filename'`, a file with the name *`filename`* is created in the `ndb_node_id_fs` directory within the `DataDir` of each data node in the cluster, where *`node_id`* is the node ID of the data node. Each undo log file is of the size specified in the SQL statement. For example, if an NDB Cluster has 4 data nodes, then the `ALTER LOGFILE GROUP` statement just shown creates 4 undo log files, 1 each on in the data directory of each of the 4 data nodes; each of these files is named `undo_2.log` and each file is 12 MB in size.

- `UNDO_BUFFER_SIZE` is limited by the amount of system memory available.
- See Section 13.1.16, “CREATE LOGFILE GROUP Syntax”, and Section 13.1.6, “ALTER LOGFILE GROUP Syntax”, for more information about these statements.

2. Now we can create a tablespace—an abstract container for files used by Disk Data tables to store data. A tablespace is associated with a particular log file group; when creating a new tablespace, you must specify the log file group it uses for undo logging. You must also specify at least one data file; you can add more data files to the tablespace after the tablespace is created. It is also possible to drop data files from a tablespace (see example later in this section).

   Assume that we wish to create a tablespace named `ts_1` which uses `lg_1` as its log file group. We want the tablespace to contain two data files, named `data_1.dat` and `data_2.dat`, whose initial sizes are 32 MB and 48 MB, respectively. (The default value for `INITIAL_SIZE` is 128 MB.) We can do this using two SQL statements, as shown here:

```
CREATE TABLESPACE ts_1
    ADD DATAFILE 'data_1.dat'
    USE LOGFILE GROUP lg_1
    INITIAL_SIZE 32M
    ENGINE NDBCLUSTER;

ALTER TABLESPACE ts_1
    ADD DATAFILE 'data_2.dat'
    INITIAL_SIZE 48M;
```

   The `CREATE TABLESPACE` statement creates a tablespace `ts_1` with the data file `data_1.dat`, and associates `ts_1` with log file group `lg_1`. The `ALTER TABLESPACE` adds the second data file (`data_2.dat`).

   Some items of note:

   - As is the case with the `.log` file extension used in this example for undo log files, there is no special significance for the `.dat` file extension; it is used merely for easy recognition.
   - When you add a data file to a tablespace using `ADD DATAFILE '`*`filename`*`'`, a file with the name *`filename`* is created in the `ndb_`*`node_id`*`_fs` directory within the `DataDir` of each data node in the cluster, where *`node_id`* is the node ID of the data node. Each data file is of the size specified in the SQL statement. For example, if an NDB Cluster has 4 data nodes, then the `ALTER TABLESPACE` statement just shown creates 4 data files, 1 each in the data directory of each of the 4 data nodes; each of these files is named `data_2.dat` and each file is 48 MB in size.
   - `CREATE TABLESPACE` statements must contain an `ENGINE` clause; only tables using the same storage engine as the tablespace can be created in the tablespace. For `ALTER TABLESPACE`, an `ENGINE` clause is accepted but is deprecated and subject to removal in a future release. For `NDB` tablespaces, the only permitted values for this option are `NDBCLUSTER` and `NDB`.
   - For more information about the `CREATE TABLESPACE` and `ALTER TABLESPACE` statements, see Section 13.1.21, “CREATE TABLESPACE Syntax”, and Section 13.1.10, “ALTER TABLESPACE Syntax”.

3. Now it is possible to create a table whose unindexed columns are stored on disk using files in tablespace `ts_1`:

```
CREATE TABLE dt_1 (
    member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
```

```
    last_name VARCHAR(50) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    dob DATE NOT NULL,
    joined DATE NOT NULL,
    INDEX(last_name, first_name)
    )
    TABLESPACE ts_1 STORAGE DISK
    ENGINE NDBCLUSTER;
```

`TABLESPACE ts_1 STORAGE DISK` tells the `NDB` storage engine to use tablespace `ts_1` for data storage on disk.

Once table `ts_1` has been created as shown, you can perform `INSERT`, `SELECT`, `UPDATE`, and `DELETE` statements on it just as you would with any other MySQL table.

It is also possible to specify whether an individual column is stored on disk or in memory by using a `STORAGE` clause as part of the column's definition in a `CREATE TABLE` or `ALTER TABLE` statement. `STORAGE DISK` causes the column to be stored on disk, and `STORAGE MEMORY` causes in-memory storage to be used. See Section 13.1.20, “CREATE TABLE Syntax”, for more information.

You can obtain information about the `NDB` disk data files and undo log files just created by querying the `FILES` table in the `INFORMATION_SCHEMA` database, as shown here:

```
mysql> SELECT
             FILE_NAME AS File, FILE_TYPE AS Type,
             TABLESPACE_NAME AS Tablespace, TABLE_NAME AS Name,
             LOGFILE_GROUP_NAME AS 'File group',
             FREE_EXTENTS AS Free, TOTAL_EXTENTS AS Total
          FROM INFORMATION_SCHEMA.FILES
          WHERE ENGINE='ndbcluster';
+--------------+----------+------------+------+------------+------+---------+
| File         | Type     | Tablespace | Name | File group | Free | Total   |
+--------------+----------+------------+------+------------+------+---------+
| ./undo_1.log | UNDO LOG | lg_1       | NULL | lg_1       |    0 | 4194304 |
| ./undo_2.log | UNDO LOG | lg_1       | NULL | lg_1       |    0 | 3145728 |
| ./data_1.dat | DATAFILE | ts_1       | NULL | lg_1       |   32 |      32 |
| ./data_2.dat | DATAFILE | ts_1       | NULL | lg_1       |   48 |      48 |
+--------------+----------+------------+------+------------+------+---------+
4 rows in set (0.00 sec)
```

For more information and examples, see Section 25.11, “The INFORMATION_SCHEMA FILES Table”.

Indexing of columns implicitly stored on disk. For table `dt_1` as defined in the example just shown, only the `dob` and `joined` columns are stored on disk. This is because there are indexes on the `id`, `last_name`, and `first_name` columns, and so data belonging to these columns is stored in RAM. Only nonindexed columns can be held on disk; indexes and indexed column data continue to be stored in memory. This tradeoff between the use of indexes and conservation of RAM is something you must keep in mind as you design Disk Data tables.

You cannot add an index to a column that has been explicitly declared `STORAGE DISK`, without first changing its storage type to `MEMORY`; any attempt to do so fails with an error. A column which *implicitly* uses disk storage can be indexed; when this is done, the column's storage type is changed to `MEMORY` automatically. By “implicitly”, we mean a column whose storage type is not declared, but which is which inherited from the parent table. In the following CREATE TABLE statement (using the tablespace `ts_1` defined previously), columns `c2` and `c3` use disk storage implicitly:

```
mysql> CREATE TABLE ti (
    ->     c1 INT PRIMARY KEY,
    ->     c2 INT,
```

```
    ->     c3 INT,
    ->     c4 INT
    -> )
    ->     STORAGE DISK
    ->     TABLESPACE ts_1
    ->     ENGINE NDBCLUSTER;
Query OK, 0 rows affected (1.31 sec)
```

Because `c2`, `c3`, and `c4` are themselves not declared with `STORAGE DISK`, it is possible to index them. Here, we add indexes to `c2` and `c3`, using, respectively, `CREATE INDEX` and `ALTER TABLE`:

```
mysql> CREATE INDEX i1 ON ti(c2);
Query OK, 0 rows affected (2.72 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE ti ADD INDEX i2(c3);
Query OK, 0 rows affected (0.92 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

`SHOW CREATE TABLE` confirms that the indexes were added.

```
mysql> SHOW CREATE TABLE ti\G
*************************** 1. row ***************************
       Table: ti
Create Table: CREATE TABLE `ti` (
  `c1` int(11) NOT NULL,
  `c2` int(11) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  `c4` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`),
  KEY `i1` (`c2`),
  KEY `i2` (`c3`)
) /*!50100 TABLESPACE `ts_1` STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

You can see using `ndb_desc` that the indexed columns (emphasized text) now use in-memory rather than on-disk storage:

```
shell> ./ndb_desc -d test t1
-- t1 --
Version: 33554433
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 317
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 4
FragmentCount: 4
PartitionBalance: FOR_RP_BY_LDM
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options:
HashMap: DEFAULT-HASHMAP-3840-4
-- Attributes --
```

```
c1 Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
c2 Int NULL AT=FIXED ST=MEMORY
c3 Int NULL AT=FIXED ST=MEMORY
c4 Int NULL AT=FIXED ST=DISK
-- Indexes --
PRIMARY KEY(c1) - UniqueHashIndex
i2(c3) - OrderedIndex
PRIMARY(c1) - OrderedIndex
i1(c2) - OrderedIndex

NDBT_ProgramExit: 0 - OK
```

Performance note. The performance of a cluster using Disk Data storage is greatly improved if Disk Data files are kept on a separate physical disk from the data node file system. This must be done for each data node in the cluster to derive any noticeable benefit.

You can use absolute and relative file system paths with `ADD UNDOFILE` and `ADD DATAFILE`; relative paths are calculated with respect to the data node's data directory. You can also use symbolic links to point to a directory on a different disk; see Section 22.5.13.2, “Using Symbolic Links with Disk Data Objects”, for more information and examples.

A log file group, a tablespace, and any Disk Data tables using these must be created in a particular order. This is also true for dropping these objects, sibject to the following constraints:

- A log file group cannot be dropped as long as any tablespaces use it.
- A tablespace cannot be dropped as long as it contains any data files.
- You cannot drop any data files from a tablespace as long as there remain any tables which are using the tablespace.
- It is not possible to drop files created in association with a different tablespace other than the one with which the files were created.

For example, to drop all the objects created so far in this section, you can use the following statements:

```
mysql> DROP TABLE dt_1;

mysql> ALTER TABLESPACE ts_1
    -> DROP DATAFILE 'data_2.dat'
    -> ENGINE NDBCLUSTER;

mysql> ALTER TABLESPACE ts_1
    -> DROP DATAFILE 'data_1.dat'
    -> ENGINE NDBCLUSTER;

mysql> DROP TABLESPACE ts_1
    -> ENGINE NDBCLUSTER;

mysql> DROP LOGFILE GROUP lg_1
    -> ENGINE NDBCLUSTER;
```

These statements must be performed in the order shown, except that the two `ALTER TABLESPACE ... DROP DATAFILE` statements may be executed in either order.

22.5.13.2 Using Symbolic Links with Disk Data Objects

The performance of an NDB Cluster that uses Disk Data storage can be greatly improved by separating the data node file system from any tablespace files (undo log files and data files), and placing these on different disks. In early versions of NDB Cluster, there was no direct support for this in NDB Cluster, but

it was possible to achieve this separation using symbolic links as described in this section. NDB Cluster supports the data node configuration parameters `FileSystemPathDD`, `FileSystemPathDataFiles`, and `FileSystemPathUndoFiles`, which make the use of symbolic links for this purpose unnecessary. For more information about these parameters, see Disk Data file system parameters.

Each data node in the cluster creates a file system in the directory named `ndb_node_id_fs` under the data node's `DataDir` as defined in the `config.ini` file. In this example, we assume that each data node host has 3 disks, aliased as `/data0`, `/data1`, and `/data2`, and that the cluster's `config.ini` includes the following:

```
[ndbd default]
DataDir= /data0
```

Our objective is to place all Disk Data log files in `/data1`, and all Disk Data data files in `/data2`, on each data node host.

Note

In this example, we assume that the cluster's data node hosts are all using Linux operating systems. For other platforms, you may need to substitute you operating system's commands for those shown here.

To accomplish this, perform the following steps:

- Under the data node file system create symbolic links pointing to the other drives:

```
shell> cd /data0/ndb_2_fs
shell> ls
D1  D10  D11  D2  D8  D9  LCP
shell> ln -s /data0 dnlogs
shell> ln -s /data1 dndata
```

  You should now have two symbolic links:

```
shell> ls -l --hide=D*
lrwxrwxrwx 1 user group   30 2007-03-19 13:58 dndata -> /data1
lrwxrwxrwx 1 user group   30 2007-03-19 13:59 dnlogs -> /data2
```

  We show this only for the data node with node ID 2; however, you must do this for *each* data node.

- Now, in the `mysql` client, create a log file group and tablespace using the symbolic links, as shown here:

```
mysql> CREATE LOGFILE GROUP lg1
    ->    ADD UNDOFILE 'dnlogs/undo1.log'
    ->    INITIAL_SIZE 150M
    ->    UNDO_BUFFER_SIZE = 1M
    ->    ENGINE=NDBCLUSTER;

mysql> CREATE TABLESPACE ts1
    ->    ADD DATAFILE 'dndata/data1.log'
    ->    USE LOGFILE GROUP lg1
    ->    INITIAL_SIZE 1G
    ->    ENGINE=NDBCLUSTER;
```

  Verify that the files were created and placed correctly as shown here:

```
shell> cd /data1
```

```
shell> ls -l
total 2099304
-rw-rw-r--  1 user group 157286400 2007-03-19 14:02 undo1.dat

shell> cd /data2
shell> ls -l
total 2099304
-rw-rw-r--  1 user group 1073741824 2007-03-19 14:02 data1.dat
```

- If you are running multiple data nodes on one host, you must take care to avoid having them try to use the same space for Disk Data files. You can make this easier by creating a symbolic link in each data node file system. Suppose you are using `/data0` for both data node file systems, but you wish to have the Disk Data files for both nodes on `/data1`. In this case, you can do something similar to what is shown here:

```
shell> cd /data0
shell> ln -s /data1/dn2 ndb_2_fs/dd
shell> ln -s /data1/dn3 ndb_3_fs/dd
shell> ls -l --hide=D* ndb_2_fs
lrwxrwxrwx 1 user group   30 2007-03-19 14:22 dd -> /data1/dn2
shell> ls -l --hide=D* ndb_3_fs
lrwxrwxrwx 1 user group   30 2007-03-19 14:22 dd -> /data1/dn3
```

- Now you can create a logfile group and tablespace using the symbolic link, like this:

```
mysql> CREATE LOGFILE GROUP lg1
    ->    ADD UNDOFILE 'dd/undo1.log'
    ->    INITIAL_SIZE 150M
    ->    UNDO_BUFFER_SIZE = 1M
    ->    ENGINE=NDBCLUSTER;

mysql> CREATE TABLESPACE ts1
    ->    ADD DATAFILE 'dd/data1.log'
    ->    USE LOGFILE GROUP lg1
    ->    INITIAL_SIZE 1G
    ->    ENGINE=NDBCLUSTER;
```

Verify that the files were created and placed correctly as shown here:

```
shell> cd /data1
shell> ls
dn2        dn3
shell> ls dn2
undo1.log        data1.log
shell> ls dn3
undo1.log        data1.log
```

22.5.13.3 NDB Cluster Disk Data Storage Requirements

The following items apply to Disk Data storage requirements:

- Variable-length columns of Disk Data tables take up a fixed amount of space. For each row, this is equal to the space required to store the largest possible value for that column.

  For general information about calculating these values, see Section 11.8, “Data Type Storage Requirements”.

  You can obtain an estimate the amount of space available in data files and undo log files by querying the `INFORMATION_SCHEMA.FILES` table. For more information and examples, see Section 25.11, “The INFORMATION_SCHEMA FILES Table”.

Note

The `OPTIMIZE TABLE` statement does not have any effect on Disk Data tables.

- In a Disk Data table, the first 256 bytes of a `TEXT` or `BLOB` column are stored in memory; only the remainder is stored on disk.
- Each row in a Disk Data table uses 8 bytes in memory to point to the data stored on disk. This means that, in some cases, converting an in-memory column to the disk-based format can actually result in greater memory usage. For example, converting a `CHAR(4)` column from memory-based to disk-based format increases the amount of `DataMemory` used per row from 4 to 8 bytes.

Important

Starting the cluster with the `--initial` option does *not* remove Disk Data files. You must remove these manually prior to performing an initial restart of the cluster.

Performance of Disk Data tables can be improved by minimizing the number of disk seeks by making sure that `DiskPageBufferMemory` is of sufficient size. You can query the `diskpagebuffer` table to help determine whether the value for this parameter needs to be increased.

22.5.14 Online Operations with ALTER TABLE in NDB Cluster

MySQL NDB Cluster 8.0 supports online table schema changes using the standard `ALTER TABLE` syntax employed by the MySQL Server (`ALGORITHM=DEFAULT|INPLACE|COPY`), and described elsewhere.

Note

Some older releases of NDB Cluster used a syntax specific to `NDB` for online `ALTER TABLE` operations. That syntax has since been removed.

Operations that add and drop indexes on variable-width columns of `NDB` tables occur online. Online operations are noncopying; that is, they do not require that indexes be re-created. They do not lock the table being altered from access by other API nodes in an NDB Cluster (but see Limitations of NDB online operations, later in this section). Such operations do not require single user mode for `NDB` table alterations made in an NDB cluster with multiple API nodes; transactions can continue uninterrupted during online DDL operations.

`ALGORITHM=INPLACE` can be used to perform online `ADD COLUMN`, `ADD INDEX` (including `CREATE INDEX` statements), and `DROP INDEX` operations on `NDB` tables. Online renaming of `NDB` tables is also supported.

Previously, columns of `NDB` tables could not be renamed online; this limitation is removed in NDB 8.0.18.

Currently you cannot add disk-based columns to `NDB` tables online. This means that, if you wish to add an in-memory column to an `NDB` table that uses a table-level `STORAGE DISK` option, you must declare the new column as using memory-based storage explicitly. For example—assuming that you have already created tablespace `ts1`—suppose that you create table `t1` as follows:

```
mysql> CREATE TABLE t1 (
     >     c1 INT NOT NULL PRIMARY KEY,
     >     c2 VARCHAR(30)
     >     )
     >     TABLESPACE ts1 STORAGE DISK
     >     ENGINE NDB;
Query OK, 0 rows affected (1.73 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can add a new in-memory column to this table online as shown here:

```
mysql> ALTER TABLE t1
     >     ADD COLUMN c3 INT COLUMN_FORMAT DYNAMIC STORAGE MEMORY,
     >     ALGORITHM=INPLACE;
Query OK, 0 rows affected (1.25 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

This statement fails if the `STORAGE MEMORY` option is omitted:

```
mysql> ALTER TABLE t1
     >     ADD COLUMN c4 INT COLUMN_FORMAT DYNAMIC,
     >     ALGORITHM=INPLACE;
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason:
Adding column(s) or add/reorganize partition not supported online. Try
ALGORITHM=COPY.
```

If you omit the `COLUMN_FORMAT DYNAMIC` option, the dynamic column format is employed automatically, but a warning is issued, as shown here:

```
mysql> ALTER ONLINE TABLE t1 ADD COLUMN c4 INT STORAGE MEMORY;
Query OK, 0 rows affected, 1 warning (1.17 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1478
Message: DYNAMIC column c4 with STORAGE DISK is not supported, column will
become FIXED


mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) NOT NULL,
  `c2` varchar(30) DEFAULT NULL,
  `c3` int(11) /*!50606 STORAGE MEMORY */ /*!50606 COLUMN_FORMAT DYNAMIC */ DEFAULT NULL,
  `c4` int(11) /*!50606 STORAGE MEMORY */ DEFAULT NULL,
  PRIMARY KEY (`c1`)
) /*!50606 TABLESPACE ts_1 STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.03 sec)
```

Note

The `STORAGE` and `COLUMN_FORMAT` keywords are supported only in NDB Cluster; in any other version of MySQL, attempting to use either of these keywords in a `CREATE TABLE` or `ALTER TABLE` statement results in an error.

It is also possible to use the statement `ALTER TABLE ... REORGANIZE PARTITION, ALGORITHM=INPLACE` with no *`partition_names`* `INTO (`*`partition_definitions`*`)` option on `NDB` tables. This can be used to redistribute NDB Cluster data among new data nodes that have been added to the cluster online. This does *not* perform any defragmentation, which requires an `OPTIMIZE TABLE` or null `ALTER TABLE` statement. For more information, see Section 22.5.15, “Adding NDB Cluster Data Nodes Online”.

Limitations of NDB online operations

Online `DROP COLUMN` operations are not supported.

Online `ALTER TABLE`, `CREATE INDEX`, or `DROP INDEX` statements that add columns or add or drop indexes are subject to the following limitations:

- A given online `ALTER TABLE` can use only one of `ADD COLUMN`, `ADD INDEX`, or `DROP INDEX`. One or more columns can be added online in a single statement; only one index may be created or dropped online in a single statement.
- The table being altered is not locked with respect to API nodes other than the one on which an online `ALTER TABLE ADD COLUMN`, `ADD INDEX`, or `DROP INDEX` operation (or `CREATE INDEX` or `DROP INDEX` statement) is run. However, the table is locked against any other operations originating on the *same* API node while the online operation is being executed.
- The table to be altered must have an explicit primary key; the hidden primary key created by the `NDB` storage engine is not sufficient for this purpose.
- The storage engine used by the table cannot be changed online.
- When used with NDB Cluster Disk Data tables, it is not possible to change the storage type (`DISK` or `MEMORY`) of a column online. This means, that when you add or drop an index in such a way that the operation would be performed online, and you want the storage type of the column or columns to be changed, you must use `ALGORITHM=COPY` in the statement that adds or drops the index.

Columns to be added online cannot use the `BLOB` or `TEXT` type, and must meet the following criteria:

- The columns must be dynamic; that is, it must be possible to create them using `COLUMN_FORMAT DYNAMIC`. If you omit the `COLUMN_FORMAT DYNAMIC` option, the dynamic column format is employed automatically.
- The columns must permit `NULL` values and not have any explicit default value other than `NULL`. Columns added online are automatically created as `DEFAULT NULL`, as can be seen here:

```
mysql> CREATE TABLE t2 (
     >     c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY
     >     ) ENGINE=NDB;
Query OK, 0 rows affected (1.44 sec)

mysql> ALTER TABLE t2
     >     ADD COLUMN c2 INT,
     >     ADD COLUMN c3 INT,
     >     ALGORITHM=INPLACE;
Query OK, 0 rows affected, 2 warnings (0.93 sec)

mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t2` (
  `c1` int(11) NOT NULL AUTO_INCREMENT,
  `c2` int(11) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

- The columns must be added following any existing columns. If you attempt to add a column online before any existing columns or using the `FIRST` keyword, the statement fails with an error.
- Existing table columns cannot be reordered online.

For online `ALTER TABLE` operations on `NDB` tables, fixed-format columns are converted to dynamic when they are added online, or when indexes are created or dropped online, as shown here (repeating the `CREATE TABLE` and `ALTER TABLE` statements just shown for the sake of clarity):

```
mysql> CREATE TABLE t2 (
    >     c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY
    >     ) ENGINE=NDB;
Query OK, 0 rows affected (1.44 sec)

mysql> ALTER TABLE t2
    >     ADD COLUMN c2 INT,
    >     ADD COLUMN c3 INT,
    >     ALGORITHM=INPLACE;
Query OK, 0 rows affected, 2 warnings (0.93 sec)

mysql> SHOW WARNINGS;
*************************** 1. row ***************************
  Level: Warning
   Code: 1478
Message: Converted FIXED field 'c2' to DYNAMIC to enable online ADD COLUMN
*************************** 2. row ***************************
  Level: Warning
   Code: 1478
Message: Converted FIXED field 'c3' to DYNAMIC to enable online ADD COLUMN
2 rows in set (0.00 sec)
```

Only the column or columns to be added online must be dynamic. Existing columns need not be; this includes the table's primary key, which may also be `FIXED`, as shown here:

```
mysql> CREATE TABLE t3 (
    >     c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY COLUMN_FORMAT FIXED
    >     ) ENGINE=NDB;
Query OK, 0 rows affected (2.10 sec)

mysql> ALTER TABLE t3 ADD COLUMN c2 INT, ALGORITHM=INPLACE;
Query OK, 0 rows affected, 1 warning (0.78 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW WARNINGS;
*************************** 1. row ***************************
  Level: Warning
   Code: 1478
Message: Converted FIXED field 'c2' to DYNAMIC to enable online ADD COLUMN
1 row in set (0.00 sec)
```

Columns are not converted from `FIXED` to `DYNAMIC` column format by renaming operations. For more information about `COLUMN_FORMAT`, see Section 13.1.20, “CREATE TABLE Syntax”.

The `KEY`, `CONSTRAINT`, and `IGNORE` keywords are supported in `ALTER TABLE` statements using `ALGORITHM=INPLACE`.

Setting `MAX_ROWS` to 0 using an online `ALTER TABLE` statement is disallowed. You must use a copying `ALTER TABLE` to perform this operation. (Bug #21960004)

22.5.15 Adding NDB Cluster Data Nodes Online

This section describes how to add NDB Cluster data nodes “online”—that is, without needing to shut down the cluster completely and restart it as part of the process.

Important

Currently, you must add new data nodes to an NDB Cluster as part of a new node group. In addition, it is not possible to change the number of replicas (or the number of nodes per node group) online.

22.5.15.1 Adding NDB Cluster Data Nodes Online: General Issues

This section provides general information about the behavior of and current limitations in adding NDB Cluster nodes online.

Redistribution of Data. The ability to add new nodes online includes a means to reorganize `NDBCLUSTER` table data and indexes so that they are distributed across all data nodes, including the new ones, by means of the `ALTER TABLE ... REORGANIZE PARTITION` statement. Table reorganization of both in-memory and Disk Data tables is supported. This redistribution does not currently include unique indexes (only ordered indexes are redistributed).

The redistribution for `NDBCLUSTER` tables already existing before the new data nodes were added is not automatic, but can be accomplished using simple SQL statements in `mysql` or another MySQL client application. However, all data and indexes added to tables created after a new node group has been added are distributed automatically among all cluster data nodes, including those added as part of the new node group.

Partial starts. It is possible to add a new node group without all of the new data nodes being started. It is also possible to add a new node group to a degraded cluster—that is, a cluster that is only partially started, or where one or more data nodes are not running. In the latter case, the cluster must have enough nodes running to be viable before the new node group can be added.

Effects on ongoing operations. Normal DML operations using NDB Cluster data are not prevented by the creation or addition of a new node group, or by table reorganization. However, it is not possible to perform DDL concurrently with table reorganization—that is, no other DDL statements can be issued while an `ALTER TABLE ... REORGANIZE PARTITION` statement is executing. In addition, during the execution of `ALTER TABLE ... REORGANIZE PARTITION` (or the execution of any other DDL statement), it is not possible to restart cluster data nodes.

Failure handling. Failures of data nodes during node group creation and table reorganization are handled as shown in the following table:

Table 22.413 Data node failure handling during node group creation and table reorganization

Failure during	Failure in “Old” data node	Failure in “New” data node	System Failure

Node group creation	• **If a node other than the master fails:** The creation of the node group is always rolled forward.
	• **If the master fails:**
	  • **If the internal commit point has been reached:** The creation of the node group is rolled forward.
	  • **If the internal commit point has not yet been reached.** The
		• **If a node other than the master fails:** The creation of the node group is always rolled forward.
		• **If the master fails:**
		  • **If the internal commit point has been reached:** The creation of the node group is rolled forward.
		  • **If the internal commit point has not yet been reached.** The
			• **If the execution of CREATE NODEGROUP has reached the internal commit point:** When restarted, the cluster includes the new node group. Otherwise it without.
			• **If the execution of CREATE NODEGROUP has not yet reached the internal commit point:** When restarted, the cluster does not include the new node group.

Failure during	Failure in “Old” data node	Failure in “New” data node	System Failure
	creation of the node group is rolled back	creation of the node group is rolled back	

Table reorganization	• **If a node other than the master fails:** The table reorganization is always rolled forward.
	• **If the master fails:**
	  • **If the internal commit point has been reached:** The table reorganization is rolled forward.
	  • **If the internal commit point has not yet been reached.** The table reorganization is rolled back.
		• **If a node other than the master fails:** The table reorganization is always rolled forward.
		• **If the master fails:**
		  • **If the internal commit point has been reached:** The table reorganization is rolled forward.
		  • **If the internal commit point has not yet been reached.** The table reorganization is rolled back.
			• **If the execution of an ALTER TABLE ... REORGANIZE PARTITION statement has reached the internal commit point:** When the cluster is restarted, the data and indexes belonging to *`table`* are distributed using the “new” data nodes.
			• **If the execution of an ALTER TABLE ... REORGANIZE PARTITION statement has not yet reached the internal commit point:** When the cluster is restarted, the data and indexes belonging to *`table`* are distributed using only the “old” data nodes.

Dropping node groups. The `ndb_mgm` client supports a `DROP NODEGROUP` command, but it is possible to drop a node group only when no data nodes in the node group contain any data. Since there is currently no way to “empty” a specific data node or node group, this command works only the following two cases:

1. After issuing `CREATE NODEGROUP` in the `ndb_mgm` client, but before issuing any `ALTER TABLE ... REORGANIZE PARTITION` statements in the `mysql` client.
2. After dropping all `NDBCLUSTER` tables using `DROP TABLE`.

   `TRUNCATE TABLE` does not work for this purpose because the data nodes continue to store the table definitions.

22.5.15.2 Adding NDB Cluster Data Nodes Online: Basic procedure

In this section, we list the basic steps required to add new data nodes to an NDB Cluster. This procedure applies whether you are using `ndbd` or `ndbmtd` binaries for the data node processes. For a more detailed example, see Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”.

Assuming that you already have a running NDB Cluster, adding data nodes online requires the following steps:

1. Edit the cluster configuration `config.ini` file, adding new `[ndbd]` sections corresponding to the nodes to be added. In the case where the cluster uses multiple management servers, these changes need to be made to all `config.ini` files used by the management servers.

You must be careful that node IDs for any new data nodes added in the `config.ini` file do not overlap node IDs used by existing nodes. In the event that you have API nodes using dynamically allocated node IDs and these IDs match node IDs that you want to use for new data nodes, it is possible to force any such API nodes to “migrate”, as described later in this procedure.

2. Perform a rolling restart of all NDB Cluster management servers.

   Important

   All management servers must be restarted with the `--reload` or `--initial` option to force the reading of the new configuration.

3. Perform a rolling restart of all existing NDB Cluster data nodes. It is not necessary (or usually even desirable) to use `--initial` when restarting the existing data nodes.

   If you are using API nodes with dynamically allocated IDs matching any node IDs that you wish to assign to new data nodes, you must restart all API nodes (including SQL nodes) before restarting any of the data nodes processes in this step. This causes any API nodes with node IDs that were previously not explicitly assigned to relinquish those node IDs and acquire new ones.

4. Perform a rolling restart of any SQL or API nodes connected to the NDB Cluster.

5. Start the new data nodes.

   The new data nodes may be started in any order. They can also be started concurrently, as long as they are started after the rolling restarts of all existing data nodes have been completed, and before proceeding to the next step.

6. Execute one or more `CREATE NODEGROUP` commands in the NDB Cluster management client to create the new node group or node groups to which the new data nodes will belong.

7. Redistribute the cluster's data among all data nodes, including the new ones. Normally this is done by issuing an `ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION` statement in the `mysql` client for each `NDBCLUSTER` table.

   Exception: For tables created using the `MAX_ROWS` option, this statement does not work; instead, use `ALTER TABLE ... ALGORITHM=INPLACE MAX_ROWS=...` to reorganize such tables. You should also bear in mind that using `MAX_ROWS` to set the number of partitions in this fashion is deprecated, and you should use `PARTITION_BALANCE` instead; see Section 13.1.20.11, “Setting NDB_TABLE Options”, for more information.

   Note

   This needs to be done only for tables already existing at the time the new node group is added. Data in tables created after the new node group is added is distributed automatically; however, data added to any given table `tbl` that existed before the new nodes were added is not distributed using the new nodes until that table has been reorganized.

8. `ALTER TABLE ... REORGANIZE PARTITION ALGORITHM=INPLACE` reorganizes partitions but does not reclaim the space freed on the “old” nodes. You can do this by issuing, for each `NDBCLUSTER` table, an `OPTIMIZE TABLE` statement in the `mysql` client.

   This works for space used by variable-width columns of in-memory `NDB` tables. `OPTIMIZE TABLE` is not supported for fixed-width columns of in-memory tables; it is also not supported for Disk Data tables.

You can add all the nodes desired, then issue several `CREATE NODEGROUP` commands in succession to add the new node groups to the cluster.

22.5.15.3 Adding NDB Cluster Data Nodes Online: Detailed Example

In this section we provide a detailed example illustrating how to add new NDB Cluster data nodes online, starting with an NDB Cluster having 2 data nodes in a single node group and concluding with a cluster having 4 data nodes in 2 node groups.

Starting configuration. For purposes of illustration, we assume a minimal configuration, and that the cluster uses a `config.ini` file containing only the following information:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 198.51.100.1

[ndbd]
Id = 2
HostName = 198.51.100.2

[mgm]
HostName = 198.51.100.10
Id = 10

[api]
Id=20
HostName = 198.51.100.20

[api]
Id=21
HostName = 198.51.100.21
```

Note

We have left a gap in the sequence between data node IDs and other nodes. This make it easier later to assign node IDs that are not already in use to data nodes which are newly added.

We also assume that you have already started the cluster using the appropriate command line or `my.cnf` options, and that running `SHOW` in the management client produces output similar to what is shown here:

```
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (8.0.19-ndb-8.0.19, Nodegroup: 0, *)
id=2    @198.51.100.2  (8.0.19-ndb-8.0.19, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (8.0.19-ndb-8.0.19)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (8.0.19-ndb-8.0.19)
```

```
id=21   @198.51.100.21  (8.0.19-ndb-8.0.19)
```

Finally, we assume that the cluster contains a single `NDBCLUSTER` table created as shown here:

```
USE n;

CREATE TABLE ips (
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    country_code CHAR(2) NOT NULL,
    type CHAR(4) NOT NULL,
    ip_address VARCHAR(15) NOT NULL,
    addresses BIGINT UNSIGNED DEFAULT NULL,
    date BIGINT UNSIGNED DEFAULT NULL
)   ENGINE NDBCLUSTER;
```

The memory usage and related information shown later in this section was generated after inserting approximately 50000 rows into this table.

Note

In this example, we show the single-threaded `ndbd` being used for the data node processes. You can also apply this example, if you are using the multithreaded `ndbmtd` by substituting `ndbmtd` for `ndbd` wherever it appears in the steps that follow.

Step 1: Update configuration file. Open the cluster global configuration file in a text editor and add `[ndbd]` sections corresponding to the 2 new data nodes. (We give these data nodes IDs 3 and 4, and assume that they are to be run on host machines at addresses 198.51.100.3 and 198.51.100.4, respectively.) After you have added the new sections, the contents of the `config.ini` file should look like what is shown here, where the additions to the file are shown in bold type:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 198.51.100.1

[ndbd]
Id = 2
HostName = 198.51.100.2

[ndbd]
Id = 3
HostName = 198.51.100.3

[ndbd]
Id = 4
HostName = 198.51.100.4

[mgm]
HostName = 198.51.100.10
Id = 10

[api]
Id=20
HostName = 198.51.100.20

[api]
```

```
Id=21
HostName = 198.51.100.21
```

Once you have made the necessary changes, save the file.

Step 2: Restart the management server. Restarting the cluster management server requires that you issue separate commands to stop the management server and then to start it again, as follows:

1. Stop the management server using the management client `STOP` command, as shown here:

   ```
   ndb_mgm> 10 STOP
   Node 10 has shut down.
   Disconnecting to allow Management Server to shutdown

   shell>
   ```

2. Because shutting down the management server causes the management client to terminate, you must start the management server from the system shell. For simplicity, we assume that `config.ini` is in the same directory as the management server binary, but in practice, you must supply the correct path to the configuration file. You must also supply the `--reload` or `--initial` option so that the management server reads the new configuration from the file rather than its configuration cache. If your shell's current directory is also the same as the directory where the management server binary is located, then you can invoke the management server as shown here:

   ```
   shell> ndb_mgmd -f config.ini --reload
   2008-12-08 17:29:23 [MgmSrvr] INFO     -- NDB Cluster Management Server. 8.0.19-ndb-8.0.19
   2008-12-08 17:29:23 [MgmSrvr] INFO     -- Reading cluster configuration from 'config.ini'
   ```

If you check the output of `SHOW` in the management client after restarting the `ndb_mgm` process, you should now see something like this:

```
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (8.0.19-ndb-8.0.19, Nodegroup: 0, *)
id=2    @198.51.100.2  (8.0.19-ndb-8.0.19, Nodegroup: 0)
id=3 (not connected, accepting connect from 198.51.100.3)
id=4 (not connected, accepting connect from 198.51.100.4)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (8.0.19-ndb-8.0.19)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (8.0.19-ndb-8.0.19)
id=21   @198.51.100.21  (8.0.19-ndb-8.0.19)
```

Step 3: Perform a rolling restart of the existing data nodes. This step can be accomplished entirely within the cluster management client using the `RESTART` command, as shown here:

```
ndb_mgm> 1 RESTART
Node 1: Node shutdown initiated
Node 1: Node shutdown completed, restarting, no start.
Node 1 is being restarted

ndb_mgm> Node 1: Start initiated (version 8.0.19)
Node 1: Started (version 8.0.19)
```

```
ndb_mgm> 2 RESTART
Node 2: Node shutdown initiated
Node 2: Node shutdown completed, restarting, no start.
Node 2 is being restarted

ndb_mgm> Node 2: Start initiated (version 8.0.19)

ndb_mgm> Node 2: Started (version 8.0.19)
```

Important

After issuing each `X RESTART` command, wait until the management client reports `Node X: Started (version ...)` *before* proceeding any further.

You can verify that all existing data nodes were restarted using the updated configuration by checking the `ndbinfo.nodes` table in the `mysql` client.

Step 4: Perform a rolling restart of all cluster API nodes. Shut down and restart each MySQL server acting as an SQL node in the cluster using `mysqladmin shutdown` followed by `mysqld_safe` (or another startup script). This should be similar to what is shown here, where *`password`* is the MySQL `root` password for a given MySQL server instance:

```
shell> mysqladmin -uroot -ppassword shutdown
081208 20:19:56 mysqld_safe mysqld from pid file
/usr/local/mysql/var/tonfisk.pid ended
shell> mysqld_safe --ndbcluster --ndb-connectstring=198.51.100.10 &
081208 20:20:06 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
081208 20:20:06 mysqld_safe Starting mysqld daemon with databases
from /usr/local/mysql/var
```

Of course, the exact input and output depend on how and where MySQL is installed on the system, as well as which options you choose to start it (and whether or not some or all of these options are specified in a `my.cnf` file).

Step 5: Perform an initial start of the new data nodes. From a system shell on each of the hosts for the new data nodes, start the data nodes as shown here, using the `--initial` option:

```
shell> ndbd -c 198.51.100.10 --initial
```

Note

Unlike the case with restarting the existing data nodes, you can start the new data nodes concurrently; you do not need to wait for one to finish starting before starting the other.

Wait until both of the new data nodes have started before proceeding with the next step. Once the new data nodes have started, you can see in the output of the management client `SHOW` command that they do not yet belong to any node group (as indicated with bold type here):

```
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (8.0.19-ndb-8.0.19, Nodegroup: 0, *)
id=2    @198.51.100.2  (8.0.19-ndb-8.0.19, Nodegroup: 0)
id=3    @198.51.100.3  (8.0.19-ndb-8.0.19, no nodegroup)
```

```
id=4    @198.51.100.4  (8.0.19-ndb-8.0.19, no nodegroup)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (8.0.19-ndb-8.0.19)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (8.0.19-ndb-8.0.19)
id=21   @198.51.100.21  (8.0.19-ndb-8.0.19)
```

Step 6: Create a new node group. You can do this by issuing a `CREATE NODEGROUP` command in the cluster management client. This command takes as its argument a comma-separated list of the node IDs of the data nodes to be included in the new node group, as shown here:

```
ndb_mgm> CREATE NODEGROUP 3,4
Nodegroup 1 created
```

By issuing `SHOW` again, you can verify that data nodes 3 and 4 have joined the new node group (again indicated in bold type):

```
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (8.0.19-ndb-8.0.19, Nodegroup: 0, *)
id=2    @198.51.100.2  (8.0.19-ndb-8.0.19, Nodegroup: 0)
id=3    @198.51.100.3  (8.0.19-ndb-8.0.19, Nodegroup: 1)
id=4    @198.51.100.4  (8.0.19-ndb-8.0.19, Nodegroup: 1)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (8.0.19-ndb-8.0.19)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (8.0.19-ndb-8.0.19)
id=21   @198.51.100.21  (8.0.19-ndb-8.0.19)
```

Step 7: Redistribute cluster data. When a node group is created, existing data and indexes are not automatically distributed to the new node group's data nodes, as you can see by issuing the appropriate `REPORT` command in the management client:

```
ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(177 32K pages of total 3200)
Node 1: Index usage is 0%(108 8K pages of total 12832)
Node 2: Data usage is 5%(177 32K pages of total 3200)
Node 2: Index usage is 0%(108 8K pages of total 12832)
Node 3: Data usage is 0%(0 32K pages of total 3200)
Node 3: Index usage is 0%(0 8K pages of total 12832)
Node 4: Data usage is 0%(0 32K pages of total 3200)
Node 4: Index usage is 0%(0 8K pages of total 12832)
```

By using `ndb_desc` with the `-p` option, which causes the output to include partitioning information, you can see that the table still uses only 2 partitions (in the `Per partition info` section of the output, shown here in bold text):

```
shell> ndb_desc -c 198.51.100.10 -d n ips -p
-- ips --
Version: 1
Fragment type: 9
K Value: 6
```

```
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 6
Number of primary keys: 1
Length of frm data: 340
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 2
TableStatus: Retrieved
-- Attributes --
id Bigint PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
country_code Char(2;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
type Char(4;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
ip_address Varchar(15;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
addresses Bigunsigned NULL AT=FIXED ST=MEMORY
date Bigunsigned NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

-- Per partition info --
Partition   Row count   Commit count  Frag fixed memory   Frag varsized memory
0           26086       26086         1572864             557056
1           26329       26329         1605632             557056

NDBT_ProgramExit: 0 - OK
```

You can cause the data to be redistributed among all of the data nodes by performing, for each `NDB` table, an `ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION` statement in the `mysql` client.

Important

`ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION` does not work on tables that were created with the `MAX_ROWS` option. Instead, use `ALTER TABLE ... ALGORITHM=INPLACE, MAX_ROWS=...` to reorganize such tables.

Keep in mind that using `MAX_ROWS` to set the number of partitions per table is deprecated, and you should use `PARTITION_BALANCE` instead; see Section 13.1.20.11, “Setting NDB_TABLE Options”, for more information.

After issuing the statement `ALTER TABLE ips ALGORITHM=INPLACE, REORGANIZE PARTITION`, you can see using `ndb_desc` that the data for this table is now stored using 4 partitions, as shown here (with the relevant portions of the output in bold type):

```
shell> ndb_desc -c 198.51.100.10 -d n ips -p
-- ips --
Version: 16777217
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 6
Number of primary keys: 1
Length of frm data: 341
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
```

```
ForceVarPart: 1
FragmentCount: 4
TableStatus: Retrieved
-- Attributes --
id Bigint PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
country_code Char(2;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
type Char(4;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
ip_address Varchar(15;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
addresses Bigunsigned NULL AT=FIXED ST=MEMORY
date Bigunsigned NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

-- Per partition info --
Partition   Row count   Commit count  Frag fixed memory   Frag varsized memory
0           12981       52296         1572864             557056
1           13236       52515         1605632             557056
2           13105       13105         819200              294912
3           13093       13093         819200              294912

NDBT_ProgramExit: 0 - OK
```

Note

Normally, `ALTER TABLE table_name [ALGORITHM=INPLACE,] REORGANIZE PARTITION` is used with a list of partition identifiers and a set of partition definitions to create a new partitioning scheme for a table that has already been explicitly partitioned. Its use here to redistribute data onto a new NDB Cluster node group is an exception in this regard; when used in this way, no other keywords or identifiers follow `REORGANIZE PARTITION`.

For more information, see Section 13.1.9, “ALTER TABLE Syntax”.

In addition, for each table, the `ALTER TABLE` statement should be followed by an `OPTIMIZE TABLE` to reclaim wasted space. You can obtain a list of all `NDBCLUSTER` tables using the following query against the `INFORMATION_SCHEMA.TABLES` table:

```
SELECT TABLE_SCHEMA, TABLE_NAME
    FROM INFORMATION_SCHEMA.TABLES
    WHERE ENGINE = 'NDBCLUSTER';
```

Note

The `INFORMATION_SCHEMA.TABLES.ENGINE` value for an NDB Cluster table is always `NDBCLUSTER`, regardless of whether the `CREATE TABLE` statement used to create the table (or `ALTER TABLE` statement used to convert an existing table from a different storage engine) used `NDB` or `NDBCLUSTER` in its `ENGINE` option.

You can see after performing these statements in the output of `ALL REPORT MEMORY` that the data and indexes are now redistributed between all cluster data nodes, as shown here:

```
ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(176 32K pages of total 3200)
Node 1: Index usage is 0%(76 8K pages of total 12832)
Node 2: Data usage is 5%(176 32K pages of total 3200)
Node 2: Index usage is 0%(76 8K pages of total 12832)
Node 3: Data usage is 2%(80 32K pages of total 3200)
```

```
Node 3: Index usage is 0%(51 8K pages of total 12832)
Node 4: Data usage is 2%(80 32K pages of total 3200)
Node 4: Index usage is 0%(50 8K pages of total 12832)
```

Note

Since only one DDL operation on `NDBCLUSTER` tables can be executed at a time, you must wait for each `ALTER TABLE ... REORGANIZE PARTITION` statement to finish before issuing the next one.

It is not necessary to issue `ALTER TABLE ... REORGANIZE PARTITION` statements for `NDBCLUSTER` tables created *after* the new data nodes have been added; data added to such tables is distributed among all data nodes automatically. However, in `NDBCLUSTER` tables that existed *prior to* the addition of the new nodes, neither existing nor new data is distributed using the new nodes until these tables have been reorganized using `ALTER TABLE ... REORGANIZE PARTITION`.

Alternative procedure, without rolling restart. It is possible to avoid the need for a rolling restart by configuring the extra data nodes, but not starting them, when first starting the cluster. We assume, as before, that you wish to start with two data nodes—nodes 1 and 2—in one node group and later to expand the cluster to four data nodes, by adding a second node group consisting of nodes 3 and 4:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 198.51.100.1

[ndbd]
Id = 2
HostName = 198.51.100.2

[ndbd]
Id = 3
HostName = 198.51.100.3
Nodegroup = 65536

[ndbd]
Id = 4
HostName = 198.51.100.4
Nodegroup = 65536

[mgm]
HostName = 198.51.100.10
Id = 10

[api]
Id=20
HostName = 198.51.100.20

[api]
Id=21
HostName = 198.51.100.21
```

The data nodes to be brought online at a later time (nodes 3 and 4) can be configured with `NodeGroup = 65536`, in which case nodes 1 and 2 can each be started as shown here:

```
shell> ndbd -c 198.51.100.10 --initial
```

The data nodes configured with `NodeGroup = 65536` are treated by the management server as though you had started nodes 1 and 2 using `--nowait-nodes=3,4` after waiting for a period of time determined by the setting for the `StartNoNodeGroupTimeout` data node configuration parameter. By default, this is 15 seconds (15000 milliseconds).

Note

`StartNoNodegroupTimeout` must be the same for all data nodes in the cluster; for this reason, you should always set it in the `[ndbd default]` section of the `config.ini` file, rather than for individual data nodes.

When you are ready to add the second node group, you need only perform the following additional steps:

1. Start data nodes 3 and 4, invoking the data node process once for each new node:

```
shell> ndbd -c 198.51.100.10 --initial
```

2. Issue the appropriate `CREATE NODEGROUP` command in the management client:

```
ndb_mgm> CREATE NODEGROUP 3,4
```

3. In the `mysql` client, issue `ALTER TABLE ... REORGANIZE PARTITION` and `OPTIMIZE TABLE` statements for each existing `NDBCLUSTER` table. (As noted elsewhere in this section, existing NDB Cluster tables cannot use the new nodes for data distribution until this has been done.)

22.5.16 Distributed MySQL Privileges with NDB_STORED_USER

NDB 8.0.18 introduces a new mechanism for sharing and synchronizing users, roles, and privileges between SQL nodes connected to an NDB Cluster. This can be enabled by granting the `NDB_STORED_USER` privilege. See the description of the privilege for usage information.

`NDB_STORED_USER` is printed in the output of `SHOW GRANTS` as with any other privilege. To verify that privileges are shared, use the `ndb_select_all` utility supplied with the NDB Cluster distribution, as shown here (some output wrapped to preserve formatting):

```
shell> ndb_select_all -d mysql ndb_sql_metadata
type    name    seq     note    sql_ddl_text
11      "'jon'@'localhost'"     0       4       "CREATE USER 'jon'@'localhost'
IDENTIFIED WITH 'caching_sha2_password' AS
'$A$005${B];3!?tI\".EFy\ZA5K5DQHrWiBvuRNYTIMeO0YeBlPpZotFRPjVTYzTA5b0' REQUIRE
NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK PASSWORD HISTORY DEFAULT PASSWORD
REUSE INTERVAL DEFAULT PASSWORD REQUIRE CURRENT DEFAULT"
12      "'jon'@'localhost'"     0       [NULL]  "GRANT USAGE ON *.* TO `jon`@`localhost`"
12      "'jon'@'localhost'"     3       [NULL]  "GRANT ALL PRIVILEGES ON `test`.* TO `jon`@`localhost`"
12      "'jon'@'localhost'"     2       [NULL]  "GRANT ALL PRIVILEGES ON `mydb`.* TO `jon`@`localhost`"
12      "'jon'@'localhost'"     1       [NULL]  "GRANT NDB_STORED_USER ON *.* TO `jon`@`localhost`"
5 rows returned
```

`ndb_sql_metadata` is a special `NDB` table that is not visible using the `mysql` or other MySQL client.

A statement granting the `NDB_STORED_USER` privilege, such as `GRANT NDB_STORED_USER ON *.* TO 'cluster_app_user'@'localhost'`, works by directing `NDB` to create a snapshot using the queries `SHOW CREATE USER cluster_app_user@localhost` and `SHOW GRANTS FOR cluster_app_user@localhost`, then storing the results in `ndb_sql_metadata`. Any other SQL nodes are then requested to read and apply the snapshot. Whenever a MySQL server starts up and joins the cluster as an SQL node it executes these stored `CREATE USER` and `GRANT` statements as part of the cluster schema synchronization process.

Whenever an SQL statement is executed on an SQL node other than the one where it originated, the statement is run in a utility thread of the `NDBCLUSTER` storage engine; this is done within a security environment equivalent to that of the MySQL replication slave applier thread.

You should be aware that changes to users with `NDB_STORED_USER` are distributed asynchronously. Because distributed schema change operations are performed synchronously, the next distributed schema change following a change to any distributed user or users serves as a synchronization point. Any pending user changes run to completion before the schema change distribution can begin; after this the schema change itself runs synchronously. For example, if a `DROP DATABASE` statement follows a `DROP USER` of a distributed user, the drop of the database cannot take place until the drop of the user has completed on all SQL nodes.

In the event that multiple `GRANT`, `REVOKE`, or other user administration statements from multiple SQL nodes cause privileges for a given user to diverge on different SQL nodes, you can fix this problem by issuing `GRANT NDB_STORED_USER` for this user on an SQL node where the privileges are known to be correct; this causes a new snapshot of the privileges to be taken and synchronized to the other SQL nodes.

NDB Cluster 8.0 does not support distribution of MySQL users and privileges across SQL nodes in an NDB Cluster by converting the MySQL privilege tables to use the `NDB` storage engine, as implemented in NDB 7.6 and earlier releases (see Distributed Privileges Using Shared Grant Tables). For information about the impact of this change on upgrades to NDB 8.0 from a previous release, see Section 22.2.8, “Upgrading and Downgrading NDB Cluster”.

22.5.17 NDB API Statistics Counters and Variables

A number of types of statistical counters relating to actions performed by or affecting `Ndb` objects are available. Such actions include starting and closing (or aborting) transactions; primary key and unique key operations; table, range, and pruned scans; threads blocked while waiting for the completion of various operations; and data and events sent and received by `NDBCLUSTER`. The counters are incremented inside the NDB kernel whenever NDB API calls are made or data is sent to or received by the data nodes. `mysqld` exposes these counters as system status variables; their values can be read in the output of `SHOW STATUS`, or by querying the `INFORMATION_SCHEMA.SESSION_STATUS` or `INFORMATION_SCHEMA.GLOBAL_STATUS` table. By comparing the values before and after statements operating on `NDB` tables, you can observe the corresponding actions taken on the API level, and thus the cost of performing the statement.

You can list all of these status variables using the following `SHOW STATUS` statement:

```
mysql> SHOW STATUS LIKE 'ndb_api%';
+-----------------------------------------------+----------+
| Variable_name                                 | Value    |
+-----------------------------------------------+----------+
| Ndb_api_wait_exec_complete_count_session      | 0        |
| Ndb_api_wait_scan_result_count_session        | 0        |
| Ndb_api_wait_meta_request_count_session       | 0        |
| Ndb_api_wait_nanos_count_session              | 0        |
| Ndb_api_bytes_sent_count_session              | 0        |
| Ndb_api_bytes_received_count_session          | 0        |
| Ndb_api_trans_start_count_session             | 0        |
| Ndb_api_trans_commit_count_session            | 0        |
| Ndb_api_trans_abort_count_session             | 0        |
| Ndb_api_trans_close_count_session             | 0        |
| Ndb_api_pk_op_count_session                   | 0        |
| Ndb_api_uk_op_count_session                   | 0        |
| Ndb_api_table_scan_count_session              | 0        |
| Ndb_api_range_scan_count_session              | 0        |
| Ndb_api_pruned_scan_count_session             | 0        |
| Ndb_api_scan_batch_count_session              | 0        |
```

```
| Ndb_api_read_row_count_session              | 0        |
| Ndb_api_trans_local_read_row_count_session | 0        |
| Ndb_api_event_data_count_injector           | 0        |
| Ndb_api_event_nondata_count_injector        | 0        |
| Ndb_api_event_bytes_count_injector          | 0        |
| Ndb_api_wait_exec_complete_count_slave      | 0        |
| Ndb_api_wait_scan_result_count_slave        | 0        |
| Ndb_api_wait_meta_request_count_slave       | 0        |
| Ndb_api_wait_nanos_count_slave              | 0        |
| Ndb_api_bytes_sent_count_slave              | 0        |
| Ndb_api_bytes_received_count_slave          | 0        |
| Ndb_api_trans_start_count_slave             | 0        |
| Ndb_api_trans_commit_count_slave            | 0        |
| Ndb_api_trans_abort_count_slave             | 0        |
| Ndb_api_trans_close_count_slave             | 0        |
| Ndb_api_pk_op_count_slave                   | 0        |
| Ndb_api_uk_op_count_slave                   | 0        |
| Ndb_api_table_scan_count_slave              | 0        |
| Ndb_api_range_scan_count_slave              | 0        |
| Ndb_api_pruned_scan_count_slave             | 0        |
| Ndb_api_scan_batch_count_slave              | 0        |
| Ndb_api_read_row_count_slave                | 0        |
| Ndb_api_trans_local_read_row_count_slave    | 0        |
| Ndb_api_wait_exec_complete_count            | 2        |
| Ndb_api_wait_scan_result_count              | 3        |
| Ndb_api_wait_meta_request_count             | 27       |
| Ndb_api_wait_nanos_count                    | 45612023 |
| Ndb_api_bytes_sent_count                    | 992      |
| Ndb_api_bytes_received_count                | 9640     |
| Ndb_api_trans_start_count                   | 2        |
| Ndb_api_trans_commit_count                  | 1        |
| Ndb_api_trans_abort_count                   | 0        |
| Ndb_api_trans_close_count                   | 2        |
| Ndb_api_pk_op_count                         | 1        |
| Ndb_api_uk_op_count                         | 0        |
| Ndb_api_table_scan_count                    | 1        |
| Ndb_api_range_scan_count                    | 0        |
| Ndb_api_pruned_scan_count                   | 0        |
| Ndb_api_scan_batch_count                    | 0        |
| Ndb_api_read_row_count                      | 1        |
| Ndb_api_trans_local_read_row_count          | 1        |
| Ndb_api_event_data_count                    | 0        |
| Ndb_api_event_nondata_count                 | 0        |
| Ndb_api_event_bytes_count                   | 0        |
+---------------------------------------------+----------+
60 rows in set (0.02 sec)
```

These status variables are also available from the `SESSION_STATUS` and `GLOBAL_STATUS` tables of the `INFORMATION_SCHEMA` database, as shown here:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS
    ->   WHERE VARIABLE_NAME LIKE 'ndb_api%';
+---------------------------------------------+----------------+
| VARIABLE_NAME                               | VARIABLE_VALUE |
+---------------------------------------------+----------------+
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SESSION    | 2              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SESSION      | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SESSION     | 1              |
| NDB_API_WAIT_NANOS_COUNT_SESSION            | 8144375        |
| NDB_API_BYTES_SENT_COUNT_SESSION            | 68             |
| NDB_API_BYTES_RECEIVED_COUNT_SESSION        | 84             |
| NDB_API_TRANS_START_COUNT_SESSION           | 1              |
| NDB_API_TRANS_COMMIT_COUNT_SESSION          | 1              |
| NDB_API_TRANS_ABORT_COUNT_SESSION           | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SESSION           | 1              |
| NDB_API_PK_OP_COUNT_SESSION                 | 1              |
```

```
| NDB_API_UK_OP_COUNT_SESSION                | 0              |
| NDB_API_TABLE_SCAN_COUNT_SESSION           | 0              |
| NDB_API_RANGE_SCAN_COUNT_SESSION           | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SESSION          | 0              |
| NDB_API_SCAN_BATCH_COUNT_SESSION           | 0              |
| NDB_API_READ_ROW_COUNT_SESSION             | 1              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SESSION | 1              |
| NDB_API_EVENT_DATA_COUNT_INJECTOR          | 0              |
| NDB_API_EVENT_NONDATA_COUNT_INJECTOR       | 0              |
| NDB_API_EVENT_BYTES_COUNT_INJECTOR         | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SLAVE     | 0              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SLAVE       | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SLAVE      | 0              |
| NDB_API_WAIT_NANOS_COUNT_SLAVE             | 0              |
| NDB_API_BYTES_SENT_COUNT_SLAVE             | 0              |
| NDB_API_BYTES_RECEIVED_COUNT_SLAVE         | 0              |
| NDB_API_TRANS_START_COUNT_SLAVE            | 0              |
| NDB_API_TRANS_COMMIT_COUNT_SLAVE           | 0              |
| NDB_API_TRANS_ABORT_COUNT_SLAVE            | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SLAVE            | 0              |
| NDB_API_PK_OP_COUNT_SLAVE                  | 0              |
| NDB_API_UK_OP_COUNT_SLAVE                  | 0              |
| NDB_API_TABLE_SCAN_COUNT_SLAVE             | 0              |
| NDB_API_RANGE_SCAN_COUNT_SLAVE             | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SLAVE            | 0              |
| NDB_API_SCAN_BATCH_COUNT_SLAVE             | 0              |
| NDB_API_READ_ROW_COUNT_SLAVE               | 0              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SLAVE   | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT           | 4              |
| NDB_API_WAIT_SCAN_RESULT_COUNT             | 3              |
| NDB_API_WAIT_META_REQUEST_COUNT            | 28             |
| NDB_API_WAIT_NANOS_COUNT                   | 53756398       |
| NDB_API_BYTES_SENT_COUNT                   | 1060           |
| NDB_API_BYTES_RECEIVED_COUNT               | 9724           |
| NDB_API_TRANS_START_COUNT                  | 3              |
| NDB_API_TRANS_COMMIT_COUNT                 | 2              |
| NDB_API_TRANS_ABORT_COUNT                  | 0              |
| NDB_API_TRANS_CLOSE_COUNT                  | 3              |
| NDB_API_PK_OP_COUNT                        | 2              |
| NDB_API_UK_OP_COUNT                        | 0              |
| NDB_API_TABLE_SCAN_COUNT                   | 1              |
| NDB_API_RANGE_SCAN_COUNT                   | 0              |
| NDB_API_PRUNED_SCAN_COUNT                  | 0              |
| NDB_API_SCAN_BATCH_COUNT                   | 0              |
| NDB_API_READ_ROW_COUNT                     | 2              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT         | 2              |
| NDB_API_EVENT_DATA_COUNT                   | 0              |
| NDB_API_EVENT_NONDATA_COUNT                | 0              |
| NDB_API_EVENT_BYTES_COUNT                  | 0              |
+--------------------------------------------+----------------+
60 rows in set (0.00 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    ->     WHERE VARIABLE_NAME LIKE 'ndb_api%';
+--------------------------------------------+----------------+
| VARIABLE_NAME                              | VARIABLE_VALUE |
+--------------------------------------------+----------------+
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SESSION   | 2              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SESSION     | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SESSION    | 1              |
| NDB_API_WAIT_NANOS_COUNT_SESSION           | 8144375        |
| NDB_API_BYTES_SENT_COUNT_SESSION           | 68             |
| NDB_API_BYTES_RECEIVED_COUNT_SESSION       | 84             |
| NDB_API_TRANS_START_COUNT_SESSION          | 1              |
| NDB_API_TRANS_COMMIT_COUNT_SESSION         | 1              |
| NDB_API_TRANS_ABORT_COUNT_SESSION          | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SESSION          | 1              |
```

```
| NDB_API_PK_OP_COUNT_SESSION                | 1              |
| NDB_API_UK_OP_COUNT_SESSION                | 0              |
| NDB_API_TABLE_SCAN_COUNT_SESSION           | 0              |
| NDB_API_RANGE_SCAN_COUNT_SESSION           | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SESSION          | 0              |
| NDB_API_SCAN_BATCH_COUNT_SESSION           | 0              |
| NDB_API_READ_ROW_COUNT_SESSION             | 1              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SESSION | 1              |
| NDB_API_EVENT_DATA_COUNT_INJECTOR          | 0              |
| NDB_API_EVENT_NONDATA_COUNT_INJECTOR       | 0              |
| NDB_API_EVENT_BYTES_COUNT_INJECTOR         | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SLAVE     | 0              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SLAVE       | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SLAVE      | 0              |
| NDB_API_WAIT_NANOS_COUNT_SLAVE             | 0              |
| NDB_API_BYTES_SENT_COUNT_SLAVE             | 0              |
| NDB_API_BYTES_RECEIVED_COUNT_SLAVE         | 0              |
| NDB_API_TRANS_START_COUNT_SLAVE            | 0              |
| NDB_API_TRANS_COMMIT_COUNT_SLAVE           | 0              |
| NDB_API_TRANS_ABORT_COUNT_SLAVE            | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SLAVE            | 0              |
| NDB_API_PK_OP_COUNT_SLAVE                  | 0              |
| NDB_API_UK_OP_COUNT_SLAVE                  | 0              |
| NDB_API_TABLE_SCAN_COUNT_SLAVE             | 0              |
| NDB_API_RANGE_SCAN_COUNT_SLAVE             | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SLAVE            | 0              |
| NDB_API_SCAN_BATCH_COUNT_SLAVE             | 0              |
| NDB_API_READ_ROW_COUNT_SLAVE               | 0              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SLAVE   | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT           | 4              |
| NDB_API_WAIT_SCAN_RESULT_COUNT             | 3              |
| NDB_API_WAIT_META_REQUEST_COUNT            | 28             |
| NDB_API_WAIT_NANOS_COUNT                   | 53756398       |
| NDB_API_BYTES_SENT_COUNT                   | 1060           |
| NDB_API_BYTES_RECEIVED_COUNT               | 9724           |
| NDB_API_TRANS_START_COUNT                  | 3              |
| NDB_API_TRANS_COMMIT_COUNT                 | 2              |
| NDB_API_TRANS_ABORT_COUNT                  | 0              |
| NDB_API_TRANS_CLOSE_COUNT                  | 3              |
| NDB_API_PK_OP_COUNT                        | 2              |
| NDB_API_UK_OP_COUNT                        | 0              |
| NDB_API_TABLE_SCAN_COUNT                   | 1              |
| NDB_API_RANGE_SCAN_COUNT                   | 0              |
| NDB_API_PRUNED_SCAN_COUNT                  | 0              |
| NDB_API_SCAN_BATCH_COUNT                   | 0              |
| NDB_API_READ_ROW_COUNT                     | 2              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT         | 2              |
| NDB_API_EVENT_DATA_COUNT                   | 0              |
| NDB_API_EVENT_NONDATA_COUNT                | 0              |
| NDB_API_EVENT_BYTES_COUNT                  | 0              |
+--------------------------------------------+----------------+
60 rows in set (0.00 sec)
```

Each `Ndb` object has its own counters. NDB API applications can read the values of the counters for use in optimization or monitoring. For multithreaded clients which use more than one `Ndb` object concurrently, it is also possible to obtain a summed view of counters from all `Ndb` objects belonging to a given `Ndb_cluster_connection`.

Four sets of these counters are exposed. One set applies to the current session only; the other 3 are global. *This is in spite of the fact that their values can be obtained as either session or global status variables in the `mysql` client*. This means that specifying the `SESSION` or `GLOBAL` keyword with `SHOW STATUS` has no effect on the values reported for NDB API statistics status variables, and the value for each of these variables is the same whether the value is obtained from the equivalent column of the `SESSION_STATUS` or the `GLOBAL_STATUS` table.

- *Session counters (session specific)*

  Session counters relate to the `Ndb` objects in use by (only) the current session. Use of such objects by other MySQL clients does not influence these counts.

  In order to minimize confusion with standard MySQL session variables, we refer to the variables that correspond to these NDB API session counters as "`_session` variables", with a leading underscore.

- *Slave counters (global)*

  This set of counters relates to the `Ndb` objects used by the replication slave SQL thread, if any. If this `mysqld` does not act as a replication slave, or does not use `NDB` tables, then all of these counts are 0.

  We refer to the related status variables as "`_slave` variables" (with a leading underscore).

- *Injector counters (global)*

  Injector counters relate to the `Ndb` object used to listen to cluster events by the binary log injector thread. Even when not writing a binary log, `mysqld` processes attached to an NDB Cluster continue to listen for some events, such as schema changes.

  We refer to the status variables that correspond to NDB API injector counters as "`_injector` variables" (with a leading underscore).

- *Server (Global) counters (global)*

  This set of counters relates to all `Ndb` objects currently used by this `mysqld`. This includes all MySQL client applications, the slave SQL thread (if any), the binlog injector, and the `NDB` utility thread.

  We refer to the status variables that correspond to these counters as "global variables" or "`mysqld`-level variables".

You can obtain values for a particular set of variables by additionally filtering for the substring `session`, `slave`, or `injector` in the variable name (along with the common prefix `Ndb_api`). For `_session` variables, this can be done as shown here:

```
mysql> SHOW STATUS LIKE 'ndb_api%session';
+-----------------------------------------------+---------+
| Variable_name                                 | Value   |
+-----------------------------------------------+---------+
| Ndb_api_wait_exec_complete_count_session      | 2       |
| Ndb_api_wait_scan_result_count_session        | 0       |
| Ndb_api_wait_meta_request_count_session       | 1       |
| Ndb_api_wait_nanos_count_session              | 8144375 |
| Ndb_api_bytes_sent_count_session              | 68      |
| Ndb_api_bytes_received_count_session          | 84      |
| Ndb_api_trans_start_count_session             | 1       |
| Ndb_api_trans_commit_count_session            | 1       |
| Ndb_api_trans_abort_count_session             | 0       |
| Ndb_api_trans_close_count_session             | 1       |
| Ndb_api_pk_op_count_session                   | 1       |
| Ndb_api_uk_op_count_session                   | 0       |
| Ndb_api_table_scan_count_session              | 0       |
| Ndb_api_range_scan_count_session              | 0       |
| Ndb_api_pruned_scan_count_session             | 0       |
| Ndb_api_scan_batch_count_session              | 0       |
| Ndb_api_read_row_count_session                | 1       |
| Ndb_api_trans_local_read_row_count_session    | 1       |
+-----------------------------------------------+---------+
```

```
18 rows in set (0.50 sec)
```

To obtain a listing of the NDB API `mysqld`-level status variables, filter for variable names beginning with `ndb_api` and ending in `_count`, like this:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS
    ->     WHERE VARIABLE_NAME LIKE 'ndb_api%count';
+-----------------------------------+----------------+
| VARIABLE_NAME                     | VARIABLE_VALUE |
+-----------------------------------+----------------+
| NDB_API_WAIT_EXEC_COMPLETE_COUNT  | 4              |
| NDB_API_WAIT_SCAN_RESULT_COUNT    | 3              |
| NDB_API_WAIT_META_REQUEST_COUNT   | 28             |
| NDB_API_WAIT_NANOS_COUNT          | 53756398       |
| NDB_API_BYTES_SENT_COUNT          | 1060           |
| NDB_API_BYTES_RECEIVED_COUNT      | 9724           |
| NDB_API_TRANS_START_COUNT         | 3              |
| NDB_API_TRANS_COMMIT_COUNT        | 2              |
| NDB_API_TRANS_ABORT_COUNT         | 0              |
| NDB_API_TRANS_CLOSE_COUNT         | 3              |
| NDB_API_PK_OP_COUNT               | 2              |
| NDB_API_UK_OP_COUNT               | 0              |
| NDB_API_TABLE_SCAN_COUNT          | 1              |
| NDB_API_RANGE_SCAN_COUNT          | 0              |
| NDB_API_PRUNED_SCAN_COUNT         | 0              |
| NDB_API_SCAN_BATCH_COUNT          | 0              |
| NDB_API_READ_ROW_COUNT            | 2              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT | 2              |
| NDB_API_EVENT_DATA_COUNT          | 0              |
| NDB_API_EVENT_NONDATA_COUNT       | 0              |
| NDB_API_EVENT_BYTES_COUNT         | 0              |
+-----------------------------------+----------------+
21 rows in set (0.09 sec)
```

Not all counters are reflected in all 4 sets of status variables. For the event counters `DataEventsRecvdCount`, `NondataEventsRecvdCount`, and `EventBytesRecvdCount`, only `_injector` and `mysqld`-level NDB API status variables are available:

```
mysql> SHOW STATUS LIKE 'ndb_api%event%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Ndb_api_event_data_count_injector     | 0     |
| Ndb_api_event_nondata_count_injector  | 0     |
| Ndb_api_event_bytes_count_injector    | 0     |
| Ndb_api_event_data_count              | 0     |
| Ndb_api_event_nondata_count           | 0     |
| Ndb_api_event_bytes_count             | 0     |
+---------------------------------------+-------+
6 rows in set (0.00 sec)
```

`_injector` status variables are not implemented for any other NDB API counters, as shown here:

```
mysql> SHOW STATUS LIKE 'ndb_api%injector%';
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Ndb_api_event_data_count_injector     | 0     |
| Ndb_api_event_nondata_count_injector  | 0     |
| Ndb_api_event_bytes_count_injector    | 0     |
+---------------------------------------+-------+
3 rows in set (0.00 sec)
```

The names of the status variables can easily be associated with the names of the corresponding counters. Each NDB API statistics counter is listed in the following table with a description as well as the names of any MySQL server status variables corresponding to this counter.

Table 22.414 NDB API statistics counters

Counter Name	Description	Status Variables (by statistic type): • **Session** • **Slave** • **Injector** • **Server**
`WaitExecCompleteCount`	Number of times thread has been blocked while waiting for execution of an operation to complete. Includes all `execute()` calls as well as implicit executes for blob operations and auto-increment not visible to clients.	• `Ndb_api_wait_exec_complete_count` • `Ndb_api_wait_exec_complete_count` • [none] • `Ndb_api_wait_exec_complete_count`
`WaitScanResultCount`	Number of times thread has been blocked while waiting for a scan-based signal, such waiting for additional results, or for a scan to close.	• `Ndb_api_wait_scan_result_count_s` • `Ndb_api_wait_scan_result_count_s` • [none] • `Ndb_api_wait_scan_result_count`
`WaitMetaRequestCount`	Number of times thread has been blocked waiting for a metadata-based signal; this can occur when waiting for a DDL operation or for an epoch to be started (or ended).	• `Ndb_api_wait_meta_request_count_` • `Ndb_api_wait_meta_request_count_` • [none] • `Ndb_api_wait_meta_request_count`
`WaitNanosCount`	Total time (in nanoseconds) spent waiting for some type of signal from the data nodes.	• `Ndb_api_wait_nanos_count_session` • `Ndb_api_wait_nanos_count_slave` • [none] • `Ndb_api_wait_nanos_count`
`BytesSentCount`	Amount of data (in bytes) sent to the data nodes	• `Ndb_api_bytes_sent_count_session` • `Ndb_api_bytes_sent_count_slave` • [none] • `Ndb_api_bytes_sent_count`
`BytesRecvdCount`	Amount of data (in bytes) received from the data nodes	• `Ndb_api_bytes_received_count_ses` • `Ndb_api_bytes_received_count_sla` • [none]

Counter Name	Description	Status Variables (by statistic type): • **Session** • **Slave** • **Injector** • **Server**
		• Ndb_api_bytes_received_count
TransStartCount	Number of transactions started.	• Ndb_api_trans_start_count_ses • Ndb_api_trans_start_count_sla • [none] • Ndb_api_trans_start_count
TransCommitCount	Number of transactions committed.	• Ndb_api_trans_commit_count_se • Ndb_api_trans_commit_count_sl • [none] • Ndb_api_trans_commit_count
TransAbortCount	Number of transactions aborted.	• Ndb_api_trans_abort_count_ses • Ndb_api_trans_abort_count_sla • [none] • Ndb_api_trans_abort_count
TransCloseCount	Number of transactions aborted. (This value may be greater than the sum of TransCommitCount and TransAbortCount.)	• Ndb_api_trans_close_count_ses • Ndb_api_trans_close_count_sla • [none] • Ndb_api_trans_close_count
PkOpCount	Number of operations based on or using primary keys. This count includes blob-part table operations, implicit unlocking operations, and auto-increment operations, as well as primary key operations normally visible to MySQL clients.	• Ndb_api_pk_op_count_session • Ndb_api_pk_op_count_slave • [none] • Ndb_api_pk_op_count
UkOpCount	Number of operations based on or using unique keys.	• Ndb_api_uk_op_count_session • Ndb_api_uk_op_count_slave • [none] • Ndb_api_uk_op_count

Counter Name	Description	Status Variables (by statistic type): • **Session** • **Slave** • **Injector** • **Server**
TableScanCount	Number of table scans that have been started. This includes scans of internal tables.	• Ndb_api_table_scan_count_session • Ndb_api_table_scan_count_slave • [none] • Ndb_api_table_scan_count
RangeScanCount	Number of range scans that have been started.	• Ndb_api_range_scan_count_session • Ndb_api_range_scan_count_slave • [none] • Ndb_api_range_scan_count
PrunedScanCount	Number of scans that have been pruned to a single partition.	• Ndb_api_pruned_scan_count_sessio • Ndb_api_pruned_scan_count_slave • [none] • Ndb_api_pruned_scan_count
ScanBatchCount	Number of batches of rows received. (A *batch* in this context is a set of scan results from a single fragment.)	• Ndb_api_scan_batch_count_session • Ndb_api_scan_batch_count_slave • [none] • Ndb_api_scan_batch_count
ReadRowCount	Total number of rows that have been read. Includes rows read using primary key, unique key, and scan operations.	• Ndb_api_read_row_count_session • Ndb_api_read_row_count_slave • [none] • Ndb_api_read_row_count
TransLocalReadRowCount	Number of rows read from the data same node on which the transaction was being run.	• Ndb_api_trans_local_read_row_cou • Ndb_api_trans_local_read_row_cou • [none] • Ndb_api_trans_local_read_row_cou
DataEventsRecvdCount	Number of row change events received.	• [none] • [none]

Counter Name	Description	Status Variables (by statistic type): • **Session** • **Slave** • **Injector** • **Server**
		• Ndb_api_event_data_count_inje • Ndb_api_event_data_count
NondataEventsRecvdCount	Number of events received, other than row change events.	• [none] • [none] • Ndb_api_event_nondata_count_i • Ndb_api_event_nondata_count
EventBytesRecvdCount	Number of bytes of events received.	• [none] • [none] • Ndb_api_event_bytes_count_inj • Ndb_api_event_bytes_count

To see all counts of committed transactions—that is, all TransCommitCount counter status variables—you can filter the results of SHOW STATUS for the substring trans_commit_count, like this:

```
mysql> SHOW STATUS LIKE '%trans_commit_count%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| Ndb_api_trans_commit_count_session | 1     |
| Ndb_api_trans_commit_count_slave   | 0     |
| Ndb_api_trans_commit_count         | 2     |
+-----------------------------------+-------+
3 rows in set (0.00 sec)
```

From this you can determine that 1 transaction has been committed in the current mysql client session, and 2 transactions have been committed on this mysqld since it was last restarted.

You can see how various NDB API counters are incremented by a given SQL statement by comparing the values of the corresponding _session status variables immediately before and after performing the statement. In this example, after getting the initial values from SHOW STATUS, we create in the test database an NDB table, named t, that has a single column:

```
mysql> SHOW STATUS LIKE 'ndb_api%session%';
+-------------------------------------------+--------+
| Variable_name                             | Value  |
+-------------------------------------------+--------+
| Ndb_api_wait_exec_complete_count_session   | 2      |
| Ndb_api_wait_scan_result_count_session     | 0      |
| Ndb_api_wait_meta_request_count_session    | 3      |
| Ndb_api_wait_nanos_count_session           | 820705 |
| Ndb_api_bytes_sent_count_session           | 132    |
| Ndb_api_bytes_received_count_session       | 372    |
```

```
| Ndb_api_trans_start_count_session          | 1      |
| Ndb_api_trans_commit_count_session         | 1      |
| Ndb_api_trans_abort_count_session          | 0      |
| Ndb_api_trans_close_count_session          | 1      |
| Ndb_api_pk_op_count_session                | 1      |
| Ndb_api_uk_op_count_session                | 0      |
| Ndb_api_table_scan_count_session           | 0      |
| Ndb_api_range_scan_count_session           | 0      |
| Ndb_api_pruned_scan_count_session          | 0      |
| Ndb_api_scan_batch_count_session           | 0      |
| Ndb_api_read_row_count_session             | 1      |
| Ndb_api_trans_local_read_row_count_session | 1      |
+--------------------------------------------+--------+
18 rows in set (0.00 sec)

mysql> USE test;
Database changed
mysql> CREATE TABLE t (c INT) ENGINE NDBCLUSTER;
Query OK, 0 rows affected (0.85 sec)
```

Now you can execute a new `SHOW STATUS` statement and observe the changes, as shown here (with the changed rows highlighted in the output):

```
mysql> SHOW STATUS LIKE 'ndb_api%session%';
+--------------------------------------------+-----------+
| Variable_name                              | Value     |
+--------------------------------------------+-----------+
| Ndb_api_wait_exec_complete_count_session   | 8         |
| Ndb_api_wait_scan_result_count_session     | 0         |
| Ndb_api_wait_meta_request_count_session    | 17        |
| Ndb_api_wait_nanos_count_session           | 706871709 |
| Ndb_api_bytes_sent_count_session           | 2376      |
| Ndb_api_bytes_received_count_session       | 3844      |
| Ndb_api_trans_start_count_session          | 4         |
| Ndb_api_trans_commit_count_session         | 4         |
| Ndb_api_trans_abort_count_session          | 0         |
| Ndb_api_trans_close_count_session          | 4         |
| Ndb_api_pk_op_count_session                | 6         |
| Ndb_api_uk_op_count_session                | 0         |
| Ndb_api_table_scan_count_session           | 0         |
| Ndb_api_range_scan_count_session           | 0         |
| Ndb_api_pruned_scan_count_session          | 0         |
| Ndb_api_scan_batch_count_session           | 0         |
| Ndb_api_read_row_count_session             | 2         |
| Ndb_api_trans_local_read_row_count_session | 1         |
+--------------------------------------------+-----------+
18 rows in set (0.00 sec)
```

Similarly, you can see the changes in the NDB API statistics counters caused by inserting a row into `t`: Insert the row, then run the same `SHOW STATUS` statement used in the previous example, as shown here:

```
mysql> INSERT INTO t VALUES (100);
Query OK, 1 row affected (0.00 sec)

mysql> SHOW STATUS LIKE 'ndb_api%session%';
+--------------------------------------------+-----------+
| Variable_name                              | Value     |
+--------------------------------------------+-----------+
| Ndb_api_wait_exec_complete_count_session   | 11        |
| Ndb_api_wait_scan_result_count_session     | 6         |
| Ndb_api_wait_meta_request_count_session    | 20        |
| Ndb_api_wait_nanos_count_session           | 707370418 |
| Ndb_api_bytes_sent_count_session           | 2724      |
| Ndb_api_bytes_received_count_session       | 4116      |
| Ndb_api_trans_start_count_session          | 7         |
```

```
| Ndb_api_trans_commit_count_session         | 6         |
| Ndb_api_trans_abort_count_session          | 0         |
| Ndb_api_trans_close_count_session          | 7         |
| Ndb_api_pk_op_count_session                | 8         |
| Ndb_api_uk_op_count_session                | 0         |
| Ndb_api_table_scan_count_session           | 1         |
| Ndb_api_range_scan_count_session           | 0         |
| Ndb_api_pruned_scan_count_session          | 0         |
| Ndb_api_scan_batch_count_session           | 0         |
| Ndb_api_read_row_count_session             | 3         |
| Ndb_api_trans_local_read_row_count_session | 2         |
+--------------------------------------------+-----------+
18 rows in set (0.00 sec)
```

We can make a number of observations from these results:

- Although we created `t` with no explicit primary key, 5 primary key operations were performed in doing so (the difference in the “before” and “after” values of `Ndb_api_pk_op_count_session`, or 6 minus 1). This reflects the creation of the hidden primary key that is a feature of all tables using the `NDB` storage engine.

- By comparing successive values for `Ndb_api_wait_nanos_count_session`, we can see that the NDB API operations implementing the `CREATE TABLE` statement waited much longer (706871709 - 820705 = 706051004 nanoseconds, or approximately 0.7 second) for responses from the data nodes than those executed by the `INSERT` (707370418 - 706871709 = 498709 ns or roughly .0005 second). The execution times reported for these statements in the `mysql` client correlate roughly with these figures.

  On platforms without sufficient (nanosecond) time resolution, small changes in the value of the `WaitNanosCount` NDB API counter due to SQL statements that execute very quickly may not always be visible in the values of `Ndb_api_wait_nanos_count_session`, `Ndb_api_wait_nanos_count_slave`, or `Ndb_api_wait_nanos_count`.

- The `INSERT` statement incremented both the `ReadRowCount` and `TransLocalReadRowCount` NDB API statistics counters, as reflected by the increased values of `Ndb_api_read_row_count_session` and `Ndb_api_trans_local_read_row_count_session`.

22.6 NDB Cluster Replication

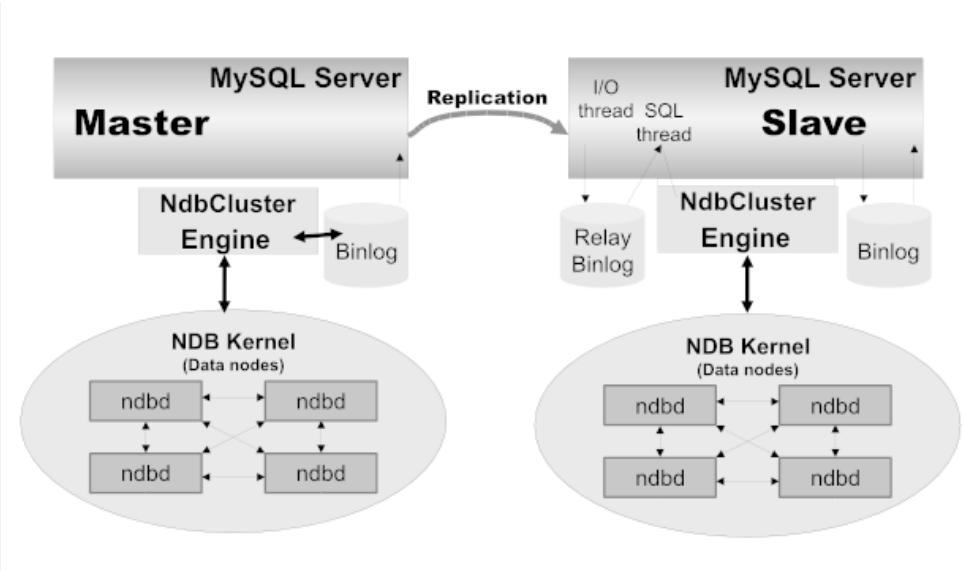
NDB Cluster supports *asynchronous replication*, more usually referred to simply as “replication”. This section explains how to set up and manage a configuration in which one group of computers operating as an NDB Cluster replicates to a second computer or group of computers. We assume some familiarity on the part of the reader with standard MySQL replication as discussed elsewhere in this Manual. (See Chapter 17, *Replication*).

Note

NDB Cluster does not support replication using GTIDs; semisynchronous replication is also not supported by the `NDB` storage engine.

Normal (non-clustered) replication involves a “master” server and a “slave” server, the master being the source of the operations and data to be replicated and the slave being the recipient of these. In NDB Cluster, replication is conceptually very similar but can be more complex in practice, as it may be extended to cover a number of different configurations including replicating between two complete clusters. Although an NDB Cluster itself depends on the `NDB` storage engine for clustering functionality, it is not necessary to use `NDB` as the storage engine for the slave's copies of the replicated tables (see Replication from NDB to other storage engines). However, for maximum availability, it is possible (and preferable) to replicate from one NDB Cluster to another, and it is this scenario that we discuss, as shown in the following figure:

Figure 22.30 NDB Cluster-to-Cluster Replication Layout



In this scenario, the replication process is one in which successive states of a master cluster are logged and saved to a slave cluster. This process is accomplished by a special thread known as the NDB binary log injector thread, which runs on each MySQL server and produces a binary log (`binlog`). This thread ensures that all changes in the cluster producing the binary log—and not just those changes that are effected through the MySQL Server—are inserted into the binary log with the correct serialization order. We refer to the MySQL replication master and replication slave servers as replication servers or replication nodes, and the data flow or line of communication between them as a *replication channel*.

For information about performing point-in-time recovery with NDB Cluster and NDB Cluster Replication, see Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”.

NDB API _slave status variables. NDB API counters can provide enhanced monitoring capabilities on NDB Cluster replication slaves. These are implemented as NDB statistics `_slave` status variables, as seen in the output of `SHOW STATUS`, or in the results of queries against the `SESSION_STATUS` or `GLOBAL_STATUS` table in a `mysql` client session connected to a MySQL Server that is acting as a slave in NDB Cluster Replication. By comparing the values of these status variables before and after the execution of statements affecting replicated `NDB` tables, you can observe the corresponding actions taken on the NDB API level by the slave, which can be useful when monitoring or troubleshooting NDB Cluster Replication. Section 22.5.17, “NDB API Statistics Counters and Variables”, provides additional information.

Replication from NDB to non-NDB tables. It is possible to replicate `NDB` tables from an NDB Cluster acting as the master to tables using other MySQL storage engines such as `InnoDB` or `MyISAM` on a slave `mysqld`. This is subject to a number of conditions; see Replication from NDB to other storage engines, and Replication from NDB to a nontransactional storage engine, for more information.

22.6.1 NDB Cluster Replication: Abbreviations and Symbols

Throughout this section, we use the following abbreviations or symbols for referring to the master and slave clusters, and to processes and commands run on the clusters or cluster nodes:

Table 22.415 Abbreviations used throughout this section referring to master and slave clusters, and to processes and commands run on nodes

Symbol or Abbreviation	Description (Refers to...)
M	The cluster serving as the (primary) replication master

Symbol or Abbreviation	Description (Refers to...)
S	The cluster acting as the (primary) replication slave
shell*M*>	Shell command to be issued on the master cluster
mysql*M*>	MySQL client command issued on a single MySQL server running as an SQL node on the master cluster
mysql*M**>	MySQL client command to be issued on all SQL nodes participating in the replication master cluster
shell*S*>	Shell command to be issued on the slave cluster
mysql*S*>	MySQL client command issued on a single MySQL server running as an SQL node on the slave cluster
mysql*S**>	MySQL client command to be issued on all SQL nodes participating in the replication slave cluster
C	Primary replication channel
C'	Secondary replication channel
M'	Secondary replication master
S'	Secondary replication slave

22.6.2 General Requirements for NDB Cluster Replication

A replication channel requires two MySQL servers acting as replication servers (one each for the master and slave). For example, this means that in the case of a replication setup with two replication channels (to provide an extra channel for redundancy), there will be a total of four replication nodes, two per cluster.

Replication of an NDB Cluster as described in this section and those following is dependent on row-based replication. This means that the replication master MySQL server must be running with `--binlog-format=ROW` or `--binlog-format=MIXED`, as described in Section 22.6.6, "Starting NDB Cluster Replication (Single Replication Channel)". For general information about row-based replication, see Section 17.2.1, "Replication Formats".

Important

If you attempt to use NDB Cluster Replication with `--binlog-format=STATEMENT`, replication fails to work properly because the `ndb_binlog_index` table on the master and the `epoch` column of the `ndb_apply_status` table on the slave are not updated (see Section 22.6.4, "NDB Cluster Replication Schema and Tables"). Instead, only updates on the MySQL server acting as the replication master propagate to the slave, and no updates from any other SQL nodes on the master cluster are replicated.

The default value for the `--binlog-format` option in NDB 8.0 is `MIXED`.

Each MySQL server used for replication in either cluster must be uniquely identified among all the MySQL replication servers participating in either cluster (you cannot have replication servers on both the master and slave clusters sharing the same ID). This can be done by starting each SQL node using the `--server-id=id` option, where *id* is a unique integer. Although it is not strictly necessary, we will assume for purposes of this discussion that all NDB Cluster binaries are of the same release version.

It is generally true in MySQL Replication that both MySQL servers (`mysqld` processes) involved must be compatible with one another with respect to both the version of the replication protocol used and the SQL feature sets which they support (see Section 17.5.2, "Replication Compatibility Between MySQL

Versions"). It is due to such differences between the binaries in the NDB Cluster and MySQL Server 8.0 distributions that NDB Cluster Replication has the additional requirement that both `mysqld` binaries come from an NDB Cluster distribution. The simplest and easiest way to assure that the `mysqld` servers are compatible is to use the same NDB Cluster distribution for all master and slave `mysqld` binaries.

We assume that the slave server or cluster is dedicated to replication of the master, and that no other data is being stored on it.

All `NDB` tables being replicated must be created using a MySQL server and client. Tables and other database objects created using the NDB API (with, for example, `Dictionary::createTable()`) are not visible to a MySQL server and so are not replicated. Updates by NDB API applications to existing tables that were created using a MySQL server can be replicated.

Note

It is possible to replicate an NDB Cluster using statement-based replication. However, in this case, the following restrictions apply:

- All updates to data rows on the cluster acting as the master must be directed to a single MySQL server.
- It is not possible to replicate a cluster using multiple simultaneous MySQL replication processes.
- Only changes made at the SQL level are replicated.

These are in addition to the other limitations of statement-based replication as opposed to row-based replication; see Section 17.2.1.1, "Advantages and Disadvantages of Statement-Based and Row-Based Replication", for more specific information concerning the differences between the two replication formats.

22.6.3 Known Issues in NDB Cluster Replication

This section discusses known problems or issues when using replication with NDB Cluster 8.0.

Loss of master-slave connection. A loss of connection can occur either between the replication master SQL node and the replication slave SQL node, or between the replication master SQL node and the data nodes in the master cluster. In the latter case, this can occur not only as a result of loss of physical connection (for example, a broken network cable), but due to the overflow of data node event buffers; if the SQL node is too slow to respond, it may be dropped by the cluster (this is controllable to some degree by adjusting the `MaxBufferedEpochs` and `TimeBetweenEpochs` configuration parameters). If this occurs, *it is entirely possible for new data to be inserted into the master cluster without being recorded in the replication master's binary log*. For this reason, to guarantee high availability, it is extremely important to maintain a backup replication channel, to monitor the primary channel, and to fail over to the secondary replication channel when necessary to keep the slave cluster synchronized with the master. NDB Cluster is not designed to perform such monitoring on its own; for this, an external application is required.

The replication master issues a "gap" event when connecting or reconnecting to the master cluster. (A gap event is a type of "incident event," which indicates an incident that occurs that affects the contents of the database but that cannot easily be represented as a set of changes. Examples of incidents are server crashes, database resynchronization, (some) software updates, and (some) hardware changes.) When the slave encounters a gap in the replication log, it stops with an error message. This message is available in the output of `SHOW SLAVE STATUS`, and indicates that the SQL thread has stopped due to an incident registered in the replication stream, and that manual intervention is required. See Section 22.6.8,

"Implementing Failover with NDB Cluster Replication", for more information about what to do in such circumstances.

Important

Because NDB Cluster is not designed on its own to monitor replication status or provide failover, if high availability is a requirement for the slave server or cluster, then you must set up multiple replication lines, monitor the master `mysqld` on the primary replication line, and be prepared fail over to a secondary line if and as necessary. This must be done manually, or possibly by means of a third-party application. For information about implementing this type of setup, see Section 22.6.7, "Using Two Replication Channels for NDB Cluster Replication", and Section 22.6.8, "Implementing Failover with NDB Cluster Replication".

However, if you are replicating from a standalone MySQL server to an NDB Cluster, one channel is usually sufficient.

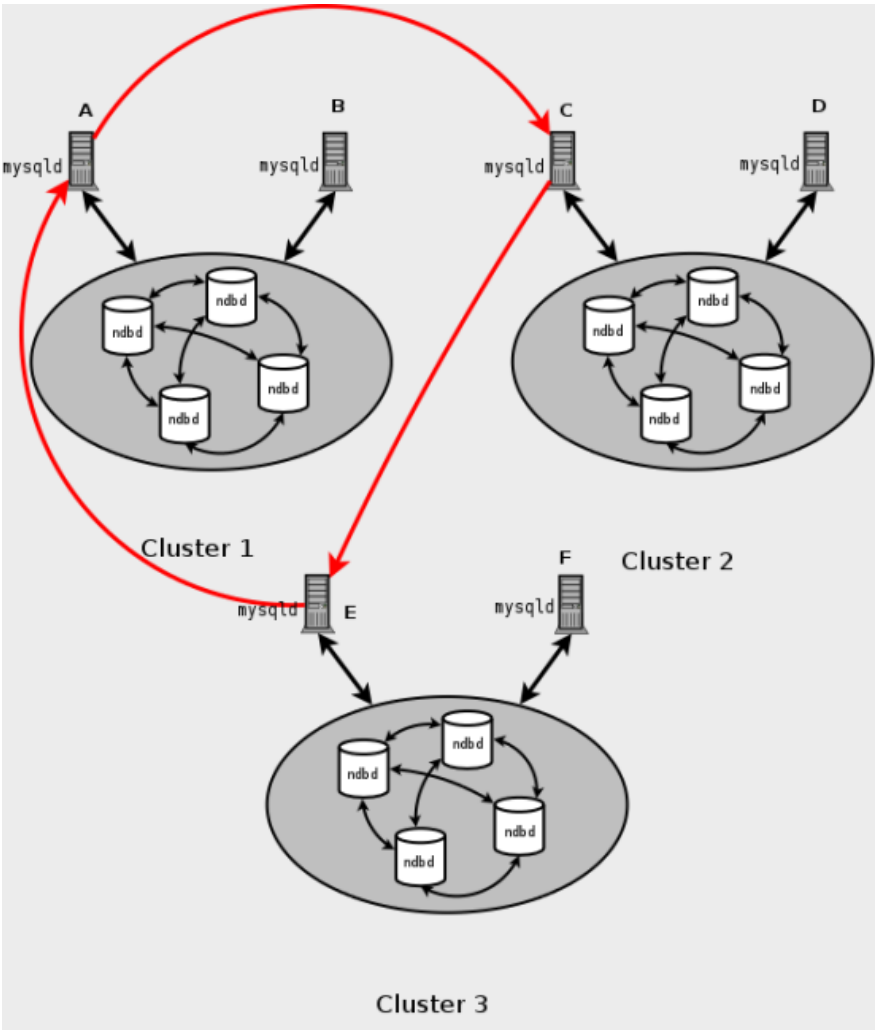
Circular replication. NDB Cluster Replication supports circular replication, as shown in the next example. The replication setup involves three NDB Clusters numbered 1, 2, and 3, in which Cluster 1 acts as the replication master for Cluster 2, Cluster 2 acts as the master for Cluster 3, and Cluster 3 acts as the master for Cluster 1, thus completing the circle. Each NDB Cluster has two SQL nodes, with SQL nodes A and B belonging to Cluster 1, SQL nodes C and D belonging to Cluster 2, and SQL nodes E and F belonging to Cluster 3.

Circular replication using these clusters is supported as long as the following conditions are met:

- The SQL nodes on all masters and slaves are the same.
- All SQL nodes acting as replication masters and slaves are started with the `log_slave_updates` system variable enabled.

This type of circular replication setup is shown in the following diagram:

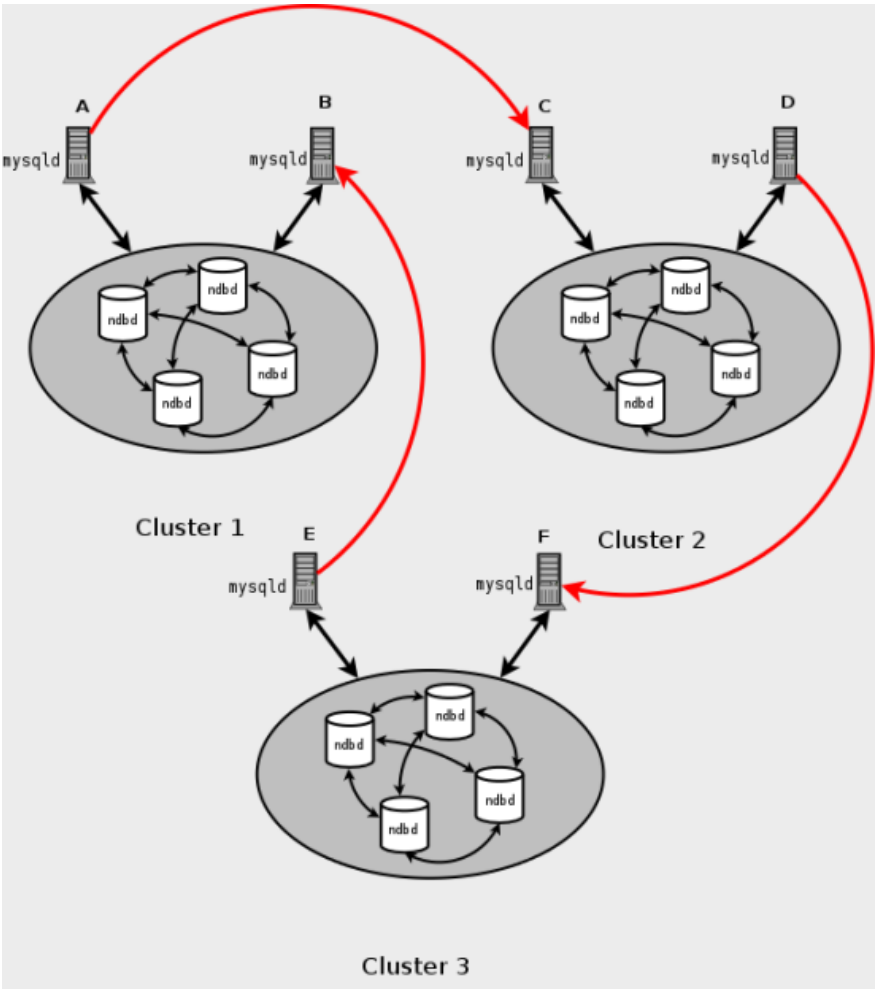
Figure 22.31 NDB Cluster Circular Replication With All Masters As Slaves



In this scenario, SQL node A in Cluster 1 replicates to SQL node C in Cluster 2; SQL node C replicates to SQL node E in Cluster 3; SQL node E replicates to SQL node A. In other words, the replication line (indicated by the curved arrows in the diagram) directly connects all SQL nodes used as replication masters and slaves.

It should also be possible to set up circular replication in which not all master SQL nodes are also slaves, as shown here:

Figure 22.32 NDB Cluster Circular Replication Where Not All Masters Are Slaves



In this case, different SQL nodes in each cluster are used as replication masters and slaves. However, you must *not* start any of the SQL nodes with the `log_slave_updates` system variable enabled. This type of circular replication scheme for NDB Cluster, in which the line of replication (again indicated by the curved arrows in the diagram) is discontinuous, should be possible, but it should be noted that it has not yet been thoroughly tested and must therefore still be considered experimental.

Note

The `NDB` storage engine uses *idempotent execution mode*, which suppresses duplicate-key and other errors that otherwise break circular replication of NDB Cluster. This is equivalent to setting the global `slave_exec_mode` system variable to `IDEMPOTENT`, although this is not necessary in NDB Cluster replication, since NDB Cluster sets this variable automatically and ignores any attempts to set it explicitly.

NDB Cluster replication and primary keys. In the event of a node failure, errors in replication of `NDB` tables without primary keys can still occur, due to the possibility of duplicate rows being inserted in such cases. For this reason, it is highly recommended that all `NDB` tables being replicated have primary keys.

NDB Cluster Replication and Unique Keys. In older versions of NDB Cluster, operations that updated values of unique key columns of `NDB` tables could result in duplicate-key errors when replicated.

This issue is solved for replication between NDB tables by deferring unique key checks until after all table row updates have been performed.

Deferring constraints in this way is currently supported only by NDB. Thus, updates of unique keys when replicating from NDB to a different storage engine such as MyISAM or InnoDB are still not supported.

The problem encountered when replicating without deferred checking of unique key updates can be illustrated using NDB table such as t, is created and populated on the master (and replicated to a slave that does not support deferred unique key updates) as shown here:

```
CREATE TABLE t (
    p INT PRIMARY KEY,
    c INT,
    UNIQUE KEY u (c)
)   ENGINE NDB;

INSERT INTO t
    VALUES (1,1), (2,2), (3,3), (4,4), (5,5);
```

The following UPDATE statement on t succeeded on the master, since the rows affected are processed in the order determined by the ORDER BY option, performed over the entire table:

```
UPDATE t SET c = c - 1 ORDER BY p;
```

However, the same statement failed with a duplicate key error or other constraint violation on the slave, because the ordering of the row updates was done for one partition at a time, rather than for the table as a whole.

Note

Every NDB table is implicitly partitioned by key when it is created. See Section 23.2.5, “KEY Partitioning”, for more information.

GTIDs not supported. Replication using global transaction IDs is not compatible with the NDB storage engine, and is not supported. Enabling GTIDs is likely to cause NDB Cluster Replication to fail.

Multithreaded slaves not supported. NDB Cluster does not support multithreaded slaves, and setting related system variables such as slave_parallel_workers, slave_checkpoint_group, and slave_checkpoint_group (or the equivalent mysqld startup options) has no effect.

This is because the slave may not be able to separate transactions occurring in one database from those in another if they are written within the same epoch. In addition, every transaction handled by the NDB storage engine involves at least two databases—the target database and the mysql system database—due to the requirement for updating the mysql.ndb_apply_status table (see Section 22.6.4, “NDB Cluster Replication Schema and Tables”). This in turn breaks the requirement for multithreading that the transaction is specific to a given database.

Restarting with --initial. Restarting the cluster with the --initial option causes the sequence of GCI and epoch numbers to start over from 0. (This is generally true of NDB Cluster and not limited to replication scenarios involving Cluster.) The MySQL servers involved in replication should in this case be restarted. After this, you should use the RESET MASTER and RESET SLAVE statements to clear the invalid ndb_binlog_index and ndb_apply_status tables, respectively.

Replication from NDB to other storage engines. It is possible to replicate an NDB table on the master to a table using a different storage engine on the slave, taking into account the restrictions listed here:

- Multi-master and circular replication are not supported (tables on both the master and the slave must use the `NDB` storage engine for this to work).
- Using a storage engine which does not perform binary logging for slave tables requires special handling.
- Use of a nontransactional storage engine for slave tables also requires special handling.
- The master `mysqld` must be started with `--ndb-log-update-as-write=0` or `--ndb-log-update-as-write=OFF`.

The next few paragraphs provide additional information about each of the issues just described.

Multiple masters not supported when replicating NDB to other storage engines. For replication from `NDB` to a different storage engine, the relationship between the two databases must be a simple master-slave one. This means that circular or master-master replication is not supported between NDB Cluster and other storage engines.

In addition, it is not possible to configure more than one replication channel when replicating between `NDB` and a different storage engine. (However, an NDB Cluster database *can* simultaneously replicate to multiple slave NDB Cluster databases.) If the master uses `NDB` tables, it is still possible to have more than one MySQL Server maintain a binary log of all changes; however, for the slave to change masters (fail over), the new master-slave relationship must be explicitly defined on the slave.

Replicating NDB to a slave storage engine that does not perform binary logging. If you attempt to replicate from an NDB Cluster to a slave that uses a storage engine that does not handle its own binary logging, the replication process aborts with the error `Binary logging not possible ... Statement cannot be written atomically since more than one engine involved and at least one engine is self-logging` (Error `1595`). It is possible to work around this issue in one of the following ways:

- **Turn off binary logging on the slave.** This can be accomplished by setting `sql_log_bin = 0`.
- **Change the storage engine used for the mysql.ndb_apply_status table.** Causing this table to use an engine that does not handle its own binary logging can also eliminate the conflict. This can be done by issuing a statement such as `ALTER TABLE mysql.ndb_apply_status ENGINE=MyISAM` on the slave. It is safe to do this when using a non-`NDB` storage engine on the slave, since you do not then need to worry about keeping multiple slave SQL nodes synchronized.
- **Filter out changes to the mysql.ndb_apply_status table on the slave.** This can be done by starting the slave SQL node with `--replicate-ignore-table=mysql.ndb_apply_status`. If you need for other tables to be ignored by replication, you might wish to use an appropriate `--replicate-wild-ignore-table` option instead.

Important

You should *not* disable replication or binary logging of `mysql.ndb_apply_status` or change the storage engine used for this table when replicating from one NDB Cluster to another. See Replication and binary log filtering rules with replication between NDB Clusters, for details.

Replication from NDB to a nontransactional storage engine. When replicating from `NDB` to a nontransactional storage engine such as `MyISAM`, you may encounter unnecessary duplicate key errors when replicating `INSERT ... ON DUPLICATE KEY UPDATE` statements. You can suppress these by using `--ndb-log-update-as-write=0`, which forces updates to be logged as writes (rather than as updates).

Replication and binary log filtering rules with replication between NDB Clusters. If you are using any of the options `--replicate-do-*`, `--replicate-ignore-*`, `--binlog-do-db`, or `--binlog-`

`ignore-db` to filter databases or tables being replicated, care must be taken not to block replication or binary logging of the `mysql.ndb_apply_status`, which is required for replication between NDB Clusters to operate properly. In particular, you must keep in mind the following:

1. Using `--replicate-do-db=db_name` (and no other `--replicate-do-*` or `--replicate-ignore-*` options) means that *only* tables in database *db_name* are replicated. In this case, you should also use `--replicate-do-db=mysql`, `--binlog-do-db=mysql`, or `--replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is populated on slaves.

   Using `--binlog-do-db=db_name` (and no other `--binlog-do-db` options) means that changes *only* to tables in database *db_name* are written to the binary log. In this case, you should also use `--replicate-do-db=mysql`, `--binlog-do-db=mysql`, or `--replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is populated on slaves.

2. Using `--replicate-ignore-db=mysql` means that no tables in the `mysql` database are replicated. In this case, you should also use `--replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is replicated.

   Using `--binlog-ignore-db=mysql` means that no changes to tables in the `mysql` database are written to the binary log. In this case, you should also use `--replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is replicated.

You should also remember that each replication rule requires the following:

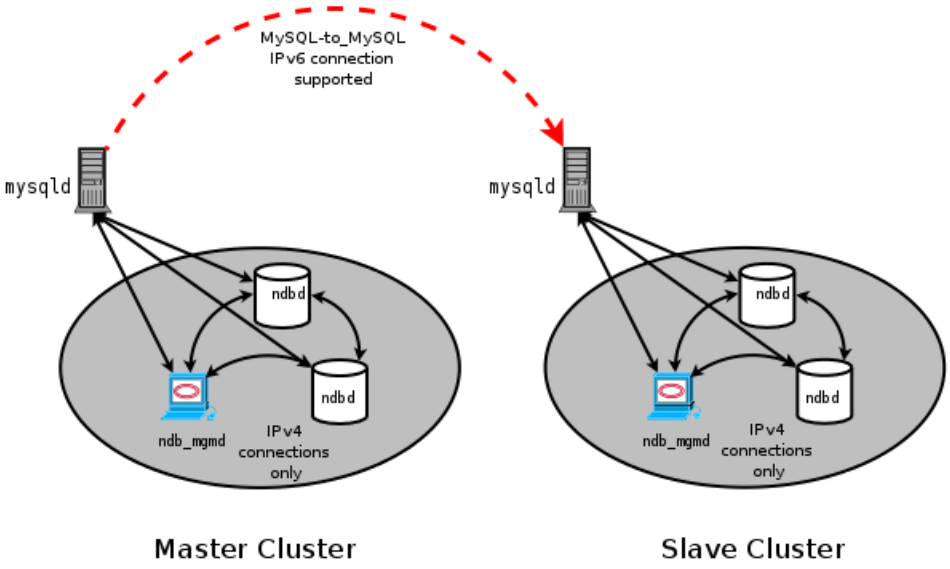
1. Its own `--replicate-do-*` or `--replicate-ignore-*` option, and that multiple rules cannot be expressed in a single replication filtering option. For information about these rules, see Section 17.1.6, “Replication and Binary Logging Options and Variables”.

2. Its own `--binlog-do-db` or `--binlog-ignore-db` option, and that multiple rules cannot be expressed in a single binary log filtering option. For information about these rules, see Section 5.4.4, “The Binary Log”.

If you are replicating an NDB Cluster to a slave that uses a storage engine other than `NDB`, the considerations just given previously may not apply, as discussed elsewhere in this section.

NDB Cluster Replication and IPv6. Currently, the NDB API and MGM API do not support IPv6. However, MySQL Servers—including those acting as SQL nodes in an NDB Cluster —can use IPv6 to contact other MySQL Servers. This means that you can replicate between NDB Clusters using IPv6 to connect the master and slave SQL nodes as shown by the dotted arrow in the following diagram:

Figure 22.33 Replication Between SQL Nodes Connected Using IPv6



However, all connections originating *within* the NDB Cluster —represented in the preceding diagram by solid arrows—must use IPv4. In other words, all NDB Cluster data nodes, management servers, and management clients must be accessible from one another using IPv4. In addition, SQL nodes must use IPv4 to communicate with the cluster.

Since there is currently no support in the NDB and MGM APIs for IPv6, any applications written using these APIs must also make all connections using IPv4.

Attribute promotion and demotion. NDB Cluster Replication includes support for attribute promotion and demotion. The implementation of the latter distinguishes between lossy and non-lossy type conversions, and their use on the slave can be controlled by setting the `slave_type_conversions` global server system variable.

For more information about attribute promotion and demotion in NDB Cluster, see Row-based replication: attribute promotion and demotion.

`NDB`, unlike `InnoDB` or `MyISAM`, does not write changes to virtual columns to the binary log; however, this has no detrimental effects on NDB Cluster Replication or replication between `NDB` and other storage engines. Changes to stored generated columns are logged.

22.6.4 NDB Cluster Replication Schema and Tables

Replication in NDB Cluster makes use of a number of dedicated tables in the `mysql` database on each MySQL Server instance acting as an SQL node in both the cluster being replicated and the replication slave (whether the slave is a single server or a cluster). These tables are created during the MySQL installation process, and include a table for storing the binary log's indexing data. Since the `ndb_binlog_index` table is local to each MySQL server and does not participate in clustering, it uses the `InnoDB` storage engine. This means that it must be created separately on each `mysqld` participating in the master cluster. (However, the binary log itself contains updates from all MySQL servers in the cluster to be replicated.) This table is defined as follows:

```
CREATE TABLE `ndb_binlog_index` (
```

```
    `Position` BIGINT(20) UNSIGNED NOT NULL,
    `File` VARCHAR(255) NOT NULL,
    `epoch` BIGINT(20) UNSIGNED NOT NULL,
    `inserts` INT(10) UNSIGNED NOT NULL,
    `updates` INT(10) UNSIGNED NOT NULL,
    `deletes` INT(10) UNSIGNED NOT NULL,
    `schemaops` INT(10) UNSIGNED NOT NULL,
    `orig_server_id` INT(10) UNSIGNED NOT NULL,
    `orig_epoch` BIGINT(20) UNSIGNED NOT NULL,
    `gci` INT(10) UNSIGNED NOT NULL,
    `next_position` bigint(20) unsigned NOT NULL,
    `next_file` varchar(255) NOT NULL,
    PRIMARY KEY (`epoch`,`orig_server_id`,`orig_epoch`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Note

If you are upgrading from an old release (prior to NDB 7.5.2), perform the MySQL upgrade procedure and ensure that the system tables are upgraded. (As of MySQL 8.0.16, start the server with the `--upgrade=FORCE` option. Prior to MySQL 8.0.16, invoke `mysql_upgrade` with the `--force` and `--upgrade-system-tables` options after starting the server.) The system table upgrade causes an `ALTER TABLE ... ENGINE=INNODB` statement to be executed for this table. Use of the `MyISAM` storage engine for this table continues to be supported for backward compatibility.

`ndb_binlog_index` may require additional disk space after being converted to `InnoDB`. If this becomes an issue, you may be able to conserve space by using an `InnoDB` tablespace for this table, changing its `ROW_FORMAT` to `COMPRESSED`, or both. For more information, see Section 13.1.21, “CREATE TABLESPACE Syntax”, and Section 13.1.20, “CREATE TABLE Syntax”, as well as Section 15.6.3, “Tablespaces”.

The size of this table is dependent on the number of epochs per binary log file and the number of binary log files. The number of epochs per binary log file normally depends on the amount of binary log generated per epoch and the size of the binary log file, with smaller epochs resulting in more epochs per file. You should be aware that empty epochs produce inserts to the `ndb_binlog_index` table, even when the `--ndb-log-empty-epochs` option is `OFF`, meaning that the number of entries per file depends on the length of time that the file is in use; that is,

```
[number of epochs per file] = [time spent per file] / TimeBetweenEpochs
```

A busy NDB Cluster writes to the binary log regularly and presumably rotates binary log files more quickly than a quiet one. This means that a “quiet” NDB Cluster with `--ndb-log-empty-epochs=ON` can actually have a much higher number of `ndb_binlog_index` rows per file than one with a great deal of activity.

When `mysqld` is started with the `--ndb-log-orig` option, the `orig_server_id` and `orig_epoch` columns store, respectively, the ID of the server on which the event originated and the epoch in which the event took place on the originating server, which is useful in NDB Cluster replication setups employing multiple masters. The `SELECT` statement used to find the closest binary log position to the highest applied epoch on the slave in a multi-master setup (see Section 22.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”) employs these two columns, which are not indexed. This can lead to performance issues when trying to fail over, since the query must perform a table scan, especially when the master has been running with `--ndb-log-empty-epochs=ON`. You can improve multi-master failover times by adding an index to these columns, as shown here:

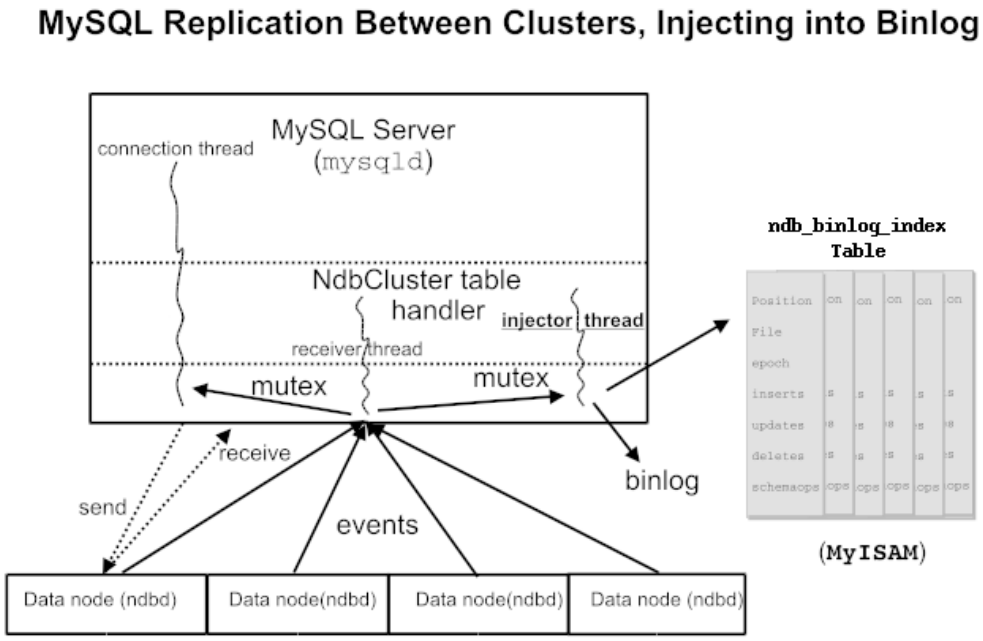
```
ALTER TABLE mysql.ndb_binlog_index
    ADD INDEX orig_lookup USING BTREE (orig_server_id, orig_epoch);
```

Adding this index provides no benefit when replicating from a single master to a single slave, since the query used to get the binary log position in such cases makes no use of `orig_server_id` or `orig_epoch`.

See Section 22.6.8, “Implementing Failover with NDB Cluster Replication”, for more information about using the `next_position` and `next_file` columns.

The following figure shows the relationship of the NDB Cluster replication master server, its binary log injector thread, and the `mysql.ndb_binlog_index` table.

Figure 22.34 The Replication Master Cluster



An additional table, named `ndb_apply_status`, is used to keep a record of the operations that have been replicated from the master to the slave. Unlike the case with `ndb_binlog_index`, the data in this table is not specific to any one SQL node in the (slave) cluster, and so `ndb_apply_status` can use the `NDBCLUSTER` storage engine, as shown here:

```
CREATE TABLE `ndb_apply_status` (
    `server_id`   INT(10) UNSIGNED NOT NULL,
    `epoch`       BIGINT(20) UNSIGNED NOT NULL,
    `log_name`    VARCHAR(255) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL,
    `start_pos`   BIGINT(20) UNSIGNED NOT NULL,
    `end_pos`     BIGINT(20) UNSIGNED NOT NULL,
    PRIMARY KEY (`server_id`) USING HASH
) ENGINE=NDBCLUSTER   DEFAULT CHARSET=latin1;
```

The `ndb_apply_status` table is populated only on slaves, which means that, on the master, this table never contains any rows; thus, there is no need to allow for `DataMemory` or `IndexMemory` to be allotted to `ndb_apply_status` there.

Because this table is populated from data originating on the master, it should be allowed to replicate; any replication filtering or binary log filtering rules that inadvertently prevent the slave from updating `ndb_apply_status` or the master from writing into the binary log may prevent replication between clusters from operating properly. For more information about potential problems arising from such filtering rules, see Replication and binary log filtering rules with replication between NDB Clusters.

The `ndb_binlog_index` and `ndb_apply_status` tables are created in the `mysql` database because they should not be explicitly replicated by the user. User intervention is normally not required to create or maintain either of these tables, since both `ndb_binlog_index` and the `ndb_apply_status` are maintained by the `NDB` binary log (binlog) injector thread. This keeps the master `mysqld` process updated to changes performed by the `NDB` storage engine. The `NDB` *binlog injector thread* receives events directly from the `NDB` storage engine. The `NDB` injector is responsible for capturing all the data events within the cluster, and ensures that all events which change, insert, or delete data are recorded in the `ndb_binlog_index` table. The slave I/O thread transfers the events from the master's binary log to the slave's relay log.

However, it is advisable to check for the existence and integrity of these tables as an initial step in preparing an NDB Cluster for replication. It is possible to view event data recorded in the binary log by querying the `mysql.ndb_binlog_index` table directly on the master. This can be also be accomplished using the `SHOW BINLOG EVENTS` statement on either the replication master or slave MySQL servers. (See Section 13.7.7.2, “SHOW BINLOG EVENTS Syntax”.)

You can also obtain useful information from the output of `SHOW ENGINE NDB STATUS`.

Note

When performing schema changes on `NDB` tables, applications should wait until the `ALTER TABLE` statement has returned in the MySQL client connection that issued the statement before attempting to use the updated definition of the table.

If the `ndb_apply_status` table does not exist on the slave, `ndb_restore` re-creates it.

Conflict resolution for NDB Cluster Replication requires the presence of an additional `mysql.ndb_replication` table. Currently, this table must be created manually. For information about how to do this, see Section 22.6.11, “NDB Cluster Replication Conflict Resolution”.

22.6.5 Preparing the NDB Cluster for Replication

Preparing the NDB Cluster for replication consists of the following steps:

1. Check all MySQL servers for version compatibility (see Section 22.6.2, “General Requirements for NDB Cluster Replication”).
2. Create a slave account on the master Cluster with the appropriate privileges:

```
mysqlM> GRANT REPLICATION SLAVE
     -> ON *.* TO 'slave_user'@'slave_host'
     -> IDENTIFIED BY 'slave_password';
```

In the previous statement, *`slave_user`* is the slave account user name, *`slave_host`* is the host name or IP address of the replication slave, and *`slave_password`* is the password to assign to this account.

For example, to create a slave user account with the name `myslave`, logging in from the host named `rep-slave`, and using the password `53cr37`, use the following `GRANT` statement:

```
mysqlM> GRANT REPLICATION SLAVE
```

```
    -> ON *.* TO 'myslave'@'rep-slave'
    -> IDENTIFIED BY '53cr37';
```

For security reasons, it is preferable to use a unique user account—not employed for any other purpose—for the replication slave account.

3. Configure the slave to use the master. Using the MySQL Monitor, this can be accomplished with the `CHANGE MASTER TO` statement:

```
mysqlS> CHANGE MASTER TO
    -> MASTER_HOST='master_host',
    -> MASTER_PORT=master_port,
    -> MASTER_USER='slave_user',
    -> MASTER_PASSWORD='slave_password';
```

In the previous statement, *`master_host`* is the host name or IP address of the replication master, *`master_port`* is the port for the slave to use for connecting to the master, *`slave_user`* is the user name set up for the slave on the master, and *`slave_password`* is the password set for that user account in the previous step.

For example, to tell the slave to replicate from the MySQL server whose host name is `rep-master`, using the replication slave account created in the previous step, use the following statement:

```
mysqlS> CHANGE MASTER TO
    -> MASTER_HOST='rep-master',
    -> MASTER_PORT=3306,
    -> MASTER_USER='myslave',
    -> MASTER_PASSWORD='53cr37';
```

For a complete list of options that can be used with this statement, see Section 13.4.2.1, “CHANGE MASTER TO Syntax”.

To provide replication backup capability, you also need to add an `--ndb-connectstring` option to the slave's `my.cnf` file prior to starting the replication process. See Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”, for details.

For additional options that can be set in `my.cnf` for replication slaves, see Section 17.1.6, “Replication and Binary Logging Options and Variables”.

4. If the master cluster is already in use, you can create a backup of the master and load this onto the slave to cut down on the amount of time required for the slave to synchronize itself with the master. If the slave is also running NDB Cluster, this can be accomplished using the backup and restore procedure described in Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”.

```
ndb-connectstring=management_host[:port]
```

In the event that you are *not* using NDB Cluster on the replication slave, you can create a backup with this command on the replication master:

```
shellM> mysqldump --master-data=1
```

Then import the resulting data dump onto the slave by copying the dump file over to the slave. After this, you can use the `mysql` client to import the data from the dumpfile into the slave database as shown here, where *`dump_file`* is the name of the file that was generated using `mysqldump` on the master, and *`db_name`* is the name of the database to be replicated:

```
shellS> mysql -u root -p db_name < dump_file
```

For a complete list of options to use with `mysqldump`, see Section 4.5.4, “`mysqldump` — A Database Backup Program”.

Note

If you copy the data to the slave in this fashion, you should make sure that the slave is started with the `--skip-slave-start` option on the command line, or else include `skip-slave-start` in the slave's `my.cnf` file to keep it from trying to connect to the master to begin replicating before all the data has been loaded. Once the data loading has completed, follow the additional steps outlined in the next two sections.

5. Ensure that each MySQL server acting as a replication master is configured with a unique server ID, and with binary logging enabled, using the row format. (See Section 17.2.1, “Replication Formats”.) These options can be set either in the master server's `my.cnf` file, or on the command line when starting the master `mysqld` process. See Section 22.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”, for information regarding the latter option.

22.6.6 Starting NDB Cluster Replication (Single Replication Channel)

This section outlines the procedure for starting NDB Cluster replication using a single replication channel.

1. Start the MySQL replication master server by issuing this command:

```
shellM> mysqld --ndbcluster --server-id=id \
        --log-bin &
```

In the previous statement, *`id`* is this server's unique ID (see Section 22.6.2, “General Requirements for NDB Cluster Replication”). This starts the server's `mysqld` process with binary logging enabled using the proper logging format.

Note

You can also start the master with `--binlog-format=MIXED`, in which case row-based replication is used automatically when replicating between clusters. `STATEMENT` based binary logging is not supported for NDB Cluster Replication (see Section 22.6.2, “General Requirements for NDB Cluster Replication”).

2. Start the MySQL replication slave server as shown here:

```
shellS> mysqld --ndbcluster --server-id=id &
```

In the command just shown, *`id`* is the slave server's unique ID. It is not necessary to enable logging on the replication slave.

Note

You should use the `--skip-slave-start` option with this command or else you should include `skip-slave-start` in the slave server's `my.cnf` file, unless you want replication to begin immediately. With the use of this option, the start of replication is delayed until the appropriate `START SLAVE` statement has been issued, as explained in Step 4 below.

3. It is necessary to synchronize the slave server with the master server's replication binary log. If binary logging has not previously been running on the master, run the following statement on the slave:

```
mysqlS> CHANGE MASTER TO
     -> MASTER_LOG_FILE='',
     -> MASTER_LOG_POS=4;
```

   This instructs the slave to begin reading the master's binary log from the log's starting point. Otherwise—that is, if you are loading data from the master using a backup—see Section 22.6.8, “Implementing Failover with NDB Cluster Replication”, for information on how to obtain the correct values to use for `MASTER_LOG_FILE` and `MASTER_LOG_POS` in such cases.

4. Finally, you must instruct the slave to begin applying replication by issuing this command from the `mysql` client on the replication slave:

```
mysqlS> START SLAVE;
```

   This also initiates the transmission of replication data from the master to the slave.

It is also possible to use two replication channels, in a manner similar to the procedure described in the next section; the differences between this and using a single replication channel are covered in Section 22.6.7, “Using Two Replication Channels for NDB Cluster Replication”.

It is also possible to improve cluster replication performance by enabling *batched updates*. This can be accomplished by setting the `slave_allow_batching` system variable on the slave `mysqld` processes. Normally, updates are applied as soon as they are received. However, the use of batching causes updates to be applied in 32 KB batches, which can result in higher throughput and less CPU usage, particularly where individual updates are relatively small.

Note

Slave batching works on a per-epoch basis; updates belonging to more than one transaction can be sent as part of the same batch.

All outstanding updates are applied when the end of an epoch is reached, even if the updates total less than 32 KB.

Batching can be turned on and off at runtime. To activate it at runtime, you can use either of these two statements:

```
SET GLOBAL slave_allow_batching = 1;
SET GLOBAL slave_allow_batching = ON;
```

If a particular batch causes problems (such as a statement whose effects do not appear to be replicated correctly), slave batching can be deactivated using either of the following statements:

```
SET GLOBAL slave_allow_batching = 0;
SET GLOBAL slave_allow_batching = OFF;
```

You can check whether slave batching is currently being used by means of an appropriate `SHOW VARIABLES` statement, like this one:

```
mysql> SHOW VARIABLES LIKE 'slave%';
+---------------------------+-------+
| Variable_name             | Value |
```

```
+--------------------------+-------+
| slave_allow_batching     | ON    |
| slave_compressed_protocol | OFF   |
| slave_load_tmpdir        | /tmp  |
| slave_net_timeout        | 3600  |
| slave_skip_errors        | OFF   |
| slave_transaction_retries | 10    |
+--------------------------+-------+
6 rows in set (0.00 sec)
```

22.6.7 Using Two Replication Channels for NDB Cluster Replication

In a more complete example scenario, we envision two replication channels to provide redundancy and thereby guard against possible failure of a single replication channel. This requires a total of four replication servers, two masters for the master cluster and two slave servers for the slave cluster. For purposes of the discussion that follows, we assume that unique identifiers are assigned as shown here:

Table 22.416 NDB Cluster replication servers described in the text

Server ID	Description
1	Master - primary replication channel (*M*)
2	Master - secondary replication channel (*M'*)
3	Slave - primary replication channel (*S*)
4	Slave - secondary replication channel (*S'*)

Setting up replication with two channels is not radically different from setting up a single replication channel. First, the `mysqld` processes for the primary and secondary replication masters must be started, followed by those for the primary and secondary slaves. Then the replication processes may be initiated by issuing the `START SLAVE` statement on each of the slaves. The commands and the order in which they need to be issued are shown here:

1. Start the primary replication master:

   ```
   shellM> mysqld --ndbcluster --server-id=1 \
                  --log-bin &
   ```

2. Start the secondary replication master:

   ```
   shellM'> mysqld --ndbcluster --server-id=2 \
                 --log-bin &
   ```

3. Start the primary replication slave server:

   ```
   shellS> mysqld --ndbcluster --server-id=3 \
                  --skip-slave-start &
   ```

4. Start the secondary replication slave:

   ```
   shellS'> mysqld --ndbcluster --server-id=4 \
                   --skip-slave-start &
   ```

5. Finally, initiate replication on the primary channel by executing the `START SLAVE` statement on the primary slave as shown here:

   ```
   mysqlS> START SLAVE;
   ```

Warning

Only the primary channel is to be started at this point. The secondary replication channel is to be started only in the event that the primary replication channel fails, as described in Section 22.6.8, “Implementing Failover with NDB Cluster Replication”. Running multiple replication channels simultaneously can result in unwanted duplicate records being created on the replication slaves.

As mentioned previously, it is not necessary to enable binary logging on replication slaves.

22.6.8 Implementing Failover with NDB Cluster Replication

In the event that the primary Cluster replication process fails, it is possible to switch over to the secondary replication channel. The following procedure describes the steps required to accomplish this.

1. Obtain the time of the most recent global checkpoint (GCP). That is, you need to determine the most recent epoch from the `ndb_apply_status` table on the slave cluster, which can be found using the following query:

```
mysqlS'> SELECT @latest:=MAX(epoch)
      ->        FROM mysql.ndb_apply_status;
```

In a circular replication topology, with a master and a slave running on each host, when you are using `ndb_log_apply_status=1`, NDB Cluster epochs are written in the slave binary logs. This means that the `ndb_apply_status` table contains information for the slave on this host as well as for any other host which acts as a slave of the master running on this host.

In this case, you need to determine the latest epoch on this slave to the exclusion of any epochs from any other slaves in this slave's binary log that were not listed in the `IGNORE_SERVER_IDS` options of the `CHANGE MASTER TO` statement used to set up this slave. The reason for excluding such epochs is that rows in the `mysql.ndb_apply_status` table whose server IDs have a match in the `IGNORE_SERVER_IDS` list used with the CHANGE MASTER TO statement used to prepare this slave's master are also considered to be from local servers, in addition to those having the slave's own server ID. You can retrieve this list as `Replicate_Ignore_Server_Ids` from the output of `SHOW SLAVE STATUS`. We assume that you have obtained this list and are substituting it for *`ignore_server_ids`* in the query shown here, which like the previous version of the query, selects the greatest epoch into a variable named `@latest`:

```
mysqlS'> SELECT @latest:=MAX(epoch)
      ->        FROM mysql.ndb_apply_status
      ->        WHERE server_id NOT IN (ignore_server_ids);
```

In some cases, it may be simpler or more efficient (or both) to use a list of the server IDs to be included and `server_id IN` *`server_id_list`* in the `WHERE` condition of the preceding query.

2. Using the information obtained from the query shown in Step 1, obtain the corresponding records from the `ndb_binlog_index` table on the master cluster.

You can use the following query to obtain the needed records from the master's `ndb_binlog_index` table:

```
mysqlM'> SELECT
      ->     @file:=SUBSTRING_INDEX(next_file, '/', -1),
      ->     @pos:=next_position
      -> FROM mysql.ndb_binlog_index
```

```
      -> WHERE epoch >= @latest
      -> ORDER BY epoch ASC LIMIT 1;
```

These are the records saved on the master since the failure of the primary replication channel. We have employed a user variable `@latest` here to represent the value obtained in Step 1. Of course, it is not possible for one `mysqld` instance to access user variables set on another server instance directly. These values must be "plugged in" to the second query manually or in application code.

Important

You must ensure that the slave `mysqld` is started with `--slave-skip-errors=ddl_exist_errors` before executing `START SLAVE`. Otherwise, replication may stop with duplicate DDL errors.

3. Now it is possible to synchronize the secondary channel by running the following query on the secondary slave server:

```
mysqlS'> CHANGE MASTER TO
      ->     MASTER_LOG_FILE='@file',
      ->     MASTER_LOG_POS=@pos;
```

Again we have employed user variables (in this case `@file` and `@pos`) to represent the values obtained in Step 2 and applied in Step 3; in practice these values must be inserted manually or using application code that can access both of the servers involved.

Note

`@file` is a string value such as `'/var/log/mysql/replication-master-bin.00001'`, and so must be quoted when used in SQL or application code. However, the value represented by `@pos` must *not* be quoted. Although MySQL normally attempts to convert strings to numbers, this case is an exception.

4. You can now initiate replication on the secondary channel by issuing the appropriate command on the secondary slave `mysqld`:

```
mysqlS'> START SLAVE;
```

Once the secondary replication channel is active, you can investigate the failure of the primary and effect repairs. The precise actions required to do this will depend upon the reasons for which the primary channel failed.

Warning

The secondary replication channel is to be started only if and when the primary replication channel has failed. Running multiple replication channels simultaneously can result in unwanted duplicate records being created on the replication slaves.

If the failure is limited to a single server, it should (in theory) be possible to replicate from *M* to *S'*, or from *M'* to *S*; however, this has not yet been tested.

22.6.9 NDB Cluster Backups With NDB Cluster Replication

This section discusses making backups and restoring from them using NDB Cluster replication. We assume that the replication servers have already been configured as covered previously (see Section 22.6.5, "Preparing the NDB Cluster for Replication", and the sections immediately following). This having been done, the procedure for making a backup and then restoring from it is as follows:

1. There are two different methods by which the backup may be started.

   - **Method A.** This method requires that the cluster backup process was previously enabled on the master server, prior to starting the replication process. This can be done by including the following line in a `[mysql_cluster]` section in the `my.cnf file`, where *`management_host`* is the IP address or host name of the `NDB` management server for the master cluster, and *`port`* is the management server's port number:

     ```
     ndb-connectstring=management_host[:port]
     ```

     Note

     The port number needs to be specified only if the default port (1186) is not being used. See Section 22.2.4, “Initial Configuration of NDB Cluster”, for more information about ports and port allocation in NDB Cluster.

     In this case, the backup can be started by executing this statement on the replication master:

     ```
     shellM> ndb_mgm -e "START BACKUP"
     ```

   - **Method B.** If the `my.cnf` file does not specify where to find the management host, you can start the backup process by passing this information to the `NDB` management client as part of the `START BACKUP` command. This can be done as shown here, where *`management_host`* and *`port`* are the host name and port number of the management server:

     ```
     shellM> ndb_mgm management_host:port -e "START BACKUP"
     ```

     In our scenario as outlined earlier (see Section 22.6.5, “Preparing the NDB Cluster for Replication”), this would be executed as follows:

     ```
     shellM> ndb_mgm rep-master:1186 -e "START BACKUP"
     ```

2. Copy the cluster backup files to the slave that is being brought on line. Each system running an `ndbd` process for the master cluster will have cluster backup files located on it, and *all* of these files must be copied to the slave to ensure a successful restore. The backup files can be copied into any directory on the computer where the slave management host resides, so long as the MySQL and NDB binaries have read permissions in that directory. In this case, we will assume that these files have been copied into the directory `/var/BACKUPS/BACKUP-1`.

   It is not necessary that the slave cluster have the same number of `ndbd` processes (data nodes) as the master; however, it is highly recommended this number be the same. It *is* necessary that the slave be started with the `--skip-slave-start` option, to prevent premature startup of the replication process.

3. Create any databases on the slave cluster that are present on the master cluster that are to be replicated to the slave.

   Important

   A `CREATE DATABASE` (or `CREATE SCHEMA`) statement corresponding to each database to be replicated must be executed on each SQL node in the slave cluster.

4. Reset the slave cluster using this statement in the MySQL Monitor:

```
mysqlS> RESET SLAVE;
```

5. You can now start the cluster restoration process on the replication slave using the `ndb_restore` command for each backup file in turn. For the first of these, it is necessary to include the `-m` option to restore the cluster metadata:

```
shellS> ndb_restore -c slave_host:port -n node-id \
        -b backup-id -m -r dir
```

   `dir` is the path to the directory where the backup files have been placed on the replication slave. For the `ndb_restore` commands corresponding to the remaining backup files, the `-m` option should *not* be used.

   For restoring from a master cluster with four data nodes (as shown in the figure in Section 22.6, "NDB Cluster Replication") where the backup files have been copied to the directory `/var/BACKUPS/BACKUP-1`, the proper sequence of commands to be executed on the slave might look like this:

```
shellS> ndb_restore -c rep-slave:1186 -n 2 -b 1 -m \
        -r ./var/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 3 -b 1 \
        -r ./var/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 4 -b 1 \
        -r ./var/BACKUPS/BACKUP-1
shellS> ndb_restore -c rep-slave:1186 -n 5 -b 1 -e \
        -r ./var/BACKUPS/BACKUP-1
```

Important

The `-e` (or `--restore-epoch`) option in the final invocation of `ndb_restore` in this example is required in order that the epoch is written to the slave `mysql.ndb_apply_status`. Without this information, the slave will not be able to synchronize properly with the master. (See Section 22.4.23, "`ndb_restore` — Restore an NDB Cluster Backup".)

6. Now you need to obtain the most recent epoch from the `ndb_apply_status` table on the slave (as discussed in Section 22.6.8, "Implementing Failover with NDB Cluster Replication"):

```
mysqlS> SELECT @latest:=MAX(epoch)
        FROM mysql.ndb_apply_status;
```

7. Using `@latest` as the epoch value obtained in the previous step, you can obtain the correct starting position `@pos` in the correct binary log file `@file` from the master's `mysql.ndb_binlog_index` table using the query shown here:

```
mysqlM> SELECT
     ->     @file:=SUBSTRING_INDEX(File, '/', -1),
     ->     @pos:=Position
     -> FROM mysql.ndb_binlog_index
     -> WHERE epoch >= @latest
     -> ORDER BY epoch ASC LIMIT 1;
```

   In the event that there is currently no replication traffic, you can get this information by running `SHOW MASTER STATUS` on the master and using the value in the `Position` column for the file whose name has the suffix with the greatest value for all files shown in the `File` column. However, in this case, you must determine this and supply it in the next step manually or by parsing the output with a script.

8. Using the values obtained in the previous step, you can now issue the appropriate `CHANGE MASTER TO` statement in the slave's `mysql` client:

```
mysqlS> CHANGE MASTER TO
     ->     MASTER_LOG_FILE='@file',
     ->     MASTER_LOG_POS=@pos;
```

9. Now that the slave “knows” from what point in which binary log file to start reading data from the master, you can cause the slave to begin replicating with this standard MySQL statement:

```
mysqlS> START SLAVE;
```

To perform a backup and restore on a second replication channel, it is necessary only to repeat these steps, substituting the host names and IDs of the secondary master and slave for those of the primary master and slave replication servers where appropriate, and running the preceding statements on them.

For additional information on performing Cluster backups and restoring Cluster from backups, see Section 22.5.3, “Online Backup of NDB Cluster”.

22.6.9.1 NDB Cluster Replication: Automating Synchronization of the Replication Slave to the Master Binary Log

It is possible to automate much of the process described in the previous section (see Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”). The following Perl script `reset-slave.pl` serves as an example of how you can do this.

```
#!/user/bin/perl -w

#  file: reset-slave.pl

#  Copyright ©2005 MySQL AB

#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.

#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.

#  You should have received a copy of the GNU General Public License
#  along with this program; if not, write to:
#  Free Software Foundation, Inc.
#  59 Temple Place, Suite 330
#  Boston, MA 02111-1307 USA
#
#  Version 1.1


######################### Includes ###################################

use DBI;

######################### Globals ####################################

my  $m_host='';
my  $m_port='';
my  $m_user='';
my  $m_pass='';
```

```
my  $s_host='';
my  $s_port='';
my  $s_user='';
my  $s_pass='';
my  $dbhM='';
my  $dbhS='';

######################## Sub Prototypes ##########################

sub CollectCommandPromptInfo;
sub ConnectToDatabases;
sub DisconnectFromDatabases;
sub GetSlaveEpoch;
sub GetMasterInfo;
sub UpdateSlave;

######################## Program Main ############################

CollectCommandPromptInfo;
ConnectToDatabases;
GetSlaveEpoch;
GetMasterInfo;
UpdateSlave;
DisconnectFromDatabases;

################## Collect Command Prompt Info ##################

sub CollectCommandPromptInfo
{
  ### Check that user has supplied correct number of command line args
  die "Usage:\n
       reset-slave >master MySQL host< >master MySQL port< \n
                   >master user< >master pass< >slave MySQL host< \n
                   >slave MySQL port< >slave user< >slave pass< \n
       All 8 arguments must be passed. Use BLANK for NULL passwords\n"
       unless @ARGV == 8;

  $m_host  =  $ARGV[0];
  $m_port  =  $ARGV[1];
  $m_user  =  $ARGV[2];
  $m_pass  =  $ARGV[3];
  $s_host  =  $ARGV[4];
  $s_port  =  $ARGV[5];
  $s_user  =  $ARGV[6];
  $s_pass  =  $ARGV[7];

  if ($m_pass eq "BLANK") { $m_pass = '';}
  if ($s_pass eq "BLANK") { $s_pass = '';}
}

###############  Make connections to both databases #############

sub ConnectToDatabases
{
  ### Connect to both master and slave cluster databases

  ### Connect to master
  $dbhM
    = DBI->connect(
    "dbi:mysql:database=mysql;host=$m_host;port=$m_port",
    "$m_user", "$m_pass")
      or die "Can't connect to Master Cluster MySQL process!
              Error: $DBI::errstr\n";

  ### Connect to slave
  $dbhS
    = DBI->connect(
```

```
          "dbi:mysql:database=mysql;host=$s_host",
          "$s_user", "$s_pass")
    or die "Can't connect to Slave Cluster MySQL process!
            Error: $DBI::errstr\n";
}

################  Disconnect from both databases ################

sub DisconnectFromDatabases
{
  ### Disconnect from master

  $dbhM->disconnect
  or warn " Disconnection failed: $DBI::errstr\n";

  ### Disconnect from slave

  $dbhS->disconnect
  or warn " Disconnection failed: $DBI::errstr\n";
}

######################  Find the last good GCI ##################

sub GetSlaveEpoch
{
  $sth = $dbhS->prepare("SELECT MAX(epoch)
                         FROM mysql.ndb_apply_status;")
      or die "Error while preparing to select epoch from slave: ",
             $dbhS->errstr;

  $sth->execute
      or die "Selecting epoch from slave error: ", $sth->errstr;

  $sth->bind_col (1, \$epoch);
  $sth->fetch;
  print "\tSlave Epoch =  $epoch\n";
  $sth->finish;
}

#######  Find the position of the last GCI in the binary log ########

sub GetMasterInfo
{
  $sth = $dbhM->prepare("SELECT
                           SUBSTRING_INDEX(File, '/', -1), Position
                         FROM mysql.ndb_binlog_index
                         WHERE epoch > $epoch
                         ORDER BY epoch ASC LIMIT 1;")
      or die "Prepare to select from master error: ", $dbhM->errstr;

  $sth->execute
      or die "Selecting from master error: ", $sth->errstr;

  $sth->bind_col (1, \$binlog);
  $sth->bind_col (2, \$binpos);
  $sth->fetch;
  print "\tMaster binary log =  $binlog\n";
  print "\tMaster binary log position =  $binpos\n";
  $sth->finish;
}

##########  Set the slave to process from that location #########

sub UpdateSlave
{
  $sth = $dbhS->prepare("CHANGE MASTER TO
                         MASTER_LOG_FILE='$binlog',
```

```
                          MASTER_LOG_POS=$binpos;")
      or die "Prepare to CHANGE MASTER error: ", $dbhS->errstr;

  $sth->execute
       or die "CHANGE MASTER on slave error: ", $sth->errstr;
  $sth->finish;
  print "\tSlave has been updated. You may now start the slave.\n";
}

# end reset-slave.pl
```

22.6.9.2 Point-In-Time Recovery Using NDB Cluster Replication

Point-in-time recovery—that is, recovery of data changes made since a given point in time—is performed after restoring a full backup that returns the server to its state when the backup was made. Performing point-in-time recovery of NDB Cluster tables with NDB Cluster and NDB Cluster Replication can be accomplished using a native `NDB` data backup (taken by issuing `CREATE BACKUP` in the `ndb_mgm` client) and restoring the `ndb_binlog_index` table (from a dump made using `mysqldump`).

To perform point-in-time recovery of NDB Cluster, it is necessary to follow the steps shown here:

1. Back up all `NDB` databases in the cluster, using the `START BACKUP` command in the `ndb_mgm` client (see Section 22.5.3, “Online Backup of NDB Cluster”).

2. At some later point, prior to restoring the cluster, make a backup of the `mysql.ndb_binlog_index` table. It is probably simplest to use `mysqldump` for this task. Also back up the binary log files at this time.

   This backup should be updated regularly—perhaps even hourly—depending on your needs.

3. (*Catastrophic failure or error occurs*.)

4. Locate the last known good backup.

5. Clear the data node file systems (using `ndbd --initial` or `ndbmtd --initial`).

   Note

   NDB Cluster Disk Data tablespace and log files are not removed by `--initial`. You must delete these manually.

6. Use `DROP TABLE` or `TRUNCATE TABLE` with the `mysql.ndb_binlog_index` table.

7. Execute `ndb_restore`, restoring all data. You must include the `--restore-epoch` option when you run `ndb_restore`, so that the `ndb_apply_status` table is populated correctly. (See Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”, for more information.)

8. Restore the `ndb_binlog_index` table from the output of `mysqldump` and restore the binary log files from backup, if necessary.

9. Find the epoch applied most recently—that is, the maximum `epoch` column value in the `ndb_apply_status` table—as the user variable `@LATEST_EPOCH` (emphasized):

   ```
   SELECT @LATEST_EPOCH:=MAX(epoch)
       FROM mysql.ndb_apply_status;
   ```

10. Find the latest binary log file (`@FIRST_FILE`) and position (`Position` column value) within this file that correspond to `@LATEST_EPOCH` in the `ndb_binlog_index` table:

```
SELECT Position, @FIRST_FILE:=File
    FROM mysql.ndb_binlog_index
    WHERE epoch > @LATEST_EPOCH ORDER BY epoch ASC LIMIT 1;
```

11. Using `mysqlbinlog`, replay the binary log events from the given file and position up to the point of the failure. (See Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.)

See also Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”, for more information about the binary log, replication, and incremental recovery.

22.6.10 NDB Cluster Replication: Multi-Master and Circular Replication

It is possible to use NDB Cluster in multi-master replication, including circular replication between a number of NDB Clusters.

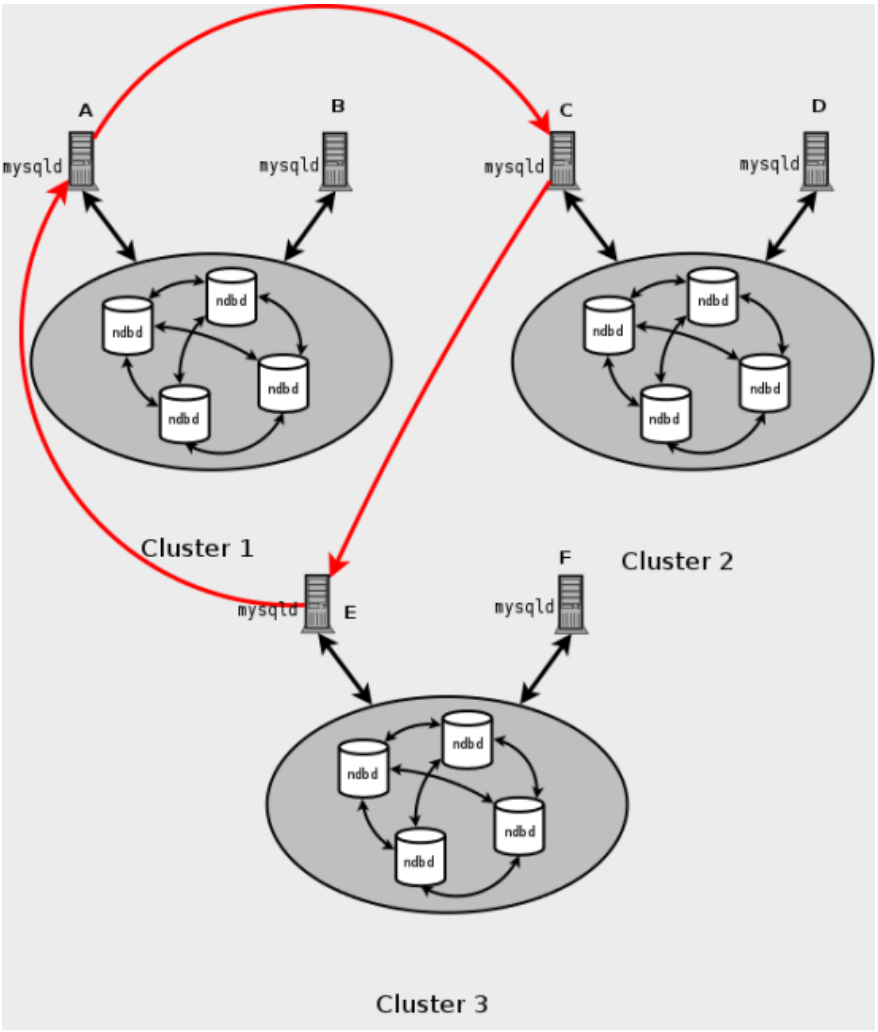
Circular replication example. In the next few paragraphs we consider the example of a replication setup involving three NDB Clusters numbered 1, 2, and 3, in which Cluster 1 acts as the replication master for Cluster 2, Cluster 2 acts as the master for Cluster 3, and Cluster 3 acts as the master for Cluster 1. Each cluster has two SQL nodes, with SQL nodes A and B belonging to Cluster 1, SQL nodes C and D belonging to Cluster 2, and SQL nodes E and F belonging to Cluster 3.

Circular replication using these clusters is supported as long as the following conditions are met:

- The SQL nodes on all masters and slaves are the same.
- All SQL nodes acting as replication masters and slaves are started with the `log_slave_updates` system variable enabled.

This type of circular replication setup is shown in the following diagram:

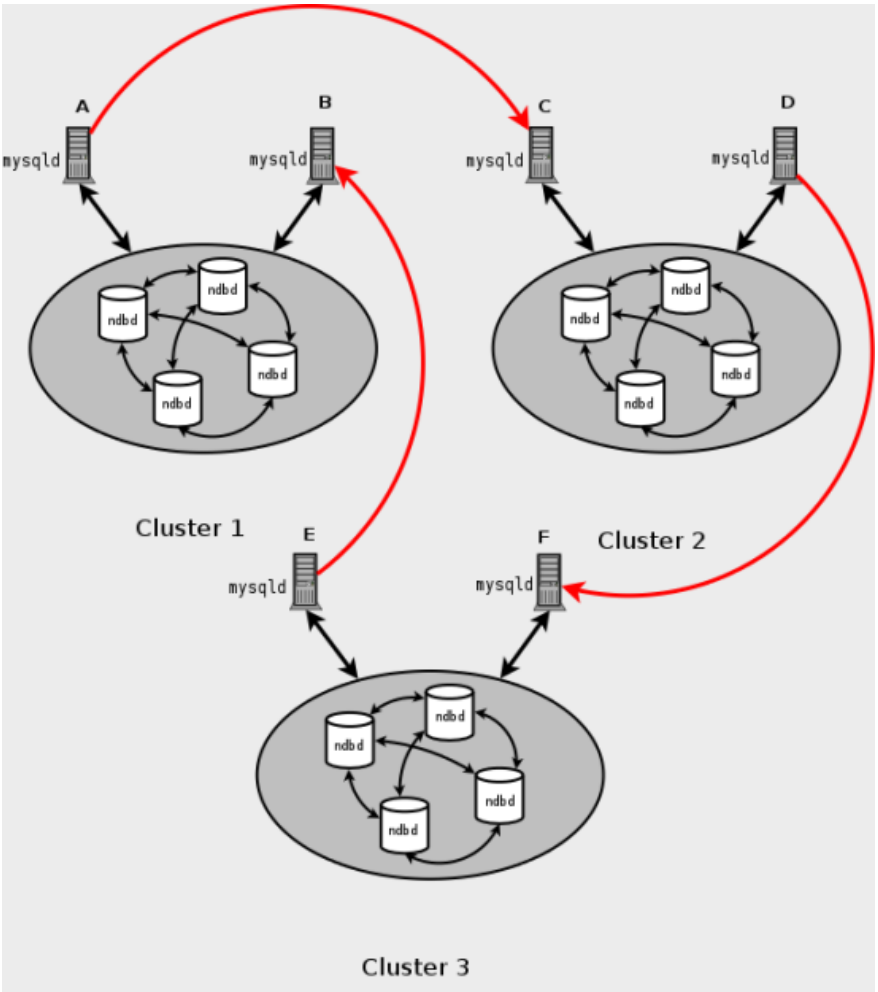
Figure 22.35 NDB Cluster Circular Replication with All Masters As Slaves



In this scenario, SQL node A in Cluster 1 replicates to SQL node C in Cluster 2; SQL node C replicates to SQL node E in Cluster 3; SQL node E replicates to SQL node A. In other words, the replication line (indicated by the curved arrows in the diagram) directly connects all SQL nodes used as replication masters and slaves.

It is also possible to set up circular replication in such a way that not all master SQL nodes are also slaves, as shown here:

Figure 22.36 NDB Cluster Circular Replication Where Not All Masters Are Slaves



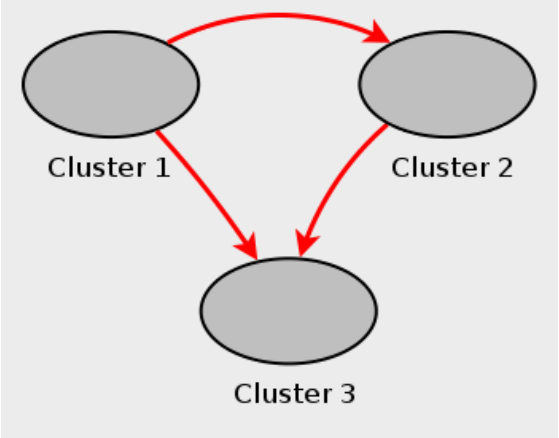
In this case, different SQL nodes in each cluster are used as replication masters and slaves. However, you must *not* start any of the SQL nodes with the `log_slave_updates` system variable enabled. This type of circular replication scheme for NDB Cluster, in which the line of replication (again indicated by the curved arrows in the diagram) is discontinuous, should be possible, but it should be noted that it has not yet been thoroughly tested and must therefore still be considered experimental.

Using NDB-native backup and restore to initialize a slave NDB Cluster. When setting up circular replication, it is possible to initialize the slave cluster by using the management client `BACKUP` command on one NDB Cluster to create a backup and then applying this backup on another NDB Cluster using `ndb_restore`. However, this does not automatically create binary logs on the second NDB Cluster 's SQL node acting as the replication slave. In order to cause the binary logs to be created, you must issue a `SHOW TABLES` statement on that SQL node; this should be done prior to running `START SLAVE`.

This is a known issue which we intend to address in a future release.

Multi-master failover example. In this section, we discuss failover in a multi-master NDB Cluster replication setup with three NDB Clusters having server IDs 1, 2, and 3. In this scenario, Cluster 1 replicates to Clusters 2 and 3; Cluster 2 also replicates to Cluster 3. This relationship is shown here:

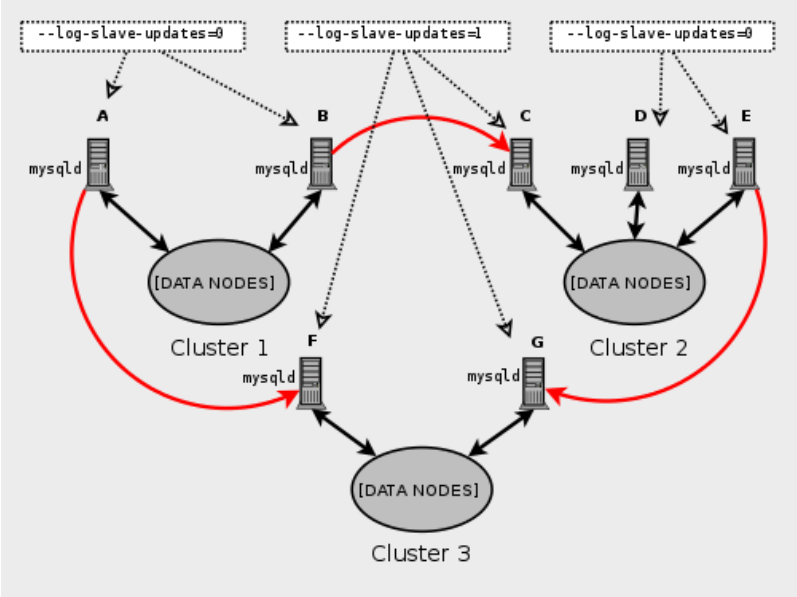
Figure 22.37 NDB Cluster Multi-Master Replication With 3 Masters



In other words, data replicates from Cluster 1 to Cluster 3 through 2 different routes: directly, and by way of Cluster 2.

Not all MySQL servers taking part in multi-master replication must act as both master and slave, and a given NDB Cluster might use different SQL nodes for different replication channels. Such a case is shown here:

Figure 22.38 NDB Cluster Multi-Master Replication, With MySQL Servers



MySQL servers acting as replication slaves must be run with the `log_slave_updates` system variable enabled. Which `mysqld` processes require this option is also shown in the preceding diagram.

Note

Using the `log_slave_updates` system variable has no effect on servers not being run as replication slaves.

The need for failover arises when one of the replicating clusters goes down. In this example, we consider the case where Cluster 1 is lost to service, and so Cluster 3 loses 2 sources of updates from Cluster 1. Because replication between NDB Clusters is asynchronous, there is no guarantee that Cluster 3's updates

originating directly from Cluster 1 are more recent than those received through Cluster 2. You can handle this by ensuring that Cluster 3 catches up to Cluster 2 with regard to updates from Cluster 1. In terms of MySQL servers, this means that you need to replicate any outstanding updates from MySQL server C to server F.

On server C, perform the following queries:

```
mysqlC> SELECT @latest:=MAX(epoch)
     ->     FROM mysql.ndb_apply_status
     ->     WHERE server_id=1;

mysqlC> SELECT
     ->     @file:=SUBSTRING_INDEX(File, '/', -1),
     ->     @pos:=Position
     ->     FROM mysql.ndb_binlog_index
     ->     WHERE orig_epoch >= @latest
     ->     AND orig_server_id = 1
     ->     ORDER BY epoch ASC LIMIT 1;
```

Note

You can improve the performance of this query, and thus likely speed up failover times significantly, by adding the appropriate index to the `ndb_binlog_index` table. See Section 22.6.4, “NDB Cluster Replication Schema and Tables”, for more information.

Copy over the values for *`@file`* and *`@pos`* manually from server C to server F (or have your application perform the equivalent). Then, on server F, execute the following `CHANGE MASTER TO` statement:

```
mysqlF> CHANGE MASTER TO
     ->     MASTER_HOST = 'serverC'
     ->     MASTER_LOG_FILE='@file',
     ->     MASTER_LOG_POS=@pos;
```

Once this has been done, you can issue a `START SLAVE` statement on MySQL server F, and any missing updates originating from server B will be replicated to server F.

The `CHANGE MASTER TO` statement also supports an `IGNORE_SERVER_IDS` option which takes a comma-separated list of server IDs and causes events originating from the corresponding servers to be ignored. For more information, see Section 13.4.2.1, “CHANGE MASTER TO Syntax”, and Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”. For information about how this option intereacts with the `ndb_log_apply_status` variable, see Section 22.6.8, “Implementing Failover with NDB Cluster Replication”.

22.6.11 NDB Cluster Replication Conflict Resolution

When using a replication setup involving multiple masters (including circular replication), it is possible that different masters may try to update the same row on the slave with different data. Conflict resolution in NDB Cluster Replication provides a means of resolving such conflicts by permitting a user-defined resolution column to be used to determine whether or not an update on a given master should be applied on the slave.

Some types of conflict resolution supported by NDB Cluster (`NDB$OLD()`, `NDB$MAX()`, `NDB$MAX_DELETE_WIN()`) implement this user-defined column as a “timestamp” column (although its type cannot be `TIMESTAMP`, as explained later in this section). These types of conflict resolution are always applied a row-by-row basis rather than a transactional basis. The epoch-based conflict resolution functions `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` compare the order in which epochs are replicated (and thus these functions are transactional). Different methods can be used to compare resolution column values on

the slave when conflicts occur, as explained later in this section; the method used can be set on a per-table basis.

You should also keep in mind that it is the application's responsibility to ensure that the resolution column is correctly populated with relevant values, so that the resolution function can make the appropriate choice when determining whether to apply an update.

Requirements. Preparations for conflict resolution must be made on both the master and the slave. These tasks are described in the following list:

- On the master writing the binary logs, you must determine which columns are sent (all columns or only those that have been updated). This is done for the MySQL Server as a whole by applying the `mysqld` startup option `--ndb-log-updated-only` (described later in this section) or on a per-table basis by entries in the `mysql.ndb_replication` table (see The ndb_replication system table).

  Note

  If you are replicating tables with very large columns (such as `TEXT` or `BLOB` columns), `--ndb-log-updated-only` can also be useful for reducing the size of the master and slave binary logs and avoiding possible replication failures due to exceeding `max_allowed_packet`.

  See Section 17.5.1.20, “Replication and max_allowed_packet”, for more information about this issue.

- On the slave, you must determine which type of conflict resolution to apply (“latest timestamp wins”, “same timestamp wins”, “primary wins”, “primary wins, complete transaction”, or none). This is done using the `mysql.ndb_replication` system table, on a per-table basis (see The ndb_replication system table).

- NDB Cluster also supports read conflict detection, that is, detecting conflicts between reads of a given row in one cluster and updates or deletes of the same row in another cluster. This requires exclusive read locks obtained by setting `ndb_log_exclusive_reads` equal to 1 on the slave. All rows read by a conflicting read are logged in the exceptions table. For more information, see Read conflict detection and resolution.

When using the functions `NDB$OLD()`, `NDB$MAX()`, and `NDB$MAX_DELETE_WIN()` for timestamp-based conflict resolution, we often refer to the column used for determining updates as a “timestamp” column. However, the data type of this column is never `TIMESTAMP`; instead, its data type should be `INT` (`INTEGER`) or `BIGINT`. The “timestamp” column should also be `UNSIGNED` and `NOT NULL`.

The `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` functions discussed later in this section work by comparing the relative order of replication epochs applied on a primary and secondary NDB Cluster, and do not make use of timestamps.

Master column control. We can see update operations in terms of “before” and “after” images—that is, the states of the table before and after the update is applied. Normally, when updating a table with a primary key, the “before” image is not of great interest; however, when we need to determine on a per-update basis whether or not to use the updated values on a replication slave, we need to make sure that both images are written to the master's binary log. This is done with the `--ndb-log-update-as-write` option for `mysqld`, as described later in this section.

Important

Whether logging of complete rows or of updated columns only is done is decided when the MySQL server is started, and cannot be changed online; you must either restart `mysqld`, or start a new `mysqld` instance with different logging options.

Logging Full or Partial Rows (--ndb-log-updated-only Option)

Property	Value
Command-Line Format	`--ndb-log-updated-only[={OFF|ON}]`
System Variable	`ndb_log_updated_only`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

For purposes of conflict resolution, there are two basic methods of logging rows, as determined by the setting of the `--ndb-log-updated-only` option for `mysqld`:

- Log complete rows
- Log only column data that has been updated—that is, column data whose value has been set, regardless of whether or not this value was actually changed. This is the default behavior.

It is usually sufficient—and more efficient—to log updated columns only; however, if you need to log full rows, you can do so by setting `--ndb-log-updated-only` to `0` or `OFF`.

--ndb-log-update-as-write Option: Logging Changed Data as Updates

Property	Value
Command-Line Format	`--ndb-log-update-as-write[={OFF|ON}]`
System Variable	`ndb_log_update_as_write`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

The setting of the MySQL Server's `--ndb-log-update-as-write` option determines whether logging is performed with or without the “before” image. Because conflict resolution is done in the MySQL Server's update handler, it is necessary to control logging on the master such that updates are updates and not writes; that is, such that updates are treated as changes in existing rows rather than the writing of new rows (even though these replace existing rows). This option is turned on by default; in other words, updates are treated as writes. (That is, updates are by default written as `write_row` events in the binary log, rather than as `update_row` events.)

To turn off the option, start the master `mysqld` with `--ndb-log-update-as-write=0` or `--ndb-log-update-as-write=OFF`. You must do this when replicating from NDB tables to tables using a different storage engine; see Replication from NDB to other storage engines, and Replication from NDB to a nontransactional storage engine, for more information.

Conflict resolution control. Conflict resolution is usually enabled on the server where conflicts can occur. Like logging method selection, it is enabled by entries in the `mysql.ndb_replication` table.

The ndb_replication system table. To enable conflict resolution, it is necessary to create an `ndb_replication` table in the `mysql` system database on the master, the slave, or both, depending on the conflict resolution type and method to be employed. This table is used to control logging and conflict resolution functions on a per-table basis, and has one row per table involved in replication. `ndb_replication` is created and filled with control information on the server where the conflict is to be resolved. In a simple master-slave setup where data can also be changed locally on the slave this will typically be the slave. In a more complex master-master (2-way) replication schema this will usually be all of the masters involved. Each row in `mysql.ndb_replication` corresponds to a table being replicated, and specifies how to log and resolve conflicts (that is, which conflict resolution function, if any, to use) for that table. The definition of the `mysql.ndb_replication` table is shown here:

```
CREATE TABLE mysql.ndb_replication  (
    db VARBINARY(63),
    table_name VARBINARY(63),
    server_id INT UNSIGNED,
    binlog_type INT UNSIGNED,
    conflict_fn VARBINARY(128),
    PRIMARY KEY USING HASH (db, table_name, server_id)
)   ENGINE=NDB
PARTITION BY KEY(db,table_name);
```

The columns in this table are described in the next few paragraphs.

db. The name of the database containing the table to be replicated. You may employ either or both of the wildcards `_` and `%` as part of the database name. Matching is similar to what is implemented for the `LIKE` operator.

table_name. The name of the table to be replicated. The table name may include either or both of the wildcards `_` and `%`. Matching is similar to what is implemented for the `LIKE` operator.

server_id. The unique server ID of the MySQL instance (SQL node) where the table resides.

binlog_type. The type of binary logging to be employed. This is determined as shown in the following table:

Table 22.417 binlog_type values, with internal values and descriptions

Value	Internal Value	Description
0	`NBT_DEFAULT`	Use server default
1	`NBT_NO_LOGGING`	Do not log this table in the binary log
2	`NBT_UPDATED_ONLY`	Only updated attributes are logged
3	`NBT_FULL`	Log full row, even if not updated (MySQL server default behavior)
4	`NBT_USE_UPDATE`	(For generating `NBT_UPDATED_ONLY_USE_UPDATE` and `NBT_FULL_USE_UPDATE` values only—not intended for separate use)
5	[*Not used*]	---
6	`NBT_UPDATED_ONLY_USE_UPDATE` (equal to `NBT_UPDATED_ONLY` | `NBT_USE_UPDATE`)	Use updated attributes, even if values are unchanged
7	`NBT_FULL_USE_UPDATE` (equal to	Use full row, even if values are unchanged

Value	Internal Value	Description
	NBT_FULL | NBT_USE_UPDATE)	

conflict_fn. The conflict resolution function to be applied. This function must be specified as one of those shown in the following list:

- NDB$OLD(column_name)
- NDB$MAX(column_name)
- NDB$MAX_DELETE_WIN()
- NDB$EPOCH() and NDB$EPOCH_TRANS()
- NDB$EPOCH_TRANS()
- NDB$EPOCH2()
- NDB$EPOCH2_TRANS()
- NULL: Indicates that conflict resolution is not to be used for the corresponding table.

These functions are described in the next few paragraphs.

NDB$OLD(column_name). If the value of *column_name* is the same on both the master and the slave, then the update is applied; otherwise, the update is not applied on the slave and an exception is written to the log. This is illustrated by the following pseudocode:

```
if (master_old_column_value == slave_current_column_value)
  apply_update();
else
  log_exception();
```

This function can be used for “same value wins” conflict resolution. This type of conflict resolution ensures that updates are not applied on the slave from the wrong master.

Important

The column value from the master's “before” image is used by this function.

NDB$MAX(column_name). If the “timestamp” column value for a given row coming from the master is higher than that on the slave, it is applied; otherwise it is not applied on the slave. This is illustrated by the following pseudocode:

```
if (master_new_column_value > slave_current_column_value)
  apply_update();
```

This function can be used for “greatest timestamp wins” conflict resolution. This type of conflict resolution ensures that, in the event of a conflict, the version of the row that was most recently updated is the version that persists.

Important

The column value from the master's “after” image is used by this function.

NDB$MAX_DELETE_WIN(). This is a variation on `NDB$MAX()`. Due to the fact that no timestamp is available for a delete operation, a delete using `NDB$MAX()` is in fact processed as `NDB$OLD`. However, for some use cases, this is not optimal. For `NDB$MAX_DELETE_WIN()`, if the “timestamp” column value for a given row adding or updating an existing row coming from the master is higher than that on the slave, it is applied. However, delete operations are treated as always having the higher value. This is illustrated in the following pseudocode:

```
if ( (master_new_column_value > slave_current_column_value)
       ||
     operation.type == "delete")
  apply_update();
```

This function can be used for “greatest timestamp, delete wins” conflict resolution. This type of conflict resolution ensures that, in the event of a conflict, the version of the row that was deleted or (otherwise) most recently updated is the version that persists.

Note

As with `NDB$MAX()`, the column value from the master's “after” image is the value used by this function.

NDB$EPOCH() and NDB$EPOCH_TRANS(). The `NDB$EPOCH()` function tracks the order in which replicated epochs are applied on a slave NDB Cluster relative to changes originating on the slave. This relative ordering is used to determine whether changes originating on the slave are concurrent with any changes that originate locally, and are therefore potentially in conflict.

Most of what follows in the description of `NDB$EPOCH()` also applies to `NDB$EPOCH_TRANS()`. Any exceptions are noted in the text.

`NDB$EPOCH()` is asymmetric, operating on one NDB Cluster in a two-cluster circular replication configuration (sometimes referred to as “active-active” replication). We refer here to cluster on which it operates as the primary, and the other as the secondary. The slave on the primary is responsible for detecting and handling conflicts, while the slave on the secondary is not involved in any conflict detection or handling.

When the slave on the primary detects conflicts, it injects events into its own binary log to compensate for these; this ensures that the secondary NDB Cluster eventually realigns itself with the primary and so keeps the primary and secondary from diverging. This compensation and realignment mechanism requires that the primary NDB Cluster always wins any conflicts with the secondary—that is, that the primary's changes are always used rather than those from the secondary in event of a conflict. This “primary always wins” rule has the following implications:

- Operations that change data, once committed on the primary, are fully persistent and will not be undone or rolled back by conflict detection and resolution.
- Data read from the primary is fully consistent. Any changes committed on the Primary (locally or from the slave) will not be reverted later.
- Operations that change data on the secondary may later be reverted if the primary determines that they are in conflict.
- Individual rows read on the secondary are self-consistent at all times, each row always reflecting either a state committed by the secondary, or one committed by the primary.
- Sets of rows read on the secondary may not necessarily be consistent at a given single point in time. For `NDB$EPOCH_TRANS()`, this is a transient state; for `NDB$EPOCH()`, it can be a persistent state.

- Assuming a period of sufficient length without any conflicts, all data on the secondary NDB Cluster (eventually) becomes consistent with the primary's data.

`NDB$EPOCH()` and `NDB$EPOCH_TRANS()` do not require any user schema modifications, or application changes to provide conflict detection. However, careful thought must be given to the schema used, and the access patterns used, to verify that the complete system behaves within specified limits.

Each of the `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` functions can take an optional parameter; this is the number of bits to use to represent the lower 32 bits of the epoch, and should be set to no less than

```
CEIL( LOG2( TimeBetweenGlobalCheckpoints / TimeBetweenEpochs ), 1)
```

For the default values of these configuration parameters (2000 and 100 milliseconds, respectively), this gives a value of 5 bits, so the default value (6) should be sufficient, unless other values are used for `TimeBetweenGlobalCheckpoints`, `TimeBetweenEpochs`, or both. A value that is too small can result in false positives, while one that is too large could lead to excessive wasted space in the database.

Both `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` insert entries for conflicting rows into the relevant exceptions tables, provided that these tables have been defined according to the same exceptions table schema rules as described elsewhere in this section (see NDB$OLD(column_name)). You need to create any exceptions table before creating the table with which it is to be used.

As with the other conflict detection functions discussed in this section, `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` are activated by including relevant entries in the `mysql.ndb_replication` table (see The ndb_replication system table). The roles of the primary and secondary NDB Clusters in this scenario are fully determined by `mysql.ndb_replication` table entries.

Because the conflict detection algorithms employed by `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` are asymmetric, you must use different values for the primary slave's and secondary slave's `server_id` entries.

A conflict between `DELETE` operations alone is not sufficient to trigger a conflict using `NDB$EPOCH()` or `NDB$EPOCH_TRANS()`, and the relative placement within epochs does not matter. (Bug #18459944)

Conflict detection status variables. Several status variables can be used to monitor conflict detection. You can see how many rows have been found in conflict by `NDB$EPOCH()` since this slave was last restarted from the current value of the `Ndb_conflict_fn_epoch` system status variable.

`Ndb_conflict_fn_epoch_trans` provides the number of rows that have been found directly in conflict by `NDB$EPOCH_TRANS()`. `Ndb_conflict_fn_epoch2` and `Ndb_conflict_fn_epoch2_trans` show the number of rows found in conflict by `NDB$EPOCH2()` and `NDB$EPOCH2_TRANS()`, respectively. The number of rows actually realigned, including those affected due to their membership in or dependency on the same transactions as other conflicting rows, is given by `Ndb_conflict_trans_row_reject_count`.

For more information, see NDB Cluster Status Variables.

Limitations on NDB$EPOCH(). The following limitations currently apply when using `NDB$EPOCH()` to perform conflict detection:

- Conflicts are detected using NDB Cluster epoch boundaries, with granularity proportional to `TimeBetweenEpochs` (default: 100 milliseconds). The minimum conflict window is the minimum time during which concurrent updates to the same data on both clusters always report a conflict. This is always a nonzero length of time, and is roughly proportional to `2 * (latency + queueing + TimeBetweenEpochs)`. This implies that—assuming the default for `TimeBetweenEpochs` and ignoring any latency between clusters (as well as any queuing delays)—the minimum conflict window

size is approximately 200 milliseconds. This minimum window should be considered when looking at expected application “race” patterns.

- Additional storage is required for tables using the `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` functions; from 1 to 32 bits extra space per row is required, depending on the value passed to the function.
- Conflicts between delete operations may result in divergence between the primary and secondary. When a row is deleted on both clusters concurrently, the conflict can be detected, but is not recorded, since the row is deleted. This means that further conflicts during the propagation of any subsequent realignment operations will not be detected, which can lead to divergence.

  Deletes should be externally serialized, or routed to one cluster only. Alternatively, a separate row should be updated transactionally with such deletes and any inserts that follow them, so that conflicts can be tracked across row deletes. This may require changes in applications.
- Only two NDB Clusters in a circular “active-active” configuration are currently supported when using `NDB$EPOCH()` or `NDB$EPOCH_TRANS()` for conflict detection.
- Tables having `BLOB` or `TEXT` columns are not currently supported with `NDB$EPOCH()` or `NDB$EPOCH_TRANS()`.

NDB$EPOCH_TRANS(). `NDB$EPOCH_TRANS()` extends the `NDB$EPOCH()` function. Conflicts are detected and handled in the same way using the “primary wins all” rule (see NDB$EPOCH() and NDB $EPOCH_TRANS()) but with the extra condition that any other rows updated in the same transaction in which the conflict occurred are also regarded as being in conflict. In other words, where `NDB$EPOCH()` realigns individual conflicting rows on the secondary, `NDB$EPOCH_TRANS()` realigns conflicting transactions.

In addition, any transactions which are detectably dependent on a conflicting transaction are also regarded as being in conflict, these dependencies being determined by the contents of the secondary cluster's binary log. Since the binary log contains only data modification operations (inserts, updates, and deletes), only overlapping data modifications are used to determine dependencies between transactions.

`NDB$EPOCH_TRANS()` is subject to the same conditions and limitations as `NDB$EPOCH()`, and in addition requires that all transaction IDs are recorded in the secondary's binary log (the `--ndb-log-transaction-id` option), which adds a variable overhead (up to 13 bytes per row). The deprecated `log_bin_use_v1_row_events` system variable, which defaults to `OFF`, must not be set to `ON` with `NDB$EPOCH_TRANS()`.

See NDB$EPOCH() and NDB$EPOCH_TRANS().

Status information. A server status variable `Ndb_conflict_fn_max` provides a count of the number of times that a row was not applied on the current SQL node due to “greatest timestamp wins” conflict resolution since the last time that `mysqld` was started.

The number of times that a row was not applied as the result of “same timestamp wins” conflict resolution on a given `mysqld` since the last time it was restarted is given by the global status variable `Ndb_conflict_fn_old`. In addition to incrementing `Ndb_conflict_fn_old`, the primary key of the row that was not used is inserted into an *exceptions table*, as explained later in this section.

NDB$EPOCH2(). The `NDB$EPOCH2()` function is similar to `NDB$EPOCH()`, except that `NDB$EPOCH2()` provides for delete-delete handling with a circular replication (“master-master”) topology. In this scenario, primary and secondary roles are assigned to the two masters by setting the `ndb_slave_conflict_role` system variable to the appropriate value on each master (usually one each of `PRIMARY`, `SECONDARY`). When this is done, modifications made by the secondary are reflected by the primary back to the secondary which then conditionally applies them.

NDB$EPOCH2_TRANS(). `NDB$EPOCH2_TRANS()` extends the `NDB$EPOCH2()` function. Conflicts are detected and handled in the same way, and assigning primary and secondary roles to the replicating clusters, but with the extra condition that any other rows updated in the same transaction in which the conflict occurred are also regarded as being in conflict. That is, `NDB$EPOCH2()` realigns individual conflicting rows on the secondary, while `NDB$EPOCH_TRANS()` realigns conflicting transactions.

Where `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` use metadata that is specified per row, per last modified epoch, to determine on the primary whether an incoming replicated row change from the secondary is concurrent with a locally committed change; concurrent changes are regarded as conflicting, with subesequent exceptions table updates and realignment of the secondary. A problem arises when a row is deleted on the primary so there is no longer any last-modified epoch available to determine whether any replicated operations conflict, which means that conflicting delete operationss are not detected. This can result in divergence, an example being a delete on one cluster which is concurrent with a delete and insert on the other; this why delete operations can be routed to only one cluster when using `NDB$EPOCH()` and `NDB$EPOCH_TRANS()`.

`NDB$EPOCH2()` bypasses the issue just described—storing information about deleted rows on the PRIMARY—by ignoring any delete-delete conflict, and by avoiding any potential resultant divergence as well. This is accomplished by reflecting any operation successfully applied on and replicated from the secondary back to the secondary. On its return to the secondary, it can be used to reapply an operation on the secondary which was deleted by an operation originating from the primary.

When using `NDB$EPOCH2()`, you should keep in mind that the secondary applies the delete from the primary, removing the new row until it is restored by a reflected operation. In theory, the subsequent insert or update on the secondary conflicts with the delete from the primary, but in this case, we choose to ignore this and allow the secondary to “win”, in the interest of preventing divergence between the clusters. In other words, after a delete, the primary does not detect conflicts, and instead adopts the secondary's following changes immediately. Because of this, the secondary's state can revisit multiple previous committed states as it progresses to a final (stable) state, and some of these may be visible.

You should also be aware that reflecting all operations from the secondary back to the primary increases the size of the primary's logbinary log, as well as demands on bandwidth, CPU usage, and disk I/O.

Application of reflected operations on the secondary depends on the state of the target row on the secondary. Whether or not reflected changes are applied on the secondary can be tracked by checking the `Ndb_conflict_reflected_op_prepare_count` and `Ndb_conflict_reflected_op_discard_count` status variables. The number of changes applied is simply the difference between these two values (note that `Ndb_conflict_reflected_op_prepare_count` is always greater than or equal to `Ndb_conflict_reflected_op_discard_count`).

Events are applied if and only if both of the following conditions are true:

- The existence of the row—that is, whether or not it exists—is in accordance with the type of event. For delete and update operations, the row must already exist. For insert operations, the row must *not* exist.
- The row was last modified by the primary. It is possible that the modification was accomplished through the execution of a reflected operation.

If both of the conditions are not met, the reflected operation is discarded by the secondary.

Conflict resolution exceptions table. To use the `NDB$OLD()` conflict resolution function, it is also necessary to create an exceptions table corresponding to each `NDB` table for which this type of conflict resolution is to be employed. This is also true when using `NDB$EPOCH()` or `NDB$EPOCH_TRANS()`. The name of this table is that of the table for which conflict resolution is to be applied, with the string `$EX` appended. (For example, if the name of the original table is `mytable`, the name of the corresponding

exceptions table name should be `mytable$EX`.) The syntax for creating the exceptions table is as shown here:

```
CREATE TABLE original_table$EX  (
    [NDB$]server_id INT UNSIGNED,
    [NDB$]master_server_id INT UNSIGNED,
    [NDB$]master_epoch BIGINT UNSIGNED,
    [NDB$]count INT UNSIGNED,

    [NDB$OP_TYPE ENUM('WRITE_ROW','UPDATE_ROW', 'DELETE_ROW',
      'REFRESH_ROW', 'READ_ROW') NOT NULL,]
    [NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST', 'ROW_ALREADY_EXISTS',
      'DATA_IN_CONFLICT', 'TRANS_IN_CONFLICT') NOT NULL,]
    [NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL,]

    original_table_pk_columns,

    [orig_table_column|orig_table_column$OLD|orig_table_column$NEW,]

    [additional_columns,]

    PRIMARY KEY([NDB$]server_id, [NDB$]master_server_id, [NDB$]master_epoch, [NDB$]count)
) ENGINE=NDB;
```

The first four columns are required. The names of the first four columns and the columns matching the original table's primary key columns are not critical; however, we suggest for reasons of clarity and consistency, that you use the names shown here for the `server_id`, `master_server_id`, `master_epoch`, and `count` columns, and that you use the same names as in the original table for the columns matching those in the original table's primary key.

If the exceptions table uses one or more of the optional columns `NDB$OP_TYPE`, `NDB$CFT_CAUSE`, or `NDB$ORIG_TRANSID` discussed later in this section, then each of the required columns must also be named using the prefix `NDB$`. If desired, you can use the `NDB$` prefix to name the required columns even if you do not define any optional columns, but in this case, all four of the required columns must be named using the prefix.

Following these columns, the columns making up the original table's primary key should be copied in the order in which they are used to define the primary key of the original table. The data types for the columns duplicating the primary key columns of the original table should be the same as (or larger than) those of the original columns. A subset of the primary key columns may be used.

Regardless of the NDB Cluster version employed, the exceptions table must use the `NDB` storage engine. (An example that uses `NDB$OLD()` with an exceptions table is shown later in this section.)

Additional columns may optionally be defined following the copied primary key columns, but not before any of them; any such extra columns cannot be `NOT NULL`. NDB Cluster supports three additional, predefined optional columns `NDB$OP_TYPE`, `NDB$CFT_CAUSE`, and `NDB$ORIG_TRANSID`, which are described in the next few paragraphs.

`NDB$OP_TYPE`: This column can be used to obtain the type of operation causing the conflict. If you use this column, define it as shown here:

```
NDB$OP_TYPE ENUM('WRITE_ROW', 'UPDATE_ROW', 'DELETE_ROW',
    'REFRESH_ROW', 'READ_ROW') NOT NULL
```

The `WRITE_ROW`, `UPDATE_ROW`, and `DELETE_ROW` operation types represent user-initiated operations. `REFRESH_ROW` operations are operations generated by conflict resolution in compensating transactions sent back to the originating cluster from the cluster that detected the conflict. `READ_ROW` operations are user-initiated read tracking operations defined with exclusive row locks.

`NDB$CFT_CAUSE`: You can define an optional column `NDB$CFT_CAUSE` which provides the cause of the registered conflict. This column, if used, is defined as shown here:

```
NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST', 'ROW_ALREADY_EXISTS',
    'DATA_IN_CONFLICT', 'TRANS_IN_CONFLICT') NOT NULL
```

`ROW_DOES_NOT_EXIST` can be reported as the cause for `UPDATE_ROW` and `WRITE_ROW` operations; `ROW_ALREADY_EXISTS` can be reported for `WRITE_ROW` events. `DATA_IN_CONFLICT` is reported when a row-based conflict function detects a conflict; `TRANS_IN_CONFLICT` is reported when a transactional conflict function rejects all of the operations belonging to a complete transaction.

`NDB$ORIG_TRANSID`: The `NDB$ORIG_TRANSID` column, if used, contains the ID of the originating transaction. This column should be defined as follows:

```
NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL
```

`NDB$ORIG_TRANSID` is a 64-bit value generated by `NDB`. This value can be used to correlate multiple exceptions table entries belonging to the same conflicting transaction from the same or different exceptions tables.

Additional reference columns which are not part of the original table's primary key can be named *`colname`*`$OLD` or *`colname`*`$NEW`. *`colname`*`$OLD` references old values in update and delete operations—that is, operations containing `DELETE_ROW` events. *`colname`*`$NEW` can be used to reference new values in insert and update operations—in other words, operations using `WRITE_ROW` events, `UPDATE_ROW` events, or both types of events. Where a conflicting operation does not supply a value for a given non-primary-key reference column, the exceptions table row contains either `NULL`, or a defined default value for that column.

Important

The `mysql.ndb_replication` table is read when a data table is set up for replication, so the row corresponding to a table to be replicated must be inserted into `mysql.ndb_replication` *before* the table to be replicated is created.

Examples

The following examples assume that you have already a working NDB Cluster replication setup, as described in Section 22.6.5, “Preparing the NDB Cluster for Replication”, and Section 22.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”.

NDB$MAX() example. Suppose you wish to enable “greatest timestamp wins” conflict resolution on table `test.t1`, using column `mycol` as the “timestamp”. This can be done using the following steps:

1. Make sure that you have started the master `mysqld` with `--ndb-log-update-as-write=OFF`.
2. On the master, perform this `INSERT` statement:

   ```
   INSERT INTO mysql.ndb_replication
       VALUES ('test', 't1', 0, NULL, 'NDB$MAX(mycol)');
   ```

   Inserting a 0 into the `server_id` indicates that all SQL nodes accessing this table should use conflict resolution. If you want to use conflict resolution on a specific `mysqld` only, use the actual server ID.

   Inserting `NULL` into the `binlog_type` column has the same effect as inserting 0 (`NBT_DEFAULT`); the server default is used.
3. Create the `test.t1` table:

```
CREATE TABLE test.t1 (
    columns
    mycol INT UNSIGNED,
    columns
) ENGINE=NDB;
```

Now, when updates are done on this table, conflict resolution is applied, and the version of the row having the greatest value for `mycol` is written to the slave.

Note

Other `binlog_type` options—such as `NBT_UPDATED_ONLY_USE_UPDATE` should be used to control logging on the master using the `ndb_replication` table rather than by using command-line options.

NDB$OLD() example. Suppose an `NDB` table such as the one defined here is being replicated, and you wish to enable “same timestamp wins” conflict resolution for updates to this table:

```
CREATE TABLE test.t2  (
    a INT UNSIGNED NOT NULL,
    b CHAR(25) NOT NULL,
    columns,
    mycol INT UNSIGNED NOT NULL,
    columns,
    PRIMARY KEY pk (a, b)
)   ENGINE=NDB;
```

The following steps are required, in the order shown:

1. First—and *prior* to creating `test.t2`—you must insert a row into the `mysql.ndb_replication` table, as shown here:

   ```
   INSERT INTO mysql.ndb_replication
       VALUES ('test', 't2', 0, NULL, 'NDB$OLD(mycol)');
   ```

   Possible values for the `binlog_type` column are shown earlier in this section. The value `'NDB$OLD(mycol)'` should be inserted into the `conflict_fn` column.

2. Create an appropriate exceptions table for `test.t2`. The table creation statement shown here includes all required columns; any additional columns must be declared following these columns, and before the definition of the table's primary key.

   ```
   CREATE TABLE test.t2$EX  (
       server_id INT UNSIGNED,
       master_server_id INT UNSIGNED,
       master_epoch BIGINT UNSIGNED,
       count INT UNSIGNED,
       a INT UNSIGNED NOT NULL,
       b CHAR(25) NOT NULL,

       [additional_columns,]

       PRIMARY KEY(server_id, master_server_id, master_epoch, count)
   )   ENGINE=NDB;
   ```

   We can include additional columns for information about the type, cause, and originating transaction ID for a given conflict. We are also not required to supply matching columns for all primary key columns in the original table. This means you can create the exceptions table like this:

```
CREATE TABLE test.t2$EX  (
    NDB$server_id INT UNSIGNED,
    NDB$master_server_id INT UNSIGNED,
    NDB$master_epoch BIGINT UNSIGNED,
    NDB$count INT UNSIGNED,
    a INT UNSIGNED NOT NULL,

    NDB$OP_TYPE ENUM('WRITE_ROW','UPDATE_ROW', 'DELETE_ROW',
      'REFRESH_ROW', 'READ_ROW') NOT NULL,
    NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST', 'ROW_ALREADY_EXISTS',
      'DATA_IN_CONFLICT', 'TRANS_IN_CONFLICT') NOT NULL,
    NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL,

    [additional_columns,]

    PRIMARY KEY(NDB$server_id, NDB$master_server_id, NDB$master_epoch, NDB$count)
)   ENGINE=NDB;
```

Note

The `NDB$` prefix is required for the four required columns since we included at least one of the columns `NDB$OP_TYPE`, `NDB$CFT_CAUSE`, or `NDB$ORIG_TRANSID` in the table definition.

3. Create the table `test.t2` as shown previously.

These steps must be followed for every table for which you wish to perform conflict resolution using `NDB$OLD()`. For each such table, there must be a corresponding row in `mysql.ndb_replication`, and there must be an exceptions table in the same database as the table being replicated.

Read conflict detection and resolution. NDB Cluster also supports tracking of read operations, which makes it possible in circular replication setups to manage conflicts between reads of a given row in one cluster and updates or deletes of the same row in another. This example uses `employee` and `department` tables to model a scenario in which an employee is moved from one department to another on the master cluster (which we refer to hereafter as cluster *A*) while the slave cluster (hereafter *B*) updates the employee count of the employee's former department in an interleaved transaction.

The data tables have been created using the following SQL statements:

```
# Employee table
CREATE TABLE employee (
    id INT PRIMARY KEY,
    name VARCHAR(2000),
    dept INT NOT NULL
)   ENGINE=NDB;

# Department table
CREATE TABLE department (
    id INT PRIMARY KEY,
    name VARCHAR(2000),
    members INT
)   ENGINE=NDB;
```

The contents of the two tables include the rows shown in the (partial) output of the following `SELECT` statements:

```
mysql> SELECT id, name, dept FROM employee;
+---------------+------+
| id   | name   | dept |
```

```
+------+--------+------+
...
| 998  |  Mike  | 3    |
| 999  |  Joe   | 3    |
| 1000 |  Mary  | 3    |
...
+------+--------+------+

mysql> SELECT id, name, members FROM department;
+-----+-------------+---------+
| id  | name        | members |
+-----+-------------+---------+
...
| 3   | Old project | 24      |
...
+-----+-------------+---------+
```

We assume that we are already using an exceptions table that includes the four required columns (and these are used for this table's primary key), the optional columns for operation type and cause, and the original table's primary key column, created using the SQL statement shown here:

```
CREATE TABLE employee$EX  (
    NDB$server_id INT UNSIGNED,
    NDB$master_server_id INT UNSIGNED,
    NDB$master_epoch BIGINT UNSIGNED,
    NDB$count INT UNSIGNED,

    NDB$OP_TYPE ENUM( 'WRITE_ROW','UPDATE_ROW', 'DELETE_ROW',
                      'REFRESH_ROW','READ_ROW') NOT NULL,
    NDB$CFT_CAUSE ENUM( 'ROW_DOES_NOT_EXIST',
                        'ROW_ALREADY_EXISTS',
                        'DATA_IN_CONFLICT',
                        'TRANS_IN_CONFLICT') NOT NULL,

    id INT NOT NULL,

    PRIMARY KEY(NDB$server_id, NDB$master_server_id, NDB$master_epoch, NDB$count)
)   ENGINE=NDB;
```

Suppose there occur the two simultaneous transactions on the two clusters. On cluster *A*, we create a new department, then move employee number 999 into that department, using the following SQL statements:

```
BEGIN;
  INSERT INTO department VALUES (4, "New project", 1);
  UPDATE employee SET dept = 4 WHERE id = 999;
COMMIT;
```

At the same time, on cluster *B*, another transaction reads from `employee`, as shown here:

```
BEGIN;
  SELECT name FROM employee WHERE id = 999;
  UPDATE department SET members = members - 1  WHERE id = 3;
commit;
```

The conflicting transactions are not normally detected by the conflict resolution mechanism, since the conflict is between a read (`SELECT`) and an update operation. You can circumvent this issue by executing `SET ndb_log_exclusive_reads = 1` on the slave cluster. Acquiring exclusive read locks in this way causes any rows read on the master to be flagged as needing conflict resolution on the slave cluster. If we enable exclusive reads in this way prior to the logging of these transactions, the read on cluster *B* is tracked and sent to cluster *A* for resolution; the conflict on the employee row will be detected and the transaction on cluster *B* is aborted.

The conflict is registered in the exceptions table (on cluster *A*) as a `READ_ROW` operation (see Conflict resolution exceptions table, for a description of operation types), as shown here:

```
mysql> SELECT id, NDB$OP_TYPE, NDB$CFT_CAUSE FROM employee$EX;
+-------+-------------+-------------------+
| id    | NDB$OP_TYPE | NDB$CFT_CAUSE     |
+-------+-------------+-------------------+
...
| 999   | READ_ROW    | TRANS_IN_CONFLICT |
+-------+-------------+-------------------+
```

Any existing rows found in the read operation are flagged. This means that multiple rows resulting from the same conflict may be logged in the exception table, as shown by examining the effects a conflict between an update on cluster *A* and a read of multiple rows on cluster *B* from the same table in simultaneous transactions. The transaction executed on cluster *A* is shown here:

```
BEGIN;
  INSERT INTO department VALUES (4, "New project", 0);
  UPDATE employee SET dept = 4 WHERE dept = 3;
  SELECT COUNT(*) INTO @count FROM employee WHERE dept = 4;
  UPDATE department SET members = @count WHERE id = 4;
COMMIT;
```

Concurrently a transaction containing the statements shown here runs on cluster *B*:

```
SET ndb_log_exclusive_reads = 1;  # Must be set if not already enabled
...
BEGIN;
  SELECT COUNT(*) INTO @count FROM employee WHERE dept = 3 FOR UPDATE;
  UPDATE department SET members = @count WHERE id = 3;
COMMIT;
```

In this case, all three rows matching the `WHERE` condition in the second transaction's `SELECT` are read, and are thus flagged in the exceptions table, as shown here:

```
mysql> SELECT id, NDB$OP_TYPE, NDB$CFT_CAUSE FROM employee$EX;
+-------+-------------+-------------------+
| id    | NDB$OP_TYPE | NDB$CFT_CAUSE     |
+-------+-------------+-------------------+
...
| 998   | READ_ROW    | TRANS_IN_CONFLICT |
| 999   | READ_ROW    | TRANS_IN_CONFLICT |
| 1000  | READ_ROW    | TRANS_IN_CONFLICT |
...
+-------+-------------+-------------------+
```

Read tracking is performed on the basis of existing rows only. A read based on a given condition track conflicts only of any rows that are *found* and not of any rows that are inserted in an interleaved transaction. This is similar to how exclusive row locking is performed in a single instance of NDB Cluster.

22.7 NDB Cluster Release Notes

Changes in NDB Cluster releases are documented separately from this reference manual; you can find release notes for the changes in each NDB Cluster 8.0 release at NDB 8.0 Release Notes, and for each NDB Cluster 7.6 release at NDB 7.6 Release Notes.

You can obtain release notes for older versions of NDB Cluster from NDB Cluster Release Notes.

Chapter 23 Partitioning

Table of Contents

23.1 Overview of Partitioning in MySQL 4258
23.2 Partitioning Types 4261
23.2.1 RANGE Partitioning 4263
23.2.2 LIST Partitioning 4267
23.2.3 COLUMNS Partitioning 4269
23.2.4 HASH Partitioning 4278
23.2.5 KEY Partitioning 4281
23.2.6 Subpartitioning 4283
23.2.7 How MySQL Partitioning Handles NULL 4284
23.3 Partition Management 4289
23.3.1 Management of RANGE and LIST Partitions 4290
23.3.2 Management of HASH and KEY Partitions 4297
23.3.3 Exchanging Partitions and Subpartitions with Tables 4298
23.3.4 Maintenance of Partitions 4306
23.3.5 Obtaining Information About Partitions 4307
23.4 Partition Pruning 4309
23.5 Partition Selection 4313
23.6 Restrictions and Limitations on Partitioning 4319
23.6.1 Partitioning Keys, Primary Keys, and Unique Keys 4324
23.6.2 Partitioning Limitations Relating to Storage Engines 4328
23.6.3 Partitioning Limitations Relating to Functions 4329

This chapter discusses *user-defined partitioning*.

Note

Table partitioning differs from partitioning as used by window functions. For information about window functions, see Section 12.21, “Window Functions”.

In MySQL 8.0, partitioning support is provided by the `InnoDB` and `NDB` storage engines.

MySQL 8.0 does not currently support partitioning of tables using any storage engine other than `InnoDB` or `NDB`, such as `MyISAM`. An attempt to create a partitioned tables using a storage engine that does not supply native partitioning support fails with `ER_CHECK_NOT_IMPLEMENTED`.

MySQL 8.0 Community binaries provided by Oracle include partitioning support provided by the `InnoDB` and `NDB` storage engines. For information about partitioning support offered in MySQL Enterprise Edition binaries, see Chapter 30, *MySQL Enterprise Edition*.

If you are compiling MySQL 8.0 from source, configuring the build with `InnoDB` support is sufficient to produce binaries with partition support for `InnoDB` tables. For more information, see Section 2.9, “Installing MySQL from Source”.

Nothing further needs to be done to enable partitioning support by `InnoDB` (for example, no special entries are required in the `my.cnf` file).

It is not possible to disable partitioning support by the `InnoDB` storage engine.

See Section 23.1, “Overview of Partitioning in MySQL”, for an introduction to partitioning and partitioning concepts.

Several types of partitioning are supported, as well as subpartitioning; see Section 23.2, “Partitioning Types”, and Section 23.2.6, “Subpartitioning”.

Section 23.3, “Partition Management”, covers methods of adding, removing, and altering partitions in existing partitioned tables.

Section 23.3.4, “Maintenance of Partitions”, discusses table maintenance commands for use with partitioned tables.

The `PARTITIONS` table in the `INFORMATION_SCHEMA` database provides information about partitions and partitioned tables. See Section 25.17, “The INFORMATION_SCHEMA PARTITIONS Table”, for more information; for some examples of queries against this table, see Section 23.2.7, “How MySQL Partitioning Handles NULL”.

For known issues with partitioning in MySQL 8.0, see Section 23.6, “Restrictions and Limitations on Partitioning”.

You may also find the following resources to be useful when working with partitioned tables.

Additional Resources. Other sources of information about user-defined partitioning in MySQL include the following:

- MySQL Partitioning Forum

  This is the official discussion forum for those interested in or experimenting with MySQL Partitioning technology. It features announcements and updates from MySQL developers and others. It is monitored by members of the Partitioning Development and Documentation Teams.

- Mikael Ronström's Blog

  MySQL Partitioning Architect and Lead Developer Mikael Ronström frequently posts articles here concerning his work with MySQL Partitioning and NDB Cluster.

- PlanetMySQL

  A MySQL news site featuring MySQL-related blogs, which should be of interest to anyone using my MySQL. We encourage you to check here for links to blogs kept by those working with MySQL Partitioning, or to have your own blog added to those covered.

23.1 Overview of Partitioning in MySQL

This section provides a conceptual overview of partitioning in MySQL 8.0.

For information on partitioning restrictions and feature limitations, see Section 23.6, “Restrictions and Limitations on Partitioning”.

The SQL standard does not provide much in the way of guidance regarding the physical aspects of data storage. The SQL language itself is intended to work independently of any data structures or media underlying the schemas, tables, rows, or columns with which it works. Nonetheless, most advanced database management systems have evolved some means of determining the physical location to be used for storing specific pieces of data in terms of the file system, hardware or even both. In MySQL, the `InnoDB` storage engine has long supported the notion of a tablespace (see Section 15.6.3,

"Tablespaces"), and the MySQL Server, even prior to the introduction of partitioning, could be configured to employ different physical directories for storing different databases (see Section 8.12.2, "Using Symbolic Links", for an explanation of how this is done).

Partitioning takes this notion a step further, by enabling you to distribute portions of individual tables across a file system according to rules which you can set largely as needed. In effect, different portions of a table are stored as separate tables in different locations. The user-selected rule by which the division of data is accomplished is known as a *partitioning function*, which in MySQL can be the modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function. The function is selected according to the partitioning type specified by the user, and takes as its parameter the value of a user-supplied expression. This expression can be a column value, a function acting on one or more column values, or a set of one or more column values, depending on the type of partitioning that is used.

In the case of `RANGE`, `LIST`, and [`LINEAR`] `HASH` partitioning, the value of the partitioning column is passed to the partitioning function, which returns an integer value representing the number of the partition in which that particular record should be stored. This function must be nonconstant and nonrandom. It may not contain any queries, but may use an SQL expression that is valid in MySQL, as long as that expression returns either `NULL` or an integer *`intval`* such that

```
-MAXVALUE <= intval <= MAXVALUE
```

(`MAXVALUE` is used to represent the least upper bound for the type of integer in question. `-MAXVALUE` represents the greatest lower bound.)

For [`LINEAR`] `KEY`, `RANGE COLUMNS`, and `LIST COLUMNS` partitioning, the partitioning expression consists of a list of one or more columns.

For [`LINEAR`] `KEY` partitioning, the partitioning function is supplied by MySQL.

For more information about permitted partitioning column types and partitioning functions, see Section 23.2, "Partitioning Types", as well as Section 13.1.20, "CREATE TABLE Syntax", which provides partitioning syntax descriptions and additional examples. For information about restrictions on partitioning functions, see Section 23.6.3, "Partitioning Limitations Relating to Functions".

This is known as *horizontal partitioning*—that is, different rows of a table may be assigned to different physical partitions. MySQL 8.0 does not support *vertical partitioning*, in which different columns of a table are assigned to different physical partitions. There are no plans at this time to introduce vertical partitioning into MySQL.

For creating partitioned tables, you must use a storage engine that supports them. In MySQL 8.0, all partitions of the same partitioned table must use the same storage engine. However, there is nothing preventing you from using different storage engines for different partitioned tables on the same MySQL server or even in the same database.

In MySQL 8.0, the only storage engines that support partitioning are `InnoDB` and `NDB`. Partitioning cannot be used with storage engines that do not support it; these include the `MyISAM`, `MERGE`, `CSV`, and `FEDERATED` storage engines.

Partitioning by `KEY` or `LINEAR KEY` is possible with `NDB`, but other types of user-defined partitioning are not supported for tables using this storage engine. In addition, an `NDB` table that employs user-defined partitioning must have an explicit primary key, and any columns referenced in the table's partitioning expression must be part of the primary key. However, if no columns are listed in the `PARTITION BY KEY` or `PARTITION BY LINEAR KEY` clause of the `CREATE TABLE` or `ALTER TABLE` statement used to create or modify a user-partitioned `NDB` table, then the table is not required to have an explicit primary key. For more information, see Section 22.1.7.1, "Noncompliance with SQL Syntax in NDB Cluster".

When creating a partitioned table, the default storage engine is used just as when creating any other table; to override this behavior, it is necessary only to use the `[STORAGE] ENGINE` option just as you would for a table that is not partitioned. The target storage engine must provide native partitioning support, or the statement fails. You should keep in mind that `[STORAGE] ENGINE` (and other table options) need to be listed *before* any partitioning options are used in a `CREATE TABLE` statement. This example shows how to create a table that is partitioned by hash into 6 partitions and which uses the `InnoDB` storage engine (regardless of the value of `default_storage_engine`):

```
CREATE TABLE ti (id INT, amount DECIMAL(7,2), tr_date DATE)
    ENGINE=INNODB
    PARTITION BY HASH( MONTH(tr_date) )
    PARTITIONS 6;
```

Each `PARTITION` clause can include a `[STORAGE] ENGINE` option, but in MySQL 8.0 this has no effect.

Unless otherwise specified, the remaining examples in this discussion assume that `default_storage_engine` is `InnoDB`.

Important

Partitioning applies to all data and indexes of a table; you cannot partition only the data and not the indexes, or vice versa, nor can you partition only a portion of the table.

Data and indexes for each partition can be assigned to a specific directory using the `DATA DIRECTORY` and `INDEX DIRECTORY` options for the `PARTITION` clause of the `CREATE TABLE` statement used to create the partitioned table.

Only the `DATA DIRECTORY` option is supported for individual partitions and subpartitions of `InnoDB` tables.

All columns used in the table's partitioning expression must be part of every unique key that the table may have, including any primary key. This means that a table such as this one, created by the following SQL statement, cannot be partitioned:

```
CREATE TABLE tnp (
    id INT NOT NULL AUTO_INCREMENT,
    ref BIGINT NOT NULL,
    name VARCHAR(255),
    PRIMARY KEY pk (id),
    UNIQUE KEY uk (name)
);
```

Because the keys `pk` and `uk` have no columns in common, there are no columns available for use in a partitioning expression. Possible workarounds in this situation include adding the `name` column to the table's primary key, adding the `id` column to `uk`, or simply removing the unique key altogether. See Section 23.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”, for more information.

In addition, `MAX_ROWS` and `MIN_ROWS` can be used to determine the maximum and minimum numbers of rows, respectively, that can be stored in each partition. See Section 23.3, “Partition Management”, for more information on these options.

The `MAX_ROWS` option can also be useful for creating NDB Cluster tables with extra partitions, thus allowing for greater storage of hash indexes. See the documentation for the `DataMemory` data node configuration parameter, as well as Section 22.1.2, “NDB Cluster Nodes, Node Groups, Replicas, and Partitions”, for more information.

Some advantages of partitioning are listed here:

- Partitioning makes it possible to store more data in one table than can be held on a single disk or file system partition.
- Data that loses its usefulness can often be easily removed from a partitioned table by dropping the partition (or partitions) containing only that data. Conversely, the process of adding new data can in some cases be greatly facilitated by adding one or more new partitions for storing specifically that data.
- Some queries can be greatly optimized in virtue of the fact that data satisfying a given `WHERE` clause can be stored only on one or more partitions, which automatically excludes any remaining partitions from the search. Because partitions can be altered after a partitioned table has been created, you can reorganize your data to enhance frequent queries that may not have been often used when the partitioning scheme was first set up. This ability to exclude non-matching partitions (and thus any rows they contain) is often referred to as *partition pruning*. For more information, see Section 23.4, “Partition Pruning”.

  In addition, MySQL supports explicit partition selection for queries. For example, `SELECT * FROM t PARTITION (p0,p1) WHERE c < 5` selects only those rows in partitions `p0` and `p1` that match the `WHERE` condition. In this case, MySQL does not check any other partitions of table `t`; this can greatly speed up queries when you already know which partition or partitions you wish to examine. Partition selection is also supported for the data modification statements `DELETE`, `INSERT`, `REPLACE`, `UPDATE`, and `LOAD DATA`, `LOAD XML`. See the descriptions of these statements for more information and examples.

23.2 Partitioning Types

This section discusses the types of partitioning which are available in MySQL 8.0. These include the types listed here:

- **RANGE partitioning.** This type of partitioning assigns rows to partitions based on column values falling within a given range. See Section 23.2.1, “RANGE Partitioning”. For information about an extension to this type, `RANGE COLUMNS`, see Section 23.2.3.1, “RANGE COLUMNS partitioning”.
- **LIST partitioning.** Similar to partitioning by `RANGE`, except that the partition is selected based on columns matching one of a set of discrete values. See Section 23.2.2, “LIST Partitioning”. For information about an extension to this type, `LIST COLUMNS`, see Section 23.2.3.2, “LIST COLUMNS partitioning”.
- **HASH partitioning.** With this type of partitioning, a partition is selected based on the value returned by a user-defined expression that operates on column values in rows to be inserted into the table. The function may consist of any expression valid in MySQL that yields a nonnegative integer value. An extension to this type, `LINEAR HASH`, is also available. See Section 23.2.4, “HASH Partitioning”.
- **KEY partitioning.** This type of partitioning is similar to partitioning by `HASH`, except that only one or more columns to be evaluated are supplied, and the MySQL server provides its own hashing function. These columns can contain other than integer values, since the hashing function supplied by MySQL guarantees an integer result regardless of the column data type. An extension to this type, `LINEAR KEY`, is also available. See Section 23.2.5, “KEY Partitioning”.

A very common use of database partitioning is to segregate data by date. Some database systems support explicit date partitioning, which MySQL does not implement in 8.0. However, it is not difficult in MySQL to create partitioning schemes based on `DATE`, `TIME`, or `DATETIME` columns, or based on expressions making use of such columns.

When partitioning by KEY or LINEAR KEY, you can use a DATE, TIME, or DATETIME column as the partitioning column without performing any modification of the column value. For example, this table creation statement is perfectly valid in MySQL:

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL,
    email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY KEY(joined)
PARTITIONS 6;
```

In MySQL 8.0, it is also possible to use a DATE or DATETIME column as the partitioning column using RANGE COLUMNS and LIST COLUMNS partitioning.

Other partitioning types require a partitioning expression that yields an integer value or NULL. If you wish to use date-based partitioning by RANGE, LIST, HASH, or LINEAR HASH, you can simply employ a function that operates on a DATE, TIME, or DATETIME column and returns such a value, as shown here:

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL,
    email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
    PARTITION p0 VALUES LESS THAN (1960),
    PARTITION p1 VALUES LESS THAN (1970),
    PARTITION p2 VALUES LESS THAN (1980),
    PARTITION p3 VALUES LESS THAN (1990),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Additional examples of partitioning using dates may be found in the following sections of this chapter:

- Section 23.2.1, “RANGE Partitioning”
- Section 23.2.4, “HASH Partitioning”
- Section 23.2.4.1, “LINEAR HASH Partitioning”

For more complex examples of date-based partitioning, see the following sections:

- Section 23.4, “Partition Pruning”
- Section 23.2.6, “Subpartitioning”

MySQL partitioning is optimized for use with the TO_DAYS(), YEAR(), and TO_SECONDS() functions. However, you can use other date and time functions that return an integer or NULL, such as WEEKDAY(), DAYOFYEAR(), or MONTH(). See Section 12.7, “Date and Time Functions”, for more information about such functions.

It is important to remember—regardless of the type of partitioning that you use—that partitions are always numbered automatically and in sequence when created, starting with 0. When a new row is inserted into a partitioned table, it is these partition numbers that are used in identifying the correct partition. For example, if your table uses 4 partitions, these partitions are numbered 0, 1, 2, and 3. For the RANGE and LIST

partitioning types, it is necessary to ensure that there is a partition defined for each partition number. For `HASH` partitioning, the user-supplied expression must evaluate to an integer value greater than `0`. For `KEY` partitioning, this issue is taken care of automatically by the hashing function which the MySQL server employs internally.

Names of partitions generally follow the rules governing other MySQL identifiers, such as those for tables and databases. However, you should note that partition names are not case-sensitive. For example, the following `CREATE TABLE` statement fails as shown:

```
mysql> CREATE TABLE t2 (val INT)
    -> PARTITION BY LIST(val)(
    ->     PARTITION mypart VALUES IN (1,3,5),
    ->     PARTITION MyPart VALUES IN (2,4,6)
    -> );
ERROR 1488 (HY000): Duplicate partition name mypart
```

Failure occurs because MySQL sees no difference between the partition names `mypart` and `MyPart`.

When you specify the number of partitions for the table, this must be expressed as a positive, nonzero integer literal with no leading zeros, and may not be an expression such as `0.8E+01` or `6-2`, even if it evaluates to an integer value. Decimal fractions are not permitted.

In the sections that follow, we do not necessarily provide all possible forms for the syntax that can be used for creating each partition type; for this information, see Section 13.1.20, “CREATE TABLE Syntax”.

23.2.1 RANGE Partitioning

A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range. Ranges should be contiguous but not overlapping, and are defined using the `VALUES LESS THAN` operator. For the next few examples, suppose that you are creating a table such as the following to hold personnel records for a chain of 20 video stores, numbered 1 through 20:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
);
```

Note

The `employees` table used here has no primary or unique keys. While the examples work as shown for purposes of the present discussion, you should keep in mind that tables are extremely likely in practice to have primary keys, unique keys, or both, and that allowable choices for partitioning columns depend on the columns used for these keys, if any are present. For a discussion of these issues, see Section 23.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”.

This table can be partitioned by range in a number of ways, depending on your needs. One way would be to use the `store_id` column. For instance, you might decide to partition the table 4 ways by adding a `PARTITION BY RANGE` clause as shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN (21)
);
```

In this partitioning scheme, all rows corresponding to employees working at stores 1 through 5 are stored in partition `p0`, to those employed at stores 6 through 10 are stored in partition `p1`, and so on. Each partition is defined in order, from lowest to highest. This is a requirement of the `PARTITION BY RANGE` syntax; you can think of it as being analogous to a series of `if ... elseif ...` statements in C or Java in this regard.

It is easy to determine that a new row containing the data `(72, 'Mitchell', 'Wilson', '1998-06-25', NULL, 13)` is inserted into partition `p2`, but what happens when your chain adds a 21st store? Under this scheme, there is no rule that covers a row whose `store_id` is greater than 20, so an error results because the server does not know where to place it. You can keep this from occurring by using a "catchall" `VALUES LESS THAN` clause in the `CREATE TABLE` statement that provides for all values greater than the highest value explicitly named:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

(As with the other examples in this chapter, we assume that the default storage engine is `InnoDB`.)

Note

Another way to avoid an error when no matching value is found is to use the `IGNORE` keyword as part of the `INSERT` statement. For an example, see Section 23.2.2, "LIST Partitioning". Also see Section 13.2.6, "INSERT Syntax", for general information about `IGNORE`.

`MAXVALUE` represents an integer value that is always greater than the largest possible integer value (in mathematical language, it serves as a *least upper bound*). Now, any rows whose `store_id` column value is greater than or equal to 16 (the highest value defined) are stored in partition `p3`. At some point in the future—when the number of stores has increased to 25, 30, or more—you can use an `ALTER TABLE` statement to add new partitions for stores 21-25, 26-30, and so on (see Section 23.3, "Partition Management", for details of how to do this).

In much the same fashion, you could partition the table based on employee job codes—that is, based on ranges of `job_code` column values. For example—assuming that two-digit job codes are used for regular (in-store) workers, three-digit codes are used for office and support personnel, and four-digit codes are used for management positions—you could create the partitioned table using the following statement:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (job_code) (
    PARTITION p0 VALUES LESS THAN (100),
    PARTITION p1 VALUES LESS THAN (1000),
    PARTITION p2 VALUES LESS THAN (10000)
);
```

In this instance, all rows relating to in-store workers would be stored in partition `p0`, those relating to office and support staff in `p1`, and those relating to managers in partition `p2`.

It is also possible to use an expression in `VALUES LESS THAN` clauses. However, MySQL must be able to evaluate the expression's return value as part of a `LESS THAN` (`<`) comparison.

Rather than splitting up the table data according to store number, you can use an expression based on one of the two `DATE` columns instead. For example, let us suppose that you wish to partition based on the year that each employee left the company; that is, the value of `YEAR(separated)`. An example of a `CREATE TABLE` statement that implements such a partitioning scheme is shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY RANGE ( YEAR(separated) ) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1996),
    PARTITION p2 VALUES LESS THAN (2001),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

In this scheme, for all employees who left before 1991, the rows are stored in partition `p0`; for those who left in the years 1991 through 1995, in `p1`; for those who left in the years 1996 through 2000, in `p2`; and for any workers who left after the year 2000, in `p3`.

It is also possible to partition a table by `RANGE`, based on the value of a `TIMESTAMP` column, using the `UNIX_TIMESTAMP()` function, as shown in this example:

```
CREATE TABLE quarterly_report_status (
    report_id INT NOT NULL,
    report_status VARCHAR(20) NOT NULL,
    report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
    PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
```

```
    PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
    PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
    PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
    PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
    PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
    PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
    PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
    PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
    PARTITION p9 VALUES LESS THAN (MAXVALUE)
);
```

Any other expressions involving `TIMESTAMP` values are not permitted. (See Bug #42849.)

Range partitioning is particularly useful when one or more of the following conditions is true:

- You want or need to delete “old” data. If you are using the partitioning scheme shown previously for the `employees` table, you can simply use `ALTER TABLE employees DROP PARTITION p0;` to delete all rows relating to employees who stopped working for the firm prior to 1991. (See Section 13.1.9, “ALTER TABLE Syntax”, and Section 23.3, “Partition Management”, for more information.) For a table with a great many rows, this can be much more efficient than running a `DELETE` query such as `DELETE FROM employees WHERE YEAR(separated) <= 1990;`.
- You want to use a column containing date or time values, or containing values arising from some other series.
- You frequently run queries that depend directly on the column used for partitioning the table. For example, when executing a query such as `EXPLAIN SELECT COUNT(*) FROM employees WHERE separated BETWEEN '2000-01-01' AND '2000-12-31' GROUP BY store_id;`, MySQL can quickly determine that only partition `p2` needs to be scanned because the remaining partitions cannot contain any records satisfying the `WHERE` clause. See Section 23.4, “Partition Pruning”, for more information about how this is accomplished.

A variant on this type of partitioning is `RANGE COLUMNS` partitioning. Partitioning by `RANGE COLUMNS` makes it possible to employ multiple columns for defining partitioning ranges that apply both to placement of rows in partitions and for determining the inclusion or exclusion of specific partitions when performing partition pruning. See Section 23.2.3.1, “RANGE COLUMNS partitioning”, for more information.

Partitioning schemes based on time intervals. If you wish to implement a partitioning scheme based on ranges or intervals of time in MySQL 8.0, you have two options:

1. Partition the table by `RANGE`, and for the partitioning expression, employ a function operating on a `DATE`, `TIME`, or `DATETIME` column and returning an integer value, as shown here:

   ```
   CREATE TABLE members (
       firstname VARCHAR(25) NOT NULL,
       lastname VARCHAR(25) NOT NULL,
       username VARCHAR(16) NOT NULL,
       email VARCHAR(35),
       joined DATE NOT NULL
   )
   PARTITION BY RANGE( YEAR(joined) ) (
       PARTITION p0 VALUES LESS THAN (1960),
       PARTITION p1 VALUES LESS THAN (1970),
       PARTITION p2 VALUES LESS THAN (1980),
       PARTITION p3 VALUES LESS THAN (1990),
       PARTITION p4 VALUES LESS THAN MAXVALUE
   );
   ```

   In MySQL 8.0, it is also possible to partition a table by `RANGE` based on the value of a `TIMESTAMP` column, using the `UNIX_TIMESTAMP()` function, as shown in this example:

```
CREATE TABLE quarterly_report_status (
    report_id INT NOT NULL,
    report_status VARCHAR(20) NOT NULL,
    report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
    PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
    PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
    PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
    PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
    PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
    PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
    PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
    PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
    PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
    PARTITION p9 VALUES LESS THAN (MAXVALUE)
);
```

In MySQL 8.0, any other expressions involving `TIMESTAMP` values are not permitted. (See Bug #42849.)

Note

It is also possible in MySQL 8.0 to use `UNIX_TIMESTAMP(timestamp_column)` as a partitioning expression for tables that are partitioned by `LIST`. However, it is usually not practical to do so.

2. Partition the table by `RANGE COLUMNS`, using a `DATE` or `DATETIME` column as the partitioning column. For example, the `members` table could be defined using the `joined` column directly, as shown here:

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL,
    email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY RANGE COLUMNS(joined) (
    PARTITION p0 VALUES LESS THAN ('1960-01-01'),
    PARTITION p1 VALUES LESS THAN ('1970-01-01'),
    PARTITION p2 VALUES LESS THAN ('1980-01-01'),
    PARTITION p3 VALUES LESS THAN ('1990-01-01'),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Note

The use of partitioning columns employing date or time types other than `DATE` or `DATETIME` is not supported with `RANGE COLUMNS`.

23.2.2 LIST Partitioning

List partitioning in MySQL is similar to range partitioning in many ways. As in partitioning by `RANGE`, each partition must be explicitly defined. The chief difference between the two types of partitioning is that, in list partitioning, each partition is defined and selected based on the membership of a column value in one of a set of value lists, rather than in one of a set of contiguous ranges of values. This is done by using `PARTITION BY LIST(expr)` where *`expr`* is a column value or an expression based on a column value and returning an integer value, and then defining each partition by means of a `VALUES IN (value_list)`, where *`value_list`* is a comma-separated list of integers.

Note

In MySQL 8.0, it is possible to match against only a list of integers (and possibly `NULL`—see Section 23.2.7, “How MySQL Partitioning Handles NULL”) when partitioning by `LIST`.

However, other column types may be used in value lists when employing `LIST COLUMN` partitioning, which is described later in this section.

Unlike the case with partitions defined by range, list partitions do not need to be declared in any particular order. For more detailed syntactical information, see Section 13.1.20, “CREATE TABLE Syntax”.

For the examples that follow, we assume that the basic definition of the table to be partitioned is provided by the `CREATE TABLE` statement shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
);
```

(This is the same table used as a basis for the examples in Section 23.2.1, “RANGE Partitioning”. As with the other partitioning examples, we assume that the `default_storage_engine` is `InnoDB`.)

Suppose that there are 20 video stores distributed among 4 franchises as shown in the following table.

Region	Store ID Numbers
North	3, 5, 6, 9, 17
East	1, 2, 10, 11, 19, 20
West	4, 12, 13, 14, 18
Central	7, 8, 15, 16

To partition this table in such a way that rows for stores belonging to the same region are stored in the same partition, you could use the `CREATE TABLE` statement shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LIST(store_id) (
    PARTITION pNorth VALUES IN (3,5,6,9,17),
    PARTITION pEast VALUES IN (1,2,10,11,19,20),
    PARTITION pWest VALUES IN (4,12,13,14,18),
    PARTITION pCentral VALUES IN (7,8,15,16)
);
```

This makes it easy to add or drop employee records relating to specific regions to or from the table. For instance, suppose that all stores in the West region are sold to another company. In MySQL 8.0, all rows relating to employees working at stores in that region can be deleted with the query `ALTER TABLE`

`employees TRUNCATE PARTITION pWest`, which can be executed much more efficiently than the equivalent `DELETE` statement `DELETE FROM employees WHERE store_id IN (4,12,13,14,18);`. (Using `ALTER TABLE employees DROP PARTITION pWest` would also delete all of these rows, but would also remove the partition `pWest` from the definition of the table; you would need to use an `ALTER TABLE ... ADD PARTITION` statement to restore the table's original partitioning scheme.)

As with `RANGE` partitioning, it is possible to combine `LIST` partitioning with partitioning by hash or key to produce a composite partitioning (subpartitioning). See Section 23.2.6, “Subpartitioning”.

Unlike the case with `RANGE` partitioning, there is no “catch-all” such as `MAXVALUE`; all expected values for the partitioning expression should be covered in `PARTITION ... VALUES IN (...)` clauses. An `INSERT` statement containing an unmatched partitioning column value fails with an error, as shown in this example:

```
mysql> CREATE TABLE h2 (
    ->   c1 INT,
    ->   c2 INT
    -> )
    -> PARTITION BY LIST(c1) (
    ->   PARTITION p0 VALUES IN (1, 4, 7),
    ->   PARTITION p1 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO h2 VALUES (3, 5);
ERROR 1525 (HY000): Table has no partition for value 3
```

When inserting multiple rows using a single `INSERT` statement into a single `InnoDB` table, `InnoDB` considers the statement a single transaction, so that the presence of any unmatched values causes the statement to fail completely, and so no rows are inserted.

You can cause this type of error to be ignored by using the `IGNORE` keyword. If you do so, rows containing unmatched partitioning column values are not inserted, but any rows with matching values *are* inserted, and no errors are reported:

```
mysql> TRUNCATE h2;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM h2;
Empty set (0.00 sec)

mysql> INSERT IGNORE INTO h2 VALUES (2, 5), (6, 10), (7, 5), (3, 1), (1, 9);
Query OK, 3 rows affected (0.00 sec)
Records: 5  Duplicates: 2  Warnings: 0

mysql> SELECT * FROM h2;
+------+------+
| c1   | c2   |
+------+------+
|    7 |    5 |
|    1 |    9 |
|    2 |    5 |
+------+------+
3 rows in set (0.00 sec)
```

MySQL 8.0 also provides support for `LIST COLUMNS` partitioning, a variant of `LIST` partitioning that enables you to use columns of types other than integer types for partitioning columns, and to use multiple columns as partitioning keys. For more information, see Section 23.2.3.2, “LIST COLUMNS partitioning”.

23.2.3 COLUMNS Partitioning

The next two sections discuss *`COLUMNS` partitioning*, which are variants on `RANGE` and `LIST` partitioning. `COLUMNS` partitioning enables the use of multiple columns in partitioning keys. All of these columns are taken into account both for the purpose of placing rows in partitions and for the determination of which partitions are to be checked for matching rows in partition pruning.

In addition, both `RANGE COLUMNS` partitioning and `LIST COLUMNS` partitioning support the use of non-integer columns for defining value ranges or list members. The permitted data types are shown in the following list:

- All integer types: `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT` (`INTEGER`), and `BIGINT`. (This is the same as with partitioning by `RANGE` and `LIST`.)

  Other numeric data types (such as `DECIMAL` or `FLOAT`) are not supported as partitioning columns.

- `DATE` and `DATETIME`.

  Columns using other data types relating to dates or times are not supported as partitioning columns.

- The following string types: `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY`.

  `TEXT` and `BLOB` columns are not supported as partitioning columns.

The discussions of `RANGE COLUMNS` and `LIST COLUMNS` partitioning in the next two sections assume that you are already familiar with partitioning based on ranges and lists as supported in MySQL 5.1 and later; for more information about these, see Section 23.2.1, “RANGE Partitioning”, and Section 23.2.2, “LIST Partitioning”, respectively.

23.2.3.1 RANGE COLUMNS partitioning

Range columns partitioning is similar to range partitioning, but enables you to define partitions using ranges based on multiple column values. In addition, you can define the ranges using columns of types other than integer types.

`RANGE COLUMNS` partitioning differs significantly from `RANGE` partitioning in the following ways:

- `RANGE COLUMNS` does not accept expressions, only names of columns.
- `RANGE COLUMNS` accepts a list of one or more columns.

  `RANGE COLUMNS` partitions are based on comparisons between *tuples* (lists of column values) rather than comparisons between scalar values. Placement of rows in `RANGE COLUMNS` partitions is also based on comparisons between tuples; this is discussed further later in this section.

- `RANGE COLUMNS` partitioning columns are not restricted to integer columns; string, `DATE` and `DATETIME` columns can also be used as partitioning columns. (See Section 23.2.3, “COLUMNS Partitioning”, for details.)

The basic syntax for creating a table partitioned by `RANGE COLUMNS` is shown here:

```
CREATE TABLE table_name
PARTITIONED BY RANGE COLUMNS(column_list) (
    PARTITION partition_name VALUES LESS THAN (value_list)[,
    PARTITION partition_name VALUES LESS THAN (value_list)][,
    ...]
)

column_list:
```

```
    column_name[, column_name][, ...]

value_list:
    value[, value][, ...]
```

Note

Not all `CREATE TABLE` options that can be used when creating partitioned tables are shown here. For complete information, see Section 13.1.20, “CREATE TABLE Syntax”.

In the syntax just shown, *`column_list`* is a list of one or more columns (sometimes called a *partitioning column list*), and *`value_list`* is a list of values (that is, it is a *partition definition value list*). A *`value_list`* must be supplied for each partition definition, and each *`value_list`* must have the same number of values as the *`column_list`* has columns. Generally speaking, if you use *`N`* columns in the `COLUMNS` clause, then each `VALUES LESS THAN` clause must also be supplied with a list of *`N`* values.

The elements in the partitioning column list and in the value list defining each partition must occur in the same order. In addition, each element in the value list must be of the same data type as the corresponding element in the column list. However, the order of the column names in the partitioning column list and the value lists does not have to be the same as the order of the table column definitions in the main part of the `CREATE TABLE` statement. As with table partitioned by `RANGE`, you can use `MAXVALUE` to represent a value such that any legal value inserted into a given column is always less than this value. Here is an example of a `CREATE TABLE` statement that helps to illustrate all of these points:

```
mysql> CREATE TABLE rcx (
    ->     a INT,
    ->     b INT,
    ->     c CHAR(3),
    ->     d INT
    -> )
    -> PARTITION BY RANGE COLUMNS(a,d,c) (
    ->     PARTITION p0 VALUES LESS THAN (5,10,'ggg'),
    ->     PARTITION p1 VALUES LESS THAN (10,20,'mmm'),
    ->     PARTITION p2 VALUES LESS THAN (15,30,'sss'),
    ->     PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
    -> );
Query OK, 0 rows affected (0.15 sec)
```

Table `rcx` contains the columns `a`, `b`, `c`, `d`. The partitioning column list supplied to the `COLUMNS` clause uses 3 of these columns, in the order `a`, `d`, `c`. Each value list used to define a partition contains 3 values in the same order; that is, each value list tuple has the form (`INT`, `INT`, `CHAR(3)`), which corresponds to the data types used by columns `a`, `d`, and `c` (in that order).

Placement of rows into partitions is determined by comparing the tuple from a row to be inserted that matches the column list in the `COLUMNS` clause with the tuples used in the `VALUES LESS THAN` clauses to define partitions of the table. Because we are comparing tuples (that is, lists or sets of values) rather than scalar values, the semantics of `VALUES LESS THAN` as used with `RANGE COLUMNS` partitions differs somewhat from the case with simple `RANGE` partitions. In `RANGE` partitioning, a row generating an expression value that is equal to a limiting value in a `VALUES LESS THAN` is never placed in the corresponding partition; however, when using `RANGE COLUMNS` partitioning, it is sometimes possible for a row whose partitioning column list's first element is equal in value to the that of the first element in a `VALUES LESS THAN` value list to be placed in the corresponding partition.

Consider the `RANGE` partitioned table created by this statement:

```
CREATE TABLE r1 (
    a INT,
```

```
    b INT
)
PARTITION BY RANGE (a)  (
    PARTITION p0 VALUES LESS THAN (5),
    PARTITION p1 VALUES LESS THAN (MAXVALUE)
);
```

If we insert 3 rows into this table such that the column value for `a` is `5` for each row, all 3 rows are stored in partition `p1` because the `a` column value is in each case not less than 5, as we can see by executing the proper query against the `INFORMATION_SCHEMA.PARTITIONS` table:

```
mysql> INSERT INTO r1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'r1';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p1             |          3 |
+----------------+------------+
2 rows in set (0.00 sec)
```

Now consider a similar table `rc1` that uses `RANGE COLUMNS` partitioning with both columns `a` and `b` referenced in the `COLUMNS` clause, created as shown here:

```
CREATE TABLE rc1 (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS(a, b) (
    PARTITION p0 VALUES LESS THAN (5, 12),
    PARTITION p3 VALUES LESS THAN (MAXVALUE, MAXVALUE)
);
```

If we insert exactly the same rows into `rc1` as we just inserted into `r1`, the distribution of the rows is quite different:

```
mysql> INSERT INTO rc1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'rc1';
+--------------+----------------+------------+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+--------------+----------------+------------+
| p            | p0             |          2 |
| p            | p1             |          1 |
+--------------+----------------+------------+
2 rows in set (0.00 sec)
```

This is because we are comparing rows rather than scalar values. We can compare the row values inserted with the limiting row value from the `VALUES THAN LESS THAN` clause used to define partition `p0` in table `rc1`, like this:

```
mysql> SELECT (5,10) < (5,12), (5,11) < (5,12), (5,12) < (5,12);
```

```
+-----------------+-----------------+-----------------+
| (5,10) < (5,12) | (5,11) < (5,12) | (5,12) < (5,12) |
+-----------------+-----------------+-----------------+
|               1 |               1 |               0 |
+-----------------+-----------------+-----------------+
1 row in set (0.00 sec)
```

The 2 tuples `(5,10)` and `(5,11)` evaluate as less than `(5,12)`, so they are stored in partition `p0`. Since 5 is not less than 5 and 12 is not less than 12, `(5,12)` is considered not less than `(5,12)`, and is stored in partition `p1`.

The `SELECT` statement in the preceding example could also have been written using explicit row constructors, like this:

```
SELECT ROW(5,10) < ROW(5,12), ROW(5,11) < ROW(5,12), ROW(5,12) < ROW(5,12);
```

For more information about the use of row constructors in MySQL, see Section 13.2.11.5, “Row Subqueries”.

For a table partitioned by `RANGE COLUMNS` using only a single partitioning column, the storing of rows in partitions is the same as that of an equivalent table that is partitioned by `RANGE`. The following `CREATE TABLE` statement creates a table partitioned by `RANGE COLUMNS` using 1 partitioning column:

```
CREATE TABLE rx (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS (a)  (
    PARTITION p0 VALUES LESS THAN (5),
    PARTITION p1 VALUES LESS THAN (MAXVALUE)
);
```

If we insert the rows `(5,10)`, `(5,11)`, and `(5,12)` into this table, we can see that their placement is the same as it is for the table `r` we created and populated earlier:

```
mysql> INSERT INTO rx VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'rx';
+--------------+----------------+------------+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+--------------+----------------+------------+
| p            | p0             |          0 |
| p            | p1             |          3 |
+--------------+----------------+------------+
2 rows in set (0.00 sec)
```

It is also possible to create tables partitioned by `RANGE COLUMNS` where limiting values for one or more columns are repeated in successive partition definitions. You can do this as long as the tuples of column values used to define the partitions are strictly increasing. For example, each of the following `CREATE TABLE` statements is valid:

```
CREATE TABLE rc2 (
    a INT,
    b INT
)
```

```
PARTITION BY RANGE COLUMNS(a,b) (
    PARTITION p0 VALUES LESS THAN (0,10),
    PARTITION p1 VALUES LESS THAN (10,20),
    PARTITION p2 VALUES LESS THAN (10,30),
    PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE)
 );

CREATE TABLE rc3 (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS(a,b) (
    PARTITION p0 VALUES LESS THAN (0,10),
    PARTITION p1 VALUES LESS THAN (10,20),
    PARTITION p2 VALUES LESS THAN (10,30),
    PARTITION p3 VALUES LESS THAN (10,35),
    PARTITION p4 VALUES LESS THAN (20,40),
    PARTITION p5 VALUES LESS THAN (MAXVALUE,MAXVALUE)
 );
```

The following statement also succeeds, even though it might appear at first glance that it would not, since the limiting value of column `b` is 25 for partition `p0` and 20 for partition `p1`, and the limiting value of column `c` is 100 for partition `p1` and 50 for partition `p2`:

```
CREATE TABLE rc4 (
    a INT,
    b INT,
    c INT
)
PARTITION BY RANGE COLUMNS(a,b,c) (
    PARTITION p0 VALUES LESS THAN (0,25,50),
    PARTITION p1 VALUES LESS THAN (10,20,100),
    PARTITION p2 VALUES LESS THAN (10,30,50)
    PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
 );
```

When designing tables partitioned by `RANGE COLUMNS`, you can always test successive partition definitions by comparing the desired tuples using the `mysql` client, like this:

```
mysql> SELECT (0,25,50) < (10,20,100), (10,20,100) < (10,30,50);
+-------------------------+-------------------------+
| (0,25,50) < (10,20,100) | (10,20,100) < (10,30,50) |
+-------------------------+-------------------------+
|                       1 |                       1 |
+-------------------------+-------------------------+
1 row in set (0.00 sec)
```

If a `CREATE TABLE` statement contains partition definitions that are not in strictly increasing order, it fails with an error, as shown in this example:

```
mysql> CREATE TABLE rcf (
    ->     a INT,
    ->     b INT,
    ->     c INT
    -> )
    -> PARTITION BY RANGE COLUMNS(a,b,c) (
    ->     PARTITION p0 VALUES LESS THAN (0,25,50),
    ->     PARTITION p1 VALUES LESS THAN (20,20,100),
    ->     PARTITION p2 VALUES LESS THAN (10,30,50),
    ->     PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
    ->  );
ERROR 1493 (HY000): VALUES LESS THAN value must be strictly increasing for each partition
```

When you get such an error, you can deduce which partition definitions are invalid by making “less than” comparisons between their column lists. In this case, the problem is with the definition of partition `p2` because the tuple used to define it is not less than the tuple used to define partition `p3`, as shown here:

```
mysql> SELECT (0,25,50) < (20,20,100), (20,20,100) < (10,30,50);
+------------------------+------------------------+
| (0,25,50) < (20,20,100) | (20,20,100) < (10,30,50) |
+------------------------+------------------------+
|                      1 |                      0 |
+------------------------+------------------------+
1 row in set (0.00 sec)
```

It is also possible for `MAXVALUE` to appear for the same column in more than one `VALUES LESS THAN` clause when using `RANGE COLUMNS`. However, the limiting values for individual columns in successive partition definitions should otherwise be increasing, there should be no more than one partition defined where `MAXVALUE` is used as the upper limit for all column values, and this partition definition should appear last in the list of `PARTITION ... VALUES LESS THAN` clauses. In addition, you cannot use `MAXVALUE` as the limiting value for the first column in more than one partition definition.

As stated previously, it is also possible with `RANGE COLUMNS` partitioning to use non-integer columns as partitioning columns. (See Section 23.2.3, “COLUMNS Partitioning”, for a complete listing of these.) Consider a table named `employees` (which is not partitioned), created using the following statement:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
);
```

Using `RANGE COLUMNS` partitioning, you can create a version of this table that stores each row in one of four partitions based on the employee's last name, like this:

```
CREATE TABLE employees_by_lname (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE COLUMNS (lname)  (
    PARTITION p0 VALUES LESS THAN ('g'),
    PARTITION p1 VALUES LESS THAN ('m'),
    PARTITION p2 VALUES LESS THAN ('t'),
    PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
```

Alternatively, you could cause the `employees` table as created previously to be partitioned using this scheme by executing the following `ALTER TABLE` statement:

```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (lname)  (
    PARTITION p0 VALUES LESS THAN ('g'),
    PARTITION p1 VALUES LESS THAN ('m'),
    PARTITION p2 VALUES LESS THAN ('t'),
```

```
    PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
```

Note

Because different character sets and collations have different sort orders, the character sets and collations in use may effect which partition of a table partitioned by `RANGE COLUMNS` a given row is stored in when using string columns as partitioning columns. In addition, changing the character set or collation for a given database, table, or column after such a table is created may cause changes in how rows are distributed. For example, when using a case-sensitive collation, `'and'` sorts before `'Andersen'`, but when using a collation that is case insensitive, the reverse is true.

For information about how MySQL handles character sets and collations, see Chapter 10, *Character Sets, Collations, Unicode*.

Similarly, you can cause the `employees` table to be partitioned in such a way that each row is stored in one of several partitions based on the decade in which the corresponding employee was hired using the `ALTER TABLE` statement shown here:

```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (hired)  (
    PARTITION p0 VALUES LESS THAN ('1970-01-01'),
    PARTITION p1 VALUES LESS THAN ('1980-01-01'),
    PARTITION p2 VALUES LESS THAN ('1990-01-01'),
    PARTITION p3 VALUES LESS THAN ('2000-01-01'),
    PARTITION p4 VALUES LESS THAN ('2010-01-01'),
    PARTITION p5 VALUES LESS THAN (MAXVALUE)
);
```

See Section 13.1.20, “CREATE TABLE Syntax”, for additional information about `PARTITION BY RANGE COLUMNS` syntax.

23.2.3.2 LIST COLUMNS partitioning

MySQL 8.0 provides support for `LIST COLUMNS` partitioning. This is a variant of `LIST` partitioning that enables the use of multiple columns as partition keys, and for columns of data types other than integer types to be used as partitioning columns; you can use string types, `DATE`, and `DATETIME` columns. (For more information about permitted data types for `COLUMNS` partitioning columns, see Section 23.2.3, “COLUMNS Partitioning”.)

Suppose that you have a business that has customers in 12 cities which, for sales and marketing purposes, you organize into 4 regions of 3 cities each as shown in the following table:

Region	Cities
1	Oskarshamn, Högsby, Mönsterås
2	Vimmerby, Hultsfred, Västervik
3	Nässjö, Eksjö, Vetlanda
4	Uppvidinge, Alvesta, Växjo

With `LIST COLUMNS` partitioning, you can create a table for customer data that assigns a row to any of 4 partitions corresponding to these regions based on the name of the city where a customer resides, as shown here:

```
CREATE TABLE customers_1 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY LIST COLUMNS(city) (
    PARTITION pRegion_1 VALUES IN('Oskarshamn', 'Högsby', 'Mönsterås'),
    PARTITION pRegion_2 VALUES IN('Vimmerby', 'Hultsfred', 'Västervik'),
    PARTITION pRegion_3 VALUES IN('Nässjö', 'Eksjö', 'Vetlanda'),
    PARTITION pRegion_4 VALUES IN('Uppvidinge', 'Alvesta', 'Växjo')
);
```

As with partitioning by `RANGE COLUMNS`, you do not need to use expressions in the `COLUMNS()` clause to convert column values into integers. (In fact, the use of expressions other than column names is not permitted with `COLUMNS()`.)

It is also possible to use `DATE` and `DATETIME` columns, as shown in the following example that uses the same name and columns as the `customers_1` table shown previously, but employs `LIST COLUMNS` partitioning based on the `renewal` column to store rows in one of 4 partitions depending on the week in February 2010 the customer's account is scheduled to renew:

```
CREATE TABLE customers_2 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY LIST COLUMNS(renewal) (
    PARTITION pWeek_1 VALUES IN('2010-02-01', '2010-02-02', '2010-02-03',
        '2010-02-04', '2010-02-05', '2010-02-06', '2010-02-07'),
    PARTITION pWeek_2 VALUES IN('2010-02-08', '2010-02-09', '2010-02-10',
        '2010-02-11', '2010-02-12', '2010-02-13', '2010-02-14'),
    PARTITION pWeek_3 VALUES IN('2010-02-15', '2010-02-16', '2010-02-17',
        '2010-02-18', '2010-02-19', '2010-02-20', '2010-02-21'),
    PARTITION pWeek_4 VALUES IN('2010-02-22', '2010-02-23', '2010-02-24',
        '2010-02-25', '2010-02-26', '2010-02-27', '2010-02-28')
);
```

This works, but becomes cumbersome to define and maintain if the number of dates involved grows very large; in such cases, it is usually more practical to employ `RANGE` or `RANGE COLUMNS` partitioning instead. In this case, since the column we wish to use as the partitioning key is a `DATE` column, we use `RANGE COLUMNS` partitioning, as shown here:

```
CREATE TABLE customers_3 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY RANGE COLUMNS(renewal) (
    PARTITION pWeek_1 VALUES LESS THAN('2010-02-09'),
    PARTITION pWeek_2 VALUES LESS THAN('2010-02-15'),
    PARTITION pWeek_3 VALUES LESS THAN('2010-02-22'),
    PARTITION pWeek_4 VALUES LESS THAN('2010-03-01')
);
```

See Section 23.2.3.1, “RANGE COLUMNS partitioning”, for more information.

In addition (as with `RANGE COLUMNS` partitioning), you can use multiple columns in the `COLUMNS()` clause.

See Section 13.1.20, “CREATE TABLE Syntax”, for additional information about `PARTITION BY LIST COLUMNS()` syntax.

23.2.4 HASH Partitioning

Partitioning by `HASH` is used primarily to ensure an even distribution of data among a predetermined number of partitions. With range or list partitioning, you must specify explicitly which partition a given column value or set of column values should be stored in; with hash partitioning, this decision is taken care of for you, and you need only specify a column value or expression based on a column value to be hashed and the number of partitions into which the partitioned table is to be divided.

To partition a table using `HASH` partitioning, it is necessary to append to the `CREATE TABLE` statement a `PARTITION BY HASH (expr)` clause, where *`expr`* is an expression that returns an integer. This can simply be the name of a column whose type is one of MySQL's integer types. In addition, you most likely want to follow this with `PARTITIONS num`, where *`num`* is a positive integer representing the number of partitions into which the table is to be divided.

Note

For simplicity, the tables in the examples that follow do not use any keys. You should be aware that, if a table has any unique keys, every column used in the partitioning expression for this table must be part of every unique key, including the primary key. See Section 23.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”, for more information.

The following statement creates a table that uses hashing on the `store_id` column and is divided into 4 partitions:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

If you do not include a `PARTITIONS` clause, the number of partitions defaults to `1`; using the `PARTITIONS` keyword without a number following it results in a syntax error.

You can also use an SQL expression that returns an integer for *`expr`*. For instance, you might want to partition based on the year in which an employee was hired. This can be done as shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
```

```
    store_id INT
)
PARTITION BY HASH( YEAR(hired) )
PARTITIONS 4;
```

expr must return a nonconstant, nonrandom integer value (in other words, it should be varying but deterministic), and must not contain any prohibited constructs as described in Section 23.6, “Restrictions and Limitations on Partitioning”. You should also keep in mind that this expression is evaluated each time a row is inserted or updated (or possibly deleted); this means that very complex expressions may give rise to performance issues, particularly when performing operations (such as batch inserts) that affect a great many rows at one time.

The most efficient hashing function is one which operates upon a single table column and whose value increases or decreases consistently with the column value, as this allows for “pruning” on ranges of partitions. That is, the more closely that the expression varies with the value of the column on which it is based, the more efficiently MySQL can use the expression for hash partitioning.

For example, where `date_col` is a column of type `DATE`, then the expression `TO_DAYS(date_col)` is said to vary directly with the value of `date_col`, because for every change in the value of `date_col`, the value of the expression changes in a consistent manner. The variance of the expression `YEAR(date_col)` with respect to `date_col` is not quite as direct as that of `TO_DAYS(date_col)`, because not every possible change in `date_col` produces an equivalent change in `YEAR(date_col)`. Even so, `YEAR(date_col)` is a good candidate for a hashing function, because it varies directly with a portion of `date_col` and there is no possible change in `date_col` that produces a disproportionate change in `YEAR(date_col)`.

By way of contrast, suppose that you have a column named `int_col` whose type is `INT`. Now consider the expression `POW(5-int_col,3) + 6`. This would be a poor choice for a hashing function because a change in the value of `int_col` is not guaranteed to produce a proportional change in the value of the expression. Changing the value of `int_col` by a given amount can produce widely differing changes in the value of the expression. For example, changing `int_col` from `5` to `6` produces a change of `-1` in the value of the expression, but changing the value of `int_col` from `6` to `7` produces a change of `-7` in the expression value.

In other words, the more closely the graph of the column value versus the value of the expression follows a straight line as traced by the equation `y=cx` where *c* is some nonzero constant, the better the expression is suited to hashing. This has to do with the fact that the more nonlinear an expression is, the more uneven the distribution of data among the partitions it tends to produce.

In theory, pruning is also possible for expressions involving more than one column value, but determining which of such expressions are suitable can be quite difficult and time-consuming. For this reason, the use of hashing expressions involving multiple columns is not particularly recommended.

When `PARTITION BY HASH` is used, the storage engine determines which partition of *num* partitions to use based on the modulus of the result of the expression. In other words, for a given expression *expr*, the partition in which the record is stored is partition number *N*, where *N = MOD(expr, num)*. Suppose that table `t1` is defined as follows, so that it has 4 partitions:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY HASH( YEAR(col3) )
    PARTITIONS 4;
```

If you insert a record into `t1` whose `col3` value is `'2005-09-15'`, then the partition in which it is stored is determined as follows:

```
MOD(YEAR('2005-09-01'),4)
```

```
=  MOD(2005,4)
=  1
```

MySQL 8.0 also supports a variant of HASH partitioning known as *linear hashing* which employs a more complex algorithm for determining the placement of new rows inserted into the partitioned table. See Section 23.2.4.1, “LINEAR HASH Partitioning”, for a description of this algorithm.

The user-supplied expression is evaluated each time a record is inserted or updated. It may also—depending on the circumstances—be evaluated when records are deleted.

23.2.4.1 LINEAR HASH Partitioning

MySQL also supports linear hashing, which differs from regular hashing in that linear hashing utilizes a linear powers-of-two algorithm whereas regular hashing employs the modulus of the hashing function's value.

Syntactically, the only difference between linear-hash partitioning and regular hashing is the addition of the LINEAR keyword in the PARTITION BY clause, as shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LINEAR HASH( YEAR(hired) )
PARTITIONS 4;
```

Given an expression *expr*, the partition in which the record is stored when linear hashing is used is partition number *N* from among *num* partitions, where *N* is derived according to the following algorithm:

1. Find the next power of 2 greater than *num*. We call this value *V*; it can be calculated as:

   ```
   V = POWER(2, CEILING(LOG(2, num)))
   ```

   (Suppose that *num* is 13. Then LOG(2,13) is 3.7004397181411. CEILING(3.7004397181411) is 4, and *V* = POWER(2,4), which is 16.)

2. Set *N* = *F*(*column_list*) & (*V* - 1).

3. While *N* >= *num*:

   - Set *V* = *V* / 2
   - Set *N* = *N* & (*V* - 1)

Suppose that the table t1, using linear hash partitioning and having 6 partitions, is created using this statement:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY LINEAR HASH( YEAR(col3) )
    PARTITIONS 6;
```

Now assume that you want to insert two records into t1 having the col3 column values '2003-04-14' and '1998-10-19'. The partition number for the first of these is determined as follows:

```
V = POWER(2, CEILING( LOG(2,6) )) = 8
N = YEAR('2003-04-14') & (8 - 1)
   = 2003 & 7
   = 3

(3 >= 6 is FALSE: record stored in partition #3)
```

The number of the partition where the second record is stored is calculated as shown here:

```
V = 8
N = YEAR('1998-10-19') & (8 - 1)
  = 1998 & 7
  = 6

(6 >= 6 is TRUE: additional step required)

N = 6 & ((8 / 2) - 1)
  = 6 & 3
  = 2

(2 >= 6 is FALSE: record stored in partition #2)
```

The advantage in partitioning by linear hash is that the adding, dropping, merging, and splitting of partitions is made much faster, which can be beneficial when dealing with tables containing extremely large amounts (terabytes) of data. The disadvantage is that data is less likely to be evenly distributed between partitions as compared with the distribution obtained using regular hash partitioning.

23.2.5 KEY Partitioning

Partitioning by key is similar to partitioning by hash, except that where hash partitioning employs a user-defined expression, the hashing function for key partitioning is supplied by the MySQL server. NDB Cluster uses `MD5()` for this purpose; for tables using other storage engines, the server employs its own internal hashing function which is based on the same algorithm as `PASSWORD()`.

The syntax rules for `CREATE TABLE ... PARTITION BY KEY` are similar to those for creating a table that is partitioned by hash. The major differences are listed here:

- `KEY` is used rather than `HASH`.
- `KEY` takes only a list of zero or more column names. Any columns used as the partitioning key must comprise part or all of the table's primary key, if the table has one. Where no column name is specified as the partitioning key, the table's primary key is used, if there is one. For example, the following `CREATE TABLE` statement is valid in MySQL 8.0:

  ```
  CREATE TABLE k1 (
      id INT NOT NULL PRIMARY KEY,
      name VARCHAR(20)
  )
  PARTITION BY KEY()
  PARTITIONS 2;
  ```

  If there is no primary key but there is a unique key, then the unique key is used for the partitioning key:

  ```
  CREATE TABLE k1 (
      id INT NOT NULL,
      name VARCHAR(20),
      UNIQUE KEY (id)
  )
  ```

```
PARTITION BY KEY()
PARTITIONS 2;
```

However, if the unique key column were not defined as `NOT NULL`, then the previous statement would fail.

In both of these cases, the partitioning key is the `id` column, even though it is not shown in the output of `SHOW CREATE TABLE` or in the `PARTITION_EXPRESSION` column of the `INFORMATION_SCHEMA.PARTITIONS` table.

Unlike the case with other partitioning types, columns used for partitioning by `KEY` are not restricted to integer or `NULL` values. For example, the following `CREATE TABLE` statement is valid:

```
CREATE TABLE tm1 (
    s1 CHAR(32) PRIMARY KEY
)
PARTITION BY KEY(s1)
PARTITIONS 10;
```

The preceding statement would *not* be valid, were a different partitioning type to be specified. (In this case, simply using `PARTITION BY KEY()` would also be valid and have the same effect as `PARTITION BY KEY(s1)`, since `s1` is the table's primary key.)

For additional information about this issue, see Section 23.6, “Restrictions and Limitations on Partitioning”.

Note

Tables using the `NDB` storage engine are implicitly partitioned by `KEY`, again using the table's primary key as the partitioning key. In the event that the NDB Cluster table has no explicit primary key, the “hidden” primary key generated by the `NDB` storage engine for each NDB Cluster table is used as the partitioning key.

If you define an explicit partitioning scheme for an `NDB` table, the table must have an explicit primary key, and any columns used in the partitioning expression must be part of this key. However, if the table uses an “empty” partitioning expression —that is, `PARTITION BY KEY()` with no column references—then no explicit primary key is required.

You can observe this partitioning using the `ndb_desc` utility (with the `-p` option).

Important

For a key-partitioned table, you cannot execute an `ALTER TABLE DROP PRIMARY KEY`, as doing so generates the error `ERROR 1466 (HY000): Field in list of fields for partition function not found in table`. This is not an issue for NDB Cluster tables which are partitioned by `KEY`; in such cases, the table is reorganized using the “hidden” primary key as the table's new partitioning key. See Chapter 22, *MySQL NDB Cluster 8.0*.

It is also possible to partition a table by linear key. Here is a simple example:

```
CREATE TABLE tk (
    col1 INT NOT NULL,
    col2 CHAR(5),
    col3 DATE
```

```
)
PARTITION BY LINEAR KEY (col1)
PARTITIONS 3;
```

The `LINEAR` keyword has the same effect on `KEY` partitioning as it does on `HASH` partitioning, with the partition number being derived using a powers-of-two algorithm rather than modulo arithmetic. See Section 23.2.4.1, “LINEAR HASH Partitioning”, for a description of this algorithm and its implications.

23.2.6 Subpartitioning

Subpartitioning—also known as *composite partitioning*—is the further division of each partition in a partitioned table. Consider the following `CREATE TABLE` statement:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) )
    SUBPARTITIONS 2 (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (2000),
        PARTITION p2 VALUES LESS THAN MAXVALUE
    );
```

Table `ts` has 3 `RANGE` partitions. Each of these partitions—`p0`, `p1`, and `p2`—is further divided into 2 subpartitions. In effect, the entire table is divided into `3 * 2 = 6` partitions. However, due to the action of the `PARTITION BY RANGE` clause, the first 2 of these store only those records with a value less than 1990 in the `purchased` column.

It is possible to subpartition tables that are partitioned by `RANGE` or `LIST`. Subpartitions may use either `HASH` or `KEY` partitioning. This is also known as *composite partitioning*.

Note

`SUBPARTITION BY HASH` and `SUBPARTITION BY KEY` generally follow the same syntax rules as `PARTITION BY HASH` and `PARTITION BY KEY`, respectively. An exception to this is that `SUBPARTITION BY KEY` (unlike `PARTITION BY KEY`) does not currently support a default column, so the column used for this purpose must be specified, even if the table has an explicit primary key. This is a known issue which we are working to address; see Issues with subpartitions, for more information and an example.

It is also possible to define subpartitions explicitly using `SUBPARTITION` clauses to specify options for individual subpartitions. For example, a more verbose fashion of creating the same table `ts` as shown in the previous example would be:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
            SUBPARTITION s2,
            SUBPARTITION s3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s4,
```

```
            SUBPARTITION s5
        )
    );
```

Some syntactical items of note are listed here:

- Each partition must have the same number of subpartitions.
- If you explicitly define any subpartitions using `SUBPARTITION` on any partition of a partitioned table, you must define them all. In other words, the following statement will fail:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s2,
            SUBPARTITION s3
        )
    );
```

  This statement would still fail even if it used `SUBPARTITIONS 2`.

- Each `SUBPARTITION` clause must include (at a minimum) a name for the subpartition. Otherwise, you may set any desired option for the subpartition or allow it to assume its default setting for that option.
- Subpartition names must be unique across the entire table. For example, the following `CREATE TABLE` statement is valid:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
            SUBPARTITION s2,
            SUBPARTITION s3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s4,
            SUBPARTITION s5
        )
    );
```

23.2.7 How MySQL Partitioning Handles NULL

Partitioning in MySQL does nothing to disallow `NULL` as the value of a partitioning expression, whether it is a column value or the value of a user-supplied expression. Even though it is permitted to use `NULL` as the value of an expression that must otherwise yield an integer, it is important to keep in mind that `NULL` is not a number. MySQL's partitioning implementation treats `NULL` as being less than any non-`NULL` value, just as `ORDER BY` does.

This means that treatment of `NULL` varies between partitioning of different types, and may produce behavior which you do not expect if you are not prepared for it. This being the case, we discuss in this

section how each MySQL partitioning type handles NULL values when determining the partition in which a row should be stored, and provide examples for each.

Handling of NULL with RANGE partitioning. If you insert a row into a table partitioned by RANGE such that the column value used to determine the partition is NULL, the row is inserted into the lowest partition. Consider these two tables in a database named p, created as follows:

```
mysql> CREATE TABLE t1 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY RANGE(c1) (
    ->     PARTITION p0 VALUES LESS THAN (0),
    ->     PARTITION p1 VALUES LESS THAN (10),
    ->     PARTITION p2 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> CREATE TABLE t2 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY RANGE(c1) (
    ->     PARTITION p0 VALUES LESS THAN (-5),
    ->     PARTITION p1 VALUES LESS THAN (0),
    ->     PARTITION p2 VALUES LESS THAN (10),
    ->     PARTITION p3 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (0.09 sec)
```

You can see the partitions created by these two CREATE TABLE statements using the following query against the PARTITIONS table in the INFORMATION_SCHEMA database:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| t1         | p0             |          0 |              0 |           0 |
| t1         | p1             |          0 |              0 |           0 |
| t1         | p2             |          0 |              0 |           0 |
| t2         | p0             |          0 |              0 |           0 |
| t2         | p1             |          0 |              0 |           0 |
| t2         | p2             |          0 |              0 |           0 |
| t2         | p3             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.00 sec)
```

(For more information about this table, see Section 25.17, “The INFORMATION_SCHEMA PARTITIONS Table”.) Now let us populate each of these tables with a single row containing a NULL in the column used as the partitioning key, and verify that the rows were inserted using a pair of SELECT statements:

```
mysql> INSERT INTO t1 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM t1;
+------+--------+
| id   | name   |
```

```
+------+--------+
| NULL | mothra |
+------+--------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
+------+--------+
| id   | name   |
+------+--------+
| NULL | mothra |
+------+--------+
1 row in set (0.00 sec)
```

You can see which partitions are used to store the inserted rows by rerunning the previous query against `INFORMATION_SCHEMA.PARTITIONS` and inspecting the output:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| t1         | p0             |          1 |             20 |          20 |
| t1         | p1             |          0 |              0 |           0 |
| t1         | p2             |          0 |              0 |           0 |
| t2         | p0             |          1 |             20 |          20 |
| t2         | p1             |          0 |              0 |           0 |
| t2         | p2             |          0 |              0 |           0 |
| t2         | p3             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.01 sec)
```

You can also demonstrate that these rows were stored in the lowest-numbered partition of each table by dropping these partitions, and then re-running the `SELECT` statements:

```
mysql> ALTER TABLE t1 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> ALTER TABLE t2 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT * FROM t1;
Empty set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

(For more information on `ALTER TABLE ... DROP PARTITION`, see Section 13.1.9, “ALTER TABLE Syntax”.)

`NULL` is also treated in this way for partitioning expressions that use SQL functions. Suppose that we define a table using a `CREATE TABLE` statement such as this one:

```
CREATE TABLE tndate (
    id INT,
    dt DATE
)
PARTITION BY RANGE( YEAR(dt) ) (
    PARTITION p0 VALUES LESS THAN (1990),
    PARTITION p1 VALUES LESS THAN (2000),
    PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

As with other MySQL functions, `YEAR(NULL)` returns `NULL`. A row with a `dt` column value of `NULL` is treated as though the partitioning expression evaluated to a value less than any other value, and so is inserted into partition `p0`.

Handling of NULL with LIST partitioning. A table that is partitioned by `LIST` admits `NULL` values if and only if one of its partitions is defined using that value-list that contains `NULL`. The converse of this is that a table partitioned by `LIST` which does not explicitly use `NULL` in a value list rejects rows resulting in a `NULL` value for the partitioning expression, as shown in this example:

```
mysql> CREATE TABLE ts1 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    ->     PARTITION p0 VALUES IN (0, 3, 6),
    ->     PARTITION p1 VALUES IN (1, 4, 7),
    ->     PARTITION p2 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts1 VALUES (9, 'mothra');
ERROR 1504 (HY000): Table has no partition for value 9

mysql> INSERT INTO ts1 VALUES (NULL, 'mothra');
ERROR 1504 (HY000): Table has no partition for value NULL
```

Only rows having a `c1` value between `0` and `8` inclusive can be inserted into `ts1`. `NULL` falls outside this range, just like the number `9`. We can create tables `ts2` and `ts3` having value lists containing `NULL`, as shown here:

```
mysql> CREATE TABLE ts2 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    ->     PARTITION p0 VALUES IN (0, 3, 6),
    ->     PARTITION p1 VALUES IN (1, 4, 7),
    ->     PARTITION p2 VALUES IN (2, 5, 8),
    ->     PARTITION p3 VALUES IN (NULL)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE ts3 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    ->     PARTITION p0 VALUES IN (0, 3, 6),
    ->     PARTITION p1 VALUES IN (1, 4, 7, NULL),
    ->     PARTITION p2 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

When defining value lists for partitioning, you can (and should) treat `NULL` just as you would any other value. For example, both `VALUES IN (NULL)` and `VALUES IN (1, 4, 7, NULL)` are valid, as are `VALUES IN (1, NULL, 4, 7)`, `VALUES IN (NULL, 1, 4, 7)`, and so on. You can insert a row having `NULL` for column `c1` into each of the tables `ts2` and `ts3`:

```
mysql> INSERT INTO ts2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO ts3 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
```

By issuing the appropriate query against `INFORMATION_SCHEMA.PARTITIONS`, you can determine which partitions were used to store the rows just inserted (we assume, as in the previous examples, that the partitioned tables were created in the `p` database):

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
    >   FROM INFORMATION_SCHEMA.PARTITIONS
    >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 'ts_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| ts2        | p0             |          0 |              0 |           0 |
| ts2        | p1             |          0 |              0 |           0 |
| ts2        | p2             |          0 |              0 |           0 |
| ts2        | p3             |          1 |             20 |          20 |
| ts3        | p0             |          0 |              0 |           0 |
| ts3        | p1             |          1 |             20 |          20 |
| ts3        | p2             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.01 sec)
```

As shown earlier in this section, you can also verify which partitions were used for storing the rows by deleting these partitions and then performing a `SELECT`.

Handling of NULL with HASH and KEY partitioning. `NULL` is handled somewhat differently for tables partitioned by `HASH` or `KEY`. In these cases, any partition expression that yields a `NULL` value is treated as though its return value were zero. We can verify this behavior by examining the effects on the file system of creating a table partitioned by `HASH` and populating it with a record containing appropriate values. Suppose that you have a table `th` (also in the `p` database) created using the following statement:

```
mysql> CREATE TABLE th (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY HASH(c1)
    -> PARTITIONS 2;
Query OK, 0 rows affected (0.00 sec)
```

The partitions belonging to this table can be viewed using the query shown here:

```
mysql> SELECT TABLE_NAME,PARTITION_NAME,TABLE_ROWS,AVG_ROW_LENGTH,DATA_LENGTH
    >   FROM INFORMATION_SCHEMA.PARTITIONS
    >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME ='th';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| th         | p0             |          0 |              0 |           0 |
| th         | p1             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
2 rows in set (0.00 sec)
```

`TABLE_ROWS` for each partition is 0. Now insert two rows into `th` whose `c1` column values are `NULL` and 0, and verify that these rows were inserted, as shown here:

```
mysql> INSERT INTO th VALUES (NULL, 'mothra'), (0, 'gigan');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM th;
```

```
+------+---------+
| c1   | c2      |
+------+---------+
| NULL | mothra  |
+------+---------+
|    0 | gigan   |
+------+---------+
2 rows in set (0.01 sec)
```

Recall that for any integer *`N`*, the value of `NULL MOD N` is always `NULL`. For tables that are partitioned by `HASH` or `KEY`, this result is treated for determining the correct partition as `0`. Checking the `INFORMATION_SCHEMA.PARTITIONS` table once again, we can see that both rows were inserted into partition `p0`:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
    >   FROM INFORMATION_SCHEMA.PARTITIONS
    >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME ='th';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| th         | p0             |          2 |             20 |          20 |
| th         | p1             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
2 rows in set (0.00 sec)
```

By repeating the last example using `PARTITION BY KEY` in place of `PARTITION BY HASH` in the definition of the table, you can verify that `NULL` is also treated like 0 for this type of partitioning.

23.3 Partition Management

There are a number of ways using SQL statements to modify partitioned tables; it is possible to add, drop, redefine, merge, or split existing partitions using the partitioning extensions to the `ALTER TABLE` statement. There are also ways to obtain information about partitioned tables and partitions. We discuss these topics in the sections that follow.

- For information about partition management in tables partitioned by `RANGE` or `LIST`, see Section 23.3.1, “Management of RANGE and LIST Partitions”.
- For a discussion of managing `HASH` and `KEY` partitions, see Section 23.3.2, “Management of HASH and KEY Partitions”.
- See Section 23.3.5, “Obtaining Information About Partitions”, for a discussion of mechanisms provided in MySQL 8.0 for obtaining information about partitioned tables and partitions.
- For a discussion of performing maintenance operations on partitions, see Section 23.3.4, “Maintenance of Partitions”.

Note

All partitions of a partitioned table must have the same number of subpartitions; it is not possible to change the subpartitioning once the table has been created.

To change a table's partitioning scheme, it is necessary only to use the `ALTER TABLE` statement with a *`partition_options`* option, which has the same syntax as that as used with `CREATE TABLE` for creating a partitioned table; this option (also) always begins with the keywords `PARTITION BY`. Suppose that the following `CREATE TABLE` statement was used to create a table that is partitioned by range:

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (1995),
        PARTITION p2 VALUES LESS THAN (2000),
        PARTITION p3 VALUES LESS THAN (2005)
    );
```

To repartition this table so that it is partitioned by key into two partitions using the `id` column value as the basis for the key, you can use this statement:

```
ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;
```

This has the same effect on the structure of the table as dropping the table and re-creating it using `CREATE TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;`.

`ALTER TABLE ... ENGINE = ...` changes only the storage engine used by the table, and leaves the table's partitioning scheme intact. The statement succeeds only if the target storage engine provides partitioning support. You can use `ALTER TABLE ... REMOVE PARTITIONING` to remove a table's partitioning; see Section 13.1.9, “ALTER TABLE Syntax”.

Important

Only a single `PARTITION BY`, `ADD PARTITION`, `DROP PARTITION`, `REORGANIZE PARTITION`, or `COALESCE PARTITION` clause can be used in a given `ALTER TABLE` statement. If you (for example) wish to drop a partition and reorganize a table's remaining partitions, you must do so in two separate `ALTER TABLE` statements (one using `DROP PARTITION` and then a second one using `REORGANIZE PARTITION`).

You can delete all rows from one or more selected partitions using `ALTER TABLE ... TRUNCATE PARTITION`.

23.3.1 Management of RANGE and LIST Partitions

Adding and dropping of range and list partitions are handled in a similar fashion, so we discuss the management of both sorts of partitioning in this section. For information about working with tables that are partitioned by hash or key, see Section 23.3.2, “Management of HASH and KEY Partitions”.

Dropping a partition from a table that is partitioned by either `RANGE` or by `LIST` can be accomplished using the `ALTER TABLE` statement with the `DROP PARTITION` option. Suppose that you have created a table that is partitioned by range and then populated with 10 records using the following `CREATE TABLE` and `INSERT` statements:

```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
    ->     PARTITION BY RANGE( YEAR(purchased) ) (
    ->         PARTITION p0 VALUES LESS THAN (1990),
    ->         PARTITION p1 VALUES LESS THAN (1995),
    ->         PARTITION p2 VALUES LESS THAN (2000),
    ->         PARTITION p3 VALUES LESS THAN (2005),
    ->         PARTITION p4 VALUES LESS THAN (2010),
    ->         PARTITION p5 VALUES LESS THAN (2015)
    ->     );
Query OK, 0 rows affected (0.28 sec)

mysql> INSERT INTO tr VALUES
    ->     (1, 'desk organiser', '2003-10-15'),
    ->     (2, 'alarm clock', '1997-11-05'),
```

```
    ->     (3, 'chair', '2009-03-10'),
    ->     (4, 'bookcase', '1989-01-10'),
    ->     (5, 'exercise bike', '2014-05-09'),
    ->     (6, 'sofa', '1987-06-05'),
    ->     (7, 'espresso maker', '2011-11-22'),
    ->     (8, 'aquarium', '1992-08-04'),
    ->     (9, 'study desk', '2006-09-16'),
    ->     (10, 'lava lamp', '1998-12-25');
Query OK, 10 rows affected (0.05 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

You can see which items should have been inserted into partition `p2` as shown here:

```
mysql> SELECT * FROM tr
    ->     WHERE purchased BETWEEN '1995-01-01' AND '1999-12-31';
+------+-------------+------------+
| id   | name        | purchased  |
+------+-------------+------------+
|    2 | alarm clock | 1997-11-05 |
|   10 | lava lamp   | 1998-12-25 |
+------+-------------+------------+
2 rows in set (0.00 sec)
```

You can also get this information using partition selection, as shown here:

```
mysql> SELECT * FROM tr PARTITION (p2);
+------+-------------+------------+
| id   | name        | purchased  |
+------+-------------+------------+
|    2 | alarm clock | 1997-11-05 |
|   10 | lava lamp   | 1998-12-25 |
+------+-------------+------------+
2 rows in set (0.00 sec)
```

See Section 23.5, “Partition Selection”, for more information.

To drop the partition named `p2`, execute the following command:

```
mysql> ALTER TABLE tr DROP PARTITION p2;
Query OK, 0 rows affected (0.03 sec)
```

Note

The `NDBCLUSTER` storage engine does not support `ALTER TABLE ... DROP PARTITION`. It does, however, support the other partitioning-related extensions to `ALTER TABLE` that are described in this chapter.

It is very important to remember that, *when you drop a partition, you also delete all the data that was stored in that partition*. You can see that this is the case by re-running the previous `SELECT` query:

```
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '1999-12-31';
Empty set (0.00 sec)
```

Note

`DROP PARTITION` is supported by native partitioning in-place APIs and may be used with `ALGORITHM={COPY|INPLACE}`. `DROP PARTITION` with `ALGORITHM=INPLACE` deletes data stored in the partition and drops the partition.

> However, `DROP PARTITION` with `ALGORITHM=COPY` or `old_alter_table=ON` rebuilds the partitioned table and attempts to move data from the dropped partition to another partition with a compatible `PARTITION ... VALUES` definition. Data that cannot be moved to another partition is deleted.

Because of this, you must have the `DROP` privilege for a table before you can execute `ALTER TABLE ... DROP PARTITION` on that table.

If you wish to drop all data from all partitions while preserving the table definition and its partitioning scheme, use the `TRUNCATE TABLE` statement. (See Section 13.1.37, “TRUNCATE TABLE Syntax”.)

If you intend to change the partitioning of a table *without* losing data, use `ALTER TABLE ... REORGANIZE PARTITION` instead. See below or in Section 13.1.9, “ALTER TABLE Syntax”, for information about `REORGANIZE PARTITION`.

If you now execute a `SHOW CREATE TABLE` statement, you can see how the partitioning makeup of the table has been changed:

```
mysql> SHOW CREATE TABLE tr\G
*************************** 1. row ***************************
       Table: tr
Create Table: CREATE TABLE `tr` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(50) DEFAULT NULL,
  `purchased` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE ( YEAR(purchased))
(PARTITION p0 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (1995) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (2005) ENGINE = InnoDB,
 PARTITION p4 VALUES LESS THAN (2010) ENGINE = InnoDB,
 PARTITION p5 VALUES LESS THAN (2015) ENGINE = InnoDB) */
1 row in set (0.00 sec)
```

When you insert new rows into the changed table with `purchased` column values between `'1995-01-01'` and `'2004-12-31'` inclusive, those rows will be stored in partition `p3`. You can verify this as follows:

```
mysql> INSERT INTO tr VALUES (11, 'pencil holder', '1995-07-12');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '2004-12-31';
+------+----------------+------------+
| id   | name           | purchased  |
+------+----------------+------------+
|    1 | desk organiser | 2003-10-15 |
|   11 | pencil holder  | 1995-07-12 |
+------+----------------+------------+
2 rows in set (0.00 sec)

mysql> ALTER TABLE tr DROP PARTITION p3;
Query OK, 0 rows affected (0.03 sec)

mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '2004-12-31';
Empty set (0.00 sec)
```

The number of rows dropped from the table as a result of `ALTER TABLE ... DROP PARTITION` is not reported by the server as it would be by the equivalent `DELETE` query.

Dropping `LIST` partitions uses exactly the same `ALTER TABLE ... DROP PARTITION` syntax as used for dropping `RANGE` partitions. However, there is one important difference in the effect this has on your use of the table afterward: You can no longer insert into the table any rows having any of the values that were included in the value list defining the deleted partition. (See Section 23.2.2, “LIST Partitioning”, for an example.)

To add a new range or list partition to a previously partitioned table, use the `ALTER TABLE ... ADD PARTITION` statement. For tables which are partitioned by `RANGE`, this can be used to add a new range to the end of the list of existing partitions. Suppose that you have a partitioned table containing membership data for your organization, which is defined as follows:

```
CREATE TABLE members (
    id INT,
    fname VARCHAR(25),
    lname VARCHAR(25),
    dob DATE
)
PARTITION BY RANGE( YEAR(dob) ) (
    PARTITION p0 VALUES LESS THAN (1980),
    PARTITION p1 VALUES LESS THAN (1990),
    PARTITION p2 VALUES LESS THAN (2000)
);
```

Suppose further that the minimum age for members is 16. As the calendar approaches the end of 2015, you realize that you will soon be admitting members who were born in 2000 (and later). You can modify the `members` table to accommodate new members born in the years 2000 to 2010 as shown here:

```
ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (2010));
```

With tables that are partitioned by range, you can use `ADD PARTITION` to add new partitions to the high end of the partitions list only. Trying to add a new partition in this manner between or before existing partitions results in an error as shown here:

```
mysql> ALTER TABLE members
     >     ADD PARTITION (
     >     PARTITION n VALUES LESS THAN (1970));
ERROR 1463 (HY000): VALUES LESS THAN value must be strictly »
   increasing for each partition
```

You can work around this problem by reorganizing the first partition into two new ones that split the range between them, like this:

```
ALTER TABLE members
    REORGANIZE PARTITION p0 INTO (
        PARTITION n0 VALUES LESS THAN (1970),
        PARTITION n1 VALUES LESS THAN (1980)
);
```

Using `SHOW CREATE TABLE` you can see that the `ALTER TABLE` statement has had the desired effect:

```
mysql> SHOW CREATE TABLE members\G
*************************** 1. row ***************************
       Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) DEFAULT NULL,
  `fname` varchar(25) DEFAULT NULL,
  `lname` varchar(25) DEFAULT NULL,
```

```
  `dob` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE ( YEAR(dob))
(PARTITION n0 VALUES LESS THAN (1970) ENGINE = InnoDB,
 PARTITION n1 VALUES LESS THAN (1980) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (2000) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (2010) ENGINE = InnoDB) */
1 row in set (0.00 sec)
```

See also Section 13.1.9.1, “ALTER TABLE Partition Operations”.

You can also use `ALTER TABLE ... ADD PARTITION` to add new partitions to a table that is partitioned by `LIST`. Suppose a table `tt` is defined using the following `CREATE TABLE` statement:

```
CREATE TABLE tt (
    id INT,
    data INT
)
PARTITION BY LIST(data) (
    PARTITION p0 VALUES IN (5, 10, 15),
    PARTITION p1 VALUES IN (6, 12, 18)
);
```

You can add a new partition in which to store rows having the `data` column values `7`, `14`, and `21` as shown:

```
ALTER TABLE tt ADD PARTITION (PARTITION p2 VALUES IN (7, 14, 21));
```

Keep in mind that you *cannot* add a new `LIST` partition encompassing any values that are already included in the value list of an existing partition. If you attempt to do so, an error will result:

```
mysql> ALTER TABLE tt ADD PARTITION
    >     (PARTITION np VALUES IN (4, 8, 12));
ERROR 1465 (HY000): Multiple definition of same constant »
                    in list partitioning
```

Because any rows with the `data` column value `12` have already been assigned to partition `p1`, you cannot create a new partition on table `tt` that includes `12` in its value list. To accomplish this, you could drop `p1`, and add `np` and then a new `p1` with a modified definition. However, as discussed earlier, this would result in the loss of all data stored in `p1`—and it is often the case that this is not what you really want to do. Another solution might appear to be to make a copy of the table with the new partitioning and to copy the data into it using `CREATE TABLE ... SELECT ...`, then drop the old table and rename the new one, but this could be very time-consuming when dealing with a large amounts of data. This also might not be feasible in situations where high availability is a requirement.

You can add multiple partitions in a single `ALTER TABLE ... ADD PARTITION` statement as shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  hired DATE NOT NULL
)
PARTITION BY RANGE( YEAR(hired) ) (
  PARTITION p1 VALUES LESS THAN (1991),
  PARTITION p2 VALUES LESS THAN (1996),
```

```
  PARTITION p3 VALUES LESS THAN (2001),
  PARTITION p4 VALUES LESS THAN (2005)
);

ALTER TABLE employees ADD PARTITION (
    PARTITION p5 VALUES LESS THAN (2010),
    PARTITION p6 VALUES LESS THAN MAXVALUE
);
```

Fortunately, MySQL's partitioning implementation provides ways to redefine partitions without losing data. Let us look first at a couple of simple examples involving `RANGE` partitioning. Recall the `members` table which is now defined as shown here:

```
mysql> SHOW CREATE TABLE members\G
*************************** 1. row ***************************
       Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) DEFAULT NULL,
  `fname` varchar(25) DEFAULT NULL,
  `lname` varchar(25) DEFAULT NULL,
  `dob` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE ( YEAR(dob))
(PARTITION n0 VALUES LESS THAN (1970) ENGINE = InnoDB,
 PARTITION n1 VALUES LESS THAN (1980) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (2000) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (2010) ENGINE = InnoDB) */
1 row in set (0.00 sec)
```

Suppose that you would like to move all rows representing members born before 1960 into a separate partition. As we have already seen, this cannot be done using `ALTER TABLE ... ADD PARTITION`. However, you can use another partition-related extension to `ALTER TABLE` to accomplish this:

```
ALTER TABLE members REORGANIZE PARTITION n0 INTO (
    PARTITION s0 VALUES LESS THAN (1960),
    PARTITION s1 VALUES LESS THAN (1970)
);
```

In effect, this command splits partition `p0` into two new partitions `s0` and `s1`. It also moves the data that was stored in `p0` into the new partitions according to the rules embodied in the two `PARTITION ... VALUES ...` clauses, so that `s0` contains only those records for which `YEAR(dob)` is less than 1960 and `s1` contains those rows in which `YEAR(dob)` is greater than or equal to 1960 but less than 1970.

A `REORGANIZE PARTITION` clause may also be used for merging adjacent partitions. You can reverse the effect of the previous statement on the `members` table as shown here:

```
ALTER TABLE members REORGANIZE PARTITION s0,s1 INTO (
    PARTITION p0 VALUES LESS THAN (1970)
);
```

No data is lost in splitting or merging partitions using `REORGANIZE PARTITION`. In executing the above statement, MySQL moves all of the records that were stored in partitions `s0` and `s1` into partition `p0`.

The general syntax for `REORGANIZE PARTITION` is shown here:

```
ALTER TABLE tbl_name
    REORGANIZE PARTITION partition_list
    INTO (partition_definitions);
```

Here, *`tbl_name`* is the name of the partitioned table, and *`partition_list`* is a comma-separated list of names of one or more existing partitions to be changed. *`partition_definitions`* is a comma-separated list of new partition definitions, which follow the same rules as for the *`partition_definitions`* list used in `CREATE TABLE`. You are not limited to merging several partitions into one, or to splitting one partition into many, when using `REORGANIZE PARTITION`. For example, you can reorganize all four partitions of the `members` table into two, like this:

```
ALTER TABLE members REORGANIZE PARTITION p0,p1,p2,p3 INTO (
    PARTITION m0 VALUES LESS THAN (1980),
    PARTITION m1 VALUES LESS THAN (2000)
);
```

You can also use `REORGANIZE PARTITION` with tables that are partitioned by `LIST`. Let us return to the problem of adding a new partition to the list-partitioned `tt` table and failing because the new partition had a value that was already present in the value-list of one of the existing partitions. We can handle this by adding a partition that contains only nonconflicting values, and then reorganizing the new partition and the existing one so that the value which was stored in the existing one is now moved to the new one:

```
ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8));
ALTER TABLE tt REORGANIZE PARTITION p1,np INTO (
    PARTITION p1 VALUES IN (6, 18),
    PARTITION np VALUES in (4, 8, 12)
);
```

Here are some key points to keep in mind when using `ALTER TABLE ... REORGANIZE PARTITION` to repartition tables that are partitioned by `RANGE` or `LIST`:

- The `PARTITION` options used to determine the new partitioning scheme are subject to the same rules as those used with a `CREATE TABLE` statement.

  A new `RANGE` partitioning scheme cannot have any overlapping ranges; a new `LIST` partitioning scheme cannot have any overlapping sets of values.

- The combination of partitions in the *`partition_definitions`* list should account for the same range or set of values overall as the combined partitions named in the *`partition_list`*.

  For example, partitions `p1` and `p2` together cover the years 1980 through 1999 in the `members` table used as an example in this section. Any reorganization of these two partitions should cover the same range of years overall.

- For tables partitioned by `RANGE`, you can reorganize only adjacent partitions; you cannot skip range partitions.

  For instance, you could not reorganize the example `members` table using a statement beginning with `ALTER TABLE members REORGANIZE PARTITION p0,p2 INTO ...` because `p0` covers the years prior to 1970 and `p2` the years from 1990 through 1999 inclusive, so these are not adjacent partitions. (You cannot skip partition `p1` in this case.)

- You cannot use `REORGANIZE PARTITION` to change the type of partitioning used by the table (for example, you cannot change `RANGE` partitions to `HASH` partitions or the reverse). You also cannot use this statement to change the partitioning expression or column. To accomplish either of these tasks without dropping and re-creating the table, you can use `ALTER TABLE ... PARTITION BY ...`, as shown here:

  ```
  ALTER TABLE members
      PARTITION BY HASH( YEAR(dob) )
  ```

```
    PARTITIONS 8;
```

23.3.2 Management of HASH and KEY Partitions

Tables which are partitioned by hash or by key are very similar to one another with regard to making changes in a partitioning setup, and both differ in a number of ways from tables which have been partitioned by range or list. For that reason, this section addresses the modification of tables partitioned by hash or by key only. For a discussion of adding and dropping of partitions of tables that are partitioned by range or list, see Section 23.3.1, “Management of RANGE and LIST Partitions”.

You cannot drop partitions from tables that are partitioned by `HASH` or `KEY` in the same way that you can from tables that are partitioned by `RANGE` or `LIST`. However, you can merge `HASH` or `KEY` partitions using `ALTER TABLE ... COALESCE PARTITION`. Suppose that a `clients` table containing data about clients is divided into 12 partitions, created as shown here:

```
CREATE TABLE clients (
    id INT,
    fname VARCHAR(30),
    lname VARCHAR(30),
    signed DATE
)
PARTITION BY HASH( MONTH(signed) )
PARTITIONS 12;
```

To reduce the number of partitions from 12 to 8, execute the following `ALTER TABLE` statement:

```
mysql> ALTER TABLE clients COALESCE PARTITION 4;
Query OK, 0 rows affected (0.02 sec)
```

`COALESCE` works equally well with tables that are partitioned by `HASH`, `KEY`, `LINEAR HASH`, or `LINEAR KEY`. Here is an example similar to the previous one, differing only in that the table is partitioned by `LINEAR KEY`:

```
mysql> CREATE TABLE clients_lk (
    ->     id INT,
    ->     fname VARCHAR(30),
    ->     lname VARCHAR(30),
    ->     signed DATE
    -> )
    -> PARTITION BY LINEAR KEY(signed)
    -> PARTITIONS 12;
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE clients_lk COALESCE PARTITION 4;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

The number following `COALESCE PARTITION` is the number of partitions to merge into the remainder—in other words, it is the number of partitions to remove from the table.

Attempting to remove more partitions than are in the table results in an error like this one:

```
mysql> ALTER TABLE clients COALESCE PARTITION 18;
ERROR 1478 (HY000): Cannot remove all partitions, use DROP TABLE instead
```

To increase the number of partitions for the `clients` table from 12 to 18, use `ALTER TABLE ... ADD PARTITION` as shown here:

```
ALTER TABLE clients ADD PARTITION PARTITIONS 6;
```

23.3.3 Exchanging Partitions and Subpartitions with Tables

In MySQL 8.0, it is possible to exchange a table partition or subpartition with a table using `ALTER TABLE pt EXCHANGE PARTITION p WITH TABLE nt`, where `pt` is the partitioned table and `p` is the partition or subpartition of `pt` to be exchanged with unpartitioned table `nt`, provided that the following statements are true:

1. Table `nt` is not itself partitioned.
2. Table `nt` is not a temporary table.
3. The structures of tables `pt` and `nt` are otherwise identical.
4. Table `nt` contains no foreign key references, and no other table has any foreign keys that refer to `nt`.
5. There are no rows in `nt` that lie outside the boundaries of the partition definition for `p`. This condition does not apply if `WITHOUT VALIDATION` is used.
6. For `InnoDB` tables, both tables use the same row format. To determine the row format of an `InnoDB` table, query `INFORMATION_SCHEMA.INNODB_TABLES`.
7. `nt` does not have any partitions that use the `DATA DIRECTORY` option. This restriction is lifted for `InnoDB` tables in MySQL 8.0.14 and later.

In addition to the `ALTER`, `INSERT`, and `CREATE` privileges usually required for `ALTER TABLE` statements, you must have the `DROP` privilege to perform `ALTER TABLE ... EXCHANGE PARTITION`.

You should also be aware of the following effects of `ALTER TABLE ... EXCHANGE PARTITION`:

- Executing `ALTER TABLE ... EXCHANGE PARTITION` does not invoke any triggers on either the partitioned table or the table to be exchanged.
- Any `AUTO_INCREMENT` columns in the exchanged table are reset.
- The `IGNORE` keyword has no effect when used with `ALTER TABLE ... EXCHANGE PARTITION`.

The syntax for `ALTER TABLE ... EXCHANGE PARTITION` is shown here, where `pt` is the partitioned table, `p` is the partition (or subpartition) to be exchanged, and `nt` is the nonpartitioned table to be exchanged with `p`:

```
ALTER TABLE pt
    EXCHANGE PARTITION p
    WITH TABLE nt;
```

Optionally, you can append `WITH VALIDATION` or `WITHOUT VALIDATION`. When `WITHOUT VALIDATION` is specified, the `ALTER TABLE ... EXCHANGE PARTITION` operation does not perform any row-by-row validation when exchanging a partition a nonpartitioned table, allowing database administrators to assume responsibility for ensuring that rows are within the boundaries of the partition definition. `WITH VALIDATION` is the default.

One and only one partition or subpartition may be exchanged with one and only one nonpartitioned table in a single `ALTER TABLE EXCHANGE PARTITION` statement. To exchange multiple partitions or subpartitions, use multiple `ALTER TABLE EXCHANGE PARTITION` statements. `EXCHANGE PARTITION` may not be combined with other `ALTER TABLE` options. The partitioning and (if applicable) subpartitioning used by the partitioned table may be of any type or types supported in MySQL 8.0.

Exchanging a Partition with a Nonpartitioned Table

Suppose that a partitioned table `e` has been created and populated using the following SQL statements:

```
CREATE TABLE e (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30)
)
    PARTITION BY RANGE (id) (
        PARTITION p0 VALUES LESS THAN (50),
        PARTITION p1 VALUES LESS THAN (100),
        PARTITION p2 VALUES LESS THAN (150),
        PARTITION p3 VALUES LESS THAN (MAXVALUE)
);

INSERT INTO e VALUES
    (1669, "Jim", "Smith"),
    (337, "Mary", "Jones"),
    (16, "Frank", "White"),
    (2005, "Linda", "Black");
```

Now we create a nonpartitioned copy of `e` named `e2`. This can be done using the `mysql` client as shown here:

```
mysql> CREATE TABLE e2 LIKE e;
Query OK, 0 rows affected (0.04 sec)

mysql> ALTER TABLE e2 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can see which partitions in table `e` contain rows by querying the `INFORMATION_SCHEMA.PARTITIONS` table, like this:

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS
           FROM INFORMATION_SCHEMA.PARTITIONS
           WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          1 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
2 rows in set (0.00 sec)
```

Note

For partitioned `InnoDB` tables, the row count given in the `TABLE_ROWS` column of the `INFORMATION_SCHEMA.PARTITIONS` table is only an estimated value used in SQL optimization, and is not always exact.

To exchange partition `p0` in table `e` with table `e2`, you can use `ALTER TABLE`, as shown here:

```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
Query OK, 0 rows affected (0.04 sec)
```

More precisely, the statement just issued causes any rows found in the partition to be swapped with those found in the table. You can observe how this has happened by querying the

INFORMATION_SCHEMA.PARTITIONS table, as before. The table row that was previously found in partition p0 is no longer present:

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS
           FROM INFORMATION_SCHEMA.PARTITIONS
           WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
4 rows in set (0.00 sec)
```

If you query table e2, you can see that the “missing” row can now be found there:

```
mysql> SELECT * FROM e2;
+----+-------+-------+
| id | fname | lname |
+----+-------+-------+
| 16 | Frank | White |
+----+-------+-------+
1 row in set (0.00 sec)
```

The table to be exchanged with the partition does not necessarily have to be empty. To demonstrate this, we first insert a new row into table e, making sure that this row is stored in partition p0 by choosing an id column value that is less than 50, and verifying this afterward by querying the PARTITIONS table:

```
mysql> INSERT INTO e VALUES (41, "Michael", "Green");
Query OK, 1 row affected (0.05 sec)

mysql> SELECT PARTITION_NAME, TABLE_ROWS
           FROM INFORMATION_SCHEMA.PARTITIONS
           WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          1 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
4 rows in set (0.00 sec)
```

Now we once again exchange partition p0 with table e2 using the same ALTER TABLE statement as previously:

```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
Query OK, 0 rows affected (0.28 sec)
```

The output of the following queries shows that the table row that was stored in partition p0 and the table row that was stored in table e2, prior to issuing the ALTER TABLE statement, have now switched places:

```
mysql> SELECT * FROM e;
+------+-------+-------+
| id   | fname | lname |
+------+-------+-------+
```

```
|   16 | Frank | White |
| 1669 | Jim   | Smith |
|  337 | Mary  | Jones |
| 2005 | Linda | Black |
+------+-------+-------+
4 rows in set (0.00 sec)

mysql> SELECT PARTITION_NAME, TABLE_ROWS
           FROM INFORMATION_SCHEMA.PARTITIONS
           WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          1 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM e2;
+----+---------+-------+
| id | fname   | lname |
+----+---------+-------+
| 41 | Michael | Green |
+----+---------+-------+
1 row in set (0.00 sec)
```

Nonmatching Rows

You should keep in mind that any rows found in the nonpartitioned table prior to issuing the `ALTER TABLE ... EXCHANGE PARTITION` statement must meet the conditions required for them to be stored in the target partition; otherwise, the statement fails. To see how this occurs, first insert a row into `e2` that is outside the boundaries of the partition definition for partition `p0` of table `e`. For example, insert a row with an `id` column value that is too large; then, try to exchange the table with the partition again:

```
mysql> INSERT INTO e2 VALUES (51, "Ellen", "McDonald");
Query OK, 1 row affected (0.08 sec)

mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
ERROR 1707 (HY000): Found row that does not match the partition
```

Only the `WITHOUT VALIDATION` option would permit this operation to succeed:

```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2 WITHOUT VALIDATION;
Query OK, 0 rows affected (0.02 sec)
```

When a partition is exchanged with a table that contains rows that do not match the partition definition, it is the responsibility of the database administrator to fix the non-matching rows, which can be performed using `REPAIR TABLE` or `ALTER TABLE ... REPAIR PARTITION`.

Exchanging Partitions Without Row-By-Row Validation

To avoid time consuming validation when exchanging a partition with a table that has many rows, it is possible to skip the row-by-row validation step by appending `WITHOUT VALIDATION` to the `ALTER TABLE ... EXCHANGE PARTITION` statement.

The following example compares the difference between execution times when exchanging a partition with a nonpartitioned table, with and without validation. The partitioned table (table `e`) contains two partitions of 1 million rows each. The rows in p0 of table e are removed and p0 is exchanged with a nonpartitioned table

of 1 million rows. The `WITH VALIDATION` operation takes 0.74 seconds. By comparison, the `WITHOUT VALIDATION` operation takes 0.01 seconds.

```
# Create a partitioned table with 1 million rows in each partition

CREATE TABLE e (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30)
)
    PARTITION BY RANGE (id) (
        PARTITION p0 VALUES LESS THAN (1000001),
        PARTITION p1 VALUES LESS THAN (2000001),
);

mysql> SELECT COUNT(*) FROM e;
| COUNT(*) |
+----------+
|  2000000 |
+----------+
1 row in set (0.27 sec)

# View the rows in each partition

SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME = 'e';
+----------------+-------------+
| PARTITION_NAME | TABLE_ROWS  |
+----------------+-------------+
| p0             |     1000000 |
| p1             |     1000000 |
+----------------+-------------+
2 rows in set (0.00 sec)

# Create a nonpartitioned table of the same structure and populate it with 1 million rows

CREATE TABLE e2 (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30)
);

mysql> SELECT COUNT(*) FROM e2;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (0.24 sec)

# Create another nonpartitioned table of the same structure and populate it with 1 million rows

CREATE TABLE e3 (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30)
);

mysql> SELECT COUNT(*) FROM e3;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (0.25 sec)

# Drop the rows from p0 of table e
```

```
mysql> DELETE FROM e WHERE id < 1000001;
Query OK, 1000000 rows affected (5.55 sec)

# Confirm that there are no rows in partition p0

mysql> SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p1             |    1000000 |
+----------------+------------+
2 rows in set (0.00 sec)

# Exchange partition p0 of table e with the table e2 'WITH VALIDATION'

mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2 WITH VALIDATION;
Query OK, 0 rows affected (0.74 sec)

# Confirm that the partition was exchanged with table e2

mysql> SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |    1000000 |
| p1             |    1000000 |
+----------------+------------+
2 rows in set (0.00 sec)

# Once again, drop the rows from p0 of table e

mysql> DELETE FROM e WHERE id < 1000001;
Query OK, 1000000 rows affected (5.55 sec)

# Confirm that there are no rows in partition p0

mysql> SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p1             |    1000000 |
+----------------+------------+
2 rows in set (0.00 sec)

# Exchange partition p0 of table e with the table e3 'WITHOUT VALIDATION'

mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e3 WITHOUT VALIDATION;
Query OK, 0 rows affected (0.01 sec)

# Confirm that the partition was exchanged with table e3

mysql> SELECT PARTITION_NAME, TABLE_ROWS FROM INFORMATION_SCHEMA.PARTITIONS WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |    1000000 |
| p1             |    1000000 |
+----------------+------------+
2 rows in set (0.00 sec)
```

If a partition is exchanged with a table that contains rows that do not match the partition definition, it is the responsibility of the database administrator to fix the non-matching rows, which can be performed using `REPAIR TABLE` or `ALTER TABLE ... REPAIR PARTITION`.

Exchanging a Subpartition with a Nonpartitioned Table

You can also exchange a subpartition of a subpartitioned table (see Section 23.2.6, “Subpartitioning”) with a nonpartitioned table using an `ALTER TABLE ... EXCHANGE PARTITION` statement. In the following example, we first create a table `es` that is partitioned by `RANGE` and subpartitioned by `KEY`, populate this table as we did table `e`, and then create an empty, nonpartitioned copy `es2` of the table, as shown here:

```
mysql> CREATE TABLE es (
    ->     id INT NOT NULL,
    ->     fname VARCHAR(30),
    ->     lname VARCHAR(30)
    -> )
    ->     PARTITION BY RANGE (id)
    ->     SUBPARTITION BY KEY (lname)
    ->     SUBPARTITIONS 2 (
    ->         PARTITION p0 VALUES LESS THAN (50),
    ->         PARTITION p1 VALUES LESS THAN (100),
    ->         PARTITION p2 VALUES LESS THAN (150),
    ->         PARTITION p3 VALUES LESS THAN (MAXVALUE)
    ->     );
Query OK, 0 rows affected (2.76 sec)

mysql> INSERT INTO es VALUES
    ->     (1669, "Jim", "Smith"),
    ->     (337, "Mary", "Jones"),
    ->     (16, "Frank", "White"),
    ->     (2005, "Linda", "Black");
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE es2 LIKE es;
Query OK, 0 rows affected (1.27 sec)

mysql> ALTER TABLE es2 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.70 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Although we did not explicitly name any of the subpartitions when creating table `es`, we can obtain generated names for these by including the `SUBPARTITION_NAME` column of the `PARTITIONS` table from `INFORMATION_SCHEMA` when selecting from that table, as shown here:

```
mysql> SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'es';
+----------------+-------------------+------------+
| PARTITION_NAME | SUBPARTITION_NAME | TABLE_ROWS |
+----------------+-------------------+------------+
| p0             | p0sp0             |          1 |
| p0             | p0sp1             |          0 |
| p1             | p1sp0             |          0 |
| p1             | p1sp1             |          0 |
| p2             | p2sp0             |          0 |
| p2             | p2sp1             |          0 |
| p3             | p3sp0             |          3 |
| p3             | p3sp1             |          0 |
+----------------+-------------------+------------+
8 rows in set (0.00 sec)
```

The following `ALTER TABLE` statement exchanges subpartition `p3sp0` in table `es` with the nonpartitioned table `es2`:

```
mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es2;
Query OK, 0 rows affected (0.29 sec)
```

You can verify that the rows were exchanged by issuing the following queries:

```
mysql> SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'es';
+----------------+-------------------+------------+
| PARTITION_NAME | SUBPARTITION_NAME | TABLE_ROWS |
+----------------+-------------------+------------+
| p0             | p0sp0             |          1 |
| p0             | p0sp1             |          0 |
| p1             | p1sp0             |          0 |
| p1             | p1sp1             |          0 |
| p2             | p2sp0             |          0 |
| p2             | p2sp1             |          0 |
| p3             | p3sp0             |          0 |
| p3             | p3sp1             |          0 |
+----------------+-------------------+------------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM es2;
+------+-------+-------+
| id   | fname | lname |
+------+-------+-------+
| 1669 | Jim   | Smith |
|  337 | Mary  | Jones |
| 2005 | Linda | Black |
+------+-------+-------+
3 rows in set (0.00 sec)
```

If a table is subpartitioned, you can exchange only a subpartition of the table—not an entire partition—with an unpartitioned table, as shown here:

```
mysql> ALTER TABLE es EXCHANGE PARTITION p3 WITH TABLE es2;
ERROR 1704 (HY000): Subpartitioned table, use subpartition instead of partition
```

Table structures are compared in a strict fashion; the number, order, names, and types of columns and indexes of the partitioned table and the nonpartitioned table must match exactly. In addition, both tables must use the same storage engine:

```
mysql> CREATE TABLE es3 LIKE e;
Query OK, 0 rows affected (1.31 sec)

mysql> ALTER TABLE es3 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.53 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE es3\G
*************************** 1. row ***************************
       Table: es3
Create Table: CREATE TABLE `es3` (
  `id` int(11) NOT NULL,
  `fname` varchar(30) DEFAULT NULL,
  `lname` varchar(30) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
1 row in set (0.00 sec)
```

```
mysql> ALTER TABLE es3 ENGINE = MyISAM;
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es3;
ERROR 1497 (HY000): The mix of handlers in the partitions is not allowed in this version of MySQL
```

23.3.4 Maintenance of Partitions

A number of table and partition maintenance tasks can be carried out on partitioned tables using SQL statements intended for such purposes.

Table maintenance of partitioned tables can be accomplished using the statements `CHECK TABLE`, `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `REPAIR TABLE`, which are supported for partitioned tables.

You can use a number of extensions to `ALTER TABLE` for performing operations of this type on one or more partitions directly, as described in the following list:

- **Rebuilding partitions.** Rebuilds the partition; this has the same effect as dropping all records stored in the partition, then reinserting them. This can be useful for purposes of defragmentation.

  Example:

  ```
  ALTER TABLE t1 REBUILD PARTITION p0, p1;
  ```

- **Optimizing partitions.** If you have deleted a large number of rows from a partition or if you have made many changes to a partitioned table with variable-length rows (that is, having `VARCHAR`, `BLOB`, or `TEXT` columns), you can use `ALTER TABLE ... OPTIMIZE PARTITION` to reclaim any unused space and to defragment the partition data file.

  Example:

  ```
  ALTER TABLE t1 OPTIMIZE PARTITION p0, p1;
  ```

  Using `OPTIMIZE PARTITION` on a given partition is equivalent to running `CHECK PARTITION`, `ANALYZE PARTITION`, and `REPAIR PARTITION` on that partition.

  Some MySQL storage engines, including `InnoDB`, do not support per-partition optimization; in these cases, `ALTER TABLE ... OPTIMIZE PARTITION` analyzes and rebuilds the entire table, and causes an appropriate warning to be issued. (Bug #11751825, Bug #42822) Use `ALTER TABLE ... REBUILD PARTITION` and `ALTER TABLE ... ANALYZE PARTITION` instead, to avoid this issue.

- **Analyzing partitions.** This reads and stores the key distributions for partitions.

  Example:

  ```
  ALTER TABLE t1 ANALYZE PARTITION p3;
  ```

- **Repairing partitions.** This repairs corrupted partitions.

  Example:

  ```
  ALTER TABLE t1 REPAIR PARTITION p0,p1;
  ```

Normally, `REPAIR PARTITION` fails when the partition contains duplicate key errors. You can use `ALTER IGNORE TABLE` with this option, in which case all rows that cannot be moved due to the presence of duplicate keys are removed from the partition (Bug #16900947).

- **Checking partitions.** You can check partitions for errors in much the same way that you can use `CHECK TABLE` with nonpartitioned tables.

  Example:

  ```
  ALTER TABLE trb3 CHECK PARTITION p1;
  ```

  This command will tell you whether the data or indexes in partition `p1` of table `t1` are corrupted. If this is the case, use `ALTER TABLE ... REPAIR PARTITION` to repair the partition.

  Normally, `CHECK PARTITION` fails when the partition contains duplicate key errors. You can use `ALTER IGNORE TABLE` with this option, in which case the statement returns the contents of each row in the partition where a duplicate key violation is found. Only the values for the columns in the partitioning expression for the table are reported. (Bug #16900947)

Each of the statements in the list just shown also supports the keyword `ALL` in place of the list of partition names. Using `ALL` causes the statement to act on all partitions in the table.

You can also truncate partitions using `ALTER TABLE ... TRUNCATE PARTITION`. This statement can be used to delete all rows from one or more partitions in much the same way that `TRUNCATE TABLE` deletes all rows from a table.

`ALTER TABLE ... TRUNCATE PARTITION ALL` truncates all partitions in the table.

23.3.5 Obtaining Information About Partitions

This section discusses obtaining information about existing partitions, which can be done in a number of ways. Methods of obtaining such information include the following:

- Using the `SHOW CREATE TABLE` statement to view the partitioning clauses used in creating a partitioned table.
- Using the `SHOW TABLE STATUS` statement to determine whether a table is partitioned.
- Querying the `INFORMATION_SCHEMA.PARTITIONS` table.
- Using the statement `EXPLAIN SELECT` to see which partitions are used by a given `SELECT`.

From MySQL 8.0.16, when insertions, deletions, or updates are made to partitioned tables, the binary log records information about the partition and (if any) the subpartition in which the row event took place. A new row event is created for a modification that takes place in a different partition or subpartition, even if the table involved is the same. So if a transaction involves three partitions or subpartitions, three row events are generated. For an update event, the partition information is recorded for both the “before” image and the “after” image. The partition information is displayed if you specify the `-v` or `--verbose` option when viewing the binary log using `mysqlbinlog`. Partition information is only recorded when row-based logging is in use (`binlog_format=ROW`).

As discussed elsewhere in this chapter, `SHOW CREATE TABLE` includes in its output the `PARTITION BY` clause used to create a partitioned table. For example:

```
mysql> SHOW CREATE TABLE trb3\G
*************************** 1. row ***************************
       Table: trb3
Create Table: CREATE TABLE `trb3` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(50) DEFAULT NULL,
  `purchased` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
/*!50100 PARTITION BY RANGE (YEAR(purchased))
(PARTITION p0 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (1995) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (2000) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (2005) ENGINE = InnoDB) */
0 row in set (0.00 sec)
```

The output from `SHOW TABLE STATUS` for partitioned tables is the same as that for nonpartitioned tables, except that the `Create_options` column contains the string `partitioned`. The `Engine` column contains the name of the storage engine used by all partitions of the table. (See Section 13.7.7.36, “SHOW TABLE STATUS Syntax”, for more information about this statement.)

You can also obtain information about partitions from `INFORMATION_SCHEMA`, which contains a `PARTITIONS` table. See Section 25.17, “The INFORMATION_SCHEMA PARTITIONS Table”.

It is possible to determine which partitions of a partitioned table are involved in a given `SELECT` query using `EXPLAIN`. The `partitions` column in the `EXPLAIN` output lists the partitions from which records would be matched by the query.

Suppose that a table `trb1` is created and populated as follows:

```
CREATE TABLE trb1 (id INT, name VARCHAR(50), purchased DATE)
    PARTITION BY RANGE(id)
    (
        PARTITION p0 VALUES LESS THAN (3),
        PARTITION p1 VALUES LESS THAN (7),
        PARTITION p2 VALUES LESS THAN (9),
        PARTITION p3 VALUES LESS THAN (11)
    );

INSERT INTO trb1 VALUES
    (1, 'desk organiser', '2003-10-15'),
    (2, 'CD player', '1993-11-05'),
    (3, 'TV set', '1996-03-10'),
    (4, 'bookcase', '1982-01-10'),
    (5, 'exercise bike', '2004-05-09'),
    (6, 'sofa', '1987-06-05'),
    (7, 'popcorn maker', '2001-11-22'),
    (8, 'aquarium', '1992-08-04'),
    (9, 'study desk', '1984-09-16'),
    (10, 'lava lamp', '1998-12-25');
```

You can see which partitions are used in a query such as `SELECT * FROM trb1;`, as shown here:

```
mysql> EXPLAIN SELECT * FROM trb1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1,p2,p3
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
```

```
          ref: NULL
         rows: 10
        Extra: Using filesort
```

In this case, all four partitions are searched. However, when a limiting condition making use of the partitioning key is added to the query, you can see that only those partitions containing matching values are scanned, as shown here:

```
mysql> EXPLAIN SELECT * FROM trb1 WHERE id < 5\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 10
        Extra: Using where
```

`EXPLAIN` also provides information about keys used and possible keys:

```
mysql> ALTER TABLE trb1 ADD PRIMARY KEY (id);
Query OK, 10 rows affected (0.03 sec)
Records: 10  Duplicates: 0  Warnings: 0

mysql> EXPLAIN SELECT * FROM trb1 WHERE id < 5\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 7
        Extra: Using where
```

If `EXPLAIN` is used to examine a query against a nonpartitioned table, no error is produced, but the value of the `partitions` column is always `NULL`.

The `rows` column of `EXPLAIN` output displays the total number of rows in the table.

See also Section 13.8.2, “EXPLAIN Syntax”.

23.4 Partition Pruning

The optimization known as *partition pruning* is based on a relatively simple concept which can be described as “Do not scan partitions where there can be no matching values”. Suppose a partitioned table `t1` is created by this statement:

```
CREATE TABLE t1 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
```

```
)
PARTITION BY RANGE( region_code ) (
    PARTITION p0 VALUES LESS THAN (64),
    PARTITION p1 VALUES LESS THAN (128),
    PARTITION p2 VALUES LESS THAN (192),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

Suppose that you wish to obtain results from a `SELECT` statement such as this one:

```
SELECT fname, lname, region_code, dob
    FROM t1
    WHERE region_code > 125 AND region_code < 130;
```

It is easy to see that none of the rows which ought to be returned are in either of the partitions `p0` or `p3`; that is, we need search only in partitions `p1` and `p2` to find matching rows. By limiting the search, it is possible to expend much less time and effort in finding matching rows than by scanning all partitions in the table. This “cutting away” of unneeded partitions is known as *pruning*. When the optimizer can make use of partition pruning in performing this query, execution of the query can be an order of magnitude faster than the same query against a nonpartitioned table containing the same column definitions and data.

The optimizer can perform pruning whenever a `WHERE` condition can be reduced to either one of the following two cases:

- *`partition_column = constant`*
- *`partition_column`* `IN (`*`constant1, constant2, ..., constantN`*`)`

In the first case, the optimizer simply evaluates the partitioning expression for the value given, determines which partition contains that value, and scans only this partition. In many cases, the equal sign can be replaced with another arithmetic comparison, including `<`, `>`, `<=`, `>=`, and `<>`. Some queries using `BETWEEN` in the `WHERE` clause can also take advantage of partition pruning. See the examples later in this section.

In the second case, the optimizer evaluates the partitioning expression for each value in the list, creates a list of matching partitions, and then scans only the partitions in this partition list.

`SELECT`, `DELETE`, and `UPDATE` statements support partition pruning. An `INSERT` statement also accesses only one partition per inserted row; this is true even for a table that is partitioned by `HASH` or `KEY` although this is not currently shown in the output of `EXPLAIN`.

Pruning can also be applied to short ranges, which the optimizer can convert into equivalent lists of values. For instance, in the previous example, the `WHERE` clause can be converted to `WHERE region_code IN (126, 127, 128, 129)`. Then the optimizer can determine that the first two values in the list are found in partition `p1`, the remaining two values in partition `p2`, and that the other partitions contain no relevant values and so do not need to be searched for matching rows.

The optimizer can also perform pruning for `WHERE` conditions that involve comparisons of the preceding types on multiple columns for tables that use `RANGE COLUMNS` or `LIST COLUMNS` partitioning.

This type of optimization can be applied whenever the partitioning expression consists of an equality or a range which can be reduced to a set of equalities, or when the partitioning expression represents an increasing or decreasing relationship. Pruning can also be applied for tables partitioned on a `DATE` or `DATETIME` column when the partitioning expression uses the `YEAR()` or `TO_DAYS()` function. Pruning can also be applied for such tables when the partitioning expression uses the `TO_SECONDS()` function.

Suppose that table `t2`, partitioned on a `DATE` column, is created using the statement shown here:

```
```

```
CREATE TABLE t2 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY RANGE( YEAR(dob) ) (
    PARTITION d0 VALUES LESS THAN (1970),
    PARTITION d1 VALUES LESS THAN (1975),
    PARTITION d2 VALUES LESS THAN (1980),
    PARTITION d3 VALUES LESS THAN (1985),
    PARTITION d4 VALUES LESS THAN (1990),
    PARTITION d5 VALUES LESS THAN (2000),
    PARTITION d6 VALUES LESS THAN (2005),
    PARTITION d7 VALUES LESS THAN MAXVALUE
);
```

The following statements using `t2` can make of use partition pruning:

```
SELECT * FROM t2 WHERE dob = '1982-06-23';

UPDATE t2 SET region_code = 8 WHERE dob BETWEEN '1991-02-15' AND '1997-04-25';

DELETE FROM t2 WHERE dob >= '1984-06-21' AND dob <= '1999-06-21'
```

In the case of the last statement, the optimizer can also act as follows:

1. *Find the partition containing the low end of the range*.

   `YEAR('1984-06-21')` yields the value `1984`, which is found in partition `d3`.

2. *Find the partition containing the high end of the range*.

   `YEAR('1999-06-21')` evaluates to `1999`, which is found in partition `d5`.

3. *Scan only these two partitions and any partitions that may lie between them*.

   In this case, this means that only partitions `d3`, `d4`, and `d5` are scanned. The remaining partitions may be safely ignored (and are ignored).

Important

Invalid `DATE` and `DATETIME` values referenced in the `WHERE` condition of a statement against a partitioned table are treated as `NULL`. This means that a query such as `SELECT * FROM partitioned_table WHERE date_column < '2008-12-00'` does not return any values (see Bug #40972).

So far, we have looked only at examples using `RANGE` partitioning, but pruning can be applied with other partitioning types as well.

Consider a table that is partitioned by `LIST`, where the partitioning expression is increasing or decreasing, such as the table `t3` shown here. (In this example, we assume for the sake of brevity that the `region_code` column is limited to values between 1 and 10 inclusive.)

```
CREATE TABLE t3 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
```

```
PARTITION BY LIST(region_code) (
    PARTITION r0 VALUES IN (1, 3),
    PARTITION r1 VALUES IN (2, 5, 8),
    PARTITION r2 VALUES IN (4, 9),
    PARTITION r3 VALUES IN (6, 7, 10)
);
```

For a statement such as `SELECT * FROM t3 WHERE region_code BETWEEN 1 AND 3`, the optimizer determines in which partitions the values 1, 2, and 3 are found (`r0` and `r1`) and skips the remaining ones (`r2` and `r3`).

For tables that are partitioned by `HASH` or `[LINEAR] KEY`, partition pruning is also possible in cases in which the `WHERE` clause uses a simple `=` relation against a column used in the partitioning expression. Consider a table created like this:

```
CREATE TABLE t4 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY KEY(region_code)
PARTITIONS 8;
```

A statement that compares a column value with a constant can be pruned:

```
UPDATE t4 WHERE region_code = 7;
```

Pruning can also be employed for short ranges, because the optimizer can turn such conditions into `IN` relations. For example, using the same table `t4` as defined previously, queries such as these can be pruned:

```
SELECT * FROM t4 WHERE region_code > 2 AND region_code < 6;

SELECT * FROM t4 WHERE region_code BETWEEN 3 AND 5;
```

In both these cases, the `WHERE` clause is transformed by the optimizer into `WHERE region_code IN (3, 4, 5)`.

Important

This optimization is used only if the range size is smaller than the number of partitions. Consider this statement:

```
DELETE FROM t4 WHERE region_code BETWEEN 4 AND 12;
```

The range in the `WHERE` clause covers 9 values (4, 5, 6, 7, 8, 9, 10, 11, 12), but `t4` has only 8 partitions. This means that the `DELETE` cannot be pruned.

When a table is partitioned by `HASH` or `[LINEAR] KEY`, pruning can be used only on integer columns. For example, this statement cannot use pruning because `dob` is a `DATE` column:

```
SELECT * FROM t4 WHERE dob >= '2001-04-14' AND dob <= '2005-10-15';
```

However, if the table stores year values in an `INT` column, then a query having `WHERE year_col >= 2001 AND year_col <= 2005` can be pruned.

Tables using a storage engine that provides automatic partitioning, such as the NDB storage engine used by MySQL Cluster can be pruned if they are explicitly partitioned.

23.5 Partition Selection

Explicit selection of partitions and subpartitions for rows matching a given WHERE condition is supported. Partition selection is similar to partition pruning, in that only specific partitions are checked for matches, but differs in two key respects:

1. The partitions to be checked are specified by the issuer of the statement, unlike partition pruning, which is automatic.

2. Whereas partition pruning applies only to queries, explicit selection of partitions is supported for both queries and a number of DML statements.

SQL statements supporting explicit partition selection are listed here:

- SELECT
- DELETE
- INSERT
- REPLACE
- UPDATE
- LOAD DATA.
- LOAD XML.

The remainder of this section discusses explicit partition selection as it applies generally to the statements just listed, and provides some examples.

Explicit partition selection is implemented using a PARTITION option. For all supported statements, this option uses the syntax shown here:

```
    PARTITION (partition_names)

    partition_names:
        partition_name, ...
```

This option always follows the name of the table to which the partition or partitions belong. *partition_names* is a comma-separated list of partitions or subpartitions to be used. Each name in this list must be the name of an existing partition or subpartition of the specified table; if any of the partitions or subpartitions are not found, the statement fails with an error (partition 'partition_name' doesn't exist). Partitions and subpartitions named in *partition_names* may be listed in any order, and may overlap.

When the PARTITION option is used, only the partitions and subpartitions listed are checked for matching rows. This option can be used in a SELECT statement to determine which rows belong to a given partition. Consider a partitioned table named employees, created and populated using the statements shown here:

```
SET @@SQL_MODE = '';

CREATE TABLE employees  (
```

```
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    fname VARCHAR(25) NOT NULL,
    lname VARCHAR(25) NOT NULL,
    store_id INT NOT NULL,
    department_id INT NOT NULL
)
    PARTITION BY RANGE(id)  (
        PARTITION p0 VALUES LESS THAN (5),
        PARTITION p1 VALUES LESS THAN (10),
        PARTITION p2 VALUES LESS THAN (15),
        PARTITION p3 VALUES LESS THAN MAXVALUE
);

INSERT INTO employees VALUES
    ('', 'Bob', 'Taylor', 3, 2), ('', 'Frank', 'Williams', 1, 2),
    ('', 'Ellen', 'Johnson', 3, 4), ('', 'Jim', 'Smith', 2, 4),
    ('', 'Mary', 'Jones', 1, 1), ('', 'Linda', 'Black', 2, 3),
    ('', 'Ed', 'Jones', 2, 1), ('', 'June', 'Wilson', 3, 1),
    ('', 'Andy', 'Smith', 1, 3), ('', 'Lou', 'Waters', 2, 4),
    ('', 'Jill', 'Stone', 1, 4), ('', 'Roger', 'White', 3, 2),
    ('', 'Howard', 'Andrews', 1, 2), ('', 'Fred', 'Goldberg', 3, 3),
    ('', 'Barbara', 'Brown', 2, 3), ('', 'Alice', 'Rogers', 2, 2),
    ('', 'Mark', 'Morgan', 3, 3), ('', 'Karen', 'Cole', 3, 2);
```

You can see which rows are stored in partition `p1` like this:

```
mysql> SELECT * FROM employees PARTITION (p1);
+----+-------+--------+----------+---------------+
| id | fname | lname  | store_id | department_id |
+----+-------+--------+----------+---------------+
|  5 | Mary  | Jones  |        1 |             1 |
|  6 | Linda | Black  |        2 |             3 |
|  7 | Ed    | Jones  |        2 |             1 |
|  8 | June  | Wilson |        3 |             1 |
|  9 | Andy  | Smith  |        1 |             3 |
+----+-------+--------+----------+---------------+
5 rows in set (0.00 sec)
```

The result is the same as obtained by the query `SELECT * FROM employees WHERE id BETWEEN 5 AND 9`.

To obtain rows from multiple partitions, supply their names as a comma-delimited list. For example, `SELECT * FROM employees PARTITION (p1, p2)` returns all rows from partitions `p1` and `p2` while excluding rows from the remaining partitions.

Any valid query against a partitioned table can be rewritten with a `PARTITION` option to restrict the result to one or more desired partitions. You can use `WHERE` conditions, `ORDER BY` and `LIMIT` options, and so on. You can also use aggregate functions with `HAVING` and `GROUP BY` options. Each of the following queries produces a valid result when run on the `employees` table as previously defined:

```
mysql> SELECT * FROM employees PARTITION (p0, p2)
    ->     WHERE lname LIKE 'S%';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
|  4 | Jim   | Smith |        2 |             4 |
| 11 | Jill  | Stone |        1 |             4 |
+----+-------+-------+----------+---------------+
2 rows in set (0.00 sec)

mysql> SELECT id, CONCAT(fname, ' ', lname) AS name
    ->     FROM employees PARTITION (p0) ORDER BY lname;
+----+----------------+
```

```
| id | name           |
+----+----------------+
|  3 | Ellen Johnson  |
|  4 | Jim Smith      |
|  1 | Bob Taylor     |
|  2 | Frank Williams |
+----+----------------+
4 rows in set (0.06 sec)

mysql> SELECT store_id, COUNT(department_id) AS c
    ->     FROM employees PARTITION (p1,p2,p3)
    ->     GROUP BY store_id HAVING c > 4;
+---+----------+
| c | store_id |
+---+----------+
| 5 |        2 |
| 5 |        3 |
+---+----------+
2 rows in set (0.00 sec)
```

Statements using partition selection can be employed with tables using any of the supported partitioning types. When a table is created using `[LINEAR] HASH` or `[LINEAR] KEY` partitioning and the names of the partitions are not specified, MySQL automatically names the partitions `p0`, `p1`, `p2`, ..., `p`*`N-1`*, where *`N`* is the number of partitions. For subpartitions not explicitly named, MySQL assigns automatically to the subpartitions in each partition `p`*`X`* the names `p`*`X`*`sp0`, `p`*`X`*`sp1`, `p`*`X`*`sp2`, ..., `p`*`X`*`sp`*`M-1`*, where *`M`* is the number of subpartitions. When executing against this table a `SELECT` (or other SQL statement for which explicit partition selection is allowed), you can use these generated names in a `PARTITION` option, as shown here:

```
mysql> CREATE TABLE employees_sub  (
    ->     id INT NOT NULL AUTO_INCREMENT,
    ->     fname VARCHAR(25) NOT NULL,
    ->     lname VARCHAR(25) NOT NULL,
    ->     store_id INT NOT NULL,
    ->     department_id INT NOT NULL,
    ->     PRIMARY KEY pk (id, lname)
    -> )
    ->     PARTITION BY RANGE(id)
    ->     SUBPARTITION BY KEY (lname)
    ->     SUBPARTITIONS 2 (
    ->         PARTITION p0 VALUES LESS THAN (5),
    ->         PARTITION p1 VALUES LESS THAN (10),
    ->         PARTITION p2 VALUES LESS THAN (15),
    ->         PARTITION p3 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (1.14 sec)

mysql> INSERT INTO employees_sub   # reuse data in employees table
    ->     SELECT * FROM employees;
Query OK, 18 rows affected (0.09 sec)
Records: 18  Duplicates: 0  Warnings: 0

mysql> SELECT id, CONCAT(fname, ' ', lname) AS name
    ->     FROM employees_sub PARTITION (p2sp1);
+----+---------------+
| id | name          |
+----+---------------+
| 10 | Lou Waters    |
| 14 | Fred Goldberg |
+----+---------------+
2 rows in set (0.00 sec)
```

You may also use a `PARTITION` option in the `SELECT` portion of an `INSERT ... SELECT` statement, as shown here:

```
mysql> CREATE TABLE employees_copy LIKE employees;
Query OK, 0 rows affected (0.28 sec)

mysql> INSERT INTO employees_copy
    ->     SELECT * FROM employees PARTITION (p2);
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM employees_copy;
+----+--------+----------+----------+---------------+
| id | fname  | lname    | store_id | department_id |
+----+--------+----------+----------+---------------+
| 10 | Lou    | Waters   |        2 |             4 |
| 11 | Jill   | Stone    |        1 |             4 |
| 12 | Roger  | White    |        3 |             2 |
| 13 | Howard | Andrews  |        1 |             2 |
| 14 | Fred   | Goldberg |        3 |             3 |
+----+--------+----------+----------+---------------+
5 rows in set (0.00 sec)
```

Partition selection can also be used with joins. Suppose we create and populate two tables using the statements shown here:

```
CREATE TABLE stores (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    city VARCHAR(30) NOT NULL
)
    PARTITION BY HASH(id)
    PARTITIONS 2;

INSERT INTO stores VALUES
    ('', 'Nambucca'), ('', 'Uranga'),
    ('', 'Bellingen'), ('', 'Grafton');

CREATE TABLE departments  (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30) NOT NULL
)
    PARTITION BY KEY(id)
    PARTITIONS 2;

INSERT INTO departments VALUES
    ('', 'Sales'), ('', 'Customer Service'),
    ('', 'Delivery'), ('', 'Accounting');
```

You can explicitly select partitions (or subpartitions, or both) from any or all of the tables in a join. (The `PARTITION` option used to select partitions from a given table immediately follows the name of the table, before all other options, including any table alias.) For example, the following query gets the name, employee ID, department, and city of all employees who work in the Sales or Delivery department (partition `p1` of the `departments` table) at the stores in either of the cities of Nambucca and Bellingen (partition `p0` of the `stores` table):

```
mysql> SELECT
    ->     e.id AS 'Employee ID', CONCAT(e.fname, ' ', e.lname) AS Name,
    ->     s.city AS City, d.name AS department
    -> FROM employees AS e
    ->     JOIN stores PARTITION (p1) AS s ON e.store_id=s.id
    ->     JOIN departments PARTITION (p0) AS d ON e.department_id=d.id
    -> ORDER BY e.lname;
+-------------+---------------+-----------+------------+
| Employee ID | Name          | City      | department |
+-------------+---------------+-----------+------------+
```

```
|          14 | Fred Goldberg | Bellingen | Delivery   |
|           5 | Mary Jones    | Nambucca  | Sales      |
|          17 | Mark Morgan   | Bellingen | Delivery   |
|           9 | Andy Smith    | Nambucca  | Delivery   |
|           8 | June Wilson   | Bellingen | Sales      |
+-------------+---------------+-----------+------------+
5 rows in set (0.00 sec)
```

For general information about joins in MySQL, see Section 13.2.10.2, “JOIN Syntax”.

When the `PARTITION` option is used with `DELETE` statements, only those partitions (and subpartitions, if any) listed with the option are checked for rows to be deleted. Any other partitions are ignored, as shown here:

```
mysql> SELECT * FROM employees WHERE fname LIKE 'j%';
+----+-------+--------+----------+---------------+
| id | fname | lname  | store_id | department_id |
+----+-------+--------+----------+---------------+
|  4 | Jim   | Smith  |        2 |             4 |
|  8 | June  | Wilson |        3 |             1 |
| 11 | Jill  | Stone  |        1 |             4 |
+----+-------+--------+----------+---------------+
3 rows in set (0.00 sec)

mysql> DELETE FROM employees PARTITION (p0, p1)
    ->     WHERE fname LIKE 'j%';
Query OK, 2 rows affected (0.09 sec)

mysql> SELECT * FROM employees WHERE fname LIKE 'j%';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
| 11 | Jill  | Stone |        1 |             4 |
+----+-------+-------+----------+---------------+
1 row in set (0.00 sec)
```

Only the two rows in partitions `p0` and `p1` matching the `WHERE` condition were deleted. As you can see from the result when the `SELECT` is run a second time, there remains a row in the table matching the `WHERE` condition, but residing in a different partition (`p2`).

`UPDATE` statements using explicit partition selection behave in the same way; only rows in the partitions referenced by the `PARTITION` option are considered when determining the rows to be updated, as can be seen by executing the following statements:

```
mysql> UPDATE employees PARTITION (p0)
    ->     SET store_id = 2 WHERE fname = 'Jill';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> SELECT * FROM employees WHERE fname = 'Jill';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
| 11 | Jill  | Stone |        1 |             4 |
+----+-------+-------+----------+---------------+
1 row in set (0.00 sec)

mysql> UPDATE employees PARTITION (p2)
    ->     SET store_id = 2 WHERE fname = 'Jill';
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM employees WHERE fname = 'Jill';
```

```
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
| 11 | Jill  | Stone |        2 |             4 |
+----+-------+-------+----------+---------------+
1 row in set (0.00 sec)
```

In the same way, when `PARTITION` is used with `DELETE`, only rows in the partition or partitions named in the partition list are checked for deletion.

For statements that insert rows, the behavior differs in that failure to find a suitable partition causes the statement to fail. This is true for both `INSERT` and `REPLACE` statements, as shown here:

```
mysql> INSERT INTO employees PARTITION (p2) VALUES (20, 'Jan', 'Jones', 1, 3);
ERROR 1729 (HY000): Found a row not matching the given partition set
mysql> INSERT INTO employees PARTITION (p3) VALUES (20, 'Jan', 'Jones', 1, 3);
Query OK, 1 row affected (0.07 sec)

mysql> REPLACE INTO employees PARTITION (p0) VALUES (20, 'Jan', 'Jones', 3, 2);
ERROR 1729 (HY000): Found a row not matching the given partition set

mysql> REPLACE INTO employees PARTITION (p3) VALUES (20, 'Jan', 'Jones', 3, 2);
Query OK, 2 rows affected (0.09 sec)
```

For statements that write multiple rows to a partitioned table that using the `InnoDB` storage engine: If any row in the list following `VALUES` cannot be written to one of the partitions specified in the *`partition_names`* list, the entire statement fails and no rows are written. This is shown for `INSERT` statements in the following example, reusing the `employees` table created previously:

```
mysql> ALTER TABLE employees
    ->     REORGANIZE PARTITION p3 INTO (
    ->         PARTITION p3 VALUES LESS THAN (20),
    ->         PARTITION p4 VALUES LESS THAN (25),
    ->         PARTITION p5 VALUES LESS THAN MAXVALUE
    ->     );
Query OK, 6 rows affected (2.09 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE employees\G
*************************** 1. row ***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `fname` varchar(25) NOT NULL,
  `lname` varchar(25) NOT NULL,
  `store_id` int(11) NOT NULL,
  `department_id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=27 DEFAULT CHARSET=utf8mb4
/*!50100 PARTITION BY RANGE (id)
(PARTITION p0 VALUES LESS THAN (5) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (10) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (15) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (20) ENGINE = InnoDB,
 PARTITION p4 VALUES LESS THAN (25) ENGINE = InnoDB,
 PARTITION p5 VALUES LESS THAN MAXVALUE ENGINE = InnoDB) */
1 row in set (0.00 sec)

mysql> INSERT INTO employees PARTITION (p3, p4) VALUES
    ->     (24, 'Tim', 'Greene', 3, 1),  (26, 'Linda', 'Mills', 2, 1);
ERROR 1729 (HY000): Found a row not matching the given partition set

mysql> INSERT INTO employees PARTITION (p3, p4. p5) VALUES
```

```
    ->     (24, 'Tim', 'Greene', 3, 1),  (26, 'Linda', 'Mills', 2, 1);
Query OK, 2 rows affected (0.06 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

The preceding is true for both `INSERT` statements and `REPLACE` statements that write multiple rows.

Partition selection is disabled for tables employing a storage engine that supplies automatic partitioning, such as `NDB`.

23.6 Restrictions and Limitations on Partitioning

This section discusses current restrictions and limitations on MySQL partitioning support.

Prohibited constructs. The following constructs are not permitted in partitioning expressions:

- Stored procedures, stored functions, UDFs, or plugins.
- Declared variables or user variables.

For a list of SQL functions which are permitted in partitioning expressions, see Section 23.6.3, “Partitioning Limitations Relating to Functions”.

Arithmetic and logical operators. Use of the arithmetic operators `+`, `-`, and `*` is permitted in partitioning expressions. However, the result must be an integer value or `NULL` (except in the case of `[LINEAR] KEY` partitioning, as discussed elsewhere in this chapter; see Section 23.2, “Partitioning Types”, for more information).

The `DIV` operator is also supported; the `/` operator is not permitted.

The bit operators `|`, `&`, `^`, `<<`, `>>`, and `~` are not permitted in partitioning expressions.

Server SQL mode. Tables employing user-defined partitioning do not preserve the SQL mode in effect at the time that they were created. As discussed elsewhere in this Manual (see Section 5.1.11, “Server SQL Modes”), the results of many MySQL functions and operators may change according to the server SQL mode. Therefore, a change in the SQL mode at any time after the creation of partitioned tables may lead to major changes in the behavior of such tables, and could easily lead to corruption or loss of data. For these reasons, *it is strongly recommended that you never change the server SQL mode after creating partitioned tables*.

Examples. The following examples illustrate some changes in behavior of partitioned tables due to a change in the server SQL mode:

1. **Error handling.** As discussed elsewhere, handling of “special” values such as zero and `NULL` can differ between different server SQL modes (see Section 5.1.11, “Server SQL Modes”). For example, `ERROR_FOR_DIVISION_BY_ZERO` can affect whether or not 0 can be inserted as a value into a table whose paritioning expression uses *`column DIV value`* or *`column MOD value`*.

2. **Table accessibility.** Sometimes a change in the server SQL mode can make partitioned tables unusable. The following `CREATE TABLE` statement can be executed successfully only if the `NO_UNSIGNED_SUBTRACTION` mode is in effect:

```
mysql> SELECT @@sql_mode;
+------------+
| @@sql_mode |
+------------+
|            |
+------------+
1 row in set (0.00 sec)
```

```
mysql> CREATE TABLE tu (c1 BIGINT UNSIGNED)
    ->   PARTITION BY RANGE(c1 - 10) (
    ->     PARTITION p0 VALUES LESS THAN (-5),
    ->     PARTITION p1 VALUES LESS THAN (0),
    ->     PARTITION p2 VALUES LESS THAN (5),
    ->     PARTITION p3 VALUES LESS THAN (10),
    ->     PARTITION p4 VALUES LESS THAN (MAXVALUE)
    -> );
ERROR 1563 (HY000): Partition constant is out of partition function domain

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@sql_mode;
+-------------------------+
| @@sql_mode              |
+-------------------------+
| NO_UNSIGNED_SUBTRACTION |
+-------------------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE tu (c1 BIGINT UNSIGNED)
    ->   PARTITION BY RANGE(c1 - 10) (
    ->     PARTITION p0 VALUES LESS THAN (-5),
    ->     PARTITION p1 VALUES LESS THAN (0),
    ->     PARTITION p2 VALUES LESS THAN (5),
    ->     PARTITION p3 VALUES LESS THAN (10),
    ->     PARTITION p4 VALUES LESS THAN (MAXVALUE)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

If you remove the `NO_UNSIGNED_SUBTRACTION` server SQL mode after creating `tu`, you may no longer be able to access this table:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tu;
ERROR 1563 (HY000): Partition constant is out of partition function domain
mysql> INSERT INTO tu VALUES (20);
ERROR 1563 (HY000): Partition constant is out of partition function domain
```

See also Section 5.1.11, “Server SQL Modes”.

Server SQL modes also impact replication of partitioned tables. Disparate SQL modes on master and slave can lead to partitioning expressions being evaluated differently; this can cause the distribution of data among partitions to be different in the master's and slave's copies of a given table, and may even cause inserts into partitioned tables that succeed on the master to fail on the slave. For best results, you should always use the same server SQL mode on the master and on the slave.

Performance considerations. Some effects of partitioning operations on performance are given in the following list:

- **File system operations.** Partitioning and repartitioning operations (such as `ALTER TABLE` with `PARTITION BY ...`, `REORGANIZE PARTITION`, or `REMOVE PARTITIONING`) depend on file system operations for their implementation. This means that the speed of these operations is affected by such factors as file system type and characteristics, disk speed, swap space, file handling efficiency of the operating system, and MySQL server options and variables that relate to file handling. In particular, you should make sure that `large_files_support` is enabled and that `open_files_limit` is set properly. Partitioning and repartitioning operations involving `InnoDB` tables may be made more efficient by enabling `innodb_file_per_table`.

See also Maximum number of partitions.

- **Table locks.** Generally, the process executing a partitioning operation on a table takes a write lock on the table. Reads from such tables are relatively unaffected; pending `INSERT` and `UPDATE` operations are performed as soon as the partitioning operation has completed. For `InnoDB`-specific exceptions to this limitation, see Partitioning Operations.

- **Indexes; partition pruning.** As with nonpartitioned tables, proper use of indexes can speed up queries on partitioned tables significantly. In addition, designing partitioned tables and queries on these tables to take advantage of *partition pruning* can improve performance dramatically. See Section 23.4, “Partition Pruning”, for more information.

  Index condition pushdown is supported for partitioned tables. See Section 8.2.1.6, “Index Condition Pushdown Optimization”.

- **Performance with LOAD DATA.** In MySQL 8.0, `LOAD DATA` uses buffering to improve performance. You should be aware that the buffer uses 130 KB memory per partition to achieve this.

Maximum number of partitions.
The maximum possible number of partitions for a given table not using the `NDB` storage engine is 8192. This number includes subpartitions.

The maximum possible number of user-defined partitions for a table using the `NDB` storage engine is determined according to the version of the NDB Cluster software being used, the number of data nodes, and other factors. See NDB and user-defined partitioning, for more information.

If, when creating tables with a large number of partitions (but less than the maximum), you encounter an error message such as `Got error ... from storage engine: Out of resources when opening file`, you may be able to address the issue by increasing the value of the `open_files_limit` system variable. However, this is dependent on the operating system, and may not be possible or advisable on all platforms; see Section B.4.2.17, “File Not Found and Similar Errors”, for more information. In some cases, using large numbers (hundreds) of partitions may also not be advisable due to other concerns, so using more partitions does not automatically lead to better results.

See also File system operations.

Foreign keys not supported for partitioned InnoDB tables.
Partitioned tables using the `InnoDB` storage engine do not support foreign keys. More specifically, this means that the following two statements are true:

1. No definition of an `InnoDB` table employing user-defined partitioning may contain foreign key references; no `InnoDB` table whose definition contains foreign key references may be partitioned.

2. No `InnoDB` table definition may contain a foreign key reference to a user-partitioned table; no `InnoDB` table with user-defined partitioning may contain columns referenced by foreign keys.

The scope of the restrictions just listed includes all tables that use the `InnoDB` storage engine. `CREATE TABLE` and `ALTER TABLE` statements that would result in tables violating these restrictions are not allowed.

ALTER TABLE ... ORDER BY. An `ALTER TABLE ... ORDER BY` *`column`* statement run against a partitioned table causes ordering of rows only within each partition.

Effects on REPLACE statements by modification of primary keys. It can be desirable in some cases (see Section 23.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”) to modify a table's primary

key. Be aware that, if your application uses REPLACE statements and you do this, the results of these statements can be drastically altered. See Section 13.2.9, “REPLACE Syntax”, for more information and an example.

FULLTEXT indexes.
Partitioned tables do not support FULLTEXT indexes or searches.

Spatial columns. Columns with spatial data types such as POINT or GEOMETRY cannot be used in partitioned tables.

Temporary tables.
Temporary tables cannot be partitioned.

Log tables. It is not possible to partition the log tables; an ALTER TABLE ... PARTITION BY ... statement on such a table fails with an error.

Data type of partitioning key.
A partitioning key must be either an integer column or an expression that resolves to an integer. Expressions employing ENUM columns cannot be used. The column or expression value may also be NULL; see Section 23.2.7, “How MySQL Partitioning Handles NULL”.

There are two exceptions to this restriction:

1. When partitioning by [LINEAR] KEY, it is possible to use columns of any valid MySQL data type other than TEXT or BLOB as partitioning keys, because the internal key-hashing functions produce the correct data type from these types. For example, the following two CREATE TABLE statements are valid:

```
CREATE TABLE tkc (c1 CHAR)
PARTITION BY KEY(c1)
PARTITIONS 4;

CREATE TABLE tke
    ( c1 ENUM('red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet') )
PARTITION BY LINEAR KEY(c1)
PARTITIONS 6;
```

2. When partitioning by RANGE COLUMNS or LIST COLUMNS, it is possible to use string, DATE, and DATETIME columns. For example, each of the following CREATE TABLE statements is valid:

```
CREATE TABLE rc (c1 INT, c2 DATE)
PARTITION BY RANGE COLUMNS(c2) (
    PARTITION p0 VALUES LESS THAN('1990-01-01'),
    PARTITION p1 VALUES LESS THAN('1995-01-01'),
    PARTITION p2 VALUES LESS THAN('2000-01-01'),
    PARTITION p3 VALUES LESS THAN('2005-01-01'),
    PARTITION p4 VALUES LESS THAN(MAXVALUE)
);

CREATE TABLE lc (c1 INT, c2 CHAR(1))
PARTITION BY LIST COLUMNS(c2) (
    PARTITION p0 VALUES IN('a', 'd', 'g', 'j', 'm', 'p', 's', 'v', 'y'),
    PARTITION p1 VALUES IN('b', 'e', 'h', 'k', 'n', 'q', 't', 'w', 'z'),
    PARTITION p2 VALUES IN('c', 'f', 'i', 'l', 'o', 'r', 'u', 'x', NULL)
);
```

Neither of the preceding exceptions applies to BLOB or TEXT column types.

Subqueries.
A partitioning key may not be a subquery, even if that subquery resolves to an integer value or NULL.

Issues with subpartitions.
Subpartitions must use `HASH` or `KEY` partitioning. Only `RANGE` and `LIST` partitions may be subpartitioned; `HASH` and `KEY` partitions cannot be subpartitioned.

`SUBPARTITION BY KEY` requires that the subpartitioning column or columns be specified explicitly, unlike the case with `PARTITION BY KEY`, where it can be omitted (in which case the table's primary key column is used by default). Consider the table created by this statement:

```
CREATE TABLE ts (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30)
);
```

You can create a table having the same columns, partitioned by `KEY`, using a statement such as this one:

```
CREATE TABLE ts (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30)
)
PARTITION BY KEY()
PARTITIONS 4;
```

The previous statement is treated as though it had been written like this, with the table's primary key column used as the partitioning column:

```
CREATE TABLE ts (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30)
)
PARTITION BY KEY(id)
PARTITIONS 4;
```

However, the following statement that attempts to create a subpartitioned table using the default column as the subpartitioning column fails, and the column must be specified for the statement to succeed, as shown here:

```
mysql> CREATE TABLE ts (
    ->     id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(30)
    -> )
    -> PARTITION BY RANGE(id)
    -> SUBPARTITION BY KEY()
    -> SUBPARTITIONS 4
    -> (
    ->     PARTITION p0 VALUES LESS THAN (100),
    ->     PARTITION p1 VALUES LESS THAN (MAXVALUE)
    -> );
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near ')

mysql> CREATE TABLE ts (
    ->     id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(30)
    -> )
    -> PARTITION BY RANGE(id)
    -> SUBPARTITION BY KEY(id)
    -> SUBPARTITIONS 4
    -> (
    ->     PARTITION p0 VALUES LESS THAN (100),
```

```
    ->     PARTITION p1 VALUES LESS THAN (MAXVALUE)
    -> );
Query OK, 0 rows affected (0.07 sec)
```

This is a known issue (see Bug #51470).

DATA DIRECTORY and INDEX DIRECTORY options. Table-level `DATA DIRECTORY` and `INDEX DIRECTORY` options are ignored (see Bug #32091). You can employ these options for individual partitions or subpartitions of `InnoDB` tables.

Repairing and rebuilding partitioned tables. The statements `CHECK TABLE`, `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `REPAIR TABLE` are supported for partitioned tables.

In addition, you can use `ALTER TABLE ... REBUILD PARTITION` to rebuild one or more partitions of a partitioned table; `ALTER TABLE ... REORGANIZE PARTITION` also causes partitions to be rebuilt. See Section 13.1.9, “ALTER TABLE Syntax”, for more information about these two statements.

`ANALYZE`, `CHECK`, `OPTIMIZE`, `REPAIR`, and `TRUNCATE` operations are supported with subpartitions. See Section 13.1.9.1, “ALTER TABLE Partition Operations”.

File name delimiters for partitions and subpartitions. Table partition and subpartition file names include generated delimiters such as `#P#` and `#SP#`. The letter case of such delimiters can vary and should not be depended upon.

23.6.1 Partitioning Keys, Primary Keys, and Unique Keys

This section discusses the relationship of partitioning keys with primary keys and unique keys. The rule governing this relationship can be expressed as follows: All columns used in the partitioning expression for a partitioned table must be part of every unique key that the table may have.

In other words, *every unique key on the table must use every column in the table's partitioning expression*. (This also includes the table's primary key, since it is by definition a unique key. This particular case is discussed later in this section.) For example, each of the following table creation statements is invalid:

```
CREATE TABLE t1 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1),
    UNIQUE KEY (col3)
)
PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

In each case, the proposed table would have at least one unique key that does not include all columns used in the partitioning expression.

Each of the following statements is valid, and represents one way in which the corresponding invalid table creation statement could be made to work:

```
CREATE TABLE t1 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2, col3)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col3)
)
PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

This example shows the error produced in such cases:

```
mysql> CREATE TABLE t3 (
    ->     col1 INT NOT NULL,
    ->     col2 DATE NOT NULL,
    ->     col3 INT NOT NULL,
    ->     col4 INT NOT NULL,
    ->     UNIQUE KEY (col1, col2),
    ->     UNIQUE KEY (col3)
    -> )
    -> PARTITION BY HASH(col1 + col3)
    -> PARTITIONS 4;
ERROR 1491 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

The `CREATE TABLE` statement fails because both `col1` and `col3` are included in the proposed partitioning key, but neither of these columns is part of both of unique keys on the table. This shows one possible fix for the invalid table definition:

```
mysql> CREATE TABLE t3 (
    ->     col1 INT NOT NULL,
    ->     col2 DATE NOT NULL,
    ->     col3 INT NOT NULL,
    ->     col4 INT NOT NULL,
    ->     UNIQUE KEY (col1, col2, col3),
    ->     UNIQUE KEY (col3)
    -> )
    -> PARTITION BY HASH(col3)
    -> PARTITIONS 4;
Query OK, 0 rows affected (0.05 sec)
```

In this case, the proposed partitioning key `col3` is part of both unique keys, and the table creation statement succeeds.

The following table cannot be partitioned at all, because there is no way to include in a partitioning key any columns that belong to both unique keys:

```
CREATE TABLE t4 (
    col1 INT NOT NULL,
    col2 INT NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
```

```
    UNIQUE KEY (col1, col3),
    UNIQUE KEY (col2, col4)
);
```

Since every primary key is by definition a unique key, this restriction also includes the table's primary key, if it has one. For example, the next two statements are invalid:

```
CREATE TABLE t5 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t6 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col3),
    UNIQUE KEY(col2)
)
PARTITION BY HASH( YEAR(col2) )
PARTITIONS 4;
```

In both cases, the primary key does not include all columns referenced in the partitioning expression. However, both of the next two statements are valid:

```
CREATE TABLE t7 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2)
)
PARTITION BY HASH(col1 + YEAR(col2))
PARTITIONS 4;

CREATE TABLE t8 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2, col4),
    UNIQUE KEY(col2, col1)
)
PARTITION BY HASH(col1 + YEAR(col2))
PARTITIONS 4;
```

If a table has no unique keys—this includes having no primary key—then this restriction does not apply, and you may use any column or columns in the partitioning expression as long as the column type is compatible with the partitioning type.

For the same reason, you cannot later add a unique key to a partitioned table unless the key includes all columns used by the table's partitioning expression. Consider the partitioned table created as shown here:

```
mysql> CREATE TABLE t_no_pk (c1 INT, c2 INT)
    ->     PARTITION BY RANGE(c1) (
    ->         PARTITION p0 VALUES LESS THAN (10),
```

```
    ->         PARTITION p1 VALUES LESS THAN (20),
    ->         PARTITION p2 VALUES LESS THAN (30),
    ->         PARTITION p3 VALUES LESS THAN (40)
    ->     );
Query OK, 0 rows affected (0.12 sec)
```

It is possible to add a primary key to `t_no_pk` using either of these `ALTER TABLE` statements:

```
#  possible PK
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c1);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

# drop this PK
mysql> ALTER TABLE t_no_pk DROP PRIMARY KEY;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0

#  use another possible PK
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c1, c2);
Query OK, 0 rows affected (0.12 sec)
Records: 0  Duplicates: 0  Warnings: 0

# drop this PK
mysql> ALTER TABLE t_no_pk DROP PRIMARY KEY;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

However, the next statement fails, because `c1` is part of the partitioning key, but is not part of the proposed primary key:

```
#  fails with error 1503
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c2);
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

Since `t_no_pk` has only `c1` in its partitioning expression, attempting to adding a unique key on `c2` alone fails. However, you can add a unique key that uses both `c1` and `c2`.

These rules also apply to existing nonpartitioned tables that you wish to partition using `ALTER TABLE ... PARTITION BY`. Consider a table `np_pk` created as shown here:

```
mysql> CREATE TABLE np_pk (
    ->     id INT NOT NULL AUTO_INCREMENT,
    ->     name VARCHAR(50),
    ->     added DATE,
    ->     PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.08 sec)
```

The following `ALTER TABLE` statement fails with an error, because the `added` column is not part of any unique key in the table:

```
mysql> ALTER TABLE np_pk
    ->     PARTITION BY HASH( TO_DAYS(added) )
    ->     PARTITIONS 4;
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

However, this statement using the `id` column for the partitioning column is valid, as shown here:

```
mysql> ALTER TABLE np_pk
```

```
    ->     PARTITION BY HASH(id)
    ->     PARTITIONS 4;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

In the case of `np_pk`, the only column that may be used as part of a partitioning expression is `id`; if you wish to partition this table using any other column or columns in the partitioning expression, you must first modify the table, either by adding the desired column or columns to the primary key, or by dropping the primary key altogether.

23.6.2 Partitioning Limitations Relating to Storage Engines

In MySQL 8.0, partitioning support is not actually provided by the MySQL Server, but rather by a table storage engine's own or native partitioning handler. In MySQL 8.0, only the `InnoDB` storage engine provides a native partitioning handler. This means that partitioned tables cannot be created using any other storage engine.

Note

MySQL Cluster's `NDB` storage engine also provides native partitioning support, but is not currently supported in MySQL 8.0.

`ALTER TABLE ... OPTIMIZE PARTITION` does not work correctly with partitioned tables that use `InnoDB`. Use `ALTER TABLE ... REBUILD PARTITION` and `ALTER TABLE ... ANALYZE PARTITION`, instead, for such tables. For more information, see Section 13.1.9.1, “ALTER TABLE Partition Operations”.

User-defined partitioning and the NDB storage engine (NDB Cluster). Partitioning by `KEY` (including `LINEAR KEY`) is the only type of partitioning supported for the `NDB` storage engine. It is not possible under normal circumstances in NDB Cluster to create an NDB Cluster table using any partitioning type other than [`LINEAR`] `KEY`, and attempting to do so fails with an error.

Exception (not for production): It is possible to override this restriction by setting the `new` system variable on NDB Cluster SQL nodes to `ON`. If you choose to do this, you should be aware that tables using partitioning types other than `[LINEAR] KEY` are not supported in production. *In such cases, you can create and use tables with partitioning types other than `KEY` or `LINEAR KEY`, but you do this entirely at your own risk*.

The maximum number of partitions that can be defined for an `NDB` table depends on the number of data nodes and node groups in the cluster, the version of the NDB Cluster software in use, and other factors. See NDB and user-defined partitioning, for more information.

The maximum amount of fixed-size data that can be stored per partition in an `NDB` table is 128 TB. Previously, this was 16 GB.

`CREATE TABLE` and `ALTER TABLE` statements that would cause a user-partitioned `NDB` table not to meet either or both of the following two requirements are not permitted, and fail with an error:

1. The table must have an explicit primary key.

2. All columns listed in the table's partitioning expression must be part of the primary key.

Exception. If a user-partitioned `NDB` table is created using an empty column-list (that is, using `PARTITION BY KEY()` or `PARTITION BY LINEAR KEY()`), then no explicit primary key is required.

Upgrading partitioned tables. When performing an upgrade, tables which are partitioned by `KEY` must be dumped and reloaded. Partitioned tables using storage engines other than `InnoDB` cannot be upgraded

from MySQL 5.7 or earlier to MySQL 8.0 or later; you must either drop the partitioning from such tables with `ALTER TABLE ... REMOVE PARTITIONING` or convert them to `InnoDB` using `ALTER TABLE ... ENGINE=INNODB` prior to the upgrade.

For information about converting `MyISAM` tables to `InnoDB`, see Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”.

23.6.3 Partitioning Limitations Relating to Functions

This section discusses limitations in MySQL Partitioning relating specifically to functions used in partitioning expressions.

Only the MySQL functions shown in the following list are allowed in partitioning expressions:

- `ABS()`
- `CEILING()` (see CEILING() and FLOOR())
- `DATEDIFF()`
- `DAY()`
- `DAYOFMONTH()`
- `DAYOFWEEK()`
- `DAYOFYEAR()`
- `EXTRACT()` (see EXTRACT() function with WEEK specifier)
- `FLOOR()` (see CEILING() and FLOOR())
- `HOUR()`
- `MICROSECOND()`
- `MINUTE()`
- `MOD()`
- `MONTH()`
- `QUARTER()`
- `SECOND()`
- `TIME_TO_SEC()`
- `TO_DAYS()`
- `TO_SECONDS()`
- `UNIX_TIMESTAMP()` (with `TIMESTAMP` columns)
- `WEEKDAY()`
- `YEAR()`
- `YEARWEEK()`

In MySQL 8.0, partition pruning is supported for the `TO_DAYS()`, `TO_SECONDS()`, `YEAR()`, and `UNIX_TIMESTAMP()` functions. See Section 23.4, “Partition Pruning”, for more information.

CEILING() and FLOOR(). Each of these functions returns an integer only if it is passed an argument of an exact numeric type, such as one of the `INT` types or `DECIMAL`. This means, for example, that the following `CREATE TABLE` statement fails with an error, as shown here:

```
mysql> CREATE TABLE t (c FLOAT) PARTITION BY LIST( FLOOR(c) )(
    ->     PARTITION p0 VALUES IN (1,3,5),
    ->     PARTITION p1 VALUES IN (2,4,6)
    -> );
ERROR 1490 (HY000): The PARTITION function returns the wrong type
```

EXTRACT() function with WEEK specifier. The value returned by the `EXTRACT()` function, when used as `EXTRACT(WEEK FROM col)`, depends on the value of the `default_week_format` system variable. For this reason, `EXTRACT()` is not permitted as a partitioning function when it specifies the unit as `WEEK`. (Bug #54483)

See Section 12.6.2, “Mathematical Functions”, for more information about the return types of these functions, as well as Section 11.2, “Numeric Types”.

Chapter 24 Stored Objects

Table of Contents

24.1 Defining Stored Programs 4332
24.2 Using Stored Routines 4333
24.2.1 Stored Routine Syntax 4334
24.2.2 Stored Routines and MySQL Privileges 4335
24.2.3 Stored Routine Metadata 4335
24.2.4 Stored Procedures, Functions, Triggers, and LAST_INSERT_ID() 4335
24.3 Using Triggers 4335
24.3.1 Trigger Syntax and Examples 4336
24.3.2 Trigger Metadata 4340
24.4 Using the Event Scheduler 4340
24.4.1 Event Scheduler Overview 4341
24.4.2 Event Scheduler Configuration 4342
24.4.3 Event Syntax 4344
24.4.4 Event Metadata 4344
24.4.5 Event Scheduler Status 4345
24.4.6 The Event Scheduler and MySQL Privileges 4346
24.5 Using Views 4349
24.5.1 View Syntax 4349
24.5.2 View Processing Algorithms 4349
24.5.3 Updatable and Insertable Views 4351
24.5.4 The View WITH CHECK OPTION Clause 4354
24.5.5 View Metadata 4354
24.6 Stored Object Access Control 4354
24.7 Stored Program Binary Logging 4357
24.8 Restrictions on Stored Programs 4363
24.9 Restrictions on Views 4367

This chapter discusses stored database objects that are defined in terms of SQL code that is stored on the server for later execution.

Stored objects include these object types:

- Stored procedure: An object created with `CREATE PROCEDURE` and invoked using the `CALL` statement. A procedure does not have a return value but can modify its parameters for later inspection by the caller. It can also generate result sets to be returned to the client program.
- Stored function: An object created with `CREATE FUNCTION` and used much like a built-in function. You invoke it in an expression and it returns a value during expression evaluation.
- Trigger: An object created with `CREATE TRIGGER` that is associated with a table. A trigger is activated when a particular event occurs for the table, such as an insert or update.
- Event: An object created with `CREATE EVENT` and invoked by the server according to schedule.
- View: An object created with `CREATE VIEW` that when referenced produces a result set. A view acts as a virtual table.

Terminology used in this document reflects the stored object hierarchy:

- Stored routines include stored procedures and functions.
- Stored programs include stored routines, triggers, and events.
- Stored objects include stored programs and views.

This chapter describes how to use stored objects. The following sections provide additional information about SQL syntax for statements related to these objects, and about object processing:

- For each object type, there are `CREATE`, `ALTER`, and `DROP` statements that control which objects exist and how they are defined. See Section 13.1, “Data Definition Statements”.
- The `CALL` statement is used to invoke stored procedures. See Section 13.2.1, “CALL Syntax”.
- Stored program definitions include a body that may use compound statements, loops, conditionals, and declared variables. See Section 13.6, “Compound-Statement Syntax”.
- Metadata changes to objects referred to by stored programs are detected and cause automatic reparsing of the affected statements when the program is next executed. For more information, see Section 8.10.3, “Caching of Prepared Statements and Stored Programs”.

24.1 Defining Stored Programs

Each stored program contains a body that consists of an SQL statement. This statement may be a compound statement made up of several statements separated by semicolon (`;`) characters. For example, the following stored procedure has a body made up of a `BEGIN ... END` block that contains a `SET` statement and a `REPEAT` loop that itself contains another `SET` statement:

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
  REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END;
```

If you use the `mysql` client program to define a stored program containing semicolon characters, a problem arises. By default, `mysql` itself recognizes the semicolon as a statement delimiter, so you must redefine the delimiter temporarily to cause `mysql` to pass the entire stored program definition to the server.

To redefine the `mysql` delimiter, use the `delimiter` command. The following example shows how to do this for the `dorepeat()` procedure just shown. The delimiter is changed to `//` to enable the entire definition to be passed to the server as a single statement, and then restored to `;` before invoking the procedure. This enables the `;` delimiter used in the procedure body to be passed through to the server rather than being interpreted by `mysql` itself.

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL dorepeat(1000);
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT @x;
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

You can redefine the delimiter to a string other than `//`, and the delimiter can consist of a single character or multiple characters. You should avoid the use of the backslash (`\`) character because that is the escape character for MySQL.

The following is an example of a function that takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use `delimiter` because the function definition contains no internal `;` statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
    -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

24.2 Using Stored Routines

MySQL supports stored routines (procedures and functions). A stored routine is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored routine instead.

Stored routines can be particularly useful in certain situations:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
- When security is paramount. Banks, for example, use stored procedures and functions for all common operations. This provides a consistent and secure environment, and routines can ensure that each operation is properly logged. In such a setup, applications and users would have no access to the database tables directly, but can only execute specific stored routines.

Stored routines can provide improved performance because less information needs to be sent between the server and the client. The tradeoff is that this does increase the load on the database server because more of the work is done on the server side and less is done on the client (application) side. Consider this if many client machines (such as Web servers) are serviced by only one or a few database servers.

Stored routines also enable you to have libraries of functions in the database server. This is a feature shared by modern application languages that enable such design internally (for example, by using classes). Using these client application language features is beneficial for the programmer even outside the scope of database use.

MySQL follows the SQL:2003 syntax for stored routines, which is also used by IBM's DB2. All syntax described here is supported and any limitations and extensions are documented where appropriate.

Additional Resources

- You may find the Stored Procedures User Forum of use when working with stored procedures and functions.
- For answers to some commonly asked questions regarding stored routines in MySQL, see Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”.
- There are some restrictions on the use of stored routines. See Section 24.8, “Restrictions on Stored Programs”.
- Binary logging for stored routines takes place as described in Section 24.7, “Stored Program Binary Logging”.

24.2.1 Stored Routine Syntax

A stored routine is either a procedure or a function. Stored routines are created with the `CREATE PROCEDURE` and `CREATE FUNCTION` statements (see Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”). A procedure is invoked using a `CALL` statement (see Section 13.2.1, “CALL Syntax”), and can only pass back values using output variables. A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar value. The body of a stored routine can use compound statements (see Section 13.6, “Compound-Statement Syntax”).

Stored routines can be dropped with the `DROP PROCEDURE` and `DROP FUNCTION` statements (see Section 13.1.29, “DROP PROCEDURE and DROP FUNCTION Syntax”), and altered with the `ALTER PROCEDURE` and `ALTER FUNCTION` statements (see Section 13.1.7, “ALTER PROCEDURE Syntax”).

A stored procedure or function is associated with a particular database. This has several implications:

- When the routine is invoked, an implicit `USE` *`db_name`* is performed (and undone when the routine terminates). `USE` statements within stored routines are not permitted.
- You can qualify routine names with the database name. This can be used to refer to a routine that is not in the current database. For example, to invoke a stored procedure `p` or function `f` that is associated with the `test` database, you can say `CALL test.p()` or `test.f()`.
- When a database is dropped, all stored routines associated with it are dropped as well.

Stored functions cannot be recursive.

Recursion in stored procedures is permitted but disabled by default. To enable recursion, set the `max_sp_recursion_depth` server system variable to a value greater than zero. Stored procedure recursion increases the demand on thread stack space. If you increase the value of `max_sp_recursion_depth`, it may be necessary to increase thread stack size by increasing the value of `thread_stack` at server startup. See Section 5.1.8, “Server System Variables”, for more information.

MySQL supports a very useful extension that enables the use of regular `SELECT` statements (that is, without using cursors or local variables) inside a stored procedure. The result set of such a query is simply sent directly to the client. Multiple `SELECT` statements generate multiple result sets, so the client must use a MySQL client library that supports multiple result sets. This means the client must use a client library from a version of MySQL at least as recent as 4.1. The client should also specify the `CLIENT_MULTI_RESULTS` option when it connects. For C programs, this can be done with the `mysql_real_connect()` C API function. See Section 28.7.7.54, “mysql_real_connect()”, and Section 28.7.23, “C API Multiple Statement Execution Support”.

24.2.2 Stored Routines and MySQL Privileges

The MySQL grant system takes stored routines into account as follows:

- The `CREATE ROUTINE` privilege is needed to create stored routines.
- The `ALTER ROUTINE` privilege is needed to alter or drop stored routines. This privilege is granted automatically to the creator of a routine if necessary, and dropped from the creator when the routine is dropped.
- The `EXECUTE` privilege is required to execute stored routines. However, this privilege is granted automatically to the creator of a routine if necessary (and dropped from the creator when the routine is dropped). Also, the default `SQL SECURITY` characteristic for a routine is `DEFINER`, which enables users who have access to the database with which the routine is associated to execute the routine.
- If the `automatic_sp_privileges` system variable is 0, the `EXECUTE` and `ALTER ROUTINE` privileges are not automatically granted to and dropped from the routine creator.
- The creator of a routine is the account used to execute the `CREATE` statement for it. This might not be the same as the account named as the `DEFINER` in the routine definition.

24.2.3 Stored Routine Metadata

Metadata about stored routines can be obtained as follows:

- Query the `ROUTINES` table of the `INFORMATION_SCHEMA` database. See Section 25.23, “The INFORMATION_SCHEMA ROUTINES Table”.
- Use the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements to see routine definitions. See Section 13.7.7.9, “SHOW CREATE PROCEDURE Syntax”.
- Use the `SHOW PROCEDURE STATUS` and `SHOW FUNCTION STATUS` statements to see routine characteristics. See Section 13.7.7.28, “SHOW PROCEDURE STATUS Syntax”.

24.2.4 Stored Procedures, Functions, Triggers, and LAST_INSERT_ID()

Within the body of a stored routine (procedure or function) or a trigger, the value of `LAST_INSERT_ID()` changes the same way as for statements executed outside the body of these kinds of objects (see Section 12.15, “Information Functions”). The effect of a stored routine or trigger upon the value of `LAST_INSERT_ID()` that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of `LAST_INSERT_ID()`, the changed value is seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements do not see a changed value.

24.3 Using Triggers

A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Some uses for triggers are to perform checks of values to be inserted into a table or to perform calculations on values involved in an update.

A trigger is defined to activate when a statement inserts, updates, or deletes rows in the associated table. These row operations are trigger events. For example, rows can be inserted by `INSERT` or `LOAD DATA` statements, and an insert trigger activates for each inserted row. A trigger can be set to activate either

before or after the trigger event. For example, you can have a trigger activate before each row that is inserted into a table or after each row that is updated.

Important

MySQL triggers activate only for changes made to tables by SQL statements. This includes changes to base tables that underlie updatable views. Triggers do not activate for changes to tables made by APIs that do not transmit SQL statements to the MySQL Server. This means that triggers are not activated by updates made using the `NDB` API.

Triggers are not activated by changes in `INFORMATION_SCHEMA` or `performance_schema` tables. Those tables are actually views and triggers are not permitted on views.

The following sections describe the syntax for creating and dropping triggers, show some examples of how to use them, and indicate how to obtain trigger metadata.

Additional Resources

- You may find the Triggers User Forum of use when working with triggers.
- For answers to commonly asked questions regarding triggers in MySQL, see Section A.5, “MySQL 8.0 FAQ: Triggers”.
- There are some restrictions on the use of triggers; see Section 24.8, “Restrictions on Stored Programs”.
- Binary logging for triggers takes place as described in Section 24.7, “Stored Program Binary Logging”.

24.3.1 Trigger Syntax and Examples

To create a trigger or drop a trigger, use the `CREATE TRIGGER` or `DROP TRIGGER` statement, described in Section 13.1.22, “CREATE TRIGGER Syntax”, and Section 13.1.34, “DROP TRIGGER Syntax”.

Here is a simple example that associates a trigger with a table, to activate for `INSERT` operations. The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
       FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.01 sec)
```

The `CREATE TRIGGER` statement creates a trigger named `ins_sum` that is associated with the `account` table. It also includes clauses that specify the trigger action time, the triggering event, and what to do when the trigger activates:

- The keyword `BEFORE` indicates the trigger action time. In this case, the trigger activates before each row inserted into the table. The other permitted keyword here is `AFTER`.
- The keyword `INSERT` indicates the trigger event; that is, the type of operation that activates the trigger. In the example, `INSERT` operations cause trigger activation. You can also create triggers for `DELETE` and `UPDATE` operations.
- The statement following `FOR EACH ROW` defines the trigger body; that is, the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering event. In the

example, the trigger body is a simple `SET` that accumulates into a user variable the values inserted into the `amount` column. The statement refers to the column as `NEW.amount` which means “the value of the `amount` column to be inserted into the new row.”

To use the trigger, set the accumulator variable to zero, execute an `INSERT` statement, and then see what value the variable has afterward:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+----------------------+
| Total amount inserted |
+----------------------+
|              1852.48 |
+----------------------+
```

In this case, the value of `@sum` after the `INSERT` statement has executed is `14.98 + 1937.50 - 100`, or `1852.48`.

To destroy the trigger, use a `DROP TRIGGER` statement. You must specify the schema name if the trigger is not in the default schema:

```
mysql> DROP TRIGGER test.ins_sum;
```

If you drop a table, any triggers for the table are also dropped.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

It is possible to define multiple triggers for a given table that have the same trigger event and action time. For example, you can have two `BEFORE UPDATE` triggers for a table. By default, triggers that have the same trigger event and action time activate in the order they were created. To affect trigger order, specify a clause after `FOR EACH ROW` that indicates `FOLLOWS` or `PRECEDES` and the name of an existing trigger that also has the same trigger event and action time. With `FOLLOWS`, the new trigger activates after the existing trigger. With `PRECEDES`, the new trigger activates before the existing trigger.

For example, the following trigger definition defines another `BEFORE INSERT` trigger for the `account` table:

```
mysql> CREATE TRIGGER ins_transaction BEFORE INSERT ON account
       FOR EACH ROW PRECEDES ins_sum
       SET
       @deposits = @deposits + IF(NEW.amount>0,NEW.amount,0),
       @withdrawals = @withdrawals + IF(NEW.amount<0,-NEW.amount,0);
Query OK, 0 rows affected (0.01 sec)
```

This trigger, `ins_transaction`, is similar to `ins_sum` but accumulates deposits and withdrawals separately. It has a `PRECEDES` clause that causes it to activate before `ins_sum`; without that clause, it would activate after `ins_sum` because it is created after `ins_sum`.

Within the trigger body, the `OLD` and `NEW` keywords enable you to access columns in the rows affected by a trigger. `OLD` and `NEW` are MySQL extensions to triggers; they are not case-sensitive.

In an `INSERT` trigger, only `NEW.col_name` can be used; there is no old row. In a `DELETE` trigger, only `OLD.col_name` can be used; there is no new row. In an `UPDATE` trigger, you can use `OLD.col_name` to

refer to the columns of a row before it is updated and `NEW.col_name` to refer to the columns of the row after it is updated.

A column named with `OLD` is read only. You can refer to it (if you have the `SELECT` privilege), but not modify it. You can refer to a column named with `NEW` if you have the `SELECT` privilege for it. In a `BEFORE` trigger, you can also change its value with `SET NEW.col_name = value` if you have the `UPDATE` privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or used to update a row. (Such a `SET` statement has no effect in an `AFTER` trigger because the row change will have already occurred.)

In a `BEFORE` trigger, the `NEW` value for an `AUTO_INCREMENT` column is 0, not the sequence number that is generated automatically when the new row actually is inserted.

By using the `BEGIN ... END` construct, you can define a trigger that executes multiple statements. Within the `BEGIN` block, you also can use other syntax that is permitted within stored routines such as conditionals and loops. However, just as for stored routines, if you use the `mysql` program to define a trigger that executes multiple statements, it is necessary to redefine the `mysql` statement delimiter so that you can use the `;` statement delimiter within the trigger definition. The following example illustrates these points. It defines an `UPDATE` trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a `BEFORE` trigger because the value must be checked before it is used to update the row:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
       FOR EACH ROW
       BEGIN
           IF NEW.amount < 0 THEN
               SET NEW.amount = 0;
           ELSEIF NEW.amount > 100 THEN
               SET NEW.amount = 100;
           END IF;
       END;//
mysql> delimiter ;
```

It can be easier to define a stored procedure separately and then invoke it from the trigger using a simple `CALL` statement. This is also advantageous if you want to execute the same code from within several triggers.

There are limitations on what can appear in statements that a trigger executes when activated:

- The trigger cannot use the `CALL` statement to invoke stored procedures that return data to the client or that use dynamic SQL. (Stored procedures are permitted to return data to the trigger through `OUT` or `INOUT` parameters.)
- The trigger cannot use statements that explicitly or implicitly begin or end a transaction, such as `START TRANSACTION`, `COMMIT`, or `ROLLBACK`. (`ROLLBACK to SAVEPOINT` is permitted because it does not end a transaction.).

See also Section 24.8, “Restrictions on Stored Programs”.

MySQL handles errors during trigger execution as follows:

- If a `BEFORE` trigger fails, the operation on the corresponding row is not performed.
- A `BEFORE` trigger is activated by the *attempt* to insert or modify the row, regardless of whether the attempt subsequently succeeds.
- An `AFTER` trigger is executed only if any `BEFORE` triggers and the row operation execute successfully.

- An error during either a `BEFORE` or `AFTER` trigger results in failure of the entire statement that caused trigger invocation.
- For transactional tables, failure of a statement should cause rollback of all changes performed by the statement. Failure of a trigger causes the statement to fail, so trigger failure also causes rollback. For nontransactional tables, such rollback cannot be done, so although the statement fails, any changes performed prior to the point of the error remain in effect.

Triggers can contain direct references to tables by name, such as the trigger named `testref` shown in this example:

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);

delimiter |

CREATE TRIGGER testref BEFORE INSERT ON test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
  END;
|

delimiter ;

INSERT INTO test3 (a3) VALUES
  (NULL), (NULL), (NULL), (NULL), (NULL),
  (NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
  (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
```

Suppose that you insert the following values into table `test1` as shown here:

```
mysql> INSERT INTO test1 VALUES
       (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

As a result, the four tables contain the following data:

```
mysql> SELECT * FROM test1;
+------+
| a1   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
```

```
8 rows in set (0.00 sec)

mysql> SELECT * FROM test2;
+------+
| a2   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test3;
+----+
| a3 |
+----+
|  2 |
|  5 |
|  6 |
|  9 |
| 10 |
+----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM test4;
+----+------+
| a4 | b4   |
+----+------+
|  1 |    3 |
|  2 |    0 |
|  3 |    1 |
|  4 |    2 |
|  5 |    0 |
|  6 |    0 |
|  7 |    1 |
|  8 |    1 |
|  9 |    0 |
| 10 |    0 |
+----+------+
10 rows in set (0.00 sec)
```

24.3.2 Trigger Metadata

Metadata about triggers can be obtained as follows:

- Query the `TRIGGERS` table of the `INFORMATION_SCHEMA` database. See Section 25.34, “The INFORMATION_SCHEMA TRIGGERS Table”.
- Use the `SHOW CREATE TRIGGER` statement. See Section 13.7.7.11, “SHOW CREATE TRIGGER Syntax”.
- Use the `SHOW TRIGGERS` statement. See Section 13.7.7.38, “SHOW TRIGGERS Syntax”.

24.4 Using the Event Scheduler

The *MySQL Event Scheduler* manages the scheduling and execution of events, that is, tasks that run according to a schedule. The following discussion covers the Event Scheduler and is divided into the following sections:

- Section 24.4.1, “Event Scheduler Overview”, provides an introduction to and conceptual overview of MySQL Events.
- Section 24.4.3, “Event Syntax”, discusses the SQL statements for creating, altering, and dropping MySQL Events.
- Section 24.4.4, “Event Metadata”, shows how to obtain information about events and how this information is stored by the MySQL Server.
- Section 24.4.6, “The Event Scheduler and MySQL Privileges”, discusses the privileges required to work with events and the ramifications that events have with regard to privileges when executing.

Stored routines require the `events` data dictionary table in the `mysql` system database. This table is created during the MySQL 8.0 installation procedure. If you are upgrading to MySQL 8.0 from an earlier version, be sure to perform the upgrade procedure to make sure that your system database is up to date. See Section 2.11, “Upgrading MySQL”.

Additional Resources

- You may find the MySQL Event Scheduler User Forum of use when working with scheduled events.
- There are some restrictions on the use of events; see Section 24.8, “Restrictions on Stored Programs”.
- Binary logging for events takes place as described in Section 24.7, “Stored Program Binary Logging”.

24.4.1 Event Scheduler Overview

MySQL Events are tasks that run according to a schedule. Therefore, we sometimes refer to them as *scheduled* events. When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time. Conceptually, this is similar to the idea of the Unix `crontab` (also known as a “cron job”) or the Windows Task Scheduler.

Scheduled tasks of this type are also sometimes known as “temporal triggers”, implying that these are objects that are triggered by the passage of time. While this is essentially correct, we prefer to use the term *events* to avoid confusion with triggers of the type discussed in Section 24.3, “Using Triggers”. Events should more specifically not be confused with “temporary triggers”. Whereas a trigger is a database object whose statements are executed in response to a specific type of event that occurs on a given table, a (scheduled) event is an object whose statements are executed in response to the passage of a specified time interval.

While there is no provision in the SQL Standard for event scheduling, there are precedents in other database systems, and you may notice some similarities between these implementations and that found in the MySQL Server.

MySQL Events have the following major features and properties:

- In MySQL, an event is uniquely identified by its name and the schema to which it is assigned.
- An event performs a specific action according to a schedule. This action consists of an SQL statement, which can be a compound statement in a `BEGIN ... END` block if desired (see Section 13.6, “Compound-Statement Syntax”). An event's timing can be either *one-time* or *recurrent*. A one-time event executes one time only. A recurrent event repeats its action at a regular interval, and the schedule for a recurring event can be assigned a specific start day and time, end day and time, both, or neither. (By default, a recurring event's schedule begins as soon as it is created, and continues indefinitely, until it is disabled or dropped.)

If a repeating event does not terminate within its scheduling interval, the result may be multiple instances of the event executing simultaneously. If this is undesirable, you should institute a mechanism to prevent simultaneous instances. For example, you could use the `GET_LOCK()` function, or row or table locking.

- Users can create, modify, and drop scheduled events using SQL statements intended for these purposes. Syntactically invalid event creation and modification statements fail with an appropriate error message. *A user may include statements in an event's action which require privileges that the user does not actually have*. The event creation or modification statement succeeds but the event's action fails. See Section 24.4.6, “The Event Scheduler and MySQL Privileges” for details.

- Many of the properties of an event can be set or modified using SQL statements. These properties include the event's name, timing, persistence (that is, whether it is preserved following the expiration of its schedule), status (enabled or disabled), action to be performed, and the schema to which it is assigned. See Section 13.1.3, “ALTER EVENT Syntax”.

  The default definer of an event is the user who created the event, unless the event has been altered, in which case the definer is the user who issued the last `ALTER EVENT` statement affecting that event. An event can be modified by any user having the `EVENT` privilege on the database for which the event is defined. See Section 24.4.6, “The Event Scheduler and MySQL Privileges”.

- An event's action statement may include most SQL statements permitted within stored routines. For restrictions, see Section 24.8, “Restrictions on Stored Programs”.

24.4.2 Event Scheduler Configuration

Events are executed by a special *event scheduler thread*; when we refer to the Event Scheduler, we actually refer to this thread. When running, the event scheduler thread and its current state can be seen by users having the `PROCESS` privilege in the output of `SHOW PROCESSLIST`, as shown in the discussion that follows.

The global `event_scheduler` system variable determines whether the Event Scheduler is enabled and running on the server. It has one of these 3 values, which affect event scheduling as described here. The default is `ON`.

- `ON`: The Event Scheduler is started; the event scheduler thread runs and executes all scheduled events.

  When the Event Scheduler is `ON`, the event scheduler thread is listed in the output of `SHOW PROCESSLIST` as a daemon process, and its state is represented as shown here:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 1
   User: root
   Host: localhost
     db: NULL
Command: Query
   Time: 0
  State: NULL
   Info: show processlist
*************************** 2. row ***************************
     Id: 2
   User: event_scheduler
   Host: localhost
     db: NULL
Command: Daemon
   Time: 3
  State: Waiting for next activation
```

```
   Info: NULL
2 rows in set (0.00 sec)
```

Event scheduling can be stopped by setting the value of `event_scheduler` to `OFF`.

- `OFF`: The Event Scheduler is stopped. The event scheduler thread does not run, is not shown in the output of `SHOW PROCESSLIST`, and no scheduled events are executed.

  When the Event Scheduler is stopped (`event_scheduler` is `OFF`), it can be started by setting the value of `event_scheduler` to `ON`. (See next item.)

- `DISABLED`: This value renders the Event Scheduler nonoperational. When the Event Scheduler is `DISABLED`, the event scheduler thread does not run (and so does not appear in the output of `SHOW PROCESSLIST`). In addition, the Event Scheduler state cannot be changed at runtime.

If the Event Scheduler status has not been set to `DISABLED`, `event_scheduler` can be toggled between `ON` and `OFF` (using `SET`). It is also possible to use `0` for `OFF`, and `1` for `ON` when setting this variable. Thus, any of the following 4 statements can be used in the `mysql` client to turn on the Event Scheduler:

```
SET GLOBAL event_scheduler = ON;
SET @@GLOBAL.event_scheduler = ON;
SET GLOBAL event_scheduler = 1;
SET @@GLOBAL.event_scheduler = 1;
```

Similarly, any of these 4 statements can be used to turn off the Event Scheduler:

```
SET GLOBAL event_scheduler = OFF;
SET @@GLOBAL.event_scheduler = OFF;
SET GLOBAL event_scheduler = 0;
SET @@GLOBAL.event_scheduler = 0;
```

Although `ON` and `OFF` have numeric equivalents, the value displayed for `event_scheduler` by `SELECT` or `SHOW VARIABLES` is always one of `OFF`, `ON`, or `DISABLED`. *`DISABLED` has no numeric equivalent*. For this reason, `ON` and `OFF` are usually preferred over `1` and `0` when setting this variable.

Note that attempting to set `event_scheduler` without specifying it as a global variable causes an error:

```
mysql< SET @@event_scheduler = OFF;
ERROR 1229 (HY000): Variable 'event_scheduler' is a GLOBAL
variable and should be set with SET GLOBAL
```

Important

It is possible to set the Event Scheduler to `DISABLED` only at server startup. If `event_scheduler` is `ON` or `OFF`, you cannot set it to `DISABLED` at runtime. Also, if the Event Scheduler is set to `DISABLED` at startup, you cannot change the value of `event_scheduler` at runtime.

To disable the event scheduler, use one of the following two methods:

- As a command-line option when starting the server:

  ```
  --event-scheduler=DISABLED
  ```

- In the server configuration file (`my.cnf`, or `my.ini` on Windows systems), include the line where it will be read by the server (for example, in a `[mysqld]` section):

```
event_scheduler=DISABLED
```

To enable the Event Scheduler, restart the server without the `--event-scheduler=DISABLED` command-line option, or after removing or commenting out the line containing `event-scheduler=DISABLED` in the server configuration file, as appropriate. Alternatively, you can use `ON` (or `1`) or `OFF` (or `0`) in place of the `DISABLED` value when starting the server.

Note

You can issue event-manipulation statements when `event_scheduler` is set to `DISABLED`. No warnings or errors are generated in such cases (provided that the statements are themselves valid). However, scheduled events cannot execute until this variable is set to `ON` (or `1`). Once this has been done, the event scheduler thread executes all events whose scheduling conditions are satisfied.

Starting the MySQL server with the `--skip-grant-tables` option causes `event_scheduler` to be set to `DISABLED`, overriding any other value set either on the command line or in the `my.cnf` or `my.ini` file (Bug #26807).

For SQL statements used to create, alter, and drop events, see Section 24.4.3, “Event Syntax”.

MySQL provides an `EVENTS` table in the `INFORMATION_SCHEMA` database. This table can be queried to obtain information about scheduled events which have been defined on the server. See Section 24.4.4, “Event Metadata”, and Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”, for more information.

For information regarding event scheduling and the MySQL privilege system, see Section 24.4.6, “The Event Scheduler and MySQL Privileges”.

24.4.3 Event Syntax

MySQL provides several SQL statements for working with scheduled events:

- New events are defined using the `CREATE EVENT` statement. See Section 13.1.13, “CREATE EVENT Syntax”.
- The definition of an existing event can be changed by means of the `ALTER EVENT` statement. See Section 13.1.3, “ALTER EVENT Syntax”.
- When a scheduled event is no longer wanted or needed, it can be deleted from the server by its definer using the `DROP EVENT` statement. See Section 13.1.25, “DROP EVENT Syntax”. Whether an event persists past the end of its schedule also depends on its `ON COMPLETION` clause, if it has one. See Section 13.1.13, “CREATE EVENT Syntax”.

  An event can be dropped by any user having the `EVENT` privilege for the database on which the event is defined. See Section 24.4.6, “The Event Scheduler and MySQL Privileges”.

24.4.4 Event Metadata

Metadata about events can be obtained as follows:

- Query the `EVENTS` table of the `INFORMATION_SCHEMA` database. See Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”.
- Use the `SHOW CREATE EVENT` statement. See Section 13.7.7.7, “SHOW CREATE EVENT Syntax”.

- Use the `SHOW EVENTS` statement. See Section 13.7.7.18, “SHOW EVENTS Syntax”.

Event Scheduler Time Representation

Each session in MySQL has a session time zone (STZ). This is the session `time_zone` value that is initialized from the server's global `time_zone` value when the session begins but may be changed during the session.

The session time zone that is current when a `CREATE EVENT` or `ALTER EVENT` statement executes is used to interpret times specified in the event definition. This becomes the event time zone (ETZ); that is, the time zone that is used for event scheduling and is in effect within the event as it executes.

For representation of event information in the data dictionary, the `execute_at`, `starts`, and `ends` times are converted to UTC and stored along with the event time zone. This enables event execution to proceed as defined regardless of any subsequent changes to the server time zone or daylight saving time effects. The `last_executed` time is also stored in UTC.

Event times can be obtained by selecting from the `INFORMATION_SCHEMA.EVENTS` table or from `SHOW EVENTS`, but they are reported as ETZ or STZ values. The following table summarizes representation of event times.

Value	`INFORMATION_SCHEMA.EVENTS`	`SHOW EVENTS`
Execute at	ETZ	ETZ
Starts	ETZ	ETZ
Ends	ETZ	ETZ
Last executed	ETZ	n/a
Created	STZ	n/a
Last altered	STZ	n/a

24.4.5 Event Scheduler Status

The Event Scheduler writes information about event execution that terminates with an error or warning to the MySQL Server's error log. See Section 24.4.6, “The Event Scheduler and MySQL Privileges” for an example.

To obtain information about the state of the Event Scheduler for debugging and troubleshooting purposes, run `mysqladmin debug` (see Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”); after running this command, the server's error log contains output relating to the Event Scheduler, similar to what is shown here:

```
Events status:
LLA = Last Locked At  LUA = Last Unlocked At
WOC = Waiting On Condition  DL = Data Locked

Event scheduler status:
State      : INITIALIZED
Thread id  : 0
LLA        : init_scheduler:313
LUA        : init_scheduler:318
WOC        : NO
Workers    : 0
Executed   : 0
Data locked: NO
```

```
Event queue status:
Element count   : 1
Data locked     : NO
Attempting lock : NO
LLA             : init_queue:148
LUA             : init_queue:168
WOC             : NO
Next activation : 0000-00-00 00:00:00
```

In statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log. For frequently executed events, it is possible for this to result in many logged messages. For example, for `SELECT ... INTO var_list` statements, if the query returns no rows, a warning with error code 1329 occurs (`No data`), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (`Result consisted of more than one row`). For either condition, you can avoid having the warnings be logged by declaring a condition handler; see Section 13.6.7.2, "DECLARE ... HANDLER Syntax". For statements that may retrieve multiple rows, another strategy is to use `LIMIT 1` to limit the result set to a single row.

24.4.6 The Event Scheduler and MySQL Privileges

To enable or disable the execution of scheduled events, it is necessary to set the value of the global `event_scheduler` system variable. This requires privileges sufficient to set global system variables. See Section 5.1.9.1, "System Variable Privileges".

The `EVENT` privilege governs the creation, modification, and deletion of events. This privilege can be bestowed using `GRANT`. For example, this `GRANT` statement confers the `EVENT` privilege for the schema named `myschema` on the user `jon@ghidora`:

```
GRANT EVENT ON myschema.* TO jon@ghidora;
```

(We assume that this user account already exists, and that we wish for it to remain unchanged otherwise.)

To grant this same user the `EVENT` privilege on all schemas, use the following statement:

```
GRANT EVENT ON *.* TO jon@ghidora;
```

The `EVENT` privilege has global or schema-level scope. Therefore, trying to grant it on a single table results in an error as shown:

```
mysql> GRANT EVENT ON myschema.mytable TO jon@ghidora;
ERROR 1144 (42000): Illegal GRANT/REVOKE command; please
consult the manual to see which privileges can be used
```

It is important to understand that an event is executed with the privileges of its definer, and that it cannot perform any actions for which its definer does not have the requisite privileges. For example, suppose that `jon@ghidora` has the `EVENT` privilege for `myschema`. Suppose also that this user has the `SELECT` privilege for `myschema`, but no other privileges for this schema. It is possible for `jon@ghidora` to create a new event such as this one:

```
CREATE EVENT e_store_ts
    ON SCHEDULE
      EVERY 10 SECOND
    DO
```

```
      INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
```

The user waits for a minute or so, and then performs a `SELECT * FROM mytable;` query, expecting to see several new rows in the table. Instead, the table is empty. Since the user does not have the `INSERT` privilege for the table in question, the event has no effect.

If you inspect the MySQL error log (*`hostname`*`.err`), you can see that the event is executing, but the action it is attempting to perform fails:

```
2013-09-24T12:41:31.261992Z 25 [ERROR] Event Scheduler:
[jon@ghidora][cookbook.e_store_ts] INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
2013-09-24T12:41:31.262022Z 25 [Note] Event Scheduler:
[jon@ghidora].[myschema.e_store_ts] event execution failed.
2013-09-24T12:41:41.271796Z 26 [ERROR] Event Scheduler:
[jon@ghidora][cookbook.e_store_ts] INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
2013-09-24T12:41:41.272761Z 26 [Note] Event Scheduler:
[jon@ghidora].[myschema.e_store_ts] event execution failed.
```

Since this user very likely does not have access to the error log, it is possible to verify whether the event's action statement is valid by executing it directly:

```
mysql> INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
ERROR 1142 (42000): INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
```

Inspection of the `INFORMATION_SCHEMA.EVENTS` table shows that `e_store_ts` exists and is enabled, but its `LAST_EXECUTED` column is `NULL`:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
     >     WHERE EVENT_NAME='e_store_ts'
     >     AND EVENT_SCHEMA='myschema'\G
*************************** 1. row ***************************
   EVENT_CATALOG: NULL
    EVENT_SCHEMA: myschema
      EVENT_NAME: e_store_ts
         DEFINER: jon@ghidora
      EVENT_BODY: SQL
EVENT_DEFINITION: INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP())
      EVENT_TYPE: RECURRING
      EXECUTE_AT: NULL
  INTERVAL_VALUE: 5
  INTERVAL_FIELD: SECOND
        SQL_MODE: NULL
          STARTS: 0000-00-00 00:00:00
            ENDS: 0000-00-00 00:00:00
          STATUS: ENABLED
   ON_COMPLETION: NOT PRESERVE
         CREATED: 2006-02-09 22:36:06
    LAST_ALTERED: 2006-02-09 22:36:06
   LAST_EXECUTED: NULL
   EVENT_COMMENT:
1 row in set (0.00 sec)
```

To rescind the `EVENT` privilege, use the `REVOKE` statement. In this example, the `EVENT` privilege on the schema `myschema` is removed from the `jon@ghidora` user account:

```
REVOKE EVENT ON myschema.* FROM jon@ghidora;
```

Important

Revoking the `EVENT` privilege from a user does not delete or disable any events that may have been created by that user.

An event is not migrated or dropped as a result of renaming or dropping the user who created it.

Suppose that the user `jon@ghidora` has been granted the `EVENT` and `INSERT` privileges on the `myschema` schema. This user then creates the following event:

```
CREATE EVENT e_insert
    ON SCHEDULE
      EVERY 7 SECOND
    DO
      INSERT INTO myschema.mytable;
```

After this event has been created, `root` revokes the `EVENT` privilege for `jon@ghidora`. However, `e_insert` continues to execute, inserting a new row into `mytable` each seven seconds. The same would be true if `root` had issued either of these statements:

- `DROP USER jon@ghidora;`
- `RENAME USER jon@ghidora TO someotherguy@ghidora;`

You can verify that this is true by examining the `INFORMATION_SCHEMA.EVENTS` table (see Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”) before and after issuing a `DROP USER` or `RENAME USER` statement.

Event definitions are stored in the data dictionary. To drop an event created by another user account, you must be the MySQL `root` user or another user with the necessary privileges.

Users' `EVENT` privileges are stored in the `Event_priv` columns of the `mysql.user` and `mysql.db` tables. In both cases, this column holds one of the values '`Y`' or '`N`'. '`N`' is the default. `mysql.user.Event_priv` is set to '`Y`' for a given user only if that user has the global `EVENT` privilege (that is, if the privilege was bestowed using `GRANT EVENT ON *.*`). For a schema-level `EVENT` privilege, `GRANT` creates a row in `mysql.db` and sets that row's `Db` column to the name of the schema, the `User` column to the name of the user, and the `Event_priv` column to '`Y`'. There should never be any need to manipulate these tables directly, since the `GRANT EVENT` and `REVOKE EVENT` statements perform the required operations on them.

Five status variables provide counts of event-related operations (but *not* of statements executed by events; see Section 24.8, “Restrictions on Stored Programs”). These are:

- `Com_create_event`: The number of `CREATE EVENT` statements executed since the last server restart.
- `Com_alter_event`: The number of `ALTER EVENT` statements executed since the last server restart.
- `Com_drop_event`: The number of `DROP EVENT` statements executed since the last server restart.
- `Com_show_create_event`: The number of `SHOW CREATE EVENT` statements executed since the last server restart.
- `Com_show_events`: The number of `SHOW EVENTS` statements executed since the last server restart.

You can view current values for all of these at one time by running the statement `SHOW STATUS LIKE '%event%';`.

24.5 Using Views

MySQL supports views, including updatable views. Views are stored queries that when invoked produce a result set. A view acts as a virtual table.

The following discussion describes the syntax for creating and dropping views, and shows some examples of how to use them.

Additional Resources

- You may find the Views User Forum of use when working with views.
- For answers to some commonly asked questions regarding views in MySQL, see Section A.6, “MySQL 8.0 FAQ: Views”.
- There are some restrictions on the use of views; see Section 24.9, “Restrictions on Views”.

24.5.1 View Syntax

The `CREATE VIEW` statement creates a new view (see Section 13.1.23, “CREATE VIEW Syntax”). To alter the definition of a view or drop a view, use `ALTER VIEW` (see Section 13.1.11, “ALTER VIEW Syntax”), or `DROP VIEW` (see Section 13.1.35, “DROP VIEW Syntax”).

A view can be created from many kinds of `SELECT` statements. It can refer to base tables or other views. It can use joins, `UNION`, and subqueries. The `SELECT` need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50), (5, 60);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
|    5 |    60 |   300 |
+------+-------+-------+
mysql> SELECT * FROM v WHERE qty = 5;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    5 |    60 |   300 |
+------+-------+-------+
```

24.5.2 View Processing Algorithms

The optional `ALGORITHM` clause for `CREATE VIEW` or `ALTER VIEW` is a MySQL extension to standard SQL. It affects how MySQL processes the view. `ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`.

- For `MERGE`, the text of a statement that refers to the view and the view definition are merged such that parts of the view definition replace corresponding parts of the statement.
- For `TEMPTABLE`, the results from the view are retrieved into a temporary table, which then is used to execute the statement.

- For UNDEFINED, MySQL chooses which algorithm to use. It prefers MERGE over TEMPTABLE if possible, because MERGE is usually more efficient and because a view cannot be updatable if a temporary table is used.
- If no ALGORITHM clause is present, the default algorithm is determined by the value of the derived_merge flag of the optimizer_switch system variable. For additional discussion, see Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”.

A reason to specify TEMPTABLE explicitly is that locks can be released on underlying tables after the temporary table has been created and before it is used to finish processing the statement. This might result in quicker lock release than the MERGE algorithm so that other clients that use the view are not blocked as long.

A view algorithm can be UNDEFINED for three reasons:

- No ALGORITHM clause is present in the CREATE VIEW statement.
- The CREATE VIEW statement has an explicit ALGORITHM = UNDEFINED clause.
- ALGORITHM = MERGE is specified for a view that can be processed only with a temporary table. In this case, MySQL generates a warning and sets the algorithm to UNDEFINED.

As mentioned earlier, MERGE is handled by merging corresponding parts of a view definition into the statement that refers to the view. The following examples briefly illustrate how the MERGE algorithm works. The examples assume that there is a view v_merge that has this definition:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 1: Suppose that we issue this statement:

```
SELECT * FROM v_merge;
```

MySQL handles the statement as follows:

- v_merge becomes t
- * becomes vc1, vc2, which corresponds to c1, c2
- The view WHERE clause is added

The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 2: Suppose that we issue this statement:

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

This statement is handled similarly to the previous one, except that vc1 < 100 becomes c1 < 100 and the view WHERE clause is added to the statement WHERE clause using an AND connective (and parentheses are added to make sure the parts of the clause are executed with correct precedence). The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

Effectively, the statement to be executed has a `WHERE` clause of this form:

```
WHERE (select WHERE) AND (view WHERE)
```

If the `MERGE` algorithm cannot be used, a temporary table must be used instead. Constructs that prevent merging are the same as those that prevent merging in derived tables and common table expressions. Examples are `SELECT DISTINCT` or `LIMIT` in the subquery. For details, see Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”.

24.5.3 Updatable and Insertable Views

Some views are updatable and references to them can be used to specify tables to be updated in data change statements. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table. Derived tables and common table expressions can also be specified in multiple-table `UPDATE` and `DELETE` statements, but can only be used for reading data to specify rows to be updated or deleted. Generally, the view references must be updatable, meaning that they may be merged and not materialized. Composite views have more complex rules.

For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable. To be more specific, a view is not updatable if it contains any of the following:

- Aggregate functions or window functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Subquery in the select list

  Nondependent subqueries in the select list fail for `INSERT`, but are okay for `UPDATE`, `DELETE`. For dependent subqueries in the select list, no data change statements are permitted.
- Certain joins (see additional join discussion later in this section)
- Reference to nonupdatable view in the `FROM` clause
- Subquery in the `WHERE` clause that refers to a table in the `FROM` clause
- Refers only to literal values (in this case, there is no underlying table to update)
- `ALGORITHM = TEMPTABLE` (use of a temporary table always makes a view nonupdatable)
- Multiple references to any column of a base table (fails for `INSERT`, okay for `UPDATE`, `DELETE`)

A generated column in a view is considered updatable because it is possible to assign to it. However, if such a column is updated explicitly, the only permitted value is `DEFAULT`. For information about generated columns, see Section 13.1.20.9, “CREATE TABLE and Generated Columns”.

It is sometimes possible for a multiple-table view to be updatable, assuming that it can be processed with the `MERGE` algorithm. For this to work, the view must use an inner join (not an outer join or a `UNION`). Also, only a single table in the view definition can be updated, so the `SET` clause must name only columns from one of the tables in the view. Views that use `UNION ALL` are not permitted even though they might be theoretically updatable.

With respect to insertability (being updatable with INSERT statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

- There must be no duplicate view column names.
- The view must contain all columns in the base table that do not have a default value.
- The view columns must be simple column references. They must not be expressions, such as these:

```
3.14159
col1 + 3
UPPER(col2)
col3 / col4
(subquery)
```

MySQL sets a flag, called the view updatability flag, at CREATE VIEW time. The flag is set to YES (true) if UPDATE and DELETE (and similar operations) are legal for the view. Otherwise, the flag is set to NO (false). The IS_UPDATABLE column in the INFORMATION_SCHEMA.VIEWS table displays the status of this flag. It means that the server always knows whether a view is updatable.

If a view is not updatable, statements such UPDATE, DELETE, and INSERT are illegal and are rejected. (Even if a view is updatable, it might not be possible to insert into it, as described elsewhere in this section.)

The updatability of views may be affected by the value of the updatable_views_with_limit system variable. See Section 5.1.8, “Server System Variables”.

For the following discussion, suppose that these tables and views exist:

```
CREATE TABLE t1 (x INTEGER);
CREATE TABLE t2 (c INTEGER);
CREATE VIEW vmat AS SELECT SUM(x) AS s FROM t1;
CREATE VIEW vup AS SELECT * FROM t2;
CREATE VIEW vjoin AS SELECT * FROM vmat JOIN vup ON vmat.s=vup.c;
```

INSERT, UPDATE, and DELETE statements are permitted as follows:

- INSERT: The insert table of an INSERT statement may be a view reference that is merged. If the view is a join view, all components of the view must be updatable (not materialized). For a multiple-table updatable view, INSERT can work if it inserts into a single table.

  This statement is invalid because one component of the join view is nonupdatable:

  ```
  INSERT INTO vjoin (c) VALUES (1);
  ```

  This statement is valid; the view contains no materialized components:

  ```
  INSERT INTO vup (c) VALUES (1);
  ```

- UPDATE: The table or tables to be updated in an UPDATE statement may be view references that are merged. If a view is a join view, at least one component of the view must be updatable (this differs from INSERT).

  In a multiple-table UPDATE statement, the updated table references of the statement must be base tables or updatable view references. Nonupdated table references may be materialized views or derived tables.

This statement is valid; column `c` is from the updatable part of the join view:

```
UPDATE vjoin SET c=c+1;
```

This statement is invalid; column `x` is from the nonupdatable part:

```
UPDATE vjoin SET x=x+1;
```

This statement is valid; the updated table reference of the multiple-table `UPDATE` is an updatable view (`vup`):

```
UPDATE vup JOIN (SELECT SUM(x) AS s FROM t1) AS dt ON ...
SET c=c+1;
```

This statement is invalid; it tries to update a materialized derived table:

```
UPDATE vup JOIN (SELECT SUM(x) AS s FROM t1) AS dt ON ...
SET s=s+1;
```

- `DELETE`: The table or tables to be deleted from in a `DELETE` statement must be merged views. Join views are not allowed (this differs from `INSERT` and `UPDATE`).

  This statement is invalid because the view is a join view:

  ```
  DELETE vjoin WHERE ...;
  ```

  This statement is valid because the view is a merged (updatable) view:

  ```
  DELETE vup WHERE ...;
  ```

  This statement is valid because it deletes from a merged (updatable) view:

  ```
  DELETE vup FROM vup JOIN (SELECT SUM(x) AS s FROM t1) AS dt ON ...;
  ```

Additional discussion and examples follow.

Earlier discussion in this section pointed out that a view is not insertable if not all columns are simple column references (for example, if it contains columns that are expressions or composite expressions). Although such a view is not insertable, it can be updatable if you update only columns that are not expressions. Consider this view:

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

This view is not insertable because `col2` is an expression. But it is updatable if the update does not try to update `col2`. This update is permissible:

```
UPDATE v SET col1 = 0;
```

This update is not permissible because it attempts to update an expression column:

```
UPDATE v SET col2 = 0;
```

If a table contains an AUTO_INCREMENT column, inserting into an insertable view on the table that does not include the AUTO_INCREMENT column does not change the value of LAST_INSERT_ID(), because the side effects of inserting default values into columns not part of the view should not be visible.

24.5.4 The View WITH CHECK OPTION Clause

The WITH CHECK OPTION clause can be given for an updatable view to prevent inserts to rows for which the WHERE clause in the *select_statement* is not true. It also prevents updates to rows for which the WHERE clause is true but the update would cause it to be not true (in other words, it prevents visible rows from being updated to nonvisible rows).

In a WITH CHECK OPTION clause for an updatable view, the LOCAL and CASCADED keywords determine the scope of check testing when the view is defined in terms of another view. When neither keyword is given, the default is CASCADED.

WITH CHECK OPTION testing is standard-compliant:

- With LOCAL, the view WHERE clause is checked, then checking recurses to underlying views and applies the same rules.
- With CASCADED, the view WHERE clause is checked, then checking recurses to underlying views, adds WITH CASCADED CHECK OPTION to them (for purposes of the check; their definitions remain unchanged), and applies the same rules.
- With no check option, the view WHERE clause is not checked, then checking recurses to underlying views, and applies the same rules.

Consider the definitions for the following table and set of views:

```
CREATE TABLE t1 (a INT);
CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
WITH CHECK OPTION;
CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
WITH LOCAL CHECK OPTION;
CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
WITH CASCADED CHECK OPTION;
```

Here the v2 and v3 views are defined in terms of another view, v1.

Inserts for v2 are checked against its LOCAL check option, then the check recurses to v1 and the rules are applied again. The rules for v1 cause a check failure. The check for v3 also fails:

```
mysql> INSERT INTO v2 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v2'
mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

24.5.5 View Metadata

Metadata about views can be obtained as follows:

- Query the VIEWS table of the INFORMATION_SCHEMA database. See Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”.
- Use the SHOW CREATE VIEW statement. See Section 13.7.7.13, “SHOW CREATE VIEW Syntax”.

24.6 Stored Object Access Control

Stored programs (procedures, functions, triggers, and events) and views are defined prior to use and, when referenced, execute within a security context that determines their privileges. These privileges are controlled by their `DEFINER` attribute and `SQL SECURITY` characteristic.

- The DEFINER Attribute
- The SQL SECURITY Characteristic
- Examples
- Risk-Minimization Guidelines

The DEFINER Attribute

All stored object definitions can include a `DEFINER` attribute that names a MySQL account. If a definition omits the `DEFINER` attribute, the default definer is the user who creates the object.

MySQL uses the following rules to control which accounts a user can specify in an object `DEFINER` attribute:

- If you have the `SET_USER_ID` or `SUPER` privilege, you can specify any account as the `DEFINER` value, although a warning is generated if the account does not exist. Additionally, as of MySQL 8.0.16, to set the `DEFINER` attribute for a stored object to an account that has the `SYSTEM_USER` privilege, you must have the `SYSTEM_USER` privilege.
- Otherwise, the only permitted account is your own, either specified literally or as `CURRENT_USER` or `CURRENT_USER()`. You cannot set the definer to some other account.

Creating a stored object with a nonexistent `DEFINER` account may have negative consequences:

- For a stored routine, an error occurs at routine execution time if the `SQL SECURITY` value is `DEFINER` but the definer account does not exist.
- For a trigger, it is not a good idea for trigger activation to occur until the account actually does exist. Otherwise, the behavior with respect to privilege checking is undefined.
- For an event, an error occurs at event execution time if the account does not exist.
- For a view, an error occurs when the view is referenced if the `SQL SECURITY` value is `DEFINER` but the definer account does not exist.

The SQL SECURITY Characteristic

Definitions for stored routines (procedures and functions) and views can include an `SQL SECURITY` characteristic with a value of `DEFINER` or `INVOKER` to specify whether the object executes in definer or invoker context. If a definition omits the `SQL SECURITY` characteristic, the default is definer context.

Triggers and events have no `SQL SECURITY` characteristic and always execute in definer context. The server invokes these objects automatically as necessary, so there is no invoking user.

Definer and invoker security contexts differ as follows:

- A stored object that executes in definer security context executes with the privileges of the account named by its `DEFINER` attribute. These privileges may be entirely different from those of the invoking user. The invoker must have appropriate privileges to reference the object (for example, `EXECUTE` to call a stored procedure or `SELECT` to select from a view), but during object execution, the invoker's privileges are ignored and only the `DEFINER` account privileges matter. If the `DEFINER` account has few privileges, the object is correspondingly limited in the operations it can perform. If the `DEFINER` account is highly

privileged (such as a `root` account), the object can perform powerful operations *no matter who invokes it.*

- A stored routine or view that executes in invoker security context can perform only operations for which the invoker has privileges. The `DEFINER` attribute has no effect during object execution.

Examples

Consider the following stored procedure, which is declared with `SQL SECURITY DEFINER` to execute in definer security context:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p1()
SQL SECURITY DEFINER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

Any user who has the `EXECUTE` privilege for `p1` can invoke it with a `CALL` statement. However, when `p1` executes, it does so in definer security context and thus executes with the privileges of `'admin'@'localhost'`, the account named in the `DEFINER` attribute. This account must have the `EXECUTE` privilege for `p1` as well as the `UPDATE` privilege for the table `t1` referenced within the object body. Otherwise, the procedure fails.

Now consider this stored procedure, which is identical to `p1` except that its `SQL SECURITY` characteristic is `INVOKER`:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p2()
SQL SECURITY INVOKER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

Unlike `p1`, `p2` executes in invoker security context and thus with the privileges of the invoking user regardless of the `DEFINER` attribute value. `p2` fails if the invoker lacks the `EXECUTE` privilege for `p2` or the `UPDATE` privilege for the table `t1`.

Risk-Minimization Guidelines

To minimize the risk potential for stored object creation and use, follow these guidelines:

- For a stored routine or view, use `SQL SECURITY INVOKER` in the object definition when possible so that it can be used only by users with permissions appropriate for the operations performed by the object.
- If you create definer-context stored objects while using an account that has the `SET_USER_ID` or `SUPER` privilege, specify an explicit `DEFINER` attribute that names an account possessing only the privileges required for the operations performed by the object. Specify a highly privileged `DEFINER` account only when absolutely necessary.
- Administrators can prevent users from creating stored objects that specify highly privileged `DEFINER` accounts by not granting them the `SET_USER_ID` or `SUPER` privilege.
- Definer-context objects should be written keeping in mind that they may be able to access data for which the invoking user has no privileges. In some cases, you can prevent references to these objects by not granting unauthorized users particular privileges:
  - A stored routine cannot be referenced by a user who does not have the `EXECUTE` privilege for it.

- A view cannot be referenced by a user who does not have the appropriate privilege for it (`SELECT` to select from it, `INSERT` to insert into it, and so forth).

However, no such control exists for triggers and events because they always execute in definer context. The server invokes these objects automatically as necessary; users do not reference them directly:

- A trigger is activated by access to the table with which it is associated, even ordinary table accesses by users with no special privileges.
- An event is executed by the server on a scheduled basis.

In both cases, if the `DEFINER` account is highly privileged, the object may be able to perform sensitive or dangerous operations. This remains true if the privileges needed to create the object are revoked from the account of the user who created it. Administrators should be especially careful about granting users object-creation privileges.

24.7 Stored Program Binary Logging

The binary log contains information about SQL statements that modify database contents. This information is stored in the form of “events” that describe the modifications. (Binary log events differ from scheduled event stored objects.) The binary log has two important purposes:

- For replication, the binary log is used on master replication servers as a record of the statements to be sent to slave servers. The master server sends the events contained in its binary log to its slaves, which execute those events to make the same data changes that were made on the master. See Section 17.2, “Replication Implementation”.
- Certain data recovery operations require use of the binary log. After a backup file has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See Section 7.3.2, “Using Backups for Recovery”.

However, if logging occurs at the statement level, there are certain binary logging issues with respect to stored programs (stored procedures and functions, triggers, and events):

- In some cases, a statement might affect different sets of rows on master and slave.
- Replicated statements executed on a slave are processed by the slave SQL thread, which has full privileges. It is possible for a procedure to follow different execution paths on master and slave servers, so a user can write a routine containing a dangerous statement that will execute only on the slave where it is processed by a thread that has full privileges.
- If a stored program that modifies data is nondeterministic, it is not repeatable. This can result in different data on master and slave, or cause restored data to differ from the original data.

This section describes how MySQL handles binary logging for stored programs. It states the current conditions that the implementation places on the use of stored programs, and what you can do to avoid logging problems. It also provides additional information about the reasons for these conditions.

In general, the issues described here result when binary logging occurs at the SQL statement level (statement-based binary logging). If you use row-based binary logging, the log contains changes made to individual rows as a result of executing SQL statements. When routines or triggers execute, row changes are logged, not the statements that make the changes. For stored procedures, this means that the `CALL` statement is not logged. For stored functions, row changes made within the function are logged, not the function invocation. For triggers, row changes made by the trigger are logged. On the slave side, only the row changes are seen, not the stored program invocation.

Mixed format binary logging (`binlog_format=MIXED`) uses statement-based binary logging, except for cases where only row-based binary logging is guaranteed to lead to proper results. With mixed format, when a stored function, stored procedure, trigger, event, or prepared statement contains anything that is not safe for statement-based binary logging, the entire statement is marked as unsafe and logged in row format. The statements used to create and drop procedures, functions, triggers, and events are always safe, and are logged in statement format. For more information about row-based, mixed, and statement-based logging, and how safe and unsafe statements are determined, see Section 17.2.1, “Replication Formats”.

Unless noted otherwise, the remarks here assume that binary logging is enabled on the server (see Section 5.4.4, “The Binary Log”.) If the binary log is not enabled, replication is not possible, nor is the binary log available for data recovery.

The conditions on the use of stored functions in MySQL can be summarized as follows. These conditions do not apply to stored procedures or Event Scheduler events and they do not apply unless binary logging is enabled.

- To create or alter a stored function, you must have the `SET_USER_ID` or `SUPER` privilege, in addition to the `CREATE ROUTINE` or `ALTER ROUTINE` privilege that is normally required. (Depending on the `DEFINER` value in the function definition, `SET_USER_ID` or `SUPER` might be required regardless of whether binary logging is enabled. See Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”.)
- When you create a stored function, you must declare either that it is deterministic or that it does not modify data. Otherwise, it may be unsafe for data recovery or replication.

  By default, for a `CREATE FUNCTION` statement to be accepted, at least one of `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

  ```
  ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
  or READS SQL DATA in its declaration and binary logging is enabled
  (you *might* want to use the less safe log_bin_trust_function_creators
  variable)
  ```

  This function is deterministic (and does not modify data), so it is safe:

  ```
  CREATE FUNCTION f1(i INT)
  RETURNS INT
  DETERMINISTIC
  READS SQL DATA
  BEGIN
    RETURN i;
  END;
  ```

  This function uses `UUID()`, which is not deterministic, so the function also is not deterministic and is not safe:

  ```
  CREATE FUNCTION f2()
  RETURNS CHAR(36) CHARACTER SET utf8
  BEGIN
    RETURN UUID();
  END;
  ```

  This function modifies data, so it may not be safe:

  ```
  CREATE FUNCTION f3(p_id INT)
  RETURNS INT
  ```

```
BEGIN
  UPDATE t SET modtime = NOW() WHERE id = p_id;
  RETURN ROW_COUNT();
END;
```

Assessment of the nature of a function is based on the “honesty” of the creator. MySQL does not check that a function declared `DETERMINISTIC` is free of statements that produce nondeterministic results.

- When you attempt to execute a stored function, if `binlog_format=STATEMENT` is set, the `DETERMINISTIC` keyword must be specified in the function definition. If this is not the case, an error is generated and the function does not run, unless `log_bin_trust_function_creators=1` is specified to override this check (see below). For recursive function calls, the `DETERMINISTIC` keyword is required on the outermost call only. If row-based or mixed binary logging is in use, the statement is accepted and replicated even if the function was defined without the `DETERMINISTIC` keyword.

- Because MySQL does not check if a function really is deterministic at creation time, the invocation of a stored function with the `DETERMINISTIC` keyword might carry out an action that is unsafe for statement-based logging, or invoke a function or procedure containing unsafe statements. If this occurs when `binlog_format=STATEMENT` is set, a warning message is issued. If row-based or mixed binary logging is in use, no warning is issued, and the statement is replicated in row-based format.

- To relax the preceding conditions on function creation (that you must have the `SUPER` privilege and that a function must be declared deterministic or to not modify data), set the global `log_bin_trust_function_creators` system variable to 1. By default, this variable has a value of 0, but you can change it like this:

  ```
  mysql> SET GLOBAL log_bin_trust_function_creators = 1;
  ```

  You can also set this variable at server startup.

  If binary logging is not enabled, `log_bin_trust_function_creators` does not apply. `SUPER` is not required for function creation unless, as described previously, the `DEFINER` value in the function definition requires it.

- For information about built-in functions that may be unsafe for replication (and thus cause stored functions that use them to be unsafe as well), see Section 17.5.1, “Replication Features and Issues”.

Triggers are similar to stored functions, so the preceding remarks regarding functions also apply to triggers with the following exception: `CREATE TRIGGER` does not have an optional `DETERMINISTIC` characteristic, so triggers are assumed to be always deterministic. However, this assumption might be invalid in some cases. For example, the `UUID()` function is nondeterministic (and does not replicate). Be careful about using such functions in triggers.

Triggers can update tables, so error messages similar to those for stored functions occur with `CREATE TRIGGER` if you do not have the required privileges. On the slave side, the slave uses the trigger `DEFINER` attribute to determine which user is considered to be the creator of the trigger.

The rest of this section provides additional detail about the logging implementation and its implications. You need not read it unless you are interested in the background on the rationale for the current logging-related conditions on stored routine use. This discussion applies only for statement-based logging, and not for row-based logging, with the exception of the first item: `CREATE` and `DROP` statements are logged as statements regardless of the logging mode.

- The server writes `CREATE EVENT`, `CREATE PROCEDURE`, `CREATE FUNCTION`, `ALTER EVENT`, `ALTER PROCEDURE`, `ALTER FUNCTION`, `DROP EVENT`, `DROP PROCEDURE`, and `DROP FUNCTION` statements to the binary log.

- A stored function invocation is logged as a `SELECT` statement if the function changes data and occurs within a statement that would not otherwise be logged. This prevents nonreplication of data changes that result from use of stored functions in nonlogged statements. For example, `SELECT` statements are not written to the binary log, but a `SELECT` might invoke a stored function that makes changes. To handle this, a `SELECT func_name()` statement is written to the binary log when the given function makes a change. Suppose that the following statements are executed on the master:

```
CREATE FUNCTION f1(a INT) RETURNS INT
BEGIN
  IF (a < 3) THEN
    INSERT INTO t2 VALUES (a);
  END IF;
  RETURN 0;
END;

CREATE TABLE t1 (a INT);
INSERT INTO t1 VALUES (1),(2),(3);

SELECT f1(a) FROM t1;
```

When the `SELECT` statement executes, the function `f1()` is invoked three times. Two of those invocations insert a row, and MySQL logs a `SELECT` statement for each of them. That is, MySQL writes the following statements to the binary log:

```
SELECT f1(1);
SELECT f1(2);
```

The server also logs a `SELECT` statement for a stored function invocation when the function invokes a stored procedure that causes an error. In this case, the server writes the `SELECT` statement to the log along with the expected error code. On the slave, if the same error occurs, that is the expected result and replication continues. Otherwise, replication stops.

- Logging stored function invocations rather than the statements executed by a function has a security implication for replication, which arises from two factors:
  - It is possible for a function to follow different execution paths on master and slave servers.
  - Statements executed on a slave are processed by the slave SQL thread which has full privileges.

  The implication is that although a user must have the `CREATE ROUTINE` privilege to create a function, the user can write a function containing a dangerous statement that will execute only on the slave where it is processed by a thread that has full privileges. For example, if the master and slave servers have server ID values of 1 and 2, respectively, a user on the master server could create and invoke an unsafe function `unsafe_func()` as follows:

```
mysql> delimiter //
mysql> CREATE FUNCTION unsafe_func () RETURNS INT
    -> BEGIN
    ->   IF @@server_id=2 THEN dangerous_statement; END IF;
    ->   RETURN 1;
    -> END;
    -> //
mysql> delimiter ;
mysql> INSERT INTO t VALUES(unsafe_func());
```

The `CREATE FUNCTION` and `INSERT` statements are written to the binary log, so the slave will execute them. Because the slave SQL thread has full privileges, it will execute the dangerous statement. Thus, the function invocation has different effects on the master and slave and is not replication-safe.

To guard against this danger for servers that have binary logging enabled, stored function creators must have the SUPER privilege, in addition to the usual CREATE ROUTINE privilege that is required. Similarly, to use ALTER FUNCTION, you must have the SUPER privilege in addition to the ALTER ROUTINE privilege. Without the SUPER privilege, an error will occur:

```
ERROR 1419 (HY000): You do not have the SUPER privilege and
binary logging is enabled (you *might* want to use the less safe
log_bin_trust_function_creators variable)
```

If you do not want to require function creators to have the SUPER privilege (for example, if all users with the CREATE ROUTINE privilege on your system are experienced application developers), set the global log_bin_trust_function_creators system variable to 1. You can also set this variable at server startup. If binary logging is not enabled, log_bin_trust_function_creators does not apply. SUPER is not required for function creation unless, as described previously, the DEFINER value in the function definition requires it.

- If a function that performs updates is nondeterministic, it is not repeatable. This can have two undesirable effects:
  - It will make a slave different from the master.
  - Restored data will be different from the original data.

  To deal with these problems, MySQL enforces the following requirement: On a master server, creation and alteration of a function is refused unless you declare the function to be deterministic or to not modify data. Two sets of function characteristics apply here:

  - The DETERMINISTIC and NOT DETERMINISTIC characteristics indicate whether a function always produces the same result for given inputs. The default is NOT DETERMINISTIC if neither characteristic is given. To declare that a function is deterministic, you must specify DETERMINISTIC explicitly.
  - The CONTAINS SQL, NO SQL, READS SQL DATA, and MODIFIES SQL DATA characteristics provide information about whether the function reads or writes data. Either NO SQL or READS SQL DATA indicates that a function does not change data, but you must specify one of these explicitly because the default is CONTAINS SQL if no characteristic is given.

  By default, for a CREATE FUNCTION statement to be accepted, at least one of DETERMINISTIC, NO SQL, or READS SQL DATA must be specified explicitly. Otherwise an error occurs:

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_function_creators
variable)
```

  If you set log_bin_trust_function_creators to 1, the requirement that functions be deterministic or not modify data is dropped.

- Stored procedure calls are logged at the statement level rather than at the CALL level. That is, the server does not log the CALL statement, it logs those statements within the procedure that actually execute. As a result, the same changes that occur on the master will be observed on slave servers. This prevents problems that could result from a procedure having different execution paths on different machines.

  In general, statements executed within a stored procedure are written to the binary log using the same rules that would apply were the statements to be executed in standalone fashion. Some special care is

taken when logging procedure statements because statement execution within procedures is not quite the same as in nonprocedure context:

- A statement to be logged might contain references to local procedure variables. These variables do not exist outside of stored procedure context, so a statement that refers to such a variable cannot be logged literally. Instead, each reference to a local variable is replaced by this construct for logging purposes:

  ```
  NAME_CONST(var_name, var_value)
  ```

  `var_name` is the local variable name, and *`var_value`* is a constant indicating the value that the variable has at the time the statement is logged. `NAME_CONST()` has a value of *`var_value`*, and a "name" of *`var_name`*. Thus, if you invoke this function directly, you get a result like this:

  ```
  mysql> SELECT NAME_CONST('myname', 14);
  +--------+
  | myname |
  +--------+
  |     14 |
  +--------+
  ```

  `NAME_CONST()` enables a logged standalone statement to be executed on a slave with the same effect as the original statement that was executed on the master within a stored procedure.

  The use of `NAME_CONST()` can result in a problem for `CREATE TABLE ... SELECT` statements when the source column expressions refer to local variables. Converting these references to `NAME_CONST()` expressions can result in column names that are different on the master and slave servers, or names that are too long to be legal column identifiers. A workaround is to supply aliases for columns that refer to local variables. Consider this statement when `myvar` has a value of 1:

  ```
  CREATE TABLE t1 SELECT myvar;
  ```

  That will be rewritten as follows:

  ```
  CREATE TABLE t1 SELECT NAME_CONST(myvar, 1);
  ```

  To ensure that the master and slave tables have the same column names, write the statement like this:

  ```
  CREATE TABLE t1 SELECT myvar AS myvar;
  ```

  The rewritten statement becomes:

  ```
  CREATE TABLE t1 SELECT NAME_CONST(myvar, 1) AS myvar;
  ```

- A statement to be logged might contain references to user-defined variables. To handle this, MySQL writes a `SET` statement to the binary log to make sure that the variable exists on the slave with the same value as on the master. For example, if a statement refers to a variable `@my_var`, that statement will be preceded in the binary log by the following statement, where *`value`* is the value of `@my_var` on the master:

  ```
  SET @my_var = value;
  ```

  - Procedure calls can occur within a committed or rolled-back transaction. Transactional context is accounted for so that the transactional aspects of procedure execution are replicated correctly. That is, the server logs those statements within the procedure that actually execute and modify data, and also logs `BEGIN`, `COMMIT`, and `ROLLBACK` statements as necessary. For example, if a procedure updates only transactional tables and is executed within a transaction that is rolled back, those updates are not logged. If the procedure occurs within a committed transaction, `BEGIN` and `COMMIT` statements are logged with the updates. For a procedure that executes within a rolled-back transaction, its statements are logged using the same rules that would apply if the statements were executed in standalone fashion:

    - Updates to transactional tables are not logged.
    - Updates to nontransactional tables are logged because rollback does not cancel them.
    - Updates to a mix of transactional and nontransactional tables are logged surrounded by `BEGIN` and `ROLLBACK` so that slaves will make the same changes and rollbacks as on the master.

- A stored procedure call is *not* written to the binary log at the statement level if the procedure is invoked from within a stored function. In that case, the only thing logged is the statement that invokes the function (if it occurs within a statement that is logged) or a `DO` statement (if it occurs within a statement that is not logged). For this reason, care should be exercised in the use of stored functions that invoke a procedure, even if the procedure is otherwise safe in itself.

24.8 Restrictions on Stored Programs

These restrictions apply to the features described in Chapter 24, *Stored Objects*.

Some of the restrictions noted here apply to all stored routines; that is, both to stored procedures and stored functions. There are also some restrictions specific to stored functions but not to stored procedures.

The restrictions for stored functions also apply to triggers. There are also some restrictions specific to triggers.

The restrictions for stored procedures also apply to the `DO` clause of Event Scheduler event definitions. There are also some restrictions specific to events.

SQL Statements Not Permitted in Stored Routines

Stored routines cannot contain arbitrary SQL statements. The following statements are not permitted:

- The locking statements `LOCK TABLES` and `UNLOCK TABLES`.
- `ALTER VIEW`.
- `LOAD DATA` and `LOAD TABLE`.
- SQL prepared statements (`PREPARE`, `EXECUTE`, `DEALLOCATE PREPARE`) can be used in stored procedures, but not stored functions or triggers. Thus, stored functions and triggers cannot use dynamic SQL (where you construct statements as strings and then execute them).
- Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. For a list of statements supported as prepared statements, see Section 13.5, “Prepared SQL Statement Syntax”. Exceptions are `SIGNAL`, `RESIGNAL`, and `GET DIAGNOSTICS`, which are not permissible as prepared statements but are permitted in stored programs.

- Because local variables are in scope only during stored program execution, references to them are not permitted in prepared statements created within a stored program. Prepared statement scope is the current session, not the stored program, so the statement could be executed after the program ends, at which point the variables would no longer be in scope. For example, `SELECT ... INTO local_var` cannot be used as a prepared statement. This restriction also applies to stored procedure and function parameters. See Section 13.5.1, “PREPARE Syntax”.

- Within all stored programs (stored procedures and functions, triggers, and events), the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. To begin a transaction in this context, use `START TRANSACTION` instead.

Restrictions for Stored Functions

The following additional statements or operations are not permitted within stored functions. They are permitted within stored procedures, except stored procedures that are invoked from within a stored function or trigger. For example, if you use `FLUSH` in a stored procedure, that stored procedure cannot be called from a stored function or trigger.

- Statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.

- Statements that return a result set. This includes `SELECT` statements that do not have an `INTO var_list` clause and other statements such as `SHOW`, `EXPLAIN`, and `CHECK TABLE`. A function can process a result set either with `SELECT ... INTO var_list` or by using a cursor and `FETCH` statements. See Section 13.2.10.1, “SELECT ... INTO Syntax”, and Section 13.6.6, “Cursors”.

- `FLUSH` statements.

- Stored functions cannot be used recursively.

- A stored function or trigger cannot modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.

- If you refer to a temporary table multiple times in a stored function under different aliases, a `Can't reopen table: 'tbl_name'` error occurs, even if the references occur in different statements within the function.

- `HANDLER ... READ` statements that invoke stored functions can cause replication errors and are disallowed.

Restrictions for Triggers

For triggers, the following additional restrictions apply:

- Triggers are not activated by foreign key actions.

- When using row-based replication, triggers on the slave are not activated by statements originating on the master. The triggers on the slave are activated when using statement-based replication. For more information, see Section 17.5.1.35, “Replication and Triggers”.

- The `RETURN` statement is not permitted in triggers, which cannot return a value. To exit a trigger immediately, use the `LEAVE` statement.

- Triggers are not permitted on tables in the `mysql` database. Nor are they permitted on `INFORMATION_SCHEMA` or `performance_schema` tables. Those tables are actually views and triggers are not permitted on views.

- The trigger cache does not detect when metadata of the underlying objects has changed. If a trigger uses a table and the table has changed since the trigger was loaded into the cache, the trigger operates using the outdated metadata.

Name Conflicts within Stored Routines

The same identifier might be used for a routine parameter, a local variable, and a table column. Also, the same local variable name can be used in nested blocks. For example:

```
CREATE PROCEDURE p (i INT)
BEGIN
  DECLARE i INT DEFAULT 0;
  SELECT i FROM t;
  BEGIN
    DECLARE i INT DEFAULT 1;
    SELECT i FROM t;
  END;
END;
```

In such cases, the identifier is ambiguous and the following precedence rules apply:

- A local variable takes precedence over a routine parameter or table column.
- A routine parameter takes precedence over a table column.
- A local variable in an inner block takes precedence over a local variable in an outer block.

The behavior that variables take precedence over table columns is nonstandard.

Replication Considerations

Use of stored routines can cause replication problems. This issue is discussed further in Section 24.7, “Stored Program Binary Logging”.

The `--replicate-wild-do-table=db_name.tbl_name` option applies to tables, views, and triggers. It does not apply to stored procedures and functions, or events. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

Debugging Considerations

There are no stored routine debugging facilities.

Unsupported Syntax from the SQL:2003 Standard

The MySQL stored routine syntax is based on the SQL:2003 standard. The following items from that standard are not currently supported:

- `UNDO` handlers
- `FOR` loops

Stored Routine Concurrency Considerations

To prevent problems of interaction between sessions, when a client issues a statement, the server uses a snapshot of routines and triggers available for execution of the statement. That is, the server calculates a list of procedures, functions, and triggers that may be used during execution of the statement, loads them,

and then proceeds to execute the statement. While the statement executes, it does not see changes to routines performed by other sessions.

For maximum concurrency, stored functions should minimize their side-effects; in particular, updating a table within a stored function can reduce concurrent operations on that table. A stored function acquires table locks before executing, to avoid inconsistency in the binary log due to mismatch of the order in which statements execute and when they appear in the log. When statement-based binary logging is used, statements that invoke a function are recorded rather than the statements executed within the function. Consequently, stored functions that update the same underlying tables do not execute in parallel. In contrast, stored procedures do not acquire table-level locks. All statements executed within stored procedures are written to the binary log, even for statement-based binary logging. See Section 24.7, “Stored Program Binary Logging”.

Event Scheduler Restrictions

The following limitations are specific to the Event Scheduler:

- Event names are handled in case-insensitive fashion. For example, you cannot have two events in the same database with the names `anEvent` and `AnEvent`.
- An event may not be created, altered, or dropped from within a stored program, if the event name is specified by means of a variable. An event also may not create, alter, or drop stored routines or triggers.
- DDL statements on events are prohibited while a `LOCK TABLES` statement is in effect.
- Event timings using the intervals `YEAR`, `QUARTER`, `MONTH`, and `YEAR_MONTH` are resolved in months; those using any other interval are resolved in seconds. There is no way to cause events scheduled to occur at the same second to execute in a given order. In addition—due to rounding, the nature of threaded applications, and the fact that a nonzero length of time is required to create events and to signal their execution—events may be delayed by as much as 1 or 2 seconds. However, the time shown in the `INFORMATION_SCHEMA.EVENTS` table's `LAST_EXECUTED` column is always accurate to within one second of the actual event execution time. (See also Bug #16522.)
- Each execution of the statements contained in the body of an event takes place in a new connection; thus, these statements has no effect in a given user session on the server's statement counts such as `Com_select` and `Com_insert` that are displayed by `SHOW STATUS`. However, such counts *are* updated in the global scope. (Bug #16422)
- Events do not support times later than the end of the Unix Epoch; this is approximately the beginning of the year 2038. Such dates are specifically not permitted by the Event Scheduler. (Bug #16396)
- References to stored functions, user-defined functions, and tables in the `ON SCHEDULE` clauses of `CREATE EVENT` and `ALTER EVENT` statements are not supported. These sorts of references are not permitted. (See Bug #22830 for more information.)

Stored routines and triggers in NDB Cluster. Stored procedures, stored functions, and triggers are all supported by tables using the `NDB` storage engine; however, it is important to keep in mind that they do *not* propagate automatically between MySQL Servers acting as Cluster SQL nodes. This is because stored routine and trigger definitions are stored in tables in the `mysql` system database using `InnoDB` tables, which are not copied between Cluster nodes.

Any stored routine or trigger that interacts with MySQL Cluster tables must be re-created by running the appropriate `CREATE PROCEDURE`, `CREATE FUNCTION`, or `CREATE TRIGGER` statements on each MySQL Server that participates in the cluster where you wish to use the stored routine or trigger. Similarly, any changes to existing stored routines or triggers must be carried out explicitly on all Cluster SQL nodes, using the appropriate `ALTER` or `DROP` statements on each MySQL Server accessing the cluster.

Warning

Do *not* attempt to work around the issue just described by converting any `mysql` database tables to use the `NDB` storage engine. *Altering the system tables in the `mysql` database is not supported* and is very likely to produce undesirable results.

24.9 Restrictions on Views

The maximum number of tables that can be referenced in the definition of a view is 61.

View processing is not optimized:

- It is not possible to create an index on a view.
- Indexes can be used for views processed using the merge algorithm. However, a view that is processed with the temptable algorithm is unable to take advantage of indexes on its underlying tables (although indexes can be used during generation of the temporary tables).

There is a general principle that you cannot modify a table and select from the same table in a subquery. See Section 13.2.11.13, “Restrictions on Subqueries”.

The same principle also applies if you select from a view that selects from the table, if the view selects from the table in a subquery and the view is evaluated using the merge algorithm. Example:

```
CREATE VIEW v1 AS
SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);

UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

If the view is evaluated using a temporary table, you *can* select from the table in the view subquery and still modify that table in the outer query. In this case the view will be stored in a temporary table and thus you are not really selecting from the table in a subquery and modifying it “at the same time.” (This is another reason you might wish to force MySQL to use the temptable algorithm by specifying `ALGORITHM = TEMPTABLE` in the view definition.)

You can use `DROP TABLE` or `ALTER TABLE` to drop or alter a table that is used in a view definition. No warning results from the `DROP` or `ALTER` operation, even though this invalidates the view. Instead, an error occurs later, when the view is used. `CHECK TABLE` can be used to check for views that have been invalidated by `DROP` or `ALTER` operations.

With regard to view updatability, the overall goal for views is that if any view is theoretically updatable, it should be updatable in practice. MySQL as quickly as possible. Many theoretically updatable views can be updated now, but limitations still exist. For details, see Section 24.5.3, “Updatable and Insertable Views”.

There exists a shortcoming with the current implementation of views. If a user is granted the basic privileges necessary to create a view (the `CREATE VIEW` and `SELECT` privileges), that user will be unable to call `SHOW CREATE VIEW` on that object unless the user is also granted the `SHOW VIEW` privilege.

That shortcoming can lead to problems backing up a database with `mysqldump`, which may fail due to insufficient privileges. This problem is described in Bug #22062.

The workaround to the problem is for the administrator to manually grant the `SHOW VIEW` privilege to users who are granted `CREATE VIEW`, since MySQL doesn't grant it implicitly when views are created.

Views do not have indexes, so index hints do not apply. Use of index hints when selecting from a view is not permitted.

`SHOW CREATE VIEW` displays view definitions using an `AS alias_name` clause for each column. If a column is created from an expression, the default alias is the expression text, which can be quite long. Aliases for column names in `CREATE VIEW` statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters). As a result, views created from the output of `SHOW CREATE VIEW` fail if any column alias exceeds 64 characters. This can cause problems in the following circumstances for views with too-long aliases:

- View definitions fail to replicate to newer slaves that enforce the column-length restriction.
- Dump files created with `mysqldump` cannot be loaded into servers that enforce the column-length restriction.

A workaround for either problem is to modify each problematic view definition to use aliases that provide shorter column names. Then the view will replicate properly, and can be dumped and reloaded without causing an error. To modify the definition, drop and create the view again with `DROP VIEW` and `CREATE VIEW`, or replace the definition with `CREATE OR REPLACE VIEW`.

For problems that occur when reloading view definitions in dump files, another workaround is to edit the dump file to modify its `CREATE VIEW` statements. However, this does not change the original view definitions, which may cause problems for subsequent dump operations.

Chapter 25 INFORMATION_SCHEMA Tables

Table of Contents

25.1 Introduction 4370
25.2 The INFORMATION_SCHEMA CHARACTER_SETS Table 4373
25.3 The INFORMATION_SCHEMA CHECK_CONSTRAINTS Table 4374
25.4 The INFORMATION_SCHEMA COLLATIONS Table 4374
25.5 The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table 4375
25.6 The INFORMATION_SCHEMA COLUMNS Table 4375
25.7 The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table 4378
25.8 The INFORMATION_SCHEMA COLUMN_STATISTICS Table 4379
25.9 The INFORMATION_SCHEMA ENGINES Table 4379
25.10 The INFORMATION_SCHEMA EVENTS Table 4381
25.11 The INFORMATION_SCHEMA FILES Table 4384
25.12 The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table 4392
25.13 The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table 4394
25.14 The INFORMATION_SCHEMA KEYWORDS Table 4395
25.15 The INFORMATION_SCHEMA OPTIMIZER_TRACE Table 4396
25.16 The INFORMATION_SCHEMA PARAMETERS Table 4396
25.17 The INFORMATION_SCHEMA PARTITIONS Table 4398
25.18 The INFORMATION_SCHEMA PLUGINS Table 4401
25.19 The INFORMATION_SCHEMA PROCESSLIST Table 4402
25.20 The INFORMATION_SCHEMA PROFILING Table 4404
25.21 The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table 4405
25.22 The INFORMATION_SCHEMA RESOURCE_GROUPS Table 4406
25.23 The INFORMATION_SCHEMA ROUTINES Table 4406
25.24 The INFORMATION_SCHEMA SCHEMATA Table 4409
25.25 The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table 4410
25.26 The INFORMATION_SCHEMA STATISTICS Table 4411
25.27 The INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS Table 4413
25.28 The INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS Table 4414
25.29 The INFORMATION_SCHEMA ST_UNITS_OF_MEASURE Table 4415
25.30 The INFORMATION_SCHEMA TABLES Table 4416
25.31 The INFORMATION_SCHEMA TABLESPACES Table 4420
25.32 The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table 4421
25.33 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table 4421
25.34 The INFORMATION_SCHEMA TRIGGERS Table 4422
25.35 The INFORMATION_SCHEMA USER_PRIVILEGES Table 4424
25.36 The INFORMATION_SCHEMA VIEWS Table 4425
25.37 The INFORMATION_SCHEMA VIEW_ROUTINE_USAGE Table 4426
25.38 The INFORMATION_SCHEMA VIEW_TABLE_USAGE Table 4427
25.39 INFORMATION_SCHEMA InnoDB Tables 4428
- 25.39.1 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table 4428
- 25.39.2 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table 4431
- 25.39.3 The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table 4435
- 25.39.4 The INFORMATION_SCHEMA INNODB_CACHED_INDEXES Table 4438
- 25.39.5 The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables 4439
- 25.39.6 The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables 4441
- 25.39.7 The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables 4442

25.39.8 The INFORMATION_SCHEMA INNODB_COLUMNS Table 4444
25.39.9 The INFORMATION_SCHEMA INNODB_DATAFILES Table 4445
25.39.10 The INFORMATION_SCHEMA INNODB_FIELDS Table 4446
25.39.11 The INFORMATION_SCHEMA INNODB_FOREIGN Table 4447
25.39.12 The INFORMATION_SCHEMA INNODB_FOREIGN_COLS Table 4447
25.39.13 The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table 4448
25.39.14 The INFORMATION_SCHEMA INNODB_FT_CONFIG Table 4449
25.39.15 The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table 4450
25.39.16 The INFORMATION_SCHEMA INNODB_FT_DELETED Table 4451
25.39.17 The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table 4452
25.39.18 The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table 4453
25.39.19 The INFORMATION_SCHEMA INNODB_INDEXES Table 4455
25.39.20 The INFORMATION_SCHEMA INNODB_LOCKS Table 4456
25.39.21 The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table 4457
25.39.22 The INFORMATION_SCHEMA INNODB_METRICS Table 4458
25.39.23 The INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES Table ... 4460
25.39.24 The INFORMATION_SCHEMA INNODB_TABLES Table 4461
25.39.25 The INFORMATION_SCHEMA INNODB_TABLESPACES Table 4462
25.39.26 The INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF Table 4464
25.39.27 The INFORMATION_SCHEMA INNODB_TABLESTATS View 4465
25.39.28 The INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO Table 4467
25.39.29 The INFORMATION_SCHEMA INNODB_TRX Table 4468
25.39.30 The INFORMATION_SCHEMA INNODB_VIRTUAL Table 4471
25.40 INFORMATION_SCHEMA Thread Pool Tables 4472
25.40.1 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table 4473
25.40.2 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table 4473
25.40.3 The INFORMATION_SCHEMA TP_THREAD_STATE Table 4473
25.41 INFORMATION_SCHEMA Connection-Control Tables 4474
25.41.1 The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table 4474
25.42 Extensions to SHOW Statements 4475

`INFORMATION_SCHEMA` provides access to database *metadata*, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges. Other terms that are sometimes used for this information are *data dictionary* and *system catalog*.

25.1 Introduction

`INFORMATION_SCHEMA` provides access to database *metadata*, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges. Other terms that are sometimes used for this information are *data dictionary* and *system catalog*.

- INFORMATION_SCHEMA Usage Notes
- Character Set Considerations
- INFORMATION_SCHEMA as Alternative to SHOW Statements
- INFORMATION_SCHEMA and Privileges
- Performance Considerations
- Standards Considerations
- Conventions in the INFORMATION_SCHEMA Reference Sections

- Related Information

INFORMATION_SCHEMA Usage Notes

`INFORMATION_SCHEMA` is a database within each MySQL instance, the place that stores information about all the other databases that the MySQL server maintains. The `INFORMATION_SCHEMA` database contains several read-only tables. They are actually views, not base tables, so there are no files associated with them, and you cannot set triggers on them. Also, there is no database directory with that name.

Although you can select `INFORMATION_SCHEMA` as the default database with a `USE` statement, you can only read the contents of tables, not perform `INSERT`, `UPDATE`, or `DELETE` operations on them.

Here is an example of a statement that retrieves information from `INFORMATION_SCHEMA`:

```
mysql> SELECT table_name, table_type, engine
       FROM information_schema.tables
       WHERE table_schema = 'db5'
       ORDER BY table_name;
+------------+------------+--------+
| table_name | table_type | engine |
+------------+------------+--------+
| fk         | BASE TABLE | InnoDB |
| fk2        | BASE TABLE | InnoDB |
| goto       | BASE TABLE | MyISAM |
| into       | BASE TABLE | MyISAM |
| k          | BASE TABLE | MyISAM |
| kurs       | BASE TABLE | MyISAM |
| loop       | BASE TABLE | MyISAM |
| pk         | BASE TABLE | InnoDB |
| t          | BASE TABLE | MyISAM |
| t2         | BASE TABLE | MyISAM |
| t3         | BASE TABLE | MyISAM |
| t7         | BASE TABLE | MyISAM |
| tables     | BASE TABLE | MyISAM |
| v          | VIEW       | NULL   |
| v2         | VIEW       | NULL   |
| v3         | VIEW       | NULL   |
| v56        | VIEW       | NULL   |
+------------+------------+--------+
17 rows in set (0.01 sec)
```

Explanation: The statement requests a list of all the tables in database `db5`, showing just three pieces of information: the name of the table, its type, and its storage engine.

Character Set Considerations

The definition for character columns (for example, `TABLES.TABLE_NAME`) is generally `VARCHAR(N) CHARACTER SET utf8` where *`N`* is at least 64. MySQL uses the default collation for this character set (`utf8_general_ci`) for all searches, sorts, comparisons, and other string operations on such columns.

Because some MySQL objects are represented as files, searches in `INFORMATION_SCHEMA` string columns can be affected by file system case sensitivity. For more information, see Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”.

INFORMATION_SCHEMA as Alternative to SHOW Statements

The `SELECT ... FROM INFORMATION_SCHEMA` statement is intended as a more consistent way to provide access to the information provided by the various `SHOW` statements that MySQL supports (`SHOW DATABASES`, `SHOW TABLES`, and so forth). Using `SELECT` has these advantages, compared to `SHOW`:

- It conforms to Codd's rules, because all access is done on tables.
- You can use the familiar syntax of the `SELECT` statement, and only need to learn some table and column names.
- The implementor need not worry about adding keywords.
- You can filter, sort, concatenate, and transform the results from `INFORMATION_SCHEMA` queries into whatever format your application needs, such as a data structure or a text representation to parse.
- This technique is more interoperable with other database systems. For example, Oracle Database users are familiar with querying tables in the Oracle data dictionary.

Because `SHOW` is familiar and widely used, the `SHOW` statements remain as an alternative. In fact, along with the implementation of `INFORMATION_SCHEMA`, there are enhancements to `SHOW` as described in Section 25.42, “Extensions to SHOW Statements”.

INFORMATION_SCHEMA and Privileges

Each MySQL user has the right to access these tables, but can see only the rows in the tables that correspond to objects for which the user has the proper access privileges. In some cases (for example, the `ROUTINE_DEFINITION` column in the `INFORMATION_SCHEMA ROUTINES` table), users who have insufficient privileges see `NULL`. These restrictions do not apply for `InnoDB` tables; you can see them with only the `PROCESS` privilege.

The same privileges apply to selecting information from `INFORMATION_SCHEMA` and viewing the same information through `SHOW` statements. In either case, you must have some privilege on an object to see information about it.

Performance Considerations

`INFORMATION_SCHEMA` queries that search for information from more than one database might take a long time and impact performance. To check the efficiency of a query, you can use `EXPLAIN`. For information about using `EXPLAIN` output to tune `INFORMATION_SCHEMA` queries, see Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”.

Standards Considerations

The implementation for the `INFORMATION_SCHEMA` table structures in MySQL follows the ANSI/ISO SQL:2003 standard Part 11 *Schemata*. Our intent is approximate compliance with SQL:2003 core feature F021 *Basic information schema*.

Users of SQL Server 2000 (which also follows the standard) may notice a strong similarity. However, MySQL has omitted many columns that are not relevant for our implementation, and added columns that are MySQL-specific. One such added column is the `ENGINE` column in the `INFORMATION_SCHEMA TABLES` table.

Although other DBMSs use a variety of names, like `syscat` or `system`, the standard name is `INFORMATION_SCHEMA`.

To avoid using any name that is reserved in the standard or in DB2, SQL Server, or Oracle, we changed the names of some columns marked “MySQL extension”. (For example, we changed `COLLATION` to `TABLE_COLLATION` in the `TABLES` table.) See the list of reserved words near the end of this article: https://web.archive.org/web/20070428032454/http://www.dbazine.com/db2/db2-disarticles/gulutzan5.

Conventions in the INFORMATION_SCHEMA Reference Sections

The following sections describe each of the tables and columns in `INFORMATION_SCHEMA`. For each column, there are three pieces of information:

- "`INFORMATION_SCHEMA` Name" indicates the name for the column in the `INFORMATION_SCHEMA` table. This corresponds to the standard SQL name unless the "Remarks" field says "MySQL extension."
- "`SHOW` Name" indicates the equivalent field name in the closest `SHOW` statement, if there is one.
- "Remarks" provides additional information where applicable. If this field is `NULL`, it means that the value of the column is always `NULL`. If this field says "MySQL extension," the column is a MySQL extension to standard SQL.

Many sections indicate what `SHOW` statement is equivalent to a `SELECT` that retrieves information from `INFORMATION_SCHEMA`. For `SHOW` statements that display information for the default database if you omit a `FROM db_name` clause, you can often select information for the default database by adding an `AND TABLE_SCHEMA = SCHEMA()` condition to the `WHERE` clause of a query that retrieves information from an `INFORMATION_SCHEMA` table.

Related Information

These sections discuss additional `INFORMATION_SCHEMA`-related topics:

- information about `INFORMATION_SCHEMA` tables specific to the `InnoDB` storage engine: Section 25.39, "INFORMATION_SCHEMA InnoDB Tables"
- information about `INFORMATION_SCHEMA` tables specific to the thread pool plugin: Section 25.40, "INFORMATION_SCHEMA Thread Pool Tables"
- information about `INFORMATION_SCHEMA` tables specific to the `CONNECTION_CONTROL` plugin: Section 25.41, "INFORMATION_SCHEMA Connection-Control Tables"
- Answers to questions that are often asked concerning the `INFORMATION_SCHEMA` database: Section A.7, "MySQL 8.0 FAQ: INFORMATION_SCHEMA"
- `INFORMATION_SCHEMA` queries and the optimizer: Section 8.2.3, "Optimizing INFORMATION_SCHEMA Queries"
- The effect of collation on `INFORMATION_SCHEMA` comparisons: Section 10.8.7, "Using Collation in INFORMATION_SCHEMA Searches"

25.2 The INFORMATION_SCHEMA CHARACTER_SETS Table

The `CHARACTER_SETS` table provides information about available character sets.

The `CHARACTER_SETS` table has these columns:

- `CHARACTER_SET_NAME`

  The character set name.

- `DEFAULT_COLLATE_NAME`

  The default collation for the character set.

- `DESCRIPTION`

A description of the character set.

- `MAXLEN`

  The maximum number of bytes required to store one character.

Notes

Character set information is also available from the `SHOW CHARACTER SET` statement. See Section 13.7.7.3, “SHOW CHARACTER SET Syntax”. The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
  [WHERE CHARACTER_SET_NAME LIKE 'wild']

SHOW CHARACTER SET
  [LIKE 'wild']
```

25.3 The INFORMATION_SCHEMA CHECK_CONSTRAINTS Table

The `CHECK_CONSTRAINTS` table (available as of MySQL 8.0.16) provides information about `CHECK` constraints defined on tables.

The `CHECK_CONSTRAINTS` table has these columns:

- `CONSTRAINT_CATALOG`

  The name of the catalog to which the constraint belongs. This value is always `def`.

- `CONSTRAINT_SCHEMA`

  The name of the schema (database) to which the constraint belongs.

- `CONSTRAINT_NAME`

  The name of the constraint.

- `CHECK_CLAUSE`

  The expression that specifies the constraint condition.

25.4 The INFORMATION_SCHEMA COLLATIONS Table

The `COLLATIONS` table provides information about collations for each character set.

The `COLLATIONS` table has these columns:

- `COLLATION_NAME`

  The collation name.

- `CHARACTER_SET_NAME`

  The name of the character set with which the collation is associated.

- `ID`

  The collation ID.

- IS_DEFAULT

  Whether the collation is the default for its character set.

- IS_COMPILED

  Whether the character set is compiled into the server.

- SORTLEN

  This is related to the amount of memory required to sort strings expressed in the character set.

- PAD_ATTRIBUTE

  The collation pad attribute.

Notes

Collation information is also available from the SHOW COLLATION statement. See Section 13.7.7.4, “SHOW COLLATION Syntax”. The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
  [WHERE COLLATION_NAME LIKE 'wild']

SHOW COLLATION
  [LIKE 'wild']
```

25.5 The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table

The COLLATION_CHARACTER_SET_APPLICABILITY table indicates what character set is applicable for what collation.

The COLLATION_CHARACTER_SET_APPLICABILITY table has these columns:

- COLLATION_NAME

  The collation name.

- CHARACTER_SET_NAME

  The name of the character set with which the collation is associated.

Notes

The COLLATION_CHARACTER_SET_APPLICABILITY columns are equivalent to the first two columns displayed by the SHOW COLLATION statement.

25.6 The INFORMATION_SCHEMA COLUMNS Table

The COLUMNS table provides information about columns in tables. The related ST_GEOMETRY_COLUMNS table provides information about table columns that store spatial data. See Section 25.27, “The INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS Table”.

The COLUMNS table has these columns:

- TABLE_CATALOG

The name of the catalog to which the table containing the column belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table containing the column belongs.

- `TABLE_NAME`

  The name of the table containing the column.

- `COLUMN_NAME`

  The name of the column.

- `ORDINAL_POSITION`

  The position of the column within the table. `ORDINAL_POSITION` is necessary because you might want to say `ORDER BY ORDINAL_POSITION`. Unlike `SHOW COLUMNS`, `SELECT` from the `COLUMNS` table does not have automatic ordering.

- `COLUMN_DEFAULT`

  The default value for the column. This is `NULL` if the column has an explicit default of `NULL`, or if the column definition includes no `DEFAULT` clause.

- `IS_NULLABLE`

  The column nullability. The value is `YES` if `NULL` values can be stored in the column, `NO` if not.

- `DATA_TYPE`

  The column data type.

  The `DATA_TYPE` value is the type name only with no other information. The `COLUMN_TYPE` value contains the type name and possibly other information such as the precision or length.

- `CHARACTER_MAXIMUM_LENGTH`

  For string columns, the maximum length in characters.

- `CHARACTER_OCTET_LENGTH`

  For string columns, the maximum length in bytes.

- `NUMERIC_PRECISION`

  For numeric columns, the numeric precision.

- `NUMERIC_SCALE`

  For numeric columns, the numeric scale.

- `DATETIME_PRECISION`

  For temporal columns, the fractional seconds precision.

- `CHARACTER_SET_NAME`

  For character string columns, the character set name.

- `COLLATION_NAME`

  For character string columns, the collation name.

- `COLUMN_TYPE`

  The column data type.

  The `DATA_TYPE` value is the type name only with no other information. The `COLUMN_TYPE` value contains the type name and possibly other information such as the precision or length.

- `COLUMN_KEY`

  Whether the column is indexed:

  - If `COLUMN_KEY` is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.
  - If `COLUMN_KEY` is `PRI`, the column is a `PRIMARY KEY` or is one of the columns in a multiple-column `PRIMARY KEY`.
  - If `COLUMN_KEY` is `UNI`, the column is the first column of a `UNIQUE` index. (A `UNIQUE` index permits multiple `NULL` values, but you can tell whether the column permits `NULL` by checking the `Null` column.)
  - If `COLUMN_KEY` is `MUL`, the column is the first column of a nonunique index in which multiple occurrences of a given value are permitted within the column.

  If more than one of the `COLUMN_KEY` values applies to a given column of a table, `COLUMN_KEY` displays the one with the highest priority, in the order `PRI`, `UNI`, `MUL`.

  A `UNIQUE` index may be displayed as `PRI` if it cannot contain `NULL` values and there is no `PRIMARY KEY` in the table. A `UNIQUE` index may display as `MUL` if several columns form a composite `UNIQUE` index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

- `EXTRA`

  Any additional information that is available about a given column. The value is nonempty in these cases:

  - `auto_increment` for columns that have the `AUTO_INCREMENT` attribute.
  - `on update CURRENT_TIMESTAMP` for `TIMESTAMP` or `DATETIME` columns that have the `ON UPDATE CURRENT_TIMESTAMP` attribute.
  - `VIRTUAL GENERATED` or `VIRTUAL STORED` for generated columns.
  - `DEFAULT_GENERATED` for columns that have an expression default value.

- `PRIVILEGES`

  The privileges you have for the column.

- `COLUMN_COMMENT`

  Any comment included in the column definition.

- `GENERATION_EXPRESSION`

For generated columns, displays the expression used to compute column values. Empty for nongenerated columns. For information about generated columns, see Section 13.1.20.9, “CREATE TABLE and Generated Columns”.

- `SRS_ID`

  This value applies to spatial columns. It contains the column `SRID` value that indicates the spatial reference system for values stored in the column. See Section 11.5.1, “Spatial Data Types”, and Section 11.5.5, “Spatial Reference System Support”. The value is `NULL` for nonspatial columns and spatial columns with no `SRID` attribute.

Notes

- In `SHOW COLUMNS`, the `Type` display includes values from several different `COLUMNS` columns.
- `CHARACTER_OCTET_LENGTH` should be the same as `CHARACTER_MAXIMUM_LENGTH`, except for multibyte character sets.
- `CHARACTER_SET_NAME` can be derived from `COLLATION_NAME`. For example, if you say `SHOW FULL COLUMNS FROM t`, and you see in the `COLLATION_NAME` column a value of `utf8_swedish_ci`, the character set is what is before the first underscore: `utf8`.

Column information is also available from the `SHOW COLUMNS` statement. See Section 13.7.7.5, “SHOW COLUMNS Syntax”. The following statements are nearly equivalent:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
  FROM INFORMATION_SCHEMA.COLUMNS
  WHERE table_name = 'tbl_name'
  [AND table_schema = 'db_name']
  [AND column_name LIKE 'wild']

SHOW COLUMNS
  FROM tbl_name
  [FROM db_name]
  [LIKE 'wild']
```

25.7 The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table

The `COLUMN_PRIVILEGES` table provides information about column privileges. It takes its values from the `mysql.columns_priv` system table.

The `COLUMN_PRIVILEGES` table has these columns:

- `GRANTEE`

  The name of the account to which the privilege is granted, in *`'user_name'@'host_name'`* format.

- `TABLE_CATALOG`

  The name of the catalog to which the table containing the column belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table containing the column belongs.

- `TABLE_NAME`

  The name of the table containing the column.

- `COLUMN_NAME`

  The name of the column.

- `PRIVILEGE_TYPE`

  The privilege granted. The value can be any privilege that can be granted at the column level; see Section 13.7.1.6, “GRANT Syntax”. Each row lists a single privilege, so there is one row per column privilege held by the grantee.

  In the output from `SHOW FULL COLUMNS`, the privileges are all in one column and in lowercase, for example, `select,insert,update,references`. In `COLUMN_PRIVILEGES`, there is one privilege per row, in uppercase.

- `IS_GRANTABLE`

  `YES` if the user has the `GRANT OPTION` privilege, `NO` otherwise. The output does not list `GRANT OPTION` as a separate row with `PRIVILEGE_TYPE='GRANT OPTION'`.

Notes

- The `COLUMN_PRIVILEGES` table is a nonstandard `INFORMATION_SCHEMA` table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES

SHOW GRANTS ...
```

25.8 The INFORMATION_SCHEMA COLUMN_STATISTICS Table

The `COLUMN_STATISTICS` table provides access to histogram statistics for column values.

For information about histogram statistics, see Section 8.9.6, “Optimizer Statistics”, and Section 13.7.3.1, “ANALYZE TABLE Syntax”.

You can see information only for columns for which you have some privilege.

The `COLUMN_STATISTICS` table has these columns:

- `SCHEMA_NAME`

  The names of the schema for which the statistics apply.

- `TABLE_NAME`

  The names of the column for which the statistics apply.

- `COLUMN_NAME`

  The names of the column for which the statistics apply.

- `HISTOGRAM`

  A `JSON` object describing the column statistics, stored as a histogram.

25.9 The INFORMATION_SCHEMA ENGINES Table

The ENGINES table provides information about storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is.

The ENGINES table has these columns:

- ENGINE

  The name of the storage engine.

- SUPPORT

  The server's level of support for the storage engine, as shown in the following table.

Value	Meaning
YES	The engine is supported and is active
DEFAULT	Like YES, plus this is the default engine
NO	The engine is not supported
DISABLED	The engine is supported but has been disabled

  A value of NO means that the server was compiled without support for the engine, so it cannot be enabled at runtime.

  A value of DISABLED occurs either because the server was started with an option that disables the engine, or because not all options required to enable it were given. In the latter case, the error log should contain a reason indicating why the option is disabled. See Section 5.4.2, “The Error Log”.

  You might also see DISABLED for a storage engine if the server was compiled to support it, but was started with a --skip-*engine_name* option. For the NDB storage engine, DISABLED means the server was compiled with support for NDB Cluster, but was not started with the --ndbcluster option.

  All MySQL servers support MyISAM tables. It is not possible to disable MyISAM.

- COMMENT

  A brief description of the storage engine.

- TRANSACTIONS

  Whether the storage engine supports transactions.

- XA

  Whether the storage engine supports XA transactions.

- SAVEPOINTS

  Whether the storage engine supports savepoints.

Notes

- The ENGINES table is a nonstandard INFORMATION_SCHEMA table.

Storage engine information is also available from the SHOW ENGINES statement. See Section 13.7.7.16, “SHOW ENGINES Syntax”. The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.ENGINES

SHOW ENGINES
```

25.10 The INFORMATION_SCHEMA EVENTS Table

The `EVENTS` table provides information about Event Manager events, which are discussed in Section 24.4, “Using the Event Scheduler”.

The `EVENTS` table has these columns:

- `EVENT_CATALOG`

  The name of the catalog to which the event belongs. This value is always `def`.

- `EVENT_SCHEMA`

  The name of the schema (database) to which the event belongs.

- `EVENT_NAME`

  The name of the event.

- `DEFINER`

  The account of the user who created the event, in `'user_name'@'host_name'` format.

- `TIME_ZONE`

  The event time zone, which is the time zone used for scheduling the event and that is in effect within the event as it executes. The default value is `SYSTEM`.

- `EVENT_BODY`

  The language used for the statements in the event's `DO` clause. The value is always `SQL`.

- `EVENT_DEFINITION`

  The text of the SQL statement making up the event's `DO` clause; in other words, the statement executed by this event.

- `EVENT_TYPE`

  The event repetition type, either `ONE TIME` (transient) or `RECURRING` (repeating).

- `EXECUTE_AT`

  For a one-time event, this is the `DATETIME` value specified in the `AT` clause of the `CREATE EVENT` statement used to create the event, or of the last `ALTER EVENT` statement that modified the event. The value shown in this column reflects the addition or subtraction of any `INTERVAL` value included in the event's `AT` clause. For example, if an event is created using `ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR`, and the event was created at 2018-02-09 14:05:30, the value shown in this column would be `'2018-02-10 20:05:30'`. If the event's timing is determined by an `EVERY` clause instead of an `AT` clause (that is, if the event is recurring), the value of this column is `NULL`.

- `INTERVAL_VALUE`

  For a recurring event, the number of intervals to wait between event executions. For a transient event, the value is always `NULL`.

- `INTERVAL_FIELD`

  The time units used for the interval which a recurring event waits before repeating. For a transient event, the value is always `NULL`.

- `SQL_MODE`

  The SQL mode in effect when the event was created or altered, and under which the event executes. For the permitted values, see Section 5.1.11, “Server SQL Modes”.

- `STARTS`

  The start date and time for a recurring event. This is displayed as a `DATETIME` value, and is `NULL` if no start date and time are defined for the event. For a transient event, this column is always `NULL`. For a recurring event whose definition includes a `STARTS` clause, this column contains the corresponding `DATETIME` value. As with the `EXECUTE_AT` column, this value resolves any expressions used. If there is no `STARTS` clause affecting the timing of the event, this column is `NULL`

- `ENDS`

  For a recurring event whose definition includes a `ENDS` clause, this column contains the corresponding `DATETIME` value. As with the `EXECUTE_AT` column, this value resolves any expressions used. If there is no `ENDS` clause affecting the timing of the event, this column is `NULL`.

- `STATUS`

  The event status. One of `ENABLED`, `DISABLED`, or `SLAVESIDE_DISABLED`. `SLAVESIDE_DISABLED` indicates that the creation of the event occurred on another MySQL server acting as a replication master and replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave. For more information, see Section 17.5.1.16, “Replication of Invoked Features”. information.

- `ON_COMPLETION`

  One of the two values `PRESERVE` or `NOT PRESERVE`.

- `CREATED`

  The date and time when the event was created. This is a `TIMESTAMP` value.

- `LAST_ALTERED`

  The date and time when the event was last modified. This is a `TIMESTAMP` value. If the event has not been modified since its creation, this value is the same as the `CREATED` value.

- `LAST_EXECUTED`

  The date and time when the event last executed. This is a `DATETIME` value. If the event has never executed, this column is `NULL`.

  `LAST_EXECUTED` indicates when the event started. As a result, the `ENDS` column is never less than `LAST_EXECUTED`.

- `EVENT_COMMENT`

  The text of the comment, if the event has one. If not, this value is empty.

- `ORIGINATOR`

The server ID of the MySQL server on which the event was created; used in replication. This value may be updated by `ALTER EVENT` to the server ID of the server on which that statement occurs, if executed on a master server. The default value is 0.

- `CHARACTER_SET_CLIENT`

  The session value of the `character_set_client` system variable when the event was created.

- `COLLATION_CONNECTION`

  The session value of the `collation_connection` system variable when the event was created.

- `DATABASE_COLLATION`

  The collation of the database with which the event is associated.

Notes

- The `EVENTS` table is a nonstandard `INFORMATION_SCHEMA` table.
- Times in the `EVENTS` table are displayed using the event time zone, the current session time zone, or UTC, as described in Section 24.4.4, “Event Metadata”.
- For more information about `SLAVESIDE_DISABLED` and the `ORIGINATOR` column, see Section 17.5.1.16, “Replication of Invoked Features”.

Example

Suppose that the user `'jon'@'ghidora'` creates an event named `e_daily`, and then modifies it a few minutes later using an `ALTER EVENT` statement, as shown here:

```
DELIMITER |

CREATE EVENT e_daily
    ON SCHEDULE
      EVERY 1 DAY
    COMMENT 'Saves total number of sessions then clears the table each day'
    DO
      BEGIN
        INSERT INTO site_activity.totals (time, total)
          SELECT CURRENT_TIMESTAMP, COUNT(*)
            FROM site_activity.sessions;
        DELETE FROM site_activity.sessions;
      END |

DELIMITER ;

ALTER EVENT e_daily
    ENABLE;
```

(Note that comments can span multiple lines.)

This user can then run the following `SELECT` statement, and obtain the output shown:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
       WHERE EVENT_NAME = 'e_daily'
       AND EVENT_SCHEMA = 'myschema'\G
```

```
*************************** 1. row ***************************
       EVENT_CATALOG: def
        EVENT_SCHEMA: myschema
          EVENT_NAME: e_daily
             DEFINER: jon@ghidora
           TIME_ZONE: SYSTEM
          EVENT_BODY: SQL
    EVENT_DEFINITION: BEGIN
        INSERT INTO site_activity.totals (time, total)
          SELECT CURRENT_TIMESTAMP, COUNT(*)
            FROM site_activity.sessions;
        DELETE FROM site_activity.sessions;
      END
          EVENT_TYPE: RECURRING
          EXECUTE_AT: NULL
      INTERVAL_VALUE: 1
      INTERVAL_FIELD: DAY
            SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
                      NO_ZERO_IN_DATE,NO_ZERO_DATE,
                      ERROR_FOR_DIVISION_BY_ZERO,
                      NO_ENGINE_SUBSTITUTION
              STARTS: 2018-08-08 11:06:34
                ENDS: NULL
              STATUS: ENABLED
       ON_COMPLETION: NOT PRESERVE
             CREATED: 2018-08-08 11:06:34
        LAST_ALTERED: 2018-08-08 11:06:34
       LAST_EXECUTED: 2018-08-08 16:06:34
       EVENT_COMMENT: Saves total number of sessions then clears the
                      table each day
          ORIGINATOR: 1
CHARACTER_SET_CLIENT: utf8mb4
COLLATION_CONNECTION: utf8mb4_0900_ai_ci
  DATABASE_COLLATION: utf8mb4_0900_ai_ci
```

Event information is also available from the `SHOW EVENTS` statement. See Section 13.7.7.18, “SHOW EVENTS Syntax”. The following statements are equivalent:

```
SELECT
    EVENT_SCHEMA, EVENT_NAME, DEFINER, TIME_ZONE, EVENT_TYPE, EXECUTE_AT,
    INTERVAL_VALUE, INTERVAL_FIELD, STARTS, ENDS, STATUS, ORIGINATOR,
    CHARACTER_SET_CLIENT, COLLATION_CONNECTION, DATABASE_COLLATION
  FROM INFORMATION_SCHEMA.EVENTS
  WHERE table_schema = 'db_name'
  [AND column_name LIKE 'wild']

SHOW EVENTS
  [FROM db_name]
  [LIKE 'wild']
```

25.11 The INFORMATION_SCHEMA FILES Table

The `FILES` table provides information about the files in which MySQL tablespace data is stored.

The `FILES` table provides information about `InnoDB` data files. In NDB Cluster, this table also provides information about the files in which NDB Cluster Disk Data tables are stored. For additional information specific to `InnoDB`, see InnoDB Notes, later in this section; for additional information specific to NDB Cluster, see NDB Notes.

The `FILES` table has these columns:

- `FILE_ID`

For InnoDB: The tablespace ID, also referred to as the space_id or fil_space_t::id.

For NDB: A file identifier. FILE_ID column values are auto-generated.

- FILE_NAME

  For InnoDB: The name of the data file. File-per-table and general tablespaces have an .ibd file name extension. Undo tablespaces are prefixed by undo. The system tablespace is prefixed by ibdata. The global temporary tablespace is prefixed by ibtmp. The file name includes the file path, which may be relative to the MySQL data directory (the value of the datadir system variable).

  For NDB: The name of an UNDO log file created by CREATE LOGFILE GROUP or ALTER LOGFILE GROUP, or of a data file created by CREATE TABLESPACE or ALTER TABLESPACE.

- FILE_TYPE

  For InnoDB: The tablespace file type. There are three possible file types for InnoDB files. TABLESPACE is the file type for any system, general, or file-per-table tablespace file that holds tables, indexes, or other forms of user data. TEMPORARY is the file type for temporary tablespaces. UNDO LOG is the file type for undo tablespaces, which hold undo records.

  For NDB: One of the values UNDO LOG, DATAFILE, or TABLESPACE.

- TABLESPACE_NAME

  For InnoDB: The SQL name for the tablespace. A general tablespace name is the SYS_TABLESPACES.NAME value. For other tablespace files, names start with innodb_, such as innodb_system, innodb_undo, and innodb_file_per_table. The file-per-table tablespace name format is innodb_file_per_table_##, where ## is the tablespace ID.

  For NDB: The name of the tablespace with which the file is associated.

- TABLE_CATALOG

  This value is always empty.

- TABLE_SCHEMA

  This value is always NULL.

- TABLE_NAME

  For InnoDB: This value is always NULL.

  For NDB: The name of the Disk Data table with which the file is associated, if any.

- LOGFILE_GROUP_NAME

  For InnoDB: This value is always NULL.

  For NDB: The name of the log file group to which the log file or data file belongs.

- LOGFILE_GROUP_NUMBER

  For InnoDB: This value is always NULL.

  For NDB: For an UNDO log file, the auto-generated ID number of the log file group to which the log file belongs.

- `ENGINE`

  For `InnoDB`: This value is always `InnoDB`.

  For `NDB`: For an NDB Cluster Disk Data log file or data file, this value is always `NDB` or `NDBCLUSTER`.

- `FULLTEXT_KEYS`

  For `InnoDB`: This value is always `NULL`.

  For `NDB`: For an NDB Cluster Disk Data log file or data file, this value is always empty.

- `DELETED_ROWS`

  For `InnoDB`: This value is always `NULL`.

- `UPDATE_COUNT`

  For `InnoDB`: This value is always `NULL`.

- `FREE_EXTENTS`

  For `InnoDB`: The number of fully free extents in the current data file.

  For `NDB`: The number of extents which have not yet been used by the file.

- `TOTAL_EXTENTS`

  For `InnoDB`: The number of full extents used in the current data file. Any partial extent at the end of the file is not counted.

  For `NDB`: The total number of extents allocated to the file.

- `EXTENT_SIZE`

  For `InnoDB`: Extent size is 1048576 (1MB) for files with a 4KB, 8KB, or 16KB page size. Extent size is 2097152 bytes (2MB) for files with a 32KB page size, and 4194304 (4MB) for files with a 64KB page size. `FILES` does not report `InnoDB` page size. Page size is defined by the `innodb_page_size` system variable. Extent size information can also be retrieved from the `INNODB_TABLESPACES` table where `FILES.FILE_ID = INNODB_TABLESPACES.SPACE`.

  For `NDB`: The size of an extent for the file in bytes.

- `INITIAL_SIZE`

  For `InnoDB`: The initial size of the file in bytes.

  For `NDB`: The size of the file in bytes. This is the same value that was used in the `INITIAL_SIZE` clause of the `CREATE LOGFILE GROUP`, `ALTER LOGFILE GROUP`, `CREATE TABLESPACE`, or `ALTER TABLESPACE` statement used to create the file.

- `MAXIMUM_SIZE`

  For `InnoDB`: The maximum number of bytes permitted in the file. The value is `NULL` for all data files except for predefined system tablespace data files. Maximum system tablespace file size is defined by `innodb_data_file_path`. Maximum global temporary tablespace file size is defined by `innodb_temp_data_file_path`. A `NULL` value for a predefined system tablespace data file indicates that a file size limit was not defined explicitly.

For NDB: For NDB Cluster Disk Data files, this value is always the same as the INITIAL_SIZE value.

- AUTOEXTEND_SIZE

  For InnoDB: AUTOEXTEND_SIZE is the auto-extend size defined by innodb_data_file_path for the system tablespace, or by innodb_temp_data_file_path for the global temporary tablespace.

  For NDB: For NDB Cluster Disk Data files, this value is always empty.

- CREATION_TIME

  For InnoDB: This value is always NULL.

  For NDB: The date and time when the file was created.

- LAST_UPDATE_TIME

  For InnoDB: This value is always NULL.

  For NDB: The date and time when the file was last modified.

- LAST_ACCESS_TIME

  For InnoDB: This value is always NULL.

  For NDB: The date and time when the file was last accessed by the server.

- RECOVER_TIME

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always 0.

- TRANSACTION_COUNTER

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always 0.

- VERSION

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- ROW_FORMAT

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- TABLE_ROWS

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- AVG_ROW_LENGTH

For InnoDB: This value is always NULL.

For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- DATA_LENGTH

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- MAX_DATA_LENGTH

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- INDEX_LENGTH

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- DATA_FREE

  For InnoDB: The total amount of free space (in bytes) for the entire tablespace. Predefined system tablespaces, which include the system tablespace and temporary table tablespaces, may have one or more data files.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- CREATE_TIME

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- UPDATE_TIME

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- CHECK_TIME

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- CHECKSUM

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, this value is always NULL.

- STATUS

  For InnoDB: This value is NORMAL by default. InnoDB file-per-table tablespaces may report IMPORTING, which indicates that the tablespace is not yet available.

For NDB: For NDB Cluster Disk Data files, this value is always NORMAL.

- EXTRA

  For InnoDB: This value is always NULL.

  For NDB: For NDB Cluster Disk Data files, the EXTRA column shows which data node the file belongs to (each data node having its own copy), as well as the size of its undo buffer. Suppose that you use this statement on an NDB Cluster with four data nodes:

```
CREATE LOGFILE GROUP mygroup
    ADD UNDOFILE 'new_undo.dat'
    INITIAL_SIZE 2G
    ENGINE NDB;
```

  After running the CREATE LOGFILE GROUP statement successfully, you should see a result similar to the one shown here for this query against the FILES table:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_TYPE, EXTRA
         FROM INFORMATION_SCHEMA.FILES
         WHERE FILE_NAME = 'new_undo.dat';

+--------------------+-----------+----------------------------------------+
| LOGFILE_GROUP_NAME | FILE_TYPE | EXTRA                                  |
+--------------------+-----------+----------------------------------------+
| mygroup            | UNDO LOG  | CLUSTER_NODE=5;UNDO_BUFFER_SIZE=8388608 |
| mygroup            | UNDO LOG  | CLUSTER_NODE=6;UNDO_BUFFER_SIZE=8388608 |
| mygroup            | UNDO LOG  | CLUSTER_NODE=7;UNDO_BUFFER_SIZE=8388608 |
| mygroup            | UNDO LOG  | CLUSTER_NODE=8;UNDO_BUFFER_SIZE=8388608 |
+--------------------+-----------+----------------------------------------+
```

  This information was not included in NDB Cluster 8.0 prior to NDB 8.0.15. (Bug #92796, Bug #28800252)

Notes

- The FILES table is a nonstandard INFORMATION_SCHEMA table.

InnoDB Notes

The following notes apply to InnoDB data files.

- Data reported by FILES is reported from the InnoDB in-memory cache for open files. By comparison, INNODB_DATAFILES reports data from the InnoDB SYS_DATAFILES internal data dictionary table.
- The data reported by FILES includes global temporary tablespace data. This data is not available in the InnoDB SYS_DATAFILES internal data dictionary table, and is therefore not reported by INNODB_DATAFILES.
- Undo tablespace data is reported by FILES when separate undo tablespaces are present, which they are by default in MySQL 8.0
- The following query returns all data pertinent to InnoDB tablespaces.

```
SELECT
  FILE_ID, FILE_NAME, FILE_TYPE, TABLESPACE_NAME, FREE_EXTENTS,
  TOTAL_EXTENTS, EXTENT_SIZE, INITIAL_SIZE, MAXIMUM_SIZE,
  AUTOEXTEND_SIZE, DATA_FREE, STATUS
```

```
FROM INFORMATION_SCHEMA.FILES WHERE ENGINE='InnoDB'\G
```

NDB Notes

- The `FILES` table provides information about Disk Data *files* only; you cannot use it for determining disk space allocation or availability for individual `NDB` tables. However, it is possible to see how much space is allocated for each `NDB` table having data stored on disk—as well as how much remains available for storage of data on disk for that table—using `ndb_desc`. For more information, see Section 22.4.9, “ndb_desc — Describe NDB Tables”.

- The `CREATION_TIME`, `LAST_UPDATE_TIME`, and `LAST_ACCESSED` values are as reported by the operating system, and are not supplied by the `NDB` storage engine. Where no value is provided by the operating system, these columns display `0000-00-00 00:00:00`.

- The difference between the `TOTAL EXTENTS` and `FREE_EXTENTS` columns is the number of extents currently in use by the file:

  ```
  SELECT TOTAL_EXTENTS - FREE_EXTENTS AS extents_used
      FROM INFORMATION_SCHEMA.FILES
      WHERE FILE_NAME = 'myfile.dat';
  ```

  To approximate the amount of disk space in use by the file, multiply that difference by the value of the `EXTENT_SIZE` column, which gives the size of an extent for the file in bytes:

  ```
  SELECT (TOTAL_EXTENTS - FREE_EXTENTS) * EXTENT_SIZE AS bytes_used
      FROM INFORMATION_SCHEMA.FILES
      WHERE FILE_NAME = 'myfile.dat';
  ```

  Similarly, you can estimate the amount of space that remains available in a given file by multiplying `FREE_EXTENTS` by `EXTENT_SIZE`:

  ```
  SELECT FREE_EXTENTS * EXTENT_SIZE AS bytes_free
      FROM INFORMATION_SCHEMA.FILES
      WHERE FILE_NAME = 'myfile.dat';
  ```

  Important

  The byte values produced by the preceding queries are approximations only, and their precision is inversely proportional to the value of `EXTENT_SIZE`. That is, the larger `EXTENT_SIZE` becomes, the less accurate the approximations are.

  It is also important to remember that once an extent is used, it cannot be freed again without dropping the data file of which it is a part. This means that deletes from a Disk Data table do *not* release disk space.

  The extent size can be set in a `CREATE TABLESPACE` statement. For more information, see Section 13.1.21, “CREATE TABLESPACE Syntax”.

- An additional row is present in the `FILES` table following the creation of a logfile group. This row has `NULL` for the value of the `FILE_NAME` column. For this row, the value of the `FILE_ID` column is always `0`, that of the `FILE_TYPE` column is always `UNDO FILE`, and that of the `STATUS` column is always `NORMAL`. The value of the `ENGINE` column is always `NDBCLUSTER`.

  The `FREE_EXTENTS` column in this row shows the total number of free extents available to all undo files belonging to a given log file group whose name and number are shown in the `LOGFILE_GROUP_NAME` and `LOGFILE_GROUP_NUMBER` columns, respectively.

Suppose there are no existing log file groups on your NDB Cluster, and you create one using the following statement:

```
mysql> CREATE LOGFILE GROUP lg1
         ADD UNDOFILE 'undofile.dat'
         INITIAL_SIZE = 16M
         UNDO_BUFFER_SIZE = 1M
         ENGINE = NDB;
```

You can now see this `NULL` row when you query the `FILES` table:

```
mysql> SELECT DISTINCT
         FILE_NAME AS File,
         FREE_EXTENTS AS Free,
         TOTAL_EXTENTS AS Total,
         EXTENT_SIZE AS Size,
         INITIAL_SIZE AS Initial
         FROM INFORMATION_SCHEMA.FILES;
+--------------+---------+---------+------+----------+
| File         | Free    | Total   | Size | Initial  |
+--------------+---------+---------+------+----------+
| undofile.dat |    NULL | 4194304 |    4 | 16777216 |
| NULL         | 4184068 |    NULL |    4 |     NULL |
+--------------+---------+---------+------+----------+
```

The total number of free extents available for undo logging is always somewhat less than the sum of the `TOTAL_EXTENTS` column values for all undo files in the log file group due to overhead required for maintaining the undo files. This can be seen by adding a second undo file to the log file group, then repeating the previous query against the `FILES` table:

```
mysql> ALTER LOGFILE GROUP lg1
         ADD UNDOFILE 'undofile02.dat'
         INITIAL_SIZE = 4M
         ENGINE = NDB;

mysql> SELECT DISTINCT
         FILE_NAME AS File,
         FREE_EXTENTS AS Free,
         TOTAL_EXTENTS AS Total,
         EXTENT_SIZE AS Size,
         INITIAL_SIZE AS Initial
         FROM INFORMATION_SCHEMA.FILES;
+----------------+---------+---------+------+----------+
| File           | Free    | Total   | Size | Initial  |
+----------------+---------+---------+------+----------+
| undofile.dat   |    NULL | 4194304 |    4 | 16777216 |
| undofile02.dat |    NULL | 1048576 |    4 |  4194304 |
| NULL           | 5223944 |    NULL |    4 |     NULL |
+----------------+---------+---------+------+----------+
```

The amount of free space in bytes which is available for undo logging by Disk Data tables using this log file group can be approximated by multiplying the number of free extents by the initial size:

```
mysql> SELECT
         FREE_EXTENTS AS 'Free Extents',
         FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
         FROM INFORMATION_SCHEMA.FILES
         WHERE LOGFILE_GROUP_NAME = 'lg1'
         AND FILE_NAME IS NULL;
+--------------+------------+
```

```
| Free Extents | Free Bytes |
+--------------+------------+
|      5223944 |   20895776 |
+--------------+------------+
```

If you create an NDB Cluster Disk Data table and then insert some rows into it, you can see approximately how much space remains for undo logging afterward, for example:

```
mysql> CREATE TABLESPACE ts1
         ADD DATAFILE 'data1.dat'
         USE LOGFILE GROUP lg1
         INITIAL_SIZE 512M
         ENGINE = NDB;

mysql> CREATE TABLE dd (
         c1 INT NOT NULL PRIMARY KEY,
         c2 INT,
         c3 DATE
         )
         TABLESPACE ts1 STORAGE DISK
         ENGINE = NDB;

mysql> INSERT INTO dd VALUES
         (NULL, 1234567890, '2007-02-02'),
         (NULL, 1126789005, '2007-02-03'),
         (NULL, 1357924680, '2007-02-04'),
         (NULL, 1642097531, '2007-02-05');

mysql> SELECT
         FREE_EXTENTS AS 'Free Extents',
         FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
         FROM INFORMATION_SCHEMA.FILES
         WHERE LOGFILE_GROUP_NAME = 'lg1'
         AND FILE_NAME IS NULL;
+--------------+------------+
| Free Extents | Free Bytes |
+--------------+------------+
|      5207565 |   20830260 |
+--------------+------------+
```

- An additional row is present in the `FILES` table for any NDB Cluster tablespace, whether or not any data files are associated with the tablespace. This row has `NULL` for the value of the `FILE_NAME` column. For this row, the value of the `FILE_ID` column is always `0`, that of the `FILE_TYPE` column is always `TABLESPACE`, and that of the `STATUS` column is always `NORMAL`. The value of the `ENGINE` column is always `NDBCLUSTER`.
- For additional information, and examples of creating and dropping NDB Cluster Disk Data objects, see Section 22.5.13, “NDB Cluster Disk Data Tables”.

25.12 The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table

The `KEY_COLUMN_USAGE` table describes which key columns have constraints. This table provides no information about functional key parts because they are expressions and the table provides information only about columns.

The `KEY_COLUMN_USAGE` table has these columns:

- `CONSTRAINT_CATALOG`

  The name of the catalog to which the constraint belongs. This value is always `def`.

- `CONSTRAINT_SCHEMA`

The name of the schema (database) to which the constraint belongs.

- `CONSTRAINT_NAME`

  The name of the constraint.

- `TABLE_CATALOG`

  The name of the catalog to which the table belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table belongs.

- `TABLE_NAME`

  The name of the table that has the constraint.

- `COLUMN_NAME`

  The name of the column that has the constraint.

  If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.

- `ORDINAL_POSITION`

  The column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.

- `POSITION_IN_UNIQUE_CONSTRAINT`

  `NULL` for unique and primary-key constraints. For foreign-key constraints, this column is the ordinal position in key of the table that is being referenced.

- `REFERENCED_TABLE_SCHEMA`

  The name of the schema referenced by the constraint.

- `REFERENCED_TABLE_NAME`

  The name of the table referenced by the constraint.

- `REFERENCED_COLUMN_NAME`

  The name of the column referenced by the constraint.

Suppose that there are two tables name `t1` and `t3` that have the following definitions:

```
CREATE TABLE t1
(
    s1 INT,
    s2 INT,
    s3 INT,
    PRIMARY KEY(s3)
) ENGINE=InnoDB;

CREATE TABLE t3
(
```

```
    s1 INT,
    s2 INT,
    s3 INT,
    KEY(s1),
    CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
) ENGINE=InnoDB;
```

For those two tables, the KEY_COLUMN_USAGE table has two rows:

- One row with CONSTRAINT_NAME = 'PRIMARY', TABLE_NAME = 't1', COLUMN_NAME = 's3', ORDINAL_POSITION = 1, POSITION_IN_UNIQUE_CONSTRAINT = NULL.

  For NDB: This value is always NULL.

- One row with CONSTRAINT_NAME = 'CO', TABLE_NAME = 't3', COLUMN_NAME = 's2', ORDINAL_POSITION = 1, POSITION_IN_UNIQUE_CONSTRAINT = 1.

25.13 The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table

The ndb_transid_mysql_connection_map table provides a mapping between NDB transactions, NDB transaction coordinators, and MySQL Servers attached to an NDB Cluster as API nodes. This information is used when populating the server_operations and server_transactions tables of the ndbinfo NDB Cluster information database.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
mysql_connection_id		MySQL Server connection ID
node_id		Transaction coordinator node ID
ndb_transid		NDB transaction ID

The mysql_connection_id is the same as the connection or session ID shown in the output of SHOW PROCESSLIST.

There are no SHOW statements associated with this table.

This is a nonstandard table, specific to NDB Cluster. It is implemented as an INFORMATION_SCHEMA plugin; you can verify that it is supported by checking the output of SHOW PLUGINS. If ndb_transid_mysql_connection_map support is enabled, the output from this statement includes a plugin having this name, of type INFORMATION SCHEMA, and having status ACTIVE, as shown here (using emphasized text):

```
mysql> SHOW PLUGINS;
+----------------------------------+--------+--------------------+---------+---------+
| Name                             | Status | Type               | Library | License |
+----------------------------------+--------+--------------------+---------+---------+
| binlog                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| mysql_native_password            | ACTIVE | AUTHENTICATION     | NULL    | GPL     |
| CSV                              | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MEMORY                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MRG_MYISAM                       | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MyISAM                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| PERFORMANCE_SCHEMA               | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| BLACKHOLE                        | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ARCHIVE                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndbcluster                       | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndbinfo                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndb_transid_mysql_connection_map | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
```

```
| InnoDB                            | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| INNODB_TRX                        | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCKS                      | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCK_WAITS                 | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP                        | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_RESET                  | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM                     | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM_RESET               | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| partition                         | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
+-----------------------------------+--------+--------------------+---------+---------+
22 rows in set (0.00 sec)
```

The plugin is enabled by default. You can disable it (or force the server not to run unless the plugin starts) by starting the server with the `--ndb-transid-mysql-connection-map` option. If the plugin is disabled, the status is shown by `SHOW PLUGINS` as `DISABLED`. The plugin cannot be enabled or disabled at runtime.

Although the names of this table and its columns are displayed using lowercase, you can use uppercase or lowercase when referring to them in SQL statements.

For this table to be created, the MySQL Server must be a binary supplied with the NDB Cluster distribution, or one built from the NDB Cluster sources with `NDB` storage engine support enabled. It is not available in the standard MySQL 8.0 Server.

25.14 The INFORMATION_SCHEMA KEYWORDS Table

The `KEYWORDS` table lists the words considered keywords by MySQL and, for each one, indicates whether it is reserved. Reserved keywords may require special treatment in some contexts, such as special quoting when used as identifiers (see Section 9.3, “Keywords and Reserved Words”). This table provides applications a runtime source of MySQL keyword information.

Prior to MySQL 8.0.13, selecting from the `KEYWORDS` table with no default database selected produced an error. (Bug #90160, Bug #27729859)

The `KEYWORDS` table has these columns:

- `WORD`

  The keyword.

- `RESERVED`

  An integer indicating whether the keyword is reserved (1) or nonreserved (0).

These queries lists all keywords, all reserved keywords, and all nonreserved keywords, respectively:

```
SELECT * FROM INFORMATION_SCHEMA.KEYWORDS;
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE RESERVED = 1;
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE RESERVED = 0;
```

The latter two queries are equivalent to:

```
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE RESERVED;
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE NOT RESERVED;
```

If you build MySQL from source, the build process generates a `keyword_list.h` header file containing an array of keywords and their reserved status. This file can be found in the `sql` directory under the build directory. This file may be useful for applications that require a static source for the keyword list.

25.15 The INFORMATION_SCHEMA OPTIMIZER_TRACE Table

The `OPTIMIZER_TRACE` table provides information produced by the optimizer tracing capability for traced statements. To enable tracking, use the `optimizer_trace` system variable. For details, see MySQL Internals: Tracing the Optimizer.

The `OPTIMIZER_TRACE` table has these columns:

- `QUERY`

  The text of the traced statement.

- `TRACE`

  The trace, in `JSON` format.

- `MISSING_BYTES_BEYOND_MAX_MEM_SIZE`

  Each remembered trace is a string that is extended as optimization progresses and appends data to it. The `optimizer_trace_max_mem_size` variable sets a limit on the total amount of memory used by all currently remembered traces. If this limit is reached, the current trace is not extended (and thus is incomplete), and the `MISSING_BYTES_BEYOND_MAX_MEM_SIZE` column shows the number of bytes missing from the trace.

- `INSUFFICIENT_PRIVILEGES`

  If a traced query uses views or stored routines that have `SQL SECURITY` with a value of `DEFINER`, it may be that a user other than the definer is denied from seeing the trace of the query. In that case, the trace is shown as empty and `INSUFFICIENT_PRIVILEGES` has a value of 1. Otherwise, the value is 0.

25.16 The INFORMATION_SCHEMA PARAMETERS Table

The `PARAMETERS` table provides information about parameters for stored routines (stored procedures and stored functions), and about return values for stored functions. The `PARAMETERS` table does not include built-in SQL functions or user-defined functions (UDFs).

The `PARAMETERS` table has these columns:

- `SPECIFIC_CATALOG`

  The name of the catalog to which the routine containing the parameter belongs. This value is always `def`.

- `SPECIFIC_SCHEMA`

  The name of the schema (database) to which the routine containing the parameter belongs.

- `SPECIFIC_NAME`

  The name of the routine containing the parameter.

- `ORDINAL_POSITION`

  For successive parameters of a stored procedure or function, the `ORDINAL_POSITION` values are 1, 2, 3, and so forth. For a stored function, there is also a row that applies to the function return value (as described by the `RETURNS` clause). The return value is not a true parameter, so the row that describes it has these unique characteristics:

  - The `ORDINAL_POSITION` value is 0.
  - The `PARAMETER_NAME` and `PARAMETER_MODE` values are `NULL` because the return value has no name and the mode does not apply.

- `PARAMETER_MODE`

  The mode of the parameter. This value is one of `IN`, `OUT`, or `INOUT`. For a stored function return value, this value is `NULL`.

- `PARAMETER_NAME`

  The name of the parameter. For a stored function return value, this value is `NULL`.

- `DATA_TYPE`

  The parameter data type.

  The `DATA_TYPE` value is the type name only with no other information. The `DTD_IDENTIFIER` value contains the type name and possibly other information such as the precision or length.

- `CHARACTER_MAXIMUM_LENGTH`

  For string parameters, the maximum length in characters.

- `CHARACTER_OCTET_LENGTH`

  For string parameters, the maximum length in bytes.

- `NUMERIC_PRECISION`

  For numeric parameters, the numeric precision.

- `NUMERIC_SCALE`

  For numeric parameters, the numeric scale.

- `DATETIME_PRECISION`

  For temporal parameters, the fractional seconds precision.

- `CHARACTER_SET_NAME`

  For character string parameters, the character set name.

- `COLLATION_NAME`

  For character string parameters, the collation name.

- `DTD_IDENTIFIER`

  The parameter data type.

  The `DATA_TYPE` value is the type name only with no other information. The `DTD_IDENTIFIER` value contains the type name and possibly other information such as the precision or length.

- `ROUTINE_TYPE`

  `PROCEDURE` for stored procedures, `FUNCTION` for stored functions.

25.17 The INFORMATION_SCHEMA PARTITIONS Table

The `PARTITIONS` table provides information about table partitions. Each row in this table corresponds to an individual partition or subpartition of a partitioned table. For more information about partitioning tables, see Chapter 23, *Partitioning*.

The `PARTITIONS` table has these columns:

- `TABLE_CATALOG`

  The name of the catalog to which the table belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table belongs.

- `TABLE_NAME`

  The name of the table containing the partition.

- `PARTITION_NAME`

  The name of the partition.

- `SUBPARTITION_NAME`

  If the `PARTITIONS` table row represents a subpartition, the name of subpartition; otherwise `NULL`.

  For `NDB`: This value is always `NULL`.

- `PARTITION_ORDINAL_POSITION`

  All partitions are indexed in the same order as they are defined, with `1` being the number assigned to the first partition. The indexing can change as partitions are added, dropped, and reorganized; the number shown is this column reflects the current order, taking into account any indexing changes.

- `SUBPARTITION_ORDINAL_POSITION`

  Subpartitions within a given partition are also indexed and reindexed in the same manner as partitions are indexed within a table.

- `PARTITION_METHOD`

  One of the values `RANGE`, `LIST`, `HASH`, `LINEAR HASH`, `KEY`, or `LINEAR KEY`; that is, one of the available partitioning types as discussed in Section 23.2, “Partitioning Types”.

- `SUBPARTITION_METHOD`

  One of the values `HASH`, `LINEAR HASH`, `KEY`, or `LINEAR KEY`; that is, one of the available subpartitioning types as discussed in Section 23.2.6, “Subpartitioning”.

- `PARTITION_EXPRESSION`

  The expression for the partitioning function used in the `CREATE TABLE` or `ALTER TABLE` statement that created the table's current partitioning scheme.

  For example, consider a partitioned table created in the `test` database using this statement:

```
CREATE TABLE tp (
```

```
    c1 INT,
    c2 INT,
    c3 VARCHAR(25)
)
PARTITION BY HASH(c1 + c2)
PARTITIONS 4;
```

The PARTITION_EXPRESSION column in a PARTITIONS table row for a partition from this table displays c1 + c2, as shown here:

```
mysql> SELECT DISTINCT PARTITION_EXPRESSION
       FROM INFORMATION_SCHEMA.PARTITIONS
       WHERE TABLE_NAME='tp' AND TABLE_SCHEMA='test';
+----------------------+
| PARTITION_EXPRESSION |
+----------------------+
| c1 + c2              |
+----------------------+
```

- SUBPARTITION_EXPRESSION

  This works in the same fashion for the subpartitioning expression that defines the subpartitioning for a table as PARTITION_EXPRESSION does for the partitioning expression used to define a table's partitioning.

  If the table has no subpartitions, this column is NULL.

- PARTITION_DESCRIPTION

  This column is used for RANGE and LIST partitions. For a RANGE partition, it contains the value set in the partition's VALUES LESS THAN clause, which can be either an integer or MAXVALUE. For a LIST partition, this column contains the values defined in the partition's VALUES IN clause, which is a list of comma-separated integer values.

  For partitions whose PARTITION_METHOD is other than RANGE or LIST, this column is always NULL.

- TABLE_ROWS

  The number of table rows in the partition.

  For partitioned InnoDB tables, the row count given in the TABLE_ROWS column is only an estimated value used in SQL optimization, and may not always be exact.

  For NDB tables, you can also obtain this information using the ndb_desc utility.

- AVG_ROW_LENGTH

  The average length of the rows stored in this partition or subpartition, in bytes. This is the same as DATA_LENGTH divided by TABLE_ROWS.

  For NDB tables, you can also obtain this information using the ndb_desc utility.

- DATA_LENGTH

  The total length of all rows stored in this partition or subpartition, in bytes; that is, the total number of bytes stored in the partition or subpartition.

  For NDB tables, you can also obtain this information using the ndb_desc utility.

- MAX_DATA_LENGTH

The maximum number of bytes that can be stored in this partition or subpartition.

For NDB tables, you can also obtain this information using the ndb_desc utility.

- INDEX_LENGTH

  The length of the index file for this partition or subpartition, in bytes.

  For partitions of NDB tables, whether the tables use implicit or explicit partitioning, the INDEX_LENGTH column value is always 0. However, you can obtain equivalent information using the ndb_desc utility.

- DATA_FREE

  The number of bytes allocated to the partition or subpartition but not used.

  For NDB tables, you can also obtain this information using the ndb_desc utility.

- CREATE_TIME

  The time that the partition or subpartition was created.

- UPDATE_TIME

  The time that the partition or subpartition was last modified.

- CHECK_TIME

  The last time that the table to which this partition or subpartition belongs was checked.

  For partitioned InnoDB tables, the value is always NULL.

- CHECKSUM

  The checksum value, if any; otherwise NULL.

- PARTITION_COMMENT

  The text of the comment, if the partition has one. If not, this value is empty.

  The maximum length for a partition comment is defined as 1024 characters, and the display width of the PARTITION_COMMENT column is also 1024, characters to match this limit.

- NODEGROUP

  This is the nodegroup to which the partition belongs. This is relevant only to NDB Cluster tables; otherwise, the value is always 0.

- TABLESPACE_NAME

  The name of the tablespace to which the partition belongs. The value is always DEFAULT, unless the table uses the NDB storage engine (see the *Notes* at the end of this section).

Notes

- The PARTITIONS table is a nonstandard INFORMATION_SCHEMA table.
- A table using any storage engine other than NDB and which is not partitioned has one row in the PARTITIONS table. However, the values of the PARTITION_NAME, SUBPARTITION_NAME,

PARTITION_ORDINAL_POSITION, SUBPARTITION_ORDINAL_POSITION, PARTITION_METHOD, SUBPARTITION_METHOD, PARTITION_EXPRESSION, SUBPARTITION_EXPRESSION, and PARTITION_DESCRIPTION columns are all NULL. Also, the PARTITION_COMMENT column in this case is blank.

- An NDB table which is not explicitly partitioned has one row in the PARTITIONS table for each data node in the NDB cluster. For each such row:
  - The SUBPARTITION_NAME, SUBPARTITION_ORDINAL_POSITION, SUBPARTITION_METHOD, PARTITION_EXPRESSION, SUBPARTITION_EXPRESSION, CREATE_TIME, UPDATE_TIME, CHECK_TIME, CHECKSUM, and TABLESPACE_NAME columns are all NULL.
  - The PARTITION_METHOD is always AUTO.
  - The NODEGROUP column is default.
  - The PARTITION_EXPRESSION and PARTITION_COMMENT columns are empty.

25.18 The INFORMATION_SCHEMA PLUGINS Table

The PLUGINS table provides information about server plugins.

The PLUGINS table has these columns:

- PLUGIN_NAME

  The name used to refer to the plugin in statements such as INSTALL PLUGIN and UNINSTALL PLUGIN.

- PLUGIN_VERSION

  The version from the plugin's general type descriptor.

- PLUGIN_STATUS

  The plugin status, one of ACTIVE, INACTIVE, DISABLED, DELETING, or DELETED.

- PLUGIN_TYPE

  The type of plugin, such as STORAGE ENGINE, INFORMATION_SCHEMA, or AUTHENTICATION.

- PLUGIN_TYPE_VERSION

  The version from the plugin's type-specific descriptor.

- PLUGIN_LIBRARY

  The name of the plugin shared library file. This is the name used to refer to the plugin file in statements such as INSTALL PLUGIN and UNINSTALL PLUGIN. This file is located in the directory named by the plugin_dir system variable. If the library name is NULL, the plugin is compiled in and cannot be uninstalled with UNINSTALL PLUGIN.

- PLUGIN_LIBRARY_VERSION

  The plugin API interface version.

- PLUGIN_AUTHOR

The plugin author.

- `PLUGIN_DESCRIPTION`

  A short description of the plugin.

- `PLUGIN_LICENSE`

  How the plugin is licensed (for example, `GPL`).

- `LOAD_OPTION`

  How the plugin was loaded. The value is `OFF`, `ON`, `FORCE`, or `FORCE_PLUS_PERMANENT`. See Section 5.6.1, “Installing and Uninstalling Plugins”.

Notes

- The `PLUGINS` table is a nonstandard `INFORMATION_SCHEMA` table.
- For plugins installed with `INSTALL PLUGIN`, the `PLUGIN_NAME` and `PLUGIN_LIBRARY` values are also registered in the `mysql.plugin` table.
- For information about plugin data structures that form the basis of the information in the `PLUGINS` table, see Section 29.2, “The MySQL Plugin API”.

Plugin information is also available from the `SHOW PLUGINS` statement. See Section 13.7.7.25, “SHOW PLUGINS Syntax”. These statements are equivalent:

```
SELECT
  PLUGIN_NAME, PLUGIN_STATUS, PLUGIN_TYPE,
  PLUGIN_LIBRARY, PLUGIN_LICENSE
FROM INFORMATION_SCHEMA.PLUGINS;

SHOW PLUGINS;
```

25.19 The INFORMATION_SCHEMA PROCESSLIST Table

The `PROCESSLIST` table provides information about which threads are running.

The `PROCESSLIST` table has these columns:

- `ID`

  The connection identifier. This is the same type of value displayed in the `Id` column of the `SHOW PROCESSLIST` statement, the `PROCESSLIST_ID` column of the Performance Schema `threads` table, and returned by the `CONNECTION_ID()` function.

- `USER`

  The MySQL user who issued the statement. A value of `system user` refers to a nonclient thread spawned by the server to handle tasks internally. This could be the I/O or SQL thread used on replication slaves or a delayed-row handler. For `system user`, there is no host specified in the `Host` column. `unauthenticated user` refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet been done. `event_scheduler` refers to the thread that monitors scheduled events (see Section 24.4, “Using the Event Scheduler”).

Note

A `USER` value of `system user` is distinct from the `SYSTEM_USER` privilege. The former designates internal threads. The latter distinguishes the system user and regular user account categories (see Section 6.2.11, “Account Categories”).

- `HOST`

  The host name of the client issuing the statement (except for `system user`, for which there is no host). The host name for TCP/IP connections is reported in *`host_name:client_port`* format to make it easier to determine which client is doing what.

- `DB`

  The default database, if one is selected; otherwise `NULL`.

- `COMMAND`

  The type of command the thread is executing. For descriptions for thread commands, see Section 8.14, “Examining Thread Information”. The value of this column corresponds to the `COM_`*`xxx`* commands of the client/server protocol and `Com_`*`xxx`* status variables. See Section 5.1.10, “Server Status Variables”

- `TIME`

  The time in seconds that the thread has been in its current state. For a slave SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. See Section 17.2.2, “Replication Implementation Details”.

- `STATE`

  An action, event, or state that indicates what the thread is doing. Descriptions for `STATE` values can be found at Section 8.14, “Examining Thread Information”.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

  For the `SHOW PROCESSLIST` statement, the value of `STATE` is `NULL`.

- `INFO`

  The statement the thread is executing, or `NULL` if it is not executing any statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a `CALL` statement executes a stored procedure that is executing a `SELECT` statement, the `INFO` value shows the `SELECT` statement.

Notes

- The `PROCESSLIST` table is a nonstandard `INFORMATION_SCHEMA` table.
- Like the output from the `SHOW PROCESSLIST` statement, the `PROCESSLIST` table shows information only about your own threads, unless you have the `PROCESS` privilege, in which case you will see information about other threads, too. As an anonymous user, you cannot see any rows at all.
- If an SQL statement refers to the `PROCESSLIST` table, MySQL populates the entire table once, when statement execution begins, so there is read consistency during the statement. There is no read consistency for a multi-statement transaction.

Process information is also available from the `mysqladmin processlist` command, the `SHOW PROCESSLIST` statement, and the Performance Schema `threads` table (see Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”, Section 13.7.7.29, “SHOW PROCESSLIST Syntax”, and Section 26.12.18.5, “The threads Table”). In contrast to the `INFORMATION_SCHEMA PROCESSLIST` table and `SHOW PROCESSLIST` statement, which have negative performance consequences because they require a mutex, access to `threads` does not require a mutex and has minimal impact on server performance. The `threads` table also shows information about background threads, which the `PROCESSLIST` table and `SHOW PROCESSLIST` do not. This means that `threads` can be used to monitor activity the other thread information sources cannot.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST

SHOW FULL PROCESSLIST
```

25.20 The INFORMATION_SCHEMA PROFILING Table

The `PROFILING` table provides statement profiling information. Its contents correspond to the information produced by the `SHOW PROFILE` and `SHOW PROFILES` statements (see Section 13.7.7.30, “SHOW PROFILE Syntax”). The table is empty unless the `profiling` session variable is set to 1.

Note

This table is deprecated and will be removed in a future MySQL release. Use the Performance Schema instead; see Section 26.19.1, “Query Profiling Using Performance Schema”.

The `PROFILING` table has these columns:

- `QUERY_ID`

  A numeric statement identifier.

- `SEQ`

  A sequence number indicating the display order for rows with the same `QUERY_ID` value.

- `STATE`

  The profiling state to which the row measurements apply.

- `DURATION`

  How long statement execution remained in the given state, in seconds.

- `CPU_USER`, `CPU_SYSTEM`

  User and system CPU use, in seconds.

- `CONTEXT_VOLUNTARY`, `CONTEXT_INVOLUNTARY`

  How many voluntary and involuntary context switches occurred.

- `BLOCK_OPS_IN`, `BLOCK_OPS_OUT`

  The number of block input and output operations.

- `MESSAGES_SENT`, `MESSAGES_RECEIVED`

  The number of communication messages sent and received.

- `PAGE_FAULTS_MAJOR`, `PAGE_FAULTS_MINOR`

  The number of major and minor page faults.

- `SWAPS`

  How many swaps occurred.

- `SOURCE_FUNCTION`, `SOURCE_FILE`, and `SOURCE_LINE`

  Information indicating where in the source code the profiled state executes.

Notes

- The `PROFILING` table is a nonstandard `INFORMATION_SCHEMA` table.

Profiling information is also available from the `SHOW PROFILE` and `SHOW PROFILES` statements. See Section 13.7.7.30, “SHOW PROFILE Syntax”. For example, the following queries are equivalent:

```
SHOW PROFILE FOR QUERY 2;

SELECT STATE, FORMAT(DURATION, 6) AS DURATION
FROM INFORMATION_SCHEMA.PROFILING
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

25.21 The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table

The `REFERENTIAL_CONSTRAINTS` table provides information about foreign keys.

The `REFERENTIAL_CONSTRAINTS` table has these columns:

- `CONSTRAINT_CATALOG`

  The name of the catalog to which the constraint belongs. This value is always `def`.

- `CONSTRAINT_SCHEMA`

  The name of the schema (database) to which the constraint belongs.

- `CONSTRAINT_NAME`

  The name of the constraint.

- `UNIQUE_CONSTRAINT_CATALOG`

  The name of the catalog containing the unique constraint that the constraint references. This value is always `def`.

- `UNIQUE_CONSTRAINT_SCHEMA`

  The name of the schema containing the unique constraint that the constraint references.

- `UNIQUE_CONSTRAINT_NAME`

The name of the unique constraint that the constraint references.

- `MATCH_OPTION`

  The value of the constraint `MATCH` attribute. The only valid value at this time is `NONE`.

- `UPDATE_RULE`

  The value of the constraint `ON UPDATE` attribute. The possible values are `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT`, `NO ACTION`.

- `DELETE_RULE`

  The value of the constraint `ON DELETE` attribute. The possible values are `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT`, `NO ACTION`.

- `TABLE_NAME`

  The name of the table. This value is the same as in the `TABLE_CONSTRAINTS` table.

- `REFERENCED_TABLE_NAME`

  The name of the table referenced by the constraint.

25.22 The INFORMATION_SCHEMA RESOURCE_GROUPS Table

The `RESOURCE_GROUPS` table provides access to information about resource groups. For general discussion of the resource group capability, see Section 8.12.5, “Resource Groups”.

You can see information only for columns for which you have some privilege.

The `RESOURCE_GROUPS` table has these columns:

- `RESOURCE_GROUP_NAME`

  The name of the resource group.

- `RESOURCE_GROUP_TYPE`

  The resource group type, either `SYSTEM` or `USER`.

- `RESOURCE_GROUP_ENABLED`

  Whether the resource group is enabled (1) or disabled (0);

- `VCPU_IDS`

  The CPU affinity; that is, the set of virtual CPUs that the resource group can use. The value is a list of comma-separated CPU numbers or ranges.

- `THREAD_PRIORITY`

  The priority for threads assigned to the resource group. The priority ranges from -20 (highest priority) to 19 (lowest priority). System resource groups have a priority that ranges from -20 to 0. User resource groups have a priority that ranges from 0 to 19.

25.23 The INFORMATION_SCHEMA ROUTINES Table

The ROUTINES table provides information about stored routines (stored procedures and stored functions). The ROUTINES table does not include built-in SQL functions or user-defined functions (UDFs).

The ROUTINES table has these columns:

- SPECIFIC_NAME

  The name of the routine.

- ROUTINE_CATALOG

  The name of the catalog to which the routine belongs. This value is always def.

- ROUTINE_SCHEMA

  The name of the schema (database) to which the routine belongs.

- ROUTINE_NAME

  The name of the routine.

- ROUTINE_TYPE

  PROCEDURE for stored procedures, FUNCTION for stored functions.

- DATA_TYPE

  If the routine is a stored function, the return value data type. If the routine is a stored procedure, this value is empty.

  The DATA_TYPE value is the type name only with no other information. The DTD_IDENTIFIER value contains the type name and possibly other information such as the precision or length.

- CHARACTER_MAXIMUM_LENGTH

  For stored function string return values, the maximum length in characters. If the routine is a stored procedure, this value is NULL.

- CHARACTER_OCTET_LENGTH

  For stored function string return values, the maximum length in bytes. If the routine is a stored procedure, this value is NULL.

- NUMERIC_PRECISION

  For stored function numeric return values, the numeric precision. If the routine is a stored procedure, this value is NULL.

- NUMERIC_SCALE

  For stored function numeric return values, the numeric scale. If the routine is a stored procedure, this value is NULL.

- DATETIME_PRECISION

  For stored function temporal return values, the fractional seconds precision. If the routine is a stored procedure, this value is NULL.

- CHARACTER_SET_NAME

For stored function character string return values, the character set name. If the routine is a stored procedure, this value is `NULL`.

- `COLLATION_NAME`

  For stored function character string return values, the collation name. If the routine is a stored procedure, this value is `NULL`.

- `DTD_IDENTIFIER`

  If the routine is a stored function, the return value data type. If the routine is a stored procedure, this value is empty.

  The `DATA_TYPE` value is the type name only with no other information. The `DTD_IDENTIFIER` value contains the type name and possibly other information such as the precision or length.

- `ROUTINE_BODY`

  The language used for the routine definition. This value is always `SQL`.

- `ROUTINE_DEFINITION`

  The text of the SQL statement executed by the routine.

- `EXTERNAL_NAME`

  This value is always `NULL`.

- `EXTERNAL_LANGUAGE`

  The language of the stored routine. The value is read from the `external_language` column of the `mysql.routines` data dictionary table.

- `PARAMETER_STYLE`

  This value is always `SQL`.

- `IS_DETERMINISTIC`

  `YES` or `NO`, depending on whether the routine is defined with the `DETERMINISTIC` characteristic.

- `SQL_DATA_ACCESS`

  The data access characteristic for the routine. The value is one of `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, or `MODIFIES SQL DATA`.

- `SQL_PATH`

  This value is always `NULL`.

- `SECURITY_TYPE`

  The routine `SQL SECURITY` characteristic. The value is one of `DEFINER` or `INVOKER`.

- `CREATED`

  The date and time when the routine was created. This is a `TIMESTAMP` value.

- `LAST_ALTERED`

The date and time when the routine was last modified. This is a `TIMESTAMP` value. If the routine has not been modified since its creation, this value is the same as the `CREATED` value.

- `SQL_MODE`

  The SQL mode in effect when the routine was created or altered, and under which the routine executes. For the permitted values, see Section 5.1.11, “Server SQL Modes”.

- `ROUTINE_COMMENT`

  The text of the comment, if the routine has one. If not, this value is empty.

- `DEFINER`

  The account of the user who created the routine, in `'user_name'@'host_name'` format.

- `CHARACTER_SET_CLIENT`

  The session value of the `character_set_client` system variable when the routine was created.

- `COLLATION_CONNECTION`

  The session value of the `collation_connection` system variable when the routine was created.

- `DATABASE_COLLATION`

  The collation of the database with which the routine is associated.

Notes

- Information about stored function return values is also available in the `PARAMETERS` table. The return value row for a stored function can be identified as the row that has an `ORDINAL_POSITION` value of 0.

25.24 The INFORMATION_SCHEMA SCHEMATA Table

A schema is a database, so the `SCHEMATA` table provides information about databases.

The `SCHEMATA` table has these columns:

- `CATALOG_NAME`

  The name of the catalog to which the schema belongs. This value is always `def`.

- `SCHEMA_NAME`

  The name of the schema.

- `DEFAULT_CHARACTER_SET_NAME`

  The schema default character set.

- `DEFAULT_COLLATION_NAME`

  The schema default collation.

- `SQL_PATH`

  This value is always `NULL`.

- `DEFAULT_ENCRYPTION`

  The schema default encryption. This column was added in MySQL 8.0.16.

Schema names are also available from the `SHOW DATABASES` statement. See Section 13.7.7.14, “SHOW DATABASES Syntax”. The following statements are equivalent:

```
SELECT SCHEMA_NAME AS `Database`
  FROM INFORMATION_SCHEMA.SCHEMATA
  [WHERE SCHEMA_NAME LIKE 'wild']

SHOW DATABASES
  [LIKE 'wild']
```

You see only those databases for which you have some kind of privilege, unless you have the global `SHOW DATABASES` privilege.

Caution

Because any static global privilege is considered a privilege for all databases, any static global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `SCHEMATA` table of `INFORMATION_SCHEMA`, except databases that have been restricted at the database level by partial revokes.

25.25 The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table

The `SCHEMA_PRIVILEGES` table provides information about schema (database) privileges. It takes its values from the `mysql.db` system table.

The `SCHEMA_PRIVILEGES` table has these columns:

- `GRANTEE`

  The name of the account to which the privilege is granted, in *`'user_name'@'host_name'`* format.

- `TABLE_CATALOG`

  The name of the catalog to which the schema belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema.

- `PRIVILEGE_TYPE`

  The privilege granted. The value can be any privilege that can be granted at the schema level; see Section 13.7.1.6, “GRANT Syntax”. Each row lists a single privilege, so there is one row per schema privilege held by the grantee.

- `IS_GRANTABLE`

  `YES` if the user has the `GRANT OPTION` privilege, `NO` otherwise. The output does not list `GRANT OPTION` as a separate row with `PRIVILEGE_TYPE='GRANT OPTION'`.

Notes

- The `SCHEMA_PRIVILEGES` table is a nonstandard `INFORMATION_SCHEMA` table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.SCHEMA_PRIVILEGES

SHOW GRANTS ...
```

25.26 The INFORMATION_SCHEMA STATISTICS Table

The `STATISTICS` table provides information about table indexes.

Columns in `STATISTICS` that represent table statistics hold cached values. The `information_schema_stats_expiry` system variable defines the period of time before cached table statistics expire. The default is 86400 seconds (24 hours). If there are no cached statistics or statistics have expired, statistics are retrieved from storage engines when querying table statistics columns. To update cached values at any time for a given table, use `ANALYZE TABLE`. To always retrieve the latest statistics directly from storage engines, set `information_schema_stats_expiry=0`. For more information, see Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”.

Note

If the `innodb_read_only` system variable is enabled, `ANALYZE TABLE` may fail because it cannot update statistics tables in the data dictionary, which use `InnoDB`. For `ANALYZE TABLE` operations that update the key distribution, failure may occur even if the operation updates the table itself (for example, if it is a `MyISAM` table). To obtain the updated distribution statistics, set `information_schema_stats_expiry=0`.

The `STATISTICS` table has these columns:

- `TABLE_CATALOG`

  The name of the catalog to which the table containing the index belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table containing the index belongs.

- `TABLE_NAME`

  The name of the table containing the index.

- `NON_UNIQUE`

  0 if the index cannot contain duplicates, 1 if it can.

- `INDEX_SCHEMA`

  The name of the schema (database) to which the index belongs.

- `INDEX_NAME`

  The name of the index. If the index is the primary key, the name is always `PRIMARY`.

- `SEQ_IN_INDEX`

  The column sequence number in the index, starting with 1.

- `COLUMN_NAME`

  The column name. See also the description for the `EXPRESSION` column.

- `COLLATION`

  How the column is sorted in the index. This can have values `A` (ascending), `D` (descending), or `NULL` (not sorted).

- `CARDINALITY`

  An estimate of the number of unique values in the index. To update this number, run `ANALYZE TABLE` or (for `MyISAM` tables) `myisamchk -a`.

  `CARDINALITY` is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

- `SUB_PART`

  The index prefix. That is, the number of indexed characters if the column is only partly indexed, `NULL` if the entire column is indexed.

  Note

  Prefix *limits* are measured in bytes. However, prefix *lengths* for index specifications in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements are interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

  For additional information about index prefixes, see Section 8.3.5, “Column Indexes”, and Section 13.1.15, “CREATE INDEX Syntax”.

- `PACKED`

  Indicates how the key is packed. `NULL` if it is not.

- `NULLABLE`

  Contains `YES` if the column may contain `NULL` values and `''` if not.

- `INDEX_TYPE`

  The index method used (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`).

- `COMMENT`

  Information about the index not described in its own column, such as `disabled` if the index is disabled.

- `INDEX_COMMENT`

  Any comment provided for the index with a `COMMENT` attribute when the index was created.

- `IS_VISIBLE`

  Whether the index is visible to the optimizer. See Section 8.3.12, “Invisible Indexes”.

- `EXPRESSION`

  MySQL 8.0.13 and higher supports functional key parts (see Functional Key Parts), which affects both the `COLUMN_NAME` and `EXPRESSION` columns:

- For a nonfunctional key part, `COLUMN_NAME` indicates the column indexed by the key part and `EXPRESSION` is `NULL`.
- For a functional key part, `COLUMN_NAME` column is `NULL` and `EXPRESSION` indicates the expression for the key part.

Notes

- There is no standard `INFORMATION_SCHEMA` table for indexes. The MySQL column list is similar to what SQL Server 2000 returns for `sp_statistics`, except that `QUALIFIER` and `OWNER` are replaced with `CATALOG` and `SCHEMA`, respectively.

Information about table indexes is also available from the `SHOW INDEX` statement. See Section 13.7.7.22, “SHOW INDEX Syntax”. The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  AND table_schema = 'db_name'

SHOW INDEX
  FROM tbl_name
  FROM db_name
```

25.27 The INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS Table

The `ST_GEOMETRY_COLUMNS` table provides information about table columns that store spatial data. This table is based on the SQL/MM (ISO/IEC 13249-3) standard, with extensions as noted. MySQL implements `ST_GEOMETRY_COLUMNS` as a view on the `INFORMATION_SCHEMA COLUMNS` table.

The `ST_GEOMETRY_COLUMNS` table has these columns:

- `TABLE_CATALOG`

  The name of the catalog to which the table containing the column belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table containing the column belongs.

- `TABLE_NAME`

  The name of the table containing the column.

- `COLUMN_NAME`

  The name of the column.

- `SRS_NAME`

  The spatial reference system (SRS) name.

- `SRS_ID`

  The spatial reference system ID (SRID).

- `GEOMETRY_TYPE_NAME`

The column data type. Permitted values are: `geometry`, `point`, `linestring`, `polygon`, `multipoint`, `multilinestring`, `multipolygon`, `geometrycollection`. This column is a MySQL extension to the standard.

25.28 The INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS Table

The `ST_SPATIAL_REFERENCE_SYSTEMS` table provides information about available spatial reference systems for spatial data. This table is based on the SQL/MM (ISO/IEC 13249-3) standard.

Entries in the `ST_SPATIAL_REFERENCE_SYSTEMS` table are based on the European Petroleum Survey Group (EPSG) data set, except for SRID 0, which corresponds to a special SRS used in MySQL that represents an infinite flat Cartesian plane with no units assigned to its axes. For additional information about SRSs, see Section 11.5.5, “Spatial Reference System Support”.

The `ST_SPATIAL_REFERENCE_SYSTEMS` table has these columns:

- `SRS_NAME`

  The spatial reference system name. This value is unique.

- `SRS_ID`

  The spatial reference system numeric ID. This value is unique.

  `SRS_ID` values represent the same kind of values passed as the SRID argument to spatial functions. SRID 0 (the unitless Cartesian plane) is special. It is always a legal spatial reference system ID and can be used in any computations on spatial data that depend on SRID values.

- `ORGANIZATION`

  The name of the organization that defined the coordinate system on which the spatial reference system is based.

- `ORGANIZATION_COORDSYS_ID`

  The numeric ID given to the spatial reference system by the organization that defined it.

- `DEFINITION`

  The spatial reference system definition. `DEFINITION` values are WKT values, represented as specified in the Open Geospatial Consortium document OGC 12-063r5.

  SRS definition parsing occurs on demand when definitions are needed by GIS functions. Parsed definitions are cached in the data dictionary cache so that parsing overhead is not incurred for every statement that needs SRS information.

- `DESCRIPTION`

  The spatial reference system description.

Notes

- The `SRS_NAME`, `ORGANIZATION`, `ORGANIZATION_COORDSYS_ID`, and `DESCRIPTION` columns contain information that may be of interest to users, but they are not used by MySQL.

Example

```
mysql> SELECT * FROM ST_SPATIAL_REFERENCE_SYSTEMS
       WHERE SRS_ID = 4326\G
*************************** 1. row ***************************
               SRS_NAME: WGS 84
                 SRS_ID: 4326
           ORGANIZATION: EPSG
ORGANIZATION_COORDSYS_ID: 4326
             DEFINITION: GEOGCS["WGS 84",DATUM["World Geodetic System 1984",
                         SPHEROID["WGS 84",6378137,298.257223563,
                         AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],
                         PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
                         UNIT["degree",0.01745329251994328,
                         AUTHORITY["EPSG","9122"]],
                         AXIS["Lat",NORTH],AXIS["Long",EAST],
                         AUTHORITY["EPSG","4326"]]
            DESCRIPTION:
```

This entry describes the SRS used for GPS systems. It has a name (`SRS_NAME`) of WGS 84 and an ID (`SRS_ID`) of 4326, which is the ID used by the European Petroleum Survey Group (EPSG).

The `DEFINITION` values for projected and geographic SRSs begin with `PROJCS` and `GEOGCS`, respectively. The definition for SRID 0 is special and has an empty `DEFINITION` value. The following query determines how many entries in the `ST_SPATIAL_REFERENCE_SYSTEMS` table correspond to projected, geographic, and other SRSs, based on `DEFINITION` values:

```
mysql> SELECT
         COUNT(*),
         CASE LEFT(DEFINITION, 6)
           WHEN 'PROJCS' THEN 'Projected'
           WHEN 'GEOGCS' THEN 'Geographic'
           ELSE 'Other'
         END AS SRS_TYPE
       FROM ST_SPATIAL_REFERENCE_SYSTEMS
       GROUP BY SRS_TYPE;
+----------+------------+
| COUNT(*) | SRS_TYPE   |
+----------+------------+
|        1 | Other      |
|     4668 | Projected  |
|      483 | Geographic |
+----------+------------+
```

To enable manipulation of SRS entries stored in the data dictionary, MySQL provides these SQL statements:

- `CREATE SPATIAL REFERENCE SYSTEM`: See Section 13.1.19, “CREATE SPATIAL REFERENCE SYSTEM Syntax”. The description for this statement includes additional information about SRS components.
- `DROP SPATIAL REFERENCE SYSTEM`: See Section 13.1.31, “DROP SPATIAL REFERENCE SYSTEM Syntax”.

25.29 The INFORMATION_SCHEMA ST_UNITS_OF_MEASURE Table

The `ST_UNITS_OF_MEASURE` table (available as of MySQL 8.0.14) provides information about acceptable units for the `ST_Distance()` function.

The `ST_UNITS_OF_MEASURE` table has these columns:

- `UNIT_NAME`

  The name of the unit.

- `UNIT_TYPE`

  The unit type (for example, `LINEAR`).

- `CONVERSION_FACTOR`

  A conversion factor used for internal calculations.

- `DESCRIPTION`

  A description of the unit.

25.30 The INFORMATION_SCHEMA TABLES Table

The `TABLES` table provides information about tables in databases.

Columns in `TABLES` that represent table statistics hold cached values. The `information_schema_stats_expiry` system variable defines the period of time before cached table statistics expire. The default is 86400 seconds (24 hours). If there are no cached statistics or statistics have expired, statistics are retrieved from storage engines when querying table statistics columns. To update cached values at any time for a given table, use `ANALYZE TABLE`. To always retrieve the latest statistics directly from storage engines, set `information_schema_stats_expiry` to `0`. For more information, see Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”.

Note

If the `innodb_read_only` system variable is enabled, `ANALYZE TABLE` may fail because it cannot update statistics tables in the data dictionary, which use `InnoDB`. For `ANALYZE TABLE` operations that update the key distribution, failure may occur even if the operation updates the table itself (for example, if it is a `MyISAM` table). To obtain the updated distribution statistics, set `information_schema_stats_expiry=0`.

The `TABLES` table has these columns:

- `TABLE_CATALOG`

  The name of the catalog to which the table belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table belongs.

- `TABLE_NAME`

  The name of the table.

- `TABLE_TYPE`

  `BASE TABLE` for a table, `VIEW` for a view, or `SYSTEM VIEW` for an `INFORMATION_SCHEMA` table.

  The `TABLES` table does not list `TEMPORARY` tables.

- `ENGINE`

The storage engine for the table. See Chapter 15, *The InnoDB Storage Engine*, and Chapter 16, *Alternative Storage Engines*.

For partitioned tables, `ENGINE` shows the name of the storage engine used by all partitions.

- `VERSION`

  This column is unused. With the removal of `.frm` files in MySQL 8.0, this column now reports a hardcoded value of `10`, which is the last `.frm` file version used in MySQL 5.7.

- `ROW_FORMAT`

  The row-storage format (`Fixed`, `Dynamic`, `Compressed`, `Redundant`, `Compact`). For `MyISAM` tables, `Dynamic` corresponds to what `myisamchk -dvv` reports as `Packed`.

- `TABLE_ROWS`

  The number of rows. Some storage engines, such as `MyISAM`, store the exact count. For other storage engines, such as `InnoDB`, this value is an approximation, and may vary from the actual value by as much as 40% to 50%. In such cases, use `SELECT COUNT(*)` to obtain an accurate count.

  `TABLE_ROWS` is `NULL` for `INFORMATION_SCHEMA` tables.

  For `InnoDB` tables, the row count is only a rough estimate used in SQL optimization. (This is also true if the `InnoDB` table is partitioned.)

- `AVG_ROW_LENGTH`

  The average row length.

- `DATA_LENGTH`

  For `MyISAM`, `DATA_LENGTH` is the length of the data file, in bytes.

  For `InnoDB`, `DATA_LENGTH` is the approximate amount of memory allocated for the clustered index, in bytes. Specifically, it is the clustered index size, in pages, multiplied by the `InnoDB` page size.

  Refer to the notes at the end of this section for information regarding other storage engines.

- `MAX_DATA_LENGTH`

  For `MyISAM`, `MAX_DATA_LENGTH` is maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

  Unused for `InnoDB`.

  Refer to the notes at the end of this section for information regarding other storage engines.

- `INDEX_LENGTH`

  For `MyISAM`, `INDEX_LENGTH` is the length of the index file, in bytes.

  For `InnoDB`, `INDEX_LENGTH` is the approximate amount of memory allocated for non-clustered indexes, in bytes. Specifically, it is the sum of non-clustered index sizes, in pages, multiplied by the `InnoDB` page size.

  Refer to the notes at the end of this section for information regarding other storage engines.

- DATA_FREE

  The number of allocated but unused bytes.

  InnoDB tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table. Free space means the number of bytes in completely free extents minus a safety margin. Even if free space displays as 0, it may be possible to insert rows as long as new extents need not be allocated.

  For NDB Cluster, DATA_FREE shows the space allocated on disk for, but not used by, a Disk Data table or fragment on disk. (In-memory data resource usage is reported by the DATA_LENGTH column.)

  For partitioned tables, this value is only an estimate and may not be absolutely correct. A more accurate method of obtaining this information in such cases is to query the INFORMATION_SCHEMA PARTITIONS table, as shown in this example:

  ```
  SELECT SUM(DATA_FREE)
      FROM  INFORMATION_SCHEMA.PARTITIONS
      WHERE TABLE_SCHEMA = 'mydb'
      AND   TABLE_NAME   = 'mytable';
  ```

  For more information, see Section 25.17, “The INFORMATION_SCHEMA PARTITIONS Table”.

- AUTO_INCREMENT

  The next AUTO_INCREMENT value.

- CREATE_TIME

  When the table was created.

- UPDATE_TIME

  When the data file was last updated. For some storage engines, this value is NULL. For example, InnoDB stores multiple tables in its system tablespace and the data file timestamp does not apply. Even with file-per-table mode with each InnoDB table in a separate .ibd file, change buffering can delay the write to the data file, so the file modification time is different from the time of the last insert, update, or delete. For MyISAM, the data file timestamp is used; however, on Windows the timestamp is not updated by updates, so the value is inaccurate.

  UPDATE_TIME displays a timestamp value for the last UPDATE, INSERT, or DELETE performed on InnoDB tables that are not partitioned. For MVCC, the timestamp value reflects the COMMIT time, which is considered the last update time. Timestamps are not persisted when the server is restarted or when the table is evicted from the InnoDB data dictionary cache.

- CHECK_TIME

  When the table was last checked. Not all storage engines update this time, in which case, the value is always NULL.

  For partitioned InnoDB tables, CHECK_TIME is always NULL.

- TABLE_COLLATION

  The table default collation. The output does not explicitly list the table default character set, but the collation name begins with the character set name.

- `CHECKSUM`

  The live checksum value, if any.

- `CREATE_OPTIONS`

  Extra options used with `CREATE TABLE`. The original options specified when `CREATE TABLE` was executed are retained. The information reported may differ from current table settings and options.

  For `InnoDB` tables, the actual `ROW_FORMAT` and `KEY_BLOCK_SIZE` options are shown. Prior to MySQL 8.0, `CREATE_OPTIONS` shows the originally supplied `ROW_FORMAT` and `KEY_BLOCK_SIZE`. For more information, see Section 13.1.20, “CREATE TABLE Syntax”.

  `CREATE_OPTIONS` shows `partitioned` for a partitioned table. Prior to MySQL 8.0.16, it also shows the `ENCRYPTION` clause specified for tables created in file-per-table tablespaces. As of MySQL 8.0.16, it only shows the encryption clause for file-per-table tablespaces if the table is encrypted or if the specified encryption differs from the schema encryption. The encryption clause is not shown for tables created in general tablespaces. To identify encrypted file-per-table and general tablespaces, query the `INNODB_TABLESPACES ENCRYPTION` column.

- `TABLE_COMMENT`

  The comment used when creating the table (or information as to why MySQL could not access the table information).

Notes

- For `NDB` tables, the output of this statement shows appropriate values for the `AVG_ROW_LENGTH` and `DATA_LENGTH` columns, with the exception that `BLOB` columns are not taken into account.
- For `NDB` tables, `DATA_LENGTH` includes data stored in main memory only; the `MAX_DATA_LENGTH` and `DATA_FREE` columns apply to Disk Data.
- For NDB Cluster Disk Data tables, `MAX_DATA_LENGTH` shows the space allocated for the disk part of a Disk Data table or fragment. (In-memory data resource usage is reported by the `DATA_LENGTH` column.)
- For `MEMORY` tables, the `DATA_LENGTH`, `MAX_DATA_LENGTH`, and `INDEX_LENGTH` values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.
- For views, most `TABLES` columns are 0 or `NULL` except that `TABLE_NAME` indicates the view name, `CREATE_TIME` indicates the creation time, and `TABLE_COMMENT` says `VIEW`.

Table information is also available from the `SHOW TABLE STATUS` and `SHOW TABLES` statements. See Section 13.7.7.36, “SHOW TABLE STATUS Syntax”, and Section 13.7.7.37, “SHOW TABLES Syntax”. The following statements are equivalent:

```
SELECT
    TABLE_NAME, ENGINE, VERSION, ROW_FORMAT, TABLE_ROWS, AVG_ROW_LENGTH,
    DATA_LENGTH, MAX_DATA_LENGTH, INDEX_LENGTH, DATA_FREE, AUTO_INCREMENT,
    CREATE_TIME, UPDATE_TIME, CHECK_TIME, TABLE_COLLATION, CHECKSUM,
    CREATE_OPTIONS, TABLE_COMMENT
  FROM INFORMATION_SCHEMA.TABLES
  WHERE table_schema = 'db_name'
  [AND table_name LIKE 'wild']

SHOW TABLE STATUS
  FROM db_name
```

```
  [LIKE 'wild']
```

The following statements are equivalent:

```
SELECT
  TABLE_NAME, TABLE_TYPE
  FROM INFORMATION_SCHEMA.TABLES
  WHERE table_schema = 'db_name'
  [AND table_name LIKE 'wild']

SHOW FULL TABLES
  FROM db_name
  [LIKE 'wild']
```

25.31 The INFORMATION_SCHEMA TABLESPACES Table

The `TABLESPACES` table provides information about active MySQL Cluster tablespaces.

The `TABLESPACES` table has these columns:

- `TABLESPACE_NAME`

  The name of the tablespace.

- `ENGINE`

  The name of the storage engine that uses the tablespace.

- `TABLESPACE_TYPE`

  The tablespace type.

- `LOGFILE_GROUP_NAME`

  The name of the logfile group assigned to the tablespace.

- `EXTENT_SIZE`

  The size in bytes of the extents used by files that belong to the tablespace.

- `AUTOEXTEND_SIZE`

  Unused.

- `MAXIMUM_SIZE`

  Unused.

- `NODEGROUP_ID`

  Unused.

- `TABLESPACE_COMMENT`

  Unused.

Notes

- The `TABLESPACES` table is a nonstandard `INFORMATION_SCHEMA` table.

- The TABLESPACES table does not provide information about InnoDB tablespaces. For InnoDB tablespace metadata, see the INFORMATION_SCHEMA INNODB_TABLESPACES and INNODB_DATAFILES tables. The FILES table also provides metadata for InnoDB tablespaces.

25.32 The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table

The TABLE_CONSTRAINTS table describes which tables have constraints.

The TABLE_CONSTRAINTS table has these columns:

- CONSTRAINT_CATALOG

  The name of the catalog to which the constraint belongs. This value is always def.

- CONSTRAINT_SCHEMA

  The name of the schema (database) to which the constraint belongs.

- TABLE_SCHEMA

  The name of the schema (database) to which the table belongs.

- TABLE_NAME

  The name of the table.

- The CONSTRAINT_TYPE

  The type of constraint. The value can be UNIQUE, PRIMARY KEY, FOREIGN KEY, or (as of MySQL 8.0.16) CHECK. This is a CHAR (not ENUM) column.

  The UNIQUE and PRIMARY KEY information is about the same as what you get from the Key_name column in the output from SHOW INDEX when the Non_unique column is 0.

- ENFORCED

  For CHECK constraints, the value is YES or NO to indicate whether the constraint is enforced. For other constraints, the value is always YES.

  This column was added in MySQL 8.0.16.

25.33 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table

The TABLE_PRIVILEGES table provides information about table privileges. It takes its values from the mysql.tables_priv system table.

The TABLE_PRIVILEGES table has these columns:

- GRANTEE

  The name of the account to which the privilege is granted, in *'user_name'@'host_name'* format.

- TABLE_CATALOG

  The name of the catalog to which the table belongs. This value is always def.

- TABLE_SCHEMA

The name of the schema (database) to which the table belongs.

- `TABLE_NAME`

  The name of the table.

- `PRIVILEGE_TYPE`

  The privilege granted. The value can be any privilege that can be granted at the table level; see Section 13.7.1.6, “GRANT Syntax”. Each row lists a single privilege, so there is one row per table privilege held by the grantee.

- `IS_GRANTABLE`

  `YES` if the user has the `GRANT OPTION` privilege, `NO` otherwise. The output does not list `GRANT OPTION` as a separate row with `PRIVILEGE_TYPE='GRANT OPTION'`.

Notes

- The `TABLE_PRIVILEGES` table is a nonstandard `INFORMATION_SCHEMA` table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES

SHOW GRANTS ...
```

25.34 The INFORMATION_SCHEMA TRIGGERS Table

The `TRIGGERS` table provides information about triggers. To see information about a table's triggers, you must have the `TRIGGER` privilege for the table.

The `TRIGGERS` table has these columns:

- `TRIGGER_CATALOG`

  The name of the catalog to which the trigger belongs. This value is always `def`.

- `TRIGGER_SCHEMA`

  The name of the schema (database) to which the trigger belongs.

- `TRIGGER_NAME`

  The name of the trigger.

- `EVENT_MANIPULATION`

  The trigger event. This is the type of operation on the associated table for which the trigger activates. The value is `INSERT` (a row was inserted), `DELETE` (a row was deleted), or `UPDATE` (a row was modified).

- `EVENT_OBJECT_CATALOG`, `EVENT_OBJECT_SCHEMA`, and `EVENT_OBJECT_TABLE`

  As noted in Section 24.3, “Using Triggers”, every trigger is associated with exactly one table. These columns indicate the catalog and schema (database) in which this table occurs, and the table name, respectively. The `EVENT_OBJECT_CATALOG` value is always `def`.

- `ACTION_ORDER`

  The ordinal position of the trigger's action within the list of triggers on the same table with the same `EVENT_MANIPULATION` and `ACTION_TIMING` values.

- `ACTION_CONDITION`

  This value is always `NULL`.

- `ACTION_STATEMENT`

  The trigger body; that is, the statement executed when the trigger activates. This text uses UTF-8 encoding.

- `ACTION_ORIENTATION`

  This value is always `ROW`.

- `ACTION_TIMING`

  Whether the trigger activates before or after the triggering event. The value is `BEFORE` or `AFTER`.

- `ACTION_REFERENCE_OLD_TABLE`

  This value is always `NULL`.

- `ACTION_REFERENCE_NEW_TABLE`

  This value is always `NULL`.

- `ACTION_REFERENCE_OLD_ROW` and `ACTION_REFERENCE_NEW_ROW`

  The old and new column identifiers, respectively. The `ACTION_REFERENCE_OLD_ROW` value is always `OLD` and the `ACTION_REFERENCE_NEW_ROW` value is always `NEW`.

- `CREATED`

  The date and time when the trigger was created. This is a `TIMESTAMP(2)` value (with a fractional part in hundredths of seconds) for triggers.

- `SQL_MODE`

  The SQL mode in effect when the trigger was created, and under which the trigger executes. For the permitted values, see Section 5.1.11, “Server SQL Modes”.

- `DEFINER`

  The account of the user who created the trigger, in `'user_name'@'host_name'` format.

- `CHARACTER_SET_CLIENT`

  The session value of the `character_set_client` system variable when the trigger was created.

- `COLLATION_CONNECTION`

  The session value of the `collation_connection` system variable when the trigger was created.

- `DATABASE_COLLATION`

  The collation of the database with which the trigger is associated.

Example

The following example uses the `ins_sum` trigger defined in Section 24.3, “Using Triggers”:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS
       WHERE TRIGGER_SCHEMA='test' AND TRIGGER_NAME='ins_sum'\G
*************************** 1. row ***************************
           TRIGGER_CATALOG: def
            TRIGGER_SCHEMA: test
              TRIGGER_NAME: ins_sum
        EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: def
       EVENT_OBJECT_SCHEMA: test
        EVENT_OBJECT_TABLE: account
              ACTION_ORDER: 1
          ACTION_CONDITION: NULL
          ACTION_STATEMENT: SET @sum = @sum + NEW.amount
        ACTION_ORIENTATION: ROW
             ACTION_TIMING: BEFORE
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
  ACTION_REFERENCE_OLD_ROW: OLD
  ACTION_REFERENCE_NEW_ROW: NEW
                   CREATED: 2018-08-08 10:10:12.61
                 SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
                           NO_ZERO_IN_DATE,NO_ZERO_DATE,
                           ERROR_FOR_DIVISION_BY_ZERO,
                           NO_ENGINE_SUBSTITUTION
                  DEFINER: me@localhost
      CHARACTER_SET_CLIENT: utf8mb4
      COLLATION_CONNECTION: utf8mb4_0900_ai_ci
        DATABASE_COLLATION: utf8mb4_0900_ai_ci
```

Trigger information is also available from the `SHOW TRIGGERS` statement. See Section 13.7.7.38, “SHOW TRIGGERS Syntax”.

25.35 The INFORMATION_SCHEMA USER_PRIVILEGES Table

The `USER_PRIVILEGES` table provides information about global privileges. It takes its values from the `mysql.user` system table.

The `USER_PRIVILEGES` table has these columns:

- `GRANTEE`

  The name of the account to which the privilege is granted, in *`'user_name'@'host_name'`* format.

- `TABLE_CATALOG`

  The name of the catalog. This value is always `def`.

- `PRIVILEGE_TYPE`

  The privilege granted. The value can be any privilege that can be granted at the global level; see Section 13.7.1.6, “GRANT Syntax”. Each row lists a single privilege, so there is one row per global privilege held by the grantee.

- `IS_GRANTABLE`

  `YES` if the user has the `GRANT OPTION` privilege, `NO` otherwise. The output does not list `GRANT OPTION` as a separate row with `PRIVILEGE_TYPE='GRANT OPTION'`.

Notes

- The `USER_PRIVILEGES` table is a nonstandard `INFORMATION_SCHEMA` table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.USER_PRIVILEGES

SHOW GRANTS ...
```

25.36 The INFORMATION_SCHEMA VIEWS Table

The `VIEWS` table provides information about views in databases. You must have the `SHOW VIEW` privilege to access this table.

The `VIEWS` table has these columns:

- `TABLE_CATALOG`

  The name of the catalog to which the view belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the view belongs.

- `TABLE_NAME`

  The name of the view.

- `VIEW_DEFINITION`

  The `SELECT` statement that provides the definition of the view. This column has most of what you see in the `Create Table` column that `SHOW CREATE VIEW` produces. Skip the words before `SELECT` and skip the words `WITH CHECK OPTION`. Suppose that the original statement was:

  ```
  CREATE VIEW v AS
    SELECT s2,s1 FROM t
    WHERE s1 > 5
    ORDER BY s1
    WITH CHECK OPTION;
  ```

  Then the view definition looks like this:

  ```
  SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
  ```

- `CHECK_OPTION`

  The value of the `CHECK_OPTION` attribute. The value is one of `NONE`, `CASCADE`, or `LOCAL`.

- `IS_UPDATABLE`

  MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE` (and similar operations) are legal for the view. Otherwise, the flag is set to `NO` (false). The `IS_UPDATABLE` column in the `VIEWS` table displays the status of this flag. It means that the server always knows whether a view is updatable.

If a view is not updatable, statements such `UPDATE`, `DELETE`, and `INSERT` are illegal and are rejected. (Even if a view is updatable, it might not be possible to insert into it; for details, refer to Section 24.5.3, “Updatable and Insertable Views”.)

- `DEFINER`

  The account of the user who created the view, in `'user_name'@'host_name'` format.

- `SECURITY_TYPE`

  The view `SQL SECURITY` characteristic. The value is one of `DEFINER` or `INVOKER`.

- `CHARACTER_SET_CLIENT`

  The session value of the `character_set_client` system variable when the view was created.

- `COLLATION_CONNECTION`

  The session value of the `collation_connection` system variable when the view was created.

Notes

MySQL permits different `sql_mode` settings to tell the server the type of SQL syntax to support. For example, you might use the `ANSI` SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (`||`), in your queries. If you then create a view that concatenates items, you might worry that changing the `sql_mode` setting to a value different from `ANSI` could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a `CONCAT()` function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA.VIEWS
       WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v';
+----------------------------------+
| VIEW_DEFINITION                  |
+----------------------------------+
| select concat('a','b') AS `col1` |
+----------------------------------+
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of `sql_mode` do not affect the results from the view. However, an additional consequence is that comments prior to `SELECT` are stripped from the definition by the server.

25.37 The INFORMATION_SCHEMA VIEW_ROUTINE_USAGE Table

The `VIEW_ROUTINE_USAGE` table (available as of MySQL 8.0.13) provides access to information about stored functions used in view definitions. The table does not list information about built-in SQL functions or user-defined functions (UDFs) used in the definitions.

You can see information only for views for which you have some privilege, and only for functions for which you have some privilege.

The VIEW_ROUTINE_USAGE table has these columns:

- TABLE_CATALOG

  The name of the catalog to which the view belongs. This value is always def.

- TABLE_SCHEMA

  The name of the schema (database) to which the view belongs.

- TABLE_NAME

  The name of the view.

- SPECIFIC_CATALOG

  The name of the catalog to which the function used in the view definition belongs. This value is always def.

- SPECIFIC_SCHEMA

  The name of the schema (database) to which the function used in the view definition belongs.

- SPECIFIC_NAME

  The name of the function used in the view definition.

25.38 The INFORMATION_SCHEMA VIEW_TABLE_USAGE Table

The VIEW_TABLE_USAGE table (available as of MySQL 8.0.13) provides access to information about tables and views used in view definitions.

You can see information only for views for which you have some privilege, and only for tables for which you have some privilege.

The VIEW_TABLE_USAGE table has these columns:

- VIEW_CATALOG

  The name of the catalog to which the view belongs. This value is always def.

- VIEW_SCHEMA

  The name of the schema (database) to which the view belongs.

- VIEW_NAME

  The name of the view.

- TABLE_CATALOG

  The name of the catalog to which the table or view used in the view definition belongs. This value is always def.

- TABLE_SCHEMA

  The name of the schema (database) to which the table or view used in the view definition belongs.

- TABLE_NAME

The name of the table or view used in the view definition.

25.39 INFORMATION_SCHEMA InnoDB Tables

This section provides table definitions for `InnoDB` `INFORMATION_SCHEMA` tables. For related information and examples, see Section 15.14, “InnoDB INFORMATION_SCHEMA Tables”.

`InnoDB` `INFORMATION_SCHEMA` tables can be used to monitor ongoing `InnoDB` activity, to detect inefficiencies before they turn into issues, or to troubleshoot performance and capacity issues. As your database becomes bigger and busier, running up against the limits of your hardware capacity, you monitor and tune these aspects to keep the database running smoothly.

25.39.1 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table

The `INNODB_BUFFER_PAGE` table provides information about each page in the `InnoDB` buffer pool.

For related usage information and examples, see Section 15.14.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”.

Warning

Querying the `INNODB_BUFFER_PAGE` table can affect performance. Do not query this table on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

The `INNODB_BUFFER_PAGE` table has these columns:

- `POOL_ID`

  The buffer pool ID. This is an identifier to distinguish between multiple buffer pool instances.

- `BLOCK_ID`

  The buffer pool block ID.

- `SPACE`

  The tablespace ID; the same value as `INNODB_TABLES.SPACE`.

- `PAGE_NUMBER`

  The page number.

- `PAGE_TYPE`

  The page type. The following table shows the permitted values.

  Table 25.1 INNODB_BUFFER_PAGE.PAGE_TYPE Values

Page Type	Description
`ALLOCATED`	Freshly allocated page
`BLOB`	Uncompressed BLOB page
`COMPRESSED_BLOB2`	Subsequent comp BLOB page

Page Type	Description
COMPRESSED_BLOB	First compressed BLOB page
ENCRYPTED_RTREE	Encrypted R-tree
EXTENT_DESCRIPTOR	Extent descriptor page
FILE_SPACE_HEADER	File space header
FIL_PAGE_TYPE_UNUSED	Unused
IBUF_BITMAP	Insert buffer bitmap
IBUF_FREE_LIST	Insert buffer free list
IBUF_INDEX	Insert buffer index
INDEX	B-tree node
INODE	Index node
LOB_DATA	Uncompressed LOB data
LOB_FIRST	First page of uncompressed LOB
LOB_INDEX	Uncompressed LOB index
PAGE_IO_COMPRESSED	Compressed page
PAGE_IO_COMPRESSED_ENCRYPTED	Compressed and encrypted page
PAGE_IO_ENCRYPTED	Encrypted page
RSEG_ARRAY	Rollback segment array
RTREE_INDEX	R-tree index
SDI_BLOB	Uncompressed SDI BLOB
SDI_COMPRESSED_BLOB	Compressed SDI BLOB
SDI_INDEX	SDI index
SYSTEM	System page
TRX_SYSTEM	Transaction system data
UNDO_LOG	Undo log page
UNKNOWN	Unknown
ZLOB_DATA	Compressed LOB data
ZLOB_FIRST	First page of compressed LOB
ZLOB_FRAG	Compressed LOB fragment
ZLOB_FRAG_ENTRY	Compressed LOB fragment index
ZLOB_INDEX	Compressed LOB index

- FLUSH_TYPE

  The flush type.

- FIX_COUNT

The number of threads using this block within the buffer pool. When zero, the block is eligible to be evicted.

- `IS_HASHED`

  Whether a hash index has been built on this page.

- `NEWEST_MODIFICATION`

  The Log Sequence Number of the youngest modification.

- `OLDEST_MODIFICATION`

  The Log Sequence Number of the oldest modification.

- `ACCESS_TIME`

  An abstract number used to judge the first access time of the page.

- `TABLE_NAME`

  The name of the table the page belongs to. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `INDEX_NAME`

  The name of the index the page belongs to. This can be the name of a clustered index or a secondary index. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `NUMBER_RECORDS`

  The number of records within the page.

- `DATA_SIZE`

  The sum of the sizes of the records. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `COMPRESSED_SIZE`

  The compressed page size. `NULL` for pages that are not compressed.

- `PAGE_STATE`

  The page state. The following table shows the permitted values.

Table 25.2 INNODB_BUFFER_PAGE.PAGE_STATE Values

Page State	Description
`FILE_PAGE`	A buffered file page
`MEMORY`	Contains a main memory object
`NOT_USED`	In the free list
`NULL`	Clean compressed pages, compressed pages in the flush list, pages used as buffer pool watch sentinels
`READY_FOR_USE`	A free page

Page State	Description
REMOVE_HASH	Hash index should be removed before placing in the free list

- IO_FIX

  Whether any I/O is pending for this page: IO_NONE = no pending I/O, IO_READ = read pending, IO_WRITE = write pending.

- IS_OLD

  Whether the block is in the sublist of old blocks in the LRU list.

- FREE_PAGE_CLOCK

  The value of the freed_page_clock counter when the block was the last placed at the head of the LRU list. The freed_page_clock counter tracks the number of blocks removed from the end of the LRU list.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE LIMIT 1\G
*************************** 1. row ***************************
            POOL_ID: 0
          BLOCK_ID: 0
             SPACE: 97
       PAGE_NUMBER: 2473
         PAGE_TYPE: INDEX
        FLUSH_TYPE: 1
         FIX_COUNT: 0
         IS_HASHED: YES
NEWEST_MODIFICATION: 733855581
OLDEST_MODIFICATION: 0
        ACCESS_TIME: 3378385672
         TABLE_NAME: `employees`.`salaries`
         INDEX_NAME: PRIMARY
     NUMBER_RECORDS: 468
          DATA_SIZE: 14976
    COMPRESSED_SIZE: 0
         PAGE_STATE: FILE_PAGE
             IO_FIX: IO_NONE
             IS_OLD: YES
    FREE_PAGE_CLOCK: 66
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.
- When tables, table rows, partitions, or indexes are deleted, associated pages remain in the buffer pool until space is required for other data. The INNODB_BUFFER_PAGE table reports information about these pages until they are evicted from the buffer pool. For more information about how the InnoDB manages buffer pool data, see Section 15.5.1, “Buffer Pool”.

25.39.2 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table

The `INNODB_BUFFER_PAGE_LRU` table provides information about the pages in the `InnoDB` buffer pool; in particular, how they are ordered in the LRU list that determines which pages to evict from the buffer pool when it becomes full.

The `INNODB_BUFFER_PAGE_LRU` table has the same columns as the `INNODB_BUFFER_PAGE` table, except that the `INNODB_BUFFER_PAGE_LRU` table has `LRU_POSITION` and `COMPRESSED` columns instead of `BLOCK_ID` and `PAGE_STATE` columns.

For related usage information and examples, see Section 15.14.5, "InnoDB INFORMATION_SCHEMA Buffer Pool Tables".

Warning

Querying the `INNODB_BUFFER_PAGE_LRU` table can affect performance. Do not query this table on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

The `INNODB_BUFFER_PAGE_LRU` table has these columns:

- `POOL_ID`

  The buffer pool ID. This is an identifier to distinguish between multiple buffer pool instances.

- `LRU_POSITION`

  The position of the page in the LRU list.

- `SPACE`

  The tablespace ID; the same value as `INNODB_TABLES.SPACE`.

- `PAGE_NUMBER`

  The page number.

- `PAGE_TYPE`

  The page type. The following table shows the permitted values.

Table 25.3 INNODB_BUFFER_PAGE_LRU.PAGE_TYPE Values

Page Type	Description
`ALLOCATED`	Freshly allocated page
`BLOB`	Uncompressed BLOB page
`COMPRESSED_BLOB2`	Subsequent comp BLOB page
`COMPRESSED_BLOB`	First compressed BLOB page
`ENCRYPTED_RTREE`	Encrypted R-tree
`EXTENT_DESCRIPTOR`	Extent descriptor page
`FILE_SPACE_HEADER`	File space header
`FIL_PAGE_TYPE_UNUSED`	Unused
`IBUF_BITMAP`	Insert buffer bitmap

Page Type	Description
IBUF_FREE_LIST	Insert buffer free list
IBUF_INDEX	Insert buffer index
INDEX	B-tree node
INODE	Index node
LOB_DATA	Uncompressed LOB data
LOB_FIRST	First page of uncompressed LOB
LOB_INDEX	Uncompressed LOB index
PAGE_IO_COMPRESSED	Compressed page
PAGE_IO_COMPRESSED_ENCRYPTED	Compressed and encrypted page
PAGE_IO_ENCRYPTED	Encrypted page
RSEG_ARRAY	Rollback segment array
RTREE_INDEX	R-tree index
SDI_BLOB	Uncompressed SDI BLOB
SDI_COMPRESSED_BLOB	Compressed SDI BLOB
SDI_INDEX	SDI index
SYSTEM	System page
TRX_SYSTEM	Transaction system data
UNDO_LOG	Undo log page
UNKNOWN	Unknown
ZLOB_DATA	Compressed LOB data
ZLOB_FIRST	First page of compressed LOB
ZLOB_FRAG	Compressed LOB fragment
ZLOB_FRAG_ENTRY	Compressed LOB fragment index
ZLOB_INDEX	Compressed LOB index

- FLUSH_TYPE

  The flush type.

- FIX_COUNT

  The number of threads using this block within the buffer pool. When zero, the block is eligible to be evicted.

- IS_HASHED

  Whether a hash index has been built on this page.

- NEWEST_MODIFICATION

  The Log Sequence Number of the youngest modification.

- `OLDEST_MODIFICATION`

  The Log Sequence Number of the oldest modification.

- `ACCESS_TIME`

  An abstract number used to judge the first access time of the page.

- `TABLE_NAME`

  The name of the table the page belongs to. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `INDEX_NAME`

  The name of the index the page belongs to. This can be the name of a clustered index or a secondary index. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `NUMBER_RECORDS`

  The number of records within the page.

- `DATA_SIZE`

  The sum of the sizes of the records. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `COMPRESSED_SIZE`

  The compressed page size. `NULL` for pages that are not compressed.

- `COMPRESSED`

  Whether the page is compressed.

- `IO_FIX`

  Whether any I/O is pending for this page: `IO_NONE` = no pending I/O, `IO_READ` = read pending, `IO_WRITE` = write pending.

- `IS_OLD`

  Whether the block is in the sublist of old blocks in the LRU list.

- `FREE_PAGE_CLOCK`

  The value of the `freed_page_clock` counter when the block was the last placed at the head of the LRU list. The `freed_page_clock` counter tracks the number of blocks removed from the end of the LRU list.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU LIMIT 1\G
*************************** 1. row ***************************
             POOL_ID: 0
       LRU_POSITION: 0
              SPACE: 97
        PAGE_NUMBER: 1984
          PAGE_TYPE: INDEX
         FLUSH_TYPE: 1
```

```
          FIX_COUNT: 0
          IS_HASHED: YES
NEWEST_MODIFICATION: 719490396
OLDEST_MODIFICATION: 0
        ACCESS_TIME: 3378383796
         TABLE_NAME: `employees`.`salaries`
         INDEX_NAME: PRIMARY
     NUMBER_RECORDS: 468
          DATA_SIZE: 14976
    COMPRESSED_SIZE: 0
         COMPRESSED: NO
             IO_FIX: IO_NONE
             IS_OLD: YES
    FREE_PAGE_CLOCK: 0
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- Querying this table can require MySQL to allocate a large block of contiguous memory, more than 64 bytes times the number of active pages in the buffer pool. This allocation could potentially cause an out-of-memory error, especially for systems with multi-gigabyte buffer pools.
- Querying this table requires MySQL to lock the data structure representing the buffer pool while traversing the LRU list, which can reduce concurrency, especially for systems with multi-gigabyte buffer pools.
- When tables, table rows, partitions, or indexes are deleted, associated pages remain in the buffer pool until space is required for other data. The `INNODB_BUFFER_PAGE_LRU` table reports information about these pages until they are evicted from the buffer pool. For more information about how the `InnoDB` manages buffer pool data, see Section 15.5.1, “Buffer Pool”.

25.39.3 The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table

The `INNODB_BUFFER_POOL_STATS` table provides much of the same buffer pool information provided in `SHOW ENGINE INNODB STATUS` output. Much of the same information may also be obtained using `InnoDB` buffer pool server status variables.

The idea of making pages in the buffer pool “young” or “not young” refers to transferring them between the sublists at the head and tail of the buffer pool data structure. Pages made “young” take longer to age out of the buffer pool, while pages made “not young” are moved much closer to the point of eviction.

For related usage information and examples, see Section 15.14.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”.

The `INNODB_BUFFER_POOL_STATS` table has these columns:

- `POOL_ID`

  The buffer pool ID. This is an identifier to distinguish between multiple buffer pool instances.

- `POOL_SIZE`

  The `InnoDB` buffer pool size in pages.

- `FREE_BUFFERS`

  The number of free pages in the `InnoDB` buffer pool.

- `DATABASE_PAGES`

  The number of pages in the `InnoDB` buffer pool containing data. This number includes both dirty and clean pages.

- `OLD_DATABASE_PAGES`

  The number of pages in the `old` buffer pool sublist.

- `MODIFIED_DATABASE_PAGES`

  The number of modified (dirty) database pages.

- `PENDING_DECOMPRESS`

  The number of pages pending decompression.

- `PENDING_READS`

  The number of pending reads.

- `PENDING_FLUSH_LRU`

  The number of pages pending flush in the LRU.

- `PENDING_FLUSH_LIST`

  The number of pages pending flush in the flush list.

- `PAGES_MADE_YOUNG`

  The number of pages made young.

- `PAGES_NOT_MADE_YOUNG`

  The number of pages not made young.

- `PAGES_MADE_YOUNG_RATE`

  The number of pages made young per second (pages made young since the last printout / time elapsed).

- `PAGES_MADE_NOT_YOUNG_RATE`

  The number of pages not made per second (pages not made young since the last printout / time elapsed).

- `NUMBER_PAGES_READ`

  The number of pages read.

- `NUMBER_PAGES_CREATED`

  The number of pages created.

- `NUMBER_PAGES_WRITTEN`

The number of pages written.

- PAGES_READ_RATE

  The number of pages read per second (pages read since the last printout / time elapsed).

- PAGES_CREATE_RATE

  The number of pages created per second (pages created since the last printout / time elapsed).

- PAGES_WRITTEN_RATE

  The number of pages written per second (pages written since the last printout / time elapsed).

- NUMBER_PAGES_GET

  The number of logical read requests.

- HIT_RATE

  The buffer pool hit rate.

- YOUNG_MAKE_PER_THOUSAND_GETS

  The number of pages made young per thousand gets.

- NOT_YOUNG_MAKE_PER_THOUSAND_GETS

  The number of pages not made young per thousand gets.

- NUMBER_PAGES_READ_AHEAD

  The number of pages read ahead.

- NUMBER_READ_AHEAD_EVICTED

  The number of pages read into the InnoDB buffer pool by the read-ahead background thread that were subsequently evicted without having been accessed by queries.

- READ_AHEAD_RATE

  The read-ahead rate per second (pages read ahead since the last printout / time elapsed).

- READ_AHEAD_EVICTED_RATE

  The number of read-ahead pages evicted without access per second (read-ahead pages not accessed since the last printout / time elapsed).

- LRU_IO_TOTAL

  Total LRU I/O.

- LRU_IO_CURRENT

  LRU I/O for the current interval.

- UNCOMPRESS_TOTAL

  The total number of pages decompressed.

- UNCOMPRESS_CURRENT

  The number of pages decompressed in the current interval.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_POOL_STATS\G
*************************** 1. row ***************************
                     POOL_ID: 0
                   POOL_SIZE: 8192
                FREE_BUFFERS: 1
              DATABASE_PAGES: 8085
          OLD_DATABASE_PAGES: 2964
     MODIFIED_DATABASE_PAGES: 0
          PENDING_DECOMPRESS: 0
               PENDING_READS: 0
           PENDING_FLUSH_LRU: 0
          PENDING_FLUSH_LIST: 0
            PAGES_MADE_YOUNG: 22821
        PAGES_NOT_MADE_YOUNG: 3544303
       PAGES_MADE_YOUNG_RATE: 357.62602199870594
   PAGES_MADE_NOT_YOUNG_RATE: 0
           NUMBER_PAGES_READ: 2389
        NUMBER_PAGES_CREATED: 12385
        NUMBER_PAGES_WRITTEN: 13111
             PAGES_READ_RATE: 0
           PAGES_CREATE_RATE: 0
          PAGES_WRITTEN_RATE: 0
            NUMBER_PAGES_GET: 33322210
                    HIT_RATE: 1000
YOUNG_MAKE_PER_THOUSAND_GETS: 18
NOT_YOUNG_MAKE_PER_THOUSAND_GETS: 0
     NUMBER_PAGES_READ_AHEAD: 2024
   NUMBER_READ_AHEAD_EVICTED: 0
          READ_AHEAD_RATE: 0
    READ_AHEAD_EVICTED_RATE: 0
               LRU_IO_TOTAL: 0
             LRU_IO_CURRENT: 0
           UNCOMPRESS_TOTAL: 0
         UNCOMPRESS_CURRENT: 0
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

25.39.4 The INFORMATION_SCHEMA INNODB_CACHED_INDEXES Table

The INNODB_CACHED_INDEXES table reports the number of index pages cached in the InnoDB buffer pool for each index.

For related usage information and examples, see Section 15.14.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”.

The INNODB_CACHED_INDEXES table has these columns:

- SPACE_ID

The tablespace ID.

- `INDEX_ID`

  An identifier for the index. Index identifiers are unique across all the databases in an instance.

- `N_CACHED_PAGES`

  The number of index pages cached in the `InnoDB` buffer pool.

Examples

This query returns the number of index pages cached in the `InnoDB` buffer pool for a specific index:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CACHED_INDEXES WHERE INDEX_ID=65\G
*************************** 1. row ***************************
      SPACE_ID: 4294967294
      INDEX_ID: 65
N_CACHED_PAGES: 45
```

This query returns the number of index pages cached in the `InnoDB` buffer pool for each index, using the `INNODB_INDEXES` and `INNODB_TABLES` tables to resolve the table name and index name for each `INDEX_ID` value.

```
SELECT
  tables.NAME AS table_name,
  indexes.NAME AS index_name,
  cached.N_CACHED_PAGES AS n_cached_pages
FROM
  INFORMATION_SCHEMA.INNODB_CACHED_INDEXES AS cached,
  INFORMATION_SCHEMA.INNODB_INDEXES AS indexes,
  INFORMATION_SCHEMA.INNODB_TABLES AS tables
WHERE
  cached.INDEX_ID = indexes.INDEX_ID
  AND indexes.TABLE_ID = tables.TABLE_ID;
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.39.5 The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables

The `INNODB_CMP` and `INNODB_CMP_RESET` tables provide status information on operations related to compressed `InnoDB` tables.

The `INNODB_CMP` and `INNODB_CMP_RESET` tables have these columns:

- `PAGE_SIZE`

  The compressed page size in bytes.

- `COMPRESS_OPS`

  The number of times a B-tree page of size `PAGE_SIZE` has been compressed. Pages are compressed whenever an empty page is created or the space for the uncompressed modification log runs out.

- `COMPRESS_OPS_OK`

  The number of times a B-tree page of size `PAGE_SIZE` has been successfully compressed. This count should never exceed `COMPRESS_OPS`.

- `COMPRESS_TIME`

  The total time in seconds used for attempts to compress B-tree pages of size `PAGE_SIZE`.

- `UNCOMPRESS_OPS`

  The number of times a B-tree page of size `PAGE_SIZE` has been uncompressed. B-tree pages are uncompressed whenever compression fails or at first access when the uncompressed page does not exist in the buffer pool.

- `UNCOMPRESS_TIME`

  The total time in seconds used for uncompressing B-tree pages of the size `PAGE_SIZE`.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMP\G
*************************** 1. row ***************************
      page_size: 1024
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
*************************** 2. row ***************************
      page_size: 2048
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
*************************** 3. row ***************************
      page_size: 4096
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
*************************** 4. row ***************************
      page_size: 8192
   compress_ops: 86955
compress_ops_ok: 81182
  compress_time: 27
 uncompress_ops: 26828
uncompress_time: 5
*************************** 5. row ***************************
      page_size: 16384
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
```

Notes

- Use these tables to measure the effectiveness of `InnoDB` table compression in your database.
- You must have the `PROCESS` privilege to query this table.

- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For usage information, see Section 15.9.1.4, “Monitoring InnoDB Table Compression at Runtime” and Section 15.14.1.3, “Using the Compression Information Schema Tables”. For general information about `InnoDB` table compression, see Section 15.9, “InnoDB Table and Page Compression”.

25.39.6 The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables

The `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` tables provide status information on compressed pages within the `InnoDB` buffer pool.

The `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` tables have these columns:

- `PAGE_SIZE`

  The block size in bytes. Each record of this table describes blocks of this size.

- `BUFFER_POOL_INSTANCE`

  A unique identifier for the buffer pool instance.

- `PAGES_USED`

  The number of blocks of size `PAGE_SIZE` that are currently in use.

- `PAGES_FREE`

  The number of blocks of size `PAGE_SIZE` that are currently available for allocation. This column shows the external fragmentation in the memory pool. Ideally, these numbers should be at most 1.

- `RELOCATION_OPS`

  The number of times a block of size `PAGE_SIZE` has been relocated. The buddy system can relocate the allocated “buddy neighbor” of a freed block when it tries to form a bigger freed block. Reading from the `INNODB_CMPMEM_RESET` table resets this count.

- `RELOCATION_TIME`

  The total time in microseconds used for relocating blocks of size `PAGE_SIZE`. Reading from the table `INNODB_CMPMEM_RESET` resets this count.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMPMEM\G
*************************** 1. row ***************************
           page_size: 1024
buffer_pool_instance: 0
          pages_used: 0
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
*************************** 2. row ***************************
           page_size: 2048
buffer_pool_instance: 0
          pages_used: 0
          pages_free: 0
      relocation_ops: 0
```

```
     relocation_time: 0
*************************** 3. row ***************************
           page_size: 4096
buffer_pool_instance: 0
          pages_used: 0
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
*************************** 4. row ***************************
           page_size: 8192
buffer_pool_instance: 0
          pages_used: 7673
          pages_free: 15
      relocation_ops: 4638
     relocation_time: 0
*************************** 5. row ***************************
           page_size: 16384
buffer_pool_instance: 0
          pages_used: 0
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
```

Notes

- Use these tables to measure the effectiveness of `InnoDB` table compression in your database.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For usage information, see Section 15.9.1.4, “Monitoring InnoDB Table Compression at Runtime” and Section 15.14.1.3, “Using the Compression Information Schema Tables”. For general information about `InnoDB` table compression, see Section 15.9, “InnoDB Table and Page Compression”.

25.39.7 The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables

The `INNODB_CMP_PER_INDEX` and `INNODB_CMP_PER_INDEX_RESET` tables provide status information on operations related to compressed `InnoDB` tables and indexes, with separate statistics for each combination of database, table, and index, to help you evaluate the performance and usefulness of compression for specific tables.

For a compressed `InnoDB` table, both the table data and all the secondary indexes are compressed. In this context, the table data is treated as just another index, one that happens to contain all the columns: the clustered index.

The `INNODB_CMP_PER_INDEX` and `INNODB_CMP_PER_INDEX_RESET` tables have these columns:

- `DATABASE_NAME`

  The schema (database) containing the applicable table.

- `TABLE_NAME`

  The table to monitor for compression statistics.

- `INDEX_NAME`

  The index to monitor for compression statistics.

- COMPRESS_OPS

  The number of compression operations attempted. Pages are compressed whenever an empty page is created or the space for the uncompressed modification log runs out.

- COMPRESS_OPS_OK

  The number of successful compression operations. Subtract from the COMPRESS_OPS value to get the number of compression failures. Divide by the COMPRESS_OPS value to get the percentage of compression failures.

- COMPRESS_TIME

  The total time in seconds used for compressing data in this index.

- UNCOMPRESS_OPS

  The number of uncompression operations performed. Compressed InnoDB pages are uncompressed whenever compression fails, or the first time a compressed page is accessed in the buffer pool and the uncompressed page does not exist.

- UNCOMPRESS_TIME

  The total time in seconds used for uncompressing data in this index.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX\G
*************************** 1. row ***************************
  database_name: employees
     table_name: salaries
     index_name: PRIMARY
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 23451
uncompress_time: 4
*************************** 2. row ***************************
  database_name: employees
     table_name: salaries
     index_name: emp_no
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 1597
uncompress_time: 0
```

Notes

- Use these tables to measure the effectiveness of InnoDB table compression for specific tables, indexes, or both.
- You must have the PROCESS privilege to query these tables.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of these tables, including data types and default values.
- Because collecting separate measurements for every index imposes substantial performance overhead, INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET statistics are not gathered by

default. You must enable the `innodb_cmp_per_index_enabled` system variable before performing the operations on compressed tables that you want to monitor.

- For usage information, see Section 15.9.1.4, “Monitoring InnoDB Table Compression at Runtime” and Section 15.14.1.3, “Using the Compression Information Schema Tables”. For general information about `InnoDB` table compression, see Section 15.9, “InnoDB Table and Page Compression”.

25.39.8 The INFORMATION_SCHEMA INNODB_COLUMNS Table

The `INNODB_COLUMNS` table provides metadata about `InnoDB` table columns.

For related usage information and examples, see Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”.

The `INNODB_COLUMNS` table has these columns:

- `TABLE_ID`

  An identifier representing the table associated with the column; the same value as `INNODB_TABLES.TABLE_ID`.

- `NAME`

  The name of the column. These names can be uppercase or lowercase depending on the `lower_case_table_names` setting. There are no special system-reserved names for columns.

- `POS`

  The ordinal position of the column within the table, starting from 0 and incrementing sequentially. When a column is dropped, the remaining columns are reordered so that the sequence has no gaps. The `POS` value for a virtual generated column encodes the column sequence number and ordinal position of the column. For more information, see the `POS` column description in Section 25.39.30, “The INFORMATION_SCHEMA INNODB_VIRTUAL Table”.

- `MTYPE`

  Stands for “main type”. A numeric identifier for the column type. 1 = `VARCHAR`, 2 = `CHAR`, 3 = `FIXBINARY`, 4 = `BINARY`, 5 = `BLOB`, 6 = `INT`, 7 = `SYS_CHILD`, 8 = `SYS`, 9 = `FLOAT`, 10 = `DOUBLE`, 11 = `DECIMAL`, 12 = `VARMYSQL`, 13 = `MYSQL`, 14 = `GEOMETRY`.

- `PRTYPE`

  The `InnoDB` “precise type”, a binary value with bits representing MySQL data type, character set code, and nullability.

- `LEN`

  The column length, for example 4 for `INT` and 8 for `BIGINT`. For character columns in multibyte character sets, this length value is the maximum length in bytes needed to represent a definition such as `VARCHAR(N)`; that is, it might be `2*N`, `3*N`, and so on depending on the character encoding.

- `HAS_DEFAULT`

  A boolean value indicating whether a column that was added instantly using `ALTER TABLE ... ADD COLUMN` with `ALGORITHM=INSTANT` has a default value. All columns added instantly have a default value, which makes this column an indicator of whether the column was added instantly.

- `DEFAULT_VALUE`

The initial default value of a column that was added instantly using `ALTER TABLE ... ADD COLUMN` with `ALGORITHM=INSTANT`. If the default value is `NULL` or was not specified, this column reports `NULL`. An explicitly specified non-`NULL` default value is shown in an internal binary format. Subsequent modifications of the column default value do not change the value reported by this column.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_COLUMNS where TABLE_ID = 71\G
*************************** 1. row ***************************
     TABLE_ID: 71
         NAME: col1
          POS: 0
        MTYPE: 6
       PRTYPE: 1027
          LEN: 4
  HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
*************************** 2. row ***************************
     TABLE_ID: 71
         NAME: col2
          POS: 1
        MTYPE: 2
       PRTYPE: 524542
          LEN: 10
  HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
*************************** 3. row ***************************
     TABLE_ID: 71
         NAME: col3
          POS: 2
        MTYPE: 1
       PRTYPE: 524303
          LEN: 10
  HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.39.9 The INFORMATION_SCHEMA INNODB_DATAFILES Table

The `INNODB_DATAFILES` table provides data file path information for `InnoDB` file-per-table and general tablespaces.

For related usage information and examples, see Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”.

Note

The `INFORMATION_SCHEMA FILES` table reports metadata for `InnoDB` tablespace types including file-per-table tablespaces, general tablespaces, the system tablespace, the global temporary tablespace, and undo tablespaces.

The `INNODB_DATAFILES` table has these columns:

- `SPACE`

The tablespace ID.

- PATH

  The tablespace data file path. If a file-per-table tablespace is created in a location outside the MySQL data directory, the path value is a fully qualified directory path. Otherwise, the path is relative to the data directory.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_DATAFILES WHERE SPACE = 57\G
*************************** 1. row ***************************
SPACE: 57
 PATH: ./test/t1.ibd
```

Notes

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

25.39.10 The INFORMATION_SCHEMA INNODB_FIELDS Table

The INNODB_FIELDS table provides metadata about the key columns (fields) of InnoDB indexes.

For related usage information and examples, see Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”.

The INNODB_FIELDS table has these columns:

- INDEX_ID

  An identifier for the index associated with this key field; the same value as INNODB_INDEXES.INDEX_ID.

- NAME

  The name of the original column from the table; the same value as INNODB_COLUMNS.NAME.

- POS

  The ordinal position of the key field within the index, starting from 0 and incrementing sequentially. When a column is dropped, the remaining columns are reordered so that the sequence has no gaps.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FIELDS WHERE INDEX_ID = 117\G
*************************** 1. row ***************************
INDEX_ID: 117
    NAME: col1
     POS: 0
```

Notes

- You must have the PROCESS privilege to query this table.

- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.39.11 The INFORMATION_SCHEMA INNODB_FOREIGN Table

The `INNODB_FOREIGN` table provides metadata about `InnoDB` foreign keys.

For related usage information and examples, see Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”.

The `INNODB_FOREIGN` table has these columns:

- `ID`

  The name (not a numeric value) of the foreign key index, preceded by the schema (database) name (for example, `test/products_fk`).

- `FOR_NAME`

  The name of the child table in this foreign key relationship.

- `REF_NAME`

  The name of the parent table in this foreign key relationship.

- `N_COLS`

  The number of columns in the foreign key index.

- `TYPE`

  A collection of bit flags with information about the foreign key column, ORed together. 0 = `ON DELETE/UPDATE RESTRICT`, 1 = `ON DELETE CASCADE`, 2 = `ON DELETE SET NULL`, 4 = `ON UPDATE CASCADE`, 8 = `ON UPDATE SET NULL`, 16 = `ON DELETE NO ACTION`, 32 = `ON UPDATE NO ACTION`.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN\G
*************************** 1. row ***************************
      ID: test/fk1
FOR_NAME: test/child
REF_NAME: test/parent
  N_COLS: 1
    TYPE: 1
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.39.12 The INFORMATION_SCHEMA INNODB_FOREIGN_COLS Table

The `INNODB_FOREIGN_COLS` table provides status information about `InnoDB` foreign key columns.

For related usage information and examples, see Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”.

The INNODB_FOREIGN_COLS table has these columns:

- ID

  The foreign key index associated with this index key field; the same value as INNODB_FOREIGN.ID.

- FOR_COL_NAME

  The name of the associated column in the child table.

- REF_COL_NAME

  The name of the associated column in the parent table.

- POS

  The ordinal position of this key field within the foreign key index, starting from 0.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN_COLS WHERE ID = 'test/fk1'\G
*************************** 1. row ***************************
          ID: test/fk1
FOR_COL_NAME: parent_id
REF_COL_NAME: id
         POS: 0
```

Notes

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

25.39.13 The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table

The INNODB_FT_BEING_DELETED table provides a snapshot of the INNODB_FT_DELETED table; it is used only during an OPTIMIZE TABLE maintenance operation. When OPTIMIZE TABLE is run, the INNODB_FT_BEING_DELETED table is emptied, and DOC_ID values are removed from the INNODB_FT_DELETED table. Because the contents of INNODB_FT_BEING_DELETED typically have a short lifetime, this table has limited utility for monitoring or debugging. For information about running OPTIMIZE TABLE on tables with FULLTEXT indexes, see Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”.

This table is empty initially. Before querying it, set the value of the innodb_ft_aux_table system variable to the name (including the database name) of the table that contains the FULLTEXT index (for example, test/articles). The output appears similar to the example provided for the INNODB_FT_DELETED table.

For related usage information and examples, see Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The INNODB_FT_BEING_DELETED table has these columns:

- DOC_ID

  The document ID of the row that is in the process of being deleted. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by

InnoDB when the table contains no suitable column. This value is used when you perform text searches, to skip rows in the INNODB_FT_INDEX_TABLE table before data for deleted rows is physically removed from the FULLTEXT index by an OPTIMIZE TABLE statement. For more information, see Optimizing InnoDB Full-Text Indexes.

Notes

- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.
- You must have the PROCESS privilege to query this table.
- For more information about InnoDB FULLTEXT search, see Section 15.6.2.4, “InnoDB FULLTEXT Indexes”, and Section 12.9, “Full-Text Search Functions”.

25.39.14 The INFORMATION_SCHEMA INNODB_FT_CONFIG Table

The INNODB_FT_CONFIG table provides metadata about the FULLTEXT index and associated processing for an InnoDB table.

This table is empty initially. Before querying it, set the value of the innodb_ft_aux_table system variable to the name (including the database name) of the table that contains the FULLTEXT index (for example, test/articles).

For related usage information and examples, see Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The INNODB_FT_CONFIG table has these columns:

- KEY

  The name designating an item of metadata for an InnoDB table containing a FULLTEXT index.

  The values for this column might change, depending on the needs for performance tuning and debugging for InnoDB full-text processing. The key names and their meanings include:

  - optimize_checkpoint_limit: The number of seconds after which an OPTIMIZE TABLE run stops.
  - synced_doc_id: The next DOC_ID to be issued.
  - stopword_table_name: The *database/table* name for a user-defined stopword table. The VALUE column is empty if there is no user-defined stopword table.
  - use_stopword: Indicates whether a stopword table is used, which is defined when the FULLTEXT index is created.

- VALUE

  The value associated with the corresponding KEY column, reflecting some limit or current value for an aspect of a FULLTEXT index for an InnoDB table.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_CONFIG;
+---------------------------+------------------+
| KEY                       | VALUE            |
```

```
+---------------------------+-------------------+
| optimize_checkpoint_limit | 180               |
| synced_doc_id             | 0                 |
| stopword_table_name       | test/my_stopwords |
| use_stopword              | 1                 |
+---------------------------+-------------------+
```

Notes

- This table is intended only for internal configuration. It is not intended for statistical information purposes.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see Section 15.6.2.4, “InnoDB FULLTEXT Indexes”, and Section 12.9, “Full-Text Search Functions”.

25.39.15 The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table

The `INNODB_FT_DEFAULT_STOPWORD` table holds a list of stopwords that are used by default when creating a `FULLTEXT` index on `InnoDB` tables. For information about the default `InnoDB` stopword list and how to define your own stopword lists, see Section 12.9.4, “Full-Text Stopwords”.

For related usage information and examples, see Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The `INNODB_FT_DEFAULT_STOPWORD` table has these columns:

- `value`

  A word that is used by default as a stopword for `FULLTEXT` indexes on `InnoDB` tables. This is not used if you override the default stopword processing with either the `innodb_ft_server_stopword_table` or the `innodb_ft_user_stopword_table` system variable.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD;
+-------+
| value |
+-------+
| a     |
| about |
| an    |
| are   |
| as    |
| at    |
| be    |
| by    |
| com   |
| de    |
| en    |
| for   |
| from  |
| how   |
| i     |
| in    |
| is    |
```

```
| it    |
| la    |
| of    |
| on    |
| or    |
| that  |
| the   |
| this  |
| to    |
| was   |
| what  |
| when  |
| where |
| who   |
| will  |
| with  |
| und   |
| the   |
| www   |
+-------+
36 rows in set (0.00 sec)
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see Section 15.6.2.4, “InnoDB FULLTEXT Indexes”, and Section 12.9, “Full-Text Search Functions”.

25.39.16 The INFORMATION_SCHEMA INNODB_FT_DELETED Table

The `INNODB_FT_DELETED` table stores rows that are deleted from the `FULLTEXT` index for an `InnoDB` table. To avoid expensive index reorganization during DML operations for an `InnoDB FULLTEXT` index, the information about newly deleted words is stored separately, filtered out of search results when you do a text search, and removed from the main search index only when you issue an `OPTIMIZE TABLE` statement for the `InnoDB` table. For more information, see Optimizing InnoDB Full-Text Indexes.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example, `test/articles`).

For related usage information and examples, see Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The `INNODB_FT_DELETED` table has these columns:

- `DOC_ID`

  The document ID of the newly deleted row. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by `InnoDB` when the table contains no suitable column. This value is used when you perform text searches, to skip rows in the `INNODB_FT_INDEX_TABLE` table before data for deleted rows is physically removed from the `FULLTEXT` index by an `OPTIMIZE TABLE` statement. For more information, see Optimizing InnoDB Full-Text Indexes.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
+--------+
| DOC_ID |
+--------+
|      6 |
|      7 |
|      8 |
+--------+
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see Section 15.6.2.4, “InnoDB FULLTEXT Indexes”, and Section 12.9, “Full-Text Search Functions”.

25.39.17 The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table

The `INNODB_FT_INDEX_CACHE` table provides token information about newly inserted rows in a `FULLTEXT` index. To avoid expensive index reorganization during DML operations, the information about newly indexed words is stored separately, and combined with the main search index only when `OPTIMIZE TABLE` is run, when the server is shut down, or when the cache size exceeds a limit defined by the `innodb_ft_cache_size` or `innodb_ft_total_cache_size` system variable.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example, `test/articles`).

For related usage information and examples, see Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The `INNODB_FT_INDEX_CACHE` table has these columns:

- `WORD`

  A word extracted from the text of a newly inserted row.

- `FIRST_DOC_ID`

  The first document ID in which this word appears in the `FULLTEXT` index.

- `LAST_DOC_ID`

  The last document ID in which this word appears in the `FULLTEXT` index.

- `DOC_COUNT`

  The number of rows in which this word appears in the `FULLTEXT` index. The same word can occur several times within the cache table, once for each combination of `DOC_ID` and `POSITION` values.

- `DOC_ID`

  The document ID of the newly inserted row. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by `InnoDB` when the table contains no suitable column.

- `POSITION`

  The position of this particular instance of the word within the relevant document identified by the `DOC_ID` value. The value does not represent an absolute position; it is an offset added to the `POSITION` of the previous instance of that word.

Notes

- This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example `test/articles`). The following example demonstrates how to use the `innodb_ft_aux_table` system variable to show information about a `FULLTEXT` index for a specified table.

```
mysql> USE test;

mysql> CREATE TABLE articles (
         id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
         title VARCHAR(200),
         body TEXT,
         FULLTEXT (title,body)
       ) ENGINE=InnoDB;

mysql> INSERT INTO articles (title,body) VALUES
       ('MySQL Tutorial','DBMS stands for DataBase ...'),
       ('How To Use MySQL Well','After you went through a ...'),
       ('Optimizing MySQL','In this tutorial we will show ...'),
       ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
       ('MySQL vs. YourSQL','In the following database comparison ...'),
       ('MySQL Security','When configured properly, MySQL ...');

mysql> SET GLOBAL innodb_ft_aux_table = 'test/articles';

mysql> SELECT WORD, DOC_COUNT, DOC_ID, POSITION
       FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE LIMIT 5;
+------------+-----------+--------+----------+
| WORD       | DOC_COUNT | DOC_ID | POSITION |
+------------+-----------+--------+----------+
| 1001       |         1 |      4 |        0 |
| after      |         1 |      2 |       22 |
| comparison |         1 |      5 |       44 |
| configured |         1 |      6 |       20 |
| database   |         2 |      1 |       31 |
+------------+-----------+--------+----------+
```

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see Section 15.6.2.4, “InnoDB FULLTEXT Indexes”, and Section 12.9, “Full-Text Search Functions”.

25.39.18 The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table

The `INNODB_FT_INDEX_TABLE` table provides information about the inverted index used to process text searches against the `FULLTEXT` index of an `InnoDB` table.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example, `test/articles`).

For related usage information and examples, see Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The INNODB_FT_INDEX_TABLE table has these columns:

- WORD

  A word extracted from the text of the columns that are part of a FULLTEXT.

- FIRST_DOC_ID

  The first document ID in which this word appears in the FULLTEXT index.

- LAST_DOC_ID

  The last document ID in which this word appears in the FULLTEXT index.

- DOC_COUNT

  The number of rows in which this word appears in the FULLTEXT index. The same word can occur several times within the cache table, once for each combination of DOC_ID and POSITION values.

- DOC_ID

  The document ID of the row containing the word. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by InnoDB when the table contains no suitable column.

- POSITION

  The position of this particular instance of the word within the relevant document identified by the DOC_ID value.

Notes

- This table is empty initially. Before querying it, set the value of the innodb_ft_aux_table system variable to the name (including the database name) of the table that contains the FULLTEXT index (for example, test/articles). The following example demonstrates how to use the innodb_ft_aux_table system variable to show information about a FULLTEXT index for a specified table. Before information for newly inserted rows appears in INNODB_FT_INDEX_TABLE, the FULLTEXT index cache must be flushed to disk. This is accomplished by running an OPTIMIZE TABLE operation on the indexed table with the innodb_optimize_fulltext_only system variable enabled. (The example disables that variable again at the end because it is intended to be enabled only temporarily.)

```
mysql> USE test;

mysql> CREATE TABLE articles (
         id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
         title VARCHAR(200),
         body TEXT,
         FULLTEXT (title,body)
       ) ENGINE=InnoDB;

mysql> INSERT INTO articles (title,body) VALUES
       ('MySQL Tutorial','DBMS stands for DataBase ...'),
       ('How To Use MySQL Well','After you went through a ...'),
       ('Optimizing MySQL','In this tutorial we will show ...'),
       ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
       ('MySQL vs. YourSQL','In the following database comparison ...'),
```

```
       ('MySQL Security','When configured properly, MySQL ...');

mysql> SET GLOBAL innodb_optimize_fulltext_only=ON;

mysql> OPTIMIZE TABLE articles;
+---------------+----------+----------+----------+
| Table         | Op       | Msg_type | Msg_text |
+---------------+----------+----------+----------+
| test.articles | optimize | status   | OK       |
+---------------+----------+----------+----------+

mysql> SET GLOBAL innodb_ft_aux_table = 'test/articles';

mysql> SELECT WORD, DOC_COUNT, DOC_ID, POSITION
       FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE LIMIT 5;
+------------+-----------+--------+----------+
| WORD       | DOC_COUNT | DOC_ID | POSITION |
+------------+-----------+--------+----------+
| 1001       |         1 |      4 |        0 |
| after      |         1 |      2 |       22 |
| comparison |         1 |      5 |       44 |
| configured |         1 |      6 |       20 |
| database   |         2 |      1 |       31 |
+------------+-----------+--------+----------+

mysql> SET GLOBAL innodb_optimize_fulltext_only=OFF;
```

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see Section 15.6.2.4, “InnoDB FULLTEXT Indexes”, and Section 12.9, “Full-Text Search Functions”.

25.39.19 The INFORMATION_SCHEMA INNODB_INDEXES Table

The `INNODB_INDEXES` table provides metadata about `InnoDB` indexes.

For related usage information and examples, see Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”.

The `INNODB_INDEXES` table has these columns:

- `INDEX_ID`

  An identifier for the index. Index identifiers are unique across all the databases in an instance.

- `NAME`

  The name of the index. Most indexes created implicitly by `InnoDB` have consistent names but the index names are not necessarily unique. Examples: `PRIMARY` for a primary key index, `GEN_CLUST_INDEX` for the index representing a primary key when one is not specified, and `ID_IND`, `FOR_IND`, and `REF_IND` for foreign key constraints.

- `TABLE_ID`

  An identifier representing the table associated with the index; the same value as `INNODB_TABLES.TABLE_ID`.

- `TYPE`

A numeric value derived from bit-level information that identifies the index type. 0 = nonunique secondary index; 1 = automatically generated clustered index (`GEN_CLUST_INDEX`); 2 = unique nonclustered index; 3 = clustered index; 32 = full-text index; 64 = spatial index; 128 = secondary index on a virtual generated column.

- `N_FIELDS`

  The number of columns in the index key. For `GEN_CLUST_INDEX` indexes, this value is 0 because the index is created using an artificial value rather than a real table column.

- `PAGE_NO`

  The root page number of the index B-tree. For full-text indexes, the `PAGE_NO` column is unused and set to -1 (`FIL_NULL`) because the full-text index is laid out in several B-trees (auxiliary tables).

- `SPACE`

  An identifier for the tablespace where the index resides. 0 means the `InnoDB` system tablespace. Any other number represents a table created with a separate `.ibd` file in file-per-table mode. This identifier stays the same after a `TRUNCATE TABLE` statement. Does all indexes for a table reside in the same tablespace as the table, this value is not necessarily unique.

- `MERGE_THRESHOLD`

  The merge threshold value for index pages. If the amount of data in an index page falls below the `MERGE_THRESHOLD` value when a row is deleted or when a row is shortened by an update operation, `InnoDB` attempts to merge the index page with the neighboring index page. The default threshold value is 50%. For more information, see Section 15.8.11, “Configuring the Merge Threshold for Index Pages”.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_INDEXES WHERE TABLE_ID = 34\G
*************************** 1. row ***************************
       INDEX_ID: 39
           NAME: GEN_CLUST_INDEX
       TABLE_ID: 34
           TYPE: 1
       N_FIELDS: 0
        PAGE_NO: 3
          SPACE: 23
MERGE_THRESHOLD: 50
*************************** 2. row ***************************
       INDEX_ID: 40
           NAME: i1
       TABLE_ID: 34
           TYPE: 0
       N_FIELDS: 1
        PAGE_NO: 4
          SPACE: 23
MERGE_THRESHOLD: 50
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.39.20 The INFORMATION_SCHEMA INNODB_LOCKS Table

The INNODB_LOCKS table provides information about each lock that an InnoDB transaction has requested but not yet acquired, and each lock that a transaction holds that is blocking another transaction.

Note

This table is deprecated and is removed as of MySQL 8.0.1. Use the Performance Schema data_locks table instead. See Section 26.12.12.1, "The data_locks Table".

Differences between INNODB_LOCKS and data_locks:

- If a transaction holds a lock, INNODB_LOCKS displays the lock only if another transaction is waiting for it. data_locks displays the lock regardless of whether any transaction is waiting for it.
- The data_locks table has no columns corresponding to LOCK_SPACE, LOCK_PAGE, or LOCK_REC.
- The INNODB_LOCKS table requires the global PROCESS privilege. The data_locks table requires the usual Performance Schema privilege of SELECT on the table to be selected from.

The following table shows the mapping from INNODB_LOCKS columns to data_locks columns. Use this information to migrate applications from one table to the other.

Table 25.4 Mapping from INNODB_LOCKS to data_locks Columns

INNODB_LOCKS Column	data_locks Column
LOCK_ID	ENGINE_LOCK_ID
LOCK_TRX_ID	ENGINE_TRANSACTION_ID
LOCK_MODE	LOCK_MODE
LOCK_TYPE	LOCK_TYPE
LOCK_TABLE (combined schema/table names)	OBJECT_SCHEMA (schema name), OBJECT_NAME (table name)
LOCK_INDEX	INDEX_NAME
LOCK_SPACE	None
LOCK_PAGE	None
LOCK_REC	None
LOCK_DATA	LOCK_DATA

25.39.21 The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table

The INNODB_LOCK_WAITS table contains one or more rows for each blocked InnoDB transaction, indicating the lock it has requested and any locks that are blocking that request.

Note

This table is deprecated and is removed as of MySQL 8.0.1. Use the Performance Schema data_lock_waits table instead. See Section 26.12.12.2, "The data_lock_waits Table".

> The tables differ in the privileges required: The `INNODB_LOCK_WAITS` table requires the global `PROCESS` privilege. The `data_lock_waits` table requires the usual Performance Schema privilege of `SELECT` on the table to be selected from.

The following table shows the mapping from `INNODB_LOCK_WAITS` columns to `data_lock_waits` columns. Use this information to migrate applications from one table to the other.

Table 25.5 Mapping from INNODB_LOCK_WAITS to data_lock_waits Columns

INNODB_LOCK_WAITS Column	data_lock_waits Column
`REQUESTING_TRX_ID`	`REQUESTING_ENGINE_TRANSACTION_ID`
`REQUESTED_LOCK_ID`	`REQUESTING_ENGINE_LOCK_ID`
`BLOCKING_TRX_ID`	`BLOCKING_ENGINE_TRANSACTION_ID`
`BLOCKING_LOCK_ID`	`BLOCKING_ENGINE_LOCK_ID`

25.39.22 The INFORMATION_SCHEMA INNODB_METRICS Table

The `INNODB_METRICS` table provides a wide variety of `InnoDB` performance information, complementing the specific focus areas of the Performance Schema tables for `InnoDB`. With simple queries, you can check the overall health of the system. With more detailed queries, you can diagnose issues such as performance bottlenecks, resource shortages, and application issues.

Each monitor represents a point within the `InnoDB` source code that is instrumented to gather counter information. Each counter can be started, stopped, and reset. You can also perform these actions for a group of counters using their common module name.

By default, relatively little data is collected. To start, stop, and reset counters, set one of the system variables `innodb_monitor_enable`, `innodb_monitor_disable`, `innodb_monitor_reset`, or `innodb_monitor_reset_all`, using the name of the counter, the name of the module, a wildcard match for such a name using the “%” character, or the special keyword `all`.

For usage information, see Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

The `INNODB_METRICS` table has these columns:

- `NAME`

  A unique name for the counter.

- `SUBSYSTEM`

  The aspect of `InnoDB` that the metric applies to.

- `COUNT`

  The value since the counter was enabled.

- `MAX_COUNT`

  The maximum value since the counter was enabled.

- `MIN_COUNT`

  The minimum value since the counter was enabled.

- `AVG_COUNT`

The average value since the counter was enabled.

- `COUNT_RESET`

  The counter value since it was last reset. (The `_RESET` columns act like the lap counter on a stopwatch: you can measure the activity during some time interval, while the cumulative figures are still available in `COUNT`, `MAX_COUNT`, and so on.)

- `MAX_COUNT_RESET`

  The maximum counter value since it was last reset.

- `MIN_COUNT_RESET`

  The minimum counter value since it was last reset.

- `AVG_COUNT_RESET`

  The average counter value since it was last reset.

- `TIME_ENABLED`

  The timestamp of the last start.

- `TIME_DISABLED`

  The timestamp of the last stop.

- `TIME_ELAPSED`

  The elapsed time in seconds since the counter started.

- `TIME_RESET`

  The timestamp of the last reset.

- `STATUS`

  Whether the counter is still running (`enabled`) or stopped (`disabled`).

- `TYPE`

  Whether the item is a cumulative counter, or measures the current value of some resource.

- `COMMENT`

  The counter description.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='dml_inserts'\G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.046153846153846156
    COUNT_RESET: 3
MAX_COUNT_RESET: 3
```

```
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: NULL
   TIME_ELAPSED: 65
     TIME_RESET: NULL
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- Transaction counter `COUNT` values may differ from the number of transaction events reported in Performance Schema `EVENTS_TRANSACTIONS_SUMMARY` tables. `InnoDB` counts only those transactions that it executes, whereas Performance Schema collects events for all non-aborted transactions initiated by the server, including empty transactions.

25.39.23 The INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES Table

The `INNODB_SESSION_TEMP_TABLESPACES` table provides metadata about session temporary tablespaces used for internal and user-created temporary tables. This table was added in MySQL 8.0.13.

The `INNODB_SESSION_TEMP_TABLESPACES` table has these columns:

- `ID`

  The process or session ID.

- `SPACE`

  The tablespace ID. A range of 400 thousand space IDs is reserved for session temporary tablespaces. Session temporary tablespaces are recreated each time the server is started. Space IDs are not persisted when the server is shut down and may be reused.

- `PATH`

  The tablespace data file path. A session temporary tablespace has an `ibt` file extension.

- `SIZE`

  The size of the tablespace, in bytes.

- `STATE`

  The state of the tablespace. `ACTIVE` indicates that the tablespace is currently used by a session. `INACTIVE` indicates that the tablespace is in the pool of available session temporary tablespaces.

- `PURPOSE`

  The purpose of the tablespace. `INTRINSIC` indicates that the tablespace is used for optimized internal temporary tables use by the optimizer. `SLAVE` indicates that the tablespace is allocated for storing user-created temporary tables on a replication slave. `USER` indicates that the tablespace is used for user-created temporary tables. `NONE` indicates that the tablespace is not in use.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SESSION_TEMP_TABLESPACES;
+----+------------+---------------------------+-------+----------+-----------+
| ID | SPACE      | PATH                      | SIZE  | STATE    | PURPOSE   |
+----+------------+---------------------------+-------+----------+-----------+
|  8 | 4294566162 | ./#innodb_temp/temp_10.ibt | 81920 | ACTIVE   | INTRINSIC |
|  8 | 4294566161 | ./#innodb_temp/temp_9.ibt  | 98304 | ACTIVE   | USER      |
|  0 | 4294566153 | ./#innodb_temp/temp_1.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566154 | ./#innodb_temp/temp_2.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566155 | ./#innodb_temp/temp_3.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566156 | ./#innodb_temp/temp_4.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566157 | ./#innodb_temp/temp_5.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566158 | ./#innodb_temp/temp_6.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566159 | ./#innodb_temp/temp_7.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566160 | ./#innodb_temp/temp_8.ibt  | 81920 | INACTIVE | NONE      |
+----+------------+---------------------------+-------+----------+-----------+
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.39.24 The INFORMATION_SCHEMA INNODB_TABLES Table

The `INNODB_TABLES` table provides metadata about `InnoDB` tables.

For related usage information and examples, see Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”.

The `INNODB_TABLES` table has these columns:

- `TABLE_ID`

  An identifier for the `InnoDB` table. This value is unique across all databases in the instance.

- `NAME`

  The name of the table, preceded by the schema (database) name where appropriate (for example, `test/t1`). Names of databases and user tables are in the same case as they were originally defined, possibly influenced by the `lower_case_table_names` setting.

- `FLAG`

  A numeric value that represents bit-level information about table format and storage characteristics.

- `N_COLS`

  The number of columns in the table. The number reported includes three hidden columns that are created by `InnoDB` (`DB_ROW_ID`, `DB_TRX_ID`, and `DB_ROLL_PTR`). The number reported also includes virtual generated columns, if present.

- `SPACE`

  An identifier for the tablespace where the table resides. 0 means the `InnoDB` system tablespace. Any other number represents either a file-per-table tablespace or a general tablespace. This identifier stays

the same after a `TRUNCATE TABLE` statement. For file-per-table tablespaces, this identifier is unique for tables across all databases in the instance.

- `ROW_FORMAT`

  The table's row format (`Compact`, `Redundant`, `Dynamic`, or `Compressed`).

- `ZIP_PAGE_SIZE`

  The zip page size. Applies only to tables with a row format of `Compressed`.

- `SPACE_TYPE`

  The type of tablespace to which the table belongs. Possible values include `System` for the system tablespace, `General` for general tablespaces, and `Single` for file-per-table tablespaces. Tables assigned to the system tablespace using `CREATE TABLE` or `ALTER TABLE TABLESPACE=innodb_system` have a `SPACE_TYPE` of `General`. For more information, see `CREATE TABLESPACE`.

- `INSTANT_COLS`

  The number of columns in the table prior to adding the first instant column using `ALTER TABLE ... ADD COLUMN` with `ALGORITHM=INSTANT`.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE TABLE_ID = 214\G
*************************** 1. row ***************************
     TABLE_ID: 214
         NAME: test/t1
         FLAG: 129
       N_COLS: 4
        SPACE: 233
   ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
   SPACE_TYPE: General
 INSTANT_COLS: 0
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.39.25 The INFORMATION_SCHEMA INNODB_TABLESPACES Table

The `INNODB_TABLESPACES` table provides metadata about `InnoDB` file-per-table, general, and undo tablespaces.

For related usage information and examples, see Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”.

Note

The `INFORMATION_SCHEMA FILES` table reports metadata for `InnoDB` tablespace types including file-per-table tablespaces, general tablespaces, the system tablespace, the global temporary tablespace, and undo tablespaces.

The `INNODB_TABLESPACES` table has these columns:

- `SPACE`

  The tablespace ID.

- `NAME`

  The schema (database) and table name.

- `FLAG`

  A numeric value that represents bit-level information about tablespace format and storage characteristics.

- `ROW_FORMAT`

  The tablespace row format (`Compact or Redundant`, `Dynamic` or `Compressed`, or `Undo`). The data in this column is interpreted from the tablespace flag information that resides in the data file.

  There is no way to determine from this flag information if the tablespace row format is `Redundant` or `Compact`, which is why one of the possible `ROW_FORMAT` values is `Compact or Redundant`.

- `PAGE_SIZE`

  The tablespace page size. The data in this column is interpreted from the tablespace flags information that resides in the `.ibd` file.

- `ZIP_PAGE_SIZE`

  The tablespace zip page size. The data in this column is interpreted from the tablespace flags information that resides in the `.ibd` file.

- `SPACE_TYPE`

  The type of tablespace. Possible values include `General` for general tablespaces, `Single` for file-per-table tablespaces, `System` for the system tablespace, and `Undo` for undo tablespaces.

- `FS_BLOCK_SIZE`

  The file system block size, which is the unit size used for hole punching. This column pertains to the `InnoDB` transparent page compression feature.

- `FILE_SIZE`

  The apparent size of the file, which represents the maximum size of the file, uncompressed. This column pertains to the `InnoDB` transparent page compression feature.

- `ALLOCATED_SIZE`

  The actual size of the file, which is the amount of space allocated on disk. This column pertains to the `InnoDB` transparent page compression feature.

- `SERVER_VERSION`

  The MySQL version that created the tablespace, or the MySQL version into which the tablespace was imported, or the version of the last major MySQL version upgrade. The value is unchanged by a release series upgrade, such as an upgrade from MySQL 8.0.*x* to 8.0.*y*. The value can be considered a “creation” marker or “certified” marker for the tablespace.

- SPACE_VERSION

  The tablespace version, used to track changes to the tablespace format.

- ENCRYPTION

  Whether the tablespace is encrypted. This column was added in MySQL 8.0.13.

- STATE

  The tablespace state. This column was added in MySQL 8.0.14.

  For file-per-table and general tablespaces, states include:

  - normal: The tablespace is normal and active.
  - discarded: The tablespace was discarded by an ALTER TABLE ... DISCARD TABLESPACE statement.
  - corrupted: The tablespace is identified by InnoDB as corrupted.

  For undo tablespaces, states include:

  - active: Rollback segments in the undo tablespace can be allocated to new transactions.
  - inactive: Rollback segments in the undo tablespace are no longer used by new transactions. The truncate process is in progress. The undo tablespace was either selected by the purge thread implicitly or was made inactive by an ALTER UNDO TABLESPACE ... SET INACTIVE statement.
  - empty: The undo tablespace was truncated and is no longer active. It is ready to be dropped or made active again by an ALTER UNDO TABLESPACE ... SET INACTIVE statement.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESPACES WHERE SPACE = 26\G
*************************** 1. row ***************************
         SPACE: 26
          NAME: test/t1
          FLAG: 0
    ROW_FORMAT: Compact or Redundant
     PAGE_SIZE: 16384
 ZIP_PAGE_SIZE: 0
    SPACE_TYPE: Single
 FS_BLOCK_SIZE: 4096
     FILE_SIZE: 98304
ALLOCATED_SIZE: 65536
SERVER_VERSION: 8.0.4
 SPACE_VERSION: 1
    ENCRYPTION: N
         STATE: normal
```

Notes

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

25.39.26 The INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF Table

The INNODB_TABLESPACES_BRIEF table provides space ID, name, path, flag, and space type metadata for file-per-table, general, undo, and system tablespaces.

INNODB_TABLESPACES provides the same metadata but loads more slowly because other metadata provided by the table, such as FS_BLOCK_SIZE, FILE_SIZE, and ALLOCATED_SIZE, must be loaded dynamically.

Space and path metadata is also provided by the INNODB_DATAFILES table.

The INNODB_TABLESPACES_BRIEF table has these columns:

- SPACE

  The tablespace ID.

- NAME

  The tablespace name. For file-per-table tablespaces, the name is in the form of *schema/table_name*.

- PATH

  The tablespace data file path. If a file-per-table tablespace is created in a location outside the MySQL data directory, the path value is a fully qualified directory path. Otherwise, the path is relative to the data directory.

- FLAG

  A numeric value that represents bit-level information about tablespace format and storage characteristics.

- SPACE_TYPE

  The type of tablespace. Possible values include General for InnoDB general tablespaces, Single for InnoDB file-per-table tablespaces, and System for the InnoDB system tablespace.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESPACES_BRIEF WHERE SPACE = 7;
+-------+---------+---------------+-------+------------+
| SPACE | NAME    | PATH          | FLAG  | SPACE_TYPE |
+-------+---------+---------------+-------+------------+
| 7     | test/t1 | ./test/t1.ibd | 16417 | Single     |
+-------+---------+---------------+-------+------------+
```

Notes

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

25.39.27 The INFORMATION_SCHEMA INNODB_TABLESTATS View

The INNODB_TABLESTATS table provides a view of low-level status information about InnoDB tables. This data is used by the MySQL optimizer to calculate which index to use when querying an InnoDB table. This information is derived from in-memory data structures rather than data stored on disk. There is no corresponding internal InnoDB system table.

InnoDB tables are represented in this view if they have been opened since the last server restart and have not aged out of the table cache. Tables for which persistent stats are available are always represented in this view.

Table statistics are updated only for DELETE or UPDATE operations that modify indexed columns. Statistics are not updated by operations that modify only nonindexed columns.

ANALYZE TABLE clears table statistics and sets the STATS_INITIALIZED column to Uninitialized. Statistics are collected again the next time the table is accessed.

For related usage information and examples, see Section 15.14.3, "InnoDB INFORMATION_SCHEMA Schema Object Tables".

The INNODB_TABLESTATS table has these columns:

- TABLE_ID

  An identifier representing the table for which statistics are available; the same value as INNODB_TABLES.TABLE_ID.

- NAME

  The name of the table; the same value as INNODB_TABLES.NAME.

- STATS_INITIALIZED

  The value is Initialized if the statistics are already collected, Uninitialized if not.

- NUM_ROWS

  The current estimated number of rows in the table. Updated after each DML operation. The value could be imprecise if uncommitted transactions are inserting into or deleting from the table.

- CLUST_INDEX_SIZE

  The number of pages on disk that store the clustered index, which holds the InnoDB table data in primary key order. This value might be null if no statistics are collected yet for the table.

- OTHER_INDEX_SIZE

  The number of pages on disk that store all secondary indexes for the table. This value might be null if no statistics are collected yet for the table.

- MODIFIED_COUNTER

  The number of rows modified by DML operations, such as INSERT, UPDATE, DELETE, and also cascade operations from foreign keys. This column is reset each time table statistics are recalculated

- AUTOINC

  The next number to be issued for any auto-increment-based operation. The rate at which the AUTOINC value changes depends on how many times auto-increment numbers have been requested and how many numbers are granted per request.

- REF_COUNT

  When this counter reaches zero, the table metadata can be evicted from the table cache.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESTATS where TABLE_ID = 71\G
*************************** 1. row ***************************
         TABLE_ID: 71
             NAME: test/t1
STATS_INITIALIZED: Initialized
         NUM_ROWS: 1
 CLUST_INDEX_SIZE: 1
 OTHER_INDEX_SIZE: 0
 MODIFIED_COUNTER: 1
          AUTOINC: 0
        REF_COUNT: 1
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.39.28 The INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO Table

The `INNODB_TEMP_TABLE_INFO` table provides information about user-created `InnoDB` temporary tables that are active in an `InnoDB` instance. It does not provide information about internal `InnoDB` temporary tables used by the optimizer. The `INNODB_TEMP_TABLE_INFO` table is created when first queried, exists only in memory, and is not persisted to disk.

For usage information and examples, see Section 15.14.7, “InnoDB INFORMATION_SCHEMA Temporary Table Info Table”.

The `INNODB_TEMP_TABLE_INFO` table has these columns:

- `TABLE_ID`

  The table ID of the temporary table.

- `NAME`

  The name of the temporary table.

- `N_COLS`

  The number of columns in the temporary table. The number includes three hidden columns created by `InnoDB` (`DB_ROW_ID`, `DB_TRX_ID`, and `DB_ROLL_PTR`).

- `SPACE`

  The ID of the temporary tablespace where the temporary table resides.

Example

```
mysql> CREATE TEMPORARY TABLE t1 (c1 INT PRIMARY KEY) ENGINE=INNODB;
```

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO\G
*************************** 1. row ***************************
TABLE_ID: 97
    NAME: #sql8c88_43_0
  N_COLS: 4
   SPACE: 76
```

Notes

- This table is useful primarily for expert-level monitoring.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.39.29 The INFORMATION_SCHEMA INNODB_TRX Table

The `INNODB_TRX` table provides information about every transaction (excluding read-only transactions) currently executing inside `InnoDB`, including whether the transaction is waiting for a lock, when the transaction started, and the SQL statement the transaction is executing, if any.

For usage information, see Section 15.14.2.1, “Using InnoDB Transaction and Locking Information”.

The `INNODB_TRX` table has these columns:

- `TRX_ID`

  A unique transaction ID number, internal to `InnoDB`. These IDs are not created for transactions that are read only and nonlocking. For details, see Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”.

- `TRX_WEIGHT`

  The weight of a transaction, reflecting (but not necessarily the exact count of) the number of rows altered and the number of rows locked by the transaction. To resolve a deadlock, `InnoDB` selects the transaction with the smallest weight as the “victim” to roll back. Transactions that have changed nontransactional tables are considered heavier than others, regardless of the number of altered and locked rows.

- `TRX_STATE`

  The transaction execution state. Permitted values are `RUNNING`, `LOCK WAIT`, `ROLLING BACK`, and `COMMITTING`.

- `TRX_STARTED`

  The transaction start time.

- `TRX_REQUESTED_LOCK_ID`

  The ID of the lock the transaction is currently waiting for, if `TRX_STATE` is `LOCK WAIT`; otherwise `NULL`. To obtain details about the lock, join this column with the `ENGINE_LOCK_ID` column of the Performance Schema `data_locks` table.

- `TRX_WAIT_STARTED`

  The time when the transaction started waiting on the lock, if `TRX_STATE` is `LOCK WAIT`; otherwise `NULL`.

- `TRX_MYSQL_THREAD_ID`

  The MySQL thread ID. To obtain details about the thread, join this column with the `ID` column of the `INFORMATION_SCHEMA PROCESSLIST` table, but see Section 15.14.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”.

- `TRX_QUERY`

  The SQL statement that is being executed by the transaction.

- `TRX_OPERATION_STATE`

  The transaction's current operation, if any; otherwise `NULL`.

- `TRX_TABLES_IN_USE`

  The number of `InnoDB` tables used while processing the current SQL statement of this transaction.

- `TRX_TABLES_LOCKED`

  The number of `InnoDB` tables that the current SQL statement has row locks on. (Because these are row locks, not table locks, the tables can usually still be read from and written to by multiple transactions, despite some rows being locked.)

- `TRX_LOCK_STRUCTS`

  The number of locks reserved by the transaction.

- `TRX_LOCK_MEMORY_BYTES`

  The total size taken up by the lock structures of this transaction in memory.

- `TRX_ROWS_LOCKED`

  The approximate number or rows locked by this transaction. The value might include delete-marked rows that are physically present but not visible to the transaction.

- `TRX_ROWS_MODIFIED`

  The number of modified and inserted rows in this transaction.

- `TRX_CONCURRENCY_TICKETS`

  A value indicating how much work the current transaction can do before being swapped out, as specified by the `innodb_concurrency_tickets` system variable.

- `TRX_ISOLATION_LEVEL`

  The isolation level of the current transaction.

- `TRX_UNIQUE_CHECKS`

  Whether unique checks are turned on or off for the current transaction. For example, they might be turned off during a bulk data load.

- `TRX_FOREIGN_KEY_CHECKS`

  Whether foreign key checks are turned on or off for the current transaction. For example, they might be turned off during a bulk data load.

- TRX_LAST_FOREIGN_KEY_ERROR

  The detailed error message for the last foreign key error, if any; otherwise NULL.

- TRX_ADAPTIVE_HASH_LATCHED

  Whether the adaptive hash index is locked by the current transaction. When the adaptive hash index search system is partitioned, a single transaction does not lock the entire adaptive hash index. Adaptive hash index partitioning is controlled by innodb_adaptive_hash_index_parts, which is set to 8 by default.

- TRX_ADAPTIVE_HASH_TIMEOUT

  Whether to relinquish the search latch immediately for the adaptive hash index, or reserve it across calls from MySQL. When there is no adaptive hash index contention, this value remains zero and statements reserve the latch until they finish. During times of contention, it counts down to zero, and statements release the latch immediately after each row lookup. When the adaptive hash index search system is partitioned (controlled by innodb_adaptive_hash_index_parts), the value remains 0.

- TRX_IS_READ_ONLY

  A value of 1 indicates the transaction is read only.

- TRX_AUTOCOMMIT_NON_LOCKING

  A value of 1 indicates the transaction is a SELECT statement that does not use the FOR UPDATE or LOCK IN SHARED MODE clauses, and is executing with autocommit enabled so that the transaction contains only this one statement. When this column and TRX_IS_READ_ONLY are both 1, InnoDB optimizes the transaction to reduce the overhead associated with transactions that change table data.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TRX\G
*************************** 1. row ***************************
                    trx_id: 1510
                 trx_state: RUNNING
               trx_started: 2014-11-19 13:24:40
     trx_requested_lock_id: NULL
          trx_wait_started: NULL
                trx_weight: 586739
       trx_mysql_thread_id: 2
                 trx_query: DELETE FROM employees.salaries WHERE salary > 65000
       trx_operation_state: updating or deleting
         trx_tables_in_use: 1
         trx_tables_locked: 1
          trx_lock_structs: 3003
     trx_lock_memory_bytes: 450768
           trx_rows_locked: 1407513
         trx_rows_modified: 583736
   trx_concurrency_tickets: 0
       trx_isolation_level: REPEATABLE READ
         trx_unique_checks: 1
    trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
 trx_adaptive_hash_latched: 0
 trx_adaptive_hash_timeout: 10000
          trx_is_read_only: 0
trx_autocommit_non_locking: 0
```

Notes

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load. Its contents are updated as described in Section 15.14.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.39.30 The INFORMATION_SCHEMA INNODB_VIRTUAL Table

The `INNODB_VIRTUAL` table provides metadata about `InnoDB` virtual generated columns and columns upon which virtual generated columns are based.

A row appears in the `INNODB_VIRTUAL` table for each column upon which a virtual generated column is based.

The `INNODB_VIRTUAL` table has these columns:

- `TABLE_ID`

  An identifier representing the table associated with the virtual column; the same value as `INNODB_TABLES.TABLE_ID`.

- `POS`

  The position value of the virtual generated column. The value is large because it encodes the column sequence number and ordinal position. The formula used to calculate the value uses a bitwise operation:

  ```
  ((nth virtual generated column for the InnoDB instance + 1) << 16)
  + the ordinal position of the virtual generated column
  ```

  For example, if the first virtual generated column in the `InnoDB` instance is the third column of the table, the formula is `(0 + 1) << 16) + 2`. The first virtual generated column in the `InnoDB` instance is always number 0. As the third column in the table, the ordinal position of the virtual generated column is 2. Ordinal positions are counted from 0.

- `BASE_POS`

  The ordinal position of the columns upon which a virtual generated column is based.

Example

```
mysql> CREATE TABLE `t1` (
         `a` int(11) DEFAULT NULL,
         `b` int(11) DEFAULT NULL,
         `c` int(11) GENERATED ALWAYS AS (a+b) VIRTUAL,
         `h` varchar(10) DEFAULT NULL
       ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_VIRTUAL
       WHERE TABLE_ID IN
         (SELECT TABLE_ID FROM INFORMATION_SCHEMA.INNODB_TABLES
          WHERE NAME LIKE "test/t1");
+----------+-------+----------+
| TABLE_ID | POS   | BASE_POS |
+----------+-------+----------+
```

```
|       98 | 65538 |        0 |
|       98 | 65538 |        1 |
+----------+-------+----------+
```

Notes

- If a constant value is assigned to a virtual generated column, as in the following table, an entry for the column does not appear in the `INNODB_VIRTUAL` table. For an entry to appear, a virtual generated column must have a base column.

  ```
  CREATE TABLE `t1` (
    `a` int(11) DEFAULT NULL,
    `b` int(11) DEFAULT NULL,
    `c` int(11) GENERATED ALWAYS AS (5) VIRTUAL
  ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
  ```

  However, metadata for such a column does appear in the `INNODB_COLUMNS` table.

- You must have the `PROCESS` privilege to query this table.

- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

25.40 INFORMATION_SCHEMA Thread Pool Tables

Note

As of MySQL 8.0.14, the thread pool `INFORMATION_SCHEMA` tables are also available as Performance Schema tables. (See Section 26.12.15, “Performance Schema Thread Pool Tables”.) The `INFORMATION_SCHEMA` tables are deprecated and will be removed in a future MySQL version. Applications should transition away from the old tables to the new tables. For example, if an application uses this query:

```
SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_STATE;
```

The application should use this query instead:

```
SELECT * FROM performance_schema.tp_thread_state;
```

The following sections describe the `INFORMATION_SCHEMA` tables associated with the thread pool plugin (see Section 5.6.3, “MySQL Enterprise Thread Pool”). They provide information about thread pool operation:

- `TP_THREAD_GROUP_STATE`: Information about thread pool thread group states
- `TP_THREAD_GROUP_STATS`: Thread group statistics
- `TP_THREAD_STATE`: Information about thread pool thread states

Rows in these tables represent snapshots in time. In the case of `TP_THREAD_STATE`, all rows for a thread group comprise a snapshot in time. Thus, the MySQL server holds the mutex of the thread group while producing the snapshot. But it does not hold mutexes on all thread groups at the same time, to prevent a statement against `TP_THREAD_STATE` from blocking the entire MySQL server.

The `INFORMATION_SCHEMA` thread pool tables are implemented by individual plugins and the decision whether to load one can be made independently of the others (see Section 5.6.3.2, “Thread Pool Installation”). However, the content of all the tables depends on the thread pool plugin being enabled. If a

table plugin is enabled but the thread pool plugin is not, the table becomes visible and can be accessed but will be empty.

25.40.1 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table

Note

As of MySQL 8.0.14, the thread pool `INFORMATION_SCHEMA` tables are also available as Performance Schema tables. (See Section 26.12.15, “Performance Schema Thread Pool Tables”.) The `INFORMATION_SCHEMA` tables are deprecated and will be removed in a future MySQL version. Applications should transition away from the old tables to the new tables. For example, if an application uses this query:

```
SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_GROUP_STATE;
```

The application should use this query instead:

```
SELECT * FROM performance_schema.tp_thread_group_state;
```

The `TP_THREAD_GROUP_STATE` table has one row per thread group in the thread pool. Each row provides information about the current state of a group.

For descriptions of the columns in the `INFORMATION_SCHEMA TP_THREAD_GROUP_STATE` table, see Section 26.12.15.1, “The tp_thread_group_state Table”. The Performance Schema `tp_thread_group_state` table has equivalent columns.

25.40.2 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table

Note

As of MySQL 8.0.14, the thread pool `INFORMATION_SCHEMA` tables are also available as Performance Schema tables. (See Section 26.12.15, “Performance Schema Thread Pool Tables”.) The `INFORMATION_SCHEMA` tables are deprecated and will be removed in a future MySQL version. Applications should transition away from the old tables to the new tables. For example, if an application uses this query:

```
SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_GROUP_STATS;
```

The application should use this query instead:

```
SELECT * FROM performance_schema.tp_thread_group_stats;
```

The `TP_THREAD_GROUP_STATS` table reports statistics per thread group. There is one row per group.

For descriptions of the columns in the `INFORMATION_SCHEMA TP_THREAD_GROUP_STATS` table, see Section 26.12.15.2, “The tp_thread_group_stats Table”. The Performance Schema `tp_thread_group_stats` table has equivalent columns.

25.40.3 The INFORMATION_SCHEMA TP_THREAD_STATE Table

Note

As of MySQL 8.0.14, the thread pool `INFORMATION_SCHEMA` tables are also available as Performance Schema tables. (See Section 26.12.15, “Performance

> Schema Thread Pool Tables".) The `INFORMATION_SCHEMA` tables are deprecated and will be removed in a future MySQL version. Applications should transition away from the old tables to the new tables. For example, if an application uses this query:
>
> ```
> SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_STATE;
> ```
>
> The application should use this query instead:
>
> ```
> SELECT * FROM performance_schema.tp_thread_state;
> ```

The `TP_THREAD_STATE` table has one row per thread created by the thread pool to handle connections.

For descriptions of the columns in the `INFORMATION_SCHEMA TP_THREAD_STATE` table, see Section 26.12.15.3, "The tp_thread_state Table". The Performance Schema `tp_thread_state` table has equivalent columns.

25.41 INFORMATION_SCHEMA Connection-Control Tables

The following sections describe the `INFORMATION_SCHEMA` tables associated with the `CONNECTION_CONTROL` plugin.

25.41.1 The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table

This table provides information about the current number of consecutive failed connection attempts per client user/host combination.

`CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` has these columns:

- `USERHOST`

  The user/host combination of a client that has failed connection attempts, in *`'user_name'@'host_name'`* format.

- `FAILED_ATTEMPTS`

  The current number of consecutive failed connection attempts for the `USERHOST` value. This counts all failed attempts, regardless of whether they were delayed. The number of attempts for which the server added a delay to its response is the difference between the `FAILED_ATTEMPTS` value and the `connection_control_failed_connections_threshold` system variable value.

Notes

- The `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` plugin must be activated for this table to be available, and the `CONNECTION_CONTROL` plugin must be activated or the table contents will always be empty. See Section 6.4.2, "The Connection-Control Plugins".
- The table contains rows only for clients that have had one or more consecutive failed connection attempts without a subsequent successful attempt. When a client connects successfully, its failed-connection count is reset to zero and the server removes any row corresponding to the client.
- Assigning a value to the `connection_control_failed_connections_threshold` system variable at runtime resets all accumulated failed-connection counters to zero, which causes the table to become empty.

25.42 Extensions to SHOW Statements

Some extensions to `SHOW` statements accompany the implementation of `INFORMATION_SCHEMA`:

- `SHOW` can be used to get information about the structure of `INFORMATION_SCHEMA` itself.
- Several `SHOW` statements accept a `WHERE` clause that provides more flexibility in specifying which rows to display.

`INFORMATION_SCHEMA` is an information database, so its name is included in the output from `SHOW DATABASES`. Similarly, `SHOW TABLES` can be used with `INFORMATION_SCHEMA` to obtain a list of its tables:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+---------------------------------------+
| Tables_in_INFORMATION_SCHEMA          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| KEY_COLUMN_USAGE                      |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| REFERENTIAL_CONSTRAINTS               |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| STATISTICS                            |
| TABLES                                |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                      |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
| VIEWS                                 |
+---------------------------------------+
```

`SHOW COLUMNS` and `DESCRIBE` can display information about the columns in individual `INFORMATION_SCHEMA` tables.

`SHOW` statements that accept a `LIKE` clause to limit the rows displayed also permit a `WHERE` clause that specifies more general conditions that selected rows must satisfy:

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW INDEX
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW TRIGGERS
SHOW VARIABLES
```

The `WHERE` clause, if present, is evaluated against the column names displayed by the `SHOW` statement. For example, the `SHOW CHARACTER SET` statement produces these output columns:

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
...
```

To use a `WHERE` clause with `SHOW CHARACTER SET`, you would refer to those column names. As an example, the following statement displays information about character sets for which the default collation contains the string `'japanese'`:

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
+---------+---------------------------+---------------------+--------+
| Charset | Description               | Default collation   | Maxlen |
+---------+---------------------------+---------------------+--------+
| ujis    | EUC-JP Japanese           | ujis_japanese_ci    |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci    |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci   |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+---------------------+--------+
```

This statement displays the multibyte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
+---------+---------------------------+---------------------+--------+
| Charset | Description               | Default collation   | Maxlen |
+---------+---------------------------+---------------------+--------+
| big5    | Big5 Traditional Chinese  | big5_chinese_ci     |      2 |
| ujis    | EUC-JP Japanese           | ujis_japanese_ci    |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci    |      2 |
| euckr   | EUC-KR Korean             | euckr_korean_ci     |      2 |
| gb2312  | GB2312 Simplified Chinese | gb2312_chinese_ci   |      2 |
| gbk     | GBK Simplified Chinese    | gbk_chinese_ci      |      2 |
| utf8    | UTF-8 Unicode             | utf8_general_ci     |      3 |
| ucs2    | UCS-2 Unicode             | ucs2_general_ci     |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci   |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+---------------------+--------+
```

Chapter 26 MySQL Performance Schema

Table of Contents

26.1 Performance Schema Quick Start 4479
26.2 Performance Schema Build Configuration 4485
26.3 Performance Schema Startup Configuration 4486
26.4 Performance Schema Runtime Configuration 4488
26.4.1 Performance Schema Event Timing 4489
26.4.2 Performance Schema Event Filtering 4491
26.4.3 Event Pre-Filtering 4493
26.4.4 Pre-Filtering by Instrument 4494
26.4.5 Pre-Filtering by Object 4495
26.4.6 Pre-Filtering by Thread 4497
26.4.7 Pre-Filtering by Consumer 4499
26.4.8 Example Consumer Configurations 4502
26.4.9 Naming Instruments or Consumers for Filtering Operations 4507
26.4.10 Determining What Is Instrumented 4508
26.5 Performance Schema Queries 4509
26.6 Performance Schema Instrument Naming Conventions 4509
26.7 Performance Schema Status Monitoring 4513
26.8 Performance Schema Atom and Molecule Events 4517
26.9 Performance Schema Tables for Current and Historical Events 4517
26.10 Performance Schema Statement Digests and Sampling 4518
26.11 Performance Schema General Table Characteristics 4523
26.12 Performance Schema Table Descriptions 4524
26.12.1 Performance Schema Table Index 4525
26.12.2 Performance Schema Setup Tables 4528
26.12.3 Performance Schema Instance Tables 4536
26.12.4 Performance Schema Wait Event Tables 4542
26.12.5 Performance Schema Stage Event Tables 4547
26.12.6 Performance Schema Statement Event Tables 4553
26.12.7 Performance Schema Transaction Tables 4564
26.12.8 Performance Schema Connection Tables 4572
26.12.9 Performance Schema Connection Attribute Tables 4575
26.12.10 Performance Schema User-Defined Variable Tables 4579
26.12.11 Performance Schema Replication Tables 4580
26.12.12 Performance Schema Lock Tables 4598
26.12.13 Performance Schema System Variable Tables 4607
26.12.14 Performance Schema Status Variable Tables 4612
26.12.15 Performance Schema Thread Pool Tables 4613
26.12.16 Performance Schema Clone Tables 4619
26.12.17 Performance Schema Summary Tables 4621
26.12.18 Performance Schema Miscellaneous Tables 4650
26.13 Performance Schema Option and Variable Reference 4662
26.14 Performance Schema Command Options 4665
26.15 Performance Schema System Variables 4666
26.16 Performance Schema Status Variables 4684
26.17 The Performance Schema Memory-Allocation Model 4687
26.18 Performance Schema and Plugins 4688
26.19 Using the Performance Schema to Diagnose Problems 4688
26.19.1 Query Profiling Using Performance Schema 4690

26.19.2 Obtaining Parent Event Information 4692
26.20 Restrictions on Performance Schema 4693

The MySQL Performance Schema is a feature for monitoring MySQL Server execution at a low level. The Performance Schema has these characteristics:

- The Performance Schema provides a way to inspect internal execution of the server at runtime. It is implemented using the `PERFORMANCE_SCHEMA` storage engine and the `performance_schema` database. The Performance Schema focuses primarily on performance data. This differs from `INFORMATION_SCHEMA`, which serves for inspection of metadata.
- The Performance Schema monitors server events. An “event” is anything the server does that takes time and has been instrumented so that timing information can be collected. In general, an event could be a function call, a wait for the operating system, a stage of an SQL statement execution such as parsing or sorting, or an entire statement or group of statements. Event collection provides access to information about synchronization calls (such as for mutexes) file and table I/O, table locks, and so forth for the server and for several storage engines.
- Performance Schema events are distinct from events written to the server's binary log (which describe data modifications) and Event Scheduler events (which are a type of stored program).
- Performance Schema events are specific to a given instance of the MySQL Server. Performance Schema tables are considered local to the server, and changes to them are not replicated or written to the binary log.
- Current events are available, as well as event histories and summaries. This enables you to determine how many times instrumented activities were performed and how much time they took. Event information is available to show the activities of specific threads, or activity associated with particular objects such as a mutex or file.
- The `PERFORMANCE_SCHEMA` storage engine collects event data using “instrumentation points” in server source code.
- Collected events are stored in tables in the `performance_schema` database. These tables can be queried using `SELECT` statements like other tables.
- Performance Schema configuration can be modified dynamically by updating tables in the `performance_schema` database through SQL statements. Configuration changes affect data collection immediately.
- Tables in the Performance Schema are in-memory tables that use no persistent on-disk storage. The contents are repopulated beginning at server startup and discarded at server shutdown.
- Monitoring is available on all platforms supported by MySQL.

  Some limitations might apply: The types of timers might vary per platform. Instruments that apply to storage engines might not be implemented for all storage engines. Instrumentation of each third-party engine is the responsibility of the engine maintainer. See also Section 26.20, “Restrictions on Performance Schema”.
- Data collection is implemented by modifying the server source code to add instrumentation. There are no separate threads associated with the Performance Schema, unlike other features such as replication or the Event Scheduler.

The Performance Schema is intended to provide access to useful information about server execution while having minimal impact on server performance. The implementation follows these design goals:

- Activating the Performance Schema causes no changes in server behavior. For example, it does not cause thread scheduling to change, and it does not cause query execution plans (as shown by `EXPLAIN`) to change.
- Server monitoring occurs continuously and unobtrusively with very little overhead. Activating the Performance Schema does not make the server unusable.
- The parser is unchanged. There are no new keywords or statements.
- Execution of server code proceeds normally even if the Performance Schema fails internally.
- When there is a choice between performing processing during event collection initially or during event retrieval later, priority is given to making collection faster. This is because collection is ongoing whereas retrieval is on demand and might never happen at all.
- Most Performance Schema tables have indexes, which gives the optimizer access to execution plans other than full table scans. For more information, see Section 8.2.4, “Optimizing Performance Schema Queries”.
- It is easy to add new instrumentation points.
- Instrumentation is versioned. If the instrumentation implementation changes, previously instrumented code will continue to work. This benefits developers of third-party plugins because it is not necessary to upgrade each plugin to stay synchronized with the latest Performance Schema changes.

Note

The MySQL `sys` schema is a set of objects that provides convenient access to data collected by the Performance Schema. The `sys` schema is installed by default. For usage instructions, see Chapter 27, *MySQL sys Schema*.

26.1 Performance Schema Quick Start

This section briefly introduces the Performance Schema with examples that show how to use it. For additional examples, see Section 26.19, “Using the Performance Schema to Diagnose Problems”.

The Performance Schema is enabled by default. To enable or disable it explicitly, start the server with the `performance_schema` variable set to an appropriate value. For example, use these lines in the server `my.cnf` file:

```
[mysqld]
performance_schema=ON
```

When the server starts, it sees `performance_schema` and attempts to initialize the Performance Schema. To verify successful initialization, use this statement:

```
mysql> SHOW VARIABLES LIKE 'performance_schema';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| performance_schema | ON    |
+--------------------+-------+
```

A value of `ON` means that the Performance Schema initialized successfully and is ready for use. A value of `OFF` means that some error occurred. Check the server error log for information about what went wrong.

The Performance Schema is implemented as a storage engine, so you will see it listed in the output from the `INFORMATION_SCHEMA.ENGINES` table or the `SHOW ENGINES` statement:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.ENGINES
       WHERE ENGINE='PERFORMANCE_SCHEMA'\G
*************************** 1. row ***************************
      ENGINE: PERFORMANCE_SCHEMA
     SUPPORT: YES
     COMMENT: Performance Schema
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO

mysql> SHOW ENGINES\G
...
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
...
```

The `PERFORMANCE_SCHEMA` storage engine operates on tables in the `performance_schema` database. You can make `performance_schema` the default database so that references to its tables need not be qualified with the database name:

```
mysql> USE performance_schema;
```

Performance Schema tables are stored in the `performance_schema` database. Information about the structure of this database and its tables can be obtained, as for any other database, by selecting from the `INFORMATION_SCHEMA` database or by using `SHOW` statements. For example, use either of these statements to see what Performance Schema tables exist:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
       WHERE TABLE_SCHEMA = 'performance_schema';
+-----------------------------------------------------+
| TABLE_NAME                                          |
+-----------------------------------------------------+
| accounts                                            |
| cond_instances                                      |
...
| events_stages_current                               |
| events_stages_history                               |
| events_stages_history_long                          |
| events_stages_summary_by_account_by_event_name      |
| events_stages_summary_by_host_by_event_name         |
| events_stages_summary_by_thread_by_event_name       |
| events_stages_summary_by_user_by_event_name         |
| events_stages_summary_global_by_event_name          |
| events_statements_current                           |
| events_statements_history                           |
| events_statements_history_long                      |
...
| file_instances                                      |
| file_summary_by_event_name                          |
| file_summary_by_instance                            |
| host_cache                                          |
| hosts                                               |
| memory_summary_by_account_by_event_name             |
| memory_summary_by_host_by_event_name                |
| memory_summary_by_thread_by_event_name              |
| memory_summary_by_user_by_event_name                |
```

```
| memory_summary_global_by_event_name                    |
| metadata_locks                                         |
| mutex_instances                                        |
| objects_summary_global_by_type                         |
| performance_timers                                     |
| replication_connection_configuration                   |
| replication_connection_status                          |
| replication_applier_configuration                      |
| replication_applier_status                             |
| replication_applier_status_by_coordinator              |
| replication_applier_status_by_worker                   |
| rwlock_instances                                       |
| session_account_connect_attrs                          |
| session_connect_attrs                                  |
| setup_actors                                           |
| setup_consumers                                        |
| setup_instruments                                      |
| setup_objects                                          |
| socket_instances                                       |
| socket_summary_by_event_name                           |
| socket_summary_by_instance                             |
| table_handles                                          |
| table_io_waits_summary_by_index_usage                  |
| table_io_waits_summary_by_table                        |
| table_lock_waits_summary_by_table                      |
| threads                                                |
| users                                                  |
+--------------------------------------------------------+

mysql> SHOW TABLES FROM performance_schema;
+--------------------------------------------------------+
| Tables_in_performance_schema                           |
+--------------------------------------------------------+
| accounts                                               |
| cond_instances                                         |
| events_stages_current                                  |
| events_stages_history                                  |
| events_stages_history_long                             |
...
```

The number of Performance Schema tables increases over time as implementation of additional instrumentation proceeds.

The name of the `performance_schema` database is lowercase, as are the names of tables within it. Queries should specify the names in lowercase.

To see the structure of individual tables, use `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE performance_schema.setup_consumers\G
*************************** 1. row ***************************
       Table: setup_consumers
Create Table: CREATE TABLE `setup_consumers` (
  `NAME` varchar(64) NOT NULL,
  `ENABLED` enum('YES','NO') NOT NULL,
  PRIMARY KEY (`NAME`)
) ENGINE=PERFORMANCE_SCHEMA DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Table structure is also available by selecting from tables such as `INFORMATION_SCHEMA.COLUMNS` or by using statements such as `SHOW COLUMNS`.

Tables in the `performance_schema` database can be grouped according to the type of information in them: Current events, event histories and summaries, object instances, and setup (configuration) information. The following examples illustrate a few uses for these tables. For detailed information about the tables in each group, see Section 26.12, “Performance Schema Table Descriptions”.

Initially, not all instruments and consumers are enabled, so the performance schema does not collect all events. To turn all of these on and enable event timing, execute two statements (the row counts may differ depending on MySQL version):

```
mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'YES', TIMED = 'YES';
Query OK, 560 rows affected (0.04 sec)
mysql> UPDATE performance_schema.setup_consumers
       SET ENABLED = 'YES';
Query OK, 10 rows affected (0.00 sec)
```

To see what the server is doing at the moment, examine the `events_waits_current` table. It contains one row per thread showing each thread's most recent monitored event:

```
mysql> SELECT *
       FROM performance_schema.events_waits_current\G
*************************** 1. row ***************************
            THREAD_ID: 0
             EVENT_ID: 5523
         END_EVENT_ID: 5523
           EVENT_NAME: wait/synch/mutex/mysys/THR_LOCK::mutex
               SOURCE: thr_lock.c:525
          TIMER_START: 201660494489586
            TIMER_END: 201660494576112
           TIMER_WAIT: 86526
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 142270668
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: 0
...
```

This event indicates that thread 0 was waiting for 86,526 picoseconds to acquire a lock on `THR_LOCK::mutex`, a mutex in the `mysys` subsystem. The first few columns provide the following information:

- The ID columns indicate which thread the event comes from and the event number.
- `EVENT_NAME` indicates what was instrumented and `SOURCE` indicates which source file contains the instrumented code.
- The timer columns show when the event started and stopped and how long it took. If an event is still in progress, the `TIMER_END` and `TIMER_WAIT` values are `NULL`. Timer values are approximate and expressed in picoseconds. For information about timers and event time collection, see Section 26.4.1, “Performance Schema Event Timing”.

The history tables contain the same kind of rows as the current-events table but have more rows and show what the server has been doing “recently” rather than “currently.” The `events_waits_history` and `events_waits_history_long` tables contain the most recent 10 events per thread and most recent 10,000 events, respectively. For example, to see information for recent events produced by thread 13, do this:

```
mysql> SELECT EVENT_ID, EVENT_NAME, TIMER_WAIT
```

```
       FROM performance_schema.events_waits_history
       WHERE THREAD_ID = 13
       ORDER BY EVENT_ID;
+----------+----------------------------------------+------------+
| EVENT_ID | EVENT_NAME                             | TIMER_WAIT |
+----------+----------------------------------------+------------+
|       86 | wait/synch/mutex/mysys/THR_LOCK::mutex |     686322 |
|       87 | wait/synch/mutex/mysys/THR_LOCK_malloc |     320535 |
|       88 | wait/synch/mutex/mysys/THR_LOCK_malloc |     339390 |
|       89 | wait/synch/mutex/mysys/THR_LOCK_malloc |     377100 |
|       90 | wait/synch/mutex/sql/LOCK_plugin       |     614673 |
|       91 | wait/synch/mutex/sql/LOCK_open         |     659925 |
|       92 | wait/synch/mutex/sql/THD::LOCK_thd_data |     494001 |
|       93 | wait/synch/mutex/mysys/THR_LOCK_malloc |     222489 |
|       94 | wait/synch/mutex/mysys/THR_LOCK_malloc |     214947 |
|       95 | wait/synch/mutex/mysys/LOCK_alarm      |     312993 |
+----------+----------------------------------------+------------+
```

As new events are added to a history table, older events are discarded if the table is full.

Summary tables provide aggregated information for all events over time. The tables in this group summarize event data in different ways. To see which instruments have been executed the most times or have taken the most wait time, sort the `events_waits_summary_global_by_event_name` table on the `COUNT_STAR` or `SUM_TIMER_WAIT` column, which correspond to a `COUNT(*)` or `SUM(TIMER_WAIT)` value, respectively, calculated over all events:

```
mysql> SELECT EVENT_NAME, COUNT_STAR
       FROM performance_schema.events_waits_summary_global_by_event_name
       ORDER BY COUNT_STAR DESC LIMIT 10;
+---------------------------------------------------+------------+
| EVENT_NAME                                        | COUNT_STAR |
+---------------------------------------------------+------------+
| wait/synch/mutex/mysys/THR_LOCK_malloc            |       6419 |
| wait/io/file/sql/FRM                              |        452 |
| wait/synch/mutex/sql/LOCK_plugin                  |        337 |
| wait/synch/mutex/mysys/THR_LOCK_open              |        187 |
| wait/synch/mutex/mysys/LOCK_alarm                 |        147 |
| wait/synch/mutex/sql/THD::LOCK_thd_data           |        115 |
| wait/io/file/myisam/kfile                         |        102 |
| wait/synch/mutex/sql/LOCK_global_system_variables |         89 |
| wait/synch/mutex/mysys/THR_LOCK::mutex            |         89 |
| wait/synch/mutex/sql/LOCK_open                    |         88 |
+---------------------------------------------------+------------+

mysql> SELECT EVENT_NAME, SUM_TIMER_WAIT
       FROM performance_schema.events_waits_summary_global_by_event_name
       ORDER BY SUM_TIMER_WAIT DESC LIMIT 10;
+----------------------------------------+----------------+
| EVENT_NAME                             | SUM_TIMER_WAIT |
+----------------------------------------+----------------+
| wait/io/file/sql/MYSQL_LOG             |     1599816582 |
| wait/synch/mutex/mysys/THR_LOCK_malloc |     1530083250 |
| wait/io/file/sql/binlog_index          |     1385291934 |
| wait/io/file/sql/FRM                   |     1292823243 |
| wait/io/file/myisam/kfile              |      411193611 |
| wait/io/file/myisam/dfile              |      322401645 |
| wait/synch/mutex/mysys/LOCK_alarm      |      145126935 |
| wait/io/file/sql/casetest              |      104324715 |
| wait/synch/mutex/sql/LOCK_plugin       |       86027823 |
| wait/io/file/sql/pid                   |       72591750 |
+----------------------------------------+----------------+
```

These results show that the `THR_LOCK_malloc` mutex is “hot,” both in terms of how often it is used and amount of time that threads wait attempting to acquire it.

Note

The `THR_LOCK_malloc` mutex is used only in debug builds. In production builds it is not hot because it is nonexistent.

Instance tables document what types of objects are instrumented. An instrumented object, when used by the server, produces an event. These tables provide event names and explanatory notes or status information. For example, the `file_instances` table lists instances of instruments for file I/O operations and their associated files:

```
mysql> SELECT *
       FROM performance_schema.file_instances\G
*************************** 1. row ***************************
 FILE_NAME: /opt/mysql-log/60500/binlog.000007
EVENT_NAME: wait/io/file/sql/binlog
OPEN_COUNT: 0
*************************** 2. row ***************************
 FILE_NAME: /opt/mysql/60500/data/mysql/tables_priv.MYI
EVENT_NAME: wait/io/file/myisam/kfile
OPEN_COUNT: 1
*************************** 3. row ***************************
 FILE_NAME: /opt/mysql/60500/data/mysql/columns_priv.MYI
EVENT_NAME: wait/io/file/myisam/kfile
OPEN_COUNT: 1
...
```

Setup tables are used to configure and display monitoring characteristics. For example, `setup_instruments` lists the set of instruments for which events can be collected and shows which of them are enabled:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments;
+----------------------------------------------------+---------+-------+
| NAME                                               | ENABLED | TIMED |
+----------------------------------------------------+---------+-------+
...
| stage/sql/end                                      | NO      | NO    |
| stage/sql/executing                                | NO      | NO    |
| stage/sql/init                                     | NO      | NO    |
| stage/sql/insert                                   | NO      | NO    |
...
| statement/sql/load                                 | YES     | YES   |
| statement/sql/grant                                | YES     | YES   |
| statement/sql/check                                | YES     | YES   |
| statement/sql/flush                                | YES     | YES   |
...
| wait/synch/mutex/sql/LOCK_global_read_lock         | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables  | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db                  | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                  | YES     | YES   |
...
| wait/synch/rwlock/sql/LOCK_grant                   | YES     | YES   |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger          | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_connect        | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_slave          | YES     | YES   |
...
| wait/io/file/sql/binlog                            | YES     | YES   |
| wait/io/file/sql/binlog_index                      | YES     | YES   |
| wait/io/file/sql/casetest                          | YES     | YES   |
| wait/io/file/sql/dbopt                             | YES     | YES   |
...
```

To understand how to interpret instrument names, see Section 26.6, “Performance Schema Instrument Naming Conventions”.

To control whether events are collected for an instrument, set its `ENABLED` value to `YES` or `NO`. For example:

```
mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'NO'
       WHERE NAME = 'wait/synch/mutex/sql/LOCK_mysql_create_db';
```

The Performance Schema uses collected events to update tables in the `performance_schema` database, which act as “consumers” of event information. The `setup_consumers` table lists the available consumers and which are enabled:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | YES     |
| events_statements_history_long | NO      |
| events_transactions_current    | YES     |
| events_transactions_history    | YES     |
| events_transactions_history_long | NO    |
| events_waits_current           | NO      |
| events_waits_history           | NO      |
| events_waits_history_long      | NO      |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
```

To control whether the Performance Schema maintains a consumer as a destination for event information, set its `ENABLED` value.

For more information about the setup tables and how to use them to control event collection, see Section 26.4.2, “Performance Schema Event Filtering”.

There are some miscellaneous tables that do not fall into any of the previous groups. For example, `performance_timers` lists the available event timers and their characteristics. For information about timers, see Section 26.4.1, “Performance Schema Event Timing”.

26.2 Performance Schema Build Configuration

The Performance Schema is mandatory and always compiled in. It is possible to exclude certain parts of the Performance Schema instrumentation. For example, to exclude stage and statement instrumentation, do this:

```
shell> cmake . \
        -DDISABLE_PSI_STAGE=1 \
        -DDISABLE_PSI_STATEMENT=1
```

For more information, see the descriptions of the `DISABLE_PSI_`*`XXX`* `CMake` options in Section 2.9.7, “MySQL Source-Configuration Options”.

If you install MySQL over a previous installation that was configured without the Performance Schema (or with an older version of the Performance Schema that has missing or out-of-date tables). One indication of this issue is the presence of messages such as the following in the error log:

```
[ERROR] Native table 'performance_schema'.'events_waits_history'
has the wrong structure
[ERROR] Native table 'performance_schema'.'events_waits_history_long'
has the wrong structure
...
```

To correct that problem, perform the MySQL upgrade procedure. See Section 2.11, “Upgrading MySQL”.

Because the Performance Schema is configured into the server at build time, a row for `PERFORMANCE_SCHEMA` appears in the output from `SHOW ENGINES`. This means that the Performance Schema is available, not that it is enabled. To enable it, you must do so at server startup, as described in the next section.

26.3 Performance Schema Startup Configuration

To use the MySQL Performance Schema, it must be enabled at server startup to enable event collection to occur.

The Performance Schema is enabled by default. To enable or disable it explicitly, start the server with the `performance_schema` variable set to an appropriate value. For example, use these lines in the server `my.cnf` file:

```
[mysqld]
performance_schema=ON
```

If the server is unable to allocate any internal buffer during Performance Schema initialization, the Performance Schema disables itself and sets `performance_schema` to `OFF`, and the server runs without instrumentation.

The Performance Schema also permits instrument and consumer configuration at server startup.

To control an instrument at server startup, use an option of this form:

```
--performance-schema-instrument='instrument_name=value'
```

Here, *`instrument_name`* is an instrument name such as `wait/synch/mutex/sql/LOCK_open`, and *`value`* is one of these values:

- `OFF`, `FALSE`, or `0`: Disable the instrument
- `ON`, `TRUE`, or `1`: Enable and time the instrument
- `COUNTED`: Enable and count (rather than time) the instrument

Each `--performance-schema-instrument` option can specify only one instrument name, but multiple instances of the option can be given to configure multiple instruments. In addition, patterns are permitted in instrument names to configure instruments that match the pattern. To configure all condition synchronization instruments as enabled and counted, use this option:

```
--performance-schema-instrument='wait/synch/cond/%=COUNTED'
```

To disable all instruments, use this option:

```
--performance-schema-instrument='%=OFF'
```

Exception: The `memory/performance_schema/%` instruments are built in and cannot be disabled at startup.

Longer instrument name strings take precedence over shorter pattern names, regardless of order. For information about specifying patterns to select instruments, see Section 26.4.9, “Naming Instruments or Consumers for Filtering Operations”.

An unrecognized instrument name is ignored. It is possible that a plugin installed later may create the instrument, at which time the name is recognized and configured.

To control a consumer at server startup, use an option of this form:

```
--performance-schema-consumer-consumer_name=value
```

Here, *`consumer_name`* is a consumer name such as `events_waits_history`, and *`value`* is one of these values:

- `OFF`, `FALSE`, or `0`: Do not collect events for the consumer
- `ON`, `TRUE`, or `1`: Collect events for the consumer

For example, to enable the `events_waits_history` consumer, use this option:

```
--performance-schema-consumer-events-waits-history=ON
```

The permitted consumer names can be found by examining the `setup_consumers` table. Patterns are not permitted. Consumer names in the `setup_consumers` table use underscores, but for consumers set at startup, dashes and underscores within the name are equivalent.

The Performance Schema includes several system variables that provide configuration information:

```
mysql> SHOW VARIABLES LIKE 'perf%';
+----------------------------------------------------+---------+
| Variable_name                                      | Value   |
+----------------------------------------------------+---------+
| performance_schema                                 | ON      |
| performance_schema_accounts_size                   | 100     |
| performance_schema_digests_size                    | 200     |
| performance_schema_events_stages_history_long_size | 10000   |
| performance_schema_events_stages_history_size      | 10      |
| performance_schema_events_statements_history_long_size | 10000   |
| performance_schema_events_statements_history_size  | 10      |
| performance_schema_events_waits_history_long_size  | 10000   |
| performance_schema_events_waits_history_size       | 10      |
| performance_schema_hosts_size                      | 100     |
| performance_schema_max_cond_classes                | 80      |
| performance_schema_max_cond_instances              | 1000    |
...
```

The `performance_schema` variable is `ON` or `OFF` to indicate whether the Performance Schema is enabled or disabled. The other variables indicate table sizes (number of rows) or memory allocation values.

Note

With the Performance Schema enabled, the number of Performance Schema instances affects the server memory footprint, perhaps to a large extent. The

> Performance Schema autoscales many parameters to use memory only as required; see Section 26.17, “The Performance Schema Memory-Allocation Model”.

To change the value of Performance Schema system variables, set them at server startup. For example, put the following lines in a `my.cnf` file to change the sizes of the history tables for wait events:

```
[mysqld]
performance_schema
performance_schema_events_waits_history_size=20
performance_schema_events_waits_history_long_size=15000
```

The Performance Schema automatically sizes the values of several of its parameters at server startup if they are not set explicitly. For example, it sizes the parameters that control the sizes of the events waits tables this way. The Performance Schema allocates memory incrementally, scaling its memory use to actual server load, instead of allocating all the memory it needs during server startup. Consequently, many sizing parameters need not be set at all. To see which parameters are autosized or autoscaled, use `mysqld --verbose --help` and examine the option descriptions, or see Section 26.15, “Performance Schema System Variables”.

For each autosized parameter that is not set at server startup, the Performance Schema determines how to set its value based on the value of the following system values, which are considered as “hints” about how you have configured your MySQL server:

```
max_connections
open_files_limit
table_definition_cache
table_open_cache
```

To override autosizing or autoscaling for a given parameter, set it to a value other than −1 at startup. In this case, the Performance Schema assigns it the specified value.

At runtime, `SHOW VARIABLES` displays the actual values that autosized parameters were set to. Autoscaled parameters display with a value of −1.

If the Performance Schema is disabled, its autosized and autoscaled parameters remain set to −1 and `SHOW VARIABLES` displays −1.

26.4 Performance Schema Runtime Configuration

Specific Performance Schema features can be enabled at runtime to control which types of event collection occur.

Performance Schema setup tables contain information about monitoring configuration:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
       WHERE TABLE_SCHEMA = 'performance_schema'
       AND TABLE_NAME LIKE 'setup%';
+-------------------+
| TABLE_NAME        |
+-------------------+
| setup_actors      |
| setup_consumers   |
| setup_instruments |
| setup_objects     |
| setup_threads     |
+-------------------+
```

You can examine the contents of these tables to obtain information about Performance Schema monitoring characteristics. If you have the `UPDATE` privilege, you can change Performance Schema operation by modifying setup tables to affect how monitoring occurs. For additional details about these tables, see Section 26.12.2, “Performance Schema Setup Tables”.

The `setup_instruments` and `setup_consumers` tables list the instruments for which events can be collected and the types of consumers for which event information actually is collected, respectively. Other setup tables enable further modification of the monitoring configuration. Section 26.4.2, “Performance Schema Event Filtering”, discusses how you can modify these tables to affect event collection.

If there are Performance Schema configuration changes that must be made at runtime using SQL statements and you would like these changes to take effect each time the server starts, put the statements in a file and start the server with the `init_file` system variable set to name the file. This strategy can also be useful if you have multiple monitoring configurations, each tailored to produce a different kind of monitoring, such as casual server health monitoring, incident investigation, application behavior troubleshooting, and so forth. Put the statements for each monitoring configuration into their own file and specify the appropriate file as the `init_file` value when you start the server.

26.4.1 Performance Schema Event Timing

Events are collected by means of instrumentation added to the server source code. Instruments time events, which is how the Performance Schema provides an idea of how long events take. It is also possible to configure instruments not to collect timing information. This section discusses the available timers and their characteristics, and how timing values are represented in events.

Performance Schema Timers

Performance Schema timers vary in precision and amount of overhead. To see what timers are available and their characteristics, check the `performance_timers` table:

```
mysql> SELECT * FROM performance_schema.performance_timers;
+-------------+-----------------+------------------+----------------+
| TIMER_NAME  | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-------------+-----------------+------------------+----------------+
| CYCLE       |      2389029850 |                1 |             72 |
| NANOSECOND  |      1000000000 |                1 |            112 |
| MICROSECOND |         1000000 |                1 |            136 |
| MILLISECOND |            1036 |                1 |            168 |
+-------------+-----------------+------------------+----------------+
```

If the values associated with a given timer name are `NULL`, that timer is not supported on your platform.

The columns have these meanings:

- The `TIMER_NAME` column shows the names of the available timers. `CYCLE` refers to the timer that is based on the CPU (processor) cycle counter.
- `TIMER_FREQUENCY` indicates the number of timer units per second. For a cycle timer, the frequency is generally related to the CPU speed. The value shown was obtained on a system with a 2.4GHz processor. The other timers are based on fixed fractions of seconds.
- `TIMER_RESOLUTION` indicates the number of timer units by which timer values increase at a time. If a timer has a resolution of 10, its value increases by 10 each time.
- `TIMER_OVERHEAD` is the minimal number of cycles of overhead to obtain one timing with the given timer. The overhead per event is twice the value displayed because the timer is invoked at the beginning and end of the event.

The Performance Schema assigns timers as follows:

- The wait timer uses `CYCLE`.
- The idle, stage, statement, and transaction timers use `NANOSECOND` on platforms where the `NANOSECOND` timer is available, `MICROSECOND` otherwise.

At server startup, the Performance Schema verifies that assumptions made at build time about timer assignments are correct, and displays a warning if a timer is not available.

To time wait events, the most important criterion is to reduce overhead, at the possible expense of the timer accuracy, so using the `CYCLE` timer is the best.

The time a statement (or stage) takes to execute is in general orders of magnitude larger than the time it takes to execute a single wait. To time statements, the most important criterion is to have an accurate measure, which is not affected by changes in processor frequency, so using a timer which is not based on cycles is the best. The default timer for statements is `NANOSECOND`. The extra “overhead” compared to the `CYCLE` timer is not significant, because the overhead caused by calling a timer twice (once when the statement starts, once when it ends) is orders of magnitude less compared to the CPU time used to execute the statement itself. Using the `CYCLE` timer has no benefit here, only drawbacks.

The precision offered by the cycle counter depends on processor speed. If the processor runs at 1 GHz (one billion cycles/second) or higher, the cycle counter delivers sub-nanosecond precision. Using the cycle counter is much cheaper than getting the actual time of day. For example, the standard `gettimeofday()` function can take hundreds of cycles, which is an unacceptable overhead for data gathering that may occur thousands or millions of times per second.

Cycle counters also have disadvantages:

- End users expect to see timings in wall-clock units, such as fractions of a second. Converting from cycles to fractions of seconds can be expensive. For this reason, the conversion is a quick and fairly rough multiplication operation.
- Processor cycle rate might change, such as when a laptop goes into power-saving mode or when a CPU slows down to reduce heat generation. If a processor's cycle rate fluctuates, conversion from cycles to real-time units is subject to error.
- Cycle counters might be unreliable or unavailable depending on the processor or the operating system. For example, on Pentiums, the instruction is `RDTSC` (an assembly-language rather than a C instruction) and it is theoretically possible for the operating system to prevent user-mode programs from using it.
- Some processor details related to out-of-order execution or multiprocessor synchronization might cause the counter to seem fast or slow by up to 1000 cycles.

MySQL works with cycle counters on x386 (Windows, macOS, Linux, Solaris, and other Unix flavors), PowerPC, and IA-64.

Performance Schema Timer Representation in Events

Rows in Performance Schema tables that store current events and historical events have three columns to represent timing information: `TIMER_START` and `TIMER_END` indicate when an event started and finished, and `TIMER_WAIT` indicates event duration.

The `setup_instruments` table has an `ENABLED` column to indicate the instruments for which to collect events. The table also has a `TIMED` column to indicate which instruments are timed. If an instrument is not enabled, it produces no events. If an enabled instrument is not timed, events produced by the instrument have `NULL` for the `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` timer values. This in turn causes

those values to be ignored when calculating aggregate time values in summary tables (sum, minimum, maximum, and average).

Internally, times within events are stored in units given by the timer in effect when event timing begins. For display when events are retrieved from Performance Schema tables, times are shown in picoseconds (trillionths of a second) to normalize them to a standard unit, regardless of which timer is selected.

The timer baseline (“time zero”) occurs at Performance Schema initialization during server startup. `TIMER_START` and `TIMER_END` values in events represent picoseconds since the baseline. `TIMER_WAIT` values are durations in picoseconds.

Picosecond values in events are approximate. Their accuracy is subject to the usual forms of error associated with conversion from one unit to another. If the `CYCLE` timer is used and the processor rate varies, there might be drift. For these reasons, it is not reasonable to look at the `TIMER_START` value for an event as an accurate measure of time elapsed since server startup. On the other hand, it is reasonable to use `TIMER_START` or `TIMER_WAIT` values in `ORDER BY` clauses to order events by start time or duration.

The choice of picoseconds in events rather than a value such as microseconds has a performance basis. One implementation goal was to show results in a uniform time unit, regardless of the timer. In an ideal world this time unit would look like a wall-clock unit and be reasonably precise; in other words, microseconds. But to convert cycles or nanoseconds to microseconds, it would be necessary to perform a division for every instrumentation. Division is expensive on many platforms. Multiplication is not expensive, so that is what is used. Therefore, the time unit is an integer multiple of the highest possible `TIMER_FREQUENCY` value, using a multiplier large enough to ensure that there is no major precision loss. The result is that the time unit is “picoseconds.” This precision is spurious, but the decision enables overhead to be minimized.

While a wait, stage, statement, or transaction event is executing, the respective current-event tables display current-event timing information:

```
events_waits_current
events_stages_current
events_statements_current
events_transactions_current
```

To make it possible to determine how long a not-yet-completed event has been running, the timer columns are set as follows:

- `TIMER_START` is populated.
- `TIMER_END` is populated with the current timer value.
- `TIMER_WAIT` is populated with the time elapsed so far (`TIMER_END` − `TIMER_START`).

Events that have not yet completed have an `END_EVENT_ID` value of `NULL`. To assess time elapsed so far for an event, use the `TIMER_WAIT` column. Therefore, to identify events that have not yet completed and have taken longer than *`N`* picoseconds thus far, monitoring applications can use this expression in queries:

```
WHERE END_EVENT_ID IS NULL AND TIMER_WAIT > N
```

Event identification as just described assumes that the corresponding instruments have `ENABLED` and `TIMED` set to `YES` and that the relevant consumers are enabled.

26.4.2 Performance Schema Event Filtering

Events are processed in a producer/consumer fashion:

- Instrumented code is the source for events and produces events to be collected. The `setup_instruments` table lists the instruments for which events can be collected, whether they are enabled, and (for enabled instruments) whether to collect timing information:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments;
+----------------------------------------------------+---------+-------+
| NAME                                               | ENABLED | TIMED |
+----------------------------------------------------+---------+-------+
...
| wait/synch/mutex/sql/LOCK_global_read_lock         | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db                  | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                  | YES     | YES   |
...
```

  The `setup_instruments` table provides the most basic form of control over event production. To further refine event production based on the type of object or thread being monitored, other tables may be used as described in Section 26.4.3, “Event Pre-Filtering”.

- Performance Schema tables are the destinations for events and consume events. The `setup_consumers` table lists the types of consumers to which event information can be sent and whether they are enabled:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | YES     |
| events_statements_history_long | NO      |
| events_transactions_current    | YES     |
| events_transactions_history    | YES     |
| events_transactions_history_long | NO    |
| events_waits_current           | NO      |
| events_waits_history           | NO      |
| events_waits_history_long      | NO      |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
```

Filtering can be done at different stages of performance monitoring:

- **Pre-filtering.** This is done by modifying Performance Schema configuration so that only certain types of events are collected from producers, and collected events update only certain consumers. To do this, enable or disable instruments or consumers. Pre-filtering is done by the Performance Schema and has a global effect that applies to all users.

  Reasons to use pre-filtering:

  - To reduce overhead. Performance Schema overhead should be minimal even with all instruments enabled, but perhaps you want to reduce it further. Or you do not care about timing events and want to disable the timing code to eliminate timing overhead.

  - To avoid filling the current-events or history tables with events in which you have no interest. Pre-filtering leaves more “room” in these tables for instances of rows for enabled instrument types. If you enable only file instruments with pre-filtering, no rows are collected for nonfile instruments. With post-filtering, nonfile events are collected, leaving fewer rows for file events.

  - To avoid maintaining some kinds of event tables. If you disable a consumer, the server does not spend time maintaining destinations for that consumer. For example, if you do not care about event histories, you can disable the history table consumers to improve performance.

- **Post-filtering.** This involves the use of `WHERE` clauses in queries that select information from Performance Schema tables, to specify which of the available events you want to see. Post-filtering is performed on a per-user basis because individual users select which of the available events are of interest.

  Reasons to use post-filtering:

  - To avoid making decisions for individual users about which event information is of interest.

  - To use the Performance Schema to investigate a performance issue when the restrictions to impose using pre-filtering are not known in advance.

The following sections provide more detail about pre-filtering and provide guidelines for naming instruments or consumers in filtering operations. For information about writing queries to retrieve information (post-filtering), see Section 26.5, “Performance Schema Queries”.

26.4.3 Event Pre-Filtering

Pre-filtering is done by the Performance Schema and has a global effect that applies to all users. Pre-filtering can be applied to either the producer or consumer stage of event processing:

- To configure pre-filtering at the producer stage, several tables can be used:

  - `setup_instruments` indicates which instruments are available. An instrument disabled in this table produces no events regardless of the contents of the other production-related setup tables. An instrument enabled in this table is permitted to produce events, subject to the contents of the other tables.

  - `setup_objects` controls whether the Performance Schema monitors particular table and stored program objects.

  - `threads` indicates whether monitoring is enabled for each server thread.

  - `setup_actors` determines the initial monitoring state for new foreground threads.

- To configure pre-filtering at the consumer stage, modify the `setup_consumers` table. This determines the destinations to which events are sent. `setup_consumers` also implicitly affects event production. If a given event will not be sent to any destination (that is, will not be consumed), the Performance Schema does not produce it.

Modifications to any of these tables affect monitoring immediately, with the exception that modifications to the `setup_actors` table affect only foreground threads created subsequent to the modification, not existing threads.

When you change the monitoring configuration, the Performance Schema does not flush the history tables. Events already collected remain in the current-events and history tables until displaced by newer events.

If you disable instruments, you might need to wait a while before events for them are displaced by newer events of interest. Alternatively, use `TRUNCATE TABLE` to empty the history tables.

After making instrumentation changes, you might want to truncate the summary tables. Generally, the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. This enables you to clear collected values and restart aggregation. That might be useful, for example, after you have made a runtime configuration change. Exceptions to this truncation behavior are noted in individual summary table sections.

The following sections describe how to use specific tables to control Performance Schema pre-filtering.

26.4.4 Pre-Filtering by Instrument

The `setup_instruments` table lists the available instruments:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments;
+----------------------------------------------------+---------+-------+
| NAME                                               | ENABLED | TIMED |
+----------------------------------------------------+---------+-------+
...
| stage/sql/end                                      | NO      | NO    |
| stage/sql/executing                                | NO      | NO    |
| stage/sql/init                                     | NO      | NO    |
| stage/sql/insert                                   | NO      | NO    |
...
| statement/sql/load                                 | YES     | YES   |
| statement/sql/grant                                | YES     | YES   |
| statement/sql/check                                | YES     | YES   |
| statement/sql/flush                                | YES     | YES   |
...
| wait/synch/mutex/sql/LOCK_global_read_lock         | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db                  | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                  | YES     | YES   |
...
| wait/synch/rwlock/sql/LOCK_grant                   | YES     | YES   |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger          | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_connect        | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_slave          | YES     | YES   |
...
| wait/io/file/sql/binlog                            | YES     | YES   |
| wait/io/file/sql/binlog_index                      | YES     | YES   |
| wait/io/file/sql/casetest                          | YES     | YES   |
| wait/io/file/sql/dbopt                             | YES     | YES   |
...
```

To control whether an instrument is enabled, set its `ENABLED` column to `YES` or `NO`. To configure whether to collect timing information for an enabled instrument, set its `TIMED` value to `YES` or `NO`. Setting the `TIMED` column affects Performance Schema table contents as described in Section 26.4.1, “Performance Schema Event Timing”.

Modifications to most `setup_instruments` rows affect monitoring immediately. For some instruments, modifications are effective only at server startup; changing them at runtime has no effect. This affects primarily mutexes, conditions, and rwlocks in the server, although there may be other instruments for which this is true.

The `setup_instruments` table provides the most basic form of control over event production. To further refine event production based on the type of object or thread being monitored, other tables may be used as described in Section 26.4.3, “Event Pre-Filtering”.

The following examples demonstrate possible operations on the `setup_instruments` table. These changes, like other pre-filtering operations, affect all users. Some of these queries use the `LIKE` operator and a pattern match instrument names. For additional information about specifying patterns to select instruments, see Section 26.4.9, “Naming Instruments or Consumers for Filtering Operations”.

- Disable all instruments:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO';
```

  Now no events will be collected.

- Disable all file instruments, adding them to the current set of disabled instruments:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME LIKE 'wait/io/file/%';
```

- Disable only file instruments, enable all other instruments:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = IF(NAME LIKE 'wait/io/file/%', 'NO', 'YES');
```

- Enable all but those instruments in the `mysys` library:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = CASE WHEN NAME LIKE '%/mysys/%' THEN 'YES' ELSE 'NO' END;
```

- Disable a specific instrument:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME = 'wait/synch/mutex/mysys/TMPDIR_mutex';
```

- To toggle the state of an instrument, “flip” its `ENABLED` value:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = IF(ENABLED = 'YES', 'NO', 'YES')
WHERE NAME = 'wait/synch/mutex/mysys/TMPDIR_mutex';
```

- Disable timing for all events:

```
UPDATE performance_schema.setup_instruments
SET TIMED = 'NO';
```

26.4.5 Pre-Filtering by Object

The `setup_objects` table controls whether the Performance Schema monitors particular table and stored program objects. The initial `setup_objects` contents look like this:

```
mysql> SELECT * FROM performance_schema.setup_objects;
+-------------+--------------------+-------------+---------+-------+
| OBJECT_TYPE | OBJECT_SCHEMA      | OBJECT_NAME | ENABLED | TIMED |
+-------------+--------------------+-------------+---------+-------+
| EVENT       | mysql              | %           | NO      | NO    |
```

```
| EVENT       | performance_schema | %           | NO      | NO    |
| EVENT       | information_schema | %           | NO      | NO    |
| EVENT       | %                  | %           | YES     | YES   |
| FUNCTION    | mysql              | %           | NO      | NO    |
| FUNCTION    | performance_schema | %           | NO      | NO    |
| FUNCTION    | information_schema | %           | NO      | NO    |
| FUNCTION    | %                  | %           | YES     | YES   |
| PROCEDURE   | mysql              | %           | NO      | NO    |
| PROCEDURE   | performance_schema | %           | NO      | NO    |
| PROCEDURE   | information_schema | %           | NO      | NO    |
| PROCEDURE   | %                  | %           | YES     | YES   |
| TABLE       | mysql              | %           | NO      | NO    |
| TABLE       | performance_schema | %           | NO      | NO    |
| TABLE       | information_schema | %           | NO      | NO    |
| TABLE       | %                  | %           | YES     | YES   |
| TRIGGER     | mysql              | %           | NO      | NO    |
| TRIGGER     | performance_schema | %           | NO      | NO    |
| TRIGGER     | information_schema | %           | NO      | NO    |
| TRIGGER     | %                  | %           | YES     | YES   |
+-------------+--------------------+-------------+---------+-------+
```

Modifications to the `setup_objects` table affect object monitoring immediately.

The `OBJECT_TYPE` column indicates the type of object to which a row applies. `TABLE` filtering affects table I/O events (`wait/io/table/sql/handler` instrument) and table lock events (`wait/lock/table/sql/handler` instrument).

The `OBJECT_SCHEMA` and `OBJECT_NAME` columns should contain a literal schema or object name, or `'%'` to match any name.

The `ENABLED` column indicates whether matching objects are monitored, and `TIMED` indicates whether to collect timing information. Setting the `TIMED` column affects Performance Schema table contents as described in Section 26.4.1, “Performance Schema Event Timing”.

The effect of the default object configuration is to instrument all objects except those in the `mysql`, `INFORMATION_SCHEMA`, and `performance_schema` databases. (Tables in the `INFORMATION_SCHEMA` database are not instrumented regardless of the contents of `setup_objects`; the row for `information_schema.%` simply makes this default explicit.)

When the Performance Schema checks for a match in `setup_objects`, it tries to find more specific matches first. For rows that match a given `OBJECT_TYPE`, the Performance Schema checks rows in this order:

- Rows with `OBJECT_SCHEMA='literal'` and `OBJECT_NAME='literal'`.
- Rows with `OBJECT_SCHEMA='literal'` and `OBJECT_NAME='%'`.
- Rows with `OBJECT_SCHEMA='%'` and `OBJECT_NAME='%'`.

For example, with a table `db1.t1`, the Performance Schema looks in `TABLE` rows for a match for `'db1'` and `'t1'`, then for `'db1'` and `'%'`, then for `'%'` and `'%'`. The order in which matching occurs matters because different matching `setup_objects` rows can have different `ENABLED` and `TIMED` values.

For table-related events, the Performance Schema combines the contents of `setup_objects` with `setup_instruments` to determine whether to enable instruments and whether to time enabled instruments:

- For tables that match a row in `setup_objects`, table instruments produce events only if `ENABLED` is `YES` in both `setup_instruments` and `setup_objects`.

- The `TIMED` values in the two tables are combined, so that timing information is collected only when both values are `YES`.

For stored program objects, the Performance Schema takes the `ENABLED` and `TIMED` columns directly from the `setup_objects` row. There is no combining of values with `setup_instruments`.

Suppose that `setup_objects` contains the following `TABLE` rows that apply to `db1`, `db2`, and `db3`:

```
+-------------+---------------+-------------+---------+-------+
| OBJECT_TYPE | OBJECT_SCHEMA | OBJECT_NAME | ENABLED | TIMED |
+-------------+---------------+-------------+---------+-------+
| TABLE       | db1           | t1          | YES     | YES   |
| TABLE       | db1           | t2          | NO      | NO    |
| TABLE       | db2           | %           | YES     | YES   |
| TABLE       | db3           | %           | NO      | NO    |
| TABLE       | %             | %           | YES     | YES   |
+-------------+---------------+-------------+---------+-------+
```

If an object-related instrument in `setup_instruments` has an `ENABLED` value of `NO`, events for the object are not monitored. If the `ENABLED` value is `YES`, event monitoring occurs according to the `ENABLED` value in the relevant `setup_objects` row:

- `db1.t1` events are monitored
- `db1.t2` events are not monitored
- `db2.t3` events are monitored
- `db3.t4` events are not monitored
- `db4.t5` events are monitored

Similar logic applies for combining the `TIMED` columns from the `setup_instruments` and `setup_objects` tables to determine whether to collect event timing information.

If a persistent table and a temporary table have the same name, matching against `setup_objects` rows occurs the same way for both. It is not possible to enable monitoring for one table but not the other. However, each table is instrumented separately.

26.4.6 Pre-Filtering by Thread

The `threads` table contains a row for each server thread. Each row contains information about a thread and indicates whether monitoring is enabled for it. For the Performance Schema to monitor a thread, these things must be true:

- The `thread_instrumentation` consumer in the `setup_consumers` table must be `YES`.
- The `threads.INSTRUMENTED` column must be `YES`.
- Monitoring occurs only for those thread events produced from instruments that are enabled in the `setup_instruments` table.

The `threads` table also indicates for each server thread whether to perform historical event logging. This includes wait, stage, statement, and transaction events and affects logging to these tables:

```
events_waits_history
events_waits_history_long
events_stages_history
```

```
events_stages_history_long
events_statements_history
events_statements_history_long
events_transactions_history
events_transactions_history_long
```

For historical event logging to occur, these things must be true:

- The appropriate history-related consumers in the `setup_consumers` table must be enabled. For example, wait event logging in the `events_waits_history` and `events_waits_history_long` tables requires the corresponding `events_waits_history` and `events_waits_history_long` consumers to be `YES`.
- The `threads.HISTORY` column must be `YES`.
- Logging occurs only for those thread events produced from instruments that are enabled in the `setup_instruments` table.

For foreground threads (resulting from client connections), the initial values of the `INSTRUMENTED` and `HISTORY` columns in `threads` table rows are determined by whether the user account associated with a thread matches any row in the `setup_actors` table. The values come from the `ENABLED` and `HISTORY` columns of the matching `setup_actors` table row.

For background threads, there is no associated user. `INSTRUMENTED` and `HISTORY` are `YES` by default and `setup_actors` is not consulted.

The initial `setup_actors` contents look like this:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+------+------+------+---------+---------+
| HOST | USER | ROLE | ENABLED | HISTORY |
+------+------+------+---------+---------+
| %    | %    | %    | YES     | YES     |
+------+------+------+---------+---------+
```

The `HOST` and `USER` columns should contain a literal host or user name, or `'%'` to match any name.

The `ENABLED` and `HISTORY` columns indicate whether to enable instrumentation and historical event logging for matching threads, subject to the other conditions described previously.

When the Performance Schema checks for a match for each new foreground thread in `setup_actors`, it tries to find more specific matches first, using the `USER` and `HOST` columns (`ROLE` is unused):

- Rows with `USER='literal'` and `HOST='literal'`.
- Rows with `USER='literal'` and `HOST='%'`.
- Rows with `USER='%'` and `HOST='literal'`.
- Rows with `USER='%'` and `HOST='%'`.

The order in which matching occurs matters because different matching `setup_actors` rows can have different `USER` and `HOST` values. This enables instrumenting and historical event logging to be applied selectively per host, user, or account (user and host combination), based on the `ENABLED` and `HISTORY` column values:

- When the best match is a row with `ENABLED=YES`, the `INSTRUMENTED` value for the thread becomes `YES`. When the best match is a row with `HISTORY=YES`, the `HISTORY` value for the thread becomes `YES`.

- When the best match is a row with `ENABLED=NO`, the `INSTRUMENTED` value for the thread becomes `NO`. When the best match is a row with `HISTORY=NO`, the `HISTORY` value for the thread becomes `NO`.
- When no match is found, the `INSTRUMENTED` and `HISTORY` values for the thread become `NO`.

The `ENABLED` and `HISTORY` columns in `setup_actors` rows can be set to `YES` or `NO` independent of one another. This means you can enable instrumentation separately from whether you collect historical events.

By default, monitoring and historical event collection are enabled for all new foreground threads because the `setup_actors` table initially contains a row with `'%'` for both `HOST` and `USER`. To perform more limited matching such as to enable monitoring only for some foreground threads, you must change this row because it matches any connection, and add rows for more specific `HOST`/`USER` combinations.

Suppose that you modify `setup_actors` as follows:

```
UPDATE performance_schema.setup_actors
SET ENABLED = 'NO', HISTORY = 'NO'
WHERE HOST = '%' AND USER = '%';
INSERT INTO performance_schema.setup_actors
(HOST,USER,ROLE,ENABLED,HISTORY)
VALUES('localhost','joe','%','YES','YES');
INSERT INTO performance_schema.setup_actors
(HOST,USER,ROLE,ENABLED,HISTORY)
VALUES('hosta.example.com','joe','%','YES','NO');
INSERT INTO performance_schema.setup_actors
(HOST,USER,ROLE,ENABLED,HISTORY)
VALUES('%','sam','%','NO','YES');
```

The `UPDATE` statement changes the default match to disable instrumentation and historical event collection. The `INSERT` statements add rows for more specific matches.

Now the Performance Schema determines how to set the `INSTRUMENTED` and `HISTORY` values for new connection threads as follows:

- If `joe` connects from the local host, the connection matches the first inserted row. The `INSTRUMENTED` and `HISTORY` values for the thread become `YES`.
- If `joe` connects from `hosta.example.com`, the connection matches the second inserted row. The `INSTRUMENTED` value for the thread becomes `YES` and the `HISTORY` value becomes `NO`.
- If `joe` connects from any other host, there is no match. The `INSTRUMENTED` and `HISTORY` values for the thread become `NO`.
- If `sam` connects from any host, the connection matches the third inserted row. The `INSTRUMENTED` value for the thread becomes `NO` and the `HISTORY` value becomes `YES`.
- For any other connection, the row with `HOST` and `USER` set to `'%'` matches. This row now has `ENABLED` and `HISTORY` set to `NO`, so the `INSTRUMENTED` and `HISTORY` values for the thread become `NO`.

Modifications to the `setup_actors` table affect only foreground threads created subsequent to the modification, not existing threads. To affect existing threads, modify the `INSTRUMENTED` and `HISTORY` columns of `threads` table rows.

26.4.7 Pre-Filtering by Consumer

The `setup_consumers` table lists the available consumer types and which are enabled:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | YES     |
| events_statements_history_long | NO      |
| events_transactions_current    | YES     |
| events_transactions_history    | YES     |
| events_transactions_history_long | NO    |
| events_waits_current           | NO      |
| events_waits_history           | NO      |
| events_waits_history_long      | NO      |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
```

Modify the `setup_consumers` table to affect pre-filtering at the consumer stage and determine the destinations to which events are sent. To enable or disable a consumer, set its `ENABLED` value to `YES` or `NO`.

Modifications to the `setup_consumers` table affect monitoring immediately.

If you disable a consumer, the server does not spend time maintaining destinations for that consumer. For example, if you do not care about historical event information, disable the history consumers:

```
UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE '%history%';
```

The consumer settings in the `setup_consumers` table form a hierarchy from higher levels to lower. The following principles apply:

- Destinations associated with a consumer receive no events unless the Performance Schema checks the consumer and the consumer is enabled.
- A consumer is checked only if all consumers it depends on (if any) are enabled.
- If a consumer is not checked, or is checked but is disabled, other consumers that depend on it are not checked.
- Dependent consumers may have their own dependent consumers.
- If an event would not be sent to any destination, the Performance Schema does not produce it.

The following lists describe the available consumer values. For discussion of several representative consumer configurations and their effect on instrumentation, see Section 26.4.8, “Example Consumer Configurations”.

- Global and Thread Consumers
- Wait Event Consumers
- Stage Event Consumers
- Statement Event Consumers

- Transaction Event Consumers
- Statement Digest Consumer

Global and Thread Consumers

- `global_instrumentation` is the highest level consumer. If `global_instrumentation` is `NO`, it disables global instrumentation. All other settings are lower level and are not checked; it does not matter what they are set to. No global or per thread information is maintained and no individual events are collected in the current-events or event-history tables. If `global_instrumentation` is `YES`, the Performance Schema maintains information for global states and also checks the `thread_instrumentation` consumer.
- `thread_instrumentation` is checked only if `global_instrumentation` is `YES`. Otherwise, if `thread_instrumentation` is `NO`, it disables thread-specific instrumentation and all lower-level settings are ignored. No information is maintained per thread and no individual events are collected in the current-events or event-history tables. If `thread_instrumentation` is `YES`, the Performance Schema maintains thread-specific information and also checks `events_xxx_current` consumers.

Wait Event Consumers

These consumers require both `global_instrumentation` and `thread_instrumentation` to be `YES` or they are not checked. If checked, they act as follows:

- `events_waits_current`, if `NO`, disables collection of individual wait events in the `events_waits_current` table. If `YES`, it enables wait event collection and the Performance Schema checks the `events_waits_history` and `events_waits_history_long` consumers.
- `events_waits_history` is not checked if `event_waits_current` is `NO`. Otherwise, an `events_waits_history` value of `NO` or `YES` disables or enables collection of wait events in the `events_waits_history` table.
- `events_waits_history_long` is not checked if `event_waits_current` is `NO`. Otherwise, an `events_waits_history_long` value of `NO` or `YES` disables or enables collection of wait events in the `events_waits_history_long` table.

Stage Event Consumers

These consumers require both `global_instrumentation` and `thread_instrumentation` to be `YES` or they are not checked. If checked, they act as follows:

- `events_stages_current`, if `NO`, disables collection of individual stage events in the `events_stages_current` table. If `YES`, it enables stage event collection and the Performance Schema checks the `events_stages_history` and `events_stages_history_long` consumers.
- `events_stages_history` is not checked if `event_stages_current` is `NO`. Otherwise, an `events_stages_history` value of `NO` or `YES` disables or enables collection of stage events in the `events_stages_history` table.
- `events_stages_history_long` is not checked if `event_stages_current` is `NO`. Otherwise, an `events_stages_history_long` value of `NO` or `YES` disables or enables collection of stage events in the `events_stages_history_long` table.

Statement Event Consumers

These consumers require both `global_instrumentation` and `thread_instrumentation` to be `YES` or they are not checked. If checked, they act as follows:

- `events_statements_current`, if `NO`, disables collection of individual statement events in the `events_statements_current` table. If `YES`, it enables statement event collection and the Performance Schema checks the `events_statements_history` and `events_statements_history_long` consumers.
- `events_statements_history` is not checked if `events_statements_current` is `NO`. Otherwise, an `events_statements_history` value of `NO` or `YES` disables or enables collection of statement events in the `events_statements_history` table.
- `events_statements_history_long` is not checked if `events_statements_current` is `NO`. Otherwise, an `events_statements_history_long` value of `NO` or `YES` disables or enables collection of statement events in the `events_statements_history_long` table.

Transaction Event Consumers

These consumers require both `global_instrumentation` and `thread_instrumentation` to be `YES` or they are not checked. If checked, they act as follows:

- `events_transactions_current`, if `NO`, disables collection of individual transaction events in the `events_transactions_current` table. If `YES`, it enables transaction event collection and the Performance Schema checks the `events_transactions_history` and `events_transactions_history_long` consumers.
- `events_transactions_history` is not checked if `events_transactions_current` is `NO`. Otherwise, an `events_transactions_history` value of `NO` or `YES` disables or enables collection of transaction events in the `events_transactions_history` table.
- `events_transactions_history_long` is not checked if `events_transactions_current` is `NO`. Otherwise, an `events_transactions_history_long` value of `NO` or `YES` disables or enables collection of transaction events in the `events_transactions_history_long` table.

Statement Digest Consumer

The `statements_digest` consumer requires `global_instrumentation` to be `YES` or it is not checked. There is no dependency on the statement event consumers, so you can obtain statistics per digest without having to collect statistics in `events_statements_current`, which is advantageous in terms of overhead. Conversely, you can get detailed statements in `events_statements_current` without digests (the `DIGEST` and `DIGEST_TEXT` columns will be `NULL`).

For more information about statement digesting, see Section 26.10, “Performance Schema Statement Digests and Sampling”.

26.4.8 Example Consumer Configurations

The consumer settings in the `setup_consumers` table form a hierarchy from higher levels to lower. The following discussion describes how consumers work, showing specific configurations and their effects as consumer settings are enabled progressively from high to low. The consumer values shown are representative. The general principles described here apply to other consumer values that may be available.

The configuration descriptions occur in order of increasing functionality and overhead. If you do not need the information provided by enabling lower-level settings, disable them and the Performance Schema will execute less code on your behalf and you will have less information to sift through.

The `setup_consumers` table contains the following hierarchy of values:

```
global_instrumentation
 thread_instrumentation
   events_waits_current
     events_waits_history
     events_waits_history_long
   events_stages_current
     events_stages_history
     events_stages_history_long
   events_statements_current
     events_statements_history
     events_statements_history_long
   events_transactions_current
     events_transactions_history
     events_transactions_history_long
 statements_digest
```

Note

In the consumer hierarchy, the consumers for waits, stages, statements, and transactions are all at the same level. This differs from the event nesting hierarchy, for which wait events nest within stage events, which nest within statement events, which nest within transaction events.

If a given consumer setting is `NO`, the Performance Schema disables the instrumentation associated with the consumer and ignores all lower-level settings. If a given setting is `YES`, the Performance Schema enables the instrumentation associated with it and checks the settings at the next lowest level. For a description of the rules for each consumer, see Section 26.4.7, “Pre-Filtering by Consumer”.

For example, if `global_instrumentation` is enabled, `thread_instrumentation` is checked. If `thread_instrumentation` is enabled, the `events_`*`xxx`*`_current` consumers are checked. If of these `events_waits_current` is enabled, `events_waits_history` and `events_waits_history_long` are checked.

Each of the following configuration descriptions indicates which setup elements the Performance Schema checks and which output tables it maintains (that is, for which tables it collects information).

- No Instrumentation
- Global Instrumentation Only
- Global and Thread Instrumentation Only
- Global, Thread, and Current-Event Instrumentation
- Global, Thread, Current-Event, and Event-History instrumentation

No Instrumentation

Server configuration state:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+---------------------------+---------+
| NAME                      | ENABLED |
+---------------------------+---------+
| global_instrumentation    | NO      |
...
+---------------------------+---------+
```

In this configuration, nothing is instrumented.

Setup elements checked:

- Table `setup_consumers`, consumer `global_instrumentation`

Output tables maintained:

- None

Global Instrumentation Only

Server configuration state:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+---------------------------+---------+
| NAME                      | ENABLED |
+---------------------------+---------+
| global_instrumentation    | YES     |
| thread_instrumentation    | NO      |
...
+---------------------------+---------+
```

In this configuration, instrumentation is maintained only for global states. Per-thread instrumentation is disabled.

Additional setup elements checked, relative to the preceding configuration:

- Table `setup_consumers`, consumer `thread_instrumentation`
- Table `setup_instruments`
- Table `setup_objects`

Additional output tables maintained, relative to the preceding configuration:

- `mutex_instances`
- `rwlock_instances`
- `cond_instances`
- `file_instances`
- `users`
- `hosts`
- `accounts`
- `socket_summary_by_event_name`
- `file_summary_by_instance`
- `file_summary_by_event_name`
- `objects_summary_global_by_type`
- `memory_summary_global_by_event_name`
- `table_lock_waits_summary_by_table`

- table_io_waits_summary_by_index_usage
- table_io_waits_summary_by_table
- events_waits_summary_by_instance
- events_waits_summary_global_by_event_name
- events_stages_summary_global_by_event_name
- events_statements_summary_global_by_event_name
- events_transactions_summary_global_by_event_name

Global and Thread Instrumentation Only

Server configuration state:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| events_waits_current             | NO      |
...
| events_stages_current            | NO      |
...
| events_statements_current        | NO      |
...
| events_transactions_current      | NO      |
...
+----------------------------------+---------+
```

In this configuration, instrumentation is maintained globally and per thread. No individual events are collected in the current-events or event-history tables.

Additional setup elements checked, relative to the preceding configuration:

- Table setup_consumers, consumers events_*xxx*_current, where *xxx* is waits, stages, statements, transactions
- Table setup_actors
- Column threads.instrumented

Additional output tables maintained, relative to the preceding configuration:

- events_*xxx*_summary_by_*yyy*_by_event_name, where *xxx* is waits, stages, statements, transactions; and *yyy* is thread, user, host, account

Global, Thread, and Current-Event Instrumentation

Server configuration state:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| global_instrumentation           | YES     |
```

```
| thread_instrumentation           | YES     |
| events_waits_current             | YES     |
| events_waits_history             | NO      |
| events_waits_history_long        | NO      |
| events_stages_current            | YES     |
| events_stages_history            | NO      |
| events_stages_history_long       | NO      |
| events_statements_current        | YES     |
| events_statements_history        | NO      |
| events_statements_history_long   | NO      |
| events_transactions_current      | YES     |
| events_transactions_history      | NO      |
| events_transactions_history_long | NO      |
...
+----------------------------------+---------+
```

In this configuration, instrumentation is maintained globally and per thread. Individual events are collected in the current-events table, but not in the event-history tables.

Additional setup elements checked, relative to the preceding configuration:

- Consumers `events_xxx_history`, where *`xxx`* is `waits`, `stages`, `statements`, `transactions`
- Consumers `events_xxx_history_long`, where *`xxx`* is `waits`, `stages`, `statements`, `transactions`

Additional output tables maintained, relative to the preceding configuration:

- `events_xxx_current`, where *`xxx`* is `waits`, `stages`, `statements`, `transactions`

Global, Thread, Current-Event, and Event-History instrumentation

The preceding configuration collects no event history because the `events_xxx_history` and `events_xxx_history_long` consumers are disabled. Those consumers can be enabled separately or together to collect event history per thread, globally, or both.

This configuration collects event history per thread, but not globally:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+----------------------------------+---------+
| NAME                             | ENABLED |
+----------------------------------+---------+
| global_instrumentation           | YES     |
| thread_instrumentation           | YES     |
| events_waits_current             | YES     |
| events_waits_history             | YES     |
| events_waits_history_long        | NO      |
| events_stages_current            | YES     |
| events_stages_history            | YES     |
| events_stages_history_long       | NO      |
| events_statements_current        | YES     |
| events_statements_history        | YES     |
| events_statements_history_long   | NO      |
| events_transactions_current      | YES     |
| events_transactions_history      | YES     |
| events_transactions_history_long | NO      |
...
+----------------------------------+---------+
```

Event-history tables maintained for this configuration:

- `events_xxx_history`, where *`xxx`* is `waits`, `stages`, `statements`, `transactions`

This configuration collects event history globally, but not per thread:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+---------------------------------+---------+
| NAME                            | ENABLED |
+---------------------------------+---------+
| global_instrumentation          | YES     |
| thread_instrumentation          | YES     |
| events_waits_current            | YES     |
| events_waits_history            | NO      |
| events_waits_history_long       | YES     |
| events_stages_current           | YES     |
| events_stages_history           | NO      |
| events_stages_history_long      | YES     |
| events_statements_current       | YES     |
| events_statements_history       | NO      |
| events_statements_history_long  | YES     |
| events_transactions_current     | YES     |
| events_transactions_history     | NO      |
| events_transactions_history_long | YES     |
...
+---------------------------------+---------+
```

Event-history tables maintained for this configuration:

- `events_xxx_history_long`, where *xxx* is `waits`, `stages`, `statements`, `transactions`

This configuration collects event history per thread and globally:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+---------------------------------+---------+
| NAME                            | ENABLED |
+---------------------------------+---------+
| global_instrumentation          | YES     |
| thread_instrumentation          | YES     |
| events_waits_current            | YES     |
| events_waits_history            | YES     |
| events_waits_history_long       | YES     |
| events_stages_current           | YES     |
| events_stages_history           | YES     |
| events_stages_history_long      | YES     |
| events_statements_current       | YES     |
| events_statements_history       | YES     |
| events_statements_history_long  | YES     |
| events_transactions_current     | YES     |
| events_transactions_history     | YES     |
| events_transactions_history_long | YES     |
...
+---------------------------------+---------+
```

Event-history tables maintained for this configuration:

- `events_xxx_history`, where *xxx* is `waits`, `stages`, `statements`, `transactions`
- `events_xxx_history_long`, where *xxx* is `waits`, `stages`, `statements`, `transactions`

26.4.9 Naming Instruments or Consumers for Filtering Operations

Names given for filtering operations can be as specific or general as required. To indicate a single instrument or consumer, specify its name in full:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME = 'wait/synch/mutex/myisammrg/MYRG_INFO::mutex';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME = 'events_waits_current';
```

To specify a group of instruments or consumers, use a pattern that matches the group members:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME LIKE 'wait/synch/mutex/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE '%history%';
```

If you use a pattern, it should be chosen so that it matches all the items of interest and no others. For example, to select all file I/O instruments, it is better to use a pattern that includes the entire instrument name prefix:

```
... WHERE NAME LIKE 'wait/io/file/%';
```

A pattern of `'%/file/%'` will match other instruments that have a component of `'/file/'` anywhere in the name. Even less suitable is the pattern `'%file%'` because it will match instruments with `'file'` anywhere in the name, such as `wait/synch/mutex/innodb/file_open_mutex`.

To check which instrument or consumer names a pattern matches, perform a simple test:

```
SELECT NAME FROM performance_schema.setup_instruments
WHERE NAME LIKE 'pattern';

SELECT NAME FROM performance_schema.setup_consumers
WHERE NAME LIKE 'pattern';
```

For information about the types of names that are supported, see Section 26.6, “Performance Schema Instrument Naming Conventions”.

26.4.10 Determining What Is Instrumented

It is always possible to determine what instruments the Performance Schema includes by checking the `setup_instruments` table. For example, to see what file-related events are instrumented for the `InnoDB` storage engine, use this query:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'wait/io/file/innodb/%';
+----------------------------------------------+---------+-------+
| NAME                                         | ENABLED | TIMED |
+----------------------------------------------+---------+-------+
| wait/io/file/innodb/innodb_tablespace_open_file | YES     | YES   |
| wait/io/file/innodb/innodb_data_file            | YES     | YES   |
| wait/io/file/innodb/innodb_log_file             | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file            | YES     | YES   |
| wait/io/file/innodb/innodb_arch_file            | YES     | YES   |
| wait/io/file/innodb/innodb_clone_file           | YES     | YES   |
+----------------------------------------------+---------+-------+
```

An exhaustive description of precisely what is instrumented is not given in this documentation, for several reasons:

- What is instrumented is the server code. Changes to this code occur often, which also affects the set of instruments.
- It is not practical to list all the instruments because there are hundreds of them.
- As described earlier, it is possible to find out by querying the `setup_instruments` table. This information is always up to date for your version of MySQL, also includes instrumentation for instrumented plugins you might have installed that are not part of the core server, and can be used by automated tools.

26.5 Performance Schema Queries

Pre-filtering limits which event information is collected and is independent of any particular user. By contrast, post-filtering is performed by individual users through the use of queries with appropriate `WHERE` clauses that restrict what event information to select from the events available after pre-filtering has been applied.

In Section 26.4.3, “Event Pre-Filtering”, an example showed how to pre-filter for file instruments. If the event tables contain both file and nonfile information, post-filtering is another way to see information only for file events. Add a `WHERE` clause to queries to restrict event selection appropriately:

```
mysql> SELECT THREAD_ID, NUMBER_OF_BYTES
       FROM performance_schema.events_waits_history
       WHERE EVENT_NAME LIKE 'wait/io/file/%'
       AND NUMBER_OF_BYTES IS NOT NULL;
+-----------+-----------------+
| THREAD_ID | NUMBER_OF_BYTES |
+-----------+-----------------+
|        11 |              66 |
|        11 |              47 |
|        11 |             139 |
|         5 |              24 |
|         5 |             834 |
+-----------+-----------------+
```

Most Performance Schema tables have indexes, which gives the optimizer access to execution plans other than full table scans. These indexes also improve performance for related objects, such as `sys` schema views that use those tables. For more information, see Section 8.2.4, “Optimizing Performance Schema Queries”.

26.6 Performance Schema Instrument Naming Conventions

An instrument name consists of a sequence of components separated by `'/'` characters. Example names:

```
wait/io/file/myisam/log
wait/io/file/mysys/charset
wait/lock/table/sql/handler
wait/synch/cond/mysys/COND_alarm
wait/synch/cond/sql/BINLOG::update_cond
wait/synch/mutex/mysys/BITMAP_mutex
wait/synch/mutex/sql/LOCK_delete
wait/synch/rwlock/sql/Query_cache_query::lock
stage/sql/closing tables
stage/sql/Sorting result
statement/com/Execute
```

```
statement/com/Query
statement/sql/create_table
statement/sql/lock_tables
errors
```

The instrument name space has a tree-like structure. The components of an instrument name from left to right provide a progression from more general to more specific. The number of components a name has depends on the type of instrument.

The interpretation of a given component in a name depends on the components to the left of it. For example, `myisam` appears in both of the following names, but `myisam` in the first name is related to file I/O, whereas in the second it is related to a synchronization instrument:

```
wait/io/file/myisam/log
wait/synch/cond/myisam/MI_SORT_INFO::cond
```

Instrument names consist of a prefix with a structure defined by the Performance Schema implementation and a suffix defined by the developer implementing the instrument code. The top-level component of an instrument prefix indicates the type of instrument. This component also determines which event timer in the `performance_timers` table applies to the instrument. For the prefix part of instrument names, the top level indicates the type of instrument.

The suffix part of instrument names comes from the code for the instruments themselves. Suffixes may include levels such as these:

- A name for the major component (a server module such as `myisam`, `innodb`, `mysys`, or `sql`) or a plugin name.
- The name of a variable in the code, in the form *`XXX`* (a global variable) or *`CCC`*`::`*`MMM`* (a member *`MMM`* in class *`CCC`*). Examples: `COND_thread_cache`, `THR_LOCK_myisam`, `BINLOG::LOCK_index`.

- Top-Level Instrument Components
- Idle Instrument Components
- Error Instrument Components
- Memory Instrument Components
- Stage Instrument Components
- Statement Instrument Components
- Thread Instrument Components
- Wait Instrument Components

Top-Level Instrument Components

- `idle`: An instrumented idle event. This instrument has no further components.
- `error`: An instrumented error event. This instrument has no further components.
- `memory`: An instrumented memory event.
- `stage`: An instrumented stage event.
- `statement`: An instrumented statement event.

- `transaction`: An instrumented transaction event. This instrument has no further components.
- `wait`: An instrumented wait event.

Idle Instrument Components

The `idle` instrument is used for idle events, which The Performance Schema generates as discussed in the description of the `socket_instances.STATE` column in Section 26.12.3.5, “The socket_instances Table”.

Error Instrument Components

The `error` instrument indicates whether to collect information for server errors and warnings. This instrument is enabled by default. The `TIMED` column for the `error` row in the `setup_instruments` table is inapplicable because timing information is not collected.

Memory Instrument Components

Memory instrumentation is enabled by default. Memory instrumentation can be enabled or disabled at startup, or dynamically at runtime by updating the `ENABLED` column of the relevant instruments in the `setup_instruments` table. Memory instruments have names of the form `memory/`*`code_area`*`/`*`instrument_name`* where *`code_area`* is a value such as `sql` or `myisam`, and *`instrument_name`* is the instrument detail.

Instruments named with the prefix `memory/performance_schema/` expose how much memory is allocated for internal buffers in the Performance Schema. The `memory/performance_schema/` instruments are built in, always enabled, and cannot be disabled at startup or runtime. Built-in memory instruments are displayed only in the `memory_summary_global_by_event_name` table. For more information, see Section 26.17, “The Performance Schema Memory-Allocation Model”.

Stage Instrument Components

Stage instruments have names of the form `stage/`*`code_area`*`/`*`stage_name`*, where *`code_area`* is a value such as `sql` or `myisam`, and *`stage_name`* indicates the stage of statement processing, such as `Sorting result` or `Sending data`. Stages correspond to the thread states displayed by `SHOW PROCESSLIST` or that are visible in the `INFORMATION_SCHEMA.PROCESSLIST` table.

Statement Instrument Components

- `statement/abstract/*`: An abstract instrument for statement operations. Abstract instruments are used during the early stages of statement classification before the exact statement type is known, then changed to a more specific statement instrument when the type is known. For a description of this process, see Section 26.12.6, “Performance Schema Statement Event Tables”.
- `statement/com`: An instrumented command operation. These have names corresponding to `COM_`*`xxx`* operations (see the `mysql_com.h` header file and `sql/sql_parse.cc`. For example, the `statement/com/Connect` and `statement/com/Init DB` instruments correspond to the `COM_CONNECT` and `COM_INIT_DB` commands.
- `statement/scheduler/event`: A single instrument to track all events executed by the Event Scheduler. This instrument comes into play when a scheduled event begins executing.
- `statement/sp`: An instrumented internal instruction executed by a stored program. For example, the `statement/sp/cfetch` and `statement/sp/freturn` instruments are used cursor fetch and function return instructions.

- `statement/sql`: An instrumented SQL statement operation. For example, the `statement/sql/create_db` and `statement/sql/select` instruments are used for `CREATE DATABASE` and `SELECT` statements.

Thread Instrument Components

Instrumented threads are displayed in the `setup_threads` table, which exposes thread class names and attributes.

Thread instruments begin with `thread` (for example, `thread/sql/parser_service` or `thread/performance_schema/setup`).

Wait Instrument Components

- `wait/io`

  An instrumented I/O operation.

  - `wait/io/file`

    An instrumented file I/O operation. For files, the wait is the time waiting for the file operation to complete (for example, a call to `fwrite()`). Due to caching, the physical file I/O on the disk might not happen within this call.

  - `wait/io/socket`

    An instrumented socket operation. Socket instruments have names of the form `wait/io/socket/sql/`*`socket_type`*. The server has a listening socket for each network protocol that it supports. The instruments associated with listening sockets for TCP/IP or Unix socket file connections have a *`socket_type`* value of `server_tcpip_socket` or `server_unix_socket`, respectively. When a listening socket detects a connection, the server transfers the connection to a new socket managed by a separate thread. The instrument for the new connection thread has a *`socket_type`* value of `client_connection`.

  - `wait/io/table`

    An instrumented table I/O operation. These include row-level accesses to persistent base tables or temporary tables. Operations that affect rows are fetch, insert, update, and delete. For a view, waits are associated with base tables referenced by the view.

    Unlike most waits, a table I/O wait can include other waits. For example, table I/O might include file I/O or memory operations. Thus, `events_waits_current` for a table I/O wait usually has two rows. For more information, see Section 26.8, “Performance Schema Atom and Molecule Events”.

    Some row operations might cause multiple table I/O waits. For example, an insert might activate a trigger that causes an update.

- `wait/lock`

  An instrumented lock operation.

  - `wait/lock/table`

    An instrumented table lock operation.

  - `wait/lock/metadata/sql/mdl`

An instrumented metadata lock operation.

- `wait/synch`

  An instrumented synchronization object. For synchronization objects, the `TIMER_WAIT` time includes the amount of time blocked while attempting to acquire a lock on the object, if any.

  - `wait/synch/cond`

    A condition is used by one thread to signal to other threads that something they were waiting for has happened. If a single thread was waiting for a condition, it can wake up and proceed with its execution. If several threads were waiting, they can all wake up and compete for the resource for which they were waiting.

  - `wait/synch/mutex`

    A mutual exclusion object used to permit access to a resource (such as a section of executable code) while preventing other threads from accessing the resource.

  - `wait/synch/prlock`

    A priority rwlock lock object.

  - `wait/synch/rwlock`

    A plain read/write lock object used to lock a specific variable for access while preventing its use by other threads. A shared read lock can be acquired simultaneously by multiple threads. An exclusive write lock can be acquired by only one thread at a time.

  - `wait/synch/sxlock`

    A shared-exclusive (SX) lock is a type of rwlock lock object that provides write access to a common resource while permitting inconsistent reads by other threads. `sxlocks` optimize concurrency and improve scalability for read-write workloads.

26.7 Performance Schema Status Monitoring

There are several status variables associated with the Performance Schema:

```
mysql> SHOW STATUS LIKE 'perf%';
+-----------------------------------------------+-------+
| Variable_name                                 | Value |
+-----------------------------------------------+-------+
| Performance_schema_accounts_lost              | 0     |
| Performance_schema_cond_classes_lost          | 0     |
| Performance_schema_cond_instances_lost        | 0     |
| Performance_schema_digest_lost                | 0     |
| Performance_schema_file_classes_lost          | 0     |
| Performance_schema_file_handles_lost          | 0     |
| Performance_schema_file_instances_lost        | 0     |
| Performance_schema_hosts_lost                 | 0     |
| Performance_schema_locker_lost                | 0     |
| Performance_schema_memory_classes_lost        | 0     |
| Performance_schema_metadata_lock_lost         | 0     |
| Performance_schema_mutex_classes_lost         | 0     |
| Performance_schema_mutex_instances_lost       | 0     |
| Performance_schema_nested_statement_lost      | 0     |
| Performance_schema_program_lost               | 0     |
```

```
| Performance_schema_rwlock_classes_lost        | 0     |
| Performance_schema_rwlock_instances_lost      | 0     |
| Performance_schema_session_connect_attrs_lost | 0     |
| Performance_schema_socket_classes_lost        | 0     |
| Performance_schema_socket_instances_lost      | 0     |
| Performance_schema_stage_classes_lost         | 0     |
| Performance_schema_statement_classes_lost     | 0     |
| Performance_schema_table_handles_lost         | 0     |
| Performance_schema_table_instances_lost       | 0     |
| Performance_schema_thread_classes_lost        | 0     |
| Performance_schema_thread_instances_lost      | 0     |
| Performance_schema_users_lost                 | 0     |
+-----------------------------------------------+-------+
```

The Performance Schema status variables provide information about instrumentation that could not be loaded or created due to memory constraints. Names for these variables have several forms:

- `Performance_schema_xxx_classes_lost` indicates how many instruments of type *`xxx`* could not be loaded.
- `Performance_schema_xxx_instances_lost` indicates how many instances of object type *`xxx`* could not be created.
- `Performance_schema_xxx_handles_lost` indicates how many instances of object type *`xxx`* could not be opened.
- `Performance_schema_locker_lost` indicates how many events are “lost” or not recorded.

For example, if a mutex is instrumented in the server source but the server cannot allocate memory for the instrumentation at runtime, it increments `Performance_schema_mutex_classes_lost`. The mutex still functions as a synchronization object (that is, the server continues to function normally), but performance data for it will not be collected. If the instrument can be allocated, it can be used for initializing instrumented mutex instances. For a singleton mutex such as a global mutex, there will be only one instance. Other mutexes have an instance per connection, or per page in various caches and data buffers, so the number of instances varies over time. Increasing the maximum number of connections or the maximum size of some buffers will increase the maximum number of instances that might be allocated at once. If the server cannot create a given instrumented mutex instance, it increments `Performance_schema_mutex_instances_lost`.

Suppose that the following conditions hold:

- The server was started with the `--performance_schema_max_mutex_classes=200` option and thus has room for 200 mutex instruments.
- 150 mutex instruments have been loaded already.
- The plugin named `plugin_a` contains 40 mutex instruments.
- The plugin named `plugin_b` contains 20 mutex instruments.

The server allocates mutex instruments for the plugins depending on how many they need and how many are available, as illustrated by the following sequence of statements:

```
INSTALL PLUGIN plugin_a
```

The server now has 150+40 = 190 mutex instruments.

```
UNINSTALL PLUGIN plugin_a;
```

The server still has 190 instruments. All the historical data generated by the plugin code is still available, but new events for the instruments are not collected.

```
INSTALL PLUGIN plugin_a;
```

The server detects that the 40 instruments are already defined, so no new instruments are created, and previously assigned internal memory buffers are reused. The server still has 190 instruments.

```
INSTALL PLUGIN plugin_b;
```

The server has room for 200-190 = 10 instruments (in this case, mutex classes), and sees that the plugin contains 20 new instruments. 10 instruments are loaded, and 10 are discarded or “lost.” The `Performance_schema_mutex_classes_lost` indicates the number of instruments (mutex classes) lost:

```
mysql> SHOW STATUS LIKE "perf%mutex_classes_lost";
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Performance_schema_mutex_classes_lost | 10    |
+---------------------------------------+-------+
1 row in set (0.10 sec)
```

The instrumentation still works and collects (partial) data for `plugin_b`.

When the server cannot create a mutex instrument, these results occur:

- No row for the instrument is inserted into the `setup_instruments` table.
- `Performance_schema_mutex_classes_lost` increases by 1.
- `Performance_schema_mutex_instances_lost` does not change. (When the mutex instrument is not created, it cannot be used to create instrumented mutex instances later.)

The pattern just described applies to all types of instruments, not just mutexes.

A value of `Performance_schema_mutex_classes_lost` greater than 0 can happen in two cases:

- To save a few bytes of memory, you start the server with `--performance_schema_max_mutex_classes=N`, where *`N`* is less than the default value. The default value is chosen to be sufficient to load all the plugins provided in the MySQL distribution, but this can be reduced if some plugins are never loaded. For example, you might choose not to load some of the storage engines in the distribution.
- You load a third-party plugin that is instrumented for the Performance Schema but do not allow for the plugin's instrumentation memory requirements when you start the server. Because it comes from a third party, the instrument memory consumption of this engine is not accounted for in the default value chosen for `performance_schema_max_mutex_classes`.

  If the server has insufficient resources for the plugin's instruments and you do not explicitly allocate more using `--performance_schema_max_mutex_classes=N`, loading the plugin leads to starvation of instruments.

If the value chosen for `performance_schema_max_mutex_classes` is too small, no error is reported in the error log and there is no failure at runtime. However, the content of the tables in the `performance_schema` database will miss events. The `Performance_schema_mutex_classes_lost`

status variable is the only visible sign to indicate that some events were dropped internally due to failure to create instruments.

If an instrument is not lost, it is known to the Performance Schema, and is used when instrumenting instances. For example, `wait/synch/mutex/sql/LOCK_delete` is the name of a mutex instrument in the `setup_instruments` table. This single instrument is used when creating a mutex in the code (in `THD::LOCK_delete`) however many instances of the mutex are needed as the server runs. In this case, `LOCK_delete` is a mutex that is per connection (`THD`), so if a server has 1000 connections, there are 1000 threads, and 1000 instrumented `LOCK_delete` mutex instances (`THD::LOCK_delete`).

If the server does not have room for all these 1000 instrumented mutexes (instances), some mutexes are created with instrumentation, and some are created without instrumentation. If the server can create only 800 instances, 200 instances are lost. The server continues to run, but increments `Performance_schema_mutex_instances_lost` by 200 to indicate that instances could not be created.

A value of `Performance_schema_mutex_instances_lost` greater than 0 can happen when the code initializes more mutexes at runtime than were allocated for `--performance_schema_max_mutex_instances=N`.

The bottom line is that if `SHOW STATUS LIKE 'perf%'` says that nothing was lost (all values are zero), the Performance Schema data is accurate and can be relied upon. If something was lost, the data is incomplete, and the Performance Schema could not record everything given the insufficient amount of memory it was given to use. In this case, the specific `Performance_schema_xxx_lost` variable indicates the problem area.

It might be appropriate in some cases to cause deliberate instrument starvation. For example, if you do not care about performance data for file I/O, you can start the server with all Performance Schema parameters related to file I/O set to 0. No memory will be allocated for file-related classes, instances, or handles, and all file events will be lost.

Use `SHOW ENGINE PERFORMANCE_SCHEMA STATUS` to inspect the internal operation of the Performance Schema code:

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS\G
...
*************************** 3. row ***************************
  Type: performance_schema
  Name: events_waits_history.size
Status: 76
*************************** 4. row ***************************
  Type: performance_schema
  Name: events_waits_history.count
Status: 10000
*************************** 5. row ***************************
  Type: performance_schema
  Name: events_waits_history.memory
Status: 760000
...
*************************** 57. row ***************************
  Type: performance_schema
  Name: performance_schema.memory
Status: 26459600
...
```

This statement is intended to help the DBA understand the effects that different Performance Schema options have on memory requirements. For a description of the field meanings, see Section 13.7.7.15, “SHOW ENGINE Syntax”.

26.8 Performance Schema Atom and Molecule Events

For a table I/O event, there are usually two rows in `events_waits_current`, not one. For example, a row fetch might result in rows like this:

```
Row# EVENT_NAME                 TIMER_START TIMER_END
---- ----------                 ----------- ---------
   1 wait/io/file/myisam/dfile        10001 10002
   2 wait/io/table/sql/handler        10000 NULL
```

The row fetch causes a file read. In the example, the table I/O fetch event started before the file I/O event but has not finished (its `TIMER_END` value is `NULL`). The file I/O event is “nested” within the table I/O event.

This occurs because, unlike other “atomic” wait events such as for mutexes or file I/O, table I/O events are “molecular” and include (overlap with) other events. In `events_waits_current`, the table I/O event usually has two rows:

- One row for the most recent table I/O wait event
- One row for the most recent wait event of any kind

Usually, but not always, the “of any kind” wait event differs from the table I/O event. As each subsidiary event completes, it disappears from `events_waits_current`. At this point, and until the next subsidiary event begins, the table I/O wait is also the most recent wait of any kind.

26.9 Performance Schema Tables for Current and Historical Events

For wait, stage, statement, and transaction events, the Performance Schema can monitor and store current events. In addition, when events end, the Performance Schema can store them in history tables. For each event type, the Performance Schema uses three tables for storing current and historical events. The tables have names of the following forms, where *`xxx`* indicates the event type (`waits`, `stages`, `statements`, `transactions`):

- `events_`*`xxx`*`_current`: The “current events” table stores the current monitored event for each thread (one row per thread).
- `events_`*`xxx`*`_history`: The “recent history” table stores the most recent events that have ended per thread (up to a maximum number of rows per thread).
- `events_`*`xxx`*`_history_long`: The “long history” table stores the most recent events that have ended globally (across all threads, up to a maximum number of rows per table).

The `_current` table for each event type contains one row per thread, so there is no system variable for configuring its maximum size. The Performance Schema autosizes the history tables, or the sizes can be configured explicitly at server startup using table-specific system variables, as indicated in the sections that describe the individual history tables. Typical autosized values are 10 rows per thread for `_history` tables, and 10,000 rows total for `_history_long` tables.

For each event type, the `_current`, `_history`, and `_history_long` tables have the same columns. The `_current` and `_history` tables have the same indexing. The `_history_long` table has no indexing.

The `_current` tables show what is currently happening within the server. When a current event ends, it is removed from its `_current` table.

The `_history` and `_history_long` tables show what has happened in the recent past. When the history tables become full, old events are discarded as new events are added. Rows expire from the `_history` and `_history_long` tables in different ways because the tables serve different purposes:

- `_history` is meant to investigate individual threads, independently of the global server load.
- `_history_long` is meant to investigate the server globally, not each thread.

The difference between the two types of history tables relates to the data retention policy. Both tables contains the same data when an event is first seen. However, data within each table expires differently over time, so that data might be preserved for a longer or shorter time in each table:

- For `_history`, when the table contains the maximum number of rows for a given thread, the oldest thread row is discarded when a new row for that thread is added.
- For `_history_long`, when the table becomes full, the oldest row is discarded when a new row is added, regardless of which thread generated either row.

When a thread ends, all its rows are discarded from the `_history` table but not from the `_history_long` table.

The following example illustrates the differences in how events are added to and discarded from the two types of history tables. The principles apply equally to all event types. The example is based on these assumptions:

- The Performance Schema is configured to retain 10 rows per thread in the `_history` table and 10,000 rows total in the `_history_long` table.
- Thread A generates 1 event per second.

  Thread B generates 100 events per second.
- No other threads are running.

After 5 seconds of execution:

- A and B have generated 5 and 500 events, respectively.
- `_history` contains 5 rows for A and 10 rows for B. Because storage per thread is limited to 10 rows, no rows have been discarded for A, whereas 490 rows have been discarded for B.
- `_history_long` contains 5 rows for A and 500 rows for B. Because the table has a maximum size of 10,000 rows, no rows have been discarded for either thread.

After 5 minutes (300 seconds) of execution:

- A and B have generated 300 and 30,000 events, respectively.
- `_history` contains 10 rows for A and 10 rows for B. Because storage per thread is limited to 10 rows, 290 rows have been discarded for A, whereas 29,990 rows have been discarded for B. Rows for A include data up to 10 seconds old, whereas rows for B include data up to only .1 seconds old.
- `_history_long` contains 10,000 rows. Because A and B together generate 101 events per second, the table contains data up to approximately 10,000/101 = 99 seconds old, with a mix of rows approximately 100 to 1 from B as opposed to A.

26.10 Performance Schema Statement Digests and Sampling

The MySQL server is capable of maintaining statement digest information. The digesting process converts each SQL statement to normalized form (the statement digest) and computes a SHA-256 hash value (the digest hash value) from the normalized result. Normalization permits statements that are similar to be grouped and summarized to expose information about the types of statements the server is executing and

how often they occur. For each digest, a representative statement that produces the digest is stored as a sample. This section describes how statement digesting and sampling occur and how they can be useful.

Digesting occurs in the parser regardless of whether the Performance Schema is available, so that other server components such as query rewrite plugins have access to statement digests.

- Statement Digest General Concepts
- Statement Digests in the Performance Schema
- Statement Digest Memory Use
- Statement Sampling

Statement Digest General Concepts

When the parser receives an SQL statement, it computes a statement digest if that digest is needed, which is true if any of the following conditions are true:

- Performance Schema digest instrumentation is enabled
- A Query Rewrite Plugin is enabled

The parser is also used by the `STATEMENT_DIGEST_TEXT()` and `STATEMENT_DIGEST()` functions, which applications can call to compute a normalized statement digest and a digest hash value, respectively, from an SQL statement.

The `max_digest_length` system variable value determines the maximum number of bytes available per session for computation of normalized statement digests. Once that amount of space is used during digest computation, truncation occurs: no further tokens from a parsed statement are collected or figure into its digest value. Statements that differ only after that many bytes of parsed tokens produce the same normalized statement digest and are considered identical if compared or if aggregated for digest statistics.

After the normalized statement has been computed, a SHA-256 hash value is computed from it. In addition:

- If any Query Rewrite Plugin is enabled, it is called and the statement digest and digest value are available to it.
- If the Performance Schema has digest instrumentation enabled, it makes a copy of the normalized statement digest, allocating a maximum of `performance_schema_max_digest_length` bytes for it. Consequently, if `performance_schema_max_digest_length` is less than `max_digest_length`, the copy is truncated relative to the original. The copy of the normalized statement digest is stored in the appropriate Performance Schema tables, along with the SHA-256 hash value computed from the original normalized statement. (If the Performance Schema truncates its copy of the normalized statement digest relative to the original, it does not recompute the SHA-256 hash value.)

Statement normalization transforms the statement text to a more standardized digest string representation that preserves the general statement structure while removing information not essential to the structure:

- Object identifiers such as database and table names are preserved.
- Literal values are converted to parameter markers. A normalized statement does not retain information such as names, passwords, dates, and so forth.
- Comments are removed and whitespace is adjusted.

Consider these statements:

```
SELECT * FROM orders WHERE customer_id=10 AND quantity>20
SELECT * FROM orders WHERE customer_id = 20 AND quantity > 100
```

To normalize these statements, the parser replaces data values by ? and adjusts whitespace. Both statements yield the same normalized form and thus are considered “the same”:

```
SELECT * FROM orders WHERE customer_id = ? AND quantity > ?
```

The normalized statement contains less information but is still representative of the original statement. Other similar statements that have different data values have the same normalized form.

Now consider these statements:

```
SELECT * FROM customers WHERE customer_id = 1000
SELECT * FROM orders WHERE customer_id = 1000
```

In this case, the normalized statements differ because the object identifiers differ:

```
SELECT * FROM customers WHERE customer_id = ?
SELECT * FROM orders WHERE customer_id = ?
```

If normalization produces a statement that exceeds the space available in the digest buffer (as determined by `max_digest_length`), truncation occurs and the text ends with “...”. Long normalized statements that differ only in the part that occurs following the “...” are considered the same. Consider these statements:

```
SELECT * FROM mytable WHERE cola = 10 AND colb = 20
SELECT * FROM mytable WHERE cola = 10 AND colc = 20
```

If the cutoff happens to be right after the `AND`, both statements have this normalized form:

```
SELECT * FROM mytable WHERE cola = ? AND ...
```

In this case, the difference in the second column name is lost and both statements are considered the same.

Statement Digests in the Performance Schema

In the Performance Schema, statement digesting involves these components:

- A `statements_digest` consumer in the `setup_consumers` table controls whether the Performance Schema maintains digest information. See Statement Digest Consumer.
- The statement event tables (`events_statements_current`, `events_statements_history`, and `events_statements_history_long`) have columns for storing normalized statement digests and the corresponding digest SHA-256 hash values:
  - `DIGEST_TEXT` is the text of the normalized statement digest. This is a copy of the original normalized statement that was computed to a maximum of `max_digest_length` bytes, further truncated as necessary to `performance_schema_max_digest_length` bytes.
  - `DIGEST` is the digest SHA-256 hash value computed from the original normalized statement.

  See Section 26.12.6, “Performance Schema Statement Event Tables”.
- The `events_statements_summary_by_digest` summary table provides aggregated statement digest information. This table aggregates information for statements per `SCHEMA_NAME` and `DIGEST`

combination. The Performance Schema uses SHA-256 hash values for aggregation because they are fast to compute and have a favorable statistical distribution that minimizes collisions. See Section 26.12.17.3, “Statement Summary Tables”.

Some Performance Tables have a column that stores original SQL statements from which digests are computed:

- The `SQL_TEXT` column of the `events_statements_current`, `events_statements_history`, and `events_statements_history_long` statement event tables.
- The `QUERY_SAMPLE_TEXT` column of the `events_statements_summary_by_digest` summary table.

The maximum space available for statement display is 1024 bytes by default. To change this value, set the `performance_schema_max_sql_text_length` system variable at server startup. Changes affect the storage required for all the columns just named.

The `performance_schema_max_digest_length` system variable determines the maximum number of bytes available per statement for digest value storage in the Performance Schema. However, the display length of statement digests may be longer than the available buffer size due to internal encoding of statement components such as keywords and literal values. Consequently, values selected from the `DIGEST_TEXT` column of statement event tables may appear to exceed the `performance_schema_max_digest_length` value.

The `events_statements_summary_by_digest` summary table provides a profile of the statements executed by the server. It shows what kinds of statements an application is executing and how often. An application developer can use this information together with other information in the table to assess the application's performance characteristics. For example, table columns that show wait times, lock times, or index use may highlight types of queries that are inefficient. This gives the developer insight into which parts of the application need attention.

The `events_statements_summary_by_digest` summary table has a fixed size. By default the Performance Schema estimates the size to use at startup. To specify the table size explicitly, set the `performance_schema_digests_size` system variable at server startup. If the table becomes full, the Performance Schema groups statements that have `SCHEMA_NAME` and `DIGEST` values not matching existing values in the table in a special row with `SCHEMA_NAME` and `DIGEST` set to `NULL`. This permits all statements to be counted. However, if the special row accounts for a significant percentage of the statements executed, it might be desirable to increase the summary table size by increasing `performance_schema_digests_size`.

Statement Digest Memory Use

For applications that generate very long statements that differ only at the end, increasing `max_digest_length` enables computation of digests that distinguish statements that would otherwise aggregate to the same digest. Conversely, decreasing `max_digest_length` causes the server to devote less memory to digest storage but increases the likelihood of longer statements aggregating to the same digest. Administrators should keep in mind that larger values result in correspondingly increased memory requirements, particularly for workloads that involve large numbers of simultaneous sessions (the server allocates `max_digest_length` bytes per session).

As described previously, normalized statement digests as computed by the parser are constrained to a maximum of `max_digest_length` bytes, whereas normalized statement digests stored in the Performance Schema use `performance_schema_max_digest_length` bytes. The following memory-use considerations apply regarding the relative values of `max_digest_length` and `performance_schema_max_digest_length`:

- If `max_digest_length` is less than `performance_schema_max_digest_length`:
  - Server components other than the Performance Schema use normalized statement digests that take up to `max_digest_length` bytes.
  - The Performance Schema does not further truncate normalized statement digests that it stores, but allocates more memory than `max_digest_length` bytes per digest, which is unnecessary.
- If `max_digest_length` equals `performance_schema_max_digest_length`:
  - Server components other than the Performance Schema use normalized statement digests that take up to `max_digest_length` bytes.
  - The Performance Schema does not further truncate normalized statement digests that it stores, and allocates the same amount of memory as `max_digest_length` bytes per digest.
- If `max_digest_length` is greater than `performance_schema_max_digest_length`:
  - Server components other than the Performance Schema use normalized statement digests that take up to `max_digest_length` bytes.
  - The Performance Schema further truncates normalized statement digests that it stores, and allocates less memory than `max_digest_length` bytes per digest.

Because the Performance Schema statement event tables might store many digests, setting `performance_schema_max_digest_length` smaller than `max_digest_length` enables administrators to balance these factors:

- The need to have long normalized statement digests available for server components outside the Performance Schema
- Many concurrent sessions, each of which allocates digest-computation memory
- The need to limit memory consumption by the Performance Schema statement event tables when storing many statement digests

The `performance_schema_max_digest_length` setting is not per session, it is per statement, and a session can store multiple statements in the `events_statements_history` table. A typical number of statements in this table is 10 per session, so each session will consume 10 times the memory indicated by the `performance_schema_max_digest_length` value, for this table alone.

Also, there are many statements (and digests) collected globally, most notably in the `events_statements_history_long` table. Here, too, *`N`* statements stored will consume *`N`* times the memory indicated by the `performance_schema_max_digest_length` value.

To assess the amount of memory used for SQL statement storage and digest computation, use the `SHOW ENGINE PERFORMANCE_SCHEMA STATUS` statement, or monitor these instruments:

```
mysql> SELECT NAME
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%.sqltext';
+---------------------------------------------------------------------+
| NAME                                                                |
+---------------------------------------------------------------------+
| memory/performance_schema/events_statements_history.sqltext      |
| memory/performance_schema/events_statements_current.sqltext      |
| memory/performance_schema/events_statements_history_long.sqltext |
+---------------------------------------------------------------------+
```

```
mysql> SELECT NAME
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'memory/performance_schema/%.tokens';
+-----------------------------------------------------------------------+
| NAME                                                                  |
+-----------------------------------------------------------------------+
| memory/performance_schema/events_statements_history.tokens            |
| memory/performance_schema/events_statements_current.tokens            |
| memory/performance_schema/events_statements_summary_by_digest.tokens |
| memory/performance_schema/events_statements_history_long.tokens       |
+-----------------------------------------------------------------------+
```

Statement Sampling

The Performance Schema uses statement sampling to collect representative statements that produce each digest value in the `events_statements_summary_by_digest` table. These columns store sample statement information: `QUERY_SAMPLE_TEXT` (the text of the statement), `QUERY_SAMPLE_SEEN` (when the statement was seen), and `QUERY_SAMPLE_TIMER_WAIT` (the statement wait or execution time). The Performance Schema updates all three columns each time it chooses a sample statement.

When a new table row is inserted, the statement that produced the row digest value is stored as the current sample statement associated with the digest. Thereafter, when the server sees other statements with the same digest value, it determines whether to use the new statement to replace the current sample statement (that is, whether to resample). Resampling policy is based on the comparative wait times of the current sample statement and new statement and, optionally, the age of the current sample statement:

- Resampling based on wait times: If the new statement wait time has a wait time greater than that of the current sample statement, it becomes the current sample statement.
- Resampling based on age: If the `performance_schema_max_digest_sample_age` system variable has a value greater than zero and the current sample statement is more than that many seconds old, the current statement is considered “too old” and the new statement replaces it. This occurs even if the new statement wait time is less than that of the current sample statement.

By default, `performance_schema_max_digest_sample_age` is 60 seconds (1 minute). To change how quickly sample statements “expire” due to age, increase or decrease the value. To disable the age-based part of the resampling policy, set `performance_schema_max_digest_sample_age` to 0.

26.11 Performance Schema General Table Characteristics

The name of the `performance_schema` database is lowercase, as are the names of tables within it. Queries should specify the names in lowercase.

Many tables in the `performance_schema` database are read only and cannot be modified:

```
mysql> TRUNCATE TABLE performance_schema.setup_instruments;
ERROR 1683 (HY000): Invalid performance_schema usage.
```

Some of the setup tables have columns that can be modified to affect Performance Schema operation; some also permit rows to be inserted or deleted. Truncation is permitted to clear collected events, so `TRUNCATE TABLE` can be used on tables containing those kinds of information, such as tables named with a prefix of `events_waits_`.

Summary tables can be truncated with `TRUNCATE TABLE`. Generally, the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. This enables you to clear collected values and restart aggregation. That might be useful, for example, after you have made a runtime configuration change. Exceptions to this truncation behavior are noted in individual summary table sections.

Privileges are as for other databases and tables:

- To retrieve from `performance_schema` tables, you must have the `SELECT` privilege.
- To change those columns that can be modified, you must have the `UPDATE` privilege.
- To truncate tables that can be truncated, you must have the `DROP` privilege.

Because only a limited set of privileges apply to Performance Schema tables, attempts to use `GRANT ALL` as shorthand for granting privileges at the database or table leval fail with an error:

```
mysql> GRANT ALL ON performance_schema.*
       TO 'u1'@'localhost';
ERROR 1044 (42000): Access denied for user 'root'@'localhost'
to database 'performance_schema'
mysql> GRANT ALL ON performance_schema.setup_instruments
       TO 'u2'@'localhost';
ERROR 1044 (42000): Access denied for user 'root'@'localhost'
to database 'performance_schema'
```

Instead, grant exactly the desired privileges:

```
mysql> GRANT SELECT ON performance_schema.*
       TO 'u1'@'localhost';
Query OK, 0 rows affected (0.03 sec)

mysql> GRANT SELECT, UPDATE ON performance_schema.setup_instruments
       TO 'u2'@'localhost';
Query OK, 0 rows affected (0.02 sec)
```

26.12 Performance Schema Table Descriptions

Tables in the `performance_schema` database can be grouped as follows:

- Setup tables. These tables are used to configure and display monitoring characteristics.
- Current events tables. The `events_waits_current` table contains the most recent event for each thread. Other similar tables contain current events at different levels of the event hierarchy: `events_stages_current` for stage events, `events_statements_current` for statement events, and `events_transactions_current` for transaction events.
- History tables. These tables have the same structure as the current events tables, but contain more rows. For example, for wait events, `events_waits_history` table contains the most recent 10 events per thread. `events_waits_history_long` contains the most recent 10,000 events. Other similar tables exist for stage, statement, and transaction histories.

  To change the sizes of the history tables, set the appropriate system variables at server startup. For example, to set the sizes of the wait event history tables, set `performance_schema_events_waits_history_size` and `performance_schema_events_waits_history_long_size`.
- Summary tables. These tables contain information aggregated over groups of events, including those that have been discarded from the history tables.
- Instance tables. These tables document what types of objects are instrumented. An instrumented object, when used by the server, produces an event. These tables provide event names and explanatory notes or status information.
- Miscellaneous tables. These do not fall into any of the other table groups.

26.12.1 Performance Schema Table Index

The following table lists each Performance Schema table and provides a short description of each one.

Table 26.1 Performance Schema Tables

Table Name	Description
accounts	Connection statistics per client account
clone_progress	Clone operation progress
clone_status	Clone operation status
cond_instances	synchronization object instances
data_lock_waits	Data lock wait relationships
data_locks	Data locks held and requested
events_errors_summary_by_account_by_error	Errors per error code and account
events_errors_summary_by_host_by_error	Errors per error code and host
events_errors_summary_by_thread_by_error	Errors per error code and host
events_errors_summary_by_user_by_error	Errors per error code and host
events_errors_summary_global_by_error	Errors per error code
events_stages_current	Current stage events
events_stages_history	Most recent stage events per thread
events_stages_history_long	Most recent stage events overall
events_stages_summary_by_account_by_event_name	Stage events per account and event name
events_stages_summary_by_host_by_event_name	Stage events per host name and event name
events_stages_summary_by_thread_by_event_name	Stage waits per thread and event name
events_stages_summary_by_user_by_event_name	Stage events per user name and event name
events_stages_summary_global_by_event_name	Stage waits per event name
events_statements_current	Current statement events
events_statements_histogram_by_digest	Statement histograms per schema and digest value
events_statements_histogram_global	Statement histogram summarized globally
events_statements_history	Most recent statement events per thread
events_statements_history_long	Most recent statement events overall
events_statements_summary_by_account_by_event_name	Statement events per account and event name
events_statements_summary_by_digest	Statement events per schema and digest value
events_statements_summary_by_host_by_event_name	Statement events per host name and event name
events_statements_summary_by_program	Statement events per stored program
events_statements_summary_by_thread_by_event_name	Statement events per thread and event name
events_statements_summary_by_user_by_event_name	Statement events per user name and event name
events_statements_summary_global_by_event_name	Statement events per event name
events_transactions_current	Current transaction events

Table Name	Description
events_transactions_history	Most recent transaction events per thread
events_transactions_history_long	Most recent transaction events overall
events_transactions_summary_by_account_by_event_name	Transaction events per account and event name
events_transactions_summary_by_host_by_event_name	Transaction events per host name and event name
events_transactions_summary_by_thread_by_event_name	Transaction events per thread and event name
events_transactions_summary_by_user_by_event_name	Transaction events per user name and event name
events_transactions_summary_global_by_event_name	Transaction events per event name
events_waits_current	Current wait events
events_waits_history	Most recent wait events per thread
events_waits_history_long	Most recent wait events overall
events_waits_summary_by_account_by_event_name	Wait events per account and event name
events_waits_summary_by_host_by_event_name	Wait events per host name and event name
events_waits_summary_by_instance	Wait events per instance
events_waits_summary_by_thread_by_event_name	Wait events per thread and event name
events_waits_summary_by_user_by_event_name	Wait events per user name and event name
events_waits_summary_global_by_event_name	Wait events per event name
file_instances	File instances
file_summary_by_event_name	File events per event name
file_summary_by_instance	File events per file instance
global_status	Global status variables
global_variables	Global system variables
host_cache	Information from the internal host cache
hosts	Connection statistics per client host name
keyring_keys	Metadata for keyring keys
log_status	Information about server logs for backup purposes
memory_summary_by_account_by_event_name	Memory operations per account and event name
memory_summary_by_host_by_event_name	Memory operations per host and event name
memory_summary_by_thread_by_event_name	Memory operations per thread and event name
memory_summary_by_user_by_event_name	Memory operations per user and event name
memory_summary_global_by_event_name	Memory operations globally per event name
metadata_locks	Metadata locks and lock requests
mutex_instances	Mutex synchronization object instances
objects_summary_global_by_type	Object summaries
performance_timers	Which event timers are available

Table Name	Description
persisted_variables	Contents of mysqld-auto.cnf file
prepared_statements_instances	Prepared statement instances and statistics
replication_applier_configuration	Configuration parameters for the transaction applier on the slave
replication_applier_status	Current status of the transaction applier on the slave
replication_applier_status_by_coordinator	SQL or coordinator thread applier status
replication_applier_status_by_worker	Worker thread applier status (empty unless slave is multithreaded)
replication_connection_configuration	Configuration parameters for connecting to the master
replication_connection_status	Current status of the connection to the master
rwlock_instances	Lock synchronization object instances
session_account_connect_attrs	Connection attributes per for the current session
session_connect_attrs	Connection attributes for all sessions
session_status	Status variables for current session
session_variables	System variables for current session
setup_actors	How to initialize monitoring for new foreground threads
setup_consumers	Consumers for which event information can be stored
setup_instruments	Classes of instrumented objects for which events can be collected
setup_objects	Which objects should be monitored
setup_threads	Instrumented thread names and attributes
socket_instances	Active connection instances
socket_summary_by_event_name	Socket waits and I/O per event name
socket_summary_by_instance	Socket waits and I/O per instance
status_by_account	Session status variables per account
status_by_host	Session status variables per host name
status_by_thread	Session status variables per session
status_by_user	Session status variables per user name
table_handles	Table locks and lock requests
table_io_waits_summary_by_index_usage	Table I/O waits per index
table_io_waits_summary_by_table	Table I/O waits per table
table_lock_waits_summary_by_table	Table lock waits per table
threads	Information about server threads
tp_thread_group_state	Information about thread pool thread group states

Table Name	Description
tp_thread_group_stats	Thread group statistics
tp_thread_state	Information about thread pool thread states
user_defined_functions	Registered user-defined functions
user_variables_by_thread	User-defined variables per thread
users	Connection statistics per client user name
variables_by_thread	Session system variables per session
variables_info	How system variables were most recently set

26.12.2 Performance Schema Setup Tables

The setup tables provide information about the current instrumentation and enable the monitoring configuration to be changed. For this reason, some columns in these tables can be changed if you have the UPDATE privilege.

The use of tables rather than individual variables for setup information provides a high degree of flexibility in modifying Performance Schema configuration. For example, you can use a single statement with standard SQL syntax to make multiple simultaneous configuration changes.

These setup tables are available:

- setup_actors: How to initialize monitoring for new foreground threads
- setup_consumers: The destinations to which event information can be sent and stored
- setup_instruments: The classes of instrumented objects for which events can be collected
- setup_objects: Which objects should be monitored
- setup_threads: Instrumented thread names and attributes

26.12.2.1 The setup_actors Table

The setup_actors table contains information that determines whether to enable monitoring and historical event logging for new foreground server threads (threads associated with client connections). This table has a maximum size of 100 rows by default. To change the table size, modify the performance_schema_setup_actors_size system variable at server startup.

For each new foreground thread, the Performance Schema matches the user and host for the thread against the rows of the setup_actors table. If a row from that table matches, its ENABLED and HISTORY column values are used to set the INSTRUMENTED and HISTORY columns, respectively, of the threads table row for the thread. This enables instrumenting and historical event logging to be applied selectively per host, user, or account (user and host combination). If there is no match, the INSTRUMENTED and HISTORY columns for the thread are set to NO.

For background threads, there is no associated user. INSTRUMENTED and HISTORY are YES by default and setup_actors is not consulted.

The initial contents of the setup_actors table match any user and host combination, so monitoring and historical event collection are enabled by default for all foreground threads:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+------+------+------+---------+---------+
| HOST | USER | ROLE | ENABLED | HISTORY |
+------+------+------+---------+---------+
```

```
| %    | %    | %    | YES     | YES     |
+------+------+------+---------+---------+
```

For information about how to use the `setup_actors` table to affect event monitoring, see Section 26.4.6, “Pre-Filtering by Thread”.

Modifications to the `setup_actors` table affect only foreground threads created subsequent to the modification, not existing threads. To affect existing threads, modify the `INSTRUMENTED` and `HISTORY` columns of `threads` table rows.

The `setup_actors` table has these columns:

- `HOST`

  The host name. This should be a literal name, or `'%'` to mean “any host.”

- `USER`

  The user name. This should be a literal name, or `'%'` to mean “any user.”

- `ROLE`

  Unused.

- `ENABLED`

  Whether to enable instrumentation for foreground threads matched by the row. The value is `YES` or `NO`.

- `HISTORY`

  Whether to log historical events for foreground threads matched by the row. The value is `YES` or `NO`.

The `setup_actors` table has these indexes:

- Primary key on (`HOST`, `USER`, `ROLE`)

`TRUNCATE TABLE` is permitted for the `setup_actors` table. It removes the rows.

26.12.2.2 The setup_consumers Table

The `setup_consumers` table lists the types of consumers for which event information can be stored and which are enabled:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | YES     |
| events_statements_history_long | NO      |
| events_transactions_current    | YES     |
| events_transactions_history    | YES     |
| events_transactions_history_long | NO    |
| events_waits_current           | NO      |
| events_waits_history           | NO      |
| events_waits_history_long      | NO      |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
```

```
+---------------------------------+---------+
```

The consumer settings in the `setup_consumers` table form a hierarchy from higher levels to lower. For detailed information about the effect of enabling different consumers, see Section 26.4.7, “Pre-Filtering by Consumer”.

Modifications to the `setup_consumers` table affect monitoring immediately.

The `setup_consumers` table has these columns:

- `NAME`

  The consumer name.

- `ENABLED`

  Whether the consumer is enabled. The value is `YES` or `NO`. This column can be modified. If you disable a consumer, the server does not spend time adding event information to it.

The `setup_consumers` table has these indexes:

- Primary key on (`NAME`)

`TRUNCATE TABLE` is not permitted for the `setup_consumers` table.

26.12.2.3 The setup_instruments Table

The `setup_instruments` table lists classes of instrumented objects for which events can be collected:

```
mysql> SELECT * FROM performance_schema.setup_instruments\G
*************************** 1. row ***************************
         NAME: wait/synch/mutex/pfs/LOCK_pfs_share_list
      ENABLED: NO
        TIMED: NO
   PROPERTIES: singleton
   VOLATILITY: 1
DOCUMENTATION: Components can provide their own performance_schema tables.
               This lock protects the list of such tables definitions.
...
*************************** 369. row ***************************
         NAME: stage/sql/executing
      ENABLED: NO
        TIMED: NO
   PROPERTIES:
   VOLATILITY: 0
DOCUMENTATION: NULL
...
*************************** 687. row ***************************
         NAME: statement/abstract/Query
      ENABLED: YES
        TIMED: YES
   PROPERTIES: mutable
   VOLATILITY: 0
DOCUMENTATION: SQL query just received from the network. At this point,
               the real statement type is unknown, the type will be
               refined after SQL parsing.
...
*************************** 696. row ***************************
         NAME: memory/performance_schema/metadata_locks
      ENABLED: YES
        TIMED: NULL
   PROPERTIES: global_statistics
   VOLATILITY: 1
```

```
DOCUMENTATION: Memory used for table performance_schema.metadata_locks
...
```

Each instrument added to the source code provides a row for the `setup_instruments` table, even when the instrumented code is not executed. When an instrument is enabled and executed, instrumented instances are created, which are visible in the *`xxx`*`_instances` tables, such as `file_instances` or `rwlock_instances`.

Modifications to most `setup_instruments` rows affect monitoring immediately. For some instruments, modifications are effective only at server startup; changing them at runtime has no effect. This affects primarily mutexes, conditions, and rwlocks in the server, although there may be other instruments for which this is true.

For more information about the role of the `setup_instruments` table in event filtering, see Section 26.4.3, “Event Pre-Filtering”.

The `setup_instruments` table has these columns:

- `NAME`

  The instrument name. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 26.6, “Performance Schema Instrument Naming Conventions”. Events produced from execution of an instrument have an `EVENT_NAME` value that is taken from the instrument `NAME` value. (Events do not really have a “name,” but this provides a way to associate events with instruments.)

- `ENABLED`

  Whether the instrument is enabled. The value is `YES` or `NO`. A disabled instrument produces no events. This column can be modified, although setting `ENABLED` has no effect for instruments that have already been created.

- `TIMED`

  Whether the instrument is timed. The value is `YES`, `NO`, or `NULL`. This column can be modified, although setting `TIMED` has no effect for instruments that have already been created.

  A `TIMED` value of `NULL` indicates that the instrument does not support timing. For example, memory operations are not timed, so their `TIMED` column is `NULL`.

  Setting `TIMED` to `NULL` for an instrument that supports timing has no effect, as does setting `TIMED` to non-`NULL` for an instrument that does not support timing.

  If an enabled instrument is not timed, the instrument code is enabled, but the timer is not. Events produced by the instrument have `NULL` for the `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` timer values. This in turn causes those values to be ignored when calculating the sum, minimum, maximum, and average time values in summary tables.

- `PROPERTIES`

  The instrument properties. This column uses the `SET` data type, so multiple flags from the following list can be set per instrument:

  - `global_statistics`: The instrument produces only global summaries. Summaries for finer levels are unavailable, such as per thread, account, user, or host. For example, most memory instruments produce only global summaries.

  - `mutable`: The instrument can “mutate” into a more specific one. This property applies only to statement instruments.

- `progress`: The instrument is capable of reporting progress data. This property applies only to stage instruments.
- `singleton`: The instrument has a single instance. For example, most global mutex locks in the server are singletons, so the corresponding instruments are as well.
- `user`: The instrument is directly related to user workload (as opposed to system workload). One such instrument is `wait/io/socket/sql/client_connection`.

- `VOLATILITY`

  The instrument volatility. Volatility values range from low to high. The values correspond to the `PSI_VOLATILITY_xxx` constants defined in the `mysql/psi/psi_base.h` header file:

```
#define PSI_VOLATILITY_UNKNOWN 0
#define PSI_VOLATILITY_PERMANENT 1
#define PSI_VOLATILITY_PROVISIONING 2
#define PSI_VOLATILITY_DDL 3
#define PSI_VOLATILITY_CACHE 4
#define PSI_VOLATILITY_SESSION 5
#define PSI_VOLATILITY_TRANSACTION 6
#define PSI_VOLATILITY_QUERY 7
#define PSI_VOLATILITY_INTRA_QUERY 8
```

  The `VOLATILITY` column is purely informational, to provide users (and the Performance Schema code) some hint about the instrument runtime behavior.

  Instruments with a low volatility index (PERMANENT = 1) are created once at server startup, and never destroyed or re-created during normal server operation. They are destroyed only during server shutdown.

  For example, the `wait/synch/mutex/pfs/LOCK_pfs_share_list` mutex is defined with a volatility of 1, which means it is created once. Possible overhead from the instrumentation itself (namely, mutex initialization) has no effect for this instrument then. Runtime overhead occurs only when locking or unlocking the mutex.

  Instruments with a higher volatility index (for example, SESSION = 5) are created and destroyed for every user session. For example, the `wait/synch/mutex/sql/THD::LOCK_query_plan` mutex is created each time a session connects, and destroyed when the session disconnects.

  This mutex is more sensitive to Performance Schema overhead, because overhead comes not only from the lock and unlock instrumentation, but also from mutex create and destroy instrumentation, which is executed more often.

  Another aspect of volatility concerns whether and when an update to the `ENABLED` column actually has some effect:

  - An update to `ENABLED` affects instrumented objects created subsequently, but has no effect on instruments already created.
  - Instruments that are more “volatile” use new settings from the `setup_instruments` table sooner.

  For example, this statement does not affect the `LOCK_query_plan` mutex for existing sessions, but does have an effect on new sessions created subsequent to the update:

```
UPDATE performance_schema.setup_instruments
```

```
SET ENABLED=value
WHERE NAME = 'wait/synch/mutex/sql/THD::LOCK_query_plan';
```

This statement actually has no effect at all:

```
UPDATE performance_schema.setup_instruments
SET ENABLED=value
WHERE NAME = 'wait/synch/mutex/pfs/LOCK_pfs_share_list';
```

This mutex is permanent, and was created already before the update is executed. The mutex is never created again, so the `ENABLED` value in `setup_instruments` is never used. To enable or disable this mutex, use the `mutex_instances` table instead.

- `DOCUMENTATION`

  A string describing the instrument purpose. The value is `NULL` if no description is available.

The `setup_instruments` table has these indexes:

- Primary key on (`NAME`)

`TRUNCATE TABLE` is not permitted for the `setup_instruments` table.

26.12.2.4 The setup_objects Table

The `setup_objects` table controls whether the Performance Schema monitors particular objects. This table has a maximum size of 100 rows by default. To change the table size, modify the `performance_schema_setup_objects_size` system variable at server startup.

The initial `setup_objects` contents look like this:

```
mysql> SELECT * FROM performance_schema.setup_objects;
+-------------+--------------------+-------------+---------+-------+
| OBJECT_TYPE | OBJECT_SCHEMA      | OBJECT_NAME | ENABLED | TIMED |
+-------------+--------------------+-------------+---------+-------+
| EVENT       | mysql              | %           | NO      | NO    |
| EVENT       | performance_schema | %           | NO      | NO    |
| EVENT       | information_schema | %           | NO      | NO    |
| EVENT       | %                  | %           | YES     | YES   |
| FUNCTION    | mysql              | %           | NO      | NO    |
| FUNCTION    | performance_schema | %           | NO      | NO    |
| FUNCTION    | information_schema | %           | NO      | NO    |
| FUNCTION    | %                  | %           | YES     | YES   |
| PROCEDURE   | mysql              | %           | NO      | NO    |
| PROCEDURE   | performance_schema | %           | NO      | NO    |
| PROCEDURE   | information_schema | %           | NO      | NO    |
| PROCEDURE   | %                  | %           | YES     | YES   |
| TABLE       | mysql              | %           | NO      | NO    |
| TABLE       | performance_schema | %           | NO      | NO    |
| TABLE       | information_schema | %           | NO      | NO    |
| TABLE       | %                  | %           | YES     | YES   |
| TRIGGER     | mysql              | %           | NO      | NO    |
| TRIGGER     | performance_schema | %           | NO      | NO    |
| TRIGGER     | information_schema | %           | NO      | NO    |
| TRIGGER     | %                  | %           | YES     | YES   |
+-------------+--------------------+-------------+---------+-------+
```

Modifications to the `setup_objects` table affect object monitoring immediately.

For object types listed in `setup_objects`, the Performance Schema uses the table to how to monitor them. Object matching is based on the `OBJECT_SCHEMA` and `OBJECT_NAME` columns. Objects for which there is no match are not monitored.

The effect of the default object configuration is to instrument all tables except those in the `mysql`, `INFORMATION_SCHEMA`, and `performance_schema` databases. (Tables in the `INFORMATION_SCHEMA` database are not instrumented regardless of the contents of `setup_objects`; the row for `information_schema.%` simply makes this default explicit.)

When the Performance Schema checks for a match in `setup_objects`, it tries to find more specific matches first. For example, with a table `db1.t1`, it looks for a match for `'db1'` and `'t1'`, then for `'db1'` and `'%'`, then for `'%'` and `'%'`. The order in which matching occurs matters because different matching `setup_objects` rows can have different `ENABLED` and `TIMED` values.

Rows can be inserted into or deleted from `setup_objects` by users with the `INSERT` or `DELETE` privilege on the table. For existing rows, only the `ENABLED` and `TIMED` columns can be modified, by users with the `UPDATE` privilege on the table.

For more information about the role of the `setup_objects` table in event filtering, see Section 26.4.3, “Event Pre-Filtering”.

The `setup_objects` table has these columns:

- `OBJECT_TYPE`

  The type of object to instrument. The value is one of `'EVENT'` (Event Scheduler event), `'FUNCTION'` (stored function), `'PROCEDURE'` (stored procedure), `'TABLE'` (base table), or `'TRIGGER'` (trigger).

  `TABLE` filtering affects table I/O events (`wait/io/table/sql/handler` instrument) and table lock events (`wait/lock/table/sql/handler` instrument).

- `OBJECT_SCHEMA`

  The schema that contains the object. This should be a literal name, or `'%'` to mean “any schema.”

- `OBJECT_NAME`

  The name of the instrumented object. This should be a literal name, or `'%'` to mean “any object.”

- `ENABLED`

  Whether events for the object are instrumented. The value is `YES` or `NO`. This column can be modified.

- `TIMED`

  Whether events for the object are timed. This column can be modified.

The `setup_objects` table has these indexes:

- Index on (`OBJECT_TYPE`, `OBJECT_SCHEMA`, `OBJECT_NAME`)

`TRUNCATE TABLE` is permitted for the `setup_objects` table. It removes the rows.

26.12.2.5 The setup_threads Table

The `setup_threads` table lists instrumented thread classes. It exposes thread class names and attributes:

```
mysql> SELECT * FROM performance_schema.setup_threads\G
*************************** 1. row ***************************
```

```
         NAME: thread/performance_schema/setup
      ENABLED: YES
      HISTORY: YES
   PROPERTIES: singleton
   VOLATILITY: 0
DOCUMENTATION: NULL
...
*************************** 4. row ***************************
         NAME: thread/sql/main
      ENABLED: YES
      HISTORY: YES
   PROPERTIES: singleton
   VOLATILITY: 0
DOCUMENTATION: NULL
*************************** 5. row ***************************
         NAME: thread/sql/one_connection
      ENABLED: YES
      HISTORY: YES
   PROPERTIES: user
   VOLATILITY: 0
DOCUMENTATION: NULL
...
*************************** 10. row ***************************
         NAME: thread/sql/event_scheduler
      ENABLED: YES
      HISTORY: YES
   PROPERTIES: singleton
   VOLATILITY: 0
DOCUMENTATION: NULL
```

The `setup_threads` table has these columns:

- `NAME`

  The instrument name. Thread instruments begin with `thread` (for example, `thread/sql/parser_service` or `thread/performance_schema/setup`).

- `ENABLED`

  Whether the instrument is enabled. The value is `YES` or `NO`. This column can be modified, although setting `ENABLED` has no effect for threads that are already running.

  For background threads, setting the `ENABLED` value controls whether `INSTRUMENTED` is set to `YES` or `NO` for threads that are subsequently created for this instrument and listed in the `threads` table. For foreground threads, this column has no effect; the `setup_actors` table takes precedence.

- `HISTORY`

  Whether to log historical events for the instrument. The value is `YES` or `NO`. This column can be modified, although setting `HISTORY` has no effect for threads that are already running.

  For background threads, setting the `HISTORY` value controls whether `HISTORY` is set to `YES` or `NO` for threads that are subsequently created for this instrument and listed in the `threads` table. For foreground threads, this column has no effect; the `setup_actors` table takes precedence.

- `PROPERTIES`

  The instrument properties. This column uses the `SET` data type, so multiple flags from the following list can be set per instrument:

  - `singleton`: The instrument has a single instance. For example, there is only one thread for the `thread/sql/main` instrument.

  - `user`: The instrument is directly related to user workload (as opposed to system workload). For example, threads such as `thread/sql/one_connection` executing a user session have the `user` property to differentiate them from system threads.

- `VOLATILITY`

  The instrument volatility. This column has the same meaning as in the `setup_instruments` table. See Section 26.12.2.3, “The setup_instruments Table”.

- `DOCUMENTATION`

  A string describing the instrument purpose. The value is `NULL` if no description is available.

The `setup_threads` table has these indexes:

- Primary key on (`NAME`)

`TRUNCATE TABLE` is not permitted for the `setup_threads` table.

26.12.2.6 The setup_timers Table

This table was removed in MySQL 8.0.4.

26.12.3 Performance Schema Instance Tables

Instance tables document what types of objects are instrumented. They provide event names and explanatory notes or status information:

- `cond_instances`: Condition synchronization object instances
- `file_instances`: File instances
- `mutex_instances`: Mutex synchronization object instances
- `rwlock_instances`: Lock synchronization object instances
- `socket_instances`: Active connection instances

These tables list instrumented synchronization objects, files, and connections. There are three types of synchronization objects: `cond`, `mutex`, and `rwlock`. Each instance table has an `EVENT_NAME` or `NAME` column to indicate the instrument associated with each row. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 26.6, “Performance Schema Instrument Naming Conventions”.

The `mutex_instances.LOCKED_BY_THREAD_ID` and `rwlock_instances.WRITE_LOCKED_BY_THREAD_ID` columns are extremely important for investigating performance bottlenecks or deadlocks. For examples of how to use them for this purpose, see Section 26.19, “Using the Performance Schema to Diagnose Problems”

26.12.3.1 The cond_instances Table

The `cond_instances` table lists all the conditions seen by the Performance Schema while the server executes. A condition is a synchronization mechanism used in the code to signal that a specific event has happened, so that a thread waiting for this condition can resume work.

When a thread is waiting for something to happen, the condition name is an indication of what the thread is waiting for, but there is no immediate way to tell which other thread, or threads, will cause the condition to happen.

The `cond_instances` table has these columns:

- `NAME`

  The instrument name associated with the condition.

- `OBJECT_INSTANCE_BEGIN`

  The address in memory of the instrumented condition.

The `cond_instances` table has these indexes:

- Primary key on (`OBJECT_INSTANCE_BEGIN`)
- Index on (`NAME`)

`TRUNCATE TABLE` is not permitted for the `cond_instances` table.

26.12.3.2 The file_instances Table

The `file_instances` table lists all the files seen by the Performance Schema when executing file I/O instrumentation. If a file on disk has never been opened, it will not be in `file_instances`. When a file is deleted from the disk, it is also removed from the `file_instances` table.

The `file_instances` table has these columns:

- `FILE_NAME`

  The file name.

- `EVENT_NAME`

  The instrument name associated with the file.

- `OPEN_COUNT`

  The count of open handles on the file. If a file was opened and then closed, it was opened 1 time, but `OPEN_COUNT` will be 0. To list all the files currently opened by the server, use `WHERE OPEN_COUNT > 0`.

The `file_instances` table has these indexes:

- Primary key on (`FILE_NAME`)
- Index on (`EVENT_NAME`)

`TRUNCATE TABLE` is not permitted for the `file_instances` table.

26.12.3.3 The mutex_instances Table

The `mutex_instances` table lists all the mutexes seen by the Performance Schema while the server executes. A mutex is a synchronization mechanism used in the code to enforce that only one thread at a given time can have access to some common resource. The resource is said to be “protected” by the mutex.

When two threads executing in the server (for example, two user sessions executing a query simultaneously) do need to access the same resource (a file, a buffer, or some piece of data), these two

threads will compete against each other, so that the first query to obtain a lock on the mutex will cause the other query to wait until the first is done and unlocks the mutex.

The work performed while holding a mutex is said to be in a “critical section,” and multiple queries do execute this critical section in a serialized way (one at a time), which is a potential bottleneck.

The `mutex_instances` table has these columns:

- `NAME`

  The instrument name associated with the mutex.

- `OBJECT_INSTANCE_BEGIN`

  The address in memory of the instrumented mutex.

- `LOCKED_BY_THREAD_ID`

  When a thread currently has a mutex locked, `LOCKED_BY_THREAD_ID` is the `THREAD_ID` of the locking thread, otherwise it is `NULL`.

The `mutex_instances` table has these indexes:

- Primary key on (`OBJECT_INSTANCE_BEGIN`)
- Index on (`NAME`)
- Index on (`LOCKED_BY_THREAD_ID`)

`TRUNCATE TABLE` is not permitted for the `mutex_instances` table.

For every mutex instrumented in the code, the Performance Schema provides the following information.

- The `setup_instruments` table lists the name of the instrumentation point, with the prefix `wait/synch/mutex/`.
- When some code creates a mutex, a row is added to the `mutex_instances` table. The `OBJECT_INSTANCE_BEGIN` column is a property that uniquely identifies the mutex.
- When a thread attempts to lock a mutex, the `events_waits_current` table shows a row for that thread, indicating that it is waiting on a mutex (in the `EVENT_NAME` column), and indicating which mutex is waited on (in the `OBJECT_INSTANCE_BEGIN` column).
- When a thread succeeds in locking a mutex:
  - `events_waits_current` shows that the wait on the mutex is completed (in the `TIMER_END` and `TIMER_WAIT` columns)
  - The completed wait event is added to the `events_waits_history` and `events_waits_history_long` tables
  - `mutex_instances` shows that the mutex is now owned by the thread (in the `THREAD_ID` column).
- When a thread unlocks a mutex, `mutex_instances` shows that the mutex now has no owner (the `THREAD_ID` column is `NULL`).
- When a mutex object is destroyed, the corresponding row is removed from `mutex_instances`.

By performing queries on both of the following tables, a monitoring application or a DBA can detect bottlenecks or deadlocks between threads that involve mutexes:

- `events_waits_current`, to see what mutex a thread is waiting for
- `mutex_instances`, to see which other thread currently owns a mutex

26.12.3.4 The rwlock_instances Table

The `rwlock_instances` table lists all the rwlock (read write lock) instances seen by the Performance Schema while the server executes. An `rwlock` is a synchronization mechanism used in the code to enforce that threads at a given time can have access to some common resource following certain rules. The resource is said to be "protected" by the `rwlock`. The access is either shared (many threads can have a read lock at the same time), exclusive (only one thread can have a write lock at a given time), or shared-exclusive (a thread can have a write lock while permitting inconsistent reads by other threads). Shared-exclusive access is otherwise known as an `sxlock` and optimizes concurrency and improves scalability for read-write workloads.

Depending on how many threads are requesting a lock, and the nature of the locks requested, access can be either granted in shared mode, exclusive mode, shared-exclusive mode or not granted at all, waiting for other threads to finish first.

The `rwlock_instances` table has these columns:

- `NAME`

  The instrument name associated with the lock.

- `OBJECT_INSTANCE_BEGIN`

  The address in memory of the instrumented lock.

- `WRITE_LOCKED_BY_THREAD_ID`

  When a thread currently has an `rwlock` locked in exclusive (write) mode, `WRITE_LOCKED_BY_THREAD_ID` is the `THREAD_ID` of the locking thread, otherwise it is `NULL`.

- `READ_LOCKED_BY_COUNT`

  When a thread currently has an `rwlock` locked in shared (read) mode, `READ_LOCKED_BY_COUNT` is incremented by 1. This is a counter only, so it cannot be used directly to find which thread holds a read lock, but it can be used to see whether there is a read contention on an `rwlock`, and see how many readers are currently active.

The `rwlock_instances` table has these indexes:

- Primary key on (`OBJECT_INSTANCE_BEGIN`)
- Index on (`NAME`)
- Index on (`WRITE_LOCKED_BY_THREAD_ID`)

`TRUNCATE TABLE` is not permitted for the `rwlock_instances` table.

By performing queries on both of the following tables, a monitoring application or a DBA may detect some bottlenecks or deadlocks between threads that involve locks:

- `events_waits_current`, to see what `rwlock` a thread is waiting for

- `rwlock_instances`, to see which other thread currently owns an `rwlock`

There is a limitation: The `rwlock_instances` can be used only to identify the thread holding a write lock, but not the threads holding a read lock.

26.12.3.5 The socket_instances Table

The `socket_instances` table provides a real-time snapshot of the active connections to the MySQL server. The table contains one row per TCP/IP or Unix socket file connection. Information available in this table provides a real-time snapshot of the active connections to the server. (Additional information is available in socket summary tables, including network activity such as socket operations and number of bytes transmitted and received; see Section 26.12.17.9, "Socket Summary Tables").

```
mysql> SELECT * FROM performance_schema.socket_instances\G
*************************** 1. row ***************************
           EVENT_NAME: wait/io/socket/sql/server_unix_socket
OBJECT_INSTANCE_BEGIN: 4316619408
            THREAD_ID: 1
            SOCKET_ID: 16
                   IP:
                 PORT: 0
                STATE: ACTIVE
*************************** 2. row ***************************
           EVENT_NAME: wait/io/socket/sql/client_connection
OBJECT_INSTANCE_BEGIN: 4316644608
            THREAD_ID: 21
            SOCKET_ID: 39
                   IP: 127.0.0.1
                 PORT: 55233
                STATE: ACTIVE
*************************** 3. row ***************************
           EVENT_NAME: wait/io/socket/sql/server_tcpip_socket
OBJECT_INSTANCE_BEGIN: 4316699040
            THREAD_ID: 1
            SOCKET_ID: 14
                   IP: 0.0.0.0
                 PORT: 50603
                STATE: ACTIVE
```

Socket instruments have names of the form `wait/io/socket/sql/socket_type` and are used like this:

1. The server has a listening socket for each network protocol that it supports. The instruments associated with listening sockets for TCP/IP or Unix socket file connections have a *`socket_type`* value of `server_tcpip_socket` or `server_unix_socket`, respectively.
2. When a listening socket detects a connection, the server transfers the connection to a new socket managed by a separate thread. The instrument for the new connection thread has a *`socket_type`* value of `client_connection`.
3. When a connection terminates, the row in `socket_instances` corresponding to it is deleted.

The `socket_instances` table has these columns:

- `EVENT_NAME`

  The name of the `wait/io/socket/*` instrument that produced the event. This is a `NAME` value from the `setup_instruments` table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 26.6, "Performance Schema Instrument Naming Conventions".

- `OBJECT_INSTANCE_BEGIN`

This column uniquely identifies the socket. The value is the address of an object in memory.

- `THREAD_ID`

  The internal thread identifier assigned by the server. Each socket is managed by a single thread, so each socket can be mapped to a thread which can be mapped to a server process.

- `SOCKET_ID`

  The internal file handle assigned to the socket.

- `IP`

  The client IP address. The value may be either an IPv4 or IPv6 address, or blank to indicate a Unix socket file connection.

- `PORT`

  The TCP/IP port number, in the range from 0 to 65535.

- `STATE`

  The socket status, either `IDLE` or `ACTIVE`. Wait times for active sockets are tracked using the corresponding socket instrument. Wait times for idle sockets are tracked using the `idle` instrument.

  A socket is idle if it is waiting for a request from the client. When a socket becomes idle, the event row in `socket_instances` that is tracking the socket switches from a status of `ACTIVE` to `IDLE`. The `EVENT_NAME` value remains `wait/io/socket/*`, but timing for the instrument is suspended. Instead, an event is generated in the `events_waits_current` table with an `EVENT_NAME` value of `idle`.

  When the next request is received, the `idle` event terminates, the socket instance switches from `IDLE` to `ACTIVE`, and timing of the socket instrument resumes.

The `socket_instances` table has these indexes:

- Primary key on (`OBJECT_INSTANCE_BEGIN`)
- Index on (`THREAD_ID`)
- Index on (`SOCKET_ID`)
- Index on (`IP`, `PORT`)

`TRUNCATE TABLE` is not permitted for the `socket_instances` table.

The `IP:PORT` column combination value identifies the connection. This combination value is used in the `OBJECT_NAME` column of the `events_waits_xxx` tables, to identify the connection from which socket events come:

- For the Unix domain listener socket (`server_unix_socket`), the port is 0, and the IP is `''`.
- For client connections via the Unix domain listener (`client_connection`), the port is 0, and the IP is `''`.
- For the TCP/IP server listener socket (`server_tcpip_socket`), the port is always the master port (for example, 3306), and the IP is always `0.0.0.0`.
- For client connections via the TCP/IP listener (`client_connection`), the port is whatever the server assigns, but never 0. The IP is the IP of the originating host (`127.0.0.1` or `::1` for the local host)

26.12.4 Performance Schema Wait Event Tables

The Performance Schema instruments waits, which are events that take time. Within the event hierarchy, wait events nest within stage events, which nest within statement events, which nest within transaction events.

These tables store wait events:

- `events_waits_current`: The current wait event for each thread.
- `events_waits_history`: The most recent wait events that have ended per thread.
- `events_waits_history_long`: The most recent wait events that have ended globally (across all threads).

The following sections describe the wait event tables. There are also summary tables that aggregate information about wait events; see Section 26.12.17.1, “Wait Event Summary Tables”.

For more information about the relationship between the three wait event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

Configuring Wait Event Collection

To control whether to collect wait events, set the state of the relevant instruments and consumers:

- The `setup_instruments` table contains instruments with names that begin with `wait`. Use these instruments to enable or disable collection of individual wait event classes.
- The `setup_consumers` table contains consumer values with names corresponding to the current and historical wait event table names. Use these consumers to filter collection of wait events.

Some wait instruments are enabled by default; others are disabled. For example:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'wait/io/file/innodb%';
+----------------------------------------------------+---------+-------+
| NAME                                               | ENABLED | TIMED |
+----------------------------------------------------+---------+-------+
| wait/io/file/innodb/innodb_tablespace_open_file | YES     | YES   |
| wait/io/file/innodb/innodb_data_file            | YES     | YES   |
| wait/io/file/innodb/innodb_log_file             | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file            | YES     | YES   |
| wait/io/file/innodb/innodb_arch_file            | YES     | YES   |
| wait/io/file/innodb/innodb_clone_file           | YES     | YES   |
+----------------------------------------------------+---------+-------+
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'wait/io/socket/%';
+---------------------------------------+---------+-------+
| NAME                                  | ENABLED | TIMED |
+---------------------------------------+---------+-------+
| wait/io/socket/sql/server_tcpip_socket | NO      | NO    |
| wait/io/socket/sql/server_unix_socket  | NO      | NO    |
| wait/io/socket/sql/client_connection   | NO      | NO    |
+---------------------------------------+---------+-------+
```

The wait consumers are disabled by default:

```
mysql> SELECT *
       FROM performance_schema.setup_consumers
       WHERE NAME LIKE 'events_waits%';
+---------------------------+---------+
| NAME                      | ENABLED |
+---------------------------+---------+
| events_waits_current      | NO      |
| events_waits_history      | NO      |
| events_waits_history_long | NO      |
+---------------------------+---------+
```

To control wait event collection at server startup, use lines like these in your `my.cnf` file:

- Enable:

```
[mysqld]
performance-schema-instrument='wait/%=ON'
performance-schema-consumer-events-waits-current=ON
performance-schema-consumer-events-waits-history=ON
performance-schema-consumer-events-waits-history-long=ON
```

- Disable:

```
[mysqld]
performance-schema-instrument='wait/%=OFF'
performance-schema-consumer-events-waits-current=OFF
performance-schema-consumer-events-waits-history=OFF
performance-schema-consumer-events-waits-history-long=OFF
```

To control wait event collection at runtime, update the `setup_instruments` and `setup_consumers` tables:

- Enable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME = 'wait/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES'
WHERE NAME LIKE 'events_waits%';
```

- Disable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME = 'wait/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE 'events_waits%';
```

To collect only specific wait events, enable only the corresponding wait instruments. To collect wait events only for specific wait event tables, enable the wait instruments but only the wait consumers corresponding to the desired tables.

For additional information about configuring event collection, see Section 26.3, “Performance Schema Startup Configuration”, and Section 26.4, “Performance Schema Runtime Configuration”.

26.12.4.1 The events_waits_current Table

The events_waits_current table contains current wait events. The table stores one row per thread showing the current status of the thread's most recent monitored wait event, so there is no system variable for configuring the table size.

Of the tables that contain wait event rows, events_waits_current is the most fundamental. Other tables that contain wait event rows are logically derived from the current events. For example, the events_waits_history and events_waits_history_long tables are collections of the most recent wait events that have ended, up to a maximum number of rows per thread and globally across all threads, respectively.

For more information about the relationship between the three wait event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect wait events, see Section 26.12.4, “Performance Schema Wait Event Tables”.

The events_waits_current table has these columns:

- THREAD_ID, EVENT_ID

  The thread associated with the event and the thread current event number when the event starts. The THREAD_ID and EVENT_ID values taken together uniquely identify the row. No two rows have the same pair of values.

- END_EVENT_ID

  This column is set to NULL when the event starts and updated to the thread current event number when the event ends.

- EVENT_NAME

  The name of the instrument that produced the event. This is a NAME value from the setup_instruments table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 26.6, “Performance Schema Instrument Naming Conventions”.

- SOURCE

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved. For example, if a mutex or lock is being blocked, you can check the context in which this occurs.

- TIMER_START, TIMER_END, TIMER_WAIT

  Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The TIMER_START and TIMER_END values indicate when event timing started and ended. TIMER_WAIT is the event elapsed time (duration).

  If an event has not finished, TIMER_END is the current timer value and TIMER_WAIT is the time elapsed so far (TIMER_END − TIMER_START).

  If an event is produced from an instrument that has TIMED = NO, timing information is not collected, and TIMER_START, TIMER_END, and TIMER_WAIT are all NULL.

  For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 26.4.1, “Performance Schema Event Timing”.

- SPINS

For a mutex, the number of spin rounds. If the value is `NULL`, the code does not use spin rounds or spinning is not instrumented.

- `OBJECT_SCHEMA`, `OBJECT_NAME`, `OBJECT_TYPE`, `OBJECT_INSTANCE_BEGIN`

  These columns identify the object "being acted on." What that means depends on the object type.

  For a synchronization object (`cond`, `mutex`, `rwlock`):

  - `OBJECT_SCHEMA`, `OBJECT_NAME`, and `OBJECT_TYPE` are `NULL`.
  - `OBJECT_INSTANCE_BEGIN` is the address of the synchronization object in memory.

  For a file I/O object:

  - `OBJECT_SCHEMA` is `NULL`.
  - `OBJECT_NAME` is the file name.
  - `OBJECT_TYPE` is `FILE`.
  - `OBJECT_INSTANCE_BEGIN` is an address in memory.

  For a socket object:

  - `OBJECT_NAME` is the `IP:PORT` value for the socket.
  - `OBJECT_INSTANCE_BEGIN` is an address in memory.

  For a table I/O object:

  - `OBJECT_SCHEMA` is the name of the schema that contains the table.
  - `OBJECT_NAME` is the table name.
  - `OBJECT_TYPE` is `TABLE` for a persistent base table or `TEMPORARY TABLE` for a temporary table.
  - `OBJECT_INSTANCE_BEGIN` is an address in memory.

  An `OBJECT_INSTANCE_BEGIN` value itself has no meaning, except that different values indicate different objects. `OBJECT_INSTANCE_BEGIN` can be used for debugging. For example, it can be used with `GROUP BY OBJECT_INSTANCE_BEGIN` to see whether the load on 1,000 mutexes (that protect, say, 1,000 pages or blocks of data) is spread evenly or just hitting a few bottlenecks. This can help you correlate with other sources of information if you see the same object address in a log file or another debugging or performance tool.

- `INDEX_NAME`

  The name of the index used. `PRIMARY` indicates the table primary index. `NULL` means that no index was used.

- `NESTING_EVENT_ID`

  The `EVENT_ID` value of the event within which this event is nested.

- `NESTING_EVENT_TYPE`

  The nesting event type. The value is `TRANSACTION`, `STATEMENT`, `STAGE`, or `WAIT`.

- `OPERATION`

  The type of operation performed, such as `lock`, `read`, or `write`.

- `NUMBER_OF_BYTES`

  The number of bytes read or written by the operation. For table I/O waits (events for the `wait/io/table/sql/handler` instrument), `NUMBER_OF_BYTES` indicates the number of rows. If the value is greater than 1, the event is for a batch I/O operation. The following discussion describes the difference between exclusively single-row reporting and reporting that reflects batch I/O.

  MySQL executes joins using a nested-loop implementation. The job of the Performance Schema instrumentation is to provide row count and accumulated execution time per table in the join. Assume a join query of the following form that is executed using a table join order of `t1`, `t2`, `t3`:

  ```
  SELECT ... FROM t1 JOIN t2 ON ... JOIN t3 ON ...
  ```

  Table “fanout” is the increase or decrease in number of rows from adding a table during join processing. If the fanout for table `t3` is greater than 1, the majority of row-fetch operations are for that table. Suppose that the join accesses 10 rows from `t1`, 20 rows from `t2` per row from `t1`, and 30 rows from `t3` per row of table `t2`. With single-row reporting, the total number of instrumented operations is:

  ```
  10 + (10 * 20) + (10 * 20 * 30) = 6210
  ```

  A significant reduction in the number of instrumented operations is achievable by aggregating them per scan (that is, per unique combination of rows from `t1` and `t2`). With batch I/O reporting, the Performance Schema produces an event for each scan of the innermost table `t3` rather than for each row, and the number of instrumented row operations reduces to:

  ```
  10 + (10 * 20) + (10 * 20) = 410
  ```

  That is a reduction of 93%, illustrating how the batch-reporting strategy significantly reduces Performance Schema overhead for table I/O by reducing the number of reporting calls. The tradeoff is lesser accuracy for event timing. Rather than time for an individual row operation as in per-row reporting, timing for batch I/O includes time spent for operations such as join buffering, aggregation, and returning rows to the client.

  For batch I/O reporting to occur, these conditions must be true:

  - Query execution accesses the innermost table of a query block (for a single-table query, that table counts as innermost)
  - Query execution does not request a single row from the table (so, for example, `eq_ref` access prevents use of batch reporting)
  - Query execution does not evaluate a subquery containing table access for the table

- `FLAGS`

  Reserved for future use.

The `events_waits_current` table has these indexes:

- Primary key on (`THREAD_ID`, `EVENT_ID`)

`TRUNCATE TABLE` is permitted for the `events_waits_current` table. It removes the rows.

26.12.4.2 The events_waits_history Table

The `events_waits_history` table contains the *`N`* most recent wait events that have ended per thread. Wait events are not added to the table until they have ended. When the table contains the maximum number of rows for a given thread, the oldest thread row is discarded when a new row for that thread is added. When a thread ends, all its rows are discarded.

The Performance Schema autosizes the value of *`N`* during server startup. To set the number of rows per thread explicitly, set the `performance_schema_events_waits_history_size` system variable at server startup.

The `events_waits_history` table has the same columns and indexing as `events_waits_current`. See Section 26.12.4.1, “The events_waits_current Table”.

`TRUNCATE TABLE` is permitted for the `events_waits_history` table. It removes the rows.

For more information about the relationship between the three wait event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect wait events, see Section 26.12.4, “Performance Schema Wait Event Tables”.

26.12.4.3 The events_waits_history_long Table

The `events_waits_history_long` table contains *`N`* the most recent wait events that have ended globally, across all threads. Wait events are not added to the table until they have ended. When the table becomes full, the oldest row is discarded when a new row is added, regardless of which thread generated either row.

The Performance Schema autosizes the value of *`N`* during server startup. To set the table size explicitly, set the `performance_schema_events_waits_history_long_size` system variable at server startup.

The `events_waits_history_long` table has the same columns as `events_waits_current`. See Section 26.12.4.1, “The events_waits_current Table”. Unlike `events_waits_current`, `events_waits_history_long` has no indexing.

`TRUNCATE TABLE` is permitted for the `events_waits_history_long` table. It removes the rows.

For more information about the relationship between the three wait event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect wait events, see Section 26.12.4, “Performance Schema Wait Event Tables”.

26.12.5 Performance Schema Stage Event Tables

The Performance Schema instruments stages, which are steps during the statement-execution process, such as parsing a statement, opening a table, or performing a `filesort` operation. Stages correspond to the thread states displayed by `SHOW PROCESSLIST` or that are visible in the `INFORMATION_SCHEMA.PROCESSLIST` table. Stages begin and end when state values change.

Within the event hierarchy, wait events nest within stage events, which nest within statement events, which nest within transaction events.

These tables store stage events:

- `events_stages_current`: The current stage event for each thread.

- `events_stages_history`: The most recent stage events that have ended per thread.
- `events_stages_history_long`: The most recent stage events that have ended globally (across all threads).

The following sections describe the stage event tables. There are also summary tables that aggregate information about stage events; see Section 26.12.17.2, “Stage Summary Tables”.

For more information about the relationship between the three stage event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

Configuring Stage Event Collection

To control whether to collect stage events, set the state of the relevant instruments and consumers:

- The `setup_instruments` table contains instruments with names that begin with `stage`. Use these instruments to enable or disable collection of individual stage event classes.
- The `setup_consumers` table contains consumer values with names corresponding to the current and historical stage event table names. Use these consumers to filter collection of stage events.

Other than those instruments that provide statement progress information, the stage instruments are disabled by default. For example:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments
       WHERE NAME RLIKE 'stage/sql/[a-c]';
+----------------------------------------------------+---------+-------+
| NAME                                               | ENABLED | TIMED |
+----------------------------------------------------+---------+-------+
| stage/sql/After create                             | NO      | NO    |
| stage/sql/allocating local table                   | NO      | NO    |
| stage/sql/altering table                           | NO      | NO    |
| stage/sql/committing alter table to storage engine | NO      | NO    |
| stage/sql/Changing master                          | NO      | NO    |
| stage/sql/Checking master version                  | NO      | NO    |
| stage/sql/checking permissions                     | NO      | NO    |
| stage/sql/cleaning up                              | NO      | NO    |
| stage/sql/closing tables                           | NO      | NO    |
| stage/sql/Connecting to master                     | NO      | NO    |
| stage/sql/converting HEAP to MyISAM                | NO      | NO    |
| stage/sql/Copying to group table                   | NO      | NO    |
| stage/sql/Copying to tmp table                     | NO      | NO    |
| stage/sql/copy to tmp table                        | NO      | NO    |
| stage/sql/Creating sort index                      | NO      | NO    |
| stage/sql/creating table                           | NO      | NO    |
| stage/sql/Creating tmp table                       | NO      | NO    |
+----------------------------------------------------+---------+-------+
```

Stage event instruments that provide statement progress information are enabled and timed by default:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments
       WHERE ENABLED='YES' AND NAME LIKE "stage/%";
+----------------------------------------------------+---------+-------+
| NAME                                               | ENABLED | TIMED |
+----------------------------------------------------+---------+-------+
| stage/sql/copy to tmp table                        | YES     | YES   |
| stage/sql/Applying batch of row changes (write)    | YES     | YES   |
| stage/sql/Applying batch of row changes (update)   | YES     | YES   |
| stage/sql/Applying batch of row changes (delete)   | YES     | YES   |
```

```
| stage/innodb/alter table (end)                        | YES     | YES   |
| stage/innodb/alter table (flush)                      | YES     | YES   |
| stage/innodb/alter table (insert)                     | YES     | YES   |
| stage/innodb/alter table (log apply index)            | YES     | YES   |
| stage/innodb/alter table (log apply table)            | YES     | YES   |
| stage/innodb/alter table (merge sort)                 | YES     | YES   |
| stage/innodb/alter table (read PK and internal sort) | YES     | YES   |
| stage/innodb/buffer pool load                         | YES     | YES   |
| stage/innodb/clone (file copy)                        | YES     | YES   |
| stage/innodb/clone (redo copy)                        | YES     | YES   |
| stage/innodb/clone (page copy)                        | YES     | YES   |
+-------------------------------------------------------+---------+-------+
```

The stage consumers are disabled by default:

```
mysql> SELECT *
       FROM performance_schema.setup_consumers
       WHERE NAME LIKE 'events_stages%';
+----------------------------+---------+
| NAME                       | ENABLED |
+----------------------------+---------+
| events_stages_current      | NO      |
| events_stages_history      | NO      |
| events_stages_history_long | NO      |
+----------------------------+---------+
```

To control stage event collection at server startup, use lines like these in your `my.cnf` file:

- Enable:

  ```
  [mysqld]
  performance-schema-instrument='stage/%=ON'
  performance-schema-consumer-events-stages-current=ON
  performance-schema-consumer-events-stages-history=ON
  performance-schema-consumer-events-stages-history-long=ON
  ```

- Disable:

  ```
  [mysqld]
  performance-schema-instrument='stage/%=OFF'
  performance-schema-consumer-events-stages-current=OFF
  performance-schema-consumer-events-stages-history=OFF
  performance-schema-consumer-events-stages-history-long=OFF
  ```

To control stage event collection at runtime, update the `setup_instruments` and `setup_consumers` tables:

- Enable:

  ```
  UPDATE performance_schema.setup_instruments
  SET ENABLED = 'YES', TIMED = 'YES'
  WHERE NAME = 'stage/%';

  UPDATE performance_schema.setup_consumers
  SET ENABLED = 'YES'
  WHERE NAME LIKE 'events_stages%';
  ```

- Disable:

  ```
  UPDATE performance_schema.setup_instruments
  ```

```
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME = 'stage/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE 'events_stages%';
```

To collect only specific stage events, enable only the corresponding stage instruments. To collect stage events only for specific stage event tables, enable the stage instruments but only the stage consumers corresponding to the desired tables.

For additional information about configuring event collection, see Section 26.3, “Performance Schema Startup Configuration”, and Section 26.4, “Performance Schema Runtime Configuration”.

Stage Event Progress Information

The Performance Schema stage event tables contain two columns that, taken together, provide a stage progress indicator for each row:

- `WORK_COMPLETED`: The number of work units completed for the stage
- `WORK_ESTIMATED`: The number of work units expected for the stage

Each column is `NULL` if no progress information is provided for an instrument. Interpretation of the information, if it is available, depends entirely on the instrument implementation. The Performance Schema tables provide a container to store progress data, but make no assumptions about the semantics of the metric itself:

- A “work unit” is an integer metric that increases over time during execution, such as the number of bytes, rows, files, or tables processed. The definition of “work unit” for a particular instrument is left to the instrumentation code providing the data.
- The `WORK_COMPLETED` value can increase one or many units at a time, depending on the instrumented code.
- The `WORK_ESTIMATED` value can change during the stage, depending on the instrumented code.

Instrumentation for a stage event progress indicator can implement any of the following behaviors:

- No progress instrumentation

  This is the most typical case, where no progress data is provided. The `WORK_COMPLETED` and `WORK_ESTIMATED` columns are both `NULL`.

- Unbounded progress instrumentation

  Only the `WORK_COMPLETED` column is meaningful. No data is provided for the `WORK_ESTIMATED` column, which displays 0.

  By querying the `events_stages_current` table for the monitored session, a monitoring application can report how much work has been performed so far, but cannot report whether the stage is near completion. Currently, no stages are instrumented like this.

- Bounded progress instrumentation

  The `WORK_COMPLETED` and `WORK_ESTIMATED` columns are both meaningful.

  This type of progress indicator is appropriate for an operation with a defined completion criterion, such as the table-copy instrument described later. By querying the `events_stages_current` table for

the monitored session, a monitoring application can report how much work has been performed so far, and can report the overall completion percentage for the stage, by computing the `WORK_COMPLETED` / `WORK_ESTIMATED` ratio.

The `stage/sql/copy to tmp table` instrument illustrates how progress indicators work. During execution of an `ALTER TABLE` statement, the `stage/sql/copy to tmp table` stage is used, and this stage can execute potentially for a long time, depending on the size of the data to copy.

The table-copy task has a defined termination (all rows copied), and the `stage/sql/copy to tmp table` stage is instrumented to provided bounded progress information: The work unit used is number of rows copied, `WORK_COMPLETED` and `WORK_ESTIMATED` are both meaningful, and their ratio indicates task percentage complete.

To enable the instrument and the relevant consumers, execute these statements:

```
UPDATE performance_schema.setup_instruments
SET ENABLED='YES'
WHERE NAME='stage/sql/copy to tmp table';

UPDATE performance_schema.setup_consumers
SET ENABLED='YES'
WHERE NAME LIKE 'events_stages_%';
```

To see the progress of an ongoing `ALTER TABLE` statement, select from the `events_stages_current` table.

26.12.5.1 The events_stages_current Table

The `events_stages_current` table contains current stage events. The table stores one row per thread showing the current status of the thread's most recent monitored stage event, so there is no system variable for configuring the table size.

Of the tables that contain stage event rows, `events_stages_current` is the most fundamental. Other tables that contain stage event rows are logically derived from the current events. For example, the `events_stages_history` and `events_stages_history_long` tables are collections of the most recent stage events that have ended, up to a maximum number of rows per thread and globally across all threads, respectively.

For more information about the relationship between the three stage event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect stage events, see Section 26.12.5, “Performance Schema Stage Event Tables”.

The `events_stages_current` table has these columns:

- `THREAD_ID`, `EVENT_ID`

  The thread associated with the event and the thread current event number when the event starts. The `THREAD_ID` and `EVENT_ID` values taken together uniquely identify the row. No two rows have the same pair of values.

- `END_EVENT_ID`

  This column is set to `NULL` when the event starts and updated to the thread current event number when the event ends.

- `EVENT_NAME`

The name of the instrument that produced the event. This is a `NAME` value from the `setup_instruments` table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 26.6, “Performance Schema Instrument Naming Conventions”.

- `SOURCE`

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved.

- `TIMER_START`, `TIMER_END`, `TIMER_WAIT`

  Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The `TIMER_START` and `TIMER_END` values indicate when event timing started and ended. `TIMER_WAIT` is the event elapsed time (duration).

  If an event has not finished, `TIMER_END` is the current timer value and `TIMER_WAIT` is the time elapsed so far (`TIMER_END` − `TIMER_START`).

  If an event is produced from an instrument that has `TIMED = NO`, timing information is not collected, and `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` are all `NULL`.

  For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 26.4.1, “Performance Schema Event Timing”.

- `WORK_COMPLETED`, `WORK_ESTIMATED`

  These columns provide stage progress information, for instruments that have been implemented to produce such information. `WORK_COMPLETED` indicates how many work units have been completed for the stage, and `WORK_ESTIMATED` indicates how many work units are expected for the stage. For more information, see Stage Event Progress Information.

- `NESTING_EVENT_ID`

  The `EVENT_ID` value of the event within which this event is nested. The nesting event for a stage event is usually a statement event.

- `NESTING_EVENT_TYPE`

  The nesting event type. The value is `TRANSACTION`, `STATEMENT`, `STAGE`, or `WAIT`.

The `events_stages_current` table has these indexes:

- Primary key on (`THREAD_ID`, `EVENT_ID`)

`TRUNCATE TABLE` is permitted for the `events_stages_current` table. It removes the rows.

26.12.5.2 The events_stages_history Table

The `events_stages_history` table contains the *`N`* most recent stage events that have ended per thread. Stage events are not added to the table until they have ended. When the table contains the maximum number of rows for a given thread, the oldest thread row is discarded when a new row for that thread is added. When a thread ends, all its rows are discarded.

The Performance Schema autosizes the value of *`N`* during server startup. To set the number of rows per thread explicitly, set the `performance_schema_events_stages_history_size` system variable at server startup.

The events_stages_history table has the same columns and indexing as events_stages_current. See Section 26.12.5.1, “The events_stages_current Table”.

TRUNCATE TABLE is permitted for the events_stages_history table. It removes the rows.

For more information about the relationship between the three stage event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect stage events, see Section 26.12.5, “Performance Schema Stage Event Tables”.

26.12.5.3 The events_stages_history_long Table

The events_stages_history_long table contains the *N* most recent stage events that have ended globally, across all threads. Stage events are not added to the table until they have ended. When the table becomes full, the oldest row is discarded when a new row is added, regardless of which thread generated either row.

The Performance Schema autosizes the value of *N* during server startup. To set the table size explicitly, set the performance_schema_events_stages_history_long_size system variable at server startup.

The events_stages_history_long table has the same columns as events_stages_current. See Section 26.12.5.1, “The events_stages_current Table”. Unlike events_stages_current, events_stages_history_long has no indexing.

TRUNCATE TABLE is permitted for the events_stages_history_long table. It removes the rows.

For more information about the relationship between the three stage event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect stage events, see Section 26.12.5, “Performance Schema Stage Event Tables”.

26.12.6 Performance Schema Statement Event Tables

The Performance Schema instruments statement execution. Statement events occur at a high level of the event hierarchy. Within the event hierarchy, wait events nest within stage events, which nest within statement events, which nest within transaction events.

These tables store statement events:

- events_statements_current: The current statement event for each thread.
- events_statements_history: The most recent statement events that have ended per thread.
- events_statements_history_long: The most recent statement events that have ended globally (across all threads).
- prepared_statements_instances: Prepared statement instances and statistics

The following sections describe the statement event tables. There are also summary tables that aggregate information about statement events; see Section 26.12.17.3, “Statement Summary Tables”.

For more information about the relationship between the three events_statements_*xxx* event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

Configuring Statement Event Collation

To control whether to collect statement events, set the state of the relevant instruments and consumers:

- The `setup_instruments` table contains instruments with names that begin with `statement`. Use these instruments to enable or disable collection of individual statement event classes.
- The `setup_consumers` table contains consumer values with names corresponding to the current and historical statement event table names, and the statement digest consumer. Use these consumers to filter collection of statement events and statement digesting.

The statement instruments are enabled by default, and the `events_statements_current`, `events_statements_history`, and `statements_digest` statement consumers are enabled by default:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'statement/%';
+----------------------------------------+---------+-------+
| NAME                                   | ENABLED | TIMED |
+----------------------------------------+---------+-------+
| statement/sql/select                   | YES     | YES   |
| statement/sql/create_table             | YES     | YES   |
| statement/sql/create_index             | YES     | YES   |
...
| statement/sp/stmt                      | YES     | YES   |
| statement/sp/set                       | YES     | YES   |
| statement/sp/set_trigger_field         | YES     | YES   |
| statement/scheduler/event              | YES     | YES   |
| statement/com/Sleep                    | YES     | YES   |
| statement/com/Quit                     | YES     | YES   |
| statement/com/Init DB                  | YES     | YES   |
...
| statement/abstract/Query               | YES     | YES   |
| statement/abstract/new_packet          | YES     | YES   |
| statement/abstract/relay_log           | YES     | YES   |
+----------------------------------------+---------+-------+
```

```
mysql> SELECT *
       FROM performance_schema.setup_consumers
       WHERE NAME LIKE '%statements%';
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_statements_current      | YES     |
| events_statements_history      | YES     |
| events_statements_history_long | NO      |
| statements_digest              | YES     |
+--------------------------------+---------+
```

To control statement event collection at server startup, use lines like these in your `my.cnf` file:

- Enable:

```
[mysqld]
performance-schema-instrument='statement/%=ON'
performance-schema-consumer-events-statements-current=ON
performance-schema-consumer-events-statements-history=ON
performance-schema-consumer-events-statements-history-long=ON
performance-schema-consumer-statements-digest=ON
```

- Disable:

```
[mysqld]
performance-schema-instrument='statement/%=OFF'
performance-schema-consumer-events-statements-current=OFF
performance-schema-consumer-events-statements-history=OFF
performance-schema-consumer-events-statements-history-long=OFF
performance-schema-consumer-statements-digest=OFF
```

To control statement event collection at runtime, update the `setup_instruments` and `setup_consumers` tables:

- Enable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME LIKE 'statement/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES'
WHERE NAME LIKE '%statements%';
```

- Disable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME LIKE 'statement/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE '%statements%';
```

To collect only specific statement events, enable only the corresponding statement instruments. To collect statement events only for specific statement event tables, enable the statement instruments but only the statement consumers corresponding to the desired tables.

For additional information about configuring event collection, see Section 26.3, “Performance Schema Startup Configuration”, and Section 26.4, “Performance Schema Runtime Configuration”.

Statement Monitoring

Statement monitoring begins from the moment the server sees that activity is requested on a thread, to the moment when all activity has ceased. Typically, this means from the time the server gets the first packet from the client to the time the server has finished sending the response. Statements within stored programs are monitored like other statements.

When the Performance Schema instruments a request (server command or SQL statement), it uses instrument names that proceed in stages from more general (or “abstract”) to more specific until it arrives at a final instrument name.

Final instrument names correspond to server commands and SQL statements:

- Server commands correspond to the `COM_xxx` codes defined in the `mysql_com.h` header file and processed in `sql/sql_parse.cc`. Examples are `COM_PING` and `COM_QUIT`. Instruments for commands have names that begin with `statement/com`, such as `statement/com/Ping` and `statement/com/Quit`.
- SQL statements are expressed as text, such as `DELETE FROM t1` or `SELECT * FROM t2`. Instruments for SQL statements have names that begin with `statement/sql`, such as `statement/sql/delete` and `statement/sql/select`.

Some final instrument names are specific to error handling:

- `statement/com/Error` accounts for messages received by the server that are out of band. It can be used to detect commands sent by clients that the server does not understand. This may be helpful for purposes such as identifying clients that are misconfigured or using a version of MySQL more recent than that of the server, or clients that are attempting to attack the server.
- `statement/sql/error` accounts for SQL statements that fail to parse. It can be used to detect malformed queries sent by clients. A query that fails to parse differs from a query that parses but fails due to an error during execution. For example, `SELECT * FROM` is malformed, and the `statement/sql/error` instrument is used. By contrast, `SELECT *` parses but fails with a `No tables used` error. In this case, `statement/sql/select` is used and the statement event contains information to indicate the nature of the error.

A request can be obtained from any of these sources:

- As a command or statement request from a client, which sends the request as packets
- As a statement string read from the relay log on a replication slave
- As an event from the Event Scheduler

The details for a request are not initially known and the Performance Schema proceeds from abstract to specific instrument names in a sequence that depends on the source of the request.

For a request received from a client:

1. When the server detects a new packet at the socket level, a new statement is started with an abstract instrument name of `statement/abstract/new_packet`.
2. When the server reads the packet number, it knows more about the type of request received, and the Performance Schema refines the instrument name. For example, if the request is a `COM_PING` packet, the instrument name becomes `statement/com/Ping` and that is the final name. If the request is a `COM_QUERY` packet, it is known to correspond to an SQL statement but not the particular type of statement. In this case, the instrument changes from one abstract name to a more specific but still abstract name, `statement/abstract/Query`, and the request requires further classification.
3. If the request is a statement, the statement text is read and given to the parser. After parsing, the exact statement type is known. If the request is, for example, an `INSERT` statement, the Performance Schema refines the instrument name from `statement/abstract/Query` to `statement/sql/insert`, which is the final name.

For a request read as a statement from the relay log on a replication slave:

1. Statements in the relay log are stored as text and are read as such. There is no network protocol, so the `statement/abstract/new_packet` instrument is not used. Instead, the initial instrument is `statement/abstract/relay_log`.
2. When the statement is parsed, the exact statement type is known. If the request is, for example, an `INSERT` statement, the Performance Schema refines the instrument name from `statement/abstract/Query` to `statement/sql/insert`, which is the final name.

The preceding description applies only for statement-based replication. For row-based replication, table I/O done on the slave as it processes row changes can be instrumented, but row events in the relay log do not appear as discrete statements.

For a request received from the Event Scheduler:

The event execution is instrumented using the name `statement/scheduler/event`. This is the final name.

Statements executed within the event body are instrumented using `statement/sql/*` names, without use of any preceding abstract instrument. An event is a stored program, and stored programs are precompiled in memory before execution. Consequently, there is no parsing at runtime and the type of each statement is known by the time it executes.

Statements executed within the event body are child statements. For example, if an event executes an `INSERT` statement, execution of the event itself is the parent, instrumented using `statement/scheduler/event`, and the `INSERT` is the child, instrumented using `statement/sql/insert`. The parent/child relationship holds *between* separate instrumented operations. This differs from the sequence of refinement that occurs *within* a single instrumented operation, from abstract to final instrument names.

For statistics to be collected for statements, it is not sufficient to enable only the final `statement/sql/*` instruments used for individual statement types. The abtract `statement/abstract/*` instruments must be enabled as well. This should not normally be an issue because all statement instruments are enabled by default. However, an application that enables or disables statement instruments selectively must take into account that disabling abstract instruments also disables statistics collection for the individual statement instruments. For example, to collect statistics for `INSERT` statements, `statement/sql/insert` must be enabled, but also `statement/abstract/new_packet` and `statement/abstract/Query`. Similarly, for replicated statements to be instrumented, `statement/abstract/relay_log` must be enabled.

No statistics are aggregated for abstract instruments such as `statement/abstract/Query` because no statement is ever classified with an abstract instrument as the final statement name.

26.12.6.1 The events_statements_current Table

The `events_statements_current` table contains current statement events. The table stores one row per thread showing the current status of the thread's most recent monitored statement event, so there is no system variable for configuring the table size.

Of the tables that contain statement event rows, `events_statements_current` is the most fundamental. Other tables that contain statement event rows are logically derived from the current events. For example, the `events_statements_history` and `events_statements_history_long` tables are collections of the most recent statement events that have ended, up to a maximum number of rows per thread and globally across all threads, respectively.

For more information about the relationship between the three `events_statements_xxx` event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect statement events, see Section 26.12.6, “Performance Schema Statement Event Tables”.

The `events_statements_current` table has these columns:

- `THREAD_ID`, `EVENT_ID`

  The thread associated with the event and the thread current event number when the event starts. The `THREAD_ID` and `EVENT_ID` values taken together uniquely identify the row. No two rows have the same pair of values.

- `END_EVENT_ID`

  This column is set to `NULL` when the event starts and updated to the thread current event number when the event ends.

- `EVENT_NAME`

  The name of the instrument from which the event was collected. This is a `NAME` value from the `setup_instruments` table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 26.6, “Performance Schema Instrument Naming Conventions”.

  For SQL statements, the `EVENT_NAME` value initially is `statement/com/Query` until the statement is parsed, then changes to a more appropriate value, as described in Section 26.12.6, “Performance Schema Statement Event Tables”.

- `SOURCE`

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved.

- `TIMER_START`, `TIMER_END`, `TIMER_WAIT`

  Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The `TIMER_START` and `TIMER_END` values indicate when event timing started and ended. `TIMER_WAIT` is the event elapsed time (duration).

  If an event has not finished, `TIMER_END` is the current timer value and `TIMER_WAIT` is the time elapsed so far (`TIMER_END` − `TIMER_START`).

  If an event is produced from an instrument that has `TIMED = NO`, timing information is not collected, and `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` are all `NULL`.

  For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 26.4.1, “Performance Schema Event Timing”.

- `LOCK_TIME`

  The time spent waiting for table locks. This value is computed in microseconds but normalized to picoseconds for easier comparison with other Performance Schema timers.

- `SQL_TEXT`

  The text of the SQL statement. For a command not associated with an SQL statement, the value is `NULL`.

  The maximum space available for statement display is 1024 bytes by default. To change this value, set the `performance_schema_max_sql_text_length` system variable at server startup. (Changing this value affects columns in other Performance Schema tables as well. See Section 26.10, “Performance Schema Statement Digests and Sampling”.)

- `DIGEST`

  The statement digest SHA-256 value as a string of 64 hexadecimal characters, or `NULL` if the `statements_digest` consumer is `no`. For more information about statement digesting, see Section 26.10, “Performance Schema Statement Digests and Sampling”.

- `DIGEST_TEXT`

  The normalized statement digest text, or `NULL` if the `statements_digest` consumer is `no`. For more information about statement digesting, see Section 26.10, “Performance Schema Statement Digests and Sampling”.

The `performance_schema_max_digest_length` system variable determines the maximum number of bytes available per session for digest value storage. However, the display length of statement digests may be longer than the available buffer size due to encoding of statement components such as keywords and literal values in digest buffer. Consequently, values selected from the `DIGEST_TEXT` column of statement event tables may appear to exceed the `performance_schema_max_digest_length` value.

- `CURRENT_SCHEMA`

  The default database for the statement, `NULL` if there is none.

- `OBJECT_SCHEMA`, `OBJECT_NAME`, `OBJECT_TYPE`

  For nested statements (stored programs), these columns contain information about the parent statement. Otherwise they are `NULL`.

- `OBJECT_INSTANCE_BEGIN`

  This column identifies the statement. The value is the address of an object in memory.

- `MYSQL_ERRNO`

  The statement error number, from the statement diagnostics area.

- `RETURNED_SQLSTATE`

  The statement SQLSTATE value, from the statement diagnostics area.

- `MESSAGE_TEXT`

  The statement error message, from the statement diagnostics area.

- `ERRORS`

  Whether an error occurred for the statement. The value is 0 if the SQLSTATE value begins with `00` (completion) or `01` (warning). The value is 1 is the SQLSTATE value is anything else.

- `WARNINGS`

  The number of warnings, from the statement diagnostics area.

- `ROWS_AFFECTED`

  The number of rows affected by the statement. For a description of the meaning of “affected,” see Section 28.7.7.1, “mysql_affected_rows()”.

- `ROWS_SENT`

  The number of rows returned by the statement.

- `ROWS_EXAMINED`

  The number of rows examined by the server layer (not counting any processing internal to storage engines).

- `CREATED_TMP_DISK_TABLES`

  Like the `Created_tmp_disk_tables` status variable, but specific to the statement.

- `CREATED_TMP_TABLES`

  Like the `Created_tmp_tables` status variable, but specific to the statement.

- `SELECT_FULL_JOIN`

  Like the `Select_full_join` status variable, but specific to the statement.

- `SELECT_FULL_RANGE_JOIN`

  Like the `Select_full_range_join` status variable, but specific to the statement.

- `SELECT_RANGE`

  Like the `Select_range` status variable, but specific to the statement.

- `SELECT_RANGE_CHECK`

  Like the `Select_range_check` status variable, but specific to the statement.

- `SELECT_SCAN`

  Like the `Select_scan` status variable, but specific to the statement.

- `SORT_MERGE_PASSES`

  Like the `Sort_merge_passes` status variable, but specific to the statement.

- `SORT_RANGE`

  Like the `Sort_range` status variable, but specific to the statement.

- `SORT_ROWS`

  Like the `Sort_rows` status variable, but specific to the statement.

- `SORT_SCAN`

  Like the `Sort_scan` status variable, but specific to the statement.

- `NO_INDEX_USED`

  1 if the statement performed a table scan without using an index, 0 otherwise.

- `NO_GOOD_INDEX_USED`

  1 if the server found no good index to use for the statement, 0 otherwise. For additional information, see the description of the `Extra` column from `EXPLAIN` output for the `Range checked for each record` value in Section 8.8.2, “EXPLAIN Output Format”.

- `NESTING_EVENT_ID`, `NESTING_EVENT_TYPE`, `NESTING_EVENT_LEVEL`

  These three columns are used with other columns to provide information as follows for top-level (unnested) statements and nested statements (executed within a stored program).

  For top level statements:

```
OBJECT_TYPE = NULL
OBJECT_SCHEMA = NULL
```

```
OBJECT_NAME = NULL
NESTING_EVENT_ID = NULL
NESTING_EVENT_TYPE = NULL
NESTING_LEVEL = 0
```

For nested statements:

```
OBJECT_TYPE = the parent statement object type
OBJECT_SCHEMA = the parent statement object schema
OBJECT_NAME = the parent statement object name
NESTING_EVENT_ID = the parent statement EVENT_ID
NESTING_EVENT_TYPE = 'STATEMENT'
NESTING_LEVEL = the parent statement NESTING_LEVEL plus one
```

- `STATEMENT_ID`

  The query ID maintained by the server at the SQL level. The value is unique for the server instance because these IDs are generated using a global counter that is incremented atomically. This column was added in MySQL 8.0.14.

The `events_statements_current` table has these indexes:

- Primary key on (`THREAD_ID`, `EVENT_ID`)

`TRUNCATE TABLE` is permitted for the `events_statements_current` table. It removes the rows.

26.12.6.2 The events_statements_history Table

The `events_statements_history` table contains the *`N`* most recent statement events that have ended per thread. Statement events are not added to the table until they have ended. When the table contains the maximum number of rows for a given thread, the oldest thread row is discarded when a new row for that thread is added. When a thread ends, all its rows are discarded.

The Performance Schema autosizes the value of *`N`* during server startup. To set the number of rows per thread explicitly, set the `performance_schema_events_statements_history_size` system variable at server startup.

The `events_statements_history` table has the same columns and indexing as `events_statements_current`. See Section 26.12.6.1, “The events_statements_current Table”.

`TRUNCATE TABLE` is permitted for the `events_statements_history` table. It removes the rows.

For more information about the relationship between the three `events_statements_`*`xxx`* event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect statement events, see Section 26.12.6, “Performance Schema Statement Event Tables”.

26.12.6.3 The events_statements_history_long Table

The `events_statements_history_long` table contains the *`N`* most recent statement events that have ended globally, across all threads. Statement events are not added to the table until they have ended. When the table becomes full, the oldest row is discarded when a new row is added, regardless of which thread generated either row.

The value of *`N`* is autosized at server startup. To set the table size explicitly, set the `performance_schema_events_statements_history_long_size` system variable at server startup.

The events_statements_history_long table has the same columns as events_statements_current. See Section 26.12.6.1, “The events_statements_current Table”. Unlike events_statements_current, events_statements_history_long has no indexing.

TRUNCATE TABLE is permitted for the events_statements_history_long table. It removes the rows.

For more information about the relationship between the three events_statements_*xxx* event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect statement events, see Section 26.12.6, “Performance Schema Statement Event Tables”.

26.12.6.4 The prepared_statements_instances Table

The Performance Schema provides instrumentation for prepared statements, for which there are two protocols:

- The binary protocol. This is accessed through the MySQL C API and maps onto underlying server commands as shown in the following table.

C API Function	Corresponding Server Command
mysql_stmt_prepare()	COM_STMT_PREPARE
mysql_stmt_execute()	COM_STMT_EXECUTE
mysql_stmt_close()	COM_STMT_CLOSE

- The text protocol. This is accessed using SQL statements and maps onto underlying server commands as shown in the following table.

SQL Statement	Corresponding Server Command
PREPARE	SQLCOM_PREPARE
EXECUTE	SQLCOM_EXECUTE
DEALLOCATE PREPARE, DROP PREPARE	SQLCOM_DEALLOCATE PREPARE

Performance Schema prepared statement instrumentation covers both protocols. The following discussion refers to the server commands rather than the C API functions or SQL statements.

Information about prepared statements is available in the prepared_statements_instances table. This table enables inspection of prepared statements used in the server and provides aggregated statistics about them. To control the size of this table, set the performance_schema_max_prepared_statements_instances system variable at server startup.

Collection of prepared statement information depends on the statement instruments shown in the following table. These instruments are enabled by default. To modify them, update the setup_instruments table.

Instrument	Server Command
statement/com/Prepare	COM_STMT_PREPARE
statement/com/Execute	COM_STMT_EXECUTE
statement/sql/prepare_sql	SQLCOM_PREPARE
statement/sql/execute_sql	SQLCOM_EXECUTE

The Performance Schema manages the contents of the prepared_statements_instances table as follows:

- Statement preparation

  A `COM_STMT_PREPARE` or `SQLCOM_PREPARE` command creates a prepared statement in the server. If the statement is successfully instrumented, a new row is added to the `prepared_statements_instances` table. If the statement cannot be instrumented, `Performance_schema_prepared_statements_lost` status variable is incremented.

- Prepared statement execution

  Execution of a `COM_STMT_EXECUTE` or `SQLCOM_PREPARE` command for an instrumented prepared statement instance updates the corresponding `prepared_statements_instances` table row.

- Prepared statement deallocation

  Execution of a `COM_STMT_CLOSE` or `SQLCOM_DEALLOCATE_PREPARE` command for an instrumented prepared statement instance removes the corresponding `prepared_statements_instances` table row. To avoid resource leaks, removal occurs even if the prepared statement instruments described previously are disabled.

The `prepared_statements_instances` table has these columns:

- `OBJECT_INSTANCE_BEGIN`

  The address in memory of the instrumented prepared statement.

- `STATEMENT_ID`

  The internal statement ID assigned by the server. The text and binary protocols both use statement IDs.

- `STATEMENT_NAME`

  For the binary protocol, this column is `NULL`. For the text protocol, this column is the external statement name assigned by the user. For example, for the following SQL statement, the name of the prepared statement is `stmt`:

  ```
  PREPARE stmt FROM 'SELECT 1';
  ```

- `SQL_TEXT`

  The prepared statement text, with `?` placeholder markers.

- `OWNER_THREAD_ID`, `OWNER_EVENT_ID`

  These columns indicate the event that created the prepared statement.

- `OWNER_OBJECT_TYPE`, `OWNER_OBJECT_SCHEMA`, `OWNER_OBJECT_NAME`

  For a prepared statement created by a client session, these columns are `NULL`. For a prepared statement created by a stored program, these columns point to the stored program. A typical user error is forgetting to deallocate prepared statements. These columns can be used to find stored programs that leak prepared statements:

  ```
  SELECT
    OWNER_OBJECT_TYPE, OWNER_OBJECT_SCHEMA, OWNER_OBJECT_NAME,
    STATEMENT_NAME, SQL_TEXT
  FROM performance_schema.prepared_statements_instances
  WHERE OWNER_OBJECT_TYPE IS NOT NULL;
  ```

- `TIMER_PREPARE`

  The time spent executing the statement preparation itself.

- `COUNT_REPREPARE`

  The number of times the statement was reprepared internally (see Section 8.10.3, “Caching of Prepared Statements and Stored Programs”). Timing statistics for repreparation are not available because it is counted as part of statement execution, not as a separate operation.

- `COUNT_EXECUTE`, `SUM_TIMER_EXECUTE`, `MIN_TIMER_EXECUTE`, `AVG_TIMER_EXECUTE`, `MAX_TIMER_EXECUTE`

  Aggregated statistics for executions of the prepared statement.

- `SUM_xxx`

  The remaining `SUM_xxx` columns are the same as for the statement summary tables (see Section 26.12.17.3, “Statement Summary Tables”).

The `prepared_statements_instances` table has these indexes:

- Primary key on (`OBJECT_INSTANCE_BEGIN`)
- Index on (`STATEMENT_ID`)
- Index on (`STATEMENT_NAME`)
- Index on (`OWNER_THREAD_ID`, `OWNER_EVENT_ID`)
- Index on (`OWNER_OBJECT_TYPE`, `OWNER_OBJECT_SCHEMA`, `OWNER_OBJECT_NAME`)

`TRUNCATE TABLE` resets the statistics columns of the `prepared_statements_instances` table.

26.12.7 Performance Schema Transaction Tables

The Performance Schema instruments transactions. Within the event hierarchy, wait events nest within stage events, which nest within statement events, which nest within transaction events.

These tables store transaction events:

- `events_transactions_current`: The current transaction event for each thread.
- `events_transactions_history`: The most recent transaction events that have ended per thread.
- `events_transactions_history_long`: The most recent transaction events that have ended globally (across all threads).

The following sections describe the transaction event tables. There are also summary tables that aggregate information about transaction events; see Section 26.12.17.5, “Transaction Summary Tables”.

For more information about the relationship between the three transaction event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

Configuring Transaction Event Collation

To control whether to collect transaction events, set the state of the relevant instruments and consumers:

- The `setup_instruments` table contains an instrument named `transaction`. Use this instrument to enable or disable collection of individual transaction event classes.
- The `setup_consumers` table contains consumer values with names corresponding to the current and historical transaction event table names. Use these consumers to filter collection of transaction events.

The `transaction` instrument and the `events_transactions_current` and `events_transactions_history` transaction consumers are enabled by default:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments
       WHERE NAME = 'transaction';
+-------------+---------+-------+
| NAME        | ENABLED | TIMED |
+-------------+---------+-------+
| transaction | YES     | YES   |
+-------------+---------+-------+
mysql> SELECT *
       FROM performance_schema.setup_consumers
       WHERE NAME LIKE 'events_transactions%';
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_transactions_current    | YES     |
| events_transactions_history    | YES     |
| events_transactions_history_long | NO    |
+--------------------------------+---------+
```

To control transaction event collection at server startup, use lines like these in your `my.cnf` file:

- Enable:

```
[mysqld]
performance-schema-instrument='transaction=ON'
performance-schema-consumer-events-transactions-current=ON
performance-schema-consumer-events-transactions-history=ON
performance-schema-consumer-events-transactions-history-long=ON
```

- Disable:

```
[mysqld]
performance-schema-instrument='transaction=OFF'
performance-schema-consumer-events-transactions-current=OFF
performance-schema-consumer-events-transactions-history=OFF
performance-schema-consumer-events-transactions-history-long=OFF
```

To control transaction event collection at runtime, update the `setup_instruments` and `setup_consumers` tables:

- Enable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME = 'transaction';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES'
WHERE NAME LIKE 'events_transactions%';
```

- Disable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME = 'transaction';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE 'events_transactions%';
```

To collect transaction events only for specific transaction event tables, enable the `transaction` instrument but only the transaction consumers corresponding to the desired tables.

For additional information about configuring event collection, see Section 26.3, “Performance Schema Startup Configuration”, and Section 26.4, “Performance Schema Runtime Configuration”.

Transaction Boundaries

In MySQL Server, transactions start explicitly with these statements:

```
START TRANSACTION | BEGIN | XA START | XA BEGIN
```

Transactions also start implicitly. For example, when the `autocommit` system variable is enabled, the start of each statement starts a new transaction.

When `autocommit` is disabled, the first statement following a committed transaction marks the start of a new transaction. Subsequent statements are part of the transaction until it is committed.

Transactions explicitly end with these statements:

```
COMMIT | ROLLBACK | XA COMMIT | XA ROLLBACK
```

Transactions also end implicitly, by execution of DDL statements, locking statements, and server administration statements.

In the following discussion, references to `START TRANSACTION` also apply to `BEGIN`, `XA START`, and `XA BEGIN`. Similarly, references to `COMMIT` and `ROLLBACK` apply to `XA COMMIT` and `XA ROLLBACK`, respectively.

The Performance Schema defines transaction boundaries similarly to that of the server. The start and end of a transaction event closely match the corresponding state transitions in the server:

- For an explicitly started transaction, the transaction event starts during processing of the `START TRANSACTION` statement.
- For an implicitly started transaction, the transaction event starts on the first statement that uses a transactional engine after the previous transaction has ended.
- For any transaction, whether explicitly or implicitly ended, the transaction event ends when the server transitions out of the active transaction state during the processing of `COMMIT` or `ROLLBACK`.

There are subtle implications to this approach:

- Transaction events in the Performance Schema do not fully include the statement events associated with the corresponding `START TRANSACTION`, `COMMIT`, or `ROLLBACK` statements. There is a trivial amount of timing overlap between the transaction event and these statements.
- Statements that work with nontransactional engines have no effect on the transaction state of the connection. For implicit transactions, the transaction event begins with the first statement that uses a

transactional engine. This means that statements operating exclusively on nontransactional tables are ignored, even following `START TRANSACTION`.

To illustrate, consider the following scenario:

```
1. SET autocommit = OFF;
2. CREATE TABLE t1 (a INT) ENGINE = InnoDB;
3. START TRANSACTION;                       -- Transaction 1 START
4. INSERT INTO t1 VALUES (1), (2), (3);
5. CREATE TABLE t2 (a INT) ENGINE = MyISAM; -- Transaction 1 COMMIT
                                            -- (implicit; DDL forces commit)
6. INSERT INTO t2 VALUES (1), (2), (3);     -- Update nontransactional table
7. UPDATE t2 SET a = a + 1;                 -- ... and again
8. INSERT INTO t1 VALUES (4), (5), (6);     -- Write to transactional table
                                            -- Transaction 2 START (implicit)
9. COMMIT;                                  -- Transaction 2 COMMIT
```

From the perspective of the server, Transaction 1 ends when table `t2` is created. Transaction 2 does not start until a transactional table is accessed, despite the intervening updates to nontransactional tables.

From the perspective of the Performance Schema, Transaction 2 starts when the server transitions into an active transaction state. Statements 6 and 7 are not included within the boundaries of Transaction 2, which is consistent with how the server writes transactions to the binary log.

Transaction Instrumentation

Three attributes define transactions:

- Access mode (read only, read write)
- Isolation level (`SERIALIZABLE`, `REPEATABLE READ`, and so forth)
- Implicit (`autocommit` enabled) or explicit (`autocommit` disabled)

To reduce complexity of the transaction instrumentation and to ensure that the collected transaction data provides complete, meaningful results, all transactions are instrumented independently of access mode, isolation level, or autocommit mode.

To selectively examine transaction history, use the attribute columns in the transaction event tables: `ACCESS_MODE`, `ISOLATION_LEVEL`, and `AUTOCOMMIT`.

The cost of transaction instrumentation can be reduced various ways, such as enabling or disabling transaction instrumentation according to user, account, host, or thread (client connection).

Transactions and Nested Events

The parent of a transaction event is the event that initiated the transaction. For an explicitly started transaction, this includes the `START TRANSACTION` and `COMMIT AND CHAIN` statements. For an implicitly started transaction, it is the first statement that uses a transactional engine after the previous transaction ends.

In general, a transaction is the top-level parent to all events initiated during the transaction, including statements that explicitly end the transaction such as `COMMIT` and `ROLLBACK`. Exceptions are statements that implicitly end a transaction, such as DDL statements, in which case the current transaction must be committed before the new statement is executed.

Transactions and Stored Programs

Transactions and stored program events are related as follows:

- Stored Procedures

  Stored procedures operate independently of transactions. A stored procedure can be started within a transaction, and a transaction can be started or ended from within a stored procedure. If called from within a transaction, a stored procedure can execute statements that force a commit of the parent transaction and then start a new transaction.

  If a stored procedure is started within a transaction, that transaction is the parent of the stored procedure event.

  If a transaction is started by a stored procedure, the stored procedure is the parent of the transaction event.

- Stored Functions

  Stored functions are restricted from causing an explicit or implicit commit or rollback. Stored function events can reside within a parent transaction event.

- Triggers

  Triggers activate as part of a statement that accesses the table with which it is associated, so the parent of a trigger event is always the statement that activates it.

  Triggers cannot issue statements that cause an explicit or implicit commit or rollback of a transaction.

- Scheduled Events

  The execution of the statements in the body of a scheduled event takes place in a new connection. Nesting of a scheduled event within a parent transaction is not applicable.

Transactions and Savepoints

Savepoint statements are recorded as separate statement events. Transaction events include separate counters for `SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, and `RELEASE SAVEPOINT` statements issued during the transaction.

Transactions and Errors

Errors and warnings that occur within a transaction are recorded in statement events, but not in the corresponding transaction event. This includes transaction-specific errors and warnings, such as a rollback on a nontransactional table or GTID consistency errors.

26.12.7.1 The events_transactions_current Table

The `events_transactions_current` table contains current transaction events. The table stores one row per thread showing the current status of the thread's most recent monitored transaction event, so there is no system variable for configuring the table size. For example:

```
mysql> SELECT *
       FROM performance_schema.events_transactions_current LIMIT 1\G
*************************** 1. row ***************************
                    THREAD_ID: 26
                     EVENT_ID: 7
                END_EVENT_ID: NULL
                  EVENT_NAME: transaction
                       STATE: ACTIVE
                      TRX_ID: NULL
                        GTID: 3E11FA47-71CA-11E1-9E33-C80AA9429562:56
```

```
                           XID: NULL
                      XA_STATE: NULL
                        SOURCE: transaction.cc:150
                   TIMER_START: 420833537900000
                     TIMER_END: NULL
                    TIMER_WAIT: NULL
                   ACCESS_MODE: READ WRITE
               ISOLATION_LEVEL: REPEATABLE READ
                    AUTOCOMMIT: NO
            NUMBER_OF_SAVEPOINTS: 0
NUMBER_OF_ROLLBACK_TO_SAVEPOINT: 0
    NUMBER_OF_RELEASE_SAVEPOINT: 0
          OBJECT_INSTANCE_BEGIN: NULL
               NESTING_EVENT_ID: 6
             NESTING_EVENT_TYPE: STATEMENT
```

Of the tables that contain transaction event rows, `events_transactions_current` is the most fundamental. Other tables that contain transaction event rows are logically derived from the current events. For example, the `events_transactions_history` and `events_transactions_history_long` tables are collections of the most recent transaction events that have ended, up to a maximum number of rows per thread and globally across all threads, respectively.

For more information about the relationship between the three transaction event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect transaction events, see Section 26.12.7, “Performance Schema Transaction Tables”.

The `events_transactions_current` table has these columns:

- `THREAD_ID`, `EVENT_ID`

  The thread associated with the event and the thread current event number when the event starts. The `THREAD_ID` and `EVENT_ID` values taken together uniquely identify the row. No two rows have the same pair of values.

- `END_EVENT_ID`

  This column is set to `NULL` when the event starts and updated to the thread current event number when the event ends.

- `EVENT_NAME`

  The name of the instrument from which the event was collected. This is a `NAME` value from the `setup_instruments` table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 26.6, “Performance Schema Instrument Naming Conventions”.

- `STATE`

  The current transaction state. The value is `ACTIVE` (after `START TRANSACTION` or `BEGIN`), `COMMITTED` (after `COMMIT`), or `ROLLED BACK` (after `ROLLBACK`).

- `TRX_ID`

  Unused.

- `GTID`

  The GTID column contains the value of `gtid_next`, which can be one of `ANONYMOUS`, `AUTOMATIC`, or a GTID using the format `UUID:NUMBER`. For transactions that use `gtid_next=AUTOMATIC`, which is

all normal client transactions, the GTID column changes when the transaction commits and the actual GTID is assigned. If `gtid_mode` is either `ON` or `ON_PERMISSIVE`, the GTID column changes to the transaction's GTID. If `gtid_mode` is either `OFF` or `OFF_PERMISSIVE`, the GTID column changes to `ANONYMOUS`.

- `XID_FORMAT_ID`, `XID_GTRID`, and `XID_BQUAL`

  The components of the XA transaction identifier. They have the format described in Section 13.3.8.1, "XA Transaction SQL Syntax".

- `XA_STATE`

  The state of the XA transaction. The value is `ACTIVE` (after `XA START`), `IDLE` (after `XA END`), `PREPARED` (after `XA PREPARE`), `ROLLED BACK` (after `XA ROLLBACK`), or `COMMITTED` (after `XA COMMIT`).

  On a replication slave, the same XA transaction can appear in the `events_transactions_current` table with different states on different threads. This is because immediately after the XA transaction is prepared, it is detached from the slave applier thread, and can be committed or rolled back by any thread on the slave. The `events_transactions_current` table displays the current status of the most recent monitored transaction event on the thread, and does not update this status when the thread is idle. So the XA transaction can still be displayed in the `PREPARED` state for the original applier thread, after it has been processed by another thread. To positively identify XA transactions that are still in the `PREPARED` state and need to be recovered, use the `XA RECOVER` statement rather than the Performance Schema transaction tables.

- `SOURCE`

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved.

- `TIMER_START`, `TIMER_END`, `TIMER_WAIT`

  Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The `TIMER_START` and `TIMER_END` values indicate when event timing started and ended. `TIMER_WAIT` is the event elapsed time (duration).

  If an event has not finished, `TIMER_END` is the current timer value and `TIMER_WAIT` is the time elapsed so far (`TIMER_END` − `TIMER_START`).

  If an event is produced from an instrument that has `TIMED = NO`, timing information is not collected, and `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` are all `NULL`.

  For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 26.4.1, "Performance Schema Event Timing".

- `ACCESS_MODE`

  The transaction access mode. The value is `READ WRITE` or `READ ONLY`.

- `ISOLATION_LEVEL`

  The transaction isolation level. The value is `REPEATABLE READ`, `READ COMMITTED`, `READ UNCOMMITTED`, or `SERIALIZABLE`.

- `AUTOCOMMIT`

Whether autcommit mode was enabled when the transaction started.

- `NUMBER_OF_SAVEPOINTS`, `NUMBER_OF_ROLLBACK_TO_SAVEPOINT`, `NUMBER_OF_RELEASE_SAVEPOINT`

  The number of `SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, and `RELEASE SAVEPOINT` statements issued during the transaction.

- `OBJECT_INSTANCE_BEGIN`

  Unused.

- `NESTING_EVENT_ID`

  The `EVENT_ID` value of the event within which this event is nested.

- `NESTING_EVENT_TYPE`

  The nesting event type. The value is `TRANSACTION`, `STATEMENT`, `STAGE`, or `WAIT`. (`TRANSACTION` will not appear because transactions cannot be nested.)

The `events_transactions_current` table has these indexes:

- Primary key on (`THREAD_ID`, `EVENT_ID`)

`TRUNCATE TABLE` is permitted for the `events_transactions_current` table. It removes the rows.

26.12.7.2 The events_transactions_history Table

The `events_transactions_history` table contains the *`N`* most recent transaction events that have ended per thread. Transaction events are not added to the table until they have ended. When the table contains the maximum number of rows for a given thread, the oldest thread row is discarded when a new row for that thread is added. When a thread ends, all its rows are discarded.

The Performance Schema autosizes the value of *`N`* during server startup. To set the number of rows per thread explicitly, set the `performance_schema_events_transactions_history_size` system variable at server startup.

The `events_transactions_history` table has the same columns and indexing as `events_transactions_current`. See Section 26.12.7.1, “The events_transactions_current Table”.

`TRUNCATE TABLE` is permitted for the `events_transactions_history` table. It removes the rows.

For more information about the relationship between the three transaction event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect transaction events, see Section 26.12.7, “Performance Schema Transaction Tables”.

26.12.7.3 The events_transactions_history_long Table

The `events_transactions_history_long` table contains the *`N`* most recent transaction events that have ended globally, across all threads. Transaction events are not added to the table until they have ended. When the table becomes full, the oldest row is discarded when a new row is added, regardless of which thread generated either row.

The Performance Schema autosizes the value of *N* is autosized at server startup. To set the table size explicitly, set the `performance_schema_events_transactions_history_long_size` system variable at server startup.

The `events_transactions_history_long` table has the same columns as `events_transactions_current`. See Section 26.12.7.1, “The events_transactions_current Table”. Unlike `events_transactions_current`, `events_transactions_history_long` has no indexing.

`TRUNCATE TABLE` is permitted for the `events_transactions_history_long` table. It removes the rows.

For more information about the relationship between the three transaction event tables, see Section 26.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect transaction events, see Section 26.12.7, “Performance Schema Transaction Tables”.

26.12.8 Performance Schema Connection Tables

When a client connects to the MySQL server, it does so under a particular user name and from a particular host. The Performance Schema provides statistics about these connections, tracking them per account (user and host combination) as well as separately per user name and host name, using these tables:

- `accounts`: Connection statistics per client account
- `hosts`: Connection statistics per client host name
- `users`: Connection statistics per client user name

The meaning of “account” in the connection tables is similar to its meaning in the MySQL grant tables in the `mysql` system database, in the sense that the term refers to a combination of user and host values. They differ in that, for grant tables, the host part of an account can be a pattern, whereas for Performance Schema tables, the host value is always a specific nonpattern host name.

Each connection table has `CURRENT_CONNECTIONS` and `TOTAL_CONNECTIONS` columns to track the current and total number of connections per “tracking value” on which its statistics are based. The tables differ in what they use for the tracking value. The `accounts` table has `USER` and `HOST` columns to track connections per user and host combination. The `users` and `hosts` tables have a `USER` and `HOST` column, respectively, to track connections per user name and host name.

The Performance Schema also counts internal threads and threads for user sessions that failed to authenticate, using rows with `USER` and `HOST` column values of `NULL`.

Suppose that clients named `user1` and `user2` each connect one time from `hosta` and `hostb`. The Performance Schema tracks the connections as follows:

- The `accounts` table has four rows, for the `user1`/`hosta`, `user1`/`hostb`, `user2`/`hosta`, and `user2`/`hostb` account values, each row counting one connection per account.
- The `hosts` table has two rows, for `hosta` and `hostb`, each row counting two connections per host name.
- The `users` table has two rows, for `user1` and `user2`, each row counting two connections per user name.

When a client connects, the Performance Schema determines which row in each connection table applies, using the tracking value appropriate to each table. If there is no such row, one is added. Then

the Performance Schema increments by one the `CURRENT_CONNECTIONS` and `TOTAL_CONNECTIONS` columns in that row.

When a client disconnects, the Performance Schema decrements by one the `CURRENT_CONNECTIONS` column in the row and leaves the `TOTAL_CONNECTIONS` column unchanged.

`TRUNCATE TABLE` is permitted for connection tables. It has these effects:

- Rows are removed for accounts, hosts, or users that have no current connections (rows with `CURRENT_CONNECTIONS = 0`).
- Nonremoved rows are reset to count only current connections: For rows with `CURRENT_CONNECTIONS > 0`, `TOTAL_CONNECTIONS` is reset to `CURRENT_CONNECTIONS`.
- Summary tables that depend on the connection table are implicitly truncated, as described later in this section.

The Performance Schema maintains summary tables that aggregate connection statistics for various event types by account, host, or user. These tables have `_summary_by_account`, `_summary_by_host`, or `_summary_by_user` in the name. To identify them, use this query:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
       WHERE TABLE_SCHEMA = 'performance_schema'
       AND TABLE_NAME REGEXP '_summary_by_(account|host|user)'
       ORDER BY TABLE_NAME;
+-----------------------------------------------------+
| TABLE_NAME                                          |
+-----------------------------------------------------+
| events_errors_summary_by_account_by_error           |
| events_errors_summary_by_host_by_error              |
| events_errors_summary_by_user_by_error              |
| events_stages_summary_by_account_by_event_name      |
| events_stages_summary_by_host_by_event_name         |
| events_stages_summary_by_user_by_event_name         |
| events_statements_summary_by_account_by_event_name  |
| events_statements_summary_by_host_by_event_name     |
| events_statements_summary_by_user_by_event_name     |
| events_transactions_summary_by_account_by_event_name |
| events_transactions_summary_by_host_by_event_name   |
| events_transactions_summary_by_user_by_event_name   |
| events_waits_summary_by_account_by_event_name       |
| events_waits_summary_by_host_by_event_name          |
| events_waits_summary_by_user_by_event_name          |
| memory_summary_by_account_by_event_name             |
| memory_summary_by_host_by_event_name                |
| memory_summary_by_user_by_event_name                |
+-----------------------------------------------------+
```

For details about individual connection summary tables, consult the section that describes tables for the summarized event type:

- Wait event summaries: Section 26.12.17.1, “Wait Event Summary Tables”
- Stage event summaries: Section 26.12.17.2, “Stage Summary Tables”
- Statement event summaries: Section 26.12.17.3, “Statement Summary Tables”
- Transaction event summaries: Section 26.12.17.5, “Transaction Summary Tables”
- Memory event summaries: Section 26.12.17.10, “Memory Summary Tables”
- Error event summaries: Section 26.12.17.11, “Error Summary Tables”

`TRUNCATE TABLE` is permitted for connection summary tables. It removes rows for accounts, hosts, or users with no connections, and resets the summary columns to zero for the remaining rows. In addition, each summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends. The following table describes the relationship between connection table truncation and implicitly truncated tables.

Table 26.2 Implicit Effects of Connection Table Truncation

Truncated Connection Table	Implicitly Truncated Summary Tables
`accounts`	Tables with names containing `_summary_by_account`, `_summary_by_thread`
`hosts`	Tables with names containing `_summary_by_account`, `_summary_by_host`, `_summary_by_thread`
`users`	Tables with names containing `_summary_by_account`, `_summary_by_user`, `_summary_by_thread`

Truncating a `_summary_global` summary table also implicitly truncates its corresponding connection and thread summary tables. For example, truncating `events_waits_summary_global_by_event_name` implicitly truncates the wait event summary tables that are aggregated by account, host, user, or thread.

26.12.8.1 The accounts Table

The `accounts` table contains a row for each account that has connected to the MySQL server. For each account, the table counts the current and total number of connections. The table size is autosized at server startup. To set the table size explicitly, set the `performance_schema_accounts_size` system variable at server startup. To disable account statistics, set this variable to 0.

The `accounts` table has the following columns. For a description of how the Performance Schema maintains rows in this table, including the effect of `TRUNCATE TABLE`, see Section 26.12.8, “Performance Schema Connection Tables”.

- `USER`

  The client user name for the connection. This is `NULL` for an internal thread, or for a user session that failed to authenticate.

- `HOST`

  The host from which the client connected. This is `NULL` for an internal thread, or for a user session that failed to authenticate.

- `CURRENT_CONNECTIONS`

  The current number of connections for the account.

- `TOTAL_CONNECTIONS`

  The total number of connections for the account.

The `accounts` table has these indexes:

- Primary key on (`USER`, `HOST`)

26.12.8.2 The hosts Table

The `hosts` table contains a row for each host from which clients have connected to the MySQL server. For each host name, the table counts the current and total number of connections. The table size is autosized

at server startup. To set the table size explicitly, set the `performance_schema_hosts_size` system variable at server startup. To disable host statistics, set this variable to 0.

The `hosts` table has the following columns. For a description of how the Performance Schema maintains rows in this table, including the effect of `TRUNCATE TABLE`, see Section 26.12.8, “Performance Schema Connection Tables”.

- `HOST`

  The host from which the client connected. This is `NULL` for an internal thread, or for a user session that failed to authenticate.

- `CURRENT_CONNECTIONS`

  The current number of connections for the host.

- `TOTAL_CONNECTIONS`

  The total number of connections for the host.

The `hosts` table has these indexes:

- Primary key on (`HOST`)

26.12.8.3 The users Table

The `users` table contains a row for each user who has connected to the MySQL server. For each user name, the table counts the current and total number of connections. The table size is autosized at server startup. To set the table size explicitly, set the `performance_schema_users_size` system variable at server startup. To disable user statistics, set this variable to 0.

The `users` table has the following columns. For a description of how the Performance Schema maintains rows in this table, including the effect of `TRUNCATE TABLE`, see Section 26.12.8, “Performance Schema Connection Tables”.

- `USER`

  The client user name for the connection. This is `NULL` for an internal thread, or for a user session that failed to authenticate.

- `CURRENT_CONNECTIONS`

  The current number of connections for the user.

- `TOTAL_CONNECTIONS`

  The total number of connections for the user.

The `users` table has these indexes:

- Primary key on (`USER`)

26.12.9 Performance Schema Connection Attribute Tables

Connection attributes are key-value pairs that application programs can pass to the server at connect time. Applications based on the C API define the attribute set using the `mysql_options()` and `mysql_options4()` functions. Other MySQL Connectors may provide their own attribute-definition methods.

These tables expose attribute information:

- `session_account_connect_attrs`: Connection attributes for the current session, and other sessions associated with the session account
- `session_connect_attrs`: Connection attributes for all sessions

Attribute names that begin with an underscore (_) are reserved for internal use and should not be created by application programs. This convention permits new attributes to be introduced by MySQL without colliding with application attributes, and enables application programs to define their own attributes that do not collide with internal attributes.

The set of connection attributes visible on a given connection varies depending on factors such as your platform, MySQL Connector used to establish the connection, or client program.

The `libmysqlclient` client library (provided in MySQL and MySQL Connector/C distributions) sets these attributes:

- `_client_name`: The client name (`libmysql` for the client library).
- `_client_version`: The client library version.
- `_os`: The operating system (for example, `Linux`, `Win64`).
- `_pid`: The client process ID.
- `_platform`: The machine platform (for example, `x86_64`).
- `_thread`: The client thread ID (Windows only).

Other MySQL Connectors may define their own connection attributes.

MySQL Connector/C++ 8.0.16 and higher defines these attributes for applications that use X DevAPI or X DevAPI for C:

- `_client_license`: The connector license (for example `GPL-2.0`).
- `_client_name`: The connector name (`mysql-connector-cpp`).
- `_client_version`: The connector version.
- `_os`: The operating system (for example, `Linux`, `Win64`).
- `_pid`: The client process ID.
- `_platform`: The machine platform (for example, `x86_64`).
- `_source_host`: The host name of the machine on which the client is running.
- `_thread`: The client thread ID (Windows only).

MySQL Connector/J defines these attributes:

- `_client_name`: The client name
- `_client_version`: The client library version
- `_os`: The operating system (for example, `Linux`, `Win64`)

- `_client_license`: The connector license type
- `_platform`: The machine platform (for example, `x86_64`)
- `_runtime_vendor`: The Java runtime environment (JRE) vendor
- `_runtime_version`: The Java runtime environment (JRE) version

MySQL Connector/NET defines these attributes:

- `_client_version`: The client library version.
- `_os`: The operating system (for example, `Linux`, `Win64`).
- `_pid`: The client process ID.
- `_platform`: The machine platform (for example, `x86_64`).
- `_program_name`: The client name.
- `_thread`: The client thread ID (Windows only).

PHP defines attributes that depend on how it was compiled:

- Compiled using `libmysqlclient`: The standard `libmysqlclient` attributes, described previously.
- Compiled using `mysqlnd`: Only the `_client_name` attribute, with a value of `mysqlnd`.

Many MySQL client programs set a `program_name` attribute with a value equal to the client name. For example, `mysqladmin` and `mysqldump` set `program_name` to `mysqladmin` and `mysqldump`, respectively. MySQL Shell sets `program_name` to `mysqlsh`.

Some MySQL client programs define additional attributes:

- `mysql` (as of MySQL 8.0.17):
  - `os_user`: The name of the operating system user running the program. Available on Unix and Unix-like systems and Windows.
  - `os_sudouser`: The value of the `SUDO_USER` environment variable. Available on Unix and Unix-like systems.

  `mysql` connection attributes for which the value is empty are not sent.
- `mysqlbinlog`:
  - `_client_role`: `binary_log_listener`
- Replication slave connections:
  - `program_name`: `mysqld`
  - `_client_role`: `binary_log_listener`
  - `_client_replication_channel_name`: The channel name.
- `FEDERATED` storage engine connections:
  - `program_name`: `mysqld`

- `_client_role`: `federated_storage`

There are limits on the amount of connection attribute data transmitted from client to server:

- A fixed limit imposed by the client prior to connect time.
- A fixed limit imposed by the server at connect time.
- A configurable limit imposed by the Performance Schema at connect time.

For connections initiated using the C API, the `libmysqlclient` library imposes a limit of 64KB on the aggregate size of connection attribute data on the client side: Calls to `mysql_options()` that cause this limit to be exceeded produce a `CR_INVALID_PARAMETER_NO` error. Other MySQL Connectors may impose their own client-side limits on how much connection attribute data can be transmitted to the server.

On the server side, these size checks on connection attribute data occur:

- The server imposes a limit of 64KB on the aggregate size of connection attribute data it will accept. If a client attempts to send more than 64KB of attribute data, the server rejects the connection. Otherwise, the server considers the attribute buffer valid and tracks the size of the longest such buffer in the `Performance_schema_session_connect_attrs_longest_seen` status variable.
- For accepted connections, the Performance Schema checks aggregate attribute size against the value of the `performance_schema_session_connect_attrs_size` system variable. If attribute size exceeds this value, these actions take place:
  - The Performance Schema truncates the attribute data and increments the `Performance_schema_session_connect_attrs_lost` status variable, which indicates the number of connections for which attribute truncation occurred.
  - The Performance Schema writes a message to the error log if the `log_error_verbosity` system variable is greater than 1:

    ```
    Connection attributes of length N were truncated
    (N bytes lost)
    for connection N, user user_name@host_name
    (as user_name), auth: {yes|no}
    ```

    The information in the warning message is intended to help DBAs identify clients for which attribute truncation occurred.
  - A `_truncated` attribute is added to the session attributes with a value indicating how many bytes were lost, if the attribute buffer has sufficient space. This enables the Performance Schema to expose per-connection truncation information in the connection attribute tables. This information can be examined without having to check the error log.

26.12.9.1 The session_account_connect_attrs Table

Application programs can provide key-value connection attributes to be passed to the server at connect time. For descriptions of common attributes, see Section 26.12.9, “Performance Schema Connection Attribute Tables”.

The `session_account_connect_attrs` table contains connection attributes only for the current session, and other sessions associated with the session account. To see connection attributes for all sessions, use the `session_connect_attrs` table.

The `session_account_connect_attrs` table contains these columns:

- `PROCESSLIST_ID`

  The connection identifier for the session.

- `ATTR_NAME`

  The attribute name.

- `ATTR_VALUE`

  The attribute value.

- `ORDINAL_POSITION`

  The order in which the attribute was added to the set of connection attributes.

The `session_account_connect_attrs` table has these indexes:

- Primary key on (`PROCESSLIST_ID`, `ATTR_NAME`)

`TRUNCATE TABLE` is not permitted for the `session_account_connect_attrs` table.

26.12.9.2 The session_connect_attrs Table

Application programs can provide key-value connection attributes to be passed to the server at connect time. For descriptions of common attributes, see Section 26.12.9, “Performance Schema Connection Attribute Tables”.

The `session_connect_attrs` table contains connection attributes for all sessions. To see connection attributes only for the current session, and other sessions associated with the session account, use the `session_account_connect_attrs` table.

The `session_connect_attrs` table contains these columns:

- `PROCESSLIST_ID`

  The connection identifier for the session.

- `ATTR_NAME`

  The attribute name.

- `ATTR_VALUE`

  The attribute value.

- `ORDINAL_POSITION`

  The order in which the attribute was added to the set of connection attributes.

The `session_connect_attrs` table has these indexes:

- Primary key on (`PROCESSLIST_ID`, `ATTR_NAME`)

`TRUNCATE TABLE` is not permitted for the `session_connect_attrs` table.

26.12.10 Performance Schema User-Defined Variable Tables

The Performance Schema provides a `user_variables_by_thread` table that exposes user-defined variables. These are variables defined within a specific session and include a `@` character preceding the name; see Section 9.4, “User-Defined Variables”.

The `user_variables_by_thread` table contains these columns:

- `THREAD_ID`

  The thread identifier of the session in which the variable is defined.

- `VARIABLE_NAME`

  The variable name, without the leading `@` character.

- `VARIABLE_VALUE`

  The variable value.

The `user_variables_by_thread` table has these indexes:

- Primary key on (`THREAD_ID`, `VARIABLE_NAME`)

`TRUNCATE TABLE` is not permitted for the `user_variables_by_thread` table.

26.12.11 Performance Schema Replication Tables

The Performance Schema provides tables that expose replication information. This is similar to the information available from the `SHOW SLAVE STATUS` statement, but representation in table form is more accessible and has usability benefits:

- `SHOW SLAVE STATUS` output is useful for visual inspection, but not so much for programmatic use. By contrast, using the Performance Schema tables, information about slave status can be searched using general `SELECT` queries, including complex `WHERE` conditions, joins, and so forth.
- Query results can be saved in tables for further analysis, or assigned to variables and thus used in stored procedures.
- The replication tables provide better diagnostic information. For multithreaded slave operation, `SHOW SLAVE STATUS` reports all coordinator and worker thread errors using the `Last_SQL_Errno` and `Last_SQL_Error` fields, so only the most recent of those errors is visible and information can be lost. The replication tables store errors on a per-thread basis without loss of information.
- The last seen transaction is visible in the replication tables on a per-worker basis. This is information not avilable from `SHOW SLAVE STATUS`.
- Developers familiar with the Performance Schema interface can extend the replication tables to provide additional information by adding rows to the tables.

Replication Table Descriptions

The Performance Schema provides the following replication-related tables:

- Tables that contain information about the connection of the slave server to the master server:
  - `replication_connection_configuration`: Configuration parameters for connecting to the master

  - `replication_connection_status`: Current status of the connection to the master

- Tables that contain general (not thread-specific) information about the transaction applier:

  - `replication_applier_configuration`: Configuration parameters for the transaction applier on the slave.

  - `replication_applier_status`: Current status of the transaction applier on the slave.

- Tables that contain information about specific threads responsible for applying transactions received from the master:

  - `replication_applier_status_by_coordinator`: Status of the coordinator thread (empty unless the slave is multithreaded).

  - `replication_applier_status_by_worker`: Status of the applier thread or worker threads if the slave is multithreaded.

- Tables that contain information about channel based replication filters:

  - `replication_applier_filters`: Provides information about the replication filters configured on specific replication channels.

  - `replication_applier_global_filters`: Provides information about global replication filters, which apply to all replication channels.

- Tables that contain information about Group Replication members:

  - `replication_group_members`: Provides network and status information for group members.

  - `replication_group_member_stats`: Provides statistical information about group members and transactions in which they participate.

  For more information see Section 18.3, “Monitoring Group Replication”.

The following Performance Schema replication tables continue to be populated when the Performance Schema is disabled:

- `replication_connection_configuration`
- `replication_connection_status`
- `replication_applier_configuration`
- `replication_applier_status`
- `replication_applier_status_by_coordinator`
- `replication_applier_status_by_worker`

The exception is local timing information (start and end timestamps for transactions) in the replication tables `replication_connection_status`, `replication_applier_status_by_coordinator`, and `replication_applier_status_by_worker`. This information is not collected when the Performance Schema is disabled.

The following sections describe each replication table in more detail, including the correspondence between the columns produced by `SHOW SLAVE STATUS` and the replication table columns in which the same information appears.

The remainder of this introduction to the replication tables describes how the Performance Schema populates them and which fields from `SHOW SLAVE STATUS` are not represented in the tables.

Replication Table Life Cycle

The Performance Schema populates the replication tables as follows:

- Prior to execution of `CHANGE MASTER TO`, the tables are empty.
- After `CHANGE MASTER TO`, the configuration parameters can be seen in the tables. At this time, there are no active slave threads, so the `THREAD_ID` columns are `NULL` and the `SERVICE_STATE` columns have a value of `OFF`.
- After `START SLAVE`, non-`NULL` `THREAD_ID` values can be seen. Threads that are idle or active have a `SERVICE_STATE` value of `ON`. The thread that connects to the master server has a value of `CONNECTING` while it establishes the connection, and `ON` thereafter as long as the connection lasts.
- After `STOP SLAVE`, the `THREAD_ID` columns become `NULL` and the `SERVICE_STATE` columns for threads that no longer exist have a value of `OFF`.
- The tables are preserved after `STOP SLAVE` or threads dying due to an error.
- The `replication_applier_status_by_worker` table is nonempty only when the slave is operating in multithreaded mode. That is, if the `slave_parallel_workers` system variable is greater than 0, this table is populated when `START SLAVE` is executed, and the number of rows shows the number of workers.

SHOW SLAVE STATUS Information Not In the Replication Tables

The information in the Performance Schema replication tables differs somewhat from the information available from `SHOW SLAVE STATUS` because the tables are oriented toward use of global transaction identifiers (GTIDs), not file names and positions, and they represent server UUID values, not server ID values. Due to these differences, several `SHOW SLAVE STATUS` columns are not preserved in the Performance Schema replication tables, or are represented a different way:

- The following fields refer to file names and positions and are not preserved:

```
Master_Log_File
Read_Master_Log_Pos
Relay_Log_File
Relay_Log_Pos
Relay_Master_Log_File
Exec_Master_Log_Pos
Until_Condition
Until_Log_File
Until_Log_Pos
```

- The `Master_Info_File` field is not preserved. It refers to the `master.info` file used for the slave's master info repository, which has been superseded by crash-safe slave tables.
- The following fields are based on `server_id`, not `server_uuid`, and are not preserved:

```
Master_Server_Id
Replicate_Ignore_Server_Ids
```

- The `Skip_Counter` field is based on event counts, not GTIDs, and is not preserved.
- These error fields are aliases for `Last_SQL_Errno` and `Last_SQL_Error`, so they are not preserved:

```
Last_Errno
Last_Error
```

In the Performance Schema, this error information is available in the `LAST_ERROR_NUMBER` and `LAST_ERROR_MESSAGE` columns of the `replication_applier_status_by_coordinator` table (and `replication_applier_status_by_worker` if the slave is multithreaded). Those tables provide more specific per-thread error information than is available from `Last_Errno` and `Last_Error`.

- Fields that provide information about command-line filtering options is not preserved:

```
Replicate_Do_DB
Replicate_Ignore_DB
Replicate_Do_Table
Replicate_Ignore_Table
Replicate_Wild_Do_Table
Replicate_Wild_Ignore_Table
```

- The `Slave_IO_State` and `Slave_SQL_Running_State` fields are not preserved. If needed, these values can be obtained from the process list by using the `THREAD_ID` column of the appropriate replication table and joining it with the `ID` column in the `INFORMATION_SCHEMA PROCESSLIST` table to select the `STATE` column of the latter table.
- The `Executed_Gtid_Set` field can show a large set with a great deal of text. Instead, the Performance Schema tables show GTIDs of transactions that are currently being applied by the slave. Alternatively, the set of executed GTIDs can be obtained from the value of the `gtid_executed` system variable.
- The `Seconds_Behind_Master` and `Relay_Log_Space` fields are in to-be-decided status and are not preserved.

Status Variables Moved to Replication Tables

As of MySQL version 5.7.5, the following status variables (previously monitored using `SHOW STATUS`) were moved to the Perfomance Schema replication tables:

- `Slave_retried_transactions`
- `Slave_last_heartbeat`
- `Slave_received_heartbeats`
- `Slave_heartbeat_period`
- `Slave_running`

These status variables are now only relevant when a single replication channel is being used because they *only* report the status of the default replication channel. When multiple replication channels exist, use the Performance Schema replication tables described in this section, which report these variables for each existing replication channel.

Replication Channels

The first column of the replication Performance Schema tables is `CHANNEL_NAME`. This enables the tables to be viewed per replication channel. In a non-multisource replication setup there is a single default replication channel. When you are using multiple replication channels on a slave, you can filter the

tables per replication channel to monitor a specific replication channel. See Section 17.2.3, “Replication Channels” and Section 17.1.4.3, “Multi-Source Replication Monitoring” for more information.

26.12.11.1 The replication_connection_configuration Table

This table shows the configuration parameters used by the slave server for connecting to the master server. Parameters stored in the table can be changed at runtime with the `CHANGE MASTER TO` statement, as indicated in the column descriptions.

Compared to the `replication_connection_status` table, `replication_connection_configuration` changes less frequently. It contains values that define how the slave connects to the master and that remain constant during the connection, whereas `replication_connection_status` contains values that change during the connection.

The `replication_connection_configuration` table has the following columns. The column descriptions indicate the corresponding `CHANGE MASTER TO` options from which the column values are taken, and the table given later in this section shows the correspondence between `replication_connection_configuration` columns and `SHOW SLAVE STATUS` columns.

- `CHANNEL_NAME`

  The replication channel which this row is displaying. There is always a default replication channel, and more replication channels can be added. See Section 17.2.3, “Replication Channels” for more information. (`CHANGE MASTER TO` option: `FOR CHANNEL`)

- `HOST`

  The master host that the slave is connected to. (`CHANGE MASTER TO` option: `MASTER_HOST`)

- `PORT`

  The port used to connect to the master. (`CHANGE MASTER TO` option: `MASTER_PORT`)

- `USER`

  The user name of the account used to connect to the master. (`CHANGE MASTER TO` option: `MASTER_USER`)

- `NETWORK_INTERFACE`

  The network interface that the slave is bound to, if any. (`CHANGE MASTER TO` option: `MASTER_BIND`)

- `AUTO_POSITION`

  1 if autopositioning is in use; otherwise 0. (`CHANGE MASTER TO` option: `MASTER_AUTO_POSITION`)

- `SSL_ALLOWED`, `SSL_CA_FILE`, `SSL_CA_PATH`, `SSL_CERTIFICATE`, `SSL_CIPHER`, `SSL_KEY`, `SSL_VERIFY_SERVER_CERTIFICATE`, `SSL_CRL_FILE`, `SSL_CRL_PATH`

  These columns show the SSL parameters used by the slave to connect to the master, if any.

  `SSL_ALLOWED` has these values:

  - `Yes` if an SSL connection to the master is permitted
  - `No` if an SSL connection to the master is not permitted
  - `Ignored` if an SSL connection is permitted but the slave server does not have SSL support enabled

`CHANGE MASTER TO` options for the other SSL columns: `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_CIPHER`, `MASTER_SSL_CRL`, `MASTER_SSL_CRLPATH`, `MASTER_SSL_KEY`, `MASTER_SSL_VERIFY_SERVER_CERT`.

- `CONNECTION_RETRY_INTERVAL`

  The number of seconds between connect retries. (`CHANGE MASTER TO` option: `MASTER_CONNECT_RETRY`)

- `CONNECTION_RETRY_COUNT`

  The number of times the slave can attempt to reconnect to the master in the event of a lost connection. (`CHANGE MASTER TO` option: `MASTER_RETRY_COUNT`)

- `HEARTBEAT_INTERVAL`

  The replication heartbeat interval on a slave, measured in seconds. (`CHANGE MASTER TO` option: `MASTER_HEARTBEAT_PERIOD`)

- `TLS_VERSION`

  The TLS version used on the master. For TLS version information, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”. (`CHANGE MASTER TO` option: `MASTER_TLS_VERSION`)

- `PUBLIC_KEY_PATH`

  The path name to a file containing a slave-side copy of the public key required by the master for RSA key pair-based password exchange. The file must be in PEM format. This column applies to slaves that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. (`CHANGE MASTER TO` option: `MASTER_PUBLIC_KEY_PATH`)

  If `PUBLIC_KEY_PATH` is given and specifies a valid public key file, it takes precedence over `GET_PUBLIC_KEY`.

- `GET_PUBLIC_KEY`

  Whether to request from the master the public key required for RSA key pair-based password exchange. This column applies to slaves that authenticate with the `caching_sha2_password` authentication plugin. For that plugin, the master does not send the public key unless requested. (`CHANGE MASTER TO` option: `MASTER_GET_PUBLIC_KEY`)

  If `PUBLIC_KEY_PATH` is given and specifies a valid public key file, it takes precedence over `GET_PUBLIC_KEY`.

- `COMPRESSION_ALGORITHMS`

  The permitted compression algorithms for connections to the master. (`CHANGE MASTER TO` option: `MASTER_COMPRESSION_ALGORITHMS`)

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This column was added in MySQL 8.0.18.

- `ZSTD_COMPRESSION_LEVEL`

  The compression level to use for connections to the master that use the `zstd` compression algorithm. (`CHANGE MASTER TO` option: `MASTER_ZSTD_COMPRESSION_LEVEL`)

For more information, see Section 4.2.6, “Connection Compression Control”.

This column was added in MySQL 8.0.18.

The `replication_connection_configuration` table has these indexes:

- Primary key on (`CHANNEL_NAME`)

`TRUNCATE TABLE` is not permitted for the `replication_connection_configuration` table.

The following table shows the correspondence between `replication_connection_configuration` columns and `SHOW SLAVE STATUS` columns.

`replication_connection_configuration` Column	`SHOW SLAVE STATUS` Column
`CHANNEL_NAME`	`Channel_name`
`HOST`	`Master_Host`
`PORT`	`Master_Port`
`USER`	`Master_User`
`NETWORK_INTERFACE`	`Master_Bind`
`AUTO_POSITION`	`Auto_Position`
`SSL_ALLOWED`	`Master_SSL_Allowed`
`SSL_CA_FILE`	`Master_SSL_CA_File`
`SSL_CA_PATH`	`Master_SSL_CA_Path`
`SSL_CERTIFICATE`	`Master_SSL_Cert`
`SSL_CIPHER`	`Master_SSL_Cipher`
`SSL_KEY`	`Master_SSL_Key`
`SSL_VERIFY_SERVER_CERTIFICATE`	`Master_SSL_Verify_Server_Cert`
`SSL_CRL_FILE`	`Master_SSL_Crl`
`SSL_CRL_PATH`	`Master_SSL_Crlpath`
`CONNECTION_RETRY_INTERVAL`	`Connect_Retry`
`CONNECTION_RETRY_COUNT`	`Master_Retry_Count`
`HEARTBEAT_INTERVAL`	None
`TLS_VERSION`	`Master_TLS_Version`
`PUBLIC_KEY_PATH`	`Master_public_key_path`
`GET_PUBLIC_KEY`	`Get_master_public_key`
`MASTER_COMPRESSION_ALGORITHMS`	[None]
`MASTER_ZSTD_COMPRESSION_LEVEL`	[None]

26.12.11.2 The replication_connection_status Table

This table shows the current status of the I/O thread that handles the slave server connection to the master server, information on the last transaction queued in the relay log, and information on the transaction currently being queued in the relay log.

Compared to the `replication_connection_configuration` table, `replication_connection_status` changes more frequently. It contains values that change during the

connection, whereas `replication_connection_configuration` contains values which define how the slave connects to the master and that remain constant during the connection.

The `replication_connection_status` table has these columns:

- `CHANNEL_NAME`

  The replication channel which this row is displaying. There is always a default replication channel, and more replication channels can be added. See Section 17.2.3, “Replication Channels” for more information.

- `GROUP_NAME`

  If this server is a member of a group, shows the name of the group the server belongs to.

- `SOURCE_UUID`

  The `server_uuid` value from the master.

- `THREAD_ID`

  The I/O thread ID.

- `SERVICE_STATE`

  `ON` (thread exists and is active or idle), `OFF` (thread no longer exists), or `CONNECTING` (thread exists and is connecting to the master).

- `RECEIVED_TRANSACTION_SET`

  The set of global transaction IDs (GTIDs) corresponding to all transactions received by this slave. Empty if GTIDs are not in use. See GTID Sets for more information.

- `LAST_ERROR_NUMBER`, `LAST_ERROR_MESSAGE`

  The error number and error message of the most recent error that caused the I/O thread to stop. An error number of 0 and message of the empty string mean “no error.” If the `LAST_ERROR_MESSAGE` value is not empty, the error values also appear in the slave's error log.

  Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.

- `LAST_ERROR_TIMESTAMP`

  A timestamp in `'YYYY-MM-DD hh:mm:ss[.fraction]'` format that shows when the most recent I/O error took place.

- `LAST_HEARTBEAT_TIMESTAMP`

  A timestamp in `'YYYY-MM-DD hh:mm:ss[.fraction]'` format that shows when the most recent heartbeat signal was received by a replication slave.

- `COUNT_RECEIVED_HEARTBEATS`

  The total number of heartbeat signals that a replication slave received since the last time it was restarted or reset, or a `CHANGE MASTER TO` statement was issued.

- `LAST_QUEUED_TRANSACTION`

  The global transaction ID (GTID) of the last transaction that was queued to the relay log.

- `LAST_QUEUED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP`

  A timestamp in *`'YYYY-MM-DD hh:mm:ss[.fraction]'`* format that shows when the last transaction queued in the relay log was committed on the original master.

- `LAST_QUEUED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP`

  A timestamp in *`'YYYY-MM-DD hh:mm:ss[.fraction]'`* format that shows when the last transaction queued in the relay log was committed on the immediate master.

- `LAST_QUEUED_TRANSACTION_START_QUEUE_TIMESTAMP`

  A timestamp in *`'YYYY-MM-DD hh:mm:ss[.fraction]'`* format that shows when the last transaction was placed in the relay log queue by this I/O thread.

- `LAST_QUEUED_TRANSACTION_END_QUEUE_TIMESTAMP`

  A timestamp in *`'YYYY-MM-DD hh:mm:ss[.fraction]'`* format that shows when the last transaction was queued to the relay log files.

- `QUEUEING_TRANSACTION`

  The global transaction ID (GTID) of the currently queueing transaction in the relay log.

- `QUEUEING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP`

  A timestamp in *`'YYYY-MM-DD hh:mm:ss[.fraction]'`* format that shows when the currently queueing transaction was committed on the original master.

- `QUEUEING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP`

  A timestamp in *`'YYYY-MM-DD hh:mm:ss[.fraction]'`* format that shows when the currently queueing transaction was committed on the immediate master.

- `QUEUEING_TRANSACTION_START_QUEUE_TIMESTAMP`

  A timestamp in *`'YYYY-MM-DD hh:mm:ss[.fraction]'`* format that shows when the first event of the currently queueing transaction was written to the relay log by this I/O thread.

When the Performance Schema is disabled, local timing information is not collected, so the fields showing the start and end timestamps for queued transactions are zero.

The `replication_connection_status` table has these indexes:

- Primary key on (`CHANNEL_NAME`)
- Index on (`THREAD_ID`)

The following table shows the correspondence between `replication_connection_status` columns and `SHOW SLAVE STATUS` columns.

`replication_connection_status` Column	**`SHOW SLAVE STATUS` Column**
`SOURCE_UUID`	`Master_UUID`
`THREAD_ID`	None
`SERVICE_STATE`	`Slave_IO_Running`
`RECEIVED_TRANSACTION_SET`	`Retrieved_Gtid_Set`
`LAST_ERROR_NUMBER`	`Last_IO_Errno`

replication_connection_status Column	SHOW SLAVE STATUS Column
LAST_ERROR_MESSAGE	Last_IO_Error
LAST_ERROR_TIMESTAMP	Last_IO_Error_Timestamp

26.12.11.3 The replication_applier_configuration Table

This table shows the configuration parameters that affect transactions applied by the slave server. Parameters stored in the table can be changed at runtime with the CHANGE MASTER TO statement, as indicated in the column descriptions.

The replication_applier_configuration table has these columns:

- CHANNEL_NAME

  The replication channel which this row is displaying. There is always a default replication channel, and more replication channels can be added. See Section 17.2.3, “Replication Channels” for more information.

- DESIRED_DELAY

  The number of seconds that the slave must lag the master. (CHANGE MASTER TO option: MASTER_DELAY) See Section 17.4.10, “Delayed Replication” for more information.

- PRIVILEGE_CHECKS_USER

  The user account that provides the security context for the channel (CHANGE MASTER TO option: PRIVILEGE_CHECKS_USER). This is escaped so that it can be copied into a SQL statement to execute individual transactions. See Section 17.3.3, “Replication Privilege Checks” for more information.

The replication_applier_configuration table has these indexes:

- Primary key on (CHANNEL_NAME)

TRUNCATE TABLE is not permitted for the replication_applier_configuration table.

The following table shows the correspondence between replication_applier_configuration columns and SHOW SLAVE STATUS columns.

replication_applier_configuration Column	SHOW SLAVE STATUS Column
DESIRED_DELAY	SQL_Delay

26.12.11.4 The replication_applier_status Table

This table shows the current general transaction execution status on the slave server. The table provides information about general aspects of transaction applier status that are not specific to any thread involved. Thread-specific status information is available in the replication_applier_status_by_coordinator table (and replication_applier_status_by_worker if the slave is multithreaded).

The replication_applier_status table has these columns:

- CHANNEL_NAME

  The replication channel which this row is displaying. There is always a default replication channel, and more replication channels can be added. See Section 17.2.3, “Replication Channels” for more information.

- SERVICE_STATE

  Shows ON when the replication channel's applier threads are active or idle, OFF means that the applier threads are not active.

- REMAINING_DELAY

  If the slave is waiting for DESIRED_DELAY seconds to pass since the master applied an transaction, this field contains the number of delay seconds remaining. At other times, this field is NULL. (The DESIRED_DELAY value is stored in the replication_applier_configuration table.) See Section 17.4.10, “Delayed Replication” for more information.

- COUNT_TRANSACTIONS_RETRIES

  Shows the number of retries that were made because the slave SQL thread failed to apply a transaction. The maximum number of retries for a given transaction is set by the slave_transaction_retries system variable. The replication_applier_status_by_worker table shows detailed information on transaction retries for a single-threaded or multithreaded slave.

The replication_applier_status table has these indexes:

- Primary key on (CHANNEL_NAME)

TRUNCATE TABLE is not permitted for the replication_applier_status table.

The following table shows the correspondence between replication_applier_status columns and SHOW SLAVE STATUS columns.

replication_applier_status Column	SHOW SLAVE STATUS Column
SERVICE_STATE	None
REMAINING_DELAY	SQL_Remaining_Delay

26.12.11.5 The replication_applier_status_by_coordinator Table

For a multithreaded slave, the slave uses multiple worker threads and a coordinator thread to manage them, and this table shows the status of the coordinator thread. For a single-threaded slave, this table is empty. For a multithreaded slave, the replication_applier_status_by_worker table shows the status of the worker threads. This table provides information about the last transaction which was buffered by the coordinator thread to a worker’s queue, as well as the transaction it is currently buffering. The start timestamp refers to when this thread read the first event of the transaction from the relay log to buffer it to a worker’s queue, while the end timestamp refers to when the last event finished buffering to the worker’s queue.

The replication_applier_status_by_coordinator table has these columns:

- CHANNEL_NAME

  The replication channel which this row is displaying. There is always a default replication channel, and more replication channels can be added. See Section 17.2.3, “Replication Channels” for more information.

- THREAD_ID

  The SQL/coordinator thread ID.

- SERVICE_STATE

ON (thread exists and is active or idle) or OFF (thread no longer exists).

- LAST_ERROR_NUMBER, LAST_ERROR_MESSAGE

  The error number and error message of the most recent error that caused the SQL/coordinator thread to stop. An error number of 0 and message which is an empty string means “no error”. If the LAST_ERROR_MESSAGE value is not empty, the error values also appear in the slave's error log.

  Issuing RESET MASTER or RESET SLAVE resets the values shown in these columns.

  All error codes and messages displayed in the LAST_ERROR_NUMBER and LAST_ERROR_MESSAGE columns correspond to error values listed in Section B.3.1, “Server Error Message Reference”.

- LAST_ERROR_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format that shows when the most recent SQL/coordinator error occurred.

- LAST_PROCESSED_TRANSACTION

  The global transaction ID (GTID) of the last transaction processed by this coordinator.

- LAST_PROCESSED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format that shows when the last transaction processed by this coordinator was committed on the original master.

- LAST_PROCESSED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format that shows when the last transaction processed by this coordinator was committed on the immediate master.

- LAST_PROCESSED_TRANSACTION_START_BUFFER_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format that shows when this coordinator thread started writing the last transaction to the buffer of a worker thread.

- LAST_PROCESSED_TRANSACTION_END_BUFFER_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format that shows when the last transaction was written to the buffer of a worker thread by this coordinator thread.

- PROCESSING_TRANSACTION

  The global transaction ID (GTID) of the transaction that this coordinator thread is currently processing.

- PROCESSING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format that shows when the currently processing transaction was committed on the original master.

- PROCESSING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format that shows when the currently processing transaction was committed on the immediate master.

- PROCESSING_TRANSACTION_START_BUFFER_TIMESTAMP

A timestamp in `'YYYY-MM-DD hh:mm:ss[.fraction]'` format that shows when this coordinator thread started writing the currently processing transaction to the buffer of a worker thread.

When the Performance Schema is disabled, local timing information is not collected, so the fields showing the start and end timestamps for buffered transactions are zero.

The `replication_applier_status_by_coordinator` table has these indexes:

- Primary key on (`CHANNEL_NAME`)
- Index on (`THREAD_ID`)

The following table shows the correspondence between `replication_applier_status_by_coordinator` columns and `SHOW SLAVE STATUS` columns.

`replication_applier_status_by_coordinator` Column	**`SHOW SLAVE STATUS` Column**
`THREAD_ID`	None
`SERVICE_STATE`	`Slave_SQL_Running`
`LAST_ERROR_NUMBER`	`Last_SQL_Errno`
`LAST_ERROR_MESSAGE`	`Last_SQL_Error`
`LAST_ERROR_TIMESTAMP`	`Last_SQL_Error_Timestamp`

26.12.11.6 The replication_applier_status_by_worker Table

This table provides details of the transactions handled by applier threads on a replication slave or Group Replication group member. For a single-threaded slave, data is shown for the slave's single applier thread. For a multithreaded slave, data is shown individually for each applier thread. The applier threads on a multithreaded slave are sometimes called workers. The number of applier threads on a replication slave or Group Replication group member is set by the `slave_parallel_workers` system variable, which is set to zero for a single-threaded slave. A multithreaded slave also has a coordinator thread to manage the applier threads, and the status of this thread is shown in the `replication_applier_status_by_coordinator` table.

All error codes and messages displayed in the columns relating to errors correspond to error values listed in Section B.3.1, “Server Error Message Reference”.

When the Performance Schema is disabled, local timing information is not collected, so the fields showing the start and end timestamps for applied transactions are zero. The start timestamps in this table refer to when the worker started applying the first event, and the end timestamps refer to when the last event of the transaction was applied.

When a replication slave is restarted by a `START SLAVE` statement, the columns beginning `APPLYING_TRANSACTION` are reset. Before MySQL 8.0.13, these columns were not reset on a slave that was operating in single-threaded mode, only on a multithreaded slave.

The `replication_applier_status_by_worker` table has these columns:

- `CHANNEL_NAME`

  The replication channel which this row is displaying. There is always a default replication channel, and more replication channels can be added. See Section 17.2.3, “Replication Channels” for more information.

- `WORKER_ID`

  The worker identifier (same value as the `id` column in the `mysql.slave_worker_info` table). After `STOP SLAVE`, the `THREAD_ID` column becomes `NULL`, but the `WORKER_ID` value is preserved.

- `THREAD_ID`

  The worker thread ID.

- `SERVICE_STATE`

  `ON` (thread exists and is active or idle) or `OFF` (thread no longer exists).

- `LAST_ERROR_NUMBER`, `LAST_ERROR_MESSAGE`

  The error number and error message of the most recent error that caused the worker thread to stop. An error number of 0 and message of the empty string mean “no error”. If the `LAST_ERROR_MESSAGE` value is not empty, the error values also appear in the slave's error log.

  Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.

- `LAST_ERROR_TIMESTAMP`

  A timestamp in `'YYYY-MM-DD hh:mm:ss[.fraction]'` format that shows when the most recent worker error occurred.

- `LAST_APPLIED_TRANSACTION`

  The global transaction ID (GTID) of the last transaction applied by this worker.

- `LAST_APPLIED_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP`

  A timestamp in `'YYYY-MM-DD hh:mm:ss[.fraction]'` format that shows when the last transaction applied by this worker was committed on the original master.

- `LAST_APPLIED_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP`

  A timestamp in `'YYYY-MM-DD hh:mm:ss[.fraction]'` format that shows when the last transaction applied by this worker was committed on the immediate master.

- `LAST_APPLIED_TRANSACTION_START_APPLY_TIMESTAMP`

  A timestamp in `'YYYY-MM-DD hh:mm:ss[.fraction]'` format that shows when this worker started applying the last applied transaction.

- `LAST_APPLIED_TRANSACTION_END_APPLY_TIMESTAMP`

  A timestamp in `'YYYY-MM-DD hh:mm:ss[.fraction]'` format that shows when this worker finished applying the last applied transaction.

- `APPLYING_TRANSACTION`

  The global transaction ID (GTID) of the transaction this worker is currently applying.

- `APPLYING_TRANSACTION_ORIGINAL_COMMIT_TIMESTAMP`

  A timestamp in `'YYYY-MM-DD hh:mm:ss[.fraction]'` format that shows when the transaction this worker is currently applying was committed on the original master.

- APPLYING_TRANSACTION_IMMEDIATE_COMMIT_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format that shows when the transaction this worker is currently applying was committed on the immediate master.

- APPLYING_TRANSACTION_START_APPLY_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format that shows when this worker started its first attempt to apply the transaction that is currently being applied. Before MySQL 8.0.13, this timestamp was refreshed when a transaction was retried due to a transient error, so it showed the timestamp for the most recent attempt to apply the transaction.

- LAST_APPLIED_TRANSACTION_RETRIES_COUNT

  The number of times the last applied transaction was retried by the worker after the first attempt. If the transaction was applied at the first attempt, this number is zero.

- LAST_APPLIED_TRANSACTION_LAST_TRANSIENT_ERROR_NUMBER

  The error number of the last transient error that caused the transaction to be retried.

- LAST_APPLIED_TRANSACTION_LAST_TRANSIENT_ERROR_MESSAGE

  The message text for the last transient error that caused the transaction to be retried.

- LAST_APPLIED_TRANSACTION_LAST_TRANSIENT_ERROR_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format for the last transient error that caused the transaction to be retried.

- APPLYING_TRANSACTION_RETRIES_COUNT

  The number of times the transaction that is currently being applied was retried until this moment. If the transaction was applied at the first attempt, this number is zero.

- APPLYING_TRANSACTION_LAST_TRANSIENT_ERROR_NUMBER

  The error number of the last transient error that caused the current transaction to be retried.

- APPLYING_TRANSACTION_LAST_TRANSIENT_ERROR_MESSAGE

  The message text for the last transient error that caused the current transaction to be retried.

- APPLYING_TRANSACTION_LAST_TRANSIENT_ERROR_TIMESTAMP

  A timestamp in *'YYYY-MM-DD hh:mm:ss[.fraction]'* format for the last transient error that caused the current transaction to be retried.

The replication_applier_status_by_worker table has these indexes:

- Primary key on (CHANNEL_NAME, WORKER_ID)
- Index on (THREAD_ID)

The following table shows the correspondence between replication_applier_status_by_worker columns and SHOW SLAVE STATUS columns.

replication_applier_status_by_worker Column	SHOW SLAVE STATUS Column
WORKER_ID	None

`replication_applier_status_by_worker` Column	`SHOW SLAVE STATUS` Column
`THREAD_ID`	None
`SERVICE_STATE`	None
`LAST_ERROR_NUMBER`	`Last_SQL_Errno`
`LAST_ERROR_MESSAGE`	`Last_SQL_Error`
`LAST_ERROR_TIMESTAMP`	`Last_SQL_Error_Timestamp`

26.12.11.7 The replication_applier_global_filters Table

This table shows the global replication filters configured on this slave. The `replication_applier_global_filters` table has the following columns:

- `FILTER_NAME`

  The type of replication filter that has been configured.

- `FILTER_RULE`

  The rules configured for the replication filter type using either `--replicate-*` command options or `CHANGE REPLICATION FILTER`.

- `CONFIGURED_BY`

  The method used to configure the replication filter, can be one of:

  - `CHANGE_REPLICATION_FILTER` configured by a global replication filter using a `CHANGE REPLICATION FILTER` statement.
  - `STARTUP_OPTIONS` configured by a global replication filter using a `--replicate-*` option.

- `ACTIVE_SINCE`

  Timestamp of when the replication filter was configured.

26.12.11.8 The replication_applier_filters Table

This table shows the replication channel specific filters configured on this slave. Each row provides information on a replication channel's configured type of filter. The `replication_applier_filters` table has the following columns:

- `CHANNEL_NAME`

  The name of replication channel with a replication filter configured.

- `FILTER_NAME`

  The type of replication filter that has been configured for this replication channel.

- `FILTER_RULE`

  The rules configured for the replication filter type using either `--replicate-*` command options or `CHANGE REPLICATION FILTER`.

- `CONFIGURED_BY`

  The method used to configure the replication filter, can be one of:

  - `CHANGE_REPLICATION_FILTER` configured by a global replication filter using a `CHANGE REPLICATION FILTER` statement.
  - `STARTUP_OPTIONS` configured by a global replication filter using a `--replicate-*` option.
  - `CHANGE_REPLICATION_FILTER_FOR_CHANNEL` configured by a channel specific replication filter using a `CHANGE REPLICATION FILTER FOR CHANNEL` statement.
  - `STARTUP_OPTIONS_FOR_CHANNEL` configured by a channel specific replication filter using a `--replicate-*` option.
- `ACTIVE_SINCE`

  Timestamp of when the replication filter was configured.
- `COUNTER`

  The number of times the replication filter has been used since it was configured.

26.12.11.9 The replication_group_members Table

This table shows network and status information for replication group members. The network addresses shown are the addresses used to connect clients to the group, and should not be confused with the member's internal group communication address specified by `group_replication_local_address`.

The `replication_group_members` table has the following columns:

- `CHANNEL_NAME`

  Name of the Group Replication channel.
- `MEMBER_ID`

  The member server UUID. This has a different value for each member in the group. This also serves as a key because it is unique to each member.
- `MEMBER_HOST`

  Network address of this member (host name or IP address). Retrieved from the member's `hostname` variable. This is the address which clients connect to, unlike the group_replication_local_address which is used for internal group communication.
- `MEMBER_PORT`

  Port on which the server is listening. Retrieved from the member's `port` variable.
- `MEMBER_STATE`

  Current state of this member; can be any one of the following:
  - `ONLINE`: The member is in a fully functioning state.
  - `RECOVERING`: The server has joined a group from which it is retrieving data.
  - `OFFLINE`: The group replication plugin is installed but has not been started.
  - `ERROR`: The member has encountered an error, either during applying transactions or during the recovery phase, and is not participating in the group's transactions.

- `UNREACHABLE`: The failure detection process suspects that this member cannot be contacted, because the group messages have timed out.

See Section 18.3.1, “Group Replication Server States”.

- `MEMBER_ROLE`

  Role of the member in the group, either `PRIMARY` or `SECONDARY`.

- `MEMBER_VERSION`

  MySQL version of the member.

The `replication_group_members` table has these indexes:

- None

`TRUNCATE TABLE` is not permitted for the `replication_group_members` table.

26.12.11.10 The replication_group_member_stats Table

This table shows statistical information for replication group members. It is populated only when Group Replication is running.

The `replication_group_member_stats` table has the following columns:

- `CHANNEL_NAME`

  Name of the Group Replication channel

- `VIEW_ID`

  Current view identifier for this group.

- `MEMBER_ID`

  The member server UUID. This has a different value for each member in the group. This also serves as a key because it is unique to each member.

- `COUNT_TRANSACTIONS_IN_QUEUE`

  The number of transactions in the queue pending conflict detection checks. Once the transactions have been checked for conflicts, if they pass the check, they are queued to be applied as well.

- `COUNT_TRANSACTIONS_CHECKED`

  The number of transactions that have been checked for conflicts.

- `COUNT_CONFLICTS_DETECTED`

  The number of transactions that have not passed the conflict detection check.

- `COUNT_TRANSACTIONS_VALIDATING`

  Number of transactions which can be used for certification, but have not been garbage collected. Can be thought of as the current size of the conflict detection database against which each transaction is certified.

- `TRANSACTIONS_COMMITTED_ALL_MEMBERS`

The transactions that have been successfully committed on all members of the replication group, shown as GTID Sets. This is updated at a fixed time interval.

- `LAST_CONFLICT_FREE_TRANSACTION`

  The transaction identifier of the last conflict free transaction which was checked.

- `COUNT_TRANSACTIONS_REMOTE_IN_APPLIER_QUEUE`

  The number of transactions that this member has received from the replication group which are waiting to be applied.

- `COUNT_TRANSACTIONS_REMOTE_APPLIED`

  Number of transactions this member has received from the group and applied.

- `COUNT_TRANSACTIONS_LOCAL_PROPOSED`

  Number of transactions which originated on this member and were sent to the group.

- `COUNT_TRANSACTIONS_LOCAL_ROLLBACK`

  Number of transactions which originated on this member and were rolled back by the group.

The `replication_group_member_stats` table has these indexes:

- None

`TRUNCATE TABLE` is not permitted for the `replication_group_member_stats` table.

26.12.12 Performance Schema Lock Tables

The Performance Schema exposes lock information through these tables:

- `data_locks`: Data locks held and requested
- `data_lock_waits`: Relationships between data lock owners and data lock requestors blocked by those owners
- `metadata_locks`: Metadata locks held and requested
- `table_handles`: Table locks held and requested

The following sections describe these tables in more detail.

26.12.12.1 The data_locks Table

The `data_locks` table shows data locks held and requested. For information about which lock requests are blocked by which held locks, see Section 26.12.12.2, “The data_lock_waits Table”.

Example data lock information:

```
mysql> SELECT * FROM performance_schema.data_locks\G
*************************** 1. row ***************************
               ENGINE: INNODB
      ENGINE_LOCK_ID: 139664434886512:1059:139664350547912
ENGINE_TRANSACTION_ID: 2569
            THREAD_ID: 46
             EVENT_ID: 12
```

```
        OBJECT_SCHEMA: test
          OBJECT_NAME: t1
       PARTITION_NAME: NULL
    SUBPARTITION_NAME: NULL
           INDEX_NAME: NULL
OBJECT_INSTANCE_BEGIN: 139664350547912
            LOCK_TYPE: TABLE
            LOCK_MODE: IX
          LOCK_STATUS: GRANTED
            LOCK_DATA: NULL
*************************** 2. row ***************************
               ENGINE: INNODB
       ENGINE_LOCK_ID: 139664434886512:2:4:1:139664350544872
ENGINE_TRANSACTION_ID: 2569
            THREAD_ID: 46
             EVENT_ID: 12
        OBJECT_SCHEMA: test
          OBJECT_NAME: t1
       PARTITION_NAME: NULL
    SUBPARTITION_NAME: NULL
           INDEX_NAME: GEN_CLUST_INDEX
OBJECT_INSTANCE_BEGIN: 139664350544872
            LOCK_TYPE: RECORD
            LOCK_MODE: X
          LOCK_STATUS: GRANTED
            LOCK_DATA: supremum pseudo-record
```

Unlike most Performance Schema data collection, there are no instruments for controlling whether data lock information is collected or system variables for controlling data lock table sizes. The Performance Schema collects information that is already available in the server, so there is no memory or CPU overhead to generate this information or need for parameters that control its collection.

Use the `data_locks` table to help diagnose performance problems that occur during times of heavy concurrent load. For `InnoDB`, see the discussion of this topic at Section 15.14.2, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”.

The `data_locks` table has these columns:

- `ENGINE`

  The storage engine that holds or requested the lock.

- `ENGINE_LOCK_ID`

  The ID of the lock held or requested by the storage engine. Tuples of (`ENGINE_LOCK_ID`, `ENGINE`) values are unique.

  Lock ID formats are internal and subject to change at any time. Applications should not rely on lock IDs having a particular format.

- `ENGINE_TRANSACTION_ID`

  The storage engine internal ID of the transaction that requested the lock. This can be considered the owner of the lock, although the lock might still be pending, not actually granted yet (`LOCK_STATUS='WAITING'`).

  If the transaction has not yet performed any write operation (is still considered read only), the column contains internal data that users should not try to interpret. Otherwise, the column is the transaction ID.

  For `InnoDB`, to obtain details about the transaction, join this column with the `TRX_ID` column of the `INFORMATION_SCHEMA INNODB_TRX` table.

- `THREAD_ID`

  The thread ID of the session that created the lock. To obtain details about the thread, join this column with the `THREAD_ID` column of the Performance Schema `threads` table.

  `THREAD_ID` can be used together with `EVENT_ID` to determine the event during which the lock data structure was created in memory. (This event might have occurred before this particular lock request occurred, if the data structure is used to store multiple locks.)

- `EVENT_ID`

  The Performance Schema event that caused the lock. Tuples of (`THREAD_ID`, `EVENT_ID`) values implicitly identify a parent event in other Performance Schema tables:

  - The parent wait event in the `events_waits_xxx` tables
  - The parent stage event in the `events_stages_xxx` tables
  - The parent statement event in the `events_statements_xxx` tables
  - The parent transaction event in the `events_transactions_current` table

  To obtain details about the parent event, join the `THREAD_ID` and `EVENT_ID` columns with the columns of like name in the appropriate parent event table. See Section 26.19.2, “Obtaining Parent Event Information”.

- `OBJECT_SCHEMA`

  The schema that contains the locked table.

- `OBJECT_NAME`

  The name of the locked table.

- `PARTITION_NAME`

  The name of the locked partition, if any; `NULL` otherwise.

- `SUBPARTITION_NAME`

  The name of the locked subpartition, if any; `NULL` otherwise.

- `INDEX_NAME`

  The name of the locked index, if any; `NULL` otherwise.

  In practice, `InnoDB` always creates an index (`GEN_CLUST_INDEX`), so `INDEX_NAME` is non-`NULL` for `InnoDB` tables.

- `OBJECT_INSTANCE_BEGIN`

  The address in memory of the lock.

- `LOCK_TYPE`

  The type of lock.

  The value is storage engine dependent. For `InnoDB`, permitted values are `RECORD` for a row-level lock, `TABLE` for a table-level lock.

- `LOCK_MODE`

  How the lock is requested.

  The value is storage engine dependent. For `InnoDB`, permitted values are `S[,GAP]`, `X[,GAP]`, `IS[,GAP]`, `IX[,GAP]`, `AUTO_INC`, and `UNKNOWN`. Lock modes other than `AUTO_INC` and `UNKNOWN` indicate gap locks, if present. For information about `S`, `X`, `IS`, `IX`, and gap locks, refer to Section 15.7.1, “InnoDB Locking”.

- `LOCK_STATUS`

  The status of the lock request.

  The value is storage engine dependent. For `InnoDB`, permitted values are `GRANTED` (lock is held) and `WAITING` (lock is being waited for).

- `LOCK_DATA`

  The data associated with the lock, if any. The value is storage engine dependent. For `InnoDB`, a value is shown if the `LOCK_TYPE` is `RECORD`, otherwise the value is `NULL`. Primary key values of the locked record are shown for a lock placed on the primary key index. Secondary index values of the locked record are shown with primary key values appended for a lock placed on a secondary index. If there is no primary key, `LOCK_DATA` shows either the key values of a selected unique index or the unique `InnoDB` internal row ID number, according to the rules governing `InnoDB` clustered index use (see Section 15.6.2.1, “Clustered and Secondary Indexes”). `LOCK_DATA` reports “supremum pseudo-record” for a lock taken on a supremum pseudo-record. If the page containing the locked record is not in the buffer pool because it was written to disk while the lock was held, `InnoDB` does not fetch the page from disk. Instead, `LOCK_DATA` reports `NULL`.

The `data_locks` table has these indexes:

- Primary key on (`ENGINE_LOCK_ID`, `ENGINE`)
- Index on (`ENGINE_TRANSACTION_ID`, `ENGINE`)
- Index on (`THREAD_ID`, `EVENT_ID`)
- Index on (`OBJECT_SCHEMA`, `OBJECT_NAME`, `PARTITION_NAME`, `SUBPARTITION_NAME`)

`TRUNCATE TABLE` is not permitted for the `data_locks` table.

26.12.12.2 The data_lock_waits Table

The `data_lock_waits` table implements a many-to-many relationship showing which data lock requests in the `data_locks` table are blocked by which held data locks in the `data_locks` table. Held locks in `data_locks` appear in `data_lock_waits` only if they block some lock request.

This information enables you to understand data lock dependencies between sessions. The table exposes not only which lock a session or transaction is waiting for, but which session or transaction currently holds that lock.

Example data lock wait information:

```
mysql> SELECT * FROM performance_schema.data_lock_waits\G
*************************** 1. row ***************************
                          ENGINE: INNODB
      REQUESTING_ENGINE_LOCK_ID: 140211201964816:2:4:2:140211086465800
```

```
REQUESTING_ENGINE_TRANSACTION_ID: 1555
            REQUESTING_THREAD_ID: 47
             REQUESTING_EVENT_ID: 5
REQUESTING_OBJECT_INSTANCE_BEGIN: 140211086465800
         BLOCKING_ENGINE_LOCK_ID: 140211201963888:2:4:2:140211086459880
  BLOCKING_ENGINE_TRANSACTION_ID: 1554
              BLOCKING_THREAD_ID: 46
               BLOCKING_EVENT_ID: 12
  BLOCKING_OBJECT_INSTANCE_BEGIN: 140211086459880
```

Unlike most Performance Schema data collection, there are no instruments for controlling whether data lock information is collected or system variables for controlling data lock table sizes. The Performance Schema collects information that is already available in the server, so there is no memory or CPU overhead to generate this information or need for parameters that control its collection.

Use the `data_lock_waits` table to help diagnose performance problems that occur during times of heavy concurrent load. For `InnoDB`, see the discussion of this topic at Section 15.14.2, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”.

Because the columns in the `data_lock_waits` table are similar to those in the `data_locks` table, the column descriptions here are abbreviated. For more detailed column descriptions, see Section 26.12.12.1, “The data_locks Table”.

The `data_lock_waits` table has these columns:

- `ENGINE`

  The storage engine that requested the lock.

- `REQUESTING_ENGINE_LOCK_ID`

  The ID of the lock requested by the storage engine. To obtain details about the lock, join this column with the `ENGINE_LOCK_ID` column of the `data_locks` table.

- `REQUESTING_ENGINE_TRANSACTION_ID`

  The storage engine internal ID of the transaction that requested the lock.

- `REQUESTING_THREAD_ID`

  The thread ID of the session that requested the lock.

- `REQUESTING_EVENT_ID`

  The Performance Schema event that caused the lock request in the session that requested the lock.

- `REQUESTING_OBJECT_INSTANCE_BEGIN`

  The address in memory of the requested lock.

- `BLOCKING_ENGINE_LOCK_ID`

  The ID of the blocking lock. To obtain details about the lock, join this column with the `ENGINE_LOCK_ID` column of the `data_locks` table.

- `BLOCKING_ENGINE_TRANSACTION_ID`

  The storage engine internal ID of the transaction that holds the blocking lock.

- `BLOCKING_THREAD_ID`

The thread ID of the session that holds the blocking lock.

- `BLOCKING_EVENT_ID`

  The Performance Schema event that caused the blocking lock in the session that holds it.

- `BLOCKING_OBJECT_INSTANCE_BEGIN`

  The address in memory of the blocking lock.

The `data_lock_waits` table has these indexes:

- Index on (`REQUESTING_ENGINE_LOCK_ID`, `ENGINE`)
- Index on (`BLOCKING_ENGINE_LOCK_ID`, `ENGINE`)
- Index on (`REQUESTING_ENGINE_TRANSACTION_ID`, `ENGINE`)
- Index on (`BLOCKING_ENGINE_TRANSACTION_ID`, `ENGINE`)
- Index on (`REQUESTING_THREAD_ID`, `REQUESTING_EVENT_ID`)
- Index on (`BLOCKING_THREAD_ID`, `BLOCKING_EVENT_ID`)

`TRUNCATE TABLE` is not permitted for the `data_lock_waits` table.

26.12.12.3 The metadata_locks Table

MySQL uses metadata locking to manage concurrent access to database objects and to ensure data consistency; see Section 8.11.4, “Metadata Locking”. Metadata locking applies not just to tables, but also to schemas, stored programs (procedures, functions, triggers, scheduled events), tablespaces, user locks acquired with the `GET_LOCK()` function (see Section 12.14, “Locking Functions”), and locks acquired with the locking service described in Section 29.3.1, “The Locking Service”.

The Performance Schema exposes metadata lock information through the `metadata_locks` table:

- Locks that have been granted (shows which sessions own which current metadata locks).
- Locks that have been requested but not yet granted (shows which sessions are waiting for which metadata locks).
- Lock requests that have been killed by the deadlock detector.
- Lock requests that have timed out and are waiting for the requesting session's lock request to be discarded.

This information enables you to understand metadata lock dependencies between sessions. You can see not only which lock a session is waiting for, but which session currently holds that lock.

The `metadata_locks` table is read only and cannot be updated. It is autosized by default; to configure the table size, set the `performance_schema_max_metadata_locks` system variable at server startup.

Metadata lock instrumentation uses the `wait/lock/metadata/sql/mdl` instrument, which is enabled by default.

To control metadata lock instrumentation state at server startup, use lines like these in your `my.cnf` file:

- Enable:

```
[mysqld]
performance-schema-instrument='wait/lock/metadata/sql/mdl=ON'
```

- Disable:

```
[mysqld]
performance-schema-instrument='wait/lock/metadata/sql/mdl=OFF'
```

To control metadata lock instrumentation state at runtime, update the `setup_instruments` table:

- Enable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME = 'wait/lock/metadata/sql/mdl';
```

- Disable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME = 'wait/lock/metadata/sql/mdl';
```

The Performance Schema maintains `metadata_locks` table content as follows, using the `LOCK_STATUS` column to indicate the status of each lock:

- When a metadata lock is requested and obtained immediately, a row with a status of `GRANTED` is inserted.
- When a metadata lock is requested and not obtained immediately, a row with a status of `PENDING` is inserted.
- When a metadata lock previously requested is granted, its row status is updated to `GRANTED`.
- When a metadata lock is released, its row is deleted.
- When a pending lock request is canceled by the deadlock detector to break a deadlock (`ER_LOCK_DEADLOCK`), its row status is updated from `PENDING` to `VICTIM`.
- When a pending lock request times out (`ER_LOCK_WAIT_TIMEOUT`), its row status is updated from `PENDING` to `TIMEOUT`.
- When granted lock or pending lock request is killed, its row status is updated from `GRANTED` or `PENDING` to `KILLED`.
- The `VICTIM`, `TIMEOUT`, and `KILLED` status values are brief and signify that the lock row is about to be deleted.
- The `PRE_ACQUIRE_NOTIFY` and `POST_RELEASE_NOTIFY` status values are brief and signify that the metadata locking subsubsystem is notifying interested storage engines while entering lock acquisition operations or leaving lock release operations.

The `metadata_locks` table has these columns:

- `OBJECT_TYPE`

  The type of lock used in the metadata lock subsystem. The value is one of `GLOBAL`, `SCHEMA`, `TABLE`, `FUNCTION`, `PROCEDURE`, `TRIGGER` (currently unused), `EVENT`, `COMMIT`, `USER LEVEL LOCK`, `TABLESPACE`, or `LOCKING SERVICE`.

A value of `USER LEVEL LOCK` indicates a lock acquired with `GET_LOCK()`. A value of `LOCKING SERVICE` indicates a lock acquired with the locking service described in Section 29.3.1, "The Locking Service".

- `OBJECT_SCHEMA`

  The schema that contains the object.

- `OBJECT_NAME`

  The name of the instrumented object.

- `OBJECT_INSTANCE_BEGIN`

  The address in memory of the instrumented object.

- `LOCK_TYPE`

  The lock type from the metadata lock subsystem. The value is one of `INTENTION_EXCLUSIVE`, `SHARED`, `SHARED_HIGH_PRIO`, `SHARED_READ`, `SHARED_WRITE`, `SHARED_UPGRADABLE`, `SHARED_NO_WRITE`, `SHARED_NO_READ_WRITE`, or `EXCLUSIVE`.

- `LOCK_DURATION`

  The lock duration from the metadata lock subsystem. The value is one of `STATEMENT`, `TRANSACTION`, or `EXPLICIT`. The `STATEMENT` and `TRANSACTION` values signify locks that are released implicitly at statement or transaction end, respectively. The `EXPLICIT` value signifies locks that survive statement or transaction end and are released by explicit action, such as global locks acquired with `FLUSH TABLES WITH READ LOCK`.

- `LOCK_STATUS`

  The lock status from the metadata lock subsystem. The value is one of `PENDING`, `GRANTED`, `VICTIM`, `TIMEOUT`, `KILLED`, `PRE_ACQUIRE_NOTIFY`, or `POST_RELEASE_NOTIFY`. The Performance Schema assigns these values as described previously.

- `SOURCE`

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved.

- `OWNER_THREAD_ID`

  The thread requesting a metadata lock.

- `OWNER_EVENT_ID`

  The event requesting a metadata lock.

The `metadata_locks` table has these indexes:

- Primary key on (`OBJECT_INSTANCE_BEGIN`)
- Index on (`OBJECT_TYPE`, `OBJECT_SCHEMA`, `OBJECT_NAME`)
- Index on (`OWNER_THREAD_ID`, `OWNER_EVENT_ID`)

`TRUNCATE TABLE` is not permitted for the `metadata_locks` table.

26.12.12.4 The table_handles Table

The Performance Schema exposes table lock information through the `table_handles` table to show the table locks currently in effect for each opened table handle. `table_handles` reports what is recorded by the table lock instrumentation. This information shows which table handles the server has open, how they are locked, and by which sessions.

The `table_handles` table is read only and cannot be updated. It is autosized by default; to configure the table size, set the `performance_schema_max_table_handles` system variable at server startup.

Table lock instrumentation uses the `wait/lock/table/sql/handler` instrument, which is enabled by default.

To control table lock instrumentation state at server startup, use lines like these in your `my.cnf` file:

- Enable:

```
[mysqld]
performance-schema-instrument='wait/lock/table/sql/handler=ON'
```

- Disable:

```
[mysqld]
performance-schema-instrument='wait/lock/table/sql/handler=OFF'
```

To control table lock instrumentation state at runtime, update the `setup_instruments` table:

- Enable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME = 'wait/lock/table/sql/handler';
```

- Disable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME = 'wait/lock/table/sql/handler';
```

The `table_handles` table has these columns:

- `OBJECT_TYPE`

  The table opened by a table handle.

- `OBJECT_SCHEMA`

  The schema that contains the object.

- `OBJECT_NAME`

  The name of the instrumented object.

- `OBJECT_INSTANCE_BEGIN`

  The table handle address in memory.

- `OWNER_THREAD_ID`

  The thread owning the table handle.

- `OWNER_EVENT_ID`

  The event which caused the table handle to be opened.

- `INTERNAL_LOCK`

  The table lock used at the SQL level. The value is one of `READ`, `READ WITH SHARED LOCKS`, `READ HIGH PRIORITY`, `READ NO INSERT`, `WRITE ALLOW WRITE`, `WRITE CONCURRENT INSERT`, `WRITE LOW PRIORITY`, or `WRITE`. For information about these lock types, see the `include/thr_lock.h` source file.

- `EXTERNAL_LOCK`

  The table lock used at the storage engine level. The value is one of `READ EXTERNAL` or `WRITE EXTERNAL`.

The `table_handles` table has these indexes:

- Primary key on (`OBJECT_INSTANCE_BEGIN`)
- Index on (`OBJECT_TYPE`, `OBJECT_SCHEMA`, `OBJECT_NAME`)
- Index on (`OWNER_THREAD_ID`, `OWNER_EVENT_ID`)

`TRUNCATE TABLE` is not permitted for the `table_handles` table.

26.12.13 Performance Schema System Variable Tables

The MySQL server maintains many system variables that indicate how it is configured (see Section 5.1.8, “Server System Variables”). System variable information is available in these Performance Schema tables:

- `global_variables`: Global system variables. An application that wants only global values should use this table.
- `session_variables`: System variables for the current session. An application that wants all system variable values for its own session should use this table. It includes the session variables for its session, as well as the values of global variables that have no session counterpart.
- `variables_by_thread`: Session system variables for each active session. An application that wants to know the session variable values for specific sessions should use this table. It includes session variables only, identified by thread ID.
- `persisted_variables`: Provides a SQL interface to the `mysqld-auto.cnf` file that stores persisted global system variable settings. See Section 26.12.13.1, “Performance Schema persisted_variables Table”.
- `variables_info`: Shows, for each system variable, the source from which it was most recently set, and its range of values. See Section 26.12.13.2, “Performance Schema variables_info Table”.

The session variable tables (`session_variables`, `variables_by_thread`) contain information only for active sessions, not terminated sessions.

`TRUNCATE TABLE` is not supported for Performance Schema system variable tables.

The `global_variables` and `session_variables` tables have these columns:

- `VARIABLE_NAME`

  The system variable name.

- `VARIABLE_VALUE`

  The system variable value. For `global_variables`, this column contains the global value. For `session_variables`, this column contains the variable value in effect for the current session.

The `global_variables` and `session_variables` tables have these indexes:

- Primary key on (`VARIABLE_NAME`)

The `variables_by_thread` table has these columns:

- `THREAD_ID`

  The thread identifier of the session in which the system variable is defined.

- `VARIABLE_NAME`

  The system variable name.

- `VARIABLE_VALUE`

  The session variable value for the session named by the `THREAD_ID` column.

The `variables_by_thread` table has these indexes:

- Primary key on (`THREAD_ID`, `VARIABLE_NAME`)

The `variables_by_thread` table contains system variable information only about foreground threads. If not all threads are instrumented by the Performance Schema, this table will miss some rows. In this case, the `Performance_schema_thread_instances_lost` status variable will be greater than zero.

26.12.13.1 Performance Schema persisted_variables Table

The `persisted_variables` table provides an SQL interface to the `mysqld-auto.cnf` file that stores persisted global system variable settings, enabling the file contents to be inspected at runtime using `SELECT` statements. Variables are persisted using `SET PERSIST` or `PERSIST_ONLY` statements; see Section 13.7.6.1, “SET Syntax for Variable Assignment”. The table contains a row for each persisted system variable in the file. Variables not persisted do not appear in the table.

For information about persisted system variables, see Section 13.7.6.1, “SET Syntax for Variable Assignment”.

Suppose that `mysqld-auto.cnf` looks like this (slightly reformatted):

```
{
  "Version": 1,
  "mysql_server": {
    "max_connections": {
      "Value": "1000",
      "Metadata": {
        "Timestamp": 1.519921706e+15,
        "User": "root",
        "Host": "localhost"
      }
```

```
    },
    "autocommit": {
      "Value": "ON",
      "Metadata": {
        "Timestamp": 1.519921707e+15,
        "User": "root",
        "Host": "localhost"
      }
    }
  }
}
```

Then `persisted_variables` has these contents:

```
mysql> SELECT * FROM performance_schema.persisted_variables;
+-----------------+----------------+
| VARIABLE_NAME   | VARIABLE_VALUE |
+-----------------+----------------+
| autocommit      | ON             |
| max_connections | 1000           |
+-----------------+----------------+
```

`persisted_variables` has these columns:

- `VARIABLE_NAME`

  The variable name listed in `mysqld-auto.cnf`.

- `VARIABLE_VALUE`

  The value listed for the variable in `mysqld-auto.cnf`.

`persisted_variables` has these indexes:

- Primary key on (`VARIABLE_NAME`)

`TRUNCATE TABLE` is not permitted for the `persisted_variables` table.

26.12.13.2 Performance Schema variables_info Table

The `variables_info` table shows, for each system variable, the source from which it was most recently set, and its range of values.

`variables_info` has these columns:

- `VARIABLE_NAME`

  The variable name.

- `VARIABLE_SOURCE`

  The source from which the variable was most recently set:

  - `COMMAND_LINE`

    The variable was set on the command line.

  - `COMPILED`

    The variable has its compiled-in default value. `COMPILED` is the value used for variables not set any other way.

- `DYNAMIC`

  The variable was set at runtime. This includes variables set within files specified using the `init_file` system variable.

- `EXPLICIT`

  The variable was set from an option file named with the `--defaults-file` option.

- `EXTRA`

  The variable was set from an option file named with the `--defaults-extra-file` option.

- `GLOBAL`

  The variable was set from a global option file. This includes option files not covered by `EXPLICIT`, `EXTRA`, `LOGIN`, `PERSISTED`, `SERVER`, or `USER`.

- `LOGIN`

  The variable was set from a user-specific login path file (`~/.mylogin.cnf`).

- `PERSISTED`

  The variable was set from a server-specific `mysqld-auto.cnf` option file. No row has this value if the server was started with `persisted_globals_load` disabled.

- `SERVER`

  The variable was set from a server-specific `$MYSQL_HOME/my.cnf` option file. For details about how `MYSQL_HOME` is set, see Section 4.2.2.2, “Using Option Files”.

- `USER`

  The variable was set from a user-specific `~/.my.cnf` option file.

- `VARIABLE_PATH`

  If the variable was set from an option file, `VARIABLE_PATH` is the path name of that file. Otherwise, the value is the empty string.

- `MIN_VALUE`, `MAX_VALUE`

  The minimum and maximum permitted values for the variable. Both are 0 for variables that have no such values (that is, variables that are not numeric).

- `SET_TIME`

  The time at which the variable was most recently set. The default is the time at which the server initialized global system variables during startup.

- `SET_USER`, `SET_HOST`

  The user name and host name of the client user that most recently set the variable. If a client connects as `user17` from host `host34.example.com` using the account `'user17'@'%.example.com`, `SET_USER` and `SET_HOST` will be `user17` and `host34.example.com`, respectively. For proxy user connections, these values correspond to the external (proxy) user, not the proxied user against which

privilege checking is performed. The default for each column is the empty string, indicating that the variable has not been set since server startup.

The `variables_info` table has these indexes:

- None

`TRUNCATE TABLE` is not permitted for the `variables_info` table.

If a variable with a `VARIABLE_SOURCE` value other than `DYNAMIC` is set at runtime, `VARIABLE_SOURCE` becomes `DYNAMIC` and `VARIABLE_PATH` becomes the empty string.

A system variable that has only a session value (such as `debug_sync`) cannot be set at startup or persisted. For session-only system variables, `VARIABLE_SOURCE` can be only `COMPILED` or `DYNAMIC`.

If a system variable has an unexpected `VARIABLE_SOURCE` value, consider your server startup method. For example, `mysqld_safe` reads option files and passes certain options it finds there as part of the command line that it uses to start `mysqld`. Consequently, some system variables that you set in option files might display in `variables_info` as `COMMAND_LINE`, rather than as `GLOBAL` or `SERVER` as you might otherwise expect.

Some sample queries that use the `variables_info` table, with representive output:

- Display variables set on the command line:

```
mysql> SELECT VARIABLE_NAME
       FROM performance_schema.variables_info
       WHERE VARIABLE_SOURCE = 'COMMAND_LINE'
       ORDER BY VARIABLE_NAME;
+---------------+
| VARIABLE_NAME |
+---------------+
| basedir       |
| datadir       |
| log_error     |
| pid_file      |
| plugin_dir    |
| port          |
+---------------+
```

- Display variables set from persistent storage:

```
mysql> SELECT VARIABLE_NAME
       FROM performance_schema.variables_info
       WHERE VARIABLE_SOURCE = 'PERSISTED'
       ORDER BY VARIABLE_NAME;
+--------------------------+
| VARIABLE_NAME            |
+--------------------------+
| event_scheduler          |
| max_connections          |
| validate_password.policy |
+--------------------------+
```

- Join `variables_info` with the `global_variables` table to display the current values of persisted variables, together with their range of values:

```
mysql> SELECT
         VI.VARIABLE_NAME, GV.VARIABLE_VALUE,
         VI.MIN_VALUE,VI.MAX_VALUE
```

```
       FROM performance_schema.variables_info AS VI
         INNER JOIN performance_schema.global_variables AS GV
         USING(VARIABLE_NAME)
       WHERE VI.VARIABLE_SOURCE = 'PERSISTED'
       ORDER BY VARIABLE_NAME;
+--------------------------+----------------+-----------+-----------+
| VARIABLE_NAME            | VARIABLE_VALUE | MIN_VALUE | MAX_VALUE |
+--------------------------+----------------+-----------+-----------+
| event_scheduler          | ON             | 0         | 0         |
| max_connections          | 200            | 1         | 100000    |
| validate_password.policy | STRONG         | 0         | 0         |
+--------------------------+----------------+-----------+-----------+
```

26.12.14 Performance Schema Status Variable Tables

The MySQL server maintains many status variables that provide information about its operation (see Section 5.1.10, “Server Status Variables”). Status variable information is available in these Performance Schema tables:

- `global_status`: Global status variables. An application that wants only global values should use this table.
- `session_status`: Status variables for the current session. An application that wants all status variable values for its own session should use this table. It includes the session variables for its session, as well as the values of global variables that have no session counterpart.
- `status_by_thread`: Session status variables for each active session. An application that wants to know the session variable values for specific sessions should use this table. It includes session variables only, identified by thread ID.

There are also summary tables that provide status variable information aggregated by account, host name, and user name. See Section 26.12.17.12, “Status Variable Summary Tables”.

The session variable tables (`session_status`, `status_by_thread`) contain information only for active sessions, not terminated sessions.

The Performance Schema collects statistics for global status variables only for threads for which the `INSTRUMENTED` value is `YES` in the `threads` table. Statistics for session status variables are always collected, regardless of the `INSTRUMENTED` value.

The Performance Schema does not collect statistics for `Com_`*`xxx`* status variables in the status variable tables. To obtain global and per-session statement execution counts, use the `events_statements_summary_global_by_event_name` and `events_statements_summary_by_thread_by_event_name` tables, respectively. For example:

```
SELECT EVENT_NAME, COUNT_STAR
FROM performance_schema.events_statements_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'statement/sql/%';
```

The `global_status` and `session_status` tables have these columns:

- `VARIABLE_NAME`

  The status variable name.

- `VARIABLE_VALUE`

  The status variable value. For `global_status`, this column contains the global value. For `session_status`, this column contains the variable value for the current session.

The `global_status` and `session_status` tables have these indexes:

- Primary key on (`VARIABLE_NAME`)

The `status_by_thread` table contains the status of each active thread. It has these columns:

- `THREAD_ID`

  The thread identifier of the session in which the status variable is defined.

- `VARIABLE_NAME`

  The status variable name.

- `VARIABLE_VALUE`

  The session variable value for the session named by the `THREAD_ID` column.

The `status_by_thread` table has these indexes:

- Primary key on (`THREAD_ID`, `VARIABLE_NAME`)

The `status_by_thread` table contains status variable information only about foreground threads. If the `performance_schema_max_thread_instances` system variable is not autoscaled (signified by a value of −1) and the maximum permitted number of instrumented thread objects is not greater than the number of background threads, the table will be empty.

The Performance Schema supports `TRUNCATE TABLE` for status variable tables as follows:

- `global_status`: Resets thread, account, host, and user status. Resets global status variables except those that the server never resets.
- `session_status`: Not supported.
- `status_by_thread`: Aggregates status for all threads to the global status and account status, then resets thread status. If account statistics are not collected, the session status is added to host and user status, if host and user status are collected.

  Account, host, and user statistics are not collected if the `performance_schema_accounts_size`, `performance_schema_hosts_size`, and `performance_schema_users_size` system variables, respectively, are set to 0.

`FLUSH STATUS` adds the session status from all active sessions to the global status variables, resets the status of all active sessions, and resets account, host, and user status values aggregated from disconnected sessions.

26.12.15 Performance Schema Thread Pool Tables

Note

The Performance Schema tables described here are available as of MySQL 8.0.14. Prior to MySQL 8.0.14, use the corresponding `INFORMATION_SCHEMA` tables instead; see Section 25.40, “INFORMATION_SCHEMA Thread Pool Tables”.

The following sections describe the Performance Schema tables associated with the thread pool plugin (see Section 5.6.3, “MySQL Enterprise Thread Pool”). They provide information about thread pool operation:

- `tp_thread_group_state`: Information about thread pool thread group states
- `tp_thread_group_stats`: Thread group statistics
- `tp_thread_state`: Information about thread pool thread states

Rows in these tables represent snapshots in time. In the case of `tp_thread_state`, all rows for a thread group comprise a snapshot in time. Thus, the MySQL server holds the mutex of the thread group while producing the snapshot. But it does not hold mutexes on all thread groups at the same time, to prevent a statement against `tp_thread_state` from blocking the entire MySQL server.

The Performance Schema thread pool tables are implemented by the thread pool plugin and are loaded and unloaded when that plugin is loaded and unloaded (see Section 5.6.3.2, “Thread Pool Installation”). No special configuration step for the tables is needed. However, the tables depend on the thread pool plugin being enabled. If the thread pool plugin is loaded but disabled, the tables are not created.

26.12.15.1 The tp_thread_group_state Table

Note

The Performance Schema table described here is available as of MySQL 8.0.14. Prior to MySQL 8.0.14, use the corresponding `INFORMATION_SCHEMA` table instead; see Section 25.40.1, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table”.

The `tp_thread_group_state` table has one row per thread group in the thread pool. Each row provides information about the current state of a group.

The `tp_thread_group_state` table has these columns:

- `TP_GROUP_ID`

  The thread group ID. This is a unique key within the table.

- `CONSUMER_THREADS`

  The number of consumer threads. There is at most one thread ready to start executing if the active threads become stalled or blocked.

- `RESERVE_THREADS`

  The number of threads in the reserved state. This means that they will not be started until there is a need to wake a new thread and there is no consumer thread. This is where most threads end up when the thread group has created more threads than needed for normal operation. Often a thread group needs additional threads for a short while and then does not need them again for a while. In this case, they go into the reserved state and remain until needed again. They take up some extra memory resources, but no extra computing resources.

- `CONNECT_THREAD_COUNT`

  The number of threads that are processing or waiting to process connection initialization and authentication. There can be a maximum of four connection threads per thread group; these threads expire after a period of inactivity.

- `CONNECTION_COUNT`

  The number of connections using this thread group.

- QUEUED_QUERIES

  The number of statements waiting in the high-priority queue.

- QUEUED_TRANSACTIONS

  The number of statements waiting in the low-priority queue. These are the initial statements for transactions that have not started, so they also represent queued transactions.

- STALL_LIMIT

  The value of the thread_pool_stall_limit system variable for the thread group. This is the same value for all thread groups.

- PRIO_KICKUP_TIMER

  The value of the thread_pool_prio_kickup_timer system variable for the thread group. This is the same value for all thread groups.

- ALGORITHM

  The value of the thread_pool_algorithm system variable for the thread group. This is the same value for all thread groups.

- THREAD_COUNT

  The number of threads started in the thread pool as part of this thread group.

- ACTIVE_THREAD_COUNT

  The number of threads active in executing statements.

- STALLED_THREAD_COUNT

  The number of stalled statements in the thread group. A stalled statement could be executing, but from a thread pool perspective it is stalled and making no progress. A long-running statement quickly ends up in this category.

- WAITING_THREAD_NUMBER

  If there is a thread handling the polling of statements in the thread group, this specifies the thread number within this thread group. It is possible that this thread could be executing a statement.

- OLDEST_QUEUED

  How long in milliseconds the oldest queued statement has been waiting for execution.

- MAX_THREAD_IDS_IN_GROUP

  The maximum thread ID of the threads in the group. This is the same as MAX(TP_THREAD_NUMBER) for the threads when selected from the tp_thread_state table. That is, these two queries are equivalent:

```
SELECT TP_GROUP_ID, MAX_THREAD_IDS_IN_GROUP
FROM tp_thread_group_state;

SELECT TP_GROUP_ID, MAX(TP_THREAD_NUMBER)
FROM tp_thread_state GROUP BY TP_GROUP_ID;
```

The tp_thread_group_state table has these indexes:

- Unique index on (TP_GROUP_ID)

TRUNCATE TABLE is not permitted for the tp_thread_group_state table.

26.12.15.2 The tp_thread_group_stats Table

Note

The Performance Schema table described here is available as of MySQL 8.0.14. Prior to MySQL 8.0.14, use the corresponding INFORMATION_SCHEMA table instead; see Section 25.40.2, "The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table".

The tp_thread_group_stats table reports statistics per thread group. There is one row per group.

The tp_thread_group_stats table has these columns:

- TP_GROUP_ID

  The thread group ID. This is a unique key within the table.

- CONNECTIONS_STARTED

  The number of connections started.

- CONNECTIONS_CLOSED

  The number of connections closed.

- QUERIES_EXECUTED

  The number of statements executed. This number is incremented when a statement starts executing, not when it finishes.

- QUERIES_QUEUED

  The number of statements received that were queued for execution. This does not count statements that the thread group was able to begin executing immediately without queuing, which can happen under the conditions described in Section 5.6.3.3, "Thread Pool Operation".

- THREADS_STARTED

  The number of threads started.

- PRIO_KICKUPS

  The number of statements that have been moved from low-priority queue to high-priority queue based on the value of the thread_pool_prio_kickup_timer system variable. If this number increases quickly, consider increasing the value of that variable. A quickly increasing counter means that the priority system is not keeping transactions from starting too early. For InnoDB, this most likely means deteriorating performance due to too many concurrent transactions..

- STALLED_QUERIES_EXECUTED

  The number of statements that have become defined as stalled due to executing for longer than the value of the thread_pool_stall_limit system variable.

- BECOME_CONSUMER_THREAD

The number of times thread have been assigned the consumer thread role.

- `BECOME_RESERVE_THREAD`

  The number of times threads have been assigned the reserve thread role.

- `BECOME_WAITING_THREAD`

  The number of times threads have been assigned the waiter thread role. When statements are queued, this happens very often, even in normal operation, so rapid increases in this value are normal in the case of a highly loaded system where statements are queued up.

- `WAKE_THREAD_STALL_CHECKER`

  The number of times the stall check thread decided to wake or create a thread to possibly handle some statements or take care of the waiter thread role.

- `SLEEP_WAITS`

  The number of `THD_WAIT_SLEEP` waits. These occur when threads go to sleep (for example, by calling the `SLEEP()` function).

- `DISK_IO_WAITS`

  The number of `THD_WAIT_DISKIO` waits. These occur when threads perform disk I/O that is likely to not hit the file system cache. Such waits occur when the buffer pool reads and writes data to disk, not for normal reads from and writes to files.

- `ROW_LOCK_WAITS`

  The number of `THD_WAIT_ROW_LOCK` waits for release of a row lock by another transaction.

- `GLOBAL_LOCK_WAITS`

  The number of `THD_WAIT_GLOBAL_LOCK` waits for a global lock to be released.

- `META_DATA_LOCK_WAITS`

  The number of `THD_WAIT_META_DATA_LOCK` waits for a metadata lock to be released.

- `TABLE_LOCK_WAITS`

  The number of `THD_WAIT_TABLE_LOCK` waits for a table to be unlocked that the statement needs to access.

- `USER_LOCK_WAITS`

  The number of `THD_WAIT_USER_LOCK` waits for a special lock constructed by the user thread.

- `BINLOG_WAITS`

  The number of `THD_WAIT_BINLOG_WAITS` waits for the binary log to become free.

- `GROUP_COMMIT_WAITS`

  The number of `THD_WAIT_GROUP_COMMIT` waits. These occur when a group commit must wait for the other parties to complete their part of a transaction.

- `FSYNC_WAITS`

The number of THD_WAIT_SYNC waits for a file sync operation.

The tp_thread_group_stats table has these indexes:

- Unique index on (TP_GROUP_ID)

TRUNCATE TABLE is not permitted for the tp_thread_group_stats table.

26.12.15.3 The tp_thread_state Table

Note

The Performance Schema table described here is available as of MySQL 8.0.14. Prior to MySQL 8.0.14, use the corresponding INFORMATION_SCHEMA table instead; see Section 25.40.3, "The INFORMATION_SCHEMA TP_THREAD_STATE Table".

The tp_thread_state table has one row per thread created by the thread pool to handle connections.

The tp_thread_state table has these columns:

- TP_GROUP_ID

  The thread group ID.

- TP_THREAD_NUMBER

  The ID of the thread within its thread group. TP_GROUP_ID and TP_THREAD_NUMBER together provide a unique key within the table.

- PROCESS_COUNT

  The 10ms interval in which the statement that uses this thread is currently executing. 0 means no statement is executing, 1 means it is in the first 10ms, and so forth.

- WAIT_TYPE

  The type of wait for the thread. NULL means the thread is not blocked. Otherwise, the thread is blocked by a call to thd_wait_begin() and the value specifies the type of wait. The *xxx*_WAIT columns of the tp_thread_group_stats table accumulate counts for each wait type.

  The WAIT_TYPE value is a string that describes the type of wait, as shown in the following table.

Table 26.3 tp_thread_state Table WAIT_TYPE Values

Wait Type	Meaning
THD_WAIT_SLEEP	Waiting for sleep
THD_WAIT_DISKIO	Waiting for Disk IO
THD_WAIT_ROW_LOCK	Waiting for row lock
THD_WAIT_GLOBAL_LOCK	Waiting for global lock
THD_WAIT_META_DATA_LOCK	Waiting for metadata lock
THD_WAIT_TABLE_LOCK	Waiting for table lock
THD_WAIT_USER_LOCK	Waiting for user lock
THD_WAIT_BINLOG	Waiting for binlog

Wait Type	Meaning
THD_WAIT_GROUP_COMMIT	Waiting for group commit
THD_WAIT_SYNC	Waiting for fsync

The tp_thread_state table has these indexes:

- Unique index on (TP_GROUP_ID, TP_THREAD_NUMBER)

TRUNCATE TABLE is not permitted for the tp_thread_state table.

26.12.16 Performance Schema Clone Tables

Note

The Performance Schema tables described here are available as of MySQL 8.0.17.

The following sections describe the Performance Schema tables associated with the clone plugin (see Section 5.6.7, “The Clone Plugin”). The tables provide information about cloning operations.

- clone_status: status information about the current or last executed cloning operation.
- clone_progress: progress information about the current or last executed cloning operation.

The Performance Schema clone tables are implemented by the clone plugin and are loaded and unloaded when that plugin is loaded and unloaded (see Section 5.6.7.1, “Installing the Clone Plugin”). No special configuration step for the tables is needed. However, the tables depend on the clone plugin being enabled. If the clone plugin is loaded but disabled, the tables are not created.

The Performance Schema clone plugin tables are used only on the recipient MySQL server instance. The data is persisted across server shutdown and restart.

26.12.16.1 The clone_status Table

Note

The Performance Schema table described here is available as of MySQL 8.0.17.

The clone_status table shows the status of the current or last executed cloning operation only. The table only ever contains one row of data, or is empty.

The clone_status table has these columns:

- ID

  A unique cloning operation identifier in the current MySQL server instance.

- PID

  Process list ID of the session executing the cloning operation.

- STATE

  Current state of the cloning operation. Values include Not Started, In Progress, Completed, and Failed.

- BEGIN_TIME

A timestamp in '*YYYY-MM-DD hh:mm:ss*[.*fraction*]' format that shows when the cloning operation started.

- END_TIME

  A timestamp in '*YYYY-MM-DD hh:mm:ss*[.*fraction*]' format that shows when the cloning operation finished. Reports NULL if the operation has not ended.

- SOURCE

  The donor MySQL server address in 'HOST:PORT' format. The column displays 'LOCAL INSTANCE' for a local cloning operation.

- DESTINATION

  The directory being cloned to.

- ERROR_NO

  The error number reported for a failed cloning operation.

- ERROR_MESSAGE

  The error message string for a failed cloning operation.

- BINLOG_FILE

  The name of the binary log file up to which data is cloned.

- BINLOG_POSITION

  The binary log file offset up to which data is cloned.

- GTID_EXECUTED

  The GTID value for the last cloned transaction.

The clone_status table is read-only. DDL, including TRUNCATE TABLE, is not permitted.

26.12.16.2 The clone_progress Table

Note

The Performance Schema table described here is available as of MySQL 8.0.17.

The clone_progress table shows progress information for the current or last executed cloning operation only.

The stages of a cloning operation include DROP DATA, FILE COPY, PAGE_COPY, REDO_COPY, FILE_SYNC, RESTART, and RECOVERY. A cloning operation produces a record for each stage. The table therefore only ever contains seven rows of data, or is empty.

The clone_progress table has these columns:

- ID

  A unique cloning operation identifier in the current MySQL server instance.

- `STAGE`

  The name of the current cloning stage. Stages include `DROP DATA`, `FILE COPY`, `PAGE_COPY`, `REDO_COPY`, `FILE_SYNC`, `RESTART`, and `RECOVERY`.

- `STATE`

  The current state of the cloning stage. States include `Not Started`, `In Progress`, and `Completed`.

- `BEGIN_TIME`

  A timestamp in *`'YYYY-MM-DD hh:mm:ss[.fraction]'`* format that shows when the cloning stage started. Reports NULL if the stage has not started.

- `END_TIME`

  A timestamp in *`'YYYY-MM-DD hh:mm:ss[.fraction]'`* format that shows when the cloning stage finished. Reports NULL if the stage has not ended.

- `THREADS`

  The number of concurrent threads used in the stage.

- `ESTIMATE`

  The estimated amount of data for the current stage, in bytes.

- `DATA`

  The amount of data transferred in current state, in bytes.

- `NETWORK`

  The amount of network data transferred in the current state, in bytes.

- `DATA_SPEED`

  The current actual speed of data transfer, in bytes per second. This value may differ from the requested maximum data transfer rate defined by `clone_max_data_bandwidth`.

- `NETWORK_SPEED`

  The current speed of network transfer in bytes per second.

The `clone_progress` table is read-only. DDL, including `TRUNCATE TABLE`, is not permitted.

26.12.17 Performance Schema Summary Tables

Summary tables provide aggregated information for terminated events over time. The tables in this group summarize event data in different ways.

Wait Event Summaries

- `events_waits_summary_by_account_by_event_name`: Wait events per account and event name
- `events_waits_summary_by_host_by_event_name`: Wait events per host name and event name
- `events_waits_summary_by_instance`: Wait events per instance

- events_waits_summary_by_thread_by_event_name: Wait events per thread and event name
- events_waits_summary_by_user_by_event_name: Wait events per user name and event name
- events_waits_summary_global_by_event_name: Wait events per event name

Stage Summaries

- events_stages_summary_by_account_by_event_name: Stage events per account and event name
- events_stages_summary_by_host_by_event_name: Stage events per host name and event name
- events_stages_summary_by_thread_by_event_name: Stage waits per thread and event name
- events_stages_summary_by_user_by_event_name: Stage events per user name and event name
- events_stages_summary_global_by_event_name: Stage waits per event name

Statement Summaries

- events_statements_histogram_by_digest: Statement histograms per schema and digest value.
- events_statements_histogram_global: Statement histogram summarized globally.
- events_statements_summary_by_account_by_event_name: Statement events per account and event name
- events_statements_summary_by_digest: Statement events per schema and digest value
- events_statements_summary_by_host_by_event_name: Statement events per host name and event name
- events_statements_summary_by_program: Statement events per stored program (stored procedures and functions, triggers, and events)
- events_statements_summary_by_thread_by_event_name: Statement events per thread and event name
- events_statements_summary_by_user_by_event_name: Statement events per user name and event name
- events_statements_summary_global_by_event_name: Statement events per event name
- prepared_statements_instances: Prepared statement instances and statistics

Transaction Summaries

- events_transactions_summary_by_account_by_event_name: Transaction events per account and event name
- events_transactions_summary_by_host_by_event_name: Transaction events per host name and event name
- events_transactions_summary_by_thread_by_event_name: Transaction events per thread and event name
- events_transactions_summary_by_user_by_event_name: Transaction events per user name and event name

- `events_transactions_summary_global_by_event_name`: Transaction events per event name

Object Wait Summaries

- `objects_summary_global_by_type`: Object summaries

File I/O Summaries

- `file_summary_by_event_name`: File events per event name
- `file_summary_by_instance`: File events per file instance

Table I/O and Lock Wait Summaries

- `table_io_waits_summary_by_index_usage`: Table I/O waits per index
- `table_io_waits_summary_by_table`: Table I/O waits per table
- `table_lock_waits_summary_by_table`: Table lock waits per table

Socket Summaries

- `socket_summary_by_instance`: Socket waits and I/O per instance
- `socket_summary_by_event_name`: Socket waits and I/O per event name

Memory Summaries

- `memory_summary_by_account_by_event_name`: Memory operations per account and event name
- `memory_summary_by_host_by_event_name`: Memory operations per host and event name
- `memory_summary_by_thread_by_event_name`: Memory operations per thread and event name
- `memory_summary_by_user_by_event_name`: Memory operations per user and event name
- `memory_summary_global_by_event_name`: Memory operations globally per event name

Error Summaries

- `events_errors_summary_by_account_by_error`: Errors per error code and account
- `events_errors_summary_by_host_by_error`: Errors per error code and host
- `events_errors_summary_by_thread_by_error`: Errors per error code and thread
- `events_errors_summary_by_user_by_error`: Errors per error code and user
- `events_errors_summary_global_by_error`: Errors per error code

Status Variable Summaries

- `status_by_account`: Status variables per account
- `status_by_host`: Status variables per host name
- `status_by_user`: Status variables per user name

Each summary table has grouping columns that determine how to group the data to be aggregated, and summary columns that contain the aggregated values. Tables that summarize events in similar ways often

have similar sets of summary columns and differ only in the grouping columns used to determine how events are aggregated.

Summary tables can be truncated with `TRUNCATE TABLE`. Generally, the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. This enables you to clear collected values and restart aggregation. That might be useful, for example, after you have made a runtime configuration change. Exceptions to this truncation behavior are noted in individual summary table sections.

26.12.17.1 Wait Event Summary Tables

The Performance Schema maintains tables for collecting current and recent wait events, and aggregates that information in summary tables. Section 26.12.4, “Performance Schema Wait Event Tables” describes the events on which wait summaries are based. See that discussion for information about the content of wait events, the current and recent wait event tables, and how to control wait event collection, which is disabled by default.

Example wait event summary information:

```
mysql> SELECT *
       FROM performance_schema.events_waits_summary_global_by_event_name\G
...
*************************** 6. row ***************************
    EVENT_NAME: wait/synch/mutex/sql/BINARY_LOG::LOCK_index
    COUNT_STAR: 8
SUM_TIMER_WAIT: 2119302
MIN_TIMER_WAIT: 196092
AVG_TIMER_WAIT: 264912
MAX_TIMER_WAIT: 569421
...
*************************** 9. row ***************************
    EVENT_NAME: wait/synch/mutex/sql/hash_filo::lock
    COUNT_STAR: 69
SUM_TIMER_WAIT: 16848828
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 244185
MAX_TIMER_WAIT: 735345
...
```

Each wait event summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- `events_waits_summary_by_account_by_event_name` has `EVENT_NAME`, `USER`, and `HOST` columns. Each row summarizes events for a given account (user and host combination) and event name.
- `events_waits_summary_by_host_by_event_name` has `EVENT_NAME` and `HOST` columns. Each row summarizes events for a given host and event name.
- `events_waits_summary_by_instance` has `EVENT_NAME` and `OBJECT_INSTANCE_BEGIN` columns. Each row summarizes events for a given event name and object. If an instrument is used to create multiple instances, each instance has a unique `OBJECT_INSTANCE_BEGIN` value and is summarized separately in this table.
- `events_waits_summary_by_thread_by_event_name` has `THREAD_ID` and `EVENT_NAME` columns. Each row summarizes events for a given thread and event name.
- `events_waits_summary_by_user_by_event_name` has `EVENT_NAME` and `USER` columns. Each row summarizes events for a given user and event name.

- `events_waits_summary_global_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name. An instrument might be used to create multiple instances of the instrumented object. For example, if there is an instrument for a mutex that is created for each connection, there are as many instances as there are connections. The summary row for the instrument summarizes over all these instances.

Each wait event summary table has these summary columns containing aggregated values:

- `COUNT_STAR`

  The number of summarized events. This value includes all events, whether timed or nontimed.

- `SUM_TIMER_WAIT`

  The total wait time of the summarized timed events. This value is calculated only for timed events because nontimed events have a wait time of `NULL`. The same is true for the other *`xxx`*`_TIMER_WAIT` values.

- `MIN_TIMER_WAIT`

  The minimum wait time of the summarized timed events.

- `AVG_TIMER_WAIT`

  The average wait time of the summarized timed events.

- `MAX_TIMER_WAIT`

  The maximum wait time of the summarized timed events.

The wait event summary tables have these indexes:

- `events_waits_summary_by_account_by_event_name`:
  - Primary key on (`USER`, `HOST`, `EVENT_NAME`)
- `events_waits_summary_by_host_by_event_name`:
  - Primary key on (`HOST`, `EVENT_NAME`)
- `events_waits_summary_by_instance`:
  - Primary key on (`OBJECT_INSTANCE_BEGIN`)
  - Index on (`EVENT_NAME`)
- `events_waits_summary_by_thread_by_event_name`:
  - Primary key on (`THREAD_ID`, `EVENT_NAME`)
- `events_waits_summary_by_user_by_event_name`:
  - Primary key on (`USER`, `EVENT_NAME`)
- `events_waits_summary_global_by_event_name`:
  - Primary key on (`EVENT_NAME`)

`TRUNCATE TABLE` is permitted for wait summary tables. It has these effects:

- For summary tables not aggregated by account, host, or user, truncation resets the summary columns to zero rather than removing rows.
- For summary tables aggregated by account, host, or user, truncation removes rows for accounts, hosts, or users with no connections, and resets the summary columns to zero for the remaining rows.

In addition, each wait summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends, or truncation of `events_waits_summary_global_by_event_name`. For details, see Section 26.12.8, “Performance Schema Connection Tables”.

26.12.17.2 Stage Summary Tables

The Performance Schema maintains tables for collecting current and recent stage events, and aggregates that information in summary tables. Section 26.12.5, “Performance Schema Stage Event Tables” describes the events on which stage summaries are based. See that discussion for information about the content of stage events, the current and historical stage event tables, and how to control stage event collection, which is disabled by default.

Example stage event summary information:

```
mysql> SELECT *
       FROM performance_schema.events_stages_summary_global_by_event_name\G
...
*************************** 5. row ***************************
    EVENT_NAME: stage/sql/checking permissions
    COUNT_STAR: 57
SUM_TIMER_WAIT: 26501888880
MIN_TIMER_WAIT: 7317456
AVG_TIMER_WAIT: 464945295
MAX_TIMER_WAIT: 12858936792
...
*************************** 9. row ***************************
    EVENT_NAME: stage/sql/closing tables
    COUNT_STAR: 37
SUM_TIMER_WAIT: 662606568
MIN_TIMER_WAIT: 1593864
AVG_TIMER_WAIT: 17907891
MAX_TIMER_WAIT: 437977248
...
```

Each stage summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- `events_stages_summary_by_account_by_event_name` has `EVENT_NAME`, `USER`, and `HOST` columns. Each row summarizes events for a given account (user and host combination) and event name.
- `events_stages_summary_by_host_by_event_name` has `EVENT_NAME` and `HOST` columns. Each row summarizes events for a given host and event name.
- `events_stages_summary_by_thread_by_event_name` has `THREAD_ID` and `EVENT_NAME` columns. Each row summarizes events for a given thread and event name.
- `events_stages_summary_by_user_by_event_name` has `EVENT_NAME` and `USER` columns. Each row summarizes events for a given user and event name.
- `events_stages_summary_global_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name.

Each stage summary table has these summary columns containing aggregated values: `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, and `MAX_TIMER_WAIT`. These columns are analogous to the columns of the same names in the wait event summary tables (see Section 26.12.17.1, "Wait Event Summary Tables"), except that the stage summary tables aggregate events from `events_stages_current` rather than `events_waits_current`.

The stage summary tables have these indexes:

- `events_stages_summary_by_account_by_event_name`:
  - Primary key on (`USER`, `HOST`, `EVENT_NAME`)
- `events_stages_summary_by_host_by_event_name`:
  - Primary key on (`HOST`, `EVENT_NAME`)
- `events_stages_summary_by_thread_by_event_name`:
  - Primary key on (`THREAD_ID`, `EVENT_NAME`)
- `events_stages_summary_by_user_by_event_name`:
  - Primary key on (`USER`, `EVENT_NAME`)
- `events_stages_summary_global_by_event_name`:
  - Primary key on (`EVENT_NAME`)

`TRUNCATE TABLE` is permitted for stage summary tables. It has these effects:

- For summary tables not aggregated by account, host, or user, truncation resets the summary columns to zero rather than removing rows.
- For summary tables aggregated by account, host, or user, truncation removes rows for accounts, hosts, or users with no connections, and resets the summary columns to zero for the remaining rows.

In addition, each stage summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends, or truncation of `events_stages_summary_global_by_event_name`. For details, see Section 26.12.8, "Performance Schema Connection Tables".

26.12.17.3 Statement Summary Tables

The Performance Schema maintains tables for collecting current and recent statement events, and aggregates that information in summary tables. Section 26.12.6, "Performance Schema Statement Event Tables" describes the events on which statement summaries are based. See that discussion for information about the content of statement events, the current and historical statement event tables, and how to control statement event collection, which is partially disabled by default.

Example statement event summary information:

```
mysql> SELECT *
       FROM performance_schema.events_statements_summary_global_by_event_name\G
*************************** 1. row ***************************
                EVENT_NAME: statement/sql/select
                COUNT_STAR: 25
            SUM_TIMER_WAIT: 1535983999000
```

```
             MIN_TIMER_WAIT: 209823000
             AVG_TIMER_WAIT: 61439359000
             MAX_TIMER_WAIT: 1363397650000
              SUM_LOCK_TIME: 20186000000
                 SUM_ERRORS: 0
               SUM_WARNINGS: 0
          SUM_ROWS_AFFECTED: 0
              SUM_ROWS_SENT: 388
          SUM_ROWS_EXAMINED: 370
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 6
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 6
     SUM_NO_GOOD_INDEX_USED: 0
...
```

Each statement summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- `events_statements_summary_by_account_by_event_name` has `EVENT_NAME`, `USER`, and `HOST` columns. Each row summarizes events for a given account (user and host combination) and event name.
- `events_statements_summary_by_digest` has `SCHEMA_NAME` and `DIGEST` columns. Each row summarizes events per schema and digest value. (The `DIGEST_TEXT` column contains the corresponding normalized statement digest text, but is neither a grouping nor a summary column. The `QUERY_SAMPLE_TEXT`, `QUERY_SAMPLE_SEEN`, and `QUERY_SAMPLE_TIMER_WAIT` columns also are neither grouping nor summary columns; they support statement sampling.)

  The maximum number of rows in the table is autosized at server startup. To set this maximum explicitly, set the `performance_schema_digests_size` system variable at server startup.
- `events_statements_summary_by_host_by_event_name` has `EVENT_NAME` and `HOST` columns. Each row summarizes events for a given host and event name.
- `events_statements_summary_by_program` has `OBJECT_TYPE`, `OBJECT_SCHEMA`, and `OBJECT_NAME` columns. Each row summarizes events for a given stored program (stored procedure or function, trigger, or event).
- `events_statements_summary_by_thread_by_event_name` has `THREAD_ID` and `EVENT_NAME` columns. Each row summarizes events for a given thread and event name.
- `events_statements_summary_by_user_by_event_name` has `EVENT_NAME` and `USER` columns. Each row summarizes events for a given user and event name.
- `events_statements_summary_global_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name.
- `prepared_statements_instances` has an `OBJECT_INSTANCE_BEGIN` column. Each row summarizes events for a given prepared statement.

Each statement summary table has these summary columns containing aggregated values (with exceptions as noted):

- `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, `MAX_TIMER_WAIT`

  These columns are analogous to the columns of the same names in the wait event summary tables (see Section 26.12.17.1, “Wait Event Summary Tables”), except that the statement summary tables aggregate events from `events_statements_current` rather than `events_waits_current`.

  The `prepared_statements_instances` table does not have these columns.

- `SUM_`*`xxx`*

  The aggregate of the corresponding *`xxx`* column in the `events_statements_current` table. For example, the `SUM_LOCK_TIME` and `SUM_ERRORS` columns in statement summary tables are the aggregates of the `LOCK_TIME` and `ERRORS` columns in `events_statements_current` table.

The `events_statements_summary_by_digest` table has these additional summary columns:

- `FIRST_SEEN`, `LAST_SEEN`

  Timestamps indicating when statements with the given digest value were first seen and most recently seen.

- `QUANTILE_95`: The 95th percentile of the statement latency, in picoseconds. This percentile is a high estimate, computed from the histogram data collected. In other words, for a given digest, 95% of the statements measured have a latency lower than `QUANTILE_95`.

  For access to the histogram data, use the tables described in Section 26.12.17.4, “Statement Histogram Summary Tables”.

- `QUANTILE_99`: Similar to `QUANTILE_95`, but for the 99th percentile.

- `QUANTILE_999`: Similar to `QUANTILE_95`, but for the 99.9th percentile.

The `events_statements_summary_by_digest` table contains the following columns. These are neither grouping nor summary columns; they support statement sampling:

- `QUERY_SAMPLE_TEXT`

  A sample SQL statement that produces the digest value in the row. This column enables applications to access, for a given digest value, a statement actually seen by the server that produces that digest. One use for this might be to run `EXPLAIN` on the statement to examine the execution plan for a representative statement associated with a frequently occurring digest.

  When the `QUERY_SAMPLE_TEXT` column is assigned a value, the `QUERY_SAMPLE_SEEN` and `QUERY_SAMPLE_TIMER_WAIT` columns are assigned values as well.

  The maximum space available for statement display is 1024 bytes by default. To change this value, set the `performance_schema_max_sql_text_length` system variable at server startup. (Changing this value affects columns in other Performance Schema tables as well. See Section 26.10, “Performance Schema Statement Digests and Sampling”.)

  For information about statement sampling, see Section 26.10, “Performance Schema Statement Digests and Sampling”.

- `QUERY_SAMPLE_SEEN`

  A timestamp indicating when the statement in the `QUERY_SAMPLE_TEXT` column was seen.

- `QUERY_SAMPLE_TIMER_WAIT`

The wait time for the sample statement in the `QUERY_SAMPLE_TEXT` column.

The `events_statements_summary_by_program` table has these additional summary columns:

- `COUNT_STATEMENTS`, `SUM_STATEMENTS_WAIT`, `MIN_STATEMENTS_WAIT`, `AVG_STATEMENTS_WAIT`, `MAX_STATEMENTS_WAIT`

  Statistics about nested statements invoked during stored program execution.

The `prepared_statements_instances` table has these additional summary columns:

- `COUNT_EXECUTE`, `SUM_TIMER_EXECUTE`, `MIN_TIMER_EXECUTE`, `AVG_TIMER_EXECUTE`, `MAX_TIMER_EXECUTE`

  Aggregated statistics for executions of the prepared statement.

The statement summary tables have these indexes:

- `events_transactions_summary_by_account_by_event_name`:
  - Primary key on (`USER`, `HOST`, `EVENT_NAME`)
- `events_statements_summary_by_digest`:
  - Primary key on (`SCHEMA_NAME`, `DIGEST`)
- `events_transactions_summary_by_host_by_event_name`:
  - Primary key on (`HOST`, `EVENT_NAME`)
- `events_statements_summary_by_program`:
  - Primary key on (`OBJECT_TYPE`, `OBJECT_SCHEMA`, `OBJECT_NAME`)
- `events_statements_summary_by_thread_by_event_name`:
  - Primary key on (`THREAD_ID`, `EVENT_NAME`)
- `events_transactions_summary_by_user_by_event_name`:
  - Primary key on (`USER`, `EVENT_NAME`)
- `events_statements_summary_global_by_event_name`:
  - Primary key on (`EVENT_NAME`)

`TRUNCATE TABLE` is permitted for statement summary tables. It has these effects:

- For `events_statements_summary_by_digest`, it removes the rows.
- For other summary tables not aggregated by account, host, or user, truncation resets the summary columns to zero rather than removing rows.
- For other summary tables aggregated by account, host, or user, truncation removes rows for accounts, hosts, or users with no connections, and resets the summary columns to zero for the remaining rows.

In addition, each statement summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends, or truncation of

events_statements_summary_global_by_event_name. For details, see Section 26.12.8, “Performance Schema Connection Tables”.

In addition, truncating events_statements_summary_by_digest implicitly truncates events_statements_histogram_by_digest, and truncating events_statements_summary_global_by_event_name implicitly truncates events_statements_histogram_global.

Statement Digest Aggregation Rules

If the statements_digest consumer is enabled, aggregation into events_statements_summary_by_digest occurs as follows when a statement completes. Aggregation is based on the DIGEST value computed for the statement.

- If a events_statements_summary_by_digest row already exists with the digest value for the statement that just completed, statistics for the statement are aggregated to that row. The LAST_SEEN column is updated to the current time.
- If no row has the digest value for the statement that just completed, and the table is not full, a new row is created for the statement. The FIRST_SEEN and LAST_SEEN columns are initialized with the current time.
- If no row has the statement digest value for the statement that just completed, and the table is full, the statistics for the statement that just completed are added to a special “catch-all” row with DIGEST = NULL, which is created if necessary. If the row is created, the FIRST_SEEN and LAST_SEEN columns are initialized with the current time. Otherwise, the LAST_SEEN column is updated with the current time.

The row with DIGEST = NULL is maintained because Performance Schema tables have a maximum size due to memory constraints. The DIGEST = NULL row permits digests that do not match other rows to be counted even if the summary table is full, using a common “other” bucket. This row helps you estimate whether the digest summary is representative:

- A DIGEST = NULL row that has a COUNT_STAR value that represents 5% of all digests shows that the digest summary table is very representative; the other rows cover 95% of the statements seen.
- A DIGEST = NULL row that has a COUNT_STAR value that represents 50% of all digests shows that the digest summary table is not very representative; the other rows cover only half the statements seen. Most likely the DBA should increase the maximum table size so that more of the rows counted in the DIGEST = NULL row would be counted using more specific rows instead. To do this, set the performance_schema_digests_size system variable to a larger value at server startup. The default size is 200.

Stored Program Instrumentation Behavior

For stored program types for which instrumentation is enabled in the setup_objects table, events_statements_summary_by_program maintains statistics for stored programs as follows:

- A row is added for an object when it is first used in the server.
- The row for an object is removed when the object is dropped.
- Statistics are aggregated in the row for an object as it executes.

See also Section 26.4.3, “Event Pre-Filtering”.

26.12.17.4 Statement Histogram Summary Tables

The Performance Schema maintains statement event summary tables that contain information about minimum, maximum, and average statement latency (see Section 26.12.17.3, “Statement Summary Tables”). Those tables permit high-level assessment of system performance. To permit assessment at a more fine-grained level, the Performance Schema also collects histogram data for statement latencies. These histograms provide additional insight into latency distributions.

Section 26.12.6, “Performance Schema Statement Event Tables” describes the events on which statement summaries are based. See that discussion for information about the content of statement events, the current and historical statement event tables, and how to control statement event collection, which is partially disabled by default.

Example statement histogram information:

```
mysql> SELECT *
       FROM performance_schema.events_statements_histogram_by_digest
       WHERE SCHEMA_NAME = 'mydb' AND DIGEST = 'bb3f69453119b2d7b3ae40673a9d4c7c'
       AND COUNT_BUCKET > 0 ORDER BY BUCKET_NUMBER\G
*************************** 1. row ***************************
           SCHEMA_NAME: mydb
                DIGEST: bb3f69453119b2d7b3ae40673a9d4c7c
         BUCKET_NUMBER: 42
      BUCKET_TIMER_LOW: 66069344
     BUCKET_TIMER_HIGH: 69183097
          COUNT_BUCKET: 1
COUNT_BUCKET_AND_LOWER: 1
       BUCKET_QUANTILE: 0.058824
*************************** 2. row ***************************
           SCHEMA_NAME: mydb
                DIGEST: bb3f69453119b2d7b3ae40673a9d4c7c
         BUCKET_NUMBER: 43
      BUCKET_TIMER_LOW: 69183097
     BUCKET_TIMER_HIGH: 72443596
          COUNT_BUCKET: 1
COUNT_BUCKET_AND_LOWER: 2
       BUCKET_QUANTILE: 0.117647
*************************** 3. row ***************************
           SCHEMA_NAME: mydb
                DIGEST: bb3f69453119b2d7b3ae40673a9d4c7c
         BUCKET_NUMBER: 44
      BUCKET_TIMER_LOW: 72443596
     BUCKET_TIMER_HIGH: 75857757
          COUNT_BUCKET: 2
COUNT_BUCKET_AND_LOWER: 4
       BUCKET_QUANTILE: 0.235294
*************************** 4. row ***************************
           SCHEMA_NAME: mydb
                DIGEST: bb3f69453119b2d7b3ae40673a9d4c7c
         BUCKET_NUMBER: 45
      BUCKET_TIMER_LOW: 75857757
     BUCKET_TIMER_HIGH: 79432823
          COUNT_BUCKET: 6
COUNT_BUCKET_AND_LOWER: 10
       BUCKET_QUANTILE: 0.625000
...
```

For example, in row 3, these values indicate that 23.52% of queries run in under 75.86 microseconds:

```
BUCKET_TIMER_HIGH: 75857757
  BUCKET_QUANTILE: 0.235294
```

In row 4, these values indicate that 62.50% of queries run in under 79.44 microseconds:

```
BUCKET_TIMER_HIGH: 79432823
  BUCKET_QUANTILE: 0.625000
```

Each statement histogram summary table has one or more grouping columns to indicate how the table aggregates events:

- `events_statements_histogram_by_digest` has `SCHEMA_NAME`, `DIGEST`, and `BUCKET_NUMBER` columns:
  - The `SCHEMA_NAME` and `DIGEST` columns identify a statement digest row in the `events_statements_summary_by_digest` table.
  - The `events_statements_histogram_by_digest` rows with the same `SCHEMA_NAME` and `DIGEST` values comprise the histogram for that schema/digest combination.
  - Within a given histogram, the `BUCKET_NUMBER` column indicates the bucket number.
- `events_statements_histogram_global` has a `BUCKET_NUMBER` column. This table summarizes latencies globally across schema name and digest values, using a single histogram. The `BUCKET_NUMBER` column indicates the bucket number within this global histogram.

A histogram consists of N buckets, where each row represents one bucket, with the bucket number indicated by the `BUCKET_NUMBER` column. Bucket numbers begin with 0.

Each statement histogram summary table has these summary columns containing aggregated values:

- `BUCKET_TIMER_LOW`, `BUCKET_TIMER_HIGH`

  A bucket counts statements that have a latency, in picoseconds, measured between `BUCKET_TIMER_LOW` and `BUCKET_TIMER_HIGH`:
  - The value of `BUCKET_TIMER_LOW` for the first bucket (`BUCKET_NUMBER` = 0) is 0.
  - The value of `BUCKET_TIMER_LOW` for a bucket (`BUCKET_NUMBER` = k) is the same as `BUCKET_TIMER_HIGH` for the previous bucket (`BUCKET_NUMBER` = $k-1$)
  - The last bucket is a catchall for statements that have a latency exceeding previous buckets in the histogram.
- `COUNT_BUCKET`

  The number of statements measured with a latency in the interval from `BUCKET_TIMER_LOW` up to but not including `BUCKET_TIMER_HIGH`.
- `COUNT_BUCKET_AND_LOWER`

  The number of statements measured with a latency in the interval from 0 up to but not including `BUCKET_TIMER_HIGH`.
- `BUCKET_QUANTILE`

  The proportion of statements that fall into this or a lower bucket. This proportion corresponds by definition to `COUNT_BUCKET_AND_LOWER / SUM(COUNT_BUCKET)` and is displayed as a convenience column.

The statement histogram summary tables have these indexes:

- `events_statements_histogram_by_digest`:

  - Unique index on (SCHEMA_NAME, DIGEST, BUCKET_NUMBER)

- events_statements_histogram_global:

  - Primary key on (BUCKET_NUMBER)

TRUNCATE TABLE is permitted for statement histogram summary tables. Truncation sets the COUNT_BUCKET and COUNT_BUCKET_AND_LOWER columns to 0.

In addition, truncating events_statements_summary_by_digest implicitly truncates events_statements_histogram_by_digest, and truncating events_statements_summary_global_by_event_name implicitly truncates events_statements_histogram_global.

26.12.17.5 Transaction Summary Tables

The Performance Schema maintains tables for collecting current and recent transaction events, and aggregates that information in summary tables. Section 26.12.7, “Performance Schema Transaction Tables” describes the events on which transaction summaries are based. See that discussion for information about the content of transaction events, the current and historical transaction event tables, and how to control transaction event collection, which is disabled by default.

Example transaction event summary information:

```
mysql> SELECT *
       FROM performance_schema.events_transactions_summary_global_by_event_name
       LIMIT 1\G
*************************** 1. row ***************************
          EVENT_NAME: transaction
         COUNT_STAR: 5
     SUM_TIMER_WAIT: 19550092000
     MIN_TIMER_WAIT: 2954148000
     AVG_TIMER_WAIT: 3910018000
     MAX_TIMER_WAIT: 5486275000
    COUNT_READ_WRITE: 5
SUM_TIMER_READ_WRITE: 19550092000
MIN_TIMER_READ_WRITE: 2954148000
AVG_TIMER_READ_WRITE: 3910018000
MAX_TIMER_READ_WRITE: 5486275000
     COUNT_READ_ONLY: 0
 SUM_TIMER_READ_ONLY: 0
 MIN_TIMER_READ_ONLY: 0
 AVG_TIMER_READ_ONLY: 0
 MAX_TIMER_READ_ONLY: 0
```

Each transaction summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the setup_instruments table:

- events_transactions_summary_by_account_by_event_name has USER, HOST, and EVENT_NAME columns. Each row summarizes events for a given account (user and host combination) and event name.

- events_transactions_summary_by_host_by_event_name has HOST and EVENT_NAME columns. Each row summarizes events for a given host and event name.

- events_transactions_summary_by_thread_by_event_name has THREAD_ID and EVENT_NAME columns. Each row summarizes events for a given thread and event name.

- events_transactions_summary_by_user_by_event_name has USER and EVENT_NAME columns. Each row summarizes events for a given user and event name.

- `events_transactions_summary_global_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name.

Each transaction summary table has these summary columns containing aggregated values:

- `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, `MAX_TIMER_WAIT`

  These columns are analogous to the columns of the same names in the wait event summary tables (see Section 26.12.17.1, “Wait Event Summary Tables”), except that the transaction summary tables aggregate events from `events_transactions_current` rather than `events_waits_current`. These columns summarize read-write and read-only transactions.

- `COUNT_READ_WRITE`, `SUM_TIMER_READ_WRITE`, `MIN_TIMER_READ_WRITE`, `AVG_TIMER_READ_WRITE`, `MAX_TIMER_READ_WRITE`

  These are similar to the `COUNT_STAR` and *`xxx`*`_TIMER_WAIT` columns, but summarize read-write transactions only. The transaction access mode specifies whether transactions operate in read/write or read-only mode.

- `COUNT_READ_ONLY`, `SUM_TIMER_READ_ONLY`, `MIN_TIMER_READ_ONLY`, `AVG_TIMER_READ_ONLY`, `MAX_TIMER_READ_ONLY`

  These are similar to the `COUNT_STAR` and *`xxx`*`_TIMER_WAIT` columns, but summarize read-only transactions only. The transaction access mode specifies whether transactions operate in read/write or read-only mode.

The transaction summary tables have these indexes:

- `events_transactions_summary_by_account_by_event_name`:
  - Primary key on (`USER`, `HOST`, `EVENT_NAME`)
- `events_transactions_summary_by_host_by_event_name`:
  - Primary key on (`HOST`, `EVENT_NAME`)
- `events_transactions_summary_by_thread_by_event_name`:
  - Primary key on (`THREAD_ID`, `EVENT_NAME`)
- `events_transactions_summary_by_user_by_event_name`:
  - Primary key on (`USER`, `EVENT_NAME`)
- `events_transactions_summary_global_by_event_name`:
  - Primary key on (`EVENT_NAME`)

`TRUNCATE TABLE` is permitted for transaction summary tables. It has these effects:

- For summary tables not aggregated by account, host, or user, truncation resets the summary columns to zero rather than removing rows.
- For summary tables aggregated by account, host, or user, truncation removes rows for accounts, hosts, or users with no connections, and resets the summary columns to zero for the remaining rows.

In addition, each transaction summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends, or truncation of

`events_transactions_summary_global_by_event_name`. For details, see Section 26.12.8, “Performance Schema Connection Tables”.

Transaction Aggregation Rules

Transaction event collection occurs without regard to isolation level, access mode, or autocommit mode.

Transaction event collection occurs for all non-aborted transactions initiated by the server, including empty transactions.

Read-write transactions are generally more resource intensive than read-only transactions, therefore transaction summary tables include separate aggregate columns for read-write and read-only transactions.

Resource requirements may also vary with transaction isolation level. However, presuming that only one isolation level would be used per server, aggregation by isolation level is not provided.

26.12.17.6 Object Wait Summary Table

The Performance Schema maintains the `objects_summary_global_by_type` table for aggregating object wait events.

Example object wait event summary information:

```
mysql> SELECT * FROM performance_schema.objects_summary_global_by_type\G
...
*************************** 3. row ***************************
   OBJECT_TYPE: TABLE
 OBJECT_SCHEMA: test
   OBJECT_NAME: t
    COUNT_STAR: 3
SUM_TIMER_WAIT: 263126976
MIN_TIMER_WAIT: 1522272
AVG_TIMER_WAIT: 87708678
MAX_TIMER_WAIT: 258428280
...
*************************** 10. row ***************************
   OBJECT_TYPE: TABLE
 OBJECT_SCHEMA: mysql
   OBJECT_NAME: user
    COUNT_STAR: 14
SUM_TIMER_WAIT: 365567592
MIN_TIMER_WAIT: 1141704
AVG_TIMER_WAIT: 26111769
MAX_TIMER_WAIT: 334783032
...
```

The `objects_summary_global_by_type` table has these grouping columns to indicate how the table aggregates events: `OBJECT_TYPE`, `OBJECT_SCHEMA`, and `OBJECT_NAME`. Each row summarizes events for the given object.

`objects_summary_global_by_type` has the same summary columns as the `events_waits_summary_by_xxx` tables. See Section 26.12.17.1, “Wait Event Summary Tables”.

The `objects_summary_global_by_type` table has these indexes:

- Primary key on (`OBJECT_TYPE`, `OBJECT_SCHEMA`, `OBJECT_NAME`)

`TRUNCATE TABLE` is permitted for the object summary table. It resets the summary columns to zero rather than removing rows.

26.12.17.7 File I/O Summary Tables

The Performance Schema maintains file I/O summary tables that aggregate information about I/O operations.

Example file I/O event summary information:

```
mysql> SELECT * FROM performance_schema.file_summary_by_event_name\G
...
*************************** 2. row ***************************
              EVENT_NAME: wait/io/file/sql/binlog
              COUNT_STAR: 31
          SUM_TIMER_WAIT: 8243784888
          MIN_TIMER_WAIT: 0
          AVG_TIMER_WAIT: 265928484
          MAX_TIMER_WAIT: 6490658832
...
mysql> SELECT * FROM performance_schema.file_summary_by_instance\G
...
*************************** 2. row ***************************
               FILE_NAME: /var/mysql/share/english/errmsg.sys
              EVENT_NAME: wait/io/file/sql/ERRMSG
              EVENT_NAME: wait/io/file/sql/ERRMSG
    OBJECT_INSTANCE_BEGIN: 4686193384
              COUNT_STAR: 5
          SUM_TIMER_WAIT: 13990154448
          MIN_TIMER_WAIT: 26349624
          AVG_TIMER_WAIT: 2798030607
          MAX_TIMER_WAIT: 8150662536
...
```

Each file I/O summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- `file_summary_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name.
- `file_summary_by_instance` has `FILE_NAME`, `EVENT_NAME`, and `OBJECT_INSTANCE_BEGIN` columns. Each row summarizes events for a given file and event name.

Each file I/O summary table has the following summary columns containing aggregated values. Some columns are more general and have values that are the same as the sum of the values of more fine-grained columns. In this way, aggregations at higher levels are available directly without the need for user-defined views that sum lower-level columns.

- `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, `MAX_TIMER_WAIT`

  These columns aggregate all I/O operations.

- `COUNT_READ`, `SUM_TIMER_READ`, `MIN_TIMER_READ`, `AVG_TIMER_READ`, `MAX_TIMER_READ`, `SUM_NUMBER_OF_BYTES_READ`

  These columns aggregate all read operations, including `FGETS`, `FGETC`, `FREAD`, and `READ`.

- `COUNT_WRITE`, `SUM_TIMER_WRITE`, `MIN_TIMER_WRITE`, `AVG_TIMER_WRITE`, `MAX_TIMER_WRITE`, `SUM_NUMBER_OF_BYTES_WRITE`

  These columns aggregate all write operations, including `FPUTS`, `FPUTC`, `FPRINTF`, `VFPRINTF`, `FWRITE`, and `PWRITE`.

- `COUNT_MISC`, `SUM_TIMER_MISC`, `MIN_TIMER_MISC`, `AVG_TIMER_MISC`, `MAX_TIMER_MISC`

These columns aggregate all other I/O operations, including `CREATE`, `DELETE`, `OPEN`, `CLOSE`, `STREAM_OPEN`, `STREAM_CLOSE`, `SEEK`, `TELL`, `FLUSH`, `STAT`, `FSTAT`, `CHSIZE`, `RENAME`, and `SYNC`. There are no byte counts for these operations.

The file I/O summary tables have these indexes:

- `file_summary_by_event_name`:
  - Primary key on (`EVENT_NAME`)
- `file_summary_by_instance`:
  - Primary key on (`OBJECT_INSTANCE_BEGIN`)
  - Index on (`FILE_NAME`)
  - Index on (`EVENT_NAME`)

`TRUNCATE TABLE` is permitted for file I/O summary tables. It resets the summary columns to zero rather than removing rows.

The MySQL server uses several techniques to avoid I/O operations by caching information read from files, so it is possible that statements you might expect to result in I/O events will not. You may be able to ensure that I/O does occur by flushing caches or restarting the server to reset its state.

26.12.17.8 Table I/O and Lock Wait Summary Tables

The following sections describe the table I/O and lock wait summary tables:

- `table_io_waits_summary_by_index_usage`: Table I/O waits per index
- `table_io_waits_summary_by_table`: Table I/O waits per table
- `table_lock_waits_summary_by_table`: Table lock waits per table

The table_io_waits_summary_by_table Table

The `table_io_waits_summary_by_table` table aggregates all table I/O wait events, as generated by the `wait/io/table/sql/handler` instrument. The grouping is by table.

The `table_io_waits_summary_by_table` table has these grouping columns to indicate how the table aggregates events: `OBJECT_TYPE`, `OBJECT_SCHEMA`, and `OBJECT_NAME`. These columns have the same meaning as in the `events_waits_current` table. They identify the table to which the row applies.

`table_io_waits_summary_by_table` has the following summary columns containing aggregated values. As indicated in the column descriptions, some columns are more general and have values that are the same as the sum of the values of more fine-grained columns. For example, columns that aggregate all writes hold the sum of the corresponding columns that aggregate inserts, updates, and deletes. In this way, aggregations at higher levels are available directly without the need for user-defined views that sum lower-level columns.

- `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, `MAX_TIMER_WAIT`

  These columns aggregate all I/O operations. They are the same as the sum of the corresponding *`xxx`*`_READ` and *`xxx`*`_WRITE` columns.

- `COUNT_READ`, `SUM_TIMER_READ`, `MIN_TIMER_READ`, `AVG_TIMER_READ`, `MAX_TIMER_READ`

These columns aggregate all read operations. They are the same as the sum of the corresponding *xxx*_FETCH columns.

- COUNT_WRITE, SUM_TIMER_WRITE, MIN_TIMER_WRITE, AVG_TIMER_WRITE, MAX_TIMER_WRITE

  These columns aggregate all write operations. They are the same as the sum of the corresponding *xxx*_INSERT, *xxx*_UPDATE, and *xxx*_DELETE columns.

- COUNT_FETCH, SUM_TIMER_FETCH, MIN_TIMER_FETCH, AVG_TIMER_FETCH, MAX_TIMER_FETCH

  These columns aggregate all fetch operations.

- COUNT_INSERT, SUM_TIMER_INSERT, MIN_TIMER_INSERT, AVG_TIMER_INSERT, MAX_TIMER_INSERT

  These columns aggregate all insert operations.

- COUNT_UPDATE, SUM_TIMER_UPDATE, MIN_TIMER_UPDATE, AVG_TIMER_UPDATE, MAX_TIMER_UPDATE

  These columns aggregate all update operations.

- COUNT_DELETE, SUM_TIMER_DELETE, MIN_TIMER_DELETE, AVG_TIMER_DELETE, MAX_TIMER_DELETE

  These columns aggregate all delete operations.

The table_io_waits_summary_by_table table has these indexes:

- Unique index on (OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME)

TRUNCATE TABLE is permitted for table I/O summary tables. It resets the summary columns to zero rather than removing rows. Truncating this table also truncates the table_io_waits_summary_by_index_usage table.

The table_io_waits_summary_by_index_usage Table

The table_io_waits_summary_by_index_usage table aggregates all table index I/O wait events, as generated by the wait/io/table/sql/handler instrument. The grouping is by table index.

The columns of table_io_waits_summary_by_index_usage are nearly identical to table_io_waits_summary_by_table. The only difference is the additional group column, INDEX_NAME, which corresponds to the name of the index that was used when the table I/O wait event was recorded:

- A value of PRIMARY indicates that table I/O used the primary index.
- A value of NULL means that table I/O used no index.
- Inserts are counted against INDEX_NAME = NULL.

The table_io_waits_summary_by_index_usage table has these indexes:

- Unique index on (OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME, INDEX_NAME)

TRUNCATE TABLE is permitted for table I/O summary tables. It resets the summary columns to zero rather than removing rows. This table is also truncated by truncation of the

table_io_waits_summary_by_table table. A DDL operation that changes the index structure of a table may cause the per-index statistics to be reset.

The table_lock_waits_summary_by_table Table

The table_lock_waits_summary_by_table table aggregates all table lock wait events, as generated by the wait/lock/table/sql/handler instrument. The grouping is by table.

This table contains information about internal and external locks:

- An internal lock corresponds to a lock in the SQL layer. This is currently implemented by a call to thr_lock(). In event rows, these locks are distinguished by the OPERATION column, which has one of these values:

```
read normal
read with shared locks
read high priority
read no insert
write allow write
write concurrent insert
write delayed
write low priority
write normal
```

- An external lock corresponds to a lock in the storage engine layer. This is currently implemented by a call to handler::external_lock(). In event rows, these locks are distinguished by the OPERATION column, which has one of these values:

```
read external
write external
```

The table_lock_waits_summary_by_table table has these grouping columns to indicate how the table aggregates events: OBJECT_TYPE, OBJECT_SCHEMA, and OBJECT_NAME. These columns have the same meaning as in the events_waits_current table. They identify the table to which the row applies.

table_lock_waits_summary_by_table has the following summary columns containing aggregated values. As indicated in the column descriptions, some columns are more general and have values that are the same as the sum of the values of more fine-grained columns. For example, columns that aggregate all locks hold the sum of the corresponding columns that aggregate read and write locks. In this way, aggregations at higher levels are available directly without the need for user-defined views that sum lower-level columns.

- COUNT_STAR, SUM_TIMER_WAIT, MIN_TIMER_WAIT, AVG_TIMER_WAIT, MAX_TIMER_WAIT

  These columns aggregate all lock operations. They are the same as the sum of the corresponding *xxx*_READ and *xxx*_WRITE columns.

- COUNT_READ, SUM_TIMER_READ, MIN_TIMER_READ, AVG_TIMER_READ, MAX_TIMER_READ

  These columns aggregate all read-lock operations. They are the same as the sum of the corresponding *xxx*_READ_NORMAL, *xxx*_READ_WITH_SHARED_LOCKS, *xxx*_READ_HIGH_PRIORITY, and *xxx*_READ_NO_INSERT columns.

- COUNT_WRITE, SUM_TIMER_WRITE, MIN_TIMER_WRITE, AVG_TIMER_WRITE, MAX_TIMER_WRITE

  These columns aggregate all write-lock operations. They are the same as the sum of the corresponding *xxx*_WRITE_ALLOW_WRITE, *xxx*_WRITE_CONCURRENT_INSERT, *xxx*_WRITE_LOW_PRIORITY, and *xxx*_WRITE_NORMAL columns.

- `COUNT_READ_NORMAL`, `SUM_TIMER_READ_NORMAL`, `MIN_TIMER_READ_NORMAL`, `AVG_TIMER_READ_NORMAL`, `MAX_TIMER_READ_NORMAL`

  These columns aggregate internal read locks.

- `COUNT_READ_WITH_SHARED_LOCKS`, `SUM_TIMER_READ_WITH_SHARED_LOCKS`, `MIN_TIMER_READ_WITH_SHARED_LOCKS`, `AVG_TIMER_READ_WITH_SHARED_LOCKS`, `MAX_TIMER_READ_WITH_SHARED_LOCKS`

  These columns aggregate internal read locks.

- `COUNT_READ_HIGH_PRIORITY`, `SUM_TIMER_READ_HIGH_PRIORITY`, `MIN_TIMER_READ_HIGH_PRIORITY`, `AVG_TIMER_READ_HIGH_PRIORITY`, `MAX_TIMER_READ_HIGH_PRIORITY`

  These columns aggregate internal read locks.

- `COUNT_READ_NO_INSERT`, `SUM_TIMER_READ_NO_INSERT`, `MIN_TIMER_READ_NO_INSERT`, `AVG_TIMER_READ_NO_INSERT`, `MAX_TIMER_READ_NO_INSERT`

  These columns aggregate internal read locks.

- `COUNT_READ_EXTERNAL`, `SUM_TIMER_READ_EXTERNAL`, `MIN_TIMER_READ_EXTERNAL`, `AVG_TIMER_READ_EXTERNAL`, `MAX_TIMER_READ_EXTERNAL`

  These columns aggregate external read locks.

- `COUNT_WRITE_ALLOW_WRITE`, `SUM_TIMER_WRITE_ALLOW_WRITE`, `MIN_TIMER_WRITE_ALLOW_WRITE`, `AVG_TIMER_WRITE_ALLOW_WRITE`, `MAX_TIMER_WRITE_ALLOW_WRITE`

  These columns aggregate internal write locks.

- `COUNT_WRITE_CONCURRENT_INSERT`, `SUM_TIMER_WRITE_CONCURRENT_INSERT`, `MIN_TIMER_WRITE_CONCURRENT_INSERT`, `AVG_TIMER_WRITE_CONCURRENT_INSERT`, `MAX_TIMER_WRITE_CONCURRENT_INSERT`

  These columns aggregate internal write locks.

- `COUNT_WRITE_LOW_PRIORITY`, `SUM_TIMER_WRITE_LOW_PRIORITY`, `MIN_TIMER_WRITE_LOW_PRIORITY`, `AVG_TIMER_WRITE_LOW_PRIORITY`, `MAX_TIMER_WRITE_LOW_PRIORITY`

  These columns aggregate internal write locks.

- `COUNT_WRITE_NORMAL`, `SUM_TIMER_WRITE_NORMAL`, `MIN_TIMER_WRITE_NORMAL`, `AVG_TIMER_WRITE_NORMAL`, `MAX_TIMER_WRITE_NORMAL`

  These columns aggregate internal write locks.

- `COUNT_WRITE_EXTERNAL`, `SUM_TIMER_WRITE_EXTERNAL`, `MIN_TIMER_WRITE_EXTERNAL`, `AVG_TIMER_WRITE_EXTERNAL`, `MAX_TIMER_WRITE_EXTERNAL`

  These columns aggregate external write locks.

The `table_lock_waits_summary_by_table` table has these indexes:

- Unique index on (`OBJECT_TYPE`, `OBJECT_SCHEMA`, `OBJECT_NAME`)

`TRUNCATE TABLE` is permitted for table lock summary tables. It resets the summary columns to zero rather than removing rows.

26.12.17.9 Socket Summary Tables

These socket summary tables aggregate timer and byte count information for socket operations:

- `socket_summary_by_event_name`: Aggregate timer and byte count statistics generated by the `wait/io/socket/*` instruments for all socket I/O operations, per socket instrument.
- `socket_summary_by_instance`: Aggregate timer and byte count statistics generated by the `wait/io/socket/*` instruments for all socket I/O operations, per socket instance. When a connection terminates, the row in `socket_summary_by_instance` corresponding to it is deleted.

The socket summary tables do not aggregate waits generated by `idle` events while sockets are waiting for the next request from the client. For `idle` event aggregations, use the wait-event summary tables; see Section 26.12.17.1, “Wait Event Summary Tables”.

Each socket summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- `socket_summary_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name.
- `socket_summary_by_instance` has an `OBJECT_INSTANCE_BEGIN` column. Each row summarizes events for a given object.

Each socket summary table has these summary columns containing aggregated values:

- `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, `MAX_TIMER_WAIT`

  These columns aggregate all operations.

- `COUNT_READ`, `SUM_TIMER_READ`, `MIN_TIMER_READ`, `AVG_TIMER_READ`, `MAX_TIMER_READ`, `SUM_NUMBER_OF_BYTES_READ`

  These columns aggregate all receive operations (`RECV`, `RECVFROM`, and `RECVMSG`).

- `COUNT_WRITE`, `SUM_TIMER_WRITE`, `MIN_TIMER_WRITE`, `AVG_TIMER_WRITE`, `MAX_TIMER_WRITE`, `SUM_NUMBER_OF_BYTES_WRITE`

  These columns aggregate all send operations (`SEND`, `SENDTO`, and `SENDMSG`).

- `COUNT_MISC`, `SUM_TIMER_MISC`, `MIN_TIMER_MISC`, `AVG_TIMER_MISC`, `MAX_TIMER_MISC`

  These columns aggregate all other socket operations, such as `CONNECT`, `LISTEN`, `ACCEPT`, `CLOSE`, and `SHUTDOWN`. There are no byte counts for these operations.

The `socket_summary_by_instance` table also has an `EVENT_NAME` column that indicates the class of the socket: `client_connection`, `server_tcpip_socket`, `server_unix_socket`. This column can be grouped on to isolate, for example, client activity from that of the server listening sockets.

The socket summary tables have these indexes:

- `socket_summary_by_event_name`:
  - Primary key on (`EVENT_NAME`)

- `socket_summary_by_instance`:
  - Primary key on (`OBJECT_INSTANCE_BEGIN`)
  - Index on (`EVENT_NAME`)

`TRUNCATE TABLE` is permitted for socket summary tables. Except for `events_statements_summary_by_digest`, tt resets the summary columns to zero rather than removing rows.

26.12.17.10 Memory Summary Tables

The Performance Schema instruments memory usage and aggregates memory usage statistics, detailed by these factors:

- Type of memory used (various caches, internal buffers, and so forth)
- Thread, account, user, host indirectly performing the memory operation

The Performance Schema instruments the following aspects of memory use

- Memory sizes used
- Operation counts
- Low and high water marks

Memory sizes help to understand or tune the memory consumption of the server.

Operation counts help to understand or tune the overall pressure the server is putting on the memory allocator, which has an impact on performance. Allocating a single byte one million times is not the same as allocating one million bytes a single time; tracking both sizes and counts can expose the difference.

Low and high water marks are critical to detect workload spikes, overall workload stability, and possible memory leaks.

Memory summary tables do not contain timing information because memory events are not timed.

For information about collecting memory usage data, see Memory Instrumentation Behavior.

Example memory event summary information:

```
mysql> SELECT *
       FROM performance_schema.memory_summary_global_by_event_name
       WHERE EVENT_NAME = 'memory/sql/TABLE'\G
*************************** 1. row ***************************
                 EVENT_NAME: memory/sql/TABLE
                COUNT_ALLOC: 1381
                 COUNT_FREE: 924
   SUM_NUMBER_OF_BYTES_ALLOC: 2059873
    SUM_NUMBER_OF_BYTES_FREE: 1407432
              LOW_COUNT_USED: 0
          CURRENT_COUNT_USED: 457
             HIGH_COUNT_USED: 461
    LOW_NUMBER_OF_BYTES_USED: 0
CURRENT_NUMBER_OF_BYTES_USED: 652441
   HIGH_NUMBER_OF_BYTES_USED: 669269
```

Each memory summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- `memory_summary_by_account_by_event_name` has `USER`, `HOST`, and `EVENT_NAME` columns. Each row summarizes events for a given account (user and host combination) and event name.
- `memory_summary_by_host_by_event_name` has `HOST` and `EVENT_NAME` columns. Each row summarizes events for a given host and event name.
- `memory_summary_by_thread_by_event_name` has `THREAD_ID` and `EVENT_NAME` columns. Each row summarizes events for a given thread and event name.
- `memory_summary_by_user_by_event_name` has `USER` and `EVENT_NAME` columns. Each row summarizes events for a given user and event name.
- `memory_summary_global_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name.

Each memory summary table has these summary columns containing aggregated values:

- `COUNT_ALLOC`, `COUNT_FREE`

  The aggregated numbers of calls to memory-allocation and memory-free functions.

- `SUM_NUMBER_OF_BYTES_ALLOC`, `SUM_NUMBER_OF_BYTES_FREE`

  The aggregated sizes of allocated and freed memory blocks.

- `CURRENT_COUNT_USED`

  The aggregated number of currently allocated blocks that have not been freed yet. This is a convenience column, equal to `COUNT_ALLOC` − `COUNT_FREE`.

- `CURRENT_NUMBER_OF_BYTES_USED`

  The aggregated size of currently allocated memory blocks that have not been freed yet. This is a convenience column, equal to `SUM_NUMBER_OF_BYTES_ALLOC` − `SUM_NUMBER_OF_BYTES_FREE`.

- `LOW_COUNT_USED`, `HIGH_COUNT_USED`

  The low and high water marks corresponding to the `CURRENT_COUNT_USED` column.

- `LOW_NUMBER_OF_BYTES_USED`, `HIGH_NUMBER_OF_BYTES_USED`

  The low and high water marks corresponding to the `CURRENT_NUMBER_OF_BYTES_USED` column.

The memory summary tables have these indexes:

- `memory_summary_by_account_by_event_name`:
  - Primary key on (`USER`, `HOST`, `EVENT_NAME`)
- `memory_summary_by_host_by_event_name`:
  - Primary key on (`HOST`, `EVENT_NAME`)
- `memory_summary_by_thread_by_event_name`:
  - Primary key on (`THREAD_ID`, `EVENT_NAME`)
- `memory_summary_by_user_by_event_name`:

  - Primary key on (USER, EVENT_NAME)

- memory_summary_global_by_event_name:

  - Primary key on (EVENT_NAME)

TRUNCATE TABLE is permitted for memory summary tables. It has these effects:

- In general, truncation resets the baseline for statistics, but does not change the server state. That is, truncating a memory table does not free memory.
- COUNT_ALLOC and COUNT_FREE are reset to a new baseline, by reducing each counter by the same value.
- Likewise, SUM_NUMBER_OF_BYTES_ALLOC and SUM_NUMBER_OF_BYTES_FREE are reset to a new baseline.
- LOW_COUNT_USED and HIGH_COUNT_USED are reset to CURRENT_COUNT_USED.
- LOW_NUMBER_OF_BYTES_USED and HIGH_NUMBER_OF_BYTES_USED are reset to CURRENT_NUMBER_OF_BYTES_USED.

In addition, each memory summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends, or truncation of memory_summary_global_by_event_name. For details, see Section 26.12.8, “Performance Schema Connection Tables”.

Memory Instrumentation Behavior

Memory instruments are listed in the setup_instruments table and have names of the form memory/*code_area*/*instrument_name*. Memory instrumentation is enabled by default.

Instruments named with the prefix memory/performance_schema/ expose how much memory is allocated for internal buffers in the Performance Schema itself. The memory/performance_schema/ instruments are built in, always enabled, and cannot be disabled at startup or runtime. Built-in memory instruments are displayed only in the memory_summary_global_by_event_name table.

To control memory instrumentation state at server startup, use lines like these in your my.cnf file:

- Enable:

```
[mysqld]
performance-schema-instrument='memory/%=ON'
```

- Disable:

```
[mysqld]
performance-schema-instrument='memory/%=OFF'
```

To control memory instrumentation state at runtime, update the ENABLED column of the relevant instruments in the setup_instruments table:

- Enable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES'
WHERE NAME LIKE 'memory/%';
```

- Disable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME LIKE 'memory/%';
```

For memory instruments, the `TIMED` column in `setup_instruments` is ignored because memory operations are not timed.

When a thread in the server executes a memory allocation that has been instrumented, these rules apply:

- If the thread is not instrumented or the memory instrument is not enabled, the memory block allocated is not instrumented.
- Otherwise (that is, both the thread and the instrument are enabled), the memory block allocated is instrumented.

For deallocation, these rules apply:

- If a memory allocation operation was instrumented, the corresponding free operation is instrumented, regardless of the current instrument or thread enabled status.
- If a memory allocation operation was not instrumented, the corresponding free operation is not instrumented, regardless of the current instrument or thread enabled status.

For the per-thread statistics, the following rules apply.

When an instrumented memory block of size *`N`* is allocated, the Performance Schema makes these updates to memory summary table columns:

- `COUNT_ALLOC`: Increased by 1
- `CURRENT_COUNT_USED`: Increased by 1
- `HIGH_COUNT_USED`: Increased if `CURRENT_COUNT_USED` is a new maximum
- `SUM_NUMBER_OF_BYTES_ALLOC`: Increased by *`N`*
- `CURRENT_NUMBER_OF_BYTES_USED`: Increased by *`N`*
- `HIGH_NUMBER_OF_BYTES_USED`: Increased if `CURRENT_NUMBER_OF_BYTES_USED` is a new maximum

When an instrumented memory block is deallocated, the Performance Schema makes these updates to memory summary table columns:

- `COUNT_FREE`: Increased by 1
- `CURRENT_COUNT_USED`: Decreased by 1
- `LOW_COUNT_USED`: Decreased if `CURRENT_COUNT_USED` is a new minimum
- `SUM_NUMBER_OF_BYTES_FREE`: Increased by *`N`*
- `CURRENT_NUMBER_OF_BYTES_USED`: Decreased by *`N`*
- `LOW_NUMBER_OF_BYTES_USED`: Decreased if `CURRENT_NUMBER_OF_BYTES_USED` is a new minimum

For higher-level aggregates (global, by account, by user, by host), the same rules apply as expected for low and high water marks.

- `LOW_COUNT_USED` and `LOW_NUMBER_OF_BYTES_USED` are lower estimates. The value reported by the Performance Schema is guaranteed to be less than or equal to the lowest count or size of memory effectively used at runtime.
- `HIGH_COUNT_USED` and `HIGH_NUMBER_OF_BYTES_USED` are higher estimates. The value reported by the Performance Schema is guaranteed to be greater than or equal to the highest count or size of memory effectively used at runtime.

For lower estimates in summary tables other than `memory_summary_global_by_event_name`, it is possible for values to go negative if memory ownership is transferred between threads.

Here is an example of estimate computation; but note that estimate implementation is subject to change:

Thread 1 uses memory in the range from 1MB to 2MB during execution, as reported by the `LOW_NUMBER_OF_BYTES_USED` and `HIGH_NUMBER_OF_BYTES_USED` columns of the `memory_summary_by_thread_by_event_name` table.

Thread 2 uses memory in the range from 10MB to 12MB during execution, as reported likewise.

When these two threads belong to the same user account, the per-account summary estimates that this account used memory in the range from 11MB to 14MB. That is, the `LOW_NUMBER_OF_BYTES_USED` for the higher level aggregate is the sum of each `LOW_NUMBER_OF_BYTES_USED` (assuming the worst case). Likewise, the `HIGH_NUMBER_OF_BYTES_USED` for the higher level aggregate is the sum of each `HIGH_NUMBER_OF_BYTES_USED` (assuming the worst case).

11MB is a lower estimate that can occur only if both threads hit the low usage mark at the same time.

14MB is a higher estimate that can occur only if both threads hit the high usage mark at the same time.

The real memory usage for this account could have been in the range from 11.5MB to 13.5MB.

For capacity planning, reporting the worst case is actually the desired behavior, as it shows what can potentially happen when sessions are uncorrelated, which is typically the case.

26.12.17.11 Error Summary Tables

The Performance Schema maintains summary tables for aggregating statistical information about server errors (and warnings). For a list of server errors, see Section B.3.1, “Server Error Message Reference”.

Collection of error information is controlled by the `error` instrument, which is enabled by default. Timing information is not collected.

Each error summary table has three columns that identify the error:

- `ERROR_NUMBER` is the numeric error value. The value is unique.
- `ERROR_NAME` is the symbolic error name corresponding to the `ERROR_NUMBER` value. The value is unique.
- `SQLSTATE` is the SQLSTATE value corresponding to the `ERROR_NUMBER` value. The value is not necessarily unique.

For example, if `ERROR_NUMBER` is 1050, `ERROR_NAME` is `ER_TABLE_EXISTS_ERROR` and `SQLSTATE` is `42S01`.

Example error event summary information:

```
mysql> SELECT *
```

```
       FROM performance_schema.events_errors_summary_global_by_error
       WHERE SUM_ERROR_RAISED <> 0\G
*************************** 1. row ***************************
     ERROR_NUMBER: 1064
       ERROR_NAME: ER_PARSE_ERROR
        SQL_STATE: 42000
 SUM_ERROR_RAISED: 1
SUM_ERROR_HANDLED: 0
       FIRST_SEEN: 2016-06-28 07:34:02
        LAST_SEEN: 2016-06-28 07:34:02
*************************** 2. row ***************************
     ERROR_NUMBER: 1146
       ERROR_NAME: ER_NO_SUCH_TABLE
        SQL_STATE: 42S02
 SUM_ERROR_RAISED: 2
SUM_ERROR_HANDLED: 0
       FIRST_SEEN: 2016-06-28 07:34:05
        LAST_SEEN: 2016-06-28 07:36:18
*************************** 3. row ***************************
     ERROR_NUMBER: 1317
       ERROR_NAME: ER_QUERY_INTERRUPTED
        SQL_STATE: 70100
 SUM_ERROR_RAISED: 1
SUM_ERROR_HANDLED: 0
       FIRST_SEEN: 2016-06-28 11:01:49
        LAST_SEEN: 2016-06-28 11:01:49
```

Each error summary table has one or more grouping columns to indicate how the table aggregates errors:

- `events_errors_summary_by_account_by_error` has `USER`, `HOST`, and `ERROR_NUMBER` columns. Each row summarizes events for a given account (user and host combination) and error.
- `events_errors_summary_by_host_by_error` has `HOST` and `ERROR_NUMBER` columns. Each row summarizes events for a given host and error.
- `events_errors_summary_by_thread_by_error` has `THREAD_ID` and `ERROR_NUMBER` columns. Each row summarizes events for a given thread and error.
- `events_errors_summary_by_user_by_error` has `USER` and `ERROR_NUMBER` columns. Each row summarizes events for a given user and error.
- `events_errors_summary_global_by_error` has an `ERROR_NUMBER` column. Each row summarizes events for a given error.

Each error summary table has these summary columns containing aggregated values:

- `SUM_ERROR_RAISED`

  This column aggregates the number of times the error occurred.

- `SUM_ERROR_HANDLED`

  This column aggregates the number of times the error was handled by an SQL exception handler.

- `FIRST_SEEN`, `LAST_SEEN`

  Timestamp indicating when the error was first seen and most recently seen.

A `NULL` row in each error summary table is used to aggregate statistics for all errors that lie out of range of the instrumented errors. For example, if MySQL Server errors lie in the range from *`M`* to *`N`* and an error is raised with number *`Q`* not in that range, the error is aggregated in the `NULL` row. The `NULL` row is the row with `ERROR_NUMBER=0`, `ERROR_NAME=NULL`, and `SQLSTATE=NULL`.

The error summary tables have these indexes:

- `events_errors_summary_by_account_by_error`:
  - Primary key on (`USER`, `HOST`, `ERROR_NUMBER`)
- `events_errors_summary_by_host_by_error`:
  - Primary key on (`HOST`, `ERROR_NUMBER`)
- `events_errors_summary_by_thread_by_error`:
  - Primary key on (`THREAD_ID`, `ERROR_NUMBER`)
- `events_errors_summary_by_user_by_error`:
  - Primary key on (`USER`, `ERROR_NUMBER`)
- `events_errors_summary_global_by_error`:
  - Primary key on (`ERROR_NUMBER`)

`TRUNCATE TABLE` is permitted for error summary tables. It has these effects:

- For summary tables not aggregated by account, host, or user, truncation resets the summary columns to zero or `NULL` rather than removing rows.
- For summary tables aggregated by account, host, or user, truncation removes rows for accounts, hosts, or users with no connections, and resets the summary columns to zero or `NULL` for the remaining rows.

In addition, each error summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends, or truncation of `events_errors_summary_global_by_error`. For details, see Section 26.12.8, “Performance Schema Connection Tables”.

26.12.17.12 Status Variable Summary Tables

The Performance Schema makes status variable information available in the tables described in Section 26.12.14, “Performance Schema Status Variable Tables”. It also makes aggregated status variable information available in summary tables, described here. Each status variable summary table has one or more grouping columns to indicate how the table aggregates status values:

- `status_by_account` has `USER`, `HOST`, and `VARIABLE_NAME` columns to summarize status variables by account.
- `status_by_host` has `HOST` and `VARIABLE_NAME` columns to summarize status variables by the host from which clients connected.
- `status_by_user` has `USER` and `VARIABLE_NAME` columns to summarize status variables by client user name.

Each status variable summary table has this summary column containing aggregated values:

- `VARIABLE_VALUE`

  The aggregated status variable value for active and terminated sessions.

The status variable summary tables have these indexes:

- `status_by_account`:
  - Primary key on (`USER`, `HOST`, `VARIABLE_NAME`)
- `status_by_host`:
  - Primary key on (`HOST`, `VARIABLE_NAME`)
- `status_by_user`:
  - Primary key on (`USER`, `VARIABLE_NAME`)

The meaning of “account” in these tables is similar to its meaning in the MySQL grant tables in the `mysql` system database, in the sense that the term refers to a combination of user and host values. They differ in that, for grant tables, the host part of an account can be a pattern, whereas for Performance Schema tables, the host value is always a specific nonpattern host name.

Account status is collected when sessions terminate. The session status counters are added to the global status counters and the corresponding account status counters. If account statistics are not collected, the session status is added to host and user status, if host and user status are collected.

Account, host, and user statistics are not collected if the `performance_schema_accounts_size`, `performance_schema_hosts_size`, and `performance_schema_users_size` system variables, respectively, are set to 0.

The Performance Schema supports `TRUNCATE TABLE` for status variable summary tables as follows; in all cases, status for active sessions is unaffected:

- `status_by_account`: Aggregates account status from terminated sessions to user and host status, then resets account status.
- `status_by_host`: Resets aggregated host status from terminated sessions.
- `status_by_user`: Resets aggregated user status from terminated sessions.

`FLUSH STATUS` adds the session status from all active sessions to the global status variables, resets the status of all active sessions, and resets account, host, and user status values aggregated from disconnected sessions.

26.12.18 Performance Schema Miscellaneous Tables

The following sections describe tables that do not fall into the table categories discussed in the preceding sections:

- `host_cache`: Information from the internal host cache
- `performance_timers`: Which event timers are available
- `threads`: Information about server threads
- `user_defined_functions`: User-defined functions registered by a server component, plugin, or `CREATE FUNCTION` statement
- `log_status`: Information about server logs for backup purposes

26.12.18.1 The host_cache Table

The `host_cache` table provides access to the contents of the host cache, which contains client host name and IP address information and is used to avoid Domain Name System (DNS) lookups. The

`host_cache_size` system variable controls the size of the host cache, as well as the size of the `host_cache` table that exposes the cache contents. For operational and configuration information about the host cache, see Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

Because the `host_cache` table exposes the contents of the host cache, it can be examined using `SELECT` statements. This may help you diagnose the causes of connection problems. The Performance Schema must be enabled or this table is empty.

The `host_cache` table has these columns:

- `IP`

  The IP address of the client that connected to the server, expressed as a string.

- `HOST`

  The resolved DNS host name for that client IP, or `NULL` if the name is unknown.

- `HOST_VALIDATED`

  Whether the IP-to-host name-to-IP DNS resolution was performed successfully for the client IP. If `HOST_VALIDATED` is `YES`, the `HOST` column is used as the host name corresponding to the IP so that additional calls to DNS can be avoided. While `HOST_VALIDATED` is `NO`, DNS resolution is attempted for each connection attempt, until it eventually completes with either a valid result or a permanent error. This information enables the server to avoid caching bad or missing host names during temporary DNS failures, which would negatively affect clients forever.

- `SUM_CONNECT_ERRORS`

  The number of connection errors that are deemed “blocking” (assessed against the `max_connect_errors` system variable). Only protocol handshake errors are counted, and only for hosts that passed validation (`HOST_VALIDATED = YES`).

  Once `SUM_CONNECT_ERRORS` for a given host reaches the value of `max_connect_errors`, new connections from that host are blocked. The `SUM_CONNECT_ERRORS` value can exceed the `max_connect_errors` value because multiple connection attempts from a host can occur simultaneously while the host is not blocked. Any or all of them can fail, independently incrementing `SUM_CONNECT_ERRORS`, possibly beyond the value of `max_connect_errors`.

  Suppose that `max_connect_errors` is 200 and `SUM_CONNECT_ERRORS` for a given host is 199. If 10 clients attempt to connect from that host simultaneously, none of them are blocked because `SUM_CONNECT_ERRORS` has not reached 200. If blocking errors occur for five of the clients, `SUM_CONNECT_ERRORS` is increased by one for each client, for a resulting `SUM_CONNECT_ERRORS` value of 204. The other five clients succeed and are not blocked because the value of `SUM_CONNECT_ERRORS` when their connection attempts began had not reached 200. New connections from the host that begin after `SUM_CONNECT_ERRORS` reaches 200 are blocked.

- `COUNT_HOST_BLOCKED_ERRORS`

  The number of connections that were blocked because `SUM_CONNECT_ERRORS` exceeded the value of the `max_connect_errors` system variable.

- `COUNT_NAMEINFO_TRANSIENT_ERRORS`

  The number of transient errors during IP-to-host name DNS resolution.

- `COUNT_NAMEINFO_PERMANENT_ERRORS`

The number of permanent errors during IP-to-host name DNS resolution.

- `COUNT_FORMAT_ERRORS`

  The number of host name format errors. MySQL does not perform matching of `Host` column values in the `mysql.user` system table against host names for which one or more of the initial components of the name are entirely numeric, such as `1.2.example.com`. The client IP address is used instead. For the rationale why this type of matching does not occur, see Section 6.2.4, “Specifying Account Names”.

- `COUNT_ADDRINFO_TRANSIENT_ERRORS`

  The number of transient errors during host name-to-IP reverse DNS resolution.

- `COUNT_ADDRINFO_PERMANENT_ERRORS`

  The number of permanent errors during host name-to-IP reverse DNS resolution.

- `COUNT_FCRDNS_ERRORS`

  The number of forward-confirmed reverse DNS errors. These errors occur when IP-to-host name-to-IP DNS resolution produces an IP address that does not match the client originating IP address.

- `COUNT_HOST_ACL_ERRORS`

  The number of errors that occur because no users are permitted to connect from the client host. In such cases, the server returns `ER_HOST_NOT_PRIVILEGED` and does not even ask for a user name or password.

- `COUNT_NO_AUTH_PLUGIN_ERRORS`

  The number of errors due to requests for an unavailable authentication plugin. A plugin can be unavailable if, for example, it was never loaded or a load attempt failed.

- `COUNT_AUTH_PLUGIN_ERRORS`

  The number of errors reported by authentication plugins.

  An authentication plugin can report different error codes to indicate the root cause of a failure. Depending on the type of error, one of these columns is incremented: `COUNT_AUTHENTICATION_ERRORS`, `COUNT_AUTH_PLUGIN_ERRORS`, `COUNT_HANDSHAKE_ERRORS`. New return codes are an optional extension to the existing plugin API. Unknown or unexpected plugin errors are counted in the `COUNT_AUTH_PLUGIN_ERRORS` column.

- `COUNT_HANDSHAKE_ERRORS`

  The number of errors detected at the wire protocol level.

- `COUNT_PROXY_USER_ERRORS`

  The number of errors detected when proxy user A is proxied to another user B who does not exist.

- `COUNT_PROXY_USER_ACL_ERRORS`

  The number of errors detected when proxy user A is proxied to another user B who does exist but for whom A does not have the `PROXY` privilege.

- `COUNT_AUTHENTICATION_ERRORS`

The number of errors caused by failed authentication.

- `COUNT_SSL_ERRORS`

  The number of errors due to SSL problems.

- `COUNT_MAX_USER_CONNECTIONS_ERRORS`

  The number of errors caused by exceeding per-user connection quotas. See Section 6.2.20, “Setting Account Resource Limits”.

- `COUNT_MAX_USER_CONNECTIONS_PER_HOUR_ERRORS`

  The number of errors caused by exceeding per-user connections-per-hour quotas. See Section 6.2.20, “Setting Account Resource Limits”.

- `COUNT_DEFAULT_DATABASE_ERRORS`

  The number of errors related to the default database. For example, the database does not exist or the user has no privileges to access it.

- `COUNT_INIT_CONNECT_ERRORS`

  The number of errors caused by execution failures of statements in the `init_connect` system variable value.

- `COUNT_LOCAL_ERRORS`

  The number of errors local to the server implementation and not related to the network, authentication, or authorization. For example, out-of-memory conditions fall into this category.

- `COUNT_UNKNOWN_ERRORS`

  The number of other, unknown errors not accounted for by other columns in this table. This column is reserved for future use, in case new error conditions must be reported, and if preserving the backward compatibility and structure of the `host_cache` table is required.

- `FIRST_SEEN`

  The timestamp of the first connection attempt seen from the client in the `IP` column.

- `LAST_SEEN`

  The timestamp of the most recent connection attempt seen from the client in the `IP` column.

- `FIRST_ERROR_SEEN`

  The timestamp of the first error seen from the client in the `IP` column.

- `LAST_ERROR_SEEN`

  The timestamp of the most recent error seen from the client in the `IP` column.

The `host_cache` table has these indexes:

- Primary key on (`IP`)
- Index on (`HOST`)

The `FLUSH HOSTS` statement, `TRUNCATE TABLE host_cache` statement, and `mysqladmin flush-hosts` command have the same effect: They clear the host cache, remove all rows from the `host_cache` table that exposes the cache contents, and unblock any blocked hosts (see Section B.4.2.5, "Host 'host_name' is blocked"). `FLUSH HOSTS` and `mysqladmin flush-hosts` require the `RELOAD` privilege. `TRUNCATE TABLE` requires the `DROP` privilege for the `host_cache` table.

26.12.18.2 The keyring_keys table

MySQL Server supports a keyring that enables internal server components and plugins to securely store sensitive information for later retrieval. See Section 6.4.4, "The MySQL Keyring".

As of MySQL 8.0.16, the `keyring_keys` table exposes metadata for keys in the keyring. Key metadata includes key IDs, key owners, and backend key IDs. The `keyring_keys` table does *not* expose any sensitive keyring data such as key contents.

The `keyring_keys` table has these columns:

- `KEY_ID`

  The key identifier.

- `KEY_OWNER`

  The owner of the key.

- `BACKEND_KEY_ID`

  The ID used for the key by the keyring backend.

The `keyring_keys` table has no indexes.

`TRUNCATE TABLE` is not permitted for the `keyring_keys` table.

26.12.18.3 The log_status Table

The `log_status` table provides information that enables an online backup tool to copy the required log files without locking those resources for the duration of the copy process.

When the `log_status` table is queried, the server blocks logging and related administrative changes for just long enough to populate the table, then releases the resources. The `log_status` table informs the online backup which point it should copy up to in the master's binary log and `gtid_executed` record, and the relay log for each replication channel. It also provides relevant information for individual storage engines, such as the last log sequence number (LSN) and the LSN of the last checkpoint taken for the `InnoDB` storage engine.

The `log_status` table has these columns:

- `SERVER_UUID`

  The server UUID for this server instance. This is the generated unique value of the read-only system variable `server_uuid`.

- `LOCAL`

  The log position state information from the master, provided as a single JSON object with the following keys:

  `binary_log_file` The name of the current binary log file.

`binary_log_position`	The current binary log position at the time the `log_status` table was accessed.
`gtid_executed`	The current value of the global server variable `gtid_executed` at the time the `log_status` table was accessed. This information is consistent with the `binary_log_file` and `binary_log_position` keys.

- `REPLICATION`

  A JSON array of channels, each with the following information:

`channel_name`	The name of the replication channel. The default replication channel's name is the empty string (“”).
`relay_log_file`	The name of the current relay log file for the replication channel.
`relay_log_pos`	The current relay log position at the time the `log_status` table was accessed.

- `STORAGE_ENGINES`

  Relevant information from individual storage engines, provided as a JSON object with one key for each applicable storage engine.

The `log_status` table has no indexes.

The `BACKUP_ADMIN` privilege is required for access to the `log_status` table.

`TRUNCATE TABLE` is not permitted for the `log_status` table.

26.12.18.4 The performance_timers Table

The `performance_timers` table shows which event timers are available:

```
mysql> SELECT * FROM performance_schema.performance_timers;
+-------------+-----------------+------------------+----------------+
| TIMER_NAME  | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-------------+-----------------+------------------+----------------+
| CYCLE       |      2389029850 |                1 |             72 |
| NANOSECOND  |      1000000000 |                1 |            112 |
| MICROSECOND |         1000000 |                1 |            136 |
| MILLISECOND |            1036 |                1 |            168 |
+-------------+-----------------+------------------+----------------+
```

If the values associated with a given timer name are `NULL`, that timer is not supported on your platform. For an explanation of how event timing occurs, see Section 26.4.1, “Performance Schema Event Timing”.

The `performance_timers` table has these columns:

- `TIMER_NAME`

  The timer name.

- `TIMER_FREQUENCY`

  The number of timer units per second. For a cycle timer, the frequency is generally related to the CPU speed. For example, on a system with a 2.4GHz processor, the `CYCLE` may be close to 2400000000.

- `TIMER_RESOLUTION`

  Indicates the number of timer units by which timer values increase. If a timer has a resolution of 10, its value increases by 10 each time.

- `TIMER_OVERHEAD`

  The minimal number of cycles of overhead to obtain one timing with the given timer. The Performance Schema determines this value by invoking the timer 20 times during initialization and picking the smallest value. The total overhead really is twice this amount because the instrumentation invokes the timer at the start and end of each event. The timer code is called only for timed events, so this overhead does not apply for nontimed events.

The `performance_timers` table has these indexes:

- None

`TRUNCATE TABLE` is not permitted for the `performance_timers` table.

26.12.18.5 The threads Table

The `threads` table contains a row for each server thread. Each row contains information about a thread and indicates whether monitoring and historical event logging are enabled for it:

```
mysql> SELECT * FROM performance_schema.threads\G
*************************** 1. row ***************************
          THREAD_ID: 1
               NAME: thread/sql/main
               TYPE: BACKGROUND
     PROCESSLIST_ID: NULL
   PROCESSLIST_USER: NULL
   PROCESSLIST_HOST: NULL
     PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: NULL
   PROCESSLIST_TIME: 80284
  PROCESSLIST_STATE: NULL
   PROCESSLIST_INFO: NULL
   PARENT_THREAD_ID: NULL
               ROLE: NULL
       INSTRUMENTED: YES
            HISTORY: YES
    CONNECTION_TYPE: NULL
       THREAD_OS_ID: 489803
     RESOURCE_GROUP: SYS_default
...
*************************** 4. row ***************************
          THREAD_ID: 51
               NAME: thread/sql/one_connection
               TYPE: FOREGROUND
     PROCESSLIST_ID: 34
   PROCESSLIST_USER: isabella
   PROCESSLIST_HOST: localhost
     PROCESSLIST_DB: performance_schema
PROCESSLIST_COMMAND: Query
   PROCESSLIST_TIME: 0
  PROCESSLIST_STATE: Sending data
   PROCESSLIST_INFO: SELECT * FROM performance_schema.threads
   PARENT_THREAD_ID: 1
               ROLE: NULL
       INSTRUMENTED: YES
            HISTORY: YES
    CONNECTION_TYPE: SSL/TLS
       THREAD_OS_ID: 755399
```

```
     RESOURCE_GROUP: USR_default
...
```

When the Performance Schema initializes, it populates the `threads` table based on the threads in existence then. Thereafter, a new row is added each time the server creates a thread.

The `INSTRUMENTED` and `HISTORY` column values for new threads are determined by the contents of the `setup_actors` table. For information about how to use the `setup_actors` table to control these columns, see Section 26.4.6, “Pre-Filtering by Thread”.

Removal of rows from the `threads` table occurs when threads end. For a thread associated with a client session, removal occurs when the session ends. If a client has auto-reconnect enabled and the session reconnects after a disconnect, the session becomes associated with a new row in the `threads` table that has a different `PROCESSLIST_ID` value. The initial `INSTRUMENTED` and `HISTORY` values for the new thread may be different from those of the original thread: The `setup_actors` table may have changed in the meantime, and if the `INSTRUMENTED` or `HISTORY` value for the original thread was changed after the row was initialized, the change does not carry over to the new thread.

The `threads` table columns with names having a prefix of `PROCESSLIST_` provide information similar to that available from the `INFORMATION_SCHEMA.PROCESSLIST` table or the `SHOW PROCESSLIST` statement. Thus, all three sources provide thread-monitoring information. Use of `threads` differs from use of the other two sources in these ways:

- Access to `threads` does not require a mutex and has minimal impact on server performance. `INFORMATION_SCHEMA.PROCESSLIST` and `SHOW PROCESSLIST` have negative performance consequences because they require a mutex.
- `threads` provides additional information for each thread, such as whether it is a foreground or background thread, and the location within the server associated with the thread.
- `threads` provides information about background threads, so it can be used to monitor activity the other thread information sources cannot.
- You can enable or disable thread monitoring (that is, whether events executed by the thread are instrumented) and historical event logging. To control the initial `INSTRUMENTED` and `HISTORY` values for new foreground threads, use the `setup_actors` table. To control these aspects of existing threads, set the `INSTRUMENTED` and `HISTORY` columns of `threads` table rows. (For more information about the conditions under which thread monitoring and historical event logging occur, see the descriptions of the `INSTRUMENTED` and `HISTORY` columns.)

For these reasons, DBAs who perform server monitoring using `INFORMATION_SCHEMA.PROCESSLIST` or `SHOW PROCESSLIST` may wish to monitor using the `threads` table instead.

Note

For `INFORMATION_SCHEMA.PROCESSLIST` and `SHOW PROCESSLIST`, information about threads for other users is shown only if the current user has the `PROCESS` privilege. That is not true of the `threads` table; all rows are shown to any user who has the `SELECT` privilege for the table. Users who should not be able to see threads for other users should not be given that privilege.

The `threads` table has these columns:

- `THREAD_ID`

  A unique thread identifier.

- `NAME`

The name associated with the thread instrumentation code in the server. For example, `thread/sql/one_connection` corresponds to the thread function in the code responsible for handling a user connection, and `thread/sql/main` stands for the `main()` function of the server.

- `TYPE`

  The thread type, either `FOREGROUND` or `BACKGROUND`. User connection threads are foreground threads. Threads associated with internal server activity are background threads. Examples are internal `InnoDB` threads, “binlog dump” threads sending information to slaves, and slave I/O and SQL threads.

- `PROCESSLIST_ID`

  For threads that are displayed in the `INFORMATION_SCHEMA.PROCESSLIST` table, this is the same value displayed in the `ID` column of that table. It is also the value displayed in the `Id` column of `SHOW PROCESSLIST` output, and the value that `CONNECTION_ID()` would return within that thread.

  For background threads (threads not associated with a user connection), `PROCESSLIST_ID` is `NULL`, so the values are not unique.

- `PROCESSLIST_USER`

  The user associated with a foreground thread, `NULL` for a background thread.

- `PROCESSLIST_HOST`

  The host name of the client associated with a foreground thread, `NULL` for a background thread.

  Unlike the `HOST` column of the `INFORMATION_SCHEMA PROCESSLIST` table or the `Host` column of `SHOW PROCESSLIST` output, the `PROCESSLIST_HOST` column does not include the port number for TCP/IP connections. To obtain this information from the Performance Schema, enable the socket instrumentation (which is not enabled by default) and examine the `socket_instances` table:

```
mysql> SELECT NAME, ENABLED, TIMED
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'wait/io/socket%';
+----------------------------------------+---------+-------+
| NAME                                   | ENABLED | TIMED |
+----------------------------------------+---------+-------+
| wait/io/socket/sql/server_tcpip_socket | NO      | NO    |
| wait/io/socket/sql/server_unix_socket  | NO      | NO    |
| wait/io/socket/sql/client_connection   | NO      | NO    |
+----------------------------------------+---------+-------+
3 rows in set (0.01 sec)

mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED='YES'
       WHERE NAME LIKE 'wait/io/socket%';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM performance_schema.socket_instances\G
*************************** 1. row ***************************
          EVENT_NAME: wait/io/socket/sql/client_connection
OBJECT_INSTANCE_BEGIN: 140612577298432
            THREAD_ID: 31
            SOCKET_ID: 53
                   IP: ::ffff:127.0.0.1
                 PORT: 55642
                STATE: ACTIVE
...
```

- `PROCESSLIST_DB`

  The default database for the thread, or `NULL` if there is none.

- `PROCESSLIST_COMMAND`

  For foreground threads, the type of command the thread is executing on behalf of the client, or `Sleep` if the session is idle. For descriptions of thread commands, see Section 8.14, “Examining Thread Information”. The value of this column corresponds to the `COM_`*`xxx`* commands of the client/server protocol and `Com_`*`xxx`* status variables. See Section 5.1.10, “Server Status Variables”

  Background threads do not execute commands on behalf of clients, so this column may be `NULL`.

- `PROCESSLIST_TIME`

  The time in seconds that the thread has been in its current state.

- `PROCESSLIST_STATE`

  An action, event, or state that indicates what the thread is doing. For descriptions of `PROCESSLIST_STATE` values, see Section 8.14, “Examining Thread Information”. If the value if `NULL`, the thread may correspond to an idle client session or the work it is doing is not instrumented with stages.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that bears investigation.

- `PROCESSLIST_INFO`

  The statement the thread is executing, or `NULL` if it is not executing any statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a `CALL` statement executes a stored procedure that is executing a `SELECT` statement, the `PROCESSLIST_INFO` value shows the `SELECT` statement.

- `PARENT_THREAD_ID`

  If this thread is a subthread (spawned by another thread), this is the `THREAD_ID` value of the spawning thread.

- `ROLE`

  Unused.

- `INSTRUMENTED`

  Whether events executed by the thread are instrumented. The value is `YES` or `NO`.

  - For foreground threads, the initial `INSTRUMENTED` value is determined by whether the user account associated with the thread matches any row in the `setup_actors` table. Matching is based on the values of the `PROCESSLIST_USER` and `PROCESSLIST_HOST` columns.

    If the thread spawns a subthread, matching occurs again for the `threads` table row created for the subthread.

  - For background threads, `INSTRUMENTED` is `YES` by default. `setup_actors` is not consulted because there is no associated user for background threads.

  - For any thread, its `INSTRUMENTED` value can be changed during the lifetime of the thread.

For monitoring of events executed by the thread to occur, these things must be true:

- The `thread_instrumentation` consumer in the `setup_consumers` table must be `YES`.
- The `threads.INSTRUMENTED` column must be `YES`.
- Monitoring occurs only for those thread events produced from instruments that have the `ENABLED` column set to `YES` in the `setup_instruments` table.

- `HISTORY`

  Whether to log historical events for the thread. The value is `YES` or `NO`.

  - For foreground threads, the initial `HISTORY` value is determined by whether the user account associated with the thread matches any row in the `setup_actors` table. Matching is based on the values of the `PROCESSLIST_USER` and `PROCESSLIST_HOST` columns.

    If the thread spawns a subthread, matching occurs again for the `threads` table row created for the subthread.
  - For background threads, `HISTORY` is `YES` by default. `setup_actors` is not consulted because there is no associated user for background threads.
  - For any thread, its `HISTORY` value can be changed during the lifetime of the thread.

  For historical event logging for the thread to occur, these things must be true:

  - The appropriate history-related consumers in the `setup_consumers` table must be enabled. For example, wait event logging in the `events_waits_history` and `events_waits_history_long` tables requires the corresponding `events_waits_history` and `events_waits_history_long` consumers to be `YES`.
  - The `threads.HISTORY` column must be `YES`.
  - Logging occurs only for those thread events produced from instruments that have the `ENABLED` column set to `YES` in the `setup_instruments` table.

- `CONNECTION_TYPE`

  The protocol used to establish the connection, or `NULL` for background threads. Permitted values are `TCP/IP` (TCP/IP connection established without encryption), `SSL/TLS` (TCP/IP connection established with encryption), `Socket` (Unix socket file connection), `Named Pipe` (Windows named pipe connection), and `Shared Memory` (Windows shared memory connection).

- `THREAD_OS_ID`

  The thread or task identifier as defined by the underlying operating system, if there is one:

  - When a MySQL thread is associated with the same operating system thread for its lifetime, `THREAD_OS_ID` contains the operating system thread ID.
  - When a MySQL thread is not associated with the same operating system thread for its lifetime, `THREAD_OS_ID` contains `NULL`. This is typical for user sessions when the thread pool plugin is used (see Section 5.6.3, “MySQL Enterprise Thread Pool”).

  For Windows, `THREAD_OS_ID` corresponds to the thread ID visible in Process Explorer (https://technet.microsoft.com/en-us/sysinternals/bb896653.aspx).

For Linux, THREAD_OS_ID corresponds to the value of the gettid() function. This value is exposed, for example, using the perf or ps -L commands, or in the proc file system (/proc/[pid]/task/[tid]). For more information, see the perf-stat(1), ps(1), and proc(5) man pages.

- RESOURCE_GROUP

  The resource group label. This value is NULL if resource groups are not supported on the current platform or server configuration (see Resource Group Restrictions).

The threads table has these indexes:

- Primary key on (THREAD_ID)
- Index on (NAME)
- Index on (PROCESSLIST_ID)
- Index on (PROCESSLIST_USER, PROCESSLIST_HOST)
- Index on (PROCESSLIST_HOST)
- Index on (THREAD_OS_ID)
- Index on (RESOURCE_GROUP)

TRUNCATE TABLE is not permitted for the threads table.

26.12.18.6 The user_defined_functions Table

The user_defined_functions table contains a row for each user-defined function (UDF) registered by a server component or plugin, or by a CREATE FUNCTION statement.

user_defined_functions has these columns:

- UDF_NAME

  The UDF name as referred to in SQL statements. The value is NULL if the function was registered by a CREATE FUNCTION statement and is in the process of unloading.

- UDF_RETURN_TYPE

  The UDF return value type. The value is one of int, decimal, real, char, or row.

- UDF_TYPE

  The UDF type. The value is one of function (scalar) or aggregate.

- UDF_LIBRARY

  The name of the UDF library file containing the executable UDF code. The file is located in the directory named by the plugin_dir system variable. The value is NULL if the UDF was registered using a server component service rather than by a CREATE FUNCTION statement.

- UDF_USAGE_COUNT

  The current UDF usage count.

The user_defined_functions table has these indexes:

- Primary key on (UDF_NAME)

`TRUNCATE TABLE` is not permitted for the `user_defined_functions` table.

26.13 Performance Schema Option and Variable Reference

Table 26.4 Performance Schema Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
performance_schema	Yes	Yes	Yes		Global	No
Performance_schema_accounts_lost				Yes	Global	No
performance_schema_accounts_size	Yes	Yes	Yes		Global	No
Performance_schema_cond_classes_lost				Yes	Global	No
Performance_schema_cond_instances_lost				Yes	Global	No
performance-schema-consumer-events-stages-current	Yes	Yes				
performance-schema-consumer-events-stages-history	Yes	Yes				
performance-schema-consumer-events-stages-history-long	Yes	Yes				
performance-schema-consumer-events-statements-current	Yes	Yes				
performance-schema-consumer-events-statements-history	Yes	Yes				
performance-schema-consumer-events-statements-history-long	Yes	Yes				
performance-schema-consumer-events-transactions-current	Yes	Yes				
performance-schema-consumer-events-transactions-history	Yes	Yes				
performance-schema-consumer-events-transactions-history-long	Yes	Yes				
performance-schema-consumer-events-waits-current	Yes	Yes				
performance-schema-consumer-events-waits-history	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
performance-schema-consumer-events-waits-history-long	Yes	Yes				
performance-schema-consumer-global-instrumentation	Yes	Yes				
performance-schema-consumer-statements-digest	Yes	Yes				
performance-schema-consumer-thread-instrumentation	Yes	Yes				
Performance_schema_digest_lost				Yes	Global	No
performance_schema_digests_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_transactions_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_transactions_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_size	Yes	Yes	Yes		Global	No
Performance_schema_file_classes_lost				Yes	Global	No
Performance_schema_file_handles_lost				Yes	Global	No
Performance_schema_file_instances_lost				Yes	Global	No
Performance_schema_hosts_lost				Yes	Global	No
performance_schema_hosts_size	Yes	Yes	Yes		Global	No
performance-schema-instrument	Yes	Yes				
Performance_schema_locker_lost				Yes	Global	No
performance_schema_max_cond_classes	Yes	Yes	Yes		Global	No
performance_schema_max_cond_instances	Yes	Yes	Yes		Global	No
performance_schema_max_digest_length	Yes	Yes	Yes		Global	No
performance_schema_max_file_classes	Yes	Yes	Yes		Global	No
performance_schema_max_file_handles	Yes	Yes	Yes		Global	No
performance_schema_max_file_instances	Yes	Yes	Yes		Global	No
performance_schema_max_memory_classes	Yes	Yes	Yes		Global	No
performance_schema_max_metadata_locks	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_classes	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_instances	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
performance_schema_max_prepared_statements_instances	Yes	Yes	Yes		Global	No
performance_schema_max_program_instances	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_classes	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_instances	Yes	Yes	Yes		Global	No
performance_schema_max_socket_classes	Yes	Yes	Yes		Global	No
performance_schema_max_socket_instances	Yes	Yes	Yes		Global	No
performance_schema_max_stage_classes	Yes	Yes	Yes		Global	No
performance_schema_max_statement_classes	Yes	Yes	Yes		Global	No
performance_schema_max_statement_stack	Yes	Yes	Yes		Global	No
performance_schema_max_table_handles	Yes	Yes	Yes		Global	No
performance_schema_max_table_instances	Yes	Yes	Yes		Global	No
performance_schema_max_thread_classes	Yes	Yes	Yes		Global	No
performance_schema_max_thread_instances	Yes	Yes	Yes		Global	No
Performance_schema_memory_classes_lost				Yes	Global	No
Performance_schema_metadata_lock_lost				Yes	Global	No
Performance_schema_mutex_classes_lost				Yes	Global	No
Performance_schema_mutex_instances_lost				Yes	Global	No
Performance_schema_nested_statement_lost				Yes	Global	No
Performance_schema_prepared_statements_lost				Yes	Global	No
Performance_schema_program_lost				Yes	Global	No
Performance_schema_rwlock_classes_lost				Yes	Global	No
Performance_schema_rwlock_instances_lost				Yes	Global	No
Performance_schema_session_connect_attrs_lost				Yes	Global	No
performance_schema_session_connect_attrs_size	Yes	Yes	Yes		Global	No
performance_schema_setup_actors_size	Yes	Yes	Yes		Global	No
performance_schema_setup_objects_size	Yes	Yes	Yes		Global	No
Performance_schema_socket_classes_lost				Yes	Global	No
Performance_schema_socket_instances_lost				Yes	Global	No
Performance_schema_stage_classes_lost				Yes	Global	No
Performance_schema_statement_classes_lost				Yes	Global	No
Performance_schema_table_handles_lost				Yes	Global	No
Performance_schema_table_instances_lost				Yes	Global	No
Performance_schema_thread_classes_lost				Yes	Global	No
Performance_schema_thread_instances_lost				Yes	Global	No
Performance_schema_users_lost				Yes	Global	No
performance_schema_users_size	Yes	Yes	Yes		Global	No

26.14 Performance Schema Command Options

Performance Schema parameters can be specified at server startup on the command line or in option files to configure Performance Schema instruments and consumers. Runtime configuration is also possible in many cases (see Section 26.4, “Performance Schema Runtime Configuration”), but startup configuration must be used when runtime configuration is too late to affect instruments that have already been initialized during the startup process.

Performance Schema consumers and instruments can be configured at startup using the following syntax. For additional details, see Section 26.3, “Performance Schema Startup Configuration”.

- `--performance-schema-consumer-`*`consumer_name`*`=value`

  Configure a Performance Schema consumer. Consumer names in the `setup_consumers` table use underscores, but for consumers set at startup, dashes and underscores within the name are equivalent. Options for configuring individual consumers are detailed later in this section.

- `--performance-schema-instrument=`*`instrument_name`*`=value`

  Configure a Performance Schema instrument. The name may be given as a pattern to configure instruments that match the pattern.

The following items configure individual consumers:

- `--performance-schema-consumer-events-stages-current=value`

  Configure the `events-stages-current` consumer.

- `--performance-schema-consumer-events-stages-history=value`

  Configure the `events-stages-history` consumer.

- `--performance-schema-consumer-events-stages-history-long=value`

  Configure the `events-stages-history-long` consumer.

- `--performance-schema-consumer-events-statements-current=value`

  Configure the `events-statements-current` consumer.

- `--performance-schema-consumer-events-statements-history=value`

  Configure the `events-statements-history` consumer.

- `--performance-schema-consumer-events-statements-history-long=value`

  Configure the `events-statements-history-long` consumer.

- `--performance-schema-consumer-events-transactions-current=value`

  Configure the Performance Schema `events-transactions-current` consumer.

- `--performance-schema-consumer-events-transactions-history=value`

  Configure the Performance Schema `events-transactions-history` consumer.

- `--performance-schema-consumer-events-transactions-history-long=value`

  Configure the Performance Schema `events-transactions-history-long` consumer.

- `--performance-schema-consumer-events-waits-current=value`

  Configure the `events-waits-current` consumer.

- `--performance-schema-consumer-events-waits-history=value`

  Configure the `events-waits-history` consumer.

- `--performance-schema-consumer-events-waits-history-long=value`

  Configure the `events-waits-history-long` consumer.

- `--performance-schema-consumer-global-instrumentation=value`

  Configure the `global-instrumentation` consumer.

- `--performance-schema-consumer-statements-digest=value`

  Configure the `statements-digest` consumer.

- `--performance-schema-consumer-thread-instrumentation=value`

  Configure the `thread-instrumentation` consumer.

26.15 Performance Schema System Variables

The Performance Schema implements several system variables that provide configuration information:

```
mysql> SHOW VARIABLES LIKE 'perf%';
+-----------------------------------------------------------+-------+
| Variable_name                                             | Value |
+-----------------------------------------------------------+-------+
| performance_schema                                        | ON    |
| performance_schema_accounts_size                          | -1    |
| performance_schema_digests_size                           | 10000 |
| performance_schema_events_stages_history_long_size        | 10000 |
| performance_schema_events_stages_history_size             | 10    |
| performance_schema_events_statements_history_long_size    | 10000 |
| performance_schema_events_statements_history_size         | 10    |
| performance_schema_events_transactions_history_long_size | 10000 |
| performance_schema_events_transactions_history_size       | 10    |
| performance_schema_events_waits_history_long_size         | 10000 |
| performance_schema_events_waits_history_size              | 10    |
| performance_schema_hosts_size                             | -1    |
| performance_schema_max_cond_classes                       | 80    |
| performance_schema_max_cond_instances                     | -1    |
| performance_schema_max_digest_length                      | 1024  |
| performance_schema_max_file_classes                       | 50    |
| performance_schema_max_file_handles                       | 32768 |
| performance_schema_max_file_instances                     | -1    |
| performance_schema_max_index_stat                         | -1    |
| performance_schema_max_memory_classes                     | 320   |
| performance_schema_max_metadata_locks                     | -1    |
| performance_schema_max_mutex_classes                      | 220   |
| performance_schema_max_mutex_instances                    | -1    |
| performance_schema_max_prepared_statements_instances      | -1    |
| performance_schema_max_program_instances                  | -1    |
| performance_schema_max_rwlock_classes                     | 40    |
| performance_schema_max_rwlock_instances                   | -1    |
| performance_schema_max_socket_classes                     | 10    |
| performance_schema_max_socket_instances                   | -1    |
| performance_schema_max_sql_text_length                    | 1024  |
```

```
| performance_schema_max_stage_classes                    | 150   |
| performance_schema_max_statement_classes                | 192   |
| performance_schema_max_statement_stack                  | 10    |
| performance_schema_max_table_handles                    | -1    |
| performance_schema_max_table_instances                  | -1    |
| performance_schema_max_table_lock_stat                  | -1    |
| performance_schema_max_thread_classes                   | 50    |
| performance_schema_max_thread_instances                 | -1    |
| performance_schema_session_connect_attrs_size           | 512   |
| performance_schema_setup_actors_size                    | -1    |
| performance_schema_setup_objects_size                   | -1    |
| performance_schema_users_size                           | -1    |
+---------------------------------------------------------+-------+
```

Performance Schema system variables can be set at server startup on the command line or in option files, and many can be set at runtime. See Section 26.13, “Performance Schema Option and Variable Reference”.

The Performance Schema automatically sizes the values of several of its parameters at server startup if they are not set explicitly. For more information, see Section 26.3, “Performance Schema Startup Configuration”.

Performance Schema system variables have the following meanings:

- `performance_schema`

Property	Value
Command-Line Format	`--performance-schema[={OFF|ON}]`
System Variable	`performance_schema`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

The value of this variable is `ON` or `OFF` to indicate whether the Performance Schema is enabled. By default, the value is `ON`. At server startup, you can specify this variable with no value or a value of `ON` or 1 to enable it, or with a value of `OFF` or 0 to disable it.

Even when the Performance Schema is disabled, it continues to populate the `global_variables`, `session_variables`, `global_status`, and `session_status` tables. This occurs as necessary to permit the results for the `SHOW VARIABLES` and `SHOW STATUS` statements to be drawn from those tables, depending on the setting of the `show_compatibility_56` system variable. The Performance Schema also populates some of the replication tables when disabled.

- `performance_schema_accounts_size`

Property	Value
Command-Line Format	`--performance-schema-accounts-size=#`
System Variable	`performance_schema_accounts_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)
Minimum Value	`-1` (signifies autoscaling; do not assign this literal value)
Maximum Value	`1048576`

The number of rows in the `accounts` table. If this variable is 0, the Performance Schema does not maintain connection statistics in the `accounts` table or status variable information in the `status_by_account` table.

- `performance_schema_digests_size`

Property	Value
Command-Line Format	`--performance-schema-digests-size=#`
System Variable	`performance_schema_digests_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`-1`
Maximum Value	`1048576`

The maximum number of rows in the `events_statements_summary_by_digest` table. If this maximum is exceeded such that a digest cannot be instrumented, the Performance Schema increments the `Performance_schema_digest_lost` status variable.

For more information about statement digesting, see Section 26.10, “Performance Schema Statement Digests and Sampling”.

- `performance_schema_error_size`

Property	Value
Command-Line Format	`--performance-schema-error-size=#`
System Variable	`performance_schema_error_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`number of server error codes`
Minimum Value	`0`
Maximum Value	`1048576`

The number of instrumented server error codes. The default value is the actual number of server error codes. Although the value can be set anywhere from 0 to its maximum, the intended use is to set it to either its default (to instrument all errors) or 0 (to instrument no errors).

Error information is aggregated in summary tables; see Section 26.12.17.11, “Error Summary Tables”. If an error occurs that is not instrumented, information for the occurrence is aggregated to the `NULL` row in each summary table; that is, to the row with `ERROR_NUMBER=0`, `ERROR_NAME=NULL`, and `SQLSTATE=NULL`.

- `performance_schema_events_stages_history_long_size`

Property	Value
Command-Line Format	`--performance-schema-events-stages-history-long-size=#`
System Variable	`performance_schema_events_stages_history_long_size`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The number of rows in the `events_stages_history_long` table.

- `performance_schema_events_stages_history_size`

Property	Value
Command-Line Format	`--performance-schema-events-stages-history-size=#`
System Variable	`performance_schema_events_stages_history_size`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The number of rows per thread in the `events_stages_history` table.

- `performance_schema_events_statements_history_long_size`

Property	Value
Command-Line Format	`--performance-schema-events-statements-history-long-size=#`
System Variable	`performance_schema_events_statements_history_long_size`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The number of rows in the `events_statements_history_long` table.

- `performance_schema_events_statements_history_size`

Property	Value
Command-Line Format	`--performance-schema-events-statements-history-size=#`
System Variable	`performance_schema_events_statements_history_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The number of rows per thread in the `events_statements_history` table.

- `performance_schema_events_transactions_history_long_size`

Property	Value
Command-Line Format	`--performance-schema-events-transactions-history-long-size=#`
System Variable	`performance_schema_events_transactions_history_long_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The number of rows in the `events_transactions_history_long` table.

- `performance_schema_events_transactions_history_size`

Property	Value
Command-Line Format	`--performance-schema-events-transactions-history-size=#`
System Variable	`performance_schema_events_transactions_history_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The number of rows per thread in the `events_transactions_history` table.

- `performance_schema_events_waits_history_long_size`

Property	Value
Command-Line Format	`--performance-schema-events-waits-history-long-size=#`
System Variable	`performance_schema_events_waits_history_long_size`

Property	Value
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)

The number of rows in the events_waits_history_long table.

- performance_schema_events_waits_history_size

Property	Value
Command-Line Format	--performance-schema-events-waits-history-size=#
System Variable	performance_schema_events_waits_history_size
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)

The number of rows per thread in the events_waits_history table.

- performance_schema_hosts_size

Property	Value
Command-Line Format	--performance-schema-hosts-size=#
System Variable	performance_schema_hosts_size
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	-1 (signifies autoscaling; do not assign this literal value)
Minimum Value	-1 (signifies autoscaling; do not assign this literal value)
Maximum Value	1048576

The number of rows in the hosts table. If this variable is 0, the Performance Schema does not maintain connection statistics in the hosts table or status variable information in the status_by_host table.

- performance_schema_max_cond_classes

Property	Value
Command-Line Format	--performance-schema-max-cond-classes=#
System Variable	performance_schema_max_cond_classes
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	80
Minimum Value	0
Maximum Value (>= 8.0.12)	1024
Maximum Value (<= 8.0.11)	256

The maximum number of condition instruments. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- performance_schema_max_cond_instances

Property	Value
Command-Line Format	--performance-schema-max-cond-instances=#
System Variable	performance_schema_max_cond_instances
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	-1 (signifies autoscaling; do not assign this literal value)

The maximum number of instrumented condition objects. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- performance_schema_max_digest_length

Property	Value
Command-Line Format	--performance-schema-max-digest-length=#
System Variable	performance_schema_max_digest_length
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	1024
Minimum Value	0
Maximum Value	1048576

The maximum number of bytes of memory reserved per statement for computation of normalized statement digest values in the Performance Schema. This variable is related to max_digest_length; see the description of that variable in Section 5.1.8, “Server System Variables”.

For more information about statement digesting, including considerations regarding memory use, see Section 26.10, “Performance Schema Statement Digests and Sampling”.

- performance_schema_max_digest_sample_age

Property	Value
Command-Line Format	`--performance-schema-max-digest-sample-age=#`
Introduced	8.0.3
System Variable	`performance_schema_max_digest_sample_age`
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	`60`
Minimum Value	`0`
Maximum Value	`1048576`

This variable affects statement sampling for the `events_statements_summary_by_digest` table. When a new table row is inserted, the statement that produced the row digest value is stored as the current sample statement associated with the digest. Thereafter, when the server sees other statements with the same digest value, it determines whether to use the new statement to replace the current sample statement (that is, whether to resample). Resampling policy is based on the comparative wait times of the current sample statement and new statement and, optionally, the age of the current sample statement:

- Resampling based on wait times: If the new statement wait time has a wait time greater than that of the current sample statement, it becomes the current sample statement.
- Resampling based on age: If the `performance_schema_max_digest_sample_age` system variable has a value greater than zero and the current sample statement is more than that many seconds old, the current statement is considered “too old” and the new statement replaces it. This occurs even if the new statement wait time is less than that of the current sample statement.

For information about statement sampling, see Section 26.10, “Performance Schema Statement Digests and Sampling”.

- `performance_schema_max_file_classes`

Property	Value
Command-Line Format	`--performance-schema-max-file-classes=#`
System Variable	`performance_schema_max_file_classes`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`80`
Minimum Value	`0`
Maximum Value (>= 8.0.12)	`1024`
Maximum Value (<= 8.0.11)	`256`

The maximum number of file instruments. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_file_handles`

Property	Value
Command-Line Format	`--performance-schema-max-file-handles=#`
System Variable	`performance_schema_max_file_handles`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`32768`

The maximum number of opened file objects. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

The value of `performance_schema_max_file_handles` should be greater than the value of `open_files_limit`: `open_files_limit` affects the maximum number of open file handles the server can support and `performance_schema_max_file_handles` affects how many of these file handles can be instrumented.

- `performance_schema_max_file_instances`

Property	Value
Command-Line Format	`--performance-schema-max-file-instances=#`
System Variable	`performance_schema_max_file_instances`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The maximum number of instrumented file objects. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_index_stat`

Property	Value
Command-Line Format	`--performance-schema-max-index-stat=#`
System Variable	`performance_schema_max_index_stat`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The maximum number of indexes for which the Performance Schema maintains statistics. If this maximum is exceeded such that index statistics are lost, the Performance Schema increments the `Performance_schema_index_stat_lost` status variable. The default value is autosized using the value of `performance_schema_max_table_instances`.

- `performance_schema_max_memory_classes`

Property	Value
Command-Line Format	`--performance-schema-max-memory-classes=#`
System Variable	`performance_schema_max_memory_classes`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.1)	`450`
Default Value (8.0.0)	`350`

The maximum number of memory instruments. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_metadata_locks`

Property	Value
Command-Line Format	`--performance-schema-max-metadata-locks=#`
System Variable	`performance_schema_max_metadata_locks`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The maximum number of metadata lock instruments. This value controls the size of the `metadata_locks` table. If this maximum is exceeded such that a metadata lock cannot be instrumented, the Performance Schema increments the `Performance_schema_metadata_lock_lost` status variable.

- `performance_schema_max_mutex_classes`

Property	Value
Command-Line Format	`--performance-schema-max-mutex-classes=#`
System Variable	`performance_schema_max_mutex_classes`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value (>= 8.0.12)	`300`
Default Value (>= 8.0.3, <= 8.0.11)	`250`
Default Value (>= 8.0.1, <= 8.0.2)	`220`
Default Value (8.0.0)	`200`
Minimum Value	`0`
Maximum Value (>= 8.0.12)	`1024`
Maximum Value (<= 8.0.11)	`256`

The maximum number of mutex instruments. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_mutex_instances`

Property	Value
Command-Line Format	`--performance-schema-max-mutex-instances=#`
System Variable	`performance_schema_max_mutex_instances`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The maximum number of instrumented mutex objects. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_prepared_statements_instances`

Property	Value
Command-Line Format	`--performance-schema-max-prepared-statements-instances=#`
System Variable	`performance_schema_max_prepared_statements_instances`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The maximum number of rows in the `prepared_statements_instances` table. If this maximum is exceeded such that a prepared statement cannot be instrumented, the Performance Schema increments the `Performance_schema_prepared_statements_lost` status variable. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

The default value of this variable is autosized based on the value of the `max_prepared_stmt_count` system variable.

- `performance_schema_max_rwlock_classes`

Property	Value
Command-Line Format	`--performance-schema-max-rwlock-classes=#`
System Variable	`performance_schema_max_rwlock_classes`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`40`
Minimum Value	`0`
Maximum Value (>= 8.0.12)	`1024`
Maximum Value (<= 8.0.11)	`256`

The maximum number of rwlock instruments. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_program_instances`

Property	Value
Command-Line Format	`--performance-schema-max-program-instances=#`
System Variable	`performance_schema_max_program_instances`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The maximum number of stored programs for which the Performance Schema maintains statistics. If this maximum is exceeded, the Performance Schema increments the `Performance_schema_program_lost` status variable. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_rwlock_instances`

Property	Value
Command-Line Format	`--performance-schema-max-rwlock-instances=#`
System Variable	`performance_schema_max_rwlock_instances`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer

Property	Value
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The maximum number of instrumented rwlock objects. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_socket_classes`

Property	Value
Command-Line Format	`--performance-schema-max-socket-classes=#`
System Variable	`performance_schema_max_socket_classes`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value (>= 8.0.12)	`1024`
Maximum Value (<= 8.0.11)	`256`

The maximum number of socket instruments. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_socket_instances`

Property	Value
Command-Line Format	`--performance-schema-max-socket-instances=#`
System Variable	`performance_schema_max_socket_instances`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The maximum number of instrumented socket objects. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_sql_text_length`

Property	Value
Command-Line Format	`--performance-schema-max-sql-text-length=#`
System Variable	`performance_schema_max_sql_text_length`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No

Property	Value
Type	Integer
Default Value	`1024`
Minimum Value	`0`
Maximum Value	`1048576`

The maximum number of bytes used to store SQL statements. The value applies to storage required for these columns:

- The `SQL_TEXT` column of the `events_statements_current`, `events_statements_history`, and `events_statements_history_long` statement event tables.
- The `QUERY_SAMPLE_TEXT` column of the `events_statements_summary_by_digest` summary table.

Any bytes in excess of `performance_schema_max_sql_text_length` are discarded and do not appear in the column. Statements differing only after that many initial bytes are indistinguishable in the column.

Decreasing the `performance_schema_max_sql_text_length` value reduces memory use but causes more statements to become indistinguishable if they differ only at the end. Increasing the value increases memory use but permits longer statements to be distinguished.

- `performance_schema_max_stage_classes`

Property	Value
Command-Line Format	`--performance-schema-max-stage-classes=#`
System Variable	`performance_schema_max_stage_classes`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`150`
Minimum Value	`0`
Maximum Value (>= 8.0.12)	`1024`
Maximum Value (<= 8.0.11)	`256`

The maximum number of stage instruments. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_statement_classes`

Property	Value
Command-Line Format	`--performance-schema-max-statement-classes=#`
System Variable	`performance_schema_max_statement_classes`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The maximum number of statement instruments. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

The default value is calculated at server build time based on the number of commands in the client/server protocol and the number of SQL statement types supported by the server.

This variable should not be changed, unless to set it to 0 to disable all statement instrumentation and save all memory associated with it. Setting the variable to nonzero values other than the default has no benefit; in particular, values larger than the default cause more memory to be allocated then is needed.

- `performance_schema_max_statement_stack`

Property	Value
Command-Line Format	`--performance-schema-max-statement-stack=#`
System Variable	`performance_schema_max_statement_stack`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`10`

The maximum depth of nested stored program calls for which the Performance Schema maintains statistics. When this maximum is exceeded, the Performance Schema increments the `Performance_schema_nested_statement_lost` status variable for each stored program statement executed.

- `performance_schema_max_table_handles`

Property	Value
Command-Line Format	`--performance-schema-max-table-handles=#`
System Variable	`performance_schema_max_table_handles`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The maximum number of opened table objects. This value controls the size of the `table_handles` table. If this maximum is exceeded such that a table handle cannot be instrumented, the Performance Schema increments the `Performance_schema_table_handles_lost` status variable. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_table_instances`

Property	Value
Command-Line Format	`--performance-schema-max-table-instances=#`
System Variable	`performance_schema_max_table_instances`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The maximum number of instrumented table objects. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_table_lock_stat`

Property	Value
Command-Line Format	`--performance-schema-max-table-lock-stat=#`
System Variable	`performance_schema_max_table_lock_stat`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The maximum number of tables for which the Performance Schema maintains lock statistics. If this maximum is exceeded such that table lock statistics are lost, the Performance Schema increments the `Performance_schema_table_lock_stat_lost` status variable.

- `performance_schema_max_thread_classes`

Property	Value
Command-Line Format	`--performance-schema-max-thread-classes=#`
System Variable	`performance_schema_max_thread_classes`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value (>= 8.0.3)	`100`
Default Value (<= 8.0.2)	`50`
Minimum Value	`0`
Maximum Value (>= 8.0.12)	`1024`
Maximum Value (<= 8.0.11)	`256`

The maximum number of thread instruments. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_thread_instances`

Property	Value
Command-Line Format	`--performance-schema-max-thread-instances=#`
System Variable	`performance_schema_max_thread_instances`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The maximum number of instrumented thread objects. The value controls the size of the `threads` table. If this maximum is exceeded such that a thread cannot be instrumented, the Performance Schema increments the `Performance_schema_thread_instances_lost` status variable. For information about how to set and use this variable, see Section 26.7, “Performance Schema Status Monitoring”.

The `max_connections` system variable affects how many threads can run in the server. `performance_schema_max_thread_instances` affects how many of these running threads can be instrumented.

The `variables_by_thread` and `status_by_thread` tables contain system and status variable information only about foreground threads. If not all threads are instrumented by the Performance Schema, this table will miss some rows. In this case, the `Performance_schema_thread_instances_lost` status variable will be greater than zero.

- `performance_schema_session_connect_attrs_size`

Property	Value
Command-Line Format	`--performance-schema-session-connect-attrs-size=#`
System Variable	`performance_schema_session_connect_attrs_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`-1`
Maximum Value	`1048576`

The amount of preallocated memory per thread reserved to hold connection attribute key-value pairs. If the aggregate size of connection attribute data sent by a client is larger than this amount, the Performance Schema truncates the attribute data, increments the `Performance_schema_session_connect_attrs_lost` status variable, and writes a message to the error log indicating that truncation occurred if the `log_error_verbosity` system variable is greater than 1. A `_truncated` attribute is also added to the session attributes with a value indicating how many bytes were lost, if the attribute buffer has sufficient space. This enables the Performance

Schema to expose per-connection truncation information in the connection attribute tables. This information can be examined without having to check the error log.

The default value of `performance_schema_session_connect_attrs_size` is autosized at server startup. This value may be small, so if truncation occurs (`Performance_schema_session_connect_attrs_lost` becomes nonzero), you may wish to set `performance_schema_session_connect_attrs_size` explicitly to a larger value.

Although the maximum permitted `performance_schema_session_connect_attrs_size` value is 1MB, the effective maximum is 64KB because the server imposes a limit of 64KB on the aggregate size of connection attribute data it will accept. If a client attempts to send more than 64KB of attribute data, the server rejects the connection. For more information, see Section 26.12.9, “Performance Schema Connection Attribute Tables”.

- `performance_schema_setup_actors_size`

Property	Value
Command-Line Format	`--performance-schema-setup-actors-size=#`
System Variable	`performance_schema_setup_actors_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The number of rows in the `setup_actors` table.

- `performance_schema_setup_objects_size`

Property	Value
Command-Line Format	`--performance-schema-setup-objects-size=#`
System Variable	`performance_schema_setup_objects_size`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)

The number of rows in the `setup_objects` table.

- `performance_schema_users_size`

Property	Value
Command-Line Format	`--performance-schema-users-size=#`
System Variable	`performance_schema_users_size`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	Integer
Default Value	`-1` (signifies autoscaling; do not assign this literal value)
Minimum Value	`-1` (signifies autoscaling; do not assign this literal value)
Maximum Value	`1048576`

The number of rows in the `users` table. If this variable is 0, the Performance Schema does not maintain connection statistics in the `users` table or status variable information in the `status_by_user` table.

26.16 Performance Schema Status Variables

The Performance Schema implements several status variables that provide information about instrumentation that could not be loaded or created due to memory constraints:

```
mysql> SHOW STATUS LIKE 'perf%';
+-------------------------------------------+-------+
| Variable_name                             | Value |
+-------------------------------------------+-------+
| Performance_schema_accounts_lost          | 0     |
| Performance_schema_cond_classes_lost      | 0     |
| Performance_schema_cond_instances_lost    | 0     |
| Performance_schema_file_classes_lost      | 0     |
| Performance_schema_file_handles_lost      | 0     |
| Performance_schema_file_instances_lost    | 0     |
| Performance_schema_hosts_lost             | 0     |
| Performance_schema_locker_lost            | 0     |
| Performance_schema_mutex_classes_lost     | 0     |
| Performance_schema_mutex_instances_lost   | 0     |
| Performance_schema_rwlock_classes_lost    | 0     |
| Performance_schema_rwlock_instances_lost  | 0     |
| Performance_schema_socket_classes_lost    | 0     |
| Performance_schema_socket_instances_lost  | 0     |
| Performance_schema_stage_classes_lost     | 0     |
| Performance_schema_statement_classes_lost | 0     |
| Performance_schema_table_handles_lost     | 0     |
| Performance_schema_table_instances_lost   | 0     |
| Performance_schema_thread_classes_lost    | 0     |
| Performance_schema_thread_instances_lost  | 0     |
| Performance_schema_users_lost             | 0     |
+-------------------------------------------+-------+
```

For information on using these variables to check Performance Schema status, see Section 26.7, “Performance Schema Status Monitoring”.

Performance Schema status variables have the following meanings:

- `Performance_schema_accounts_lost`

  The number of times a row could not be added to the `accounts` table because it was full.

- `Performance_schema_cond_classes_lost`

  How many condition instruments could not be loaded.

- `Performance_schema_cond_instances_lost`

  How many condition instrument instances could not be created.

- `Performance_schema_digest_lost`

  The number of digest instances that could not be instrumented in the `events_statements_summary_by_digest` table. This can be nonzero if the value of `performance_schema_digests_size` is too small.

- `Performance_schema_file_classes_lost`

  How many file instruments could not be loaded.

- `Performance_schema_file_handles_lost`

  How many file instrument instances could not be opened.

- `Performance_schema_file_instances_lost`

  How many file instrument instances could not be created.

- `Performance_schema_hosts_lost`

  The number of times a row could not be added to the `hosts` table because it was full.

- `Performance_schema_index_stat_lost`

  The number of indexes for which statistics were lost. This can be nonzero if the value of `performance_schema_max_index_stat` is too small.

- `Performance_schema_locker_lost`

  How many events are “lost” or not recorded, due to the following conditions:

  - Events are recursive (for example, waiting for A caused a wait on B, which caused a wait on C).
  - The depth of the nested events stack is greater than the limit imposed by the implementation.

  Events recorded by the Performance Schema are not recursive, so this variable should always be 0.

- `Performance_schema_memory_classes_lost`

  The number of times a memory instrument could not be loaded.

- `Performance_schema_metadata_lock_lost`

  The number of metadata locks that could not be instrumented in the `metadata_locks` table. This can be nonzero if the value of `performance_schema_max_metadata_locks` is too small.

- `Performance_schema_mutex_classes_lost`

  How many mutex instruments could not be loaded.

- `Performance_schema_mutex_instances_lost`

  How many mutex instrument instances could not be created.

- `Performance_schema_nested_statement_lost`

  The number of stored program statements for which statistics were lost. This can be nonzero if the value of `performance_schema_max_statement_stack` is too small.

- `Performance_schema_prepared_statements_lost`

The number of prepared statements that could not be instrumented in the `prepared_statements_instances` table. This can be nonzero if the value of `performance_schema_max_prepared_statements_instances` is too small.

- `Performance_schema_program_lost`

  The number of stored programs for which statistics were lost. This can be nonzero if the value of `performance_schema_max_program_instances` is too small.

- `Performance_schema_rwlock_classes_lost`

  How many rwlock instruments could not be loaded.

- `Performance_schema_rwlock_instances_lost`

  How many rwlock instrument instances could not be created.

- `Performance_schema_session_connect_attrs_longest_seen`

  In addition to the connection attribute size-limit check performed by the Performance Schema against the value of the `performance_schema_session_connect_attrs_size` system variable, the server performs a preliminary check, imposing a limit of 64KB on the aggregate size of connection attribute data it will accept. If a client attempts to send more than 64KB of attribute data, the server rejects the connection. Otherwise, the server considers the attribute buffer valid and tracks the size of the longest such buffer in the `Performance_schema_session_connect_attrs_longest_seen` status variable. If this value is larger than `performance_schema_session_connect_attrs_size`, DBAs may wish to increase the latter value, or, alternatively, investigate which clients are sending large amounts of attribute data.

  For more information about connection attributes, see Section 26.12.9, “Performance Schema Connection Attribute Tables”.

- `Performance_schema_session_connect_attrs_lost`

  The number of connections for which connection attribute truncation has occurred. For a given connection, if the client sends connection attribute key-value pairs for which the aggregate size is larger than the reserved storage permitted by the value of the `performance_schema_session_connect_attrs_size` system variable, the Performance Schema truncates the attribute data and increments `Performance_schema_session_connect_attrs_lost`. If this value is nonzero, you may wish to set `performance_schema_session_connect_attrs_size` to a larger value.

  For more information about connection attributes, see Section 26.12.9, “Performance Schema Connection Attribute Tables”.

- `Performance_schema_socket_classes_lost`

  How many socket instruments could not be loaded.

- `Performance_schema_socket_instances_lost`

  How many socket instrument instances could not be created.

- `Performance_schema_stage_classes_lost`

  How many stage instruments could not be loaded.

- `Performance_schema_statement_classes_lost`

  How many statement instruments could not be loaded.

- `Performance_schema_table_handles_lost`

  How many table instrument instances could not be opened. This can be nonzero if the value of `performance_schema_max_table_handles` is too small.

- `Performance_schema_table_instances_lost`

  How many table instrument instances could not be created.

- `Performance_schema_table_lock_stat_lost`

  The number of tables for which lock statistics were lost. This can be nonzero if the value of `performance_schema_max_table_lock_stat` is too small.

- `Performance_schema_thread_classes_lost`

  How many thread instruments could not be loaded.

- `Performance_schema_thread_instances_lost`

  The number of thread instances that could not be instrumented in the `threads` table. This can be nonzero if the value of `performance_schema_max_thread_instances` is too small.

- `Performance_schema_users_lost`

  The number of times a row could not be added to the `users` table because it was full.

26.17 The Performance Schema Memory-Allocation Model

The Performance Schema uses this memory allocation model:

- May allocate memory at server startup
- May allocate additional memory during server operation
- Never free memory during server operation (although it might be recycled)
- Free all memory used at shutdown

The result is to relax memory constraints so that the Performance Schema can be used with less configuration, and to decrease the memory footprint so that consumption scales with server load. Memory used depends on the load actually seen, not the load estimated or explicitly configured for.

Several Performance Schema sizing parameters are autoscaled and need not be configured explicitly unless you want to establish an explicit limit on memory allocation:

```
performance_schema_accounts_size
performance_schema_hosts_size
performance_schema_max_cond_instances
performance_schema_max_file_instances
performance_schema_max_index_stat
performance_schema_max_metadata_locks
performance_schema_max_mutex_instances
performance_schema_max_prepared_statements_instances
performance_schema_max_program_instances
```

```
performance_schema_max_rwlock_instances
performance_schema_max_socket_instances
performance_schema_max_table_handles
performance_schema_max_table_instances
performance_schema_max_table_lock_stat
performance_schema_max_thread_instances
performance_schema_users_size
```

For an autoscaled parameter, configuration works like this:

- With the value set to -1 (the default), the parameter is autoscaled:
  - The corresponding internal buffer is empty initially and no memory is allocated.
  - As the Performance Schema collects data, memory is allocated in the corresponding buffer. The buffer size is unbounded, and may grow with the load.
- With the value set to 0:
  - The corresponding internal buffer is empty initially and no memory is allocated.
- With the value set to *N* > 0:
  - The corresponding internal buffer is empty initially and no memory is allocated.
  - As the Performance Schema collects data, memory is allocated in the corresponding buffer, until the buffer size reaches *N*.
  - Once the buffer size reaches *N*, no more memory is allocated. Data collected by the Performance Schema for this buffer is lost, and any corresponding “lost instance” counters are incremented.

To see how much memory the Performance Schema is using, check the instruments designed for that purpose. The Performance Schema allocates memory internally and associates each buffer with a dedicated instrument so that memory consumption can be traced to individual buffers. Instruments named with the prefix `memory/performance_schema/` expose how much memory is allocated for these internal buffers. The buffers are global to the server, so the instruments are displayed only in the `memory_summary_global_by_event_name` table, and not in other `memory_summary_by_xxx_by_event_name` tables.

This query shows the information associated with the memory instruments:

```
SELECT * FROM performance_schema.memory_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'memory/performance_schema/%';
```

26.18 Performance Schema and Plugins

Removing a plugin with `UNINSTALL PLUGIN` does not affect information already collected for code in that plugin. Time spent executing the code while the plugin was loaded was still spent even if the plugin is unloaded later. The associated event information, including aggregate information, remains readable in `performance_schema` database tables. For additional information about the effect of plugin installation and removal, see Section 26.7, “Performance Schema Status Monitoring”.

A plugin implementor who instruments plugin code should document its instrumentation characteristics to enable those who load the plugin to account for its requirements. For example, a third-party storage engine should include in its documentation how much memory the engine needs for mutex and other instruments.

26.19 Using the Performance Schema to Diagnose Problems

The Performance Schema is a tool to help a DBA do performance tuning by taking real measurements instead of “wild guesses.” This section demonstrates some ways to use the Performance Schema for this purpose. The discussion here relies on the use of event filtering, which is described in Section 26.4.2, “Performance Schema Event Filtering”.

The following example provides one methodology that you can use to analyze a repeatable problem, such as investigating a performance bottleneck. To begin, you should have a repeatable use case where performance is deemed “too slow” and needs optimization, and you should enable all instrumentation (no pre-filtering at all).

1. Run the use case.
2. Using the Performance Schema tables, analyze the root cause of the performance problem. This analysis will rely heavily on post-filtering.
3. For problem areas that are ruled out, disable the corresponding instruments. For example, if analysis shows that the issue is not related to file I/O in a particular storage engine, disable the file I/O instruments for that engine. Then truncate the history and summary tables to remove previously collected events.
4. Repeat the process at step 1.

   At each iteration, the Performance Schema output, particularly the `events_waits_history_long` table, will contain less and less “noise” caused by nonsignificant instruments, and given that this table has a fixed size, will contain more and more data relevant to the analysis of the problem at hand.

   At each iteration, investigation should lead closer and closer to the root cause of the problem, as the “signal/noise” ratio will improve, making analysis easier.
5. Once a root cause of performance bottleneck is identified, take the appropriate corrective action, such as:
   - Tune the server parameters (cache sizes, memory, and so forth).
   - Tune a query by writing it differently,
   - Tune the database schema (tables, indexes, and so forth).
   - Tune the code (this applies to storage engine or server developers only).
6. Start again at step 1, to see the effects of the changes on performance.

The `mutex_instances.LOCKED_BY_THREAD_ID` and `rwlock_instances.WRITE_LOCKED_BY_THREAD_ID` columns are extremely important for investigating performance bottlenecks or deadlocks. This is made possible by Performance Schema instrumentation as follows:

1. Suppose that thread 1 is stuck waiting for a mutex.
2. You can determine what the thread is waiting for:

   ```
   SELECT * FROM performance_schema.events_waits_current
   WHERE THREAD_ID = thread_1;
   ```

   Say the query result identifies that the thread is waiting for mutex A, found in `events_waits_current.OBJECT_INSTANCE_BEGIN`.

3. You can determine which thread is holding mutex A:

```
SELECT * FROM performance_schema.mutex_instances
WHERE OBJECT_INSTANCE_BEGIN = mutex_A;
```

   Say the query result identifies that it is thread 2 holding mutex A, as found in `mutex_instances.LOCKED_BY_THREAD_ID`.

4. You can see what thread 2 is doing:

```
SELECT * FROM performance_schema.events_waits_current
WHERE THREAD_ID = thread_2;
```

26.19.1 Query Profiling Using Performance Schema

The following example demonstrates how to use Performance Schema statement events and stage events to retrieve data comparable to profiling information provided by `SHOW PROFILES` and `SHOW PROFILE` statements.

The `setup_actors` table can be used to limit the collection of historical events by host, user, or account to reduce runtime overhead and the amount of data collected in history tables. The first step of the example shows how to limit collection of historical events to a specific user.

Performance Schema displays event timer information in picoseconds (trillionths of a second) to normalize timing data to a standard unit. In the following example, `TIMER_WAIT` values are divided by 1000000000000 to show data in units of seconds. Values are also truncated to 6 decimal places to display data in the same format as `SHOW PROFILES` and `SHOW PROFILE` statements.

1. Limit the collection of historical events to the user that will run the query. By default, `setup_actors` is configured to allow monitoring and historical event collection for all foreground threads:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+------+------+------+---------+---------+
| HOST | USER | ROLE | ENABLED | HISTORY |
+------+------+------+---------+---------+
| %    | %    | %    | YES     | YES     |
+------+------+------+---------+---------+
```

   Update the default row in the `setup_actors` table to disable historical event collection and monitoring for all foreground threads, and insert a new row that enables monitoring and historical event collection for the user that will run the query:

```
mysql> UPDATE performance_schema.setup_actors
       SET ENABLED = 'NO', HISTORY = 'NO'
       WHERE HOST = '%' AND USER = '%';

mysql> INSERT INTO performance_schema.setup_actors
       (HOST,USER,ROLE,ENABLED,HISTORY)
       VALUES('localhost','test_user','%','YES','YES');
```

   Data in the `setup_actors` table should now appear similar to the following:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+-----------+-----------+------+---------+---------+
| HOST      | USER      | ROLE | ENABLED | HISTORY |
+-----------+-----------+------+---------+---------+
```

```
| %         | %         | %    | NO      | NO      |
| localhost | test_user | %    | YES     | YES     |
+-----------+-----------+------+---------+---------+
```

2. Ensure that statement and stage instrumentation is enabled by updating the `setup_instruments` table. Some instruments may already be enabled by default.

```
mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'YES', TIMED = 'YES'
       WHERE NAME LIKE '%statement/%';

mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'YES', TIMED = 'YES'
       WHERE NAME LIKE '%stage/%';
```

3. Ensure that `events_statements_*` and `events_stages_*` consumers are enabled. Some consumers may already be enabled by default.

```
mysql> UPDATE performance_schema.setup_consumers
       SET ENABLED = 'YES'
       WHERE NAME LIKE '%events_statements_%';

mysql> UPDATE performance_schema.setup_consumers
       SET ENABLED = 'YES'
       WHERE NAME LIKE '%events_stages_%';
```

4. Under the user account you are monitoring, run the statement that you want to profile. For example:

```
mysql> SELECT * FROM employees.employees WHERE emp_no = 10001;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date |
+--------+------------+------------+-----------+--------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
+--------+------------+------------+-----------+--------+------------+
```

5. Identify the `EVENT_ID` of the statement by querying the `events_statements_history_long` table. This step is similar to running `SHOW PROFILES` to identify the `Query_ID`. The following query produces output similar to `SHOW PROFILES`:

```
mysql> SELECT EVENT_ID, TRUNCATE(TIMER_WAIT/1000000000000,6) as Duration, SQL_TEXT
       FROM performance_schema.events_statements_history_long WHERE SQL_TEXT like '%10001%';
+----------+----------+-----------------------------------------------------------+
| event_id | duration | sql_text                                                  |
+----------+----------+-----------------------------------------------------------+
|       31 | 0.028310 | SELECT * FROM employees.employees WHERE emp_no = 10001 |
+----------+----------+-----------------------------------------------------------+
```

6. Query the `events_stages_history_long` table to retrieve the statement's stage events. Stages are linked to statements using event nesting. Each stage event record has a `NESTING_EVENT_ID` column that contains the `EVENT_ID` of the parent statement.

```
mysql> SELECT event_name AS Stage, TRUNCATE(TIMER_WAIT/1000000000000,6) AS Duration
       FROM performance_schema.events_stages_history_long WHERE NESTING_EVENT_ID=31;
+------------------------------+----------+
| Stage                        | Duration |
+------------------------------+----------+
| stage/sql/starting           | 0.000080 |
| stage/sql/checking permissions | 0.000005 |
| stage/sql/Opening tables     | 0.027759 |
| stage/sql/init               | 0.000052 |
```

```
| stage/sql/System lock          | 0.000009 |
| stage/sql/optimizing           | 0.000006 |
| stage/sql/statistics           | 0.000082 |
| stage/sql/preparing            | 0.000008 |
| stage/sql/executing            | 0.000000 |
| stage/sql/Sending data         | 0.000017 |
| stage/sql/end                  | 0.000001 |
| stage/sql/query end            | 0.000004 |
| stage/sql/closing tables       | 0.000006 |
| stage/sql/freeing items        | 0.000272 |
| stage/sql/cleaning up          | 0.000001 |
+--------------------------------+----------+
```

26.19.2 Obtaining Parent Event Information

The `data_locks` table shows data locks held and requested. Rows of this table have a `THREAD_ID` column indicating the thread ID of the session that owns the lock, and an `EVENT_ID` column indicating the Performance Schema event that caused the lock. Tuples of (`THREAD_ID`, `EVENT_ID`) values implicitly identify a parent event in other Performance Schema tables:

- The parent wait event in the `events_waits_xxx` tables
- The parent stage event in the `events_stages_xxx` tables
- The parent statement event in the `events_statements_xxx` tables
- The parent transaction event in the `events_transactions_current` table

To obtain details about the parent event, join the `THREAD_ID` and `EVENT_ID` columns with the columns of like name in the appropriate parent event table. The relation is based on a nested set data model, so the join has several clauses. Given parent and child tables represented by `parent` and `child`, respectively, the join looks like this:

```
WHERE
  parent.THREAD_ID = child.THREAD_ID        /* 1 */
  AND parent.EVENT_ID < child.EVENT_ID      /* 2 */
  AND (
    child.EVENT_ID <= parent.END_EVENT_ID   /* 3a */
    OR parent.END_EVENT_ID IS NULL          /* 3b */
  )
```

The condititions for the join are:

1. The parent and child events are in the same thread.
2. The child event begins after the parent event, so its `EVENT_ID` value is greater than that of the parent.
3. The parent event has either completed or is still running.

To find lock information, `data_locks` is the table containing child events.

The `data_locks` table shows only existing locks, so these considerations apply regarding which table contains the parent event:

- For transactions, the only choice is `events_transactions_current`. If a transaction is completed, it may be in the transaction history tables, but the locks are gone already.
- For statements, it all depends on whether the statement that took a lock is a statement in a transaction that has already completed (use `events_statements_history`) or the statement is still running (use `events_statements_current`).

- For stages, the logic is similar to that for statements; use `events_stages_history` or `events_stages_current`.
- For waits, the logic is similar to that for statements; use `events_waits_history` or `events_waits_current`. However, so many waits are recorded that the wait that caused a lock is most likely gone from the history tables already.

Wait, stage, and statement events disappear quickly from the history. If a statement that executed a long time ago took a lock but is in a still-open transaction, it might not be possible to find the statement, but it is possible to find the transaction.

This is why the nested set data model works better for locating parent events. Following links in a parent/child relationship (data lock -> parent wait -> parent stage -> parent transaction) does not work well when intermediate nodes are already gone from the history tables.

The following scenario illustrates how to find the parent transaction of a statement in which a lock was taken:

Session A:

```
[1] START TRANSACTION;
[2] SELECT * FROM t1 WHERE pk = 1;
[3] SELECT 'Hello, world';
```

Session B:

```
SELECT ...
FROM performance_schema.events_transactions_current AS parent
  INNER JOIN performance_schema.data_locks AS child
WHERE
  parent.THREAD_ID = child.THREAD_ID
  AND parent.EVENT_ID < child.EVENT_ID
  AND (
    child.EVENT_ID <= parent.END_EVENT_ID
    OR parent.END_EVENT_ID IS NULL
  );
```

The query for session B should show statement [2] as owning a data lock on the record with `pk=1`.

If session A executes more statements, [2] fades out of the history table.

The query should show the transaction that started in [1], regardless of how many statements, stages, or waits were executed.

To see more data, you can also use the `events_xxx_history_long` tables, except for transactions, assuming no other query runs in the server (so that history is preserved).

26.20 Restrictions on Performance Schema

The Performance Schema avoids using mutexes to collect or produce data, so there are no guarantees of consistency and results can sometimes be incorrect. Event values in `performance_schema` tables are nondeterministic and nonrepeatable.

If you save event information in another table, you should not assume that the original events will still be available later. For example, if you select events from a `performance_schema` table into a temporary table, intending to join that table with the original table later, there might be no matches.

`mysqldump` and `BACKUP DATABASE` ignore tables in the `performance_schema` database.

Tables in the `performance_schema` database cannot be locked with `LOCK TABLES`, except the `setup_xxx` tables.

Tables in the `performance_schema` database cannot be indexed.

Tables in the `performance_schema` database are not replicated.

The types of timers might vary per platform. The `performance_timers` table shows which event timers are available. If the values in this table for a given timer name are `NULL`, that timer is not supported on your platform.

Instruments that apply to storage engines might not be implemented for all storage engines. Instrumentation of each third-party engine is the responsibility of the engine maintainer.

Chapter 27 MySQL sys Schema

Table of Contents

27.1 Prerequisites for Using the sys Schema 4695
27.2 Using the sys Schema 4696
27.3 sys Schema Progress Reporting 4697
27.4 sys Schema Object Reference 4698
27.4.1 sys Schema Object Index 4698
27.4.2 sys Schema Tables and Triggers 4702
27.4.3 sys Schema Views 4705
27.4.4 sys Schema Stored Procedures 4747
27.4.5 sys Schema Stored Functions 4767

MySQL 8.0 includes the `sys` schema, a set of objects that helps DBAs and developers interpret data collected by the Performance Schema. `sys` schema objects can be used for typical tuning and diagnosis use cases. Objects in this schema include:

- Views that summarize Performance Schema data into more easily understandable form.
- Stored procedures that perform operations such as Performance Schema configuration and generating diagnostic reports.
- Stored functions that query Performance Schema configuration and provide formatting services.

For new installations, the `sys` schema is installed by default during data directory initialization if you use `mysqld` with the `--initialize` or `--initialize-insecure` option. If this is not desired, you can drop the `sys` schema manually after initialization if it is unneeded.

The MySQL upgrade procedure produces an error if a `sys` schema exists but has no `version` view, on the assumption that absence of this view indicates a user-created `sys` schema. To upgrade in this case, remove or rename the existing `sys` schema first.

`sys` schema objects have a `DEFINER` of `'mysql.sys'@'localhost'`. Use of the dedicated `mysql.sys` account avoids problems that occur if a DBA renames or removes the `root` account.

27.1 Prerequisites for Using the sys Schema

Before using the `sys` schema, the prerequisites described in this section must be satisfied.

Because the `sys` schema provides an alternative means of accessing the Performance Schema, the Performance Schema must be enabled for the `sys` schema to work. See Section 26.3, “Performance Schema Startup Configuration”.

For full access to the `sys` schema, a user must have these privileges:

- `SELECT` on all `sys` tables and views
- `EXECUTE` on all `sys` stored procedures and functions
- `INSERT` and `UPDATE` for the `sys_config` table, if changes are to be made to it
- Additional privileges for certain `sys` schema stored procedures and functions, as noted in their descriptions (for example, the `ps_setup_save()` procedure)

It is also necessary to have privileges for the objects underlying the `sys` schema objects:

- `SELECT` on any Performance Schema tables accessed by `sys` schema objects, and `UPDATE` for any tables to be updated using `sys` schema objects
- `PROCESS` for the `INFORMATION_SCHEMA INNODB_BUFFER_PAGE` table

Certain Performance Schema instruments and consumers must be enabled and (for instruments) timed to take full advantage of `sys` schema capabilities:

- All `wait` instruments
- All `stage` instruments
- All `statement` instruments
- *`xxx`*`_current` and *`xxx`*`_history_long` consumers for all events

You can use the `sys` schema itself to enable all of the additional instruments and consumers:

```
CALL sys.ps_setup_enable_instrument('wait');
CALL sys.ps_setup_enable_instrument('stage');
CALL sys.ps_setup_enable_instrument('statement');
CALL sys.ps_setup_enable_consumer('current');
CALL sys.ps_setup_enable_consumer('history_long');
```

Note

For many uses of the `sys` schema, the default Performance Schema is sufficient for data collection. Enabling all the instruments and consumers just mentioned has a performance impact, so it is preferable to enable only the additional configuration you need. Also, remember that if you enable additional configuration, you can easily restore the default configuration like this:

```
CALL sys.ps_setup_reset_to_default(TRUE);
```

27.2 Using the sys Schema

You can make the `sys` schema the default schema so that references to its objects need not be qualified with the schema name:

```
mysql> USE sys;
Database changed
mysql> SELECT * FROM version;
+-------------+---------------+
| sys_version | mysql_version |
+-------------+---------------+
| 2.0.0       | 8.0.13-debug  |
+-------------+---------------+
```

(The `version` view shows the `sys` schema and MySQL server versions.)

To access `sys` schema objects while a different schema is the default (or simply to be explicit), qualify object references with the schema name:

```
mysql> SELECT * FROM sys.version;
+-------------+---------------+
| sys_version | mysql_version |
+-------------+---------------+
| 2.0.0       | 8.0.13-debug  |
+-------------+---------------+
```

The sys schema contains many views that summarize Performance Schema tables in various ways. Most of these views come in pairs, such that one member of the pair has the same name as the other member, plus a x$ prefix. For example, the host_summary_by_file_io view summarizes file I/O grouped by host and displays latencies converted from picoseconds to more readable values (with units);

```
mysql> SELECT * FROM sys.host_summary_by_file_io;
+------------+-------+------------+
| host       | ios   | io_latency |
+------------+-------+------------+
| localhost  | 67570 | 5.38 s     |
| background |  3468 | 4.18 s     |
+------------+-------+------------+
```

The x$host_summary_by_file_io view summarizes the same data but displays unformatted picosecond latencies:

```
mysql> SELECT * FROM sys.x$host_summary_by_file_io;
+------------+-------+---------------+
| host       | ios   | io_latency    |
+------------+-------+---------------+
| localhost  | 67574 | 5380678125144 |
| background |  3474 | 4758696829416 |
+------------+-------+---------------+
```

The view without the x$ prefix is intended to provide output that is more user friendly and easier for humans to read. The view with the x$ prefix that displays the same values in raw form is intended more for use with other tools that perform their own processing on the data. For additional information about the differences between non-x$ and x$ views, see Section 27.4.3, “sys Schema Views”.

To examine sys schema object definitions, use the appropriate SHOW statement or INFORMATION_SCHEMA query. For example, to examine the definitions of the session view and format_bytes() function, use these statements:

```
mysql> SHOW CREATE VIEW sys.session;
mysql> SHOW CREATE FUNCTION sys.format_bytes;
```

However, those statements display the definitions in relatively unformatted form. To view object definitions with more readable formatting, access the individual .sql files found under the scripts/sys_schema in MySQL source distributions. Prior to MySQL 8.0.18, the sources are maintained in a separate distribution available from the sys schema development website at https://github.com/mysql/mysql-sys.

Neither mysqldump nor mysqlpump dump the sys schema by default. To generate a dump file, name the sys schema explicitly on the command line using either of these commands:

```
mysqldump --databases --routines sys > sys_dump.sql
mysqlpump sys > sys_dump.sql
```

To reinstall the schema from the dump file, use this command:

```
mysql < sys_dump.sql
```

27.3 sys Schema Progress Reporting

The following sys schema views provide progress reporting for long-running transactions:

```
processlist
```

```
session
x$processlist
x$session
```

Assuming that the required instruments and consumers are enabled, the `progress` column of these views shows the percentage of work completed for stages that support progress reporting.

Stage progress reporting requires that the `events_stages_current` consumer be enabled, as well as the instruments for which progress information is desired. Instruments for these stages currently support progress reporting:

```
stage/sql/Copying to tmp table
stage/innodb/alter table (end)
stage/innodb/alter table (flush)
stage/innodb/alter table (insert)
stage/innodb/alter table (log apply index)
stage/innodb/alter table (log apply table)
stage/innodb/alter table (merge sort)
stage/innodb/alter table (read PK and internal sort)
stage/innodb/buffer pool load
```

For stages that do not support estimated and completed work reporting, or if the required instruments or consumers are not enabled, the `progress` column is `NULL`.

27.4 sys Schema Object Reference

The `sys` schema includes tables and triggers, views, and stored procedures and functions. The following sections provide details for each of these objects.

27.4.1 sys Schema Object Index

The following tables list `sys` schema objects and provide a short description of each one.

Table 27.1 sys Schema Tables and Triggers

Table or Trigger Name	Description
`sys_config`	`sys` schema configuration options
`sys_config_insert_set_user`	`sys_config` insert trigger
`sys_config_update_set_user`	`sys_config` update trigger

Table 27.2 sys Schema Views

View Name	Description
`host_summary`, `x$host_summary`	Statement activity, file I/O, and connections, grouped by host
`host_summary_by_file_io`, `x$host_summary_by_file_io`	File I/O, grouped by host
`host_summary_by_file_io_type`, `x$host_summary_by_file_io_type`	File I/O, grouped by host and event type
`host_summary_by_stages`, `x$host_summary_by_stages`	Statement stages, grouped by host
`host_summary_by_statement_latency`, `x$host_summary_by_statement_latency`	Statement statistics, grouped by host
`host_summary_by_statement_type`, `x$host_summary_by_statement_type`	Statements executed, grouped by host and statement

View Name	Description
innodb_buffer_stats_by_schema, x$innodb_buffer_stats_by_schema	InnoDB buffer information, grouped by schema
innodb_buffer_stats_by_table, x$innodb_buffer_stats_by_table	InnoDB buffer information, grouped by schema and table
innodb_lock_waits, x$innodb_lock_waits	InnoDB lock information
io_by_thread_by_latency, x$io_by_thread_by_latency	I/O consumers, grouped by thread
io_global_by_file_by_bytes, x$io_global_by_file_by_bytes	Global I/O consumers, grouped by file and bytes
io_global_by_file_by_latency, x$io_global_by_file_by_latency	Global I/O consumers, grouped by file and latency
io_global_by_wait_by_bytes, x$io_global_by_wait_by_bytes	Global I/O consumers, grouped by bytes
io_global_by_wait_by_latency, x$io_global_by_wait_by_latency	Global I/O consumers, grouped by latency
latest_file_io, x$latest_file_io	Most recent I/O, grouped by file and thread
memory_by_host_by_current_bytes, x$memory_by_host_by_current_bytes	Memory use, grouped by host
memory_by_thread_by_current_bytes, x$memory_by_thread_by_current_bytes	Memory use, grouped by thread
memory_by_user_by_current_bytes, x$memory_by_user_by_current_bytes	Memory use, grouped by user
memory_global_by_current_bytes, x$memory_global_by_current_bytes	Memory use, grouped by allocation type
memory_global_total, x$memory_global_total	Total memory use
metrics	Server metrics
processlist, x$processlist	Processlist information
ps_check_lost_instrumentation	Variables that have lost instruments
schema_auto_increment_columns	AUTO_INCREMENT column information
schema_index_statistics, x$schema_index_statistics	Index statistics
schema_object_overview	Types of objects within each schema
schema_redundant_indexes	Duplicate or redundant indexes
schema_table_lock_waits, x$schema_table_lock_waits	Sessions waiting for metadata locks
schema_table_statistics, x$schema_table_statistics	Table statistics
schema_table_statistics_with_buffer, x$schema_table_statistics_with_buffer	Table statistics, including InnoDB buffer pool statistics
schema_tables_with_full_table_scans, x$schema_tables_with_full_table_scans	Tables being accessed with full scans

View Name	Description
schema_unused_indexes	Indexes not in active use
session, x$session	Processlist information for user sessions
session_ssl_status	Connection SSL information
statement_analysis, x$statement_analysis	Statement aggregate statistics
statements_with_errors_or_warnings, x$statements_with_errors_or_warnings	Statements that have produced errors or warnings
statements_with_full_table_scans, x$statements_with_full_table_scans	Statements that have done full table scans
statements_with_runtimes_in_95th_percentile, x$statements_with_runtimes_in_95th_percentile	Statements with highest average runtime
statements_with_sorting, x$statements_with_sorting	Statements that performed sorts
statements_with_temp_tables, x$statements_with_temp_tables	Statements that used temporary tables
user_summary, x$user_summary	User statement and connection activity
user_summary_by_file_io, x$user_summary_by_file_io	File I/O, grouped by user
user_summary_by_file_io_type, x$user_summary_by_file_io_type	File I/O, grouped by user and event
user_summary_by_stages, x$user_summary_by_stages	Stage events, grouped by user
user_summary_by_statement_latency, x$user_summary_by_statement_latency	Statement statistics, grouped by user
user_summary_by_statement_type, x$user_summary_by_statement_type	Statements executed, grouped by user and statement
version	Current sys schema and MySQL server versions
wait_classes_global_by_avg_latency, x$wait_classes_global_by_avg_latency	Wait class average latency, grouped by event class
wait_classes_global_by_latency, x$wait_classes_global_by_latency	Wait class total latency, grouped by event class
waits_by_host_by_latency, x$waits_by_host_by_latency	Wait events, grouped by host and event
waits_by_user_by_latency, x$waits_by_user_by_latency	Wait events, grouped by user and event
waits_global_by_latency, x$waits_global_by_latency	Wait events, grouped by event
x$ps_digest_95th_percentile_by_avg_us	Helper view for 95th-percentile views
x$ps_digest_avg_latency_distribution	Helper view for 95th-percentile views
x$ps_schema_table_statistics_io	Helper view for table-statistics views
x$schema_flattened_keys	Helper view for schema_redundant_indexes

Table 27.3 sys Schema Stored Procedures

Procedure Name	Description
create_synonym_db()	Create synonym for schema
diagnostics()	Collect system diagnostic information
execute_prepared_stmt()	Execute prepared statement
ps_setup_disable_background_threads()	Disable background thread instrumentation
ps_setup_disable_consumer()	Disable consumers
ps_setup_disable_instrument()	Disable instruments
ps_setup_disable_thread()	Disable instrumentation for thread
ps_setup_enable_background_threads()	Enable background thread instrumentation
ps_setup_enable_consumer()	Enable consumers
ps_setup_enable_instrument()	Enable instruments
ps_setup_enable_thread()	Enable instrumentation for thread
ps_setup_reload_saved()	Reload saved Performance Schema configuration
ps_setup_reset_to_default()	Reset saved Performance Schema configuration
ps_setup_save()	Save Performance Schema configuration
ps_setup_show_disabled()	Display disabled Performance Schema configuration
ps_setup_show_disabled_consumers()	Display disabled Performance Schema consumers
ps_setup_show_disabled_instruments()	Display disabled Performance Schema instruments
ps_setup_show_enabled()	Display enabled Performance Schema configuration
ps_setup_show_enabled_consumers()	Display enabled Performance Schema consumers
ps_setup_show_enabled_instruments()	Display enabled Performance Schema instruments
ps_statement_avg_latency_histogram()	Display statement latency histogram
ps_trace_statement_digest()	Trace Performance Schema instrumentation for digest
ps_trace_thread()	Dump Performance Schema data for thread
ps_truncate_all_tables()	Truncate Performance Schema summary tables
statement_performance_analyzer()	Report of statements running on server
table_exists()	Whether a table exists

Table 27.4 sys Schema Stored Functions

Function Name	Description
`extract_schema_from_file_name()`	Extract schema name from file path name
`extract_table_from_file_name()`	Extract table name from file path name
`format_bytes()`	Convert byte count to value with units
`format_path()`	Replace data and temp-file directories in path name with symbolic values
`format_statement()`	Truncate long statement to fixed length
`format_time()`	Convert picoseconds value to value with units
`list_add()`	Add item to list
`list_drop()`	Remove item from list
`ps_is_account_enabled()`	Check whether account instrumentation is enabled
`ps_is_consumer_enabled()`	Check whether consumer is enabled
`ps_is_instrument_default_enabled()`	Check whether instrument is enabled
`ps_is_instrument_default_timed()`	Check whether instrument is timed
`ps_is_thread_instrumented()`	Check whether thread is instrumented
`ps_thread_account()`	Return account for thread ID
`ps_thread_id()`	Return thread ID for connection ID
`ps_thread_stack()`	Return event information for thread ID
`ps_thread_trx_info()`	Return transaction information for thread ID
`quote_identifier()`	Return string as quoted identifier
`sys_get_config()`	Return `sys` schema configuration option
`version_major()`	MySQL server major version number
`version_minor()`	MySQL server minor version number
`version_patch()`	MySQL server patch release version number

27.4.2 sys Schema Tables and Triggers

The following sections describe `sys` schema tables and triggers.

27.4.2.1 The sys_config Table

This table contains `sys` schema configuration options, one row per option. Configuration changes made by updating this table persist across client sessions and server restarts.

The `sys_config` table has these columns:

- `variable`

  The configuration option name.

- `value`

The configuration option value.

- `set_time`

  The timestamp of the most recent modification to the row.

- `set_by`

  The account that made the most recent modification to the row. The value is `NULL` if the row has not been changed since the `sys` schema was installed.

As an efficiency measure to minimize the number of direct reads from the `sys_config` table, `sys` schema functions that use a value from this table check for a user-defined variable with a corresponding name, which is the user-defined variable having the same name plus a `@sys.` prefix. (For example, the variable corresponding to the `diagnostics.include_raw` option is `@sys.diagnostics.include_raw`.) If the user-defined variable exists in the current session and is non-`NULL`, the function uses its value in preference to the value in the `sys_config` table. Otherwise, the function reads and uses the value from the table. In the latter case, the calling function conventionally also sets the corresponding user-defined variable to the table value so that further references to the configuration option within the same session use the variable and need not read the table again.

For example, the `statement_truncate_len` option controls the maximum length of statements returned by the `format_statement()` function. The default is 64. To temporarily change the value to 32 for your current session, set the corresponding `@sys.statement_truncate_len` user-defined variable:

```
mysql> SET @stmt = 'SELECT variable, value, set_time, set_by FROM sys_config';
mysql> SELECT sys.format_statement(@stmt);
+-----------------------------------------------------------+
| sys.format_statement(@stmt)                               |
+-----------------------------------------------------------+
| SELECT variable, value, set_time, set_by FROM sys_config |
+-----------------------------------------------------------+
mysql> SET @sys.statement_truncate_len = 32;
mysql> SELECT sys.format_statement(@stmt);
+----------------------------------+
| sys.format_statement(@stmt)      |
+----------------------------------+
| SELECT variabl ... ROM sys_config |
+----------------------------------+
```

Subsequent invocations of `format_statement()` within the session continue to use the user-defined variable value (32), rather than the value stored in the table (64).

To stop using the user-defined variable and revert to using the value in the table, set the variable to `NULL` within your session:

```
mysql> SET @sys.statement_truncate_len = NULL;
mysql> SELECT sys.format_statement(@stmt);
+-----------------------------------------------------------+
| sys.format_statement(@stmt)                               |
+-----------------------------------------------------------+
| SELECT variable, value, set_time, set_by FROM sys_config |
+-----------------------------------------------------------+
```

Alternatively, end your current session (causing the user-defined variable to no longer exist) and begin a new session.

The conventional relationship just described between options in the `sys_config` table and user-defined variables can be exploited to make temporary configuration changes that end when your session ends.

However, if you set a user-defined variable and then subsequently change the corresponding table value within the same session, the changed table value will not be used in that session as long as the user-defined variable exists with a non-`NULL` value. (The changed table value *will* be used in other sessions that do not have the user-defined variable assigned.)

The following list describes the options in the `sys_config` table and the corresponding user-defined variables:

- `diagnostics.allow_i_s_tables`, `@sys.diagnostics.allow_i_s_tables`

  If this option is `ON`, the `diagnostics()` procedure is permitted to perform table scans on the `INFORMATION_SCHEMA.TABLES` table. This can be expensive if there are many tables. The default is `OFF`.

- `diagnostics.include_raw`, `@sys.diagnostics.include_raw`

  If this option is `ON`, the `diagnostics()` procedure includes the raw output from querying the `metrics` view. The default is `OFF`.

- `ps_thread_trx_info.max_length`, `@sys.ps_thread_trx_info.max_length`

  The maximum length for JSON output produced by the `ps_thread_trx_info()` function. The default is 65535.

- `statement_performance_analyzer.limit`, `@sys.statement_performance_analyzer.limit`

  The maximum number of rows to return for views that have no built-in limit. (For example, the `statements_with_runtimes_in_95th_percentile` view has a built-in limit in the sense that it returns only statements with average execution time in the 95th percentile.) The default is 100.

- `statement_performance_analyzer.view`, `@sys.statement_performance_analyzer.view`

  The custom query or view to be used by the `statement_performance_analyzer()` procedure (which is itself invoked by the `diagnostics()` procedure). If the option value contains a space, it is interpreted as a query. Otherwise, it must be the name of an existing view that queries the Performance Schema `events_statements_summary_by_digest` table. There cannot be any `LIMIT` clause in the query or view definition if the `statement_performance_analyzer.limit` configuration option is greater than 0. The default is `NULL` (no custom view defined).

- `statement_truncate_len`, `@sys.statement_truncate_len`

  The maximum length of statements returned by the `format_statement()` function. Longer statements are truncated to this length. The default is 64.

Other options can be added to the `sys_config` table. For example, the `diagnostics()` and `execute_prepared_stmt()` procedures use the `debug` option if it exists, but this option is not part of the `sys_config` table by default because debug output normally is enabled only temporarily, by setting the corresponding `@sys.debug` user-defined variable. To enable debug output without having to set that variable in individual sessions, add the option to the table:

```
mysql> INSERT INTO sys.sys_config (variable, value) VALUES('debug', 'ON');
```

To change the debug setting in the table, do two things. First, modify the value in the table itself:

```
mysql> UPDATE sys.sys_config
```

```
      SET value = 'OFF'
      WHERE variable = 'debug';
```

Second, to also ensure that procedure invocations within the current session use the changed value from the table, set the corresponding user-defined variable to `NULL`:

```
mysql> SET @sys.debug = NULL;
```

27.4.2.2 The sys_config_insert_set_user Trigger

For rows added to the `sys_config` table by `INSERT` statements, the `sys_config_insert_set_user` trigger sets the `set_by` column to the current user.

27.4.2.3 The sys_config_update_set_user Trigger

The `sys_config_update_set_user` trigger for the `sys_config` table is similar to the `sys_config_insert_set_user` trigger, but for `UPDATE` statements.

27.4.3 sys Schema Views

The following sections describe `sys` schema views.

The `sys` schema contains many views that summarize Performance Schema tables in various ways. Most of these views come in pairs, such that one member of the pair has the same name as the other member, plus a `x$` prefix. For example, the `host_summary_by_file_io` view summarizes file I/O grouped by host and displays latencies converted from picoseconds to more readable values (with units);

```
mysql> SELECT * FROM sys.host_summary_by_file_io;
+------------+-------+------------+
| host       | ios   | io_latency |
+------------+-------+------------+
| localhost  | 67570 | 5.38 s     |
| background |  3468 | 4.18 s     |
+------------+-------+------------+
```

The `x$host_summary_by_file_io` view summarizes the same data but displays unformatted picosecond latencies:

```
mysql> SELECT * FROM sys.x$host_summary_by_file_io;
+------------+-------+---------------+
| host       | ios   | io_latency    |
+------------+-------+---------------+
| localhost  | 67574 | 5380678125144 |
| background |  3474 | 4758696829416 |
+------------+-------+---------------+
```

The view without the `x$` prefix is intended to provide output that is more user friendly and easier to read. The view with the `x$` prefix that displays the same values in raw form is intended more for use with other tools that perform their own processing on the data.

Views without the `x$` prefix differ from the corresponding `x$` views in these ways:

- Byte counts are formatted with size units using `format_bytes()`.
- Time values are formatted with temporal units using `format_time()`.
- SQL statements are truncated to a maximum display width using `format_statement()`.

- Path name are shortened using `format_path()`.

27.4.3.1 The host_summary and x$host_summary Views

These views summarize statement activity, file I/O, and connections, grouped by host.

The `host_summary` and `x$host_summary` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `statements`

  The total number of statements for the host.

- `statement_latency`

  The total wait time of timed statements for the host.

- `statement_avg_latency`

  The average wait time per timed statement for the host.

- `table_scans`

  The total number of table scans for the host.

- `file_ios`

  The total number of file I/O events for the host.

- `file_io_latency`

  The total wait time of timed file I/O events for the host.

- `current_connections`

  The current number of connections for the host.

- `total_connections`

  The total number of connections for the host.

- `unique_users`

  The number of distinct users for the host.

- `current_memory`

  The current amount of allocated memory for the host.

- `total_memory_allocated`

  The total amount of allocated memory for the host.

27.4.3.2 The host_summary_by_file_io and x$host_summary_by_file_io Views

These views summarize file I/O, grouped by host. By default, rows are sorted by descending total file I/O latency.

The `host_summary_by_file_io` and `x$host_summary_by_file_io` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `ios`

  The total number of file I/O events for the host.

- `io_latency`

  The total wait time of timed file I/O events for the host.

27.4.3.3 The host_summary_by_file_io_type and x$host_summary_by_file_io_type Views

These views summarize file I/O, grouped by host and event type. By default, rows are sorted by host and descending total I/O latency.

The `host_summary_by_file_io_type` and `x$host_summary_by_file_io_type` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `event_name`

  The file I/O event name.

- `total`

  The total number of occurrences of the file I/O event for the host.

- `total_latency`

  The total wait time of timed occurrences of the file I/O event for the host.

- `max_latency`

  The maximum single wait time of timed occurrences of the file I/O event for the host.

27.4.3.4 The host_summary_by_stages and x$host_summary_by_stages Views

These views summarize statement stages, grouped by host. By default, rows are sorted by host and descending total latency.

The `host_summary_by_stages` and `x$host_summary_by_stages` views have these columns:

- `host`

The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `event_name`

  The stage event name.

- `total`

  The total number of occurrences of the stage event for the host.

- `total_latency`

  The total wait time of timed occurrences of the stage event for the host.

- `avg_latency`

  The average wait time per timed occurrence of the stage event for the host.

27.4.3.5 The host_summary_by_statement_latency and x$host_summary_by_statement_latency Views

These views summarize overall statement statistics, grouped by host. By default, rows are sorted by descending total latency.

The `host_summary_by_statement_latency` and `x$host_summary_by_statement_latency` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `total`

  The total number of statements for the host.

- `total_latency`

  The total wait time of timed statements for the host.

- `max_latency`

  The maximum single wait time of timed statements for the host.

- `lock_latency`

  The total time waiting for locks by timed statements for the host.

- `rows_sent`

  The total number of rows returned by statements for the host.

- `rows_examined`

  The total number of rows read from storage engines by statements for the host.

- `rows_affected`

  The total number of rows affected by statements for the host.

- `full_scans`

  The total number of full table scans by statements for the host.

27.4.3.6 The host_summary_by_statement_type and x$host_summary_by_statement_type Views

These views summarize informaion about statements executed, grouped by host and statement type. By default, rows are sorted by host and descending total latency.

The `host_summary_by_statement_type` and `x$host_summary_by_statement_type` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `statement`

  The final component of the statement event name.

- `total`

  The total number of occurrences of the statement event for the host.

- `total_latency`

  The total wait time of timed occurrences of the statement event for the host.

- `max_latency`

  The maximum single wait time of timed occurrences of the statement event for the host.

- `lock_latency`

  The total time waiting for locks by timed occurrences of the statement event for the host.

- `rows_sent`

  The total number of rows returned by occurrences of the statement event for the host.

- `rows_examined`

  The total number of rows read from storage engines by occurrences of the statement event for the host.

- `rows_affected`

  The total number of rows affected by occurrences of the statement event for the host.

- `full_scans`

  The total number of full table scans by occurrences of the statement event for the host.

27.4.3.7 The innodb_buffer_stats_by_schema and x$innodb_buffer_stats_by_schema Views

These views summarize the information in the `INFORMATION_SCHEMA INNODB_BUFFER_PAGE` table, grouped by schema. By default, rows are sorted by descending buffer size.

Warning

Querying views that access the `INNODB_BUFFER_PAGE` table can affect performance. Do not query these views on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

The `innodb_buffer_stats_by_schema` and `x$innodb_buffer_stats_by_schema` views have these columns:

- `object_schema`

  The schema name for the object, or `InnoDB System` if the table belongs to the `InnoDB` storage engine.

- `allocated`

  The total number of bytes allocated for the schema.

- `data`

  The total number of data bytes allocated for the schema.

- `pages`

  The total number of pages allocated for the schema.

- `pages_hashed`

  The total number of hashed pages allocated for the schema.

- `pages_old`

  The total number of old pages allocated for the schema.

- `rows_cached`

  The total number of cached rows for the schema.

27.4.3.8 The innodb_buffer_stats_by_table and x$innodb_buffer_stats_by_table Views

These views summarize the information in the `INFORMATION_SCHEMA INNODB_BUFFER_PAGE` table, grouped by schema and table. By default, rows are sorted by descending buffer size.

Warning

Querying views that access the `INNODB_BUFFER_PAGE` table can affect performance. Do not query these views on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

The `innodb_buffer_stats_by_table` and `x$innodb_buffer_stats_by_table` views have these columns:

- `object_schema`

  The schema name for the object, or `InnoDB System` if the table belongs to the `InnoDB` storage engine.

- `object_name`

  The table name.

- `allocated`

  The total number of bytes allocated for the table.

- `data`

  The number of data bytes allocated for the table.

- `pages`

  The total number of pages allocated for the table.

- `pages_hashed`

  The number of hashed pages allocated for the table.

- `pages_old`

  The number of old pages allocated for the table.

- `rows_cached`

  The number of cached rows for the table.

27.4.3.9 The innodb_lock_waits and x$innodb_lock_waits Views

These views summarize the `InnoDB` locks that transactions are waiting for. By default, rows are sorted by descending lock age.

The `innodb_lock_waits` and `x$innodb_lock_waits` views have these columns:

- `wait_started`

  The time at which the lock wait started.

- `wait_age`

  How long the lock has been waited for, as a `TIME` value.

- `wait_age_secs`

  How long the lock has been waited for, in seconds.

- `locked_table_schema`

  The schema that contains the locked table.

- `locked_table_name`

The name of the locked table.

- `locked_table_partition`

  The name of the locked partition, if any; `NULL` otherwise.

- `locked_table_subpartition`

  The name of the locked subpartition, if any; `NULL` otherwise.

- `locked_index`

  The name of the locked index.

- `locked_type`

  The type of the waiting lock.

- `waiting_trx_id`

  The ID of the waiting transaction.

- `waiting_trx_started`

  The time at which the waiting transaction started.

- `waiting_trx_age`

  How long the waiting transaction has been waiting, as a `TIME` value.

- `waiting_trx_rows_locked`

  The number of rows locked by the waiting transaction.

- `waiting_trx_rows_modified`

  The number of rows modified by the waiting transaction.

- `waiting_pid`

  The processlist ID of the waiting transaction.

- `waiting_query`

  The statement that is waiting for the lock.

- `waiting_lock_id`

  The ID of the waiting lock.

- `waiting_lock_mode`

  The mode of the waiting lock.

- `blocking_trx_id`

  The ID of the transaction that is blocking the waiting lock.

- `blocking_pid`

The processlist ID of the blocking transaction.

- `blocking_query`

  The statement the blocking transaction is executing. This field reports NULL if the session that issued the blocking query becomes idle. For more information, see Identifying a Blocking Query After the Issuing Session Becomes Idle.

- `blocking_lock_id`

  The ID of the lock that is blocking the waiting lock.

- `blocking_lock_mode`

  The mode of the lock that is blocking the waiting lock.

- `blocking_trx_started`

  The time at which the blocking transaction started.

- `blocking_trx_age`

  How long the blocking transaction has been executing, as a `TIME` value.

- `blocking_trx_rows_locked`

  The number of rows locked by the blocking transaction.

- `blocking_trx_rows_modified`

  The number of rows modified by the blocking transaction.

- `sql_kill_blocking_query`

  The `KILL` statement to execute to kill the blocking statement.

- `sql_kill_blocking_connection`

  The `KILL` statement to execute to kill the session running the blocking statement.

27.4.3.10 The io_by_thread_by_latency and x$io_by_thread_by_latency Views

These views summarize I/O consumers to display time waiting for I/O, grouped by thread. By default, rows are sorted by descending total I/O latency.

The `io_by_thread_by_latency` and `x$io_by_thread_by_latency` views have these columns:

- `user`

  For foreground threads, the account associated with the thread. For background threads, the thread name.

- `total`

  The total number of I/O events for the thread.

- `total_latency`

  The total wait time of timed I/O events for the thread.

- `min_latency`

  The minimum single wait time of timed I/O events for the thread.

- `avg_latency`

  The average wait time per timed I/O event for the thread.

- `max_latency`

  The maximum single wait time of timed I/O events for the thread.

- `thread_id`

  The thread ID.

- `processlist_id`

  For foreground threads, the processlist ID of the thread. For background threads, `NULL`.

27.4.3.11 The io_global_by_file_by_bytes and x$io_global_by_file_by_bytes Views

These views summarize global I/O consumers to display amount of I/O, grouped by file. By default, rows are sorted by descending total I/O (bytes read and written).

The `io_global_by_file_by_bytes` and `x$io_global_by_file_by_bytes` views have these columns:

- `file`

  The file path name.

- `count_read`

  The total number of read events for the file.

- `total_read`

  The total number of bytes read from the file.

- `avg_read`

  The average number of bytes per read from the file.

- `count_write`

  The total number of write events for the file.

- `total_written`

  The total number of bytes written to the file.

- `avg_write`

  The average number of bytes per write to the file.

- `total`

  The total number of bytes read and written for the file.

- `write_pct`

  The percentage of total bytes of I/O that were writes.

27.4.3.12 The io_global_by_file_by_latency and x$io_global_by_file_by_latency Views

These views summarize global I/O consumers to display time waiting for I/O, grouped by file. By default, rows are sorted by descending total latency.

The `io_global_by_file_by_latency` and `x$io_global_by_file_by_latency` views have these columns:

- `file`

  The file path name.

- `total`

  The total number of I/O events for the file.

- `total_latency`

  The total wait time of timed I/O events for the file.

- `count_read`

  The total number of read I/O events for the file.

- `read_latency`

  The total wait time of timed read I/O events for the file.

- `count_write`

  The total number of write I/O events for the file.

- `write_latency`

  The total wait time of timed write I/O events for the file.

- `count_misc`

  The total number of other I/O events for the file.

- `misc_latency`

  The total wait time of timed other I/O events for the file.

27.4.3.13 The io_global_by_wait_by_bytes and x$io_global_by_wait_by_bytes Views

These views summarize global I/O consumers to display amount of I/O and time waiting for I/O, grouped by event. By default, rows are sorted by descending total I/O (bytes read and written).

The `io_global_by_wait_by_bytes` and `x$io_global_by_wait_by_bytes` views have these columns:

- `event_name`

  The I/O event name, with the `wait/io/file/` prefix stripped.

- `total`

  The total number of occurrences of the I/O event.

- `total_latency`

  The total wait time of timed occurrences of the I/O event.

- `min_latency`

  The minimum single wait time of timed occurrences of the I/O event.

- `avg_latency`

  The average wait time per timed occurrence of the I/O event.

- `max_latency`

  The maximum single wait time of timed occurrences of the I/O event.

- `count_read`

  The number of read requests for the I/O event.

- `total_read`

  The number of bytes read for the I/O event.

- `avg_read`

  The average number of bytes per read for the I/O event.

- `count_write`

  The number of write requests for the I/O event.

- `total_written`

  The number of bytes written for the I/O event.

- `avg_written`

  The average number of bytes per write for the I/O event.

- `total_requested`

  The total number of bytes read and written for the I/O event.

27.4.3.14 The io_global_by_wait_by_latency and x$io_global_by_wait_by_latency Views

These views summarize global I/O consumers to display amount of I/O and time waiting for I/O, grouped by event. By default, rows are sorted by descending total latency.

The `io_global_by_wait_by_latency` and `x$io_global_by_wait_by_latency` views have these columns:

- `event_name`

  The I/O event name, with the `wait/io/file/` prefix stripped.

- `total`

  The total number of occurrences of the I/O event.

- `total_latency`

  The total wait time of timed occurrences of the I/O event.

- `avg_latency`

  The average wait time per timed occurrence of the I/O event.

- `max_latency`

  The maximum single wait time of timed occurrences of the I/O event.

- `read_latency`

  The total wait time of timed read occurrences of the I/O event.

- `write_latency`

  The total wait time of timed write occurrences of the I/O event.

- `misc_latency`

  The total wait time of timed other occurrences of the I/O event.

- `count_read`

  The number of read requests for the I/O event.

- `total_read`

  The number of bytes read for the I/O event.

- `avg_read`

  The average number of bytes per read for the I/O event.

- `count_write`

  The number of write requests for the I/O event.

- `total_written`

  The number of bytes written for the I/O event.

- `avg_written`

  The average number of bytes per write for the I/O event.

27.4.3.15 The latest_file_io and x$latest_file_io Views

These views summarize file I/O activity, grouped by file and thread. By default, rows are sorted with most recent I/O first.

The `latest_file_io` and `x$latest_file_io` views have these columns:

- `thread`

For foreground threads, the account associated with the thread (and port number for TCP/IP connections). For background threads, the thread name and thread ID

- `file`

  The file path name.

- `latency`

  The wait time of the file I/O event.

- `operation`

  The type of operation.

- `requested`

  The number of data bytes requested for the file I/O event.

27.4.3.16 The memory_by_host_by_current_bytes and x$memory_by_host_by_current_bytes Views

These views summarize memory use, grouped by host. By default, rows are sorted by descending amount of memory used.

The `memory_by_host_by_current_bytes` and `x$memory_by_host_by_current_bytes` views have these columns:

- `host`

  The host from which the client connected. Rows for which the `HOST` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `current_count_used`

  The current number of allocated memory blocks that have not been freed yet for the host.

- `current_allocated`

  The current number of allocated bytes that have not been freed yet for the host.

- `current_avg_alloc`

  The current number of allocated bytes per memory block for the host.

- `current_max_alloc`

  The largest single current memory allocation in bytes for the host.

- `total_allocated`

  The total memory allocation in bytes for the host.

27.4.3.17 The memory_by_thread_by_current_bytes and x$memory_by_thread_by_current_bytes Views

These views summarize memory use, grouped by thread. By default, rows are sorted by descending amount of memory used.

The `memory_by_thread_by_current_bytes` and `x$memory_by_thread_by_current_bytes` views have these columns:

- `thread_id`

  The thread ID.

- `user`

  The thread user or thread name.

- `current_count_used`

  The current number of allocated memory blocks that have not been freed yet for the thread.

- `current_allocated`

  The current number of allocated bytes that have not been freed yet for the thread.

- `current_avg_alloc`

  The current number of allocated bytes per memory block for the thread.

- `current_max_alloc`

  The largest single current memory allocation in bytes for the thread.

- `total_allocated`

  The total memory allocation in bytes for the thread.

27.4.3.18 The memory_by_user_by_current_bytes and x$memory_by_user_by_current_bytes Views

These views summarize memory use, grouped by user. By default, rows are sorted by descending amount of memory used.

The `memory_by_user_by_current_bytes` and `x$memory_by_user_by_current_bytes` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `current_count_used`

  The current number of allocated memory blocks that have not been freed yet for the user.

- `current_allocated`

  The current number of allocated bytes that have not been freed yet for the user.

- `current_avg_alloc`

The current number of allocated bytes per memory block for the user.

- `current_max_alloc`

  The largest single current memory allocation in bytes for the user.

- `total_allocated`

  The total memory allocation in bytes for the user.

27.4.3.19 The memory_global_by_current_bytes and x$memory_global_by_current_bytes Views

These views summarize memory use, grouped by allocation type (that is, by event). By default, rows are sorted by descending amount of memory used.

The `memory_global_by_current_bytes` and `x$memory_global_by_current_bytes` views have these columns:

- `event_name`

  The memory event name.

- `current_count`

  The total number of occurrences of the event.

- `current_alloc`

  The current number of allocated bytes that have not been freed yet for the event.

- `current_avg_alloc`

  The current number of allocated bytes per memory block for the event.

- `high_count`

  The high-water mark for number of memory blocks allocated for the event.

- `high_alloc`

  The high-water mark for number of bytes allocated for the event.

- `high_avg_alloc`

  The high-water mark for average number of bytes per memory block allocated for the event.

27.4.3.20 The memory_global_total and x$memory_global_total Views

These views summarize total memory use within the server.

The `memory_global_total` and `x$memory_global_total` views have these columns:

- `total_allocated`

  The total bytes of memory allocated within the server.

27.4.3.21 The metrics View

This view summarizes MySQL server metrics to show variable names, values, types, and whether they are enabled. By default, rows are sorted by variable type and name.

The `metrics` view includes this information:

- Global status variables from the Performance Schema `global_status` table
- `InnoDB` metrics from the `INFORMATION_SCHEMA INNODB_METRICS` table
- Current and total memory allocation, based on the Performance Schema memory instrumentation
- The current time (human readable and Unix timestamp formats)

There is some duplication of information between the `global_status` and `INNODB_METRICS` tables, which the `metrics` view eliminates.

The `metrics` view has these columns:

- `Variable_name`

  The metric name. The metric type determines the source from which the name is taken:

  - For global status variables: The `VARIABLE_NAME` column of the `global_status` table
  - For `InnoDB` metrics: The `NAME` column of the `INNODB_METRICS` table
  - For other metrics: A view-provided descriptive string

- `Variable_value`

  The metric value. The metric type determines the source from which the value is taken:

  - For global status variables: The `VARIABLE_VALUE` column of the `global_status` table
  - For `InnoDB` metrics: The `COUNT` column of the `INNODB_METRICS` table
  - For memory metrics: The relevant column from the Performance Schema `memory_summary_global_by_event_name` table
  - For the current time: The value of `NOW(3)` or `UNIX_TIMESTAMP(NOW(3))`

- `Type`

  The metric type:

  - For global status variables: `Global Status`
  - For `InnoDB` metrics: `InnoDB Metrics - %`, where `%` is replaced by the value of the `SUBSYSTEM` column of the `INNODB_METRICS` table
  - For memory metrics: `Performance Schema`
  - For the current time: `System Time`

- `Enabled`

  Whether the metric is enabled:

  - For global status variables: `YES`

- For `InnoDB` metrics: `YES` if the `STATUS` column of the `INNODB_METRICS` table is `enabled`, `NO` otherwise
- For memory metrics: `NO`, `YES`, or `PARTIAL` (currently, `PARTIAL` occurs only for memory metrics and indicates that not all `memory/%` instruments are enabled; Performance Schema memory instruments are always enabled)
- For the current time: `YES`

27.4.3.22 The processlist and x$processlist Views

These views summarize processlist information. They provide more complete information than the `SHOW PROCESSLIST` statement and the `INFORMATION_SCHEMA PROCESSLIST` table, and are also nonblocking. By default, rows are sorted by descending process time and descending wait time.

The column descriptions here are brief. For additional information, see the description of the Performance Schema `threads` table at Section 26.12.18.5, “The threads Table”.

The `processlist` and `x$processlist` views have these columns:

- `thd_id`

  The thread ID.

- `conn_id`

  The connection ID.

- `user`

  The thread user or thread name.

- `db`

  The default database for the thread, or `NULL` if there is none.

- `command`

  For foreground threads, the type of command the thread is executing on behalf of the client, or `Sleep` if the session is idle.

- `state`

  An action, event, or state that indicates what the thread is doing.

- `time`

  The time in seconds that the thread has been in its current state.

- `current_statement`

  The statement the thread is executing, or `NULL` if it is not executing any statement.

- `statement_latency`

  How long the statement has been executing.

- `progress`

The percentage of work completed for stages that support progress reporting. See Section 27.3, “sys Schema Progress Reporting”.

- `lock_latency`

  The time spent waiting for locks by the current statement.

- `rows_examined`

  The number of rows read from storage engines by the current statement.

- `rows_sent`

  The number of rows returned by the current statement.

- `rows_affected`

  The number of rows affected by the current statement.

- `tmp_tables`

  The number of internal in-memory temporary tables created by the current statement.

- `tmp_disk_tables`

  The number of internal on-disk temporary tables created by the current statement.

- `full_scan`

  The number of full table scans performed by the current statement.

- `last_statement`

  The last statement executed by the thread, if there is no currently executing statement or wait.

- `last_statement_latency`

  How long the last statement executed.

- `current_memory`

  The number of bytes allocated by the thread.

- `last_wait`

  The name of the most recent wait event for the thread.

- `last_wait_latency`

  The wait time of the most recent wait event for the thread.

- `source`

  The source file and line number containing the instrumented code that produced the event.

- `trx_latency`

  The wait time of the current transaction for the thread.

- `trx_state`

  The state for the current transaction for the thread.

- `trx_autocommit`

  Whether autocommit mode was enabled when the current transaction started.

- `pid`

  The client process ID.

- `program_name`

  The client program name.

27.4.3.23 The ps_check_lost_instrumentation View

This view returns information about lost Performance Schema instruments, to indicate whether the Performance Schema is unable to monitor all runtime data.

The `ps_check_lost_instrumentation` view has these columns:

- `variable_name`

  The Performance Schema status variable name indicating which type of instrument was lost.

- `variable_value`

  The number of instruments lost.

27.4.3.24 The schema_auto_increment_columns View

This view indicates which tables have `AUTO_INCREMENT` columns and provides information about those columns, such as the current and maximum column values and the usage ratio (ratio of used to possible values). By default, rows are sorted by descending usage ratio and maximum column value.

Tables in these schemas are excluded from view output: `mysql`, `sys`, `INFORMATION_SCHEMA`, `performance_schema`.

The `schema_auto_increment_columns` view has these columns:

- `table_schema`

  The schema that contains the table.

- `table_name`

  The table that contains the `AUTO_INCREMENT` column.

- `column_name`

  The name of the `AUTO_INCREMENT` column.

- `data_type`

  The data type of the column.

- `column_type`

The column type of the column, which is the data type plus possibly other information. For example, for a column with a `bigint(20) unsigned` column type, the data type is just `bigint`.

- `is_signed`

  Whether the column type is signed.

- `is_unsigned`

  Whether the column type is unsigned.

- `max_value`

  The maximum permitted value for the column.

- `auto_increment`

  The current `AUTO_INCREMENT` value for the column.

- `auto_increment_ratio`

  The ratio of used to permitted values for the column. This indicates how much of the sequence of values is “used up.”

27.4.3.25 The schema_index_statistics and x$schema_index_statistics Views

These views provide index statistics. By default, rows are sorted by descending total index latency.

The `schema_index_statistics` and `x$schema_index_statistics` views have these columns:

- `table_schema`

  The schema that contains the table.

- `table_name`

  The table that contains the index.

- `index_name`

  The name of the index.

- `rows_selected`

  The total number of rows read using the index.

- `select_latency`

  The total wait time of timed reads using the index.

- `rows_inserted`

  The total number of rows inserted into the index.

- `insert_latency`

  The total wait time of timed inserts into the index.

- `rows_updated`

The total number of rows updated in the index.

- `update_latency`

  The total wait time of timed updates in the index.

- `rows_deleted`

  The total number of rows deleted from the index.

- `delete_latency`

  The total wait time of timed deletes from the index.

27.4.3.26 The schema_object_overview View

This view summarizes the types of objects within each schema. By default, rows are sorted by schema and object type.

Note

For MySQL instances with a large number of objects, this view might take a long time to execute.

The `schema_object_overview` view has these columns:

- `db`

  The schema name.

- `object_type`

  The object type: `BASE TABLE`, `INDEX (index_type)`, `EVENT`, `FUNCTION`, `PROCEDURE`, `TRIGGER`, `VIEW`.

- `count`

  The number of objects in the schema of the given type.

27.4.3.27 The schema_redundant_indexes and x$schema_flattened_keys Views

The `schema_redundant_indexes` view displays indexes that duplicate other indexes or are made redundant by them. The `x$schema_flattened_keys` view is a helper view for `schema_redundant_indexes`.

In the following column descriptions, the dominant index is the one that makes the redundant index redundant.

The `schema_redundant_indexes` view has these columns:

- `table_schema`

  The schema that contains the table.

- `table_name`

  The table that contains the index.

- `redundant_index_name`

  The name of the redundant index.

- `redundant_index_columns`

  The names of the columns in the redundant index.

- `redundant_index_non_unique`

  The number of nonunique columns in the redundant index.

- `dominant_index_name`

  The name of the dominant index.

- `dominant_index_columns`

  The names of the columns in the dominant index.

- `dominant_index_non_unique`

  The number of nonunique columns in the dominant index.

- `subpart_exists`

  Whether the index indexes only part of a column.

- `sql_drop_index`

  The statement to execute to drop the redundant index.

The `x$schema_flattened_keys` view has these columns:

- `table_schema`

  The schema that contains the table.

- `table_name`

  The table that contains the index.

- `index_name`

  An index name.

- `non_unique`

  The number of nonunique columns in the index.

- `subpart_exists`

  Whether the index indexes only part of a column.

- `index_columns`

  The name of the columns in the index.

27.4.3.28 The schema_table_lock_waits and x$schema_table_lock_waits Views

These views display which sessions are blocked waiting on metadata locks, and what is blocking them.

The column descriptions here are brief. For additional information, see the description of the Performance Schema `metadata_locks` table at Section 26.12.12.3, “The metadata_locks Table”.

The `schema_table_lock_waits` and `x$schema_table_lock_waits` views have these columns:

- `object_schema`

  The schema containing the object to be locked.

- `object_name`

  The name of the instrumented object.

- `waiting_thread_id`

  The thread ID of the thread that is waiting for the lock.

- `waiting_pid`

  The processlist ID of the thread that is waiting for the lock.

- `waiting_account`

  The account associated with the session that is waiting for the lock.

- `waiting_lock_type`

  The type of the waiting lock.

- `waiting_lock_duration`

  How long the waiting lock has been waiting.

- `waiting_query`

  The statement that is waiting for the lock.

- `waiting_query_secs`

  How long the statement has been waiting, in seconds.

- `waiting_query_rows_affected`

  The number of rows affected by the statement.

- `waiting_query_rows_examined`

  The number of rows read from storage engines by the statement.

- `blocking_thread_id`

  The thread ID of the thread that is blocking the waiting lock.

- `blocking_pid`

  The processlist ID of the thread that is blocking the waiting lock.

- `blocking_account`

The account associated with the thread that is blocking the waiting lock.

- `blocking_lock_type`

  The type of lock that is blocking the waiting lock.

- `blocking_lock_duration`

  How long the blocking lock has been held.

- `sql_kill_blocking_query`

  The `KILL` statement to execute to kill the blocking statement.

- `sql_kill_blocking_connection`

  The `KILL` statement to execute to kill the session running the blocking statement.

27.4.3.29 The schema_table_statistics and x$schema_table_statistics Views

These views summarize table statistics. By default, rows are sorted by descending total wait time (tables with most contention first).

These views user a helper view, `x$ps_schema_table_statistics_io`.

The `schema_table_statistics` and `x$schema_table_statistics` views have these columns:

- `table_schema`

  The schema that contains the table.

- `table_name`

  The table name.

- `total_latency`

  The total wait time of timed I/O events for the table.

- `rows_fetched`

  The total number of rows read from the table.

- `fetch_latency`

  The total wait time of timed read I/O events for the table.

- `rows_inserted`

  The total number of rows inserted into the table.

- `insert_latency`

  The total wait time of timed insert I/O events for the table.

- `rows_updated`

  The total number of rows updated in the table.

- `update_latency`

  The total wait time of timed update I/O events for the table.

- `rows_deleted`

  The total number of rows deleted from the table.

- `delete_latency`

  The total wait time of timed delete I/O events for the table.

- `io_read_requests`

  The total number of read requests for the table.

- `io_read`

  The total number of bytes read from the table.

- `io_read_latency`

  The total wait time of reads from the table.

- `io_write_requests`

  The total number of write requests for the table.

- `io_write`

  The total number of bytes written to the table.

- `io_write_latency`

  The total wait time of writes to the table.

- `io_misc_requests`

  The total number of miscellaneous I/O requests for the table.

- `io_misc_latency`

  The total wait time of miscellaneous I/O requests for the table.

27.4.3.30 The schema_table_statistics_with_buffer and x $schema_table_statistics_with_buffer Views

These views summarize table statistics, including `InnoDB` buffer pool statistics. By default, rows are sorted by descending total wait time (tables with most contention first).

These views user a helper view, `x$ps_schema_table_statistics_io`.

The `schema_table_statistics_with_buffer` and `x $schema_table_statistics_with_buffer` views have these columns:

- `table_schema`

  The schema that contains the table.

- `table_name`

  The table name.

- `rows_fetched`

  The total number of rows read from the table.

- `fetch_latency`

  The total wait time of timed read I/O events for the table.

- `rows_inserted`

  The total number of rows inserted into the table.

- `insert_latency`

  The total wait time of timed insert I/O events for the table.

- `rows_updated`

  The total number of rows updated in the table.

- `update_latency`

  The total wait time of timed update I/O events for the table.

- `rows_deleted`

  The total number of rows deleted from the table.

- `delete_latency`

  The total wait time of timed delete I/O events for the table.

- `io_read_requests`

  The total number of read requests for the table.

- `io_read`

  The total number of bytes read from the table.

- `io_read_latency`

  The total wait time of reads from the table.

- `io_write_requests`

  The total number of write requests for the table.

- `io_write`

  The total number of bytes written to the table.

- `io_write_latency`

  The total wait time of writes to the table.

- `io_misc_requests`

  The total number of miscellaneous I/O requests for the table.

- `io_misc_latency`

  The total wait time of miscellaneous I/O requests for the table.

- `innodb_buffer_allocated`

  The total number of `InnoDB` buffer bytes allocated for the table.

- `innodb_buffer_data`

  The total number of `InnoDB` data bytes allocated for the table.

- `innodb_buffer_free`

  The total number of `InnoDB` nondata bytes allocated for the table (`innodb_buffer_allocated` − `innodb_buffer_data`).

- `innodb_buffer_pages`

  The total number of `InnoDB` pages allocated for the table.

- `innodb_buffer_pages_hashed`

  The total number of `InnoDB` hashed pages allocated for the table.

- `innodb_buffer_pages_old`

  The total number of `InnoDB` old pages allocated for the table.

- `innodb_buffer_rows_cached`

  The total number of `InnoDB` cached rows for the table.

27.4.3.31 The schema_tables_with_full_table_scans and x $schema_tables_with_full_table_scans Views

These views display which tables are being accessed with full table scans. By default, rows are sorted by descending rows scanned.

The `schema_tables_with_full_table_scans` and `x $schema_tables_with_full_table_scans` views have these columns:

- `object_schema`

  The schema name.

- `object_name`

  The table name.

- `rows_full_scanned`

  The total number of rows scanned by full scans of the table.

- `latency`

The total wait time of full scans of the table.

27.4.3.32 The schema_unused_indexes View

These views display indexes for which there are no events, which indicates that they are not being used. By default, rows are sorted by schema and table.

This view is most useful when the server has been up and processing long enough that its workload is representative. Otherwise, presence of an index in this view may not be meaningful.

The `schema_unused_indexes` view has these columns:

- `object_schema`

  The schema name.

- `object_name`

  The table name.

- `index_name`

  The unused index name.

27.4.3.33 The session and x$session Views

These views are similar to `processlist` and `x$processlist`, but they filter out background processes to display only user sessions. For descriptions of the columns, see Section 27.4.3.22, “The processlist and x$processlist Views”.

27.4.3.34 The session_ssl_status View

For each connection, this view displays the SSL version, cipher, and count of reused SSL sessions.

The `session_ssl_status` view has these columns:

- `thread_id`

  The thread ID for the connection.

- `ssl_version`

  The version of SSL used for the connection.

- `ssl_cipher`

  The SSL cipher used for the connection.

- `ssl_sessions_reused`

  The number of reused SSL sessions for the connection.

27.4.3.35 The statement_analysis and x$statement_analysis Views

These views list normalized statements with aggregated statistics. The content mimics the MySQL Enterprise Monitor Query Analysis view. By default, rows are sorted by descending total latency.

The `statement_analysis` and `x$statement_analysis` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `full_scan`

  The total number of full table scans performed by occurrences of the statement.

- `exec_count`

  The total number of times the statement has executed.

- `err_count`

  The total number of errors produced by occurrences of the statement.

- `warn_count`

  The total number of warnings produced by occurrences of the statement.

- `total_latency`

  The total wait time of timed occurrences of the statement.

- `max_latency`

  The maximum single wait time of timed occurrences of the statement.

- `avg_latency`

  The average wait time per timed occurrence of the statement.

- `lock_latency`

  The total time waiting for locks by timed occurrences of the statement.

- `rows_sent`

  The total number of rows returned by occurrences of the statement.

- `rows_sent_avg`

  The average number of rows returned per occurrence of the statement.

- `rows_examined`

  The total number of rows read from storage engines by occurrences of the statement.

- `rows_examined_avg`

  The average number of rows read from storage engines per occurrence of the statement.

- `rows_affected`

  The total number of rows affected by occurrences of the statement.

- `rows_affected_avg`

  The average number of rows affected per occurrence of the statement.

- `tmp_tables`

  The total number of internal in-memory temporary tables created by occurrences of the statement.

- `tmp_disk_tables`

  The total number of internal on-disk temporary tables created by occurrences of the statement.

- `rows_sorted`

  The total number of rows sorted by occurrences of the statement.

- `sort_merge_passes`

  The total number of sort merge passes by occurrences of the statement.

- `digest`

  The statement digest.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

27.4.3.36 The statements_with_errors_or_warnings and x$statements_with_errors_or_warnings Views

These views display normalized statements that have produced errors or warnings. By default, rows are sorted by descending error and warning counts.

The `statements_with_errors_or_warnings` and `x$statements_with_errors_or_warnings` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `exec_count`

  The total number of times the statement has executed.

- `errors`

  The total number of errors produced by occurrences of the statement.

- `error_pct`

The percentage of statement occurrences that produced errors.

- `warnings`

  The total number of warnings produced by occurrences of the statement.

- `warning_pct`

  The percentage of statement occurrences that produced warnings.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

- `digest`

  The statement digest.

27.4.3.37 The statements_with_full_table_scans and x$statements_with_full_table_scans Views

These views display normalized statements that have done full table scans. By default, rows are sorted by descending percentage of time a full scan was done and descending total latency.

The `statements_with_full_table_scans` and `x$statements_with_full_table_scans` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `exec_count`

  The total number of times the statement has executed.

- `total_latency`

  The total wait time of timed statement events for the statement.

- `no_index_used_count`

  The total number of times no index was used to scan the table.

- `no_good_index_used_count`

  The total number of times no good index was used to scan the table.

- `no_index_used_pct`

  The percentage of the time no index was used to scan the table.

- `rows_sent`

  The total number of rows returned from the table.

- `rows_examined`

  The total number of rows read from the storage engine for the table.

- `rows_sent_avg`

  The average number of rows returned from the table.

- `rows_examined_avg`

  The average number of rows read from the storage engine for the table.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

- `digest`

  The statement digest.

27.4.3.38 The statements_with_runtimes_in_95th_percentile and x$statements_with_runtimes_in_95th_percentile Views

These views list statements with runtimes in the 95th percentile. By default, rows are sorted by descending average latency.

Both views use two helper views, `x$ps_digest_avg_latency_distribution` and `x$ps_digest_95th_percentile_by_avg_us`.

The `statements_with_runtimes_in_95th_percentile` and `x$statements_with_runtimes_in_95th_percentile` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `full_scan`

  The total number of full table scans performed by occurrences of the statement.

- `exec_count`

  The total number of times the statement has executed.

- `err_count`

  The total number of errors produced by occurrences of the statement.

- `warn_count`

  The total number of warnings produced by occurrences of the statement.

- `total_latency`

  The total wait time of timed occurrences of the statement.

- `max_latency`

  The maximum single wait time of timed occurrences of the statement.

- `avg_latency`

  The average wait time per timed occurrence of the statement.

- `rows_sent`

  The total number of rows returned by occurrences of the statement.

- `rows_sent_avg`

  The average number of rows returned per occurrence of the statement.

- `rows_examined`

  The total number of rows read from storage engines by occurrences of the statement.

- `rows_examined_avg`

  The average number of rows read from storage engines per occurrence of the statement.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

- `digest`

  The statement digest.

27.4.3.39 The statements_with_sorting and x$statements_with_sorting Views

These views list normalized statements that have performed sorts. By default, rows are sorted by descending total latency.

The `statements_with_sorting` and `x$statements_with_sorting` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `exec_count`

The total number of times the statement has executed.

- `total_latency`

  The total wait time of timed occurrences of the statement.

- `sort_merge_passes`

  The total number of sort merge passes by occurrences of the statement.

- `avg_sort_merges`

  The average number of sort merge passes per occurrence of the statement.

- `sorts_using_scans`

  The total number of sorts using table scans by occurrences of the statement.

- `sort_using_range`

  The total number of sorts using range accesses by occurrences of the statement.

- `rows_sorted`

  The total number of rows sorted by occurrences of the statement.

- `avg_rows_sorted`

  The average number of rows sorted per occurrence of the statement.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

- `digest`

  The statement digest.

27.4.3.40 The statements_with_temp_tables and x$statements_with_temp_tables Views

These views list normalized statements that have used temporary tables. By default, rows are sorted by descending number of on-disk temporary tables used and descending number of in-memory temporary tables used.

The `statements_with_temp_tables` and `x$statements_with_temp_tables` views have these columns:

- `query`

  The normalized statement string.

- `db`

  The default database for the statement, or `NULL` if there is none.

- `exec_count`

  The total number of times the statement has executed.

- `total_latency`

  The total wait time of timed occurrences of the statement.

- `memory_tmp_tables`

  The total number of internal in-memory temporary tables created by occurrences of the statement.

- `disk_tmp_tables`

  The total number of internal on-disk temporary tables created by occurrences of the statement.

- `avg_tmp_tables_per_query`

  The average number of internal temporary tables created per occurrence of the statement.

- `tmp_tables_to_disk_pct`

  The percentage of internal in-memory temporary tables that were converted to on-disk tables.

- `first_seen`

  The time at which the statement was first seen.

- `last_seen`

  The time at which the statement was most recently seen.

- `digest`

  The statement digest.

27.4.3.41 The user_summary and x$user_summary Views

These views summarize statement activity, file I/O, and connections, grouped by user. By default, rows are sorted by descending total latency.

The `user_summary` and `x$user_summary` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `statements`

  The total number of statements for the user.

- `statement_latency`

  The total wait time of timed statements for the user.

- `statement_avg_latency`

  The average wait time per timed statement for the user.

- `table_scans`

  The total number of table scans for the user.

- `file_ios`

  The total number of file I/O events for the user.

- `file_io_latency`

  The total wait time of timed file I/O events for the user.

- `current_connections`

  The current number of connections for the user.

- `total_connections`

  The total number of connections for the user.

- `unique_hosts`

  The number of distinct hosts from which connections for the user have originated.

- `current_memory`

  The current amount of allocated memory for the user.

- `total_memory_allocated`

  The total amount of allocated memory for the user.

27.4.3.42 The user_summary_by_file_io and x$user_summary_by_file_io Views

These views summarize file I/O, grouped by user. By default, rows are sorted by descending total file I/O latency.

The `user_summary_by_file_io` and `x$user_summary_by_file_io` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `ios`

  The total number of file I/O events for the user.

- `io_latency`

  The total wait time of timed file I/O events for the user.

27.4.3.43 The user_summary_by_file_io_type and x$user_summary_by_file_io_type Views

These views summarize file I/O, grouped by user and event type. By default, rows are sorted by user and descending total latency.

The `user_summary_by_file_io_type` and `x$user_summary_by_file_io_type` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `event_name`

  The file I/O event name.

- `total`

  The total number of occurrences of the file I/O event for the user.

- `latency`

  The total wait time of timed occurrences of the file I/O event for the user.

- `max_latency`

  The maximum single wait time of timed occurrences of the file I/O event for the user.

27.4.3.44 The user_summary_by_stages and x$user_summary_by_stages Views

These views summarize stages, grouped by user. By default, rows are sorted by user and descending total stage latency.

The `user_summary_by_stages` and `x$user_summary_by_stages` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `event_name`

  The stage event name.

- `total`

  The total number of occurrences of the stage event for the user.

- `total_latency`

  The total wait time of timed occurrences of the stage event for the user.

- `avg_latency`

  The average wait time per timed occurrence of the stage event for the user.

27.4.3.45 The user_summary_by_statement_latency and x$user_summary_by_statement_latency Views

These views summarize overall statement statistics, grouped by user. By default, rows are sorted by descending total latency.

The `user_summary_by_statement_latency` and `x$user_summary_by_statement_latency` views have these columns:

- `user`

The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `total`

  The total number of statements for the user.

- `total_latency`

  The total wait time of timed statements for the user.

- `max_latency`

  The maximum single wait time of timed statements for the user.

- `lock_latency`

  The total time waiting for locks by timed statements for the user.

- `rows_sent`

  The total number of rows returned by statements for the user.

- `rows_examined`

  The total number of rows read from storage engines by statements for the user.

- `rows_affected`

  The total number of rows affected by statements for the user.

- `full_scans`

  The total number of full table scans by statements for the user.

27.4.3.46 The user_summary_by_statement_type and x$user_summary_by_statement_type Views

These views summarize informaion about statements executed, grouped by user and statement type. By default, rows are sorted by user and descending total latency.

The `user_summary_by_statement_type` and `x$user_summary_by_statement_type` views have these columns:

- `user`

  The client user name. Rows for which the `USER` column in the underlying Performance Schema table is `NULL` are assumed to be for background threads and are reported with a host name of `background`.

- `statement`

  The final component of the statement event name.

- `total`

  The total number of occurrences of the statement event for the user.

- `total_latency`

The total wait time of timed occurrences of the statement event for the user.

- `max_latency`

  The maximum single wait time of timed occurrences of the statement event for the user.

- `lock_latency`

  The total time waiting for locks by timed occurrences of the statement event for the user.

- `rows_sent`

  The total number of rows returned by occurrences of the statement event for the user.

- `rows_examined`

  The total number of rows read from storage engines by occurrences of the statement event for the user.

- `rows_affected`

  The total number of rows affected by occurrences of the statement event for the user.

- `full_scans`

  The total number of full table scans by occurrences of the statement event for the user.

27.4.3.47 The version View

This view provides the current `sys` schema and MySQL server versions.

Note

As of MySQL 8.0.18, this view is deprecated and will be removed in a future MySQL version. Affected applications should be adjusted to use an alternative instead. For example, use the `VERSION()` function to retrieve the MySQL server version.

The `version` view has these columns:

- `sys_version`

  The `sys` schema version.

- `mysql_version`

  The MySQL server version.

27.4.3.48 The wait_classes_global_by_avg_latency and x$wait_classes_global_by_avg_latency Views

These views summarize wait class average latencies, grouped by event class. By default, rows are sorted by descending average latency. Idle events are ignored.

An event class is determined by stripping from the event name everything after the first three components. For example, the class for `wait/io/file/sql/slow_log` is `wait/io/file`.

The `wait_classes_global_by_avg_latency` and `x$wait_classes_global_by_avg_latency` views have these columns:

- `event_class`

  The event class.

- `total`

  The total number of occurrences of events in the class.

- `total_latency`

  The total wait time of timed occurrences of events in the class.

- `min_latency`

  The minimum single wait time of timed occurrences of events in the class.

- `avg_latency`

  The average wait time per timed occurrence of events in the class.

- `max_latency`

  The maximum single wait time of timed occurrences of events in the class.

27.4.3.49 The wait_classes_global_by_latency and x$wait_classes_global_by_latency Views

These views summarize wait class total latencies, grouped by event class. By default, rows are sorted by descending total latency. Idle events are ignored.

An event class is determined by stripping from the event name everything after the first three components. For example, the class for `wait/io/file/sql/slow_log` is `wait/io/file`.

The `wait_classes_global_by_latency` and `x$wait_classes_global_by_latency` views have these columns:

- `event_class`

  The event class.

- `total`

  The total number of occurrences of events in the class.

- `total_latency`

  The total wait time of timed occurrences of events in the class.

- `min_latency`

  The minimum single wait time of timed occurrences of events in the class.

- `avg_latency`

  The average wait time per timed occurrence of events in the class.

- `max_latency`

  The maximum single wait time of timed occurrences of events in the class.

27.4.3.50 The waits_by_host_by_latency and x$waits_by_host_by_latency Views

These views summarize wait events, grouped by host and event. By default, rows are sorted by host and descending total latency. Idle events are ignored.

The `waits_by_host_by_latency` and `x$waits_by_host_by_latency` views have these columns:

- `host`

  The host from which the connection originated.

- `event`

  The event name.

- `total`

  The total number of occurrences of the event for the host.

- `total_latency`

  The total wait time of timed occurrences of the event for the host.

- `avg_latency`

  The average wait time per timed occurrence of the event for the host.

- `max_latency`

  The maximum single wait time of timed occurrences of the event for the host.

27.4.3.51 The waits_by_user_by_latency and x$waits_by_user_by_latency Views

These views summarize wait events, grouped by user and event. By default, rows are sorted by user and descending total latency. Idle events are ignored.

The `waits_by_user_by_latency` and `x$waits_by_user_by_latency` views have these columns:

- `user`

  The user associated with the connection.

- `event`

  The event name.

- `total`

  The total number of occurrences of the event for the user.

- `total_latency`

  The total wait time of timed occurrences of the event for the user.

- `avg_latency`

  The average wait time per timed occurrence of the event for the user.

- `max_latency`

The maximum single wait time of timed occurrences of the event for the user.

27.4.3.52 The waits_global_by_latency and x$waits_global_by_latency Views

These views summarize wait events, grouped by event. By default, rows are sorted by descending total latency. Idle events are ignored.

The `waits_global_by_latency` and `x$waits_global_by_latency` views have these columns:

- `events`

  The event name.

- `total`

  The total number of occurrences of the event.

- `total_latency`

  The total wait time of timed occurrences of the event.

- `avg_latency`

  The average wait time per timed occurrence of the event.

- `max_latency`

  The maximum single wait time of timed occurrences of the event.

27.4.4 sys Schema Stored Procedures

The following sections describe `sys` schema stored procedures.

27.4.4.1 The create_synonym_db() Procedure

Given a schema name, this procedure creates a synonym schema containing views that refer to all the tables and views in the original schema. This can be used, for example, to create a shorter name by which to refer to a schema with a long name (such as `info` rather than `INFORMATION_SCHEMA`).

Parameters

- `in_db_name VARCHAR(64)`: The name of the schema for which to create the synonym.
- `in_synonym VARCHAR(64)`: The name to use for the synonym schema. This schema must not already exist.

Example

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| world              |
```

```
+--------------------+
mysql> CALL sys.create_synonym_db('INFORMATION_SCHEMA', 'info');
+---------------------------------------+
| summary                               |
+---------------------------------------+
| Created 63 views in the info database |
+---------------------------------------+
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| info               |
| mysql              |
| performance_schema |
| sys                |
| world              |
+--------------------+
mysql> SHOW FULL TABLES FROM info;
+---------------------------------------+------------+
| Tables_in_info                        | Table_type |
+---------------------------------------+------------+
| character_sets                        | VIEW       |
| collation_character_set_applicability | VIEW       |
| collations                            | VIEW       |
| column_privileges                     | VIEW       |
| columns                               | VIEW       |
...
```

27.4.4.2 The diagnostics() Procedure

Creates a report of the current server status for diagnostic purposes.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Data collected for `diagnostics()` includes this information:

- Information from the `metrics` view (see Section 27.4.3.21, “The metrics View”)
- Information from other relevant `sys` schema views, such as the one that detemines queries in the 95th percentile
- Information from the `ndbinfo` schema, if the MySQL server is part of NDB Cluster
- Replication status (both master and slave)

Some of the sys schema views are calculated as initial (optional), overall, and delta values:

- The initial view is the content of the view at the start of the `diagnostics()` procedure. This output is the same as the start values used for the delta view. The initial view is included if the `diagnostics.include_raw` configuration option is `ON`.
- The overall view is the content of the view at the end of the `diagnostics()` procedure. This output is the same as the end values used for the delta view. The overall view is always included.
- The delta view is the difference from the beginning to the end of procedure execution. The minimum and maximum values are the minimum and maximum values from the end view, respectively. They do not necessarily reflect the minimum and maximum values in the monitored period. Except for the `metrics` view, the delta is calculated only between the first and last outputs.

Parameters

- `in_max_runtime INT UNSIGNED`: The maximum data collection time in seconds. Use `NULL` to collect data for the default of 60 seconds. Otherwise, use a value greater than 0.
- `in_interval INT UNSIGNED`: The sleep time between data collections in seconds. Use `NULL` to sleep for the default of 30 seconds. Otherwise, use a value greater than 0.
- `in_auto_config ENUM('current', 'medium', 'full')`: The Performance Schema configuration to use. Permitted values are:
  - `current`: Use the current instrument and consumer settings.
  - `medium`: Enable some instruments and consumers.
  - `full`: Enable all instruments and consumers.

Note

The more instruments and consumers enabled, the more impact on MySQL server performance. Be careful with the `medium` setting and especially the `full` setting, which has a large performance impact.

Use of the `medium` or `full` setting requires the `SUPER` privilege.

If a setting other than `current` is chosen, the current settings are restored at the end of the procedure.

Configuration Options

`diagnostics()` operation can be modified using the following configuration options or their corresponding user-defined variables (see Section 27.4.2.1, “The sys_config Table”):

- `debug`, `@sys.debug`

  If this option is `ON`, produce debugging output. The default is `OFF`.

- `diagnostics.allow_i_s_tables`, `@sys.diagnostics.allow_i_s_tables`

  If this option is `ON`, the `diagnostics()` procedure is permitted to perform table scans on the `INFORMATION_SCHEMA.TABLES` table. This can be expensive if there are many tables. The default is `OFF`.

- `diagnostics.include_raw`, `@sys.diagnostics.include_raw`

  If this option is `ON`, the `diagnostics()` procedure output includes the raw output from querying the `metrics` view. The default is `OFF`.

- `statement_truncate_len`, `@sys.statement_truncate_len`

  The maximum length of statements returned by the `format_statement()` function. Longer statements are truncated to this length. The default is 64.

Example

Create a diagnostics report that starts an iteration every 30 seconds and runs for at most 120 seconds using the current Performance Schema settings:

```
mysql> CALL sys.diagnostics(120, 30, 'current');
```

To capture the output from the `diagnostics()` procedure in a file as it runs, use the `mysql` client `tee filename` and `notee` commands (see Section 4.5.1.2, “mysql Client Commands”):

```
mysql> tee diag.out;
mysql> CALL sys.diagnostics(120, 30, 'current');
mysql> notee;
```

27.4.4.3 The execute_prepared_stmt() Procedure

Given an SQL statement as a string, executes it as a prepared statement. The prepared statement is deallocated after execution, so it is not subject to reuse. Thus, this procedure is useful primarily for executing dynamic statements on a one-time basis.

This procedure uses `sys_execute_prepared_stmt` as the prepared statement name. If that statement name exists when the procedure is called, its previous content is destroyed.

Parameters

- `in_query LONGTEXT CHARACTER SET utf8`: The statement string to execute.

Configuration Options

`execute_prepared_stmt()` operation can be modified using the following configuration options or their corresponding user-defined variables (see Section 27.4.2.1, “The sys_config Table”):

- `debug`, `@sys.debug`

  If this option is `ON`, produce debugging output. The default is `OFF`.

Example

```
mysql> CALL sys.execute_prepared_stmt('SELECT COUNT(*) FROM mysql.user');
+----------+
| COUNT(*) |
+----------+
|       15 |
+----------+
```

27.4.4.4 The ps_setup_disable_background_threads() Procedure

Disables Performance Schema instrumentation for all background threads. Produces a result set indicating how many background threads were disabled. Already disabled threads do not count.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_disable_background_threads();
+--------------------------------+
| summary                        |
+--------------------------------+
| Disabled 24 background threads |
```

```
+------------------------------+
```

27.4.4.5 The ps_setup_disable_consumer() Procedure

Disables Performance Schema consumers with names that contain the argument. Produces a result set indicating how many consumers were disabled. Already disabled consumers do not count.

Parameters

- `consumer VARCHAR(128)`: The value used to match consumer names, which are identified by using `%consumer%` as an operand for a `LIKE` pattern match.

  A value of `''` matches all consumers.

Example

Disable all statement consumers:

```
mysql> CALL sys.ps_setup_disable_consumer('statement');
+----------------------+
| summary              |
+----------------------+
| Disabled 4 consumers |
+----------------------+
```

27.4.4.6 The ps_setup_disable_instrument() Procedure

Disables Performance Schema instruments with names that contain the argument. Produces a result set indicating how many instruments were disabled. Already disabled instruments do not count.

Parameters

- `in_pattern VARCHAR(128)`: The value used to match instrument names, which are identified by using `%in_pattern%` as an operand for a `LIKE` pattern match.

  A value of `''` matches all instruments.

Example

Disable a specific instrument:

```
mysql> CALL sys.ps_setup_disable_instrument('wait/lock/metadata/sql/mdl');
+-----------------------+
| summary               |
+-----------------------+
| Disabled 1 instrument |
+-----------------------+
```

Disable all mutex instruments:

```
mysql> CALL sys.ps_setup_disable_instrument('mutex');
+--------------------------+
| summary                  |
+--------------------------+
| Disabled 177 instruments |
+--------------------------+
```

27.4.4.7 The ps_setup_disable_thread() Procedure

Given a connection ID, disables Performance Schema instrumentation for the thread. Produces a result set indicating how many threads were disabled. Already disabled threads do not count.

Parameters

- `in_connection_id BIGINT`: The connection ID. This is a value of the type given in the `PROCESSLIST_ID` column of the Performance Schema `threads` table or the `Id` column of `SHOW PROCESSLIST` output.

Example

Disable a specific connection by its connection ID:

```
mysql> CALL sys.ps_setup_disable_thread(225);
+-------------------+
| summary           |
+-------------------+
| Disabled 1 thread |
+-------------------+
```

Disable the current connection:

```
mysql> CALL sys.ps_setup_disable_thread(CONNECTION_ID());
+-------------------+
| summary           |
+-------------------+
| Disabled 1 thread |
+-------------------+
```

27.4.4.8 The ps_setup_enable_background_threads() Procedure

Enables Performance Schema instrumentation for all background threads. Produces a result set indicating how many background threads were enabled. Already enabled threads do not count.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_enable_background_threads();
+------------------------------+
| summary                      |
+------------------------------+
| Enabled 24 background threads |
+------------------------------+
```

27.4.4.9 The ps_setup_enable_consumer() Procedure

Enables Performance Schema consumers with names that contain the argument. Produces a result set indicating how many consumers were enabled. Already enabled consumers do not count.

Parameters

- `consumer VARCHAR(128)`: The value used to match consumer names, which are identified by using `%consumer%` as an operand for a `LIKE` pattern match.

  A value of `''` matches all consumers.

Example

Enable all statement consumers:

```
mysql> CALL sys.ps_setup_enable_consumer('statement');
+--------------------+
| summary            |
+--------------------+
| Enabled 4 consumers |
+--------------------+
```

27.4.4.10 The ps_setup_enable_instrument() Procedure

Enables Performance Schema instruments with names that contain the argument. Produces a result set indicating how many instruments were enabled. Already enabled instruments do not count.

Parameters

- `in_pattern VARCHAR(128)`: The value used to match instrument names, which are identified by using `%in_pattern%` as an operand for a `LIKE` pattern match.

  A value of `''` matches all instruments.

Example

Enable a specific instrument:

```
mysql> CALL sys.ps_setup_enable_instrument('wait/lock/metadata/sql/mdl');
+----------------------+
| summary              |
+----------------------+
| Enabled 1 instrument |
+----------------------+
```

Enable all mutex instruments:

```
mysql> CALL sys.ps_setup_enable_instrument('mutex');
+-------------------------+
| summary                 |
+-------------------------+
| Enabled 177 instruments |
+-------------------------+
```

27.4.4.11 The ps_setup_enable_thread() Procedure

Given a connection ID, enables Performance Schema instrumentation for the thread. Produces a result set indicating how many threads were enabled. Already enabled threads do not count.

Parameters

- `in_connection_id BIGINT`: The connection ID. This is a value of the type given in the `PROCESSLIST_ID` column of the Performance Schema `threads` table or the `Id` column of `SHOW PROCESSLIST` output.

Example

Enable a specific connection by its connection ID:

```
mysql> CALL sys.ps_setup_enable_thread(225);
+------------------+
| summary          |
+------------------+
| Enabled 1 thread |
+------------------+
```

Enable the current connection:

```
mysql> CALL sys.ps_setup_enable_thread(CONNECTION_ID());
+------------------+
| summary          |
+------------------+
| Enabled 1 thread |
+------------------+
```

27.4.4.12 The ps_setup_reload_saved() Procedure

Reloads a Performance Schema configuration saved earlier within the same session using `ps_setup_save()`. For more information, see the description of `ps_setup_save()`.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Parameters

None.

27.4.4.13 The ps_setup_reset_to_default() Procedure

Resets the Performance Schema configuration to its default settings.

Parameters

- `in_verbose BOOLEAN`: Whether to display information about each setup stage during procedure execution. This includes the SQL statements executed.

Example

```
mysql> CALL sys.ps_setup_reset_to_default(TRUE)\G
*************************** 1. row ***************************
status: Resetting: setup_actors
DELETE
FROM performance_schema.setup_actors
WHERE NOT (HOST = '%' AND USER = '%' AND ROLE = '%')

*************************** 1. row ***************************
status: Resetting: setup_actors
INSERT IGNORE INTO performance_schema.setup_actors
VALUES ('%', '%', '%')

...
```

27.4.4.14 The ps_setup_save() Procedure

Saves the current Performance Schema configuration. This enables you to alter the configuration temporarily for debugging or other purposes, then restore it to the previous state by invoking the `ps_setup_reload_saved()` procedure.

To prevent other simultaneous calls to save the configuration, `ps_setup_save()` acquires an advisory lock named `sys.ps_setup_save` by calling the `GET_LOCK()` function. `ps_setup_save()` takes a timeout parameter to indicate how many seconds to wait if the lock already exists (which indicates that some other session has a saved configuration outstanding). If the timeout expires without obtaining the lock, `ps_setup_save()` fails.

It is intended you call `ps_setup_reload_saved()` later within the *same* session as `ps_setup_save()` because the configuration is saved in `TEMPORARY` tables. `ps_setup_save()` drops the temporary tables and releases the lock. If you end your session without invoking `ps_setup_save()`, the tables and lock disappear automatically.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Parameters

- `in_timeout INT`: How many seconds to wait to obtain the `sys.ps_setup_save` lock. A negative timeout value means infinite timeout.

Example

```
mysql> CALL sys.ps_setup_save(10);

... make Performance Schema configuration changes ...

mysql> CALL sys.ps_setup_reload_saved();
```

27.4.4.15 The ps_setup_show_disabled() Procedure

Displays all currently disabled Performance Schema configuration.

Parameters

- `in_show_instruments BOOLEAN`: Whether to display disabled instruments. This might be a long list.
- `in_show_threads BOOLEAN`: Whether to display disabled threads.

Example

```
mysql> CALL sys.ps_setup_show_disabled(TRUE, TRUE);
+----------------------------+
| performance_schema_enabled |
+----------------------------+
|                          1 |
+----------------------------+

+---------------+
| enabled_users |
+---------------+
| '%'@'%'       |
+---------------+

+-------------+---------------------+---------+-------+
| object_type | objects             | enabled | timed |
+-------------+---------------------+---------+-------+
| EVENT       | mysql.%             | NO      | NO    |
| EVENT       | performance_schema.% | NO      | NO    |
| EVENT       | information_schema.% | NO      | NO    |
```

```
| FUNCTION    | mysql.%              | NO      | NO    |
| FUNCTION    | performance_schema.% | NO      | NO    |
| FUNCTION    | information_schema.% | NO      | NO    |
| PROCEDURE   | mysql.%              | NO      | NO    |
| PROCEDURE   | performance_schema.% | NO      | NO    |
| PROCEDURE   | information_schema.% | NO      | NO    |
| TABLE       | mysql.%              | NO      | NO    |
| TABLE       | performance_schema.% | NO      | NO    |
| TABLE       | information_schema.% | NO      | NO    |
| TRIGGER     | mysql.%              | NO      | NO    |
| TRIGGER     | performance_schema.% | NO      | NO    |
| TRIGGER     | information_schema.% | NO      | NO    |
+-------------+----------------------+---------+-------+

...
```

27.4.4.16 The ps_setup_show_disabled_consumers() Procedure

Displays all currently disabled Performance Schema consumers.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_show_disabled_consumers();
+----------------------------------+
| disabled_consumers               |
+----------------------------------+
| events_stages_current            |
| events_stages_history            |
| events_stages_history_long       |
| events_statements_history        |
| events_statements_history_long   |
| events_transactions_history      |
| events_transactions_history_long |
| events_waits_current             |
| events_waits_history             |
| events_waits_history_long        |
+----------------------------------+
```

27.4.4.17 The ps_setup_show_disabled_instruments() Procedure

Displays all currently disabled Performance Schema instruments. This might be a long list.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_show_disabled_instruments()\G
*************************** 1. row ***************************
disabled_instruments: wait/synch/mutex/sql/TC_LOG_MMAP::LOCK_tc
               timed: NO
*************************** 2. row ***************************
disabled_instruments: wait/synch/mutex/sql/THD::LOCK_query_plan
               timed: NO
*************************** 3. row ***************************
disabled_instruments: wait/synch/mutex/sql/MYSQL_BIN_LOG::LOCK_commit
               timed: NO
```

```
...
```

27.4.4.18 The ps_setup_show_enabled() Procedure

Displays all currently enabled Performance Schema configuration.

Parameters

- `in_show_instruments BOOLEAN`: Whether to display enabled instruments. This might be a long list.
- `in_show_threads BOOLEAN`: Whether to display enabled threads.

Example

```
mysql> CALL sys.ps_setup_show_enabled(FALSE, FALSE);
+----------------------------+
| performance_schema_enabled |
+----------------------------+
|                          1 |
+----------------------------+
1 row in set (0.01 sec)

+---------------+
| enabled_users |
+---------------+
| '%'@'%'       |
+---------------+
1 row in set (0.01 sec)

+-------------+---------+---------+-------+
| object_type | objects | enabled | timed |
+-------------+---------+---------+-------+
| EVENT       | %.%     | YES     | YES   |
| FUNCTION    | %.%     | YES     | YES   |
| PROCEDURE   | %.%     | YES     | YES   |
| TABLE       | %.%     | YES     | YES   |
| TRIGGER     | %.%     | YES     | YES   |
+-------------+---------+---------+-------+
5 rows in set (0.02 sec)

+----------------------------+
| enabled_consumers          |
+----------------------------+
| events_statements_current  |
| events_statements_history  |
| events_transactions_current |
| events_transactions_history |
| global_instrumentation     |
| statements_digest          |
| thread_instrumentation     |
+----------------------------+
```

27.4.4.19 The ps_setup_show_enabled_consumers() Procedure

Displays all currently enabled Performance Schema consumers.

Parameters

None.

Example

```

```

```
mysql> CALL sys.ps_setup_show_enabled_consumers();
+----------------------------+
| enabled_consumers          |
+----------------------------+
| events_statements_current  |
| events_statements_history  |
| events_transactions_current |
| events_transactions_history |
| global_instrumentation     |
| statements_digest          |
| thread_instrumentation     |
+----------------------------+
```

27.4.4.20 The ps_setup_show_enabled_instruments() Procedure

Displays all currently enabled Performance Schema instruments. This might be a long list.

Parameters

None.

Example

```
mysql> CALL sys.ps_setup_show_enabled_instruments()\G
*************************** 1. row ***************************
enabled_instruments: wait/io/file/sql/map
              timed: YES
*************************** 2. row ***************************
enabled_instruments: wait/io/file/sql/binlog
              timed: YES
*************************** 3. row ***************************
enabled_instruments: wait/io/file/sql/binlog_cache
              timed: YES
...
```

27.4.4.21 The ps_statement_avg_latency_histogram() Procedure

Displays a textual histogram graph of the average latency values across all normalized statements tracked within the Performance Schema `events_statements_summary_by_digest` table.

This procedure can be used to display a very high-level picture of the latency distribution of statements running within this MySQL instance.

Parameters

None.

Example

The histogram output in statement units. For example, `* = 2 units` in the histogram legend means that each `*` character represents 2 statements.

```
mysql> CALL sys.ps_statement_avg_latency_histogram()\G
*************************** 1. row ***************************
Performance Schema Statement Digest Average Latency Histogram:

  . = 1 unit
  * = 2 units
  # = 3 units
```

```
(0 - 66ms)     88  | ############################
(66 - 133ms)   14  | ..............
(133 - 199ms)  4   | ....
(199 - 265ms)  5   | **
(265 - 332ms)  1   | .
(332 - 398ms)  0   |
(398 - 464ms)  1   | .
(464 - 531ms)  0   |
(531 - 597ms)  0   |
(597 - 663ms)  0   |
(663 - 730ms)  0   |
(730 - 796ms)  0   |
(796 - 863ms)  0   |
(863 - 929ms)  0   |
(929 - 995ms)  0   |
(995 - 1062ms) 0   |

  Total Statements: 114; Buckets: 16; Bucket Size: 66 ms;
```

27.4.4.22 The ps_trace_statement_digest() Procedure

Traces all Performance Schema instrumentation for a specific statement digest.

If you find a statement of interest within the Performance Schema `events_statements_summary_by_digest` table, specify its `DIGEST` column MD5 value to this procedure and indicate the polling duration and interval. The result is a report of all statistics tracked within Performance Schema for that digest for the interval.

The procedure also attempts to execute `EXPLAIN` for the longest running example of the digest during the interval. This attempt might fail because the Performance Schema truncates long `SQL_TEXT` values. Consequently, `EXPLAIN` will fail due to parse errors.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Parameters

- `in_digest VARCHAR(32)`: The statement digest identifier to analyze.
- `in_runtime INT`: How long to run the analysis in seconds.
- `in_interval DECIMAL(2,2)`: The analysis interval in seconds (which can be fractional) at which to try to take snapshots.
- `in_start_fresh BOOLEAN`: Whether to truncate the Performance Schema `events_statements_history_long` and `events_stages_history_long` tables before starting.
- `in_auto_enable BOOLEAN`: Whether to automatically enable required consumers.

Example

```
mysql> CALL sys.ps_trace_statement_digest('891ec6860f98ba46d89dd20b0c03652c', 10, 0.1, TRUE, TRUE);
+--------------------+
| SUMMARY STATISTICS |
+--------------------+
| SUMMARY STATISTICS |
+--------------------+
1 row in set (9.11 sec)
```

```
+------------+-----------+-----------+-----------+---------------+------------+------------+
| executions | exec_time | lock_time | rows_sent | rows_examined | tmp_tables | full_scans |
+------------+-----------+-----------+-----------+---------------+------------+------------+
|         21 | 4.11 ms   | 2.00 ms   |         0 |            21 |          0 |          0 |
+------------+-----------+-----------+-----------+---------------+------------+------------+
1 row in set (9.11 sec)

+----------------------------------------+-------+-----------+
| event_name                             | count | latency   |
+----------------------------------------+-------+-----------+
| stage/sql/statistics                   |    16 | 546.92 us |
| stage/sql/freeing items                |    18 | 520.11 us |
| stage/sql/init                         |    51 | 466.80 us |
...
| stage/sql/cleaning up                  |    18 | 11.92 us  |
| stage/sql/executing                    |    16 | 6.95 us   |
+----------------------------------------+-------+-----------+
17 rows in set (9.12 sec)

+--------------------------+
| LONGEST RUNNING STATEMENT |
+--------------------------+
| LONGEST RUNNING STATEMENT |
+--------------------------+
1 row in set (9.16 sec)

+-----------+-----------+-----------+-----------+---------------+------------+-----------+
| thread_id | exec_time | lock_time | rows_sent | rows_examined | tmp_tables | full_scan |
+-----------+-----------+-----------+-----------+---------------+------------+-----------+
|    166646 | 618.43 us | 1.00 ms   |         0 |             1 |          0 |         0 |
+-----------+-----------+-----------+-----------+---------------+------------+-----------+
1 row in set (9.16 sec)

# Truncated for clarity...
+-------------------------------------------------------------------+
| sql_text                                                          |
+-------------------------------------------------------------------+
| select hibeventhe0_.id as id1382_, hibeventhe0_.createdTime ... |
+-------------------------------------------------------------------+
1 row in set (9.17 sec)

+----------------------------------------+-----------+
| event_name                             | latency   |
+----------------------------------------+-----------+
| stage/sql/init                         | 8.61 us   |
| stage/sql/init                         | 331.07 ns |
...
| stage/sql/freeing items                | 30.46 us  |
| stage/sql/cleaning up                  | 662.13 ns |
+----------------------------------------+-----------+
18 rows in set (9.23 sec)

+----+-------------+--------------+-------+---------------+-----------+---------+-------------+------+-------+
| id | select_type | table        | type  | possible_keys | key       | key_len | ref         | rows | Extra |
+----+-------------+--------------+-------+---------------+-----------+---------+-------------+------+-------+
|  1 | SIMPLE      | hibeventhe0_ | const | fixedTime     | fixedTime | 775     | const,const |    1 | NULL  |
+----+-------------+--------------+-------+---------------+-----------+---------+-------------+------+-------+
1 row in set (9.27 sec)

Query OK, 0 rows affected (9.28 sec)
```

27.4.4.23 The ps_trace_thread() Procedure

Dumps all Performance Schema data for an instrumented thread to a `.dot` formatted graph file (for the DOT graph description language). Each result set returned from the procedure should be used for a complete graph.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Parameters

- `in_thread_id INT`: The thread to trace.
- `in_outfile VARCHAR(255)`: The name to use for the `.dot` output file.
- `in_max_runtime DECIMAL(20,2)`: The maximum number of seconds (which can be fractional) to collect data. Use `NULL` to collect data for the default of 60 seconds.
- `in_interval DECIMAL(20,2)`: The number of seconds (which can be fractional) to sleep between data collections. Use `NULL` to sleep for the default of 1 second.
- `in_start_fresh BOOLEAN`: Whether to reset all Performance Schema data before tracing.
- `in_auto_setup BOOLEAN`: Whether to disable all other threads and enable all instruments and consumers. This also resets the settings at the end of the run.
- `in_debug BOOLEAN`: Whether to include `file:lineno` information in the graph.

Example

```
mysql> CALL sys.ps_trace_thread(25, CONCAT('/tmp/stack-', REPLACE(NOW(), ' ', '-'), '.dot'), NULL, NULL, TR
+-------------------+
| summary           |
+-------------------+
| Disabled 1 thread |
+-------------------+
1 row in set (0.00 sec)

+-----------------------------------------+
| Info                                    |
+-----------------------------------------+
| Data collection starting for THREAD_ID = 25 |
+-----------------------------------------+
1 row in set (0.03 sec)

+------------------------------------------------------------+
| Info                                                       |
+------------------------------------------------------------+
| Stack trace written to /tmp/stack-2014-02-16-21:18:41.dot |
+------------------------------------------------------------+
1 row in set (60.07 sec)

+-----------------------------------------------------------------------+
| Convert to PDF                                                        |
+-----------------------------------------------------------------------+
| dot -Tpdf -o /tmp/stack_25.pdf /tmp/stack-2014-02-16-21:18:41.dot |
+-----------------------------------------------------------------------+
1 row in set (60.07 sec)

+-----------------------------------------------------------------------+
| Convert to PNG                                                        |
+-----------------------------------------------------------------------+
| dot -Tpng -o /tmp/stack_25.png /tmp/stack-2014-02-16-21:18:41.dot |
+-----------------------------------------------------------------------+
1 row in set (60.07 sec)

+------------------+
| summary          |
```

```
+------------------+
| Enabled 1 thread |
+------------------+
1 row in set (60.32 sec)
```

27.4.4.24 The ps_truncate_all_tables() Procedure

Truncates all Performance Schema summary tables, resetting all aggregated instrumentation as a snapshot. Produces a result set indicating how many tables were truncated.

Parameters

- `in_verbose BOOLEAN`: Whether to display each `TRUNCATE TABLE` statement before executing it.

Example

```
mysql> CALL sys.ps_truncate_all_tables(FALSE);
+---------------------+
| summary             |
+---------------------+
| Truncated 49 tables |
+---------------------+
```

27.4.4.25 The statement_performance_analyzer() Procedure

Creates a report of the statements running on the server. The views are calculated based on the overall and/or delta activity.

This procedure disables binary logging during its execution by manipulating the session value of the `sql_log_bin` system variable. That is a restricted operation, so the procedure requires privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges”.

Parameters

- `in_action ENUM('snapshot', 'overall', 'delta', 'create_tmp', 'create_table', 'save', 'cleanup')`: The action to take. These values are permitted:
  - `snapshot`: Store a snapshot. The default is to make a snapshot of the current content of the Performance Schema `events_statements_summary_by_digest` table. By setting `in_table`, this can be overwritten to copy the content of the specified table. The snapshot is stored in the `sys` schema `tmp_digests` temporary table.
  - `overall`: Generate an analysis based on the content of the table specified by `in_table`. For the overall analysis, `in_table` can be `NOW()` to use a fresh snapshot. This overwrites an existing snapshot. Use `NULL` for `in_table` to use the existing snapshot. If `in_table` is `NULL` and no snapshot exists, a new snapshot is created. The `in_views` parameter and the `statement_performance_analyzer.limit` configuration option affect the operation of this procedure.
  - `delta`: Generate a delta analysis. The delta is calculated between the reference table specified by `in_table` and the snapshot, which must exist. This action uses the `sys` schema `tmp_digests_delta` temporary table. The `in_views` parameter and the `statement_performance_analyzer.limit` configuration option affect the operation of this procedure.
  - `create_table`: Create a regular table suitable for storing the snapshot for later use (for example, for calculating deltas).

  - `create_tmp`: Create a temporary table suitable for storing the snapshot for later use (for example, for calculating deltas).
  - `save`: Save the snapshot in the table specified by `in_table`. The table must exist and have the correct structure. If no snapshot exists, a new snapshot is created.
  - `cleanup`: Remove the temporary tables used for the snapshot and delta.
- `in_table VARCHAR(129)`: The table parameter used for some of the actions specified by the `in_action` parameter. Use the format *`db_name.tbl_name`* or *`tbl_name`* without using any backtick (`` ` ``) identifier-quoting characters. Periods (`.`) are not supported in database and table names.

  The meaning of the `in_table` value for each `in_action` value is detailed in the individual `in_action` value descriptions.
- `in_views SET ('with_runtimes_in_95th_percentile', 'analysis', 'with_errors_or_warnings', 'with_full_table_scans', 'with_sorting', 'with_temp_tables', 'custom')`: Which views to include. This parameter is a `SET` value, so it can contain multiple view names, separated by commas. The default is to include all views except `custom`. The following values are permitted:
  - `with_runtimes_in_95th_percentile`: Use the `statements_with_runtimes_in_95th_percentile` view.
  - `analysis`: Use the `statement_analysis` view.
  - `with_errors_or_warnings`: Use the `statements_with_errors_or_warnings` view.
  - `with_full_table_scans`: Use the `statements_with_full_table_scans` view.
  - `with_sorting`: Use the `statements_with_sorting` view.
  - `with_temp_tables`: Use the `statements_with_temp_tables` view.
  - `custom`: Use a custom view. This view must be specified using the `statement_performance_analyzer.view` configuration option to name a query or an existing view.

Configuration Options

`statement_performance_analyzer()` operation can be modified using the following configuration options or their corresponding user-defined variables (see Section 27.4.2.1, “The sys_config Table”):

- `debug`, `@sys.debug`

  If this option is `ON`, produce debugging output. The default is `OFF`.
- `statement_performance_analyzer.limit`, `@sys.statement_performance_analyzer.limit`

  The maximum number of rows to return for views that have no built-in limit. The default is 100.
- `statement_performance_analyzer.view`, `@sys.statement_performance_analyzer.view`

  The custom query or view to be used. If the option value contains a space, it is interpreted as a query. Otherwise, it must be the name of an existing view that queries the Performance Schema `events_statements_summary_by_digest` table. There cannot be any `LIMIT` clause in the query

or view definition if the `statement_performance_analyzer.limit` configuration option is greater than 0. If specifying a view, use the same format as for the `in_table` parameter. The default is `NULL` (no custom view defined).

Example

To create a report with the queries in the 95th percentile since the last truncation of `events_statements_summary_by_digest` and with a one-minute delta period:

1. Create a temporary table to store the initial snapshot.
2. Create the initial snapshot.
3. Save the initial snapshot in the temporary table.
4. Wait one minute.
5. Create a new snapshot.
6. Perform analysis based on the new snapshot.
7. Perform analysis based on the delta between the initial and new snapshots.

```
mysql> CALL sys.statement_performance_analyzer('create_tmp', 'mydb.tmp_digests_ini', NULL);
Query OK, 0 rows affected (0.08 sec)

mysql> CALL sys.statement_performance_analyzer('snapshot', NULL, NULL);
Query OK, 0 rows affected (0.02 sec)

mysql> CALL sys.statement_performance_analyzer('save', 'mydb.tmp_digests_ini', NULL);
Query OK, 0 rows affected (0.00 sec)

mysql> DO SLEEP(60);
Query OK, 0 rows affected (1 min 0.00 sec)

mysql> CALL sys.statement_performance_analyzer('snapshot', NULL, NULL);
Query OK, 0 rows affected (0.02 sec)

mysql> CALL sys.statement_performance_analyzer('overall', NULL, 'with_runtimes_in_95th_percentile');
+----------------------------------------+
| Next Output                            |
+----------------------------------------+
| Queries with Runtime in 95th Percentile |
+----------------------------------------+
1 row in set (0.05 sec)

...

mysql> CALL sys.statement_performance_analyzer('delta', 'mydb.tmp_digests_ini', 'with_runtimes_in_95th_percent
+----------------------------------------+
| Next Output                            |
+----------------------------------------+
| Queries with Runtime in 95th Percentile |
+----------------------------------------+
1 row in set (0.03 sec)

...
```

Create an overall report of the 95th percentile queries and the top 10 queries with full table scans:

```
mysql> CALL sys.statement_performance_analyzer('snapshot', NULL, NULL);
```

```
Query OK, 0 rows affected (0.01 sec)

mysql> SET @sys.statement_performance_analyzer.limit = 10;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL sys.statement_performance_analyzer('overall', NULL, 'with_runtimes_in_95th_percentile,with_ful
+----------------------------------------+
| Next Output                            |
+----------------------------------------+
| Queries with Runtime in 95th Percentile |
+----------------------------------------+
1 row in set (0.01 sec)

...

+-------------------------------------+
| Next Output                         |
+-------------------------------------+
| Top 10 Queries with Full Table Scan |
+-------------------------------------+
1 row in set (0.09 sec)

...
```

Use a custom view showing the top 10 queries sorted by total execution time, refreshing the view every minute using the `watch` command in Linux:

```
mysql> CREATE OR REPLACE VIEW mydb.my_statements AS
       SELECT sys.format_statement(DIGEST_TEXT) AS query,
              SCHEMA_NAME AS db,
              COUNT_STAR AS exec_count,
              sys.format_time(SUM_TIMER_WAIT) AS total_latency,
              sys.format_time(AVG_TIMER_WAIT) AS avg_latency,
              ROUND(IFNULL(SUM_ROWS_SENT / NULLIF(COUNT_STAR, 0), 0)) AS rows_sent_avg,
              ROUND(IFNULL(SUM_ROWS_EXAMINED / NULLIF(COUNT_STAR, 0), 0)) AS rows_examined_avg,
              ROUND(IFNULL(SUM_ROWS_AFFECTED / NULLIF(COUNT_STAR, 0), 0)) AS rows_affected_avg,
              DIGEST AS digest
         FROM performance_schema.events_statements_summary_by_digest
       ORDER BY SUM_TIMER_WAIT DESC;
Query OK, 0 rows affected (0.10 sec)

mysql> CALL sys.statement_performance_analyzer('create_table', 'mydb.digests_prev', NULL);
Query OK, 0 rows affected (0.10 sec)

shell> watch -n 60 "mysql sys --table -e \"
> SET @sys.statement_performance_analyzer.view = 'mydb.my_statements';
> SET @sys.statement_performance_analyzer.limit = 10;
> CALL statement_performance_analyzer('snapshot', NULL, NULL);
> CALL statement_performance_analyzer('delta', 'mydb.digests_prev', 'custom');
> CALL statement_performance_analyzer('save', 'mydb.digests_prev', NULL);
> \""

Every 60.0s: mysql sys --table -e "        ...  Mon Dec 22 10:58:51 2014

+---------------------------------+
| Next Output                     |
+---------------------------------+
| Top 10 Queries Using Custom View |
+---------------------------------+
+------------------+-------+------------+---------------+-------------+---------------+------------------
| query            | db    | exec_count | total_latency | avg_latency | rows_sent_avg | rows_examined_avg
+------------------+-------+------------+---------------+-------------+---------------+------------------
...
```

27.4.4.26 The table_exists() Procedure

Tests whether a given table exists as a regular table, a `TEMPORARY` table, or a view. The procedure returns the table type in an `OUT` parameter. If both a temporary and a permanent table exist with the given name, `TEMPORARY` is returned.

Parameters

- `in_db VARCHAR(64)`: The name of the database in which to check for table existance.
- `in_table VARCHAR(64)`: The name of the table to check the existance of.
- `out_exists ENUM('', 'BASE TABLE', 'VIEW', 'TEMPORARY')`: The return value. This is an `OUT` parameter, so it must be a variable into which the table type can be stored. When the procedure returns, the variable has one of the following values to indicate whether the table exists:
  - `''`: The table name does not exist as a base table, `TEMPORARY` table, or view.
  - `BASE TABLE`: The table name exists as a base (permanent) table.
  - `VIEW`: The table name exists as a view.
  - `TEMPORARY`: The table name exists as a `TEMPORARY` table.

Example

```
mysql> CREATE DATABASE db1;
Query OK, 1 row affected (0.01 sec)

mysql> USE db1;
Database changed
mysql> CREATE TABLE t1 (id INT PRIMARY KEY);
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE t2 (id INT PRIMARY KEY);
Query OK, 0 rows affected (0.20 sec)

mysql> CREATE view v_t1 AS SELECT * FROM t1;
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TEMPORARY TABLE t1 (id INT PRIMARY KEY);
Query OK, 0 rows affected (0.00 sec)

mysql> CALL sys.table_exists('db1', 't1', @exists); SELECT @exists;
Query OK, 0 rows affected (0.01 sec)

+-----------+
| @exists   |
+-----------+
| TEMPORARY |
+-----------+
1 row in set (0.00 sec)

mysql> CALL sys.table_exists('db1', 't2', @exists); SELECT @exists;
Query OK, 0 rows affected (0.02 sec)

+------------+
| @exists    |
+------------+
| BASE TABLE |
+------------+
1 row in set (0.00 sec)
```

```
mysql> CALL sys.table_exists('db1', 'v_t1', @exists); SELECT @exists;
Query OK, 0 rows affected (0.02 sec)

+---------+
| @exists |
+---------+
| VIEW    |
+---------+
1 row in set (0.00 sec)

mysql> CALL sys.table_exists('db1', 't3', @exists); SELECT @exists;
Query OK, 0 rows affected (0.00 sec)

+---------+
| @exists |
+---------+
|         |
+---------+
1 row in set (0.00 sec)
```

27.4.5 sys Schema Stored Functions

The following sections describe `sys` schema stored functions.

27.4.5.1 The extract_schema_from_file_name() Function

Given a file path name, returns the path component that represents the schema name. This function assumes that the file name lies within the schema directory. For this reason, it will not work with partitions or tables defined using their own `DATA_DIRECTORY` table option.

This function is useful when extracting file I/O information from the Performance Schema that includes file path names. It provides a convenient way to display schema names, which can be more easily understood than full path names, and can be used in joins against object schema names.

Parameters

- `path VARCHAR(512)`: The full path to a data file from which to extract the schema name.

Return Value

A `VARCHAR(64)` value.

Example

```
mysql> SELECT sys.extract_schema_from_file_name('/usr/local/mysql/data/world/City.ibd');
+-----------------------------------------------------------------------+
| sys.extract_schema_from_file_name('/usr/local/mysql/data/world/City.ibd') |
+-----------------------------------------------------------------------+
| world                                                                 |
+-----------------------------------------------------------------------+
```

27.4.5.2 The extract_table_from_file_name() Function

Given a file path name, returns the path component that represents the table name.

This function is useful when extracting file I/O information from the Performance Schema that includes file path names. It provides a convenient way to display table names, which can be more easily understood than full path names, and can be used in joins against object table names.

Parameters

- `path VARCHAR(512)`: The full path to a data file from which to extract the table name.

Return Value

A `VARCHAR(64)` value.

Example

```
mysql> SELECT sys.extract_table_from_file_name('/usr/local/mysql/data/world/City.ibd');
+--------------------------------------------------------------------------+
| sys.extract_table_from_file_name('/usr/local/mysql/data/world/City.ibd') |
+--------------------------------------------------------------------------+
| City                                                                     |
+--------------------------------------------------------------------------+
```

27.4.5.3 The format_bytes() Function

Note

As of MySQL 8.0.16, `format_bytes()` is deprecated and will be removed in a future MySQL version. Applications that use it should be migrated to use the built-in `FORMAT_BYTES()` function instead. See Section 12.22, “Performance Schema Functions”

Given a byte count, converts it to human-readable format and returns a string consisting of a value and a units indicator. Depending on the size of the value, the units part is `bytes`, `KiB` (kibibytes), `MiB` (mebibytes), `GiB` (gibibytes), `TiB` (tebibytes), or `PiB` (pebibytes).

Parameters

- `bytes TEXT`: The byte count to format.

Return Value

A `TEXT` value.

Example

```
mysql> SELECT sys.format_bytes(512), sys.format_bytes(18446644073709551615);
+-----------------------+---------------------------------------+
| sys.format_bytes(512) | sys.format_bytes(18446644073709551615) |
+-----------------------+---------------------------------------+
| 512 bytes             | 16383.91 PiB                          |
+-----------------------+---------------------------------------+
```

27.4.5.4 The format_path() Function

Given a path name, returns the modified path name after replacing subpaths that match the values of the following system variables, in order:

```
datadir
tmpdir
slave_load_tmpdir
innodb_data_home_dir
innodb_log_group_home_dir
```

```
innodb_undo_directory
basedir
```

A value that matches the value of system variable *`sysvar`* is replaced with the string `@@GLOBAL.sysvar`.

Parameters

- `path VARCHAR(512)`: The path name to format.

Return Value

A `VARCHAR(512) CHARACTER SET utf8` value.

Example

```
mysql> SELECT sys.format_path('/usr/local/mysql/data/world/City.ibd');
+-----------------------------------------------------+
| sys.format_path('/usr/local/mysql/data/world/City.ibd') |
+-----------------------------------------------------+
| /usr/local/mysql/data/world/City.ibd                |
+-----------------------------------------------------+
```

27.4.5.5 The format_statement() Function

Given a string (normally representing an SQL statement), reduces it to the length given by the `statement_truncate_len` configuration option, and returns the result. No truncation occurs if the string is shorter than `statement_truncate_len`. Otherwise, the middle part of the string is replaced by an ellipsis (`...`).

This function is useful for formatting possibly lengthy statements retrieved from Performance Schema tables to a known fixed maximum length.

Parameters

- `statement LONGTEXT`: The statement to format.

Configuration Options

`format_statement()` operation can be modified using the following configuration options or their corresponding user-defined variables (see Section 27.4.2.1, “The sys_config Table”):

- `statement_truncate_len`, `@sys.statement_truncate_len`

  The maximum length of statements returned by the `format_statement()` function. Longer statements are truncated to this length. The default is 64.

Return Value

A `LONGTEXT` value.

Example

By default, `format_statement()` truncates statements to be no more than 64 characters. Setting `@sys.statement_truncate_len` changes the truncation length for the current session:

```
mysql> SET @stmt = 'SELECT variable, value, set_time, set_by FROM sys_config';
mysql> SELECT sys.format_statement(@stmt);
+-----------------------------------------------------------+
```

```
| sys.format_statement(@stmt)                              |
+----------------------------------------------------------+
| SELECT variable, value, set_time, set_by FROM sys_config |
+----------------------------------------------------------+
mysql> SET @sys.statement_truncate_len = 32;
mysql> SELECT sys.format_statement(@stmt);
+-----------------------------------+
| sys.format_statement(@stmt)       |
+-----------------------------------+
| SELECT variabl ... ROM sys_config |
+-----------------------------------+
```

27.4.5.6 The format_time() Function

Note

As of MySQL 8.0.16, `format_time()` is deprecated and will be removed in a future MySQL version. Applications that use it should be migrated to use the built-in `FORMAT_PICO_TIME()` function instead. See Section 12.22, “Performance Schema Functions”

Given a Performance Schema latency or wait time in picoseconds, converts it to human-readable format and returns a string consisting of a value and a units indicator. Depending on the size of the value, the units part is `ps` (nanoseconds), `ns` (nanoseconds), `us` (microseconds), `ms` (milliseconds), `s` (seconds), `m` (minutes), `h` (hours), `d` (days), or `w` (weeks).

Parameters

- `picoseconds TEXT`: The picoseconds value to format.

Return Value

A `TEXT` value.

Example

```
mysql> SELECT sys.format_time(3501), sys.format_time(188732396662000);
+----------------------+-----------------------------------+
| sys.format_time(3501) | sys.format_time(188732396662000) |
+----------------------+-----------------------------------+
| 3.50 ns              | 3.15 m                            |
+----------------------+-----------------------------------+
```

27.4.5.7 The list_add() Function

Adds a value to a comma-separated list of values and returns the result.

This function and `list_drop()` can be useful for manipulating the value of system variables such as `sql_mode` and `optimizer_switch` that take a comma-separated list of values.

Parameters

- `in_list TEXT`: The list to be modified.
- `in_add_value TEXT`: The value to add to the list.

Return Value

A `TEXT` value.

Example

```
mysql> SELECT @@sql_mode;
+-------------------------------------+
| @@sql_mode                          |
+-------------------------------------+
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES |
+-------------------------------------+
mysql> SET @@sql_mode = sys.list_add(@@sql_mode, 'NO_ENGINE_SUBSTITUTION');
mysql> SELECT @@sql_mode;
+---------------------------------------------------------------+
| @@sql_mode                                                    |
+---------------------------------------------------------------+
| ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION |
+---------------------------------------------------------------+
mysql> SET @@sql_mode = sys.list_drop(@@sql_mode, 'ONLY_FULL_GROUP_BY');
mysql> SELECT @@sql_mode;
+-----------------------------------------+
| @@sql_mode                              |
+-----------------------------------------+
| STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION |
+-----------------------------------------+
```

27.4.5.8 The list_drop() Function

Removes a value from a comma-separated list of values and returns the result. For more information, see the description of `list_add()`

Parameters

- `in_list TEXT`: The list to be modified.
- `in_drop_value TEXT`: The value to drop from the list.

Return Value

A `TEXT` value.

27.4.5.9 The ps_is_account_enabled() Function

Returns `YES` or `NO` to indicate whether Performance Schema instrumentation for a given account is enabled.

Parameters

- `in_host VARCHAR(60)`: The host name of the account to check.
- `in_user VARCHAR(32)`: The user name of the account to check.

Return Value

An `ENUM('YES','NO')` value.

Example

```
mysql> SELECT sys.ps_is_account_enabled('localhost', 'root');
+-----------------------------------------------+
| sys.ps_is_account_enabled('localhost', 'root') |
+-----------------------------------------------+
```

```
| YES                                            |
+------------------------------------------------+
```

27.4.5.10 The ps_is_consumer_enabled() Function

Returns `YES` or `NO` to indicate whether a given Performance Schema consumer is enabled, or `NULL` if the argument is `NULL`. If the argument is not a valid consumer name, an error occurs. (Prior to MySQL 8.0.18, the function returns `NULL` if the argument is not a valid consumer name.)

This function accounts for the consumer hierarchy, so a consumer is not considered enabled unless all consumers on which depends are also enabled. For information about the consumer hierarchy, see Section 26.4.7, “Pre-Filtering by Consumer”.

Parameters

- `in_consumer VARCHAR(64)`: The name of the consumer to check.

Return Value

An `ENUM('YES','NO')` value.

Example

```
mysql> SELECT sys.ps_is_consumer_enabled('thread_instrumentation');
+-----------------------------------------------------+
| sys.ps_is_consumer_enabled('thread_instrumentation') |
+-----------------------------------------------------+
| YES                                                 |
+-----------------------------------------------------+
```

27.4.5.11 The ps_is_instrument_default_enabled() Function

Returns `YES` or `NO` to indicate whether a given Performance Schema instrument is enabled by default.

Parameters

- `in_instrument VARCHAR(128)`: The name of the instrument to check.

Return Value

An `ENUM('YES','NO')` value.

Example

```
mysql> SELECT sys.ps_is_instrument_default_enabled('memory/innodb/row_log_buf');
+-------------------------------------------------------------------+
| sys.ps_is_instrument_default_enabled('memory/innodb/row_log_buf') |
+-------------------------------------------------------------------+
| NO                                                                |
+-------------------------------------------------------------------+
mysql> SELECT sys.ps_is_instrument_default_enabled('statement/sql/alter_user');
+-------------------------------------------------------------------+
| sys.ps_is_instrument_default_enabled('statement/sql/alter_user') |
+-------------------------------------------------------------------+
| YES                                                               |
+-------------------------------------------------------------------+
```

27.4.5.12 The ps_is_instrument_default_timed() Function

Returns `YES` or `NO` to indicate whether a given Performance Schema instrument is timed by default.

Parameters

- `in_instrument VARCHAR(128)`: The name of the instrument to check.

Return Value

An `ENUM('YES','NO')` value.

Example

```
mysql> SELECT sys.ps_is_instrument_default_timed('memory/innodb/row_log_buf');
+-------------------------------------------------------------+
| sys.ps_is_instrument_default_timed('memory/innodb/row_log_buf') |
+-------------------------------------------------------------+
| NO                                                          |
+-------------------------------------------------------------+
mysql> SELECT sys.ps_is_instrument_default_timed('statement/sql/alter_user');
+-------------------------------------------------------------+
| sys.ps_is_instrument_default_timed('statement/sql/alter_user') |
+-------------------------------------------------------------+
| YES                                                         |
+-------------------------------------------------------------+
```

27.4.5.13 The ps_is_thread_instrumented() Function

Returns `YES` or `NO` to indicate whether Performance Schema instrumentation for a given connection ID is enabled, `UNKNOWN` if the ID is unknown, or `NULL` if the ID is `NULL`.

Parameters

- `in_connection_id BIGINT UNSIGNED`: The connection ID. This is a value of the type given in the `PROCESSLIST_ID` column of the Performance Schema `threads` table or the `Id` column of `SHOW PROCESSLIST` output.

Return Value

An `ENUM('YES','NO','UNKNOWN')` value.

Example

```
mysql> SELECT sys.ps_is_thread_instrumented(43);
+-----------------------------------+
| sys.ps_is_thread_instrumented(43) |
+-----------------------------------+
| UNKNOWN                           |
+-----------------------------------+
mysql> SELECT sys.ps_is_thread_instrumented(CONNECTION_ID());
+-----------------------------------------------+
| sys.ps_is_thread_instrumented(CONNECTION_ID()) |
+-----------------------------------------------+
| YES                                           |
+-----------------------------------------------+
```

27.4.5.14 The ps_thread_account() Function

Given a Performance Schema thread ID, returns the *`user_name@host_name`* account associated with the thread.

Parameters

- `in_thread_id BIGINT UNSIGNED`: The thread ID for which to return the account. The value should match the `THREAD_ID` column from some Performance Schema `threads` table row.

Return Value

A `TEXT` value.

Example

```
mysql> SELECT sys.ps_thread_account(sys.ps_thread_id(CONNECTION_ID()));
+-----------------------------------------------------------+
| sys.ps_thread_account(sys.ps_thread_id(CONNECTION_ID())) |
+-----------------------------------------------------------+
| root@localhost                                            |
+-----------------------------------------------------------+
```

27.4.5.15 The ps_thread_id() Function

Note

As of MySQL 8.0.16, `ps_thread_id()` is deprecated and will be removed in a future MySQL version. Applications that use it should be migrated to use the built-in `PS_THREAD_ID()` and `PS_CURRENT_THREAD_ID()` functions instead. See Section 12.22, “Performance Schema Functions”

Returns the Performance Schema thread ID assigned to a given connection ID, or the thread ID for the current connection if the connection ID is `NULL`.

Parameters

- `in_connection_id BIGINT UNSIGNED`: The ID of the connection for which to return the thread ID. This is a value of the type given in the `PROCESSLIST_ID` column of the Performance Schema `threads` table or the `Id` column of `SHOW PROCESSLIST` output.

Return Value

A `BIGINT UNSIGNED` value.

Example

```
mysql> SELECT sys.ps_thread_id(260);
+----------------------+
| sys.ps_thread_id(260) |
+----------------------+
|                  285 |
+----------------------+
```

27.4.5.16 The ps_thread_stack() Function

Returns a JSON formatted stack of all statements, stages, and events within the Performance Schema for a given thread ID.

Parameters

- `in_thread_id BIGINT`: The ID of the thread to trace. The value should match the `THREAD_ID` column from some Performance Schema `threads` table row.

- `in_verbose BOOLEAN`: Whether to include `file:lineno` information in the events.

Return Value

A `LONGTEXT CHARACTER SET latin1` value.

Example

```
mysql> SELECT sys.ps_thread_stack(37, FALSE) AS thread_stack\G
*************************** 1. row ***************************
thread_stack: {"rankdir": "LR","nodesep": "0.10",
"stack_created": "2014-02-19 13:39:03", "mysql_version": "8.0.2-dmr-debug-log",
"mysql_user": "root@localhost","events": [{"nesting_event_id": "0",
"event_id": "10", "timer_wait": 256.35, "event_info": "sql/select",
"wait_info": "select @@version_comment limit 1\nerrors: 0\nwarnings: 0\nlock time:
...
```

27.4.5.17 The ps_thread_trx_info() Function

Returns a JSON object containing information about a given thread. The information includes the current transaction, and the statements it has already executed, derived from the Performance Schema `events_transactions_current` and `events_statements_history` tables. (The consumers for those tables must be enabled to obtain full data in the JSON object.)

If the output exceeds the truncation length (65535 by default), a JSON error object is returned, such as:

```
{ "error": "Trx info truncated: Row 6 was cut by GROUP_CONCAT()" }
```

Similar error objects are returned for other warnings and exceptions raised during function execution.

Parameters

- `in_thread_id BIGINT UNSIGNED`: The thread ID for which to return transaction information. The value should match the `THREAD_ID` column from some Performance Schema `threads` table row.

Configuration Options

`ps_thread_trx_info()` operation can be modified using the following configuration options or their corresponding user-defined variables (see Section 27.4.2.1, “The sys_config Table”):

- `ps_thread_trx_info.max_length`, `@sys.ps_thread_trx_info.max_length`

  The maximum length of the output. The default is 65535.

Return Value

A `LONGTEXT` value.

Example

```
mysql> SELECT sys.ps_thread_trx_info(48)\G
*************************** 1. row ***************************
sys.ps_thread_trx_info(48): [
  {
    "time": "790.70 us",
    "state": "COMMITTED",
    "mode": "READ WRITE",
```

```
    "autocommitted": "NO",
    "gtid": "AUTOMATIC",
    "isolation": "REPEATABLE READ",
    "statements_executed": [
      {
        "sql_text": "INSERT INTO info VALUES (1, \'foo\')",
        "time": "471.02 us",
        "schema": "trx",
        "rows_examined": 0,
        "rows_affected": 1,
        "rows_sent": 0,
        "tmp_tables": 0,
        "tmp_disk_tables": 0,
        "sort_rows": 0,
        "sort_merge_passes": 0
      },
      {
        "sql_text": "COMMIT",
        "time": "254.42 us",
        "schema": "trx",
        "rows_examined": 0,
        "rows_affected": 0,
        "rows_sent": 0,
        "tmp_tables": 0,
        "tmp_disk_tables": 0,
        "sort_rows": 0,
        "sort_merge_passes": 0
      }
    ]
  },
  {
    "time": "426.20 us",
    "state": "COMMITTED",
    "mode": "READ WRITE",
    "autocommitted": "NO",
    "gtid": "AUTOMATIC",
    "isolation": "REPEATABLE READ",
    "statements_executed": [
      {
        "sql_text": "INSERT INTO info VALUES (2, \'bar\')",
        "time": "107.33 us",
        "schema": "trx",
        "rows_examined": 0,
        "rows_affected": 1,
        "rows_sent": 0,
        "tmp_tables": 0,
        "tmp_disk_tables": 0,
        "sort_rows": 0,
        "sort_merge_passes": 0
      },
      {
        "sql_text": "COMMIT",
        "time": "213.23 us",
        "schema": "trx",
        "rows_examined": 0,
        "rows_affected": 0,
        "rows_sent": 0,
        "tmp_tables": 0,
        "tmp_disk_tables": 0,
        "sort_rows": 0,
        "sort_merge_passes": 0
      }
    ]
  }
]
```


27.4.5.18 The quote_identifier() Function

Given a string argument, this function produces a quoted identifier suitable for inclusion in SQL statements. This is useful when a value to be used as an identifier is a reserved word or contains backtick (`) characters.

Parameters

`in_identifier TEXT`: The identifier to quote.

Return Value

A `TEXT` value.

Example

```
mysql> SELECT sys.quote_identifier('plain');
+------------------------------+
| sys.quote_identifier('plain') |
+------------------------------+
| `plain`                      |
+------------------------------+
mysql> SELECT sys.quote_identifier('trick`ier');
+---------------------------------+
| sys.quote_identifier('trick`ier') |
+---------------------------------+
| `trick``ier`                     |
+---------------------------------+
mysql> SELECT sys.quote_identifier('integer');
+--------------------------------+
| sys.quote_identifier('integer') |
+--------------------------------+
| `integer`                      |
+--------------------------------+
```

27.4.5.19 The sys_get_config() Function

Given a configuration option name, returns the option value from the `sys_config` table, or the provided default value (which may be `NULL`) if the option does not exist in the table.

If `sys_get_config()` returns the default value and that value is `NULL`, it is expected that the caller is able to handle `NULL` for the given configuration option.

By convention, routines that call `sys_get_config()` first check whether the corresponding user-defined variable exists and is non-`NULL`. If so, the routine uses the variable value without reading the `sys_config` table. If the variable does not exist or is `NULL`, the routine reads the option value from the table and sets the user-defined variable to that value. For more information about the relationship between configuration options and their corresponding user-defined variables, see Section 27.4.2.1, “The sys_config Table”.

If you want to check whether the configuration option has already been set and, if not, use the return value of `sys_get_config()`, you can use `IFNULL(...)` (see example later). However, this should not be done inside a loop (for example, for each row in a result set) because for repeated calls where the assignment is needed only in the first iteration, using `IFNULL(...)` is expected to be significantly slower than using an `IF (...) THEN ... END IF;` block (see example later).

Parameters

- `in_variable_name VARCHAR(128)`: The name of the configuration option for which to return the value.

- `in_default_value VARCHAR(128)`: The default value to return if the configuration option is not found in the `sys_config` table.

Return Value

A `VARCHAR(128)` value.

Example

Get a configuration value from the `sys_config` table, falling back to 128 as the default if the option is not present in the table:

```
mysql> SELECT sys.sys_get_config('statement_truncate_len', 128) AS Value;
+-------+
| Value |
+-------+
| 64    |
+-------+
```

One-liner example: Check whether the option is already set; if not, assign the `IFNULL(...)` result (using the value from the `sys_config` table):

```
mysql> SET @sys.statement_truncate_len =
       IFNULL(@sys.statement_truncate_len,
              sys.sys_get_config('statement_truncate_len', 64));
```

`IF (...) THEN ... END IF;` block example: Check whether the option is already set; if not, assign the value from the `sys_config` table:

```
IF (@sys.statement_truncate_len IS NULL) THEN
  SET @sys.statement_truncate_len = sys.sys_get_config('statement_truncate_len', 64);
END IF;
```

27.4.5.20 The version_major() Function

This function returns the major version of the MySQL server.

Parameters

None.

Return Value

A `TINYINT UNSIGNED` value.

Example

```
mysql> SELECT VERSION(), sys.version_major();
+--------------+---------------------+
| VERSION()    | sys.version_major() |
+--------------+---------------------+
| 8.0.13-debug |                   8 |
+--------------+---------------------+
```

27.4.5.21 The version_minor() Function

This function returns the minor version of the MySQL server.

Parameters

None.

Return Value

A `TINYINT UNSIGNED` value.

Example

```
mysql> SELECT VERSION(), sys.version_minor();
+--------------+---------------------+
| VERSION()    | sys.version_minor() |
+--------------+---------------------+
| 8.0.13-debug |                   0 |
+--------------+---------------------+
```

27.4.5.22 The version_patch() Function

This function returns the patch release version of the MySQL server.

Parameters

None.

Return Value

A `TINYINT UNSIGNED` value.

Example

```
mysql> SELECT VERSION(), sys.version_patch();
+--------------+---------------------+
| VERSION()    | sys.version_patch() |
+--------------+---------------------+
| 8.0.13-debug |                  13 |
+--------------+---------------------+
```

Chapter 28 Connectors and APIs

Table of Contents

28.1 MySQL Connector/C 4784
28.2 MySQL Connector/C++ 4784
28.3 MySQL Connector/J 4784
28.4 MySQL Connector/NET 4784
28.5 MySQL Connector/ODBC 4785
28.6 MySQL Connector/Python 4785
28.7 MySQL C API 4785
28.7.1 MySQL C API Implementations 4786
28.7.2 Simultaneous MySQL Server and Connector/C Installations 4786
28.7.3 Example C API Client Programs 4787
28.7.4 Building and Running C API Client Programs 4788
28.7.5 C API Data Structures 4793
28.7.6 C API Function Overview 4799
28.7.7 C API Function Descriptions 4804
28.7.8 C API Prepared Statements 4874
28.7.9 C API Prepared Statement Data Structures 4874
28.7.10 C API Prepared Statement Function Overview 4881
28.7.11 C API Prepared Statement Function Descriptions 4883
28.7.12 C API Asynchronous Interface 4908
28.7.13 C API Asynchronous Interface Data Structures 4913
28.7.14 C API Asynchronous Function Overview 4913
28.7.15 C API Asynchronous Function Descriptions 4914
28.7.16 C API Threaded Function Descriptions 4918
28.7.17 C API Client Plugin Functions 4919
28.7.18 C API Binary Log Interface 4922
28.7.19 C API Binary Log Data Structures 4922
28.7.20 C API Binary Log Function Overview 4924
28.7.21 C API Binary Log Function Descriptions 4924
28.7.22 C API Encrypted Connection Support 4927
28.7.23 C API Multiple Statement Execution Support 4928
28.7.24 C API Prepared Statement Handling of Date and Time Values 4931
28.7.25 C API Prepared CALL Statement Support 4932
28.7.26 C API Prepared Statement Problems 4936
28.7.27 C API Optional Result Set Metadata 4936
28.7.28 C API Automatic Reconnection Control 4937
28.7.29 C API Common Issues 4938
28.8 MySQL PHP API 4940
28.9 MySQL Perl API 4940
28.10 MySQL Python API 4940
28.11 MySQL Ruby APIs 4941
28.11.1 The MySQL/Ruby API 4941
28.11.2 The Ruby/MySQL API 4941
28.12 MySQL Tcl API 4941
28.13 MySQL Eiffel Wrapper 4941

MySQL Connectors provide connectivity to the MySQL server for client programs. APIs provide low-level access to the MySQL protocol and MySQL resources. Both Connectors and the APIs enable you to

connect and execute MySQL statements from another language or environment, including ODBC, Java (JDBC), Perl, Python, PHP, Ruby, and native C MySQL instances.

MySQL Connectors

Oracle develops a number of connectors:

- Connector/C is a standalone replacement for the MySQL Client Library (`libmysqlclient`), to be used for C applications.
- Connector/C++ enables C++ applications to connect to MySQL.
- Connector/J provides driver support for connecting to MySQL from Java applications using the standard Java Database Connectivity (JDBC) API.
- Connector/NET enables developers to create .NET applications that connect to MySQL. Connector/NET implements a fully functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that use Connector/NET can be written in any supported .NET language.

  MySQL for Visual Studio works with Connector/NET and Microsoft Visual Studio 2012, 2013, 2015, and 2017. MySQL for Visual Studio provides access to MySQL objects and data from Visual Studio. As a Visual Studio package, it integrates directly into Server Explorer providing the ability to create new connections and work with MySQL database objects.
- Connector/ODBC provides driver support for connecting to MySQL using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix, and OS X platforms.
- Connector/Python provides driver support for connecting to MySQL from Python applications using an API that is compliant with the Python DB API version 2.0. No additional Python modules or MySQL client libraries are required.

The MySQL C API

For direct access to using MySQL natively within a C application, the C API provides low-level access to the MySQL client/server protocol through the `libmysqlclient` client library. This is the primary method used to connect to an instance of the MySQL server, and is used both by MySQL command-line clients and many of the MySQL Connectors and third-party APIs detailed here.

`libmysqlclient` is included in MySQL distributions and in Connector/C distributions.

See also Section 28.7.1, “MySQL C API Implementations”.

To access MySQL from a C application, or to build an interface to MySQL for a language not supported by the Connectors or APIs in this chapter, the C API is where to start. A number of programmer's utilities are available to help with the process; see Section 4.7, “MySQL Program Development Utilities”.

Third-Party MySQL APIs

The remaining APIs described in this chapter provide an interface to MySQL from specific application languages. These third-party solutions are not developed or supported by Oracle. Basic information on their usage and abilities is provided here for reference purposes only.

All the third-party language APIs are developed using one of two methods, using `libmysqlclient` or by implementing a *native driver*. The two solutions offer different benefits:

- Using *`libmysqlclient`* offers complete compatibility with MySQL because it uses the same libraries as the MySQL client applications. However, the feature set is limited to the implementation and interfaces exposed through `libmysqlclient` and the performance may be lower as data is copied between the native language, and the MySQL API components.
- *Native drivers* are an implementation of the MySQL network protocol entirely within the host language or environment. Native drivers are fast, as there is less copying of data between components, and they can offer advanced functionality not available through the standard MySQL API. Native drivers are also easier for end users to build and deploy because no copy of the MySQL client libraries is needed to build the native driver components.

Table 28.1, “MySQL APIs and Interfaces” lists many of the libraries and interfaces available for MySQL.

Table 28.1 MySQL APIs and Interfaces

Environment	API	Type	Notes
Ada	GNU Ada MySQL Bindings	`libmysqlclient`	See MySQL Bindings for GNU Ada
C	C API	`libmysqlclient`	See Section 28.7, “MySQL C API”.
C	Connector/C	Replacement for `libmysqlclient`	See MySQL Connector/C Developer Guide.
C++	Connector/C++	`libmysqlclient`	See MySQL Connector/C++ 8.0 Developer Guide.
	MySQL++	`libmysqlclient`	See MySQL++ website.
	MySQL wrapped	`libmysqlclient`	See MySQL wrapped.
Cocoa	MySQL-Cocoa	`libmysqlclient`	Compatible with the Objective-C Cocoa environment. See http://mysql-cocoa.sourceforge.net/
D	MySQL for D	`libmysqlclient`	See MySQL for D.
Eiffel	Eiffel MySQL	`libmysqlclient`	See Section 28.13, “MySQL Eiffel Wrapper”.
Erlang	`erlang-mysql-driver`	`libmysqlclient`	See `erlang-mysql-driver`.
Haskell	Haskell MySQL Bindings	Native Driver	See Brian O'Sullivan's pure Haskell MySQL bindings.
	`hsql-mysql`	`libmysqlclient`	See MySQL driver for Haskell.
Java/ JDBC	Connector/J	Native Driver	See MySQL Connector/J 5.1 Developer Guide.
Kaya	MyDB	`libmysqlclient`	See MyDB.
Lua	LuaSQL	`libmysqlclient`	See LuaSQL.
.NET/ Mono	Connector/NET	Native Driver	See MySQL Connector/NET Developer Guide.
Objective Caml	OBjective Caml MySQL Bindings	`libmysqlclient`	See MySQL Bindings for Objective Caml.
Octave	Database bindings for GNU Octave	`libmysqlclient`	See Database bindings for GNU Octave.
ODBC	Connector/ODBC	`libmysqlclient`	See MySQL Connector/ODBC Developer Guide.
Perl	`DBI`/`DBD::mysql`	`libmysqlclient`	See Section 28.9, “MySQL Perl API”.

Environment	API	Type	Notes
	`Net::MySQL`	Native Driver	See `Net::MySQL` at CPAN
PHP	`mysql`, `ext/mysql` interface (deprecated)	`libmysqlclient`	See Original MySQL API.
	`mysqli`, `ext/mysqli` interface	`libmysqlclient`	See MySQL Improved Extension.
	`PDO_MYSQL`	`libmysqlclient`	See MySQL Functions (PDO_MYSQL).
	PDO mysqlnd	Native Driver	
Python	Connector/Python	Native Driver	See MySQL Connector/Python Developer Guide.
Python	Connector/Python C Extension	`libmysqlclient`	See MySQL Connector/Python Developer Guide.
	MySQLdb	`libmysqlclient`	See Section 28.10, “MySQL Python API”.
Ruby	MySQL/Ruby	`libmysqlclient`	Uses `libmysqlclient`. See Section 28.11.1, “The MySQL/Ruby API”.
	Ruby/MySQL	Native Driver	See Section 28.11.2, “The Ruby/MySQL API”.
Scheme	`Myscsh`	`libmysqlclient`	See `Myscsh`.
SPL	`sql_mysql`	`libmysqlclient`	See `sql_mysql` for SPL.
Tcl	MySQLtcl	`libmysqlclient`	See Section 28.12, “MySQL Tcl API”.

28.1 MySQL Connector/C

The MySQL Connector/C manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/C Developer Guide
- Release notes: MySQL Connector/C Release Notes

28.2 MySQL Connector/C++

The MySQL Connector/C++ manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/C++ 8.0 Developer Guide
- Release notes: MySQL Connector/C++ Release Notes

28.3 MySQL Connector/J

The MySQL Connector/J manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/J Developer Guide
- Release notes: MySQL Connector/J Release Notes

28.4 MySQL Connector/NET

The MySQL Connector/NET manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/NET Developer Guide
- Release notes: MySQL Connector/NET Release Notes

28.5 MySQL Connector/ODBC

The MySQL Connector/ODBC manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/ODBC Developer Guide
- Release notes: MySQL Connector/ODBC Release Notes

28.6 MySQL Connector/Python

The MySQL Connector/Python manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/Python Developer Guide
- Release notes: MySQL Connector/Python Release Notes

28.7 MySQL C API

The C API provides low-level access to the MySQL client/server protocol and enables C programs to access database contents. The C API code is distributed with MySQL and implemented in the `libmysqlclient` library. See Section 28.7.1, “MySQL C API Implementations”.

Most other client APIs use the `libmysqlclient` library to communicate with the MySQL server. (Exceptions are Connector/J and Connector/NET.) This means that, for example, you can take advantage of many of the same environment variables that are used by other client programs because they are referenced from the library. For a list of these variables, see Section 4.1, “Overview of MySQL Programs”.

For instructions on building client programs using the C API, see Section 28.7.4.1, “Building C API Client Programs”. For programming with threads, see Section 28.7.4.3, “Writing C API Threaded Client Programs”.

Note

If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, the programs were probably compiled using old header or library files. In this case, check the date of the `mysql.h` file and `libmysqlclient.a` library used for compilation to verify that they are from the new MySQL distribution. If not, recompile the programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example, from `libmysqlclient.so.17` to `libmysqlclient.so.18`). For additional compatibility information, see Section 28.7.4.4, “Running C API Client Programs”.

Clients have a maximum communication buffer size. The size of the buffer that is allocated initially (16KB) is automatically increased up to the maximum size (16MB by default). Because buffer sizes are increased only as demand warrants, simply increasing the maximum limit does not in itself cause more resources to be used. This size check is mostly a precaution against erroneous statements and communication packets.

The communication buffer must be large enough to contain a single SQL statement (for client-to-server traffic) and one row of returned data (for server-to-client traffic). Each session's communication buffer is dynamically enlarged to handle any query or row up to the maximum limit. For example, if you have `BLOB` values that contain up to 16MB of data, you must have a communication buffer limit of at least 16MB (in both server and client). The default maximum built into the client library is 1GB, but the default maximum in the server is 1MB. You can increase this by changing the value of the `max_allowed_packet` parameter at server startup. See Section 5.1.1, “Configuring the Server”.

The MySQL server shrinks each communication buffer to `net_buffer_length` bytes after each query. For clients, the size of the buffer associated with a connection is not decreased until the connection is closed, at which time client memory is reclaimed.

28.7.1 MySQL C API Implementations

The MySQL C API is a C-based API that client applications written in C can use to communicate with MySQL Server. Client programs refer to C API header files at compile time and link to a C API library file, `libmysqlclient`, at link time.

There are two ways to obtain the C API header and library files required to build C API client programs:

- Install a MySQL Server distribution.
- Install a Connector/C distribution.

For both MySQL Server and Connector/C, you can install a binary distribution that contains the C API files pre-built, or you can use a source distribution and build the C API files yourself.

Normally, you install either a MySQL Server distribution or a Connector/C distribution, but not both. For information about issues involved with simultaneous MySQL Server and Connector/C installations, see Section 28.7.2, “Simultaneous MySQL Server and Connector/C Installations”.

The names of the library files to use when linking C API client applications depend on the library type and platform for which a distribution is built:

- On Unix (and Unix-like) systems, the static library is `libmysqlclient.a`. The dynamic library is `libmysqlclient.so` on most Unix systems and `libmysqlclient.dylib` on OS X.
- On Windows, the static library is `mysqlclient.lib` and the dynamic library is `libmysql.dll`. Windows distributions also include `libmysql.lib`, a static import library needed for using the dynamic library.

  Windows distributions also include a set of debug libraries. These have the same names as the nondebug libraries, but are located in the `lib/debug` library. You must use the debug libraries when compiling clients built using the debug C runtime.

On Unix, you may also see libraries that include `_r` in the names. Before MySQL 5.5, these were built as thread-safe (re-entrant) libraries separately from the non-`_r` libraries. As of 5.5, both libraries are the same and the `_r` names are symbolic links to the corresponding non-`_r` names. There is no need to use the `_r` libraries. For example, if you use `mysql_config` to obtain linker flags, you can use `mysql_config --libs` in all cases, even for threaded clients. There is no need to use `mysql_config --libs_r`.

28.7.2 Simultaneous MySQL Server and Connector/C Installations

MySQL Server and Connector/C installation packages both provide the files needed to build and run MySQL C API client programs. This section discusses when it is possible to install both products on the

same system. For some packaging formats, this is possible without conflict. For others, both products cannot be installed at the same time.

This discussion assumes the use of similar package types for both products (for example, RPM packages for both products). It does not try to describe coexistence between packaging types (for example, use of RPM packages for one product and a `tar` file package for the other). Nor does it describe coexistence of packages provided by Oracle and those provided by third-party vendors.

If you install both products, it may be necessary to adjust your development tools or runtime environment to choose one set of header files and libraries over the other. See Section 28.7.4.1, “Building C API Client Programs”, and Section 28.7.4.4, “Running C API Client Programs”.

`tar` and Zip file packages install under the directory into which you unpack them. For example, you can unpack MySQL Server and Connector/C `tar` packages under `/usr/local` and they will unpack into distinct directory names without conflict.

Windows MSI installers use their own installation directory, so MySQL Server and Connector/C installers do not conflict.

OS X DMG packages install under the same parent directory but in a different subdirectory, so there is no conflict. For example:

```
/usr/local/mysql-5.6.11-osx10.7-x86_64/
/usr/local/mysql-connector-c-6.1.0-osx10.7-x86/
```

Solaris PKG packages install under the same parent directory but in a different subdirectory, so there is no conflict. For example:

```
/opt/mysql/mysql
/opt/mysql/connector-c
```

The Solaris Connector/C installer does not create any symlinks from system directories such as `/usr/bin` or `/usr/lib` into the installation directory. That must be done manually if desired after installation.

For RPM installations, there are several types of RPM packages. MySQL Server `shared` and `devel` RPM packages are similar to the corresponding Connector/C RPM packages. These RPM package types cannot coexist because the MySQL Server and Connector/C RPM packages use the same installation locations for the client library-related files. This means the following conditions hold:

- If MySQL Server `shared` and `devel` RPM packages are installed, they provide the C API headers and libraries, and there is no need to install the Connector/C RPM packages. To install the Connector/C packages anyway, you must first remove the corresponding MySQL Server packages.
- To install MySQL Server RPM packages if you already have Connector/C RPM packages installed, you must first remove the Connector/C RPM packages.

MySQL Server RPM packages other than `shared` and `devel` do not conflict with Connector/C packages and can be installed if Connector/C is installed. This includes the main server RPM that includes the `mysqld` server itself.

28.7.3 Example C API Client Programs

Many of the clients in MySQL source distributions are written in C, such as `mysql`, `mysqladmin`, and `mysqlshow`. If you are looking for examples that demonstrate how to use the C API, take a look at these clients: Obtain a source distribution and look in its `client` directory. See Section 2.1.2, “How to Get MySQL”.

28.7.4 Building and Running C API Client Programs

The following sections provide information on building client programs that use the C API. Topics include compiling and linking clients, writing threaded clients, and troubleshooting runtime problems.

28.7.4.1 Building C API Client Programs

This section provides guidelines for compiling C programs that use the MySQL C API.

Compiling MySQL Clients on Unix

The examples here use `gcc` as the compiler. A different compiler might be appropriate on some systems (for example, `clang` on OS X or FreeBSD, or Sun Studio on Solaris). Adjust the examples as necessary.

You may need to specify an `-I` option when you compile client programs that use MySQL header files, so that the compiler can find them. For example, if the header files are installed in `/usr/local/mysql/include`, use this option in the compile command:

```
-I/usr/local/mysql/include
```

MySQL clients must be linked using the `-lmysqlclient` option in the link command. You may also need to specify a `-L` option to tell the linker where to find the library. For example, if the library is installed in `/usr/local/mysql/lib`, use these options in the link command:

```
-L/usr/local/mysql/lib -lmysqlclient
```

The path names may differ on your system. Adjust the `-I` and `-L` options as necessary.

To make it simpler to compile MySQL programs on Unix, use the `mysql_config` script. See Section 4.7.1, “`mysql_config` — Display Options for Compiling Clients”.

`mysql_config` displays the options needed for compiling or linking:

```
shell> mysql_config --cflags
shell> mysql_config --libs
```

You can run those commands to get the proper options and add them manually to compilation or link commands. Alternatively, include the output from `mysql_config` directly within command lines using backticks:

```
shell> gcc -c `mysql_config --cflags` progname.c
shell> gcc -o progname progname.o `mysql_config --libs`
```

On Unix, linking uses dynamic libraries by default. To link to the static client library instead, add its path name to the link command. For example, if the library is located in `/usr/local/mysql/lib`, link like this:

```
shell> gcc -o progname progname.o /usr/local/mysql/lib/libmysqlclient.a
```

Or use `mysql_config` to provide the library name:

```
shell> gcc -o progname progname.o `mysql_config --variable=pkglibdir`/libmysqlclient.a
```

`mysql_config` does not currently provide a way to list all libraries needed for static linking, so it might be necessary to name additional libraries on the link command (for example, `-lnsl -lsocket` on Solaris). To get an idea which libraries to add, use `mysql_config --libs` and `ldd libmysqlclient.so` (or `otool -L libmysqlclient.dylib` on OS X).

`pkg-config` can be used as an alternative to `mysql_config` for obtaining information such as compiler flags or link libraries required to compile MySQL applications. For example, the following pairs of commands are equivalent:

```
mysql_config --cflags
pkg-config --cflags mysqlclient

mysql_config --libs
pkg-config --libs mysqlclient
```

To produce flags for static linking, use this command:

```
pkg-config --static --libs mysqlclient
```

For more information, see Section 28.7.4.2, “Building C API Client Programs Using pkg-config”.

Compiling MySQL Clients on Microsoft Windows

To specify header and library file locations, use the facilities provided by your development environment.

To build C API clients on Windows, you must link in the C client library, as well as the Windows ws2_32 sockets library and Secur32 security library.

You link your code with either the dynamic or static C client library. On Windows, the static library is named `mysqlclient.lib` and the dynamic library is named `libmysql.dll`. In addition, the `libmysql.lib` static import library is needed for using the dynamic library. If the static C client library is used, the client application must be compiled with the same version of Visual Studio used to compile the C client library (which is Visual Studio 2015 for the static C client library built by Oracle).

Note

The MySQL Connector/C is a standalone, drop-in replacement of the MySQL C client libraries that come with the MySQL server distribution. The Oracle-built MySQL Connector/C contains currently two versions of the static client library, one built with Visual Studio 2013 and the other one with Visual Studio 2015; use the one that matches the Visual Studio version you use to compile your application.

When using the Oracle-built MySQL C client library (or MySQL Connector/C), following these rules when it comes to linking the C runtime for your client application:

- For the Community version of the MySQL C client library (or the Community version of MySQL Connector/C):
  - *For version 8.0.0* (or MySQL Connector/C Community 6.1.9 and before):
    - If linking to the static C client library, link statically to the C runtime (use the `/MT` compiler option).
    - If linking to the dynamic C client library, link either statically or dynamically to the C runtime (use either `/MT` or `/MD` compiler option).
  - *For version 8.0.1 and later* (or MySQL Connector/C Community 6.1.10 and later): Always link dynamically to the C runtime (use the `/MD` compiler option), whether you are linking to the static or dynamic C client library. Also, target hosts running the client application need to have the Visual C++ Redistributable for Visual Studio 2015 installed.
- For the Commercial version of the MySQL C client library (or the Commercial version of MySQL Connector/C):

- If linking to the static C client library, link statically to the C runtime (use the `/MT` compiler option).
- If linking to the dynamic C client library, link either statically or dynamically to the C runtime (use either `/MT` or `/MD` compiler option).

In general, when linking to a static MySQL C client library, the client library and the client application must use the same compiler option when it comes to linking the C runtime—that is, if your C client library is compiled with the `/MT` option, your client application should also be compiled with the `/MT` option, and so on (see the MSDN page describing the C library linking options for more details). Follow this rule when you are building your own static MySQL C client library (or MySQL Connector/C) from source and linking you client application to it.

Note

Debug Mode: Because of the above-mentioned rule, you cannot build your application in debug mode (with the `/MTd` or `/MDd` compiler option) and link it to the static C client library built by Oracle, which is *not* built with the debug options; instead, you will have to build the static client library from source with the debug options.

Troubleshooting Problems Linking to the MySQL Client Library

The MySQL client library includes SSL support built in. It is unnecessary to specify either `-lssl` or `-lcrypto` at link time. Doing so may in fact result in problems at runtime.

If the linker cannot find the MySQL client library, you might get undefined-reference errors for symbols that start with `mysql_`, such as those shown here:

```
/tmp/ccFKsdPa.o: In function `main':
/tmp/ccFKsdPa.o(.text+0xb): undefined reference to `mysql_init'
/tmp/ccFKsdPa.o(.text+0x31): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x69): undefined reference to `mysql_error'
/tmp/ccFKsdPa.o(.text+0x9a): undefined reference to `mysql_close'
```

You should be able to solve this problem by adding `-L`*`dir_path`* `-lmysqlclient` at the end of your link command, where *`dir_path`* represents the path name of the directory where the client library is located. To determine the correct directory, try this command:

```
shell> mysql_config --libs
```

The output from `mysql_config` might indicate other libraries that should be specified on the link command as well. You can include `mysql_config` output directly in your compile or link command using backticks. For example:

```
shell> gcc -o progname progname.o `mysql_config --libs`
```

If an error occurs at link time that the `floor` symbol is undefined, link to the math library by adding `-lm` to the end of the compile/link line. Similarly, if you get undefined-reference errors for other functions that should exist on your system, such as `connect()`, check the manual page for the function in question to determine which libraries you should add to the link command.

If you get undefined-reference errors such as the following for functions that do not exist on your system, it usually means that your MySQL client library was compiled on a system that is not 100% compatible with yours:

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

In this case, you should download the latest MySQL or Connector/C source distribution and compile the MySQL client library yourself. See Section 2.9, “Installing MySQL from Source”, and MySQL Connector/C Developer Guide.

28.7.4.2 Building C API Client Programs Using pkg-config

MySQL distributions contain a `mysqlclient.pc` file that provides information about MySQL configuration for use by the `pkg-config` command. This enables `pkg-config` to be used as an alternative to `mysql_config` for obtaining information such as compiler flags or link libraries required to compile MySQL applications. For example, the following pairs of commands are equivalent:

```
mysql_config --cflags
pkg-config --cflags mysqlclient

mysql_config --libs
pkg-config --libs mysqlclient
```

The last `pkg-config` command produces flags for dynamic linking. To produce flags for static linking, use this command:

```
pkg-config --static --libs mysqlclient
```

On some platforms, the output with and without `--static` might be the same.

Note

If `pkg-config` does not find MySQL information, it might be necessary to set the `PKG_CONFIG_PATH` environment variable to the directory in which the `mysqlclient.pc` file is located, which by default is usually the `pkgconfig` directory under the MySQL library directory. For example (adjust the location appropriately):

```
export PKG_CONFIG_PATH=/usr/local/mysql/lib/pkgconfig # sh, bash, ...
setenv PKG_CONFIG_PATH /usr/local/mysql/lib/pkgconfig # csh, tcsh, ...
```

The `mysqlconfig.pc` installation location can be controlled using the `INSTALL_PKGCONFIGDIR` `CMake` option. See Section 2.9.7, “MySQL Source-Configuration Options”.

The `--variable` option takes a configuration variable name and displays the variable value:

```
pkg-config --variable=prefix mysqlclient     # installation prefix directory
pkg-config --variable=includedir mysqlclient # header file directory
pkg-config --variable=libdir mysqlclient     # library directory
```

To see which variable values `pkg-config` can display using the `--variable` option, use this command:

```
pkg-config --print-variables mysqlclient
```

You can use `pkg-config` within a command line using backticks to include the output that it produces for particular options. For example, to compile and link a MySQL client program, use `pkg-config` as follows:

```
gcc -c `pkg-config --cflags mysqlclient` progname.c
gcc -o progname progname.o `pkg-config --libs mysqlclient`
```

28.7.4.3 Writing C API Threaded Client Programs

The client library is almost thread-safe. The biggest problem is that the subroutines in `sql/net_serv.cc` that read from sockets are not interrupt-safe. This was done with the thought that you might want to have your own alarm that can break a long read to a server. If you install interrupt handlers for the `SIGPIPE` interrupt, socket handling should be thread-safe.

To avoid aborting the program when a connection terminates, MySQL blocks `SIGPIPE` on the first call to `mysql_library_init()`, `mysql_init()`, or `mysql_connect()`. To use your own `SIGPIPE` handler, first call `mysql_library_init()`, then install your handler.

If “undefined symbol” errors occur when linking against the `libmysqlclient` client library, in most cases this is because you have not included the thread libraries on the link/compile command.

The client library is thread-safe per connection. You can let two threads share the same connection with the following caveats:

- Multiple threads cannot send a query to the MySQL server at the same time on the same connection. In particular, you must ensure that between calls to `mysql_query()` and `mysql_store_result()` in one thread, no other thread uses the same connection. You must have a mutex lock around your pair of `mysql_query()` and `mysql_store_result()` calls. After `mysql_store_result()` returns, the lock can be released and other threads may query the same connection.

  If you use POSIX threads, you can use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to establish and release a mutex lock.

- Many threads can access different result sets that are retrieved with `mysql_store_result()`.

- To use `mysql_use_result()`, you must ensure that no other thread is using the same connection until the result set is closed. However, it really is best for threaded clients that share the same connection to use `mysql_store_result()`.

You need to know the following if you have a thread that did not create the connection to the MySQL database but is calling MySQL functions:

When you call `mysql_init()`, MySQL creates a thread-specific variable for the thread that is used by the debug library (among other things). If you call a MySQL function before the thread has called `mysql_init()`, the thread does not have the necessary thread-specific variables in place and you are likely to end up with a core dump sooner or later. To avoid problems, you must do the following:

1. Call `mysql_library_init()` before any other MySQL functions. It is not thread-safe, so call it before threads are created, or protect the call with a mutex.

2. Arrange for `mysql_thread_init()` to be called early in the thread handler before calling any MySQL function. If you call `mysql_init()`, it will call `mysql_thread_init()` for you.

3. In the thread, call `mysql_thread_end()` before calling `pthread_exit()`. This frees the memory used by MySQL thread-specific variables.

The preceding notes regarding `mysql_init()` also apply to `mysql_connect()`, which calls `mysql_init()`.

28.7.4.4 Running C API Client Programs

If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, the programs were probably compiled using old header or library files. In this case, check the date of the `mysql.h` file and `libmysqlclient.a` library used for compilation to

verify that they are from the new MySQL distribution. If not, recompile the programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example, from `libmysqlclient.so.17` to `libmysqlclient.so.18`).

The major client library version determines compatibility. (For example, for `libmysqlclient.so.18.1.0`, the major version is 18.) For this reason, the libraries shipped with newer versions of MySQL are drop-in replacements for older versions that have the same major number. As long as the major library version is the same, you can upgrade the library and old applications should continue to work with it.

Undefined-reference errors might occur at runtime when you try to execute a MySQL program. If these errors specify symbols that start with `mysql_` or indicate that the `libmysqlclient` library cannot be found, it means that your system cannot find the shared `libmysqlclient.so` library. The solution to this problem is to tell your system to search for shared libraries in the directory where that library is located. Use whichever of the following methods is appropriate for your system:

- Add the path of the directory where `libmysqlclient.so` is located to the `LD_LIBRARY_PATH` or `LD_LIBRARY` environment variable.
- On OS X, add the path of the directory where `libmysqlclient.dylib` is located to the `DYLD_LIBRARY_PATH` environment variable.
- Copy the shared-library files (such as `libmysqlclient.so`) to some directory that is searched by your system, such as `/lib`, and update the shared library information by executing `ldconfig`. Be sure to copy all related files. A shared library might exist under several names, using symlinks to provide the alternate names.

28.7.4.5 C API Server and Client Library Versions

The string and numeric forms of the MySQL server version are available at compile time as the values of the `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` macros, and at runtime as the values of the `mysql_get_server_info()` and `mysql_get_server_version()` functions.

The MySQL client library version depends on the type of distribution that provides the library:

- For MySQL distributions, the client library version is the MySQL version. The string and numeric forms of this version are available at compile time as the values of the `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` macros, and at runtime as the values of the `mysql_get_client_info()` and `mysql_get_client_version()` functions.

  The `LIBMYSQL_VERSION` and `LIBMYSQL_VERSION_ID` macros have the same values as `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` and the two sets of macros can be used interchangeably.

- For Connector/C distributions, the client library version is the Connector/C version. The string and numeric forms of this version are available at compile time as the values of the `LIBMYSQL_VERSION` and `LIBMYSQL_VERSION_ID` macros, and at runtime as the values of the `mysql_get_client_info()` and `mysql_get_client_version()` functions.

  The `MYSQL_SERVER_VERSION` and `MYSQL_VERSION_ID` macros indicate the string and numeric forms of the MySQL version on which the Connector/C distribution is based.

28.7.5 C API Data Structures

This section describes C API data structures other than those used for prepared statements, the asychronous interface, or the replication stream interface. For information about those, see Section 28.7.9,

"C API Prepared Statement Data Structures", Section 28.7.13, "C API Asynchronous Interface Data Structures", and Section 28.7.19, "C API Binary Log Data Structures".

- `MYSQL`

  This structure represents handler for one database connection. It is used for almost all MySQL functions. Do not try to make a copy of a `MYSQL` structure. There is no guarantee that such a copy will be usable.

- `MYSQL_RES`

  This structure represents the result of a query that returns rows (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). The information returned from a query is called the *result set* in the remainder of this section.

- `MYSQL_ROW`

  This is a type-safe representation of one row of data. It is currently implemented as an array of counted byte strings. (You cannot treat these as null-terminated strings if field values may contain binary data, because such values may contain null bytes internally.) Rows are obtained by calling `mysql_fetch_row()`.

- `MYSQL_FIELD`

  This structure contains metadata: information about a field, such as the field's name, type, and size. Its members are described in more detail later in this section. You may obtain the `MYSQL_FIELD` structures for each field by calling `mysql_fetch_field()` repeatedly. Field values are not part of this structure; they are contained in a `MYSQL_ROW` structure.

- `MYSQL_FIELD_OFFSET`

  This is a type-safe representation of an offset into a MySQL field list. (Used by `mysql_field_seek()`.) Offsets are field numbers within a row, beginning at zero.

- `my_ulonglong`

  A type used for 64-bit unsigned integers. The `my_ulonglong` type was used before MySQL 8.0.18. As of MySQL 8.0.18, use the `uint64_t` C type instead.

- `my_bool`

  A boolean type, for values that are true (nonzero) or false (zero). The `my_bool` type was used before MySQL 8.0. As of MySQL 8.0, use the `bool` or `int` C type instead.

Note

The change from `my_bool` to `bool` means that the `mysql.h` header file requires a C++ or C99 compiler to compile.

The `MYSQL_FIELD` structure contains the members described in the following list. The definitions apply primarily for columns of result sets such as those produced by `SELECT` statements. `MYSQL_FIELD` structures are also used to provide metadata for `OUT` and `INOUT` parameters returned from stored procedures executed using prepared `CALL` statements. For such parameters, some of the structure members have a meaning different from the meaning for column values.

- `char * name`

  The name of the field, as a null-terminated string. If the field was given an alias with an `AS` clause, the value of `name` is the alias. For a procedure parameter, the parameter name.

- `char * org_name`

The name of the field, as a null-terminated string. Aliases are ignored. For expressions, the value is an empty string. For a procedure parameter, the parameter name.

- `char * table`

  The name of the table containing this field, if it is not a calculated field. For calculated fields, the `table` value is an empty string. If the column is selected from a view, `table` names the view. If the table or view was given an alias with an `AS` clause, the value of `table` is the alias. For a `UNION`, the value is the empty string. For a procedure parameter, the procedure name.

- `char * org_table`

  The name of the table, as a null-terminated string. Aliases are ignored. If the column is selected from a view, `org_table` names the view. If the column is selected from a derived table, `org_table` names the base table. If a derived table wraps a view, `org_table` still names the base table. If the column is an expression, `org_table` is the empty string. For a `UNION`, the value is the empty string. For a procedure parameter, the value is the procedure name.

- `char * db`

  The name of the database that the field comes from, as a null-terminated string. If the field is a calculated field, `db` is an empty string. For a `UNION`, the value is the empty string. For a procedure parameter, the name of the database containing the procedure.

- `char * catalog`

  The catalog name. This value is always `"def"`.

- `char * def`

  The default value of this field, as a null-terminated string. This is set only if you use `mysql_list_fields()`.

- `unsigned long length`

  The width of the field. This corresponds to the display length, in bytes.

  The server determines the `length` value before it generates the result set, so this is the minimum length required for a data type capable of holding the largest possible value from the result column, without knowing in advance the actual values that will be produced by the query for the result set.

- `unsigned long max_length`

  The maximum width of the field for the result set (the length in bytes of the longest field value for the rows actually in the result set). If you use `mysql_store_result()` or `mysql_list_fields()`, this contains the maximum length for the field. If you use `mysql_use_result()`, the value of this variable is zero.

  The value of `max_length` is the length of the string representation of the values in the result set. For example, if you retrieve a `FLOAT` column and the “widest” value is `-12.345`, `max_length` is 7 (the length of `'-12.345'`).

  If you are using prepared statements, `max_length` is not set by default because for the binary protocol the lengths of the values depend on the types of the values in the result set. (See Section 28.7.9, “C API Prepared Statement Data Structures”.) If you want the `max_length` values anyway, enable the `STMT_ATTR_UPDATE_MAX_LENGTH` option with `mysql_stmt_attr_set()` and the lengths will be set

when you call `mysql_stmt_store_result()`. (See Section 28.7.11.3, “mysql_stmt_attr_set()”, and Section 28.7.11.28, “mysql_stmt_store_result()”.)

- `unsigned int name_length`

  The length of `name`.

- `unsigned int org_name_length`

  The length of `org_name`.

- `unsigned int table_length`

  The length of `table`.

- `unsigned int org_table_length`

  The length of `org_table`.

- `unsigned int db_length`

  The length of `db`.

- `unsigned int catalog_length`

  The length of `catalog`.

- `unsigned int def_length`

  The length of `def`.

- `unsigned int flags`

  Bit-flags that describe the field. The `flags` value may have zero or more of the bits set that are shown in the following table.

Flag Value	Flag Description
`NOT_NULL_FLAG`	Field cannot be `NULL`
`PRI_KEY_FLAG`	Field is part of a primary key
`UNIQUE_KEY_FLAG`	Field is part of a unique key
`MULTIPLE_KEY_FLAG`	Field is part of a nonunique key
`UNSIGNED_FLAG`	Field has the `UNSIGNED` attribute
`ZEROFILL_FLAG`	Field has the `ZEROFILL` attribute
`BINARY_FLAG`	Field has the `BINARY` attribute
`AUTO_INCREMENT_FLAG`	Field has the `AUTO_INCREMENT` attribute
`ENUM_FLAG`	Field is an `ENUM`
`SET_FLAG`	Field is a `SET`
`BLOB_FLAG`	Field is a `BLOB` or `TEXT` (deprecated)
`TIMESTAMP_FLAG`	Field is a `TIMESTAMP` (deprecated)
`NUM_FLAG`	Field is numeric; see additional notes following table
`NO_DEFAULT_VALUE_FLAG`	Field has no default value; see additional notes following table

Some of these flags indicate data type information and are superseded by or used in conjunction with the `MYSQL_TYPE_xxx` value in the `field->type` member described later:

- To check for `BLOB` or `TIMESTAMP` values, check whether `type` is `MYSQL_TYPE_BLOB` or `MYSQL_TYPE_TIMESTAMP`. (The `BLOB_FLAG` and `TIMESTAMP_FLAG` flags are unneeded.)
- `ENUM` and `SET` values are returned as strings. For these, check that the `type` value is `MYSQL_TYPE_STRING` and that the `ENUM_FLAG` or `SET_FLAG` flag is set in the `flags` value.

`NUM_FLAG` indicates that a column is numeric. This includes columns with a type of `MYSQL_TYPE_DECIMAL`, `MYSQL_TYPE_NEWDECIMAL`, `MYSQL_TYPE_TINY`, `MYSQL_TYPE_SHORT`, `MYSQL_TYPE_LONG`, `MYSQL_TYPE_FLOAT`, `MYSQL_TYPE_DOUBLE`, `MYSQL_TYPE_NULL`, `MYSQL_TYPE_LONGLONG`, `MYSQL_TYPE_INT24`, and `MYSQL_TYPE_YEAR`.

`NO_DEFAULT_VALUE_FLAG` indicates that a column has no `DEFAULT` clause in its definition. This does not apply to `NULL` columns (because such columns have a default of `NULL`), or to `AUTO_INCREMENT` columns (which have an implied default value).

The following example illustrates a typical use of the `flags` value:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field cannot be null\n");
```

You may use the convenience macros shown in the following table to determine the boolean status of the `flags` value.

Flag Status	Description
`IS_NOT_NULL(flags)`	True if this field is defined as `NOT NULL`
`IS_PRI_KEY(flags)`	True if this field is a primary key
`IS_BLOB(flags)`	True if this field is a `BLOB` or `TEXT` (deprecated; test `field->type` instead)

- `unsigned int decimals`

  The number of decimals for numeric fields, and the fractional seconds precision for temporal fields.

- `unsigned int charsetnr`

  An ID number that indicates the character set/collation pair for the field.

  Normally, character values in result sets are converted to the character set indicated by the `character_set_results` system variable. In this case, `charsetnr` corresponds to the character set indicated by that variable. Character set conversion can be suppressed by setting `character_set_results` to `NULL`. In this case, `charsetnr` corresponds to the character set of the original table column or expression. See also Section 10.4, “Connection Character Sets and Collations”.

  To distinguish between binary and nonbinary data for string data types, check whether the `charsetnr` value is 63. If so, the character set is `binary`, which indicates binary rather than nonbinary data. This enables you to distinguish `BINARY` from `CHAR`, `VARBINARY` from `VARCHAR`, and the `BLOB` types from the `TEXT` types.

  `charsetnr` values are the same as those displayed in the `Id` column of the `SHOW COLLATION` statement or the `ID` column of the `INFORMATION_SCHEMA COLLATIONS` table. You can use those information sources to see which character set and collation specific `charsetnr` values indicate:

```
mysql> SHOW COLLATION WHERE Id = 63;
+-----------+---------+----+---------+----------+---------+
| Collation | Charset | Id | Default | Compiled | Sortlen |
+-----------+---------+----+---------+----------+---------+
| binary    | binary  | 63 | Yes     | Yes      |       1 |
+-----------+---------+----+---------+----------+---------+

mysql> SELECT COLLATION_NAME, CHARACTER_SET_NAME
       FROM INFORMATION_SCHEMA.COLLATIONS WHERE ID = 33;
+-----------------+--------------------+
| COLLATION_NAME  | CHARACTER_SET_NAME |
+-----------------+--------------------+
| utf8_general_ci | utf8               |
+-----------------+--------------------+
```

- `enum enum_field_types type`

  The type of the field. The `type` value may be one of the `MYSQL_TYPE_` symbols shown in the following table.

Type Value	Type Description
`MYSQL_TYPE_TINY`	`TINYINT` field
`MYSQL_TYPE_SHORT`	`SMALLINT` field
`MYSQL_TYPE_LONG`	`INTEGER` field
`MYSQL_TYPE_INT24`	`MEDIUMINT` field
`MYSQL_TYPE_LONGLONG`	`BIGINT` field
`MYSQL_TYPE_DECIMAL`	`DECIMAL` or `NUMERIC` field
`MYSQL_TYPE_NEWDECIMAL`	Precision math `DECIMAL` or `NUMERIC`
`MYSQL_TYPE_FLOAT`	`FLOAT` field
`MYSQL_TYPE_DOUBLE`	`DOUBLE` or `REAL` field
`MYSQL_TYPE_BIT`	`BIT` field
`MYSQL_TYPE_TIMESTAMP`	`TIMESTAMP` field
`MYSQL_TYPE_DATE`	`DATE` field
`MYSQL_TYPE_TIME`	`TIME` field
`MYSQL_TYPE_DATETIME`	`DATETIME` field
`MYSQL_TYPE_YEAR`	`YEAR` field
`MYSQL_TYPE_STRING`	`CHAR` or `BINARY` field
`MYSQL_TYPE_VAR_STRING`	`VARCHAR` or `VARBINARY` field
`MYSQL_TYPE_BLOB`	`BLOB` or `TEXT` field (use `max_length` to determine the maximum length)
`MYSQL_TYPE_SET`	`SET` field
`MYSQL_TYPE_ENUM`	`ENUM` field
`MYSQL_TYPE_GEOMETRY`	Spatial field
`MYSQL_TYPE_NULL`	`NULL`-type field

The MYSQL_TYPE_TIME2, MYSQL_TYPE_DATETIME2, and MYSQL_TYPE_TIMESTAMP2) type codes are used only on the server side. Clients see the MYSQL_TYPE_TIME, MYSQL_TYPE_DATETIME, and MYSQL_TYPE_TIMESTAMP codes.

You can use the IS_NUM() macro to test whether a field has a numeric type. Pass the type value to IS_NUM() and it evaluates to TRUE if the field is numeric:

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

ENUM and SET values are returned as strings. For these, check that the type value is MYSQL_TYPE_STRING and that the ENUM_FLAG or SET_FLAG flag is set in the flags value.

28.7.6 C API Function Overview

The following list summarizes the functions available in the C API. For greater detail, see the descriptions in Section 28.7.7, “C API Function Descriptions”.

- my_init(): Initializes global variables and the thread handler in thread-safe programs.
- mysql_affected_rows(): Returns the number of rows changed/deleted/inserted by the last UPDATE, DELETE, or INSERT query.
- mysql_autocommit(): Toggles autocommit mode on/off.
- mysql_change_user(): Changes the user and database on an open connection.
- mysql_character_set_name(): Returns the default character set name for the current connection.
- mysql_client_find_plugin(): Returns a pointer to a plugin.
- mysql_client_register_plugin(): Registers a plugin.
- mysql_close(): Closes a server connection.
- mysql_commit(): Commits the transaction.
- mysql_connect(): Connects to a MySQL server. This function is deprecated; use mysql_real_connect() instead.
- mysql_create_db(): Creates a database. This function is deprecated; use the SQL statement CREATE DATABASE instead.
- mysql_data_seek(): Seeks to an arbitrary row number in a query result set.
- mysql_debug(): Does a DBUG_PUSH with the given string.
- mysql_drop_db(): Drops a database. This function is deprecated; use the SQL statement DROP DATABASE instead.
- mysql_dump_debug_info(): Causes the server to write debug information to the log.
- mysql_eof(): Determines whether the last row of a result set has been read. This function is deprecated; use mysql_errno() or mysql_error() instead.
- mysql_errno(): Returns the error number for the most recently invoked MySQL function.

- `mysql_error()`: Returns the error message for the most recently invoked MySQL function.
- `mysql_escape_string()`: Escapes special characters in a string for use in an SQL statement.
- `mysql_fetch_field()`: Returns the type of the next table field.
- `mysql_fetch_field_direct()`: Returns the type of a table field, given a field number.
- `mysql_fetch_fields()`: Returns an array of all field structures.
- `mysql_fetch_lengths()`: Returns the lengths of all columns in the current row.
- `mysql_fetch_row()`: Fetches the next row from the result set.
- `mysql_field_count()`: Returns the number of result columns for the most recent statement.
- `mysql_field_seek()`: Puts the column cursor on a specified column.
- `mysql_field_tell()`: Returns the position of the field cursor used for the last `mysql_fetch_field()`.
- `mysql_free_result()`: Frees memory used by a result set.
- `mysql_get_character_set_info()`: Returns information about default character set.
- `mysql_get_client_info()`: Returns client version information as a string.
- `mysql_get_client_version()`: Returns client version information as an integer.
- `mysql_get_host_info()`: Returns a string describing the connection.
- `mysql_get_option()`: Returns the value of a `mysql_options()` option.
- `mysql_get_proto_info()`: Returns the protocol version used by the connection.
- `mysql_get_server_info()`: Returns the server version number.
- `mysql_get_server_version()`: Returns the server version number as an integer.
- `mysql_get_ssl_cipher()`: Returns the current SSL cipher.
- `mysql_hex_string()`: Encodes a string in hexadecimal format.
- `mysql_info()`: Returns information about the most recently executed query.
- `mysql_init()`: Gets or initializes a `MYSQL` structure.
- `mysql_insert_id()`: Returns the ID generated for an `AUTO_INCREMENT` column by the previous query.
- `mysql_kill()`: Kills a given thread.
- `mysql_library_end()`: Finalizes the MySQL C API library.
- `mysql_library_init()`: Initializes the MySQL C API library.
- `mysql_list_dbs()`: Returns database names matching a simple regular expression.
- `mysql_list_fields()`: Returns field names matching a simple regular expression.

- `mysql_list_processes()`: Returns a list of the current server threads.
- `mysql_list_tables()`: Returns table names matching a simple regular expression.
- `mysql_load_plugin()`: Loads a plugin.
- `mysql_load_plugin_v()`: Loads a plugin.
- `mysql_more_results()`: Checks whether any more results exist.
- `mysql_next_result()`: Returns/initiates the next result in multiple-result executions.
- `mysql_num_fields()`: Returns the number of columns in a result set.
- `mysql_num_rows()`: Returns the number of rows in a result set.
- `mysql_options()`: Sets connect options for `mysql_real_connect()`.
- `mysql_options4()`: Sets connect options for `mysql_real_connect()`.
- `mysql_ping()`: Checks whether the connection to the server is working, reconnecting as necessary.
- `mysql_plugin_options()`: Sets a plugin option.
- `mysql_query()`: Executes an SQL query specified as a null-terminated string.
- `mysql_real_connect()`: Connects to a MySQL server.
- `mysql_real_escape_string()`: Escapes special characters in a string for use in an SQL statement, taking into account the current character set of the connection.
- `mysql_real_escape_string_quote()`: Escapes special characters in a string for use in an SQL statement, taking into account the current character set of the connection and the quoting context.
- `mysql_real_query()`: Executes an SQL query specified as a counted string.
- `mysql_refresh()`: Flushes or resets tables and caches.
- `mysql_reload()`: Tells the server to reload the grant tables.
- `mysql_reset_connection()`: Resets the connection to clear session state.
- `mysql_reset_server_public_key()`: Clears a cached RSA public key from the client library.
- `mysql_result_metadata()`: Whether a result set has metadata.
- `mysql_rollback()`: Rolls back the transaction.
- `mysql_row_seek()`: Seeks to a row offset in a result set, using value returned from `mysql_row_tell()`.
- `mysql_row_tell()`: Returns the row cursor position.
- `mysql_select_db()`: Selects a database.
- `mysql_server_end()`: Finalizes the MySQL C API library.
- `mysql_server_init()`: Initializes the MySQL C API library.
- `mysql_session_track_get_first()`: Gets the first part of session state-change information.

- `mysql_session_track_get_next()`: Gets the next part of session state-change information.
- `mysql_set_character_set()`: Sets the default character set for the current connection.
- `mysql_set_local_infile_default()`: Sets the `LOAD DATA LOCAL` handler callbacks to their default values.
- `mysql_set_local_infile_handler()`: Installs application-specific `LOAD DATA LOCAL` handler callbacks.
- `mysql_set_server_option()`: Sets an option for the connection (like `multi-statements`).
- `mysql_sqlstate()`: Returns the SQLSTATE error code for the last error.
- `mysql_shutdown()`: Shuts down the database server.
- `mysql_ssl_set()`: Prepares to establish an SSL connection to the server.
- `mysql_stat()`: Returns the server status as a string.
- `mysql_store_result()`: Retrieves a complete result set to the client.
- `mysql_thread_end()`: Finalizes a thread handler.
- `mysql_thread_id()`: Returns the current thread ID.
- `mysql_thread_init()`: Initializes a thread handler.
- `mysql_thread_safe()`: Returns 1 if the clients are compiled as thread-safe.
- `mysql_use_result()`: Initiates a row-by-row result set retrieval.
- `mysql_warning_count()`: Returns the warning count for the previous SQL statement.

Application programs should use this general outline for interacting with MySQL:

1. Initialize the MySQL client library by calling `mysql_library_init()`.
2. Initialize a connection handler by calling `mysql_init()` and connect to the server by calling `mysql_real_connect()`.
3. Issue SQL statements and process their results. (The following discussion provides more information about how to do this.)
4. Close the connection to the MySQL server by calling `mysql_close()`.
5. End use of the MySQL client library by calling `mysql_library_end()`.

The purpose of calling `mysql_library_init()` and `mysql_library_end()` is to provide proper initialization and finalization of the MySQL client library. For applications that are linked with the client library, they provide improved memory management. If you do not call `mysql_library_end()`, a block of memory remains allocated. (This does not increase the amount of memory used by the application, but some memory leak detectors will complain about it.)

In a nonmultithreaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multithreaded environment, and thus neither is `mysql_init()`, which calls `mysql_library_init()`. You must either call `mysql_library_init()` prior to spawning any threads,

or else use a mutex to protect the call, whether you invoke `mysql_library_init()` or indirectly through `mysql_init()`. This should be done prior to any other client library call.

To connect to the server, call `mysql_init()` to initialize a connection handler, then call `mysql_real_connect()` with that handler (along with other information such as the host name, user name, and password). Upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API older than 5.0.3, or `0` in newer versions. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. You can use the `MYSQL_OPT_RECONNECT` option to `mysql_options()` to control reconnection behavior. When you are done with the connection, call `mysql_close()` to terminate it. Do not use the handler after it has been closed.

While a connection is active, the client may send SQL statements to the server using `mysql_query()` or `mysql_real_query()`. The difference between the two is that `mysql_query()` expects the query to be specified as a null-terminated string whereas `mysql_real_query()` expects a counted string. If the string contains binary data (which may include null bytes), you must use `mysql_real_query()`.

For each non-`SELECT` query (for example, `INSERT`, `UPDATE`, `DELETE`), you can find out how many rows were changed (affected) by calling `mysql_affected_rows()`.

For `SELECT` queries, you retrieve the selected rows as a result set. (Note that some statements are `SELECT`-like in that they return rows. These include `SHOW`, `DESCRIBE`, and `EXPLAIN`. Treat these statements the same way as `SELECT` statements.)

There are two ways for a client to process result sets. One way is to retrieve the entire result set all at once by calling `mysql_store_result()`. This function acquires from the server all the rows returned by the query and stores them in the client. The second way is for the client to initiate a row-by-row result set retrieval by calling `mysql_use_result()`. This function initializes the retrieval, but does not actually get any rows from the server.

In both cases, you access rows by calling `mysql_fetch_row()`. With `mysql_store_result()`, `mysql_fetch_row()` accesses rows that have previously been fetched from the server. With `mysql_use_result()`, `mysql_fetch_row()` actually retrieves the row from the server. Information about the size of the data in each row is available by calling `mysql_fetch_lengths()`.

After you are done with a result set, call `mysql_free_result()` to free the memory used for it.

The two retrieval mechanisms are complementary. Choose the approach that is most appropriate for each client application. In practice, clients tend to use `mysql_store_result()` more commonly.

An advantage of `mysql_store_result()` is that because the rows have all been fetched to the client, you not only can access rows sequentially, you can move back and forth in the result set using `mysql_data_seek()` or `mysql_row_seek()` to change the current row position within the result set. You can also find out how many rows there are by calling `mysql_num_rows()`. On the other hand, the memory requirements for `mysql_store_result()` may be very high for large result sets and you are more likely to encounter out-of-memory conditions.

An advantage of `mysql_use_result()` is that the client requires less memory for the result set because it maintains only one row at a time (and because there is less allocation overhead, `mysql_use_result()` can be faster). Disadvantages are that you must process each row quickly to avoid tying up the server, you do not have random access to rows within the result set (you can only access rows sequentially), and the number of rows in the result set is unknown until you have retrieved them all. Furthermore, you *must* retrieve all the rows even if you determine in mid-retrieval that you've found the information you were looking for.

The API makes it possible for clients to respond appropriately to statements (retrieving rows only as necessary) without knowing whether the statement is a `SELECT`. You can do this by calling `mysql_store_result()` after each `mysql_query()` (or `mysql_real_query()`). If the result set call succeeds, the statement was a `SELECT` and you can read the rows. If the result set call fails, call `mysql_field_count()` to determine whether a result was actually to be expected. If `mysql_field_count()` returns zero, the statement returned no data (indicating that it was an `INSERT`, `UPDATE`, `DELETE`, and so forth), and was not expected to return rows. If `mysql_field_count()` is nonzero, the statement should have returned rows, but did not. This indicates that the statement was a `SELECT` that failed. See the description for `mysql_field_count()` for an example of how this can be done.

Both `mysql_store_result()` and `mysql_use_result()` enable you to obtain information about the fields that make up the result set (the number of fields, their names and types, and so forth). You can access field information sequentially within the row by calling `mysql_fetch_field()` repeatedly, or by field number within the row by calling `mysql_fetch_field_direct()`. The current field cursor position may be changed by calling `mysql_field_seek()`. Setting the field cursor affects subsequent calls to `mysql_fetch_field()`. You can also get information for fields all at once by calling `mysql_fetch_fields()`.

For detecting and reporting errors, MySQL provides access to error information by means of the `mysql_errno()` and `mysql_error()` functions. These return the error code or error message for the most recently invoked function that can succeed or fail, enabling you to determine when an error occurred and what it was.

28.7.7 C API Function Descriptions

This section describes C API functions other than those used for prepared statements, the asychronous interface, or the replication stream interface. For information about those, see Section 28.7.11, “C API Prepared Statement Function Descriptions”, Section 28.7.15, “C API Asynchronous Function Descriptions”, and Section 28.7.21, “C API Binary Log Function Descriptions”.

In the descriptions here, a parameter or return value of `NULL` means `NULL` in the sense of the C programming language, not a MySQL `NULL` value.

Functions that return a value generally return a pointer or an integer. Unless specified otherwise, functions returning a pointer return a non-`NULL` value to indicate success or a `NULL` value to indicate an error, and functions returning an integer return zero to indicate success or nonzero to indicate an error. Note that “nonzero” means just that. Unless the function description says otherwise, do not test against a value other than zero:

```
if (result)                  /* correct */
    ... error ...

if (result < 0)              /* incorrect */
    ... error ...

if (result == -1)            /* incorrect */
    ... error ...
```

When a function returns an error, the **Errors** subsection of the function description lists the possible types of errors. You can find out which of these occurred by calling `mysql_errno()`. A string representation of the error may be obtained by calling `mysql_error()`.

28.7.7.1 mysql_affected_rows()

```
uint64_t mysql_affected_rows(MYSQL *mysql)
```

Description

`mysql_affected_rows()` may be called immediately after executing a statement with `mysql_query()` or `mysql_real_query()`. It returns the number of rows changed, deleted, or inserted by the last statement if it was an `UPDATE`, `DELETE`, or `INSERT`. For `SELECT` statements, `mysql_affected_rows()` works like `mysql_num_rows()`.

For `UPDATE` statements, the affected-rows value by default is the number of rows actually changed. If you specify the `CLIENT_FOUND_ROWS` flag to `mysql_real_connect()` when connecting to `mysqld`, the affected-rows value is the number of rows “found”; that is, matched by the `WHERE` clause.

For `REPLACE` statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

For `INSERT ... ON DUPLICATE KEY UPDATE` statements, the affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the `CLIENT_FOUND_ROWS` flag, the affected-rows value is 1 (not 0) if an existing row is set to its current values.

Following a `CALL` statement for a stored procedure, `mysql_affected_rows()` returns the value that it would return for the last statement executed within the procedure, or `0` if that statement would return `-1`. Within the procedure, you can use `ROW_COUNT()` at the SQL level to obtain the affected-rows value for individual statements.

`mysql_affected_rows()` returns a meaningful value for a wide range of statements. For details, see the description for `ROW_COUNT()` in Section 12.15, “Information Functions”.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an `UPDATE` statement, no rows matched the `WHERE` clause in the query or that no query has yet been executed. -1 indicates that the query returned an error or that, for a `SELECT` query, `mysql_affected_rows()` was called prior to calling `mysql_store_result()`.

Because `mysql_affected_rows()` returns an unsigned value, you can check for -1 by comparing the return value to `(uint64_t)-1` (or to `(uint64_t)~0`, which is equivalent).

Errors

None.

Example

```
char *stmt = "UPDATE products SET cost=cost*1.25
              WHERE group=10";
mysql_query(&mysql,stmt);
printf("%ld products updated",
       (long) mysql_affected_rows(&mysql));
```

28.7.7.2 mysql_autocommit()

```
bool mysql_autocommit(MYSQL *mysql, bool mode)
```

Description

Sets autocommit mode on if `mode` is 1, off if `mode` is 0.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

28.7.7.3 mysql_change_user()

```
bool mysql_change_user(MYSQL *mysql, const char *user, const char *password,
const char *db)
```

Description

Changes the user and causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that include no explicit database specifier.

`mysql_change_user()` fails if the connected user cannot be authenticated or does not have permission to use the database. In this case, the user and database are not changed.

Pass a `db` parameter of `NULL` if you do not want to have a default database.

This function resets the session state as if one had done a new connect and reauthenticated. (See Section 28.7.28, “C API Automatic Reconnection Control”.) It always performs a `ROLLBACK` of any active transactions, closes and drops all temporary tables, and unlocks all locked tables. Session system variables are reset to the values of the corresponding global system variables. Prepared statements are released and `HANDLER` variables are closed. Locks acquired with `GET_LOCK()` are released. These effects occur even if the user did not change.

To reset the connection state in a more lightweight manner without changing the user, use `mysql_reset_connection()`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

The same that you can get from `mysql_real_connect()`, plus:

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

- `ER_UNKNOWN_COM_ERROR`

  The MySQL server does not implement this command (probably an old server).

- `ER_ACCESS_DENIED_ERROR`

  The user or password was wrong.

- `ER_BAD_DB_ERROR`

  The database did not exist.

- `ER_DBACCESS_DENIED_ERROR`

  The user did not have access rights to the database.

- `ER_WRONG_DB_NAME`

  The database name was too long.

Example

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
   fprintf(stderr, "Failed to change user.  Error: %s\n",
           mysql_error(&mysql));
}
```

28.7.7.4 mysql_character_set_name()

```
const char *mysql_character_set_name(MYSQL *mysql)
```

Description

Returns the default character set name for the current connection.

Return Values

The default character set name

Errors

None.

28.7.7.5 mysql_close()

```
void mysql_close(MYSQL *mysql)
```

Description

Closes a previously opened connection. `mysql_close()` also deallocates the connection handler pointed to by `mysql` if the handler was allocated automatically by `mysql_init()` or `mysql_connect()`. Do not use the handler after it has been closed.

Return Values

None.

Errors

None.

28.7.7.6 mysql_commit()

```
bool mysql_commit(MYSQL *mysql)
```

Description

Commits the current transaction.

The action of this function is subject to the value of the `completion_type` system variable. In particular, if the value of `completion_type` is `RELEASE` (or 2), the server performs a release after terminating a transaction and closes the client connection. Call `mysql_close()` from the client program to close the connection from the client side.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

28.7.7.7 mysql_connect()

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const
char *passwd)
```

Description

This function is deprecated. Use `mysql_real_connect()` instead.

`mysql_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The meanings of the parameters are the same as for the corresponding parameters for `mysql_real_connect()` with the difference that the connection parameter may be `NULL`. In this case, the C API allocates memory for the connection structure automatically and frees it when you call `mysql_close()`. The disadvantage of this approach is that you cannot retrieve an error message if the connection fails. (To get error information from `mysql_errno()` or `mysql_error()`, you must provide a valid `MYSQL` pointer.)

Return Values

Same as for `mysql_real_connect()`.

Errors

Same as for `mysql_real_connect()`.

28.7.7.8 mysql_create_db()

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

Description

Creates the database named by the `db` parameter.

This function is deprecated. Use `mysql_query()` to issue an SQL `CREATE DATABASE` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

```
if(mysql_create_db(&mysql, "my_database"))
{
   fprintf(stderr, "Failed to create new database.  Error: %s\n",
           mysql_error(&mysql));
}
```

28.7.7.9 mysql_data_seek()

```
void mysql_data_seek(MYSQL_RES *result, uint64_t offset)
```

Description

Seeks to an arbitrary row in a query result set. The `offset` value is a row number. Specify a value in the range from `0` to `mysql_num_rows(result)-1`.

This function requires that the result set structure contains the entire result of the query, so `mysql_data_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

None.

Errors

None.

28.7.7.10 mysql_debug()

```
void mysql_debug(const char *debug)
```

Description

Does a `DBUG_PUSH` with the given string. `mysql_debug()` uses the Fred Fish debug library. To use this function, you must compile the client library to support debugging. See Section 29.5.4, “The DBUG Package”.

Return Values

None.

Errors

None.

Example

The call shown here causes the client library to generate a trace file in `/tmp/client.trace` on the client machine:

```
mysql_debug("d:t:O,/tmp/client.trace");
```

28.7.7.11 mysql_drop_db()

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Description

Drops the database named by the `db` parameter.

This function is deprecated. Use `mysql_query()` to issue an SQL `DROP DATABASE` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

```
if(mysql_drop_db(&mysql, "my_database"))
  fprintf(stderr, "Failed to drop the database: Error: %s\n",
          mysql_error(&mysql));
```

28.7.7.12 mysql_dump_debug_info()

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Description

Instructs the server to write debugging information to the error log. The connected user must have the `SUPER` privilege.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.13 mysql_eof()

```
bool mysql_eof(MYSQL_RES *result)
```

Description

This function is deprecated. `mysql_errno()` or `mysql_error()` may be used instead.

`mysql_eof()` determines whether the last row of a result set has been read.

If you acquire a result set from a successful call to `mysql_store_result()`, the client receives the entire set in one operation. In this case, a `NULL` return from `mysql_fetch_row()` always means the

end of the result set has been reached and it is unnecessary to call `mysql_eof()`. When used with `mysql_store_result()`, `mysql_eof()` always returns true.

On the other hand, if you use `mysql_use_result()` to initiate a result set retrieval, the rows of the set are obtained from the server one by one as you call `mysql_fetch_row()` repeatedly. Because an error may occur on the connection during this process, a `NULL` return value from `mysql_fetch_row()` does not necessarily mean the end of the result set was reached normally. In this case, you can use `mysql_eof()` to determine what happened. `mysql_eof()` returns a nonzero value if the end of the result set was reached and zero if an error occurred.

Historically, `mysql_eof()` predates the standard MySQL error functions `mysql_errno()` and `mysql_error()`. Because those error functions provide the same information, their use is preferred over `mysql_eof()`, which is deprecated. (In fact, they provide more information, because `mysql_eof()` returns only a boolean value whereas the error functions indicate a reason for the error when one occurs.)

Return Values

Zero for success. Nonzero if the end of the result set has been reached.

Errors

None.

Example

The following example shows how you might use `mysql_eof()`:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

However, you can achieve the same effect with the standard MySQL error functions:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

28.7.7.14 mysql_errno()

```
unsigned int mysql_errno(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_errno()` returns the error code for the most recently invoked API function that can succeed or fail. A return value of zero means that no error occurred. Client

error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at Appendix B, *Errors, Error Codes, and Common Problems*.

Note

Some functions such as `mysql_fetch_row()` do not set `mysql_errno()` if they succeed. A rule of thumb is that all functions that have to ask the server for information reset `mysql_errno()` if they succeed.

MySQL-specific error numbers returned by `mysql_errno()` differ from SQLSTATE values returned by `mysql_sqlstate()`. For example, the `mysql` client program displays errors using the following format, where `1146` is the `mysql_errno()` value and `'42S02'` is the corresponding `mysql_sqlstate()` value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Return Values

An error code value for the last `mysql_xxx()` call, if it failed. zero means no error occurred.

Errors

None.

28.7.7.15 mysql_error()

```
const char *mysql_error(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_error()` returns a null-terminated string containing the error message for the most recently invoked API function that failed. If a function did not fail, the return value of `mysql_error()` may be the previous error or an empty string to indicate no error.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_error()` if they succeed.

For functions that reset `mysql_error()`, either of these two tests can be used to check for an error:

```
if(*mysql_error(&mysql))
{
  // an error occurred
}

if(mysql_error(&mysql)[0])
{
  // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. You can choose error messages in several different languages. See Section 10.12, “Setting the Error Message Language”.

Return Values

A null-terminated character string that describes the error. An empty string if no error occurred.

Errors

None.

28.7.7.16 mysql_escape_string()

Note

Do not use this function. `mysql_escape_string()` does not have arguments that enable it to respect the current character set or the quoting context. Use `mysql_real_escape_string_quote()` instead.

28.7.7.17 mysql_fetch_field()

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

Description

Returns the definition of one column of a result set as a `MYSQL_FIELD` structure. Call this function repeatedly to retrieve information about all columns in the result set. `mysql_fetch_field()` returns `NULL` when no more fields are left.

For metadata-optional connections, this function returns `NULL` when the `resultset_metadata` system variable is set to `NONE`. To check whether a result set has metadata, use the `mysql_result_metadata()` function. For details about managing result set metadata transfer, see Section 28.7.27, “C API Optional Result Set Metadata”.

`mysql_fetch_field()` is reset to return information about the first field each time you execute a new `SELECT` query. The field returned by `mysql_fetch_field()` is also affected by calls to `mysql_field_seek()`.

If you've called `mysql_query()` to perform a `SELECT` on a table but have not called `mysql_store_result()`, MySQL returns the default blob length (8KB) if you call `mysql_fetch_field()` to ask for the length of a `BLOB` field. (The 8KB size is chosen because MySQL does not know the maximum length for the `BLOB`. This should be made configurable sometime.) Once you've retrieved the result set, `field->max_length` contains the length of the largest value for this column in the specific query.

Return Values

The `MYSQL_FIELD` structure for the current column. `NULL` if no columns are left or the result set has no metadata.

Errors

None.

Example

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
```

```
}
```

28.7.7.18 mysql_fetch_field_direct()

```
MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int fieldnr)
```

Description

Given a field number `fieldnr` for a column within a result set, returns that column's field definition as a `MYSQL_FIELD` structure. Use this function to retrieve the definition for an arbitrary column. Specify a value for `fieldnr` in the range from 0 to `mysql_num_fields(result)-1`.

For metadata-optional connections, this function returns `NULL` when the `resultset_metadata` system variable is set to `NONE`. To check whether a result set has metadata, use the `mysql_result_metadata()` function. For details about managing result set metadata transfer, see Section 28.7.27, “C API Optional Result Set Metadata”.

Return Values

The `MYSQL_FIELD` structure for the specified column. `NULL` if the result set has no metadata.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}
```

28.7.7.19 mysql_fetch_fields()

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

Description

Returns an array of all `MYSQL_FIELD` structures for a result set. Each structure provides the field definition for one column of the result set.

For metadata-optional connections, this function returns `NULL` when the `resultset_metadata` system variable is set to `NONE`. To check whether a result set has metadata, use the `mysql_result_metadata()` function. For details about managing result set metadata transfer, see Section 28.7.27, “C API Optional Result Set Metadata”.

Return Values

An array of `MYSQL_FIELD` structures for all columns of a result set. `NULL` if the result set has no metadata.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
   printf("Field %u is %s\n", i, fields[i].name);
}
```

28.7.7.20 mysql_fetch_lengths()

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Description

Returns the lengths of the columns of the current row within a result set. If you plan to copy field values, this length information is also useful for optimization, because you can avoid calling `strlen()`. In addition, if the result set contains binary data, you **must** use this function to determine the size of the data, because `strlen()` returns incorrect results for any field containing null characters.

The length for empty columns and for columns containing `NULL` values is zero. To see how to distinguish these two cases, see the description for `mysql_fetch_row()`.

Return Values

An array of unsigned long integers representing the size of each column (not including any terminating null bytes). `NULL` if an error occurred.

Errors

`mysql_fetch_lengths()` is valid only for the current row of the result set. It returns `NULL` if you call it before calling `mysql_fetch_row()` or after retrieving all rows in the result.

Example

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
         printf("Column %u is %lu bytes in length.\n",
                i, lengths[i]);
    }
```

```
}
```

28.7.7.21 mysql_fetch_row()

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

Description

Note

`mysql_fetch_row()` is a synchronous function. Its asynchronous counterpart is `mysql_fetch_row_nonblocking()`, for use by applications that require asynchronous communication with the server. See Section 28.7.12, “C API Asynchronous Interface”.

`mysql_fetch_row()` retrieves the next row of a result set:

- When used after `mysql_store_result()` or `mysql_store_result_nonblocking()`, `mysql_fetch_row()` returns `NULL` if there are no more rows to retrieve.
- When used after `mysql_use_result()`, `mysql_fetch_row()` returns `NULL` if there are no more rows to retrieve or an error occurred.

The number of values in the row is given by `mysql_num_fields(result)`. If `row` holds the return value from a call to `mysql_fetch_row()`, pointers to the values are accessed as `row[0]` to `row[mysql_num_fields(result)-1]`. `NULL` values in the row are indicated by `NULL` pointers.

The lengths of the field values in the row may be obtained by calling `mysql_fetch_lengths()`. Empty fields and fields containing `NULL` both have length 0; you can distinguish these by checking the pointer for the field value. If the pointer is `NULL`, the field is `NULL`; otherwise, the field is empty.

Return Values

A `MYSQL_ROW` structure for the next row, or `NULL`. The meaning of a `NULL` return depends on which function was called preceding `mysql_fetch_row()`:

- When used after `mysql_store_result()` or `mysql_store_result_nonblocking()`, `mysql_fetch_row()` returns `NULL` if there are no more rows to retrieve.
- When used after `mysql_use_result()`, `mysql_fetch_row()` returns `NULL` if there are no more rows to retrieve or an error occurred. To determine whether an error occurred, check whether `mysql_error()` returns a nonempty string or `mysql_errno()` returns nonzero.

Errors

Errors are not reset between calls to `mysql_fetch_row()`

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
   unsigned long *lengths;
   lengths = mysql_fetch_lengths(result);
   for(i = 0; i < num_fields; i++)
   {
       printf("[%.*s] ", (int) lengths[i],
              row[i] ? row[i] : "NULL");
   }
   printf("\n");
}
```

28.7.7.22 mysql_field_count()

```
unsigned int mysql_field_count(MYSQL *mysql)
```

Description

Returns the number of columns for the most recent query on the connection.

The normal use of this function is when `mysql_store_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a nonempty result. This enables the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
```

```
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
    }
}
```

An alternative is to replace the `mysql_field_count(&mysql)` call with `mysql_errno(&mysql)`. In this case, you are checking directly for an error from `mysql_store_result()` rather than inferring from the value of `mysql_field_count()` whether the statement was a `SELECT`.

28.7.7.23 mysql_field_seek()

```
MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET
offset)
```

Description

Sets the field cursor to the given offset. The next call to `mysql_fetch_field()` retrieves the field definition of the column associated with that offset.

To seek to the beginning of a row, pass an `offset` value of zero.

Return Values

The previous value of the field cursor.

Errors

None.

28.7.7.24 mysql_field_tell()

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

Description

Returns the position of the field cursor used for the last `mysql_fetch_field()`. This value can be used as an argument to `mysql_field_seek()`.

Return Values

The current offset of the field cursor.

Errors

None.

28.7.7.25 mysql_free_result()

```
void mysql_free_result(MYSQL_RES *result)
```

Description

Note

`mysql_free_result()` is a synchronous function. Its asynchronous counterpart is `mysql_free_result_nonblocking()`, for use by applications that require asynchronous communication with the server. See Section 28.7.12, “C API Asynchronous Interface”.

`mysql_free_result()` frees the memory allocated for a result set by `mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()`, and so forth. When you are done with a result set, you must free the memory it uses by calling `mysql_free_result()`.

Do not attempt to access a result set after freeing it.

Return Values

None.

Errors

None.

28.7.7.26 mysql_get_character_set_info()

```
void mysql_get_character_set_info(MYSQL *mysql, MY_CHARSET_INFO *cs)
```

Description

This function provides information about the default client character set. The default character set may be changed with the `mysql_set_character_set()` function.

Example

This example shows the fields that are available in the `MY_CHARSET_INFO` structure:

```
if (!mysql_set_character_set(&mysql, "utf8"))
{
    MY_CHARSET_INFO cs;
    mysql_get_character_set_info(&mysql, &cs);
    printf("character set information:\n");
    printf("character set+collation number: %d\n", cs.number);
    printf("character set name: %s\n", cs.name);
    printf("collation name: %s\n", cs.csname);
    printf("comment: %s\n", cs.comment);
    printf("directory: %s\n", cs.dir);
    printf("multi byte character min. length: %d\n", cs.mbminlen);
    printf("multi byte character max. length: %d\n", cs.mbmaxlen);
}
```

28.7.7.27 mysql_get_client_info()

```
const char *mysql_get_client_info(void)
```

Description

Returns a string that represents the MySQL client library version (for example, `"8.0.19"`).

The function value is the version of MySQL or Connector/C that provides the client library. For more information, see Section 28.7.4.5, “C API Server and Client Library Versions”.

Return Values

A character string that represents the MySQL client library version.

Errors

None.

28.7.7.28 mysql_get_client_version()

```
unsigned long mysql_get_client_version(void)
```

Description

Returns an integer that represents the MySQL client library version. The value has the format `XYYZZ` where `X` is the major version, `YY` is the release level (or minor version), and `ZZ` is the sub-version within the release level:

```
major_version*10000 + release_level*100 + sub_version
```

For example, `"8.0.19"` is returned as `80019`.

The function value is the version of MySQL or Connector/C that provides the client library. For more information, see Section 28.7.4.5, “C API Server and Client Library Versions”.

Return Values

An integer that represents the MySQL client library version.

Errors

None.

28.7.7.29 mysql_get_host_info()

```
const char *mysql_get_host_info(MYSQL *mysql)
```

Description

Returns a string describing the type of connection in use, including the server host name.

Return Values

A character string representing the server host name and the connection type.

Errors

None.

28.7.7.30 mysql_get_option()

```
int mysql_get_option(MYSQL *mysql, enum mysql_option option, const void *arg)
```

Description

Returns the current value of an option settable using `mysql_options()`. The value should be treated as read only.

The `option` argument is the option for which you want its value. The `arg` argument is a pointer to a variable in which to store the option value. `arg` must be a pointer to a variable of the type appropriate for the `option` argument. The following table shows which variable type to use for each `option` value.

`arg` Type	**Applicable `option` Values**
`unsigned int`	`MYSQL_OPT_CONNECT_TIMEOUT`, `MYSQL_OPT_PROTOCOL`, `MYSQL_OPT_READ_TIMEOUT`, `MYSQL_OPT_RETRY_COUNT`, `MYSQL_OPT_SSL_FIPS_MODE`, `MYSQL_OPT_SSL_MODE`, `MYSQL_OPT_WRITE_TIMEOUT`, `MYSQL_OPT_ZSTD_COMPRESSION_LEVEL`
`unsigned long`	`MYSQL_OPT_MAX_ALLOWED_PACKET`, `MYSQL_OPT_NET_BUFFER_LENGTH`
`bool`	`MYSQL_ENABLE_CLEARTEXT_PLUGIN`, `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS`, `MYSQL_OPT_GET_SERVER_PUBLIC_KEY`, `MYSQL_OPT_LOCAL_INFILE`, `MYSQL_OPT_OPTIONAL_RESULTSET_METADATA`, `MYSQL_OPT_RECONNECT`, `MYSQL_REPORT_DATA_TRUNCATION`
`const char *`	`MYSQL_DEFAULT_AUTH`, `MYSQL_OPT_BIND`, `MYSQL_OPT_COMPRESSION_ALGORITHMS`, `MYSQL_OPT_SSL_CA`, `MYSQL_OPT_SSL_CAPATH`, `MYSQL_OPT_SSL_CERT`, `MYSQL_OPT_SSL_CIPHER`, `MYSQL_OPT_SSL_CRL`, `MYSQL_OPT_SSL_CRLPATH`, `MYSQL_OPT_SSL_KEY`, `MYSQL_OPT_TLS_CIPHERSUITES`, `MYSQL_OPT_TLS_VERSION`, `MYSQL_PLUGIN_DIR`, `MYSQL_READ_DEFAULT_FILE`, `MYSQL_READ_DEFAULT_GROUP`, `MYSQL_SERVER_PUBLIC_KEY`, `MYSQL_SET_CHARSET_DIR`, `MYSQL_SET_CHARSET_NAME`, `MYSQL_SHARED_MEMORY_BASE_NAME`
argument not used	`MYSQL_OPT_COMPRESS`
cannot be queried (error is returned)	`MYSQL_INIT_COMMAND`, `MYSQL_OPT_CONNECT_ATTR_DELETE`, `MYSQL_OPT_CONNECT_ATTR_RESET`, `MYSQL_OPT_NAMED_PIPE`

Return Values

Zero for success. Nonzero if an error occurred; this occurs for `option` values that cannot be queried.

Example

The following call tests the `MYSQL_OPT_RECONNECT` option. After the call returns successfully, the value of `reconnect` is true or false to indicate whether automatic reconnection is enabled.

```
bool reconnect;

if (mysql_get_option(mysql, MYSQL_OPT_RECONNECT, &reconnect))
  fprintf(stderr, "mysql_get_options() failed\n");
```

28.7.7.31 mysql_get_proto_info()

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

Description

Returns the protocol version used by current connection.

Return Values

An unsigned integer representing the protocol version used by the current connection.

Errors

None.

28.7.7.32 mysql_get_server_info()

```
const char *mysql_get_server_info(MYSQL *mysql)
```

Description

Returns a string that represents the MySQL server version (for example, `"8.0.19"`).

Return Values

A character string that represents the MySQL server version.

Errors

None.

28.7.7.33 mysql_get_server_version()

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Description

Returns an integer that represents the MySQL server version. The value has the format `XYYZZ` where `X` is the major version, `YY` is the release level (or minor version), and `ZZ` is the sub-version within the release level:

```
major_version*10000 + release_level*100 + sub_version
```

For example, `"8.0.19"` is returned as `80019`.

This function is useful in client programs for determining whether some version-specific server capability exists.

Return Values

An integer that represents the MySQL server version.

Errors

None.

28.7.7.34 mysql_get_ssl_cipher()

```
const char *mysql_get_ssl_cipher(MYSQL *mysql)
```

Description

`mysql_get_ssl_cipher()` returns the encryption cipher used for the given connection to the server. `mysql` is the connection handler returned from `mysql_init()`.

Return Values

A string naming the encryption cipher used for the connection, or `NULL` if the connection is not encrypted.

28.7.7.35 mysql_hex_string()

```
unsigned long mysql_hex_string(char *to, const char *from, unsigned long
length)
```

Description

This function creates a legal SQL string for use in an SQL statement. See Section 9.1.1, “String Literals”.

The string in the `from` argument is encoded in hexadecimal format, with each character encoded as two hexadecimal digits. The result is placed in the `to` argument, followed by a terminating null byte.

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. When `mysql_hex_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null byte.

The return value can be placed into an SQL statement using either `X'value'` or `0xvalue` format. However, the return value does not include the `X'...'` or `0x`. The caller must supply whichever of those is desired.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"X'");
end += mysql_hex_string(end,"What is this",12);
end = strmov(end,"',X'");
end += mysql_hex_string(end,"binary data: \0\r\n",16);
end = strmov(end,"')");

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
```

```
}
```

The `strmov()` function used in the example is included in the `libmysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the encoded string that is placed into `to`, not including the terminating null character.

Errors

None.

28.7.7.36 mysql_info()

```
const char *mysql_info(MYSQL *mysql)
```

Description

Retrieves a string providing information about the most recently executed statement, but only for the statements listed here. For other statements, `mysql_info()` returns `NULL`. The format of the string varies depending on the type of statement, as described here. The numbers are illustrative only; the string contains values appropriate for the statement.

- `INSERT INTO ... SELECT ...`

  String format: `Records: 100 Duplicates: 0 Warnings: 0`

- `INSERT INTO ... VALUES (...),(...),(...)...`

  String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `LOAD DATA`

  String format: `Records: 1 Deleted: 0 Skipped: 0 Warnings: 0`

- `ALTER TABLE`

  String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `UPDATE`

  String format: `Rows matched: 40 Changed: 40 Warnings: 0`

`mysql_info()` returns a non-`NULL` value for `INSERT ... VALUES` only for the multiple-row form of the statement (that is, only if multiple value lists are specified).

Return Values

A character string representing additional information about the most recently executed statement. `NULL` if no information is available for the statement.

Errors

None.

28.7.7.37 mysql_init()

```
MYSQL *mysql_init(MYSQL *mysql)
```

Description

Allocates or initializes a `MYSQL` object suitable for `mysql_real_connect()`. If `mysql` is a `NULL` pointer, the function allocates, initializes, and returns a new object. Otherwise, the object is initialized and the address of the object is returned. If `mysql_init()` allocates a new object, it is freed when `mysql_close()` is called to close the connection.

In a nonmultithreaded environment, `mysql_init()` invokes `mysql_library_init()` automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multithreaded environment, and thus neither is `mysql_init()`. Before calling `mysql_init()`, either call `mysql_library_init()` prior to spawning any threads, or use a mutex to protect the `mysql_library_init()` call. This should be done prior to any other client library call.

Return Values

An initialized `MYSQL*` handler. `NULL` if there was insufficient memory to allocate a new object.

Errors

In case of insufficient memory, `NULL` is returned.

28.7.7.38 mysql_insert_id()

```
uint64_t mysql_insert_id(MYSQL *mysql)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the previous `INSERT` or `UPDATE` statement. Use this function after you have performed an `INSERT` statement into a table that contains an `AUTO_INCREMENT` field, or have used `INSERT` or `UPDATE` to set a column value with `LAST_INSERT_ID(expr)`.

The return value of `mysql_insert_id()` is always zero unless explicitly updated under one of the following conditions:

- `INSERT` statements that store a value into an `AUTO_INCREMENT` column. This is true whether the value is automatically generated by storing the special values `NULL` or `0` into the column, or is an explicit nonspecial value.
- In the case of a multiple-row `INSERT` statement, `mysql_insert_id()` returns the first automatically generated `AUTO_INCREMENT` value that was successfully inserted.

  If no rows are successfully inserted, `mysql_insert_id()` returns 0.
- If an `INSERT ... SELECT` statement is executed, and no automatically generated value is successfully inserted, `mysql_insert_id()` returns the ID of the last inserted row.
- If an `INSERT ... SELECT` statement uses `LAST_INSERT_ID(expr)`, `mysql_insert_id()` returns *`expr`*.
- `INSERT` statements that generate an `AUTO_INCREMENT` value by inserting `LAST_INSERT_ID(expr)` into any column or by updating any column to `LAST_INSERT_ID(expr)`.
- If the previous statement returned an error, the value of `mysql_insert_id()` is undefined.

The return value of `mysql_insert_id()` can be simplified to the following sequence:

1. If there is an `AUTO_INCREMENT` column, and an automatically generated value was successfully inserted, return the first such value.
2. If `LAST_INSERT_ID(expr)` occurred in the statement, return *`expr`*, even if there was an `AUTO_INCREMENT` column in the affected table.
3. The return value varies depending on the statement used. When called after an `INSERT` statement:
   - If there is an `AUTO_INCREMENT` column in the table, and there were some explicit values for this column that were successfully inserted into the table, return the last of the explicit values.

   When called after an `INSERT ... ON DUPLICATE KEY UPDATE` statement:

   - If there is an `AUTO_INCREMENT` column in the table and there were some explicit successfully inserted values or some updated values, return the last of the inserted or updated values.

`mysql_insert_id()` returns `0` if the previous statement does not use an `AUTO_INCREMENT` value. If you need to save the value for later, be sure to call `mysql_insert_id()` immediately after the statement that generates the value.

The value of `mysql_insert_id()` is affected only by statements issued within the current client connection. It is not affected by statements issued by other clients.

The `LAST_INSERT_ID()` SQL function will contain the value of the first automatically generated value that was successfully inserted. `LAST_INSERT_ID()` is not reset between statements because the value of that function is maintained in the server. Another difference from `mysql_insert_id()` is that `LAST_INSERT_ID()` is not updated if you set an `AUTO_INCREMENT` column to a specific nonspecial value. See Section 12.15, “Information Functions”.

`mysql_insert_id()` returns `0` following a `CALL` statement for a stored procedure that generates an `AUTO_INCREMENT` value because in this case `mysql_insert_id()` applies to `CALL` and not the statement within the procedure. Within the procedure, you can use `LAST_INSERT_ID()` at the SQL level to obtain the `AUTO_INCREMENT` value.

The reason for the differences between `LAST_INSERT_ID()` and `mysql_insert_id()` is that `LAST_INSERT_ID()` is made easy to use in scripts while `mysql_insert_id()` tries to provide more exact information about what happens to the `AUTO_INCREMENT` column.

Note

The OK packet used in the client/server protocol holds information such as is used for session state tracking. When clients read the OK packet to know whether there is a session state change, this resets values such as the last insert ID and the number of affected rows. Such changes cause `mysql_insert_id()` to return 0 after execution of commands including but not necessarily limited to `COM_PING`, `COM_REFRESH`, and `COM_INIT_DB`.

Return Values

Described in the preceding discussion.

Errors

- `ER_AUTO_INCREMENT_CONFLICT`

A user-specified AUTO_INCREMENT value in a multi INSERT statement falls within the range between the current AUTO_INCREMENT value and the sum of the current and number of rows affected values.

28.7.7.39 mysql_kill()

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Description

Note

mysql_kill() is deprecated and will be removed in a future version of MySQL. Instead, use mysql_query() to execute a KILL statement.

Asks the server to kill the thread specified by pid.

This function is deprecated. Use mysql_query() to issue an SQL KILL statement instead.

mysql_kill() cannot handle values larger than 32 bits, but to guard against killing the wrong thread returns an error in these cases:

- If given an ID larger than 32 bits, mysql_kill() returns a CR_INVALID_CONN_HANDLE error.
- After the server's internal thread ID counter reaches a value larger than 32 bits, it returns an ER_DATA_OUT_OF_RANGE error for any mysql_kill() invocation and mysql_kill() fails.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_INVALID_CONN_HANDLE

  The pid was larger than 32 bits.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

- ER_DATA_OUT_OF_RANGE

  The server's internal thread ID counter has reached a value larger than 32 bits, at which point it rejects all mysql_kill() invocations.

28.7.7.40 mysql_library_end()

```
void mysql_library_end(void)
```

Description

This function finalizes the MySQL client library. Call it when you are done using the library (for example, after disconnecting from the server).

Note

To avoid memory leaks after the application is done using the library (for example, after closing the connection to the server), be sure to call `mysql_library_end()` explicitly. This enables memory managment to be performed to clean up and free resources used by the library.

For usage information, see Section 28.7.6, “C API Function Overview”, and Section 28.7.7.41, “mysql_library_init()”.

28.7.7.41 mysql_library_init()

```
int mysql_library_init(int argc, char **argv, char **groups)
```

Description

Call this function to initialize the MySQL client library before you call any other MySQL function.

Note

To avoid memory leaks after the application is done using the library (for example, after closing the connection to the server), be sure to call `mysql_library_end()` explicitly. This enables memory managment to be performed to clean up and free resources used by the library. See Section 28.7.7.40, “mysql_library_end()”.

In a nonmultithreaded environment, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary. However, `mysql_library_init()` is not thread-safe in a multithreaded environment, and thus neither is `mysql_init()`, which calls `mysql_library_init()`. You must either call `mysql_library_init()` prior to spawning any threads, or else use a mutex to protect the call, whether you invoke `mysql_library_init()` or indirectly through `mysql_init()`. Do this prior to any other client library call.

The `argc`, `argv`, and `groups` arguments are unused. In older MySQL versions, they were used for applications linked against the embedded server, which is no longer supported. The call now should be written as `mysql_library_init(0, NULL, NULL)`.

```
#include <mysql.h>
#include <stdlib.h>

int main(void) {
  if (mysql_library_init(0, NULL, NULL)) {
    fprintf(stderr, "could not initialize MySQL client library\n");
    exit(1);
  }

  /* Use any MySQL API functions here */

  mysql_library_end();
```

```
  return EXIT_SUCCESS;
}
```

Return Values

Zero for success. Nonzero if an error occurred.

28.7.7.42 mysql_list_dbs()

```
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of database names on the server that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters `%` or `_`, or may be a `NULL` pointer to match all databases. Calling `mysql_list_dbs()` is similar to executing the query `SHOW DATABASES [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.43 mysql_list_fields()

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char *wild)
```

Description

Note

`mysql_list_fields()` is deprecated and will be removed in a future version of MySQL. Instead, use `mysql_query()` to execute a `SHOW COLUMNS` statement.

Returns an empty result set for which the metadata provides information about the columns in the given table that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters `%` or `_`, or may be a `NULL` pointer to match all fields. Calling `mysql_list_fields()` is similar to executing the query `SHOW COLUMNS FROM tbl_name [LIKE wild]`.

It is preferable to use `SHOW COLUMNS FROM tbl_name` instead of `mysql_list_fields()`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

```
int i;
MYSQL_RES *tbl_cols = mysql_list_fields(mysql, "mytbl", "f%");

unsigned int field_cnt = mysql_num_fields(tbl_cols);
printf("Number of columns: %d\n", field_cnt);

for (i=0; i < field_cnt; ++i)
{
  /* col describes i-th column of the table */
  MYSQL_FIELD *col = mysql_fetch_field_direct(tbl_cols, i);
  printf ("Column %d: %s\n", i, col->name);
}
mysql_free_result(tbl_cols);
```

28.7.7.44 mysql_list_processes()

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

Description

Note

`mysql_list_processes()` is deprecated and will be removed in a future version of MySQL. Instead, use `mysql_query()` to execute a `SHOW PROCESSLIST` statement.

Returns a result set describing the current server threads. This is the same kind of information as that reported by `mysqladmin processlist` or a `SHOW PROCESSLIST` query.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.45 mysql_list_tables()

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of table names in the current database that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters `%` or `_`, or may be a `NULL` pointer to match all tables. Calling `mysql_list_tables()` is similar to executing the query `SHOW TABLES [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

28.7.7.46 mysql_more_results()

```
bool mysql_more_results(MYSQL *mysql)
```

Description

This function is used when you execute multiple statements specified as a single statement string, or when you execute `CALL` statements, which can return multiple result sets.

`mysql_more_results()` true if more results exist from the currently executed statement, in which case the application must call `mysql_next_result()` to fetch the results.

Return Values

`TRUE` (1) if more results exist. `FALSE` (0) if no more results exist.

In most cases, you can call `mysql_next_result()` instead to test whether more results exist and initiate retrieval if so.

See Section 28.7.23, “C API Multiple Statement Execution Support”, and Section 28.7.7.47, “mysql_next_result()”.

Errors

None.

28.7.7.47 mysql_next_result()

```
int mysql_next_result(MYSQL *mysql)
```

Description

Note

`mysql_next_result()` is a synchronous function. Its asynchronous counterpart is `mysql_next_result_nonblocking()`, for use by applications that require asynchronous communication with the server. See Section 28.7.12, “C API Asynchronous Interface”.

`mysql_next_result()` is used when you execute multiple statements specified as a single statement string, or when you use `CALL` statements to execute stored procedures, which can return multiple result sets.

`mysql_next_result()` reads the next statement result and returns a status to indicate whether more results exist. If `mysql_next_result()` returns an error, there are no more results.

Before each call to `mysql_next_result()`, you must call `mysql_free_result()` for the current statement if it is a statement that returned a result set (rather than just a result status).

After calling `mysql_next_result()` the state of the connection is as if you had called `mysql_real_query()` or `mysql_query()` for the next statement. This means that you can call `mysql_store_result()`, `mysql_warning_count()`, `mysql_affected_rows()`, and so forth.

If your program uses `CALL` statements to execute stored procedures, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any

result sets that might be returned by statements executed within the procedure. Because `CALL` can return multiple results, process them using a loop that calls `mysql_next_result()` to determine whether there are more results.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). `CLIENT_MULTI_RESULTS` is enabled by default.

It is also possible to test whether there are more results by calling `mysql_more_results()`. However, this function does not change the connection state, so if it returns true, you must still call `mysql_next_result()` to advance to the next result.

For an example that shows how to use `mysql_next_result()`, see Section 28.7.23, "C API Multiple Statement Execution Support".

Return Values

Return Value	Description
0	Successful and there are more results
-1	Successful and there are no more results
>0	An error occurred

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order. For example, if you did not call `mysql_use_result()` for a previous result set.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.48 mysql_num_fields()

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

To pass a `MYSQL*` argument instead, use `unsigned int mysql_field_count(MYSQL *mysql)`.

Description

Returns the number of columns in a result set.

You can get the number of columns either from a pointer to a result set or to a connection handler. You would use the connection handler if `mysql_store_result()` or `mysql_use_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a nonempty result. This enables the

client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(&mysql))
        {
           fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
```

An alternative (if you know that your query should have returned a result set) is to replace the `mysql_errno(&mysql)` call with a check whether `mysql_field_count(&mysql)` returns 0. This happens only if something went wrong.

28.7.7.49 mysql_num_rows()

```
uint64_t mysql_num_rows(MYSQL_RES *result)
```

Description

Returns the number of rows in the result set.

The use of `mysql_num_rows()` depends on whether you use `mysql_store_result()` or `mysql_use_result()` to return the result set. If you use `mysql_store_result()`,

`mysql_num_rows()` may be called immediately. If you use `mysql_use_result()`, `mysql_num_rows()` does not return the correct value until all the rows in the result set have been retrieved.

`mysql_num_rows()` is intended for use with statements that return a result set, such as `SELECT`. For statements such as `INSERT`, `UPDATE`, or `DELETE`, the number of affected rows can be obtained with `mysql_affected_rows()`.

Return Values

The number of rows in the result set.

Errors

None.

28.7.7.50 mysql_options()

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const void *arg)
```

Description

Can be used to set extra connect options and affect behavior for a connection. This function may be called multiple times to set several options. To retrieve option values, use `mysql_get_option()`.

Call `mysql_options()` after `mysql_init()` and before `mysql_connect()` or `mysql_real_connect()`.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, specify a pointer to the value of the integer as the `arg` argument.

Options for information such as SSL certificate and key files are used to establish an encrypted connection if such connections are available, but do not enforce any requirement that the connection obtained be encrypted. To require an encrypted connection, use the technique described in Section 28.7.22, “C API Encrypted Connection Support”.

The following list describes the possible options, their effect, and how `arg` is used for each option. For option descriptions that indicate `arg` is unused, its value is irrelevant; it is conventional to pass 0.

- `MYSQL_DEFAULT_AUTH` (argument type: `char *`)

  The name of the authentication plugin to use.

- `MYSQL_ENABLE_CLEARTEXT_PLUGIN` (argument type: `bool *`)

  Enable the `mysql_clear_password` cleartext authentication plugin. See Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”.

- `MYSQL_INIT_COMMAND` (argument type: `char *`)

  SQL statement to execute when connecting to the MySQL server. Automatically re-executed if reconnection occurs.

- `MYSQL_OPT_BIND` (argument: `char *`)

  The network interface from which to connect to the server. This is used when the client host has multiple network interfaces. The argument is a host name or IP address (specified as a string).

- `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` (argument type: `bool *`)

  Indicate whether the client can handle expired passwords. See Section 6.2.16, “Server Handling of Expired Passwords”.

- `MYSQL_OPT_COMPRESS` (argument: not used)

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

  As of MySQL 8.0.18, `MYSQL_OPT_COMPRESS` becomes a legacy option, due to the introduction of the `MYSQL_OPT_COMPRESSION_ALGORITHMS` option for more control over connection compression (see Connection Compression Configuration). The meaning of `MYSQL_OPT_COMPRESS` depends on whether `MYSQL_OPT_COMPRESSION_ALGORITHMS` is specified:

  - When `MYSQL_OPT_COMPRESSION_ALGORITHMS` is not specified, enabling `MYSQL_OPT_COMPRESS` is equivalent to specifying a client-side algorithm set of `zlib,uncompressed`.

  - When `MYSQL_OPT_COMPRESSION_ALGORITHMS` is specified, enabling `MYSQL_OPT_COMPRESS` is equivalent to specifying an algorithm set of `zlib` and the full client-side algorithm set is the union of `zlib` plus the algorithms specified by `MYSQL_OPT_COMPRESSION_ALGORITHMS`. For example, with `MYSQL_OPT_COMPRESS` enabled and `MYSQL_OPT_COMPRESSION_ALGORITHMS` set to `zlib,zstd`, the permitted-algorithm set is `zlib` plus `zlib,zstd`; that is, `zlib,zstd`. With `MYSQL_OPT_COMPRESS` enabled and `MYSQL_OPT_COMPRESSION_ALGORITHMS` set to `zstd,uncompressed`, the permitted-algorithm set is `zlib` plus `zstd,uncompressed`; that is, `zlib,zstd,uncompressed`.

  As of MySQL 8.0.18, `MYSQL_OPT_COMPRESS` is deprecated. It will be removed in a future MySQL version. See Legacy Connection Compression Configuration.

- `MYSQL_OPT_COMPRESSION_ALGORITHMS` (argument type: `const char *`)

  The permitted compression algorithms for connections to the server. The available algorithms are the same as for the `protocol_compression_algorithms` system variable. If this option is not specified, the default value is `uncompressed`.

  For more information, see Section 4.2.6, “Connection Compression Control”.

  This option was added in MySQL 8.0.18.

- `MYSQL_OPT_CONNECT_ATTR_DELETE` (argument type: `char *`)

  Given a key name, this option deletes a key-value pair from the current set of connection attributes to pass to the server at connect time. The argument is a pointer to a null-terminated string naming the key. Comparison of the key name with existing keys is case-sensitive.

  See also the description for the `MYSQL_OPT_CONNECT_ATTR_RESET` option, as well as the description for the `MYSQL_OPT_CONNECT_ATTR_ADD` option in the description of the `mysql_options4()` function. That function description also includes a usage example.

  The Performance Schema exposes connection attributes through the `session_connect_attrs` and `session_account_connect_attrs` tables. See Section 26.12.9, “Performance Schema Connection Attribute Tables”.

- `MYSQL_OPT_CONNECT_ATTR_RESET` (argument not used)

  This option resets (clears) the current set of connection attributes to pass to the server at connect time.

See also the description for the MYSQL_OPT_CONNECT_ATTR_DELETE option, as well as the description for the MYSQL_OPT_CONNECT_ATTR_ADD option in the description of the mysql_options4() function. That function description also includes a usage example.

The Performance Schema exposes connection attributes through the session_connect_attrs and session_account_connect_attrs tables. See Section 26.12.9, "Performance Schema Connection Attribute Tables".

- MYSQL_OPT_CONNECT_TIMEOUT (argument type: unsigned int *)

  The connect timeout in seconds.

- MYSQL_OPT_GET_SERVER_PUBLIC_KEY (argument type: bool *)

  Enables the client to request from the server the public key required for RSA key pair-based password exchange. This option applies to clients that authenticate with the caching_sha2_password authentication plugin. For that plugin, the server does not send the public key unless requested. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If MYSQL_SERVER_PUBLIC_KEY is given and specifies a valid public key file, it takes precedence over MYSQL_OPT_GET_SERVER_PUBLIC_KEY.

  For information about the caching_sha2_password plugin, see Section 6.4.1.2, "Caching SHA-2 Pluggable Authentication".

- MYSQL_OPT_LOCAL_INFILE (argument type: optional pointer to unsigned int)

  This option affects client-side LOCAL capability for LOAD DATA operations. By default, LOCAL capability is determined by the default compiled into the MySQL client library (see Section 13.2.7, "LOAD DATA Syntax"). To control this capability explicitly, invoke mysql_options() to set the MYSQL_OPT_LOCAL_INFILE option:

  - LOCAL is disabled if the pointer points to an unsigned int that has a zero value.
  - LOCAL is enabled if no pointer is given or if the pointer points to an unsigned int that has a nonzero value.

  Successful use of a LOCAL load operation by a client also requires that the server permits it.

- MYSQL_OPT_MAX_ALLOWED_PACKET (argument: unsigned long *)

  This option sets the max_allowed_packet system variable. If the mysql argument is non-NULL, the call sets the session system variable value for that session. If mysql is NULL, the call sets the global system variable value.

- MYSQL_OPT_NAMED_PIPE (argument: not used)

  Use a named pipe to connect to the MySQL server on Windows, if the server permits named-pipe connections.

- MYSQL_OPT_NET_BUFFER_LENGTH (argument: unsigned long *)

  This option sets the net_buffer_length system variable. If the mysql argument is non-NULL, the call sets the session system variable value for that session. If mysql is NULL, the call sets the global system variable value.

- `MYSQL_OPT_OPTIONAL_RESULTSET_METADATA` (argument type: `bool *`)

  This flag makes result set metadata optional. It is an alternative way of setting the `CLIENT_OPTIONAL_RESULTSET_METADATA` connection flag for the `mysql_real_connect()` function. For details about managing result set metadata transfer, see Section 28.7.27, “C API Optional Result Set Metadata”.

- `MYSQL_OPT_PROTOCOL` (argument type: `unsigned int *`)

  Type of protocol to use. Specify one of the enum values of `mysql_protocol_type` defined in `mysql.h`.

- `MYSQL_OPT_READ_TIMEOUT` (argument type: `unsigned int *`)

  The timeout in seconds for each attempt to read from the server. There are retries if necessary, so the total effective timeout value is three times the option value. You can set the value so that a lost connection can be detected earlier than the TCP/IP `Close_Wait_Timeout` value of 10 minutes.

- `MYSQL_OPT_RECONNECT` (argument type: `bool *`)

  Enable or disable automatic reconnection to the server if the connection is found to have been lost. Reconnect is off by default; this option provides a way to set reconnection behavior explicitly. See Section 28.7.28, “C API Automatic Reconnection Control”.

- `MYSQL_OPT_RETRY_COUNT` (argument type: `unsigned int *`)

  The retry count for I/O-related system calls that are interrupted while connecting to the server or communicating with it. If this option is not specified, the default value is 1 (1 retry if the initial call is interrupted for 2 tries total).

  This option can be used only by clients that link against a C client library compiled with NDB Cluster support.

- `MYSQL_OPT_SSL_CA` (argument type: `char *`)

  The path name of the Certificate Authority (CA) certificate file. This option, if used, must specify the same certificate used by the server.

- `MYSQL_OPT_SSL_CAPATH` (argument type: `char *`)

  The path name of the directory that contains trusted SSL CA certificate files.

- `MYSQL_OPT_SSL_CERT` (argument type: `char *`)

  The path name of the client public key certificate file.

- `MYSQL_OPT_SSL_CIPHER` (argument type: `char *`)

  The list of permissible ciphers for SSL encryption.

- `MYSQL_OPT_SSL_CRL` (argument type: `char *`)

  The path name of the file containing certificate revocation lists.

- `MYSQL_OPT_SSL_CRLPATH` (argument type: `char *`)

  The path name of the directory that contains files containing certificate revocation lists.

- `MYSQL_OPT_SSL_FIPS_MODE` (argument type: `unsigned int *`)

Controls whether to enable FIPS mode on the client side. The `MYSQL_OPT_SSL_FIPS_MODE` option differs from other `MYSQL_OPT_SSL_xxx` options in that it is not used to establish encrypted connections, but rather to affect which cryptographic operations are permitted. See Section 6.5, “FIPS Support”.

Permitted option values are `SSL_FIPS_MODE_OFF`, `SSL_FIPS_MODE_ON`, and `SSL_FIPS_MODE_STRICT`.

Note

If the OpenSSL FIPS Object Module is not available, the only permitted value for `MYSQL_OPT_SSL_FIPS_MODE` is `SSL_FIPS_MODE_OFF`. In this case, setting `MYSQL_OPT_SSL_FIPS_MODE` to `SSL_FIPS_MODE_ON` or `SSL_FIPS_MODE_STRICT` causes the client to produce a warning at startup and to operate in non-FIPS mode.

- `MYSQL_OPT_SSL_KEY` (argument type: `char *`)

  The path name of the client private key file.

- `MYSQL_OPT_SSL_MODE` (argument type: `unsigned int *`)

  The security state to use for the connection to the server: `SSL_MODE_DISABLED`, `SSL_MODE_PREFERRED`, `SSL_MODE_REQUIRED`, `SSL_MODE_VERIFY_CA`, `SSL_MODE_VERIFY_IDENTITY`. If this option is not specified, the default is `SSL_MODE_PREFERRED`. These modes are the permitted values of the `mysql_ssl_mode` enumeration defined in `mysql.h`. For more information about the security states, see the description of `--ssl-mode` in Command Options for Encrypted Connections.

- `MYSQL_OPT_TLS_CIPHERSUITES` (argument type: `char *`)

  Which ciphersuites the client permits for encrypted connections that use TLSv1.3. The value is a list of one or more colon-separated ciphersuite names. The ciphersuites that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  This option was added in MySQL 8.0.16.

- `MYSQL_OPT_TLS_VERSION` (argument type: `char *`)

  Which protocols the client permits for encrypted connections. The value is a list of one or more comma-separated protocol versions. The protocols that can be named for this option depend on the SSL library used to compile MySQL. For details, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `MYSQL_OPT_USE_RESULT` (argument: not used)

  This option is unused.

- `MYSQL_OPT_WRITE_TIMEOUT` (argument type: `unsigned int *`)

  The timeout in seconds for each attempt to write to the server. There is a retry if necessary, so the total effective timeout value is two times the option value.

- `MYSQL_OPT_ZSTD_COMPRESSION_LEVEL` (argument type: `unsigned int *`)

  The compression level to use for connections to the server that use the `zstd` compression algorithm. The permitted levels are from 1 to 22, with larger values indicating increasing levels of compression. If

this option is not specified, the default `zstd` compression level is 3. The compression level setting has no effect on connections that do not use `zstd` compression.

For more information, see Section 4.2.6, “Connection Compression Control”.

This option was added in MySQL 8.0.18.

- `MYSQL_PLUGIN_DIR` (argument type: `char *`)

  The directory in which to look for client plugins.

- `MYSQL_READ_DEFAULT_FILE` (argument type: `char *`)

  Read options from the named option file instead of from `my.cnf`.

- `MYSQL_READ_DEFAULT_GROUP` (argument type: `char *`)

  Read options from the named group from `my.cnf` or the file specified with `MYSQL_READ_DEFAULT_FILE`.

- `MYSQL_REPORT_DATA_TRUNCATION` (argument type: `bool *`)

  Enable or disable reporting of data truncation errors for prepared statements using the `error` member of `MYSQL_BIND` structures. (Default: enabled.)

- `MYSQL_SERVER_PUBLIC_KEY` (argument type: `char *`)

  The path name of the file containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. The file must be in PEM format. This option applies to clients that authenticate with the `sha256_password` or `caching_sha2_password` authentication plugin. This option is ignored for accounts that do not authenticate with one of those plugins. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  If `MYSQL_SERVER_PUBLIC_KEY` is given and specifies a valid public key file, it takes precedence over `MYSQL_OPT_GET_SERVER_PUBLIC_KEY`.

  For information about the `sha256_password` and `caching_sha2_password` plugins, see Section 6.4.1.3, “SHA-256 Pluggable Authentication”, and Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

- `MYSQL_SET_CHARSET_DIR` (argument type: `char *`)

  The path name of the directory that contains character set definition files.

- `MYSQL_SET_CHARSET_NAME` (argument type: `char *`)

  The name of the character set to use as the default character set. The argument can be `MYSQL_AUTODETECT_CHARSET_NAME` to cause the character set to be autodetected based on the operating system setting (see Section 10.4, “Connection Character Sets and Collations”).

- `MYSQL_SHARED_MEMORY_BASE_NAME` (argument type: `char *`)

  The name of the shared-memory object for communication to the server on Windows, if the server supports shared-memory connections. Specify the same value as used for the `shared_memory_base_name` system variable. of the `mysqld` server you want to connect to.

The `client` group is always read if you use `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP`.

The specified group in the option file may contain the following options.

Option	Description
`character-sets-dir=dir_name`	The directory where character sets are installed.
`compress`	Use the compressed client/server protocol.
`connect-timeout=seconds`	The connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.
`database=db_name`	Connect to this database if no database was specified in the connect command.
`debug`	Debug options.
`default-character-set=charset_name`	The default character set to use.
`disable-local-infile`	Disable use of `LOAD DATA LOCAL`.
`enable-cleartext-plugin`	Enable the `mysql_clear_password` cleartext authentication plugin.
`host=host_name`	Default host name.
`init-command=stmt`	Statement to execute when connecting to MySQL server. Automatically re-executed if reconnection occurs.
`interactive-timeout=seconds`	Same as specifying `CLIENT_INTERACTIVE` to `mysql_real_connect()`. See Section 28.7.7.54, “mysql_real_connect()”.
`local-infile[={0|1}]`	If no argument or nonzero argument, enable use of `LOAD DATA LOCAL`; otherwise disable.
`max_allowed_packet=bytes`	Maximum size of packet that client can read from server.
`multi-queries`, `multi-results`	Enable multiple result sets from multiple-statement executions or stored procedures.
`multi-statements`	Enable the client to send multiple statements in a single string (separated by `;` characters).
`password=password`	Default password.
`pipe`	Use named pipes to connect to a MySQL server on Windows.
`port=port_num`	Default port number.
`protocol={TCP|SOCKET|PIPE|MEMORY}`	The protocol to use when connecting to the server.
`return-found-rows`	Tell `mysql_info()` to return found rows instead of updated rows when using `UPDATE`.
`shared-memory-base-name=name`	Shared-memory name to use to connect to server.
`socket={file_name|pipe_name}`	Default socket file.
`ssl-ca=file_name`	Certificate Authority file.
`ssl-capath=dir_name`	Certificate Authority directory.
`ssl-cert=file_name`	Certificate file.
`ssl-cipher=cipher_list`	Permissible SSL ciphers.
`ssl-key=file_name`	Key file.

Option	Description
`timeout=seconds`	Like `connect-timeout`.
`user`	Default user.

`timeout` has been replaced by `connect-timeout`, but `timeout` is still supported for backward compatibility.

For more information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Return Values

Zero for success. Nonzero if you specify an unknown option.

Example

The following `mysql_options()` calls request the use of compression in the client/server protocol, cause options to be read from the `[odbc]` group in option files, and disable transaction autocommit mode:

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
mysql_options(&mysql,MYSQL_INIT_COMMAND,"SET autocommit=0");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

28.7.7.51 mysql_options4()

```
int mysql_options4(MYSQL *mysql, enum mysql_option option, const void *arg1,
const void *arg2)
```

Description

`mysql_options4()` is similar to `mysql_options()` but has an extra fourth argument so that two values can be passed for the option specified in the second argument.

The following list describes the permitted options, their effect, and how `arg1` and `arg2` are used.

- `MYSQL_OPT_CONNECT_ATTR_ADD` (argument types: `char *, char *`)

  This option adds an attribute key-value pair to the current set of connection attributes to pass to the server at connect time. Both arguments are pointers to null-terminated strings. The first and second strings indicate the key and value, respectively. If the key is empty or already exists in the current set of connection attributes, an error occurs. Comparison of the key name with existing keys is case-sensitive.

  Key names that begin with an underscore (_) are reserved for internal use and should not be created by application programs. This convention permits new attributes to be introduced by MySQL without colliding with application attributes.

  `mysql_options4()` imposes a limit of 64KB on the aggregate size of connection attribute data it will accept. For calls that cause this limit to be exceeded, a `CR_INVALID_PARAMETER_NO` error occurs. Attribute size-limit checks also occur on the server side. For details, see Section 26.12.9, “Performance Schema Connection Attribute Tables”, which also describes how the

Performance Schema exposes connection attributes through the `session_connect_attrs` and `session_account_connect_attrs` tables.

See also the descriptions for the `MYSQL_OPT_CONNECT_ATTR_RESET` and `MYSQL_OPT_CONNECT_ATTR_DELETE` options in the description of the `mysql_options()` function.

Return Values

Zero for success. Nonzero if you specify an unknown option.

Errors

- `CR_DUPLICATE_CONNECTION_ATTR`

  A duplicate attribute name was specified.

- `CR_INVALID_PARAMETER_NO`

  A key name was empty or the amount of key-value connection attribute data exceeds 64KB limit.

- `CR_OUT_OF_MEMORY`

  Out of memory.

Example

This example demonstrates the calls that specify connection attributes:

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_CONNECT_ATTR_RESET, 0);
mysql_options4(&mysql,MYSQL_OPT_CONNECT_ATTR_ADD, "key1", "value1");
mysql_options4(&mysql,MYSQL_OPT_CONNECT_ATTR_ADD, "key2", "value2");
mysql_options4(&mysql,MYSQL_OPT_CONNECT_ATTR_ADD, "key3", "value3");
mysql_options(&mysql,MYSQL_OPT_CONNECT_ATTR_DELETE, "key1");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
        mysql_error(&mysql));
}
```

28.7.7.52 mysql_ping()

```
int mysql_ping(MYSQL *mysql)
```

Description

Checks whether the connection to the server is working. If the connection has gone down and auto-reconnect is enabled an attempt to reconnect is made. If the connection is down and auto-reconnect is disabled, `mysql_ping()` returns an error.

Auto-reconnect is disabled by default. To enable it, call `mysql_options()` with the `MYSQL_OPT_RECONNECT` option. For details, see Section 28.7.7.50, “mysql_options()”.

`mysql_ping()` can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

If `mysql_ping()`) does cause a reconnect, there is no explicit indication of it. To determine whether a reconnect occurs, call `mysql_thread_id()` to get the original connection identifier before calling `mysql_ping()`, then call `mysql_thread_id()` again to see whether the identifier has changed.

If reconnect occurs, some characteristics of the connection will have been reset. For details about these characteristics, see Section 28.7.28, “C API Automatic Reconnection Control”.

Return Values

Zero if the connection to the server is active. Nonzero if an error occurred. A nonzero return does not indicate whether the MySQL server itself is down; the connection might be broken for other reasons such as network problems.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.53 mysql_query()

```
int mysql_query(MYSQL *mysql, const char *stmt_str)
```

Description

Executes the SQL statement pointed to by the null-terminated string `stmt_str`. Normally, the string must consist of a single SQL statement without a terminating semicolon (`;`) or `\g`. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Section 28.7.23, “C API Multiple Statement Execution Support”.

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the `\0` character, which `mysql_query()` interprets as the end of the statement string.)

If you want to know whether the statement returns a result set, you can use `mysql_field_count()` to check for this. See Section 28.7.7.22, “mysql_field_count()”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

An unknown error occurred.

28.7.7.54 mysql_real_connect()

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
const char *passwd, const char *db, unsigned int port, const char *unix_socket,
unsigned long client_flag)
```

Description

Note

`mysql_real_connect()` is a synchronous function. Its asynchronous counterpart is `mysql_real_connect_nonblocking()`, for use by applications that require asynchronous communication with the server. See Section 28.7.12, "C API Asynchronous Interface".

`mysql_real_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_real_connect()` must complete successfully before you can execute any other API functions that require a valid `MYSQL` connection handler structure.

The parameters are specified as follows:

- For the first parameter, specify the address of an existing `MYSQL` structure. Before calling `mysql_real_connect()`, call `mysql_init()` to initialize the `MYSQL` structure. You can change a lot of connect options with the `mysql_options()` call. See Section 28.7.7.50, "mysql_options()".
- The value of `host` may be either a host name or an IP address. The client attempts to connect as follows:
  - If `host` is `NULL` or the string `"localhost"`, a connection to the local host is assumed:
    - On Windows, the client connects using a shared-memory connection, if the server has shared-memory connections enabled.
    - On Unix, the client connects using a Unix socket file. The `unix_socket` parameter or the `MYSQL_UNIX_PORT` environment variable may be used to specify the socket name.
  - On Windows, if `host` is `"."`, or TCP/IP is not enabled and no `unix_socket` is specified or the host is empty, the client connects using a named pipe, if the server has named-pipe connections enabled. If named-pipe connections are not enabled, an error occurs.
  - Otherwise, TCP/IP is used.

  You can also influence the type of connection to use with the `MYSQL_OPT_PROTOCOL` or `MYSQL_OPT_NAMED_PIPE` options to `mysql_options()`. The type of connection must be supported by the server.
- The `user` parameter contains the user's MySQL login ID. If `user` is `NULL` or the empty string `""`, the current user is assumed. Under Unix, this is the current login name. Under Windows ODBC, the current user name must be specified explicitly. See the Connector/ODBC section of Chapter 28, *Connectors and APIs*.
- The `passwd` parameter contains the password for `user`. If `passwd` is `NULL`, only entries in the `user` table for the user that have a blank (empty) password field are checked for a match. This enables the database administrator to set up the MySQL privilege system in such a way that users get different privileges depending on whether they have specified a password.

Note

Do not attempt to encrypt the password before calling `mysql_real_connect()`; password encryption is handled automatically by the client API.

- The `user` and `passwd` parameters use whatever character set has been configured for the `MYSQL` object. By default, this is `utf8mb4`, but can be changed by calling `mysql_options(mysql, MYSQL_SET_CHARSET_NAME, "charset_name")` prior to connecting.
- `db` is the database name. If `db` is not `NULL`, the connection sets the default database to this value.
- If `port` is not 0, the value is used as the port number for the TCP/IP connection. Note that the `host` parameter determines the type of the connection.
- If `unix_socket` is not `NULL`, the string specifies the socket or named pipe to use. Note that the `host` parameter determines the type of the connection.
- The value of `client_flag` is usually 0, but can be set to a combination of the following flags to enable certain features:
  - `CAN_HANDLE_EXPIRED_PASSWORDS`: The client can handle expired passwords. For more information, see Section 6.2.16, “Server Handling of Expired Passwords”.
  - `CLIENT_COMPRESS`: Use compression in the client/server protocol.
  - `CLIENT_FOUND_ROWS`: Return the number of found (matched) rows, not the number of changed rows.
  - `CLIENT_IGNORE_SIGPIPE`: Prevents the client library from installing a `SIGPIPE` signal handler. This can be used to avoid conflicts with a handler that the application has already installed.
  - `CLIENT_IGNORE_SPACE`: Permit spaces after function names. Makes all functions names reserved words.
  - `CLIENT_INTERACTIVE`: Permit `interactive_timeout` seconds of inactivity (rather than `wait_timeout` seconds) before closing the connection. The client's session `wait_timeout` variable is set to the value of the session `interactive_timeout` variable.
  - `CLIENT_LOCAL_FILES`: Enable `LOAD DATA LOCAL` handling.
  - `CLIENT_MULTI_RESULTS`: Tell the server that the client can handle multiple result sets from multiple-statement executions or stored procedures. This flag is automatically enabled if `CLIENT_MULTI_STATEMENTS` is enabled. See the note following this table for more information about this flag.
  - `CLIENT_MULTI_STATEMENTS`: Tell the server that the client may send multiple statements in a single string (separated by `;` characters). If this flag is not set, multiple-statement execution is disabled. See the note following this table for more information about this flag.
  - `CLIENT_NO_SCHEMA` Do not permit *`db_name.tbl_name.col_name`* syntax. This is for ODBC. It causes the parser to generate an error if you use that syntax, which is useful for trapping bugs in some ODBC programs.
  - `CLIENT_ODBC`: Unused.
  - `CLIENT_OPTIONAL_RESULTSET_METADATA`: This flag makes result set metadata optional. Suppression of metadata transfer can improve performance, particularly for sessions that execute

many queries that return few rows each. For details about managing result set metadata transfer, see Section 28.7.27, “C API Optional Result Set Metadata”.

- `CLIENT_SSL`: Use SSL (encrypted protocol). Do not set this option within an application program; it is set internally in the client library. Instead, use `mysql_options()` or `mysql_ssl_set()` before calling `mysql_real_connect()`.
- `CLIENT_REMEMBER_OPTIONS` Remember options specified by calls to `mysql_options()`. Without this option, if `mysql_real_connect()` fails, you must repeat the `mysql_options()` calls before trying to connect again. With this option, the `mysql_options()` calls need not be repeated.

If your program uses `CALL` statements to execute stored procedures, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. Because `CALL` can return multiple results, process them using a loop that calls `mysql_next_result()` to determine whether there are more results.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). `CLIENT_MULTI_RESULTS` is enabled by default.

If you enable `CLIENT_MULTI_STATEMENTS` or `CLIENT_MULTI_RESULTS`, process the result for every call to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see Section 28.7.23, “C API Multiple Statement Execution Support”.

For some parameters, it is possible to have the value taken from an option file rather than from an explicit value in the `mysql_real_connect()` call. To do this, call `mysql_options()` with the `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP` option before calling `mysql_real_connect()`. Then, in the `mysql_real_connect()` call, specify the “no-value” value for each parameter to be read from an option file:

- For `host`, specify a value of `NULL` or the empty string (`""`).
- For `user`, specify a value of `NULL` or the empty string.
- For `passwd`, specify a value of `NULL`. (For the password, a value of the empty string in the `mysql_real_connect()` call cannot be overridden in an option file, because the empty string indicates explicitly that the MySQL account must have an empty password.)
- For `db`, specify a value of `NULL` or the empty string.
- For `port`, specify a value of 0.
- For `unix_socket`, specify a value of `NULL`.

If no value is found in an option file for a parameter, its default value is used as indicated in the descriptions given earlier in this section.

Return Values

A `MYSQL*` connection handler if the connection was successful, `NULL` if the connection was unsuccessful. For a successful connection, the return value is the same as the value of the first parameter.

Errors

- `CR_CONN_HOST_ERROR`

Failed to connect to the MySQL server.

- `CR_CONNECTION_ERROR`

  Failed to connect to the local MySQL server.

- `CR_IPSOCK_ERROR`

  Failed to create an IP socket.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SOCKET_CREATE_ERROR`

  Failed to create a Unix socket.

- `CR_UNKNOWN_HOST`

  Failed to find the IP address for the host name.

- `CR_VERSION_ERROR`

  A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version.

- `CR_NAMEDPIPEOPEN_ERROR`

  Failed to create a named pipe on Windows.

- `CR_NAMEDPIPEWAIT_ERROR`

  Failed to wait for a named pipe on Windows.

- `CR_NAMEDPIPESETSTATE_ERROR`

  Failed to get a pipe handler on Windows.

- `CR_SERVER_LOST`

  If `connect_timeout` > 0 and it took longer than `connect_timeout` seconds to connect to the server or if the server died while executing the `init-command`.

- `CR_ALREADY_CONNECTED`

  The `MYSQL` connection handler is already connected.

Example

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"your_prog_name");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

By using `mysql_options()` the MySQL client library reads the `[client]` and `[your_prog_name]` sections in the `my.cnf` file which ensures that your program works, even if someone has set up MySQL in some nonstandard way.

Upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API older than 5.0.3, or `0` in newer versions. A value of `1` for this flag indicates that if a statement cannot be performed because of a lost connection, to try reconnecting to the server before giving up. You can use the `MYSQL_OPT_RECONNECT` option to `mysql_options()` to control reconnection behavior.

28.7.7.55 mysql_real_escape_string()

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const char
*from, unsigned long length)
```

Description

This function creates a legal SQL string for use in an SQL statement. See Section 9.1.1, “String Literals”.

Note

`mysql_real_escape_string()` fails and produces an `CR_INSECURE_API_ERR` error if the `NO_BACKSLASH_ESCAPES` SQL mode is enabled. In this case, the function cannot escape quote characters except by doubling them, and to do this properly, it must know more information about the quoting context than is available. Instead, use `mysql_real_escape_string_quote()`, which takes an extra argument for specifying the quoting context.

The `mysql` argument must be a valid, open connection because character escaping depends on the character set in use by the server.

The string in the `from` argument is encoded to produce an escaped SQL string, taking into account the current character set of the connection. The result is placed in the `to` argument, followed by a terminating null byte.

Characters encoded are `\`, `'`, `"`, `NUL` (ASCII 0), `\n`, `\r`, and Control+Z. Strictly speaking, MySQL requires only that backslash and the quote character used to quote the string in the query be escaped. `mysql_real_escape_string()` quotes the other characters to make them easier to read in log files. For comparison, see the quoting rules for literal strings and the `QUOTE()` SQL function in Section 9.1.1, “String Literals”, and Section 12.5, “String Functions and Operators”.

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. (In the worst case, each character may need to be encoded as using two bytes, and there must be room for the terminating null byte.) When `mysql_real_escape_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null byte.

If you must change the character set of the connection, use the `mysql_set_character_set()` function rather than executing a `SET NAMES` (or `SET CHARACTER SET`) statement. `mysql_set_character_set()` works like `SET NAMES` but also affects the character set used by `mysql_real_escape_string()`, which `SET NAMES` does not.

Example

The following example inserts two escaped strings into an `INSERT` statement, each within single quote characters:

```
char query[1000],*end;

end = my_stpcpy(query,"INSERT INTO test_table VALUES('");
end += mysql_real_escape_string(&mysql,end,"What is this",12);
end = my_stpcpy(end,"','");
end += mysql_real_escape_string(&mysql,end,"binary data: \0\r\n",16);
end = my_stpcpy(end,"')");

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

The `my_stpcpy()` function used in the example is included in the `libmysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the encoded string that is placed into the `to` argument, not including the terminating null byte, or -1 if an error occurs.

Because `mysql_real_escape_string()` returns an unsigned value, you can check for -1 by comparing the return value to `(unsigned long)-1` (or to `(unsigned long)~0`, which is equivalent).

Errors

- `CR_INSECURE_API_ERR`

  This error occurs if the `NO_BACKSLASH_ESCAPES` SQL mode is enabled because, in that case, `mysql_real_escape_string()` cannot be guaranteed to produce a properly encoded result. To avoid this error, use `mysql_real_escape_string_quote()` instead.

28.7.7.56 mysql_real_escape_string_quote()

```
unsigned long mysql_real_escape_string_quote(MYSQL *mysql, char *to, const char
*from, unsigned long length, char quote)
```

Description

This function creates a legal SQL string for use in an SQL statement. See Section 9.1.1, “String Literals”.

The `mysql` argument must be a valid, open connection because character escaping depends on the character set in use by the server.

The string in the `from` argument is encoded to produce an escaped SQL string, taking into account the current character set of the connection. The result is placed in the `to` argument, followed by a terminating null byte.

Characters encoded are `\`, `'`, `"`, `NUL` (ASCII 0), `\n`, `\r`, Control+Z, and `` ` ``. Strictly speaking, MySQL requires only that backslash and the quote character used to quote the string in the query be escaped. `mysql_real_escape_string_quote()` quotes the other characters to make them easier to read in log files. For comparison, see the quoting rules for literal strings and the `QUOTE()` SQL function in Section 9.1.1, “String Literals”, and Section 12.5, “String Functions and Operators”.

Note

If the `ANSI_QUOTES` SQL mode is enabled, `mysql_real_escape_string_quote()` cannot be used to escape double quote characters for use within double-quoted identifiers. (The function cannot tell whether the mode is enabled to determine the proper escaping character.)

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. (In the worst case, each character may need to be encoded as using two bytes, and there must be room for the terminating null byte.) When `mysql_real_escape_string_quote()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null byte.

The `quote` argument indicates the context in which the escaped string is to be placed. Suppose that you intend to escape the `from` argument and insert the escaped string (designated here by *`str`*) into one of the following statements:

```
1) SELECT * FROM table WHERE name = 'str'
2) SELECT * FROM table WHERE name = "str"
3) SELECT * FROM `str` WHERE id = 103
```

To perform escaping properly for each statement, call `mysql_real_escape_string_quote()` as follows, where the final argument indicates the quoting context:

```
1) len = mysql_real_escape_string_quote(&mysql,to,from,from_len,'\'');
2) len = mysql_real_escape_string_quote(&mysql,to,from,from_len,'"');
3) len = mysql_real_escape_string_quote(&mysql,to,from,from_len,'`');
```

If you must change the character set of the connection, use the `mysql_set_character_set()` function rather than executing a `SET NAMES` (or `SET CHARACTER SET`) statement. `mysql_set_character_set()` works like `SET NAMES` but also affects the character set used by `mysql_real_escape_string_quote()`, which `SET NAMES` does not.

Example

The following example inserts two escaped strings into an `INSERT` statement, each within single quote characters:

```
char query[1000],*end;

end = my_stpcpy(query,"INSERT INTO test_table VALUES('");
end += mysql_real_escape_string_quote(&mysql,end,"What is this",12,'\'');
end = my_stpcpy(end,"','");
end += mysql_real_escape_string_quote(&mysql,end,"binary data: \0\r\n",16,'\'');
end = my_stpcpy(end,"')");

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

The `my_stpcpy()` function used in the example is included in the `libmysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the encoded string that is placed into the `to` argument, not including the terminating null byte.

Errors

None.

28.7.7.57 mysql_real_query()

```
int mysql_real_query(MYSQL *mysql, const char *stmt_str, unsigned long length)
```

Description

Note

`mysql_real_query()` is a synchronous function. Its asynchronous counterpart is `mysql_real_query_nonblocking()`, for use by applications that require asynchronous communication with the server. See Section 28.7.12, “C API Asynchronous Interface”.

`mysql_real_query()` executes the SQL statement pointed to by `stmt_str`, a string `length` bytes long. Normally, the string must consist of a single SQL statement without a terminating semicolon (`;`) or `\g`. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Section 28.7.23, “C API Multiple Statement Execution Support”.

`mysql_query()` cannot be used for statements that contain binary data; you must use `mysql_real_query()` instead. (Binary data may contain the `\0` character, which `mysql_query()` interprets as the end of the statement string.) In addition, `mysql_real_query()` is faster than `mysql_query()` because it does not call `strlen()` on the statement string.

If you want to know whether the statement returns a result set, you can use `mysql_field_count()` to check for this. See Section 28.7.7.22, “mysql_field_count()”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.58 mysql_refresh()

```
int mysql_refresh(MYSQL *mysql, unsigned int options)
```

Description

Note

`mysql_refresh()` is deprecated and will be removed in a future version of MySQL. Instead, use `mysql_query()` to execute a `FLUSH` statement.

This function flushes tables or caches, or resets replication server information. The connected user must have the `RELOAD` privilege.

The `options` argument is a bitmask composed from any combination of the following values. Multiple values can be OR'ed together to perform multiple operations with a single call.

- `REFRESH_GRANT`

  Refresh the grant tables, like `FLUSH PRIVILEGES`.

- `REFRESH_LOG`

  Flush the logs, like `FLUSH LOGS`.

- `REFRESH_TABLES`

  Flush the table cache, like `FLUSH TABLES`.

- `REFRESH_HOSTS`

  Flush the host cache, like `FLUSH HOSTS`.

- `REFRESH_STATUS`

  Reset status variables, like `FLUSH STATUS`.

- `REFRESH_THREADS`

  Flush the thread cache.

- `REFRESH_SLAVE`

  On a slave replication server, reset the master server information and restart the slave, like `RESET SLAVE`.

- `REFRESH_MASTER`

  On a master replication server, remove the binary log files listed in the binary log index and truncate the index file, like `RESET MASTER`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.59 mysql_reload()

```
int mysql_reload(MYSQL *mysql)
```

Description

Asks the MySQL server to reload the grant tables. The connected user must have the `RELOAD` privilege.

This function is deprecated. Use `mysql_query()` to issue an SQL `FLUSH PRIVILEGES` statement instead.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.60 mysql_reset_connection()

```
int mysql_reset_connection(MYSQL *mysql)
```

Description

Resets the connection to clear the session state.

`mysql_reset_connection()` has effects similar to `mysql_change_user()` or an auto-reconnect except that the connection is not closed and reopened, and reauthentication is not done. The write set session history is reset. See Section 28.7.7.3, “mysql_change_user()”, and Section 28.7.28, “C API Automatic Reconnection Control”.

The connection-related state is affected as follows:

- Any active transactions are rolled back and autocommit mode is reset.
- All table locks are released.
- All `TEMPORARY` tables are closed (and dropped).
- Session system variables are reinitialized to the values of the corresponding global system variables, including system variables that are set implicitly by statements such as `SET NAMES`.
- User variable settings are lost.
- Prepared statements are released.
- `HANDLER` variables are closed.

- The value of `LAST_INSERT_ID()` is reset to 0.
- Locks acquired with `GET_LOCK()` are released.

Return Values

Zero for success. Nonzero if an error occurred.

28.7.7.61 mysql_reset_server_public_key()

```
void mysql_reset_server_public_key(void)
```

Description

Clears from the client library any cached copy of the public key required by the server for RSA key pair-based password exchange. This might be necessary when the server has been restarted with a different RSA key pair after the client program had called `mysql_options()` with the `MYSQL_SERVER_PUBLIC_KEY` option to specify the RSA public key. In such cases, connection failure can occur due to key mismatch. To fix this problem, the client can use either of the following approaches:

- The client can call `mysql_reset_server_public_key()` to clear the cached key and try again, after the public key file on the client side has been replaced with a file containing the new public key.
- The client can call `mysql_reset_server_public_key()` to clear the cached key, then call `mysql_options()` with the `MYSQL_OPT_GET_SERVER_PUBLIC_KEY` option (instead of `MYSQL_SERVER_PUBLIC_KEY`) to request the required public key from the server Do not use both `MYSQL_OPT_GET_SERVER_PUBLIC_KEY` and `MYSQL_SERVER_PUBLIC_KEY` because in that case, `MYSQL_SERVER_PUBLIC_KEY` takes precedence.

Return Values

None.

Errors

None.

28.7.7.62 mysql_result_metadata()

```
enum enum_resultset_metadata mysql_result_metadata(MYSQL_RES *result)
```

Description

`mysql_result_metadata()` returns a value that indicates whether a result set has metadata. It can be useful for metadata-optional connections when the client does not know in advance whether particular result sets have metadata. For example, if a client executes a stored procedure that returns multiple result sets and might change the `resultset_metadata` system variable, the client can invoke `mysql_result_metadata()` for each result set to determine whether it has metadata.

For details about managing result set metadata transfer, see Section 28.7.27, “C API Optional Result Set Metadata”.

Return Values

`mysql_result_metadata()` returns one of these values:

```
enum enum_resultset_metadata {
 RESULTSET_METADATA_NONE= 0,
 RESULTSET_METADATA_FULL= 1
```

```
};
```

28.7.7.63 mysql_rollback()

```
bool mysql_rollback(MYSQL *mysql)
```

Description

Rolls back the current transaction.

The action of this function is subject to the value of the `completion_type` system variable. In particular, if the value of `completion_type` is `RELEASE` (or 2), the server performs a release after terminating a transaction and closes the client connection. Call `mysql_close()` from the client program to close the connection from the client side.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

28.7.7.64 mysql_row_seek()

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a query result set. The `offset` value is a row offset, typically a value returned from `mysql_row_tell()` or from `mysql_row_seek()`. This value is not a row number; to seek to a row within a result set by number, use `mysql_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_row_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

Errors

None.

28.7.7.65 mysql_row_tell()

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

Description

Returns the current position of the row cursor for the last `mysql_fetch_row()`. This value can be used as an argument to `mysql_row_seek()`.

Use `mysql_row_tell()` only after `mysql_store_result()`, not after `mysql_use_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

28.7.7.66 mysql_select_db()

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

Description

Causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that include no explicit database specifier.

`mysql_select_db()` fails unless the connected user can be authenticated as having permission to use the database or some object within it.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.67 mysql_server_end()

```
void mysql_server_end(void)
```

Description

This function finalizes the MySQL client library, which should be done when you are done using the library. However, `mysql_server_end()` is deprecated and `mysql_library_end()` should be used instead. See Section 28.7.7.40, “mysql_library_end()”.

Note

To avoid memory leaks after the application is done using the library (for example, after closing the connection to the server), be sure to call `mysql_server_end()` (or `mysql_library_end()`) explicitly. This enables memory managment to be performed to clean up and free resources used by the library.

Return Values

None.

28.7.7.68 mysql_server_init()

```
int mysql_server_init(int argc, char **argv, char **groups)
```

Description

This function initializes the MySQL client library, which must be done before you call any other MySQL function. However, `mysql_server_init()` is deprecated and you should call `mysql_library_init()` instead. See Section 28.7.7.41, “mysql_library_init()”.

Note

To avoid memory leaks after the application is done using the library (for example, after closing the connection to the server), be sure to call `mysql_server_end()` (or `mysql_library_end()`) explicitly. This enables memory managment to be performed to clean up and free resources used by the library. See Section 28.7.7.40, “mysql_library_end()”.

Return Values

Zero for success. Nonzero if an error occurred.

28.7.7.69 mysql_session_track_get_first()

```
int mysql_session_track_get_first(MYSQL *mysql, enum enum_session_state_type
type, const char **data, size_t *length)
```

Description

MySQL implements a session tracker mechanism whereby the server returns information about session state changes to clients. To control which notifications the server provides about state changes, client applications set system variables having names of the form `session_track_xxx`, such as `session_track_state_change`, `session_track_schema`, and `session_track_system_variables`. See Section 5.1.15, “Server Tracking of Client Session State Changes”.

Change notification occurs in the MySQL client/server protocol, which includes tracker information in OK packets so that session state changes can be detected. To enable client applications to extract state-change information from OK packets, the MySQL C API provides a pair of functions:

- `mysql_session_track_get_first()` fetches the first part of the state-change information received from the server.
- `mysql_session_track_get_next()` fetches any remaining state-change information received from the server. Following a successful call to `mysql_session_track_get_first()`, call this function repeatedly as long as it returns success.

The `mysql_session_track_get_first()` parameters are used as follows. These descriptions also apply to `mysql_session_track_get_next()`, which takes the same parameters.

- `mysql`: The connection handler.
- `type`: The tracker type indicating what kind of information to retrieve. Permitted tracker values are the members of the `enum_session_state_type` enumeration defined in `mysql_com.h`:

```
enum enum_session_state_type
{
  SESSION_TRACK_SYSTEM_VARIABLES,            /* Session system variables */
  SESSION_TRACK_SCHEMA,                      /* Current schema */
  SESSION_TRACK_STATE_CHANGE,                /* Session state changes */
  SESSION_TRACK_GTIDS,                       /* GTIDs */
  SESSION_TRACK_TRANSACTION_CHARACTERISTICS, /* Transaction characteristics */
  SESSION_TRACK_TRANSACTION_STATE            /* Transaction state */
};
```

The members of that enumeration may change over time as MySQL implements additional session-information trackers. To make it easy for applications to loop over all possible tracker types regardless of the number of members, the `SESSION_TRACK_BEGIN` and `SESSION_TRACK_END` symbols are defined to be equal to the first and last members of the `enum_session_state_type` enumeration. The example code shown later in this section demonstrates this technique. (Of course, if the enumeration members change, you must recompile your application to enable it to take account of new trackers.)

- `data`: The address of a `const char *` variable. Following a successful call, this variable points to the returned data, which should be considered read only.
- `length`: The address of a `size_t` variable. Following a successful call, this variable contains the length of the data pointed to by the `data` parameter.

The following discussion describes how to interpret the `data` and `length` values according to the `type` value. It also indicates which system variable enables notifications for each tracker type.

- `SESSION_TRACK_SCHEMA`: This tracker type indicates that the default schema has been set. `data` is a string containing the new default schema name. `length` is the string length.

  To enable notifications for this tracker type, enable the `session_track_schema` system variable.

- `SESSION_TRACK_SYSTEM_VARIABLES`: This tracker type indicates that one or more tracked session system variables have been assigned a value. When a session system variable is assigned, two values per variable are returned (in separate calls). For the first call, `data` is a string containing the variable name and `length` is the string length. For the second call, `data` is a string containing the variable value and `length` is the string length.

  By default, notification is enabled for `time_zone`, `autocommit`, `character_set_client`, `character_set_results`, and `character_set_connection`. To change the default notification for this tracker type, set the `session_track_schema` system variable to a list of comma-separated variables for which to track changes, or `*` to track changes for all variables. To disable notification of session variable assignments, set `session_track_system_variables` to the empty string.

- `SESSION_TRACK_STATE_CHANGE`: This tracker type indicates a change to some tracked attribute of session state. `data` is a byte containing a boolean flag that indicates whether session state changes occurred. `length` should be 1. The flag is represented as an ASCII value, not a binary (for example, `'1'`, not `0x01`).

  To enable notifications for this tracker type, enable the `session_track_state_change` system variable.

  This tracker reports changes for these attributes of session state:

  - The default schema (database).
  - Session-specific values for system variables.
  - User-defined variables.

  - Temporary tables.
  - Prepared statements.

- `SESSION_TRACK_GTIDS`: This tracker type indicates that GTIDS are available. `data` contains encoded GTID data, `length` indicates the data length. Once the `data` value has been extracted, it must be further interpreted into three parts: Encoding specification, length of GTIDs string, GTIDs string. Currently, there is only one encoding specification, so this should always be 0. The GTIDs string is in the standard format for specifying a set of GTID values; see GTID Sets.

  To enable notifications for this tracker type, set the `session_track_gtids` system variable.

- `SESSION_TRACK_TRANSACTION_CHARACTERISTICS`: This tracker type indicates that transaction characteristics are available. `data` is a string containing the characteristics data. `length` is the string length. The characteristics tracker data string may be empty, or it may contain one or more SQL statements, each terminated by a semicolon:

  - If no characteristics apply, the string is empty. The session defaults apply. (For isolation level and access mode, these defaults are given by the session values of the `transaction_isolation` and `transaction_read_only` system variables.)
  - If a transaction was explicitly started, the string contains the statement or statements required to restart the transaction with the same characteristics. As a general rule, this is a `START TRANSACTION` statement (possibly with one or more of `READ ONLY`, `READ WRITE`, and `WITH CONSISTENT SNAPSHOT`). If any characteristics apply that cannot be passed to `START TRANSACTION`, such as `ISOLATION LEVEL`, a suitable `SET TRANSACTION` statement is prepended (for example, `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; START TRANSACTION READ WRITE;`).
  - If a transaction was not explicitly started, but one-shot characteristics that apply only to the next transaction were set up, a `SET TRANSACTION` statement suitable for replicating that setup is generated (for example, `SET TRANSACTION READ ONLY;`).

    Next-transaction characteristics can be set using `SET TRANSACTION` without any `GLOBAL` or `SESSION` keyword, or by setting the `transaction_isolation` and `transaction_read_only` system variables using the syntax that applies only to the next transaction:

    ```
    SET @@transaction_isolation = value;
    SET @@transaction_read_only = value;
    ```

    For more information about transaction characteristic scope levels and how they are set, see Transaction Characteristic Scope.

  To enable notifications for this tracker type, set the `session_track_transaction_info` system variable to `CHARACTERISTICS` (which also enables the `SESSION_TRACK_TRANSACTION_STATE` tracker type).

  Transaction characteristics tracking enables the client to determine how to restart a transaction in another session so it has the same characteristics as in the original session.

  Because characteristics may be set using `SET TRANSACTION` before a transaction is started, it is not safe for the client to assume that there are no transaction characteristics if no transaction is active. It is therefore unsafe not to track transaction characteristics and just switch the connection when no transaction is active (whether this is detected by the transaction state tracker or the traditional `SERVER_STATUS_IN_TRANS` flag). A client *must* subscribe to the transaction characteristics tracker if it may wish to switch its session to another connection at some point and transactions may be used.

The characteristics tracker tracks changes to the one-shot characteristics that apply only to the next transaction. It does not track changes to the session variables. Therefore, the client additionally must track the `transaction_isolation` and `transaction_read_only` system variables to correctly determine the session defaults that apply when next-transaction characteristic values are empty. (To track these variables, list them in the value of the `session_track_system_variables` system variable.)

- `SESSION_TRACK_TRANSACTION_STATE`: This tracker type indicates that transaction state information is available. `data` is a string containing ASCII characters, each of which indicates some aspect of the transaction state. `length` is the string length (always 8).

  To enable notifications for this tracker type, set the `session_track_transaction_info` system variable to `STATE`.

  Transaction state tracking enables the client to determine whether a transaction is in progress and whether it could be moved to a different session without being rolled back.

  The scope of the tracker item is the transaction. All state-indicating flags persist until the transaction is committed or rolled back. As statements are added to the transaction, additional flags may be set in successive tracker data values. However, no flags are cleared until the transaction ends.

  Transaction state is reported as a string containing a sequence of ASCII characters. Each active state has a unique character assigned to it as well as a fixed position in the sequence. The following list describes the permitted values for positions 1 through 8 of the sequence:

  - Position 1: Whether an active transaction is ongoing.
    - `T`: An explicitly started transaction is ongoing.
    - `I`: An implicitly started transaction (`autocommit=0`) is ongoing.
    - `_`: There is no active transaction.
  - Position 2: Whether nontransactional tables were read in the context of the current transaction.
    - `r`: One or more nontransactional tables were read.
    - `_`: No nontransactional tables were read so far.
  - Position 3: Whether transactional tables were read in the context of the current transaction.
    - `R`: One or more transactional tables were read.
    - `_`: No transactional tables were read so far.
  - Position 4: Whether unsafe writes (writes to nontransactional tables) were performed in the context of the current transaction.
    - `w`: One or more nontransactional tables were written.
    - `_`: No nontransactional tables were written so far.
  - Position 5: Whether any transactional tables were written in the context of the current transaction.
    - `W`: One or more transactional tables were written.

  - `_`: No transactional tables were written so far.
- Position 6: Whether any unsafe statements were executed in the context of the current transaction. Statements containing nondeterministic constructs such as `RAND()` or `UUID()` are unsafe for statement-based replication.
  - `s`: One or more unsafe statements were executed.
  - `_`: No unsafe statements were executed so far.
- Position 7: Whether a result set was sent to the client during the current transaction.
  - `S`: A result set was sent.
  - `_`: No result sets were sent so far.
- Position 8: Whether a `LOCK TABLES` statement is in effect.
  - `L`: Tables are explicitly locked with `LOCK TABLES`.
  - `_`: `LOCK TABLES` is not active in the session.

Consider a session consisting of the following statements, including one to enable the transaction state tracker:

```
1. SET @@SESSION.session_track_transaction_info='STATE';
2. START TRANSACTION;
3. SELECT 1;
4. INSERT INTO t1 () VALUES();
5. INSERT INTO t1 () VALUES(1, RAND());
6. COMMIT;
```

With transaction state tracking enabled, the following `data` values result from those statements:

```
1. ________
2. T_______
3. T_____S_
4. T___W_S_
5. T___WsS_
6. ________
```

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

Example

The following example shows how to call `mysql_session_track_get_first()` and `mysql_session_track_get_next()` to retrieve and display all available session state-change information following successful execution of an SQL statement string (represented by `stmt_str`). It is assumed that the application has set the `session_track_xxx` system variables that enable the notifications it wishes to receive.

```
printf("Execute: %s\n", stmt_str);

if (mysql_query(mysql, stmt_str) != 0)
{
  fprintf(stderr, "Error %u: %s\n",
          mysql_errno(mysql), mysql_error(mysql));
  return;
}

MYSQL_RES *result = mysql_store_result(mysql);
if (result) /* there is a result set to fetch */
{
  /* ... process rows here ... */
  printf("Number of rows returned: %lu\n",
         (unsigned long) mysql_num_rows(result));
  mysql_free_result(result);
}
else        /* there is no result set */
{
  if (mysql_field_count(mysql) == 0)
  {
    printf("Number of rows affected: %lu\n",
           (unsigned long) mysql_affected_rows(mysql));
  }
  else      /* an error occurred */
  {
    fprintf(stderr, "Error %u: %s\n",
            mysql_errno(mysql), mysql_error(mysql));
  }
}

/* extract any available session state-change information */
enum enum_session_state_type type;
for (type = SESSION_TRACK_BEGIN; type <= SESSION_TRACK_END; type++)
{
  const char *data;
  size_t length;

  if (mysql_session_track_get_first(mysql, type, &data, &length) == 0)
  {
    /* print info type and initial data */
    printf("Type=%d:\n", type);
    printf("mysql_session_track_get_first(): length=%d; data=%*.*s\n",
           (int) length, (int) length, (int) length, data);

    /* check for more data */
    while (mysql_session_track_get_next(mysql, type, &data, &length) == 0)
    {
      printf("mysql_session_track_get_next(): length=%d; data=%*.*s\n",
             (int) length, (int) length, (int) length, data);
    }
  }
}
```

28.7.7.70 mysql_session_track_get_next()

```
int mysql_session_track_get_next(MYSQL *mysql, enum enum_session_state_type
type, const char **data, size_t *length)
```

Description

This function fetches additional session state-change information received from the server, following that retrieved by `mysql_session_track_get_first()`. The parameters for `mysql_session_track_get_next()` are the same as for `mysql_session_track_get_first()`.

Following a successful call to `mysql_session_track_get_first()`, call `mysql_session_track_get_next()` repeatedly until it returns nonzero to indicate no more information is available. The calling sequence for `mysql_session_track_get_next()` is similar to that for `mysql_session_track_get_first()`. For more information and an example that demonstrates both functions, see Section 28.7.7.69, “mysql_session_track_get_first()”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

None.

28.7.7.71 mysql_set_character_set()

```
int mysql_set_character_set(MYSQL *mysql, const char *csname)
```

Description

This function is used to set the default character set for the current connection. The string `csname` specifies a valid character set name. The connection collation becomes the default collation of the character set. This function works like the `SET NAMES` statement, but also sets the value of `mysql->charset`, and thus affects the character set used by `mysql_real_escape_string()`

Return Values

Zero for success. Nonzero if an error occurred.

Example

```
MYSQL mysql;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}

if (!mysql_set_character_set(&mysql, "utf8"))
{
    printf("New client character set: %s\n",
           mysql_character_set_name(&mysql));
}
```

28.7.7.72 mysql_set_local_infile_default()

```
void mysql_set_local_infile_default(MYSQL *mysql);
```

Description

Sets the `LOAD DATA LOCAL` callback functions to the defaults used internally by the C client library. The library calls this function automatically if `mysql_set_local_infile_handler()` has not been called or does not supply valid functions for each of its callbacks.

Return Values

None.

Errors

None.

28.7.7.73 mysql_set_local_infile_handler()

```
void mysql_set_local_infile_handler(MYSQL *mysql, int (*local_infile_init)(void
**, const char *, void *), int (*local_infile_read)(void *, char *, unsigned
int), void (*local_infile_end)(void *), int (*local_infile_error)(void *,
char*, unsigned int), void *userdata);
```

Description

This function installs callbacks to be used during the execution of `LOAD DATA LOCAL` statements. It enables application programs to exert control over local (client-side) data file reading. The arguments are the connection handler, a set of pointers to callback functions, and a pointer to a data area that the callbacks can use to share information.

To use `mysql_set_local_infile_handler()`, you must write the following callback functions:

```
int
local_infile_init(void **ptr, const char *filename, void *userdata);
```

The initialization function. This is called once to do any setup necessary, open the data file, allocate data structures, and so forth. The first `void**` argument is a pointer to a pointer. You can set the pointer (that is, `*ptr`) to a value that will be passed to each of the other callbacks (as a `void*`). The callbacks can use this pointed-to value to maintain state information. The `userdata` argument is the same value that is passed to `mysql_set_local_infile_handler()`.

Make the initialization function return zero for success, nonzero for an error.

```
int
local_infile_read(void *ptr, char *buf, unsigned int buf_len);
```

The data-reading function. This is called repeatedly to read the data file. `buf` points to the buffer where the read data is stored, and `buf_len` is the maximum number of bytes that the callback can read and store in the buffer. (It can read fewer bytes, but should not read more.)

The return value is the number of bytes read, or zero when no more data could be read (this indicates EOF). Return a value less than zero if an error occurs.

```
void
local_infile_end(void *ptr)
```

The termination function. This is called once after `local_infile_read()` has returned zero (EOF) or an error. Within this function, deallocate any memory allocated by `local_infile_init()` and perform any other cleanup necessary. It is invoked even if the initialization function returns an error.

```
int
local_infile_error(void *ptr,
                   char *error_msg,
                   unsigned int error_msg_len);
```

The error-handling function. This is called to get a textual error message to return to the user in case any of your other functions returns an error. `error_msg` points to the buffer into which the message is written,

and `error_msg_len` is the length of the buffer. Write the message as a null-terminated string, at most `error_msg_len`−1 bytes long.

The return value is the error number.

Typically, the other callbacks store the error message in the data structure pointed to by `ptr`, so that `local_infile_error()` can copy the message from there into `error_msg`.

After calling `mysql_set_local_infile_handler()` in your C code and passing pointers to your callback functions, you can then issue a `LOAD DATA LOCAL` statement (for example, by using `mysql_query()`). The client library automatically invokes your callbacks. The file name specified in `LOAD DATA LOCAL` will be passed as the second parameter to the `local_infile_init()` callback.

Return Values

None.

Errors

None.

28.7.7.74 mysql_set_server_option()

```
int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)
```

Description

Enables or disables an option for the connection. `option` can have one of the following values.

Option	Description
`MYSQL_OPTION_MULTI_STATEMENTS_ON`	Enable multiple-statement support
`MYSQL_OPTION_MULTI_STATEMENTS_OFF`	Disable multiple-statement support

If you enable multiple-statement support, you should retrieve results from calls to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see Section 28.7.23, “C API Multiple Statement Execution Support”.

Enabling multiple-statement support with `MYSQL_OPTION_MULTI_STATEMENTS_ON` does not have quite the same effect as enabling it by passing the `CLIENT_MULTI_STATEMENTS` flag to `mysql_real_connect()`: `CLIENT_MULTI_STATEMENTS` also enables `CLIENT_MULTI_RESULTS`. If you are using the `CALL` SQL statement in your programs, multiple-result support must be enabled; this means that `MYSQL_OPTION_MULTI_STATEMENTS_ON` by itself is insufficient to permit the use of `CALL`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `ER_UNKNOWN_COM_ERROR`

  The server did not support `mysql_set_server_option()` (which is the case that the server is older than 4.1.1) or the server did not support the option one tried to set.

28.7.7.75 mysql_shutdown()

```
int mysql_shutdown(MYSQL *mysql, enum mysql_enum_shutdown_level shutdown_level)
```

Description

Note

`mysql_shutdown()` is deprecated and will be removed in a future version of MySQL. Instead, use `mysql_query()` to execute a `SHUTDOWN` statement.

Asks the database server to shut down. The connected user must have the `SHUTDOWN` privilege. MySQL servers support only one type of shutdown; `shutdown_level` must be equal to `SHUTDOWN_DEFAULT`. Dynamically linked executables which have been compiled with older versions of the `libmysqlclient` headers and call `mysql_shutdown()` need to be used with the old `libmysqlclient` dynamic library.

An alternative to `mysql_shutdown()` is to use the `SHUTDOWN` SQL statement.

The shutdown process is described in Section 5.1.17, “The Server Shutdown Process”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.76 mysql_sqlstate()

```
const char *mysql_sqlstate(MYSQL *mysql)
```

Description

Returns a null-terminated string containing the SQLSTATE error code for the most recently executed SQL statement. The error code consists of five characters. `'00000'` means “no error.” The values are specified

by ANSI SQL and ODBC. For a list of possible values, see Appendix B, *Errors, Error Codes, and Common Problems*.

SQLSTATE values returned by `mysql_sqlstate()` differ from MySQL-specific error numbers returned by `mysql_errno()`. For example, the `mysql` client program displays errors using the following format, where `1146` is the `mysql_errno()` value and `'42S02'` is the corresponding `mysql_sqlstate()` value:

```
shell> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Not all MySQL error numbers are mapped to SQLSTATE error codes. The value `'HY000'` (general error) is used for unmapped error numbers.

If you call `mysql_sqlstate()` after `mysql_real_connect()` fails, `mysql_sqlstate()` might not return a useful value. For example, this happens if a host is blocked by the server and the connection is closed without any SQLSTATE value being sent to the client.

Return Values

A null-terminated character string containing the SQLSTATE error code.

See Also

See Section 28.7.7.14, “mysql_errno()”, Section 28.7.7.15, “mysql_error()”, and Section 28.7.11.27, “mysql_stmt_sqlstate()”.

28.7.7.77 mysql_ssl_set()

```
bool mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert, const char
*ca, const char *capath, const char *cipher)
```

Description

`mysql_ssl_set()` is used for establishing encrypted connections using SSL. The `mysql` argument must be a valid connection handler. Any unused SSL arguments may be given as `NULL`.

If used, `mysql_ssl_set()` must be called before `mysql_real_connect()`. `mysql_ssl_set()` does nothing unless SSL support is enabled in the client library.

It is optional to call `mysql_ssl_set()` to obtain an encrypted connection because by default, MySQL programs attempt to connect using encryption if the server supports encrypted connections, falling back to an unencrypted connection if an encrypted connection cannot be established (see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”). `mysql_ssl_set()` may be useful to applications that must specify particular certificate and key files, encryption ciphers, and so forth.

`mysql_ssl_set()` specifies SSL information such as certificate and key files for establishing an encrypted connection if such connections are available, but does not enforce any requirement that the connection obtained be encrypted. To require an encrypted connection, use the technique described in Section 28.7.22, “C API Encrypted Connection Support”.

For additional security relative to that provided by the default encryption, clients can supply a CA certificate matching the one used by the server and enable host name identity verification. In this way, the server and client place their trust in the same CA certificate and the client verifies that the host to which it connected is the one intended. For details, see Section 28.7.22, “C API Encrypted Connection Support”.

`mysql_ssl_set()` is a convenience function that is essentially equivalent to this set of `mysql_options()` calls:

```
mysql_options(mysql, MYSQL_OPT_SSL_KEY,    key);
mysql_options(mysql, MYSQL_OPT_SSL_CERT,   cert);
mysql_options(mysql, MYSQL_OPT_SSL_CA,     ca);
mysql_options(mysql, MYSQL_OPT_SSL_CAPATH, capath);
mysql_options(mysql, MYSQL_OPT_SSL_CIPHER, cipher);
```

Because of that equivalence, applications can, instead of calling `mysql_ssl_set()`, call `mysql_options()` directly, omitting calls for those options for which the option value is `NULL`. Moreover, `mysql_options()` offers encrypted-connection options not available using `mysql_ssl_set()`, such as `MYSQL_OPT_SSL_MODE` to specify the security state of the connection, and `MYSQL_OPT_TLS_VERSION` to specify the protocols the client permits for encrypted connections.

Arguments:

- `mysql`: The connection handler returned from `mysql_init()`.
- `key`: The path name of the client private key file.
- `cert`: The path name of the client public key certificate file.
- `ca`: The path name of the Certificate Authority (CA) certificate file. This option, if used, must specify the same certificate used by the server.
- `capath`: The path name of the directory that contains trusted SSL CA certificate files.
- `cipher`: The list of permissible ciphers for SSL encryption.

Return Values

This function always returns `0`. If SSL setup is incorrect, a subsequent `mysql_real_connect()` call returns an error when you attempt to connect.

28.7.7.78 mysql_stat()

```
const char *mysql_stat(MYSQL *mysql)
```

Description

Returns a character string containing information similar to that provided by the `mysqladmin status` command. This includes uptime in seconds and the number of running threads, questions, reloads, and open tables.

Return Values

A character string describing the server status. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.79 mysql_store_result()

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Description

Note

`mysql_store_result()` is a synchronous function. Its asynchronous counterpart is `mysql_store_result_nonblocking()`, for use by applications that require asynchronous communication with the server. See Section 28.7.12, “C API Asynchronous Interface”.

After invoking `mysql_query()` or `mysql_real_query()`, you must call `mysql_store_result()` or `mysql_use_result()` for every statement that successfully produces a result set (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth). You must also call `mysql_free_result()` after you are done with the result set.

You need not call `mysql_store_result()` or `mysql_use_result()` for other statements, but it does not do any harm or cause any notable performance degradation if you call `mysql_store_result()` in all cases. You can detect whether the statement has a result set by checking whether `mysql_store_result()` returns a nonzero value (more about this later).

If you enable multiple-statement support, you should retrieve results from calls to `mysql_query()` or `mysql_real_query()` by using a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see Section 28.7.23, “C API Multiple Statement Execution Support”.

If you want to know whether a statement should return a result set, you can use `mysql_field_count()` to check for this. See Section 28.7.7.22, “mysql_field_count()”.

`mysql_store_result()` reads the entire result of a query to the client, allocates a `MYSQL_RES` structure, and places the result into this structure.

`mysql_store_result()` returns `NULL` if the statement did not return a result set (for example, if it was an `INSERT` statement), or an error occurred and reading of the result set failed.

An empty result set is returned if there are no rows returned. (An empty result set differs from a null pointer as a return value.)

After you have called `mysql_store_result()` and gotten back a result that is not a null pointer, you can call `mysql_num_rows()` to find out how many rows are in the result set.

You can call `mysql_fetch_row()` to fetch rows from the result set, or `mysql_row_seek()` and `mysql_row_tell()` to obtain or set the current row position within the result set.

See Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”.

Return Values

A pointer to a `MYSQL_RES` result structure with the results. `NULL` if the statement did not return a result set or an error occurred. To determine whether an error occurred, check whether `mysql_error()` returns a nonempty string, `mysql_errno()` returns nonzero, or `mysql_field_count()` returns zero.

Errors

`mysql_store_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.80 mysql_thread_id()

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

Description

Returns the thread ID of the current connection. This value can be used as an argument to `mysql_kill()` to kill the thread.

If the connection is lost and you reconnect with `mysql_ping()`, the thread ID changes. This means you should not get the thread ID and store it for later. You should get it when you need it.

Note

This function does not work correctly if thread IDs become larger than 32 bits, which can occur on some systems. To avoid problems with `mysql_thread_id()`, do not use it. To get the connection ID, execute a `SELECT CONNECTION_ID()` query and retrieve the result.

Return Values

The thread ID of the current connection.

Errors

None.

28.7.7.81 mysql_use_result()

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

Description

After invoking `mysql_query()` or `mysql_real_query()`, you must call `mysql_store_result()` or `mysql_use_result()` for every statement that successfully produces a result set (`SELECT`, `SHOW`,

`DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth). You must also call `mysql_free_result()` after you are done with the result set.

`mysql_use_result()` initiates a result set retrieval but does not actually read the result set into the client like `mysql_store_result()` does. Instead, each row must be retrieved individually by making calls to `mysql_fetch_row()`. This reads the result of a query directly from the server without storing it in a temporary table or local buffer, which is somewhat faster and uses much less memory than `mysql_store_result()`. The client allocates memory only for the current row and a communication buffer that may grow up to `max_allowed_packet` bytes.

On the other hand, you should not use `mysql_use_result()` for locking reads if you are doing a lot of processing for each row on the client side, or if the output is sent to a screen on which the user may type a `^S` (stop scroll). This ties up the server and prevent other threads from updating any tables from which the data is being fetched.

When using `mysql_use_result()`, you must execute `mysql_fetch_row()` until a `NULL` value is returned, otherwise, the unfetched rows are returned as part of the result set for your next query. The C API gives the error `Commands out of sync; you can't run this command now` if you forget to do this!

You may not use `mysql_data_seek()`, `mysql_row_seek()`, `mysql_row_tell()`, `mysql_num_rows()`, or `mysql_affected_rows()` with a result returned from `mysql_use_result()`, nor may you issue other queries until `mysql_use_result()` has finished. (However, after you have fetched all the rows, `mysql_num_rows()` accurately returns the number of rows fetched.)

You must call `mysql_free_result()` once you are done with the result set.

Return Values

A `MYSQL_RES` result structure. `NULL` if an error occurred.

Errors

`mysql_use_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.7.82 mysql_warning_count()

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

Description

Returns the number of errors, warnings, and notes generated during execution of the previous SQL statement.

Return Values

The warning count.

Errors

None.

28.7.8 C API Prepared Statements

The MySQL client/server protocol provides for the use of prepared statements. This capability uses the `MYSQL_STMT` statement handler data structure returned by the `mysql_stmt_init()` initialization function. Prepared execution is an efficient way to execute a statement more than once. The statement is first parsed to prepare it for execution. Then it is executed one or more times at a later time, using the statement handler returned by the initialization function.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Prepared statements might not provide a performance increase in some situations. For best results, test your application both with prepared and nonprepared statements and choose whichever yields best performance.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

For a list of SQL statements that can be used as prepared statements, see Section 13.5, “Prepared SQL Statement Syntax”.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.10.3, “Caching of Prepared Statements and Stored Programs”.

28.7.9 C API Prepared Statement Data Structures

Prepared statements use several data structures:

- To obtain a statement handler, pass a `MYSQL` connection handler to `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT` data structure. This structure is used for further operations with the statement. To specify the statement to prepare, pass the `MYSQL_STMT` pointer and the statement string to `mysql_stmt_prepare()`.
- To provide input parameters for a prepared statement, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_param()`. To receive output column values, set up `MYSQL_BIND` structures and pass them to `mysql_stmt_bind_result()`.

- The `MYSQL_TIME` structure is used to transfer temporal data in both directions.

The following discussion describes the prepared statement data types in detail. For examples that show how to use them, see Section 28.7.11.10, “mysql_stmt_execute()”, and Section 28.7.11.11, “mysql_stmt_fetch()”.

- `MYSQL_STMT`

  This structure is a handler for a prepared statement. A handler is created by calling `mysql_stmt_init()`, which returns a pointer to a `MYSQL_STMT`. The handler is used for all subsequent operations with the statement until you close it with `mysql_stmt_close()`, at which point the handler becomes invalid and should no longer be used.

  The `MYSQL_STMT` structure has no members intended for application use. Applications should not try to copy a `MYSQL_STMT` structure. There is no guarantee that such a copy will be usable.

  Multiple statement handlers can be associated with a single connection. The limit on the number of handlers depends on the available system resources.

- `MYSQL_BIND`

  This structure is used both for statement input (data values sent to the server) and output (result values returned from the server):

  - For input, use `MYSQL_BIND` structures with `mysql_stmt_bind_param()` to bind parameter data values to buffers for use by `mysql_stmt_execute()`.
  - For output, use `MYSQL_BIND` structures with `mysql_stmt_bind_result()` to bind buffers to result set columns, for use in fetching rows with `mysql_stmt_fetch()`.

  To use a `MYSQL_BIND` structure, zero its contents to initialize it, then set its members appropriately. For example, to declare and initialize an array of three `MYSQL_BIND` structures, use this code:

```
MYSQL_BIND bind[3];
memset(bind, 0, sizeof(bind));
```

  The `MYSQL_BIND` structure contains the following members for use by application programs. For several of the members, the manner of use depends on whether the structure is used for input or output.

  - `enum enum_field_types buffer_type`

    The type of the buffer. This member indicates the data type of the C language variable bound to a statement parameter or result set column. For input, `buffer_type` indicates the type of the variable containing the value to be sent to the server. For output, it indicates the type of the variable into which a value received from the server should be stored. For permissible `buffer_type` values, see Section 28.7.9.1, “C API Prepared Statement Type Codes”.

  - `void *buffer`

    A pointer to the buffer to be used for data transfer. This is the address of a C language variable.

    For input, `buffer` is a pointer to the variable in which you store the data value for a statement parameter. When you call `mysql_stmt_execute()`, MySQL use the value stored in the variable in place of the corresponding parameter marker in the statement (specified with `?` in the statement string).

For output, `buffer` is a pointer to the variable in which to return a result set column value. When you call `mysql_stmt_fetch()`, MySQL stores a column value from the current row of the result set in this variable. You can access the value when the call returns.

To minimize the need for MySQL to perform type conversions between C language values on the client side and SQL values on the server side, use C variables that have types similar to those of the corresponding SQL values:

- For numeric data types, `buffer` should point to a variable of the proper numeric C type. For integer variables (which can be `char` for single-byte values or an integer type for larger values), you should also indicate whether the variable has the `unsigned` attribute by setting the `is_unsigned` member, described later.
- For character (nonbinary) and binary string data types, `buffer` should point to a character buffer.
- For date and time data types, `buffer` should point to a `MYSQL_TIME` structure.

For guidelines about mapping between C types and SQL types and notes about type conversions, see Section 28.7.9.1, “C API Prepared Statement Type Codes”, and Section 28.7.9.2, “C API Prepared Statement Type Conversions”.

- `unsigned long buffer_length`

  The actual size of `*buffer` in bytes. This indicates the maximum amount of data that can be stored in the buffer. For character and binary C data, the `buffer_length` value specifies the length of `*buffer` when used with `mysql_stmt_bind_param()` to specify input values, or the maximum number of output data bytes that can be fetched into the buffer when used with `mysql_stmt_bind_result()`.

- `unsigned long *length`

  A pointer to an `unsigned long` variable that indicates the actual number of bytes of data stored in `*buffer`. `length` is used for character or binary C data.

  For input parameter data binding, set `*length` to indicate the actual length of the parameter value stored in `*buffer`. This is used by `mysql_stmt_execute()`.

  For output value binding, MySQL sets `*length` when you call `mysql_stmt_fetch()`. The `mysql_stmt_fetch()` return value determines how to interpret the length:

  - If the return value is 0, `*length` indicates the actual length of the parameter value.
  - If the return value is `MYSQL_DATA_TRUNCATED`, `*length` indicates the nontruncated length of the parameter value. In this case, the minimum of `*length` and `buffer_length` indicates the actual length of the value.

  `length` is ignored for numeric and temporal data types because the `buffer_type` value determines the length of the data value.

  If you must determine the length of a returned value before fetching it, see Section 28.7.11.11, “mysql_stmt_fetch()”, for some strategies.

- `bool *is_null`

  This member points to a `bool` variable that is true if a value is `NULL`, false if it is not `NULL`. For input, set `*is_null` to true to indicate that you are passing a `NULL` value as a statement parameter.

`is_null` is a *pointer* to a boolean scalar, not a boolean scalar, to provide flexibility in how you specify `NULL` values:

- If your data values are always `NULL`, use `MYSQL_TYPE_NULL` as the `buffer_type` value when you bind the column. The other `MYSQL_BIND` members, including `is_null`, do not matter.
- If your data values are always `NOT NULL`, set `is_null = (bool*) 0`, and set the other members appropriately for the variable you are binding.
- In all other cases, set the other members appropriately and set `is_null` to the address of a `bool` variable. Set that variable's value to true or false appropriately between executions to indicate whether the corresponding data value is `NULL` or `NOT NULL`, respectively.

For output, when you fetch a row, MySQL sets the value pointed to by `is_null` to true or false according to whether the result set column value returned from the statement is or is not `NULL`.

- `bool is_unsigned`

  This member applies for C variables with data types that can be `unsigned` (`char`, `short int`, `int`, `long long int`). Set `is_unsigned` to true if the variable pointed to by `buffer` is `unsigned` and false otherwise. For example, if you bind a `signed char` variable to `buffer`, specify a type code of `MYSQL_TYPE_TINY` and set `is_unsigned` to false. If you bind an `unsigned char` instead, the type code is the same but `is_unsigned` should be true. (For `char`, it is not defined whether it is signed or unsigned, so it is best to be explicit about signedness by using `signed char` or `unsigned char`.)

  `is_unsigned` applies only to the C language variable on the client side. It indicates nothing about the signedness of the corresponding SQL value on the server side. For example, if you use an `int` variable to supply a value for a `BIGINT UNSIGNED` column, `is_unsigned` should be false because `int` is a signed type. If you use an `unsigned int` variable to supply a value for a `BIGINT` column, `is_unsigned` should be true because `unsigned int` is an unsigned type. MySQL performs the proper conversion between signed and unsigned values in both directions, although a warning occurs if truncation results.

- `bool *error`

  For output, set this member to point to a `bool` variable to have truncation information for the parameter stored there after a row fetching operation. When truncation reporting is enabled, `mysql_stmt_fetch()` returns `MYSQL_DATA_TRUNCATED` and `*error` is true in the `MYSQL_BIND` structures for parameters in which truncation occurred. Truncation indicates loss of sign or significant digits, or that a string was too long to fit in a column. Truncation reporting is enabled by default, but can be controlled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option.

- `MYSQL_TIME`

This structure is used to send and receive `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` data directly to and from the server. Set the `buffer` member to point to a `MYSQL_TIME` structure, and set the `buffer_type` member of a `MYSQL_BIND` structure to one of the temporal types (`MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATE`, `MYSQL_TYPE_DATETIME`, `MYSQL_TYPE_TIMESTAMP`).

The `MYSQL_TIME` structure contains the members listed in the following table.

Member	Description
`unsigned int year`	The year

Member	Description
`unsigned int month`	The month of the year
`unsigned int day`	The day of the month
`unsigned int hour`	The hour of the day
`unsigned int minute`	The minute of the hour
`unsigned int second`	The second of the minute
`bool neg`	A boolean flag indicating whether the time is negative
`unsigned long second_part`	The fractional part of the second in microseconds

Only those parts of a `MYSQL_TIME` structure that apply to a given type of temporal value are used. The `year`, `month`, and `day` elements are used for `DATE`, `DATETIME`, and `TIMESTAMP` values. The `hour`, `minute`, and `second` elements are used for `TIME`, `DATETIME`, and `TIMESTAMP` values. See Section 28.7.24, “C API Prepared Statement Handling of Date and Time Values”.

28.7.9.1 C API Prepared Statement Type Codes

The `buffer_type` member of `MYSQL_BIND` structures indicates the data type of the C language variable bound to a statement parameter or result set column. For input, `buffer_type` indicates the type of the variable containing the value to be sent to the server. For output, it indicates the type of the variable into which a value received from the server should be stored.

The following table shows the permissible values for the `buffer_type` member of `MYSQL_BIND` structures for input values sent to the server. The table shows the C variable types that you can use, the corresponding type codes, and the SQL data types for which the supplied value can be used without conversion. Choose the `buffer_type` value according to the data type of the C language variable that you are binding. For the integer types, you should also set the `is_unsigned` member to indicate whether the variable is signed or unsigned.

Input Variable C Type	`buffer_type` Value	SQL Type of Destination Value
`signed char`	`MYSQL_TYPE_TINY`	`TINYINT`
`short int`	`MYSQL_TYPE_SHORT`	`SMALLINT`
`int`	`MYSQL_TYPE_LONG`	`INT`
`long long int`	`MYSQL_TYPE_LONGLONG`	`BIGINT`
`float`	`MYSQL_TYPE_FLOAT`	`FLOAT`
`double`	`MYSQL_TYPE_DOUBLE`	`DOUBLE`
`MYSQL_TIME`	`MYSQL_TYPE_TIME`	`TIME`
`MYSQL_TIME`	`MYSQL_TYPE_DATE`	`DATE`
`MYSQL_TIME`	`MYSQL_TYPE_DATETIME`	`DATETIME`
`MYSQL_TIME`	`MYSQL_TYPE_TIMESTAMP`	`TIMESTAMP`
`char[]`	`MYSQL_TYPE_STRING`	`TEXT`, `CHAR`, `VARCHAR`
`char[]`	`MYSQL_TYPE_BLOB`	`BLOB`, `BINARY`, `VARBINARY`
	`MYSQL_TYPE_NULL`	`NULL`

Use `MYSQL_TYPE_NULL` as indicated in the description for the `is_null` member in Section 28.7.9, “C API Prepared Statement Data Structures”.

For input string data, use MYSQL_TYPE_STRING or MYSQL_TYPE_BLOB depending on whether the value is a character (nonbinary) or binary string:

- MYSQL_TYPE_STRING indicates character input string data. The value is assumed to be in the character set indicated by the character_set_client system variable. If the server stores the value into a column with a different character set, it converts the value to that character set.
- MYSQL_TYPE_BLOB indicates binary input string data. The value is treated as having the binary character set. That is, it is treated as a byte string and no conversion occurs.

The following table shows the permissible values for the buffer_type member of MYSQL_BIND structures for output values received from the server. The table shows the SQL types of received values, the corresponding type codes that such values have in result set metadata, and the recommended C language data types to bind to the MYSQL_BIND structure to receive the SQL values without conversion. Choose the buffer_type value according to the data type of the C language variable that you are binding. For the integer types, you should also set the is_unsigned member to indicate whether the variable is signed or unsigned.

SQL Type of Received Value	buffer_type Value	Output Variable C Type
TINYINT	MYSQL_TYPE_TINY	signed char
SMALLINT	MYSQL_TYPE_SHORT	short int
MEDIUMINT	MYSQL_TYPE_INT24	int
INT	MYSQL_TYPE_LONG	int
BIGINT	MYSQL_TYPE_LONGLONG	long long int
FLOAT	MYSQL_TYPE_FLOAT	float
DOUBLE	MYSQL_TYPE_DOUBLE	double
DECIMAL	MYSQL_TYPE_NEWDECIMAL	char[]
YEAR	MYSQL_TYPE_SHORT	short int
TIME	MYSQL_TYPE_TIME	MYSQL_TIME
DATE	MYSQL_TYPE_DATE	MYSQL_TIME
DATETIME	MYSQL_TYPE_DATETIME	MYSQL_TIME
TIMESTAMP	MYSQL_TYPE_TIMESTAMP	MYSQL_TIME
CHAR, BINARY	MYSQL_TYPE_STRING	char[]
VARCHAR, VARBINARY	MYSQL_TYPE_VAR_STRING	char[]
TINYBLOB, TINYTEXT	MYSQL_TYPE_TINY_BLOB	char[]
BLOB, TEXT	MYSQL_TYPE_BLOB	char[]
MEDIUMBLOB, MEDIUMTEXT	MYSQL_TYPE_MEDIUM_BLOB	char[]
LONGBLOB, LONGTEXT	MYSQL_TYPE_LONG_BLOB	char[]
BIT	MYSQL_TYPE_BIT	char[]

28.7.9.2 C API Prepared Statement Type Conversions

Prepared statements transmit data between the client and server using C language variables on the client side that correspond to SQL values on the server side. If there is a mismatch between the C variable type on the client side and the corresponding SQL value type on the server side, MySQL performs implicit type conversions in both directions.

MySQL knows the type code for the SQL value on the server side. The `buffer_type` value in the `MYSQL_BIND` structure indicates the type code of the C variable that holds the value on the client side. The two codes together tell MySQL what conversion must be performed, if any. Here are some examples:

- If you use `MYSQL_TYPE_LONG` with an `int` variable to pass an integer value to the server that is to be stored into a `FLOAT` column, MySQL converts the value to floating-point format before storing it.
- If you fetch an SQL `MEDIUMINT` column value, but specify a `buffer_type` value of `MYSQL_TYPE_LONGLONG` and use a C variable of type `long long int` as the destination buffer, MySQL converts the `MEDIUMINT` value (which requires less than 8 bytes) for storage into the `long long int` (an 8-byte variable).
- If you fetch a numeric column with a value of 255 into a `char[4]` character array and specify a `buffer_type` value of `MYSQL_TYPE_STRING`, the resulting value in the array is a 4-byte string `'255\0'`.
- MySQL returns `DECIMAL` values as the string representation of the original server-side value, which is why the corresponding C type is `char[]`. For example, `12.345` is returned to the client as `'12.345'`. If you specify `MYSQL_TYPE_NEWDECIMAL` and bind a string buffer to the `MYSQL_BIND` structure, `mysql_stmt_fetch()` stores the value in the buffer as a string without conversion. If instead you specify a numeric variable and type code, `mysql_stmt_fetch()` converts the string-format `DECIMAL` value to numeric form.
- For the `MYSQL_TYPE_BIT` type code, `BIT` values are returned into a string buffer, which is why the corresponding C type is `char[]`. The value represents a bit string that requires interpretation on the client side. To return the value as a type that is easier to deal with, you can cause the value to be cast to integer using either of the following types of expressions:

  ```
  SELECT bit_col + 0 FROM t
  SELECT CAST(bit_col AS UNSIGNED) FROM t
  ```

  To retrieve the value, bind an integer variable large enough to hold the value and specify the appropriate corresponding integer type code.

Before binding variables to the `MYSQL_BIND` structures that are to be used for fetching column values, you can check the type codes for each column of the result set. This might be desirable if you want to determine which variable types would be best to use to avoid type conversions. To get the type codes, call `mysql_stmt_result_metadata()` after executing the prepared statement with `mysql_stmt_execute()`. The metadata provides access to the type codes for the result set as described in Section 28.7.11.23, “mysql_stmt_result_metadata()”, and Section 28.7.5, “C API Data Structures”.

To determine whether output string values in a result set returned from the server contain binary or nonbinary data, check whether the `charsetnr` value of the result set metadata is 63 (see Section 28.7.5, “C API Data Structures”). If so, the character set is `binary`, which indicates binary rather than nonbinary data. This enables you to distinguish `BINARY` from `CHAR`, `VARBINARY` from `VARCHAR`, and the `BLOB` types from the `TEXT` types.

If you cause the `max_length` member of the `MYSQL_FIELD` column metadata structures to be set (by calling `mysql_stmt_attr_set()`), be aware that the `max_length` values for the result set indicate the lengths of the longest string representation of the result values, not the lengths of the binary representation. That is, `max_length` does not necessarily correspond to the size of the buffers needed to fetch the values with the binary protocol used for prepared statements. Choose the size of the buffers according to the types of the variables into which you fetch the values. For example, a `TINYINT` column containing the value -128 might have a `max_length` value of 4. But the binary representation of any

`TINYINT` value requires only 1 byte for storage, so you can supply a `signed char` variable in which to store the value and set `is_unsigned` to indicate that values are signed.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.10.3, “Caching of Prepared Statements and Stored Programs”.

28.7.10 C API Prepared Statement Function Overview

The following list summarizes the functions available for prepared statement processing. For greater detail, see the descriptions in Section 28.7.11, “C API Prepared Statement Function Descriptions”.

- `mysql_stmt_affected_rows()`: Returns the number of rows changed, deleted, or inserted by prepared `UPDATE`, `DELETE`, or `INSERT` statement.
- `mysql_stmt_attr_get()`: Gets the value of an attribute for a prepared statement.
- `mysql_stmt_attr_set()`: Sets an attribute for a prepared statement.
- `mysql_stmt_bind_param()`: Associates application data buffers with the parameter markers in a prepared SQL statement.
- `mysql_stmt_bind_result()`: Associates application data buffers with columns in a result set.
- `mysql_stmt_close()`: Frees memory used by a prepared statement.
- `mysql_stmt_data_seek()`: Seeks to an arbitrary row number in a statement result set.
- `mysql_stmt_errno()`: Returns the error number for the last statement execution.
- `mysql_stmt_error()`: Returns the error message for the last statement execution.
- `mysql_stmt_execute()`: Executes a prepared statement.
- `mysql_stmt_fetch()`: Fetches the next row of data from a result set and returns data for all bound columns.
- `mysql_stmt_fetch_column()`: Fetches data for one column of the current row of a result set.
- `mysql_stmt_field_count()`: Returns the number of result columns for the most recent statement.
- `mysql_stmt_free_result()`: Frees the resources allocated to a statement handler.
- `mysql_stmt_init()`: Allocates memory for a `MYSQL_STMT` structure and initializes it.
- `mysql_stmt_insert_id()`: Returns the ID generated for an `AUTO_INCREMENT` column by a prepared statement.
- `mysql_stmt_next_result()`: Returns/initiates the next result in a multiple-result execution.
- `mysql_stmt_num_rows()`: Returns the row count from a buffered statement result set.
- `mysql_stmt_param_count()`: Returns the number of parameters in a prepared statement.
- `mysql_stmt_param_metadata()`: Returns parameter metadata in the form of a result set. (This function actually does nothing.)
- `mysql_stmt_prepare()`: Prepares an SQL statement string for execution.
- `mysql_stmt_reset()`: Resets the statement buffers in the server.
- `mysql_stmt_result_metadata()`: Returns prepared statement metadata in the form of a result set.

- `mysql_stmt_row_seek()`: Seeks to a row offset in a statement result set, using value returned from `mysql_stmt_row_tell()`.
- `mysql_stmt_row_tell()`: Returns the statement row cursor position.
- `mysql_stmt_send_long_data()`: Sends long data in chunks to server.
- `mysql_stmt_sqlstate()`: Returns the SQLSTATE error code for the last statement execution.
- `mysql_stmt_store_result()`: Retrieves a complete result set to the client.

Call `mysql_stmt_init()` to create a statement handler, then `mysql_stmt_prepare()` to prepare the statement string, `mysql_stmt_bind_param()` to supply the parameter data, and `mysql_stmt_execute()` to execute the statement. You can repeat the `mysql_stmt_execute()` by changing parameter values in the respective buffers supplied through `mysql_stmt_bind_param()`.

You can send text or binary data in chunks to server using `mysql_stmt_send_long_data()`. See Section 28.7.11.26, “mysql_stmt_send_long_data()”.

If the statement is a `SELECT` or any other statement that produces a result set, `mysql_stmt_prepare()` also returns the result set metadata information in the form of a `MYSQL_RES` result set through `mysql_stmt_result_metadata()`.

You can supply the result buffers using `mysql_stmt_bind_result()`, so that the `mysql_stmt_fetch()` automatically returns data to these buffers. This is row-by-row fetching.

When statement execution has been completed, close the statement handler using `mysql_stmt_close()` so that all resources associated with it can be freed. At that point the handler becomes invalid and should no longer be used.

If you obtained a `SELECT` statement's result set metadata by calling `mysql_stmt_result_metadata()`, you should also free the metadata using `mysql_free_result()`.

Execution Steps

To prepare and execute a statement, an application follows these steps:

1. Create a prepared statement handler with `mysql_stmt_init()`. To prepare the statement on the server, call `mysql_stmt_prepare()` and pass it a string containing the SQL statement.
2. If the statement will produce a result set, call `mysql_stmt_result_metadata()` to obtain the result set metadata. This metadata is itself in the form of result set, albeit a separate one from the one that contains the rows returned by the query. The metadata result set indicates how many columns are in the result and contains information about each column.
3. Set the values of any parameters using `mysql_stmt_bind_param()`. All parameters must be set. Otherwise, statement execution returns an error or produces unexpected results.
4. Call `mysql_stmt_execute()` to execute the statement.
5. If the statement produces a result set, bind the data buffers to use for retrieving the row values by calling `mysql_stmt_bind_result()`.
6. Fetch the data into the buffers row by row by calling `mysql_stmt_fetch()` repeatedly until no more rows are found.
7. Repeat steps 3 through 6 as necessary, by changing the parameter values and re-executing the statement.

When `mysql_stmt_prepare()` is called, the MySQL client/server protocol performs these actions:

- The server parses the statement and sends the okay status back to the client by assigning a statement ID. It also sends total number of parameters, a column count, and its metadata if it is a result set oriented statement. All syntax and semantics of the statement are checked by the server during this call.
- The client uses this statement ID for the further operations, so that the server can identify the statement from among its pool of statements.

When `mysql_stmt_execute()` is called, the MySQL client/server protocol performs these actions:

- The client uses the statement handler and sends the parameter data to the server.
- The server identifies the statement using the ID provided by the client, replaces the parameter markers with the newly supplied data, and executes the statement. If the statement produces a result set, the server sends the data back to the client. Otherwise, it sends an okay status and the number of rows changed, deleted, or inserted.

When `mysql_stmt_fetch()` is called, the MySQL client/server protocol performs these actions:

- The client reads the data from the current row of the result set and places it into the application data buffers by doing the necessary conversions. If the application buffer type is same as that of the field type returned from the server, the conversions are straightforward.

If an error occurs, you can get the statement error number, error message, and SQLSTATE code using `mysql_stmt_errno()`, `mysql_stmt_error()`, and `mysql_stmt_sqlstate()`, respectively.

Prepared Statement Logging

For prepared statements that are executed with the `mysql_stmt_prepare()` and `mysql_stmt_execute()` C API functions, the server writes `Prepare` and `Execute` lines to the general query log so that you can tell when statements are prepared and executed.

Suppose that you prepare and execute a statement as follows:

1. Call `mysql_stmt_prepare()` to prepare the statement string `"SELECT ?"`.
2. Call `mysql_stmt_bind_param()` to bind the value `3` to the parameter in the prepared statement.
3. Call `mysql_stmt_execute()` to execute the prepared statement.

As a result of the preceding calls, the server writes the following lines to the general query log:

```
Prepare  [1] SELECT ?
Execute  [1] SELECT 3
```

Each `Prepare` and `Execute` line in the log is tagged with a `[N]` statement identifier so that you can keep track of which prepared statement is being logged. *`N`* is a positive integer. If there are multiple prepared statements active simultaneously for the client, *`N`* may be greater than 1. Each `Execute` lines shows a prepared statement after substitution of data values for `?` parameters.

28.7.11 C API Prepared Statement Function Descriptions

To prepare and execute queries, use the functions described in detail in the following sections.

All functions that operate with a `MYSQL_STMT` structure begin with the prefix `mysql_stmt_`.

To create a `MYSQL_STMT` handler, use the `mysql_stmt_init()` function.

28.7.11.1 mysql_stmt_affected_rows()

```
uint64_t mysql_stmt_affected_rows(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_affected_rows()` may be called immediately after executing a statement with `mysql_stmt_execute()`. It is like `mysql_affected_rows()` but for prepared statements. For a description of what the affected-rows value returned by this function means, See Section 28.7.7.1, “mysql_affected_rows()”.

Errors

None.

Example

See the Example in Section 28.7.11.10, “mysql_stmt_execute()”.

28.7.11.2 mysql_stmt_attr_get()

```
bool mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
void *arg)
```

Description

Can be used to get the current value for a statement attribute.

The `option` argument is the option that you want to get; the `arg` should point to a variable that should contain the option value. If the option is an integer, `arg` should point to the value of the integer.

See Section 28.7.11.3, “mysql_stmt_attr_set()”, for a list of options and option types.

Return Values

Zero for success. Nonzero if `option` is unknown.

Errors

None.

28.7.11.3 mysql_stmt_attr_set()

```
bool mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option,
const void *arg)
```

Description

Can be used to affect behavior for a prepared statement. This function may be called multiple times to set several options.

The `option` argument is the option that you want to set. The `arg` argument is the value for the option. `arg` should point to a variable that is set to the desired attribute value. The variable type is as indicated in the following table.

The following table shows the possible `option` values.

Option	Argument Type	Function
`STMT_ATTR_UPDATE_MAX_LENGTH`	`bool *`	If set to 1, causes `mysql_stmt_store_result()` to update the metadata `MYSQL_FIELD->max_length` value.

Option	Argument Type	Function
STMT_ATTR_CURSOR_TYPE	unsigned long *	Type of cursor to open for statement when mysql_stmt_execute() is invoked. *arg can be CURSOR_TYPE_NO_CURSOR (the default) or CURSOR_TYPE_READ_ONLY.
STMT_ATTR_PREFETCH_ROWS	unsigned long *	Number of rows to fetch from server at a time when using a cursor. *arg can be in the range from 1 to the maximum value of unsigned long. The default is 1.

If you use the STMT_ATTR_CURSOR_TYPE option with CURSOR_TYPE_READ_ONLY, a cursor is opened for the statement when you invoke mysql_stmt_execute(). If there is already an open cursor from a previous mysql_stmt_execute() call, it closes the cursor before opening a new one. mysql_stmt_reset() also closes any open cursor before preparing the statement for re-execution. mysql_stmt_free_result() closes any open cursor.

If you open a cursor for a prepared statement, mysql_stmt_store_result() is unnecessary, because that function causes the result set to be buffered on the client side.

Return Values

Zero for success. Nonzero if option is unknown.

Errors

None.

Example

The following example opens a cursor for a prepared statement and sets the number of rows to fetch at a time to 5:

```
MYSQL_STMT *stmt;
int rc;
unsigned long type;
unsigned long prefetch_rows = 5;

stmt = mysql_stmt_init(mysql);
type = (unsigned long) CURSOR_TYPE_READ_ONLY;
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_CURSOR_TYPE, (void*) &type);
/* ... check return value ... */
rc = mysql_stmt_attr_set(stmt, STMT_ATTR_PREFETCH_ROWS,
                         (void*) &prefetch_rows);
/* ... check return value ... */
```

28.7.11.4 mysql_stmt_bind_param()

```
bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

mysql_stmt_bind_param() is used to bind input data for the parameter markers in the SQL statement that was passed to mysql_stmt_prepare(). It uses MYSQL_BIND structures to supply the data. bind is the address of an array of MYSQL_BIND structures. The client library expects the array to contain one element for each ? parameter marker that is present in the query.

Suppose that you prepare the following statement:

```
INSERT INTO mytbl VALUES(?,?,?)
```

When you bind the parameters, the array of `MYSQL_BIND` structures must contain three elements, and can be declared like this:

```
MYSQL_BIND bind[3];
```

Section 28.7.9, “C API Prepared Statement Data Structures”, describes the members of each `MYSQL_BIND` element and how they should be set to provide input values.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_UNSUPPORTED_PARAM_TYPE`

  The conversion is not supported. Possibly the `buffer_type` value is invalid or is not one of the supported types.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

See the Example in Section 28.7.11.10, “mysql_stmt_execute()”.

28.7.11.5 mysql_stmt_bind_result()

```
bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_result()` is used to associate (that is, bind) output columns in the result set to data buffers and length buffers. When `mysql_stmt_fetch()` is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified buffers.

All columns must be bound to buffers prior to calling `mysql_stmt_fetch()`. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain one element for each column of the result set. If you do not bind columns to `MYSQL_BIND` structures, `mysql_stmt_fetch()` simply ignores the data fetch. The buffers should be large enough to hold the data values, because the protocol does not return data values in chunks.

A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time `mysql_stmt_fetch()` is called. Suppose that an application binds the columns in a result set and calls `mysql_stmt_fetch()`. The client/server protocol returns data in the bound buffers. Then suppose that the application binds the columns to a different set of buffers. The protocol places data into the newly bound buffers when the next call to `mysql_stmt_fetch()` occurs.

To bind a column, an application calls `mysql_stmt_bind_result()` and passes the type, address, and length of the output buffer into which the value should be stored. Section 28.7.9, “C API Prepared Statement Data Structures”, describes the members of each `MYSQL_BIND` element and how they should be set to receive output values.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_UNSUPPORTED_PARAM_TYPE`

  The conversion is not supported. Possibly the `buffer_type` value is invalid or is not one of the supported types.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

See the Example in Section 28.7.11.11, “mysql_stmt_fetch()”.

28.7.11.6 mysql_stmt_close()

```
bool mysql_stmt_close(MYSQL_STMT *stmt)
```

Description

Closes the prepared statement. `mysql_stmt_close()` also deallocates the statement handler pointed to by `stmt`, which at that point becomes invalid and should no longer be used. For a failed `mysql_stmt_close()` call, do not call `mysql_stmt_error()`, or `mysql_stmt_errno()`, or `mysql_stmt_sqlstate()` to obtain error information because `mysql_stmt_close()` makes the statement handler invalid. Call `mysql_error()`, `mysql_errno()`, or `mysql_sqlstate()` instead.

If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

See the Example in Section 28.7.11.10, “mysql_stmt_execute()”.

28.7.11.7 mysql_stmt_data_seek()

```
void mysql_stmt_data_seek(MYSQL_STMT *stmt, uint64_t offset)
```

Description

Seeks to an arbitrary row in a statement result set. The `offset` value is a row number and should be in the range from `0` to `mysql_stmt_num_rows(stmt)-1`.

This function requires that the statement result set structure contains the entire result of the last executed query, so `mysql_stmt_data_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

None.

Errors

None.

28.7.11.8 mysql_stmt_errno()

```
unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_errno()` returns the error code for the most recently invoked statement API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. Errors also are listed at Appendix B, *Errors, Error Codes, and Common Problems*.

If the failed statement API function was `mysql_stmt_close()`, do not call or `mysql_stmt_errno()` to obtain error information because `mysql_stmt_close()` makes the statement handler invalid. Call `mysql_errno()` instead.

Return Values

An error code value. Zero if no error occurred.

Errors

None.

28.7.11.9 mysql_stmt_error()

```
const char *mysql_stmt_error(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_error()` returns a null-terminated string containing the error message for the most recently invoked statement API function that can succeed or fail. An empty string (`""`) is returned if no error occurred. Either of these two tests can be used to check for an error:

```
if(*mysql_stmt_errno(stmt))
{
  // an error occurred
}
```

```
if (mysql_stmt_error(stmt)[0])
{
  // an error occurred
}
```

If the failed statement API function was `mysql_stmt_close()`, do not call `mysql_stmt_error()` to obtain error information because `mysql_stmt_close()` makes the statement handler invalid. Call `mysql_error()` instead.

The language of the client error messages may be changed by recompiling the MySQL client library. You can choose error messages in several different languages.

Return Values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

28.7.11.10 mysql_stmt_execute()

```
int mysql_stmt_execute(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_execute()` executes the prepared query associated with the statement handler. The currently bound parameter marker values are sent to server during this call, and the server replaces the markers with this newly supplied data.

Statement processing following `mysql_stmt_execute()` depends on the type of statement:

- For an `UPDATE`, `DELETE`, or `INSERT`, the number of changed, deleted, or inserted rows can be found by calling `mysql_stmt_affected_rows()`.
- For a statement such as `SELECT` that generates a result set, you must call `mysql_stmt_fetch()` to fetch the data prior to calling any other functions that result in query processing. For more information on how to fetch the results, refer to Section 28.7.11.11, “mysql_stmt_fetch()”.

  Do not following invocation of `mysql_stmt_execute()` with a call to `mysql_store_result()` or `mysql_use_result()`. Those functions are not intended for processing results from prepared statements.

For statements that generate a result set, you can request that `mysql_stmt_execute()` open a cursor for the statement by calling `mysql_stmt_attr_set()` before executing the statement. If you execute a statement multiple times, `mysql_stmt_execute()` closes any open cursor before opening a new one.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.10.3, “Caching of Prepared Statements and Stored Programs”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

The following example demonstrates how to create and populate a table using `mysql_stmt_init()`, `mysql_stmt_prepare()`, `mysql_stmt_param_count()`, `mysql_stmt_bind_param()`, `mysql_stmt_execute()`, and `mysql_stmt_affected_rows()`. The `mysql` variable is assumed to be a valid connection handler. For an example that shows how to retrieve data, see Section 28.7.11.11, “mysql_stmt_fetch()”.

```
#define STRING_SIZE 50

#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(col1 INT,\
                                                     col2 VARCHAR(40),\
                                                     col3 SMALLINT,\
                                                     col4 TIMESTAMP)"
#define INSERT_SAMPLE "INSERT INTO \
                       test_table(col1,col2,col3) \
                       VALUES(?,?,?)"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[3];
uint64_t      affected_rows;
int           param_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
unsigned long str_length;
bool          is_null;

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
  fprintf(stderr, " DROP TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
  fprintf(stderr, " CREATE TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; the server */
/*  sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
```

```
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Bind the data for all 3 parameters */

memset(bind, 0, sizeof(bind));

/* INTEGER PARAM */
/* This is a number type, so there is no need
   to specify buffer_length */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= 0;
bind[0].length= 0;

/* STRING PARAM */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= 0;
bind[1].length= &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
  fprintf(stderr, " mysql_stmt_bind_param() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Specify the data values for the first row */
int_data= 10;             /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string  */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
```

```
  fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %lu\n",
                (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Specify data values for second row,
   then re-execute the statement */
int_data= 1000;
strncpy(str_data, "
        The most popular Open Source database",
        STRING_SIZE);
str_length= strlen(str_data);
small_data= 1000;         /* smallint */
is_null= 0;               /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute, 2 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 2): %lu\n",
                (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Close the statement */
if (mysql_stmt_close(stmt))
{
  /* mysql_stmt_close() invalidates stmt, so call          */
  /* mysql_error(mysql) rather than mysql_stmt_error(stmt) */
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}
```

Note

For complete examples on the use of prepared statement functions, refer to the file `tests/mysql_client_test.c`. This file can be obtained from a MySQL source distribution or from the source repository (see Section 2.9, “Installing MySQL from Source”).

28.7.11.11 mysql_stmt_fetch()

```
int mysql_stmt_fetch(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_fetch()` returns the next row in the result set. It can be called only while the result set exists; that is, after a call to `mysql_stmt_execute()` for a statement such as `SELECT` that produces a result set.

`mysql_stmt_fetch()` returns row data using the buffers bound by `mysql_stmt_bind_result()`. It returns the data in those buffers for all the columns in the current row set and the lengths are returned to the `length` pointer. All columns must be bound by the application before it calls `mysql_stmt_fetch()`.

`mysql_stmt_fetch()` typically occurs within a loop, to ensure that all result set rows are fetched. For example:

```
int status;

while (1)
{
  status = mysql_stmt_fetch(stmt);

  if (status == 1 || status == MYSQL_NO_DATA)
    break;

  /* handle current row here */
}

/* if desired, handle status == 1 case and display error here */
```

By default, result sets are fetched unbuffered a row at a time from the server. To buffer the entire result set on the client, call `mysql_stmt_store_result()` after binding the data buffers and before calling `mysql_stmt_fetch()`.

If a fetched data value is a `NULL` value, the `*is_null` value of the corresponding `MYSQL_BIND` structure contains TRUE (1). Otherwise, the data and its length are returned in the `*buffer` and `*length` elements based on the buffer type specified by the application. Each numeric and temporal type has a fixed length, as listed in the following table. The length of the string types depends on the length of the actual data value, as indicated by `data_length`.

Type	Length
`MYSQL_TYPE_TINY`	1
`MYSQL_TYPE_SHORT`	2
`MYSQL_TYPE_LONG`	4
`MYSQL_TYPE_LONGLONG`	8
`MYSQL_TYPE_FLOAT`	4
`MYSQL_TYPE_DOUBLE`	8
`MYSQL_TYPE_TIME`	`sizeof(MYSQL_TIME)`
`MYSQL_TYPE_DATE`	`sizeof(MYSQL_TIME)`
`MYSQL_TYPE_DATETIME`	`sizeof(MYSQL_TIME)`
`MYSQL_TYPE_STRING`	`data length`
`MYSQL_TYPE_BLOB`	`data_length`

In some cases, you might want to determine the length of a column value before fetching it with `mysql_stmt_fetch()`. For example, the value might be a long string or `BLOB` value for which you want to know how much space must be allocated. To accomplish this, use one of these strategies:

- Before invoking `mysql_stmt_fetch()` to retrieve individual rows, pass `STMT_ATTR_UPDATE_MAX_LENGTH` to `mysql_stmt_attr_set()`, then invoke `mysql_stmt_store_result()` to buffer the entire result on the client side. Setting the `STMT_ATTR_UPDATE_MAX_LENGTH` attribute causes the maximal length of column values to be indicated by the `max_length` member of the result set metadata returned by `mysql_stmt_result_metadata()`.
- Invoke `mysql_stmt_fetch()` with a zero-length buffer for the column in question and a pointer in which the real length can be stored. Then use the real length with `mysql_stmt_fetch_column()`.

```
real_length= 0;

bind[0].buffer= 0;
bind[0].buffer_length= 0;
bind[0].length= &real_length
mysql_stmt_bind_result(stmt, bind);

mysql_stmt_fetch(stmt);
if (real_length > 0)
{
  data= malloc(real_length);
  bind[0].buffer= data;
  bind[0].buffer_length= real_length;
  mysql_stmt_fetch_column(stmt, bind, 0, 0);
}
```

Return Values

Return Value	Description
0	Success, the data has been fetched to application data buffers.
1	Error occurred. Error code and message can be obtained by calling `mysql_stmt_errno()` and `mysql_stmt_error()`.
`MYSQL_NO_DATA`	Success, no more data exists
`MYSQL_DATA_TRUNCATED`	Data truncation occurred

`MYSQL_DATA_TRUNCATED` is returned when truncation reporting is enabled. To determine which column values were truncated when this value is returned, check the `error` members of the `MYSQL_BIND` structures used for fetching values. Truncation reporting is enabled by default, but can be controlled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

  Although `mysql_stmt_fetch()` can produce this error, it is more likely to occur for the *following* C API call if `mysql_stmt_fetch()` is not called enough times to read the entire result set (that is, enough times to return `MYSQL_NO_DATA`).

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

- `CR_UNSUPPORTED_PARAM_TYPE`

  The buffer type is `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, but the data type is not `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP`.

- All other unsupported conversion errors are returned from `mysql_stmt_bind_result()`.

Example

The following example demonstrates how to fetch data from a table using `mysql_stmt_result_metadata()`, `mysql_stmt_bind_result()`, and `mysql_stmt_fetch()`. (This example expects to retrieve the two rows inserted by the example shown in Section 28.7.11.10, “mysql_stmt_execute()”.) The `mysql` variable is assumed to be a valid connection handler.

```
#define STRING_SIZE 50

#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 \
                       FROM test_table"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[4];
MYSQL_RES     *prepare_meta_result;
MYSQL_TIME    ts;
unsigned long length[4];
int           param_count, column_count, row_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
bool          is_null[4];
bool          error[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), SELECT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}
```

```
/* Fetch result set meta information */
prepare_meta_result = mysql_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
  fprintf(stderr,
         " mysql_stmt_result_metadata(), \
           returned no meta information\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get total columns in the query */
column_count= mysql_num_fields(prepare_meta_result);
fprintf(stdout,
        " total columns in SELECT statement: %d\n",
        column_count);

if (column_count != 4) /* validate column count */
{
  fprintf(stderr, " invalid column count returned by MySQL\n");
  exit(0);
}

/* Execute the SELECT query */
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */

memset(bind, 0, sizeof(bind));

/* INTEGER COLUMN */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];
bind[0].error= &error[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];
bind[1].error= &error[1];

/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];
bind[2].error= &error[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
bind[3].length= &length[3];
bind[3].error= &error[3];

/* Bind the result buffers */
if (mysql_stmt_bind_result(stmt, bind))
{
```

```
  fprintf(stderr, " mysql_stmt_bind_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Now buffer all results to client (optional step) */
if (mysql_stmt_store_result(stmt))
{
  fprintf(stderr, " mysql_stmt_store_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ...\n");
while (!mysql_stmt_fetch(stmt))
{
  row_count++;
  fprintf(stdout, "  row %d\n", row_count);

  /* column 1 */
  fprintf(stdout, "   column1 (integer)  : ");
  if (is_null[0])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

  /* column 2 */
  fprintf(stdout, "   column2 (string)   : ");
  if (is_null[1])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

  /* column 3 */
  fprintf(stdout, "   column3 (smallint) : ");
  if (is_null[2])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", small_data, length[2]);

  /* column 4 */
  fprintf(stdout, "   column4 (timestamp): ");
  if (is_null[3])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %04d-%02d-%02d %02d:%02d:%02d (%ld)\n",
                     ts.year, ts.month, ts.day,
                     ts.hour, ts.minute, ts.second,
                     length[3]);
  fprintf(stdout, "\n");
}

/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
  fprintf(stderr, " MySQL failed to return all rows\n");
  exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);


/* Close the statement */
```

```
if (mysql_stmt_close(stmt))
{
  /* mysql_stmt_close() invalidates stmt, so call          */
  /* mysql_error(mysql) rather than mysql_stmt_error(stmt) */
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}
```

28.7.11.12 mysql_stmt_fetch_column()

```
int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, unsigned int
column, unsigned long offset)
```

Description

Fetches one column from the current result set row. `bind` provides the buffer where data should be placed. It should be set up the same way as for `mysql_stmt_bind_result()`. `column` indicates which column to fetch. The first column is numbered 0. `offset` is the offset within the data value at which to begin retrieving data. This can be used for fetching the data value in pieces. The beginning of the value is offset 0.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_INVALID_PARAMETER_NO`

  Invalid column number.

- `CR_NO_DATA`

  The end of the result set has already been reached.

28.7.11.13 mysql_stmt_field_count()

```
unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)
```

Description

Returns the number of columns for the most recent statement for the statement handler. This value is zero for statements such as `INSERT` or `DELETE` that do not produce result sets.

`mysql_stmt_field_count()` can be called after you have prepared a statement by invoking `mysql_stmt_prepare()`.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

28.7.11.14 mysql_stmt_free_result()

```
bool mysql_stmt_free_result(MYSQL_STMT *stmt)
```

Description

Releases memory associated with the result set produced by execution of the prepared statement. If there is a cursor open for the statement, `mysql_stmt_free_result()` closes it.

Return Values

Zero for success. Nonzero if an error occurred.

28.7.11.15 mysql_stmt_init()

```
MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)
```

Description

Creates and returns a `MYSQL_STMT` handler. The handler should be freed with `mysql_stmt_close()`, at which point the handler becomes invalid and should no longer be used.

See also Section 28.7.9, “C API Prepared Statement Data Structures”, for more information.

Return Values

A pointer to a `MYSQL_STMT` structure in case of success. `NULL` if out of memory.

Errors

- `CR_OUT_OF_MEMORY`

  Out of memory.

28.7.11.16 mysql_stmt_insert_id()

```
uint64_t mysql_stmt_insert_id(MYSQL_STMT *stmt)
```

Description

Returns the value generated for an `AUTO_INCREMENT` column by the prepared `INSERT` or `UPDATE` statement. Use this function after you have executed a prepared `INSERT` statement on a table which contains an `AUTO_INCREMENT` field.

See Section 28.7.7.38, “mysql_insert_id()”, for more information.

Return Values

Value for `AUTO_INCREMENT` column which was automatically generated or explicitly set during execution of prepared statement, or value generated by `LAST_INSERT_ID(`*`expr`*`)` function. Return value is undefined if statement does not set `AUTO_INCREMENT` value.

Errors

None.

28.7.11.17 mysql_stmt_next_result()

```
int mysql_stmt_next_result(MYSQL_STMT *mysql)
```

Description

This function is used when you use prepared `CALL` statements to execute stored procedures, which can return multiple result sets. Use a loop that calls `mysql_stmt_next_result()` to determine whether there are more results. If a procedure has `OUT` or `INOUT` parameters, their values will be returned as a single-row result set following any other result sets. The values will appear in the order in which they are declared in the procedure parameter list.

For information about the effect of unhandled conditions on procedure parameters, see Section 13.6.7.8, “Condition Handling and OUT or INOUT Parameters”.

`mysql_stmt_next_result()` returns a status to indicate whether more results exist. If `mysql_stmt_next_result()` returns an error, there are no more results.

Before each call to `mysql_stmt_next_result()`, you must call `mysql_stmt_free_result()` for the current result if it produced a result set (rather than just a result status).

After calling `mysql_stmt_next_result()` the state of the connection is as if you had called `mysql_stmt_execute()`. This means that you can call `mysql_stmt_bind_result()`, `mysql_stmt_affected_rows()`, and so forth.

It is also possible to test whether there are more results by calling `mysql_more_results()`. However, this function does not change the connection state, so if it returns true, you must still call `mysql_stmt_next_result()` to advance to the next result.

For an example that shows how to use `mysql_stmt_next_result()`, see Section 28.7.25, “C API Prepared CALL Statement Support”.

Return Values

Return Value	Description
0	Successful and there are more results
-1	Successful and there are no more results
>0	An error occurred

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.11.18 mysql_stmt_num_rows()

```
uint64_t mysql_stmt_num_rows(MYSQL_STMT *stmt)
```

Description

Returns the number of rows in the result set.

The use of `mysql_stmt_num_rows()` depends on whether you used `mysql_stmt_store_result()` to buffer the entire result set in the statement handler. If you use `mysql_stmt_store_result()`, `mysql_stmt_num_rows()` may be called immediately. Otherwise, the row count is unavailable unless you count the rows as you fetch them.

`mysql_stmt_num_rows()` is intended for use with statements that return a result set, such as `SELECT`. For statements such as `INSERT`, `UPDATE`, or `DELETE`, the number of affected rows can be obtained with `mysql_stmt_affected_rows()`.

Return Values

The number of rows in the result set.

Errors

None.

28.7.11.19 mysql_stmt_param_count()

```
unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)
```

Description

Returns the number of parameter markers present in the prepared statement.

Return Values

An unsigned long integer representing the number of parameters in a statement.

Errors

None.

Example

See the Example in Section 28.7.11.10, “mysql_stmt_execute()”.

28.7.11.20 mysql_stmt_param_metadata()

```
MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)
```

This function currently does nothing.

28.7.11.21 mysql_stmt_prepare()

```
int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *stmt_str, unsigned long
length)
```

Description

Given the statement handler returned by `mysql_stmt_init()`, prepares the SQL statement pointed to by the string `stmt_str` and returns a status value. The string length should be given by the `length` argument. The string must consist of a single SQL statement. You should not add a terminating semicolon (`;`) or `\g` to the statement.

The application can include one or more parameter markers in the SQL statement by embedding question mark (`?`) characters into the SQL string at the appropriate positions.

The markers are legal only in certain places in SQL statements. For example, they are permitted in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value. However, they are not permitted for identifiers (such as table or column names), or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

The parameter markers must be bound to application variables using `mysql_stmt_bind_param()` before executing the statement.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.10.3, “Caching of Prepared Statements and Stored Programs”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

If the prepare operation was unsuccessful (that is, `mysql_stmt_prepare()` returns nonzero), the error message can be obtained by calling `mysql_stmt_error()`.

Example

See the Example in Section 28.7.11.10, “mysql_stmt_execute()”.

28.7.11.22 mysql_stmt_reset()

```
bool mysql_stmt_reset(MYSQL_STMT *stmt)
```

Description

Resets a prepared statement on client and server to state after prepare. It resets the statement on the server, data sent using `mysql_stmt_send_long_data()`, unbuffered result sets and current errors. It does not clear bindings or stored result sets. Stored result sets will be cleared when executing the prepared statement (or closing it).

To re-prepare the statement with another query, use `mysql_stmt_prepare()`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.11.23 mysql_stmt_result_metadata()

```
MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)
```

Description

If a statement passed to `mysql_stmt_prepare()` is one that produces a result set, `mysql_stmt_result_metadata()` returns the result set metadata in the form of a pointer to a `MYSQL_RES` structure that can be used to process the meta information such as number of fields and individual field information. This result set pointer can be passed as an argument to any of the field-based API functions that process result set metadata, such as:

- `mysql_num_fields()`
- `mysql_fetch_field()`
- `mysql_fetch_field_direct()`
- `mysql_fetch_fields()`
- `mysql_field_count()`

- `mysql_field_seek()`
- `mysql_field_tell()`
- `mysql_free_result()`

The result set structure should be freed when you are done with it, which you can do by passing it to `mysql_free_result()`. This is similar to the way you free a result set obtained from a call to `mysql_store_result()`.

The result set returned by `mysql_stmt_result_metadata()` contains only metadata. It does not contain any row results. The rows are obtained by using the statement handler with `mysql_stmt_fetch()`.

Return Values

A `MYSQL_RES` result structure. `NULL` if no meta information exists for the prepared query.

Errors

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

See the Example in Section 28.7.11.11, “mysql_stmt_fetch()”.

28.7.11.24 mysql_stmt_row_seek()

```
MYSQL_ROW_OFFSET mysql_stmt_row_seek(MYSQL_STMT *stmt, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a statement result set. The `offset` value is a row offset that should be a value returned from `mysql_stmt_row_tell()` or from `mysql_stmt_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_stmt_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_stmt_row_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_stmt_row_seek()`.

Errors

None.

28.7.11.25 mysql_stmt_row_tell()

```
MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)
```

Description

Returns the current position of the row cursor for the last `mysql_stmt_fetch()`. This value can be used as an argument to `mysql_stmt_row_seek()`.

You should use `mysql_stmt_row_tell()` only after `mysql_stmt_store_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

28.7.11.26 mysql_stmt_send_long_data()

```
bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int parameter_number,
const char *data, unsigned long length)
```

Description

Enables an application to send parameter data to the server in pieces (or “chunks”). Call this function after `mysql_stmt_bind_param()` and before `mysql_stmt_execute()`. It can be called multiple times to send the parts of a character or binary data value for a column, which must be one of the `TEXT` or `BLOB` data types.

`parameter_number` indicates which parameter to associate the data with. Parameters are numbered beginning with 0. `data` is a pointer to a buffer containing data to be sent, and `length` indicates the number of bytes in the buffer.

Note

The next `mysql_stmt_execute()` call ignores the bind buffer for all parameters that have been used with `mysql_stmt_send_long_data()` since last `mysql_stmt_execute()` or `mysql_stmt_reset()`.

If you want to reset/forget the sent data, you can do it with `mysql_stmt_reset()`. See Section 28.7.11.22, “mysql_stmt_reset()”.

The `max_allowed_packet` system variable controls the maximum size of parameter values that can be sent with `mysql_stmt_send_long_data()`.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_INVALID_BUFFER_USE`

  The parameter does not have a string or binary type.

- `CR_INVALID_PARAMETER_NO`

Invalid parameter number.

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

The following example demonstrates how to send the data for a `TEXT` column in chunks. It inserts the data value `'MySQL - The most popular Open Source database'` into the `text_column` column. The `mysql` variable is assumed to be a valid connection handler.

```
#define INSERT_QUERY "INSERT INTO \
                      test_long_data(text_column) VALUES(?)"

MYSQL_BIND bind[1];
long       length;

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
 memset(bind, 0, sizeof(bind));
 bind[0].buffer_type= MYSQL_TYPE_STRING;
 bind[0].length= &length;
 bind[0].is_null= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
  fprintf(stderr, "\n param bind failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply data in chunks to server */
 if (mysql_stmt_send_long_data(stmt,0,"MySQL",5))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
```

```
 /* Supply the next piece of data */
 if (mysql_stmt_send_long_data(stmt,0,
           " - The most popular Open Source database",40))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Now, execute the query */
 if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, "\n mysql_stmt_execute failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
```

28.7.11.27 mysql_stmt_sqlstate()

```
const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_sqlstate()` returns a null-terminated string containing the SQLSTATE error code for the most recently invoked prepared statement API function that can succeed or fail. The error code consists of five characters. `"00000"` means “no error.” The values are specified by ANSI SQL and ODBC. For a list of possible values, see Appendix B, *Errors, Error Codes, and Common Problems*.

Not all MySQL errors are mapped to SQLSTATE codes. The value `"HY000"` (general error) is used for unmapped errors.

If the failed statement API function was `mysql_stmt_close()`, do not call `mysql_stmt_sqlstate()` to obtain error information because `mysql_stmt_close()` makes the statement handler invalid. Call `mysql_sqlstate()` instead.

Return Values

A null-terminated character string containing the SQLSTATE error code.

28.7.11.28 mysql_stmt_store_result()

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```

Description

Result sets are produced by calling `mysql_stmt_execute()` to executed prepared statements for SQL statements such as `SELECT`, `SHOW`, `DESCRIBE`, and `EXPLAIN`. By default, result sets for successfully executed prepared statements are not buffered on the client and `mysql_stmt_fetch()` fetches them one at a time from the server. To cause the complete result set to be buffered on the client, call `mysql_stmt_store_result()` after binding data buffers with `mysql_stmt_bind_result()` and before calling `mysql_stmt_fetch()` to fetch rows. (For an example, see Section 28.7.11.11, “mysql_stmt_fetch()”.)

`mysql_stmt_store_result()` is optional for result set processing, unless you will call `mysql_stmt_data_seek()`, `mysql_stmt_row_seek()`, or `mysql_stmt_row_tell()`. Those functions require a seekable result set.

It is unnecessary to call `mysql_stmt_store_result()` after executing an SQL statement that does not produce a result set, but if you do, it does not harm or cause any notable performance problem. You can detect whether the statement produced a result set by checking if `mysql_stmt_result_metadata()` returns `NULL`. For more information, refer to Section 28.7.11.23, “mysql_stmt_result_metadata()”.

Note

MySQL does not by default calculate `MYSQL_FIELD->max_length` for all columns in `mysql_stmt_store_result()` because calculating this would slow down `mysql_stmt_store_result()` considerably and most applications do not need `max_length`. If you want `max_length` to be updated, you can call `mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)` to enable this. See Section 28.7.11.3, “mysql_stmt_attr_set()”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

28.7.12 C API Asynchronous Interface

As of MySQL 8.0.16, the C API includes asynchronous functions that enable nonblocking communication with the MySQL server. Asynchronous functions enable development of applications that differ from the query processing model based on synchronous functions that block if reads from or writes to the server connection must wait. Using the asynchronous functions, an application can check whether work on the server connection is ready to proceed. If not, the application can perform other work before checking again later.

For example, an application might open multiple connections to the server and use them to submit multiple statements for execution. The application then can poll the connections to see which of them have results to be fetched, while doing other work.

The asynchronous C API functions cover operations that might otherwise block when reading to or writing from the server connection: The initial connection operation, sending a query, reading the result, and so forth. Each asynchronous function has the same name as its synchronous counterpart, plus a `_nonblocking` suffix:

- `mysql_fetch_row_nonblocking()`: Asynchronously fetches the next row from the result set.
- `mysql_free_result_nonblocking()`: Asynchronously frees memory used by a result set.
- `mysql_next_result_nonblocking()`: Asynchronously returns/initiates the next result in multiple-result executions.
- `mysql_real_connect_nonblocking()`: Asynchronously connects to a MySQL server.
- `mysql_real_query_nonblocking()`: Asynchronously executes an SQL query specified as a counted string.
- `mysql_store_result_nonblocking()`: Asynchronously retrieves a complete result set to the client.

Applications can mix asynchronous and synchronous functions if there are operations that need not be done asynchronously or for which the asynchronous functions do not apply.

The following discussion describes in more detail how to use asynchronous C API functions.

- Asynchronous Function Calling Conventions
- Example Program
- Asynchronous Function Restrictions

Asynchronous Function Calling Conventions

All asynchronous C API functions return an `enum net_async_status` value. The return value can be one of the following values to indicate operation status:

- `NET_ASYNC_NOT_READY`: The operation is still in progress and not yet complete.
- `NET_ASYNC_COMPLETE`: The operation completed successfully.
- `NET_ASYNC_ERROR`: The operation terminated in error.
- `NET_ASYNC_COMPLETE_NO_MORE_RESULTS`: The operation completed successfully and no more results are available. This status applies only to `mysql_next_result_nonblocking()`.

In general, to use an asynchronous function, do this:

- Call the function repeatedly until it no longer returns a status of `NET_ASYNC_NOT_READY`.
- Check whether the final status indicates successful completion (`NET_ASYNC_COMPLETE`) or an error (`NET_ASYNC_ERROR`).

The following examples illustrate some typical calling patterns. *`function(args)`* represents an asychronous function and its argument list.

- If it is desirable to perform other processing while the operation is in progress:

```
enum net_async_status status;

status = function(args);
while (status == NET_ASYNC_NOT_READY) {
  /* perform other processing */
  other_processing ();
  /* invoke same function and arguments again */
```

```
  status = function(args);
}
if (status == NET_ASYNC_ERROR) {
  /* call failed; handle error */
} else {
  /* call successful; handle result */
}
```

- If there is no need to perform other processing while the operation is in progress:

```
enum net_async_status status;

while ((status = function(args)) == NET_ASYNC_NOT_READY)
  ; /* empty loop */
if (status == NET_ASYNC_ERROR) {
  /* call failed; handle error */
} else {
  /* call successful; handle result */
}
```

- If the function success/failure result does not matter and you want to ensure only that the operation has completed:

```
while (function (args) != NET_ASYNC_COMPLETE)
  ; /* empty loop */
```

For `mysql_next_result_nonblocking()`, it is also necessary to account for the `NET_ASYNC_COMPLETE_NO_MORE_RESULTS` status, which indicates that the operation completed successfully and no more results are available. Use it like this:

```
while ((status = mysql_next_result_nonblocking()) != NET_ASYNC_COMPLETE) {
  if (status == NET_ASYNC_COMPLETE_NO_MORE_RESULTS) {
    /* no more results */
  }
  else if (status == NET_ASYNC_ERROR) {
    /* handle error by calling mysql_error(); */
    break;
  }
}
```

In most cases, arguments for the asynchronous functions are the same as for the corresponding synchronous functions. Exceptions are `mysql_fetch_row_nonblocking()` and `mysql_store_result_nonblocking()`, each of which takes an extra argument compared to its synchronous counterpart. For details, see Section 28.7.15.1, “mysql_fetch_row_nonblocking()”, and Section 28.7.15.6, “mysql_store_result_nonblocking()”.

Example Program

This section shows an example C++ program that illustrates use of asynchronous C API functions.

To set up the SQL objects used by the program, execute the following statements. Substitute a different database or user as desired; in this case, you will need to make some adjustments to the program as well.

```
CREATE DATABASE db;
USE db;
CREATE TABLE test_table (id INT NOT NULL);
INSERT INTO test_table VALUES (10), (20), (30);

CREATE USER 'testuser'@'localhost' IDENTIFIED BY 'testpass';
```

```
GRANT ALL ON db.* TO 'testuser'@'localhost';
```

Create a file named `async_app.cc` containing the following program. Adjust the connection parameters as necessary.

```
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <mysql.h>
#include <mysqld_error.h>

using namespace std;

/* change following connection parameters as necessary */
static const char * c_host = "localhost";
static const char * c_user = "testuser";
static const char * c_auth = "testpass";
static int          c_port = 3306;
static const char * c_sock = "/usr/local/mysql/mysql.sock";
static const char * c_dbnm = "db";

void perform_arithmetic() {
  cout<<"dummy function invoked\n";
  for (int i = 0; i < 1000; i++)
    i*i;
}

int main(int argc, char ** argv)
{
  MYSQL *mysql_local;
  MYSQL_RES *result;
  MYSQL_ROW row;
  net_async_status status;
  const char *stmt_text;

  if (!(mysql_local = mysql_init(NULL))) {
    cout<<"mysql_init() failed\n";
    exit(1);
  }
  while ((status = mysql_real_connect_nonblocking(mysql_local, c_host, c_user,
                                                  c_auth, c_dbnm, c_port,
                                                  c_sock, 0))
           == NET_ASYNC_NOT_READY)
    ; /* empty loop */
  if (status == NET_ASYNC_ERROR) {
    cout<<"mysql_real_connect_nonblocking() failed\n";
    exit(1);
  }

  /* run query asynchronously */
  stmt_text = "SELECT * FROM test_table ORDER BY id";
  status = mysql_real_query_nonblocking(mysql_local, stmt_text,
                                        (unsigned long)strlen(stmt_text));
  /* do some other task before checking function result */
  perform_arithmetic();
  while (status == NET_ASYNC_NOT_READY) {
    status = mysql_real_query_nonblocking(mysql_local, stmt_text,
                                          (unsigned long)strlen(stmt_text));
    perform_arithmetic();
  }
  if (status == NET_ASYNC_ERROR) {
    cout<<"mysql_real_query_nonblocking() failed\n";
    exit(1);
  }

  /* retrieve query result asynchronously */
```

```
  status = mysql_store_result_nonblocking(mysql_local, &result);
  /* do some other task before checking function result */
  perform_arithmetic();
  while (status == NET_ASYNC_NOT_READY) {
    status = mysql_store_result_nonblocking(mysql_local, &result);
    perform_arithmetic();
  }
  if (status == NET_ASYNC_ERROR) {
    cout<<"mysql_store_result_nonblocking() failed\n";
    exit(1);
  }
  if (result == NULL) {
    cout<<"mysql_store_result_nonblocking() found 0 records\n";
    exit(1);
  }

  /* fetch a row synchronously */
  row = mysql_fetch_row(result);
  if (row != NULL && strcmp(row[0], "10") == 0)
    cout<<"ROW: " << row[0] << "\n";
  else
    cout<<"incorrect result fetched\n";

  /* fetch a row asynchronously, but without doing other work */
  while (mysql_fetch_row_nonblocking(result, &row) != NET_ASYNC_COMPLETE)
    ; /* empty loop */
  /* 2nd row fetched */
  if (row != NULL && strcmp(row[0], "20") == 0)
    cout<<"ROW: " << row[0] << "\n";
  else
    cout<<"incorrect result fetched\n";

  /* fetch a row asynchronously, doing other work while waiting */
  status = mysql_fetch_row_nonblocking(result, &row);
  /* do some other task before checking function result */
  perform_arithmetic();
  while (status != NET_ASYNC_COMPLETE) {
    status = mysql_fetch_row_nonblocking(result, &row);
    perform_arithmetic();
  }
  /* 3rd row fetched */
  if (row != NULL && strcmp(row[0], "30") == 0)
    cout<<"ROW: " << row[0] << "\n";
  else
    cout<<"incorrect result fetched\n";

  /* fetch a row asynchronously (no more rows expected) */
  while ((status = mysql_fetch_row_nonblocking(result, &row))
           != NET_ASYNC_COMPLETE)
    ; /* empty loop */
  if (row == NULL)
    cout <<"No more rows to process.\n";
  else
    cout <<"More rows found than expected.\n";

  /* free result set memory asynchronously */
  while (mysql_free_result_nonblocking(result) != NET_ASYNC_COMPLETE)
    ; /* empty loop */

  mysql_close(mysql_local);
}
```

Compile the program using a command similar to this; adjust the compiler and options as necessary:

```
gcc -g async_app.cc -std=c++11 \
  -I/usr/local/mysql/include \
```

```
  -o async_app -L/usr/lib64/ -lstdc++ \
  -L/usr/local/mysql/lib/ -lmysqlclient
```

Run the program. The results should be similar to what you see here, although you might see a varying number of `dummy function invoked` instances.

```
dummy function invoked
dummy function invoked
ROW: 10
ROW: 20
dummy function invoked
ROW: 30
No more rows to process.
```

To experiment with the program, add and remove rows from `test_table`, running the program again after each change.

Asynchronous Function Restrictions

These restrictions apply to the use of asynchronous C API functions:

- `mysql_real_connect_nonblocking()` can be used only for accounts that authenticate with one of these authentication plugins: `mysql_native_password`, `sha256_password`, or `caching_sha2_password`.
- `mysql_real_connect_nonblocking()` can be used only to establish TCP/IP or Unix socket file connections.
- These statements are not supported and must be processed using synchronous C API functions: `LOAD DATA`, `LOAD XML`.
- Protocol compression is not supported for asynchronous C API functions.

28.7.13 C API Asynchronous Interface Data Structures

This section describes data structures specific to asynchronous C API functions. For information about general-purpose C API data structures, see Section 28.7.5, “C API Data Structures”.

- `enum net_async_status`

  The enumeration type used to express the return status of asynchronous C API functions. The following table shows the permitted status values.

Enumeration Status Value	Description
`NET_ASYNC_COMPLETE`	Asynchronous operation is complete
`NET_ASYNC_NOT_READY`	Asynchronous operation is still in progress
`NET_ASYNC_ERROR`	Asynchronous operation terminated in error
`NET_ASYNC_COMPLETE_NO_MORE_RESULTS`	For `mysql_next_result_nonblocking()`; indicates no more results available

  For more information, see Section 28.7.12, “C API Asynchronous Interface”.

28.7.14 C API Asynchronous Function Overview

The following list summarizes the functions available for asynchronous interaction with the MySQL server. For greater detail, see the descriptions in Section 28.7.15, “C API Asynchronous Function Descriptions”.

- `mysql_fetch_row_nonblocking()`: Asynchronously fetches the next row from the result set.
- `mysql_free_result_nonblocking()`: Asynchronously frees memory used by a result set.
- `mysql_next_result_nonblocking()`: Asynchronously returns/initiates the next result in multiple-result executions.
- `mysql_real_connect_nonblocking()`: Asynchronously connects to a MySQL server.
- `mysql_real_query_nonblocking()`: Asynchronously executes an SQL query specified as a counted string.
- `mysql_store_result_nonblocking()`: Asynchronously retrieves a complete result set to the client.

28.7.15 C API Asynchronous Function Descriptions

To interact asynchronously with the MySQL server, use the functions described in the following sections. For descriptions of their synchronous counterparts, see Section 28.7.7, “C API Function Descriptions”.

28.7.15.1 mysql_fetch_row_nonblocking()

```
enum net_async_status mysql_fetch_row_nonblocking(MYSQL_RES *result, MYSQL_ROW
*row)
```

Description

Note

`mysql_fetch_row_nonblocking()` is an asynchronous function. It is the counterpart of the `mysql_fetch_row()` synchronous function, for use by applications that require asynchronous communication with the server. For general information about writing asynchronous C API applications, see Section 28.7.12, “C API Asynchronous Interface”.

`mysql_fetch_row_nonblocking()` is used similarly to `mysql_fetch_row()`. For details about the latter, see Section 28.7.7.21, “mysql_fetch_row()”. The two functions differ as follows:

- `mysql_fetch_row()` returns a `MYSQL_ROW` value containing the next row, or `NULL`. The meaning of a `NULL` return depends on which function was called preceding `mysql_fetch_row()`:
  - When used after `mysql_store_result()` or `mysql_store_result_nonblocking()`, `mysql_fetch_row()` returns `NULL` if there are no more rows to retrieve.
  - When used after `mysql_use_result()`, `mysql_fetch_row()` returns `NULL` if there are no more rows to retrieve or an error occurred.
- `mysql_fetch_row_nonblocking()` returns an `enum net_async_status` status indicator and takes a second `row` argument that provides a pointer to a `MYSQL_ROW` value. When the return status is `NET_ASYNC_COMPLETE`, the `row` argument is a pointer to a `MYSQL_ROW` value containing the next row, or `NULL`. The meaning of `NULL` depends on which function was called preceding `mysql_fetch_row_nonblocking()`:
  - When used after `mysql_store_result()` or `mysql_store_result_nonblocking()`, the `row` argument is `NULL` if there are no more rows to retrieve.
  -

When used after `mysql_use_result()`, the `row` argument is `NULL` if there are no more rows to retrieve or an error occurred.

`mysql_fetch_row_nonblocking()` was added in MySQL 8.0.16.

Return Values

Returns an `enum net_async_status` value. See the description in Section 28.7.13, “C API Asynchronous Interface Data Structures”. A `NET_ASYNC_ERROR` return status indicates an error.

Example

See Section 28.7.12, “C API Asynchronous Interface”.

28.7.15.2 mysql_free_result_nonblocking()

```
enum net_async_status mysql_free_result_nonblocking(MYSQL_RES *result)
```

Description

Note

`mysql_free_result_nonblocking()` is an asynchronous function. It is the counterpart of the `mysql_free_result()` synchronous function, for use by applications that require asynchronous communication with the server. For general information about writing asynchronous C API applications, see Section 28.7.12, “C API Asynchronous Interface”.

`mysql_free_result_nonblocking()` is used similarly to `mysql_free_result()`. For details about the latter, see Section 28.7.7.25, “mysql_free_result()”. The two functions differ as follows:

- `mysql_free_result()` does not return a value.
- `mysql_free_result_nonblocking()` returns an `enum net_async_status` status indicator.

`mysql_free_result_nonblocking()` was added in MySQL 8.0.16.

Return Values

Returns an `enum net_async_status` value. See the description in Section 28.7.13, “C API Asynchronous Interface Data Structures”. A `NET_ASYNC_ERROR` return status indicates an error.

Example

See Section 28.7.12, “C API Asynchronous Interface”.

28.7.15.3 mysql_next_result_nonblocking()

```
enum net_async_status mysql_next_result_nonblocking(MYSQL *mysql)
```

Description

Note

`mysql_next_result_nonblocking()` is an asynchronous function. It is the counterpart of the `mysql_next_result()` synchronous function, for use by applications that require asynchronous communication with the server. For general

> information about writing asynchronous C API applications, see Section 28.7.12, “C API Asynchronous Interface”.

`mysql_next_result_nonblocking()` is used similarly to `mysql_next_result()`. For details about the latter, see Section 28.7.7.47, “mysql_next_result()”. The two functions differ as follows:

- `mysql_next_result()` returns an integer status indicator.
- `mysql_next_result_nonblocking()` returns an `enum net_async_status` status indicator.

`mysql_next_result_nonblocking()` was added in MySQL 8.0.16.

Return Values

Returns an `enum net_async_status` value. See the description in Section 28.7.13, “C API Asynchronous Interface Data Structures”. A `NET_ASYNC_COMPLETE_NO_MORE_RESULTS` return status indicates there are no more results available. A `NET_ASYNC_ERROR` return status indicates an error.

Example

See Section 28.7.12, “C API Asynchronous Interface”.

28.7.15.4 mysql_real_connect_nonblocking()

```
enum net_async_status mysql_real_connect_nonblocking(MYSQL *mysql, const char
*host, const char *user, const char *passwd, const char *db, unsigned int port,
const char *unix_socket, unsigned long client_flag)
```

Description

Note

`mysql_real_connect_nonblocking()` is an asynchronous function. It is the counterpart of the `mysql_real_connect()` synchronous function, for use by applications that require asynchronous communication with the server. For general information about writing asynchronous C API applications, see Section 28.7.12, “C API Asynchronous Interface”.

`mysql_real_connect_nonblocking()` is used similarly to `mysql_real_connect()`. For details about the latter, see Section 28.7.7.54, “mysql_real_connect()”. The two functions differ as follows:

- `mysql_real_connect()` returns a connection handler or `NULL`.
- `mysql_real_connect_nonblocking()` returns an `enum net_async_status` status indicator.

`mysql_real_connect_nonblocking()` was added in MySQL 8.0.16.

Return Values

Returns an `enum net_async_status` value. See the description in Section 28.7.13, “C API Asynchronous Interface Data Structures”. A `NET_ASYNC_ERROR` return status indicates an error.

Example

See Section 28.7.12, “C API Asynchronous Interface”.

28.7.15.5 mysql_real_query_nonblocking()

```
enum net_async_status mysql_real_query_nonblocking(MYSQL *mysql, const char
*stmt_str, unsigned long length)
```

Description

Note

`mysql_real_query_nonblocking()` is an asynchronous function. It is the counterpart of the `mysql_real_query()` synchronous function, for use by applications that require asynchronous communication with the server. For general information about writing asynchronous C API applications, see Section 28.7.12, “C API Asynchronous Interface”.

`mysql_real_query_nonblocking()` is used similarly to `mysql_real_query()`. For details about the latter, see Section 28.7.7.57, “mysql_real_query()”. The two functions differ as follows:

- `mysql_real_query()` returns an integer status indicator.
- `mysql_real_query_nonblocking()` returns an `enum net_async_status` status indicator.

`mysql_real_query_nonblocking()` was added in MySQL 8.0.16.

Return Values

Returns an `enum net_async_status` value. See the description in Section 28.7.13, “C API Asynchronous Interface Data Structures”. A `NET_ASYNC_ERROR` return status indicates an error.

Example

See Section 28.7.12, “C API Asynchronous Interface”.

28.7.15.6 mysql_store_result_nonblocking()

```
enum net_async_status mysql_store_result_nonblocking(MYSQL *mysql, MYSQL_RES
**result )
```

Description

Note

`mysql_store_result_nonblocking()` is an asynchronous function. It is the counterpart of the `mysql_store_result()` synchronous function, for use by applications that require asynchronous communication with the server. For general information about writing asynchronous C API applications, see Section 28.7.12, “C API Asynchronous Interface”.

`mysql_store_result_nonblocking()` is used similarly to `mysql_store_result()`. For details about the latter, see Section 28.7.7.79, “mysql_store_result()”. The two functions differ as follows:

- `mysql_store_result()` returns a pointer to a `MYSQL_RESULT` value that contains the result set, or `NULL` if there is no result set or an error occurred.
- `mysql_store_result_nonblocking()` returns an `enum net_async_status` status indicator and takes a second `result` argument that is the address of a pointer to a `MYSQL_RESULT` into which to store the result set. When the return status is `NET_ASYNC_COMPLETE`, the `result` argument is `NULL` if there is no result set or an error occurred.

`mysql_store_result_nonblocking()` was added in MySQL 8.0.16.

Return Values

Returns an `enum net_async_status` value. See the description in Section 28.7.13, “C API Asynchronous Interface Data Structures”. A `NET_ASYNC_ERROR` return status indicates an error.

When the return status is `NET_ASYNC_COMPLETE`, the `result` argument is `NULL` if there is no result set or an error occurred. To determine whether an error occurred, check whether `mysql_error()` returns a nonempty string, `mysql_errno()` returns nonzero, or `mysql_field_count()` returns zero.

Example

See Section 28.7.12, “C API Asynchronous Interface”.

28.7.16 C API Threaded Function Descriptions

To create a threaded client, use the functions described in the following sections. See also Section 28.7.4.3, “Writing C API Threaded Client Programs”.

28.7.16.1 mysql_thread_end()

```
void mysql_thread_end(void)
```

Description

Call this function before calling `pthread_exit()` to free memory allocated by `mysql_thread_init()`.

`mysql_thread_end()` *is not invoked automatically by the client library*.

- For release/production builds without debugging support enabled, `mysql_thread_end()` need not be called.
- For debug builds, `mysql_thread_init()` allocates debugging information for the DBUG package (see Section 29.5.4, “The DBUG Package”). `mysql_thread_end()` must be called for each `mysql_thread_init()` call to avoid a memory leak.

Return Values

None.

28.7.16.2 mysql_thread_init()

```
bool mysql_thread_init(void)
```

Description

This function must be called early within each created thread to initialize thread-specific variables. However, you may not necessarily need to invoke it explicitly: calling `mysql_thread_init()` is automatically handled by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()`, and `mysql_connect()`. If you invoke any of those functions, `mysql_thread_init()` is called for you.

Return Values

Zero for success. Nonzero if an error occurred.

28.7.16.3 mysql_thread_safe()

```
unsigned int mysql_thread_safe(void)
```

Description

This function indicates whether the client library is compiled as thread-safe.

Return Values

1 if the client library is thread-safe, 0 otherwise.

28.7.17 C API Client Plugin Functions

This section describes functions used for the client-side plugin API. They enable management of client plugins. For a description of the `st_mysql_client_plugin` structure used by these functions, see Client Plugin Descriptors.

It is unlikely that a client program needs to call the functions in this section. For example, a client that supports the use of authentication plugins normally causes a plugin to be loaded by calling `mysql_options()` to set the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options:

```
char *plugin_dir = "path_to_plugin_dir";
char *default_auth = "plugin_name";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
mysql_options(&mysql, MYSQL_DEFAULT_AUTH, default_auth);
```

Typically, the program will also accept `--plugin-dir` and `--default-auth` options that enable users to override the default values.

28.7.17.1 mysql_client_find_plugin()

```
struct st_mysql_client_plugin *mysql_client_find_plugin(MYSQL *mysql, const
char *name, int type)
```

Description

Returns a pointer to a loaded plugin, loading the plugin first if necessary. An error occurs if the type is invalid or the plugin cannot be found or loaded.

Specify the parameters as follows:

- `mysql`: A pointer to a `MYSQL` structure. The plugin API does not require a connection to a MySQL server, but this structure must be properly initialized. The structure is used to obtain connection-related information.
- `name`: The plugin name.
- `type`: The plugin type.

Return Values

A pointer to the plugin for success. `NULL` if an error occurred.

Errors

To check for errors, call the `mysql_error()` or `mysql_errno()` function. See Section 28.7.7.15, “mysql_error()”, and Section 28.7.7.14, “mysql_errno()”.

Example

```
MYSQL mysql;
struct st_mysql_client_plugin *p;

if ((p = mysql_client_find_plugin(&mysql, "myplugin",
                                  MYSQL_CLIENT_AUTHENTICATION_PLUGIN, 0)))
{
  printf("Plugin version: %d.%d.%d\n", p->version[0], p->version[1], p->version[2]);
}
```

28.7.17.2 mysql_client_register_plugin()

```
struct st_mysql_client_plugin *mysql_client_register_plugin(MYSQL *mysql,
struct st_mysql_client_plugin *plugin)
```

Description

Adds a plugin structure to the list of loaded plugins. An error occurs if the plugin is already loaded.

Specify the parameters as follows:

- `mysql`: A pointer to a `MYSQL` structure. The plugin API does not require a connection to a MySQL server, but this structure must be properly initialized. The structure is used to obtain connection-related information.
- `plugin`: A pointer to the plugin structure.

Return Values

A pointer to the plugin for success. `NULL` if an error occurred.

Errors

To check for errors, call the `mysql_error()` or `mysql_errno()` function. See Section 28.7.7.15, “mysql_error()”, and Section 28.7.7.14, “mysql_errno()”.

28.7.17.3 mysql_load_plugin()

```
struct st_mysql_client_plugin *mysql_load_plugin(MYSQL *mysql, const char
*name, int type, int argc, ...)
```

Description

Loads a MySQL client plugin, specified by name and type. An error occurs if the type is invalid or the plugin cannot be loaded.

It is not possible to load multiple plugins of the same type. An error occurs if you try to load a plugin of a type already loaded.

Specify the parameters as follows:

- `mysql`: A pointer to a `MYSQL` structure. The plugin API does not require a connection to a MySQL server, but this structure must be properly initialized. The structure is used to obtain connection-related information.
- `name`: The name of the plugin to load.
- `type`: The type of plugin to load, or −1 to disable type checking. If type is not −1, only plugins matching the type are considered for loading.
- `argc`: The number of following arguments (0 if there are none). Interpretation of any following arguments depends on the plugin type.

Another way to cause plugins to be loaded is to set the `LIBMYSQL_PLUGINS` environment variable to a list of semicolon-separated plugin names. For example:

```
shell> export LIBMYSQL_PLUGINS="myplugin1;myplugin2"
```

Plugins named by `LIBMYSQL_PLUGINS` are loaded when the client program calls `mysql_library_init()`. No error is reported if problems occur loading these plugins.

The `LIBMYSQL_PLUGIN_DIR` environment variable can be set to the path name of the directory in which to look for client plugins. This variable is used in two ways:

- During client plugin preloading, the value of the `--plugin-dir` option is not available, so client plugin loading fails unless the plugins are located in the hardwired default directory. If the plugins are located elsewhere, `LIBMYSQL_PLUGIN_DIR` environment variable can be set to the proper directory to enable plugin preloading to succeed.
- For explicit client plugin loading, the `mysql_load_plugin()` and `mysql_load_plugin_v()` C API functions use the `LIBMYSQL_PLUGIN_DIR` value if it exists and the `--plugin-dir` option was not given. If `--plugin-dir` is given, `mysql_load_plugin()` and `mysql_load_plugin_v()` ignore `LIBMYSQL_PLUGIN_DIR`.

Return Values

A pointer to the plugin if it was loaded successfully. `NULL` if an error occurred.

Errors

To check for errors, call the `mysql_error()` or `mysql_errno()` function. See Section 28.7.7.15, “mysql_error()”, and Section 28.7.7.14, “mysql_errno()”.

Example

```
MYSQL mysql;

if(!mysql_load_plugin(&mysql, "myplugin",
                      MYSQL_CLIENT_AUTHENTICATION_PLUGIN, 0))
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
    exit(-1);
}
```

See Also

See also Section 28.7.17.3, “mysql_load_plugin()”, Section 28.7.7.15, “mysql_error()”, Section 28.7.7.14, “mysql_errno()”.

28.7.17.4 mysql_load_plugin_v()

```
struct st_mysql_client_plugin *mysql_load_plugin_v(MYSQL *mysql, const char
*name, int type, int argc, va_list args)
```

Description

This function is equivalent to `mysql_load_plugin()`, but it accepts a `va_list` instead of a variable list of parameters.

See Also

See also Section 28.7.17.3, “mysql_load_plugin()”.

28.7.17.5 mysql_plugin_options()

```
int mysql_plugin_options(struct st_mysql_client_plugin *plugin, const char
*option, const void *value)
```

Description

Passes an option type and value to a plugin. This function can be called multiple times to set several options. If the plugin does not have an option handler, an error occurs.

Specify the parameters as follows:

- `plugin`: A pointer to the plugin structure.
- `option`: The option to be set.
- `value`: A pointer to the option value.

Return Values

Zero for success, 1 if an error occurred. If the plugin has an option handler, that handler should also return zero for success and 1 if an error occurred.

28.7.18 C API Binary Log Interface

The MySQL client/server protocol includes a client interface for reading a stream of replication events from a MySQL server binary log. This capability uses the `MYSQL_RPL` data structure and a small set of functions to manage communication between a client program and the server from which the binary log is to be read. The following sections describe aspects of this interface in more detail.

28.7.19 C API Binary Log Data Structures

C API functions for processing a replication event stream from a server require a connection handler (a `MYSQL *` pointer) and a pointer to a `MYSQL_RPL` structure that describes the steam of replication events to read from the server binary log. For example:

```
MYSQL *mysql = mysql_real_connect(...);

MYSQL_RPL rpl;

# ... initialize MYSQL_RPL members ...
```

```
int result = mysql_binlog_open(mysql, &rpl);
```

This section describes the `MYSQL_RPL` structure members. Connection handlers are described in Section 28.7.5, “C API Data Structures”.

The applicable `MYSQL_RPL` members depend on the binary log operation to be performed:

- Before calling `mysql_binlog_open()`, the caller must set the `MYSQL_RPL` members from `file_name_length` through `flags`. In addition, if `flags` has the `MYSQL_RPL_GTID` flag set, the caller must set the members from `gtid_set_encoded_size` through `gtid_set_arg`.
- After a successful `mysql_binlog_fetch()` call, the caller examines the `size` and `buffer` members.

`MYSQL_RPL` structure member descriptions:

- `file_name_length`

  The length of the name of the binary log file to read. This member is used in conjunction with `file_name`; see the `file_name` description.

- `file_name`

  The name of the binary log file to read:

  - If `file_name` is `NULL`, the client library sets it to the empty string and sets `file_name_length` to 0.
  - If `file_name` is not `NULL`, `file_name_length` must either be the length of the name or 0. If `file_name_length` is 0, the client library sets it to the length of the name, in which case, `file_name` must be given as a null-terminated string.

  To read from the beginning of the binary log without having to know the name of the oldest binary log file, set `file_name` to `NULL` or the empty string, and `start_position` to 4.

- `start_position`

  The position at which to start reading the binary log. The position of the first event in any given binary log file is 4.

- `server_id`

  The server ID to use for identifying to the server from which the binary log is read.

- `flags`

  The union of flags that affect binary log reading, or 0 if no flags are set. These flag values are permitted:

  - `MYSQL_RPL_SKIP_HEARTBEAT`

    Set this flag to cause `mysql_binlog_fetch()` to skip heartbeat events.

  - `MYSQL_RPL_GTID`

    Set this flag to read GTID (global transaction ID) data. If set, you must initialize the `MYSQL_RPL` structure GTID-related members from `gtid_set_encoded_size` to `gtid_set_arg` before calling `mysql_binlog_open()`.

    It is beyond the scope of this documentation to describe in detail how client programs use those GTID-related members. For more information, examine the `mysqlbinlog.cc` source file. For information about GTID-based replication, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

- `gtid_set_encoded_size`

  The size of GTID set data, or 0.

- `fix_gtid_set`

  The address of a callback function for `mysql_binlog_open()` to call to fill the command packet GTID set, or `NULL` if there is no such function. The callback function, if used, should have this calling signature:

```
void my_callback(MYSQL_RPL *rpl, unsigned char *packet_gtid_set);
```

- `gtid_set_arg`

  Either a pointer to GTID set data (if `fix_gtid_set` is `NULL`), or a pointer to a value to be made available for use within the callback function (if `fix_gtid_set` is not `NULL`). `gtid_set_arg` is a generic pointer, so it can point to any kind of value (for example, a string, a structure, or a function). Its interpretation within the callback depends on how the callback intends to use it.

- `size`

  After a successful `mysql_binlog_fetch()` call, the size of the returned binary log event. The value is 0 for an EOF event, greater than 0 for a non-EOF event.

- `buffer`

  After a successful `mysql_binlog_fetch()` call, a pointer to the binary log event contents.

28.7.20 C API Binary Log Function Overview

The following list summarizes the functions available for reading a replication event stream from a binary log. For greater detail, see the descriptions in Section 28.7.21, “C API Binary Log Function Descriptions”.

- `mysql_binlog_close()`: Closes the replication event stream.
- `mysql_binlog_fetch()`: Reads an event from the replication event stream.
- `mysql_binlog_open()`: Opens the replication event stream.

28.7.21 C API Binary Log Function Descriptions

The following sections provide detailed descriptions of the functions that enable reading the stream of replication events from a MySQL server binary log.

The following simple example program demonstrates the binary log C API functions. Program notes:

- `mysql` is assumed to be a valid connection handler.
- The initial `SET` statement sets the `@master_binlog_checksum` user-defined variable that the server takes as an indication that the client is checksum-aware. This client does nothing with checksums, but without this statement, a server that includes checksums in binary log events will return an error for the first attempt to read an event containing a checksum. The value assigned to the variable is immaterial; what matters is that the variable exist. existence.

```
if (mysql_query(mysql, "SET @master_binlog_checksum='ALL'"))
{
  fprintf(stderr, "mysql_query() failed\n");
  fprintf(stderr, "Error %u: %s\n",
```

```
           mysql_errno(mysql), mysql_error(mysql));
  exit(1);
}

MYSQL_RPL rpl;

rpl.file_name_length = 0;
rpl.file_name = NULL;
rpl.start_position = 4;
rpl.server_id = 0;
rpl.flags = 0;

if (mysql_binlog_open(mysql, &rpl))
{
  fprintf(stderr, "mysql_binlog_open() failed\n");
  fprintf(stderr, "Error %u: %s\n",
          mysql_errno(mysql), mysql_error(mysql));
  exit(1);
}
for (;;)  /* read events until error or EOF */
{
  if (mysql_binlog_fetch(mysql, &rpl))
  {
    fprintf(stderr, "mysql_binlog_fetch() failed\n");
    fprintf(stderr, "Error %u: %s\n",
            mysql_errno(mysql), mysql_error(mysql));
    break;
  }
  if (rpl.size == 0)  /* EOF */
  {
    fprintf(stderr, "EOF event received\n");
    break;
  }
  fprintf(stderr, "Event received of size %lu.\n", rpl.size);
}
mysql_binlog_close(mysql, &rpl);
```

For additional examples that show how to use these functions, look in a MySQL source distribution for these source files:

- `mysqlbinlog.cc` in the `client` directory
- `mysql_client_test.c` in the `testclients` directory

28.7.21.1 mysql_binlog_close()

```
void mysql_binlog_close(MYSQL *mysql, MYSQL_RPL *rpl)
```

Description

Close a replication event stream.

Arguments:

- `mysql`: The connection handler returned from `mysql_init()`. The handler remains open after the `mysql_binlog_close()` call.
- `rpl`: The replication stream structure. After calling `mysql_binlog_close()`, this structure should not be used further without reinitializing it and calling `mysql_binlog_open()` again.

Errors

None.

Example

See Section 28.7.21, “C API Binary Log Function Descriptions”.

28.7.21.2 mysql_binlog_fetch()

```
int mysql_binlog_fetch(MYSQL *mysql, MYSQL_RPL *rpl)
```

Description

Fetch one event from the replication event stream.

Arguments:

- `mysql`: The connection handler returned from `mysql_init()`.
- `rpl`: The replication stream structure. After a successful call, the `size` member indicates the event size, which is 0 for an EOF event. For a non-EOF event, `size` is greater than 0 and the `buffer` member points to the event contents.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

Example

See Section 28.7.21, “C API Binary Log Function Descriptions”.

28.7.21.3 mysql_binlog_open()

```
int mysql_binlog_open(MYSQL *mysql, MYSQL_RPL *rpl)
```

Description

Open a new replication event stream, to read a MySQL server binary log.

Arguments:

- `mysql`: The connection handler returned from `mysql_init()`.
- `rpl`: A `MYSQL_RPL` structure that has been initialized to indicate the replication event stream source. For a description of the structure members and how to initialize them, see Section 28.7.19, “C API Binary Log Data Structures”.

Return Values

Zero for success. Nonzero if an error occurred.

Errors

- `CR_FILE_NAME_TOO_LONG`

  The specified binary log file name was too long.

- `CR_OUT_OF_MEMORY`

  Out of memory.

Example

See Section 28.7.21, “C API Binary Log Function Descriptions”.

28.7.22 C API Encrypted Connection Support

This section describes how C applications use the C API capabilities for encrypted connections. By default, MySQL programs attempt to connect using encryption if the server supports encrypted connections, falling back to an unencrypted connection if an encrypted connection cannot be established (see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”). For applications that require control beyond the default behavior over how encrypted connections are established, the C API provides these capabilities:

- The `mysql_options()` function enables applications to set the appropriate SSL/TLS options before calling `mysql_real_connect()`. For example, to require the use of an encrypted connection, see Enforcing an Encrypted Connection.
- The `mysql_get_ssl_cipher()` function enables applications to determine, after a connection has been established, whether the connection uses encryption. A `NULL` return value indicates that encryption is not being used. A non-`NULL` return value indicates an encrypted connection and names the encryption cipher. See Section 28.7.7.34, “mysql_get_ssl_cipher()”.
- C API Options for Encrypted Connections
- Enforcing an Encrypted Connection
- Improving Security of Encrypted Connections

C API Options for Encrypted Connections

`mysql_options()` provides the following options for control over use of encrypted connections. For option details, see Section 28.7.7.50, “mysql_options()”.

- `MYSQL_OPT_SSL_CA`: The path name of the Certificate Authority (CA) certificate file. This option, if used, must specify the same certificate used by the server.
- `MYSQL_OPT_SSL_CAPATH`: The path name of the directory that contains trusted SSL CA certificate files.
- `MYSQL_OPT_SSL_CERT`: The path name of the client public key certificate file.
- `MYSQL_OPT_SSL_CIPHER`: The list of encryption ciphers the client permits for connections that use TLS protocols up through TLSv1.2.
- `MYSQL_OPT_SSL_CRL`: The path name of the file containing certificate revocation lists.
- `MYSQL_OPT_SSL_CRLPATH`: The path name of the directory that contains certificate revocation list files.
- `MYSQL_OPT_SSL_KEY`: The path name of the client private key file.
- `MYSQL_OPT_SSL_MODE`: The connection security state.
- `MYSQL_OPT_TLS_CIPHERSUITES`: The list of encryption ciphersuites the client permits for connections that use TLSv1.3.
- `MYSQL_OPT_TLS_VERSION`: The encryption protocols the client permits.

`mysql_ssl_set()` can be used as a convenience routine that is equivalent to a set of `mysql_options()` calls that specify certificate and key files, encryption ciphers, and so forth. See Section 28.7.7.77, “mysql_ssl_set()”.

Enforcing an Encrypted Connection

`mysql_options()` options for information such as SSL certificate and key files are used to establish an encrypted connection if such connections are available, but do not enforce any requirement that the connection obtained be encrypted. To require an encrypted connection, use the following technique:

1. Call `mysql_options()` as necessary supply the appropriate SSL parameters (certificate and key files, encryption ciphers, and so forth).
2. Call `mysql_options()` to pass the `MYSQL_OPT_SSL_MODE` option with a value of `SSL_MODE_REQUIRED` or one of the more-restrictive option values.
3. Call `mysql_real_connect()` to connect to the server. The call fails if an encrypted connection cannot be obtained; exit with an error.

Improving Security of Encrypted Connections

For additional security relative to that provided by the default encryption, clients can supply a CA certificate matching the one used by the server and enable host name identity verification. In this way, the server and client place their trust in the same CA certificate and the client verifies that the host to which it connected is the one intended:

- To specify the CA certificate, call `mysql_options()` to pass the `MYSQL_OPT_SSL_CA` (or `MYSQL_OPT_SSL_CAPATH`) option, and call `mysql_options()` to pass the `MYSQL_OPT_SSL_MODE` option with a value of `SSL_MODE_VERIFY_CA`.
- To enable host name identity verification as well, call `mysql_options()` to pass the `MYSQL_OPT_SSL_MODE` option with a value of `SSL_MODE_VERIFY_IDENTITY` rather than `SSL_MODE_VERIFY_CA`.

Note

Host name identity verification with `SSL_MODE_VERIFY_IDENTITY` does not work with self-signed certificates created automatically by the server, or manually using `mysql_ssl_rsa_setup` (see Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”). Such self-signed certificates do not contain the server name as the Common Name value.

Host name identity verification also does not work with certificates that specify the Common Name using wildcards because that name is compared verbatim to the server name.

28.7.23 C API Multiple Statement Execution Support

By default, `mysql_query()` and `mysql_real_query()` interpret their statement string argument as a single statement to be executed, and you process the result according to whether the statement produces a result set (a set of rows, as for `SELECT`) or an affected-rows count (as for `INSERT`, `UPDATE`, and so forth).

MySQL also supports the execution of a string containing multiple statements separated by semicolon (`;`) characters. This capability is enabled by special options that are specified either when you connect to the server with `mysql_real_connect()` or after connecting by calling `mysql_set_server_option()`.

Executing a multiple-statement string can produce multiple result sets or row-count indicators. Processing these results involves a different approach than for the single-statement case: After handling the result from the first statement, it is necessary to check whether more results exist and process them in turn

if so. To support multiple-result processing, the C API includes the `mysql_more_results()` and `mysql_next_result()` functions. These functions are used at the end of a loop that iterates as long as more results are available. *Failure to process the result this way may result in a dropped connection to the server.*

Multiple-result processing also is required if you execute `CALL` statements for stored procedures. Results from a stored procedure have these characteristics:

- Statements within the procedure may produce result sets (for example, if it executes `SELECT` statements). These result sets are returned in the order that they are produced as the procedure executes.

  In general, the caller cannot know how many result sets a procedure will return. Procedure execution may depend on loops or conditional statements that cause the execution path to differ from one call to the next. Therefore, you must be prepared to retrieve multiple results.

- The final result from the procedure is a status result that includes no result set. The status indicates whether the procedure succeeded or an error occurred.

The multiple statement and result capabilities can be used only with `mysql_query()` or `mysql_real_query()`. They cannot be used with the prepared statement interface. Prepared statement handlers are defined to work only with strings that contain a single statement. See Section 28.7.8, “C API Prepared Statements”.

To enable multiple-statement execution and result processing, the following options may be used:

- The `mysql_real_connect()` function has a `flags` argument for which two option values are relevant:

  - `CLIENT_MULTI_RESULTS` enables the client program to process multiple results. This option *must* be enabled if you execute `CALL` statements for stored procedures that produce result sets. Otherwise, such procedures result in an error `Error 1312 (0A000): PROCEDURE proc_name can't return a result set in the given context`. `CLIENT_MULTI_RESULTS` is enabled by default.

  - `CLIENT_MULTI_STATEMENTS` enables `mysql_query()` and `mysql_real_query()` to execute statement strings containing multiple statements separated by semicolons. This option also enables `CLIENT_MULTI_RESULTS` implicitly, so a `flags` argument of `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()` is equivalent to an argument of `CLIENT_MULTI_STATEMENTS | CLIENT_MULTI_RESULTS`. That is, `CLIENT_MULTI_STATEMENTS` is sufficient to enable multiple-statement execution and all multiple-result processing.

- After the connection to the server has been established, you can use the `mysql_set_server_option()` function to enable or disable multiple-statement execution by passing it an argument of `MYSQL_OPTION_MULTI_STATEMENTS_ON` or `MYSQL_OPTION_MULTI_STATEMENTS_OFF`. Enabling multiple-statement execution with this function also enables processing of “simple” results for a multiple-statement string where each statement produces a single result, but is *not* sufficient to permit processing of stored procedures that produce result sets.

The following procedure outlines a suggested strategy for handling multiple statements:

1. Pass `CLIENT_MULTI_STATEMENTS` to `mysql_real_connect()`, to fully enable multiple-statement execution and multiple-result processing.

2. After calling `mysql_query()` or `mysql_real_query()` and verifying that it succeeds, enter a loop within which you process statement results.

3. For each iteration of the loop, handle the current statement result, retrieving either a result set or an affected-rows count. If an error occurs, exit the loop.

4. At the end of the loop, call `mysql_next_result()` to check whether another result exists and initiate retrieval for it if so. If no more results are available, exit the loop.

One possible implementation of the preceding strategy is shown following. The final part of the loop can be reduced to a simple test of whether `mysql_next_result()` returns nonzero. The code as written distinguishes between no more results and an error, which enables a message to be printed for the latter occurrence.

```
/* connect to server with the CLIENT_MULTI_STATEMENTS option */
if (mysql_real_connect (mysql, host_name, user_name, password,
    db_name, port_num, socket_name, CLIENT_MULTI_STATEMENTS) == NULL)
{
  printf("mysql_real_connect() failed\n");
  mysql_close(mysql);
  exit(1);
}

/* execute multiple statements */
status = mysql_query(mysql,
                     "DROP TABLE IF EXISTS test_table;\
                      CREATE TABLE test_table(id INT);\
                      INSERT INTO test_table VALUES(10);\
                      UPDATE test_table SET id=20 WHERE id=10;\
                      SELECT * FROM test_table;\
                      DROP TABLE test_table");
if (status)
{
  printf("Could not execute statement(s)");
  mysql_close(mysql);
  exit(0);
}

/* process each statement result */
do {
  /* did current statement return data? */
  result = mysql_store_result(mysql);
  if (result)
  {
    /* yes; process rows and free the result set */
    process_result_set(mysql, result);
    mysql_free_result(result);
  }
  else          /* no result set or error */
  {
    if (mysql_field_count(mysql) == 0)
    {
      printf("%lld rows affected\n",
             mysql_affected_rows(mysql));
    }
    else  /* some error occurred */
    {
      printf("Could not retrieve result set\n");
      break;
    }
  }
  /* more results? -1 = no, >0 = error, 0 = yes (keep looping) */
  if ((status = mysql_next_result(mysql)) > 0)
    printf("Could not execute statement\n");
} while (status == 0);

mysql_close(mysql);
```

28.7.24 C API Prepared Statement Handling of Date and Time Values

The binary (prepared statement) protocol enables you to send and receive date and time values (`DATE`, `TIME`, `DATETIME`, and `TIMESTAMP`), using the `MYSQL_TIME` structure. The members of this structure are described in Section 28.7.9, “C API Prepared Statement Data Structures”.

To send temporal data values, create a prepared statement using `mysql_stmt_prepare()`. Then, before calling `mysql_stmt_execute()` to execute the statement, use the following procedure to set up each temporal parameter:

1. In the `MYSQL_BIND` structure associated with the data value, set the `buffer_type` member to the type that indicates what kind of temporal value you're sending. For `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` values, set `buffer_type` to `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, respectively.
2. Set the `buffer` member of the `MYSQL_BIND` structure to the address of the `MYSQL_TIME` structure in which you pass the temporal value.
3. Fill in the members of the `MYSQL_TIME` structure that are appropriate for the type of temporal value to pass.

Use `mysql_stmt_bind_param()` to bind the parameter data to the statement. Then you can call `mysql_stmt_execute()`.

To retrieve temporal values, the procedure is similar, except that you set the `buffer_type` member to the type of value you expect to receive, and the `buffer` member to the address of a `MYSQL_TIME` structure into which the returned value should be placed. Use `mysql_stmt_bind_result()` to bind the buffers to the statement after calling `mysql_stmt_execute()` and before fetching the results.

Here is a simple example that inserts `DATE`, `TIME`, and `TIMESTAMP` data. The `mysql` variable is assumed to be a valid connection handler.

```
MYSQL_TIME  ts;
MYSQL_BIND  bind[3];
MYSQL_STMT  *stmt;

strmov(query, "INSERT INTO test_table(date_field, time_field, \
                           timestamp_field) VALUES(?,?,?");

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(mysql, query, strlen(query)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

/* set up input buffers for all 3 parameters */
bind[0].buffer_type= MYSQL_TYPE_DATE;
bind[0].buffer= (char *)&ts;
bind[0].is_null= 0;
bind[0].length= 0;
...
bind[1]= bind[2]= bind[0];
...
```

```
mysql_stmt_bind_param(stmt, bind);

/* supply the data to be sent in the ts structure */
ts.year= 2002;
ts.month= 02;
ts.day= 03;

ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_stmt_execute(stmt);
..
```

28.7.25 C API Prepared CALL Statement Support

This section describes prepared-statement support in the C API for stored procedures executed using `CALL` statements:

Stored procedures executed using prepared `CALL` statements can be used in the following ways:

- A stored procedure can produce any number of result sets. The number of columns and the data types of the columns need not be the same for all result sets.
- The final values of `OUT` and `INOUT` parameters are available to the calling application after the procedure returns. These parameters are returned as an extra single-row result set following any result sets produced by the procedure itself. The row contains the values of the `OUT` and `INOUT` parameters in the order in which they are declared in the procedure parameter list.

  For information about the effect of unhandled conditions on procedure parameters, see Section 13.6.7.8, “Condition Handling and OUT or INOUT Parameters”.

The following discussion shows how to use these capabilities through the C API for prepared statements. To use prepared `CALL` statements through the `PREPARE` and `EXECUTE` statements, see Section 13.2.1, “CALL Syntax”.

An application that executes a prepared `CALL` statement should use a loop that fetches a result and then invokes `mysql_stmt_next_result()` to determine whether there are more results. The results consist of any result sets produced by the stored procedure followed by a final status value that indicates whether the procedure terminated successfully.

If the procedure has `OUT` or `INOUT` parameters, the result set preceding the final status value contains their values. To determine whether a result set contains parameter values, test whether the `SERVER_PS_OUT_PARAMS` bit is set in the `server_status` member of the `MYSQL` connection handler:

```
mysql->server_status & SERVER_PS_OUT_PARAMS
```

The following example uses a prepared `CALL` statement to execute a stored procedure that produces multiple result sets and that provides parameter values back to the caller by means of `OUT` and `INOUT` parameters. The procedure takes parameters of all three types (`IN`, `OUT`, `INOUT`), displays their initial values, assigns new values, displays the updated values, and returns. The expected return information from the procedure therefore consists of multiple result sets and a final status:

- One result set from a `SELECT` that displays the initial parameter values: `10`, `NULL`, `30`. (The `OUT` parameter is assigned a value by the caller, but this assignment is expected to be ineffective: `OUT` parameters are seen as `NULL` within a procedure until assigned a value within the procedure.)
- One result set from a `SELECT` that displays the modified parameter values: `100`, `200`, `300`.

- One result set containing the final `OUT` and `INOUT` parameter values: `200`, `300`.
- A final status packet.

The code to execute the procedure:

```
MYSQL_STMT *stmt;
MYSQL_BIND ps_params[3];  /* input parameter buffers */
int        int_data[3];   /* input/output values */
bool       is_null[3];    /* output value nullability */
int        status;

/* set up stored procedure */
status = mysql_query(mysql, "DROP PROCEDURE IF EXISTS p1");
test_error(mysql, status);

status = mysql_query(mysql,
  "CREATE PROCEDURE p1("
  "  IN p_in INT, "
  "  OUT p_out INT, "
  "  INOUT p_inout INT) "
  "BEGIN "
  "  SELECT p_in, p_out, p_inout; "
  "  SET p_in = 100, p_out = 200, p_inout = 300; "
  "  SELECT p_in, p_out, p_inout; "
  "END");
test_error(mysql, status);

/* initialize and prepare CALL statement with parameter placeholders */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  printf("Could not initialize statement\n");
  exit(1);
}
status = mysql_stmt_prepare(stmt, "CALL p1(?, ?, ?)", 16);
test_stmt_error(stmt, status);

/* initialize parameters: p_in, p_out, p_inout (all INT) */
memset(ps_params, 0, sizeof (ps_params));

ps_params[0].buffer_type = MYSQL_TYPE_LONG;
ps_params[0].buffer = (char *) &int_data[0];
ps_params[0].length = 0;
ps_params[0].is_null = 0;

ps_params[1].buffer_type = MYSQL_TYPE_LONG;
ps_params[1].buffer = (char *) &int_data[1];
ps_params[1].length = 0;
ps_params[1].is_null = 0;

ps_params[2].buffer_type = MYSQL_TYPE_LONG;
ps_params[2].buffer = (char *) &int_data[2];
ps_params[2].length = 0;
ps_params[2].is_null = 0;

/* bind parameters */
status = mysql_stmt_bind_param(stmt, ps_params);
test_stmt_error(stmt, status);

/* assign values to parameters and execute statement */
int_data[0]= 10;  /* p_in */
int_data[1]= 20;  /* p_out */
int_data[2]= 30;  /* p_inout */

status = mysql_stmt_execute(stmt);
```

```
test_stmt_error(stmt, status);

/* process results until there are no more */
do {
  int i;
  int num_fields;       /* number of columns in result */
  MYSQL_FIELD *fields;  /* for result set metadata */
  MYSQL_BIND *rs_bind;  /* for output buffers */

  /* the column count is > 0 if there is a result set */
  /* 0 if the result is only the final status packet */
  num_fields = mysql_stmt_field_count(stmt);

  if (num_fields > 0)
  {
    /* there is a result set to fetch */
    printf("Number of columns in result: %d\n", (int) num_fields);

    /* what kind of result set is this? */
    printf("Data: ");
    if(mysql->server_status & SERVER_PS_OUT_PARAMS)
      printf("this result set contains OUT/INOUT parameters\n");
    else
      printf("this result set is produced by the procedure\n");

    MYSQL_RES *rs_metadata = mysql_stmt_result_metadata(stmt);
    test_stmt_error(stmt, rs_metadata == NULL);

    fields = mysql_fetch_fields(rs_metadata);

    rs_bind = (MYSQL_BIND *) malloc(sizeof (MYSQL_BIND) * num_fields);
    if (!rs_bind)
    {
      printf("Cannot allocate output buffers\n");
      exit(1);
    }
    memset(rs_bind, 0, sizeof (MYSQL_BIND) * num_fields);

    /* set up and bind result set output buffers */
    for (i = 0; i < num_fields; ++i)
    {
      rs_bind[i].buffer_type = fields[i].type;
      rs_bind[i].is_null = &is_null[i];

      switch (fields[i].type)
      {
        case MYSQL_TYPE_LONG:
          rs_bind[i].buffer = (char *) &(int_data[i]);
          rs_bind[i].buffer_length = sizeof (int_data);
          break;

        default:
          fprintf(stderr, "ERROR: unexpected type: %d.\n", fields[i].type);
          exit(1);
      }
    }

    status = mysql_stmt_bind_result(stmt, rs_bind);
    test_stmt_error(stmt, status);

    /* fetch and display result set rows */
    while (1)
    {
      status = mysql_stmt_fetch(stmt);

      if (status == 1 || status == MYSQL_NO_DATA)
        break;
```

```
      for (i = 0; i < num_fields; ++i)
      {
        switch (rs_bind[i].buffer_type)
        {
          case MYSQL_TYPE_LONG:
            if (*rs_bind[i].is_null)
              printf(" val[%d] = NULL;", i);
            else
              printf(" val[%d] = %ld;",
                     i, (long) *((int *) rs_bind[i].buffer));
            break;

          default:
            printf("  unexpected type (%d)\n",
              rs_bind[i].buffer_type);
        }
      }
      printf("\n");
    }

    mysql_free_result(rs_metadata); /* free metadata */
    free(rs_bind);                  /* free output buffers */
  }
  else
  {
    /* no columns = final status packet */
    printf("End of procedure output\n");
  }

  /* more results? -1 = no, >0 = error, 0 = yes (keep looking) */
  status = mysql_stmt_next_result(stmt);
  if (status > 0)
    test_stmt_error(stmt, status);
} while (status == 0);

mysql_stmt_close(stmt);
```

Execution of the procedure should produce the following output:

```
Number of columns in result: 3
Data: this result set is produced by the procedure
 val[0] = 10; val[1] = NULL; val[2] = 30;
Number of columns in result: 3
Data: this result set is produced by the procedure
 val[0] = 100; val[1] = 200; val[2] = 300;
Number of columns in result: 2
Data: this result set contains OUT/INOUT parameters
 val[0] = 200; val[1] = 300;
End of procedure output
```

The code uses two utility routines, `test_error()` and `test_stmt_error()`, to check for errors and terminate after printing diagnostic information if an error occurred:

```
static void test_error(MYSQL *mysql, int status)
{
  if (status)
  {
    fprintf(stderr, "Error: %s (errno: %d)\n",
            mysql_error(mysql), mysql_errno(mysql));
    exit(1);
  }
}

static void test_stmt_error(MYSQL_STMT *stmt, int status)
```

```
{
  if (status)
  {
    fprintf(stderr, "Error: %s (errno: %d)\n",
            mysql_stmt_error(stmt), mysql_stmt_errno(stmt));
    exit(1);
  }
}
```

28.7.26 C API Prepared Statement Problems

Here follows a list of the currently known problems with prepared statements:

- `TIME`, `TIMESTAMP`, and `DATETIME` do not support parts of seconds (for example, from `DATE_FORMAT()`).
- When converting an integer to string, `ZEROFILL` is honored with prepared statements in some cases where the MySQL server does not print the leading zeros. (For example, with `MIN(number-with-zerofill)`).
- When converting a floating-point number to a string in the client, the rightmost digits of the converted value may differ slightly from those of the original value.
- Prepared statements do not support multi-statements (that is, multiple statements within a single string separated by `;` characters).
- The capabilities of prepared `CALL` statements are described in Section 28.7.25, “C API Prepared CALL Statement Support”.

28.7.27 C API Optional Result Set Metadata

When a client executes a statement that produces a result set, MySQL by default makes available both the data the result set contains, and result set metadata that provides information about the result set data. Metadata is contained in the `MYSQL_FIELD` structure (see Section 28.7.5, “C API Data Structures”), which is returned by the `mysql_fetch_field()`, `mysql_fetch_field_direct()`, and `mysql_fetch_fields()` functions.

Clients can indicate on a per-connection basis that result set metadata is optional and that the client will indicate to the server whether to return it. Suppression of metadata transfer can improve performance, particularly for sessions that execute many queries that return few rows each.

There are two ways to indicate that result set metadata is optional for a connection. They are equivalent, so either one suffices:

- At connect time, enable the `CLIENT_OPTIONAL_RESULTSET_METADATA` flag for the `client_flag` argument of `mysql_real_connect()`.
- Prior to connect time, enable the `MYSQL_OPT_OPTIONAL_RESULTSET_METADATA` option for `mysql_options()`.

For metadata-optional connections, the client sets the `resultset_metadata` system variable to control whether the server returns result set metadata. Permitted values are `FULL` (return all metadata; this is the default) and `NONE` (return no metadata).

For metadata-optional connections, the `mysql_fetch_field()`, `mysql_fetch_field_direct()`, and `mysql_fetch_fields()` functions return `NULL` when the `resultset_metadata` system variable is set to `NONE`.

To check whether a result set has metadata, use the `mysql_result_metadata()` function. This function returns `RESULTSET_METADATA_FULL` or `RESULTSET_METADATA_NONE` to indicate that the result set has full metadata or no metadata, respectively.

`mysql_result_metadata()` can be useful if the client does not know in advance whether a result set has metadata. For example, if a client executes a stored procedure that returns multiple result sets and might change the `resultset_metadata` system variable, the client can invoke `mysql_result_metadata()` for each result set to determine whether it has metadata.

For connections that are not metadata-optional, setting `resultset_metadata` to `NONE` produces an error.

28.7.28 C API Automatic Reconnection Control

The MySQL client library can perform an automatic reconnection to the server if it finds that the connection is down when you attempt to send a statement to the server to be executed. If auto-reconnect is enabled, the library tries once to reconnect to the server and send the statement again.

Auto-reconnect is disabled by default.

If it is important for your application to know that the connection has been dropped (so that it can exit or take action to adjust for the loss of state information), be sure that auto-reconnect is disabled. To ensure this, call `mysql_options()` with the `MYSQL_OPT_RECONNECT` option:

```
bool reconnect = 0;
mysql_options(&mysql, MYSQL_OPT_RECONNECT, &reconnect);
```

If the connection has gone down, the effect of `mysql_ping()` depends on the auto-reconnect state. If auto-reconnect is enabled, `mysql_ping()` performs a reconnect. Otherwise, it returns an error.

Some client programs might provide the capability of controlling automatic reconnection. For example, `mysql` reconnects by default, but the `--skip-reconnect` option can be used to suppress this behavior.

If an automatic reconnection does occur (for example, as a result of calling `mysql_ping()`), there is no explicit indication of it. To check for reconnection, call `mysql_thread_id()` to get the original connection identifier before calling `mysql_ping()`, then call `mysql_thread_id()` again to see whether the identifier changed.

Automatic reconnection can be convenient because you need not implement your own reconnect code, but if a reconnection does occur, several aspects of the connection state are reset on the server side and your application will not be notified.

The connection-related state is affected as follows:

- Any active transactions are rolled back and autocommit mode is reset.
- All table locks are released.
- All `TEMPORARY` tables are closed (and dropped).
- Session system variables are reinitialized to the values of the corresponding global system variables, including system variables that are set implicitly by statements such as `SET NAMES`.
- User variable settings are lost.
- Prepared statements are released.

- `HANDLER` variables are closed.
- The value of `LAST_INSERT_ID()` is reset to 0.
- Locks acquired with `GET_LOCK()` are released.
- The association of the client with the Performance Schema `threads` table row that determines connection thread instrumentation is lost. If the client reconnects after a disconnect, the session is associated with a new row in the `threads` table and the thread monitoring state may be different. See Section 26.12.18.5, “The threads Table”.

If reconnection occurs, any SQL statement specified by calling `mysql_options()` with the `MYSQL_INIT_COMMAND` option is re-executed.

If the connection drops, it is possible that the session associated with the connection on the server side will still be running if the server has not yet detected that the client is no longer connected. In this case, any locks held by the original connection still belong to that session, so you may want to kill it by calling `mysql_kill()`.

28.7.29 C API Common Issues

28.7.29.1 Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success

It is possible for `mysql_store_result()` to return `NULL` following a successful call to `mysql_query()`. When this happens, it means one of the following conditions occurred:

- There was a `malloc()` failure (for example, if the result set was too large).
- The data could not be read (an error occurred on the connection).
- The query returned no data (for example, it was an `INSERT`, `UPDATE`, or `DELETE`).

You can always check whether the statement should have produced a nonempty result by calling `mysql_field_count()`. If `mysql_field_count()` returns zero, the result is empty and the last query was a statement that does not return values (for example, an `INSERT` or a `DELETE`). If `mysql_field_count()` returns a nonzero value, the statement should have produced a nonempty result. See the description of the `mysql_field_count()` function for an example.

You can test for an error by calling `mysql_error()` or `mysql_errno()`.

28.7.29.2 What Results You Can Get from a Query

In addition to the result set returned by a query, you can also get the following information:

- `mysql_affected_rows()` returns the number of rows affected by the last query when doing an `INSERT`, `UPDATE`, or `DELETE`.

  For a fast re-create, use `TRUNCATE TABLE`.

- `mysql_num_rows()` returns the number of rows in a result set. With `mysql_store_result()`, `mysql_num_rows()` may be called as soon as `mysql_store_result()` returns. With `mysql_use_result()`, `mysql_num_rows()` may be called only after you have fetched all the rows with `mysql_fetch_row()`.
- `mysql_insert_id()` returns the ID generated by the last query that inserted a row into a table with an `AUTO_INCREMENT` index. See Section 28.7.7.38, “mysql_insert_id()”.

- Some queries (`LOAD DATA`, `INSERT INTO ... SELECT`, `UPDATE`) return additional information. The result is returned by `mysql_info()`. See the description for `mysql_info()` for the format of the string that it returns. `mysql_info()` returns a `NULL` pointer if there is no additional information.

28.7.29.3 How to Get the Unique ID for the Last Inserted Row

If you insert a record into a table that contains an `AUTO_INCREMENT` column, you can obtain the value stored into that column by calling the `mysql_insert_id()` function.

You can check from your C applications whether a value was stored in an `AUTO_INCREMENT` column by executing the following code (which assumes that you've checked that the statement succeeded). It determines whether the query was an `INSERT` with an `AUTO_INCREMENT` index:

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

When a new `AUTO_INCREMENT` value has been generated, you can also obtain it by executing a `SELECT LAST_INSERT_ID()` statement with `mysql_query()` and retrieving the value from the result set returned by the statement.

When inserting multiple values, the last automatically incremented value is returned.

For `LAST_INSERT_ID()`, the most recently generated ID is maintained in the server on a per-connection basis. It is not changed by another client. It is not even changed if you update another `AUTO_INCREMENT` column with a nonmagic value (that is, a value that is not `NULL` and not `0`). Using `LAST_INSERT_ID()` and `AUTO_INCREMENT` columns simultaneously from multiple clients is perfectly valid. Each client will receive the last inserted ID for the last statement *that* client executed.

If you want to use the ID that was generated for one table and insert it into a second table, you can use SQL statements like this:

```
INSERT INTO foo (auto,text)
    VALUES(NULL,'text');         # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
    VALUES(LAST_INSERT_ID(),'text');  # use ID in second table
```

`mysql_insert_id()` returns the value stored into an `AUTO_INCREMENT` column, whether that value is automatically generated by storing `NULL` or `0` or was specified as an explicit value. `LAST_INSERT_ID()` returns only automatically generated `AUTO_INCREMENT` values. If you store an explicit value other than `NULL` or `0`, it does not affect the value returned by `LAST_INSERT_ID()`.

For more information on obtaining the last ID in an `AUTO_INCREMENT` column:

- For information on `LAST_INSERT_ID()`, which can be used within an SQL statement, see Section 12.15, “Information Functions”.
- For information on `mysql_insert_id()`, the function you use from within the C API, see Section 28.7.7.38, “mysql_insert_id()”.
- For information on obtaining the auto-incremented value when using Connector/J, see Retrieving `AUTO_INCREMENT` Column Values through JDBC.
- For information on obtaining the auto-incremented value when using Connector/ODBC, see Obtaining Auto-Increment Values.

28.8 MySQL PHP API

The MySQL PHP API manual is now published in standalone form, not as part of the MySQL Reference Manual. See MySQL and PHP.

28.9 MySQL Perl API

The Perl `DBI` module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI with MySQL, install the following:

1. The `DBI` module.
2. The `DBD::mysql` module. This is the DataBase Driver (DBD) module for Perl.
3. Optionally, the DBD module for any other type of database server you want to access.

Perl DBI is the recommended Perl interface. It replaces an older interface called `mysqlperl`, which should be considered obsolete.

These sections contain information about using Perl with MySQL and writing MySQL applications in Perl:

- For installation instructions for Perl DBI support, see Section 2.13, “Perl Installation Notes”.
- For an example of reading options from option files, see Section 5.8.4, “Using Client Programs in a Multiple-Server Environment”.
- For secure coding tips, see Section 6.1.1, “Security Guidelines”.
- For debugging tips, see Section 29.5.1.4, “Debugging mysqld under gdb”.
- For some Perl-specific environment variables, see Section 4.9, “MySQL Environment Variables”.
- For considerations for running on OS X, see Section 2.4, “Installing MySQL on macOS”.
- For ways to quote string literals, see Section 9.1.1, “String Literals”.

DBI information is available at the command line, online, or in printed form:

- Once you have the `DBI` and `DBD::mysql` modules installed, you can get information about them at the command line with the `perldoc` command:

  ```
  shell> perldoc DBI
  shell> perldoc DBI::FAQ
  shell> perldoc DBD::mysql
  ```

  You can also use `pod2man`, `pod2html`, and so on to translate this information into other formats.

- For online information about Perl DBI, visit the DBI website, http://dbi.perl.org/. That site hosts a general DBI mailing list.
- For printed information, the official DBI book is *Programming the Perl DBI* (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000). Information about the book is available at the DBI website, http://dbi.perl.org/.

28.10 MySQL Python API

`MySQLdb` is a third-party driver that provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at http://sourceforge.net/projects/mysql-python/.

The new MySQL Connector/Python component provides an interface to the same Python API, and is built into the MySQL Server and supported by Oracle. See MySQL Connector/Python Developer Guide for details on the Connector, as well as coding guidelines for Python applications and sample Python code.

28.11 MySQL Ruby APIs

Two APIs are available for Ruby programmers developing MySQL applications:

- The MySQL/Ruby API is based on the `libmysqlclient` API library. For information on installing and using the MySQL/Ruby API, see Section 28.11.1, “The MySQL/Ruby API”.
- The Ruby/MySQL API is written to use the native MySQL network protocol (a native driver). For information on installing and using the Ruby/MySQL API, see Section 28.11.2, “The Ruby/MySQL API”.

For background and syntax information about the Ruby language, see Ruby Programming Language.

28.11.1 The MySQL/Ruby API

The MySQL/Ruby module provides access to MySQL databases using Ruby through `libmysqlclient`.

For information on installing the module, and the functions exposed, see MySQL/Ruby.

28.11.2 The Ruby/MySQL API

The Ruby/MySQL module provides access to MySQL databases using Ruby through a native driver interface using the MySQL network protocol.

For information on installing the module, and the functions exposed, see Ruby/MySQL.

28.12 MySQL Tcl API

`MySQLtcl` is a simple API for accessing a MySQL database server from the Tcl programming language. It can be found at http://www.xdobry.de/mysqltcl/.

28.13 MySQL Eiffel Wrapper

Eiffel MySQL is an interface to the MySQL database server using the Eiffel programming language, written by Michael Ravits. It can be found at http://efsa.sourceforge.net/archive/ravits/mysql.htm.

Chapter 29 Extending MySQL

Table of Contents

29.1 MySQL Internals 4943
29.1.1 MySQL Threads 4943
29.1.2 The MySQL Test Suite 4944
29.2 The MySQL Plugin API 4945
29.2.1 Types of Plugins 4945
29.2.2 Plugin API Characteristics 4950
29.2.3 Plugin API Components 4951
29.2.4 Writing Plugins 4952
29.3 MySQL Services for Plugins 5010
29.3.1 The Locking Service 5012
29.3.2 The Keyring Service 5018
29.4 Adding New Functions to MySQL 5020
29.4.1 Features of the User-Defined Function Interface 5021
29.4.2 Adding a New User-Defined Function 5021
29.4.3 Adding a New Native Function 5031
29.5 Debugging and Porting MySQL 5033
29.5.1 Debugging a MySQL Server 5033
29.5.2 Debugging a MySQL Client 5039
29.5.3 The LOCK_ORDER Tool 5040
29.5.4 The DBUG Package 5046

29.1 MySQL Internals

This chapter describes what you need to know when working on the MySQL code. To track or contribute to MySQL development, follow the instructions in Section 2.9.5, “Installing MySQL Using a Development Source Tree”. If you are interested in MySQL internals, you should also join the MySQL Community Slack. Feel free to ask questions about the code and to send patches that you would like to contribute to the MySQL project!

Note

The MySQL source code contains internal documentation written using Doxygen. This documentation is useful for understanding how MySQL works from a developer perspective. The generated Doxygen content is available at https://dev.mysql.com/doc/index-other.html. It is also possible to generate this content locally from a MySQL source distribution using the instructions at Section 2.9.10, “Generating MySQL Doxygen Documentation Content”.

29.1.1 MySQL Threads

The MySQL server creates the following threads:

- Connection manager threads handle client connection requests on the network interfaces that the server listens to. On all platforms, one manager thread handles TCP/IP connection requests. On Unix, this manager thread also handles Unix socket file connection requests. On Windows, a manager thread handles shared-memory connection requests, and another handles named-pipe connection requests. The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.

- Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

  For information about tuning the parameters that control thread resources, see Section 8.12.4.1, "How MySQL Handles Client Connections".

- On a master replication server, connections from slave servers are handled like client connections: There is one thread per connected slave.

- On a slave replication server, an I/O thread is started to connect to the master server and read updates from it. An SQL thread is started to apply updates read from the master. These two threads run independently and can be started and stopped independently.

- A signal thread handles all signals. This thread also normally handles alarms and calls `process_alarm()` to force timeouts on connections that have been idle too long.

- If `InnoDB` is used, there will be additional read and write threads by default. The number of these are controlled by the `innodb_read_io_threads` and `innodb_write_io_threads` parameters. See Section 15.13, "InnoDB Startup Options and System Variables".

- If the server is started with the `--flush_time=val` option, a dedicated thread is created to flush all tables every *`val`* seconds.

- If the event scheduler is active, there is one thread for the scheduler, and a thread for each event currently running. See Section 24.4.1, "Event Scheduler Overview".

`mysqladmin processlist` only shows the connection, replication, and event threads.

29.1.2 The MySQL Test Suite

The test system that is included in Unix source and binary distributions makes it possible for users and developers to perform regression tests on the MySQL code. These tests can be run on Unix.

You can also write your own test cases. For information, including system requirements, see The MySQL Test Framework in the MySQL Server Doxygen documentation, available at https://dev.mysql.com/doc/index-other.html.

The current set of test cases does not test everything in MySQL, but it should catch most obvious bugs in the SQL processing code, operating system or library issues, and is quite thorough in testing replication. Our goal is to have the tests cover 100% of the code. We welcome contributions to our test suite. You may especially want to contribute tests that examine the functionality critical to your system because this ensures that all future MySQL releases work well with your applications.

The test system consists of a test language interpreter (`mysqltest`), a Perl script to run all tests (`mysql-test-run.pl`), the actual test cases written in a special test language, and their expected results. To run the test suite on your system after a build, type `make test` from the source root directory, or change location to the `mysql-test` directory and type `./mysql-test-run.pl`. If you have installed a binary distribution, change location to the `mysql-test` directory under the installation root directory (for example, `/usr/local/mysql/mysql-test`), and run `./mysql-test-run.pl`. All tests should succeed. If any do not, feel free to try to find out why and report the problem if it indicates a bug in MySQL. See Section 1.7, "How to Report Bugs or Problems".

If one test fails, you should run `mysql-test-run.pl` with the `--force` option to check whether any other tests fail.

If you have a copy of `mysqld` running on the machine where you want to run the test suite, you do not have to stop it, as long as it is not using ports `9306` or `9307`. If either of those ports is taken, you should set the `MTR_BUILD_THREAD` environment variable to an appropriate value, and the test suite will use a different set of ports for master, slave, and NDB). For example:

```
shell> export MTR_BUILD_THREAD=31
shell> ./mysql-test-run.pl [options] [test_name]
```

In the `mysql-test` directory, you can run an individual test case with `./mysql-test-run.pl test_name`.

If you have a question about the test suite, or have a test case to contribute, join the MySQL Community Slack.

29.2 The MySQL Plugin API

MySQL supports a plugin API that enables creation of server components. Plugins can be loaded at server startup, or loaded and unloaded at runtime without restarting the server. The API is generic and does not specify what plugins can do. The components supported by this interface include, but are not limited to, storage engines, full-text parser plugins, and server extensions.

For example, full-text parser plugins can be used to replace or augment the built-in full-text parser. A plugin can parse text into words using rules that differ from those used by the built-in parser. This can be useful if you need to parse text with characteristics different from those expected by the built-in parser.

The plugin interface is more general than the older user-defined function (UDF) interface.

The plugin interface uses the `plugin` table in the `mysql` database to record information about plugins that have been installed permanently with the `INSTALL PLUGIN` statement. This table is created as part of the MySQL installation process. Plugins can also be installed for a single server invocation with the `--plugin-load` option. Plugins installed this way are not recorded in the `plugin` table. See Section 5.6.1, “Installing and Uninstalling Plugins”.

MySQL supports an API for client plugins in addition to that for server plugins. This is used, for example, by authentication plugins where a server-side plugin and a client-side plugin cooperate to enable clients to connect to the server through a variety of authentication methods.

Note

The MySQL source code contains internal documentation written using Doxygen. This documentation is useful for understanding how MySQL works from a developer perspective. The generated Doxygen content is available at https://dev.mysql.com/doc/index-other.html. It is also possible to generate this content locally from a MySQL source distribution using the instructions at Section 2.9.10, “Generating MySQL Doxygen Documentation Content”.

Additional Resources

The book *MySQL 5.1 Plugin Development* by Sergei Golubchik and Andrew Hutchings provides a wealth of detail about the plugin API. Despite the fact that the book's title refers to MySQL Server 5.1, most of the information in it applies to later versions as well.

29.2.1 Types of Plugins

The plugin API enables creation of plugins that implement several capabilities:

- Storage engines
- Full-text parsers
- Daemons
- `INFORMATION_SCHEMA` tables
- Semisynchronous replication
- Auditing
- Authentication
- Password validation and strength checking
- Protocol tracing
- Query rewriting
- Secure keyring storage and retrieval

The following sections provide an overview of these plugin types.

Storage Engine Plugins

The pluggable storage engine architecture used by MySQL Server enables storage engines to be written as plugins and loaded into and unloaded from a running server. For a description of this architecture, see Section 16.11, “Overview of MySQL Storage Engine Architecture”.

For information on how to use the plugin API to write storage engines, see MySQL Internals: Writing a Custom Storage Engine.

Full-Text Parser Plugins

MySQL has a built-in parser that it uses by default for full-text operations (parsing text to be indexed, or parsing a query string to determine the terms to be used for a search). The built-in full-text parser is supported with `InnoDB` and `MyISAM` tables.

MySQL also has a character-based ngram full-text parser that supports Chinese, Japanese, and Korean (CJK), and a word-based MeCab parser plugin that supports Japanese, for use with `InnoDB` and `MyISAM` tables.

For full-text processing, “parsing” means extracting words (or “tokens”, in the case of an n-gram character-based parser) from text or a query string based on rules that define which character sequences make up a word and where word boundaries lie.

When parsing for indexing purposes, the parser passes each word to the server, which adds it to a full-text index. When parsing a query string, the parser passes each word to the server, which accumulates the words for use in a search.

The parsing properties of the built-in full-text parser are described in Section 12.9, “Full-Text Search Functions”. These properties include rules for determining how to extract words from text. The parser is influenced by certain system variables that cause words shorter or longer to be excluded, and by the stopword list that identifies common words to be ignored. For more information, see Section 12.9.4, “Full-Text Stopwords”, and Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”.

The plugin API enables you to use a full-text parser other than the default built-in full-text parser. For example, if you are working with Japanese, you may choose to use the MeCab full-text parser. The plugin

API also enables you to provide a full-text parser of your own so that you have control over the basic duties of a parser. A parser plugin can operate in either of two roles:

- The plugin can replace the built-in parser. In this role, the plugin reads the input to be parsed, splits it up into words, and passes the words to the server (either for indexing or for token accumulation). The ngram and MeCab parsers operate as replacements for the built-in full-text parser.

  You may choose to provide your own full-text parser if you need to use different rules from those of the built-in parser for determining how to split up input into words. For example, the built-in parser considers the text “case-sensitive” to consist of two words “case” and “sensitive,” whereas an application might need to treat the text as a single word.

- The plugin can act in conjunction with the built-in parser by serving as a front end for it. In this role, the plugin extracts text from the input and passes the text to the parser, which splits up the text into words using its normal parsing rules. This parsing is affected by the `innodb_ft_xxx` or `ft_xxx` system variables and the stopword list.

  One reason to use a parser this way is that you need to index content such as PDF documents, XML documents, or `.doc` files. The built-in parser is not intended for those types of input but a plugin can pull out the text from these input sources and pass it to the built-in parser.

It is also possible for a parser plugin to operate in both roles. That is, it could extract text from noncleartext input (the front end role), and also parse the text into words (thus replacing the built-in parser).

A full-text plugin is associated with full-text indexes on a per-index basis. That is, when you install a parser plugin initially, that does not cause it to be used for any full-text operations. It simply becomes available. For example, a full-text parser plugin becomes available to be named in a `WITH PARSER` clause when creating individual `FULLTEXT` indexes. To create such an index at table-creation time, do this:

```
CREATE TABLE t
(
  doc CHAR(255),
  FULLTEXT INDEX (doc) WITH PARSER parser_name
) ENGINE=InnoDB;
```

Or you can add the index after the table has been created:

```
ALTER TABLE t ADD FULLTEXT INDEX (doc) WITH PARSER parser_name;
```

The only SQL change for associating the parser with the index is the `WITH PARSER` clause. Searches are specified as before, with no changes needed for queries.

When you associate a parser plugin with a `FULLTEXT` index, the plugin is required for using the index. If the parser plugin is dropped, any index associated with it becomes unusable. Any attempt to use a table for which a plugin is not available results in an error, although `DROP TABLE` is still possible.

For more information about full-text plugins, see Section 29.2.4.4, “Writing Full-Text Parser Plugins”. MySQL 8.0 supports full-text plugins with `MyISAM` and `InnoDB`.

Daemon Plugins

A daemon plugin is a simple type of plugin used for code that should be run by the server but that does not communicate with it. MySQL distributions include an example daemon plugin that writes periodic heartbeat messages to a file.

For more information about daemon plugins, see Section 29.2.4.5, “Writing Daemon Plugins”.

INFORMATION_SCHEMA Plugins

`INFORMATION_SCHEMA` plugins enable the creation of tables containing server metadata that are exposed to users through the `INFORMATION_SCHEMA` database. For example, `InnoDB` uses `INFORMATION_SCHEMA` plugins to provide tables that contain information about current transactions and locks.

For more information about `INFORMATION_SCHEMA` plugins, see Section 29.2.4.6, "Writing INFORMATION_SCHEMA Plugins".

Semisynchronous Replication Plugins

MySQL replication is asynchronous by default. With semisynchronous replication, a commit performed on the master side blocks before returning to the session that performed the transaction until at least one slave acknowledges that it has received and logged the events for the transaction. Semisynchronous replication is implemented through complementary master and client plugins. See Section 17.4.9, "Semisynchronous Replication".

For more information about semisynchronous replication plugins, see Section 29.2.4.7, "Writing Semisynchronous Replication Plugins".

Audit Plugins

The MySQL server provides a pluggable audit interface that enables information about server operations to be reported to interested parties. Audit notification occurs for these operations (although the interface is general and the server could be modified to report others):

- Write a message to the general query log (if the log is enabled)
- Write a message to the error log
- Send a query result to a client

Audit plugins may register with the audit interface to receive notification about server operations. When an auditable event occurs within the server, the server determines whether notification is needed. For each registered audit plugin, the server checks the event against those event classes in which the plugin is interested and passes the event to the plugin if there is a match.

This interface enables audit plugins to receive notifications only about operations in event classes they consider significant and to ignore others. The interface provides for categorization of operations into event classes and further division into event subclasses within each class.

When an audit plugin is notified of an auditable event, it receives a pointer to the current THD structure and a pointer to a structure that contains information about the event. The plugin can examine the event and perform whatever auditing actions are appropriate. For example, the plugin can see what statement produced a result set or was logged, the number of rows in a result, who the current user was for an operation, or the error code for failed operations.

For more information about audit plugins, see Section 29.2.4.8, "Writing Audit Plugins".

Authentication Plugins

MySQL supports pluggable authentication. Authentication plugins exist on both the server and client sides. Plugins on the server side implement authentication methods for use by clients when they connect to the server. A plugin on the client side communicates with a server-side plugin to provide the authentication information that it requires. A client-side plugin may interact with the user, performing tasks such as soliciting a password or other authentication credentials to be sent to the server. See Section 6.2.17, "Pluggable Authentication".

Pluggable authentication also enables proxy user capability, in which one user takes the identity of another user. A server-side authentication plugin can return to the server the name of the user whose identity the connecting user should have. See Section 6.2.18, “Proxy Users”.

For more information about authentication plugins, see Section 29.2.4.9, “Writing Authentication Plugins”.

Password-Validation Plugins

The MySQL server provides an interface for writing plugins that test passwords. Such a plugin implements two capabilities:

- Rejection of too-weak passwords in statements that assign passwords (such as `CREATE USER` and `ALTER USER` statements).
- Assessing the strength of potential passwords for the `VALIDATE_PASSWORD_STRENGTH()` SQL function.

For information about writing this type of plugin, see Section 29.2.4.10, “Writing Password-Validation Plugins”.

Protocol Trace Plugins

MySQL supports the use of protocol trace plugins: client-side plugins that implement tracing of communication between a client and the server that takes place using the client/server protocol.

For more information about protocol trace plugins, see Section 29.2.4.11, “Writing Protocol Trace Plugins”.

Query Rewrite Plugins

MySQL Server supports query rewrite plugins that can examine and possibly modify statements received by the server before the server executes them. A query rewrite plugin takes statements either before or after the server has parsed them.

A preparse query rewrite plugin has these characteristics:

- The plugin enables rewriting of SQL statements arriving at the server before the server processes them.
- The plugin receives a statement string and may return a different string.

A postparse query rewrite plugin has these characteristics:

- The plugin enables statement rewriting based on parse trees.
- The server parses each statement and passes its parse tree to the plugin, which may traverse the tree. The plugin can return the original tree to the server for further processing, or construct a different tree and return that instead.
- The plugin can use the `mysql_parser` plugin service for these purposes:
  - To activate statement digest calculation and obtain the normalized version of statements independent of whether the Performance Schema produces digests.
  - To traverse parse trees.
  - To parse statements. This is useful if the plugin constructs a new statement string from the parse tree. The plugin can have the server parse the string to produce a new tree, then return that tree as the representation of the rewritten statement.

For more information about plugin services, see Section 29.3, “MySQL Services for Plugins”.

Preparse and postparse query rewrite plugins share these characteristics:

- If a query rewrite plugin is installed, the `--log-raw` option affects statement logging as follows:
  - Without `--log-raw`, the server logs the statement returned by the query rewrite plugin. This may differ from the statement as received.
  - With `--log-raw`, the server logs the original statement as received.
- If a plugin rewrites a statement, the server decides whether to write it to the binary log (and thus to any replication slaves) based on the rewritten statement, not the original statement. If a plugin rewrites only `SELECT` statements to `SELECT` statements, there is no impact on binary logging because the server does not write `SELECT` statements to the binary log.
- If a plugin rewrites a statement, the server produces a `Note` message that the client can view using `SHOW WARNINGS`. Messages have this format, where *`stmt_in`* is the original statement and *`stmt_out`* is the rewritten statement:

```
Query 'stmt_in' rewritten to 'stmt_out' by a query rewrite plugin
```

MySQL distributions include a postparse query rewrite plugin named `Rewriter`. This plugin is rule based. You can add rows to its rules table to cause `SELECT` statement rewriting. For more information, see Section 5.6.4, “The Rewriter Query Rewrite Plugin”.

Query rewrite plugins use the same API as audit plugins. For more information about audit plugins, see Section 29.2.4.8, “Writing Audit Plugins”.

Keyring Plugins

MySQL Server supports keyring plugins that enable internal server components and plugins to securely store sensitive information for later retrieval.

All MySQL distributions include a keyring plugin named `keyring_file`. MySQL Enterprise Edition distributions include additional keyring plugins. See Section 6.4.4, “The MySQL Keyring”.

For more information about keyring plugins, see Section 29.2.4.12, “Writing Keyring Plugins”.

29.2.2 Plugin API Characteristics

The server plugin API has these characteristics:

- All plugins have several things in common.

  Each plugin has a name that it can be referred to in SQL statements, as well as other metadata such as an author and a description that provide other information. This information can be examined in the `INFORMATION_SCHEMA.PLUGINS` table or using the `SHOW PLUGINS` statement.

- The plugin framework is extendable to accommodate different kinds of plugins.

  Although some aspects of the plugin API are common to all types of plugins, the API also permits type-specific interface elements so that different types of plugins can be created. A plugin with one purpose can have an interface most appropriate to its own requirements and not the requirements of some other plugin type.

  Interfaces for several types of plugins exist, such as storage engines, full-text parser, and `INFORMATION_SCHEMA` tables. Others can be added.

- Plugins can expose information to users.

  A plugin can implement system and status variables that are available through the `SHOW VARIABLES` and `SHOW STATUS` statements.

- The plugin API includes versioning information.

  The version information included in the plugin API enables a plugin library and each plugin that it contains to be self-identifying with respect to the API version that was used to build the library. If the API changes over time, the version numbers will change, but a server can examine a given plugin library's version information to determine whether it supports the plugins in the library.

  There are two types of version numbers. The first is the version for the general plugin framework itself. Each plugin library includes this kind of version number. The second type of version applies to individual plugins. Each specific type of plugin has a version for its interface, so each plugin in a library has a type-specific version number. For example, a library containing a full-text parser plugin has a general plugin API version number, and the plugin has a version number specific to the full-text plugin interface.

- The plugin API implements security restrictions.

  A plugin library must be installed in a specific dedicated directory for which the location is controlled by the server and cannot be changed at runtime. Also, the library must contain specific symbols that identify it as a plugin library. The server will not load something as a plugin if it was not built as a plugin.

- Plugins have access to server services.

  The services interface exposes server functionality that plugins can access using ordinary function calls. For details, see Section 29.3, “MySQL Services for Plugins”.

In some respects, the server plugin API is similar to the older user-defined function (UDF) API that it supersedes, but the plugin API has several advantages over the older interface. For example, UDFs had no versioning information. Also, the newer plugin interface eliminates the security issues of the older UDF interface. The older interface for writing nonplugin UDFs permitted libraries to be loaded from any directory searched by the system's dynamic linker, and the symbols that identified the UDF library were relatively nonspecific.

The client plugin API has similar architectural characteristics, but client plugins have no direct access to the server the way server plugins do.

29.2.3 Plugin API Components

The server plugin implementation comprises several components.

SQL statements:

- `INSTALL PLUGIN` registers a plugin in the `mysql.plugin` table and loads the plugin code.
- `UNINSTALL PLUGIN` unregisters a plugin from the `mysql.plugin` table and unloads the plugin code.
- The `WITH PARSER` clause for full-text index creation associates a full-text parser plugin with a given `FULLTEXT` index.
- `SHOW PLUGINS` displays information about server plugins.

Command-line options and system variables:

- The `--plugin-load` option enables plugins to be loaded at server startup time.

- The `plugin_dir` system variable indicates the location of the directory where all plugins must be installed. The value of this variable can be specified at server startup with a `--plugin_dir=dir_name` option. `mysql_config --plugindir` displays the default plugin directory path name.

For additional information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

Plugin-related tables:

- The `INFORMATION_SCHEMA.PLUGINS` table contains plugin information.
- The `mysql.plugin` table lists each plugin that was installed with `INSTALL PLUGIN` and is required for plugin use. For new MySQL installations, this table is created during the installation process.

The client plugin implementation is simpler:

- For the `mysql_options()` C API function, the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options enable client programs to load authentication plugins.
- There are C API functions that enable management of client plugins.

To examine how MySQL implements plugins, consult the following source files in a MySQL source distribution:

- In the `include/mysql` directory, `plugin.h` exposes the public plugin API. This file should be examined by anyone who wants to write a plugin library. `plugin_xxx.h` files provide additional information that pertains to specific types of plugins. `client_plugin.h` contains information specific to client plugins.
- In the `sql` directory, `sql_plugin.h` and `sql_plugin.cc` comprise the internal plugin implementation. `sql_acl.cc` is where the server uses authentication plugins. These files need not be consulted by plugin developers. They may be of interest for those who want to know more about how the server handles plugins.
- In the `sql-common` directory, `client_plugin.h` implements the C API client plugin functions, and `client.c` implements client authentication support. These files need not be consulted by plugin developers. They may be of interest for those who want to know more about how the server handles plugins.

29.2.4 Writing Plugins

To create a plugin library, you must provide the required descriptor information that indicates what plugins the library file contains, and write the interface functions for each plugin.

Every server plugin must have a general descriptor that provides information to the plugin API, and a type-specific descriptor that provides information about the plugin interface for a given type of plugin. The structure of the general descriptor is the same for all plugin types. The structure of the type-specific descriptor varies among plugin types and is determined by the requirements of what the plugin needs to do. The server plugin interface also enables plugins to expose status and system variables. These variables become visible through the `SHOW STATUS` and `SHOW VARIABLES` statements and the corresponding `INFORMATION_SCHEMA` tables.

For client-side plugins, the architecture is a bit different. Each plugin must have a descriptor, but there is no division into separate general and type-specific descriptors. Instead, the descriptor begins with a fixed set of members common to all client plugin types, and the common members are followed by any additional members required to implement the specific plugin type.

A server plugin contains code that becomes part of the running server, so when you write the plugin, you are bound by any and all constraints that otherwise apply to writing server code. For example, you may

have problems if you attempt to use functions from the `libstdc++` library. These constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to plugins originally written for older servers. For information about these constraints, see Section 2.9.7, “MySQL Source-Configuration Options”, and Section 2.9.8, “Dealing with Problems Compiling MySQL”.

Client plugin writers should avoid dependencies on what symbols the calling application has because you cannot be sure what applications will use the plugin.

29.2.4.1 Overview of Plugin Writing

These conditions apply to plugin writing:

- MySQL header files used by plugins contain C++ code, so plugins must be compiled as C++ code.
- You must compile plugins with the entire server source code present, not just the libraries and header files.
- Compiled plugins are not compatible across server versions. For a plugin compiled against MySQL 8.0.*X*, there is no guarantee it will work with a MySQL 8.0.*Y* server without recompiling for MySQL 8.0.*Y*.
- Plugins are loaded and unloaded dynamically, so your operating system must support dynamic loading and you must have compiled the calling application dynamically (not statically). For server plugins, this means that `mysqld` must be linked dynamically.

The following procedure provides an overview of the steps needed to create a plugin library. The next sections provide additional details on setting plugin data structures and writing specific types of plugins.

1. In the plugin source file, include the header files that the plugin library needs. The `plugin.h` file is required, and the library might require other files as well. For example:

   ```
   #include <stdlib.h>
   #include <ctype.h>
   #include <mysql/plugin.h>
   ```

2. Set up the descriptor information for the plugin library file. For server plugins, write the library descriptor, which must contain the general plugin descriptor for each server plugin in the file. For more information, see Server Plugin Library and Plugin Descriptors. In addition, set up the type-specific descriptor for each server plugin in the library. Each plugin's general descriptor points to its type-specific descriptor.

   For client plugins, write the client descriptor. For more information, see Client Plugin Descriptors.

3. Write the plugin interface functions for each plugin. For example, each plugin's general plugin descriptor points to the initialization and deinitialization functions that the server should invoke when it loads and unloads the plugin. The plugin's type-specific description may also point to interface functions.

4. For server plugins, set up the status and system variables, if there are any.

5. Compile the plugin library as a shared library and install it in the plugin directory. For more information, see Section 29.2.4.3, “Compiling and Installing Plugin Libraries”.

6. For server plugins, register the plugin with the server. For more information, see Section 5.6.1, “Installing and Uninstalling Plugins”.

7. Test the plugin to verify that it works properly.

29.2.4.2 Plugin Data Structures

A plugin library file includes descriptor information to indicate what plugins it contains.

If the plugin library contains any server plugins, it must include the following descriptor information:

- A library descriptor indicates the general server plugin API version number used by the library and contains a general plugin descriptor for each server plugin in the library. To provide the framework for this descriptor, invoke two macros from the `plugin.h` header file:

  ```
  mysql_declare_plugin(name)
   ... one or more server plugin descriptors here ...
  mysql_declare_plugin_end;
  ```

  The macros expand to provide a declaration for the API version automatically. You must provide the plugin descriptors.

- Within the library descriptor, each general server plugin is described by a `st_mysql_plugin` structure. This plugin descriptor structure contains information that is common to every type of server plugin: A value that indicates the plugin type; the plugin name, author, description, and license type; pointers to the initialization and deinitialization functions that the server invokes when it loads and unloads the plugin, and pointers to any status or system variables the plugin implements.

- Each general server plugin descriptor within the library descriptor also contains a pointer to a type-specific plugin descriptor. The structure of the type-specific descriptors varies from one plugin type to another because each type of plugin can have its own API. A type-specific plugin descriptor contains a type-specific API version number and pointers to the functions that are needed to implement that plugin type. For example, a full-text parser plugin has initialization and deinitialization functions, and a main parsing function. The server invokes these functions when it uses the plugin to parse text.

The plugin library also contains the interface functions that are referenced by the general and type-specific descriptors for each plugin in the library.

If the plugin library contains a client plugin, it must include a descriptor for the plugin. The descriptor begins with a fixed set of members common to all client plugins, followed by any members specific to the plugin type. To provide the descriptor framework, invoke two macros from the `client_plugin.h` header file:

```
mysql_declare_client_plugin(plugin_type)
   ... members common to all client plugins ...
   ... type-specific extra members ...
mysql_end_client_plugin;
```

The plugin library also contains any interface functions referenced by the client descriptor.

The `mysql_declare_plugin()` and `mysql_declare_client_plugin()` macros differ somewhat in how they can be invoked, which has implications for the contents of plugin libraries. The following guidelines summarize the rules:

- `mysql_declare_plugin()` and `mysql_declare_client_plugin()` can both be used in the same source file, which means that a plugin library can contain both server and client plugins. However, each of `mysql_declare_plugin()` and `mysql_declare_client_plugin()` can be used at most once.

- `mysql_declare_plugin()` permits multiple server plugin declarations, so a plugin library can contain multiple server plugins.

- `mysql_declare_client_plugin()` permits only a single client plugin declaration. To create multiple client plugins, separate plugin libraries must be used.

When a client program looks for a client plugin that is in a plugin library and not built into `libmysqlclient`, it looks for a file with a base name that is the same as the plugin name. For example, if a program needs to use a client authentication plugin named `auth_xxx` on a system that uses `.so` as the library suffix, it looks in the file named `auth_xxx.so`. (On OS X, the program looks first for `auth_xxx.dylib`, then for `auth_xxx.so`.) For this reason, if a plugin library contains a client plugin, the library must have the same base name as that plugin.

The same is not true for a library that contains server plugins. The `--plugin-load` option and the `INSTALL PLUGIN` statement provide the library file name explicitly, so there need be no explicit relationship between the library name and the name of any server plugins it contains.

Server Plugin Library and Plugin Descriptors

Every plugin library that contains server plugins must include a library descriptor that contains the general plugin descriptor for each server plugin in the file. This section discusses how to write the library and general descriptors for server plugins.

The library descriptor must define two symbols:

- `_mysql_plugin_interface_version_` specifies the version number of the general plugin framework. This is given by the `MYSQL_PLUGIN_INTERFACE_VERSION` symbol, which is defined in the `plugin.h` file.
- `_mysql_plugin_declarations_` defines an array of plugin declarations, terminated by a declaration with all members set to 0. Each declaration is an instance of the `st_mysql_plugin` structure (also defined in `plugin.h`). There must be one of these for each server plugin in the library.

If the server does not find those two symbols in a library, it does not accept it as a legal plugin library and rejects it with an error. This prevents use of a library for plugin purposes unless it was built specifically as a plugin library.

The conventional way to define the two required symbols is by using the `mysql_declare_plugin()` and `mysql_declare_plugin_end` macros from the `plugin.h` file:

```
mysql_declare_plugin(name)
 ... one or more server plugin descriptors here ...
mysql_declare_plugin_end;
```

Each server plugin must have a general descriptor that provides information to the server plugin API. The general descriptor has the same structure for all plugin types. The `st_mysql_plugin` structure in the `plugin.h` file defines this descriptor:

```
struct st_mysql_plugin
{
  int type;             /* the plugin type (a MYSQL_XXX_PLUGIN value)   */
  void *info;           /* pointer to type-specific plugin descriptor   */
  const char *name;     /* plugin name                                  */
  const char *author;   /* plugin author (for I_S.PLUGINS)              */
  const char *descr;    /* general descriptive text (for I_S.PLUGINS)   */
  int license;          /* the plugin license (PLUGIN_LICENSE_XXX)      */
  int (*init)(void *);  /* the function to invoke when plugin is loaded */
  int (*deinit)(void *);/* the function to invoke when plugin is unloaded */
  unsigned int version; /* plugin version (for I_S.PLUGINS)             */
  struct st_mysql_show_var *status_vars;
  struct st_mysql_sys_var **system_vars;
  void * __reserved1;   /* reserved for dependency checking             */
  unsigned long flags;  /* flags for plugin */
```

```
};
```

The `st_mysql_plugin` descriptor structure members are used as follows. `char *` members should be specified as null-terminated strings.

- `type`: The plugin type. This must be one of the plugin-type values from `plugin.h`:

  ```
  /*
    The allowable types of plugins
  */
  #define MYSQL_UDF_PLUGIN              0  /* User-defined function        */
  #define MYSQL_STORAGE_ENGINE_PLUGIN   1  /* Storage Engine               */
  #define MYSQL_FTPARSER_PLUGIN         2  /* Full-text parser plugin      */
  #define MYSQL_DAEMON_PLUGIN           3  /* The daemon/raw plugin type */
  #define MYSQL_INFORMATION_SCHEMA_PLUGIN  4  /* The I_S plugin type */
  #define MYSQL_AUDIT_PLUGIN            5  /* The Audit plugin type        */
  #define MYSQL_REPLICATION_PLUGIN      6  /* The replication plugin type */
  #define MYSQL_AUTHENTICATION_PLUGIN   7  /* The authentication plugin type */
  ...
  ```

  For example, for a full-text parser plugin, the `type` value is `MYSQL_FTPARSER_PLUGIN`.

- `info`: A pointer to the type-specific descriptor for the plugin. This descriptor's structure depends on the particular type of plugin, unlike that of the general plugin descriptor structure. For version-control purposes, the first member of the type-specific descriptor for every plugin type is expected to be the interface version for the type. This enables the server to check the type-specific version for every plugin no matter its type. Following the version number, the descriptor includes any other members needed, such as callback functions and other information needed by the server to invoke the plugin properly. Later sections on writing particular types of server plugins describe the structure of their type-specific descriptors.

- `name`: A string that gives the plugin name. This is the name that will be listed in the `mysql.plugin` table and by which you refer to the plugin in SQL statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`, or with the `--plugin-load` option. The name is also visible in the `INFORMATION_SCHEMA.PLUGINS` table or the output from `SHOW PLUGINS`.

  The plugin name should not begin with the name of any server option. If it does, the server will fail to initialize it. For example, the server has a `--socket` option, so you should not use a plugin name such as `socket`, `socket_plugin`, and so forth.

- `author`: A string naming the plugin author. This can be whatever you like.

- `desc`: A string that provides a general description of the plugin. This can be whatever you like.

- `license`: The plugin license type. The value can be one of `PLUGIN_LICENSE_PROPRIETARY`, `PLUGIN_LICENSE_GPL`, or `PLUGIN_LICENSE_BSD`.

- `init`: A once-only initialization function, or `NULL` if there is no such function. The server executes this function when it loads the plugin, which happens for `INSTALL PLUGIN` or, for plugins listed in the `mysql.plugin` table, at server startup. The function takes one argument that points to the internal structure used to identify the plugin. It returns zero for success and nonzero for failure.

- `deinit`: A once-only deinitialization function, or `NULL` if there is no such function. The server executes this function when it unloads the plugin, which happens for `UNINSTALL PLUGIN` or, for plugins listed in the `mysql.plugin` table, at server shutdown. The function takes one argument that points to the internal structure used to identify the plugin It returns zero for success and nonzero for failure.

- `version`: The plugin version number. When the plugin is installed, this value can be retrieved from the `INFORMATION_SCHEMA.PLUGINS` table. The value includes major and minor numbers. If you write

the value as a hex constant, the format is `0xMMNN`, where *`MM`* and `NN` are the major and minor numbers, respectively. For example, `0x0302` represents version 3.2.

- `status_vars`: A pointer to a structure for status variables associated with the plugin, or `NULL` if there are no such variables. When the plugin is installed, these variables are displayed in the output of the `SHOW STATUS` statement.

  The `status_vars` member, if not `NULL`, points to an array of `st_mysql_show_var` structures that describe status variables. See Server Plugin Status and System Variables.

- `system_vars`: A pointer to a structure for system variables associated with the plugin, or `NULL` if there are no such variables. These options and system variables can be used to help initialize variables within the plugin. When the plugin is installed, these variables are displayed in the output of the `SHOW VARIABLES` statement.

  The `system_vars` member, if not `NULL`, points to an array of `st_mysql_sys_var` structures that describe system variables. See Server Plugin Status and System Variables.

- `__reserved1`: A placeholder for the future. It should be set to `NULL`.

- `flags`: Plugin flags. Individual bits correspond to different flags. The value should be set to the OR of the applicable flags. These flags are available:

  ```
  #define PLUGIN_OPT_NO_INSTALL   1UL   /* Not dynamically loadable */
  #define PLUGIN_OPT_NO_UNINSTALL 2UL   /* Not dynamically unloadable */
  #define PLUGIN_OPT_ALLOW_EARLY  4UL   /* allow --early-plugin-load */
  ```

  The flags have the following meanings when enabled:

  - `PLUGIN_OPT_NO_INSTALL`: The plugin cannot be loaded at runtime with the `INSTALL PLUGIN` statement. This is appropriate for plugins that must be loaded at server startup with the `--plugin-load`, `--plugin-load-add`, or `--early-plugin-load` option.

  - `PLUGIN_OPT_NO_UNINSTALL`: The plugin cannot be unloaded at runtime with the `UNINSTALL PLUGIN` statement.

  - `PLUGIN_OPT_ALLOW_EARLY`: The plugin can be loaded early in the server startup sequence with the `--early-plugin-load` option. This flag has no effect on whether a plugin can be loaded at server startup with the `--plugin-load` or `--plugin-load-add` option, or at runtime with the `INSTALL PLUGIN` statement.

    This flag was added in MySQL 8.0.17. All plugins compiled using MySQL distributions prior to 8.0.17 do not have this flag set. When loading these into pre-8.0.17 servers this does not matter, but attempts to use `--early-plugin-load` to load plugin binaries compiled using pre-8.0.17 MySQL distributions into a 8.0.17 or higher server will fail. The plugins must be recompiled against MySQL 8.0.17 or higher.

The server invokes the `init` and `deinit` functions in the general plugin descriptor only when loading and unloading the plugin. They have nothing to do with use of the plugin such as happens when an SQL statement causes the plugin to be invoked.

For example, the descriptor information for a library that contains a single full-text parser plugin named `simple_parser` looks like this:

```
mysql_declare_plugin(ftexample)
{
```

```
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  "simple_parser",            /* name                            */
  "Oracle Corporation",       /* author                          */
  "Simple Full-Text Parser",  /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
mysql_declare_plugin_end;
```

For a full-text parser plugin, the type must be `MYSQL_FTPARSER_PLUGIN`. This is the value that identifies the plugin as being legal for use in a `WITH PARSER` clause when creating a `FULLTEXT` index. (No other plugin type is legal for this clause.)

`plugin.h` defines the `mysql_declare_plugin()` and `mysql_declare_plugin_end` macros like this:

```
#ifndef MYSQL_DYNAMIC_PLUGIN
#define __MYSQL_DECLARE_PLUGIN(NAME, VERSION, PSIZE, DECLS) \
MYSQL_PLUGIN_EXPORT int VERSION= MYSQL_PLUGIN_INTERFACE_VERSION; \
MYSQL_PLUGIN_EXPORT int PSIZE= sizeof(struct st_mysql_plugin); \
MYSQL_PLUGIN_EXPORT struct st_mysql_plugin DECLS[]= {
#else
#define __MYSQL_DECLARE_PLUGIN(NAME, VERSION, PSIZE, DECLS) \
MYSQL_PLUGIN_EXPORT int _mysql_plugin_interface_version_= MYSQL_PLUGIN_INTERFACE_VERSION; \
MYSQL_PLUGIN_EXPORT int _mysql_sizeof_struct_st_plugin_= sizeof(struct st_mysql_plugin); \
MYSQL_PLUGIN_EXPORT struct st_mysql_plugin _mysql_plugin_declarations_[]= {
#endif

#define mysql_declare_plugin(NAME) \
__MYSQL_DECLARE_PLUGIN(NAME, \
                builtin_ ## NAME ## _plugin_interface_version, \
                builtin_ ## NAME ## _sizeof_struct_st_plugin, \
                builtin_ ## NAME ## _plugin)

#define mysql_declare_plugin_end ,{0,0,0,0,0,0,0,0,0,0,0,0,0}}
```

Note

Those declarations define the `_mysql_plugin_interface_version_` symbol only if the `MYSQL_DYNAMIC_PLUGIN` symbol is defined. This means that `-DMYSQL_DYNAMIC_PLUGIN` must be provided as part of the compilation command to build the plugin as a shared library.

When the macros are used as just shown, they expand to the following code, which defines both of the required symbols (`_mysql_plugin_interface_version_` and `_mysql_plugin_declarations_`):

```
int _mysql_plugin_interface_version_= MYSQL_PLUGIN_INTERFACE_VERSION;
int _mysql_sizeof_struct_st_plugin_= sizeof(struct st_mysql_plugin);
struct st_mysql_plugin _mysql_plugin_declarations_[]= {
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  "simple_parser",            /* name                            */
  "Oracle Corporation",       /* author                          */
  "Simple Full-Text Parser",  /* description                     */
```

```
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
  ,{0,0,0,0,0,0,0,0,0,0,0,0}}
};
```

The preceding example declares a single plugin in the general descriptor, but it is possible to declare multiple plugins. List the declarations one after the other between `mysql_declare_plugin()` and `mysql_declare_plugin_end`, separated by commas.

MySQL server plugins must be compiled as C++ code. One C++ feature that you should not use is nonconstant variables to initialize global structures. Members of structures such as the `st_mysql_plugin` structure should be initialized only with constant variables. The `simple_parser` descriptor shown earlier is permissible in a C++ plugin because it satisfies that requirement:

```
mysql_declare_plugin(ftexample)
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  "simple_parser",            /* name                            */
  "Oracle Corporation",       /* author                          */
  "Simple Full-Text Parser",  /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
mysql_declare_plugin_end;
```

Here is another valid way to write the general descriptor. It uses constant variables to indicate the plugin name, author, and description:

```
const char *simple_parser_name = "simple_parser";
const char *simple_parser_author = "Oracle Corporation";
const char *simple_parser_description = "Simple Full-Text Parser";

mysql_declare_plugin(ftexample)
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  simple_parser_name,         /* name                            */
  simple_parser_author,       /* author                          */
  simple_parser_description,  /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
mysql_declare_plugin_end;
```

However, the following general descriptor is invalid. It uses structure members to indicate the plugin name, author, and description, but structures are not considered constant initializers in C++:

```
typedef struct
{
  const char *name;
  const char *author;
  const char *description;
} plugin_info;

plugin_info parser_info = {
  "simple_parser",
  "Oracle Corporation",
  "Simple Full-Text Parser"
};

mysql_declare_plugin(ftexample)
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  parser_info.name,           /* name                            */
  parser_info.author,         /* author                          */
  parser_info.description,    /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
mysql_declare_plugin_end;
```

Server Plugin Status and System Variables

The server plugin interface enables plugins to expose status and system variables using the `status_vars` and `system_vars` members of the general plugin descriptor.

The `status_vars` member of the general plugin descriptor, if not 0, points to an array of `st_mysql_show_var` structures, each of which describes one status variable, followed by a structure with all members set to 0. The `st_mysql_show_var` structure has this definition:

```
struct st_mysql_show_var {
  const char *name;
  char *value;
  enum enum_mysql_show_type type;
};
```

The following table shows the permissible status variable `type` values and what the corresponding variable should be.

Table 29.1 Server Plugin Status Variable Types

Variable Type	Meaning
`SHOW_BOOL`	Pointer to a boolean variable
`SHOW_INT`	Pointer to an integer variable
`SHOW_LONG`	Pointer to a long integer variable
`SHOW_LONGLONG`	Pointer to a longlong integer variable

Variable Type	Meaning
SHOW_CHAR	A string
SHOW_CHAR_PTR	Pointer to a string
SHOW_ARRAY	Pointer to another st_mysql_show_var array
SHOW_FUNC	Pointer to a function
SHOW_DOUBLE	Pointer to a double

For the SHOW_FUNC type, the function is called and fills in its out parameter, which then provides information about the variable to be displayed. The function has this signature:

```
#define SHOW_VAR_FUNC_BUFF_SIZE 1024

typedef int (*mysql_show_var_func) (void *thd,
                                    struct st_mysql_show_var *out,
                                    char *buf);
```

The system_vars member, if not 0, points to an array of st_mysql_sys_var structures, each of which describes one system variable (which can also be set from the command-line or configuration file), followed by a structure with all members set to 0. The st_mysql_sys_var structure is defined as follows:

```
struct st_mysql_sys_var {
 int flags;
 const char *name, *comment;
 int (*check)(THD*, struct st_mysql_sys_var *, void*, st_mysql_value*);
 void (*update)(THD*, struct st_mysql_sys_var *, void*, const void*);
};
```

Additional fields are append as required depending upon the flags.

For convenience, a number of macros are defined that make creating new system variables within a plugin much simpler.

Throughout the macros, the following fields are available:

- name: An unquoted identifier for the system variable.
- varname: The identifier for the static variable. Where not available, it is the same as the name field.
- opt: Additional use flags for the system variable. The following table shows the permissible flags.

Table 29.2 Server Plugin System Variable Flags

Flag Value	Description
PLUGIN_VAR_READONLY	The system variable is read only
PLUGIN_VAR_NOSYSVAR	The system variable is not user visible at runtime
PLUGIN_VAR_NOCMDOPT	The system variable is not configurable from the command line
PLUGIN_VAR_NOCMDARG	No argument is required at the command line (typically used for boolean variables)
PLUGIN_VAR_RQCMDARG	An argument is required at the command line (this is the default)
PLUGIN_VAR_OPCMDARG	An argument is optional at the command line
PLUGIN_VAR_MEMALLOC	Used for string variables; indicates that memory is to be allocated for storage of the string

- `comment`: A descriptive comment to be displayed in the server help message. `NULL` if this variable is to be hidden.
- `check`: The check function, `NULL` for default.
- `update`: The update function, `NULL` for default.
- `default`: The variable default value.
- `minimum`: The variable minimum value.
- `maximum`: The variable maximum value.
- `blocksize`: The variable block size. When the value is set, it is rounded to the nearest multiple of `blocksize`.

A system variable may be accessed either by using the static variable directly or by using the `SYSVAR()`accessor macro. The `SYSVAR()` macro is provided for completeness. Usually it should be used only when the code cannot directly access the underlying variable.

For example:

```
static int my_foo;
static MYSQL_SYSVAR_INT(foo_var, my_foo,
                        PLUGIN_VAR_RQCMDARG, "foo comment",
                        NULL, NULL, 0, 0, INT_MAX, 0);
 ...
   SYSVAR(foo_var)= value;
   value= SYSVAR(foo_var);
   my_foo= value;
   value= my_foo;
```

Session variables may be accessed only through the `THDVAR()` accessor macro. For example:

```
static MYSQL_THDVAR_BOOL(some_flag,
                         PLUGIN_VAR_NOCMDARG, "flag comment",
                         NULL, NULL, FALSE);
 ...
   if (THDVAR(thd, some_flag))
   {
     do_something();
     THDVAR(thd, some_flag)= FALSE;
   }
```

All global and session system variables must be published to `mysqld` before use. This is done by constructing a `NULL`-terminated array of the variables and linking to it in the plugin public interface. For example:

```
static struct st_mysql_sys_var *my_plugin_vars[]= {
  MYSQL_SYSVAR(foo_var),
  MYSQL_SYSVAR(some_flag),
  NULL
};
mysql_declare_plugin(fooplug)
{
  MYSQL_..._PLUGIN,
  &plugin_data,
  "fooplug",
  "foo author",
  "This does foo!",
  PLUGIN_LICENSE_GPL,
```

```
  foo_init,
  foo_fini,
  0x0001,
  NULL,
  my_plugin_vars,
  NULL,
  0
}
mysql_declare_plugin_end;
```

The following convenience macros enable you to declare different types of system variables:

- Boolean system variables of type `bool`, which is a 1-byte boolean. (0 = `false`, 1 = `true`)

```
MYSQL_THDVAR_BOOL(name, opt, comment, check, update, default)
MYSQL_SYSVAR_BOOL(name, varname, opt, comment, check, update, default)
```

- String system variables of type `char*`, which is a pointer to a null-terminated string.

```
MYSQL_THDVAR_STR(name, opt, comment, check, update, default)
MYSQL_SYSVAR_STR(name, varname, opt, comment, check, update, default)
```

- Integer system variables, of which there are several varieties.
  - An `int` system variable, which is typically a 4-byte signed word.

```
MYSQL_THDVAR_INT(name, opt, comment, check, update, default, min, max, blk)
MYSQL_SYSVAR_INT(name, varname, opt, comment, check, update, default,
                 minimum, maximum, blocksize)
```

  - An `unsigned int` system variable, which is typically a 4-byte unsigned word.

```
MYSQL_THDVAR_UINT(name, opt, comment, check, update, default, min, max, blk)
MYSQL_SYSVAR_UINT(name, varname, opt, comment, check, update, default,
                  minimum, maximum, blocksize)
```

  - A `long` system variable, which is typically either a 4- or 8-byte signed word.

```
MYSQL_THDVAR_LONG(name, opt, comment, check, update, default, min, max, blk)
MYSQL_SYSVAR_LONG(name, varname, opt, comment, check, update, default,
                  minimum, maximum, blocksize)
```

  - An `unsigned long` system variable, which is typically either a 4- or 8-byte unsigned word.

```
MYSQL_THDVAR_ULONG(name, opt, comment, check, update, default, min, max, blk)
MYSQL_SYSVAR_ULONG(name, varname, opt, comment, check, update, default,
                   minimum, maximum, blocksize)
```

  - A `long long` system variable, which is typically an 8-byte signed word.

```
MYSQL_THDVAR_LONGLONG(name, opt, comment, check, update,
                      default, minimum, maximum, blocksize)
MYSQL_SYSVAR_LONGLONG(name, varname, opt, comment, check, update,
                      default, minimum, maximum, blocksize)
```

  - An `unsigned long long` system variable, which is typically an 8-byte unsigned word.

```
```

```
MYSQL_THDVAR_ULONGLONG(name, opt, comment, check, update,
                       default, minimum, maximum, blocksize)
MYSQL_SYSVAR_ULONGLONG(name, varname, opt, comment, check, update,
                       default, minimum, maximum, blocksize)
```

- A `double` system variable, which is typically an 8-byte signed word.

```
MYSQL_THDVAR_DOUBLE(name, opt, comment, check, update,
                    default, minimum, maximum, blocksize)
MYSQL_SYSVAR_DOUBLE(name, varname, opt, comment, check, update,
                    default, minimum, maximum, blocksize)
```

- An `unsigned long` system variable, which is typically either a 4- or 8-byte unsigned word. The range of possible values is an ordinal of the number of elements in the `typelib`, starting from 0.

```
MYSQL_THDVAR_ENUM(name, opt, comment, check, update, default, typelib)
MYSQL_SYSVAR_ENUM(name, varname, opt, comment, check, update,
                  default, typelib)
```

- An `unsigned long long` system variable, which is typically an 8-byte unsigned word. Each bit represents an element in the `typelib`.

```
MYSQL_THDVAR_SET(name, opt, comment, check, update, default, typelib)
MYSQL_SYSVAR_SET(name, varname, opt, comment, check, update,
                 default, typelib)
```

Internally, all mutable and plugin system variables are stored in a `HASH` structure.

Display of the server command-line help text is handled by compiling a `DYNAMIC_ARRAY` of all variables relevant to command-line options, sorting them, and then iterating through them to display each option.

When a command-line option has been handled, it is then removed from the `argv` by the `handle_option()` function (`my_getopt.c`); in effect, it is consumed.

The server processes command-line options during the plugin installation process, immediately after the plugin has been successfully loaded but before the plugin initialization function has been called

Plugins loaded at runtime do not benefit from any configuration options and must have usable defaults. Once they are installed, they are loaded at `mysqld` initialization time and configuration options can be set at the command line or within `my.cnf`.

Plugins should consider the `thd` parameter to be read only.

Client Plugin Descriptors

Each client plugin must have a descriptor that provides information to the client plugin API. The descriptor structure begins with a fixed set of members common to all client plugins, followed by any members specific to the plugin type.

The `st_mysql_client_plugin` structure in the `client_plugin.h` file defines a “generic” descriptor that contains the common members:

```
struct st_mysql_client_plugin
{
  int type;
  unsigned int interface_version;
  const char *name;
  const char *author;
```

```
  const char *desc;
  unsigned int version[3];
  const char *license;
  void *mysql_api;
  int (*init)(char *, size_t, int, va_list);
  int (*deinit)();
  int (*options)(const char *option, const void *);
};
```

The common `st_mysql_client_plugin` descriptor structure members are used as follows. `char *` members should be specified as null-terminated strings.

- `type`: The plugin type. This must be one of the plugin-type values from `client_plugin.h`, such as `MYSQL_CLIENT_AUTHENTICATION_PLUGIN`.
- `interface_version`: The plugin interface version. For example, this is `MYSQL_CLIENT_AUTHENTICATION_PLUGIN_INTERFACE_VERSION` for an authentication plugin.
- `name`: A string that gives the plugin name. This is the name by which you refer to the plugin when you call `mysql_options()` with the `MYSQL_DEFAULT_AUTH` option or specify the `--default-auth` option to a MySQL client program.
- `author`: A string naming the plugin author. This can be whatever you like.
- `desc`: A string that provides a general description of the plugin. This can be whatever you like.
- `version`: The plugin version as an array of three integers indicating the major, minor, and teeny versions. For example, `{1,2,3}` indicates version 1.2.3.
- `license`: A string that specifies the license type.
- `mysql_api`: For internal use. Specify it as `NULL` in the plugin descriptor.
- `init`: A once-only initialization function, or `NULL` if there is no such function. The client library executes this function when it loads the plugin. The function returns zero for success and nonzero for failure.

  The `init` function uses its first two arguments to return an error message if an error occurs. The first argument is a pointer to a `char` buffer, and the second argument indicates the buffer length. Any message returned by the `init` function must be null-terminated, so the maximum message length is the buffer length minus one. The next arguments are passed to `mysql_load_plugin()`. The first indicates how many more arguments there are (0 if none), followed by any remaining arguments.
- `deinit`: A once-only deinitialization function, or `NULL` if there is no such function. The client library executes this function when it unloads the plugin. The function takes no arguments. It returns zero for success and nonzero for failure.
- `options`: A function for handling options passed to the plugin, or `NULL` if there is no such function. The function takes two arguments representing the option name and a pointer to its value. The function returns zero for success and nonzero for failure.

For a given client plugin type, the common descriptor members may be followed by additional members necessary to implement plugins of that type. For example, the `st_mysql_client_plugin_AUTHENTICATION` structure for authentication plugins has a function at the end that the client library calls to perform authentication.

To declare a plugin, use the `mysql_declare_client_plugin()` and `mysql_end_client_plugin` macros:

```
mysql_declare_client_plugin(plugin_type)
```

```
   ... members common to all client plugins ...
   ... type-specific extra members ...
mysql_end_client_plugin;
```

Do not specify the `type` or `interface_version` member explicitly. The `mysql_declare_client_plugin()` macro uses the *`plugin_type`* argument to generate their values automatically. For example, declare an authentication client plugin like this:

```
mysql_declare_client_plugin(AUTHENTICATION)
  "my_auth_plugin",
  "Author Name",
  "My Client Authentication Plugin",
  {1,0,0},
  "GPL",
  NULL,
  my_auth_init,
  my_auth_deinit,
  my_auth_options,
  my_auth_main
mysql_end_client_plugin;
```

This declaration uses the `AUTHENTICATION` argument to set the `type` and `interface_version` members to `MYSQL_CLIENT_AUTHENTICATION_PLUGIN` and `MYSQL_CLIENT_AUTHENTICATION_PLUGIN_INTERFACE_VERSION`.

Depending on the plugin type, the descriptor may have other members following the common members. For example, for an authentication plugin, there is a function (`my_auth_main()` in the descriptor just shown) that handles communication with the server. See Section 29.2.4.9, “Writing Authentication Plugins”.

Normally, a client program that supports the use of authentication plugins causes a plugin to be loaded by calling `mysql_options()` to set the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options:

```
char *plugin_dir = "path_to_plugin_dir";
char *default_auth = "plugin_name";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
mysql_options(&mysql, MYSQL_DEFAULT_AUTH, default_auth);
```

Typically, the program will also accept `--plugin-dir` and `--default-auth` options that enable users to override the default values.

Should a client program require lower-level plugin management, the client library contains functions that take an `st_mysql_client_plugin` argument. See Section 28.7.17, “C API Client Plugin Functions”.

29.2.4.3 Compiling and Installing Plugin Libraries

After your plugin is written, you must compile it and install it. The procedure for compiling shared objects varies from system to system. If you build your library using `CMake`, it should be able to generate the correct compilation commands for your system. If the library is named `somepluglib`, you should end up with a shared library file that has a name something like `somepluglib.so`. (The `.so` file name suffix might differ on your system.)

To use `CMake`, you'll need to set up the configuration files to enable the plugin to be compiled and installed. Use the plugin examples under the `plugin` directory of a MySQL source distribution as a guide.

Create `CMakeLists.txt`, which should look something like this:

```
```

```
MYSQL_ADD_PLUGIN(somepluglib somepluglib.c
  MODULE_ONLY MODULE_OUTPUT_NAME "somepluglib")
```

When `CMake` generates the `Makefile`, it should take care of passing to the compilation command the `-DMYSQL_DYNAMIC_PLUGIN` flag, and passing to the linker the `-lmysqlservices` flag, which is needed to link in any functions from services provided through the plugin services interface. See Section 29.3, “MySQL Services for Plugins”.

Run `CMake`, then run `make`:

```
shell> cmake .
shell> make
```

If you need to specify configuration options to `CMake`, see Section 2.9.7, “MySQL Source-Configuration Options”, for a list. For example, you might want to specify `CMAKE_INSTALL_PREFIX` to indicate the MySQL base directory under which the plugin should be installed. You can see what value to use for this option with `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'basedir';
+---------------+-----------------+
| Variable_name | Value           |
+---------------+-----------------+
| base          | /usr/local/mysql |
+---------------+-----------------+
```

The location of the plugin directory where you should install the library is given by the `plugin_dir` system variable. For example:

```
mysql> SHOW VARIABLES LIKE 'plugin_dir';
+---------------+-----------------------------------+
| Variable_name | Value                             |
+---------------+-----------------------------------+
| plugin_dir    | /usr/local/mysql/lib/mysql/plugin |
+---------------+-----------------------------------+
```

To install the plugin library, use `make`:

```
shell> make install
```

Verify that `make install` installed the plugin library in the proper directory. After installing it, make sure that the library permissions permit it to be executed by the server.

29.2.4.4 Writing Full-Text Parser Plugins

MySQL supports server-side full-text parser plugins with `MyISAM` and `InnoDB`. For introductory information about full-text parser plugins, see Full-Text Parser Plugins.

A full-text parser plugin can be used to replace or modify the built-in full-text parser. This section describes how to write a full-text parser plugin named `simple_parser`. This plugin performs parsing based on simpler rules than those used by the MySQL built-in full-text parser: Words are nonempty runs of whitespace characters.

The instructions use the source code in the `plugin/fulltext` directory of MySQL source distributions, so change location into that directory. The following procedure describes how the plugin library is created:

1. To write a full-text parser plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin.h>
```

`plugin.h` defines the `MYSQL_FTPARSER_PLUGIN` server plugin type and the data structures needed to declare the plugin.

2. Set up the library descriptor for the plugin library file.

   This descriptor contains the general plugin descriptor for the server plugin. For a full-text parser plugin, the type must be `MYSQL_FTPARSER_PLUGIN`. This is the value that identifies the plugin as being legal for use in a `WITH PARSER` clause when creating a `FULLTEXT` index. (No other plugin type is legal for this clause.)

   For example, the library descriptor for a library that contains a single full-text parser plugin named `simple_parser` looks like this:

```
mysql_declare_plugin(ftexample)
{
  MYSQL_FTPARSER_PLUGIN,      /* type                            */
  &simple_parser_descriptor,  /* descriptor                      */
  "simple_parser",            /* name                            */
  "Oracle Corporation",       /* author                          */
  "Simple Full-Text Parser",  /* description                     */
  PLUGIN_LICENSE_GPL,         /* plugin license                  */
  simple_parser_plugin_init,  /* init function (when loaded)     */
  simple_parser_plugin_deinit,/* deinit function (when unloaded) */
  0x0001,                     /* version                         */
  simple_status,              /* status variables                */
  simple_system_variables,    /* system variables                */
  NULL,
  0
}
mysql_declare_plugin_end;
```

   The `name` member (`simple_parser`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `SHOW PLUGINS` or `INFORMATION_SCHEMA.PLUGINS`.

   For more information, see Server Plugin Library and Plugin Descriptors.

3. Set up the type-specific plugin descriptor.

   Each general plugin descriptor in the library descriptor points to a type-specific descriptor. For a full-text parser plugin, the type-specific descriptor is an instance of the `st_mysql_ftparser` structure in the `plugin.h` file:

```
struct st_mysql_ftparser
{
  int interface_version;
  int (*parse)(MYSQL_FTPARSER_PARAM *param);
  int (*init)(MYSQL_FTPARSER_PARAM *param);
  int (*deinit)(MYSQL_FTPARSER_PARAM *param);
};
```

   As shown by the structure definition, the descriptor has an interface version number and contains pointers to three functions.

   The interface version number is specified using a symbol, which is in the form: `MYSQL_xxx_INTERFACE_VERSION`. For full-text parser plugins, the symbol is

`MYSQL_FTPARSER_INTERFACE_VERSION`. In the source code, you will find the actual interface version number for the full-text parser plugin defined in `include/mysql/plugin_ftparser.h`. The current interface version number is `0x0101`.

The `init` and `deinit` members should point to a function or be set to 0 if the function is not needed. The `parse` member must point to the function that performs the parsing.

In the `simple_parser` declaration, that descriptor is indicated by `&simple_parser_descriptor`. The descriptor specifies the version number for the full-text plugin interface (as given by `MYSQL_FTPARSER_INTERFACE_VERSION`), and the plugin's parsing, initialization, and deinitialization functions:

```
static struct st_mysql_ftparser simple_parser_descriptor=
{
  MYSQL_FTPARSER_INTERFACE_VERSION, /* interface version      */
  simple_parser_parse,              /* parsing function       */
  simple_parser_init,               /* parser init function   */
  simple_parser_deinit              /* parser deinit function */
};
```

A full-text parser plugin is used in two different contexts, indexing and searching. In both contexts, the server calls the initialization and deinitialization functions at the beginning and end of processing each SQL statement that causes the plugin to be invoked. However, during statement processing, the server calls the main parsing function in context-specific fashion:

- For indexing, the server calls the parser for each column value to be indexed.
- For searching, the server calls the parser to parse the search string. The parser might also be called for rows processed by the statement. In natural language mode, there is no need for the server to call the parser. For boolean mode phrase searches or natural language searches with query expansion, the parser is used to parse column values for information that is not in the index. Also, if a boolean mode search is done for a column that has no `FULLTEXT` index, the built-in parser will be called. (Plugins are associated with specific indexes. If there is no index, no plugin is used.)

The plugin declaration in the general plugin descriptor has `init` and `deinit` members that point initialization and deinitialization functions, and so does the type-specific plugin descriptor to which it points. However, these pairs of functions have different purposes and are invoked for different reasons:

- For the plugin declaration in the general plugin descriptor, the initialization and deinitialization functions are invoked when the plugin is loaded and unloaded.
- For the type-specific plugin descriptor, the initialization and deinitialization functions are invoked per SQL statement for which the plugin is used.

Each interface function named in the plugin descriptor should return zero for success or nonzero for failure, and each of them receives an argument that points to a `MYSQL_FTPARSER_PARAM` structure containing the parsing context. The structure has this definition:

```
typedef struct st_mysql_ftparser_param
{
  int (*mysql_parse)(struct st_mysql_ftparser_param *,
                     char *doc, int doc_len);
  int (*mysql_add_word)(struct st_mysql_ftparser_param *,
                        char *word, int word_len,
                        MYSQL_FTPARSER_BOOLEAN_INFO *boolean_info);
  void *ftparser_state;
  void *mysql_ftparam;
```

```
  struct charset_info_st *cs;
  char *doc;
  int length;
  int flags;
  enum enum_ftparser_mode mode;
} MYSQL_FTPARSER_PARAM;
```

The structure members are used as follows:

- `mysql_parse`: A pointer to a callback function that invokes the server's built-in parser. Use this callback when the plugin acts as a front end to the built-in parser. That is, when the plugin parsing function is called, it should process the input to extract the text and pass the text to the `mysql_parse` callback.

  The first parameter for this callback function should be the `param` value itself:

  ```
  param->mysql_parse(param, ...);
  ```

  A front end plugin can extract text and pass it all at once to the built-in parser, or it can extract and pass text to the built-in parser a piece at a time. However, in this case, the built-in parser treats the pieces of text as though there are implicit word breaks between them.

- `mysql_add_word`: A pointer to a callback function that adds a word to a full-text index or to the list of search terms. Use this callback when the parser plugin replaces the built-in parser. That is, when the plugin parsing function is called, it should parse the input into words and invoke the `mysql_add_word` callback for each word.

  The first parameter for this callback function should be the `param` value itself:

  ```
  param->mysql_add_word(param, ...);
  ```

- `ftparser_state`: This is a generic pointer. The plugin can set it to point to information to be used internally for its own purposes.

- `mysql_ftparam`: This is set by the server. It is passed as the first argument to the `mysql_parse` or `mysql_add_word` callback.

- `cs`: A pointer to information about the character set of the text, or 0 if no information is available.

- `doc`: A pointer to the text to be parsed.

- `length`: The length of the text to be parsed, in bytes.

- `flags`: Parser flags. This is zero if there are no special flags. The only nonzero flag is `MYSQL_FTFLAGS_NEED_COPY`, which means that `mysql_add_word()` must save a copy of the word (that is, it cannot use a pointer to the word because the word is in a buffer that will be overwritten.)

  This flag might be set or reset by MySQL before calling the parser plugin, by the parser plugin itself, or by the `mysql_parse()` function.

- `mode`: The parsing mode. This value will be one of the following constants:

  - `MYSQL_FTPARSER_SIMPLE_MODE`: Parse in fast and simple mode, which is used for indexing and for natural language queries. The parser should pass to the server only those words that should

be indexed. If the parser uses length limits or a stopword list to determine which words to ignore, it should not pass such words to the server.

- `MYSQL_FTPARSER_WITH_STOPWORDS`: Parse in stopword mode. This is used in boolean searches for phrase matching. The parser should pass all words to the server, even stopwords or words that are outside any normal length limits.
- `MYSQL_FTPARSER_FULL_BOOLEAN_INFO`: Parse in boolean mode. This is used for parsing boolean query strings. The parser should recognize not only words but also boolean-mode operators and pass them to the server as tokens using the `mysql_add_word` callback. To tell the server what kind of token is being passed, the plugin needs to fill in a `MYSQL_FTPARSER_BOOLEAN_INFO` structure and pass a pointer to it.

Note

For `MyISAM`, the stopword list and `ft_min_word_len` and `ft_max_word_len` are checked inside the tokenizer. For `InnoDB`, the stopword list and equivalent word length variable settings (`innodb_ft_min_token_size` and `innodb_ft_max_token_size`) are checked outside of the tokenizer. As a result, `InnoDB` plugin parsers do not need to check the stopword list, `innodb_ft_min_token_size`, or `innodb_ft_max_token_size`. Instead, it is recommended that all words be returned to `InnoDB`. However, if you want to check stopwords within your plugin parser, use `MYSQL_FTPARSER_SIMPLE_MODE`, which is for full-text search index and natural language search. For `MYSQL_FTPARSER_WITH_STOPWORDS` and `MYSQL_FTPARSER_FULL_BOOLEAN_INFO` modes, it is recommended that all words be returned to `InnoDB` including stopwords, in case of phrase searches.

If the parser is called in boolean mode, the `param->mode` value will be `MYSQL_FTPARSER_FULL_BOOLEAN_INFO`. The `MYSQL_FTPARSER_BOOLEAN_INFO` structure that the parser uses for passing token information to the server looks like this:

```
typedef struct st_mysql_ftparser_boolean_info
{
  enum enum_ft_token_type type;
  int yesno;
  int weight_adjust;
  char wasign;
  char trunc;
  int position;
  /* These are parser state and must be removed. */
  char prev;
  char *quot;
} MYSQL_FTPARSER_BOOLEAN_INFO;
```

The parser should fill in the structure members as follows:

- `type`: The token type. The following table shows the permissible types.

Table 29.3 Full-Text Parser Token Types

Token Value	Meaning
`FT_TOKEN_EOF`	End of data
`FT_TOKEN_WORD`	A regular word
`FT_TOKEN_LEFT_PAREN`	The beginning of a group or subexpression

Token Value	Meaning
FT_TOKEN_RIGHT_PAREN	The end of a group or subexpression
FT_TOKEN_STOPWORD	A stopword

- yesno: Whether the word must be present for a match to occur. 0 means that the word is optional but increases the match relevance if it is present. Values larger than 0 mean that the word must be present. Values smaller than 0 mean that the word must not be present.
- weight_adjust: A weighting factor that determines how much a match for the word counts. It can be used to increase or decrease the word's importance in relevance calculations. A value of zero indicates no weight adjustment. Values greater than or less than zero mean higher or lower weight, respectively. The examples at Section 12.9.2, “Boolean Full-Text Searches”, that use the < and > operators illustrate how weighting works.
- wasign: The sign of the weighting factor. A negative value acts like the ~ boolean-search operator, which causes the word's contribution to the relevance to be negative.
- trunc: Whether matching should be done as if the boolean-mode * truncation operator had been given.
- position: Start position of the word in the document, in bytes. Used by InnoDB full-text search. For existing plugins that are called in boolean mode, support must be added for the position member.

Plugins should not use the prev and quot members of the MYSQL_FTPARSER_BOOLEAN_INFO structure.

Note

The plugin parser framework does not support:

- The @distance boolean operator.
- A leading plus sign (+) or minus sign (-) boolean operator followed by a space and then a word ('+ apple' or '- apple'). The leading plus or minus sign must be directly adjacent to the word, for example: '+apple' or '-apple'.

For information about boolean full-text search operators, see Section 12.9.2, “Boolean Full-Text Searches”.

4. Set up the plugin interface functions.

   The general plugin descriptor in the library descriptor names the initialization and deinitialization functions that the server should invoke when it loads and unloads the plugin. For simple_parser, these functions do nothing but return zero to indicate that they succeeded:

```
static int simple_parser_plugin_init(void *arg __attribute__((unused)))
{
  return(0);
}

static int simple_parser_plugin_deinit(void *arg __attribute__((unused)))
{
  return(0);
}
```

Because those functions do not actually do anything, you could omit them and specify 0 for each of them in the plugin declaration.

The type-specific plugin descriptor for `simple_parser` names the initialization, deinitialization, and parsing functions that the server invokes when the plugin is used. For `simple_parser`, the initialization and deinitialization functions do nothing:

```
static int simple_parser_init(MYSQL_FTPARSER_PARAM *param
                              __attribute__((unused)))
{
  return(0);
}

static int simple_parser_deinit(MYSQL_FTPARSER_PARAM *param
                                __attribute__((unused)))
{
  return(0);
}
```

Here too, because those functions do nothing, you could omit them and specify 0 for each of them in the plugin descriptor.

The main parsing function, `simple_parser_parse()`, acts as a replacement for the built-in full-text parser, so it needs to split text into words and pass each word to the server. The parsing function's first argument is a pointer to a structure that contains the parsing context. This structure has a `doc` member that points to the text to be parsed, and a `length` member that indicates how long the text is. The simple parsing done by the plugin considers nonempty runs of whitespace characters to be words, so it identifies words like this:

```
static int simple_parser_parse(MYSQL_FTPARSER_PARAM *param)
{
  char *end, *start, *docend= param->doc + param->length;

  for (end= start= param->doc;; end++)
  {
    if (end == docend)
    {
      if (end > start)
        add_word(param, start, end - start);
      break;
    }
    else if (isspace(*end))
    {
      if (end > start)
        add_word(param, start, end - start);
      start= end + 1;
    }
  }
  return(0);
}
```

As the parser finds each word, it invokes a function `add_word()` to pass the word to the server. `add_word()` is a helper function only; it is not part of the plugin interface. The parser passes the parsing context pointer to `add_word()`, as well as a pointer to the word and a length value:

```
static void add_word(MYSQL_FTPARSER_PARAM *param, char *word, size_t len)
{
  MYSQL_FTPARSER_BOOLEAN_INFO bool_info=
    { FT_TOKEN_WORD, 0, 0, 0, 0, 0, ' ', 0 };
```

```
  param->mysql_add_word(param, word, len, &bool_info);
}
```

For boolean-mode parsing, `add_word()` fills in the members of the `bool_info` structure as described earlier in the discussion of the `st_mysql_ftparser_boolean_info` structure.

5. Set up the status variables. For the `simple_parser` plugin, the following status variable array sets up one status variable with a value that is static text, and another with a value that is stored in a long integer variable:

```
long number_of_calls= 0;

struct st_mysql_show_var simple_status[]=
{
  {"simple_parser_static", (char *)"just a static text", SHOW_CHAR},
  {"simple_parser_called", (char *)&number_of_calls,     SHOW_LONG},
  {0,0,0}
};
```

By using status variable names that begin with the plugin name, you can easily display the variables for a plugin with `SHOW STATUS`:

```
mysql> SHOW STATUS LIKE 'simple_parser%';
+----------------------+--------------------+
| Variable_name        | Value              |
+----------------------+--------------------+
| simple_parser_static | just a static text |
| simple_parser_called | 0                  |
+----------------------+--------------------+
```

6. To compile and install a plugin library file, use the instructions in Section 29.2.4.3, “Compiling and Installing Plugin Libraries”. To make the library file available for use, install it in the plugin directory (the directory named by the `plugin_dir` system variable). For the `simple_parser` plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of `mypluglib.so` (the `.so` suffix might differ depending on your platform).

7. To use the plugin, register it with the server. For example, to register the plugin at runtime, use this statement (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN simple_parser SONAME 'mypluglib.so';
```

For additional information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

8. To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement. See Section 5.6.2, “Obtaining Server Plugin Information”.

9. Test the plugin to verify that it works properly.

Create a table that contains a string column and associate the parser plugin with a `FULLTEXT` index on the column:

```
mysql> CREATE TABLE t (c VARCHAR(255),
    ->   FULLTEXT (c) WITH PARSER simple_parser
    -> ) ENGINE=MyISAM;
Query OK, 0 rows affected (0.01 sec)
```

Insert some text into the table and try some searches. These should verify that the parser plugin treats all nonwhitespace characters as word characters:

```
mysql> INSERT INTO t VALUES
    ->   ('utf8mb4_0900_as_cs is a case-sensitive collation'),
    ->   ('I\'d like a case of oranges'),
    ->   ('this is sensitive information'),
    ->   ('another row'),
    ->   ('yet another row');
Query OK, 5 rows affected (0.02 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT c FROM t;
+-------------------------------------------------+
| c                                               |
+-------------------------------------------------+
| utf8mb4_0900_as_cs is a case-sensitive collation |
| I'd like a case of oranges                      |
| this is sensitive information                   |
| another row                                     |
| yet another row                                 |
+-------------------------------------------------+
5 rows in set (0.00 sec)

mysql> SELECT MATCH(c) AGAINST('case') FROM t;
+--------------------------+
| MATCH(c) AGAINST('case') |
+--------------------------+
|                        0 |
|          1.2968142032623 |
|                        0 |
|                        0 |
|                        0 |
+--------------------------+
5 rows in set (0.00 sec)

mysql> SELECT MATCH(c) AGAINST('sensitive') FROM t;
+-------------------------------+
| MATCH(c) AGAINST('sensitive') |
+-------------------------------+
|                             0 |
|                             0 |
|               1.3253291845322 |
|                             0 |
|                             0 |
+-------------------------------+
5 rows in set (0.01 sec)

mysql> SELECT MATCH(c) AGAINST('case-sensitive') FROM t;
+-----------------------------------+
| MATCH(c) AGAINST('case-sensitive') |
+-----------------------------------+
|                   1.3109166622162 |
|                                 0 |
|                                 0 |
|                                 0 |
|                                 0 |
+-----------------------------------+
5 rows in set (0.01 sec)

mysql> SELECT MATCH(c) AGAINST('I\'d') FROM t;
+--------------------------+
| MATCH(c) AGAINST('I\'d') |
+--------------------------+
|                        0 |
```

```
|          1.2968142032623 |
|                         0 |
|                         0 |
|                         0 |
+--------------------------+
5 rows in set (0.01 sec)
```

Neither “case” nor “insensitive” match “case-insensitive” the way that they would for the built-in parser.

29.2.4.5 Writing Daemon Plugins

A daemon plugin is a simple type of plugin used for code that should be run by the server but that does not communicate with it. This section describes how to write a daemon server plugin, using the example plugin found in the `plugin/daemon_example` directory of MySQL source distributions. That directory contains the `daemon_example.cc` source file for a daemon plugin named `daemon_example` that writes a heartbeat string at regular intervals to a file named `mysql-heartbeat.log` in the data directory.

To write a daemon plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin.h>
```

`plugin.h` defines the `MYSQL_DAEMON_PLUGIN` server plugin type and the data structures needed to declare the plugin.

The `daemon_example.cc` file sets up the library descriptor as follows. The library descriptor includes a single general server plugin descriptor.

```
mysql_declare_plugin(daemon_example)
{
  MYSQL_DAEMON_PLUGIN,
  &daemon_example_plugin,
  "daemon_example",
  "Brian Aker",
  "Daemon example, creates a heartbeat beat file in mysql-heartbeat.log",
  PLUGIN_LICENSE_GPL,
  daemon_example_plugin_init, /* Plugin Init */
  daemon_example_plugin_deinit, /* Plugin Deinit */
  0x0100 /* 1.0 */,
  NULL,                       /* status variables                */
  NULL,                       /* system variables                */
  NULL,                       /* config options                  */
  0,                          /* flags                           */
}
mysql_declare_plugin_end;
```

The `name` member (`daemon_example`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `SHOW PLUGINS` or `INFORMATION_SCHEMA.PLUGINS`.

The second member of the plugin descriptor, `daemon_example_plugin`, points to the type-specific daemon plugin descriptor. This structure consists only of the type-specific API version number:

```
struct st_mysql_daemon daemon_example_plugin=
{ MYSQL_DAEMON_INTERFACE_VERSION  };
```

The type-specific structure has no interface functions. There is no communication between the server and the plugin, except that the server calls the initialization and deinitialization functions from the general plugin descriptor to start and stop the plugin:

- `daemon_example_plugin_init()` opens the heartbeat file and spawns a thread that wakes up periodically and writes the next message to the file.
- `daemon_example_plugin_deinit()` closes the file and performs other cleanup.

To compile and install a plugin library file, use the instructions in Section 29.2.4.3, “Compiling and Installing Plugin Libraries”. To make the library file available for use, install it in the plugin directory (the directory named by the `plugin_dir` system variable). For the `daemon_example` plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of `libdaemon_example.so` (the `.so` suffix might differ depending on your platform).

To use the plugin, register it with the server. For example, to register the plugin at runtime, use this statement (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN daemon_example SONAME 'libdaemon_example.so';
```

For additional information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement. See Section 5.6.2, “Obtaining Server Plugin Information”.

While the plugin is loaded, it writes a heartbeat string at regular intervals to a file named `mysql-heartbeat.log` in the data directory. This file grows without limit, so after you have satistifed yourself that the plugin operates correctly, unload it:

```
UNINSTALL PLUGIN daemon_example;
```

29.2.4.6 Writing INFORMATION_SCHEMA Plugins

This section describes how to write a server-side `INFORMATION_SCHEMA` table plugin. For example code that implements such plugins, see the `sql/sql_show.cc` file of a MySQL source distribution. You can also look at the example plugins found in the `InnoDB` source. See the `handler/i_s.cc` and `handler/ha_innodb.cc` files within the `InnoDB` source tree (in the `storage/innobase` directory).

To write an `INFORMATION_SCHEMA` table plugin, include the following header files in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <sql_class.h>
#include <table.h>
```

These header files are located in the `sql` directory of MySQL source distributions. They contain C++ structures, so the source file for an `INFORMATION_SCHEMA` plugin must be compiled as C++ code.

The source file for the example plugin developed here is named `simple_i_s_table.cc`. It creates a simple `INFORMATION_SCHEMA` table named `SIMPLE_I_S_TABLE` that has two columns named `NAME` and `VALUE`. The general descriptor for a plugin library that implements the table looks like this:

```
mysql_declare_plugin(simple_i_s_library)
{
  MYSQL_INFORMATION_SCHEMA_PLUGIN,
  &simple_table_info,               /* type-specific descriptor */
  "SIMPLE_I_S_TABLE",               /* table name */
  "Author Name",                    /* author */
```

```
  "Simple INFORMATION_SCHEMA table", /* description */
  PLUGIN_LICENSE_GPL,                /* license type */
  simple_table_init,                 /* init function */
  NULL,
  0x0100,                            /* version = 1.0 */
  NULL,                              /* no status variables */
  NULL,                              /* no system variables */
  NULL,                              /* no reserved information */
  0                                  /* no flags */
}
mysql_declare_plugin_end;
```

The `name` member (`SIMPLE_I_S_TABLE`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `SHOW PLUGINS` or `INFORMATION_SCHEMA.PLUGINS`.

The `simple_table_info` member of the general descriptor points to the type-specific descriptor, which consists only of the type-specific API version number:

```
static struct st_mysql_information_schema simple_table_info =
{ MYSQL_INFORMATION_SCHEMA_INTERFACE_VERSION };
```

The general descriptor points to the initialization and deinitialization functions:

- The initialization function provides information about the table structure and a function that populates the table.
- The deinitialization function performs any required cleanup. If no cleanup is needed, this descriptor member can be `NULL` (as in the example shown).

The initialization function should return 0 for success, 1 if an error occurs. The function receives a generic pointer, which it should interpret as a pointer to the table structure:

```
static int table_init(void *ptr)
{
  ST_SCHEMA_TABLE *schema_table= (ST_SCHEMA_TABLE*)ptr;

  schema_table->fields_info= simple_table_fields;
  schema_table->fill_table= simple_fill_table;
  return 0;
}
```

The function should set these two members of the table structure:

- `fields_info`: An array of `ST_FIELD_INFO` structures that contain information about each column.
- `fill_table`: A function that populates the table.

The array pointed to by `fields_info` should contain one element per column of the `INFORMATION_SCHEMA` plus a terminating element. The following `simple_table_fields` array for the example plugin indicates that `SIMPLE_I_S_TABLE` has two columns. `NAME` is string-valued with a length of 10 and `VALUE` is integer-valued with a display width of 20. The last structure marks the end of the array.

```
static ST_FIELD_INFO simple_table_fields[]=
{
  {"NAME", 10, MYSQL_TYPE_STRING, 0, 0 0, 0},
  {"VALUE", 6, MYSQL_TYPE_LONG, 0, MY_I_S_UNSIGNED, 0, 0},
  {0, 0, MYSQL_TYPE_NULL, 0, 0, 0, 0}
};
```

For more information about the column information structure, see the definition of `ST_FIELD_INFO` in the `table.h` header file. The permissible `MYSQL_TYPE_xxx` type values are those used in the C API; see Section 28.7.5, “C API Data Structures”.

The `fill_table` member should be set to a function that populates the table and returns 0 for success, 1 if an error occurs. For the example plugin, the `simple_fill_table()` function looks like this:

```
static int simple_fill_table(THD *thd, TABLE_LIST *tables, Item *cond)
{
  TABLE *table= tables->table;

  table->field[0]->store("Name 1", 6, system_charset_info);
  table->field[1]->store(1);
  if (schema_table_store_record(thd, table))
    return 1;
  table->field[0]->store("Name 2", 6, system_charset_info);
  table->field[1]->store(2);
  if (schema_table_store_record(thd, table))
    return 1;
  return 0;
}
```

For each row of the `INFORMATION_SCHEMA` table, this function initializes each column, then calls `schema_table_store_record()` to install the row. The `store()` method arguments depend on the type of value to be stored. For column 0 (`NAME`, a string), `store()` takes a pointer to a string, its length, and information about the character set of the string:

```
store(const char *to, uint length, CHARSET_INFO *cs);
```

For column 1 (`VALUE`, an integer), `store()` takes the value and a flag indicating whether it is unsigned:

```
store(longlong nr, bool unsigned_value);
```

For other examples of how to populate `INFORMATION_SCHEMA` tables, search for instances of `schema_table_store_record()` in `sql_show.cc`.

To compile and install a plugin library file, use the instructions in Section 29.2.4.3, “Compiling and Installing Plugin Libraries”. To make the library file available for use, install it in the plugin directory (the directory named by the `plugin_dir` system variable).

To test the plugin, install it:

```
mysql> INSTALL PLUGIN SIMPLE_I_S_TABLE SONAME 'simple_i_s_table.so';
```

Verify that the table is present:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
    -> WHERE TABLE_NAME = 'SIMPLE_I_S_TABLE';
+------------------+
| TABLE_NAME       |
+------------------+
| SIMPLE_I_S_TABLE |
+------------------+
```

Try to select from it:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SIMPLE_I_S_TABLE;
```

```
+--------+-------+
| NAME   | VALUE |
+--------+-------+
| Name 1 |     1 |
| Name 2 |     2 |
+--------+-------+
```

Uninstall it:

```
mysql> UNINSTALL PLUGIN SIMPLE_I_S_TABLE;
```

29.2.4.7 Writing Semisynchronous Replication Plugins

This section describes how to write server-side semisynchronous replication plugins, using the example plugins found in the `plugin/semisync` directory of MySQL source distributions. That directory contains the source files for master and slave plugins named `rpl_semi_sync_master` and `rpl_semi_sync_slave`. The information here covers only how to set up the plugin framework. For details about how the plugins implement replication functions, see the source.

To write a semisynchronous replication plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin.h>
```

`plugin.h` defines the `MYSQL_REPLICATION_PLUGIN` server plugin type and the data structures needed to declare the plugin.

For the master side, `semisync_master_plugin.cc` contains this general descriptor for a plugin named `rpl_semi_sync_master`:

```
mysql_declare_plugin(semi_sync_master)
{
  MYSQL_REPLICATION_PLUGIN,
  &semi_sync_master_plugin,
  "rpl_semi_sync_master",
  "He Zhenxing",
  "Semi-synchronous replication master",
  PLUGIN_LICENSE_GPL,
  semi_sync_master_plugin_init, /* Plugin Init */
  semi_sync_master_plugin_deinit, /* Plugin Deinit */
  0x0100 /* 1.0 */,
  semi_sync_master_status_vars, /* status variables */
  semi_sync_master_system_vars, /* system variables */
  NULL,                         /* config options */
  0,                            /* flags */
}
mysql_declare_plugin_end;
```

For the slave side, `semisync_slave_plugin.cc` contains this general descriptor for a plugin named `rpl_semi_sync_slave`:

```
mysql_declare_plugin(semi_sync_slave)
{
  MYSQL_REPLICATION_PLUGIN,
  &semi_sync_slave_plugin,
  "rpl_semi_sync_slave",
  "He Zhenxing",
  "Semi-synchronous replication slave",
```

```
  PLUGIN_LICENSE_GPL,
  semi_sync_slave_plugin_init, /* Plugin Init */
  semi_sync_slave_plugin_deinit, /* Plugin Deinit */
  0x0100 /* 1.0 */,
  semi_sync_slave_status_vars,  /* status variables */
  semi_sync_slave_system_vars,  /* system variables */
  NULL,                         /* config options */
  0,                            /* flags */
}
mysql_declare_plugin_end;
```

For both the master and slave plugins, the general descriptor has pointers to the type-specific descriptor, the initialization and deinitialization functions, and to the status and system variables implemented by the plugin. For information about variable setup, see Server Plugin Status and System Variables. The following remarks discuss the type-specific descriptor and the initialization and deinitialization functions for the master plugin but apply similarly to the slave plugin.

The `semi_sync_master_plugin` member of the master general descriptor points to the type-specific descriptor, which consists only of the type-specific API version number:

```
struct Mysql_replication semi_sync_master_plugin= {
  MYSQL_REPLICATION_INTERFACE_VERSION
};
```

The initialization and deinitialization function declarations look like this:

```
static int semi_sync_master_plugin_init(void *p);
static int semi_sync_master_plugin_deinit(void *p);
```

The initialization function uses the pointer to register transaction and binary logging “observers” with the server. After successful initialization, the server takes care of invoking the observers at the appropriate times. (For details on the observers, see the source files.) The deinitialization function cleans up by deregistering the observers. Each function returns 0 for success or 1 if an error occurs.

To compile and install a plugin library file, use the instructions in Section 29.2.4.3, “Compiling and Installing Plugin Libraries”. To make the library file available for use, install it in the plugin directory (the directory named by the `plugin_dir` system variable). For the `rpl_semi_sync_master` and `rpl_semi_sync_slave` plugins, they are compiled and installed when you build MySQL from source. They are also included in binary distributions. The build process produces shared object libraries with names of `semisync_master.so` and `semisync_slave.so` (the `.so` suffix might differ depending on your platform).

29.2.4.8 Writing Audit Plugins

This section describes how to write a server-side audit plugin, using the example plugin found in the `plugin/audit_null` directory of MySQL source distributions. The `audit_null.c` and `audit_null_variables.h` source files in that directory implement an audit plugin named `NULL_AUDIT`.

Note

Other examples of plugins that use the audit plugin API are the query rewrite plugin (see Section 5.6.4, “The Rewriter Query Rewrite Plugin”) and the Version Tokens plugin (see Section 5.6.6, “Version Tokens”).

Within the server, the pluggable audit interface is implemented in the `sql_audit.h` and `sql_audit.cc` files in the `sql` directory of MySQL source distributions. Additionally, several places in the server call the audit interface when an auditable event occurs, so that registered audit plugins can be notified about

the event if necessary. To see where such calls occur, search the server source files for invocations of functions with names of the form `mysql_audit_xxx()`. Audit notification occurs for server operations such as these:

- Client connect and disconnect events
- Writing a message to the general query log (if the log is enabled)
- Writing a message to the error log
- Sending a query result to a client

To write an audit plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin_audit.h>
```

`plugin_audit.h` includes `plugin.h`, so you need not include the latter file explicitly. `plugin.h` defines the `MYSQL_AUDIT_PLUGIN` server plugin type and the data structures needed to declare the plugin. `plugin_audit.h` defines data structures specific to audit plugins.

Audit Plugin General Descriptor

An audit plugin, like any MySQL server plugin, has a general plugin descriptor (see Server Plugin Library and Plugin Descriptors) and a type-specific plugin descriptor. In `audit_null.c`, the general descriptor for `audit_null` looks like this:

```
mysql_declare_plugin(audit_null)
{
  MYSQL_AUDIT_PLUGIN,         /* type                            */
  &audit_null_descriptor,     /* descriptor                      */
  "NULL_AUDIT",               /* name                            */
  "Oracle Corp",              /* author                          */
  "Simple NULL Audit",        /* description                     */
  PLUGIN_LICENSE_GPL,
  audit_null_plugin_init,     /* init function (when loaded)     */
  audit_null_plugin_deinit,   /* deinit function (when unloaded) */
  0x0003,                     /* version                         */
  simple_status,              /* status variables                */
  system_variables,           /* system variables                */
  NULL,
  0,
}
mysql_declare_plugin_end;
```

The first member, `MYSQL_AUDIT_PLUGIN`, identifies this plugin as an audit plugin.

`audit_null_descriptor` points to the type-specific plugin descriptor, described later.

The `name` member (`NULL_AUDIT`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `INFORMATION_SCHEMA.PLUGINS` or `SHOW PLUGINS`.

The `audit_null_plugin_init` initialization function performs plugin initialization when the plugin is loaded. The `audit_null_plugin_deinit` function performs cleanup when the plugin is unloaded.

The general plugin descriptor also refers to `simple_status` and `system_variables`, structures that expose several status and system variables. When the plugin is enabled, these variables can be inspected

using SHOW statements (SHOW STATUS, SHOW VARIABLES) or the appropriate Performance Schema tables.

The simple_status structure declares several status variables with names of the form Audit_null_*xxx*. NULL_AUDIT increments the Audit_null_called status variable for every notification that it receives. The other status variables are more specific and NULL_AUDIT increments them only for notifications of specific events.

system_variables is an array of system variable elements, each of which is defined using a MYSQL_THDVAR_*xxx* macro. These system variables have names of the form null_audit_*xxx*. These variables can be used to communicate with the plugin at runtime.

Audit Plugin Type-Specific Descriptor

The audit_null_descriptor value in the general plugin descriptor points to the type-specific plugin descriptor. For audit plugins, this descriptor has the following structure (defined in plugin_audit.h):

```
struct st_mysql_audit
{
  int interface_version;
  void (*release_thd)(MYSQL_THD);
  int (*event_notify)(MYSQL_THD, mysql_event_class_t, const void *);
  unsigned long class_mask[MYSQL_AUDIT_CLASS_MASK_SIZE];
};
```

The type-specific descriptor for audit plugins has these members:

- interface_version: By convention, type-specific plugin descriptors begin with the interface version for the given plugin type. The server checks interface_version when it loads the plugin to see whether the plugin is compatible with it. For audit plugins, the value of the interface_version member is MYSQL_AUDIT_INTERFACE_VERSION (defined in plugin_audit.h).
- release_thd: A function that the server calls to inform the plugin that it is being dissociated from its thread context. This should be NULL if there is no such function.
- event_notify: A function that the server calls to notify the plugin that an auditable event has occurred. This function should not be NULL; that would not make sense because no auditing would occur.
- class_mask: An array of MYSQL_AUDIT_CLASS_MASK_SIZE elements. Each element specifies a bitmask for a given event class to indicate the subclasses for which the plugin wants notification. (This is how the plugin “subscribes” to events of interest.) An element should be 0 to ignore events for the corresponding event class.

The server uses the event_notify and release_thd functions together. They are called within the context of a specific thread, and a thread might perform an activity that produces several event notifications. The first time the server calls event_notify for a thread, it creates a binding of the plugin to the thread. The plugin cannot be uninstalled while this binding exists. When no more events for the thread will occur, the server informs the plugin of this by calling the release_thd function, and then destroys the binding. For example, when a client issues a statement, the thread processing the statement might notify audit plugins about the result set produced by the statement and about the statement being logged. After these notifications occur, the server releases the plugin before putting the thread to sleep until the client issues another statement.

This design enables the plugin to allocate resources needed for a given thread in the first call to the event_notify function and release them in the release_thd function:

```
event_notify function:
```

```
  if memory is needed to service the thread
    allocate memory
  ... rest of notification processing ...

release_thd function:
  if memory was allocated
    release memory
  ... rest of release processing ...
```

That is more efficient than allocating and releasing memory repeatedly in the notification function.

For the `NULL_AUDIT` audit plugin, the type-specific plugin descriptor looks like this:

```
static struct st_mysql_audit audit_null_descriptor=
{
  MYSQL_AUDIT_INTERFACE_VERSION,                  /* interface version    */
  NULL,                                           /* release_thd function */
  audit_null_notify,                              /* notify function      */
  { (unsigned long) MYSQL_AUDIT_GENERAL_ALL,
    (unsigned long) MYSQL_AUDIT_CONNECTION_ALL,
    (unsigned long) MYSQL_AUDIT_PARSE_ALL,
    (unsigned long) MYSQL_AUDIT_AUTHORIZATION_ALL,
    (unsigned long) MYSQL_AUDIT_TABLE_ACCESS_ALL,
    (unsigned long) MYSQL_AUDIT_GLOBAL_VARIABLE_ALL,
    (unsigned long) MYSQL_AUDIT_SERVER_STARTUP_ALL,
    (unsigned long) MYSQL_AUDIT_SERVER_SHUTDOWN_ALL,
    (unsigned long) MYSQL_AUDIT_COMMAND_ALL,
    (unsigned long) MYSQL_AUDIT_QUERY_ALL,
    (unsigned long) MYSQL_AUDIT_STORED_PROGRAM_ALL }
};
```

The server calls `audit_null_notify()` to pass audit event information to the plugin. There is no `release_thd` function.

The `class_mask` member is an array that indicates which event classes the plugin subscribes to. As shown, the array contents subscribe to all subclasses of all event classes that are available. To ignore all notifications for a given event class, specify the corresponding `class_mask` element as 0.

The number of `class_mask` elements corresponds to the number of event classes, each of which is listed in the `mysql_event_class_t` enumeration defined in `plugin_audit.h`:

```
typedef enum
{
  MYSQL_AUDIT_GENERAL_CLASS          = 0,
  MYSQL_AUDIT_CONNECTION_CLASS       = 1,
  MYSQL_AUDIT_PARSE_CLASS            = 2,
  MYSQL_AUDIT_AUTHORIZATION_CLASS    = 3,
  MYSQL_AUDIT_TABLE_ACCESS_CLASS     = 4,
  MYSQL_AUDIT_GLOBAL_VARIABLE_CLASS  = 5,
  MYSQL_AUDIT_SERVER_STARTUP_CLASS   = 6,
  MYSQL_AUDIT_SERVER_SHUTDOWN_CLASS  = 7,
  MYSQL_AUDIT_COMMAND_CLASS          = 8,
  MYSQL_AUDIT_QUERY_CLASS            = 9,
  MYSQL_AUDIT_STORED_PROGRAM_CLASS   = 10,
  /* This item must be last in the list. */
  MYSQL_AUDIT_CLASS_MASK_SIZE
} mysql_event_class_t;
```

For any given event class, `plugin_audit.h` defines bitmask symbols for individual event subclasses, as well as an *`xxx`*`_ALL` symbol that is the union of the all subclass bitmasks. For example, for `MYSQL_AUDIT_CONNECTION_CLASS` (the class that covers connect and disconnect events), `plugin_audit.h` defines these symbols:

```
typedef enum
{
  /** occurs after authentication phase is completed. */
  MYSQL_AUDIT_CONNECTION_CONNECT          = 1 << 0,
  /** occurs after connection is terminated. */
  MYSQL_AUDIT_CONNECTION_DISCONNECT       = 1 << 1,
  /** occurs after COM_CHANGE_USER RPC is completed. */
  MYSQL_AUDIT_CONNECTION_CHANGE_USER      = 1 << 2,
  /** occurs before authentication. */
  MYSQL_AUDIT_CONNECTION_PRE_AUTHENTICATE = 1 << 3
} mysql_event_connection_subclass_t;

#define MYSQL_AUDIT_CONNECTION_ALL (MYSQL_AUDIT_CONNECTION_CONNECT | \
                                    MYSQL_AUDIT_CONNECTION_DISCONNECT | \
                                    MYSQL_AUDIT_CONNECTION_CHANGE_USER | \
                                    MYSQL_AUDIT_CONNECTION_PRE_AUTHENTICATE)
```

To subscribe to all subclasses of the connection event class (as the `NULL_AUDIT` plugin does), a plugin specifies `MYSQL_AUDIT_CONNECTION_ALL` in the corresponding `class_mask` element (`class_mask[1]` in this case). To subscribe to only some subclasses, the plugin sets the `class_mask` element to the union of the subclasses of interest. For example, to subscribe only to the connect and change-user subclasses, the plugin sets `class_mask[1]` to this value:

```
MYSQL_AUDIT_CONNECTION_CONNECT | MYSQL_AUDIT_CONNECTION_CHANGE_USER
```

Audit Plugin Notification Function

Most of the work for an audit plugin occurs in the notification function (the `event_notify` member of the type-specific plugin descriptor). The server calls this function for each auditable event. Audit plugin notification functions have this prototype:

```
int (*event_notify)(MYSQL_THD, mysql_event_class_t, const void *);
```

The second and third parameters of the `event_notify` function prototype represent the event class and a generic pointer to an event structure. (Events in different classes have different structures. The notification function can use the event class value to determine which event structure applies.) The function processes the event and returns a status indicating whether the server should continue processing the event or terminate it.

For `NULL_AUDIT`, the notification function is `audit_null_notify()`. This function increments a global event counter (which the plugin exposes as the value of the `Audit_null_called` status value), and then examines the event class to determine how to process the event structure:

```
static int audit_null_notify(MYSQL_THD thd __attribute__((unused)),
                             mysql_event_class_t event_class,
                             const void *event)
{
  ...

  number_of_calls++;

  if (event_class == MYSQL_AUDIT_GENERAL_CLASS)
  {
    const struct mysql_event_general *event_general=
                                    (const struct mysql_event_general *)event;
    ...
  }
  else if (event_class == MYSQL_AUDIT_CONNECTION_CLASS)
  {
```

```
    const struct mysql_event_connection *event_connection=
                                (const struct mysql_event_connection *) event;
    ...

  }
  else if (event_class == MYSQL_AUDIT_PARSE_CLASS)
  {
    const struct mysql_event_parse *event_parse =
                                      (const struct mysql_event_parse *)event;
    ...
  }
  ...
}
```

The notification function interprets the `event` argument according to the value of `event_class`. The `event` argument is a generic pointer to the event record, the structure of which differs per event class. (The `plugin_audit.h` file contains the structures that define the contents of each event class.) For each class, `audit_null_notify()` casts the event to the appropriate class-specific structure and then checks its subclass to determine which subclass counter to increment. For example, the code to handle events in the connection-event class looks like this:

```
else if (event_class == MYSQL_AUDIT_CONNECTION_CLASS)
{
  const struct mysql_event_connection *event_connection=
                              (const struct mysql_event_connection *) event;

  switch (event_connection->event_subclass)
  {
  case MYSQL_AUDIT_CONNECTION_CONNECT:
    number_of_calls_connection_connect++;
    break;
  case MYSQL_AUDIT_CONNECTION_DISCONNECT:
    number_of_calls_connection_disconnect++;
    break;
  case MYSQL_AUDIT_CONNECTION_CHANGE_USER:
    number_of_calls_connection_change_user++;
    break;
  case MYSQL_AUDIT_CONNECTION_PRE_AUTHENTICATE:
    number_of_calls_connection_pre_authenticate++;
      break;
  default:
    break;
  }
}
```

Note

The general event class (`MYSQL_AUDIT_GENERAL_CLASS`) is deprecated as of MySQL 5.7.9 and will be removed in a future MySQL release. To reduce plugin overhead, it is preferable to subscribe only to the more specific event classes of interest.

For some event classes, the `NULL_AUDIT` plugin performs other processing in addition to incrementing a counter. In any case, when the notification function finishes processing the event, it should return a status indicating whether the server should continue processing the event or terminate it.

Audit Plugin Error Handling

Audit plugin notification functions can report a status value for the current event two ways:

- Use the notification function return value. In this case, the function returns zero if the server should continue processing the event, or nonzero if the server should terminate the event.

- Call the `my_message()` function to set the error state before returning from the notification function. In this case, the notification function return value is ignored and the server terminates event processing with an error. The `my_message()` arguments indicate which error to report, and its message. For example:

```
my_message(ER_AUDIT_API_ABORT, "This is my error message.", MYF(0));
```

  Some events cannot be aborted. A nonzero return value is not taken into consideration and the `my_message()` error call must follow an `is_error()` check. For example:

```
if (!thd->get_stmt_da()->is_error())
{
  my_message(ER_AUDIT_API_ABORT, "This is my error message.", MYF(0));
}
```

Some events cannot be terminated:

- `MYSQL_AUDIT_CONNECTION_DISCONNECT`: The server cannot prevent a client from disconnecting.
- `MYSQL_AUDIT_COMMAND_END`: This event provides the status of a command that has finished executing, so there is no purpose to terminating it.

If an audit plugin returns nonzero status for a nonterminable event, the server ignores the status and continues processing the event. This is also true if an audit plugin uses the `my_message()` function to terminate a nonterminable event.

Audit Plugin Usage

To compile and install a plugin library file, use the instructions in Section 29.2.4.3, “Compiling and Installing Plugin Libraries”. To make the library file available for use, install it in the plugin directory (the directory named by the `plugin_dir` system variable). For the `NULL_AUDIT` plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of `adt_null.so` (the `.so` suffix might differ depending on your platform).

To register the plugin at runtime, use this statement (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN NULL_AUDIT SONAME 'adt_null.so';
```

For additional information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement. See Section 5.6.2, “Obtaining Server Plugin Information”.

While the audit plugin is installed, it exposes status variables that indicate the events for which the plugin has been called:

```
mysql> SHOW STATUS LIKE 'Audit_null%';
+----------------------------------------+--------+
| Variable_name                          | Value  |
+----------------------------------------+--------+
| Audit_null_authorization_column        | 0      |
| Audit_null_authorization_db            | 0      |
| Audit_null_authorization_procedure     | 0      |
| Audit_null_authorization_proxy         | 0      |
| Audit_null_authorization_table         | 0      |
```

```
| Audit_null_authorization_user          | 0      |
| Audit_null_called                      | 185547 |
| Audit_null_command_end                 | 20999  |
| Audit_null_command_start               | 21001  |
| Audit_null_connection_change_user      | 0      |
| Audit_null_connection_connect          | 5823   |
| Audit_null_connection_disconnect       | 5818   |
| Audit_null_connection_pre_authenticate | 5823   |
| Audit_null_general_error               | 1      |
| Audit_null_general_log                 | 26559  |
| Audit_null_general_result              | 19922  |
| Audit_null_general_status              | 21000  |
| Audit_null_global_variable_get         | 0      |
| Audit_null_global_variable_set         | 0      |
| Audit_null_parse_postparse             | 14648  |
| Audit_null_parse_preparse              | 14648  |
| Audit_null_query_nested_start          | 6      |
| Audit_null_query_nested_status_end     | 6      |
| Audit_null_query_start                 | 14648  |
| Audit_null_query_status_end            | 14647  |
| Audit_null_server_shutdown             | 0      |
| Audit_null_server_startup              | 1      |
| Audit_null_table_access_delete         | 104    |
| Audit_null_table_access_insert         | 2839   |
| Audit_null_table_access_read           | 97842  |
| Audit_null_table_access_update         | 278    |
+----------------------------------------+--------+
```

`Audit_null_called` counts all events, and the other variables count instances of specific event subclasses. For example, the preceding `SHOW STATUS` statement causes the server to send a result to the client and to write a message to the general query log if that log is enabled. Thus, a client that issues the statement repeatedly causes `Audit_null_called`, `Audit_null_general_result`, and `Audit_null_general_log` to be incremented each time. Notifications occur whether or not that log is enabled.

The status variables values are aggregated across all sessions. There are no counters for individual sessions.

`NULL_AUDIT` exposes several system variables that enable communication with the plugin at runtime:

```
mysql> SHOW VARIABLES LIKE 'null_audit%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| null_audit_abort_message          |       |
| null_audit_abort_value            | 1     |
| null_audit_event_order_check      |       |
| null_audit_event_order_check_exact | 1     |
| null_audit_event_order_started    | 0     |
| null_audit_event_record           |       |
| null_audit_event_record_def       |       |
+-----------------------------------+-------+
```

To check the order of audit API calls, set the `null_audit_event_order_check` variable to the expected event order. For example:

```
SET null_audit_event_order_check =
      'MYSQL_AUDIT_CONNECTION_PRE_AUTHENTICATE;;;'
      'MYSQL_AUDIT_GENERAL_LOG;;;'
      'MYSQL_AUDIT_CONNECTION_CONNECT;;';
```

The statement takes advantage of the SQL syntax that concatenates adjacent strings into a single string.

The format of the value is:

```
'event_name;event_data;command' [';event_name;event_data;command'] ...
```

After the event order is matched, the `null_audit_event_order_check` value is replaced with a value of `EVENT-ORDER-OK`.

Specifying a command value of `ABORT_RET` makes it possible to abort the audit API call on the specified event. The following example aborts `INSERT` statement execution when its `MYSQL_AUDIT_QUERY_STATUS_END` event occurs:

```
SET null_audit_event_order_check =
   'MYSQL_AUDIT_COMMAND_START;command_id="3";;'
   'MYSQL_AUDIT_GENERAL_LOG;;;'
   'MYSQL_AUDIT_QUERY_START;;;'
   'MYSQL_AUDIT_QUERY_STATUS_END;;ABORT_RET';
```

After the audit plugin matches the preceding sequence, it aborts event processing and sends an error message to the client:

```
ERROR 3164 (HY000): Aborted by Audit API ('MYSQL_AUDIT_QUERY_STATUS_END';1).
```

Returning a nonzero value from the audit API notification routine is the standard way to abort event execution. It is also possible to specify a custom error code by setting the `null_audit_abort_value` variable to the value that the notification routine should return:

```
SET null_audit_abort_value = 123;
```

Aborting a sequence results in a standard message with the custom error code. Suppose that you set audit log system variables like this:

```
SET null_audit_abort_value = 123;
SET null_audit_event_order_check =
    'MYSQL_AUDIT_COMMAND_START;command_id="3";;'
    'MYSQL_AUDIT_GENERAL_LOG;;;'
    'MYSQL_AUDIT_QUERY_START;;ABORT_RET';
```

Then execution of `SELECT 1` results in this error:

```
ERROR 3164 (HY000): Aborted by Audit API ('MYSQL_AUDIT_QUERY_START';123).
```

An event can be also aborted with a custom message, specified by setting the `null_audit_abort_message` variable: Suppose that you set audit log system variables like this:

```
SET null_audit_abort_message = 'Custom error text.';
SET null_audit_event_order_check =
    'MYSQL_AUDIT_COMMAND_START;command_id="3";;'
    'MYSQL_AUDIT_GENERAL_LOG;;;'
    'MYSQL_AUDIT_QUERY_START;;ABORT_RET';
```

Then aborting a sequence results in the following error:

```
ERROR 3164 (HY000): Custom error text.
```

For test-creation purposes, it is possible to record events that pass through the plugin. Recording starts by specifying start and end events in the `null_audit_event_record_def` variable:

```
SET null_audit_event_record_def =
    'MYSQL_AUDIT_COMMAND_START;MYSQL_AUDIT_COMMAND_END';
```

Statement execution results in storing the events that occur in the `null_audit_event_record` variable.

To disable the plugin after testing it, use this statement to unload it:

```
UNINSTALL PLUGIN NULL_AUDIT;
```

29.2.4.9 Writing Authentication Plugins

MySQL supports pluggable authentication, in which plugins are invoked to authenticate client connections. Authentication plugins enable the use of authentication methods other than the built-in method of passwords stored in the `mysql.user` system table. For example, plugins can be written to access external authentication methods. Also, authentication plugins can support the proxy user capability, such that the connecting user is a proxy for another user and is treated, for purposes of access control, as having the privileges of a different user. For more information, see Section 6.2.17, “Pluggable Authentication”, and Section 6.2.18, “Proxy Users”.

An authentication plugin can be written for the server side or the client side. Server-side plugins use the same plugin API that is used for the other server plugin types such as full-text parser or audit plugins (although with a different type-specific descriptor). Client-side plugins use the client plugin API.

Several header files contain information relevant to authentication plugins:

- `plugin.h`: Defines the `MYSQL_AUTHENTICATION_PLUGIN` server plugin type.
- `client_plugin.h`: Defines the API for client plugins. This includes the client plugin descriptor and function prototypes for client plugin C API calls (see Section 28.7.17, “C API Client Plugin Functions”).
- `plugin_auth.h`: Defines the part of the server plugin API specific to authentication plugins. This includes the type-specific descriptor for server-side authentication plugins and the `MYSQL_SERVER_AUTH_INFO` structure.
- `plugin_auth_common.h`: Contains common elements of client and server authentication plugins. This includes return value definitions and the `MYSQL_PLUGIN_VIO` structure.

To write an authentication plugin, include the following header files in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

- For a source file that implements a server authentication plugin, include this file:

  ```
  #include <mysql/plugin_auth.h>
  ```

- For a source file that implements a client authentication plugin, or both client and server plugins, include these files:

  ```
  #include <mysql/plugin_auth.h>
  #include <mysql/client_plugin.h>
  #include <mysql.h>
  ```

`plugin_auth.h` includes `plugin.h` and `plugin_auth_common.h`, so you need not include the latter files explicitly.

This section describes how to write a pair of simple server and client authentication plugins that work together.

Warning

These plugins accept any non-empty password and the password is sent as cleartext. This is insecure, so the plugins *should not be used in production environments.*

The server-side and client-side plugins developed here both are named `auth_simple`. As described in Section 29.2.4.2, “Plugin Data Structures”, the plugin library file must have the same base name as the client plugin, so the source file name is `auth_simple.c` and produces a library named `auth_simple.so` (assuming that your system uses `.so` as the suffix for library files).

In MySQL source distributions, authentication plugin source is located in the `plugin/auth` directory and can be examined as a guide to writing other authentication plugins. Also, to see how the built-in authentication plugins are implemented, see `sql/sql_acl.cc` for plugins that are built in to the MySQL server and `sql-common/client.c` for plugins that are built in to the `libmysqlclient` client library. (For the built-in client plugins, note that the `auth_plugin_t` structures used there differ from the structures used with the usual client plugin declaration macros. In particular, the first two members are provided explicitly, not by declaration macros.)

Writing the Server-Side Authentication Plugin

Declare the server-side plugin with the usual general descriptor format that is used for all server plugin types (see Server Plugin Library and Plugin Descriptors). For the `auth_simple` plugin, the descriptor looks like this:

```
mysql_declare_plugin(auth_simple)
{
  MYSQL_AUTHENTICATION_PLUGIN,
  &auth_simple_handler,                /* type-specific descriptor */
  "auth_simple",                       /* plugin name */
  "Author Name",                       /* author */
  "Any-password authentication plugin", /* description */
  PLUGIN_LICENSE_GPL,                  /* license type */
  NULL,                                /* no init function */
  NULL,                                /* no deinit function */
  0x0100,                              /* version = 1.0 */
  NULL,                                /* no status variables */
  NULL,                                /* no system variables */
  NULL,                                /* no reserved information */
  0                                    /* no flags */
}
mysql_declare_plugin_end;
```

The `name` member (`auth_simple`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `SHOW PLUGINS` or `INFORMATION_SCHEMA.PLUGINS`.

The `auth_simple_handler` member of the general descriptor points to the type-specific descriptor. For an authentication plugin, the type-specific descriptor is an instance of the `st_mysql_auth` structure (defined in `plugin_auth.h`):

```
struct st_mysql_auth
{
  int interface_version;
  const char *client_auth_plugin;
  int (*authenticate_user)(MYSQL_PLUGIN_VIO *vio, MYSQL_SERVER_AUTH_INFO *info);
  int (*generate_authentication_string)(char *outbuf,
      unsigned int *outbuflen, const char *inbuf, unsigned int inbuflen);
  int (*validate_authentication_string)(char* const inbuf, unsigned int buflen);
```

```
  int (*set_salt)(const char *password, unsigned int password_len,
                  unsigned char* salt, unsigned char *salt_len);
  const unsigned long authentication_flags;
};
```

The `st_mysql_auth` structure has these members:

- `interface_version`: The type-specific API version number, always `MYSQL_AUTHENTICATION_INTERFACE_VERSION`
- `client_auth_plugin`: The client plugin name
- `authenticate_user`: A pointer to the main plugin function that communicates with the client
- `generate_authentication_string`: A pointer to a plugin function that generates a password digest from an authentication string
- `validate_authentication_string`: A pointer to a plugin function that validates a password digest
- `set_salt`: A pointer to a plugin function that converts a scrambled password to binary form
- `authentication_flags`: A flags word

The `client_auth_plugin` member should indicate the name of the client plugin if a specific plugin is required. A value of `NULL` means “any plugin.” In the latter case, whatever plugin the client uses will do. This is useful if the server plugin does not care about the client plugin or what user name or password it sends. For example, this might be true if the server plugin authenticates only local clients and uses some property of the operating system rather than the information sent by the client plugin.

For `auth_simple`, the type-specific descriptor looks like this:

```
static struct st_mysql_auth auth_simple_handler =
{
  MYSQL_AUTHENTICATION_INTERFACE_VERSION,
  "auth_simple",              /* required client-side plugin name */
  auth_simple_server          /* server-side plugin main function */
  generate_auth_string_hash, /* generate digest from password string */
  validate_auth_string_hash, /* validate password digest */
  set_salt,                   /* generate password salt value */
  AUTH_FLAG_PRIVILEGED_USER_FOR_PASSWORD_CHANGE
};
```

The main function, `auth_simple_server()`, takes two arguments representing an I/O structure and a `MYSQL_SERVER_AUTH_INFO` structure. The structure definition, found in `plugin_auth.h`, looks like this:

```
typedef struct st_mysql_server_auth_info
{
  char *user_name;
  unsigned int user_name_length;
  const char *auth_string;
  unsigned long auth_string_length;
  char authenticated_as[MYSQL_USERNAME_LENGTH+1];
  char external_user[512];
  int  password_used;
  const char *host_or_ip;
  unsigned int host_or_ip_length;
} MYSQL_SERVER_AUTH_INFO;
```

The character set for string members is UTF-8. If there is a `_length` member associated with a string, it indicates the string length in bytes. Strings are also null-terminated.

When an authentication plugin is invoked by the server, it should interpret the `MYSQL_SERVER_AUTH_INFO` structure members as follows. Some of these are used to set the value of SQL functions or system variables within the client session, as indicated.

- `user_name`: The user name sent by the client. The value becomes the `USER()` function value.
- `user_name_length`: The length of `user_name` in bytes.
- `auth_string`: The value of the `authentication_string` column of the row in the `mysql.user` system table for the matching account name (that is, the row that matches the client user name and host name and that the server uses to determine how to authenticate the client).

  Suppose that you create an account using the following statement:

  ```
  CREATE USER 'my_user'@'localhost'
    IDENTIFIED WITH my_plugin AS 'my_auth_string';
  ```

  When `my_user` connects from the local host, the server invokes `my_plugin` and passes `'my_auth_string'` to it as the `auth_string` value.
- `auth_string_length`: The length of `auth_string` in bytes.
- `authenticated_as`: The server sets this to the user name (the value of `user_name`). The plugin can alter it to indicate that the client should have the privileges of a different user. For example, if the plugin supports proxy users, the initial value is the name of the connecting (proxy) user, and the plugin can change this member to the proxied user name. The server then treats the proxy user as having the privileges of the proxied user (assuming that the other conditions for proxy user support are satisfied; see Implementing Proxy User Support in Authentication Plugins). The value is represented as a string at most `MYSQL_USER_NAME_LENGTH` bytes long, plus a terminating null. The value becomes the `CURRENT_USER()` function value.
- `external_user`: The server sets this to the empty string (null terminated). Its value becomes the `external_user` system variable value. If the plugin wants that system variable to have a different value, it should set this member accordingly (for example, to the connecting user name). The value is represented as a string at most 511 bytes long, plus a terminating null.
- `password_used`: This member applies when authentication fails. The plugin can set it or ignore it. The value is used to construct the failure error message of `Authentication fails. Password used: %s`. The value of `password_used` determines how `%s` is handled, as shown in the following table.

`password_used`	`%s` Handling
0	NO
1	YES
2	There will be no `%s`

- `host_or_ip`: The name of the client host if it can be resolved, or the IP address otherwise.
- `host_or_ip_length`: The length of `host_or_ip` in bytes.

The `auth_simple` main function, `auth_simple_server()`, reads the password (a null-terminated string) from the client and succeeds if the password is nonempty (first byte not null):

```
static int auth_simple_server (MYSQL_PLUGIN_VIO *vio,
                               MYSQL_SERVER_AUTH_INFO *info)
{
```

```
  unsigned char *pkt;
  int pkt_len;

  /* read the password as null-terminated string, fail on error */
  if ((pkt_len= vio->read_packet(vio, &pkt)) < 0)
    return CR_ERROR;

  /* fail on empty password */
  if (!pkt_len || *pkt == '\0')
  {
    info->password_used= PASSWORD_USED_NO;
    return CR_ERROR;
  }

  /* accept any nonempty password */
  info->password_used= PASSWORD_USED_YES;

  return CR_OK;
}
```

The main function should return one of the error codes shown in the following table.

Error Code	Meaning
`CR_OK`	Success
`CR_OK_HANDSHAKE_COMPLETE`	Do not send a status packet back to client
`CR_ERROR`	Error
`CR_AUTH_USER_CREDENTIALS`	Authentication failure
`CR_AUTH_HANDSHAKE`	Authentication handshake failure
`CR_AUTH_PLUGIN_ERROR`	Internal plugin error

For an example of how the handshake works, see the `plugin/auth/dialog.c` source file.

The server counts plugin errors in the Performance Schema `host_cache` table.

`auth_simple_server()` is so basic that it does not use the authentication information structure except to set the member that indicates whether a password was received.

A plugin that supports proxy users must return to the server the name of the proxied user (the MySQL user whose privileges the client user should get). To do this, the plugin must set the `info->authenticated_as` member to the proxied user name. For information about proxying, see Section 6.2.18, “Proxy Users”, and Implementing Proxy User Support in Authentication Plugins.

The `generate_authentication_string` member of the plugin descriptor takes the password and generates a password hash (digest) from it:

- The first two arguments are pointers to the output buffer and its maximum length in bytes. The function should write the password hash to the output buffer and reset the length to the actual hash length.
- The second two arguments indicate the password input buffer and its length in bytes.
- The function returns 0 for success, 1 if an error occurred.

For the `auth_simple` plugin, the `generate_auth_string_hash()` function implements the `generate_authentication_string` member. It just makes a copy of the password, unless it is too long to fit in the output buffer.

```
int generate_auth_string_hash(char *outbuf, unsigned int *buflen,
```

```
                              const char *inbuf, unsigned int inbuflen)
{
  /*
    fail if buffer specified by server cannot be copied to output buffer
  */
  if (*buflen < inbuflen)
    return 1;   /* error */
  strncpy(outbuf, inbuf, inbuflen);
  *buflen= strlen(inbuf);
  return 0;     /* success */
}
```

The `validate_authentication_string` member of the plugin descriptor validates a password hash:

- The arguments are a pointer to the password hash and its length in bytes.
- The function returns 0 for success, 1 if the password hash cannot be validated.

For the `auth_simple` plugin, the `validate_auth_string_hash()` function implements the `validate_authentication_string` member. It returns success unconditionally:

```
int validate_auth_string_hash(char* const inbuf  __attribute__((unused)),
                              unsigned int buflen  __attribute__((unused)))
{
  return 0;     /* success */
}
```

The `set_salt` member of the plugin descriptor is used only by the `mysql_native_password` plugin (see Section 6.4.1.1, “Native Pluggable Authentication”). For other authentication plugins, you can use this trivial implementation:

```
int set_salt(const char* password __attribute__((unused)),
             unsigned int password_len __attribute__((unused)),
             unsigned char* salt __attribute__((unused)),
             unsigned char* salt_len)
{
  *salt_len= 0;
  return 0;     /* success */
}
```

The `authentication_flags` member of the plugin descriptor contains flags that affect plugin operation. The permitted flags are:

- `AUTH_FLAG_PRIVILEGED_USER_FOR_PASSWORD_CHANGE`: Credential changes are a privileged operation. If this flag is set, the server requires that the user has the global `CREATE USER` privilege or the `UPDATE` privilege for the `mysql` database.
- `AUTH_FLAG_USES_INTERNAL_STORAGE`: Whether the plugin uses internal storage (in the `authentication_string` column of `mysql.user` rows). If this flag is not set, attempts to set the password fail and the server produces a warning.

Writing the Client-Side Authentication Plugin

Declare the client-side plugin descriptor with the `mysql_declare_client_plugin()` and `mysql_end_client_plugin` macros (see Client Plugin Descriptors). For the `auth_simple` plugin, the descriptor looks like this:

```
mysql_declare_client_plugin(AUTHENTICATION)
  "auth_simple",                        /* plugin name */
```

```
  "Author Name",                        /* author */
  "Any-password authentication plugin", /* description */
  {1,0,0},                              /* version = 1.0.0 */
  "GPL",                                /* license type */
  NULL,                                 /* for internal use */
  NULL,                                 /* no init function */
  NULL,                                 /* no deinit function */
  NULL,                                 /* no option-handling function */
  auth_simple_client                    /* main function */
mysql_end_client_plugin;
```

The descriptor members from the plugin name through the option-handling function are common to all client plugin types. (For descriptions, see Client Plugin Descriptors.) Following the common members, the descriptor has an additional member specific to authentication plugins. This is the “main” function, which handles communication with the server. The function takes two arguments representing an I/O structure and a connection handler. For our simple any-password plugin, the main function does nothing but write to the server the password provided by the user:

```
static int auth_simple_client (MYSQL_PLUGIN_VIO *vio, MYSQL *mysql)
{
  int res;

  /* send password as null-terminated string as cleartext */
  res= vio->write_packet(vio, (const unsigned char *) mysql->passwd,
                         strlen(mysql->passwd) + 1);

  return res ? CR_ERROR : CR_OK;
}
```

The main function should return one of the error codes shown in the following table.

Error Code	Meaning
`CR_OK`	Success
`CR_OK_HANDSHAKE_COMPLETE`	Success, client done
`CR_ERROR`	Error

`CR_OK_HANDSHAKE_COMPLETE` indicates that the client has done its part successfully and has read the last packet. A client plugin may return `CR_OK_HANDSHAKE_COMPLETE` if the number of round trips in the authentication protocol is not known in advance and the plugin must read another packet to determine whether authentication is finished.

Using the Authentication Plugins

To compile and install a plugin library file, use the instructions in Section 29.2.4.3, “Compiling and Installing Plugin Libraries”. To make the library file available for use, install it in the plugin directory (the directory named by the `plugin_dir` system variable).

Register the server-side plugin with the server. For example, to load the plugin at server startup, use a `--plugin-load=auth_simple.so` option (adjust the `.so` suffix for your platform as necessary).

Create a user for whom the server will use the `auth_simple` plugin for authentication:

```
mysql> CREATE USER 'x'@'localhost'
    -> IDENTIFIED WITH auth_simple;
```

Use a client program to connect to the server as user `x`. The server-side `auth_simple` plugin communicates with the client program that it should use the client-side `auth_simple` plugin, and the latter sends the password to the server. The server plugin should reject connections that send an empty

password and accept connections that send a nonempty password. Invoke the client program each way to verify this:

```
shell> mysql --user=x --skip-password
ERROR 1045 (28000): Access denied for user 'x'@'localhost' (using password: NO)

shell> mysql --user=x --password
Enter password: abc
mysql>
```

Because the server plugin accepts any nonempty password, it should be considered insecure. After testing the plugin to verify that it works, restart the server without the `--plugin-load` option so as not to indavertently leave the server running with an insecure authentication plugin loaded. Also, drop the user with `DROP USER 'x'@'localhost'`.

For additional information about loading and using authentication plugins, see Section 5.6.1, “Installing and Uninstalling Plugins”, and Section 6.2.17, “Pluggable Authentication”.

If you are writing a client program that supports the use of authentication plugins, normally such a program causes a plugin to be loaded by calling `mysql_options()` to set the `MYSQL_DEFAULT_AUTH` and `MYSQL_PLUGIN_DIR` options:

```
char *plugin_dir = "path_to_plugin_dir";
char *default_auth = "plugin_name";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
mysql_options(&mysql, MYSQL_DEFAULT_AUTH, default_auth);
```

Typically, the program will also accept `--plugin-dir` and `--default-auth` options that enable users to override the default values.

Should a client program require lower-level plugin management, the client library contains functions that take an `st_mysql_client_plugin` argument. See Section 28.7.17, “C API Client Plugin Functions”.

Implementing Proxy User Support in Authentication Plugins

One of the capabilities that pluggable authentication makes possible is proxy users (see Section 6.2.18, “Proxy Users”). For a server-side authentication plugin to participate in proxy user support, these conditions must be satisfied:

- When a connecting client should be treated as a proxy user, the plugin must return a different name in the `authenticated_as` member of the `MYSQL_SERVER_AUTH_INFO` structure, to indicate the proxied user name. It may also optionally set the `external_user` member, to set the value of the `external_user` system variable.
- Proxy user accounts must be set up to be authenticated by the plugin. Use the `CREATE USER` or `GRANT` statement to associate accounts with plugins.
- Proxy user accounts must have the `PROXY` privilege for the proxied accounts. Use the `GRANT` statement to grant this privilege.

In other words, the only aspect of proxy user support required of the plugin is that it set `authenticated_as` to the proxied user name. The rest is optional (setting `external_user`) or done by the DBA using SQL statements.

How does an authentication plugin determine which proxied user to return when the proxy user connects? That depends on the plugin. Typically, the plugin maps clients to proxied users based on the authentication

string passed to it by the server. This string comes from the `AS` part of the `IDENTIFIED WITH` clause of the `CREATE USER` statement that specifies use of the plugin for authentication.

The plugin developer determines the syntax rules for the authentication string and implements the plugin according to those rules. Suppose that a plugin takes a comma-separated list of pairs that map external users to MySQL users. For example:

```
CREATE USER ''@'%.example.com'
  IDENTIFIED WITH my_plugin AS 'extuser1=mysqlusera, extuser2=mysqluserb'
CREATE USER ''@'%.example.org'
  IDENTIFIED WITH my_plugin AS 'extuser1=mysqluserc, extuser2=mysqluserd'
```

When the server invokes a plugin to authenticate a client, it passes the appropriate authentication string to the plugin. The plugin is responsible to:

1. Parse the string into its components to determine the mapping to use
2. Compare the client user name to the mapping
3. Return the proper MySQL user name

For example, if `extuser2` connects from an `example.com` host, the server passes `'extuser1=mysqlusera, extuser2=mysqluserb'` to the plugin, and the plugin should copy `mysqluserb` into `authenticated_as`, with a terminating null byte. If `extuser2` connects from an `example.org` host, the server passes `'extuser1=mysqluserc, extuser2=mysqluserd'`, and the plugin should copy `mysqluserd` instead.

If there is no match in the mapping, the action depends on the plugin. If a match is required, the plugin likely will return an error. Or the plugin might simply return the client name; in this case, it should not change `authenticated_as`, and the server will not treat the client as a proxy.

The following example demonstrates how to handle proxy users using a plugin named `auth_simple_proxy`. Like the `auth_simple` plugin described earlier, `auth_simple_proxy` accepts any nonempty password as valid (and thus should not be used in production environments). In addition, it examines the `auth_string` authentication string member and uses these very simple rules for interpreting it:

- If the string is empty, the plugin returns the user name as given and no proxying occurs. That is, the plugin leaves the value of `authenticated_as` unchanged.
- If the string is nonempty, the plugin treats it as the name of the proxied user and copies it to `authenticated_as` so that proxying occurs.

For testing, set up one account that is not proxied according to the preceding rules, and one that is. This means that one account has no `AS` clause, and one includes an `AS` clause that names the proxied user:

```
CREATE USER 'plugin_user1'@'localhost'
  IDENTIFIED WITH auth_simple_proxy;
CREATE USER 'plugin_user2'@'localhost'
  IDENTIFIED WITH auth_simple_proxy AS 'proxied_user';
```

In addition, create an account for the proxied user and grant `plugin_user2` the `PROXY` privilege for it:

```
CREATE USER 'proxied_user'@'localhost'
  IDENTIFIED BY 'proxied_user_pass';
GRANT PROXY
  ON 'proxied_user'@'localhost'
  TO 'plugin_user2'@'localhost';
```

Before the server invokes an authentication plugin, it sets `authenticated_as` to the client user name. To indicate that the user is a proxy, the plugin should set `authenticated_as` to the proxied user name. For `auth_simple_proxy`, this means that it must examine the `auth_string` value, and, if the value is nonempty, copy it to the `authenticated_as` member to return it as the name of the proxied user. In addition, when proxying occurs, the plugin sets the `external_user` member to the client user name; this becomes the value of the `external_user` system variable.

```
static int auth_simple_proxy_server (MYSQL_PLUGIN_VIO *vio,
                                     MYSQL_SERVER_AUTH_INFO *info)
{
  unsigned char *pkt;
  int pkt_len;

  /* read the password as null-terminated string, fail on error */
  if ((pkt_len= vio->read_packet(vio, &pkt)) < 0)
    return CR_ERROR;

  /* fail on empty password */
  if (!pkt_len || *pkt == '\0')
  {
    info->password_used= PASSWORD_USED_NO;
    return CR_ERROR;
  }

  /* accept any nonempty password */
  info->password_used= PASSWORD_USED_YES;

  /* if authentication string is nonempty, use as proxied user name */
  /* and use client name as external_user value */
  if (info->auth_string_length > 0)
  {
    strcpy (info->authenticated_as, info->auth_string);
    strcpy (info->external_user, info->user_name);
  }

  return CR_OK;
}
```

After a successful connection, the `USER()` function should indicate the connecting client user and host name, and `CURRENT_USER()` should indicate the account whose privileges apply during the session. The latter value should be the connecting user account if no proxying occurs or the proxied account if proxying does occur.

Compile and install the plugin, then test it. First, connect as `plugin_user1`:

```
shell> mysql --user=plugin_user1 --password
Enter password: x
```

In this case, there should be no proxying:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user, @@external_user\G
*************************** 1. row ***************************
        USER(): plugin_user1@localhost
 CURRENT_USER(): plugin_user1@localhost
   @@proxy_user: NULL
@@external_user: NULL
```

Then connect as `plugin_user2`:

```
shell> mysql --user=plugin_user2 --password
```

```
Enter password: x
```

In this case, `plugin_user2` should be proxied to `proxied_user`:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user, @@external_user\G
*************************** 1. row ***************************
         USER(): plugin_user2@localhost
 CURRENT_USER(): proxied_user@localhost
   @@proxy_user: 'plugin_user2'@'localhost'
@@external_user: 'plugin_user2'@'localhost'
```

29.2.4.10 Writing Password-Validation Plugins

This section describes how to write a server-side password-validation plugin. The instructions are based on the source code in the `plugin/password_validation` directory of MySQL source distributions. The `validate_password.cc` source file in that directory implements the plugin named `validate_password`.

Note

In MySQL 8.0.4, the `validate_password` plugin was reimplemented as the `validate_password` component. The plugin form of `validate_password` is still available but is now deprecated and will be removed in a future version of MySQL. MySQL installations that use the plugin should make the transition to using the component instead. See Section 6.4.3.3, “Transitioning to the Password Validation Component”.

To write a password-validation plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin_validate_password.h>
```

`plugin_validate_password.h` includes `plugin.h`, so you need not include the latter file explicitly. `plugin.h` defines the `MYSQL_VALIDATE_PASSWORD_PLUGIN` server plugin type and the data structures needed to declare the plugin. `plugin_validate_password.h` defines data structures specific to password-validation plugins.

A password-validation plugin, like any MySQL server plugin, has a general plugin descriptor (see Server Plugin Library and Plugin Descriptors). In `validate_password.cc`, the general descriptor for `validate_password` looks like this:

```
mysql_declare_plugin(validate_password)
{
  MYSQL_VALIDATE_PASSWORD_PLUGIN,     /*   type                            */
  &validate_password_descriptor,      /*   descriptor                      */
  "validate_password",                /*   name                            */
  "Oracle Corporation",               /*   author                          */
  "check password strength",          /*   description                     */
  PLUGIN_LICENSE_GPL,
  validate_password_init,             /*   init function (when loaded)     */
  validate_password_deinit,           /*   deinit function (when unloaded) */
  0x0100,                             /*   version                         */
  NULL,
  validate_password_system_variables, /*   system variables                */
  NULL,
  0,
}
mysql_declare_plugin_end;
```

The `name` member (`validate_password`) indicates the name to use for references to the plugin in statements such as `INSTALL PLUGIN` or `UNINSTALL PLUGIN`. This is also the name displayed by `INFORMATION_SCHEMA.PLUGINS` or `SHOW PLUGINS`.

The general descriptor also refers to `validate_password_system_variables`, a structure that exposes several system variables to the `SHOW VARIABLES` statement:

```
static struct st_mysql_sys_var* validate_password_system_variables[]= {
  MYSQL_SYSVAR(length),
  MYSQL_SYSVAR(number_count),
  MYSQL_SYSVAR(mixed_case_count),
  MYSQL_SYSVAR(special_char_count),
  MYSQL_SYSVAR(policy),
  MYSQL_SYSVAR(dictionary_file),
  NULL
};
```

The `validate_password_init` initialization function reads the dictionary file if one was specified, and the `validate_password_deinit` function frees data structures associated with the file.

The `validate_password_descriptor` value in the general descriptor points to the type-specific descriptor. For password-validation plugins, this descriptor has the following structure:

```
struct st_mysql_validate_password
{
  int interface_version;
  /*
    This function returns TRUE for passwords which satisfy the password
    policy (as chosen by plugin variable) and FALSE for all other
    password
  */
  int (*validate_password)(mysql_string_handle password);
  /*
    This function returns the password strength (0-100) depending
    upon the policies
  */
  int (*get_password_strength)(mysql_string_handle password);
};
```

The type-specific descriptor has these members:

- `interface_version`: By convention, type-specific plugin descriptors begin with the interface version for the given plugin type. The server checks `interface_version` when it loads the plugin to see whether the plugin is compatible with it. For password-validation plugins, the value of the `interface_version` member is `MYSQL_VALIDATE_PASSWORD_INTERFACE_VERSION` (defined in `plugin_validate_password.h`).
- `validate_password`: A function that the server calls to test whether a password satisfies the current password policy. It returns 1 if the password is okay and 0 otherwise. The argument is the password, passed as a `mysql_string_handle` value. This data type is implemented by the `mysql_string` server service. For details, see the `string_service.h` and `string_service.cc` source files in the `sql` directory.
- `get_password_strength`: A function that the server calls to assess the strength of a password. It returns a value from 0 (weak) to 100 (strong). The argument is the password, passed as a `mysql_string_handle` value.

For the `validate_password` plugin, the type-specific descriptor looks like this:

```
```

```
static struct st_mysql_validate_password validate_password_descriptor=
{
  MYSQL_VALIDATE_PASSWORD_INTERFACE_VERSION,
  validate_password,                        /* validate function          */
  get_password_strength                     /* validate strength function */
};
```

To compile and install a plugin library file, use the instructions in Section 29.2.4.3, “Compiling and Installing Plugin Libraries”. To make the library file available for use, install it in the plugin directory (the directory named by the `plugin_dir` system variable). For the `validate_password` plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of `validate_password.so` (the `.so` suffix might differ depending on your platform).

To register the plugin at runtime, use this statement (adjust the `.so` suffix for your platform as necessary):

```
INSTALL PLUGIN validate_password SONAME 'validate_password.so';
```

For additional information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement. See Section 5.6.2, “Obtaining Server Plugin Information”.

While the `validate_password` plugin is installed, it exposes system variables that indicate the password-checking parameters:

```
mysql> SHOW VARIABLES LIKE 'validate_password%';
+--------------------------------------+--------+
| Variable_name                        | Value  |
+--------------------------------------+--------+
| validate_password_dictionary_file    |        |
| validate_password_length             | 8      |
| validate_password_mixed_case_count   | 1      |
| validate_password_number_count       | 1      |
| validate_password_policy             | MEDIUM |
| validate_password_special_char_count | 1      |
+--------------------------------------+--------+
```

For descriptions of these variables, see Section 6.4.3.2, “Password Validation Options and Variables”.

To disable the plugin after testing it, use this statement to unload it:

```
UNINSTALL PLUGIN validate_password;
```

29.2.4.11 Writing Protocol Trace Plugins

MySQL supports the use of protocol trace plugins: client-side plugins that implement tracing of communication between a client and the server that takes place using the client/server protocol.

Using the Test Protocol Trace Plugin

MySQL includes a test protocol trace plugin that serves to illustrate the information available from such plugins, and as a guide to writing other protocol trace plugins. To see how the test plugin works, use a MySQL source distribution; binary distributions are built with the test plugin disabled.

Enable the test protocol trace plugin by configuring MySQL with the `WITH_TEST_TRACE_PLUGIN` `CMake` option enabled. This causes the test trace plugin to be built and MySQL client programs to load it, but the plugin has no effect by default. Control the plugin using these environment variables:

- `MYSQL_TEST_TRACE_DEBUG`: Set this variable to a value other than 0 to cause the test plugin to produce diagnostic output on `stderr`.
- `MYSQL_TEST_TRACE_CRASH`: Set this variable to a value other than 0 to cause the test plugin to abort the client program if it detects an invalid trace event.

Caution

Diagnostic output from the test protocol trace plugin can disclose passwords and other sensitive information.

Given a MySQL installation built from source with the test plugin enabled, you can see a trace of the communication between the `mysql` client and the MySQL server as follows:

```
shell> export MYSQL_TEST_TRACE_DEBUG=1
shqll> mysql
test_trace: Test trace plugin initialized
test_trace: Starting tracing in stage CONNECTING
test_trace: stage: CONNECTING, event: CONNECTING
test_trace: stage: CONNECTING, event: CONNECTED
test_trace: stage: WAIT_FOR_INIT_PACKET, event: READ_PACKET
test_trace: stage: WAIT_FOR_INIT_PACKET, event: PACKET_RECEIVED
test_trace: packet received: 87 bytes
  0A 35 2E 37 2E 33 2D 6D  31 33 2D 64 65 62 75 67   .5.7.3-m13-debug
  2D 6C 6F 67 00 04 00 00  00 2B 7C 4F 55 3F 79 67   -log.....+|OU?yg
test_trace: 004: stage: WAIT_FOR_INIT_PACKET, event: INIT_PACKET_RECEIVED
test_trace: 004: stage: AUTHENTICATE, event: AUTH_PLUGIN
test_trace: 004: Using authentication plugin: mysql_native_password
test_trace: 004: stage: AUTHENTICATE, event: SEND_AUTH_RESPONSE
test_trace: 004: sending packet: 188 bytes
  85 A6 7F 00 00 00 01  21 00 00 00 00 00 00 00   .?......!.......
  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00   ................
...
mysql> quit
test_trace: 008: stage: READY_FOR_COMMAND, event: SEND_COMMAND
test_trace: 008: QUIT
test_trace: 008: stage: READY_FOR_COMMAND, event: PACKET_SENT
test_trace: 008: packet sent: 0 bytes
test_trace: 008: stage: READY_FOR_COMMAND, event: DISCONNECTED
test_trace: 008: Connection  closed
test_trace: 008: Tracing connection has ended
Bye
test_trace: Test trace plugin de-initialized
```

To disable trace output, do this:

```
shell> MYSQL_TEST_TRACE_DEBUG=
```

Using Your Own Protocol Trace Plugins

Note

To use your own protocol trace plugins, you must configure MySQL with the `WITH_TEST_TRACE_PLUGIN CMake` option *disabled* because only one protocol trace plugin can be loaded at a time and an error occurs for attempts to load a second one. If you have already built MySQL with the test protocol trace plugin enabled to see how it works, you must rebuild MySQL without it before you can use your own plugins.

This section discusses how to write a basic protocol trace plugin named `simple_trace`. This plugin provides a framework showing how to set up the client plugin descriptor and create the trace-related

callback functions. In `simple_trace`, these functions are rudimentary and do little other than illustrate the arguments required. To see in detail how a trace plugin can make use of trace event information, check the source file for the test protocol trace plugin (`test_trace_plugin.cc` in the `libmysql` directory of a MySQL source distribution). However, note that the `st_mysql_client_plugin_TRACE` structure used there differs from the structures used with the usual client plugin declaration macros. In particular, the first two members are defined explicitly, not implicitly by declaration macros.

Several header files contain information relevant to protocol trace plugins:

- `client_plugin.h`: Defines the API for client plugins. This includes the client plugin descriptor and function prototypes for client plugin C API calls (see Section 28.7.17, “C API Client Plugin Functions”).
- `plugin_trace.h`: Contains declarations for client-side plugins of type `MYSQL_CLIENT_TRACE_PLUGIN`. It also contains descriptions of the permitted protocol stages, transitions between stages, and the types of events permitted at each stage.

To write a protocol trace plugin, include the following header files in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin_trace.h>
#include <mysql.h>
```

`plugin_trace.h` includes `client_plugin.h`, so you need not include the latter file explicitly.

Declare the client-side plugin descriptor with the `mysql_declare_client_plugin()` and `mysql_end_client_plugin` macros (see Client Plugin Descriptors). For the `simple_trace` plugin, the descriptor looks like this:

```
mysql_declare_client_plugin(TRACE)
  "simple_trace",                 /* plugin name */
  "Author Name",                  /* author */
  "Simple protocol trace plugin", /* description */
  {1,0,0},                        /* version = 1.0.0 */
  "GPL",                          /* license type */
  NULL,                           /* for internal use */
  plugin_init,                    /* initialization function */
  plugin_deinit,                  /* deinitialization function */
  plugin_options,                 /* option-handling function */
  trace_start,                    /* start-trace function */
  trace_stop,                     /* stop-trace function */
  trace_event                     /* event-handling function */
mysql_end_client_plugin;
```

The descriptor members from the plugin name through the option-handling function are common to all client plugin types. The members following the common members implement trace event handling.

Function members for which the plugin needs no processing can be declared as `NULL` in the descriptor, in which case you need not write any corresponding function. For illustration purposes and to show the argument syntax, the following discussion implements all functions listed in the descriptor, even though some of them do nothing,

The initialization, deinitialization, and options functions common to all client plugins are declared as follows. For a description of the arguments and return values, see Client Plugin Descriptors.

```
static int
plugin_init(char *errbuf, size_t errbuf_len, int argc, va_list args)
{
  return 0;
}
```

```
static int
plugin_deinit()
{
  return 0;
}

static int
plugin_options(const char *option, const void *value)
{
  return 0;
}
```

The trace-specific members of the client plugin descriptor are callback functions. The following descriptions provide more detail on how they are used. Each has a first argument that is a pointer to the plugin instance in case your implementation needs to access it.

`trace_start()`: This function is called at the start of each traced connection (each connection that starts after the plugin is loaded). It is passed the connection handler and the protocol stage at which tracing starts. `trace_start()` allocates memory needed by the `trace_event()` function, if any, and returns a pointer to it. If no memory is needed, this function returns `NULL`.

```
static void*
trace_start(struct st_mysql_client_plugin_TRACE *self,
            MYSQL *conn,
            enum protocol_stage stage)
{
  struct st_trace_data *plugin_data= malloc(sizeof(struct st_trace_data));

  fprintf(stderr, "Initializing trace: stage %d\n", stage);
  if (plugin_data)
  {
    memset(plugin_data, 0, sizeof(struct st_trace_data));
    fprintf(stderr, "Trace initialized\n");
    return plugin_data;
  }
  fprintf(stderr, "Could not initialize trace\n");
  exit(1);
}
```

`trace_stop()`: This function is called when tracing of the connection ends. That usually happens when the connection is closed, but can happen earlier. For example, `trace_event()` can return a nonzero value at any time and that causes tracing of the connection to terminate. `trace_stop()` is then called even though the connection has not ended.

`trace_stop()` is passed the connection handler and a pointer to the memory allocated by `trace_start()` (`NULL` if none). If the pointer is non-`NULL`, `trace_stop()` should deallocate the memory. This function returns no value.

```
static void
trace_stop(struct st_mysql_client_plugin_TRACE *self,
           MYSQL *conn,
           void *plugin_data)
{
  fprintf(stderr, "Terminating trace\n");
  if (plugin_data)
    free(plugin_data);
}
```

`trace_event()`: This function is called for each event occurrence. It is passed a pointer to the memory allocated by `trace_start()` (`NULL` if none), the connection handler, the current protocol stage and event codes, and event data. This function returns 0 to continue tracing, nonzero if tracing should stop.

```
static int
trace_event(struct st_mysql_client_plugin_TRACE *self,
            void *plugin_data,
            MYSQL *conn,
            enum protocol_stage stage,
            enum trace_event event,
            struct st_trace_event_args args)
{
  fprintf(stderr, "Trace event received: stage %d, event %d\n", stage, event);
  if (event == TRACE_EVENT_DISCONNECTED)
    fprintf(stderr, "Connection closed\n");
  return 0;
}
```

The tracing framework shuts down tracing of the connection when the connection ends, so `trace_event()` should return nonzero only if you want to terminate tracing of the connection early. Suppose that you want to trace only connections for a certain MySQL account. After authentication, you can check the user name for the connection and stop tracing if it is not the user in whom you are interested.

For each call to `trace_event()`, the `st_trace_event_args` structure contains the event data. It has this definition:

```
struct st_trace_event_args
{
  const char           *plugin_name;
  int                   cmd;
  const unsigned char  *hdr;
  size_t                hdr_len;
  const unsigned char  *pkt;
  size_t                pkt_len;
};
```

For different event types, the `st_trace_event_args` structure contains the information described following. All lengths are in bytes. Unused members are set to `0`/`NULL`.

`AUTH_PLUGIN` event:

```
plugin_name  The name of the plugin
```

`SEND_COMMAND` event:

```
cmd          The command code
hdr          Pointer to the command packet header
hdr_len      Length of the header
pkt          Pointer to the command arguments
pkt_len      Length of the arguments
```

Other `SEND_`*`xxx`* and *`xxx`*`_RECEIVED` events:

```
pkt          Pointer to the data sent or received
pkt_len      Length of the data
```

`PACKET_SENT` event:

```
pkt_len      Number of bytes sent
```

To compile and install a plugin library file, use the instructions in Section 29.2.4.3, “Compiling and Installing Plugin Libraries”. To make the library file available for use, install it in the plugin directory (the directory named by the `plugin_dir` system variable).

After the plugin library file is compiled and installed in the plugin directory, you can test it easily by setting the `LIBMYSQL_PLUGINS` environment variable to the plugin name, which affects any client program that uses that variable. `mysql` is one such program:

```
shell> export LIBMYSQL_PLUGINS=simple_trace
shqll> mysql
Initializing trace: stage 0
Trace initialized
Trace event received: stage 0, event 1
Trace event received: stage 0, event 2
...
Welcome to the MySQL monitor.  Commands end with ; or \g.
Trace event received
Trace event received
...
mysql> SELECT 1;
Trace event received: stage 4, event 12
Trace event received: stage 4, event 16
...
Trace event received: stage 8, event 14
Trace event received: stage 8, event 15
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)

mysql> quit
Trace event received: stage 4, event 12
Trace event received: stage 4, event 16
Trace event received: stage 4, event 3
Connection closed
Terminating trace
Bye
```

To stop the trace plugin from being loaded, do this:

```
shell> LIBMYSQL_PLUGINS=
```

It is also possible to write client programs that directly load the plugin. You can tell the client where the plugin directory is located by calling `mysql_options()` to set the `MYSQL_PLUGIN_DIR` option:

```
char *plugin_dir = "path_to_plugin_dir";

/* ... process command-line options ... */

mysql_options(&mysql, MYSQL_PLUGIN_DIR, plugin_dir);
```

Typically, the program will also accept a `--plugin-dir` option that enables users to override the default value.

Should a client program require lower-level plugin management, the client library contains functions that take an `st_mysql_client_plugin` argument. See Section 28.7.17, “C API Client Plugin Functions”.

29.2.4.12 Writing Keyring Plugins

MySQL Server supports a keyring service that enables internal server components and plugins to securely store sensitive information for later retrieval. This section describes how to write a server-side keyring plugin that can be used by service functions to perform key-management operations. For general keyring information, see Section 6.4.4, “The MySQL Keyring”.

The instructions here are based on the source code in the `plugin/keyring` directory of MySQL source distributions. The source files in that directory implement a plugin named `keyring_file` that uses a file local to the server host for data storage.

To write a keyring plugin, include the following header file in the plugin source file. Other MySQL or general header files might also be needed, depending on the plugin capabilities and requirements.

```
#include <mysql/plugin_keyring.h>
```

`plugin_keyring.h` includes `plugin.h`, so you need not include the latter file explicitly. `plugin.h` defines the `MYSQL_KEYRING_PLUGIN` server plugin type and the data structures needed to declare the plugin. `plugin_keyring.h` defines data structures specific to keyring plugins.

A keyring plugin, like any MySQL server plugin, has a general plugin descriptor (see Server Plugin Library and Plugin Descriptors). In `keyring.cc`, the general descriptor for `keyring_file` looks like this:

```
mysql_declare_plugin(keyring_file)
{
  MYSQL_KEYRING_PLUGIN,     /* type                                  */
  &keyring_descriptor,      /* descriptor                            */
  "keyring_file",           /* name                                  */
  "Oracle Corporation",     /* author                                */
  "store/fetch authentication data to/from a flat file", /* description */
  PLUGIN_LICENSE_GPL,
  keyring_init,             /* init function (when loaded)           */
  keyring_deinit,           /* deinit function (when unloaded)       */
  0x0100,                   /* version                               */
  NULL,                     /* status variables                      */
  keyring_system_variables, /* system variables                      */
  NULL,
  0,
}
mysql_declare_plugin_end;
```

The `name` member (`keyring_file`) indicates the plugin name. This is the name displayed by `INFORMATION_SCHEMA.PLUGINS` or `SHOW PLUGINS`.

The general descriptor also refers to `keyring_system_variables`, a structure that exposes a system variable to the `SHOW VARIABLES` statement:

```
static struct st_mysql_sys_var *keyring_system_variables[]= {
  MYSQL_SYSVAR(data),
  NULL
};
```

The `keyring_init` initialization function creates the data file if it does not exist, then reads it and initializes the keystore. The `keyring_deinit` function frees data structures associated with the file.

The `keyring_descriptor` value in the general descriptor points to the type-specific descriptor. For keyring plugins, this descriptor has the following structure:

```
struct st_mysql_keyring
{
```

```
  int interface_version;
  bool (*mysql_key_store)(const char *key_id, const char *key_type,
                          const char* user_id, const void *key, size_t key_len);
  bool (*mysql_key_fetch)(const char *key_id, char **key_type,
                          const char *user_id, void **key, size_t *key_len);
  bool (*mysql_key_remove)(const char *key_id, const char *user_id);
  bool (*mysql_key_generate)(const char *key_id, const char *key_type,
                             const char *user_id, size_t key_len);
};
```

The type-specific descriptor has these members:

- `interface_version`: By convention, type-specific plugin descriptors begin with the interface version for the given plugin type. The server checks `interface_version` when it loads the plugin to see whether the plugin is compatible with it. For keyring plugins, the value of the `interface_version` member is `MYSQL_KEYRING_INTERFACE_VERSION` (defined in `plugin_keyring.h`).
- `mysql_key_store`: A function that obfuscates and stores a key in the keyring.
- `mysql_key_fetch`: A function that deobfuscates and retrieves a key from the keyring.
- `mysql_key_remove`: A function that removes a key from the keyring.
- `mysql_key_generate`: A function that generates a new random key and stores it in the keyring.

For the `keyring_file` plugin, the type-specific descriptor looks like this:

```
static struct st_mysql_keyring keyring_descriptor=
{
  MYSQL_KEYRING_INTERFACE_VERSION,
  mysql_key_store,
  mysql_key_fetch,
  mysql_key_remove,
  mysql_key_generate
};
```

The `mysql_key_`*`xxx`* functions implemented by a keyring plugin are analogous to the `my_key_`*`xxx`* functions exposed by the keyring service API. For example, the `mysql_key_store` plugin function is analogous to the `my_key_store` keyring service function. For information about the arguments to keyring service functions and how they are used, see Section 29.3.2, “The Keyring Service”.

To compile and install a plugin library file, use the instructions in Section 29.2.4.3, “Compiling and Installing Plugin Libraries”. To make the library file available for use, install it in the plugin directory (the directory named by the `plugin_dir` system variable). For the `keyring_file` plugin, it is compiled and installed when you build MySQL from source. It is also included in binary distributions. The build process produces a shared object library with a name of `keyring_file.so` (the `.so` suffix might differ depending on your platform).

Keyring plugins typically are loaded early during the server startup process so that they are available to built-in plugins and storage engines that might depend on them. For `keyring_file`, use these lines in the server `my.cnf` file (adjust the `.so` suffix for your platform as necessary):

```
[mysqld]
early-plugin-load=keyring_file.so
```

For additional information about plugin loading, see Section 5.6.1, “Installing and Uninstalling Plugins”.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.6.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'keyring%';
+--------------+---------------+
| PLUGIN_NAME  | PLUGIN_STATUS |
+--------------+---------------+
| keyring_file | ACTIVE        |
+--------------+---------------+
```

While the `keyring_file` plugin is installed, it exposes a system variable that indicates the location of the data file it uses for secure information storage:

```
mysql> SHOW VARIABLES LIKE 'keyring_file%';
+-------------------+---------------------------------+
| Variable_name     | Value                           |
+-------------------+---------------------------------+
| keyring_file_data | /usr/local/mysql/keyring/keyring |
+-------------------+---------------------------------+
```

For a description of the `keyring_file_data` variable, see Section 5.1.8, “Server System Variables”.

To disable the plugin after testing it, restart the server without an `--early-plugin-load` option that names the plugin.

29.3 MySQL Services for Plugins

MySQL server plugins have access to server “plugin services.” The plugin services interface exposes server functionality that plugins can call. It complements the plugin API and has these characteristics:

- Services enable plugins to access code inside the server using ordinary function calls. Services are also available to user-defined functions (UDFs).
- Services are portable and work on multiple platforms.
- The interface includes a versioning mechanism so that service versions supported by the server can be checked at load time against plugin versions. Versioning protects against incompatibilities between the version of a service that the server provides and the version of the service expected or required by a plugin.
- For information about plugins for testing plugin services, see the Plugins for Testing Plugin Services section of the MySQL Server Doxygen documentation, available at https://dev.mysql.com/doc/index-other.html.

The plugin services interface differs from the plugin API as follows:

- The plugin API enables plugins to be used by the server. The calling initiative lies with the server to invoke plugins. This enables plugins to extend server functionality or register to receive notifications about server processing.
- The plugin services interface enables plugins to call code inside the server. The calling initiative lies with plugins to invoke service functions. This enables functionality already implemented in the server to be used by many plugins; they need not individually implement it themselves.

To determine what services exist and what functions they provide, look in the `include/mysql` directory of a MySQL source distribution. The relevant files are:

- `plugin.h` includes `services.h`, which is the “umbrella” header that includes all available service-specific header files.

- Service-specific headers have names of the form `service_xxx.h`.

Each service-specific header should contain comments that provide full usage documentation for a given service, including what service functions are available, their calling sequences, and return values.

For developers who wish to modify the server to add a new service, see MySQL Internals: MySQL Services for Plugins.

Available services include the following:

- `get_sysvar_source`: A service that enables plugins to retrieve the source of system variable settings.
- `locking_service`: A service that implements locks with three attributes: Lock namespace, lock name, and lock mode. This locking interface is available at two levels: 1) As a C language interface, callable as a plugin service from server plugins or user-defined functions; 2) At the SQL level, as a set of user-defined functions that map onto calls to the service routines. For more information, see Section 29.3.1, “The Locking Service”.
- `my_plugin_log_service`: A service that enables plugins to report errors and specify error messages. The server writes the messages to its error log.
- `status_variable_registration`. A service for registering status variables.
- `my_thd_scheduler`: A service for plugins to select a thread scheduler.
- `mysql_keyring`: A service for keyring storage. For more information, see Section 29.3.2, “The Keyring Service”.
- `mysql_password_policy`: A service for password validation and strength checking.
- `plugin_registry_service`: MySQL Server includes a component-based infrastructure for improving server extensibility; see Section 5.5, “MySQL Server Components”. However, MySQL plugins use an interface that predates the component interface. The `plugin_registry_service` enables plugins to access the component registry and its services.
- `security_context`: A service that enables plugins to examine or manipulate thread security contexts. This service provides setter and getter routines to access attributes of the server `Security_context` class, which includes attributes such as operating system user and host, authenticated user and host, and client IP address.
- `thd_alloc`: A memory-allocation service.
- `thd_wait`: A service for plugins to report when they are going to sleep or stall.

The remainder of this section describes how a plugin uses server functionality that is available as a service. See also the source for the “daemon” example plugin, which uses the `my_snprintf` service. Within a MySQL source distribution, that plugin is located in the `plugin/daemon_example` directory.

To use a service or services from within a plugin, the plugin source file must include the `plugin.h` header file to access service-related information:

```
#include <mysql/plugin.h>
```

This does not represent any additional setup cost. A plugin must include that file anyway because it contains definitions and structures that every plugin needs.

To access a service, a plugin calls service functions like any other function.

To report an error that the server will write to it error log, first choose an error level. `mysql/service_my_plugin_log.h` defines these levels:

```
enum plugin_log_level
{
  MY_ERROR_LEVEL,
  MY_WARNING_LEVEL,
  MY_INFORMATION_LEVEL
};
```

Then invoke `my_plugin_log_message()`:

```
int my_plugin_log_message(MYSQL_PLUGIN *plugin, enum plugin_log_level level,
                          const char *format, ...);
```

For example:

```
my_plugin_log_message(plugin_ptr, MY_ERROR_LEVEL, "Cannot initialize plugin");
```

Some services *for* plugins may be provided *by* plugins and thus are available only if the service-providing plugin is loaded. Any MySQL component that uses such a service should check whether the service is available.

When you build your plugin, use the `-lmysqlservices` flag at link time to link in the `libmysqlservices` library. For example, for `CMake`, put this in the top-level `CMakeLists.txt` file:

```
FIND_LIBRARY(MYSQLSERVICES_LIB mysqlservices
 PATHS "${MYSQL_SRCDIR}/libservices" NO_DEFAULT_PATH)
```

Put this in the `CMakeLists.txt` file in the directory containing the plugin source:

```
# the plugin needs the mysql services library for error logging
TARGET_LINK_LIBRARIES (your_plugin_library_name ${MYSQLSERVICES_LIB})
```

29.3.1 The Locking Service

MySQL distributions provide a locking interface that is available at two levels:

- As a C language interface, callable as a plugin service from server plugins or user-defined functions
- At the SQL level, as a set of user-defined functions that map onto calls to the service routines

For general information about plugin services, see Section 29.3, “MySQL Services for Plugins”. For general information about user-defined functions, see Section 29.4.2, “Adding a New User-Defined Function”.

The locking interface has these characteristics:

- Locks have three attributes: Lock namespace, lock name, and lock mode:
  - Locks are identified by the combination of namespace and lock name. The namespace enables different applications to use the same lock names without colliding by creating locks in separate namespaces. For example, if applications A and B use namespaces of `ns1` and `ns2`, respectively, each application can use lock names `lock1` and `lock2` without interfering with the other application.
  - A lock mode is either read or write. Read locks are shared: If a session has a read lock on a given lock identifier, other sessions can acquire a read lock on the same identifier. Write locks are exclusive: If a

session has a write lock on a given lock identifier, other sessions cannot acquire a read or write lock on the same identifier.

- Namespace and lock names must be non-`NULL`, nonempty, and have a maximum length of 64 characters. A namespace or lock name specified as `NULL`, the empty string, or a string longer than 64 characters results in an `ER_LOCKING_SERVICE_WRONG_NAME` error.
- The locking interface treats namespace and lock names as binary strings, so comparisons are case-sensitive.
- The locking interface provides functions to acquire locks and release locks. No special privilege is required to call these functions. Privilege checking is the responsibility of the calling application.
- Locks can be waited for if not immediately available. Lock acquisition calls take an integer timeout value that indicates how many seconds to wait to acquire locks before giving up. If the timeout is reached without successful lock acquisition, an `ER_LOCKING_SERVICE_TIMEOUT` error occurs. If the timeout is 0, there is no waiting and the call produces an error if locks cannot be acquired immediately.
- The locking interface detects deadlock between lock-acquisition calls in different sessions. In this case, the locking service chooses a caller and terminates its lock-acquisition request with an `ER_LOCKING_SERVICE_DEADLOCK` error. This error does not cause transactions to roll back. To choose a session in case of deadlock, the locking service prefers sessions that hold read locks over sessions that hold write locks.
- A session can acquire multiple locks with a single lock-acquisition call. For a given call, lock acquisition is atomic: The call succeeeds if all locks are acquired. If acquisition of any lock fails, the call acquires no locks and fails, typically with an `ER_LOCKING_SERVICE_TIMEOUT` or `ER_LOCKING_SERVICE_DEADLOCK` error.
- A session can acquire multiple locks for the same lock identifier (namespace and lock name combination). These lock instances can be read locks, write locks, or a mix of both.
- Locks acquired within a session are released explicitly by calling a release-locks function, or implicitly when the session terminates (either normally or abnormally). Locks are not released when transactions commit or roll back.
- Within a session, all locks for a given namespace when released are released together.

The interface provided by the locking service is distinct from that provided by `GET_LOCK()` and related SQL functions (see Section 12.14, “Locking Functions”). For example, `GET_LOCK()` does not implement namespaces and provides only exclusive locks, not distinct read and write locks.

29.3.1.1 The Locking Service C Interface

This section describes how to use the locking service C language interface. To use the UDF interface instead, see Section 29.3.1.2, “The Locking Service UDF Interface” For general characteristics of the locking service interface, see Section 29.3.1, “The Locking Service”. For general information about plugin services, see Section 29.3, “MySQL Services for Plugins”.

Source files that use the locking service should include this header file:

```
#include <mysql/service_locking.h>
```

To acquire one or more locks, call this function:

```
int mysql_acquire_locking_service_locks(MYSQL_THD opaque_thd,
                                        const char* lock_namespace,
```

```
                                        const char**lock_names,
                                        size_t lock_num,
                                        enum enum_locking_service_lock_type lock_type,
                                        unsigned long lock_timeout);
```

The arguments have these meanings:

- `opaque_thd`: A thread handle. If specified as `NULL`, the handle for the current thread is used.
- `lock_namespace`: A null-terminated string that indicates the lock namespace.
- `lock_names`: An array of null-terminated strings that provides the names of the locks to acquire.
- `lock_num`: The number of names in the `lock_names` array.
- `lock_type`: The lock mode, either `LOCKING_SERVICE_READ` or `LOCKING_SERVICE_WRITE` to acquire read locks or write locks, respectively.
- `lock_timeout`: An integer number of seconds to wait to acquire the locks before giving up.

To release locks acquired for a given namespace, call this function:

```
int mysql_release_locking_service_locks(MYSQL_THD opaque_thd,
                                        const char* lock_namespace);
```

The arguments have these meanings:

- `opaque_thd`: A thread handle. If specified as `NULL`, the handle for the current thread is used.
- `lock_namespace`: A null-terminated string that indicates the lock namespace.

Locks acquired or waited for by the locking service can be monitored at the SQL level using the Performance Schema. For details, see Locking Service Monitoring.

29.3.1.2 The Locking Service UDF Interface

This section describes how to use the locking service user-defined function (UDF) interface. To use the C language interface instead, see Section 29.3.1.1, “The Locking Service C Interface” For general characteristics of the locking service interface, see Section 29.3.1, “The Locking Service”. For general information about user-defined functions, see Section 29.4.2, “Adding a New User-Defined Function”.

Installing or Uninstalling the UDF Locking Interface

The locking service routines described in Section 29.3.1.1, “The Locking Service C Interface” need not be installed because they are built into the server. The same is not true of the user-defined functions (UDFs) that map onto calls to the service routines: The UDFs must be installed before use. This section describes how to do that. For general information about UDF installation, see Section 5.7.1, “Installing and Uninstalling User-Defined Functions”.

The locking service UDFs are implemented in a plugin library file located in the directory named by the `plugin_dir` system variable. The file base name is `locking_service`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To install the locking service UDFs, use the `CREATE FUNCTION` statement (adjust the `.so` suffix for your platform as necessary):

```
CREATE FUNCTION service_get_read_locks RETURNS INT
  SONAME 'locking_service.so';
CREATE FUNCTION service_get_write_locks RETURNS INT
```

```
  SONAME 'locking_service.so';
CREATE FUNCTION service_release_locks RETURNS INT
  SONAME 'locking_service.so';
```

If the UDFs are used on a master replication server, install them on all slave servers as well to avoid replication problems.

Once installed, the UDFs remain installed until uninstalled. To remove them, use the `DROP FUNCTION` statement:

```
DROP FUNCTION service_get_read_locks;
DROP FUNCTION service_get_write_locks;
DROP FUNCTION service_release_locks;
```

Using the UDF Locking Interface

Before using the locking service UDFs, install them according to the instructions provided at Installing or Uninstalling the UDF Locking Interface.

To acquire one or more read locks, call this function:

```
mysql> SELECT service_get_read_locks('mynamespace', 'rlock1', 'rlock2', 10);
+-------------------------------------------------------------+
| service_get_read_locks('mynamespace', 'rlock1', 'rlock2', 10) |
+-------------------------------------------------------------+
|                                                           1 |
+-------------------------------------------------------------+
```

The first argument is the lock namespace. The final argument is an integer timeout indicating how many seconds to wait to acquire the locks before giving up. The arguments in between are the lock names.

For the example just shown, the function acquires locks with lock identifiers `(mynamespace, rlock1)` and `(mynamespace, rlock2)`.

To acquire write locks rather than read locks, call this function:

```
mysql> SELECT service_get_write_locks('mynamespace', 'wlock1', 'wlock2', 10);
+-------------------------------------------------------------+
| service_get_write_locks('mynamespace', 'wlock1', 'wlock2', 10) |
+-------------------------------------------------------------+
|                                                           1 |
+-------------------------------------------------------------+
```

In this case, the lock identifiers are `(mynamespace, wlock1)` and `(mynamespace, wlock2)`.

To release all locks for a namespace, use this function:

```
mysql> SELECT service_release_locks('mynamespace');
+--------------------------------------+
| service_release_locks('mynamespace') |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
```

Each locking function returns nonzero for success. If the function fails, an error occurs. For example, the following error occurs because lock names cannot be empty:

```
mysql> SELECT service_get_read_locks('mynamespace', '', 10);
ERROR 3131 (42000): Incorrect locking service lock name ''.
```

A session can acquire multiple locks for the same lock identifier. As long as a different session does not have a write lock for an identifier, the session can acquire any number of read or write locks. Each lock request for the identifier acquires a new lock. The following statements acquire three write locks with the same identifier, then three read locks for the same identifier:

```
SELECT service_get_write_locks('ns', 'lock1', 'lock1', 'lock1', 0);
SELECT service_get_read_locks('ns', 'lock1', 'lock1', 'lock1', 0);
```

If you examine the Performance Schema `metadata_locks` table at this point, you will find that the session holds six distinct locks with the same `(ns, lock1)` identifier. (For details, see Locking Service Monitoring.)

Because the session holds at least one write lock on `(ns, lock1)`, no other session can acquire a lock for it, either read or write. If the session held only read locks for the identifier, other sessions could acquire read locks for it, but not write locks.

Locks for a single lock-acquisition call are acquired atomically, but atomicity does not hold across calls. Thus, for a statement such as the following, where `service_get_write_locks()` is called once per row of the result set, atomicity holds for each individual call, but not for the statement as a whole:

```
SELECT service_get_write_locks('ns', 'lock1', 'lock2', 0) FROM t1 WHERE ... ;
```

Caution

Because the locking service returns a separate lock for each successful request for a given lock identifier, it is possible for a single statement to acquire a large number of locks. For example:

```
INSERT INTO ... SELECT service_get_write_locks('ns', t1.col_name, 0) FROM t1;
```

These types of statements may have certain adverse effects. For example, if the statement fails part way through and rolls back, locks acquired up to the point of failure will still exist. If the intent is for there to be a correspondence between rows inserted and locks acquired, that intent will not be satisfied. Also, if it is important that locks are granted in a certain order, be aware that result set order may differ depending on which execution plan the optimizer chooses. For these reasons, it may be best to limit applications to a single lock-acquisition call per statement.

Locking Service Monitoring

The locking service is implemented using the MySQL Server metadata locks framework, so you monitor locking service locks acquired or waited for by examining the Performance Schema `metadata_locks` table.

First, enable the metadata lock instrument:

```
mysql> UPDATE performance_schema.setup_instruments SET ENABLED = 'YES'
    -> WHERE NAME = 'wait/lock/metadata/sql/mdl';
```

Then acquire some locks and check the contents of the `metadata_locks` table:

```
mysql> SELECT service_get_write_locks('mynamespace', 'lock1', 0);
+-----------------------------------------------------+
| service_get_write_locks('mynamespace', 'lock1', 0) |
+-----------------------------------------------------+
```

```
|                                                  1 |
+----------------------------------------------------+
mysql> SELECT service_get_read_locks('mynamespace', 'lock2', 0);
+----------------------------------------------------+
| service_get_read_locks('mynamespace', 'lock2', 0) |
+----------------------------------------------------+
|                                                  1 |
+----------------------------------------------------+
mysql> SELECT OBJECT_TYPE, OBJECT_SCHEMA, OBJECT_NAME, LOCK_TYPE, LOCK_STATUS
    -> FROM performance_schema.metadata_locks
    -> WHERE OBJECT_TYPE = 'LOCKING SERVICE'\G
*************************** 1. row ***************************
  OBJECT_TYPE: LOCKING SERVICE
OBJECT_SCHEMA: mynamespace
  OBJECT_NAME: lock1
    LOCK_TYPE: EXCLUSIVE
  LOCK_STATUS: GRANTED
*************************** 2. row ***************************
  OBJECT_TYPE: LOCKING SERVICE
OBJECT_SCHEMA: mynamespace
  OBJECT_NAME: lock2
    LOCK_TYPE: SHARED
  LOCK_STATUS: GRANTED
```

Locking service locks have an `OBJECT_TYPE` value of `LOCKING SERVICE`. This is distinct from, for example, locks acquired with the `GET_LOCK()` function, which have an `OBJECT_TYPE` of `USER LEVEL LOCK`.

The lock namespace, name, and mode appear in the `OBJECT_SCHEMA`, `OBJECT_NAME`, and `LOCK_TYPE` columns. Read and write locks have `LOCK_TYPE` values of `SHARED` and `EXCLUSIVE`, respectively.

The `LOCK_STATUS` value is `GRANTED` for an acquired lock, `PENDING` for a lock that is being waited for. You will see `PENDING` if one session holds a write lock and another session is attempting to acquire a lock having the same identifier.

Locking Service UDF Interface Reference

The SQL interface to the locking service implements the user-defined functions described in this section. For usage examples, see Using the UDF Locking Interface.

The functions share these characteristics:

- The return value is nonzero for success. Otherwise, an error occurs.
- Namespace and lock names must be non-`NULL`, nonempty, and have a maximum length of 64 characters.
- Timeout values must be integers indicating how many seconds to wait to acquire locks before giving up with an error. If the timeout is 0, there is no waiting and the function produces an error if locks cannot be acquired immediately.

These locking service UDFs are available:

- `service_get_read_locks(`*`namespace, lock_name`*`[, `*`lock_name`*`] ..., `*`timeout`*`)`

  Acquires one or more read (shared) locks in the given namespace using the given lock names, timing out with an error if the locks are not acquired within the given timeout value.

- `service_get_write_locks(`*`namespace, lock_name`*`[, `*`lock_name`*`] ..., `*`timeout`*`)`

  Acquires one or more write (exclusive) locks in the given namespace using the given lock names, timing out with an error if the locks are not acquired within the given timeout value.

- `service_release_locks(namespace)`

  For the given namespace, releases all locks that were acquired within the current session using `service_get_read_locks()` and `service_get_write_locks()`.

  It is not an error for there to be no locks in the namespace.

29.3.2 The Keyring Service

MySQL Server supports a keyring service that enables internal server components and plugins to securely store sensitive information for later retrieval. This section describes how to use the keyring service functions to store, retrieve, and remove keys in the MySQL keyring keystore. An SQL interface to the keyring service functions is also available as a set of user-defined functions (UDFs); see Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”. For general keyring information, see Section 6.4.4, “The MySQL Keyring”.

The keyring service uses whatever underlying keyring plugin is enabled, if any. If no keyring plugin is enabled, keyring service calls fail.

A “record” in the keystore consists of data (the key itself) and a unique identifier through which the key is accessed. The identifier has two parts:

- `key_id`: The key ID or name. `key_id` values that begin with `mysql_` are reserved by MySQL Server.
- `user_id`: The session effective user ID. If there is no user context, this value can be `NULL`. The value need not actually be a “user”; the meaning depends on the application.

  Functions that implement the keyring UDF interface pass the value of `CURRENT_USER()` as the `user_id` value to keyring service functions.

The keyring service functions have these characteristics in common:

- Each function returns 0 for success, 1 for failure.
- The `key_id` and `user_id` arguments form a unique combination indicating which key in the keyring to use.
- The `key_type` argument provides additional information about the key, such as its encryption method or intended use.
- Keyring service functions treat key IDs, user names, types, and values as binary strings, so comparisons are case sensitive. For example, IDs of `MyKey` and `mykey` refer to different keys.

These keyring service functions are available:

- `my_key_fetch()`

  Deobfuscates and retrieves a key from the keyring, along with its type. The function allocates the memory for the buffers used to store the returned key and key type. The caller should zero or obfuscate the memory when it is no longer needed, then free it.

  Syntax:

```
bool my_key_fetch(const char *key_id, const char **key_type,
                  const char* user_id, void **key, size_t *key_len)
```

  Arguments:

- `key_id`, `user_id`: Null-terminated strings that as a pair form a unique identifier indicating which key to fetch.
- `key_type`: The address of a buffer pointer. The function stores into it a pointer to a null-terminated string that provides additional information about the key (stored when the key was added).
- `key`: The address of a buffer pointer. The function stores into it a pointer to the buffer containing the fetched key data.
- `key_len`: The address of a variable into which the function stores the size in bytes of the `*key` buffer.

Return value:

Returns 0 for success, 1 for failure.

- `my_key_generate()`

  Generates a new random key of a given type and length and stores it in the keyring. The key has a length of `key_len` and is associated with the identifier formed from `key_id` and `user_id`. The type and length values must be consistent with the values supported by the underlying keyring plugin. See Section 6.4.4.7, “Supported Keyring Key Types and Lengths”.

  Syntax:

```
bool my_key_generate(const char *key_id, const char *key_type,
                     const char *user_id, size_t key_len)
```

  Arguments:

  - `key_id`, `user_id`: Null-terminated strings that as a pair form a unique identifier for the key to be generated.
  - `key_type`: A null-terminated string that provides additional information about the key.
  - `key_len`: The size in bytes of the key to be generated.

  Return value:

  Returns 0 for success, 1 for failure.

- `my_key_remove()`

  Removes a key from the keyring.

  Syntax:

```
bool my_key_remove(const char *key_id, const char* user_id)
```

  Arguments:

  - `key_id`, `user_id`: Null-terminated strings that as a pair form a unique identifier for the key to be removed.

  Return value:

  Returns 0 for success, 1 for failure.

- `my_key_store()`

  Obfuscates and stores a key in the keyring.

  Syntax:

  ```
  bool my_key_store(const char *key_id, const char *key_type,
                    const char* user_id, void *key, size_t key_len)
  ```

  Arguments:

  - `key_id`, `user_id`: Null-terminated strings that as a pair form a unique identifier for the key to be stored.
  - `key_type`: A null-terminated string that provides additional information about the key.
  - `key`: The buffer containing the key data to be stored.
  - `key_len`: The size in bytes of the `key` buffer.

  Return value:

  Returns 0 for success, 1 for failure.

29.4 Adding New Functions to MySQL

There are three ways to add new functions to MySQL:

- You can add functions through the user-defined function (UDF) interface. User-defined functions are compiled as library files and then added to and removed from the server dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. See Section 13.7.4.1, “CREATE FUNCTION Syntax for User-Defined Functions”.
- You can add functions as native (built-in) MySQL functions. Native functions are compiled into the `mysqld` server and become available on a permanent basis.
- Another way to add functions is by creating stored functions. These are written using SQL statements rather than by compiling object code. The syntax for writing stored functions is not covered here. See Section 24.2, “Using Stored Routines”.

Each method of creating compiled functions has advantages and disadvantages:

- If you write user-defined functions, you must install object files in addition to the server itself. If you compile your function into the server, you need not do that.
- Native functions require you to modify a source distribution. UDFs do not. You can add UDFs to a binary MySQL distribution. No access to MySQL source is necessary.
- If you upgrade your MySQL distribution, you can continue to use your previously installed UDFs, unless you upgrade to a newer version for which the UDF interface changes. For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they can be invoked in SQL statements just like native functions such as `ABS()` or `SOUNDEX()`.

See Section 9.2.5, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

The following sections describe features of the UDF interface, provide instructions for writing UDFs, discuss security precautions that MySQL takes to prevent UDF misuse, and describe how to add native MySQL functions.

For example source code that illustrates how to write UDFs, take a look at the `sql/udf_example.cc` file that is provided in MySQL source distributions.

Note

The MySQL source code contains internal documentation written using Doxygen. This documentation is useful for understanding how MySQL works from a developer perspective. The generated Doxygen content is available at https://dev.mysql.com/doc/index-other.html. It is also possible to generate this content locally from a MySQL source distribution using the instructions at Section 2.9.10, “Generating MySQL Doxygen Documentation Content”.

29.4.1 Features of the User-Defined Function Interface

The MySQL interface for user-defined functions provides the following features and capabilities:

- Functions can return string, integer, or real values and can accept arguments of those same types.
- You can define simple functions that operate on a single row at a time, or aggregate functions that operate on groups of rows.
- Information is provided to functions that enables them to check the number, types, and names of the arguments passed to them.
- You can tell MySQL to coerce arguments to a given type before passing them to a function.
- You can indicate that a function returns `NULL` or that an error occurred.

29.4.2 Adding a New User-Defined Function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. MySQL source distributions include a file `sql/udf_example.cc` that defines five UDF functions. Consult this file to see how UDF calling conventions work. The `include/mysql_com.h` header file defines UDF-related symbols and data structures, although you need not include this header file directly; it is included by `mysql.h`.

A UDF contains code that becomes part of the running server, so when you write a UDF, you are bound by any and all constraints that apply to writing server code. For example, you may have problems if you attempt to use functions from the `libstdc++` library. These constraints may change in future versions of the server, so it is possible that server upgrades will require revisions to UDFs that were originally written for older servers. For information about these constraints, see Section 2.9.7, “MySQL Source-Configuration Options”, and Section 2.9.8, “Dealing with Problems Compiling MySQL”.

To be able to use UDFs, you must link `mysqld` dynamically. If you want to use a UDF that needs to access symbols from `mysqld` (for example, the `metaphone` function in `sql/udf_example.cc` uses `default_charset_info`), you must link the program with `-rdynamic` (see `man dlopen`).

For each function that you want to use in SQL statements, you should define corresponding C (or C++) functions. In the following discussion, the name “xxx” is used for an example function name. To distinguish between SQL and C/C++ usage, `XXX()` (uppercase) indicates an SQL function call, and `xxx()` (lowercase) indicates a C/C++ function call.

Note

When using C++ you can encapsulate your C functions within:

```
extern "C" { ... }
```

This ensures that your C++ function names remain readable in the completed UDF.

The following list describes the C/C++ functions that you write to implement the interface for a function named `XXX()`. The main function, `xxx()`, is required. In addition, a UDF requires at least one of the other functions described here, for reasons discussed in Section 29.4.2.6, “UDF Security Precautions”.

- `xxx()`

  The main function. This is where the function result is computed. The correspondence between the SQL function data type and the return type of your C/C++ function is shown here.

SQL Type	C/C++ Type
`STRING`	`char *`
`INTEGER`	`long long`
`REAL`	`double`

  It is also possible to declare a `DECIMAL` function, but currently the value is returned as a string, so you should write the UDF as though it were a `STRING` function. `ROW` functions are not implemented.

- `xxx_init()`

  The initialization function for `xxx()`. If present, it can be used for the following purposes:

  - To check the number of arguments to `XXX()`.
  - To verify that the arguments are of a required type or, alternatively, to tell MySQL to coerce arguments to the required types when the main function is called.
  - To allocate any memory required by the main function.
  - To specify the maximum length of the result.
  - To specify (for `REAL` functions) the maximum number of decimal places in the result.
  - To specify whether the result can be `NULL`.

- `xxx_deinit()`

  The deinitialization function for `xxx()`. If present, it should deallocate any memory allocated by the initialization function.

When an SQL statement invokes `XXX()`, MySQL calls the initialization function `xxx_init()` to let it perform any required setup, such as argument checking or memory allocation. If `xxx_init()` returns an error, MySQL aborts the SQL statement with an error message and does not call the main or deinitialization functions. Otherwise, MySQL calls the main function `xxx()` once for each row. After all rows have been processed, MySQL calls the deinitialization function `xxx_deinit()` so that it can perform any required cleanup.

For aggregate functions that work like `SUM()`, you must also provide the following functions:

- `xxx_clear()`

  Reset the current aggregate value but do not insert the argument as the initial aggregate value for a new group.

- `xxx_add()`

  Add the argument to the current aggregate value.

MySQL handles aggregate UDFs as follows:

1. Call `xxx_init()` to let the aggregate function allocate any memory it needs for storing results.
2. Sort the table according to the `GROUP BY` expression.
3. Call `xxx_clear()` for the first row in each new group.
4. Call `xxx_add()` for each row that belongs in the same group.
5. Call `xxx()` to get the result for the aggregate when the group changes or after the last row has been processed.
6. Repeat steps 3 to 5 until all rows has been processed
7. Call `xxx_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe. This includes not just the main function, but the initialization and deinitialization functions as well, and also the additional functions required by aggregate functions. A consequence of this requirement is that you are not permitted to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

29.4.2.1 UDF Calling Sequences for Simple Functions

This section describes the different functions that you need to define when you create a simple UDF. Section 29.4.2, “Adding a New User-Defined Function”, describes the order in which MySQL calls these functions.

The main `xxx()` function should be declared as shown in this section. Note that the return type and parameters differ, depending on whether you declare the SQL function `XXX()` to return `STRING`, `INTEGER`, or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

`DECIMAL` functions return string values and should be declared the same way as `STRING` functions. `ROW` functions are not implemented.

The initialization and deinitialization functions are declared like this:

```
bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

The `initid` parameter is passed to all three functions. It points to a `UDF_INIT` structure that is used to communicate information between functions. The `UDF_INIT` structure members follow. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

- `bool maybe_null`

  `xxx_init()` should set `maybe_null` to `1` if `xxx()` can return `NULL`. The default value is `1` if any of the arguments are declared `maybe_null`.

- `unsigned int decimals`

  The number of decimal digits to the right of the decimal point. The default value is the maximum number of decimal digits in the arguments passed to the main function. For example, if the function is passed `1.34`, `1.345`, and `1.3`, the default would be 3, because `1.345` has 3 decimal digits.

  For arguments that have no fixed number of decimals, the `decimals` value is set to 31, which is 1 more than the maximum number of decimals permitted for the `DECIMAL`, `FLOAT`, and `DOUBLE` data types. This value is available as the constant `NOT_FIXED_DEC` in the `mysql_com.h` header file.

  A `decimals` value of 31 is used for arguments in cases such as a `FLOAT` or `DOUBLE` column declared without an explicit number of decimals (for example, `FLOAT` rather than `FLOAT(10,3)`) and for floating-point constants such as `1345E-3`. It is also used for string and other nonnumber arguments that might be converted within the function to numeric form.

  The value to which the `decimals` member is initialized is only a default. It can be changed within the function to reflect the actual calculation performed. The default is determined such that the largest number of decimals of the arguments is used. If the number of decimals is `NOT_FIXED_DEC` for even one of the arguments, that is the value used for `decimals`.

- `unsigned int max_length`

  The maximum length of the result. The default `max_length` value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimal digits indicated by `initid->decimals`. (For numeric functions, the length includes any sign or decimal point characters.)

  If you want to return a blob value, you can set `max_length` to 65KB or 16MB. This memory is not allocated, but the value is used to decide which data type to use if there is a need to temporarily store the data.

- `char *ptr`

  A pointer that the function can use for its own purposes. For example, functions can use `initid->ptr` to communicate allocated memory among themselves. `xxx_init()` should allocate the memory and assign it to this pointer:

  ```
  initid->ptr = allocated_memory;
  ```

In `xxx()` and `xxx_deinit()`, refer to `initid->ptr` to use or deallocate the memory.

- `bool const_item`

  `xxx_init()` should set `const_item` to `1` if `xxx()` always returns the same value and to `0` otherwise.

29.4.2.2 UDF Calling Sequences for Aggregate Functions

This section describes the different functions that you need to define when you create an aggregate UDF. Section 29.4.2, “Adding a New User-Defined Function”, describes the order in which MySQL calls these functions.

- `xxx_reset()`

  This function is called when MySQL finds the first row in a new group. It should reset any internal summary variables and then use the given `UDF_ARGS` argument as the first value in your internal summary value for the group. Declare `xxx_reset()` as follows:

```
void xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
               char *is_null, char *error);
```

  `xxx_reset()` is not needed or used in MySQL 8.0, in which the UDF interface uses `xxx_clear()` instead. However, you can define both `xxx_reset()` and `xxx_clear()` if you want to have your UDF work with older versions of the server. (If you do include both functions, the `xxx_reset()` function in many cases can be implemented internally by calling `xxx_clear()` to reset all variables, and then calling `xxx_add()` to add the `UDF_ARGS` argument as the first value in the group.)

- `xxx_clear()`

  This function is called when MySQL needs to reset the summary results. It is called at the beginning for each new group but can also be called to reset the values for a query where there were no matching rows. Declare `xxx_clear()` as follows:

```
void xxx_clear(UDF_INIT *initid, char *is_null, char *error);
```

  `is_null` is set to point to `CHAR(0)` before calling `xxx_clear()`.

  If something went wrong, you can store a value in the variable to which the `error` argument points. `error` points to a single-byte variable, not to a string buffer.

  `xxx_clear()` is required by MySQL 8.0.

- `xxx_add()`

  This function is called for all rows that belong to the same group. You should use it to add the value in the `UDF_ARGS` argument to your internal summary variable.

```
void xxx_add(UDF_INIT *initid, UDF_ARGS *args,
             char *is_null, char *error);
```

The `xxx()` function for an aggregate UDF should be declared the same way as for a nonaggregate UDF. See Section 29.4.2.1, “UDF Calling Sequences for Simple Functions”.

For an aggregate UDF, MySQL calls the `xxx()` function after all rows in the group have been processed. You should normally never access its `UDF_ARGS` argument here but instead return a value based on your internal summary variables.

Return value handling in `xxx()` should be done the same way as for a nonaggregate UDF. See Section 29.4.2.4, “UDF Return Values and Error Handling”.

The `xxx_reset()` and `xxx_add()` functions handle their `UDF_ARGS` argument the same way as functions for nonaggregate UDFs. See Section 29.4.2.3, “UDF Argument Processing”.

The pointer arguments to `is_null` and `error` are the same for all calls to `xxx_reset()`, `xxx_clear()`, `xxx_add()` and `xxx()`. You can use this to remember that you got an error or whether the `xxx()` function should return `NULL`. You should not store a string into `*error`! `error` points to a single-byte variable, not to a string buffer.

`*is_null` is reset for each group (before calling `xxx_clear()`). `*error` is never reset.

If `*is_null` or `*error` are set when `xxx()` returns, MySQL returns `NULL` as the result for the group function.

29.4.2.3 UDF Argument Processing

The `args` parameter points to a `UDF_ARGS` structure that has the members listed here:

- `unsigned int arg_count`

  The number of arguments. Check this value in the initialization function if you require your function to be called with a particular number of arguments. For example:

```
if (args->arg_count != 2)
{
    strcpy(message,"XXX() requires two arguments");
    return 1;
}
```

  For other `UDF_ARGS` member values that are arrays, array references are zero-based. That is, refer to array members using index values from 0 to `args->arg_count` − 1.

- `enum Item_result *arg_type`

  A pointer to an array containing the types for each argument. The possible type values are `STRING_RESULT`, `INT_RESULT`, `REAL_RESULT`, and `DECIMAL_RESULT`.

  To make sure that arguments are of a given type and return an error if they are not, check the `arg_type` array in the initialization function. For example:

```
if (args->arg_type[0] != STRING_RESULT ||
    args->arg_type[1] != INT_RESULT)
{
    strcpy(message,"XXX() requires a string and an integer");
    return 1;
}
```

  Arguments of type `DECIMAL_RESULT` are passed as strings, so you should handle them the same way as `STRING_RESULT` values.

  As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the `arg_type` elements to the types you want. This causes MySQL to coerce arguments to those types for each call to `xxx()`. For example, to specify that the first two arguments should be coerced to string and integer, respectively, do this in `xxx_init()`:

```
args->arg_type[0] = STRING_RESULT;
```

```
args->arg_type[1] = INT_RESULT;
```

Exact-value decimal arguments such as `1.3` or `DECIMAL` column values are passed with a type of `DECIMAL_RESULT`. However, the values are passed as strings. If you want to receive a number, use the initialization function to specify that the argument should be coerced to a `REAL_RESULT` value:

```
args->arg_type[2] = REAL_RESULT;
```

- `char **args`

  `args->args` communicates information to the initialization function about the general nature of the arguments passed to your function. For a constant argument `i`, `args->args[i]` points to the argument value. (See later for instructions on how to access the value properly.) For a nonconstant argument, `args->args[i]` is `0`. A constant argument is an expression that uses only constants, such as `3` or `4*7-2` or `SIN(3.14)`. A nonconstant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with nonconstant arguments.

  For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed.

  If argument `i` represents `NULL`, `args->args[i]` is a null pointer (0). If the argument is not `NULL`, functions can refer to it as follows:

  - An argument of type `STRING_RESULT` is given as a string pointer plus a length, to enable handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. Do not assume that the string is null-terminated.
  - For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a `long long` value:

    ```
    long long int_val;
    int_val = *((long long*) args->args[i]);
    ```

  - For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a `double` value:

    ```
    double    real_val;
    real_val = *((double*) args->args[i]);
    ```

  - For an argument of type `DECIMAL_RESULT`, the value is passed as a string and should be handled like a `STRING_RESULT` value.
  - `ROW_RESULT` arguments are not implemented.

- `unsigned long *lengths`

  For the initialization function, the `lengths` array indicates the maximum string length for each argument. You should not change these. For each invocation of the main function, `lengths` contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialization function).

- `char *maybe_null`

  For the initialization function, the `maybe_null` array indicates for each argument whether the argument value might be null (0 if no, 1 if yes).

- `char **attributes`

`args->attributes` communicates information about the names of the UDF arguments. For argument `i`, the attribute name is available as a string in `args->attributes[i]` and the attribute length is `args->attribute_lengths[i]`. Do not assume that the string is null-terminated.

By default, the name of a UDF argument is the text of the expression used to specify the argument. For UDFs, an argument may also have an optional `[AS] alias_name` clause, in which case the argument name is `alias_name`. The `attributes` value for each argument thus depends on whether an alias was given.

Suppose that a UDF `my_udf()` is invoked as follows:

```
SELECT my_udf(expr1, expr2 AS alias1, expr3 alias2);
```

In this case, the `attributes` and `attribute_lengths` arrays will have these values:

```
args->attributes[0] = "expr1"
args->attribute_lengths[0] = 5

args->attributes[1] = "alias1"
args->attribute_lengths[1] = 6

args->attributes[2] = "alias2"
args->attribute_lengths[2] = 6
```

- `unsigned long *attribute_lengths`

  The `attribute_lengths` array indicates the length of each argument name.

29.4.2.4 UDF Return Values and Error Handling

The initialization function should return `0` if no error occurred and `1` otherwise. If an error occurs, `xxx_init()` should store a null-terminated error message in the `message` parameter. The message is returned to the client. The message buffer is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long long` and `double` functions. A string function should return a pointer to the result and set `*length` to the length (in bytes) of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

MySQL passes a buffer to the `xxx()` function using the `result` parameter. This buffer is sufficiently long to hold 255 characters, which can be multibyte characters. The `xxx()` function can store the result in this buffer if it fits, in which case the return value should be a pointer to the buffer. If the function stores the result in a different buffer, it should return a pointer to that buffer.

If your string function does not use the supplied buffer (for example, if it needs to return a string longer than 255 characters), you must allocate the space for your own buffer with `malloc()` in your `xxx_init()` function or your `xxx()` function and free it in your `xxx_deinit()` function. You can store the allocated memory in the `ptr` slot in the `UDF_INIT` structure for reuse by future `xxx()` calls. See Section 29.4.2.1, “UDF Calling Sequences for Simple Functions”.

To indicate a return value of `NULL` in the main function, set `*is_null` to `1`:

```
*is_null = 1;
```

To indicate an error return in the main function, set `*error` to `1`:

```
*error = 1;
```

If `xxx()` sets `*error` to `1` for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` is not even called for subsequent rows.)

29.4.2.5 UDF Compiling and Installing

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described here for the example UDF file `sql/udf_example.cc` that is included in MySQL source distributions. For additional information about UDF installation, see Section 5.7.1, “Installing and Uninstalling User-Defined Functions”.

If a UDF will be referred to in statements that will be replicated to slave servers, you must ensure that every slave also has the function available. Otherwise, replication fails on the slaves when they attempt to invoke the function.

The immediately following instructions are for Unix. Instructions for Windows are given later in this section.

The `udf_example.cc` file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it is more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `sequence([const int])` returns a sequence starting from the given number or 1 if no number has been given.
- `lookup()` returns the IP address for a host name.
- `reverse_lookup()` returns the host name for an IP address. The function may be called either with a single string argument of the form `'xxx.xxx.xxx.xxx'` or with four numbers.
- `avgcost()` returns an average cost. This is an aggregate function.

A dynamically loadable file should be compiled as a sharable library file, using a command something like this:

```
gcc -shared -o udf_example.so udf_example.cc
```

If you are using `gcc` with `CMake` (which is how MySQL itself is configured), you should be able to create `udf_example.so` with a simpler command:

```
make udf_example
```

After compiling a shared object containing UDFs, you must install it and tell MySQL about it. Compiling a shared object from `udf_example.cc` using `gcc` directly produces a file named `udf_example.so`. Copy the shared object to the server's plugin directory and name it `udf_example.so`. This directory is given by the value of the `plugin_dir` system variable.

On some systems, the `ldconfig` program that configures the dynamic linker does not recognize a shared object unless its name begins with `lib`. In this case you should rename a file such as `udf_example.so` to `libudf_example.so`.

On Windows, compile user-defined functions using the following procedure:

1. Obtain a MySQL source distribution. See Section 2.1.2, “How to Get MySQL”.
2. Obtain the `CMake` build utility, if necessary, from http://www.cmake.org. (Version 2.6 or later is required).
3. In the source tree, look in the `sql` directory for files named `udf_example.def` and `udf_example.cc`. Copy both files from this directory to your working directory.
4. Create a `CMake` `makefile` (`CMakeLists.txt`) with these contents:

```
PROJECT(udf_example)

# Path for MySQL include directory
INCLUDE_DIRECTORIES("c:/mysql/include")

ADD_DEFINITIONS("-DHAVE_DLOPEN")
ADD_LIBRARY(udf_example MODULE udf_example.cc udf_example.def)
TARGET_LINK_LIBRARIES(udf_example wsock32)
```

5. Create the VC project and solution files, substituting an appropriate *`generator`* value:

```
cmake -G "generator"
```

   Invoking `cmake --help` shows you a list of valid generators.

6. Create `udf_example.dll`:

```
devenv udf_example.sln /build Release
```

On all platforms, after the shared library file has been copied to the `plugin_dir` directory, notify `mysqld` about the new functions with the following statements. If library files have a suffix different from `.so` on your system, substitute the correct suffix throughout (for example, `.dll` on Windows).

```
CREATE FUNCTION metaphon RETURNS STRING
  SONAME 'udf_example.so';
CREATE FUNCTION myfunc_double RETURNS REAL
  SONAME 'udf_example.so';
CREATE FUNCTION myfunc_int RETURNS INTEGER
  SONAME 'udf_example.so';
CREATE FUNCTION sequence RETURNS INTEGER
  SONAME 'udf_example.so';
CREATE FUNCTION lookup RETURNS STRING
  SONAME 'udf_example.so';
CREATE FUNCTION reverse_lookup RETURNS STRING
  SONAME 'udf_example.so';
CREATE AGGREGATE FUNCTION avgcost RETURNS REAL
  SONAME 'udf_example.so';
```

Once installed, a function remains installed until it is uninstalled.

To remove functions, use `DROP FUNCTION`:

```
DROP FUNCTION metaphon;
```

```
DROP FUNCTION myfunc_double;
DROP FUNCTION myfunc_int;
DROP FUNCTION sequence;
DROP FUNCTION lookup;
DROP FUNCTION reverse_lookup;
DROP FUNCTION avgcost;
```

The `CREATE FUNCTION` and `DROP FUNCTION` statements update the `func` table in the `mysql` system database that serves as a UDF registry. The function's name, type and shared library name are saved in the `mysql.func` table. To create functions, you must have the `INSERT` privilege for the `mysql` database. To drop functions, you must have the `DELETE` privilege for the `mysql` database.

You cannnot use `CREATE FUNCTION` to reinstall a function that has previously been installed. To reinstall a function, first remove it with `DROP FUNCTION`, then install it with `CREATE FUNCTION`. You would need to do this, for example, if you upgrade to a new version of MySQL that provides an updated implementation of the function, or you recompile a new version of a function that you have written. Otherwise, the server continues to use the old version.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, the server does not load UDFs during startup and UDFs are unavailable.

29.4.2.6 UDF Security Precautions

MySQL takes several measures to prevent misuse of user-defined functions.

UDF library files cannot be placed in arbitrary directories. They must be located in the server's plugin directory. This directory is given by the value of the `plugin_dir` system variable.

To use `CREATE FUNCTION` or `DROP FUNCTION`, you must have the `INSERT` or `DELETE` privilege, respectively, for the `mysql` database. This is necessary because those statements add and delete rows from the `mysql.func` table.

UDFs should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. `mysqld` also supports an `--allow-suspicious-udfs` option that controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off, to prevent attempts at loading functions from shared library files other than those containing legitimate UDFs. If you have older UDFs that contain only the `xxx` symbol and that cannot be recompiled to include an auxiliary symbol, it may be necessary to specify the `--allow-suspicious-udfs` option. Otherwise, you should avoid enabling this capability.

29.4.3 Adding a New Native Function

To add a new native MySQL function, use the procedure described here, which requires that you use a source distribution. You cannot add native functions to a binary distribution because it is necessary to modify MySQL source code and compile MySQL from the modified source. If you migrate to another version of MySQL (for example, when a new version is released), you must repeat the procedure with the new version.

If the new native function will be referred to in statements that will be replicated to slave servers, you must ensure that every slave server also has the function available. Otherwise, replication will fail on the slaves when they attempt to invoke the function.

To add a new native function, follow these steps to modify source files in the `sql` directory:

1. Create a subclass for the function in `item_create.cc`:

- If the function takes a fixed number of arguments, create a subclass of `Create_func_arg0`, `Create_func_arg1`, `Create_func_arg2`, or `Create_func_arg3`, respectively, depending on whether the function takes zero, one, two, or three arguments. For examples, see the `Create_func_uuid`, `Create_func_abs`, `Create_func_pow`, and `Create_func_lpad` classes.
- If the function takes a variable number of arguments, create a subclass of `Create_native_func`. For an example, see `Create_func_concat`.

2. To provide a name by which the function can be referred to in SQL statements, register the name in `item_create.cc` by adding a line to this array:

```
static Native_func_registry func_array[]
```

   You can register several names for the same function. For example, see the lines for `"LCASE"` and `"LOWER"`, which are aliases for `Create_func_lcase`.

3. In `item_func.h`, declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.

4. In `item_func.cc`, add one of the following declarations, depending on whether you are defining a numeric or string function:

```
double   Item_func_newname::val()
longlong Item_func_newname::val_int()
String  *Item_func_newname::Str(String *str)
```

   If you inherit your object from any of the standard items (like `Item_num_func`), you probably only have to define one of these functions and let the parent object take care of the other functions. For example, the `Item_str_func` class defines a `val()` function that executes `atof()` on the value returned by `::str()`.

5. If the function is nondeterministic, include the following statement in the item constructor to indicate that function results should not be cached:

```
current_thd->lex->safe_to_cache_query=0;
```

   A function is nondeterministic if, given fixed values for its arguments, it can return different results for different invocations.

6. You should probably also define the following object function:

```
void Item_func_newname::fix_length_and_dec()
```

   This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function cannot return a `NULL` value. The function can check whether any of the function arguments can return `NULL` by checking the arguments' `maybe_null` variable. Look at `Item_func_mod::fix_length_and_dec` for a typical example of how to do this.

All functions must be thread-safe. In other words, do not use any global or static variables in the functions without protecting them with mutexes.

If you want to return `NULL` from `::val()`, `::val_int()`, or `::str()`, you should set `null_value` to 1 and return 0.

For `::str()` object functions, there are additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result. (For more information about the `String` type, take a look at the `sql_string.h` file.)
- The `::str()` function should return the string that holds the result, or `(char*) 0` if the result is `NULL`.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

29.5 Debugging and Porting MySQL

This section helps you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See https://www.mysql.com/support/supportedplatforms/database.html. If you have created a new port of MySQL, please let us know so that we can list it here and on our website (http://www.mysql.com/), recommending it to other users.

Note

If you create a new port of MySQL, you are free to copy and distribute it under the GPL license, but it does not make you a copyright holder of MySQL.

A working POSIX thread library is needed for the server.

To build MySQL from source, your system must satisfy the tool requirements listed at Section 2.9, “Installing MySQL from Source”.

If you run into problems with a new port, you may have to do some debugging of MySQL! See Section 29.5.1, “Debugging a MySQL Server”.

Note

Before you start debugging `mysqld`, first get the test program `mysys/thr_lock` to work. This ensures that your thread installation has even a remote chance to work!

Note

The MySQL source code contains internal documentation written using Doxygen. This documentation is useful for understanding how MySQL works from a developer perspective. The generated Doxygen content is available at https://dev.mysql.com/doc/index-other.html. It is also possible to generate this content locally from a MySQL source distribution using the instructions at Section 2.9.10, “Generating MySQL Doxygen Documentation Content”.

29.5.1 Debugging a MySQL Server

If you are using some functionality that is very new in MySQL, you can try to run `mysqld` with the `--skip-new` (which disables all new, potentially unsafe functionality). See Section B.4.3.3, “What to Do If MySQL Keeps Crashing”.

If `mysqld` does not want to start, verify that you have no `my.cnf` files that interfere with your setup! You can check your `my.cnf` arguments with `mysqld --print-defaults` and avoid using them by starting with `mysqld --no-defaults ...`.

If `mysqld` starts to eat up CPU or memory or if it “hangs,” you can use `mysqladmin processlist status` to find out if someone is executing a query that takes a long time. It may be a good idea to run `mysqladmin -i10 processlist status` in some window if you are experiencing performance problems or problems when new clients cannot connect.

The command `mysqladmin debug` dumps some information about locks in use, used memory and query usage to the MySQL log file. This may help solve some problems. This command also provides some useful information even if you have not compiled MySQL for debugging!

If the problem is that some tables are getting slower and slower you should try to optimize the table with `OPTIMIZE TABLE` or `myisamchk`. See Chapter 5, *MySQL Server Administration*. You should also check the slow queries with `EXPLAIN`.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See Section 2.1, “General Installation Guidance”.

29.5.1.1 Compiling MySQL for Debugging

If you have some very specific problem, you can always try to debug MySQL. To do this you must configure MySQL with the `-DWITH_DEBUG=1` option. You can check whether MySQL was compiled with debugging by doing: `mysqld --help`. If the `--debug` flag is listed with the options then you have debugging enabled. `mysqladmin ver` also lists the `mysqld` version as `mysql ... --debug` in this case.

If `mysqld` stops crashing when you configure it with the `-DWITH_DEBUG=1` CMake option, you probably have found a compiler bug or a timing bug within MySQL. In this case, you can try to add `-g` using the `CMAKE_C_FLAGS` and `CMAKE_CXX_FLAGS` CMake options and not use `-DWITH_DEBUG=1`. If `mysqld` dies, you can at least attach to it with `gdb` or use `gdb` on the core file to find out what happened.

When you configure MySQL for debugging you automatically enable a lot of extra safety check functions that monitor the health of `mysqld`. If they find something “unexpected,” an entry is written to `stderr`, which `mysqld_safe` directs to the error log! This also means that if you are having some unexpected problems with MySQL and are using a source distribution, the first thing you should do is to configure MySQL for debugging. If you believe that you have found a bug, please use the instructions at Section 1.7, “How to Report Bugs or Problems”.

In the Windows MySQL distribution, `mysqld.exe` is by default compiled with support for trace files.

29.5.1.2 Creating Trace Files

If the `mysqld` server does not start or it crashes easily, you can try to create a trace file to find the problem.

To do this, you must have a `mysqld` that has been compiled with debugging support. You can check this by executing `mysqld -V`. If the version number ends with `-debug`, it is compiled with support for trace files. (On Windows, the debugging server is named `mysqld-debug` rather than `mysqld`.)

Start the `mysqld` server with a trace log in `/tmp/mysqld.trace` on Unix or `\mysqld.trace` on Windows:

```
shell> mysqld --debug
```

On Windows, you should also use the `--standalone` flag to not start `mysqld` as a service. In a console window, use this command:

```
C:\> mysqld-debug --debug --standalone
```

After this, you can use the `mysql.exe` command-line tool in a second console window to reproduce the problem. You can stop the `mysqld` server with `mysqladmin shutdown`.

The trace file can become **very large**! To generate a smaller trace file, you can use debugging options something like this:

```
mysqld --debug=d,info,error,query,general,where:O,/tmp/mysqld.trace
```

This only prints information with the most interesting tags to the trace file.

If you file a bug, please add only those lines from the trace file to the bug report that indicate where something seems to go wrong. If you cannot locate the wrong place, open a bug report and upload the whole trace file to the report, so that a MySQL developer can take a look at it. For instructions, see Section 1.7, “How to Report Bugs or Problems”.

The trace file is made with the **DBUG** package by Fred Fish. See Section 29.5.4, “The DBUG Package”.

29.5.1.3 Using WER with PDB to create a Windows crashdump

Program Database files (with suffix `pdb`) are included in the **ZIP Archive Debug Binaries & Test Suite** distribution of MySQL. These files provide information for debugging your MySQL installation in the event of a problem. This is a separate download from the standard MSI or Zip file.

Note

The PDB files are available in a separate file labeled "ZIP Archive Debug Binaries & Test Suite".

The PDB file contains more detailed information about `mysqld` and other tools that enables more detailed trace and dump files to be created. You can use these with `WinDbg` or Visual Studio to debug `mysqld`.

For more information on PDB files, see Microsoft Knowledge Base Article 121366. For more information on the debugging options available, see Debugging Tools for Windows.

To use WinDbg, either install the full Windows Driver Kit (WDK) or install the standalone version.

Important

The `.exe` and `.pdb` files must be an exact match (both version number and MySQL server edition) or WinDBG will complain while attempting to load the symbols.

1. To generate a minidump `mysqld.dmp`, enable the `core-file` option under the [mysqld] section in `my.ini`. Restart the MySQL server after making these changes.
2. Create a directory to store the generated files, such as `c:\symbols`
3. Determine the path to your `windbg.exe` executable using the Find GUI or from the command line, for example: `dir /s /b windbg.exe` -- a common default is *C:\Program Files\Debugging Tools for Windows (x64)\windbg.exe*
4. Launch `windbg.exe` giving it the paths to `mysqld.exe`, `mysqld.pdb`, `mysqld.dmp`, and the source code. Alternatively, pass in each path from the WinDbg GUI. For example:

```
windbg.exe -i "C:\mysql-8.0.19-winx64\bin\"^
 -z "C:\mysql-8.0.19-winx64\data\mysqld.dmp"^
 -srcpath "E:\ade\mysql_archives\8.0\8.0.19\mysql-8.0.19"^
 -y "C:\mysql-8.0.19-winx64\bin;SRV*c:\symbols*http://msdl.microsoft.com/download/symbols"^
 -v -n -c "!analyze -vvvvv"
```

Note

The `^` character and newline are removed by the Windows command line processor, so be sure the spaces remain intact.

29.5.1.4 Debugging mysqld under gdb

On most systems you can also start `mysqld` from `gdb` to get more information if `mysqld` crashes.

With some older `gdb` versions on Linux you must use `run --one-thread` if you want to be able to debug `mysqld` threads. In this case, you can only have one thread active at a time. It is best to upgrade to `gdb` 5.1 because thread debugging works much better with this version!

NPTL threads (the new thread library on Linux) may cause problems while running `mysqld` under `gdb`. Some symptoms are:

- `mysqld` hangs during startup (before it writes `ready for connections`).
- `mysqld` crashes during a `pthread_mutex_lock()` or `pthread_mutex_unlock()` call.

In this case, you should set the following environment variable in the shell before starting `gdb`:

```
LD_ASSUME_KERNEL=2.4.1
export LD_ASSUME_KERNEL
```

When running `mysqld` under `gdb`, you should disable the stack trace with `--skip-stack-trace` to be able to catch segfaults within `gdb`.

Use the `--gdb` option to `mysqld` to install an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling.

It is very hard to debug MySQL under `gdb` if you do a lot of new connections the whole time as `gdb` does not free the memory for old threads. You can avoid this problem by starting `mysqld` with `thread_cache_size` set to a value equal to `max_connections` + 1. In most cases just using `--thread_cache_size=5'` helps a lot!

If you want to get a core dump on Linux if `mysqld` dies with a SIGSEGV signal, you can start `mysqld` with the `--core-file` option. This core file can be used to make a backtrace that may help you find out why `mysqld` died:

```
shell> gdb mysqld core
gdb>   backtrace full
gdb>   quit
```

See Section B.4.3.3, “What to Do If MySQL Keeps Crashing”.

If you are using `gdb` 4.17.x or above on Linux, you should install a `.gdb` file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

If you have problems debugging threads with `gdb`, you should download gdb 5.x and try this instead. The new `gdb` version has very improved thread handling!

Here is an example how to debug `mysqld`:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
```

```
...
backtrace full # Do this when mysqld crashes
```

Include the preceding output in a bug report, which you can file using the instructions in Section 1.7, “How to Report Bugs or Problems”.

If `mysqld` hangs, you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where `mysqld` has hung.

```
strace /tmp/log libexec/mysqld
```

If you are using the Perl `DBI` interface, you can turn on debugging information by using the `trace` method or by setting the `DBI_TRACE` environment variable.

29.5.1.5 Using a Stack Trace

On some operating systems, the error log contains a stack trace if `mysqld` dies unexpectedly. You can use this to find out where (and maybe why) `mysqld` died. See Section 5.4.2, “The Error Log”. To get a stack trace, you must not compile `mysqld` with the `-fomit-frame-pointer` option to gcc. See Section 29.5.1.1, “Compiling MySQL for Debugging”.

A stack trace in the error log looks something like this:

```
mysqld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqld died. If you see no messages after
this, something went terribly wrong...

stack_bottom = 0x41fd0110 thread_stack 0x40000
mysqld(my_print_stacktrace+0x32)[0x9da402]
mysqld(handle_segfault+0x28a)[0x6648e9]
/lib/libpthread.so.0[0x7f1a5af000f0]
/lib/libc.so.6(strcmp+0x2)[0x7f1a5a10f0f2]
mysqld(_Z21check_change_passwordP3THDPKcS2_Pcj+0x7c)[0x7412cb]
mysqld(_ZN16set_var_password5checkEP3THD+0xd0)[0x688354]
mysqld(_Z17sql_set_variablesP3THDP4ListI12set_var_baseE+0x68)[0x688494]
mysqld(_Z21mysql_execute_commandP3THD+0x41a0)[0x67a170]
mysqld(_Z11mysql_parseP3THDPKcjPS2_+0x282)[0x67f0ad]
mysqld(_Z16dispatch_command19enum_server_commandP3THDPcj+0xbb7[0x67fdf8]
mysqld(_Z10do_commandP3THD+0x24d)[0x6811b6]
mysqld(handle_one_connection+0x11c)[0x66e05e]
```

If resolution of function names for the trace fails, the trace contains less information:

```
mysqld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqld died. If you see no messages after
this, something went terribly wrong...

stack_bottom = 0x41fd0110 thread_stack 0x40000
[0x9da402]
[0x6648e9]
[0x7f1a5af000f0]
[0x7f1a5a10f0f2]
[0x7412cb]
[0x688354]
[0x688494]
[0x67a170]
[0x67f0ad]
[0x67fdf8]
[0x6811b6]
```

```
[0x66e05e]
```

Newer versions of `glibc` stack trace functions also print the address as relative to the object. On `glibc`-based systems (Linux), the trace for a crash within a plugin looks something like:

```
plugin/auth/auth_test_plugin.so(+0x9a6)[0x7ff4d11c29a6]
```

To translate the relative address (`+0x9a6`) into a file name and line number, use this command:

```
shell> addr2line -fie auth_test_plugin.so 0x9a6
auth_test_plugin
mysql-trunk/plugin/auth/test_plugin.c:65
```

The `addr2line` utility is part of the `binutils` package on Linux.

On Solaris, the procedure is similar. The Solaris `printstack()` already prints relative addresses:

```
plugin/auth/auth_test_plugin.so:0x1510
```

To translate, use this command:

```
shell> gaddr2line -fie auth_test_plugin.so 0x1510
mysql-trunk/plugin/auth/test_plugin.c:88
```

Windows already prints the address, function name and line:

```
000007FEF07E10A4 auth_test_plugin.dll!auth_test_plugin()[test_plugin.c:72]
```

29.5.1.6 Using Server Logs to Find Causes of Errors in mysqld

Note that before starting `mysqld` with the general query log enabled, you should check all your tables with `myisamchk`. See Chapter 5, *MySQL Server Administration*.

If `mysqld` dies or hangs, you should start `mysqld` with the general query log enabled. See Section 5.4.3, “The General Query Log”. When `mysqld` dies again, you can examine the end of the log file for the query that killed `mysqld`.

If you use the default general query log file, the log is stored in the database directory as *`host_name`*`.log` In most cases it is the last query in the log file that killed `mysqld`, but if possible you should verify this by restarting `mysqld` and executing the found query from the `mysql` command-line tools. If this works, you should also test all complicated queries that did not complete.

You can also try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that `mysqld` is using indexes properly. See Section 13.8.2, “EXPLAIN Syntax”.

You can find the queries that take a long time to execute by starting `mysqld` with the slow query log enabled. See Section 5.4.5, “The Slow Query Log”.

If you find the text `mysqld restarted` in the error log (normally a file named *`host_name`*`.err`) you probably have found a query that causes `mysqld` to fail. If this happens, you should check all your tables with `myisamchk` (see Chapter 5, *MySQL Server Administration*), and test the queries in the MySQL log files to see whether one fails. If you find such a query, try first upgrading to the newest MySQL version. If this does not help, report a bug, see Section 1.7, “How to Report Bugs or Problems”.

If you have started `mysqld` with the `myisam_recover_options` system variable set, MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking`

`table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further. See Section 5.1.7, “Server Command Options”.

When the server detects `MyISAM` table corruption, it writes additional information to the error log, such as the name and line number of the source file, and the list of threads accessing the table. Example: `Got an error from thread_id=1, mi_dynrec.c:368`. This is useful information to include in bug reports.

It is not a good sign if `mysqld` did die unexpectedly, but in this case, you should not investigate the `Checking table...` messages, but instead try to find out why `mysqld` died.

29.5.1.7 Making a Test Case If You Experience Table Corruption

The following procedure applies to `MyISAM` tables. For information about steps to take when encountering `InnoDB` table corruption, see Section 1.7, “How to Report Bugs or Problems”.

If you encounter corrupted `MyISAM` tables or if `mysqld` always fails after some update statements, you can test whether the issue is reproducible by doing the following:

1. Stop the MySQL daemon with `mysqladmin shutdown`.
2. Make a backup of the tables to guard against the very unlikely case that the repair does something bad.
3. Check all tables with `myisamchk -s database/*.MYI`. Repair any corrupted tables with `myisamchk -r database/*table*.MYI`.
4. Make a second backup of the tables.
5. Remove (or move away) any old log files from the MySQL data directory if you need more space.
6. Start `mysqld` with the binary log enabled. If you want to find a statement that crashes `mysqld`, you should start the server with the general query log enabled as well. See Section 5.4.3, “The General Query Log”, and Section 5.4.4, “The Binary Log”.
7. When you have gotten a crashed table, stop the `mysqld` server.
8. Restore the backup.
9. Restart the `mysqld` server *without* the binary log enabled.
10. Re-execute the statements with `mysqlbinlog binary-log-file | mysql`. The binary log is saved in the MySQL database directory with the name `hostname-bin.NNNNNN`.
11. If the tables are corrupted again or you can get `mysqld` to die with the above command, you have found a reproducible bug. FTP the tables and the binary log to our bugs database using the instructions given in Section 1.7, “How to Report Bugs or Problems”. If you are a support customer, you can use the MySQL Customer Support Center (https://www.mysql.com/support/) to alert the MySQL team about the problem and have it fixed as soon as possible.

29.5.2 Debugging a MySQL Client

To be able to debug a MySQL client with the integrated debug package, you should configure MySQL with `-DWITH_DEBUG=1`. See Section 2.9.7, “MySQL Source-Configuration Options”.

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:O,/tmp/client.trace
```

```
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in `/tmp/client.trace`.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running `mysql` in debugging mode (assuming that you have compiled MySQL with debugging on):

```
shell> mysql --debug=d:t:O,/tmp/client.trace
```

This provides useful information in case you mail a bug report. See Section 1.7, “How to Report Bugs or Problems”.

If your client crashes at some 'legal' looking code, you should check that your `mysql.h` include file matches your MySQL library file. A very common mistake is to use an old `mysql.h` file from an old MySQL installation with new MySQL library.

29.5.3 The LOCK_ORDER Tool

The MySQL server is a multithreaded application that uses numerous internal locking and lock-related primitives, such as mutexes, rwlocks (including prlocks and sxlocks), conditions, and files. Within the server, the set of lock-related objects changes with implementation of new features and code refactoring for performance improvements. As with any multithreaded application that uses locking primitives, there is always a risk of encountering a deadlock during execution when multiple locks are held at once. For MySQL, the effect of a deadlock is catastrophic, causing a complete loss of service.

As of MySQL 8.0.17, to enable detection of lock-acquisition deadlocks and enforcement that runtime execution is free of them, MySQL supports LOCK_ORDER tooling. This enables a lock-order dependency graph to be defined as part of server design, and server runtime checking to ensure that lock acquisition is acyclic and that execution paths comply with the graph.

This section provides information about using the LOCK_ORDER tool, but only at a basic level. For complete details, see the Lock Order section of the MySQL Server Doxygen documentation, available at https://dev.mysql.com/doc/index-other.html.

The LOCK_ORDER tool is intended for debugging the server, not for production use.

To use the LOCK_ORDER tool, follow this procedure:

1. Build MySQL from source, configuring it with the `-DWITH_LOCK_ORDER=ON` `CMake` option so that the build includes LOCK_ORDER tooling.

   Note

   With the `WITH_LOCK_ORDER` option enabled, MySQL builds require the `flex` program.

2. To run the server with the LOCK_ORDER tool enabled, enable the `lock_order` system variable at server startup. Several other system variables for LOCK_ORDER configuration are available as well.

3. For MySQL test suite operation, `mysql-test-run.pl` has a `--lock-order` option that controls whether to enable the LOCK_ORDER tool during test case execution.

The system variables described following configure operation of the LOCK_ORDER tool, assuming that MySQL has been built to include LOCK_ORDER tooling. The primary variable is `lock_order`, which indicates whether to enable the LOCK_ORDER tool at runtime:

- If `lock_order` is disabled (the default), no other LOCK_ORDER system variables have any effect.

- If `lock_order` is enabled, the other system variables configure which LOCK_ORDER features to enable.

Note

In general, it is intended that the LOCK_ORDER tool be configured by executing `mysql-test-run.pl` with the `--lock-order` option, and for `mysql-test-run.pl` to set LOCK_ORDER system variables to appropriate values.

All LOCK_ORDER system variables must be set at server startup. At runtime, their values are visible but cannot be changed.

Some system variables exist in pairs, such as `lock_order_debug_loop` and `lock_order_trace_loop`. For such pairs, the variables are distinguished as follows when the condition occurs with which they are associated:

- If the `_debug_` variable is enabled, a debug assertion is raised.
- If the `_trace_` variable is enabled, an error is printed to the logs.

Table 29.4 LOCK_ORDER System Variable Summary

Variable Name	Variable Type	Variable Scope
lock_order	Boolean	Global
lock_order_debug_loop	Boolean	Global
lock_order_debug_missing_arc	Boolean	Global
lock_order_debug_missing_key	Boolean	Global
lock_order_debug_missing_unlock	Boolean	Global
lock_order_dependencies	File name	Global
lock_order_extra_dependencies	File name	Global
lock_order_output_directory	Directory name	Global
lock_order_print_txt	Boolean	Global
lock_order_trace_loop	Boolean	Global
lock_order_trace_missing_arc	Boolean	Global
lock_order_trace_missing_key	Boolean	Global
lock_order_trace_missing_unlock	Boolean	Global

- `lock_order`

Property	Value
Command-Line Format	`--lock-order[={OFF|ON}]`
Introduced	8.0.17
System Variable	`lock_order`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether to enable the LOCK_ORDER tool at runtime. If `lock_order` is disabled (the default), no other LOCK_ORDER system variables have any effect. If `lock_order` is enabled, the other system variables configure which LOCK_ORDER features to enable.

If `lock_order` is enabled, an error is raised if the server encounters a lock-acquisition sequence that is not declared in the lock-order graph.

- `lock_order_debug_loop`

Property	Value
Command-Line Format	`--lock-order-debug-loop[={OFF|ON}]`
Introduced	8.0.17
System Variable	`lock_order_debug_loop`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether the LOCK_ORDER tool causes a debug assertion failure when it encounters a dependency that is flagged as a loop in the lock-order graph.

- `lock_order_debug_missing_arc`

Property	Value
Command-Line Format	`--lock-order-debug-missing-arc[={OFF|ON}]`
Introduced	8.0.17
System Variable	`lock_order_debug_missing_arc`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether the LOCK_ORDER tool causes a debug assertion failure when it encounters a dependency that is not declared in the lock-order graph.

- `lock_order_debug_missing_key`

Property	Value
Command-Line Format	`--lock-order-debug-missing-key[={OFF|ON}]`
Introduced	8.0.17
System Variable	`lock_order_debug_missing_key`
Scope	Global
Dynamic	No

Property	Value
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Whether the LOCK_ORDER tool causes a debug assertion failure when it encounters an object that is not properly instrumented with the Performance Schema.

- lock_order_debug_missing_unlock

Property	Value
Command-Line Format	--lock-order-debug-missing-unlock[={OFF|ON}]
Introduced	8.0.17
System Variable	lock_order_debug_missing_unlock
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Whether the LOCK_ORDER tool causes a debug assertion failure when it encounters a lock that is destroyed while still held.

- lock_order_dependencies

Property	Value
Command-Line Format	--lock-order-dependencies=file_name
Introduced	8.0.17
System Variable	lock_order_dependencies
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	empty string

The path to the lock_order_dependencies.txt file that defines the server lock-order dependency graph.

It is permitted to specify no dependencies. An empty dependency graph is used in this case.

- lock_order_extra_dependencies

Property	Value
Command-Line Format	--lock-order-extra-dependencies=file_name
Introduced	8.0.17
System Variable	lock_order_extra_dependencies

Property	Value
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	File name
Default Value	`empty string`

The path to a file containing additional dependencies for the lock-order dependency graph. This is useful to amend the primary server dependency graph, defined in the `lock_order_dependencies.txt` file, with additional dependencies describing the behavior of third party code. (The alternative is to modify `lock_order_dependencies.txt` itself, which is not encouraged.)

If this variable is not set, no secondary file is used.

- `lock_order_output_directory`

Property	Value
Command-Line Format	`--lock-order-output-directory=dir_name`
Introduced	8.0.17
System Variable	`lock_order_output_directory`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name
Default Value	`empty string`

The directory where the LOCK_ORDER tool writes its logs. If this variable is not set, the default is the current directory.

- `lock_order_print_txt`

Property	Value
Command-Line Format	`--lock-order-print-txt[={OFF|ON}]`
Introduced	8.0.17
System Variable	`lock_order_print_txt`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether the LOCK_ORDER tool performs a lock-order graph analysis and prints a textual report. The report includes any lock-acquisition cycles detected.

- `lock_order_trace_loop`

Property	Value
Command-Line Format	`--lock-order-trace-loop[={OFF|ON}]`
Introduced	8.0.17
System Variable	`lock_order_trace_loop`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether the LOCK_ORDER tool prints a trace in the log file when it encounters a dependency that is flagged as a loop in the lock-order graph.

- `lock_order_trace_missing_arc`

Property	Value
Command-Line Format	`--lock-order-trace-missing-arc[={OFF|ON}]`
Introduced	8.0.17
System Variable	`lock_order_trace_missing_arc`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether the LOCK_ORDER tool prints a trace in the log file when it encounters a dependency that is not declared in the lock-order graph.

- `lock_order_trace_missing_key`

Property	Value
Command-Line Format	`--lock-order-trace-missing-key[={OFF|ON}]`
Introduced	8.0.17
System Variable	`lock_order_trace_missing_key`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`OFF`

Whether the LOCK_ORDER tool prints a trace in the log file when it encounters an object that is not properly instrumented with the Performance Schema.

- `lock_order_trace_missing_unlock`

Property	Value
Command-Line Format	`--lock-order-trace-missing-unlock[={OFF|ON}]`
Introduced	8.0.17
System Variable	`lock_order_trace_missing_unlock`
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	`ON`

Whether the LOCK_ORDER tool prints a trace in the log file when it encounters a lock that is destroyed while still held.

29.5.4 The DBUG Package

The MySQL server and most MySQL clients are compiled with the DBUG package originally created by Fred Fish. When you have configured MySQL for debugging, this package makes it possible to get a trace file of what the program is doing. See Section 29.5.1.2, “Creating Trace Files”.

This section summarizes the argument values that you can specify in debug options on the command line for MySQL programs that have been built with debugging support.

The DBUG package can be used by invoking a program with the `--debug[=debug_options]` or `-#[debug_options]` option. If you specify the `--debug` or `-#` option without a *`debug_options`* value, most MySQL programs use a default value. The server default is `d:t:i:o,/tmp/mysqld.trace` on Unix and `d:t:i:O,\mysqld.trace` on Windows. The effect of this default is:

- `d`: Enable output for all debug macros
- `t`: Trace function calls and exits
- `i`: Add PID to output lines
- `o,/tmp/mysqld.trace`, `O,\mysqld.trace`: Set the debug output file.

Most client programs use a default *`debug_options`* value of `d:t:o,/tmp/`*`program_name`*`.trace`, regardless of platform.

Here are some example debug control strings as they might be specified on a shell command line:

```
--debug=d:t
--debug=d:f,main,subr1:F:L:t,20
--debug=d,input,output,files:n
--debug=d:t:i:O,\\mysqld.trace
```

For `mysqld`, it is also possible to change DBUG settings at runtime by setting the `debug` system variable. This variable has global and session values:

```
mysql> SET GLOBAL debug = 'debug_options';
mysql> SET SESSION debug = 'debug_options';
```

Changing the global `debug` value requires privileges sufficient to set global system variables. Changing the session `debug` value requires privileges sufficient to set restricted session system variables. See Section 5.1.9.1, “System Variable Privileges”.

The *`debug_options`* value is a sequence of colon-separated fields:

```
field_1:field_2:...:field_N
```

Each field within the value consists of a mandatory flag character, optionally preceded by a `+` or `-` character, and optionally followed by a comma-delimited list of modifiers:

```
[+|-]flag[,modifier,modifier,...,modifier]
```

The following table describes the permitted flag characters. Unrecognized flag characters are silently ignored.

Flag	Description
`d`	Enable output from DBUG_*XXX* macros for the current state. May be followed by a list of keywords, which enables output only for the DBUG macros with that keyword. An empty list of keywords enables output for all macros. In MySQL, common debug macro keywords to enable are `enter`, `exit`, `error`, `warning`, `info`, and `loop`.
`D`	Delay after each debugger output line. The argument is the delay, in tenths of seconds, subject to machine capabilities. For example, `D,20` specifies a delay of two seconds.
`f`	Limit debugging, tracing, and profiling to the list of named functions. An empty list enables all functions. The appropriate `d` or `t` flags must still be given; this flag only limits their actions if they are enabled.
`F`	Identify the source file name for each line of debug or trace output.
`i`	Identify the process with the PID or thread ID for each line of debug or trace output.
`L`	Identify the source file line number for each line of debug or trace output.
`n`	Print the current function nesting depth for each line of debug or trace output.
`N`	Number each line of debug output.
`o`	Redirect the debugger output stream to the specified file. The default output is `stderr`.
`O`	Like `o`, but the file is really flushed between each write. When needed, the file is closed and reopened between each write.
`p`	Limit debugger actions to specified processes. A process must be identified with the `DBUG_PROCESS` macro and match one in the list for debugger actions to occur.
`P`	Print the current process name for each line of debug or trace output.
`r`	When pushing a new state, do not inherit the previous state's function nesting level. Useful when the output is to start at the left margin.
`S`	Do function `_sanity(_file_,_line_)` at each debugged function until `_sanity()` returns something that differs from 0.
`t`	Enable function call/exit trace lines. May be followed by a list (containing only one modifier) giving a numeric maximum trace level, beyond which no output occurs for either debugging or tracing macros. The default is a compile time option.

The leading `+` or `-` character and trailing list of modifiers are used for flag characters such as `d` or `f` that can enable a debug operation for all applicable modifiers or just some of them:

- With no leading + or -, the flag value is set to exactly the modifier list as given.
- With a leading + or -, the modifiers in the list are added to or subtracted from the current modifier list.

The following examples show how this works for the d flag. An empty d list enabled output for all debug macros. A nonempty list enables output only for the macro keywords in the list.

These statements set the d value to the modifier list as given:

```
mysql> SET debug = 'd';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| d       |
+---------+
mysql> SET debug = 'd,error,warning';
mysql> SELECT @@debug;
+-----------------+
| @@debug         |
+-----------------+
| d,error,warning |
+-----------------+
```

A leading + or - adds to or subtracts from the current d value:

```
mysql> SET debug = '+d,loop';
mysql> SELECT @@debug;
+----------------------+
| @@debug              |
+----------------------+
| d,error,warning,loop |
+----------------------+

mysql> SET debug = '-d,error,loop';
mysql> SELECT @@debug;
+-----------+
| @@debug   |
+-----------+
| d,warning |
+-----------+
```

Adding to “all macros enabled” results in no change:

```
mysql> SET debug = 'd';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| d       |
+---------+

mysql> SET debug = '+d,loop';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| d       |
+---------+
```

Disabling all enabled macros disables the d flag entirely:

```
mysql> SET debug = 'd,error,loop';
mysql> SELECT @@debug;
+--------------+
| @@debug      |
+--------------+
| d,error,loop |
+--------------+

mysql> SET debug = '-d,error,loop';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
|         |
+---------+
```

Chapter 30 MySQL Enterprise Edition

Table of Contents

30.1 MySQL Enterprise Monitor Overview 5051
30.2 MySQL Enterprise Backup Overview 5052
30.3 MySQL Enterprise Security Overview 5053
30.4 MySQL Enterprise Encryption Overview 5053
30.5 MySQL Enterprise Audit Overview 5053
30.6 MySQL Enterprise Firewall Overview 5054
30.7 MySQL Enterprise Thread Pool Overview 5054
30.8 MySQL Enterprise Data Masking and De-Identification Overview 5054

MySQL Enterprise Edition is a commercial product. Like MySQL Community Edition, MySQL Enterprise Edition includes MySQL Server, a fully integrated transaction-safe, ACID-compliant database with full commit, rollback, crash-recovery, and row-level locking capabilities. In addition, MySQL Enterprise Edition includes the following components designed to provide monitoring and online backup, as well as improved security and scalability:

The following sections briefly discuss each of these components and indicate where to find more detailed information. To learn more about commercial products, see https://www.mysql.com/products/.

- MySQL Enterprise Monitor
- MySQL Enterprise Backup
- MySQL Enterprise Security
- MySQL Enterprise Encryption
- MySQL Enterprise Audit
- MySQL Enterprise Firewall
- MySQL Enterprise Thread Pool
- MySQL Enterprise Data Masking and De-Identification

30.1 MySQL Enterprise Monitor Overview

MySQL Enterprise Monitor is an enterprise monitoring system for MySQL that keeps an eye on your MySQL servers, notifies you of potential issues and problems, and advises you how to fix the issues. MySQL Enterprise Monitor can monitor all kinds of configurations, from a single MySQL server that is important to your business, all the way up to a huge farm of MySQL servers powering a busy website.

The following discussion briefly summarizes the basic components that make up the MySQL Enterprise Monitor product. For more information, see the MySQL Enterprise Monitor manual, available at https://dev.mysql.com/doc/mysql-monitor/en/.

MySQL Enterprise Monitor components can be installed in various configurations depending on your database and network topology, to give you the best combination of reliable and responsive monitoring data, with minimal overhead on the database server machines. A typical MySQL Enterprise Monitor installation consists of:

- One or more MySQL servers to monitor. MySQL Enterprise Monitor can monitor both Community and Enterprise MySQL server releases.
- A MySQL Enterprise Monitor Agent for each monitored host.
- A single MySQL Enterprise Service Manager, which collates information from the agents and provides the user interface to the collected data.

MySQL Enterprise Monitor is designed to monitor one or more MySQL servers. The monitoring information is collected by using an agent, *MySQL Enterprise Monitor Agent*. The agent communicates with the hosts and MySQL servers that it monitors, collecting variables, status and health information, and sending this information to the MySQL Enterprise Service Manager.

The information collected by the agent about each MySQL server and host you are monitoring is sent to the *MySQL Enterprise Service Manager*. This server collates all of the information from the agents. As it collates the information sent by the agents, the MySQL Enterprise Service Manager continually tests the collected data, comparing the status of the server to reasonable values. When thresholds are reached, the server can trigger an event (including an alarm and notification) to highlight a potential issue, such as low memory, high CPU usage, or more complex conditions such insufficient buffer sizes and status information. We call each test, with its associated threshold value, a *rule*.

These rules, and the alarms and notifications, are each known as a *MySQL Enterprise Advisors*. Advisors form a critical part of the MySQL Enterprise Service Manager, as they provide warning information and troubleshooting advice about potential problems.

The MySQL Enterprise Service Manager includes a web server, and you interact with it through any web browser. This interface, the MySQL Enterprise Monitor User Interface, displays all of the information collected by the agents, and lets you view all of your servers and their current status as a group or individually. You control and configure all aspects of the service using the MySQL Enterprise Monitor User Interface.

The information supplied by the MySQL Enterprise Monitor Agent processes also includes statistical and query information, which you can view in the form of graphs. For example, you can view aspects such as server load, query numbers, or index usage information as a graph over time. The graph lets you pinpoint problems or potential issues on your server, and can help diagnose the impact from database or external problems (such as external system or network failure) by examining the data from a specific time interval.

The MySQL Enterprise Monitor Agent can also be configured to collect detailed information about the queries executed on your server, including the row counts and performance times for executing each query. You can correlate the detailed query data with the graphical information to identify which queries were executing when you experienced a particularly high load, index or other issue. The query data is supported by a system called Query Analyzer, and the data can be presented in different ways depending on your needs.

30.2 MySQL Enterprise Backup Overview

MySQL Enterprise Backup performs hot backup operations for MySQL databases. The product is architected for efficient and reliable backups of tables created by the InnoDB storage engine. For completeness, it can also back up tables from MyISAM and other storage engines.

The following discussion briefly summarizes MySQL Enterprise Backup. For more information, see the MySQL Enterprise Backup manual, available at https://dev.mysql.com/doc/mysql-enterprise-backup/en/.

Hot backups are performed while the database is running and applications are reading and writing to it. This type of backup does not block normal database operations, and it captures even changes that occur while the backup is happening. For these reasons, hot backups are desirable when your database

"grows up" -- when the data is large enough that the backup takes significant time, and when your data is important enough to your business that you must capture every last change, without taking your application, website, or web service offline.

MySQL Enterprise Backup does a hot backup of all tables that use the InnoDB storage engine. For tables using MyISAM or other non-InnoDB storage engines, it does a "warm" backup, where the database continues to run, but those tables cannot be modified while being backed up. For efficient backup operations, you can designate InnoDB as the default storage engine for new tables, or convert existing tables to use the InnoDB storage engine.

30.3 MySQL Enterprise Security Overview

MySQL Enterprise Edition provides plugins that implement security features using external services:

- MySQL Enterprise Edition includes an authentication plugin that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as Unix passwords or an LDAP directory. For more information, see Section 6.4.1.5, "PAM Pluggable Authentication".
- MySQL Enterprise Edition includes an authentication plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password. For more information, see Section 6.4.1.6, "Windows Pluggable Authentication".
- MySQL Enterprise Edition 5.6 and higher includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. These functions enable masking existing data using several methods such as obfuscation (removing identifying characteristics), generation of formatted random data, and data replacement or substitution. For more information, see Section 30.4, "MySQL Enterprise Encryption Overview".
- MySQL Enterprise Edition 5.7 and higher includes a keyring plugin that uses Oracle Key Vault as a back end for keyring storage. For more information, see Section 6.4.4, "The MySQL Keyring".

For other related Enterprise security features, see Section 30.4, "MySQL Enterprise Encryption Overview".

30.4 MySQL Enterprise Encryption Overview

MySQL Enterprise Edition 5.6 and higher includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. These functions enable Enterprise applications to perform the following operations:

- Implement added data protection using public-key asymmetric cryptography
- Create public and private keys and digital signatures
- Perform asymmetric encryption and decryption
- Use cryptographic hashing for digital signing and data verification and validation

For more information, see Section 12.19, "MySQL Enterprise Encryption Functions".

For other related Enterprise security features, see Section 30.3, "MySQL Enterprise Security Overview".

30.5 MySQL Enterprise Audit Overview

MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

For more information, see Section 6.4.5, “MySQL Enterprise Audit”.

30.6 MySQL Enterprise Firewall Overview

MySQL Enterprise Edition 5.6 and higher includes MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against whitelists of accepted statement patterns. This helps harden MySQL Server against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics.

Each MySQL account registered with the firewall has its own statement whitelist, enabling protection to be tailored per account. For a given account, the firewall can operate in recording or protecting mode, for training in the accepted statement patterns or protection against unacceptable statements.

For more information, see Section 6.4.7, “MySQL Enterprise Firewall”.

30.7 MySQL Enterprise Thread Pool Overview

MySQL Enterprise Edition includes MySQL Enterprise Thread Pool, implemented using a server plugin. The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. In MySQL Enterprise Edition, a thread pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The plugin implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections.

For more information, see Section 5.6.3, “MySQL Enterprise Thread Pool”.

30.8 MySQL Enterprise Data Masking and De-Identification Overview

MySQL Enterprise Edition 5.7 and higher includes MySQL Enterprise Data Masking and De-Identification, implemented as a plugin library containing a plugin and a set of user-defined functions. Data masking hides sensitive information by replacing real values with substitutes. MySQL Enterprise Data Masking and De-Identification functions enable masking existing data using several methods such as obfuscation (removing identifying characteristics), generation of formatted random data, and data replacement or substitution.

For more information, see Section 6.4.8, “MySQL Enterprise Data Masking and De-Identification”.

Chapter 31 MySQL Workbench

MySQL Workbench provides a graphical tool for working with MySQL servers and databases. MySQL Workbench fully supports MySQL versions 5.5 and higher.

The following discussion briefly describes MySQL Workbench capabilities. For more information, see the MySQL Workbench manual, available at https://dev.mysql.com/doc/workbench/en/.

MySQL Workbench provides five main areas of functionality:

- **SQL Development**: Enables you to create and manage connections to database servers. As well as enabling you to configure connection parameters, MySQL Workbench provides the capability to execute SQL queries on the database connections using the built-in SQL Editor. This functionality replaces that previously provided by the Query Browser standalone application.
- **Data Modeling**: Enables you to create models of your database schema graphically, reverse and forward engineer between a schema and a live database, and edit all aspects of your database using the comprehensive Table Editor. The Table Editor provides easy-to-use facilities for editing Tables, Columns, Indexes, Triggers, Partitioning, Options, Inserts and Privileges, Routines and Views.
- **Server Administration**: Enables you to create and administer server instances.
- **Data Migration**: Allows you to migrate from Microsoft SQL Server, Sybase ASE, SQLite, SQL Anywhere, PostreSQL, and other RDBMS tables, objects and data to MySQL. Migration also supports migrating from earlier versions of MySQL to the latest releases.
- **MySQL Enterprise Support**: Support for Enterprise products such as MySQL Enterprise Backup and MySQL Audit.

MySQL Workbench is available in two editions, the Community Edition and the Commercial Edition. The Community Edition is available free of charge. The Commercial Edition provides additional Enterprise features, such as database documentation generation, at low cost.

Appendix A MySQL 8.0 Frequently Asked Questions

Table of Contents

A.1 MySQL 8.0 FAQ: General 5057
A.2 MySQL 8.0 FAQ: Storage Engines 5058
A.3 MySQL 8.0 FAQ: Server SQL Mode 5059
A.4 MySQL 8.0 FAQ: Stored Procedures and Functions 5060
A.5 MySQL 8.0 FAQ: Triggers 5064
A.6 MySQL 8.0 FAQ: Views 5067
A.7 MySQL 8.0 FAQ: INFORMATION_SCHEMA 5067
A.8 MySQL 8.0 FAQ: Migration 5068
A.9 MySQL 8.0 FAQ: Security 5069
A.10 MySQL 8.0 FAQ: NDB Cluster 5070
A.11 MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets 5071
A.12 MySQL 8.0 FAQ: Connectors & APIs 5082
A.13 MySQL 8.0 FAQ: C API, Connector/C, libmysql 5082
A.14 MySQL 8.0 FAQ: Replication 5083
A.15 MySQL 8.0 FAQ: MySQL Enterprise Thread Pool 5087
A.16 MySQL 8.0 FAQ: InnoDB Change Buffer 5088
A.17 MySQL 8.0 FAQ: InnoDB Data-at-Rest Encryption 5090
A.18 MySQL 8.0 FAQ: Virtualization Support 5092

A.1 MySQL 8.0 FAQ: General

A.1.1 Which version of MySQL is production-ready (GA)? 5057
A.1.2 Why did MySQL version numbering skip versions 6 and 7 and go straight to 8.0? 5058
A.1.3 Can MySQL 8.0 do subqueries? 5058
A.1.4 Can MySQL 8.0 perform multiple-table inserts, updates, and deletes? 5058
A.1.5 Does MySQL 8.0 have Sequences? 5058
A.1.6 Does MySQL 8.0 have a `NOW()` function with fractions of seconds? 5058
A.1.7 Does MySQL 8.0 work with multi-core processors? 5058
A.1.8 Why do I see multiple processes for `mysqld`? 5058
A.1.9 Can MySQL 8.0 perform ACID transactions? 5058

A.1.1. Which version of MySQL is production-ready (GA)?

MySQL 8.0, 5.7, and MySQL 5.6 are supported for production use.

MySQL 8.0 achieved General Availability (GA) status with MySQL 8.0.11, which was released for production use on 19 April 2018.

MySQL 5.7 achieved General Availability (GA) status with MySQL 5.7.9, which was released for production use on 21 October 2015.

MySQL 5.6 achieved General Availability (GA) status with MySQL 5.6.10, which was released for production use on 5 February 2013.

MySQL 5.5 achieved General Availability (GA) status with MySQL 5.5.8, which was released for production use on 3 December 2010. The MySQL 5.5 series is no longer current, but still supported in production.

MySQL 5.1 achieved General Availability (GA) status with MySQL 5.1.30, which was released for production use on 14 November 2008. Active development for MySQL 5.1 has ended.

MySQL 5.0 achieved General Availability (GA) status with MySQL 5.0.15, which was released for production use on 19 October 2005. Active development for MySQL 5.0 has ended.

A.1.2. Why did MySQL version numbering skip versions 6 and 7 and go straight to 8.0?

Due to the many new and important features we were introducing in this MySQL version, we decided to start a fresh new series. As the series numbers 6 and 7 had actually been used before by MySQL, we went to 8.0.

A.1.3. Can MySQL 8.0 do subqueries?

Yes. See Section 13.2.11, “Subquery Syntax”.

A.1.4. Can MySQL 8.0 perform multiple-table inserts, updates, and deletes?

Yes. For the syntax required to perform multiple-table updates, see Section 13.2.12, “UPDATE Syntax”; for that required to perform multiple-table deletes, see Section 13.2.2, “DELETE Syntax”.

A multiple-table insert can be accomplished using a trigger whose `FOR EACH ROW` clause contains multiple `INSERT` statements within a `BEGIN ... END` block. See Section 24.3, “Using Triggers”.

A.1.5. Does MySQL 8.0 have Sequences?

No. However, MySQL has an `AUTO_INCREMENT` system, which in MySQL 8.0 can also handle inserts in a multi-master replication setup. With the `auto_increment_increment` and `auto_increment_offset` system variables, you can set each server to generate auto-increment values that don't conflict with other servers. The `auto_increment_increment` value should be greater than the number of servers, and each server should have a unique offset.

A.1.6. Does MySQL 8.0 have a `NOW()` function with fractions of seconds?

Yes, see Section 11.3.5, “Fractional Seconds in Time Values”.

A.1.7. Does MySQL 8.0 work with multi-core processors?

Yes. MySQL is fully multithreaded, and will make use of multiple CPUs, provided that the operating system supports them.

A.1.8. Why do I see multiple processes for `mysqld`?

When using LinuxThreads, you should see a minimum of three `mysqld` processes running. These are in fact threads. There is one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

A.1.9. Can MySQL 8.0 perform ACID transactions?

Yes. All current MySQL versions support transactions. The `InnoDB` storage engine offers full ACID transactions with row-level locking, multi-versioning, nonlocking repeatable reads, and all four SQL standard isolation levels.

The `NDB` storage engine supports the `READ COMMITTED` transaction isolation level only.

A.2 MySQL 8.0 FAQ: Storage Engines

A.2.1 Where can I obtain complete documentation for MySQL storage engines? 5059
A.2.2 Are there any new storage engines in MySQL 8.0? 5059
A.2.3 Have any storage engines been removed in MySQL 8.0? 5059
A.2.4 Can I prevent the use of a particular storage engine? 5059

A.2.5 Is there an advantage to using the InnoDB storage engine exclusively, as opposed to a combination of InnoDB and non-InnoDB storage engines? 5059
A.2.6 What are the unique benefits of the ARCHIVE storage engine? 5059

A.2.1. Where can I obtain complete documentation for MySQL storage engines?

See Chapter 16, *Alternative Storage Engines*. That chapter contains information about all MySQL storage engines except for the InnoDB storage engine and the NDB storage engine (used for MySQL Cluster). InnoDB is covered in Chapter 15, *The InnoDB Storage Engine*. NDB is covered in Chapter 22, *MySQL NDB Cluster 8.0*.

A.2.2. Are there any new storage engines in MySQL 8.0?

No. InnoDB is the default storage engine for new tables. See Section 15.1, “Introduction to InnoDB” for details.

A.2.3. Have any storage engines been removed in MySQL 8.0?

The PARTITION storage engine plugin which provided partitioning support is replaced by a native partitioning handler. As part of this change, the server can no longer be built using -DWITH_PARTITION_STORAGE_ENGINE. partition is also no longer displayed in the output of SHOW PLUGINS, or shown in the INFORMATION_SCHEMA.PLUGINS table.

In order to support partitioning of a given table, the storage engine used for the table must now provide its own (“native”) partitioning handler. InnoDB is the only storage engine supported in MySQL 8.0 that includes a native partitioning handler. An attempt to create partitioned tables in MySQL 8.0 using any other storage engine fails. (The NDB storage engine used by MySQL Cluster also provides its own partitioning handler, but is currently not supported by MySQL 8.0.)

A.2.4. Can I prevent the use of a particular storage engine?

Yes. The disabled_storage_engines configuration option defines which storage engines cannot be used to create tables or tablespaces. By default, disabled_storage_engines is empty (no engines disabled), but it can be set to a comma-separated list of one or more engines.

A.2.5. Is there an advantage to using the InnoDB storage engine exclusively, as opposed to a combination of InnoDB and non-InnoDB storage engines?

Yes. Using InnoDB tables exclusively can simplify backup and recovery operations. MySQL Enterprise Backup does a hot backup of all tables that use the InnoDB storage engine. For tables using MyISAM or other non-InnoDB storage engines, it does a “warm” backup, where the database continues to run, but those tables cannot be modified while being backed up. See Section 30.2, “MySQL Enterprise Backup Overview”.

A.2.6. What are the unique benefits of the ARCHIVE storage engine?

The ARCHIVE storage engine stores large amounts of data without indexes; it has a small footprint, and performs selects using table scans. See Section 16.5, “The ARCHIVE Storage Engine”, for details.

A.3 MySQL 8.0 FAQ: Server SQL Mode

A.3.1 What are server SQL modes? 5060
A.3.2 How many server SQL modes are there? 5060
A.3.3 How do you determine the server SQL mode? 5060
A.3.4 Is the mode dependent on the database or connection? 5060
A.3.5 Can the rules for strict mode be extended? 5060

A.3.6 Does strict mode impact performance? 5060
A.3.7 What is the default server SQL mode when MySQL 8.0 is installed? 5060

A.3.1. What are server SQL modes?

Server SQL modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers. The MySQL Server apply these modes individually to different clients. For more information, see Section 5.1.11, “Server SQL Modes”.

A.3.2. How many server SQL modes are there?

Each mode can be independently switched on and off. See Section 5.1.11, “Server SQL Modes”, for a complete list of available modes.

A.3.3. How do you determine the server SQL mode?

You can set the default SQL mode (for `mysqld` startup) with the `--sql-mode` option. Using the statement `SET [GLOBAL|SESSION] sql_mode='modes'`, you can change the settings from within a connection, either locally to the connection, or to take effect globally. You can retrieve the current mode by issuing a `SELECT @@sql_mode` statement.

A.3.4. Is the mode dependent on the database or connection?

A mode is not linked to a particular database. Modes can be set locally to the session (connection), or globally for the server. you can change these settings using `SET [GLOBAL|SESSION] sql_mode='modes'`.

A.3.5. Can the rules for strict mode be extended?

When we refer to *strict mode*, we mean a mode where at least one of the modes `TRADITIONAL`, `STRICT_TRANS_TABLES`, or `STRICT_ALL_TABLES` is enabled. Options can be combined, so you can add restrictions to a mode. See Section 5.1.11, “Server SQL Modes”, for more information.

A.3.6. Does strict mode impact performance?

The intensive validation of input data that some settings requires more time than if the validation is not done. While the performance impact is not that great, if you do not require such validation (perhaps your application already handles all of this), then MySQL gives you the option of leaving strict mode disabled. However, if you do require it, strict mode can provide such validation.

A.3.7. What is the default server SQL mode when MySQL 8.0 is installed?

The default SQL mode in MySQL 8.0 includes these modes: `ONLY_FULL_GROUP_BY`, `STRICT_TRANS_TABLES`, `NO_ZERO_IN_DATE`, `NO_ZERO_DATE`, `ERROR_FOR_DIVISION_BY_ZERO`, and `NO_ENGINE_SUBSTITUTION`.

For information about all available modes and default MySQL behavior, see Section 5.1.11, “Server SQL Modes”.

A.4 MySQL 8.0 FAQ: Stored Procedures and Functions

A.4.1 Does MySQL 8.0 support stored procedures and functions? 5061
A.4.2 Where can I find documentation for MySQL stored procedures and stored functions? 5061
A.4.3 Is there a discussion forum for MySQL stored procedures? 5061
A.4.4 Where can I find the ANSI SQL 2003 specification for stored procedures? 5061
A.4.5 How do you manage stored routines? 5061
A.4.6 Is there a way to view all stored procedures and stored functions in a given database? 5061

A.4.7 Where are stored procedures stored? 5062
A.4.8 Is it possible to group stored procedures or stored functions into packages? 5062
A.4.9 Can a stored procedure call another stored procedure? 5062
A.4.10 Can a stored procedure call a trigger? 5062
A.4.11 Can a stored procedure access tables? 5062
A.4.12 Do stored procedures have a statement for raising application errors? 5062
A.4.13 Do stored procedures provide exception handling? 5062
A.4.14 Can MySQL 8.0 stored routines return result sets? 5062
A.4.15 Is `WITH RECOMPILE` supported for stored procedures? 5062
A.4.16 Is there a MySQL equivalent to using `mod_plsql` as a gateway on Apache to talk directly to a stored procedure in the database? 5063
A.4.17 Can I pass an array as input to a stored procedure? 5063
A.4.18 Can I pass a cursor as an `IN` parameter to a stored procedure? 5063
A.4.19 Can I return a cursor as an `OUT` parameter from a stored procedure? 5063
A.4.20 Can I print out a variable's value within a stored routine for debugging purposes? 5063
A.4.21 Can I commit or roll back transactions inside a stored procedure? 5063
A.4.22 Do MySQL 8.0 stored procedures and functions work with replication? 5063
A.4.23 Are stored procedures and functions created on a master server replicated to a slave? 5063
A.4.24 How are actions that take place inside stored procedures and functions replicated? 5063
A.4.25 Are there special security requirements for using stored procedures and functions together with replication? 5063
A.4.26 What limitations exist for replicating stored procedure and function actions? 5064
A.4.27 Do the preceding limitations affect the ability of MySQL to do point-in-time recovery? 5064
A.4.28 What is being done to correct the aforementioned limitations? 5064

A.4.1. Does MySQL 8.0 support stored procedures and functions?

Yes. MySQL 8.0 supports two types of stored routines, stored procedures and stored functions.

A.4.2. Where can I find documentation for MySQL stored procedures and stored functions?

See Section 24.2, “Using Stored Routines”.

A.4.3. Is there a discussion forum for MySQL stored procedures?

Yes. See https://forums.mysql.com/list.php?98.

A.4.4. Where can I find the ANSI SQL 2003 specification for stored procedures?

Unfortunately, the official specifications are not freely available (ANSI makes them available for purchase). However, there are books, such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer, that provide a comprehensive overview of the standard, including coverage of stored procedures.

A.4.5. How do you manage stored routines?

It is always good practice to use a clear naming scheme for your stored routines. You can manage stored procedures with `CREATE [FUNCTION|PROCEDURE]`, `ALTER [FUNCTION|PROCEDURE]`, `DROP [FUNCTION|PROCEDURE]`, and `SHOW CREATE [FUNCTION|PROCEDURE]`. You can obtain information about existing stored procedures using the `ROUTINES` table in the `INFORMATION_SCHEMA` database (see Section 25.23, “The INFORMATION_SCHEMA ROUTINES Table”).

A.4.6. Is there a way to view all stored procedures and stored functions in a given database?

Yes. For a database named *`dbname`*, use this query on the `INFORMATION_SCHEMA.ROUTINES` table:

```
SELECT ROUTINE_TYPE, ROUTINE_NAME
    FROM INFORMATION_SCHEMA.ROUTINES
    WHERE ROUTINE_SCHEMA='dbname';
```

For more information, see Section 25.23, “The INFORMATION_SCHEMA ROUTINES Table”.

The body of a stored routine can be viewed using `SHOW CREATE FUNCTION` (for a stored function) or `SHOW CREATE PROCEDURE` (for a stored procedure). See Section 13.7.7.9, “SHOW CREATE PROCEDURE Syntax”, for more information.

A.4.7. Where are stored procedures stored?

Stored procedures are stored in the `mysql.routines` and `mysql.parameters` tables, which are part of the data dictionary. You cannot access these tables directly. Instead, query the `INFORMATION_SCHEMA ROUTINES` and `PARAMETERS` tables. See Section 25.23, “The INFORMATION_SCHEMA ROUTINES Table”, and Section 25.16, “The INFORMATION_SCHEMA PARAMETERS Table”.

You can also use `SHOW CREATE FUNCTION` to obtain information about stored functions, and `SHOW CREATE PROCEDURE` to obtain information about stored procedures. See Section 13.7.7.9, “SHOW CREATE PROCEDURE Syntax”.

A.4.8. Is it possible to group stored procedures or stored functions into packages?

No. This is not supported in MySQL 8.0.

A.4.9. Can a stored procedure call another stored procedure?

Yes.

A.4.10. Can a stored procedure call a trigger?

A stored procedure can execute an SQL statement, such as an `UPDATE`, that causes a trigger to activate.

A.4.11. Can a stored procedure access tables?

Yes. A stored procedure can access one or more tables as required.

A.4.12. Do stored procedures have a statement for raising application errors?

Yes. MySQL 8.0 implements the SQL standard `SIGNAL` and `RESIGNAL` statements. See Section 13.6.7, “Condition Handling”.

A.4.13. Do stored procedures provide exception handling?

MySQL implements `HANDLER` definitions according to the SQL standard. See Section 13.6.7.2, “DECLARE ... HANDLER Syntax”, for details.

A.4.14. Can MySQL 8.0 stored routines return result sets?

Stored procedures can, but stored functions cannot. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You need to use the MySQL 4.1 (or higher) client/server protocol for this to work. This means that, for example, in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

A.4.15. Is `WITH RECOMPILE` supported for stored procedures?

Not in MySQL 8.0.

A.4.16 Is there a MySQL equivalent to using `mod_plsql` as a gateway on Apache to talk directly to a stored procedure in the database?

There is no equivalent in MySQL 8.0.

A.4.17 Can I pass an array as input to a stored procedure?

Not in MySQL 8.0.

A.4.18 Can I pass a cursor as an `IN` parameter to a stored procedure?

In MySQL 8.0, cursors are available inside stored procedures only.

A.4.19 Can I return a cursor as an `OUT` parameter from a stored procedure?

In MySQL 8.0, cursors are available inside stored procedures only. However, if you do not open a cursor on a `SELECT`, the result will be sent directly to the client. You can also `SELECT INTO` variables. See Section 13.2.10, “SELECT Syntax”.

A.4.20 Can I print out a variable's value within a stored routine for debugging purposes?

Yes, you can do this in a *stored procedure*, but not in a stored function. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You will need to use the MySQL 4.1 (or above) client/server protocol for this to work. This means that, for example, in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

A.4.21 Can I commit or roll back transactions inside a stored procedure?

Yes. However, you cannot perform transactional operations within a stored function.

A.4.22 Do MySQL 8.0 stored procedures and functions work with replication?

Yes, standard actions carried out in stored procedures and functions are replicated from a master MySQL server to a slave server. There are a few limitations that are described in detail in Section 24.7, “Stored Program Binary Logging”.

A.4.23 Are stored procedures and functions created on a master server replicated to a slave?

Yes, creation of stored procedures and functions carried out through normal DDL statements on a master server are replicated to a slave, so the objects will exist on both servers. `ALTER` and `DROP` statements for stored procedures and functions are also replicated.

A.4.24 How are actions that take place inside stored procedures and functions replicated?

MySQL records each DML event that occurs in a stored procedure and replicates those individual actions to a slave server. The actual calls made to execute stored procedures are not replicated.

Stored functions that change data are logged as function invocations, not as the DML events that occur inside each function.

A.4.25 Are there special security requirements for using stored procedures and functions together with replication?

Yes. Because a slave server has authority to execute any statement read from a master's binary log, special security constraints exist for using stored functions with replication. If replication or binary logging in general (for the purpose of point-in-time recovery) is active, then MySQL DBAs have two security options open to them:

1. Any user wishing to create stored functions must be granted the `SUPER` privilege.

2. Alternatively, a DBA can set the `log_bin_trust_function_creators` system variable to 1, which enables anyone with the standard `CREATE ROUTINE` privilege to create stored functions.

A.4.26. What limitations exist for replicating stored procedure and function actions?

Nondeterministic (random) or time-based actions embedded in stored procedures may not replicate properly. By their very nature, randomly produced results are not predictable and cannot be exactly reproduced, and therefore, random actions replicated to a slave will not mirror those performed on a master. Declaring stored functions to be `DETERMINISTIC` or setting the `log_bin_trust_function_creators` system variable to 0 will not allow random-valued operations to be invoked.

In addition, time-based actions cannot be reproduced on a slave because the timing of such actions in a stored procedure is not reproducible through the binary log used for replication. It records only DML events and does not factor in timing constraints.

Finally, nontransactional tables for which errors occur during large DML actions (such as bulk inserts) may experience replication issues in that a master may be partially updated from DML activity, but no updates are done to the slave because of the errors that occurred. A workaround is for a function's DML actions to be carried out with the `IGNORE` keyword so that updates on the master that cause errors are ignored and updates that do not cause errors are replicated to the slave.

A.4.27. Do the preceding limitations affect the ability of MySQL to do point-in-time recovery?

The same limitations that affect replication do affect point-in-time recovery.

A.4.28. What is being done to correct the aforementioned limitations?

You can choose either statement-based replication or row-based replication. The original replication implementation is based on statement-based binary logging. Row-based binary logging resolves the limitations mentioned earlier.

Mixed replication is also available (by starting the server with `--binlog-format=mixed`). This hybrid form of replication “knows” whether statement-level replication can safely be used, or row-level replication is required.

For additional information, see Section 17.2.1, “Replication Formats”.

A.5 MySQL 8.0 FAQ: Triggers

A.5.1 Where can I find the documentation for MySQL 8.0 triggers? 5065
A.5.2 Is there a discussion forum for MySQL Triggers? 5065
A.5.3 Does MySQL 8.0 have statement-level or row-level triggers? 5065
A.5.4 Are there any default triggers? 5065
A.5.5 How are triggers managed in MySQL? 5065
A.5.6 Is there a way to view all triggers in a given database? 5065
A.5.7 Where are triggers stored? 5065
A.5.8 Can a trigger call a stored procedure? 5065
A.5.9 Can triggers access tables? 5065
A.5.10 Can a table have multiple triggers with the same trigger event and action time? 5065
A.5.11 Can triggers call an external application through a UDF? 5066
A.5.12 Is it possible for a trigger to update tables on a remote server? 5066

A.5.13 Do triggers work with replication? 5066
A.5.14 How are actions carried out through triggers on a master replicated to a slave? 5066

A.5.1. Where can I find the documentation for MySQL 8.0 triggers?

See Section 24.3, “Using Triggers”.

A.5.2. Is there a discussion forum for MySQL Triggers?

Yes. It is available at https://forums.mysql.com/list.php?99.

A.5.3. Does MySQL 8.0 have statement-level or row-level triggers?

In MySQL 8.0, all triggers are `FOR EACH ROW`; that is, the trigger is activated for each row that is inserted, updated, or deleted. MySQL 8.0 does not support triggers using `FOR EACH STATEMENT`.

A.5.4. Are there any default triggers?

Not explicitly. MySQL does have specific special behavior for some `TIMESTAMP` columns, as well as for columns which are defined using `AUTO_INCREMENT`.

A.5.5. How are triggers managed in MySQL?

In MySQL 8.0, triggers can be created using the `CREATE TRIGGER` statement, and dropped using `DROP TRIGGER`. See Section 13.1.22, “CREATE TRIGGER Syntax”, and Section 13.1.34, “DROP TRIGGER Syntax”, for more about these statements.

Information about triggers can be obtained by querying the `INFORMATION_SCHEMA.TRIGGERS` table. See Section 25.34, “The INFORMATION_SCHEMA TRIGGERS Table”.

A.5.6. Is there a way to view all triggers in a given database?

Yes. You can obtain a listing of all triggers defined on database `dbname` using a query on the `INFORMATION_SCHEMA.TRIGGERS` table such as the one shown here:

```
SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_STATEMENT
    FROM INFORMATION_SCHEMA.TRIGGERS
    WHERE TRIGGER_SCHEMA='dbname';
```

For more information about this table, see Section 25.34, “The INFORMATION_SCHEMA TRIGGERS Table”.

You can also use the `SHOW TRIGGERS` statement, which is specific to MySQL. See Section 13.7.7.38, “SHOW TRIGGERS Syntax”.

A.5.7. Where are triggers stored?

Triggers are stored in the `mysql.triggers` system table, which is part of the data dictionary.

A.5.8. Can a trigger call a stored procedure?

Yes.

A.5.9. Can triggers access tables?

A trigger can access both old and new data in its own table. A trigger can also affect other tables, but it is not permitted to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.

A.5.10. Can a table have multiple triggers with the same trigger event and action time?

In MySQL 8.0, it is possible to define multiple triggers for a given table that have the same trigger event and action time. For example, you can have two `BEFORE UPDATE` triggers for a table. By default, triggers that have the same trigger event and action time activate in the order they were created. To affect trigger order, specify a clause after `FOR EACH ROW` that indicates `FOLLOWS` or `PRECEDES` and the name of an existing trigger that also has the same trigger event and action time. With `FOLLOWS`, the new trigger activates after the existing trigger. With `PRECEDES`, the new trigger activates before the existing trigger.

A.5.11 Can triggers call an external application through a UDF?

Yes. For example, a trigger could invoke the `sys_exec()` UDF.

A.5.12 Is it possible for a trigger to update tables on a remote server?

Yes. A table on a remote server could be updated using the `FEDERATED` storage engine. (See Section 16.8, “The FEDERATED Storage Engine”).

A.5.13 Do triggers work with replication?

Yes. However, the way in which they work depends whether you are using MySQL's “classic” statement-based or row-based replication format.

When using statement-based replication, triggers on the slave are executed by statements that are executed on the master (and replicated to the slave).

When using row-based replication, triggers are not executed on the slave due to statements that were run on the master and then replicated to the slave. Instead, when using row-based replication, the changes caused by executing the trigger on the master are applied on the slave.

For more information, see Section 17.5.1.35, “Replication and Triggers”.

A.5.14 How are actions carried out through triggers on a master replicated to a slave?

Again, this depends on whether you are using statement-based or row-based replication.

Statement-based replication. First, the triggers that exist on a master must be re-created on the slave server. Once this is done, the replication flow works as any other standard DML statement that participates in replication. For example, consider a table `EMP` that has an `AFTER` insert trigger, which exists on a master MySQL server. The same `EMP` table and `AFTER` insert trigger exist on the slave server as well. The replication flow would be:

1. An `INSERT` statement is made to `EMP`.
2. The `AFTER` trigger on `EMP` activates.
3. The `INSERT` statement is written to the binary log.
4. The replication slave picks up the `INSERT` statement to `EMP` and executes it.
5. The `AFTER` trigger on `EMP` that exists on the slave activates.

Row-based replication. When you use row-based replication, the changes caused by executing the trigger on the master are applied on the slave. However, the triggers themselves are not actually executed on the slave under row-based replication. This is because, if both the master and the slave applied the changes from the master and, in addition, the trigger causing these changes were applied on the slave, the changes would in effect be applied twice on the slave, leading to different data on the master and the slave.

In most cases, the outcome is the same for both row-based and statement-based replication. However, if you use different triggers on the master and slave, you cannot use row-based replication. (This is because the row-based format replicates the changes made by triggers executing on the master to the slaves, rather than the statements that caused the triggers to execute, and the corresponding triggers on the slave are not executed.) Instead, any statements causing such triggers to be executed must be replicated using statement-based replication.

For more information, see Section 17.5.1.35, “Replication and Triggers”.

A.6 MySQL 8.0 FAQ: Views

A.6.1 Where can I find documentation covering MySQL Views? 5067
A.6.2 Is there a discussion forum for MySQL Views? 5067
A.6.3 What happens to a view if an underlying table is dropped or renamed? 5067
A.6.4 Does MySQL 8.0 have table snapshots? 5067
A.6.5 Does MySQL 8.0 have materialized views? 5067
A.6.6 Can you insert into views that are based on joins? 5067

A.6.1. Where can I find documentation covering MySQL Views?

See Section 24.5, “Using Views”.

A.6.2. Is there a discussion forum for MySQL Views?

Yes. See https://forums.mysql.com/list.php?100

A.6.3. What happens to a view if an underlying table is dropped or renamed?

After a view has been created, it is possible to drop or alter a table or view to which the definition refers. To check a view definition for problems of this kind, use the `CHECK TABLE` statement. (See Section 13.7.3.2, “CHECK TABLE Syntax”.)

A.6.4. Does MySQL 8.0 have table snapshots?

No.

A.6.5. Does MySQL 8.0 have materialized views?

No.

A.6.6. Can you insert into views that are based on joins?

It is possible, provided that your `INSERT` statement has a column list that makes it clear there is only one table involved.

You *cannot* insert into multiple tables with a single insert on a view.

A.7 MySQL 8.0 FAQ: INFORMATION_SCHEMA

A.7.1 Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database? 5068
A.7.2 Is there a discussion forum for `INFORMATION_SCHEMA`? 5068
A.7.3 Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`? 5068
A.7.4 What is the difference between the Oracle Data Dictionary and MySQL `INFORMATION_SCHEMA`? 5068
A.7.5 Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database? 5068

A.7.1. Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?

See Chapter 25, *INFORMATION_SCHEMA Tables*

A.7.2. Is there a discussion forum for `INFORMATION_SCHEMA`?

See https://forums.mysql.com/list.php?101.

A.7.3. Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?

Unfortunately, the official specifications are not freely available. (ANSI makes them available for purchase.) However, there are books available, such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer, that provide a comprehensive overview of the standard, including `INFORMATION_SCHEMA`.

A.7.4. What is the difference between the Oracle Data Dictionary and MySQL `INFORMATION_SCHEMA`?

Both Oracle and MySQL provide metadata in tables. However, Oracle and MySQL use different table names and column names. The MySQL implementation is more similar to those found in DB2 and SQL Server, which also support `INFORMATION_SCHEMA` as defined in the SQL standard.

A.7.5. Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?

No. Since applications may rely on a certain standard structure, this should not be modified. For this reason, *we cannot support bugs or other issues which result from modifying `INFORMATION_SCHEMA` tables or data*.

A.8 MySQL 8.0 FAQ: Migration

A.8.1 Where can I find information on how to migrate from MySQL 5.7 to MySQL 8.0? 5068
A.8.2 How has storage engine (table type) support changed in MySQL 8.0 from previous versions? ... 5068

A.8.1. Where can I find information on how to migrate from MySQL 5.7 to MySQL 8.0?

For detailed upgrade information, see Section 2.11, “Upgrading MySQL”. Do not skip a major version when upgrading, but rather complete the process in steps, upgrading from one major version to the next in each step. This may seem more complicated, but it will you save time and trouble. If you encounter problems during the upgrade, their origin will be easier to identify, either by you or, if you have a MySQL Enterprise subscription, by MySQL support.

A.8.2. How has storage engine (table type) support changed in MySQL 8.0 from previous versions?

Storage engine support has changed as follows:

- Support for `ISAM` tables was removed in MySQL 5.0 and you should now use the `MyISAM` storage engine in place of `ISAM`. To convert a table *`tblname`* from `ISAM` to `MyISAM`, simply issue a statement such as this one:

  ```
  ALTER TABLE tblname ENGINE=MYISAM;
  ```

- Internal `RAID` for `MyISAM` tables was also removed in MySQL 5.0. This was formerly used to allow large tables in file systems that did not support file sizes greater than 2GB. All modern file systems allow for larger tables; in addition, there are now other solutions such as `MERGE` tables and views.

- The `VARCHAR` column type now retains trailing spaces in all storage engines.

- `MEMORY` tables (formerly known as `HEAP` tables) can also contain `VARCHAR` columns.

A.9 MySQL 8.0 FAQ: Security

A.9.1 Where can I find documentation that addresses security issues for MySQL? 5069
A.9.2 What is the default authentication plugin in MySQL 8.0? 5069
A.9.3 Does MySQL 8.0 have native support for SSL? 5069
A.9.4 Is SSL support built into MySQL binaries, or must I recompile the binary myself to enable it? 5069
A.9.5 Does MySQL 8.0 have built-in authentication against LDAP directories? 5070
A.9.6 Does MySQL 8.0 include support for Roles Based Access Control (RBAC)? 5070

A.9.1. Where can I find documentation that addresses security issues for MySQL?

The best place to start is Chapter 6, *Security*.

Other portions of the MySQL Documentation which you may find useful with regard to specific security concerns include the following:

- Section 6.1.1, “Security Guidelines”.
- Section 6.1.3, “Making MySQL Secure Against Attackers”.
- Section B.4.3.2, “How to Reset the Root Password”.
- Section 6.1.5, “How to Run MySQL as a Normal User”.
- Section 29.4.2.6, “UDF Security Precautions”.
- Section 6.1.4, “Security-Related mysqld Options and Variables”.
- Section 6.1.6, “Security Issues with LOAD DATA LOCAL”.
- Section 2.10, “Postinstallation Setup and Testing”.
- Section 6.3, “Using Encrypted Connections”.

A.9.2. What is the default authentication plugin in MySQL 8.0?

The default authentication plugin in MySQL 8.0 is `caching_sha2_password`. For information about this plugin, see Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”.

The `caching_sha2_password` plugin provides more secure password encryption than the `mysql_native_password` plugin (the default plugin in previous MySQL series). For information about the implications of this change of default plugin for server operation and compatibility of the server with clients and connectors, see caching_sha2_password as the Preferred Authentication Plugin.

For general information about pluggable authentication and other available authentication plugins, see Section 6.2.17, “Pluggable Authentication”, and Section 6.4.1, “Authentication Plugins”.

A.9.3. Does MySQL 8.0 have native support for SSL?

Most 8.0 binaries have support for SSL connections between the client and server. See Section 6.3, “Using Encrypted Connections”.

You can also tunnel a connection using SSH, if (for example) the client application does not support SSL connections. For an example, see Section 6.3.4, “Connecting to MySQL Remotely from Windows with SSH”.

A.9.4. Is SSL support built into MySQL binaries, or must I recompile the binary myself to enable it?

Most 8.0 binaries have SSL enabled for client/server connections that are secured, authenticated, or both. See Section 6.3, “Using Encrypted Connections”.

A.9.5. Does MySQL 8.0 have built-in authentication against LDAP directories?

The Enterprise edition includes a PAM Authentication Plugin that supports authentication against an LDAP directory.

A.9.6. Does MySQL 8.0 include support for Roles Based Access Control (RBAC)?

Not at this time.

A.10 MySQL 8.0 FAQ: NDB Cluster

In the following section, we answer questions that are frequently asked about MySQL NDB Cluster and the `NDB` storage engine.

A.10.1 Which versions of the MySQL software support NDB Cluster? Do I have to compile from source? 5070
A.10.2 What do “NDB” and “NDBCLUSTER” mean? 5070
A.10.3 How many computers do I need to run an NDB Cluster, and why? 5070

A.10.1 Which versions of the MySQL software support NDB Cluster? Do I have to compile from source?

NDB Cluster is not supported in MySQL Server 8.0 releases; it is released as a separate product. NDB Cluster 8.0 is now available as a Developer Preview release for evaluation and testing of new features in the `NDBCLUSTER` storage engine; for more information, see Chapter 22, *MySQL NDB Cluster 8.0*. You are strongly advised to use NDB Cluster 7.6 for any new deployments; if you are using an older version of NDB Cluster, we recommend that you upgrade to this version soon as possible. For an overview of improvements made in NDB Cluster 7.6, see What is New in NDB Cluster 7.6.

NDB Cluster 7.6, based on MySQL Server 5.7 and version 7.6 of the `NDB` storage engine, is is the most recent General Availability (GA) version of NDB Cluster, based on version 7.6 of the `NDB` storage engine and MySQL Server 5.7. NDB Cluster 7.6 is available for production use; new deployments intended for production should use the latest GA release in this series, which is currently NDB Cluster 7.6.12. You can obtain the most recent NDB Cluster 7.6 release from https://dev.mysql.com/downloads/cluster/. For information about new features and other important changes in this series, see What is New in NDB Cluster 7.6.

For detailed information about deploying and using NDB Cluster, see Chapter 22, *MySQL NDB Cluster 8.0*.

A.10.2 What do “NDB” and “NDBCLUSTER” mean?

“NDB” stands for “**N**etwork **D**ata**b**ase”. `NDB` and `NDBCLUSTER` are both names for the storage engine that enables clustering support with MySQL. `NDB` is preferred, but either name is correct.

A.10.3 How many computers do I need to run an NDB Cluster, and why?

A minimum of three computers is required to run a viable cluster. However, the minimum *recommended* number of computers in an NDB Cluster is four: one each to run the management and SQL nodes, and two computers to serve as data nodes. The purpose of the two data nodes is to provide redundancy; the management node must run on a separate machine to guarantee continued arbitration services in the event that one of the data nodes fails.

To provide increased throughput and high availability, you should use multiple SQL nodes (MySQL Servers connected to the cluster). It is also possible (although not strictly necessary) to run multiple management servers.

A.11 MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets

This set of Frequently Asked Questions derives from the experience of MySQL's Support and Development groups in handling many inquiries about CJK (Chinese-Japanese-Korean) issues.

A.11.1 What CJK character sets are available in MySQL? 5071
A.11.2 I have inserted CJK characters into my table. Why does SELECT display them as “?” characters? 5072
A.11.3 What problems should I be aware of when working with the Big5 Chinese character set? 5074
A.11.4 Why do Japanese character set conversions fail? 5074
A.11.5 What should I do if I want to convert SJIS 81CA to cp932? 5075
A.11.6 How does MySQL represent the Yen (¥) sign? 5075
A.11.7 Of what issues should I be aware when working with Korean character sets in MySQL? 5076
A.11.8 Why do I get Incorrect string value error messages? 5076
A.11.9 Why does my GUI front end or browser display CJK characters incorrectly in my application using Access, PHP, or another API? 5077
A.11.10 I've upgraded to MySQL 8.0. How can I revert to behavior like that in MySQL 4.0 with regard to character sets? 5078
A.11.11 Why do some LIKE and FULLTEXT searches with CJK characters fail? 5079
A.11.12 How do I know whether character X is available in all character sets? 5079
A.11.13 Why do CJK strings sort incorrectly in Unicode? (I) 5080
A.11.14 Why do CJK strings sort incorrectly in Unicode? (II) 5081
A.11.15 Why are my supplementary characters rejected by MySQL? 5081
A.11.16 Should “CJK” be “CJKV”? 5081
A.11.17 Does MySQL permit CJK characters to be used in database and table names? 5081
A.11.18 Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean? ... 5081
A.11.19 Where can I get help with CJK and related issues in MySQL? 5081

A.11.1 What CJK character sets are available in MySQL?

The list of CJK character sets may vary depending on your MySQL version. For example, the gb18030 character set is not supported prior to MySQL 5.7.4. However, since the name of the applicable language appears in the DESCRIPTION column for every entry in the INFORMATION_SCHEMA.CHARACTER_SETS table, you can obtain a current list of all the non-Unicode CJK character sets using this query:

```
mysql> SELECT CHARACTER_SET_NAME, DESCRIPTION
       FROM INFORMATION_SCHEMA.CHARACTER_SETS
       WHERE DESCRIPTION LIKE '%Chin%'
       OR DESCRIPTION LIKE '%Japanese%'
       OR DESCRIPTION LIKE '%Korean%'
       ORDER BY CHARACTER_SET_NAME;
+--------------------+---------------------------------+
| CHARACTER_SET_NAME | DESCRIPTION                     |
+--------------------+---------------------------------+
| big5               | Big5 Traditional Chinese        |
| cp932              | SJIS for Windows Japanese       |
| eucjpms            | UJIS for Windows Japanese       |
| euckr              | EUC-KR Korean                   |
| gb18030            | China National Standard GB18030 |
```

```
| gb2312             | GB2312 Simplified Chinese       |
| gbk                | GBK Simplified Chinese          |
| sjis               | Shift-JIS Japanese              |
| ujis               | EUC-JP Japanese                 |
+--------------------+---------------------------------+
```

(For more information, see Section 25.2, “The INFORMATION_SCHEMA CHARACTER_SETS Table”.)

MySQL supports three variants of the *GB* (*Guojia Biaozhun*, or *National Standard*, or *Simplified Chinese*) character sets which are official in the People's Republic of China: `gb2312`, `gbk`, and (as of MySQL 5.7.4) `gb18030`.

Sometimes people try to insert `gbk` characters into `gb2312`, and it works most of the time because `gbk` is a superset of `gb2312`. But eventually they try to insert a rarer Chinese character and it does not work. (For an example, see Bug #16072).

Here, we try to clarify exactly what characters are legitimate in `gb2312` or `gbk`, with reference to the official documents. Please check these references before reporting `gb2312` or `gbk` bugs:

- The MySQL `gbk` character set is in reality “Microsoft code page 936”. This differs from the official `gbk` for characters `A1A4` (middle dot), `A1AA` (em dash), `A6E0-A6F5`, and `A8BB-A8C0`.
- For a listing of `gbk`/Unicode mappings, see http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP936.TXT.

It is also possible to store CJK characters in Unicode character sets, although the available collations may not sort characters quite as you expect:

- The `utf8` and `ucs2` character sets support the characters from Unicode Basic Multilingual Plane (BMP). These characters have code point values between `U+0000` and `U+FFFF`.
- The `utf8mb4`, `utf16`, `utf16le`, and `utf32` character sets support BMP characters, as well as supplementary characters that lie outside the BMP. Supplementary characters have code point values between `U+10000` and `U+10FFFF`.

The collation used for a Unicode character set determines the ability to sort (that is, distinguish) characters in the set:

- Collations based on Unicode Collation Algorithm (UCA) 4.0.0 distinguish only BMP characters.
- Collations based on UCA 5.2.0 or 9.0.0 distinguish BMP and supplementary characters.
- Non-UCA collations may not distinguish all Unicode characters. For example, the `utf8mb4` default collation is `utf8mb4_general_ci`, which distinguishes only BMP characters.

Moreover, distinguishing characters is not the same as ordering them per the conventions of a given CJK language. Currently, MySQL has only one CJK-specific UCA collation, `gb18030_unicode_520_ci` (which requires use of the non-Unicode `gb18030` character set).

For information about Unicode collations and their differentiating properties, including collation properties for supplementary characters, see Section 10.10.1, “Unicode Character Sets”.

A.11.2 I have inserted CJK characters into my table. Why does `SELECT` display them as “?” characters?

This problem is usually due to a setting in MySQL that does not match the settings for the application program or the operating system. Here are some common steps for correcting these types of issues:

- *Be certain of what MySQL version you are using*.

  Use the statement `SELECT VERSION();` to determine this.

- *Make sure that the database is actually using the desired character set*.

  People often think that the client character set is always the same as either the server character set or the character set used for display purposes. However, both of these are false assumptions. You can make sure by checking the result of `SHOW CREATE TABLE tablename` or, better yet, by using this statement:

```
SELECT character_set_name, collation_name
    FROM information_schema.columns
    WHERE table_schema = your_database_name
        AND table_name = your_table_name
        AND column_name = your_column_name;
```

- *Determine the hexadecimal value of the character or characters that are not being displayed correctly*.

  You can obtain this information for a column `column_name` in the table `table_name` using the following query:

```
SELECT HEX(column_name)
FROM table_name;
```

  `3F` is the encoding for the `?` character; this means that `?` is the character actually stored in the column. This most often happens because of a problem converting a particular character from your client character set to the target character set.

- *Make sure that a round trip is possible. When you select `literal` (or `_introducer hexadecimal-value`), do you obtain `literal` as a result*?

  For example, the Japanese Katakana character *Pe* (`ペ'`) exists in all CJK character sets, and has the code point value (hexadecimal coding) `0x30da`. To test a round trip for this character, use this query:

```
SELECT 'ペ' AS `ペ`;         /* or SELECT _ucs2 0x30da; */
```

  If the result is not also ペ, the round trip failed.

  For bug reports regarding such failures, we might ask you to follow up with `SELECT HEX('ペ');`. Then we can determine whether the client encoding is correct.

- *Make sure that the problem is not with the browser or other application, rather than with MySQL*.

  Use the `mysql` client program to accomplish this task. If `mysql` displays characters correctly but your application does not, your problem is probably due to system settings.

  To determine your settings, use the `SHOW VARIABLES` statement, whose output should resemble what is shown here:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+----------------------------------------+
```

```
| Variable_name            | Value                                  |
+--------------------------+----------------------------------------+
| character_set_client     | utf8                                   |
| character_set_connection | utf8                                   |
| character_set_database   | latin1                                 |
| character_set_filesystem | binary                                 |
| character_set_results    | utf8                                   |
| character_set_server     | latin1                                 |
| character_set_system     | utf8                                   |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+----------------------------------------+
```

These are typical character-set settings for an international-oriented client (notice the use of `utf8` Unicode) connected to a server in the West (`latin1` is a West Europe character set).

Although Unicode (usually the `utf8` variant on Unix, and the `ucs2` variant on Windows) is preferable to Latin, it is often not what your operating system utilities support best. Many Windows users find that a Microsoft character set, such as `cp932` for Japanese Windows, is suitable.

If you cannot control the server settings, and you have no idea what setting your underlying computer uses, try changing to a common character set for the country that you're in (`euckr` = Korea; `gb18030`, `gb2312` or `gbk` = People's Republic of China; `big5` = Taiwan; `sjis`, `ujis`, `cp932`, or `eucjpms` = Japan; `ucs2` or `utf8` = anywhere). Usually it is necessary to change only the client and connection and results settings. The `SET NAMES`. statement changes all three at once. For example:

```
SET NAMES 'big5';
```

Once the setting is correct, you can make it permanent by editing `my.cnf` or `my.ini`. For example you might add lines looking like these:

```
[mysqld]
character-set-server=big5
[client]
default-character-set=big5
```

It is also possible that there are issues with the API configuration setting being used in your application; see *Why does my GUI front end or browser not display CJK characters correctly...?* for more information.

A.11.3 What problems should I be aware of when working with the Big5 Chinese character set?

MySQL supports the Big5 character set which is common in Hong Kong and Taiwan (Republic of China). The MySQL `big5` character set is in reality Microsoft code page 950, which is very similar to the original `big5` character set.

A feature request for adding `HKSCS` extensions has been filed. People who need this extension may find the suggested patch for Bug #13577 to be of interest.

A.11.4 Why do Japanese character set conversions fail?

MySQL supports the `sjis`, `ujis`, `cp932`, and `eucjpms` character sets, as well as Unicode. A common need is to convert between character sets. For example, there might be a Unix server (typically with `sjis` or `ujis`) and a Windows client (typically with `cp932`).

In the following conversion table, the `ucs2` column represents the source, and the `sjis`, `cp932`, `ujis`, and `eucjpms` columns represent the destinations; that is, the last 4 columns provide the

hexadecimal result when we use `CONVERT(ucs2)` or we assign a `ucs2` column containing the value to an `sjis`, `cp932`, `ujis`, or `eucjpms` column.

Character Name	ucs2	sjis	cp932	ujis	eucjpms
BROKEN BAR	00A6	3F	3F	8FA2C3	3F
FULLWIDTH BROKEN BAR	FFE4	3F	FA55	3F	8FA2
YEN SIGN	00A5	3F	3F	20	3F
FULLWIDTH YEN SIGN	FFE5	818F	818F	A1EF	3F
TILDE	007E	7E	7E	7E	7E
OVERLINE	203E	3F	3F	20	3F
HORIZONTAL BAR	2015	815C	815C	A1BD	A1BD
EM DASH	2014	3F	3F	3F	3F
REVERSE SOLIDUS	005C	815F	5C	5C	5C
FULLWIDTH ""	FF3C	3F	815F	3F	A1C0
WAVE DASH	301C	8160	3F	A1C1	3F
FULLWIDTH TILDE	FF5E	3F	8160	3F	A1C1
DOUBLE VERTICAL LINE	2016	8161	3F	A1C2	3F
PARALLEL TO	2225	3F	8161	3F	A1C2
MINUS SIGN	2212	817C	3F	A1DD	3F
FULLWIDTH HYPHEN-MINUS	FF0D	3F	817C	3F	A1DD
CENT SIGN	00A2	8191	3F	A1F1	3F
FULLWIDTH CENT SIGN	FFE0	3F	8191	3F	A1F1
POUND SIGN	00A3	8192	3F	A1F2	3F
FULLWIDTH POUND SIGN	FFE1	3F	8192	3F	A1F2
NOT SIGN	00AC	81CA	3F	A2CC	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA	3F	A2CC

Now consider the following portion of the table.

	ucs2	sjis	cp932
NOT SIGN	00AC	81CA	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA

This means that MySQL converts the `NOT SIGN` (Unicode `U+00AC`) to `sjis` code point `0x81CA` and to `cp932` code point `3F`. (`3F` is the question mark (“?”. This is what is always used when the conversion cannot be performed.)

A.11.5. What should I do if I want to convert SJIS `81CA` to `cp932`?

Our answer is: “?”. There are disadvantages to this, and many people would prefer a “loose” conversion, so that `81CA (NOT SIGN)` in `sjis` becomes `81CA (FULLWIDTH NOT SIGN)` in `cp932`.

A.11.6. How does MySQL represent the Yen (¥) sign?

A problem arises because some versions of Japanese character sets (both `sjis` and `euc`) treat `5C` as a *reverse solidus* (`\`, also known as a backslash), whereas others treat it as a yen sign (¥).

MySQL follows only one version of the JIS (Japanese Industrial Standards) standard description. In MySQL, *`5C` is always the reverse solidus (`\`)*.

A.11.7. Of what issues should I be aware when working with Korean character sets in MySQL?

In theory, while there have been several versions of the `euckr` (*Extended Unix Code Korea*) character set, only one problem has been noted. We use the “ASCII” variant of EUC-KR, in which the code point `0x5c` is REVERSE SOLIDUS, that is `\`, instead of the “KS-Roman” variant of EUC-KR, in which the code point `0x5c` is `WON SIGN` (₩). This means that you cannot convert Unicode `U+20A9` to `euckr`:

```
mysql> SELECT
           CONVERT('₩' USING euckr) AS euckr,
           HEX(CONVERT('₩' USING euckr)) AS hexeuckr;
+-------+----------+
| euckr | hexeuckr |
+-------+----------+
| ?     | 3F       |
+-------+----------+
```

A.11.8. Why do I get `Incorrect string value` error messages?

To see the problem, create a table with one Unicode (`ucs2`) column and one Chinese (`gb2312`) column.

```
mysql> CREATE TABLE ch
       (ucs2 CHAR(3) CHARACTER SET ucs2,
       gb2312 CHAR(3) CHARACTER SET gb2312);
```

In nonstrict SQL mode, try to place the rare character 𠜎 in both columns.

```
mysql> SET sql_mode = '';
mysql> INSERT INTO ch VALUES ('A𠜎B','A𠜎B');
Query OK, 1 row affected, 1 warning (0.00 sec)
```

The `INSERT` produces a warning. Use the following statement to see what it is:

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1366
Message: Incorrect string value: '\xE6\xB1\x8CB' for column 'gb2312' at row 1
```

So it is a warning about the `gb2312` column only.

```
mysql> SELECT ucs2,HEX(ucs2),gb2312,HEX(gb2312) FROM ch;
+-------+--------------+--------+-------------+
| ucs2  | HEX(ucs2)    | gb2312 | HEX(gb2312) |
+-------+--------------+--------+-------------+
| A𠜎B | 00416C4C0042 | A?B    | 413F42      |
+-------+--------------+--------+-------------+
```

Several things need explanation here:

1. The 浏 character is not in the `gb2312` character set, as described earlier.

2. If you are using an old version of MySQL, you may see a different message.

3. A warning occurs rather than an error because MySQL is not set to use strict SQL mode. In nonstrict mode, MySQL tries to do what it can, to get the best fit, rather than give up. With strict SQL mode, the `Incorrect string value` message occurs as an error rather than a warning, and the `INSERT` fails.

A.11.9. Why does my GUI front end or browser display CJK characters incorrectly in my application using Access, PHP, or another API?

Obtain a direct connection to the server using the `mysql` client, and try the same query there. If `mysql` responds correctly, the trouble may be that your application interface requires initialization. Use `mysql` to tell you what character set or sets it uses with the statement `SHOW VARIABLES LIKE 'char%';`. If you are using Access, you are most likely connecting with Connector/ODBC. In this case, you should check Configuring Connector/ODBC. If, for example, you use `big5`, you would enter `SET NAMES 'big5'`. (In this case, no `;` character is required.) If you are using ASP, you might need to add `SET NAMES` in the code. Here is an example that has worked in the past:

```
<%
Session.CodePage=0
Dim strConnection
Dim Conn
strConnection="driver={MySQL ODBC 3.51 Driver};server=server;uid=username;" \
               & "pwd=password;database=database;stmt=SET NAMES 'big5';"
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open strConnection
%>
```

In much the same way, if you are using any character set other than `latin1` with Connector/NET, you must specify the character set in the connection string. See Connecting to MySQL Using Connector/NET, for more information.

If you are using PHP, try this:

```
<?php
  $link = new mysqli($host, $usr, $pwd, $db);

  if( mysqli_connect_errno() )
  {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
  }

  $link->query("SET NAMES 'utf8'");
?>
```

In this case, we used `SET NAMES` to change `character_set_client`, `character_set_connection`, and `character_set_results`.

Another issue often encountered in PHP applications has to do with assumptions made by the browser. Sometimes adding or changing a `<meta>` tag suffices to correct the problem: for example, to insure that the user agent interprets page content as `UTF-8`, include `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">` in the `<head>` section of the HTML page.

If you are using Connector/J, see Using Character Sets and Unicode.

A.11.10 I've upgraded to MySQL 8.0. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?

In MySQL Version 4.0, there was a single "global" character set for both server and client, and the decision as to which character to use was made by the server administrator. This changed starting with MySQL Version 4.1. What happens now is a "handshake", as described in Section 10.4, "Connection Character Sets and Collations":

> When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and `character_set_connection` system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

The effect of this is that you cannot control the client character set by starting `mysqld` with `--character-set-server=utf8`. However, some Asian customers prefer the MySQL 4.0 behavior. To make it possible to retain this behavior, we added a `mysqld` switch, `--character-set-client-handshake`, which can be turned off with `--skip-character-set-client-handshake`. If you start `mysqld` with `--skip-character-set-client-handshake`, then, when a client connects, it sends to the server the name of the character set that it wants to use. However, *the server ignores this request from the client*.

By way of example, suppose that your favorite server character set is `latin1`. Suppose further that the client uses `utf8` because this is what the client's operating system supports. Start the server with `latin1` as its default character set:

```
mysqld --character-set-server=latin1
```

And then start the client with the default character set `utf8`:

```
mysql --default-character-set=utf8
```

The resulting settings can be seen by viewing the output of `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+----------------------------------------+
| Variable_name            | Value                                  |
+--------------------------+----------------------------------------+
| character_set_client     | utf8                                   |
| character_set_connection | utf8                                   |
| character_set_database   | latin1                                 |
| character_set_filesystem | binary                                 |
| character_set_results    | utf8                                   |
| character_set_server     | latin1                                 |
| character_set_system     | utf8                                   |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+----------------------------------------+
```

Now stop the client, and stop the server using `mysqladmin`. Then start the server again, but this time tell it to skip the handshake like so:

```
mysqld --character-set-server=utf8 --skip-character-set-client-handshake
```

Start the client with `utf8` once again as the default character set, then display the resulting settings:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+----------------------------------------+
| Variable_name            | Value                                  |
+--------------------------+----------------------------------------+
| character_set_client     | latin1                                 |
| character_set_connection | latin1                                 |
| character_set_database   | latin1                                 |
| character_set_filesystem | binary                                 |
| character_set_results    | latin1                                 |
| character_set_server     | latin1                                 |
| character_set_system     | utf8                                   |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+----------------------------------------+
```

As you can see by comparing the differing results from `SHOW VARIABLES`, the server ignores the client's initial settings if the `--skip-character-set-client-handshake` option is used.

A.11.11. Why do some `LIKE` and `FULLTEXT` searches with CJK characters fail?

For `LIKE` searches, there is a very simple problem with binary string column types such as `BINARY` and `BLOB`: we must know where characters end. With multibyte character sets, different characters might have different octet lengths. For example, in `utf8`, `A` requires one byte but 𠜎 requires three bytes, as shown here:

```
+-------------------------+--------------------------+
| OCTET_LENGTH(_utf8 'A') | OCTET_LENGTH(_utf8 '𠜎') |
+-------------------------+--------------------------+
|                       1 |                        3 |
+-------------------------+--------------------------+
```

If we do not know where the first character in a string ends, we do not know where the second character begins, in which case even very simple searches such as `LIKE '_A%'` fail. The solution is to use a nonbinary string column type defined to have the proper CJK character set. For example: `mycol TEXT CHARACTER SET sjis`. Alternatively, convert to a CJK character set before comparing.

This is one reason why MySQL cannot permit encodings of nonexistent characters. If it is not strict about rejecting bad input, it has no way of knowing where characters end.

For `FULLTEXT` searches, we must know where words begin and end. With Western languages, this is rarely a problem because most (if not all) of these use an easy-to-identify word boundary: the space character. However, this is not usually the case with Asian writing. We could use arbitrary halfway measures, like assuming that all Han characters represent words, or (for Japanese) depending on changes from Katakana to Hiragana due to grammatical endings. However, the only sure solution requires a comprehensive word list, which means that we would have to include a dictionary in the server for each Asian language supported. This is simply not feasible.

A.11.12. How do I know whether character *X* is available in all character sets?

The majority of simplified Chinese and basic nonhalfwidth Japanese Kana characters appear in all CJK character sets. The following stored procedure accepts a `UCS-2` Unicode character, converts it to other character sets, and displays the results in hexadecimal.

```
DELIMITER //
```

```
CREATE PROCEDURE p_convert(ucs2_char CHAR(1) CHARACTER SET ucs2)
BEGIN

CREATE TABLE tj
             (ucs2 CHAR(1) character set ucs2,
              utf8 CHAR(1) character set utf8,
              big5 CHAR(1) character set big5,
              cp932 CHAR(1) character set cp932,
              eucjpms CHAR(1) character set eucjpms,
              euckr CHAR(1) character set euckr,
              gb2312 CHAR(1) character set gb2312,
              gbk CHAR(1) character set gbk,
              sjis CHAR(1) character set sjis,
              ujis CHAR(1) character set ujis);

INSERT INTO tj (ucs2) VALUES (ucs2_char);

UPDATE tj SET utf8=ucs2,
              big5=ucs2,
              cp932=ucs2,
              eucjpms=ucs2,
              euckr=ucs2,
              gb2312=ucs2,
              gbk=ucs2,
              sjis=ucs2,
              ujis=ucs2;

/* If there are conversion problems, UPDATE produces warnings. */

SELECT hex(ucs2) AS ucs2,
       hex(utf8) AS utf8,
       hex(big5) AS big5,
       hex(cp932) AS cp932,
       hex(eucjpms) AS eucjpms,
       hex(euckr) AS euckr,
       hex(gb2312) AS gb2312,
       hex(gbk) AS gbk,
       hex(sjis) AS sjis,
       hex(ujis) AS ujis
FROM tj;

DROP TABLE tj;

END//

DELIMITER ;
```

The input can be any single `ucs2` character, or it can be the code value (hexadecimal representation) of that character. For example, from Unicode's list of `ucs2` encodings and names (http://www.unicode.org/Public/UNIDATA/UnicodeData.txt), we know that the Katakana character *Pe* appears in all CJK character sets, and that its code value is `X'30DA'`. If we use this value as the argument to `p_convert()`, the result is as shown here:

```
mysql> CALL p_convert(X'30DA');
+------+--------+------+-------+---------+-------+--------+------+------+------+
| ucs2 | utf8   | big5 | cp932 | eucjpms | euckr | gb2312 | gbk  | sjis | ujis |
+------+--------+------+-------+---------+-------+--------+------+------+------+
| 30DA | E3839A | C772 | 8379  | A5DA    | ABDA  | A5DA   | A5DA | 8379 | A5DA |
+------+--------+------+-------+---------+-------+--------+------+------+------+
```

Since none of the column values is `3F` (that is, the question mark character, `?`), we know that every conversion worked.

A.11.13. Why do CJK strings sort incorrectly in Unicode? (I)

CJK sorting problems that occurred in older MySQL versions can be solved as of MySQL 8.0 by using the `utf8mb4` character set and the `utf8mb4_ja_0900_as_cs` collation.

A.11.14 Why do CJK strings sort incorrectly in Unicode? (II)

CJK sorting problems that occurred in older MySQL versions can be solved as of MySQL 8.0 by using the `utf8mb4` character set and the `utf8mb4_ja_0900_as_cs` collation.

A.11.15 Why are my supplementary characters rejected by MySQL?

Supplementary characters lie outside the Unicode *Basic Multilingual Plane / Plane 0*. BMP characters have code point values between `U+0000` and `U+FFFF`. Supplementary characters have code point values between `U+10000` and `U+10FFFF`.

To store supplementary characters, you must use a character set that permits them:

- The `utf8` and `ucs2` character sets support BMP characters only.

  The `utf8` character set permits only `UTF-8` characters that take up to three bytes. This has led to reports such as that found in Bug #12600, which we rejected as “not a bug”. With `utf8`, MySQL must truncate an input string when it encounters bytes that it does no understand. Otherwise, it is unknown how long the bad multibyte character is.

  One possible workaround is to use `ucs2` instead of `utf8`, in which case the “bad” characters are changed to question marks. However, no truncation takes place. You can also change the data type to `BLOB` or `BINARY`, which perform no validity checking.

- The `utf8mb4`, `utf16`, `utf16le`, and `utf32` character sets support BMP characters, as well as supplementary characters outside the BMP.

A.11.16 Should “CJK” be “CJKV”?

No. The term “CJKV” (*Chinese Japanese Korean Vietnamese*) refers to Vietnamese character sets which contain Han (originally Chinese) characters. MySQL supports the modern Vietnamese script with Western characters, but does not support the old Vietnamese script using Han characters.

As of MySQL 5.6, there are Vietnamese collations for Unicode character sets, as described in Section 10.10.1, “Unicode Character Sets”.

A.11.17 Does MySQL permit CJK characters to be used in database and table names?

Yes.

A.11.18 Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?

The Japanese translation of the MySQL 5.6 manual can be downloaded from https://dev.mysql.com/doc/.

A.11.19 Where can I get help with CJK and related issues in MySQL?

The following resources are available:

- A listing of MySQL user groups can be found at https://wikis.oracle.com/display/mysql/List+of+MySQL+User+Groups.
- View feature requests relating to character set issues at http://tinyurl.com/y6xcuf.
- Visit the MySQL Character Sets, Collation, Unicode Forum. http://forums.mysql.com/ also provides foreign-language forums.

A.12 MySQL 8.0 FAQ: Connectors & APIs

For common questions, issues, and answers relating to the MySQL Connectors and other APIs, see the following areas of the Manual:

- Section 28.7.29, “C API Common Issues”
- Common Problems with MySQL and PHP
- Connector/ODBC Notes and Tips
- Connector/NET Programming
- MySQL Connector/J 8.0 Developer Guide

A.13 MySQL 8.0 FAQ: C API, Connector/C, libmysql

Frequently asked questions about MySQL C API, Connector/C, and libmysql.

A.13.1 What is “MySQL Native C API”? What are typical benefits and use cases? 5082
A.13.2 What is “Connector/C”? 5082
A.13.3 Which version of libmysql should I use? 5082
A.13.4 What if I want to use the “NoSQL” X DevAPI? 5082
A.13.5 How to I download libmysql? 5082
A.13.6 Where is the documentation? 5083
A.13.7 How do I report bugs? 5083
A.13.8 Is it possible to compile the library myself? 5083

A.13.1 What is “MySQL Native C API”? What are typical benefits and use cases?

libmysql is a C-based API that you can use in C applications to connect with the MySQL database server. It is also itself used as the foundation for drivers for standard database APIs like ODBC, Perl's DBI, and Python's DB API.

A.13.2 What is “Connector/C”?

libmysql and Connector/C are synonyms these days. In the past there were separate packages for installing; now we default to the libmysql packaging alone.

A.13.3 Which version of libmysql should I use?

For MySQL 8.0, 5.7, 5.6, and 5.5, we recommend libmysql 8.0, also known as MySQL Connector/C 8.0.

A.13.4 What if I want to use the “NoSQL” X DevAPI?

For C-language and X DevApi Document Store for MySQL 8.0, we recommend MySQL Connector/C++. Connector/C++ 8.0 has compatible C headers. (This is not applicable to MySQL 5.7 or before.)

A.13.5 How to I download libmysql?

Download Connector/C 8.0 (libmysqlclient 8.0).

- Linux: The Client Utilities Package is available from the MySQL Community Server download page.
- Repos: The Client Utilities Package is available from the Yum, APT, SuSE repositories.
- Windows: The Client Utilities Package is available from Windows Installer.

A.13.6. Where is the documentation?

Online documentation:

- MySQL Connector/C Installation Instructions
- Documentation
- Change History

A.13.7. How do I report bugs?

Please report any bugs or inconsistencies you observe to our Bugs Database. Select the C API Client as shown.

A.13.8. Is it possible to compile the library myself?

Yes, you can download the libmysqlclient source code and compile it on our own. Here's an example:

```
$ git clone --depth 1 https://github.com/mysql/mysql-server
$ cd mysql-server
$ mkdir build
$ cd build
$ cmake .. -GNinja -DDOWNLOAD_BOOST=1
           -DWITH_BOOST=/tmp -DCMAKE_BUILD_TYPE=Release -DWITHOUT_SERVER=ON
           -DWITH_SSL=system $ ninja libmysqlclient.a
$ ls -la archive_output_directory/libmysqlclient.a
-rw-rw-r-- 1 kg kg 8,5M wrz 5 04:57 archive_output_directory/libmysqlclient.a
```

Note

This example uses https://ninja-build.org/ as a build system instead of make.

A.14 MySQL 8.0 FAQ: Replication

In the following section, we provide answers to questions that are most frequently asked about MySQL Replication.

A.14.1 Must the slave be connected to the master all the time? 5084
A.14.2 Must I enable networking on my master and slave to enable replication? 5084
A.14.3 How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave? 5084
A.14.4 How do I force the master to block updates until the slave catches up? 5084
A.14.5 What issues should I be aware of when setting up two-way replication? 5085
A.14.6 How can I use replication to improve performance of my system? 5085
A.14.7 What should I do to prepare client code in my own applications to use performance-enhancing replication? 5085
A.14.8 When and how much can MySQL replication improve the performance of my system? 5085
A.14.9 How can I use replication to provide redundancy or high availability? 5086
A.14.10 How do I tell whether a master server is using statement-based or row-based binary logging format? 5086
A.14.11 How do I tell a slave to use row-based replication? 5087
A.14.12 How do I prevent GRANT and REVOKE statements from replicating to slave machines? 5087
A.14.13 Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on OS X and Windows)? 5087

A.14.14 Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)? 5087

A.14.1 Must the slave be connected to the master all the time?

No, it does not. The slave can go down or stay disconnected for hours or even days, and then reconnect and catch up on updates. For example, you can set up a master/slave relationship over a dial-up link where the link is up only sporadically and for short periods of time. The implication of this is that, at any given time, the slave is not guaranteed to be in synchrony with the master unless you take some special measures.

To ensure that catchup can occur for a slave that has been disconnected, you must not remove binary log files from the master that contain information that has not yet been replicated to the slaves. Asynchronous replication can work only if the slave is able to continue reading the binary log from the point where it last read events.

A.14.2 Must I enable networking on my master and slave to enable replication?

Yes, networking must be enabled on the master and slave. If networking is not enabled, the slave cannot connect to the master and transfer the binary log. Check that the `skip-networking` option has not been enabled in the configuration file for either server.

A.14.3 How do I know how late a slave is compared to the master? In other words, how do I know the date of the last statement replicated by the slave?

Check the `Seconds_Behind_Master` column in the output from `SHOW SLAVE STATUS`. See Section 17.1.7.1, “Checking Replication Status”.

When the slave SQL thread executes an event read from the master, it modifies its own time to the event timestamp. (This is why `TIMESTAMP` is well replicated.) In the `Time` column in the output of `SHOW PROCESSLIST`, the number of seconds displayed for the slave SQL thread is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. You can use this to determine the date of the last replicated event. Note that if your slave has been disconnected from the master for one hour, and then reconnects, you may immediately see large `Time` values such as 3600 for the slave SQL thread in `SHOW PROCESSLIST`. This is because the slave is executing statements that are one hour old. See Section 17.2.2, “Replication Implementation Details”.

A.14.4 How do I force the master to block updates until the slave catches up?

Use the following procedure:

1. On the master, execute these statements:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

   Record the replication coordinates (the current binary log file name and position) from the output of the `SHOW` statement.

2. On the slave, issue the following statement, where the arguments to the `MASTER_POS_WAIT()` function are the replication coordinate values obtained in the previous step:

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_pos);
```

   The `SELECT` statement blocks until the slave reaches the specified log file and position. At that point, the slave is in synchrony with the master and the statement returns.

3. On the master, issue the following statement to enable the master to begin processing updates again:

```
mysql> UNLOCK TABLES;
```

A.14.5. What issues should I be aware of when setting up two-way replication?

MySQL replication currently does not support any locking protocol between master and slave to guarantee the atomicity of a distributed (cross-server) update. In other words, it is possible for client A to make an update to co-master 1, and in the meantime, before it propagates to co-master 2, client B could make an update to co-master 2 that makes the update of client A work differently than it did on co-master 1. Thus, when the update of client A makes it to co-master 2, it produces tables that are different from what you have on co-master 1, even after all the updates from co-master 2 have also propagated. This means that you should not chain two servers together in a two-way replication relationship unless you are sure that your updates can safely happen in any order, or unless you take care of mis-ordered updates somehow in the client code.

You should also realize that two-way replication actually does not improve performance very much (if at all) as far as updates are concerned. Each server must do the same number of updates, just as you would have a single server do. The only difference is that there is a little less lock contention because the updates originating on another server are serialized in one slave thread. Even this benefit might be offset by network delays.

A.14.6. How can I use replication to improve performance of my system?

Set up one server as the master and direct all writes to it. Then configure as many slaves as you have the budget and rackspace for, and distribute the reads among the master and the slaves. You can also start the slaves with the `--skip-innodb` option, enable the `low_priority_updates` system variable, and set the `delay_key_write` system variable to `ALL` to get speed improvements on the slave end. In this case, the slave uses nontransactional `MyISAM` tables instead of `InnoDB` tables to get more speed by eliminating transactional overhead.

A.14.7. What should I do to prepare client code in my own applications to use performance-enhancing replication?

See the guide to using replication as a scale-out solution, Section 17.4.5, “Using Replication for Scale-Out”.

A.14.8. When and how much can MySQL replication improve the performance of my system?

MySQL replication is most beneficial for a system that processes frequent reads and infrequent writes. In theory, by using a single-master/multiple-slave setup, you can scale the system by adding more slaves until you either run out of network bandwidth, or your update load grows to the point that the master cannot handle it.

To determine how many slaves you can use before the added benefits begin to level out, and how much you can improve performance of your site, you must know your query patterns, and determine empirically by benchmarking the relationship between the throughput for reads and writes on a typical master and a typical slave. The example here shows a rather simplified calculation of what you can get with replication for a hypothetical system. Let `reads` and `writes` denote the number of reads and writes per second, respectively.

Let's say that system load consists of 10% writes and 90% reads, and we have determined by benchmarking that `reads` is 1200 - 2 * `writes`. In other words, the system can do 1,200 reads per second with no writes, the average write is twice as slow as the average read, and the relationship

is linear. Suppose that the master and each slave have the same capacity, and that we have one master and *N* slaves. Then we have for each server (master or slave):

`reads` = 1200 - 2 * `writes`

`reads` = 9 * `writes` / (*N* + 1) (reads are split, but writes replicated to all slaves)

9 * `writes` / (*N* + 1) + 2 * `writes` = 1200

`writes` = 1200 / (2 + 9/(*N* + 1))

The last equation indicates the maximum number of writes for *N* slaves, given a maximum possible read rate of 1,200 per second and a ratio of nine reads per write.

This analysis yields the following conclusions:

- If *N* = 0 (which means we have no replication), our system can handle about 1200/11 = 109 writes per second.
- If *N* = 1, we get up to 184 writes per second.
- If *N* = 8, we get up to 400 writes per second.
- If *N* = 17, we get up to 480 writes per second.
- Eventually, as *N* approaches infinity (and our budget negative infinity), we can get very close to 600 writes per second, increasing system throughput about 5.5 times. However, with only eight servers, we increase it nearly four times.

These computations assume infinite network bandwidth and neglect several other factors that could be significant on your system. In many cases, you may not be able to perform a computation similar to the one just shown that accurately predicts what will happen on your system if you add *N* replication slaves. However, answering the following questions should help you decide whether and by how much replication will improve the performance of your system:

- What is the read/write ratio on your system?
- How much more write load can one server handle if you reduce the reads?
- For how many slaves do you have bandwidth available on your network?

A.14.9 How can I use replication to provide redundancy or high availability?

How you implement redundancy is entirely dependent on your application and circumstances. High-availability solutions (with automatic failover) require active monitoring and either custom scripts or third party tools to provide the failover support from the original MySQL server to the slave.

To handle the process manually, you should be able to switch from a failed master to a pre-configured slave by altering your application to talk to the new server or by adjusting the DNS for the MySQL server from the failed server to the new server.

For more information and some example solutions, see Section 17.4.8, “Switching Masters During Failover”.

A.14.10 How do I tell whether a master server is using statement-based or row-based binary logging format?

Check the value of the `binlog_format` system variable:

```
mysql> SHOW VARIABLES LIKE 'binlog_format';
```

The value shown will be one of `STATEMENT`, `ROW`, or `MIXED`. For `MIXED` mode, statement-based logging is used by default but replication switches automatically to row-based logging under certain conditions, such as unsafe statements. For information about when this may occur, see Section 5.4.4.3, “Mixed Binary Logging Format”.

A.14.11. How do I tell a slave to use row-based replication?

Slaves automatically know which format to use.

A.14.12. How do I prevent `GRANT` and `REVOKE` statements from replicating to slave machines?

Start the server with the `--replicate-wild-ignore-table=mysql.%` option to ignore replication for tables in the `mysql` database.

A.14.13. Does replication work on mixed operating systems (for example, the master runs on Linux while slaves run on OS X and Windows)?

Yes.

A.14.14. Does replication work on mixed hardware architectures (for example, the master runs on a 64-bit machine while slaves run on 32-bit machines)?

Yes.

A.15 MySQL 8.0 FAQ: MySQL Enterprise Thread Pool

A.15.1 What is the Thread Pool and what problem does it solve? 5087
A.15.2 How does the Thread Pool limit and manage concurrent sessions and transactions for optimal performance and throughput? 5087
A.15.3 How is the Thread Pool different from the client side Connection Pool? 5088
A.15.4 When should I use the Thread Pool? 5088
A.15.5 Are there recommended Thread Pool configurations? 5088

A.15.1. What is the Thread Pool and what problem does it solve?

The MySQL Thread Pool is a MySQL server plugin that extends the default connection-handling capabilities of the MySQL server to limit the number of concurrently executing statements/queries and transactions to ensure that each has sufficient CPU and memory resources to fulfill its task. For MySQL 8.0, the Thread Pool plugin is included in MySQL Enterprise Edition, a commercial product.

The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. The Thread Pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The Thread Pool plugin increases server performance by efficiently managing statement execution threads for large numbers of client connections, especially on modern multi-CPU/Core systems.

For more information, see Section 5.6.3, “MySQL Enterprise Thread Pool”.

A.15.2. How does the Thread Pool limit and manage concurrent sessions and transactions for optimal performance and throughput?

The Thread Pool uses a “divide and conquer” approach to limiting and balancing concurrency. Unlike the default connection handling of the MySQL Server, the Thread Pool separates

connections and threads, so there is no fixed relationship between connections and the threads that execute statements received from those connections. The Thread Pool then manages client connections within configurable thread groups, where they are prioritized and queued based on the nature of the work they were submitted to accomplish.

For more information, see Section 5.6.3.3, “Thread Pool Operation”.

A.15.3 How is the Thread Pool different from the client side Connection Pool?

The MySQL Connection Pool operates on the client side to ensure that a MySQL client does not constantly connect to and disconnect from the MySQL server. It is designed to cache idle connections in the MySQL client for use by other users as they are needed. This minimizes the overhead and expense of establishing and tearing down connections as queries are submitted to the MySQL server. The MySQL Connection Pool has no visibility as to the query handling capabilities or load of the back-end MySQL server. By contrast, the Thread Pool operates on the MySQL server side and is designed to manage the execution of inbound concurrent connections and queries as they are received from the client connections accessing the back-end MySQL database. Because of the separation of duties, the MySQL Connection Pool and Thread Pool are orthogonal and can be used independent of each other.

MySQL Connection Pooling via the MySQL Connectors is covered in Chapter 28, *Connectors and APIs*.

A.15.4 When should I use the Thread Pool?

There are a few rules of thumb to consider for optimal Thread Pool use cases:

The MySQL `Threads_running` variable keeps track of the number of concurrent statements currently executing in the MySQL Server. If this variable consistently exceeds a region where the server won't operate optimally (usually going beyond 40 for InnoDB workloads), the Thread Pool will be beneficial, especially in extreme parallel overload situations.

If you are using the `innodb_thread_concurrency` to limit the number of concurrently executing statements, you will find the Thread Pool solves the same problem, only better, by assigning connections to thread groups, then queuing executions based on transactional content, user defined designations, and so forth.

Lastly, if your workload comprises mainly short queries, the Thread Pool will be beneficial.

To learn more, see Section 5.6.3.4, “Thread Pool Tuning”.

A.15.5 Are there recommended Thread Pool configurations?

The Thread Pool has a number of user case driven configuration parameters that affect its performance. To learn about these and tips on tuning, see Section 5.6.3.4, “Thread Pool Tuning”.

A.16 MySQL 8.0 FAQ: InnoDB Change Buffer

A.16.1 What types of operations modify secondary indexes and result in change buffering? 5089
A.16.2 What is the benefit of the `InnoDB` change buffer? 5089
A.16.3 Does the change buffer support other types of indexes? 5089
A.16.4 How much space does `InnoDB` use for the change buffer? 5089
A.16.5 How do I determine the current size of the change buffer? 5089
A.16.6 When does change buffer merging occur? 5089
A.16.7 When is the change buffer flushed? 5090
A.16.8 When should the change buffer be used? 5090

A.16.9 When should the change buffer not be used? 5090
A.16.10 Where can I find additional information about the change buffer? 5090

A.16.1 What types of operations modify secondary indexes and result in change buffering?

`INSERT`, `UPDATE`, and `DELETE` operations can modify secondary indexes. If an affected index page is not in the buffer pool, the changes can be buffered in the change buffer.

A.16.2 What is the benefit of the `InnoDB` change buffer?

Buffering secondary index changes when secondary index pages are not in the buffer pool avoids expensive random access I/O operations that would be required to immediately read in affected index pages from disk. Buffered changes can be applied later, in batches, as pages are read into the buffer pool by other read operations.

A.16.3 Does the change buffer support other types of indexes?

No. The change buffer only supports secondary indexes. Clustered indexes, full-text indexes, and spatial indexes are not supported. Full-text indexes have their own caching mechanism.

A.16.4 How much space does `InnoDB` use for the change buffer?

Prior to the introduction of the `innodb_change_buffer_max_size` configuration option in MySQL 5.6, the maximum size of the on-disk change buffer in the system tablespace was 1/3 of the `InnoDB` buffer pool size.

In MySQL 5.6 and later, the `innodb_change_buffer_max_size` configuration option defines the maximum size of the change buffer as a percentage of the total buffer pool size. By default, `innodb_change_buffer_max_size` is set to 25. The maximum setting is 50.

`InnoDB` does not buffer an operation if it would cause the on-disk change buffer to exceed the defined limit.

Change buffer pages are not required to persist in the buffer pool and may be evicted by LRU operations.

A.16.5 How do I determine the current size of the change buffer?

The current size of the change buffer is reported by `SHOW ENGINE INNODB STATUS \G`, under the `INSERT BUFFER AND ADAPTIVE HASH INDEX` heading. For example:

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2, 0 merges
```

Relevant data points include:

- `size`: The number of pages used within the change buffer. Change buffer size is equal to `seg size - (1 + free list len)`. The `1 +` value represents the change buffer header page.
- `seg size`: The size of the change buffer, in pages.

For information about monitoring change buffer status, see Section 15.5.2, “Change Buffer”.

A.16.6 When does change buffer merging occur?

- When a page is read into the buffer pool, buffered changes are merged upon completion of the read, before the page is made available.

- Change buffer merging is performed as a background task. The `innodb_io_capacity` parameter sets an upper limit on the I/O activity performed by `InnoDB` background tasks such as merging data from the change buffer.
- A change buffer merge is performed during crash recovery. Changes are applied from the change buffer (in the system tablespace) to leaf pages of secondary indexes as index pages are read into the buffer pool.
- The change buffer is fully durable and will survive a system crash. Upon restart, change buffer merge operations resume as part of normal operations.
- A full merge of the change buffer can be forced as part of a slow server shutdown using `--innodb-fast-shutdown=0`.

A.16.7 When is the change buffer flushed?

Updated pages are flushed by the same flushing mechanism that flushes the other pages that occupy the buffer pool.

A.16.8 When should the change buffer be used?

The change buffer is a feature designed to reduce random I/O to secondary indexes as indexes grow larger and no longer fit in the `InnoDB` buffer pool. Generally, the change buffer should be used when the entire data set does not fit into the buffer pool, when there is substantial DML activity that modifies secondary index pages, or when there are lots of secondary indexes that are regularly changed by DML activity.

A.16.9 When should the change buffer not be used?

You might consider disabling the change buffer if the entire data set fits within the `InnoDB` buffer pool, if you have relatively few secondary indexes, or if you are using solid-state storage, where random reads are about as fast as sequential reads. Before making configuration changes, it is recommended that you run tests using a representative workload to determine if disabling the change buffer provides any benefit.

A.16.10 Where can I find additional information about the change buffer?

See Section 15.5.2, “Change Buffer”.

A.17 MySQL 8.0 FAQ: InnoDB Data-at-Rest Encryption

A.17.1 Is data decrypted for users who are authorized to see it? 5091
A.17.2 What is the overhead associated with `InnoDB` data-at-rest encryption? 5091
A.17.3 What are the encryption algorithms used with `InnoDB` data-at-rest encryption? 5091
A.17.4 Is it possible to use 3rd party encryption algorithms in place of the one provided by the `InnoDB` data-at-rest encryption feature? 5091
A.17.5 Can indexed columns be encrypted? 5091
A.17.6 What data types and data lengths does `InnoDB` data-at-rest encryption support? 5091
A.17.7 Does data remain encrypted on the network? 5091
A.17.8 Does database memory contain cleartext or encrypted data? 5091
A.17.9 How do I know which data to encrypt? 5091
A.17.10 How is `InnoDB` data-at-rest encryption different from encryption functions MySQL already provides? 5092
A.17.11 Does the transportable tablespaces feature work with `InnoDB` data-at-rest encryption? 5092
A.17.12 Does compression work with `InnoDB` data-at-rest encryption? 5092

A.17.13 Can I use mysqlpump or mysqldump with encrypted tables? 5092
A.17.14 How do I change (rotate, re-key) the master encryption key? 5092
A.17.15 How do I migrate data from a cleartext InnoDB tablespace to an encrypted InnoDB tablespace? 5092

A.17.1 Is data decrypted for users who are authorized to see it?

Yes. InnoDB data-at-rest encryption is designed to transparently apply encryption within the database without impacting existing applications. Returning data in encrypted format would break most existing applications. InnoDB data-at-rest encryption provides the benefit of encryption without the overhead associated with traditional database encryption solutions, which would typically require expensive and substantial changes to applications, database triggers, and views.

A.17.2 What is the overhead associated with InnoDB data-at-rest encryption?

There is no additional storage overhead. According to internal benchmarks, performance overhead amounts to a single digit percentage difference.

A.17.3 What are the encryption algorithms used with InnoDB data-at-rest encryption?

InnoDB data-at-rest encryption supports the Advanced Encryption Standard (AES256) block-based encryption algorithm. It uses Electronic Codebook (ECB) block encryption mode for tablespace key encryption and Cipher Block Chaining (CBC) block encryption mode for data encryption.

A.17.4 Is it possible to use 3rd party encryption algorithms in place of the one provided by the InnoDB data-at-rest encryption feature?

No, it is not possible to use other encryption algorithms. The provided encryption algorithm is broadly accepted.

A.17.5 Can indexed columns be encrypted?

InnoDB data-at-rest encryption supports all indexes transparently.

A.17.6 What data types and data lengths does InnoDB data-at-rest encryption support?

InnoDB data-at-rest encryption supports all supported data types. There is no data length limitation.

A.17.7 Does data remain encrypted on the network?

Data encrypted by the InnoDB data-at-rest feature is decrypted when it is read from the tablespace file. Thus, if the data is on the network, it is in cleartext form. However, data on the network can be encrypted using MySQL network encryption, which encrypts data traveling to and from a database using SSL/TLS.

A.17.8 Does database memory contain cleartext or encrypted data?

With InnoDB data-at-rest encryption, in-memory data is decrypted, which provides complete transparency.

A.17.9 How do I know which data to encrypt?

Compliance with the PCI-DSS standard requires that credit card numbers (Primary Account Number, or 'PAN') be stored in encrypted form. Breach Notification Laws (for example, CA SB 1386, CA AB 1950, and similar laws in 43+ more US states) require encryption of first name, last name, driver license number, and other PII data. In early 2008, CA AB 1298 added medical and health insurance information to PII data. Additionally, industry specific privacy and security standards may require encryption of certain assets. For example, assets such as pharmaceutical research results,

oil field exploration results, financial contracts, or personal data of law enforcement informants may require encryption. In the health care industry, the privacy of patient data, health records and X-ray images is of the highest importance.

A.17.10 How is `InnoDB` data-at-rest encryption different from encryption functions MySQL already provides?

There are symmetric and asymmetric encryption APIs in MySQL that can be used to manually encrypt data within the database. However, the application must manage encryption keys and perform required encryption and decryption operations by calling API functions. `InnoDB` data-at-rest encryption requires no application changes, is transparent to end users, and provides automated, built-in key management.

A.17.11 Does the transportable tablespaces feature work with `InnoDB` data-at-rest encryption?

Yes. It is supported for encrypted file-per-table tablespaces. For more information, see Exporting Encrypted Tablespaces.

A.17.12 Does compression work with `InnoDB` data-at-rest encryption?

Customers using `InnoDB` data-at-rest encryption receive the full benefit of compression because compression is applied before data blocks are encrypted.

A.17.13 Can I use `mysqlpump` or `mysqldump` with encrypted tables?

Yes. Because these utilities create logical backups, the data dumped from encrypted tables is not encrypted.

A.17.14 How do I change (rotate, re-key) the master encryption key?

`InnoDB` data-at-rest encryption uses a two tier key mechanism. When data-at-rest encryption is used, individual tablespace keys are stored in the header of the underlying tablespace data file. Tablespace keys are encrypted using the master encryption key. The master encryption key is generated when tablespace encryption is enabled, and is stored outside the database. The master encryption key is rotated using the `ALTER INSTANCE ROTATE INNODB MASTER KEY` statement, which generates a new master encryption key, stores the key, and rotates the key into use.

A.17.15 How do I migrate data from a cleartext `InnoDB` tablespace to an encrypted `InnoDB` tablespace?

Transferring data from one tablespace to another is not required. To encrypt data in an `InnoDB` file-per-table tablespace, run `ALTER TABLE tbl_name ENCRYPTION = 'Y'`. To encrypt a general tablespace or the `mysql` tablespace, run `ALTER TABLESPACE tablespace_name ENCRYPTION = 'Y'`. Encryption support for general tablespaces was introduced in MySQL 8.0.13. Encryption support for the `mysql` system tablespace is available as of MySQL 8.0.16.

A.18 MySQL 8.0 FAQ: Virtualization Support

A.18.1 Is MySQL supported on virtualized environments such as Oracle VM, VMWare, Docker, Microsoft Hyper-V, or others? 5092

A.18.1 Is MySQL supported on virtualized environments such as Oracle VM, VMWare, Docker, Microsoft Hyper-V, or others?

MySQL is supported on virtualized environments, but is certified only for Oracle VM. Contact Oracle Support for more information.

Be aware of potential problems when using virtualization software. The usual ones are related to performance, performance degradations, slowness, or unpredictability of disk, I/O, network, and memory.

Appendix B Errors, Error Codes, and Common Problems

Table of Contents

B.1 Error Message Sources and Components 5093
B.2 Error Information Interfaces 5096
B.3 Error Message Reference 5098
B.3.1 Server Error Message Reference 5098
B.3.2 Client Error Message Reference 5604
B.3.3 Global Error Message Reference 5609
B.4 Problems and Common Errors 5617
B.4.1 How to Determine What Is Causing a Problem 5617
B.4.2 Common Errors When Using MySQL Programs 5618
B.4.3 Administration-Related Issues 5630
B.4.4 Query-Related Issues 5639
B.4.5 Optimizer-Related Issues 5646
B.4.6 Table Definition-Related Issues 5647
B.4.7 Known Issues in MySQL 5648

This appendix describes the types of error information MySQL provides and how to obtain information about them. It also lists the error messages that the MySQL server and MySQL client programs generate. The final section is for troubleshooting. It describes common problems and errors that may occur and potential resolutions.

Additional Resources

Other sections that discuss error-related topics include:

- Information about configuring where and how the server writes the error log: Section 5.4.2, “The Error Log”
- Information about the character set used for error messages: Section 10.6, “Error Message Character Set”
- Information about the language used for error messages: Section 10.12, “Setting the Error Message Language”
- Information about errors related to `InnoDB`: Section 15.20.4, “InnoDB Error Handling”

B.1 Error Message Sources and Components

This section discusses how error messages originate within MySQL and the components they contain.

- Error Message Sources
- Error Message Components
- Error Code Ranges

Error Message Sources

Error messages can originate on the server side or the client side:

- On the server side, error messages may occur during the startup and shutdown processes, as a result of issues that occur during SQL statement execution, and so forth.
  - The MySQL server writes some error messages to its error log. These indicate issues of interest to database administrators or that require DBA action.
  - The server sends other error messages to client programs. These indicate issues pertaining only to a particular client. The MySQL client library takes errors received from the server and makes them available to the host client program.
- Client-side error messages are generated from within the MySQL client library, usually involving problems communicating with the server.

Example server-side error messages written to the error log:

- This message produced during the startup process provides a status or progress indicator:

```
2018-10-28T13:01:32.735983Z 0 [Note] [MY-010303] [Server] Skipping
generation of SSL certificates as options related to SSL are specified.
```

- This message indicates an issue that requires DBA action:

```
2018-10-02T03:20:39.410387Z 768 [ERROR] [MY-010045] [Server] Event Scheduler:
[evtuser@localhost][myschema.e_daily] Unknown database 'mydb'
```

Example server-side error message sent to client programs, as displayed by the `mysql` client:

```
mysql> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Example client-side error message originating from within the client library, as displayed by the `mysql` client:

```
shell> mysql -h no-such-host
ERROR 2005 (HY000): Unknown MySQL server host 'no-such-host' (0)
```

Whether an error originates from within the client library or is received from the server, a MySQL client program may respond in varying ways. As just illustrated, the client may display the error message so the user can take corrective measures. The client may instead internally attempt to resolve or retry a failed operation, or take other action.

Error Message Components

When an error occurs, error information includes several components: an error code, SQLSTATE value, and message string. These components have the following characteristics:

- Error code: This value is numeric. It is MySQL-specific and is not portable to other database systems.

  Each error number has a corresponding symbolic value. Examples:

  - The symbol for server error number `1146` is `ER_NO_SUCH_TABLE`.
  - The symbol for client error number `2005` is `CR_UNKNOWN_HOST`.

  The set of error codes used in error messages is partitioned into distinct ranges; see Error Code Ranges.

Error codes are stable across General Availability (GA) releases of a given MySQL series. Before a series reaches GA status, new codes may still be under development and are subject to change.

- SQLSTATE value: This value is a five-character string (for example, `'42S02'`). SQLSTATE values are taken from ANSI SQL and ODBC and are more standardized than the numeric error codes. The first two characters of an SQLSTATE value indicate the error class:

  - Class = `'00'` indicates success.
  - Class = `'01'` indicates a warning.
  - Class = `'02'` indicates “not found.” This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. This condition also occurs for `SELECT ... INTO var_list` statements that retrieve no rows.
  - Class > `'02'` indicates an exception.

  For server-side errors, not all MySQL error numbers have corresponding SQLSTATE values. In these cases, `'HY000'` (general error) is used.

  For client-side errors, the SQLSTATE value is always `'HY000'` (general error), so it is not meaningful for distinguishing one client error from another.

- Message string: This string provides a textual description of the error.

Error Code Ranges

The set of error codes used in error messages is partitioned into distinct ranges, each with its own purpose:

- 1 to 999: Global error codes. This error code range is called “global” because it is a shared range that is used by the server as well as by clients.

  When an error in this range originates on the server side, the server writes it to the error log, padding the error code with leading zeros to six digits and adding a prefix of `MY-`.

  When an error in this range originates on the client side, the client library makes it available to the client program with no zero-padding or prefix.

- 1,000 to 1,999: Server error codes reserved for messages sent to clients.
- 2,000 to 2,999: Client error codes reserved for use by the client library.
- 3,000 to 4,999: Server error codes reserved for messages sent to clients.
- 5,000 to 5,999: Error codes reserved for use by X Plugin for messages sent to clients.
- 10,000 to 49,999: Server error codes reserved for messages to be written to the error log (not sent to clients).

  When an error in this range occurs, the server writes it to the error log, padding the error code with leading zeros to six digits and adding a prefix of `MY-`.

- 50,000 to 51,999: Error codes reserved for use by third parties.

The server handles error messages written to the error log differently from error messages sent to clients:

- When the server writes a message to the error log, it pads the error code with leading zeros to six digits and adds a prefix of `MY-` (examples: `MY-000022`, `MY-010048`).
- When the server sends a message to a client program, it adds no zero-padding or prefix to the error code (examples: `1036`, `3013`).

B.2 Error Information Interfaces

Error messages can originate on the server side or the client side, and each error message includes an error code, SQLSTATE value, and message string, as described in Section B.1, “Error Message Sources and Components”. For lists of server-side, client-side, and global (shared between server and clients) errors, see Section B.3, “Error Message Reference”.

For error checking from within programs, use error code numbers or symbols, not error message strings. Message strings do not change often, but it is possible. Also, if the database administrator changes the language setting, that affects the language of message strings; see Section 10.12, “Setting the Error Message Language”.

Error information in MySQL is available in the server error log, at the SQL level, from within client programs, and at the command line.

- Error Log
- SQL Error Message Interface
- Client Error Message Interface
- Command-Line Error Message Interface

Error Log

On the server side, some messages are intended for the error log. For information about configuring where and how the server writes the log, see Section 5.4.2, “The Error Log”.

Other server error messages are intended to be sent to client programs and are available as described in Client Error Message Interface.

The range within which a particular error code lies determines whether the server writes an error message to the error log or sends it to clients. For information about these ranges, see Error Code Ranges.

SQL Error Message Interface

At the SQL level, there are several sources of error information in MySQL:

- SQL statement warning and error information is available through the `SHOW WARNINGS` and `SHOW ERRORS` statements. The `warning_count` system variable indicates the number of errors, warnings, and notes (with notes excluded if the `sql_notes` system variable is disabled). The `error_count` system variable indicates the number of errors. Its value excludes warnings and notes.
- The `GET DIAGNOSTICS` statement may be used to inspect the diagnostic information in the diagnostics area. See Section 13.6.7.3, “GET DIAGNOSTICS Syntax”.
- `SHOW SLAVE STATUS` statement output includes information about replication errors occurring on replication slave servers.
- `SHOW ENGINE INNODB STATUS` statement output includes information about the most recent foreign key error if a `CREATE TABLE` statement for an `InnoDB` table fails.

Client Error Message Interface

Client programs receive errors from two sources:

- Errors that originate on the client side from within the MySQL client library.
- Errors that originate on the server side and are sent to the client by the server. These are received within the client library, which makes them available to the host client program.

The range within which a particular error code lies determines whether it originated from within the client library or was received by the client from the server. For information about these ranges, see Error Code Ranges.

Regardless of whether an error originates from within the client library or is received from the server, a MySQL client program obtains the error code, SQLSTATE value, message string, and other related information by calling C API functions in the client library:

- `mysql_errno()` returns the MySQL error code.
- `mysql_sqlstate()` returns the SQLSTATE value.
- `mysql_error()` returns the message string.
- `mysql_stmt_errno()`, `mysql_stmt_sqlstate()`, and `mysql_stmt_error()` are the corresponding error functions for prepared statements.
- `mysql_warning_count()` returns the number of errors, warnings, and notes for the most recent statement.

For descriptions of the client library error functions, see Section 28.7, “MySQL C API”.

A MySQL client program may respond to an error in varying ways. The client may display the error message so the user can take corrective measures, internally attempt to resolve or retry a failed operation, or take other action. For example, (using the `mysql` client), a failure to connect to the server might result in this message:

```
shell> mysql -h no-such-host
ERROR 2005 (HY000): Unknown MySQL server host 'no-such-host' (0)
```

Command-Line Error Message Interface

The `perror` program provides information from the command line about error numbers. See Section 4.8.2, “`perror` — Display MySQL Error Message Information”.

```
shell> perror 1231
MySQL error code MY-001231 (ER_WRONG_VALUE_FOR_VAR): Variable '%-.64s'
can't be set to the value of '%-.200s'
```

For MySQL NDB Cluster errors, use `ndb_perror`. See Section 22.4.16, “`ndb_perror` — Obtain NDB Error Message Information”.

```
shell> ndb_perror 323
NDB error code 323: Invalid nodegroup id, nodegroup already existing:
Permanent error: Application error
```

B.3 Error Message Reference

This section provides a reference for the various types of error messages in MySQL:

- Error messages produced only by the server: Section B.3.1, “Server Error Message Reference”
- Error messages that originate only from within the MySQL client library: Section B.3.2, “Client Error Message Reference”
- Error messages that are shared between the server and the client library: Section B.3.3, “Global Error Message Reference”

B.3.1 Server Error Message Reference

The MySQL server writes some error messages to its error log, and sends others to client programs.

Example server-side error messages written to the error log:

```
2018-10-28T13:01:32.735983Z 0 [Note] [MY-010303] [Server] Skipping
generation of SSL certificates as options related to SSL are specified.

2018-10-02T03:20:39.410387Z 768 [ERROR] [MY-010045] [Server] Event Scheduler:
[evtuser@localhost][myschema.e_daily] Unknown database 'mydb'
```

Example server-side error message sent to client programs, as displayed by the `mysql` client:

```
mysql> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Each server error message includes an error code, SQLSTATE value, and message string, as described in Section B.1, “Error Message Sources and Components”. These components are available as described in Section B.2, “Error Information Interfaces”.

In addition to the errors in the following list, the server can also produce error messages that have error codes in the range from 1 to 999. See Section B.3.3, “Global Error Message Reference”

- Error number: `1002`; Symbol: `ER_NO`; SQLSTATE: `HY000`

  Message: NO

  Used in the construction of other messages.

- Error number: `1003`; Symbol: `ER_YES`; SQLSTATE: `HY000`

  Message: YES

  Used in the construction of other messages.

  Extended `EXPLAIN` format generates Note messages. `ER_YES` is used in the `Code` column for these messages in subsequent `SHOW WARNINGS` output.

- Error number: `1004`; Symbol: `ER_CANT_CREATE_FILE`; SQLSTATE: `HY000`

  Message: Can't create file '%s' (errno: %d - %s)

  Occurs for failure to create or copy a file needed for some operation.

  Possible causes: Permissions problem for source file; destination file already exists but is not writeable.

- Error number: 1005; Symbol: ER_CANT_CREATE_TABLE; SQLSTATE: HY000

  Message: Can't create table '%s' (errno: %d - %s)

  InnoDB reports this error when a table cannot be created. If the error message refers to error 150, table creation failed because a foreign key constraint was not correctly formed. If the error message refers to error −1, table creation probably failed because the table includes a column name that matched the name of an internal InnoDB table.

- Error number: 1006; Symbol: ER_CANT_CREATE_DB; SQLSTATE: HY000

  Message: Can't create database '%s' (errno: %d - %s)

- Error number: 1007; Symbol: ER_DB_CREATE_EXISTS; SQLSTATE: HY000

  Message: Can't create database '%s'; database exists

  An attempt to create a database failed because the database already exists.

  Drop the database first if you really want to replace an existing database, or add an IF NOT EXISTS clause to the CREATE DATABASE statement if to retain an existing database without having the statement produce an error.

- Error number: 1008; Symbol: ER_DB_DROP_EXISTS; SQLSTATE: HY000

  Message: Can't drop database '%s'; database doesn't exist

- Error number: 1010; Symbol: ER_DB_DROP_RMDIR; SQLSTATE: HY000

  Message: Error dropping database (can't rmdir '%s', errno: %d - %s)

- Error number: 1012; Symbol: ER_CANT_FIND_SYSTEM_REC; SQLSTATE: HY000

  Message: Can't read record in system table

  Returned by InnoDB for attempts to access InnoDB INFORMATION_SCHEMA tables when InnoDB is unavailable.

- Error number: 1013; Symbol: ER_CANT_GET_STAT; SQLSTATE: HY000

  Message: Can't get status of '%s' (errno: %d - %s)

- Error number: 1015; Symbol: ER_CANT_LOCK; SQLSTATE: HY000

  Message: Can't lock file (errno: %d - %s)

- Error number: 1016; Symbol: ER_CANT_OPEN_FILE; SQLSTATE: HY000

  Message: Can't open file: '%s' (errno: %d - %s)

  InnoDB reports this error when the table from the InnoDB data files cannot be found.

- Error number: 1017; Symbol: ER_FILE_NOT_FOUND; SQLSTATE: HY000

  Message: Can't find file: '%s' (errno: %d - %s)

- Error number: 1018; Symbol: ER_CANT_READ_DIR; SQLSTATE: HY000

  Message: Can't read dir of '%s' (errno: %d - %s)

- Error number: 1020; Symbol: ER_CHECKREAD; SQLSTATE: HY000

  Message: Record has changed since last read in table '%s'

- Error number: 1022; Symbol: ER_DUP_KEY; SQLSTATE: 23000

  Message: Can't write; duplicate key in table '%s'

- Error number: 1024; Symbol: ER_ERROR_ON_READ; SQLSTATE: HY000

  Message: Error reading file '%s' (errno: %d - %s)

- Error number: 1025; Symbol: ER_ERROR_ON_RENAME; SQLSTATE: HY000

  Message: Error on rename of '%s' to '%s' (errno: %d - %s)

- Error number: 1026; Symbol: ER_ERROR_ON_WRITE; SQLSTATE: HY000

  Message: Error writing file '%s' (errno: %d - %s)

- Error number: 1027; Symbol: ER_FILE_USED; SQLSTATE: HY000

  Message: '%s' is locked against change

- Error number: 1028; Symbol: ER_FILSORT_ABORT; SQLSTATE: HY000

  Message: Sort aborted

  ER_FILSORT_ABORT was removed after 8.0.18.

- Error number: 1030; Symbol: ER_GET_ERRNO; SQLSTATE: HY000

  Message: Got error %d - '%s' from storage engine

  Check the %d value to see what the OS error means. For example, 28 indicates that you have run out of disk space.

- Error number: 1031; Symbol: ER_ILLEGAL_HA; SQLSTATE: HY000

  Message: Table storage engine for '%s' doesn't have this option

- Error number: 1032; Symbol: ER_KEY_NOT_FOUND; SQLSTATE: HY000

  Message: Can't find record in '%s'

- Error number: 1033; Symbol: ER_NOT_FORM_FILE; SQLSTATE: HY000

  Message: Incorrect information in file: '%s'

- Error number: 1034; Symbol: ER_NOT_KEYFILE; SQLSTATE: HY000

  Message: Incorrect key file for table '%s'; try to repair it

- Error number: 1035; Symbol: ER_OLD_KEYFILE; SQLSTATE: HY000

  Message: Old key file for table '%s'; repair it!

- Error number: 1036; Symbol: ER_OPEN_AS_READONLY; SQLSTATE: HY000

  Message: Table '%s' is read only

- Error number: 1037; Symbol: ER_OUTOFMEMORY; SQLSTATE: HY001

  Message: Out of memory; restart server and try again (needed %d bytes)

- Error number: 1038; Symbol: ER_OUT_OF_SORTMEMORY; SQLSTATE: HY001

  Message: Out of sort memory, consider increasing server sort buffer size

- Error number: 1040; Symbol: ER_CON_COUNT_ERROR; SQLSTATE: 08004

  Message: Too many connections

- Error number: 1041; Symbol: ER_OUT_OF_RESOURCES; SQLSTATE: HY000

  Message: Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space

- Error number: 1042; Symbol: ER_BAD_HOST_ERROR; SQLSTATE: 08S01

  Message: Can't get hostname for your address

- Error number: 1043; Symbol: ER_HANDSHAKE_ERROR; SQLSTATE: 08S01

  Message: Bad handshake

- Error number: 1044; Symbol: ER_DBACCESS_DENIED_ERROR; SQLSTATE: 42000

  Message: Access denied for user '%s'@'%s' to database '%s'

- Error number: 1045; Symbol: ER_ACCESS_DENIED_ERROR; SQLSTATE: 28000

  Message: Access denied for user '%s'@'%s' (using password: %s)

- Error number: 1046; Symbol: ER_NO_DB_ERROR; SQLSTATE: 3D000

  Message: No database selected

- Error number: 1047; Symbol: ER_UNKNOWN_COM_ERROR; SQLSTATE: 08S01

  Message: Unknown command

- Error number: 1048; Symbol: ER_BAD_NULL_ERROR; SQLSTATE: 23000

  Message: Column '%s' cannot be null

- Error number: 1049; Symbol: ER_BAD_DB_ERROR; SQLSTATE: 42000

  Message: Unknown database '%s'

- Error number: 1050; Symbol: ER_TABLE_EXISTS_ERROR; SQLSTATE: 42S01

  Message: Table '%s' already exists

- Error number: 1051; Symbol: ER_BAD_TABLE_ERROR; SQLSTATE: 42S02

  Message: Unknown table '%s'

- Error number: 1052; Symbol: ER_NON_UNIQ_ERROR; SQLSTATE: 23000

  Message: Column '%s' in %s is ambiguous

```
%s = column name
%s = location of column (for example, "field list")
```

Likely cause: A column appears in a query without appropriate qualification, such as in a select list or ON clause.

Examples:

```
mysql> SELECT i FROM t INNER JOIN t AS t2;
ERROR 1052 (23000): Column 'i' in field list is ambiguous

mysql> SELECT * FROM t LEFT JOIN t AS t2 ON i = i;
ERROR 1052 (23000): Column 'i' in on clause is ambiguous
```

Resolution:

- Qualify the column with the appropriate table name:

  ```
  mysql> SELECT t2.i FROM t INNER JOIN t AS t2;
  ```

- Modify the query to avoid the need for qualification:

  ```
  mysql> SELECT * FROM t LEFT JOIN t AS t2 USING (i);
  ```

- Error number: 1053; Symbol: ER_SERVER_SHUTDOWN; SQLSTATE: 08S01

  Message: Server shutdown in progress

- Error number: 1054; Symbol: ER_BAD_FIELD_ERROR; SQLSTATE: 42S22

  Message: Unknown column '%s' in '%s'

- Error number: 1055; Symbol: ER_WRONG_FIELD_WITH_GROUP; SQLSTATE: 42000

  Message: '%s' isn't in GROUP BY

- Error number: 1056; Symbol: ER_WRONG_GROUP_FIELD; SQLSTATE: 42000

  Message: Can't group on '%s'

- Error number: 1057; Symbol: ER_WRONG_SUM_SELECT; SQLSTATE: 42000

  Message: Statement has sum functions and columns in same statement

- Error number: 1058; Symbol: ER_WRONG_VALUE_COUNT; SQLSTATE: 21S01

  Message: Column count doesn't match value count

- Error number: 1059; Symbol: ER_TOO_LONG_IDENT; SQLSTATE: 42000

  Message: Identifier name '%s' is too long

- Error number: 1060; Symbol: ER_DUP_FIELDNAME; SQLSTATE: 42S21

  Message: Duplicate column name '%s'

- Error number: 1061; Symbol: ER_DUP_KEYNAME; SQLSTATE: 42000

Message: Duplicate key name '%s'

- Error number: 1062; Symbol: ER_DUP_ENTRY; SQLSTATE: 23000

  Message: Duplicate entry '%s' for key %d

  The message returned with this error uses the format string for ER_DUP_ENTRY_WITH_KEY_NAME.

- Error number: 1063; Symbol: ER_WRONG_FIELD_SPEC; SQLSTATE: 42000

  Message: Incorrect column specifier for column '%s'

- Error number: 1064; Symbol: ER_PARSE_ERROR; SQLSTATE: 42000

  Message: %s near '%s' at line %d

- Error number: 1065; Symbol: ER_EMPTY_QUERY; SQLSTATE: 42000

  Message: Query was empty

- Error number: 1066; Symbol: ER_NONUNIQ_TABLE; SQLSTATE: 42000

  Message: Not unique table/alias: '%s'

- Error number: 1067; Symbol: ER_INVALID_DEFAULT; SQLSTATE: 42000

  Message: Invalid default value for '%s'

- Error number: 1068; Symbol: ER_MULTIPLE_PRI_KEY; SQLSTATE: 42000

  Message: Multiple primary key defined

- Error number: 1069; Symbol: ER_TOO_MANY_KEYS; SQLSTATE: 42000

  Message: Too many keys specified; max %d keys allowed

- Error number: 1070; Symbol: ER_TOO_MANY_KEY_PARTS; SQLSTATE: 42000

  Message: Too many key parts specified; max %d parts allowed

- Error number: 1071; Symbol: ER_TOO_LONG_KEY; SQLSTATE: 42000

  Message: Specified key was too long; max key length is %d bytes

- Error number: 1072; Symbol: ER_KEY_COLUMN_DOES_NOT_EXITS; SQLSTATE: 42000

  Message: Key column '%s' doesn't exist in table

- Error number: 1073; Symbol: ER_BLOB_USED_AS_KEY; SQLSTATE: 42000

  Message: BLOB column '%s' can't be used in key specification with the used table type

- Error number: 1074; Symbol: ER_TOO_BIG_FIELDLENGTH; SQLSTATE: 42000

  Message: Column length too big for column '%s' (max = %lu); use BLOB or TEXT instead

- Error number: 1075; Symbol: ER_WRONG_AUTO_KEY; SQLSTATE: 42000

  Message: Incorrect table definition; there can be only one auto column and it must be defined as a key

- Error number: 1076; Symbol: ER_READY; SQLSTATE: HY000

  Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d

- Error number: 1077; Symbol: ER_NORMAL_SHUTDOWN; SQLSTATE: HY000

  Message: %s: Normal shutdown

  ER_NORMAL_SHUTDOWN was removed after 8.0.4.

- Error number: 1079; Symbol: ER_SHUTDOWN_COMPLETE; SQLSTATE: HY000

  Message: %s: Shutdown complete

- Error number: 1080; Symbol: ER_FORCING_CLOSE; SQLSTATE: 08S01

  Message: %s: Forcing close of thread %ld user: '%s'

- Error number: 1081; Symbol: ER_IPSOCK_ERROR; SQLSTATE: 08S01

  Message: Can't create IP socket

- Error number: 1082; Symbol: ER_NO_SUCH_INDEX; SQLSTATE: 42S12

  Message: Table '%s' has no index like the one used in CREATE INDEX; recreate the table

- Error number: 1083; Symbol: ER_WRONG_FIELD_TERMINATORS; SQLSTATE: 42000

  Message: Field separator argument is not what is expected; check the manual

- Error number: 1084; Symbol: ER_BLOBS_AND_NO_TERMINATED; SQLSTATE: 42000

  Message: You can't use fixed rowlength with BLOBs; please use 'fields terminated by'

- Error number: 1085; Symbol: ER_TEXTFILE_NOT_READABLE; SQLSTATE: HY000

  Message: The file '%s' must be in the database directory or be readable by all

- Error number: 1086; Symbol: ER_FILE_EXISTS_ERROR; SQLSTATE: HY000

  Message: File '%s' already exists

- Error number: 1087; Symbol: ER_LOAD_INFO; SQLSTATE: HY000

  Message: Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld

- Error number: 1088; Symbol: ER_ALTER_INFO; SQLSTATE: HY000

  Message: Records: %ld Duplicates: %ld

- Error number: 1089; Symbol: ER_WRONG_SUB_KEY; SQLSTATE: HY000

  Message: Incorrect prefix key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique prefix keys

- Error number: 1090; Symbol: ER_CANT_REMOVE_ALL_FIELDS; SQLSTATE: 42000

  Message: You can't delete all columns with ALTER TABLE; use DROP TABLE instead

- Error number: 1091; Symbol: ER_CANT_DROP_FIELD_OR_KEY; SQLSTATE: 42000

Message: Can't DROP '%s'; check that column/key exists

- Error number: 1092; Symbol: ER_INSERT_INFO; SQLSTATE: HY000

  Message: Records: %ld Duplicates: %ld Warnings: %ld

- Error number: 1093; Symbol: ER_UPDATE_TABLE_USED; SQLSTATE: HY000

  Message: You can't specify target table '%s' for update in FROM clause

  This error occurs for attempts to select from and modify the same table within a single statement. If the select attempt occurs within a derived table, you can avoid this error by setting the derived_merge flag of the optimizer_switch system variable to force the subquery to be materialized into a temporary table, which effectively causes it to be a different table from the one modified. See Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”.

- Error number: 1094; Symbol: ER_NO_SUCH_THREAD; SQLSTATE: HY000

  Message: Unknown thread id: %lu

- Error number: 1095; Symbol: ER_KILL_DENIED_ERROR; SQLSTATE: HY000

  Message: You are not owner of thread %lu

- Error number: 1096; Symbol: ER_NO_TABLES_USED; SQLSTATE: HY000

  Message: No tables used

- Error number: 1097; Symbol: ER_TOO_BIG_SET; SQLSTATE: HY000

  Message: Too many strings for column %s and SET

- Error number: 1098; Symbol: ER_NO_UNIQUE_LOGFILE; SQLSTATE: HY000

  Message: Can't generate a unique log-filename %s.(1-999)

- Error number: 1099; Symbol: ER_TABLE_NOT_LOCKED_FOR_WRITE; SQLSTATE: HY000

  Message: Table '%s' was locked with a READ lock and can't be updated

- Error number: 1100; Symbol: ER_TABLE_NOT_LOCKED; SQLSTATE: HY000

  Message: Table '%s' was not locked with LOCK TABLES

- Error number: 1101; Symbol: ER_BLOB_CANT_HAVE_DEFAULT; SQLSTATE: 42000

  Message: BLOB, TEXT, GEOMETRY or JSON column '%s' can't have a default value

- Error number: 1102; Symbol: ER_WRONG_DB_NAME; SQLSTATE: 42000

  Message: Incorrect database name '%s'

- Error number: 1103; Symbol: ER_WRONG_TABLE_NAME; SQLSTATE: 42000

  Message: Incorrect table name '%s'

- Error number: 1104; Symbol: ER_TOO_BIG_SELECT; SQLSTATE: 42000

Message: The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET MAX_JOIN_SIZE=# if the SELECT is okay

- Error number: 1105; Symbol: ER_UNKNOWN_ERROR; SQLSTATE: HY000

  Message: Unknown error

- Error number: 1106; Symbol: ER_UNKNOWN_PROCEDURE; SQLSTATE: 42000

  Message: Unknown procedure '%s'

- Error number: 1107; Symbol: ER_WRONG_PARAMCOUNT_TO_PROCEDURE; SQLSTATE: 42000

  Message: Incorrect parameter count to procedure '%s'

- Error number: 1108; Symbol: ER_WRONG_PARAMETERS_TO_PROCEDURE; SQLSTATE: HY000

  Message: Incorrect parameters to procedure '%s'

- Error number: 1109; Symbol: ER_UNKNOWN_TABLE; SQLSTATE: 42S02

  Message: Unknown table '%s' in %s

- Error number: 1110; Symbol: ER_FIELD_SPECIFIED_TWICE; SQLSTATE: 42000

  Message: Column '%s' specified twice

- Error number: 1111; Symbol: ER_INVALID_GROUP_FUNC_USE; SQLSTATE: HY000

  Message: Invalid use of group function

- Error number: 1112; Symbol: ER_UNSUPPORTED_EXTENSION; SQLSTATE: 42000

  Message: Table '%s' uses an extension that doesn't exist in this MySQL version

- Error number: 1113; Symbol: ER_TABLE_MUST_HAVE_COLUMNS; SQLSTATE: 42000

  Message: A table must have at least 1 column

- Error number: 1114; Symbol: ER_RECORD_FILE_FULL; SQLSTATE: HY000

  Message: The table '%s' is full

  InnoDB reports this error when the system tablespace runs out of free space. Reconfigure the system tablespace to add a new data file.

- Error number: 1115; Symbol: ER_UNKNOWN_CHARACTER_SET; SQLSTATE: 42000

  Message: Unknown character set: '%s'

- Error number: 1116; Symbol: ER_TOO_MANY_TABLES; SQLSTATE: HY000

  Message: Too many tables; MySQL can only use %d tables in a join

- Error number: 1117; Symbol: ER_TOO_MANY_FIELDS; SQLSTATE: HY000

  Message: Too many columns

- Error number: 1118; Symbol: ER_TOO_BIG_ROWSIZE; SQLSTATE: 42000

Message: Row size too large. The maximum row size for the used table type, not counting BLOBs, is %ld. This includes storage overhead, check the manual. You have to change some columns to TEXT or BLOBs

- Error number: 1119; Symbol: ER_STACK_OVERRUN; SQLSTATE: HY000

  Message: Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld --thread_stack=#' to specify a bigger stack if needed

- Error number: 1120; Symbol: ER_WRONG_OUTER_JOIN; SQLSTATE: 42000

  Message: Cross dependency found in OUTER JOIN; examine your ON conditions

  ER_WRONG_OUTER_JOIN was removed after 8.0.0.

- Error number: 1120; Symbol: ER_WRONG_OUTER_JOIN_UNUSED; SQLSTATE: 42000

  Message: Cross dependency found in OUTER JOIN; examine your ON conditions

  ER_WRONG_OUTER_JOIN_UNUSED was added in 8.0.1.

- Error number: 1121; Symbol: ER_NULL_COLUMN_IN_INDEX; SQLSTATE: 42000

  Message: Table handler doesn't support NULL in given index. Please change column '%s' to be NOT NULL or use another handler

- Error number: 1122; Symbol: ER_CANT_FIND_UDF; SQLSTATE: HY000

  Message: Can't load function '%s'

- Error number: 1123; Symbol: ER_CANT_INITIALIZE_UDF; SQLSTATE: HY000

  Message: Can't initialize function '%s'; %s

- Error number: 1124; Symbol: ER_UDF_NO_PATHS; SQLSTATE: HY000

  Message: No paths allowed for shared library

- Error number: 1125; Symbol: ER_UDF_EXISTS; SQLSTATE: HY000

  Message: Function '%s' already exists

- Error number: 1126; Symbol: ER_CANT_OPEN_LIBRARY; SQLSTATE: HY000

  Message: Can't open shared library '%s' (errno: %d %s)

- Error number: 1127; Symbol: ER_CANT_FIND_DL_ENTRY; SQLSTATE: HY000

  Message: Can't find symbol '%s' in library

- Error number: 1128; Symbol: ER_FUNCTION_NOT_DEFINED; SQLSTATE: HY000

  Message: Function '%s' is not defined

- Error number: 1129; Symbol: ER_HOST_IS_BLOCKED; SQLSTATE: HY000

  Message: Host '%s' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'

- Error number: 1130; Symbol: ER_HOST_NOT_PRIVILEGED; SQLSTATE: HY000

  Message: Host '%s' is not allowed to connect to this MySQL server

- Error number: 1131; Symbol: ER_PASSWORD_ANONYMOUS_USER; SQLSTATE: 42000

  Message: You are using MySQL as an anonymous user and anonymous users are not allowed to change passwords

- Error number: 1132; Symbol: ER_PASSWORD_NOT_ALLOWED; SQLSTATE: 42000

  Message: You must have privileges to update tables in the mysql database to be able to change passwords for others

- Error number: 1133; Symbol: ER_PASSWORD_NO_MATCH; SQLSTATE: 42000

  Message: Can't find any matching row in the user table

- Error number: 1134; Symbol: ER_UPDATE_INFO; SQLSTATE: HY000

  Message: Rows matched: %ld Changed: %ld Warnings: %ld

- Error number: 1135; Symbol: ER_CANT_CREATE_THREAD; SQLSTATE: HY000

  Message: Can't create a new thread (errno %d); if you are not out of available memory, you can consult the manual for a possible OS-dependent bug

- Error number: 1136; Symbol: ER_WRONG_VALUE_COUNT_ON_ROW; SQLSTATE: 21S01

  Message: Column count doesn't match value count at row %ld

- Error number: 1137; Symbol: ER_CANT_REOPEN_TABLE; SQLSTATE: HY000

  Message: Can't reopen table: '%s'

- Error number: 1138; Symbol: ER_INVALID_USE_OF_NULL; SQLSTATE: 22004

  Message: Invalid use of NULL value

- Error number: 1139; Symbol: ER_REGEXP_ERROR; SQLSTATE: 42000

  Message: Got error '%s' from regexp

- Error number: 1140; Symbol: ER_MIX_OF_GROUP_FUNC_AND_FIELDS; SQLSTATE: 42000

  Message: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause

- Error number: 1141; Symbol: ER_NONEXISTING_GRANT; SQLSTATE: 42000

  Message: There is no such grant defined for user '%s' on host '%s'

- Error number: 1142; Symbol: ER_TABLEACCESS_DENIED_ERROR; SQLSTATE: 42000

  Message: %s command denied to user '%s'@'%s' for table '%s'

- Error number: 1143; Symbol: ER_COLUMNACCESS_DENIED_ERROR; SQLSTATE: 42000

  Message: %s command denied to user '%s'@'%s' for column '%s' in table '%s'

- Error number: 1144; Symbol: ER_ILLEGAL_GRANT_FOR_TABLE; SQLSTATE: 42000

  Message: Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used

- Error number: 1145; Symbol: ER_GRANT_WRONG_HOST_OR_USER; SQLSTATE: 42000

  Message: The host or user argument to GRANT is too long

- Error number: 1146; Symbol: ER_NO_SUCH_TABLE; SQLSTATE: 42S02

  Message: Table '%s.%s' doesn't exist

- Error number: 1147; Symbol: ER_NONEXISTING_TABLE_GRANT; SQLSTATE: 42000

  Message: There is no such grant defined for user '%s' on host '%s' on table '%s'

- Error number: 1148; Symbol: ER_NOT_ALLOWED_COMMAND; SQLSTATE: 42000

  Message: The used command is not allowed with this MySQL version

- Error number: 1149; Symbol: ER_SYNTAX_ERROR; SQLSTATE: 42000

  Message: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use

- Error number: 1152; Symbol: ER_ABORTING_CONNECTION; SQLSTATE: 08S01

  Message: Aborted connection %ld to db: '%s' user: '%s' (%s)

- Error number: 1153; Symbol: ER_NET_PACKET_TOO_LARGE; SQLSTATE: 08S01

  Message: Got a packet bigger than 'max_allowed_packet' bytes

- Error number: 1154; Symbol: ER_NET_READ_ERROR_FROM_PIPE; SQLSTATE: 08S01

  Message: Got a read error from the connection pipe

- Error number: 1155; Symbol: ER_NET_FCNTL_ERROR; SQLSTATE: 08S01

  Message: Got an error from fcntl()

- Error number: 1156; Symbol: ER_NET_PACKETS_OUT_OF_ORDER; SQLSTATE: 08S01

  Message: Got packets out of order

- Error number: 1157; Symbol: ER_NET_UNCOMPRESS_ERROR; SQLSTATE: 08S01

  Message: Couldn't uncompress communication packet

- Error number: 1158; Symbol: ER_NET_READ_ERROR; SQLSTATE: 08S01

  Message: Got an error reading communication packets

- Error number: 1159; Symbol: ER_NET_READ_INTERRUPTED; SQLSTATE: 08S01

  Message: Got timeout reading communication packets

- Error number: 1160; Symbol: ER_NET_ERROR_ON_WRITE; SQLSTATE: 08S01

Message: Got an error writing communication packets

- Error number: 1161; Symbol: ER_NET_WRITE_INTERRUPTED; SQLSTATE: 08S01

  Message: Got timeout writing communication packets

- Error number: 1162; Symbol: ER_TOO_LONG_STRING; SQLSTATE: 42000

  Message: Result string is longer than 'max_allowed_packet' bytes

- Error number: 1163; Symbol: ER_TABLE_CANT_HANDLE_BLOB; SQLSTATE: 42000

  Message: The used table type doesn't support BLOB/TEXT columns

- Error number: 1164; Symbol: ER_TABLE_CANT_HANDLE_AUTO_INCREMENT; SQLSTATE: 42000

  Message: The used table type doesn't support AUTO_INCREMENT columns

- Error number: 1166; Symbol: ER_WRONG_COLUMN_NAME; SQLSTATE: 42000

  Message: Incorrect column name '%s'

- Error number: 1167; Symbol: ER_WRONG_KEY_COLUMN; SQLSTATE: 42000

  Message: The used storage engine can't index column '%s'

- Error number: 1168; Symbol: ER_WRONG_MRG_TABLE; SQLSTATE: HY000

  Message: Unable to open underlying table which is differently defined or of non-MyISAM type or doesn't exist

- Error number: 1169; Symbol: ER_DUP_UNIQUE; SQLSTATE: 23000

  Message: Can't write, because of unique constraint, to table '%s'

- Error number: 1170; Symbol: ER_BLOB_KEY_WITHOUT_LENGTH; SQLSTATE: 42000

  Message: BLOB/TEXT column '%s' used in key specification without a key length

- Error number: 1171; Symbol: ER_PRIMARY_CANT_HAVE_NULL; SQLSTATE: 42000

  Message: All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead

- Error number: 1172; Symbol: ER_TOO_MANY_ROWS; SQLSTATE: 42000

  Message: Result consisted of more than one row

- Error number: 1173; Symbol: ER_REQUIRES_PRIMARY_KEY; SQLSTATE: 42000

  Message: This table type requires a primary key

- Error number: 1175; Symbol: ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE; SQLSTATE: HY000

  Message: You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column. %s

- Error number: 1176; Symbol: ER_KEY_DOES_NOT_EXITS; SQLSTATE: 42000

Message: Key '%s' doesn't exist in table '%s'

- Error number: 1177; Symbol: ER_CHECK_NO_SUCH_TABLE; SQLSTATE: 42000

  Message: Can't open table

- Error number: 1178; Symbol: ER_CHECK_NOT_IMPLEMENTED; SQLSTATE: 42000

  Message: The storage engine for the table doesn't support %s

- Error number: 1179; Symbol: ER_CANT_DO_THIS_DURING_AN_TRANSACTION; SQLSTATE: 25000

  Message: You are not allowed to execute this command in a transaction

- Error number: 1180; Symbol: ER_ERROR_DURING_COMMIT; SQLSTATE: HY000

  Message: Got error %d - '%s' during COMMIT

- Error number: 1181; Symbol: ER_ERROR_DURING_ROLLBACK; SQLSTATE: HY000

  Message: Got error %d - '%s' during ROLLBACK

- Error number: 1182; Symbol: ER_ERROR_DURING_FLUSH_LOGS; SQLSTATE: HY000

  Message: Got error %d during FLUSH_LOGS

- Error number: 1184; Symbol: ER_NEW_ABORTING_CONNECTION; SQLSTATE: 08S01

  Message: Aborted connection %u to db: '%s' user: '%s' host: '%s' (%s)

- Error number: 1188; Symbol: ER_MASTER; SQLSTATE: HY000

  Message: Error from master: '%s'

- Error number: 1189; Symbol: ER_MASTER_NET_READ; SQLSTATE: 08S01

  Message: Net error reading from master

- Error number: 1190; Symbol: ER_MASTER_NET_WRITE; SQLSTATE: 08S01

  Message: Net error writing to master

- Error number: 1191; Symbol: ER_FT_MATCHING_KEY_NOT_FOUND; SQLSTATE: HY000

  Message: Can't find FULLTEXT index matching the column list

- Error number: 1192; Symbol: ER_LOCK_OR_ACTIVE_TRANSACTION; SQLSTATE: HY000

  Message: Can't execute the given command because you have active locked tables or an active transaction

- Error number: 1193; Symbol: ER_UNKNOWN_SYSTEM_VARIABLE; SQLSTATE: HY000

  Message: Unknown system variable '%s'

- Error number: 1194; Symbol: ER_CRASHED_ON_USAGE; SQLSTATE: HY000

  Message: Table '%s' is marked as crashed and should be repaired

- Error number: 1195; Symbol: ER_CRASHED_ON_REPAIR; SQLSTATE: HY000

Message: Table '%s' is marked as crashed and last (automatic?) repair failed

- Error number: 1196; Symbol: ER_WARNING_NOT_COMPLETE_ROLLBACK; SQLSTATE: HY000

  Message: Some non-transactional changed tables couldn't be rolled back

- Error number: 1197; Symbol: ER_TRANS_CACHE_FULL; SQLSTATE: HY000

  Message: Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again

- Error number: 1199; Symbol: ER_SLAVE_NOT_RUNNING; SQLSTATE: HY000

  Message: This operation requires a running slave; configure slave and do START SLAVE

- Error number: 1200; Symbol: ER_BAD_SLAVE; SQLSTATE: HY000

  Message: The server is not configured as slave; fix in config file or with CHANGE MASTER TO

- Error number: 1201; Symbol: ER_MASTER_INFO; SQLSTATE: HY000

  Message: Could not initialize master info structure; more error messages can be found in the MySQL error log

- Error number: 1202; Symbol: ER_SLAVE_THREAD; SQLSTATE: HY000

  Message: Could not create slave thread; check system resources

- Error number: 1203; Symbol: ER_TOO_MANY_USER_CONNECTIONS; SQLSTATE: 42000

  Message: User %s already has more than 'max_user_connections' active connections

- Error number: 1204; Symbol: ER_SET_CONSTANTS_ONLY; SQLSTATE: HY000

  Message: You may only use constant expressions with SET

- Error number: 1205; Symbol: ER_LOCK_WAIT_TIMEOUT; SQLSTATE: HY000

  Message: Lock wait timeout exceeded; try restarting transaction

  InnoDB reports this error when lock wait timeout expires. The statement that waited too long was rolled back (not the entire transaction). You can increase the value of the innodb_lock_wait_timeout configuration option if SQL statements should wait longer for other transactions to complete, or decrease it if too many long-running transactions are causing locking problems and reducing concurrency on a busy system.

- Error number: 1206; Symbol: ER_LOCK_TABLE_FULL; SQLSTATE: HY000

  Message: The total number of locks exceeds the lock table size

  InnoDB reports this error when the total number of locks exceeds the amount of memory devoted to managing locks. To avoid this error, increase the value of innodb_buffer_pool_size. Within an individual application, a workaround may be to break a large operation into smaller pieces. For example, if the error occurs for a large INSERT, perform several smaller INSERT operations.

- Error number: 1207; Symbol: ER_READ_ONLY_TRANSACTION; SQLSTATE: 25000

  Message: Update locks cannot be acquired during a READ UNCOMMITTED transaction

- Error number: 1210; Symbol: ER_WRONG_ARGUMENTS; SQLSTATE: HY000

  Message: Incorrect arguments to %s

- Error number: 1211; Symbol: ER_NO_PERMISSION_TO_CREATE_USER; SQLSTATE: 42000

  Message: '%s'@'%s' is not allowed to create new users

- Error number: 1213; Symbol: ER_LOCK_DEADLOCK; SQLSTATE: 40001

  Message: Deadlock found when trying to get lock; try restarting transaction

  InnoDB reports this error when a transaction encounters a deadlock and is automatically rolled back so that your application can take corrective action. To recover from this error, run all the operations in this transaction again. A deadlock occurs when requests for locks arrive in inconsistent order between transactions. The transaction that was rolled back released all its locks, and the other transaction can now get all the locks it requested. Thus, when you re-run the transaction that was rolled back, it might have to wait for other transactions to complete, but typically the deadlock does not recur. If you encounter frequent deadlocks, make the sequence of locking operations (LOCK TABLES, SELECT ... FOR UPDATE, and so on) consistent between the different transactions or applications that experience the issue. See Section 15.7.5, “Deadlocks in InnoDB” for details.

- Error number: 1214; Symbol: ER_TABLE_CANT_HANDLE_FT; SQLSTATE: HY000

  Message: The used table type doesn't support FULLTEXT indexes

- Error number: 1215; Symbol: ER_CANNOT_ADD_FOREIGN; SQLSTATE: HY000

  Message: Cannot add foreign key constraint

- Error number: 1216; Symbol: ER_NO_REFERENCED_ROW; SQLSTATE: 23000

  Message: Cannot add or update a child row: a foreign key constraint fails

  InnoDB reports this error when you try to add a row but there is no parent row, and a foreign key constraint fails. Add the parent row first.

- Error number: 1217; Symbol: ER_ROW_IS_REFERENCED; SQLSTATE: 23000

  Message: Cannot delete or update a parent row: a foreign key constraint fails

  InnoDB reports this error when you try to delete a parent row that has children, and a foreign key constraint fails. Delete the children first.

- Error number: 1218; Symbol: ER_CONNECT_TO_MASTER; SQLSTATE: 08S01

  Message: Error connecting to master: %s

- Error number: 1220; Symbol: ER_ERROR_WHEN_EXECUTING_COMMAND; SQLSTATE: HY000

  Message: Error when executing command %s: %s

- Error number: 1221; Symbol: ER_WRONG_USAGE; SQLSTATE: HY000

  Message: Incorrect usage of %s and %s

- Error number: 1222; Symbol: ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT; SQLSTATE: 21000

  Message: The used SELECT statements have a different number of columns

- Error number: 1223; Symbol: ER_CANT_UPDATE_WITH_READLOCK; SQLSTATE: HY000

  Message: Can't execute the query because you have a conflicting read lock
- Error number: 1224; Symbol: ER_MIXING_NOT_ALLOWED; SQLSTATE: HY000

  Message: Mixing of transactional and non-transactional tables is disabled
- Error number: 1225; Symbol: ER_DUP_ARGUMENT; SQLSTATE: HY000

  Message: Option '%s' used twice in statement
- Error number: 1226; Symbol: ER_USER_LIMIT_REACHED; SQLSTATE: 42000

  Message: User '%s' has exceeded the '%s' resource (current value: %ld)
- Error number: 1227; Symbol: ER_SPECIFIC_ACCESS_DENIED_ERROR; SQLSTATE: 42000

  Message: Access denied; you need (at least one of) the %s privilege(s) for this operation
- Error number: 1228; Symbol: ER_LOCAL_VARIABLE; SQLSTATE: HY000

  Message: Variable '%s' is a SESSION variable and can't be used with SET GLOBAL
- Error number: 1229; Symbol: ER_GLOBAL_VARIABLE; SQLSTATE: HY000

  Message: Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL
- Error number: 1230; Symbol: ER_NO_DEFAULT; SQLSTATE: 42000

  Message: Variable '%s' doesn't have a default value
- Error number: 1231; Symbol: ER_WRONG_VALUE_FOR_VAR; SQLSTATE: 42000

  Message: Variable '%s' can't be set to the value of '%s'
- Error number: 1232; Symbol: ER_WRONG_TYPE_FOR_VAR; SQLSTATE: 42000

  Message: Incorrect argument type to variable '%s'
- Error number: 1233; Symbol: ER_VAR_CANT_BE_READ; SQLSTATE: HY000

  Message: Variable '%s' can only be set, not read
- Error number: 1234; Symbol: ER_CANT_USE_OPTION_HERE; SQLSTATE: 42000

  Message: Incorrect usage/placement of '%s'
- Error number: 1235; Symbol: ER_NOT_SUPPORTED_YET; SQLSTATE: 42000

  Message: This version of MySQL doesn't yet support '%s'
- Error number: 1236; Symbol: ER_MASTER_FATAL_ERROR_READING_BINLOG; SQLSTATE: HY000

  Message: Got fatal error %d from master when reading data from binary log: '%s'
- Error number: 1237; Symbol: ER_SLAVE_IGNORED_TABLE; SQLSTATE: HY000

  Message: Slave SQL thread ignored the query because of replicate-*-table rules
- Error number: 1238; Symbol: ER_INCORRECT_GLOBAL_LOCAL_VAR; SQLSTATE: HY000

Message: Variable '%s' is a %s variable

- Error number: 1239; Symbol: ER_WRONG_FK_DEF; SQLSTATE: 42000

  Message: Incorrect foreign key definition for '%s': %s

- Error number: 1240; Symbol: ER_KEY_REF_DO_NOT_MATCH_TABLE_REF; SQLSTATE: HY000

  Message: Key reference and table reference don't match

- Error number: 1241; Symbol: ER_OPERAND_COLUMNS; SQLSTATE: 21000

  Message: Operand should contain %d column(s)

- Error number: 1242; Symbol: ER_SUBQUERY_NO_1_ROW; SQLSTATE: 21000

  Message: Subquery returns more than 1 row

- Error number: 1243; Symbol: ER_UNKNOWN_STMT_HANDLER; SQLSTATE: HY000

  Message: Unknown prepared statement handler (%.*s) given to %s

- Error number: 1244; Symbol: ER_CORRUPT_HELP_DB; SQLSTATE: HY000

  Message: Help database is corrupt or does not exist

- Error number: 1246; Symbol: ER_AUTO_CONVERT; SQLSTATE: HY000

  Message: Converting column '%s' from %s to %s

- Error number: 1247; Symbol: ER_ILLEGAL_REFERENCE; SQLSTATE: 42S22

  Message: Reference '%s' not supported (%s)

- Error number: 1248; Symbol: ER_DERIVED_MUST_HAVE_ALIAS; SQLSTATE: 42000

  Message: Every derived table must have its own alias

- Error number: 1249; Symbol: ER_SELECT_REDUCED; SQLSTATE: 01000

  Message: Select %u was reduced during optimization

- Error number: 1250; Symbol: ER_TABLENAME_NOT_ALLOWED_HERE; SQLSTATE: 42000

  Message: Table '%s' from one of the SELECTs cannot be used in %s

- Error number: 1251; Symbol: ER_NOT_SUPPORTED_AUTH_MODE; SQLSTATE: 08004

  Message: Client does not support authentication protocol requested by server; consider upgrading MySQL client

- Error number: 1252; Symbol: ER_SPATIAL_CANT_HAVE_NULL; SQLSTATE: 42000

  Message: All parts of a SPATIAL index must be NOT NULL

- Error number: 1253; Symbol: ER_COLLATION_CHARSET_MISMATCH; SQLSTATE: 42000

  Message: COLLATION '%s' is not valid for CHARACTER SET '%s'

- Error number: 1256; Symbol: ER_TOO_BIG_FOR_UNCOMPRESS; SQLSTATE: HY000

Message: Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)

- Error number: 1257; Symbol: ER_ZLIB_Z_MEM_ERROR; SQLSTATE: HY000

  Message: ZLIB: Not enough memory

- Error number: 1258; Symbol: ER_ZLIB_Z_BUF_ERROR; SQLSTATE: HY000

  Message: ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)

- Error number: 1259; Symbol: ER_ZLIB_Z_DATA_ERROR; SQLSTATE: HY000

  Message: ZLIB: Input data corrupted

- Error number: 1260; Symbol: ER_CUT_VALUE_GROUP_CONCAT; SQLSTATE: HY000

  Message: Row %u was cut by GROUP_CONCAT()

- Error number: 1261; Symbol: ER_WARN_TOO_FEW_RECORDS; SQLSTATE: 01000

  Message: Row %ld doesn't contain data for all columns

- Error number: 1262; Symbol: ER_WARN_TOO_MANY_RECORDS; SQLSTATE: 01000

  Message: Row %ld was truncated; it contained more data than there were input columns

- Error number: 1263; Symbol: ER_WARN_NULL_TO_NOTNULL; SQLSTATE: 22004

  Message: Column set to default value; NULL supplied to NOT NULL column '%s' at row %ld

- Error number: 1264; Symbol: ER_WARN_DATA_OUT_OF_RANGE; SQLSTATE: 22003

  Message: Out of range value for column '%s' at row %ld

- Error number: 1265; Symbol: WARN_DATA_TRUNCATED; SQLSTATE: 01000

  Message: Data truncated for column '%s' at row %ld

- Error number: 1266; Symbol: ER_WARN_USING_OTHER_HANDLER; SQLSTATE: HY000

  Message: Using storage engine %s for table '%s'

- Error number: 1267; Symbol: ER_CANT_AGGREGATE_2COLLATIONS; SQLSTATE: HY000

  Message: Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'

- Error number: 1269; Symbol: ER_REVOKE_GRANTS; SQLSTATE: HY000

  Message: Can't revoke all privileges for one or more of the requested users

- Error number: 1270; Symbol: ER_CANT_AGGREGATE_3COLLATIONS; SQLSTATE: HY000

  Message: Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'

- Error number: 1271; Symbol: ER_CANT_AGGREGATE_NCOLLATIONS; SQLSTATE: HY000

  Message: Illegal mix of collations for operation '%s'

- Error number: 1272; Symbol: ER_VARIABLE_IS_NOT_STRUCT; SQLSTATE: HY000

  Message: Variable '%s' is not a variable component (can't be used as XXXX.variable_name)

- Error number: 1273; Symbol: ER_UNKNOWN_COLLATION; SQLSTATE: HY000

  Message: Unknown collation: '%s'

- Error number: 1274; Symbol: ER_SLAVE_IGNORED_SSL_PARAMS; SQLSTATE: HY000

  Message: SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started

- Error number: 1275; Symbol: ER_SERVER_IS_IN_SECURE_AUTH_MODE; SQLSTATE: HY000

  Message: Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format

  ER_SERVER_IS_IN_SECURE_AUTH_MODE was removed after 8.0.15.

- Error number: 1276; Symbol: ER_WARN_FIELD_RESOLVED; SQLSTATE: HY000

  Message: Field or reference '%s%s%s%s%s' of SELECT #%d was resolved in SELECT #%d

- Error number: 1277; Symbol: ER_BAD_SLAVE_UNTIL_COND; SQLSTATE: HY000

  Message: Incorrect parameter or combination of parameters for START SLAVE UNTIL

- Error number: 1278; Symbol: ER_MISSING_SKIP_SLAVE; SQLSTATE: HY000

  Message: It is recommended to use --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart

- Error number: 1279; Symbol: ER_UNTIL_COND_IGNORED; SQLSTATE: HY000

  Message: SQL thread is not to be started so UNTIL options are ignored

- Error number: 1280; Symbol: ER_WRONG_NAME_FOR_INDEX; SQLSTATE: 42000

  Message: Incorrect index name '%s'

- Error number: 1281; Symbol: ER_WRONG_NAME_FOR_CATALOG; SQLSTATE: 42000

  Message: Incorrect catalog name '%s'

- Error number: 1282; Symbol: ER_WARN_QC_RESIZE; SQLSTATE: HY000

  Message: Query cache failed to set size %lu; new query cache size is %lu

  ER_WARN_QC_RESIZE was removed after 8.0.2.

- Error number: 1283; Symbol: ER_BAD_FT_COLUMN; SQLSTATE: HY000

  Message: Column '%s' cannot be part of FULLTEXT index

- Error number: 1284; Symbol: ER_UNKNOWN_KEY_CACHE; SQLSTATE: HY000

  Message: Unknown key cache '%s'

- Error number: 1285; Symbol: ER_WARN_HOSTNAME_WONT_WORK; SQLSTATE: HY000

Message: MySQL is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work

- Error number: 1286; Symbol: ER_UNKNOWN_STORAGE_ENGINE; SQLSTATE: 42000

  Message: Unknown storage engine '%s'

- Error number: 1287; Symbol: ER_WARN_DEPRECATED_SYNTAX; SQLSTATE: HY000

  Message: '%s' is deprecated and will be removed in a future release. Please use %s instead

- Error number: 1288; Symbol: ER_NON_UPDATABLE_TABLE; SQLSTATE: HY000

  Message: The target table %s of the %s is not updatable

- Error number: 1289; Symbol: ER_FEATURE_DISABLED; SQLSTATE: HY000

  Message: The '%s' feature is disabled; you need MySQL built with '%s' to have it working

- Error number: 1290; Symbol: ER_OPTION_PREVENTS_STATEMENT; SQLSTATE: HY000

  Message: The MySQL server is running with the %s option so it cannot execute this statement

- Error number: 1291; Symbol: ER_DUPLICATED_VALUE_IN_TYPE; SQLSTATE: HY000

  Message: Column '%s' has duplicated value '%s' in %s

- Error number: 1292; Symbol: ER_TRUNCATED_WRONG_VALUE; SQLSTATE: 22007

  Message: Truncated incorrect %s value: '%s'

- Error number: 1294; Symbol: ER_INVALID_ON_UPDATE; SQLSTATE: HY000

  Message: Invalid ON UPDATE clause for '%s' column

- Error number: 1295; Symbol: ER_UNSUPPORTED_PS; SQLSTATE: HY000

  Message: This command is not supported in the prepared statement protocol yet

- Error number: 1296; Symbol: ER_GET_ERRMSG; SQLSTATE: HY000

  Message: Got error %d '%s' from %s

- Error number: 1297; Symbol: ER_GET_TEMPORARY_ERRMSG; SQLSTATE: HY000

  Message: Got temporary error %d '%s' from %s

- Error number: 1298; Symbol: ER_UNKNOWN_TIME_ZONE; SQLSTATE: HY000

  Message: Unknown or incorrect time zone: '%s'

- Error number: 1299; Symbol: ER_WARN_INVALID_TIMESTAMP; SQLSTATE: HY000

  Message: Invalid TIMESTAMP value in column '%s' at row %ld

- Error number: 1300; Symbol: ER_INVALID_CHARACTER_STRING; SQLSTATE: HY000

  Message: Invalid %s character string: '%s'

- Error number: 1301; Symbol: ER_WARN_ALLOWED_PACKET_OVERFLOWED; SQLSTATE: HY000

  Message: Result of %s() was larger than max_allowed_packet (%ld) - truncated

- Error number: 1302; Symbol: ER_CONFLICTING_DECLARATIONS; SQLSTATE: HY000

  Message: Conflicting declarations: '%s%s' and '%s%s'

- Error number: 1303; Symbol: ER_SP_NO_RECURSIVE_CREATE; SQLSTATE: 2F003

  Message: Can't create a %s from within another stored routine

- Error number: 1304; Symbol: ER_SP_ALREADY_EXISTS; SQLSTATE: 42000

  Message: %s %s already exists

- Error number: 1305; Symbol: ER_SP_DOES_NOT_EXIST; SQLSTATE: 42000

  Message: %s %s does not exist

- Error number: 1306; Symbol: ER_SP_DROP_FAILED; SQLSTATE: HY000

  Message: Failed to DROP %s %s

- Error number: 1307; Symbol: ER_SP_STORE_FAILED; SQLSTATE: HY000

  Message: Failed to CREATE %s %s

- Error number: 1308; Symbol: ER_SP_LILABEL_MISMATCH; SQLSTATE: 42000

  Message: %s with no matching label: %s

- Error number: 1309; Symbol: ER_SP_LABEL_REDEFINE; SQLSTATE: 42000

  Message: Redefining label %s

- Error number: 1310; Symbol: ER_SP_LABEL_MISMATCH; SQLSTATE: 42000

  Message: End-label %s without match

- Error number: 1311; Symbol: ER_SP_UNINIT_VAR; SQLSTATE: 01000

  Message: Referring to uninitialized variable %s

- Error number: 1312; Symbol: ER_SP_BADSELECT; SQLSTATE: 0A000

  Message: PROCEDURE %s can't return a result set in the given context

- Error number: 1313; Symbol: ER_SP_BADRETURN; SQLSTATE: 42000

  Message: RETURN is only allowed in a FUNCTION

- Error number: 1314; Symbol: ER_SP_BADSTATEMENT; SQLSTATE: 0A000

  Message: %s is not allowed in stored procedures

- Error number: 1315; Symbol: ER_UPDATE_LOG_DEPRECATED_IGNORED; SQLSTATE: 42000

  Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been ignored.

- Error number: 1316; Symbol: ER_UPDATE_LOG_DEPRECATED_TRANSLATED; SQLSTATE: 42000

  Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been translated to SET SQL_LOG_BIN.

- Error number: 1317; Symbol: ER_QUERY_INTERRUPTED; SQLSTATE: 70100

  Message: Query execution was interrupted

- Error number: 1318; Symbol: ER_SP_WRONG_NO_OF_ARGS; SQLSTATE: 42000

  Message: Incorrect number of arguments for %s %s; expected %u, got %u

- Error number: 1319; Symbol: ER_SP_COND_MISMATCH; SQLSTATE: 42000

  Message: Undefined CONDITION: %s

- Error number: 1320; Symbol: ER_SP_NORETURN; SQLSTATE: 42000

  Message: No RETURN found in FUNCTION %s

- Error number: 1321; Symbol: ER_SP_NORETURNEND; SQLSTATE: 2F005

  Message: FUNCTION %s ended without RETURN

- Error number: 1322; Symbol: ER_SP_BAD_CURSOR_QUERY; SQLSTATE: 42000

  Message: Cursor statement must be a SELECT

- Error number: 1323; Symbol: ER_SP_BAD_CURSOR_SELECT; SQLSTATE: 42000

  Message: Cursor SELECT must not have INTO

- Error number: 1324; Symbol: ER_SP_CURSOR_MISMATCH; SQLSTATE: 42000

  Message: Undefined CURSOR: %s

- Error number: 1325; Symbol: ER_SP_CURSOR_ALREADY_OPEN; SQLSTATE: 24000

  Message: Cursor is already open

- Error number: 1326; Symbol: ER_SP_CURSOR_NOT_OPEN; SQLSTATE: 24000

  Message: Cursor is not open

- Error number: 1327; Symbol: ER_SP_UNDECLARED_VAR; SQLSTATE: 42000

  Message: Undeclared variable: %s

- Error number: 1328; Symbol: ER_SP_WRONG_NO_OF_FETCH_ARGS; SQLSTATE: HY000

  Message: Incorrect number of FETCH variables

- Error number: 1329; Symbol: ER_SP_FETCH_NO_DATA; SQLSTATE: 02000

  Message: No data - zero rows fetched, selected, or processed

- Error number: 1330; Symbol: ER_SP_DUP_PARAM; SQLSTATE: 42000

  Message: Duplicate parameter: %s

- Error number: 1331; Symbol: ER_SP_DUP_VAR; SQLSTATE: 42000

  Message: Duplicate variable: %s
- Error number: 1332; Symbol: ER_SP_DUP_COND; SQLSTATE: 42000

  Message: Duplicate condition: %s
- Error number: 1333; Symbol: ER_SP_DUP_CURS; SQLSTATE: 42000

  Message: Duplicate cursor: %s
- Error number: 1334; Symbol: ER_SP_CANT_ALTER; SQLSTATE: HY000

  Message: Failed to ALTER %s %s
- Error number: 1335; Symbol: ER_SP_SUBSELECT_NYI; SQLSTATE: 0A000

  Message: Subquery value not supported
- Error number: 1336; Symbol: ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG; SQLSTATE: 0A000

  Message: %s is not allowed in stored function or trigger
- Error number: 1337; Symbol: ER_SP_VARCOND_AFTER_CURSHNDLR; SQLSTATE: 42000

  Message: Variable or condition declaration after cursor or handler declaration
- Error number: 1338; Symbol: ER_SP_CURSOR_AFTER_HANDLER; SQLSTATE: 42000

  Message: Cursor declaration after handler declaration
- Error number: 1339; Symbol: ER_SP_CASE_NOT_FOUND; SQLSTATE: 20000

  Message: Case not found for CASE statement
- Error number: 1340; Symbol: ER_FPARSER_TOO_BIG_FILE; SQLSTATE: HY000

  Message: Configuration file '%s' is too big
- Error number: 1341; Symbol: ER_FPARSER_BAD_HEADER; SQLSTATE: HY000

  Message: Malformed file type header in file '%s'
- Error number: 1342; Symbol: ER_FPARSER_EOF_IN_COMMENT; SQLSTATE: HY000

  Message: Unexpected end of file while parsing comment '%s'
- Error number: 1343; Symbol: ER_FPARSER_ERROR_IN_PARAMETER; SQLSTATE: HY000

  Message: Error while parsing parameter '%s' (line: '%s')
- Error number: 1344; Symbol: ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER; SQLSTATE: HY000

  Message: Unexpected end of file while skipping unknown parameter '%s'
- Error number: 1345; Symbol: ER_VIEW_NO_EXPLAIN; SQLSTATE: HY000

  Message: EXPLAIN/SHOW can not be issued; lacking privileges for underlying table

- Error number: 1347; Symbol: ER_WRONG_OBJECT; SQLSTATE: HY000

  Message: '%s.%s' is not %s

  The named object is incorrect for the type of operation attempted on it. It must be an object of the named type. Example: HANDLER OPEN requires a base table, not a view. It fails if attempted on an INFORMATION_SCHEMA table that is implemented as a view on data dictionary tables.

- Error number: 1348; Symbol: ER_NONUPDATEABLE_COLUMN; SQLSTATE: HY000

  Message: Column '%s' is not updatable

- Error number: 1350; Symbol: ER_VIEW_SELECT_CLAUSE; SQLSTATE: HY000

  Message: View's SELECT contains a '%s' clause

- Error number: 1351; Symbol: ER_VIEW_SELECT_VARIABLE; SQLSTATE: HY000

  Message: View's SELECT contains a variable or parameter

- Error number: 1352; Symbol: ER_VIEW_SELECT_TMPTABLE; SQLSTATE: HY000

  Message: View's SELECT refers to a temporary table '%s'

- Error number: 1353; Symbol: ER_VIEW_WRONG_LIST; SQLSTATE: HY000

  Message: In definition of view, derived table or common table expression, SELECT list and column names list have different column counts

- Error number: 1354; Symbol: ER_WARN_VIEW_MERGE; SQLSTATE: HY000

  Message: View merge algorithm can't be used here for now (assumed undefined algorithm)

- Error number: 1355; Symbol: ER_WARN_VIEW_WITHOUT_KEY; SQLSTATE: HY000

  Message: View being updated does not have complete key of underlying table in it

- Error number: 1356; Symbol: ER_VIEW_INVALID; SQLSTATE: HY000

  Message: View '%s.%s' references invalid table(s) or column(s) or function(s) or definer/invoker of view lack rights to use them

- Error number: 1357; Symbol: ER_SP_NO_DROP_SP; SQLSTATE: HY000

  Message: Can't drop or alter a %s from within another stored routine

- Error number: 1359; Symbol: ER_TRG_ALREADY_EXISTS; SQLSTATE: HY000

  Message: Trigger already exists

- Error number: 1360; Symbol: ER_TRG_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: Trigger does not exist

- Error number: 1361; Symbol: ER_TRG_ON_VIEW_OR_TEMP_TABLE; SQLSTATE: HY000

  Message: Trigger's '%s' is view or temporary table

- Error number: 1362; Symbol: ER_TRG_CANT_CHANGE_ROW; SQLSTATE: HY000

Message: Updating of %s row is not allowed in %strigger

- Error number: 1363; Symbol: ER_TRG_NO_SUCH_ROW_IN_TRG; SQLSTATE: HY000

  Message: There is no %s row in %s trigger

- Error number: 1364; Symbol: ER_NO_DEFAULT_FOR_FIELD; SQLSTATE: HY000

  Message: Field '%s' doesn't have a default value

- Error number: 1365; Symbol: ER_DIVISION_BY_ZERO; SQLSTATE: 22012

  Message: Division by 0

- Error number: 1366; Symbol: ER_TRUNCATED_WRONG_VALUE_FOR_FIELD; SQLSTATE: HY000

  Message: Incorrect %s value: '%s' for column '%s' at row %ld

- Error number: 1367; Symbol: ER_ILLEGAL_VALUE_FOR_TYPE; SQLSTATE: 22007

  Message: Illegal %s '%s' value found during parsing

- Error number: 1368; Symbol: ER_VIEW_NONUPD_CHECK; SQLSTATE: HY000

  Message: CHECK OPTION on non-updatable view '%s.%s'

- Error number: 1369; Symbol: ER_VIEW_CHECK_FAILED; SQLSTATE: HY000

  Message: CHECK OPTION failed '%s.%s'

- Error number: 1370; Symbol: ER_PROCACCESS_DENIED_ERROR; SQLSTATE: 42000

  Message: %s command denied to user '%s'@'%s' for routine '%s'

- Error number: 1371; Symbol: ER_RELAY_LOG_FAIL; SQLSTATE: HY000

  Message: Failed purging old relay logs: %s

- Error number: 1373; Symbol: ER_UNKNOWN_TARGET_BINLOG; SQLSTATE: HY000

  Message: Target log not found in binlog index

- Error number: 1374; Symbol: ER_IO_ERR_LOG_INDEX_READ; SQLSTATE: HY000

  Message: I/O error reading log index file

- Error number: 1375; Symbol: ER_BINLOG_PURGE_PROHIBITED; SQLSTATE: HY000

  Message: Server configuration does not permit binlog purge

- Error number: 1376; Symbol: ER_FSEEK_FAIL; SQLSTATE: HY000

  Message: Failed on fseek()

- Error number: 1377; Symbol: ER_BINLOG_PURGE_FATAL_ERR; SQLSTATE: HY000

  Message: Fatal error during log purge

- Error number: 1378; Symbol: ER_LOG_IN_USE; SQLSTATE: HY000

Message: A purgeable log is in use, will not purge

- Error number: 1379; Symbol: ER_LOG_PURGE_UNKNOWN_ERR; SQLSTATE: HY000

  Message: Unknown error during log purge

- Error number: 1380; Symbol: ER_RELAY_LOG_INIT; SQLSTATE: HY000

  Message: Failed initializing relay log position: %s

- Error number: 1381; Symbol: ER_NO_BINARY_LOGGING; SQLSTATE: HY000

  Message: You are not using binary logging

- Error number: 1382; Symbol: ER_RESERVED_SYNTAX; SQLSTATE: HY000

  Message: The '%s' syntax is reserved for purposes internal to the MySQL server

- Error number: 1390; Symbol: ER_PS_MANY_PARAM; SQLSTATE: HY000

  Message: Prepared statement contains too many placeholders

- Error number: 1391; Symbol: ER_KEY_PART_0; SQLSTATE: HY000

  Message: Key part '%s' length cannot be 0

- Error number: 1392; Symbol: ER_VIEW_CHECKSUM; SQLSTATE: HY000

  Message: View text checksum failed

- Error number: 1393; Symbol: ER_VIEW_MULTIUPDATE; SQLSTATE: HY000

  Message: Can not modify more than one base table through a join view '%s.%s'

- Error number: 1394; Symbol: ER_VIEW_NO_INSERT_FIELD_LIST; SQLSTATE: HY000

  Message: Can not insert into join view '%s.%s' without fields list

- Error number: 1395; Symbol: ER_VIEW_DELETE_MERGE_VIEW; SQLSTATE: HY000

  Message: Can not delete from join view '%s.%s'

- Error number: 1396; Symbol: ER_CANNOT_USER; SQLSTATE: HY000

  Message: Operation %s failed for %s

- Error number: 1397; Symbol: ER_XAER_NOTA; SQLSTATE: XAE04

  Message: XAER_NOTA: Unknown XID

- Error number: 1398; Symbol: ER_XAER_INVAL; SQLSTATE: XAE05

  Message: XAER_INVAL: Invalid arguments (or unsupported command)

- Error number: 1399; Symbol: ER_XAER_RMFAIL; SQLSTATE: XAE07

  Message: XAER_RMFAIL: The command cannot be executed when global transaction is in the %s state

- Error number: 1400; Symbol: ER_XAER_OUTSIDE; SQLSTATE: XAE09

Message: XAER_OUTSIDE: Some work is done outside global transaction

- Error number: 1401; Symbol: ER_XAER_RMERR; SQLSTATE: XAE03

  Message: XAER_RMERR: Fatal error occurred in the transaction branch - check your data for consistency

- Error number: 1402; Symbol: ER_XA_RBROLLBACK; SQLSTATE: XA100

  Message: XA_RBROLLBACK: Transaction branch was rolled back

- Error number: 1403; Symbol: ER_NONEXISTING_PROC_GRANT; SQLSTATE: 42000

  Message: There is no such grant defined for user '%s' on host '%s' on routine '%s'

- Error number: 1404; Symbol: ER_PROC_AUTO_GRANT_FAIL; SQLSTATE: HY000

  Message: Failed to grant EXECUTE and ALTER ROUTINE privileges

- Error number: 1405; Symbol: ER_PROC_AUTO_REVOKE_FAIL; SQLSTATE: HY000

  Message: Failed to revoke all privileges to dropped routine

- Error number: 1406; Symbol: ER_DATA_TOO_LONG; SQLSTATE: 22001

  Message: Data too long for column '%s' at row %ld

- Error number: 1407; Symbol: ER_SP_BAD_SQLSTATE; SQLSTATE: 42000

  Message: Bad SQLSTATE: '%s'

- Error number: 1408; Symbol: ER_STARTUP; SQLSTATE: HY000

  Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d %s

- Error number: 1409; Symbol: ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR; SQLSTATE: HY000

  Message: Can't load value from file with fixed size rows to variable

- Error number: 1410; Symbol: ER_CANT_CREATE_USER_WITH_GRANT; SQLSTATE: 42000

  Message: You are not allowed to create a user with GRANT

- Error number: 1411; Symbol: ER_WRONG_VALUE_FOR_TYPE; SQLSTATE: HY000

  Message: Incorrect %s value: '%s' for function %s

- Error number: 1412; Symbol: ER_TABLE_DEF_CHANGED; SQLSTATE: HY000

  Message: Table definition has changed, please retry transaction

- Error number: 1413; Symbol: ER_SP_DUP_HANDLER; SQLSTATE: 42000

  Message: Duplicate handler declared in the same block

- Error number: 1414; Symbol: ER_SP_NOT_VAR_ARG; SQLSTATE: 42000

  Message: OUT or INOUT argument %d for routine %s is not a variable or NEW pseudo-variable in BEFORE trigger

- Error number: 1415; Symbol: ER_SP_NO_RETSET; SQLSTATE: 0A000

  Message: Not allowed to return a result set from a %s

- Error number: 1416; Symbol: ER_CANT_CREATE_GEOMETRY_OBJECT; SQLSTATE: 22003

  Message: Cannot get geometry object from data you send to the GEOMETRY field

- Error number: 1418; Symbol: ER_BINLOG_UNSAFE_ROUTINE; SQLSTATE: HY000

  Message: This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA in its declaration and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

- Error number: 1419; Symbol: ER_BINLOG_CREATE_ROUTINE_NEED_SUPER; SQLSTATE: HY000

  Message: You do not have the SUPER privilege and binary logging is enabled (you *might* want to use the less safe log_bin_trust_function_creators variable)

- Error number: 1421; Symbol: ER_STMT_HAS_NO_OPEN_CURSOR; SQLSTATE: HY000

  Message: The statement (%lu) has no open cursor.

- Error number: 1422; Symbol: ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG; SQLSTATE: HY000

  Message: Explicit or implicit commit is not allowed in stored function or trigger.

- Error number: 1423; Symbol: ER_NO_DEFAULT_FOR_VIEW_FIELD; SQLSTATE: HY000

  Message: Field of view '%s.%s' underlying table doesn't have a default value

- Error number: 1424; Symbol: ER_SP_NO_RECURSION; SQLSTATE: HY000

  Message: Recursive stored functions and triggers are not allowed.

- Error number: 1425; Symbol: ER_TOO_BIG_SCALE; SQLSTATE: 42000

  Message: Too big scale %d specified for column '%s'. Maximum is %lu.

- Error number: 1426; Symbol: ER_TOO_BIG_PRECISION; SQLSTATE: 42000

  Message: Too-big precision %d specified for '%s'. Maximum is %lu.

- Error number: 1427; Symbol: ER_M_BIGGER_THAN_D; SQLSTATE: 42000

  Message: For float(M,D), double(M,D) or decimal(M,D), M must be >= D (column '%s').

- Error number: 1428; Symbol: ER_WRONG_LOCK_OF_SYSTEM_TABLE; SQLSTATE: HY000

  Message: You can't combine write-locking of system tables with other tables or lock types

- Error number: 1429; Symbol: ER_CONNECT_TO_FOREIGN_DATA_SOURCE; SQLSTATE: HY000

  Message: Unable to connect to foreign data source: %s

- Error number: 1430; Symbol: ER_QUERY_ON_FOREIGN_DATA_SOURCE; SQLSTATE: HY000

  Message: There was a problem processing the query on the foreign data source. Data source error: %s

- Error number: 1431; Symbol: ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST; SQLSTATE: HY000

Message: The foreign data source you are trying to reference does not exist. Data source error: %s

- Error number: 1432; Symbol: ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE; SQLSTATE: HY000

  Message: Can't create federated table. The data source connection string '%s' is not in the correct format

- Error number: 1433; Symbol: ER_FOREIGN_DATA_STRING_INVALID; SQLSTATE: HY000

  Message: The data source connection string '%s' is not in the correct format

- Error number: 1435; Symbol: ER_TRG_IN_WRONG_SCHEMA; SQLSTATE: HY000

  Message: Trigger in wrong schema

- Error number: 1436; Symbol: ER_STACK_OVERRUN_NEED_MORE; SQLSTATE: HY000

  Message: Thread stack overrun: %ld bytes used of a %ld byte stack, and %ld bytes needed. Use 'mysqld --thread_stack=#' to specify a bigger stack.

- Error number: 1437; Symbol: ER_TOO_LONG_BODY; SQLSTATE: 42000

  Message: Routine body for '%s' is too long

- Error number: 1438; Symbol: ER_WARN_CANT_DROP_DEFAULT_KEYCACHE; SQLSTATE: HY000

  Message: Cannot drop default keycache

- Error number: 1439; Symbol: ER_TOO_BIG_DISPLAYWIDTH; SQLSTATE: 42000

  Message: Display width out of range for column '%s' (max = %lu)

- Error number: 1440; Symbol: ER_XAER_DUPID; SQLSTATE: XAE08

  Message: XAER_DUPID: The XID already exists

- Error number: 1441; Symbol: ER_DATETIME_FUNCTION_OVERFLOW; SQLSTATE: 22008

  Message: Datetime function: %s field overflow

- Error number: 1442; Symbol: ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG; SQLSTATE: HY000

  Message: Can't update table '%s' in stored function/trigger because it is already used by statement which invoked this stored function/trigger.

- Error number: 1443; Symbol: ER_VIEW_PREVENT_UPDATE; SQLSTATE: HY000

  Message: The definition of table '%s' prevents operation %s on table '%s'.

- Error number: 1444; Symbol: ER_PS_NO_RECURSION; SQLSTATE: HY000

  Message: The prepared statement contains a stored routine call that refers to that same statement. It's not allowed to execute a prepared statement in such a recursive manner

- Error number: 1445; Symbol: ER_SP_CANT_SET_AUTOCOMMIT; SQLSTATE: HY000

  Message: Not allowed to set autocommit from a stored function or trigger

- Error number: 1447; Symbol: ER_VIEW_FRM_NO_USER; SQLSTATE: HY000

  Message: View '%s'.'%s' has no definer information (old table format). Current user is used as definer. Please recreate the view!

- Error number: 1448; Symbol: ER_VIEW_OTHER_USER; SQLSTATE: HY000

  Message: You need the SUPER privilege for creation view with '%s'@'%s' definer

- Error number: 1449; Symbol: ER_NO_SUCH_USER; SQLSTATE: HY000

  Message: The user specified as a definer ('%s'@'%s') does not exist

- Error number: 1450; Symbol: ER_FORBID_SCHEMA_CHANGE; SQLSTATE: HY000

  Message: Changing schema from '%s' to '%s' is not allowed.

- Error number: 1451; Symbol: ER_ROW_IS_REFERENCED_2; SQLSTATE: 23000

  Message: Cannot delete or update a parent row: a foreign key constraint fails (%s)

  InnoDB reports this error when you try to delete a parent row that has children, and a foreign key constraint fails. Delete the children first.

- Error number: 1452; Symbol: ER_NO_REFERENCED_ROW_2; SQLSTATE: 23000

  Message: Cannot add or update a child row: a foreign key constraint fails (%s)

  InnoDB reports this error when you try to add a row but there is no parent row, and a foreign key constraint fails. Add the parent row first.

- Error number: 1453; Symbol: ER_SP_BAD_VAR_SHADOW; SQLSTATE: 42000

  Message: Variable '%s' must be quoted with `...`, or renamed

- Error number: 1454; Symbol: ER_TRG_NO_DEFINER; SQLSTATE: HY000

  Message: No definer attribute for trigger '%s'.'%s'. It's disallowed to create trigger without definer.

- Error number: 1455; Symbol: ER_OLD_FILE_FORMAT; SQLSTATE: HY000

  Message: '%s' has an old format, you should re-create the '%s' object(s)

- Error number: 1456; Symbol: ER_SP_RECURSION_LIMIT; SQLSTATE: HY000

  Message: Recursive limit %d (as set by the max_sp_recursion_depth variable) was exceeded for routine %s

- Error number: 1458; Symbol: ER_SP_WRONG_NAME; SQLSTATE: 42000

  Message: Incorrect routine name '%s'

- Error number: 1459; Symbol: ER_TABLE_NEEDS_UPGRADE; SQLSTATE: HY000

  Message: Table upgrade required. Please do "REPAIR TABLE `%s`" or dump/reload to fix it!

- Error number: 1460; Symbol: ER_SP_NO_AGGREGATE; SQLSTATE: 42000

  Message: AGGREGATE is not supported for stored functions

- Error number: 1461; Symbol: ER_MAX_PREPARED_STMT_COUNT_REACHED; SQLSTATE: 42000

  Message: Can't create more than max_prepared_stmt_count statements (current value: %lu)

- Error number: 1462; Symbol: ER_VIEW_RECURSIVE; SQLSTATE: HY000

  Message: `%s`.`%s` contains view recursion

- Error number: 1463; Symbol: ER_NON_GROUPING_FIELD_USED; SQLSTATE: 42000

  Message: Non-grouping field '%s' is used in %s clause

- Error number: 1464; Symbol: ER_TABLE_CANT_HANDLE_SPKEYS; SQLSTATE: HY000

  Message: The used table type doesn't support SPATIAL indexes

- Error number: 1465; Symbol: ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA; SQLSTATE: HY000

  Message: Triggers can not be created on system tables

- Error number: 1466; Symbol: ER_REMOVED_SPACES; SQLSTATE: HY000

  Message: Leading spaces are removed from name '%s'

- Error number: 1467; Symbol: ER_AUTOINC_READ_FAILED; SQLSTATE: HY000

  Message: Failed to read auto-increment value from storage engine

- Error number: 1468; Symbol: ER_USERNAME; SQLSTATE: HY000

  Message: user name

- Error number: 1469; Symbol: ER_HOSTNAME; SQLSTATE: HY000

  Message: host name

- Error number: 1470; Symbol: ER_WRONG_STRING_LENGTH; SQLSTATE: HY000

  Message: String '%s' is too long for %s (should be no longer than %d)

- Error number: 1471; Symbol: ER_NON_INSERTABLE_TABLE; SQLSTATE: HY000

  Message: The target table %s of the %s is not insertable-into

- Error number: 1472; Symbol: ER_ADMIN_WRONG_MRG_TABLE; SQLSTATE: HY000

  Message: Table '%s' is differently defined or of non-MyISAM type or doesn't exist

- Error number: 1473; Symbol: ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT; SQLSTATE: HY000

  Message: Too high level of nesting for select

- Error number: 1474; Symbol: ER_NAME_BECOMES_EMPTY; SQLSTATE: HY000

  Message: Name '%s' has become ''

- Error number: 1475; Symbol: ER_AMBIGUOUS_FIELD_TERM; SQLSTATE: HY000

  Message: First character of the FIELDS TERMINATED string is ambiguous; please use non-optional and non-empty FIELDS ENCLOSED BY

- Error number: 1476; Symbol: ER_FOREIGN_SERVER_EXISTS; SQLSTATE: HY000

  Message: The foreign server, %s, you are trying to create already exists.

- Error number: 1477; Symbol: ER_FOREIGN_SERVER_DOESNT_EXIST; SQLSTATE: HY000

  Message: The foreign server name you are trying to reference does not exist. Data source error: %s

- Error number: 1478; Symbol: ER_ILLEGAL_HA_CREATE_OPTION; SQLSTATE: HY000

  Message: Table storage engine '%s' does not support the create option '%s'

- Error number: 1479; Symbol: ER_PARTITION_REQUIRES_VALUES_ERROR; SQLSTATE: HY000

  Message: Syntax error: %s PARTITIONING requires definition of VALUES %s for each partition

- Error number: 1480; Symbol: ER_PARTITION_WRONG_VALUES_ERROR; SQLSTATE: HY000

  Message: Only %s PARTITIONING can use VALUES %s in partition definition

- Error number: 1481; Symbol: ER_PARTITION_MAXVALUE_ERROR; SQLSTATE: HY000

  Message: MAXVALUE can only be used in last partition definition

- Error number: 1484; Symbol: ER_PARTITION_WRONG_NO_PART_ERROR; SQLSTATE: HY000

  Message: Wrong number of partitions defined, mismatch with previous setting

- Error number: 1485; Symbol: ER_PARTITION_WRONG_NO_SUBPART_ERROR; SQLSTATE: HY000

  Message: Wrong number of subpartitions defined, mismatch with previous setting

- Error number: 1486; Symbol: ER_WRONG_EXPR_IN_PARTITION_FUNC_ERROR; SQLSTATE: HY000

  Message: Constant, random or timezone-dependent expressions in (sub)partitioning function are not allowed

- Error number: 1488; Symbol: ER_FIELD_NOT_FOUND_PART_ERROR; SQLSTATE: HY000

  Message: Field in list of fields for partition function not found in table

- Error number: 1490; Symbol: ER_INCONSISTENT_PARTITION_INFO_ERROR; SQLSTATE: HY000

  Message: The partition info in the frm file is not consistent with what can be written into the frm file

- Error number: 1491; Symbol: ER_PARTITION_FUNC_NOT_ALLOWED_ERROR; SQLSTATE: HY000

  Message: The %s function returns the wrong type

- Error number: 1492; Symbol: ER_PARTITIONS_MUST_BE_DEFINED_ERROR; SQLSTATE: HY000

  Message: For %s partitions each partition must be defined

- Error number: 1493; Symbol: ER_RANGE_NOT_INCREASING_ERROR; SQLSTATE: HY000

  Message: VALUES LESS THAN value must be strictly increasing for each partition

- Error number: 1494; Symbol: ER_INCONSISTENT_TYPE_OF_FUNCTIONS_ERROR; SQLSTATE: HY000

Message: VALUES value must be of same type as partition function

- Error number: 1495; Symbol: ER_MULTIPLE_DEF_CONST_IN_LIST_PART_ERROR; SQLSTATE: HY000

  Message: Multiple definition of same constant in list partitioning

- Error number: 1496; Symbol: ER_PARTITION_ENTRY_ERROR; SQLSTATE: HY000

  Message: Partitioning can not be used stand-alone in query

- Error number: 1497; Symbol: ER_MIX_HANDLER_ERROR; SQLSTATE: HY000

  Message: The mix of handlers in the partitions is not allowed in this version of MySQL

- Error number: 1498; Symbol: ER_PARTITION_NOT_DEFINED_ERROR; SQLSTATE: HY000

  Message: For the partitioned engine it is necessary to define all %s

- Error number: 1499; Symbol: ER_TOO_MANY_PARTITIONS_ERROR; SQLSTATE: HY000

  Message: Too many partitions (including subpartitions) were defined

- Error number: 1500; Symbol: ER_SUBPARTITION_ERROR; SQLSTATE: HY000

  Message: It is only possible to mix RANGE/LIST partitioning with HASH/KEY partitioning for subpartitioning

- Error number: 1501; Symbol: ER_CANT_CREATE_HANDLER_FILE; SQLSTATE: HY000

  Message: Failed to create specific handler file

- Error number: 1502; Symbol: ER_BLOB_FIELD_IN_PART_FUNC_ERROR; SQLSTATE: HY000

  Message: A BLOB field is not allowed in partition function

- Error number: 1503; Symbol: ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF; SQLSTATE: HY000

  Message: A %s must include all columns in the table's partitioning function

- Error number: 1504; Symbol: ER_NO_PARTS_ERROR; SQLSTATE: HY000

  Message: Number of %s = 0 is not an allowed value

- Error number: 1505; Symbol: ER_PARTITION_MGMT_ON_NONPARTITIONED; SQLSTATE: HY000

  Message: Partition management on a not partitioned table is not possible

- Error number: 1506; Symbol: ER_FOREIGN_KEY_ON_PARTITIONED; SQLSTATE: HY000

  Message: Foreign keys are not yet supported in conjunction with partitioning

- Error number: 1507; Symbol: ER_DROP_PARTITION_NON_EXISTENT; SQLSTATE: HY000

  Message: Error in list of partitions to %s

- Error number: 1508; Symbol: ER_DROP_LAST_PARTITION; SQLSTATE: HY000

  Message: Cannot remove all partitions, use DROP TABLE instead

- Error number: 1509; Symbol: ER_COALESCE_ONLY_ON_HASH_PARTITION; SQLSTATE: HY000

  Message: COALESCE PARTITION can only be used on HASH/KEY partitions

- Error number: 1510; Symbol: ER_REORG_HASH_ONLY_ON_SAME_NO; SQLSTATE: HY000

  Message: REORGANIZE PARTITION can only be used to reorganize partitions not to change their numbers

- Error number: 1511; Symbol: ER_REORG_NO_PARAM_ERROR; SQLSTATE: HY000

  Message: REORGANIZE PARTITION without parameters can only be used on auto-partitioned tables using HASH PARTITIONs

- Error number: 1512; Symbol: ER_ONLY_ON_RANGE_LIST_PARTITION; SQLSTATE: HY000

  Message: %s PARTITION can only be used on RANGE/LIST partitions

- Error number: 1513; Symbol: ER_ADD_PARTITION_SUBPART_ERROR; SQLSTATE: HY000

  Message: Trying to Add partition(s) with wrong number of subpartitions

- Error number: 1514; Symbol: ER_ADD_PARTITION_NO_NEW_PARTITION; SQLSTATE: HY000

  Message: At least one partition must be added

- Error number: 1515; Symbol: ER_COALESCE_PARTITION_NO_PARTITION; SQLSTATE: HY000

  Message: At least one partition must be coalesced

- Error number: 1516; Symbol: ER_REORG_PARTITION_NOT_EXIST; SQLSTATE: HY000

  Message: More partitions to reorganize than there are partitions

- Error number: 1517; Symbol: ER_SAME_NAME_PARTITION; SQLSTATE: HY000

  Message: Duplicate partition name %s

- Error number: 1518; Symbol: ER_NO_BINLOG_ERROR; SQLSTATE: HY000

  Message: It is not allowed to shut off binlog on this command

- Error number: 1519; Symbol: ER_CONSECUTIVE_REORG_PARTITIONS; SQLSTATE: HY000

  Message: When reorganizing a set of partitions they must be in consecutive order

- Error number: 1520; Symbol: ER_REORG_OUTSIDE_RANGE; SQLSTATE: HY000

  Message: Reorganize of range partitions cannot change total ranges except for last partition where it can extend the range

- Error number: 1521; Symbol: ER_PARTITION_FUNCTION_FAILURE; SQLSTATE: HY000

  Message: Partition function not supported in this version for this handler

- Error number: 1523; Symbol: ER_LIMITED_PART_RANGE; SQLSTATE: HY000

  Message: The %s handler only supports 32 bit integers in VALUES

- Error number: 1524; Symbol: ER_PLUGIN_IS_NOT_LOADED; SQLSTATE: HY000

Message: Plugin '%s' is not loaded

- Error number: 1525; Symbol: ER_WRONG_VALUE; SQLSTATE: HY000

  Message: Incorrect %s value: '%s'

- Error number: 1526; Symbol: ER_NO_PARTITION_FOR_GIVEN_VALUE; SQLSTATE: HY000

  Message: Table has no partition for value %s

- Error number: 1527; Symbol: ER_FILEGROUP_OPTION_ONLY_ONCE; SQLSTATE: HY000

  Message: It is not allowed to specify %s more than once

- Error number: 1528; Symbol: ER_CREATE_FILEGROUP_FAILED; SQLSTATE: HY000

  Message: Failed to create %s

- Error number: 1529; Symbol: ER_DROP_FILEGROUP_FAILED; SQLSTATE: HY000

  Message: Failed to drop %s

- Error number: 1530; Symbol: ER_TABLESPACE_AUTO_EXTEND_ERROR; SQLSTATE: HY000

  Message: The handler doesn't support autoextend of tablespaces

- Error number: 1531; Symbol: ER_WRONG_SIZE_NUMBER; SQLSTATE: HY000

  Message: A size parameter was incorrectly specified, either number or on the form 10M

- Error number: 1532; Symbol: ER_SIZE_OVERFLOW_ERROR; SQLSTATE: HY000

  Message: The size number was correct but we don't allow the digit part to be more than 2 billion

- Error number: 1533; Symbol: ER_ALTER_FILEGROUP_FAILED; SQLSTATE: HY000

  Message: Failed to alter: %s

- Error number: 1534; Symbol: ER_BINLOG_ROW_LOGGING_FAILED; SQLSTATE: HY000

  Message: Writing one row to the row-based binary log failed

- Error number: 1537; Symbol: ER_EVENT_ALREADY_EXISTS; SQLSTATE: HY000

  Message: Event '%s' already exists

- Error number: 1539; Symbol: ER_EVENT_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: Unknown event '%s'

- Error number: 1542; Symbol: ER_EVENT_INTERVAL_NOT_POSITIVE_OR_TOO_BIG; SQLSTATE: HY000

  Message: INTERVAL is either not positive or too big

- Error number: 1543; Symbol: ER_EVENT_ENDS_BEFORE_STARTS; SQLSTATE: HY000

  Message: ENDS is either invalid or before STARTS

- Error number: 1544; Symbol: ER_EVENT_EXEC_TIME_IN_THE_PAST; SQLSTATE: HY000

  Message: Event execution time is in the past. Event has been disabled

- Error number: 1551; Symbol: ER_EVENT_SAME_NAME; SQLSTATE: HY000

  Message: Same old and new event name

- Error number: 1553; Symbol: ER_DROP_INDEX_FK; SQLSTATE: HY000

  Message: Cannot drop index '%s': needed in a foreign key constraint

  InnoDB reports this error when you attempt to drop the last index that can enforce a particular referential constraint.

  For optimal performance with DML statements, InnoDB requires an index to exist on foreign key columns, so that UPDATE and DELETE operations on a parent table can easily check whether corresponding rows exist in the child table. MySQL creates or drops such indexes automatically when needed, as a side-effect of CREATE TABLE, CREATE INDEX, and ALTER TABLE statements.

  When you drop an index, InnoDB checks if the index is used for checking a foreign key constraint. It is still OK to drop the index if there is another index that can be used to enforce the same constraint. InnoDB prevents you from dropping the last index that can enforce a particular referential constraint.

- Error number: 1554; Symbol: ER_WARN_DEPRECATED_SYNTAX_WITH_VER; SQLSTATE: HY000

  Message: The syntax '%s' is deprecated and will be removed in MySQL %s. Please use %s instead

- Error number: 1556; Symbol: ER_CANT_LOCK_LOG_TABLE; SQLSTATE: HY000

  Message: You can't use locks with log tables.

- Error number: 1557; Symbol: ER_FOREIGN_DUPLICATE_KEY_OLD_UNUSED; SQLSTATE: 23000

  Message: Upholding foreign key constraints for table '%s', entry '%s', key %d would lead to a duplicate entry

- Error number: 1558; Symbol: ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE; SQLSTATE: HY000

  Message: The column count of mysql.%s is wrong. Expected %d, found %d. Created with MySQL %d, now running %d. Please perform the MySQL upgrade procedure.

- Error number: 1559; Symbol: ER_TEMP_TABLE_PREVENTS_SWITCH_OUT_OF_RBR; SQLSTATE: HY000

  Message: Cannot switch out of the row-based binary log format when the session has open temporary tables

  ER_TEMP_TABLE_PREVENTS_SWITCH_OUT_OF_RBR was removed after 8.0.12.

- Error number: 1560; Symbol: ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_FORMAT; SQLSTATE: HY000

  Message: Cannot change the binary logging format inside a stored function or trigger

- Error number: 1562; Symbol: ER_PARTITION_NO_TEMPORARY; SQLSTATE: HY000

  Message: Cannot create temporary table with partitions

- Error number: 1563; Symbol: ER_PARTITION_CONST_DOMAIN_ERROR; SQLSTATE: HY000

  Message: Partition constant is out of partition function domain

- Error number: 1564; Symbol: ER_PARTITION_FUNCTION_IS_NOT_ALLOWED; SQLSTATE: HY000

  Message: This partition function is not allowed

- Error number: 1565; Symbol: ER_DDL_LOG_ERROR; SQLSTATE: HY000

  Message: Error in DDL log

  ER_DDL_LOG_ERROR was removed after 8.0.1.

- Error number: 1566; Symbol: ER_NULL_IN_VALUES_LESS_THAN; SQLSTATE: HY000

  Message: Not allowed to use NULL value in VALUES LESS THAN

- Error number: 1567; Symbol: ER_WRONG_PARTITION_NAME; SQLSTATE: HY000

  Message: Incorrect partition name

- Error number: 1568; Symbol: ER_CANT_CHANGE_TX_CHARACTERISTICS; SQLSTATE: 25001

  Message: Transaction characteristics can't be changed while a transaction is in progress

- Error number: 1569; Symbol: ER_DUP_ENTRY_AUTOINCREMENT_CASE; SQLSTATE: HY000

  Message: ALTER TABLE causes auto_increment resequencing, resulting in duplicate entry '%s' for key '%s'

- Error number: 1571; Symbol: ER_EVENT_SET_VAR_ERROR; SQLSTATE: HY000

  Message: Error during starting/stopping of the scheduler. Error code %u

- Error number: 1572; Symbol: ER_PARTITION_MERGE_ERROR; SQLSTATE: HY000

  Message: Engine cannot be used in partitioned tables

- Error number: 1575; Symbol: ER_BASE64_DECODE_ERROR; SQLSTATE: HY000

  Message: Decoding of base64 string failed

- Error number: 1576; Symbol: ER_EVENT_RECURSION_FORBIDDEN; SQLSTATE: HY000

  Message: Recursion of EVENT DDL statements is forbidden when body is present

- Error number: 1578; Symbol: ER_ONLY_INTEGERS_ALLOWED; SQLSTATE: HY000

  Message: Only integers allowed as number here

- Error number: 1579; Symbol: ER_UNSUPORTED_LOG_ENGINE; SQLSTATE: HY000

  Message: This storage engine cannot be used for log tables

- Error number: 1580; Symbol: ER_BAD_LOG_STATEMENT; SQLSTATE: HY000

  Message: You cannot '%s' a log table if logging is enabled

- Error number: 1581; Symbol: ER_CANT_RENAME_LOG_TABLE; SQLSTATE: HY000

Message: Cannot rename '%s'. When logging enabled, rename to/from log table must rename two tables: the log table to an archive table and another table back to '%s'

- Error number: 1582; Symbol: ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT; SQLSTATE: 42000

  Message: Incorrect parameter count in the call to native function '%s'

- Error number: 1583; Symbol: ER_WRONG_PARAMETERS_TO_NATIVE_FCT; SQLSTATE: 42000

  Message: Incorrect parameters in the call to native function '%s'

- Error number: 1584; Symbol: ER_WRONG_PARAMETERS_TO_STORED_FCT; SQLSTATE: 42000

  Message: Incorrect parameters in the call to stored function %s

- Error number: 1585; Symbol: ER_NATIVE_FCT_NAME_COLLISION; SQLSTATE: HY000

  Message: This function '%s' has the same name as a native function

- Error number: 1586; Symbol: ER_DUP_ENTRY_WITH_KEY_NAME; SQLSTATE: 23000

  Message: Duplicate entry '%s' for key '%s'

  The format string for this error is also used with ER_DUP_ENTRY.

- Error number: 1587; Symbol: ER_BINLOG_PURGE_EMFILE; SQLSTATE: HY000

  Message: Too many files opened, please execute the command again

- Error number: 1588; Symbol: ER_EVENT_CANNOT_CREATE_IN_THE_PAST; SQLSTATE: HY000

  Message: Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was dropped immediately after creation.

- Error number: 1589; Symbol: ER_EVENT_CANNOT_ALTER_IN_THE_PAST; SQLSTATE: HY000

  Message: Event execution time is in the past and ON COMPLETION NOT PRESERVE is set. The event was not changed. Specify a time in the future.

- Error number: 1591; Symbol: ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT; SQLSTATE: HY000

  Message: Table has no partition for some existing values

- Error number: 1592; Symbol: ER_BINLOG_UNSAFE_STATEMENT; SQLSTATE: HY000

  Message: Unsafe statement written to the binary log using statement format since BINLOG_FORMAT = STATEMENT. %s

- Error number: 1593; Symbol: ER_BINLOG_FATAL_ERROR; SQLSTATE: HY000

  Message: Fatal error: %s

  ER_BINLOG_FATAL_ERROR was added in 8.0.11.

- Error number: 1598; Symbol: ER_BINLOG_LOGGING_IMPOSSIBLE; SQLSTATE: HY000

  Message: Binary logging not possible. Message: %s

- Error number: 1599; Symbol: ER_VIEW_NO_CREATION_CTX; SQLSTATE: HY000

Message: View `%s`.`%s` has no creation context

- Error number: 1600; Symbol: ER_VIEW_INVALID_CREATION_CTX; SQLSTATE: HY000

  Message: Creation context of view `%s`.`%s' is invalid

- Error number: 1602; Symbol: ER_TRG_CORRUPTED_FILE; SQLSTATE: HY000

  Message: Corrupted TRG file for table `%s`.`%s`

- Error number: 1603; Symbol: ER_TRG_NO_CREATION_CTX; SQLSTATE: HY000

  Message: Triggers for table `%s`.`%s` have no creation context

- Error number: 1604; Symbol: ER_TRG_INVALID_CREATION_CTX; SQLSTATE: HY000

  Message: Trigger creation context of table `%s`.`%s` is invalid

- Error number: 1605; Symbol: ER_EVENT_INVALID_CREATION_CTX; SQLSTATE: HY000

  Message: Creation context of event `%s`.`%s` is invalid

- Error number: 1606; Symbol: ER_TRG_CANT_OPEN_TABLE; SQLSTATE: HY000

  Message: Cannot open table for trigger `%s`.`%s`

- Error number: 1609; Symbol: ER_NO_FORMAT_DESCRIPTION_EVENT_BEFORE_BINLOG_STATEMENT; SQLSTATE: HY000

  Message: The BINLOG statement of type `%s` was not preceded by a format description BINLOG statement.

- Error number: 1610; Symbol: ER_SLAVE_CORRUPT_EVENT; SQLSTATE: HY000

  Message: Corrupted replication event was detected

- Error number: 1612; Symbol: ER_LOG_PURGE_NO_FILE; SQLSTATE: HY000

  Message: Being purged log %s was not found

- Error number: 1613; Symbol: ER_XA_RBTIMEOUT; SQLSTATE: XA106

  Message: XA_RBTIMEOUT: Transaction branch was rolled back: took too long

- Error number: 1614; Symbol: ER_XA_RBDEADLOCK; SQLSTATE: XA102

  Message: XA_RBDEADLOCK: Transaction branch was rolled back: deadlock was detected

- Error number: 1615; Symbol: ER_NEED_REPREPARE; SQLSTATE: HY000

  Message: Prepared statement needs to be re-prepared

- Error number: 1617; Symbol: WARN_NO_MASTER_INFO; SQLSTATE: HY000

  Message: The master info structure does not exist

- Error number: 1618; Symbol: WARN_OPTION_IGNORED; SQLSTATE: HY000

  Message: <%s> option ignored

- Error number: 1619; Symbol: ER_PLUGIN_DELETE_BUILTIN; SQLSTATE: HY000

  Message: Built-in plugins cannot be deleted

- Error number: 1620; Symbol: WARN_PLUGIN_BUSY; SQLSTATE: HY000

  Message: Plugin is busy and will be uninstalled on shutdown

- Error number: 1621; Symbol: ER_VARIABLE_IS_READONLY; SQLSTATE: HY000

  Message: %s variable '%s' is read-only. Use SET %s to assign the value

- Error number: 1622; Symbol: ER_WARN_ENGINE_TRANSACTION_ROLLBACK; SQLSTATE: HY000

  Message: Storage engine %s does not support rollback for this statement. Transaction rolled back and must be restarted

- Error number: 1624; Symbol: ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE; SQLSTATE: HY000

  Message: The requested value for the heartbeat period is either negative or exceeds the maximum allowed (%s seconds).

- Error number: 1625; Symbol: ER_NDB_REPLICATION_SCHEMA_ERROR; SQLSTATE: HY000

  Message: Bad schema for mysql.ndb_replication table. Message: %s

- Error number: 1626; Symbol: ER_CONFLICT_FN_PARSE_ERROR; SQLSTATE: HY000

  Message: Error in parsing conflict function. Message: %s

- Error number: 1627; Symbol: ER_EXCEPTIONS_WRITE_ERROR; SQLSTATE: HY000

  Message: Write to exceptions table failed. Message: %s

- Error number: 1628; Symbol: ER_TOO_LONG_TABLE_COMMENT; SQLSTATE: HY000

  Message: Comment for table '%s' is too long (max = %lu)

- Error number: 1629; Symbol: ER_TOO_LONG_FIELD_COMMENT; SQLSTATE: HY000

  Message: Comment for field '%s' is too long (max = %lu)

- Error number: 1630; Symbol: ER_FUNC_INEXISTENT_NAME_COLLISION; SQLSTATE: 42000

  Message: FUNCTION %s does not exist. Check the 'Function Name Parsing and Resolution' section in the Reference Manual

- Error number: 1631; Symbol: ER_DATABASE_NAME; SQLSTATE: HY000

  Message: Database

- Error number: 1632; Symbol: ER_TABLE_NAME; SQLSTATE: HY000

  Message: Table

- Error number: 1633; Symbol: ER_PARTITION_NAME; SQLSTATE: HY000

  Message: Partition

- Error number: 1634; Symbol: ER_SUBPARTITION_NAME; SQLSTATE: HY000

Message: Subpartition

- Error number: 1635; Symbol: ER_TEMPORARY_NAME; SQLSTATE: HY000

  Message: Temporary

- Error number: 1636; Symbol: ER_RENAMED_NAME; SQLSTATE: HY000

  Message: Renamed

- Error number: 1637; Symbol: ER_TOO_MANY_CONCURRENT_TRXS; SQLSTATE: HY000

  Message: Too many active concurrent transactions

- Error number: 1638; Symbol: WARN_NON_ASCII_SEPARATOR_NOT_IMPLEMENTED; SQLSTATE: HY000

  Message: Non-ASCII separator arguments are not fully supported

- Error number: 1639; Symbol: ER_DEBUG_SYNC_TIMEOUT; SQLSTATE: HY000

  Message: debug sync point wait timed out

- Error number: 1640; Symbol: ER_DEBUG_SYNC_HIT_LIMIT; SQLSTATE: HY000

  Message: debug sync point hit limit reached

- Error number: 1641; Symbol: ER_DUP_SIGNAL_SET; SQLSTATE: 42000

  Message: Duplicate condition information item '%s'

- Error number: 1642; Symbol: ER_SIGNAL_WARN; SQLSTATE: 01000

  Message: Unhandled user-defined warning condition

- Error number: 1643; Symbol: ER_SIGNAL_NOT_FOUND; SQLSTATE: 02000

  Message: Unhandled user-defined not found condition

- Error number: 1644; Symbol: ER_SIGNAL_EXCEPTION; SQLSTATE: HY000

  Message: Unhandled user-defined exception condition

- Error number: 1645; Symbol: ER_RESIGNAL_WITHOUT_ACTIVE_HANDLER; SQLSTATE: 0K000

  Message: RESIGNAL when handler not active

- Error number: 1646; Symbol: ER_SIGNAL_BAD_CONDITION_TYPE; SQLSTATE: HY000

  Message: SIGNAL/RESIGNAL can only use a CONDITION defined with SQLSTATE

- Error number: 1647; Symbol: WARN_COND_ITEM_TRUNCATED; SQLSTATE: HY000

  Message: Data truncated for condition item '%s'

- Error number: 1648; Symbol: ER_COND_ITEM_TOO_LONG; SQLSTATE: HY000

  Message: Data too long for condition item '%s'

- Error number: 1649; Symbol: ER_UNKNOWN_LOCALE; SQLSTATE: HY000

Message: Unknown locale: '%s'

- Error number: 1650; Symbol: ER_SLAVE_IGNORE_SERVER_IDS; SQLSTATE: HY000

  Message: The requested server id %d clashes with the slave startup option --replicate-same-server-id

- Error number: 1651; Symbol: ER_QUERY_CACHE_DISABLED; SQLSTATE: HY000

  Message: Query cache is disabled; restart the server with query_cache_type=1 to enable it

  ER_QUERY_CACHE_DISABLED was removed after 8.0.2.

- Error number: 1652; Symbol: ER_SAME_NAME_PARTITION_FIELD; SQLSTATE: HY000

  Message: Duplicate partition field name '%s'

- Error number: 1653; Symbol: ER_PARTITION_COLUMN_LIST_ERROR; SQLSTATE: HY000

  Message: Inconsistency in usage of column lists for partitioning

- Error number: 1654; Symbol: ER_WRONG_TYPE_COLUMN_VALUE_ERROR; SQLSTATE: HY000

  Message: Partition column values of incorrect type

- Error number: 1655; Symbol: ER_TOO_MANY_PARTITION_FUNC_FIELDS_ERROR; SQLSTATE: HY000

  Message: Too many fields in '%s'

- Error number: 1656; Symbol: ER_MAXVALUE_IN_VALUES_IN; SQLSTATE: HY000

  Message: Cannot use MAXVALUE as value in VALUES IN

- Error number: 1657; Symbol: ER_TOO_MANY_VALUES_ERROR; SQLSTATE: HY000

  Message: Cannot have more than one value for this type of %s partitioning

- Error number: 1658; Symbol: ER_ROW_SINGLE_PARTITION_FIELD_ERROR; SQLSTATE: HY000

  Message: Row expressions in VALUES IN only allowed for multi-field column partitioning

- Error number: 1659; Symbol: ER_FIELD_TYPE_NOT_ALLOWED_AS_PARTITION_FIELD; SQLSTATE: HY000

  Message: Field '%s' is of a not allowed type for this type of partitioning

- Error number: 1660; Symbol: ER_PARTITION_FIELDS_TOO_LONG; SQLSTATE: HY000

  Message: The total length of the partitioning fields is too large

- Error number: 1661; Symbol: ER_BINLOG_ROW_ENGINE_AND_STMT_ENGINE; SQLSTATE: HY000

  Message: Cannot execute statement: impossible to write to binary log since both row-incapable engines and statement-incapable engines are involved.

- Error number: 1662; Symbol: ER_BINLOG_ROW_MODE_AND_STMT_ENGINE; SQLSTATE: HY000

  Message: Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = ROW and at least one table uses a storage engine limited to statement-based logging.

- Error number: 1663; Symbol: ER_BINLOG_UNSAFE_AND_STMT_ENGINE; SQLSTATE: HY000

  Message: Cannot execute statement: impossible to write to binary log since statement is unsafe, storage engine is limited to statement-based logging, and BINLOG_FORMAT = MIXED. %s

- Error number: 1664; Symbol: ER_BINLOG_ROW_INJECTION_AND_STMT_ENGINE; SQLSTATE: HY000

  Message: Cannot execute statement: impossible to write to binary log since statement is in row format and at least one table uses a storage engine limited to statement-based logging.

- Error number: 1665; Symbol: ER_BINLOG_STMT_MODE_AND_ROW_ENGINE; SQLSTATE: HY000

  Message: Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine limited to row-based logging.%s

- Error number: 1666; Symbol: ER_BINLOG_ROW_INJECTION_AND_STMT_MODE; SQLSTATE: HY000

  Message: Cannot execute statement: impossible to write to binary log since statement is in row format and BINLOG_FORMAT = STATEMENT.

- Error number: 1667; Symbol: ER_BINLOG_MULTIPLE_ENGINES_AND_SELF_LOGGING_ENGINE; SQLSTATE: HY000

  Message: Cannot execute statement: impossible to write to binary log since more than one engine is involved and at least one engine is self-logging.

- Error number: 1668; Symbol: ER_BINLOG_UNSAFE_LIMIT; SQLSTATE: HY000

  Message: The statement is unsafe because it uses a LIMIT clause. This is unsafe because the set of rows included cannot be predicted.

- Error number: 1670; Symbol: ER_BINLOG_UNSAFE_SYSTEM_TABLE; SQLSTATE: HY000

  Message: The statement is unsafe because it uses the general log, slow query log, or performance_schema table(s). This is unsafe because system tables may differ on slaves.

- Error number: 1671; Symbol: ER_BINLOG_UNSAFE_AUTOINC_COLUMNS; SQLSTATE: HY000

  Message: Statement is unsafe because it invokes a trigger or a stored function that inserts into an AUTO_INCREMENT column. Inserted values cannot be logged correctly.

- Error number: 1672; Symbol: ER_BINLOG_UNSAFE_UDF; SQLSTATE: HY000

  Message: Statement is unsafe because it uses a UDF which may not return the same value on the slave.

- Error number: 1673; Symbol: ER_BINLOG_UNSAFE_SYSTEM_VARIABLE; SQLSTATE: HY000

  Message: Statement is unsafe because it uses a system variable that may have a different value on the slave.

- Error number: 1674; Symbol: ER_BINLOG_UNSAFE_SYSTEM_FUNCTION; SQLSTATE: HY000

  Message: Statement is unsafe because it uses a system function that may return a different value on the slave.

- Error number: 1675; Symbol: ER_BINLOG_UNSAFE_NONTRANS_AFTER_TRANS; SQLSTATE: HY000

Message: Statement is unsafe because it accesses a non-transactional table after accessing a transactional table within the same transaction.

- Error number: 1676; Symbol: ER_MESSAGE_AND_STATEMENT; SQLSTATE: HY000

  Message: %s Statement: %s

- Error number: 1677; Symbol: ER_SLAVE_CONVERSION_FAILED; SQLSTATE: HY000

  Message: Column %d of table '%s.%s' cannot be converted from type '%s' to type '%s'

  ER_SLAVE_CONVERSION_FAILED was removed after 8.0.4.

- Error number: 1678; Symbol: ER_SLAVE_CANT_CREATE_CONVERSION; SQLSTATE: HY000

  Message: Can't create conversion table for table '%s.%s'

- Error number: 1679; Symbol: ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_BINLOG_FORMAT; SQLSTATE: HY000

  Message: Cannot modify @@session.binlog_format inside a transaction

- Error number: 1680; Symbol: ER_PATH_LENGTH; SQLSTATE: HY000

  Message: The path specified for %s is too long.

- Error number: 1681; Symbol: ER_WARN_DEPRECATED_SYNTAX_NO_REPLACEMENT; SQLSTATE: HY000

  Message: '%s' is deprecated and will be removed in a future release.

- Error number: 1682; Symbol: ER_WRONG_NATIVE_TABLE_STRUCTURE; SQLSTATE: HY000

  Message: Native table '%s'.'%s' has the wrong structure

- Error number: 1683; Symbol: ER_WRONG_PERFSCHEMA_USAGE; SQLSTATE: HY000

  Message: Invalid performance_schema usage.

- Error number: 1684; Symbol: ER_WARN_I_S_SKIPPED_TABLE; SQLSTATE: HY000

  Message: Table '%s'.'%s' was skipped since its definition is being modified by concurrent DDL statement

- Error number: 1685; Symbol: ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_BINLOG_DIRECT; SQLSTATE: HY000

  Message: Cannot modify @@session.binlog_direct_non_transactional_updates inside a transaction

- Error number: 1686; Symbol: ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_DIRECT; SQLSTATE: HY000

  Message: Cannot change the binlog direct flag inside a stored function or trigger

- Error number: 1687; Symbol: ER_SPATIAL_MUST_HAVE_GEOM_COL; SQLSTATE: 42000

  Message: A SPATIAL index may only contain a geometrical type column

- Error number: 1688; Symbol: ER_TOO_LONG_INDEX_COMMENT; SQLSTATE: HY000

Message: Comment for index '%s' is too long (max = %lu)

- Error number: 1689; Symbol: ER_LOCK_ABORTED; SQLSTATE: HY000

  Message: Wait on a lock was aborted due to a pending exclusive lock

- Error number: 1690; Symbol: ER_DATA_OUT_OF_RANGE; SQLSTATE: 22003

  Message: %s value is out of range in '%s'

- Error number: 1691; Symbol: ER_WRONG_SPVAR_TYPE_IN_LIMIT; SQLSTATE: HY000

  Message: A variable of a non-integer based type in LIMIT clause

- Error number: 1692; Symbol: ER_BINLOG_UNSAFE_MULTIPLE_ENGINES_AND_SELF_LOGGING_ENGINE; SQLSTATE: HY000

  Message: Mixing self-logging and non-self-logging engines in a statement is unsafe.

- Error number: 1693; Symbol: ER_BINLOG_UNSAFE_MIXED_STATEMENT; SQLSTATE: HY000

  Message: Statement accesses nontransactional table as well as transactional or temporary table, and writes to any of them.

- Error number: 1694; Symbol: ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_SQL_LOG_BIN; SQLSTATE: HY000

  Message: Cannot modify @@session.sql_log_bin inside a transaction

- Error number: 1695; Symbol: ER_STORED_FUNCTION_PREVENTS_SWITCH_SQL_LOG_BIN; SQLSTATE: HY000

  Message: Cannot change the sql_log_bin inside a stored function or trigger

- Error number: 1696; Symbol: ER_FAILED_READ_FROM_PAR_FILE; SQLSTATE: HY000

  Message: Failed to read from the .par file

- Error number: 1697; Symbol: ER_VALUES_IS_NOT_INT_TYPE_ERROR; SQLSTATE: HY000

  Message: VALUES value for partition '%s' must have type INT

- Error number: 1698; Symbol: ER_ACCESS_DENIED_NO_PASSWORD_ERROR; SQLSTATE: 28000

  Message: Access denied for user '%s'@'%s'

- Error number: 1699; Symbol: ER_SET_PASSWORD_AUTH_PLUGIN; SQLSTATE: HY000

  Message: SET PASSWORD has no significance for users authenticating via plugins

- Error number: 1701; Symbol: ER_TRUNCATE_ILLEGAL_FK; SQLSTATE: 42000

  Message: Cannot truncate a table referenced in a foreign key constraint (%s)

- Error number: 1702; Symbol: ER_PLUGIN_IS_PERMANENT; SQLSTATE: HY000

  Message: Plugin '%s' is force_plus_permanent and can not be unloaded

- Error number: 1703; Symbol: ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MIN; SQLSTATE: HY000

Message: The requested value for the heartbeat period is less than 1 millisecond. The value is reset to 0, meaning that heartbeating will effectively be disabled.

- Error number: 1704; Symbol: ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MAX; SQLSTATE: HY000

  Message: The requested value for the heartbeat period exceeds the value of `slave_net_timeout' seconds. A sensible value for the period should be less than the timeout.

- Error number: 1705; Symbol: ER_STMT_CACHE_FULL; SQLSTATE: HY000

  Message: Multi-row statements required more than 'max_binlog_stmt_cache_size' bytes of storage; increase this mysqld variable and try again

- Error number: 1706; Symbol: ER_MULTI_UPDATE_KEY_CONFLICT; SQLSTATE: HY000

  Message: Primary key/partition key update is not allowed since the table is updated both as '%s' and '%s'.

- Error number: 1707; Symbol: ER_TABLE_NEEDS_REBUILD; SQLSTATE: HY000

  Message: Table rebuild required. Please do "ALTER TABLE `%s` FORCE" or dump/reload to fix it!

- Error number: 1708; Symbol: WARN_OPTION_BELOW_LIMIT; SQLSTATE: HY000

  Message: The value of '%s' should be no less than the value of '%s'

- Error number: 1709; Symbol: ER_INDEX_COLUMN_TOO_LONG; SQLSTATE: HY000

  Message: Index column size too large. The maximum column size is %lu bytes.

- Error number: 1710; Symbol: ER_ERROR_IN_TRIGGER_BODY; SQLSTATE: HY000

  Message: Trigger '%s' has an error in its body: '%s'

- Error number: 1711; Symbol: ER_ERROR_IN_UNKNOWN_TRIGGER_BODY; SQLSTATE: HY000

  Message: Unknown trigger has an error in its body: '%s'

- Error number: 1712; Symbol: ER_INDEX_CORRUPT; SQLSTATE: HY000

  Message: Index %s is corrupted

- Error number: 1713; Symbol: ER_UNDO_RECORD_TOO_BIG; SQLSTATE: HY000

  Message: Undo log record is too big.

- Error number: 1714; Symbol: ER_BINLOG_UNSAFE_INSERT_IGNORE_SELECT; SQLSTATE: HY000

  Message: INSERT IGNORE... SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are ignored. This order cannot be predicted and may differ on master and the slave.

- Error number: 1715; Symbol: ER_BINLOG_UNSAFE_INSERT_SELECT_UPDATE; SQLSTATE: HY000

  Message: INSERT... SELECT... ON DUPLICATE KEY UPDATE is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are updated. This order cannot be predicted and may differ on master and the slave.

- Error number: 1716; Symbol: ER_BINLOG_UNSAFE_REPLACE_SELECT; SQLSTATE: HY000

  Message: REPLACE... SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are replaced. This order cannot be predicted and may differ on master and the slave.

- Error number: 1717; Symbol: ER_BINLOG_UNSAFE_CREATE_IGNORE_SELECT; SQLSTATE: HY000

  Message: CREATE... IGNORE SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are ignored. This order cannot be predicted and may differ on master and the slave.

- Error number: 1718; Symbol: ER_BINLOG_UNSAFE_CREATE_REPLACE_SELECT; SQLSTATE: HY000

  Message: CREATE... REPLACE SELECT is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are replaced. This order cannot be predicted and may differ on master and the slave.

- Error number: 1719; Symbol: ER_BINLOG_UNSAFE_UPDATE_IGNORE; SQLSTATE: HY000

  Message: UPDATE IGNORE is unsafe because the order in which rows are updated determines which (if any) rows are ignored. This order cannot be predicted and may differ on master and the slave.

- Error number: 1720; Symbol: ER_PLUGIN_NO_UNINSTALL; SQLSTATE: HY000

  Message: Plugin '%s' is marked as not dynamically uninstallable. You have to stop the server to uninstall it.

- Error number: 1721; Symbol: ER_PLUGIN_NO_INSTALL; SQLSTATE: HY000

  Message: Plugin '%s' is marked as not dynamically installable. You have to stop the server to install it.

- Error number: 1722; Symbol: ER_BINLOG_UNSAFE_WRITE_AUTOINC_SELECT; SQLSTATE: HY000

  Message: Statements writing to a table with an auto-increment column after selecting from another table are unsafe because the order in which rows are retrieved determines what (if any) rows will be written. This order cannot be predicted and may differ on master and the slave.

- Error number: 1723; Symbol: ER_BINLOG_UNSAFE_CREATE_SELECT_AUTOINC; SQLSTATE: HY000

  Message: CREATE TABLE... SELECT... on a table with an auto-increment column is unsafe because the order in which rows are retrieved by the SELECT determines which (if any) rows are inserted. This order cannot be predicted and may differ on master and the slave.

- Error number: 1724; Symbol: ER_BINLOG_UNSAFE_INSERT_TWO_KEYS; SQLSTATE: HY000

  Message: INSERT... ON DUPLICATE KEY UPDATE on a table with more than one UNIQUE KEY is unsafe

- Error number: 1725; Symbol: ER_TABLE_IN_FK_CHECK; SQLSTATE: HY000

  Message: Table is being used in foreign key check.

- Error number: 1726; Symbol: ER_UNSUPPORTED_ENGINE; SQLSTATE: HY000

  Message: Storage engine '%s' does not support system tables. [%s.%s]

- Error number: 1727; Symbol: ER_BINLOG_UNSAFE_AUTOINC_NOT_FIRST; SQLSTATE: HY000

Message: INSERT into autoincrement field which is not the first part in the composed primary key is unsafe.

- Error number: 1728; Symbol: ER_CANNOT_LOAD_FROM_TABLE_V2; SQLSTATE: HY000

  Message: Cannot load from %s.%s. The table is probably corrupted

- Error number: 1729; Symbol: ER_MASTER_DELAY_VALUE_OUT_OF_RANGE; SQLSTATE: HY000

  Message: The requested value %s for the master delay exceeds the maximum %u

- Error number: 1730; Symbol: ER_ONLY_FD_AND_RBR_EVENTS_ALLOWED_IN_BINLOG_STATEMENT; SQLSTATE: HY000

  Message: Only Format_description_log_event and row events are allowed in BINLOG statements (but %s was provided)

- Error number: 1731; Symbol: ER_PARTITION_EXCHANGE_DIFFERENT_OPTION; SQLSTATE: HY000

  Message: Non matching attribute '%s' between partition and table

- Error number: 1732; Symbol: ER_PARTITION_EXCHANGE_PART_TABLE; SQLSTATE: HY000

  Message: Table to exchange with partition is partitioned: '%s'

- Error number: 1733; Symbol: ER_PARTITION_EXCHANGE_TEMP_TABLE; SQLSTATE: HY000

  Message: Table to exchange with partition is temporary: '%s'

- Error number: 1734; Symbol: ER_PARTITION_INSTEAD_OF_SUBPARTITION; SQLSTATE: HY000

  Message: Subpartitioned table, use subpartition instead of partition

- Error number: 1735; Symbol: ER_UNKNOWN_PARTITION; SQLSTATE: HY000

  Message: Unknown partition '%s' in table '%s'

- Error number: 1736; Symbol: ER_TABLES_DIFFERENT_METADATA; SQLSTATE: HY000

  Message: Tables have different definitions

- Error number: 1737; Symbol: ER_ROW_DOES_NOT_MATCH_PARTITION; SQLSTATE: HY000

  Message: Found a row that does not match the partition

- Error number: 1738; Symbol: ER_BINLOG_CACHE_SIZE_GREATER_THAN_MAX; SQLSTATE: HY000

  Message: Option binlog_cache_size (%lu) is greater than max_binlog_cache_size (%lu); setting binlog_cache_size equal to max_binlog_cache_size.

- Error number: 1739; Symbol: ER_WARN_INDEX_NOT_APPLICABLE; SQLSTATE: HY000

  Message: Cannot use %s access on index '%s' due to type or collation conversion on field '%s'

- Error number: 1740; Symbol: ER_PARTITION_EXCHANGE_FOREIGN_KEY; SQLSTATE: HY000

  Message: Table to exchange with partition has foreign key references: '%s'

- Error number: 1742; Symbol: ER_RPL_INFO_DATA_TOO_LONG; SQLSTATE: HY000

Message: Data for column '%s' too long

- Error number: 1745; Symbol: ER_BINLOG_STMT_CACHE_SIZE_GREATER_THAN_MAX; SQLSTATE: HY000

  Message: Option binlog_stmt_cache_size (%lu) is greater than max_binlog_stmt_cache_size (%lu); setting binlog_stmt_cache_size equal to max_binlog_stmt_cache_size.

- Error number: 1746; Symbol: ER_CANT_UPDATE_TABLE_IN_CREATE_TABLE_SELECT; SQLSTATE: HY000

  Message: Can't update table '%s' while '%s' is being created.

- Error number: 1747; Symbol: ER_PARTITION_CLAUSE_ON_NONPARTITIONED; SQLSTATE: HY000

  Message: PARTITION () clause on non partitioned table

- Error number: 1748; Symbol: ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET; SQLSTATE: HY000

  Message: Found a row not matching the given partition set

- Error number: 1750; Symbol: ER_CHANGE_RPL_INFO_REPOSITORY_FAILURE; SQLSTATE: HY000

  Message: Failure while changing the type of replication repository: %s.

- Error number: 1751; Symbol: ER_WARNING_NOT_COMPLETE_ROLLBACK_WITH_CREATED_TEMP_TABLE; SQLSTATE: HY000

  Message: The creation of some temporary tables could not be rolled back.

- Error number: 1752; Symbol: ER_WARNING_NOT_COMPLETE_ROLLBACK_WITH_DROPPED_TEMP_TABLE; SQLSTATE: HY000

  Message: Some temporary tables were dropped, but these operations could not be rolled back.

- Error number: 1753; Symbol: ER_MTS_FEATURE_IS_NOT_SUPPORTED; SQLSTATE: HY000

  Message: %s is not supported in multi-threaded slave mode. %s

- Error number: 1754; Symbol: ER_MTS_UPDATED_DBS_GREATER_MAX; SQLSTATE: HY000

  Message: The number of modified databases exceeds the maximum %d; the database names will not be included in the replication event metadata.

- Error number: 1755; Symbol: ER_MTS_CANT_PARALLEL; SQLSTATE: HY000

  Message: Cannot execute the current event group in the parallel mode. Encountered event %s, relay-log name %s, position %s which prevents execution of this event group in parallel mode. Reason: %s.

- Error number: 1756; Symbol: ER_MTS_INCONSISTENT_DATA; SQLSTATE: HY000

  Message: %s

- Error number: 1757; Symbol: ER_FULLTEXT_NOT_SUPPORTED_WITH_PARTITIONING; SQLSTATE: HY000

  Message: FULLTEXT index is not supported for partitioned tables.

- Error number: 1758; Symbol: ER_DA_INVALID_CONDITION_NUMBER; SQLSTATE: 35000

  Message: Invalid condition number

- Error number: 1759; Symbol: ER_INSECURE_PLAIN_TEXT; SQLSTATE: HY000

  Message: Sending passwords in plain text without SSL/TLS is extremely insecure.

- Error number: 1760; Symbol: ER_INSECURE_CHANGE_MASTER; SQLSTATE: HY000

  Message: Storing MySQL user name or password information in the master info repository is not secure and is therefore not recommended. Please consider using the USER and PASSWORD connection options for START SLAVE; see the 'START SLAVE Syntax' in the MySQL Manual for more information.

- Error number: 1761; Symbol: ER_FOREIGN_DUPLICATE_KEY_WITH_CHILD_INFO; SQLSTATE: 23000

  Message: Foreign key constraint for table '%s', record '%s' would lead to a duplicate entry in table '%s', key '%s'

- Error number: 1762; Symbol: ER_FOREIGN_DUPLICATE_KEY_WITHOUT_CHILD_INFO; SQLSTATE: 23000

  Message: Foreign key constraint for table '%s', record '%s' would lead to a duplicate entry in a child table

- Error number: 1763; Symbol: ER_SQLTHREAD_WITH_SECURE_SLAVE; SQLSTATE: HY000

  Message: Setting authentication options is not possible when only the Slave SQL Thread is being started.

- Error number: 1764; Symbol: ER_TABLE_HAS_NO_FT; SQLSTATE: HY000

  Message: The table does not have FULLTEXT index to support this query

- Error number: 1765; Symbol: ER_VARIABLE_NOT_SETTABLE_IN_SF_OR_TRIGGER; SQLSTATE: HY000

  Message: The system variable %s cannot be set in stored functions or triggers.

- Error number: 1766; Symbol: ER_VARIABLE_NOT_SETTABLE_IN_TRANSACTION; SQLSTATE: HY000

  Message: The system variable %s cannot be set when there is an ongoing transaction.

- Error number: 1769; Symbol: ER_SET_STATEMENT_CANNOT_INVOKE_FUNCTION; SQLSTATE: HY000

  Message: The statement 'SET %s' cannot invoke a stored function.

- Error number: 1770; Symbol: ER_GTID_NEXT_CANT_BE_AUTOMATIC_IF_GTID_NEXT_LIST_IS_NON_NULL; SQLSTATE: HY000

  Message: The system variable @@SESSION.GTID_NEXT cannot be 'AUTOMATIC' when @@SESSION.GTID_NEXT_LIST is non-NULL.

- Error number: 1772; Symbol: ER_MALFORMED_GTID_SET_SPECIFICATION; SQLSTATE: HY000

  Message: Malformed GTID set specification '%s'.

- Error number: 1773; Symbol: ER_MALFORMED_GTID_SET_ENCODING; SQLSTATE: HY000

  Message: Malformed GTID set encoding.

- Error number: 1774; Symbol: ER_MALFORMED_GTID_SPECIFICATION; SQLSTATE: HY000

  Message: Malformed GTID specification '%s'.

- Error number: 1775; Symbol: ER_GNO_EXHAUSTED; SQLSTATE: HY000

  Message: Impossible to generate Global Transaction Identifier: the integer component reached the maximal value. Restart the server with a new server_uuid.

- Error number: 1776; Symbol: ER_BAD_SLAVE_AUTO_POSITION; SQLSTATE: HY000

  Message: Parameters MASTER_LOG_FILE, MASTER_LOG_POS, RELAY_LOG_FILE and RELAY_LOG_POS cannot be set when MASTER_AUTO_POSITION is active.

- Error number: 1777; Symbol: ER_AUTO_POSITION_REQUIRES_GTID_MODE_NOT_OFF; SQLSTATE: HY000

  Message: CHANGE MASTER TO MASTER_AUTO_POSITION = 1 cannot be executed because @@GLOBAL.GTID_MODE = OFF.

- Error number: 1778; Symbol: ER_CANT_DO_IMPLICIT_COMMIT_IN_TRX_WHEN_GTID_NEXT_IS_SET; SQLSTATE: HY000

  Message: Cannot execute statements with implicit commit inside a transaction when @@SESSION.GTID_NEXT == 'UUID:NUMBER'.

- Error number: 1779; Symbol: ER_GTID_MODE_ON_REQUIRES_ENFORCE_GTID_CONSISTENCY_ON; SQLSTATE: HY000

  Message: GTID_MODE = ON requires ENFORCE_GTID_CONSISTENCY = ON.

- Error number: 1781; Symbol: ER_CANT_SET_GTID_NEXT_TO_GTID_WHEN_GTID_MODE_IS_OFF; SQLSTATE: HY000

  Message: @@SESSION.GTID_NEXT cannot be set to UUID:NUMBER when @@GLOBAL.GTID_MODE = OFF.

- Error number: 1782; Symbol: ER_CANT_SET_GTID_NEXT_TO_ANONYMOUS_WHEN_GTID_MODE_IS_ON; SQLSTATE: HY000

  Message: @@SESSION.GTID_NEXT cannot be set to ANONYMOUS when @@GLOBAL.GTID_MODE = ON.

- Error number: 1783; Symbol: ER_CANT_SET_GTID_NEXT_LIST_TO_NON_NULL_WHEN_GTID_MODE_IS_OFF; SQLSTATE: HY000

  Message: @@SESSION.GTID_NEXT_LIST cannot be set to a non-NULL value when @@GLOBAL.GTID_MODE = OFF.

- Error number: 1785; Symbol: ER_GTID_UNSAFE_NON_TRANSACTIONAL_TABLE; SQLSTATE: HY000

  Message: Statement violates GTID consistency: Updates to non-transactional tables can only be done in either autocommitted statements or single-statement transactions, and never in the same statement as updates to transactional tables.

- Error number: 1786; Symbol: ER_GTID_UNSAFE_CREATE_SELECT; SQLSTATE: HY000

  Message: Statement violates GTID consistency: CREATE TABLE ... SELECT.

- Error number: 1787; Symbol: ER_GTID_UNSAFE_CREATE_DROP_TEMPORARY_TABLE_IN_TRANSACTION; SQLSTATE: HY000

  Message: Statement violates GTID consistency: CREATE TEMPORARY TABLE and DROP TEMPORARY TABLE can only be executed outside transactional context. These statements are also not allowed in a function or trigger because functions and triggers are also considered to be multi-statement transactions.

  ER_GTID_UNSAFE_CREATE_DROP_TEMPORARY_TABLE_IN_TRANSACTION was removed after 8.0.12.

- Error number: 1788; Symbol: ER_GTID_MODE_CAN_ONLY_CHANGE_ONE_STEP_AT_A_TIME; SQLSTATE: HY000

  Message: The value of @@GLOBAL.GTID_MODE can only be changed one step at a time: OFF <-> OFF_PERMISSIVE <-> ON_PERMISSIVE <-> ON. Also note that this value must be stepped up or down simultaneously on all servers. See the Manual for instructions.

- Error number: 1789; Symbol: ER_MASTER_HAS_PURGED_REQUIRED_GTIDS; SQLSTATE: HY000

  Message: Cannot replicate because the master purged required binary logs. Replicate the missing transactions from elsewhere, or provision a new slave from backup. Consider increasing the master's binary log expiration period. To find the missing transactions, see the master's error log or the manual for GTID_SUBTRACT.

- Error number: 1790; Symbol: ER_CANT_SET_GTID_NEXT_WHEN_OWNING_GTID; SQLSTATE: HY000

  Message: @@SESSION.GTID_NEXT cannot be changed by a client that owns a GTID. The client owns %s. Ownership is released on COMMIT or ROLLBACK.

- Error number: 1791; Symbol: ER_UNKNOWN_EXPLAIN_FORMAT; SQLSTATE: HY000

  Message: Unknown EXPLAIN format name: '%s'

- Error number: 1792; Symbol: ER_CANT_EXECUTE_IN_READ_ONLY_TRANSACTION; SQLSTATE: 25006

  Message: Cannot execute statement in a READ ONLY transaction.

- Error number: 1793; Symbol: ER_TOO_LONG_TABLE_PARTITION_COMMENT; SQLSTATE: HY000

  Message: Comment for table partition '%s' is too long (max = %lu)

- Error number: 1794; Symbol: ER_SLAVE_CONFIGURATION; SQLSTATE: HY000

  Message: Slave is not configured or failed to initialize properly. You must at least set --server-id to enable either a master or a slave. Additional error messages can be found in the MySQL error log.

- Error number: 1795; Symbol: ER_INNODB_FT_LIMIT; SQLSTATE: HY000

  Message: InnoDB presently supports one FULLTEXT index creation at a time

- Error number: 1796; Symbol: ER_INNODB_NO_FT_TEMP_TABLE; SQLSTATE: HY000

  Message: Cannot create FULLTEXT index on temporary InnoDB table

- Error number: 1797; Symbol: ER_INNODB_FT_WRONG_DOCID_COLUMN; SQLSTATE: HY000

  Message: Column '%s' is of wrong type for an InnoDB FULLTEXT index

- Error number: 1798; Symbol: ER_INNODB_FT_WRONG_DOCID_INDEX; SQLSTATE: HY000

  Message: Index '%s' is of wrong type for an InnoDB FULLTEXT index

- Error number: 1799; Symbol: ER_INNODB_ONLINE_LOG_TOO_BIG; SQLSTATE: HY000

  Message: Creating index '%s' required more than 'innodb_online_alter_log_max_size' bytes of modification log. Please try again.

- Error number: 1800; Symbol: ER_UNKNOWN_ALTER_ALGORITHM; SQLSTATE: HY000

  Message: Unknown ALGORITHM '%s'

- Error number: 1801; Symbol: ER_UNKNOWN_ALTER_LOCK; SQLSTATE: HY000

  Message: Unknown LOCK type '%s'

- Error number: 1802; Symbol: ER_MTS_CHANGE_MASTER_CANT_RUN_WITH_GAPS; SQLSTATE: HY000

  Message: CHANGE MASTER cannot be executed when the slave was stopped with an error or killed in MTS mode. Consider using RESET SLAVE or START SLAVE UNTIL.

- Error number: 1803; Symbol: ER_MTS_RECOVERY_FAILURE; SQLSTATE: HY000

  Message: Cannot recover after SLAVE errored out in parallel execution mode. Additional error messages can be found in the MySQL error log.

- Error number: 1804; Symbol: ER_MTS_RESET_WORKERS; SQLSTATE: HY000

  Message: Cannot clean up worker info tables. Additional error messages can be found in the MySQL error log.

- Error number: 1805; Symbol: ER_COL_COUNT_DOESNT_MATCH_CORRUPTED_V2; SQLSTATE: HY000

  Message: Column count of %s.%s is wrong. Expected %d, found %d. The table is probably corrupted

- Error number: 1806; Symbol: ER_SLAVE_SILENT_RETRY_TRANSACTION; SQLSTATE: HY000

  Message: Slave must silently retry current transaction

- Error number: 1807; Symbol: ER_DISCARD_FK_CHECKS_RUNNING; SQLSTATE: HY000

  Message: There is a foreign key check running on table '%s'. Cannot discard the table.

- Error number: 1808; Symbol: ER_TABLE_SCHEMA_MISMATCH; SQLSTATE: HY000

  Message: Schema mismatch (%s)

- Error number: 1809; Symbol: ER_TABLE_IN_SYSTEM_TABLESPACE; SQLSTATE: HY000

  Message: Table '%s' in system tablespace

- Error number: 1810; Symbol: ER_IO_READ_ERROR; SQLSTATE: HY000

  Message: IO Read error: (%lu, %s) %s

- Error number: 1811; Symbol: ER_IO_WRITE_ERROR; SQLSTATE: HY000

  Message: IO Write error: (%lu, %s) %s

- Error number: 1812; Symbol: ER_TABLESPACE_MISSING; SQLSTATE: HY000

  Message: Tablespace is missing for table %s.

- Error number: 1813; Symbol: ER_TABLESPACE_EXISTS; SQLSTATE: HY000

  Message: Tablespace '%s' exists.

- Error number: 1814; Symbol: ER_TABLESPACE_DISCARDED; SQLSTATE: HY000

  Message: Tablespace has been discarded for table '%s'

- Error number: 1815; Symbol: ER_INTERNAL_ERROR; SQLSTATE: HY000

  Message: Internal error: %s

- Error number: 1816; Symbol: ER_INNODB_IMPORT_ERROR; SQLSTATE: HY000

  Message: ALTER TABLE %s IMPORT TABLESPACE failed with error %lu : '%s'

- Error number: 1817; Symbol: ER_INNODB_INDEX_CORRUPT; SQLSTATE: HY000

  Message: Index corrupt: %s

- Error number: 1818; Symbol: ER_INVALID_YEAR_COLUMN_LENGTH; SQLSTATE: HY000

  Message: Supports only YEAR or YEAR(4) column.

- Error number: 1819; Symbol: ER_NOT_VALID_PASSWORD; SQLSTATE: HY000

  Message: Your password does not satisfy the current policy requirements

- Error number: 1820; Symbol: ER_MUST_CHANGE_PASSWORD; SQLSTATE: HY000

  Message: You must reset your password using ALTER USER statement before executing this statement.

- Error number: 1821; Symbol: ER_FK_NO_INDEX_CHILD; SQLSTATE: HY000

  Message: Failed to add the foreign key constraint. Missing index for constraint '%s' in the foreign table '%s'

- Error number: 1822; Symbol: ER_FK_NO_INDEX_PARENT; SQLSTATE: HY000

  Message: Failed to add the foreign key constraint. Missing index for constraint '%s' in the referenced table '%s'

- Error number: 1823; Symbol: ER_FK_FAIL_ADD_SYSTEM; SQLSTATE: HY000

  Message: Failed to add the foreign key constraint '%s' to system tables

- Error number: 1824; Symbol: ER_FK_CANNOT_OPEN_PARENT; SQLSTATE: HY000

  Message: Failed to open the referenced table '%s'

- Error number: 1825; Symbol: ER_FK_INCORRECT_OPTION; SQLSTATE: HY000

Message: Failed to add the foreign key constraint on table '%s'. Incorrect options in FOREIGN KEY constraint '%s'

- Error number: 1826; Symbol: ER_FK_DUP_NAME; SQLSTATE: HY000

  Message: Duplicate foreign key constraint name '%s'

- Error number: 1827; Symbol: ER_PASSWORD_FORMAT; SQLSTATE: HY000

  Message: The password hash doesn't have the expected format.

- Error number: 1828; Symbol: ER_FK_COLUMN_CANNOT_DROP; SQLSTATE: HY000

  Message: Cannot drop column '%s': needed in a foreign key constraint '%s'

- Error number: 1829; Symbol: ER_FK_COLUMN_CANNOT_DROP_CHILD; SQLSTATE: HY000

  Message: Cannot drop column '%s': needed in a foreign key constraint '%s' of table '%s'

- Error number: 1830; Symbol: ER_FK_COLUMN_NOT_NULL; SQLSTATE: HY000

  Message: Column '%s' cannot be NOT NULL: needed in a foreign key constraint '%s' SET NULL

- Error number: 1831; Symbol: ER_DUP_INDEX; SQLSTATE: HY000

  Message: Duplicate index '%s' defined on the table '%s.%s'. This is deprecated and will be disallowed in a future release.

- Error number: 1832; Symbol: ER_FK_COLUMN_CANNOT_CHANGE; SQLSTATE: HY000

  Message: Cannot change column '%s': used in a foreign key constraint '%s'

- Error number: 1833; Symbol: ER_FK_COLUMN_CANNOT_CHANGE_CHILD; SQLSTATE: HY000

  Message: Cannot change column '%s': used in a foreign key constraint '%s' of table '%s'

- Error number: 1835; Symbol: ER_MALFORMED_PACKET; SQLSTATE: HY000

  Message: Malformed communication packet.

- Error number: 1836; Symbol: ER_READ_ONLY_MODE; SQLSTATE: HY000

  Message: Running in read-only mode

- Error number: 1837; Symbol: ER_GTID_NEXT_TYPE_UNDEFINED_GROUP; SQLSTATE: HY000

  Message: When @@SESSION.GTID_NEXT is set to a GTID, you must explicitly set it to a different value after a COMMIT or ROLLBACK. Please check GTID_NEXT variable manual page for detailed explanation. Current @@SESSION.GTID_NEXT is '%s'.

  ER_GTID_NEXT_TYPE_UNDEFINED_GROUP was removed after 8.0.4.

- Error number: 1837; Symbol: ER_GTID_NEXT_TYPE_UNDEFINED_GTID; SQLSTATE: HY000

  Message: When @@SESSION.GTID_NEXT is set to a GTID, you must explicitly set it to a different value after a COMMIT or ROLLBACK. Please check GTID_NEXT variable manual page for detailed explanation. Current @@SESSION.GTID_NEXT is '%s'.

  ER_GTID_NEXT_TYPE_UNDEFINED_GTID was added in 8.0.11.

- Error number: 1838; Symbol: ER_VARIABLE_NOT_SETTABLE_IN_SP; SQLSTATE: HY000

  Message: The system variable %s cannot be set in stored procedures.

- Error number: 1840; Symbol: ER_CANT_SET_GTID_PURGED_WHEN_GTID_EXECUTED_IS_NOT_EMPTY; SQLSTATE: HY000

  Message: @@GLOBAL.GTID_PURGED can only be set when @@GLOBAL.GTID_EXECUTED is empty.

- Error number: 1841; Symbol: ER_CANT_SET_GTID_PURGED_WHEN_OWNED_GTIDS_IS_NOT_EMPTY; SQLSTATE: HY000

  Message: @@GLOBAL.GTID_PURGED can only be set when there are no ongoing transactions (not even in other clients).

- Error number: 1842; Symbol: ER_GTID_PURGED_WAS_CHANGED; SQLSTATE: HY000

  Message: @@GLOBAL.GTID_PURGED was changed from '%s' to '%s'.

- Error number: 1843; Symbol: ER_GTID_EXECUTED_WAS_CHANGED; SQLSTATE: HY000

  Message: @@GLOBAL.GTID_EXECUTED was changed from '%s' to '%s'.

- Error number: 1844; Symbol: ER_BINLOG_STMT_MODE_AND_NO_REPL_TABLES; SQLSTATE: HY000

  Message: Cannot execute statement: impossible to write to binary log since BINLOG_FORMAT = STATEMENT, and both replicated and non replicated tables are written to.

- Error number: 1845; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED; SQLSTATE: 0A000

  Message: %s is not supported for this operation. Try %s.

- Error number: 1846; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON; SQLSTATE: 0A000

  Message: %s is not supported. Reason: %s. Try %s.

- Error number: 1847; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COPY; SQLSTATE: HY000

  Message: COPY algorithm requires a lock

- Error number: 1848; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_PARTITION; SQLSTATE: HY000

  Message: Partition specific operations do not yet support LOCK/ALGORITHM

- Error number: 1849; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_RENAME; SQLSTATE: HY000

  Message: Columns participating in a foreign key are renamed

- Error number: 1850; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COLUMN_TYPE; SQLSTATE: HY000

  Message: Cannot change column type INPLACE

- Error number: 1851; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_CHECK; SQLSTATE: HY000

Message: Adding foreign keys needs foreign_key_checks=OFF

- Error number: 1853; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOPK; SQLSTATE: HY000

  Message: Dropping a primary key is not allowed without also adding a new primary key

- Error number: 1854; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_AUTOINC; SQLSTATE: HY000

  Message: Adding an auto-increment column requires a lock

- Error number: 1855; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_HIDDEN_FTS; SQLSTATE: HY000

  Message: Cannot replace hidden FTS_DOC_ID with a user-visible one

- Error number: 1856; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_CHANGE_FTS; SQLSTATE: HY000

  Message: Cannot drop or rename FTS_DOC_ID

- Error number: 1857; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FTS; SQLSTATE: HY000

  Message: Fulltext index creation requires a lock

- Error number: 1858; Symbol: ER_SQL_SLAVE_SKIP_COUNTER_NOT_SETTABLE_IN_GTID_MODE; SQLSTATE: HY000

  Message: sql_slave_skip_counter can not be set when the server is running with @@GLOBAL.GTID_MODE = ON. Instead, for each transaction that you want to skip, generate an empty transaction with the same GTID as the transaction

- Error number: 1859; Symbol: ER_DUP_UNKNOWN_IN_INDEX; SQLSTATE: 23000

  Message: Duplicate entry for key '%s'

- Error number: 1860; Symbol: ER_IDENT_CAUSES_TOO_LONG_PATH; SQLSTATE: HY000

  Message: Long database name and identifier for object resulted in path length exceeding %d characters. Path: '%s'.

- Error number: 1861; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOT_NULL; SQLSTATE: HY000

  Message: cannot silently convert NULL values, as required in this SQL_MODE

- Error number: 1862; Symbol: ER_MUST_CHANGE_PASSWORD_LOGIN; SQLSTATE: HY000

  Message: Your password has expired. To log in you must change it using a client that supports expired passwords.

- Error number: 1863; Symbol: ER_ROW_IN_WRONG_PARTITION; SQLSTATE: HY000

  Message: Found a row in wrong partition %s

- Error number: 1864; Symbol: ER_MTS_EVENT_BIGGER_PENDING_JOBS_SIZE_MAX; SQLSTATE: HY000

Message: Cannot schedule event %s, relay-log name %s, position %s to Worker thread because its size %lu exceeds %lu of slave_pending_jobs_size_max.

- Error number: 1866; Symbol: ER_BINLOG_LOGICAL_CORRUPTION; SQLSTATE: HY000

  Message: The binary log file '%s' is logically corrupted: %s

- Error number: 1867; Symbol: ER_WARN_PURGE_LOG_IN_USE; SQLSTATE: HY000

  Message: file %s was not purged because it was being read by %d thread(s), purged only %d out of %d files.

- Error number: 1868; Symbol: ER_WARN_PURGE_LOG_IS_ACTIVE; SQLSTATE: HY000

  Message: file %s was not purged because it is the active log file.

- Error number: 1869; Symbol: ER_AUTO_INCREMENT_CONFLICT; SQLSTATE: HY000

  Message: Auto-increment value in UPDATE conflicts with internally generated values

- Error number: 1870; Symbol: WARN_ON_BLOCKHOLE_IN_RBR; SQLSTATE: HY000

  Message: Row events are not logged for %s statements that modify BLACKHOLE tables in row format. Table(s): '%s'

- Error number: 1871; Symbol: ER_SLAVE_MI_INIT_REPOSITORY; SQLSTATE: HY000

  Message: Slave failed to initialize master info structure from the repository

- Error number: 1872; Symbol: ER_SLAVE_RLI_INIT_REPOSITORY; SQLSTATE: HY000

  Message: Slave failed to initialize relay log info structure from the repository

- Error number: 1873; Symbol: ER_ACCESS_DENIED_CHANGE_USER_ERROR; SQLSTATE: 28000

  Message: Access denied trying to change to user '%s'@'%s' (using password: %s). Disconnecting.

- Error number: 1874; Symbol: ER_INNODB_READ_ONLY; SQLSTATE: HY000

  Message: InnoDB is in read only mode.

- Error number: 1875; Symbol: ER_STOP_SLAVE_SQL_THREAD_TIMEOUT; SQLSTATE: HY000

  Message: STOP SLAVE command execution is incomplete: Slave SQL thread got the stop signal, thread is busy, SQL thread will stop once the current task is complete.

- Error number: 1876; Symbol: ER_STOP_SLAVE_IO_THREAD_TIMEOUT; SQLSTATE: HY000

  Message: STOP SLAVE command execution is incomplete: Slave IO thread got the stop signal, thread is busy, IO thread will stop once the current task is complete.

- Error number: 1877; Symbol: ER_TABLE_CORRUPT; SQLSTATE: HY000

  Message: Operation cannot be performed. The table '%s.%s' is missing, corrupt or contains bad data.

- Error number: 1878; Symbol: ER_TEMP_FILE_WRITE_FAILURE; SQLSTATE: HY000

  Message: Temporary file write failure.

- Error number: 1879; Symbol: ER_INNODB_FT_AUX_NOT_HEX_ID; SQLSTATE: HY000

  Message: Upgrade index name failed, please use create index(alter table) algorithm copy to rebuild index.

- Error number: 1880; Symbol: ER_OLD_TEMPORALS_UPGRADED; SQLSTATE: HY000

  Message: TIME/TIMESTAMP/DATETIME columns of old format have been upgraded to the new format.

- Error number: 1881; Symbol: ER_INNODB_FORCED_RECOVERY; SQLSTATE: HY000

  Message: Operation not allowed when innodb_force_recovery > 0.

- Error number: 1882; Symbol: ER_AES_INVALID_IV; SQLSTATE: HY000

  Message: The initialization vector supplied to %s is too short. Must be at least %d bytes long

- Error number: 1883; Symbol: ER_PLUGIN_CANNOT_BE_UNINSTALLED; SQLSTATE: HY000

  Message: Plugin '%s' cannot be uninstalled now. %s

- Error number: 1884; Symbol: ER_GTID_UNSAFE_BINLOG_SPLITTABLE_STATEMENT_AND_GTID_GROUP; SQLSTATE: HY000

  Message: Cannot execute statement because it needs to be written to the binary log as multiple statements, and this is not allowed when @@SESSION.GTID_NEXT == 'UUID:NUMBER'.

  ER_GTID_UNSAFE_BINLOG_SPLITTABLE_STATEMENT_AND_GTID_GROUP was removed after 8.0.4.

- Error number: 1884; Symbol: ER_GTID_UNSAFE_BINLOG_SPLITTABLE_STATEMENT_AND_ASSIGNED_GTID; SQLSTATE: HY000

  Message: Cannot execute statement because it needs to be written to the binary log as multiple statements, and this is not allowed when @@SESSION.GTID_NEXT == 'UUID:NUMBER'.

  ER_GTID_UNSAFE_BINLOG_SPLITTABLE_STATEMENT_AND_ASSIGNED_GTID was added in 8.0.11.

- Error number: 1885; Symbol: ER_SLAVE_HAS_MORE_GTIDS_THAN_MASTER; SQLSTATE: HY000

  Message: Slave has more GTIDs than the master has, using the master's SERVER_UUID. This may indicate that the end of the binary log was truncated or that the last binary log file was lost, e.g., after a power or disk failure when sync_binlog != 1. The master may or may not have rolled back transactions that were already replicated to the slave. Suggest to replicate any transactions that master has rolled back from slave to master, and/or commit empty transactions on master to account for transactions that have been committed on master but are not included in GTID_EXECUTED.

- Error number: 1886; Symbol: ER_MISSING_KEY; SQLSTATE: HY000

  Message: The table '%s.%s' does not have the necessary key(s) defined on it. Please check the table definition and create index(s) accordingly.

  ER_MISSING_KEY was added in 8.0.4.

- Error number: 1887; Symbol: WARN_NAMED_PIPE_ACCESS_EVERYONE; SQLSTATE: HY000

  Message: Setting named_pipe_full_access_group='%s' is insecure. Consider using a Windows group with fewer members.

  WARN_NAMED_PIPE_ACCESS_EVERYONE was added in 8.0.17.

- Error number: 3000; Symbol: ER_FILE_CORRUPT; SQLSTATE: HY000

  Message: File %s is corrupted

- Error number: 3001; Symbol: ER_ERROR_ON_MASTER; SQLSTATE: HY000

  Message: Query partially completed on the master (error on master: %d) and was aborted. There is a chance that your master is inconsistent at this point. If you are sure that your master is ok, run this query manually on the slave and then restart the slave with SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1; START SLAVE;. Query:'%s'

- Error number: 3003; Symbol: ER_STORAGE_ENGINE_NOT_LOADED; SQLSTATE: HY000

  Message: Storage engine for table '%s'.'%s' is not loaded.

- Error number: 3004; Symbol: ER_GET_STACKED_DA_WITHOUT_ACTIVE_HANDLER; SQLSTATE: 0Z002

  Message: GET STACKED DIAGNOSTICS when handler not active

- Error number: 3005; Symbol: ER_WARN_LEGACY_SYNTAX_CONVERTED; SQLSTATE: HY000

  Message: %s is no longer supported. The statement was converted to %s.

- Error number: 3006; Symbol: ER_BINLOG_UNSAFE_FULLTEXT_PLUGIN; SQLSTATE: HY000

  Message: Statement is unsafe because it uses a fulltext parser plugin which may not return the same value on the slave.

- Error number: 3007; Symbol: ER_CANNOT_DISCARD_TEMPORARY_TABLE; SQLSTATE: HY000

  Message: Cannot DISCARD/IMPORT tablespace associated with temporary table

- Error number: 3008; Symbol: ER_FK_DEPTH_EXCEEDED; SQLSTATE: HY000

  Message: Foreign key cascade delete/update exceeds max depth of %d.

- Error number: 3009; Symbol: ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE_V2; SQLSTATE: HY000

  Message: The column count of %s.%s is wrong. Expected %d, found %d. Created with MySQL %d, now running %d. Please perform the MySQL upgrade procedure.

- Error number: 3010; Symbol: ER_WARN_TRIGGER_DOESNT_HAVE_CREATED; SQLSTATE: HY000

  Message: Trigger %s.%s.%s does not have CREATED attribute.

- Error number: 3011; Symbol: ER_REFERENCED_TRG_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: Referenced trigger '%s' for the given action time and event type does not exist.

- Error number: 3012; Symbol: ER_EXPLAIN_NOT_SUPPORTED; SQLSTATE: HY000

  Message: EXPLAIN FOR CONNECTION command is supported only for SELECT/UPDATE/INSERT/DELETE/REPLACE

- Error number: 3013; Symbol: ER_INVALID_FIELD_SIZE; SQLSTATE: HY000

  Message: Invalid size for column '%s'.

- Error number: 3014; Symbol: ER_MISSING_HA_CREATE_OPTION; SQLSTATE: HY000

  Message: Table storage engine '%s' found required create option missing

- Error number: 3015; Symbol: ER_ENGINE_OUT_OF_MEMORY; SQLSTATE: HY000

  Message: Out of memory in storage engine '%s'.

- Error number: 3016; Symbol: ER_PASSWORD_EXPIRE_ANONYMOUS_USER; SQLSTATE: HY000

  Message: The password for anonymous user cannot be expired.

- Error number: 3017; Symbol: ER_SLAVE_SQL_THREAD_MUST_STOP; SQLSTATE: HY000

  Message: This operation cannot be performed with a running slave sql thread; run STOP SLAVE SQL_THREAD first

- Error number: 3018; Symbol: ER_NO_FT_MATERIALIZED_SUBQUERY; SQLSTATE: HY000

  Message: Cannot create FULLTEXT index on materialized subquery

- Error number: 3019; Symbol: ER_INNODB_UNDO_LOG_FULL; SQLSTATE: HY000

  Message: Undo Log error: %s

- Error number: 3020; Symbol: ER_INVALID_ARGUMENT_FOR_LOGARITHM; SQLSTATE: 2201E

  Message: Invalid argument for logarithm

- Error number: 3021; Symbol: ER_SLAVE_CHANNEL_IO_THREAD_MUST_STOP; SQLSTATE: HY000

  Message: This operation cannot be performed with a running slave io thread; run STOP SLAVE IO_THREAD FOR CHANNEL '%s' first.

- Error number: 3022; Symbol: ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO; SQLSTATE: HY000

  Message: This operation may not be safe when the slave has temporary tables. The tables will be kept open until the server restarts or until the tables are deleted by any replicated DROP statement. Suggest to wait until slave_open_temp_tables = 0.

- Error number: 3023; Symbol: ER_WARN_ONLY_MASTER_LOG_FILE_NO_POS; SQLSTATE: HY000

  Message: CHANGE MASTER TO with a MASTER_LOG_FILE clause but no MASTER_LOG_POS clause may not be safe. The old position value may not be valid for the new binary log file.

- Error number: 3024; Symbol: ER_QUERY_TIMEOUT; SQLSTATE: HY000

  Message: Query execution was interrupted, maximum statement execution time exceeded

- Error number: 3025; Symbol: ER_NON_RO_SELECT_DISABLE_TIMER; SQLSTATE: HY000

  Message: Select is not a read only statement, disabling timer

- Error number: 3026; Symbol: ER_DUP_LIST_ENTRY; SQLSTATE: HY000

  Message: Duplicate entry '%s'.

- Error number: 3028; Symbol: ER_AGGREGATE_ORDER_FOR_UNION; SQLSTATE: HY000

  Message: Expression #%u of ORDER BY contains aggregate function and applies to a UNION

- Error number: 3029; Symbol: ER_AGGREGATE_ORDER_NON_AGG_QUERY; SQLSTATE: HY000

  Message: Expression #%u of ORDER BY contains aggregate function and applies to the result of a non-aggregated query

- Error number: 3030; Symbol: ER_SLAVE_WORKER_STOPPED_PREVIOUS_THD_ERROR; SQLSTATE: HY000

  Message: Slave worker has stopped after at least one previous worker encountered an error when slave-preserve-commit-order was enabled. To preserve commit order, the last transaction executed by this thread has not been committed. When restarting the slave after fixing any failed threads, you should fix this worker as well.

- Error number: 3031; Symbol: ER_DONT_SUPPORT_SLAVE_PRESERVE_COMMIT_ORDER; SQLSTATE: HY000

  Message: slave_preserve_commit_order is not supported %s.

- Error number: 3032; Symbol: ER_SERVER_OFFLINE_MODE; SQLSTATE: HY000

  Message: The server is currently in offline mode

- Error number: 3033; Symbol: ER_GIS_DIFFERENT_SRIDS; SQLSTATE: HY000

  Message: Binary geometry function %s given two geometries of different srids: %u and %u, which should have been identical.

  Geometry values passed as arguments to spatial functions must have the same SRID value.

- Error number: 3034; Symbol: ER_GIS_UNSUPPORTED_ARGUMENT; SQLSTATE: HY000

  Message: Calling geometry function %s with unsupported types of arguments.

  A spatial function was called with a combination of argument types that the function does not support.

- Error number: 3035; Symbol: ER_GIS_UNKNOWN_ERROR; SQLSTATE: HY000

  Message: Unknown GIS error occurred in function %s.

- Error number: 3036; Symbol: ER_GIS_UNKNOWN_EXCEPTION; SQLSTATE: HY000

  Message: Unknown exception caught in GIS function %s.

- Error number: 3037; Symbol: ER_GIS_INVALID_DATA; SQLSTATE: 22023

  Message: Invalid GIS data provided to function %s.

  A spatial function was called with an argument not recognized as a valid geometry value.

- Error number: 3038; Symbol: ER_BOOST_GEOMETRY_EMPTY_INPUT_EXCEPTION; SQLSTATE: HY000

  Message: The geometry has no data in function %s.

- Error number: 3039; Symbol: ER_BOOST_GEOMETRY_CENTROID_EXCEPTION; SQLSTATE: HY000

  Message: Unable to calculate centroid because geometry is empty in function %s.

- Error number: 3040; Symbol: ER_BOOST_GEOMETRY_OVERLAY_INVALID_INPUT_EXCEPTION; SQLSTATE: HY000

Message: Geometry overlay calculation error: geometry data is invalid in function %s.

- Error number: 3041; Symbol: ER_BOOST_GEOMETRY_TURN_INFO_EXCEPTION; SQLSTATE: HY000

  Message: Geometry turn info calculation error: geometry data is invalid in function %s.

- Error number: 3042; Symbol: ER_BOOST_GEOMETRY_SELF_INTERSECTION_POINT_EXCEPTION; SQLSTATE: HY000

  Message: Analysis procedures of intersection points interrupted unexpectedly in function %s.

- Error number: 3043; Symbol: ER_BOOST_GEOMETRY_UNKNOWN_EXCEPTION; SQLSTATE: HY000

  Message: Unknown exception thrown in function %s.

- Error number: 3044; Symbol: ER_STD_BAD_ALLOC_ERROR; SQLSTATE: HY000

  Message: Memory allocation error: %s in function %s.

- Error number: 3045; Symbol: ER_STD_DOMAIN_ERROR; SQLSTATE: HY000

  Message: Domain error: %s in function %s.

- Error number: 3046; Symbol: ER_STD_LENGTH_ERROR; SQLSTATE: HY000

  Message: Length error: %s in function %s.

- Error number: 3047; Symbol: ER_STD_INVALID_ARGUMENT; SQLSTATE: HY000

  Message: Invalid argument error: %s in function %s.

- Error number: 3048; Symbol: ER_STD_OUT_OF_RANGE_ERROR; SQLSTATE: HY000

  Message: Out of range error: %s in function %s.

- Error number: 3049; Symbol: ER_STD_OVERFLOW_ERROR; SQLSTATE: HY000

  Message: Overflow error: %s in function %s.

- Error number: 3050; Symbol: ER_STD_RANGE_ERROR; SQLSTATE: HY000

  Message: Range error: %s in function %s.

- Error number: 3051; Symbol: ER_STD_UNDERFLOW_ERROR; SQLSTATE: HY000

  Message: Underflow error: %s in function %s.

- Error number: 3052; Symbol: ER_STD_LOGIC_ERROR; SQLSTATE: HY000

  Message: Logic error: %s in function %s.

- Error number: 3053; Symbol: ER_STD_RUNTIME_ERROR; SQLSTATE: HY000

  Message: Runtime error: %s in function %s.

- Error number: 3054; Symbol: ER_STD_UNKNOWN_EXCEPTION; SQLSTATE: HY000

  Message: Unknown exception: %s in function %s.

- Error number: 3055; Symbol: ER_GIS_DATA_WRONG_ENDIANESS; SQLSTATE: HY000

Message: Geometry byte string must be little endian.

- Error number: 3056; Symbol: ER_CHANGE_MASTER_PASSWORD_LENGTH; SQLSTATE: HY000

  Message: The password provided for the replication user exceeds the maximum length of 32 characters

- Error number: 3057; Symbol: ER_USER_LOCK_WRONG_NAME; SQLSTATE: 42000

  Message: Incorrect user-level lock name '%s'.

- Error number: 3058; Symbol: ER_USER_LOCK_DEADLOCK; SQLSTATE: HY000

  Message: Deadlock found when trying to get user-level lock; try rolling back transaction/releasing locks and restarting lock acquisition.

  This error is returned when the metdata locking subsystem detects a deadlock for an attempt to acquire a named lock with GET_LOCK.

- Error number: 3059; Symbol: ER_REPLACE_INACCESSIBLE_ROWS; SQLSTATE: HY000

  Message: REPLACE cannot be executed as it requires deleting rows that are not in the view

- Error number: 3060; Symbol: ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_GIS; SQLSTATE: HY000

  Message: Do not support online operation on table with GIS index

- Error number: 3061; Symbol: ER_ILLEGAL_USER_VAR; SQLSTATE: 42000

  Message: User variable name '%s' is illegal

- Error number: 3062; Symbol: ER_GTID_MODE_OFF; SQLSTATE: HY000

  Message: Cannot %s when GTID_MODE = OFF.

- Error number: 3064; Symbol: ER_INCORRECT_TYPE; SQLSTATE: HY000

  Message: Incorrect type for argument %s in function %s.

- Error number: 3065; Symbol: ER_FIELD_IN_ORDER_NOT_SELECT; SQLSTATE: HY000

  Message: Expression #%u of ORDER BY clause is not in SELECT list, references column '%s' which is not in SELECT list; this is incompatible with %s

- Error number: 3066; Symbol: ER_AGGREGATE_IN_ORDER_NOT_SELECT; SQLSTATE: HY000

  Message: Expression #%u of ORDER BY clause is not in SELECT list, contains aggregate function; this is incompatible with %s

- Error number: 3067; Symbol: ER_INVALID_RPL_WILD_TABLE_FILTER_PATTERN; SQLSTATE: HY000

  Message: Supplied filter list contains a value which is not in the required format 'db_pattern.table_pattern'

- Error number: 3068; Symbol: ER_NET_OK_PACKET_TOO_LARGE; SQLSTATE: 08S01

  Message: OK packet too large

- Error number: 3069; Symbol: ER_INVALID_JSON_DATA; SQLSTATE: HY000

  Message: Invalid JSON data provided to function %s: %s

- Error number: 3070; Symbol: ER_INVALID_GEOJSON_MISSING_MEMBER; SQLSTATE: HY000

  Message: Invalid GeoJSON data provided to function %s: Missing required member '%s'

- Error number: 3071; Symbol: ER_INVALID_GEOJSON_WRONG_TYPE; SQLSTATE: HY000

  Message: Invalid GeoJSON data provided to function %s: Member '%s' must be of type '%s'

- Error number: 3072; Symbol: ER_INVALID_GEOJSON_UNSPECIFIED; SQLSTATE: HY000

  Message: Invalid GeoJSON data provided to function %s

- Error number: 3073; Symbol: ER_DIMENSION_UNSUPPORTED; SQLSTATE: HY000

  Message: Unsupported number of coordinate dimensions in function %s: Found %u, expected %u

- Error number: 3074; Symbol: ER_SLAVE_CHANNEL_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: Slave channel '%s' does not exist.

- Error number: 3076; Symbol: ER_SLAVE_CHANNEL_NAME_INVALID_OR_TOO_LONG; SQLSTATE: HY000

  Message: Couldn't create channel: Channel name is either invalid or too long.

- Error number: 3077; Symbol: ER_SLAVE_NEW_CHANNEL_WRONG_REPOSITORY; SQLSTATE: HY000

  Message: To have multiple channels, repository cannot be of type FILE; Please check the repository configuration and convert them to TABLE.

- Error number: 3079; Symbol: ER_SLAVE_MULTIPLE_CHANNELS_CMD; SQLSTATE: HY000

  Message: Multiple channels exist on the slave. Please provide channel name as an argument.

- Error number: 3080; Symbol: ER_SLAVE_MAX_CHANNELS_EXCEEDED; SQLSTATE: HY000

  Message: Maximum number of replication channels allowed exceeded.

- Error number: 3081; Symbol: ER_SLAVE_CHANNEL_MUST_STOP; SQLSTATE: HY000

  Message: This operation cannot be performed with running replication threads; run STOP SLAVE FOR CHANNEL '%s' first

- Error number: 3082; Symbol: ER_SLAVE_CHANNEL_NOT_RUNNING; SQLSTATE: HY000

  Message: This operation requires running replication threads; configure slave and run START SLAVE FOR CHANNEL '%s'

- Error number: 3083; Symbol: ER_SLAVE_CHANNEL_WAS_RUNNING; SQLSTATE: HY000

  Message: Replication thread(s) for channel '%s' are already runnning.

- Error number: 3084; Symbol: ER_SLAVE_CHANNEL_WAS_NOT_RUNNING; SQLSTATE: HY000

  Message: Replication thread(s) for channel '%s' are already stopped.

- Error number: 3085; Symbol: ER_SLAVE_CHANNEL_SQL_THREAD_MUST_STOP; SQLSTATE: HY000

  Message: This operation cannot be performed with a running slave sql thread; run STOP SLAVE SQL_THREAD FOR CHANNEL '%s' first.

- Error number: 3086; Symbol: ER_SLAVE_CHANNEL_SQL_SKIP_COUNTER; SQLSTATE: HY000

  Message: When sql_slave_skip_counter > 0, it is not allowed to start more than one SQL thread by using 'START SLAVE [SQL_THREAD]'. Value of sql_slave_skip_counter can only be used by one SQL thread at a time. Please use 'START SLAVE [SQL_THREAD] FOR CHANNEL' to start the SQL thread which will use the value of sql_slave_skip_counter.

- Error number: 3087; Symbol: ER_WRONG_FIELD_WITH_GROUP_V2; SQLSTATE: HY000

  Message: Expression #%u of %s is not in GROUP BY clause and contains nonaggregated column '%s' which is not functionally dependent on columns in GROUP BY clause; this is incompatible with sql_mode=only_full_group_by

- Error number: 3088; Symbol: ER_MIX_OF_GROUP_FUNC_AND_FIELDS_V2; SQLSTATE: HY000

  Message: In aggregated query without GROUP BY, expression #%u of %s contains nonaggregated column '%s'; this is incompatible with sql_mode=only_full_group_by

- Error number: 3089; Symbol: ER_WARN_DEPRECATED_SYSVAR_UPDATE; SQLSTATE: HY000

  Message: Updating '%s' is deprecated. It will be made read-only in a future release.

- Error number: 3090; Symbol: ER_WARN_DEPRECATED_SQLMODE; SQLSTATE: HY000

  Message: Changing sql mode '%s' is deprecated. It will be removed in a future release.

- Error number: 3091; Symbol: ER_CANNOT_LOG_PARTIAL_DROP_DATABASE_WITH_GTID; SQLSTATE: HY000

  Message: DROP DATABASE failed; some tables may have been dropped but the database directory remains. The GTID has not been added to GTID_EXECUTED and the statement was not written to the binary log. Fix this as follows: (1) remove all files from the database directory %s; (2) SET GTID_NEXT='%s'; (3) DROP DATABASE `%s`.

- Error number: 3092; Symbol: ER_GROUP_REPLICATION_CONFIGURATION; SQLSTATE: HY000

  Message: The server is not configured properly to be an active member of the group. Please see more details on error log.

- Error number: 3093; Symbol: ER_GROUP_REPLICATION_RUNNING; SQLSTATE: HY000

  Message: The START GROUP_REPLICATION command failed since the group is already running.

- Error number: 3094; Symbol: ER_GROUP_REPLICATION_APPLIER_INIT_ERROR; SQLSTATE: HY000

  Message: The START GROUP_REPLICATION command failed as the applier module failed to start.

- Error number: 3095; Symbol: ER_GROUP_REPLICATION_STOP_APPLIER_THREAD_TIMEOUT; SQLSTATE: HY000

  Message: The STOP GROUP_REPLICATION command execution is incomplete: The applier thread got the stop signal while it was busy. The applier thread will stop once the current task is complete.

- Error number: 3096; Symbol: ER_GROUP_REPLICATION_COMMUNICATION_LAYER_SESSION_ERROR; SQLSTATE: HY000

  Message: The START GROUP_REPLICATION command failed as there was an error when initializing the group communication layer.

- Error number: 3097; Symbol: ER_GROUP_REPLICATION_COMMUNICATION_LAYER_JOIN_ERROR; SQLSTATE: HY000

  Message: The START GROUP_REPLICATION command failed as there was an error when joining the communication group.

- Error number: 3098; Symbol: ER_BEFORE_DML_VALIDATION_ERROR; SQLSTATE: HY000

  Message: The table does not comply with the requirements by an external plugin.

- Error number: 3099; Symbol: ER_PREVENTS_VARIABLE_WITHOUT_RBR; SQLSTATE: HY000

  Message: Cannot change the value of variable %s without binary log format as ROW.

  transaction_write_set_extraction option value is set and binlog_format is not ROW.

- Error number: 3100; Symbol: ER_RUN_HOOK_ERROR; SQLSTATE: HY000

  Message: Error on observer while running replication hook '%s'.

- Error number: 3101; Symbol: ER_TRANSACTION_ROLLBACK_DURING_COMMIT; SQLSTATE: 40000

  Message: Plugin instructed the server to rollback the current transaction.

  When using Group Replication, this means that a transaction failed the group certification process, due to one or more members detecting a potential conflict, and was thus rolled back. See Chapter 18, *Group Replication*.

- Error number: 3102; Symbol: ER_GENERATED_COLUMN_FUNCTION_IS_NOT_ALLOWED; SQLSTATE: HY000

  Message: Expression of generated column '%s' contains a disallowed function.

- Error number: 3103; Symbol: ER_UNSUPPORTED_ALTER_INPLACE_ON_VIRTUAL_COLUMN; SQLSTATE: HY000

  Message: INPLACE ADD or DROP of virtual columns cannot be combined with other ALTER TABLE actions

- Error number: 3104; Symbol: ER_WRONG_FK_OPTION_FOR_GENERATED_COLUMN; SQLSTATE: HY000

  Message: Cannot define foreign key with %s clause on a generated column.

- Error number: 3105; Symbol: ER_NON_DEFAULT_VALUE_FOR_GENERATED_COLUMN; SQLSTATE: HY000

  Message: The value specified for generated column '%s' in table '%s' is not allowed.

- Error number: 3106; Symbol: ER_UNSUPPORTED_ACTION_ON_GENERATED_COLUMN; SQLSTATE: HY000

  Message: '%s' is not supported for generated columns.

- Error number: 3107; Symbol: ER_GENERATED_COLUMN_NON_PRIOR; SQLSTATE: HY000

  Message: Generated column can refer only to generated columns defined prior to it.

  To address this issue, change the table definition to define each generated column later than any generated columns to which it refers.

- Error number: 3108; Symbol: ER_DEPENDENT_BY_GENERATED_COLUMN; SQLSTATE: HY000

  Message: Column '%s' has a generated column dependency.

  You cannot drop or rename a generated column if another column refers to it. You must either drop those columns as well, or redefine them not to refer to the generated column.

- Error number: 3109; Symbol: ER_GENERATED_COLUMN_REF_AUTO_INC; SQLSTATE: HY000

  Message: Generated column '%s' cannot refer to auto-increment column.

- Error number: 3110; Symbol: ER_FEATURE_NOT_AVAILABLE; SQLSTATE: HY000

  Message: The '%s' feature is not available; you need to remove '%s' or use MySQL built with '%s'

- Error number: 3111; Symbol: ER_CANT_SET_GTID_MODE; SQLSTATE: HY000

  Message: SET @@GLOBAL.GTID_MODE = %s is not allowed because %s.

- Error number: 3112; Symbol: ER_CANT_USE_AUTO_POSITION_WITH_GTID_MODE_OFF; SQLSTATE: HY000

  Message: The replication receiver thread%s cannot start in AUTO_POSITION mode: this server uses @@GLOBAL.GTID_MODE = OFF.

- Error number: 3116; Symbol: ER_CANT_ENFORCE_GTID_CONSISTENCY_WITH_ONGOING_GTID_VIOLATING_TX; SQLSTATE: HY000

  Message: Cannot set ENFORCE_GTID_CONSISTENCY = ON because there are ongoing transactions that violate GTID consistency.

  ER_CANT_SET_ENFORCE_GTID_CONSISTENCY_ON_WITH_ONGOING_GTID_VIOLATING_TRANSACTIONS was renamed to ER_CANT_ENFORCE_GTID_CONSISTENCY_WITH_ONGOING_GTID_VIOLATING_TX.

- Error number: 3117; Symbol: ER_ENFORCE_GTID_CONSISTENCY_WARN_WITH_ONGOING_GTID_VIOLATING_TX; SQLSTATE: HY000

  Message: There are ongoing transactions that violate GTID consistency.

  ER_SET_ENFORCE_GTID_CONSISTENCY_WARN_WITH_ONGOING_GTID_VIOLATING_TRANSACTIONS was renamed to ER_ENFORCE_GTID_CONSISTENCY_WARN_WITH_ONGOING_GTID_VIOLATING_TX.

- Error number: 3118; Symbol: ER_ACCOUNT_HAS_BEEN_LOCKED; SQLSTATE: HY000

  Message: Access denied for user '%s'@'%s'. Account is locked.

  The account was locked with CREATE USER ... ACCOUNT LOCK or ALTER USER ... ACCOUNT LOCK. An administrator can unlock it with ALTER USER ... ACCOUNT UNLOCK.

- Error number: 3119; Symbol: ER_WRONG_TABLESPACE_NAME; SQLSTATE: 42000

Message: Incorrect tablespace name `%s`

- Error number: 3120; Symbol: ER_TABLESPACE_IS_NOT_EMPTY; SQLSTATE: HY000

  Message: Tablespace `%s` is not empty.

- Error number: 3121; Symbol: ER_WRONG_FILE_NAME; SQLSTATE: HY000

  Message: Incorrect File Name '%s'.

- Error number: 3122; Symbol: ER_BOOST_GEOMETRY_INCONSISTENT_TURNS_EXCEPTION; SQLSTATE: HY000

  Message: Inconsistent intersection points.

- Error number: 3123; Symbol: ER_WARN_OPTIMIZER_HINT_SYNTAX_ERROR; SQLSTATE: HY000

  Message: Optimizer hint syntax error

- Error number: 3124; Symbol: ER_WARN_BAD_MAX_EXECUTION_TIME; SQLSTATE: HY000

  Message: Unsupported MAX_EXECUTION_TIME

- Error number: 3125; Symbol: ER_WARN_UNSUPPORTED_MAX_EXECUTION_TIME; SQLSTATE: HY000

  Message: MAX_EXECUTION_TIME hint is supported by top-level standalone SELECT statements only

  The MAX_EXECUTION_TIME optimizer hint is supported only for SELECT statements.

- Error number: 3126; Symbol: ER_WARN_CONFLICTING_HINT; SQLSTATE: HY000

  Message: Hint %s is ignored as conflicting/duplicated

- Error number: 3127; Symbol: ER_WARN_UNKNOWN_QB_NAME; SQLSTATE: HY000

  Message: Query block name %s is not found for %s hint

- Error number: 3128; Symbol: ER_UNRESOLVED_HINT_NAME; SQLSTATE: HY000

  Message: Unresolved name %s for %s hint

- Error number: 3129; Symbol: ER_WARN_ON_MODIFYING_GTID_EXECUTED_TABLE; SQLSTATE: HY000

  Message: Please do not modify the %s table. This is a mysql internal system table to store GTIDs for committed transactions. Modifying it can lead to an inconsistent GTID state.

- Error number: 3130; Symbol: ER_PLUGGABLE_PROTOCOL_COMMAND_NOT_SUPPORTED; SQLSTATE: HY000

  Message: Command not supported by pluggable protocols

- Error number: 3131; Symbol: ER_LOCKING_SERVICE_WRONG_NAME; SQLSTATE: 42000

  Message: Incorrect locking service lock name '%s'.

  A locking service name was specified as NULL, the empty string, or a string longer than 64 characters. Namespace and lock names must be non-NULL, nonempty, and no more than 64 characters long.

- Error number: 3132; Symbol: ER_LOCKING_SERVICE_DEADLOCK; SQLSTATE: HY000

  Message: Deadlock found when trying to get locking service lock; try releasing locks and restarting lock acquisition.

- Error number: 3133; Symbol: ER_LOCKING_SERVICE_TIMEOUT; SQLSTATE: HY000

  Message: Service lock wait timeout exceeded.

- Error number: 3134; Symbol: ER_GIS_MAX_POINTS_IN_GEOMETRY_OVERFLOWED; SQLSTATE: HY000

  Message: Parameter %s exceeds the maximum number of points in a geometry (%lu) in function %s.

- Error number: 3135; Symbol: ER_SQL_MODE_MERGED; SQLSTATE: HY000

  Message: 'NO_ZERO_DATE', 'NO_ZERO_IN_DATE' and 'ERROR_FOR_DIVISION_BY_ZERO' sql modes should be used with strict mode. They will be merged with strict mode in a future release.

- Error number: 3136; Symbol: ER_VTOKEN_PLUGIN_TOKEN_MISMATCH; SQLSTATE: HY000

  Message: Version token mismatch for %.*s. Correct value %.*s

  The client has set its version_tokens_session system variable to the list of tokens it requires the server to match, but the server token list has at least one matching token name that has a value different from what the client requires. See Section 5.6.6, “Version Tokens”.

- Error number: 3137; Symbol: ER_VTOKEN_PLUGIN_TOKEN_NOT_FOUND; SQLSTATE: HY000

  Message: Version token %.*s not found.

  The client has set its version_tokens_session system variable to the list of tokens it requires the server to match, but the server token list is missing at least one of those tokens. See Section 5.6.6, “Version Tokens”.

- Error number: 3138; Symbol: ER_CANT_SET_VARIABLE_WHEN_OWNING_GTID; SQLSTATE: HY000

  Message: Variable %s cannot be changed by a client that owns a GTID. The client owns %s. Ownership is released on COMMIT or ROLLBACK.

- Error number: 3139; Symbol: ER_SLAVE_CHANNEL_OPERATION_NOT_ALLOWED; SQLSTATE: HY000

  Message: %s cannot be performed on channel '%s'.

- Error number: 3140; Symbol: ER_INVALID_JSON_TEXT; SQLSTATE: 22032

  Message: Invalid JSON text: "%s" at position %u in value for column '%s'.

- Error number: 3141; Symbol: ER_INVALID_JSON_TEXT_IN_PARAM; SQLSTATE: 22032

  Message: Invalid JSON text in argument %u to function %s: "%s" at position %u.%s

- Error number: 3142; Symbol: ER_INVALID_JSON_BINARY_DATA; SQLSTATE: HY000

  Message: The JSON binary value contains invalid data.

- Error number: 3143; Symbol: ER_INVALID_JSON_PATH; SQLSTATE: 42000

  Message: Invalid JSON path expression. The error is around character position %u.%s

- Error number: 3144; Symbol: ER_INVALID_JSON_CHARSET; SQLSTATE: 22032

  Message: Cannot create a JSON value from a string with CHARACTER SET '%s'.

- Error number: 3145; Symbol: ER_INVALID_JSON_CHARSET_IN_FUNCTION; SQLSTATE: 22032

  Message: Invalid JSON character data provided to function %s: '%s'; utf8 is required.

- Error number: 3146; Symbol: ER_INVALID_TYPE_FOR_JSON; SQLSTATE: 22032

  Message: Invalid data type for JSON data in argument %u to function %s; a JSON string or JSON type is required.

- Error number: 3147; Symbol: ER_INVALID_CAST_TO_JSON; SQLSTATE: 22032

  Message: Cannot CAST value to JSON.

- Error number: 3148; Symbol: ER_INVALID_JSON_PATH_CHARSET; SQLSTATE: 42000

  Message: A path expression must be encoded in the utf8 character set. The path expression '%s' is encoded in character set '%s'.

- Error number: 3149; Symbol: ER_INVALID_JSON_PATH_WILDCARD; SQLSTATE: 42000

  Message: In this situation, path expressions may not contain the * and ** tokens or an array range.

- Error number: 3150; Symbol: ER_JSON_VALUE_TOO_BIG; SQLSTATE: 22032

  Message: The JSON value is too big to be stored in a JSON column.

- Error number: 3151; Symbol: ER_JSON_KEY_TOO_BIG; SQLSTATE: 22032

  Message: The JSON object contains a key name that is too long.

- Error number: 3152; Symbol: ER_JSON_USED_AS_KEY; SQLSTATE: 42000

  Message: JSON column '%s' supports indexing only via generated columns on a specified JSON path.

- Error number: 3153; Symbol: ER_JSON_VACUOUS_PATH; SQLSTATE: 42000

  Message: The path expression '$' is not allowed in this context.

- Error number: 3154; Symbol: ER_JSON_BAD_ONE_OR_ALL_ARG; SQLSTATE: 42000

  Message: The oneOrAll argument to %s may take these values: 'one' or 'all'.

- Error number: 3155; Symbol: ER_NUMERIC_JSON_VALUE_OUT_OF_RANGE; SQLSTATE: 22003

  Message: Out of range JSON value for CAST to %s%s from column %s at row %ld

- Error number: 3156; Symbol: ER_INVALID_JSON_VALUE_FOR_CAST; SQLSTATE: 22018

  Message: Invalid JSON value for CAST to %s%s from column %s at row %ld

- Error number: 3157; Symbol: ER_JSON_DOCUMENT_TOO_DEEP; SQLSTATE: 22032

  Message: The JSON document exceeds the maximum depth.

- Error number: 3158; Symbol: ER_JSON_DOCUMENT_NULL_KEY; SQLSTATE: 22032

  Message: JSON documents may not contain NULL member names.

- Error number: 3159; Symbol: ER_SECURE_TRANSPORT_REQUIRED; SQLSTATE: HY000

  Message: Connections using insecure transport are prohibited while --require_secure_transport=ON.

  With the require_secure_transport system variable, clients can connect only using secure transports. Qualifying connections are those using SSL, a Unix socket file, or shared memory.

- Error number: 3160; Symbol: ER_NO_SECURE_TRANSPORTS_CONFIGURED; SQLSTATE: HY000

  Message: No secure transports (SSL or Shared Memory) are configured, unable to set --require_secure_transport=ON.

  The require_secure_transport system variable cannot be enabled if the server does not support at least one secure transport. Configure the server with the required SSL keys/certificates to enable SSL connections, or enable the shared_memory system variable to enable shared-memory connections.

- Error number: 3161; Symbol: ER_DISABLED_STORAGE_ENGINE; SQLSTATE: HY000

  Message: Storage engine %s is disabled (Table creation is disallowed).

  An attempt was made to create a table or tablespace using a storage engine listed in the value of the disabled_storage_engines system variable, or to change an existing table or tablespace to such an engine. Choose a different storage engine.

- Error number: 3162; Symbol: ER_USER_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: Authorization ID %s does not exist.

- Error number: 3163; Symbol: ER_USER_ALREADY_EXISTS; SQLSTATE: HY000

  Message: Authorization ID %s already exists.

- Error number: 3164; Symbol: ER_AUDIT_API_ABORT; SQLSTATE: HY000

  Message: Aborted by Audit API ('%s';%d).

  This error indicates that an audit plugin terminated execution of an event. The message typically indicates the event subclass name and a numeric status value.

- Error number: 3165; Symbol: ER_INVALID_JSON_PATH_ARRAY_CELL; SQLSTATE: 42000

  Message: A path expression is not a path to a cell in an array.

- Error number: 3166; Symbol: ER_BUFPOOL_RESIZE_INPROGRESS; SQLSTATE: HY000

  Message: Another buffer pool resize is already in progress.

- Error number: 3167; Symbol: ER_FEATURE_DISABLED_SEE_DOC; SQLSTATE: HY000

  Message: The '%s' feature is disabled; see the documentation for '%s'

- Error number: 3168; Symbol: ER_SERVER_ISNT_AVAILABLE; SQLSTATE: HY000

  Message: Server isn't available

- Error number: 3169; Symbol: ER_SESSION_WAS_KILLED; SQLSTATE: HY000

  Message: Session was killed

- Error number: 3170; Symbol: ER_CAPACITY_EXCEEDED; SQLSTATE: HY000

Message: Memory capacity of %llu bytes for '%s' exceeded. %s

- Error number: 3171; Symbol: ER_CAPACITY_EXCEEDED_IN_RANGE_OPTIMIZER; SQLSTATE: HY000

  Message: Range optimization was not done for this query.

- Error number: 3173; Symbol: ER_CANT_WAIT_FOR_EXECUTED_GTID_SET_WHILE_OWNING_A_GTID; SQLSTATE: HY000

  Message: The client holds ownership of the GTID %s. Therefore, WAIT_FOR_EXECUTED_GTID_SET cannot wait for this GTID.

- Error number: 3174; Symbol: ER_CANNOT_ADD_FOREIGN_BASE_COL_VIRTUAL; SQLSTATE: HY000

  Message: Cannot add foreign key on the base column of indexed virtual column.

- Error number: 3175; Symbol: ER_CANNOT_CREATE_VIRTUAL_INDEX_CONSTRAINT; SQLSTATE: HY000

  Message: Cannot create index on virtual column whose base column has foreign constraint.

- Error number: 3176; Symbol: ER_ERROR_ON_MODIFYING_GTID_EXECUTED_TABLE; SQLSTATE: HY000

  Message: Please do not modify the %s table with an XA transaction. This is an internal system table used to store GTIDs for committed transactions. Although modifying it can lead to an inconsistent GTID state, if neccessary you can modify it with a non-XA transaction.

- Error number: 3177; Symbol: ER_LOCK_REFUSED_BY_ENGINE; SQLSTATE: HY000

  Message: Lock acquisition refused by storage engine.

- Error number: 3178; Symbol: ER_UNSUPPORTED_ALTER_ONLINE_ON_VIRTUAL_COLUMN; SQLSTATE: HY000

  Message: ADD COLUMN col...VIRTUAL, ADD INDEX(col)

- Error number: 3179; Symbol: ER_MASTER_KEY_ROTATION_NOT_SUPPORTED_BY_SE; SQLSTATE: HY000

  Message: Master key rotation is not supported by storage engine.

- Error number: 3181; Symbol: ER_MASTER_KEY_ROTATION_BINLOG_FAILED; SQLSTATE: HY000

  Message: Write to binlog failed. However, master key rotation has been completed successfully.

- Error number: 3182; Symbol: ER_MASTER_KEY_ROTATION_SE_UNAVAILABLE; SQLSTATE: HY000

  Message: Storage engine is not available.

- Error number: 3183; Symbol: ER_TABLESPACE_CANNOT_ENCRYPT; SQLSTATE: HY000

  Message: This tablespace can't be encrypted.

- Error number: 3184; Symbol: ER_INVALID_ENCRYPTION_OPTION; SQLSTATE: HY000

  Message: Invalid encryption option.

- Error number: 3185; Symbol: ER_CANNOT_FIND_KEY_IN_KEYRING; SQLSTATE: HY000

  Message: Can't find master key from keyring, please check in the server log if a keyring plugin is loaded and initialized successfully.

- Error number: 3186; Symbol: ER_CAPACITY_EXCEEDED_IN_PARSER; SQLSTATE: HY000

  Message: Parser bailed out for this query.

- Error number: 3187; Symbol: ER_UNSUPPORTED_ALTER_ENCRYPTION_INPLACE; SQLSTATE: HY000

  Message: Cannot alter encryption attribute by inplace algorithm.

- Error number: 3188; Symbol: ER_KEYRING_UDF_KEYRING_SERVICE_ERROR; SQLSTATE: HY000

  Message: Function '%s' failed because underlying keyring service returned an error. Please check if a keyring plugin is installed and that provided arguments are valid for the keyring you are using.

- Error number: 3189; Symbol: ER_USER_COLUMN_OLD_LENGTH; SQLSTATE: HY000

  Message: It seems that your db schema is old. The %s column is 77 characters long and should be 93 characters long. Please perform the MySQL upgrade procedure.

- Error number: 3190; Symbol: ER_CANT_RESET_MASTER; SQLSTATE: HY000

  Message: RESET MASTER is not allowed because %s.

- Error number: 3191; Symbol: ER_GROUP_REPLICATION_MAX_GROUP_SIZE; SQLSTATE: HY000

  Message: The START GROUP_REPLICATION command failed since the group already has 9 members.

- Error number: 3192; Symbol: ER_CANNOT_ADD_FOREIGN_BASE_COL_STORED; SQLSTATE: HY000

  Message: Cannot add foreign key on the base column of stored column.

- Error number: 3193; Symbol: ER_TABLE_REFERENCED; SQLSTATE: HY000

  Message: Cannot complete the operation because table is referenced by another connection.

- Error number: 3197; Symbol: ER_XA_RETRY; SQLSTATE: HY000

  Message: The resource manager is not able to commit the transaction branch at this time. Please retry later.

  ER_XA_RETRY was added in 8.0.2.

- Error number: 3198; Symbol: ER_KEYRING_AWS_UDF_AWS_KMS_ERROR; SQLSTATE: HY000

  Message: Function %s failed due to: %s.

  ER_KEYRING_AWS_UDF_AWS_KMS_ERROR was added in 8.0.2.

- Error number: 3199; Symbol: ER_BINLOG_UNSAFE_XA; SQLSTATE: HY000

  Message: Statement is unsafe because it is being used inside a XA transaction. Concurrent XA transactions may deadlock on slaves when replicated using statements.

  ER_BINLOG_UNSAFE_XA was added in 8.0.2.

- Error number: 3200; Symbol: ER_UDF_ERROR; SQLSTATE: HY000

  Message: %s UDF failed; %s

  ER_UDF_ERROR was added in 8.0.4.

- Error number: 3201; Symbol: ER_KEYRING_MIGRATION_FAILURE; SQLSTATE: HY000

  Message: Can not perform keyring migration : %s

  ER_KEYRING_MIGRATION_FAILURE was added in 8.0.4.

- Error number: 3202; Symbol: ER_KEYRING_ACCESS_DENIED_ERROR; SQLSTATE: 42000

  Message: Access denied; you need %s privileges for this operation

  ER_KEYRING_ACCESS_DENIED_ERROR was added in 8.0.4.

- Error number: 3203; Symbol: ER_KEYRING_MIGRATION_STATUS; SQLSTATE: HY000

  Message: Keyring migration %s.

  ER_KEYRING_MIGRATION_STATUS was added in 8.0.4.

- Error number: 3218; Symbol: ER_AUDIT_LOG_UDF_READ_INVALID_MAX_ARRAY_LENGTH_ARG_VALUE; SQLSTATE: HY000

  Message: Invalid "max_array_length" argument value.

  ER_AUDIT_LOG_UDF_READ_INVALID_MAX_ARRAY_LENGTH_ARG_VALUE was added in 8.0.11.

- Error number: 3500; Symbol: ER_UNSUPPORT_COMPRESSED_TEMPORARY_TABLE; SQLSTATE: HY000

  Message: CREATE TEMPORARY TABLE is not allowed with ROW_FORMAT=COMPRESSED or KEY_BLOCK_SIZE.

- Error number: 3501; Symbol: ER_ACL_OPERATION_FAILED; SQLSTATE: HY000

  Message: The ACL operation failed due to the following error from SE: errcode %d - %s

- Error number: 3502; Symbol: ER_UNSUPPORTED_INDEX_ALGORITHM; SQLSTATE: HY000

  Message: This storage engine does not support the %s index algorithm, storage engine default was used instead.

- Error number: 3503; Symbol: ER_NO_SUCH_DB; SQLSTATE: 42Y07

  Message: Database '%s' doesn't exist

- Error number: 3504; Symbol: ER_TOO_BIG_ENUM; SQLSTATE: HY000

  Message: Too many enumeration values for column %s.

- Error number: 3505; Symbol: ER_TOO_LONG_SET_ENUM_VALUE; SQLSTATE: HY000

  Message: Too long enumeration/set value for column %s.

- Error number: 3506; Symbol: ER_INVALID_DD_OBJECT; SQLSTATE: HY000

Message: %s dictionary object is invalid. (%s)

- Error number: 3507; Symbol: ER_UPDATING_DD_TABLE; SQLSTATE: HY000

  Message: Failed to update %s dictionary object.

- Error number: 3508; Symbol: ER_INVALID_DD_OBJECT_ID; SQLSTATE: HY000

  Message: Dictionary object id (%lu) does not exist.

- Error number: 3509; Symbol: ER_INVALID_DD_OBJECT_NAME; SQLSTATE: HY000

  Message: Dictionary object name '%s' is invalid. (%s)

- Error number: 3510; Symbol: ER_TABLESPACE_MISSING_WITH_NAME; SQLSTATE: HY000

  Message: Tablespace %s doesn't exist.

- Error number: 3511; Symbol: ER_TOO_LONG_ROUTINE_COMMENT; SQLSTATE: HY000

  Message: Comment for routine '%s' is too long (max = %lu)

- Error number: 3512; Symbol: ER_SP_LOAD_FAILED; SQLSTATE: HY000

  Message: Failed to load routine '%s'.

- Error number: 3513; Symbol: ER_INVALID_BITWISE_OPERANDS_SIZE; SQLSTATE: HY000

  Message: Binary operands of bitwise operators must be of equal length

- Error number: 3514; Symbol: ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE; SQLSTATE: HY000

  Message: Aggregate bitwise functions cannot accept arguments longer than 511 bytes; consider using the SUBSTRING() function

- Error number: 3515; Symbol: ER_WARN_UNSUPPORTED_HINT; SQLSTATE: HY000

  Message: Hints aren't supported in %s

- Error number: 3516; Symbol: ER_UNEXPECTED_GEOMETRY_TYPE; SQLSTATE: 22S01

  Message: %s value is a geometry of unexpected type %s in %s.

- Error number: 3517; Symbol: ER_SRS_PARSE_ERROR; SQLSTATE: SR002

  Message: Can't parse the spatial reference system definition of SRID %u.

- Error number: 3518; Symbol: ER_SRS_PROJ_PARAMETER_MISSING; SQLSTATE: SR003

  Message: The spatial reference system definition for SRID %u does not specify the mandatory %s (EPSG %u) projection parameter.

- Error number: 3519; Symbol: ER_WARN_SRS_NOT_FOUND; SQLSTATE: 01000

  Message: There's no spatial reference system with SRID %u.

- Error number: 3520; Symbol: ER_SRS_NOT_CARTESIAN; SQLSTATE: 22S00

Message: Function %s is only defined for Cartesian spatial reference systems, but one of its arguments is in SRID %u, which is not Cartesian.

- Error number: 3521; Symbol: ER_SRS_NOT_CARTESIAN_UNDEFINED; SQLSTATE: SR001

  Message: Function %s is only defined for Cartesian spatial reference systems, but one of its arguments is in SRID %u, which has not been defined.

- Error number: 3522; Symbol: ER_PK_INDEX_CANT_BE_INVISIBLE; SQLSTATE: HY000

  Message: A primary key index cannot be invisible

- Error number: 3523; Symbol: ER_UNKNOWN_AUTHID; SQLSTATE: HY000

  Message: Unknown authorization ID `%s`@`%s`

- Error number: 3524; Symbol: ER_FAILED_ROLE_GRANT; SQLSTATE: HY000

  Message: Failed to grant %s` to %s

- Error number: 3525; Symbol: ER_OPEN_ROLE_TABLES; SQLSTATE: HY000

  Message: Failed to open the security system tables

- Error number: 3526; Symbol: ER_FAILED_DEFAULT_ROLES; SQLSTATE: HY000

  Message: Failed to set default roles

- Error number: 3527; Symbol: ER_COMPONENTS_NO_SCHEME; SQLSTATE: HY000

  Message: Cannot find schema in specified URN: '%s'.

- Error number: 3528; Symbol: ER_COMPONENTS_NO_SCHEME_SERVICE; SQLSTATE: HY000

  Message: Cannot acquire scheme load service implementation for schema '%s' in specified URN: '%s'.

- Error number: 3529; Symbol: ER_COMPONENTS_CANT_LOAD; SQLSTATE: HY000

  Message: Cannot load component from specified URN: '%s'.

- Error number: 3530; Symbol: ER_ROLE_NOT_GRANTED; SQLSTATE: HY000

  Message: `%s`@`%s` is not granted to `%s`@`%s`

- Error number: 3531; Symbol: ER_FAILED_REVOKE_ROLE; SQLSTATE: HY000

  Message: Could not revoke role from `%s`@`%s`

- Error number: 3532; Symbol: ER_RENAME_ROLE; SQLSTATE: HY000

  Message: Renaming of a role identifier is forbidden

- Error number: 3533; Symbol: ER_COMPONENTS_CANT_ACQUIRE_SERVICE_IMPLEMENTATION; SQLSTATE: HY000

  Message: Cannot acquire specified service implementation: '%s'.

- Error number: 3534; Symbol: ER_COMPONENTS_CANT_SATISFY_DEPENDENCY; SQLSTATE: HY000

Message: Cannot satisfy dependency for service '%s' required by component '%s'.

- Error number: 3535; Symbol: ER_COMPONENTS_LOAD_CANT_REGISTER_SERVICE_IMPLEMENTATION; SQLSTATE: HY000

  Message: Cannot register service implementation '%s' provided by component '%s'.

- Error number: 3536; Symbol: ER_COMPONENTS_LOAD_CANT_INITIALIZE; SQLSTATE: HY000

  Message: Initialization method provided by component '%s' failed.

- Error number: 3537; Symbol: ER_COMPONENTS_UNLOAD_NOT_LOADED; SQLSTATE: HY000

  Message: Component specified by URN '%s' to unload has not been loaded before.

- Error number: 3538; Symbol: ER_COMPONENTS_UNLOAD_CANT_DEINITIALIZE; SQLSTATE: HY000

  Message: De-initialization method provided by component '%s' failed.

- Error number: 3539; Symbol: ER_COMPONENTS_CANT_RELEASE_SERVICE; SQLSTATE: HY000

  Message: Release of previously acquired service implementation failed.

- Error number: 3540; Symbol: ER_COMPONENTS_UNLOAD_CANT_UNREGISTER_SERVICE; SQLSTATE: HY000

  Message: Unregistration of service implementation '%s' provided by component '%s' failed during unloading of the component.

- Error number: 3541; Symbol: ER_COMPONENTS_CANT_UNLOAD; SQLSTATE: HY000

  Message: Cannot unload component from specified URN: '%s'.

- Error number: 3542; Symbol: ER_WARN_UNLOAD_THE_NOT_PERSISTED; SQLSTATE: HY000

  Message: The Persistent Dynamic Loader was used to unload a component '%s', but it was not used to load that component before.

- Error number: 3543; Symbol: ER_COMPONENT_TABLE_INCORRECT; SQLSTATE: HY000

  Message: The mysql.component table is missing or has an incorrect definition.

- Error number: 3544; Symbol: ER_COMPONENT_MANIPULATE_ROW_FAILED; SQLSTATE: HY000

  Message: Failed to manipulate component '%s' persistence data. Error code %d from storage engine.

- Error number: 3545; Symbol: ER_COMPONENTS_UNLOAD_DUPLICATE_IN_GROUP; SQLSTATE: HY000

  Message: The component with specified URN: '%s' was specified in group more than once.

- Error number: 3546; Symbol: ER_CANT_SET_GTID_PURGED_DUE_SETS_CONSTRAINTS; SQLSTATE: HY000

  Message: @@GLOBAL.GTID_PURGED cannot be changed: %s

- Error number: 3547; Symbol: ER_CANNOT_LOCK_USER_MANAGEMENT_CACHES; SQLSTATE: HY000

  Message: Can not lock user management caches for processing.

- Error number: 3548; Symbol: ER_SRS_NOT_FOUND; SQLSTATE: SR001

  Message: There's no spatial reference system with SRID %u.

- Error number: 3549; Symbol: ER_VARIABLE_NOT_PERSISTED; SQLSTATE: HY000

  Message: Variables cannot be persisted. Please retry.

- Error number: 3550; Symbol: ER_IS_QUERY_INVALID_CLAUSE; SQLSTATE: HY000

  Message: Information schema queries do not support the '%s' clause.

- Error number: 3551; Symbol: ER_UNABLE_TO_STORE_STATISTICS; SQLSTATE: HY000

  Message: Unable to store dynamic %s statistics into data dictionary.

- Error number: 3552; Symbol: ER_NO_SYSTEM_SCHEMA_ACCESS; SQLSTATE: HY000

  Message: Access to system schema '%s' is rejected.

- Error number: 3553; Symbol: ER_NO_SYSTEM_TABLESPACE_ACCESS; SQLSTATE: HY000

  Message: Access to system tablespace '%s' is rejected.

- Error number: 3554; Symbol: ER_NO_SYSTEM_TABLE_ACCESS; SQLSTATE: HY000

  Message: Access to %s '%s.%s' is rejected.

- Error number: 3555; Symbol: ER_NO_SYSTEM_TABLE_ACCESS_FOR_DICTIONARY_TABLE; SQLSTATE: HY000

  Message: data dictionary table

  ER_NO_SYSTEM_TABLE_ACCESS_FOR_DICTIONARY_TABLE was added in 8.0.1.

- Error number: 3556; Symbol: ER_NO_SYSTEM_TABLE_ACCESS_FOR_SYSTEM_TABLE; SQLSTATE: HY000

  Message: system table

  ER_NO_SYSTEM_TABLE_ACCESS_FOR_SYSTEM_TABLE was added in 8.0.1.

- Error number: 3557; Symbol: ER_NO_SYSTEM_TABLE_ACCESS_FOR_TABLE; SQLSTATE: HY000

  Message: table

  ER_NO_SYSTEM_TABLE_ACCESS_FOR_TABLE was added in 8.0.1.

- Error number: 3558; Symbol: ER_INVALID_OPTION_KEY; SQLSTATE: 22023

  Message: Invalid option key '%s' in function %s.

  ER_INVALID_OPTION_KEY was added in 8.0.1.

- Error number: 3559; Symbol: ER_INVALID_OPTION_VALUE; SQLSTATE: 22023

  Message: Invalid value '%s' for option '%s' in function '%s'.

  ER_INVALID_OPTION_VALUE was added in 8.0.1.

- Error number: 3560; Symbol: ER_INVALID_OPTION_KEY_VALUE_PAIR; SQLSTATE: 22023

  Message: The string '%s' is not a valid key %c value pair in function %s.

  ER_INVALID_OPTION_KEY_VALUE_PAIR was added in 8.0.1.

- Error number: 3561; Symbol: ER_INVALID_OPTION_START_CHARACTER; SQLSTATE: 22023

  Message: The options argument in function %s starts with the invalid character '%c'.

  ER_INVALID_OPTION_START_CHARACTER was added in 8.0.1.

- Error number: 3562; Symbol: ER_INVALID_OPTION_END_CHARACTER; SQLSTATE: 22023

  Message: The options argument in function %s ends with the invalid character '%c'.

  ER_INVALID_OPTION_END_CHARACTER was added in 8.0.1.

- Error number: 3563; Symbol: ER_INVALID_OPTION_CHARACTERS; SQLSTATE: 22023

  Message: The options argument in function %s contains the invalid character sequence '%s'.

  ER_INVALID_OPTION_CHARACTERS was added in 8.0.1.

- Error number: 3564; Symbol: ER_DUPLICATE_OPTION_KEY; SQLSTATE: 22023

  Message: Duplicate option key '%s' in funtion '%s'.

  ER_DUPLICATE_OPTION_KEY was added in 8.0.1.

- Error number: 3565; Symbol: ER_WARN_SRS_NOT_FOUND_AXIS_ORDER; SQLSTATE: 01000

  Message: There's no spatial reference system with SRID %u. The axis order is unknown.

  ER_WARN_SRS_NOT_FOUND_AXIS_ORDER was added in 8.0.1.

- Error number: 3566; Symbol: ER_NO_ACCESS_TO_NATIVE_FCT; SQLSTATE: HY000

  Message: Access to native function '%s' is rejected.

  ER_NO_ACCESS_TO_NATIVE_FCT was added in 8.0.1.

- Error number: 3567; Symbol: ER_RESET_MASTER_TO_VALUE_OUT_OF_RANGE; SQLSTATE: HY000

  Message: The requested value '%llu' for the next binary log index is out of range. Please use a value between '1' and '%lu'.

  ER_RESET_MASTER_TO_VALUE_OUT_OF_RANGE was added in 8.0.1.

- Error number: 3568; Symbol: ER_UNRESOLVED_TABLE_LOCK; SQLSTATE: HY000

  Message: Unresolved table name %s in locking clause.

  ER_UNRESOLVED_TABLE_LOCK was added in 8.0.1.

- Error number: 3569; Symbol: ER_DUPLICATE_TABLE_LOCK; SQLSTATE: HY000

  Message: Table %s appears in multiple locking clauses.

  ER_DUPLICATE_TABLE_LOCK was added in 8.0.1.

- Error number: 3570; Symbol: ER_BINLOG_UNSAFE_SKIP_LOCKED; SQLSTATE: HY000

  Message: Statement is unsafe because it uses SKIP LOCKED. The set of inserted values is non-deterministic.

  ER_BINLOG_UNSAFE_SKIP_LOCKED was added in 8.0.1.

- Error number: 3571; Symbol: ER_BINLOG_UNSAFE_NOWAIT; SQLSTATE: HY000

  Message: Statement is unsafe because it uses NOWAIT. Whether the command will succeed or fail is not deterministic.

  ER_BINLOG_UNSAFE_NOWAIT was added in 8.0.1.

- Error number: 3572; Symbol: ER_LOCK_NOWAIT; SQLSTATE: HY000

  Message: Statement aborted because lock(s) could not be acquired immediately and NOWAIT is set.

  ER_LOCK_NOWAIT was added in 8.0.1.

- Error number: 3573; Symbol: ER_CTE_RECURSIVE_REQUIRES_UNION; SQLSTATE: HY000

  Message: Recursive Common Table Expression '%s' should contain a UNION

  ER_CTE_RECURSIVE_REQUIRES_UNION was added in 8.0.1.

- Error number: 3574; Symbol: ER_CTE_RECURSIVE_REQUIRES_NONRECURSIVE_FIRST; SQLSTATE: HY000

  Message: Recursive Common Table Expression '%s' should have one or more non-recursive query blocks followed by one or more recursive ones

  ER_CTE_RECURSIVE_REQUIRES_NONRECURSIVE_FIRST was added in 8.0.1.

- Error number: 3575; Symbol: ER_CTE_RECURSIVE_FORBIDS_AGGREGATION; SQLSTATE: HY000

  Message: Recursive Common Table Expression '%s' can contain neither aggregation nor window functions in recursive query block

  ER_CTE_RECURSIVE_FORBIDS_AGGREGATION was added in 8.0.1.

- Error number: 3576; Symbol: ER_CTE_RECURSIVE_FORBIDDEN_JOIN_ORDER; SQLSTATE: HY000

  Message: In recursive query block of Recursive Common Table Expression '%s', the recursive table must neither be in the right argument of a LEFT JOIN, nor be forced to be non-first with join order hints

  ER_CTE_RECURSIVE_FORBIDDEN_JOIN_ORDER was added in 8.0.1.

- Error number: 3577; Symbol: ER_CTE_RECURSIVE_REQUIRES_SINGLE_REFERENCE; SQLSTATE: HY000

  Message: In recursive query block of Recursive Common Table Expression '%s', the recursive table must be referenced only once, and not in any subquery

  ER_CTE_RECURSIVE_REQUIRES_SINGLE_REFERENCE was added in 8.0.1.

- Error number: 3578; Symbol: ER_SWITCH_TMP_ENGINE; SQLSTATE: HY000

  Message: '%s' requires @@internal_tmp_disk_storage_engine=InnoDB

ER_SWITCH_TMP_ENGINE was added in 8.0.1.

- Error number: 3579; Symbol: ER_WINDOW_NO_SUCH_WINDOW; SQLSTATE: HY000

  Message: Window name '%s' is not defined.

  ER_WINDOW_NO_SUCH_WINDOW was added in 8.0.2.

- Error number: 3580; Symbol: ER_WINDOW_CIRCULARITY_IN_WINDOW_GRAPH; SQLSTATE: HY000

  Message: There is a circularity in the window dependency graph.

  ER_WINDOW_CIRCULARITY_IN_WINDOW_GRAPH was added in 8.0.2.

- Error number: 3581; Symbol: ER_WINDOW_NO_CHILD_PARTITIONING; SQLSTATE: HY000

  Message: A window which depends on another cannot define partitioning.

  ER_WINDOW_NO_CHILD_PARTITIONING was added in 8.0.2.

- Error number: 3582; Symbol: ER_WINDOW_NO_INHERIT_FRAME; SQLSTATE: HY000

  Message: Window '%s' has a frame definition, so cannot be referenced by another window.

  ER_WINDOW_NO_INHERIT_FRAME was added in 8.0.2.

- Error number: 3583; Symbol: ER_WINDOW_NO_REDEFINE_ORDER_BY; SQLSTATE: HY000

  Message: Window '%s' cannot inherit '%s' since both contain an ORDER BY clause.

  ER_WINDOW_NO_REDEFINE_ORDER_BY was added in 8.0.2.

- Error number: 3584; Symbol: ER_WINDOW_FRAME_START_ILLEGAL; SQLSTATE: HY000

  Message: Window '%s': frame start cannot be UNBOUNDED FOLLOWING.

  ER_WINDOW_FRAME_START_ILLEGAL was added in 8.0.2.

- Error number: 3585; Symbol: ER_WINDOW_FRAME_END_ILLEGAL; SQLSTATE: HY000

  Message: Window '%s': frame end cannot be UNBOUNDED PRECEDING.

  ER_WINDOW_FRAME_END_ILLEGAL was added in 8.0.2.

- Error number: 3586; Symbol: ER_WINDOW_FRAME_ILLEGAL; SQLSTATE: HY000

  Message: Window '%s': frame start or end is negative, NULL or of non-integral type

  ER_WINDOW_FRAME_ILLEGAL was added in 8.0.2.

- Error number: 3587; Symbol: ER_WINDOW_RANGE_FRAME_ORDER_TYPE; SQLSTATE: HY000

  Message: Window '%s' with RANGE N PRECEDING/FOLLOWING frame requires exactly one ORDER BY expression, of numeric or temporal type

  ER_WINDOW_RANGE_FRAME_ORDER_TYPE was added in 8.0.2.

- Error number: 3588; Symbol: ER_WINDOW_RANGE_FRAME_TEMPORAL_TYPE; SQLSTATE: HY000

Message: Window '%s' with RANGE frame has ORDER BY expression of datetime type. Only INTERVAL bound value allowed.

ER_WINDOW_RANGE_FRAME_TEMPORAL_TYPE was added in 8.0.2.

- Error number: 3589; Symbol: ER_WINDOW_RANGE_FRAME_NUMERIC_TYPE; SQLSTATE: HY000

  Message: Window '%s' with RANGE frame has ORDER BY expression of numeric type, INTERVAL bound value not allowed.

  ER_WINDOW_RANGE_FRAME_NUMERIC_TYPE was added in 8.0.2.

- Error number: 3590; Symbol: ER_WINDOW_RANGE_BOUND_NOT_CONSTANT; SQLSTATE: HY000

  Message: Window '%s' has a non-constant frame bound.

  ER_WINDOW_RANGE_BOUND_NOT_CONSTANT was added in 8.0.2.

- Error number: 3591; Symbol: ER_WINDOW_DUPLICATE_NAME; SQLSTATE: HY000

  Message: Window '%s' is defined twice.

  ER_WINDOW_DUPLICATE_NAME was added in 8.0.2.

- Error number: 3592; Symbol: ER_WINDOW_ILLEGAL_ORDER_BY; SQLSTATE: HY000

  Message: Window '%s': ORDER BY or PARTITION BY uses legacy position indication which is not supported, use expression.

  ER_WINDOW_ILLEGAL_ORDER_BY was added in 8.0.2.

- Error number: 3593; Symbol: ER_WINDOW_INVALID_WINDOW_FUNC_USE; SQLSTATE: HY000

  Message: You cannot use the window function '%s' in this context.'

  ER_WINDOW_INVALID_WINDOW_FUNC_USE was added in 8.0.2.

- Error number: 3594; Symbol: ER_WINDOW_INVALID_WINDOW_FUNC_ALIAS_USE; SQLSTATE: HY000

  Message: You cannot use the alias '%s' of an expression containing a window function in this context.'

  ER_WINDOW_INVALID_WINDOW_FUNC_ALIAS_USE was added in 8.0.2.

- Error number: 3595; Symbol: ER_WINDOW_NESTED_WINDOW_FUNC_USE_IN_WINDOW_SPEC; SQLSTATE: HY000

  Message: You cannot nest a window function in the specification of window '%s'.

  ER_WINDOW_NESTED_WINDOW_FUNC_USE_IN_WINDOW_SPEC was added in 8.0.2.

- Error number: 3596; Symbol: ER_WINDOW_ROWS_INTERVAL_USE; SQLSTATE: HY000

  Message: Window '%s': INTERVAL can only be used with RANGE frames.

  ER_WINDOW_ROWS_INTERVAL_USE was added in 8.0.2.

- Error number: 3597; Symbol: ER_WINDOW_NO_GROUP_ORDER; SQLSTATE: HY000

Message: ASC or DESC with GROUP BY isn't allowed with window functions; put ASC or DESC in ORDER BY

ER_WINDOW_NO_GROUP_ORDER was added in 8.0.2, removed after 8.0.12.

- Error number: 3597; Symbol: ER_WINDOW_NO_GROUP_ORDER_UNUSED; SQLSTATE: HY000

  Message: ASC or DESC with GROUP BY isn't allowed with window functions; put ASC or DESC in ORDER BY

  ER_WINDOW_NO_GROUP_ORDER_UNUSED was added in 8.0.13.

- Error number: 3598; Symbol: ER_WINDOW_EXPLAIN_JSON; SQLSTATE: HY000

  Message: To get information about window functions use EXPLAIN FORMAT=JSON

  ER_WINDOW_EXPLAIN_JSON was added in 8.0.2.

- Error number: 3599; Symbol: ER_WINDOW_FUNCTION_IGNORES_FRAME; SQLSTATE: HY000

  Message: Window function '%s' ignores the frame clause of window '%s' and aggregates over the whole partition

  ER_WINDOW_FUNCTION_IGNORES_FRAME was added in 8.0.2.

- Error number: 3600; Symbol: ER_WINDOW_SE_NOT_ACCEPTABLE; SQLSTATE: HY000

  Message: Windowing requires @@internal_tmp_mem_storage_engine=TempTable.

  ER_WINDOW_SE_NOT_ACCEPTABLE was added in 8.0.2, removed after 8.0.3.

- Error number: 3600; Symbol: ER_WL9236_NOW_UNUSED; SQLSTATE: HY000

  Message: Windowing requires @@internal_tmp_mem_storage_engine=TempTable.

  ER_WL9236_NOW_UNUSED was added in 8.0.4.

- Error number: 3601; Symbol: ER_INVALID_NO_OF_ARGS; SQLSTATE: HY000

  Message: Too many arguments for function %s: %lu; maximum allowed is %s.

  ER_INVALID_NO_OF_ARGS was added in 8.0.1.

- Error number: 3602; Symbol: ER_FIELD_IN_GROUPING_NOT_GROUP_BY; SQLSTATE: HY000

  Message: Argument #%u of GROUPING function is not in GROUP BY

  ER_FIELD_IN_GROUPING_NOT_GROUP_BY was added in 8.0.1.

- Error number: 3603; Symbol: ER_TOO_LONG_TABLESPACE_COMMENT; SQLSTATE: HY000

  Message: Comment for tablespace '%s' is too long (max = %lu)

  ER_TOO_LONG_TABLESPACE_COMMENT was added in 8.0.1.

- Error number: 3604; Symbol: ER_ENGINE_CANT_DROP_TABLE; SQLSTATE: HY000

  Message: Storage engine can't drop table '%s'

ER_ENGINE_CANT_DROP_TABLE was added in 8.0.1.

- Error number: 3605; Symbol: ER_ENGINE_CANT_DROP_MISSING_TABLE; SQLSTATE: HY000

  Message: Storage engine can't drop table '%s' because it is missing. Use DROP TABLE IF EXISTS to remove it from data-dictionary.

  ER_ENGINE_CANT_DROP_MISSING_TABLE was added in 8.0.1.

- Error number: 3606; Symbol: ER_TABLESPACE_DUP_FILENAME; SQLSTATE: HY000

  Message: Duplicate file name for tablespace '%s'

  ER_TABLESPACE_DUP_FILENAME was added in 8.0.1.

- Error number: 3607; Symbol: ER_DB_DROP_RMDIR2; SQLSTATE: HY000

  Message: Problem while dropping database. Can't remove database directory (%s). Please remove it manually.

  ER_DB_DROP_RMDIR2 was added in 8.0.1.

- Error number: 3608; Symbol: ER_IMP_NO_FILES_MATCHED; SQLSTATE: HY000

  Message: No SDI files matched the pattern '%s'

  ER_IMP_NO_FILES_MATCHED was added in 8.0.1.

- Error number: 3609; Symbol: ER_IMP_SCHEMA_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: Schema '%s', referenced in SDI, does not exist.

  ER_IMP_SCHEMA_DOES_NOT_EXIST was added in 8.0.1.

- Error number: 3610; Symbol: ER_IMP_TABLE_ALREADY_EXISTS; SQLSTATE: HY000

  Message: Table '%s.%s', referenced in SDI, already exists.

  ER_IMP_TABLE_ALREADY_EXISTS was added in 8.0.1.

- Error number: 3611; Symbol: ER_IMP_INCOMPATIBLE_MYSQLD_VERSION; SQLSTATE: HY000

  Message: Imported mysqld_version (%llu) is not compatible with current (%llu)

  ER_IMP_INCOMPATIBLE_MYSQLD_VERSION was added in 8.0.1.

- Error number: 3612; Symbol: ER_IMP_INCOMPATIBLE_DD_VERSION; SQLSTATE: HY000

  Message: Imported dd version (%u) is not compatible with current (%u)

  ER_IMP_INCOMPATIBLE_DD_VERSION was added in 8.0.1.

- Error number: 3613; Symbol: ER_IMP_INCOMPATIBLE_SDI_VERSION; SQLSTATE: HY000

  Message: Imported sdi version (%llu) is not compatible with current (%llu)

  ER_IMP_INCOMPATIBLE_SDI_VERSION was added in 8.0.1.

- Error number: 3614; Symbol: ER_WARN_INVALID_HINT; SQLSTATE: HY000

Message: Invalid number of arguments for hint %s

ER_WARN_INVALID_HINT was added in 8.0.1.

- Error number: 3615; Symbol: ER_VAR_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: Variable %s does not exist in persisted config file

  ER_VAR_DOES_NOT_EXIST was added in 8.0.1.

- Error number: 3616; Symbol: ER_LONGITUDE_OUT_OF_RANGE; SQLSTATE: 22S02

  Message: Longitude %f is out of range in function %s. It must be within (%f, %f].

  ER_LONGITUDE_OUT_OF_RANGE was added in 8.0.1.

- Error number: 3617; Symbol: ER_LATITUDE_OUT_OF_RANGE; SQLSTATE: 22S03

  Message: Latitude %f is out of range in function %s. It must be within [%f, %f].

  ER_LATITUDE_OUT_OF_RANGE was added in 8.0.1.

- Error number: 3618; Symbol: ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS; SQLSTATE: 22S00

  Message: %s(%s) has not been implemented for geographic spatial reference systems.

  ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS was added in 8.0.1.

- Error number: 3619; Symbol: ER_ILLEGAL_PRIVILEGE_LEVEL; SQLSTATE: HY000

  Message: Illegal privilege level specified for %s

  ER_ILLEGAL_PRIVILEGE_LEVEL was added in 8.0.1.

- Error number: 3620; Symbol: ER_NO_SYSTEM_VIEW_ACCESS; SQLSTATE: HY000

  Message: Access to system view INFORMATION_SCHEMA.'%s' is rejected.

  ER_NO_SYSTEM_VIEW_ACCESS was added in 8.0.2.

- Error number: 3621; Symbol: ER_COMPONENT_FILTER_FLABBERGASTED; SQLSTATE: HY000

  Message: The log-filter component "%s" got confused at "%s" ...

  ER_COMPONENT_FILTER_FLABBERGASTED was added in 8.0.2.

- Error number: 3622; Symbol: ER_PART_EXPR_TOO_LONG; SQLSTATE: HY000

  Message: Partitioning expression is too long.

  ER_PART_EXPR_TOO_LONG was added in 8.0.2.

- Error number: 3623; Symbol: ER_UDF_DROP_DYNAMICALLY_REGISTERED; SQLSTATE: HY000

  Message: DROP FUNCTION can't drop a dynamically registered user defined function

  ER_UDF_DROP_DYNAMICALLY_REGISTERED was added in 8.0.2.

- Error number: 3624; Symbol: ER_UNABLE_TO_STORE_COLUMN_STATISTICS; SQLSTATE: HY000

Message: Unable to store column statistics for column '%s' in table '%s'.'%s'

ER_UNABLE_TO_STORE_COLUMN_STATISTICS was added in 8.0.2.

- Error number: 3625; Symbol: ER_UNABLE_TO_UPDATE_COLUMN_STATISTICS; SQLSTATE: HY000

  Message: Unable to update column statistics for column '%s' in table '%s'.'%s'

  ER_UNABLE_TO_UPDATE_COLUMN_STATISTICS was added in 8.0.2.

- Error number: 3626; Symbol: ER_UNABLE_TO_DROP_COLUMN_STATISTICS; SQLSTATE: HY000

  Message: Unable to remove column statistics for column '%s' in table '%s'.'%s'

  ER_UNABLE_TO_DROP_COLUMN_STATISTICS was added in 8.0.2.

- Error number: 3627; Symbol: ER_UNABLE_TO_BUILD_HISTOGRAM; SQLSTATE: HY000

  Message: Unable to build histogram statistics for column '%s' in table '%s'.'%s'

  ER_UNABLE_TO_BUILD_HISTOGRAM was added in 8.0.2.

- Error number: 3628; Symbol: ER_MANDATORY_ROLE; SQLSTATE: HY000

  Message: The role %s is a mandatory role and can't be revoked or dropped. The restriction can be lifted by excluding the role identifier from the global variable mandatory_roles.

  ER_MANDATORY_ROLE was added in 8.0.2.

- Error number: 3629; Symbol: ER_MISSING_TABLESPACE_FILE; SQLSTATE: HY000

  Message: Tablespace '%s' does not have a file named '%s'

  ER_MISSING_TABLESPACE_FILE was added in 8.0.3.

- Error number: 3630; Symbol: ER_PERSIST_ONLY_ACCESS_DENIED_ERROR; SQLSTATE: 42000

  Message: Access denied; you need %s privileges for this operation

  ER_PERSIST_ONLY_ACCESS_DENIED_ERROR was added in 8.0.3.

- Error number: 3631; Symbol: ER_CMD_NEED_SUPER; SQLSTATE: HY000

  Message: You need the SUPER privilege for command '%s'

  ER_CMD_NEED_SUPER was added in 8.0.3.

- Error number: 3632; Symbol: ER_PATH_IN_DATADIR; SQLSTATE: HY000

  Message: Path is within the current data directory '%s'

  ER_PATH_IN_DATADIR was added in 8.0.3.

- Error number: 3633; Symbol: ER_DDL_IN_PROGRESS; SQLSTATE: HY000

  Message: Concurrent DDL is performed during the operation. Please try again.

  ER_DDL_IN_PROGRESS was added in 8.0.3, removed after 8.0.16.

- Error number: 3633; Symbol: ER_CLONE_DDL_IN_PROGRESS; SQLSTATE: HY000

  Message: Concurrent DDL is performed during clone operation. Please try again.

  ER_CLONE_DDL_IN_PROGRESS was added in 8.0.17.

- Error number: 3634; Symbol: ER_TOO_MANY_CONCURRENT_CLONES; SQLSTATE: HY000

  Message: Too many concurrent clone operations. Maximum allowed - %d.

  ER_TOO_MANY_CONCURRENT_CLONES was added in 8.0.3, removed after 8.0.16.

- Error number: 3634; Symbol: ER_CLONE_TOO_MANY_CONCURRENT_CLONES; SQLSTATE: HY000

  Message: Too many concurrent clone operations. Maximum allowed - %d.

  ER_CLONE_TOO_MANY_CONCURRENT_CLONES was added in 8.0.17.

- Error number: 3635; Symbol: ER_APPLIER_LOG_EVENT_VALIDATION_ERROR; SQLSTATE: HY000

  Message: The table in transaction %s does not comply with the requirements by an external plugin.

  ER_APPLIER_LOG_EVENT_VALIDATION_ERROR was added in 8.0.3.

- Error number: 3636; Symbol: ER_CTE_MAX_RECURSION_DEPTH; SQLSTATE: HY000

  Message: Recursive query aborted after %u iterations. Try increasing @@cte_max_recursion_depth to a larger value.

  ER_CTE_MAX_RECURSION_DEPTH was added in 8.0.3.

- Error number: 3637; Symbol: ER_NOT_HINT_UPDATABLE_VARIABLE; SQLSTATE: HY000

  Message: Variable %s cannot be set using SET_VAR hint.

  ER_NOT_HINT_UPDATABLE_VARIABLE was added in 8.0.3.

- Error number: 3638; Symbol: ER_CREDENTIALS_CONTRADICT_TO_HISTORY; SQLSTATE: HY000

  Message: Cannot use these credentials for '%.*s@%.*s' because they contradict the password history policy

  ER_CREDENTIALS_CONTRADICT_TO_HISTORY was added in 8.0.3.

- Error number: 3639; Symbol: ER_WARNING_PASSWORD_HISTORY_CLAUSES_VOID; SQLSTATE: HY000

  Message: Non-zero password history clauses ignored for user '%s'@'%s' as its authentication plugin %s does not support password history

  ER_WARNING_PASSWORD_HISTORY_CLAUSES_VOID was added in 8.0.3.

- Error number: 3640; Symbol: ER_CLIENT_DOES_NOT_SUPPORT; SQLSTATE: HY000

  Message: The client doesn't support %s

  ER_CLIENT_DOES_NOT_SUPPORT was added in 8.0.3.

- Error number: 3641; Symbol: ER_I_S_SKIPPED_TABLESPACE; SQLSTATE: HY000

Message: Tablespace '%s' was skipped since its definition is being modified by concurrent DDL statement

ER_I_S_SKIPPED_TABLESPACE was added in 8.0.3.

- Error number: 3642; Symbol: ER_TABLESPACE_ENGINE_MISMATCH; SQLSTATE: HY000

  Message: Engine '%s' does not match stored engine '%s' for tablespace '%s'

  ER_TABLESPACE_ENGINE_MISMATCH was added in 8.0.3.

- Error number: 3643; Symbol: ER_WRONG_SRID_FOR_COLUMN; SQLSTATE: HY000

  Message: The SRID of the geometry does not match the SRID of the column '%s'. The SRID of the geometry is %lu, but the SRID of the column is %lu. Consider changing the SRID of the geometry or the SRID property of the column.

  ER_WRONG_SRID_FOR_COLUMN was added in 8.0.3.

- Error number: 3644; Symbol: ER_CANNOT_ALTER_SRID_DUE_TO_INDEX; SQLSTATE: HY000

  Message: The SRID specification on the column '%s' cannot be changed because there is a spatial index on the column. Please remove the spatial index before altering the SRID specification.

  ER_CANNOT_ALTER_SRID_DUE_TO_INDEX was added in 8.0.3.

- Error number: 3645; Symbol: ER_WARN_BINLOG_PARTIAL_UPDATES_DISABLED; SQLSTATE: HY000

  Message: When %s, the option binlog_row_value_options=%s will be ignored and updates will be written in full format to binary log.

  ER_WARN_BINLOG_PARTIAL_UPDATES_DISABLED was added in 8.0.3.

- Error number: 3646; Symbol: ER_WARN_BINLOG_V1_ROW_EVENTS_DISABLED; SQLSTATE: HY000

  Message: When %s, the option log_bin_use_v1_row_events=1 will be ignored and row events will be written in new format to binary log.

  ER_WARN_BINLOG_V1_ROW_EVENTS_DISABLED was added in 8.0.3.

- Error number: 3647; Symbol: ER_WARN_BINLOG_PARTIAL_UPDATES_SUGGESTS_PARTIAL_IMAGES; SQLSTATE: HY000

  Message: When %s, the option binlog_row_value_options=%s will be used only for the after-image. Full values will be written in the before-image, so the saving in disk space due to binlog_row_value_options is limited to less than 50%%.

  ER_WARN_BINLOG_PARTIAL_UPDATES_SUGGESTS_PARTIAL_IMAGES was added in 8.0.3.

- Error number: 3648; Symbol: ER_COULD_NOT_APPLY_JSON_DIFF; SQLSTATE: HY000

  Message: Could not apply JSON diff in table %.*s, column %s.

  ER_COULD_NOT_APPLY_JSON_DIFF was added in 8.0.3.

- Error number: 3649; Symbol: ER_CORRUPTED_JSON_DIFF; SQLSTATE: HY000

Message: Corrupted JSON diff for table %.*s, column %s.

ER_CORRUPTED_JSON_DIFF was added in 8.0.3.

- Error number: 3650; Symbol: ER_RESOURCE_GROUP_EXISTS; SQLSTATE: HY000

  Message: Resource Group '%s' exists

  ER_RESOURCE_GROUP_EXISTS was added in 8.0.3.

- Error number: 3651; Symbol: ER_RESOURCE_GROUP_NOT_EXISTS; SQLSTATE: HY000

  Message: Resource Group '%s' does not exist.

  ER_RESOURCE_GROUP_NOT_EXISTS was added in 8.0.3.

- Error number: 3652; Symbol: ER_INVALID_VCPU_ID; SQLSTATE: HY000

  Message: Invalid cpu id %u

  ER_INVALID_VCPU_ID was added in 8.0.3.

- Error number: 3653; Symbol: ER_INVALID_VCPU_RANGE; SQLSTATE: HY000

  Message: Invalid VCPU range %u-%u

  ER_INVALID_VCPU_RANGE was added in 8.0.3.

- Error number: 3654; Symbol: ER_INVALID_THREAD_PRIORITY; SQLSTATE: HY000

  Message: Invalid thread priority value %d for %s resource group %s. Allowed range is [%d, %d].

  ER_INVALID_THREAD_PRIORITY was added in 8.0.3.

- Error number: 3655; Symbol: ER_DISALLOWED_OPERATION; SQLSTATE: HY000

  Message: %s operation is disallowed on %s

  ER_DISALLOWED_OPERATION was added in 8.0.3.

- Error number: 3656; Symbol: ER_RESOURCE_GROUP_BUSY; SQLSTATE: HY000

  Message: Resource group %s is busy.

  ER_RESOURCE_GROUP_BUSY was added in 8.0.3.

- Error number: 3657; Symbol: ER_RESOURCE_GROUP_DISABLED; SQLSTATE: HY000

  Message: Resource group %s is disabled.

  ER_RESOURCE_GROUP_DISABLED was added in 8.0.3.

- Error number: 3658; Symbol: ER_FEATURE_UNSUPPORTED; SQLSTATE: HY000

  Message: Feature %s is unsupported (%s).

  ER_FEATURE_UNSUPPORTED was added in 8.0.3.

- Error number: 3659; Symbol: ER_ATTRIBUTE_IGNORED; SQLSTATE: HY000

Message: Attribute %s is ignored (%s).

ER_ATTRIBUTE_IGNORED was added in 8.0.3.

- Error number: 3660; Symbol: ER_INVALID_THREAD_ID; SQLSTATE: HY000

  Message: Invalid thread id (%llu).

  ER_INVALID_THREAD_ID was added in 8.0.3.

- Error number: 3661; Symbol: ER_RESOURCE_GROUP_BIND_FAILED; SQLSTATE: HY000

  Message: Unable to bind resource group %s with thread id (%llu).(%s).

  ER_RESOURCE_GROUP_BIND_FAILED was added in 8.0.3.

- Error number: 3662; Symbol: ER_INVALID_USE_OF_FORCE_OPTION; SQLSTATE: HY000

  Message: Option FORCE invalid as DISABLE option is not specified.

  ER_INVALID_USE_OF_FORCE_OPTION was added in 8.0.3.

- Error number: 3663; Symbol: ER_GROUP_REPLICATION_COMMAND_FAILURE; SQLSTATE: HY000

  Message: The %s command encountered a failure. %s

  ER_GROUP_REPLICATION_COMMAND_FAILURE was added in 8.0.4.

- Error number: 3664; Symbol: ER_SDI_OPERATION_FAILED; SQLSTATE: HY000

  Message: Failed to %s SDI '%s.%s' in tablespace '%s'.

  ER_SDI_OPERATION_FAILED was added in 8.0.3.

- Error number: 3665; Symbol: ER_MISSING_JSON_TABLE_VALUE; SQLSTATE: 22035

  Message: Missing value for JSON_TABLE column '%s'

  ER_MISSING_JSON_TABLE_VALUE was added in 8.0.4.

- Error number: 3666; Symbol: ER_WRONG_JSON_TABLE_VALUE; SQLSTATE: 2203F

  Message: Can't store an array or an object in the scalar JSON_TABLE column '%s'

  ER_WRONG_JSON_TABLE_VALUE was added in 8.0.4.

- Error number: 3667; Symbol: ER_TF_MUST_HAVE_ALIAS; SQLSTATE: 42000

  Message: Every table function must have an alias

  ER_TF_MUST_HAVE_ALIAS was added in 8.0.4.

- Error number: 3668; Symbol: ER_TF_FORBIDDEN_JOIN_TYPE; SQLSTATE: HY000

  Message: INNER or LEFT JOIN must be used for LATERAL references made by '%s'

  ER_TF_FORBIDDEN_JOIN_TYPE was added in 8.0.4.

- Error number: 3669; Symbol: ER_JT_VALUE_OUT_OF_RANGE; SQLSTATE: 22003

Message: Value is out of range for JSON_TABLE's column '%s'

ER_JT_VALUE_OUT_OF_RANGE was added in 8.0.4.

- Error number: 3670; Symbol: ER_JT_MAX_NESTED_PATH; SQLSTATE: 42000

  Message: More than supported %u NESTED PATHs were found in JSON_TABLE '%s'

  ER_JT_MAX_NESTED_PATH was added in 8.0.4.

- Error number: 3671; Symbol: ER_PASSWORD_EXPIRATION_NOT_SUPPORTED_BY_AUTH_METHOD; SQLSTATE: HY000

  Message: The selected authentication method %.*s does not support password expiration

  ER_PASSWORD_EXPIRATION_NOT_SUPPORTED_BY_AUTH_METHOD was added in 8.0.4.

- Error number: 3672; Symbol: ER_INVALID_GEOJSON_CRS_NOT_TOP_LEVEL; SQLSTATE: HY000

  Message: Invalid GeoJSON data provided to function %s: Member 'crs' must be specified in the top level object.

  ER_INVALID_GEOJSON_CRS_NOT_TOP_LEVEL was added in 8.0.4.

- Error number: 3673; Symbol: ER_BAD_NULL_ERROR_NOT_IGNORED; SQLSTATE: 23000

  Message: Column '%s' cannot be null

  ER_BAD_NULL_ERROR_NOT_IGNORED was added in 8.0.4.

- Error number: 3674; Symbol: WARN_USELESS_SPATIAL_INDEX; SQLSTATE: HY000

  Message: The spatial index on column '%s' will not be used by the query optimizer since the column does not have an SRID attribute. Consider adding an SRID attribute to the column.

  WARN_USELESS_SPATIAL_INDEX was added in 8.0.4.

- Error number: 3675; Symbol: ER_DISK_FULL_NOWAIT; SQLSTATE: HY000

  Message: Create table/tablespace '%s' failed, as disk is full

  ER_DISK_FULL_NOWAIT was added in 8.0.4.

- Error number: 3676; Symbol: ER_PARSE_ERROR_IN_DIGEST_FN; SQLSTATE: HY000

  Message: Could not parse argument to digest function: "%s".

  ER_PARSE_ERROR_IN_DIGEST_FN was added in 8.0.4.

- Error number: 3677; Symbol: ER_UNDISCLOSED_PARSE_ERROR_IN_DIGEST_FN; SQLSTATE: HY000

  Message: Could not parse argument to digest function.

  ER_UNDISCLOSED_PARSE_ERROR_IN_DIGEST_FN was added in 8.0.4.

- Error number: 3678; Symbol: ER_SCHEMA_DIR_EXISTS; SQLSTATE: HY000

Message: Schema directory '%s' already exists. This must be resolved manually (e.g. by moving the schema directory to another location).

`ER_SCHEMA_DIR_EXISTS` was added in 8.0.4.

- Error number: `3679`; Symbol: `ER_SCHEMA_DIR_MISSING`; SQLSTATE: `HY000`

  Message: Schema directory '%s' does not exist

  `ER_SCHEMA_DIR_MISSING` was added in 8.0.4.

- Error number: `3680`; Symbol: `ER_SCHEMA_DIR_CREATE_FAILED`; SQLSTATE: `HY000`

  Message: Failed to create schema directory '%s' (errno: %d - %s)

  `ER_SCHEMA_DIR_CREATE_FAILED` was added in 8.0.4.

- Error number: `3681`; Symbol: `ER_SCHEMA_DIR_UNKNOWN`; SQLSTATE: `HY000`

  Message: Schema '%s' does not exist, but schema directory '%s' was found. This must be resolved manually (e.g. by moving the schema directory to another location).

  `ER_SCHEMA_DIR_UNKNOWN` was added in 8.0.4.

- Error number: `3682`; Symbol: `ER_ONLY_IMPLEMENTED_FOR_SRID_0_AND_4326`; SQLSTATE: `22S00`

  Message: Function %s is only defined for SRID 0 and SRID 4326.

  `ER_ONLY_IMPLEMENTED_FOR_SRID_0_AND_4326` was added in 8.0.4.

- Error number: `3683`; Symbol: `ER_BINLOG_EXPIRE_LOG_DAYS_AND_SECS_USED_TOGETHER`; SQLSTATE: `HY000`

  Message: The option expire_logs_days and binlog_expire_logs_seconds cannot be used together. Please use binlog_expire_logs_seconds to set the expire time (expire_logs_days is deprecated)

  `ER_BINLOG_EXPIRE_LOG_DAYS_AND_SECS_USED_TOGETHER` was added in 8.0.11.

- Error number: `3684`; Symbol: `ER_REGEXP_STRING_NOT_TERMINATED`; SQLSTATE: `HY000`

  Message: An output string could not be zero-terminated because the length exceeds the buffer capacity.

  `ER_REGEXP_STRING_NOT_TERMINATED` was added in 8.0.4, removed after 8.0.4.

- Error number: `3684`; Symbol: `ER_REGEXP_BUFFER_OVERFLOW`; SQLSTATE: `HY000`

  Message: The result string is larger than the result buffer.

  `ER_REGEXP_BUFFER_OVERFLOW` was added in 8.0.4.

- Error number: `3685`; Symbol: `ER_REGEXP_ILLEGAL_ARGUMENT`; SQLSTATE: `HY000`

  Message: Illegal argument to a regular expression.

  `ER_REGEXP_ILLEGAL_ARGUMENT` was added in 8.0.4.

- Error number: `3686`; Symbol: `ER_REGEXP_INDEX_OUTOFBOUNDS_ERROR`; SQLSTATE: `HY000`

Message: Index out of bounds in regular expression search.

ER_REGEXP_INDEX_OUTOFBOUNDS_ERROR was added in 8.0.4.

- Error number: 3687; Symbol: ER_REGEXP_INTERNAL_ERROR; SQLSTATE: HY000

  Message: Internal error in the regular expression library.

  ER_REGEXP_INTERNAL_ERROR was added in 8.0.4.

- Error number: 3688; Symbol: ER_REGEXP_RULE_SYNTAX; SQLSTATE: HY000

  Message: Syntax error in regular expression on line %u, character %u.

  ER_REGEXP_RULE_SYNTAX was added in 8.0.4.

- Error number: 3689; Symbol: ER_REGEXP_BAD_ESCAPE_SEQUENCE; SQLSTATE: HY000

  Message: Unrecognized escape sequence in regular expression.

  ER_REGEXP_BAD_ESCAPE_SEQUENCE was added in 8.0.4.

- Error number: 3690; Symbol: ER_REGEXP_UNIMPLEMENTED; SQLSTATE: HY000

  Message: The regular expression contains a feature that is not implemented in this library version.

  ER_REGEXP_UNIMPLEMENTED was added in 8.0.4.

- Error number: 3691; Symbol: ER_REGEXP_MISMATCHED_PAREN; SQLSTATE: HY000

  Message: Mismatched parenthesis in regular expression.

  ER_REGEXP_MISMATCHED_PAREN was added in 8.0.4.

- Error number: 3692; Symbol: ER_REGEXP_BAD_INTERVAL; SQLSTATE: HY000

  Message: Incorrect description of a {min,max} interval.

  ER_REGEXP_BAD_INTERVAL was added in 8.0.4.

- Error number: 3693; Symbol: ER_REGEXP_MAX_LT_MIN; SQLSTATE: HY000

  Message: The maximum is less than the minumum in a {min,max} interval.

  ER_REGEXP_MAX_LT_MIN was added in 8.0.4.

- Error number: 3694; Symbol: ER_REGEXP_INVALID_BACK_REF; SQLSTATE: HY000

  Message: Invalid back-reference in regular expression.

  ER_REGEXP_INVALID_BACK_REF was added in 8.0.4.

- Error number: 3695; Symbol: ER_REGEXP_LOOK_BEHIND_LIMIT; SQLSTATE: HY000

  Message: The look-behind assertion exceeds the limit in regular expression.

  ER_REGEXP_LOOK_BEHIND_LIMIT was added in 8.0.4.

- Error number: 3696; Symbol: ER_REGEXP_MISSING_CLOSE_BRACKET; SQLSTATE: HY000

Message: The regular expression contains an unclosed bracket expression.

ER_REGEXP_MISSING_CLOSE_BRACKET was added in 8.0.4.

- Error number: 3697; Symbol: ER_REGEXP_INVALID_RANGE; SQLSTATE: HY000

  Message: The regular expression contains an [x-y] character range where x comes after y.

  ER_REGEXP_INVALID_RANGE was added in 8.0.4.

- Error number: 3698; Symbol: ER_REGEXP_STACK_OVERFLOW; SQLSTATE: HY000

  Message: Overflow in the regular expression backtrack stack.

  ER_REGEXP_STACK_OVERFLOW was added in 8.0.4.

- Error number: 3699; Symbol: ER_REGEXP_TIME_OUT; SQLSTATE: HY000

  Message: Timeout exceeded in regular expression match.

  ER_REGEXP_TIME_OUT was added in 8.0.4.

- Error number: 3700; Symbol: ER_REGEXP_PATTERN_TOO_BIG; SQLSTATE: HY000

  Message: The regular expression pattern exceeds limits on size or complexity.

  ER_REGEXP_PATTERN_TOO_BIG was added in 8.0.4.

- Error number: 3701; Symbol: ER_CANT_SET_ERROR_LOG_SERVICE; SQLSTATE: HY000

  Message: Value for %s got confusing at or around "%s". Syntax may be wrong, component may not be INSTALLed, or a component that does not support instances may be listed more than once.

  ER_CANT_SET_ERROR_LOG_SERVICE was added in 8.0.4.

- Error number: 3702; Symbol: ER_EMPTY_PIPELINE_FOR_ERROR_LOG_SERVICE; SQLSTATE: HY000

  Message: Setting an empty %s pipeline disables error logging!

  ER_EMPTY_PIPELINE_FOR_ERROR_LOG_SERVICE was added in 8.0.4.

- Error number: 3703; Symbol: ER_COMPONENT_FILTER_DIAGNOSTICS; SQLSTATE: HY000

  Message: filter %s: %s

  ER_COMPONENT_FILTER_DIAGNOSTICS was added in 8.0.4.

- Error number: 3704; Symbol: ER_INNODB_CANNOT_BE_IGNORED; SQLSTATE: HY000

  Message: ignore-builtin-innodb is ignored and will be removed in future releases.

  ER_INNODB_CANNOT_BE_IGNORED was added in 8.0.2, removed after 8.0.2.

- Error number: 3704; Symbol: ER_NOT_IMPLEMENTED_FOR_CARTESIAN_SRS; SQLSTATE: 22S00

  Message: %s(%s) has not been implemented for Cartesian spatial reference systems.

  ER_NOT_IMPLEMENTED_FOR_CARTESIAN_SRS was added in 8.0.4.

- Error number: 3705; Symbol: ER_NOT_IMPLEMENTED_FOR_PROJECTED_SRS; SQLSTATE: 22S00

  Message: %s(%s) has not been implemented for projected spatial reference systems.

  ER_NOT_IMPLEMENTED_FOR_PROJECTED_SRS was added in 8.0.4.

- Error number: 3706; Symbol: ER_NONPOSITIVE_RADIUS; SQLSTATE: 22003

  Message: Invalid radius provided to function %s: Radius must be greater than zero.

  ER_NONPOSITIVE_RADIUS was added in 8.0.4.

- Error number: 3707; Symbol: ER_RESTART_SERVER_FAILED; SQLSTATE: HY000

  Message: Restart server failed (%s).

  ER_RESTART_SERVER_FAILED was added in 8.0.4.

- Error number: 3708; Symbol: ER_SRS_MISSING_MANDATORY_ATTRIBUTE; SQLSTATE: SR006

  Message: Missing mandatory attribute %s.

  ER_SRS_MISSING_MANDATORY_ATTRIBUTE was added in 8.0.4.

- Error number: 3709; Symbol: ER_SRS_MULTIPLE_ATTRIBUTE_DEFINITIONS; SQLSTATE: SR006

  Message: Multiple definitions of attribute %s.

  ER_SRS_MULTIPLE_ATTRIBUTE_DEFINITIONS was added in 8.0.4.

- Error number: 3710; Symbol: ER_SRS_NAME_CANT_BE_EMPTY_OR_WHITESPACE; SQLSTATE: SR006

  Message: The spatial reference system name can't be an empty string or start or end with whitespace.

  ER_SRS_NAME_CANT_BE_EMPTY_OR_WHITESPACE was added in 8.0.4.

- Error number: 3711; Symbol: ER_SRS_ORGANIZATION_CANT_BE_EMPTY_OR_WHITESPACE; SQLSTATE: SR006

  Message: The organization name can't be an empty string or start or end with whitespace.

  ER_SRS_ORGANIZATION_CANT_BE_EMPTY_OR_WHITESPACE was added in 8.0.4.

- Error number: 3712; Symbol: ER_SRS_ID_ALREADY_EXISTS; SQLSTATE: SR004

  Message: There is already a spatial reference system with SRID %u.

  ER_SRS_ID_ALREADY_EXISTS was added in 8.0.4.

- Error number: 3713; Symbol: ER_WARN_SRS_ID_ALREADY_EXISTS; SQLSTATE: 01S00

  Message: There is already a spatial reference system with SRID %u.

  ER_WARN_SRS_ID_ALREADY_EXISTS was added in 8.0.4.

- Error number: 3714; Symbol: ER_CANT_MODIFY_SRID_0; SQLSTATE: SR000

  Message: SRID 0 is not modifiable.

ER_CANT_MODIFY_SRID_0 was added in 8.0.4.

- Error number: 3715; Symbol: ER_WARN_RESERVED_SRID_RANGE; SQLSTATE: 01S01

  Message: The SRID range [%u, %u] has been reserved for system use. SRSs in this range may be added, modified or removed without warning during upgrade.

  ER_WARN_RESERVED_SRID_RANGE was added in 8.0.4.

- Error number: 3716; Symbol: ER_CANT_MODIFY_SRS_USED_BY_COLUMN; SQLSTATE: SR005

  Message: Can't modify SRID %u. There is at least one column depending on it.

  ER_CANT_MODIFY_SRS_USED_BY_COLUMN was added in 8.0.4.

- Error number: 3717; Symbol: ER_SRS_INVALID_CHARACTER_IN_ATTRIBUTE; SQLSTATE: SR006

  Message: Invalid character in attribute %s.

  ER_SRS_INVALID_CHARACTER_IN_ATTRIBUTE was added in 8.0.4.

- Error number: 3718; Symbol: ER_SRS_ATTRIBUTE_STRING_TOO_LONG; SQLSTATE: SR006

  Message: Attribute %s is too long. The maximum length is %u characters.

  ER_SRS_ATTRIBUTE_STRING_TOO_LONG was added in 8.0.4.

- Error number: 3719; Symbol: ER_DEPRECATED_UTF8_ALIAS; SQLSTATE: HY000

  Message: 'utf8' is currently an alias for the character set UTF8MB3, but will be an alias for UTF8MB4 in a future release. Please consider using UTF8MB4 in order to be unambiguous.

  ER_DEPRECATED_UTF8_ALIAS was added in 8.0.11.

- Error number: 3720; Symbol: ER_DEPRECATED_NATIONAL; SQLSTATE: HY000

  Message: NATIONAL/NCHAR/NVARCHAR implies the character set UTF8MB3, which will be replaced by UTF8MB4 in a future release. Please consider using CHAR(x) CHARACTER SET UTF8MB4 in order to be unambiguous.

  ER_DEPRECATED_NATIONAL was added in 8.0.11.

- Error number: 3721; Symbol: ER_INVALID_DEFAULT_UTF8MB4_COLLATION; SQLSTATE: HY000

  Message: Invalid default collation %s: utf8mb4_0900_ai_ci or utf8mb4_general_ci expected

  ER_INVALID_DEFAULT_UTF8MB4_COLLATION was added in 8.0.11.

- Error number: 3722; Symbol: ER_UNABLE_TO_COLLECT_LOG_STATUS; SQLSTATE: HY000

  Message: Unable to collect information for column '%s': %s.

  ER_UNABLE_TO_COLLECT_LOG_STATUS was added in 8.0.11.

- Error number: 3723; Symbol: ER_RESERVED_TABLESPACE_NAME; SQLSTATE: HY000

  Message: The table '%s' may not be created in the reserved tablespace '%s'.

  ER_RESERVED_TABLESPACE_NAME was added in 8.0.11.

- Error number: 3724; Symbol: ER_UNABLE_TO_SET_OPTION; SQLSTATE: HY000

  Message: This option cannot be set %s.

  ER_UNABLE_TO_SET_OPTION was added in 8.0.11.

- Error number: 3725; Symbol: ER_SLAVE_POSSIBLY_DIVERGED_AFTER_DDL; SQLSTATE: HY000

  Message: A commit for an atomic DDL statement was unsuccessful on the master and the slave. The slave supports atomic DDL statements but the master does not, so the action taken by the slave and master might differ. Check that their states have not diverged before proceeding.

  ER_SLAVE_POSSIBLY_DIVERGED_AFTER_DDL was added in 8.0.11.

- Error number: 3726; Symbol: ER_SRS_NOT_GEOGRAPHIC; SQLSTATE: 22S00

  Message: Function %s is only defined for geographic spatial reference systems, but one of its arguments is in SRID %u, which is not geographic.

  ER_SRS_NOT_GEOGRAPHIC was added in 8.0.12.

- Error number: 3727; Symbol: ER_POLYGON_TOO_LARGE; SQLSTATE: 22023

  Message: Function %s encountered a polygon that was too large. Polygons must cover less than half the planet.

  ER_POLYGON_TOO_LARGE was added in 8.0.12.

- Error number: 3728; Symbol: ER_SPATIAL_UNIQUE_INDEX; SQLSTATE: HY000

  Message: Spatial indexes can't be primary or unique indexes.

  ER_SPATIAL_UNIQUE_INDEX was added in 8.0.12.

- Error number: 3729; Symbol: ER_INDEX_TYPE_NOT_SUPPORTED_FOR_SPATIAL_INDEX; SQLSTATE: HY000

  Message: The index type %s is not supported for spatial indexes.

  ER_INDEX_TYPE_NOT_SUPPORTED_FOR_SPATIAL_INDEX was added in 8.0.12.

- Error number: 3730; Symbol: ER_FK_CANNOT_DROP_PARENT; SQLSTATE: HY000

  Message: Cannot drop table '%s' referenced by a foreign key constraint '%s' on table '%s'.

  ER_FK_CANNOT_DROP_PARENT was added in 8.0.12.

- Error number: 3731; Symbol: ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE; SQLSTATE: 22S02

  Message: A parameter of function %s contains a geometry with longitude %f, which is out of range. It must be within (%f, %f].

  ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE was added in 8.0.12.

- Error number: 3732; Symbol: ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE; SQLSTATE: 22S03

Message: A parameter of function %s contains a geometry with latitude %f, which is out of range. It must be within [%f, %f].

ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE was added in 8.0.12.

- Error number: 3733; Symbol: ER_FK_CANNOT_USE_VIRTUAL_COLUMN; SQLSTATE: HY000

  Message: Foreign key '%s' uses virtual column '%s' which is not supported.

  ER_FK_CANNOT_USE_VIRTUAL_COLUMN was added in 8.0.12.

- Error number: 3734; Symbol: ER_FK_NO_COLUMN_PARENT; SQLSTATE: HY000

  Message: Failed to add the foreign key constraint. Missing column '%s' for constraint '%s' in the referenced table '%s'

  ER_FK_NO_COLUMN_PARENT was added in 8.0.12.

- Error number: 3735; Symbol: ER_CANT_SET_ERROR_SUPPRESSION_LIST; SQLSTATE: HY000

  Message: %s: Could not add suppression rule for code "%s". Rule-set may be full, or code may not correspond to an error-log message.

  ER_CANT_SET_ERROR_SUPPRESSION_LIST was added in 8.0.13.

- Error number: 3736; Symbol: ER_SRS_GEOGCS_INVALID_AXES; SQLSTATE: SR002

  Message: The spatial reference system definition for SRID %u specifies invalid geographic axes '%s' and '%s'. One axis must be NORTH or SOUTH and the other must be EAST or WEST.

  ER_SRS_GEOGCS_INVALID_AXES was added in 8.0.13.

- Error number: 3737; Symbol: ER_SRS_INVALID_SEMI_MAJOR_AXIS; SQLSTATE: SR002

  Message: The length of the semi-major axis must be a positive number.

  ER_SRS_INVALID_SEMI_MAJOR_AXIS was added in 8.0.13.

- Error number: 3738; Symbol: ER_SRS_INVALID_INVERSE_FLATTENING; SQLSTATE: SR002

  Message: The inverse flattening must be larger than 1.0, or 0.0 if the ellipsoid is a sphere.

  ER_SRS_INVALID_INVERSE_FLATTENING was added in 8.0.13.

- Error number: 3739; Symbol: ER_SRS_INVALID_ANGULAR_UNIT; SQLSTATE: SR002

  Message: The angular unit conversion factor must be a positive number.

  ER_SRS_INVALID_ANGULAR_UNIT was added in 8.0.13.

- Error number: 3740; Symbol: ER_SRS_INVALID_PRIME_MERIDIAN; SQLSTATE: SR002

  Message: The prime meridian must be within (-180, 180] degrees, specified in the SRS angular unit.

  ER_SRS_INVALID_PRIME_MERIDIAN was added in 8.0.13.

- Error number: 3741; Symbol: ER_TRANSFORM_SOURCE_SRS_NOT_SUPPORTED; SQLSTATE: 22S00

  Message: Transformation from SRID %u is not supported.

ER_TRANSFORM_SOURCE_SRS_NOT_SUPPORTED was added in 8.0.13.

- Error number: 3742; Symbol: ER_TRANSFORM_TARGET_SRS_NOT_SUPPORTED; SQLSTATE: 22S00

  Message: Transformation to SRID %u is not supported.

  ER_TRANSFORM_TARGET_SRS_NOT_SUPPORTED was added in 8.0.13.

- Error number: 3743; Symbol: ER_TRANSFORM_SOURCE_SRS_MISSING_TOWGS84; SQLSTATE: 22S00

  Message: Transformation from SRID %u is not supported. The spatial reference system has no TOWGS84 clause.

  ER_TRANSFORM_SOURCE_SRS_MISSING_TOWGS84 was added in 8.0.13.

- Error number: 3744; Symbol: ER_TRANSFORM_TARGET_SRS_MISSING_TOWGS84; SQLSTATE: 22S00

  Message: Transformation to SRID %u is not supported. The spatial reference system has no TOWGS84 clause.

  ER_TRANSFORM_TARGET_SRS_MISSING_TOWGS84 was added in 8.0.13.

- Error number: 3745; Symbol: ER_TEMP_TABLE_PREVENTS_SWITCH_SESSION_BINLOG_FORMAT; SQLSTATE: HY000

  Message: Changing @@session.binlog_format is disallowed when the session has open temporary table(s). You could wait until these temporary table(s) are dropped and try again.

  ER_TEMP_TABLE_PREVENTS_SWITCH_SESSION_BINLOG_FORMAT was added in 8.0.13.

- Error number: 3746; Symbol: ER_TEMP_TABLE_PREVENTS_SWITCH_GLOBAL_BINLOG_FORMAT; SQLSTATE: HY000

  Message: Changing @@global.binlog_format or @@persist.binlog_format is disallowed when any replication channel has open temporary table(s). You could wait until Slave_open_temp_tables = 0 and try again

  ER_TEMP_TABLE_PREVENTS_SWITCH_GLOBAL_BINLOG_FORMAT was added in 8.0.13.

- Error number: 3747; Symbol: ER_RUNNING_APPLIER_PREVENTS_SWITCH_GLOBAL_BINLOG_FORMAT; SQLSTATE: HY000

  Message: Changing @@global.binlog_format or @@persist.binlog_format is disallowed when any replication channel applier thread is running. You could execute STOP SLAVE SQL_THREAD and try again.

  ER_RUNNING_APPLIER_PREVENTS_SWITCH_GLOBAL_BINLOG_FORMAT was added in 8.0.13.

- Error number: 3748; Symbol: ER_CLIENT_GTID_UNSAFE_CREATE_DROP_TEMP_TABLE_IN_TRX_IN_SBR; SQLSTATE: HY000

  Message: Statement violates GTID consistency: CREATE TEMPORARY TABLE and DROP TEMPORARY TABLE are not allowed inside a transaction or inside a procedure in a transactional context when @@session.binlog_format=STATEMENT.

ER_CLIENT_GTID_UNSAFE_CREATE_DROP_TEMP_TABLE_IN_TRX_IN_SBR was added in 8.0.13.

- Error number: 3750; Symbol: ER_TABLE_WITHOUT_PK; SQLSTATE: HY000

  Message: Unable to create or change a table without a primary key, when the system variable 'sql_require_primary_key' is set. Add a primary key to the table or unset this variable to avoid this message. Note that tables without a primary key can cause performance problems in row-based replication, so please consult your DBA before changing this setting.

  ER_TABLE_WITHOUT_PK was added in 8.0.13.

- Error number: 3751; Symbol: WARN_DATA_TRUNCATED_FUNCTIONAL_INDEX; SQLSTATE: 01000

  Message: Data truncated for functional index '%s' at row %ld

  WARN_DATA_TRUNCATED_FUNCTIONAL_INDEX was added in 8.0.13, removed after 8.0.16.

- Error number: 3751; Symbol: ER_WARN_DATA_TRUNCATED_FUNCTIONAL_INDEX; SQLSTATE: 01000

  Message: Data truncated for functional index '%s' at row %ld

  ER_WARN_DATA_TRUNCATED_FUNCTIONAL_INDEX was added in 8.0.17.

- Error number: 3752; Symbol: ER_WARN_DATA_OUT_OF_RANGE_FUNCTIONAL_INDEX; SQLSTATE: 22003

  Message: Value is out of range for functional index '%s' at row %ld

  ER_WARN_DATA_OUT_OF_RANGE_FUNCTIONAL_INDEX was added in 8.0.13.

- Error number: 3753; Symbol: ER_FUNCTIONAL_INDEX_ON_JSON_OR_GEOMETRY_FUNCTION; SQLSTATE: 42000

  Message: Cannot create a functional index on a function that returns a JSON or GEOMETRY value.

  ER_FUNCTIONAL_INDEX_ON_JSON_OR_GEOMETRY_FUNCTION was added in 8.0.13.

- Error number: 3754; Symbol: ER_FUNCTIONAL_INDEX_REF_AUTO_INCREMENT; SQLSTATE: HY000

  Message: Functional index '%s' cannot refer to an auto-increment column.

  ER_FUNCTIONAL_INDEX_REF_AUTO_INCREMENT was added in 8.0.13.

- Error number: 3755; Symbol: ER_CANNOT_DROP_COLUMN_FUNCTIONAL_INDEX; SQLSTATE: HY000

  Message: Cannot drop column '%s' because it is used by a functional index. In order to drop the column, you must remove the functional index.

  ER_CANNOT_DROP_COLUMN_FUNCTIONAL_INDEX was added in 8.0.13.

- Error number: 3756; Symbol: ER_FUNCTIONAL_INDEX_PRIMARY_KEY; SQLSTATE: HY000

  Message: The primary key cannot be a functional index

  ER_FUNCTIONAL_INDEX_PRIMARY_KEY was added in 8.0.13.

- Error number: 3757; Symbol: ER_FUNCTIONAL_INDEX_ON_LOB; SQLSTATE: HY000

Message: Cannot create a functional index on an expression that returns a BLOB or TEXT. Please consider using CAST.

ER_FUNCTIONAL_INDEX_ON_LOB was added in 8.0.13.

- Error number: 3758; Symbol: ER_FUNCTIONAL_INDEX_FUNCTION_IS_NOT_ALLOWED; SQLSTATE: HY000

  Message: Expression of functional index '%s' contains a disallowed function.

  ER_FUNCTIONAL_INDEX_FUNCTION_IS_NOT_ALLOWED was added in 8.0.13.

- Error number: 3759; Symbol: ER_FULLTEXT_FUNCTIONAL_INDEX; SQLSTATE: HY000

  Message: Fulltext functional index is not supported.

  ER_FULLTEXT_FUNCTIONAL_INDEX was added in 8.0.13.

- Error number: 3760; Symbol: ER_SPATIAL_FUNCTIONAL_INDEX; SQLSTATE: HY000

  Message: Spatial functional index is not supported.

  ER_SPATIAL_FUNCTIONAL_INDEX was added in 8.0.13.

- Error number: 3761; Symbol: ER_WRONG_KEY_COLUMN_FUNCTIONAL_INDEX; SQLSTATE: HY000

  Message: The used storage engine cannot index the expression '%s'.

  ER_WRONG_KEY_COLUMN_FUNCTIONAL_INDEX was added in 8.0.13.

- Error number: 3762; Symbol: ER_FUNCTIONAL_INDEX_ON_FIELD; SQLSTATE: HY000

  Message: Functional index on a column is not supported. Consider using a regular index instead.

  ER_FUNCTIONAL_INDEX_ON_FIELD was added in 8.0.13.

- Error number: 3763; Symbol: ER_GENERATED_COLUMN_NAMED_FUNCTION_IS_NOT_ALLOWED; SQLSTATE: HY000

  Message: Expression of generated column '%s' contains a disallowed function: %s.

  ER_GENERATED_COLUMN_NAMED_FUNCTION_IS_NOT_ALLOWED was added in 8.0.13.

- Error number: 3764; Symbol: ER_GENERATED_COLUMN_ROW_VALUE; SQLSTATE: HY000

  Message: Expression of generated column '%s' cannot refer to a row value.

  ER_GENERATED_COLUMN_ROW_VALUE was added in 8.0.13.

- Error number: 3765; Symbol: ER_GENERATED_COLUMN_VARIABLES; SQLSTATE: HY000

  Message: Expression of generated column '%s' cannot refer user or system variables.

  ER_GENERATED_COLUMN_VARIABLES was added in 8.0.13.

- Error number: 3766; Symbol: ER_DEPENDENT_BY_DEFAULT_GENERATED_VALUE; SQLSTATE: HY000

Message: Column '%s' of table '%s' has a default value expression dependency and cannot be dropped or renamed.

ER_DEPENDENT_BY_DEFAULT_GENERATED_VALUE was added in 8.0.13.

- Error number: 3767; Symbol: ER_DEFAULT_VAL_GENERATED_NON_PRIOR; SQLSTATE: HY000

  Message: Default value expression of column '%s' cannot refer to a column defined after it if that column is a generated column or has an expression as default value.

  ER_DEFAULT_VAL_GENERATED_NON_PRIOR was added in 8.0.13.

- Error number: 3768; Symbol: ER_DEFAULT_VAL_GENERATED_REF_AUTO_INC; SQLSTATE: HY000

  Message: Default value expression of column '%s' cannot refer to an auto-increment column.

  ER_DEFAULT_VAL_GENERATED_REF_AUTO_INC was added in 8.0.13.

- Error number: 3769; Symbol: ER_DEFAULT_VAL_GENERATED_FUNCTION_IS_NOT_ALLOWED; SQLSTATE: HY000

  Message: Default value expression of column '%s' contains a disallowed function.

  ER_DEFAULT_VAL_GENERATED_FUNCTION_IS_NOT_ALLOWED was added in 8.0.13.

- Error number: 3770; Symbol: ER_DEFAULT_VAL_GENERATED_NAMED_FUNCTION_IS_NOT_ALLOWED; SQLSTATE: HY000

  Message: Default value expression of column '%s' contains a disallowed function: %s.

  ER_DEFAULT_VAL_GENERATED_NAMED_FUNCTION_IS_NOT_ALLOWED was added in 8.0.13.

- Error number: 3771; Symbol: ER_DEFAULT_VAL_GENERATED_ROW_VALUE; SQLSTATE: HY000

  Message: Default value expression of column '%s' cannot refer to a row value.

  ER_DEFAULT_VAL_GENERATED_ROW_VALUE was added in 8.0.13.

- Error number: 3772; Symbol: ER_DEFAULT_VAL_GENERATED_VARIABLES; SQLSTATE: HY000

  Message: Default value expression of column '%s' cannot refer user or system variables.

  ER_DEFAULT_VAL_GENERATED_VARIABLES was added in 8.0.13.

- Error number: 3773; Symbol: ER_DEFAULT_AS_VAL_GENERATED; SQLSTATE: HY000

  Message: DEFAULT function cannot be used with default value expressions

  ER_DEFAULT_AS_VAL_GENERATED was added in 8.0.13.

- Error number: 3774; Symbol: ER_UNSUPPORTED_ACTION_ON_DEFAULT_VAL_GENERATED; SQLSTATE: HY000

  Message: '%s' is not supported for default value expressions.

  ER_UNSUPPORTED_ACTION_ON_DEFAULT_VAL_GENERATED was added in 8.0.13.

- Error number: 3775; Symbol: ER_GTID_UNSAFE_ALTER_ADD_COL_WITH_DEFAULT_EXPRESSION; SQLSTATE: HY000

Message: Statement violates GTID consistency: ALTER TABLE ... ADD COLUMN .. with expression as DEFAULT.

`ER_GTID_UNSAFE_ALTER_ADD_COL_WITH_DEFAULT_EXPRESSION` was added in 8.0.13.

- Error number: `3776`; Symbol: `ER_FK_CANNOT_CHANGE_ENGINE`; SQLSTATE: `HY000`

  Message: Cannot change table's storage engine because the table participates in a foreign key constraint.

  `ER_FK_CANNOT_CHANGE_ENGINE` was added in 8.0.13.

- Error number: `3777`; Symbol: `ER_WARN_DEPRECATED_USER_SET_EXPR`; SQLSTATE: `HY000`

  Message: Setting user variables within expressions is deprecated and will be removed in a future release. Consider alternatives: 'SET variable=expression, ...', or 'SELECT expression(s) INTO variables(s)'.

  `ER_WARN_DEPRECATED_USER_SET_EXPR` was added in 8.0.13.

- Error number: `3778`; Symbol: `ER_WARN_DEPRECATED_UTF8MB3_COLLATION`; SQLSTATE: `HY000`

  Message: '%s' is a collation of the deprecated character set UTF8MB3. Please consider using UTF8MB4 with an appropriate collation instead.

  `ER_WARN_DEPRECATED_UTF8MB3_COLLATION` was added in 8.0.13.

- Error number: `3779`; Symbol: `ER_WARN_DEPRECATED_NESTED_COMMENT_SYNTAX`; SQLSTATE: `HY000`

  Message: Nested comment syntax is deprecated and will be removed in a future release.

  `ER_WARN_DEPRECATED_NESTED_COMMENT_SYNTAX` was added in 8.0.13.

- Error number: `3780`; Symbol: `ER_FK_INCOMPATIBLE_COLUMNS`; SQLSTATE: `HY000`

  Message: Referencing column '%s' and referenced column '%s' in foreign key constraint '%s' are incompatible.

  `ER_FK_INCOMPATIBLE_COLUMNS` was added in 8.0.14.

- Error number: `3781`; Symbol: `ER_GR_HOLD_WAIT_TIMEOUT`; SQLSTATE: `HY000`

  Message: Timeout exceeded for held statement while new Group Replication primary member is applying backlog.

  `ER_GR_HOLD_WAIT_TIMEOUT` was added in 8.0.14.

- Error number: `3782`; Symbol: `ER_GR_HOLD_KILLED`; SQLSTATE: `HY000`

  Message: Held statement aborted because Group Replication plugin got shut down or thread was killed while new primary member was applying backlog.

  `ER_GR_HOLD_KILLED` was added in 8.0.14.

- Error number: `3783`; Symbol: `ER_GR_HOLD_MEMBER_STATUS_ERROR`; SQLSTATE: `HY000`

Message: Held statement was aborted due to member being in error state, while backlog is being applied during Group Replication primary election.

ER_GR_HOLD_MEMBER_STATUS_ERROR was added in 8.0.14.

- Error number: 3784; Symbol: ER_RPL_ENCRYPTION_FAILED_TO_FETCH_KEY; SQLSTATE: HY000

  Message: Failed to fetch key from keyring, please check if keyring plugin is loaded.

  ER_RPL_ENCRYPTION_FAILED_TO_FETCH_KEY was added in 8.0.14.

- Error number: 3785; Symbol: ER_RPL_ENCRYPTION_KEY_NOT_FOUND; SQLSTATE: HY000

  Message: Can't find key from keyring, please check in the server log if a keyring plugin is loaded and initialized successfully.

  ER_RPL_ENCRYPTION_KEY_NOT_FOUND was added in 8.0.14.

- Error number: 3786; Symbol: ER_RPL_ENCRYPTION_KEYRING_INVALID_KEY; SQLSTATE: HY000

  Message: Fetched an invalid key from keyring.

  ER_RPL_ENCRYPTION_KEYRING_INVALID_KEY was added in 8.0.14.

- Error number: 3787; Symbol: ER_RPL_ENCRYPTION_HEADER_ERROR; SQLSTATE: HY000

  Message: Error reading a replication log encryption header: %s.

  ER_RPL_ENCRYPTION_HEADER_ERROR was added in 8.0.14.

- Error number: 3788; Symbol: ER_RPL_ENCRYPTION_FAILED_TO_ROTATE_LOGS; SQLSTATE: HY000

  Message: Failed to rotate some logs after changing binlog encryption settings. Please fix the problem and rotate the logs manually.

  ER_RPL_ENCRYPTION_FAILED_TO_ROTATE_LOGS was added in 8.0.14.

- Error number: 3789; Symbol: ER_RPL_ENCRYPTION_KEY_EXISTS_UNEXPECTED; SQLSTATE: HY000

  Message: Key %s exists unexpected.

  ER_RPL_ENCRYPTION_KEY_EXISTS_UNEXPECTED was added in 8.0.14.

- Error number: 3790; Symbol: ER_RPL_ENCRYPTION_FAILED_TO_GENERATE_KEY; SQLSTATE: HY000

  Message: Failed to generate key, please check if keyring plugin is loaded.

  ER_RPL_ENCRYPTION_FAILED_TO_GENERATE_KEY was added in 8.0.14.

- Error number: 3791; Symbol: ER_RPL_ENCRYPTION_FAILED_TO_STORE_KEY; SQLSTATE: HY000

  Message: Failed to store key, please check if keyring plugin is loaded.

  ER_RPL_ENCRYPTION_FAILED_TO_STORE_KEY was added in 8.0.14.

- Error number: 3792; Symbol: ER_RPL_ENCRYPTION_FAILED_TO_REMOVE_KEY; SQLSTATE: HY000

Message: Failed to remove key, please check if keyring plugin is loaded.

ER_RPL_ENCRYPTION_FAILED_TO_REMOVE_KEY was added in 8.0.14.

- Error number: 3793; Symbol: ER_RPL_ENCRYPTION_UNABLE_TO_CHANGE_OPTION; SQLSTATE: HY000

  Message: Failed to change binlog_encryption value. %s.

  ER_RPL_ENCRYPTION_UNABLE_TO_CHANGE_OPTION was added in 8.0.14.

- Error number: 3794; Symbol: ER_RPL_ENCRYPTION_MASTER_KEY_RECOVERY_FAILED; SQLSTATE: HY000

  Message: Unable to recover binlog encryption master key, please check if keyring plugin is loaded.

  ER_RPL_ENCRYPTION_MASTER_KEY_RECOVERY_FAILED was added in 8.0.14.

- Error number: 3795; Symbol: ER_SLOW_LOG_MODE_IGNORED_WHEN_NOT_LOGGING_TO_FILE; SQLSTATE: HY000

  Message: slow query log file format changed as requested, but setting will have no effect when not actually logging to a file.

  ER_SLOW_LOG_MODE_IGNORED_WHEN_NOT_LOGGING_TO_FILE was added in 8.0.14.

- Error number: 3796; Symbol: ER_GRP_TRX_CONSISTENCY_NOT_ALLOWED; SQLSTATE: HY000

  Message: The option group_replication_consistency cannot be used on the current member state.

  ER_GRP_TRX_CONSISTENCY_NOT_ALLOWED was added in 8.0.14.

- Error number: 3797; Symbol: ER_GRP_TRX_CONSISTENCY_BEFORE; SQLSTATE: HY000

  Message: Error while waiting for group transactions commit on group_replication_consistency= 'BEFORE'.

  ER_GRP_TRX_CONSISTENCY_BEFORE was added in 8.0.14.

- Error number: 3798; Symbol: ER_GRP_TRX_CONSISTENCY_AFTER_ON_TRX_BEGIN; SQLSTATE: HY000

  Message: Error while waiting for transactions with group_replication_consistency= 'AFTER' to commit.

  ER_GRP_TRX_CONSISTENCY_AFTER_ON_TRX_BEGIN was added in 8.0.14.

- Error number: 3799; Symbol: ER_GRP_TRX_CONSISTENCY_BEGIN_NOT_ALLOWED; SQLSTATE: HY000

  Message: The Group Replication plugin is stopping, therefore new transactions are not allowed to start.

  ER_GRP_TRX_CONSISTENCY_BEGIN_NOT_ALLOWED was added in 8.0.14.

- Error number: 3800; Symbol: ER_FUNCTIONAL_INDEX_ROW_VALUE_IS_NOT_ALLOWED; SQLSTATE: HY000

  Message: Expression of functional index '%s' cannot refer to a row value.

ER_FUNCTIONAL_INDEX_ROW_VALUE_IS_NOT_ALLOWED was added in 8.0.14.

- Error number: 3801; Symbol: ER_RPL_ENCRYPTION_FAILED_TO_ENCRYPT; SQLSTATE: HY000

  Message: Failed to encrypt content to write into binlog file: %s.

  ER_RPL_ENCRYPTION_FAILED_TO_ENCRYPT was added in 8.0.14.

- Error number: 3802; Symbol: ER_PAGE_TRACKING_NOT_STARTED; SQLSTATE: HY000

  Message: Page Tracking is not started yet.

  ER_PAGE_TRACKING_NOT_STARTED was added in 8.0.16.

- Error number: 3803; Symbol: ER_PAGE_TRACKING_RANGE_NOT_TRACKED; SQLSTATE: HY000

  Message: Tracking was not enabled for the LSN range specified

  ER_PAGE_TRACKING_RANGE_NOT_TRACKED was added in 8.0.16.

- Error number: 3804; Symbol: ER_PAGE_TRACKING_CANNOT_PURGE; SQLSTATE: HY000

  Message: Cannot purge data when concurrent clone is in progress. Try later.

  ER_PAGE_TRACKING_CANNOT_PURGE was added in 8.0.16.

- Error number: 3805; Symbol: ER_RPL_ENCRYPTION_CANNOT_ROTATE_BINLOG_MASTER_KEY; SQLSTATE: HY000

  Message: Cannot rotate binary log master key when 'binlog-encryption' is off.

  ER_RPL_ENCRYPTION_CANNOT_ROTATE_BINLOG_MASTER_KEY was added in 8.0.16.

- Error number: 3806; Symbol: ER_BINLOG_MASTER_KEY_RECOVERY_OUT_OF_COMBINATION; SQLSTATE: HY000

  Message: Unable to recover binary log master key, the combination of new_master_key_seqno=%u, master_key_seqno=%u and old_master_key_seqno=%u are wrong.

  ER_BINLOG_MASTER_KEY_RECOVERY_OUT_OF_COMBINATION was added in 8.0.16.

- Error number: 3807; Symbol: ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_OPERATE_KEY; SQLSTATE: HY000

  Message: Failed to operate binary log master key on keyring, please check if keyring plugin is loaded. The statement had no effect: the old binary log master key is still in use, the keyring, binary and relay log files are unchanged, and the server could not start using a new binary log master key for encrypting new binary and relay log files.

  ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_OPERATE_KEY was added in 8.0.16.

- Error number: 3808; Symbol: ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_ROTATE_LOGS; SQLSTATE: HY000

  Message: Failed to rotate one or more binary or relay log files. A new binary log master key was generated and will be used to encrypt new binary and relay log files. There may still exist binary or relay log files using the previous binary log master key.

ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_ROTATE_LOGS was added in 8.0.16.

- Error number: 3809; Symbol: ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_REENCRYPT_LOG; SQLSTATE: HY000

  Message: %s. A new binary log master key was generated and will be used to encrypt new binary and relay log files. There may still exist binary or relay log files using the previous binary log master key.

  ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_REENCRYPT_LOG was added in 8.0.16.

- Error number: 3810; Symbol: ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_UNUSED_KEYS; SQLSTATE: HY000

  Message: Failed to remove unused binary log encryption keys from the keyring, please check if keyring plugin is loaded. The unused binary log encryption keys may still exist in the keyring, and they will be removed upon server restart or next 'ALTER INSTANCE ROTATE BINLOG MASTER KEY' execution.

  ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_UNUSED_KEYS was added in 8.0.16.

- Error number: 3811; Symbol: ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_AUX_KEY; SQLSTATE: HY000

  Message: Failed to remove auxiliary binary log encryption key from keyring, please check if keyring plugin is loaded. The cleanup of the binary log master key rotation process did not finish as expected and the cleanup will take place upon server restart or next 'ALTER INSTANCE ROTATE BINLOG MASTER KEY' execution.

  ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_AUX_KEY was added in 8.0.16.

- Error number: 3812; Symbol: ER_NON_BOOLEAN_EXPR_FOR_CHECK_CONSTRAINT; SQLSTATE: HY000

  Message: An expression of non-boolean type specified to a check constraint '%s'.

  ER_NON_BOOLEAN_EXPR_FOR_CHECK_CONSTRAINT was added in 8.0.16.

- Error number: 3813; Symbol: ER_COLUMN_CHECK_CONSTRAINT_REFERENCES_OTHER_COLUMN; SQLSTATE: HY000

  Message: Column check constraint '%s' references other column.

  ER_COLUMN_CHECK_CONSTRAINT_REFERENCES_OTHER_COLUMN was added in 8.0.16.

- Error number: 3814; Symbol: ER_CHECK_CONSTRAINT_NAMED_FUNCTION_IS_NOT_ALLOWED; SQLSTATE: HY000

  Message: An expression of a check constraint '%s' contains disallowed function: %s.

  ER_CHECK_CONSTRAINT_NAMED_FUNCTION_IS_NOT_ALLOWED was added in 8.0.16.

- Error number: 3815; Symbol: ER_CHECK_CONSTRAINT_FUNCTION_IS_NOT_ALLOWED; SQLSTATE: HY000

  Message: An expression of a check constraint '%s' contains disallowed function.

  ER_CHECK_CONSTRAINT_FUNCTION_IS_NOT_ALLOWED was added in 8.0.16.

- Error number: 3816; Symbol: ER_CHECK_CONSTRAINT_VARIABLES; SQLSTATE: HY000

Message: An expression of a check constraint '%s' cannot refer to a user or system variable.

ER_CHECK_CONSTRAINT_VARIABLES was added in 8.0.16.

- Error number: 3817; Symbol: ER_CHECK_CONSTRAINT_ROW_VALUE; SQLSTATE: HY000

  Message: Check constraint '%s' cannot refer to a row value.

  ER_CHECK_CONSTRAINT_ROW_VALUE was added in 8.0.16.

- Error number: 3818; Symbol: ER_CHECK_CONSTRAINT_REFERS_AUTO_INCREMENT_COLUMN; SQLSTATE: HY000

  Message: Check constraint '%s' cannot refer to an auto-increment column.

  ER_CHECK_CONSTRAINT_REFERS_AUTO_INCREMENT_COLUMN was added in 8.0.16.

- Error number: 3819; Symbol: ER_CHECK_CONSTRAINT_VIOLATED; SQLSTATE: HY000

  Message: Check constraint '%s' is violated.

  ER_CHECK_CONSTRAINT_VIOLATED was added in 8.0.16.

- Error number: 3820; Symbol: ER_CHECK_CONSTRAINT_REFERS_UNKNOWN_COLUMN; SQLSTATE: HY000

  Message: Check constraint '%s' refers to non-existing column '%s'.

  ER_CHECK_CONSTRAINT_REFERS_UNKNOWN_COLUMN was added in 8.0.16.

- Error number: 3821; Symbol: ER_CHECK_CONSTRAINT_NOT_FOUND; SQLSTATE: HY000

  Message: Check constraint '%s' is not found in the table.

  ER_CHECK_CONSTRAINT_NOT_FOUND was added in 8.0.16.

- Error number: 3822; Symbol: ER_CHECK_CONSTRAINT_DUP_NAME; SQLSTATE: HY000

  Message: Duplicate check constraint name '%s'.

  ER_CHECK_CONSTRAINT_DUP_NAME was added in 8.0.16.

- Error number: 3823; Symbol: ER_CHECK_CONSTRAINT_CLAUSE_USING_FK_REFER_ACTION_COLUMN; SQLSTATE: HY000

  Message: Column '%s' cannot be used in a check constraint '%s': needed in a foreign key constraint '%s' referential action.

  ER_CHECK_CONSTRAINT_CLAUSE_USING_FK_REFER_ACTION_COLUMN was added in 8.0.16.

- Error number: 3824; Symbol: WARN_UNENCRYPTED_TABLE_IN_ENCRYPTED_DB; SQLSTATE: HY000

  Message: Creating an unencrypted table in a database with default encryption enabled.

  WARN_UNENCRYPTED_TABLE_IN_ENCRYPTED_DB was added in 8.0.16.

- Error number: 3825; Symbol: ER_INVALID_ENCRYPTION_REQUEST; SQLSTATE: HY000

  Message: Request to create %s table while using an %s tablespace.

ER_INVALID_ENCRYPTION_REQUEST was added in 8.0.16.

- Error number: 3826; Symbol: ER_CANNOT_SET_TABLE_ENCRYPTION; SQLSTATE: HY000

  Message: Table encryption differ from its database default encryption, and user doesn't have enough privilege.

  ER_CANNOT_SET_TABLE_ENCRYPTION was added in 8.0.16.

- Error number: 3827; Symbol: ER_CANNOT_SET_DATABASE_ENCRYPTION; SQLSTATE: HY000

  Message: Database default encryption differ from 'default_table_encryption' setting, and user doesn't have enough privilege.

  ER_CANNOT_SET_DATABASE_ENCRYPTION was added in 8.0.16.

- Error number: 3828; Symbol: ER_CANNOT_SET_TABLESPACE_ENCRYPTION; SQLSTATE: HY000

  Message: Tablespace encryption differ from 'default_table_encryption' setting, and user doesn't have enough privilege.

  ER_CANNOT_SET_TABLESPACE_ENCRYPTION was added in 8.0.16.

- Error number: 3829; Symbol: ER_TABLESPACE_CANNOT_BE_ENCRYPTED; SQLSTATE: HY000

  Message: This tablespace can't be encrypted, because one of table's schema has default encryption OFF and user doesn't have enough privilege.

  ER_TABLESPACE_CANNOT_BE_ENCRYPTED was added in 8.0.16.

- Error number: 3830; Symbol: ER_TABLESPACE_CANNOT_BE_DECRYPTED; SQLSTATE: HY000

  Message: This tablespace can't be decrypted, because one of table's schema has default encryption ON and user doesn't have enough privilege.

  ER_TABLESPACE_CANNOT_BE_DECRYPTED was added in 8.0.16.

- Error number: 3831; Symbol: ER_TABLESPACE_TYPE_UNKNOWN; SQLSTATE: HY000

  Message: Cannot determine the type of the tablespace named '%s'.

  ER_TABLESPACE_TYPE_UNKNOWN was added in 8.0.16.

- Error number: 3832; Symbol: ER_TARGET_TABLESPACE_UNENCRYPTED; SQLSTATE: HY000

  Message: Source tablespace is encrypted but target tablespace is not.

  ER_TARGET_TABLESPACE_UNENCRYPTED was added in 8.0.16.

- Error number: 3833; Symbol: ER_CANNOT_USE_ENCRYPTION_CLAUSE; SQLSTATE: HY000

  Message: ENCRYPTION clause is not valid for %s tablespace.

  ER_CANNOT_USE_ENCRYPTION_CLAUSE was added in 8.0.16.

- Error number: 3834; Symbol: ER_INVALID_MULTIPLE_CLAUSES; SQLSTATE: HY000

  Message: Multiple %s clauses

ER_INVALID_MULTIPLE_CLAUSES was added in 8.0.16.

- Error number: 3835; Symbol: ER_UNSUPPORTED_USE_OF_GRANT_AS; SQLSTATE: HY000

  Message: GRANT ... AS is currently supported only for global privileges.

  ER_UNSUPPORTED_USE_OF_GRANT_AS was added in 8.0.16.

- Error number: 3836; Symbol: ER_UKNOWN_AUTH_ID_OR_ACCESS_DENIED_FOR_GRANT_AS; SQLSTATE: HY000

  Message: Either some of the authorization IDs in the AS clause are invalid or the current user lacks privileges to execute the statement.

  ER_UKNOWN_AUTH_ID_OR_ACCESS_DENIED_FOR_GRANT_AS was added in 8.0.16.

- Error number: 3837; Symbol: ER_DEPENDENT_BY_FUNCTIONAL_INDEX; SQLSTATE: HY000

  Message: Column '%s' has a functional index dependency and cannot be dropped or renamed.

  ER_DEPENDENT_BY_FUNCTIONAL_INDEX was added in 8.0.17.

- Error number: 3838; Symbol: ER_PLUGIN_NOT_EARLY; SQLSTATE: HY000

  Message: Plugin '%s' is not to be used as an "early" plugin. Don't add it to --early-plugin-load, keyring migration etc.

  ER_PLUGIN_NOT_EARLY was added in 8.0.17.

- Error number: 3839; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_START_SUBDIR_PATH; SQLSTATE: HY000

  Message: Redo log archiving start prohibits path name in 'subdir' argument

  ER_INNODB_REDO_LOG_ARCHIVE_START_SUBDIR_PATH was added in 8.0.17.

- Error number: 3840; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_START_TIMEOUT; SQLSTATE: HY000

  Message: Redo log archiving start timed out

  ER_INNODB_REDO_LOG_ARCHIVE_START_TIMEOUT was added in 8.0.17.

- Error number: 3841; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_DIRS_INVALID; SQLSTATE: HY000

  Message: Server variable 'innodb_redo_log_archive_dirs' is NULL or empty

  ER_INNODB_REDO_LOG_ARCHIVE_DIRS_INVALID was added in 8.0.17.

- Error number: 3842; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_LABEL_NOT_FOUND; SQLSTATE: HY000

  Message: Label '%s' not found in server variable 'innodb_redo_log_archive_dirs'

  ER_INNODB_REDO_LOG_ARCHIVE_LABEL_NOT_FOUND was added in 8.0.17.

- Error number: 3843; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_DIR_EMPTY; SQLSTATE: HY000

Message: Directory is empty after label '%s' in server variable 'innodb_redo_log_archive_dirs'

ER_INNODB_REDO_LOG_ARCHIVE_DIR_EMPTY was added in 8.0.17.

- Error number: 3844; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_NO_SUCH_DIR; SQLSTATE: HY000

  Message: Redo log archive directory '%s' does not exist or is not a directory

  ER_INNODB_REDO_LOG_ARCHIVE_NO_SUCH_DIR was added in 8.0.17.

- Error number: 3845; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_DIR_CLASH; SQLSTATE: HY000

  Message: Redo log archive directory '%s' is in, under, or over server directory '%s' - '%s'

  ER_INNODB_REDO_LOG_ARCHIVE_DIR_CLASH was added in 8.0.17.

- Error number: 3846; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_DIR_PERMISSIONS; SQLSTATE: HY000

  Message: Redo log archive directory '%s' is accessible to all OS users

  ER_INNODB_REDO_LOG_ARCHIVE_DIR_PERMISSIONS was added in 8.0.17.

- Error number: 3847; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_FILE_CREATE; SQLSTATE: HY000

  Message: Cannot create redo log archive file '%s' (OS errno: %d - %s)

  ER_INNODB_REDO_LOG_ARCHIVE_FILE_CREATE was added in 8.0.17.

- Error number: 3848; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_ACTIVE; SQLSTATE: HY000

  Message: Redo log archiving has been started on '%s' - Call innodb_redo_log_archive_stop() first

  ER_INNODB_REDO_LOG_ARCHIVE_ACTIVE was added in 8.0.17.

- Error number: 3849; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_INACTIVE; SQLSTATE: HY000

  Message: Redo log archiving is not active

  ER_INNODB_REDO_LOG_ARCHIVE_INACTIVE was added in 8.0.17.

- Error number: 3850; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_FAILED; SQLSTATE: HY000

  Message: Redo log archiving failed: %s

  ER_INNODB_REDO_LOG_ARCHIVE_FAILED was added in 8.0.17.

- Error number: 3851; Symbol: ER_INNODB_REDO_LOG_ARCHIVE_SESSION; SQLSTATE: HY000

  Message: Redo log archiving has not been started by this session

  ER_INNODB_REDO_LOG_ARCHIVE_SESSION was added in 8.0.17.

- Error number: 3852; Symbol: ER_STD_REGEX_ERROR; SQLSTATE: HY000

  Message: Regex error: %s in function %s.

  ER_STD_REGEX_ERROR was added in 8.0.17.

- Error number: 3853; Symbol: ER_INVALID_JSON_TYPE; SQLSTATE: 22032

  Message: Invalid JSON type in argument %u to function %s; an %s is required.

  ER_INVALID_JSON_TYPE was added in 8.0.17.

- Error number: 3854; Symbol: ER_CANNOT_CONVERT_STRING; SQLSTATE: HY000

  Message: Cannot convert string '%s' from %s to %s

  ER_CANNOT_CONVERT_STRING was added in 8.0.17.

- Error number: 3855; Symbol: ER_DEPENDENT_BY_PARTITION_FUNC; SQLSTATE: HY000

  Message: Column '%s' has a partitioning function dependency and cannot be dropped or renamed.

  ER_DEPENDENT_BY_PARTITION_FUNC was added in 8.0.17.

- Error number: 3856; Symbol: ER_WARN_DEPRECATED_FLOAT_AUTO_INCREMENT; SQLSTATE: HY000

  Message: AUTO_INCREMENT support for FLOAT/DOUBLE columns is deprecated and will be removed in a future release. Consider removing AUTO_INCREMENT from column '%s'.

  ER_WARN_DEPRECATED_FLOAT_AUTO_INCREMENT was added in 8.0.17.

- Error number: 3857; Symbol: ER_RPL_CANT_STOP_SLAVE_WHILE_LOCKED_BACKUP; SQLSTATE: HY000

  Message: Cannot stop the slave SQL thread while the instance is locked for backup. Try running `UNLOCK INSTANCE` first.

  ER_RPL_CANT_STOP_SLAVE_WHILE_LOCKED_BACKUP was added in 8.0.17.

- Error number: 3858; Symbol: ER_WARN_DEPRECATED_FLOAT_DIGITS; SQLSTATE: HY000

  Message: Specifying number of digits for floating point data types is deprecated and will be removed in a future release.

  ER_WARN_DEPRECATED_FLOAT_DIGITS was added in 8.0.17.

- Error number: 3859; Symbol: ER_WARN_DEPRECATED_FLOAT_UNSIGNED; SQLSTATE: HY000

  Message: UNSIGNED for decimal and floating point data types is deprecated and support for it will be removed in a future release.

  ER_WARN_DEPRECATED_FLOAT_UNSIGNED was added in 8.0.17.

- Error number: 3860; Symbol: ER_WARN_DEPRECATED_INTEGER_DISPLAY_WIDTH; SQLSTATE: HY000

  Message: Integer display width is deprecated and will be removed in a future release.

  ER_WARN_DEPRECATED_INTEGER_DISPLAY_WIDTH was added in 8.0.17.

- Error number: 3861; Symbol: ER_WARN_DEPRECATED_ZEROFILL; SQLSTATE: HY000

  Message: The ZEROFILL attribute is deprecated and will be removed in a future release. Use the LPAD function to zero-pad numbers, or store the formatted numbers in a CHAR column.

ER_WARN_DEPRECATED_ZEROFILL was added in 8.0.17.

- Error number: 3862; Symbol: ER_CLONE_DONOR; SQLSTATE: HY000

  Message: Clone Donor Error: %s.

  ER_CLONE_DONOR was added in 8.0.17.

- Error number: 3863; Symbol: ER_CLONE_PROTOCOL; SQLSTATE: HY000

  Message: Clone received unexpected response from Donor : %s.

  ER_CLONE_PROTOCOL was added in 8.0.17.

- Error number: 3864; Symbol: ER_CLONE_DONOR_VERSION; SQLSTATE: HY000

  Message: Clone Donor MySQL version: %s is different from Recipient MySQL version %s.

  ER_CLONE_DONOR_VERSION was added in 8.0.17.

- Error number: 3865; Symbol: ER_CLONE_OS; SQLSTATE: HY000

  Message: Clone Donor OS: %s is different from Recipient OS: %s.

  ER_CLONE_OS was added in 8.0.17.

- Error number: 3866; Symbol: ER_CLONE_PLATFORM; SQLSTATE: HY000

  Message: Clone Donor platform: %s is different from Recipient platform: %s.

  ER_CLONE_PLATFORM was added in 8.0.17.

- Error number: 3867; Symbol: ER_CLONE_CHARSET; SQLSTATE: HY000

  Message: Clone Donor collation: %s is unavailable in Recipient.

  ER_CLONE_CHARSET was added in 8.0.17.

- Error number: 3868; Symbol: ER_CLONE_CONFIG; SQLSTATE: HY000

  Message: Clone Configuration %s: Donor value: %s is different from Recipient value: %s.

  ER_CLONE_CONFIG was added in 8.0.17.

- Error number: 3869; Symbol: ER_CLONE_SYS_CONFIG; SQLSTATE: HY000

  Message: Clone system configuration: %s

  ER_CLONE_SYS_CONFIG was added in 8.0.17.

- Error number: 3870; Symbol: ER_CLONE_PLUGIN_MATCH; SQLSTATE: HY000

  Message: Clone Donor plugin %s is not active in Recipient.

  ER_CLONE_PLUGIN_MATCH was added in 8.0.17.

- Error number: 3871; Symbol: ER_CLONE_LOOPBACK; SQLSTATE: HY000

  Message: Clone cannot use loop back connection while cloning into current data directory.

ER_CLONE_LOOPBACK was added in 8.0.17.

- Error number: 3872; Symbol: ER_CLONE_ENCRYPTION; SQLSTATE: HY000

  Message: Clone needs SSL connection for encrypted table.

  ER_CLONE_ENCRYPTION was added in 8.0.17.

- Error number: 3873; Symbol: ER_CLONE_DISK_SPACE; SQLSTATE: HY000

  Message: Clone estimated database size is %s. Available space %s is not enough.

  ER_CLONE_DISK_SPACE was added in 8.0.17.

- Error number: 3874; Symbol: ER_CLONE_IN_PROGRESS; SQLSTATE: HY000

  Message: Concurrent clone in progress. Please try after clone is complete.

  ER_CLONE_IN_PROGRESS was added in 8.0.17.

- Error number: 3875; Symbol: ER_CLONE_DISALLOWED; SQLSTATE: HY000

  Message: The clone operation cannot be executed when %s.

  ER_CLONE_DISALLOWED was added in 8.0.17.

- Error number: 3876; Symbol: ER_CANNOT_GRANT_ROLES_TO_ANONYMOUS_USER; SQLSTATE: HY000

  Message: Cannot grant roles to an anonymous user.

  ER_CANNOT_GRANT_ROLES_TO_ANONYMOUS_USER was added in 8.0.16.

- Error number: 3877; Symbol: ER_SECONDARY_ENGINE_PLUGIN; SQLSTATE: HY000

  Message: %s

  ER_SECONDARY_ENGINE_PLUGIN was added in 8.0.14.

- Error number: 3878; Symbol: ER_SECOND_PASSWORD_CANNOT_BE_EMPTY; SQLSTATE: HY000

  Message: Empty password can not be retained as second password for user '%s'@'%s'.

  ER_SECOND_PASSWORD_CANNOT_BE_EMPTY was added in 8.0.14.

- Error number: 3879; Symbol: ER_DB_ACCESS_DENIED; SQLSTATE: HY000

  Message: Access denied for AuthId `%s`@`%s` to database '%s'.

  ER_DB_ACCESS_DENIED was added in 8.0.16.

- Error number: 3880; Symbol: ER_DA_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES; SQLSTATE: HY000

  Message: Cannot set mandatory_roles: AuthId `%s`@`%s` has '%s' privilege.

  ER_DA_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES was added in 8.0.17.

- Error number: 3881; Symbol: ER_DA_RPL_GTID_TABLE_CANNOT_OPEN; SQLSTATE: HY000

Message: Gtid table is not ready to be used. Table '%s.%s' cannot be opened.

ER_DA_RPL_GTID_TABLE_CANNOT_OPEN was added in 8.0.17.

- Error number: 3882; Symbol: ER_GEOMETRY_IN_UNKNOWN_LENGTH_UNIT; SQLSTATE: SU001

  Message: The geometry passed to function %s is in SRID 0, which doesn't specify a length unit. Can't convert to '%s'.

  ER_GEOMETRY_IN_UNKNOWN_LENGTH_UNIT was added in 8.0.14.

- Error number: 3883; Symbol: ER_DA_PLUGIN_INSTALL_ERROR; SQLSTATE: HY000

  Message: Error installing plugin '%s': %s

  ER_DA_PLUGIN_INSTALL_ERROR was added in 8.0.17.

- Error number: 3884; Symbol: ER_NO_SESSION_TEMP; SQLSTATE: HY000

  Message: Storage engine could not allocate temporary tablespace for this session.

  ER_NO_SESSION_TEMP was added in 8.0.13.

- Error number: 3885; Symbol: ER_DA_UNKNOWN_ERROR_NUMBER; SQLSTATE: HY000

  Message: Got unknown error: %d

  ER_DA_UNKNOWN_ERROR_NUMBER was added in 8.0.17.

- Error number: 3886; Symbol: ER_COLUMN_CHANGE_SIZE; SQLSTATE: HY000

  Message: Could not change column '%s' of table '%s'. The resulting size of index '%s' would exceed the max key length of %d bytes.

  ER_COLUMN_CHANGE_SIZE was added in 8.0.14.

- Error number: 3887; Symbol: ER_REGEXP_INVALID_CAPTURE_GROUP_NAME; SQLSTATE: HY000

  Message: A capture group has an invalid name.

  ER_REGEXP_INVALID_CAPTURE_GROUP_NAME was added in 8.0.11.

- Error number: 3888; Symbol: ER_DA_SSL_LIBRARY_ERROR; SQLSTATE: HY000

  Message: Failed to set up SSL because of the following SSL library error: %s

  ER_DA_SSL_LIBRARY_ERROR was added in 8.0.17.

- Error number: 3889; Symbol: ER_SECONDARY_ENGINE; SQLSTATE: HY000

  Message: Secondary engine operation failed. %s.

  ER_SECONDARY_ENGINE was added in 8.0.13.

- Error number: 3890; Symbol: ER_SECONDARY_ENGINE_DDL; SQLSTATE: HY000

  Message: DDLs on a table with a secondary engine defined are not allowed.

  ER_SECONDARY_ENGINE_DDL was added in 8.0.13.

- Error number: 3891; Symbol: ER_INCORRECT_CURRENT_PASSWORD; SQLSTATE: HY000

  Message: Incorrect current password. Specify the correct password which has to be replaced.

  ER_INCORRECT_CURRENT_PASSWORD was added in 8.0.13.

- Error number: 3892; Symbol: ER_MISSING_CURRENT_PASSWORD; SQLSTATE: HY000

  Message: Current password needs to be specified in the REPLACE clause in order to change it.

  ER_MISSING_CURRENT_PASSWORD was added in 8.0.13.

- Error number: 3893; Symbol: ER_CURRENT_PASSWORD_NOT_REQUIRED; SQLSTATE: HY000

  Message: Do not specify the current password while changing it for other users.

  ER_CURRENT_PASSWORD_NOT_REQUIRED was added in 8.0.13.

- Error number: 3894; Symbol: ER_PASSWORD_CANNOT_BE_RETAINED_ON_PLUGIN_CHANGE; SQLSTATE: HY000

  Message: Current password can not be retained for user '%s'@'%s' because authentication plugin is being changed.

  ER_PASSWORD_CANNOT_BE_RETAINED_ON_PLUGIN_CHANGE was added in 8.0.14.

- Error number: 3895; Symbol: ER_CURRENT_PASSWORD_CANNOT_BE_RETAINED; SQLSTATE: HY000

  Message: Current password can not be retained for user '%s'@'%s' because new password is empty.

  ER_CURRENT_PASSWORD_CANNOT_BE_RETAINED was added in 8.0.14.

- Error number: 3896; Symbol: ER_PARTIAL_REVOKES_EXIST; SQLSTATE: HY000

  Message: At least one partial revoke exists on a database. The system variable '@@partial_revokes' must be set to ON.

  ER_PARTIAL_REVOKES_EXIST was added in 8.0.16.

- Error number: 3897; Symbol: ER_CANNOT_GRANT_SYSTEM_PRIV_TO_MANDATORY_ROLE; SQLSTATE: HY000

  Message: AuthId `%s`@`%s` is set as mandatory_roles. Cannot grant the '%s' privilege.

  ER_CANNOT_GRANT_SYSTEM_PRIV_TO_MANDATORY_ROLE was added in 8.0.16.

- Error number: 3898; Symbol: ER_XA_REPLICATION_FILTERS; SQLSTATE: HY000

  Message: The use of replication filters with XA transactions is not supported, and can lead to an undefined state in the replication slave.

  ER_XA_REPLICATION_FILTERS was added in 8.0.12.

- Error number: 3899; Symbol: ER_UNSUPPORTED_SQL_MODE; SQLSTATE: HY000

  Message: sql_mode=0x%08x is not supported.

  ER_UNSUPPORTED_SQL_MODE was added in 8.0.11.

- Error number: 3900; Symbol: ER_REGEXP_INVALID_FLAG; SQLSTATE: HY000

Message: Invalid match mode flag in regular expression.

ER_REGEXP_INVALID_FLAG was added in 8.0.12.

- Error number: 3901; Symbol: ER_PARTIAL_REVOKE_AND_DB_GRANT_BOTH_EXISTS; SQLSTATE: HY000

  Message: '%s' privilege for database '%s' exists both as partial revoke and mysql.db simultaneously. It could mean that the 'mysql' schema is corrupted.

  ER_PARTIAL_REVOKE_AND_DB_GRANT_BOTH_EXISTS was added in 8.0.16.

- Error number: 3902; Symbol: ER_UNIT_NOT_FOUND; SQLSTATE: SU001

  Message: There's no unit of measure named '%s'.

  ER_UNIT_NOT_FOUND was added in 8.0.14.

- Error number: 3903; Symbol: ER_INVALID_JSON_VALUE_FOR_FUNC_INDEX; SQLSTATE: 22018

  Message: Invalid JSON value for CAST for functional index '%s'.

  ER_INVALID_JSON_VALUE_FOR_FUNC_INDEX was added in 8.0.16.

- Error number: 3904; Symbol: ER_JSON_VALUE_OUT_OF_RANGE_FOR_FUNC_INDEX; SQLSTATE: 22003

  Message: Out of range JSON value for CAST for functional index '%s'.

  ER_JSON_VALUE_OUT_OF_RANGE_FOR_FUNC_INDEX was added in 8.0.16.

- Error number: 3905; Symbol: ER_EXCEEDED_MV_KEYS_NUM; SQLSTATE: HY000

  Message: Exceeded max number of values per record for multi-valued index '%s' by %u value(s).

  ER_EXCEEDED_MV_KEYS_NUM was added in 8.0.16.

- Error number: 3906; Symbol: ER_EXCEEDED_MV_KEYS_SPACE; SQLSTATE: HY000

  Message: Exceeded max total length of values per record for multi-valued index '%s' by %u bytes.

  ER_EXCEEDED_MV_KEYS_SPACE was added in 8.0.16.

- Error number: 3907; Symbol: ER_FUNCTIONAL_INDEX_DATA_IS_TOO_LONG; SQLSTATE: 22001

  Message: Data too long for functional index '%s'.

  ER_FUNCTIONAL_INDEX_DATA_IS_TOO_LONG was added in 8.0.16.

- Error number: 3908; Symbol: ER_WRONG_MVI_VALUE; SQLSTATE: HY000

  Message: Cannot store an array or an object in a scalar key part of the index '%s'.

  ER_WRONG_MVI_VALUE was added in 8.0.16.

- Error number: 3909; Symbol: ER_WARN_FUNC_INDEX_NOT_APPLICABLE; SQLSTATE: HY000

  Message: Cannot use functional index '%s' due to type or collation conversion.

ER_WARN_FUNC_INDEX_NOT_APPLICABLE was added in 8.0.16.

- Error number: 3910; Symbol: ER_GRP_RPL_UDF_ERROR; SQLSTATE: HY000

  Message: The function '%s' failed. %s

  ER_GRP_RPL_UDF_ERROR was added in 8.0.13.

- Error number: 3911; Symbol: ER_UPDATE_GTID_PURGED_WITH_GR; SQLSTATE: HY000

  Message: Cannot update GTID_PURGED with the Group Replication plugin running

  ER_UPDATE_GTID_PURGED_WITH_GR was added in 8.0.12.

- Error number: 3912; Symbol: ER_GROUPING_ON_TIMESTAMP_IN_DST; SQLSTATE: HY000

  Message: Grouping on temporal is non-deterministic for timezones having DST. Please consider switching to UTC for this query.

  ER_GROUPING_ON_TIMESTAMP_IN_DST was added in 8.0.17.

- Error number: 3913; Symbol: ER_TABLE_NAME_CAUSES_TOO_LONG_PATH; SQLSTATE: HY000

  Message: Long database name and identifier for object resulted in a path length too long for table '%s'. Please check the path limit for your OS.

  ER_TABLE_NAME_CAUSES_TOO_LONG_PATH was added in 8.0.4.

- Error number: 3914; Symbol: ER_AUDIT_LOG_INSUFFICIENT_PRIVILEGE; SQLSTATE: HY000

  Message: Request ignored for '%s'@'%s'. Role needed to perform operation: '%s'

  ER_AUDIT_LOG_INSUFFICIENT_PRIVILEGE was added in 8.0.16.

- Error number: 3916; Symbol: ER_DA_GRP_RPL_STARTED_AUTO_REJOIN; SQLSTATE: HY000

  Message: Started auto-rejoin procedure attempt %lu of %lu

  ER_DA_GRP_RPL_STARTED_AUTO_REJOIN was added in 8.0.17.

- Error number: 3917; Symbol: ER_SYSVAR_CHANGE_DURING_QUERY; SQLSTATE: HY000

  Message: A plugin was loaded or unloaded during a query, a system variable table was changed.

  ER_SYSVAR_CHANGE_DURING_QUERY was added in 8.0.18.

- Error number: 3918; Symbol: ER_GLOBSTAT_CHANGE_DURING_QUERY; SQLSTATE: HY000

  Message: A plugin was loaded or unloaded during a query, a global status variable was changed.

  ER_GLOBSTAT_CHANGE_DURING_QUERY was added in 8.0.18.

- Error number: 3919; Symbol: ER_GRP_RPL_MESSAGE_SERVICE_INIT_FAILURE; SQLSTATE: HY000

  Message: The START GROUP_REPLICATION command failed to start its message service.

  ER_GRP_RPL_MESSAGE_SERVICE_INIT_FAILURE was added in 8.0.18.

- Error number: 3920; Symbol: ER_CHANGE_MASTER_WRONG_COMPRESSION_ALGORITHM_CLIENT; SQLSTATE: HY000

  Message: Invalid MASTER_COMPRESSION_ALGORITHMS '%s' for channel '%s'.

  ER_CHANGE_MASTER_WRONG_COMPRESSION_ALGORITHM_CLIENT was added in 8.0.18.

- Error number: 3921; Symbol: ER_CHANGE_MASTER_WRONG_COMPRESSION_LEVEL_CLIENT; SQLSTATE: HY000

  Message: Invalid MASTER_ZSTD_COMPRESSION_LEVEL %u for channel '%s'.

  ER_CHANGE_MASTER_WRONG_COMPRESSION_LEVEL_CLIENT was added in 8.0.18.

- Error number: 3922; Symbol: ER_WRONG_COMPRESSION_ALGORITHM_CLIENT; SQLSTATE: HY000

  Message: Invalid compression algorithm '%s'.

  ER_WRONG_COMPRESSION_ALGORITHM_CLIENT was added in 8.0.18.

- Error number: 3923; Symbol: ER_WRONG_COMPRESSION_LEVEL_CLIENT; SQLSTATE: HY000

  Message: Invalid zstd compression level for algorithm '%s'.

  ER_WRONG_COMPRESSION_LEVEL_CLIENT was added in 8.0.18.

- Error number: 3924; Symbol: ER_CHANGE_MASTER_WRONG_COMPRESSION_ALGORITHM_LIST_CLIENT; SQLSTATE: HY000

  Message: Specified compression algorithm list '%s' exceeds total count of 3 for channel '%s'.

  ER_CHANGE_MASTER_WRONG_COMPRESSION_ALGORITHM_LIST_CLIENT was added in 8.0.18.

- Error number: 3925; Symbol: ER_CLIENT_PRIVILEGE_CHECKS_USER_CANNOT_BE_ANONYMOUS; SQLSTATE: HY000

  Message: PRIVILEGE_CHECKS_USER for replication channel '%s' was set to ``@`%s`, but anonymous users are disallowed for PRIVILEGE_CHECKS_USER.

  ER_CLIENT_PRIVILEGE_CHECKS_USER_CANNOT_BE_ANONYMOUS was added in 8.0.18.

- Error number: 3926; Symbol: ER_CLIENT_PRIVILEGE_CHECKS_USER_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: PRIVILEGE_CHECKS_USER for replication channel '%s' was set to `%s`@`%s`, but this is not an existing user.

  ER_CLIENT_PRIVILEGE_CHECKS_USER_DOES_NOT_EXIST was added in 8.0.18.

- Error number: 3927; Symbol: ER_CLIENT_PRIVILEGE_CHECKS_USER_CORRUPT; SQLSTATE: HY000

  Message: Invalid, corrupted PRIVILEGE_CHECKS_USER was found in the replication configuration repository for channel '%s'. Use CHANGE MASTER TO PRIVILEGE_CHECKS_USER to correct the configuration.

  ER_CLIENT_PRIVILEGE_CHECKS_USER_CORRUPT was added in 8.0.18.

- Error number: 3928; Symbol: ER_CLIENT_PRIVILEGE_CHECKS_USER_NEEDS_RPL_APPLIER_PRIV; SQLSTATE: HY000

  Message: PRIVILEGE_CHECKS_USER for replication channel '%s' was set to `%s`@`%s`, but this user does not have REPLICATION_APPLIER privilege.

  ER_CLIENT_PRIVILEGE_CHECKS_USER_NEEDS_RPL_APPLIER_PRIV was added in 8.0.18.

- Error number: 3929; Symbol: ER_WARN_DA_PRIVILEGE_NOT_REGISTERED; SQLSTATE: HY000

  Message: Dynamic privilege '%s' is not registered with the server.

  ER_WARN_DA_PRIVILEGE_NOT_REGISTERED was added in 8.0.18.

- Error number: 3930; Symbol: ER_CLIENT_KEYRING_UDF_KEY_INVALID; SQLSTATE: HY000

  Message: Function '%s' failed because key is invalid.

  ER_CLIENT_KEYRING_UDF_KEY_INVALID was added in 8.0.19.

- Error number: 3931; Symbol: ER_CLIENT_KEYRING_UDF_KEY_TYPE_INVALID; SQLSTATE: HY000

  Message: Function '%s' failed because key type is invalid.

  ER_CLIENT_KEYRING_UDF_KEY_TYPE_INVALID was added in 8.0.19.

- Error number: 3932; Symbol: ER_CLIENT_KEYRING_UDF_KEY_TOO_LONG; SQLSTATE: HY000

  Message: Function '%s' failed because key length is too long.

  ER_CLIENT_KEYRING_UDF_KEY_TOO_LONG was added in 8.0.19.

- Error number: 3933; Symbol: ER_CLIENT_KEYRING_UDF_KEY_TYPE_TOO_LONG; SQLSTATE: HY000

  Message: Function '%s' failed because key type is too long.

  ER_CLIENT_KEYRING_UDF_KEY_TYPE_TOO_LONG was added in 8.0.19.

- Error number: 3934; Symbol: ER_JSON_SCHEMA_VALIDATION_ERROR_WITH_DETAILED_REPORT; SQLSTATE: HY000

  Message: %s.

  ER_JSON_SCHEMA_VALIDATION_ERROR_WITH_DETAILED_REPORT was added in 8.0.19.

- Error number: 3935; Symbol: ER_DA_UDF_INVALID_CHARSET_SPECIFIED; SQLSTATE: HY000

  Message: Invalid character set '%s' was specified. It must be either character set name or collation name as supported by server.

  ER_DA_UDF_INVALID_CHARSET_SPECIFIED was added in 8.0.19.

- Error number: 3936; Symbol: ER_DA_UDF_INVALID_CHARSET; SQLSTATE: HY000

  Message: Invalid character set '%s' was specified. It must be a character set name as supported by server.

  ER_DA_UDF_INVALID_CHARSET was added in 8.0.19.

- Error number: 3937; Symbol: ER_DA_UDF_INVALID_COLLATION; SQLSTATE: HY000

  Message: Invalid collation '%s' was specified. It must be a collation name as supported by server.

  ER_DA_UDF_INVALID_COLLATION was added in 8.0.19.

- Error number: 3938; Symbol: ER_DA_UDF_INVALID_EXTENSION_ARGUMENT_TYPE; SQLSTATE: HY000

  Message: Invalid extension argument type '%s' was specified. Refer the MySQL manual for the valid UDF extension arguments type.

  ER_DA_UDF_INVALID_EXTENSION_ARGUMENT_TYPE was added in 8.0.19.

- Error number: MY-010000; Symbol: ER_PARSER_TRACE; SQLSTATE: XX999

  Message: Parser saw: %s

  ER_PARSER_TRACE was added in 8.0.2.

- Error number: MY-010001; Symbol: ER_BOOTSTRAP_CANT_THREAD; SQLSTATE: HY000

  Message: Can't create thread to handle bootstrap (errno: %d)

  ER_BOOTSTRAP_CANT_THREAD was added in 8.0.2.

- Error number: MY-010002; Symbol: ER_TRIGGER_INVALID_VALUE; SQLSTATE: HY000

  Message: Trigger for table '%s'.'%s': invalid %s value (%s).

  ER_TRIGGER_INVALID_VALUE was added in 8.0.2.

- Error number: MY-010003; Symbol: ER_OPT_WRONG_TREE; SQLSTATE: HY000

  Message: Wrong tree: %s

  ER_OPT_WRONG_TREE was added in 8.0.2.

- Error number: MY-010004; Symbol: ER_DD_FAILSAFE; SQLSTATE: HY000

  Message: Error: Invalid %s

  ER_DD_FAILSAFE was added in 8.0.2.

- Error number: MY-010005; Symbol: ER_DD_NO_WRITES_NO_REPOPULATION; SQLSTATE: HY000

  Message: Skip re-populating collations and character sets tables in %s%sread-only mode.

  ER_DD_NO_WRITES_NO_REPOPULATION was added in 8.0.2.

- Error number: MY-010006; Symbol: ER_DD_VERSION_FOUND; SQLSTATE: HY000

  Message: Using data dictionary with version '%d'.

  ER_DD_VERSION_FOUND was added in 8.0.2.

- Error number: MY-010007; Symbol: ER_DD_VERSION_INSTALLED; SQLSTATE: HY000

  Message: Installed data dictionary with version %d

ER_DD_VERSION_INSTALLED was added in 8.0.2.

- Error number: MY-010008; Symbol: ER_DD_VERSION_UNSUPPORTED; SQLSTATE: HY000

  Message: Data Dictionary version '%d' not supported.

  ER_DD_VERSION_UNSUPPORTED was added in 8.0.2.

- Error number: MY-010009; Symbol: ER_LOG_SYSLOG_FACILITY_FAIL; SQLSTATE: HY000

  Message: Failed to set syslog facility to "%s", setting to "%s" (%d) instead.

  ER_LOG_SYSLOG_FACILITY_FAIL was added in 8.0.2, removed after 8.0.12.

- Error number: MY-010010; Symbol: ER_LOG_SYSLOG_CANNOT_OPEN; SQLSTATE: HY000

  Message: Cannot open %s; check privileges, or remove syseventlog from --log-error-services!

  ER_LOG_SYSLOG_CANNOT_OPEN was added in 8.0.2.

- Error number: MY-010011; Symbol: ER_LOG_SLOW_CANNOT_OPEN; SQLSTATE: HY000

  Message: either restart the query logging by using "SET GLOBAL SLOW_QUERY_LOG=ON" or

  ER_LOG_SLOW_CANNOT_OPEN was added in 8.0.2.

- Error number: MY-010012; Symbol: ER_LOG_GENERAL_CANNOT_OPEN; SQLSTATE: HY000

  Message: either restart the query logging by using "SET GLOBAL GENERAL_LOG=ON" or

  ER_LOG_GENERAL_CANNOT_OPEN was added in 8.0.2.

- Error number: MY-010013; Symbol: ER_LOG_CANNOT_WRITE; SQLSTATE: HY000

  Message: Failed to write to %s: %s

  ER_LOG_CANNOT_WRITE was added in 8.0.2.

- Error number: MY-010014; Symbol: ER_RPL_ZOMBIE_ENCOUNTERED; SQLSTATE: HY000

  Message: While initializing dump thread for slave with %s <%s>, found a zombie dump thread with the same %s. Master is killing the zombie dump thread(%u).

  ER_RPL_ZOMBIE_ENCOUNTERED was added in 8.0.2.

- Error number: MY-010015; Symbol: ER_RPL_GTID_TABLE_CANNOT_OPEN; SQLSTATE: HY000

  Message: Gtid table is not ready to be used. Table '%s.%s' cannot be opened.

  ER_RPL_GTID_TABLE_CANNOT_OPEN was added in 8.0.2.

- Error number: MY-010016; Symbol: ER_SYSTEM_SCHEMA_NOT_FOUND; SQLSTATE: HY000

  Message: System schema directory does not exist.

  ER_SYSTEM_SCHEMA_NOT_FOUND was added in 8.0.2.

- Error number: MY-010017; Symbol: ER_DD_INIT_UPGRADE_FAILED; SQLSTATE: HY000

Message: Error in initializing dictionary, upgrade will do a cleanup and exit

ER_DD_INIT_UPGRADE_FAILED was added in 8.0.2.

- Error number: MY-010018; Symbol: ER_VIEW_UNKNOWN_CHARSET_OR_COLLATION; SQLSTATE: HY000

  Message: View '%s'.'%s': unknown charset name and/or collation name (client: '%s'; connection: '%s').

  ER_VIEW_UNKNOWN_CHARSET_OR_COLLATION was added in 8.0.2.

- Error number: MY-010019; Symbol: ER_DD_VIEW_CANT_ALLOC_CHARSET; SQLSTATE: HY000

  Message: Error in allocating memory for character set name for view %s.%s.

  ER_DD_VIEW_CANT_ALLOC_CHARSET was added in 8.0.2.

- Error number: MY-010020; Symbol: ER_DD_INIT_FAILED; SQLSTATE: HY000

  Message: Data Dictionary initialization failed.

  ER_DD_INIT_FAILED was added in 8.0.2.

- Error number: MY-010021; Symbol: ER_DD_UPDATING_PLUGIN_MD_FAILED; SQLSTATE: HY000

  Message: Failed to update plugin metadata in dictionary tables.

  ER_DD_UPDATING_PLUGIN_MD_FAILED was added in 8.0.2.

- Error number: MY-010022; Symbol: ER_DD_POPULATING_TABLES_FAILED; SQLSTATE: HY000

  Message: Failed to Populate DD tables.

  ER_DD_POPULATING_TABLES_FAILED was added in 8.0.2.

- Error number: MY-010023; Symbol: ER_DD_VIEW_CANT_CREATE; SQLSTATE: HY000

  Message: Error in Creating View %s.%s

  ER_DD_VIEW_CANT_CREATE was added in 8.0.2.

- Error number: MY-010024; Symbol: ER_DD_METADATA_NOT_FOUND; SQLSTATE: HY000

  Message: Unable to start server. Cannot find the meta data for data dictionary table '%s'.

  ER_DD_METADATA_NOT_FOUND was added in 8.0.2.

- Error number: MY-010025; Symbol: ER_DD_CACHE_NOT_EMPTY_AT_SHUTDOWN; SQLSTATE: HY000

  Message: Dictionary cache not empty at shutdown.

  ER_DD_CACHE_NOT_EMPTY_AT_SHUTDOWN was added in 8.0.2.

- Error number: MY-010026; Symbol: ER_DD_OBJECT_REMAINS; SQLSTATE: HY000

  Message: Dictionary objects used but not released.

  ER_DD_OBJECT_REMAINS was added in 8.0.2.

- Error number: MY-010027; Symbol: ER_DD_OBJECT_REMAINS_IN_RELEASER; SQLSTATE: HY000

  Message: Dictionary objects left in default releaser.

  ER_DD_OBJECT_REMAINS_IN_RELEASER was added in 8.0.2.

- Error number: MY-010028; Symbol: ER_DD_OBJECT_RELEASER_REMAINS; SQLSTATE: HY000

  Message: Dictionary object auto releaser not deleted

  ER_DD_OBJECT_RELEASER_REMAINS was added in 8.0.2.

- Error number: MY-010029; Symbol: ER_DD_CANT_GET_OBJECT_KEY; SQLSTATE: HY000

  Message: Error: Unable to create primary object key

  ER_DD_CANT_GET_OBJECT_KEY was added in 8.0.2.

- Error number: MY-010030; Symbol: ER_DD_CANT_CREATE_OBJECT_KEY; SQLSTATE: HY000

  Message: Error: Unable to create object key

  ER_DD_CANT_CREATE_OBJECT_KEY was added in 8.0.2.

- Error number: MY-010031; Symbol: ER_CANT_CREATE_HANDLE_MGR_THREAD; SQLSTATE: HY000

  Message: Can't create handle_manager thread (errno= %d)

  ER_CANT_CREATE_HANDLE_MGR_THREAD was added in 8.0.2.

- Error number: MY-010032; Symbol: ER_RPL_REPO_HAS_GAPS; SQLSTATE: HY000

  Message: It is not possible to change the type of the relay log's repository because there are workers' repositories with gaps. Please, fix the gaps first before doing such change.

  ER_RPL_REPO_HAS_GAPS was added in 8.0.2.

- Error number: MY-010033; Symbol: ER_INVALID_VALUE_FOR_ENFORCE_GTID_CONSISTENCY; SQLSTATE: HY000

  Message: option 'enforce-gtid-consistency': value '%s' was not recognized. Setting enforce-gtid-consistency to OFF.

  ER_INVALID_VALUE_FOR_ENFORCE_GTID_CONSISTENCY was added in 8.0.2.

- Error number: MY-010034; Symbol: ER_CHANGED_ENFORCE_GTID_CONSISTENCY; SQLSTATE: HY000

  Message: Changed ENFORCE_GTID_CONSISTENCY from %s to %s.

  ER_CHANGED_ENFORCE_GTID_CONSISTENCY was added in 8.0.2.

- Error number: MY-010035; Symbol: ER_CHANGED_GTID_MODE; SQLSTATE: HY000

  Message: Changed GTID_MODE from %s to %s.

  ER_CHANGED_GTID_MODE was added in 8.0.2.

- Error number: MY-010036; Symbol: ER_DISABLED_STORAGE_ENGINE_AS_DEFAULT; SQLSTATE: HY000

Message: %s is set to a disabled storage engine %s.

ER_DISABLED_STORAGE_ENGINE_AS_DEFAULT was added in 8.0.2.

- Error number: MY-010037; Symbol: ER_DEBUG_SYNC_HIT; SQLSTATE: HY000

  Message: Debug sync points hit: %s

  ER_DEBUG_SYNC_HIT was added in 8.0.2.

- Error number: MY-010038; Symbol: ER_DEBUG_SYNC_EXECUTED; SQLSTATE: HY000

  Message: Debug sync points executed: %s

  ER_DEBUG_SYNC_EXECUTED was added in 8.0.2.

- Error number: MY-010039; Symbol: ER_DEBUG_SYNC_THREAD_MAX; SQLSTATE: HY000

  Message: Debug sync points max active per thread: %s

  ER_DEBUG_SYNC_THREAD_MAX was added in 8.0.2.

- Error number: MY-010040; Symbol: ER_DEBUG_SYNC_OOM; SQLSTATE: HY000

  Message: Debug Sync Facility disabled due to lack of memory.

  ER_DEBUG_SYNC_OOM was added in 8.0.2.

- Error number: MY-010041; Symbol: ER_CANT_INIT_TC_LOG; SQLSTATE: HY000

  Message: Can't init tc log

  ER_CANT_INIT_TC_LOG was added in 8.0.2.

- Error number: MY-010042; Symbol: ER_EVENT_CANT_INIT_QUEUE; SQLSTATE: HY000

  Message: Event Scheduler: Can't initialize the execution queue

  ER_EVENT_CANT_INIT_QUEUE was added in 8.0.2.

- Error number: MY-010043; Symbol: ER_EVENT_PURGING_QUEUE; SQLSTATE: HY000

  Message: Event Scheduler: Purging the queue. %u events

  ER_EVENT_PURGING_QUEUE was added in 8.0.2.

- Error number: MY-010044; Symbol: ER_EVENT_LAST_EXECUTION; SQLSTATE: HY000

  Message: Event Scheduler: Last execution of %s.%s. %s

  ER_EVENT_LAST_EXECUTION was added in 8.0.2.

- Error number: MY-010045; Symbol: ER_EVENT_MESSAGE_STACK; SQLSTATE: HY000

  Message: %*s

  ER_EVENT_MESSAGE_STACK was added in 8.0.2.

- Error number: MY-010046; Symbol: ER_EVENT_EXECUTION_FAILED; SQLSTATE: HY000

Message: Event Scheduler: [%s].[%s.%s] event execution failed.

ER_EVENT_EXECUTION_FAILED was added in 8.0.2.

- Error number: MY-010047; Symbol: ER_CANT_INIT_SCHEDULER_THREAD; SQLSTATE: HY000

  Message: Event Scheduler: Cannot initialize the scheduler thread

  ER_CANT_INIT_SCHEDULER_THREAD was added in 8.0.2.

- Error number: MY-010048; Symbol: ER_SCHEDULER_STOPPED; SQLSTATE: HY000

  Message: Event Scheduler: Stopped

  ER_SCHEDULER_STOPPED was added in 8.0.2.

- Error number: MY-010049; Symbol: ER_CANT_CREATE_SCHEDULER_THREAD; SQLSTATE: HY000

  Message: Event scheduler: Failed to start scheduler, Can not create thread for event scheduler (errno=%d)

  ER_CANT_CREATE_SCHEDULER_THREAD was added in 8.0.2.

- Error number: MY-010050; Symbol: ER_SCHEDULER_WAITING; SQLSTATE: HY000

  Message: Event Scheduler: Waiting for the scheduler thread to reply

  ER_SCHEDULER_WAITING was added in 8.0.2.

- Error number: MY-010051; Symbol: ER_SCHEDULER_STARTED; SQLSTATE: HY000

  Message: Event Scheduler: scheduler thread started with id %u

  ER_SCHEDULER_STARTED was added in 8.0.2.

- Error number: MY-010052; Symbol: ER_SCHEDULER_STOPPING_FAILED_TO_GET_EVENT; SQLSTATE: HY000

  Message: Event Scheduler: Serious error during getting next event to execute. Stopping

  ER_SCHEDULER_STOPPING_FAILED_TO_GET_EVENT was added in 8.0.2.

- Error number: MY-010053; Symbol: ER_SCHEDULER_STOPPING_FAILED_TO_CREATE_WORKER; SQLSTATE: HY000

  Message: Event_scheduler::execute_top: Can not create event worker thread (errno=%d). Stopping event scheduler

  ER_SCHEDULER_STOPPING_FAILED_TO_CREATE_WORKER was added in 8.0.2.

- Error number: MY-010054; Symbol: ER_SCHEDULER_KILLING; SQLSTATE: HY000

  Message: Event Scheduler: Killing the scheduler thread, thread id %u

  ER_SCHEDULER_KILLING was added in 8.0.2.

- Error number: MY-010055; Symbol: ER_UNABLE_TO_RESOLVE_IP; SQLSTATE: HY000

  Message: IP address '%s' could not be resolved: %s

ER_UNABLE_TO_RESOLVE_IP was added in 8.0.2.

- Error number: MY-010056; Symbol: ER_UNABLE_TO_RESOLVE_HOSTNAME; SQLSTATE: HY000

  Message: Host name '%s' could not be resolved: %s

  ER_UNABLE_TO_RESOLVE_HOSTNAME was added in 8.0.2.

- Error number: MY-010057; Symbol: ER_HOSTNAME_RESEMBLES_IPV4; SQLSTATE: HY000

  Message: IP address '%s' has been resolved to the host name '%s', which resembles IPv4-address itself.

  ER_HOSTNAME_RESEMBLES_IPV4 was added in 8.0.2.

- Error number: MY-010058; Symbol: ER_HOSTNAME_DOESNT_RESOLVE_TO; SQLSTATE: HY000

  Message: Hostname '%s' does not resolve to '%s'.

  ER_HOSTNAME_DOESNT_RESOLVE_TO was added in 8.0.2.

- Error number: MY-010059; Symbol: ER_ADDRESSES_FOR_HOSTNAME_HEADER; SQLSTATE: HY000

  Message: Hostname '%s' has the following IP addresses:

  ER_ADDRESSES_FOR_HOSTNAME_HEADER was added in 8.0.2.

- Error number: MY-010060; Symbol: ER_ADDRESSES_FOR_HOSTNAME_LIST_ITEM; SQLSTATE: HY000

  Message: - %s

  ER_ADDRESSES_FOR_HOSTNAME_LIST_ITEM was added in 8.0.2.

- Error number: MY-010061; Symbol: ER_TRG_WITHOUT_DEFINER; SQLSTATE: HY000

  Message: Definer clause is missing in Trigger of Table %s. Rebuild Trigger to fix definer.

  ER_TRG_WITHOUT_DEFINER was added in 8.0.2.

- Error number: MY-010062; Symbol: ER_TRG_NO_CLIENT_CHARSET; SQLSTATE: HY000

  Message: Client character set is missing for trigger of table %s. Using default character set.

  ER_TRG_NO_CLIENT_CHARSET was added in 8.0.2.

- Error number: MY-010063; Symbol: ER_PARSING_VIEW; SQLSTATE: HY000

  Message: Error in parsing view %s.%s

  ER_PARSING_VIEW was added in 8.0.2.

- Error number: MY-010064; Symbol: ER_COMPONENTS_INFRASTRUCTURE_BOOTSTRAP; SQLSTATE: HY000

  Message: Failed to bootstrap components infrastructure.

  ER_COMPONENTS_INFRASTRUCTURE_BOOTSTRAP was added in 8.0.2.

- Error number: MY-010065; Symbol: ER_COMPONENTS_INFRASTRUCTURE_SHUTDOWN; SQLSTATE: HY000

  Message: Failed to shutdown components infrastructure.

  ER_COMPONENTS_INFRASTRUCTURE_SHUTDOWN was added in 8.0.2.

- Error number: MY-010066; Symbol: ER_COMPONENTS_PERSIST_LOADER_BOOTSTRAP; SQLSTATE: HY000

  Message: Failed to bootstrap persistent components loader.

  ER_COMPONENTS_PERSIST_LOADER_BOOTSTRAP was added in 8.0.2.

- Error number: MY-010067; Symbol: ER_DEPART_WITH_GRACE; SQLSTATE: HY000

  Message: Giving %d client threads a chance to die gracefully

  ER_DEPART_WITH_GRACE was added in 8.0.2.

- Error number: MY-010068; Symbol: ER_CA_SELF_SIGNED; SQLSTATE: HY000

  Message: CA certificate %s is self signed.

  ER_CA_SELF_SIGNED was added in 8.0.2.

- Error number: MY-010069; Symbol: ER_SSL_LIBRARY_ERROR; SQLSTATE: HY000

  Message: Failed to set up SSL because of the following SSL library error: %s

  ER_SSL_LIBRARY_ERROR was added in 8.0.2.

- Error number: MY-010070; Symbol: ER_NO_THD_NO_UUID; SQLSTATE: HY000

  Message: Failed to generate a server UUID because it is failed to allocate the THD.

  ER_NO_THD_NO_UUID was added in 8.0.2.

- Error number: MY-010071; Symbol: ER_UUID_SALT; SQLSTATE: HY000

  Message: Salting uuid generator variables, current_pid: %lu, server_start_time: %lu, bytes_sent: %llu,

  ER_UUID_SALT was added in 8.0.2.

- Error number: MY-010072; Symbol: ER_UUID_IS; SQLSTATE: HY000

  Message: Generated uuid: '%s', server_start_time: %lu, bytes_sent: %llu

  ER_UUID_IS was added in 8.0.2.

- Error number: MY-010073; Symbol: ER_UUID_INVALID; SQLSTATE: HY000

  Message: The server_uuid stored in auto.cnf file is not a valid UUID.

  ER_UUID_INVALID was added in 8.0.2.

- Error number: MY-010074; Symbol: ER_UUID_SCRUB; SQLSTATE: HY000

  Message: Garbage characters found at the end of the server_uuid value in auto.cnf file. It should be of length '%d' (UUID_LENGTH). Clear it and restart the server.

ER_UUID_SCRUB was added in 8.0.2.

- Error number: MY-010075; Symbol: ER_CREATING_NEW_UUID; SQLSTATE: HY000

  Message: No existing UUID has been found, so we assume that this is the first time that this server has been started. Generating a new UUID: %s.

  ER_CREATING_NEW_UUID was added in 8.0.2.

- Error number: MY-010076; Symbol: ER_CANT_CREATE_UUID; SQLSTATE: HY000

  Message: Initialization of the server's UUID failed because it could not be read from the auto.cnf file. If this is a new server, the initialization failed because it was not possible to generate a new UUID.

  ER_CANT_CREATE_UUID was added in 8.0.2.

- Error number: MY-010077; Symbol: ER_UNKNOWN_UNSUPPORTED_STORAGE_ENGINE; SQLSTATE: HY000

  Message: Unknown/unsupported storage engine: %s

  ER_UNKNOWN_UNSUPPORTED_STORAGE_ENGINE was added in 8.0.2.

- Error number: MY-010078; Symbol: ER_SECURE_AUTH_VALUE_UNSUPPORTED; SQLSTATE: HY000

  Message: Unsupported value 0 for secure-auth

  ER_SECURE_AUTH_VALUE_UNSUPPORTED was added in 8.0.2.

- Error number: MY-010079; Symbol: ER_INVALID_INSTRUMENT; SQLSTATE: HY000

  Message: Invalid instrument name or value for performance_schema_instrument '%s'

  ER_INVALID_INSTRUMENT was added in 8.0.2.

- Error number: MY-010080; Symbol: ER_INNODB_MANDATORY; SQLSTATE: HY000

  Message: The use of InnoDB is mandatory since MySQL 5.7. The former options like '--innodb=0/1/OFF/ON' or '--skip-innodb' are ignored.

  ER_INNODB_MANDATORY was added in 8.0.2.

- Error number: MY-010082; Symbol: ER_OLD_PASSWORDS_NO_MIDDLE_GROUND; SQLSTATE: HY000

  Message: Invalid old_passwords mode: 1. Valid values are 2 and 0

  ER_OLD_PASSWORDS_NO_MIDDLE_GROUND was added in 8.0.2, removed after 8.0.15.

- Error number: MY-010083; Symbol: ER_VERBOSE_REQUIRES_HELP; SQLSTATE: HY000

  Message: --verbose is for use with --help; did you mean --log-error-verbosity?

  ER_VERBOSE_REQUIRES_HELP was added in 8.0.2.

- Error number: MY-010084; Symbol: ER_POINTLESS_WITHOUT_SLOWLOG; SQLSTATE: HY000

  Message: options --log-slow-admin-statements, --log-queries-not-using-indexes and --log-slow-slave-statements have no effect if --slow-query-log is not set

ER_POINTLESS_WITHOUT_SLOWLOG was added in 8.0.2.

- Error number: MY-010085; Symbol: ER_WASTEFUL_NET_BUFFER_SIZE; SQLSTATE: HY000

  Message: net_buffer_length (%lu) is set to be larger than max_allowed_packet (%lu). Please rectify.

  ER_WASTEFUL_NET_BUFFER_SIZE was added in 8.0.2.

- Error number: MY-010086; Symbol: ER_DEPRECATED_TIMESTAMP_IMPLICIT_DEFAULTS; SQLSTATE: HY000

  Message: TIMESTAMP with implicit DEFAULT value is deprecated. Please use --explicit_defaults_for_timestamp server option (see documentation for more details).

  ER_DEPRECATED_TIMESTAMP_IMPLICIT_DEFAULTS was added in 8.0.2.

- Error number: MY-010087; Symbol: ER_FT_BOOL_SYNTAX_INVALID; SQLSTATE: HY000

  Message: Invalid ft-boolean-syntax string: %s

  ER_FT_BOOL_SYNTAX_INVALID was added in 8.0.2.

- Error number: MY-010088; Symbol: ER_CREDENTIALLESS_AUTO_USER_BAD; SQLSTATE: HY000

  Message: 'NO_AUTO_CREATE_USER' sql mode was not set.

  ER_CREDENTIALLESS_AUTO_USER_BAD was added in 8.0.2.

- Error number: MY-010089; Symbol: ER_CONNECTION_HANDLING_OOM; SQLSTATE: HY000

  Message: Could not allocate memory for connection handling

  ER_CONNECTION_HANDLING_OOM was added in 8.0.2.

- Error number: MY-010090; Symbol: ER_THREAD_HANDLING_OOM; SQLSTATE: HY000

  Message: Could not allocate memory for thread handling

  ER_THREAD_HANDLING_OOM was added in 8.0.2.

- Error number: MY-010091; Symbol: ER_CANT_CREATE_TEST_FILE; SQLSTATE: HY000

  Message: Can't create test file %s

  ER_CANT_CREATE_TEST_FILE was added in 8.0.2.

- Error number: MY-010092; Symbol: ER_CANT_CREATE_PID_FILE; SQLSTATE: HY000

  Message: Can't start server: can't create PID file: %s

  ER_CANT_CREATE_PID_FILE was added in 8.0.2.

- Error number: MY-010093; Symbol: ER_CANT_REMOVE_PID_FILE; SQLSTATE: HY000

  Message: Unable to delete pid file: %s

  ER_CANT_REMOVE_PID_FILE was added in 8.0.2.

- Error number: MY-010094; Symbol: ER_CANT_CREATE_SHUTDOWN_THREAD; SQLSTATE: HY000

Message: Can't create thread to handle shutdown requests (errno= %d)

ER_CANT_CREATE_SHUTDOWN_THREAD was added in 8.0.2.

- Error number: MY-010095; Symbol: ER_SEC_FILE_PRIV_CANT_ACCESS_DIR; SQLSTATE: HY000

  Message: Failed to access directory for --secure-file-priv. Please make sure that directory exists and is accessible by MySQL Server. Supplied value : %s

  ER_SEC_FILE_PRIV_CANT_ACCESS_DIR was added in 8.0.2.

- Error number: MY-010096; Symbol: ER_SEC_FILE_PRIV_IGNORED; SQLSTATE: HY000

  Message: Ignoring --secure-file-priv value as server is running with --initialize(-insecure).

  ER_SEC_FILE_PRIV_IGNORED was added in 8.0.2.

- Error number: MY-010097; Symbol: ER_SEC_FILE_PRIV_EMPTY; SQLSTATE: HY000

  Message: Insecure configuration for --secure-file-priv: Current value does not restrict location of generated files. Consider setting it to a valid, non-empty path.

  ER_SEC_FILE_PRIV_EMPTY was added in 8.0.2.

- Error number: MY-010098; Symbol: ER_SEC_FILE_PRIV_NULL; SQLSTATE: HY000

  Message: --secure-file-priv is set to NULL. Operations related to importing and exporting data are disabled

  ER_SEC_FILE_PRIV_NULL was added in 8.0.2.

- Error number: MY-010099; Symbol: ER_SEC_FILE_PRIV_DIRECTORY_INSECURE; SQLSTATE: HY000

  Message: Insecure configuration for --secure-file-priv: %s is accessible through --secure-file-priv. Consider choosing a different directory.

  ER_SEC_FILE_PRIV_DIRECTORY_INSECURE was added in 8.0.2.

- Error number: MY-010100; Symbol: ER_SEC_FILE_PRIV_CANT_STAT; SQLSTATE: HY000

  Message: Failed to get stat for directory pointed out by --secure-file-priv

  ER_SEC_FILE_PRIV_CANT_STAT was added in 8.0.2.

- Error number: MY-010101; Symbol: ER_SEC_FILE_PRIV_DIRECTORY_PERMISSIONS; SQLSTATE: HY000

  Message: Insecure configuration for --secure-file-priv: Location is accessible to all OS users. Consider choosing a different directory.

  ER_SEC_FILE_PRIV_DIRECTORY_PERMISSIONS was added in 8.0.2.

- Error number: MY-010102; Symbol: ER_SEC_FILE_PRIV_ARGUMENT_TOO_LONG; SQLSTATE: HY000

  Message: Value for --secure-file-priv is longer than maximum limit of %d

ER_SEC_FILE_PRIV_ARGUMENT_TOO_LONG was added in 8.0.2.

- Error number: MY-010103; Symbol: ER_CANT_CREATE_NAMED_PIPES_THREAD; SQLSTATE: HY000

  Message: Can't create thread to handle named pipes (errno= %d)

  ER_CANT_CREATE_NAMED_PIPES_THREAD was added in 8.0.2.

- Error number: MY-010104; Symbol: ER_CANT_CREATE_TCPIP_THREAD; SQLSTATE: HY000

  Message: Can't create thread to handle TCP/IP (errno= %d)

  ER_CANT_CREATE_TCPIP_THREAD was added in 8.0.2.

- Error number: MY-010105; Symbol: ER_CANT_CREATE_SHM_THREAD; SQLSTATE: HY000

  Message: Can't create thread to handle shared memory (errno= %d)

  ER_CANT_CREATE_SHM_THREAD was added in 8.0.2.

- Error number: MY-010106; Symbol: ER_CANT_CREATE_INTERRUPT_THREAD; SQLSTATE: HY000

  Message: Can't create interrupt-thread (error %d, errno: %d)

  ER_CANT_CREATE_INTERRUPT_THREAD was added in 8.0.2.

- Error number: MY-010107; Symbol: ER_WRITABLE_CONFIG_REMOVED; SQLSTATE: HY000

  Message: World-writable config file '%s' has been removed.

  ER_WRITABLE_CONFIG_REMOVED was added in 8.0.2.

- Error number: MY-010108; Symbol: ER_CORE_VALUES; SQLSTATE: HY000

  Message: setrlimit could not change the size of core files to 'infinity'; We may not be able to generate a core file on signals

  ER_CORE_VALUES was added in 8.0.2.

- Error number: MY-010109; Symbol: ER_WRONG_DATETIME_SPEC; SQLSTATE: HY000

  Message: Wrong date/time format specifier: %s

  ER_WRONG_DATETIME_SPEC was added in 8.0.2.

- Error number: MY-010110; Symbol: ER_RPL_BINLOG_FILTERS_OOM; SQLSTATE: HY000

  Message: Could not allocate replication and binlog filters: %s

  ER_RPL_BINLOG_FILTERS_OOM was added in 8.0.2.

- Error number: MY-010111; Symbol: ER_KEYCACHE_OOM; SQLSTATE: HY000

  Message: Cannot allocate the keycache

  ER_KEYCACHE_OOM was added in 8.0.2.

- Error number: MY-010112; Symbol: ER_CONFIRMING_THE_FUTURE; SQLSTATE: HY000

Message: Current time has got past year 2038. Validating current time with %d iterations before initiating the normal server shutdown process.

ER_CONFIRMING_THE_FUTURE was added in 8.0.2.

- Error number: MY-010113; Symbol: ER_BACK_IN_TIME; SQLSTATE: HY000

  Message: Iteration %d: Obtained valid current time from system

  ER_BACK_IN_TIME was added in 8.0.2.

- Error number: MY-010114; Symbol: ER_FUTURE_DATE; SQLSTATE: HY000

  Message: Iteration %d: Current time obtained from system is greater than 2038

  ER_FUTURE_DATE was added in 8.0.2.

- Error number: MY-010115; Symbol: ER_UNSUPPORTED_DATE; SQLSTATE: HY000

  Message: This MySQL server doesn't support dates later then 2038

  ER_UNSUPPORTED_DATE was added in 8.0.2.

- Error number: MY-010116; Symbol: ER_STARTING_AS; SQLSTATE: HY000

  Message: %s (mysqld %s) starting as process %lu

  ER_STARTING_AS was added in 8.0.2.

- Error number: MY-010117; Symbol: ER_SHUTTING_DOWN_SLAVE_THREADS; SQLSTATE: HY000

  Message: Shutting down slave threads

  ER_SHUTTING_DOWN_SLAVE_THREADS was added in 8.0.2.

- Error number: MY-010118; Symbol: ER_DISCONNECTING_REMAINING_CLIENTS; SQLSTATE: HY000

  Message: Forcefully disconnecting %d remaining clients

  ER_DISCONNECTING_REMAINING_CLIENTS was added in 8.0.2.

- Error number: MY-010119; Symbol: ER_ABORTING; SQLSTATE: HY000

  Message: Aborting

  ER_ABORTING was added in 8.0.2.

- Error number: MY-010120; Symbol: ER_BINLOG_END; SQLSTATE: HY000

  Message: Binlog end

  ER_BINLOG_END was added in 8.0.2.

- Error number: MY-010121; Symbol: ER_CALL_ME_LOCALHOST; SQLSTATE: HY000

  Message: gethostname failed, using '%s' as hostname

  ER_CALL_ME_LOCALHOST was added in 8.0.2.

- Error number: MY-010122; Symbol: ER_USER_REQUIRES_ROOT; SQLSTATE: HY000

  Message: One can only use the --user switch if running as root

  ER_USER_REQUIRES_ROOT was added in 8.0.2.

- Error number: MY-010123; Symbol: ER_REALLY_RUN_AS_ROOT; SQLSTATE: HY000

  Message: Fatal error: Please read "Security" section of the manual to find out how to run mysqld as root!

  ER_REALLY_RUN_AS_ROOT was added in 8.0.2.

- Error number: MY-010124; Symbol: ER_USER_WHAT_USER; SQLSTATE: HY000

  Message: Fatal error: Can't change to run as user '%s' ; Please check that the user exists!

  ER_USER_WHAT_USER was added in 8.0.2.

- Error number: MY-010125; Symbol: ER_TRANSPORTS_WHAT_TRANSPORTS; SQLSTATE: HY000

  Message: Server is started with --require-secure-transport=ON but no secure transports (SSL or Shared Memory) are configured.

  ER_TRANSPORTS_WHAT_TRANSPORTS was added in 8.0.2.

- Error number: MY-010126; Symbol: ER_FAIL_SETGID; SQLSTATE: HY000

  Message: setgid: %s

  ER_FAIL_SETGID was added in 8.0.2.

- Error number: MY-010127; Symbol: ER_FAIL_SETUID; SQLSTATE: HY000

  Message: setuid: %s

  ER_FAIL_SETUID was added in 8.0.2.

- Error number: MY-010128; Symbol: ER_FAIL_SETREGID; SQLSTATE: HY000

  Message: setregid: %s

  ER_FAIL_SETREGID was added in 8.0.2.

- Error number: MY-010129; Symbol: ER_FAIL_SETREUID; SQLSTATE: HY000

  Message: setreuid: %s

  ER_FAIL_SETREUID was added in 8.0.2.

- Error number: MY-010130; Symbol: ER_FAIL_CHROOT; SQLSTATE: HY000

  Message: chroot: %s

  ER_FAIL_CHROOT was added in 8.0.2.

- Error number: MY-010131; Symbol: ER_WIN_LISTEN_BUT_HOW; SQLSTATE: HY000

  Message: TCP/IP, --shared-memory, or --named-pipe should be configured on NT OS

  ER_WIN_LISTEN_BUT_HOW was added in 8.0.2.

- Error number: MY-010132; Symbol: ER_NOT_RIGHT_NOW; SQLSTATE: HY000

  Message: CTRL-C ignored during startup

  ER_NOT_RIGHT_NOW was added in 8.0.2.

- Error number: MY-010133; Symbol: ER_FIXING_CLIENT_CHARSET; SQLSTATE: HY000

  Message: '%s' can not be used as client character set. '%s' will be used as default client character set.

  ER_FIXING_CLIENT_CHARSET was added in 8.0.2.

- Error number: MY-010134; Symbol: ER_OOM; SQLSTATE: HY000

  Message: Out of memory

  ER_OOM was added in 8.0.2.

- Error number: MY-010135; Symbol: ER_FAILED_TO_LOCK_MEM; SQLSTATE: HY000

  Message: Failed to lock memory. Errno: %d

  ER_FAILED_TO_LOCK_MEM was added in 8.0.2.

- Error number: MY-010136; Symbol: ER_MYINIT_FAILED; SQLSTATE: HY000

  Message: my_init() failed.

  ER_MYINIT_FAILED was added in 8.0.2.

- Error number: MY-010137; Symbol: ER_BEG_INITFILE; SQLSTATE: HY000

  Message: Execution of init_file \'%s\' started.

  ER_BEG_INITFILE was added in 8.0.2.

- Error number: MY-010138; Symbol: ER_END_INITFILE; SQLSTATE: HY000

  Message: Execution of init_file \'%s\' ended.

  ER_END_INITFILE was added in 8.0.2.

- Error number: MY-010139; Symbol: ER_CHANGED_MAX_OPEN_FILES; SQLSTATE: HY000

  Message: Changed limits: max_open_files: %lu (requested %lu)

  ER_CHANGED_MAX_OPEN_FILES was added in 8.0.2.

- Error number: MY-010140; Symbol: ER_CANT_INCREASE_MAX_OPEN_FILES; SQLSTATE: HY000

  Message: Could not increase number of max_open_files to more than %lu (request: %lu)

  ER_CANT_INCREASE_MAX_OPEN_FILES was added in 8.0.2.

- Error number: MY-010141; Symbol: ER_CHANGED_MAX_CONNECTIONS; SQLSTATE: HY000

  Message: Changed limits: max_connections: %lu (requested %lu)

  ER_CHANGED_MAX_CONNECTIONS was added in 8.0.2.

- Error number: MY-010142; Symbol: ER_CHANGED_TABLE_OPEN_CACHE; SQLSTATE: HY000

  Message: Changed limits: table_open_cache: %lu (requested %lu)

  ER_CHANGED_TABLE_OPEN_CACHE was added in 8.0.2.

- Error number: MY-010143; Symbol: ER_THE_USER_ABIDES; SQLSTATE: HY000

  Message: Ignoring user change to '%s' because the user was set to '%s' earlier on the command line

  ER_THE_USER_ABIDES was added in 8.0.2.

- Error number: MY-010144; Symbol: ER_RPL_CANT_ADD_DO_TABLE; SQLSTATE: HY000

  Message: Could not add do table rule '%s'!

  ER_RPL_CANT_ADD_DO_TABLE was added in 8.0.2.

- Error number: MY-010145; Symbol: ER_RPL_CANT_ADD_IGNORE_TABLE; SQLSTATE: HY000

  Message: Could not add ignore table rule '%s'!

  ER_RPL_CANT_ADD_IGNORE_TABLE was added in 8.0.2.

- Error number: MY-010146; Symbol: ER_TRACK_VARIABLES_BOGUS; SQLSTATE: HY000

  Message: The variable session_track_system_variables either has duplicate values or invalid values.

  ER_TRACK_VARIABLES_BOGUS was added in 8.0.2.

- Error number: MY-010147; Symbol: ER_EXCESS_ARGUMENTS; SQLSTATE: HY000

  Message: Too many arguments (first extra is '%s').

  ER_EXCESS_ARGUMENTS was added in 8.0.2.

- Error number: MY-010148; Symbol: ER_VERBOSE_HINT; SQLSTATE: HY000

  Message: Use --verbose --help to get a list of available options!

  ER_VERBOSE_HINT was added in 8.0.2.

- Error number: MY-010149; Symbol: ER_CANT_READ_ERRMSGS; SQLSTATE: HY000

  Message: Unable to read errmsg.sys file

  ER_CANT_READ_ERRMSGS was added in 8.0.2.

- Error number: MY-010150; Symbol: ER_CANT_INIT_DBS; SQLSTATE: HY000

  Message: Can't init databases

  ER_CANT_INIT_DBS was added in 8.0.2.

- Error number: MY-010151; Symbol: ER_LOG_OUTPUT_CONTRADICTORY; SQLSTATE: HY000

  Message: There were other values specified to log-output besides NONE. Disabling slow and general logs anyway.

  ER_LOG_OUTPUT_CONTRADICTORY was added in 8.0.2.

- Error number: MY-010152; Symbol: ER_NO_CSV_NO_LOG_TABLES; SQLSTATE: HY000

  Message: CSV engine is not present, falling back to the log files

  ER_NO_CSV_NO_LOG_TABLES was added in 8.0.2.

- Error number: MY-010153; Symbol: ER_RPL_REWRITEDB_MISSING_ARROW; SQLSTATE: HY000

  Message: Bad syntax in replicate-rewrite-db - missing '->'!

  ER_RPL_REWRITEDB_MISSING_ARROW was added in 8.0.2.

- Error number: MY-010154; Symbol: ER_RPL_REWRITEDB_EMPTY_FROM; SQLSTATE: HY000

  Message: Bad syntax in replicate-rewrite-db - empty FROM db!

  ER_RPL_REWRITEDB_EMPTY_FROM was added in 8.0.2.

- Error number: MY-010155; Symbol: ER_RPL_REWRITEDB_EMPTY_TO; SQLSTATE: HY000

  Message: Bad syntax in replicate-rewrite-db - empty TO db!

  ER_RPL_REWRITEDB_EMPTY_TO was added in 8.0.2.

- Error number: MY-010156; Symbol: ER_LOG_FILES_GIVEN_LOG_OUTPUT_IS_TABLE; SQLSTATE: HY000

  Message: Although a path was specified for the %s, log tables are used. To enable logging to files use the --log-output=file option.

  ER_LOG_FILES_GIVEN_LOG_OUTPUT_IS_TABLE was added in 8.0.2.

- Error number: MY-010157; Symbol: ER_LOG_FILE_INVALID; SQLSTATE: HY000

  Message: Invalid value for %s: %s

  ER_LOG_FILE_INVALID was added in 8.0.2.

- Error number: MY-010158; Symbol: ER_LOWER_CASE_TABLE_NAMES_CS_DD_ON_CI_FS_UNSUPPORTED; SQLSTATE: HY000

  Message: The server option 'lower_case_table_names' is configured to use case sensitive table names but the data directory is on a case-insensitive file system which is an unsupported combination. Please consider either using a case sensitive file system for your data directory or switching to a case-insensitive table name mode.

  ER_LOWER_CASE_TABLE_NAMES_CS_DD_ON_CI_FS_UNSUPPORTED was added in 8.0.2.

- Error number: MY-010159; Symbol: ER_LOWER_CASE_TABLE_NAMES_USING_2; SQLSTATE: HY000

  Message: Setting lower_case_table_names=2 because file system for %s is case insensitive

  ER_LOWER_CASE_TABLE_NAMES_USING_2 was added in 8.0.2.

- Error number: MY-010160; Symbol: ER_LOWER_CASE_TABLE_NAMES_USING_0; SQLSTATE: HY000

  Message: lower_case_table_names was set to 2, even though your the file system '%s' is case sensitive. Now setting lower_case_table_names to 0 to avoid future problems.

ER_LOWER_CASE_TABLE_NAMES_USING_0 was added in 8.0.2.

- Error number: MY-010161; Symbol: ER_NEED_LOG_BIN; SQLSTATE: HY000

  Message: You need to use --log-bin to make %s work.

  ER_NEED_LOG_BIN was added in 8.0.2.

- Error number: MY-010162; Symbol: ER_NEED_FILE_INSTEAD_OF_DIR; SQLSTATE: HY000

  Message: Path '%s' is a directory name, please specify a file name for %s option

  ER_NEED_FILE_INSTEAD_OF_DIR was added in 8.0.2.

- Error number: MY-010163; Symbol: ER_LOG_BIN_BETTER_WITH_NAME; SQLSTATE: HY000

  Message: No argument was provided to --log-bin, and --log-bin-index was not used; so replication may break when this MySQL server acts as a master and has his hostname changed!! Please use '--log-bin=%s' to avoid this problem.

  ER_LOG_BIN_BETTER_WITH_NAME was added in 8.0.2.

- Error number: MY-010164; Symbol: ER_BINLOG_NEEDS_SERVERID; SQLSTATE: HY000

  Message: You have enabled the binary log, but you haven't provided the mandatory server-id. Please refer to the proper server start-up parameters documentation

  ER_BINLOG_NEEDS_SERVERID was added in 8.0.2.

- Error number: MY-010165; Symbol: ER_RPL_CANT_MAKE_PATHS; SQLSTATE: HY000

  Message: Unable to create replication path names: out of memory or path names too long (path name exceeds %d or file name exceeds %d).

  ER_RPL_CANT_MAKE_PATHS was added in 8.0.2.

- Error number: MY-010166; Symbol: ER_CANT_INITIALIZE_GTID; SQLSTATE: HY000

  Message: Failed to initialize GTID structures.

  ER_CANT_INITIALIZE_GTID was added in 8.0.2.

- Error number: MY-010167; Symbol: ER_CANT_INITIALIZE_EARLY_PLUGINS; SQLSTATE: HY000

  Message: Failed to initialize early plugins.

  ER_CANT_INITIALIZE_EARLY_PLUGINS was added in 8.0.2.

- Error number: MY-010168; Symbol: ER_CANT_INITIALIZE_BUILTIN_PLUGINS; SQLSTATE: HY000

  Message: Failed to initialize builtin plugins.

  ER_CANT_INITIALIZE_BUILTIN_PLUGINS was added in 8.0.2.

- Error number: MY-010169; Symbol: ER_CANT_INITIALIZE_DYNAMIC_PLUGINS; SQLSTATE: HY000

  Message: Failed to initialize dynamic plugins.

ER_CANT_INITIALIZE_DYNAMIC_PLUGINS was added in 8.0.2.

- Error number: MY-010170; Symbol: ER_PERFSCHEMA_INIT_FAILED; SQLSTATE: HY000

  Message: Performance schema disabled (reason: init failed).

  ER_PERFSCHEMA_INIT_FAILED was added in 8.0.2.

- Error number: MY-010171; Symbol: ER_STACKSIZE_UNEXPECTED; SQLSTATE: HY000

  Message: Asked for %lu thread stack, but got %ld

  ER_STACKSIZE_UNEXPECTED was added in 8.0.2.

- Error number: MY-010172; Symbol: ER_CANT_SET_DATADIR; SQLSTATE: HY000

  Message: failed to set datadir to %s

  ER_CANT_SET_DATADIR was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010173; Symbol: ER_CANT_STAT_DATADIR; SQLSTATE: HY000

  Message: Can't read data directory's stats (%d): %s. Assuming that it's not owned by the same user/group

  ER_CANT_STAT_DATADIR was added in 8.0.2.

- Error number: MY-010174; Symbol: ER_CANT_CHOWN_DATADIR; SQLSTATE: HY000

  Message: Can't change data directory owner to %s

  ER_CANT_CHOWN_DATADIR was added in 8.0.2.

- Error number: MY-010175; Symbol: ER_CANT_SET_UP_PERSISTED_VALUES; SQLSTATE: HY000

  Message: Setting persistent options failed.

  ER_CANT_SET_UP_PERSISTED_VALUES was added in 8.0.2.

- Error number: MY-010176; Symbol: ER_CANT_SAVE_GTIDS; SQLSTATE: HY000

  Message: Failed to save the set of Global Transaction Identifiers of the last binary log into the mysql.gtid_executed table while the server was shutting down. The next server restart will make another attempt to save Global Transaction Identifiers into the table.

  ER_CANT_SAVE_GTIDS was added in 8.0.2.

- Error number: MY-010177; Symbol: ER_AUTH_CANT_SET_DEFAULT_PLUGIN; SQLSTATE: HY000

  Message: Can't start server: Invalid value for --default-authentication-plugin

  ER_AUTH_CANT_SET_DEFAULT_PLUGIN was added in 8.0.2.

- Error number: MY-010178; Symbol: ER_CANT_JOIN_SHUTDOWN_THREAD; SQLSTATE: HY000

  Message: Could not join %sthread. error:%d

  ER_CANT_JOIN_SHUTDOWN_THREAD was added in 8.0.2.

- Error number: MY-010179; Symbol: ER_CANT_HASH_DO_AND_IGNORE_RULES; SQLSTATE: HY000

  Message: An error occurred while building do_table and ignore_table rules to hashes for global replication filter.

  ER_CANT_HASH_DO_AND_IGNORE_RULES was added in 8.0.2.

- Error number: MY-010180; Symbol: ER_CANT_OPEN_CA; SQLSTATE: HY000

  Message: Error opening CA certificate file

  ER_CANT_OPEN_CA was added in 8.0.2.

- Error number: MY-010181; Symbol: ER_CANT_ACCESS_CAPATH; SQLSTATE: HY000

  Message: Error accessing directory pointed by --ssl-capath

  ER_CANT_ACCESS_CAPATH was added in 8.0.2.

- Error number: MY-010182; Symbol: ER_SSL_TRYING_DATADIR_DEFAULTS; SQLSTATE: HY000

  Message: Found %s, %s and %s in data directory. Trying to enable SSL support using them.

  ER_SSL_TRYING_DATADIR_DEFAULTS was added in 8.0.2.

- Error number: MY-010183; Symbol: ER_AUTO_OPTIONS_FAILED; SQLSTATE: HY000

  Message: Failed to create %s(file: '%s', errno %d)

  ER_AUTO_OPTIONS_FAILED was added in 8.0.2.

- Error number: MY-010184; Symbol: ER_CANT_INIT_TIMER; SQLSTATE: HY000

  Message: Failed to initialize timer component (errno %d).

  ER_CANT_INIT_TIMER was added in 8.0.2.

- Error number: MY-010185; Symbol: ER_SERVERID_TOO_LARGE; SQLSTATE: HY000

  Message: server-id configured is too large to represent with server-id-bits configured.

  ER_SERVERID_TOO_LARGE was added in 8.0.2.

- Error number: MY-010186; Symbol: ER_DEFAULT_SE_UNAVAILABLE; SQLSTATE: HY000

  Message: Default%s storage engine (%s) is not available

  ER_DEFAULT_SE_UNAVAILABLE was added in 8.0.2.

- Error number: MY-010187; Symbol: ER_CANT_OPEN_ERROR_LOG; SQLSTATE: HY000

  Message: Could not open file '%s' for error logging%s%s

  ER_CANT_OPEN_ERROR_LOG was added in 8.0.2.

- Error number: MY-010188; Symbol: ER_INVALID_ERROR_LOG_NAME; SQLSTATE: HY000

  Message: Invalid log file name after expanding symlinks: '%s'

  ER_INVALID_ERROR_LOG_NAME was added in 8.0.2.

- Error number: MY-010189; Symbol: ER_RPL_INFINITY_DENIED; SQLSTATE: HY000

  Message: using --replicate-same-server-id in conjunction with --log-slave-updates is impossible, it would lead to infinite loops in this server.

  ER_RPL_INFINITY_DENIED was added in 8.0.2.

- Error number: MY-010190; Symbol: ER_RPL_INFINITY_IGNORED; SQLSTATE: HY000

  Message: using --replicate-same-server-id in conjunction with --log-slave-updates would lead to infinite loops in this server. However this will be ignored as the --log-bin option is not defined or your server is running with global transaction identiers enabled.

  ER_RPL_INFINITY_IGNORED was added in 8.0.2.

- Error number: MY-010191; Symbol: ER_NDB_TABLES_NOT_READY; SQLSTATE: HY000

  Message: NDB : Tables not available after %lu seconds. Consider increasing --ndb-wait-setup value

  ER_NDB_TABLES_NOT_READY was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010192; Symbol: ER_TABLE_CHECK_INTACT; SQLSTATE: HY000

  Message: %s

  ER_TABLE_CHECK_INTACT was added in 8.0.2.

- Error number: MY-010193; Symbol: ER_DD_TABLESPACE_NOT_FOUND; SQLSTATE: HY000

  Message: Unable to start server. The data dictionary tablespace '%s' does not exist.

  ER_DD_TABLESPACE_NOT_FOUND was added in 8.0.2.

- Error number: MY-010194; Symbol: ER_DD_TRG_CONNECTION_COLLATION_MISSING; SQLSTATE: HY000

  Message: Connection collation is missing for trigger of table %s. Using default connection collation.

  ER_DD_TRG_CONNECTION_COLLATION_MISSING was added in 8.0.2.

- Error number: MY-010195; Symbol: ER_DD_TRG_DB_COLLATION_MISSING; SQLSTATE: HY000

  Message: Database collation is missing for trigger of table %s. Using Default character set.

  ER_DD_TRG_DB_COLLATION_MISSING was added in 8.0.2.

- Error number: MY-010196; Symbol: ER_DD_TRG_DEFINER_OOM; SQLSTATE: HY000

  Message: Error in Memory allocation for Definer %s for Trigger.

  ER_DD_TRG_DEFINER_OOM was added in 8.0.2.

- Error number: MY-010197; Symbol: ER_DD_TRG_FILE_UNREADABLE; SQLSTATE: HY000

  Message: Error in reading %s.TRG file.

  ER_DD_TRG_FILE_UNREADABLE was added in 8.0.2.

- Error number: MY-010198; Symbol: ER_TRG_CANT_PARSE; SQLSTATE: HY000

Message: Error in parsing Triggers from %s.TRG file.

ER_TRG_CANT_PARSE was added in 8.0.2.

- Error number: MY-010199; Symbol: ER_DD_TRG_CANT_ADD; SQLSTATE: HY000

  Message: Error in creating DD entry for Trigger %s.%s

  ER_DD_TRG_CANT_ADD was added in 8.0.2.

- Error number: MY-010200; Symbol: ER_DD_CANT_RESOLVE_VIEW; SQLSTATE: HY000

  Message: Resolving dependency for the view '%s.%s' failed. View is no more valid to use

  ER_DD_CANT_RESOLVE_VIEW was added in 8.0.2.

- Error number: MY-010201; Symbol: ER_DD_VIEW_WITHOUT_DEFINER; SQLSTATE: HY000

  Message: %s.%s has no definer (as per an old view format). Current user is used as definer. Please recreate the view.

  ER_DD_VIEW_WITHOUT_DEFINER was added in 8.0.2.

- Error number: MY-010202; Symbol: ER_PLUGIN_INIT_FAILED; SQLSTATE: HY000

  Message: Plugin '%s' init function returned error.

  ER_PLUGIN_INIT_FAILED was added in 8.0.2.

- Error number: MY-010203; Symbol: ER_RPL_TRX_DELEGATES_INIT_FAILED; SQLSTATE: HY000

  Message: Initialization of transaction delegates failed. Please report a bug.

  ER_RPL_TRX_DELEGATES_INIT_FAILED was added in 8.0.2.

- Error number: MY-010204; Symbol: ER_RPL_BINLOG_STORAGE_DELEGATES_INIT_FAILED; SQLSTATE: HY000

  Message: Initialization binlog storage delegates failed. Please report a bug.

  ER_RPL_BINLOG_STORAGE_DELEGATES_INIT_FAILED was added in 8.0.2.

- Error number: MY-010205; Symbol: ER_RPL_BINLOG_TRANSMIT_DELEGATES_INIT_FAILED; SQLSTATE: HY000

  Message: Initialization of binlog transmit delegates failed. Please report a bug.

  ER_RPL_BINLOG_TRANSMIT_DELEGATES_INIT_FAILED was added in 8.0.2.

- Error number: MY-010206; Symbol: ER_RPL_BINLOG_RELAY_DELEGATES_INIT_FAILED; SQLSTATE: HY000

  Message: Initialization binlog relay IO delegates failed. Please report a bug.

  ER_RPL_BINLOG_RELAY_DELEGATES_INIT_FAILED was added in 8.0.2.

- Error number: MY-010207; Symbol: ER_RPL_PLUGIN_FUNCTION_FAILED; SQLSTATE: HY000

  Message: Run function '...' in plugin '%s' failed

ER_RPL_PLUGIN_FUNCTION_FAILED was added in 8.0.2.

- Error number: MY-010208; Symbol: ER_SQL_HA_READ_FAILED; SQLSTATE: HY000

  Message: mysql_ha_read: Got error %d when reading table '%s'

  ER_SQL_HA_READ_FAILED was added in 8.0.2.

- Error number: MY-010209; Symbol: ER_SR_BOGUS_VALUE; SQLSTATE: HY000

  Message: Stored routine '%s'.'%s': invalid value in column %s.

  ER_SR_BOGUS_VALUE was added in 8.0.2.

- Error number: MY-010210; Symbol: ER_SR_INVALID_CONTEXT; SQLSTATE: HY000

  Message: Invalid creation context '%s.%s'.

  ER_SR_INVALID_CONTEXT was added in 8.0.2.

- Error number: MY-010211; Symbol: ER_READING_TABLE_FAILED; SQLSTATE: HY000

  Message: Got error %d when reading table '%s'

  ER_READING_TABLE_FAILED was added in 8.0.2.

- Error number: MY-010212; Symbol: ER_DES_FILE_WRONG_KEY; SQLSTATE: HY000

  Message: load_des_file: Found wrong key_number: %c

  ER_DES_FILE_WRONG_KEY was added in 8.0.2.

- Error number: MY-010213; Symbol: ER_CANT_SET_PERSISTED; SQLSTATE: HY000

  Message: Failed to set persisted options.

  ER_CANT_SET_PERSISTED was added in 8.0.2.

- Error number: MY-010214; Symbol: ER_JSON_PARSE_ERROR; SQLSTATE: HY000

  Message: JSON parsing error

  ER_JSON_PARSE_ERROR was added in 8.0.2.

- Error number: MY-010215; Symbol: ER_CONFIG_OPTION_WITHOUT_GROUP; SQLSTATE: HY000

  Message: Found option without preceding group in config file

  ER_CONFIG_OPTION_WITHOUT_GROUP was added in 8.0.2.

- Error number: MY-010216; Symbol: ER_VALGRIND_DO_QUICK_LEAK_CHECK; SQLSTATE: HY000

  Message: VALGRIND_DO_QUICK_LEAK_CHECK

  ER_VALGRIND_DO_QUICK_LEAK_CHECK was added in 8.0.2.

- Error number: MY-010217; Symbol: ER_VALGRIND_COUNT_LEAKS; SQLSTATE: HY000

  Message: VALGRIND_COUNT_LEAKS reports %lu leaked bytes for query '%.*s'

ER_VALGRIND_COUNT_LEAKS was added in 8.0.2.

- Error number: MY-010218; Symbol: ER_LOAD_DATA_INFILE_FAILED_IN_UNEXPECTED_WAY; SQLSTATE: HY000

  Message: LOAD DATA INFILE in the slave SQL Thread can only read from --slave-load-tmpdir. Please, report a bug.

  ER_LOAD_DATA_INFILE_FAILED_IN_UNEXPECTED_WAY was added in 8.0.2.

- Error number: MY-010219; Symbol: ER_UNKNOWN_ERROR_NUMBER; SQLSTATE: HY000

  Message: Got unknown error: %d

  ER_UNKNOWN_ERROR_NUMBER was added in 8.0.2.

- Error number: MY-010220; Symbol: ER_UDF_CANT_ALLOC_FOR_STRUCTURES; SQLSTATE: HY000

  Message: Can't allocate memory for udf structures

  ER_UDF_CANT_ALLOC_FOR_STRUCTURES was added in 8.0.2.

- Error number: MY-010221; Symbol: ER_UDF_CANT_ALLOC_FOR_FUNCTION; SQLSTATE: HY000

  Message: Can't alloc memory for udf function: '%s'

  ER_UDF_CANT_ALLOC_FOR_FUNCTION was added in 8.0.2.

- Error number: MY-010222; Symbol: ER_UDF_INVALID_ROW_IN_FUNCTION_TABLE; SQLSTATE: HY000

  Message: Invalid row in mysql.func table for function '%s'

  ER_UDF_INVALID_ROW_IN_FUNCTION_TABLE was added in 8.0.2.

- Error number: MY-010223; Symbol: ER_UDF_CANT_OPEN_FUNCTION_TABLE; SQLSTATE: HY000

  Message: Could not open the mysql.func table. Please perform the MySQL upgrade procedure.

  ER_UDF_CANT_OPEN_FUNCTION_TABLE was added in 8.0.2.

- Error number: MY-010224; Symbol: ER_XA_RECOVER_FOUND_TRX_IN_SE; SQLSTATE: HY000

  Message: Found %d prepared transaction(s) in %s

  ER_XA_RECOVER_FOUND_TRX_IN_SE was added in 8.0.2.

- Error number: MY-010225; Symbol: ER_XA_RECOVER_FOUND_XA_TRX; SQLSTATE: HY000

  Message: Found %d prepared XA transactions

  ER_XA_RECOVER_FOUND_XA_TRX was added in 8.0.2.

- Error number: MY-010226; Symbol: ER_XA_IGNORING_XID; SQLSTATE: HY000

  Message: ignore xid %s

  ER_XA_IGNORING_XID was added in 8.0.2.

- Error number: MY-010227; Symbol: ER_XA_COMMITTING_XID; SQLSTATE: HY000

  Message: commit xid %s

  ER_XA_COMMITTING_XID was added in 8.0.2.

- Error number: MY-010228; Symbol: ER_XA_ROLLING_BACK_XID; SQLSTATE: HY000

  Message: rollback xid %s

  ER_XA_ROLLING_BACK_XID was added in 8.0.2.

- Error number: MY-010229; Symbol: ER_XA_STARTING_RECOVERY; SQLSTATE: HY000

  Message: Starting crash recovery...

  ER_XA_STARTING_RECOVERY was added in 8.0.2.

- Error number: MY-010230; Symbol: ER_XA_NO_MULTI_2PC_HEURISTIC_RECOVER; SQLSTATE: HY000

  Message: --tc-heuristic-recover rollback strategy is not safe on systems with more than one 2-phase-commit-capable storage engine. Aborting crash recovery.

  ER_XA_NO_MULTI_2PC_HEURISTIC_RECOVER was added in 8.0.2.

- Error number: MY-010231; Symbol: ER_XA_RECOVER_EXPLANATION; SQLSTATE: HY000

  Message: Found %d prepared transactions! It means that mysqld was not shut down properly last time and critical recovery information (last binlog or %s file) was manually deleted after a crash. You have to start mysqld with --tc-heuristic-recover switch to commit or rollback pending transactions.

  ER_XA_RECOVER_EXPLANATION was added in 8.0.2.

- Error number: MY-010232; Symbol: ER_XA_RECOVERY_DONE; SQLSTATE: HY000

  Message: Crash recovery finished.

  ER_XA_RECOVERY_DONE was added in 8.0.2.

- Error number: MY-010233; Symbol: ER_TRX_GTID_COLLECT_REJECT; SQLSTATE: HY000

  Message: Failed to collect GTID to send in the response packet!

  ER_TRX_GTID_COLLECT_REJECT was added in 8.0.2.

- Error number: MY-010234; Symbol: ER_SQL_AUTHOR_DEFAULT_ROLES_FAIL; SQLSTATE: HY000

  Message: MYSQL.DEFAULT_ROLES couldn't be updated for authorization identifier %s

  ER_SQL_AUTHOR_DEFAULT_ROLES_FAIL was added in 8.0.2.

- Error number: MY-010235; Symbol: ER_SQL_USER_TABLE_CREATE_WARNING; SQLSTATE: HY000

  Message: Following users were specified in CREATE USER IF NOT EXISTS but they already exist. Corresponding entry in binary log used default authentication plugin '%s' to rewrite authentication information (if any) for them: %s

  ER_SQL_USER_TABLE_CREATE_WARNING was added in 8.0.2.

- Error number: MY-010236; Symbol: ER_SQL_USER_TABLE_ALTER_WARNING; SQLSTATE: HY000

  Message: Following users were specified in ALTER USER IF EXISTS but they do not exist. Corresponding entry in binary log used default authentication plugin '%s' to rewrite authentication information (if any) for them: %s

  ER_SQL_USER_TABLE_ALTER_WARNING was added in 8.0.2.

- Error number: MY-010237; Symbol: ER_ROW_IN_WRONG_PARTITION_PLEASE_REPAIR; SQLSTATE: HY000

  Message: Table '%s' corrupted: row in wrong partition: %s -- Please REPAIR the table!

  ER_ROW_IN_WRONG_PARTITION_PLEASE_REPAIR was added in 8.0.2.

- Error number: MY-010238; Symbol: ER_MYISAM_CRASHED_ERROR_IN_THREAD; SQLSTATE: HY000

  Message: Got an error from thread_id=%u, %s:%d

  ER_MYISAM_CRASHED_ERROR_IN_THREAD was added in 8.0.2.

- Error number: MY-010239; Symbol: ER_MYISAM_CRASHED_ERROR_IN; SQLSTATE: HY000

  Message: Got an error from unknown thread, %s:%d

  ER_MYISAM_CRASHED_ERROR_IN was added in 8.0.2.

- Error number: MY-010240; Symbol: ER_TOO_MANY_STORAGE_ENGINES; SQLSTATE: HY000

  Message: Too many storage engines!

  ER_TOO_MANY_STORAGE_ENGINES was added in 8.0.2.

- Error number: MY-010241; Symbol: ER_SE_TYPECODE_CONFLICT; SQLSTATE: HY000

  Message: Storage engine '%s' has conflicting typecode. Assigning value %d.

  ER_SE_TYPECODE_CONFLICT was added in 8.0.2.

- Error number: MY-010242; Symbol: ER_TRX_WRITE_SET_OOM; SQLSTATE: HY000

  Message: Out of memory on transaction write set extraction

  ER_TRX_WRITE_SET_OOM was added in 8.0.2.

- Error number: MY-010243; Symbol: ER_HANDLERTON_OOM; SQLSTATE: HY000

  Message: Unable to allocate memory for plugin '%s' handlerton.

  ER_HANDLERTON_OOM was added in 8.0.2.

- Error number: MY-010244; Symbol: ER_CONN_SHM_LISTENER; SQLSTATE: HY000

  Message: Shared memory setting up listener

  ER_CONN_SHM_LISTENER was added in 8.0.2.

- Error number: MY-010245; Symbol: ER_CONN_SHM_CANT_CREATE_SERVICE; SQLSTATE: HY000

  Message: Can't create shared memory service: %s. : %s

ER_CONN_SHM_CANT_CREATE_SERVICE was added in 8.0.2.

- Error number: MY-010246; Symbol: ER_CONN_SHM_CANT_CREATE_CONNECTION; SQLSTATE: HY000

  Message: Can't create shared memory connection: %s. : %s

  ER_CONN_SHM_CANT_CREATE_CONNECTION was added in 8.0.2.

- Error number: MY-010247; Symbol: ER_CONN_PIP_CANT_CREATE_EVENT; SQLSTATE: HY000

  Message: Can't create event, last error=%u

  ER_CONN_PIP_CANT_CREATE_EVENT was added in 8.0.2.

- Error number: MY-010248; Symbol: ER_CONN_PIP_CANT_CREATE_PIPE; SQLSTATE: HY000

  Message: Can't create new named pipe!: %s

  ER_CONN_PIP_CANT_CREATE_PIPE was added in 8.0.2.

- Error number: MY-010249; Symbol: ER_CONN_PER_THREAD_NO_THREAD; SQLSTATE: HY000

  Message: Can't create thread to handle new connection(errno= %d)

  ER_CONN_PER_THREAD_NO_THREAD was added in 8.0.2.

- Error number: MY-010250; Symbol: ER_CONN_TCP_NO_SOCKET; SQLSTATE: HY000

  Message: Failed to create a socket for %s '%s': errno: %d.

  ER_CONN_TCP_NO_SOCKET was added in 8.0.2.

- Error number: MY-010251; Symbol: ER_CONN_TCP_CREATED; SQLSTATE: HY000

  Message: Server socket created on IP: '%s'.

  ER_CONN_TCP_CREATED was added in 8.0.2.

- Error number: MY-010252; Symbol: ER_CONN_TCP_ADDRESS; SQLSTATE: HY000

  Message: Server hostname (bind-address): '%s'; port: %d

  ER_CONN_TCP_ADDRESS was added in 8.0.2.

- Error number: MY-010253; Symbol: ER_CONN_TCP_IPV6_AVAILABLE; SQLSTATE: HY000

  Message: IPv6 is available.

  ER_CONN_TCP_IPV6_AVAILABLE was added in 8.0.2.

- Error number: MY-010254; Symbol: ER_CONN_TCP_IPV6_UNAVAILABLE; SQLSTATE: HY000

  Message: IPv6 is not available.

  ER_CONN_TCP_IPV6_UNAVAILABLE was added in 8.0.2.

- Error number: MY-010255; Symbol: ER_CONN_TCP_ERROR_WITH_STRERROR; SQLSTATE: HY000

Message: Can't create IP socket: %s

ER_CONN_TCP_ERROR_WITH_STRERROR was added in 8.0.2.

- Error number: MY-010256; Symbol: ER_CONN_TCP_CANT_RESOLVE_HOSTNAME; SQLSTATE: HY000

  Message: Can't start server: cannot resolve hostname!

  ER_CONN_TCP_CANT_RESOLVE_HOSTNAME was added in 8.0.2.

- Error number: MY-010257; Symbol: ER_CONN_TCP_IS_THERE_ANOTHER_USING_PORT; SQLSTATE: HY000

  Message: Do you already have another mysqld server running on port: %d ?

  ER_CONN_TCP_IS_THERE_ANOTHER_USING_PORT was added in 8.0.2.

- Error number: MY-010258; Symbol: ER_CONN_UNIX_IS_THERE_ANOTHER_USING_SOCKET; SQLSTATE: HY000

  Message: Do you already have another mysqld server running on socket: %s ?

  ER_CONN_UNIX_IS_THERE_ANOTHER_USING_SOCKET was added in 8.0.2.

- Error number: MY-010259; Symbol: ER_CONN_UNIX_PID_CLAIMED_SOCKET_FILE; SQLSTATE: HY000

  Message: Another process with pid %d is using unix socket file.

  ER_CONN_UNIX_PID_CLAIMED_SOCKET_FILE was added in 8.0.2.

- Error number: MY-010260; Symbol: ER_CONN_TCP_CANT_RESET_V6ONLY; SQLSTATE: HY000

  Message: Failed to reset IPV6_V6ONLY flag (error: %d). The server will listen to IPv6 addresses only.

  ER_CONN_TCP_CANT_RESET_V6ONLY was added in 8.0.2.

- Error number: MY-010261; Symbol: ER_CONN_TCP_BIND_RETRY; SQLSTATE: HY000

  Message: Retrying bind on TCP/IP port %u

  ER_CONN_TCP_BIND_RETRY was added in 8.0.2.

- Error number: MY-010262; Symbol: ER_CONN_TPC_BIND_FAIL; SQLSTATE: HY000

  Message: Can't start server: Bind on TCP/IP port: %s

  ER_CONN_TPC_BIND_FAIL was added in 8.0.2, removed after 8.0.11.

- Error number: MY-010262; Symbol: ER_CONN_TCP_BIND_FAIL; SQLSTATE: HY000

  Message: Can't start server: Bind on TCP/IP port: %s

  ER_CONN_TCP_BIND_FAIL was added in 8.0.12.

- Error number: MY-010263; Symbol: ER_CONN_TCP_IP_NOT_LOGGED; SQLSTATE: HY000

  Message: Fails to print out IP-address.

ER_CONN_TCP_IP_NOT_LOGGED was added in 8.0.2.

- Error number: MY-010264; Symbol: ER_CONN_TCP_RESOLVE_INFO; SQLSTATE: HY000

  Message: - '%s' resolves to '%s';

  ER_CONN_TCP_RESOLVE_INFO was added in 8.0.2.

- Error number: MY-010265; Symbol: ER_CONN_TCP_START_FAIL; SQLSTATE: HY000

  Message: Can't start server: listen() on TCP/IP port: %s

  ER_CONN_TCP_START_FAIL was added in 8.0.2.

- Error number: MY-010266; Symbol: ER_CONN_TCP_LISTEN_FAIL; SQLSTATE: HY000

  Message: listen() on TCP/IP failed with error %d

  ER_CONN_TCP_LISTEN_FAIL was added in 8.0.2.

- Error number: MY-010267; Symbol: ER_CONN_UNIX_PATH_TOO_LONG; SQLSTATE: HY000

  Message: The socket file path is too long (> %u): %s

  ER_CONN_UNIX_PATH_TOO_LONG was added in 8.0.2.

- Error number: MY-010268; Symbol: ER_CONN_UNIX_LOCK_FILE_FAIL; SQLSTATE: HY000

  Message: Unable to setup unix socket lock file.

  ER_CONN_UNIX_LOCK_FILE_FAIL was added in 8.0.2.

- Error number: MY-010269; Symbol: ER_CONN_UNIX_NO_FD; SQLSTATE: HY000

  Message: Can't start server: UNIX Socket : %s

  ER_CONN_UNIX_NO_FD was added in 8.0.2.

- Error number: MY-010270; Symbol: ER_CONN_UNIX_NO_BIND_NO_START; SQLSTATE: HY000

  Message: Can't start server : Bind on unix socket: %s

  ER_CONN_UNIX_NO_BIND_NO_START was added in 8.0.2.

- Error number: MY-010271; Symbol: ER_CONN_UNIX_LISTEN_FAILED; SQLSTATE: HY000

  Message: listen() on Unix socket failed with error %d

  ER_CONN_UNIX_LISTEN_FAILED was added in 8.0.2.

- Error number: MY-010272; Symbol: ER_CONN_UNIX_LOCK_FILE_GIVING_UP; SQLSTATE: HY000

  Message: Unable to create unix socket lock file %s after retries.

  ER_CONN_UNIX_LOCK_FILE_GIVING_UP was added in 8.0.2.

- Error number: MY-010273; Symbol: ER_CONN_UNIX_LOCK_FILE_CANT_CREATE; SQLSTATE: HY000

Message: Could not create unix socket lock file %s.

ER_CONN_UNIX_LOCK_FILE_CANT_CREATE was added in 8.0.2.

- Error number: MY-010274; Symbol: ER_CONN_UNIX_LOCK_FILE_CANT_OPEN; SQLSTATE: HY000

  Message: Could not open unix socket lock file %s.

  ER_CONN_UNIX_LOCK_FILE_CANT_OPEN was added in 8.0.2.

- Error number: MY-010275; Symbol: ER_CONN_UNIX_LOCK_FILE_CANT_READ; SQLSTATE: HY000

  Message: Could not read unix socket lock file %s.

  ER_CONN_UNIX_LOCK_FILE_CANT_READ was added in 8.0.2.

- Error number: MY-010276; Symbol: ER_CONN_UNIX_LOCK_FILE_EMPTY; SQLSTATE: HY000

  Message: Unix socket lock file is empty %s.

  ER_CONN_UNIX_LOCK_FILE_EMPTY was added in 8.0.2.

- Error number: MY-010277; Symbol: ER_CONN_UNIX_LOCK_FILE_PIDLESS; SQLSTATE: HY000

  Message: Invalid pid in unix socket lock file %s.

  ER_CONN_UNIX_LOCK_FILE_PIDLESS was added in 8.0.2.

- Error number: MY-010278; Symbol: ER_CONN_UNIX_LOCK_FILE_CANT_WRITE; SQLSTATE: HY000

  Message: Could not write unix socket lock file %s errno %d.

  ER_CONN_UNIX_LOCK_FILE_CANT_WRITE was added in 8.0.2.

- Error number: MY-010279; Symbol: ER_CONN_UNIX_LOCK_FILE_CANT_DELETE; SQLSTATE: HY000

  Message: Could not remove unix socket lock file %s errno %d.

  ER_CONN_UNIX_LOCK_FILE_CANT_DELETE was added in 8.0.2.

- Error number: MY-010280; Symbol: ER_CONN_UNIX_LOCK_FILE_CANT_SYNC; SQLSTATE: HY000

  Message: Could not sync unix socket lock file %s errno %d.

  ER_CONN_UNIX_LOCK_FILE_CANT_SYNC was added in 8.0.2.

- Error number: MY-010281; Symbol: ER_CONN_UNIX_LOCK_FILE_CANT_CLOSE; SQLSTATE: HY000

  Message: Could not close unix socket lock file %s errno %d.

  ER_CONN_UNIX_LOCK_FILE_CANT_CLOSE was added in 8.0.2.

- Error number: MY-010282; Symbol: ER_CONN_SOCKET_SELECT_FAILED; SQLSTATE: HY000

  Message: mysqld: Got error %d from select

  ER_CONN_SOCKET_SELECT_FAILED was added in 8.0.2.

- Error number: MY-010283; Symbol: ER_CONN_SOCKET_ACCEPT_FAILED; SQLSTATE: HY000

  Message: Error in accept: %s

  ER_CONN_SOCKET_ACCEPT_FAILED was added in 8.0.2.

- Error number: MY-010284; Symbol: ER_AUTH_RSA_CANT_FIND; SQLSTATE: HY000

  Message: RSA %s key file not found: %s. Some authentication plugins will not work.

  ER_AUTH_RSA_CANT_FIND was added in 8.0.2.

- Error number: MY-010285; Symbol: ER_AUTH_RSA_CANT_PARSE; SQLSTATE: HY000

  Message: Failure to parse RSA %s key (file exists): %s: %s

  ER_AUTH_RSA_CANT_PARSE was added in 8.0.2.

- Error number: MY-010286; Symbol: ER_AUTH_RSA_CANT_READ; SQLSTATE: HY000

  Message: Failure to read key file: %s

  ER_AUTH_RSA_CANT_READ was added in 8.0.2.

- Error number: MY-010287; Symbol: ER_AUTH_RSA_FILES_NOT_FOUND; SQLSTATE: HY000

  Message: RSA key files not found. Some authentication plugins will not work.

  ER_AUTH_RSA_FILES_NOT_FOUND was added in 8.0.2.

- Error number: MY-010288; Symbol: ER_CONN_ATTR_TRUNCATED; SQLSTATE: HY000

  Message: Connection attributes of length %lu were truncated (%d bytes lost) for connection %llu, user %s@%s (as %s), auth: %s

  ER_CONN_ATTR_TRUNCATED was added in 8.0.2.

- Error number: MY-010289; Symbol: ER_X509_CIPHERS_MISMATCH; SQLSTATE: HY000

  Message: X.509 ciphers mismatch: should be '%s' but is '%s'

  ER_X509_CIPHERS_MISMATCH was added in 8.0.2.

- Error number: MY-010290; Symbol: ER_X509_ISSUER_MISMATCH; SQLSTATE: HY000

  Message: X.509 issuer mismatch: should be '%s' but is '%s'

  ER_X509_ISSUER_MISMATCH was added in 8.0.2.

- Error number: MY-010291; Symbol: ER_X509_SUBJECT_MISMATCH; SQLSTATE: HY000

  Message: X.509 subject mismatch: should be '%s' but is '%s'

  ER_X509_SUBJECT_MISMATCH was added in 8.0.2.

- Error number: MY-010292; Symbol: ER_AUTH_CANT_ACTIVATE_ROLE; SQLSTATE: HY000

  Message: Failed to activate default role %s for %s

  ER_AUTH_CANT_ACTIVATE_ROLE was added in 8.0.2.

- Error number: MY-010293; Symbol: ER_X509_NEEDS_RSA_PRIVKEY; SQLSTATE: HY000

  Message: Could not generate RSA private key required for X.509 certificate.

  ER_X509_NEEDS_RSA_PRIVKEY was added in 8.0.2.
- Error number: MY-010294; Symbol: ER_X509_CANT_WRITE_KEY; SQLSTATE: HY000

  Message: Could not write key file: %s

  ER_X509_CANT_WRITE_KEY was added in 8.0.2.
- Error number: MY-010295; Symbol: ER_X509_CANT_CHMOD_KEY; SQLSTATE: HY000

  Message: Could not set file permission for %s

  ER_X509_CANT_CHMOD_KEY was added in 8.0.2.
- Error number: MY-010296; Symbol: ER_X509_CANT_READ_CA_KEY; SQLSTATE: HY000

  Message: Could not read CA key file: %s

  ER_X509_CANT_READ_CA_KEY was added in 8.0.2.
- Error number: MY-010297; Symbol: ER_X509_CANT_READ_CA_CERT; SQLSTATE: HY000

  Message: Could not read CA certificate file: %s

  ER_X509_CANT_READ_CA_CERT was added in 8.0.2.
- Error number: MY-010298; Symbol: ER_X509_CANT_CREATE_CERT; SQLSTATE: HY000

  Message: Could not generate X.509 certificate.

  ER_X509_CANT_CREATE_CERT was added in 8.0.2.
- Error number: MY-010299; Symbol: ER_X509_CANT_WRITE_CERT; SQLSTATE: HY000

  Message: Could not write certificate file: %s

  ER_X509_CANT_WRITE_CERT was added in 8.0.2.
- Error number: MY-010300; Symbol: ER_AUTH_CANT_CREATE_RSA_PAIR; SQLSTATE: HY000

  Message: Could not generate RSA Private/Public key pair

  ER_AUTH_CANT_CREATE_RSA_PAIR was added in 8.0.2.
- Error number: MY-010301; Symbol: ER_AUTH_CANT_WRITE_PRIVKEY; SQLSTATE: HY000

  Message: Could not write private key file: %s

  ER_AUTH_CANT_WRITE_PRIVKEY was added in 8.0.2.
- Error number: MY-010302; Symbol: ER_AUTH_CANT_WRITE_PUBKEY; SQLSTATE: HY000

  Message: Could not write public key file: %s

  ER_AUTH_CANT_WRITE_PUBKEY was added in 8.0.2.

- Error number: MY-010303; Symbol: ER_AUTH_SSL_CONF_PREVENTS_CERT_GENERATION; SQLSTATE: HY000

  Message: Skipping generation of SSL certificates as options related to SSL are specified.

  ER_AUTH_SSL_CONF_PREVENTS_CERT_GENERATION was added in 8.0.2.

- Error number: MY-010304; Symbol: ER_AUTH_USING_EXISTING_CERTS; SQLSTATE: HY000

  Message: Skipping generation of SSL certificates as certificate files are present in data directory.

  ER_AUTH_USING_EXISTING_CERTS was added in 8.0.2.

- Error number: MY-010305; Symbol: ER_AUTH_CERTS_SAVED_TO_DATADIR; SQLSTATE: HY000

  Message: Auto generated SSL certificates are placed in data directory.

  ER_AUTH_CERTS_SAVED_TO_DATADIR was added in 8.0.2.

- Error number: MY-010306; Symbol: ER_AUTH_CERT_GENERATION_DISABLED; SQLSTATE: HY000

  Message: Skipping generation of SSL certificates as --auto_generate_certs is set to OFF.

  ER_AUTH_CERT_GENERATION_DISABLED was added in 8.0.2.

- Error number: MY-010307; Symbol: ER_AUTH_RSA_CONF_PREVENTS_KEY_GENERATION; SQLSTATE: HY000

  Message: Skipping generation of RSA key pair through %s as options related to RSA keys are specified.

  ER_AUTH_RSA_CONF_PREVENTS_KEY_GENERATION was added in 8.0.2.

- Error number: MY-010308; Symbol: ER_AUTH_KEY_GENERATION_SKIPPED_PAIR_PRESENT; SQLSTATE: HY000

  Message: Skipping generation of RSA key pair through %s as key files are present in data directory.

  ER_AUTH_KEY_GENERATION_SKIPPED_PAIR_PRESENT was added in 8.0.2.

- Error number: MY-010309; Symbol: ER_AUTH_KEYS_SAVED_TO_DATADIR; SQLSTATE: HY000

  Message: Auto generated RSA key files through %s are placed in data directory.

  ER_AUTH_KEYS_SAVED_TO_DATADIR was added in 8.0.2.

- Error number: MY-010310; Symbol: ER_AUTH_KEY_GENERATION_DISABLED; SQLSTATE: HY000

  Message: Skipping generation of RSA key pair as %s is set to OFF.

  ER_AUTH_KEY_GENERATION_DISABLED was added in 8.0.2.

- Error number: MY-010311; Symbol: ER_AUTHCACHE_PROXIES_PRIV_SKIPPED_NEEDS_RESOLVE; SQLSTATE: HY000

  Message: 'proxies_priv' entry '%s@%s %s@%s' ignored in --skip-name-resolve mode.

  ER_AUTHCACHE_PROXIES_PRIV_SKIPPED_NEEDS_RESOLVE was added in 8.0.2.

- Error number: MY-010312; Symbol: ER_AUTHCACHE_PLUGIN_MISSING; SQLSTATE: HY000

Message: The plugin '%.*s' used to authenticate user '%s'@'%.*s' is not loaded. Nobody can currently login using this account.

ER_AUTHCACHE_PLUGIN_MISSING was added in 8.0.2.

- Error number: MY-010313; Symbol: ER_AUTHCACHE_PLUGIN_CONFIG; SQLSTATE: HY000

  Message: The plugin '%s' is used to authenticate user '%s'@'%.*s', %s configured. Nobody can currently login using this account.

  ER_AUTHCACHE_PLUGIN_CONFIG was added in 8.0.2.

- Error number: MY-010314; Symbol: ER_AUTHCACHE_ROLE_TABLES_DODGY; SQLSTATE: HY000

  Message: Could not load mysql.role_edges and mysql.default_roles tables. ACL DDLs will not work unless mysql_upgrade is executed.

  ER_AUTHCACHE_ROLE_TABLES_DODGY was added in 8.0.2, removed after 8.0.4.

- Error number: MY-010315; Symbol: ER_AUTHCACHE_USER_SKIPPED_NEEDS_RESOLVE; SQLSTATE: HY000

  Message: 'user' entry '%s@%s' ignored in --skip-name-resolve mode.

  ER_AUTHCACHE_USER_SKIPPED_NEEDS_RESOLVE was added in 8.0.2.

- Error number: MY-010316; Symbol: ER_AUTHCACHE_USER_TABLE_DODGY; SQLSTATE: HY000

  Message: Fatal error: Could not read the column 'authentication_string' from table 'mysql.user'. Please perform the MySQL upgrade procedure.

  ER_AUTHCACHE_USER_TABLE_DODGY was added in 8.0.2.

- Error number: MY-010317; Symbol: ER_AUTHCACHE_USER_IGNORED_DEPRECATED_PASSWORD; SQLSTATE: HY000

  Message: User entry '%s'@'%s' has a deprecated pre-4.1 password. The user will be ignored and no one can login with this user anymore.

  ER_AUTHCACHE_USER_IGNORED_DEPRECATED_PASSWORD was added in 8.0.2.

- Error number: MY-010318; Symbol: ER_AUTHCACHE_USER_IGNORED_NEEDS_PLUGIN; SQLSTATE: HY000

  Message: User entry '%s'@'%s' has an empty plugin value. The user will be ignored and no one can login with this user anymore.

  ER_AUTHCACHE_USER_IGNORED_NEEDS_PLUGIN was added in 8.0.2.

- Error number: MY-010319; Symbol: ER_AUTHCACHE_USER_IGNORED_INVALID_PASSWORD; SQLSTATE: HY000

  Message: Found invalid password for user: '%s@%s'; Ignoring user

  ER_AUTHCACHE_USER_IGNORED_INVALID_PASSWORD was added in 8.0.2.

- Error number: MY-010320; Symbol: ER_AUTHCACHE_EXPIRED_PASSWORD_UNSUPPORTED; SQLSTATE: HY000

Message: 'user' entry '%s@%s' has the password ignore flag raised, but its authentication plugin doesn't support password expiration. The user id will be ignored.

ER_AUTHCACHE_EXPIRED_PASSWORD_UNSUPPORTED was added in 8.0.2.

- Error number: MY-010321; Symbol: ER_NO_SUPER_WITHOUT_USER_PLUGIN; SQLSTATE: HY000

  Message: Some of the user accounts with SUPER privileges were disabled because of empty mysql.user.plugin value. If you are upgrading from MySQL 5.6 to MySQL 5.7 it means that substitution for the empty plugin column was not possible. Probably because of pre 4.1 password hash. If your account is disabled you will need to perform the MySQL upgrade procedure. For complete instructions on how to upgrade MySQL to a new version please see the 'Upgrading MySQL' section from the MySQL manual.

  ER_NO_SUPER_WITHOUT_USER_PLUGIN was added in 8.0.2.

- Error number: MY-010322; Symbol: ER_AUTHCACHE_DB_IGNORED_EMPTY_NAME; SQLSTATE: HY000

  Message: Found an entry in the 'db' table with empty database name; Skipped

  ER_AUTHCACHE_DB_IGNORED_EMPTY_NAME was added in 8.0.2.

- Error number: MY-010323; Symbol: ER_AUTHCACHE_DB_SKIPPED_NEEDS_RESOLVE; SQLSTATE: HY000

  Message: 'db' entry '%s %s@%s' ignored in --skip-name-resolve mode.

  ER_AUTHCACHE_DB_SKIPPED_NEEDS_RESOLVE was added in 8.0.2.

- Error number: MY-010324; Symbol: ER_AUTHCACHE_DB_ENTRY_LOWERCASED_REVOKE_WILL_FAIL; SQLSTATE: HY000

  Message: 'db' entry '%s %s@%s' had database in mixed case that has been forced to lowercase because lower_case_table_names is set. It will not be possible to remove this privilege using REVOKE.

  ER_AUTHCACHE_DB_ENTRY_LOWERCASED_REVOKE_WILL_FAIL was added in 8.0.2.

- Error number: MY-010325; Symbol: ER_AUTHCACHE_TABLE_PROXIES_PRIV_MISSING; SQLSTATE: HY000

  Message: The system table mysql.proxies_priv is missing. Please perform the MySQL upgrade procedure.

  ER_AUTHCACHE_TABLE_PROXIES_PRIV_MISSING was added in 8.0.2.

- Error number: MY-010326; Symbol: ER_AUTHCACHE_CANT_OPEN_AND_LOCK_PRIVILEGE_TABLES; SQLSTATE: HY000

  Message: Fatal error: Can't open and lock privilege tables: %s

  ER_AUTHCACHE_CANT_OPEN_AND_LOCK_PRIVILEGE_TABLES was added in 8.0.2.

- Error number: MY-010327; Symbol: ER_AUTHCACHE_CANT_INIT_GRANT_SUBSYSTEM; SQLSTATE: HY000

  Message: Fatal: can't initialize grant subsystem - '%s'

ER_AUTHCACHE_CANT_INIT_GRANT_SUBSYSTEM was added in 8.0.2.

- Error number: MY-010328; Symbol: ER_AUTHCACHE_PROCS_PRIV_SKIPPED_NEEDS_RESOLVE; SQLSTATE: HY000

  Message: 'procs_priv' entry '%s %s@%s' ignored in --skip-name-resolve mode.

  ER_AUTHCACHE_PROCS_PRIV_SKIPPED_NEEDS_RESOLVE was added in 8.0.2.

- Error number: MY-010329; Symbol: ER_AUTHCACHE_PROCS_PRIV_ENTRY_IGNORED_BAD_ROUTINE_TYPE; SQLSTATE: HY000

  Message: 'procs_priv' entry '%s' ignored, bad routine type

  ER_AUTHCACHE_PROCS_PRIV_ENTRY_IGNORED_BAD_ROUTINE_TYPE was added in 8.0.2.

- Error number: MY-010330; Symbol: ER_AUTHCACHE_TABLES_PRIV_SKIPPED_NEEDS_RESOLVE; SQLSTATE: HY000

  Message: 'tables_priv' entry '%s %s@%s' ignored in --skip-name-resolve mode.

  ER_AUTHCACHE_TABLES_PRIV_SKIPPED_NEEDS_RESOLVE was added in 8.0.2.

- Error number: MY-010331; Symbol: ER_USER_NOT_IN_EXTRA_USERS_BINLOG_POSSIBLY_INCOMPLETE; SQLSTATE: HY000

  Message: Failed to add %s in extra_users. Binary log entry may miss some of the users.

  ER_USER_NOT_IN_EXTRA_USERS_BINLOG_POSSIBLY_INCOMPLETE was added in 8.0.2.

- Error number: MY-010332; Symbol: ER_DD_SCHEMA_NOT_FOUND; SQLSTATE: HY000

  Message: Unable to start server. The data dictionary schema '%s' does not exist.

  ER_DD_SCHEMA_NOT_FOUND was added in 8.0.2.

- Error number: MY-010333; Symbol: ER_DD_TABLE_NOT_FOUND; SQLSTATE: HY000

  Message: Unable to start server. The data dictionary table '%s' does not exist.

  ER_DD_TABLE_NOT_FOUND was added in 8.0.2.

- Error number: MY-010334; Symbol: ER_DD_SE_INIT_FAILED; SQLSTATE: HY000

  Message: Failed to initialize DD Storage Engine

  ER_DD_SE_INIT_FAILED was added in 8.0.2.

- Error number: MY-010335; Symbol: ER_DD_ABORTING_PARTIAL_UPGRADE; SQLSTATE: HY000

  Message: Found partially upgraded DD. Aborting upgrade and deleting all DD tables. Start the upgrade process again.

  ER_DD_ABORTING_PARTIAL_UPGRADE was added in 8.0.2.

- Error number: MY-010336; Symbol: ER_DD_FRM_EXISTS_FOR_TABLE; SQLSTATE: HY000

  Message: Found .frm file with same name as one of the Dictionary Tables.

ER_DD_FRM_EXISTS_FOR_TABLE was added in 8.0.2.

- Error number: MY-010337; Symbol: ER_DD_CREATED_FOR_UPGRADE; SQLSTATE: HY000

  Message: Created Data Dictionary for upgrade

  ER_DD_CREATED_FOR_UPGRADE was added in 8.0.2.

- Error number: MY-010338; Symbol: ER_ERRMSG_CANT_FIND_FILE; SQLSTATE: HY000

  Message: Can't find error-message file '%s'. Check error-message file location and 'lc-messages-dir' configuration directive.

  ER_ERRMSG_CANT_FIND_FILE was added in 8.0.2.

- Error number: MY-010339; Symbol: ER_ERRMSG_LOADING_55_STYLE; SQLSTATE: HY000

  Message: Using pre 5.5 semantics to load error messages from %s. If this is not intended, refer to the documentation for valid usage of --lc-messages-dir and --language parameters.

  ER_ERRMSG_LOADING_55_STYLE was added in 8.0.2.

- Error number: MY-010340; Symbol: ER_ERRMSG_MISSING_IN_FILE; SQLSTATE: HY000

  Message: Error message file '%s' had only %d error messages, but it should contain at least %d error messages. Check that the above file is the right version for this program!

  ER_ERRMSG_MISSING_IN_FILE was added in 8.0.2.

- Error number: MY-010341; Symbol: ER_ERRMSG_OOM; SQLSTATE: HY000

  Message: Not enough memory for messagefile '%s'

  ER_ERRMSG_OOM was added in 8.0.2.

- Error number: MY-010342; Symbol: ER_ERRMSG_CANT_READ; SQLSTATE: HY000

  Message: Can't read from messagefile '%s'

  ER_ERRMSG_CANT_READ was added in 8.0.2.

- Error number: MY-010343; Symbol: ER_TABLE_INCOMPATIBLE_DECIMAL_FIELD; SQLSTATE: HY000

  Message: Found incompatible DECIMAL field '%s' in %s; Please do "ALTER TABLE `%s` FORCE" to fix it!

  ER_TABLE_INCOMPATIBLE_DECIMAL_FIELD was added in 8.0.2.

- Error number: MY-010344; Symbol: ER_TABLE_INCOMPATIBLE_YEAR_FIELD; SQLSTATE: HY000

  Message: Found incompatible YEAR(x) field '%s' in %s; Please do "ALTER TABLE `%s` FORCE" to fix it!

  ER_TABLE_INCOMPATIBLE_YEAR_FIELD was added in 8.0.2.

- Error number: MY-010345; Symbol: ER_INVALID_CHARSET_AND_DEFAULT_IS_MB; SQLSTATE: HY000

Message: '%s' had no or invalid character set, and default character set is multi-byte, so character column sizes may have changed

ER_INVALID_CHARSET_AND_DEFAULT_IS_MB was added in 8.0.2.

- Error number: MY-010346; Symbol: ER_TABLE_WRONG_KEY_DEFINITION; SQLSTATE: HY000

  Message: Found wrong key definition in %s; Please do "ALTER TABLE `%s` FORCE " to fix it!

  ER_TABLE_WRONG_KEY_DEFINITION was added in 8.0.2.

- Error number: MY-010347; Symbol: ER_CANT_OPEN_FRM_FILE; SQLSTATE: HY000

  Message: Unable to open file %s

  ER_CANT_OPEN_FRM_FILE was added in 8.0.2.

- Error number: MY-010348; Symbol: ER_CANT_READ_FRM_FILE; SQLSTATE: HY000

  Message: Error in reading file %s

  ER_CANT_READ_FRM_FILE was added in 8.0.2.

- Error number: MY-010349; Symbol: ER_TABLE_CREATED_WITH_DIFFERENT_VERSION; SQLSTATE: HY000

  Message: Table '%s' was created with a different version of MySQL and cannot be read

  ER_TABLE_CREATED_WITH_DIFFERENT_VERSION was added in 8.0.2.

- Error number: MY-010350; Symbol: ER_VIEW_UNPARSABLE; SQLSTATE: HY000

  Message: Unable to read view %s

  ER_VIEW_UNPARSABLE was added in 8.0.2.

- Error number: MY-010351; Symbol: ER_FILE_TYPE_UNKNOWN; SQLSTATE: HY000

  Message: File %s has unknown type in its header.

  ER_FILE_TYPE_UNKNOWN was added in 8.0.2.

- Error number: MY-010352; Symbol: ER_INVALID_INFO_IN_FRM; SQLSTATE: HY000

  Message: Incorrect information in file %s

  ER_INVALID_INFO_IN_FRM was added in 8.0.2.

- Error number: MY-010353; Symbol: ER_CANT_OPEN_AND_LOCK_PRIVILEGE_TABLES; SQLSTATE: HY000

  Message: Can't open and lock privilege tables: %s

  ER_CANT_OPEN_AND_LOCK_PRIVILEGE_TABLES was added in 8.0.2.

- Error number: MY-010354; Symbol: ER_AUDIT_PLUGIN_DOES_NOT_SUPPORT_AUDIT_AUTH_EVENTS; SQLSTATE: HY000

Message: Plugin '%s' cannot subscribe to MYSQL_AUDIT_AUTHORIZATION events. Currently not supported.

ER_AUDIT_PLUGIN_DOES_NOT_SUPPORT_AUDIT_AUTH_EVENTS was added in 8.0.2.

- Error number: MY-010355; Symbol: ER_AUDIT_PLUGIN_HAS_INVALID_DATA; SQLSTATE: HY000

  Message: Plugin '%s' has invalid data.

  ER_AUDIT_PLUGIN_HAS_INVALID_DATA was added in 8.0.2.

- Error number: MY-010356; Symbol: ER_TZ_OOM_INITIALIZING_TIME_ZONES; SQLSTATE: HY000

  Message: Fatal error: OOM while initializing time zones

  ER_TZ_OOM_INITIALIZING_TIME_ZONES was added in 8.0.2.

- Error number: MY-010357; Symbol: ER_TZ_CANT_OPEN_AND_LOCK_TIME_ZONE_TABLE; SQLSTATE: HY000

  Message: Can't open and lock time zone table: %s trying to live without them

  ER_TZ_CANT_OPEN_AND_LOCK_TIME_ZONE_TABLE was added in 8.0.2.

- Error number: MY-010358; Symbol: ER_TZ_OOM_LOADING_LEAP_SECOND_TABLE; SQLSTATE: HY000

  Message: Fatal error: Out of memory while loading mysql.time_zone_leap_second table

  ER_TZ_OOM_LOADING_LEAP_SECOND_TABLE was added in 8.0.2.

- Error number: MY-010359; Symbol: ER_TZ_TOO_MANY_LEAPS_IN_LEAP_SECOND_TABLE; SQLSTATE: HY000

  Message: Fatal error: While loading mysql.time_zone_leap_second table: too much leaps

  ER_TZ_TOO_MANY_LEAPS_IN_LEAP_SECOND_TABLE was added in 8.0.2.

- Error number: MY-010360; Symbol: ER_TZ_ERROR_LOADING_LEAP_SECOND_TABLE; SQLSTATE: HY000

  Message: Fatal error: Error while loading mysql.time_zone_leap_second table

  ER_TZ_ERROR_LOADING_LEAP_SECOND_TABLE was added in 8.0.2.

- Error number: MY-010361; Symbol: ER_TZ_UNKNOWN_OR_ILLEGAL_DEFAULT_TIME_ZONE; SQLSTATE: HY000

  Message: Fatal error: Illegal or unknown default time zone '%s'

  ER_TZ_UNKNOWN_OR_ILLEGAL_DEFAULT_TIME_ZONE was added in 8.0.2.

- Error number: MY-010362; Symbol: ER_TZ_CANT_FIND_DESCRIPTION_FOR_TIME_ZONE; SQLSTATE: HY000

  Message: Can't find description of time zone '%.*s'

  ER_TZ_CANT_FIND_DESCRIPTION_FOR_TIME_ZONE was added in 8.0.2.

- Error number: MY-010363; Symbol: ER_TZ_CANT_FIND_DESCRIPTION_FOR_TIME_ZONE_ID; SQLSTATE: HY000

  Message: Can't find description of time zone '%u'

  ER_TZ_CANT_FIND_DESCRIPTION_FOR_TIME_ZONE_ID was added in 8.0.2.

- Error number: MY-010364; Symbol: ER_TZ_TRANSITION_TYPE_TABLE_TYPE_TOO_LARGE; SQLSTATE: HY000

  Message: Error while loading time zone description from mysql.time_zone_transition_type table: too big transition type id

  ER_TZ_TRANSITION_TYPE_TABLE_TYPE_TOO_LARGE was added in 8.0.2.

- Error number: MY-010365; Symbol: ER_TZ_TRANSITION_TYPE_TABLE_ABBREVIATIONS_EXCEED_SPACE; SQLSTATE: HY000

  Message: Error while loading time zone description from mysql.time_zone_transition_type table: not enough room for abbreviations

  ER_TZ_TRANSITION_TYPE_TABLE_ABBREVIATIONS_EXCEED_SPACE was added in 8.0.2.

- Error number: MY-010366; Symbol: ER_TZ_TRANSITION_TYPE_TABLE_LOAD_ERROR; SQLSTATE: HY000

  Message: Error while loading time zone description from mysql.time_zone_transition_type table

  ER_TZ_TRANSITION_TYPE_TABLE_LOAD_ERROR was added in 8.0.2.

- Error number: MY-010367; Symbol: ER_TZ_TRANSITION_TABLE_TOO_MANY_TRANSITIONS; SQLSTATE: HY000

  Message: Error while loading time zone description from mysql.time_zone_transition table: too much transitions

  ER_TZ_TRANSITION_TABLE_TOO_MANY_TRANSITIONS was added in 8.0.2.

- Error number: MY-010368; Symbol: ER_TZ_TRANSITION_TABLE_BAD_TRANSITION_TYPE; SQLSTATE: HY000

  Message: Error while loading time zone description from mysql.time_zone_transition table: bad transition type id

  ER_TZ_TRANSITION_TABLE_BAD_TRANSITION_TYPE was added in 8.0.2.

- Error number: MY-010369; Symbol: ER_TZ_TRANSITION_TABLE_LOAD_ERROR; SQLSTATE: HY000

  Message: Error while loading time zone description from mysql.time_zone_transition table

  ER_TZ_TRANSITION_TABLE_LOAD_ERROR was added in 8.0.2.

- Error number: MY-010370; Symbol: ER_TZ_NO_TRANSITION_TYPES_IN_TIME_ZONE; SQLSTATE: HY000

  Message: loading time zone without transition types

  ER_TZ_NO_TRANSITION_TYPES_IN_TIME_ZONE was added in 8.0.2.

- Error number: MY-010371; Symbol: ER_TZ_OOM_LOADING_TIME_ZONE_DESCRIPTION; SQLSTATE: HY000

  Message: Out of memory while loading time zone description

  ER_TZ_OOM_LOADING_TIME_ZONE_DESCRIPTION was added in 8.0.2.

- Error number: MY-010372; Symbol: ER_TZ_CANT_BUILD_MKTIME_MAP; SQLSTATE: HY000

  Message: Unable to build mktime map for time zone

  ER_TZ_CANT_BUILD_MKTIME_MAP was added in 8.0.2.

- Error number: MY-010373; Symbol: ER_TZ_OOM_WHILE_LOADING_TIME_ZONE; SQLSTATE: HY000

  Message: Out of memory while loading time zone

  ER_TZ_OOM_WHILE_LOADING_TIME_ZONE was added in 8.0.2.

- Error number: MY-010374; Symbol: ER_TZ_OOM_WHILE_SETTING_TIME_ZONE; SQLSTATE: HY000

  Message: Fatal error: Out of memory while setting new time zone

  ER_TZ_OOM_WHILE_SETTING_TIME_ZONE was added in 8.0.2.

- Error number: MY-010375; Symbol: ER_SLAVE_SQL_THREAD_STOPPED_UNTIL_CONDITION_BAD; SQLSTATE: HY000

  Message: Slave SQL thread is stopped because UNTIL condition is bad(%s:%llu).

  ER_SLAVE_SQL_THREAD_STOPPED_UNTIL_CONDITION_BAD was added in 8.0.2.

- Error number: MY-010376; Symbol: ER_SLAVE_SQL_THREAD_STOPPED_UNTIL_POSITION_REACHED; SQLSTATE: HY000

  Message: Slave SQL thread stopped because it reached its UNTIL position %llu

  ER_SLAVE_SQL_THREAD_STOPPED_UNTIL_POSITION_REACHED was added in 8.0.2.

- Error number: MY-010377; Symbol: ER_SLAVE_SQL_THREAD_STOPPED_BEFORE_GTIDS_ALREADY_APPLIED; SQLSTATE: HY000

  Message: Slave SQL thread stopped because UNTIL SQL_BEFORE_GTIDS %s is already applied

  ER_SLAVE_SQL_THREAD_STOPPED_BEFORE_GTIDS_ALREADY_APPLIED was added in 8.0.2.

- Error number: MY-010378; Symbol: ER_SLAVE_SQL_THREAD_STOPPED_BEFORE_GTIDS_REACHED; SQLSTATE: HY000

  Message: Slave SQL thread stopped because it reached UNTIL SQL_BEFORE_GTIDS %s

  ER_SLAVE_SQL_THREAD_STOPPED_BEFORE_GTIDS_REACHED was added in 8.0.2.

- Error number: MY-010379; Symbol: ER_SLAVE_SQL_THREAD_STOPPED_AFTER_GTIDS_REACHED; SQLSTATE: HY000

  Message: Slave SQL thread stopped because it reached UNTIL SQL_AFTER_GTIDS %s

  ER_SLAVE_SQL_THREAD_STOPPED_AFTER_GTIDS_REACHED was added in 8.0.2.

- Error number: MY-010380; Symbol: ER_SLAVE_SQL_THREAD_STOPPED_GAP_TRX_PROCESSED; SQLSTATE: HY000

  Message: Slave SQL thread stopped according to UNTIL SQL_AFTER_MTS_GAPS as it has processed all gap transactions left from the previous slave session.

  ER_SLAVE_SQL_THREAD_STOPPED_GAP_TRX_PROCESSED was added in 8.0.2.

- Error number: MY-010381; Symbol: ER_GROUP_REPLICATION_PLUGIN_NOT_INSTALLED; SQLSTATE: HY000

  Message: Group Replication plugin is not installed.

  ER_GROUP_REPLICATION_PLUGIN_NOT_INSTALLED was added in 8.0.2.

- Error number: MY-010382; Symbol: ER_GTID_ALREADY_ADDED_BY_USER; SQLSTATE: HY000

  Message: The transaction owned GTID is already in the %s table, which is caused by an explicit modifying from user client.

  ER_GTID_ALREADY_ADDED_BY_USER was added in 8.0.2.

- Error number: MY-010383; Symbol: ER_FAILED_TO_DELETE_FROM_GTID_EXECUTED_TABLE; SQLSTATE: HY000

  Message: Failed to delete the row: '%s' from the gtid_executed table.

  ER_FAILED_TO_DELETE_FROM_GTID_EXECUTED_TABLE was added in 8.0.2.

- Error number: MY-010384; Symbol: ER_FAILED_TO_COMPRESS_GTID_EXECUTED_TABLE; SQLSTATE: HY000

  Message: Failed to compress the gtid_executed table.

  ER_FAILED_TO_COMPRESS_GTID_EXECUTED_TABLE was added in 8.0.2.

- Error number: MY-010385; Symbol: ER_FAILED_TO_COMPRESS_GTID_EXECUTED_TABLE_OOM; SQLSTATE: HY000

  Message: Failed to compress the gtid_executed table, because it is failed to allocate the THD.

  ER_FAILED_TO_COMPRESS_GTID_EXECUTED_TABLE_OOM was added in 8.0.2.

- Error number: MY-010386; Symbol: ER_FAILED_TO_INIT_THREAD_ATTR_FOR_GTID_TABLE_COMPRESSION; SQLSTATE: HY000

  Message: Failed to initialize thread attribute when creating compression thread.

  ER_FAILED_TO_INIT_THREAD_ATTR_FOR_GTID_TABLE_COMPRESSION was added in 8.0.2.

- Error number: MY-010387; Symbol: ER_FAILED_TO_CREATE_GTID_TABLE_COMPRESSION_THREAD; SQLSTATE: HY000

  Message: Can not create thread to compress gtid_executed table (errno= %d)

  ER_FAILED_TO_CREATE_GTID_TABLE_COMPRESSION_THREAD was added in 8.0.2.

- Error number: MY-010388; Symbol: ER_FAILED_TO_JOIN_GTID_TABLE_COMPRESSION_THREAD; SQLSTATE: HY000

Message: Could not join gtid_executed table compression thread. error:%d

ER_FAILED_TO_JOIN_GTID_TABLE_COMPRESSION_THREAD was added in 8.0.2.

- Error number: MY-010389; Symbol: ER_NPIPE_FAILED_TO_INIT_SECURITY_DESCRIPTOR; SQLSTATE: HY000

  Message: Can't start server : Initialize security descriptor: %s

  ER_NPIPE_FAILED_TO_INIT_SECURITY_DESCRIPTOR was added in 8.0.2.

- Error number: MY-010390; Symbol: ER_NPIPE_FAILED_TO_SET_SECURITY_DESCRIPTOR; SQLSTATE: HY000

  Message: Can't start server : Set security descriptor: %s

  ER_NPIPE_FAILED_TO_SET_SECURITY_DESCRIPTOR was added in 8.0.2.

- Error number: MY-010391; Symbol: ER_NPIPE_PIPE_ALREADY_IN_USE; SQLSTATE: HY000

  Message: Can't start server : Named Pipe "%s" already in use.

  ER_NPIPE_PIPE_ALREADY_IN_USE was added in 8.0.2.

- Error number: MY-010392; Symbol: ER_NDB_SLAVE_SAW_EPOCH_LOWER_THAN_PREVIOUS_ON_START; SQLSTATE: HY000

  Message: NDB Slave : At SQL thread start applying epoch %llu/%llu (%llu) from Master ServerId %u which is lower than previously applied epoch %llu/%llu (%llu). Group Master Log : %s Group Master Log Pos : %llu. Check slave positioning.

  ER_NDB_SLAVE_SAW_EPOCH_LOWER_THAN_PREVIOUS_ON_START was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010393; Symbol: ER_NDB_SLAVE_SAW_EPOCH_LOWER_THAN_PREVIOUS; SQLSTATE: HY000

  Message: NDB Slave : SQL thread stopped as applying epoch %llu/%llu (%llu) from Master ServerId %u which is lower than previously applied epoch %llu/%llu (%llu). Group Master Log : %s Group Master Log Pos : %llu

  ER_NDB_SLAVE_SAW_EPOCH_LOWER_THAN_PREVIOUS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010394; Symbol: ER_NDB_SLAVE_SAW_ALREADY_COMMITTED_EPOCH; SQLSTATE: HY000

  Message: NDB Slave : SQL thread stopped as attempted to reapply already committed epoch %llu/%llu (%llu) from server id %u. Group Master Log : %s Group Master Log Pos : %llu.

  ER_NDB_SLAVE_SAW_ALREADY_COMMITTED_EPOCH was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010395; Symbol: ER_NDB_SLAVE_PREVIOUS_EPOCH_NOT_COMMITTED; SQLSTATE: HY000

  Message: NDB Slave : SQL thread stopped as attempting to apply new epoch %llu/%llu (%llu) while lower received epoch %llu/%llu (%llu) has not been committed. Master server id : %u. Group Master Log : %s Group Master Log Pos : %llu.

ER_NDB_SLAVE_PREVIOUS_EPOCH_NOT_COMMITTED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010396; Symbol: ER_NDB_SLAVE_MISSING_DATA_FOR_TIMESTAMP_COLUMN; SQLSTATE: HY000

  Message: NDB Slave: missing data for %s timestamp column %u.

  ER_NDB_SLAVE_MISSING_DATA_FOR_TIMESTAMP_COLUMN was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010397; Symbol: ER_NDB_SLAVE_LOGGING_EXCEPTIONS_TO; SQLSTATE: HY000

  Message: NDB Slave: Table %s.%s logging exceptions to %s.%s

  ER_NDB_SLAVE_LOGGING_EXCEPTIONS_TO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010398; Symbol: ER_NDB_SLAVE_LOW_EPOCH_RESOLUTION; SQLSTATE: HY000

  Message: NDB Slave: Table %s.%s : %s, low epoch resolution

  ER_NDB_SLAVE_LOW_EPOCH_RESOLUTION was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010399; Symbol: ER_NDB_INFO_FOUND_UNEXPECTED_FIELD_TYPE; SQLSTATE: HY000

  Message: Found unexpected field type %u

  ER_NDB_INFO_FOUND_UNEXPECTED_FIELD_TYPE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010400; Symbol: ER_NDB_INFO_FAILED_TO_CREATE_NDBINFO; SQLSTATE: HY000

  Message: Failed to create NdbInfo

  ER_NDB_INFO_FAILED_TO_CREATE_NDBINFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010401; Symbol: ER_NDB_INFO_FAILED_TO_INIT_NDBINFO; SQLSTATE: HY000

  Message: Failed to init NdbInfo

  ER_NDB_INFO_FAILED_TO_INIT_NDBINFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010402; Symbol: ER_NDB_CLUSTER_WRONG_NUMBER_OF_FUNCTION_ARGUMENTS; SQLSTATE: HY000

  Message: ndb_serialize_cond: Unexpected mismatch of found and expected number of function arguments %u

  ER_NDB_CLUSTER_WRONG_NUMBER_OF_FUNCTION_ARGUMENTS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010403; Symbol: ER_NDB_CLUSTER_SCHEMA_INFO; SQLSTATE: HY000

  Message: %s - %s.%s

  ER_NDB_CLUSTER_SCHEMA_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010404; Symbol: ER_NDB_CLUSTER_GENERIC_MESSAGE; SQLSTATE: HY000

  Message: %s

  ER_NDB_CLUSTER_GENERIC_MESSAGE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010405; Symbol: ER_RPL_CANT_OPEN_INFO_TABLE; SQLSTATE: HY000

  Message: Info table is not ready to be used. Table '%s.%s' cannot be opened.

  ER_RPL_CANT_OPEN_INFO_TABLE was added in 8.0.2.

- Error number: MY-010406; Symbol: ER_RPL_CANT_SCAN_INFO_TABLE; SQLSTATE: HY000

  Message: Info table is not ready to be used. Table '%s.%s' cannot be scanned.

  ER_RPL_CANT_SCAN_INFO_TABLE was added in 8.0.2.

- Error number: MY-010407; Symbol: ER_RPL_CORRUPTED_INFO_TABLE; SQLSTATE: HY000

  Message: Corrupted table %s.%s. Check out table definition.

  ER_RPL_CORRUPTED_INFO_TABLE was added in 8.0.2.

- Error number: MY-010408; Symbol: ER_RPL_CORRUPTED_KEYS_IN_INFO_TABLE; SQLSTATE: HY000

  Message: Info table has a problem with its key field(s). Table '%s.%s' expected field #%u to be '%s' but found '%s' instead.

  ER_RPL_CORRUPTED_KEYS_IN_INFO_TABLE was added in 8.0.2.

- Error number: MY-010409; Symbol: ER_RPL_WORKER_ID_IS; SQLSTATE: HY000

  Message: Choosing worker id %lu, the following is going to be %lu

  ER_RPL_WORKER_ID_IS was added in 8.0.2.

- Error number: MY-010410; Symbol: ER_RPL_INCONSISTENT_TIMESTAMPS_IN_TRX; SQLSTATE: HY000

  Message: Transaction is tagged with inconsistent logical timestamps: sequence_number (%lld) <= last_committed (%lld)

  ER_RPL_INCONSISTENT_TIMESTAMPS_IN_TRX was added in 8.0.2.

- Error number: MY-010411; Symbol: ER_RPL_INCONSISTENT_SEQUENCE_NO_IN_TRX; SQLSTATE: HY000

  Message: Transaction's sequence number is inconsistent with that of a preceding one: sequence_number (%lld) <= previous sequence_number (%lld)

  ER_RPL_INCONSISTENT_SEQUENCE_NO_IN_TRX was added in 8.0.2.

- Error number: MY-010412; Symbol: ER_RPL_CHANNELS_REQUIRE_TABLES_AS_INFO_REPOSITORIES; SQLSTATE: HY000

  Message: For the creation of replication channels the master info and relay log info repositories must be set to TABLE

ER_RPL_CHANNELS_REQUIRE_TABLES_AS_INFO_REPOSITORIES was added in 8.0.2.

- Error number: MY-010413; Symbol: ER_RPL_CHANNELS_REQUIRE_NON_ZERO_SERVER_ID; SQLSTATE: HY000

  Message: For the creation of replication channels the server id must be different from 0

  ER_RPL_CHANNELS_REQUIRE_NON_ZERO_SERVER_ID was added in 8.0.2.

- Error number: MY-010414; Symbol: ER_RPL_REPO_SHOULD_BE_TABLE; SQLSTATE: HY000

  Message: Slave: Wrong repository. Repository should be TABLE

  ER_RPL_REPO_SHOULD_BE_TABLE was added in 8.0.2.

- Error number: MY-010415; Symbol: ER_RPL_ERROR_CREATING_MASTER_INFO; SQLSTATE: HY000

  Message: Error creating master info: %s.

  ER_RPL_ERROR_CREATING_MASTER_INFO was added in 8.0.2.

- Error number: MY-010416; Symbol: ER_RPL_ERROR_CHANGING_MASTER_INFO_REPO_TYPE; SQLSTATE: HY000

  Message: Error changing the type of master info's repository: %s.

  ER_RPL_ERROR_CHANGING_MASTER_INFO_REPO_TYPE was added in 8.0.2.

- Error number: MY-010417; Symbol: ER_RPL_CHANGING_RELAY_LOG_INFO_REPO_TYPE_FAILED_DUE_TO_GAPS; SQLSTATE: HY000

  Message: It is not possible to change the type of the relay log repository because there are workers repositories with possible execution gaps. The value of --relay_log_info_repository is altered to one of the found Worker repositories. The gaps have to be sorted out before resuming with the type change.

  ER_RPL_CHANGING_RELAY_LOG_INFO_REPO_TYPE_FAILED_DUE_TO_GAPS was added in 8.0.2.

- Error number: MY-010418; Symbol: ER_RPL_ERROR_CREATING_RELAY_LOG_INFO; SQLSTATE: HY000

  Message: Error creating relay log info: %s.

  ER_RPL_ERROR_CREATING_RELAY_LOG_INFO was added in 8.0.2.

- Error number: MY-010419; Symbol: ER_RPL_ERROR_CHANGING_RELAY_LOG_INFO_REPO_TYPE; SQLSTATE: HY000

  Message: Error changing the type of relay log info's repository: %s.

  ER_RPL_ERROR_CHANGING_RELAY_LOG_INFO_REPO_TYPE was added in 8.0.2.

- Error number: MY-010420; Symbol: ER_RPL_FAILED_TO_DELETE_FROM_SLAVE_WORKERS_INFO_REPOSITORY; SQLSTATE: HY000

  Message: Could not delete from Slave Workers info repository.

  ER_RPL_FAILED_TO_DELETE_FROM_SLAVE_WORKERS_INFO_REPOSITORY was added in 8.0.2.

- Error number: MY-010421; Symbol: ER_RPL_FAILED_TO_RESET_STATE_IN_SLAVE_INFO_REPOSITORY; SQLSTATE: HY000

  Message: Could not store the reset Slave Worker state into the slave info repository.

  ER_RPL_FAILED_TO_RESET_STATE_IN_SLAVE_INFO_REPOSITORY was added in 8.0.2.

- Error number: MY-010422; Symbol: ER_RPL_ERROR_CHECKING_REPOSITORY; SQLSTATE: HY000

  Message: Error in checking %s repository info type of %s.

  ER_RPL_ERROR_CHECKING_REPOSITORY was added in 8.0.2.

- Error number: MY-010423; Symbol: ER_RPL_SLAVE_GENERIC_MESSAGE; SQLSTATE: HY000

  Message: Slave: %s

  ER_RPL_SLAVE_GENERIC_MESSAGE was added in 8.0.2.

- Error number: MY-010424; Symbol: ER_RPL_SLAVE_COULD_NOT_CREATE_CHANNEL_LIST; SQLSTATE: HY000

  Message: Slave: Could not create channel list

  ER_RPL_SLAVE_COULD_NOT_CREATE_CHANNEL_LIST was added in 8.0.2.

- Error number: MY-010425; Symbol: ER_RPL_MULTISOURCE_REQUIRES_TABLE_TYPE_REPOSITORIES; SQLSTATE: HY000

  Message: Slave: This slave was a multisourced slave previously which is supported only by both TABLE based master info and relay log info repositories. Found one or both of the info repos to be type FILE. Set both repos to type TABLE.

  ER_RPL_MULTISOURCE_REQUIRES_TABLE_TYPE_REPOSITORIES was added in 8.0.2.

- Error number: MY-010426; Symbol: ER_RPL_SLAVE_FAILED_TO_INIT_A_MASTER_INFO_STRUCTURE; SQLSTATE: HY000

  Message: Slave: Failed to initialize the master info structure for channel '%s'; its record may still be present in 'mysql.slave_master_info' table, consider deleting it.

  ER_RPL_SLAVE_FAILED_TO_INIT_A_MASTER_INFO_STRUCTURE was added in 8.0.2.

- Error number: MY-010427; Symbol: ER_RPL_SLAVE_FAILED_TO_INIT_MASTER_INFO_STRUCTURE; SQLSTATE: HY000

  Message: Failed to initialize the master info structure%s

  ER_RPL_SLAVE_FAILED_TO_INIT_MASTER_INFO_STRUCTURE was added in 8.0.2.

- Error number: MY-010428; Symbol: ER_RPL_SLAVE_FAILED_TO_CREATE_CHANNEL_FROM_MASTER_INFO; SQLSTATE: HY000

  Message: Slave: Failed to create a channel from master info table repository.

  ER_RPL_SLAVE_FAILED_TO_CREATE_CHANNEL_FROM_MASTER_INFO was added in 8.0.2.

- Error number: MY-010429; Symbol: ER_RPL_FAILED_TO_CREATE_NEW_INFO_FILE; SQLSTATE: HY000

Message: Failed to create a new info file (file '%s', errno %d)

ER_RPL_FAILED_TO_CREATE_NEW_INFO_FILE was added in 8.0.2.

- Error number: MY-010430; Symbol: ER_RPL_FAILED_TO_CREATE_CACHE_FOR_INFO_FILE; SQLSTATE: HY000

  Message: Failed to create a cache on info file (file '%s')

  ER_RPL_FAILED_TO_CREATE_CACHE_FOR_INFO_FILE was added in 8.0.2.

- Error number: MY-010431; Symbol: ER_RPL_FAILED_TO_OPEN_INFO_FILE; SQLSTATE: HY000

  Message: Failed to open the existing info file (file '%s', errno %d)

  ER_RPL_FAILED_TO_OPEN_INFO_FILE was added in 8.0.2.

- Error number: MY-010432; Symbol: ER_RPL_GTID_MEMORY_FINALLY_AVAILABLE; SQLSTATE: HY000

  Message: Server overcomes the temporary 'out of memory' in '%d' tries while allocating a new chunk of intervals for storing GTIDs.

  ER_RPL_GTID_MEMORY_FINALLY_AVAILABLE was added in 8.0.2.

- Error number: MY-010433; Symbol: ER_SERVER_COST_UNKNOWN_COST_CONSTANT; SQLSTATE: HY000

  Message: Unknown cost constant "%s" in mysql.server_cost table

  ER_SERVER_COST_UNKNOWN_COST_CONSTANT was added in 8.0.2.

- Error number: MY-010434; Symbol: ER_SERVER_COST_INVALID_COST_CONSTANT; SQLSTATE: HY000

  Message: Invalid value for cost constant "%s" in mysql.server_cost table: %.1f

  ER_SERVER_COST_INVALID_COST_CONSTANT was added in 8.0.2.

- Error number: MY-010435; Symbol: ER_ENGINE_COST_UNKNOWN_COST_CONSTANT; SQLSTATE: HY000

  Message: Unknown cost constant "%s" in mysql.engine_cost table

  ER_ENGINE_COST_UNKNOWN_COST_CONSTANT was added in 8.0.2.

- Error number: MY-010436; Symbol: ER_ENGINE_COST_UNKNOWN_STORAGE_ENGINE; SQLSTATE: HY000

  Message: Unknown storage engine "%s" in mysql.engine_cost table

  ER_ENGINE_COST_UNKNOWN_STORAGE_ENGINE was added in 8.0.2.

- Error number: MY-010437; Symbol: ER_ENGINE_COST_INVALID_DEVICE_TYPE_FOR_SE; SQLSTATE: HY000

  Message: Invalid device type %d for "%s" storage engine for cost constant "%s" in mysql.engine_cost table

ER_ENGINE_COST_INVALID_DEVICE_TYPE_FOR_SE was added in 8.0.2.

- Error number: MY-010438; Symbol: ER_ENGINE_COST_INVALID_CONST_CONSTANT_FOR_SE_AND_DEVICE; SQLSTATE: HY000

  Message: Invalid value for cost constant "%s" for "%s" storage engine and device type %d in mysql.engine_cost table: %.1f

  ER_ENGINE_COST_INVALID_CONST_CONSTANT_FOR_SE_AND_DEVICE was added in 8.0.2.

- Error number: MY-010439; Symbol: ER_SERVER_COST_FAILED_TO_READ; SQLSTATE: HY000

  Message: Error while reading from mysql.server_cost table.

  ER_SERVER_COST_FAILED_TO_READ was added in 8.0.2.

- Error number: MY-010440; Symbol: ER_ENGINE_COST_FAILED_TO_READ; SQLSTATE: HY000

  Message: Error while reading from mysql.engine_cost table.

  ER_ENGINE_COST_FAILED_TO_READ was added in 8.0.2.

- Error number: MY-010441; Symbol: ER_FAILED_TO_OPEN_COST_CONSTANT_TABLES; SQLSTATE: HY000

  Message: Failed to open optimizer cost constant tables

  ER_FAILED_TO_OPEN_COST_CONSTANT_TABLES was added in 8.0.2.

- Error number: MY-010442; Symbol: ER_RPL_UNSUPPORTED_UNIGNORABLE_EVENT_IN_STREAM; SQLSTATE: HY000

  Message: Unsupported non-ignorable event fed into the event stream.

  ER_RPL_UNSUPPORTED_UNIGNORABLE_EVENT_IN_STREAM was added in 8.0.2.

- Error number: MY-010443; Symbol: ER_RPL_GTID_LOG_EVENT_IN_STREAM; SQLSTATE: HY000

  Message: GTID_LOG_EVENT or ANONYMOUS_GTID_LOG_EVENT is not expected in an event stream %s.

  ER_RPL_GTID_LOG_EVENT_IN_STREAM was added in 8.0.2.

- Error number: MY-010444; Symbol: ER_RPL_UNEXPECTED_BEGIN_IN_STREAM; SQLSTATE: HY000

  Message: QUERY(BEGIN) is not expected in an event stream in the middle of a %s.

  ER_RPL_UNEXPECTED_BEGIN_IN_STREAM was added in 8.0.2.

- Error number: MY-010445; Symbol: ER_RPL_UNEXPECTED_COMMIT_ROLLBACK_OR_XID_LOG_EVENT_IN_STREAM; SQLSTATE: HY000

  Message: QUERY(COMMIT or ROLLBACK) or XID_LOG_EVENT is not expected in an event stream %s.

  ER_RPL_UNEXPECTED_COMMIT_ROLLBACK_OR_XID_LOG_EVENT_IN_STREAM was added in 8.0.2.

- Error number: MY-010446; Symbol: ER_RPL_UNEXPECTED_XA_ROLLBACK_IN_STREAM; SQLSTATE: HY000

  Message: QUERY(XA ROLLBACK) is not expected in an event stream %s.

  ER_RPL_UNEXPECTED_XA_ROLLBACK_IN_STREAM was added in 8.0.2.

- Error number: MY-010447; Symbol: ER_EVENT_EXECUTION_FAILED_CANT_AUTHENTICATE_USER; SQLSTATE: HY000

  Message: Event Scheduler: [%s].[%s.%s] execution failed, failed to authenticate the user.

  ER_EVENT_EXECUTION_FAILED_CANT_AUTHENTICATE_USER was added in 8.0.2.

- Error number: MY-010448; Symbol: ER_EVENT_EXECUTION_FAILED_USER_LOST_EVEN_PRIVILEGE; SQLSTATE: HY000

  Message: Event Scheduler: [%s].[%s.%s] execution failed, user no longer has EVENT privilege.

  ER_EVENT_EXECUTION_FAILED_USER_LOST_EVEN_PRIVILEGE was added in 8.0.2.

- Error number: MY-010449; Symbol: ER_EVENT_ERROR_DURING_COMPILATION; SQLSTATE: HY000

  Message: Event Scheduler: %serror during compilation of %s.%s

  ER_EVENT_ERROR_DURING_COMPILATION was added in 8.0.2.

- Error number: MY-010450; Symbol: ER_EVENT_DROPPING; SQLSTATE: HY000

  Message: Event Scheduler: Dropping %s.%s

  ER_EVENT_DROPPING was added in 8.0.2.

- Error number: MY-010451; Symbol: ER_NDB_SCHEMA_GENERIC_MESSAGE; SQLSTATE: HY000

  Message: Ndb schema[%s.%s]: %s

  ER_NDB_SCHEMA_GENERIC_MESSAGE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010452; Symbol: ER_RPL_INCOMPATIBLE_DECIMAL_IN_RBR; SQLSTATE: HY000

  Message: In RBR mode, Slave received incompatible DECIMAL field (old-style decimal field) from Master while creating conversion table. Please consider changing datatype on Master to new style decimal by executing ALTER command for column Name: %s.%s.%s.

  ER_RPL_INCOMPATIBLE_DECIMAL_IN_RBR was added in 8.0.2.

- Error number: MY-010453; Symbol: ER_INIT_ROOT_WITHOUT_PASSWORD; SQLSTATE: HY000

  Message: root@localhost is created with an empty password ! Please consider switching off the --initialize-insecure option.

  ER_INIT_ROOT_WITHOUT_PASSWORD was added in 8.0.2.

- Error number: MY-010454; Symbol: ER_INIT_GENERATING_TEMP_PASSWORD_FOR_ROOT; SQLSTATE: HY000

  Message: A temporary password is generated for root@localhost: %s

ER_INIT_GENERATING_TEMP_PASSWORD_FOR_ROOT was added in 8.0.2.

- Error number: MY-010455; Symbol: ER_INIT_CANT_OPEN_BOOTSTRAP_FILE; SQLSTATE: HY000

  Message: Failed to open the bootstrap file %s

  ER_INIT_CANT_OPEN_BOOTSTRAP_FILE was added in 8.0.2.

- Error number: MY-010456; Symbol: ER_INIT_BOOTSTRAP_COMPLETE; SQLSTATE: HY000

  Message: Bootstrapping complete

  ER_INIT_BOOTSTRAP_COMPLETE was added in 8.0.2.

- Error number: MY-010457; Symbol: ER_INIT_DATADIR_NOT_EMPTY_WONT_INITIALIZE; SQLSTATE: HY000

  Message: --initialize specified but the data directory has files in it. Aborting.

  ER_INIT_DATADIR_NOT_EMPTY_WONT_INITIALIZE was added in 8.0.2.

- Error number: MY-010458; Symbol: ER_INIT_DATADIR_EXISTS_WONT_INITIALIZE; SQLSTATE: HY000

  Message: --initialize specified on an existing data directory.

  ER_INIT_DATADIR_EXISTS_WONT_INITIALIZE was added in 8.0.2.

- Error number: MY-010459; Symbol: ER_INIT_DATADIR_EXISTS_AND_PATH_TOO_LONG_WONT_INITIALIZE; SQLSTATE: HY000

  Message: --initialize specified but the data directory exists and the path is too long. Aborting.

  ER_INIT_DATADIR_EXISTS_AND_PATH_TOO_LONG_WONT_INITIALIZE was added in 8.0.2.

- Error number: MY-010460; Symbol: ER_INIT_DATADIR_EXISTS_AND_NOT_WRITABLE_WONT_INITIALIZE; SQLSTATE: HY000

  Message: --initialize specified but the data directory exists and is not writable. Aborting.

  ER_INIT_DATADIR_EXISTS_AND_NOT_WRITABLE_WONT_INITIALIZE was added in 8.0.2.

- Error number: MY-010461; Symbol: ER_INIT_CREATING_DD; SQLSTATE: HY000

  Message: Creating the data directory %s

  ER_INIT_CREATING_DD was added in 8.0.2.

- Error number: MY-010462; Symbol: ER_RPL_BINLOG_STARTING_DUMP; SQLSTATE: HY000

  Message: Start binlog_dump to master_thread_id(%u) slave_server(%u), pos(%s, %llu)

  ER_RPL_BINLOG_STARTING_DUMP was added in 8.0.2.

- Error number: MY-010463; Symbol: ER_RPL_BINLOG_MASTER_SENDS_HEARTBEAT; SQLSTATE: HY000

  Message: master sends heartbeat message

ER_RPL_BINLOG_MASTER_SENDS_HEARTBEAT was added in 8.0.2.

- Error number: MY-010464; Symbol: ER_RPL_BINLOG_SKIPPING_REMAINING_HEARTBEAT_INFO; SQLSTATE: HY000

  Message: the rest of heartbeat info skipped ...

  ER_RPL_BINLOG_SKIPPING_REMAINING_HEARTBEAT_INFO was added in 8.0.2.

- Error number: MY-010465; Symbol: ER_RPL_BINLOG_MASTER_USES_CHECKSUM_AND_SLAVE_CANT; SQLSTATE: HY000

  Message: Master is configured to log replication events with checksum, but will not send such events to slaves that cannot process them

  ER_RPL_BINLOG_MASTER_USES_CHECKSUM_AND_SLAVE_CANT was added in 8.0.2.

- Error number: MY-010466; Symbol: ER_NDB_QUERY_FAILED; SQLSTATE: HY000

  Message: NDB: Query '%s' failed, error: %d: %s

  ER_NDB_QUERY_FAILED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010467; Symbol: ER_KILLING_THREAD; SQLSTATE: HY000

  Message: Killing thread %lu

  ER_KILLING_THREAD was added in 8.0.2.

- Error number: MY-010468; Symbol: ER_DETACHING_SESSION_LEFT_BY_PLUGIN; SQLSTATE: HY000

  Message: Plugin %s is deinitializing a thread but left a session attached. Detaching it forcefully.

  ER_DETACHING_SESSION_LEFT_BY_PLUGIN was added in 8.0.2.

- Error number: MY-010469; Symbol: ER_CANT_DETACH_SESSION_LEFT_BY_PLUGIN; SQLSTATE: HY000

  Message: Failed to detach the session.

  ER_CANT_DETACH_SESSION_LEFT_BY_PLUGIN was added in 8.0.2.

- Error number: MY-010470; Symbol: ER_DETACHED_SESSIONS_LEFT_BY_PLUGIN; SQLSTATE: HY000

  Message: Closed forcefully %u session%s left opened by plugin %s

  ER_DETACHED_SESSIONS_LEFT_BY_PLUGIN was added in 8.0.2.

- Error number: MY-010471; Symbol: ER_FAILED_TO_DECREMENT_NUMBER_OF_THREADS; SQLSTATE: HY000

  Message: Failed to decrement the number of threads

  ER_FAILED_TO_DECREMENT_NUMBER_OF_THREADS was added in 8.0.2.

- Error number: MY-010472; Symbol: ER_PLUGIN_DID_NOT_DEINITIALIZE_THREADS; SQLSTATE: HY000

  Message: Plugin %s did not deinitialize %u threads

  ER_PLUGIN_DID_NOT_DEINITIALIZE_THREADS was added in 8.0.2.

- Error number: MY-010473; Symbol: ER_KILLED_THREADS_OF_PLUGIN; SQLSTATE: HY000

  Message: Killed %u threads of plugin %s

  ER_KILLED_THREADS_OF_PLUGIN was added in 8.0.2.

- Error number: MY-010474; Symbol: ER_NDB_SLAVE_MAX_REPLICATED_EPOCH_UNKNOWN; SQLSTATE: HY000

  Message: NDB Slave : Could not determine maximum replicated epoch from %s.%s at Slave start, error %u %s

  ER_NDB_SLAVE_MAX_REPLICATED_EPOCH_UNKNOWN was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010475; Symbol: ER_NDB_SLAVE_MAX_REPLICATED_EPOCH_SET_TO; SQLSTATE: HY000

  Message: NDB Slave : MaxReplicatedEpoch set to %llu (%u/%u) at Slave start

  ER_NDB_SLAVE_MAX_REPLICATED_EPOCH_SET_TO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010476; Symbol: ER_NDB_NODE_ID_AND_MANAGEMENT_SERVER_INFO; SQLSTATE: HY000

  Message: NDB: NodeID is %lu, management server '%s:%lu'

  ER_NDB_NODE_ID_AND_MANAGEMENT_SERVER_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010477; Symbol: ER_NDB_DISCONNECT_INFO; SQLSTATE: HY000

  Message: tid %u: node[%u] transaction_hint=%u, transaction_no_hint=%u

  ER_NDB_DISCONNECT_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010478; Symbol: ER_NDB_COLUMN_DEFAULTS_DIFFER; SQLSTATE: HY000

  Message: NDB Internal error: Default values differ for column %u, ndb_default: %d

  ER_NDB_COLUMN_DEFAULTS_DIFFER was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010479; Symbol: ER_NDB_COLUMN_SHOULD_NOT_HAVE_NATIVE_DEFAULT; SQLSTATE: HY000

  Message: NDB Internal error: Column %u has native default, but shouldn't. Flags=%u, type=%u

  ER_NDB_COLUMN_SHOULD_NOT_HAVE_NATIVE_DEFAULT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010480; Symbol: ER_NDB_FIELD_INFO; SQLSTATE: HY000

  Message: field[name: '%s', type: %u, real_type: %u, flags: 0x%x, is_null: %d]

  ER_NDB_FIELD_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010481; Symbol: ER_NDB_COLUMN_INFO; SQLSTATE: HY000

  Message: ndbCol[name: '%s', type: %u, column_no: %d, nullable: %d]

  ER_NDB_COLUMN_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010482; Symbol: ER_NDB_OOM_IN_FIX_UNIQUE_INDEX_ATTR_ORDER; SQLSTATE: HY000

  Message: fix_unique_index_attr_order: my_malloc(%u) failure

  ER_NDB_OOM_IN_FIX_UNIQUE_INDEX_ATTR_ORDER was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010483; Symbol: ER_NDB_SLAVE_MALFORMED_EVENT_RECEIVED_ON_TABLE; SQLSTATE: HY000

  Message: NDB Slave : Malformed event received on table %s cannot parse. Stopping Slave.

  ER_NDB_SLAVE_MALFORMED_EVENT_RECEIVED_ON_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010484; Symbol: ER_NDB_SLAVE_CONFLICT_FUNCTION_REQUIRES_ROLE; SQLSTATE: HY000

  Message: NDB Slave : Conflict function %s defined on table %s requires ndb_slave_conflict_role variable to be set. Stopping slave.

  ER_NDB_SLAVE_CONFLICT_FUNCTION_REQUIRES_ROLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010485; Symbol: ER_NDB_SLAVE_CONFLICT_DETECTION_REQUIRES_TRANSACTION_IDS; SQLSTATE: HY000

  Message: NDB Slave : Transactional conflict detection defined on table %s, but events received without transaction ids. Check --ndb-log-transaction-id setting on upstream Cluster.

  ER_NDB_SLAVE_CONFLICT_DETECTION_REQUIRES_TRANSACTION_IDS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010486; Symbol: ER_NDB_SLAVE_BINLOG_MISSING_INFO_FOR_CONFLICT_DETECTION; SQLSTATE: HY000

  Message: NDB Slave : Binlog event on table %s missing info necessary for conflict detection. Check binlog format options on upstream cluster.

  ER_NDB_SLAVE_BINLOG_MISSING_INFO_FOR_CONFLICT_DETECTION was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010487; Symbol: ER_NDB_ERROR_IN_READAUTOINCREMENTVALUE; SQLSTATE: HY000

  Message: Error %lu in readAutoIncrementValue(): %s

  ER_NDB_ERROR_IN_READAUTOINCREMENTVALUE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010488; Symbol: ER_NDB_FOUND_UNCOMMITTED_AUTOCOMMIT; SQLSTATE: HY000

  Message: found uncommitted autocommit+rbwr transaction, commit status: %d

  ER_NDB_FOUND_UNCOMMITTED_AUTOCOMMIT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010489; Symbol: ER_NDB_SLAVE_TOO_MANY_RETRIES; SQLSTATE: HY000

  Message: Ndb slave retried transaction %u time(s) in vain. Giving up.

  ER_NDB_SLAVE_TOO_MANY_RETRIES was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010490; Symbol: ER_NDB_SLAVE_ERROR_IN_UPDATE_CREATE_INFO; SQLSTATE: HY000

  Message: Error %lu in ::update_create_info(): %s

  ER_NDB_SLAVE_ERROR_IN_UPDATE_CREATE_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010491; Symbol: ER_NDB_SLAVE_CANT_ALLOCATE_TABLE_SHARE; SQLSTATE: HY000

  Message: NDB: allocating table share for %s failed

  ER_NDB_SLAVE_CANT_ALLOCATE_TABLE_SHARE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010492; Symbol: ER_NDB_BINLOG_ERROR_INFO_FROM_DA; SQLSTATE: HY000

  Message: NDB Binlog: (%d)%s

  ER_NDB_BINLOG_ERROR_INFO_FROM_DA was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010493; Symbol: ER_NDB_BINLOG_CREATE_TABLE_EVENT; SQLSTATE: HY000

  Message: NDB Binlog: CREATE TABLE Event: %s

  ER_NDB_BINLOG_CREATE_TABLE_EVENT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010494; Symbol: ER_NDB_BINLOG_FAILED_CREATE_TABLE_EVENT_OPERATIONS; SQLSTATE: HY000

  Message: NDB Binlog: FAILED CREATE TABLE event operations. Event: %s

  ER_NDB_BINLOG_FAILED_CREATE_TABLE_EVENT_OPERATIONS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010495; Symbol: ER_NDB_BINLOG_RENAME_EVENT; SQLSTATE: HY000

  Message: NDB Binlog: RENAME Event: %s

  ER_NDB_BINLOG_RENAME_EVENT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010496; Symbol: ER_NDB_BINLOG_FAILED_CREATE_EVENT_OPERATIONS_DURING_RENAME; SQLSTATE: HY000

  Message: NDB Binlog: FAILED create event operations during RENAME. Event %s

  ER_NDB_BINLOG_FAILED_CREATE_EVENT_OPERATIONS_DURING_RENAME was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010497; Symbol: ER_NDB_UNEXPECTED_RENAME_TYPE; SQLSTATE: HY000

  Message: Unexpected rename case detected, sql_command: %d

  ER_NDB_UNEXPECTED_RENAME_TYPE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010498; Symbol: ER_NDB_ERROR_IN_GET_AUTO_INCREMENT; SQLSTATE: HY000

  Message: Error %lu in ::get_auto_increment(): %s

  ER_NDB_ERROR_IN_GET_AUTO_INCREMENT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010499; Symbol: ER_NDB_CREATING_SHARE_IN_OPEN; SQLSTATE: HY000

  Message: Calling ndbcluster_create_binlog_setup(%s) in ::open

  ER_NDB_CREATING_SHARE_IN_OPEN was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010500; Symbol: ER_NDB_TABLE_OPENED_READ_ONLY; SQLSTATE: HY000

  Message: table '%s' opened read only

  ER_NDB_TABLE_OPENED_READ_ONLY was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010501; Symbol: ER_NDB_INITIALIZE_GIVEN_CLUSTER_PLUGIN_DISABLED; SQLSTATE: HY000

  Message: NDB: '--initialize' -> ndbcluster plugin disabled

  ER_NDB_INITIALIZE_GIVEN_CLUSTER_PLUGIN_DISABLED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010502; Symbol: ER_NDB_BINLOG_FORMAT_CHANGED_FROM_STMT_TO_MIXED; SQLSTATE: HY000

  Message: NDB: Changed global value of binlog_format from STATEMENT to MIXED

  ER_NDB_BINLOG_FORMAT_CHANGED_FROM_STMT_TO_MIXED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010503; Symbol: ER_NDB_TRAILING_SHARE_RELEASED_BY_CLOSE_CACHED_TABLES; SQLSTATE: HY000

  Message: NDB_SHARE: trailing share %s, released by close_cached_tables

  ER_NDB_TRAILING_SHARE_RELEASED_BY_CLOSE_CACHED_TABLES was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010504; Symbol: ER_NDB_SHARE_ALREADY_EXISTS; SQLSTATE: HY000

  Message: NDB_SHARE: %s already exists use_count=%d. Moving away for safety, but possible memleak.

  ER_NDB_SHARE_ALREADY_EXISTS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010505; Symbol: ER_NDB_HANDLE_TRAILING_SHARE_INFO; SQLSTATE: HY000

  Message: handle_trailing_share: %s use_count: %u

  ER_NDB_HANDLE_TRAILING_SHARE_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010506; Symbol: ER_NDB_CLUSTER_GET_SHARE_INFO; SQLSTATE: HY000

  Message: ndbcluster_get_share: %s use_count: %u

ER_NDB_CLUSTER_GET_SHARE_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010507; Symbol: ER_NDB_CLUSTER_REAL_FREE_SHARE_INFO; SQLSTATE: HY000

  Message: ndbcluster_real_free_share: %s use_count: %u

  ER_NDB_CLUSTER_REAL_FREE_SHARE_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010508; Symbol: ER_NDB_CLUSTER_REAL_FREE_SHARE_DROP_FAILED; SQLSTATE: HY000

  Message: ndbcluster_real_free_share: %s, still open - ignored 'free' (leaked?)

  ER_NDB_CLUSTER_REAL_FREE_SHARE_DROP_FAILED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010509; Symbol: ER_NDB_CLUSTER_FREE_SHARE_INFO; SQLSTATE: HY000

  Message: ndbcluster_free_share: %s use_count: %u

  ER_NDB_CLUSTER_FREE_SHARE_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010510; Symbol: ER_NDB_CLUSTER_MARK_SHARE_DROPPED_INFO; SQLSTATE: HY000

  Message: ndbcluster_mark_share_dropped: %s use_count: %u

  ER_NDB_CLUSTER_MARK_SHARE_DROPPED_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010511; Symbol: ER_NDB_CLUSTER_MARK_SHARE_DROPPED_DESTROYING_SHARE; SQLSTATE: HY000

  Message: ndbcluster_mark_share_dropped: destroys share %s

  ER_NDB_CLUSTER_MARK_SHARE_DROPPED_DESTROYING_SHARE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010512; Symbol: ER_NDB_CLUSTER_OOM_THD_NDB; SQLSTATE: HY000

  Message: Could not allocate Thd_ndb object

  ER_NDB_CLUSTER_OOM_THD_NDB was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010513; Symbol: ER_NDB_BINLOG_NDB_TABLES_INITIALLY_READ_ONLY; SQLSTATE: HY000

  Message: NDB Binlog: Ndb tables initially read only.

  ER_NDB_BINLOG_NDB_TABLES_INITIALLY_READ_ONLY was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010514; Symbol: ER_NDB_UTIL_THREAD_OOM; SQLSTATE: HY000

  Message: ndb util thread: malloc failure, query cache not maintained properly

  ER_NDB_UTIL_THREAD_OOM was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010515; Symbol: ER_NDB_ILLEGAL_VALUE_FOR_NDB_RECV_THREAD_CPU_MASK; SQLSTATE: HY000

Message: Trying to set ndb_recv_thread_cpu_mask to illegal value = %s, ignored

ER_NDB_ILLEGAL_VALUE_FOR_NDB_RECV_THREAD_CPU_MASK was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010516; Symbol: ER_NDB_TOO_MANY_CPUS_IN_NDB_RECV_THREAD_CPU_MASK; SQLSTATE: HY000

  Message: Trying to set too many CPU's in ndb_recv_thread_cpu_mask, ignored this variable, erroneus value = %s

  ER_NDB_TOO_MANY_CPUS_IN_NDB_RECV_THREAD_CPU_MASK was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010517; Symbol: ER_DBUG_CHECK_SHARES_OPEN; SQLSTATE: HY000

  Message: dbug_check_shares open:

  ER_DBUG_CHECK_SHARES_OPEN was added in 8.0.2.

- Error number: MY-010518; Symbol: ER_DBUG_CHECK_SHARES_INFO; SQLSTATE: HY000

  Message: %s.%s: state: %s(%u) use_count: %u

  ER_DBUG_CHECK_SHARES_INFO was added in 8.0.2.

- Error number: MY-010519; Symbol: ER_DBUG_CHECK_SHARES_DROPPED; SQLSTATE: HY000

  Message: dbug_check_shares dropped:

  ER_DBUG_CHECK_SHARES_DROPPED was added in 8.0.2.

- Error number: MY-010520; Symbol: ER_INVALID_OR_OLD_TABLE_OR_DB_NAME; SQLSTATE: HY000

  Message: Invalid (old?) table or database name '%s'

  ER_INVALID_OR_OLD_TABLE_OR_DB_NAME was added in 8.0.2.

- Error number: MY-010521; Symbol: ER_TC_RECOVERING_AFTER_CRASH_USING; SQLSTATE: HY000

  Message: Recovering after a crash using %s

  ER_TC_RECOVERING_AFTER_CRASH_USING was added in 8.0.2.

- Error number: MY-010522; Symbol: ER_TC_CANT_AUTO_RECOVER_WITH_TC_HEURISTIC_RECOVER; SQLSTATE: HY000

  Message: Cannot perform automatic crash recovery when --tc-heuristic-recover is used

  ER_TC_CANT_AUTO_RECOVER_WITH_TC_HEURISTIC_RECOVER was added in 8.0.2.

- Error number: MY-010523; Symbol: ER_TC_BAD_MAGIC_IN_TC_LOG; SQLSTATE: HY000

  Message: Bad magic header in tc log

  ER_TC_BAD_MAGIC_IN_TC_LOG was added in 8.0.2.

- Error number: MY-010524; Symbol: ER_TC_NEED_N_SE_SUPPORTING_2PC_FOR_RECOVERY; SQLSTATE: HY000

  Message: Recovery failed! You must enable exactly %d storage engines that support two-phase commit protocol

  ER_TC_NEED_N_SE_SUPPORTING_2PC_FOR_RECOVERY was added in 8.0.2.

- Error number: MY-010525; Symbol: ER_TC_RECOVERY_FAILED_THESE_ARE_YOUR_OPTIONS; SQLSTATE: HY000

  Message: Crash recovery failed. Either correct the problem (if it's, for example, out of memory error) and restart, or delete tc log and start mysqld with --tc-heuristic-recover={commit|rollback}

  ER_TC_RECOVERY_FAILED_THESE_ARE_YOUR_OPTIONS was added in 8.0.2.

- Error number: MY-010526; Symbol: ER_TC_HEURISTIC_RECOVERY_MODE; SQLSTATE: HY000

  Message: Heuristic crash recovery mode

  ER_TC_HEURISTIC_RECOVERY_MODE was added in 8.0.2.

- Error number: MY-010527; Symbol: ER_TC_HEURISTIC_RECOVERY_FAILED; SQLSTATE: HY000

  Message: Heuristic crash recovery failed

  ER_TC_HEURISTIC_RECOVERY_FAILED was added in 8.0.2.

- Error number: MY-010528; Symbol: ER_TC_RESTART_WITHOUT_TC_HEURISTIC_RECOVER; SQLSTATE: HY000

  Message: Please restart mysqld without --tc-heuristic-recover

  ER_TC_RESTART_WITHOUT_TC_HEURISTIC_RECOVER was added in 8.0.2.

- Error number: MY-010529; Symbol: ER_RPL_SLAVE_FAILED_TO_CREATE_OR_RECOVER_INFO_REPOSITORIES; SQLSTATE: HY000

  Message: Failed to create or recover replication info repositories.

  ER_RPL_SLAVE_FAILED_TO_CREATE_OR_RECOVER_INFO_REPOSITORIES was added in 8.0.2.

- Error number: MY-010530; Symbol: ER_RPL_SLAVE_AUTO_POSITION_IS_1_AND_GTID_MODE_IS_OFF; SQLSTATE: HY000

  Message: Detected misconfiguration: replication channel '%s' was configured with AUTO_POSITION = 1, but the server was started with --gtid-mode=off. Either reconfigure replication using CHANGE MASTER TO MASTER_AUTO_POSITION = 0 FOR CHANNEL '%s', or change GTID_MODE to some value other than OFF, before starting the slave receiver thread.

  ER_RPL_SLAVE_AUTO_POSITION_IS_1_AND_GTID_MODE_IS_OFF was added in 8.0.2.

- Error number: MY-010531; Symbol: ER_RPL_SLAVE_CANT_START_SLAVE_FOR_CHANNEL; SQLSTATE: HY000

  Message: Slave: Could not start slave for channel '%s'. operation discontinued

  ER_RPL_SLAVE_CANT_START_SLAVE_FOR_CHANNEL was added in 8.0.2.

- Error number: MY-010532; Symbol: ER_RPL_SLAVE_CANT_STOP_SLAVE_FOR_CHANNEL; SQLSTATE: HY000

  Message: Slave: Could not stop slave for channel '%s' operation discontinued

  ER_RPL_SLAVE_CANT_STOP_SLAVE_FOR_CHANNEL was added in 8.0.2.

- Error number: MY-010533; Symbol: ER_RPL_RECOVERY_NO_ROTATE_EVENT_FROM_MASTER; SQLSTATE: HY000

  Message: Error during --relay-log-recovery: Could not locate rotate event from the master.

  ER_RPL_RECOVERY_NO_ROTATE_EVENT_FROM_MASTER was added in 8.0.2.

- Error number: MY-010534; Symbol: ER_RPL_RECOVERY_ERROR_READ_RELAY_LOG; SQLSTATE: HY000

  Message: Error during --relay-log-recovery: Error reading events from relay log: %d

  ER_RPL_RECOVERY_ERROR_READ_RELAY_LOG was added in 8.0.2.

- Error number: MY-010535; Symbol: ER_RPL_RECOVERY_ERROR_FREEING_IO_CACHE; SQLSTATE: HY000

  Message: Error during --relay-log-recovery: Error while freeing IO_CACHE object

  ER_RPL_RECOVERY_ERROR_FREEING_IO_CACHE was added in 8.0.2, removed after 8.0.12.

- Error number: MY-010536; Symbol: ER_RPL_RECOVERY_SKIPPED_GROUP_REPLICATION_CHANNEL; SQLSTATE: HY000

  Message: Relay log recovery skipped for group replication channel.

  ER_RPL_RECOVERY_SKIPPED_GROUP_REPLICATION_CHANNEL was added in 8.0.2.

- Error number: MY-010537; Symbol: ER_RPL_RECOVERY_ERROR; SQLSTATE: HY000

  Message: Error during --relay-log-recovery: %s

  ER_RPL_RECOVERY_ERROR was added in 8.0.2.

- Error number: MY-010538; Symbol: ER_RPL_RECOVERY_IO_ERROR_READING_RELAY_LOG_INDEX; SQLSTATE: HY000

  Message: Error during --relay-log-recovery: Could not read relay log index file due to an IO error.

  ER_RPL_RECOVERY_IO_ERROR_READING_RELAY_LOG_INDEX was added in 8.0.2.

- Error number: MY-010539; Symbol: ER_RPL_RECOVERY_FILE_MASTER_POS_INFO; SQLSTATE: HY000

  Message: Recovery from master pos %ld and file %s%s. Previous relay log pos and relay log file had been set to %lld, %s respectively.

  ER_RPL_RECOVERY_FILE_MASTER_POS_INFO was added in 8.0.2.

- Error number: MY-010540; Symbol: ER_RPL_RECOVERY_REPLICATE_SAME_SERVER_ID_REQUIRES_POSITION; SQLSTATE: HY000

Message: Error during --relay-log-recovery: replicate_same_server_id is in use and sql thread's positions are not initialized, hence relay log recovery cannot happen.

ER_RPL_RECOVERY_REPLICATE_SAME_SERVER_ID_REQUIRES_POSITION was added in 8.0.2.

- Error number: MY-010541; Symbol: ER_RPL_MTS_RECOVERY_STARTING_COORDINATOR; SQLSTATE: HY000

  Message: MTS recovery: starting coordinator thread to fill MTS gaps.

  ER_RPL_MTS_RECOVERY_STARTING_COORDINATOR was added in 8.0.2.

- Error number: MY-010542; Symbol: ER_RPL_MTS_RECOVERY_FAILED_TO_START_COORDINATOR; SQLSTATE: HY000

  Message: MTS recovery: failed to start the coordinator thread. Check the error log for additional details.

  ER_RPL_MTS_RECOVERY_FAILED_TO_START_COORDINATOR was added in 8.0.2.

- Error number: MY-010543; Symbol: ER_RPL_MTS_AUTOMATIC_RECOVERY_FAILED; SQLSTATE: HY000

  Message: MTS recovery: automatic recovery failed. Either the slave server had stopped due to an error during an earlier session or relay logs are corrupted.Fix the cause of the slave side error and restart the slave server or consider using RESET SLAVE.

  ER_RPL_MTS_AUTOMATIC_RECOVERY_FAILED was added in 8.0.2.

- Error number: MY-010544; Symbol: ER_RPL_MTS_RECOVERY_CANT_OPEN_RELAY_LOG; SQLSTATE: HY000

  Message: Failed to open the relay log '%s' (relay_log_pos %s).

  ER_RPL_MTS_RECOVERY_CANT_OPEN_RELAY_LOG was added in 8.0.2.

- Error number: MY-010545; Symbol: ER_RPL_MTS_RECOVERY_SUCCESSFUL; SQLSTATE: HY000

  Message: MTS recovery: completed successfully.

  ER_RPL_MTS_RECOVERY_SUCCESSFUL was added in 8.0.2.

- Error number: MY-010546; Symbol: ER_RPL_SERVER_ID_MISSING; SQLSTATE: HY000

  Message: Server id not set, will not start slave%s

  ER_RPL_SERVER_ID_MISSING was added in 8.0.2.

- Error number: MY-010547; Symbol: ER_RPL_CANT_CREATE_SLAVE_THREAD; SQLSTATE: HY000

  Message: Can't create slave thread%s.

  ER_RPL_CANT_CREATE_SLAVE_THREAD was added in 8.0.2.

- Error number: MY-010548; Symbol: ER_RPL_SLAVE_IO_THREAD_WAS_KILLED; SQLSTATE: HY000

  Message: The slave IO thread%s was killed while executing initialization query '%s'

  ER_RPL_SLAVE_IO_THREAD_WAS_KILLED was added in 8.0.2.

- Error number: MY-010549; Symbol: ER_RPL_SLAVE_MASTER_UUID_HAS_CHANGED; SQLSTATE: HY000

  Message: The master's UUID has changed, although this should not happen unless you have changed it manually. The old UUID was %s.

  ER_RPL_SLAVE_MASTER_UUID_HAS_CHANGED was added in 8.0.2.

- Error number: MY-010550; Symbol: ER_RPL_SLAVE_USES_CHECKSUM_AND_MASTER_PRE_50; SQLSTATE: HY000

  Message: Found a master with MySQL server version older than 5.0. With checksums enabled on the slave, replication might not work correctly. To ensure correct replication, restart the slave server with --slave_sql_verify_checksum=0.

  ER_RPL_SLAVE_USES_CHECKSUM_AND_MASTER_PRE_50 was added in 8.0.2.

- Error number: MY-010551; Symbol: ER_RPL_SLAVE_SECONDS_BEHIND_MASTER_DUBIOUS; SQLSTATE: HY000

  Message: "SELECT UNIX_TIMESTAMP()" failed on master, do not trust column Seconds_Behind_Master of SHOW SLAVE STATUS. Error: %s (%d)

  ER_RPL_SLAVE_SECONDS_BEHIND_MASTER_DUBIOUS was added in 8.0.2.

- Error number: MY-010552; Symbol: ER_RPL_SLAVE_CANT_FLUSH_MASTER_INFO_FILE; SQLSTATE: HY000

  Message: Failed to flush master info file.

  ER_RPL_SLAVE_CANT_FLUSH_MASTER_INFO_FILE was added in 8.0.2.

- Error number: MY-010553; Symbol: ER_RPL_SLAVE_REPORT_HOST_TOO_LONG; SQLSTATE: HY000

  Message: The length of report_host is %zu. It is larger than the max length(%d), so this slave cannot be registered to the master%s.

  ER_RPL_SLAVE_REPORT_HOST_TOO_LONG was added in 8.0.2.

- Error number: MY-010554; Symbol: ER_RPL_SLAVE_REPORT_USER_TOO_LONG; SQLSTATE: HY000

  Message: The length of report_user is %zu. It is larger than the max length(%d), so this slave cannot be registered to the master%s.

  ER_RPL_SLAVE_REPORT_USER_TOO_LONG was added in 8.0.2.

- Error number: MY-010555; Symbol: ER_RPL_SLAVE_REPORT_PASSWORD_TOO_LONG; SQLSTATE: HY000

  Message: The length of report_password is %zu. It is larger than the max length(%d), so this slave cannot be registered to the master%s.

  ER_RPL_SLAVE_REPORT_PASSWORD_TOO_LONG was added in 8.0.2.

- Error number: MY-010556; Symbol: ER_RPL_SLAVE_ERROR_RETRYING; SQLSTATE: HY000

  Message: Error on %s: %d %s, will retry in %d secs

  ER_RPL_SLAVE_ERROR_RETRYING was added in 8.0.2.

- Error number: MY-010557; Symbol: ER_RPL_SLAVE_ERROR_READING_FROM_SERVER; SQLSTATE: HY000

  Message: Error reading packet from server%s: %s (server_errno=%d)

  ER_RPL_SLAVE_ERROR_READING_FROM_SERVER was added in 8.0.2.

- Error number: MY-010558; Symbol: ER_RPL_SLAVE_DUMP_THREAD_KILLED_BY_MASTER; SQLSTATE: HY000

  Message: Slave%s: received end packet from server due to dump thread being killed on master. Dump threads are killed for example during master shutdown, explicitly by a user, or when the master receives a binlog send request from a duplicate server UUID <%s> : Error %s

  ER_RPL_SLAVE_DUMP_THREAD_KILLED_BY_MASTER was added in 8.0.2.

- Error number: MY-010559; Symbol: ER_RPL_MTS_STATISTICS; SQLSTATE: HY000

  Message: Multi-threaded slave statistics%s: seconds elapsed = %lu; events assigned = %llu; worker queues filled over overrun level = %lu; waited due a Worker queue full = %lu; waited due the total size = %lu; waited at clock conflicts = %llu waited (count) when Workers occupied = %lu waited when Workers occupied = %llu

  ER_RPL_MTS_STATISTICS was added in 8.0.2.

- Error number: MY-010560; Symbol: ER_RPL_MTS_RECOVERY_COMPLETE; SQLSTATE: HY000

  Message: Slave%s: MTS Recovery has completed at relay log %s, position %llu master log %s, position %llu.

  ER_RPL_MTS_RECOVERY_COMPLETE was added in 8.0.2.

- Error number: MY-010561; Symbol: ER_RPL_SLAVE_CANT_INIT_RELAY_LOG_POSITION; SQLSTATE: HY000

  Message: Error initializing relay log position%s: %s

  ER_RPL_SLAVE_CANT_INIT_RELAY_LOG_POSITION was added in 8.0.2.

- Error number: MY-010562; Symbol: ER_RPL_SLAVE_CONNECTED_TO_MASTER_REPLICATION_STARTED; SQLSTATE: HY000

  Message: Slave I/O thread%s: connected to master '%s@%s:%d',replication started in log '%s' at position %s

  ER_RPL_SLAVE_CONNECTED_TO_MASTER_REPLICATION_STARTED was added in 8.0.2.

- Error number: MY-010563; Symbol: ER_RPL_SLAVE_IO_THREAD_KILLED; SQLSTATE: HY000

  Message: Slave I/O thread%s killed while connecting to master

  ER_RPL_SLAVE_IO_THREAD_KILLED was added in 8.0.2.

- Error number: MY-010564; Symbol: ER_RPL_SLAVE_IO_THREAD_CANT_REGISTER_ON_MASTER; SQLSTATE: HY000

  Message: Slave I/O thread couldn't register on master

  ER_RPL_SLAVE_IO_THREAD_CANT_REGISTER_ON_MASTER was added in 8.0.2.

- Error number: MY-010565; Symbol: ER_RPL_SLAVE_FORCING_TO_RECONNECT_IO_THREAD; SQLSTATE: HY000

  Message: Forcing to reconnect slave I/O thread%s

  ER_RPL_SLAVE_FORCING_TO_RECONNECT_IO_THREAD was added in 8.0.2.

- Error number: MY-010566; Symbol: ER_RPL_SLAVE_ERROR_REQUESTING_BINLOG_DUMP; SQLSTATE: HY000

  Message: Failed on request_dump()%s

  ER_RPL_SLAVE_ERROR_REQUESTING_BINLOG_DUMP was added in 8.0.2.

- Error number: MY-010567; Symbol: ER_RPL_LOG_ENTRY_EXCEEDS_SLAVE_MAX_ALLOWED_PACKET; SQLSTATE: HY000

  Message: Log entry on master is longer than slave_max_allowed_packet (%lu) on slave. If the entry is correct, restart the server with a higher value of slave_max_allowed_packet

  ER_RPL_LOG_ENTRY_EXCEEDS_SLAVE_MAX_ALLOWED_PACKET was added in 8.0.2.

- Error number: MY-010568; Symbol: ER_RPL_SLAVE_STOPPING_AS_MASTER_OOM; SQLSTATE: HY000

  Message: Stopping slave I/O thread due to out-of-memory error from master

  ER_RPL_SLAVE_STOPPING_AS_MASTER_OOM was added in 8.0.2.

- Error number: MY-010569; Symbol: ER_RPL_SLAVE_IO_THREAD_ABORTED_WAITING_FOR_RELAY_LOG_SPACE; SQLSTATE: HY000

  Message: Slave I/O thread aborted while waiting for relay log space

  ER_RPL_SLAVE_IO_THREAD_ABORTED_WAITING_FOR_RELAY_LOG_SPACE was added in 8.0.2.

- Error number: MY-010570; Symbol: ER_RPL_SLAVE_IO_THREAD_EXITING; SQLSTATE: HY000

  Message: Slave I/O thread exiting%s, read up to log '%s', position %s

  ER_RPL_SLAVE_IO_THREAD_EXITING was added in 8.0.2.

- Error number: MY-010571; Symbol: ER_RPL_SLAVE_CANT_INITIALIZE_SLAVE_WORKER; SQLSTATE: HY000

  Message: Failed during slave worker initialization%s

  ER_RPL_SLAVE_CANT_INITIALIZE_SLAVE_WORKER was added in 8.0.2.

- Error number: MY-010572; Symbol: ER_RPL_MTS_GROUP_RECOVERY_RELAY_LOG_INFO_FOR_WORKER; SQLSTATE: HY000

  Message: Slave: MTS group recovery relay log info based on Worker-Id %lu, group_relay_log_name %s, group_relay_log_pos %llu group_master_log_name %s, group_master_log_pos %llu

  ER_RPL_MTS_GROUP_RECOVERY_RELAY_LOG_INFO_FOR_WORKER was added in 8.0.2.

- Error number: MY-010573; Symbol: ER_RPL_ERROR_LOOKING_FOR_LOG; SQLSTATE: HY000

Message: Error looking for %s.

ER_RPL_ERROR_LOOKING_FOR_LOG was added in 8.0.2.

- Error number: MY-010574; Symbol: ER_RPL_MTS_GROUP_RECOVERY_RELAY_LOG_INFO; SQLSTATE: HY000

  Message: Slave: MTS group recovery relay log info group_master_log_name %s, event_master_log_pos %llu.

  ER_RPL_MTS_GROUP_RECOVERY_RELAY_LOG_INFO was added in 8.0.2.

- Error number: MY-010575; Symbol: ER_RPL_CANT_FIND_FOLLOWUP_FILE; SQLSTATE: HY000

  Message: Error looking for file after %s.

  ER_RPL_CANT_FIND_FOLLOWUP_FILE was added in 8.0.2.

- Error number: MY-010576; Symbol: ER_RPL_MTS_CHECKPOINT_PERIOD_DIFFERS_FROM_CNT; SQLSTATE: HY000

  Message: This an error cnt != mts_checkpoint_period

  ER_RPL_MTS_CHECKPOINT_PERIOD_DIFFERS_FROM_CNT was added in 8.0.2.

- Error number: MY-010577; Symbol: ER_RPL_SLAVE_WORKER_THREAD_CREATION_FAILED; SQLSTATE: HY000

  Message: Failed during slave worker thread creation%s

  ER_RPL_SLAVE_WORKER_THREAD_CREATION_FAILED was added in 8.0.2.

- Error number: MY-010578; Symbol: ER_RPL_SLAVE_WORKER_THREAD_CREATION_FAILED_WITH_ERRNO; SQLSTATE: HY000

  Message: Failed during slave worker thread creation%s (errno= %d)

  ER_RPL_SLAVE_WORKER_THREAD_CREATION_FAILED_WITH_ERRNO was added in 8.0.2.

- Error number: MY-010579; Symbol: ER_RPL_SLAVE_FAILED_TO_INIT_PARTITIONS_HASH; SQLSTATE: HY000

  Message: Failed to init partitions hash

  ER_RPL_SLAVE_FAILED_TO_INIT_PARTITIONS_HASH was added in 8.0.2.

- Error number: MY-010580; Symbol: ER_RPL_SLAVE_NDB_TABLES_NOT_AVAILABLE; SQLSTATE: HY000

  Message: Slave SQL thread : NDB : Tables not available after %lu seconds. Consider increasing --ndb-wait-setup value

  ER_RPL_SLAVE_NDB_TABLES_NOT_AVAILABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010581; Symbol: ER_RPL_SLAVE_SQL_THREAD_STARTING; SQLSTATE: HY000

  Message: Slave SQL thread%s initialized, starting replication in log '%s' at position %s, relay log '%s' position: %s

ER_RPL_SLAVE_SQL_THREAD_STARTING was added in 8.0.2.

- Error number: MY-010582; Symbol: ER_RPL_SLAVE_SKIP_COUNTER_EXECUTED; SQLSTATE: HY000

  Message: 'SQL_SLAVE_SKIP_COUNTER=%ld' executed at relay_log_file='%s', relay_log_pos='%ld', master_log_name='%s', master_log_pos='%ld' and new position at relay_log_file='%s', relay_log_pos='%ld', master_log_name='%s', master_log_pos='%ld'

  ER_RPL_SLAVE_SKIP_COUNTER_EXECUTED was added in 8.0.2.

- Error number: MY-010583; Symbol: ER_RPL_SLAVE_ADDITIONAL_ERROR_INFO_FROM_DA; SQLSTATE: HY000

  Message: Slave (additional info): %s Error_code: MY-%06d

  ER_RPL_SLAVE_ADDITIONAL_ERROR_INFO_FROM_DA was added in 8.0.2.

- Error number: MY-010584; Symbol: ER_RPL_SLAVE_ERROR_INFO_FROM_DA; SQLSTATE: HY000

  Message: Slave: %s Error_code: MY-%06d

  ER_RPL_SLAVE_ERROR_INFO_FROM_DA was added in 8.0.2.

- Error number: MY-010585; Symbol: ER_RPL_SLAVE_ERROR_LOADING_USER_DEFINED_LIBRARY; SQLSTATE: HY000

  Message: Error loading user-defined library, slave SQL thread aborted. Install the missing library, and restart the slave SQL thread with "SLAVE START". We stopped at log '%s' position %s

  ER_RPL_SLAVE_ERROR_LOADING_USER_DEFINED_LIBRARY was added in 8.0.2.

- Error number: MY-010586; Symbol: ER_RPL_SLAVE_ERROR_RUNNING_QUERY; SQLSTATE: HY000

  Message: Error running query, slave SQL thread aborted. Fix the problem, and restart the slave SQL thread with "SLAVE START". We stopped at log '%s' position %s

  ER_RPL_SLAVE_ERROR_RUNNING_QUERY was added in 8.0.2.

- Error number: MY-010587; Symbol: ER_RPL_SLAVE_SQL_THREAD_EXITING; SQLSTATE: HY000

  Message: Slave SQL thread%s exiting, replication stopped in log '%s' at position %s

  ER_RPL_SLAVE_SQL_THREAD_EXITING was added in 8.0.2.

- Error number: MY-010588; Symbol: ER_RPL_SLAVE_READ_INVALID_EVENT_FROM_MASTER; SQLSTATE: HY000

  Message: Read invalid event from master: '%s', master could be corrupt but a more likely cause of this is a bug

  ER_RPL_SLAVE_READ_INVALID_EVENT_FROM_MASTER was added in 8.0.2.

- Error number: MY-010589; Symbol: ER_RPL_SLAVE_QUEUE_EVENT_FAILED_INVALID_CONFIGURATION; SQLSTATE: HY000

  Message: The queue event failed for channel '%s' as its configuration is invalid.

ER_RPL_SLAVE_QUEUE_EVENT_FAILED_INVALID_CONFIGURATION was added in 8.0.2.

- Error number: MY-010590; Symbol: ER_RPL_SLAVE_IO_THREAD_DETECTED_UNEXPECTED_EVENT_SEQUENCE; SQLSTATE: HY000

  Message: An unexpected event sequence was detected by the IO thread while queuing the event received from master '%s' binary log file, at position %llu.

  ER_RPL_SLAVE_IO_THREAD_DETECTED_UNEXPECTED_EVENT_SEQUENCE was added in 8.0.2.

- Error number: MY-010591; Symbol: ER_RPL_SLAVE_CANT_USE_CHARSET; SQLSTATE: HY000

  Message: '%s' can not be used as client character set. '%s' will be used as default client character set while connecting to master.

  ER_RPL_SLAVE_CANT_USE_CHARSET was added in 8.0.2.

- Error number: MY-010592; Symbol: ER_RPL_SLAVE_CONNECTED_TO_MASTER_REPLICATION_RESUMED; SQLSTATE: HY000

  Message: Slave%s: connected to master '%s@%s:%d',replication resumed in log '%s' at position %s

  ER_RPL_SLAVE_CONNECTED_TO_MASTER_REPLICATION_RESUMED was added in 8.0.2.

- Error number: MY-010593; Symbol: ER_RPL_SLAVE_NEXT_LOG_IS_ACTIVE; SQLSTATE: HY000

  Message: next log '%s' is active

  ER_RPL_SLAVE_NEXT_LOG_IS_ACTIVE was added in 8.0.2.

- Error number: MY-010594; Symbol: ER_RPL_SLAVE_NEXT_LOG_IS_INACTIVE; SQLSTATE: HY000

  Message: next log '%s' is not active

  ER_RPL_SLAVE_NEXT_LOG_IS_INACTIVE was added in 8.0.2.

- Error number: MY-010595; Symbol: ER_RPL_SLAVE_SQL_THREAD_IO_ERROR_READING_EVENT; SQLSTATE: HY000

  Message: Slave SQL thread%s: I/O error reading event (errno: %d cur_log->error: %d)

  ER_RPL_SLAVE_SQL_THREAD_IO_ERROR_READING_EVENT was added in 8.0.2.

- Error number: MY-010596; Symbol: ER_RPL_SLAVE_ERROR_READING_RELAY_LOG_EVENTS; SQLSTATE: HY000

  Message: Error reading relay log event%s: %s

  ER_RPL_SLAVE_ERROR_READING_RELAY_LOG_EVENTS was added in 8.0.2.

- Error number: MY-010597; Symbol: ER_SLAVE_CHANGE_MASTER_TO_EXECUTED; SQLSTATE: HY000

  Message: 'CHANGE MASTER TO%s executed'. Previous state master_host='%s', master_port= %u, master_log_file='%s', master_log_pos= %ld, master_bind='%s'. New state master_host='%s', master_port= %u, master_log_file='%s', master_log_pos= %ld, master_bind='%s'.

  ER_SLAVE_CHANGE_MASTER_TO_EXECUTED was added in 8.0.2.

- Error number: MY-010598; Symbol: ER_RPL_SLAVE_NEW_MASTER_INFO_NEEDS_REPOS_TYPE_OTHER_THAN_FILE; SQLSTATE: HY000

  Message: Slave: Cannot create new master info structure when repositories are of type FILE. Convert slave repositories to TABLE to replicate from multiple sources.

  ER_RPL_SLAVE_NEW_MASTER_INFO_NEEDS_REPOS_TYPE_OTHER_THAN_FILE was added in 8.0.2.

- Error number: MY-010599; Symbol: ER_RPL_FAILED_TO_STAT_LOG_IN_INDEX; SQLSTATE: HY000

  Message: log %s listed in the index, but failed to stat.

  ER_RPL_FAILED_TO_STAT_LOG_IN_INDEX was added in 8.0.2.

- Error number: MY-010600; Symbol: ER_RPL_LOG_NOT_FOUND_WHILE_COUNTING_RELAY_LOG_SPACE; SQLSTATE: HY000

  Message: Could not find first log while counting relay log space.

  ER_RPL_LOG_NOT_FOUND_WHILE_COUNTING_RELAY_LOG_SPACE was added in 8.0.2.

- Error number: MY-010601; Symbol: ER_SLAVE_CANT_USE_TEMPDIR; SQLSTATE: HY000

  Message: Unable to use slave's temporary directory '%s'.

  ER_SLAVE_CANT_USE_TEMPDIR was added in 8.0.2.

- Error number: MY-010602; Symbol: ER_RPL_RELAY_LOG_NEEDS_FILE_NOT_DIRECTORY; SQLSTATE: HY000

  Message: Path '%s' is a directory name, please specify a file name for --relay-log option.

  ER_RPL_RELAY_LOG_NEEDS_FILE_NOT_DIRECTORY was added in 8.0.2.

- Error number: MY-010603; Symbol: ER_RPL_RELAY_LOG_INDEX_NEEDS_FILE_NOT_DIRECTORY; SQLSTATE: HY000

  Message: Path '%s' is a directory name, please specify a file name for --relay-log-index option.

  ER_RPL_RELAY_LOG_INDEX_NEEDS_FILE_NOT_DIRECTORY was added in 8.0.2.

- Error number: MY-010604; Symbol: ER_RPL_PLEASE_USE_OPTION_RELAY_LOG; SQLSTATE: HY000

  Message: Neither --relay-log nor --relay-log-index were used; so replication may break when this MySQL server acts as a slave and has his hostname changed!! Please use '--relay-log=%s' to avoid this problem.

  ER_RPL_PLEASE_USE_OPTION_RELAY_LOG was added in 8.0.2.

- Error number: MY-010605; Symbol: ER_RPL_OPEN_INDEX_FILE_FAILED; SQLSTATE: HY000

  Message: Failed in open_index_file() called from Relay_log_info::rli_init_info().

  ER_RPL_OPEN_INDEX_FILE_FAILED was added in 8.0.2.

- Error number: MY-010606; Symbol: ER_RPL_CANT_INITIALIZE_GTID_SETS_IN_RLI_INIT_INFO; SQLSTATE: HY000

Message: Failed in init_gtid_sets() called from Relay_log_info::rli_init_info().

ER_RPL_CANT_INITIALIZE_GTID_SETS_IN_RLI_INIT_INFO was added in 8.0.2.

- Error number: MY-010607; Symbol: ER_RPL_CANT_OPEN_LOG_IN_RLI_INIT_INFO; SQLSTATE: HY000

  Message: Failed in open_log() called from Relay_log_info::rli_init_info().

  ER_RPL_CANT_OPEN_LOG_IN_RLI_INIT_INFO was added in 8.0.2.

- Error number: MY-010608; Symbol: ER_RPL_ERROR_WRITING_RELAY_LOG_CONFIGURATION; SQLSTATE: HY000

  Message: Error writing relay log configuration.

  ER_RPL_ERROR_WRITING_RELAY_LOG_CONFIGURATION was added in 8.0.2.

- Error number: MY-010609; Symbol: ER_NDB_OOM_GET_NDB_BLOBS_VALUE; SQLSTATE: HY000

  Message: get_ndb_blobs_value: my_malloc(%u) failed

  ER_NDB_OOM_GET_NDB_BLOBS_VALUE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010610; Symbol: ER_NDB_THREAD_TIMED_OUT; SQLSTATE: HY000

  Message: NDB: Thread id %u timed out (30s) waiting for epoch %u/%u to be handled. Progress : %u/%u -> %u/%u.

  ER_NDB_THREAD_TIMED_OUT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010611; Symbol: ER_NDB_TABLE_IS_NOT_DISTRIBUTED; SQLSTATE: HY000

  Message: NDB: Inconsistency detected in distributed privilege tables. Table '%s.%s' is not distributed

  ER_NDB_TABLE_IS_NOT_DISTRIBUTED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010612; Symbol: ER_NDB_CREATING_TABLE; SQLSTATE: HY000

  Message: NDB: Creating %s.%s

  ER_NDB_CREATING_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010613; Symbol: ER_NDB_FLUSHING_TABLE_INFO; SQLSTATE: HY000

  Message: NDB: Flushing %s.%s

  ER_NDB_FLUSHING_TABLE_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010614; Symbol: ER_NDB_CLEANING_STRAY_TABLES; SQLSTATE: HY000

  Message: NDB: Cleaning stray tables from database '%s'

  ER_NDB_CLEANING_STRAY_TABLES was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010615; Symbol: ER_NDB_DISCOVERED_MISSING_DB; SQLSTATE: HY000

  Message: NDB: Discovered missing database '%s'

ER_NDB_DISCOVERED_MISSING_DB was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010616; Symbol: ER_NDB_DISCOVERED_REMAINING_DB; SQLSTATE: HY000

  Message: NDB: Discovered remaining database '%s'

  ER_NDB_DISCOVERED_REMAINING_DB was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010617; Symbol: ER_NDB_CLUSTER_FIND_ALL_DBS_RETRY; SQLSTATE: HY000

  Message: NDB: ndbcluster_find_all_databases retry: %u - %s

  ER_NDB_CLUSTER_FIND_ALL_DBS_RETRY was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010618; Symbol: ER_NDB_CLUSTER_FIND_ALL_DBS_FAIL; SQLSTATE: HY000

  Message: NDB: ndbcluster_find_all_databases fail: %u - %s

  ER_NDB_CLUSTER_FIND_ALL_DBS_FAIL was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010619; Symbol: ER_NDB_SKIPPING_SETUP_TABLE; SQLSTATE: HY000

  Message: NDB: skipping setup table %s.%s, in state %d

  ER_NDB_SKIPPING_SETUP_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010620; Symbol: ER_NDB_FAILED_TO_SET_UP_TABLE; SQLSTATE: HY000

  Message: NDB: failed to setup table %s.%s, error: %d, %s

  ER_NDB_FAILED_TO_SET_UP_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010621; Symbol: ER_NDB_MISSING_FRM_DISCOVERING; SQLSTATE: HY000

  Message: NDB: missing frm for %s.%s, discovering...

  ER_NDB_MISSING_FRM_DISCOVERING was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010622; Symbol: ER_NDB_MISMATCH_IN_FRM_DISCOVERING; SQLSTATE: HY000

  Message: NDB: mismatch in frm for %s.%s, discovering...

  ER_NDB_MISMATCH_IN_FRM_DISCOVERING was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010623; Symbol: ER_NDB_BINLOG_CLEANING_UP_SETUP_LEFTOVERS; SQLSTATE: HY000

  Message: ndb_binlog_setup: Clean up leftovers

  ER_NDB_BINLOG_CLEANING_UP_SETUP_LEFTOVERS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010624; Symbol: ER_NDB_WAITING_INFO; SQLSTATE: HY000

  Message: NDB %s: waiting max %u sec for %s %s. epochs: (%u/%u,%u/%u,%u/%u) injector proc_info: %s

  ER_NDB_WAITING_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010625; Symbol: ER_NDB_WAITING_INFO_WITH_MAP; SQLSTATE: HY000

  Message: NDB %s: waiting max %u sec for %s %s. epochs: (%u/%u,%u/%u,%u/%u) injector proc_info: %s map: %x%08x

  ER_NDB_WAITING_INFO_WITH_MAP was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010626; Symbol: ER_NDB_TIMEOUT_WHILE_DISTRIBUTING; SQLSTATE: HY000

  Message: NDB %s: distributing %s timed out. Ignoring...

  ER_NDB_TIMEOUT_WHILE_DISTRIBUTING was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010627; Symbol: ER_NDB_NOT_WAITING_FOR_DISTRIBUTING; SQLSTATE: HY000

  Message: NDB %s: not waiting for distributing %s

  ER_NDB_NOT_WAITING_FOR_DISTRIBUTING was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010628; Symbol: ER_NDB_DISTRIBUTED_INFO; SQLSTATE: HY000

  Message: NDB: distributed %s.%s(%u/%u) type: %s(%u) query: \'%s\' to %x%08x

  ER_NDB_DISTRIBUTED_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010629; Symbol: ER_NDB_DISTRIBUTION_COMPLETE; SQLSTATE: HY000

  Message: NDB: distribution of %s.%s(%u/%u) type: %s(%u) query: \'%s\' - complete!

  ER_NDB_DISTRIBUTION_COMPLETE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010630; Symbol: ER_NDB_SCHEMA_DISTRIBUTION_FAILED; SQLSTATE: HY000

  Message: NDB Schema dist: Data node: %d failed, subscriber bitmask %x%08x

  ER_NDB_SCHEMA_DISTRIBUTION_FAILED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010631; Symbol: ER_NDB_SCHEMA_DISTRIBUTION_REPORTS_SUBSCRIBE; SQLSTATE: HY000

  Message: NDB Schema dist: Data node: %d reports subscribe from node %d, subscriber bitmask %x %08x

  ER_NDB_SCHEMA_DISTRIBUTION_REPORTS_SUBSCRIBE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010632; Symbol: ER_NDB_SCHEMA_DISTRIBUTION_REPORTS_UNSUBSCRIBE; SQLSTATE: HY000

  Message: NDB Schema dist: Data node: %d reports unsubscribe from node %d, subscriber bitmask %x %08x

  ER_NDB_SCHEMA_DISTRIBUTION_REPORTS_UNSUBSCRIBE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010633; Symbol: ER_NDB_BINLOG_CANT_DISCOVER_TABLE_FROM_SCHEMA_EVENT; SQLSTATE: HY000

  Message: NDB Binlog: Could not discover table '%s.%s' from binlog schema event '%s' from node %d. my_errno: %d

ER_NDB_BINLOG_CANT_DISCOVER_TABLE_FROM_SCHEMA_EVENT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010634; Symbol: ER_NDB_BINLOG_SIGNALLING_UNKNOWN_VALUE; SQLSTATE: HY000

  Message: NDB: unknown value for binlog signalling 0x%X, %s not logged

  ER_NDB_BINLOG_SIGNALLING_UNKNOWN_VALUE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010635; Symbol: ER_NDB_BINLOG_REPLY_TO; SQLSTATE: HY000

  Message: NDB: reply to %s.%s(%u/%u) from %s to %x%08x

  ER_NDB_BINLOG_REPLY_TO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010636; Symbol: ER_NDB_BINLOG_CANT_RELEASE_SLOCK; SQLSTATE: HY000

  Message: NDB: Could not release slock on '%s.%s', Error code: %d Message: %s

  ER_NDB_BINLOG_CANT_RELEASE_SLOCK was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010637; Symbol: ER_NDB_CANT_FIND_TABLE; SQLSTATE: HY000

  Message: NDB schema: Could not find table '%s.%s' in NDB

  ER_NDB_CANT_FIND_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010638; Symbol: ER_NDB_DISCARDING_EVENT_NO_OBJ; SQLSTATE: HY000

  Message: NDB: Discarding event...no obj: %s (%u/%u)

  ER_NDB_DISCARDING_EVENT_NO_OBJ was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010639; Symbol: ER_NDB_DISCARDING_EVENT_ID_VERSION_MISMATCH; SQLSTATE: HY000

  Message: NDB: Discarding event...key: %s non matching id/version [%u/%u] != [%u/%u]

  ER_NDB_DISCARDING_EVENT_ID_VERSION_MISMATCH was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010640; Symbol: ER_NDB_CLEAR_SLOCK_INFO; SQLSTATE: HY000

  Message: NDB: CLEAR_SLOCK key: %s(%u/%u) %x%08x, from %s to %x%08x

  ER_NDB_CLEAR_SLOCK_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010641; Symbol: ER_NDB_BINLOG_SKIPPING_LOCAL_TABLE; SQLSTATE: HY000

  Message: NDB Binlog: Skipping locally defined table '%s.%s' from binlog schema event '%s' from node %d.

  ER_NDB_BINLOG_SKIPPING_LOCAL_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010642; Symbol: ER_NDB_BINLOG_ONLINE_ALTER_RENAME; SQLSTATE: HY000

  Message: NDB Binlog: handling online alter/rename

ER_NDB_BINLOG_ONLINE_ALTER_RENAME was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010643; Symbol: ER_NDB_BINLOG_CANT_REOPEN_SHADOW_TABLE; SQLSTATE: HY000

  Message: NDB Binlog: Failed to re-open shadow table %s.%s

  ER_NDB_BINLOG_CANT_REOPEN_SHADOW_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010644; Symbol: ER_NDB_BINLOG_ONLINE_ALTER_RENAME_COMPLETE; SQLSTATE: HY000

  Message: NDB Binlog: handling online alter/rename done

  ER_NDB_BINLOG_ONLINE_ALTER_RENAME_COMPLETE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010645; Symbol: ER_NDB_BINLOG_SKIPPING_DROP_OF_LOCAL_TABLE; SQLSTATE: HY000

  Message: NDB Binlog: Skipping drop of locally defined table '%s.%s' from binlog schema event '%s' from node %d.

  ER_NDB_BINLOG_SKIPPING_DROP_OF_LOCAL_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010646; Symbol: ER_NDB_BINLOG_SKIPPING_RENAME_OF_LOCAL_TABLE; SQLSTATE: HY000

  Message: NDB Binlog: Skipping renaming locally defined table '%s.%s' from binlog schema event '%s' from node %d.

  ER_NDB_BINLOG_SKIPPING_RENAME_OF_LOCAL_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010647; Symbol: ER_NDB_BINLOG_SKIPPING_DROP_OF_DB_CONTAINING_LOCAL_TABLES; SQLSTATE: HY000

  Message: NDB Binlog: Skipping drop database '%s' since it contained local tables binlog schema event '%s' from node %d.

  ER_NDB_BINLOG_SKIPPING_DROP_OF_DB_CONTAINING_LOCAL_TABLES was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010648; Symbol: ER_NDB_BINLOG_GOT_DIST_PRIV_EVENT_FLUSHING_PRIVILEGES; SQLSTATE: HY000

  Message: Got dist_priv event: %s, flushing privileges

  ER_NDB_BINLOG_GOT_DIST_PRIV_EVENT_FLUSHING_PRIVILEGES was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010649; Symbol: ER_NDB_BINLOG_GOT_SCHEMA_EVENT; SQLSTATE: HY000

  Message: NDB: got schema event on %s.%s(%u/%u) query: '%s' type: %s(%d) node: %u slock: %x %08x

  ER_NDB_BINLOG_GOT_SCHEMA_EVENT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010650; Symbol: ER_NDB_BINLOG_SKIPPING_OLD_SCHEMA_OPERATION; SQLSTATE: HY000

Message: NDB schema: Skipping old schema operation(RENAME_TABLE_NEW) on %s.%s

ER_NDB_BINLOG_SKIPPING_OLD_SCHEMA_OPERATION was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010651; Symbol: ER_NDB_CLUSTER_FAILURE; SQLSTATE: HY000

  Message: NDB Schema dist: cluster failure at epoch %u/%u.

  ER_NDB_CLUSTER_FAILURE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010652; Symbol: ER_NDB_TABLES_INITIALLY_READ_ONLY_ON_RECONNECT; SQLSTATE: HY000

  Message: NDB Binlog: ndb tables initially read only on reconnect.

  ER_NDB_TABLES_INITIALLY_READ_ONLY_ON_RECONNECT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010653; Symbol: ER_NDB_IGNORING_UNKNOWN_EVENT; SQLSTATE: HY000

  Message: NDB Schema dist: unknown event %u, ignoring...

  ER_NDB_IGNORING_UNKNOWN_EVENT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010654; Symbol: ER_NDB_BINLOG_OPENING_INDEX; SQLSTATE: HY000

  Message: NDB Binlog: Opening ndb_binlog_index: %d, '%s'

  ER_NDB_BINLOG_OPENING_INDEX was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010655; Symbol: ER_NDB_BINLOG_CANT_LOCK_NDB_BINLOG_INDEX; SQLSTATE: HY000

  Message: NDB Binlog: Unable to lock table ndb_binlog_index

  ER_NDB_BINLOG_CANT_LOCK_NDB_BINLOG_INDEX was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010656; Symbol: ER_NDB_BINLOG_INJECTING_RANDOM_WRITE_FAILURE; SQLSTATE: HY000

  Message: NDB Binlog: Injecting random write failure

  ER_NDB_BINLOG_INJECTING_RANDOM_WRITE_FAILURE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010657; Symbol: ER_NDB_BINLOG_CANT_WRITE_TO_NDB_BINLOG_INDEX; SQLSTATE: HY000

  Message: NDB Binlog: Failed writing to ndb_binlog_index for epoch %u/%u orig_server_id %u orig_epoch %u/%u with error %d.

  ER_NDB_BINLOG_CANT_WRITE_TO_NDB_BINLOG_INDEX was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010658; Symbol: ER_NDB_BINLOG_WRITING_TO_NDB_BINLOG_INDEX; SQLSTATE: HY000

  Message: NDB Binlog: Writing row (%s) to ndb_binlog_index - %s

  ER_NDB_BINLOG_WRITING_TO_NDB_BINLOG_INDEX was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010659; Symbol: ER_NDB_BINLOG_CANT_COMMIT_TO_NDB_BINLOG_INDEX; SQLSTATE: HY000

  Message: NDB Binlog: Failed committing transaction to ndb_binlog_index with error %d.

  ER_NDB_BINLOG_CANT_COMMIT_TO_NDB_BINLOG_INDEX was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010660; Symbol: ER_NDB_BINLOG_WRITE_TO_NDB_BINLOG_INDEX_FAILED_AFTER_KILL; SQLSTATE: HY000

  Message: NDB Binlog: Failed writing to ndb_binlog_index table while retrying after kill during shutdown

  ER_NDB_BINLOG_WRITE_TO_NDB_BINLOG_INDEX_FAILED_AFTER_KILL was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010661; Symbol: ER_NDB_BINLOG_USING_SERVER_ID_0_SLAVES_WILL_NOT; SQLSTATE: HY000

  Message: NDB: server id set to zero - changes logged to bin log with server id zero will be logged with another server id by slave mysqlds

  ER_NDB_BINLOG_USING_SERVER_ID_0_SLAVES_WILL_NOT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010662; Symbol: ER_NDB_SERVER_ID_RESERVED_OR_TOO_LARGE; SQLSTATE: HY000

  Message: NDB: server id provided is too large to be represented in opt_server_id_bits or is reserved

  ER_NDB_SERVER_ID_RESERVED_OR_TOO_LARGE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010663; Symbol: ER_NDB_BINLOG_NDB_LOG_TRANSACTION_ID_REQUIRES_V2_ROW_EVENTS; SQLSTATE: HY000

  Message: NDB: --ndb-log-transaction-id requires v2 Binlog row events but server is using v1.

  ER_NDB_BINLOG_NDB_LOG_TRANSACTION_ID_REQUIRES_V2_ROW_EVENTS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010664; Symbol: ER_NDB_BINLOG_NDB_LOG_APPLY_STATUS_FORCING_FULL_USE_WRITE; SQLSTATE: HY000

  Message: NDB: ndb-log-apply-status forcing %s.%s to FULL USE_WRITE

  ER_NDB_BINLOG_NDB_LOG_APPLY_STATUS_FORCING_FULL_USE_WRITE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010665; Symbol: ER_NDB_BINLOG_GENERIC_MESSAGE; SQLSTATE: HY000

  Message: NDB Binlog: %s

  ER_NDB_BINLOG_GENERIC_MESSAGE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010666; Symbol: ER_NDB_CONFLICT_GENERIC_MESSAGE; SQLSTATE: HY000

  Message: %s

  ER_NDB_CONFLICT_GENERIC_MESSAGE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010667; Symbol: ER_NDB_TRANS_DEPENDENCY_TRACKER_ERROR; SQLSTATE: HY000

  Message: %s

  ER_NDB_TRANS_DEPENDENCY_TRACKER_ERROR was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010668; Symbol: ER_NDB_CONFLICT_FN_PARSE_ERROR; SQLSTATE: HY000

  Message: NDB Slave: Table %s.%s : Parse error on conflict fn : %s

  ER_NDB_CONFLICT_FN_PARSE_ERROR was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010669; Symbol: ER_NDB_CONFLICT_FN_SETUP_ERROR; SQLSTATE: HY000

  Message: NDB Slave: Table %s.%s : %s

  ER_NDB_CONFLICT_FN_SETUP_ERROR was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010670; Symbol: ER_NDB_BINLOG_FAILED_TO_GET_TABLE; SQLSTATE: HY000

  Message: NDB Binlog: Failed to get table %s from ndb: %s, %d

  ER_NDB_BINLOG_FAILED_TO_GET_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010671; Symbol: ER_NDB_BINLOG_NOT_LOGGING; SQLSTATE: HY000

  Message: NDB Binlog: NOT logging %s

  ER_NDB_BINLOG_NOT_LOGGING was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010672; Symbol: ER_NDB_BINLOG_CREATE_TABLE_EVENT_FAILED; SQLSTATE: HY000

  Message: NDB Binlog: FAILED CREATE (DISCOVER) TABLE Event: %s

  ER_NDB_BINLOG_CREATE_TABLE_EVENT_FAILED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010673; Symbol: ER_NDB_BINLOG_CREATE_TABLE_EVENT_INFO; SQLSTATE: HY000

  Message: NDB Binlog: CREATE (DISCOVER) TABLE Event: %s

  ER_NDB_BINLOG_CREATE_TABLE_EVENT_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010674; Symbol: ER_NDB_BINLOG_DISCOVER_TABLE_EVENT_INFO; SQLSTATE: HY000

  Message: NDB Binlog: DISCOVER TABLE Event: %s

  ER_NDB_BINLOG_DISCOVER_TABLE_EVENT_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010675; Symbol: ER_NDB_BINLOG_BLOB_REQUIRES_PK; SQLSTATE: HY000

  Message: NDB Binlog: logging of table %s with BLOB attribute and no PK is not supported

  ER_NDB_BINLOG_BLOB_REQUIRES_PK was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010676; Symbol: ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB; SQLSTATE: HY000

Message: NDB Binlog: Unable to create event in database. Event: %s Error Code: %d Message: %s

ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010677; Symbol: ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB_AND_CANT_DROP; SQLSTATE: HY000

  Message: NDB Binlog: Unable to create event in database. Attempt to correct with drop failed. Event: %s Error Code: %d Message: %s

  ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB_AND_CANT_DROP was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010678; Symbol: ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB_DROPPED; SQLSTATE: HY000

  Message: NDB Binlog: Unable to create event in database. Attempt to correct with drop ok, but create failed. Event: %s Error Code: %d Message: %s

  ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB_DROPPED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010679; Symbol: ER_NDB_BINLOG_DISCOVER_REUSING_OLD_EVENT_OPS; SQLSTATE: HY000

  Message: NDB Binlog: discover reusing old ev op

  ER_NDB_BINLOG_DISCOVER_REUSING_OLD_EVENT_OPS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010680; Symbol: ER_NDB_BINLOG_CREATING_NDBEVENTOPERATION_FAILED; SQLSTATE: HY000

  Message: NDB Binlog: Creating NdbEventOperation failed for %s

  ER_NDB_BINLOG_CREATING_NDBEVENTOPERATION_FAILED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010681; Symbol: ER_NDB_BINLOG_CANT_CREATE_BLOB; SQLSTATE: HY000

  Message: NDB Binlog: Creating NdbEventOperation blob field %u handles failed (code=%d) for %s

  ER_NDB_BINLOG_CANT_CREATE_BLOB was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010682; Symbol: ER_NDB_BINLOG_NDBEVENT_EXECUTE_FAILED; SQLSTATE: HY000

  Message: NDB Binlog: ndbevent->execute failed for %s; %d %s

  ER_NDB_BINLOG_NDBEVENT_EXECUTE_FAILED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010683; Symbol: ER_NDB_CREATE_EVENT_OPS_LOGGING_INFO; SQLSTATE: HY000

  Message: NDB Binlog: logging %s (%s,%s)

  ER_NDB_CREATE_EVENT_OPS_LOGGING_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010684; Symbol: ER_NDB_BINLOG_CANT_DROP_EVENT_FROM_DB; SQLSTATE: HY000

Message: NDB Binlog: Unable to drop event in database. Event: %s Error Code: %d Message: %s

ER_NDB_BINLOG_CANT_DROP_EVENT_FROM_DB was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010685; Symbol: ER_NDB_TIMED_OUT_IN_DROP_TABLE; SQLSTATE: HY000

  Message: NDB %s: %s timed out. Ignoring...

  ER_NDB_TIMED_OUT_IN_DROP_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010686; Symbol: ER_NDB_BINLOG_UNHANDLED_ERROR_FOR_TABLE; SQLSTATE: HY000

  Message: NDB Binlog: unhandled error %d for table %s

  ER_NDB_BINLOG_UNHANDLED_ERROR_FOR_TABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010687; Symbol: ER_NDB_BINLOG_CLUSTER_FAILURE; SQLSTATE: HY000

  Message: NDB Binlog: cluster failure for %s at epoch %u/%u.

  ER_NDB_BINLOG_CLUSTER_FAILURE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010688; Symbol: ER_NDB_BINLOG_UNKNOWN_NON_DATA_EVENT; SQLSTATE: HY000

  Message: NDB Binlog: unknown non data event %d for %s. Ignoring...

  ER_NDB_BINLOG_UNKNOWN_NON_DATA_EVENT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010689; Symbol: ER_NDB_BINLOG_INJECTOR_DISCARDING_ROW_EVENT_METADATA; SQLSTATE: HY000

  Message: NDB: Binlog Injector discarding row event meta data as server is using v1 row events. (%u %x)

  ER_NDB_BINLOG_INJECTOR_DISCARDING_ROW_EVENT_METADATA was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010690; Symbol: ER_NDB_REMAINING_OPEN_TABLES; SQLSTATE: HY000

  Message: remove_all_event_operations: Remaining open tables:

  ER_NDB_REMAINING_OPEN_TABLES was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010691; Symbol: ER_NDB_REMAINING_OPEN_TABLE_INFO; SQLSTATE: HY000

  Message: %s.%s, use_count: %u

  ER_NDB_REMAINING_OPEN_TABLE_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010692; Symbol: ER_NDB_COULD_NOT_GET_APPLY_STATUS_SHARE; SQLSTATE: HY000

  Message: NDB: Could not get apply status share

  ER_NDB_COULD_NOT_GET_APPLY_STATUS_SHARE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010693; Symbol: ER_NDB_BINLOG_SERVER_SHUTDOWN_DURING_NDB_CLUSTER_START; SQLSTATE: HY000

  Message: NDB Binlog: Server shutdown detected while waiting for ndbcluster to start...

  ER_NDB_BINLOG_SERVER_SHUTDOWN_DURING_NDB_CLUSTER_START was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010694; Symbol: ER_NDB_BINLOG_CLUSTER_RESTARTED_RESET_MASTER_SUGGESTED; SQLSTATE: HY000

  Message: NDB Binlog: cluster has been restarted --initial or with older filesystem. ndb_latest_handled_binlog_epoch: %u/%u, while current epoch: %u/%u. RESET MASTER should be issued. Resetting ndb_latest_handled_binlog_epoch.

  ER_NDB_BINLOG_CLUSTER_RESTARTED_RESET_MASTER_SUGGESTED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010695; Symbol: ER_NDB_BINLOG_CLUSTER_HAS_RECONNECTED; SQLSTATE: HY000

  Message: NDB Binlog: cluster has reconnected. Changes to the database that occurred while disconnected will not be in the binlog

  ER_NDB_BINLOG_CLUSTER_HAS_RECONNECTED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010696; Symbol: ER_NDB_BINLOG_STARTING_LOG_AT_EPOCH; SQLSTATE: HY000

  Message: NDB Binlog: starting log at epoch %u/%u

  ER_NDB_BINLOG_STARTING_LOG_AT_EPOCH was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010697; Symbol: ER_NDB_BINLOG_NDB_TABLES_WRITABLE; SQLSTATE: HY000

  Message: NDB Binlog: ndb tables writable

  ER_NDB_BINLOG_NDB_TABLES_WRITABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010698; Symbol: ER_NDB_BINLOG_SHUTDOWN_DETECTED; SQLSTATE: HY000

  Message: NDB Binlog: Server shutdown detected...

  ER_NDB_BINLOG_SHUTDOWN_DETECTED was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010699; Symbol: ER_NDB_BINLOG_LOST_SCHEMA_CONNECTION_WAITING; SQLSTATE: HY000

  Message: NDB Binlog: Just lost schema connection, hanging around

  ER_NDB_BINLOG_LOST_SCHEMA_CONNECTION_WAITING was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010700; Symbol: ER_NDB_BINLOG_LOST_SCHEMA_CONNECTION_CONTINUING; SQLSTATE: HY000

  Message: NDB Binlog: ...and on our way

  ER_NDB_BINLOG_LOST_SCHEMA_CONNECTION_CONTINUING was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010701; Symbol: ER_NDB_BINLOG_ERROR_HANDLING_SCHEMA_EVENT; SQLSTATE: HY000

  Message: NDB: error %lu (%s) on handling binlog schema event

  ER_NDB_BINLOG_ERROR_HANDLING_SCHEMA_EVENT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010702; Symbol: ER_NDB_BINLOG_CANT_INJECT_APPLY_STATUS_WRITE_ROW; SQLSTATE: HY000

  Message: NDB Binlog: Failed to inject apply status write row

  ER_NDB_BINLOG_CANT_INJECT_APPLY_STATUS_WRITE_ROW was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010703; Symbol: ER_NDB_BINLOG_ERROR_DURING_GCI_ROLLBACK; SQLSTATE: HY000

  Message: NDB Binlog: Error during ROLLBACK of GCI %u/%u. Error: %d

  ER_NDB_BINLOG_ERROR_DURING_GCI_ROLLBACK was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010704; Symbol: ER_NDB_BINLOG_ERROR_DURING_GCI_COMMIT; SQLSTATE: HY000

  Message: NDB Binlog: Error during COMMIT of GCI. Error: %d

  ER_NDB_BINLOG_ERROR_DURING_GCI_COMMIT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010705; Symbol: ER_NDB_BINLOG_LATEST_TRX_IN_EPOCH_NOT_IN_BINLOG; SQLSTATE: HY000

  Message: NDB Binlog: latest transaction in epoch %u/%u not in binlog as latest handled epoch is %u/%u

  ER_NDB_BINLOG_LATEST_TRX_IN_EPOCH_NOT_IN_BINLOG was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010706; Symbol: ER_NDB_BINLOG_RELEASING_EXTRA_SHARE_REFERENCES; SQLSTATE: HY000

  Message: NDB Binlog: Release extra share references

  ER_NDB_BINLOG_RELEASING_EXTRA_SHARE_REFERENCES was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010707; Symbol: ER_NDB_BINLOG_REMAINING_OPEN_TABLES; SQLSTATE: HY000

  Message: NDB Binlog: remaining open tables:

  ER_NDB_BINLOG_REMAINING_OPEN_TABLES was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010708; Symbol: ER_NDB_BINLOG_REMAINING_OPEN_TABLE_INFO; SQLSTATE: HY000

  Message: %s.%s state: %u use_count: %u

  ER_NDB_BINLOG_REMAINING_OPEN_TABLE_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010709; Symbol: ER_TREE_CORRUPT_PARENT_SHOULD_POINT_AT_PARENT; SQLSTATE: HY000

  Message: Wrong tree: Parent doesn't point at parent

  ER_TREE_CORRUPT_PARENT_SHOULD_POINT_AT_PARENT was added in 8.0.2.

- Error number: MY-010710; Symbol: ER_TREE_CORRUPT_ROOT_SHOULD_BE_BLACK; SQLSTATE: HY000

  Message: Wrong tree: Root should be black

  ER_TREE_CORRUPT_ROOT_SHOULD_BE_BLACK was added in 8.0.2.

- Error number: MY-010711; Symbol: ER_TREE_CORRUPT_2_CONSECUTIVE_REDS; SQLSTATE: HY000

  Message: Wrong tree: Found two red in a row

  ER_TREE_CORRUPT_2_CONSECUTIVE_REDS was added in 8.0.2.

- Error number: MY-010712; Symbol: ER_TREE_CORRUPT_RIGHT_IS_LEFT; SQLSTATE: HY000

  Message: Wrong tree: Found right == left

  ER_TREE_CORRUPT_RIGHT_IS_LEFT was added in 8.0.2.

- Error number: MY-010713; Symbol: ER_TREE_CORRUPT_INCORRECT_BLACK_COUNT; SQLSTATE: HY000

  Message: Wrong tree: Incorrect black-count: %d - %d

  ER_TREE_CORRUPT_INCORRECT_BLACK_COUNT was added in 8.0.2.

- Error number: MY-010714; Symbol: ER_WRONG_COUNT_FOR_ORIGIN; SQLSTATE: HY000

  Message: Use_count: Wrong count %lu for origin %p

  ER_WRONG_COUNT_FOR_ORIGIN was added in 8.0.2.

- Error number: MY-010715; Symbol: ER_WRONG_COUNT_FOR_KEY; SQLSTATE: HY000

  Message: Use_count: Wrong count for key at %p, %lu should be %lu

  ER_WRONG_COUNT_FOR_KEY was added in 8.0.2.

- Error number: MY-010716; Symbol: ER_WRONG_COUNT_OF_ELEMENTS; SQLSTATE: HY000

  Message: Wrong number of elements: %u (should be %u) for tree at %p

  ER_WRONG_COUNT_OF_ELEMENTS was added in 8.0.2.

- Error number: MY-010717; Symbol: ER_RPL_ERROR_READING_SLAVE_WORKER_CONFIGURATION; SQLSTATE: HY000

  Message: Error reading slave worker configuration

  ER_RPL_ERROR_READING_SLAVE_WORKER_CONFIGURATION was added in 8.0.2.

- Error number: MY-010718; Symbol: ER_RPL_ERROR_WRITING_SLAVE_WORKER_CONFIGURATION; SQLSTATE: HY000

  Message: Error writing slave worker configuration

  ER_RPL_ERROR_WRITING_SLAVE_WORKER_CONFIGURATION was added in 8.0.2.

- Error number: MY-010719; Symbol: ER_RPL_FAILED_TO_OPEN_RELAY_LOG; SQLSTATE: HY000

  Message: Failed to open relay log %s, error: %s

  ER_RPL_FAILED_TO_OPEN_RELAY_LOG was added in 8.0.2.

- Error number: MY-010720; Symbol: ER_RPL_WORKER_CANT_READ_RELAY_LOG; SQLSTATE: HY000

  Message: Error when worker read relay log events,relay log name %s, position %llu

  ER_RPL_WORKER_CANT_READ_RELAY_LOG was added in 8.0.2.

- Error number: MY-010721; Symbol: ER_RPL_WORKER_CANT_FIND_NEXT_RELAY_LOG; SQLSTATE: HY000

  Message: Failed to find next relay log when retrying the transaction, current relay log is %s

  ER_RPL_WORKER_CANT_FIND_NEXT_RELAY_LOG was added in 8.0.2.

- Error number: MY-010722; Symbol: ER_RPL_MTS_SLAVE_COORDINATOR_HAS_WAITED; SQLSTATE: HY000

  Message: Multi-threaded slave: Coordinator has waited %lu times hitting slave_pending_jobs_size_max; current event size = %zu.

  ER_RPL_MTS_SLAVE_COORDINATOR_HAS_WAITED was added in 8.0.2.

- Error number: MY-010723; Symbol: ER_BINLOG_FAILED_TO_WRITE_DROP_FOR_TEMP_TABLES; SQLSTATE: HY000

  Message: Failed to write the DROP statement for temporary tables to binary log

  ER_BINLOG_FAILED_TO_WRITE_DROP_FOR_TEMP_TABLES was added in 8.0.2.

- Error number: MY-010724; Symbol: ER_BINLOG_OOM_WRITING_DELETE_WHILE_OPENING_HEAP_TABLE; SQLSTATE: HY000

  Message: When opening HEAP table, could not allocate memory to write 'DELETE FROM `%s`.`%s`' to the binary log

  ER_BINLOG_OOM_WRITING_DELETE_WHILE_OPENING_HEAP_TABLE was added in 8.0.2.

- Error number: MY-010725; Symbol: ER_FAILED_TO_REPAIR_TABLE; SQLSTATE: HY000

  Message: Couldn't repair table: %s.%s

  ER_FAILED_TO_REPAIR_TABLE was added in 8.0.2.

- Error number: MY-010726; Symbol: ER_FAILED_TO_REMOVE_TEMP_TABLE; SQLSTATE: HY000

  Message: Could not remove temporary table: '%s', error: %d

ER_FAILED_TO_REMOVE_TEMP_TABLE was added in 8.0.2.

- Error number: MY-010727; Symbol: ER_SYSTEM_TABLE_NOT_TRANSACTIONAL; SQLSTATE: HY000

  Message: System table '%.*s' is expected to be transactional.

  ER_SYSTEM_TABLE_NOT_TRANSACTIONAL was added in 8.0.2.

- Error number: MY-010728; Symbol: ER_RPL_ERROR_WRITING_MASTER_CONFIGURATION; SQLSTATE: HY000

  Message: Error writing master configuration.

  ER_RPL_ERROR_WRITING_MASTER_CONFIGURATION was added in 8.0.2.

- Error number: MY-010729; Symbol: ER_RPL_ERROR_READING_MASTER_CONFIGURATION; SQLSTATE: HY000

  Message: Error reading master configuration.

  ER_RPL_ERROR_READING_MASTER_CONFIGURATION was added in 8.0.2.

- Error number: MY-010730; Symbol: ER_RPL_SSL_INFO_IN_MASTER_INFO_IGNORED; SQLSTATE: HY000

  Message: SSL information in the master info file are ignored because this MySQL slave was compiled without SSL support.

  ER_RPL_SSL_INFO_IN_MASTER_INFO_IGNORED was added in 8.0.2.

- Error number: MY-010731; Symbol: ER_PLUGIN_FAILED_DEINITIALIZATION; SQLSTATE: HY000

  Message: Plugin '%s' of type %s failed deinitialization

  ER_PLUGIN_FAILED_DEINITIALIZATION was added in 8.0.2.

- Error number: MY-010732; Symbol: ER_PLUGIN_HAS_NONZERO_REFCOUNT_AFTER_DEINITIALIZATION; SQLSTATE: HY000

  Message: Plugin '%s' has ref_count=%d after deinitialization.

  ER_PLUGIN_HAS_NONZERO_REFCOUNT_AFTER_DEINITIALIZATION was added in 8.0.2.

- Error number: MY-010733; Symbol: ER_PLUGIN_SHUTTING_DOWN_PLUGIN; SQLSTATE: HY000

  Message: Shutting down plugin '%s'

  ER_PLUGIN_SHUTTING_DOWN_PLUGIN was added in 8.0.2.

- Error number: MY-010734; Symbol: ER_PLUGIN_REGISTRATION_FAILED; SQLSTATE: HY000

  Message: Plugin '%s' registration as a %s failed.

  ER_PLUGIN_REGISTRATION_FAILED was added in 8.0.2.

- Error number: MY-010735; Symbol: ER_PLUGIN_CANT_OPEN_PLUGIN_TABLE; SQLSTATE: HY000

  Message: Could not open the mysql.plugin table. Please perform the MySQL upgrade procedure.

ER_PLUGIN_CANT_OPEN_PLUGIN_TABLE was added in 8.0.2.

- Error number: MY-010736; Symbol: ER_PLUGIN_CANT_LOAD; SQLSTATE: HY000

  Message: Couldn't load plugin named '%s' with soname '%s'.

  ER_PLUGIN_CANT_LOAD was added in 8.0.2.

- Error number: MY-010737; Symbol: ER_PLUGIN_LOAD_PARAMETER_TOO_LONG; SQLSTATE: HY000

  Message: plugin-load parameter too long

  ER_PLUGIN_LOAD_PARAMETER_TOO_LONG was added in 8.0.2.

- Error number: MY-010738; Symbol: ER_PLUGIN_FORCING_SHUTDOWN; SQLSTATE: HY000

  Message: Plugin '%s' will be forced to shutdown

  ER_PLUGIN_FORCING_SHUTDOWN was added in 8.0.2.

- Error number: MY-010739; Symbol: ER_PLUGIN_HAS_NONZERO_REFCOUNT_AFTER_SHUTDOWN; SQLSTATE: HY000

  Message: Plugin '%s' has ref_count=%d after shutdown.

  ER_PLUGIN_HAS_NONZERO_REFCOUNT_AFTER_SHUTDOWN was added in 8.0.2.

- Error number: MY-010740; Symbol: ER_PLUGIN_UNKNOWN_VARIABLE_TYPE; SQLSTATE: HY000

  Message: Unknown variable type code 0x%x in plugin '%s'.

  ER_PLUGIN_UNKNOWN_VARIABLE_TYPE was added in 8.0.2.

- Error number: MY-010741; Symbol: ER_PLUGIN_VARIABLE_SET_READ_ONLY; SQLSTATE: HY000

  Message: Server variable %s of plugin %s was forced to be read-only: string variable without update_func and PLUGIN_VAR_MEMALLOC flag

  ER_PLUGIN_VARIABLE_SET_READ_ONLY was added in 8.0.2.

- Error number: MY-010742; Symbol: ER_PLUGIN_VARIABLE_MISSING_NAME; SQLSTATE: HY000

  Message: Missing variable name in plugin '%s'.

  ER_PLUGIN_VARIABLE_MISSING_NAME was added in 8.0.2.

- Error number: MY-010743; Symbol: ER_PLUGIN_VARIABLE_NOT_ALLOCATED_THREAD_LOCAL; SQLSTATE: HY000

  Message: Thread local variable '%s' not allocated in plugin '%s'.

  ER_PLUGIN_VARIABLE_NOT_ALLOCATED_THREAD_LOCAL was added in 8.0.2.

- Error number: MY-010744; Symbol: ER_PLUGIN_OOM; SQLSTATE: HY000

  Message: Out of memory for plugin '%s'.

  ER_PLUGIN_OOM was added in 8.0.2.

- Error number: MY-010745; Symbol: ER_PLUGIN_BAD_OPTIONS; SQLSTATE: HY000

  Message: Bad options for plugin '%s'.

  ER_PLUGIN_BAD_OPTIONS was added in 8.0.2.

- Error number: MY-010746; Symbol: ER_PLUGIN_PARSING_OPTIONS_FAILED; SQLSTATE: HY000

  Message: Parsing options for plugin '%s' failed.

  ER_PLUGIN_PARSING_OPTIONS_FAILED was added in 8.0.2.

- Error number: MY-010747; Symbol: ER_PLUGIN_DISABLED; SQLSTATE: HY000

  Message: Plugin '%s' is disabled.

  ER_PLUGIN_DISABLED was added in 8.0.2.

- Error number: MY-010748; Symbol: ER_PLUGIN_HAS_CONFLICTING_SYSTEM_VARIABLES; SQLSTATE: HY000

  Message: Plugin '%s' has conflicting system variables

  ER_PLUGIN_HAS_CONFLICTING_SYSTEM_VARIABLES was added in 8.0.2.

- Error number: MY-010749; Symbol: ER_PLUGIN_CANT_SET_PERSISTENT_OPTIONS; SQLSTATE: HY000

  Message: Setting persistent options for plugin '%s' failed.

  ER_PLUGIN_CANT_SET_PERSISTENT_OPTIONS was added in 8.0.2.

- Error number: MY-010750; Symbol: ER_MY_NET_WRITE_FAILED_FALLING_BACK_ON_STDERR; SQLSTATE: HY000

  Message: Failed on my_net_write, writing to stderr instead: %s

  ER_MY_NET_WRITE_FAILED_FALLING_BACK_ON_STDERR was added in 8.0.2.

- Error number: MY-010751; Symbol: ER_RETRYING_REPAIR_WITHOUT_QUICK; SQLSTATE: HY000

  Message: Retrying repair of: '%s' without quick

  ER_RETRYING_REPAIR_WITHOUT_QUICK was added in 8.0.2.

- Error number: MY-010752; Symbol: ER_RETRYING_REPAIR_WITH_KEYCACHE; SQLSTATE: HY000

  Message: Retrying repair of: '%s' with keycache

  ER_RETRYING_REPAIR_WITH_KEYCACHE was added in 8.0.2.

- Error number: MY-010753; Symbol: ER_FOUND_ROWS_WHILE_REPAIRING; SQLSTATE: HY000

  Message: Found %s of %s rows when repairing '%s'

  ER_FOUND_ROWS_WHILE_REPAIRING was added in 8.0.2.

- Error number: MY-010754; Symbol: ER_ERROR_DURING_OPTIMIZE_TABLE; SQLSTATE: HY000

  Message: Warning: Optimize table got errno %d on %s.%s, retrying

ER_ERROR_DURING_OPTIMIZE_TABLE was added in 8.0.2.

- Error number: MY-010755; Symbol: ER_ERROR_ENABLING_KEYS; SQLSTATE: HY000

  Message: Warning: Enabling keys got errno %d on %s.%s, retrying

  ER_ERROR_ENABLING_KEYS was added in 8.0.2.

- Error number: MY-010756; Symbol: ER_CHECKING_TABLE; SQLSTATE: HY000

  Message: Checking table: '%s'

  ER_CHECKING_TABLE was added in 8.0.2.

- Error number: MY-010757; Symbol: ER_RECOVERING_TABLE; SQLSTATE: HY000

  Message: Recovering table: '%s'

  ER_RECOVERING_TABLE was added in 8.0.2.

- Error number: MY-010758; Symbol: ER_CANT_CREATE_TABLE_SHARE_FROM_FRM; SQLSTATE: HY000

  Message: Error in creating TABLE_SHARE from %s.frm file.

  ER_CANT_CREATE_TABLE_SHARE_FROM_FRM was added in 8.0.2.

- Error number: MY-010759; Symbol: ER_CANT_LOCK_TABLE; SQLSTATE: HY000

  Message: Unable to acquire lock on %s.%s

  ER_CANT_LOCK_TABLE was added in 8.0.2.

- Error number: MY-010760; Symbol: ER_CANT_ALLOC_TABLE_OBJECT; SQLSTATE: HY000

  Message: Error in allocation memory for TABLE object.

  ER_CANT_ALLOC_TABLE_OBJECT was added in 8.0.2.

- Error number: MY-010761; Symbol: ER_CANT_CREATE_HANDLER_OBJECT_FOR_TABLE; SQLSTATE: HY000

  Message: Error in creating handler object for table %s.%s

  ER_CANT_CREATE_HANDLER_OBJECT_FOR_TABLE was added in 8.0.2.

- Error number: MY-010762; Symbol: ER_CANT_SET_HANDLER_REFERENCE_FOR_TABLE; SQLSTATE: HY000

  Message: Error in setting handler reference for table %s.%s

  ER_CANT_SET_HANDLER_REFERENCE_FOR_TABLE was added in 8.0.2.

- Error number: MY-010763; Symbol: ER_CANT_LOCK_TABLESPACE; SQLSTATE: HY000

  Message: Unable to acquire lock on tablespace name %s

  ER_CANT_LOCK_TABLESPACE was added in 8.0.2.

- Error number: MY-010764; Symbol: ER_CANT_UPGRADE_GENERATED_COLUMNS_TO_DD; SQLSTATE: HY000

  Message: Error in processing generated columns for table %s.%s

  ER_CANT_UPGRADE_GENERATED_COLUMNS_TO_DD was added in 8.0.2.

- Error number: MY-010765; Symbol: ER_DD_ERROR_CREATING_ENTRY; SQLSTATE: HY000

  Message: Error in Creating DD entry for %s.%s

  ER_DD_ERROR_CREATING_ENTRY was added in 8.0.2.

- Error number: MY-010766; Symbol: ER_DD_CANT_FETCH_TABLE_DATA; SQLSTATE: HY000

  Message: Error in fetching %s.%s table data from dictionary

  ER_DD_CANT_FETCH_TABLE_DATA was added in 8.0.2.

- Error number: MY-010767; Symbol: ER_DD_CANT_FIX_SE_DATA; SQLSTATE: HY000

  Message: Error in fixing SE data for %s.%s

  ER_DD_CANT_FIX_SE_DATA was added in 8.0.2.

- Error number: MY-010768; Symbol: ER_DD_CANT_CREATE_SP; SQLSTATE: HY000

  Message: Error in creating stored program '%s.%s'

  ER_DD_CANT_CREATE_SP was added in 8.0.2.

- Error number: MY-010769; Symbol: ER_CANT_OPEN_DB_OPT_USING_DEFAULT_CHARSET; SQLSTATE: HY000

  Message: Unable to open db.opt file %s. Using default Character set.

  ER_CANT_OPEN_DB_OPT_USING_DEFAULT_CHARSET was added in 8.0.2.

- Error number: MY-010770; Symbol: ER_CANT_CREATE_CACHE_FOR_DB_OPT; SQLSTATE: HY000

  Message: Unable to intialize IO cache to open db.opt file %s.

  ER_CANT_CREATE_CACHE_FOR_DB_OPT was added in 8.0.2.

- Error number: MY-010771; Symbol: ER_CANT_IDENTIFY_CHARSET_USING_DEFAULT; SQLSTATE: HY000

  Message: Unable to identify the charset in %s. Using default character set.

  ER_CANT_IDENTIFY_CHARSET_USING_DEFAULT was added in 8.0.2.

- Error number: MY-010772; Symbol: ER_DB_OPT_NOT_FOUND_USING_DEFAULT_CHARSET; SQLSTATE: HY000

  Message: db.opt file not found for %s database. Using default Character set.

  ER_DB_OPT_NOT_FOUND_USING_DEFAULT_CHARSET was added in 8.0.2.

- Error number: MY-010773; Symbol: ER_EVENT_CANT_GET_TIMEZONE_FROM_FIELD; SQLSTATE: HY000

Message: Event '%s'.'%s': invalid value in column mysql.event.time_zone.

This error can occur for upgrades to MySQL 8.0 from previous versions. During the upgrade process, the server reads the `mysql.event` system table and migrates its contents to the internal data dictionary.

`ER_EVENT_CANT_GET_TIMEZONE_FROM_FIELD` was added in 8.0.2.

- Error number: `MY-010774`; Symbol: `ER_EVENT_CANT_FIND_TIMEZONE`; SQLSTATE: `HY000`

  Message: Event '%s'.'%s': has invalid time zone value

  This error can occur for upgrades to MySQL 8.0 from previous versions. During the upgrade process, the server reads the `mysql.event` system table and migrates its contents to the internal data dictionary.

  `ER_EVENT_CANT_FIND_TIMEZONE` was added in 8.0.2.

- Error number: `MY-010775`; Symbol: `ER_EVENT_CANT_GET_CHARSET`; SQLSTATE: `HY000`

  Message: Event '%s'.'%s': invalid value in column mysql.event.character_set_client.

  This error can occur for upgrades to MySQL 8.0 from previous versions. During the upgrade process, the server reads the `mysql.event` system table and migrates its contents to the internal data dictionary.

  `ER_EVENT_CANT_GET_CHARSET` was added in 8.0.2.

- Error number: `MY-010776`; Symbol: `ER_EVENT_CANT_GET_COLLATION`; SQLSTATE: `HY000`

  Message: Event '%s'.'%s': invalid value in column mysql.event.collation_connection.

  This error can occur for upgrades to MySQL 8.0 from previous versions. During the upgrade process, the server reads the `mysql.event` system table and migrates its contents to the internal data dictionary.

  `ER_EVENT_CANT_GET_COLLATION` was added in 8.0.2.

- Error number: `MY-010777`; Symbol: `ER_EVENT_CANT_OPEN_TABLE_MYSQL_EVENT`; SQLSTATE: `HY000`

  Message: Failed to open mysql.event Table.

  This error can occur for upgrades to MySQL 8.0 from previous versions. During the upgrade process, the server reads the `mysql.event` system table and migrates its contents to the internal data dictionary.

  `ER_EVENT_CANT_OPEN_TABLE_MYSQL_EVENT` was added in 8.0.2.

- Error number: `MY-010778`; Symbol: `ER_CANT_PARSE_STORED_ROUTINE_BODY`; SQLSTATE: `HY000`

  Message: Parsing '%s.%s' routine body failed.%s

  `ER_CANT_PARSE_STORED_ROUTINE_BODY` was added in 8.0.2.

- Error number: `MY-010779`; Symbol: `ER_CANT_OPEN_TABLE_MYSQL_PROC`; SQLSTATE: `HY000`

  Message: Failed to open mysql.proc Table.

  This error can occur for upgrades to MySQL 8.0 from previous versions. During the upgrade process, the server reads the `mysql.proc` system table and migrates its contents to the internal data dictionary.

  `ER_CANT_OPEN_TABLE_MYSQL_PROC` was added in 8.0.2.

- Error number: MY-010780; Symbol: ER_CANT_READ_TABLE_MYSQL_PROC; SQLSTATE: HY000

  Message: Failed to read mysql.proc table.

  This error can occur for upgrades to MySQL 8.0 from previous versions. During the upgrade process, the server reads the mysql.proc system table and migrates its contents to the internal data dictionary.

  ER_CANT_READ_TABLE_MYSQL_PROC was added in 8.0.2.

- Error number: MY-010781; Symbol: ER_FILE_EXISTS_DURING_UPGRADE; SQLSTATE: HY000

  Message: Found %s file in mysql schema. DD will create .ibd file with same name. Please rename table and start upgrade process again.

  ER_FILE_EXISTS_DURING_UPGRADE was added in 8.0.2.

- Error number: MY-010782; Symbol: ER_CANT_OPEN_DATADIR_AFTER_UPGRADE_FAILURE; SQLSTATE: HY000

  Message: Unable to open the data directory %s during clean up after upgrade failed

  ER_CANT_OPEN_DATADIR_AFTER_UPGRADE_FAILURE was added in 8.0.2.

- Error number: MY-010783; Symbol: ER_CANT_SET_PATH_FOR; SQLSTATE: HY000

  Message: Failed to set path %s

  ER_CANT_SET_PATH_FOR was added in 8.0.2.

- Error number: MY-010784; Symbol: ER_CANT_OPEN_DIR; SQLSTATE: HY000

  Message: Failed to open dir %s

  ER_CANT_OPEN_DIR was added in 8.0.2.

- Error number: MY-010785; Symbol: ER_NDB_EMPTY_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS; SQLSTATE: HY000

  Message: NDB: Found empty nodeid specified in --ndb-cluster-connection-pool-nodeids='%s'.

  ER_NDB_EMPTY_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010786; Symbol: ER_NDB_CANT_PARSE_NDB_CLUSTER_CONNECTION_POOL_NODEIDS; SQLSTATE: HY000

  Message: NDB: Could not parse '%s' in --ndb-cluster-connection-pool-nodeids='%s'.

  ER_NDB_CANT_PARSE_NDB_CLUSTER_CONNECTION_POOL_NODEIDS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010787; Symbol: ER_NDB_INVALID_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS; SQLSTATE: HY000

  Message: NDB: Invalid nodeid %d in --ndb-cluster-connection-pool-nodeids='%s'.

  ER_NDB_INVALID_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010788; Symbol: ER_NDB_DUPLICATE_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS; SQLSTATE: HY000

  Message: NDB: Found duplicate nodeid %d in --ndb-cluster-connection-pool-nodeids='%s'.

  ER_NDB_DUPLICATE_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010789; Symbol: ER_NDB_POOL_SIZE_MUST_MATCH_NDB_CLUSTER_CONNECTION_POOL_NODEIDS; SQLSTATE: HY000

  Message: NDB: The size of the cluster connection pool must be equal to the number of nodeids in --ndb-cluster-connection-pool-nodeids='%s'.

  ER_NDB_POOL_SIZE_MUST_MATCH_NDB_CLUSTER_CONNECTION_POOL_NODEIDS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010790; Symbol: ER_NDB_NODEID_NOT_FIRST_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS; SQLSTATE: HY000

  Message: NDB: The nodeid specified by --ndb-nodeid must be equal to the first nodeid in --ndb-cluster-connection-pool-nodeids='%s'.

  ER_NDB_NODEID_NOT_FIRST_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010791; Symbol: ER_NDB_USING_NODEID; SQLSTATE: HY000

  Message: NDB: using nodeid %u

  ER_NDB_USING_NODEID was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010792; Symbol: ER_NDB_CANT_ALLOC_GLOBAL_NDB_CLUSTER_CONNECTION; SQLSTATE: HY000

  Message: NDB: failed to allocate global ndb cluster connection

  ER_NDB_CANT_ALLOC_GLOBAL_NDB_CLUSTER_CONNECTION was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010793; Symbol: ER_NDB_CANT_ALLOC_GLOBAL_NDB_OBJECT; SQLSTATE: HY000

  Message: NDB: failed to allocate global ndb object

  ER_NDB_CANT_ALLOC_GLOBAL_NDB_OBJECT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010794; Symbol: ER_NDB_USING_NODEID_LIST; SQLSTATE: HY000

  Message: NDB[%u]: using nodeid %u

  ER_NDB_USING_NODEID_LIST was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010795; Symbol: ER_NDB_CANT_ALLOC_NDB_CLUSTER_CONNECTION; SQLSTATE: HY000

Message: NDB[%u]: failed to allocate cluster connect object

ER_NDB_CANT_ALLOC_NDB_CLUSTER_CONNECTION was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010796; Symbol: ER_NDB_STARTING_CONNECT_THREAD; SQLSTATE: HY000

  Message: NDB[%u]: starting connect thread

  ER_NDB_STARTING_CONNECT_THREAD was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010797; Symbol: ER_NDB_NODE_INFO; SQLSTATE: HY000

  Message: NDB[%u]: NodeID: %d, %s

  ER_NDB_NODE_INFO was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010798; Symbol: ER_NDB_CANT_START_CONNECT_THREAD; SQLSTATE: HY000

  Message: NDB[%u]: failed to start connect thread

  ER_NDB_CANT_START_CONNECT_THREAD was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010799; Symbol: ER_NDB_GENERIC_ERROR; SQLSTATE: HY000

  Message: NDB: error (%u) %s

  ER_NDB_GENERIC_ERROR was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010800; Symbol: ER_NDB_CPU_MASK_TOO_SHORT; SQLSTATE: HY000

  Message: Ignored receive thread CPU mask, mask too short, %u CPUs needed in mask, only %u CPUs provided

  ER_NDB_CPU_MASK_TOO_SHORT was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010801; Symbol: ER_EVENT_ERROR_CREATING_QUERY_TO_WRITE_TO_BINLOG; SQLSTATE: HY000

  Message: Event Error: An error occurred while creating query string, before writing it into binary log.

  ER_EVENT_ERROR_CREATING_QUERY_TO_WRITE_TO_BINLOG was added in 8.0.2.

- Error number: MY-010802; Symbol: ER_EVENT_SCHEDULER_ERROR_LOADING_FROM_DB; SQLSTATE: HY000

  Message: Event Scheduler: Error while loading from disk.

  ER_EVENT_SCHEDULER_ERROR_LOADING_FROM_DB was added in 8.0.2.

- Error number: MY-010803; Symbol: ER_EVENT_SCHEDULER_ERROR_GETTING_EVENT_OBJECT; SQLSTATE: HY000

  Message: Event Scheduler: Error getting event object.

  ER_EVENT_SCHEDULER_ERROR_GETTING_EVENT_OBJECT was added in 8.0.2.

- Error number: MY-010804; Symbol: ER_EVENT_SCHEDULER_GOT_BAD_DATA_FROM_TABLE; SQLSTATE: HY000

Message: Event Scheduler: Error while loading events from mysql.events.The table probably contains bad data or is corrupted

ER_EVENT_SCHEDULER_GOT_BAD_DATA_FROM_TABLE was added in 8.0.2.

- Error number: MY-010805; Symbol: ER_EVENT_CANT_GET_LOCK_FOR_DROPPING_EVENT; SQLSTATE: HY000

  Message: Unable to obtain lock for dropping event %s from schema %s

  ER_EVENT_CANT_GET_LOCK_FOR_DROPPING_EVENT was added in 8.0.2.

- Error number: MY-010806; Symbol: ER_EVENT_UNABLE_TO_DROP_EVENT; SQLSTATE: HY000

  Message: Unable to drop event %s from schema %s

  ER_EVENT_UNABLE_TO_DROP_EVENT was added in 8.0.2.

- Error number: MY-010807; Symbol: ER_BINLOG_ATTACHING_THREAD_MEMORY_FINALLY_AVAILABLE; SQLSTATE: HY000

  Message: Server overcomes the temporary 'out of memory' in '%d' tries while attaching to session thread during the group commit phase.

  ER_BINLOG_ATTACHING_THREAD_MEMORY_FINALLY_AVAILABLE was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010808; Symbol: ER_BINLOG_CANT_RESIZE_CACHE; SQLSTATE: HY000

  Message: Unable to resize binlog IOCACHE auxiliary file

  ER_BINLOG_CANT_RESIZE_CACHE was added in 8.0.2.

- Error number: MY-010809; Symbol: ER_BINLOG_FILE_BEING_READ_NOT_PURGED; SQLSTATE: HY000

  Message: file %s was not purged because it was being read by thread number %u

  ER_BINLOG_FILE_BEING_READ_NOT_PURGED was added in 8.0.2.

- Error number: MY-010810; Symbol: ER_BINLOG_IO_ERROR_READING_HEADER; SQLSTATE: HY000

  Message: I/O error reading the header from the binary log, errno=%d, io cache code=%d

  ER_BINLOG_IO_ERROR_READING_HEADER was added in 8.0.2.

- Error number: MY-010811; Symbol: ER_BINLOG_CANT_OPEN_LOG; SQLSTATE: HY000

  Message: Failed to open log (file '%s', errno %d)

  ER_BINLOG_CANT_OPEN_LOG was added in 8.0.2, removed after 8.0.12.

- Error number: MY-010812; Symbol: ER_BINLOG_CANT_CREATE_CACHE_FOR_LOG; SQLSTATE: HY000

  Message: Failed to create a cache on log (file '%s')

  ER_BINLOG_CANT_CREATE_CACHE_FOR_LOG was added in 8.0.2, removed after 8.0.12.

- Error number: MY-010813; Symbol: ER_BINLOG_FILE_EXTENSION_NUMBER_EXHAUSTED; SQLSTATE: HY000

  Message: Log filename extension number exhausted: %06lu. Please fix this by archiving old logs and updating the index files.

  ER_BINLOG_FILE_EXTENSION_NUMBER_EXHAUSTED was added in 8.0.2.

- Error number: MY-010814; Symbol: ER_BINLOG_FILE_NAME_TOO_LONG; SQLSTATE: HY000

  Message: Log filename too large: %s%s (%zu). Please fix this by archiving old logs and updating the index files.

  ER_BINLOG_FILE_NAME_TOO_LONG was added in 8.0.2.

- Error number: MY-010815; Symbol: ER_BINLOG_FILE_EXTENSION_NUMBER_RUNNING_LOW; SQLSTATE: HY000

  Message: Next log extension: %lu. Remaining log filename extensions: %lu. Please consider archiving some logs.

  ER_BINLOG_FILE_EXTENSION_NUMBER_RUNNING_LOW was added in 8.0.2.

- Error number: MY-010816; Symbol: ER_BINLOG_CANT_OPEN_FOR_LOGGING; SQLSTATE: HY000

  Message: Could not open %s for logging (error %d). Turning logging off for the whole duration of the MySQL server process. To turn it on again: fix the cause, shutdown the MySQL server and restart it.

  ER_BINLOG_CANT_OPEN_FOR_LOGGING was added in 8.0.2.

- Error number: MY-010817; Symbol: ER_BINLOG_FAILED_TO_SYNC_INDEX_FILE; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::open_index_file failed to sync the index file.

  ER_BINLOG_FAILED_TO_SYNC_INDEX_FILE was added in 8.0.2.

- Error number: MY-010818; Symbol: ER_BINLOG_ERROR_READING_GTIDS_FROM_RELAY_LOG; SQLSTATE: HY000

  Message: Error reading GTIDs from relaylog: %d

  ER_BINLOG_ERROR_READING_GTIDS_FROM_RELAY_LOG was added in 8.0.2.

- Error number: MY-010819; Symbol: ER_BINLOG_EVENTS_READ_FROM_RELAY_LOG_INFO; SQLSTATE: HY000

  Message: %lu events read in relaylog file '%s' for updating Retrieved_Gtid_Set and/or IO thread transaction parser state.

  ER_BINLOG_EVENTS_READ_FROM_RELAY_LOG_INFO was added in 8.0.2.

- Error number: MY-010820; Symbol: ER_BINLOG_ERROR_READING_GTIDS_FROM_BINARY_LOG; SQLSTATE: HY000

  Message: Error reading GTIDs from binary log: %d

  ER_BINLOG_ERROR_READING_GTIDS_FROM_BINARY_LOG was added in 8.0.2.

- Error number: MY-010821; Symbol: ER_BINLOG_EVENTS_READ_FROM_BINLOG_INFO; SQLSTATE: HY000

  Message: Read %lu events from binary log file '%s' to determine the GTIDs purged from binary logs.

  ER_BINLOG_EVENTS_READ_FROM_BINLOG_INFO was added in 8.0.2.

- Error number: MY-010822; Symbol: ER_BINLOG_CANT_GENERATE_NEW_FILE_NAME; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::open failed to generate new file name.

  ER_BINLOG_CANT_GENERATE_NEW_FILE_NAME was added in 8.0.2.

- Error number: MY-010823; Symbol: ER_BINLOG_FAILED_TO_SYNC_INDEX_FILE_IN_OPEN; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::open failed to sync the index file.

  ER_BINLOG_FAILED_TO_SYNC_INDEX_FILE_IN_OPEN was added in 8.0.2.

- Error number: MY-010824; Symbol: ER_BINLOG_CANT_USE_FOR_LOGGING; SQLSTATE: HY000

  Message: Could not use %s for logging (error %d). Turning logging off for the whole duration of the MySQL server process. To turn it on again: fix the cause, shutdown the MySQL server and restart it.

  ER_BINLOG_CANT_USE_FOR_LOGGING was added in 8.0.2.

- Error number: MY-010825; Symbol: ER_BINLOG_FAILED_TO_CLOSE_INDEX_FILE_WHILE_REBUILDING; SQLSTATE: HY000

  Message: While rebuilding index file %s: Failed to close the index file.

  ER_BINLOG_FAILED_TO_CLOSE_INDEX_FILE_WHILE_REBUILDING was added in 8.0.2.

- Error number: MY-010826; Symbol: ER_BINLOG_FAILED_TO_DELETE_INDEX_FILE_WHILE_REBUILDING; SQLSTATE: HY000

  Message: While rebuilding index file %s: Failed to delete the existing index file. It could be that file is being used by some other process.

  ER_BINLOG_FAILED_TO_DELETE_INDEX_FILE_WHILE_REBUILDING was added in 8.0.2.

- Error number: MY-010827; Symbol: ER_BINLOG_FAILED_TO_RENAME_INDEX_FILE_WHILE_REBUILDING; SQLSTATE: HY000

  Message: While rebuilding index file %s: Failed to rename the new index file to the existing index file.

  ER_BINLOG_FAILED_TO_RENAME_INDEX_FILE_WHILE_REBUILDING was added in 8.0.2.

- Error number: MY-010828; Symbol: ER_BINLOG_FAILED_TO_OPEN_INDEX_FILE_AFTER_REBUILDING; SQLSTATE: HY000

  Message: After rebuilding the index file %s: Failed to open the index file.

  ER_BINLOG_FAILED_TO_OPEN_INDEX_FILE_AFTER_REBUILDING was added in 8.0.2.

- Error number: MY-010829; Symbol: ER_BINLOG_CANT_APPEND_LOG_TO_TMP_INDEX; SQLSTATE: HY000

Message: MYSQL_BIN_LOG::add_log_to_index failed to append log file name: %s, to crash safe index file.

ER_BINLOG_CANT_APPEND_LOG_TO_TMP_INDEX was added in 8.0.2.

- Error number: MY-010830; Symbol: ER_BINLOG_CANT_LOCATE_OLD_BINLOG_OR_RELAY_LOG_FILES; SQLSTATE: HY000

  Message: Failed to locate old binlog or relay log files

  ER_BINLOG_CANT_LOCATE_OLD_BINLOG_OR_RELAY_LOG_FILES was added in 8.0.2.

- Error number: MY-010831; Symbol: ER_BINLOG_CANT_DELETE_FILE; SQLSTATE: HY000

  Message: Failed to delete file '%s'

  ER_BINLOG_CANT_DELETE_FILE was added in 8.0.2.

- Error number: MY-010832; Symbol: ER_BINLOG_CANT_SET_TMP_INDEX_NAME; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::set_crash_safe_index_file_name failed to set file name.

  ER_BINLOG_CANT_SET_TMP_INDEX_NAME was added in 8.0.2.

- Error number: MY-010833; Symbol: ER_BINLOG_FAILED_TO_OPEN_TEMPORARY_INDEX_FILE; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::open_crash_safe_index_file failed to open temporary index file.

  ER_BINLOG_FAILED_TO_OPEN_TEMPORARY_INDEX_FILE was added in 8.0.2.

- Error number: MY-010834; Symbol: ER_BINLOG_ERROR_GETTING_NEXT_LOG_FROM_INDEX; SQLSTATE: HY000

  Message: next log error: %d offset: %s log: %s included: %d

  ER_BINLOG_ERROR_GETTING_NEXT_LOG_FROM_INDEX was added in 8.0.2, removed after 8.0.12.

- Error number: MY-010835; Symbol: ER_BINLOG_CANT_OPEN_TMP_INDEX; SQLSTATE: HY000

  Message: %s failed to open the crash safe index file.

  ER_BINLOG_CANT_OPEN_TMP_INDEX was added in 8.0.2.

- Error number: MY-010836; Symbol: ER_BINLOG_CANT_COPY_INDEX_TO_TMP; SQLSTATE: HY000

  Message: %s failed to copy index file to crash safe index file.

  ER_BINLOG_CANT_COPY_INDEX_TO_TMP was added in 8.0.2.

- Error number: MY-010837; Symbol: ER_BINLOG_CANT_CLOSE_TMP_INDEX; SQLSTATE: HY000

  Message: %s failed to close the crash safe index file.

  ER_BINLOG_CANT_CLOSE_TMP_INDEX was added in 8.0.2.

- Error number: MY-010838; Symbol: ER_BINLOG_CANT_MOVE_TMP_TO_INDEX; SQLSTATE: HY000

  Message: %s failed to move crash safe index file to index file.

ER_BINLOG_CANT_MOVE_TMP_TO_INDEX was added in 8.0.2.

- Error number: MY-010839; Symbol: ER_BINLOG_PURGE_LOGS_CALLED_WITH_FILE_NOT_IN_INDEX; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::purge_logs was called with file %s not listed in the index.

  ER_BINLOG_PURGE_LOGS_CALLED_WITH_FILE_NOT_IN_INDEX was added in 8.0.2.

- Error number: MY-010840; Symbol: ER_BINLOG_PURGE_LOGS_CANT_SYNC_INDEX_FILE; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::purge_logs failed to sync the index file.

  ER_BINLOG_PURGE_LOGS_CANT_SYNC_INDEX_FILE was added in 8.0.2.

- Error number: MY-010841; Symbol: ER_BINLOG_PURGE_LOGS_CANT_COPY_TO_REGISTER_FILE; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::purge_logs failed to copy %s to register file.

  ER_BINLOG_PURGE_LOGS_CANT_COPY_TO_REGISTER_FILE was added in 8.0.2.

- Error number: MY-010842; Symbol: ER_BINLOG_PURGE_LOGS_CANT_FLUSH_REGISTER_FILE; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::purge_logs failed to flush register file.

  ER_BINLOG_PURGE_LOGS_CANT_FLUSH_REGISTER_FILE was added in 8.0.2.

- Error number: MY-010843; Symbol: ER_BINLOG_PURGE_LOGS_CANT_UPDATE_INDEX_FILE; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::purge_logs failed to update the index file

  ER_BINLOG_PURGE_LOGS_CANT_UPDATE_INDEX_FILE was added in 8.0.2.

- Error number: MY-010844; Symbol: ER_BINLOG_PURGE_LOGS_FAILED_TO_PURGE_LOG; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::purge_logs failed to process registered files that would be purged.

  ER_BINLOG_PURGE_LOGS_FAILED_TO_PURGE_LOG was added in 8.0.2.

- Error number: MY-010845; Symbol: ER_BINLOG_FAILED_TO_SET_PURGE_INDEX_FILE_NAME; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::set_purge_index_file_name failed to set file name.

  ER_BINLOG_FAILED_TO_SET_PURGE_INDEX_FILE_NAME was added in 8.0.2.

- Error number: MY-010846; Symbol: ER_BINLOG_FAILED_TO_OPEN_REGISTER_FILE; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::open_purge_index_file failed to open register file.

  ER_BINLOG_FAILED_TO_OPEN_REGISTER_FILE was added in 8.0.2.

- Error number: MY-010847; Symbol: ER_BINLOG_FAILED_TO_REINIT_REGISTER_FILE; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::purge_index_entry failed to reinit register file for read

  ER_BINLOG_FAILED_TO_REINIT_REGISTER_FILE was added in 8.0.2.

- Error number: MY-010848; Symbol: ER_BINLOG_FAILED_TO_READ_REGISTER_FILE; SQLSTATE: HY000

  Message: MYSQL_BIN_LOG::purge_index_entry error %d reading from register file.

  ER_BINLOG_FAILED_TO_READ_REGISTER_FILE was added in 8.0.2.

- Error number: MY-010849; Symbol: ER_CANT_STAT_FILE; SQLSTATE: HY000

  Message: Failed to execute mysql_file_stat on file '%s'

  ER_CANT_STAT_FILE was added in 8.0.2.

- Error number: MY-010850; Symbol: ER_BINLOG_CANT_DELETE_LOG_FILE_DOES_INDEX_MATCH_FILES; SQLSTATE: HY000

  Message: Failed to delete log file '%s'; consider examining correspondence of your binlog index file to the actual binlog files

  ER_BINLOG_CANT_DELETE_LOG_FILE_DOES_INDEX_MATCH_FILES was added in 8.0.2.

- Error number: MY-010851; Symbol: ER_BINLOG_CANT_DELETE_FILE_AND_READ_BINLOG_INDEX; SQLSTATE: HY000

  Message: Failed to delete file '%s' and read the binlog index file

  ER_BINLOG_CANT_DELETE_FILE_AND_READ_BINLOG_INDEX was added in 8.0.2.

- Error number: MY-010852; Symbol: ER_BINLOG_FAILED_TO_DELETE_LOG_FILE; SQLSTATE: HY000

  Message: Failed to delete log file '%s'

  ER_BINLOG_FAILED_TO_DELETE_LOG_FILE was added in 8.0.2.

- Error number: MY-010853; Symbol: ER_BINLOG_LOGGING_INCIDENT_TO_STOP_SLAVES; SQLSTATE: HY000

  Message: %s An incident event has been written to the binary log which will stop the slaves.

  ER_BINLOG_LOGGING_INCIDENT_TO_STOP_SLAVES was added in 8.0.2.

- Error number: MY-010854; Symbol: ER_BINLOG_CANT_FIND_LOG_IN_INDEX; SQLSTATE: HY000

  Message: find_log_pos() failed (error: %d)

  ER_BINLOG_CANT_FIND_LOG_IN_INDEX was added in 8.0.2.

- Error number: MY-010855; Symbol: ER_BINLOG_RECOVERING_AFTER_CRASH_USING; SQLSTATE: HY000

  Message: Recovering after a crash using %s

ER_BINLOG_RECOVERING_AFTER_CRASH_USING was added in 8.0.2.

- Error number: MY-010856; Symbol: ER_BINLOG_CANT_OPEN_CRASHED_BINLOG; SQLSTATE: HY000

  Message: Failed to open the crashed binlog file when master server is recovering it.

  ER_BINLOG_CANT_OPEN_CRASHED_BINLOG was added in 8.0.2.

- Error number: MY-010857; Symbol: ER_BINLOG_CANT_TRIM_CRASHED_BINLOG; SQLSTATE: HY000

  Message: Failed to trim the crashed binlog file when master server is recovering it.

  ER_BINLOG_CANT_TRIM_CRASHED_BINLOG was added in 8.0.2.

- Error number: MY-010858; Symbol: ER_BINLOG_CRASHED_BINLOG_TRIMMED; SQLSTATE: HY000

  Message: Crashed binlog file %s size is %llu, but recovered up to %llu. Binlog trimmed to %llu bytes.

  ER_BINLOG_CRASHED_BINLOG_TRIMMED was added in 8.0.2.

- Error number: MY-010859; Symbol: ER_BINLOG_CANT_CLEAR_IN_USE_FLAG_FOR_CRASHED_BINLOG; SQLSTATE: HY000

  Message: Failed to clear LOG_EVENT_BINLOG_IN_USE_F for the crashed binlog file when master server is recovering it.

  ER_BINLOG_CANT_CLEAR_IN_USE_FLAG_FOR_CRASHED_BINLOG was added in 8.0.2.

- Error number: MY-010860; Symbol: ER_BINLOG_FAILED_TO_RUN_AFTER_SYNC_HOOK; SQLSTATE: HY000

  Message: Failed to run 'after_sync' hooks

  ER_BINLOG_FAILED_TO_RUN_AFTER_SYNC_HOOK was added in 8.0.2.

- Error number: MY-010861; Symbol: ER_TURNING_LOGGING_OFF_FOR_THE_DURATION; SQLSTATE: HY000

  Message: %s Hence turning logging off for the whole duration of the MySQL server process. To turn it on again: fix the cause, shutdown the MySQL server and restart it.

  ER_TURNING_LOGGING_OFF_FOR_THE_DURATION was added in 8.0.2.

- Error number: MY-010862; Symbol: ER_BINLOG_FAILED_TO_RUN_AFTER_FLUSH_HOOK; SQLSTATE: HY000

  Message: Failed to run 'after_flush' hooks

  ER_BINLOG_FAILED_TO_RUN_AFTER_FLUSH_HOOK was added in 8.0.2.

- Error number: MY-010863; Symbol: ER_BINLOG_CRASH_RECOVERY_FAILED; SQLSTATE: HY000

  Message: Crash recovery failed. Either correct the problem (if it's, for example, out of memory error) and restart, or delete (or rename) binary log and start mysqld with --tc-heuristic-recover={commit|rollback}

  ER_BINLOG_CRASH_RECOVERY_FAILED was added in 8.0.2.

- Error number: MY-010864; Symbol: ER_BINLOG_WARNING_SUPPRESSED; SQLSTATE: HY000

  Message: The following warning was suppressed %d times during the last %d seconds in the error log

  ER_BINLOG_WARNING_SUPPRESSED was added in 8.0.2.

- Error number: MY-010865; Symbol: ER_NDB_LOG_ENTRY; SQLSTATE: HY000

  Message: NDB: %s

  ER_NDB_LOG_ENTRY was added in 8.0.2.

- Error number: MY-010866; Symbol: ER_NDB_LOG_ENTRY_WITH_PREFIX; SQLSTATE: HY000

  Message: NDB %s: %s

  ER_NDB_LOG_ENTRY_WITH_PREFIX was added in 8.0.2.

- Error number: MY-010867; Symbol: ER_NDB_BINLOG_CANT_CREATE_PURGE_THD; SQLSTATE: HY000

  Message: NDB: Unable to purge %s.%s File=%s (failed to setup thd)

  ER_NDB_BINLOG_CANT_CREATE_PURGE_THD was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010868; Symbol: ER_INNODB_UNKNOWN_COLLATION; SQLSTATE: HY000

  Message: Unknown collation #%lu.

  ER_INNODB_UNKNOWN_COLLATION was added in 8.0.2.

- Error number: MY-010869; Symbol: ER_INNODB_INVALID_LOG_GROUP_HOME_DIR; SQLSTATE: HY000

  Message: syntax error in innodb_log_group_home_dir

  ER_INNODB_INVALID_LOG_GROUP_HOME_DIR was added in 8.0.2.

- Error number: MY-010870; Symbol: ER_INNODB_INVALID_INNODB_UNDO_DIRECTORY; SQLSTATE: HY000

  Message: syntax error in innodb_undo_directory

  ER_INNODB_INVALID_INNODB_UNDO_DIRECTORY was added in 8.0.2.

- Error number: MY-010871; Symbol: ER_INNODB_ILLEGAL_COLON_IN_POOL; SQLSTATE: HY000

  Message: InnoDB: innodb_buffer_pool_filename cannot have colon (:) in the file name.

  ER_INNODB_ILLEGAL_COLON_IN_POOL was added in 8.0.2.

- Error number: MY-010872; Symbol: ER_INNODB_INVALID_PAGE_SIZE; SQLSTATE: HY000

  Message: InnoDB: Invalid page size=%lu.

  ER_INNODB_INVALID_PAGE_SIZE was added in 8.0.2.

- Error number: MY-010873; Symbol: ER_INNODB_DIRTY_WATER_MARK_NOT_LOW; SQLSTATE: HY000

Message: InnoDB: innodb_max_dirty_pages_pct_lwm cannot be set higher than innodb_max_dirty_pages_pct. Setting innodb_max_dirty_pages_pct_lwm to %lf

ER_INNODB_DIRTY_WATER_MARK_NOT_LOW was added in 8.0.2.

- Error number: MY-010874; Symbol: ER_INNODB_IO_CAPACITY_EXCEEDS_MAX; SQLSTATE: HY000

  Message: InnoDB: innodb_io_capacity cannot be set higher than innodb_io_capacity_max. Setting innodb_io_capacity to %lu

  ER_INNODB_IO_CAPACITY_EXCEEDS_MAX was added in 8.0.2.

- Error number: MY-010875; Symbol: ER_INNODB_FILES_SAME; SQLSTATE: HY000

  Message: %s and %s file names seem to be the same.

  ER_INNODB_FILES_SAME was added in 8.0.2.

- Error number: MY-010876; Symbol: ER_INNODB_UNREGISTERED_TRX_ACTIVE; SQLSTATE: HY000

  Message: Transaction not registered for MySQL 2PC, but transaction is active

  ER_INNODB_UNREGISTERED_TRX_ACTIVE was added in 8.0.2.

- Error number: MY-010877; Symbol: ER_INNODB_CLOSING_CONNECTION_ROLLS_BACK; SQLSTATE: HY000

  Message: MySQL is closing a connection that has an active InnoDB transaction. %llu row modifications will roll back.

  ER_INNODB_CLOSING_CONNECTION_ROLLS_BACK was added in 8.0.2.

- Error number: MY-010878; Symbol: ER_INNODB_TRX_XLATION_TABLE_OOM; SQLSTATE: HY000

  Message: InnoDB: fail to allocate memory for index translation table. Number of Index:%lu, array size: %lu

  ER_INNODB_TRX_XLATION_TABLE_OOM was added in 8.0.2.

- Error number: MY-010879; Symbol: ER_INNODB_CANT_FIND_INDEX_IN_INNODB_DD; SQLSTATE: HY000

  Message: Cannot find index %s in InnoDB index dictionary.

  ER_INNODB_CANT_FIND_INDEX_IN_INNODB_DD was added in 8.0.2.

- Error number: MY-010880; Symbol: ER_INNODB_INDEX_COLUMN_INFO_UNLIKE_MYSQLS; SQLSTATE: HY000

  Message: Found index %s whose column info does not match that of MySQL.

  ER_INNODB_INDEX_COLUMN_INFO_UNLIKE_MYSQLS was added in 8.0.2.

- Error number: MY-010881; Symbol: ER_INNODB_CANT_OPEN_TABLE; SQLSTATE: HY000

  Message: Failed to open table %s.

  ER_INNODB_CANT_OPEN_TABLE was added in 8.0.2.

- Error number: MY-010882; Symbol: ER_INNODB_CANT_BUILD_INDEX_XLATION_TABLE_FOR; SQLSTATE: HY000

  Message: Build InnoDB index translation table for Table %s failed

  ER_INNODB_CANT_BUILD_INDEX_XLATION_TABLE_FOR was added in 8.0.2.

- Error number: MY-010883; Symbol: ER_INNODB_PK_NOT_IN_MYSQL; SQLSTATE: HY000

  Message: Table %s has a primary key in InnoDB data dictionary, but not in MySQL!

  ER_INNODB_PK_NOT_IN_MYSQL was added in 8.0.2.

- Error number: MY-010884; Symbol: ER_INNODB_PK_ONLY_IN_MYSQL; SQLSTATE: HY000

  Message: Table %s has no primary key in InnoDB data dictionary, but has one in MySQL! If you created the table with a MySQL version < 3.23.54 and did not define a primary key, but defined a unique key with all non-NULL columns, then MySQL internally treats that key as the primary key. You can fix this error by dump + DROP + CREATE + reimport of the table.

  ER_INNODB_PK_ONLY_IN_MYSQL was added in 8.0.2.

- Error number: MY-010885; Symbol: ER_INNODB_CLUSTERED_INDEX_PRIVATE; SQLSTATE: HY000

  Message: Table %s key_used_on_scan is %lu even though there is no primary key inside InnoDB.

  ER_INNODB_CLUSTERED_INDEX_PRIVATE was added in 8.0.2.

- Error number: MY-010886; Symbol: ER_INNODB_PARTITION_TABLE_LOWERCASED; SQLSTATE: HY000

  Message: Partition table %s opened after converting to lower case. The table may have been moved from a case-insensitive file system. Please recreate the table in the current file system.

  ER_INNODB_PARTITION_TABLE_LOWERCASED was added in 8.0.2.

- Error number: MY-010887; Symbol: ER_ERRMSG_REPLACEMENT_DODGY; SQLSTATE: HY000

  Message: Cannot replace error message (%s,%s,%s) "%s" with "%s"; wrong number or type of %% subsitutions.

  ER_ERRMSG_REPLACEMENT_DODGY was added in 8.0.2.

- Error number: MY-010888; Symbol: ER_ERRMSG_REPLACEMENTS_FAILED; SQLSTATE: HY000

  Message: Table for error message replacements could not be found or read, or one or more replacements could not be applied.

  ER_ERRMSG_REPLACEMENTS_FAILED was added in 8.0.2.

- Error number: MY-010889; Symbol: ER_NPIPE_CANT_CREATE; SQLSTATE: HY000

  Message: %s: %s

  ER_NPIPE_CANT_CREATE was added in 8.0.2.

- Error number: MY-010890; Symbol: ER_PARTITION_MOVE_CREATED_DUPLICATE_ROW_PLEASE_FIX; SQLSTATE: HY000

Message: Table '%s': Delete from part %d failed with error %d. But it was already inserted into part %d, when moving the misplaced row! Please manually fix the duplicate row: %s

ER_PARTITION_MOVE_CREATED_DUPLICATE_ROW_PLEASE_FIX was added in 8.0.2.

- Error number: MY-010891; Symbol: ER_AUDIT_CANT_ABORT_COMMAND; SQLSTATE: HY000

  Message: Command '%s' cannot be aborted. The trigger error was (%d) [%s]: %s

  ER_AUDIT_CANT_ABORT_COMMAND was added in 8.0.2.

- Error number: MY-010892; Symbol: ER_AUDIT_CANT_ABORT_EVENT; SQLSTATE: HY000

  Message: Event '%s' cannot be aborted. The trigger error was (%d) [%s]: %s

  ER_AUDIT_CANT_ABORT_EVENT was added in 8.0.2.

- Error number: MY-010893; Symbol: ER_AUDIT_WARNING; SQLSTATE: HY000

  Message: %s. The trigger error was (%d) [%s]: %s

  ER_AUDIT_WARNING was added in 8.0.2.

- Error number: MY-010894; Symbol: ER_NDB_NUMBER_OF_CHANNELS; SQLSTATE: HY000

  Message: Slave SQL: Configuration with number of replication masters = %u' is not supported when applying to Ndb

  ER_NDB_NUMBER_OF_CHANNELS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010895; Symbol: ER_NDB_SLAVE_PARALLEL_WORKERS; SQLSTATE: HY000

  Message: Slave SQL: Configuration 'slave_parallel_workers = %lu' is not supported when applying to Ndb

  ER_NDB_SLAVE_PARALLEL_WORKERS was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010896; Symbol: ER_NDB_DISTRIBUTING_ERR; SQLSTATE: HY000

  Message: NDB %s: distributing %s err: %u

  ER_NDB_DISTRIBUTING_ERR was added in 8.0.2, removed after 8.0.13.

- Error number: MY-010897; Symbol: ER_RPL_SLAVE_INSECURE_CHANGE_MASTER; SQLSTATE: HY000

  Message: Storing MySQL user name or password information in the master info repository is not secure and is therefore not recommended. Please consider using the USER and PASSWORD connection options for START SLAVE; see the 'START SLAVE Syntax' in the MySQL Manual for more information.

  ER_RPL_SLAVE_INSECURE_CHANGE_MASTER was added in 8.0.4.

- Error number: MY-010898; Symbol: ER_RPL_SLAVE_FLUSH_RELAY_LOGS_NOT_ALLOWED; SQLSTATE: HY000

  Message: FLUSH RELAY LOGS cannot be performed on channel '%s'.

  ER_RPL_SLAVE_FLUSH_RELAY_LOGS_NOT_ALLOWED was added in 8.0.4, removed after 8.0.15.

- Error number: MY-010899; Symbol: ER_RPL_SLAVE_INCORRECT_CHANNEL; SQLSTATE: HY000

  Message: Slave channel '%s' does not exist.

  ER_RPL_SLAVE_INCORRECT_CHANNEL was added in 8.0.4.

- Error number: MY-010900; Symbol: ER_FAILED_TO_FIND_DL_ENTRY; SQLSTATE: HY000

  Message: Can't find symbol '%s' in library.

  ER_FAILED_TO_FIND_DL_ENTRY was added in 8.0.4.

- Error number: MY-010901; Symbol: ER_FAILED_TO_OPEN_SHARED_LIBRARY; SQLSTATE: HY000

  Message: Can't open shared library '%s' (errno: %d %s).

  ER_FAILED_TO_OPEN_SHARED_LIBRARY was added in 8.0.4.

- Error number: MY-010902; Symbol: ER_THREAD_PRIORITY_IGNORED; SQLSTATE: HY000

  Message: Thread priority attribute setting in Resource Group SQL shall be ignored due to unsupported platform or insufficient privilege.

  ER_THREAD_PRIORITY_IGNORED was added in 8.0.4.

- Error number: MY-010903; Symbol: ER_BINLOG_CACHE_SIZE_TOO_LARGE; SQLSTATE: HY000

  Message: Option binlog_cache_size (%lu) is greater than max_binlog_cache_size (%lu); setting binlog_cache_size equal to max_binlog_cache_size.

  ER_BINLOG_CACHE_SIZE_TOO_LARGE was added in 8.0.4.

- Error number: MY-010904; Symbol: ER_BINLOG_STMT_CACHE_SIZE_TOO_LARGE; SQLSTATE: HY000

  Message: Option binlog_stmt_cache_size (%lu) is greater than max_binlog_stmt_cache_size (%lu); setting binlog_stmt_cache_size equal to max_binlog_stmt_cache_size.

  ER_BINLOG_STMT_CACHE_SIZE_TOO_LARGE was added in 8.0.4.

- Error number: MY-010905; Symbol: ER_FAILED_TO_GENERATE_UNIQUE_LOGFILE; SQLSTATE: HY000

  Message: Can't generate a unique log-filename %s.(1-999).

  ER_FAILED_TO_GENERATE_UNIQUE_LOGFILE was added in 8.0.4.

- Error number: MY-010906; Symbol: ER_FAILED_TO_READ_FILE; SQLSTATE: HY000

  Message: Error reading file '%s' (errno: %d - %s)

  ER_FAILED_TO_READ_FILE was added in 8.0.4.

- Error number: MY-010907; Symbol: ER_FAILED_TO_WRITE_TO_FILE; SQLSTATE: HY000

  Message: Error writing file '%s' (errno: %d - %s)

  ER_FAILED_TO_WRITE_TO_FILE was added in 8.0.4.

- Error number: MY-010908; Symbol: ER_BINLOG_UNSAFE_MESSAGE_AND_STATEMENT; SQLSTATE: HY000

  Message: %s Statement: %s

  ER_BINLOG_UNSAFE_MESSAGE_AND_STATEMENT was added in 8.0.4.

- Error number: MY-010909; Symbol: ER_FORCE_CLOSE_THREAD; SQLSTATE: HY000

  Message: %s: Forcing close of thread %ld user: '%s'.

  ER_FORCE_CLOSE_THREAD was added in 8.0.4.

- Error number: MY-010910; Symbol: ER_SERVER_SHUTDOWN_COMPLETE; SQLSTATE: HY000

  Message: %s: Shutdown complete (mysqld %s) %s.

  ER_SERVER_SHUTDOWN_COMPLETE was added in 8.0.4.

- Error number: MY-010911; Symbol: ER_RPL_CANT_HAVE_SAME_BASENAME; SQLSTATE: HY000

  Message: Cannot have same base name '%s' for both binary and relay logs. Please check %s (default '%s' if --log-bin option is not used, default '%s' if --log-bin option is used without argument) and %s (default '%s') options to ensure they do not conflict.

  ER_RPL_CANT_HAVE_SAME_BASENAME was added in 8.0.3.

- Error number: MY-010912; Symbol: ER_RPL_GTID_MODE_REQUIRES_ENFORCE_GTID_CONSISTENCY_ON; SQLSTATE: HY000

  Message: GTID_MODE = ON requires ENFORCE_GTID_CONSISTENCY = ON.

  ER_RPL_GTID_MODE_REQUIRES_ENFORCE_GTID_CONSISTENCY_ON was added in 8.0.4.

- Error number: MY-010913; Symbol: ER_WARN_NO_SERVERID_SPECIFIED; SQLSTATE: HY000

  Message: You have not provided a mandatory server-id. Servers in a replication topology must have unique server-ids. Please refer to the proper server start-up parameters documentation.

  ER_WARN_NO_SERVERID_SPECIFIED was added in 8.0.3.

- Error number: MY-010914; Symbol: ER_ABORTING_USER_CONNECTION; SQLSTATE: HY000

  Message: Aborted connection %u to db: '%s' user: '%s' host: '%s' (%s).

  ER_ABORTING_USER_CONNECTION was added in 8.0.4.

- Error number: MY-010915; Symbol: ER_SQL_MODE_MERGED_WITH_STRICT_MODE; SQLSTATE: HY000

  Message: 'NO_ZERO_DATE', 'NO_ZERO_IN_DATE' and 'ERROR_FOR_DIVISION_BY_ZERO' sql modes should be used with strict mode. They will be merged with strict mode in a future release.

  ER_SQL_MODE_MERGED_WITH_STRICT_MODE was added in 8.0.4.

- Error number: MY-010916; Symbol: ER_GTID_PURGED_WAS_UPDATED; SQLSTATE: HY000

  Message: @@GLOBAL.GTID_PURGED was changed from '%s' to '%s'.

ER_GTID_PURGED_WAS_UPDATED was added in 8.0.4.

- Error number: MY-010917; Symbol: ER_GTID_EXECUTED_WAS_UPDATED; SQLSTATE: HY000

  Message: @@GLOBAL.GTID_EXECUTED was changed from '%s' to '%s'.

  ER_GTID_EXECUTED_WAS_UPDATED was added in 8.0.4.

- Error number: MY-010918; Symbol: ER_DEPRECATE_MSG_WITH_REPLACEMENT; SQLSTATE: HY000

  Message: '%s' is deprecated and will be removed in a future release. Please use %s instead.

  ER_DEPRECATE_MSG_WITH_REPLACEMENT was added in 8.0.4.

- Error number: MY-010919; Symbol: ER_TRG_CREATION_CTX_NOT_SET; SQLSTATE: HY000

  Message: Triggers for table `%s`.`%s` have no creation context

  ER_TRG_CREATION_CTX_NOT_SET was added in 8.0.4.

- Error number: MY-010920; Symbol: ER_FILE_HAS_OLD_FORMAT; SQLSTATE: HY000

  Message: '%s' has an old format, you should re-create the '%s' object(s)

  ER_FILE_HAS_OLD_FORMAT was added in 8.0.4.

- Error number: MY-010921; Symbol: ER_VIEW_CREATION_CTX_NOT_SET; SQLSTATE: HY000

  Message: View `%s`.`%s` has no creation context

  ER_VIEW_CREATION_CTX_NOT_SET was added in 8.0.4.

- Error number: MY-010923; Symbol: ER_TABLE_UPGRADE_REQUIRED; SQLSTATE: HY000

  Message: Table upgrade required. Please do "REPAIR TABLE `%s`" or dump/reload to fix it!

  ER_TABLE_UPGRADE_REQUIRED was added in 8.0.4.

- Error number: MY-010924; Symbol: ER_GET_ERRNO_FROM_STORAGE_ENGINE; SQLSTATE: HY000

  Message: Got error %d - '%s' from storage engine.

  ER_GET_ERRNO_FROM_STORAGE_ENGINE was added in 8.0.4.

- Error number: MY-010925; Symbol: ER_ACCESS_DENIED_ERROR_WITHOUT_PASSWORD; SQLSTATE: HY000

  Message: Access denied for user '%s'@'%s'

  ER_ACCESS_DENIED_ERROR_WITHOUT_PASSWORD was added in 8.0.4.

- Error number: MY-010926; Symbol: ER_ACCESS_DENIED_ERROR_WITH_PASSWORD; SQLSTATE: HY000

  Message: Access denied for user '%s'@'%s' (using password: %s)

  ER_ACCESS_DENIED_ERROR_WITH_PASSWORD was added in 8.0.4.

- Error number: MY-010927; Symbol: ER_ACCESS_DENIED_FOR_USER_ACCOUNT_LOCKED; SQLSTATE: HY000

  Message: Access denied for user '%s'@'%s'. Account is locked.

  ER_ACCESS_DENIED_FOR_USER_ACCOUNT_LOCKED was added in 8.0.4.

- Error number: MY-010928; Symbol: ER_MUST_CHANGE_EXPIRED_PASSWORD; SQLSTATE: HY000

  Message: Your password has expired. To log in you must change it using a client that supports expired passwords.

  ER_MUST_CHANGE_EXPIRED_PASSWORD was added in 8.0.4.

- Error number: MY-010929; Symbol: ER_SYSTEM_TABLES_NOT_SUPPORTED_BY_STORAGE_ENGINE; SQLSTATE: HY000

  Message: Storage engine '%s' does not support system tables. [%s.%s].

  ER_SYSTEM_TABLES_NOT_SUPPORTED_BY_STORAGE_ENGINE was added in 8.0.4.

- Error number: MY-010930; Symbol: ER_FILESORT_TERMINATED; SQLSTATE: HY000

  Message: Sort aborted

  ER_FILESORT_TERMINATED was added in 8.0.4, removed after 8.0.18.

- Error number: MY-010931; Symbol: ER_SERVER_STARTUP_MSG; SQLSTATE: HY000

  Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d %s.

  ER_SERVER_STARTUP_MSG was added in 8.0.4.

- Error number: MY-010932; Symbol: ER_FAILED_TO_FIND_LOCALE_NAME; SQLSTATE: HY000

  Message: Unknown locale: '%s'.

  ER_FAILED_TO_FIND_LOCALE_NAME was added in 8.0.4.

- Error number: MY-010933; Symbol: ER_FAILED_TO_FIND_COLLATION_NAME; SQLSTATE: HY000

  Message: Unknown collation: '%s'.

  ER_FAILED_TO_FIND_COLLATION_NAME was added in 8.0.4.

- Error number: MY-010934; Symbol: ER_SERVER_OUT_OF_RESOURCES; SQLSTATE: HY000

  Message: Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space

  ER_SERVER_OUT_OF_RESOURCES was added in 8.0.4.

- Error number: MY-010935; Symbol: ER_SERVER_OUTOFMEMORY; SQLSTATE: HY000

  Message: Out of memory; restart server and try again (needed %d bytes)

  ER_SERVER_OUTOFMEMORY was added in 8.0.4.

- Error number: MY-010936; Symbol: ER_INVALID_COLLATION_FOR_CHARSET; SQLSTATE: HY000

Message: COLLATION '%s' is not valid for CHARACTER SET '%s'

ER_INVALID_COLLATION_FOR_CHARSET was added in 8.0.4.

- Error number: MY-010937; Symbol: ER_CANT_START_ERROR_LOG_SERVICE; SQLSTATE: HY000

  Message: Failed to set %s at or around "%s" -- service is valid, but can not be initialized; please check its configuration and make sure it can read its input(s) and write to its output(s).

  ER_CANT_START_ERROR_LOG_SERVICE was added in 8.0.4.

- Error number: MY-010938; Symbol: ER_CREATING_NEW_UUID_FIRST_START; SQLSTATE: HY000

  Message: Generating a new UUID: %s.

  ER_CREATING_NEW_UUID_FIRST_START was added in 8.0.4.

- Error number: MY-010939; Symbol: ER_FAILED_TO_GET_ABSOLUTE_PATH; SQLSTATE: HY000

  Message: Failed to get absolute path of program executable %s

  ER_FAILED_TO_GET_ABSOLUTE_PATH was added in 8.0.4.

- Error number: MY-010940; Symbol: ER_PERFSCHEMA_COMPONENTS_INFRASTRUCTURE_BOOTSTRAP; SQLSTATE: HY000

  Message: Failed to bootstrap performance schema components infrastructure.

  ER_PERFSCHEMA_COMPONENTS_INFRASTRUCTURE_BOOTSTRAP was added in 8.0.4.

- Error number: MY-010941; Symbol: ER_PERFSCHEMA_COMPONENTS_INFRASTRUCTURE_SHUTDOWN; SQLSTATE: HY000

  Message: Failed to deinit performance schema components infrastructure.

  ER_PERFSCHEMA_COMPONENTS_INFRASTRUCTURE_SHUTDOWN was added in 8.0.4.

- Error number: MY-010942; Symbol: ER_DUP_FD_OPEN_FAILED; SQLSTATE: HY000

  Message: Could not open duplicate fd for %s: %s.

  ER_DUP_FD_OPEN_FAILED was added in 8.0.4.

- Error number: MY-010943; Symbol: ER_SYSTEM_VIEW_INIT_FAILED; SQLSTATE: HY000

  Message: System views initialization failed.

  ER_SYSTEM_VIEW_INIT_FAILED was added in 8.0.4.

- Error number: MY-010944; Symbol: ER_RESOURCE_GROUP_POST_INIT_FAILED; SQLSTATE: HY000

  Message: Resource group post initialization failed.

  ER_RESOURCE_GROUP_POST_INIT_FAILED was added in 8.0.4.

- Error number: MY-010945; Symbol: ER_RESOURCE_GROUP_SUBSYSTEM_INIT_FAILED; SQLSTATE: HY000

Message: Resource Group subsystem initialization failed.

ER_RESOURCE_GROUP_SUBSYSTEM_INIT_FAILED was added in 8.0.4.

- Error number: MY-010946; Symbol: ER_FAILED_START_MYSQLD_DAEMON; SQLSTATE: HY000

  Message: Failed to start mysqld daemon. Check mysqld error log.

  ER_FAILED_START_MYSQLD_DAEMON was added in 8.0.4.

- Error number: MY-010947; Symbol: ER_CANNOT_CHANGE_TO_ROOT_DIR; SQLSTATE: HY000

  Message: Cannot change to root directory: %s.

  ER_CANNOT_CHANGE_TO_ROOT_DIR was added in 8.0.4.

- Error number: MY-010948; Symbol: ER_PERSISTENT_PRIVILEGES_BOOTSTRAP; SQLSTATE: HY000

  Message: Failed to bootstrap persistent privileges.

  ER_PERSISTENT_PRIVILEGES_BOOTSTRAP was added in 8.0.4.

- Error number: MY-010949; Symbol: ER_BASEDIR_SET_TO; SQLSTATE: HY000

  Message: Basedir set to %s.

  ER_BASEDIR_SET_TO was added in 8.0.4.

- Error number: MY-010950; Symbol: ER_RPL_FILTER_ADD_WILD_DO_TABLE_FAILED; SQLSTATE: HY000

  Message: Could not add wild do table rule '%s'!

  ER_RPL_FILTER_ADD_WILD_DO_TABLE_FAILED was added in 8.0.4.

- Error number: MY-010951; Symbol: ER_RPL_FILTER_ADD_WILD_IGNORE_TABLE_FAILED; SQLSTATE: HY000

  Message: Could not add wild ignore table rule '%s'!

  ER_RPL_FILTER_ADD_WILD_IGNORE_TABLE_FAILED was added in 8.0.4.

- Error number: MY-010952; Symbol: ER_PRIVILEGE_SYSTEM_INIT_FAILED; SQLSTATE: HY000

  Message: The privilege system failed to initialize correctly. For complete instructions on how to upgrade MySQL to a new version please see the 'Upgrading MySQL' section from the MySQL manual.

  ER_PRIVILEGE_SYSTEM_INIT_FAILED was added in 8.0.4.

- Error number: MY-010953; Symbol: ER_CANNOT_SET_LOG_ERROR_SERVICES; SQLSTATE: HY000

  Message: Cannot set services "%s" requested in --log-error-services, using defaults.

  ER_CANNOT_SET_LOG_ERROR_SERVICES was added in 8.0.4.

- Error number: MY-010954; Symbol: ER_PERFSCHEMA_TABLES_INIT_FAILED; SQLSTATE: HY000

  Message: Performance schema tables initialization failed.

ER_PERFSCHEMA_TABLES_INIT_FAILED was added in 8.0.4.

- Error number: MY-010955; Symbol: ER_TX_EXTRACTION_ALGORITHM_FOR_BINLOG_TX_DEPEDENCY_TRACKING; SQLSTATE: HY000

  Message: The transaction_write_set_extraction must be set to %s when binlog_transaction_dependency_tracking is %s.

  ER_TX_EXTRACTION_ALGORITHM_FOR_BINLOG_TX_DEPEDENCY_TRACKING was added in 8.0.4.

- Error number: MY-010956; Symbol: ER_INVALID_REPLICATION_TIMESTAMPS; SQLSTATE: HY000

  Message: Invalid replication timestamps: original commit timestamp is more recent than the immediate commit timestamp. This may be an issue if delayed replication is active. Make sure that servers have their clocks set to the correct time. No further message will be emitted until after timestamps become valid again.

  ER_INVALID_REPLICATION_TIMESTAMPS was added in 8.0.4.

- Error number: MY-010957; Symbol: ER_RPL_TIMESTAMPS_RETURNED_TO_NORMAL; SQLSTATE: HY000

  Message: The replication timestamps have returned to normal values.

  ER_RPL_TIMESTAMPS_RETURNED_TO_NORMAL was added in 8.0.4.

- Error number: MY-010958; Symbol: ER_BINLOG_FILE_OPEN_FAILED; SQLSTATE: HY000

  Message: %s.

  ER_BINLOG_FILE_OPEN_FAILED was added in 8.0.4.

- Error number: MY-010959; Symbol: ER_BINLOG_EVENT_WRITE_TO_STMT_CACHE_FAILED; SQLSTATE: HY000

  Message: Failed to write an incident event into stmt_cache.

  ER_BINLOG_EVENT_WRITE_TO_STMT_CACHE_FAILED was added in 8.0.4.

- Error number: MY-010960; Symbol: ER_SLAVE_RELAY_LOG_TRUNCATE_INFO; SQLSTATE: HY000

  Message: Relaylog file %s size was %llu, but was truncated at %llu.

  ER_SLAVE_RELAY_LOG_TRUNCATE_INFO was added in 8.0.4.

- Error number: MY-010961; Symbol: ER_SLAVE_RELAY_LOG_PURGE_FAILED; SQLSTATE: HY000

  Message: Unable to purge relay log files. %s:%s.

  ER_SLAVE_RELAY_LOG_PURGE_FAILED was added in 8.0.4.

- Error number: MY-010962; Symbol: ER_RPL_SLAVE_FILTER_CREATE_FAILED; SQLSTATE: HY000

  Message: Slave: failed in creating filter for channel '%s'.

  ER_RPL_SLAVE_FILTER_CREATE_FAILED was added in 8.0.4.

- Error number: MY-010963; Symbol: ER_RPL_SLAVE_GLOBAL_FILTERS_COPY_FAILED; SQLSTATE: HY000

  Message: Slave: failed in copying the global filters to its own per-channel filters on configuration for channel '%s'.

  ER_RPL_SLAVE_GLOBAL_FILTERS_COPY_FAILED was added in 8.0.4.

- Error number: MY-010964; Symbol: ER_RPL_SLAVE_RESET_FILTER_OPTIONS; SQLSTATE: HY000

  Message: There are per-channel replication filter(s) configured for channel '%s' which does not exist. The filter(s) have been discarded.

  ER_RPL_SLAVE_RESET_FILTER_OPTIONS was added in 8.0.4.

- Error number: MY-010965; Symbol: ER_MISSING_GRANT_SYSTEM_TABLE; SQLSTATE: HY000

  Message: The system table mysql.global_grants is missing. Please perform the MySQL upgrade procedure.

  ER_MISSING_GRANT_SYSTEM_TABLE was added in 8.0.4.

- Error number: MY-010966; Symbol: ER_MISSING_ACL_SYSTEM_TABLE; SQLSTATE: HY000

  Message: ACL table mysql.%.*s missing. Some operations may fail.

  ER_MISSING_ACL_SYSTEM_TABLE was added in 8.0.4.

- Error number: MY-010967; Symbol: ER_ANONYMOUS_AUTH_ID_NOT_ALLOWED_IN_MANDATORY_ROLES; SQLSTATE: HY000

  Message: Can't set mandatory_role %s@%s: Anonymous authorization IDs are not allowed as roles.

  ER_ANONYMOUS_AUTH_ID_NOT_ALLOWED_IN_MANDATORY_ROLES was added in 8.0.4.

- Error number: MY-010968; Symbol: ER_UNKNOWN_AUTH_ID_IN_MANDATORY_ROLE; SQLSTATE: HY000

  Message: Can't set mandatory_role: There's no such authorization ID %s@%s.

  ER_UNKNOWN_AUTH_ID_IN_MANDATORY_ROLE was added in 8.0.4.

- Error number: MY-010969; Symbol: ER_WRITE_ROW_TO_PARTITION_FAILED; SQLSTATE: HY000

  Message: Table '%s' failed to move/insert a row from part %d into part %d: %s.

  ER_WRITE_ROW_TO_PARTITION_FAILED was added in 8.0.4.

- Error number: MY-010970; Symbol: ER_RESOURCE_GROUP_METADATA_UPDATE_SKIPPED; SQLSTATE: HY000

  Message: Skipped updating resource group metadata in InnoDB read only mode.

  ER_RESOURCE_GROUP_METADATA_UPDATE_SKIPPED was added in 8.0.4.

- Error number: MY-010971; Symbol: ER_FAILED_TO_PERSIST_RESOURCE_GROUP_METADATA; SQLSTATE: HY000

  Message: Failed to persist resource group %s to Data Dictionary.

ER_FAILED_TO_PERSIST_RESOURCE_GROUP_METADATA was added in 8.0.4.

- Error number: MY-010972; Symbol: ER_FAILED_TO_DESERIALIZE_RESOURCE_GROUP; SQLSTATE: HY000

  Message: Failed to deserialize resource group %s.

  ER_FAILED_TO_DESERIALIZE_RESOURCE_GROUP was added in 8.0.4.

- Error number: MY-010973; Symbol: ER_FAILED_TO_UPDATE_RESOURCE_GROUP; SQLSTATE: HY000

  Message: Update of resource group %s failed.

  ER_FAILED_TO_UPDATE_RESOURCE_GROUP was added in 8.0.4.

- Error number: MY-010974; Symbol: ER_RESOURCE_GROUP_VALIDATION_FAILED; SQLSTATE: HY000

  Message: Validation of resource group %s failed. Resource group is disabled.

  ER_RESOURCE_GROUP_VALIDATION_FAILED was added in 8.0.4.

- Error number: MY-010975; Symbol: ER_FAILED_TO_ALLOCATE_MEMORY_FOR_RESOURCE_GROUP; SQLSTATE: HY000

  Message: Unable to allocate memory for Resource Group %s.

  ER_FAILED_TO_ALLOCATE_MEMORY_FOR_RESOURCE_GROUP was added in 8.0.4.

- Error number: MY-010976; Symbol: ER_FAILED_TO_ALLOCATE_MEMORY_FOR_RESOURCE_GROUP_HASH; SQLSTATE: HY000

  Message: Failed to allocate memory for resource group hash.

  ER_FAILED_TO_ALLOCATE_MEMORY_FOR_RESOURCE_GROUP_HASH was added in 8.0.4.

- Error number: MY-010977; Symbol: ER_FAILED_TO_ADD_RESOURCE_GROUP_TO_MAP; SQLSTATE: HY000

  Message: Failed to add resource group %s to resource group map.

  ER_FAILED_TO_ADD_RESOURCE_GROUP_TO_MAP was added in 8.0.4.

- Error number: MY-010978; Symbol: ER_RESOURCE_GROUP_IS_DISABLED; SQLSTATE: HY000

  Message: Resource group feature is disabled. (Server is compiled with DISABLE_PSI_THREAD).

  ER_RESOURCE_GROUP_IS_DISABLED was added in 8.0.4.

- Error number: MY-010979; Symbol: ER_FAILED_TO_APPLY_RESOURCE_GROUP_CONTROLLER; SQLSTATE: HY000

  Message: Unable to apply resource group controller %s.

  ER_FAILED_TO_APPLY_RESOURCE_GROUP_CONTROLLER was added in 8.0.4.

- Error number: MY-010980; Symbol: ER_FAILED_TO_ACQUIRE_LOCK_ON_RESOURCE_GROUP; SQLSTATE: HY000

  Message: Unable to acquire lock on the resource group %s. Hint to switch resource group shall be ignored.

  ER_FAILED_TO_ACQUIRE_LOCK_ON_RESOURCE_GROUP was added in 8.0.4.

- Error number: MY-010981; Symbol: ER_PFS_NOTIFICATION_FUNCTION_REGISTER_FAILED; SQLSTATE: HY000

  Message: PFS %s notification function registration failed.

  ER_PFS_NOTIFICATION_FUNCTION_REGISTER_FAILED was added in 8.0.4.

- Error number: MY-010982; Symbol: ER_RES_GRP_SET_THR_AFFINITY_FAILED; SQLSTATE: HY000

  Message: Unable to bind thread id %llu to cpu id %u (error code %d - %s).

  ER_RES_GRP_SET_THR_AFFINITY_FAILED was added in 8.0.4.

- Error number: MY-010983; Symbol: ER_RES_GRP_SET_THR_AFFINITY_TO_CPUS_FAILED; SQLSTATE: HY000

  Message: Unable to bind thread id %llu to cpu ids (error code %d - %s).

  ER_RES_GRP_SET_THR_AFFINITY_TO_CPUS_FAILED was added in 8.0.4.

- Error number: MY-010984; Symbol: ER_RES_GRP_THD_UNBIND_FROM_CPU_FAILED; SQLSTATE: HY000

  Message: Unbind thread id %llu failed. (error code %d - %s).

  ER_RES_GRP_THD_UNBIND_FROM_CPU_FAILED was added in 8.0.4.

- Error number: MY-010985; Symbol: ER_RES_GRP_SET_THREAD_PRIORITY_FAILED; SQLSTATE: HY000

  Message: Setting thread priority %d to thread id %llu failed. (error code %d - %s).

  ER_RES_GRP_SET_THREAD_PRIORITY_FAILED was added in 8.0.4.

- Error number: MY-010986; Symbol: ER_RES_GRP_FAILED_TO_DETERMINE_NICE_CAPABILITY; SQLSTATE: HY000

  Message: Unable to determine CAP_SYS_NICE capability.

  ER_RES_GRP_FAILED_TO_DETERMINE_NICE_CAPABILITY was added in 8.0.4.

- Error number: MY-010987; Symbol: ER_RES_GRP_FAILED_TO_GET_THREAD_HANDLE; SQLSTATE: HY000

  Message: %s failed: Failed to get handle for thread %llu.

  ER_RES_GRP_FAILED_TO_GET_THREAD_HANDLE was added in 8.0.4.

- Error number: MY-010988; Symbol: ER_RES_GRP_GET_THREAD_PRIO_NOT_SUPPORTED; SQLSTATE: HY000

Message: Retrieval of thread priority unsupported on %s.

ER_RES_GRP_GET_THREAD_PRIO_NOT_SUPPORTED was added in 8.0.4.

- Error number: MY-010989; Symbol: ER_RES_GRP_FAILED_DETERMINE_CPU_COUNT; SQLSTATE: HY000

  Message: Unable to determine the number of CPUs.

  ER_RES_GRP_FAILED_DETERMINE_CPU_COUNT was added in 8.0.4.

- Error number: MY-010990; Symbol: ER_RES_GRP_FEATURE_NOT_AVAILABLE; SQLSTATE: HY000

  Message: Resource group feature shall not be available. Incompatible thread handling option.

  ER_RES_GRP_FEATURE_NOT_AVAILABLE was added in 8.0.4.

- Error number: MY-010991; Symbol: ER_RES_GRP_INVALID_THREAD_PRIORITY; SQLSTATE: HY000

  Message: Invalid thread priority %d for a %s resource group. Allowed range is [%d, %d].

  ER_RES_GRP_INVALID_THREAD_PRIORITY was added in 8.0.4.

- Error number: MY-010992; Symbol: ER_RES_GRP_SOLARIS_PROCESSOR_BIND_TO_CPUID_FAILED; SQLSTATE: HY000

  Message: bind_to_cpu failed: processor_bind for cpuid %u failed (error code %d - %s).

  ER_RES_GRP_SOLARIS_PROCESSOR_BIND_TO_CPUID_FAILED was added in 8.0.4.

- Error number: MY-010993; Symbol: ER_RES_GRP_SOLARIS_PROCESSOR_BIND_TO_THREAD_FAILED; SQLSTATE: HY000

  Message: bind_to_cpu failed: processor_bind for thread %%llx with cpu id %u (error code %d - %s).

  ER_RES_GRP_SOLARIS_PROCESSOR_BIND_TO_THREAD_FAILED was added in 8.0.4.

- Error number: MY-010994; Symbol: ER_RES_GRP_SOLARIS_PROCESSOR_AFFINITY_FAILED; SQLSTATE: HY000

  Message: %s failed: processor_affinity failed (error code %d - %s).

  ER_RES_GRP_SOLARIS_PROCESSOR_AFFINITY_FAILED was added in 8.0.4.

- Error number: MY-010995; Symbol: ER_DD_UPGRADE_RENAME_IDX_STATS_FILE_FAILED; SQLSTATE: HY000

  Message: Error in renaming mysql_index_stats.ibd.

  ER_DD_UPGRADE_RENAME_IDX_STATS_FILE_FAILED was added in 8.0.4.

- Error number: MY-010996; Symbol: ER_DD_UPGRADE_DD_OPEN_FAILED; SQLSTATE: HY000

  Message: Error in opening data directory %s.

  ER_DD_UPGRADE_DD_OPEN_FAILED was added in 8.0.4.

- Error number: MY-010997; Symbol: ER_DD_UPGRADE_FAILED_TO_FETCH_TABLESPACES; SQLSTATE: HY000

  Message: Error in fetching list of tablespaces.

  ER_DD_UPGRADE_FAILED_TO_FETCH_TABLESPACES was added in 8.0.4.

- Error number: MY-010998; Symbol: ER_DD_UPGRADE_FAILED_TO_ACQUIRE_TABLESPACE; SQLSTATE: HY000

  Message: Error in acquiring Tablespace for SDI insertion %s.

  ER_DD_UPGRADE_FAILED_TO_ACQUIRE_TABLESPACE was added in 8.0.4.

- Error number: MY-010999; Symbol: ER_DD_UPGRADE_FAILED_TO_RESOLVE_TABLESPACE_ENGINE; SQLSTATE: HY000

  Message: Error in resolving Engine name for tablespace %s with engine %s.

  ER_DD_UPGRADE_FAILED_TO_RESOLVE_TABLESPACE_ENGINE was added in 8.0.4.

- Error number: MY-011000; Symbol: ER_FAILED_TO_CREATE_SDI_FOR_TABLESPACE; SQLSTATE: HY000

  Message: Error in creating SDI for %s tablespace.

  ER_FAILED_TO_CREATE_SDI_FOR_TABLESPACE was added in 8.0.4.

- Error number: MY-011001; Symbol: ER_FAILED_TO_STORE_SDI_FOR_TABLESPACE; SQLSTATE: HY000

  Message: Error in storing SDI for %s tablespace.

  ER_FAILED_TO_STORE_SDI_FOR_TABLESPACE was added in 8.0.4.

- Error number: MY-011002; Symbol: ER_DD_UPGRADE_FAILED_TO_FETCH_TABLES; SQLSTATE: HY000

  Message: Error in fetching list of tables.

  ER_DD_UPGRADE_FAILED_TO_FETCH_TABLES was added in 8.0.4.

- Error number: MY-011003; Symbol: ER_DD_UPGRADE_DD_POPULATED; SQLSTATE: HY000

  Message: Finished populating Data Dictionary tables with data.

  ER_DD_UPGRADE_DD_POPULATED was added in 8.0.4.

- Error number: MY-011004; Symbol: ER_DD_UPGRADE_INFO_FILE_OPEN_FAILED; SQLSTATE: HY000

  Message: Could not open the upgrade info file '%s' in the MySQL servers datadir, errno: %d.

  ER_DD_UPGRADE_INFO_FILE_OPEN_FAILED was added in 8.0.4.

- Error number: MY-011005; Symbol: ER_DD_UPGRADE_INFO_FILE_CLOSE_FAILED; SQLSTATE: HY000

  Message: Could not close the upgrade info file '%s' in the MySQL servers datadir, errno: %d.

ER_DD_UPGRADE_INFO_FILE_CLOSE_FAILED was added in 8.0.4.

- Error number: MY-011006; Symbol: ER_DD_UPGRADE_TABLESPACE_MIGRATION_FAILED; SQLSTATE: HY000

  Message: Got error %d from SE while migrating tablespaces.

  ER_DD_UPGRADE_TABLESPACE_MIGRATION_FAILED was added in 8.0.4.

- Error number: MY-011007; Symbol: ER_DD_UPGRADE_FAILED_TO_CREATE_TABLE_STATS; SQLSTATE: HY000

  Message: Error in creating TABLE statistics entry. Fix statistics data by using ANALYZE command.

  ER_DD_UPGRADE_FAILED_TO_CREATE_TABLE_STATS was added in 8.0.4.

- Error number: MY-011008; Symbol: ER_DD_UPGRADE_TABLE_STATS_MIGRATE_COMPLETED; SQLSTATE: HY000

  Message: Finished migrating TABLE statistics data.

  ER_DD_UPGRADE_TABLE_STATS_MIGRATE_COMPLETED was added in 8.0.4.

- Error number: MY-011009; Symbol: ER_DD_UPGRADE_FAILED_TO_CREATE_INDEX_STATS; SQLSTATE: HY000

  Message: Error in creating Index statistics entry. Fix statistics data by using ANALYZE command.

  ER_DD_UPGRADE_FAILED_TO_CREATE_INDEX_STATS was added in 8.0.4.

- Error number: MY-011010; Symbol: ER_DD_UPGRADE_INDEX_STATS_MIGRATE_COMPLETED; SQLSTATE: HY000

  Message: Finished migrating INDEX statistics data.

  ER_DD_UPGRADE_INDEX_STATS_MIGRATE_COMPLETED was added in 8.0.4.

- Error number: MY-011011; Symbol: ER_DD_UPGRADE_FAILED_FIND_VALID_DATA_DIR; SQLSTATE: HY000

  Message: Failed to find valid data directory.

  ER_DD_UPGRADE_FAILED_FIND_VALID_DATA_DIR was added in 8.0.4.

- Error number: MY-011012; Symbol: ER_DD_UPGRADE_START; SQLSTATE: HY000

  Message: Starting upgrade of data directory.

  ER_DD_UPGRADE_START was added in 8.0.4.

- Error number: MY-011013; Symbol: ER_DD_UPGRADE_FAILED_INIT_DD_SE; SQLSTATE: HY000

  Message: Failed to initialize DD Storage Engine.

  ER_DD_UPGRADE_FAILED_INIT_DD_SE was added in 8.0.4.

- Error number: MY-011014; Symbol: ER_DD_UPGRADE_FOUND_PARTIALLY_UPGRADED_DD_ABORT; SQLSTATE: HY000

Message: Found partially upgraded DD. Aborting upgrade and deleting all DD tables. Start the upgrade process again.

`ER_DD_UPGRADE_FOUND_PARTIALLY_UPGRADED_DD_ABORT` was added in 8.0.4.

- Error number: `MY-011015`; Symbol: `ER_DD_UPGRADE_FOUND_PARTIALLY_UPGRADED_DD_CONTINUE`; SQLSTATE: `HY000`

  Message: Found partially upgraded DD. Upgrade will continue and start the server.

  `ER_DD_UPGRADE_FOUND_PARTIALLY_UPGRADED_DD_CONTINUE` was added in 8.0.4.

- Error number: `MY-011016`; Symbol: `ER_DD_UPGRADE_SE_LOGS_FAILED`; SQLSTATE: `HY000`

  Message: Error in upgrading engine logs.

  `ER_DD_UPGRADE_SE_LOGS_FAILED` was added in 8.0.4.

- Error number: `MY-011017`; Symbol: `ER_DD_UPGRADE_SDI_INFO_UPDATE_FAILED`; SQLSTATE: `HY000`

  Message: Error in updating SDI information.

  `ER_DD_UPGRADE_SDI_INFO_UPDATE_FAILED` was added in 8.0.4.

- Error number: `MY-011018`; Symbol: `ER_SKIP_UPDATING_METADATA_IN_SE_RO_MODE`; SQLSTATE: `HY000`

  Message: Skip updating %s metadata in InnoDB read-only mode.

  `ER_SKIP_UPDATING_METADATA_IN_SE_RO_MODE` was added in 8.0.4.

- Error number: `MY-011019`; Symbol: `ER_CREATED_SYSTEM_WITH_VERSION`; SQLSTATE: `HY000`

  Message: Created system views with I_S version %d.

  `ER_CREATED_SYSTEM_WITH_VERSION` was added in 8.0.4.

- Error number: `MY-011020`; Symbol: `ER_UNKNOWN_ERROR_DETECTED_IN_SE`; SQLSTATE: `HY000`

  Message: Unknown error detected %d in handler.

  `ER_UNKNOWN_ERROR_DETECTED_IN_SE` was added in 8.0.4.

- Error number: `MY-011021`; Symbol: `ER_READ_LOG_EVENT_FAILED`; SQLSTATE: `HY000`

  Message: Error in Log_event::read_log_event(): '%s', data_len: %lu, event_type: %d.

  `ER_READ_LOG_EVENT_FAILED` was added in 8.0.4.

- Error number: `MY-011022`; Symbol: `ER_ROW_DATA_TOO_BIG_TO_WRITE_IN_BINLOG`; SQLSTATE: `HY000`

  Message: The row data is greater than 4GB, which is too big to write to the binary log.

  `ER_ROW_DATA_TOO_BIG_TO_WRITE_IN_BINLOG` was added in 8.0.4.

- Error number: `MY-011023`; Symbol: `ER_FAILED_TO_CONSTRUCT_DROP_EVENT_QUERY`; SQLSTATE: `HY000`

Message: Unable to construct DROP EVENT SQL query string.

ER_FAILED_TO_CONSTRUCT_DROP_EVENT_QUERY was added in 8.0.4.

- Error number: MY-011024; Symbol: ER_FAILED_TO_BINLOG_DROP_EVENT; SQLSTATE: HY000

  Message: Unable to binlog drop event %s.%s.

  ER_FAILED_TO_BINLOG_DROP_EVENT was added in 8.0.4.

- Error number: MY-011025; Symbol: ER_FAILED_TO_START_SLAVE_THREAD; SQLSTATE: HY000

  Message: Failed to start slave threads for channel '%s'.

  ER_FAILED_TO_START_SLAVE_THREAD was added in 8.0.4.

- Error number: MY-011026; Symbol: ER_RPL_IO_THREAD_KILLED; SQLSTATE: HY000

  Message: %s%s.

  ER_RPL_IO_THREAD_KILLED was added in 8.0.4.

- Error number: MY-011027; Symbol: ER_SLAVE_RECONNECT_FAILED; SQLSTATE: HY000

  Message: Failed registering on master, reconnecting to try again, log '%s' at position %s. %s.

  ER_SLAVE_RECONNECT_FAILED was added in 8.0.4.

- Error number: MY-011028; Symbol: ER_SLAVE_KILLED_AFTER_RECONNECT; SQLSTATE: HY000

  Message: Slave I/O thread killed during or after reconnect.

  ER_SLAVE_KILLED_AFTER_RECONNECT was added in 8.0.4.

- Error number: MY-011029; Symbol: ER_SLAVE_NOT_STARTED_ON_SOME_CHANNELS; SQLSTATE: HY000

  Message: Some of the channels are not created/initialized properly. Check for additional messages above. You will not be able to start replication on those channels until the issue is resolved and the server restarted.

  ER_SLAVE_NOT_STARTED_ON_SOME_CHANNELS was added in 8.0.4.

- Error number: MY-011030; Symbol: ER_FAILED_TO_ADD_RPL_FILTER; SQLSTATE: HY000

  Message: Failed to add a replication filter into filter map for channel '%s'.

  ER_FAILED_TO_ADD_RPL_FILTER was added in 8.0.4.

- Error number: MY-011031; Symbol: ER_PER_CHANNEL_RPL_FILTER_CONF_FOR_GRP_RPL; SQLSTATE: HY000

  Message: There are per-channel replication filter(s) configured for group replication channel '%s' which is disallowed. The filter(s) have been discarded.

  ER_PER_CHANNEL_RPL_FILTER_CONF_FOR_GRP_RPL was added in 8.0.4.

- Error number: MY-011032; Symbol: ER_RPL_FILTERS_NOT_ATTACHED_TO_CHANNEL; SQLSTATE: HY000

Message: There are per-channel replication filter(s) configured for channel '%s' which does not exist. The filter(s) have been discarded.

ER_RPL_FILTERS_NOT_ATTACHED_TO_CHANNEL was added in 8.0.4.

- Error number: MY-011033; Symbol: ER_FAILED_TO_BUILD_DO_AND_IGNORE_TABLE_HASHES; SQLSTATE: HY000

  Message: An error occurred while building do_table and ignore_table rules to hashes for per-channel filter.

  ER_FAILED_TO_BUILD_DO_AND_IGNORE_TABLE_HASHES was added in 8.0.4.

- Error number: MY-011034; Symbol: ER_CLONE_PLUGIN_NOT_LOADED; SQLSTATE: HY000

  Message: Clone plugin not loaded.

  ER_CLONE_PLUGIN_NOT_LOADED was added in 8.0.4, removed after 8.0.16.

- Error number: MY-011034; Symbol: ER_CLONE_PLUGIN_NOT_LOADED_TRACE; SQLSTATE: HY000

  Message: Clone plugin cannot be loaded.

  ER_CLONE_PLUGIN_NOT_LOADED_TRACE was added in 8.0.17.

- Error number: MY-011035; Symbol: ER_CLONE_HANDLER_EXISTS; SQLSTATE: HY000

  Message: Clone Handler exists.

  ER_CLONE_HANDLER_EXISTS was added in 8.0.4, removed after 8.0.16.

- Error number: MY-011035; Symbol: ER_CLONE_HANDLER_EXIST_TRACE; SQLSTATE: HY000

  Message: Clone Handler exists.

  ER_CLONE_HANDLER_EXIST_TRACE was added in 8.0.17.

- Error number: MY-011036; Symbol: ER_FAILED_TO_CREATE_CLONE_HANDLER; SQLSTATE: HY000

  Message: Could not create Clone Handler.

  ER_FAILED_TO_CREATE_CLONE_HANDLER was added in 8.0.4, removed after 8.0.16.

- Error number: MY-011036; Symbol: ER_CLONE_CREATE_HANDLER_FAIL_TRACE; SQLSTATE: HY000

  Message: Could not create Clone Handler.

  ER_CLONE_CREATE_HANDLER_FAIL_TRACE was added in 8.0.17.

- Error number: MY-011037; Symbol: ER_CYCLE_TIMER_IS_NOT_AVAILABLE; SQLSTATE: HY000

  Message: The CYCLE timer is not available. WAIT events in the performance_schema will not be timed.

  ER_CYCLE_TIMER_IS_NOT_AVAILABLE was added in 8.0.4.

- Error number: MY-011038; Symbol: ER_NANOSECOND_TIMER_IS_NOT_AVAILABLE; SQLSTATE: HY000

Message: The NANOSECOND timer is not available. IDLE/STAGE/STATEMENT/TRANSACTION events in the performance_schema will not be timed.

ER_NANOSECOND_TIMER_IS_NOT_AVAILABLE was added in 8.0.4.

- Error number: MY-011039; Symbol: ER_MICROSECOND_TIMER_IS_NOT_AVAILABLE; SQLSTATE: HY000

  Message: The MICROSECOND timer is not available. IDLE/STAGE/STATEMENT/TRANSACTION events in the performance_schema will not be timed.

  ER_MICROSECOND_TIMER_IS_NOT_AVAILABLE was added in 8.0.4.

- Error number: MY-011040; Symbol: ER_PFS_MALLOC_ARRAY_OVERFLOW; SQLSTATE: HY000

  Message: Failed to allocate memory for %zu chunks each of size %zu for buffer '%s' due to overflow.

  ER_PFS_MALLOC_ARRAY_OVERFLOW was added in 8.0.4.

- Error number: MY-011041; Symbol: ER_PFS_MALLOC_ARRAY_OOM; SQLSTATE: HY000

  Message: Failed to allocate %zu bytes for buffer '%s' due to out-of-memory.

  ER_PFS_MALLOC_ARRAY_OOM was added in 8.0.4.

- Error number: MY-011042; Symbol: ER_INNODB_FAILED_TO_FIND_IDX_WITH_KEY_NO; SQLSTATE: HY000

  Message: InnoDB could not find index %s key no %u for table %s through its index translation table.

  ER_INNODB_FAILED_TO_FIND_IDX_WITH_KEY_NO was added in 8.0.4.

- Error number: MY-011043; Symbol: ER_INNODB_FAILED_TO_FIND_IDX; SQLSTATE: HY000

  Message: Cannot find index %s in InnoDB index translation table.

  ER_INNODB_FAILED_TO_FIND_IDX was added in 8.0.4.

- Error number: MY-011044; Symbol: ER_INNODB_FAILED_TO_FIND_IDX_FROM_DICT_CACHE; SQLSTATE: HY000

  Message: InnoDB could not find key no %u with name %s from dict cache for table %s.

  ER_INNODB_FAILED_TO_FIND_IDX_FROM_DICT_CACHE was added in 8.0.4.

- Error number: MY-011045; Symbol: ER_INNODB_ACTIVE_INDEX_CHANGE_FAILED; SQLSTATE: HY000

  Message: InnoDB: change_active_index(%u) failed.

  ER_INNODB_ACTIVE_INDEX_CHANGE_FAILED was added in 8.0.4.

- Error number: MY-011046; Symbol: ER_INNODB_DIFF_IN_REF_LEN; SQLSTATE: HY000

  Message: Stored ref len is %lu, but table ref len is %lu.

  ER_INNODB_DIFF_IN_REF_LEN was added in 8.0.4.

- Error number: MY-011047; Symbol: ER_WRONG_TYPE_FOR_COLUMN_PREFIX_IDX_FLD; SQLSTATE: HY000

  Message: MySQL is trying to create a column prefix index field, on an inappropriate data type. Table name %s, column name %s.

  ER_WRONG_TYPE_FOR_COLUMN_PREFIX_IDX_FLD was added in 8.0.4.

- Error number: MY-011048; Symbol: ER_INNODB_CANNOT_CREATE_TABLE; SQLSTATE: HY000

  Message: Cannot create table %s.

  ER_INNODB_CANNOT_CREATE_TABLE was added in 8.0.4.

- Error number: MY-011049; Symbol: ER_INNODB_INTERNAL_INDEX; SQLSTATE: HY000

  Message: Found index %s in InnoDB index list but not its MySQL index number. It could be an InnoDB internal index.

  ER_INNODB_INTERNAL_INDEX was added in 8.0.4.

- Error number: MY-011050; Symbol: ER_INNODB_IDX_CNT_MORE_THAN_DEFINED_IN_MYSQL; SQLSTATE: HY000

  Message: InnoDB: Table %s contains %lu indexes inside InnoDB, which is different from the number of indexes %u defined in MySQL.

  ER_INNODB_IDX_CNT_MORE_THAN_DEFINED_IN_MYSQL was added in 8.0.4.

- Error number: MY-011051; Symbol: ER_INNODB_IDX_CNT_FEWER_THAN_DEFINED_IN_MYSQL; SQLSTATE: HY000

  Message: Table %s contains fewer indexes inside InnoDB than are defined in the MySQL. Have you mixed up with data dictionary from different installation?

  ER_INNODB_IDX_CNT_FEWER_THAN_DEFINED_IN_MYSQL was added in 8.0.4.

- Error number: MY-011052; Symbol: ER_INNODB_IDX_COLUMN_CNT_DIFF; SQLSTATE: HY000

  Message: Index %s of %s has %lu columns unique inside InnoDB, but MySQL is asking statistics for %lu columns. Have you mixed data dictionary from different installation?

  ER_INNODB_IDX_COLUMN_CNT_DIFF was added in 8.0.4.

- Error number: MY-011053; Symbol: ER_INNODB_USE_MONITOR_GROUP_NAME; SQLSTATE: HY000

  Message: Monitor counter '%s' cannot be turned on/off individually. Please use its module name to turn on/off the counters in the module as a group.

  ER_INNODB_USE_MONITOR_GROUP_NAME was added in 8.0.4.

- Error number: MY-011054; Symbol: ER_INNODB_MONITOR_DEFAULT_VALUE_NOT_DEFINED; SQLSTATE: HY000

  Message: Default value is not defined for this set option. Please specify correct counter or module name.

  ER_INNODB_MONITOR_DEFAULT_VALUE_NOT_DEFINED was added in 8.0.4.

- Error number: MY-011055; Symbol: ER_INNODB_MONITOR_IS_ENABLED; SQLSTATE: HY000

Message: InnoDB: Monitor %s is already enabled.

ER_INNODB_MONITOR_IS_ENABLED was added in 8.0.4.

- Error number: MY-011056; Symbol: ER_INNODB_INVALID_MONITOR_COUNTER_NAME; SQLSTATE: HY000

  Message: Invalid monitor counter : %s.

  ER_INNODB_INVALID_MONITOR_COUNTER_NAME was added in 8.0.4.

- Error number: MY-011057; Symbol: ER_WIN_LOAD_LIBRARY_FAILED; SQLSTATE: HY000

  Message: LoadLibrary("%s") failed: GetLastError returns %lu.

  ER_WIN_LOAD_LIBRARY_FAILED was added in 8.0.4.

- Error number: MY-011058; Symbol: ER_PARTITION_HANDLER_ADMIN_MSG; SQLSTATE: HY000

  Message: %s.

  ER_PARTITION_HANDLER_ADMIN_MSG was added in 8.0.4.

- Error number: MY-011059; Symbol: ER_RPL_RLI_INIT_INFO_MSG; SQLSTATE: HY000

  Message: %s.

  ER_RPL_RLI_INIT_INFO_MSG was added in 8.0.4.

- Error number: MY-011060; Symbol: ER_DD_UPGRADE_TABLE_INTACT_ERROR; SQLSTATE: HY000

  Message: %s.

  ER_DD_UPGRADE_TABLE_INTACT_ERROR was added in 8.0.4.

- Error number: MY-011061; Symbol: ER_SERVER_INIT_COMPILED_IN_COMMANDS; SQLSTATE: HY000

  Message: %s.

  ER_SERVER_INIT_COMPILED_IN_COMMANDS was added in 8.0.4.

- Error number: MY-011062; Symbol: ER_MYISAM_CHECK_METHOD_ERROR; SQLSTATE: HY000

  Message: %s.

  ER_MYISAM_CHECK_METHOD_ERROR was added in 8.0.4.

- Error number: MY-011063; Symbol: ER_MYISAM_CRASHED_ERROR; SQLSTATE: HY000

  Message: %s.

  ER_MYISAM_CRASHED_ERROR was added in 8.0.4.

- Error number: MY-011064; Symbol: ER_WAITPID_FAILED; SQLSTATE: HY000

  Message: Unable to wait for process %lld.

  ER_WAITPID_FAILED was added in 8.0.4.

- Error number: MY-011065; Symbol: ER_FAILED_TO_FIND_MYSQLD_STATUS; SQLSTATE: HY000

  Message: Unable to determine if daemon is running: %s (rc=%d).

  ER_FAILED_TO_FIND_MYSQLD_STATUS was added in 8.0.4.

- Error number: MY-011066; Symbol: ER_INNODB_ERROR_LOGGER_MSG; SQLSTATE: HY000

  Message: %s

  ER_INNODB_ERROR_LOGGER_MSG was added in 8.0.4.

- Error number: MY-011067; Symbol: ER_INNODB_ERROR_LOGGER_FATAL_MSG; SQLSTATE: HY000

  Message: [FATAL] InnoDB: %s

  ER_INNODB_ERROR_LOGGER_FATAL_MSG was added in 8.0.4.

- Error number: MY-011068; Symbol: ER_DEPRECATED_SYNTAX_WITH_REPLACEMENT; SQLSTATE: HY000

  Message: The syntax '%s' is deprecated and will be removed in a future release. Please use %s instead.

  ER_DEPRECATED_SYNTAX_WITH_REPLACEMENT was added in 8.0.4.

- Error number: MY-011069; Symbol: ER_DEPRECATED_SYNTAX_NO_REPLACEMENT; SQLSTATE: HY000

  Message: The syntax '%s' is deprecated and will be removed in a future release.

  ER_DEPRECATED_SYNTAX_NO_REPLACEMENT was added in 8.0.4.

- Error number: MY-011070; Symbol: ER_DEPRECATE_MSG_NO_REPLACEMENT; SQLSTATE: HY000

  Message: '%s' is deprecated and will be removed in a future release.

  ER_DEPRECATE_MSG_NO_REPLACEMENT was added in 8.0.4.

- Error number: MY-011071; Symbol: ER_LOG_PRINTF_MSG; SQLSTATE: HY000

  Message: %s

  ER_LOG_PRINTF_MSG was added in 8.0.4.

- Error number: MY-011072; Symbol: ER_BINLOG_LOGGING_NOT_POSSIBLE; SQLSTATE: HY000

  Message: Binary logging not possible. Message: %s.

  ER_BINLOG_LOGGING_NOT_POSSIBLE was added in 8.0.4.

- Error number: MY-011073; Symbol: ER_FAILED_TO_SET_PERSISTED_OPTIONS; SQLSTATE: HY000

  Message: Failed to set persisted options.

  ER_FAILED_TO_SET_PERSISTED_OPTIONS was added in 8.0.4.

- Error number: MY-011074; Symbol: ER_COMPONENTS_FAILED_TO_ACQUIRE_SERVICE_IMPLEMENTATION; SQLSTATE: HY000

Message: Cannot acquire specified service implementation: '%s'.

ER_COMPONENTS_FAILED_TO_ACQUIRE_SERVICE_IMPLEMENTATION was added in 8.0.4.

- Error number: MY-011075; Symbol: ER_RES_GRP_INVALID_VCPU_RANGE; SQLSTATE: HY000

  Message: Invalid VCPU range %u-%u.

  ER_RES_GRP_INVALID_VCPU_RANGE was added in 8.0.4.

- Error number: MY-011076; Symbol: ER_RES_GRP_INVALID_VCPU_ID; SQLSTATE: HY000

  Message: Invalid cpu id %u.

  ER_RES_GRP_INVALID_VCPU_ID was added in 8.0.4.

- Error number: MY-011077; Symbol: ER_ERROR_DURING_FLUSH_LOG_COMMIT_PHASE; SQLSTATE: HY000

  Message: Got error %d during FLUSH_LOGS.

  ER_ERROR_DURING_FLUSH_LOG_COMMIT_PHASE was added in 8.0.4.

- Error number: MY-011078; Symbol: ER_DROP_DATABASE_FAILED_RMDIR_MANUALLY; SQLSTATE: HY000

  Message: Problem while dropping database. Can't remove database directory (%s). Please remove it manually.

  ER_DROP_DATABASE_FAILED_RMDIR_MANUALLY was added in 8.0.4.

- Error number: MY-011079; Symbol: ER_BINLOG_EXPIRAY_LOG_DAYS_AND_SECS_USED_TOGETHER; SQLSTATE: HY000

  Message: The option expire_logs_days cannot be used together with option binlog_expire_logs_seconds. Therefore, value of expire_logs_days is ignored.

  ER_BINLOG_EXPIRAY_LOG_DAYS_AND_SECS_USED_TOGETHER was added in 8.0.4, removed after 8.0.4.

- Error number: MY-011079; Symbol: ER_EXPIRE_LOGS_DAYS_IGNORED; SQLSTATE: HY000

  Message: The option expire_logs_days cannot be used together with option binlog_expire_logs_seconds. Therefore, value of expire_logs_days is ignored.

  ER_EXPIRE_LOGS_DAYS_IGNORED was added in 8.0.4.

- Error number: MY-011080; Symbol: ER_BINLOG_MALFORMED_OR_OLD_RELAY_LOG; SQLSTATE: HY000

  Message: malformed or very old relay log which does not have FormatDescriptor.

  ER_BINLOG_MALFORMED_OR_OLD_RELAY_LOG was added in 8.0.4.

- Error number: MY-011081; Symbol: ER_DD_UPGRADE_VIEW_COLUMN_NAME_TOO_LONG; SQLSTATE: HY000

Message: Upgrade of view '%s.%s' failed. Re-create the view with the explicit column name lesser than 64 characters.

ER_DD_UPGRADE_VIEW_COLUMN_NAME_TOO_LONG was added in 8.0.4.

- Error number: MY-011082; Symbol: ER_TABLE_NEEDS_DUMP_UPGRADE; SQLSTATE: HY000

  Message: Table upgrade required for `%s`.`%s`. Please dump/reload table to fix it!

  ER_TABLE_NEEDS_DUMP_UPGRADE was added in 8.0.4.

- Error number: MY-011083; Symbol: ER_DD_UPGRADE_FAILED_TO_UPDATE_VER_NO_IN_TABLESPACE; SQLSTATE: HY000

  Message: Error in updating version number in %s tablespace.

  ER_DD_UPGRADE_FAILED_TO_UPDATE_VER_NO_IN_TABLESPACE was added in 8.0.4.

- Error number: MY-011084; Symbol: ER_KEYRING_MIGRATION_FAILED; SQLSTATE: HY000

  Message: Keyring migration failed.

  ER_KEYRING_MIGRATION_FAILED was added in 8.0.4.

- Error number: MY-011085; Symbol: ER_KEYRING_MIGRATION_SUCCESSFUL; SQLSTATE: HY000

  Message: Keyring migration successful.

  ER_KEYRING_MIGRATION_SUCCESSFUL was added in 8.0.4.

- Error number: MY-011086; Symbol: ER_RESTART_RECEIVED_INFO; SQLSTATE: HY000

  Message: Received RESTART from user %s. Restarting mysqld (Version: %s).

  ER_RESTART_RECEIVED_INFO was added in 8.0.4.

- Error number: MY-011087; Symbol: ER_LCTN_CHANGED; SQLSTATE: HY000

  Message: Different lower_case_table_names settings for server ('%u') and data dictionary ('%u').

  ER_LCTN_CHANGED was added in 8.0.4.

- Error number: MY-011088; Symbol: ER_DD_INITIALIZE; SQLSTATE: HY000

  Message: Data dictionary initializing version '%u'.

  ER_DD_INITIALIZE was added in 8.0.4.

- Error number: MY-011089; Symbol: ER_DD_RESTART; SQLSTATE: HY000

  Message: Data dictionary restarting version '%u'.

  ER_DD_RESTART was added in 8.0.4.

- Error number: MY-011090; Symbol: ER_DD_UPGRADE; SQLSTATE: HY000

  Message: Data dictionary upgrading from version '%u' to '%u'.

  ER_DD_UPGRADE was added in 8.0.4.

- Error number: MY-011091; Symbol: ER_DD_UPGRADE_OFF; SQLSTATE: HY000

  Message: Data dictionary upgrade prohibited by the command line option '--no_dd_upgrade'.

  ER_DD_UPGRADE_OFF was added in 8.0.4.

- Error number: MY-011092; Symbol: ER_DD_UPGRADE_VERSION_NOT_SUPPORTED; SQLSTATE: HY000

  Message: Upgrading the data dictionary from dictionary version '%u' is not supported.

  ER_DD_UPGRADE_VERSION_NOT_SUPPORTED was added in 8.0.4.

- Error number: MY-011093; Symbol: ER_DD_UPGRADE_SCHEMA_UNAVAILABLE; SQLSTATE: HY000

  Message: Upgrading the data dictionary failed, temporary schema name '%s' not available.

  ER_DD_UPGRADE_SCHEMA_UNAVAILABLE was added in 8.0.4.

- Error number: MY-011094; Symbol: ER_DD_MINOR_DOWNGRADE; SQLSTATE: HY000

  Message: Data dictionary minor downgrade from version '%u' to '%u'.

  ER_DD_MINOR_DOWNGRADE was added in 8.0.4.

- Error number: MY-011095; Symbol: ER_DD_MINOR_DOWNGRADE_VERSION_NOT_SUPPORTED; SQLSTATE: HY000

  Message: Minor downgrade of the Data dictionary from dictionary version '%u' is not supported.

  ER_DD_MINOR_DOWNGRADE_VERSION_NOT_SUPPORTED was added in 8.0.4.

- Error number: MY-011096; Symbol: ER_DD_NO_VERSION_FOUND; SQLSTATE: HY000

  Message: No data dictionary version number found.

  ER_DD_NO_VERSION_FOUND was added in 8.0.4.

- Error number: MY-011097; Symbol: ER_THREAD_POOL_NOT_SUPPORTED_ON_PLATFORM; SQLSTATE: HY000

  Message: Thread pool not supported, requires a minimum of %s.

  ER_THREAD_POOL_NOT_SUPPORTED_ON_PLATFORM was added in 8.0.4.

- Error number: MY-011098; Symbol: ER_THREAD_POOL_SIZE_TOO_LOW; SQLSTATE: HY000

  Message: thread_pool_size=0 means thread pool disabled, Allowed range of thread_pool_size is %d-%d.

  ER_THREAD_POOL_SIZE_TOO_LOW was added in 8.0.4.

- Error number: MY-011099; Symbol: ER_THREAD_POOL_SIZE_TOO_HIGH; SQLSTATE: HY000

  Message: thread_pool_size=%lu is too high, %d is maximum, thread pool is disabled. Allowed range of thread_pool_size is %d-%d.

  ER_THREAD_POOL_SIZE_TOO_HIGH was added in 8.0.4.

- Error number: MY-011100; Symbol: ER_THREAD_POOL_ALGORITHM_INVALID; SQLSTATE: HY000

Message: thread_pool_algorithm can be set to 0 and 1, 0 indicates the default low concurrency algorithm, 1 means a high concurrency algorithm.

`ER_THREAD_POOL_ALGORITHM_INVALID` was added in 8.0.4.

- Error number: `MY-011101`; Symbol: `ER_THREAD_POOL_INVALID_STALL_LIMIT`; SQLSTATE: `HY000`

  Message: thread_pool_stall_limit can be %d at minimum and %d at maximum, smaller values would render the thread pool fairly useless and higher values could make it possible to have undetected deadlock issues in the MySQL Server.

  `ER_THREAD_POOL_INVALID_STALL_LIMIT` was added in 8.0.4.

- Error number: `MY-011102`; Symbol: `ER_THREAD_POOL_INVALID_PRIO_KICKUP_TIMER`; SQLSTATE: `HY000`

  Message: Invalid value of thread_pool_prio_kickup_timer specified. Value of thread_pool_prio_kickup_timer should be in range 0-4294967294.

  `ER_THREAD_POOL_INVALID_PRIO_KICKUP_TIMER` was added in 8.0.4.

- Error number: `MY-011103`; Symbol: `ER_THREAD_POOL_MAX_UNUSED_THREADS_INVALID`; SQLSTATE: `HY000`

  Message: thread_pool_max_unused_threads cannot be set higher than %d.

  `ER_THREAD_POOL_MAX_UNUSED_THREADS_INVALID` was added in 8.0.4.

- Error number: `MY-011104`; Symbol: `ER_THREAD_POOL_CON_HANDLER_INIT_FAILED`; SQLSTATE: `HY000`

  Message: Failed to instantiate the connection handler object.

  `ER_THREAD_POOL_CON_HANDLER_INIT_FAILED` was added in 8.0.4.

- Error number: `MY-011105`; Symbol: `ER_THREAD_POOL_INIT_FAILED`; SQLSTATE: `HY000`

  Message: Failed to initialize thread pool plugin.

  `ER_THREAD_POOL_INIT_FAILED` was added in 8.0.4.

- Error number: `MY-011106`; Symbol: `ER_THREAD_POOL_PLUGIN_STARTED`; SQLSTATE: `HY000`

  Message: Thread pool plugin started successfully with parameters: thread_pool_size = %lu, thread_pool_algorithm = %s, thread_pool_stall_limit = %u, thread_pool_prio_kickup_timer = %u, thread_pool_max_unused_threads = %u, thread_pool_high_priority_connection = %d.

  `ER_THREAD_POOL_PLUGIN_STARTED` was added in 8.0.4.

- Error number: `MY-011107`; Symbol: `ER_THREAD_POOL_CANNOT_SET_THREAD_SPECIFIC_DATA`; SQLSTATE: `HY000`

  Message: Can't setup connection teardown thread-specific data.

  `ER_THREAD_POOL_CANNOT_SET_THREAD_SPECIFIC_DATA` was added in 8.0.4.

- Error number: MY-011108; Symbol: ER_THREAD_POOL_FAILED_TO_CREATE_CONNECT_HANDLER_THD; SQLSTATE: HY000

  Message: Creation of connect handler thread failed.

  ER_THREAD_POOL_FAILED_TO_CREATE_CONNECT_HANDLER_THD was added in 8.0.4.

- Error number: MY-011109; Symbol: ER_THREAD_POOL_FAILED_TO_CREATE_THD_AND_AUTH_CONN; SQLSTATE: HY000

  Message: Failed to create thd and authenticate connection.

  ER_THREAD_POOL_FAILED_TO_CREATE_THD_AND_AUTH_CONN was added in 8.0.4.

- Error number: MY-011110; Symbol: ER_THREAD_POOL_FAILED_PROCESS_CONNECT_EVENT; SQLSTATE: HY000

  Message: Failed to process connection event.

  ER_THREAD_POOL_FAILED_PROCESS_CONNECT_EVENT was added in 8.0.4.

- Error number: MY-011111; Symbol: ER_THREAD_POOL_FAILED_TO_CREATE_POOL; SQLSTATE: HY000

  Message: Can't create pool thread (error %d, errno: %d).

  ER_THREAD_POOL_FAILED_TO_CREATE_POOL was added in 8.0.4.

- Error number: MY-011112; Symbol: ER_THREAD_POOL_RATE_LIMITED_ERROR_MSGS; SQLSTATE: HY000

  Message: %.*s.

  ER_THREAD_POOL_RATE_LIMITED_ERROR_MSGS was added in 8.0.4.

- Error number: MY-011113; Symbol: ER_TRHEAD_POOL_LOW_LEVEL_INIT_FAILED; SQLSTATE: HY000

  Message: tp_group_low_level_init() failed.

  ER_TRHEAD_POOL_LOW_LEVEL_INIT_FAILED was added in 8.0.4.

- Error number: MY-011114; Symbol: ER_THREAD_POOL_LOW_LEVEL_REARM_FAILED; SQLSTATE: HY000

  Message: Rearm failed even after 30 seconds, can't continue without notify socket.

  ER_THREAD_POOL_LOW_LEVEL_REARM_FAILED was added in 8.0.4.

- Error number: MY-011115; Symbol: ER_THREAD_POOL_BUFFER_TOO_SMALL; SQLSTATE: HY000

  Message: %s: %s buffer is too small

  ER_THREAD_POOL_BUFFER_TOO_SMALL was added in 8.0.4.

- Error number: MY-011116; Symbol: ER_MECAB_NOT_SUPPORTED; SQLSTATE: HY000

  Message: Mecab v%s is not supported, the lowest version supported is v%s.

ER_MECAB_NOT_SUPPORTED was added in 8.0.4.

- Error number: MY-011117; Symbol: ER_MECAB_NOT_VERIFIED; SQLSTATE: HY000

  Message: Mecab v%s is not verified, the highest version supported is v%s.

  ER_MECAB_NOT_VERIFIED was added in 8.0.4.

- Error number: MY-011118; Symbol: ER_MECAB_CREATING_MODEL; SQLSTATE: HY000

  Message: Mecab: Trying createModel(%s).

  ER_MECAB_CREATING_MODEL was added in 8.0.4.

- Error number: MY-011119; Symbol: ER_MECAB_FAILED_TO_CREATE_MODEL; SQLSTATE: HY000

  Message: Mecab: createModel() failed: %s.

  ER_MECAB_FAILED_TO_CREATE_MODEL was added in 8.0.4.

- Error number: MY-011120; Symbol: ER_MECAB_FAILED_TO_CREATE_TRIGGER; SQLSTATE: HY000

  Message: Mecab: createTagger() failed: %s.

  ER_MECAB_FAILED_TO_CREATE_TRIGGER was added in 8.0.4.

- Error number: MY-011121; Symbol: ER_MECAB_UNSUPPORTED_CHARSET; SQLSTATE: HY000

  Message: Mecab: Unsupported dictionary charset %s.

  ER_MECAB_UNSUPPORTED_CHARSET was added in 8.0.4.

- Error number: MY-011122; Symbol: ER_MECAB_CHARSET_LOADED; SQLSTATE: HY000

  Message: Mecab: Loaded dictionary charset is %s.

  ER_MECAB_CHARSET_LOADED was added in 8.0.4.

- Error number: MY-011123; Symbol: ER_MECAB_PARSE_FAILED; SQLSTATE: HY000

  Message: Mecab: parse() failed: %s.

  ER_MECAB_PARSE_FAILED was added in 8.0.4.

- Error number: MY-011124; Symbol: ER_MECAB_OOM_WHILE_PARSING_TEXT; SQLSTATE: HY000

  Message: Mecab: parse() failed: out of memory.

  ER_MECAB_OOM_WHILE_PARSING_TEXT was added in 8.0.4.

- Error number: MY-011125; Symbol: ER_MECAB_CREATE_LATTICE_FAILED; SQLSTATE: HY000

  Message: Mecab: createLattice() failed: %s.

  ER_MECAB_CREATE_LATTICE_FAILED was added in 8.0.4.

- Error number: MY-011126; Symbol: ER_SEMISYNC_TRACE_ENTER_FUNC; SQLSTATE: HY000

  Message: ---> %s enter.

ER_SEMISYNC_TRACE_ENTER_FUNC was added in 8.0.4.

- Error number: MY-011127; Symbol: ER_SEMISYNC_TRACE_EXIT_WITH_INT_EXIT_CODE; SQLSTATE: HY000

  Message: <--- %s exit (%d).

  ER_SEMISYNC_TRACE_EXIT_WITH_INT_EXIT_CODE was added in 8.0.4.

- Error number: MY-011128; Symbol: ER_SEMISYNC_TRACE_EXIT_WITH_BOOL_EXIT_CODE; SQLSTATE: HY000

  Message: <--- %s exit (%s).

  ER_SEMISYNC_TRACE_EXIT_WITH_BOOL_EXIT_CODE was added in 8.0.4.

- Error number: MY-011129; Symbol: ER_SEMISYNC_TRACE_EXIT; SQLSTATE: HY000

  Message: <--- %s exit.

  ER_SEMISYNC_TRACE_EXIT was added in 8.0.4.

- Error number: MY-011130; Symbol: ER_SEMISYNC_RPL_INIT_FOR_TRX; SQLSTATE: HY000

  Message: Semi-sync replication initialized for transactions.

  ER_SEMISYNC_RPL_INIT_FOR_TRX was added in 8.0.4.

- Error number: MY-011131; Symbol: ER_SEMISYNC_FAILED_TO_ALLOCATE_TRX_NODE; SQLSTATE: HY000

  Message: %s: transaction node allocation failed for: (%s, %lu).

  ER_SEMISYNC_FAILED_TO_ALLOCATE_TRX_NODE was added in 8.0.4.

- Error number: MY-011132; Symbol: ER_SEMISYNC_BINLOG_WRITE_OUT_OF_ORDER; SQLSTATE: HY000

  Message: %s: binlog write out-of-order, tail (%s, %lu), new node (%s, %lu).

  ER_SEMISYNC_BINLOG_WRITE_OUT_OF_ORDER was added in 8.0.4.

- Error number: MY-011133; Symbol: ER_SEMISYNC_INSERT_LOG_INFO_IN_ENTRY; SQLSTATE: HY000

  Message: %s: insert (%s, %lu) in entry(%u).

  ER_SEMISYNC_INSERT_LOG_INFO_IN_ENTRY was added in 8.0.4.

- Error number: MY-011134; Symbol: ER_SEMISYNC_PROBE_LOG_INFO_IN_ENTRY; SQLSTATE: HY000

  Message: %s: probe (%s, %lu) in entry(%u).

  ER_SEMISYNC_PROBE_LOG_INFO_IN_ENTRY was added in 8.0.4.

- Error number: MY-011135; Symbol: ER_SEMISYNC_CLEARED_ALL_ACTIVE_TRANSACTION_NODES; SQLSTATE: HY000

Message: %s: cleared all nodes.

ER_SEMISYNC_CLEARED_ALL_ACTIVE_TRANSACTION_NODES was added in 8.0.4.

- Error number: MY-011136; Symbol: ER_SEMISYNC_CLEARED_ACTIVE_TRANSACTION_TILL_POS; SQLSTATE: HY000

  Message: %s: cleared %d nodes back until pos (%s, %lu).

  ER_SEMISYNC_CLEARED_ACTIVE_TRANSACTION_TILL_POS was added in 8.0.4.

- Error number: MY-011137; Symbol: ER_SEMISYNC_REPLY_MAGIC_NO_ERROR; SQLSTATE: HY000

  Message: Read semi-sync reply magic number error.

  ER_SEMISYNC_REPLY_MAGIC_NO_ERROR was added in 8.0.4.

- Error number: MY-011138; Symbol: ER_SEMISYNC_REPLY_PKT_LENGTH_TOO_SMALL; SQLSTATE: HY000

  Message: Read semi-sync reply length error: packet is too small.

  ER_SEMISYNC_REPLY_PKT_LENGTH_TOO_SMALL was added in 8.0.4.

- Error number: MY-011139; Symbol: ER_SEMISYNC_REPLY_BINLOG_FILE_TOO_LARGE; SQLSTATE: HY000

  Message: Read semi-sync reply binlog file length too large.

  ER_SEMISYNC_REPLY_BINLOG_FILE_TOO_LARGE was added in 8.0.4.

- Error number: MY-011140; Symbol: ER_SEMISYNC_SERVER_REPLY; SQLSTATE: HY000

  Message: %s: Got reply(%s, %lu) from server %u.

  ER_SEMISYNC_SERVER_REPLY was added in 8.0.4.

- Error number: MY-011141; Symbol: ER_SEMISYNC_FUNCTION_CALLED_TWICE; SQLSTATE: HY000

  Message: %s called twice.

  ER_SEMISYNC_FUNCTION_CALLED_TWICE was added in 8.0.4.

- Error number: MY-011142; Symbol: ER_SEMISYNC_RPL_ENABLED_ON_MASTER; SQLSTATE: HY000

  Message: Semi-sync replication enabled on the master.

  ER_SEMISYNC_RPL_ENABLED_ON_MASTER was added in 8.0.4.

- Error number: MY-011143; Symbol: ER_SEMISYNC_MASTER_OOM; SQLSTATE: HY000

  Message: Cannot allocate memory to enable semi-sync on the master.

  ER_SEMISYNC_MASTER_OOM was added in 8.0.4.

- Error number: MY-011144; Symbol: ER_SEMISYNC_DISABLED_ON_MASTER; SQLSTATE: HY000

  Message: Semi-sync replication disabled on the master.

ER_SEMISYNC_DISABLED_ON_MASTER was added in 8.0.4.

- Error number: MY-011145; Symbol: ER_SEMISYNC_FORCED_SHUTDOWN; SQLSTATE: HY000

  Message: SEMISYNC: Forced shutdown. Some updates might not be replicated.

  ER_SEMISYNC_FORCED_SHUTDOWN was added in 8.0.4.

- Error number: MY-011146; Symbol: ER_SEMISYNC_MASTER_GOT_REPLY_AT_POS; SQLSTATE: HY000

  Message: %s: Got reply at (%s, %lu).

  ER_SEMISYNC_MASTER_GOT_REPLY_AT_POS was added in 8.0.4.

- Error number: MY-011147; Symbol: ER_SEMISYNC_MASTER_SIGNAL_ALL_WAITING_THREADS; SQLSTATE: HY000

  Message: %s: signal all waiting threads.

  ER_SEMISYNC_MASTER_SIGNAL_ALL_WAITING_THREADS was added in 8.0.4.

- Error number: MY-011148; Symbol: ER_SEMISYNC_MASTER_TRX_WAIT_POS; SQLSTATE: HY000

  Message: %s: wait pos (%s, %lu), repl(%d).

  ER_SEMISYNC_MASTER_TRX_WAIT_POS was added in 8.0.4.

- Error number: MY-011149; Symbol: ER_SEMISYNC_BINLOG_REPLY_IS_AHEAD; SQLSTATE: HY000

  Message: %s: Binlog reply is ahead (%s, %lu).

  ER_SEMISYNC_BINLOG_REPLY_IS_AHEAD was added in 8.0.4.

- Error number: MY-011150; Symbol: ER_SEMISYNC_MOVE_BACK_WAIT_POS; SQLSTATE: HY000

  Message: %s: move back wait position (%s, %lu).

  ER_SEMISYNC_MOVE_BACK_WAIT_POS was added in 8.0.4.

- Error number: MY-011151; Symbol: ER_SEMISYNC_INIT_WAIT_POS; SQLSTATE: HY000

  Message: %s: init wait position (%s, %lu).

  ER_SEMISYNC_INIT_WAIT_POS was added in 8.0.4.

- Error number: MY-011152; Symbol: ER_SEMISYNC_WAIT_TIME_FOR_BINLOG_SENT; SQLSTATE: HY000

  Message: %s: wait %lu ms for binlog sent (%s, %lu).

  ER_SEMISYNC_WAIT_TIME_FOR_BINLOG_SENT was added in 8.0.4.

- Error number: MY-011153; Symbol: ER_SEMISYNC_WAIT_FOR_BINLOG_TIMEDOUT; SQLSTATE: HY000

  Message: Timeout waiting for reply of binlog (file: %s, pos: %lu), semi-sync up to file %s, position %lu.

  ER_SEMISYNC_WAIT_FOR_BINLOG_TIMEDOUT was added in 8.0.4.

- Error number: MY-011154; Symbol: ER_SEMISYNC_WAIT_TIME_ASSESSMENT_FOR_COMMIT_TRX_FAILED; SQLSTATE: HY000

  Message: Assessment of waiting time for commitTrx failed at wait position (%s, %lu).

  ER_SEMISYNC_WAIT_TIME_ASSESSMENT_FOR_COMMIT_TRX_FAILED was added in 8.0.4.

- Error number: MY-011155; Symbol: ER_SEMISYNC_RPL_SWITCHED_OFF; SQLSTATE: HY000

  Message: Semi-sync replication switched OFF.

  ER_SEMISYNC_RPL_SWITCHED_OFF was added in 8.0.4.

- Error number: MY-011156; Symbol: ER_SEMISYNC_RPL_SWITCHED_ON; SQLSTATE: HY000

  Message: Semi-sync replication switched ON at (%s, %lu).

  ER_SEMISYNC_RPL_SWITCHED_ON was added in 8.0.4.

- Error number: MY-011157; Symbol: ER_SEMISYNC_NO_SPACE_IN_THE_PKT; SQLSTATE: HY000

  Message: No enough space in the packet for semi-sync extra header, semi-sync replication disabled.

  ER_SEMISYNC_NO_SPACE_IN_THE_PKT was added in 8.0.4.

- Error number: MY-011158; Symbol: ER_SEMISYNC_SYNC_HEADER_UPDATE_INFO; SQLSTATE: HY000

  Message: %s: server(%d), (%s, %lu) sync(%d), repl(%d).

  ER_SEMISYNC_SYNC_HEADER_UPDATE_INFO was added in 8.0.4.

- Error number: MY-011159; Symbol: ER_SEMISYNC_FAILED_TO_INSERT_TRX_NODE; SQLSTATE: HY000

  Message: Semi-sync failed to insert tranx_node for binlog file: %s, position: %lu.

  ER_SEMISYNC_FAILED_TO_INSERT_TRX_NODE was added in 8.0.4.

- Error number: MY-011160; Symbol: ER_SEMISYNC_TRX_SKIPPED_AT_POS; SQLSTATE: HY000

  Message: %s: Transaction skipped at (%s, %lu).

  ER_SEMISYNC_TRX_SKIPPED_AT_POS was added in 8.0.4.

- Error number: MY-011161; Symbol: ER_SEMISYNC_MASTER_FAILED_ON_NET_FLUSH; SQLSTATE: HY000

  Message: Semi-sync master failed on net_flush() before waiting for slave reply.

  ER_SEMISYNC_MASTER_FAILED_ON_NET_FLUSH was added in 8.0.4.

- Error number: MY-011162; Symbol: ER_SEMISYNC_RECEIVED_ACK_IS_SMALLER; SQLSTATE: HY000

  Message: The received ack is smaller than m_greatest_ack.

  ER_SEMISYNC_RECEIVED_ACK_IS_SMALLER was added in 8.0.4.

- Error number: MY-011163; Symbol: ER_SEMISYNC_ADD_ACK_TO_SLOT; SQLSTATE: HY000

Message: Add the ack into slot %u.

ER_SEMISYNC_ADD_ACK_TO_SLOT was added in 8.0.4.

- Error number: MY-011164; Symbol: ER_SEMISYNC_UPDATE_EXISTING_SLAVE_ACK; SQLSTATE: HY000

  Message: Update an exsiting ack in slot %u.

  ER_SEMISYNC_UPDATE_EXISTING_SLAVE_ACK was added in 8.0.4.

- Error number: MY-011165; Symbol: ER_SEMISYNC_FAILED_TO_START_ACK_RECEIVER_THD; SQLSTATE: HY000

  Message: Failed to start semi-sync ACK receiver thread, could not create thread(errno:%d).

  ER_SEMISYNC_FAILED_TO_START_ACK_RECEIVER_THD was added in 8.0.4.

- Error number: MY-011166; Symbol: ER_SEMISYNC_STARTING_ACK_RECEIVER_THD; SQLSTATE: HY000

  Message: Starting ack receiver thread.

  ER_SEMISYNC_STARTING_ACK_RECEIVER_THD was added in 8.0.4.

- Error number: MY-011167; Symbol: ER_SEMISYNC_FAILED_TO_WAIT_ON_DUMP_SOCKET; SQLSTATE: HY000

  Message: Failed to wait on semi-sync dump sockets, error: errno=%d.

  ER_SEMISYNC_FAILED_TO_WAIT_ON_DUMP_SOCKET was added in 8.0.4.

- Error number: MY-011168; Symbol: ER_SEMISYNC_STOPPING_ACK_RECEIVER_THREAD; SQLSTATE: HY000

  Message: Stopping ack receiver thread.

  ER_SEMISYNC_STOPPING_ACK_RECEIVER_THREAD was added in 8.0.4.

- Error number: MY-011169; Symbol: ER_SEMISYNC_FAILED_REGISTER_SLAVE_TO_RECEIVER; SQLSTATE: HY000

  Message: Failed to register slave to semi-sync ACK receiver thread.

  ER_SEMISYNC_FAILED_REGISTER_SLAVE_TO_RECEIVER was added in 8.0.4.

- Error number: MY-011170; Symbol: ER_SEMISYNC_START_BINLOG_DUMP_TO_SLAVE; SQLSTATE: HY000

  Message: Start %s binlog_dump to slave (server_id: %d), pos(%s, %lu).

  ER_SEMISYNC_START_BINLOG_DUMP_TO_SLAVE was added in 8.0.4.

- Error number: MY-011171; Symbol: ER_SEMISYNC_STOP_BINLOG_DUMP_TO_SLAVE; SQLSTATE: HY000

  Message: Stop %s binlog_dump to slave (server_id: %d).

ER_SEMISYNC_STOP_BINLOG_DUMP_TO_SLAVE was added in 8.0.4.

- Error number: MY-011172; Symbol: ER_SEMISYNC_UNREGISTER_TRX_OBSERVER_FAILED; SQLSTATE: HY000

  Message: unregister_trans_observer failed.

  ER_SEMISYNC_UNREGISTER_TRX_OBSERVER_FAILED was added in 8.0.4.

- Error number: MY-011173; Symbol: ER_SEMISYNC_UNREGISTER_BINLOG_STORAGE_OBSERVER_FAILED; SQLSTATE: HY000

  Message: unregister_binlog_storage_observer failed.

  ER_SEMISYNC_UNREGISTER_BINLOG_STORAGE_OBSERVER_FAILED was added in 8.0.4.

- Error number: MY-011174; Symbol: ER_SEMISYNC_UNREGISTER_BINLOG_TRANSMIT_OBSERVER_FAILED; SQLSTATE: HY000

  Message: unregister_binlog_transmit_observer failed.

  ER_SEMISYNC_UNREGISTER_BINLOG_TRANSMIT_OBSERVER_FAILED was added in 8.0.4.

- Error number: MY-011175; Symbol: ER_SEMISYNC_UNREGISTERED_REPLICATOR; SQLSTATE: HY000

  Message: unregister_replicator OK.

  ER_SEMISYNC_UNREGISTERED_REPLICATOR was added in 8.0.4.

- Error number: MY-011176; Symbol: ER_SEMISYNC_SOCKET_FD_TOO_LARGE; SQLSTATE: HY000

  Message: Semisync slave socket fd is %u. select() cannot handle if the socket fd is bigger than %u (FD_SETSIZE).

  ER_SEMISYNC_SOCKET_FD_TOO_LARGE was added in 8.0.4.

- Error number: MY-011177; Symbol: ER_SEMISYNC_SLAVE_REPLY; SQLSTATE: HY000

  Message: %s: reply - %d.

  ER_SEMISYNC_SLAVE_REPLY was added in 8.0.4.

- Error number: MY-011178; Symbol: ER_SEMISYNC_MISSING_MAGIC_NO_FOR_SEMISYNC_PKT; SQLSTATE: HY000

  Message: Missing magic number for semi-sync packet, packet len: %lu.

  ER_SEMISYNC_MISSING_MAGIC_NO_FOR_SEMISYNC_PKT was added in 8.0.4.

- Error number: MY-011179; Symbol: ER_SEMISYNC_SLAVE_START; SQLSTATE: HY000

  Message: Slave I/O thread: Start %s replication to master '%s@%s:%d' in log '%s' at position %lu.

  ER_SEMISYNC_SLAVE_START was added in 8.0.4.

- Error number: MY-011180; Symbol: ER_SEMISYNC_SLAVE_REPLY_WITH_BINLOG_INFO; SQLSTATE: HY000

Message: %s: reply (%s, %lu).

ER_SEMISYNC_SLAVE_REPLY_WITH_BINLOG_INFO was added in 8.0.4.

- Error number: MY-011181; Symbol: ER_SEMISYNC_SLAVE_NET_FLUSH_REPLY_FAILED; SQLSTATE: HY000

  Message: Semi-sync slave net_flush() reply failed.

  ER_SEMISYNC_SLAVE_NET_FLUSH_REPLY_FAILED was added in 8.0.4.

- Error number: MY-011182; Symbol: ER_SEMISYNC_SLAVE_SEND_REPLY_FAILED; SQLSTATE: HY000

  Message: Semi-sync slave send reply failed: %s (%d).

  ER_SEMISYNC_SLAVE_SEND_REPLY_FAILED was added in 8.0.4.

- Error number: MY-011183; Symbol: ER_SEMISYNC_EXECUTION_FAILED_ON_MASTER; SQLSTATE: HY000

  Message: Execution failed on master: %s; error %d

  ER_SEMISYNC_EXECUTION_FAILED_ON_MASTER was added in 8.0.4.

- Error number: MY-011184; Symbol: ER_SEMISYNC_NOT_SUPPORTED_BY_MASTER; SQLSTATE: HY000

  Message: Master server does not support semi-sync, fallback to asynchronous replication

  ER_SEMISYNC_NOT_SUPPORTED_BY_MASTER was added in 8.0.4.

- Error number: MY-011185; Symbol: ER_SEMISYNC_SLAVE_SET_FAILED; SQLSTATE: HY000

  Message: Set 'rpl_semi_sync_slave=1' on master failed

  ER_SEMISYNC_SLAVE_SET_FAILED was added in 8.0.4.

- Error number: MY-011186; Symbol: ER_SEMISYNC_FAILED_TO_STOP_ACK_RECEIVER_THD; SQLSTATE: HY000

  Message: Failed to stop ack receiver thread on my_thread_join, errno(%d).

  ER_SEMISYNC_FAILED_TO_STOP_ACK_RECEIVER_THD was added in 8.0.4.

- Error number: MY-011187; Symbol: ER_FIREWALL_FAILED_TO_READ_FIREWALL_TABLES; SQLSTATE: HY000

  Message: Failed to read the firewall tables

  ER_FIREWALL_FAILED_TO_READ_FIREWALL_TABLES was added in 8.0.4.

- Error number: MY-011188; Symbol: ER_FIREWALL_FAILED_TO_REG_DYNAMIC_PRIVILEGES; SQLSTATE: HY000

  Message: Failed to register dynamic privileges

  ER_FIREWALL_FAILED_TO_REG_DYNAMIC_PRIVILEGES was added in 8.0.4.

- Error number: MY-011189; Symbol: ER_FIREWALL_RECORDING_STMT_WAS_TRUNCATED; SQLSTATE: HY000

  Message: Statement was truncated and not recorded: %s

  ER_FIREWALL_RECORDING_STMT_WAS_TRUNCATED was added in 8.0.4.

- Error number: MY-011190; Symbol: ER_FIREWALL_RECORDING_STMT_WITHOUT_TEXT; SQLSTATE: HY000

  Message: Statement with no text was not recorded

  ER_FIREWALL_RECORDING_STMT_WITHOUT_TEXT was added in 8.0.4.

- Error number: MY-011191; Symbol: ER_FIREWALL_SUSPICIOUS_STMT; SQLSTATE: HY000

  Message: SUSPICIOUS STATEMENT from '%s'. Reason: %s Statement: %s

  ER_FIREWALL_SUSPICIOUS_STMT was added in 8.0.4.

- Error number: MY-011192; Symbol: ER_FIREWALL_ACCESS_DENIED; SQLSTATE: HY000

  Message: ACCESS DENIED for '%s'. Reason: %s Statement: %s

  ER_FIREWALL_ACCESS_DENIED was added in 8.0.4.

- Error number: MY-011193; Symbol: ER_FIREWALL_SKIPPED_UNKNOWN_USER_MODE; SQLSTATE: HY000

  Message: Skipped unknown user mode '%s'

  ER_FIREWALL_SKIPPED_UNKNOWN_USER_MODE was added in 8.0.4.

- Error number: MY-011194; Symbol: ER_FIREWALL_RELOADING_CACHE; SQLSTATE: HY000

  Message: Reloading cache from disk

  ER_FIREWALL_RELOADING_CACHE was added in 8.0.4.

- Error number: MY-011195; Symbol: ER_FIREWALL_RESET_FOR_USER; SQLSTATE: HY000

  Message: FIREWALL RESET for '%s'

  ER_FIREWALL_RESET_FOR_USER was added in 8.0.4.

- Error number: MY-011196; Symbol: ER_FIREWALL_STATUS_FLUSHED; SQLSTATE: HY000

  Message: Counters are reset to zero

  ER_FIREWALL_STATUS_FLUSHED was added in 8.0.4.

- Error number: MY-011197; Symbol: ER_KEYRING_LOGGER_ERROR_MSG; SQLSTATE: HY000

  Message: %s

  ER_KEYRING_LOGGER_ERROR_MSG was added in 8.0.4.

- Error number: MY-011198; Symbol: ER_AUDIT_LOG_FILTER_IS_NOT_INSTALLED; SQLSTATE: HY000

Message: Audit Log plugin supports a filtering, which has not been installed yet. Audit Log plugin will run in the legacy mode, which will be disabled in the next release.

`ER_AUDIT_LOG_FILTER_IS_NOT_INSTALLED` was added in 8.0.4.

- Error number: `MY-011199`; Symbol: `ER_AUDIT_LOG_SWITCHING_TO_INCLUDE_LIST`; SQLSTATE: `HY000`

  Message: Previously exclude list is used, now we start using include list, exclude list is set to NULL.

  `ER_AUDIT_LOG_SWITCHING_TO_INCLUDE_LIST` was added in 8.0.4.

- Error number: `MY-011200`; Symbol: `ER_AUDIT_LOG_CANNOT_SET_LOG_POLICY_WITH_OTHER_POLICIES`; SQLSTATE: `HY000`

  Message: Cannot set audit_log_policy simultaneously with either audit_log_connection_policy or audit_log_statement_policy, setting audit_log_connection_policy and audit_log_statement_policy based on audit_log_policy.

  `ER_AUDIT_LOG_CANNOT_SET_LOG_POLICY_WITH_OTHER_POLICIES` was added in 8.0.4.

- Error number: `MY-011201`; Symbol: `ER_AUDIT_LOG_ONLY_INCLUDE_LIST_USED`; SQLSTATE: `HY000`

  Message: Both include and exclude lists provided, include list is preferred, exclude list is set to NULL.

  `ER_AUDIT_LOG_ONLY_INCLUDE_LIST_USED` was added in 8.0.4.

- Error number: `MY-011202`; Symbol: `ER_AUDIT_LOG_INDEX_MAP_CANNOT_ACCESS_DIR`; SQLSTATE: `HY000`

  Message: Could not access '%s' directory.

  `ER_AUDIT_LOG_INDEX_MAP_CANNOT_ACCESS_DIR` was added in 8.0.4.

- Error number: `MY-011203`; Symbol: `ER_AUDIT_LOG_WRITER_RENAME_FILE_FAILED`; SQLSTATE: `HY000`

  Message: Could not rename file from '%s' to '%s'.

  `ER_AUDIT_LOG_WRITER_RENAME_FILE_FAILED` was added in 8.0.4.

- Error number: `MY-011204`; Symbol: `ER_AUDIT_LOG_WRITER_DEST_FILE_ALREADY_EXISTS`; SQLSTATE: `HY000`

  Message: File '%s' should not exist. It may be incomplete. The server crashed.

  `ER_AUDIT_LOG_WRITER_DEST_FILE_ALREADY_EXISTS` was added in 8.0.4.

- Error number: `MY-011205`; Symbol: `ER_AUDIT_LOG_WRITER_RENAME_FILE_FAILED_REMOVE_FILE_MANUALLY`; SQLSTATE: `HY000`

  Message: Could not rename file from '%s' to '%s'. Remove the file manually.

  `ER_AUDIT_LOG_WRITER_RENAME_FILE_FAILED_REMOVE_FILE_MANUALLY` was added in 8.0.4.

- Error number: `MY-011206`; Symbol: `ER_AUDIT_LOG_WRITER_INCOMPLETE_FILE_RENAMED`; SQLSTATE: `HY000`

Message: Incomplete file renamed from '%s' to '%s'.

ER_AUDIT_LOG_WRITER_INCOMPLETE_FILE_RENAMED was added in 8.0.4.

- Error number: MY-011207; Symbol: ER_AUDIT_LOG_WRITER_FAILED_TO_WRITE_TO_FILE; SQLSTATE: HY000

  Message: Error writing file \'%s\' (errno: %d - %s).

  ER_AUDIT_LOG_WRITER_FAILED_TO_WRITE_TO_FILE was added in 8.0.4.

- Error number: MY-011208; Symbol: ER_AUDIT_LOG_EC_WRITER_FAILED_TO_INIT_ENCRYPTION; SQLSTATE: HY000

  Message: Could not initialize audit log file encryption.

  ER_AUDIT_LOG_EC_WRITER_FAILED_TO_INIT_ENCRYPTION was added in 8.0.4.

- Error number: MY-011209; Symbol: ER_AUDIT_LOG_EC_WRITER_FAILED_TO_INIT_COMPRESSION; SQLSTATE: HY000

  Message: Could not initialize audit log file compression.

  ER_AUDIT_LOG_EC_WRITER_FAILED_TO_INIT_COMPRESSION was added in 8.0.4.

- Error number: MY-011210; Symbol: ER_AUDIT_LOG_EC_WRITER_FAILED_TO_CREATE_FILE; SQLSTATE: HY000

  Message: Could not create '%s' file for audit logging.

  ER_AUDIT_LOG_EC_WRITER_FAILED_TO_CREATE_FILE was added in 8.0.4.

- Error number: MY-011211; Symbol: ER_AUDIT_LOG_RENAME_LOG_FILE_BEFORE_FLUSH; SQLSTATE: HY000

  Message: Audit log file (%s) must be manually renamed before audit_log_flush is set to true.

  ER_AUDIT_LOG_RENAME_LOG_FILE_BEFORE_FLUSH was added in 8.0.4.

- Error number: MY-011212; Symbol: ER_AUDIT_LOG_FILTER_RESULT_MSG; SQLSTATE: HY000

  Message: %s

  ER_AUDIT_LOG_FILTER_RESULT_MSG was added in 8.0.4.

- Error number: MY-011213; Symbol: ER_AUDIT_LOG_JSON_READER_FAILED_TO_PARSE; SQLSTATE: HY000

  Message: Error parsing JSON event. Event not accessible.

  ER_AUDIT_LOG_JSON_READER_FAILED_TO_PARSE was added in 8.0.4.

- Error number: MY-011214; Symbol: ER_AUDIT_LOG_JSON_READER_BUF_TOO_SMALL; SQLSTATE: HY000

  Message: Buffer is too small to hold JSON event. Number of events skipped: %zu.

  ER_AUDIT_LOG_JSON_READER_BUF_TOO_SMALL was added in 8.0.4.

- Error number: MY-011215; Symbol: ER_AUDIT_LOG_JSON_READER_FAILED_TO_OPEN_FILE; SQLSTATE: HY000

  Message: Could not open JSON file for reading. Reading next file if exists.

  ER_AUDIT_LOG_JSON_READER_FAILED_TO_OPEN_FILE was added in 8.0.4.

- Error number: MY-011216; Symbol: ER_AUDIT_LOG_JSON_READER_FILE_PARSING_ERROR; SQLSTATE: HY000

  Message: JSON file parsing error. Reading next file if exists

  ER_AUDIT_LOG_JSON_READER_FILE_PARSING_ERROR was added in 8.0.4.

- Error number: MY-011217; Symbol: ER_AUDIT_LOG_FILTER_INVALID_COLUMN_COUNT; SQLSTATE: HY000

  Message: Invalid column count in the '%s.%s' table.

  ER_AUDIT_LOG_FILTER_INVALID_COLUMN_COUNT was added in 8.0.4.

- Error number: MY-011218; Symbol: ER_AUDIT_LOG_FILTER_INVALID_COLUMN_DEFINITION; SQLSTATE: HY000

  Message: Invalid column definition of the '%s.%s' table.

  ER_AUDIT_LOG_FILTER_INVALID_COLUMN_DEFINITION was added in 8.0.4.

- Error number: MY-011219; Symbol: ER_AUDIT_LOG_FILTER_FAILED_TO_STORE_TABLE_FLDS; SQLSTATE: HY000

  Message: Could not store field of the %s table.

  ER_AUDIT_LOG_FILTER_FAILED_TO_STORE_TABLE_FLDS was added in 8.0.4.

- Error number: MY-011220; Symbol: ER_AUDIT_LOG_FILTER_FAILED_TO_UPDATE_TABLE; SQLSTATE: HY000

  Message: Could not update %s table.

  ER_AUDIT_LOG_FILTER_FAILED_TO_UPDATE_TABLE was added in 8.0.4.

- Error number: MY-011221; Symbol: ER_AUDIT_LOG_FILTER_FAILED_TO_INSERT_INTO_TABLE; SQLSTATE: HY000

  Message: Could not insert into %s table.

  ER_AUDIT_LOG_FILTER_FAILED_TO_INSERT_INTO_TABLE was added in 8.0.4.

- Error number: MY-011222; Symbol: ER_AUDIT_LOG_FILTER_FAILED_TO_DELETE_FROM_TABLE; SQLSTATE: HY000

  Message: Could not delete from %s table.

  ER_AUDIT_LOG_FILTER_FAILED_TO_DELETE_FROM_TABLE was added in 8.0.4.

- Error number: MY-011223; Symbol: ER_AUDIT_LOG_FILTER_FAILED_TO_INIT_TABLE_FOR_READ; SQLSTATE: HY000

Message: Could not initialize %s table for reading.

ER_AUDIT_LOG_FILTER_FAILED_TO_INIT_TABLE_FOR_READ was added in 8.0.4.

- Error number: MY-011224; Symbol: ER_AUDIT_LOG_FILTER_FAILED_TO_READ_TABLE; SQLSTATE: HY000

  Message: Could not read %s table.

  ER_AUDIT_LOG_FILTER_FAILED_TO_READ_TABLE was added in 8.0.4.

- Error number: MY-011225; Symbol: ER_AUDIT_LOG_FILTER_FAILED_TO_CLOSE_TABLE_AFTER_READING; SQLSTATE: HY000

  Message: Could not close %s table reading.

  ER_AUDIT_LOG_FILTER_FAILED_TO_CLOSE_TABLE_AFTER_READING was added in 8.0.4.

- Error number: MY-011226; Symbol: ER_AUDIT_LOG_FILTER_USER_AND_HOST_CANNOT_BE_EMPTY; SQLSTATE: HY000

  Message: Both user and host columns of %s table cannot be empty.

  ER_AUDIT_LOG_FILTER_USER_AND_HOST_CANNOT_BE_EMPTY was added in 8.0.4.

- Error number: MY-011227; Symbol: ER_AUDIT_LOG_FILTER_FLD_FILTERNAME_CANNOT_BE_EMPTY; SQLSTATE: HY000

  Message: Filtername column of %s table cannot be empty.

  ER_AUDIT_LOG_FILTER_FLD_FILTERNAME_CANNOT_BE_EMPTY was added in 8.0.4.

- Error number: MY-011228; Symbol: ER_VALIDATE_PWD_DICT_FILE_NOT_SPECIFIED; SQLSTATE: HY000

  Message: Dictionary file not specified

  ER_VALIDATE_PWD_DICT_FILE_NOT_SPECIFIED was added in 8.0.4.

- Error number: MY-011229; Symbol: ER_VALIDATE_PWD_DICT_FILE_NOT_LOADED; SQLSTATE: HY000

  Message: Dictionary file not loaded

  ER_VALIDATE_PWD_DICT_FILE_NOT_LOADED was added in 8.0.4.

- Error number: MY-011230; Symbol: ER_VALIDATE_PWD_DICT_FILE_TOO_BIG; SQLSTATE: HY000

  Message: Dictionary file size exceeded MAX_DICTIONARY_FILE_LENGTH, not loaded

  ER_VALIDATE_PWD_DICT_FILE_TOO_BIG was added in 8.0.4.

- Error number: MY-011231; Symbol: ER_VALIDATE_PWD_FAILED_TO_READ_DICT_FILE; SQLSTATE: HY000

  Message: Exception while reading the dictionary file

  ER_VALIDATE_PWD_FAILED_TO_READ_DICT_FILE was added in 8.0.4.

- Error number: MY-011232; Symbol: ER_VALIDATE_PWD_FAILED_TO_GET_FLD_FROM_SECURITY_CTX; SQLSTATE: HY000

  Message: Can't retrieve the %s from the security context

  ER_VALIDATE_PWD_FAILED_TO_GET_FLD_FROM_SECURITY_CTX was added in 8.0.4.

- Error number: MY-011233; Symbol: ER_VALIDATE_PWD_FAILED_TO_GET_SECURITY_CTX; SQLSTATE: HY000

  Message: Can't retrieve the security context

  ER_VALIDATE_PWD_FAILED_TO_GET_SECURITY_CTX was added in 8.0.4.

- Error number: MY-011234; Symbol: ER_VALIDATE_PWD_LENGTH_CHANGED; SQLSTATE: HY000

  Message: Effective value of validate_password_length is changed. New value is %d

  ER_VALIDATE_PWD_LENGTH_CHANGED was added in 8.0.4.

- Error number: MY-011235; Symbol: ER_REWRITER_QUERY_ERROR_MSG; SQLSTATE: HY000

  Message: %s

  ER_REWRITER_QUERY_ERROR_MSG was added in 8.0.4.

- Error number: MY-011236; Symbol: ER_REWRITER_QUERY_FAILED; SQLSTATE: HY000

  Message: Rewritten query failed to parse:%s

  ER_REWRITER_QUERY_FAILED was added in 8.0.4.

- Error number: MY-011237; Symbol: ER_XPLUGIN_STARTUP_FAILED; SQLSTATE: HY000

  Message: Startup failed with error "%s"

  ER_XPLUGIN_STARTUP_FAILED was added in 8.0.4.

- Error number: MY-011238; Symbol: ER_XPLUGIN_SERVER_EXITING; SQLSTATE: HY000

  Message: Exiting

  ER_XPLUGIN_SERVER_EXITING was added in 8.0.4, removed after 8.0.12.

- Error number: MY-011239; Symbol: ER_XPLUGIN_SERVER_EXITED; SQLSTATE: HY000

  Message: Exit done

  ER_XPLUGIN_SERVER_EXITED was added in 8.0.4, removed after 8.0.12.

- Error number: MY-011240; Symbol: ER_XPLUGIN_USING_SSL_CONF_FROM_SERVER; SQLSTATE: HY000

  Message: Using SSL configuration from MySQL Server

  ER_XPLUGIN_USING_SSL_CONF_FROM_SERVER was added in 8.0.4.

- Error number: MY-011241; Symbol: ER_XPLUGIN_USING_SSL_CONF_FROM_MYSQLX; SQLSTATE: HY000

Message: Using SSL configuration from Mysqlx Plugin

ER_XPLUGIN_USING_SSL_CONF_FROM_MYSQLX was added in 8.0.4.

- Error number: MY-011242; Symbol: ER_XPLUGIN_FAILED_TO_USE_SSL_CONF; SQLSTATE: HY000

  Message: Neither MySQL Server nor Mysqlx Plugin has valid SSL configuration

  ER_XPLUGIN_FAILED_TO_USE_SSL_CONF was added in 8.0.4.

- Error number: MY-011243; Symbol: ER_XPLUGIN_USING_SSL_FOR_TLS_CONNECTION; SQLSTATE: HY000

  Message: Using %s for TLS connections

  ER_XPLUGIN_USING_SSL_FOR_TLS_CONNECTION was added in 8.0.4.

- Error number: MY-011244; Symbol: ER_XPLUGIN_REFERENCE_TO_SECURE_CONN_WITH_XPLUGIN; SQLSTATE: HY000

  Message: For more information, please see the Using Secure Connections with X Plugin section in the MySQL documentation

  ER_XPLUGIN_REFERENCE_TO_SECURE_CONN_WITH_XPLUGIN was added in 8.0.4.

- Error number: MY-011245; Symbol: ER_XPLUGIN_ERROR_MSG; SQLSTATE: HY000

  Message: %s

  ER_XPLUGIN_ERROR_MSG was added in 8.0.4.

- Error number: MY-011246; Symbol: ER_SHA_PWD_FAILED_TO_PARSE_AUTH_STRING; SQLSTATE: HY000

  Message: Failed to parse stored authentication string for %s. Please check if mysql.user table not corrupted

  ER_SHA_PWD_FAILED_TO_PARSE_AUTH_STRING was added in 8.0.4.

- Error number: MY-011247; Symbol: ER_SHA_PWD_FAILED_TO_GENERATE_MULTI_ROUND_HASH; SQLSTATE: HY000

  Message: Error in generating multi-round hash for %s. Plugin can not perform authentication without it. This may be a transient problem

  ER_SHA_PWD_FAILED_TO_GENERATE_MULTI_ROUND_HASH was added in 8.0.4.

- Error number: MY-011248; Symbol: ER_SHA_PWD_AUTH_REQUIRES_RSA_OR_SSL; SQLSTATE: HY000

  Message: Authentication requires either RSA keys or SSL encryption

  ER_SHA_PWD_AUTH_REQUIRES_RSA_OR_SSL was added in 8.0.4.

- Error number: MY-011249; Symbol: ER_SHA_PWD_RSA_KEY_TOO_LONG; SQLSTATE: HY000

  Message: RSA key cipher length of %u is too long. Max value is %u

ER_SHA_PWD_RSA_KEY_TOO_LONG was added in 8.0.4.

- Error number: MY-011250; Symbol: ER_PLUGIN_COMMON_FAILED_TO_OPEN_FILTER_TABLES; SQLSTATE: HY000

  Message: Failed to open the %s filter tables

  ER_PLUGIN_COMMON_FAILED_TO_OPEN_FILTER_TABLES was added in 8.0.4.

- Error number: MY-011251; Symbol: ER_PLUGIN_COMMON_FAILED_TO_OPEN_TABLE; SQLSTATE: HY000

  Message: Failed to open '%s.%s' %s table

  ER_PLUGIN_COMMON_FAILED_TO_OPEN_TABLE was added in 8.0.4.

- Error number: MY-011252; Symbol: ER_AUTH_LDAP_ERROR_LOGGER_ERROR_MSG; SQLSTATE: HY000

  Message: %s

  ER_AUTH_LDAP_ERROR_LOGGER_ERROR_MSG was added in 8.0.4.

- Error number: MY-011253; Symbol: ER_CONN_CONTROL_ERROR_MSG; SQLSTATE: HY000

  Message: %s

  ER_CONN_CONTROL_ERROR_MSG was added in 8.0.4.

- Error number: MY-011254; Symbol: ER_GRP_RPL_ERROR_MSG; SQLSTATE: HY000

  Message: %s

  ER_GRP_RPL_ERROR_MSG was added in 8.0.4.

- Error number: MY-011255; Symbol: ER_SHA_PWD_SALT_FOR_USER_CORRUPT; SQLSTATE: HY000

  Message: Password salt for user '%s' is corrupt

  ER_SHA_PWD_SALT_FOR_USER_CORRUPT was added in 8.0.4.

- Error number: MY-011256; Symbol: ER_SYS_VAR_COMPONENT_OOM; SQLSTATE: HY000

  Message: Out of memory for component system variable '%s'.

  ER_SYS_VAR_COMPONENT_OOM was added in 8.0.4.

- Error number: MY-011257; Symbol: ER_SYS_VAR_COMPONENT_VARIABLE_SET_READ_ONLY; SQLSTATE: HY000

  Message: variable %s of component %s was forced to be read-only: string variable without update_func and PLUGIN_VAR_MEMALLOC flag.

  ER_SYS_VAR_COMPONENT_VARIABLE_SET_READ_ONLY was added in 8.0.4.

- Error number: MY-011258; Symbol: ER_SYS_VAR_COMPONENT_UNKNOWN_VARIABLE_TYPE; SQLSTATE: HY000

  Message: Unknown variable type code 0x%x in component '%s'.

ER_SYS_VAR_COMPONENT_UNKNOWN_VARIABLE_TYPE was added in 8.0.4.

- Error number: MY-011259; Symbol: ER_SYS_VAR_COMPONENT_FAILED_TO_PARSE_VARIABLE_OPTIONS; SQLSTATE: HY000

  Message: Parsing options for variable '%s' failed.

  ER_SYS_VAR_COMPONENT_FAILED_TO_PARSE_VARIABLE_OPTIONS was added in 8.0.4.

- Error number: MY-011260; Symbol: ER_SYS_VAR_COMPONENT_FAILED_TO_MAKE_VARIABLE_PERSISTENT; SQLSTATE: HY000

  Message: Setting persistent options for component variable '%s' failed.

  ER_SYS_VAR_COMPONENT_FAILED_TO_MAKE_VARIABLE_PERSISTENT was added in 8.0.4.

- Error number: MY-011261; Symbol: ER_COMPONENT_FILTER_CONFUSED; SQLSTATE: HY000

  Message: The log-filter component "%s" got confused at "%s" (state: %s) ...

  ER_COMPONENT_FILTER_CONFUSED was added in 8.0.4.

- Error number: MY-011262; Symbol: ER_STOP_SLAVE_IO_THREAD_DISK_SPACE; SQLSTATE: HY000

  Message: Waiting until I/O thread for channel '%s' finish writing to disk before stopping. Free some disk space or use 'KILL' to abort I/O thread operation. Notice that aborting the I/O thread while rotating the relay log might corrupt the relay logs, requiring a server restart to fix it.

  ER_STOP_SLAVE_IO_THREAD_DISK_SPACE was added in 8.0.2.

- Error number: MY-011263; Symbol: ER_LOG_FILE_CANNOT_OPEN; SQLSTATE: HY000

  Message: Could not use %s for logging (error %d - %s). Turning logging off for the server process. To turn it on again: fix the cause, then%s restart the MySQL server.

  ER_LOG_FILE_CANNOT_OPEN was added in 8.0.2.

- Error number: MY-011268; Symbol: ER_PERSIST_OPTION_STATUS; SQLSTATE: HY000

  Message: Configuring persisted options failed: "%s".

  ER_PERSIST_OPTION_STATUS was added in 8.0.11.

- Error number: MY-011269; Symbol: ER_NOT_IMPLEMENTED_GET_TABLESPACE_STATISTICS; SQLSTATE: HY000

  Message: The storage engine '%s' does not provide dynamic table statistics

  ER_NOT_IMPLEMENTED_GET_TABLESPACE_STATISTICS was added in 8.0.11.

- Error number: MY-011272; Symbol: ER_SSL_FIPS_MODE_ERROR; SQLSTATE: HY000

  Message: SSL fips mode error: %s

  ER_SSL_FIPS_MODE_ERROR was added in 8.0.11.

- Error number: MY-011273; Symbol: ER_CONN_INIT_CONNECT_IGNORED; SQLSTATE: HY000

Message: init_connect variable is ignored for user: %s host: %s due to expired password.

ER_CONN_INIT_CONNECT_IGNORED was added in 8.0.11.

- Error number: MY-011275; Symbol: ER_REWRITER_OOM; SQLSTATE: HY000

  Message: Out of memory.

  ER_REWRITER_OOM was added in 8.0.11.

- Error number: MY-011276; Symbol: ER_REWRITER_TABLE_MALFORMED_ERROR; SQLSTATE: HY000

  Message: Wrong column count or names when loading rules.

  ER_REWRITER_TABLE_MALFORMED_ERROR was added in 8.0.11.

- Error number: MY-011277; Symbol: ER_REWRITER_LOAD_FAILED; SQLSTATE: HY000

  Message: Some rules failed to load.

  ER_REWRITER_LOAD_FAILED was added in 8.0.11.

- Error number: MY-011278; Symbol: ER_REWRITER_READ_FAILED; SQLSTATE: HY000

  Message: Got error from storage engine while refreshing rewrite rules.

  ER_REWRITER_READ_FAILED was added in 8.0.11.

- Error number: MY-011279; Symbol: ER_CONN_CONTROL_EVENT_COORDINATOR_INIT_FAILED; SQLSTATE: HY000

  Message: Failed to initialize Connection_event_coordinator

  ER_CONN_CONTROL_EVENT_COORDINATOR_INIT_FAILED was added in 8.0.11.

- Error number: MY-011280; Symbol: ER_CONN_CONTROL_STAT_CONN_DELAY_TRIGGERED_UPDATE_FAILED; SQLSTATE: HY000

  Message: Failed to update connection delay triggered stats

  ER_CONN_CONTROL_STAT_CONN_DELAY_TRIGGERED_UPDATE_FAILED was added in 8.0.11.

- Error number: MY-011281; Symbol: ER_CONN_CONTROL_STAT_CONN_DELAY_TRIGGERED_RESET_FAILED; SQLSTATE: HY000

  Message: Failed to reset connection delay triggered stats

  ER_CONN_CONTROL_STAT_CONN_DELAY_TRIGGERED_RESET_FAILED was added in 8.0.11.

- Error number: MY-011282; Symbol: ER_CONN_CONTROL_INVALID_CONN_DELAY_TYPE; SQLSTATE: HY000

  Message: Unexpected option type for connection delay.

  ER_CONN_CONTROL_INVALID_CONN_DELAY_TYPE was added in 8.0.11.

- Error number: MY-011283; Symbol: ER_CONN_CONTROL_DELAY_ACTION_INIT_FAILED; SQLSTATE: HY000

Message: Failed to initialize Connection_delay_action

ER_CONN_CONTROL_DELAY_ACTION_INIT_FAILED was added in 8.0.11.

- Error number: MY-011284; Symbol: ER_CONN_CONTROL_FAILED_TO_SET_CONN_DELAY; SQLSTATE: HY000

  Message: Could not set %s delay for connection delay.

  ER_CONN_CONTROL_FAILED_TO_SET_CONN_DELAY was added in 8.0.11.

- Error number: MY-011285; Symbol: ER_CONN_CONTROL_FAILED_TO_UPDATE_CONN_DELAY_HASH; SQLSTATE: HY000

  Message: Failed to update connection delay hash for account : %s

  ER_CONN_CONTROL_FAILED_TO_UPDATE_CONN_DELAY_HASH was added in 8.0.11.

- Error number: MY-011286; Symbol: ER_XPLUGIN_FORCE_STOP_CLIENT; SQLSTATE: HY000

  Message: %s: Force stopping client because exception occurred: %s

  ER_XPLUGIN_FORCE_STOP_CLIENT was added in 8.0.11.

- Error number: MY-011287; Symbol: ER_XPLUGIN_MAX_AUTH_ATTEMPTS_REACHED; SQLSTATE: HY000

  Message: %s.%u: Maximum number of authentication attempts reached, login failed.

  ER_XPLUGIN_MAX_AUTH_ATTEMPTS_REACHED was added in 8.0.11.

- Error number: MY-011288; Symbol: ER_XPLUGIN_BUFFER_PAGE_ALLOC_FAILED; SQLSTATE: HY000

  Message: Error allocating Buffer_page: %s

  ER_XPLUGIN_BUFFER_PAGE_ALLOC_FAILED was added in 8.0.11.

- Error number: MY-011289; Symbol: ER_XPLUGIN_DETECTED_HANGING_CLIENTS; SQLSTATE: HY000

  Message: Detected %u hanging client(s)

  ER_XPLUGIN_DETECTED_HANGING_CLIENTS was added in 8.0.11.

- Error number: MY-011290; Symbol: ER_XPLUGIN_FAILED_TO_ACCEPT_CLIENT; SQLSTATE: HY000

  Message: Error accepting client

  ER_XPLUGIN_FAILED_TO_ACCEPT_CLIENT was added in 8.0.11.

- Error number: MY-011291; Symbol: ER_XPLUGIN_FAILED_TO_SCHEDULE_CLIENT; SQLSTATE: HY000

  Message: Internal error scheduling client for execution

  ER_XPLUGIN_FAILED_TO_SCHEDULE_CLIENT was added in 8.0.11.

- Error number: MY-011292; Symbol: ER_XPLUGIN_FAILED_TO_PREPARE_IO_INTERFACES; SQLSTATE: HY000

  Message: Preparation of I/O interfaces failed, X Protocol won't be accessible

  ER_XPLUGIN_FAILED_TO_PREPARE_IO_INTERFACES was added in 8.0.11.

- Error number: MY-011293; Symbol: ER_XPLUGIN_SRV_SESSION_INIT_THREAD_FAILED; SQLSTATE: HY000

  Message: srv_session_init_thread returned error

  ER_XPLUGIN_SRV_SESSION_INIT_THREAD_FAILED was added in 8.0.11.

- Error number: MY-011294; Symbol: ER_XPLUGIN_UNABLE_TO_USE_USER_SESSION_ACCOUNT; SQLSTATE: HY000

  Message: Unable to use user mysql.session account when connecting the server for internal plugin requests.

  ER_XPLUGIN_UNABLE_TO_USE_USER_SESSION_ACCOUNT was added in 8.0.11.

- Error number: MY-011295; Symbol: ER_XPLUGIN_REFERENCE_TO_USER_ACCOUNT_DOC_SECTION; SQLSTATE: HY000

  Message: For more information, please see the X Plugin User Account section in the MySQL documentation

  ER_XPLUGIN_REFERENCE_TO_USER_ACCOUNT_DOC_SECTION was added in 8.0.11.

- Error number: MY-011296; Symbol: ER_XPLUGIN_UNEXPECTED_EXCEPTION_DISPATCHING_CMD; SQLSTATE: HY000

  Message: %s: Unexpected exception dispatching command: %s

  ER_XPLUGIN_UNEXPECTED_EXCEPTION_DISPATCHING_CMD was added in 8.0.11.

- Error number: MY-011297; Symbol: ER_XPLUGIN_EXCEPTION_IN_TASK_SCHEDULER; SQLSTATE: HY000

  Message: Exception in post: %s

  ER_XPLUGIN_EXCEPTION_IN_TASK_SCHEDULER was added in 8.0.11.

- Error number: MY-011298; Symbol: ER_XPLUGIN_TASK_SCHEDULING_FAILED; SQLSTATE: HY000

  Message: Internal error scheduling task

  ER_XPLUGIN_TASK_SCHEDULING_FAILED was added in 8.0.11.

- Error number: MY-011299; Symbol: ER_XPLUGIN_EXCEPTION_IN_EVENT_LOOP; SQLSTATE: HY000

  Message: Exception in event loop: "%s": %s

  ER_XPLUGIN_EXCEPTION_IN_EVENT_LOOP was added in 8.0.11.

- Error number: MY-011300; Symbol: ER_XPLUGIN_LISTENER_SETUP_FAILED; SQLSTATE: HY000

Message: Setup of %s failed, %s

ER_XPLUGIN_LISTENER_SETUP_FAILED was added in 8.0.11.

- Error number: MY-011301; Symbol: ER_XPLUING_NET_STARTUP_FAILED; SQLSTATE: HY000

  Message: %s

  ER_XPLUING_NET_STARTUP_FAILED was added in 8.0.11.

- Error number: MY-011302; Symbol: ER_XPLUGIN_FAILED_AT_SSL_CONF; SQLSTATE: HY000

  Message: Failed at SSL configuration: "%s"

  ER_XPLUGIN_FAILED_AT_SSL_CONF was added in 8.0.11.

- Error number: MY-011303; Symbol: ER_XPLUGIN_CLIENT_SSL_HANDSHAKE_FAILED; SQLSTATE: HY000

  Message: Error during SSL handshake for client connection (%i)

  ER_XPLUGIN_CLIENT_SSL_HANDSHAKE_FAILED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011304; Symbol: ER_XPLUGIN_SSL_HANDSHAKE_WITH_SERVER_FAILED; SQLSTATE: HY000

  Message: %s: Error during SSL handshake

  ER_XPLUGIN_SSL_HANDSHAKE_WITH_SERVER_FAILED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011305; Symbol: ER_XPLUGIN_FAILED_TO_CREATE_SESSION_FOR_CONN; SQLSTATE: HY000

  Message: %s: Error creating session for connection from %s

  ER_XPLUGIN_FAILED_TO_CREATE_SESSION_FOR_CONN was added in 8.0.11.

- Error number: MY-011306; Symbol: ER_XPLUGIN_FAILED_TO_INITIALIZE_SESSION; SQLSTATE: HY000

  Message: %s: Error initializing session for connection: %s

  ER_XPLUGIN_FAILED_TO_INITIALIZE_SESSION was added in 8.0.11.

- Error number: MY-011307; Symbol: ER_XPLUGIN_MESSAGE_TOO_LONG; SQLSTATE: HY000

  Message: %s: Message of size %u received, exceeding the limit of %i

  ER_XPLUGIN_MESSAGE_TOO_LONG was added in 8.0.11.

- Error number: MY-011308; Symbol: ER_XPLUGIN_UNINITIALIZED_MESSAGE; SQLSTATE: HY000

  Message: Message is not properly initialized: %s

  ER_XPLUGIN_UNINITIALIZED_MESSAGE was added in 8.0.11.

- Error number: MY-011309; Symbol: ER_XPLUGIN_FAILED_TO_SET_MIN_NUMBER_OF_WORKERS; SQLSTATE: HY000

Message: Unable to set minimal number of workers to %u; actual value is %i

ER_XPLUGIN_FAILED_TO_SET_MIN_NUMBER_OF_WORKERS was added in 8.0.11.

- Error number: MY-011310; Symbol: ER_XPLUGIN_UNABLE_TO_ACCEPT_CONNECTION; SQLSTATE: HY000

  Message: Unable to accept connection, disconnecting client

  ER_XPLUGIN_UNABLE_TO_ACCEPT_CONNECTION was added in 8.0.11.

- Error number: MY-011311; Symbol: ER_XPLUGIN_ALL_IO_INTERFACES_DISABLED; SQLSTATE: HY000

  Message: All I/O interfaces are disabled, X Protocol won't be accessible

  ER_XPLUGIN_ALL_IO_INTERFACES_DISABLED was added in 8.0.11.

- Error number: MY-011312; Symbol: ER_XPLUGIN_INVALID_MSG_DURING_CLIENT_INIT; SQLSTATE: HY000

  Message: %s: Invalid message %i received during client initialization

  ER_XPLUGIN_INVALID_MSG_DURING_CLIENT_INIT was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011313; Symbol: ER_XPLUGIN_CLOSING_CLIENTS_ON_SHUTDOWN; SQLSTATE: HY000

  Message: %s: closing client because of shutdown (state: %i)

  ER_XPLUGIN_CLOSING_CLIENTS_ON_SHUTDOWN was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011314; Symbol: ER_XPLUGIN_ERROR_READING_SOCKET; SQLSTATE: HY000

  Message: %s: Error reading from socket %s (%i)

  ER_XPLUGIN_ERROR_READING_SOCKET was added in 8.0.11.

- Error number: MY-011315; Symbol: ER_XPLUGIN_PEER_DISCONNECTED_WHILE_READING_MSG_BODY; SQLSTATE: HY000

  Message: %s: peer disconnected while reading message body

  ER_XPLUGIN_PEER_DISCONNECTED_WHILE_READING_MSG_BODY was added in 8.0.11.

- Error number: MY-011316; Symbol: ER_XPLUGIN_READ_FAILED_CLOSING_CONNECTION; SQLSTATE: HY000

  Message: client_id:%s - %s while reading from socket, closing connection

  ER_XPLUGIN_READ_FAILED_CLOSING_CONNECTION was added in 8.0.11.

- Error number: MY-011317; Symbol: ER_XPLUGIN_INVALID_AUTH_METHOD; SQLSTATE: HY000

  Message: %s.%u: Invalid authentication method %s

  ER_XPLUGIN_INVALID_AUTH_METHOD was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011318; Symbol: ER_XPLUGIN_UNEXPECTED_MSG_DURING_AUTHENTICATION; SQLSTATE: HY000

  Message: %s: Unexpected message of type %i received during authentication

  ER_XPLUGIN_UNEXPECTED_MSG_DURING_AUTHENTICATION was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011319; Symbol: ER_XPLUGIN_ERROR_WRITING_TO_CLIENT; SQLSTATE: HY000

  Message: Error writing to client: %s (%i)

  ER_XPLUGIN_ERROR_WRITING_TO_CLIENT was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011320; Symbol: ER_XPLUGIN_SCHEDULER_STARTED; SQLSTATE: HY000

  Message: Scheduler "%s" started.

  ER_XPLUGIN_SCHEDULER_STARTED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011321; Symbol: ER_XPLUGIN_SCHEDULER_STOPPED; SQLSTATE: HY000

  Message: Scheduler "%s" stopped.

  ER_XPLUGIN_SCHEDULER_STOPPED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011322; Symbol: ER_XPLUGIN_LISTENER_SYS_VARIABLE_ERROR; SQLSTATE: HY000

  Message: Please see the MySQL documentation for '%s' system variables to fix the error

  ER_XPLUGIN_LISTENER_SYS_VARIABLE_ERROR was added in 8.0.11.

- Error number: MY-011323; Symbol: ER_XPLUGIN_LISTENER_STATUS_MSG; SQLSTATE: HY000

  Message: X Plugin ready for connections. %s

  ER_XPLUGIN_LISTENER_STATUS_MSG was added in 8.0.11.

- Error number: MY-011324; Symbol: ER_XPLUGIN_RETRYING_BIND_ON_PORT; SQLSTATE: HY000

  Message: Retrying `bind()` on TCP/IP port %i

  ER_XPLUGIN_RETRYING_BIND_ON_PORT was added in 8.0.11.

- Error number: MY-011325; Symbol: ER_XPLUGIN_SHUTDOWN_TRIGGERED; SQLSTATE: HY000

  Message: Shutdown triggered by mysqld abort flag

  ER_XPLUGIN_SHUTDOWN_TRIGGERED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011326; Symbol: ER_XPLUGIN_USER_ACCOUNT_WITH_ALL_PERMISSIONS; SQLSTATE: HY000

  Message: Using %s account for authentication which has all required permissions

  ER_XPLUGIN_USER_ACCOUNT_WITH_ALL_PERMISSIONS was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011327; Symbol: ER_XPLUGIN_EXISTING_USER_ACCOUNT_WITH_INCOMPLETE_GRANTS; SQLSTATE: HY000

  Message: Using existing %s account for authentication. Incomplete grants will be fixed

  ER_XPLUGIN_EXISTING_USER_ACCOUNT_WITH_INCOMPLETE_GRANTS was added in 8.0.11.

- Error number: MY-011328; Symbol: ER_XPLUGIN_SERVER_STARTS_HANDLING_CONNECTIONS; SQLSTATE: HY000

  Message: Server starts handling incoming connections

  ER_XPLUGIN_SERVER_STARTS_HANDLING_CONNECTIONS was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011329; Symbol: ER_XPLUGIN_SERVER_STOPPED_HANDLING_CONNECTIONS; SQLSTATE: HY000

  Message: Stopped handling incoming connections

  ER_XPLUGIN_SERVER_STOPPED_HANDLING_CONNECTIONS was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011330; Symbol: ER_XPLUGIN_FAILED_TO_INTERRUPT_SESSION; SQLSTATE: HY000

  Message: %s: Could not interrupt client session

  ER_XPLUGIN_FAILED_TO_INTERRUPT_SESSION was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011331; Symbol: ER_XPLUGIN_CLIENT_RELEASE_TRIGGERED; SQLSTATE: HY000

  Message: %s: release triggered by timeout in state:%i

  ER_XPLUGIN_CLIENT_RELEASE_TRIGGERED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011332; Symbol: ER_XPLUGIN_IPv6_AVAILABLE; SQLSTATE: HY000

  Message: IPv6 is available

  ER_XPLUGIN_IPv6_AVAILABLE was added in 8.0.11.

- Error number: MY-011333; Symbol: ER_XPLUGIN_UNIX_SOCKET_NOT_CONFIGURED; SQLSTATE: HY000

  Message: UNIX socket not configured

  ER_XPLUGIN_UNIX_SOCKET_NOT_CONFIGURED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011334; Symbol: ER_XPLUGIN_CLIENT_KILL_MSG; SQLSTATE: HY000

  Message: Kill client: %i %s

  ER_XPLUGIN_CLIENT_KILL_MSG was added in 8.0.11.

- Error number: MY-011335; Symbol: ER_XPLUGIN_FAILED_TO_GET_SECURITY_CTX; SQLSTATE: HY000

  Message: Could not get security context for session

ER_XPLUGIN_FAILED_TO_GET_SECURITY_CTX was added in 8.0.11.

- Error number: MY-011336; Symbol: ER_XPLUGIN_FAILED_TO_SWITCH_SECURITY_CTX_TO_ROOT; SQLSTATE: HY000

  Message: Unable to switch security context to root

  ER_XPLUGIN_FAILED_TO_SWITCH_SECURITY_CTX_TO_ROOT was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011337; Symbol: ER_XPLUGIN_FAILED_TO_CLOSE_SQL_SESSION; SQLSTATE: HY000

  Message: Error closing SQL session

  ER_XPLUGIN_FAILED_TO_CLOSE_SQL_SESSION was added in 8.0.11.

- Error number: MY-011338; Symbol: ER_XPLUGIN_FAILED_TO_EXECUTE_ADMIN_CMD; SQLSTATE: HY000

  Message: Error executing admin command %s: %s

  ER_XPLUGIN_FAILED_TO_EXECUTE_ADMIN_CMD was added in 8.0.11.

- Error number: MY-011339; Symbol: ER_XPLUGIN_EMPTY_ADMIN_CMD; SQLSTATE: HY000

  Message: Error executing empty admin command

  ER_XPLUGIN_EMPTY_ADMIN_CMD was added in 8.0.11.

- Error number: MY-011340; Symbol: ER_XPLUGIN_FAILED_TO_GET_SYS_VAR; SQLSTATE: HY000

  Message: Unable to retrieve system variable \'%s\'

  ER_XPLUGIN_FAILED_TO_GET_SYS_VAR was added in 8.0.11.

- Error number: MY-011341; Symbol: ER_XPLUGIN_FAILED_TO_GET_CREATION_STMT; SQLSTATE: HY000

  Message: Unable to get creation stmt for collection \'%s\'; query result size: %lu

  ER_XPLUGIN_FAILED_TO_GET_CREATION_STMT was added in 8.0.11.

- Error number: MY-011342; Symbol: ER_XPLUGIN_FAILED_TO_GET_ENGINE_INFO; SQLSTATE: HY000

  Message: Unable to get engine info for collection \'%s\'; creation stmt: %s

  ER_XPLUGIN_FAILED_TO_GET_ENGINE_INFO was added in 8.0.11.

- Error number: MY-011343; Symbol: ER_XPLUGIN_FAIL_TO_GET_RESULT_DATA; SQLSTATE: HY000

  Message: Error getting result data: %s

  ER_XPLUGIN_FAIL_TO_GET_RESULT_DATA was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011344; Symbol: ER_XPLUGIN_CAPABILITY_EXPIRED_PASSWORD; SQLSTATE: HY000

  Message: Capability expired password failed with error: %s

  ER_XPLUGIN_CAPABILITY_EXPIRED_PASSWORD was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011345; Symbol: ER_XPLUGIN_FAILED_TO_SET_SO_REUSEADDR_FLAG; SQLSTATE: HY000

  Message: Failed to set SO_REUSEADDR flag (error: %d).

  ER_XPLUGIN_FAILED_TO_SET_SO_REUSEADDR_FLAG was added in 8.0.11.

- Error number: MY-011346; Symbol: ER_XPLUGIN_FAILED_TO_OPEN_INTERNAL_SESSION; SQLSTATE: HY000

  Message: Could not open internal MySQL session

  ER_XPLUGIN_FAILED_TO_OPEN_INTERNAL_SESSION was added in 8.0.11.

- Error number: MY-011347; Symbol: ER_XPLUGIN_FAILED_TO_SWITCH_CONTEXT; SQLSTATE: HY000

  Message: Unable to switch context to user %s

  ER_XPLUGIN_FAILED_TO_SWITCH_CONTEXT was added in 8.0.11.

- Error number: MY-011348; Symbol: ER_XPLUGIN_FAILED_TO_UNREGISTER_UDF; SQLSTATE: HY000

  Message: Can\'t unregister \'%s\' user defined function

  ER_XPLUGIN_FAILED_TO_UNREGISTER_UDF was added in 8.0.11.

- Error number: MY-011349; Symbol: ER_XPLUGIN_GET_PEER_ADDRESS_FAILED; SQLSTATE: HY000

  Message: %s: get peer address failed, can't resolve IP to hostname

  ER_XPLUGIN_GET_PEER_ADDRESS_FAILED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011350; Symbol: ER_XPLUGIN_CAPABILITY_CLIENT_INTERACTIVE_FAILED; SQLSTATE: HY000

  Message: Capability client interactive failed with error: %s

  ER_XPLUGIN_CAPABILITY_CLIENT_INTERACTIVE_FAILED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011351; Symbol: ER_XPLUGIN_FAILED_TO_RESET_IPV6_V6ONLY_FLAG; SQLSTATE: HY000

  Message: Failed to reset IPV6_V6ONLY flag (error: %d). The server will listen to IPv6 addresses only.

  ER_XPLUGIN_FAILED_TO_RESET_IPV6_V6ONLY_FLAG was added in 8.0.11.

- Error number: MY-011352; Symbol: ER_KEYRING_INVALID_KEY_TYPE; SQLSTATE: HY000

Message: Invalid key type

ER_KEYRING_INVALID_KEY_TYPE was added in 8.0.11.

- Error number: MY-011353; Symbol: ER_KEYRING_INVALID_KEY_LENGTH; SQLSTATE: HY000

  Message: Invalid key length for given block cipher

  ER_KEYRING_INVALID_KEY_LENGTH was added in 8.0.11.

- Error number: MY-011354; Symbol: ER_KEYRING_FAILED_TO_CREATE_KEYRING_DIR; SQLSTATE: HY000

  Message: Could not create keyring directory. The keyring_file will stay unusable until correct path to the keyring directory gets provided

  ER_KEYRING_FAILED_TO_CREATE_KEYRING_DIR was added in 8.0.11.

- Error number: MY-011355; Symbol: ER_KEYRING_FILE_INIT_FAILED; SQLSTATE: HY000

  Message: keyring_file initialization failure. Please check if the keyring_file_data points to readable keyring file or keyring file can be created in the specified location. The keyring_file will stay unusable until correct path to the keyring file gets provided

  ER_KEYRING_FILE_INIT_FAILED was added in 8.0.11.

- Error number: MY-011356; Symbol: ER_KEYRING_INTERNAL_EXCEPTION_FAILED_FILE_INIT; SQLSTATE: HY000

  Message: keyring_file initialization failure due to internal exception inside the plugin.

  ER_KEYRING_INTERNAL_EXCEPTION_FAILED_FILE_INIT was added in 8.0.11.

- Error number: MY-011357; Symbol: ER_KEYRING_FAILED_TO_GENERATE_KEY; SQLSTATE: HY000

  Message: Failed to generate a key due to internal exception inside keyring_file plugin

  ER_KEYRING_FAILED_TO_GENERATE_KEY was added in 8.0.11.

- Error number: MY-011358; Symbol: ER_KEYRING_CHECK_KEY_FAILED_DUE_TO_INVALID_KEY; SQLSTATE: HY000

  Message: Error while %s key: invalid key_type

  ER_KEYRING_CHECK_KEY_FAILED_DUE_TO_INVALID_KEY was added in 8.0.11.

- Error number: MY-011359; Symbol: ER_KEYRING_CHECK_KEY_FAILED_DUE_TO_EMPTY_KEY_ID; SQLSTATE: HY000

  Message: Error while %s key: key_id cannot be empty

  ER_KEYRING_CHECK_KEY_FAILED_DUE_TO_EMPTY_KEY_ID was added in 8.0.11.

- Error number: MY-011360; Symbol: ER_KEYRING_OPERATION_FAILED_DUE_TO_INTERNAL_ERROR; SQLSTATE: HY000

  Message: Failed to %s due to internal exception inside %s plugin

ER_KEYRING_OPERATION_FAILED_DUE_TO_INTERNAL_ERROR was added in 8.0.11.

- Error number: MY-011361; Symbol: ER_KEYRING_INCORRECT_FILE; SQLSTATE: HY000

  Message: Incorrect Keyring file

  ER_KEYRING_INCORRECT_FILE was added in 8.0.11.

- Error number: MY-011362; Symbol: ER_KEYRING_FOUND_MALFORMED_BACKUP_FILE; SQLSTATE: HY000

  Message: Found malformed keyring backup file - removing it

  ER_KEYRING_FOUND_MALFORMED_BACKUP_FILE was added in 8.0.11.

- Error number: MY-011363; Symbol: ER_KEYRING_FAILED_TO_RESTORE_FROM_BACKUP_FILE; SQLSTATE: HY000

  Message: Error while restoring keyring from backup file cannot overwrite keyring with backup

  ER_KEYRING_FAILED_TO_RESTORE_FROM_BACKUP_FILE was added in 8.0.11.

- Error number: MY-011364; Symbol: ER_KEYRING_FAILED_TO_FLUSH_KEYRING_TO_FILE; SQLSTATE: HY000

  Message: Error while flushing in-memory keyring into keyring file

  ER_KEYRING_FAILED_TO_FLUSH_KEYRING_TO_FILE was added in 8.0.11.

- Error number: MY-011365; Symbol: ER_KEYRING_FAILED_TO_GET_FILE_STAT; SQLSTATE: HY000

  Message: Error while reading stat for %s.Please check if file %s was not removed. OS returned this error: %s

  ER_KEYRING_FAILED_TO_GET_FILE_STAT was added in 8.0.11.

- Error number: MY-011366; Symbol: ER_KEYRING_FAILED_TO_REMOVE_FILE; SQLSTATE: HY000

  Message: Could not remove file %s OS retuned this error: %s

  ER_KEYRING_FAILED_TO_REMOVE_FILE was added in 8.0.11.

- Error number: MY-011367; Symbol: ER_KEYRING_FAILED_TO_TRUNCATE_FILE; SQLSTATE: HY000

  Message: Could not truncate file %s. OS retuned this error: %s

  ER_KEYRING_FAILED_TO_TRUNCATE_FILE was added in 8.0.11.

- Error number: MY-011368; Symbol: ER_KEYRING_UNKNOWN_ERROR; SQLSTATE: HY000

  Message: Unknown error %d

  ER_KEYRING_UNKNOWN_ERROR was added in 8.0.11.

- Error number: MY-011369; Symbol: ER_KEYRING_FAILED_TO_SET_KEYRING_FILE_DATA; SQLSTATE: HY000

Message: keyring_file_data cannot be set to new value as the keyring file cannot be created/accessed in the provided path

ER_KEYRING_FAILED_TO_SET_KEYRING_FILE_DATA was added in 8.0.11.

- Error number: MY-011370; Symbol: ER_KEYRING_FILE_IO_ERROR; SQLSTATE: HY000

  Message: %s

  ER_KEYRING_FILE_IO_ERROR was added in 8.0.11.

- Error number: MY-011371; Symbol: ER_KEYRING_FAILED_TO_LOAD_KEYRING_CONTENT; SQLSTATE: HY000

  Message: Error while loading keyring content. The keyring might be malformed

  ER_KEYRING_FAILED_TO_LOAD_KEYRING_CONTENT was added in 8.0.11.

- Error number: MY-011372; Symbol: ER_KEYRING_FAILED_TO_FLUSH_KEYS_TO_KEYRING; SQLSTATE: HY000

  Message: Could not flush keys to keyring

  ER_KEYRING_FAILED_TO_FLUSH_KEYS_TO_KEYRING was added in 8.0.11.

- Error number: MY-011373; Symbol: ER_KEYRING_FAILED_TO_FLUSH_KEYS_TO_KEYRING_BACKUP; SQLSTATE: HY000

  Message: Could not flush keys to keyring's backup

  ER_KEYRING_FAILED_TO_FLUSH_KEYS_TO_KEYRING_BACKUP was added in 8.0.11.

- Error number: MY-011374; Symbol: ER_KEYRING_KEY_FETCH_FAILED_DUE_TO_EMPTY_KEY_ID; SQLSTATE: HY000

  Message: Error while fetching key: key_id cannot be empty

  ER_KEYRING_KEY_FETCH_FAILED_DUE_TO_EMPTY_KEY_ID was added in 8.0.11.

- Error number: MY-011375; Symbol: ER_KEYRING_FAILED_TO_REMOVE_KEY_DUE_TO_EMPTY_ID; SQLSTATE: HY000

  Message: Error while removing key: key_id cannot be empty

  ER_KEYRING_FAILED_TO_REMOVE_KEY_DUE_TO_EMPTY_ID was added in 8.0.11.

- Error number: MY-011376; Symbol: ER_KEYRING_OKV_INCORRECT_KEY_VAULT_CONFIGURED; SQLSTATE: HY000

  Message: For keyring_okv to be initialized, please point keyring_okv_conf_dir variable to a directory with Oracle Key Vault configuration file and ssl materials

  ER_KEYRING_OKV_INCORRECT_KEY_VAULT_CONFIGURED was added in 8.0.11.

- Error number: MY-011377; Symbol: ER_KEYRING_OKV_INIT_FAILED_DUE_TO_INCORRECT_CONF; SQLSTATE: HY000

Message: keyring_okv initialization failure. Please check that the keyring_okv_conf_dir points to a readable directory and that the directory contains Oracle Key Vault configuration file and ssl materials. Please also check that Oracle Key Vault is up and running.

ER_KEYRING_OKV_INIT_FAILED_DUE_TO_INCORRECT_CONF was added in 8.0.11.

- Error number: MY-011378; Symbol: ER_KEYRING_OKV_INIT_FAILED_DUE_TO_INTERNAL_ERROR; SQLSTATE: HY000

  Message: keyring_okv initialization failure due to internal exception inside the plugin

  ER_KEYRING_OKV_INIT_FAILED_DUE_TO_INTERNAL_ERROR was added in 8.0.11.

- Error number: MY-011379; Symbol: ER_KEYRING_OKV_INVALID_KEY_TYPE; SQLSTATE: HY000

  Message: Invalid key type

  ER_KEYRING_OKV_INVALID_KEY_TYPE was added in 8.0.11.

- Error number: MY-011380; Symbol: ER_KEYRING_OKV_INVALID_KEY_LENGTH_FOR_CIPHER; SQLSTATE: HY000

  Message: Invalid key length for given block cipher

  ER_KEYRING_OKV_INVALID_KEY_LENGTH_FOR_CIPHER was added in 8.0.11.

- Error number: MY-011381; Symbol: ER_KEYRING_OKV_FAILED_TO_GENERATE_KEY_DUE_TO_INTERNAL_ERROR; SQLSTATE: HY000

  Message: Failed to generate a key due to internal exception inside keyring_okv plugin

  ER_KEYRING_OKV_FAILED_TO_GENERATE_KEY_DUE_TO_INTERNAL_ERROR was added in 8.0.11.

- Error number: MY-011382; Symbol: ER_KEYRING_OKV_FAILED_TO_FIND_SERVER_ENTRY; SQLSTATE: HY000

  Message: Could not find entry for server in configuration file %s

  ER_KEYRING_OKV_FAILED_TO_FIND_SERVER_ENTRY was added in 8.0.11.

- Error number: MY-011383; Symbol: ER_KEYRING_OKV_FAILED_TO_FIND_STANDBY_SERVER_ENTRY; SQLSTATE: HY000

  Message: Could not find entry for standby server in configuration file %s

  ER_KEYRING_OKV_FAILED_TO_FIND_STANDBY_SERVER_ENTRY was added in 8.0.11.

- Error number: MY-011384; Symbol: ER_KEYRING_OKV_FAILED_TO_PARSE_CONF_FILE; SQLSTATE: HY000

  Message: Could not parse the %s file provided

  ER_KEYRING_OKV_FAILED_TO_PARSE_CONF_FILE was added in 8.0.11.

- Error number: MY-011385; Symbol: ER_KEYRING_OKV_FAILED_TO_LOAD_KEY_UID; SQLSTATE: HY000

  Message: Could not load keys' uids from the OKV server

ER_KEYRING_OKV_FAILED_TO_LOAD_KEY_UID was added in 8.0.11.

- Error number: MY-011386; Symbol: ER_KEYRING_OKV_FAILED_TO_INIT_SSL_LAYER; SQLSTATE: HY000

  Message: Could not initialize ssl layer

  ER_KEYRING_OKV_FAILED_TO_INIT_SSL_LAYER was added in 8.0.11.

- Error number: MY-011387; Symbol: ER_KEYRING_OKV_FAILED_TO_INIT_CLIENT; SQLSTATE: HY000

  Message: Could not initialize OKV client

  ER_KEYRING_OKV_FAILED_TO_INIT_CLIENT was added in 8.0.11.

- Error number: MY-011388; Symbol: ER_KEYRING_OKV_CONNECTION_TO_SERVER_FAILED; SQLSTATE: HY000

  Message: Could not connect to the OKV server

  ER_KEYRING_OKV_CONNECTION_TO_SERVER_FAILED was added in 8.0.11.

- Error number: MY-011389; Symbol: ER_KEYRING_OKV_FAILED_TO_REMOVE_KEY; SQLSTATE: HY000

  Message: Could not remove the key

  ER_KEYRING_OKV_FAILED_TO_REMOVE_KEY was added in 8.0.11.

- Error number: MY-011390; Symbol: ER_KEYRING_OKV_FAILED_TO_ADD_ATTRIBUTE; SQLSTATE: HY000

  Message: Could not add attribute, attribute_name=%s attribute value=%s

  ER_KEYRING_OKV_FAILED_TO_ADD_ATTRIBUTE was added in 8.0.11.

- Error number: MY-011391; Symbol: ER_KEYRING_OKV_FAILED_TO_GENERATE_KEY; SQLSTATE: HY000

  Message: Could not generate the key.

  ER_KEYRING_OKV_FAILED_TO_GENERATE_KEY was added in 8.0.11.

- Error number: MY-011392; Symbol: ER_KEYRING_OKV_FAILED_TO_STORE_KEY; SQLSTATE: HY000

  Message: Could not store the key.

  ER_KEYRING_OKV_FAILED_TO_STORE_KEY was added in 8.0.11.

- Error number: MY-011393; Symbol: ER_KEYRING_OKV_FAILED_TO_ACTIVATE_KEYS; SQLSTATE: HY000

  Message: Could not activate the key.

  ER_KEYRING_OKV_FAILED_TO_ACTIVATE_KEYS was added in 8.0.11.

- Error number: MY-011394; Symbol: ER_KEYRING_OKV_FAILED_TO_FETCH_KEY; SQLSTATE: HY000

  Message: Could not fetch generated key

  ER_KEYRING_OKV_FAILED_TO_FETCH_KEY was added in 8.0.11.

- Error number: MY-011395; Symbol: ER_KEYRING_OKV_FAILED_TO_STORE_OR_GENERATE_KEY; SQLSTATE: HY000

  Message: Could not store/generate the key - failed to set key attribute x-key-ready

  ER_KEYRING_OKV_FAILED_TO_STORE_OR_GENERATE_KEY was added in 8.0.11.

- Error number: MY-011396; Symbol: ER_KEYRING_OKV_FAILED_TO_RETRIEVE_KEY_SIGNATURE; SQLSTATE: HY000

  Message: Could not retrieve key signature from custom attributes

  ER_KEYRING_OKV_FAILED_TO_RETRIEVE_KEY_SIGNATURE was added in 8.0.11.

- Error number: MY-011397; Symbol: ER_KEYRING_OKV_FAILED_TO_RETRIEVE_KEY; SQLSTATE: HY000

  Message: Could not retrieve key from OKV

  ER_KEYRING_OKV_FAILED_TO_RETRIEVE_KEY was added in 8.0.11.

- Error number: MY-011398; Symbol: ER_KEYRING_OKV_FAILED_TO_LOAD_SSL_TRUST_STORE; SQLSTATE: HY000

  Message: Error loading trust store

  ER_KEYRING_OKV_FAILED_TO_LOAD_SSL_TRUST_STORE was added in 8.0.11.

- Error number: MY-011399; Symbol: ER_KEYRING_OKV_FAILED_TO_SET_CERTIFICATE_FILE; SQLSTATE: HY000

  Message: Error setting the certificate file.

  ER_KEYRING_OKV_FAILED_TO_SET_CERTIFICATE_FILE was added in 8.0.11.

- Error number: MY-011400; Symbol: ER_KEYRING_OKV_FAILED_TO_SET_KEY_FILE; SQLSTATE: HY000

  Message: Error setting the key file.

  ER_KEYRING_OKV_FAILED_TO_SET_KEY_FILE was added in 8.0.11.

- Error number: MY-011401; Symbol: ER_KEYRING_OKV_KEY_MISMATCH; SQLSTATE: HY000

  Message: Private key does not match the certificate public key

  ER_KEYRING_OKV_KEY_MISMATCH was added in 8.0.11.

- Error number: MY-011402; Symbol: ER_KEYRING_ENCRYPTED_FILE_INCORRECT_KEYRING_FILE; SQLSTATE: HY000

  Message: Incorrect Keyring file

ER_KEYRING_ENCRYPTED_FILE_INCORRECT_KEYRING_FILE was added in 8.0.11.

- Error number: MY-011403; Symbol: ER_KEYRING_ENCRYPTED_FILE_DECRYPTION_FAILED; SQLSTATE: HY000

  Message: Keyring_encrypted_file decryption failed. Please verify --keyring-encrypted-file-password option value.

  ER_KEYRING_ENCRYPTED_FILE_DECRYPTION_FAILED was added in 8.0.11.

- Error number: MY-011404; Symbol: ER_KEYRING_ENCRYPTED_FILE_FOUND_MALFORMED_BACKUP_FILE; SQLSTATE: HY000

  Message: Found malformed keyring backup file - removing it

  ER_KEYRING_ENCRYPTED_FILE_FOUND_MALFORMED_BACKUP_FILE was added in 8.0.11.

- Error number: MY-011405; Symbol: ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_RESTORE_KEYRING; SQLSTATE: HY000

  Message: Error while restoring keyring from backup file cannot overwrite keyring with backup

  ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_RESTORE_KEYRING was added in 8.0.11.

- Error number: MY-011406; Symbol: ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_FLUSH_KEYRING; SQLSTATE: HY000

  Message: Error while flushing in-memory keyring into keyring file

  ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_FLUSH_KEYRING was added in 8.0.11.

- Error number: MY-011407; Symbol: ER_KEYRING_ENCRYPTED_FILE_ENCRYPTION_FAILED; SQLSTATE: HY000

  Message: Keyring_encrypted_file encryption failed. Please verify --keyring-encrypted-file-password option value.

  ER_KEYRING_ENCRYPTED_FILE_ENCRYPTION_FAILED was added in 8.0.11.

- Error number: MY-011408; Symbol: ER_KEYRING_ENCRYPTED_FILE_INVALID_KEYRING_DIR; SQLSTATE: HY000

  Message: keyring_encrypted_file_data cannot be set to new value as the keyring file cannot be created/accessed in the provided path

  ER_KEYRING_ENCRYPTED_FILE_INVALID_KEYRING_DIR was added in 8.0.11.

- Error number: MY-011409; Symbol: ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_CREATE_KEYRING_DIR; SQLSTATE: HY000

  Message: Could not create keyring directory The keyring_encrypted_file will stay unusable until correct path to the keyring directory gets provided

  ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_CREATE_KEYRING_DIR was added in 8.0.11.

- Error number: MY-011410; Symbol: ER_KEYRING_ENCRYPTED_FILE_PASSWORD_IS_INVALID; SQLSTATE: HY000

Message: The keyring_encrypted_file_password must be set to a valid value.

ER_KEYRING_ENCRYPTED_FILE_PASSWORD_IS_INVALID was added in 8.0.11.

- Error number: MY-011411; Symbol: ER_KEYRING_ENCRYPTED_FILE_PASSWORD_IS_TOO_LONG; SQLSTATE: HY000

  Message: Too long keyring_encrypted_file_password value.

  ER_KEYRING_ENCRYPTED_FILE_PASSWORD_IS_TOO_LONG was added in 8.0.11.

- Error number: MY-011412; Symbol: ER_KEYRING_ENCRYPTED_FILE_INIT_FAILURE; SQLSTATE: HY000

  Message: keyring_encrypted_file initialization failure. Please check if the keyring_encrypted_file_data points to readable keyring file or keyring file can be created in the specified location or password to decrypt keyring file is correct.

  ER_KEYRING_ENCRYPTED_FILE_INIT_FAILURE was added in 8.0.11.

- Error number: MY-011413; Symbol: ER_KEYRING_ENCRYPTED_FILE_INIT_FAILED_DUE_TO_INTERNAL_ERROR; SQLSTATE: HY000

  Message: keyring_encrypted_file initialization failure due to internal exception inside the plugin

  ER_KEYRING_ENCRYPTED_FILE_INIT_FAILED_DUE_TO_INTERNAL_ERROR was added in 8.0.11.

- Error number: MY-011414; Symbol: ER_KEYRING_ENCRYPTED_FILE_GEN_KEY_FAILED_DUE_TO_INTERNAL_ERROR; SQLSTATE: HY000

  Message: Failed to generate a key due to internal exception inside keyring_encrypted_file plugin

  ER_KEYRING_ENCRYPTED_FILE_GEN_KEY_FAILED_DUE_TO_INTERNAL_ERROR was added in 8.0.11.

- Error number: MY-011415; Symbol: ER_KEYRING_AWS_FAILED_TO_SET_CMK_ID; SQLSTATE: HY000

  Message: keyring_aws_cmk_id cannot be set to the new value as AWS KMS seems to not understand the id provided. Please check that CMK id provided is correct.

  ER_KEYRING_AWS_FAILED_TO_SET_CMK_ID was added in 8.0.11.

- Error number: MY-011416; Symbol: ER_KEYRING_AWS_FAILED_TO_SET_REGION; SQLSTATE: HY000

  Message: keyring_aws_region cannot be set to the new value as AWS KMS seems to not understand the region provided. Please check that region provided is correct.

  ER_KEYRING_AWS_FAILED_TO_SET_REGION was added in 8.0.11.

- Error number: MY-011417; Symbol: ER_KEYRING_AWS_FAILED_TO_OPEN_CONF_FILE; SQLSTATE: HY000

  Message: Could not open keyring_aws configuration file: %s. OS returned this error: %s

  ER_KEYRING_AWS_FAILED_TO_OPEN_CONF_FILE was added in 8.0.11.

- Error number: MY-011418; Symbol: ER_KEYRING_AWS_FAILED_TO_ACCESS_KEY_ID_FROM_CONF_FILE; SQLSTATE: HY000

  Message: Could not read AWS access key id from keyring_aws configuration file: %s. OS returned this error: %s

  ER_KEYRING_AWS_FAILED_TO_ACCESS_KEY_ID_FROM_CONF_FILE was added in 8.0.11.

- Error number: MY-011419; Symbol: ER_KEYRING_AWS_FAILED_TO_ACCESS_KEY_FROM_CONF_FILE; SQLSTATE: HY000

  Message: Could not read AWS access key from keyring_aws configuration file: %s. OS returned this error: %s

  ER_KEYRING_AWS_FAILED_TO_ACCESS_KEY_FROM_CONF_FILE was added in 8.0.11.

- Error number: MY-011420; Symbol: ER_KEYRING_AWS_INVALID_CONF_FILE_PATH; SQLSTATE: HY000

  Message: Path to keyring aws configuration file cannot be empty

  ER_KEYRING_AWS_INVALID_CONF_FILE_PATH was added in 8.0.11.

- Error number: MY-011421; Symbol: ER_KEYRING_AWS_INVALID_DATA_FILE_PATH; SQLSTATE: HY000

  Message: Path to keyring_aws storage file cannot be empty.

  ER_KEYRING_AWS_INVALID_DATA_FILE_PATH was added in 8.0.11.

- Error number: MY-011422; Symbol: ER_KEYRING_AWS_FAILED_TO_ACCESS_OR_CREATE_KEYRING_DIR; SQLSTATE: HY000

  Message: Unable to create/access keyring directory.

  ER_KEYRING_AWS_FAILED_TO_ACCESS_OR_CREATE_KEYRING_DIR was added in 8.0.11.

- Error number: MY-011423; Symbol: ER_KEYRING_AWS_FAILED_TO_ACCESS_OR_CREATE_KEYRING_DATA_FILE; SQLSTATE: HY000

  Message: Unable to create/access keyring_aws storage file. Please check if keyring_aws_data_file points to location where keyring file can be created/accessed. Please also make sure that MySQL server's user has high enough privileges to access this location.

  ER_KEYRING_AWS_FAILED_TO_ACCESS_OR_CREATE_KEYRING_DATA_FILE was added in 8.0.11.

- Error number: MY-011424; Symbol: ER_KEYRING_AWS_FAILED_TO_INIT_DUE_TO_INTERNAL_ERROR; SQLSTATE: HY000

  Message: keyring_aws initialization failed due to internal error when initializing synchronization primitive - %s. OS returned this error: %s:

  ER_KEYRING_AWS_FAILED_TO_INIT_DUE_TO_INTERNAL_ERROR was added in 8.0.11.

- Error number: MY-011425; Symbol: ER_KEYRING_AWS_FAILED_TO_ACCESS_DATA_FILE; SQLSTATE: HY000

Message: Could not access keyring_aws storage file in the path provided. Please check if the keyring_aws directory can be accessed by MySQL Server

ER_KEYRING_AWS_FAILED_TO_ACCESS_DATA_FILE was added in 8.0.11.

- Error number: MY-011426; Symbol: ER_KEYRING_AWS_CMK_ID_NOT_SET; SQLSTATE: HY000

  Message: keyring_aws_cmk_id has to be set

  ER_KEYRING_AWS_CMK_ID_NOT_SET was added in 8.0.11.

- Error number: MY-011427; Symbol: ER_KEYRING_AWS_FAILED_TO_GET_KMS_CREDENTIAL_FROM_CONF_FILE; SQLSTATE: HY000

  Message: Could not get AWS KMS credentials from the configuration file

  ER_KEYRING_AWS_FAILED_TO_GET_KMS_CREDENTIAL_FROM_CONF_FILE was added in 8.0.11.

- Error number: MY-011428; Symbol: ER_KEYRING_AWS_INIT_FAILURE; SQLSTATE: HY000

  Message: keyring_aws initialization failure.

  ER_KEYRING_AWS_INIT_FAILURE was added in 8.0.11.

- Error number: MY-011429; Symbol: ER_KEYRING_AWS_FAILED_TO_INIT_DUE_TO_PLUGIN_INTERNAL_ERROR; SQLSTATE: HY000

  Message: keyring_aws initialization failure due to internal exception inside the plugin

  ER_KEYRING_AWS_FAILED_TO_INIT_DUE_TO_PLUGIN_INTERNAL_ERROR was added in 8.0.11.

- Error number: MY-011430; Symbol: ER_KEYRING_AWS_INVALID_KEY_LENGTH_FOR_CIPHER; SQLSTATE: HY000

  Message: Invalid key length for given block cipher

  ER_KEYRING_AWS_INVALID_KEY_LENGTH_FOR_CIPHER was added in 8.0.11.

- Error number: MY-011431; Symbol: ER_KEYRING_AWS_FAILED_TO_GENERATE_KEY_DUE_TO_INTERNAL_ERROR; SQLSTATE: HY000

  Message: Failed to generate a key due to internal exception inside keyring_file plugin

  ER_KEYRING_AWS_FAILED_TO_GENERATE_KEY_DUE_TO_INTERNAL_ERROR was added in 8.0.11.

- Error number: MY-011432; Symbol: ER_KEYRING_AWS_INCORRECT_FILE; SQLSTATE: HY000

  Message: Incorrect Keyring file

  ER_KEYRING_AWS_INCORRECT_FILE was added in 8.0.11.

- Error number: MY-011433; Symbol: ER_KEYRING_AWS_FOUND_MALFORMED_BACKUP_FILE; SQLSTATE: HY000

  Message: Found malformed keyring backup file - removing it

  ER_KEYRING_AWS_FOUND_MALFORMED_BACKUP_FILE was added in 8.0.11.

- Error number: MY-011434; Symbol: ER_KEYRING_AWS_FAILED_TO_RESTORE_FROM_BACKUP_FILE; SQLSTATE: HY000

  Message: Error while restoring keyring from backup file cannot overwrite keyring with backup

  ER_KEYRING_AWS_FAILED_TO_RESTORE_FROM_BACKUP_FILE was added in 8.0.11.

- Error number: MY-011435; Symbol: ER_KEYRING_AWS_FAILED_TO_FLUSH_KEYRING_TO_FILE; SQLSTATE: HY000

  Message: Error while flushing in-memory keyring into keyring file

  ER_KEYRING_AWS_FAILED_TO_FLUSH_KEYRING_TO_FILE was added in 8.0.11.

- Error number: MY-011436; Symbol: ER_KEYRING_AWS_INCORRECT_REGION; SQLSTATE: HY000

  Message: Wrong region

  ER_KEYRING_AWS_INCORRECT_REGION was added in 8.0.11.

- Error number: MY-011437; Symbol: ER_KEYRING_AWS_FAILED_TO_CONNECT_KMS; SQLSTATE: HY000

  Message: Could not connect to AWS KMS with the credentials provided. Please make sure they are correct. AWS KMS returned this error: %s

  ER_KEYRING_AWS_FAILED_TO_CONNECT_KMS was added in 8.0.11.

- Error number: MY-011438; Symbol: ER_KEYRING_AWS_FAILED_TO_GENERATE_NEW_KEY; SQLSTATE: HY000

  Message: Could not generate a new key. AWS KMS returned this error: %s

  ER_KEYRING_AWS_FAILED_TO_GENERATE_NEW_KEY was added in 8.0.11.

- Error number: MY-011439; Symbol: ER_KEYRING_AWS_FAILED_TO_ENCRYPT_KEY; SQLSTATE: HY000

  Message: Could not encrypt key. AWS KMS returned this error: %s

  ER_KEYRING_AWS_FAILED_TO_ENCRYPT_KEY was added in 8.0.11.

- Error number: MY-011440; Symbol: ER_KEYRING_AWS_FAILED_TO_RE_ENCRYPT_KEY; SQLSTATE: HY000

  Message: Could not re-encrypt key. AWS KMS returned this error: %s

  ER_KEYRING_AWS_FAILED_TO_RE_ENCRYPT_KEY was added in 8.0.11.

- Error number: MY-011441; Symbol: ER_KEYRING_AWS_FAILED_TO_DECRYPT_KEY; SQLSTATE: HY000

  Message: Could not decrypt key. AWS KMS returned this error: %s

  ER_KEYRING_AWS_FAILED_TO_DECRYPT_KEY was added in 8.0.11.

- Error number: MY-011442; Symbol: ER_KEYRING_AWS_FAILED_TO_ROTATE_CMK; SQLSTATE: HY000

Message: Could not rotate the CMK. AWS KMS returned this error: %s

ER_KEYRING_AWS_FAILED_TO_ROTATE_CMK was added in 8.0.11.

- Error number: MY-011443; Symbol: ER_GRP_RPL_GTID_ALREADY_USED; SQLSTATE: HY000

  Message: The requested GTID '%s:%lld' was already used, the transaction will rollback.

  ER_GRP_RPL_GTID_ALREADY_USED was added in 8.0.11.

- Error number: MY-011444; Symbol: ER_GRP_RPL_APPLIER_THD_KILLED; SQLSTATE: HY000

  Message: The group replication applier thread was killed.

  ER_GRP_RPL_APPLIER_THD_KILLED was added in 8.0.11.

- Error number: MY-011445; Symbol: ER_GRP_RPL_EVENT_HANDLING_ERROR; SQLSTATE: HY000

  Message: Error at event handling! Got error: %d.

  ER_GRP_RPL_EVENT_HANDLING_ERROR was added in 8.0.11.

- Error number: MY-011446; Symbol: ER_GRP_RPL_ERROR_GTID_EXECUTION_INFO; SQLSTATE: HY000

  Message: Error when extracting group GTID execution information, some recovery operations may face future issues.

  ER_GRP_RPL_ERROR_GTID_EXECUTION_INFO was added in 8.0.11.

- Error number: MY-011447; Symbol: ER_GRP_RPL_CERTIFICATE_SIZE_ERROR; SQLSTATE: HY000

  Message: An error occurred when trying to reduce the Certification information size for transmission.

  ER_GRP_RPL_CERTIFICATE_SIZE_ERROR was added in 8.0.11.

- Error number: MY-011448; Symbol: ER_GRP_RPL_CREATE_APPLIER_CACHE_ERROR; SQLSTATE: HY000

  Message: Failed to create group replication pipeline applier cache!

  ER_GRP_RPL_CREATE_APPLIER_CACHE_ERROR was added in 8.0.11.

- Error number: MY-011449; Symbol: ER_GRP_RPL_UNBLOCK_WAITING_THD; SQLSTATE: HY000

  Message: Unblocking the group replication thread waiting for applier to start, as the start group replication was killed.

  ER_GRP_RPL_UNBLOCK_WAITING_THD was added in 8.0.11.

- Error number: MY-011450; Symbol: ER_GRP_RPL_APPLIER_PIPELINE_NOT_DISPOSED; SQLSTATE: HY000

  Message: The group replication applier pipeline was not properly disposed. Check the error log for further info.

  ER_GRP_RPL_APPLIER_PIPELINE_NOT_DISPOSED was added in 8.0.11.

- Error number: MY-011451; Symbol: ER_GRP_RPL_APPLIER_THD_EXECUTION_ABORTED; SQLSTATE: HY000

  Message: The applier thread execution was aborted. Unable to process more transactions, this member will now leave the group.

  ER_GRP_RPL_APPLIER_THD_EXECUTION_ABORTED was added in 8.0.11.

- Error number: MY-011452; Symbol: ER_GRP_RPL_APPLIER_EXECUTION_FATAL_ERROR; SQLSTATE: HY000

  Message: Fatal error during execution on the Applier process of Group Replication. The server will now leave the group.

  ER_GRP_RPL_APPLIER_EXECUTION_FATAL_ERROR was added in 8.0.11.

- Error number: MY-011453; Symbol: ER_GRP_RPL_ERROR_STOPPING_CHANNELS; SQLSTATE: HY000

  Message: Error stopping all replication channels while server was leaving the group. %s

  ER_GRP_RPL_ERROR_STOPPING_CHANNELS was added in 8.0.11.

- Error number: MY-011454; Symbol: ER_GRP_RPL_ERROR_SENDING_SINGLE_PRIMARY_MSSG; SQLSTATE: HY000

  Message: Error sending single primary message informing that primary did apply relay logs.

  ER_GRP_RPL_ERROR_SENDING_SINGLE_PRIMARY_MSSG was added in 8.0.11.

- Error number: MY-011455; Symbol: ER_GRP_RPL_UPDATE_TRANS_SNAPSHOT_VER_ERROR; SQLSTATE: HY000

  Message: Error updating transaction snapshot version after transaction being positively certified.

  ER_GRP_RPL_UPDATE_TRANS_SNAPSHOT_VER_ERROR was added in 8.0.11.

- Error number: MY-011456; Symbol: ER_GRP_RPL_SIDNO_FETCH_ERROR; SQLSTATE: HY000

  Message: Error fetching transaction sidno after transaction being positively certified.

  ER_GRP_RPL_SIDNO_FETCH_ERROR was added in 8.0.11.

- Error number: MY-011457; Symbol: ER_GRP_RPL_BROADCAST_COMMIT_TRANS_MSSG_FAILED; SQLSTATE: HY000

  Message: Broadcast of committed transactions message failed.

  ER_GRP_RPL_BROADCAST_COMMIT_TRANS_MSSG_FAILED was added in 8.0.11.

- Error number: MY-011458; Symbol: ER_GRP_RPL_GROUP_NAME_PARSE_ERROR; SQLSTATE: HY000

  Message: Unable to parse the group name during the Certification module initialization.

  ER_GRP_RPL_GROUP_NAME_PARSE_ERROR was added in 8.0.11.

- Error number: MY-011459; Symbol: ER_GRP_RPL_ADD_GRPSID_TO_GRPGTIDSID_MAP_ERROR; SQLSTATE: HY000

Message: Unable to add the group_sid in the group_gtid_sid_map during the Certification module initialization.

ER_GRP_RPL_ADD_GRPSID_TO_GRPGTIDSID_MAP_ERROR was added in 8.0.11.

- Error number: MY-011460; Symbol: ER_GRP_RPL_UPDATE_GRPGTID_EXECUTED_ERROR; SQLSTATE: HY000

  Message: Error updating group_gtid_executed GITD set during the Certification module initialization.

  ER_GRP_RPL_UPDATE_GRPGTID_EXECUTED_ERROR was added in 8.0.11.

- Error number: MY-011461; Symbol: ER_GRP_RPL_DONOR_TRANS_INFO_ERROR; SQLSTATE: HY000

  Message: Unable to handle the donor's transaction information when initializing the conflict detection component. Possible out of memory error.

  ER_GRP_RPL_DONOR_TRANS_INFO_ERROR was added in 8.0.11.

- Error number: MY-011462; Symbol: ER_GRP_RPL_SERVER_CONN_ERROR; SQLSTATE: HY000

  Message: Error when establishing a server connection during the Certification module initialization.

  ER_GRP_RPL_SERVER_CONN_ERROR was added in 8.0.11.

- Error number: MY-011463; Symbol: ER_GRP_RPL_ERROR_FETCHING_GTID_EXECUTED_SET; SQLSTATE: HY000

  Message: Error when extracting this member GTID executed set. Certification module can't be properly initialized.

  ER_GRP_RPL_ERROR_FETCHING_GTID_EXECUTED_SET was added in 8.0.11.

- Error number: MY-011464; Symbol: ER_GRP_RPL_ADD_GTID_TO_GRPGTID_EXECUTED_ERROR; SQLSTATE: HY000

  Message: Error while adding the server GTID EXECUTED set to the group_gtid_execute during the Certification module initialization.

  ER_GRP_RPL_ADD_GTID_TO_GRPGTID_EXECUTED_ERROR was added in 8.0.11.

- Error number: MY-011465; Symbol: ER_GRP_RPL_ERROR_FETCHING_GTID_SET; SQLSTATE: HY000

  Message: Error when extracting this member retrieved set for its applier. Certification module can't be properly initialized.

  ER_GRP_RPL_ERROR_FETCHING_GTID_SET was added in 8.0.11.

- Error number: MY-011466; Symbol: ER_GRP_RPL_ADD_RETRIEVED_SET_TO_GRP_GTID_EXECUTED_ERROR; SQLSTATE: HY000

  Message: Error while adding the member retrieved set to the group_gtid_executed during the Certification module initialization.

  ER_GRP_RPL_ADD_RETRIEVED_SET_TO_GRP_GTID_EXECUTED_ERROR was added in 8.0.11.

- Error number: MY-011467; Symbol: ER_GRP_RPL_CERTIFICATION_INITIALIZATION_FAILURE; SQLSTATE: HY000

  Message: Error during Certification module initialization.

  ER_GRP_RPL_CERTIFICATION_INITIALIZATION_FAILURE was added in 8.0.11.

- Error number: MY-011468; Symbol: ER_GRP_RPL_UPDATE_LAST_CONFLICT_FREE_TRANS_ERROR; SQLSTATE: HY000

  Message: Unable to update last conflict free transaction, this transaction will not be tracked on performance_schema.replication_group_member_stats.last_conflict_free_transaction.

  ER_GRP_RPL_UPDATE_LAST_CONFLICT_FREE_TRANS_ERROR was added in 8.0.11.

- Error number: MY-011469; Symbol: ER_GRP_RPL_UPDATE_TRANS_SNAPSHOT_REF_VER_ERROR; SQLSTATE: HY000

  Message: Error updating transaction snapshot version reference for internal storage.

  ER_GRP_RPL_UPDATE_TRANS_SNAPSHOT_REF_VER_ERROR was added in 8.0.11.

- Error number: MY-011470; Symbol: ER_GRP_RPL_FETCH_TRANS_SIDNO_ERROR; SQLSTATE: HY000

  Message: Error fetching transaction sidno while adding to the group_gtid_executed set.

  ER_GRP_RPL_FETCH_TRANS_SIDNO_ERROR was added in 8.0.11.

- Error number: MY-011471; Symbol: ER_GRP_RPL_ERROR_VERIFYING_SIDNO; SQLSTATE: HY000

  Message: Error while ensuring the sidno be present in the group_gtid_executed.

  ER_GRP_RPL_ERROR_VERIFYING_SIDNO was added in 8.0.11.

- Error number: MY-011472; Symbol: ER_GRP_RPL_CANT_GENERATE_GTID; SQLSTATE: HY000

  Message: Impossible to generate Global Transaction Identifier: the integer component reached the maximal value. Restart the group with a new group_replication_group_name.

  ER_GRP_RPL_CANT_GENERATE_GTID was added in 8.0.11.

- Error number: MY-011473; Symbol: ER_GRP_RPL_INVALID_GTID_SET; SQLSTATE: HY000

  Message: Invalid stable transactions set.

  ER_GRP_RPL_INVALID_GTID_SET was added in 8.0.11.

- Error number: MY-011474; Symbol: ER_GRP_RPL_UPDATE_GTID_SET_ERROR; SQLSTATE: HY000

  Message: Error updating stable transactions set.

  ER_GRP_RPL_UPDATE_GTID_SET_ERROR was added in 8.0.11.

- Error number: MY-011475; Symbol: ER_GRP_RPL_RECEIVED_SET_MISSING_GTIDS; SQLSTATE: HY000

  Message: There was an error when filling the missing GTIDs on the applier channel received set. Despite not critical, on the long run this may cause performance issues.

ER_GRP_RPL_RECEIVED_SET_MISSING_GTIDS was added in 8.0.11.

- Error number: MY-011476; Symbol: ER_GRP_RPL_SKIP_COMPUTATION_TRANS_COMMITTED; SQLSTATE: HY000

  Message: Skipping the computation of the Transactions_committed_all_members field as an older instance of this computation is still ongoing.

  ER_GRP_RPL_SKIP_COMPUTATION_TRANS_COMMITTED was added in 8.0.11.

- Error number: MY-011477; Symbol: ER_GRP_RPL_NULL_PACKET; SQLSTATE: HY000

  Message: Null packet on certifier's queue.

  ER_GRP_RPL_NULL_PACKET was added in 8.0.11.

- Error number: MY-011478; Symbol: ER_GRP_RPL_CANT_READ_GTID; SQLSTATE: HY000

  Message: Error reading GTIDs from the message.

  ER_GRP_RPL_CANT_READ_GTID was added in 8.0.11.

- Error number: MY-011479; Symbol: ER_GRP_RPL_PROCESS_GTID_SET_ERROR; SQLSTATE: HY000

  Message: Error processing stable transactions set.

  ER_GRP_RPL_PROCESS_GTID_SET_ERROR was added in 8.0.11.

- Error number: MY-011480; Symbol: ER_GRP_RPL_PROCESS_INTERSECTION_GTID_SET_ERROR; SQLSTATE: HY000

  Message: Error processing intersection of stable transactions set.

  ER_GRP_RPL_PROCESS_INTERSECTION_GTID_SET_ERROR was added in 8.0.11.

- Error number: MY-011481; Symbol: ER_GRP_RPL_SET_STABLE_TRANS_ERROR; SQLSTATE: HY000

  Message: Error setting stable transactions set.

  ER_GRP_RPL_SET_STABLE_TRANS_ERROR was added in 8.0.11.

- Error number: MY-011482; Symbol: ER_GRP_RPL_CANT_READ_GRP_GTID_EXTRACTED; SQLSTATE: HY000

  Message: Error reading group_gtid_extracted from the View_change_log_event.

  ER_GRP_RPL_CANT_READ_GRP_GTID_EXTRACTED was added in 8.0.11.

- Error number: MY-011483; Symbol: ER_GRP_RPL_CANT_READ_WRITE_SET_ITEM; SQLSTATE: HY000

  Message: Error reading the write set item '%s' from the View_change_log_event.

  ER_GRP_RPL_CANT_READ_WRITE_SET_ITEM was added in 8.0.11.

- Error number: MY-011484; Symbol: ER_GRP_RPL_INIT_CERTIFICATION_INFO_FAILURE; SQLSTATE: HY000

  Message: Error during certification_info initialization.

ER_GRP_RPL_INIT_CERTIFICATION_INFO_FAILURE was added in 8.0.11.

- Error number: MY-011485; Symbol: ER_GRP_RPL_CONFLICT_DETECTION_DISABLED; SQLSTATE: HY000

  Message: Primary had applied all relay logs, disabled conflict detection.

  ER_GRP_RPL_CONFLICT_DETECTION_DISABLED was added in 8.0.11.

- Error number: MY-011486; Symbol: ER_GRP_RPL_MSG_DISCARDED; SQLSTATE: HY000

  Message: Message received while the plugin is not ready, message discarded.

  ER_GRP_RPL_MSG_DISCARDED was added in 8.0.11.

- Error number: MY-011487; Symbol: ER_GRP_RPL_MISSING_GRP_RPL_APPLIER; SQLSTATE: HY000

  Message: Message received without a proper group replication applier.

  ER_GRP_RPL_MISSING_GRP_RPL_APPLIER was added in 8.0.11.

- Error number: MY-011488; Symbol: ER_GRP_RPL_CERTIFIER_MSSG_PROCESS_ERROR; SQLSTATE: HY000

  Message: Error processing message in Certifier.

  ER_GRP_RPL_CERTIFIER_MSSG_PROCESS_ERROR was added in 8.0.11.

- Error number: MY-011489; Symbol: ER_GRP_RPL_SRV_NOT_ONLINE; SQLSTATE: HY000

  Message: This server was not declared online since it is on status %s.

  ER_GRP_RPL_SRV_NOT_ONLINE was added in 8.0.11.

- Error number: MY-011490; Symbol: ER_GRP_RPL_SRV_ONLINE; SQLSTATE: HY000

  Message: This server was declared online within the replication group.

  ER_GRP_RPL_SRV_ONLINE was added in 8.0.11.

- Error number: MY-011491; Symbol: ER_GRP_RPL_DISABLE_SRV_READ_MODE_RESTRICTED; SQLSTATE: HY000

  Message: When declaring the plugin online it was not possible to disable the server read mode settings. Try to disable it manually.

  ER_GRP_RPL_DISABLE_SRV_READ_MODE_RESTRICTED was added in 8.0.11.

- Error number: MY-011492; Symbol: ER_GRP_RPL_MEM_ONLINE; SQLSTATE: HY000

  Message: The member with address %s:%u was declared online within the replication group.

  ER_GRP_RPL_MEM_ONLINE was added in 8.0.11.

- Error number: MY-011493; Symbol: ER_GRP_RPL_MEM_UNREACHABLE; SQLSTATE: HY000

  Message: Member with address %s:%u has become unreachable.

ER_GRP_RPL_MEM_UNREACHABLE was added in 8.0.11.

- Error number: MY-011494; Symbol: ER_GRP_RPL_MEM_REACHABLE; SQLSTATE: HY000

  Message: Member with address %s:%u is reachable again.

  ER_GRP_RPL_MEM_REACHABLE was added in 8.0.11.

- Error number: MY-011495; Symbol: ER_GRP_RPL_SRV_BLOCKED; SQLSTATE: HY000

  Message: This server is not able to reach a majority of members in the group. This server will now block all updates. The server will remain blocked until contact with the majority is restored. It is possible to use group_replication_force_members to force a new group membership.

  ER_GRP_RPL_SRV_BLOCKED was added in 8.0.11.

- Error number: MY-011496; Symbol: ER_GRP_RPL_SRV_BLOCKED_FOR_SECS; SQLSTATE: HY000

  Message: This server is not able to reach a majority of members in the group. This server will now block all updates. The server will remain blocked for the next %lu seconds. Unless contact with the majority is restored, after this time the member will error out and leave the group. It is possible to use group_replication_force_members to force a new group membership.

  ER_GRP_RPL_SRV_BLOCKED_FOR_SECS was added in 8.0.11.

- Error number: MY-011497; Symbol: ER_GRP_RPL_CHANGE_GRP_MEM_NOT_PROCESSED; SQLSTATE: HY000

  Message: A group membership change was received but the plugin is already leaving due to the configured timeout on group_replication_unreachable_majority_timeout option.

  ER_GRP_RPL_CHANGE_GRP_MEM_NOT_PROCESSED was added in 8.0.11.

- Error number: MY-011498; Symbol: ER_GRP_RPL_MEMBER_CONTACT_RESTORED; SQLSTATE: HY000

  Message: The member has resumed contact with a majority of the members in the group. Regular operation is restored and transactions are unblocked.

  ER_GRP_RPL_MEMBER_CONTACT_RESTORED was added in 8.0.11.

- Error number: MY-011499; Symbol: ER_GRP_RPL_MEMBER_REMOVED; SQLSTATE: HY000

  Message: Members removed from the group: %s

  ER_GRP_RPL_MEMBER_REMOVED was added in 8.0.11.

- Error number: MY-011500; Symbol: ER_GRP_RPL_PRIMARY_MEMBER_LEFT_GRP; SQLSTATE: HY000

  Message: Primary server with address %s left the group. Electing new Primary.

  ER_GRP_RPL_PRIMARY_MEMBER_LEFT_GRP was added in 8.0.11.

- Error number: MY-011501; Symbol: ER_GRP_RPL_MEMBER_ADDED; SQLSTATE: HY000

  Message: Members joined the group: %s

ER_GRP_RPL_MEMBER_ADDED was added in 8.0.11.

- Error number: MY-011502; Symbol: ER_GRP_RPL_MEMBER_EXIT_PLUGIN_ERROR; SQLSTATE: HY000

  Message: There was a previous plugin error while the member joined the group. The member will now exit the group.

  ER_GRP_RPL_MEMBER_EXIT_PLUGIN_ERROR was added in 8.0.11.

- Error number: MY-011503; Symbol: ER_GRP_RPL_MEMBER_CHANGE; SQLSTATE: HY000

  Message: Group membership changed to %s on view %s.

  ER_GRP_RPL_MEMBER_CHANGE was added in 8.0.11.

- Error number: MY-011504; Symbol: ER_GRP_RPL_MEMBER_LEFT_GRP; SQLSTATE: HY000

  Message: Group membership changed: This member has left the group.

  ER_GRP_RPL_MEMBER_LEFT_GRP was added in 8.0.11.

- Error number: MY-011505; Symbol: ER_GRP_RPL_MEMBER_EXPELLED; SQLSTATE: HY000

  Message: Member was expelled from the group due to network failures, changing member status to ERROR.

  ER_GRP_RPL_MEMBER_EXPELLED was added in 8.0.11.

- Error number: MY-011506; Symbol: ER_GRP_RPL_SESSION_OPEN_FAILED; SQLSTATE: HY000

  Message: Unable to open session to (re)set read only mode. Skipping.

  ER_GRP_RPL_SESSION_OPEN_FAILED was added in 8.0.11.

- Error number: MY-011507; Symbol: ER_GRP_RPL_NEW_PRIMARY_ELECTED; SQLSTATE: HY000

  Message: A new primary with address %s:%u was elected. %s

  ER_GRP_RPL_NEW_PRIMARY_ELECTED was added in 8.0.11.

- Error number: MY-011508; Symbol: ER_GRP_RPL_DISABLE_READ_ONLY_FAILED; SQLSTATE: HY000

  Message: Unable to disable super read only flag. Try to disable it manually

  ER_GRP_RPL_DISABLE_READ_ONLY_FAILED was added in 8.0.11.

- Error number: MY-011509; Symbol: ER_GRP_RPL_ENABLE_READ_ONLY_FAILED; SQLSTATE: HY000

  Message: Unable to set super read only flag. Try to set it manually.

  ER_GRP_RPL_ENABLE_READ_ONLY_FAILED was added in 8.0.11.

- Error number: MY-011510; Symbol: ER_GRP_RPL_SRV_PRIMARY_MEM; SQLSTATE: HY000

  Message: This server is working as primary member.

ER_GRP_RPL_SRV_PRIMARY_MEM was added in 8.0.11.

- Error number: MY-011511; Symbol: ER_GRP_RPL_SRV_SECONDARY_MEM; SQLSTATE: HY000

  Message: This server is working as secondary member with primary member address %s:%u.

  ER_GRP_RPL_SRV_SECONDARY_MEM was added in 8.0.11.

- Error number: MY-011512; Symbol: ER_GRP_RPL_NO_SUITABLE_PRIMARY_MEM; SQLSTATE: HY000

  Message: Unable to set any member as primary. No suitable candidate.

  ER_GRP_RPL_NO_SUITABLE_PRIMARY_MEM was added in 8.0.11.

- Error number: MY-011513; Symbol: ER_GRP_RPL_SUPER_READ_ONLY_ACTIVATE_ERROR; SQLSTATE: HY000

  Message: Error when activating super_read_only mode on start. The member will now exit the group.

  ER_GRP_RPL_SUPER_READ_ONLY_ACTIVATE_ERROR was added in 8.0.11.

- Error number: MY-011514; Symbol: ER_GRP_RPL_EXCEEDS_AUTO_INC_VALUE; SQLSTATE: HY000

  Message: Group contains %lu members which is greater than auto_increment_increment value of %lu. This can lead to a higher transactional abort rate.

  ER_GRP_RPL_EXCEEDS_AUTO_INC_VALUE was added in 8.0.11.

- Error number: MY-011515; Symbol: ER_GRP_RPL_DATA_NOT_PROVIDED_BY_MEM; SQLSTATE: HY000

  Message: Member with address '%s:%u' didn't provide any data during the last group change. Group information can be outdated and lead to errors on recovery.

  ER_GRP_RPL_DATA_NOT_PROVIDED_BY_MEM was added in 8.0.11.

- Error number: MY-011516; Symbol: ER_GRP_RPL_MEMBER_ALREADY_EXISTS; SQLSTATE: HY000

  Message: There is already a member with server_uuid %s. The member will now exit the group.

  ER_GRP_RPL_MEMBER_ALREADY_EXISTS was added in 8.0.11.

- Error number: MY-011517; Symbol: ER_GRP_RPL_GRP_CHANGE_INFO_EXTRACT_ERROR; SQLSTATE: HY000

  Message: Error when extracting information for group change. Operations and checks made to group joiners may be incomplete.

  ER_GRP_RPL_GRP_CHANGE_INFO_EXTRACT_ERROR was added in 8.0.11.

- Error number: MY-011518; Symbol: ER_GRP_RPL_GTID_EXECUTED_EXTRACT_ERROR; SQLSTATE: HY000

  Message: Error when extracting this member GTID executed set. Operations and checks made to group joiners may be incomplete.

  ER_GRP_RPL_GTID_EXECUTED_EXTRACT_ERROR was added in 8.0.11.

- Error number: MY-011519; Symbol: ER_GRP_RPL_GTID_SET_EXTRACT_ERROR; SQLSTATE: HY000

  Message: Error when extracting this member retrieved set for its applier. Operations and checks made to group joiners may be incomplete.

  ER_GRP_RPL_GTID_SET_EXTRACT_ERROR was added in 8.0.11.

- Error number: MY-011520; Symbol: ER_GRP_RPL_START_FAILED; SQLSTATE: HY000

  Message: The START GROUP_REPLICATION command failed since the group already has 9 members.

  ER_GRP_RPL_START_FAILED was added in 8.0.11.

- Error number: MY-011521; Symbol: ER_GRP_RPL_MEMBER_VER_INCOMPATIBLE; SQLSTATE: HY000

  Message: Member version is incompatible with the group.

  ER_GRP_RPL_MEMBER_VER_INCOMPATIBLE was added in 8.0.11.

- Error number: MY-011522; Symbol: ER_GRP_RPL_TRANS_NOT_PRESENT_IN_GRP; SQLSTATE: HY000

  Message: The member contains transactions not present in the group. The member will now exit the group.

  ER_GRP_RPL_TRANS_NOT_PRESENT_IN_GRP was added in 8.0.11.

- Error number: MY-011523; Symbol: ER_GRP_RPL_TRANS_GREATER_THAN_GRP; SQLSTATE: HY000

  Message: It was not possible to assess if the member has more transactions than the group. The member will now exit the group.

  ER_GRP_RPL_TRANS_GREATER_THAN_GRP was added in 8.0.11.

- Error number: MY-011524; Symbol: ER_GRP_RPL_MEMBER_VERSION_LOWER_THAN_GRP; SQLSTATE: HY000

  Message: Member version is lower than some group member, but since option 'group_replication_allow_local_lower_version_join is enabled, member will be allowed to join.

  ER_GRP_RPL_MEMBER_VERSION_LOWER_THAN_GRP was added in 8.0.11.

- Error number: MY-011525; Symbol: ER_GRP_RPL_LOCAL_GTID_SETS_PROCESS_ERROR; SQLSTATE: HY000

  Message: Error processing local GTID sets when comparing this member transactions against the group.

  ER_GRP_RPL_LOCAL_GTID_SETS_PROCESS_ERROR was added in 8.0.11.

- Error number: MY-011526; Symbol: ER_GRP_RPL_MEMBER_TRANS_GREATER_THAN_GRP; SQLSTATE: HY000

  Message: This member has more executed transactions than those present in the group. Local transactions: %s > Group transactions: %s

  ER_GRP_RPL_MEMBER_TRANS_GREATER_THAN_GRP was added in 8.0.11.

- Error number: MY-011527; Symbol: ER_GRP_RPL_BLOCK_SIZE_DIFF_FROM_GRP; SQLSTATE: HY000

  Message: The member is configured with a group_replication_gtid_assignment_block_size option value '%llu' different from the group '%llu'. The member will now exit the group.

  ER_GRP_RPL_BLOCK_SIZE_DIFF_FROM_GRP was added in 8.0.11.

- Error number: MY-011528; Symbol: ER_GRP_RPL_TRANS_WRITE_SET_EXTRACT_DIFF_FROM_GRP; SQLSTATE: HY000

  Message: The member is configured with a transaction-write-set-extraction option value '%s' different from the group '%s'. The member will now exit the group.

  ER_GRP_RPL_TRANS_WRITE_SET_EXTRACT_DIFF_FROM_GRP was added in 8.0.11.

- Error number: MY-011529; Symbol: ER_GRP_RPL_MEMBER_CFG_INCOMPATIBLE_WITH_GRP_CFG; SQLSTATE: HY000

  Message: The member configuration is not compatible with the group configuration. Variables such as single_primary_mode or enforce_update_everywhere_checks must have the same value on every server in the group. (member configuration option: [%s], group configuration option: [%s]).

  ER_GRP_RPL_MEMBER_CFG_INCOMPATIBLE_WITH_GRP_CFG was added in 8.0.11.

- Error number: MY-011530; Symbol: ER_GRP_RPL_MEMBER_STOP_RPL_CHANNELS_ERROR; SQLSTATE: HY000

  Message: Error stopping all replication channels while server was leaving the group. %s

  ER_GRP_RPL_MEMBER_STOP_RPL_CHANNELS_ERROR was added in 8.0.11.

- Error number: MY-011531; Symbol: ER_GRP_RPL_PURGE_APPLIER_LOGS; SQLSTATE: HY000

  Message: Detected previous RESET MASTER invocation or an issue exists in the group replication applier relay log. Purging existing applier logs.

  ER_GRP_RPL_PURGE_APPLIER_LOGS was added in 8.0.11.

- Error number: MY-011532; Symbol: ER_GRP_RPL_RESET_APPLIER_MODULE_LOGS_ERROR; SQLSTATE: HY000

  Message: Unknown error occurred while resetting applier's module logs.

  ER_GRP_RPL_RESET_APPLIER_MODULE_LOGS_ERROR was added in 8.0.11.

- Error number: MY-011533; Symbol: ER_GRP_RPL_APPLIER_THD_SETUP_ERROR; SQLSTATE: HY000

  Message: Failed to setup the group replication applier thread.

  ER_GRP_RPL_APPLIER_THD_SETUP_ERROR was added in 8.0.11.

- Error number: MY-011534; Symbol: ER_GRP_RPL_APPLIER_THD_START_ERROR; SQLSTATE: HY000

  Message: Error while starting the group replication applier thread

  ER_GRP_RPL_APPLIER_THD_START_ERROR was added in 8.0.11.

- Error number: MY-011535; Symbol: ER_GRP_RPL_APPLIER_THD_STOP_ERROR; SQLSTATE: HY000

  Message: Failed to stop the group replication applier thread.

  ER_GRP_RPL_APPLIER_THD_STOP_ERROR was added in 8.0.11.

- Error number: MY-011536; Symbol: ER_GRP_RPL_FETCH_TRANS_DATA_FAILED; SQLSTATE: HY000

  Message: Failed to fetch transaction data containing required transaction info for applier

  ER_GRP_RPL_FETCH_TRANS_DATA_FAILED was added in 8.0.11.

- Error number: MY-011537; Symbol: ER_GRP_RPL_SLAVE_IO_THD_PRIMARY_UNKNOWN; SQLSTATE: HY000

  Message: Can't start slave IO THREAD of channel '%s' when group replication is running with single-primary mode and the primary member is not known.

  ER_GRP_RPL_SLAVE_IO_THD_PRIMARY_UNKNOWN was added in 8.0.11.

- Error number: MY-011538; Symbol: ER_GRP_RPL_SALVE_IO_THD_ON_SECONDARY_MEMBER; SQLSTATE: HY000

  Message: Can't start slave IO THREAD of channel '%s' when group replication is running with single-primary mode on a secondary member.

  ER_GRP_RPL_SALVE_IO_THD_ON_SECONDARY_MEMBER was added in 8.0.11.

- Error number: MY-011539; Symbol: ER_GRP_RPL_SLAVE_SQL_THD_PRIMARY_UNKNOWN; SQLSTATE: HY000

  Message: Can't start slave SQL THREAD of channel '%s' when group replication is running with single-primary mode and the primary member is not known.

  ER_GRP_RPL_SLAVE_SQL_THD_PRIMARY_UNKNOWN was added in 8.0.11.

- Error number: MY-011540; Symbol: ER_GRP_RPL_SLAVE_SQL_THD_ON_SECONDARY_MEMBER; SQLSTATE: HY000

  Message: Can't start slave SQL THREAD of channel '%s' when group replication is running with single-primary mode on a secondary member.

  ER_GRP_RPL_SLAVE_SQL_THD_ON_SECONDARY_MEMBER was added in 8.0.11.

- Error number: MY-011541; Symbol: ER_GRP_RPL_NEEDS_INNODB_TABLE; SQLSTATE: HY000

  Message: Table %s does not use the InnoDB storage engine. This is not compatible with Group Replication.

  ER_GRP_RPL_NEEDS_INNODB_TABLE was added in 8.0.11.

- Error number: MY-011542; Symbol: ER_GRP_RPL_PRIMARY_KEY_NOT_DEFINED; SQLSTATE: HY000

  Message: Table %s does not have any PRIMARY KEY. This is not compatible with Group Replication.

  ER_GRP_RPL_PRIMARY_KEY_NOT_DEFINED was added in 8.0.11.

- Error number: MY-011543; Symbol: ER_GRP_RPL_FK_WITH_CASCADE_UNSUPPORTED; SQLSTATE: HY000

  Message: Table %s has a foreign key with 'CASCADE', 'SET NULL' or 'SET DEFAULT' clause. This is not compatible with Group Replication.

  ER_GRP_RPL_FK_WITH_CASCADE_UNSUPPORTED was added in 8.0.11.

- Error number: MY-011544; Symbol: ER_GRP_RPL_AUTO_INC_RESET; SQLSTATE: HY000

  Message: auto_increment_increment is reset to %lu

  ER_GRP_RPL_AUTO_INC_RESET was added in 8.0.11.

- Error number: MY-011545; Symbol: ER_GRP_RPL_AUTO_INC_OFFSET_RESET; SQLSTATE: HY000

  Message: auto_increment_offset is reset to %lu

  ER_GRP_RPL_AUTO_INC_OFFSET_RESET was added in 8.0.11.

- Error number: MY-011546; Symbol: ER_GRP_RPL_AUTO_INC_SET; SQLSTATE: HY000

  Message: auto_increment_increment is set to %lu

  ER_GRP_RPL_AUTO_INC_SET was added in 8.0.11.

- Error number: MY-011547; Symbol: ER_GRP_RPL_AUTO_INC_OFFSET_SET; SQLSTATE: HY000

  Message: auto_increment_offset is set to %lu

  ER_GRP_RPL_AUTO_INC_OFFSET_SET was added in 8.0.11.

- Error number: MY-011548; Symbol: ER_GRP_RPL_FETCH_TRANS_CONTEXT_FAILED; SQLSTATE: HY000

  Message: Failed to fetch transaction context containing required transaction info for certification

  ER_GRP_RPL_FETCH_TRANS_CONTEXT_FAILED was added in 8.0.11.

- Error number: MY-011549; Symbol: ER_GRP_RPL_FETCH_FORMAT_DESC_LOG_EVENT_FAILED; SQLSTATE: HY000

  Message: Failed to fetch Format_description_log_event containing required server info for applier

  ER_GRP_RPL_FETCH_FORMAT_DESC_LOG_EVENT_FAILED was added in 8.0.11.

- Error number: MY-011550; Symbol: ER_GRP_RPL_FETCH_TRANS_CONTEXT_LOG_EVENT_FAILED; SQLSTATE: HY000

  Message: Failed to fetch Transaction_context_log_event containing required transaction info for certification

  ER_GRP_RPL_FETCH_TRANS_CONTEXT_LOG_EVENT_FAILED was added in 8.0.11.

- Error number: MY-011551; Symbol: ER_GRP_RPL_FETCH_SNAPSHOT_VERSION_FAILED; SQLSTATE: HY000

  Message: Failed to read snapshot version from transaction context event required for certification

ER_GRP_RPL_FETCH_SNAPSHOT_VERSION_FAILED was added in 8.0.11.

- Error number: MY-011552; Symbol: ER_GRP_RPL_FETCH_GTID_LOG_EVENT_FAILED; SQLSTATE: HY000

  Message: Failed to fetch Gtid_log_event containing required transaction info for certification

  ER_GRP_RPL_FETCH_GTID_LOG_EVENT_FAILED was added in 8.0.11.

- Error number: MY-011553; Symbol: ER_GRP_RPL_UPDATE_SERV_CERTIFICATE_FAILED; SQLSTATE: HY000

  Message: Unable to update certification result on server side, thread_id: %lu

  ER_GRP_RPL_UPDATE_SERV_CERTIFICATE_FAILED was added in 8.0.11.

- Error number: MY-011554; Symbol: ER_GRP_RPL_ADD_GTID_INFO_WITH_LOCAL_GTID_FAILED; SQLSTATE: HY000

  Message: Unable to add gtid information to the group_gtid_executed set when gtid was provided for local transactions

  ER_GRP_RPL_ADD_GTID_INFO_WITH_LOCAL_GTID_FAILED was added in 8.0.11.

- Error number: MY-011555; Symbol: ER_GRP_RPL_ADD_GTID_INFO_WITHOUT_LOCAL_GTID_FAILED; SQLSTATE: HY000

  Message: Unable to add gtid information to the group_gtid_executed set when no gtid was provided for local transactions

  ER_GRP_RPL_ADD_GTID_INFO_WITHOUT_LOCAL_GTID_FAILED was added in 8.0.11.

- Error number: MY-011556; Symbol: ER_GRP_RPL_NOTIFY_CERTIFICATION_OUTCOME_FAILED; SQLSTATE: HY000

  Message: Failed to notify certification outcome

  ER_GRP_RPL_NOTIFY_CERTIFICATION_OUTCOME_FAILED was added in 8.0.11.

- Error number: MY-011557; Symbol: ER_GRP_RPL_ADD_GTID_INFO_WITH_REMOTE_GTID_FAILED; SQLSTATE: HY000

  Message: Unable to add gtid information to the group_gtid_executed set when gtid was provided for remote transactions

  ER_GRP_RPL_ADD_GTID_INFO_WITH_REMOTE_GTID_FAILED was added in 8.0.11.

- Error number: MY-011558; Symbol: ER_GRP_RPL_ADD_GTID_INFO_WITHOUT_REMOTE_GTID_FAILED; SQLSTATE: HY000

  Message: Unable to add gtid information to the group_gtid_executed set when gtid was not provided for remote transactions

  ER_GRP_RPL_ADD_GTID_INFO_WITHOUT_REMOTE_GTID_FAILED was added in 8.0.11.

- Error number: MY-011559; Symbol: ER_GRP_RPL_FETCH_VIEW_CHANGE_LOG_EVENT_FAILED; SQLSTATE: HY000

Message: Failed to fetch View_change_log_event containing required info for certification

ER_GRP_RPL_FETCH_VIEW_CHANGE_LOG_EVENT_FAILED was added in 8.0.11.

- Error number: MY-011560; Symbol: ER_GRP_RPL_CONTACT_WITH_SRV_FAILED; SQLSTATE: HY000

  Message: Error when contacting the server to ensure the proper logging of a group change in the binlog

  ER_GRP_RPL_CONTACT_WITH_SRV_FAILED was added in 8.0.11.

- Error number: MY-011561; Symbol: ER_GRP_RPL_SRV_WAIT_TIME_OUT; SQLSTATE: HY000

  Message: Timeout when waiting for the server to execute local transactions in order assure the group change proper logging

  ER_GRP_RPL_SRV_WAIT_TIME_OUT was added in 8.0.11.

- Error number: MY-011562; Symbol: ER_GRP_RPL_FETCH_LOG_EVENT_FAILED; SQLSTATE: HY000

  Message: Failed to fetch Log_event containing required server info for applier

  ER_GRP_RPL_FETCH_LOG_EVENT_FAILED was added in 8.0.11.

- Error number: MY-011563; Symbol: ER_GRP_RPL_START_GRP_RPL_FAILED; SQLSTATE: HY000

  Message: Unable to start Group Replication. Replication applier infrastructure is not initialized since the server was started with --initialize, --initialize-insecure or --upgrade=MINIMAL on a server upgrade.

  ER_GRP_RPL_START_GRP_RPL_FAILED was added in 8.0.11.

- Error number: MY-011564; Symbol: ER_GRP_RPL_CONN_INTERNAL_PLUGIN_FAIL; SQLSTATE: HY000

  Message: Failed to establish an internal server connection to execute plugin operations

  ER_GRP_RPL_CONN_INTERNAL_PLUGIN_FAIL was added in 8.0.11.

- Error number: MY-011565; Symbol: ER_GRP_RPL_SUPER_READ_ON; SQLSTATE: HY000

  Message: Setting super_read_only=ON.

  ER_GRP_RPL_SUPER_READ_ON was added in 8.0.11.

- Error number: MY-011566; Symbol: ER_GRP_RPL_SUPER_READ_OFF; SQLSTATE: HY000

  Message: Setting super_read_only=OFF.

  ER_GRP_RPL_SUPER_READ_OFF was added in 8.0.11.

- Error number: MY-011567; Symbol: ER_GRP_RPL_KILLED_SESSION_ID; SQLSTATE: HY000

  Message: killed session id: %d status: %d

  ER_GRP_RPL_KILLED_SESSION_ID was added in 8.0.11.

- Error number: MY-011568; Symbol: ER_GRP_RPL_KILLED_FAILED_ID; SQLSTATE: HY000

  Message: killed failed id: %d failed: %d

ER_GRP_RPL_KILLED_FAILED_ID was added in 8.0.11.

- Error number: MY-011569; Symbol: ER_GRP_RPL_INTERNAL_QUERY; SQLSTATE: HY000

  Message: Internal query: %s result in error. Error number: %ld

  ER_GRP_RPL_INTERNAL_QUERY was added in 8.0.11.

- Error number: MY-011570; Symbol: ER_GRP_RPL_COPY_FROM_EMPTY_STRING; SQLSTATE: HY000

  Message: Error copying from empty string

  ER_GRP_RPL_COPY_FROM_EMPTY_STRING was added in 8.0.11.

- Error number: MY-011571; Symbol: ER_GRP_RPL_QUERY_FAIL; SQLSTATE: HY000

  Message: Query execution resulted in failure. errno: %d

  ER_GRP_RPL_QUERY_FAIL was added in 8.0.11.

- Error number: MY-011572; Symbol: ER_GRP_RPL_CREATE_SESSION_UNABLE; SQLSTATE: HY000

  Message: Unable to create a session for executing the queries on the server

  ER_GRP_RPL_CREATE_SESSION_UNABLE was added in 8.0.11.

- Error number: MY-011573; Symbol: ER_GRP_RPL_MEMBER_NOT_FOUND; SQLSTATE: HY000

  Message: The member with address %s:%u has unexpectedly disappeared, killing the current group replication recovery connection

  ER_GRP_RPL_MEMBER_NOT_FOUND was added in 8.0.11.

- Error number: MY-011574; Symbol: ER_GRP_RPL_MAXIMUM_CONNECTION_RETRIES_REACHED; SQLSTATE: HY000

  Message: Maximum number of retries when trying to connect to a donor reached. Aborting group replication incremental recovery.

  ER_GRP_RPL_MAXIMUM_CONNECTION_RETRIES_REACHED was added in 8.0.11.

- Error number: MY-011575; Symbol: ER_GRP_RPL_ALL_DONORS_LEFT_ABORT_RECOVERY; SQLSTATE: HY000

  Message: All donors left. Aborting group replication incremental recovery.

  ER_GRP_RPL_ALL_DONORS_LEFT_ABORT_RECOVERY was added in 8.0.11.

- Error number: MY-011576; Symbol: ER_GRP_RPL_ESTABLISH_RECOVERY_WITH_DONOR; SQLSTATE: HY000

  Message: Establishing group recovery connection with a possible donor. Attempt %d/%d

  ER_GRP_RPL_ESTABLISH_RECOVERY_WITH_DONOR was added in 8.0.11.

- Error number: MY-011577; Symbol: ER_GRP_RPL_ESTABLISH_RECOVERY_WITH_ANOTHER_DONOR; SQLSTATE: HY000

  Message: Retrying group recovery connection with another donor. Attempt %d/%d

ER_GRP_RPL_ESTABLISH_RECOVERY_WITH_ANOTHER_DONOR was added in 8.0.11.

- Error number: MY-011578; Symbol: ER_GRP_RPL_NO_VALID_DONOR; SQLSTATE: HY000

  Message: No valid donors exist in the group, retrying

  ER_GRP_RPL_NO_VALID_DONOR was added in 8.0.11.

- Error number: MY-011579; Symbol: ER_GRP_RPL_CONFIG_RECOVERY; SQLSTATE: HY000

  Message: Error when configuring the asynchronous recovery channel connection to the donor.

  ER_GRP_RPL_CONFIG_RECOVERY was added in 8.0.11.

- Error number: MY-011580; Symbol: ER_GRP_RPL_ESTABLISHING_CONN_GRP_REC_DONOR; SQLSTATE: HY000

  Message: Establishing connection to a group replication recovery donor %s at %s port: %d.

  ER_GRP_RPL_ESTABLISHING_CONN_GRP_REC_DONOR was added in 8.0.11.

- Error number: MY-011581; Symbol: ER_GRP_RPL_CREATE_GRP_RPL_REC_CHANNEL; SQLSTATE: HY000

  Message: Error while creating the group replication recovery channel with donor %s at %s port: %d.

  ER_GRP_RPL_CREATE_GRP_RPL_REC_CHANNEL was added in 8.0.11.

- Error number: MY-011582; Symbol: ER_GRP_RPL_DONOR_SERVER_CONN; SQLSTATE: HY000

  Message: There was an error when connecting to the donor server. Please check that group_replication_recovery channel credentials and all MEMBER_HOST column values of performance_schema.replication_group_members table are correct and DNS resolvable.

  ER_GRP_RPL_DONOR_SERVER_CONN was added in 8.0.11.

- Error number: MY-011583; Symbol: ER_GRP_RPL_CHECK_STATUS_TABLE; SQLSTATE: HY000

  Message: For details please check performance_schema.replication_connection_status table and error log messages of Slave I/O for channel group_replication_recovery.

  ER_GRP_RPL_CHECK_STATUS_TABLE was added in 8.0.11.

- Error number: MY-011584; Symbol: ER_GRP_RPL_STARTING_GRP_REC; SQLSTATE: HY000

  Message: Error while starting the group replication incremental recovery receiver/applier threads

  ER_GRP_RPL_STARTING_GRP_REC was added in 8.0.11.

- Error number: MY-011585; Symbol: ER_GRP_RPL_DONOR_CONN_TERMINATION; SQLSTATE: HY000

  Message: Terminating existing group replication donor connection and purging the corresponding logs.

  ER_GRP_RPL_DONOR_CONN_TERMINATION was added in 8.0.11.

- Error number: MY-011586; Symbol: ER_GRP_RPL_STOPPING_GRP_REC; SQLSTATE: HY000

  Message: Error when stopping the group replication incremental recovery's donor connection

ER_GRP_RPL_STOPPING_GRP_REC was added in 8.0.11.

- Error number: MY-011587; Symbol: ER_GRP_RPL_PURGE_REC; SQLSTATE: HY000

  Message: Error when purging the group replication recovery's relay logs

  ER_GRP_RPL_PURGE_REC was added in 8.0.11.

- Error number: MY-011588; Symbol: ER_GRP_RPL_UNABLE_TO_KILL_CONN_REC_DONOR_APPLIER; SQLSTATE: HY000

  Message: Unable to kill the current group replication recovery donor connection after an applier error. Incremental recovery will shutdown.

  ER_GRP_RPL_UNABLE_TO_KILL_CONN_REC_DONOR_APPLIER was added in 8.0.11.

- Error number: MY-011589; Symbol: ER_GRP_RPL_UNABLE_TO_KILL_CONN_REC_DONOR_FAILOVER; SQLSTATE: HY000

  Message: Unable to kill the current group replication recovery donor connection during failover. Incremental recovery will shutdown.

  ER_GRP_RPL_UNABLE_TO_KILL_CONN_REC_DONOR_FAILOVER was added in 8.0.11.

- Error number: MY-011590; Symbol: ER_GRP_RPL_FAILED_TO_NOTIFY_GRP_MEMBERSHIP_EVENT; SQLSTATE: HY000

  Message: Unexpected error when notifying an internal component named %s regarding a group membership event.

  ER_GRP_RPL_FAILED_TO_NOTIFY_GRP_MEMBERSHIP_EVENT was added in 8.0.11.

- Error number: MY-011591; Symbol: ER_GRP_RPL_FAILED_TO_BROADCAST_GRP_MEMBERSHIP_NOTIFICATION; SQLSTATE: HY000

  Message: An undefined error was found while broadcasting an internal group membership notification! This is likely to happen if your components or plugins are not properly loaded or are malfunctioning!

  ER_GRP_RPL_FAILED_TO_BROADCAST_GRP_MEMBERSHIP_NOTIFICATION was added in 8.0.11.

- Error number: MY-011592; Symbol: ER_GRP_RPL_FAILED_TO_BROADCAST_MEMBER_STATUS_NOTIFICATION; SQLSTATE: HY000

  Message: An undefined error was found while broadcasting an internal group member status notification! This is likely to happen if your components or plugins are not properly loaded or are malfunctioning!

  ER_GRP_RPL_FAILED_TO_BROADCAST_MEMBER_STATUS_NOTIFICATION was added in 8.0.11.

- Error number: MY-011593; Symbol: ER_GRP_RPL_OOM_FAILED_TO_GENERATE_IDENTIFICATION_HASH; SQLSTATE: HY000

  Message: No memory to generate write identification hash

  ER_GRP_RPL_OOM_FAILED_TO_GENERATE_IDENTIFICATION_HASH was added in 8.0.11.

- Error number: MY-011594; Symbol: ER_GRP_RPL_WRITE_IDENT_HASH_BASE64_ENCODING_FAILED; SQLSTATE: HY000

Message: Base 64 encoding of the write identification hash failed

ER_GRP_RPL_WRITE_IDENT_HASH_BASE64_ENCODING_FAILED was added in 8.0.11.

- Error number: MY-011595; Symbol: ER_GRP_RPL_INVALID_BINLOG_FORMAT; SQLSTATE: HY000

  Message: Binlog format should be ROW for Group Replication

  ER_GRP_RPL_INVALID_BINLOG_FORMAT was added in 8.0.11.

- Error number: MY-011596; Symbol: ER_GRP_RPL_BINLOG_CHECKSUM_SET; SQLSTATE: HY000

  Message: binlog_checksum should be NONE for Group Replication

  ER_GRP_RPL_BINLOG_CHECKSUM_SET was added in 8.0.11.

- Error number: MY-011597; Symbol: ER_GRP_RPL_TRANS_WRITE_SET_EXTRACTION_NOT_SET; SQLSTATE: HY000

  Message: A transaction_write_set_extraction algorithm should be selected when running Group Replication

  ER_GRP_RPL_TRANS_WRITE_SET_EXTRACTION_NOT_SET was added in 8.0.11.

- Error number: MY-011598; Symbol: ER_GRP_RPL_UNSUPPORTED_TRANS_ISOLATION; SQLSTATE: HY000

  Message: Transaction isolation level (tx_isolation) is set to SERIALIZABLE, which is not compatible with Group Replication

  ER_GRP_RPL_UNSUPPORTED_TRANS_ISOLATION was added in 8.0.11.

- Error number: MY-011599; Symbol: ER_GRP_RPL_CANNOT_EXECUTE_TRANS_WHILE_STOPPING; SQLSTATE: HY000

  Message: Transaction cannot be executed while Group Replication is stopping.

  ER_GRP_RPL_CANNOT_EXECUTE_TRANS_WHILE_STOPPING was added in 8.0.11.

- Error number: MY-011600; Symbol: ER_GRP_RPL_CANNOT_EXECUTE_TRANS_WHILE_RECOVERING; SQLSTATE: HY000

  Message: Transaction cannot be executed while Group Replication is recovering. Try again when the server is ONLINE.

  ER_GRP_RPL_CANNOT_EXECUTE_TRANS_WHILE_RECOVERING was added in 8.0.11.

- Error number: MY-011601; Symbol: ER_GRP_RPL_CANNOT_EXECUTE_TRANS_IN_ERROR_STATE; SQLSTATE: HY000

  Message: Transaction cannot be executed while Group Replication is on ERROR state. Check for errors and restart the plugin

  ER_GRP_RPL_CANNOT_EXECUTE_TRANS_IN_ERROR_STATE was added in 8.0.11.

- Error number: MY-011602; Symbol: ER_GRP_RPL_CANNOT_EXECUTE_TRANS_IN_OFFLINE_MODE; SQLSTATE: HY000

Message: Transaction cannot be executed while Group Replication is OFFLINE. Check for errors and restart the plugin

`ER_GRP_RPL_CANNOT_EXECUTE_TRANS_IN_OFFLINE_MODE` was added in 8.0.11.

- Error number: `MY-011603`; Symbol: `ER_GRP_RPL_MULTIPLE_CACHE_TYPE_NOT_SUPPORTED_FOR_SESSION`; SQLSTATE: `HY000`

  Message: We can only use one cache type at a time on session %u

  `ER_GRP_RPL_MULTIPLE_CACHE_TYPE_NOT_SUPPORTED_FOR_SESSION` was added in 8.0.11.

- Error number: `MY-011604`; Symbol: `ER_GRP_RPL_FAILED_TO_REINIT_BINLOG_CACHE_FOR_READ`; SQLSTATE: `HY000`

  Message: Failed to reinit binlog cache log for read on session %u

  `ER_GRP_RPL_FAILED_TO_REINIT_BINLOG_CACHE_FOR_READ` was added in 8.0.11.

- Error number: `MY-011605`; Symbol: `ER_GRP_RPL_FAILED_TO_CREATE_TRANS_CONTEXT`; SQLSTATE: `HY000`

  Message: Failed to create the context of the current transaction on session %u

  `ER_GRP_RPL_FAILED_TO_CREATE_TRANS_CONTEXT` was added in 8.0.11.

- Error number: `MY-011606`; Symbol: `ER_GRP_RPL_FAILED_TO_EXTRACT_TRANS_WRITE_SET`; SQLSTATE: `HY000`

  Message: Failed to extract the set of items written during the execution of the current transaction on session %u

  `ER_GRP_RPL_FAILED_TO_EXTRACT_TRANS_WRITE_SET` was added in 8.0.11.

- Error number: `MY-011607`; Symbol: `ER_GRP_RPL_FAILED_TO_GATHER_TRANS_WRITE_SET`; SQLSTATE: `HY000`

  Message: Failed to gather the set of items written during the execution of the current transaction on session %u

  `ER_GRP_RPL_FAILED_TO_GATHER_TRANS_WRITE_SET` was added in 8.0.11.

- Error number: `MY-011608`; Symbol: `ER_GRP_RPL_TRANS_SIZE_EXCEEDS_LIMIT`; SQLSTATE: `HY000`

  Message: Error on session %u. Transaction of size %llu exceeds specified limit %lu. To increase the limit please adjust group_replication_transaction_size_limit option.

  `ER_GRP_RPL_TRANS_SIZE_EXCEEDS_LIMIT` was added in 8.0.11.

- Error number: `MY-011609`; Symbol: `ER_GRP_RPL_REINIT_OF_INTERNAL_CACHE_FOR_READ_FAILED`; SQLSTATE: `HY000`

  Message: Error while re-initializing an internal cache, for read operations, on session %u

  `ER_GRP_RPL_REINIT_OF_INTERNAL_CACHE_FOR_READ_FAILED` was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011610; Symbol: ER_GRP_RPL_APPENDING_DATA_TO_INTERNAL_CACHE_FAILED; SQLSTATE: HY000

  Message: Error while appending data to an internal cache on session %u

  ER_GRP_RPL_APPENDING_DATA_TO_INTERNAL_CACHE_FAILED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011611; Symbol: ER_GRP_RPL_WRITE_TO_BINLOG_CACHE_FAILED; SQLSTATE: HY000

  Message: Error while writing binary log cache on session %u

  ER_GRP_RPL_WRITE_TO_BINLOG_CACHE_FAILED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011611; Symbol: ER_GRP_RPL_WRITE_TO_TRANSACTION_MESSAGE_FAILED; SQLSTATE: HY000

  Message: Error while writing to transaction message on session %u

  ER_GRP_RPL_WRITE_TO_TRANSACTION_MESSAGE_FAILED was added in 8.0.13.

- Error number: MY-011612; Symbol: ER_GRP_RPL_FAILED_TO_REGISTER_TRANS_OUTCOME_NOTIFICTION; SQLSTATE: HY000

  Message: Unable to register for getting notifications regarding the outcome of the transaction on session %u

  ER_GRP_RPL_FAILED_TO_REGISTER_TRANS_OUTCOME_NOTIFICTION was added in 8.0.11.

- Error number: MY-011613; Symbol: ER_GRP_RPL_MSG_TOO_LONG_BROADCASTING_TRANS_FAILED; SQLSTATE: HY000

  Message: Error broadcasting transaction to the group on session %u. Message is too big.

  ER_GRP_RPL_MSG_TOO_LONG_BROADCASTING_TRANS_FAILED was added in 8.0.11.

- Error number: MY-011614; Symbol: ER_GRP_RPL_BROADCASTING_TRANS_TO_GRP_FAILED; SQLSTATE: HY000

  Message: Error while broadcasting the transaction to the group on session %u

  ER_GRP_RPL_BROADCASTING_TRANS_TO_GRP_FAILED was added in 8.0.11.

- Error number: MY-011615; Symbol: ER_GRP_RPL_ERROR_WHILE_WAITING_FOR_CONFLICT_DETECTION; SQLSTATE: HY000

  Message: Error while waiting for conflict detection procedure to finish on session %u

  ER_GRP_RPL_ERROR_WHILE_WAITING_FOR_CONFLICT_DETECTION was added in 8.0.11.

- Error number: MY-011616; Symbol: ER_GRP_RPL_REINIT_OF_INTERNAL_CACHE_FOR_WRITE_FAILED; SQLSTATE: HY000

  Message: Error while re-initializing an internal cache, for write operations, on session %u

  ER_GRP_RPL_REINIT_OF_INTERNAL_CACHE_FOR_WRITE_FAILED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011617; Symbol: ER_GRP_RPL_FAILED_TO_CREATE_COMMIT_CACHE; SQLSTATE: HY000

  Message: Failed to create group replication commit cache on session %u

  ER_GRP_RPL_FAILED_TO_CREATE_COMMIT_CACHE was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011618; Symbol: ER_GRP_RPL_REINIT_OF_COMMIT_CACHE_FOR_WRITE_FAILED; SQLSTATE: HY000

  Message: Failed to reinit group replication commit cache for write on session %u

  ER_GRP_RPL_REINIT_OF_COMMIT_CACHE_FOR_WRITE_FAILED was added in 8.0.11, removed after 8.0.12.

- Error number: MY-011619; Symbol: ER_GRP_RPL_PREV_REC_SESSION_RUNNING; SQLSTATE: HY000

  Message: A previous recovery session is still running. Please stop the group replication plugin and wait for it to stop

  ER_GRP_RPL_PREV_REC_SESSION_RUNNING was added in 8.0.11.

- Error number: MY-011620; Symbol: ER_GRP_RPL_FATAL_REC_PROCESS; SQLSTATE: HY000

  Message: Fatal error during the incremental recovery process of Group Replication. The server will leave the group.

  ER_GRP_RPL_FATAL_REC_PROCESS was added in 8.0.11.

- Error number: MY-011621; Symbol: ER_GRP_RPL_WHILE_STOPPING_REP_CHANNEL; SQLSTATE: HY000

  Message: Error stopping all replication channels while server was leaving the group. %s

  ER_GRP_RPL_WHILE_STOPPING_REP_CHANNEL was added in 8.0.11, removed after 8.0.17.

- Error number: MY-011622; Symbol: ER_GRP_RPL_UNABLE_TO_EVALUATE_APPLIER_STATUS; SQLSTATE: HY000

  Message: Unable to evaluate the group replication applier execution status. Group replication recovery will shutdown to avoid data corruption.

  ER_GRP_RPL_UNABLE_TO_EVALUATE_APPLIER_STATUS was added in 8.0.11.

- Error number: MY-011623; Symbol: ER_GRP_RPL_ONLY_ONE_SERVER_ALIVE; SQLSTATE: HY000

  Message: Only one server alive. Declaring this server as online within the replication group

  ER_GRP_RPL_ONLY_ONE_SERVER_ALIVE was added in 8.0.11.

- Error number: MY-011624; Symbol: ER_GRP_RPL_CERTIFICATION_REC_PROCESS; SQLSTATE: HY000

  Message: Error when processing certification information in the incremental recovery process

  ER_GRP_RPL_CERTIFICATION_REC_PROCESS was added in 8.0.11.

- Error number: MY-011625; Symbol: ER_GRP_RPL_UNABLE_TO_ENSURE_EXECUTION_REC; SQLSTATE: HY000

  Message: Unable to ensure the execution of group transactions received during recovery.

  ER_GRP_RPL_UNABLE_TO_ENSURE_EXECUTION_REC was added in 8.0.11.

- Error number: MY-011626; Symbol: ER_GRP_RPL_WHILE_SENDING_MSG_REC; SQLSTATE: HY000

  Message: Error while sending message in the group replication incremental recovery process.

  ER_GRP_RPL_WHILE_SENDING_MSG_REC was added in 8.0.11.

- Error number: MY-011627; Symbol: ER_GRP_RPL_READ_UNABLE_FOR_SUPER_READ_ONLY; SQLSTATE: HY000

  Message: Unable to read the server value for the super_read_only variable.

  ER_GRP_RPL_READ_UNABLE_FOR_SUPER_READ_ONLY was added in 8.0.11.

- Error number: MY-011628; Symbol: ER_GRP_RPL_READ_UNABLE_FOR_READ_ONLY_SUPER_READ_ONLY; SQLSTATE: HY000

  Message: Unable to read the server values for the read_only and super_read_only variables.

  ER_GRP_RPL_READ_UNABLE_FOR_READ_ONLY_SUPER_READ_ONLY was added in 8.0.11.

- Error number: MY-011629; Symbol: ER_GRP_RPL_UNABLE_TO_RESET_SERVER_READ_MODE; SQLSTATE: HY000

  Message: Unable to reset the server read mode settings. Try to reset them manually.

  ER_GRP_RPL_UNABLE_TO_RESET_SERVER_READ_MODE was added in 8.0.11.

- Error number: MY-011630; Symbol: ER_GRP_RPL_UNABLE_TO_CERTIFY_PLUGIN_TRANS; SQLSTATE: HY000

  Message: Due to a plugin error, some transactions were unable to be certified and will now rollback.

  ER_GRP_RPL_UNABLE_TO_CERTIFY_PLUGIN_TRANS was added in 8.0.11.

- Error number: MY-011631; Symbol: ER_GRP_RPL_UNBLOCK_CERTIFIED_TRANS; SQLSTATE: HY000

  Message: Error when trying to unblock non certified or consistent transactions. Check for consistency errors when restarting the service

  ER_GRP_RPL_UNBLOCK_CERTIFIED_TRANS was added in 8.0.11.

- Error number: MY-011632; Symbol: ER_GRP_RPL_SERVER_WORKING_AS_SECONDARY; SQLSTATE: HY000

  Message: This server is working as secondary member with primary member address %s:%u.

  ER_GRP_RPL_SERVER_WORKING_AS_SECONDARY was added in 8.0.11.

- Error number: MY-011633; Symbol: ER_GRP_RPL_FAILED_TO_START_WITH_INVALID_SERVER_ID; SQLSTATE: HY000

Message: Unable to start Group Replication. Replication applier infrastructure is not initialized since the server was started with server_id=0. Please, restart the server with server_id larger than 0.

`ER_GRP_RPL_FAILED_TO_START_WITH_INVALID_SERVER_ID` was added in 8.0.11.

- Error number: `MY-011634`; Symbol: `ER_GRP_RPL_FORCE_MEMBERS_MUST_BE_EMPTY`; SQLSTATE: `HY000`

  Message: group_replication_force_members must be empty on group start. Current value: '%s'

  `ER_GRP_RPL_FORCE_MEMBERS_MUST_BE_EMPTY` was added in 8.0.11.

- Error number: `MY-011635`; Symbol: `ER_GRP_RPL_PLUGIN_STRUCT_INIT_NOT_POSSIBLE_ON_SERVER_START`; SQLSTATE: `HY000`

  Message: It was not possible to guarantee the initialization of plugin structures on server start

  `ER_GRP_RPL_PLUGIN_STRUCT_INIT_NOT_POSSIBLE_ON_SERVER_START` was added in 8.0.11.

- Error number: `MY-011636`; Symbol: `ER_GRP_RPL_FAILED_TO_ENABLE_SUPER_READ_ONLY_MODE`; SQLSTATE: `HY000`

  Message: Could not enable the server read only mode and guarantee a safe recovery execution

  `ER_GRP_RPL_FAILED_TO_ENABLE_SUPER_READ_ONLY_MODE` was added in 8.0.11.

- Error number: `MY-011637`; Symbol: `ER_GRP_RPL_FAILED_TO_INIT_COMMUNICATION_ENGINE`; SQLSTATE: `HY000`

  Message: Error on group communication engine initialization

  `ER_GRP_RPL_FAILED_TO_INIT_COMMUNICATION_ENGINE` was added in 8.0.11.

- Error number: `MY-011638`; Symbol: `ER_GRP_RPL_FAILED_TO_START_ON_SECONDARY_WITH_ASYNC_CHANNELS`; SQLSTATE: `HY000`

  Message: Can't start group replication on secondary member with single-primary mode while asynchronous replication channels are running.

  `ER_GRP_RPL_FAILED_TO_START_ON_SECONDARY_WITH_ASYNC_CHANNELS` was added in 8.0.11.

- Error number: `MY-011639`; Symbol: `ER_GRP_RPL_FAILED_TO_START_COMMUNICATION_ENGINE`; SQLSTATE: `HY000`

  Message: Error on group communication engine start

  `ER_GRP_RPL_FAILED_TO_START_COMMUNICATION_ENGINE` was added in 8.0.11.

- Error number: `MY-011640`; Symbol: `ER_GRP_RPL_TIMEOUT_ON_VIEW_AFTER_JOINING_GRP`; SQLSTATE: `HY000`

  Message: Timeout on wait for view after joining group

  `ER_GRP_RPL_TIMEOUT_ON_VIEW_AFTER_JOINING_GRP` was added in 8.0.11.

- Error number: `MY-011641`; Symbol: `ER_GRP_RPL_FAILED_TO_CALL_GRP_COMMUNICATION_INTERFACE`; SQLSTATE: `HY000`

Message: Error calling group communication interfaces

ER_GRP_RPL_FAILED_TO_CALL_GRP_COMMUNICATION_INTERFACE was added in 8.0.11.

- Error number: MY-011642; Symbol: ER_GRP_RPL_MEMBER_SERVER_UUID_IS_INCOMPATIBLE_WITH_GRP; SQLSTATE: HY000

  Message: Member server_uuid is incompatible with the group. Server_uuid %s matches group_name %s.

  ER_GRP_RPL_MEMBER_SERVER_UUID_IS_INCOMPATIBLE_WITH_GRP was added in 8.0.11.

- Error number: MY-011643; Symbol: ER_GRP_RPL_MEMBER_CONF_INFO; SQLSTATE: HY000

  Message: Member configuration: member_id: %lu; member_uuid: "%s"; single-primary mode: "%s"; group_replication_auto_increment_increment: %lu;

  ER_GRP_RPL_MEMBER_CONF_INFO was added in 8.0.11.

- Error number: MY-011644; Symbol: ER_GRP_RPL_FAILED_TO_CONFIRM_IF_SERVER_LEFT_GRP; SQLSTATE: HY000

  Message: Unable to confirm whether the server has left the group or not. Check performance_schema.replication_group_members to check group membership information.

  ER_GRP_RPL_FAILED_TO_CONFIRM_IF_SERVER_LEFT_GRP was added in 8.0.11.

- Error number: MY-011645; Symbol: ER_GRP_RPL_SERVER_IS_ALREADY_LEAVING; SQLSTATE: HY000

  Message: Skipping leave operation: concurrent attempt to leave the group is on-going.

  ER_GRP_RPL_SERVER_IS_ALREADY_LEAVING was added in 8.0.11.

- Error number: MY-011646; Symbol: ER_GRP_RPL_SERVER_ALREADY_LEFT; SQLSTATE: HY000

  Message: Skipping leave operation: member already left the group.

  ER_GRP_RPL_SERVER_ALREADY_LEFT was added in 8.0.11.

- Error number: MY-011647; Symbol: ER_GRP_RPL_WAITING_FOR_VIEW_UPDATE; SQLSTATE: HY000

  Message: Going to wait for view modification

  ER_GRP_RPL_WAITING_FOR_VIEW_UPDATE was added in 8.0.11.

- Error number: MY-011648; Symbol: ER_GRP_RPL_TIMEOUT_RECEIVING_VIEW_CHANGE_ON_SHUTDOWN; SQLSTATE: HY000

  Message: While leaving the group due to a stop, shutdown or failure there was a timeout receiving a view change. This can lead to a possible inconsistent state. Check the log for more details

  ER_GRP_RPL_TIMEOUT_RECEIVING_VIEW_CHANGE_ON_SHUTDOWN was added in 8.0.11.

- Error number: MY-011649; Symbol: ER_GRP_RPL_REQUESTING_NON_MEMBER_SERVER_TO_LEAVE; SQLSTATE: HY000

Message: Requesting to leave the group despite of not being a member

ER_GRP_RPL_REQUESTING_NON_MEMBER_SERVER_TO_LEAVE was added in 8.0.11.

- Error number: MY-011650; Symbol: ER_GRP_RPL_IS_STOPPING; SQLSTATE: HY000

  Message: Plugin 'group_replication' is stopping.

  ER_GRP_RPL_IS_STOPPING was added in 8.0.11.

- Error number: MY-011651; Symbol: ER_GRP_RPL_IS_STOPPED; SQLSTATE: HY000

  Message: Plugin 'group_replication' has been stopped.

  ER_GRP_RPL_IS_STOPPED was added in 8.0.11.

- Error number: MY-011652; Symbol: ER_GRP_RPL_FAILED_TO_ENABLE_READ_ONLY_MODE_ON_SHUTDOWN; SQLSTATE: HY000

  Message: On plugin shutdown it was not possible to enable the server read only mode. Local transactions will be accepted and committed.

  ER_GRP_RPL_FAILED_TO_ENABLE_READ_ONLY_MODE_ON_SHUTDOWN was added in 8.0.11.

- Error number: MY-011653; Symbol: ER_GRP_RPL_RECOVERY_MODULE_TERMINATION_TIMED_OUT_ON_SHUTDOWN; SQLSTATE: HY000

  Message: On shutdown there was a timeout on the Group Replication recovery module termination. Check the log for more details

  ER_GRP_RPL_RECOVERY_MODULE_TERMINATION_TIMED_OUT_ON_SHUTDOWN was added in 8.0.11.

- Error number: MY-011654; Symbol: ER_GRP_RPL_APPLIER_TERMINATION_TIMED_OUT_ON_SHUTDOWN; SQLSTATE: HY000

  Message: On shutdown there was a timeout on the Group Replication applier termination.

  ER_GRP_RPL_APPLIER_TERMINATION_TIMED_OUT_ON_SHUTDOWN was added in 8.0.11.

- Error number: MY-011655; Symbol: ER_GRP_RPL_FAILED_TO_SHUTDOWN_REGISTRY_MODULE; SQLSTATE: HY000

  Message: Unexpected failure while shutting down registry module!

  ER_GRP_RPL_FAILED_TO_SHUTDOWN_REGISTRY_MODULE was added in 8.0.11.

- Error number: MY-011656; Symbol: ER_GRP_RPL_FAILED_TO_INIT_HANDLER; SQLSTATE: HY000

  Message: Failure during Group Replication handler initialization

  ER_GRP_RPL_FAILED_TO_INIT_HANDLER was added in 8.0.11.

- Error number: MY-011657; Symbol: ER_GRP_RPL_FAILED_TO_REGISTER_SERVER_STATE_OBSERVER; SQLSTATE: HY000

  Message: Failure when registering the server state observers

  ER_GRP_RPL_FAILED_TO_REGISTER_SERVER_STATE_OBSERVER was added in 8.0.11.

- Error number: MY-011658; Symbol: ER_GRP_RPL_FAILED_TO_REGISTER_TRANS_STATE_OBSERVER; SQLSTATE: HY000

  Message: Failure when registering the transactions state observers

  ER_GRP_RPL_FAILED_TO_REGISTER_TRANS_STATE_OBSERVER was added in 8.0.11.

- Error number: MY-011659; Symbol: ER_GRP_RPL_FAILED_TO_REGISTER_BINLOG_STATE_OBSERVER; SQLSTATE: HY000

  Message: Failure when registering the binlog state observers

  ER_GRP_RPL_FAILED_TO_REGISTER_BINLOG_STATE_OBSERVER was added in 8.0.11.

- Error number: MY-011660; Symbol: ER_GRP_RPL_FAILED_TO_START_ON_BOOT; SQLSTATE: HY000

  Message: Unable to start Group Replication on boot

  ER_GRP_RPL_FAILED_TO_START_ON_BOOT was added in 8.0.11.

- Error number: MY-011661; Symbol: ER_GRP_RPL_FAILED_TO_STOP_ON_PLUGIN_UNINSTALL; SQLSTATE: HY000

  Message: Failure when stopping Group Replication on plugin uninstall

  ER_GRP_RPL_FAILED_TO_STOP_ON_PLUGIN_UNINSTALL was added in 8.0.11.

- Error number: MY-011662; Symbol: ER_GRP_RPL_FAILED_TO_UNREGISTER_SERVER_STATE_OBSERVER; SQLSTATE: HY000

  Message: Failure when unregistering the server state observers

  ER_GRP_RPL_FAILED_TO_UNREGISTER_SERVER_STATE_OBSERVER was added in 8.0.11.

- Error number: MY-011663; Symbol: ER_GRP_RPL_FAILED_TO_UNREGISTER_TRANS_STATE_OBSERVER; SQLSTATE: HY000

  Message: Failure when unregistering the transactions state observers

  ER_GRP_RPL_FAILED_TO_UNREGISTER_TRANS_STATE_OBSERVER was added in 8.0.11.

- Error number: MY-011664; Symbol: ER_GRP_RPL_FAILED_TO_UNREGISTER_BINLOG_STATE_OBSERVER; SQLSTATE: HY000

  Message: Failure when unregistering the binlog state observers

  ER_GRP_RPL_FAILED_TO_UNREGISTER_BINLOG_STATE_OBSERVER was added in 8.0.11.

- Error number: MY-011665; Symbol: ER_GRP_RPL_ALL_OBSERVERS_UNREGISTERED; SQLSTATE: HY000

  Message: All Group Replication server observers have been successfully unregistered

  ER_GRP_RPL_ALL_OBSERVERS_UNREGISTERED was added in 8.0.11.

- Error number: MY-011666; Symbol: ER_GRP_RPL_FAILED_TO_PARSE_THE_GRP_NAME; SQLSTATE: HY000

Message: Unable to parse the group name.

ER_GRP_RPL_FAILED_TO_PARSE_THE_GRP_NAME was added in 8.0.11.

- Error number: MY-011667; Symbol: ER_GRP_RPL_FAILED_TO_GENERATE_SIDNO_FOR_GRP; SQLSTATE: HY000

  Message: Unable to parse the group name.

  ER_GRP_RPL_FAILED_TO_GENERATE_SIDNO_FOR_GRP was added in 8.0.11.

- Error number: MY-011668; Symbol: ER_GRP_RPL_APPLIER_NOT_STARTED_DUE_TO_RUNNING_PREV_SHUTDOWN; SQLSTATE: HY000

  Message: Cannot start the Group Replication applier as a previous shutdown is still running: The thread will stop once its task is complete.

  ER_GRP_RPL_APPLIER_NOT_STARTED_DUE_TO_RUNNING_PREV_SHUTDOWN was added in 8.0.11.

- Error number: MY-011669; Symbol: ER_GRP_RPL_FAILED_TO_INIT_APPLIER_MODULE; SQLSTATE: HY000

  Message: Unable to initialize the Group Replication applier module.

  ER_GRP_RPL_FAILED_TO_INIT_APPLIER_MODULE was added in 8.0.11.

- Error number: MY-011670; Symbol: ER_GRP_RPL_APPLIER_INITIALIZED; SQLSTATE: HY000

  Message: Group Replication applier module successfully initialized!

  ER_GRP_RPL_APPLIER_INITIALIZED was added in 8.0.11.

- Error number: MY-011671; Symbol: ER_GRP_RPL_COMMUNICATION_SSL_CONF_INFO; SQLSTATE: HY000

  Message: Group communication SSL configuration: group_replication_ssl_mode: "%s"; server_key_file: "%s"; server_cert_file: "%s"; client_key_file: "%s"; client_cert_file: "%s"; ca_file: "%s"; ca_path: "%s"; cipher: "%s"; tls_version: "%s"; tls_ciphersuites: "%s"; crl_file: "%s"; crl_path: "%s"; ssl_fips_mode: "%s"

  ER_GRP_RPL_COMMUNICATION_SSL_CONF_INFO was added in 8.0.11.

- Error number: MY-011672; Symbol: ER_GRP_RPL_ABORTS_AS_SSL_NOT_SUPPORTED_BY_MYSQLD; SQLSTATE: HY000

  Message: MySQL server does not have SSL support and group_replication_ssl_mode is "%s", START GROUP_REPLICATION will abort

  ER_GRP_RPL_ABORTS_AS_SSL_NOT_SUPPORTED_BY_MYSQLD was added in 8.0.11.

- Error number: MY-011673; Symbol: ER_GRP_RPL_SSL_DISABLED; SQLSTATE: HY000

  Message: Group communication SSL configuration: group_replication_ssl_mode: "%s"

  ER_GRP_RPL_SSL_DISABLED was added in 8.0.11.

- Error number: MY-011674; Symbol: ER_GRP_RPL_UNABLE_TO_INIT_COMMUNICATION_ENGINE; SQLSTATE: HY000

Message: Unable to initialize the group communication engine

ER_GRP_RPL_UNABLE_TO_INIT_COMMUNICATION_ENGINE was added in 8.0.11.

- Error number: MY-011675; Symbol: ER_GRP_RPL_BINLOG_DISABLED; SQLSTATE: HY000

  Message: Binlog must be enabled for Group Replication

  ER_GRP_RPL_BINLOG_DISABLED was added in 8.0.11.

- Error number: MY-011676; Symbol: ER_GRP_RPL_GTID_MODE_OFF; SQLSTATE: HY000

  Message: Gtid mode should be ON for Group Replication

  ER_GRP_RPL_GTID_MODE_OFF was added in 8.0.11.

- Error number: MY-011677; Symbol: ER_GRP_RPL_LOG_SLAVE_UPDATES_NOT_SET; SQLSTATE: HY000

  Message: LOG_SLAVE_UPDATES should be ON for Group Replication

  ER_GRP_RPL_LOG_SLAVE_UPDATES_NOT_SET was added in 8.0.11.

- Error number: MY-011678; Symbol: ER_GRP_RPL_INVALID_TRANS_WRITE_SET_EXTRACTION_VALUE; SQLSTATE: HY000

  Message: Extraction of transaction write sets requires an hash algorithm configuration. Please, double check that the parameter transaction-write-set-extraction is set to a valid algorithm.

  ER_GRP_RPL_INVALID_TRANS_WRITE_SET_EXTRACTION_VALUE was added in 8.0.11.

- Error number: MY-011679; Symbol: ER_GRP_RPL_RELAY_LOG_INFO_REPO_MUST_BE_TABLE; SQLSTATE: HY000

  Message: Relay log info repository must be set to TABLE

  ER_GRP_RPL_RELAY_LOG_INFO_REPO_MUST_BE_TABLE was added in 8.0.11.

- Error number: MY-011680; Symbol: ER_GRP_RPL_MASTER_INFO_REPO_MUST_BE_TABLE; SQLSTATE: HY000

  Message: Master info repository must be set to TABLE.

  ER_GRP_RPL_MASTER_INFO_REPO_MUST_BE_TABLE was added in 8.0.11.

- Error number: MY-011681; Symbol: ER_GRP_RPL_INCORRECT_TYPE_SET_FOR_PARALLEL_APPLIER; SQLSTATE: HY000

  Message: In order to use parallel applier on Group Replication, parameter slave-parallel-type must be set to 'LOGICAL_CLOCK'.

  ER_GRP_RPL_INCORRECT_TYPE_SET_FOR_PARALLEL_APPLIER was added in 8.0.11.

- Error number: MY-011682; Symbol: ER_GRP_RPL_SLAVE_PRESERVE_COMMIT_ORDER_NOT_SET; SQLSTATE: HY000

  Message: Group Replication requires slave-preserve-commit-order to be set to ON when using more than 1 applier threads.

ER_GRP_RPL_SLAVE_PRESERVE_COMMIT_ORDER_NOT_SET was added in 8.0.11.

- Error number: MY-011683; Symbol: ER_GRP_RPL_SINGLE_PRIM_MODE_NOT_ALLOWED_WITH_UPDATE_EVERYWHERE; SQLSTATE: HY000

  Message: It is not allowed to run single primary mode with 'enforce_update_everywhere_checks' enabled.

  ER_GRP_RPL_SINGLE_PRIM_MODE_NOT_ALLOWED_WITH_UPDATE_EVERYWHERE was added in 8.0.11.

- Error number: MY-011684; Symbol: ER_GRP_RPL_MODULE_TERMINATE_ERROR; SQLSTATE: HY000

  Message: error_message: %s

  ER_GRP_RPL_MODULE_TERMINATE_ERROR was added in 8.0.11.

- Error number: MY-011685; Symbol: ER_GRP_RPL_GRP_NAME_OPTION_MANDATORY; SQLSTATE: HY000

  Message: The group name option is mandatory

  ER_GRP_RPL_GRP_NAME_OPTION_MANDATORY was added in 8.0.11.

- Error number: MY-011686; Symbol: ER_GRP_RPL_GRP_NAME_IS_TOO_LONG; SQLSTATE: HY000

  Message: The group name '%s' is not a valid UUID, its length is too big

  ER_GRP_RPL_GRP_NAME_IS_TOO_LONG was added in 8.0.11.

- Error number: MY-011687; Symbol: ER_GRP_RPL_GRP_NAME_IS_NOT_VALID_UUID; SQLSTATE: HY000

  Message: The group name '%s' is not a valid UUID

  ER_GRP_RPL_GRP_NAME_IS_NOT_VALID_UUID was added in 8.0.11.

- Error number: MY-011688; Symbol: ER_GRP_RPL_FLOW_CTRL_MIN_QUOTA_GREATER_THAN_MAX_QUOTA; SQLSTATE: HY000

  Message: group_replication_flow_control_min_quota cannot be larger than group_replication_flow_control_max_quota

  ER_GRP_RPL_FLOW_CTRL_MIN_QUOTA_GREATER_THAN_MAX_QUOTA was added in 8.0.11.

- Error number: MY-011689; Symbol: ER_GRP_RPL_FLOW_CTRL_MIN_RECOVERY_QUOTA_GREATER_THAN_MAX_QUOTA; SQLSTATE: HY000

  Message: group_replication_flow_control_min_recovery_quota cannot be larger than group_replication_flow_control_max_quota

  ER_GRP_RPL_FLOW_CTRL_MIN_RECOVERY_QUOTA_GREATER_THAN_MAX_QUOTA was added in 8.0.11.

- Error number: MY-011690; Symbol: ER_GRP_RPL_FLOW_CTRL_MAX_QUOTA_SMALLER_THAN_MIN_QUOTAS; SQLSTATE: HY000

Message: group_replication_flow_control_max_quota cannot be smaller than group_replication_flow_control_min_quota or group_replication_flow_control_min_recovery_quota

ER_GRP_RPL_FLOW_CTRL_MAX_QUOTA_SMALLER_THAN_MIN_QUOTAS was added in 8.0.11.

- Error number: MY-011691; Symbol: ER_GRP_RPL_INVALID_SSL_RECOVERY_STRING; SQLSTATE: HY000

  Message: The given value for recovery ssl option '%s' is invalid as its length is beyond the limit

  ER_GRP_RPL_INVALID_SSL_RECOVERY_STRING was added in 8.0.11.

- Error number: MY-011692; Symbol: ER_GRP_RPL_SUPPORTS_ONLY_ONE_FORCE_MEMBERS_SET; SQLSTATE: HY000

  Message: There is one group_replication_force_members operation already ongoing

  ER_GRP_RPL_SUPPORTS_ONLY_ONE_FORCE_MEMBERS_SET was added in 8.0.11.

- Error number: MY-011693; Symbol: ER_GRP_RPL_FORCE_MEMBERS_SET_UPDATE_NOT_ALLOWED; SQLSTATE: HY000

  Message: group_replication_force_members can only be updated when Group Replication is running and a majority of the members are unreachable

  ER_GRP_RPL_FORCE_MEMBERS_SET_UPDATE_NOT_ALLOWED was added in 8.0.11.

- Error number: MY-011694; Symbol: ER_GRP_RPL_GRP_COMMUNICATION_INIT_WITH_CONF; SQLSTATE: HY000

  Message: Initialized group communication with configuration: group_replication_group_name: '%s'; group_replication_local_address: '%s'; group_replication_group_seeds: '%s'; group_replication_bootstrap_group: '%s'; group_replication_poll_spin_loops: %lu; group_replication_compression_threshold: %lu; group_replication_ip_whitelist: '%s'; group_replication_communication_debug_options: '%s'; group_replication_member_expel_timeout: '%lu'; group_replication_communication_max_message_size: %lu; group_replication_message_cache_size: '%luu'

  ER_GRP_RPL_GRP_COMMUNICATION_INIT_WITH_CONF was added in 8.0.11.

- Error number: MY-011695; Symbol: ER_GRP_RPL_UNKNOWN_GRP_RPL_APPLIER_PIPELINE_REQUESTED; SQLSTATE: HY000

  Message: Unknown group replication applier pipeline requested

  ER_GRP_RPL_UNKNOWN_GRP_RPL_APPLIER_PIPELINE_REQUESTED was added in 8.0.11.

- Error number: MY-011696; Symbol: ER_GRP_RPL_FAILED_TO_BOOTSTRAP_EVENT_HANDLING_INFRASTRUCTURE; SQLSTATE: HY000

  Message: Unable to bootstrap group replication event handling infrastructure. Unknown handler type: %d

  ER_GRP_RPL_FAILED_TO_BOOTSTRAP_EVENT_HANDLING_INFRASTRUCTURE was added in 8.0.11.

- Error number: MY-011697; Symbol: ER_GRP_RPL_APPLIER_HANDLER_NOT_INITIALIZED; SQLSTATE: HY000

Message: One of the group replication applier handlers is null due to an initialization error

ER_GRP_RPL_APPLIER_HANDLER_NOT_INITIALIZED was added in 8.0.11.

- Error number: MY-011698; Symbol: ER_GRP_RPL_APPLIER_HANDLER_IS_IN_USE; SQLSTATE: HY000

  Message: A group replication applier handler, marked as unique, is already in use.

  ER_GRP_RPL_APPLIER_HANDLER_IS_IN_USE was added in 8.0.11.

- Error number: MY-011699; Symbol: ER_GRP_RPL_APPLIER_HANDLER_ROLE_IS_IN_USE; SQLSTATE: HY000

  Message: A group replication applier handler role, that was marked as unique, is already in use.

  ER_GRP_RPL_APPLIER_HANDLER_ROLE_IS_IN_USE was added in 8.0.11.

- Error number: MY-011700; Symbol: ER_GRP_RPL_FAILED_TO_INIT_APPLIER_HANDLER; SQLSTATE: HY000

  Message: Error on group replication applier handler initialization

  ER_GRP_RPL_FAILED_TO_INIT_APPLIER_HANDLER was added in 8.0.11.

- Error number: MY-011701; Symbol: ER_GRP_RPL_SQL_SERVICE_FAILED_TO_INIT_SESSION_THREAD; SQLSTATE: HY000

  Message: Error when initializing a session thread for internal server connection.

  ER_GRP_RPL_SQL_SERVICE_FAILED_TO_INIT_SESSION_THREAD was added in 8.0.11.

- Error number: MY-011702; Symbol: ER_GRP_RPL_SQL_SERVICE_COMM_SESSION_NOT_INITIALIZED; SQLSTATE: HY000

  Message: Error running internal SQL query: %s. The internal server communication session is not initialized

  ER_GRP_RPL_SQL_SERVICE_COMM_SESSION_NOT_INITIALIZED was added in 8.0.11.

- Error number: MY-011703; Symbol: ER_GRP_RPL_SQL_SERVICE_SERVER_SESSION_KILLED; SQLSTATE: HY000

  Message: Error running internal SQL query: %s. The internal server session was killed or server is shutting down.

  ER_GRP_RPL_SQL_SERVICE_SERVER_SESSION_KILLED was added in 8.0.11.

- Error number: MY-011704; Symbol: ER_GRP_RPL_SQL_SERVICE_FAILED_TO_RUN_SQL_QUERY; SQLSTATE: HY000

  Message: Error running internal SQL query: %s. Got internal SQL error: %s(%d)

  ER_GRP_RPL_SQL_SERVICE_FAILED_TO_RUN_SQL_QUERY was added in 8.0.11.

- Error number: MY-011705; Symbol: ER_GRP_RPL_SQL_SERVICE_SERVER_INTERNAL_FAILURE; SQLSTATE: HY000

Message: Error running internal SQL query: %s. Internal failure.

ER_GRP_RPL_SQL_SERVICE_SERVER_INTERNAL_FAILURE was added in 8.0.11.

- Error number: MY-011706; Symbol: ER_GRP_RPL_SQL_SERVICE_RETRIES_EXCEEDED_ON_SESSION_STATE; SQLSTATE: HY000

  Message: Error, maximum number of retries exceeded when waiting for the internal server session state to be operating

  ER_GRP_RPL_SQL_SERVICE_RETRIES_EXCEEDED_ON_SESSION_STATE was added in 8.0.11.

- Error number: MY-011707; Symbol: ER_GRP_RPL_SQL_SERVICE_FAILED_TO_FETCH_SECURITY_CTX; SQLSTATE: HY000

  Message: Error when trying to fetch security context when contacting the server for internal plugin requests.

  ER_GRP_RPL_SQL_SERVICE_FAILED_TO_FETCH_SECURITY_CTX was added in 8.0.11.

- Error number: MY-011708; Symbol: ER_GRP_RPL_SQL_SERVICE_SERVER_ACCESS_DENIED_FOR_USER; SQLSTATE: HY000

  Message: There was an error when trying to access the server with user: %s. Make sure the user is present in the server and that the MySQL upgrade procedure was run correctly.

  ER_GRP_RPL_SQL_SERVICE_SERVER_ACCESS_DENIED_FOR_USER was added in 8.0.11.

- Error number: MY-011709; Symbol: ER_GRP_RPL_SQL_SERVICE_MAX_CONN_ERROR_FROM_SERVER; SQLSTATE: HY000

  Message: Failed to establish an internal server connection to execute plugin operations since the server does not have available connections, please increase @@GLOBAL.MAX_CONNECTIONS. Server error: %i.

  ER_GRP_RPL_SQL_SERVICE_MAX_CONN_ERROR_FROM_SERVER was added in 8.0.11.

- Error number: MY-011710; Symbol: ER_GRP_RPL_SQL_SERVICE_SERVER_ERROR_ON_CONN; SQLSTATE: HY000

  Message: Failed to establish an internal server connection to execute plugin operations. Server error: %i. Server error message: %s

  ER_GRP_RPL_SQL_SERVICE_SERVER_ERROR_ON_CONN was added in 8.0.11.

- Error number: MY-011711; Symbol: ER_GRP_RPL_UNREACHABLE_MAJORITY_TIMEOUT_FOR_MEMBER; SQLSTATE: HY000

  Message: This member could not reach a majority of the members for more than %ld seconds. The member will now leave the group as instructed by the group_replication_unreachable_majority_timeout option.

  ER_GRP_RPL_UNREACHABLE_MAJORITY_TIMEOUT_FOR_MEMBER was added in 8.0.11.

- Error number: MY-011712; Symbol: ER_GRP_RPL_SERVER_SET_TO_READ_ONLY_DUE_TO_ERRORS; SQLSTATE: HY000

Message: The server was automatically set into read only mode after an error was detected.

ER_GRP_RPL_SERVER_SET_TO_READ_ONLY_DUE_TO_ERRORS was added in 8.0.11.

- Error number: MY-011713; Symbol: ER_GRP_RPL_GMS_LISTENER_FAILED_TO_LOG_NOTIFICATION; SQLSTATE: HY000

  Message: Unable to log notification to table (errno: %lu) (res: %d)! Message: %s

  ER_GRP_RPL_GMS_LISTENER_FAILED_TO_LOG_NOTIFICATION was added in 8.0.11.

- Error number: MY-011714; Symbol: ER_GRP_RPL_GRP_COMMUNICATION_ENG_INIT_FAILED; SQLSTATE: HY000

  Message: Failure in group communication engine '%s' initialization

  ER_GRP_RPL_GRP_COMMUNICATION_ENG_INIT_FAILED was added in 8.0.11.

- Error number: MY-011715; Symbol: ER_GRP_RPL_SET_GRP_COMMUNICATION_ENG_LOGGER_FAILED; SQLSTATE: HY000

  Message: Unable to set the group communication engine logger

  ER_GRP_RPL_SET_GRP_COMMUNICATION_ENG_LOGGER_FAILED was added in 8.0.11.

- Error number: MY-011716; Symbol: ER_GRP_RPL_DEBUG_OPTIONS; SQLSTATE: HY000

  Message: Current debug options are: '%s'.

  ER_GRP_RPL_DEBUG_OPTIONS was added in 8.0.11.

- Error number: MY-011717; Symbol: ER_GRP_RPL_INVALID_DEBUG_OPTIONS; SQLSTATE: HY000

  Message: Some debug options in '%s' are not valid.

  ER_GRP_RPL_INVALID_DEBUG_OPTIONS was added in 8.0.11.

- Error number: MY-011718; Symbol: ER_GRP_RPL_EXIT_GRP_GCS_ERROR; SQLSTATE: HY000

  Message: Error calling group communication interfaces while trying to leave the group

  ER_GRP_RPL_EXIT_GRP_GCS_ERROR was added in 8.0.11.

- Error number: MY-011719; Symbol: ER_GRP_RPL_GRP_MEMBER_OFFLINE; SQLSTATE: HY000

  Message: Member is not ONLINE, it is not possible to force a new group membership

  ER_GRP_RPL_GRP_MEMBER_OFFLINE was added in 8.0.11.

- Error number: MY-011720; Symbol: ER_GRP_RPL_GCS_INTERFACE_ERROR; SQLSTATE: HY000

  Message: Error calling group communication interfaces

  ER_GRP_RPL_GCS_INTERFACE_ERROR was added in 8.0.11.

- Error number: MY-011721; Symbol: ER_GRP_RPL_FORCE_MEMBER_VALUE_SET_ERROR; SQLSTATE: HY000

  Message: Error setting group_replication_force_members value '%s' on group communication interfaces

ER_GRP_RPL_FORCE_MEMBER_VALUE_SET_ERROR was added in 8.0.11.

- Error number: MY-011722; Symbol: ER_GRP_RPL_FORCE_MEMBER_VALUE_SET; SQLSTATE: HY000

  Message: The group_replication_force_members value '%s' was set in the group communication interfaces

  ER_GRP_RPL_FORCE_MEMBER_VALUE_SET was added in 8.0.11.

- Error number: MY-011723; Symbol: ER_GRP_RPL_FORCE_MEMBER_VALUE_TIME_OUT; SQLSTATE: HY000

  Message: Timeout on wait for view after setting group_replication_force_members value '%s' into group communication interfaces

  ER_GRP_RPL_FORCE_MEMBER_VALUE_TIME_OUT was added in 8.0.11.

- Error number: MY-011724; Symbol: ER_GRP_RPL_BROADCAST_COMMIT_MSSG_TOO_BIG; SQLSTATE: HY000

  Message: Broadcast of committed transactions message failed. Message is too big.

  ER_GRP_RPL_BROADCAST_COMMIT_MSSG_TOO_BIG was added in 8.0.11.

- Error number: MY-011725; Symbol: ER_GRP_RPL_SEND_STATS_ERROR; SQLSTATE: HY000

  Message: Error while sending stats message

  ER_GRP_RPL_SEND_STATS_ERROR was added in 8.0.11.

- Error number: MY-011726; Symbol: ER_GRP_RPL_MEMBER_STATS_INFO; SQLSTATE: HY000

  Message: Flow control - update member stats: %s stats certifier_queue %d, applier_queue %d certified %ld (%ld), applied %ld (%ld), local %ld (%ld), quota %ld (%ld) mode=%d

  ER_GRP_RPL_MEMBER_STATS_INFO was added in 8.0.11.

- Error number: MY-011727; Symbol: ER_GRP_RPL_FLOW_CONTROL_STATS; SQLSTATE: HY000

  Message: Flow control: throttling to %ld commits per %ld sec, with %d writing and %d non-recovering members, min capacity %lld, lim throttle %lld

  ER_GRP_RPL_FLOW_CONTROL_STATS was added in 8.0.11.

- Error number: MY-011728; Symbol: ER_GRP_RPL_UNABLE_TO_CONVERT_PACKET_TO_EVENT; SQLSTATE: HY000

  Message: Unable to convert a packet into an event on the applier. Error: %s

  ER_GRP_RPL_UNABLE_TO_CONVERT_PACKET_TO_EVENT was added in 8.0.11.

- Error number: MY-011729; Symbol: ER_GRP_RPL_PIPELINE_CREATE_FAILED; SQLSTATE: HY000

  Message: Failed to create group replication pipeline cache.

  ER_GRP_RPL_PIPELINE_CREATE_FAILED was added in 8.0.11.

- Error number: MY-011730; Symbol: ER_GRP_RPL_PIPELINE_REINIT_FAILED_WRITE; SQLSTATE: HY000

  Message: Failed to reinit group replication pipeline cache for write.

  ER_GRP_RPL_PIPELINE_REINIT_FAILED_WRITE was added in 8.0.11.

- Error number: MY-011731; Symbol: ER_GRP_RPL_UNABLE_TO_CONVERT_EVENT_TO_PACKET; SQLSTATE: HY000

  Message: Unable to convert the event into a packet on the applier. Error: %s

  ER_GRP_RPL_UNABLE_TO_CONVERT_EVENT_TO_PACKET was added in 8.0.11.

- Error number: MY-011732; Symbol: ER_GRP_RPL_PIPELINE_FLUSH_FAIL; SQLSTATE: HY000

  Message: Failed to flush group replication pipeline cache.

  ER_GRP_RPL_PIPELINE_FLUSH_FAIL was added in 8.0.11.

- Error number: MY-011733; Symbol: ER_GRP_RPL_PIPELINE_REINIT_FAILED_READ; SQLSTATE: HY000

  Message: Failed to reinit group replication pipeline cache for read.

  ER_GRP_RPL_PIPELINE_REINIT_FAILED_READ was added in 8.0.11.

- Error number: MY-011734; Symbol: ER_GRP_RPL_STOP_REP_CHANNEL; SQLSTATE: HY000

  Message: Error stopping all replication channels while server was leaving the group. Got error: %d. Please check the error log for more details.

  ER_GRP_RPL_STOP_REP_CHANNEL was added in 8.0.11, removed after 8.0.17.

- Error number: MY-011735; Symbol: ER_GRP_RPL_GCS_GR_ERROR_MSG; SQLSTATE: HY000

  Message: %s

  ER_GRP_RPL_GCS_GR_ERROR_MSG was added in 8.0.11.

- Error number: MY-011736; Symbol: ER_GRP_RPL_SLAVE_IO_THREAD_UNBLOCKED; SQLSTATE: HY000

  Message: The slave IO thread of channel '%s' is unblocked as the member is declared ONLINE now.

  ER_GRP_RPL_SLAVE_IO_THREAD_UNBLOCKED was added in 8.0.11.

- Error number: MY-011737; Symbol: ER_GRP_RPL_SLAVE_IO_THREAD_ERROR_OUT; SQLSTATE: HY000

  Message: The slave IO thread of channel '%s' will error out as the member failed to come ONLINE.

  ER_GRP_RPL_SLAVE_IO_THREAD_ERROR_OUT was added in 8.0.11.

- Error number: MY-011738; Symbol: ER_GRP_RPL_SLAVE_APPLIER_THREAD_UNBLOCKED; SQLSTATE: HY000

  Message: The slave applier thread of channel '%s' is unblocked as the member is declared ONLINE now.

ER_GRP_RPL_SLAVE_APPLIER_THREAD_UNBLOCKED was added in 8.0.11.

- Error number: MY-011739; Symbol: ER_GRP_RPL_SLAVE_APPLIER_THREAD_ERROR_OUT; SQLSTATE: HY000

  Message: The slave applier thread of channel '%s' will error out as the member failed to come ONLINE.

  ER_GRP_RPL_SLAVE_APPLIER_THREAD_ERROR_OUT was added in 8.0.11.

- Error number: MY-011740; Symbol: ER_LDAP_AUTH_FAILED_TO_CREATE_OR_GET_CONNECTION; SQLSTATE: HY000

  Message: LDAP authentication initialize: failed to create/ get connection from the pool.

  ER_LDAP_AUTH_FAILED_TO_CREATE_OR_GET_CONNECTION was added in 8.0.11.

- Error number: MY-011741; Symbol: ER_LDAP_AUTH_DEINIT_FAILED; SQLSTATE: HY000

  Message: LDAP authentication de_initialize Failed

  ER_LDAP_AUTH_DEINIT_FAILED was added in 8.0.11.

- Error number: MY-011742; Symbol: ER_LDAP_AUTH_SKIPPING_USER_GROUP_SEARCH; SQLSTATE: HY000

  Message: Skipping group search, No group attribute mentioned

  ER_LDAP_AUTH_SKIPPING_USER_GROUP_SEARCH was added in 8.0.11.

- Error number: MY-011743; Symbol: ER_LDAP_AUTH_POOL_DISABLE_MAX_SIZE_ZERO; SQLSTATE: HY000

  Message: Pool max size is 0, connection pool is disabled

  ER_LDAP_AUTH_POOL_DISABLE_MAX_SIZE_ZERO was added in 8.0.11.

- Error number: MY-011744; Symbol: ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_OBJECT_CREATOR; SQLSTATE: HY000

  Message: Connection pool initialization, failed to create LDAP object creator

  ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_OBJECT_CREATOR was added in 8.0.11.

- Error number: MY-011745; Symbol: ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_OBJECT; SQLSTATE: HY000

  Message: Connection pool initialization, failed to create LDAP object

  ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_OBJECT was added in 8.0.11.

- Error number: MY-011746; Symbol: ER_LDAP_AUTH_TLS_CONF; SQLSTATE: HY000

  Message: LDAP TLS configuration

  ER_LDAP_AUTH_TLS_CONF was added in 8.0.11.

- Error number: MY-011747; Symbol: ER_LDAP_AUTH_TLS_CONNECTION; SQLSTATE: HY000

  Message: LDAP TLS connection

ER_LDAP_AUTH_TLS_CONNECTION was added in 8.0.11.

- Error number: MY-011748; Symbol: ER_LDAP_AUTH_CONN_POOL_NOT_CREATED; SQLSTATE: HY000

  Message: LDAP pool is not created.

  ER_LDAP_AUTH_CONN_POOL_NOT_CREATED was added in 8.0.11.

- Error number: MY-011749; Symbol: ER_LDAP_AUTH_CONN_POOL_INITIALIZING; SQLSTATE: HY000

  Message: LDAP pool is initializing

  ER_LDAP_AUTH_CONN_POOL_INITIALIZING was added in 8.0.11.

- Error number: MY-011750; Symbol: ER_LDAP_AUTH_CONN_POOL_DEINITIALIZING; SQLSTATE: HY000

  Message: LDAP pool is de-initializing

  ER_LDAP_AUTH_CONN_POOL_DEINITIALIZING was added in 8.0.11.

- Error number: MY-011751; Symbol: ER_LDAP_AUTH_ZERO_MAX_POOL_SIZE_UNCHANGED; SQLSTATE: HY000

  Message: Pool max size old and new values are 0

  ER_LDAP_AUTH_ZERO_MAX_POOL_SIZE_UNCHANGED was added in 8.0.11.

- Error number: MY-011752; Symbol: ER_LDAP_AUTH_POOL_REINITIALIZING; SQLSTATE: HY000

  Message: LDAP pool is re-initializing

  ER_LDAP_AUTH_POOL_REINITIALIZING was added in 8.0.11.

- Error number: MY-011753; Symbol: ER_LDAP_AUTH_FAILED_TO_WRITE_PACKET; SQLSTATE: HY000

  Message: Plug-in has failed to write the packet.

  ER_LDAP_AUTH_FAILED_TO_WRITE_PACKET was added in 8.0.11.

- Error number: MY-011754; Symbol: ER_LDAP_AUTH_SETTING_USERNAME; SQLSTATE: HY000

  Message: Setting LDAP user name as : %s

  ER_LDAP_AUTH_SETTING_USERNAME was added in 8.0.11.

- Error number: MY-011755; Symbol: ER_LDAP_AUTH_USER_AUTH_DATA; SQLSTATE: HY000

  Message: User authentication data: %s size: %lu

  ER_LDAP_AUTH_USER_AUTH_DATA was added in 8.0.11.

- Error number: MY-011756; Symbol: ER_LDAP_AUTH_INFO_FOR_USER; SQLSTATE: HY000

  Message: User is authenticated as: %s external user: %s

ER_LDAP_AUTH_INFO_FOR_USER was added in 8.0.11.

- Error number: MY-011757; Symbol: ER_LDAP_AUTH_USER_GROUP_SEARCH_INFO; SQLSTATE: HY000

  Message: Group search information base DN: %s scope: %d filter: %s attribute: %s

  ER_LDAP_AUTH_USER_GROUP_SEARCH_INFO was added in 8.0.11.

- Error number: MY-011758; Symbol: ER_LDAP_AUTH_GRP_SEARCH_SPECIAL_HDL; SQLSTATE: HY000

  Message: Special handling for group search, {GA} found

  ER_LDAP_AUTH_GRP_SEARCH_SPECIAL_HDL was added in 8.0.11.

- Error number: MY-011759; Symbol: ER_LDAP_AUTH_GRP_IS_FULL_DN; SQLSTATE: HY000

  Message: Group search special handling, group full DN found.

  ER_LDAP_AUTH_GRP_IS_FULL_DN was added in 8.0.11.

- Error number: MY-011760; Symbol: ER_LDAP_AUTH_USER_NOT_FOUND_IN_ANY_GRP; SQLSTATE: HY000

  Message: User %s is not member of any group.

  ER_LDAP_AUTH_USER_NOT_FOUND_IN_ANY_GRP was added in 8.0.11.

- Error number: MY-011761; Symbol: ER_LDAP_AUTH_USER_FOUND_IN_MANY_GRPS; SQLSTATE: HY000

  Message: User %s is member of more than one group

  ER_LDAP_AUTH_USER_FOUND_IN_MANY_GRPS was added in 8.0.11.

- Error number: MY-011762; Symbol: ER_LDAP_AUTH_USER_HAS_MULTIPLE_GRP_NAMES; SQLSTATE: HY000

  Message: For user %s has multiple user group names. Please check if group attribute name is correct

  ER_LDAP_AUTH_USER_HAS_MULTIPLE_GRP_NAMES was added in 8.0.11.

- Error number: MY-011763; Symbol: ER_LDAP_AUTH_SEARCHED_USER_GRP_NAME; SQLSTATE: HY000

  Message: Searched group name: %s

  ER_LDAP_AUTH_SEARCHED_USER_GRP_NAME was added in 8.0.11.

- Error number: MY-011764; Symbol: ER_LDAP_AUTH_OBJECT_CREATE_TIMESTAMP; SQLSTATE: HY000

  Message: LDAP authentication object creation time_stamp: %s dn: %s

  ER_LDAP_AUTH_OBJECT_CREATE_TIMESTAMP was added in 8.0.11.

- Error number: MY-011765; Symbol: ER_LDAP_AUTH_CERTIFICATE_NAME; SQLSTATE: HY000

Message: Certificate name: %s

ER_LDAP_AUTH_CERTIFICATE_NAME was added in 8.0.11.

- Error number: MY-011766; Symbol: ER_LDAP_AUTH_FAILED_TO_POOL_DEINIT; SQLSTATE: HY000

  Message: Failed to pool de-initialized: pool is already reconstructing

  ER_LDAP_AUTH_FAILED_TO_POOL_DEINIT was added in 8.0.11.

- Error number: MY-011767; Symbol: ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_RECONSTRUCTING; SQLSTATE: HY000

  Message: Pool initialization failed: pool is already initialized

  ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_RECONSTRUCTING was added in 8.0.11.

- Error number: MY-011768; Symbol: ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_INIT_STATE; SQLSTATE: HY000

  Message: Pool initialization failed: pool is initializing

  ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_INIT_STATE was added in 8.0.11.

- Error number: MY-011769; Symbol: ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_DEINIT_STATE; SQLSTATE: HY000

  Message: Pool initialization failed: pool is de-initializing

  ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_DEINIT_STATE was added in 8.0.11.

- Error number: MY-011770; Symbol: ER_LDAP_AUTH_FAILED_TO_DEINITIALIZE_POOL_IN_RECONSTRUCT_STATE; SQLSTATE: HY000

  Message: Failed to pool deinitialized: pool is already reconstructing

  ER_LDAP_AUTH_FAILED_TO_DEINITIALIZE_POOL_IN_RECONSTRUCT_STATE was added in 8.0.11.

- Error number: MY-011771; Symbol: ER_LDAP_AUTH_FAILED_TO_DEINITIALIZE_NOT_READY_POOL; SQLSTATE: HY000

  Message: Failed to pool deinitialized : pool is not ready

  ER_LDAP_AUTH_FAILED_TO_DEINITIALIZE_NOT_READY_POOL was added in 8.0.11.

- Error number: MY-011772; Symbol: ER_LDAP_AUTH_FAILED_TO_GET_CONNECTION_AS_PLUGIN_NOT_READY; SQLSTATE: HY000

  Message: Ldap_connection_pool::get: Failed to return connection as plug-in is not ready/initializing/de-initializing

  ER_LDAP_AUTH_FAILED_TO_GET_CONNECTION_AS_PLUGIN_NOT_READY was added in 8.0.11.

- Error number: MY-011773; Symbol: ER_LDAP_AUTH_CONNECTION_POOL_INIT_FAILED; SQLSTATE: HY000

  Message: Connection pool has failed to initialized

ER_LDAP_AUTH_CONNECTION_POOL_INIT_FAILED was added in 8.0.11.

- Error number: MY-011774; Symbol: ER_LDAP_AUTH_MAX_ALLOWED_CONNECTION_LIMIT_HIT; SQLSTATE: HY000

  Message: Ldap_connetion_pool::get LDAP maximum connection allowed size is reached. Increase the maximum limit.

  ER_LDAP_AUTH_MAX_ALLOWED_CONNECTION_LIMIT_HIT was added in 8.0.11.

- Error number: MY-011775; Symbol: ER_LDAP_AUTH_MAX_POOL_SIZE_SET_FAILED; SQLSTATE: HY000

  Message: Set max pool size failed.

  ER_LDAP_AUTH_MAX_POOL_SIZE_SET_FAILED was added in 8.0.11.

- Error number: MY-011776; Symbol: ER_LDAP_AUTH_PLUGIN_FAILED_TO_READ_PACKET; SQLSTATE: HY000

  Message: Plug-in has failed to read the packet from client

  ER_LDAP_AUTH_PLUGIN_FAILED_TO_READ_PACKET was added in 8.0.11.

- Error number: MY-011777; Symbol: ER_LDAP_AUTH_CREATING_LDAP_CONNECTION; SQLSTATE: HY000

  Message: Ldap_authentication::initialize: creating new LDAP connection.

  ER_LDAP_AUTH_CREATING_LDAP_CONNECTION was added in 8.0.11.

- Error number: MY-011778; Symbol: ER_LDAP_AUTH_GETTING_CONNECTION_FROM_POOL; SQLSTATE: HY000

  Message: Ldap_authentication::initialize: getting connection from pool.

  ER_LDAP_AUTH_GETTING_CONNECTION_FROM_POOL was added in 8.0.11.

- Error number: MY-011779; Symbol: ER_LDAP_AUTH_RETURNING_CONNECTION_TO_POOL; SQLSTATE: HY000

  Message: Ldap_authentication::de_initialize putting back connection in the pool

  ER_LDAP_AUTH_RETURNING_CONNECTION_TO_POOL was added in 8.0.11.

- Error number: MY-011780; Symbol: ER_LDAP_AUTH_SEARCH_USER_GROUP_ATTR_NOT_FOUND; SQLSTATE: HY000

  Message: Ldap_authentication::search_user_group no group attribute found

  ER_LDAP_AUTH_SEARCH_USER_GROUP_ATTR_NOT_FOUND was added in 8.0.11.

- Error number: MY-011781; Symbol: ER_LDAP_AUTH_LDAP_INFO_NULL; SQLSTATE: HY000

  Message: Ldap_connetion_pool::put ldap info null

  ER_LDAP_AUTH_LDAP_INFO_NULL was added in 8.0.11.

- Error number: MY-011782; Symbol: ER_LDAP_AUTH_FREEING_CONNECTION; SQLSTATE: HY000

  Message: Ldap_connection_pool::put connection is freeing.

  ER_LDAP_AUTH_FREEING_CONNECTION was added in 8.0.11.

- Error number: MY-011783; Symbol: ER_LDAP_AUTH_CONNECTION_PUSHED_TO_POOL; SQLSTATE: HY000

  Message: Ldap_connection_pool::put connection in pushed in the pool

  ER_LDAP_AUTH_CONNECTION_PUSHED_TO_POOL was added in 8.0.11.

- Error number: MY-011784; Symbol: ER_LDAP_AUTH_CONNECTION_CREATOR_ENTER; SQLSTATE: HY000

  Message: Ldap_connection_creator::Ldap_connection_creator

  ER_LDAP_AUTH_CONNECTION_CREATOR_ENTER was added in 8.0.11.

- Error number: MY-011785; Symbol: ER_LDAP_AUTH_STARTING_TLS; SQLSTATE: HY000

  Message: starting TLS

  ER_LDAP_AUTH_STARTING_TLS was added in 8.0.11.

- Error number: MY-011786; Symbol: ER_LDAP_AUTH_CONNECTION_GET_LDAP_INFO_NULL; SQLSTATE: HY000

  Message: Ldap_connection_pool::get: (ldap_info == NULL)|| (*ldap_info)

  ER_LDAP_AUTH_CONNECTION_GET_LDAP_INFO_NULL was added in 8.0.11.

- Error number: MY-011787; Symbol: ER_LDAP_AUTH_DELETING_CONNECTION_KEY; SQLSTATE: HY000

  Message: Ldap_connection_pool::deinit: deleting connection key %s

  ER_LDAP_AUTH_DELETING_CONNECTION_KEY was added in 8.0.11.

- Error number: MY-011788; Symbol: ER_LDAP_AUTH_POOLED_CONNECTION_KEY; SQLSTATE: HY000

  Message: Ldap_connection_pool::get pooled connection key: %s

  ER_LDAP_AUTH_POOLED_CONNECTION_KEY was added in 8.0.11.

- Error number: MY-011789; Symbol: ER_LDAP_AUTH_CREATE_CONNECTION_KEY; SQLSTATE: HY000

  Message: Ldap_connection_pool::get create connection key: %s

  ER_LDAP_AUTH_CREATE_CONNECTION_KEY was added in 8.0.11.

- Error number: MY-011790; Symbol: ER_LDAP_AUTH_COMMUNICATION_HOST_INFO; SQLSTATE: HY000

  Message: LDAP communication host %s port %u

ER_LDAP_AUTH_COMMUNICATION_HOST_INFO was added in 8.0.11.

- Error number: MY-011791; Symbol: ER_LDAP_AUTH_METHOD_TO_CLIENT; SQLSTATE: HY000

  Message: Sending authentication method to client : %s

  ER_LDAP_AUTH_METHOD_TO_CLIENT was added in 8.0.11.

- Error number: MY-011792; Symbol: ER_LDAP_AUTH_SASL_REQUEST_FROM_CLIENT; SQLSTATE: HY000

  Message: SASL request received from mysql client: %s

  ER_LDAP_AUTH_SASL_REQUEST_FROM_CLIENT was added in 8.0.11.

- Error number: MY-011793; Symbol: ER_LDAP_AUTH_SASL_PROCESS_SASL; SQLSTATE: HY000

  Message: Ldap_sasl_authentication::process_sasl rc: %s

  ER_LDAP_AUTH_SASL_PROCESS_SASL was added in 8.0.11.

- Error number: MY-011794; Symbol: ER_LDAP_AUTH_SASL_BIND_SUCCESS_INFO; SQLSTATE: HY000

  Message: Ldap_sasl_authentication::process_sasl sasl bind succeed. dn: %s method: %s server credential: %s

  ER_LDAP_AUTH_SASL_BIND_SUCCESS_INFO was added in 8.0.11.

- Error number: MY-011795; Symbol: ER_LDAP_AUTH_STARTED_FOR_USER; SQLSTATE: HY000

  Message: LDAP authentication started for user name: %s

  ER_LDAP_AUTH_STARTED_FOR_USER was added in 8.0.11.

- Error number: MY-011796; Symbol: ER_LDAP_AUTH_DISTINGUISHED_NAME; SQLSTATE: HY000

  Message: %s

  ER_LDAP_AUTH_DISTINGUISHED_NAME was added in 8.0.11.

- Error number: MY-011797; Symbol: ER_LDAP_AUTH_INIT_FAILED; SQLSTATE: HY000

  Message: LDAP authentication initialize is failed with: %s

  ER_LDAP_AUTH_INIT_FAILED was added in 8.0.11.

- Error number: MY-011798; Symbol: ER_LDAP_AUTH_OR_GROUP_RETRIEVAL_FAILED; SQLSTATE: HY000

  Message: LDAP authentication failed or group retrieval failed: %s

  ER_LDAP_AUTH_OR_GROUP_RETRIEVAL_FAILED was added in 8.0.11.

- Error number: MY-011799; Symbol: ER_LDAP_AUTH_USER_GROUP_SEARCH_FAILED; SQLSTATE: HY000

  Message: Search user group has failed: %s

ER_LDAP_AUTH_USER_GROUP_SEARCH_FAILED was added in 8.0.11.

- Error number: MY-011800; Symbol: ER_LDAP_AUTH_USER_BIND_FAILED; SQLSTATE: HY000

  Message: LDAP user bind has failed: %s

  ER_LDAP_AUTH_USER_BIND_FAILED was added in 8.0.11.

- Error number: MY-011801; Symbol: ER_LDAP_AUTH_POOL_GET_FAILED_TO_CREATE_CONNECTION; SQLSTATE: HY000

  Message: Connection pool get: Failed to create LDAP connection. %s

  ER_LDAP_AUTH_POOL_GET_FAILED_TO_CREATE_CONNECTION was added in 8.0.11.

- Error number: MY-011802; Symbol: ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_CONNECTION; SQLSTATE: HY000

  Message: Failed to create new LDAP connection: %s

  ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_CONNECTION was added in 8.0.11.

- Error number: MY-011803; Symbol: ER_LDAP_AUTH_FAILED_TO_ESTABLISH_TLS_CONNECTION; SQLSTATE: HY000

  Message: Failed to establish TLS connection: %s

  ER_LDAP_AUTH_FAILED_TO_ESTABLISH_TLS_CONNECTION was added in 8.0.11.

- Error number: MY-011804; Symbol: ER_LDAP_AUTH_FAILED_TO_SEARCH_DN; SQLSTATE: HY000

  Message: Failed to search user full dn: %s

  ER_LDAP_AUTH_FAILED_TO_SEARCH_DN was added in 8.0.11.

- Error number: MY-011805; Symbol: ER_LDAP_AUTH_CONNECTION_POOL_REINIT_ENTER; SQLSTATE: HY000

  Message: Ldap_connection_pool::reinit

  ER_LDAP_AUTH_CONNECTION_POOL_REINIT_ENTER was added in 8.0.11.

- Error number: MY-011806; Symbol: ER_SYSTEMD_NOTIFY_PATH_TOO_LONG; SQLSTATE: HY000

  Message: The path '%s', from the NOTIFY_SOCKET environment variable, is too long. At %u bytes it exceeds the limit of %u bytes for an AF_UNIX socket.

  ER_SYSTEMD_NOTIFY_PATH_TOO_LONG was added in 8.0.11.

- Error number: MY-011807; Symbol: ER_SYSTEMD_NOTIFY_CONNECT_FAILED; SQLSTATE: HY000

  Message: Failed to connect to systemd notification socket named %s. Error: '%s'

  ER_SYSTEMD_NOTIFY_CONNECT_FAILED was added in 8.0.11.

- Error number: MY-011808; Symbol: ER_SYSTEMD_NOTIFY_WRITE_FAILED; SQLSTATE: HY000

  Message: Failed to write '%s' to systemd notification. Error: '%s'

ER_SYSTEMD_NOTIFY_WRITE_FAILED was added in 8.0.11.

- Error number: MY-011809; Symbol: ER_FOUND_MISSING_GTIDS; SQLSTATE: HY000

  Message: Cannot replicate to server with server_uuid='%s' because the present server has purged required binary logs. The connecting server needs to replicate the missing transactions from elsewhere, or be replaced by a new server created from a more recent backup. To prevent this error in the future, consider increasing the binary log expiration period on the present server. The missing transactions are '%s'.

  ER_FOUND_MISSING_GTIDS was added in 8.0.11.

- Error number: MY-011810; Symbol: ER_PID_FILE_PRIV_DIRECTORY_INSECURE; SQLSTATE: HY000

  Message: Insecure configuration for --pid-file: Location '%s' in the path is accessible to all OS users. Consider choosing a different directory.

  ER_PID_FILE_PRIV_DIRECTORY_INSECURE was added in 8.0.11.

- Error number: MY-011811; Symbol: ER_CANT_CHECK_PID_PATH; SQLSTATE: HY000

  Message: Can't start server: can't check PID filepath: %s

  ER_CANT_CHECK_PID_PATH was added in 8.0.11.

- Error number: MY-011812; Symbol: ER_VALIDATE_PWD_STATUS_VAR_REGISTRATION_FAILED; SQLSTATE: HY000

  Message: validate_password status variables registration failed.

  ER_VALIDATE_PWD_STATUS_VAR_REGISTRATION_FAILED was added in 8.0.11.

- Error number: MY-011813; Symbol: ER_VALIDATE_PWD_STATUS_VAR_UNREGISTRATION_FAILED; SQLSTATE: HY000

  Message: validate_password status variables unregistration failed.

  ER_VALIDATE_PWD_STATUS_VAR_UNREGISTRATION_FAILED was added in 8.0.11.

- Error number: MY-011814; Symbol: ER_VALIDATE_PWD_DICT_FILE_OPEN_FAILED; SQLSTATE: HY000

  Message: Dictionary file open failed

  ER_VALIDATE_PWD_DICT_FILE_OPEN_FAILED was added in 8.0.11.

- Error number: MY-011815; Symbol: ER_VALIDATE_PWD_COULD_BE_NULL; SQLSTATE: HY000

  Message: given password string could be null

  ER_VALIDATE_PWD_COULD_BE_NULL was added in 8.0.11.

- Error number: MY-011816; Symbol: ER_VALIDATE_PWD_STRING_CONV_TO_LOWERCASE_FAILED; SQLSTATE: HY000

  Message: failed to convert the password string to lower case

ER_VALIDATE_PWD_STRING_CONV_TO_LOWERCASE_FAILED was added in 8.0.11.

- Error number: MY-011817; Symbol: ER_VALIDATE_PWD_STRING_CONV_TO_BUFFER_FAILED; SQLSTATE: HY000

  Message: failed to convert the password string into a buffer

  ER_VALIDATE_PWD_STRING_CONV_TO_BUFFER_FAILED was added in 8.0.11.

- Error number: MY-011818; Symbol: ER_VALIDATE_PWD_STRING_HANDLER_MEM_ALLOCATION_FAILED; SQLSTATE: HY000

  Message: memory allocation failed for string handler

  ER_VALIDATE_PWD_STRING_HANDLER_MEM_ALLOCATION_FAILED was added in 8.0.11.

- Error number: MY-011819; Symbol: ER_VALIDATE_PWD_STRONG_POLICY_DICT_FILE_UNSPECIFIED; SQLSTATE: HY000

  Message: Since the validate_password_policy is mentioned as Strong, dictionary file must be specified

  ER_VALIDATE_PWD_STRONG_POLICY_DICT_FILE_UNSPECIFIED was added in 8.0.11.

- Error number: MY-011820; Symbol: ER_VALIDATE_PWD_CONVERT_TO_BUFFER_FAILED; SQLSTATE: HY000

  Message: convert_to_buffer service failed

  ER_VALIDATE_PWD_CONVERT_TO_BUFFER_FAILED was added in 8.0.11.

- Error number: MY-011821; Symbol: ER_VALIDATE_PWD_VARIABLE_REGISTRATION_FAILED; SQLSTATE: HY000

  Message: %s variable registration failed.

  ER_VALIDATE_PWD_VARIABLE_REGISTRATION_FAILED was added in 8.0.11.

- Error number: MY-011822; Symbol: ER_VALIDATE_PWD_VARIABLE_UNREGISTRATION_FAILED; SQLSTATE: HY000

  Message: %s variable unregistration failed.

  ER_VALIDATE_PWD_VARIABLE_UNREGISTRATION_FAILED was added in 8.0.11.

- Error number: MY-011823; Symbol: ER_KEYRING_MIGRATION_EXTRA_OPTIONS; SQLSTATE: HY000

  Message: Please specify options specific to keyring migration. Any additional options can be ignored. NOTE: Although some options are valid, migration tool can still report error example: plugin variables for which plugin is not loaded yet.

  ER_KEYRING_MIGRATION_EXTRA_OPTIONS was added in 8.0.11.

- Error number: MY-011825; Symbol: ER_IB_MSG_0; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_0 was added in 8.0.11.

- Error number: MY-011826; Symbol: ER_IB_MSG_1; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1 was added in 8.0.11.
- Error number: MY-011827; Symbol: ER_IB_MSG_2; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_2 was added in 8.0.11.
- Error number: MY-011828; Symbol: ER_IB_MSG_3; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_3 was added in 8.0.11.
- Error number: MY-011829; Symbol: ER_IB_MSG_4; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_4 was added in 8.0.11.
- Error number: MY-011830; Symbol: ER_IB_MSG_5; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_5 was added in 8.0.11.
- Error number: MY-011831; Symbol: ER_IB_MSG_6; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_6 was added in 8.0.11.
- Error number: MY-011832; Symbol: ER_IB_MSG_7; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_7 was added in 8.0.11.
- Error number: MY-011833; Symbol: ER_IB_MSG_8; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_8 was added in 8.0.11.
- Error number: MY-011834; Symbol: ER_IB_MSG_9; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_9 was added in 8.0.11.
- Error number: MY-011835; Symbol: ER_IB_MSG_10; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_10 was added in 8.0.11.

- Error number: MY-011836; Symbol: ER_IB_MSG_11; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_11 was added in 8.0.11.
- Error number: MY-011837; Symbol: ER_IB_MSG_12; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_12 was added in 8.0.11.
- Error number: MY-011838; Symbol: ER_IB_MSG_13; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_13 was added in 8.0.11.
- Error number: MY-011839; Symbol: ER_IB_MSG_14; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_14 was added in 8.0.11.
- Error number: MY-011840; Symbol: ER_IB_MSG_15; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_15 was added in 8.0.11.
- Error number: MY-011841; Symbol: ER_IB_MSG_16; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_16 was added in 8.0.11.
- Error number: MY-011842; Symbol: ER_IB_MSG_17; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_17 was added in 8.0.11.
- Error number: MY-011843; Symbol: ER_IB_MSG_18; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_18 was added in 8.0.11.
- Error number: MY-011844; Symbol: ER_IB_MSG_19; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_19 was added in 8.0.11.
- Error number: MY-011845; Symbol: ER_IB_MSG_20; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_20 was added in 8.0.11.

- Error number: MY-011846; Symbol: ER_IB_MSG_21; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_21 was added in 8.0.11.
- Error number: MY-011847; Symbol: ER_IB_MSG_22; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_22 was added in 8.0.11.
- Error number: MY-011848; Symbol: ER_IB_MSG_23; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_23 was added in 8.0.11.
- Error number: MY-011849; Symbol: ER_IB_MSG_24; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_24 was added in 8.0.11.
- Error number: MY-011850; Symbol: ER_IB_MSG_25; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_25 was added in 8.0.11.
- Error number: MY-011851; Symbol: ER_IB_MSG_26; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_26 was added in 8.0.11.
- Error number: MY-011852; Symbol: ER_IB_MSG_27; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_27 was added in 8.0.11.
- Error number: MY-011853; Symbol: ER_IB_MSG_28; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_28 was added in 8.0.11.
- Error number: MY-011854; Symbol: ER_IB_MSG_29; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_29 was added in 8.0.11.
- Error number: MY-011855; Symbol: ER_IB_MSG_30; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_30 was added in 8.0.11.

- Error number: MY-011856; Symbol: ER_IB_MSG_31; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_31 was added in 8.0.11.
- Error number: MY-011857; Symbol: ER_IB_MSG_32; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_32 was added in 8.0.11.
- Error number: MY-011858; Symbol: ER_IB_MSG_33; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_33 was added in 8.0.11.
- Error number: MY-011859; Symbol: ER_IB_MSG_34; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_34 was added in 8.0.11.
- Error number: MY-011860; Symbol: ER_IB_MSG_35; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_35 was added in 8.0.11.
- Error number: MY-011861; Symbol: ER_IB_MSG_36; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_36 was added in 8.0.11.
- Error number: MY-011862; Symbol: ER_IB_MSG_37; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_37 was added in 8.0.11.
- Error number: MY-011863; Symbol: ER_IB_MSG_38; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_38 was added in 8.0.11.
- Error number: MY-011864; Symbol: ER_IB_MSG_39; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_39 was added in 8.0.11.
- Error number: MY-011865; Symbol: ER_IB_MSG_40; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_40 was added in 8.0.11.

- Error number: MY-011866; Symbol: ER_IB_MSG_41; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_41 was added in 8.0.11.
- Error number: MY-011867; Symbol: ER_IB_MSG_42; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_42 was added in 8.0.11.
- Error number: MY-011868; Symbol: ER_IB_MSG_43; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_43 was added in 8.0.11.
- Error number: MY-011869; Symbol: ER_IB_MSG_44; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_44 was added in 8.0.11.
- Error number: MY-011870; Symbol: ER_IB_MSG_45; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_45 was added in 8.0.11.
- Error number: MY-011871; Symbol: ER_IB_MSG_46; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_46 was added in 8.0.11.
- Error number: MY-011872; Symbol: ER_IB_MSG_47; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_47 was added in 8.0.11.
- Error number: MY-011873; Symbol: ER_IB_MSG_48; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_48 was added in 8.0.11.
- Error number: MY-011874; Symbol: ER_IB_MSG_49; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_49 was added in 8.0.11.
- Error number: MY-011875; Symbol: ER_IB_MSG_50; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_50 was added in 8.0.11.

- Error number: MY-011876; Symbol: ER_IB_MSG_51; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_51 was added in 8.0.11.
- Error number: MY-011877; Symbol: ER_IB_MSG_52; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_52 was added in 8.0.11.
- Error number: MY-011878; Symbol: ER_IB_MSG_53; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_53 was added in 8.0.11.
- Error number: MY-011879; Symbol: ER_IB_MSG_54; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_54 was added in 8.0.11.
- Error number: MY-011880; Symbol: ER_IB_MSG_55; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_55 was added in 8.0.11.
- Error number: MY-011881; Symbol: ER_IB_MSG_56; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_56 was added in 8.0.11.
- Error number: MY-011882; Symbol: ER_IB_MSG_57; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_57 was added in 8.0.11.
- Error number: MY-011883; Symbol: ER_IB_MSG_58; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_58 was added in 8.0.11.
- Error number: MY-011884; Symbol: ER_IB_MSG_59; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_59 was added in 8.0.11.
- Error number: MY-011885; Symbol: ER_IB_MSG_60; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_60 was added in 8.0.11.

- Error number: MY-011886; Symbol: ER_IB_MSG_61; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_61 was added in 8.0.11.
- Error number: MY-011887; Symbol: ER_IB_MSG_62; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_62 was added in 8.0.11.
- Error number: MY-011888; Symbol: ER_IB_MSG_63; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_63 was added in 8.0.11.
- Error number: MY-011889; Symbol: ER_IB_MSG_64; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_64 was added in 8.0.11.
- Error number: MY-011890; Symbol: ER_IB_MSG_65; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_65 was added in 8.0.11.
- Error number: MY-011891; Symbol: ER_IB_MSG_66; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_66 was added in 8.0.11.
- Error number: MY-011892; Symbol: ER_IB_MSG_67; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_67 was added in 8.0.11.
- Error number: MY-011893; Symbol: ER_IB_MSG_68; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_68 was added in 8.0.11.
- Error number: MY-011894; Symbol: ER_IB_MSG_69; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_69 was added in 8.0.11.
- Error number: MY-011895; Symbol: ER_IB_MSG_70; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_70 was added in 8.0.11.

- Error number: MY-011896; Symbol: ER_IB_MSG_71; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_71 was added in 8.0.11.
- Error number: MY-011897; Symbol: ER_IB_MSG_72; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_72 was added in 8.0.11.
- Error number: MY-011898; Symbol: ER_IB_MSG_73; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_73 was added in 8.0.11.
- Error number: MY-011899; Symbol: ER_IB_MSG_74; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_74 was added in 8.0.11.
- Error number: MY-011900; Symbol: ER_IB_MSG_75; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_75 was added in 8.0.11.
- Error number: MY-011901; Symbol: ER_IB_MSG_76; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_76 was added in 8.0.11.
- Error number: MY-011902; Symbol: ER_IB_MSG_77; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_77 was added in 8.0.11.
- Error number: MY-011903; Symbol: ER_IB_MSG_78; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_78 was added in 8.0.11.
- Error number: MY-011904; Symbol: ER_IB_MSG_79; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_79 was added in 8.0.11.
- Error number: MY-011905; Symbol: ER_IB_MSG_80; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_80 was added in 8.0.11.

- Error number: MY-011906; Symbol: ER_IB_MSG_81; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_81 was added in 8.0.11.
- Error number: MY-011907; Symbol: ER_IB_MSG_82; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_82 was added in 8.0.11.
- Error number: MY-011908; Symbol: ER_IB_MSG_83; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_83 was added in 8.0.11.
- Error number: MY-011909; Symbol: ER_IB_MSG_84; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_84 was added in 8.0.11.
- Error number: MY-011910; Symbol: ER_IB_MSG_85; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_85 was added in 8.0.11.
- Error number: MY-011911; Symbol: ER_IB_MSG_86; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_86 was added in 8.0.11.
- Error number: MY-011912; Symbol: ER_IB_MSG_87; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_87 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011913; Symbol: ER_IB_MSG_88; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_88 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011914; Symbol: ER_IB_MSG_89; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_89 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011915; Symbol: ER_IB_MSG_90; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_90 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-011916; Symbol: ER_IB_MSG_91; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_91 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011917; Symbol: ER_IB_MSG_92; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_92 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011918; Symbol: ER_IB_MSG_93; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_93 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011919; Symbol: ER_IB_MSG_94; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_94 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011920; Symbol: ER_IB_MSG_95; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_95 was added in 8.0.11.
- Error number: MY-011921; Symbol: ER_IB_MSG_96; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_96 was added in 8.0.11.
- Error number: MY-011922; Symbol: ER_IB_MSG_97; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_97 was added in 8.0.11.
- Error number: MY-011923; Symbol: ER_IB_MSG_98; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_98 was added in 8.0.11.
- Error number: MY-011924; Symbol: ER_IB_MSG_99; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_99 was added in 8.0.11.
- Error number: MY-011925; Symbol: ER_IB_MSG_100; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_100 was added in 8.0.11.

- Error number: MY-011926; Symbol: ER_IB_MSG_101; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_101 was added in 8.0.11.
- Error number: MY-011927; Symbol: ER_IB_MSG_102; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_102 was added in 8.0.11.
- Error number: MY-011928; Symbol: ER_IB_MSG_103; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_103 was added in 8.0.11.
- Error number: MY-011929; Symbol: ER_IB_MSG_104; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_104 was added in 8.0.11.
- Error number: MY-011930; Symbol: ER_IB_MSG_105; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_105 was added in 8.0.11.
- Error number: MY-011931; Symbol: ER_IB_MSG_106; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_106 was added in 8.0.11.
- Error number: MY-011932; Symbol: ER_IB_MSG_107; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_107 was added in 8.0.11.
- Error number: MY-011933; Symbol: ER_IB_MSG_108; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_108 was added in 8.0.11.
- Error number: MY-011934; Symbol: ER_IB_MSG_109; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_109 was added in 8.0.11.
- Error number: MY-011935; Symbol: ER_IB_MSG_110; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_110 was added in 8.0.11.

- Error number: MY-011936; Symbol: ER_IB_MSG_111; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_111 was added in 8.0.11.
- Error number: MY-011937; Symbol: ER_IB_MSG_112; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_112 was added in 8.0.11.
- Error number: MY-011938; Symbol: ER_IB_MSG_113; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_113 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011939; Symbol: ER_IB_MSG_114; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_114 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011940; Symbol: ER_IB_MSG_115; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_115 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011941; Symbol: ER_IB_MSG_116; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_116 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011942; Symbol: ER_IB_MSG_117; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_117 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011943; Symbol: ER_IB_MSG_118; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_118 was added in 8.0.11, removed after 8.0.13.
- Error number: MY-011944; Symbol: ER_IB_MSG_119; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_119 was added in 8.0.11.
- Error number: MY-011945; Symbol: ER_IB_MSG_120; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_120 was added in 8.0.11.

- Error number: MY-011946; Symbol: ER_IB_MSG_121; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_121 was added in 8.0.11.
- Error number: MY-011947; Symbol: ER_IB_MSG_122; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_122 was added in 8.0.11.
- Error number: MY-011948; Symbol: ER_IB_MSG_123; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_123 was added in 8.0.11.
- Error number: MY-011949; Symbol: ER_IB_MSG_124; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_124 was added in 8.0.11.
- Error number: MY-011950; Symbol: ER_IB_MSG_125; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_125 was added in 8.0.11.
- Error number: MY-011951; Symbol: ER_IB_MSG_126; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_126 was added in 8.0.11.
- Error number: MY-011952; Symbol: ER_IB_MSG_127; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_127 was added in 8.0.11.
- Error number: MY-011953; Symbol: ER_IB_MSG_128; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_128 was added in 8.0.11.
- Error number: MY-011954; Symbol: ER_IB_MSG_129; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_129 was added in 8.0.11.
- Error number: MY-011955; Symbol: ER_IB_MSG_130; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_130 was added in 8.0.11.

- Error number: MY-011956; Symbol: ER_IB_MSG_131; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_131 was added in 8.0.11.
- Error number: MY-011957; Symbol: ER_IB_MSG_132; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_132 was added in 8.0.11.
- Error number: MY-011958; Symbol: ER_IB_MSG_133; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_133 was added in 8.0.11.
- Error number: MY-011959; Symbol: ER_IB_MSG_134; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_134 was added in 8.0.11.
- Error number: MY-011960; Symbol: ER_IB_MSG_135; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_135 was added in 8.0.11.
- Error number: MY-011961; Symbol: ER_IB_MSG_136; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_136 was added in 8.0.11.
- Error number: MY-011962; Symbol: ER_IB_MSG_137; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_137 was added in 8.0.11.
- Error number: MY-011963; Symbol: ER_IB_MSG_138; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_138 was added in 8.0.11.
- Error number: MY-011964; Symbol: ER_IB_MSG_139; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_139 was added in 8.0.11.
- Error number: MY-011965; Symbol: ER_IB_MSG_140; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_140 was added in 8.0.11.

- Error number: MY-011966; Symbol: ER_IB_MSG_141; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_141 was added in 8.0.11.
- Error number: MY-011967; Symbol: ER_IB_MSG_142; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_142 was added in 8.0.11.
- Error number: MY-011968; Symbol: ER_IB_MSG_143; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_143 was added in 8.0.11.
- Error number: MY-011969; Symbol: ER_IB_MSG_144; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_144 was added in 8.0.11.
- Error number: MY-011970; Symbol: ER_IB_MSG_145; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_145 was added in 8.0.11.
- Error number: MY-011971; Symbol: ER_IB_MSG_146; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_146 was added in 8.0.11.
- Error number: MY-011972; Symbol: ER_IB_MSG_147; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_147 was added in 8.0.11.
- Error number: MY-011973; Symbol: ER_IB_MSG_148; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_148 was added in 8.0.11.
- Error number: MY-011974; Symbol: ER_IB_MSG_149; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_149 was added in 8.0.11, removed after 8.0.16.
- Error number: MY-011974; Symbol: ER_IB_CLONE_INTERNAL; SQLSTATE: HY000

  Message: %s

  ER_IB_CLONE_INTERNAL was added in 8.0.17.

- Error number: MY-011975; Symbol: ER_IB_MSG_150; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_150 was added in 8.0.11, removed after 8.0.16.

- Error number: MY-011975; Symbol: ER_IB_CLONE_TIMEOUT; SQLSTATE: HY000

  Message: %s

  ER_IB_CLONE_TIMEOUT was added in 8.0.17.

- Error number: MY-011976; Symbol: ER_IB_MSG_151; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_151 was added in 8.0.11, removed after 8.0.16.

- Error number: MY-011976; Symbol: ER_IB_CLONE_STATUS_FILE; SQLSTATE: HY000

  Message: %s

  ER_IB_CLONE_STATUS_FILE was added in 8.0.17.

- Error number: MY-011977; Symbol: ER_IB_MSG_152; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_152 was added in 8.0.11, removed after 8.0.16.

- Error number: MY-011977; Symbol: ER_IB_CLONE_SQL; SQLSTATE: HY000

  Message: %s

  ER_IB_CLONE_SQL was added in 8.0.17.

- Error number: MY-011978; Symbol: ER_IB_MSG_153; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_153 was added in 8.0.11, removed after 8.0.16.

- Error number: MY-011978; Symbol: ER_IB_CLONE_VALIDATE; SQLSTATE: HY000

  Message: %s

  ER_IB_CLONE_VALIDATE was added in 8.0.17.

- Error number: MY-011979; Symbol: ER_IB_MSG_154; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_154 was added in 8.0.11, removed after 8.0.16.

- Error number: MY-011979; Symbol: ER_IB_CLONE_PUNCH_HOLE; SQLSTATE: HY000

  Message: %s

  ER_IB_CLONE_PUNCH_HOLE was added in 8.0.17.

- Error number: MY-011980; Symbol: ER_IB_MSG_155; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_155 was added in 8.0.11, removed after 8.0.16.

- Error number: MY-011980; Symbol: ER_IB_CLONE_GTID_PERSIST; SQLSTATE: HY000

  Message: %s

  ER_IB_CLONE_GTID_PERSIST was added in 8.0.17.

- Error number: MY-011981; Symbol: ER_IB_MSG_156; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_156 was added in 8.0.11.

- Error number: MY-011982; Symbol: ER_IB_MSG_157; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_157 was added in 8.0.11.

- Error number: MY-011983; Symbol: ER_IB_MSG_158; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_158 was added in 8.0.11.

- Error number: MY-011984; Symbol: ER_IB_MSG_159; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_159 was added in 8.0.11.

- Error number: MY-011985; Symbol: ER_IB_MSG_160; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_160 was added in 8.0.11.

- Error number: MY-011986; Symbol: ER_IB_MSG_161; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_161 was added in 8.0.11.

- Error number: MY-011987; Symbol: ER_IB_MSG_162; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_162 was added in 8.0.11.

- Error number: MY-011988; Symbol: ER_IB_MSG_163; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_163 was added in 8.0.11.

- Error number: MY-011989; Symbol: ER_IB_MSG_164; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_164 was added in 8.0.11.
- Error number: MY-011990; Symbol: ER_IB_MSG_165; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_165 was added in 8.0.11.
- Error number: MY-011991; Symbol: ER_IB_MSG_166; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_166 was added in 8.0.11.
- Error number: MY-011992; Symbol: ER_IB_MSG_167; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_167 was added in 8.0.11.
- Error number: MY-011993; Symbol: ER_IB_MSG_168; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_168 was added in 8.0.11.
- Error number: MY-011994; Symbol: ER_IB_MSG_169; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_169 was added in 8.0.11.
- Error number: MY-011995; Symbol: ER_IB_MSG_170; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_170 was added in 8.0.11.
- Error number: MY-011996; Symbol: ER_IB_MSG_171; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_171 was added in 8.0.11.
- Error number: MY-011997; Symbol: ER_IB_MSG_172; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_172 was added in 8.0.11.
- Error number: MY-011998; Symbol: ER_IB_MSG_173; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_173 was added in 8.0.11.

- Error number: MY-011999; Symbol: ER_IB_MSG_174; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_174 was added in 8.0.11.
- Error number: MY-012000; Symbol: ER_IB_MSG_175; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_175 was added in 8.0.11.
- Error number: MY-012001; Symbol: ER_IB_MSG_176; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_176 was added in 8.0.11.
- Error number: MY-012002; Symbol: ER_IB_MSG_177; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_177 was added in 8.0.11.
- Error number: MY-012003; Symbol: ER_IB_MSG_178; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_178 was added in 8.0.11.
- Error number: MY-012004; Symbol: ER_IB_MSG_179; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_179 was added in 8.0.11.
- Error number: MY-012005; Symbol: ER_IB_MSG_180; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_180 was added in 8.0.11.
- Error number: MY-012006; Symbol: ER_IB_MSG_181; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_181 was added in 8.0.11.
- Error number: MY-012007; Symbol: ER_IB_MSG_182; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_182 was added in 8.0.11.
- Error number: MY-012008; Symbol: ER_IB_MSG_183; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_183 was added in 8.0.11.

- Error number: MY-012009; Symbol: ER_IB_MSG_184; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_184 was added in 8.0.11.
- Error number: MY-012010; Symbol: ER_IB_MSG_185; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_185 was added in 8.0.11.
- Error number: MY-012011; Symbol: ER_IB_MSG_186; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_186 was added in 8.0.11.
- Error number: MY-012012; Symbol: ER_IB_MSG_187; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_187 was added in 8.0.11.
- Error number: MY-012013; Symbol: ER_IB_MSG_188; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_188 was added in 8.0.11.
- Error number: MY-012014; Symbol: ER_IB_MSG_189; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_189 was added in 8.0.11.
- Error number: MY-012015; Symbol: ER_IB_MSG_190; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_190 was added in 8.0.11.
- Error number: MY-012016; Symbol: ER_IB_MSG_191; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_191 was added in 8.0.11.
- Error number: MY-012017; Symbol: ER_IB_MSG_192; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_192 was added in 8.0.11.
- Error number: MY-012018; Symbol: ER_IB_MSG_193; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_193 was added in 8.0.11.

- Error number: MY-012019; Symbol: ER_IB_MSG_194; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_194 was added in 8.0.11.
- Error number: MY-012020; Symbol: ER_IB_MSG_195; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_195 was added in 8.0.11.
- Error number: MY-012021; Symbol: ER_IB_MSG_196; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_196 was added in 8.0.11.
- Error number: MY-012022; Symbol: ER_IB_MSG_197; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_197 was added in 8.0.11.
- Error number: MY-012023; Symbol: ER_IB_MSG_198; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_198 was added in 8.0.11.
- Error number: MY-012024; Symbol: ER_IB_MSG_199; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_199 was added in 8.0.11.
- Error number: MY-012025; Symbol: ER_IB_MSG_200; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_200 was added in 8.0.11.
- Error number: MY-012026; Symbol: ER_IB_MSG_201; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_201 was added in 8.0.11.
- Error number: MY-012027; Symbol: ER_IB_MSG_202; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_202 was added in 8.0.11.
- Error number: MY-012028; Symbol: ER_IB_MSG_203; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_203 was added in 8.0.11.

- Error number: MY-012029; Symbol: ER_IB_MSG_204; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_204 was added in 8.0.11.
- Error number: MY-012030; Symbol: ER_IB_MSG_205; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_205 was added in 8.0.11.
- Error number: MY-012031; Symbol: ER_IB_MSG_206; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_206 was added in 8.0.11.
- Error number: MY-012032; Symbol: ER_IB_MSG_207; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_207 was added in 8.0.11.
- Error number: MY-012033; Symbol: ER_IB_MSG_208; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_208 was added in 8.0.11.
- Error number: MY-012034; Symbol: ER_IB_MSG_209; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_209 was added in 8.0.11.
- Error number: MY-012035; Symbol: ER_IB_MSG_210; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_210 was added in 8.0.11.
- Error number: MY-012036; Symbol: ER_IB_MSG_211; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_211 was added in 8.0.11.
- Error number: MY-012037; Symbol: ER_IB_MSG_212; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_212 was added in 8.0.11.
- Error number: MY-012038; Symbol: ER_IB_MSG_213; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_213 was added in 8.0.11.

- Error number: MY-012039; Symbol: ER_IB_MSG_214; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_214 was added in 8.0.11.
- Error number: MY-012040; Symbol: ER_IB_MSG_215; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_215 was added in 8.0.11.
- Error number: MY-012041; Symbol: ER_IB_MSG_216; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_216 was added in 8.0.11.
- Error number: MY-012042; Symbol: ER_IB_MSG_217; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_217 was added in 8.0.11.
- Error number: MY-012043; Symbol: ER_IB_MSG_218; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_218 was added in 8.0.11.
- Error number: MY-012044; Symbol: ER_IB_MSG_219; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_219 was added in 8.0.11.
- Error number: MY-012045; Symbol: ER_IB_MSG_220; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_220 was added in 8.0.11.
- Error number: MY-012046; Symbol: ER_IB_MSG_221; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_221 was added in 8.0.11.
- Error number: MY-012047; Symbol: ER_IB_MSG_222; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_222 was added in 8.0.11.
- Error number: MY-012048; Symbol: ER_IB_MSG_223; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_223 was added in 8.0.11.

- Error number: MY-012049; Symbol: ER_IB_MSG_224; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_224 was added in 8.0.11.
- Error number: MY-012050; Symbol: ER_IB_MSG_225; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_225 was added in 8.0.11.
- Error number: MY-012051; Symbol: ER_IB_MSG_226; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_226 was added in 8.0.11.
- Error number: MY-012052; Symbol: ER_IB_MSG_227; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_227 was added in 8.0.11.
- Error number: MY-012053; Symbol: ER_IB_MSG_228; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_228 was added in 8.0.11.
- Error number: MY-012054; Symbol: ER_IB_MSG_229; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_229 was added in 8.0.11.
- Error number: MY-012055; Symbol: ER_IB_MSG_230; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_230 was added in 8.0.11.
- Error number: MY-012056; Symbol: ER_IB_MSG_231; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_231 was added in 8.0.11.
- Error number: MY-012057; Symbol: ER_IB_MSG_232; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_232 was added in 8.0.11.
- Error number: MY-012058; Symbol: ER_IB_MSG_233; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_233 was added in 8.0.11.

- Error number: MY-012059; Symbol: ER_IB_MSG_234; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_234 was added in 8.0.11.
- Error number: MY-012060; Symbol: ER_IB_MSG_235; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_235 was added in 8.0.11.
- Error number: MY-012061; Symbol: ER_IB_MSG_236; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_236 was added in 8.0.11.
- Error number: MY-012062; Symbol: ER_IB_MSG_237; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_237 was added in 8.0.11.
- Error number: MY-012063; Symbol: ER_IB_MSG_238; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_238 was added in 8.0.11.
- Error number: MY-012064; Symbol: ER_IB_MSG_239; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_239 was added in 8.0.11.
- Error number: MY-012065; Symbol: ER_IB_MSG_240; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_240 was added in 8.0.11.
- Error number: MY-012066; Symbol: ER_IB_MSG_241; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_241 was added in 8.0.11.
- Error number: MY-012067; Symbol: ER_IB_MSG_242; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_242 was added in 8.0.11.
- Error number: MY-012068; Symbol: ER_IB_MSG_243; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_243 was added in 8.0.11.

- Error number: MY-012069; Symbol: ER_IB_MSG_244; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_244 was added in 8.0.11.
- Error number: MY-012070; Symbol: ER_IB_MSG_245; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_245 was added in 8.0.11.
- Error number: MY-012071; Symbol: ER_IB_MSG_246; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_246 was added in 8.0.11.
- Error number: MY-012072; Symbol: ER_IB_MSG_247; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_247 was added in 8.0.11.
- Error number: MY-012073; Symbol: ER_IB_MSG_248; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_248 was added in 8.0.11.
- Error number: MY-012074; Symbol: ER_IB_MSG_249; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_249 was added in 8.0.11.
- Error number: MY-012075; Symbol: ER_IB_MSG_250; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_250 was added in 8.0.11.
- Error number: MY-012076; Symbol: ER_IB_MSG_251; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_251 was added in 8.0.11.
- Error number: MY-012077; Symbol: ER_IB_MSG_252; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_252 was added in 8.0.11.
- Error number: MY-012078; Symbol: ER_IB_MSG_253; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_253 was added in 8.0.11.

- Error number: MY-012079; Symbol: ER_IB_MSG_254; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_254 was added in 8.0.11.
- Error number: MY-012080; Symbol: ER_IB_MSG_255; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_255 was added in 8.0.11.
- Error number: MY-012081; Symbol: ER_IB_MSG_256; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_256 was added in 8.0.11.
- Error number: MY-012082; Symbol: ER_IB_MSG_257; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_257 was added in 8.0.11.
- Error number: MY-012083; Symbol: ER_IB_MSG_258; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_258 was added in 8.0.11.
- Error number: MY-012084; Symbol: ER_IB_MSG_259; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_259 was added in 8.0.11.
- Error number: MY-012085; Symbol: ER_IB_MSG_260; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_260 was added in 8.0.11.
- Error number: MY-012086; Symbol: ER_IB_MSG_261; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_261 was added in 8.0.11.
- Error number: MY-012087; Symbol: ER_IB_MSG_262; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_262 was added in 8.0.11.
- Error number: MY-012088; Symbol: ER_IB_MSG_263; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_263 was added in 8.0.11.

- Error number: MY-012089; Symbol: ER_IB_MSG_264; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_264 was added in 8.0.11.

- Error number: MY-012090; Symbol: ER_IB_MSG_265; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_265 was added in 8.0.11.

- Error number: MY-012091; Symbol: ER_IB_MSG_266; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_266 was added in 8.0.11.

- Error number: MY-012092; Symbol: ER_IB_MSG_267; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_267 was added in 8.0.11.

- Error number: MY-012093; Symbol: ER_IB_MSG_268; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_268 was added in 8.0.11.

- Error number: MY-012094; Symbol: ER_IB_MSG_269; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_269 was added in 8.0.11.

- Error number: MY-012095; Symbol: ER_IB_MSG_270; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_270 was added in 8.0.11.

- Error number: MY-012096; Symbol: ER_IB_MSG_271; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_271 was added in 8.0.11.

- Error number: MY-012097; Symbol: ER_IB_MSG_272; SQLSTATE: HY000

  Message: Table flags are 0x%lx in the data dictionary but the flags in file %s are 0x%llx!

  ER_IB_MSG_272 was added in 8.0.11.

- Error number: MY-012098; Symbol: ER_IB_MSG_273; SQLSTATE: HY000

  Message: Can't read encryption key from file %s!

  ER_IB_MSG_273 was added in 8.0.11.

- Error number: MY-012099; Symbol: ER_IB_MSG_274; SQLSTATE: HY000

  Message: Cannot close file %s, because n_pending_flushes %zu

  ER_IB_MSG_274 was added in 8.0.11.
- Error number: MY-012100; Symbol: ER_IB_MSG_275; SQLSTATE: HY000

  Message: Cannot close file %s, because modification count %lld != flush count %lld

  ER_IB_MSG_275 was added in 8.0.11.
- Error number: MY-012101; Symbol: ER_IB_MSG_276; SQLSTATE: HY000

  Message: Cannot close file %s, because it is in use

  ER_IB_MSG_276 was added in 8.0.11.
- Error number: MY-012102; Symbol: ER_IB_MSG_277; SQLSTATE: HY000

  Message: Open file list len in shard %zu is %llu

  ER_IB_MSG_277 was added in 8.0.11.
- Error number: MY-012103; Symbol: ER_IB_MSG_278; SQLSTATE: HY000

  Message: Tablespace %s, waiting for IO to stop for %lld seconds

  ER_IB_MSG_278 was added in 8.0.11.
- Error number: MY-012104; Symbol: ER_IB_MSG_279; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_279 was added in 8.0.11.
- Error number: MY-012105; Symbol: ER_IB_MSG_280; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_280 was added in 8.0.11.
- Error number: MY-012106; Symbol: ER_IB_MSG_281; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_281 was added in 8.0.11.
- Error number: MY-012107; Symbol: ER_IB_MSG_282; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_282 was added in 8.0.11.
- Error number: MY-012108; Symbol: ER_IB_MSG_283; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_283 was added in 8.0.11.

- Error number: MY-012109; Symbol: ER_IB_MSG_284; SQLSTATE: HY000

  Message: You must raise the value of innodb_open_files in my.cnf! Remember that InnoDB keeps all log files and all system tablespace files open for the whole time mysqld is running, and needs to open also some .ibd files if the file-per-table storage model is used. Current open files %zu, max allowed open files %zu.

  ER_IB_MSG_284 was added in 8.0.11.

- Error number: MY-012110; Symbol: ER_IB_MSG_285; SQLSTATE: HY000

  Message: Max tablespace id is too high, %lu

  ER_IB_MSG_285 was added in 8.0.11.

- Error number: MY-012111; Symbol: ER_IB_MSG_286; SQLSTATE: HY000

  Message: Trying to access missing tablespace %lu

  ER_IB_MSG_286 was added in 8.0.11.

- Error number: MY-012112; Symbol: ER_IB_MSG_287; SQLSTATE: HY000

  Message: Trying to close/delete tablespace '%s' but there are %lu pending operations on it.

  ER_IB_MSG_287 was added in 8.0.11.

- Error number: MY-012113; Symbol: ER_IB_MSG_288; SQLSTATE: HY000

  Message: Trying to delete/close tablespace '%s' but there are %lu flushes and %zu pending I/O's on it.

  ER_IB_MSG_288 was added in 8.0.11.

- Error number: MY-012114; Symbol: ER_IB_MSG_289; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_289 was added in 8.0.11.

- Error number: MY-012115; Symbol: ER_IB_MSG_290; SQLSTATE: HY000

  Message: Cannot delete tablespace %lu because it is not found in the tablespace memory cache.

  ER_IB_MSG_290 was added in 8.0.11, removed after 8.0.17.

- Error number: MY-012116; Symbol: ER_IB_MSG_291; SQLSTATE: HY000

  Message: While deleting tablespace %lu in DISCARD TABLESPACE. File rename/delete failed: %s

  ER_IB_MSG_291 was added in 8.0.11.

- Error number: MY-012117; Symbol: ER_IB_MSG_292; SQLSTATE: HY000

  Message: Cannot delete tablespace %lu in DISCARD TABLESPACE: %s

  ER_IB_MSG_292 was added in 8.0.11.

- Error number: MY-012118; Symbol: ER_IB_MSG_293; SQLSTATE: HY000

  Message: Cannot rename '%s' to '%s' for space ID %lu because the source file does not exist.

ER_IB_MSG_293 was added in 8.0.11.

- Error number: MY-012119; Symbol: ER_IB_MSG_294; SQLSTATE: HY000

  Message: Cannot rename '%s' to '%s' for space ID %lu because the target file exists. Remove the target file and try again.

  ER_IB_MSG_294 was added in 8.0.11.

- Error number: MY-012120; Symbol: ER_IB_MSG_295; SQLSTATE: HY000

  Message: Cannot rename file '%s' (space id %lu) retried %llu times. There are either pending IOs or flushes or the file is being extended.

  ER_IB_MSG_295 was added in 8.0.11.

- Error number: MY-012121; Symbol: ER_IB_MSG_296; SQLSTATE: HY000

  Message: Cannot find space id %lu in the tablespace memory cache, though the file '%s' in a rename operation should have that ID.

  ER_IB_MSG_296 was added in 8.0.11.

- Error number: MY-012122; Symbol: ER_IB_MSG_297; SQLSTATE: HY000

  Message: Rename waiting for IO to resume

  ER_IB_MSG_297 was added in 8.0.11.

- Error number: MY-012123; Symbol: ER_IB_MSG_298; SQLSTATE: HY000

  Message: Cannot find tablespace for '%s' in the tablespace memory cache

  ER_IB_MSG_298 was added in 8.0.11.

- Error number: MY-012124; Symbol: ER_IB_MSG_299; SQLSTATE: HY000

  Message: Cannot find tablespace for '%s' in the tablespace memory cache

  ER_IB_MSG_299 was added in 8.0.11.

- Error number: MY-012125; Symbol: ER_IB_MSG_300; SQLSTATE: HY000

  Message: Tablespace '%s' is already in the tablespace memory cache

  ER_IB_MSG_300 was added in 8.0.11.

- Error number: MY-012126; Symbol: ER_IB_MSG_301; SQLSTATE: HY000

  Message: Cannot create file '%s'

  ER_IB_MSG_301 was added in 8.0.11.

- Error number: MY-012127; Symbol: ER_IB_MSG_302; SQLSTATE: HY000

  Message: The file '%s' already exists though the corresponding table did not exist. Have you moved InnoDB .ibd files around without using the SQL commands DISCARD TABLESPACE and IMPORT TABLESPACE, or did mysqld crash in the middle of CREATE TABLE? You can resolve the problem by removing the file '%s' under the 'datadir' of MySQL.

ER_IB_MSG_302 was added in 8.0.11, removed after 8.0.16.

- Error number: MY-012127; Symbol: ER_IB_MSG_UNEXPECTED_FILE_EXISTS; SQLSTATE: HY000

  Message: The file '%s' already exists though the corresponding table did not exist. Have you moved InnoDB .ibd files around without using the SQL commands DISCARD TABLESPACE and IMPORT TABLESPACE, or did mysqld crash in the middle of CREATE TABLE? You can resolve the problem by removing the file '%s' under the 'datadir' of MySQL.

  ER_IB_MSG_UNEXPECTED_FILE_EXISTS was added in 8.0.17.

- Error number: MY-012128; Symbol: ER_IB_MSG_303; SQLSTATE: HY000

  Message: posix_fallocate(): Failed to preallocate data for file %s, desired size %llu Operating system error number %d. Check that the disk is not full or a disk quota exceeded. Make sure the file system supports this function. Some operating system error numbers are described at %s operating-system-error-codes.html

  ER_IB_MSG_303 was added in 8.0.11.

- Error number: MY-012129; Symbol: ER_IB_MSG_304; SQLSTATE: HY000

  Message: Could not write the first page to tablespace '%s'

  ER_IB_MSG_304 was added in 8.0.11.

- Error number: MY-012130; Symbol: ER_IB_MSG_305; SQLSTATE: HY000

  Message: File flush of tablespace '%s' failed

  ER_IB_MSG_305 was added in 8.0.11.

- Error number: MY-012131; Symbol: ER_IB_MSG_306; SQLSTATE: HY000

  Message: Could not find a valid tablespace file for `%s`. %s

  ER_IB_MSG_306 was added in 8.0.11.

- Error number: MY-012132; Symbol: ER_IB_MSG_307; SQLSTATE: HY000

  Message: Ignoring data file '%s' with space ID %lu. Another data file called '%s' exists with the same space ID

  ER_IB_MSG_307 was added in 8.0.11.

- Error number: MY-012133; Symbol: ER_IB_MSG_308; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_308 was added in 8.0.11.

- Error number: MY-012134; Symbol: ER_IB_MSG_309; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_309 was added in 8.0.11.

- Error number: MY-012135; Symbol: ER_IB_MSG_310; SQLSTATE: HY000

Message: %s

ER_IB_MSG_310 was added in 8.0.11.

- Error number: MY-012136; Symbol: ER_IB_MSG_311; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_311 was added in 8.0.11.

- Error number: MY-012137; Symbol: ER_IB_MSG_312; SQLSTATE: HY000

  Message: Can't set encryption information for tablespace %s!

  ER_IB_MSG_312 was added in 8.0.11.

- Error number: MY-012138; Symbol: ER_IB_MSG_313; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_313 was added in 8.0.11.

- Error number: MY-012139; Symbol: ER_IB_MSG_314; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_314 was added in 8.0.11.

- Error number: MY-012140; Symbol: ER_IB_MSG_315; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_315 was added in 8.0.11.

- Error number: MY-012141; Symbol: ER_IB_MSG_316; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_316 was added in 8.0.11.

- Error number: MY-012142; Symbol: ER_IB_MSG_317; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_317 was added in 8.0.11.

- Error number: MY-012143; Symbol: ER_IB_MSG_318; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_318 was added in 8.0.11.

- Error number: MY-012144; Symbol: ER_IB_MSG_319; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_319 was added in 8.0.11.

- Error number: MY-012145; Symbol: ER_IB_MSG_320; SQLSTATE: HY000

Message: %s

ER_IB_MSG_320 was added in 8.0.11.

- Error number: MY-012146; Symbol: ER_IB_MSG_321; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_321 was added in 8.0.11.

- Error number: MY-012147; Symbol: ER_IB_MSG_322; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_322 was added in 8.0.11.

- Error number: MY-012148; Symbol: ER_IB_MSG_323; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_323 was added in 8.0.11.

- Error number: MY-012149; Symbol: ER_IB_MSG_324; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_324 was added in 8.0.11.

- Error number: MY-012150; Symbol: ER_IB_MSG_325; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_325 was added in 8.0.11.

- Error number: MY-012151; Symbol: ER_IB_MSG_326; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_326 was added in 8.0.11.

- Error number: MY-012152; Symbol: ER_IB_MSG_327; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_327 was added in 8.0.11.

- Error number: MY-012153; Symbol: ER_IB_MSG_328; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_328 was added in 8.0.11.

- Error number: MY-012154; Symbol: ER_IB_MSG_329; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_329 was added in 8.0.11.

- Error number: MY-012155; Symbol: ER_IB_MSG_330; SQLSTATE: HY000

Message: %s

ER_IB_MSG_330 was added in 8.0.11.

- Error number: MY-012156; Symbol: ER_IB_MSG_331; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_331 was added in 8.0.11.

- Error number: MY-012157; Symbol: ER_IB_MSG_332; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_332 was added in 8.0.11.

- Error number: MY-012158; Symbol: ER_IB_MSG_333; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_333 was added in 8.0.11.

- Error number: MY-012159; Symbol: ER_IB_MSG_334; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_334 was added in 8.0.11.

- Error number: MY-012160; Symbol: ER_IB_MSG_335; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_335 was added in 8.0.11.

- Error number: MY-012161; Symbol: ER_IB_MSG_336; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_336 was added in 8.0.11.

- Error number: MY-012162; Symbol: ER_IB_MSG_337; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_337 was added in 8.0.11.

- Error number: MY-012163; Symbol: ER_IB_MSG_338; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_338 was added in 8.0.11.

- Error number: MY-012164; Symbol: ER_IB_MSG_339; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_339 was added in 8.0.11.

- Error number: MY-012165; Symbol: ER_IB_MSG_340; SQLSTATE: HY000

Message: %s

ER_IB_MSG_340 was added in 8.0.11.

- Error number: MY-012166; Symbol: ER_IB_MSG_341; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_341 was added in 8.0.11.

- Error number: MY-012167; Symbol: ER_IB_MSG_342; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_342 was added in 8.0.11.

- Error number: MY-012168; Symbol: ER_IB_MSG_343; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_343 was added in 8.0.11.

- Error number: MY-012169; Symbol: ER_IB_MSG_344; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_344 was added in 8.0.11.

- Error number: MY-012170; Symbol: ER_IB_MSG_345; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_345 was added in 8.0.11.

- Error number: MY-012171; Symbol: ER_IB_MSG_346; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_346 was added in 8.0.11.

- Error number: MY-012172; Symbol: ER_IB_MSG_347; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_347 was added in 8.0.11.

- Error number: MY-012173; Symbol: ER_IB_MSG_348; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_348 was added in 8.0.11.

- Error number: MY-012174; Symbol: ER_IB_MSG_349; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_349 was added in 8.0.11.

- Error number: MY-012175; Symbol: ER_IB_MSG_350; SQLSTATE: HY000

Message: %s

ER_IB_MSG_350 was added in 8.0.11.

- Error number: MY-012176; Symbol: ER_IB_MSG_351; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_351 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012177; Symbol: ER_IB_MSG_352; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_352 was added in 8.0.11.

- Error number: MY-012178; Symbol: ER_IB_MSG_353; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_353 was added in 8.0.11.

- Error number: MY-012179; Symbol: ER_IB_MSG_354; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_354 was added in 8.0.11.

- Error number: MY-012180; Symbol: ER_IB_MSG_355; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_355 was added in 8.0.11.

- Error number: MY-012181; Symbol: ER_IB_MSG_356; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_356 was added in 8.0.11.

- Error number: MY-012182; Symbol: ER_IB_MSG_357; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_357 was added in 8.0.11.

- Error number: MY-012183; Symbol: ER_IB_MSG_358; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_358 was added in 8.0.11.

- Error number: MY-012184; Symbol: ER_IB_MSG_359; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_359 was added in 8.0.11.

- Error number: MY-012185; Symbol: ER_IB_MSG_360; SQLSTATE: HY000

Message: %s

ER_IB_MSG_360 was added in 8.0.11.

- Error number: MY-012186; Symbol: ER_IB_MSG_361; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_361 was added in 8.0.11.

- Error number: MY-012187; Symbol: ER_IB_MSG_362; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_362 was added in 8.0.11.

- Error number: MY-012188; Symbol: ER_IB_MSG_363; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_363 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012189; Symbol: ER_IB_MSG_364; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_364 was added in 8.0.11.

- Error number: MY-012190; Symbol: ER_IB_MSG_365; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_365 was added in 8.0.11.

- Error number: MY-012191; Symbol: ER_IB_MSG_366; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_366 was added in 8.0.11, removed after 8.0.18.

- Error number: MY-012191; Symbol: ER_IB_MSG_IGNORE_SCAN_PATH; SQLSTATE: HY000

  Message: Scan path '%s' is ignored because %s

  ER_IB_MSG_IGNORE_SCAN_PATH was added in 8.0.19.

- Error number: MY-012192; Symbol: ER_IB_MSG_367; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_367 was added in 8.0.11.

- Error number: MY-012193; Symbol: ER_IB_MSG_368; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_368 was added in 8.0.11.

- Error number: MY-012194; Symbol: ER_IB_MSG_369; SQLSTATE: HY000

Message: %s

ER_IB_MSG_369 was added in 8.0.11.

- Error number: MY-012195; Symbol: ER_IB_MSG_370; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_370 was added in 8.0.11.

- Error number: MY-012196; Symbol: ER_IB_MSG_371; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_371 was added in 8.0.11.

- Error number: MY-012197; Symbol: ER_IB_MSG_372; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_372 was added in 8.0.11.

- Error number: MY-012198; Symbol: ER_IB_MSG_373; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_373 was added in 8.0.11.

- Error number: MY-012199; Symbol: ER_IB_MSG_374; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_374 was added in 8.0.11.

- Error number: MY-012200; Symbol: ER_IB_MSG_375; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_375 was added in 8.0.11.

- Error number: MY-012201; Symbol: ER_IB_MSG_376; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_376 was added in 8.0.11.

- Error number: MY-012202; Symbol: ER_IB_MSG_377; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_377 was added in 8.0.11.

- Error number: MY-012203; Symbol: ER_IB_MSG_378; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_378 was added in 8.0.11.

- Error number: MY-012204; Symbol: ER_IB_MSG_379; SQLSTATE: HY000

Message: %s

ER_IB_MSG_379 was added in 8.0.11.

- Error number: MY-012205; Symbol: ER_IB_MSG_380; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_380 was added in 8.0.11.

- Error number: MY-012206; Symbol: ER_IB_MSG_381; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_381 was added in 8.0.11.

- Error number: MY-012207; Symbol: ER_IB_MSG_382; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_382 was added in 8.0.11.

- Error number: MY-012208; Symbol: ER_IB_MSG_383; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_383 was added in 8.0.11.

- Error number: MY-012209; Symbol: ER_IB_MSG_384; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_384 was added in 8.0.11.

- Error number: MY-012210; Symbol: ER_IB_MSG_385; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_385 was added in 8.0.11.

- Error number: MY-012211; Symbol: ER_IB_MSG_386; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_386 was added in 8.0.11.

- Error number: MY-012212; Symbol: ER_IB_MSG_387; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_387 was added in 8.0.11.

- Error number: MY-012213; Symbol: ER_IB_MSG_388; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_388 was added in 8.0.11.

- Error number: MY-012214; Symbol: ER_IB_MSG_389; SQLSTATE: HY000

Message: %s

ER_IB_MSG_389 was added in 8.0.11.

- Error number: MY-012215; Symbol: ER_IB_MSG_390; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_390 was added in 8.0.11.

- Error number: MY-012216; Symbol: ER_IB_MSG_391; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_391 was added in 8.0.11.

- Error number: MY-012217; Symbol: ER_IB_MSG_392; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_392 was added in 8.0.11.

- Error number: MY-012218; Symbol: ER_IB_MSG_393; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_393 was added in 8.0.11.

- Error number: MY-012219; Symbol: ER_IB_MSG_394; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_394 was added in 8.0.11.

- Error number: MY-012220; Symbol: ER_IB_MSG_395; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_395 was added in 8.0.11.

- Error number: MY-012221; Symbol: ER_IB_MSG_396; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_396 was added in 8.0.11.

- Error number: MY-012222; Symbol: ER_IB_MSG_397; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_397 was added in 8.0.11.

- Error number: MY-012223; Symbol: ER_IB_MSG_398; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_398 was added in 8.0.11.

- Error number: MY-012224; Symbol: ER_IB_MSG_399; SQLSTATE: HY000

Message: %s

ER_IB_MSG_399 was added in 8.0.11.

- Error number: MY-012225; Symbol: ER_IB_MSG_400; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_400 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012226; Symbol: ER_IB_MSG_401; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_401 was added in 8.0.11.

- Error number: MY-012227; Symbol: ER_IB_MSG_402; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_402 was added in 8.0.11.

- Error number: MY-012228; Symbol: ER_IB_MSG_403; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_403 was added in 8.0.11.

- Error number: MY-012229; Symbol: ER_IB_MSG_404; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_404 was added in 8.0.11.

- Error number: MY-012230; Symbol: ER_IB_MSG_405; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_405 was added in 8.0.11.

- Error number: MY-012231; Symbol: ER_IB_MSG_406; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_406 was added in 8.0.11.

- Error number: MY-012232; Symbol: ER_IB_MSG_407; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_407 was added in 8.0.11.

- Error number: MY-012233; Symbol: ER_IB_MSG_408; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_408 was added in 8.0.11.

- Error number: MY-012234; Symbol: ER_IB_MSG_409; SQLSTATE: HY000

Message: %s

ER_IB_MSG_409 was added in 8.0.11.

- Error number: MY-012235; Symbol: ER_IB_MSG_410; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_410 was added in 8.0.11.

- Error number: MY-012236; Symbol: ER_IB_MSG_411; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_411 was added in 8.0.11.

- Error number: MY-012237; Symbol: ER_IB_MSG_412; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_412 was added in 8.0.11.

- Error number: MY-012238; Symbol: ER_IB_MSG_413; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_413 was added in 8.0.11.

- Error number: MY-012239; Symbol: ER_IB_MSG_414; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_414 was added in 8.0.11.

- Error number: MY-012240; Symbol: ER_IB_MSG_415; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_415 was added in 8.0.11.

- Error number: MY-012241; Symbol: ER_IB_MSG_416; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_416 was added in 8.0.11.

- Error number: MY-012242; Symbol: ER_IB_MSG_417; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_417 was added in 8.0.11.

- Error number: MY-012243; Symbol: ER_IB_MSG_418; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_418 was added in 8.0.11.

- Error number: MY-012244; Symbol: ER_IB_MSG_419; SQLSTATE: HY000

Message: %s

ER_IB_MSG_419 was added in 8.0.11.

- Error number: MY-012245; Symbol: ER_IB_MSG_420; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_420 was added in 8.0.11.

- Error number: MY-012246; Symbol: ER_IB_MSG_421; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_421 was added in 8.0.11.

- Error number: MY-012247; Symbol: ER_IB_MSG_422; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_422 was added in 8.0.11.

- Error number: MY-012248; Symbol: ER_IB_MSG_423; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_423 was added in 8.0.11.

- Error number: MY-012249; Symbol: ER_IB_MSG_424; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_424 was added in 8.0.11.

- Error number: MY-012250; Symbol: ER_IB_MSG_425; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_425 was added in 8.0.11.

- Error number: MY-012251; Symbol: ER_IB_MSG_426; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_426 was added in 8.0.11.

- Error number: MY-012252; Symbol: ER_IB_MSG_427; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_427 was added in 8.0.11.

- Error number: MY-012253; Symbol: ER_IB_MSG_428; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_428 was added in 8.0.11.

- Error number: MY-012254; Symbol: ER_IB_MSG_429; SQLSTATE: HY000

Message: %s

ER_IB_MSG_429 was added in 8.0.11.

- Error number: MY-012255; Symbol: ER_IB_MSG_430; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_430 was added in 8.0.11.

- Error number: MY-012256; Symbol: ER_IB_MSG_431; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_431 was added in 8.0.11.

- Error number: MY-012257; Symbol: ER_IB_MSG_432; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_432 was added in 8.0.11.

- Error number: MY-012258; Symbol: ER_IB_MSG_433; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_433 was added in 8.0.11.

- Error number: MY-012259; Symbol: ER_IB_MSG_434; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_434 was added in 8.0.11.

- Error number: MY-012260; Symbol: ER_IB_MSG_435; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_435 was added in 8.0.11.

- Error number: MY-012261; Symbol: ER_IB_MSG_436; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_436 was added in 8.0.11.

- Error number: MY-012262; Symbol: ER_IB_MSG_437; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_437 was added in 8.0.11.

- Error number: MY-012263; Symbol: ER_IB_MSG_438; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_438 was added in 8.0.11.

- Error number: MY-012264; Symbol: ER_IB_MSG_439; SQLSTATE: HY000

Message: %s

ER_IB_MSG_439 was added in 8.0.11.

- Error number: MY-012265; Symbol: ER_IB_MSG_440; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_440 was added in 8.0.11.

- Error number: MY-012266; Symbol: ER_IB_MSG_441; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_441 was added in 8.0.11.

- Error number: MY-012267; Symbol: ER_IB_MSG_442; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_442 was added in 8.0.11.

- Error number: MY-012268; Symbol: ER_IB_MSG_443; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_443 was added in 8.0.11.

- Error number: MY-012269; Symbol: ER_IB_MSG_444; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_444 was added in 8.0.11.

- Error number: MY-012270; Symbol: ER_IB_MSG_445; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_445 was added in 8.0.11.

- Error number: MY-012271; Symbol: ER_IB_MSG_446; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_446 was added in 8.0.11.

- Error number: MY-012272; Symbol: ER_IB_MSG_447; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_447 was added in 8.0.11.

- Error number: MY-012273; Symbol: ER_IB_MSG_448; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_448 was added in 8.0.11.

- Error number: MY-012274; Symbol: ER_IB_MSG_449; SQLSTATE: HY000

Message: %s

ER_IB_MSG_449 was added in 8.0.11.

- Error number: MY-012275; Symbol: ER_IB_MSG_450; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_450 was added in 8.0.11.

- Error number: MY-012276; Symbol: ER_IB_MSG_451; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_451 was added in 8.0.11.

- Error number: MY-012277; Symbol: ER_IB_MSG_452; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_452 was added in 8.0.11.

- Error number: MY-012278; Symbol: ER_IB_MSG_453; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_453 was added in 8.0.11.

- Error number: MY-012279; Symbol: ER_IB_MSG_454; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_454 was added in 8.0.11.

- Error number: MY-012280; Symbol: ER_IB_MSG_455; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_455 was added in 8.0.11.

- Error number: MY-012281; Symbol: ER_IB_MSG_456; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_456 was added in 8.0.11.

- Error number: MY-012282; Symbol: ER_IB_MSG_457; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_457 was added in 8.0.11.

- Error number: MY-012283; Symbol: ER_IB_MSG_458; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_458 was added in 8.0.11.

- Error number: MY-012284; Symbol: ER_IB_MSG_459; SQLSTATE: HY000

Message: %s

ER_IB_MSG_459 was added in 8.0.11.

- Error number: MY-012285; Symbol: ER_IB_MSG_460; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_460 was added in 8.0.11.

- Error number: MY-012286; Symbol: ER_IB_MSG_461; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_461 was added in 8.0.11.

- Error number: MY-012287; Symbol: ER_IB_MSG_462; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_462 was added in 8.0.11.

- Error number: MY-012288; Symbol: ER_IB_MSG_463; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_463 was added in 8.0.11.

- Error number: MY-012289; Symbol: ER_IB_MSG_464; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_464 was added in 8.0.11.

- Error number: MY-012290; Symbol: ER_IB_MSG_465; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_465 was added in 8.0.11.

- Error number: MY-012291; Symbol: ER_IB_MSG_466; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_466 was added in 8.0.11.

- Error number: MY-012292; Symbol: ER_IB_MSG_467; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_467 was added in 8.0.11.

- Error number: MY-012293; Symbol: ER_IB_MSG_468; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_468 was added in 8.0.11.

- Error number: MY-012294; Symbol: ER_IB_MSG_469; SQLSTATE: HY000

Message: %s

ER_IB_MSG_469 was added in 8.0.11.

- Error number: MY-012295; Symbol: ER_IB_MSG_470; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_470 was added in 8.0.11.

- Error number: MY-012296; Symbol: ER_IB_MSG_471; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_471 was added in 8.0.11.

- Error number: MY-012297; Symbol: ER_IB_MSG_472; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_472 was added in 8.0.11.

- Error number: MY-012298; Symbol: ER_IB_MSG_473; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_473 was added in 8.0.11.

- Error number: MY-012299; Symbol: ER_IB_MSG_474; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_474 was added in 8.0.11.

- Error number: MY-012300; Symbol: ER_IB_MSG_475; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_475 was added in 8.0.11.

- Error number: MY-012301; Symbol: ER_IB_MSG_476; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_476 was added in 8.0.11.

- Error number: MY-012302; Symbol: ER_IB_MSG_477; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_477 was added in 8.0.11.

- Error number: MY-012303; Symbol: ER_IB_MSG_478; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_478 was added in 8.0.11.

- Error number: MY-012304; Symbol: ER_IB_MSG_479; SQLSTATE: HY000

Message: %s

ER_IB_MSG_479 was added in 8.0.11.

- Error number: MY-012305; Symbol: ER_IB_MSG_480; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_480 was added in 8.0.11.

- Error number: MY-012306; Symbol: ER_IB_MSG_481; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_481 was added in 8.0.11.

- Error number: MY-012307; Symbol: ER_IB_MSG_482; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_482 was added in 8.0.11.

- Error number: MY-012308; Symbol: ER_IB_MSG_483; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_483 was added in 8.0.11.

- Error number: MY-012309; Symbol: ER_IB_MSG_484; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_484 was added in 8.0.11.

- Error number: MY-012310; Symbol: ER_IB_MSG_485; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_485 was added in 8.0.11.

- Error number: MY-012311; Symbol: ER_IB_MSG_486; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_486 was added in 8.0.11.

- Error number: MY-012312; Symbol: ER_IB_MSG_487; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_487 was added in 8.0.11.

- Error number: MY-012313; Symbol: ER_IB_MSG_488; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_488 was added in 8.0.11.

- Error number: MY-012314; Symbol: ER_IB_MSG_489; SQLSTATE: HY000

Message: %s

ER_IB_MSG_489 was added in 8.0.11.

- Error number: MY-012315; Symbol: ER_IB_MSG_490; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_490 was added in 8.0.11.

- Error number: MY-012316; Symbol: ER_IB_MSG_491; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_491 was added in 8.0.11.

- Error number: MY-012317; Symbol: ER_IB_MSG_492; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_492 was added in 8.0.11.

- Error number: MY-012318; Symbol: ER_IB_MSG_493; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_493 was added in 8.0.11.

- Error number: MY-012319; Symbol: ER_IB_MSG_494; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_494 was added in 8.0.11.

- Error number: MY-012320; Symbol: ER_IB_MSG_495; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_495 was added in 8.0.11.

- Error number: MY-012321; Symbol: ER_IB_MSG_496; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_496 was added in 8.0.11.

- Error number: MY-012322; Symbol: ER_IB_MSG_497; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_497 was added in 8.0.11.

- Error number: MY-012323; Symbol: ER_IB_MSG_498; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_498 was added in 8.0.11.

- Error number: MY-012324; Symbol: ER_IB_MSG_499; SQLSTATE: HY000

Message: %s

ER_IB_MSG_499 was added in 8.0.11.

- Error number: MY-012325; Symbol: ER_IB_MSG_500; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_500 was added in 8.0.11.

- Error number: MY-012326; Symbol: ER_IB_MSG_501; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_501 was added in 8.0.11.

- Error number: MY-012327; Symbol: ER_IB_MSG_502; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_502 was added in 8.0.11.

- Error number: MY-012328; Symbol: ER_IB_MSG_503; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_503 was added in 8.0.11.

- Error number: MY-012329; Symbol: ER_IB_MSG_504; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_504 was added in 8.0.11.

- Error number: MY-012330; Symbol: ER_IB_MSG_505; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_505 was added in 8.0.11.

- Error number: MY-012331; Symbol: ER_IB_MSG_506; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_506 was added in 8.0.11.

- Error number: MY-012332; Symbol: ER_IB_MSG_507; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_507 was added in 8.0.11.

- Error number: MY-012333; Symbol: ER_IB_MSG_508; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_508 was added in 8.0.11.

- Error number: MY-012334; Symbol: ER_IB_MSG_509; SQLSTATE: HY000

Message: %s

ER_IB_MSG_509 was added in 8.0.11.

- Error number: MY-012335; Symbol: ER_IB_MSG_510; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_510 was added in 8.0.11.

- Error number: MY-012336; Symbol: ER_IB_MSG_511; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_511 was added in 8.0.11.

- Error number: MY-012337; Symbol: ER_IB_MSG_512; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_512 was added in 8.0.11.

- Error number: MY-012338; Symbol: ER_IB_MSG_513; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_513 was added in 8.0.11.

- Error number: MY-012339; Symbol: ER_IB_MSG_514; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_514 was added in 8.0.11.

- Error number: MY-012340; Symbol: ER_IB_MSG_515; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_515 was added in 8.0.11.

- Error number: MY-012341; Symbol: ER_IB_MSG_516; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_516 was added in 8.0.11.

- Error number: MY-012342; Symbol: ER_IB_MSG_517; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_517 was added in 8.0.11.

- Error number: MY-012343; Symbol: ER_IB_MSG_518; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_518 was added in 8.0.11.

- Error number: MY-012344; Symbol: ER_IB_MSG_519; SQLSTATE: HY000

Message: %s

ER_IB_MSG_519 was added in 8.0.11.

- Error number: MY-012345; Symbol: ER_IB_MSG_520; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_520 was added in 8.0.11.

- Error number: MY-012346; Symbol: ER_IB_MSG_521; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_521 was added in 8.0.11.

- Error number: MY-012347; Symbol: ER_IB_MSG_522; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_522 was added in 8.0.11.

- Error number: MY-012348; Symbol: ER_IB_MSG_523; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_523 was added in 8.0.11.

- Error number: MY-012349; Symbol: ER_IB_MSG_524; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_524 was added in 8.0.11.

- Error number: MY-012350; Symbol: ER_IB_MSG_525; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_525 was added in 8.0.11.

- Error number: MY-012351; Symbol: ER_IB_MSG_526; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_526 was added in 8.0.11.

- Error number: MY-012352; Symbol: ER_IB_MSG_527; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_527 was added in 8.0.11.

- Error number: MY-012353; Symbol: ER_IB_MSG_528; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_528 was added in 8.0.11.

- Error number: MY-012354; Symbol: ER_IB_MSG_529; SQLSTATE: HY000

Message: %s

ER_IB_MSG_529 was added in 8.0.11.

- Error number: MY-012355; Symbol: ER_IB_MSG_530; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_530 was added in 8.0.11.

- Error number: MY-012356; Symbol: ER_IB_MSG_531; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_531 was added in 8.0.11.

- Error number: MY-012357; Symbol: ER_IB_MSG_532; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_532 was added in 8.0.11.

- Error number: MY-012358; Symbol: ER_IB_MSG_533; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_533 was added in 8.0.11.

- Error number: MY-012359; Symbol: ER_IB_MSG_534; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_534 was added in 8.0.11.

- Error number: MY-012360; Symbol: ER_IB_MSG_535; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_535 was added in 8.0.11.

- Error number: MY-012361; Symbol: ER_IB_MSG_536; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_536 was added in 8.0.11.

- Error number: MY-012362; Symbol: ER_IB_MSG_537; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_537 was added in 8.0.11.

- Error number: MY-012363; Symbol: ER_IB_MSG_538; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_538 was added in 8.0.11.

- Error number: MY-012364; Symbol: ER_IB_MSG_539; SQLSTATE: HY000

Message: %s

ER_IB_MSG_539 was added in 8.0.11.

- Error number: MY-012365; Symbol: ER_IB_MSG_540; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_540 was added in 8.0.11.

- Error number: MY-012366; Symbol: ER_IB_MSG_541; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_541 was added in 8.0.11.

- Error number: MY-012367; Symbol: ER_IB_MSG_542; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_542 was added in 8.0.11.

- Error number: MY-012368; Symbol: ER_IB_MSG_543; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_543 was added in 8.0.11.

- Error number: MY-012369; Symbol: ER_IB_MSG_544; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_544 was added in 8.0.11.

- Error number: MY-012370; Symbol: ER_IB_MSG_545; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_545 was added in 8.0.11.

- Error number: MY-012371; Symbol: ER_IB_MSG_546; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_546 was added in 8.0.11.

- Error number: MY-012372; Symbol: ER_IB_MSG_547; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_547 was added in 8.0.11.

- Error number: MY-012373; Symbol: ER_IB_MSG_548; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_548 was added in 8.0.11.

- Error number: MY-012374; Symbol: ER_IB_MSG_549; SQLSTATE: HY000

Message: %s

ER_IB_MSG_549 was added in 8.0.11.

- Error number: MY-012375; Symbol: ER_IB_MSG_550; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_550 was added in 8.0.11.

- Error number: MY-012376; Symbol: ER_IB_MSG_551; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_551 was added in 8.0.11.

- Error number: MY-012377; Symbol: ER_IB_MSG_552; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_552 was added in 8.0.11.

- Error number: MY-012378; Symbol: ER_IB_MSG_553; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_553 was added in 8.0.11.

- Error number: MY-012379; Symbol: ER_IB_MSG_554; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_554 was added in 8.0.11.

- Error number: MY-012380; Symbol: ER_IB_MSG_555; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_555 was added in 8.0.11.

- Error number: MY-012381; Symbol: ER_IB_MSG_556; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_556 was added in 8.0.11.

- Error number: MY-012382; Symbol: ER_IB_MSG_557; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_557 was added in 8.0.11.

- Error number: MY-012383; Symbol: ER_IB_MSG_558; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_558 was added in 8.0.11.

- Error number: MY-012384; Symbol: ER_IB_MSG_559; SQLSTATE: HY000

Message: %s

ER_IB_MSG_559 was added in 8.0.11.

- Error number: MY-012385; Symbol: ER_IB_MSG_560; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_560 was added in 8.0.11.

- Error number: MY-012386; Symbol: ER_IB_MSG_561; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_561 was added in 8.0.11.

- Error number: MY-012387; Symbol: ER_IB_MSG_562; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_562 was added in 8.0.11.

- Error number: MY-012388; Symbol: ER_IB_MSG_563; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_563 was added in 8.0.11.

- Error number: MY-012389; Symbol: ER_IB_MSG_564; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_564 was added in 8.0.11.

- Error number: MY-012390; Symbol: ER_IB_MSG_565; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_565 was added in 8.0.11.

- Error number: MY-012391; Symbol: ER_IB_MSG_566; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_566 was added in 8.0.11.

- Error number: MY-012392; Symbol: ER_IB_MSG_567; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_567 was added in 8.0.11.

- Error number: MY-012393; Symbol: ER_IB_MSG_568; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_568 was added in 8.0.11.

- Error number: MY-012394; Symbol: ER_IB_MSG_569; SQLSTATE: HY000

Message: %s

ER_IB_MSG_569 was added in 8.0.11.

- Error number: MY-012395; Symbol: ER_IB_MSG_570; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_570 was added in 8.0.11.

- Error number: MY-012396; Symbol: ER_IB_MSG_571; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_571 was added in 8.0.11.

- Error number: MY-012397; Symbol: ER_IB_MSG_572; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_572 was added in 8.0.11, removed after 8.0.18.

- Error number: MY-012398; Symbol: ER_IB_MSG_573; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_573 was added in 8.0.11.

- Error number: MY-012399; Symbol: ER_IB_MSG_574; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_574 was added in 8.0.11.

- Error number: MY-012400; Symbol: ER_IB_MSG_575; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_575 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012401; Symbol: ER_IB_MSG_576; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_576 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012402; Symbol: ER_IB_MSG_577; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_577 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012403; Symbol: ER_IB_MSG_578; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_578 was added in 8.0.11.

- Error number: MY-012404; Symbol: ER_IB_MSG_579; SQLSTATE: HY000

Message: %s

ER_IB_MSG_579 was added in 8.0.11.

- Error number: MY-012405; Symbol: ER_IB_MSG_580; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_580 was added in 8.0.11.

- Error number: MY-012406; Symbol: ER_IB_MSG_581; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_581 was added in 8.0.11.

- Error number: MY-012407; Symbol: ER_IB_MSG_582; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_582 was added in 8.0.11.

- Error number: MY-012408; Symbol: ER_IB_MSG_583; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_583 was added in 8.0.11.

- Error number: MY-012409; Symbol: ER_IB_MSG_584; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_584 was added in 8.0.11.

- Error number: MY-012410; Symbol: ER_IB_MSG_585; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_585 was added in 8.0.11.

- Error number: MY-012411; Symbol: ER_IB_MSG_586; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_586 was added in 8.0.11.

- Error number: MY-012412; Symbol: ER_IB_MSG_587; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_587 was added in 8.0.11.

- Error number: MY-012413; Symbol: ER_IB_MSG_588; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_588 was added in 8.0.11.

- Error number: MY-012414; Symbol: ER_IB_MSG_589; SQLSTATE: HY000

Message: %s

ER_IB_MSG_589 was added in 8.0.11.

- Error number: MY-012415; Symbol: ER_IB_MSG_590; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_590 was added in 8.0.11.

- Error number: MY-012416; Symbol: ER_IB_MSG_591; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_591 was added in 8.0.11.

- Error number: MY-012417; Symbol: ER_IB_MSG_592; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_592 was added in 8.0.11.

- Error number: MY-012418; Symbol: ER_IB_MSG_593; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_593 was added in 8.0.11.

- Error number: MY-012419; Symbol: ER_IB_MSG_594; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_594 was added in 8.0.11.

- Error number: MY-012420; Symbol: ER_IB_MSG_595; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_595 was added in 8.0.11.

- Error number: MY-012421; Symbol: ER_IB_MSG_596; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_596 was added in 8.0.11.

- Error number: MY-012422; Symbol: ER_IB_MSG_597; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_597 was added in 8.0.11.

- Error number: MY-012423; Symbol: ER_IB_MSG_598; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_598 was added in 8.0.11.

- Error number: MY-012424; Symbol: ER_IB_MSG_599; SQLSTATE: HY000

Message: %s

ER_IB_MSG_599 was added in 8.0.11.

- Error number: MY-012425; Symbol: ER_IB_MSG_600; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_600 was added in 8.0.11.

- Error number: MY-012426; Symbol: ER_IB_MSG_601; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_601 was added in 8.0.11.

- Error number: MY-012427; Symbol: ER_IB_MSG_602; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_602 was added in 8.0.11.

- Error number: MY-012428; Symbol: ER_IB_MSG_603; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_603 was added in 8.0.11.

- Error number: MY-012429; Symbol: ER_IB_MSG_604; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_604 was added in 8.0.11.

- Error number: MY-012430; Symbol: ER_IB_MSG_605; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_605 was added in 8.0.11.

- Error number: MY-012431; Symbol: ER_IB_MSG_606; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_606 was added in 8.0.11.

- Error number: MY-012432; Symbol: ER_IB_MSG_607; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_607 was added in 8.0.11.

- Error number: MY-012433; Symbol: ER_IB_MSG_608; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_608 was added in 8.0.11.

- Error number: MY-012434; Symbol: ER_IB_MSG_609; SQLSTATE: HY000

Message: %s

ER_IB_MSG_609 was added in 8.0.11.

- Error number: MY-012435; Symbol: ER_IB_MSG_610; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_610 was added in 8.0.11.

- Error number: MY-012436; Symbol: ER_IB_MSG_611; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_611 was added in 8.0.11.

- Error number: MY-012437; Symbol: ER_IB_MSG_612; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_612 was added in 8.0.11.

- Error number: MY-012438; Symbol: ER_IB_MSG_613; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_613 was added in 8.0.11.

- Error number: MY-012439; Symbol: ER_IB_MSG_614; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_614 was added in 8.0.11.

- Error number: MY-012440; Symbol: ER_IB_MSG_615; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_615 was added in 8.0.11.

- Error number: MY-012441; Symbol: ER_IB_MSG_616; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_616 was added in 8.0.11.

- Error number: MY-012442; Symbol: ER_IB_MSG_617; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_617 was added in 8.0.11.

- Error number: MY-012443; Symbol: ER_IB_MSG_618; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_618 was added in 8.0.11.

- Error number: MY-012444; Symbol: ER_IB_MSG_619; SQLSTATE: HY000

Message: %s

ER_IB_MSG_619 was added in 8.0.11.

- Error number: MY-012445; Symbol: ER_IB_MSG_620; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_620 was added in 8.0.11.

- Error number: MY-012446; Symbol: ER_IB_MSG_621; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_621 was added in 8.0.11.

- Error number: MY-012447; Symbol: ER_IB_MSG_622; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_622 was added in 8.0.11.

- Error number: MY-012448; Symbol: ER_IB_MSG_623; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_623 was added in 8.0.11.

- Error number: MY-012449; Symbol: ER_IB_MSG_624; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_624 was added in 8.0.11.

- Error number: MY-012450; Symbol: ER_IB_MSG_625; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_625 was added in 8.0.11.

- Error number: MY-012451; Symbol: ER_IB_MSG_626; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_626 was added in 8.0.11.

- Error number: MY-012452; Symbol: ER_IB_MSG_627; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_627 was added in 8.0.11.

- Error number: MY-012453; Symbol: ER_IB_MSG_628; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_628 was added in 8.0.11.

- Error number: MY-012454; Symbol: ER_IB_MSG_629; SQLSTATE: HY000

Message: %s

ER_IB_MSG_629 was added in 8.0.11.

- Error number: MY-012455; Symbol: ER_IB_MSG_630; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_630 was added in 8.0.11.

- Error number: MY-012456; Symbol: ER_IB_MSG_631; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_631 was added in 8.0.11.

- Error number: MY-012457; Symbol: ER_IB_MSG_632; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_632 was added in 8.0.11.

- Error number: MY-012458; Symbol: ER_IB_MSG_633; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_633 was added in 8.0.11.

- Error number: MY-012459; Symbol: ER_IB_MSG_634; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_634 was added in 8.0.11.

- Error number: MY-012460; Symbol: ER_IB_MSG_635; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_635 was added in 8.0.11.

- Error number: MY-012461; Symbol: ER_IB_MSG_636; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_636 was added in 8.0.11.

- Error number: MY-012462; Symbol: ER_IB_MSG_637; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_637 was added in 8.0.11.

- Error number: MY-012463; Symbol: ER_IB_MSG_638; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_638 was added in 8.0.11.

- Error number: MY-012464; Symbol: ER_IB_MSG_639; SQLSTATE: HY000

Message: %s

ER_IB_MSG_639 was added in 8.0.11.

- Error number: MY-012465; Symbol: ER_IB_MSG_640; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_640 was added in 8.0.11.

- Error number: MY-012466; Symbol: ER_IB_MSG_641; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_641 was added in 8.0.11.

- Error number: MY-012467; Symbol: ER_IB_MSG_642; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_642 was added in 8.0.11.

- Error number: MY-012468; Symbol: ER_IB_MSG_643; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_643 was added in 8.0.11.

- Error number: MY-012469; Symbol: ER_IB_MSG_644; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_644 was added in 8.0.11.

- Error number: MY-012470; Symbol: ER_IB_MSG_645; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_645 was added in 8.0.11.

- Error number: MY-012471; Symbol: ER_IB_MSG_646; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_646 was added in 8.0.11.

- Error number: MY-012472; Symbol: ER_IB_MSG_647; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_647 was added in 8.0.11.

- Error number: MY-012473; Symbol: ER_IB_MSG_648; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_648 was added in 8.0.11.

- Error number: MY-012474; Symbol: ER_IB_MSG_649; SQLSTATE: HY000

Message: %s

ER_IB_MSG_649 was added in 8.0.11.

- Error number: MY-012475; Symbol: ER_IB_MSG_650; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_650 was added in 8.0.11.

- Error number: MY-012476; Symbol: ER_IB_MSG_651; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_651 was added in 8.0.11.

- Error number: MY-012477; Symbol: ER_IB_MSG_652; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_652 was added in 8.0.11.

- Error number: MY-012478; Symbol: ER_IB_MSG_653; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_653 was added in 8.0.11, removed after 8.0.15.

- Error number: MY-012478; Symbol: ER_IB_MSG_DDL_LOG_DELETE_BY_ID_OK; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_DDL_LOG_DELETE_BY_ID_OK was added in 8.0.16.

- Error number: MY-012479; Symbol: ER_IB_MSG_654; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_654 was added in 8.0.11.

- Error number: MY-012480; Symbol: ER_IB_MSG_655; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_655 was added in 8.0.11.

- Error number: MY-012481; Symbol: ER_IB_MSG_656; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_656 was added in 8.0.11.

- Error number: MY-012482; Symbol: ER_IB_MSG_657; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_657 was added in 8.0.11.

- Error number: MY-012483; Symbol: ER_IB_MSG_658; SQLSTATE: HY000

Message: %s

ER_IB_MSG_658 was added in 8.0.11.

- Error number: MY-012484; Symbol: ER_IB_MSG_659; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_659 was added in 8.0.11.

- Error number: MY-012485; Symbol: ER_IB_MSG_660; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_660 was added in 8.0.11.

- Error number: MY-012486; Symbol: ER_IB_MSG_661; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_661 was added in 8.0.11.

- Error number: MY-012487; Symbol: ER_IB_MSG_662; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_662 was added in 8.0.11.

- Error number: MY-012488; Symbol: ER_IB_MSG_663; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_663 was added in 8.0.11.

- Error number: MY-012489; Symbol: ER_IB_MSG_664; SQLSTATE: HY000

  Message: The transaction log size is too large for innodb_log_buffer_size (%lu >= %lu / 2). Trying to extend it.

  ER_IB_MSG_664 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012490; Symbol: ER_IB_MSG_665; SQLSTATE: HY000

  Message: innodb_log_buffer_size was extended to %lu bytes.

  ER_IB_MSG_665 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012491; Symbol: ER_IB_MSG_666; SQLSTATE: HY000

  Message: The transaction log files are too small for the single transaction log (size=%lu). So, the last checkpoint age might exceed the log group capacity %llu.

  ER_IB_MSG_666 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012492; Symbol: ER_IB_MSG_667; SQLSTATE: HY000

  Message: The age of the last checkpoint is %llu, which exceeds the log group capacity %llu.

  ER_IB_MSG_667 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012493; Symbol: ER_IB_MSG_668; SQLSTATE: HY000

  Message: Cannot continue operation. ib_logfiles are too small for innodb_thread_concurrency %lu. The combined size of ib_logfiles should be bigger than 200 kB * innodb_thread_concurrency. To get mysqld to start up, set innodb_thread_concurrency in my.cnf to a lower value, for example, to 8. After an ERROR-FREE shutdown of mysqld you can adjust the size of ib_logfiles. %s

  ER_IB_MSG_668 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012494; Symbol: ER_IB_MSG_669; SQLSTATE: HY000

  Message: Redo log was encrypted, but keyring plugin is not loaded.

  ER_IB_MSG_669 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012495; Symbol: ER_IB_MSG_670; SQLSTATE: HY000

  Message: Read redo log encryption metadata successful.

  ER_IB_MSG_670 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012496; Symbol: ER_IB_MSG_671; SQLSTATE: HY000

  Message: Can't set redo log tablespace encryption metadata.

  ER_IB_MSG_671 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012497; Symbol: ER_IB_MSG_672; SQLSTATE: HY000

  Message: Cannot read the encryption information in log file header, please check if keyring plugin loaded and the key file exists.

  ER_IB_MSG_672 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012498; Symbol: ER_IB_MSG_673; SQLSTATE: HY000

  Message: Can't set redo log tablespace to be encrypted in read-only mode.

  ER_IB_MSG_673 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012499; Symbol: ER_IB_MSG_674; SQLSTATE: HY000

  Message: Can't set redo log tablespace to be encrypted.

  ER_IB_MSG_674 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012500; Symbol: ER_IB_MSG_675; SQLSTATE: HY000

  Message: Can't set redo log tablespace to be encrypted.

  ER_IB_MSG_675 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012501; Symbol: ER_IB_MSG_676; SQLSTATE: HY000

  Message: Redo log encryption is enabled.

  ER_IB_MSG_676 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012502; Symbol: ER_IB_MSG_677; SQLSTATE: HY000

Message: Flush Waiting for archiver to to catch up lag LSN: %llu

ER_IB_MSG_677 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012503; Symbol: ER_IB_MSG_678; SQLSTATE: HY000

  Message: Flush overwriting data to archive - wait too long (1 minute) lag LSN: %llu

  ER_IB_MSG_678 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012504; Symbol: ER_IB_MSG_679; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_679 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012505; Symbol: ER_IB_MSG_680; SQLSTATE: HY000

  Message: Starting shutdown...

  ER_IB_MSG_680 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012506; Symbol: ER_IB_MSG_681; SQLSTATE: HY000

  Message: Waiting for %s to exit

  ER_IB_MSG_681 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012507; Symbol: ER_IB_MSG_682; SQLSTATE: HY000

  Message: Waiting for %lu active transactions to finish

  ER_IB_MSG_682 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012508; Symbol: ER_IB_MSG_683; SQLSTATE: HY000

  Message: Waiting for master thread to be suspended

  ER_IB_MSG_683 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012509; Symbol: ER_IB_MSG_684; SQLSTATE: HY000

  Message: Waiting for page_cleaner to finish flushing of buffer pool

  ER_IB_MSG_684 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012510; Symbol: ER_IB_MSG_685; SQLSTATE: HY000

  Message: Pending checkpoint_writes: %lu. Pending log flush writes: %lu.

  ER_IB_MSG_685 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012511; Symbol: ER_IB_MSG_686; SQLSTATE: HY000

  Message: Waiting for %lu buffer page I/Os to complete

  ER_IB_MSG_686 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012512; Symbol: ER_IB_MSG_687; SQLSTATE: HY000

Message: MySQL has requested a very fast shutdown without flushing the InnoDB buffer pool to data files. At the next mysqld startup InnoDB will do a crash recovery!

`ER_IB_MSG_687` was added in 8.0.11, removed after 8.0.13.

- Error number: `MY-012513`; Symbol: `ER_IB_MSG_688`; SQLSTATE: `HY000`

  Message: Background thread %s woke up during shutdown

  `ER_IB_MSG_688` was added in 8.0.11, removed after 8.0.13.

- Error number: `MY-012514`; Symbol: `ER_IB_MSG_689`; SQLSTATE: `HY000`

  Message: Waiting for archiver to finish archiving page and log

  `ER_IB_MSG_689` was added in 8.0.11, removed after 8.0.13.

- Error number: `MY-012515`; Symbol: `ER_IB_MSG_690`; SQLSTATE: `HY000`

  Message: Background thread %s woke up during shutdown

  `ER_IB_MSG_690` was added in 8.0.11, removed after 8.0.13.

- Error number: `MY-012516`; Symbol: `ER_IB_MSG_691`; SQLSTATE: `HY000`

  Message: Waiting for dirty buffer pages to be flushed

  `ER_IB_MSG_691` was added in 8.0.11, removed after 8.0.13.

- Error number: `MY-012517`; Symbol: `ER_IB_MSG_692`; SQLSTATE: `HY000`

  Message: Log sequence number at shutdown %llu is lower than at startup %llu!

  `ER_IB_MSG_692` was added in 8.0.11, removed after 8.0.13.

- Error number: `MY-012518`; Symbol: `ER_IB_MSG_693`; SQLSTATE: `HY000`

  Message: Waiting for archiver to finish archiving page and log

  `ER_IB_MSG_693` was added in 8.0.11, removed after 8.0.13.

- Error number: `MY-012519`; Symbol: `ER_IB_MSG_694`; SQLSTATE: `HY000`

  Message: ################ CORRUPT LOG RECORD FOUND ################

  `ER_IB_MSG_694` was added in 8.0.11.

- Error number: `MY-012520`; Symbol: `ER_IB_MSG_695`; SQLSTATE: `HY000`

  Message: Log record type %d, page %lu:%lu. Log parsing proceeded successfully up to %llu. Previous log record type %d, is multi %llu Recv offset %zd, prev %llu

  `ER_IB_MSG_695` was added in 8.0.11.

- Error number: `MY-012521`; Symbol: `ER_IB_MSG_696`; SQLSTATE: `HY000`

  Message: Hex dump starting %llu bytes before and ending %llu bytes after the corrupted record:

  `ER_IB_MSG_696` was added in 8.0.11.

- Error number: MY-012522; Symbol: ER_IB_MSG_697; SQLSTATE: HY000

  Message: Set innodb_force_recovery to ignore this error.

  ER_IB_MSG_697 was added in 8.0.11.

- Error number: MY-012523; Symbol: ER_IB_MSG_698; SQLSTATE: HY000

  Message: The log file may have been corrupt and it is possible that the log scan did not proceed far enough in recovery! Please run CHECK TABLE on your InnoDB tables to check that they are ok! If mysqld crashes after this recovery; %s

  ER_IB_MSG_698 was added in 8.0.11.

- Error number: MY-012524; Symbol: ER_IB_MSG_699; SQLSTATE: HY000

  Message: %llu pages with log records were left unprocessed!

  ER_IB_MSG_699 was added in 8.0.11.

- Error number: MY-012525; Symbol: ER_IB_MSG_700; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_700 was added in 8.0.11.

- Error number: MY-012526; Symbol: ER_IB_MSG_701; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_701 was added in 8.0.11.

- Error number: MY-012527; Symbol: ER_IB_MSG_702; SQLSTATE: HY000

  Message: Invalid redo log header checksum.

  ER_IB_MSG_702 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012528; Symbol: ER_IB_MSG_703; SQLSTATE: HY000

  Message: Unsupported redo log format (%lu). The redo log was created before MySQL 5.7.9

  ER_IB_MSG_703 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-012529; Symbol: ER_IB_MSG_704; SQLSTATE: HY000

  Message: Redo log format is v%lu. The redo log was created before MySQL 8.0.3.

  ER_IB_MSG_704 was added in 8.0.11.

- Error number: MY-012530; Symbol: ER_IB_MSG_705; SQLSTATE: HY000

  Message: Unknown redo log format (%lu). Please follow the instructions at %s upgrading-downgrading.html.

  ER_IB_MSG_705 was added in 8.0.11.

- Error number: MY-012531; Symbol: ER_IB_MSG_706; SQLSTATE: HY000

Message: No valid checkpoint found (corrupted redo log). You can try --innodb-force-recovery=6 as a last resort.

`ER_IB_MSG_706` was added in 8.0.11.

- Error number: `MY-012532`; Symbol: `ER_IB_MSG_707`; SQLSTATE: `HY000`

  Message: Applying a batch of %llu redo log records ...

  `ER_IB_MSG_707` was added in 8.0.11.

- Error number: `MY-012533`; Symbol: `ER_IB_MSG_708`; SQLSTATE: `HY000`

  Message: %s

  `ER_IB_MSG_708` was added in 8.0.11.

- Error number: `MY-012534`; Symbol: `ER_IB_MSG_709`; SQLSTATE: `HY000`

  Message: %s

  `ER_IB_MSG_709` was added in 8.0.11.

- Error number: `MY-012535`; Symbol: `ER_IB_MSG_710`; SQLSTATE: `HY000`

  Message: Apply batch completed!

  `ER_IB_MSG_710` was added in 8.0.11.

- Error number: `MY-012536`; Symbol: `ER_IB_MSG_711`; SQLSTATE: `HY000`

  Message: %s

  `ER_IB_MSG_711` was added in 8.0.11.

- Error number: `MY-012537`; Symbol: `ER_IB_MSG_712`; SQLSTATE: `HY000`

  Message: %s

  `ER_IB_MSG_712` was added in 8.0.11.

- Error number: `MY-012538`; Symbol: `ER_IB_MSG_713`; SQLSTATE: `HY000`

  Message: %s

  `ER_IB_MSG_713` was added in 8.0.11.

- Error number: `MY-012539`; Symbol: `ER_IB_MSG_714`; SQLSTATE: `HY000`

  Message: %s

  `ER_IB_MSG_714` was added in 8.0.11.

- Error number: `MY-012540`; Symbol: `ER_IB_MSG_715`; SQLSTATE: `HY000`

  Message: %s

  `ER_IB_MSG_715` was added in 8.0.11.

- Error number: MY-012541; Symbol: ER_IB_MSG_716; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_716 was added in 8.0.11.

- Error number: MY-012542; Symbol: ER_IB_MSG_717; SQLSTATE: HY000

  Message: An optimized(without redo logging) DDL operation has been performed. All modified pages may not have been flushed to the disk yet. This offline backup may not be consistent

  ER_IB_MSG_717 was added in 8.0.11.

- Error number: MY-012543; Symbol: ER_IB_MSG_718; SQLSTATE: HY000

  Message: Extending tablespace : %lu space name: %s to new size: %lu pages during recovery.

  ER_IB_MSG_718 was added in 8.0.11.

- Error number: MY-012544; Symbol: ER_IB_MSG_719; SQLSTATE: HY000

  Message: Could not extend tablespace: %lu space name: %s to new size: %lu pages during recovery.

  ER_IB_MSG_719 was added in 8.0.11.

- Error number: MY-012545; Symbol: ER_IB_MSG_720; SQLSTATE: HY000

  Message: Log block %llu at lsn %llu has valid header, but checksum field contains %lu, should be %lu.

  ER_IB_MSG_720 was added in 8.0.11.

- Error number: MY-012546; Symbol: ER_IB_MSG_721; SQLSTATE: HY000

  Message: Recovery skipped, --innodb-read-only set!

  ER_IB_MSG_721 was added in 8.0.11.

- Error number: MY-012547; Symbol: ER_IB_MSG_722; SQLSTATE: HY000

  Message: Log scan progressed past the checkpoint LSN %llu.

  ER_IB_MSG_722 was added in 8.0.11.

- Error number: MY-012548; Symbol: ER_IB_MSG_723; SQLSTATE: HY000

  Message: Log parsing buffer overflow. Recovery may have failed! Please set log_buffer_size to a value higher than %lu.

  ER_IB_MSG_723 was added in 8.0.11.

- Error number: MY-012549; Symbol: ER_IB_MSG_724; SQLSTATE: HY000

  Message: Set innodb_force_recovery to ignore this error.

  ER_IB_MSG_724 was added in 8.0.11.

- Error number: MY-012550; Symbol: ER_IB_MSG_725; SQLSTATE: HY000

  Message: Doing recovery: scanned up to log sequence number %llu

ER_IB_MSG_725 was added in 8.0.11.

- Error number: MY-012551; Symbol: ER_IB_MSG_726; SQLSTATE: HY000

  Message: Database was not shutdown normally!

  ER_IB_MSG_726 was added in 8.0.11.

- Error number: MY-012552; Symbol: ER_IB_MSG_727; SQLSTATE: HY000

  Message: Starting crash recovery.

  ER_IB_MSG_727 was added in 8.0.11.

- Error number: MY-012553; Symbol: ER_IB_MSG_728; SQLSTATE: HY000

  Message: The user has set SRV_FORCE_NO_LOG_REDO on, skipping log redo

  ER_IB_MSG_728 was added in 8.0.11.

- Error number: MY-012554; Symbol: ER_IB_MSG_729; SQLSTATE: HY000

  Message: Cannot restore from mysqlbackup, InnoDB running in read-only mode!

  ER_IB_MSG_729 was added in 8.0.11.

- Error number: MY-012555; Symbol: ER_IB_MSG_730; SQLSTATE: HY000

  Message: The log file was created by mysqlbackup --apply-log at %s. The following crash recovery is part of a normal restore.

  ER_IB_MSG_730 was added in 8.0.11.

- Error number: MY-012556; Symbol: ER_IB_MSG_731; SQLSTATE: HY000

  Message: Opening cloned database

  ER_IB_MSG_731 was added in 8.0.11.

- Error number: MY-012557; Symbol: ER_IB_MSG_732; SQLSTATE: HY000

  Message: Redo log is from an earlier version, v%lu.

  ER_IB_MSG_732 was added in 8.0.11.

- Error number: MY-012558; Symbol: ER_IB_MSG_733; SQLSTATE: HY000

  Message: Redo log format v%lu not supported. Current supported format is v%lu.

  ER_IB_MSG_733 was added in 8.0.11.

- Error number: MY-012559; Symbol: ER_IB_MSG_734; SQLSTATE: HY000

  Message: Are you sure you are using the right ib_logfiles to start up the database? Log sequence number in the ib_logfiles is %llu, less than the log sequence number in the first system tablespace file header, %llu.

  ER_IB_MSG_734 was added in 8.0.11.

- Error number: MY-012560; Symbol: ER_IB_MSG_735; SQLSTATE: HY000

  Message: The log sequence number %llu in the system tablespace does not match the log sequence number %llu in the ib_logfiles!

  ER_IB_MSG_735 was added in 8.0.11.

- Error number: MY-012561; Symbol: ER_IB_MSG_736; SQLSTATE: HY000

  Message: Can't initiate database recovery, running in read-only-mode.

  ER_IB_MSG_736 was added in 8.0.11.

- Error number: MY-012562; Symbol: ER_IB_MSG_737; SQLSTATE: HY000

  Message: We scanned the log up to %llu. A checkpoint was at %llu and the maximum LSN on a database page was %llu. It is possible that the database is now corrupt!

  ER_IB_MSG_737 was added in 8.0.11.

- Error number: MY-012563; Symbol: ER_IB_MSG_738; SQLSTATE: HY000

  Message: Waiting for recv_writer to finish flushing of buffer pool

  ER_IB_MSG_738 was added in 8.0.11.

- Error number: MY-012564; Symbol: ER_IB_MSG_739; SQLSTATE: HY000

  Message: Recovery parsing buffer extended to %zu.

  ER_IB_MSG_739 was added in 8.0.11.

- Error number: MY-012565; Symbol: ER_IB_MSG_740; SQLSTATE: HY000

  Message: Out of memory while resizing recovery parsing buffer.

  ER_IB_MSG_740 was added in 8.0.11.

- Error number: MY-012566; Symbol: ER_IB_MSG_741; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_741 was added in 8.0.11.

- Error number: MY-012567; Symbol: ER_IB_MSG_742; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_742 was added in 8.0.11.

- Error number: MY-012568; Symbol: ER_IB_MSG_743; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_743 was added in 8.0.11.

- Error number: MY-012569; Symbol: ER_IB_MSG_744; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_744 was added in 8.0.11.

- Error number: MY-012570; Symbol: ER_IB_MSG_745; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_745 was added in 8.0.11.

- Error number: MY-012571; Symbol: ER_IB_MSG_746; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_746 was added in 8.0.11.

- Error number: MY-012572; Symbol: ER_IB_MSG_747; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_747 was added in 8.0.11.

- Error number: MY-012573; Symbol: ER_IB_MSG_748; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_748 was added in 8.0.11.

- Error number: MY-012574; Symbol: ER_IB_MSG_749; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_749 was added in 8.0.11.

- Error number: MY-012575; Symbol: ER_IB_MSG_750; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_750 was added in 8.0.11.

- Error number: MY-012576; Symbol: ER_IB_MSG_751; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_751 was added in 8.0.11.

- Error number: MY-012577; Symbol: ER_IB_MSG_752; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_752 was added in 8.0.11.

- Error number: MY-012578; Symbol: ER_IB_MSG_753; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_753 was added in 8.0.11.

- Error number: MY-012579; Symbol: ER_IB_MSG_754; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_754 was added in 8.0.11.

- Error number: MY-012580; Symbol: ER_IB_MSG_755; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_755 was added in 8.0.11.

- Error number: MY-012581; Symbol: ER_IB_MSG_756; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_756 was added in 8.0.11.

- Error number: MY-012582; Symbol: ER_IB_MSG_757; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_757 was added in 8.0.11.

- Error number: MY-012583; Symbol: ER_IB_MSG_758; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_758 was added in 8.0.11.

- Error number: MY-012584; Symbol: ER_IB_MSG_759; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_759 was added in 8.0.11.

- Error number: MY-012585; Symbol: ER_IB_MSG_760; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_760 was added in 8.0.11.

- Error number: MY-012586; Symbol: ER_IB_MSG_761; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_761 was added in 8.0.11.

- Error number: MY-012587; Symbol: ER_IB_MSG_762; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_762 was added in 8.0.11.

- Error number: MY-012588; Symbol: ER_IB_MSG_763; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_763 was added in 8.0.11.

- Error number: MY-012589; Symbol: ER_IB_MSG_764; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_764 was added in 8.0.11.

- Error number: MY-012590; Symbol: ER_IB_MSG_765; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_765 was added in 8.0.11.

- Error number: MY-012591; Symbol: ER_IB_MSG_766; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_766 was added in 8.0.11.

- Error number: MY-012592; Symbol: ER_IB_MSG_767; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_767 was added in 8.0.11.

- Error number: MY-012593; Symbol: ER_IB_MSG_768; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_768 was added in 8.0.11.

- Error number: MY-012594; Symbol: ER_IB_MSG_769; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_769 was added in 8.0.11.

- Error number: MY-012595; Symbol: ER_IB_MSG_770; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_770 was added in 8.0.11.

- Error number: MY-012596; Symbol: ER_IB_MSG_771; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_771 was added in 8.0.11.

- Error number: MY-012597; Symbol: ER_IB_MSG_772; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_772 was added in 8.0.11.

- Error number: MY-012598; Symbol: ER_IB_MSG_773; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_773 was added in 8.0.11.

- Error number: MY-012599; Symbol: ER_IB_MSG_774; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_774 was added in 8.0.11.

- Error number: MY-012600; Symbol: ER_IB_MSG_775; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_775 was added in 8.0.11.

- Error number: MY-012601; Symbol: ER_IB_MSG_776; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_776 was added in 8.0.11.

- Error number: MY-012602; Symbol: ER_IB_MSG_777; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_777 was added in 8.0.11.

- Error number: MY-012603; Symbol: ER_IB_MSG_778; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_778 was added in 8.0.11.

- Error number: MY-012604; Symbol: ER_IB_MSG_779; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_779 was added in 8.0.11.

- Error number: MY-012605; Symbol: ER_IB_MSG_780; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_780 was added in 8.0.11.

- Error number: MY-012606; Symbol: ER_IB_MSG_781; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_781 was added in 8.0.11.

- Error number: MY-012607; Symbol: ER_IB_MSG_782; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_782 was added in 8.0.11.

- Error number: MY-012608; Symbol: ER_IB_MSG_783; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_783 was added in 8.0.11.

- Error number: MY-012609; Symbol: ER_IB_MSG_784; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_784 was added in 8.0.11.

- Error number: MY-012610; Symbol: ER_IB_MSG_785; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_785 was added in 8.0.11.

- Error number: MY-012611; Symbol: ER_IB_MSG_786; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_786 was added in 8.0.11.

- Error number: MY-012612; Symbol: ER_IB_MSG_787; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_787 was added in 8.0.11.

- Error number: MY-012613; Symbol: ER_IB_MSG_788; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_788 was added in 8.0.11.

- Error number: MY-012614; Symbol: ER_IB_MSG_789; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_789 was added in 8.0.11.

- Error number: MY-012615; Symbol: ER_IB_MSG_790; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_790 was added in 8.0.11.

- Error number: MY-012616; Symbol: ER_IB_MSG_791; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_791 was added in 8.0.11.

- Error number: MY-012617; Symbol: ER_IB_MSG_792; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_792 was added in 8.0.11.

- Error number: MY-012618; Symbol: ER_IB_MSG_793; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_793 was added in 8.0.11.

- Error number: MY-012619; Symbol: ER_IB_MSG_794; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_794 was added in 8.0.11.

- Error number: MY-012620; Symbol: ER_IB_MSG_795; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_795 was added in 8.0.11.

- Error number: MY-012621; Symbol: ER_IB_MSG_796; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_796 was added in 8.0.11.

- Error number: MY-012622; Symbol: ER_IB_MSG_797; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_797 was added in 8.0.11.

- Error number: MY-012623; Symbol: ER_IB_MSG_798; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_798 was added in 8.0.11.

- Error number: MY-012624; Symbol: ER_IB_MSG_799; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_799 was added in 8.0.11.

- Error number: MY-012625; Symbol: ER_IB_MSG_800; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_800 was added in 8.0.11.

- Error number: MY-012626; Symbol: ER_IB_MSG_801; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_801 was added in 8.0.11.

- Error number: MY-012627; Symbol: ER_IB_MSG_802; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_802 was added in 8.0.11.

- Error number: MY-012628; Symbol: ER_IB_MSG_803; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_803 was added in 8.0.11.

- Error number: MY-012629; Symbol: ER_IB_MSG_804; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_804 was added in 8.0.11.

- Error number: MY-012630; Symbol: ER_IB_MSG_805; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_805 was added in 8.0.11.

- Error number: MY-012631; Symbol: ER_IB_MSG_806; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_806 was added in 8.0.11.

- Error number: MY-012632; Symbol: ER_IB_MSG_807; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_807 was added in 8.0.11.

- Error number: MY-012633; Symbol: ER_IB_MSG_808; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_808 was added in 8.0.11.

- Error number: MY-012634; Symbol: ER_IB_MSG_809; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_809 was added in 8.0.11.

- Error number: MY-012635; Symbol: ER_IB_MSG_810; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_810 was added in 8.0.11.

- Error number: MY-012636; Symbol: ER_IB_MSG_811; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_811 was added in 8.0.11.

- Error number: MY-012637; Symbol: ER_IB_MSG_812; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_812 was added in 8.0.11.

- Error number: MY-012638; Symbol: ER_IB_MSG_813; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_813 was added in 8.0.11.

- Error number: MY-012639; Symbol: ER_IB_MSG_814; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_814 was added in 8.0.11.

- Error number: MY-012640; Symbol: ER_IB_MSG_815; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_815 was added in 8.0.11.

- Error number: MY-012641; Symbol: ER_IB_MSG_816; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_816 was added in 8.0.11.

- Error number: MY-012642; Symbol: ER_IB_MSG_817; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_817 was added in 8.0.11.

- Error number: MY-012643; Symbol: ER_IB_MSG_818; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_818 was added in 8.0.11.

- Error number: MY-012644; Symbol: ER_IB_MSG_819; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_819 was added in 8.0.11.

- Error number: MY-012645; Symbol: ER_IB_MSG_820; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_820 was added in 8.0.11.

- Error number: MY-012646; Symbol: ER_IB_MSG_821; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_821 was added in 8.0.11.

- Error number: MY-012647; Symbol: ER_IB_MSG_822; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_822 was added in 8.0.11.

- Error number: MY-012648; Symbol: ER_IB_MSG_823; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_823 was added in 8.0.11.

- Error number: MY-012649; Symbol: ER_IB_MSG_824; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_824 was added in 8.0.11.

- Error number: MY-012650; Symbol: ER_IB_MSG_825; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_825 was added in 8.0.11.

- Error number: MY-012651; Symbol: ER_IB_MSG_826; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_826 was added in 8.0.11.

- Error number: MY-012652; Symbol: ER_IB_MSG_827; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_827 was added in 8.0.11.

- Error number: MY-012653; Symbol: ER_IB_MSG_828; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_828 was added in 8.0.11.

- Error number: MY-012654; Symbol: ER_IB_MSG_829; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_829 was added in 8.0.11.

- Error number: MY-012655; Symbol: ER_IB_MSG_830; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_830 was added in 8.0.11.

- Error number: MY-012656; Symbol: ER_IB_MSG_831; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_831 was added in 8.0.11.

- Error number: MY-012657; Symbol: ER_IB_MSG_832; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_832 was added in 8.0.11.

- Error number: MY-012658; Symbol: ER_IB_MSG_833; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_833 was added in 8.0.11.

- Error number: MY-012659; Symbol: ER_IB_MSG_834; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_834 was added in 8.0.11.

- Error number: MY-012660; Symbol: ER_IB_MSG_835; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_835 was added in 8.0.11.

- Error number: MY-012661; Symbol: ER_IB_MSG_836; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_836 was added in 8.0.11.

- Error number: MY-012662; Symbol: ER_IB_MSG_837; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_837 was added in 8.0.11.

- Error number: MY-012663; Symbol: ER_IB_MSG_838; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_838 was added in 8.0.11.

- Error number: MY-012664; Symbol: ER_IB_MSG_839; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_839 was added in 8.0.11.

- Error number: MY-012665; Symbol: ER_IB_MSG_840; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_840 was added in 8.0.11.

- Error number: MY-012666; Symbol: ER_IB_MSG_841; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_841 was added in 8.0.11.

- Error number: MY-012667; Symbol: ER_IB_MSG_842; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_842 was added in 8.0.11.

- Error number: MY-012668; Symbol: ER_IB_MSG_843; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_843 was added in 8.0.11.

- Error number: MY-012669; Symbol: ER_IB_MSG_844; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_844 was added in 8.0.11.

- Error number: MY-012670; Symbol: ER_IB_MSG_845; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_845 was added in 8.0.11.

- Error number: MY-012671; Symbol: ER_IB_MSG_846; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_846 was added in 8.0.11.

- Error number: MY-012672; Symbol: ER_IB_MSG_847; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_847 was added in 8.0.11.

- Error number: MY-012673; Symbol: ER_IB_MSG_848; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_848 was added in 8.0.11.

- Error number: MY-012674; Symbol: ER_IB_MSG_849; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_849 was added in 8.0.11.

- Error number: MY-012675; Symbol: ER_IB_MSG_850; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_850 was added in 8.0.11.

- Error number: MY-012676; Symbol: ER_IB_MSG_851; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_851 was added in 8.0.11.

- Error number: MY-012677; Symbol: ER_IB_MSG_852; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_852 was added in 8.0.11.

- Error number: MY-012678; Symbol: ER_IB_MSG_853; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_853 was added in 8.0.11.

- Error number: MY-012679; Symbol: ER_IB_MSG_854; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_854 was added in 8.0.11.

- Error number: MY-012680; Symbol: ER_IB_MSG_855; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_855 was added in 8.0.11.

- Error number: MY-012681; Symbol: ER_IB_MSG_856; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_856 was added in 8.0.11.

- Error number: MY-012682; Symbol: ER_IB_MSG_857; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_857 was added in 8.0.11.

- Error number: MY-012683; Symbol: ER_IB_MSG_858; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_858 was added in 8.0.11.

- Error number: MY-012684; Symbol: ER_IB_MSG_859; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_859 was added in 8.0.11.

- Error number: MY-012685; Symbol: ER_IB_MSG_860; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_860 was added in 8.0.11.

- Error number: MY-012686; Symbol: ER_IB_MSG_861; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_861 was added in 8.0.11.

- Error number: MY-012687; Symbol: ER_IB_MSG_862; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_862 was added in 8.0.11.

- Error number: MY-012688; Symbol: ER_IB_MSG_863; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_863 was added in 8.0.11.

- Error number: MY-012689; Symbol: ER_IB_MSG_864; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_864 was added in 8.0.11.

- Error number: MY-012690; Symbol: ER_IB_MSG_865; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_865 was added in 8.0.11.

- Error number: MY-012691; Symbol: ER_IB_MSG_866; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_866 was added in 8.0.11.

- Error number: MY-012692; Symbol: ER_IB_MSG_867; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_867 was added in 8.0.11.

- Error number: MY-012693; Symbol: ER_IB_MSG_868; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_868 was added in 8.0.11.

- Error number: MY-012694; Symbol: ER_IB_MSG_869; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_869 was added in 8.0.11.

- Error number: MY-012695; Symbol: ER_IB_MSG_870; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_870 was added in 8.0.11.

- Error number: MY-012696; Symbol: ER_IB_MSG_871; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_871 was added in 8.0.11.

- Error number: MY-012697; Symbol: ER_IB_MSG_872; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_872 was added in 8.0.11.

- Error number: MY-012698; Symbol: ER_IB_MSG_873; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_873 was added in 8.0.11.

- Error number: MY-012699; Symbol: ER_IB_MSG_874; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_874 was added in 8.0.11.

- Error number: MY-012700; Symbol: ER_IB_MSG_875; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_875 was added in 8.0.11.

- Error number: MY-012701; Symbol: ER_IB_MSG_876; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_876 was added in 8.0.11.

- Error number: MY-012702; Symbol: ER_IB_MSG_877; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_877 was added in 8.0.11.

- Error number: MY-012703; Symbol: ER_IB_MSG_878; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_878 was added in 8.0.11.

- Error number: MY-012704; Symbol: ER_IB_MSG_879; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_879 was added in 8.0.11.

- Error number: MY-012705; Symbol: ER_IB_MSG_880; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_880 was added in 8.0.11.

- Error number: MY-012706; Symbol: ER_IB_MSG_881; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_881 was added in 8.0.11.

- Error number: MY-012707; Symbol: ER_IB_MSG_882; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_882 was added in 8.0.11.

- Error number: MY-012708; Symbol: ER_IB_MSG_883; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_883 was added in 8.0.11.

- Error number: MY-012709; Symbol: ER_IB_MSG_884; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_884 was added in 8.0.11.

- Error number: MY-012710; Symbol: ER_IB_MSG_885; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_885 was added in 8.0.11.

- Error number: MY-012711; Symbol: ER_IB_MSG_886; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_886 was added in 8.0.11.

- Error number: MY-012712; Symbol: ER_IB_MSG_887; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_887 was added in 8.0.11.

- Error number: MY-012713; Symbol: ER_IB_MSG_888; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_888 was added in 8.0.11.

- Error number: MY-012714; Symbol: ER_IB_MSG_889; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_889 was added in 8.0.11.

- Error number: MY-012715; Symbol: ER_IB_MSG_890; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_890 was added in 8.0.11.

- Error number: MY-012716; Symbol: ER_IB_MSG_891; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_891 was added in 8.0.11.

- Error number: MY-012717; Symbol: ER_IB_MSG_892; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_892 was added in 8.0.11.

- Error number: MY-012718; Symbol: ER_IB_MSG_893; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_893 was added in 8.0.11.

- Error number: MY-012719; Symbol: ER_IB_MSG_894; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_894 was added in 8.0.11.

- Error number: MY-012720; Symbol: ER_IB_MSG_895; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_895 was added in 8.0.11.

- Error number: MY-012721; Symbol: ER_IB_MSG_896; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_896 was added in 8.0.11.

- Error number: MY-012722; Symbol: ER_IB_MSG_897; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_897 was added in 8.0.11.

- Error number: MY-012723; Symbol: ER_IB_MSG_898; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_898 was added in 8.0.11.

- Error number: MY-012724; Symbol: ER_IB_MSG_899; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_899 was added in 8.0.11.

- Error number: MY-012725; Symbol: ER_IB_MSG_900; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_900 was added in 8.0.11.

- Error number: MY-012726; Symbol: ER_IB_MSG_901; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_901 was added in 8.0.11.

- Error number: MY-012727; Symbol: ER_IB_MSG_902; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_902 was added in 8.0.11.

- Error number: MY-012728; Symbol: ER_IB_MSG_903; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_903 was added in 8.0.11.

- Error number: MY-012729; Symbol: ER_IB_MSG_904; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_904 was added in 8.0.11.

- Error number: MY-012730; Symbol: ER_IB_MSG_905; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_905 was added in 8.0.11.

- Error number: MY-012731; Symbol: ER_IB_MSG_906; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_906 was added in 8.0.11.

- Error number: MY-012732; Symbol: ER_IB_MSG_907; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_907 was added in 8.0.11.

- Error number: MY-012733; Symbol: ER_IB_MSG_908; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_908 was added in 8.0.11.

- Error number: MY-012734; Symbol: ER_IB_MSG_909; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_909 was added in 8.0.11.

- Error number: MY-012735; Symbol: ER_IB_MSG_910; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_910 was added in 8.0.11.

- Error number: MY-012736; Symbol: ER_IB_MSG_911; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_911 was added in 8.0.11.

- Error number: MY-012737; Symbol: ER_IB_MSG_912; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_912 was added in 8.0.11.

- Error number: MY-012738; Symbol: ER_IB_MSG_913; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_913 was added in 8.0.11.

- Error number: MY-012739; Symbol: ER_IB_MSG_914; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_914 was added in 8.0.11.

- Error number: MY-012740; Symbol: ER_IB_MSG_915; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_915 was added in 8.0.11.

- Error number: MY-012741; Symbol: ER_IB_MSG_916; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_916 was added in 8.0.11.

- Error number: MY-012742; Symbol: ER_IB_MSG_917; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_917 was added in 8.0.11.

- Error number: MY-012743; Symbol: ER_IB_MSG_918; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_918 was added in 8.0.11.

- Error number: MY-012744; Symbol: ER_IB_MSG_919; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_919 was added in 8.0.11.

- Error number: MY-012745; Symbol: ER_IB_MSG_920; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_920 was added in 8.0.11.

- Error number: MY-012746; Symbol: ER_IB_MSG_921; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_921 was added in 8.0.11.

- Error number: MY-012747; Symbol: ER_IB_MSG_922; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_922 was added in 8.0.11.

- Error number: MY-012748; Symbol: ER_IB_MSG_923; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_923 was added in 8.0.11.

- Error number: MY-012749; Symbol: ER_IB_MSG_924; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_924 was added in 8.0.11.

- Error number: MY-012750; Symbol: ER_IB_MSG_925; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_925 was added in 8.0.11.

- Error number: MY-012751; Symbol: ER_IB_MSG_926; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_926 was added in 8.0.11.

- Error number: MY-012752; Symbol: ER_IB_MSG_927; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_927 was added in 8.0.11.

- Error number: MY-012753; Symbol: ER_IB_MSG_928; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_928 was added in 8.0.11.

- Error number: MY-012754; Symbol: ER_IB_MSG_929; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_929 was added in 8.0.11.

- Error number: MY-012755; Symbol: ER_IB_MSG_930; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_930 was added in 8.0.11.

- Error number: MY-012756; Symbol: ER_IB_MSG_931; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_931 was added in 8.0.11.

- Error number: MY-012757; Symbol: ER_IB_MSG_932; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_932 was added in 8.0.11.

- Error number: MY-012758; Symbol: ER_IB_MSG_933; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_933 was added in 8.0.11.

- Error number: MY-012759; Symbol: ER_IB_MSG_934; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_934 was added in 8.0.11.

- Error number: MY-012760; Symbol: ER_IB_MSG_935; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_935 was added in 8.0.11.

- Error number: MY-012761; Symbol: ER_IB_MSG_936; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_936 was added in 8.0.11.

- Error number: MY-012762; Symbol: ER_IB_MSG_937; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_937 was added in 8.0.11.

- Error number: MY-012763; Symbol: ER_IB_MSG_938; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_938 was added in 8.0.11.

- Error number: MY-012764; Symbol: ER_IB_MSG_939; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_939 was added in 8.0.11.

- Error number: MY-012765; Symbol: ER_IB_MSG_940; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_940 was added in 8.0.11.

- Error number: MY-012766; Symbol: ER_IB_MSG_941; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_941 was added in 8.0.11.

- Error number: MY-012767; Symbol: ER_IB_MSG_942; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_942 was added in 8.0.11.

- Error number: MY-012768; Symbol: ER_IB_MSG_943; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_943 was added in 8.0.11.

- Error number: MY-012769; Symbol: ER_IB_MSG_944; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_944 was added in 8.0.11.

- Error number: MY-012770; Symbol: ER_IB_MSG_945; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_945 was added in 8.0.11.

- Error number: MY-012771; Symbol: ER_IB_MSG_946; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_946 was added in 8.0.11.

- Error number: MY-012772; Symbol: ER_IB_MSG_947; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_947 was added in 8.0.11.

- Error number: MY-012773; Symbol: ER_IB_MSG_948; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_948 was added in 8.0.11.

- Error number: MY-012774; Symbol: ER_IB_MSG_949; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_949 was added in 8.0.11.

- Error number: MY-012775; Symbol: ER_IB_MSG_950; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_950 was added in 8.0.11.

- Error number: MY-012776; Symbol: ER_IB_MSG_951; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_951 was added in 8.0.11.

- Error number: MY-012777; Symbol: ER_IB_MSG_952; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_952 was added in 8.0.11.

- Error number: MY-012778; Symbol: ER_IB_MSG_953; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_953 was added in 8.0.11.

- Error number: MY-012779; Symbol: ER_IB_MSG_954; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_954 was added in 8.0.11.

- Error number: MY-012780; Symbol: ER_IB_MSG_955; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_955 was added in 8.0.11.

- Error number: MY-012781; Symbol: ER_IB_MSG_956; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_956 was added in 8.0.11.

- Error number: MY-012782; Symbol: ER_IB_MSG_957; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_957 was added in 8.0.11.

- Error number: MY-012783; Symbol: ER_IB_MSG_958; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_958 was added in 8.0.11.

- Error number: MY-012784; Symbol: ER_IB_MSG_959; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_959 was added in 8.0.11.

- Error number: MY-012785; Symbol: ER_IB_MSG_960; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_960 was added in 8.0.11.

- Error number: MY-012786; Symbol: ER_IB_MSG_961; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_961 was added in 8.0.11.

- Error number: MY-012787; Symbol: ER_IB_MSG_962; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_962 was added in 8.0.11.

- Error number: MY-012788; Symbol: ER_IB_MSG_963; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_963 was added in 8.0.11.

- Error number: MY-012789; Symbol: ER_IB_MSG_964; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_964 was added in 8.0.11.

- Error number: MY-012790; Symbol: ER_IB_MSG_965; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_965 was added in 8.0.11.

- Error number: MY-012791; Symbol: ER_IB_MSG_966; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_966 was added in 8.0.11.

- Error number: MY-012792; Symbol: ER_IB_MSG_967; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_967 was added in 8.0.11.

- Error number: MY-012793; Symbol: ER_IB_MSG_968; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_968 was added in 8.0.11.

- Error number: MY-012794; Symbol: ER_IB_MSG_969; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_969 was added in 8.0.11.

- Error number: MY-012795; Symbol: ER_IB_MSG_970; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_970 was added in 8.0.11.

- Error number: MY-012796; Symbol: ER_IB_MSG_971; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_971 was added in 8.0.11.

- Error number: MY-012797; Symbol: ER_IB_MSG_972; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_972 was added in 8.0.11.

- Error number: MY-012798; Symbol: ER_IB_MSG_973; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_973 was added in 8.0.11.

- Error number: MY-012799; Symbol: ER_IB_MSG_974; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_974 was added in 8.0.11.

- Error number: MY-012800; Symbol: ER_IB_MSG_975; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_975 was added in 8.0.11.

- Error number: MY-012801; Symbol: ER_IB_MSG_976; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_976 was added in 8.0.11.

- Error number: MY-012802; Symbol: ER_IB_MSG_977; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_977 was added in 8.0.11.

- Error number: MY-012803; Symbol: ER_IB_MSG_978; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_978 was added in 8.0.11.

- Error number: MY-012804; Symbol: ER_IB_MSG_979; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_979 was added in 8.0.11.

- Error number: MY-012805; Symbol: ER_IB_MSG_980; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_980 was added in 8.0.11.

- Error number: MY-012806; Symbol: ER_IB_MSG_981; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_981 was added in 8.0.11.

- Error number: MY-012807; Symbol: ER_IB_MSG_982; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_982 was added in 8.0.11.

- Error number: MY-012808; Symbol: ER_IB_MSG_983; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_983 was added in 8.0.11.

- Error number: MY-012809; Symbol: ER_IB_MSG_984; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_984 was added in 8.0.11.

- Error number: MY-012810; Symbol: ER_IB_MSG_985; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_985 was added in 8.0.11.

- Error number: MY-012811; Symbol: ER_IB_MSG_986; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_986 was added in 8.0.11.

- Error number: MY-012812; Symbol: ER_IB_MSG_987; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_987 was added in 8.0.11.

- Error number: MY-012813; Symbol: ER_IB_MSG_988; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_988 was added in 8.0.11.

- Error number: MY-012814; Symbol: ER_IB_MSG_989; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_989 was added in 8.0.11.

- Error number: MY-012815; Symbol: ER_IB_MSG_990; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_990 was added in 8.0.11.

- Error number: MY-012816; Symbol: ER_IB_MSG_991; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_991 was added in 8.0.11.

- Error number: MY-012817; Symbol: ER_IB_MSG_992; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_992 was added in 8.0.11.

- Error number: MY-012818; Symbol: ER_IB_MSG_993; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_993 was added in 8.0.11.

- Error number: MY-012819; Symbol: ER_IB_MSG_994; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_994 was added in 8.0.11.

- Error number: MY-012820; Symbol: ER_IB_MSG_995; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_995 was added in 8.0.11.

- Error number: MY-012821; Symbol: ER_IB_MSG_996; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_996 was added in 8.0.11.

- Error number: MY-012822; Symbol: ER_IB_MSG_997; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_997 was added in 8.0.11.

- Error number: MY-012823; Symbol: ER_IB_MSG_998; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_998 was added in 8.0.11.

- Error number: MY-012824; Symbol: ER_IB_MSG_999; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_999 was added in 8.0.11.

- Error number: MY-012825; Symbol: ER_IB_MSG_1000; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1000 was added in 8.0.11.

- Error number: MY-012826; Symbol: ER_IB_MSG_1001; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1001 was added in 8.0.11.

- Error number: MY-012827; Symbol: ER_IB_MSG_1002; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1002 was added in 8.0.11.

- Error number: MY-012828; Symbol: ER_IB_MSG_1003; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1003 was added in 8.0.11.

- Error number: MY-012829; Symbol: ER_IB_MSG_1004; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1004 was added in 8.0.11.

- Error number: MY-012830; Symbol: ER_IB_MSG_1005; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1005 was added in 8.0.11.

- Error number: MY-012831; Symbol: ER_IB_MSG_1006; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1006 was added in 8.0.11.

- Error number: MY-012832; Symbol: ER_IB_MSG_1007; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1007 was added in 8.0.11.

- Error number: MY-012833; Symbol: ER_IB_MSG_1008; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1008 was added in 8.0.11.

- Error number: MY-012834; Symbol: ER_IB_MSG_1009; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1009 was added in 8.0.11.

- Error number: MY-012835; Symbol: ER_IB_MSG_1010; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1010 was added in 8.0.11.

- Error number: MY-012836; Symbol: ER_IB_MSG_1011; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1011 was added in 8.0.11.

- Error number: MY-012837; Symbol: ER_IB_MSG_1012; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1012 was added in 8.0.11.

- Error number: MY-012838; Symbol: ER_IB_MSG_1013; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1013 was added in 8.0.11.

- Error number: MY-012839; Symbol: ER_IB_MSG_1014; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1014 was added in 8.0.11.

- Error number: MY-012840; Symbol: ER_IB_MSG_1015; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1015 was added in 8.0.11.

- Error number: MY-012841; Symbol: ER_IB_MSG_1016; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1016 was added in 8.0.11.

- Error number: MY-012842; Symbol: ER_IB_MSG_1017; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1017 was added in 8.0.11.

- Error number: MY-012843; Symbol: ER_IB_MSG_1018; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1018 was added in 8.0.11.

- Error number: MY-012844; Symbol: ER_IB_MSG_1019; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1019 was added in 8.0.11.

- Error number: MY-012845; Symbol: ER_IB_MSG_1020; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1020 was added in 8.0.11.

- Error number: MY-012846; Symbol: ER_IB_MSG_1021; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1021 was added in 8.0.11.

- Error number: MY-012847; Symbol: ER_IB_MSG_1022; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1022 was added in 8.0.11.

- Error number: MY-012848; Symbol: ER_IB_MSG_1023; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1023 was added in 8.0.11.

- Error number: MY-012849; Symbol: ER_IB_MSG_1024; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1024 was added in 8.0.11.

- Error number: MY-012850; Symbol: ER_IB_MSG_1025; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1025 was added in 8.0.11.

- Error number: MY-012851; Symbol: ER_IB_MSG_1026; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1026 was added in 8.0.11.

- Error number: MY-012852; Symbol: ER_IB_MSG_1027; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1027 was added in 8.0.11.

- Error number: MY-012853; Symbol: ER_IB_MSG_1028; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1028 was added in 8.0.11.

- Error number: MY-012854; Symbol: ER_IB_MSG_1029; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1029 was added in 8.0.11.

- Error number: MY-012855; Symbol: ER_IB_MSG_1030; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1030 was added in 8.0.11.

- Error number: MY-012856; Symbol: ER_IB_MSG_1031; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1031 was added in 8.0.11.

- Error number: MY-012857; Symbol: ER_IB_MSG_1032; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1032 was added in 8.0.11.

- Error number: MY-012858; Symbol: ER_IB_MSG_1033; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1033 was added in 8.0.11.

- Error number: MY-012859; Symbol: ER_IB_MSG_1034; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1034 was added in 8.0.11.

- Error number: MY-012860; Symbol: ER_IB_MSG_1035; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1035 was added in 8.0.11.

- Error number: MY-012861; Symbol: ER_IB_MSG_1036; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1036 was added in 8.0.11.

- Error number: MY-012862; Symbol: ER_IB_MSG_1037; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1037 was added in 8.0.11.

- Error number: MY-012863; Symbol: ER_IB_MSG_1038; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1038 was added in 8.0.11.

- Error number: MY-012864; Symbol: ER_IB_MSG_1039; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1039 was added in 8.0.11.

- Error number: MY-012865; Symbol: ER_IB_MSG_1040; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1040 was added in 8.0.11.

- Error number: MY-012866; Symbol: ER_IB_MSG_1041; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1041 was added in 8.0.11.

- Error number: MY-012867; Symbol: ER_IB_MSG_1042; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1042 was added in 8.0.11.

- Error number: MY-012868; Symbol: ER_IB_MSG_1043; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1043 was added in 8.0.11.

- Error number: MY-012869; Symbol: ER_IB_MSG_1044; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1044 was added in 8.0.11.

- Error number: MY-012870; Symbol: ER_IB_MSG_1045; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1045 was added in 8.0.11.

- Error number: MY-012871; Symbol: ER_IB_MSG_1046; SQLSTATE: HY000

  Message: Old log sequence number %llu was greater than the new log sequence number %llu. Please submit a bug report to http://bugs.mysql.com

  ER_IB_MSG_1046 was added in 8.0.11.

- Error number: MY-012872; Symbol: ER_IB_MSG_1047; SQLSTATE: HY000

  Message: Semaphore wait has lasted > %llu seconds. We intentionally crash the server because it appears to be hung.

  ER_IB_MSG_1047 was added in 8.0.11.

- Error number: MY-012873; Symbol: ER_IB_MSG_1048; SQLSTATE: HY000

  Message: Waiting for %llu table(s) to be dropped

  ER_IB_MSG_1048 was added in 8.0.11.

- Error number: MY-012874; Symbol: ER_IB_MSG_1049; SQLSTATE: HY000

  Message: Waiting for change buffer merge to complete number of bytes of change buffer just merged: %llu

  ER_IB_MSG_1049 was added in 8.0.11.

- Error number: MY-012875; Symbol: ER_IB_MSG_1050; SQLSTATE: HY000

  Message: Can't set undo tablespace(s) to be encrypted since --innodb_undo_tablespaces=0.

  ER_IB_MSG_1050 was added in 8.0.11, removed after 8.0.15.

- Error number: MY-012876; Symbol: ER_IB_MSG_1051; SQLSTATE: HY000

  Message: Can't set undo tablespace(s) to be encrypted in read-only-mode.

  ER_IB_MSG_1051 was added in 8.0.11.

- Error number: MY-012877; Symbol: ER_IB_MSG_1052; SQLSTATE: HY000

  Message: Can't set undo tablespace '%s' to be encrypted.

  ER_IB_MSG_1052 was added in 8.0.11.

- Error number: MY-012878; Symbol: ER_IB_MSG_1053; SQLSTATE: HY000

  Message: Can't set undo tablespace '%s' to be encrypted. Failed to write header page.

  ER_IB_MSG_1053 was added in 8.0.11.

- Error number: MY-012879; Symbol: ER_IB_MSG_1054; SQLSTATE: HY000

  Message: Can't set undo tablespace '%s' to be encrypted. Error %d - %s

  ER_IB_MSG_1054 was added in 8.0.11.
- Error number: MY-012880; Symbol: ER_IB_MSG_1055; SQLSTATE: HY000

  Message: Encryption is enabled for undo tablespace '%s'.

  ER_IB_MSG_1055 was added in 8.0.11.
- Error number: MY-012881; Symbol: ER_IB_MSG_1056; SQLSTATE: HY000

  Message: Can't rotate encryption on undo tablespace '%s'.

  ER_IB_MSG_1056 was added in 8.0.11.
- Error number: MY-012882; Symbol: ER_IB_MSG_1057; SQLSTATE: HY000

  Message: Encryption is enabled for undo tablespace '%s'.

  ER_IB_MSG_1057 was added in 8.0.11.
- Error number: MY-012883; Symbol: ER_IB_MSG_1058; SQLSTATE: HY000

  Message: os_file_get_status() failed on '%s'. Can't determine file permissions.

  ER_IB_MSG_1058 was added in 8.0.11.
- Error number: MY-012884; Symbol: ER_IB_MSG_1059; SQLSTATE: HY000

  Message: %s can't be opened in %s mode.

  ER_IB_MSG_1059 was added in 8.0.11.
- Error number: MY-012885; Symbol: ER_IB_MSG_1060; SQLSTATE: HY000

  Message: '%s' not a regular file.

  ER_IB_MSG_1060 was added in 8.0.11.
- Error number: MY-012886; Symbol: ER_IB_MSG_1061; SQLSTATE: HY000

  Message: Cannot create %s

  ER_IB_MSG_1061 was added in 8.0.11.
- Error number: MY-012887; Symbol: ER_IB_MSG_1062; SQLSTATE: HY000

  Message: Setting log file %s size to %llu MB

  ER_IB_MSG_1062 was added in 8.0.11.
- Error number: MY-012888; Symbol: ER_IB_MSG_1063; SQLSTATE: HY000

  Message: Cannot set log file %s to size %llu MB

  ER_IB_MSG_1063 was added in 8.0.11.

- Error number: MY-012889; Symbol: ER_IB_MSG_1064; SQLSTATE: HY000

  Message: Cannot create log files in read-only mode

  ER_IB_MSG_1064 was added in 8.0.11.

- Error number: MY-012890; Symbol: ER_IB_MSG_1065; SQLSTATE: HY000

  Message: Redo log encryption is enabled, but the keyring plugin is not loaded.

  ER_IB_MSG_1065 was added in 8.0.11.

- Error number: MY-012891; Symbol: ER_IB_MSG_1066; SQLSTATE: HY000

  Message: Cannot create file for log file %s.

  ER_IB_MSG_1066 was added in 8.0.11.

- Error number: MY-012892; Symbol: ER_IB_MSG_1067; SQLSTATE: HY000

  Message: Renaming log file %s to %s

  ER_IB_MSG_1067 was added in 8.0.11.

- Error number: MY-012893; Symbol: ER_IB_MSG_1068; SQLSTATE: HY000

  Message: New log files created, LSN=%llu

  ER_IB_MSG_1068 was added in 8.0.11.

- Error number: MY-012894; Symbol: ER_IB_MSG_1069; SQLSTATE: HY000

  Message: Unable to open '%s'.

  ER_IB_MSG_1069 was added in 8.0.11.

- Error number: MY-012895; Symbol: ER_IB_MSG_1070; SQLSTATE: HY000

  Message: Cannot create construction log file '%s' for undo tablespace '%s'.

  ER_IB_MSG_1070 was added in 8.0.11.

- Error number: MY-012896; Symbol: ER_IB_MSG_1071; SQLSTATE: HY000

  Message: Creating UNDO Tablespace %s

  ER_IB_MSG_1071 was added in 8.0.11.

- Error number: MY-012897; Symbol: ER_IB_MSG_1072; SQLSTATE: HY000

  Message: Setting file %s size to %llu MB

  ER_IB_MSG_1072 was added in 8.0.11.

- Error number: MY-012898; Symbol: ER_IB_MSG_1073; SQLSTATE: HY000

  Message: Physically writing the file full

  ER_IB_MSG_1073 was added in 8.0.11.

- Error number: MY-012899; Symbol: ER_IB_MSG_1074; SQLSTATE: HY000

  Message: Error in creating %s: probably out of disk space

  ER_IB_MSG_1074 was added in 8.0.11.

- Error number: MY-012900; Symbol: ER_IB_MSG_1075; SQLSTATE: HY000

  Message: Can't set encryption metadata for space %s

  ER_IB_MSG_1075 was added in 8.0.11.

- Error number: MY-012901; Symbol: ER_IB_MSG_1076; SQLSTATE: HY000

  Message: Cannot read first page of '%s' - %s

  ER_IB_MSG_1076 was added in 8.0.11.

- Error number: MY-012902; Symbol: ER_IB_MSG_1077; SQLSTATE: HY000

  Message: Undo tablespace number %lu was being truncated when mysqld quit.

  ER_IB_MSG_1077 was added in 8.0.11.

- Error number: MY-012903; Symbol: ER_IB_MSG_1078; SQLSTATE: HY000

  Message: Cannot recover a truncated undo tablespace in read-only mode

  ER_IB_MSG_1078 was added in 8.0.11.

- Error number: MY-012904; Symbol: ER_IB_MSG_1079; SQLSTATE: HY000

  Message: Reconstructing undo tablespace number %lu.

  ER_IB_MSG_1079 was added in 8.0.11.

- Error number: MY-012905; Symbol: ER_IB_MSG_1080; SQLSTATE: HY000

  Message: Cannot create %s because %s already uses Space ID=%lu! Did you change innodb_undo_directory?

  ER_IB_MSG_1080 was added in 8.0.11.

- Error number: MY-012906; Symbol: ER_IB_MSG_1081; SQLSTATE: HY000

  Message: UNDO tablespace %s must be %s

  ER_IB_MSG_1081 was added in 8.0.11.

- Error number: MY-012907; Symbol: ER_IB_MSG_1082; SQLSTATE: HY000

  Message: Error creating file for %s

  ER_IB_MSG_1082 was added in 8.0.11.

- Error number: MY-012908; Symbol: ER_IB_MSG_1083; SQLSTATE: HY000

  Message: Error reading encryption for %s

  ER_IB_MSG_1083 was added in 8.0.11.

- Error number: MY-012909; Symbol: ER_IB_MSG_1084; SQLSTATE: HY000

  Message: Unable to open undo tablespace number %lu

  ER_IB_MSG_1084 was added in 8.0.11.

- Error number: MY-012910; Symbol: ER_IB_MSG_1085; SQLSTATE: HY000

  Message: Opened %llu existing undo tablespaces.

  ER_IB_MSG_1085 was added in 8.0.11.

- Error number: MY-012911; Symbol: ER_IB_MSG_1086; SQLSTATE: HY000

  Message: Cannot create undo tablespaces since innodb_%s has been set. Using %llu existing undo tablespaces.

  ER_IB_MSG_1086 was added in 8.0.11.

- Error number: MY-012912; Symbol: ER_IB_MSG_1087; SQLSTATE: HY000

  Message: Cannot continue InnoDB startup in %s mode because there are no existing undo tablespaces found.

  ER_IB_MSG_1087 was added in 8.0.11.

- Error number: MY-012913; Symbol: ER_IB_MSG_1088; SQLSTATE: HY000

  Message: Could not create undo tablespace '%s'.

  ER_IB_MSG_1088 was added in 8.0.11.

- Error number: MY-012914; Symbol: ER_IB_MSG_1089; SQLSTATE: HY000

  Message: Error %d - %s - opening newly created undo tablespace '%s'.

  ER_IB_MSG_1089 was added in 8.0.11.

- Error number: MY-012915; Symbol: ER_IB_MSG_1090; SQLSTATE: HY000

  Message: Created %llu undo tablespaces.

  ER_IB_MSG_1090 was added in 8.0.11.

- Error number: MY-012916; Symbol: ER_IB_MSG_1091; SQLSTATE: HY000

  Message: Unable to create encrypted undo tablespace number %lu. please check if the keyring plugin is initialized correctly

  ER_IB_MSG_1091 was added in 8.0.11.

- Error number: MY-012917; Symbol: ER_IB_MSG_1092; SQLSTATE: HY000

  Message: Encryption is enabled for undo tablespace number %lu.

  ER_IB_MSG_1092 was added in 8.0.11.

- Error number: MY-012918; Symbol: ER_IB_MSG_1093; SQLSTATE: HY000

  Message: Unable to initialize the header page in undo tablespace number %lu.

ER_IB_MSG_1093 was added in 8.0.11.

- Error number: MY-012919; Symbol: ER_IB_MSG_1094; SQLSTATE: HY000

  Message: Cannot delete old undo tablespaces because they contain undo logs for XA PREPARED transactions.

  ER_IB_MSG_1094 was added in 8.0.11.

- Error number: MY-012920; Symbol: ER_IB_MSG_1095; SQLSTATE: HY000

  Message: Upgrading %zu existing undo tablespaces that were tracked in the system tablespace to %lu new independent undo tablespaces.

  ER_IB_MSG_1095 was added in 8.0.11.

- Error number: MY-012921; Symbol: ER_IB_MSG_1096; SQLSTATE: HY000

  Message: Deleting %llu new independent undo tablespaces that we just created.

  ER_IB_MSG_1096 was added in 8.0.11.

- Error number: MY-012922; Symbol: ER_IB_MSG_1097; SQLSTATE: HY000

  Message: Waiting for purge to start

  ER_IB_MSG_1097 was added in 8.0.11.

- Error number: MY-012923; Symbol: ER_IB_MSG_1098; SQLSTATE: HY000

  Message: Creating shared tablespace for temporary tables

  ER_IB_MSG_1098 was added in 8.0.11.

- Error number: MY-012924; Symbol: ER_IB_MSG_1099; SQLSTATE: HY000

  Message: The %s data file must be writable!

  ER_IB_MSG_1099 was added in 8.0.11.

- Error number: MY-012925; Symbol: ER_IB_MSG_1100; SQLSTATE: HY000

  Message: Could not create the shared %s.

  ER_IB_MSG_1100 was added in 8.0.11.

- Error number: MY-012926; Symbol: ER_IB_MSG_1101; SQLSTATE: HY000

  Message: Unable to create the shared %s.

  ER_IB_MSG_1101 was added in 8.0.11.

- Error number: MY-012927; Symbol: ER_IB_MSG_1102; SQLSTATE: HY000

  Message: The %s data file cannot be re-opened after check_file_spec() succeeded!

  ER_IB_MSG_1102 was added in 8.0.11.

- Error number: MY-012928; Symbol: ER_IB_MSG_1103; SQLSTATE: HY000

Message: %d threads created by InnoDB had not exited at shutdown!

ER_IB_MSG_1103 was added in 8.0.11.

- Error number: MY-012929; Symbol: ER_IB_MSG_1104; SQLSTATE: HY000

  Message: InnoDB Database creation was aborted %swith error %s. You may need to delete the ibdata1 file before trying to start up again.

  ER_IB_MSG_1104 was added in 8.0.11.

- Error number: MY-012930; Symbol: ER_IB_MSG_1105; SQLSTATE: HY000

  Message: Plugin initialization aborted %swith error %s.

  ER_IB_MSG_1105 was added in 8.0.11.

- Error number: MY-012931; Symbol: ER_IB_MSG_1106; SQLSTATE: HY000

  Message: Waiting for %llu buffer page I/Os to complete

  ER_IB_MSG_1106 was added in 8.0.11.

- Error number: MY-012932; Symbol: ER_IB_MSG_1107; SQLSTATE: HY000

  Message: PUNCH HOLE support available

  ER_IB_MSG_1107 was added in 8.0.11.

- Error number: MY-012933; Symbol: ER_IB_MSG_1108; SQLSTATE: HY000

  Message: PUNCH HOLE support not available

  ER_IB_MSG_1108 was added in 8.0.11.

- Error number: MY-012934; Symbol: ER_IB_MSG_1109; SQLSTATE: HY000

  Message: Size of InnoDB's ulint is %zu but size of void* is %zu. The sizes should be the same so that on a 64-bit platforms you can allocate more than 4 GB of memory.

  ER_IB_MSG_1109 was added in 8.0.11.

- Error number: MY-012935; Symbol: ER_IB_MSG_1110; SQLSTATE: HY000

  Message: Database upgrade cannot be accomplished in read-only mode.

  ER_IB_MSG_1110 was added in 8.0.11.

- Error number: MY-012936; Symbol: ER_IB_MSG_1111; SQLSTATE: HY000

  Message: Database upgrade cannot be accomplished with innodb_force_recovery > 0

  ER_IB_MSG_1111 was added in 8.0.11.

- Error number: MY-012937; Symbol: ER_IB_MSG_1112; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1112 was added in 8.0.11.

- Error number: MY-012938; Symbol: ER_IB_MSG_1113; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1113 was added in 8.0.11.

- Error number: MY-012939; Symbol: ER_IB_MSG_1114; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1114 was added in 8.0.11.

- Error number: MY-012940; Symbol: ER_IB_MSG_1115; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1115 was added in 8.0.11.

- Error number: MY-012941; Symbol: ER_IB_MSG_1116; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1116 was added in 8.0.11.

- Error number: MY-012942; Symbol: ER_IB_MSG_1117; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1117 was added in 8.0.11.

- Error number: MY-012943; Symbol: ER_IB_MSG_1118; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1118 was added in 8.0.11.

- Error number: MY-012944; Symbol: ER_IB_MSG_1119; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1119 was added in 8.0.11.

- Error number: MY-012945; Symbol: ER_IB_MSG_1120; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1120 was added in 8.0.11.

- Error number: MY-012946; Symbol: ER_IB_MSG_1121; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1121 was added in 8.0.11.

- Error number: MY-012947; Symbol: ER_IB_MSG_1122; SQLSTATE: HY000

  Message: MySQL was built without a memory barrier capability on this architecture, which might allow a mutex/rw_lock violation under high thread concurrency. This may cause a hang.

  ER_IB_MSG_1122 was added in 8.0.11.

- Error number: MY-012948; Symbol: ER_IB_MSG_1123; SQLSTATE: HY000

  Message: Compressed tables use zlib %s

  ER_IB_MSG_1123 was added in 8.0.11.

- Error number: MY-012949; Symbol: ER_IB_MSG_1124; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1124 was added in 8.0.11.

- Error number: MY-012950; Symbol: ER_IB_MSG_1125; SQLSTATE: HY000

  Message: Startup called second time during the process lifetime. In the MySQL Embedded Server Library you cannot call server_init() more than once during the process lifetime.

  ER_IB_MSG_1125 was added in 8.0.11.

- Error number: MY-012951; Symbol: ER_IB_MSG_1126; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1126 was added in 8.0.11.

- Error number: MY-012952; Symbol: ER_IB_MSG_1127; SQLSTATE: HY000

  Message: Unable to create monitor file %s: %s

  ER_IB_MSG_1127 was added in 8.0.11.

- Error number: MY-012953; Symbol: ER_IB_MSG_1128; SQLSTATE: HY000

  Message: Disabling background log and ibuf IO write threads.

  ER_IB_MSG_1128 was added in 8.0.11.

- Error number: MY-012954; Symbol: ER_IB_MSG_1129; SQLSTATE: HY000

  Message: Cannot initialize AIO sub-system

  ER_IB_MSG_1129 was added in 8.0.11.

- Error number: MY-012955; Symbol: ER_IB_MSG_1130; SQLSTATE: HY000

  Message: Initializing buffer pool, total size = %lf%c, instances = %lu, chunk size =%lf%c

  ER_IB_MSG_1130 was added in 8.0.11.

- Error number: MY-012956; Symbol: ER_IB_MSG_1131; SQLSTATE: HY000

  Message: Cannot allocate memory for the buffer pool

  ER_IB_MSG_1131 was added in 8.0.11.

- Error number: MY-012957; Symbol: ER_IB_MSG_1132; SQLSTATE: HY000

  Message: Completed initialization of buffer pool

  ER_IB_MSG_1132 was added in 8.0.11.

- Error number: MY-012958; Symbol: ER_IB_MSG_1133; SQLSTATE: HY000

  Message: Small buffer pool size (%lluM), the flst_validate() debug function can cause a deadlock if the buffer pool fills up.

  ER_IB_MSG_1133 was added in 8.0.11.

- Error number: MY-012959; Symbol: ER_IB_MSG_1134; SQLSTATE: HY000

  Message: Could not open or create the system tablespace. If you tried to add new data files to the system tablespace, and it failed here, you should now edit innodb_data_file_path in my.cnf back to what it was, and remove the new ibdata files InnoDB created in this failed attempt. InnoDB only wrote those files full of zeros, but did not yet use them in any way. But be careful: do not remove old data files which contain your precious data!

  ER_IB_MSG_1134 was added in 8.0.11.

- Error number: MY-012960; Symbol: ER_IB_MSG_1135; SQLSTATE: HY000

  Message: Cannot create log files because data files are corrupt or the database was not shut down cleanly after creating the data files.

  ER_IB_MSG_1135 was added in 8.0.11.

- Error number: MY-012961; Symbol: ER_IB_MSG_1136; SQLSTATE: HY000

  Message: Only one log file found

  ER_IB_MSG_1136 was added in 8.0.11.

- Error number: MY-012962; Symbol: ER_IB_MSG_1137; SQLSTATE: HY000

  Message: Log file %s size %llu is not a multiple of innodb_page_size

  ER_IB_MSG_1137 was added in 8.0.11.

- Error number: MY-012963; Symbol: ER_IB_MSG_1138; SQLSTATE: HY000

  Message: Log file %s is of different size %llu bytes than other log files %llu bytes!

  ER_IB_MSG_1138 was added in 8.0.11.

- Error number: MY-012964; Symbol: ER_IB_MSG_1139; SQLSTATE: HY000

  Message: Use --innodb-directories to find the tablespace files. If that fails then use --innodb-force-recovery=1 to ignore this and to permanently lose all changes to the missing tablespace(s)

  ER_IB_MSG_1139 was added in 8.0.11.

- Error number: MY-012965; Symbol: ER_IB_MSG_1140; SQLSTATE: HY000

  Message: The log file may have been corrupt and it is possible that the log scan or parsing did not proceed far enough in recovery. Please run CHECK TABLE on your InnoDB tables to check that they are ok! It may be safest to recover your InnoDB database from a backup!

  ER_IB_MSG_1140 was added in 8.0.11.

- Error number: MY-012966; Symbol: ER_IB_MSG_1141; SQLSTATE: HY000

Message: Cannot resize log files in read-only mode.

ER_IB_MSG_1141 was added in 8.0.11.

- Error number: MY-012967; Symbol: ER_IB_MSG_1142; SQLSTATE: HY000

  Message: Cannot open DD tablespace.

  ER_IB_MSG_1142 was added in 8.0.11.

- Error number: MY-012968; Symbol: ER_IB_MSG_1143; SQLSTATE: HY000

  Message: Starting to delete and rewrite log files.

  ER_IB_MSG_1143 was added in 8.0.11.

- Error number: MY-012969; Symbol: ER_IB_MSG_1144; SQLSTATE: HY000

  Message: Undo from 5.7 found. It will be purged

  ER_IB_MSG_1144 was added in 8.0.11.

- Error number: MY-012970; Symbol: ER_IB_MSG_1145; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1145 was added in 8.0.11.

- Error number: MY-012971; Symbol: ER_IB_MSG_1146; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1146 was added in 8.0.11.

- Error number: MY-012972; Symbol: ER_IB_MSG_1147; SQLSTATE: HY000

  Message: Tablespace size stored in header is %lu pages, but the sum of data file sizes is %lu pages

  ER_IB_MSG_1147 was added in 8.0.11.

- Error number: MY-012973; Symbol: ER_IB_MSG_1148; SQLSTATE: HY000

  Message: Cannot start InnoDB. The tail of the system tablespace is missing. Have you edited innodb_data_file_path in my.cnf in an inappropriate way, removing ibdata files from there? You can set innodb_force_recovery=1 in my.cnf to force a startup if you are trying to recover a badly corrupt database.

  ER_IB_MSG_1148 was added in 8.0.11.

- Error number: MY-012974; Symbol: ER_IB_MSG_1149; SQLSTATE: HY000

  Message: Tablespace size stored in header is %lu pages, but the sum of data file sizes is only %lu pages

  ER_IB_MSG_1149 was added in 8.0.11.

- Error number: MY-012975; Symbol: ER_IB_MSG_1150; SQLSTATE: HY000

Message: Cannot start InnoDB. The tail of the system tablespace is missing. Have you edited innodb_data_file_path in my.cnf in an InnoDB: inappropriate way, removing ibdata files from there? You can set innodb_force_recovery=1 in my.cnf to force InnoDB: a startup if you are trying to recover a badly corrupt database.

`ER_IB_MSG_1150` was added in 8.0.11.

- Error number: `MY-012976`; Symbol: `ER_IB_MSG_1151`; SQLSTATE: `HY000`

  Message: %s started; log sequence number %llu

  `ER_IB_MSG_1151` was added in 8.0.11.

- Error number: `MY-012977`; Symbol: `ER_IB_MSG_1152`; SQLSTATE: `HY000`

  Message: Waiting for purge to complete

  `ER_IB_MSG_1152` was added in 8.0.11.

- Error number: `MY-012978`; Symbol: `ER_IB_MSG_1153`; SQLSTATE: `HY000`

  Message: Waiting for dict_stats_thread to exit

  `ER_IB_MSG_1153` was added in 8.0.11.

- Error number: `MY-012979`; Symbol: `ER_IB_MSG_1154`; SQLSTATE: `HY000`

  Message: Query counter shows %llu queries still inside InnoDB at shutdown

  `ER_IB_MSG_1154` was added in 8.0.11.

- Error number: `MY-012980`; Symbol: `ER_IB_MSG_1155`; SQLSTATE: `HY000`

  Message: Shutdown completed; log sequence number %llu

  `ER_IB_MSG_1155` was added in 8.0.11.

- Error number: `MY-012981`; Symbol: `ER_IB_MSG_1156`; SQLSTATE: `HY000`

  Message: Cannot continue operation.

  `ER_IB_MSG_1156` was added in 8.0.11.

- Error number: `MY-012982`; Symbol: `ER_IB_MSG_1157`; SQLSTATE: `HY000`

  Message: %s

  `ER_IB_MSG_1157` was added in 8.0.11.

- Error number: `MY-012983`; Symbol: `ER_IB_MSG_1158`; SQLSTATE: `HY000`

  Message: %s

  `ER_IB_MSG_1158` was added in 8.0.11.

- Error number: `MY-012984`; Symbol: `ER_IB_MSG_1159`; SQLSTATE: `HY000`

  Message: %s

ER_IB_MSG_1159 was added in 8.0.11.

- Error number: MY-012985; Symbol: ER_IB_MSG_1160; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1160 was added in 8.0.11.

- Error number: MY-012986; Symbol: ER_IB_MSG_1161; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1161 was added in 8.0.11.

- Error number: MY-012987; Symbol: ER_IB_MSG_1162; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1162 was added in 8.0.11.

- Error number: MY-012988; Symbol: ER_IB_MSG_1163; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1163 was added in 8.0.11.

- Error number: MY-012989; Symbol: ER_IB_MSG_1164; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1164 was added in 8.0.11.

- Error number: MY-012990; Symbol: ER_IB_MSG_1165; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1165 was added in 8.0.11.

- Error number: MY-012991; Symbol: ER_IB_MSG_1166; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1166 was added in 8.0.11.

- Error number: MY-012992; Symbol: ER_IB_MSG_1167; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1167 was added in 8.0.11.

- Error number: MY-012993; Symbol: ER_IB_MSG_1168; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1168 was added in 8.0.11.

- Error number: MY-012994; Symbol: ER_IB_MSG_1169; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1169 was added in 8.0.11.

- Error number: MY-012995; Symbol: ER_IB_MSG_1170; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1170 was added in 8.0.11.

- Error number: MY-012996; Symbol: ER_IB_MSG_1171; SQLSTATE: HY000

  Message: Cannot create truncate log for undo tablespace '%s'.

  ER_IB_MSG_1171 was added in 8.0.11.

- Error number: MY-012997; Symbol: ER_IB_MSG_1172; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1172 was added in 8.0.11.

- Error number: MY-012998; Symbol: ER_IB_MSG_1173; SQLSTATE: HY000

  Message: Failed to truncate undo tablespace '%s'.

  ER_IB_MSG_1173 was added in 8.0.11.

- Error number: MY-012999; Symbol: ER_IB_MSG_1174; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1174 was added in 8.0.11.

- Error number: MY-013000; Symbol: ER_IB_MSG_1175; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1175 was added in 8.0.11.

- Error number: MY-013001; Symbol: ER_IB_MSG_1176; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1176 was added in 8.0.11.

- Error number: MY-013002; Symbol: ER_IB_MSG_1177; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1177 was added in 8.0.11.

- Error number: MY-013003; Symbol: ER_IB_MSG_1178; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1178 was added in 8.0.11.

- Error number: MY-013004; Symbol: ER_IB_MSG_1179; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1179 was added in 8.0.11.

- Error number: MY-013005; Symbol: ER_IB_MSG_1180; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1180 was added in 8.0.11.

- Error number: MY-013006; Symbol: ER_IB_MSG_1181; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1181 was added in 8.0.11.

- Error number: MY-013007; Symbol: ER_IB_MSG_1182; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1182 was added in 8.0.11.

- Error number: MY-013008; Symbol: ER_IB_MSG_1183; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1183 was added in 8.0.11.

- Error number: MY-013009; Symbol: ER_IB_MSG_1184; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1184 was added in 8.0.11.

- Error number: MY-013010; Symbol: ER_IB_MSG_1185; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1185 was added in 8.0.11.

- Error number: MY-013011; Symbol: ER_IB_MSG_1186; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1186 was added in 8.0.11.

- Error number: MY-013012; Symbol: ER_IB_MSG_1187; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1187 was added in 8.0.11.

- Error number: MY-013013; Symbol: ER_IB_MSG_1188; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1188 was added in 8.0.11.

- Error number: MY-013014; Symbol: ER_IB_MSG_1189; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1189 was added in 8.0.11.

- Error number: MY-013015; Symbol: ER_IB_MSG_1190; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1190 was added in 8.0.11.

- Error number: MY-013016; Symbol: ER_IB_MSG_1191; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1191 was added in 8.0.11.

- Error number: MY-013017; Symbol: ER_IB_MSG_1192; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1192 was added in 8.0.11.

- Error number: MY-013018; Symbol: ER_IB_MSG_1193; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1193 was added in 8.0.11.

- Error number: MY-013019; Symbol: ER_IB_MSG_1194; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1194 was added in 8.0.11.

- Error number: MY-013020; Symbol: ER_IB_MSG_1195; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1195 was added in 8.0.11.

- Error number: MY-013021; Symbol: ER_IB_MSG_1196; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1196 was added in 8.0.11.

- Error number: MY-013022; Symbol: ER_IB_MSG_1197; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1197 was added in 8.0.11.

- Error number: MY-013023; Symbol: ER_IB_MSG_1198; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1198 was added in 8.0.11.

- Error number: MY-013024; Symbol: ER_IB_MSG_1199; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1199 was added in 8.0.11.

- Error number: MY-013025; Symbol: ER_IB_MSG_1200; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1200 was added in 8.0.11.

- Error number: MY-013026; Symbol: ER_IB_MSG_1201; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1201 was added in 8.0.11.

- Error number: MY-013027; Symbol: ER_IB_MSG_1202; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1202 was added in 8.0.11.

- Error number: MY-013028; Symbol: ER_IB_MSG_1203; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1203 was added in 8.0.11.

- Error number: MY-013029; Symbol: ER_IB_MSG_1204; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1204 was added in 8.0.11.

- Error number: MY-013030; Symbol: ER_IB_MSG_1205; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1205 was added in 8.0.11.

- Error number: MY-013031; Symbol: ER_IB_MSG_1206; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1206 was added in 8.0.11.

- Error number: MY-013032; Symbol: ER_IB_MSG_1207; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1207 was added in 8.0.11.

- Error number: MY-013033; Symbol: ER_IB_MSG_1208; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1208 was added in 8.0.11.

- Error number: MY-013034; Symbol: ER_IB_MSG_1209; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1209 was added in 8.0.11.

- Error number: MY-013035; Symbol: ER_IB_MSG_1210; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1210 was added in 8.0.11.

- Error number: MY-013036; Symbol: ER_IB_MSG_1211; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1211 was added in 8.0.11.

- Error number: MY-013037; Symbol: ER_IB_MSG_1212; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1212 was added in 8.0.11.

- Error number: MY-013038; Symbol: ER_IB_MSG_1213; SQLSTATE: HY000

  Message: gettimeofday() failed: %s

  ER_IB_MSG_1213 was added in 8.0.11.

- Error number: MY-013039; Symbol: ER_IB_MSG_1214; SQLSTATE: HY000

  Message: Can't create UNDO tablespace %s %s

  ER_IB_MSG_1214 was added in 8.0.11.

- Error number: MY-013040; Symbol: ER_IB_MSG_1215; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1215 was added in 8.0.11.

- Error number: MY-013041; Symbol: ER_IB_MSG_1216; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1216 was added in 8.0.11.

- Error number: MY-013042; Symbol: ER_IB_MSG_1217; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1217 was added in 8.0.11.

- Error number: MY-013043; Symbol: ER_IB_MSG_1218; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1218 was added in 8.0.11.

- Error number: MY-013044; Symbol: ER_IB_MSG_1219; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1219 was added in 8.0.11.

- Error number: MY-013045; Symbol: ER_IB_MSG_1220; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1220 was added in 8.0.11.

- Error number: MY-013046; Symbol: ER_IB_MSG_1221; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1221 was added in 8.0.11.

- Error number: MY-013047; Symbol: ER_IB_MSG_1222; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1222 was added in 8.0.11.

- Error number: MY-013048; Symbol: ER_IB_MSG_1223; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1223 was added in 8.0.11.

- Error number: MY-013049; Symbol: ER_IB_MSG_1224; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1224 was added in 8.0.11.

- Error number: MY-013050; Symbol: ER_IB_MSG_1225; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1225 was added in 8.0.11.

- Error number: MY-013051; Symbol: ER_IB_MSG_1226; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1226 was added in 8.0.11.

- Error number: MY-013052; Symbol: ER_IB_MSG_1227; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1227 was added in 8.0.11.

- Error number: MY-013053; Symbol: ER_IB_MSG_1228; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1228 was added in 8.0.11.

- Error number: MY-013054; Symbol: ER_IB_MSG_1229; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1229 was added in 8.0.11.

- Error number: MY-013055; Symbol: ER_IB_MSG_1230; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1230 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-013056; Symbol: ER_IB_MSG_1231; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1231 was added in 8.0.11.

- Error number: MY-013057; Symbol: ER_IB_MSG_1232; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1232 was added in 8.0.11.

- Error number: MY-013058; Symbol: ER_IB_MSG_1233; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1233 was added in 8.0.11.

- Error number: MY-013059; Symbol: ER_IB_MSG_1234; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1234 was added in 8.0.11.

- Error number: MY-013060; Symbol: ER_IB_MSG_1235; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1235 was added in 8.0.11.

- Error number: MY-013061; Symbol: ER_IB_MSG_1236; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1236 was added in 8.0.11.

- Error number: MY-013062; Symbol: ER_IB_MSG_1237; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1237 was added in 8.0.11.

- Error number: MY-013063; Symbol: ER_IB_MSG_1238; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1238 was added in 8.0.11.

- Error number: MY-013064; Symbol: ER_IB_MSG_1239; SQLSTATE: HY000

  Message: %s

ER_IB_MSG_1239 was added in 8.0.11.

- Error number: MY-013065; Symbol: ER_IB_MSG_1240; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1240 was added in 8.0.11.

- Error number: MY-013066; Symbol: ER_IB_MSG_1241; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1241 was added in 8.0.11.

- Error number: MY-013067; Symbol: ER_IB_MSG_1242; SQLSTATE: HY000

  Message: Can't set redo log tablespace to be encrypted in read-only mode.

  ER_IB_MSG_1242 was added in 8.0.11.

- Error number: MY-013068; Symbol: ER_IB_MSG_1243; SQLSTATE: HY000

  Message: Can't set redo log tablespace to be encrypted.

  ER_IB_MSG_1243 was added in 8.0.11.

- Error number: MY-013069; Symbol: ER_IB_MSG_1244; SQLSTATE: HY000

  Message: Can't set redo log tablespace to be encrypted.

  ER_IB_MSG_1244 was added in 8.0.11.

- Error number: MY-013070; Symbol: ER_IB_MSG_1245; SQLSTATE: HY000

  Message: Redo log encryption is enabled.

  ER_IB_MSG_1245 was added in 8.0.11.

- Error number: MY-013071; Symbol: ER_IB_MSG_1246; SQLSTATE: HY000

  Message: Waiting for archiver to finish archiving page and log

  ER_IB_MSG_1246 was added in 8.0.11.

- Error number: MY-013072; Symbol: ER_IB_MSG_1247; SQLSTATE: HY000

  Message: Starting shutdown...

  ER_IB_MSG_1247 was added in 8.0.11.

- Error number: MY-013073; Symbol: ER_IB_MSG_1248; SQLSTATE: HY000

  Message: Waiting for %s to exit.

  ER_IB_MSG_1248 was added in 8.0.11.

- Error number: MY-013074; Symbol: ER_IB_MSG_1249; SQLSTATE: HY000

  Message: Waiting for %lu active transactions to finish

ER_IB_MSG_1249 was added in 8.0.11.

- Error number: MY-013075; Symbol: ER_IB_MSG_1250; SQLSTATE: HY000

  Message: Waiting for master thread to be suspended.

  ER_IB_MSG_1250 was added in 8.0.11.

- Error number: MY-013076; Symbol: ER_IB_MSG_1251; SQLSTATE: HY000

  Message: Waiting for page_cleaner to finish flushing of buffer pool.

  ER_IB_MSG_1251 was added in 8.0.11.

- Error number: MY-013077; Symbol: ER_IB_MSG_1252; SQLSTATE: HY000

  Message: Waiting for %lu buffer page I/Os to complete.

  ER_IB_MSG_1252 was added in 8.0.11.

- Error number: MY-013078; Symbol: ER_IB_MSG_1253; SQLSTATE: HY000

  Message: MySQL has requested a very fast shutdown without flushing the InnoDB buffer pool to data files. At the next mysqld startup InnoDB will do a crash recovery!

  ER_IB_MSG_1253 was added in 8.0.11.

- Error number: MY-013079; Symbol: ER_IB_MSG_1254; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1254 was added in 8.0.11, removed after 8.0.13.

- Error number: MY-013080; Symbol: ER_IB_MSG_1255; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1255 was added in 8.0.11.

- Error number: MY-013081; Symbol: ER_IB_MSG_1256; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1256 was added in 8.0.11.

- Error number: MY-013082; Symbol: ER_IB_MSG_1257; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1257 was added in 8.0.11.

- Error number: MY-013083; Symbol: ER_IB_MSG_1258; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1258 was added in 8.0.11.

- Error number: MY-013084; Symbol: ER_IB_MSG_1259; SQLSTATE: HY000

Message: %s

ER_IB_MSG_1259 was added in 8.0.11.

- Error number: MY-013085; Symbol: ER_IB_MSG_1260; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1260 was added in 8.0.11.

- Error number: MY-013086; Symbol: ER_IB_MSG_1261; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1261 was added in 8.0.11.

- Error number: MY-013087; Symbol: ER_IB_MSG_1262; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1262 was added in 8.0.11.

- Error number: MY-013088; Symbol: ER_IB_MSG_1263; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1263 was added in 8.0.11.

- Error number: MY-013089; Symbol: ER_IB_MSG_1264; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1264 was added in 8.0.11.

- Error number: MY-013090; Symbol: ER_IB_MSG_1265; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1265 was added in 8.0.11.

- Error number: MY-013091; Symbol: ER_IB_MSG_1266; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1266 was added in 8.0.11.

- Error number: MY-013092; Symbol: ER_IB_MSG_1267; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1267 was added in 8.0.11.

- Error number: MY-013093; Symbol: ER_IB_MSG_1268; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1268 was added in 8.0.11.

- Error number: MY-013094; Symbol: ER_IB_MSG_1269; SQLSTATE: HY000

Message: %s

ER_IB_MSG_1269 was added in 8.0.11.

- Error number: MY-013095; Symbol: ER_IB_MSG_1270; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1270 was added in 8.0.11.

- Error number: MY-013096; Symbol: ER_RPL_SLAVE_SQL_THREAD_STOP_CMD_EXEC_TIMEOUT; SQLSTATE: HY000

  Message: STOP SLAVE command execution is incomplete: Slave SQL thread got the stop signal, thread is busy, SQL thread will stop once the current task is complete.

  ER_RPL_SLAVE_SQL_THREAD_STOP_CMD_EXEC_TIMEOUT was added in 8.0.11.

- Error number: MY-013097; Symbol: ER_RPL_SLAVE_IO_THREAD_STOP_CMD_EXEC_TIMEOUT; SQLSTATE: HY000

  Message: STOP SLAVE command execution is incomplete: Slave IO thread got the stop signal, thread is busy, IO thread will stop once the current task is complete.

  ER_RPL_SLAVE_IO_THREAD_STOP_CMD_EXEC_TIMEOUT was added in 8.0.11.

- Error number: MY-013098; Symbol: ER_RPL_GTID_UNSAFE_STMT_ON_NON_TRANS_TABLE; SQLSTATE: HY000

  Message: Statement violates GTID consistency: Updates to non-transactional tables can only be done in either autocommitted statements or single-statement transactions, and never in the same statement as updates to transactional tables.

  ER_RPL_GTID_UNSAFE_STMT_ON_NON_TRANS_TABLE was added in 8.0.11.

- Error number: MY-013099; Symbol: ER_RPL_GTID_UNSAFE_STMT_CREATE_SELECT; SQLSTATE: HY000

  Message: Statement violates GTID consistency: CREATE TABLE ... SELECT.

  ER_RPL_GTID_UNSAFE_STMT_CREATE_SELECT was added in 8.0.11.

- Error number: MY-013100; Symbol: ER_RPL_GTID_UNSAFE_STMT_ON_TEMPORARY_TABLE; SQLSTATE: HY000

  Message: Statement violates GTID consistency: CREATE TEMPORARY TABLE and DROP TEMPORARY TABLE can only be executed outside transactional context. These statements are also not allowed in a function or trigger because functions and triggers are also considered to be multi-statement transactions.

  ER_RPL_GTID_UNSAFE_STMT_ON_TEMPORARY_TABLE was added in 8.0.11, removed after 8.0.12.

- Error number: MY-013101; Symbol: ER_BINLOG_ROW_VALUE_OPTION_IGNORED; SQLSTATE: HY000

  Message: When %s, the option binlog_row_value_options=%s will be ignored and updates will be written in full format to binary log.

ER_BINLOG_ROW_VALUE_OPTION_IGNORED was added in 8.0.11.

- Error number: MY-013102; Symbol: ER_BINLOG_USE_V1_ROW_EVENTS_IGNORED; SQLSTATE: HY000

  Message: When %s, the option log_bin_use_v1_row_events=1 will be ignored and row events will be written in new format to binary log.

  ER_BINLOG_USE_V1_ROW_EVENTS_IGNORED was added in 8.0.11.

- Error number: MY-013103; Symbol: ER_BINLOG_ROW_VALUE_OPTION_USED_ONLY_FOR_AFTER_IMAGES; SQLSTATE: HY000

  Message: When %s, the option binlog_row_value_options=%s will be used only for the after-image. Full values will be written in the before-image, so the saving in disk space due to binlog_row_value_options is limited to less than 50%%.

  ER_BINLOG_ROW_VALUE_OPTION_USED_ONLY_FOR_AFTER_IMAGES was added in 8.0.11.

- Error number: MY-013104; Symbol: ER_CONNECTION_ABORTED; SQLSTATE: HY000

  Message: Aborted connection %u to db: '%s' user: '%s' host: '%s' (%s).

  ER_CONNECTION_ABORTED was added in 8.0.11.

- Error number: MY-013105; Symbol: ER_NORMAL_SERVER_SHUTDOWN; SQLSTATE: HY000

  Message: %s: Normal shutdown.

  ER_NORMAL_SERVER_SHUTDOWN was added in 8.0.11.

- Error number: MY-013106; Symbol: ER_KEYRING_MIGRATE_FAILED; SQLSTATE: HY000

  Message: Can not perform keyring migration : %s.

  ER_KEYRING_MIGRATE_FAILED was added in 8.0.11.

- Error number: MY-013107; Symbol: ER_GRP_RPL_LOWER_CASE_TABLE_NAMES_DIFF_FROM_GRP; SQLSTATE: HY000

  Message: The member is configured with a lower_case_table_names option value '%u' different from the group '%u'. The member will now exit the group. If there is existing data on member, it may be incompatible with group if it was created with a lower_case_table_names value different from the group.

  ER_GRP_RPL_LOWER_CASE_TABLE_NAMES_DIFF_FROM_GRP was added in 8.0.11.

- Error number: MY-013108; Symbol: ER_OOM_SAVE_GTIDS; SQLSTATE: HY000

  Message: An out-of-memory error occurred while saving the set of GTIDs from the last binary log into the mysql.gtid_executed table

  ER_OOM_SAVE_GTIDS was added in 8.0.11.

- Error number: MY-013109; Symbol: ER_LCTN_NOT_FOUND; SQLSTATE: HY000

  Message: The lower_case_table_names setting for the data dictionary was not found. Starting the server using lower_case_table_names = '%u'.

ER_LCTN_NOT_FOUND was added in 8.0.11.

- Error number: MY-013111; Symbol: ER_COMPONENT_FILTER_WRONG_VALUE; SQLSTATE: HY000

  Message: Variable '%s' can't be set to the value of '%s'

  ER_COMPONENT_FILTER_WRONG_VALUE was added in 8.0.11.

- Error number: MY-013112; Symbol: ER_XPLUGIN_FAILED_TO_STOP_SERVICES; SQLSTATE: HY000

  Message: Stopping services failed with error "%s"

  ER_XPLUGIN_FAILED_TO_STOP_SERVICES was added in 8.0.11.

- Error number: MY-013113; Symbol: ER_INCONSISTENT_ERROR; SQLSTATE: HY000

  Message: Query caused different errors on master and slave. Error on master: message (format)='%s' error code=%d; Error on slave:actual message='%s', error code=%d. Default database:'%s'. Query:'%s'

- Error number: MY-013114; Symbol: ER_SERVER_MASTER_FATAL_ERROR_READING_BINLOG; SQLSTATE: HY000

  Message: Got fatal error %d from master when reading data from binary log: '%s'

  ER_SERVER_MASTER_FATAL_ERROR_READING_BINLOG was added in 8.0.11.

- Error number: MY-013115; Symbol: ER_NETWORK_READ_EVENT_CHECKSUM_FAILURE; SQLSTATE: HY000

  Message: Replication event checksum verification failed while reading from network.

- Error number: MY-013116; Symbol: ER_SLAVE_CREATE_EVENT_FAILURE; SQLSTATE: HY000

  Message: Failed to create %s

- Error number: MY-013117; Symbol: ER_SLAVE_FATAL_ERROR; SQLSTATE: HY000

  Message: Fatal error: %s

- Error number: MY-013118; Symbol: ER_SLAVE_HEARTBEAT_FAILURE; SQLSTATE: HY000

  Message: Unexpected master's heartbeat data: %s

- Error number: MY-013119; Symbol: ER_SLAVE_INCIDENT; SQLSTATE: HY000

  Message: The incident %s occurred on the master. Message: %s

- Error number: MY-013120; Symbol: ER_SLAVE_MASTER_COM_FAILURE; SQLSTATE: HY000

  Message: Master command %s failed: %s

- Error number: MY-013121; Symbol: ER_SLAVE_RELAY_LOG_READ_FAILURE; SQLSTATE: HY000

  Message: Relay log read failure: %s

- Error number: MY-013122; Symbol: ER_SLAVE_RELAY_LOG_WRITE_FAILURE; SQLSTATE: HY000

  Message: Relay log write failure: %s

- Error number: MY-013123; Symbol: ER_SERVER_SLAVE_MI_INIT_REPOSITORY; SQLSTATE: HY000

  Message: Slave failed to initialize master info structure from the repository

  ER_SERVER_SLAVE_MI_INIT_REPOSITORY was added in 8.0.11.

- Error number: MY-013124; Symbol: ER_SERVER_SLAVE_RLI_INIT_REPOSITORY; SQLSTATE: HY000

  Message: Slave failed to initialize relay log info structure from the repository

  ER_SERVER_SLAVE_RLI_INIT_REPOSITORY was added in 8.0.11.

- Error number: MY-013125; Symbol: ER_SERVER_NET_PACKET_TOO_LARGE; SQLSTATE: HY000

  Message: Got a packet bigger than 'max_allowed_packet' bytes

  ER_SERVER_NET_PACKET_TOO_LARGE was added in 8.0.11.

- Error number: MY-013126; Symbol: ER_SERVER_NO_SYSTEM_TABLE_ACCESS; SQLSTATE: HY000

  Message: Access to %s '%s.%s' is rejected.

  ER_SERVER_NO_SYSTEM_TABLE_ACCESS was added in 8.0.11.

- Error number: MY-013127; Symbol: ER_SERVER_UNKNOWN_ERROR; SQLSTATE: HY000

  Message: Unknown error

  ER_SERVER_UNKNOWN_ERROR was added in 8.0.11.

- Error number: MY-013128; Symbol: ER_SERVER_UNKNOWN_SYSTEM_VARIABLE; SQLSTATE: HY000

  Message: Unknown system variable '%s'

  ER_SERVER_UNKNOWN_SYSTEM_VARIABLE was added in 8.0.11.

- Error number: MY-013129; Symbol: ER_SERVER_NO_SESSION_TO_SEND_TO; SQLSTATE: HY000

  Message: A message intended for a client cannot be sent there as no client-session is attached. Therefore, we're sending the information to the error-log instead: MY-%06d - %s

  ER_SERVER_NO_SESSION_TO_SEND_TO was added in 8.0.11.

- Error number: MY-013130; Symbol: ER_SERVER_NEW_ABORTING_CONNECTION; SQLSTATE: 08S01

  Message: Aborted connection %u to db: '%s' user: '%s' host: '%s' (%s; diagnostics area: MY-%06d - %s)

  ER_SERVER_NEW_ABORTING_CONNECTION was added in 8.0.11.

- Error number: MY-013131; Symbol: ER_SERVER_OUT_OF_SORTMEMORY; SQLSTATE: HY000

  Message: Out of sort memory, consider increasing server sort buffer size!

  ER_SERVER_OUT_OF_SORTMEMORY was added in 8.0.11.

- Error number: MY-013132; Symbol: ER_SERVER_RECORD_FILE_FULL; SQLSTATE: HY000

  Message: The table '%s' is full!

ER_SERVER_RECORD_FILE_FULL was added in 8.0.11.

- Error number: MY-013133; Symbol: ER_SERVER_DISK_FULL_NOWAIT; SQLSTATE: HY000

  Message: Create table/tablespace '%s' failed, as disk is full.

  ER_SERVER_DISK_FULL_NOWAIT was added in 8.0.11.

- Error number: MY-013134; Symbol: ER_SERVER_HANDLER_ERROR; SQLSTATE: HY000

  Message: Handler reported error %d - %s

  ER_SERVER_HANDLER_ERROR was added in 8.0.11.

- Error number: MY-013135; Symbol: ER_SERVER_NOT_FORM_FILE; SQLSTATE: HY000

  Message: Incorrect information in file: '%s'

  ER_SERVER_NOT_FORM_FILE was added in 8.0.11.

- Error number: MY-013136; Symbol: ER_SERVER_CANT_OPEN_FILE; SQLSTATE: HY000

  Message: Can't open file: '%s' (OS errno: %d - %s)

  ER_SERVER_CANT_OPEN_FILE was added in 8.0.11.

- Error number: MY-013137; Symbol: ER_SERVER_FILE_NOT_FOUND; SQLSTATE: HY000

  Message: Can't find file: '%s' (OS errno: %d - %s)

  ER_SERVER_FILE_NOT_FOUND was added in 8.0.11.

- Error number: MY-013138; Symbol: ER_SERVER_FILE_USED; SQLSTATE: HY000

  Message: '%s' is locked against change (OS errno: %d - %s)

  ER_SERVER_FILE_USED was added in 8.0.11.

- Error number: MY-013139; Symbol: ER_SERVER_CANNOT_LOAD_FROM_TABLE_V2; SQLSTATE: HY000

  Message: Cannot load from %s.%s. The table is probably corrupted!

  ER_SERVER_CANNOT_LOAD_FROM_TABLE_V2 was added in 8.0.11.

- Error number: MY-013140; Symbol: ER_ERROR_INFO_FROM_DA; SQLSTATE: HY000

  Message: Error in diagnostics area: MY-%06d - %s

  ER_ERROR_INFO_FROM_DA was added in 8.0.11.

- Error number: MY-013141; Symbol: ER_SERVER_TABLE_CHECK_FAILED; SQLSTATE: HY000

  Message: Incorrect definition of table %s.%s: expected column '%s' at position %d, found '%s'.

  ER_SERVER_TABLE_CHECK_FAILED was added in 8.0.11.

- Error number: MY-013142; Symbol: ER_SERVER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE_V2; SQLSTATE: HY000

Message: The column count of %s.%s is wrong. Expected %d, found %d. Created with MySQL %d, now running %d. Please perform the MySQL upgrade procedure.

ER_SERVER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE_V2 was added in 8.0.11.

- Error number: MY-013143; Symbol: ER_SERVER_COL_COUNT_DOESNT_MATCH_CORRUPTED_V2; SQLSTATE: HY000

  Message: Column count of %s.%s is wrong. Expected %d, found %d. The table is probably corrupted

  ER_SERVER_COL_COUNT_DOESNT_MATCH_CORRUPTED_V2 was added in 8.0.11.

- Error number: MY-013144; Symbol: ER_SERVER_ACL_TABLE_ERROR; SQLSTATE: HY000

  Message:

  ER_SERVER_ACL_TABLE_ERROR was added in 8.0.11.

- Error number: MY-013145; Symbol: ER_SERVER_SLAVE_INIT_QUERY_FAILED; SQLSTATE: HY000

  Message: Slave SQL thread aborted. Can't execute init_slave query, MY-%06d - '%s'

  ER_SERVER_SLAVE_INIT_QUERY_FAILED was added in 8.0.11.

- Error number: MY-013146; Symbol: ER_SERVER_SLAVE_CONVERSION_FAILED; SQLSTATE: HY000

  Message: Column %d of table '%s.%s' cannot be converted from type '%s' to type '%s'

  ER_SERVER_SLAVE_CONVERSION_FAILED was added in 8.0.11.

- Error number: MY-013147; Symbol: ER_SERVER_SLAVE_IGNORED_TABLE; SQLSTATE: HY000

  Message: Slave SQL thread ignored the query because of replicate-*-table rules

  ER_SERVER_SLAVE_IGNORED_TABLE was added in 8.0.11.

- Error number: MY-013148; Symbol: ER_CANT_REPLICATE_ANONYMOUS_WITH_AUTO_POSITION; SQLSTATE: HY000

  Message: Cannot replicate anonymous transaction when AUTO_POSITION = 1, at file %s, position %lld.

- Error number: MY-013149; Symbol: ER_CANT_REPLICATE_ANONYMOUS_WITH_GTID_MODE_ON; SQLSTATE: HY000

  Message: Cannot replicate anonymous transaction when @@GLOBAL.GTID_MODE = ON, at file %s, position %lld.

- Error number: MY-013150; Symbol: ER_CANT_REPLICATE_GTID_WITH_GTID_MODE_OFF; SQLSTATE: HY000

  Message: Cannot replicate GTID-transaction when @@GLOBAL.GTID_MODE = OFF, at file %s, position %lld.

- Error number: MY-013151; Symbol: ER_SERVER_TEST_MESSAGE; SQLSTATE: HY000

  Message: Simulated error

  ER_SERVER_TEST_MESSAGE was added in 8.0.11.

- Error number: MY-013152; Symbol: ER_AUDIT_LOG_JSON_FILTER_PARSING_ERROR; SQLSTATE: HY000

  Message: %s

  ER_AUDIT_LOG_JSON_FILTER_PARSING_ERROR was added in 8.0.11.

- Error number: MY-013153; Symbol: ER_AUDIT_LOG_JSON_FILTERING_NOT_ENABLED; SQLSTATE: HY000

  Message: Audit Log filtering has not been installed.

  ER_AUDIT_LOG_JSON_FILTERING_NOT_ENABLED was added in 8.0.11.

- Error number: MY-013154; Symbol: ER_PLUGIN_FAILED_TO_OPEN_TABLES; SQLSTATE: HY000

  Message: Failed to open the %s filter tables.

  ER_PLUGIN_FAILED_TO_OPEN_TABLES was added in 8.0.11.

- Error number: MY-013155; Symbol: ER_PLUGIN_FAILED_TO_OPEN_TABLE; SQLSTATE: HY000

  Message: Failed to open '%s.%s' %s table.

  ER_PLUGIN_FAILED_TO_OPEN_TABLE was added in 8.0.11.

- Error number: MY-013156; Symbol: ER_AUDIT_LOG_JSON_FILTER_NAME_CANNOT_BE_EMPTY; SQLSTATE: HY000

  Message: Filter name cannot be empty.

  ER_AUDIT_LOG_JSON_FILTER_NAME_CANNOT_BE_EMPTY was added in 8.0.11.

- Error number: MY-013157; Symbol: ER_AUDIT_LOG_USER_NAME_INVALID_CHARACTER; SQLSTATE: HY000

  Message: Invalid character in the user name.

  ER_AUDIT_LOG_USER_NAME_INVALID_CHARACTER was added in 8.0.11.

- Error number: MY-013158; Symbol: ER_AUDIT_LOG_UDF_INSUFFICIENT_PRIVILEGE; SQLSTATE: HY000

  Message: Request ignored for '%s'@'%s'. SUPER or AUDIT_ADMIN needed to perform operation

  ER_AUDIT_LOG_UDF_INSUFFICIENT_PRIVILEGE was added in 8.0.11.

- Error number: MY-013159; Symbol: ER_AUDIT_LOG_NO_KEYRING_PLUGIN_INSTALLED; SQLSTATE: HY000

  Message: No keyring plugin installed.

  ER_AUDIT_LOG_NO_KEYRING_PLUGIN_INSTALLED was added in 8.0.11.

- Error number: MY-013160; Symbol: ER_AUDIT_LOG_HOST_NAME_INVALID_CHARACTER; SQLSTATE: HY000

  Message: Invalid character in the host name.

ER_AUDIT_LOG_HOST_NAME_INVALID_CHARACTER was added in 8.0.11.

- Error number: MY-013161; Symbol: ER_AUDIT_LOG_ENCRYPTION_PASSWORD_HAS_NOT_BEEN_SET; SQLSTATE: HY000

  Message: Audit log encryption password has not been set; it will be generated automatically. Use audit_log_encryption_password_get to obtain the password or audit_log_encryption_password_set to set a new one.

  ER_AUDIT_LOG_ENCRYPTION_PASSWORD_HAS_NOT_BEEN_SET was added in 8.0.11.

- Error number: MY-013162; Symbol: ER_AUDIT_LOG_COULD_NOT_CREATE_AES_KEY; SQLSTATE: HY000

  Message: Could not create AES key. OpenSSL's EVP_BytesToKey function failed.

  ER_AUDIT_LOG_COULD_NOT_CREATE_AES_KEY was added in 8.0.11.

- Error number: MY-013163; Symbol: ER_AUDIT_LOG_ENCRYPTION_PASSWORD_CANNOT_BE_FETCHED; SQLSTATE: HY000

  Message: Audit log encryption password cannot be fetched from the keyring. Password used so far is used for encryption.

  ER_AUDIT_LOG_ENCRYPTION_PASSWORD_CANNOT_BE_FETCHED was added in 8.0.11.

- Error number: MY-013164; Symbol: ER_COULD_NOT_REINITIALIZE_AUDIT_LOG_FILTERS; SQLSTATE: HY000

  Message: Could not reinitialize audit log filters.

  ER_COULD_NOT_REINITIALIZE_AUDIT_LOG_FILTERS was added in 8.0.11.

- Error number: MY-013165; Symbol: ER_AUDIT_LOG_JSON_USER_NAME_CANNOT_BE_EMPTY; SQLSTATE: HY000

  Message: User cannot be empty.

  ER_AUDIT_LOG_JSON_USER_NAME_CANNOT_BE_EMPTY was added in 8.0.11.

- Error number: MY-013166; Symbol: ER_AUDIT_LOG_USER_FIRST_CHARACTER_MUST_BE_ALPHANUMERIC; SQLSTATE: HY000

  Message: First character of the user name must be alphanumeric.

  ER_AUDIT_LOG_USER_FIRST_CHARACTER_MUST_BE_ALPHANUMERIC was added in 8.0.11.

- Error number: MY-013167; Symbol: ER_AUDIT_LOG_JSON_FILTER_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: Specified filter has not been found.

  ER_AUDIT_LOG_JSON_FILTER_DOES_NOT_EXIST was added in 8.0.11.

- Error number: MY-013168; Symbol: ER_IB_MSG_1271; SQLSTATE: HY000

  Message: Cannot upgrade server earlier than 5.7 to 8.0

ER_IB_MSG_1271 was added in 8.0.11.

- Error number: MY-013169; Symbol: ER_STARTING_INIT; SQLSTATE: HY000

  Message: %s (mysqld %s) initializing of server in progress as process %lu

  ER_STARTING_INIT was added in 8.0.11.

- Error number: MY-013170; Symbol: ER_ENDING_INIT; SQLSTATE: HY000

  Message: %s (mysqld %s) initializing of server has completed

  ER_ENDING_INIT was added in 8.0.11.

- Error number: MY-013171; Symbol: ER_IB_MSG_1272; SQLSTATE: HY000

  Message: Cannot boot server version %lu on data directory built by version %llu. Downgrade is not supported

  ER_IB_MSG_1272 was added in 8.0.11.

- Error number: MY-013172; Symbol: ER_SERVER_SHUTDOWN_INFO; SQLSTATE: HY000

  Message: Received SHUTDOWN from user %s. Shutting down mysqld (Version: %s).

  ER_SERVER_SHUTDOWN_INFO was added in 8.0.12.

- Error number: MY-013173; Symbol: ER_GRP_RPL_PLUGIN_ABORT; SQLSTATE: HY000

  Message: The plugin encountered a critical error and will abort: %s

  ER_GRP_RPL_PLUGIN_ABORT was added in 8.0.12.

- Error number: MY-013177; Symbol: ER_AUDIT_LOG_TABLE_DEFINITION_NOT_UPDATED; SQLSTATE: HY000

  Message: '%s.%s' table definition has not been upgraded; Please perform the MySQL upgrade procedure.

  ER_AUDIT_LOG_TABLE_DEFINITION_NOT_UPDATED was added in 8.0.12.

- Error number: MY-013178; Symbol: ER_DD_INITIALIZE_SQL_ERROR; SQLSTATE: HY000

  Message: Execution of server-side SQL statement '%s' failed with error code = %d, error message = '%s'.

  ER_DD_INITIALIZE_SQL_ERROR was added in 8.0.12.

- Error number: MY-013179; Symbol: ER_NO_PATH_FOR_SHARED_LIBRARY; SQLSTATE: HY000

  Message: No paths allowed for shared library.

  ER_NO_PATH_FOR_SHARED_LIBRARY was added in 8.0.13.

- Error number: MY-013180; Symbol: ER_UDF_ALREADY_EXISTS; SQLSTATE: HY000

  Message: Function '%s' already exists.

  ER_UDF_ALREADY_EXISTS was added in 8.0.13.

- Error number: MY-013181; Symbol: ER_SET_EVENT_FAILED; SQLSTATE: HY000

  Message: Got Error: %ld from SetEvent.

  ER_SET_EVENT_FAILED was added in 8.0.13.

- Error number: MY-013182; Symbol: ER_FAILED_TO_ALLOCATE_SSL_BIO; SQLSTATE: HY000

  Message: Error allocating SSL BIO.

  ER_FAILED_TO_ALLOCATE_SSL_BIO was added in 8.0.13.

- Error number: MY-013183; Symbol: ER_IB_MSG_1273; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1273 was added in 8.0.13.

- Error number: MY-013184; Symbol: ER_PID_FILEPATH_LOCATIONS_INACCESSIBLE; SQLSTATE: HY000

  Message: One or several locations were inaccessible while checking PID filepath.

  ER_PID_FILEPATH_LOCATIONS_INACCESSIBLE was added in 8.0.13.

- Error number: MY-013185; Symbol: ER_UNKNOWN_VARIABLE_IN_PERSISTED_CONFIG_FILE; SQLSTATE: HY000

  Message: Currently unknown variable '%s' was read from the persisted config file.

  ER_UNKNOWN_VARIABLE_IN_PERSISTED_CONFIG_FILE was added in 8.0.13.

- Error number: MY-013186; Symbol: ER_FAILED_TO_HANDLE_DEFAULTS_FILE; SQLSTATE: HY000

  Message: Fatal error in defaults handling. Program aborted!

  ER_FAILED_TO_HANDLE_DEFAULTS_FILE was added in 8.0.13.

- Error number: MY-013187; Symbol: ER_DUPLICATE_SYS_VAR; SQLSTATE: HY000

  Message: Duplicate variable name '%s'.

  ER_DUPLICATE_SYS_VAR was added in 8.0.13.

- Error number: MY-013188; Symbol: ER_FAILED_TO_INIT_SYS_VAR; SQLSTATE: HY000

  Message: Failed to initialize system variables.

  ER_FAILED_TO_INIT_SYS_VAR was added in 8.0.13.

- Error number: MY-013189; Symbol: ER_SYS_VAR_NOT_FOUND; SQLSTATE: HY000

  Message: Variable name '%s' not found.

  ER_SYS_VAR_NOT_FOUND was added in 8.0.13.

- Error number: MY-013190; Symbol: ER_IB_MSG_1274; SQLSTATE: HY000

  Message: Some (%d) threads are still active

ER_IB_MSG_1274 was added in 8.0.13.

- Error number: MY-013191; Symbol: ER_IB_MSG_1275; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1275 was added in 8.0.13.

- Error number: MY-013192; Symbol: ER_TARGET_TS_UNENCRYPTED; SQLSTATE: HY000

  Message: Source tablespace is encrypted but target tablespace is not.

  ER_TARGET_TS_UNENCRYPTED was added in 8.0.13, removed after 8.0.15.

- Error number: MY-013193; Symbol: ER_IB_MSG_1276; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1276 was added in 8.0.13, removed after 8.0.13.

- Error number: MY-013193; Symbol: ER_IB_MSG_WAIT_FOR_ENCRYPT_THREAD; SQLSTATE: HY000

  Message: Waiting for tablespace_alter_encrypt_thread to exit

  ER_IB_MSG_WAIT_FOR_ENCRYPT_THREAD was added in 8.0.14.

- Error number: MY-013194; Symbol: ER_IB_MSG_1277; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1277 was added in 8.0.13.

- Error number: MY-013195; Symbol: ER_IB_MSG_1278; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1278 was added in 8.0.13, removed after 8.0.13.

- Error number: MY-013195; Symbol: ER_IB_MSG_NO_ENCRYPT_PROGRESS_FOUND; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_NO_ENCRYPT_PROGRESS_FOUND was added in 8.0.14.

- Error number: MY-013196; Symbol: ER_IB_MSG_1279; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1279 was added in 8.0.13, removed after 8.0.13.

- Error number: MY-013196; Symbol: ER_IB_MSG_RESUME_OP_FOR_SPACE; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_RESUME_OP_FOR_SPACE was added in 8.0.14.

- Error number: MY-013197; Symbol: ER_IB_MSG_1280; SQLSTATE: HY000

Message: %s

ER_IB_MSG_1280 was added in 8.0.13.

- Error number: MY-013198; Symbol: ER_IB_MSG_1281; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1281 was added in 8.0.13.

- Error number: MY-013199; Symbol: ER_IB_MSG_1282; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1282 was added in 8.0.13.

- Error number: MY-013200; Symbol: ER_IB_MSG_1283; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1283 was added in 8.0.13.

- Error number: MY-013201; Symbol: ER_IB_MSG_1284; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_1284 was added in 8.0.13.

- Error number: MY-013202; Symbol: ER_CANT_SET_ERROR_SUPPRESSION_LIST_FROM_COMMAND_LINE; SQLSTATE: HY000

  Message: %s: Could not add suppression rule for code "%s". Rule-set may be full, or code may not correspond to an error-log message.

  ER_CANT_SET_ERROR_SUPPRESSION_LIST_FROM_COMMAND_LINE was added in 8.0.13.

- Error number: MY-013203; Symbol: ER_INVALID_VALUE_OF_BIND_ADDRESSES; SQLSTATE: HY000

  Message: Invalid value for command line option bind-addresses: '%s'

  ER_INVALID_VALUE_OF_BIND_ADDRESSES was added in 8.0.13.

- Error number: MY-013204; Symbol: ER_RELAY_LOG_SPACE_LIMIT_DISABLED; SQLSTATE: HY000

  Message: Ignoring the @@global.relay_log_space_limit option because @@global.relay_log_purge is disabled.

  ER_RELAY_LOG_SPACE_LIMIT_DISABLED was added in 8.0.13.

- Error number: MY-013205; Symbol: ER_GRP_RPL_ERROR_GTID_SET_EXTRACTION; SQLSTATE: HY000

  Message: Error when extracting GTID execution information: %s

  ER_GRP_RPL_ERROR_GTID_SET_EXTRACTION was added in 8.0.13.

- Error number: MY-013206; Symbol: ER_GRP_RPL_MISSING_GRP_RPL_ACTION_COORDINATOR; SQLSTATE: HY000

Message: Message received without a proper group coordinator module.

ER_GRP_RPL_MISSING_GRP_RPL_ACTION_COORDINATOR was added in 8.0.13.

- Error number: MY-013207; Symbol: ER_GRP_RPL_JOIN_WHEN_GROUP_ACTION_RUNNING; SQLSTATE: HY000

  Message: A member cannot join the group while a group configuration operation is running.

  ER_GRP_RPL_JOIN_WHEN_GROUP_ACTION_RUNNING was added in 8.0.13.

- Error number: MY-013208; Symbol: ER_GRP_RPL_JOINER_EXIT_WHEN_GROUP_ACTION_RUNNING; SQLSTATE: HY000

  Message: A member is joining the group while a group configuration operation is occurring. The member will now leave the group

  ER_GRP_RPL_JOINER_EXIT_WHEN_GROUP_ACTION_RUNNING was added in 8.0.13.

- Error number: MY-013209; Symbol: ER_GRP_RPL_CHANNEL_THREAD_WHEN_GROUP_ACTION_RUNNING; SQLSTATE: HY000

  Message: Can't start slave %s when group replication is running a group configuration operation.

  ER_GRP_RPL_CHANNEL_THREAD_WHEN_GROUP_ACTION_RUNNING was added in 8.0.13.

- Error number: MY-013210; Symbol: ER_GRP_RPL_APPOINTED_PRIMARY_NOT_PRESENT; SQLSTATE: HY000

  Message: A primary election was invoked but the requested primary member is not in the group. Request ignored.

  ER_GRP_RPL_APPOINTED_PRIMARY_NOT_PRESENT was added in 8.0.13.

- Error number: MY-013211; Symbol: ER_GRP_RPL_ERROR_ON_MESSAGE_SENDING; SQLSTATE: HY000

  Message: Error while sending message. Context: %s

  ER_GRP_RPL_ERROR_ON_MESSAGE_SENDING was added in 8.0.13.

- Error number: MY-013212; Symbol: ER_GRP_RPL_CONFIGURATION_ACTION_ERROR; SQLSTATE: HY000

  Message: Error while executing a group configuration operation: %s

  ER_GRP_RPL_CONFIGURATION_ACTION_ERROR was added in 8.0.13.

- Error number: MY-013213; Symbol: ER_GRP_RPL_CONFIGURATION_ACTION_LOCAL_TERMINATION; SQLSTATE: HY000

  Message: Configuration operation '%s' terminated. %s

  ER_GRP_RPL_CONFIGURATION_ACTION_LOCAL_TERMINATION was added in 8.0.13.

- Error number: MY-013214; Symbol: ER_GRP_RPL_CONFIGURATION_ACTION_START; SQLSTATE: HY000

Message: Starting group operation local execution: %s

ER_GRP_RPL_CONFIGURATION_ACTION_START was added in 8.0.13.

- Error number: MY-013215; Symbol: ER_GRP_RPL_CONFIGURATION_ACTION_END; SQLSTATE: HY000

  Message: Termination of group operation local execution: %s

  ER_GRP_RPL_CONFIGURATION_ACTION_END was added in 8.0.13.

- Error number: MY-013216; Symbol: ER_GRP_RPL_CONFIGURATION_ACTION_KILLED_ERROR; SQLSTATE: HY000

  Message: A configuration change was killed in this member. The member will now leave the group as its configuration may have diverged.

  ER_GRP_RPL_CONFIGURATION_ACTION_KILLED_ERROR was added in 8.0.13.

- Error number: MY-013217; Symbol: ER_GRP_RPL_PRIMARY_ELECTION_PROCESS_ERROR; SQLSTATE: HY000

  Message: There was an issue on the primary election process: %s The member will now leave the group.

  ER_GRP_RPL_PRIMARY_ELECTION_PROCESS_ERROR was added in 8.0.13.

- Error number: MY-013218; Symbol: ER_GRP_RPL_PRIMARY_ELECTION_STOP_ERROR; SQLSTATE: HY000

  Message: There was an issue when stopping a previous election process: %s

  ER_GRP_RPL_PRIMARY_ELECTION_STOP_ERROR was added in 8.0.13.

- Error number: MY-013219; Symbol: ER_GRP_RPL_NO_STAGE_SERVICE; SQLSTATE: HY000

  Message: It was not possible to initialize stage logging for this task. The operation will still run without stage tracking.

  ER_GRP_RPL_NO_STAGE_SERVICE was added in 8.0.13.

- Error number: MY-013220; Symbol: ER_GRP_RPL_UDF_REGISTER_ERROR; SQLSTATE: HY000

  Message: Could not execute the installation of Group Replication UDF function: %s. Check if the function is already present, if so, try to remove it

  ER_GRP_RPL_UDF_REGISTER_ERROR was added in 8.0.13.

- Error number: MY-013221; Symbol: ER_GRP_RPL_UDF_UNREGISTER_ERROR; SQLSTATE: HY000

  Message: Could not uninstall Group Replication UDF functions. Try to remove them manually if present.

  ER_GRP_RPL_UDF_UNREGISTER_ERROR was added in 8.0.13.

- Error number: MY-013222; Symbol: ER_GRP_RPL_UDF_REGISTER_SERVICE_ERROR; SQLSTATE: HY000

Message: Could not execute the installation of Group Replication UDF functions. Check for other errors in the log and try to reinstall the plugin

ER_GRP_RPL_UDF_REGISTER_SERVICE_ERROR was added in 8.0.13.

- Error number: MY-013223; Symbol: ER_GRP_RPL_SERVER_UDF_ERROR; SQLSTATE: HY000

  Message: The function '%s' failed. %s

  ER_GRP_RPL_SERVER_UDF_ERROR was added in 8.0.17.

- Error number: MY-013227; Symbol: ER_SERVER_WRONG_VALUE_FOR_VAR; SQLSTATE: HY000

  Message: Variable '%s' can't be set to the value of '%s'

  ER_SERVER_WRONG_VALUE_FOR_VAR was added in 8.0.13.

- Error number: MY-013228; Symbol: ER_COULD_NOT_CREATE_WINDOWS_REGISTRY_KEY; SQLSTATE: HY000

  Message: %s was unable to create a new Windows registry key %s for %s; continuing to use the previous ident.

  ER_COULD_NOT_CREATE_WINDOWS_REGISTRY_KEY was added in 8.0.13.

- Error number: MY-013229; Symbol: ER_SERVER_GTID_UNSAFE_CREATE_DROP_TEMP_TABLE_IN_TRX_IN_SBR; SQLSTATE: HY000

  Message: Statement violates GTID consistency: CREATE TEMPORARY TABLE and DROP TEMPORARY TABLE are not allowed inside a transaction or inside a procedure in a transactional context when @@session.binlog_format=STATEMENT.

  ER_SERVER_GTID_UNSAFE_CREATE_DROP_TEMP_TABLE_IN_TRX_IN_SBR was added in 8.0.13.

- Error number: MY-013233; Symbol: ER_XPLUGIN_FAILED_TO_SWITCH_SECURITY_CTX; SQLSTATE: HY000

  Message: Unable to switch security context to user: %s

  ER_XPLUGIN_FAILED_TO_SWITCH_SECURITY_CTX was added in 8.0.13.

- Error number: MY-013234; Symbol: ER_RPL_GTID_UNSAFE_ALTER_ADD_COL_WITH_DEFAULT_EXPRESSION; SQLSTATE: HY000

  Message: Statement violates GTID consistency: ALTER TABLE ... ADD COLUMN .. with expression as DEFAULT.

  ER_RPL_GTID_UNSAFE_ALTER_ADD_COL_WITH_DEFAULT_EXPRESSION was added in 8.0.13.

- Error number: MY-013235; Symbol: ER_UPGRADE_PARSE_ERROR; SQLSTATE: HY000

  Message: Error in parsing %s '%s'.'%s' during upgrade. %s

  ER_UPGRADE_PARSE_ERROR was added in 8.0.13.

- Error number: MY-013236; Symbol: ER_DATA_DIRECTORY_UNUSABLE; SQLSTATE: HY000

Message: The designated data directory %s is unusable. You can remove all files that the server added to it.

ER_DATA_DIRECTORY_UNUSABLE was added in 8.0.13.

- Error number: MY-013237; Symbol: ER_LDAP_AUTH_USER_GROUP_SEARCH_ROOT_BIND; SQLSTATE: HY000

  Message: Group search rebinding via root DN: %s

  ER_LDAP_AUTH_USER_GROUP_SEARCH_ROOT_BIND was added in 8.0.13.

- Error number: MY-013238; Symbol: ER_PLUGIN_INSTALL_ERROR; SQLSTATE: HY000

  Message: Error installing plugin '%s': %s

  ER_PLUGIN_INSTALL_ERROR was added in 8.0.13.

- Error number: MY-013239; Symbol: ER_PLUGIN_UNINSTALL_ERROR; SQLSTATE: HY000

  Message: Error uninstalling plugin '%s': %s

  ER_PLUGIN_UNINSTALL_ERROR was added in 8.0.13.

- Error number: MY-013240; Symbol: ER_SHARED_TABLESPACE_USED_BY_PARTITIONED_TABLE; SQLSTATE: HY000

  Message: Partitioned table '%s' is not allowed to use shared tablespace '%s'. Please move all partitions to file-per-table tablespaces before upgrade.

  ER_SHARED_TABLESPACE_USED_BY_PARTITIONED_TABLE was added in 8.0.13.

- Error number: MY-013241; Symbol: ER_UNKNOWN_TABLESPACE_TYPE; SQLSTATE: HY000

  Message: Cannot determine the type of the tablespace named '%s'.

  ER_UNKNOWN_TABLESPACE_TYPE was added in 8.0.13.

- Error number: MY-013242; Symbol: ER_WARN_DEPRECATED_UTF8_ALIAS_OPTION; SQLSTATE: HY000

  Message: %s: 'utf8' is currently an alias for the character set UTF8MB3, but will be an alias for UTF8MB4 in a future release. Please consider using UTF8MB4 in order to be unambiguous.

  ER_WARN_DEPRECATED_UTF8_ALIAS_OPTION was added in 8.0.13.

- Error number: MY-013243; Symbol: ER_WARN_DEPRECATED_UTF8MB3_CHARSET_OPTION; SQLSTATE: HY000

  Message: %s: The character set UTF8MB3 is deprecated and will be removed in a future release. Please consider using UTF8MB4 instead.

  ER_WARN_DEPRECATED_UTF8MB3_CHARSET_OPTION was added in 8.0.13.

- Error number: MY-013244; Symbol: ER_WARN_DEPRECATED_UTF8MB3_COLLATION_OPTION; SQLSTATE: HY000

Message: %s: '%s' is a collation of the deprecated character set UTF8MB3. Please consider using UTF8MB4 with an appropriate collation instead.

ER_WARN_DEPRECATED_UTF8MB3_COLLATION_OPTION was added in 8.0.13.

- Error number: MY-013245; Symbol: ER_SSL_MEMORY_INSTRUMENTATION_INIT_FAILED; SQLSTATE: HY000

  Message: The SSL library function %s failed. This is typically caused by the SSL library already being used. As a result the SSL memory allocation will not be instrumented.

  ER_SSL_MEMORY_INSTRUMENTATION_INIT_FAILED was added in 8.0.13.

- Error number: MY-013246; Symbol: ER_IB_MSG_MADV_DONTDUMP_UNSUPPORTED; SQLSTATE: HY000

  Message: Disabling @@core_file because @@innodb_buffer_pool_in_core_file is disabled, yet MADV_DONTDUMP is not supported on this platform

  ER_IB_MSG_MADV_DONTDUMP_UNSUPPORTED was added in 8.0.14.

- Error number: MY-013247; Symbol: ER_IB_MSG_MADVISE_FAILED; SQLSTATE: HY000

  Message: Disabling @@core_file because @@innodb_buffer_pool_in_core_file is disabled, yet madvise(%p,%zu,%s) failed with %s

  ER_IB_MSG_MADVISE_FAILED was added in 8.0.14.

- Error number: MY-013249; Symbol: ER_WARN_REMOVED_SQL_MODE; SQLSTATE: HY000

  Message: sql_mode=0x%08x has been removed and will be ignored

  ER_WARN_REMOVED_SQL_MODE was added in 8.0.14.

- Error number: MY-013250; Symbol: ER_IB_MSG_FAILED_TO_ALLOCATE_WAIT; SQLSTATE: HY000

  Message: Failed to allocate memory for a pool of size %zu bytes. Will wait for %zu seconds for a thread to free a resource.

  ER_IB_MSG_FAILED_TO_ALLOCATE_WAIT was added in 8.0.14.

- Error number: MY-013251; Symbol: ER_IB_MSG_NUM_POOLS; SQLSTATE: HY000

  Message: Number of pools: %zu

  ER_IB_MSG_NUM_POOLS was added in 8.0.14.

- Error number: MY-013252; Symbol: ER_IB_MSG_USING_UNDO_SPACE; SQLSTATE: HY000

  Message: Using undo tablespace '%s'.

  ER_IB_MSG_USING_UNDO_SPACE was added in 8.0.14.

- Error number: MY-013253; Symbol: ER_IB_MSG_FAIL_TO_SAVE_SPACE_STATE; SQLSTATE: HY000

  Message: %s Unable to save the current state of tablespace '%s' to the data dictionary

ER_IB_MSG_FAIL_TO_SAVE_SPACE_STATE was added in 8.0.14.

- Error number: MY-013254; Symbol: ER_IB_MSG_MAX_UNDO_SPACES_REACHED; SQLSTATE: HY000

  Message: Cannot create undo tablespace %s at %s because %d undo tablespaces already exist.

  ER_IB_MSG_MAX_UNDO_SPACES_REACHED was added in 8.0.14.

- Error number: MY-013255; Symbol: ER_IB_MSG_ERROR_OPENING_NEW_UNDO_SPACE; SQLSTATE: HY000

  Message: Error %d opening newly created undo tablespace %s.

  ER_IB_MSG_ERROR_OPENING_NEW_UNDO_SPACE was added in 8.0.14.

- Error number: MY-013256; Symbol: ER_IB_MSG_FAILED_SDI_Z_BUF_ERROR; SQLSTATE: HY000

  Message: SDI Compression failed, Z_BUF_ERROR

  ER_IB_MSG_FAILED_SDI_Z_BUF_ERROR was added in 8.0.14.

- Error number: MY-013257; Symbol: ER_IB_MSG_FAILED_SDI_Z_MEM_ERROR; SQLSTATE: HY000

  Message: SDI Compression failed, Z_MEM_ERROR

  ER_IB_MSG_FAILED_SDI_Z_MEM_ERROR was added in 8.0.14.

- Error number: MY-013258; Symbol: ER_IB_MSG_SDI_Z_STREAM_ERROR; SQLSTATE: HY000

  Message: SDI Compression failed, Z_STREAM_ERROR

  ER_IB_MSG_SDI_Z_STREAM_ERROR was added in 8.0.14.

- Error number: MY-013259; Symbol: ER_IB_MSG_SDI_Z_UNKNOWN_ERROR; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_SDI_Z_UNKNOWN_ERROR was added in 8.0.14.

- Error number: MY-013260; Symbol: ER_IB_MSG_FOUND_WRONG_UNDO_SPACE; SQLSTATE: HY000

  Message: Expected to find undo tablespace '%s' for Space ID=%lu, but found '%s' instead! Did you change innodb_undo_directory?

  ER_IB_MSG_FOUND_WRONG_UNDO_SPACE was added in 8.0.14.

- Error number: MY-013261; Symbol: ER_IB_MSG_NOT_END_WITH_IBU; SQLSTATE: HY000

  Message: Cannot use %s as an undo tablespace because it does not end with '.ibu'.

  ER_IB_MSG_NOT_END_WITH_IBU was added in 8.0.14.

- Error number: MY-013262; Symbol: ER_IB_MSG_UNDO_TRUNC_EMPTY_FILE; SQLSTATE: HY000

  Message: ib_undo_trunc_empty_file

  ER_IB_MSG_UNDO_TRUNC_EMPTY_FILE was added in 8.0.14.

- Error number: MY-013263; Symbol: ER_IB_MSG_UNDO_TRUNC_BEFOR_DD_UPDATE; SQLSTATE: HY000

  Message: ib_undo_trunc_before_dd_update

  ER_IB_MSG_UNDO_TRUNC_BEFOR_DD_UPDATE was added in 8.0.14.

- Error number: MY-013264; Symbol: ER_IB_MSG_UNDO_TRUNC_BEFORE_UNDO_LOGGING; SQLSTATE: HY000

  Message: ib_undo_trunc_before_done_logging

  ER_IB_MSG_UNDO_TRUNC_BEFORE_UNDO_LOGGING was added in 8.0.14.

- Error number: MY-013265; Symbol: ER_IB_MSG_UNDO_TRUNK_BEFORE_RSEG; SQLSTATE: HY000

  Message: ib_undo_trunc_before_rsegs

  ER_IB_MSG_UNDO_TRUNK_BEFORE_RSEG was added in 8.0.14.

- Error number: MY-013266; Symbol: ER_IB_MSG_FAILED_TO_FINISH_TRUNCATE; SQLSTATE: HY000

  Message: %s Failed to finish truncating Undo Tablespace '%s'

  ER_IB_MSG_FAILED_TO_FINISH_TRUNCATE was added in 8.0.14.

- Error number: MY-013267; Symbol: ER_IB_MSG_DEPRECATED_INNODB_UNDO_TABLESPACES; SQLSTATE: HY000

  Message: The setting INNODB_UNDO_TABLESPACES is deprecated and is no longer used. InnoDB always creates 2 undo tablespaces to start with. If you need more, please use CREATE UNDO TABLESPACE.

  ER_IB_MSG_DEPRECATED_INNODB_UNDO_TABLESPACES was added in 8.0.14.

- Error number: MY-013268; Symbol: ER_IB_MSG_DIR_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: The directory '%s' does not exist.

  ER_IB_MSG_DIR_DOES_NOT_EXIST was added in 8.0.14.

- Error number: MY-013269; Symbol: ER_IB_MSG_LOCK_FREE_HASH_USAGE_STATS; SQLSTATE: HY000

  Message: %s

  ER_IB_MSG_LOCK_FREE_HASH_USAGE_STATS was added in 8.0.14.

- Error number: MY-013270; Symbol: ER_CLONE_REMOTE_ERROR; SQLSTATE: HY000

  Message: Clone Remote Error: %s.

  ER_CLONE_REMOTE_ERROR was added in 8.0.14, removed after 8.0.16.

- Error number: MY-013270; Symbol: ER_CLONE_DONOR_TRACE; SQLSTATE: HY000

  Message: Clone donor reported : %s.

ER_CLONE_DONOR_TRACE was added in 8.0.17.

- Error number: MY-013271; Symbol: ER_CLONE_PROTOCOL_ERROR; SQLSTATE: HY000

  Message: Clone Protocol Error: %s.

  ER_CLONE_PROTOCOL_ERROR was added in 8.0.14, removed after 8.0.16.

- Error number: MY-013271; Symbol: ER_CLONE_PROTOCOL_TRACE; SQLSTATE: HY000

  Message: Clone received unexpected response from donor : %s.

  ER_CLONE_PROTOCOL_TRACE was added in 8.0.17.

- Error number: MY-013272; Symbol: ER_CLONE_INFO_CLIENT; SQLSTATE: HY000

  Message: Client: %s.

  ER_CLONE_INFO_CLIENT was added in 8.0.14, removed after 8.0.16.

- Error number: MY-013272; Symbol: ER_CLONE_CLIENT_TRACE; SQLSTATE: HY000

  Message: Client: %s.

  ER_CLONE_CLIENT_TRACE was added in 8.0.17.

- Error number: MY-013273; Symbol: ER_CLONE_INFO_SERVER; SQLSTATE: HY000

  Message: Server: %s.

  ER_CLONE_INFO_SERVER was added in 8.0.14, removed after 8.0.16.

- Error number: MY-013273; Symbol: ER_CLONE_SERVER_TRACE; SQLSTATE: HY000

  Message: Server: %s.

  ER_CLONE_SERVER_TRACE was added in 8.0.17.

- Error number: MY-013274; Symbol: ER_THREAD_POOL_PFS_TABLES_INIT_FAILED; SQLSTATE: HY000

  Message: Failed to initialize the performance schema tables service.

  ER_THREAD_POOL_PFS_TABLES_INIT_FAILED was added in 8.0.14.

- Error number: MY-013275; Symbol: ER_THREAD_POOL_PFS_TABLES_ADD_FAILED; SQLSTATE: HY000

  Message: Failed to add thread pool performance schema tables.

  ER_THREAD_POOL_PFS_TABLES_ADD_FAILED was added in 8.0.14.

- Error number: MY-013276; Symbol: ER_CANT_SET_DATA_DIR; SQLSTATE: HY000

  Message: Failed to set datadir to \'%s\' (OS errno: %d - %s)

  ER_CANT_SET_DATA_DIR was added in 8.0.14.

- Error number: MY-013277; Symbol: ER_INNODB_INVALID_INNODB_UNDO_DIRECTORY_LOCATION; SQLSTATE: HY000

  Message: The innodb_undo_directory is not allowed to be an ancestor of the datadir.

  ER_INNODB_INVALID_INNODB_UNDO_DIRECTORY_LOCATION was added in 8.0.14.

- Error number: MY-013278; Symbol: ER_SERVER_RPL_ENCRYPTION_FAILED_TO_FETCH_KEY; SQLSTATE: HY000

  Message: Failed to fetch key from keyring, please check if keyring plugin is loaded.

  ER_SERVER_RPL_ENCRYPTION_FAILED_TO_FETCH_KEY was added in 8.0.14.

- Error number: MY-013279; Symbol: ER_SERVER_RPL_ENCRYPTION_KEY_NOT_FOUND; SQLSTATE: HY000

  Message: Can't find key from keyring, please check in the server log if a keyring plugin is loaded and initialized successfully.

  ER_SERVER_RPL_ENCRYPTION_KEY_NOT_FOUND was added in 8.0.14.

- Error number: MY-013280; Symbol: ER_SERVER_RPL_ENCRYPTION_KEYRING_INVALID_KEY; SQLSTATE: HY000

  Message: Fetched an invalid key from keyring.

  ER_SERVER_RPL_ENCRYPTION_KEYRING_INVALID_KEY was added in 8.0.14.

- Error number: MY-013281; Symbol: ER_SERVER_RPL_ENCRYPTION_HEADER_ERROR; SQLSTATE: HY000

  Message: Error reading a replication log encryption header: %s.

  ER_SERVER_RPL_ENCRYPTION_HEADER_ERROR was added in 8.0.14.

- Error number: MY-013282; Symbol: ER_SERVER_RPL_ENCRYPTION_FAILED_TO_ROTATE_LOGS; SQLSTATE: HY000

  Message: Failed to rotate some logs after changing binlog encryption settings. Please fix the problem and rotate the logs manually.

  ER_SERVER_RPL_ENCRYPTION_FAILED_TO_ROTATE_LOGS was added in 8.0.14.

- Error number: MY-013283; Symbol: ER_SERVER_RPL_ENCRYPTION_KEY_EXISTS_UNEXPECTED; SQLSTATE: HY000

  Message: Key %s exists unexpected.

  ER_SERVER_RPL_ENCRYPTION_KEY_EXISTS_UNEXPECTED was added in 8.0.14.

- Error number: MY-013284; Symbol: ER_SERVER_RPL_ENCRYPTION_FAILED_TO_GENERATE_KEY; SQLSTATE: HY000

  Message: Failed to generate key, please check if keyring plugin is loaded.

  ER_SERVER_RPL_ENCRYPTION_FAILED_TO_GENERATE_KEY was added in 8.0.14.

- Error number: MY-013285; Symbol: ER_SERVER_RPL_ENCRYPTION_FAILED_TO_STORE_KEY; SQLSTATE: HY000

  Message: Failed to store key, please check if keyring plugin is loaded.

  ER_SERVER_RPL_ENCRYPTION_FAILED_TO_STORE_KEY was added in 8.0.14.

- Error number: MY-013286; Symbol: ER_SERVER_RPL_ENCRYPTION_FAILED_TO_REMOVE_KEY; SQLSTATE: HY000

  Message: Failed to remove key, please check if keyring plugin is loaded.

  ER_SERVER_RPL_ENCRYPTION_FAILED_TO_REMOVE_KEY was added in 8.0.14.

- Error number: MY-013287; Symbol: ER_SERVER_RPL_ENCRYPTION_MASTER_KEY_RECOVERY_FAILED; SQLSTATE: HY000

  Message: Unable to recover binlog encryption master key, please check if keyring plugin is loaded.

  ER_SERVER_RPL_ENCRYPTION_MASTER_KEY_RECOVERY_FAILED was added in 8.0.14.

- Error number: MY-013288; Symbol: ER_SERVER_RPL_ENCRYPTION_UNABLE_TO_INITIALIZE; SQLSTATE: HY000

  Message: Failed to initialize binlog encryption, please check if keyring plugin is loaded.

  ER_SERVER_RPL_ENCRYPTION_UNABLE_TO_INITIALIZE was added in 8.0.14.

- Error number: MY-013289; Symbol: ER_SERVER_RPL_ENCRYPTION_UNABLE_TO_ROTATE_MASTER_KEY_AT_STARTUP; SQLSTATE: HY000

  Message: Failed to rotate binlog encryption master key at startup, please check if keyring plugin is loaded.

  ER_SERVER_RPL_ENCRYPTION_UNABLE_TO_ROTATE_MASTER_KEY_AT_STARTUP was added in 8.0.14.

- Error number: MY-013290; Symbol: ER_SERVER_RPL_ENCRYPTION_IGNORE_ROTATE_MASTER_KEY_AT_STARTUP; SQLSTATE: HY000

  Message: Ignoring binlog_rotate_encryption_master_key_at_startup because binlog_encryption option is disabled.

  ER_SERVER_RPL_ENCRYPTION_IGNORE_ROTATE_MASTER_KEY_AT_STARTUP was added in 8.0.14.

- Error number: MY-013291; Symbol: ER_INVALID_ADMIN_ADDRESS; SQLSTATE: HY000

  Message: Invalid value for command line option admin-address: '%s'

  ER_INVALID_ADMIN_ADDRESS was added in 8.0.14.

- Error number: MY-013292; Symbol: ER_SERVER_STARTUP_ADMIN_INTERFACE; SQLSTATE: HY000

  Message: Admin interface ready for connections, address: '%s' port: %d

  ER_SERVER_STARTUP_ADMIN_INTERFACE was added in 8.0.14.

- Error number: MY-013293; Symbol: ER_CANT_CREATE_ADMIN_THREAD; SQLSTATE: HY000

Message: Can't create thread to handle admin connections (errno= %d)

ER_CANT_CREATE_ADMIN_THREAD was added in 8.0.14.

- Error number: MY-013294; Symbol: ER_WARNING_RETAIN_CURRENT_PASSWORD_CLAUSE_VOID; SQLSTATE: HY000

  Message: RETAIN CURRENT PASSWORD ignored for user '%s'@'%s' as its authentication plugin %s does not support multiple passwords.

  ER_WARNING_RETAIN_CURRENT_PASSWORD_CLAUSE_VOID was added in 8.0.14.

- Error number: MY-013295; Symbol: ER_WARNING_DISCARD_OLD_PASSWORD_CLAUSE_VOID; SQLSTATE: HY000

  Message: DISCARD OLD PASSWORD ignored for user '%s'@'%s' as its authentication plugin %s does not support multiple passwords.

  ER_WARNING_DISCARD_OLD_PASSWORD_CLAUSE_VOID was added in 8.0.14.

- Error number: MY-013299; Symbol: ER_WARNING_AUTHCACHE_INVALID_USER_ATTRIBUTES; SQLSTATE: HY000

  Message: Can not read and process value of User_attributes column from mysql.user table for user: '%s@%s'; Ignoring user.

  ER_WARNING_AUTHCACHE_INVALID_USER_ATTRIBUTES was added in 8.0.14.

- Error number: MY-013300; Symbol: ER_MYSQL_NATIVE_PASSWORD_SECOND_PASSWORD_USED_INFORMATION; SQLSTATE: HY000

  Message: Second password was used for login by user: '%s'@'%s'.

  ER_MYSQL_NATIVE_PASSWORD_SECOND_PASSWORD_USED_INFORMATION was added in 8.0.14.

- Error number: MY-013301; Symbol: ER_SHA256_PASSWORD_SECOND_PASSWORD_USED_INFORMATION; SQLSTATE: HY000

  Message: Second password was used for login by user: '%s'@'%s'.

  ER_SHA256_PASSWORD_SECOND_PASSWORD_USED_INFORMATION was added in 8.0.14.

- Error number: MY-013302; Symbol: ER_CACHING_SHA2_PASSWORD_SECOND_PASSWORD_USED_INFORMATION; SQLSTATE: HY000

  Message: Second password was used for login by user: '%s'@'%s'.

  ER_CACHING_SHA2_PASSWORD_SECOND_PASSWORD_USED_INFORMATION was added in 8.0.14.

- Error number: MY-013303; Symbol: ER_GRP_RPL_SEND_TRX_PREPARED_MESSAGE_FAILED; SQLSTATE: HY000

  Message: Error sending transaction '%d:%lld' prepared message from session '%u'.

  ER_GRP_RPL_SEND_TRX_PREPARED_MESSAGE_FAILED was added in 8.0.14.

- Error number: MY-013304; Symbol: ER_GRP_RPL_RELEASE_COMMIT_AFTER_GROUP_PREPARE_FAILED; SQLSTATE: HY000

Message: Error releasing transaction '%d:%lld' for commit on session '%u' after being prepared on all group members.

ER_GRP_RPL_RELEASE_COMMIT_AFTER_GROUP_PREPARE_FAILED was added in 8.0.14.

- Error number: MY-013305; Symbol: ER_GRP_RPL_TRX_ALREADY_EXISTS_ON_TCM_ON_AFTER_CERTIFICATION; SQLSTATE: HY000

  Message: Transaction '%d:%lld' already exists on Group Replication consistency manager while being registered after conflict detection.

  ER_GRP_RPL_TRX_ALREADY_EXISTS_ON_TCM_ON_AFTER_CERTIFICATION was added in 8.0.14.

- Error number: MY-013306; Symbol: ER_GRP_RPL_FAILED_TO_INSERT_TRX_ON_TCM_ON_AFTER_CERTIFICATION; SQLSTATE: HY000

  Message: Error registering transaction '%d:%lld' on Group Replication consistency manager after conflict detection.

  ER_GRP_RPL_FAILED_TO_INSERT_TRX_ON_TCM_ON_AFTER_CERTIFICATION was added in 8.0.14.

- Error number: MY-013307; Symbol: ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_GROUP_PREPARE_FAILED; SQLSTATE: HY000

  Message: Error registering transaction '%d:%lld' from session '%u' to wait for being prepared on all group members.

  ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_GROUP_PREPARE_FAILED was added in 8.0.14.

- Error number: MY-013308; Symbol: ER_GRP_RPL_TRX_WAIT_FOR_GROUP_PREPARE_FAILED; SQLSTATE: HY000

  Message: Error on transaction '%d:%lld' from session '%u' while waiting for being prepared on all group members.

  ER_GRP_RPL_TRX_WAIT_FOR_GROUP_PREPARE_FAILED was added in 8.0.14.

- Error number: MY-013309; Symbol: ER_GRP_RPL_TRX_DOES_NOT_EXIST_ON_TCM_ON_HANDLE_REMOTE_PREPARE; SQLSTATE: HY000

  Message: Transaction '%d:%lld' does not exist on Group Replication consistency manager while receiving remote transaction prepare.

  ER_GRP_RPL_TRX_DOES_NOT_EXIST_ON_TCM_ON_HANDLE_REMOTE_PREPARE was added in 8.0.14.

- Error number: MY-013310; Symbol: ER_GRP_RPL_RELEASE_BEGIN_TRX_AFTER_DEPENDENCIES_COMMIT_FAILED; SQLSTATE: HY000

  Message: Error releasing transaction '%d:%lld' for execution on session '%u' after its dependencies did complete commit.

  ER_GRP_RPL_RELEASE_BEGIN_TRX_AFTER_DEPENDENCIES_COMMIT_FAILED was added in 8.0.14.

- Error number: MY-013311; Symbol: ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_DEPENDENCIES_FAILED; SQLSTATE: HY000

Message: Error registering transaction from session '%u' to wait for its dependencies to complete commit.

ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_DEPENDENCIES_FAILED was added in 8.0.14.

- Error number: MY-013312; Symbol: ER_GRP_RPL_WAIT_FOR_DEPENDENCIES_FAILED; SQLSTATE: HY000

  Message: Error on session '%u' while waiting for its dependencies to complete commit.

  ER_GRP_RPL_WAIT_FOR_DEPENDENCIES_FAILED was added in 8.0.14.

- Error number: MY-013313; Symbol: ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_SYNC_BEFORE_EXECUTION_FAILED; SQLSTATE: HY000

  Message: Error registering transaction from session '%u' to wait for sync before execution.

  ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_SYNC_BEFORE_EXECUTION_FAILED was added in 8.0.14.

- Error number: MY-013314; Symbol: ER_GRP_RPL_SEND_TRX_SYNC_BEFORE_EXECUTION_FAILED; SQLSTATE: HY000

  Message: Error sending sync before execution message from session '%u'.

  ER_GRP_RPL_SEND_TRX_SYNC_BEFORE_EXECUTION_FAILED was added in 8.0.14.

- Error number: MY-013315; Symbol: ER_GRP_RPL_TRX_WAIT_FOR_SYNC_BEFORE_EXECUTION_FAILED; SQLSTATE: HY000

  Message: Error on transaction from session '%u' while waiting for sync before execution.

  ER_GRP_RPL_TRX_WAIT_FOR_SYNC_BEFORE_EXECUTION_FAILED was added in 8.0.14.

- Error number: MY-013316; Symbol: ER_GRP_RPL_RELEASE_BEGIN_TRX_AFTER_WAIT_FOR_SYNC_BEFORE_EXEC; SQLSTATE: HY000

  Message: Error releasing transaction for execution on session '%u' after wait for sync before execution.

  ER_GRP_RPL_RELEASE_BEGIN_TRX_AFTER_WAIT_FOR_SYNC_BEFORE_EXEC was added in 8.0.14.

- Error number: MY-013317; Symbol: ER_GRP_RPL_TRX_WAIT_FOR_GROUP_GTID_EXECUTED; SQLSTATE: HY000

  Message: Error waiting for group executed transactions commit on session '%u'.

  ER_GRP_RPL_TRX_WAIT_FOR_GROUP_GTID_EXECUTED was added in 8.0.14.

- Error number: MY-013320; Symbol: ER_WARN_PROPERTY_STRING_PARSE_FAILED; SQLSTATE: HY000

  Message: Could not parse key-value pairs in property string '%s'

  ER_WARN_PROPERTY_STRING_PARSE_FAILED was added in 8.0.14.

- Error number: MY-013321; Symbol: ER_INVALID_PROPERTY_KEY; SQLSTATE: HY000

Message: Property key '%s' is invalid.

ER_INVALID_PROPERTY_KEY was added in 8.0.14.

- Error number: MY-013322; Symbol: ER_GRP_RPL_GTID_SET_EXTRACT_ERROR_DURING_RECOVERY; SQLSTATE: HY000

  Message: Error when extracting the group_replication_applier channel received transactions set. Unable to ensure the execution of group transactions received during recovery.

  ER_GRP_RPL_GTID_SET_EXTRACT_ERROR_DURING_RECOVERY was added in 8.0.14.

- Error number: MY-013323; Symbol: ER_SERVER_RPL_ENCRYPTION_FAILED_TO_ENCRYPT; SQLSTATE: HY000

  Message: Failed to encrypt content to write into binlog file: %s.

  ER_SERVER_RPL_ENCRYPTION_FAILED_TO_ENCRYPT was added in 8.0.14.

- Error number: MY-013324; Symbol: ER_CANNOT_GET_SERVER_VERSION_FROM_TABLESPACE_HEADER; SQLSTATE: HY000

  Message: Cannot get the server version number from the dictionary tablespace header.

  ER_CANNOT_GET_SERVER_VERSION_FROM_TABLESPACE_HEADER was added in 8.0.14.

- Error number: MY-013325; Symbol: ER_CANNOT_SET_SERVER_VERSION_IN_TABLESPACE_HEADER; SQLSTATE: HY000

  Message: Cannot set the server version number in the dictionary tablespace header.

  ER_CANNOT_SET_SERVER_VERSION_IN_TABLESPACE_HEADER was added in 8.0.14.

- Error number: MY-013326; Symbol: ER_SERVER_UPGRADE_VERSION_NOT_SUPPORTED; SQLSTATE: HY000

  Message: Upgrading the server from server version '%u' is not supported.

  ER_SERVER_UPGRADE_VERSION_NOT_SUPPORTED was added in 8.0.14.

- Error number: MY-013327; Symbol: ER_SERVER_UPGRADE_FROM_VERSION; SQLSTATE: HY000

  Message: MySQL server upgrading from version '%u' to '%u'.

  ER_SERVER_UPGRADE_FROM_VERSION was added in 8.0.14.

- Error number: MY-013328; Symbol: ER_GRP_RPL_ERROR_ON_CERT_DB_INSTALL; SQLSTATE: HY000

  Message: The certification information could not be set in this server: '%s'

  ER_GRP_RPL_ERROR_ON_CERT_DB_INSTALL was added in 8.0.14.

- Error number: MY-013329; Symbol: ER_GRP_RPL_FORCE_MEMBERS_WHEN_LEAVING; SQLSTATE: HY000

  Message: A request to force a new group membership was issued when the member is leaving the group.

ER_GRP_RPL_FORCE_MEMBERS_WHEN_LEAVING was added in 8.0.14.

- Error number: MY-013330; Symbol: ER_TRG_WRONG_ORDER; SQLSTATE: HY000

  Message: Trigger %s.%s for table %s.%s is listed in wrong order. Please drop and recreate all triggers for the table.

  ER_TRG_WRONG_ORDER was added in 8.0.14.

- Error number: MY-013332; Symbol: ER_LDAP_AUTH_GRP_SEARCH_NOT_SPECIAL_HDL; SQLSTATE: HY000

  Message: Special handling for group search, {GA} not found

  ER_LDAP_AUTH_GRP_SEARCH_NOT_SPECIAL_HDL was added in 8.0.14.

- Error number: MY-013333; Symbol: ER_LDAP_AUTH_GRP_USER_OBJECT_HAS_GROUP_INFO; SQLSTATE: HY000

  Message: User group retrieval: User object has group information

  ER_LDAP_AUTH_GRP_USER_OBJECT_HAS_GROUP_INFO was added in 8.0.14.

- Error number: MY-013334; Symbol: ER_LDAP_AUTH_GRP_INFO_FOUND_IN_MANY_OBJECTS; SQLSTATE: HY000

  Message: Group information found in multiple user objects. Search filter configuration is incorrect.

  ER_LDAP_AUTH_GRP_INFO_FOUND_IN_MANY_OBJECTS was added in 8.0.14.

- Error number: MY-013335; Symbol: ER_LDAP_AUTH_GRP_INCORRECT_ATTRIBUTE; SQLSTATE: HY000

  Message: User group retrieval: no group attribute found. Incorrect group search attribute.

  ER_LDAP_AUTH_GRP_INCORRECT_ATTRIBUTE was added in 8.0.14.

- Error number: MY-013336; Symbol: ER_LDAP_AUTH_GRP_NULL_ATTRIBUTE_VALUE; SQLSTATE: HY000

  Message: User group retrieval: Group attribute values is NULL.

  ER_LDAP_AUTH_GRP_NULL_ATTRIBUTE_VALUE was added in 8.0.14.

- Error number: MY-013337; Symbol: ER_LDAP_AUTH_GRP_DN_PARSING_FAILED; SQLSTATE: HY000

  Message: User group retrieval: parsing DN failed.

  ER_LDAP_AUTH_GRP_DN_PARSING_FAILED was added in 8.0.14.

- Error number: MY-013338; Symbol: ER_LDAP_AUTH_GRP_OBJECT_HAS_USER_INFO; SQLSTATE: HY000

  Message: User group retrieval: Group object has user information

  ER_LDAP_AUTH_GRP_OBJECT_HAS_USER_INFO was added in 8.0.14.

- Error number: MY-013339; Symbol: ER_LDAP_AUTH_LDAPS; SQLSTATE: HY000

  Message: Reserved port for ldaps using ldaps

  ER_LDAP_AUTH_LDAPS was added in 8.0.14.

- Error number: MY-013340; Symbol: ER_LDAP_MAPPING_GET_USER_PROXY; SQLSTATE: HY000

  Message: Get user proxy

  ER_LDAP_MAPPING_GET_USER_PROXY was added in 8.0.14.

- Error number: MY-013341; Symbol: ER_LDAP_MAPPING_USER_DONT_BELONG_GROUP; SQLSTATE: HY000

  Message: Get user proxy: User doesn't belongs to any group, user name will be treated as authenticated user.

  ER_LDAP_MAPPING_USER_DONT_BELONG_GROUP was added in 8.0.14.

- Error number: MY-013342; Symbol: ER_LDAP_MAPPING_INFO; SQLSTATE: HY000

  Message: Get user proxy: configured mapping info: %s

  ER_LDAP_MAPPING_INFO was added in 8.0.14.

- Error number: MY-013343; Symbol: ER_LDAP_MAPPING_EMPTY_MAPPING; SQLSTATE: HY000

  Message: Get user proxy: User doesn't have group mapping information, First LDAP group will be treated as authenticated user.

  ER_LDAP_MAPPING_EMPTY_MAPPING was added in 8.0.14.

- Error number: MY-013344; Symbol: ER_LDAP_MAPPING_PROCESS_MAPPING; SQLSTATE: HY000

  Message: Process group proxy mapping

  ER_LDAP_MAPPING_PROCESS_MAPPING was added in 8.0.14.

- Error number: MY-013345; Symbol: ER_LDAP_MAPPING_CHECK_DELIMI_QUOTE; SQLSTATE: HY000

  Message: Check delimiter after quote

  ER_LDAP_MAPPING_CHECK_DELIMI_QUOTE was added in 8.0.14.

- Error number: MY-013346; Symbol: ER_LDAP_MAPPING_PROCESS_DELIMITER; SQLSTATE: HY000

  Message: Processing delimiter

  ER_LDAP_MAPPING_PROCESS_DELIMITER was added in 8.0.14.

- Error number: MY-013347; Symbol: ER_LDAP_MAPPING_PROCESS_DELIMITER_EQUAL_NOT_FOUND; SQLSTATE: HY000

  Message: Processing delimiter, separator = not found, resetting position

  ER_LDAP_MAPPING_PROCESS_DELIMITER_EQUAL_NOT_FOUND was added in 8.0.14.

- Error number: MY-013348; Symbol: ER_LDAP_MAPPING_PROCESS_DELIMITER_TRY_COMMA; SQLSTATE: HY000

  Message: "Processing delimiter, failed to get data for = separator try for separator ,."

  ER_LDAP_MAPPING_PROCESS_DELIMITER_TRY_COMMA was added in 8.0.14.

- Error number: MY-013349; Symbol: ER_LDAP_MAPPING_PROCESS_DELIMITER_COMMA_NOT_FOUND; SQLSTATE: HY000

  Message: Processing delimiter, separator , not found, resetting position

  ER_LDAP_MAPPING_PROCESS_DELIMITER_COMMA_NOT_FOUND was added in 8.0.14.

- Error number: MY-013350; Symbol: ER_LDAP_MAPPING_NO_SEPEARATOR_END_OF_GROUP; SQLSTATE: HY000

  Message: Processing delimiter: No mapping separator is found, end of group information

  ER_LDAP_MAPPING_NO_SEPEARATOR_END_OF_GROUP was added in 8.0.14.

- Error number: MY-013351; Symbol: ER_LDAP_MAPPING_GETTING_NEXT_MAPPING; SQLSTATE: HY000

  Message: Getting next mapping information

  ER_LDAP_MAPPING_GETTING_NEXT_MAPPING was added in 8.0.14.

- Error number: MY-013352; Symbol: ER_LDAP_MAPPING_PARSING_CURRENT_STATE; SQLSTATE: HY000

  Message: Parsing mapping, current state: %d delimiter char: %c

  ER_LDAP_MAPPING_PARSING_CURRENT_STATE was added in 8.0.14.

- Error number: MY-013353; Symbol: ER_LDAP_MAPPING_PARSING_MAPPING_INFO; SQLSTATE: HY000

  Message: Parsing mapping info, LDAP group: %s MySQL proxy: %s

  ER_LDAP_MAPPING_PARSING_MAPPING_INFO was added in 8.0.14.

- Error number: MY-013354; Symbol: ER_LDAP_MAPPING_PARSING_ERROR; SQLSTATE: HY000

  Message: Mapping parsing error

  ER_LDAP_MAPPING_PARSING_ERROR was added in 8.0.14.

- Error number: MY-013355; Symbol: ER_LDAP_MAPPING_TRIMMING_SPACES; SQLSTATE: HY000

  Message: Trimming left spaces

  ER_LDAP_MAPPING_TRIMMING_SPACES was added in 8.0.14.

- Error number: MY-013356; Symbol: ER_LDAP_MAPPING_IS_QUOTE; SQLSTATE: HY000

  Message: Checking if current characters is quote

  ER_LDAP_MAPPING_IS_QUOTE was added in 8.0.14.

- Error number: MY-013357; Symbol: ER_LDAP_MAPPING_NON_DESIRED_STATE; SQLSTATE: HY000

  Message: Not desired state or un-defined states.

  ER_LDAP_MAPPING_NON_DESIRED_STATE was added in 8.0.14.

- Error number: MY-013358; Symbol: ER_INVALID_NAMED_PIPE_FULL_ACCESS_GROUP; SQLSTATE: HY000

  Message: Invalid value for named_pipe_full_access_group.

  ER_INVALID_NAMED_PIPE_FULL_ACCESS_GROUP was added in 8.0.14.

- Error number: MY-013359; Symbol: ER_PREPARE_FOR_SECONDARY_ENGINE; SQLSTATE: HY000

  Message: Retry the statement using a secondary storage engine.

  ER_PREPARE_FOR_SECONDARY_ENGINE was added in 8.0.16.

- Error number: MY-013360; Symbol: ER_SERVER_WARN_DEPRECATED; SQLSTATE: HY000

  Message: '%s' is deprecated and will be removed in a future release. Please use %s instead

  ER_SERVER_WARN_DEPRECATED was added in 8.0.16.

- Error number: MY-013361; Symbol: ER_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES; SQLSTATE: HY000

  Message: Cannot set mandatory_roles: AuthId `%s`@`%s` has '%s' privilege.

  ER_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES was added in 8.0.16.

- Error number: MY-013362; Symbol: ER_SERVER_BINLOG_MASTER_KEY_RECOVERY_OUT_OF_COMBINATION; SQLSTATE: HY000

  Message: Unable to recover binary log master key, the combination of new_master_key_seqno=%u, master_key_seqno=%u and old_master_key_seqno=%u are wrong.

  ER_SERVER_BINLOG_MASTER_KEY_RECOVERY_OUT_OF_COMBINATION was added in 8.0.16.

- Error number: MY-013363; Symbol: ER_SERVER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_AUX_KEY; SQLSTATE: HY000

  Message: Failed to remove auxiliary binary log encryption key from keyring, please check if keyring plugin is loaded. The cleanup of the binary log master key rotation process did not finish as expected and the cleanup will take place upon server restart or next 'ALTER INSTANCE ROTATE BINLOG MASTER KEY' execution.

  ER_SERVER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_AUX_KEY was added in 8.0.16.

- Error number: MY-013368; Symbol: ER_TURNING_ON_PARTIAL_REVOKES; SQLSTATE: HY000

  Message: At least one partial revoke exists on a database. Turning ON the system variable '@@partial_revokes'.

  ER_TURNING_ON_PARTIAL_REVOKES was added in 8.0.16.

- Error number: MY-013369; Symbol: ER_WARN_PARTIAL_REVOKE_AND_DB_GRANT; SQLSTATE: HY000

Message: For user '%s'@'%s', one or more privileges granted through mysql.db for database '%s', conflict with partial revoke. It could mean 'mysql' schema is corrupted.

`ER_WARN_PARTIAL_REVOKE_AND_DB_GRANT` was added in 8.0.16.

- Error number: `MY-013370`; Symbol: `ER_WARN_INCORRECT_PRIVILEGE_FOR_DB_RESTRICTIONS`; SQLSTATE: `HY000`

  Message: For user %s, ignored restrictions for privilege(s) '%s' for database '%s' as these are not valid database privileges.

  `ER_WARN_INCORRECT_PRIVILEGE_FOR_DB_RESTRICTIONS` was added in 8.0.16.

- Error number: `MY-013371`; Symbol: `ER_WARN_INVALID_DB_RESTRICTIONS`; SQLSTATE: `HY000`

  Message: For user %s, ignored restrictions for privilege(s) '%s' for database '%s' as corresponding global privilege(s) are not granted.

  `ER_WARN_INVALID_DB_RESTRICTIONS` was added in 8.0.16.

- Error number: `MY-013372`; Symbol: `ER_GRP_RPL_INVALID_COMMUNICATION_PROTOCOL`; SQLSTATE: `HY000`

  Message: '%s' is an invalid value for group_replication_communication_protocol_join, please use a MySQL version between 5.7.14 and this server's version

  `ER_GRP_RPL_INVALID_COMMUNICATION_PROTOCOL` was added in 8.0.16.

- Error number: `MY-013373`; Symbol: `ER_GRP_RPL_STARTED_AUTO_REJOIN`; SQLSTATE: `HY000`

  Message: Started auto-rejoin procedure attempt %lu of %lu

  `ER_GRP_RPL_STARTED_AUTO_REJOIN` was added in 8.0.16.

- Error number: `MY-013374`; Symbol: `ER_GRP_RPL_TIMEOUT_RECEIVED_VC_ON_REJOIN`; SQLSTATE: `HY000`

  Message: Timeout while waiting for a view change event during the auto-rejoin procedure

  `ER_GRP_RPL_TIMEOUT_RECEIVED_VC_ON_REJOIN` was added in 8.0.16.

- Error number: `MY-013375`; Symbol: `ER_GRP_RPL_FINISHED_AUTO_REJOIN`; SQLSTATE: `HY000`

  Message: Auto-rejoin procedure attempt %lu of %lu finished. Member was%s able to join the group.

  `ER_GRP_RPL_FINISHED_AUTO_REJOIN` was added in 8.0.16.

- Error number: `MY-013376`; Symbol: `ER_GRP_RPL_DEFAULT_TABLE_ENCRYPTION_DIFF_FROM_GRP`; SQLSTATE: `HY000`

  Message: The member is configured with a default_table_encryption option value '%d' different from the group '%d'. The member will now exit the group.

  `ER_GRP_RPL_DEFAULT_TABLE_ENCRYPTION_DIFF_FROM_GRP` was added in 8.0.16.

- Error number: `MY-013377`; Symbol: `ER_SERVER_UPGRADE_OFF`; SQLSTATE: `HY000`

Message: Server shutting down because upgrade is required, yet prohibited by the command line option '--upgrade=NONE'.

ER_SERVER_UPGRADE_OFF was added in 8.0.16.

- Error number: MY-013378; Symbol: ER_SERVER_UPGRADE_SKIP; SQLSTATE: HY000

  Message: Server upgrade is required, but skipped by command line option '--upgrade=MINIMAL'.

  ER_SERVER_UPGRADE_SKIP was added in 8.0.16.

- Error number: MY-013379; Symbol: ER_SERVER_UPGRADE_PENDING; SQLSTATE: HY000

  Message: Server upgrade started with version %d, but server upgrade of version %d is still pending.

  ER_SERVER_UPGRADE_PENDING was added in 8.0.16.

- Error number: MY-013380; Symbol: ER_SERVER_UPGRADE_FAILED; SQLSTATE: HY000

  Message: Failed to upgrade server.

  ER_SERVER_UPGRADE_FAILED was added in 8.0.16.

- Error number: MY-013381; Symbol: ER_SERVER_UPGRADE_STATUS; SQLSTATE: HY000

  Message: Server upgrade from '%d' to '%d' %s.

  ER_SERVER_UPGRADE_STATUS was added in 8.0.16.

- Error number: MY-013382; Symbol: ER_SERVER_UPGRADE_REPAIR_REQUIRED; SQLSTATE: HY000

  Message: Table '%s' requires repair.

  ER_SERVER_UPGRADE_REPAIR_REQUIRED was added in 8.0.16.

- Error number: MY-013383; Symbol: ER_SERVER_UPGRADE_REPAIR_STATUS; SQLSTATE: HY000

  Message: Table '%s' repair %s.

  ER_SERVER_UPGRADE_REPAIR_STATUS was added in 8.0.16.

- Error number: MY-013384; Symbol: ER_SERVER_UPGRADE_INFO_FILE; SQLSTATE: HY000

  Message: Could not open server upgrade info file '%s' for writing. Please make sure the file is writable.

  ER_SERVER_UPGRADE_INFO_FILE was added in 8.0.16.

- Error number: MY-013385; Symbol: ER_SERVER_UPGRADE_SYS_SCHEMA; SQLSTATE: HY000

  Message: Upgrading the sys schema.

  ER_SERVER_UPGRADE_SYS_SCHEMA was added in 8.0.16.

- Error number: MY-013386; Symbol: ER_SERVER_UPGRADE_MYSQL_TABLES; SQLSTATE: HY000

  Message: Running queries to upgrade MySQL server.

  ER_SERVER_UPGRADE_MYSQL_TABLES was added in 8.0.16.

- Error number: MY-013387; Symbol: ER_SERVER_UPGRADE_SYSTEM_TABLES; SQLSTATE: HY000

  Message: Upgrading system table data.

  ER_SERVER_UPGRADE_SYSTEM_TABLES was added in 8.0.16.

- Error number: MY-013388; Symbol: ER_SERVER_UPGRADE_EMPTY_SYS; SQLSTATE: HY000

  Message: Found empty sys database. Installing the sys schema.

  ER_SERVER_UPGRADE_EMPTY_SYS was added in 8.0.16.

- Error number: MY-013389; Symbol: ER_SERVER_UPGRADE_NO_SYS_VERSION; SQLSTATE: HY000

  Message: A sys schema exists with no sys.version view. If you have a user created sys schema, this must be renamed for the upgrade to succeed.

  ER_SERVER_UPGRADE_NO_SYS_VERSION was added in 8.0.16.

- Error number: MY-013390; Symbol: ER_SERVER_UPGRADE_SYS_VERSION_EMPTY; SQLSTATE: HY000

  Message: A sys schema exists with a sys.version view, but it returns no results.

  ER_SERVER_UPGRADE_SYS_VERSION_EMPTY was added in 8.0.16.

- Error number: MY-013391; Symbol: ER_SERVER_UPGRADE_SYS_SCHEMA_OUTDATED; SQLSTATE: HY000

  Message: Found outdated sys schema version %s.

  ER_SERVER_UPGRADE_SYS_SCHEMA_OUTDATED was added in 8.0.16.

- Error number: MY-013392; Symbol: ER_SERVER_UPGRADE_SYS_SCHEMA_UP_TO_DATE; SQLSTATE: HY000

  Message: The sys schema is already up to date (version %s).

  ER_SERVER_UPGRADE_SYS_SCHEMA_UP_TO_DATE was added in 8.0.16.

- Error number: MY-013393; Symbol: ER_SERVER_UPGRADE_SYS_SCHEMA_OBJECT_COUNT; SQLSTATE: HY000

  Message: Found %d sys %s, but expected %d. Re-installing the sys schema.

  ER_SERVER_UPGRADE_SYS_SCHEMA_OBJECT_COUNT was added in 8.0.16.

- Error number: MY-013394; Symbol: ER_SERVER_UPGRADE_CHECKING_DB; SQLSTATE: HY000

  Message: Checking '%s' schema.

  ER_SERVER_UPGRADE_CHECKING_DB was added in 8.0.16.

- Error number: MY-013395; Symbol: ER_IB_MSG_DDL_LOG_DELETE_BY_ID_TMCT; SQLSTATE: HY000

  Message: Too many concurrent transactions while clearing the DDL Log. Please increase the number of Rollback Segments.

ER_IB_MSG_DDL_LOG_DELETE_BY_ID_TMCT was added in 8.0.16.

- Error number: MY-013396; Symbol: ER_IB_MSG_POST_RECOVER_DDL_LOG_RECOVER; SQLSTATE: HY000

  Message: Error in DDL Log recovery during Post-Recovery processing.

  ER_IB_MSG_POST_RECOVER_DDL_LOG_RECOVER was added in 8.0.16.

- Error number: MY-013397; Symbol: ER_IB_MSG_POST_RECOVER_POST_TS_ENCRYPT; SQLSTATE: HY000

  Message: Error in Post-Tablespace-Encryption during Post-Recovery processing.

  ER_IB_MSG_POST_RECOVER_POST_TS_ENCRYPT was added in 8.0.16.

- Error number: MY-013398; Symbol: ER_IB_MSG_DDL_LOG_FAIL_POST_DDL; SQLSTATE: HY000

  Message: Error in DLL Log cleanup during Post-DDL processing.

  ER_IB_MSG_DDL_LOG_FAIL_POST_DDL was added in 8.0.16.

- Error number: MY-013399; Symbol: ER_SERVER_BINLOG_UNSAFE_SYSTEM_FUNCTION; SQLSTATE: HY000

  Message: '%s' statement is unsafe because it uses a system function that may return a different value on the slave.

  ER_SERVER_BINLOG_UNSAFE_SYSTEM_FUNCTION was added in 8.0.16.

- Error number: MY-013400; Symbol: ER_SERVER_UPGRADE_HELP_TABLE_STATUS; SQLSTATE: HY000

  Message: Upgrade of help tables %s.

  ER_SERVER_UPGRADE_HELP_TABLE_STATUS was added in 8.0.16.

- Error number: MY-013401; Symbol: ER_GRP_RPL_SRV_GTID_WAIT_ERROR; SQLSTATE: HY000

  Message: Error when waiting for the server to execute local transactions in order assure the group change proper logging

  ER_GRP_RPL_SRV_GTID_WAIT_ERROR was added in 8.0.16.

- Error number: MY-013402; Symbol: ER_GRP_DELAYED_VCLE_LOGGING; SQLSTATE: HY000

  Message: Unable to log the group change View log event in its exaction position in the log. This will not however affect the group replication recovery process or the overall plugin process.

  ER_GRP_DELAYED_VCLE_LOGGING was added in 8.0.16.

- Error number: MY-013404; Symbol: ER_BINLOG_UNABLE_TO_ROTATE_GTID_TABLE_READONLY; SQLSTATE: HY000

  Message: Unable to create a new binlog file: Table `mysql.gtid_executed` couldn't be opened. %s

  ER_BINLOG_UNABLE_TO_ROTATE_GTID_TABLE_READONLY was added in 8.0.16.

- Error number: MY-013405; Symbol: ER_NETWORK_NAMESPACES_NOT_SUPPORTED; SQLSTATE: HY000

  Message: Network Namespaces is not supported on this platform

  ER_NETWORK_NAMESPACES_NOT_SUPPORTED was added in 8.0.16.

- Error number: MY-013406; Symbol: ER_UNKNOWN_NETWORK_NAMESPACE; SQLSTATE: HY000

  Message: Unknown network namespace '%s'

  ER_UNKNOWN_NETWORK_NAMESPACE was added in 8.0.16.

- Error number: MY-013407; Symbol: ER_NETWORK_NAMESPACE_NOT_ALLOWED_FOR_WILDCARD_ADDRESS; SQLSTATE: HY000

  Message: Network namespace not allowed for wildcard interface address

  ER_NETWORK_NAMESPACE_NOT_ALLOWED_FOR_WILDCARD_ADDRESS was added in 8.0.16.

- Error number: MY-013408; Symbol: ER_SETNS_FAILED; SQLSTATE: HY000

  Message: setns() failed with error '%s'

  ER_SETNS_FAILED was added in 8.0.16.

- Error number: MY-013409; Symbol: ER_WILDCARD_NOT_ALLOWED_FOR_MULTIADDRESS_BIND; SQLSTATE: HY000

  Message: Wildcard address value not allowed for multivalued bind address

  ER_WILDCARD_NOT_ALLOWED_FOR_MULTIADDRESS_BIND was added in 8.0.16.

- Error number: MY-013410; Symbol: ER_NETWORK_NAMESPACE_FILE_PATH_TOO_LONG; SQLSTATE: HY000

  Message: The path to a special network namespace file is too long. (got %u > max %u)

  ER_NETWORK_NAMESPACE_FILE_PATH_TOO_LONG was added in 8.0.16.

- Error number: MY-013411; Symbol: ER_IB_MSG_TOO_LONG_PATH; SQLSTATE: HY000

  Message: Cannot create tablespace '%s'. The filepath is too long for this OS.

  ER_IB_MSG_TOO_LONG_PATH was added in 8.0.16.

- Error number: MY-013412; Symbol: ER_IB_RECV_FIRST_REC_GROUP_INVALID; SQLSTATE: HY000

  Message: The last block of redo had corrupted first_rec_group and became fixed (%u -> %u).

  ER_IB_RECV_FIRST_REC_GROUP_INVALID was added in 8.0.16.

- Error number: MY-013413; Symbol: ER_DD_UPGRADE_COMPLETED; SQLSTATE: HY000

  Message: Data dictionary upgrade from version '%u' to '%u' completed.

  ER_DD_UPGRADE_COMPLETED was added in 8.0.16.

- Error number: MY-013414; Symbol: ER_SSL_SERVER_CERT_VERIFY_FAILED; SQLSTATE: HY000

Message: Server SSL certificate doesn't verify: %s

ER_SSL_SERVER_CERT_VERIFY_FAILED was added in 8.0.16.

- Error number: MY-013415; Symbol: ER_PERSIST_OPTION_USER_TRUNCATED; SQLSTATE: HY000

  Message: Truncated a user name for %s that was too long while reading the persisted variables file

  ER_PERSIST_OPTION_USER_TRUNCATED was added in 8.0.16.

- Error number: MY-013416; Symbol: ER_PERSIST_OPTION_HOST_TRUNCATED; SQLSTATE: HY000

  Message: Truncated a host name for %s that was too long while reading the persisted variables file

  ER_PERSIST_OPTION_HOST_TRUNCATED was added in 8.0.16.

- Error number: MY-013417; Symbol: ER_NET_WAIT_ERROR; SQLSTATE: HY000

  Message: The wait_timeout period was exceeded, the idle time since last command was too long.

  ER_NET_WAIT_ERROR was added in 8.0.16.

- Error number: MY-013418; Symbol: ER_IB_MSG_1285; SQLSTATE: HY000

  Message: '%s' found not encrypted while '%s' is ON. Trying to encrypt it now.

  ER_IB_MSG_1285 was added in 8.0.16.

- Error number: MY-013419; Symbol: ER_IB_MSG_CLOCK_MONOTONIC_UNSUPPORTED; SQLSTATE: HY000

  Message: CLOCK_MONOTONIC is unsupported, so do not change the system time when MySQL is running !

  ER_IB_MSG_CLOCK_MONOTONIC_UNSUPPORTED was added in 8.0.16.

- Error number: MY-013420; Symbol: ER_IB_MSG_CLOCK_GETTIME_FAILED; SQLSTATE: HY000

  Message: clock_gettime() failed: %s

  ER_IB_MSG_CLOCK_GETTIME_FAILED was added in 8.0.16.

- Error number: MY-013421; Symbol: ER_PLUGIN_NOT_EARLY_DUP; SQLSTATE: HY000

  Message: Plugin '%s' is not to be used as an "early" plugin. Don't add it to --early-plugin-load, keyring migration etc.

  ER_PLUGIN_NOT_EARLY_DUP was added in 8.0.16.

- Error number: MY-013422; Symbol: ER_PLUGIN_NO_INSTALL_DUP; SQLSTATE: HY000

  Message: Plugin '%s' is marked as not dynamically installable. You have to stop the server to install it.

  ER_PLUGIN_NO_INSTALL_DUP was added in 8.0.16.

- Error number: MY-013423; Symbol: ER_WARN_DEPRECATED_SQL_CALC_FOUND_ROWS; SQLSTATE: HY000

Message: SQL_CALC_FOUND_ROWS is deprecated and will be removed in a future release. Consider using two separate queries instead.

ER_WARN_DEPRECATED_SQL_CALC_FOUND_ROWS was added in 8.0.16.

- Error number: MY-013424; Symbol: ER_WARN_DEPRECATED_FOUND_ROWS; SQLSTATE: HY000

  Message: FOUND_ROWS() is deprecated and will be removed in a future release. Consider using COUNT(*) instead.

  ER_WARN_DEPRECATED_FOUND_ROWS was added in 8.0.16.

- Error number: MY-013425; Symbol: ER_BINLOG_UNSAFE_DEFAULT_EXPRESSION_IN_SUBSTATEMENT; SQLSTATE: HY000

  Message: The statement is unsafe because it invokes a trigger or a stored function that modifies a table that has a column with a DEFAULT expression that may return a different value on the slave.

  ER_BINLOG_UNSAFE_DEFAULT_EXPRESSION_IN_SUBSTATEMENT was added in 8.0.16.

- Error number: MY-013426; Symbol: ER_GRP_RPL_MEMBER_VER_READ_COMPATIBLE; SQLSTATE: HY000

  Message: Member version is read compatible with the group.

  ER_GRP_RPL_MEMBER_VER_READ_COMPATIBLE was added in 8.0.16.

- Error number: MY-013427; Symbol: ER_LOCK_ORDER_INIT_FAILED; SQLSTATE: HY000

  Message: Lock order disabled (reason: init failed).

  ER_LOCK_ORDER_INIT_FAILED was added in 8.0.16.

- Error number: MY-013428; Symbol: ER_AUDIT_LOG_KEYRING_ID_TIMESTAMP_VALUE_IS_INVALID; SQLSTATE: HY000

  Message: Keyring ID timestamp value is invalid: '%s'

  ER_AUDIT_LOG_KEYRING_ID_TIMESTAMP_VALUE_IS_INVALID was added in 8.0.16.

- Error number: MY-013429; Symbol: ER_AUDIT_LOG_FILE_NAME_TIMESTAMP_VALUE_IS_MISSING_OR_INVALID; SQLSTATE: HY000

  Message: Cannot process audit log file. File name timestamp value is missing or invalid: '%s'

  ER_AUDIT_LOG_FILE_NAME_TIMESTAMP_VALUE_IS_MISSING_OR_INVALID was added in 8.0.16.

- Error number: MY-013430; Symbol: ER_AUDIT_LOG_FILE_NAME_DOES_NOT_HAVE_REQUIRED_FORMAT; SQLSTATE: HY000

  Message: Cannot process audit log file. File name does not have required format: '%s'

  ER_AUDIT_LOG_FILE_NAME_DOES_NOT_HAVE_REQUIRED_FORMAT was added in 8.0.16.

- Error number: MY-013431; Symbol: ER_AUDIT_LOG_FILE_NAME_KEYRING_ID_VALUE_IS_MISSING; SQLSTATE: HY000

  Message: Cannot process audit log file. File name keyring ID value is missing: '%s'

ER_AUDIT_LOG_FILE_NAME_KEYRING_ID_VALUE_IS_MISSING was added in 8.0.16.

- Error number: MY-013432; Symbol: ER_AUDIT_LOG_FILE_HAS_BEEN_SUCCESSFULLY_PROCESSED; SQLSTATE: HY000

  Message: Audit log file has been successfully processed: '%s'

  ER_AUDIT_LOG_FILE_HAS_BEEN_SUCCESSFULLY_PROCESSED was added in 8.0.16.

- Error number: MY-013433; Symbol: ER_AUDIT_LOG_COULD_NOT_OPEN_FILE_FOR_READING; SQLSTATE: HY000

  Message: Could not open audit log file for reading: '%s'

  ER_AUDIT_LOG_COULD_NOT_OPEN_FILE_FOR_READING was added in 8.0.16.

- Error number: MY-013434; Symbol: ER_AUDIT_LOG_INVALID_FILE_CONTENT; SQLSTATE: HY000

  Message: Invalid audit log file content: '%s'

  ER_AUDIT_LOG_INVALID_FILE_CONTENT was added in 8.0.16.

- Error number: MY-013435; Symbol: ER_AUDIT_LOG_CANNOT_READ_PASSWORD; SQLSTATE: HY000

  Message: Cannot read password: '%s'

  ER_AUDIT_LOG_CANNOT_READ_PASSWORD was added in 8.0.16.

- Error number: MY-013436; Symbol: ER_AUDIT_LOG_CANNOT_STORE_PASSWORD; SQLSTATE: HY000

  Message: Cannot store password: '%s'

  ER_AUDIT_LOG_CANNOT_STORE_PASSWORD was added in 8.0.16.

- Error number: MY-013437; Symbol: ER_AUDIT_LOG_CANNOT_REMOVE_PASSWORD; SQLSTATE: HY000

  Message: Cannot remove password: '%s'

  ER_AUDIT_LOG_CANNOT_REMOVE_PASSWORD was added in 8.0.16.

- Error number: MY-013438; Symbol: ER_AUDIT_LOG_PASSWORD_HAS_BEEN_COPIED; SQLSTATE: HY000

  Message: 'audit_log' password has been copied into '%s' and will be removed with first purged password.

  ER_AUDIT_LOG_PASSWORD_HAS_BEEN_COPIED was added in 8.0.16.

- Error number: MY-013447; Symbol: ER_LDAP_EMPTY_USERDN_PASSWORD; SQLSTATE: HY000

  Message: Empty user dn or password is not allowed, not attempting LDAP bind.

  ER_LDAP_EMPTY_USERDN_PASSWORD was added in 8.0.16.

- Error number: MY-013449; Symbol: ER_ACL_WRONG_OR_MISSING_ACL_TABLES_LOG; SQLSTATE: HY000

Message: The current layout of the ACL tables does not conform to the server's expected layout. They're either altered, missing or not upgraded from a previous version. However a best effort attempt to read data from these tables will still be made.

ER_ACL_WRONG_OR_MISSING_ACL_TABLES_LOG was added in 8.0.17.

- Error number: MY-013450; Symbol: ER_LOCK_ORDER_FAILED_WRITE_FILE; SQLSTATE: HY000

  Message: LOCK_ORDER: Failed to write to file <%s>.

  ER_LOCK_ORDER_FAILED_WRITE_FILE was added in 8.0.17.

- Error number: MY-013451; Symbol: ER_LOCK_ORDER_FAILED_READ_FILE; SQLSTATE: HY000

  Message: LOCK_ORDER: Failed to read from file <%s>.

  ER_LOCK_ORDER_FAILED_READ_FILE was added in 8.0.17.

- Error number: MY-013452; Symbol: ER_LOCK_ORDER_MESSAGE; SQLSTATE: HY000

  Message: LOCK_ORDER message: %s

  ER_LOCK_ORDER_MESSAGE was added in 8.0.17.

- Error number: MY-013453; Symbol: ER_LOCK_ORDER_DEPENDENCIES_SYNTAX; SQLSTATE: HY000

  Message: Lock order dependencies file <%s> (%d:%d) - (%d:%d) : %s

  ER_LOCK_ORDER_DEPENDENCIES_SYNTAX was added in 8.0.17.

- Error number: MY-013454; Symbol: ER_LOCK_ORDER_SCANNER_SYNTAX; SQLSTATE: HY000

  Message: Lock order scanner: (%d:%d) - (%d:%d) : %s

  ER_LOCK_ORDER_SCANNER_SYNTAX was added in 8.0.17.

- Error number: MY-013455; Symbol: ER_DATA_DIRECTORY_UNUSABLE_DELETABLE; SQLSTATE: HY000

  Message: The newly created data directory %s by --initialize is unusable. You can remove it.

  ER_DATA_DIRECTORY_UNUSABLE_DELETABLE was added in 8.0.17.

- Error number: MY-013456; Symbol: ER_IB_MSG_BTREE_LEVEL_LIMIT_EXCEEDED; SQLSTATE: HY000

  Message: No. of B-tree level created for index %s has crossed the permissible limit. If debug option innodb_limit_optimistic_insert_debug is being used try tweaking it to include more records in a page.

  ER_IB_MSG_BTREE_LEVEL_LIMIT_EXCEEDED was added in 8.0.17.

- Error number: MY-013457; Symbol: ER_IB_CLONE_START_STOP; SQLSTATE: HY000

  Message: %s

  ER_IB_CLONE_START_STOP was added in 8.0.17.

- Error number: MY-013458; Symbol: ER_IB_CLONE_OPERATION; SQLSTATE: HY000

Message: %s

ER_IB_CLONE_OPERATION was added in 8.0.17.

- Error number: MY-013459; Symbol: ER_IB_CLONE_RESTART; SQLSTATE: HY000

  Message: %s

  ER_IB_CLONE_RESTART was added in 8.0.17.

- Error number: MY-013460; Symbol: ER_IB_CLONE_USER_DATA; SQLSTATE: HY000

  Message: Clone removing all user data for provisioning: %s

  ER_IB_CLONE_USER_DATA was added in 8.0.17.

- Error number: MY-013461; Symbol: ER_IB_CLONE_NON_INNODB_TABLE; SQLSTATE: HY000

  Message: Non innodb table: %s.%s is not cloned and is empty.

  ER_IB_CLONE_NON_INNODB_TABLE was added in 8.0.17.

- Error number: MY-013462; Symbol: ER_CLONE_SHUTDOWN_TRACE; SQLSTATE: HY000

  Message: Clone shutting down server as RESTART failed. Please start server to complete clone operation.

  ER_CLONE_SHUTDOWN_TRACE was added in 8.0.17.

- Error number: MY-013463; Symbol: ER_GRP_RPL_GTID_PURGED_EXTRACT_ERROR; SQLSTATE: HY000

  Message: Error when extracting this member GTID purged set. Operations and checks made to group joiners may be incomplete.

  ER_GRP_RPL_GTID_PURGED_EXTRACT_ERROR was added in 8.0.17.

- Error number: MY-013464; Symbol: ER_GRP_RPL_CLONE_PROCESS_PREPARE_ERROR; SQLSTATE: HY000

  Message: There was an issue when configuring the remote cloning process: %s

  ER_GRP_RPL_CLONE_PROCESS_PREPARE_ERROR was added in 8.0.17.

- Error number: MY-013465; Symbol: ER_GRP_RPL_CLONE_PROCESS_EXEC_ERROR; SQLSTATE: HY000

  Message: There was an issue when cloning from another server: %s

  ER_GRP_RPL_CLONE_PROCESS_EXEC_ERROR was added in 8.0.17.

- Error number: MY-013466; Symbol: ER_GRP_RPL_RECOVERY_EVAL_ERROR; SQLSTATE: HY000

  Message: There was an issue when trying to evaluate the best distributed recovery strategy while joining.%s

  ER_GRP_RPL_RECOVERY_EVAL_ERROR was added in 8.0.17.

- Error number: MY-013467; Symbol: ER_GRP_RPL_NO_POSSIBLE_RECOVERY; SQLSTATE: HY000

Message: No valid or ONLINE members exist to get the missing data from the group. For cloning check if donors of the same version and with clone plugin installed exist. For incremental recovery check if you have donors where the required data was not purged from the binary logs.

`ER_GRP_RPL_NO_POSSIBLE_RECOVERY` was added in 8.0.17.

- Error number: `MY-013468`; Symbol: `ER_GRP_RPL_CANT_KILL_THREAD`; SQLSTATE: `HY000`

  Message: The group replication plugin could not kill the plugin routine for %s. %s

  `ER_GRP_RPL_CANT_KILL_THREAD` was added in 8.0.17.

- Error number: `MY-013469`; Symbol: `ER_GRP_RPL_RECOVERY_STRAT_CLONE_THRESHOLD`; SQLSTATE: `HY000`

  Message: Due to the number of missing transactions being higher than the configured threshold of %llu, this member will start distributed recovery using clone.

  `ER_GRP_RPL_RECOVERY_STRAT_CLONE_THRESHOLD` was added in 8.0.17.

- Error number: `MY-013470`; Symbol: `ER_GRP_RPL_RECOVERY_STRAT_CLONE_PURGED`; SQLSTATE: `HY000`

  Message: As no ONLINE member has the missing data for recovering in its binary logs, this member will start distributed recovery using clone.

  `ER_GRP_RPL_RECOVERY_STRAT_CLONE_PURGED` was added in 8.0.17.

- Error number: `MY-013471`; Symbol: `ER_GRP_RPL_RECOVERY_STRAT_CHOICE`; SQLSTATE: `HY000`

  Message: Distributed recovery will transfer data using: %s

  `ER_GRP_RPL_RECOVERY_STRAT_CHOICE` was added in 8.0.17.

- Error number: `MY-013472`; Symbol: `ER_GRP_RPL_RECOVERY_STRAT_FALLBACK`; SQLSTATE: `HY000`

  Message: Due to some issue on the previous step distributed recovery is now executing: %s

  `ER_GRP_RPL_RECOVERY_STRAT_FALLBACK` was added in 8.0.17.

- Error number: `MY-013473`; Symbol: `ER_GRP_RPL_RECOVERY_STRAT_NO_FALLBACK`; SQLSTATE: `HY000`

  Message: Due to a critical cloning error or lack of donors, distributed recovery cannot be executed. The member will now leave the group.

  `ER_GRP_RPL_RECOVERY_STRAT_NO_FALLBACK` was added in 8.0.17.

- Error number: `MY-013474`; Symbol: `ER_GRP_RPL_SLAVE_THREAD_ERROR_ON_CLONE`; SQLSTATE: `HY000`

  Message: The '%s' thread of channel '%s' will error out as the server will attempt to clone another server

  `ER_GRP_RPL_SLAVE_THREAD_ERROR_ON_CLONE` was added in 8.0.17.

- Error number: `MY-013475`; Symbol: `ER_UNKNOWN_TABLE_IN_UPGRADE`; SQLSTATE: `HY000`

Message: Unknown table '%s'

ER_UNKNOWN_TABLE_IN_UPGRADE was added in 8.0.17.

- Error number: MY-013476; Symbol: ER_IDENT_CAUSES_TOO_LONG_PATH_IN_UPGRADE; SQLSTATE: HY000

  Message: Long database name and identifier for object resulted in path length exceeding %d characters. Path: '%s'.

  ER_IDENT_CAUSES_TOO_LONG_PATH_IN_UPGRADE was added in 8.0.17.

- Error number: MY-013477; Symbol: ER_XA_CANT_CREATE_MDL_BACKUP; SQLSTATE: HY000

  Message: XA: Failed to take MDL Lock backup of PREPARED XA transaction during client disconnect.

  ER_XA_CANT_CREATE_MDL_BACKUP was added in 8.0.13.

- Error number: MY-013478; Symbol: ER_AUDIT_LOG_SUPER_PRIVILEGE_REQUIRED; SQLSTATE: HY000

  Message: SUPER privilege or AUDIT_ADMIN role required for '%s'@'%s' user.

  ER_AUDIT_LOG_SUPER_PRIVILEGE_REQUIRED was added in 8.0.11.

- Error number: MY-013479; Symbol: ER_AUDIT_LOG_UDF_INVALID_ARGUMENT_TYPE; SQLSTATE: HY000

  Message: Invalid argument type

  ER_AUDIT_LOG_UDF_INVALID_ARGUMENT_TYPE was added in 8.0.11.

- Error number: MY-013480; Symbol: ER_AUDIT_LOG_UDF_INVALID_ARGUMENT_COUNT; SQLSTATE: HY000

  Message: Invalid argument count

  ER_AUDIT_LOG_UDF_INVALID_ARGUMENT_COUNT was added in 8.0.11.

- Error number: MY-013481; Symbol: ER_AUDIT_LOG_HAS_NOT_BEEN_INSTALLED; SQLSTATE: HY000

  Message: audit_log plugin has not been installed using INSTALL PLUGIN syntax.

  ER_AUDIT_LOG_HAS_NOT_BEEN_INSTALLED was added in 8.0.11.

- Error number: MY-013482; Symbol: ER_AUDIT_LOG_UDF_READ_INVALID_MAX_ARRAY_LENGTH_ARG_TYPE; SQLSTATE: HY000

  Message: Invalid "max_array_length" argument type.

  ER_AUDIT_LOG_UDF_READ_INVALID_MAX_ARRAY_LENGTH_ARG_TYPE was added in 8.0.11.

- Error number: MY-013483; Symbol: ER_LOG_CANNOT_WRITE_EXTENDED; SQLSTATE: HY000

  Message: Failed to write to %s: %s (%s)

  ER_LOG_CANNOT_WRITE_EXTENDED was added in 8.0.17.

- Error number: MY-013484; Symbol: ER_UPGRADE_WITH_PARTITIONED_TABLES_REJECTED; SQLSTATE: HY000

  Message: Upgrading from server version %d with partitioned tables and lower_case_table_names == 1 on a case sensitive file system may cause issues, and is therefore prohibited. To upgrade anyway, restart the new server version with the command line option 'upgrade=FORCE'. When upgrade is completed, please execute 'RENAME TABLE <part_table_name> TO <new_table_name>; RENAME TABLE <new_table_name> TO <part_table_name>;' for each of the partitioned tables. Please see the documentation for further information.

  ER_UPGRADE_WITH_PARTITIONED_TABLES_REJECTED was added in 8.0.17.

- Error number: MY-013485; Symbol: ER_KEYRING_AWS_INCORRECT_PROXY; SQLSTATE: HY000

  Message: Incorrect environment variable %s, invalid port: %s

  ER_KEYRING_AWS_INCORRECT_PROXY was added in 8.0.18.

- Error number: MY-013486; Symbol: ER_GRP_RPL_SERVER_SET_TO_OFFLINE_MODE_DUE_TO_ERRORS; SQLSTATE: HY000

  Message: The server was automatically set into offline mode after an error was detected.

  ER_GRP_RPL_SERVER_SET_TO_OFFLINE_MODE_DUE_TO_ERRORS was added in 8.0.18.

- Error number: MY-013487; Symbol: ER_GRP_RPL_MESSAGE_SERVICE_FATAL_ERROR; SQLSTATE: HY000

  Message: Message delivery error on message service of Group Replication. The server will now leave the group.

  ER_GRP_RPL_MESSAGE_SERVICE_FATAL_ERROR was added in 8.0.18.

- Error number: MY-013488; Symbol: ER_WARN_WRONG_COMPRESSION_ALGORITHM_LOG; SQLSTATE: HY000

  Message: Invalid MASTER_COMPRESSION_ALGORITHMS '%s' found in repository for channel '%s'. Resetting to 'uncompressed' (no compression).

  ER_WARN_WRONG_COMPRESSION_ALGORITHM_LOG was added in 8.0.18.

- Error number: MY-013489; Symbol: ER_WARN_WRONG_COMPRESSION_LEVEL_LOG; SQLSTATE: HY000

  Message: Invalid MASTER_ZSTD_COMPRESSION_LEVEL found in repository for channel '%s'. Resetting to %u.

  ER_WARN_WRONG_COMPRESSION_LEVEL_LOG was added in 8.0.18.

- Error number: MY-013490; Symbol: ER_PROTOCOL_COMPRESSION_RESET_LOG; SQLSTATE: HY000

  Message: Option --protocol-compression-algorithms is reset to default value.

  ER_PROTOCOL_COMPRESSION_RESET_LOG was added in 8.0.18.

- Error number: MY-013491; Symbol: ER_XPLUGIN_COMPRESSION_ERROR; SQLSTATE: HY000

  Message: Fatal error while compressing outgoing data - %s

ER_XPLUGIN_COMPRESSION_ERROR was added in 8.0.18.

- Error number: MY-013492; Symbol: ER_MYSQLBACKUP_MSG; SQLSTATE: HY000

  Message: %s

  ER_MYSQLBACKUP_MSG was added in 8.0.18.

- Error number: MY-013493; Symbol: ER_WARN_UNKNOWN_KEYRING_AWS_REGION; SQLSTATE: HY000

  Message: Unknown keyring_aws_region '%s'. Connection to AWS KMS may fail.

  ER_WARN_UNKNOWN_KEYRING_AWS_REGION was added in 8.0.18.

- Error number: MY-013494; Symbol: ER_WARN_LOG_PRIVILEGE_CHECKS_USER_DOES_NOT_EXIST; SQLSTATE: HY000

  Message: PRIVILEGE_CHECKS_USER for replication channel '%s' was set to `%s`@`%s`, but this is not an existing user. Correct this before starting replication threads.

  ER_WARN_LOG_PRIVILEGE_CHECKS_USER_DOES_NOT_EXIST was added in 8.0.18.

- Error number: MY-013495; Symbol: ER_WARN_LOG_PRIVILEGE_CHECKS_USER_CORRUPT; SQLSTATE: HY000

  Message: Invalid, corrupted PRIVILEGE_CHECKS_USER was found in the replication configuration repository for channel '%s'. Use CHANGE MASTER TO PRIVILEGE_CHECKS_USER to correct the configuration.

  ER_WARN_LOG_PRIVILEGE_CHECKS_USER_CORRUPT was added in 8.0.18.

- Error number: MY-013496; Symbol: ER_WARN_LOG_PRIVILEGE_CHECKS_USER_NEEDS_RPL_APPLIER_PRIV; SQLSTATE: HY000

  Message: PRIVILEGE_CHECKS_USER for replication channel '%s' was set to `%s`@`%s`, but this user does not have REPLICATION_APPLIER privilege. Correct this before starting the replication threads.

  ER_WARN_LOG_PRIVILEGE_CHECKS_USER_NEEDS_RPL_APPLIER_PRIV was added in 8.0.18.

- Error number: MY-013497; Symbol: ER_FILE_PRIVILEGE_FOR_REPLICATION_CHECKS; SQLSTATE: HY000

  Message: The PRIVILEGE_CHECKS_USER for channel '%s' would need FILE privilege to execute a LOAD DATA INFILE statement replicated in statement format. Consider using binlog_format=ROW on master and slave_allow_load_data_events=0 on this server. If the replicated events are trusted, recover from the failure by temporarily granting FILE to the PRIVILEGE_CHECKS_USER.

  ER_FILE_PRIVILEGE_FOR_REPLICATION_CHECKS was added in 8.0.18.

- Error number: MY-013498; Symbol: ER_RPL_SLAVE_SQL_THREAD_STARTING_WITH_PRIVILEGE_CHECKS; SQLSTATE: HY000

  Message: Slave SQL thread%s initialized, starting replication in log '%s' at position %s, relay log '%s' position: %s, user: '%s'@'%s', roles: %s

  ER_RPL_SLAVE_SQL_THREAD_STARTING_WITH_PRIVILEGE_CHECKS was added in 8.0.18.

- Error number: MY-013499; Symbol: ER_AUDIT_LOG_CANNOT_GENERATE_PASSWORD; SQLSTATE: HY000

  Message: Cannot generate password: '%s'

  ER_AUDIT_LOG_CANNOT_GENERATE_PASSWORD was added in 8.0.18.

- Error number: MY-013500; Symbol: ER_INIT_FAILED_TO_GENERATE_ROOT_PASSWORD; SQLSTATE: HY000

  Message: Failed to generate a random password for root. Probabably not enough enthropy.

  ER_INIT_FAILED_TO_GENERATE_ROOT_PASSWORD was added in 8.0.18.

- Error number: MY-013501; Symbol: ER_PLUGIN_LOAD_OPTIONS_IGNORED; SQLSTATE: HY000

  Message: Ignoring --plugin-load[_add] list as the server is running with --initialize(-insecure).

  ER_PLUGIN_LOAD_OPTIONS_IGNORED was added in 8.0.18.

- Error number: MY-013502; Symbol: ER_WARN_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES; SQLSTATE: HY000

  Message: Cannot set mandatory_roles: AuthId `%s`@`%s` has '%s' privilege. AuthId(s) set in the mandatory_roles are ignored.

  ER_WARN_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES was added in 8.0.18.

- Error number: MY-013503; Symbol: ER_IB_MSG_SKIP_HIDDEN_DIR; SQLSTATE: HY000

  Message: Directory '%s' will not be scanned because it is a hidden directory.

  ER_IB_MSG_SKIP_HIDDEN_DIR was added in 8.0.18.

- Error number: MY-013504; Symbol: ER_WARN_RPL_RECOVERY_NO_ROTATE_EVENT_FROM_MASTER_EOF; SQLSTATE: HY000

  Message: Server was not able to find a rotate event from master server to initialize relay log recovery for channel '%s'. Skipping relay log recovery for the channel.

  ER_WARN_RPL_RECOVERY_NO_ROTATE_EVENT_FROM_MASTER_EOF was added in 8.0.19.

- Error number: MY-013505; Symbol: ER_IB_LOB_ROLLBACK_INDEX_LEN; SQLSTATE: HY000

  Message: Rolling back LOB for transaction %llu undo number %llu : current index length %llu. (iteration %llu)

  ER_IB_LOB_ROLLBACK_INDEX_LEN was added in 8.0.19.

- Error number: MY-013506; Symbol: ER_CANT_PROCESS_EXPRESSION_FOR_GENERATED_COLUMN_TO_DD; SQLSTATE: HY000

  Message: Error in processing (possibly deprecated) expression or function '%s' for generated column %s.%s.%s

  ER_CANT_PROCESS_EXPRESSION_FOR_GENERATED_COLUMN_TO_DD was added in 8.0.19.

B.3.2 Client Error Message Reference

Client error messages originate from within the MySQL client library. Here is an example client error message, as displayed by the `mysql` client:

```
shell> mysql -h no-such-host
ERROR 2005 (HY000): Unknown MySQL server host 'no-such-host' (0)
```

Each client error message includes an error code, SQLSTATE value, and message string, as described in Section B.1, “Error Message Sources and Components”. These components are available as described in Section B.2, “Error Information Interfaces”. For client errors, the SQLSTATE value is always `'HY000'` (general error), so it is not meaningful for distinguishing one client error from another.

The client library also makes available to host client programs any errors that originate on the server side and are recieved by the client from the server. For a list of server-side errors, see Section B.3.1, “Server Error Message Reference”.

In addition to the errors in the following list, the client library can also produce error messages that have error codes in the range from 1 to 999. See Section B.3.3, “Global Error Message Reference”

- Error number: `2000`; Symbol: `CR_UNKNOWN_ERROR`;

  Message: Unknown MySQL error
- Error number: `2001`; Symbol: `CR_SOCKET_CREATE_ERROR`;

  Message: Can't create UNIX socket (%d)
- Error number: `2002`; Symbol: `CR_CONNECTION_ERROR`;

  Message: Can't connect to local MySQL server through socket '%s' (%d)
- Error number: `2003`; Symbol: `CR_CONN_HOST_ERROR`;

  Message: Can't connect to MySQL server on '%s' (%d)
- Error number: `2004`; Symbol: `CR_IPSOCK_ERROR`;

  Message: Can't create TCP/IP socket (%d)
- Error number: `2005`; Symbol: `CR_UNKNOWN_HOST`;

  Message: Unknown MySQL server host '%s' (%d)
- Error number: `2006`; Symbol: `CR_SERVER_GONE_ERROR`;

  Message: MySQL server has gone away
- Error number: `2007`; Symbol: `CR_VERSION_ERROR`;

  Message: Protocol mismatch; server version = %d, client version = %d
- Error number: `2008`; Symbol: `CR_OUT_OF_MEMORY`;

  Message: MySQL client ran out of memory
- Error number: `2009`; Symbol: `CR_WRONG_HOST_INFO`;

  Message: Wrong host info

- Error number: 2010; Symbol: CR_LOCALHOST_CONNECTION;

  Message: Localhost via UNIX socket

- Error number: 2011; Symbol: CR_TCP_CONNECTION;

  Message: %s via TCP/IP

- Error number: 2012; Symbol: CR_SERVER_HANDSHAKE_ERR;

  Message: Error in server handshake

- Error number: 2013; Symbol: CR_SERVER_LOST;

  Message: Lost connection to MySQL server during query

- Error number: 2014; Symbol: CR_COMMANDS_OUT_OF_SYNC;

  Message: Commands out of sync; you can't run this command now

  Commands were executed in an improper order. This error occurs when a function is called that is not appropriate for the current state of the connection. For example, if mysql_stmt_fetch() is not called enough times to read an entire result set (that is, enough times to return MYSQL_NO_DATA), this error may occur for the following C API call.

- Error number: 2015; Symbol: CR_NAMEDPIPE_CONNECTION;

  Message: Named pipe: %s

- Error number: 2016; Symbol: CR_NAMEDPIPEWAIT_ERROR;

  Message: Can't wait for named pipe to host: %s pipe: %s (%lu)

- Error number: 2017; Symbol: CR_NAMEDPIPEOPEN_ERROR;

  Message: Can't open named pipe to host: %s pipe: %s (%lu)

- Error number: 2018; Symbol: CR_NAMEDPIPESETSTATE_ERROR;

  Message: Can't set state of named pipe to host: %s pipe: %s (%lu)

- Error number: 2019; Symbol: CR_CANT_READ_CHARSET;

  Message: Can't initialize character set %s (path: %s)

- Error number: 2020; Symbol: CR_NET_PACKET_TOO_LARGE;

  Message: Got packet bigger than 'max_allowed_packet' bytes

- Error number: 2021; Symbol: CR_EMBEDDED_CONNECTION;

  Message: Embedded server

- Error number: 2022; Symbol: CR_PROBE_SLAVE_STATUS;

  Message: Error on SHOW SLAVE STATUS:

- Error number: 2023; Symbol: CR_PROBE_SLAVE_HOSTS;

  Message: Error on SHOW SLAVE HOSTS:

- Error number: 2024; Symbol: CR_PROBE_SLAVE_CONNECT;

  Message: Error connecting to slave:

- Error number: 2025; Symbol: CR_PROBE_MASTER_CONNECT;

  Message: Error connecting to master:

- Error number: 2026; Symbol: CR_SSL_CONNECTION_ERROR;

  Message: SSL connection error: %s

- Error number: 2027; Symbol: CR_MALFORMED_PACKET;

  Message: Malformed packet

- Error number: 2028; Symbol: CR_WRONG_LICENSE;

  Message: This client library is licensed only for use with MySQL servers having '%s' license

- Error number: 2029; Symbol: CR_NULL_POINTER;

  Message: Invalid use of null pointer

- Error number: 2030; Symbol: CR_NO_PREPARE_STMT;

  Message: Statement not prepared

- Error number: 2031; Symbol: CR_PARAMS_NOT_BOUND;

  Message: No data supplied for parameters in prepared statement

- Error number: 2032; Symbol: CR_DATA_TRUNCATED;

  Message: Data truncated

- Error number: 2033; Symbol: CR_NO_PARAMETERS_EXISTS;

  Message: No parameters exist in the statement

- Error number: 2034; Symbol: CR_INVALID_PARAMETER_NO;

  Message: Invalid parameter number

  The column number for mysql_stmt_fetch_column() was invalid.

  The parameter number for mysql_stmt_send_long_data() was invalid.

  A key name was empty or the amount of connection attribute data for mysql_options4() exceeds the 64KB limit.

- Error number: 2035; Symbol: CR_INVALID_BUFFER_USE;

  Message: Can't send long data for non-string/non-binary data types (parameter: %d)

- Error number: 2036; Symbol: CR_UNSUPPORTED_PARAM_TYPE;

  Message: Using unsupported buffer type: %d (parameter: %d)

- Error number: 2037; Symbol: CR_SHARED_MEMORY_CONNECTION;

Message: Shared memory: %s

- Error number: 2038; Symbol: CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR;

  Message: Can't open shared memory; client could not create request event (%lu)

- Error number: 2039; Symbol: CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR;

  Message: Can't open shared memory; no answer event received from server (%lu)

- Error number: 2040; Symbol: CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR;

  Message: Can't open shared memory; server could not allocate file mapping (%lu)

- Error number: 2041; Symbol: CR_SHARED_MEMORY_CONNECT_MAP_ERROR;

  Message: Can't open shared memory; server could not get pointer to file mapping (%lu)

- Error number: 2042; Symbol: CR_SHARED_MEMORY_FILE_MAP_ERROR;

  Message: Can't open shared memory; client could not allocate file mapping (%lu)

- Error number: 2043; Symbol: CR_SHARED_MEMORY_MAP_ERROR;

  Message: Can't open shared memory; client could not get pointer to file mapping (%lu)

- Error number: 2044; Symbol: CR_SHARED_MEMORY_EVENT_ERROR;

  Message: Can't open shared memory; client could not create %s event (%lu)

- Error number: 2045; Symbol: CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR;

  Message: Can't open shared memory; no answer from server (%lu)

- Error number: 2046; Symbol: CR_SHARED_MEMORY_CONNECT_SET_ERROR;

  Message: Can't open shared memory; cannot send request event to server (%lu)

- Error number: 2047; Symbol: CR_CONN_UNKNOW_PROTOCOL;

  Message: Wrong or unknown protocol

- Error number: 2048; Symbol: CR_INVALID_CONN_HANDLE;

  Message: Invalid connection handle

- Error number: 2049; Symbol: CR_UNUSED_1;

  Message: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)

- Error number: 2050; Symbol: CR_FETCH_CANCELED;

  Message: Row retrieval was canceled by mysql_stmt_close() call

- Error number: 2051; Symbol: CR_NO_DATA;

  Message: Attempt to read column without prior row fetch

- Error number: 2052; Symbol: CR_NO_STMT_METADATA;

  Message: Prepared statement contains no metadata

- Error number: 2053; Symbol: CR_NO_RESULT_SET;

  Message: Attempt to read a row while there is no result set associated with the statement

- Error number: 2054; Symbol: CR_NOT_IMPLEMENTED;

  Message: This feature is not implemented yet

- Error number: 2055; Symbol: CR_SERVER_LOST_EXTENDED;

  Message: Lost connection to MySQL server at '%s', system error: %d

- Error number: 2056; Symbol: CR_STMT_CLOSED;

  Message: Statement closed indirectly because of a preceding %s() call

- Error number: 2057; Symbol: CR_NEW_STMT_METADATA;

  Message: The number of columns in the result set differs from the number of bound buffers. You must reset the statement, rebind the result set columns, and execute the statement again

- Error number: 2058; Symbol: CR_ALREADY_CONNECTED;

  Message: This handle is already connected. Use a separate handle for each connection.

- Error number: 2059; Symbol: CR_AUTH_PLUGIN_CANNOT_LOAD;

  Message: Authentication plugin '%s' cannot be loaded: %s

- Error number: 2060; Symbol: CR_DUPLICATE_CONNECTION_ATTR;

  Message: There is an attribute with the same name already

  A duplicate connection attribute name was specified for mysql_options4().

- Error number: 2061; Symbol: CR_AUTH_PLUGIN_ERR;

  Message: Authentication plugin '%s' reported error: %s

- Error number: 2062; Symbol: CR_INSECURE_API_ERR;

  Message: Insecure API function call: '%s' Use instead: '%s'

  An insecure function call was detected. Modify the application to use the suggested alternative function instead.

- Error number: 2063; Symbol: CR_FILE_NAME_TOO_LONG;

  Message: File name is too long

  CR_FILE_NAME_TOO_LONG was added in 8.0.1.

- Error number: 2064; Symbol: CR_SSL_FIPS_MODE_ERR;

  Message: Set FIPS mode ON/STRICT failed

`CR_SSL_FIPS_MODE_ERR` was added in 8.0.11.

- Error number: `2065`; Symbol: `CR_COMPRESSION_NOT_SUPPORTED`;

  Message: Compression protocol not supported with asynchronous protocol

  `CR_COMPRESSION_NOT_SUPPORTED` was added in 8.0.16.

- Error number: `2066`; Symbol: `CR_COMPRESSION_WRONGLY_CONFIGURED`;

  Message: Connection failed due to wrongly configured compression algorithm

  `CR_COMPRESSION_WRONGLY_CONFIGURED` was added in 8.0.18.

B.3.3 Global Error Message Reference

This section lists “global” error messages that are shared in the sense that they can be produced by the MySQL server or by MySQL client programs. These errors have error codes in the range from 1 to 999.

Each global error message includes an error code, SQLSTATE value, and message string, as described in Section B.1, “Error Message Sources and Components”. These components are available as described in Section B.2, “Error Information Interfaces”. For global errors, the SQLSTATE value is always `'HY000'` (general error), so it is not meaningful for distinguishing one client error from another.

If the server writes a message to the error log that has a global error code, it pads the error code with leading zeros to six digits and adds a prefix of `MY-` (example: `MY-000022`). The following list displays error codes in that format.

If a client returns a message with a global error code, it adds no zero-padding or prefix to the error code.

- Error number: `1`; Symbol: `EE_CANTCREATEFILE`;

  Message: Can't create/write to file '%s' (OS errno %d - %s)

- Error number: `2`; Symbol: `EE_READ`;

  Message: Error reading file '%s' (OS errno %d - %s)

- Error number: `3`; Symbol: `EE_WRITE`;

  Message: Error writing file '%s' (OS errno %d - %s)

- Error number: `4`; Symbol: `EE_BADCLOSE`;

  Message: Error on close of '%s' (OS errno %d - %s)

- Error number: `5`; Symbol: `EE_OUTOFMEMORY`;

  Message: Out of memory (Needed %u bytes)

- Error number: `6`; Symbol: `EE_DELETE`;

  Message: Error on delete of '%s' (OS errno %d - %s)

- Error number: `7`; Symbol: `EE_LINK`;

  Message: Error on rename of '%s' to '%s' (OS errno %d - %s)

- Error number: 9; Symbol: EE_EOFERR;

  Message: Unexpected EOF found when reading file '%s' (OS errno %d - %s)

- Error number: 10; Symbol: EE_CANTLOCK;

  Message: Can't lock file (OS errno %d - %s)

- Error number: 11; Symbol: EE_CANTUNLOCK;

  Message: Can't unlock file (OS errno %d - %s)

- Error number: 12; Symbol: EE_DIR;

  Message: Can't read dir of '%s' (OS errno %d - %s)

- Error number: 13; Symbol: EE_STAT;

  Message: Can't get stat of '%s' (OS errno %d - %s)

- Error number: 14; Symbol: EE_CANT_CHSIZE;

  Message: Can't change size of file (OS errno %d - %s)

- Error number: 15; Symbol: EE_CANT_OPEN_STREAM;

  Message: Can't open stream from handle (OS errno %d - %s)

- Error number: 16; Symbol: EE_GETWD;

  Message: Can't get working directory (OS errno %d - %s)

- Error number: 17; Symbol: EE_SETWD;

  Message: Can't change dir to '%s' (OS errno %d - %s)

- Error number: 18; Symbol: EE_LINK_WARNING;

  Message: Warning: '%s' had %d links

- Error number: 19; Symbol: EE_OPEN_WARNING;

  Message: Warning: %d files and %d streams are left open

- Error number: 20; Symbol: EE_DISK_FULL;

  Message: Disk is full writing '%s' (OS errno %d - %s). Waiting for someone to free space...

- Error number: 21; Symbol: EE_CANT_MKDIR;

  Message: Can't create directory '%s' (OS errno %d - %s)

- Error number: 22; Symbol: EE_UNKNOWN_CHARSET;

  Message: Character set '%s' is not a compiled character set and is not specified in the '%s' file

- Error number: 23; Symbol: EE_OUT_OF_FILERESOURCES;

  Message: Out of resources when opening file '%s' (OS errno %d - %s)

- Error number: 24; Symbol: EE_CANT_READLINK;

  Message: Can't read value for symlink '%s' (Error %d - %s)

- Error number: 25; Symbol: EE_CANT_SYMLINK;

  Message: Can't create symlink '%s' pointing at '%s' (Error %d - %s)

- Error number: 26; Symbol: EE_REALPATH;

  Message: Error on realpath() on '%s' (Error %d - %s)

- Error number: 27; Symbol: EE_SYNC;

  Message: Can't sync file '%s' to disk (OS errno %d - %s)

- Error number: 28; Symbol: EE_UNKNOWN_COLLATION;

  Message: Collation '%s' is not a compiled collation and is not specified in the '%s' file

- Error number: 29; Symbol: EE_FILENOTFOUND;

  Message: File '%s' not found (OS errno %d - %s)

- Error number: 30; Symbol: EE_FILE_NOT_CLOSED;

  Message: File '%s' (fileno: %d) was not closed

- Error number: 31; Symbol: EE_CHANGE_OWNERSHIP;

  Message: Cannot change ownership of the file '%s' (OS errno %d - %s)

- Error number: 32; Symbol: EE_CHANGE_PERMISSIONS;

  Message: Cannot change permissions of the file '%s' (OS errno %d - %s)

- Error number: 33; Symbol: EE_CANT_SEEK;

  Message: Cannot seek in file '%s' (OS errno %d - %s)

- Error number: 34; Symbol: EE_CAPACITY_EXCEEDED;

  Message: Memory capacity exceeded (capacity %llu bytes)

- Error number: 35; Symbol: EE_DISK_FULL_WITH_RETRY_MSG;

  Message: Disk is full writing '%s' (OS errno %d - %s). Waiting for someone to free space... Retry in %d secs. Message reprinted in %d secs.

  EE_DISK_FULL_WITH_RETRY_MSG was added in 8.0.13.

- Error number: 36; Symbol: EE_FAILED_TO_CREATE_TIMER;

  Message: Failed to create timer (OS errno %d).

  EE_FAILED_TO_CREATE_TIMER was added in 8.0.13.

- Error number: 37; Symbol: EE_FAILED_TO_DELETE_TIMER;

  Message: Failed to delete timer (OS errno %d).

EE_FAILED_TO_DELETE_TIMER was added in 8.0.13.

- Error number: 38; Symbol: EE_FAILED_TO_CREATE_TIMER_QUEUE;

  Message: Failed to create timer queue (OS errno %d).

  EE_FAILED_TO_CREATE_TIMER_QUEUE was added in 8.0.13.

- Error number: 39; Symbol: EE_FAILED_TO_START_TIMER_NOTIFY_THREAD;

  Message: Failed to start timer notify thread.

  EE_FAILED_TO_START_TIMER_NOTIFY_THREAD was added in 8.0.13.

- Error number: 40; Symbol: EE_FAILED_TO_CREATE_TIMER_NOTIFY_THREAD_INTERRUPT_EVENT;

  Message: Failed to create event to interrupt timer notifier thread (OS errno %d).

  EE_FAILED_TO_CREATE_TIMER_NOTIFY_THREAD_INTERRUPT_EVENT was added in 8.0.13.

- Error number: 41; Symbol: EE_EXITING_TIMER_NOTIFY_THREAD;

  Message: Failed to register timer event with queue (OS errno %d), exiting timer notifier thread.

  EE_EXITING_TIMER_NOTIFY_THREAD was added in 8.0.13.

- Error number: 42; Symbol: EE_WIN_LIBRARY_LOAD_FAILED;

  Message: LoadLibrary("kernel32.dll") failed: GetLastError returns %lu.

  EE_WIN_LIBRARY_LOAD_FAILED was added in 8.0.13.

- Error number: 43; Symbol: EE_WIN_RUN_TIME_ERROR_CHECK;

  Message: %s.

  EE_WIN_RUN_TIME_ERROR_CHECK was added in 8.0.13.

- Error number: 44; Symbol: EE_FAILED_TO_DETERMINE_LARGE_PAGE_SIZE;

  Message: Failed to determine large page size.

  EE_FAILED_TO_DETERMINE_LARGE_PAGE_SIZE was added in 8.0.13.

- Error number: 45; Symbol: EE_FAILED_TO_KILL_ALL_THREADS;

  Message: Error in my_thread_global_end(): %d thread(s) did not exit.

  EE_FAILED_TO_KILL_ALL_THREADS was added in 8.0.13.

- Error number: 46; Symbol: EE_FAILED_TO_CREATE_IO_COMPLETION_PORT;

  Message: Failed to create IO completion port (OS errno %d).

  EE_FAILED_TO_CREATE_IO_COMPLETION_PORT was added in 8.0.13.

- Error number: 47; Symbol: EE_FAILED_TO_OPEN_DEFAULTS_FILE;

  Message: Failed to open required defaults file: %s

EE_FAILED_TO_OPEN_DEFAULTS_FILE was added in 8.0.13.

- Error number: 48; Symbol: EE_FAILED_TO_HANDLE_DEFAULTS_FILE;

  Message: Fatal error in defaults handling. Program aborted!

  EE_FAILED_TO_HANDLE_DEFAULTS_FILE was added in 8.0.13.

- Error number: 49; Symbol: EE_WRONG_DIRECTIVE_IN_CONFIG_FILE;

  Message: Wrong '!%s' directive in config file %s at line %d.

  EE_WRONG_DIRECTIVE_IN_CONFIG_FILE was added in 8.0.13.

- Error number: 50; Symbol: EE_SKIPPING_DIRECTIVE_DUE_TO_MAX_INCLUDE_RECURSION;

  Message: Skipping '%s' directive as maximum include recursion level was reached in file %s at line %d.

  EE_SKIPPING_DIRECTIVE_DUE_TO_MAX_INCLUDE_RECURSION was added in 8.0.13.

- Error number: 51; Symbol: EE_INCORRECT_GRP_DEFINITION_IN_CONFIG_FILE;

  Message: Wrong group definition in config file %s at line %d.

  EE_INCORRECT_GRP_DEFINITION_IN_CONFIG_FILE was added in 8.0.13.

- Error number: 52; Symbol: EE_OPTION_WITHOUT_GRP_IN_CONFIG_FILE;

  Message: Found option without preceding group in config file %s at line %d.

  EE_OPTION_WITHOUT_GRP_IN_CONFIG_FILE was added in 8.0.13.

- Error number: 53; Symbol: EE_CONFIG_FILE_PERMISSION_ERROR;

  Message: %s should be readable/writable only by current user.

  EE_CONFIG_FILE_PERMISSION_ERROR was added in 8.0.13.

- Error number: 54; Symbol: EE_IGNORE_WORLD_WRITABLE_CONFIG_FILE;

  Message: World-writable config file '%s' is ignored.

  EE_IGNORE_WORLD_WRITABLE_CONFIG_FILE was added in 8.0.13.

- Error number: 55; Symbol: EE_USING_DISABLED_OPTION;

  Message: %s: Option '%s' was used, but is disabled.

  EE_USING_DISABLED_OPTION was added in 8.0.13.

- Error number: 56; Symbol: EE_USING_DISABLED_SHORT_OPTION;

  Message: %s: Option '-%c' was used, but is disabled.

  EE_USING_DISABLED_SHORT_OPTION was added in 8.0.13.

- Error number: 57; Symbol: EE_USING_PASSWORD_ON_CLI_IS_INSECURE;

  Message: Using a password on the command line interface can be insecure.

EE_USING_PASSWORD_ON_CLI_IS_INSECURE was added in 8.0.13.

- Error number: 58; Symbol: EE_UNKNOWN_SUFFIX_FOR_VARIABLE;

  Message: Unknown suffix '%c' used for variable '%s' (value '%s').

  EE_UNKNOWN_SUFFIX_FOR_VARIABLE was added in 8.0.13.

- Error number: 59; Symbol: EE_SSL_ERROR_FROM_FILE;

  Message: SSL error: %s from '%s'.

  EE_SSL_ERROR_FROM_FILE was added in 8.0.13.

- Error number: 60; Symbol: EE_SSL_ERROR;

  Message: SSL error: %s.

  EE_SSL_ERROR was added in 8.0.13.

- Error number: 61; Symbol: EE_NET_SEND_ERROR_IN_BOOTSTRAP;

  Message: %d %s.

  EE_NET_SEND_ERROR_IN_BOOTSTRAP was added in 8.0.13.

- Error number: 62; Symbol: EE_PACKETS_OUT_OF_ORDER;

  Message: Packets out of order (found %u, expected %u).

  EE_PACKETS_OUT_OF_ORDER was added in 8.0.13.

- Error number: 63; Symbol: EE_UNKNOWN_PROTOCOL_OPTION;

  Message: Unknown option to protocol: %s.

  EE_UNKNOWN_PROTOCOL_OPTION was added in 8.0.13.

- Error number: 64; Symbol: EE_FAILED_TO_LOCATE_SERVER_PUBLIC_KEY;

  Message: Failed to locate server public key '%s'.

  EE_FAILED_TO_LOCATE_SERVER_PUBLIC_KEY was added in 8.0.13.

- Error number: 65; Symbol: EE_PUBLIC_KEY_NOT_IN_PEM_FORMAT;

  Message: Public key is not in Privacy Enhanced Mail format: '%s'.

  EE_PUBLIC_KEY_NOT_IN_PEM_FORMAT was added in 8.0.13.

- Error number: 66; Symbol: EE_DEBUG_INFO;

  Message: %s.

  EE_DEBUG_INFO was added in 8.0.13.

- Error number: 67; Symbol: EE_UNKNOWN_VARIABLE;

  Message: unknown variable '%s'.

EE_UNKNOWN_VARIABLE was added in 8.0.13.

- Error number: 68; Symbol: EE_UNKNOWN_OPTION;

  Message: unknown option '--%s'.

  EE_UNKNOWN_OPTION was added in 8.0.13.

- Error number: 69; Symbol: EE_UNKNOWN_SHORT_OPTION;

  Message: %s: unknown option '-%c'.

  EE_UNKNOWN_SHORT_OPTION was added in 8.0.13.

- Error number: 70; Symbol: EE_OPTION_WITHOUT_ARGUMENT;

  Message: %s: option '--%s' cannot take an argument.

  EE_OPTION_WITHOUT_ARGUMENT was added in 8.0.13.

- Error number: 71; Symbol: EE_OPTION_REQUIRES_ARGUMENT;

  Message: %s: option '--%s' requires an argument.

  EE_OPTION_REQUIRES_ARGUMENT was added in 8.0.13.

- Error number: 72; Symbol: EE_SHORT_OPTION_REQUIRES_ARGUMENT;

  Message: %s: option '-%c' requires an argument.

  EE_SHORT_OPTION_REQUIRES_ARGUMENT was added in 8.0.13.

- Error number: 73; Symbol: EE_OPTION_IGNORED_DUE_TO_INVALID_VALUE;

  Message: %s: ignoring option '--%s' due to invalid value '%s'.

  EE_OPTION_IGNORED_DUE_TO_INVALID_VALUE was added in 8.0.13.

- Error number: 74; Symbol: EE_OPTION_WITH_EMPTY_VALUE;

  Message: %s: Empty value for '%s' specified.

  EE_OPTION_WITH_EMPTY_VALUE was added in 8.0.13.

- Error number: 75; Symbol: EE_FAILED_TO_ASSIGN_MAX_VALUE_TO_OPTION;

  Message: %s: Maximum value of '%s' cannot be set.

  EE_FAILED_TO_ASSIGN_MAX_VALUE_TO_OPTION was added in 8.0.13.

- Error number: 76; Symbol: EE_INCORRECT_BOOLEAN_VALUE_FOR_OPTION;

  Message: option '%s': boolean value '%s' was not recognized. Set to OFF.

  EE_INCORRECT_BOOLEAN_VALUE_FOR_OPTION was added in 8.0.13.

- Error number: 77; Symbol: EE_FAILED_TO_SET_OPTION_VALUE;

  Message: %s: Error while setting value '%s' to '%s'.

EE_FAILED_TO_SET_OPTION_VALUE was added in 8.0.13.

- Error number: 78; Symbol: EE_INCORRECT_INT_VALUE_FOR_OPTION;

  Message: Incorrect integer value: '%s'.

  EE_INCORRECT_INT_VALUE_FOR_OPTION was added in 8.0.13.
- Error number: 79; Symbol: EE_INCORRECT_UINT_VALUE_FOR_OPTION;

  Message: Incorrect unsigned integer value: '%s'.

  EE_INCORRECT_UINT_VALUE_FOR_OPTION was added in 8.0.13.
- Error number: 80; Symbol: EE_ADJUSTED_SIGNED_VALUE_FOR_OPTION;

  Message: option '%s': signed value %s adjusted to %s.

  EE_ADJUSTED_SIGNED_VALUE_FOR_OPTION was added in 8.0.13.
- Error number: 81; Symbol: EE_ADJUSTED_UNSIGNED_VALUE_FOR_OPTION;

  Message: option '%s': unsigned value %s adjusted to %s.

  EE_ADJUSTED_UNSIGNED_VALUE_FOR_OPTION was added in 8.0.13.
- Error number: 82; Symbol: EE_ADJUSTED_ULONGLONG_VALUE_FOR_OPTION;

  Message: option '%s': value %s adjusted to %s.

  EE_ADJUSTED_ULONGLONG_VALUE_FOR_OPTION was added in 8.0.13.
- Error number: 83; Symbol: EE_ADJUSTED_DOUBLE_VALUE_FOR_OPTION;

  Message: option '%s': value %g adjusted to %g.

  EE_ADJUSTED_DOUBLE_VALUE_FOR_OPTION was added in 8.0.13.
- Error number: 84; Symbol: EE_INVALID_DECIMAL_VALUE_FOR_OPTION;

  Message: Invalid decimal value for option '%s'.

  EE_INVALID_DECIMAL_VALUE_FOR_OPTION was added in 8.0.13.
- Error number: 85; Symbol: EE_COLLATION_PARSER_ERROR;

  Message: %s.

  EE_COLLATION_PARSER_ERROR was added in 8.0.13.
- Error number: 86; Symbol: EE_FAILED_TO_RESET_BEFORE_PRIMARY_IGNORABLE_CHAR;

  Message: Failed to reset before a primary ignorable character %s.

  EE_FAILED_TO_RESET_BEFORE_PRIMARY_IGNORABLE_CHAR was added in 8.0.13.
- Error number: 87; Symbol: EE_FAILED_TO_RESET_BEFORE_TERTIARY_IGNORABLE_CHAR;

  Message: Failed to reset before a tertiary ignorable character %s.

EE_FAILED_TO_RESET_BEFORE_TERTIARY_IGNORABLE_CHAR was added in 8.0.13.

- Error number: 88; Symbol: EE_SHIFT_CHAR_OUT_OF_RANGE;

  Message: Shift character out of range: %s.

  EE_SHIFT_CHAR_OUT_OF_RANGE was added in 8.0.13.

- Error number: 89; Symbol: EE_RESET_CHAR_OUT_OF_RANGE;

  Message: Reset character out of range: %s.

  EE_RESET_CHAR_OUT_OF_RANGE was added in 8.0.13.

- Error number: 90; Symbol: EE_UNKNOWN_LDML_TAG;

  Message: Unknown LDML tag: '%.*s'.

  EE_UNKNOWN_LDML_TAG was added in 8.0.13.

- Error number: 91; Symbol: EE_FAILED_TO_RESET_BEFORE_SECONDARY_IGNORABLE_CHAR;

  Message: Failed to reset before a secondary ignorable character %s.

  EE_FAILED_TO_RESET_BEFORE_SECONDARY_IGNORABLE_CHAR was added in 8.0.16.

B.4 Problems and Common Errors

This section lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

B.4.1 How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:

  - The keyboard does not work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light does not change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)
  - The mouse pointer does not move.
  - The machine does not answer to a remote machine's pings.
  - Other programs that are not related to MySQL do not behave correctly.
  - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

  In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as glibc) are up to date.

It is always good to use a machine with ECC memory to discover memory problems early.

- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See Section 5.4, “MySQL Server Logs”.
- If you do not think you have hardware problems, you should try to find out which program is causing problems. Try using `top`, `ps`, Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use `top`, `df`, or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it does not want to die, there is probably a bug in the operating system.

If you have examined all other possibilities and concluded that the MySQL server or a MySQL client is causing the problem, it is time to create a bug report, see Section 1.7, “How to Report Bugs or Problems”. In the bug report, try to give a complete description of how the system is behaving and what you think is happening. Also state why you think that MySQL is causing the problem. Take into consideration all the situations described in this chapter. State any problems exactly how they appear when you examine your system. Use the “copy and paste” method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only “the system does not work.” This provides us with no information about what could be the problem.

If a program fails, it is always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with `top`. Let the program run for a while, it may simply be evaluating something computationally intensive.
- If the `mysqld` server is causing problems, can you get any response from it with `mysqladmin -u root ping` or `mysqladmin -u root processlist`?
- What does a client program say when you try to connect to the MySQL server? (Try with `mysql`, for example.) Does the client jam? Do you get any output from the program?

When sending a bug report, you should follow the outline described in Section 1.7, “How to Report Bugs or Problems”.

B.4.2 Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

B.4.2.1 Access denied

An `Access denied` error can have many causes. Often the problem is related to the MySQL accounts that the server permits client programs to use when connecting. See Section 6.2, “Access Control and Account Management”, and Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”.

B.4.2.2 Can't connect to [local] MySQL server

A MySQL client on Unix can connect to the `mysqld` server in two different ways: By using a Unix socket file to connect through a file in the file system (default `/tmp/mysql.sock`), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you do not specify a host name or if you specify the special host name `localhost`.

If the MySQL server is running on Windows, you can connect using TCP/IP. If the server is started with the `named_pipe` system variable enabled, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is `MySQL` by default. If you do not give a host name when connecting to `mysqld`, a MySQL client first tries to connect to the named pipe. If that does not work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using `.` as the host name.

The error (2002) `Can't connect to ...` normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket file name or TCP/IP port number when trying to connect to the server. You should also check that the TCP/IP port you are using has not been blocked by a firewall or port blocking service.

The error (2003) `Can't connect to MySQL server on '`*`server`*`' (10061)` indicates that the network connection has been refused. You should check that there is a MySQL server running, that it has network connections enabled, and that the network port you specified is the one configured on the server.

Start by checking whether there is a process named `mysqld` running on your server host. (Use `ps xa | grep mysqld` on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See Section 2.10.2, “Starting the Server”.

If a `mysqld` process is running, you can check it by trying the following commands. The port number or Unix socket file name might be different in your setup. `host_ip` represents the IP address of the machine where the server is running.

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h `hostname` version variables
shell> mysqladmin -h `hostname` --port=3306 version
shell> mysqladmin -h host_ip version
shell> mysqladmin --protocol=SOCKET --socket=/tmp/mysql.sock version
```

Note the use of backticks rather than forward quotation marks with the `hostname` command; these cause the output of `hostname` (that is, the current host name) to be substituted into the `mysqladmin` command. If you have no `hostname` command or are running on Windows, you can manually type the host name of your machine (without backticks) following the `-h` option. You can also try `-h 127.0.0.1` to connect with TCP/IP to the local host.

Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with `--skip-networking`, it will not accept TCP/IP connections at all. If the server was started with the `bind_address` system variable set to `127.0.0.1`, it will listen for TCP/IP connections only locally on the loopback interface and will not accept remote connections.

Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or Windows Firewall may need to be configured not to block the MySQL port.

Here are some reasons the `Can't connect to local MySQL server` error might occur:

- `mysqld` is not running on the local host. Check your operating system's process list to ensure the `mysqld` process is present.
- You're running a MySQL server on Windows with many TCP/IP connections to it. If you're experiencing that quite often your clients get that error, you can find a workaround here: Connection to MySQL Server Failing on Windows.
- Someone has removed the Unix socket file that `mysqld` uses (`/tmp/mysql.sock` by default). For example, you might have a `cron` job that removes old files from the `/tmp` directory. You can always run `mysqladmin version` to check whether the Unix socket file that `mysqladmin` is trying to use really exists. The fix in this case is to change the `cron` job to not remove `mysql.sock` or to place the socket file somewhere else. See Section B.4.3.6, “How to Protect or Change the MySQL Unix Socket File”.
- You have started the `mysqld` server with the `--socket=/path/to/socket` option, but forgotten to tell client programs the new name of the socket file. If you change the socket path name for the server, you must also notify the MySQL clients. You can do this by providing the same `--socket` option when you run client programs. You also need to ensure that clients have permission to access the `mysql.sock` file. To find out where the socket file is, you can do:

  ```
  shell> netstat -ln | grep mysql
  ```

  See Section B.4.3.6, “How to Protect or Change the MySQL Unix Socket File”.
- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other `mysqld` threads (for example, with `kill`) before you can restart the MySQL server. See Section B.4.3.3, “What to Do If MySQL Keeps Crashing”.
- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or socket file so that the server and clients can access them, or restart `mysqld` with a `--socket` option that specifies a socket file name in a directory where the server can create it and where client programs can access it.

If you get the error message `Can't connect to MySQL server on some_host`, you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing `telnet some_host 3306` and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.
- If the server is running on the local host, try using `mysqladmin -h localhost variables` to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the `port` variable.)
- If you are running under Linux and Security-Enhanced Linux (SELinux) is enabled, make sure you have disabled SELinux protection for the `mysqld` process.

Connection to MySQL Server Failing on Windows

When you're running a MySQL server on Windows with many TCP/IP connections to it, and you're experiencing that quite often your clients get a `Can't connect to MySQL server` error, the reason might be that Windows does not allow for enough ephemeral (short-lived) ports to serve those connections.

The purpose of TIME_WAIT is to keep a connection accepting packets even after the connection has been closed. This is because Internet routing can cause a packet to take a slow route to its destination and it may arrive after both sides have agreed to close. If the port is in use for a new connection, that packet from the old connection could break the protocol or compromise personal information from the original connection. The TIME_WAIT delay prevents this by ensuring that the port cannot be reused until after some time has been permitted for those delayed packets to arrive.

It is safe to reduce TIME_WAIT greatly on LAN connections because there is little chance of packets arriving at very long delays, as they could through the Internet with its comparatively large distances and latencies.

Windows permits ephemeral (short-lived) TCP ports to the user. After any port is closed it will remain in a TIME_WAIT status for 120 seconds. The port will not be available again until this time expires. The default range of port numbers depends on the version of Windows, with a more limited number of ports in older versions:

- Windows through Server 2003: Ports in range 1025–5000
- Windows Vista, Server 2008, and newer: Ports in range 49152–65535

With a small stack of available TCP ports (5000) and a high number of TCP ports being open and closed over a short period of time along with the TIME_WAIT status you have a good chance for running out of ports. There are two ways to address this problem:

- Reduce the number of TCP ports consumed quickly by investigating connection pooling or persistent connections where possible
- Tune some settings in the Windows registry (see below)

Important

The following procedure involves modifying the Windows registry. Before you modify the registry, make sure to back it up and make sure that you understand how to restore it if a problem occurs. For information about how to back up, restore, and edit the registry, view the following article in the Microsoft Knowledge Base: http://support.microsoft.com/kb/256986/EN-US/.

1. Start Registry Editor (Regedt32.exe).
2. Locate the following key in the registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

3. On the Edit menu, click Add Value, and then add the following registry value:

```
Value Name: MaxUserPort
Data Type: REG_DWORD
Value: 65534
```

   This sets the number of ephemeral ports available to any user. The valid range is between 5000 and 65534 (decimal). The default value is 0x1388 (5000 decimal).

4. On the Edit menu, click Add Value, and then add the following registry value:

```
Value Name: TcpTimedWaitDelay
Data Type: REG_DWORD
```

```
Value: 30
```

This sets the number of seconds to hold a TCP port connection in `TIME_WAIT` state before closing. The valid range is between 30 and 300 decimal, although you may wish to check with Microsoft for the latest permitted values. The default value is 0x78 (120 decimal).

5. Quit Registry Editor.

6. Reboot the machine.

Note: Undoing the above should be as simple as deleting the registry entries you've created.

B.4.2.3 Lost connection to MySQL server

There are three likely causes for this error message.

Usually it indicates network connectivity trouble and you should check the condition of your network if this error occurs frequently. If the error message includes “during query,” this is probably the case you are experiencing.

Sometimes the “during query” form happens when millions of rows are being sent as part of one or more queries. If you know that this is happening, you should try increasing `net_read_timeout` from its default of 30 seconds to 60 seconds or longer, sufficient for the data transfer to complete.

More rarely, it can happen when the client is attempting the initial connection to the server. In this case, if your `connect_timeout` value is set to only a few seconds, you may be able to resolve the problem by increasing it to ten seconds, perhaps more if you have a very long distance or slow connection. You can determine whether you are experiencing this more uncommon cause by using `SHOW GLOBAL STATUS LIKE 'Aborted_connects'`. It will increase by one for each initial connection attempt that the server aborts. You may see “reading authorization packet” as part of the error message; if so, that also suggests that this is the solution that you need.

If the cause is none of those just described, you may be experiencing a problem with `BLOB` values that are larger than `max_allowed_packet`, which can cause this error with some clients. Sometime you may see an `ER_NET_PACKET_TOO_LARGE` error, and that confirms that you need to increase `max_allowed_packet`.

B.4.2.4 Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a `--password` or `-p` option that has no following password value:

```
shell> mysql -u user_name -p
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the `Enter password:` prompt. This occurs when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

B.4.2.5 Host 'host_name' is blocked

If the following error occurs, it means that `mysqld` has received many connection requests from the given host that were interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

The value of the `max_connect_errors` system variable determines how many successive interrupted connection requests are permitted. After `max_connect_errors` failed requests without a successful connection, `mysqld` assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connections until you flush the host cache by executing a `FLUSH HOSTS` statement, a `TRUNCATE TABLE` statement that truncates the Performance Schema `host_cache` table, or a `mysqladmin flush-hosts` command.

To adjust the permitted number of successive connection errors, set `max_connect_errors` at server startup. For example, put these lines in the server `my.cnf` file:

```
[mysqld]
max_connect_errors=10000
```

The value can also be set at runtime:

```
SET GLOBAL max_connect_errors=10000;
```

If you get the `Host 'host_name' is blocked` error message for a given host, you should first verify that there is nothing wrong with TCP/IP connections from that host. If you are having network problems, it does no good to increase the value of `max_connect_errors`.

For more information about how the host cache works, see Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”.

B.4.2.6 Too many connections

If clients encounter `Too many connections` errors when attempting to connect to the `mysqld` server, all available connections are in use by other clients.

The permitted number of connections is controlled by the `max_connections` system variable. The default value is 151 to improve performance when MySQL is used with the Apache Web server. To support more connections, set `max_connections` to a larger value.

`mysqld` actually permits `max_connections` + 1 client connections. The extra connection is reserved for use by accounts that have the `CONNECTION_ADMIN` or `SUPER` privilege. By granting the privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See Section 13.7.7.29, “SHOW PROCESSLIST Syntax”.

The server also permits administrative connections on a dedicated interface. For more information about how the server handles client connections, see Section 8.12.4.1, “How MySQL Handles Client Connections”.

B.4.2.7 Out of memory

If you issue a query using the `mysql` client program and receive an error like the following one, it means that `mysql` does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke `mysql` with the `--quick` option. This causes it to use the `mysql_use_result()` C API function to retrieve the result set, which places less of a load on the client (but more on the server).

B.4.2.8 MySQL server has gone away

This section also covers the related `Lost connection to server during query` error.

The most common reason for the `MySQL server has gone away` error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent).

Error Code	Description
`CR_SERVER_GONE_ERROR`	The client couldn't send a question to the server.
`CR_SERVER_LOST`	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the `wait_timeout` variable when you start `mysqld`. See Section 5.1.8, “Server System Variables”.

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the `mysql` command-line client).

Some other common reasons for the `MySQL server has gone away` error are:

- You (or the db administrator) has killed the running thread with a `KILL` statement or a `mysqladmin kill` command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- A client application running on a different host does not have the necessary privileges to connect to the MySQL server from that host.
- You got a timeout from the TCP/IP connection on the client side. This may happen if you have been using the commands: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)` or `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)`. In this case increasing the timeout may help solve the problem.
- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the `reconnect` flag in the `MYSQL` structure is equal to 0).
- You are using a Windows client and the server had dropped the connection (probably because `wait_timeout` expired) before the command was issued.

  The problem on Windows is that in some cases MySQL does not get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from the connection.

  The solution to this is to either do a `mysql_ping()` on the connection if there has been a long time since the last query (this is what Connector/ODBC does) or set `wait_timeout` on the `mysqld` server so high that it in practice never times out.

- You can also get these errors if you send a query to the server that is incorrect or too large. If `mysqld` receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big `BLOB` columns), you can increase the query limit by setting the server's `max_allowed_packet` variable, which has a default value of 64MB. You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in Section B.4.2.9, “Packet Too Large”.

  An `INSERT` or `REPLACE` statement that inserts a great many rows can also cause these sorts of errors. Either one of these statements sends a single request to the server irrespective of the number of rows to be inserted; thus, you can often avoid the error by reducing the number of rows sent per `INSERT` or `REPLACE`.

- It is also possible to see this error if host name lookups fail (for example, if the DNS server on which your server or network relies goes down). This is because MySQL is dependent on the host system for name resolution, but has no way of knowing whether it is working—from MySQL's point of view the problem is indistinguishable from any other network timeout.

  You may also see the `MySQL server has gone away` error if MySQL is started with the `--skip-networking` option.

  Another networking issue that can cause this error occurs if the MySQL port (default 3306) is blocked by your firewall, thus preventing any connections at all to the MySQL server.

- You can also encounter this error with applications that fork child processes, all of which try to use the same connection to the MySQL server. This can be avoided by using a separate connection for each child process.

- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing `mysqladmin version` and examining the server's uptime. If the client connection was broken because `mysqld` crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See Section B.4.3.3, “What to Do If MySQL Keeps Crashing”.

You can obtain more information about lost connections by starting `mysqld` with the `log_error_verbosity` system variable set to 3. This logs some of the disconnection messages in the `hostname.err` file. See Section 5.4.2, “The Error Log”.

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether the MySQL server died. You can find information about this in the server error log. See Section B.4.3.3, “What to Do If MySQL Keeps Crashing”.

- If a specific query kills `mysqld` and the tables involved were checked with `CHECK TABLE` before you ran the query, can you provide a reproducible test case? See Section 29.5, “Debugging and Porting MySQL”.

- What is the value of the `wait_timeout` system variable in the MySQL server? (`mysqladmin variables` gives you the value of this variable.)

- Have you tried to run `mysqld` with the general query log enabled to determine whether the problem query appears in the log? (See Section 5.4.3, “The General Query Log”.)

See also Section B.4.2.10, “Communication Errors and Aborted Connections”, and Section 1.7, “How to Report Bugs or Problems”.

B.4.2.9 Packet Too Large

A communication packet is a single SQL statement sent to the MySQL server, a single row that is sent to the client, or a binary log event sent from a master replication server to a slave.

The largest possible packet that can be transmitted to or from a MySQL 8.0 server or client is 1GB.

When a MySQL client or the `mysqld` server receives a packet bigger than `max_allowed_packet` bytes, it issues an `ER_NET_PACKET_TOO_LARGE` error and closes the connection. With some clients, you may also get a `Lost connection to MySQL server during query` error if the communication packet is too large.

Both the client and the server have their own `max_allowed_packet` variable, so if you want to handle big packets, you must increase this variable both in the client and in the server.

If you are using the `mysql` client program, its default `max_allowed_packet` variable is 16MB. To set a larger value, start `mysql` like this:

```
shell> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default `max_allowed_packet` value is 64MB. You can increase this if the server needs to handle big queries (for example, if you are working with big `BLOB` columns). For example, to set the variable to 128MB, start the server like this:

```
shell> mysqld --max_allowed_packet=128M
```

You can also use an option file to set `max_allowed_packet`. For example, to set the size for the server to 128MB, add the following lines in an option file:

```
[mysqld]
max_allowed_packet=128M
```

It is safe to increase the value of this variable because the extra memory is allocated only when needed. For example, `mysqld` allocates more memory only when you issue a long query or when `mysqld` must return a large result row. The small default value of the variable is a precaution to catch incorrect packets between the client and server and also to ensure that you do not run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large `BLOB` values but have not given `mysqld` access to enough memory to handle the query. If you suspect this is the case, try adding `ulimit -d 256000` to the beginning of the `mysqld_safe` script and restarting `mysqld`.

B.4.2.10 Communication Errors and Aborted Connections

If connection problems occur such as communication errors or aborted connections, use these sources of information to diagnose problems:

- The error log. See Section 5.4.2, “The Error Log”.
- The general query log. See Section 5.4.3, “The General Query Log”.
- The `Aborted_xxx` and `Connection_errors_xxx` status variables. See Section 5.1.10, “Server Status Variables”.

- The host cache, which is accessible using the Performance Schema `host_cache` table. See Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”, and Section 26.12.18.1, “The host_cache Table”.

If the `log_error_verbosity` system variable is set to 3, you might find messages like this in your error log:

```
[Note] Aborted connection 854 to db: 'employees' user: 'josh'
```

If a client is unable even to connect, the server increments the `Aborted_connects` status variable. Unsuccessful connection attempts can occur for the following reasons:

- A client attempts to access a database but has no privileges for it.
- A client uses an incorrect password.
- A connection packet does not contain the right information.
- It takes more than `connect_timeout` seconds to obtain a connect packet. See Section 5.1.8, “Server System Variables”.

If these kinds of things happen, it might indicate that someone is trying to break into your server! If the general query log is enabled, messages for these types of problems are logged to it.

If a client successfully connects but later disconnects improperly or is terminated, the server increments the `Aborted_clients` status variable, and logs an `Aborted connection` message to the error log. The cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout` or `interactive_timeout` seconds without issuing any requests to the server. See Section 5.1.8, “Server System Variables”.
- The client program ended abruptly in the middle of a data transfer.

Other reasons for problems with aborted connections or aborted clients:

- The `max_allowed_packet` variable value is too small or queries require more memory than you have allocated for `mysqld`. See Section B.4.2.9, “Packet Too Large”.
- Use of Ethernet protocol with Linux, both half and full duplex. Some Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file using FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. Switch the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and test the results to determine the best setting.
- A problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.

See also Section B.4.2.8, “MySQL server has gone away”.

B.4.2.11 The table is full

If a table-full error occurs, it may be that the disk is full or that the table has reached its maximum size. The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. See Section 8.4.6, “Limits on Table Size”.

B.4.2.12 Can't create/write to file

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start `mysqld` with the `--tmpdir` option or to add the option to the `[mysqld]` section of your option file. For example, to specify a directory of `C:\temp`, use these lines:

```
[mysqld]
tmpdir=C:/temp
```

The `C:\temp` directory must exist and have sufficient space for the MySQL server to write to. See Section 4.2.2.2, “Using Option Files”.

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the `tmpdir` directory.

Check also the error code that you get with `perror`. One reason the server cannot write to a table is that the file system is full:

```
shell> perror 28
OS error code  28:  No space left on device
```

If you get an error of the following type during startup, it indicates that the file system or directory used for storing data files is write protected. Provided that the write error is to a test file, the error is not serious and can be safely ignored.

```
Can't create test file /usr/local/mysql/data/master.lower-test
```

B.4.2.13 Commands out of sync

If you get `Commands out of sync; you can't run this command now` in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without calling `mysql_use_result()` or `mysql_store_result()` in between.

B.4.2.14 Ignoring user

If you get the following error, it means that when `mysqld` was started or when it reloaded the grant tables, it found an account in the `user` table that had an invalid password.

`Found wrong password for user '`*`some_user`*`'@'`*`some_host`*`'; ignoring user`

As a result, the account is simply ignored by the permission system. To fix this problem, assign a new, valid password to the account.

B.4.2.15 Table 'tbl_name' doesn't exist

If you get either of the following errors, it usually means that no table exists in the default database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case sensitive if they are located on a file system that has case-sensitive file names.
- Even for file systems that are not case-sensitive, such as on Windows, all references to a given table within a query must use the same lettercase.

You can check which tables are in the default database with `SHOW TABLES`. See Section 13.7.7, “SHOW Syntax”.

B.4.2.16 Can't initialize character set

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

This error can have any of the following causes:

- The character set is a multibyte character set and you have no support for the character set in the client. In this case, you need to recompile the client by running `CMake` with the `-DDEFAULT_CHARSET=charset_name` option. See Section 2.9.7, “MySQL Source-Configuration Options”.

  All standard MySQL binaries are compiled with support for all multibyte character sets.

- The character set is a simple character set that is not compiled into `mysqld`, and the character set definition files are not in the place where the client expects to find them.

  In this case, you need to use one of the following methods to solve the problem:

  - Recompile the client with support for the character set. See Section 2.9.7, “MySQL Source-Configuration Options”.
  - Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.
  - Copy the character definition files to the path where the client expects them to be.

B.4.2.17 File Not Found and Similar Errors

If you get `ERROR 'file_name' not found (errno: 23)`, `Can't open file: file_name (errno: 24)`, or any other error with `errno 23` or `errno 24` from MySQL, it means that you have not allocated enough file descriptors for the MySQL server. You can use the `perror` utility to get a description of what the error number means:

```
shell> perror 23
OS error code  23:  File table overflow
shell> perror 24
OS error code  24:  Too many open files
shell> perror 11
OS error code  11:  Resource temporarily unavailable
```

The problem here is that `mysqld` is trying to keep open too many files simultaneously. You can either tell `mysqld` not to open so many files at once or increase the number of file descriptors available to `mysqld`.

To tell `mysqld` to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the `table_open_cache` system variable (the default value is 64). This may not entirely prevent running out of file descriptors because in some circumstances the server may attempt to extend the cache size temporarily, as described in Section 8.4.3.1, “How MySQL Opens and Closes Tables”. Reducing the value of `max_connections` also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to `mysqld`, you can use the `--open-files-limit` option to `mysqld_safe` or set the `open_files_limit` system variable. See Section 5.1.8, “Server System Variables”. The easiest way to set these values is to add an option to your option file. See Section 4.2.2.2, “Using Option Files”. If you have an old version of `mysqld` that does not support setting the open files limit, you can edit the `mysqld_safe` script. There is a commented-out line `ulimit -n 256` in the script. You can remove the `#` character to uncomment this line, and change the number `256` to set the number of file descriptors to be made available to `mysqld`.

`--open-files-limit` and `ulimit` can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a “hard” limit that can be overridden only if you start `mysqld_safe` or `mysqld` as `root` (just remember that you also need to start the server with the `--user` option in this case so that it does not continue to run as `root` after it starts up). If you need to increase the operating system limit on the number of file descriptors available to each process, consult the documentation for your system.

Note

If you run the `tcsh` shell, `ulimit` does not work! `tcsh` also reports incorrect values when you ask for the current limits. In this case, you should start `mysqld_safe` using `sh`.

B.4.2.18 Table-Corruption Issues

If you have started `mysqld` with the `myisam_recover_options` system variable set, MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further.

When the server detects `MyISAM` table corruption, it writes additional information to the error log, such as the name and line number of the source file, and the list of threads accessing the table. Example: `Got an error from thread_id=1, mi_dynrec.c:368`. This is useful information to include in bug reports.

See also Section 5.1.7, “Server Command Options”, and Section 29.5.1.7, “Making a Test Case If You Experience Table Corruption”.

B.4.3 Administration-Related Issues

B.4.3.1 Problems with File Permissions

If you have problems with file permissions, the `UMASK` or `UMASK_DIR` environment variable might be set incorrectly when `mysqld` starts. For example, MySQL might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/file_name' (Errcode: 13)
```

The default `UMASK` and `UMASK_DIR` values are `0640` and `0750`, respectively. MySQL assumes that the value for `UMASK` or `UMASK_DIR` is in octal if it starts with a zero. For example, setting `UMASK=0600` is equivalent to `UMASK=384` because 0600 octal is 384 decimal.

To change the default `UMASK` value, start `mysqld_safe` as follows:

```
shell> UMASK=384  # = 600 in octal
shell> export UMASK
shell> mysqld_safe &
```

By default, MySQL creates database directories with an access permission value of `0750`. To modify this behavior, set the `UMASK_DIR` variable. If you set its value, new directories are created with the combined `UMASK` and `UMASK_DIR` values. For example, to give group access to all new directories, start `mysqld_safe` as follows:

```
shell> UMASK_DIR=504  # = 770 in octal
shell> export UMASK_DIR
shell> mysqld_safe &
```

For additional details, see Section 4.9, “MySQL Environment Variables”.

B.4.3.2 How to Reset the Root Password

If you have never assigned a `root` password for MySQL, the server does not require a password at all for connecting as `root`. However, this is insecure. For instructions on assigning a password, see Section 2.10.4, “Securing the Initial MySQL Account”.

If you know the `root` password and want to change it, see Section 13.7.1.1, “ALTER USER Syntax”, and Section 13.7.1.10, “SET PASSWORD Syntax”.

If you assigned a `root` password previously but have forgotten it, you can assign a new password. The following sections provide instructions for Windows and Unix and Unix-like systems, as well as generic instructions that apply to any system.

Resetting the Root Password: Windows Systems

On Windows, use the following procedure to reset the password for the MySQL `'root'@'localhost'` account. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

1. Log on to your system as Administrator.
2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager: From the **Start** menu, select **Control Panel**, then **Administrative Tools**, then **Services**. Find the MySQL service in the list and stop it.

   If your server is not running as a service, you may need to use the Task Manager to force it to stop.

3. Create a text file containing the password-assignment statement on a single line. Replace the password with the password that you want to use.

   ```
   ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';
   ```

4. Save the file. This example assumes that you name the file `C:\mysql-init.txt`.

5. Open a console window to get to the command prompt: From the **Start** menu, select **Run**, then enter `cmd` as the command to be run.

6. Start the MySQL server with the `init_file` system variable set to name the file (notice that the backslash in the option value is doubled):

   ```
   C:\> cd "C:\Program Files\MySQL\MySQL Server 8.0\bin"
   C:\> mysqld --init-file=C:\\mysql-init.txt
   ```

   If you installed MySQL to a different location, adjust the `cd` command accordingly.

   The server executes the contents of the file named by the `init_file` system variable at startup, changing the `'root'@'localhost'` account password.

   To have server output to appear in the console window rather than in a log file, add the `--console` option to the `mysqld` command.

   If you installed MySQL using the MySQL Installation Wizard, you may need to specify a `--defaults-file` option. For example:

   ```
   C:\> mysqld
            --defaults-file="C:\\ProgramData\\MySQL\\MySQL Server 8.0\\my.ini"
            --init-file=C:\\mysql-init.txt
   ```

   The appropriate `--defaults-file` setting can be found using the Services Manager: From the **Start** menu, select **Control Panel**, then **Administrative Tools**, then **Services**. Find the MySQL service in the list, right-click it, and choose the `Properties` option. The `Path to executable` field contains the `--defaults-file` setting.

7. After the server has started successfully, delete `C:\mysql-init.txt`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the MySQL server and restart it normally. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.

Resetting the Root Password: Unix and Unix-Like Systems

On Unix, use the following procedure to reset the password for the MySQL `'root'@'localhost'` account. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

The instructions assume that you will start the MySQL server from the Unix login account that you normally use for running it. For example, if you run the server using the `mysql` login account, you should log in as `mysql` before using the instructions. Alternatively, you can log in as `root`, but in this case you *must* start `mysqld` with the `--user=mysql` option. If you start the server as `root` without using `--user=mysql`, the server may create `root`-owned files in the data directory, such as log files, and these may cause permission-related problems for future server startups. If that happens, you will need to either change the ownership of the files to `mysql` or remove them.

1. Log on to your system as the Unix user that the MySQL server runs as (for example, `mysql`).

2. Stop the MySQL server if it is running. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, host name, and configuration. Common locations are `/var/lib/mysql/`, `/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the file name has an extension of `.pid` and begins with either `mysqld` or your system's host name.

   Stop the MySQL server by sending a normal `kill` (not `kill -9`) to the `mysqld` process. Use the actual path name of the `.pid` file in the following command:

   ```
   shell> kill `cat /mysql-data-directory/host_name.pid`
   ```

   Use backticks (not forward quotation marks) with the `cat` command. These cause the output of `cat` to be substituted into the `kill` command.

3. Create a text file containing the password-assignment statement on a single line. Replace the password with the password that you want to use.

   ```
   ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';
   ```

4. Save the file. This example assumes that you name the file `/home/me/mysql-init`. The file contains the password, so do not save it where it can be read by other users. If you are not logged in as `mysql` (the user the server runs as), make sure that the file has permissions that permit `mysql` to read it.

5. Start the MySQL server with the `init_file` system variable set to name the file:

   ```
   shell> mysqld --init-file=/home/me/mysql-init &
   ```

   The server executes the contents of the file named by the `init_file` system variable at startup, changing the `'root'@'localhost'` account password.

   Other options may be necessary as well, depending on how you normally start your server. For example, `--defaults-file` may be needed before the `init_file` argument.

6. After the server has started successfully, delete `/home/me/mysql-init`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally.

Resetting the Root Password: Generic Instructions

The preceding sections provide password-resetting instructions specifically for Windows and Unix and Unix-like systems. Alternatively, on any platform, you can reset the password using the `mysql` client (but this approach is less secure):

1. Stop the MySQL server if necessary, then restart it with the `--skip-grant-tables` option. This enables anyone to connect without a password and with all privileges, and disables account-management statements such as `ALTER USER` and `SET PASSWORD`. Because this is insecure, if the server is started with the `--skip-grant-tables` option, it enables `--skip-networking` automatically to prevent remote connections.

2. Connect to the MySQL server using the `mysql` client; no password is necessary because the server was started with `--skip-grant-tables`:

   ```
   ```

```
shell> mysql
```

3. In the `mysql` client, tell the server to reload the grant tables so that account-management statements work:

   ```
   mysql> FLUSH PRIVILEGES;
   ```

   Then change the `'root'@'localhost'` account password. Replace the password with the password that you want to use. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

   ```
   mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'MyNewPass';
   ```

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally (without the `--skip-grant-tables` and `--skip-networking` options).

B.4.3.3 What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This does not mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the `mysqld` server dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died and restarted, you may find the reason by looking in the server's error log. See Section 5.4.2, “The Error Log”.

On some systems, you can find in the error log a stack trace of where `mysqld` died. Note that the variable values written in the error log may not always be 100% correct.

Many server crashes are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with the `delay_key_write` system variable enabled, in which case data files are written but not index files.) This means that data file contents are safe even if `mysqld` crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting `mysqld` with the `--flush` option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in `mysqld` that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as `mysqld` without locking the table properly.
- You are running many `mysqld` servers using the same data directory on a system that does not support good file system locks (normally handled by the `lockd` lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused `mysqld`.
- You have found a bug in the data storage code. This isn't likely, but it is at least possible. In this case, you can try to change the storage engine to another engine by using `ALTER TABLE` on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others crash for you. Try the following things:

- Stop the `mysqld` server with `mysqladmin shutdown`, run `myisamchk --silent --force */*.MYI` from the data directory to check all `MyISAM` tables, and restart `mysqld`. This ensures that you are running from a clean state. See Chapter 5, *MySQL Server Administration*.
- Start `mysqld` with the general query log enabled (see Section 5.4.3, “The General Query Log”). Then try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See Section 5.4.3, “The General Query Log”. If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have isolated the bug and should submit a bug report for it. See Section 1.7, “How to Report Bugs or Problems”.
- Try to make a test case that we can use to repeat the problem. See Section 29.5, “Debugging and Porting MySQL”.
- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)
- Configuring MySQL for debugging makes it much easier to gather information about possible errors if something goes wrong. Reconfigure MySQL with the `-DWITH_DEBUG=1` option to `CMake` and then recompile. See Section 29.5, “Debugging and Porting MySQL”.
- Make sure that you have applied the latest patches for your operating system.
- Use the `--skip-external-locking` option to `mysqld`. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` option tells `mysqld` not to use external locking. (This means that you cannot run two `mysqld` servers on the same data directory and that you must be careful if you use `myisamchk`. Nevertheless, it may be instructive to try the option as a test.)
- If `mysqld` appears to be running but not responding, try `mysqladmin -u root processlist`. Sometimes `mysqld` is not hung even though it seems unresponsive. The problem may be that all connections are in use, or there may be some internal lock problem. `mysqladmin -u root processlist` usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.
- Run the command `mysqladmin -i 5 status` or `mysqladmin -i 5 -r status` in a separate window to produce statistics while running other queries.
- Try the following:

  1. Start `mysqld` from `gdb` (or another debugger). See Section 29.5, “Debugging and Porting MySQL”.
  2. Run your test scripts.
  3. Print the backtrace and the local variables at the three lowest levels. In `gdb`, you can do this with the following commands when `mysqld` has crashed inside `gdb`:

     ```
     backtrace
     info local
     up
     info local
     up
     info local
     ```

     With `gdb`, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread N`, where *`N`* is the thread ID.

- Try to simulate your application with a Perl script to force MySQL to crash or misbehave.
- Send a normal bug report. See Section 1.7, “How to Report Bugs or Problems”. Be even more detailed than usual. Because MySQL works for many people, the crash might result from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only `VARCHAR` columns (not `BLOB` or `TEXT` columns), you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

  The current dynamic row code has been in use for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.
- Consider the possibility of hardware faults when diagnosing problems. Defective hardware can be the cause of data corruption. Pay particular attention to your memory and disk subsystems when troubleshooting hardware.

B.4.3.4 How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as “no space left on device”), and to quota-exceeded errors (such as “write failed” or “user block limit reached”).

This section is relevant for writes to `MyISAM` tables. It also applies for writes to binary log files and binary log index file, except that references to “row” and “record” should be understood to mean “event.”

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- Alternatively, to abort the thread, use `mysqladmin kill`. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several “locked” threads, killing the one thread that is waiting on the disk-full condition enables the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA` or after an `ALTER TABLE` statement. All of these statements may create large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these operations, it removes the big temporary files and mark the table as crashed. The exception is that for `ALTER TABLE`, the old table is left unchanged.

B.4.3.5 Where MySQL Stores Temporary Files

On Unix, MySQL uses the value of the `TMPDIR` environment variable as the path name of the directory in which to store temporary files. If `TMPDIR` is not set, MySQL uses the system default, which is usually `/tmp`, `/var/tmp`, or `/usr/tmp`.

On Windows, MySQL checks in order the values of the `TMPDIR`, `TEMP`, and `TMP` environment variables. For the first one found to be set, MySQL uses it and does not check those remaining. If none of `TMPDIR`, `TEMP`, or `TMP` are set, MySQL uses the Windows system default, which is usually `C:\windows\temp\`.

If the file system containing your temporary file directory is too small, you can use the `mysqld --tmpdir` option to specify a directory in a file system where you have enough space.

The `--tmpdir` option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (`:`) on Unix and semicolon characters (`;`) on Windows.

Note

To spread the load effectively, these paths should be located on different *physical* disks, not different partitions of the same disk.

If the MySQL server is acting as a replication slave, you can use the `--slave-load-tmpdir` option to specify a separate directory for holding temporary files when replicating `LOAD DATA` statements. This directory should be in a disk-based file system (not a memory-based file system) so that the temporary files used to replicate LOAD DATA can survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process. However, replication can now continue after a restart if the temporary files have been removed.

MySQL arranges that temporary files are removed if `mysqld` is terminated. On platforms that support it (such as Unix), this is done by unlinking the file after opening it. The disadvantage of this is that the name does not appear in directory listings and you do not see a big temporary file that fills up the file system in which the temporary file directory is located. (In such cases, `lsof +L1` may be helpful in identifying large files associated with `mysqld`.)

When sorting (`ORDER BY` or `GROUP BY`), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some statements, MySQL creates temporary SQL tables that are not hidden and have names that begin with `#sql`.

Some `SELECT` queries creates temporary SQL tables to hold intermediate results.

DDL operations that rebuild the table and are not performed online using the `ALGORITHM=INPLACE` technique create a temporary copy of the original table in the same directory as the original table.

Online DDL operations may use temporary log files for recording concurrent DML, temporary sort files when creating an index, and temporary intermediate tables files when rebuilding the table. For more information, see Section 15.12.3, “Online DDL Space Requirements”.

`InnoDB` user-created temporary tables and on-disk internal temporary tables are created in a temporary tablespace file named `ibtmp1` in the MySQL data directory. For more information, see Section 15.6.3.5, “Temporary Tablespaces”.

See also Section 15.14.7, “InnoDB INFORMATION_SCHEMA Temporary Table Info Table”.

The optional `EXTENDED` modifier causes `SHOW TABLES` to list hidden tables created by failed `ALTER TABLE` statements. See Section 13.7.7.37, “SHOW TABLES Syntax”.

B.4.3.6 How to Protect or Change the MySQL Unix Socket File

The default location for the Unix socket file that the server uses for communication with local clients is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

On some versions of Unix, anyone can delete files in the `/tmp` directory or other similar directories used for temporary files. If the socket file is located in such a directory on your system, this might cause problems.

On most versions of Unix, you can protect your `/tmp` directory so that files can be deleted only by their owners or the superuser (`root`). To do this, set the `sticky` bit on the `/tmp` directory by logging in as `root` and using the following command:

```
shell> chmod +t /tmp
```

You can check whether the `sticky` bit is set by executing `ls -ld /tmp`. If the last permission character is `t`, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in `/etc/my.cnf`:

  ```
  [mysqld]
  socket=/path/to/socket

  [client]
  socket=/path/to/socket
  ```

  See Section 4.2.2.2, “Using Option Files”.

- Specify a `--socket` option on the command line to `mysqld_safe` and when you run client programs.
- Set the `MYSQL_UNIX_PORT` environment variable to the path of the Unix socket file.
- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the `MYSQL_UNIX_ADDR` option when you run `CMake`. See Section 2.9.7, “MySQL Source-Configuration Options”.

You can test whether the new socket location works by attempting to connect to the server with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

B.4.3.7 Time Zone Problems

If you have a problem with `SELECT NOW()` returning values in UTC and not your local time, you have to tell the server your current time zone. The same applies if `UNIX_TIMESTAMP()` returns the wrong value. This should be done for the environment in which the server runs (for example, in `mysqld_safe` or `mysql.server`). See Section 4.9, “MySQL Environment Variables”.

You can set the time zone for the server with the `--timezone=`*`timezone_name`* option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`.

The permissible values for `--timezone` or `TZ` are system dependent. Consult your operating system documentation to see what values are acceptable.

B.4.4 Query-Related Issues

B.4.4.1 Case Sensitivity in String Searches

For nonbinary strings (`CHAR`, `VARCHAR`, `TEXT`), string searches use the collation of the comparison operands. For binary strings (`BINARY`, `VARBINARY`, `BLOB`), comparisons use the numeric values of the bytes in the operands; this means that for alphabetic characters, comparisons will be case-sensitive.

A comparison between a nonbinary string and binary string is treated as a comparison of binary strings.

Simple comparison operations (`>=, >, =, <, <=`, sorting, and grouping) are based on each character's "sort value." Characters with the same sort value are treated as the same character. For example, if `e` and `é` have the same sort value in a given collation, they compare as equal.

The default character set and collation are `utf8mb4` and `utf8mb4_0900_ai_ci`, so nonbinary string comparisons are case insensitive by default. This means that if you search with *`col_name`* `LIKE 'a%'`, you get all column values that start with `A` or `a`. To make this search case-sensitive, make sure that one of the operands has a case-sensitive or binary collation. For example, if you are comparing a column and a string that both have the `utf8mb4` character set, you can use the `COLLATE` operator to cause either operand to have the `utf8mb4_0900_as_cs` or `utf8mb4_bin` collation:

```
col_name COLLATE utf8mb4_0900_as_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE utf8mb4_0900_as_cs
col_name COLLATE utf8mb4_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE utf8mb4_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case-sensitive or binary collation. See Section 13.1.20, "CREATE TABLE Syntax".

To cause a case-sensitive comparison of nonbinary strings to be case insensitive, use `COLLATE` to name a case-insensitive collation. The strings in the following example normally are case-sensitive, but `COLLATE` changes the comparison to be case insensitive:

```
mysql> SET NAMES 'utf8mb4';
mysql> SET @s1 = 'MySQL' COLLATE utf8mb4_bin,
           @s2 = 'mysql' COLLATE utf8mb4_bin;
mysql> SELECT @s1 = @s2;
+-----------+
| @s1 = @s2 |
+-----------+
|         0 |
+-----------+
mysql> SELECT @s1 COLLATE utf8mb4_0900_ai_ci = @s2;
+--------------------------------------+
| @s1 COLLATE utf8mb4_0900_ai_ci = @s2 |
+--------------------------------------+
|                                    1 |
+--------------------------------------+
```

A binary string is case-sensitive in comparisons. To compare the string as case insensitive, convert it to a nonbinary string and use `COLLATE` to name a case-insensitive collation:

```
mysql> SET @s = BINARY 'MySQL';
mysql> SELECT @s = 'mysql';
```

```
+--------------+
| @s = 'mysql' |
+--------------+
|            0 |
+--------------+
mysql> SELECT CONVERT(@s USING utf8mb4) COLLATE utf8mb4_0900_ai_ci = 'mysql';
+-------------------------------------------------------------------+
| CONVERT(@s USING utf8mb4) COLLATE utf8mb4_0900_ai_ci = 'mysql' |
+-------------------------------------------------------------------+
|                                                                 1 |
+-------------------------------------------------------------------+
```

To determine whether a value will compare as a nonbinary or binary string, use the `COLLATION()` function. This example shows that `VERSION()` returns a string that has a case-insensitive collation, so comparisons are case insensitive:

```
mysql> SELECT COLLATION(VERSION());
+----------------------+
| COLLATION(VERSION()) |
+----------------------+
| utf8_general_ci      |
+----------------------+
```

For binary strings, the collation value is `binary`, so comparisons will be case sensitive. One context in which you will see `binary` is for compression functions, which return binary strings as a general rule: string:

```
mysql> SELECT COLLATION(COMPRESS('x'));
+--------------------------+
| COLLATION(COMPRESS('x')) |
+--------------------------+
| binary                   |
+--------------------------+
```

To check the sort value of a string, the `WEIGHT_STRING()` may be helpful. See Section 12.5, “String Functions and Operators”.

B.4.4.2 Problems Using DATE Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is permitted. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

```
SELECT * FROM t1 WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in a numeric context and vice versa. MySQL also permits a “relaxed” string format when updating and in a `WHERE` clause that compares a date to a `DATE`, `DATETIME`, or `TIMESTAMP` column. “Relaxed” format means that any punctuation character may be used as the separator between parts. For example, `'2004-08-15'` and `'2004#08#15'` are equivalent. MySQL can also convert a string containing no separators (such as `'20040815'`), provided it makes sense as a date.

When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` to a constant string with the `<`, `<=`, `=`, `>=`, `>`, or `BETWEEN` operators, MySQL normally converts the string to an internal long integer for faster comparison (and also for a bit more “relaxed” string checking). However, this conversion is subject to the following exceptions:

- When you compare two columns

- When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` column to an expression
- When you use any comparison method other than those just listed, such as `IN` or `STRCMP()`.

For those exceptions, the comparison is done by converting the objects to strings and performing a string comparison.

To be on the safe side, assume that strings are compared as strings and use the appropriate string functions if you want to compare a temporal value to a string.

The special "zero" date `'0000-00-00'` can be stored and retrieved as `'0000-00-00'.` When a `'0000-00-00'` date is used through Connector/ODBC, it is automatically converted to `NULL` because ODBC cannot handle that kind of date.

Because MySQL performs the conversions just described, the following statements work (assume that `idate` is a `DATE` column):

```
INSERT INTO t1 (idate) VALUES (19970505);
INSERT INTO t1 (idate) VALUES ('19970505');
INSERT INTO t1 (idate) VALUES ('97-05-05');
INSERT INTO t1 (idate) VALUES ('1997.05.05');
INSERT INTO t1 (idate) VALUES ('1997 05 05');
INSERT INTO t1 (idate) VALUES ('0000-00-00');

SELECT idate FROM t1 WHERE idate >= '1997-05-05';
SELECT idate FROM t1 WHERE idate >= 19970505;
SELECT MOD(idate,100) FROM t1 WHERE idate >= 19970505;
SELECT idate FROM t1 WHERE idate >= '19970505';
```

However, the following statement does not work:

```
SELECT idate FROM t1 WHERE STRCMP(idate,'20030505')=0;
```

`STRCMP()` is a string function, so it converts `idate` to a string in `'YYYY-MM-DD'` format and performs a string comparison. It does not convert `'20030505'` to the date `'2003-05-05'` and perform a date comparison.

If you enable the `ALLOW_INVALID_DATES` SQL mode, MySQL permits you to store dates that are given only limited checking: MySQL requires only that the day is in the range from 1 to 31 and the month is in the range from 1 to 12. This makes MySQL very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation).

MySQL permits you to store dates where the day or month and day are zero. This is convenient if you want to store a birthdate in a `DATE` column and you know only part of the date. To disallow zero month or day parts in dates, enable the `NO_ZERO_IN_DATE` mode.

MySQL permits you to store a "zero" value of `'0000-00-00'` as a "dummy date." This is in some cases more convenient than using `NULL` values. If a date to be stored in a `DATE` column cannot be converted to any reasonable value, MySQL stores `'0000-00-00'`. To disallow `'0000-00-00'`, enable the `NO_ZERO_DATE` mode.

To have MySQL check all dates and accept only legal dates (unless overridden by `IGNORE`), set the `sql_mode` system variable to `"NO_ZERO_IN_DATE,NO_ZERO_DATE"`.

B.4.4.3 Problems with NULL Values

The concept of the NULL value is a common source of confusion for newcomers to SQL, who often think that NULL is the same thing as an empty string ''. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Both statements insert a value into the phone column, but the first inserts a NULL value and the second inserts an empty string. The meaning of the first can be regarded as “phone number is not known” and the meaning of the second can be regarded as “the person is known to have no phone, and thus no phone number.”

To help with NULL handling, you can use the IS NULL and IS NOT NULL operators and the IFNULL() function.

In SQL, the NULL value is never true in comparison to any other value, even NULL. An expression that contains NULL always produces a NULL value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return NULL:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

To search for column values that are NULL, you cannot use an expr = NULL test. The following statement returns no rows, because expr = NULL is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for NULL values, you must use the IS NULL test. The following statements show how to find the NULL phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = '';
```

See Section 3.3.4.6, “Working with NULL Values”, for additional information and examples.

You can add an index on a column that can have NULL values if you are using the MyISAM, InnoDB, or MEMORY storage engine. Otherwise, you must declare an indexed column NOT NULL, and you cannot insert NULL into the column.

When reading data with LOAD DATA, empty or missing columns are updated with ''. To load a NULL value into a column, use \N in the data file. The literal word NULL may also be used under some circumstances. See Section 13.2.7, “LOAD DATA Syntax”.

When using DISTINCT, GROUP BY, or ORDER BY, all NULL values are regarded as equal.

When using ORDER BY, NULL values are presented first, or last if you specify DESC to sort in descending order.

Aggregate (summary) functions such as COUNT(), MIN(), and SUM() ignore NULL values. The exception to this is COUNT(*), which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-NULL values in the age column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some data types, MySQL handles `NULL` values specially. If you insert `NULL` into a `TIMESTAMP` column, the current date and time is inserted. If you insert `NULL` into an integer or floating-point column that has the `AUTO_INCREMENT` attribute, the next number in the sequence is inserted.

B.4.4.4 Problems with Column Aliases

An alias can be used in a query select list to give a column a different name. You can use the alias in `GROUP BY`, `ORDER BY`, or `HAVING` clauses to refer to the column:

```
SELECT SQRT(a*b) AS root FROM tbl_name
  GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name
  GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL disallows references to column aliases in a `WHERE` clause. This restriction is imposed because when the `WHERE` clause is evaluated, the column value may not yet have been determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl_name
  WHERE cnt > 0 GROUP BY id;
```

The `WHERE` clause determines which rows should be included in the `GROUP BY` clause, but it refers to the alias of a column value that is not known until after the rows have been selected, and grouped by the `GROUP BY`.

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
SELECT 1 AS `one`, 2 AS 'two';
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal. For example, this statement groups by the values in column `id`, referenced using the alias `` `a` ``:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY `a`;
```

But this statement groups by the literal string `'a'` and will not work as expected:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY 'a';
```

B.4.4.5 Rollback Failure for Nontransactional Tables

If you receive the following message when trying to perform a `ROLLBACK`, it means that one or more of the tables you used in the transaction do not support transactions:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

These nontransactional tables are not affected by the `ROLLBACK` statement.

If you were not deliberately mixing transactional and nontransactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can

happen if you try to create a table using a transactional storage engine that is not supported by your `mysqld` server (or that was disabled with a startup option). If `mysqld` does not support a storage engine, it instead creates the table as a `MyISAM` table, which is nontransactional.

You can check the storage engine for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

See Section 13.7.7.36, “SHOW TABLE STATUS Syntax”, and Section 13.7.7.10, “SHOW CREATE TABLE Syntax”.

To check which storage engines your `mysqld` server supports, use this statement:

```
SHOW ENGINES;
```

See Section 13.7.7.16, “SHOW ENGINES Syntax” for full details.

B.4.4.6 Deleting Rows from Related Tables

If the total length of the `DELETE` statement for `related_table` is more than the default value of the `max_allowed_packet` system variable, you should split it into smaller parts and execute multiple `DELETE` statements. You probably get the fastest `DELETE` by specifying only 100 to 1,000 `related_column` values per statement if the `related_column` is indexed. If the `related_column` isn't indexed, the speed is independent of the number of arguments in the `IN` clause.

B.4.4.7 Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that returns no rows, you should use the following procedure to find out what is wrong:

1. Test the query with `EXPLAIN` to check whether you can find something that is obviously wrong. See Section 13.8.2, “EXPLAIN Syntax”.

2. Select only those columns that are used in the `WHERE` clause.

3. Remove one table at a time from the query until it returns some rows. If the tables are large, it is a good idea to use `LIMIT 10` with the query.

4. Issue a `SELECT` for the column that should have matched a row against the table that was last removed from the query.

5. If you are comparing `FLOAT` or `DOUBLE` columns with numbers that have decimals, you cannot use equality (`=`) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some cases, changing the `FLOAT` to a `DOUBLE` fixes this. See Section B.4.4.8, “Problems with Floating-Point Values”.

6. If you still cannot figure out what is wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file by dumping the tables with `mysqldump --quick db_name tbl_name_1 ... tbl_name_n > query.sql`. Open the file in an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your `SELECT` statement at the end of the file.

   Verify that the test file demonstrates the problem by executing these commands:

   ```
   ```

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

Attach the test file to a bug report, which you can file using the instructions in Section 1.7, “How to Report Bugs or Problems”.

B.4.4.8 Problems with Floating-Point Values

Floating-point numbers sometimes cause confusion because they are approximate and not stored as exact values. A floating-point value as written in an SQL statement may not be the same as the value represented internally. Attempts to treat floating-point values as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. The `FLOAT` and `DOUBLE` data types are subject to these issues. For `DECIMAL` columns, MySQL performs operations with a precision of 65 decimal digits, which should solve most common inaccuracy problems.

The following example uses `DOUBLE` to demonstrate how calculations that are done using floating-point operations are subject to floating-point error.

```
mysql> CREATE TABLE t1 (i INT, d1 DOUBLE, d2 DOUBLE);
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
    -> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
    -> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
    -> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
    -> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
    -> (6, 0.00, 0.00), (6, -51.40, 0.00);

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
    -> FROM t1 GROUP BY i HAVING a <> b;

+------+-------+------+
| i    | a     | b    |
+------+-------+------+
|    1 |  21.4 | 21.4 |
|    2 |  76.8 | 76.8 |
|    3 |   7.4 |  7.4 |
|    4 |  15.4 | 15.4 |
|    5 |   7.2 |  7.2 |
|    6 | -51.4 |    0 |
+------+-------+------+
```

The result is correct. Although the first five records look like they should not satisfy the comparison (the values of `a` and `b` do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on factors such as computer architecture or the compiler version or optimization level. For example, different CPUs may evaluate floating-point numbers differently.

If columns `d1` and `d2` had been defined as `DECIMAL` rather than `DOUBLE`, the result of the `SELECT` query would have contained only one row—the last one shown above.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) > 0.0001;
+------+-------+------+
| i    | a     | b    |
```

```
+------+-------+------+
|    6 | -51.4 |    0 |
+------+-------+------+
1 row in set (0.00 sec)
```

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) <= 0.0001;
+------+------+------+
| i    | a    | b    |
+------+------+------+
|    1 | 21.4 | 21.4 |
|    2 | 76.8 | 76.8 |
|    3 |  7.4 |  7.4 |
|    4 | 15.4 | 15.4 |
|    5 |  7.2 |  7.2 |
+------+------+------+
5 rows in set (0.03 sec)
```

Floating-point values are subject to platform or implementation dependencies. Suppose that you execute the following statements:

```
CREATE TABLE t1(c1 FLOAT(53,0), c2 FLOAT(53,0));
INSERT INTO t1 VALUES('1e+52','-1e+52');
SELECT * FROM t1;
```

On some platforms, the `SELECT` statement returns `inf` and `-inf`. On others, it returns `0` and `-0`.

An implication of the preceding issues is that if you attempt to create a replication slave by dumping table contents with `mysqldump` on the master and reloading the dump file into the slave, tables containing floating-point columns might differ between the two hosts.

B.4.5 Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL does not have enough information about the data at hand and has to make “educated” guesses about the data.

For the cases when MySQL does not do the "right" thing, tools that you have available to help MySQL are:

- Use the `EXPLAIN` statement to get information about how MySQL processes a query. To use it, just add the keyword `EXPLAIN` to the front of your `SELECT` statement:

  ```
  mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
  ```

  `EXPLAIN` is discussed in more detail in Section 13.8.2, “EXPLAIN Syntax”.

- Use `ANALYZE TABLE` *`tbl_name`* to update the key distributions for the scanned table. See Section 13.7.3.1, “ANALYZE TABLE Syntax”.

- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

  ```
  SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
  WHERE t1.col_name=t2.col_name;
  ```

`USE INDEX` and `IGNORE INDEX` may also be useful. See Section 8.9.4, “Index Hints”.

- Global and table-level `STRAIGHT_JOIN`. See Section 13.2.10, “SELECT Syntax”.
- You can tune global or thread-specific system variables. For example, start `mysqld` with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.8, “Server System Variables”.

B.4.6 Table Definition-Related Issues

B.4.6.1 Problems with ALTER TABLE

If you get a duplicate-key error when using `ALTER TABLE` to change the character set or collation of a character column, the cause is either that the new column collation maps two keys to the same value or that the table is corrupted. In the latter case, you should run `REPAIR TABLE` on the table. `REPAIR TABLE` works for `MyISAM`, `ARCHIVE`, and `CSV` tables.

If you use `ALTER TABLE` on a transactional table or if you are using Windows, `ALTER TABLE` unlocks the table if you had done a `LOCK TABLE` on it. This is done because `InnoDB` and these operating systems cannot drop a table that is in use.

B.4.6.2 TEMPORARY Table Problems

Temporary tables created with `CREATE TEMPORARY TABLE` have the following limitations:

- `TEMPORARY` tables are supported only by the `InnoDB`, `MEMORY`, `MyISAM`, and `MERGE` storage engines.
- Temporary tables are not supported for NDB Cluster.
- The `SHOW TABLES` statement does not list `TEMPORARY` tables.
- To rename `TEMPORARY` tables, `RENAME TABLE` does not work. Use `ALTER TABLE` instead:

```
ALTER TABLE old_name RENAME new_name;
```

- You cannot refer to a `TEMPORARY` table more than once in the same query. For example, the following does not work:

```
SELECT * FROM temp_table JOIN temp_table AS t2;
```

The statement produces this error:

```
ERROR 1137: Can't reopen table: 'temp_table'
```

You can work around this issue if your query permits use of a common table expression (CTE) rather than a `TEMPORARY` table. For example, this fails with the `Can't reopen table` error:

```
CREATE TEMPORARY TABLE t SELECT 1 AS col_a, 2 AS col_b;
SELECT * FROM t AS t1 JOIN t AS t2;
```

To avoid the error, use a `WITH` clause that defines a CTE, rather than the `TEMPORARY` table:

```
WITH cte AS (SELECT 1 AS col_a, 2 AS col_b)
```

```
SELECT * FROM cte AS t1 JOIN cte AS t2;
```

- The `Can't reopen table` error also occurs if you refer to a temporary table multiple times in a stored function under different aliases, even if the references occur in different statements within the function. It may occur for temporary tables created outside stored functions and referred to across multiple calling and callee functions.
- If a `TEMPORARY` is created with the same name as an existing non-`TEMPORARY` table, the non-`TEMPORARY` table is hidden until the `TEMPORARY` table is dropped, even if the tables use different storage engines.
- There are known issues in using temporary tables with replication. See Section 17.5.1.30, "Replication and Temporary Tables", for more information.

B.4.7 Known Issues in MySQL

This section lists known issues in recent versions of MySQL.

For information about platform-specific issues, see the installation and porting instructions in Section 2.1, "General Installation Guidance", and Section 29.5, "Debugging and Porting MySQL".

The following problems are known:

- Subquery optimization for `IN` is not as effective as for `=`.
- Even if you use `lower_case_table_names=2` (which enables MySQL to remember the case used for databases and table names), MySQL does not remember the case used for database names for the function `DATABASE()` or within the various logs (on case-insensitive systems).
- Dropping a `FOREIGN KEY` constraint does not work in replication because the constraint may have another name on the slave.
- `REPLACE` (and `LOAD DATA` with the `REPLACE` option) does not trigger `ON DELETE CASCADE`.
- `DISTINCT` with `ORDER BY` does not work inside `GROUP_CONCAT()` if you do not use all and only those columns that are in the `DISTINCT` list.
- When inserting a big integer value (between 2^{63} and $2^{64}-1$) into a decimal or string column, it is inserted as a negative value because the number is evaluated in a signed integer context.
- With statement-based binary logging, the master writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases. However, it is possible for the data on the master and slave to become different if a query is designed in such a way that the data modification is nondeterministic (generally not a recommended practice, even outside of replication).

  For example:

  - `CREATE TABLE ... SELECT` or `INSERT ... SELECT` statements that insert zero or `NULL` values into an `AUTO_INCREMENT` column.
  - `DELETE` if you are deleting rows from a table that has foreign keys with `ON DELETE CASCADE` properties.
  - `REPLACE ... SELECT`, `INSERT IGNORE ... SELECT` if you have duplicate key values in the inserted data.

  If and only if the preceding queries have no `ORDER BY` clause guaranteeing a deterministic order.

For example, for `INSERT ... SELECT` with no `ORDER BY`, the `SELECT` may return rows in a different order (which results in a row having different ranks, hence getting a different number in the `AUTO_INCREMENT` column), depending on the choices made by the optimizers on the master and slave.

A query is optimized differently on the master and slave only if:

- The table is stored using a different storage engine on the master than on the slave. (It is possible to use different storage engines on the master and slave. For example, you can use `InnoDB` on the master, but `MyISAM` on the slave if the slave has less available disk space.)
- MySQL buffer sizes (`key_buffer_size`, and so on) are different on the master and slave.
- The master and slave run different MySQL versions, and the optimizer code differs between these versions.

This problem may also affect database restoration using `mysqlbinlog|mysql`.

The easiest way to avoid this problem is to add an `ORDER BY` clause to the aforementioned nondeterministic queries to ensure that the rows are always stored or modified in the same order. Using row-based or mixed logging format also avoids the problem.

- Log file names are based on the server host name if you do not specify a file name with the startup option. To retain the same log file names if you change your host name to something else, you must explicitly use options such as `--log-bin=`*`old_host_name`*`-bin`. See Section 5.1.7, “Server Command Options”. Alternatively, rename the old files to reflect your host name change. If these are binary logs, you must edit the binary log index file and fix the binary log file names there as well. (The same is true for the relay logs on a slave server.)
- `mysqlbinlog` does not delete temporary files left after a `LOAD DATA` statement. See Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.
- `RENAME` does not work with `TEMPORARY` tables or tables used in a `MERGE` table.
- When using `SET CHARACTER SET`, you cannot use translated characters in database, table, and column names.
- Prior to MySQL 8.0.17, you cannot use `_` or `%` with `ESCAPE` in `LIKE ... ESCAPE`.
- The server uses only the first `max_sort_length` bytes when comparing data values. This means that values cannot reliably be used in `GROUP BY`, `ORDER BY`, or `DISTINCT` if they differ only after the first `max_sort_length` bytes. To work around this, increase the variable value. The default value of `max_sort_length` is 1024 and can be changed at server startup time or at runtime.
- Numeric calculations are done with `BIGINT` or `DOUBLE` (both are normally 64 bits long). Which precision you get depends on the function. The general rule is that bit functions are performed with `BIGINT` precision, `IF()` and `ELT()` with `BIGINT` or `DOUBLE` precision, and the rest with `DOUBLE` precision. You should try to avoid using unsigned long long values if they resolve to be larger than 63 bits (9223372036854775807) for anything other than bit fields.
- You can have up to 255 `ENUM` and `SET` columns in one table.
- In `MIN()`, `MAX()`, and other aggregate functions, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set.
- In an `UPDATE` statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments `KEY` by `2`, **not** `1`:

```
mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- The optimizer may handle `DISTINCT` differently when you are using “hidden” columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns do not participate in the `DISTINCT` comparison.

  An example of this is:

```
SELECT DISTINCT mp3id FROM band_downloads
       WHERE userid = 9 ORDER BY id DESC;
```

  and

```
SELECT DISTINCT band_downloads.mp3id
       FROM band_downloads,band_mp3
       WHERE band_downloads.userid = 9
       AND band_mp3.id = band_downloads.mp3id
       ORDER BY band_downloads.id DESC;
```

  In the second case, you may get two identical rows in the result set (because the values in the hidden `id` column may differ).

  Note that this happens only for queries that do not have the `ORDER BY` columns in the result.

- If you execute a `PROCEDURE` on a query that returns an empty set, in some cases the `PROCEDURE` does not transform the columns.

- Creation of a table of type `MERGE` does not check whether the underlying tables are compatible types.

- If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table and then add a normal index on the `MERGE` table, the key order is different for the tables if there was an old, non-`UNIQUE` key in the table. This is because `ALTER TABLE` puts `UNIQUE` indexes before normal indexes to be able to detect duplicate keys as early as possible.

Appendix C Indexes

Table of Contents

General Index 5651
C Function Index 6087
Command Index 6105
Function Index 6152
INFORMATION_SCHEMA Index 6194
Join Types Index 6209
Operator Index 6211
Option Index 6221
Privileges Index 6326
SQL Modes Index 6343
Statement/Syntax Index 6346
Status Variable Index 6446
System Variable Index 6478
Transaction Isolation Level Index 6571

General Index

Symbols

! (logical NOT), 1926
!= (not equal), 1921
", 1679
%, 1966
% (modulo), 1971
% (wildcard character), 1671
& (bitwise AND), 2053
&& (logical AND), 1927
() (parentheses), 1919
(Control+Z) \Z, 1671, 2406
* (multiplication), 1965
+ (addition), 1965
- (subtraction), 1965
- (unary minus), 1965
--master-info-repository option, 3317
--password option, 1068
--relay-log-info-repository option, 3317
->, 2123
->>, 2125
-? option (NDB Cluster programs), 4068
-c option (NDB Cluster programs), 4069
-c option (ndb_mgmd) (OBSOLETE), 3960
-d option (ndb_index_stat), 4006
-d option (ndb_mgmd), 3961
-e option (ndb_mgm), 3967
-f option (ndb_mgmd), 3960
-l option (ndbinfo_select_all), 3956
-n option (ndbd), 3952

-n option (ndbmtd), 3952
-p option, 1068
-P option (ndb_mgmd), 3965
-V option (NDB Cluster programs), 4070
-v option (ndb_mgmd), 3966
.ibd file, 2336
.my.cnf option file, 321, 322, 343, 1063, 1069, 1164
.MYD file, 2336
.MYI file, 2336
.mylogin.cnf option file, 321, 540
.mysql_history file, 406, 1069
.pid (process ID) file, 1443
.sdi file, 2389
/ (division), 1965
/etc/passwd, 1072, 2428
3306 port, 215, 691
33060 port, 215
:= (assignment operator), 1928
:= (assignment), 1720
< (less than), 1921
<< (left shift), 308, 2054
<= (less than or equal), 1921
<=> (equal to), 1921
<> (not equal), 1921
= (assignment operator), 1929
= (assignment), 1720
= (equal), 1920
> (greater than), 1922
>= (greater than or equal), 1922
>> (right shift), 2054
@master_binlog_checksum user-defined variable, 4924
[api] (NDB Cluster), 3748
[computer] (NDB Cluster), 3749
[mgm] (NDB Cluster), 3747
[mysqld] (NDB Cluster), 3748
[ndbd default] (NDB Cluster), 3740
[ndbd] (NDB Cluster), 3740
[ndb_mgmd] (NDB Cluster), 3747
[shm] (NDB Cluster), 3749
[tcp] (NDB Cluster), 3749
\" (double quote), 1671, 2142
\' (single quote), 1671
\. (mysql client command), 303, 410
\0 (ASCII NUL), 1670, 2406
\b (backspace), 1671, 2142, 2406
\f (formfeed), 2142
\n (linefeed), 1671, 2142, 2406
\n (newline), 1671, 2142, 2406
\N (NULL), 2406
\r (carriage return), 1671, 2142, 2406
\t (tab), 1671, 2142, 2406
\u (Unicode character), 2142
\Z (Control+Z) ASCII 26, 1671, 2406
\\ (escape), 1671, 2142

^ (bitwise XOR), 2054
_ (wildcard character), 1671
_ai collation sufffix, 1739
_as collation sufffix, 1739
_bin collation sufffix, 1739, 1763
_ci collation sufffix, 1739
_cs collation sufffix, 1739
_ks collation sufffix, 1739
_rowid
  SELECT statements, 2287, 2317, 2317
`, 1679
| (bitwise OR), 2053
|| (logical OR), 1927
~ (invert bits), 2055

A

abort-on-error option
  ndb_import, 3994
  ndb_move_data, 4011
abort-slave-event-count option
  mysqld, 3239
aborted clients, 5626
aborted connection, 5626
ABS(), 1967
access control, 1077, 1107
access denied errors, 5618
access privileges, 1077
account
  default, 240
  root, 240
account categories, 1123
account locking, 1100, 1160
  ALTER USER, 2573
  CREATE USER statement, 2583
  Locked_connects status variable, 905
account management, 1077
account names, 1104
accounts
  adding privileges, 1112
  creating, 1112
  deleting, 1115
  reserved, 1115
accounts table
  performance_schema, 4574
account_locked column
  user table, 1100
ACID, 2718, 2722, 6575
ACLs, 1077
Acl_cache_items_count status variable, 893
ACOS(), 1967
activate_all_roles_on_login system variable, 705
activating plugins, 994
ActiveState Perl, 275

adaptive flushing, 6575
adaptive hash index, 2735, 6575
add-drop-database option
  mysqldump, 447
  mysqlpump, 475
add-drop-table option
  mysqldump, 447
  mysqlpump, 475
add-drop-trigger option
  mysqldump, 447
add-drop-user option
  mysqlpump, 475
add-locks option
  mysqldump, 458
  mysqlpump, 475
ADDDATE(), 1978
adding
  character sets, 1796
  native functions, 5031
  new account privileges, 1112
  new functions, 5020
  new user privileges, 1112
  user-defined functions, 5021
addition (+), 1965
ADDTIME(), 1978
ADD_GDB_INDEX option
  CMake, 211
AdminAPI checking MySQL version, 3631
AdminAPI functions
  Cluster.checkInstanceState(), 3640
  Cluster.describe(), 3623
  Cluster.dissolve(), 3640
  Cluster.forceQuorumUsingPartitionOf(), 3636
  Cluster.options(), 3646
  Cluster.rejoinInstance(), 3636
  Cluster.removeInstance(), 3635
  Cluster.rescan(), 3639
  Cluster.setPrimaryInstance(), 3646
  Cluster.status(), 3623
  Cluster.switchToMultiPrimaryMode(), 3646
  Cluster.switchToSinglePrimaryMode(), 3646
  dba.checkInstanceConfiguration(), 3621
  dba.configureLocalInstance(), 3622
  dba.deleteSandboxInstance(), 3634
  dba.getCluster(), 3622
  dba.killSandboxInstance(), 3634
  dba.rebootClusterFromCompleteOutage(), 3638
  dba.startSandboxInstance(), 3634
  dba.stopSandboxInstance(), 3634
administration
  server, 415
administration of NDB Cluster, 3966
administrative programs, 316
admin_address system variable, 706

admin_port system variable, 707
AES_DECRYPT(), 2057
AES_ENCRYPT(), 2057
After create
  thread state, 1656
age
  calculating, 290
ai-increment option
  ndb_import, 3994
ai-offset option
  ndb_import, 3995
ai-prefetch-sz option
  ndb_import, 3995
AIO, 6576
alias names
  case sensitivity, 1684
aliases
  for expressions, 2189
  for tables, 2423
  in GROUP BY clauses, 2189
  names, 1679
  on expressions, 2422
ALL, 2441
  SELECT modifier, 2427
ALL join type
  optimizer, 1575
ALL privilege, 1082
ALL PRIVILEGES privilege, 1082
all-databases option
  mysqlcheck, 429
  mysqldump, 455
  mysqlpump, 475
all-in-1 option
  mysqlcheck, 429
all-tablespaces option
  mysqldump, 447
allow-keywords option
  mysqldump, 447
allow-mismatches option
  innochecksum, 512
allow-suspicious-udfs option
  mysqld, 675
ALLOW_INVALID_DATES SQL mode, 916
ALTER COLUMN, 2261
ALTER DATABASE, 2244
ALTER EVENT, 2245
  and replication, 3375
ALTER FUNCTION, 2247
ALTER INSTANCE, 2247
ALTER LOGFILE GROUP, 2248
  (see also NDB Cluster Disk Data)
ALTER privilege, 1082
ALTER PROCEDURE, 2249
ALTER RESOURCE GROUP statement, 2604

ALTER ROUTINE privilege, 1082
ALTER SCHEMA, 2244
ALTER SERVER, 2250
ALTER TABLE, 2250, 2261, 5647
  and replication log tables, 3317
  monitoring, 3055
  ROW_FORMAT, 2897
ALTER TABLESPACE
  general tablespace, 2274
  NDB Cluster Disk Data, 2274
  undo tablespace, 2274
ALTER USER statement, 1135, 2560
ALTER VIEW, 2276
altering
  database, 2244
  schema, 2244
altering table
  thread state, 1657
altering user accounts, 2560
analyze option
  myisamchk, 525
  mysqlcheck, 429
ANALYZE TABLE
  and partitioning, 4306
ANALYZE TABLE statement, 2608
Analyzing
  thread state, 1656
AND
  bitwise, 2053
  logical, 1927
anonymous user, 1108, 1110
ANSI mode
  running, 57
ansi option
  mysqld, 675
ANSI SQL mode, 915, 921
ANSI_QUOTES SQL mode, 916
ANY, 2440
ANY_VALUE(), 2214
Apache, 311
API node (NDB Cluster)
  defined, 3654
API nodes (see SQL nodes)
APIs, 4781
  list of, 72
  Perl, 4940
append option
  ndb_restore, 4025
application programming interface (API), 6576
APPLICATION_PASSWORD_ADMIN privilege, 1088
apply, 6576
apply-slave-statements option
  mysqldump, 449
apply_status table (OBSOLETE), 4223

(see also NDB Cluster replication)
approximate-value literals, 2228
approximate-value numeric literals, 1672, 2229
Arbitration, 3824
ArbitrationDelay, 3773, 3864
ArbitrationRank, 3772, 3864
ArbitrationTimeout, 3824
arbitrator_validity_detail
ndbinfo table, 4122
arbitrator_validity_summary
ndbinfo table, 4122
ARCHIVE storage engine, 3119, 3139
argument processing, 5026
arithmetic expressions, 1965
arithmetic functions, 2043
arithmetic operators, 2043
.ARM file, 6575
array
JSON, 1874
.ARZ file, 6575
AS, 2423, 2430
ASCII(), 1934
ASIN(), 1967
assigning roles, 2603
assignment operator
:=, 1928
=, 1929
assignment operators, 1928
ASYMMETRIC_DECRYPT(), 2167
ASYMMETRIC_DERIVE(), 2168
ASYMMETRIC_ENCRYPT(), 2168
ASYMMETRIC_SIGN(), 2169
ASYMMETRIC_VERIFY(), 2169
asynchronous C API
data structures, 4913
functions, 4913
asynchronous I/O, 2855, 6576
asynchronous interface
C API, 4908
asynchronous replication (see NDB Cluster replication)
ATAN(), 1968
ATAN2(), 1968
atomic, 6576
atomic DDL, 2238, 6576
atomic instruction, 6576
attackers
security against, 1071
attribute demotion
replication, 3369
attribute promotion
replication, 3369
attributes
resource groups, 1647
audit API UDFs

audit_api_message_emit_udf(), 1383
audit log filtering
legacy mode, 1346, 1358, 1361
rule based, 1344
audit log filtering UDFs
audit_log_encryption_password_get(), 1365
audit_log_encryption_password_set(), 1366
audit_log_filter_flush(), 1366
audit_log_filter_remove_filter(), 1367
audit_log_filter_remove_user(), 1367
audit_log_filter_set_filter(), 1368
audit_log_filter_set_user(), 1368
audit_log_read(), 1369
audit_log_read_bookmark(), 1369
audit plugin
sha2_cache_cleaner, 1199
audit plugins, 4948
audit-log option
mysqld, 1371
AUDIT_ADMIN privilege, 1088
audit_api_message_emit_udf() audit API UDF, 1383
audit_log plugin, 1312
installing, 1313
audit_log_buffer_size system variable, 1372
audit_log_compression system variable, 1372
audit_log_connection_policy system variable, 1373
audit_log_current_session system variable, 1373
Audit_log_current_size status variable, 1382
audit_log_encryption system variable, 1374
audit_log_encryption_password_get() audit log filtering UDF, 1365
audit_log_encryption_password_set() audit log filtering UDF, 1366
Audit_log_events status variable, 1382
Audit_log_events_filtered status variable, 1382
Audit_log_events_lost status variable, 1382
Audit_log_events_written status variable, 1382
Audit_log_event_max_drop_size status variable, 1382
audit_log_exclude_accounts system variable, 1374
audit_log_file system variable, 1375
audit_log_filter table
system table, 948
audit_log_filter_flush() audit log filtering UDF, 1366
audit_log_filter_id system variable, 1375
audit_log_filter_remove_filter() audit log filtering UDF, 1367
audit_log_filter_remove_user() audit log filtering UDF, 1367
audit_log_filter_set_filter() audit log filtering UDF, 1368
audit_log_filter_set_user() audit log filtering UDF, 1368
audit_log_flush system variable, 1376
audit_log_format system variable, 1376
audit_log_include_accounts system variable, 1376
audit_log_password_history_keep_days system variable, 1377
audit_log_policy system variable, 1378
audit_log_read() audit log filtering UDF, 1369
audit_log_read_bookmark() audit log filtering UDF, 1369
audit_log_read_buffer_size system variable, 1379

audit_log_rotate_on_size system variable, 1380
audit_log_statement_policy system variable, 1380
audit_log_strategy system variable, 1381
Audit_log_total_size status variable, 1382
audit_log_user table
  system table, 948
Audit_log_write_waits status variable, 1382
authentication
  for the InnoDB memcached interface, 3087
  LDAP, 1221
  SASL, 1221
authentication plugin
  authentication_ldap_sasl, 1221
  authentication_ldap_sasl_client, 1221
  authentication_ldap_simple, 1221
  authentication_pam, 1205
  authentication_windows, 1216
  authentication_windows_client, 1216
  auth_socket, 1237
  auth_test_plugin, 1240
  caching_sha2_password, 1194
  mysql_clear_password, 1204
  mysql_clear_plugin, 1221
  mysql_native_password, 1194
  mysql_no_login, 1234
  sha256_password, 1200
  test_plugin_server, 1240
authentication plugins, 4948
  client/server compatibility, 1150
  client/server protocol, 1151
AUTHENTICATION_LDAP_CLIENT_LOG environment variable, 1248
authentication_ldap_sasl_auth_method_name system variable, 1243
authentication_ldap_sasl_bind_base_dn system variable, 1243
authentication_ldap_sasl_bind_root_dn system variable, 1244
authentication_ldap_sasl_bind_root_pwd system variable, 1244
authentication_ldap_sasl_ca_path system variable, 1245
authentication_ldap_sasl_group_search_attr system variable, 1245
authentication_ldap_sasl_group_search_filter system variable, 1246
authentication_ldap_sasl_init_pool_size system variable, 1246
authentication_ldap_sasl_log_status system variable, 1247
authentication_ldap_sasl_max_pool_size system variable, 1248
authentication_ldap_sasl_server_host system variable, 1248
authentication_ldap_sasl_server_port system variable, 1249
authentication_ldap_sasl_tls system variable, 1249
authentication_ldap_sasl_user_search_attr system variable, 1250
authentication_ldap_simple_auth_method_name system variable, 1250
authentication_ldap_simple_bind_base_dn system variable, 1251
authentication_ldap_simple_bind_root_dn system variable, 1251
authentication_ldap_simple_bind_root_pwd system variable, 1252
authentication_ldap_simple_ca_path system variable, 1252
authentication_ldap_simple_group_search_attr system variable, 1253
authentication_ldap_simple_group_search_filter system variable, 1253
authentication_ldap_simple_init_pool_size system variable, 1254
authentication_ldap_simple_log_status system variable, 1255

authentication_ldap_simple_max_pool_size system variable, 1255
authentication_ldap_simple_server_host system variable, 1256
authentication_ldap_simple_server_port system variable, 1257
authentication_ldap_simple_tls system variable, 1258
authentication_ldap_simple_user_search_attr system variable, 1258
authentication_pam authentication plugin, 1205
AUTHENTICATION_PAM_LOG environment variable, 578, 1215
authentication_windows authentication plugin, 1216
authentication_windows_client authentication plugin, 1216
authentication_windows_log_level system variable, 707
authentication_windows_use_principal_name system variable, 708
auth_socket authentication plugin, 1237
auth_test_plugin authentication plugin, 1240
auto-generate-sql option
  mysqlslap, 498
auto-generate-sql-add-autoincrement option
  mysqlslap, 498
auto-generate-sql-execute-number option
  mysqlslap, 498
auto-generate-sql-guid-primary option
  mysqlslap, 499
auto-generate-sql-load-type option
  mysqlslap, 499
auto-generate-sql-secondary-indexes option
  mysqlslap, 499
auto-generate-sql-unique-query-number option
  mysqlslap, 499
auto-generate-sql-unique-write-number option
  mysqlslap, 499
auto-generate-sql-write-number option
  mysqlslap, 499
auto-inc lock, 2810
auto-increment, 2745, 2745, 2751, 6577
auto-increment locking, 6577
auto-rehash option
  mysql, 389
auto-repair option
  mysqlcheck, 429
auto-vertical-output option
  mysql, 389
auto.cnf file, 3205
  and SHOW SLAVE HOSTS statement, 2669
autocommit, 6577
autocommit mode, 2818
autocommit system variable, 708
automatic_sp_privileges system variable, 709
AutoReconnect
  API and SQL nodes, 3867
autowrapped JSON values, 1879
auto_generate_certs system variable, 709
AUTO_INCREMENT, 309, 1830
  and NULL values, 5642
  and replication, 3364
auto_increment_increment system variable, 3214

auto_increment_offset system variable, 3217
availability, 6577
AVG(), 2172
AVG(DISTINCT), 2172
avoid_temporal_upgrade system variable, 710

B

B-tree, 6577
B-tree indexes, 1530, 2758
background threads, 2856
  read, 2855
  write, 2855
backslash
  escape character, 1669
backspace (\b), 1671, 2142, 2406
backticks, 6578
backup, 6578
BACKUP Events (NDB Cluster), 4095
backup identifiers
  native backup and restore, 4080
backup option
  myisamchk, 523
  myisampack, 535
BackupDataBufferSize, 3833, 4082
BackupDataDir, 3781
BackupDiskWriteSpeedPct, 3833
backupid option
  ndb_restore, 4026
BackupLogBufferSize, 3834, 4082
BackupMaxWriteSize, 3835, 4082
BackupMemory, 3834, 4082
BackupReportFrequency, 3835
backups, 1419, 5052
  databases and tables, 436, 470
  in NDB Cluster, 4021, 4078, 4078, 4079, 4082
  in NDB Cluster replication, 4230
  InnoDB, 3068
  with mysqldump, 1429
backups, troubleshooting
  in NDB Cluster, 4083
BackupWriteSize, 3835, 4082
BACKUP_ADMIN privilege, 1088
backup_path option
  ndb_restore, 4025
back_log system variable, 710
base column, 6578
base64-output option
  mysqlbinlog, 551
basedir option
  mysql.server, 363
  mysqld, 675
  mysqld_safe, 356
basedir system variable, 711

batch mode, 301
batch option
  mysql, 389
batch SQL files, 385
BatchByteSize, 3864
Batched Key Access
  optimization, 1481, 1482
batched updates (NDB Cluster Replication), 4227
BatchSize, 3865
BatchSizePerLocalScan, 3792
BEGIN, 2470, 2520
  labels, 2520
  XA transactions, 2486
BENCHMARK(), 2065
benchmarks, 1652
beta, 6578
BETWEEN ... AND, 1922
big5, 5071
BIGINT data type, 1820
big_tables system variable, 711
BIN(), 1934
BINARY, 2029
binary collation, 1763
BINARY data type, 1826, 1845
binary distributions
  installing, 93
binary log, 971, 6578
  C API, 4922
  event groups, 2505
binary log C API
  data structures, 4922
  functions, 4924
binary log encryption, 3333
binary logging
  ALTER USER, 2573
  and NDB Cluster, 3683
  CREATE USER, 2584
binary-as-hex option
  mysql, 389
binary-mode option
  mysql, 389
bind-address option
  mysql, 389
  mysqladmin, 420
  mysqlbinlog, 552
  mysqlcheck, 429
  mysqldump, 442
  mysqlimport, 464
  mysqlpump, 475
  mysqlshow, 490
  mysql_upgrade, 379
bind-address option (ndb_mgmd), 3959
bind_address system variable, 712
binlog, 6579

Binlog Dump
  thread command, 1654
BINLOG statement, 2689
  mysqlbinlog output, 564
binlog-checksum option
  mysqld, 3268
binlog-do-db option
  mysqld, 3266
binlog-ignore-db option
  mysqld, 3268
binlog-row-event-max-size option
  mysqlbinlog, 552
  mysqld, 3264
BINLOG_ADMIN privilege, 1088
binlog_cache_size system variable, 3270
binlog_checksum system variable, 3270
binlog_direct_non_transactional_updates system variable, 3271
binlog_encryption system variable, 3272
BINLOG_ENCRYPTION_ADMIN privilege, 1088
binlog_error_action system variable, 3272
binlog_expire_logs_seconds, 3273
binlog_format
  BLACKHOLE, 3365
binlog_format system variable, 3274
binlog_group_commit_sync_delay, 3275
binlog_group_commit_sync_no_delay_count, 3276
binlog_gtid_simple_recovery, 3293
binlog_index table (OBSOLETE) (see NDB Cluster replication)
binlog_max_flush_queue_time system variable, 3277
binlog_order_commits system variable, 3277
binlog_rotate_encryption_master_key_at_startup system variable, 3278
binlog_rows_query_log_events system variable, 3282
binlog_row_event_max_size system variable, 3278
binlog_row_image system variable, 3279
binlog_row_metadata system variable, 3280
binlog_row_value_options system variable, 3281
binlog_stmt_cache_size system variable, 3282
binlog_transaction_dependency_history_size system variable, 3284
binlog_transaction_dependency_tracking system variable, 3283
BIN_TO_UUID(), 2215
BIT data type, 1819
bit functions, 2043
  example, 308
bit operations
  bit-value literals, 1679
  hexadecimal literals, 1677
bit operators, 2043
bit-value literal introducer, 1678
bit-value literals, 1677
  bit operations, 1679
BIT_AND(), 2172
BIT_COUNT, 308
BIT_COUNT(), 2055
BIT_LENGTH(), 1935

BIT_OR, 308
BIT_OR(), 2173
BIT_XOR(), 2173
BLACKHOLE
  binlog_format, 3365
  replication, 3365
BLACKHOLE storage engine, 3119, 3141
blind query expansion, 2008, 6579
BLOB columns
  default values, 1847
  indexing, 1526, 2314
  inserting binary data, 1672
  size, 1896
BLOB data type, 1827, 1846
blob-info option
  ndb_desc, 3987
Block Nested-Loop
  optimization, 1481, 1482
Block Nested-Loop join algorithm, 1468
block-search option
  myisamchk, 525
blocks
  ndbinfo table, 4123
block_encryption_mode system variable, 713
BOOL data type, 1819
BOOLEAN data type, 1819
boolean literals, 1679
boolean options, 327
Boolean search, 2002
bottleneck, 6579
bounce, 6579
brackets
  square, 1818
browser-start-page option
  ndb_setup.py, 4049
buddy allocator, 3020, 6579
buffer, 6579
buffer pool, 1615, 2839, 2844, 2845, 2846, 2847, 2849, 6579
  and compressed tables, 2884
  monitoring, 2729, 2843, 2851
buffer pool instance, 6580
buffer sizes, 1615, 2839
  client, 4781
bugs
  known, 5648
  NDB Cluster
    reporting, 3989
  reporting, 2, 52
bugs database, 52
bugs.mysql.com, 52
BuildIndexThreads, 3838
building
  client programs, 4788
BUILD_CONFIG option

CMake, 206
built-in, 6580
bulk loading
for InnoDB tables, 1557
for MyISAM tables, 1565
bulk_insert_buffer_size system variable, 714, 3127
BUNDLE_RUNTIME_LIBRARIES option
CMake, 207
business rules, 6580

C

C API, 4781
asynchronous interface, 4908
binary log, 4922
data structures, 4793
data types, 4785
example programs, 4787
FAQ, 5082
functions, 4799
linking problems, 4790
C++, 4784
C:\my.cnf option file, 1063
ca-certs-file option
ndb_setup.py, 4050
cache, 6580
CACHE INDEX statement, 2689
caches
clearing, 2691
cache_policies table, 3106
caching_sha2_password
authentication method unknown to the client error, 250
cannot be loaded error, 250
is not supported error, 250
caching_sha2_password authentication plugin, 1194
compatibility, 249
caching_sha2_password_auto_generate_rsa_keys system variable, 715
caching_sha2_password_private_key_path system variable, 715
caching_sha2_password_public_key_path system variable, 716
Caching_sha2_password_rsa_public_key status variable, 893
calculating
aggregate value for a set of rows, 2171
cardinality, 2657
dates, 290
calendar, 1997
CALL, 2381
calling sequences for aggregate functions
UDF, 5025
calling sequences for simple functions
UDF, 5023
can't create/write to file, 5628
Can't reopen table
error message, 5647
CAN_ACCESS_COLUMN(), 2213

CAN_ACCESS_DATABASE(), 2213
CAN_ACCESS_TABLE(), 2213
CAN_ACCESS_VIEW(), 2213
cardinality, 1502, 6580
carriage return (\r), 1671, 2142, 2406
CASE, 1930, 2523
case sensitivity
  access checking, 1104
  account names, 1105
  in identifiers, 1684
  in names, 1684
  in searches, 5639
  in string comparisons, 1948
  of database names, 58
  of replication filtering options, 3324
  of table names, 58
CAST, 2030
cast functions, 2026
cast operators, 2026
casts, 1914, 1920, 2026
catalogs table
  data dictionary table, 943
CC environment variable, 226, 578
CEIL(), 1968
CEILING(), 1968
cert-file option
  ndb_setup.py, 4050
.cfg file, 6580
cflags option
  mysql_config, 574
change buffer, 2731, 6581
  monitoring, 2733
change buffering, 6581
  disabling, 2732
CHANGE MASTER TO, 2494
  in NDB Cluster, 4225
CHANGE REPLICATION FILTER, 2501
Change user
  thread command, 1654
changes to privileges, 1134
changing
  column, 2259
  field, 2259
  socket location, 362, 5638
  table, 2250, 2261, 5647
Changing master
  thread state, 1665
channel, 3314
  commands, 3314
CHAR data type, 1825, 1843
CHAR VARYING data type, 1826
CHAR(), 1935
CHARACTER data type, 1825
character set introducer, 1747

character set repertoire, 1772
character sets, 1732
  adding, 1796
  and replication, 3365
  Asian, 1789
  Baltic, 1788
  binary, 1794
  Central European, 1787
  Cyrillic, 1789
  Middle East, 1788
  repertoire, 1735
  restrictions, 1795
  South European, 1788
  Unicode, 1778
  West European, 1785
CHARACTER VARYING data type, 1826
character-set-client-handshake option
  mysqld, 675
character-sets-dir option
  myisamchk, 523
  myisampack, 535
  mysql, 389
  mysqladmin, 420
  mysqlbinlog, 552
  mysqlcheck, 430
  mysqldump, 449
  mysqlimport, 464
  mysqlpump, 475
  mysqlshow, 490
  mysql_upgrade, 380
  ndb_move_data, 4011
character-sets-dir option (NDB Cluster programs), 4066, 4066
characters
  multibyte, 1799
CHARACTER_LENGTH(), 1935
CHARACTER_SETS
  INFORMATION_SCHEMA table, 4373
character_sets table
  data dictionary table, 943
character_sets_dir system variable, 719
character_set_client system variable, 716
character_set_connection system variable, 717
character_set_database system variable, 717
character_set_filesystem system variable, 718
character_set_results system variable, 718
character_set_server system variable, 718
character_set_system system variable, 719
charset command
  mysql, 400
charset option
  comp_err, 368
CHARSET(), 2066
CHAR_LENGTH(), 1935
CHECK constraints

ALTER TABLE, 2263
CREATE TABLE, 2346
RENAME TABLE, 2379
SHOW CREATE TABLE, 2640
check option
myisamchk, 522
mysqlcheck, 430
check options
myisamchk, 522
CHECK TABLE
and partitioning, 4306
CHECK TABLE statement, 2611
check-only-changed option
myisamchk, 522
mysqlcheck, 430
check-orphans option
ndb_blob_tool, 3969
check-upgrade option
mysqlcheck, 430
checking
tables for errors, 1439
Checking master version
thread state, 1663
checking permissions
thread state, 1656
Checking table
thread state, 1656
checkpoint, 6581
CHECKPOINT Events (NDB Cluster), 4090
Checksum, 3935
checksum, 6581
Checksum (NDB Cluster), 3940
checksum errors, 187
CHECKSUM TABLE
and replication, 3365
CHECKSUM TABLE statement, 2615
CHECK_CONSTRAINTS
INFORMATION_SCHEMA table, 4374
check_constraints table
data dictionary table, 943
check_proxy_users system variable, 719, 1160
child table, 6582
Chinese, Japanese, Korean character sets
frequently asked questions, 5071
choosing
a MySQL version, 78
data types, 1897
chroot option
mysqld, 676
circular replication
in NDB Cluster, 4215, 4237, 4241
CJK (Chinese, Japanese, Korean)
Access, PHP, etc., 5071
availability of specific characters, 5071

big5, 5071
character sets available, 5071
characters displayed as question marks, 5071
CJKV, 5071
collations, 5071, 5071
conversion problems with Japanese character sets, 5071
data truncation, 5071
Database and table names, 5071
documentation in Chinese, 5071
documentation in Japanese, 5071
documentation in Korean, 5071
FAQ, 5071
gb2312, gbk, 5071
Japanese character sets, 5071
Korean character set, 5071
LIKE and FULLTEXT, 5071
MySQL 4.0 behavior, 5071
ORDER BY treatment, 5071, 5071
problems with Access, PHP, etc., 5071
problems with Big5 character sets (Chinese), 5071
problems with data truncation, 5071
problems with euckr character set (Korean), 5071
problems with GB character sets (Chinese), 5071
problems with LIKE and FULLTEXT, 5071
problems with Yen sign (Japanese), 5071
rejected characters, 5071
sort order problems, 5071, 5071
sorting problems, 5071, 5071
testing availability of characters, 5071
Unicode collations, 5071
Vietnamese, 5071
Yen sign, 5071
clean page, 6582
clean shutdown, 940, 1058, 3380, 6582
cleaning up
thread state, 1656
clear command
mysql, 401
Clearing
thread state, 1666
clearing
caches, 2691
client, 6582
client connections, 1642
client programs, 315
building, 4788
client tools, 4781
clients
debugging, 5039
threaded, 4792
CLONE, 2624
clone plugin, 1030
clone_progress table, 1044
clone_status table, 1044

cloning compressed data, 1040
cloning data locally, 1033
cloning encrypted data, 1040
cloning for replication, 1040
cloning remote data, 1034, 1037
cloning to a named directory, 1038
Com_clone status variable, 1048
configuring an encrypted connection, 1039
directories and files, 1042
failure handling, 1042
installing, 1032
limitations, 1054
monitoring, 1043
monitoring stage events, 1045
performance schema instruments, 1046
remote cloning prerequisites, 1035
stopping a cloning operation, 1048
system variables, 1048
CLONE_ADMIN privilege, 1089
clone_autotune_concurrency system variable, 1049
clone_buffer_size system variable, 1049
clone_ddl_timeout system variable, 1050
clone_enable_compression system variable, 1050
clone_max_concurrency system variable, 1051
clone_max_data_bandwidth system variable, 1051
clone_max_network_bandwidth system variable, 1052
clone_progress table, 1044
performance_schema, 4620
clone_ssl_ca system variable, 1052
clone_ssl_cert system variable, 1053
clone_ssl_key system variable, 1053
clone_status table, 1044
performance_schema, 4619
clone_valid_donor_list system variable, 1053
cloning data, 1030
cloning tables, 2337
CLOSE, 2529
Close stmt
thread command, 1654
closing
tables, 1544
closing tables
thread state, 1656
cluster database (OBSOLETE) (see NDB Cluster replication)
cluster logs, 4087, 4088
cluster.binlog_index table (OBSOLETE) (see NDB Cluster replication)
clustered index, 6582
InnoDB, 2757
Clustering (see NDB Cluster)
CLUSTERLOG commands (NDB Cluster), 4088
CLUSTERLOG STATISTICS command (NDB Cluster), 4095
cluster_locks
ndbinfo table, 4123
cluster_operations

ndbinfo table, 4124
cluster_replication database (OBSOLETE) (see NDB Cluster replication)
cluster_transactions
ndbinfo table, 4125
CMake
ADD_GDB_INDEX option, 211
BUILD_CONFIG option, 206
BUNDLE_RUNTIME_LIBRARIES option, 207
CMAKE_BUILD_TYPE option, 207
CMAKE_CXX_FLAGS option, 223
CMAKE_C_FLAGS option, 223
CMAKE_INSTALL_PREFIX option, 207
COMPILATION_COMMENT option, 211
COMPILATION_COMMENT_SERVER option, 211
CPACK_MONOLITHIC_INSTALL option, 207
DEFAULT_CHARSET option, 211
DEFAULT_COLLATION option, 212
DISABLE_PSI_COND option, 212
DISABLE_PSI_DATA_LOCK option, 213
DISABLE_PSI_ERROR option, 213
DISABLE_PSI_FILE option, 212
DISABLE_PSI_IDLE option, 212
DISABLE_PSI_MEMORY option, 212
DISABLE_PSI_METADATA option, 212
DISABLE_PSI_MUTEX option, 212
DISABLE_PSI_PS option, 213
DISABLE_PSI_RWLOCK option, 212
DISABLE_PSI_SOCKET option, 212
DISABLE_PSI_SP option, 212
DISABLE_PSI_STAGE option, 212
DISABLE_PSI_STATEMENT option, 212
DISABLE_PSI_STATEMENT_DIGEST option, 212
DISABLE_PSI_TABLE option, 212
DISABLE_PSI_THREAD option, 213
DISABLE_PSI_TRANSACTION option, 213
DISABLE_SHARED option, 213
DOWNLOAD_BOOST option, 213
DOWNLOAD_BOOST_TIMEOUT option, 213
ENABLED_LOCAL_INFILE option, 214
ENABLED_PROFILING option, 214
ENABLE_DEBUG_SYNC option, 213
ENABLE_DOWNLOADS option, 213
ENABLE_DTRACE option, 213
ENABLE_EXPERIMENTAL_SYSVARS option, 213
ENABLE_GCOV option, 214
ENABLE_GPROF option, 214
FORCE_INSOURCE_BUILD option, 207
FORCE_UNSUPPORTED_COMPILER option, 214
FPROFILE_GENERATE option, 214
FPROFILE_USE option, 214
IGNORE_AIO_CHECK option, 215
INSTALL_BINDIR option, 207
INSTALL_DOCDIR option, 207
INSTALL_DOCREADMEDIR option, 207

INSTALL_INCLUDEDIR option, 207
INSTALL_INFODIR option, 207
INSTALL_LAYOUT option, 208
INSTALL_LIBDIR option, 208
INSTALL_MANDIR option, 208
INSTALL_MYSQLKEYRINGDIR option, 208
INSTALL_MYSQLSHAREDIR option, 208
INSTALL_MYSQLTESTDIR option, 208
INSTALL_PKGCONFIGDIR option, 208
INSTALL_PLUGINDIR option, 208
INSTALL_SBINDIR option, 209
INSTALL_SECURE_FILE_PRIVDIR option, 209
INSTALL_SHAREDIR option, 209
INSTALL_STATIC_LIBRARIES option, 209
INSTALL_SUPPORTFILESDIR option, 209
LINK_RANDOMIZE option, 209
LINK_RANDOMIZE_SEED option, 209
MAX_INDEXES option, 215
MEMCACHED_HOME option, 224
MUTEX_TYPE option, 215
MYSQLX_TCP_PORT option, 215
MYSQLX_UNIX_ADDR option, 215
MYSQL_DATADIR option, 209
MYSQL_MAINTAINER_MODE option, 215
MYSQL_PROJECT_NAME option, 215
MYSQL_TCP_PORT option, 215
MYSQL_UNIX_ADDR option, 215
ODBC_INCLUDES option, 209
ODBC_LIB_DIR option, 209
OPTIMIZER_TRACE option, 216
options, 200
REPRODUCIBLE_BUILD option, 216
running after prior invocation, 196, 225
SYSCONFDIR option, 209
SYSTEMD_PID_DIR option, 209
SYSTEMD_SERVICE_NAME option, 210
TMPDIR option, 210
USE_LD_GOLD option, 216
USE_LD_LLD option, 216
VERSION file, 227
WIN_DEBUG_NO_INLINE option, 216
WITH_ANT option, 216
WITH_ASAN option, 216
WITH_ASAN_SCOPE option, 216
WITH_AUTHENTICATION_LDAP option, 216
WITH_AUTHENTICATION_PAM option, 217
WITH_AWS_SDK option, 217
WITH_BOOST option, 217
WITH_BUNDLED_LIBEVENT option, 224
WITH_BUNDLED_MEMCACHED option, 224
WITH_CLASSPATH option, 225
WITH_CLIENT_PROTOCOL_TRACING option, 217
WITH_CURL option, 218
WITH_DEBUG option, 218

WITH_DEFAULT_COMPILER_OPTIONS option, 223
WITH_DEFAULT_FEATURE_SET option, 218
WITH_EDITLINE option, 218
WITH_ERROR_INSERT option, 225
WITH_GMOCK option, 219
WITH_ICU option, 218
WITH_INNODB_EXTRA_DEBUG option, 218
WITH_INNODB_MEMCACHED option, 219
WITH_JEMALLOC option, 219
WITH_KEYRING_TEST option, 219
WITH_LIBEVENT option, 219
WITH_LIBWRAP option, 219
WITH_LOCK_ORDER option, 219
WITH_LSAN option, 219
WITH_LTO option, 220
WITH_LZ4 option, 220
WITH_LZMA option, 221
WITH_MECAB option, 220
WITH_MSAN option, 220
WITH_MSCRT_DEBUG option, 220
WITH_MYSQLX option, 220
WITH_NDBCLUSTER option, 225
WITH_NDBCLUSTER_STORAGE_ENGINE option, 225
WITH_NDBMTD option, 225
WITH_NDB_BINLOG option, 225
WITH_NDB_DEBUG option, 225
WITH_NDB_JAVA option, 225
WITH_NDB_PORT option, 225
WITH_NDB_TEST option, 225
WITH_NUMA option, 220
WITH_PLUGIN_NDBCLUSTER option, 225
WITH_PROTOBUF option, 220
WITH_RAPID option, 221
WITH_RAPIDJSON option, 221
WITH_RE2 option, 221
WITH_ROUTER option, 221
WITH_SSL option, 221
WITH_SYSTEMD option, 222
WITH_SYSTEM_LIBS option, 222
WITH_TEST_TRACE_PLUGIN option, 222
WITH_TSAN option, 223
WITH_UBSAN option, 223
WITH_UNIT_TESTS option, 223
WITH_UNIXODBC option, 223
WITH_VALGRIND option, 223
WITH_ZLIB option, 223
WITH_ZSTD option, 223
CMakeCache.txt file, 225
CMAKE_BUILD_TYPE option
CMake, 207
CMAKE_CXX_FLAGS option
CMake, 223
CMAKE_C_FLAGS option
CMake, 223

CMAKE_INSTALL_PREFIX option
  CMake, 207
COALESCE(), 1923
coercibility
  collation, 1762
COERCIBILITY(), 2066
cold backup, 6582
collating
  strings, 1799
collation
  adding, 1799
  coercibility, 1762
  INFORMATION_SCHEMA, 1767
  modifying, 1800
COLLATION(), 2067
collations, 1732
  Asian, 1789
  Baltic, 1788
  binary, 1763, 1794
  Central European, 1787
  Cyrillic, 1789
  Middle East, 1788
  naming conventions, 1739
  NO PAD, 1765, 1779, 1844
  PAD SPACE, 1765, 1779, 1844
  South European, 1788
  Unicode, 1778
  West European, 1785
  _ai suffix, 1739
  _as suffix, 1739
  _bin suffix, 1739, 1763
  _ci suffix, 1739
  _ks suffix, 1739
  _ss suffix, 1739
COLLATIONS
  INFORMATION_SCHEMA table, 4374
collations table
  data dictionary table, 944
COLLATION_CHARACTER_SET_APPLICABILITY
  INFORMATION_SCHEMA table, 4375
collation_connection system variable, 720
collation_database system variable, 720
collation_server system variable, 721
color option
  ndb_top, 4058
column, 6582
  changing, 2259
  types, 1818
column alias
  problems, 5643
  quoting, 1680, 5643
column comments, 2315
column format, 2315
column index, 6583

column names
  case sensitivity, 1684
column prefix, 6583
column storage, 2316
column-names option
  mysql, 389
column-statistics option
  mysqldump, 457
  mysqlpump, 476
column-type-info option
  mysql, 389
columns
  displaying, 487
  indexes, 1525
  names, 1679
  other types, 1898
  selecting, 288
  storage requirements, 1893
COLUMNS
  INFORMATION_SCHEMA table, 4375
columns option
  mysqlimport, 464
columns partitioning, 4270
columns per table
  maximum, 1550
columns table
  data dictionary table, 944
columns_priv table
  system table, 946, 1096
COLUMN_PRIVILEGES
  INFORMATION_SCHEMA table, 4378
COLUMN_STATISTICS
  INFORMATION_SCHEMA table, 4379
column_statistics table
  system table, 944, 1612
column_type_elements table
  data dictionary table, 944
comma-separated values data, reading, 2404, 2429
command option precedence, 319
command options
  mysql, 385
  mysqladmin, 418
  mysqld, 673
command options (NDB Cluster)
  mysqld, 3872
  ndbd, 3947
  ndbinfo_select_all, 3956
  ndb_mgm, 3967
  ndb_mgmd, 3958
command syntax, 4
command-line history
  mysql, 406
command-line options (NDB Cluster), 4065
command-line tool, 124, 385

commands
  for binary distribution, 95
commands out of sync, 5628
comment syntax, 1727
comments
  adding, 1727
  starting, 62
comments option
  mysql, 390
  mysqldump, 447
COMMIT, 2470
  XA transactions, 2486
commit, 6583
commit option
  mysqlslap, 499
committing alter table to storage engine
  thread state, 1657
Committing events to binlog
  thread state, 1665
common table expressions, 305, 2458
  optimization, 1502, 1512
compact option
  mysqldump, 452
compact row format, 2895, 6583
comparison operators, 1919
comparisons
  access checking, 1104
  account names, 1105
compatibility
  with ODBC, 827, 1821, 1914, 1925, 2315, 2432
  with Oracle, 59, 2180, 2259, 2702
  with PostgreSQL, 60
  with standard SQL, 56
compatible option
  mysqldump, 452
COMPILATION_COMMENT option
  CMake, 211
COMPILATION_COMMENT_SERVER option
  CMake, 211
compiling
  user-defined functions, 5029
compiling clients
  on Unix, 4788
  on Windows, 4789
compiling MySQL server
  problems, 225
complete-insert option
  mysqldump, 453
  mysqlpump, 476
completion_type system variable, 721
component installing
  validate_password, 1267
component service
  status_variable_registration, 5011

component table
  system table, 946
component uninstalling
  validate_password, 1267
components
  installing, 2622
  security, 1193
  server, 989
  uninstalling, 2624
composite index, 6583
composite partitioning, 4283
compound statements, 2520
compress option, 340
  mysql, 390
  mysqladmin, 420
  mysqlbinlog, 552
  mysqlcheck, 430
  mysqldump, 442
  mysqlimport, 464
  mysqlpump, 476
  mysqlshow, 490
  mysqlslap, 499
  mysql_upgrade, 380
COMPRESS(), 2059
compress-output option
  mysqlpump, 476
compressed backup, 6583
compressed row format, 2896, 6584
compressed table, 6584
compressed tables, 534, 3130
CompressedBackup, 3807
CompressedLCP, 3808
compression, 2875, 2889, 6584
  algorithms, 2882
  application and schema design, 2880
  BLOBs, VARCHAR and TEXT, 2884
  buffer pool considerations, 2884
  compressed page size, 2881
  configuration characteristics, 2881
  connection, 349
  data and indexes, 2883
  data characteristics, 2878
  enabling for a table, 2875
  implementation, 2882
  information schema, 3019
  KEY_BLOCK_SIZE, 2881
  log file format, 2885
  modification log, 2883
  monitoring, 2881
  overflow pages, 2884
  overview, 2875
  tuning, 2877
  workload characteristics, 2880
compression failure, 6584

compression-algorithms option, 340
  mysql, 390
  mysqladmin, 420
  mysqlbinlog, 552
  mysqlcheck, 430
  mysqldump, 442
  mysqlimport, 464
  mysqlpump, 476
  mysqlshow, 490
  mysqlslap, 499
  mysql_upgrade, 380
Compression_algorithm status variable, 894
Compression_level status variable, 895
comp_err, 314, 367
  charset option, 368
  debug option, 368
  debug-info option, 368
  header_file option, 368
  help option, 368
  in_file option, 368
  name_file option, 368
  out_dir option, 368
  out_file option, 368
  statefile option, 368
  version option, 368
CONCAT(), 1935
concatenation
  string, 1669, 1935
CONCAT_WS(), 1936
concurrency, 2718, 6584
  of commits, 2951
  of threads, 3012
  tickets, 2953
concurrency option
  mysqlslap, 499
concurrent inserts, 1623, 1626
concurrent_insert system variable, 722
condition handling
  INOUT parameters, 2559
  OUT parameters, 2559
Conditions, 2531
conditions, 2649, 2686
cond_instances table
  performance_schema, 4536
config-cache option (ndb_mgmd), 3959
config-file option
  my_print_defaults, 575
  ndb_config, 3973
config-file option (ndb_mgmd), 3960
config.ini (NDB Cluster), 3728, 3759, 3760, 3966
configdir option (ndb_mgmd), 3961
ConfigGenerationNumber, 3870
configinfo option
  ndb_config, 3972

configuration
  NDB Cluster, 3740
  server, 582
configuration file, 6584
configuration files, 1164
configure option
  MySQLInstallerConsole, 124
configuring backups
  in NDB Cluster, 4082
configuring NDB Cluster, 3685, 3737, 3966, 4084
Configuring NDB Cluster (concepts), 3654
config_from_node option
  ndb_config, 3973
config_nodes
  ndbinfo table, 4126
config_options table, 3106
config_params
  ndbinfo table, 4127
config_values
  ndbinfo table, 4128
conflict detection status variables
  NDB Cluster Replication, 4247
conflict resolution
  and ndb_replication system table, 4244
  enabling, 4243
  in NDB Cluster Replication, 4241
  mysqld startup options, 4243
Connect
  thread command, 1654
connect command
  mysql, 401
CONNECT command (NDB Cluster), 4074
connect option
  ndb_restore, 4026
Connect Out
  thread command, 1654
connect-delay option (ndbd), 3949
connect-delay option (ndbmtd), 3949
connect-expired-password option
  mysql, 390
connect-retries option (NDB Cluster programs), 4066
connect-retries option (ndbd), 3949
connect-retries option (ndbmtd), 3949
connect-retries option (ndb_mgm), 3967
connect-retry-delay option (NDB Cluster programs), 4067
connect-retry-delay option (ndbd), 3949
connect-retry-delay option (ndbmtd), 3949
connect-string option (NDB Cluster programs), 4069
connect-timeout option
  mysql, 390
ConnectBackoffMaxTime, 3869
ConnectCheckIntervalDelay, 3817
connecting
  parameters, 344

remotely with SSH, 1192
to the server, 277, 340
using a URI-like connection string, 343, 347
using key-value pairs, 343, 348
verification, 1107
Connecting to master
thread state, 1663
connection
aborted, 5626
connection compression, 349
CONNECTION Events (NDB Cluster), 4090
connection string (see NDB Cluster)
connection-server-id option
mysqlbinlog, 553
connection-timeout option (ndb_error_reporter), 3990
ConnectionMap, 3861
connections option
ndb_config, 3974
ndb_import, 3995
CONNECTION_ADMIN privilege, 1089
CONNECTION_CONTROL plugin
installing, 1259
status variables, 1264
system variables, 1263
Connection_control_delay_generated status variable, 1264
connection_control_failed_connections_threshold system variable, 1263
CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS
INFORMATION_SCHEMA table, 4474
CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS plugin
installing, 1259
connection_control_max_connection_delay system variable, 1263
connection_control_min_connection_delay system variable, 1264
CONNECTION_ID(), 2067
Connector/C, 4781, 4784
FAQ, 5082
Connector/C++, 4781, 4784
Connector/J, 4781, 4784
Connector/NET, 4781, 4784
Connector/ODBC, 4781, 4785
Connector/Python, 4781, 4785
Connectors, 4781
connect_timeout system variable, 723
connect_timeout variable, 426
consistent read, 6585
consistent reads, 2819
console option
mysqld, 676
const table
optimizer, 1573, 2427
constant table, 1450
constraint, 6585
constraints, 62
foreign keys, 2339
containers table, 3106

continue option
  ndb_import, 3995
contributing companies
  list of, 73
contributors
  list of, 66
control flow functions, 1929
Control+C
  statement termination, 385, 397, 2465
CONV(), 1968
conventions
  syntax, 3
  typographical, 3
CONVERT, 2030
CONVERT TO, 2264
converting HEAP to ondisk
  thread state, 1656
CONVERT_TZ(), 1978
copy to tmp table
  thread state, 1656
copying databases, 272
copying tables, 2338
Copying to group table
  thread state, 1656
Copying to tmp table
  thread state, 1657
Copying to tmp table on disk
  thread state, 1657
core-file option, 2852
  mysqld, 676
core-file option (NDB Cluster programs), 4067
core-file-size option
  mysqld_safe, 356
core_file system variable, 723, 2852
correct-checksum option
  myisamchk, 523
correlated subqueries, 2443
corruption, 3113
  InnoDB, 3069
COS(), 1968
cost model
  optimizer, 1608
COT(), 1969
count option
  innochecksum, 510
  myisam_ftdump, 515
  mysqladmin, 421
  mysqlshow, 490
COUNT(), 2174
COUNT(DISTINCT), 2175
counter, 6585
counters
  ndbinfo table, 4130
counting

table rows, 296
covering index, 6585
CPACK_MONOLITHIC_INSTALL option
CMake, 207
CPU-bound, 6585
cpustat
ndbinfo table, 4132
cpustat_1sec
ndbinfo table, 4133
cpustat_20sec
ndbinfo table, 4133
cpustat_50ms
ndbinfo table, 4132
crash, 5033, 6586
recovery, 1439
repeated, 5634
replication, 3380
crash recovery, 6586
InnoDB, 3069, 3071
crash-safe replication, 3224, 3240, 3346
CrashOnCorruptedTuple, 3808
CRC32(), 1969
CREATE ... IF NOT EXISTS
and replication, 3366
CREATE DATABASE, 2276
Create DB
thread command, 1654
CREATE EVENT, 2277
and replication, 3375
CREATE FUNCTION, 2299
CREATE FUNCTION statement, 2620
CREATE INDEX, 2282
CREATE LOGFILE GROUP, 2297
(see also NDB Cluster Disk Data)
CREATE NODEGROUP command (NDB Cluster), 4076
create option
mysqlslap, 499
CREATE privilege, 1083
CREATE PROCEDURE, 2299
CREATE RESOURCE GROUP statement, 2605
CREATE ROLE privilege, 1083
CREATE ROLE statement, 2573
CREATE ROUTINE privilege, 1083
CREATE SCHEMA, 2276
CREATE SERVER, 2304
CREATE SPATIAL REFERENCE SYSTEM, 2305
CREATE TABLE, 2310
DIRECTORY options
and replication, 3371
KEY_BLOCK_SIZE, 2881
NDB_TABLE options, 2355
options for table compression, 2875
ROW_FORMAT, 2897
statement retention, 2335

table definition, 2335
CREATE TABLE ... SELECT
and replication, 3366
CREATE TABLESPACE, 2358
general tablespace, 2358
undo tablespace, 2358
CREATE TABLESPACE privilege, 1083
CREATE TEMPORARY TABLES privilege, 1083
CREATE TRIGGER, 2365
CREATE USER privilege, 1083
CREATE USER statement, 1112, 1135, 2574
CREATE VIEW, 2367
CREATE VIEW privilege, 1083
create-options option
mysqldump, 453
create-schema option
mysqlslap, 500
create_admin_listener_thread system variable, 723
CREATE_ASYMMETRIC_PRIV_KEY(), 2169
CREATE_ASYMMETRIC_PUB_KEY(), 2170
CREATE_DH_PARAMETERS(), 2170
CREATE_DIGEST(), 2170
create_synonym_db() procedure
sys schema, 4747
creating
bug reports, 52
database, 2276
databases, 282
default startup options, 321
function, 2620
schema, 2276
tables, 284
Creating index
thread state, 1657
creating roles, 2573
Creating sort index
thread state, 1657
creating table
thread state, 1657
Creating tmp table
thread state, 1657
creating user accounts, 2574
CROSS JOIN, 2430
CRUD, 6586
CR_ALREADY_CONNECTED error code, 5608
CR_AUTH_PLUGIN_CANNOT_LOAD error code, 5608
CR_AUTH_PLUGIN_ERR error code, 5608
CR_CANT_READ_CHARSET error code, 5605
CR_COMMANDS_OUT_OF_SYNC error code, 5605
CR_COMPRESSION_NOT_SUPPORTED error code, 5609
CR_COMPRESSION_WRONGLY_CONFIGURED error code, 5609
CR_CONNECTION_ERROR error code, 5604
CR_CONN_HOST_ERROR error code, 5604
CR_CONN_UNKNOW_PROTOCOL error code, 5607

CR_DATA_TRUNCATED error code, 5606
CR_DUPLICATE_CONNECTION_ATTR error code, 5608
CR_EMBEDDED_CONNECTION error code, 5605
CR_FETCH_CANCELED error code, 5607
CR_FILE_NAME_TOO_LONG error code, 5608
CR_INSECURE_API_ERR error code, 5608
CR_INVALID_BUFFER_USE error code, 5606
CR_INVALID_CONN_HANDLE error code, 5607
CR_INVALID_PARAMETER_NO error code, 5606
CR_IPSOCK_ERROR error code, 5604
CR_LOCALHOST_CONNECTION error code, 5605
CR_MALFORMED_PACKET error code, 5606
CR_NAMEDPIPEOPEN_ERROR error code, 5605
CR_NAMEDPIPESETSTATE_ERROR error code, 5605
CR_NAMEDPIPEWAIT_ERROR error code, 5605
CR_NAMEDPIPE_CONNECTION error code, 5605
CR_NET_PACKET_TOO_LARGE error code, 5605
CR_NEW_STMT_METADATA error code, 5608
CR_NOT_IMPLEMENTED error code, 5608
CR_NO_DATA error code, 5607
CR_NO_PARAMETERS_EXISTS error code, 5606
CR_NO_PREPARE_STMT error code, 5606
CR_NO_RESULT_SET error code, 5608
CR_NO_STMT_METADATA error code, 5608
CR_NULL_POINTER error code, 5606
CR_OUT_OF_MEMORY error code, 5604
CR_PARAMS_NOT_BOUND error code, 5606
CR_PROBE_MASTER_CONNECT error code, 5606
CR_PROBE_SLAVE_CONNECT error code, 5606
CR_PROBE_SLAVE_HOSTS error code, 5605
CR_PROBE_SLAVE_STATUS error code, 5605
CR_SERVER_GONE_ERROR, 5624
CR_SERVER_GONE_ERROR error code, 5604
CR_SERVER_HANDSHAKE_ERR error code, 5605
CR_SERVER_LOST error code, 5605
CR_SERVER_LOST_ERROR, 5624
CR_SERVER_LOST_EXTENDED error code, 5608
CR_SHARED_MEMORY_CONNECTION error code, 5606
CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR error code, 5607
CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR error code, 5607
CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR error code, 5607
CR_SHARED_MEMORY_CONNECT_MAP_ERROR error code, 5607
CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR error code, 5607
CR_SHARED_MEMORY_CONNECT_SET_ERROR error code, 5607
CR_SHARED_MEMORY_EVENT_ERROR error code, 5607
CR_SHARED_MEMORY_FILE_MAP_ERROR error code, 5607
CR_SHARED_MEMORY_MAP_ERROR error code, 5607
CR_SOCKET_CREATE_ERROR error code, 5604
CR_SSL_CONNECTION_ERROR error code, 5606
CR_SSL_FIPS_MODE_ERR error code, 5608
CR_STMT_CLOSED error code, 5608
CR_TCP_CONNECTION error code, 5605
CR_UNKNOWN_ERROR error code, 5604
CR_UNKNOWN_HOST error code, 5604

CR_UNSUPPORTED_PARAM_TYPE error code, 5606
CR_UNUSED_1 error code, 5607
CR_VERSION_ERROR error code, 5604
CR_WRONG_HOST_INFO error code, 5604
CR_WRONG_LICENSE error code, 5606
CSV data, reading, 2404, 2429
csv option
  mysqlslap, 500
CSV storage engine, 3119, 3137
cte_max_recursion_depth system variable, 724
CUME_DIST(), 2194
CURDATE(), 1979
current row
  window functions, 2200
CURRENT_DATE, 1979
CURRENT_ROLE(), 2067
CURRENT_TIME, 1979
CURRENT_TIMESTAMP, 1979
Current_tls_ca status variable, 896
Current_tls_capath status variable, 896
Current_tls_cert status variable, 896
Current_tls_cipher status variable, 897
Current_tls_ciphersuites status variable, 897
Current_tls_crl status variable, 897
Current_tls_crlpath status variable, 897
Current_tls_key status variable, 897
Current_tls_version status variable, 897
CURRENT_USER(), 2068
cursor, 6586
Cursors, 2528
CURTIME(), 1979
CXX environment variable, 226, 578
cxxflags option
  mysql_config, 574

D

Daemon
  thread command, 1654
daemon option (ndb_mgmd), 3961
daemon plugins, 4947
daemonize option
  mysqld, 676
daemon_memcached_enable_binlog system variable, 2932
daemon_memcached_engine_lib_name system variable, 2932
daemon_memcached_engine_lib_path system variable, 2933
daemon_memcached_option system variable, 2933
daemon_memcached_r_batch_size system variable, 2933
daemon_memcached_w_batch_size system variable, 2934
data
  importing, 410, 461
  loading into tables, 285
  retrieving, 286
  size, 1540

data dictionary, 2709, 2768, 6586
  benefits, 2709
  dictionary object cache, 2711
  INFORMATION_SCHEMA integration, 2712
  limitations, 2716
  metadata file removal, 2711
  operational implications, 2715
  schema, 2709
  transactional storage, 2711
data dictionary tables
  catalogs table, 943
  character_sets table, 943
  check_constraints table, 943
  collations table, 944
  columns table, 944
  column_type_elements table, 944
  dd_properties table, 944
  events table, 944
  foreign_keys table, 944
  foreign_key_column_usage table, 944
  indexes table, 944
  index_column_usage table, 944
  index_partitions table, 944
  index_stats table, 944
  innodb_ddl_log table, 944
  parameters table, 944
  parameter_type_elements table, 944
  resource_groups table, 944
  routines table, 944
  schemata table, 944
  st_spatial_reference_systems table, 944
  tables table, 944
  tablespaces table, 944
  tablespace_files table, 944
  table_partitions table, 944
  table_partition_values table, 944
  table_stats table, 944
  triggers table, 944
  view_routine_usage table, 945
  view_table_usage table, 945
data directory, 6586
  mysql_upgrade_info file, 248, 377
DATA DIRECTORY
  and replication, 3371
data encryption, 2795
data files, 6587
Data Masking plugin
  installing, 1400
  uninstalling, 1400
data node (NDB Cluster)
  defined, 3654
data nodes
  memory allocation, 3839
data nodes (NDB Cluster), 3947, 3956

Data on disk (NDB Cluster)
and INFORMATION_SCHEMA.FILES table, 4384
data structures
asynchronous C API, 4913
binary log C API, 4922
C API, 4793
prepared statement C API, 4874
Data truncation with CJK characters, 5071
data type
BIGINT, 1820
BINARY, 1826, 1845
BIT, 1819
BLOB, 1827, 1846
BOOL, 1819, 1898
BOOLEAN, 1819, 1898
CHAR, 1825, 1843
CHAR VARYING, 1826
CHARACTER, 1825
CHARACTER VARYING, 1826
DATE, 1822, 1834
DATETIME, 1822, 1834
DEC, 1821
DECIMAL, 1821, 2228
DOUBLE, 1821
DOUBLE PRECISION, 1822
ENUM, 1827, 1848
FIXED, 1821
FLOAT, 1821, 1821, 1821
GEOMETRY, 1855
GEOMETRYCOLLECTION, 1855
INT, 1820
INTEGER, 1820
LINESTRING, 1855
LONG, 1846
LONGBLOB, 1827
LONGTEXT, 1827
MEDIUMBLOB, 1827
MEDIUMINT, 1820
MEDIUMTEXT, 1827
MULTILINESTRING, 1855
MULTIPOINT, 1855
MULTIPOLYGON, 1855
NATIONAL CHAR, 1825
NATIONAL VARCHAR, 1826
NCHAR, 1825
NUMERIC, 1821
NVARCHAR, 1826
POINT, 1855
POLYGON, 1855
REAL, 1822
SET, 1828, 1851
SMALLINT, 1820
TEXT, 1827, 1846
TIME, 1823, 1836

TIMESTAMP, 1823, 1834
TINYBLOB, 1826
TINYINT, 1819
TINYTEXT, 1827
VARBINARY, 1826, 1845
VARCHAR, 1826, 1843
VARCHARACTER, 1826
YEAR, 1823, 1836
data types, 1818
C API, 4785
overview, 1818
data warehouse, 6587
data-at-rest encryption, 2795
data-file-length option
myisamchk, 523
database, 6587
altering, 2244
creating, 2276
deleting, 2371
renaming, 2378
Database information
obtaining, 2631
database metadata, 4370
database names
case sensitivity, 58, 1684
database objects
metadata, 1737
database option
mysql, 390
mysqlbinlog, 553
ndb_blob_tool, 3969
ndb_desc, 3987
ndb_move_data, 4011
ndb_show_tables, 4052
database option (ndb_index_stat), 4006
DATABASE(), 2069
databases
backups, 1419
copying, 272
creating, 282, 2276
defined, 5
displaying, 487
dumping, 436, 470
information about, 300
names, 1679
replicating, 3159
selecting, 283, 2707
symbolic links, 1633
using, 282
databases option
mysqlcheck, 430
mysqldump, 455
mysqlpump, 476
DataDir, 3773, 3780

datadir option
  mysql.server, 363
  mysqld, 677
  mysqld_safe, 356
  mysql_ssl_rsa_setup, 374
datadir system variable, 724
DataMemory, 3781
data_locks table
  performance_schema, 4598, 4692
data_lock_waits table
  performance_schema, 4601
DATE, 5640
date and time functions, 1975
Date and Time types, 1833
date calculations, 290
DATE columns
  problems, 5640
DATE data type, 1822, 1834
date data types
  storage requirements, 1895
date literals, 1672
date values
  problems, 1835
DATE(), 1979
DATEDIFF(), 1979
dates
  used with partitioning, 4261
  used with partitioning (examples), 4265, 4278, 4283, 4310
DATETIME data type, 1822, 1834
datetime_format system variable, 725
DATE_ADD(), 1980
date_format system variable, 725
DATE_FORMAT(), 1980
DATE_SUB(), 1980, 1982
DAY(), 1982, 2062
Daylight Saving Time, 934, 1538, 1994
DAYNAME(), 1982
DAYOFMONTH(), 1982
DAYOFWEEK(), 1982
DAYOFYEAR(), 1983
db table
  sorting, 1111
  system table, 240, 946, 1096
db-workers option
  ndb_import, 3996
DBI interface, 4940
DBI->quote, 1672
DBI->trace, 5037
DBI/DBD interface, 4940
DBI_TRACE environment variable, 578, 5037
DBI_USER environment variable, 578
DBUG package, 5046
DCL, 2585, 2598, 6587
DDL, 2238, 2238, 6587

DDL log, 987
ddl-rewriter option
  mysqld, 1018
ddl_rewriter plugin, 1016
  installing, 1017
dd_properties table
  data dictionary table, 944
deadlock, 1622, 2479, 2824, 2828, 2829, 2830, 2830, 2994, 4689, 6588
deadlock detection, 6588
DEALLOCATE PREPARE, 2515, 2519
deb file
  MySQL APT Repository, 160
  MySQL SLES Repository, 160
Debug
  thread command, 1654
debug option
  comp_err, 368
  ibd2sdi, 507
  myisamchk, 519
  myisampack, 535
  mysql, 390
  mysqladmin, 421
  mysqlbinlog, 554
  mysqlcheck, 430
  mysqld, 677
  mysqldump, 448
  mysqldumpslow, 572
  mysqlimport, 464
  mysqlpump, 476
  mysqlshow, 490
  mysqlslap, 500
  mysql_config_editor, 544
  mysql_upgrade, 380
  my_print_defaults, 575
debug option (NDB Cluster programs), 4067
debug system variable, 725
debug-check option
  mysql, 390
  mysqladmin, 421
  mysqlbinlog, 554
  mysqlcheck, 430
  mysqldump, 448
  mysqlimport, 464
  mysqlpump, 476
  mysqlshow, 490
  mysqlslap, 500
  mysql_upgrade, 380
debug-info option
  comp_err, 368
  mysql, 391
  mysqladmin, 421
  mysqlbinlog, 554
  mysqlcheck, 430
  mysqldump, 448

mysqlimport, 464
mysqlpump, 477
mysqlshow, 490
mysqlslap, 500
mysql_upgrade, 380
debug-level option
ndb_setup.py, 4050
debug-sync-timeout option
mysqld, 678
debugging
client, 5039
server, 5033
debugging support, 200
debug_sync system variable, 726
DEC data type, 1821
decimal arithmetic, 2228
DECIMAL data type, 1821, 2228
decimal point, 1818
DECLARE, 2521
DECODE(), 2059
decode_bits myisamchk variable, 521
DEFAULT
constraint, 64
default
privileges, 240
default account, 240
default host name, 340
default installation location, 93
default options, 321
default proxy user, 1157
default role
ALTER USER, 2568
CREATE USER statement, 2578
default roles, 2600
DEFAULT value clause, 1890, 2315
default values, 1890, 2315, 2393
BLOB and TEXT columns, 1847
explicit, 1890
implicit, 1890
suppression, 64
DEFAULT(), 2216
default-auth option, 333
mysql, 391
mysqladmin, 421
mysqlbinlog, 554
mysqlcheck, 431
mysqldump, 443
mysqlimport, 464
mysqlpump, 477
mysqlshow, 490
mysqlslap, 500
mysql_upgrade, 380
default-character-set option
mysql, 391

mysqladmin, 421
mysqlcheck, 430
mysqldump, 449
mysqlimport, 464
mysqlpump, 477
mysqlshow, 490
mysql_upgrade, 380
default-parallelism option
mysqlpump, 477
default-time-zone option
mysqld, 678
DefaultHashMapSize, 3796, 3868
DefaultOperationRedoProblemAction
API and SQL nodes, 3867
defaults-extra-file option, 326
myisamchk, 520
mysql, 391
mysqladmin, 421
mysqlbinlog, 554
mysqlcheck, 431
mysqld, 678
mysqldump, 446
mysqld_multi, 364
mysqld_safe, 356
mysqlimport, 465
mysqlpump, 477
mysqlshow, 490
mysqlslap, 500
mysql_secure_installation, 370
mysql_upgrade, 380
my_print_defaults, 575
NDB client programs, 4068
defaults-file option, 326
myisamchk, 520
mysql, 391
mysqladmin, 421
mysqlbinlog, 554
mysqlcheck, 431
mysqld, 678
mysqldump, 446
mysqld_multi, 364
mysqld_safe, 357
mysqlimport, 465
mysqlpump, 477
mysqlshow, 491
mysqlslap, 500
mysql_secure_installation, 370
mysql_upgrade, 380
my_print_defaults, 575
NDB client programs, 4068
defaults-group-suffix option, 327
myisamchk, 520
mysql, 391
mysqladmin, 421

mysqlbinlog, 554
mysqlcheck, 431
mysqld, 679
mysqldump, 446
mysqlimport, 465
mysqlpump, 477
mysqlshow, 491
mysqlslap, 500
mysql_secure_installation, 370
mysql_upgrade, 381
my_print_defaults, 576
NDB client programs, 4068
default_authentication_plugin system variable, 726
DEFAULT_CHARSET option
CMake, 211
DEFAULT_COLLATION option
CMake, 212
default_collation_for_utf8mb4 system variable, 727
default_password_lifetime system variable, 728
default_roles table
system table, 946, 1096
default_storage_engine system variable, 728
default_table_encryption, 2797
default_table_encryption variable, 729
default_tmp_storage_engine system variable, 729
default_week_format system variable, 730
defer-table-indexes option
mysqlpump, 477
DEFINER privileges, 2653, 4354
DEGREES(), 1969
delay option (ndbinfo_select_all), 3956
DELAYED, 2399
INSERT modifier, 2396
Delayed insert
thread command, 1654
delayed replication, 3361
delayed_insert_limit system variable, 731
delayed_insert_timeout system variable, 731
delayed_queue_size system variable, 732
delay_key_write system variable, 730, 3127
DELETE, 2382
and NDB Cluster, 3677
delete, 6588
delete buffering, 6588
delete option
mysqlimport, 465
delete option (ndb_index_stat), 4007
DELETE privilege, 1083
delete-master-logs option
mysqldump, 449
delete-orphans option
ndb_blob_tool, 3969
deleting
accounts, 1115

database, 2371
foreign key, 2262, 2343
function, 2621
index, 2261, 2372
primary key, 2261
rows, 5644
schema, 2371
table, 2374
user, 2585
users, 2585
deleting from main table
thread state, 1657
deleting from reference tables
thread state, 1657
deletion
mysql.sock, 5638
delimiter command
mysql, 401
delimiter option
mysql, 391
mysqlslap, 501
ndb_select_all, 4046
demo_test table, 3079
denormalized, 6588
DENSE_RANK(), 2195
deprecated features in MySQL 8.0, 9
derived tables, 2444
lateral, 1578, 2448
materialization prevention, 1514
optimization, 1502, 1512
updatable views, 4351
des-key-file option
mysqld, 679
DESC, 2701
descending index, 6588
descending indexes, 1537
descending option
ndb_select_all, 4045
DESCRIBE, 300, 2701
description option
myisamchk, 525
design
issues, 5648
DES_DECRYPT(), 2059
DES_ENCRYPT(), 2059
detach option
mysqlslap, 501
development of NDB Cluster, 3661
development source tree, 197
diagnostics() procedure
sys schema, 4748
dictionary collation, German, 1786, 1786
dictionary object cache, 2711, 6588
DictTrace, 3832

dict_obj_info
  ndbinfo table, 4134
dict_obj_types
  ndbinfo table, 4134
diff-default option
  ndb_config, 3974
digits, 1818
directory structure
  default, 93
dirty page, 2935, 6589
dirty read, 6589
disable named command
  mysql, 391
--disable option prefix, 327
disable-indexes option
  ndb_restore, 4026
disable-keys option
  mysqldump, 457
disable-log-bin option
  mysqlbinlog, 555
disabled_storage_engines system variable, 732
DISABLE_PSI_COND option
  CMake, 212
DISABLE_PSI_DATA_LOCK option
  CMake, 213
DISABLE_PSI_ERROR option
  CMake, 213
DISABLE_PSI_FILE option
  CMake, 212
DISABLE_PSI_IDLE option
  CMake, 212
DISABLE_PSI_MEMORY option
  CMake, 212
DISABLE_PSI_METADATA option
  CMake, 212
DISABLE_PSI_MUTEX option
  CMake, 212
DISABLE_PSI_PS option
  CMake, 213
DISABLE_PSI_RWLOCK option
  CMake, 212
DISABLE_PSI_SOCKET option
  CMake, 212
DISABLE_PSI_SP option
  CMake, 212
DISABLE_PSI_STAGE option
  CMake, 212
DISABLE_PSI_STATEMENT option
  CMake, 212
DISABLE_PSI_STATEMENT_DIGEST option
  CMake, 212
DISABLE_PSI_TABLE option
  CMake, 212
DISABLE_PSI_THREAD option

CMake, 213
DISABLE_PSI_TRANSACTION option
CMake, 213
DISABLE_SHARED option
CMake, 213
DISCARD TABLESPACE, 2265, 2739
discard_or_import_tablespace
thread state, 1657
disconnect-slave-event-count option
mysqld, 3239
disconnecting
from the server, 277
disconnect_on_expired_password system variable, 733
Disk Data tables (NDB Cluster) (see NDB Cluster Disk Data)
disk failure
InnoDB, 3069
disk full, 5636
disk I/O, 1559
disk option
ndb_select_all, 4046
disk performance, 1631
disk-based, 6589
disk-bound, 6589
DiskIOThreadPool, 3851, 3856
Diskless, 3808
diskpagebuffer
ndbinfo table, 4137
DiskPageBufferEntries, 3850
DiskPageBufferMemory, 3850, 3856
disks
splitting data across, 1634
DiskSyncSize, 3822
disk_write_speed_aggregate
ndbinfo table, 4135
disk_write_speed_aggregate_node
ndbinfo table, 4136
disk_write_speed_base
ndbinfo table, 4135
display size, 1818
display triggers, 2682
display width, 1818
displaying
database information, 487
information
Cardinality, 2657
Collation, 2657
SHOW, 2631, 2636, 2682
SHOW statement, 2655, 2658
table status, 2679
DISTINCT, 289, 1495
AVG(), 2172
COUNT(), 2175
MAX(), 2179
MIN(), 2179

SELECT modifier, 2427
SUM(), 2180
DISTINCTROW
SELECT modifier, 2427
distinguished name
LDAP authentication, 1223
distributed privileges (NDB Cluster), 4199
DIV, 1966
division (/), 1965
div_precision_increment system variable, 734
DML, 2381, 6589
DELETE statement, 2382
INSERT statement, 2391
UPDATE statement, 2455
DMR
MySQL releases, 78
DN (see distinguished name)
DNS, 1644
DO, 2386
DocBook XML
documentation source format, 2
Docker, 270
Docker images
on Windows, 177
document id, 6589
document store, 3529
MySQL as a, 3529
Documentation
in Chinese, 5071
in Japanese, 5071
in Korean, 5071
Documenters
list of, 70
dont-ignore-systab-0 option
ndb_restore, 4026
DOUBLE data type, 1821
DOUBLE PRECISION data type, 1822
double quote (\"), 1671, 2142
doublewrite buffer, 902, 2900, 2958, 6589
downgrades
NDB Cluster, 3736, 4085
downgrading, 274
downloading, 79
DOWNLOAD_BOOST option
CMake, 213
DOWNLOAD_BOOST_TIMEOUT option
CMake, 213
dragnet.log_error_filter_rules system variable, 734
dragnet.Status status variable, 898
drop, 6590
DROP ... IF EXISTS
and replication, 3372
DROP DATABASE, 2371
Drop DB

thread command, 1654
DROP EVENT, 2372
DROP FOREIGN KEY, 2262, 2343
DROP FUNCTION, 2373
DROP FUNCTION statement, 2621
DROP INDEX, 2261, 2372
DROP LOGFILE GROUP, 2373
(see also NDB Cluster Disk Data)
DROP NODEGROUP command (NDB Cluster), 4077
DROP PREPARE, 2519
DROP PRIMARY KEY, 2261
DROP privilege, 1083
DROP PROCEDURE, 2373
DROP RESOURCE GROUP statement, 2607
DROP ROLE privilege, 1083
DROP ROLE statement, 2584
DROP SCHEMA, 2371
DROP SERVER, 2373
DROP SPATIAL REFERENCE SYSTEM, 2374
DROP TABLE, 2374
and NDB Cluster, 3677
DROP TABLESPACE
general tablespace, 2375
NDB Cluster Disk Data, 2375
undo tablespace, 2375
DROP TRIGGER, 2377
DROP USER statement, 1112, 2585
DROP VIEW, 2377
drop-source option
ndb_move_data, 4011
dropping
accounts, 1115
user, 2585
dropping roles, 2584
dry-scp option (ndb_error_reporter), 3990
DUAL, 2422
dual passwords, 1143
dump option
myisam_ftdump, 515
ndb_redo_log_reader, 4019
dump option (ndb_index_stat), 4007
dump-date option
mysqldump, 448
dump-file option
ibd2sdi, 507
ndb_blob_tool, 3970
dump-slave option
mysqldump, 449
DUMPFILE, 2429
dumping
databases and tables, 436, 470
Duplicate Weedout
semijoin strategy, 1505
DYLD_LIBRARY_PATH environment variable, 4793

dynamic privileges, 1093
dynamic row format, 2896, 6590
dynamic table characteristics, 3129

E

early adopter, 6590
early-plugin-load option
  mysqld, 679
edit command
  mysql, 401
EE_ADJUSTED_DOUBLE_VALUE_FOR_OPTION error code, 5616
EE_ADJUSTED_SIGNED_VALUE_FOR_OPTION error code, 5616
EE_ADJUSTED_ULONGLONG_VALUE_FOR_OPTION error code, 5616
EE_ADJUSTED_UNSIGNED_VALUE_FOR_OPTION error code, 5616
EE_BADCLOSE error code, 5609
EE_CANTCREATEFILE error code, 5609
EE_CANTLOCK error code, 5610
EE_CANTUNLOCK error code, 5610
EE_CANT_CHSIZE error code, 5610
EE_CANT_MKDIR error code, 5610
EE_CANT_OPEN_STREAM error code, 5610
EE_CANT_READLINK error code, 5611
EE_CANT_SEEK error code, 5611
EE_CANT_SYMLINK error code, 5611
EE_CAPACITY_EXCEEDED error code, 5611
EE_CHANGE_OWNERSHIP error code, 5611
EE_CHANGE_PERMISSIONS error code, 5611
EE_COLLATION_PARSER_ERROR error code, 5616
EE_CONFIG_FILE_PERMISSION_ERROR error code, 5613
EE_DEBUG_INFO error code, 5614
EE_DELETE error code, 5609
EE_DIR error code, 5610
EE_DISK_FULL error code, 5610
EE_DISK_FULL_WITH_RETRY_MSG error code, 5611
EE_EOFERR error code, 5610
EE_EXITING_TIMER_NOTIFY_THREAD error code, 5612
EE_FAILED_TO_ASSIGN_MAX_VALUE_TO_OPTION error code, 5615
EE_FAILED_TO_CREATE_IO_COMPLETION_PORT error code, 5612
EE_FAILED_TO_CREATE_TIMER error code, 5611
EE_FAILED_TO_CREATE_TIMER_NOTIFY_THREAD_INTERRUPT_EVENT error code, 5612
EE_FAILED_TO_CREATE_TIMER_QUEUE error code, 5612
EE_FAILED_TO_DELETE_TIMER error code, 5611
EE_FAILED_TO_DETERMINE_LARGE_PAGE_SIZE error code, 5612
EE_FAILED_TO_HANDLE_DEFAULTS_FILE error code, 5613
EE_FAILED_TO_KILL_ALL_THREADS error code, 5612
EE_FAILED_TO_LOCATE_SERVER_PUBLIC_KEY error code, 5614
EE_FAILED_TO_OPEN_DEFAULTS_FILE error code, 5612
EE_FAILED_TO_RESET_BEFORE_PRIMARY_IGNORABLE_CHAR error code, 5616
EE_FAILED_TO_RESET_BEFORE_SECONDARY_IGNORABLE_CHAR error code, 5617
EE_FAILED_TO_RESET_BEFORE_TERTIARY_IGNORABLE_CHAR error code, 5616
EE_FAILED_TO_SET_OPTION_VALUE error code, 5615
EE_FAILED_TO_START_TIMER_NOTIFY_THREAD error code, 5612
EE_FILENOTFOUND error code, 5611

EE_FILE_NOT_CLOSED error code, 5611
EE_GETWD error code, 5610
EE_IGNORE_WORLD_WRITABLE_CONFIG_FILE error code, 5613
EE_INCORRECT_BOOLEAN_VALUE_FOR_OPTION error code, 5615
EE_INCORRECT_GRP_DEFINITION_IN_CONFIG_FILE error code, 5613
EE_INCORRECT_INT_VALUE_FOR_OPTION error code, 5616
EE_INCORRECT_UINT_VALUE_FOR_OPTION error code, 5616
EE_INVALID_DECIMAL_VALUE_FOR_OPTION error code, 5616
EE_LINK error code, 5609
EE_LINK_WARNING error code, 5610
EE_NET_SEND_ERROR_IN_BOOTSTRAP error code, 5614
EE_OPEN_WARNING error code, 5610
EE_OPTION_IGNORED_DUE_TO_INVALID_VALUE error code, 5615
EE_OPTION_REQUIRES_ARGUMENT error code, 5615
EE_OPTION_WITHOUT_ARGUMENT error code, 5615
EE_OPTION_WITHOUT_GRP_IN_CONFIG_FILE error code, 5613
EE_OPTION_WITH_EMPTY_VALUE error code, 5615
EE_OUTOFMEMORY error code, 5609
EE_OUT_OF_FILERESOURCES error code, 5610
EE_PACKETS_OUT_OF_ORDER error code, 5614
EE_PUBLIC_KEY_NOT_IN_PEM_FORMAT error code, 5614
EE_READ error code, 5609
EE_REALPATH error code, 5611
EE_RESET_CHAR_OUT_OF_RANGE error code, 5617
EE_SETWD error code, 5610
EE_SHIFT_CHAR_OUT_OF_RANGE error code, 5617
EE_SHORT_OPTION_REQUIRES_ARGUMENT error code, 5615
EE_SKIPPING_DIRECTIVE_DUE_TO_MAX_INCLUDE_RECURSION error code, 5613
EE_SSL_ERROR error code, 5614
EE_SSL_ERROR_FROM_FILE error code, 5614
EE_STAT error code, 5610
EE_SYNC error code, 5611
EE_UNKNOWN_CHARSET error code, 5610
EE_UNKNOWN_COLLATION error code, 5611
EE_UNKNOWN_LDML_TAG error code, 5617
EE_UNKNOWN_OPTION error code, 5615
EE_UNKNOWN_PROTOCOL_OPTION error code, 5614
EE_UNKNOWN_SHORT_OPTION error code, 5615
EE_UNKNOWN_SUFFIX_FOR_VARIABLE error code, 5614
EE_UNKNOWN_VARIABLE error code, 5614
EE_USING_DISABLED_OPTION error code, 5613
EE_USING_DISABLED_SHORT_OPTION error code, 5613
EE_USING_PASSWORD_ON_CLI_IS_INSECURE error code, 5613
EE_WIN_LIBRARY_LOAD_FAILED error code, 5612
EE_WIN_RUN_TIME_ERROR_CHECK error code, 5612
EE_WRITE error code, 5609
EE_WRONG_DIRECTIVE_IN_CONFIG_FILE error code, 5613
ego command
  mysql, 401
Eiffel Wrapper, 4941
ELT(), 1936
--enable option prefix, 327
enable-cleartext-plugin option
  mysql, 392

mysqladmin, 422
mysqlcheck, 431
mysqldump, 443
mysqlimport, 465
mysqlshow, 491
mysqlslap, 501
ENABLED_LOCAL_INFILE option
CMake, 214
ENABLED_PROFILING option
CMake, 214
EnablePartialLcp, 3798
EnableRedoControl, 3802
ENABLE_DEBUG_SYNC option
CMake, 213
ENABLE_DOWNLOADS option
CMake, 213
ENABLE_DTRACE option
CMake, 213
ENABLE_EXPERIMENTAL_SYSVARS option
CMake, 213
ENABLE_GCOV option
CMake, 214
ENABLE_GPROF option
CMake, 214
ENCODE(), 2060
ENCRYPT(), 2060
encrypted connections, 1169
as mandatory, 1176
command options, 334
encryption, 1071, 1169, 2795
binary log files, 3333
encryption functions, 2055
ENCRYPTION_KEY_ADMIN privilege, 1089
end
thread state, 1657
END, 2520
end-page option
innochecksum, 511
end_markers_in_json system variable, 735
enforce_gtid_consistency system variable, 3294
engine condition pushdown, 1464
engine option
mysqlslap, 501
ENGINES
INFORMATION_SCHEMA table, 4379
engine_cost
system table, 1609
engine_cost table
system table, 947
ENTER SINGLE USER MODE command (NDB Cluster), 4075
entering
queries, 278
enterprise components
MySQL Enterprise Audit, 1312, 5054

MySQL Enterprise Backup, 5052
MySQL Enterprise Data Masking and De-Identification, 5054
MySQL Enterprise Encryption, 5053
MySQL Enterprise Firewall, 1386, 5054
MySQL Enterprise Monitor, 5051
MySQL Enterprise Security, 1205, 1216, 1221, 5053
MySQL Enterprise Thread Pool, 999, 5054
ENUM
size, 1897
ENUM data type, 1827, 1848
environment variable
AUTHENTICATION_LDAP_CLIENT_LOG, 1248
AUTHENTICATION_PAM_LOG, 578, 1215
CC, 226, 578
CXX, 226, 578
DBI_TRACE, 578, 5037
DBI_USER, 578
DYLD_LIBRARY_PATH, 4793
HOME, 406, 578
LD)_PRELOAD, 184
LD_LIBRARY_PATH, 276, 4793
LD_RUN_PATH, 276, 578
LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN, 578
LIBMYSQL_PLUGINS, 578, 4921
LIBMYSQL_PLUGIN_DIR, 578, 4921
MYSQLD_OPTS, 184
MYSQLX_TCP_PORT, 578
MYSQLX_UNIX_PORT, 578
MYSQL_DEBUG, 317, 578, 5039
MYSQL_GROUP_SUFFIX, 578
MYSQL_HISTFILE, 406, 578
MYSQL_HISTIGNORE, 406, 578
MYSQL_HOME, 578
MYSQL_HOST, 343, 578
MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD, 578, 2166
MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD, 578, 2166
MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD, 578, 2166
MYSQL_PS1, 578
MYSQL_PWD, 578
MYSQL_TCP_PORT, 317, 578, 1062, 1063
MYSQL_TEST_LOGIN_FILE, 327, 541, 578
MYSQL_TEST_TRACE_CRASH, 578, 5002
MYSQL_TEST_TRACE_DEBUG, 578, 5002
MYSQL_UNIX_PORT, 317, 578, 1062, 1063
NOTIFY_SOCKET, 184, 578
PATH, 132, 138, 238, 318, 578
PKG_CONFIG_PATH, 578, 4791
SUDO_USER, 4577
TMPDIR, 317, 578, 5636
TZ, 184, 578, 931, 5638
UMASK, 578, 5631
UMASK_DIR, 578, 5631
USER, 343, 578
environment variables, 317, 353, 1164

list of, 578
equal (=), 1920
eq_ref join type
optimizer, 1573
Errcode, 577
errins-delay option
ndb_import, 3996
errins-type option
ndb_import, 3996
errno, 577
Error
thread command, 1654
error code
CR_ALREADY_CONNECTED, 5608
CR_AUTH_PLUGIN_CANNOT_LOAD, 5608
CR_AUTH_PLUGIN_ERR, 5608
CR_CANT_READ_CHARSET, 5605
CR_COMMANDS_OUT_OF_SYNC, 5605
CR_COMPRESSION_NOT_SUPPORTED, 5609
CR_COMPRESSION_WRONGLY_CONFIGURED, 5609
CR_CONNECTION_ERROR, 5604
CR_CONN_HOST_ERROR, 5604
CR_CONN_UNKNOW_PROTOCOL, 5607
CR_DATA_TRUNCATED, 5606
CR_DUPLICATE_CONNECTION_ATTR, 5608
CR_EMBEDDED_CONNECTION, 5605
CR_FETCH_CANCELED, 5607
CR_FILE_NAME_TOO_LONG, 5608
CR_INSECURE_API_ERR, 5608
CR_INVALID_BUFFER_USE, 5606
CR_INVALID_CONN_HANDLE, 5607
CR_INVALID_PARAMETER_NO, 5606
CR_IPSOCK_ERROR, 5604
CR_LOCALHOST_CONNECTION, 5605
CR_MALFORMED_PACKET, 5606
CR_NAMEDPIPEOPEN_ERROR, 5605
CR_NAMEDPIPESETSTATE_ERROR, 5605
CR_NAMEDPIPEWAIT_ERROR, 5605
CR_NAMEDPIPE_CONNECTION, 5605
CR_NET_PACKET_TOO_LARGE, 5605
CR_NEW_STMT_METADATA, 5608
CR_NOT_IMPLEMENTED, 5608
CR_NO_DATA, 5607
CR_NO_PARAMETERS_EXISTS, 5606
CR_NO_PREPARE_STMT, 5606
CR_NO_RESULT_SET, 5608
CR_NO_STMT_METADATA, 5608
CR_NULL_POINTER, 5606
CR_OUT_OF_MEMORY, 5604
CR_PARAMS_NOT_BOUND, 5606
CR_PROBE_MASTER_CONNECT, 5606
CR_PROBE_SLAVE_CONNECT, 5606
CR_PROBE_SLAVE_HOSTS, 5605
CR_PROBE_SLAVE_STATUS, 5605

CR_SERVER_GONE_ERROR, 5604
CR_SERVER_HANDSHAKE_ERR, 5605
CR_SERVER_LOST, 5605
CR_SERVER_LOST_EXTENDED, 5608
CR_SHARED_MEMORY_CONNECTION, 5606
CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR, 5607
CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR, 5607
CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR, 5607
CR_SHARED_MEMORY_CONNECT_MAP_ERROR, 5607
CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR, 5607
CR_SHARED_MEMORY_CONNECT_SET_ERROR, 5607
CR_SHARED_MEMORY_EVENT_ERROR, 5607
CR_SHARED_MEMORY_FILE_MAP_ERROR, 5607
CR_SHARED_MEMORY_MAP_ERROR, 5607
CR_SOCKET_CREATE_ERROR, 5604
CR_SSL_CONNECTION_ERROR, 5606
CR_SSL_FIPS_MODE_ERR, 5608
CR_STMT_CLOSED, 5608
CR_TCP_CONNECTION, 5605
CR_UNKNOWN_ERROR, 5604
CR_UNKNOWN_HOST, 5604
CR_UNSUPPORTED_PARAM_TYPE, 5606
CR_UNUSED_1, 5607
CR_VERSION_ERROR, 5604
CR_WRONG_HOST_INFO, 5604
CR_WRONG_LICENSE, 5606
EE_ADJUSTED_DOUBLE_VALUE_FOR_OPTION, 5616
EE_ADJUSTED_SIGNED_VALUE_FOR_OPTION, 5616
EE_ADJUSTED_ULONGLONG_VALUE_FOR_OPTION, 5616
EE_ADJUSTED_UNSIGNED_VALUE_FOR_OPTION, 5616
EE_BADCLOSE, 5609
EE_CANTCREATEFILE, 5609
EE_CANTLOCK, 5610
EE_CANTUNLOCK, 5610
EE_CANT_CHSIZE, 5610
EE_CANT_MKDIR, 5610
EE_CANT_OPEN_STREAM, 5610
EE_CANT_READLINK, 5611
EE_CANT_SEEK, 5611
EE_CANT_SYMLINK, 5611
EE_CAPACITY_EXCEEDED, 5611
EE_CHANGE_OWNERSHIP, 5611
EE_CHANGE_PERMISSIONS, 5611
EE_COLLATION_PARSER_ERROR, 5616
EE_CONFIG_FILE_PERMISSION_ERROR, 5613
EE_DEBUG_INFO, 5614
EE_DELETE, 5609
EE_DIR, 5610
EE_DISK_FULL, 5610
EE_DISK_FULL_WITH_RETRY_MSG, 5611
EE_EOFERR, 5610
EE_EXITING_TIMER_NOTIFY_THREAD, 5612
EE_FAILED_TO_ASSIGN_MAX_VALUE_TO_OPTION, 5615
EE_FAILED_TO_CREATE_IO_COMPLETION_PORT, 5612

EE_FAILED_TO_CREATE_TIMER, 5611
EE_FAILED_TO_CREATE_TIMER_NOTIFY_THREAD_INTERRUPT_EVENT, 5612
EE_FAILED_TO_CREATE_TIMER_QUEUE, 5612
EE_FAILED_TO_DELETE_TIMER, 5611
EE_FAILED_TO_DETERMINE_LARGE_PAGE_SIZE, 5612
EE_FAILED_TO_HANDLE_DEFAULTS_FILE, 5613
EE_FAILED_TO_KILL_ALL_THREADS, 5612
EE_FAILED_TO_LOCATE_SERVER_PUBLIC_KEY, 5614
EE_FAILED_TO_OPEN_DEFAULTS_FILE, 5612
EE_FAILED_TO_RESET_BEFORE_PRIMARY_IGNORABLE_CHAR, 5616
EE_FAILED_TO_RESET_BEFORE_SECONDARY_IGNORABLE_CHAR, 5617
EE_FAILED_TO_RESET_BEFORE_TERTIARY_IGNORABLE_CHAR, 5616
EE_FAILED_TO_SET_OPTION_VALUE, 5615
EE_FAILED_TO_START_TIMER_NOTIFY_THREAD, 5612
EE_FILENOTFOUND, 5611
EE_FILE_NOT_CLOSED, 5611
EE_GETWD, 5610
EE_IGNORE_WORLD_WRITABLE_CONFIG_FILE, 5613
EE_INCORRECT_BOOLEAN_VALUE_FOR_OPTION, 5615
EE_INCORRECT_GRP_DEFINITION_IN_CONFIG_FILE, 5613
EE_INCORRECT_INT_VALUE_FOR_OPTION, 5616
EE_INCORRECT_UINT_VALUE_FOR_OPTION, 5616
EE_INVALID_DECIMAL_VALUE_FOR_OPTION, 5616
EE_LINK, 5609
EE_LINK_WARNING, 5610
EE_NET_SEND_ERROR_IN_BOOTSTRAP, 5614
EE_OPEN_WARNING, 5610
EE_OPTION_IGNORED_DUE_TO_INVALID_VALUE, 5615
EE_OPTION_REQUIRES_ARGUMENT, 5615
EE_OPTION_WITHOUT_ARGUMENT, 5615
EE_OPTION_WITHOUT_GRP_IN_CONFIG_FILE, 5613
EE_OPTION_WITH_EMPTY_VALUE, 5615
EE_OUTOFMEMORY, 5609
EE_OUT_OF_FILERESOURCES, 5610
EE_PACKETS_OUT_OF_ORDER, 5614
EE_PUBLIC_KEY_NOT_IN_PEM_FORMAT, 5614
EE_READ, 5609
EE_REALPATH, 5611
EE_RESET_CHAR_OUT_OF_RANGE, 5617
EE_SETWD, 5610
EE_SHIFT_CHAR_OUT_OF_RANGE, 5617
EE_SHORT_OPTION_REQUIRES_ARGUMENT, 5615
EE_SKIPPING_DIRECTIVE_DUE_TO_MAX_INCLUDE_RECURSION, 5613
EE_SSL_ERROR, 5614
EE_SSL_ERROR_FROM_FILE, 5614
EE_STAT, 5610
EE_SYNC, 5611
EE_UNKNOWN_CHARSET, 5610
EE_UNKNOWN_COLLATION, 5611
EE_UNKNOWN_LDML_TAG, 5617
EE_UNKNOWN_OPTION, 5615
EE_UNKNOWN_PROTOCOL_OPTION, 5614
EE_UNKNOWN_SHORT_OPTION, 5615
EE_UNKNOWN_SUFFIX_FOR_VARIABLE, 5614

EE_UNKNOWN_VARIABLE, 5614
EE_USING_DISABLED_OPTION, 5613
EE_USING_DISABLED_SHORT_OPTION, 5613
EE_USING_PASSWORD_ON_CLI_IS_INSECURE, 5613
EE_WIN_LIBRARY_LOAD_FAILED, 5612
EE_WIN_RUN_TIME_ERROR_CHECK, 5612
EE_WRITE, 5609
EE_WRONG_DIRECTIVE_IN_CONFIG_FILE, 5613
ER_ABORTING, 5232
ER_ABORTING_CONNECTION, 5109
ER_ABORTING_USER_CONNECTION, 5323
ER_ACCESS_DENIED_CHANGE_USER_ERROR, 5156
ER_ACCESS_DENIED_ERROR, 5101
ER_ACCESS_DENIED_ERROR_WITHOUT_PASSWORD, 5324
ER_ACCESS_DENIED_ERROR_WITH_PASSWORD, 5324
ER_ACCESS_DENIED_FOR_USER_ACCOUNT_LOCKED, 5325
ER_ACCESS_DENIED_NO_PASSWORD_ERROR, 5143
ER_ACCOUNT_HAS_BEEN_LOCKED, 5166
ER_ACL_OPERATION_FAILED, 5173
ER_ACL_WRONG_OR_MISSING_ACL_TABLES_LOG, 5596
ER_ADDRESSES_FOR_HOSTNAME_HEADER, 5226
ER_ADDRESSES_FOR_HOSTNAME_LIST_ITEM, 5226
ER_ADD_PARTITION_NO_NEW_PARTITION, 5132
ER_ADD_PARTITION_SUBPART_ERROR, 5132
ER_ADMIN_WRONG_MRG_TABLE, 5129
ER_AES_INVALID_IV, 5157
ER_AGGREGATE_IN_ORDER_NOT_SELECT, 5162
ER_AGGREGATE_ORDER_FOR_UNION, 5159
ER_AGGREGATE_ORDER_NON_AGG_QUERY, 5160
ER_ALTER_FILEGROUP_FAILED, 5133
ER_ALTER_INFO, 5104
ER_ALTER_OPERATION_NOT_SUPPORTED, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_AUTOINC, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_CHANGE_FTS, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COLUMN_TYPE, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COPY, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_CHECK, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_RENAME, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FTS, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_GIS, 5162
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_HIDDEN_FTS, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOPK, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOT_NULL, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_PARTITION, 5154
ER_AMBIGUOUS_FIELD_TERM, 5129
ER_ANONYMOUS_AUTH_ID_NOT_ALLOWED_IN_MANDATORY_ROLES, 5329
ER_APPLIER_LOG_EVENT_VALIDATION_ERROR, 5186
ER_ATTRIBUTE_IGNORED, 5188
ER_AUDIT_API_ABORT, 5170
ER_AUDIT_CANT_ABORT_COMMAND, 5321
ER_AUDIT_CANT_ABORT_EVENT, 5321
ER_AUDIT_LOG_CANNOT_GENERATE_PASSWORD, 5603
ER_AUDIT_LOG_CANNOT_READ_PASSWORD, 5596

ER_AUDIT_LOG_CANNOT_REMOVE_PASSWORD, 5596
ER_AUDIT_LOG_CANNOT_SET_LOG_POLICY_WITH_OTHER_POLICIES, 5356
ER_AUDIT_LOG_CANNOT_STORE_PASSWORD, 5596
ER_AUDIT_LOG_COULD_NOT_CREATE_AES_KEY, 5566
ER_AUDIT_LOG_COULD_NOT_OPEN_FILE_FOR_READING, 5596
ER_AUDIT_LOG_EC_WRITER_FAILED_TO_CREATE_FILE, 5357
ER_AUDIT_LOG_EC_WRITER_FAILED_TO_INIT_COMPRESSION, 5357
ER_AUDIT_LOG_EC_WRITER_FAILED_TO_INIT_ENCRYPTION, 5357
ER_AUDIT_LOG_ENCRYPTION_PASSWORD_CANNOT_BE_FETCHED, 5566
ER_AUDIT_LOG_ENCRYPTION_PASSWORD_HAS_NOT_BEEN_SET, 5566
ER_AUDIT_LOG_FILE_HAS_BEEN_SUCCESSFULLY_PROCESSED, 5596
ER_AUDIT_LOG_FILE_NAME_DOES_NOT_HAVE_REQUIRED_FORMAT, 5595
ER_AUDIT_LOG_FILE_NAME_KEYRING_ID_VALUE_IS_MISSING, 5595
ER_AUDIT_LOG_FILE_NAME_TIMESTAMP_VALUE_IS_MISSING_OR_INVALID, 5595
ER_AUDIT_LOG_FILTER_FAILED_TO_CLOSE_TABLE_AFTER_READING, 5359
ER_AUDIT_LOG_FILTER_FAILED_TO_DELETE_FROM_TABLE, 5358
ER_AUDIT_LOG_FILTER_FAILED_TO_INIT_TABLE_FOR_READ, 5358
ER_AUDIT_LOG_FILTER_FAILED_TO_INSERT_INTO_TABLE, 5358
ER_AUDIT_LOG_FILTER_FAILED_TO_READ_TABLE, 5359
ER_AUDIT_LOG_FILTER_FAILED_TO_STORE_TABLE_FLDS, 5358
ER_AUDIT_LOG_FILTER_FAILED_TO_UPDATE_TABLE, 5358
ER_AUDIT_LOG_FILTER_FLD_FILTERNAME_CANNOT_BE_EMPTY, 5359
ER_AUDIT_LOG_FILTER_INVALID_COLUMN_COUNT, 5358
ER_AUDIT_LOG_FILTER_INVALID_COLUMN_DEFINITION, 5358
ER_AUDIT_LOG_FILTER_IS_NOT_INSTALLED, 5355
ER_AUDIT_LOG_FILTER_RESULT_MSG, 5357
ER_AUDIT_LOG_FILTER_USER_AND_HOST_CANNOT_BE_EMPTY, 5359
ER_AUDIT_LOG_HAS_NOT_BEEN_INSTALLED, 5600
ER_AUDIT_LOG_HOST_NAME_INVALID_CHARACTER, 5565
ER_AUDIT_LOG_INDEX_MAP_CANNOT_ACCESS_DIR, 5356
ER_AUDIT_LOG_INSUFFICIENT_PRIVILEGE, 5217
ER_AUDIT_LOG_INVALID_FILE_CONTENT, 5596
ER_AUDIT_LOG_JSON_FILTERING_NOT_ENABLED, 5565
ER_AUDIT_LOG_JSON_FILTER_DOES_NOT_EXIST, 5566
ER_AUDIT_LOG_JSON_FILTER_NAME_CANNOT_BE_EMPTY, 5565
ER_AUDIT_LOG_JSON_FILTER_PARSING_ERROR, 5565
ER_AUDIT_LOG_JSON_READER_BUF_TOO_SMALL, 5357
ER_AUDIT_LOG_JSON_READER_FAILED_TO_OPEN_FILE, 5358
ER_AUDIT_LOG_JSON_READER_FAILED_TO_PARSE, 5357
ER_AUDIT_LOG_JSON_READER_FILE_PARSING_ERROR, 5358
ER_AUDIT_LOG_JSON_USER_NAME_CANNOT_BE_EMPTY, 5566
ER_AUDIT_LOG_KEYRING_ID_TIMESTAMP_VALUE_IS_INVALID, 5595
ER_AUDIT_LOG_NO_KEYRING_PLUGIN_INSTALLED, 5565
ER_AUDIT_LOG_ONLY_INCLUDE_LIST_USED, 5356
ER_AUDIT_LOG_PASSWORD_HAS_BEEN_COPIED, 5596
ER_AUDIT_LOG_RENAME_LOG_FILE_BEFORE_FLUSH, 5357
ER_AUDIT_LOG_SUPER_PRIVILEGE_REQUIRED, 5600
ER_AUDIT_LOG_SWITCHING_TO_INCLUDE_LIST, 5356
ER_AUDIT_LOG_TABLE_DEFINITION_NOT_UPDATED, 5567
ER_AUDIT_LOG_UDF_INSUFFICIENT_PRIVILEGE, 5565
ER_AUDIT_LOG_UDF_INVALID_ARGUMENT_COUNT, 5600
ER_AUDIT_LOG_UDF_INVALID_ARGUMENT_TYPE, 5600
ER_AUDIT_LOG_UDF_READ_INVALID_MAX_ARRAY_LENGTH_ARG_TYPE, 5600
ER_AUDIT_LOG_UDF_READ_INVALID_MAX_ARRAY_LENGTH_ARG_VALUE, 5173

ER_AUDIT_LOG_USER_FIRST_CHARACTER_MUST_BE_ALPHANUMERIC, 5566
ER_AUDIT_LOG_USER_NAME_INVALID_CHARACTER, 5565
ER_AUDIT_LOG_WRITER_DEST_FILE_ALREADY_EXISTS, 5356
ER_AUDIT_LOG_WRITER_FAILED_TO_WRITE_TO_FILE, 5357
ER_AUDIT_LOG_WRITER_INCOMPLETE_FILE_RENAMED, 5356
ER_AUDIT_LOG_WRITER_RENAME_FILE_FAILED, 5356
ER_AUDIT_LOG_WRITER_RENAME_FILE_FAILED_REMOVE_FILE_MANUALLY, 5356
ER_AUDIT_PLUGIN_DOES_NOT_SUPPORT_AUDIT_AUTH_EVENTS, 5257
ER_AUDIT_PLUGIN_HAS_INVALID_DATA, 5258
ER_AUDIT_WARNING, 5321
ER_AUTHCACHE_CANT_INIT_GRANT_SUBSYSTEM, 5254
ER_AUTHCACHE_CANT_OPEN_AND_LOCK_PRIVILEGE_TABLES, 5254
ER_AUTHCACHE_DB_ENTRY_LOWERCASED_REVOKE_WILL_FAIL, 5254
ER_AUTHCACHE_DB_IGNORED_EMPTY_NAME, 5254
ER_AUTHCACHE_DB_SKIPPED_NEEDS_RESOLVE, 5254
ER_AUTHCACHE_EXPIRED_PASSWORD_UNSUPPORTED, 5253
ER_AUTHCACHE_PLUGIN_CONFIG, 5253
ER_AUTHCACHE_PLUGIN_MISSING, 5252
ER_AUTHCACHE_PROCS_PRIV_ENTRY_IGNORED_BAD_ROUTINE_TYPE, 5255
ER_AUTHCACHE_PROCS_PRIV_SKIPPED_NEEDS_RESOLVE, 5255
ER_AUTHCACHE_PROXIES_PRIV_SKIPPED_NEEDS_RESOLVE, 5252
ER_AUTHCACHE_ROLE_TABLES_DODGY, 5253
ER_AUTHCACHE_TABLES_PRIV_SKIPPED_NEEDS_RESOLVE, 5255
ER_AUTHCACHE_TABLE_PROXIES_PRIV_MISSING, 5254
ER_AUTHCACHE_USER_IGNORED_DEPRECATED_PASSWORD, 5253
ER_AUTHCACHE_USER_IGNORED_INVALID_PASSWORD, 5253
ER_AUTHCACHE_USER_IGNORED_NEEDS_PLUGIN, 5253
ER_AUTHCACHE_USER_SKIPPED_NEEDS_RESOLVE, 5253
ER_AUTHCACHE_USER_TABLE_DODGY, 5253
ER_AUTH_CANT_ACTIVATE_ROLE, 5250
ER_AUTH_CANT_CREATE_RSA_PAIR, 5251
ER_AUTH_CANT_SET_DEFAULT_PLUGIN, 5238
ER_AUTH_CANT_WRITE_PRIVKEY, 5251
ER_AUTH_CANT_WRITE_PUBKEY, 5251
ER_AUTH_CERTS_SAVED_TO_DATADIR, 5252
ER_AUTH_CERT_GENERATION_DISABLED, 5252
ER_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES, 5588
ER_AUTH_KEYS_SAVED_TO_DATADIR, 5252
ER_AUTH_KEY_GENERATION_DISABLED, 5252
ER_AUTH_KEY_GENERATION_SKIPPED_PAIR_PRESENT, 5252
ER_AUTH_LDAP_ERROR_LOGGER_ERROR_MSG, 5362
ER_AUTH_RSA_CANT_FIND, 5250
ER_AUTH_RSA_CANT_PARSE, 5250
ER_AUTH_RSA_CANT_READ, 5250
ER_AUTH_RSA_CONF_PREVENTS_KEY_GENERATION, 5252
ER_AUTH_RSA_FILES_NOT_FOUND, 5250
ER_AUTH_SSL_CONF_PREVENTS_CERT_GENERATION, 5252
ER_AUTH_USING_EXISTING_CERTS, 5252
ER_AUTOINC_READ_FAILED, 5129
ER_AUTO_CONVERT, 5115
ER_AUTO_INCREMENT_CONFLICT, 5156
ER_AUTO_OPTIONS_FAILED, 5239
ER_AUTO_POSITION_REQUIRES_GTID_MODE_NOT_OFF, 5149
ER_BACK_IN_TIME, 5232

ER_BAD_DB_ERROR, 5101
ER_BAD_FIELD_ERROR, 5102
ER_BAD_FT_COLUMN, 5117
ER_BAD_HOST_ERROR, 5101
ER_BAD_LOG_STATEMENT, 5135
ER_BAD_NULL_ERROR, 5101
ER_BAD_NULL_ERROR_NOT_IGNORED, 5190
ER_BAD_SLAVE, 5112
ER_BAD_SLAVE_AUTO_POSITION, 5149
ER_BAD_SLAVE_UNTIL_COND, 5117
ER_BAD_TABLE_ERROR, 5101
ER_BASE64_DECODE_ERROR, 5135
ER_BASEDIR_SET_TO, 5327
ER_BEFORE_DML_VALIDATION_ERROR, 5165
ER_BEG_INITFILE, 5234
ER_BINLOG_ATTACHING_THREAD_MEMORY_FINALLY_AVAILABLE, 5311
ER_BINLOG_CACHE_SIZE_GREATER_THAN_MAX, 5146
ER_BINLOG_CACHE_SIZE_TOO_LARGE, 5322
ER_BINLOG_CANT_APPEND_LOG_TO_TMP_INDEX, 5313
ER_BINLOG_CANT_CLEAR_IN_USE_FLAG_FOR_CRASHED_BINLOG, 5317
ER_BINLOG_CANT_CLOSE_TMP_INDEX, 5314
ER_BINLOG_CANT_COPY_INDEX_TO_TMP, 5314
ER_BINLOG_CANT_CREATE_CACHE_FOR_LOG, 5311
ER_BINLOG_CANT_DELETE_FILE, 5314
ER_BINLOG_CANT_DELETE_FILE_AND_READ_BINLOG_INDEX, 5316
ER_BINLOG_CANT_DELETE_LOG_FILE_DOES_INDEX_MATCH_FILES, 5316
ER_BINLOG_CANT_FIND_LOG_IN_INDEX, 5316
ER_BINLOG_CANT_GENERATE_NEW_FILE_NAME, 5313
ER_BINLOG_CANT_LOCATE_OLD_BINLOG_OR_RELAY_LOG_FILES, 5314
ER_BINLOG_CANT_MOVE_TMP_TO_INDEX, 5314
ER_BINLOG_CANT_OPEN_CRASHED_BINLOG, 5317
ER_BINLOG_CANT_OPEN_FOR_LOGGING, 5312
ER_BINLOG_CANT_OPEN_LOG, 5311
ER_BINLOG_CANT_OPEN_TMP_INDEX, 5314
ER_BINLOG_CANT_RESIZE_CACHE, 5311
ER_BINLOG_CANT_SET_TMP_INDEX_NAME, 5314
ER_BINLOG_CANT_TRIM_CRASHED_BINLOG, 5317
ER_BINLOG_CANT_USE_FOR_LOGGING, 5313
ER_BINLOG_CRASHED_BINLOG_TRIMMED, 5317
ER_BINLOG_CRASH_RECOVERY_FAILED, 5317
ER_BINLOG_CREATE_ROUTINE_NEED_SUPER, 5126
ER_BINLOG_END, 5232
ER_BINLOG_ERROR_GETTING_NEXT_LOG_FROM_INDEX, 5314
ER_BINLOG_ERROR_READING_GTIDS_FROM_BINARY_LOG, 5312
ER_BINLOG_ERROR_READING_GTIDS_FROM_RELAY_LOG, 5312
ER_BINLOG_EVENTS_READ_FROM_BINLOG_INFO, 5313
ER_BINLOG_EVENTS_READ_FROM_RELAY_LOG_INFO, 5312
ER_BINLOG_EVENT_WRITE_TO_STMT_CACHE_FAILED, 5328
ER_BINLOG_EXPIRAY_LOG_DAYS_AND_SECS_USED_TOGETHER, 5342
ER_BINLOG_EXPIRE_LOG_DAYS_AND_SECS_USED_TOGETHER, 5191
ER_BINLOG_FAILED_TO_CLOSE_INDEX_FILE_WHILE_REBUILDING, 5313
ER_BINLOG_FAILED_TO_DELETE_INDEX_FILE_WHILE_REBUILDING, 5313
ER_BINLOG_FAILED_TO_DELETE_LOG_FILE, 5316
ER_BINLOG_FAILED_TO_OPEN_INDEX_FILE_AFTER_REBUILDING, 5313

ER_BINLOG_FAILED_TO_OPEN_REGISTER_FILE, 5315
ER_BINLOG_FAILED_TO_OPEN_TEMPORARY_INDEX_FILE, 5314
ER_BINLOG_FAILED_TO_READ_REGISTER_FILE, 5316
ER_BINLOG_FAILED_TO_REINIT_REGISTER_FILE, 5316
ER_BINLOG_FAILED_TO_RENAME_INDEX_FILE_WHILE_REBUILDING, 5313
ER_BINLOG_FAILED_TO_RUN_AFTER_FLUSH_HOOK, 5317
ER_BINLOG_FAILED_TO_RUN_AFTER_SYNC_HOOK, 5317
ER_BINLOG_FAILED_TO_SET_PURGE_INDEX_FILE_NAME, 5315
ER_BINLOG_FAILED_TO_SYNC_INDEX_FILE, 5312
ER_BINLOG_FAILED_TO_SYNC_INDEX_FILE_IN_OPEN, 5313
ER_BINLOG_FAILED_TO_WRITE_DROP_FOR_TEMP_TABLES, 5301
ER_BINLOG_FATAL_ERROR, 5136
ER_BINLOG_FILE_BEING_READ_NOT_PURGED, 5311
ER_BINLOG_FILE_EXTENSION_NUMBER_EXHAUSTED, 5312
ER_BINLOG_FILE_EXTENSION_NUMBER_RUNNING_LOW, 5312
ER_BINLOG_FILE_NAME_TOO_LONG, 5312
ER_BINLOG_FILE_OPEN_FAILED, 5328
ER_BINLOG_IO_ERROR_READING_HEADER, 5311
ER_BINLOG_LOGGING_IMPOSSIBLE, 5136
ER_BINLOG_LOGGING_INCIDENT_TO_STOP_SLAVES, 5316
ER_BINLOG_LOGGING_NOT_POSSIBLE, 5341
ER_BINLOG_LOGICAL_CORRUPTION, 5156
ER_BINLOG_MALFORMED_OR_OLD_RELAY_LOG, 5342
ER_BINLOG_MASTER_KEY_RECOVERY_OUT_OF_COMBINATION, 5205
ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_AUX_KEY, 5206
ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_UNUSED_KEYS, 5206
ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_OPERATE_KEY, 5205
ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_REENCRYPT_LOG, 5206
ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_ROTATE_LOGS, 5205
ER_BINLOG_MULTIPLE_ENGINES_AND_SELF_LOGGING_ENGINE, 5141
ER_BINLOG_NEEDS_SERVERID, 5237
ER_BINLOG_OOM_WRITING_DELETE_WHILE_OPENING_HEAP_TABLE, 5301
ER_BINLOG_PURGE_EMFILE, 5136
ER_BINLOG_PURGE_FATAL_ERR, 5123
ER_BINLOG_PURGE_LOGS_CALLED_WITH_FILE_NOT_IN_INDEX, 5315
ER_BINLOG_PURGE_LOGS_CANT_COPY_TO_REGISTER_FILE, 5315
ER_BINLOG_PURGE_LOGS_CANT_FLUSH_REGISTER_FILE, 5315
ER_BINLOG_PURGE_LOGS_CANT_SYNC_INDEX_FILE, 5315
ER_BINLOG_PURGE_LOGS_CANT_UPDATE_INDEX_FILE, 5315
ER_BINLOG_PURGE_LOGS_FAILED_TO_PURGE_LOG, 5315
ER_BINLOG_PURGE_PROHIBITED, 5123
ER_BINLOG_RECOVERING_AFTER_CRASH_USING, 5316
ER_BINLOG_ROW_ENGINE_AND_STMT_ENGINE, 5140
ER_BINLOG_ROW_INJECTION_AND_STMT_ENGINE, 5141
ER_BINLOG_ROW_INJECTION_AND_STMT_MODE, 5141
ER_BINLOG_ROW_LOGGING_FAILED, 5133
ER_BINLOG_ROW_MODE_AND_STMT_ENGINE, 5140
ER_BINLOG_ROW_VALUE_OPTION_IGNORED, 5559
ER_BINLOG_ROW_VALUE_OPTION_USED_ONLY_FOR_AFTER_IMAGES, 5560
ER_BINLOG_STMT_CACHE_SIZE_GREATER_THAN_MAX, 5147
ER_BINLOG_STMT_CACHE_SIZE_TOO_LARGE, 5322
ER_BINLOG_STMT_MODE_AND_NO_REPL_TABLES, 5154
ER_BINLOG_STMT_MODE_AND_ROW_ENGINE, 5141
ER_BINLOG_UNABLE_TO_ROTATE_GTID_TABLE_READONLY, 5592

ER_BINLOG_UNSAFE_AND_STMT_ENGINE, 5141
ER_BINLOG_UNSAFE_AUTOINC_COLUMNS, 5141
ER_BINLOG_UNSAFE_AUTOINC_NOT_FIRST, 5145
ER_BINLOG_UNSAFE_CREATE_IGNORE_SELECT, 5145
ER_BINLOG_UNSAFE_CREATE_REPLACE_SELECT, 5145
ER_BINLOG_UNSAFE_CREATE_SELECT_AUTOINC, 5145
ER_BINLOG_UNSAFE_DEFAULT_EXPRESSION_IN_SUBSTATEMENT, 5595
ER_BINLOG_UNSAFE_FULLTEXT_PLUGIN, 5158
ER_BINLOG_UNSAFE_INSERT_IGNORE_SELECT, 5144
ER_BINLOG_UNSAFE_INSERT_SELECT_UPDATE, 5144
ER_BINLOG_UNSAFE_INSERT_TWO_KEYS, 5145
ER_BINLOG_UNSAFE_LIMIT, 5141
ER_BINLOG_UNSAFE_MESSAGE_AND_STATEMENT, 5323
ER_BINLOG_UNSAFE_MIXED_STATEMENT, 5143
ER_BINLOG_UNSAFE_MULTIPLE_ENGINES_AND_SELF_LOGGING_ENGINE, 5143
ER_BINLOG_UNSAFE_NONTRANS_AFTER_TRANS, 5141
ER_BINLOG_UNSAFE_NOWAIT, 5179
ER_BINLOG_UNSAFE_REPLACE_SELECT, 5145
ER_BINLOG_UNSAFE_ROUTINE, 5126
ER_BINLOG_UNSAFE_SKIP_LOCKED, 5179
ER_BINLOG_UNSAFE_STATEMENT, 5136
ER_BINLOG_UNSAFE_SYSTEM_FUNCTION, 5141
ER_BINLOG_UNSAFE_SYSTEM_TABLE, 5141
ER_BINLOG_UNSAFE_SYSTEM_VARIABLE, 5141
ER_BINLOG_UNSAFE_UDF, 5141
ER_BINLOG_UNSAFE_UPDATE_IGNORE, 5145
ER_BINLOG_UNSAFE_WRITE_AUTOINC_SELECT, 5145
ER_BINLOG_UNSAFE_XA, 5172
ER_BINLOG_USE_V1_ROW_EVENTS_IGNORED, 5560
ER_BINLOG_WARNING_SUPPRESSED, 5318
ER_BLOBS_AND_NO_TERMINATED, 5104
ER_BLOB_CANT_HAVE_DEFAULT, 5105
ER_BLOB_FIELD_IN_PART_FUNC_ERROR, 5131
ER_BLOB_KEY_WITHOUT_LENGTH, 5110
ER_BLOB_USED_AS_KEY, 5103
ER_BOOST_GEOMETRY_CENTROID_EXCEPTION, 5160
ER_BOOST_GEOMETRY_EMPTY_INPUT_EXCEPTION, 5160
ER_BOOST_GEOMETRY_INCONSISTENT_TURNS_EXCEPTION, 5167
ER_BOOST_GEOMETRY_OVERLAY_INVALID_INPUT_EXCEPTION, 5160
ER_BOOST_GEOMETRY_SELF_INTERSECTION_POINT_EXCEPTION, 5161
ER_BOOST_GEOMETRY_TURN_INFO_EXCEPTION, 5161
ER_BOOST_GEOMETRY_UNKNOWN_EXCEPTION, 5161
ER_BOOTSTRAP_CANT_THREAD, 5220
ER_BUFPOOL_RESIZE_INPROGRESS, 5170
ER_CACHING_SHA2_PASSWORD_SECOND_PASSWORD_USED_INFORMATION, 5581
ER_CALL_ME_LOCALHOST, 5232
ER_CANNOT_ADD_FOREIGN, 5113
ER_CANNOT_ADD_FOREIGN_BASE_COL_STORED, 5172
ER_CANNOT_ADD_FOREIGN_BASE_COL_VIRTUAL, 5171
ER_CANNOT_ALTER_SRID_DUE_TO_INDEX, 5187
ER_CANNOT_CHANGE_TO_ROOT_DIR, 5327
ER_CANNOT_CONVERT_STRING, 5211
ER_CANNOT_CREATE_VIRTUAL_INDEX_CONSTRAINT, 5171
ER_CANNOT_DISCARD_TEMPORARY_TABLE, 5158

ER_CANNOT_DROP_COLUMN_FUNCTIONAL_INDEX, 5199
ER_CANNOT_FIND_KEY_IN_KEYRING, 5172
ER_CANNOT_GET_SERVER_VERSION_FROM_TABLESPACE_HEADER, 5584
ER_CANNOT_GRANT_ROLES_TO_ANONYMOUS_USER, 5213
ER_CANNOT_GRANT_SYSTEM_PRIV_TO_MANDATORY_ROLE, 5215
ER_CANNOT_LOAD_FROM_TABLE_V2, 5146
ER_CANNOT_LOCK_USER_MANAGEMENT_CACHES, 5176
ER_CANNOT_LOG_PARTIAL_DROP_DATABASE_WITH_GTID, 5164
ER_CANNOT_SET_DATABASE_ENCRYPTION, 5208
ER_CANNOT_SET_LOG_ERROR_SERVICES, 5327
ER_CANNOT_SET_SERVER_VERSION_IN_TABLESPACE_HEADER, 5584
ER_CANNOT_SET_TABLESPACE_ENCRYPTION, 5208
ER_CANNOT_SET_TABLE_ENCRYPTION, 5208
ER_CANNOT_USER, 5124
ER_CANNOT_USE_ENCRYPTION_CLAUSE, 5208
ER_CANT_ACCESS_CAPATH, 5239
ER_CANT_AGGREGATE_2COLLATIONS, 5116
ER_CANT_AGGREGATE_3COLLATIONS, 5116
ER_CANT_AGGREGATE_NCOLLATIONS, 5116
ER_CANT_ALLOC_TABLE_OBJECT, 5305
ER_CANT_CHANGE_TX_CHARACTERISTICS, 5135
ER_CANT_CHECK_PID_PATH, 5428
ER_CANT_CHOWN_DATADIR, 5238
ER_CANT_CREATE_ADMIN_THREAD, 5580
ER_CANT_CREATE_CACHE_FOR_DB_OPT, 5306
ER_CANT_CREATE_DB, 5099
ER_CANT_CREATE_FILE, 5098
ER_CANT_CREATE_GEOMETRY_OBJECT, 5126
ER_CANT_CREATE_HANDLER_FILE, 5131
ER_CANT_CREATE_HANDLER_OBJECT_FOR_TABLE, 5305
ER_CANT_CREATE_HANDLE_MGR_THREAD, 5223
ER_CANT_CREATE_INTERRUPT_THREAD, 5231
ER_CANT_CREATE_NAMED_PIPES_THREAD, 5231
ER_CANT_CREATE_PID_FILE, 5229
ER_CANT_CREATE_SCHEDULER_THREAD, 5225
ER_CANT_CREATE_SHM_THREAD, 5231
ER_CANT_CREATE_SHUTDOWN_THREAD, 5229
ER_CANT_CREATE_TABLE, 5099
ER_CANT_CREATE_TABLE_SHARE_FROM_FRM, 5305
ER_CANT_CREATE_TCPIP_THREAD, 5231
ER_CANT_CREATE_TEST_FILE, 5229
ER_CANT_CREATE_THREAD, 5108
ER_CANT_CREATE_USER_WITH_GRANT, 5125
ER_CANT_CREATE_UUID, 5228
ER_CANT_DETACH_SESSION_LEFT_BY_PLUGIN, 5271
ER_CANT_DO_IMPLICIT_COMMIT_IN_TRX_WHEN_GTID_NEXT_IS_SET, 5149
ER_CANT_DO_THIS_DURING_AN_TRANSACTION, 5111
ER_CANT_DROP_FIELD_OR_KEY, 5104
ER_CANT_ENFORCE_GTID_CONSISTENCY_WITH_ONGOING_GTID_VIOLATING_TX, 5166
ER_CANT_EXECUTE_IN_READ_ONLY_TRANSACTION, 5150
ER_CANT_FIND_DL_ENTRY, 5107
ER_CANT_FIND_SYSTEM_REC, 5099
ER_CANT_FIND_UDF, 5107
ER_CANT_GET_STAT, 5099

ER_CANT_HASH_DO_AND_IGNORE_RULES, 5239
ER_CANT_IDENTIFY_CHARSET_USING_DEFAULT, 5306
ER_CANT_INCREASE_MAX_OPEN_FILES, 5234
ER_CANT_INITIALIZE_BUILTIN_PLUGINS, 5237
ER_CANT_INITIALIZE_DYNAMIC_PLUGINS, 5237
ER_CANT_INITIALIZE_EARLY_PLUGINS, 5237
ER_CANT_INITIALIZE_GTID, 5237
ER_CANT_INITIALIZE_UDF, 5107
ER_CANT_INIT_DBS, 5235
ER_CANT_INIT_SCHEDULER_THREAD, 5225
ER_CANT_INIT_TC_LOG, 5224
ER_CANT_INIT_TIMER, 5239
ER_CANT_JOIN_SHUTDOWN_THREAD, 5238
ER_CANT_LOCK, 5099
ER_CANT_LOCK_LOG_TABLE, 5134
ER_CANT_LOCK_TABLE, 5305
ER_CANT_LOCK_TABLESPACE, 5305
ER_CANT_MODIFY_SRID_0, 5194
ER_CANT_MODIFY_SRS_USED_BY_COLUMN, 5195
ER_CANT_OPEN_AND_LOCK_PRIVILEGE_TABLES, 5257
ER_CANT_OPEN_CA, 5239
ER_CANT_OPEN_DATADIR_AFTER_UPGRADE_FAILURE, 5308
ER_CANT_OPEN_DB_OPT_USING_DEFAULT_CHARSET, 5306
ER_CANT_OPEN_DIR, 5308
ER_CANT_OPEN_ERROR_LOG, 5239
ER_CANT_OPEN_FILE, 5099
ER_CANT_OPEN_FRM_FILE, 5257
ER_CANT_OPEN_LIBRARY, 5107
ER_CANT_OPEN_TABLE_MYSQL_PROC, 5307
ER_CANT_PARSE_STORED_ROUTINE_BODY, 5307
ER_CANT_PROCESS_EXPRESSION_FOR_GENERATED_COLUMN_TO_DD, 5603
ER_CANT_READ_DIR, 5099
ER_CANT_READ_ERRMSGS, 5235
ER_CANT_READ_FRM_FILE, 5257
ER_CANT_READ_TABLE_MYSQL_PROC, 5308
ER_CANT_REMOVE_ALL_FIELDS, 5104
ER_CANT_REMOVE_PID_FILE, 5229
ER_CANT_RENAME_LOG_TABLE, 5135
ER_CANT_REOPEN_TABLE, 5108
ER_CANT_REPLICATE_ANONYMOUS_WITH_AUTO_POSITION, 5564
ER_CANT_REPLICATE_ANONYMOUS_WITH_GTID_MODE_ON, 5564
ER_CANT_REPLICATE_GTID_WITH_GTID_MODE_OFF, 5564
ER_CANT_RESET_MASTER, 5172
ER_CANT_SAVE_GTIDS, 5238
ER_CANT_SET_DATADIR, 5238
ER_CANT_SET_DATA_DIR, 5578
ER_CANT_SET_ERROR_LOG_SERVICE, 5193
ER_CANT_SET_ERROR_SUPPRESSION_LIST, 5197
ER_CANT_SET_ERROR_SUPPRESSION_LIST_FROM_COMMAND_LINE, 5570
ER_CANT_SET_GTID_MODE, 5166
ER_CANT_SET_GTID_NEXT_LIST_TO_NON_NULL_WHEN_GTID_MODE_IS_OFF, 5149
ER_CANT_SET_GTID_NEXT_TO_ANONYMOUS_WHEN_GTID_MODE_IS_ON, 5149
ER_CANT_SET_GTID_NEXT_TO_GTID_WHEN_GTID_MODE_IS_OFF, 5149
ER_CANT_SET_GTID_NEXT_WHEN_OWNING_GTID, 5150

ER_CANT_SET_GTID_PURGED_DUE_SETS_CONSTRAINTS, 5176
ER_CANT_SET_GTID_PURGED_WHEN_GTID_EXECUTED_IS_NOT_EMPTY, 5154
ER_CANT_SET_GTID_PURGED_WHEN_OWNED_GTIDS_IS_NOT_EMPTY, 5154
ER_CANT_SET_HANDLER_REFERENCE_FOR_TABLE, 5305
ER_CANT_SET_PATH_FOR, 5308
ER_CANT_SET_PERSISTED, 5242
ER_CANT_SET_UP_PERSISTED_VALUES, 5238
ER_CANT_SET_VARIABLE_WHEN_OWNING_GTID, 5168
ER_CANT_START_ERROR_LOG_SERVICE, 5326
ER_CANT_STAT_DATADIR, 5238
ER_CANT_STAT_FILE, 5316
ER_CANT_UPDATE_TABLE_IN_CREATE_TABLE_SELECT, 5147
ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG, 5127
ER_CANT_UPDATE_WITH_READLOCK, 5114
ER_CANT_UPGRADE_GENERATED_COLUMNS_TO_DD, 5306
ER_CANT_USE_AUTO_POSITION_WITH_GTID_MODE_OFF, 5166
ER_CANT_USE_OPTION_HERE, 5114
ER_CANT_WAIT_FOR_EXECUTED_GTID_SET_WHILE_OWNING_A_GTID, 5171
ER_CAPACITY_EXCEEDED, 5170
ER_CAPACITY_EXCEEDED_IN_PARSER, 5172
ER_CAPACITY_EXCEEDED_IN_RANGE_OPTIMIZER, 5171
ER_CA_SELF_SIGNED, 5227
ER_CHANGED_ENFORCE_GTID_CONSISTENCY, 5223
ER_CHANGED_GTID_MODE, 5223
ER_CHANGED_MAX_CONNECTIONS, 5234
ER_CHANGED_MAX_OPEN_FILES, 5234
ER_CHANGED_TABLE_OPEN_CACHE, 5235
ER_CHANGE_MASTER_PASSWORD_LENGTH, 5162
ER_CHANGE_MASTER_WRONG_COMPRESSION_ALGORITHM_CLIENT, 5218
ER_CHANGE_MASTER_WRONG_COMPRESSION_ALGORITHM_LIST_CLIENT, 5218
ER_CHANGE_MASTER_WRONG_COMPRESSION_LEVEL_CLIENT, 5218
ER_CHANGE_RPL_INFO_REPOSITORY_FAILURE, 5147
ER_CHECKING_TABLE, 5305
ER_CHECKREAD, 5100
ER_CHECK_CONSTRAINT_CLAUSE_USING_FK_REFER_ACTION_COLUMN, 5207
ER_CHECK_CONSTRAINT_DUP_NAME, 5207
ER_CHECK_CONSTRAINT_FUNCTION_IS_NOT_ALLOWED, 5206
ER_CHECK_CONSTRAINT_NAMED_FUNCTION_IS_NOT_ALLOWED, 5206
ER_CHECK_CONSTRAINT_NOT_FOUND, 5207
ER_CHECK_CONSTRAINT_REFERS_AUTO_INCREMENT_COLUMN, 5207
ER_CHECK_CONSTRAINT_REFERS_UNKNOWN_COLUMN, 5207
ER_CHECK_CONSTRAINT_ROW_VALUE, 5207
ER_CHECK_CONSTRAINT_VARIABLES, 5206
ER_CHECK_CONSTRAINT_VIOLATED, 5207
ER_CHECK_NOT_IMPLEMENTED, 5111
ER_CHECK_NO_SUCH_TABLE, 5111
ER_CLIENT_DOES_NOT_SUPPORT, 5186
ER_CLIENT_GTID_UNSAFE_CREATE_DROP_TEMP_TABLE_IN_TRX_IN_SBR, 5198
ER_CLIENT_KEYRING_UDF_KEY_INVALID, 5219
ER_CLIENT_KEYRING_UDF_KEY_TOO_LONG, 5219
ER_CLIENT_KEYRING_UDF_KEY_TYPE_INVALID, 5219
ER_CLIENT_KEYRING_UDF_KEY_TYPE_TOO_LONG, 5219
ER_CLIENT_PRIVILEGE_CHECKS_USER_CANNOT_BE_ANONYMOUS, 5218
ER_CLIENT_PRIVILEGE_CHECKS_USER_CORRUPT, 5218

ER_CLIENT_PRIVILEGE_CHECKS_USER_DOES_NOT_EXIST, 5218
ER_CLIENT_PRIVILEGE_CHECKS_USER_NEEDS_RPL_APPLIER_PRIV, 5219
ER_CLONE_CHARSET, 5212
ER_CLONE_CLIENT_TRACE, 5578
ER_CLONE_CONFIG, 5212
ER_CLONE_CREATE_HANDLER_FAIL_TRACE, 5337
ER_CLONE_DDL_IN_PROGRESS, 5186
ER_CLONE_DISALLOWED, 5213
ER_CLONE_DISK_SPACE, 5213
ER_CLONE_DONOR, 5212
ER_CLONE_DONOR_TRACE, 5577
ER_CLONE_DONOR_VERSION, 5212
ER_CLONE_ENCRYPTION, 5213
ER_CLONE_HANDLER_EXISTS, 5337
ER_CLONE_HANDLER_EXIST_TRACE, 5337
ER_CLONE_INFO_CLIENT, 5578
ER_CLONE_INFO_SERVER, 5578
ER_CLONE_IN_PROGRESS, 5213
ER_CLONE_LOOPBACK, 5212
ER_CLONE_OS, 5212
ER_CLONE_PLATFORM, 5212
ER_CLONE_PLUGIN_MATCH, 5212
ER_CLONE_PLUGIN_NOT_LOADED, 5337
ER_CLONE_PLUGIN_NOT_LOADED_TRACE, 5337
ER_CLONE_PROTOCOL, 5212
ER_CLONE_PROTOCOL_ERROR, 5578
ER_CLONE_PROTOCOL_TRACE, 5578
ER_CLONE_REMOTE_ERROR, 5577
ER_CLONE_SERVER_TRACE, 5578
ER_CLONE_SHUTDOWN_TRACE, 5598
ER_CLONE_SYS_CONFIG, 5212
ER_CLONE_TOO_MANY_CONCURRENT_CLONES, 5186
ER_CMD_NEED_SUPER, 5185
ER_COALESCE_ONLY_ON_HASH_PARTITION, 5132
ER_COALESCE_PARTITION_NO_PARTITION, 5132
ER_COLLATION_CHARSET_MISMATCH, 5115
ER_COLUMNACCESS_DENIED_ERROR, 5108
ER_COLUMN_CHANGE_SIZE, 5214
ER_COLUMN_CHECK_CONSTRAINT_REFERENCES_OTHER_COLUMN, 5206
ER_COL_COUNT_DOESNT_MATCH_CORRUPTED_V2, 5151
ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE, 5134
ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE_V2, 5158
ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG, 5126
ER_COMPONENTS_CANT_ACQUIRE_SERVICE_IMPLEMENTATION, 5175
ER_COMPONENTS_CANT_LOAD, 5175
ER_COMPONENTS_CANT_RELEASE_SERVICE, 5176
ER_COMPONENTS_CANT_SATISFY_DEPENDENCY, 5175
ER_COMPONENTS_CANT_UNLOAD, 5176
ER_COMPONENTS_FAILED_TO_ACQUIRE_SERVICE_IMPLEMENTATION, 5341
ER_COMPONENTS_INFRASTRUCTURE_BOOTSTRAP, 5226
ER_COMPONENTS_INFRASTRUCTURE_SHUTDOWN, 5227
ER_COMPONENTS_LOAD_CANT_INITIALIZE, 5176
ER_COMPONENTS_LOAD_CANT_REGISTER_SERVICE_IMPLEMENTATION, 5176
ER_COMPONENTS_NO_SCHEME, 5175

ER_COMPONENTS_NO_SCHEME_SERVICE, 5175
ER_COMPONENTS_PERSIST_LOADER_BOOTSTRAP, 5227
ER_COMPONENTS_UNLOAD_CANT_DEINITIALIZE, 5176
ER_COMPONENTS_UNLOAD_CANT_UNREGISTER_SERVICE, 5176
ER_COMPONENTS_UNLOAD_DUPLICATE_IN_GROUP, 5176
ER_COMPONENTS_UNLOAD_NOT_LOADED, 5176
ER_COMPONENT_FILTER_CONFUSED, 5363
ER_COMPONENT_FILTER_DIAGNOSTICS, 5193
ER_COMPONENT_FILTER_FLABBERGASTED, 5184
ER_COMPONENT_FILTER_WRONG_VALUE, 5561
ER_COMPONENT_MANIPULATE_ROW_FAILED, 5176
ER_COMPONENT_TABLE_INCORRECT, 5176
ER_COND_ITEM_TOO_LONG, 5139
ER_CONFIG_OPTION_WITHOUT_GROUP, 5242
ER_CONFIRMING_THE_FUTURE, 5231
ER_CONFLICTING_DECLARATIONS, 5119
ER_CONFLICT_FN_PARSE_ERROR, 5138
ER_CONNECTION_ABORTED, 5560
ER_CONNECTION_HANDLING_OOM, 5229
ER_CONNECT_TO_FOREIGN_DATA_SOURCE, 5126
ER_CONNECT_TO_MASTER, 5113
ER_CONN_ATTR_TRUNCATED, 5250
ER_CONN_CONTROL_DELAY_ACTION_INIT_FAILED, 5364
ER_CONN_CONTROL_ERROR_MSG, 5362
ER_CONN_CONTROL_EVENT_COORDINATOR_INIT_FAILED, 5364
ER_CONN_CONTROL_FAILED_TO_SET_CONN_DELAY, 5365
ER_CONN_CONTROL_FAILED_TO_UPDATE_CONN_DELAY_HASH, 5365
ER_CONN_CONTROL_INVALID_CONN_DELAY_TYPE, 5364
ER_CONN_CONTROL_STAT_CONN_DELAY_TRIGGERED_RESET_FAILED, 5364
ER_CONN_CONTROL_STAT_CONN_DELAY_TRIGGERED_UPDATE_FAILED, 5364
ER_CONN_INIT_CONNECT_IGNORED, 5363
ER_CONN_PER_THREAD_NO_THREAD, 5246
ER_CONN_PIP_CANT_CREATE_EVENT, 5246
ER_CONN_PIP_CANT_CREATE_PIPE, 5246
ER_CONN_SHM_CANT_CREATE_CONNECTION, 5246
ER_CONN_SHM_CANT_CREATE_SERVICE, 5245
ER_CONN_SHM_LISTENER, 5245
ER_CONN_SOCKET_ACCEPT_FAILED, 5250
ER_CONN_SOCKET_SELECT_FAILED, 5249
ER_CONN_TCP_ADDRESS, 5246
ER_CONN_TCP_BIND_FAIL, 5247
ER_CONN_TCP_BIND_RETRY, 5247
ER_CONN_TCP_CANT_RESET_V6ONLY, 5247
ER_CONN_TCP_CANT_RESOLVE_HOSTNAME, 5247
ER_CONN_TCP_CREATED, 5246
ER_CONN_TCP_ERROR_WITH_STRERROR, 5246
ER_CONN_TCP_IPV6_AVAILABLE, 5246
ER_CONN_TCP_IPV6_UNAVAILABLE, 5246
ER_CONN_TCP_IP_NOT_LOGGED, 5247
ER_CONN_TCP_IS_THERE_ANOTHER_USING_PORT, 5247
ER_CONN_TCP_LISTEN_FAIL, 5248
ER_CONN_TCP_NO_SOCKET, 5246
ER_CONN_TCP_RESOLVE_INFO, 5248
ER_CONN_TCP_START_FAIL, 5248

ER_CONN_TPC_BIND_FAIL, 5247
ER_CONN_UNIX_IS_THERE_ANOTHER_USING_SOCKET, 5247
ER_CONN_UNIX_LISTEN_FAILED, 5248
ER_CONN_UNIX_LOCK_FILE_CANT_CLOSE, 5249
ER_CONN_UNIX_LOCK_FILE_CANT_CREATE, 5248
ER_CONN_UNIX_LOCK_FILE_CANT_DELETE, 5249
ER_CONN_UNIX_LOCK_FILE_CANT_OPEN, 5249
ER_CONN_UNIX_LOCK_FILE_CANT_READ, 5249
ER_CONN_UNIX_LOCK_FILE_CANT_SYNC, 5249
ER_CONN_UNIX_LOCK_FILE_CANT_WRITE, 5249
ER_CONN_UNIX_LOCK_FILE_EMPTY, 5249
ER_CONN_UNIX_LOCK_FILE_FAIL, 5248
ER_CONN_UNIX_LOCK_FILE_GIVING_UP, 5248
ER_CONN_UNIX_LOCK_FILE_PIDLESS, 5249
ER_CONN_UNIX_NO_BIND_NO_START, 5248
ER_CONN_UNIX_NO_FD, 5248
ER_CONN_UNIX_PATH_TOO_LONG, 5248
ER_CONN_UNIX_PID_CLAIMED_SOCKET_FILE, 5247
ER_CONSECUTIVE_REORG_PARTITIONS, 5132
ER_CON_COUNT_ERROR, 5101
ER_CORE_VALUES, 5231
ER_CORRUPTED_JSON_DIFF, 5187
ER_CORRUPT_HELP_DB, 5115
ER_COULD_NOT_APPLY_JSON_DIFF, 5187
ER_COULD_NOT_CREATE_WINDOWS_REGISTRY_KEY, 5573
ER_COULD_NOT_REINITIALIZE_AUDIT_LOG_FILTERS, 5566
ER_CRASHED_ON_REPAIR, 5111
ER_CRASHED_ON_USAGE, 5111
ER_CREATED_SYSTEM_WITH_VERSION, 5335
ER_CREATE_FILEGROUP_FAILED, 5133
ER_CREATING_NEW_UUID, 5228
ER_CREATING_NEW_UUID_FIRST_START, 5326
ER_CREDENTIALLESS_AUTO_USER_BAD, 5229
ER_CREDENTIALS_CONTRADICT_TO_HISTORY, 5186
ER_CTE_MAX_RECURSION_DEPTH, 5186
ER_CTE_RECURSIVE_FORBIDDEN_JOIN_ORDER, 5179
ER_CTE_RECURSIVE_FORBIDS_AGGREGATION, 5179
ER_CTE_RECURSIVE_REQUIRES_NONRECURSIVE_FIRST, 5179
ER_CTE_RECURSIVE_REQUIRES_SINGLE_REFERENCE, 5179
ER_CTE_RECURSIVE_REQUIRES_UNION, 5179
ER_CURRENT_PASSWORD_CANNOT_BE_RETAINED, 5215
ER_CURRENT_PASSWORD_NOT_REQUIRED, 5215
ER_CUT_VALUE_GROUP_CONCAT, 5116
ER_CYCLE_TIMER_IS_NOT_AVAILABLE, 5337
ER_DATABASE_NAME, 5138
ER_DATA_DIRECTORY_UNUSABLE, 5573
ER_DATA_DIRECTORY_UNUSABLE_DELETABLE, 5597
ER_DATA_OUT_OF_RANGE, 5143
ER_DATA_TOO_LONG, 5125
ER_DATETIME_FUNCTION_OVERFLOW, 5127
ER_DA_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES, 5213
ER_DA_GRP_RPL_STARTED_AUTO_REJOIN, 5217
ER_DA_INVALID_CONDITION_NUMBER, 5148
ER_DA_PLUGIN_INSTALL_ERROR, 5214

ER_DA_RPL_GTID_TABLE_CANNOT_OPEN, 5213
ER_DA_SSL_LIBRARY_ERROR, 5214
ER_DA_UDF_INVALID_CHARSET, 5219
ER_DA_UDF_INVALID_CHARSET_SPECIFIED, 5219
ER_DA_UDF_INVALID_COLLATION, 5220
ER_DA_UDF_INVALID_EXTENSION_ARGUMENT_TYPE, 5220
ER_DA_UNKNOWN_ERROR_NUMBER, 5214
ER_DBACCESS_DENIED_ERROR, 5101
ER_DBUG_CHECK_SHARES_DROPPED, 5277
ER_DBUG_CHECK_SHARES_INFO, 5277
ER_DBUG_CHECK_SHARES_OPEN, 5277
ER_DB_ACCESS_DENIED, 5213
ER_DB_CREATE_EXISTS, 5099
ER_DB_DROP_EXISTS, 5099
ER_DB_DROP_RMDIR, 5099
ER_DB_DROP_RMDIR2, 5183
ER_DB_OPT_NOT_FOUND_USING_DEFAULT_CHARSET, 5306
ER_DDL_IN_PROGRESS, 5185
ER_DDL_LOG_ERROR, 5135
ER_DD_ABORTING_PARTIAL_UPGRADE, 5255
ER_DD_CACHE_NOT_EMPTY_AT_SHUTDOWN, 5222
ER_DD_CANT_CREATE_OBJECT_KEY, 5223
ER_DD_CANT_CREATE_SP, 5306
ER_DD_CANT_FETCH_TABLE_DATA, 5306
ER_DD_CANT_FIX_SE_DATA, 5306
ER_DD_CANT_GET_OBJECT_KEY, 5223
ER_DD_CANT_RESOLVE_VIEW, 5241
ER_DD_CREATED_FOR_UPGRADE, 5256
ER_DD_ERROR_CREATING_ENTRY, 5306
ER_DD_FAILSAFE, 5220
ER_DD_FRM_EXISTS_FOR_TABLE, 5255
ER_DD_INITIALIZE, 5343
ER_DD_INITIALIZE_SQL_ERROR, 5567
ER_DD_INIT_FAILED, 5222
ER_DD_INIT_UPGRADE_FAILED, 5221
ER_DD_METADATA_NOT_FOUND, 5222
ER_DD_MINOR_DOWNGRADE, 5344
ER_DD_MINOR_DOWNGRADE_VERSION_NOT_SUPPORTED, 5344
ER_DD_NO_VERSION_FOUND, 5344
ER_DD_NO_WRITES_NO_REPOPULATION, 5220
ER_DD_OBJECT_RELEASER_REMAINS, 5223
ER_DD_OBJECT_REMAINS, 5222
ER_DD_OBJECT_REMAINS_IN_RELEASER, 5223
ER_DD_POPULATING_TABLES_FAILED, 5222
ER_DD_RESTART, 5343
ER_DD_SCHEMA_NOT_FOUND, 5255
ER_DD_SE_INIT_FAILED, 5255
ER_DD_TABLESPACE_NOT_FOUND, 5240
ER_DD_TABLE_NOT_FOUND, 5255
ER_DD_TRG_CANT_ADD, 5241
ER_DD_TRG_CONNECTION_COLLATION_MISSING, 5240
ER_DD_TRG_DB_COLLATION_MISSING, 5240
ER_DD_TRG_DEFINER_OOM, 5240
ER_DD_TRG_FILE_UNREADABLE, 5240

ER_DD_UPDATING_PLUGIN_MD_FAILED, 5222
ER_DD_UPGRADE, 5343
ER_DD_UPGRADE_COMPLETED, 5593
ER_DD_UPGRADE_DD_OPEN_FAILED, 5332
ER_DD_UPGRADE_DD_POPULATED, 5333
ER_DD_UPGRADE_FAILED_FIND_VALID_DATA_DIR, 5334
ER_DD_UPGRADE_FAILED_INIT_DD_SE, 5334
ER_DD_UPGRADE_FAILED_TO_ACQUIRE_TABLESPACE, 5333
ER_DD_UPGRADE_FAILED_TO_CREATE_INDEX_STATS, 5334
ER_DD_UPGRADE_FAILED_TO_CREATE_TABLE_STATS, 5334
ER_DD_UPGRADE_FAILED_TO_FETCH_TABLES, 5333
ER_DD_UPGRADE_FAILED_TO_FETCH_TABLESPACES, 5333
ER_DD_UPGRADE_FAILED_TO_RESOLVE_TABLESPACE_ENGINE, 5333
ER_DD_UPGRADE_FAILED_TO_UPDATE_VER_NO_IN_TABLESPACE, 5343
ER_DD_UPGRADE_FOUND_PARTIALLY_UPGRADED_DD_ABORT, 5334
ER_DD_UPGRADE_FOUND_PARTIALLY_UPGRADED_DD_CONTINUE, 5335
ER_DD_UPGRADE_INDEX_STATS_MIGRATE_COMPLETED, 5334
ER_DD_UPGRADE_INFO_FILE_CLOSE_FAILED, 5333
ER_DD_UPGRADE_INFO_FILE_OPEN_FAILED, 5333
ER_DD_UPGRADE_OFF, 5344
ER_DD_UPGRADE_RENAME_IDX_STATS_FILE_FAILED, 5332
ER_DD_UPGRADE_SCHEMA_UNAVAILABLE, 5344
ER_DD_UPGRADE_SDI_INFO_UPDATE_FAILED, 5335
ER_DD_UPGRADE_SE_LOGS_FAILED, 5335
ER_DD_UPGRADE_START, 5334
ER_DD_UPGRADE_TABLESPACE_MIGRATION_FAILED, 5334
ER_DD_UPGRADE_TABLE_INTACT_ERROR, 5340
ER_DD_UPGRADE_TABLE_STATS_MIGRATE_COMPLETED, 5334
ER_DD_UPGRADE_VERSION_NOT_SUPPORTED, 5344
ER_DD_UPGRADE_VIEW_COLUMN_NAME_TOO_LONG, 5342
ER_DD_VERSION_FOUND, 5220
ER_DD_VERSION_INSTALLED, 5220
ER_DD_VERSION_UNSUPPORTED, 5221
ER_DD_VIEW_CANT_ALLOC_CHARSET, 5222
ER_DD_VIEW_CANT_CREATE, 5222
ER_DD_VIEW_WITHOUT_DEFINER, 5241
ER_DEBUG_SYNC_EXECUTED, 5224
ER_DEBUG_SYNC_HIT, 5224
ER_DEBUG_SYNC_HIT_LIMIT, 5139
ER_DEBUG_SYNC_OOM, 5224
ER_DEBUG_SYNC_THREAD_MAX, 5224
ER_DEBUG_SYNC_TIMEOUT, 5139
ER_DEFAULT_AS_VAL_GENERATED, 5201
ER_DEFAULT_SE_UNAVAILABLE, 5239
ER_DEFAULT_VAL_GENERATED_FUNCTION_IS_NOT_ALLOWED, 5201
ER_DEFAULT_VAL_GENERATED_NAMED_FUNCTION_IS_NOT_ALLOWED, 5201
ER_DEFAULT_VAL_GENERATED_NON_PRIOR, 5201
ER_DEFAULT_VAL_GENERATED_REF_AUTO_INC, 5201
ER_DEFAULT_VAL_GENERATED_ROW_VALUE, 5201
ER_DEFAULT_VAL_GENERATED_VARIABLES, 5201
ER_DEPART_WITH_GRACE, 5227
ER_DEPENDENT_BY_DEFAULT_GENERATED_VALUE, 5200
ER_DEPENDENT_BY_FUNCTIONAL_INDEX, 5209
ER_DEPENDENT_BY_GENERATED_COLUMN, 5166

ER_DEPENDENT_BY_PARTITION_FUNC, 5211
ER_DEPRECATED_NATIONAL, 5195
ER_DEPRECATED_SYNTAX_NO_REPLACEMENT, 5341
ER_DEPRECATED_SYNTAX_WITH_REPLACEMENT, 5341
ER_DEPRECATED_TIMESTAMP_IMPLICIT_DEFAULTS, 5229
ER_DEPRECATED_UTF8_ALIAS, 5195
ER_DEPRECATE_MSG_NO_REPLACEMENT, 5341
ER_DEPRECATE_MSG_WITH_REPLACEMENT, 5324
ER_DERIVED_MUST_HAVE_ALIAS, 5115
ER_DES_FILE_WRONG_KEY, 5242
ER_DETACHED_SESSIONS_LEFT_BY_PLUGIN, 5271
ER_DETACHING_SESSION_LEFT_BY_PLUGIN, 5271
ER_DIMENSION_UNSUPPORTED, 5163
ER_DISABLED_STORAGE_ENGINE, 5170
ER_DISABLED_STORAGE_ENGINE_AS_DEFAULT, 5223
ER_DISALLOWED_OPERATION, 5188
ER_DISCARD_FK_CHECKS_RUNNING, 5151
ER_DISCONNECTING_REMAINING_CLIENTS, 5232
ER_DISK_FULL_NOWAIT, 5190
ER_DIVISION_BY_ZERO, 5123
ER_DONT_SUPPORT_SLAVE_PRESERVE_COMMIT_ORDER, 5160
ER_DROP_DATABASE_FAILED_RMDIR_MANUALLY, 5342
ER_DROP_FILEGROUP_FAILED, 5133
ER_DROP_INDEX_FK, 5134
ER_DROP_LAST_PARTITION, 5131
ER_DROP_PARTITION_NON_EXISTENT, 5131
ER_DUPLICATED_VALUE_IN_TYPE, 5118
ER_DUPLICATE_OPTION_KEY, 5178
ER_DUPLICATE_SYS_VAR, 5568
ER_DUPLICATE_TABLE_LOCK, 5178
ER_DUP_ARGUMENT, 5114
ER_DUP_ENTRY, 5103
ER_DUP_ENTRY_AUTOINCREMENT_CASE, 5135
ER_DUP_ENTRY_WITH_KEY_NAME, 5136
ER_DUP_FD_OPEN_FAILED, 5326
ER_DUP_FIELDNAME, 5102
ER_DUP_INDEX, 5153
ER_DUP_KEY, 5100
ER_DUP_KEYNAME, 5102
ER_DUP_LIST_ENTRY, 5159
ER_DUP_SIGNAL_SET, 5139
ER_DUP_UNIQUE, 5110
ER_DUP_UNKNOWN_IN_INDEX, 5155
ER_EMPTY_PIPELINE_FOR_ERROR_LOG_SERVICE, 5193
ER_EMPTY_QUERY, 5103
ER_ENDING_INIT, 5567
ER_END_INITFILE, 5234
ER_ENFORCE_GTID_CONSISTENCY_WARN_WITH_ONGOING_GTID_VIOLATING_TX, 5166
ER_ENGINE_CANT_DROP_MISSING_TABLE, 5183
ER_ENGINE_CANT_DROP_TABLE, 5182
ER_ENGINE_COST_FAILED_TO_READ, 5268
ER_ENGINE_COST_INVALID_CONST_CONSTANT_FOR_SE_AND_DEVICE, 5268
ER_ENGINE_COST_INVALID_DEVICE_TYPE_FOR_SE, 5267
ER_ENGINE_COST_UNKNOWN_COST_CONSTANT, 5267

ER_ENGINE_COST_UNKNOWN_STORAGE_ENGINE, 5267
ER_ENGINE_OUT_OF_MEMORY, 5159
ER_ERRMSG_CANT_FIND_FILE, 5256
ER_ERRMSG_CANT_READ, 5256
ER_ERRMSG_LOADING_55_STYLE, 5256
ER_ERRMSG_MISSING_IN_FILE, 5256
ER_ERRMSG_OOM, 5256
ER_ERRMSG_REPLACEMENTS_FAILED, 5320
ER_ERRMSG_REPLACEMENT_DODGY, 5320
ER_ERROR_DURING_COMMIT, 5111
ER_ERROR_DURING_FLUSH_LOGS, 5111
ER_ERROR_DURING_FLUSH_LOG_COMMIT_PHASE, 5342
ER_ERROR_DURING_OPTIMIZE_TABLE, 5304
ER_ERROR_DURING_ROLLBACK, 5111
ER_ERROR_ENABLING_KEYS, 5305
ER_ERROR_INFO_FROM_DA, 5563
ER_ERROR_IN_TRIGGER_BODY, 5144
ER_ERROR_IN_UNKNOWN_TRIGGER_BODY, 5144
ER_ERROR_ON_MASTER, 5158
ER_ERROR_ON_MODIFYING_GTID_EXECUTED_TABLE, 5171
ER_ERROR_ON_READ, 5100
ER_ERROR_ON_RENAME, 5100
ER_ERROR_ON_WRITE, 5100
ER_ERROR_WHEN_EXECUTING_COMMAND, 5113
ER_EVENT_ALREADY_EXISTS, 5133
ER_EVENT_CANNOT_ALTER_IN_THE_PAST, 5136
ER_EVENT_CANNOT_CREATE_IN_THE_PAST, 5136
ER_EVENT_CANT_FIND_TIMEZONE, 5307
ER_EVENT_CANT_GET_CHARSET, 5307
ER_EVENT_CANT_GET_COLLATION, 5307
ER_EVENT_CANT_GET_LOCK_FOR_DROPPING_EVENT, 5311
ER_EVENT_CANT_GET_TIMEZONE_FROM_FIELD, 5306
ER_EVENT_CANT_INIT_QUEUE, 5224
ER_EVENT_CANT_OPEN_TABLE_MYSQL_EVENT, 5307
ER_EVENT_DOES_NOT_EXIST, 5133
ER_EVENT_DROPPING, 5269
ER_EVENT_ENDS_BEFORE_STARTS, 5133
ER_EVENT_ERROR_CREATING_QUERY_TO_WRITE_TO_BINLOG, 5310
ER_EVENT_ERROR_DURING_COMPILATION, 5269
ER_EVENT_EXECUTION_FAILED, 5224
ER_EVENT_EXECUTION_FAILED_CANT_AUTHENTICATE_USER, 5269
ER_EVENT_EXECUTION_FAILED_USER_LOST_EVEN_PRIVILEGE, 5269
ER_EVENT_EXEC_TIME_IN_THE_PAST, 5134
ER_EVENT_INTERVAL_NOT_POSITIVE_OR_TOO_BIG, 5133
ER_EVENT_INVALID_CREATION_CTX, 5137
ER_EVENT_LAST_EXECUTION, 5224
ER_EVENT_MESSAGE_STACK, 5224
ER_EVENT_PURGING_QUEUE, 5224
ER_EVENT_RECURSION_FORBIDDEN, 5135
ER_EVENT_SAME_NAME, 5134
ER_EVENT_SCHEDULER_ERROR_GETTING_EVENT_OBJECT, 5310
ER_EVENT_SCHEDULER_ERROR_LOADING_FROM_DB, 5310
ER_EVENT_SCHEDULER_GOT_BAD_DATA_FROM_TABLE, 5310
ER_EVENT_SET_VAR_ERROR, 5135

ER_EVENT_UNABLE_TO_DROP_EVENT, 5311
ER_EXCEEDED_MV_KEYS_NUM, 5216
ER_EXCEEDED_MV_KEYS_SPACE, 5216
ER_EXCEPTIONS_WRITE_ERROR, 5138
ER_EXCESS_ARGUMENTS, 5235
ER_EXPIRE_LOGS_DAYS_IGNORED, 5342
ER_EXPLAIN_NOT_SUPPORTED, 5158
ER_FAILED_DEFAULT_ROLES, 5175
ER_FAILED_READ_FROM_PAR_FILE, 5143
ER_FAILED_REVOKE_ROLE, 5175
ER_FAILED_ROLE_GRANT, 5175
ER_FAILED_START_MYSQLD_DAEMON, 5327
ER_FAILED_TO_ACQUIRE_LOCK_ON_RESOURCE_GROUP, 5331
ER_FAILED_TO_ADD_RESOURCE_GROUP_TO_MAP, 5330
ER_FAILED_TO_ADD_RPL_FILTER, 5336
ER_FAILED_TO_ALLOCATE_MEMORY_FOR_RESOURCE_GROUP, 5330
ER_FAILED_TO_ALLOCATE_MEMORY_FOR_RESOURCE_GROUP_HASH, 5330
ER_FAILED_TO_ALLOCATE_SSL_BIO, 5568
ER_FAILED_TO_APPLY_RESOURCE_GROUP_CONTROLLER, 5330
ER_FAILED_TO_BINLOG_DROP_EVENT, 5336
ER_FAILED_TO_BUILD_DO_AND_IGNORE_TABLE_HASHES, 5337
ER_FAILED_TO_COMPRESS_GTID_EXECUTED_TABLE, 5261
ER_FAILED_TO_COMPRESS_GTID_EXECUTED_TABLE_OOM, 5261
ER_FAILED_TO_CONSTRUCT_DROP_EVENT_QUERY, 5335
ER_FAILED_TO_CREATE_CLONE_HANDLER, 5337
ER_FAILED_TO_CREATE_GTID_TABLE_COMPRESSION_THREAD, 5261
ER_FAILED_TO_CREATE_SDI_FOR_TABLESPACE, 5333
ER_FAILED_TO_DECREMENT_NUMBER_OF_THREADS, 5271
ER_FAILED_TO_DELETE_FROM_GTID_EXECUTED_TABLE, 5261
ER_FAILED_TO_DESERIALIZE_RESOURCE_GROUP, 5330
ER_FAILED_TO_FIND_COLLATION_NAME, 5325
ER_FAILED_TO_FIND_DL_ENTRY, 5322
ER_FAILED_TO_FIND_LOCALE_NAME, 5325
ER_FAILED_TO_FIND_MYSQLD_STATUS, 5341
ER_FAILED_TO_GENERATE_UNIQUE_LOGFILE, 5322
ER_FAILED_TO_GET_ABSOLUTE_PATH, 5326
ER_FAILED_TO_HANDLE_DEFAULTS_FILE, 5568
ER_FAILED_TO_INIT_SYS_VAR, 5568
ER_FAILED_TO_INIT_THREAD_ATTR_FOR_GTID_TABLE_COMPRESSION, 5261
ER_FAILED_TO_JOIN_GTID_TABLE_COMPRESSION_THREAD, 5261
ER_FAILED_TO_LOCK_MEM, 5234
ER_FAILED_TO_OPEN_COST_CONSTANT_TABLES, 5268
ER_FAILED_TO_OPEN_SHARED_LIBRARY, 5322
ER_FAILED_TO_PERSIST_RESOURCE_GROUP_METADATA, 5329
ER_FAILED_TO_READ_FILE, 5322
ER_FAILED_TO_REMOVE_TEMP_TABLE, 5301
ER_FAILED_TO_REPAIR_TABLE, 5301
ER_FAILED_TO_SET_PERSISTED_OPTIONS, 5341
ER_FAILED_TO_START_SLAVE_THREAD, 5336
ER_FAILED_TO_STORE_SDI_FOR_TABLESPACE, 5333
ER_FAILED_TO_UPDATE_RESOURCE_GROUP, 5330
ER_FAILED_TO_WRITE_TO_FILE, 5322
ER_FAIL_CHROOT, 5233
ER_FAIL_SETGID, 5233

ER_FAIL_SETREGID, 5233
ER_FAIL_SETREUID, 5233
ER_FAIL_SETUID, 5233
ER_FEATURE_DISABLED, 5118
ER_FEATURE_DISABLED_SEE_DOC, 5170
ER_FEATURE_NOT_AVAILABLE, 5166
ER_FEATURE_UNSUPPORTED, 5188
ER_FIELD_IN_GROUPING_NOT_GROUP_BY, 5182
ER_FIELD_IN_ORDER_NOT_SELECT, 5162
ER_FIELD_NOT_FOUND_PART_ERROR, 5130
ER_FIELD_SPECIFIED_TWICE, 5106
ER_FIELD_TYPE_NOT_ALLOWED_AS_PARTITION_FIELD, 5140
ER_FILEGROUP_OPTION_ONLY_ONCE, 5133
ER_FILESORT_TERMINATED, 5325
ER_FILE_CORRUPT, 5158
ER_FILE_EXISTS_DURING_UPGRADE, 5308
ER_FILE_EXISTS_ERROR, 5104
ER_FILE_HAS_OLD_FORMAT, 5324
ER_FILE_NOT_FOUND, 5099
ER_FILE_PRIVILEGE_FOR_REPLICATION_CHECKS, 5602
ER_FILE_TYPE_UNKNOWN, 5257
ER_FILE_USED, 5100
ER_FILSORT_ABORT, 5100
ER_FIREWALL_ACCESS_DENIED, 5355
ER_FIREWALL_FAILED_TO_READ_FIREWALL_TABLES, 5354
ER_FIREWALL_FAILED_TO_REG_DYNAMIC_PRIVILEGES, 5354
ER_FIREWALL_RECORDING_STMT_WAS_TRUNCATED, 5355
ER_FIREWALL_RECORDING_STMT_WITHOUT_TEXT, 5355
ER_FIREWALL_RELOADING_CACHE, 5355
ER_FIREWALL_RESET_FOR_USER, 5355
ER_FIREWALL_SKIPPED_UNKNOWN_USER_MODE, 5355
ER_FIREWALL_STATUS_FLUSHED, 5355
ER_FIREWALL_SUSPICIOUS_STMT, 5355
ER_FIXING_CLIENT_CHARSET, 5234
ER_FK_CANNOT_CHANGE_ENGINE, 5202
ER_FK_CANNOT_DROP_PARENT, 5196
ER_FK_CANNOT_OPEN_PARENT, 5152
ER_FK_CANNOT_USE_VIRTUAL_COLUMN, 5197
ER_FK_COLUMN_CANNOT_CHANGE, 5153
ER_FK_COLUMN_CANNOT_CHANGE_CHILD, 5153
ER_FK_COLUMN_CANNOT_DROP, 5153
ER_FK_COLUMN_CANNOT_DROP_CHILD, 5153
ER_FK_COLUMN_NOT_NULL, 5153
ER_FK_DEPTH_EXCEEDED, 5158
ER_FK_DUP_NAME, 5153
ER_FK_FAIL_ADD_SYSTEM, 5152
ER_FK_INCOMPATIBLE_COLUMNS, 5202
ER_FK_INCORRECT_OPTION, 5152
ER_FK_NO_COLUMN_PARENT, 5197
ER_FK_NO_INDEX_CHILD, 5152
ER_FK_NO_INDEX_PARENT, 5152
ER_FORBID_SCHEMA_CHANGE, 5128
ER_FORCE_CLOSE_THREAD, 5323
ER_FORCING_CLOSE, 5104

ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST, 5126
ER_FOREIGN_DATA_STRING_INVALID, 5127
ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE, 5127
ER_FOREIGN_DUPLICATE_KEY_OLD_UNUSED, 5134
ER_FOREIGN_DUPLICATE_KEY_WITHOUT_CHILD_INFO, 5148
ER_FOREIGN_DUPLICATE_KEY_WITH_CHILD_INFO, 5148
ER_FOREIGN_KEY_ON_PARTITIONED, 5131
ER_FOREIGN_SERVER_DOESNT_EXIST, 5130
ER_FOREIGN_SERVER_EXISTS, 5130
ER_FOUND_MISSING_GTIDS, 5428
ER_FOUND_ROWS_WHILE_REPAIRING, 5304
ER_FPARSER_BAD_HEADER, 5121
ER_FPARSER_EOF_IN_COMMENT, 5121
ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER, 5121
ER_FPARSER_ERROR_IN_PARAMETER, 5121
ER_FPARSER_TOO_BIG_FILE, 5121
ER_FSEEK_FAIL, 5123
ER_FT_BOOL_SYNTAX_INVALID, 5229
ER_FT_MATCHING_KEY_NOT_FOUND, 5111
ER_FULLTEXT_FUNCTIONAL_INDEX, 5200
ER_FULLTEXT_NOT_SUPPORTED_WITH_PARTITIONING, 5147
ER_FUNCTIONAL_INDEX_DATA_IS_TOO_LONG, 5216
ER_FUNCTIONAL_INDEX_FUNCTION_IS_NOT_ALLOWED, 5200
ER_FUNCTIONAL_INDEX_ON_FIELD, 5200
ER_FUNCTIONAL_INDEX_ON_JSON_OR_GEOMETRY_FUNCTION, 5199
ER_FUNCTIONAL_INDEX_ON_LOB, 5199
ER_FUNCTIONAL_INDEX_PRIMARY_KEY, 5199
ER_FUNCTIONAL_INDEX_REF_AUTO_INCREMENT, 5199
ER_FUNCTIONAL_INDEX_ROW_VALUE_IS_NOT_ALLOWED, 5204
ER_FUNCTION_NOT_DEFINED, 5107
ER_FUNC_INEXISTENT_NAME_COLLISION, 5138
ER_FUTURE_DATE, 5232
ER_GENERATED_COLUMN_FUNCTION_IS_NOT_ALLOWED, 5165
ER_GENERATED_COLUMN_NAMED_FUNCTION_IS_NOT_ALLOWED, 5200
ER_GENERATED_COLUMN_NON_PRIOR, 5166
ER_GENERATED_COLUMN_REF_AUTO_INC, 5166
ER_GENERATED_COLUMN_ROW_VALUE, 5200
ER_GENERATED_COLUMN_VARIABLES, 5200
ER_GEOMETRY_IN_UNKNOWN_LENGTH_UNIT, 5214
ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE, 5196
ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE, 5196
ER_GET_ERRMSG, 5118
ER_GET_ERRNO, 5100
ER_GET_ERRNO_FROM_STORAGE_ENGINE, 5324
ER_GET_STACKED_DA_WITHOUT_ACTIVE_HANDLER, 5158
ER_GET_TEMPORARY_ERRMSG, 5118
ER_GIS_DATA_WRONG_ENDIANESS, 5161
ER_GIS_DIFFERENT_SRIDS, 5160
ER_GIS_INVALID_DATA, 5160
ER_GIS_MAX_POINTS_IN_GEOMETRY_OVERFLOWED, 5168
ER_GIS_UNKNOWN_ERROR, 5160
ER_GIS_UNKNOWN_EXCEPTION, 5160
ER_GIS_UNSUPPORTED_ARGUMENT, 5160
ER_GLOBAL_VARIABLE, 5114

ER_GLOBSTAT_CHANGE_DURING_QUERY, 5217
ER_GNO_EXHAUSTED, 5149
ER_GRANT_WRONG_HOST_OR_USER, 5109
ER_GROUPING_ON_TIMESTAMP_IN_DST, 5217
ER_GROUP_REPLICATION_APPLIER_INIT_ERROR, 5164
ER_GROUP_REPLICATION_COMMAND_FAILURE, 5189
ER_GROUP_REPLICATION_COMMUNICATION_LAYER_JOIN_ERROR, 5165
ER_GROUP_REPLICATION_COMMUNICATION_LAYER_SESSION_ERROR, 5165
ER_GROUP_REPLICATION_CONFIGURATION, 5164
ER_GROUP_REPLICATION_MAX_GROUP_SIZE, 5172
ER_GROUP_REPLICATION_PLUGIN_NOT_INSTALLED, 5261
ER_GROUP_REPLICATION_RUNNING, 5164
ER_GROUP_REPLICATION_STOP_APPLIER_THREAD_TIMEOUT, 5164
ER_GRP_DELAYED_VCLE_LOGGING, 5592
ER_GRP_RPL_ABORTS_AS_SSL_NOT_SUPPORTED_BY_MYSQLD, 5411
ER_GRP_RPL_ADD_GRPSID_TO_GRPGTIDSID_MAP_ERROR, 5385
ER_GRP_RPL_ADD_GTID_INFO_WITHOUT_LOCAL_GTID_FAILED, 5397
ER_GRP_RPL_ADD_GTID_INFO_WITHOUT_REMOTE_GTID_FAILED, 5397
ER_GRP_RPL_ADD_GTID_INFO_WITH_LOCAL_GTID_FAILED, 5397
ER_GRP_RPL_ADD_GTID_INFO_WITH_REMOTE_GTID_FAILED, 5397
ER_GRP_RPL_ADD_GTID_TO_GRPGTID_EXECUTED_ERROR, 5386
ER_GRP_RPL_ADD_RETRIEVED_SET_TO_GRP_GTID_EXECUTED_ERROR, 5386
ER_GRP_RPL_ALL_DONORS_LEFT_ABORT_RECOVERY, 5399
ER_GRP_RPL_ALL_OBSERVERS_UNREGISTERED, 5410
ER_GRP_RPL_APPENDING_DATA_TO_INTERNAL_CACHE_FAILED, 5404
ER_GRP_RPL_APPLIER_EXECUTION_FATAL_ERROR, 5385
ER_GRP_RPL_APPLIER_HANDLER_IS_IN_USE, 5415
ER_GRP_RPL_APPLIER_HANDLER_NOT_INITIALIZED, 5414
ER_GRP_RPL_APPLIER_HANDLER_ROLE_IS_IN_USE, 5415
ER_GRP_RPL_APPLIER_INITIALIZED, 5411
ER_GRP_RPL_APPLIER_NOT_STARTED_DUE_TO_RUNNING_PREV_SHUTDOWN, 5411
ER_GRP_RPL_APPLIER_PIPELINE_NOT_DISPOSED, 5384
ER_GRP_RPL_APPLIER_TERMINATION_TIMED_OUT_ON_SHUTDOWN, 5409
ER_GRP_RPL_APPLIER_THD_EXECUTION_ABORTED, 5385
ER_GRP_RPL_APPLIER_THD_KILLED, 5384
ER_GRP_RPL_APPLIER_THD_SETUP_ERROR, 5394
ER_GRP_RPL_APPLIER_THD_START_ERROR, 5394
ER_GRP_RPL_APPLIER_THD_STOP_ERROR, 5395
ER_GRP_RPL_APPOINTED_PRIMARY_NOT_PRESENT, 5571
ER_GRP_RPL_AUTO_INC_OFFSET_RESET, 5396
ER_GRP_RPL_AUTO_INC_OFFSET_SET, 5396
ER_GRP_RPL_AUTO_INC_RESET, 5396
ER_GRP_RPL_AUTO_INC_SET, 5396
ER_GRP_RPL_BINLOG_CHECKSUM_SET, 5402
ER_GRP_RPL_BINLOG_DISABLED, 5412
ER_GRP_RPL_BLOCK_SIZE_DIFF_FROM_GRP, 5394
ER_GRP_RPL_BROADCASTING_TRANS_TO_GRP_FAILED, 5404
ER_GRP_RPL_BROADCAST_COMMIT_MSSG_TOO_BIG, 5418
ER_GRP_RPL_BROADCAST_COMMIT_TRANS_MSSG_FAILED, 5385
ER_GRP_RPL_CANNOT_EXECUTE_TRANS_IN_ERROR_STATE, 5402
ER_GRP_RPL_CANNOT_EXECUTE_TRANS_IN_OFFLINE_MODE, 5402
ER_GRP_RPL_CANNOT_EXECUTE_TRANS_WHILE_RECOVERING, 5402
ER_GRP_RPL_CANNOT_EXECUTE_TRANS_WHILE_STOPPING, 5402
ER_GRP_RPL_CANT_GENERATE_GTID, 5387

ER_GRP_RPL_CANT_KILL_THREAD, 5599
ER_GRP_RPL_CANT_READ_GRP_GTID_EXTRACTED, 5388
ER_GRP_RPL_CANT_READ_GTID, 5388
ER_GRP_RPL_CANT_READ_WRITE_SET_ITEM, 5388
ER_GRP_RPL_CERTIFICATE_SIZE_ERROR, 5384
ER_GRP_RPL_CERTIFICATION_INITIALIZATION_FAILURE, 5387
ER_GRP_RPL_CERTIFICATION_REC_PROCESS, 5405
ER_GRP_RPL_CERTIFIER_MSSG_PROCESS_ERROR, 5389
ER_GRP_RPL_CHANGE_GRP_MEM_NOT_PROCESSED, 5390
ER_GRP_RPL_CHANNEL_THREAD_WHEN_GROUP_ACTION_RUNNING, 5571
ER_GRP_RPL_CHECK_STATUS_TABLE, 5400
ER_GRP_RPL_CLONE_PROCESS_EXEC_ERROR, 5598
ER_GRP_RPL_CLONE_PROCESS_PREPARE_ERROR, 5598
ER_GRP_RPL_COMMUNICATION_SSL_CONF_INFO, 5411
ER_GRP_RPL_CONFIGURATION_ACTION_END, 5572
ER_GRP_RPL_CONFIGURATION_ACTION_ERROR, 5571
ER_GRP_RPL_CONFIGURATION_ACTION_KILLED_ERROR, 5572
ER_GRP_RPL_CONFIGURATION_ACTION_LOCAL_TERMINATION, 5571
ER_GRP_RPL_CONFIGURATION_ACTION_START, 5571
ER_GRP_RPL_CONFIG_RECOVERY, 5400
ER_GRP_RPL_CONFLICT_DETECTION_DISABLED, 5389
ER_GRP_RPL_CONN_INTERNAL_PLUGIN_FAIL, 5398
ER_GRP_RPL_CONTACT_WITH_SRV_FAILED, 5398
ER_GRP_RPL_COPY_FROM_EMPTY_STRING, 5399
ER_GRP_RPL_CREATE_APPLIER_CACHE_ERROR, 5384
ER_GRP_RPL_CREATE_GRP_RPL_REC_CHANNEL, 5400
ER_GRP_RPL_CREATE_SESSION_UNABLE, 5399
ER_GRP_RPL_DATA_NOT_PROVIDED_BY_MEM, 5392
ER_GRP_RPL_DEBUG_OPTIONS, 5417
ER_GRP_RPL_DEFAULT_TABLE_ENCRYPTION_DIFF_FROM_GRP, 5589
ER_GRP_RPL_DISABLE_READ_ONLY_FAILED, 5391
ER_GRP_RPL_DISABLE_SRV_READ_MODE_RESTRICTED, 5389
ER_GRP_RPL_DONOR_CONN_TERMINATION, 5400
ER_GRP_RPL_DONOR_SERVER_CONN, 5400
ER_GRP_RPL_DONOR_TRANS_INFO_ERROR, 5386
ER_GRP_RPL_ENABLE_READ_ONLY_FAILED, 5391
ER_GRP_RPL_ERROR_FETCHING_GTID_EXECUTED_SET, 5386
ER_GRP_RPL_ERROR_FETCHING_GTID_SET, 5386
ER_GRP_RPL_ERROR_GTID_EXECUTION_INFO, 5384
ER_GRP_RPL_ERROR_GTID_SET_EXTRACTION, 5570
ER_GRP_RPL_ERROR_MSG, 5362
ER_GRP_RPL_ERROR_ON_CERT_DB_INSTALL, 5584
ER_GRP_RPL_ERROR_ON_MESSAGE_SENDING, 5571
ER_GRP_RPL_ERROR_SENDING_SINGLE_PRIMARY_MSSG, 5385
ER_GRP_RPL_ERROR_STOPPING_CHANNELS, 5385
ER_GRP_RPL_ERROR_VERIFYING_SIDNO, 5387
ER_GRP_RPL_ERROR_WHILE_WAITING_FOR_CONFLICT_DETECTION, 5404
ER_GRP_RPL_ESTABLISHING_CONN_GRP_REC_DONOR, 5400
ER_GRP_RPL_ESTABLISH_RECOVERY_WITH_ANOTHER_DONOR, 5399
ER_GRP_RPL_ESTABLISH_RECOVERY_WITH_DONOR, 5399
ER_GRP_RPL_EVENT_HANDLING_ERROR, 5384
ER_GRP_RPL_EXCEEDS_AUTO_INC_VALUE, 5392
ER_GRP_RPL_EXIT_GRP_GCS_ERROR, 5417
ER_GRP_RPL_FAILED_TO_BOOTSTRAP_EVENT_HANDLING_INFRASTRUCTURE, 5414

ER_GRP_RPL_FAILED_TO_BROADCAST_GRP_MEMBERSHIP_NOTIFICATION, 5401
ER_GRP_RPL_FAILED_TO_BROADCAST_MEMBER_STATUS_NOTIFICATION, 5401
ER_GRP_RPL_FAILED_TO_CALL_GRP_COMMUNICATION_INTERFACE, 5407
ER_GRP_RPL_FAILED_TO_CONFIRM_IF_SERVER_LEFT_GRP, 5408
ER_GRP_RPL_FAILED_TO_CREATE_COMMIT_CACHE, 5405
ER_GRP_RPL_FAILED_TO_CREATE_TRANS_CONTEXT, 5403
ER_GRP_RPL_FAILED_TO_ENABLE_READ_ONLY_MODE_ON_SHUTDOWN, 5409
ER_GRP_RPL_FAILED_TO_ENABLE_SUPER_READ_ONLY_MODE, 5407
ER_GRP_RPL_FAILED_TO_EXTRACT_TRANS_WRITE_SET, 5403
ER_GRP_RPL_FAILED_TO_GATHER_TRANS_WRITE_SET, 5403
ER_GRP_RPL_FAILED_TO_GENERATE_SIDNO_FOR_GRP, 5411
ER_GRP_RPL_FAILED_TO_INIT_APPLIER_HANDLER, 5415
ER_GRP_RPL_FAILED_TO_INIT_APPLIER_MODULE, 5411
ER_GRP_RPL_FAILED_TO_INIT_COMMUNICATION_ENGINE, 5407
ER_GRP_RPL_FAILED_TO_INIT_HANDLER, 5409
ER_GRP_RPL_FAILED_TO_INSERT_TRX_ON_TCM_ON_AFTER_CERTIFICATION, 5582
ER_GRP_RPL_FAILED_TO_NOTIFY_GRP_MEMBERSHIP_EVENT, 5401
ER_GRP_RPL_FAILED_TO_PARSE_THE_GRP_NAME, 5410
ER_GRP_RPL_FAILED_TO_REGISTER_BINLOG_STATE_OBSERVER, 5410
ER_GRP_RPL_FAILED_TO_REGISTER_SERVER_STATE_OBSERVER, 5409
ER_GRP_RPL_FAILED_TO_REGISTER_TRANS_OUTCOME_NOTIFICTION, 5404
ER_GRP_RPL_FAILED_TO_REGISTER_TRANS_STATE_OBSERVER, 5410
ER_GRP_RPL_FAILED_TO_REINIT_BINLOG_CACHE_FOR_READ, 5403
ER_GRP_RPL_FAILED_TO_SHUTDOWN_REGISTRY_MODULE, 5409
ER_GRP_RPL_FAILED_TO_START_COMMUNICATION_ENGINE, 5407
ER_GRP_RPL_FAILED_TO_START_ON_BOOT, 5410
ER_GRP_RPL_FAILED_TO_START_ON_SECONDARY_WITH_ASYNC_CHANNELS, 5407
ER_GRP_RPL_FAILED_TO_START_WITH_INVALID_SERVER_ID, 5406
ER_GRP_RPL_FAILED_TO_STOP_ON_PLUGIN_UNINSTALL, 5410
ER_GRP_RPL_FAILED_TO_UNREGISTER_BINLOG_STATE_OBSERVER, 5410
ER_GRP_RPL_FAILED_TO_UNREGISTER_SERVER_STATE_OBSERVER, 5410
ER_GRP_RPL_FAILED_TO_UNREGISTER_TRANS_STATE_OBSERVER, 5410
ER_GRP_RPL_FATAL_REC_PROCESS, 5405
ER_GRP_RPL_FETCH_FORMAT_DESC_LOG_EVENT_FAILED, 5396
ER_GRP_RPL_FETCH_GTID_LOG_EVENT_FAILED, 5397
ER_GRP_RPL_FETCH_LOG_EVENT_FAILED, 5398
ER_GRP_RPL_FETCH_SNAPSHOT_VERSION_FAILED, 5396
ER_GRP_RPL_FETCH_TRANS_CONTEXT_FAILED, 5396
ER_GRP_RPL_FETCH_TRANS_CONTEXT_LOG_EVENT_FAILED, 5396
ER_GRP_RPL_FETCH_TRANS_DATA_FAILED, 5395
ER_GRP_RPL_FETCH_TRANS_SIDNO_ERROR, 5387
ER_GRP_RPL_FETCH_VIEW_CHANGE_LOG_EVENT_FAILED, 5397
ER_GRP_RPL_FINISHED_AUTO_REJOIN, 5589
ER_GRP_RPL_FK_WITH_CASCADE_UNSUPPORTED, 5396
ER_GRP_RPL_FLOW_CONTROL_STATS, 5418
ER_GRP_RPL_FLOW_CTRL_MAX_QUOTA_SMALLER_THAN_MIN_QUOTAS, 5413
ER_GRP_RPL_FLOW_CTRL_MIN_QUOTA_GREATER_THAN_MAX_QUOTA, 5413
ER_GRP_RPL_FLOW_CTRL_MIN_RECOVERY_QUOTA_GREATER_THAN_MAX_QUOTA, 5413
ER_GRP_RPL_FORCE_MEMBERS_MUST_BE_EMPTY, 5407
ER_GRP_RPL_FORCE_MEMBERS_SET_UPDATE_NOT_ALLOWED, 5414
ER_GRP_RPL_FORCE_MEMBERS_WHEN_LEAVING, 5584
ER_GRP_RPL_FORCE_MEMBER_VALUE_SET, 5418
ER_GRP_RPL_FORCE_MEMBER_VALUE_SET_ERROR, 5417
ER_GRP_RPL_FORCE_MEMBER_VALUE_TIME_OUT, 5418

ER_GRP_RPL_GCS_GR_ERROR_MSG, 5419
ER_GRP_RPL_GCS_INTERFACE_ERROR, 5417
ER_GRP_RPL_GMS_LISTENER_FAILED_TO_LOG_NOTIFICATION, 5417
ER_GRP_RPL_GROUP_NAME_PARSE_ERROR, 5385
ER_GRP_RPL_GRP_CHANGE_INFO_EXTRACT_ERROR, 5392
ER_GRP_RPL_GRP_COMMUNICATION_ENG_INIT_FAILED, 5417
ER_GRP_RPL_GRP_COMMUNICATION_INIT_WITH_CONF, 5414
ER_GRP_RPL_GRP_MEMBER_OFFLINE, 5417
ER_GRP_RPL_GRP_NAME_IS_NOT_VALID_UUID, 5413
ER_GRP_RPL_GRP_NAME_IS_TOO_LONG, 5413
ER_GRP_RPL_GRP_NAME_OPTION_MANDATORY, 5413
ER_GRP_RPL_GTID_ALREADY_USED, 5384
ER_GRP_RPL_GTID_EXECUTED_EXTRACT_ERROR, 5392
ER_GRP_RPL_GTID_MODE_OFF, 5412
ER_GRP_RPL_GTID_PURGED_EXTRACT_ERROR, 5598
ER_GRP_RPL_GTID_SET_EXTRACT_ERROR, 5393
ER_GRP_RPL_GTID_SET_EXTRACT_ERROR_DURING_RECOVERY, 5584
ER_GRP_RPL_INCORRECT_TYPE_SET_FOR_PARALLEL_APPLIER, 5412
ER_GRP_RPL_INIT_CERTIFICATION_INFO_FAILURE, 5388
ER_GRP_RPL_INTERNAL_QUERY, 5399
ER_GRP_RPL_INVALID_BINLOG_FORMAT, 5402
ER_GRP_RPL_INVALID_COMMUNICATION_PROTOCOL, 5589
ER_GRP_RPL_INVALID_DEBUG_OPTIONS, 5417
ER_GRP_RPL_INVALID_GTID_SET, 5387
ER_GRP_RPL_INVALID_SSL_RECOVERY_STRING, 5414
ER_GRP_RPL_INVALID_TRANS_WRITE_SET_EXTRACTION_VALUE, 5412
ER_GRP_RPL_IS_STOPPED, 5409
ER_GRP_RPL_IS_STOPPING, 5409
ER_GRP_RPL_JOINER_EXIT_WHEN_GROUP_ACTION_RUNNING, 5571
ER_GRP_RPL_JOIN_WHEN_GROUP_ACTION_RUNNING, 5571
ER_GRP_RPL_KILLED_FAILED_ID, 5398
ER_GRP_RPL_KILLED_SESSION_ID, 5398
ER_GRP_RPL_LOCAL_GTID_SETS_PROCESS_ERROR, 5393
ER_GRP_RPL_LOG_SLAVE_UPDATES_NOT_SET, 5412
ER_GRP_RPL_LOWER_CASE_TABLE_NAMES_DIFF_FROM_GRP, 5560
ER_GRP_RPL_MASTER_INFO_REPO_MUST_BE_TABLE, 5412
ER_GRP_RPL_MAXIMUM_CONNECTION_RETRIES_REACHED, 5399
ER_GRP_RPL_MEMBER_ADDED, 5390
ER_GRP_RPL_MEMBER_ALREADY_EXISTS, 5392
ER_GRP_RPL_MEMBER_CFG_INCOMPATIBLE_WITH_GRP_CFG, 5394
ER_GRP_RPL_MEMBER_CHANGE, 5391
ER_GRP_RPL_MEMBER_CONF_INFO, 5408
ER_GRP_RPL_MEMBER_CONTACT_RESTORED, 5390
ER_GRP_RPL_MEMBER_EXIT_PLUGIN_ERROR, 5391
ER_GRP_RPL_MEMBER_EXPELLED, 5391
ER_GRP_RPL_MEMBER_LEFT_GRP, 5391
ER_GRP_RPL_MEMBER_NOT_FOUND, 5399
ER_GRP_RPL_MEMBER_REMOVED, 5390
ER_GRP_RPL_MEMBER_SERVER_UUID_IS_INCOMPATIBLE_WITH_GRP, 5408
ER_GRP_RPL_MEMBER_STATS_INFO, 5418
ER_GRP_RPL_MEMBER_STOP_RPL_CHANNELS_ERROR, 5394
ER_GRP_RPL_MEMBER_TRANS_GREATER_THAN_GRP, 5393
ER_GRP_RPL_MEMBER_VERSION_LOWER_THAN_GRP, 5393
ER_GRP_RPL_MEMBER_VER_INCOMPATIBLE, 5393

ER_GRP_RPL_MEMBER_VER_READ_COMPATIBLE, 5595
ER_GRP_RPL_MEM_ONLINE, 5389
ER_GRP_RPL_MEM_REACHABLE, 5390
ER_GRP_RPL_MEM_UNREACHABLE, 5389
ER_GRP_RPL_MESSAGE_SERVICE_FATAL_ERROR, 5601
ER_GRP_RPL_MESSAGE_SERVICE_INIT_FAILURE, 5217
ER_GRP_RPL_MISSING_GRP_RPL_ACTION_COORDINATOR, 5570
ER_GRP_RPL_MISSING_GRP_RPL_APPLIER, 5389
ER_GRP_RPL_MODULE_TERMINATE_ERROR, 5413
ER_GRP_RPL_MSG_DISCARDED, 5389
ER_GRP_RPL_MSG_TOO_LONG_BROADCASTING_TRANS_FAILED, 5404
ER_GRP_RPL_MULTIPLE_CACHE_TYPE_NOT_SUPPORTED_FOR_SESSION, 5403
ER_GRP_RPL_NEEDS_INNODB_TABLE, 5395
ER_GRP_RPL_NEW_PRIMARY_ELECTED, 5391
ER_GRP_RPL_NOTIFY_CERTIFICATION_OUTCOME_FAILED, 5397
ER_GRP_RPL_NO_POSSIBLE_RECOVERY, 5598
ER_GRP_RPL_NO_STAGE_SERVICE, 5572
ER_GRP_RPL_NO_SUITABLE_PRIMARY_MEM, 5392
ER_GRP_RPL_NO_VALID_DONOR, 5400
ER_GRP_RPL_NULL_PACKET, 5388
ER_GRP_RPL_ONLY_ONE_SERVER_ALIVE, 5405
ER_GRP_RPL_OOM_FAILED_TO_GENERATE_IDENTIFICATION_HASH, 5401
ER_GRP_RPL_PIPELINE_CREATE_FAILED, 5418
ER_GRP_RPL_PIPELINE_FLUSH_FAIL, 5419
ER_GRP_RPL_PIPELINE_REINIT_FAILED_READ, 5419
ER_GRP_RPL_PIPELINE_REINIT_FAILED_WRITE, 5419
ER_GRP_RPL_PLUGIN_ABORT, 5567
ER_GRP_RPL_PLUGIN_STRUCT_INIT_NOT_POSSIBLE_ON_SERVER_START, 5407
ER_GRP_RPL_PREV_REC_SESSION_RUNNING, 5405
ER_GRP_RPL_PRIMARY_ELECTION_PROCESS_ERROR, 5572
ER_GRP_RPL_PRIMARY_ELECTION_STOP_ERROR, 5572
ER_GRP_RPL_PRIMARY_KEY_NOT_DEFINED, 5395
ER_GRP_RPL_PRIMARY_MEMBER_LEFT_GRP, 5390
ER_GRP_RPL_PROCESS_GTID_SET_ERROR, 5388
ER_GRP_RPL_PROCESS_INTERSECTION_GTID_SET_ERROR, 5388
ER_GRP_RPL_PURGE_APPLIER_LOGS, 5394
ER_GRP_RPL_PURGE_REC, 5401
ER_GRP_RPL_QUERY_FAIL, 5399
ER_GRP_RPL_READ_UNABLE_FOR_READ_ONLY_SUPER_READ_ONLY, 5406
ER_GRP_RPL_READ_UNABLE_FOR_SUPER_READ_ONLY, 5406
ER_GRP_RPL_RECEIVED_SET_MISSING_GTIDS, 5387
ER_GRP_RPL_RECOVERY_EVAL_ERROR, 5598
ER_GRP_RPL_RECOVERY_MODULE_TERMINATION_TIMED_OUT_ON_SHUTDOWN, 5409
ER_GRP_RPL_RECOVERY_STRAT_CHOICE, 5599
ER_GRP_RPL_RECOVERY_STRAT_CLONE_PURGED, 5599
ER_GRP_RPL_RECOVERY_STRAT_CLONE_THRESHOLD, 5599
ER_GRP_RPL_RECOVERY_STRAT_FALLBACK, 5599
ER_GRP_RPL_RECOVERY_STRAT_NO_FALLBACK, 5599
ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_DEPENDENCIES_FAILED, 5582
ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_GROUP_PREPARE_FAILED, 5582
ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_SYNC_BEFORE_EXECUTION_FAILED, 5583
ER_GRP_RPL_REINIT_OF_COMMIT_CACHE_FOR_WRITE_FAILED, 5405
ER_GRP_RPL_REINIT_OF_INTERNAL_CACHE_FOR_READ_FAILED, 5403
ER_GRP_RPL_REINIT_OF_INTERNAL_CACHE_FOR_WRITE_FAILED, 5404

ER_GRP_RPL_RELAY_LOG_INFO_REPO_MUST_BE_TABLE, 5412
ER_GRP_RPL_RELEASE_BEGIN_TRX_AFTER_DEPENDENCIES_COMMIT_FAILED, 5582
ER_GRP_RPL_RELEASE_BEGIN_TRX_AFTER_WAIT_FOR_SYNC_BEFORE_EXEC, 5583
ER_GRP_RPL_RELEASE_COMMIT_AFTER_GROUP_PREPARE_FAILED, 5581
ER_GRP_RPL_REQUESTING_NON_MEMBER_SERVER_TO_LEAVE, 5408
ER_GRP_RPL_RESET_APPLIER_MODULE_LOGS_ERROR, 5394
ER_GRP_RPL_SALVE_IO_THD_ON_SECONDARY_MEMBER, 5395
ER_GRP_RPL_SEND_STATS_ERROR, 5418
ER_GRP_RPL_SEND_TRX_PREPARED_MESSAGE_FAILED, 5581
ER_GRP_RPL_SEND_TRX_SYNC_BEFORE_EXECUTION_FAILED, 5583
ER_GRP_RPL_SERVER_ALREADY_LEFT, 5408
ER_GRP_RPL_SERVER_CONN_ERROR, 5386
ER_GRP_RPL_SERVER_IS_ALREADY_LEAVING, 5408
ER_GRP_RPL_SERVER_SET_TO_OFFLINE_MODE_DUE_TO_ERRORS, 5601
ER_GRP_RPL_SERVER_SET_TO_READ_ONLY_DUE_TO_ERRORS, 5416
ER_GRP_RPL_SERVER_UDF_ERROR, 5573
ER_GRP_RPL_SERVER_WORKING_AS_SECONDARY, 5406
ER_GRP_RPL_SESSION_OPEN_FAILED, 5391
ER_GRP_RPL_SET_GRP_COMMUNICATION_ENG_LOGGER_FAILED, 5417
ER_GRP_RPL_SET_STABLE_TRANS_ERROR, 5388
ER_GRP_RPL_SIDNO_FETCH_ERROR, 5385
ER_GRP_RPL_SINGLE_PRIM_MODE_NOT_ALLOWED_WITH_UPDATE_EVERYWHERE, 5413
ER_GRP_RPL_SKIP_COMPUTATION_TRANS_COMMITTED, 5388
ER_GRP_RPL_SLAVE_APPLIER_THREAD_ERROR_OUT, 5420
ER_GRP_RPL_SLAVE_APPLIER_THREAD_UNBLOCKED, 5419
ER_GRP_RPL_SLAVE_IO_THD_PRIMARY_UNKNOWN, 5395
ER_GRP_RPL_SLAVE_IO_THREAD_ERROR_OUT, 5419
ER_GRP_RPL_SLAVE_IO_THREAD_UNBLOCKED, 5419
ER_GRP_RPL_SLAVE_PRESERVE_COMMIT_ORDER_NOT_SET, 5412
ER_GRP_RPL_SLAVE_SQL_THD_ON_SECONDARY_MEMBER, 5395
ER_GRP_RPL_SLAVE_SQL_THD_PRIMARY_UNKNOWN, 5395
ER_GRP_RPL_SLAVE_THREAD_ERROR_ON_CLONE, 5599
ER_GRP_RPL_SQL_SERVICE_COMM_SESSION_NOT_INITIALIZED, 5415
ER_GRP_RPL_SQL_SERVICE_FAILED_TO_FETCH_SECURITY_CTX, 5416
ER_GRP_RPL_SQL_SERVICE_FAILED_TO_INIT_SESSION_THREAD, 5415
ER_GRP_RPL_SQL_SERVICE_FAILED_TO_RUN_SQL_QUERY, 5415
ER_GRP_RPL_SQL_SERVICE_MAX_CONN_ERROR_FROM_SERVER, 5416
ER_GRP_RPL_SQL_SERVICE_RETRIES_EXCEEDED_ON_SESSION_STATE, 5416
ER_GRP_RPL_SQL_SERVICE_SERVER_ACCESS_DENIED_FOR_USER, 5416
ER_GRP_RPL_SQL_SERVICE_SERVER_ERROR_ON_CONN, 5416
ER_GRP_RPL_SQL_SERVICE_SERVER_INTERNAL_FAILURE, 5415
ER_GRP_RPL_SQL_SERVICE_SERVER_SESSION_KILLED, 5415
ER_GRP_RPL_SRV_BLOCKED, 5390
ER_GRP_RPL_SRV_BLOCKED_FOR_SECS, 5390
ER_GRP_RPL_SRV_GTID_WAIT_ERROR, 5592
ER_GRP_RPL_SRV_NOT_ONLINE, 5389
ER_GRP_RPL_SRV_ONLINE, 5389
ER_GRP_RPL_SRV_PRIMARY_MEM, 5391
ER_GRP_RPL_SRV_SECONDARY_MEM, 5392
ER_GRP_RPL_SRV_WAIT_TIME_OUT, 5398
ER_GRP_RPL_SSL_DISABLED, 5411
ER_GRP_RPL_STARTED_AUTO_REJOIN, 5589
ER_GRP_RPL_STARTING_GRP_REC, 5400
ER_GRP_RPL_START_FAILED, 5393

ER_GRP_RPL_START_GRP_RPL_FAILED, 5398
ER_GRP_RPL_STOPPING_GRP_REC, 5400
ER_GRP_RPL_STOP_REP_CHANNEL, 5419
ER_GRP_RPL_SUPER_READ_OFF, 5398
ER_GRP_RPL_SUPER_READ_ON, 5398
ER_GRP_RPL_SUPER_READ_ONLY_ACTIVATE_ERROR, 5392
ER_GRP_RPL_SUPPORTS_ONLY_ONE_FORCE_MEMBERS_SET, 5414
ER_GRP_RPL_TIMEOUT_ON_VIEW_AFTER_JOINING_GRP, 5407
ER_GRP_RPL_TIMEOUT_RECEIVED_VC_ON_REJOIN, 5589
ER_GRP_RPL_TIMEOUT_RECEIVING_VIEW_CHANGE_ON_SHUTDOWN, 5408
ER_GRP_RPL_TRANS_GREATER_THAN_GRP, 5393
ER_GRP_RPL_TRANS_NOT_PRESENT_IN_GRP, 5393
ER_GRP_RPL_TRANS_SIZE_EXCEEDS_LIMIT, 5403
ER_GRP_RPL_TRANS_WRITE_SET_EXTRACTION_NOT_SET, 5402
ER_GRP_RPL_TRANS_WRITE_SET_EXTRACT_DIFF_FROM_GRP, 5394
ER_GRP_RPL_TRX_ALREADY_EXISTS_ON_TCM_ON_AFTER_CERTIFICATION, 5582
ER_GRP_RPL_TRX_DOES_NOT_EXIST_ON_TCM_ON_HANDLE_REMOTE_PREPARE, 5582
ER_GRP_RPL_TRX_WAIT_FOR_GROUP_GTID_EXECUTED, 5583
ER_GRP_RPL_TRX_WAIT_FOR_GROUP_PREPARE_FAILED, 5582
ER_GRP_RPL_TRX_WAIT_FOR_SYNC_BEFORE_EXECUTION_FAILED, 5583
ER_GRP_RPL_UDF_ERROR, 5217
ER_GRP_RPL_UDF_REGISTER_ERROR, 5572
ER_GRP_RPL_UDF_REGISTER_SERVICE_ERROR, 5572
ER_GRP_RPL_UDF_UNREGISTER_ERROR, 5572
ER_GRP_RPL_UNABLE_TO_CERTIFY_PLUGIN_TRANS, 5406
ER_GRP_RPL_UNABLE_TO_CONVERT_EVENT_TO_PACKET, 5419
ER_GRP_RPL_UNABLE_TO_CONVERT_PACKET_TO_EVENT, 5418
ER_GRP_RPL_UNABLE_TO_ENSURE_EXECUTION_REC, 5406
ER_GRP_RPL_UNABLE_TO_EVALUATE_APPLIER_STATUS, 5405
ER_GRP_RPL_UNABLE_TO_INIT_COMMUNICATION_ENGINE, 5411
ER_GRP_RPL_UNABLE_TO_KILL_CONN_REC_DONOR_APPLIER, 5401
ER_GRP_RPL_UNABLE_TO_KILL_CONN_REC_DONOR_FAILOVER, 5401
ER_GRP_RPL_UNABLE_TO_RESET_SERVER_READ_MODE, 5406
ER_GRP_RPL_UNBLOCK_CERTIFIED_TRANS, 5406
ER_GRP_RPL_UNBLOCK_WAITING_THD, 5384
ER_GRP_RPL_UNKNOWN_GRP_RPL_APPLIER_PIPELINE_REQUESTED, 5414
ER_GRP_RPL_UNREACHABLE_MAJORITY_TIMEOUT_FOR_MEMBER, 5416
ER_GRP_RPL_UNSUPPORTED_TRANS_ISOLATION, 5402
ER_GRP_RPL_UPDATE_GRPGTID_EXECUTED_ERROR, 5386
ER_GRP_RPL_UPDATE_GTID_SET_ERROR, 5387
ER_GRP_RPL_UPDATE_LAST_CONFLICT_FREE_TRANS_ERROR, 5387
ER_GRP_RPL_UPDATE_SERV_CERTIFICATE_FAILED, 5397
ER_GRP_RPL_UPDATE_TRANS_SNAPSHOT_REF_VER_ERROR, 5387
ER_GRP_RPL_UPDATE_TRANS_SNAPSHOT_VER_ERROR, 5385
ER_GRP_RPL_WAITING_FOR_VIEW_UPDATE, 5408
ER_GRP_RPL_WAIT_FOR_DEPENDENCIES_FAILED, 5583
ER_GRP_RPL_WHILE_SENDING_MSG_REC, 5406
ER_GRP_RPL_WHILE_STOPPING_REP_CHANNEL, 5405
ER_GRP_RPL_WRITE_IDENT_HASH_BASE64_ENCODING_FAILED, 5401
ER_GRP_RPL_WRITE_TO_BINLOG_CACHE_FAILED, 5404
ER_GRP_RPL_WRITE_TO_TRANSACTION_MESSAGE_FAILED, 5404
ER_GRP_TRX_CONSISTENCY_AFTER_ON_TRX_BEGIN, 5204
ER_GRP_TRX_CONSISTENCY_BEFORE, 5204
ER_GRP_TRX_CONSISTENCY_BEGIN_NOT_ALLOWED, 5204

ER_GRP_TRX_CONSISTENCY_NOT_ALLOWED, 5204
ER_GR_HOLD_KILLED, 5202
ER_GR_HOLD_MEMBER_STATUS_ERROR, 5202
ER_GR_HOLD_WAIT_TIMEOUT, 5202
ER_GTID_ALREADY_ADDED_BY_USER, 5261
ER_GTID_EXECUTED_WAS_CHANGED, 5154
ER_GTID_EXECUTED_WAS_UPDATED, 5324
ER_GTID_MODE_CAN_ONLY_CHANGE_ONE_STEP_AT_A_TIME, 5150
ER_GTID_MODE_OFF, 5162
ER_GTID_MODE_ON_REQUIRES_ENFORCE_GTID_CONSISTENCY_ON, 5149
ER_GTID_NEXT_CANT_BE_AUTOMATIC_IF_GTID_NEXT_LIST_IS_NON_NULL, 5148
ER_GTID_NEXT_TYPE_UNDEFINED_GROUP, 5153
ER_GTID_NEXT_TYPE_UNDEFINED_GTID, 5153
ER_GTID_PURGED_WAS_CHANGED, 5154
ER_GTID_PURGED_WAS_UPDATED, 5323
ER_GTID_UNSAFE_ALTER_ADD_COL_WITH_DEFAULT_EXPRESSION, 5201
ER_GTID_UNSAFE_BINLOG_SPLITTABLE_STATEMENT_AND_ASSIGNED_GTID, 5157
ER_GTID_UNSAFE_BINLOG_SPLITTABLE_STATEMENT_AND_GTID_GROUP, 5157
ER_GTID_UNSAFE_CREATE_DROP_TEMPORARY_TABLE_IN_TRANSACTION, 5150
ER_GTID_UNSAFE_CREATE_SELECT, 5150
ER_GTID_UNSAFE_NON_TRANSACTIONAL_TABLE, 5149
ER_HANDLERTON_OOM, 5245
ER_HANDSHAKE_ERROR, 5101
ER_HOSTNAME, 5129
ER_HOSTNAME_DOESNT_RESOLVE_TO, 5226
ER_HOSTNAME_RESEMBLES_IPV4, 5226
ER_HOST_IS_BLOCKED, 5107
ER_HOST_NOT_PRIVILEGED, 5108
ER_IB_CLONE_GTID_PERSIST, 5446
ER_IB_CLONE_INTERNAL, 5444
ER_IB_CLONE_NON_INNODB_TABLE, 5598
ER_IB_CLONE_OPERATION, 5597
ER_IB_CLONE_PUNCH_HOLE, 5445
ER_IB_CLONE_RESTART, 5598
ER_IB_CLONE_SQL, 5445
ER_IB_CLONE_START_STOP, 5597
ER_IB_CLONE_STATUS_FILE, 5445
ER_IB_CLONE_TIMEOUT, 5445
ER_IB_CLONE_USER_DATA, 5598
ER_IB_CLONE_VALIDATE, 5445
ER_IB_LOB_ROLLBACK_INDEX_LEN, 5603
ER_IB_MSG_0, 5429
ER_IB_MSG_1, 5430
ER_IB_MSG_10, 5430
ER_IB_MSG_100, 5439
ER_IB_MSG_1000, 5531
ER_IB_MSG_1001, 5531
ER_IB_MSG_1002, 5531
ER_IB_MSG_1003, 5531
ER_IB_MSG_1004, 5531
ER_IB_MSG_1005, 5532
ER_IB_MSG_1006, 5532
ER_IB_MSG_1007, 5532
ER_IB_MSG_1008, 5532

ER_IB_MSG_1009, 5532
ER_IB_MSG_101, 5440
ER_IB_MSG_1010, 5532
ER_IB_MSG_1011, 5532
ER_IB_MSG_1012, 5532
ER_IB_MSG_1013, 5532
ER_IB_MSG_1014, 5532
ER_IB_MSG_1015, 5533
ER_IB_MSG_1016, 5533
ER_IB_MSG_1017, 5533
ER_IB_MSG_1018, 5533
ER_IB_MSG_1019, 5533
ER_IB_MSG_102, 5440
ER_IB_MSG_1020, 5533
ER_IB_MSG_1021, 5533
ER_IB_MSG_1022, 5533
ER_IB_MSG_1023, 5533
ER_IB_MSG_1024, 5533
ER_IB_MSG_1025, 5534
ER_IB_MSG_1026, 5534
ER_IB_MSG_1027, 5534
ER_IB_MSG_1028, 5534
ER_IB_MSG_1029, 5534
ER_IB_MSG_103, 5440
ER_IB_MSG_1030, 5534
ER_IB_MSG_1031, 5534
ER_IB_MSG_1032, 5534
ER_IB_MSG_1033, 5534
ER_IB_MSG_1034, 5534
ER_IB_MSG_1035, 5535
ER_IB_MSG_1036, 5535
ER_IB_MSG_1037, 5535
ER_IB_MSG_1038, 5535
ER_IB_MSG_1039, 5535
ER_IB_MSG_104, 5440
ER_IB_MSG_1040, 5535
ER_IB_MSG_1041, 5535
ER_IB_MSG_1042, 5535
ER_IB_MSG_1043, 5535
ER_IB_MSG_1044, 5535
ER_IB_MSG_1045, 5536
ER_IB_MSG_1046, 5536
ER_IB_MSG_1047, 5536
ER_IB_MSG_1048, 5536
ER_IB_MSG_1049, 5536
ER_IB_MSG_105, 5440
ER_IB_MSG_1050, 5536
ER_IB_MSG_1051, 5536
ER_IB_MSG_1052, 5536
ER_IB_MSG_1053, 5536
ER_IB_MSG_1054, 5537
ER_IB_MSG_1055, 5537
ER_IB_MSG_1056, 5537
ER_IB_MSG_1057, 5537

ER_IB_MSG_1058, 5537
ER_IB_MSG_1059, 5537
ER_IB_MSG_106, 5440
ER_IB_MSG_1060, 5537
ER_IB_MSG_1061, 5537
ER_IB_MSG_1062, 5537
ER_IB_MSG_1063, 5537
ER_IB_MSG_1064, 5538
ER_IB_MSG_1065, 5538
ER_IB_MSG_1066, 5538
ER_IB_MSG_1067, 5538
ER_IB_MSG_1068, 5538
ER_IB_MSG_1069, 5538
ER_IB_MSG_107, 5440
ER_IB_MSG_1070, 5538
ER_IB_MSG_1071, 5538
ER_IB_MSG_1072, 5538
ER_IB_MSG_1073, 5538
ER_IB_MSG_1074, 5539
ER_IB_MSG_1075, 5539
ER_IB_MSG_1076, 5539
ER_IB_MSG_1077, 5539
ER_IB_MSG_1078, 5539
ER_IB_MSG_1079, 5539
ER_IB_MSG_108, 5440
ER_IB_MSG_1080, 5539
ER_IB_MSG_1081, 5539
ER_IB_MSG_1082, 5539
ER_IB_MSG_1083, 5539
ER_IB_MSG_1084, 5540
ER_IB_MSG_1085, 5540
ER_IB_MSG_1086, 5540
ER_IB_MSG_1087, 5540
ER_IB_MSG_1088, 5540
ER_IB_MSG_1089, 5540
ER_IB_MSG_109, 5440
ER_IB_MSG_1090, 5540
ER_IB_MSG_1091, 5540
ER_IB_MSG_1092, 5540
ER_IB_MSG_1093, 5540
ER_IB_MSG_1094, 5541
ER_IB_MSG_1095, 5541
ER_IB_MSG_1096, 5541
ER_IB_MSG_1097, 5541
ER_IB_MSG_1098, 5541
ER_IB_MSG_1099, 5541
ER_IB_MSG_11, 5431
ER_IB_MSG_110, 5440
ER_IB_MSG_1100, 5541
ER_IB_MSG_1101, 5541
ER_IB_MSG_1102, 5541
ER_IB_MSG_1103, 5541
ER_IB_MSG_1104, 5542
ER_IB_MSG_1105, 5542

ER_IB_MSG_1106, 5542
ER_IB_MSG_1107, 5542
ER_IB_MSG_1108, 5542
ER_IB_MSG_1109, 5542
ER_IB_MSG_111, 5441
ER_IB_MSG_1110, 5542
ER_IB_MSG_1111, 5542
ER_IB_MSG_1112, 5542
ER_IB_MSG_1113, 5543
ER_IB_MSG_1114, 5543
ER_IB_MSG_1115, 5543
ER_IB_MSG_1116, 5543
ER_IB_MSG_1117, 5543
ER_IB_MSG_1118, 5543
ER_IB_MSG_1119, 5543
ER_IB_MSG_112, 5441
ER_IB_MSG_1120, 5543
ER_IB_MSG_1121, 5543
ER_IB_MSG_1122, 5543
ER_IB_MSG_1123, 5544
ER_IB_MSG_1124, 5544
ER_IB_MSG_1125, 5544
ER_IB_MSG_1126, 5544
ER_IB_MSG_1127, 5544
ER_IB_MSG_1128, 5544
ER_IB_MSG_1129, 5544
ER_IB_MSG_113, 5441
ER_IB_MSG_1130, 5544
ER_IB_MSG_1131, 5544
ER_IB_MSG_1132, 5544
ER_IB_MSG_1133, 5545
ER_IB_MSG_1134, 5545
ER_IB_MSG_1135, 5545
ER_IB_MSG_1136, 5545
ER_IB_MSG_1137, 5545
ER_IB_MSG_1138, 5545
ER_IB_MSG_1139, 5545
ER_IB_MSG_114, 5441
ER_IB_MSG_1140, 5545
ER_IB_MSG_1141, 5545
ER_IB_MSG_1142, 5546
ER_IB_MSG_1143, 5546
ER_IB_MSG_1144, 5546
ER_IB_MSG_1145, 5546
ER_IB_MSG_1146, 5546
ER_IB_MSG_1147, 5546
ER_IB_MSG_1148, 5546
ER_IB_MSG_1149, 5546
ER_IB_MSG_115, 5441
ER_IB_MSG_1150, 5546
ER_IB_MSG_1151, 5547
ER_IB_MSG_1152, 5547
ER_IB_MSG_1153, 5547
ER_IB_MSG_1154, 5547

ER_IB_MSG_1155, 5547
ER_IB_MSG_1156, 5547
ER_IB_MSG_1157, 5547
ER_IB_MSG_1158, 5547
ER_IB_MSG_1159, 5547
ER_IB_MSG_116, 5441
ER_IB_MSG_1160, 5548
ER_IB_MSG_1161, 5548
ER_IB_MSG_1162, 5548
ER_IB_MSG_1163, 5548
ER_IB_MSG_1164, 5548
ER_IB_MSG_1165, 5548
ER_IB_MSG_1166, 5548
ER_IB_MSG_1167, 5548
ER_IB_MSG_1168, 5548
ER_IB_MSG_1169, 5548
ER_IB_MSG_117, 5441
ER_IB_MSG_1170, 5549
ER_IB_MSG_1171, 5549
ER_IB_MSG_1172, 5549
ER_IB_MSG_1173, 5549
ER_IB_MSG_1174, 5549
ER_IB_MSG_1175, 5549
ER_IB_MSG_1176, 5549
ER_IB_MSG_1177, 5549
ER_IB_MSG_1178, 5549
ER_IB_MSG_1179, 5549
ER_IB_MSG_118, 5441
ER_IB_MSG_1180, 5550
ER_IB_MSG_1181, 5550
ER_IB_MSG_1182, 5550
ER_IB_MSG_1183, 5550
ER_IB_MSG_1184, 5550
ER_IB_MSG_1185, 5550
ER_IB_MSG_1186, 5550
ER_IB_MSG_1187, 5550
ER_IB_MSG_1188, 5550
ER_IB_MSG_1189, 5550
ER_IB_MSG_119, 5441
ER_IB_MSG_1190, 5551
ER_IB_MSG_1191, 5551
ER_IB_MSG_1192, 5551
ER_IB_MSG_1193, 5551
ER_IB_MSG_1194, 5551
ER_IB_MSG_1195, 5551
ER_IB_MSG_1196, 5551
ER_IB_MSG_1197, 5551
ER_IB_MSG_1198, 5551
ER_IB_MSG_1199, 5551
ER_IB_MSG_12, 5431
ER_IB_MSG_120, 5441
ER_IB_MSG_1200, 5552
ER_IB_MSG_1201, 5552
ER_IB_MSG_1202, 5552

ER_IB_MSG_1203, 5552
ER_IB_MSG_1204, 5552
ER_IB_MSG_1205, 5552
ER_IB_MSG_1206, 5552
ER_IB_MSG_1207, 5552
ER_IB_MSG_1208, 5552
ER_IB_MSG_1209, 5552
ER_IB_MSG_121, 5442
ER_IB_MSG_1210, 5553
ER_IB_MSG_1211, 5553
ER_IB_MSG_1212, 5553
ER_IB_MSG_1213, 5553
ER_IB_MSG_1214, 5553
ER_IB_MSG_1215, 5553
ER_IB_MSG_1216, 5553
ER_IB_MSG_1217, 5553
ER_IB_MSG_1218, 5553
ER_IB_MSG_1219, 5553
ER_IB_MSG_122, 5442
ER_IB_MSG_1220, 5554
ER_IB_MSG_1221, 5554
ER_IB_MSG_1222, 5554
ER_IB_MSG_1223, 5554
ER_IB_MSG_1224, 5554
ER_IB_MSG_1225, 5554
ER_IB_MSG_1226, 5554
ER_IB_MSG_1227, 5554
ER_IB_MSG_1228, 5554
ER_IB_MSG_1229, 5554
ER_IB_MSG_123, 5442
ER_IB_MSG_1230, 5555
ER_IB_MSG_1231, 5555
ER_IB_MSG_1232, 5555
ER_IB_MSG_1233, 5555
ER_IB_MSG_1234, 5555
ER_IB_MSG_1235, 5555
ER_IB_MSG_1236, 5555
ER_IB_MSG_1237, 5555
ER_IB_MSG_1238, 5555
ER_IB_MSG_1239, 5555
ER_IB_MSG_124, 5442
ER_IB_MSG_1240, 5556
ER_IB_MSG_1241, 5556
ER_IB_MSG_1242, 5556
ER_IB_MSG_1243, 5556
ER_IB_MSG_1244, 5556
ER_IB_MSG_1245, 5556
ER_IB_MSG_1246, 5556
ER_IB_MSG_1247, 5556
ER_IB_MSG_1248, 5556
ER_IB_MSG_1249, 5556
ER_IB_MSG_125, 5442
ER_IB_MSG_1250, 5557
ER_IB_MSG_1251, 5557

ER_IB_MSG_1252, 5557
ER_IB_MSG_1253, 5557
ER_IB_MSG_1254, 5557
ER_IB_MSG_1255, 5557
ER_IB_MSG_1256, 5557
ER_IB_MSG_1257, 5557
ER_IB_MSG_1258, 5557
ER_IB_MSG_1259, 5557
ER_IB_MSG_126, 5442
ER_IB_MSG_1260, 5558
ER_IB_MSG_1261, 5558
ER_IB_MSG_1262, 5558
ER_IB_MSG_1263, 5558
ER_IB_MSG_1264, 5558
ER_IB_MSG_1265, 5558
ER_IB_MSG_1266, 5558
ER_IB_MSG_1267, 5558
ER_IB_MSG_1268, 5558
ER_IB_MSG_1269, 5558
ER_IB_MSG_127, 5442
ER_IB_MSG_1270, 5559
ER_IB_MSG_1271, 5566
ER_IB_MSG_1272, 5567
ER_IB_MSG_1273, 5568
ER_IB_MSG_1274, 5568
ER_IB_MSG_1275, 5569
ER_IB_MSG_1276, 5569
ER_IB_MSG_1277, 5569
ER_IB_MSG_1278, 5569
ER_IB_MSG_1279, 5569
ER_IB_MSG_128, 5442
ER_IB_MSG_1280, 5569
ER_IB_MSG_1281, 5570
ER_IB_MSG_1282, 5570
ER_IB_MSG_1283, 5570
ER_IB_MSG_1284, 5570
ER_IB_MSG_1285, 5594
ER_IB_MSG_129, 5442
ER_IB_MSG_13, 5431
ER_IB_MSG_130, 5442
ER_IB_MSG_131, 5443
ER_IB_MSG_132, 5443
ER_IB_MSG_133, 5443
ER_IB_MSG_134, 5443
ER_IB_MSG_135, 5443
ER_IB_MSG_136, 5443
ER_IB_MSG_137, 5443
ER_IB_MSG_138, 5443
ER_IB_MSG_139, 5443
ER_IB_MSG_14, 5431
ER_IB_MSG_140, 5443
ER_IB_MSG_141, 5444
ER_IB_MSG_142, 5444
ER_IB_MSG_143, 5444

ER_IB_MSG_144, 5444
ER_IB_MSG_145, 5444
ER_IB_MSG_146, 5444
ER_IB_MSG_147, 5444
ER_IB_MSG_148, 5444
ER_IB_MSG_149, 5444
ER_IB_MSG_15, 5431
ER_IB_MSG_150, 5445
ER_IB_MSG_151, 5445
ER_IB_MSG_152, 5445
ER_IB_MSG_153, 5445
ER_IB_MSG_154, 5445
ER_IB_MSG_155, 5446
ER_IB_MSG_156, 5446
ER_IB_MSG_157, 5446
ER_IB_MSG_158, 5446
ER_IB_MSG_159, 5446
ER_IB_MSG_16, 5431
ER_IB_MSG_160, 5446
ER_IB_MSG_161, 5446
ER_IB_MSG_162, 5446
ER_IB_MSG_163, 5446
ER_IB_MSG_164, 5447
ER_IB_MSG_165, 5447
ER_IB_MSG_166, 5447
ER_IB_MSG_167, 5447
ER_IB_MSG_168, 5447
ER_IB_MSG_169, 5447
ER_IB_MSG_17, 5431
ER_IB_MSG_170, 5447
ER_IB_MSG_171, 5447
ER_IB_MSG_172, 5447
ER_IB_MSG_173, 5447
ER_IB_MSG_174, 5448
ER_IB_MSG_175, 5448
ER_IB_MSG_176, 5448
ER_IB_MSG_177, 5448
ER_IB_MSG_178, 5448
ER_IB_MSG_179, 5448
ER_IB_MSG_18, 5431
ER_IB_MSG_180, 5448
ER_IB_MSG_181, 5448
ER_IB_MSG_182, 5448
ER_IB_MSG_183, 5448
ER_IB_MSG_184, 5449
ER_IB_MSG_185, 5449
ER_IB_MSG_186, 5449
ER_IB_MSG_187, 5449
ER_IB_MSG_188, 5449
ER_IB_MSG_189, 5449
ER_IB_MSG_19, 5431
ER_IB_MSG_190, 5449
ER_IB_MSG_191, 5449
ER_IB_MSG_192, 5449

ER_IB_MSG_193, 5449
ER_IB_MSG_194, 5450
ER_IB_MSG_195, 5450
ER_IB_MSG_196, 5450
ER_IB_MSG_197, 5450
ER_IB_MSG_198, 5450
ER_IB_MSG_199, 5450
ER_IB_MSG_2, 5430
ER_IB_MSG_20, 5431
ER_IB_MSG_200, 5450
ER_IB_MSG_201, 5450
ER_IB_MSG_202, 5450
ER_IB_MSG_203, 5450
ER_IB_MSG_204, 5451
ER_IB_MSG_205, 5451
ER_IB_MSG_206, 5451
ER_IB_MSG_207, 5451
ER_IB_MSG_208, 5451
ER_IB_MSG_209, 5451
ER_IB_MSG_21, 5432
ER_IB_MSG_210, 5451
ER_IB_MSG_211, 5451
ER_IB_MSG_212, 5451
ER_IB_MSG_213, 5451
ER_IB_MSG_214, 5452
ER_IB_MSG_215, 5452
ER_IB_MSG_216, 5452
ER_IB_MSG_217, 5452
ER_IB_MSG_218, 5452
ER_IB_MSG_219, 5452
ER_IB_MSG_22, 5432
ER_IB_MSG_220, 5452
ER_IB_MSG_221, 5452
ER_IB_MSG_222, 5452
ER_IB_MSG_223, 5452
ER_IB_MSG_224, 5453
ER_IB_MSG_225, 5453
ER_IB_MSG_226, 5453
ER_IB_MSG_227, 5453
ER_IB_MSG_228, 5453
ER_IB_MSG_229, 5453
ER_IB_MSG_23, 5432
ER_IB_MSG_230, 5453
ER_IB_MSG_231, 5453
ER_IB_MSG_232, 5453
ER_IB_MSG_233, 5453
ER_IB_MSG_234, 5454
ER_IB_MSG_235, 5454
ER_IB_MSG_236, 5454
ER_IB_MSG_237, 5454
ER_IB_MSG_238, 5454
ER_IB_MSG_239, 5454
ER_IB_MSG_24, 5432
ER_IB_MSG_240, 5454

ER_IB_MSG_241, 5454
ER_IB_MSG_242, 5454
ER_IB_MSG_243, 5454
ER_IB_MSG_244, 5455
ER_IB_MSG_245, 5455
ER_IB_MSG_246, 5455
ER_IB_MSG_247, 5455
ER_IB_MSG_248, 5455
ER_IB_MSG_249, 5455
ER_IB_MSG_25, 5432
ER_IB_MSG_250, 5455
ER_IB_MSG_251, 5455
ER_IB_MSG_252, 5455
ER_IB_MSG_253, 5455
ER_IB_MSG_254, 5456
ER_IB_MSG_255, 5456
ER_IB_MSG_256, 5456
ER_IB_MSG_257, 5456
ER_IB_MSG_258, 5456
ER_IB_MSG_259, 5456
ER_IB_MSG_26, 5432
ER_IB_MSG_260, 5456
ER_IB_MSG_261, 5456
ER_IB_MSG_262, 5456
ER_IB_MSG_263, 5456
ER_IB_MSG_264, 5457
ER_IB_MSG_265, 5457
ER_IB_MSG_266, 5457
ER_IB_MSG_267, 5457
ER_IB_MSG_268, 5457
ER_IB_MSG_269, 5457
ER_IB_MSG_27, 5432
ER_IB_MSG_270, 5457
ER_IB_MSG_271, 5457
ER_IB_MSG_272, 5457
ER_IB_MSG_273, 5457
ER_IB_MSG_274, 5458
ER_IB_MSG_275, 5458
ER_IB_MSG_276, 5458
ER_IB_MSG_277, 5458
ER_IB_MSG_278, 5458
ER_IB_MSG_279, 5458
ER_IB_MSG_28, 5432
ER_IB_MSG_280, 5458
ER_IB_MSG_281, 5458
ER_IB_MSG_282, 5458
ER_IB_MSG_283, 5458
ER_IB_MSG_284, 5459
ER_IB_MSG_285, 5459
ER_IB_MSG_286, 5459
ER_IB_MSG_287, 5459
ER_IB_MSG_288, 5459
ER_IB_MSG_289, 5459
ER_IB_MSG_29, 5432

ER_IB_MSG_290, 5459
ER_IB_MSG_291, 5459
ER_IB_MSG_292, 5459
ER_IB_MSG_293, 5459
ER_IB_MSG_294, 5460
ER_IB_MSG_295, 5460
ER_IB_MSG_296, 5460
ER_IB_MSG_297, 5460
ER_IB_MSG_298, 5460
ER_IB_MSG_299, 5460
ER_IB_MSG_3, 5430
ER_IB_MSG_30, 5432
ER_IB_MSG_300, 5460
ER_IB_MSG_301, 5460
ER_IB_MSG_302, 5460
ER_IB_MSG_303, 5461
ER_IB_MSG_304, 5461
ER_IB_MSG_305, 5461
ER_IB_MSG_306, 5461
ER_IB_MSG_307, 5461
ER_IB_MSG_308, 5461
ER_IB_MSG_309, 5461
ER_IB_MSG_31, 5433
ER_IB_MSG_310, 5461
ER_IB_MSG_311, 5462
ER_IB_MSG_312, 5462
ER_IB_MSG_313, 5462
ER_IB_MSG_314, 5462
ER_IB_MSG_315, 5462
ER_IB_MSG_316, 5462
ER_IB_MSG_317, 5462
ER_IB_MSG_318, 5462
ER_IB_MSG_319, 5462
ER_IB_MSG_32, 5433
ER_IB_MSG_320, 5462
ER_IB_MSG_321, 5463
ER_IB_MSG_322, 5463
ER_IB_MSG_323, 5463
ER_IB_MSG_324, 5463
ER_IB_MSG_325, 5463
ER_IB_MSG_326, 5463
ER_IB_MSG_327, 5463
ER_IB_MSG_328, 5463
ER_IB_MSG_329, 5463
ER_IB_MSG_33, 5433
ER_IB_MSG_330, 5463
ER_IB_MSG_331, 5464
ER_IB_MSG_332, 5464
ER_IB_MSG_333, 5464
ER_IB_MSG_334, 5464
ER_IB_MSG_335, 5464
ER_IB_MSG_336, 5464
ER_IB_MSG_337, 5464
ER_IB_MSG_338, 5464

ER_IB_MSG_339, 5464
ER_IB_MSG_34, 5433
ER_IB_MSG_340, 5464
ER_IB_MSG_341, 5465
ER_IB_MSG_342, 5465
ER_IB_MSG_343, 5465
ER_IB_MSG_344, 5465
ER_IB_MSG_345, 5465
ER_IB_MSG_346, 5465
ER_IB_MSG_347, 5465
ER_IB_MSG_348, 5465
ER_IB_MSG_349, 5465
ER_IB_MSG_35, 5433
ER_IB_MSG_350, 5465
ER_IB_MSG_351, 5466
ER_IB_MSG_352, 5466
ER_IB_MSG_353, 5466
ER_IB_MSG_354, 5466
ER_IB_MSG_355, 5466
ER_IB_MSG_356, 5466
ER_IB_MSG_357, 5466
ER_IB_MSG_358, 5466
ER_IB_MSG_359, 5466
ER_IB_MSG_36, 5433
ER_IB_MSG_360, 5466
ER_IB_MSG_361, 5467
ER_IB_MSG_362, 5467
ER_IB_MSG_363, 5467
ER_IB_MSG_364, 5467
ER_IB_MSG_365, 5467
ER_IB_MSG_366, 5467
ER_IB_MSG_367, 5467
ER_IB_MSG_368, 5467
ER_IB_MSG_369, 5467
ER_IB_MSG_37, 5433
ER_IB_MSG_370, 5468
ER_IB_MSG_371, 5468
ER_IB_MSG_372, 5468
ER_IB_MSG_373, 5468
ER_IB_MSG_374, 5468
ER_IB_MSG_375, 5468
ER_IB_MSG_376, 5468
ER_IB_MSG_377, 5468
ER_IB_MSG_378, 5468
ER_IB_MSG_379, 5468
ER_IB_MSG_38, 5433
ER_IB_MSG_380, 5469
ER_IB_MSG_381, 5469
ER_IB_MSG_382, 5469
ER_IB_MSG_383, 5469
ER_IB_MSG_384, 5469
ER_IB_MSG_385, 5469
ER_IB_MSG_386, 5469
ER_IB_MSG_387, 5469

ER_IB_MSG_388, 5469
ER_IB_MSG_389, 5469
ER_IB_MSG_39, 5433
ER_IB_MSG_390, 5470
ER_IB_MSG_391, 5470
ER_IB_MSG_392, 5470
ER_IB_MSG_393, 5470
ER_IB_MSG_394, 5470
ER_IB_MSG_395, 5470
ER_IB_MSG_396, 5470
ER_IB_MSG_397, 5470
ER_IB_MSG_398, 5470
ER_IB_MSG_399, 5470
ER_IB_MSG_4, 5430
ER_IB_MSG_40, 5433
ER_IB_MSG_400, 5471
ER_IB_MSG_401, 5471
ER_IB_MSG_402, 5471
ER_IB_MSG_403, 5471
ER_IB_MSG_404, 5471
ER_IB_MSG_405, 5471
ER_IB_MSG_406, 5471
ER_IB_MSG_407, 5471
ER_IB_MSG_408, 5471
ER_IB_MSG_409, 5471
ER_IB_MSG_41, 5434
ER_IB_MSG_410, 5472
ER_IB_MSG_411, 5472
ER_IB_MSG_412, 5472
ER_IB_MSG_413, 5472
ER_IB_MSG_414, 5472
ER_IB_MSG_415, 5472
ER_IB_MSG_416, 5472
ER_IB_MSG_417, 5472
ER_IB_MSG_418, 5472
ER_IB_MSG_419, 5472
ER_IB_MSG_42, 5434
ER_IB_MSG_420, 5473
ER_IB_MSG_421, 5473
ER_IB_MSG_422, 5473
ER_IB_MSG_423, 5473
ER_IB_MSG_424, 5473
ER_IB_MSG_425, 5473
ER_IB_MSG_426, 5473
ER_IB_MSG_427, 5473
ER_IB_MSG_428, 5473
ER_IB_MSG_429, 5473
ER_IB_MSG_43, 5434
ER_IB_MSG_430, 5474
ER_IB_MSG_431, 5474
ER_IB_MSG_432, 5474
ER_IB_MSG_433, 5474
ER_IB_MSG_434, 5474
ER_IB_MSG_435, 5474

ER_IB_MSG_436, 5474
ER_IB_MSG_437, 5474
ER_IB_MSG_438, 5474
ER_IB_MSG_439, 5474
ER_IB_MSG_44, 5434
ER_IB_MSG_440, 5475
ER_IB_MSG_441, 5475
ER_IB_MSG_442, 5475
ER_IB_MSG_443, 5475
ER_IB_MSG_444, 5475
ER_IB_MSG_445, 5475
ER_IB_MSG_446, 5475
ER_IB_MSG_447, 5475
ER_IB_MSG_448, 5475
ER_IB_MSG_449, 5475
ER_IB_MSG_45, 5434
ER_IB_MSG_450, 5476
ER_IB_MSG_451, 5476
ER_IB_MSG_452, 5476
ER_IB_MSG_453, 5476
ER_IB_MSG_454, 5476
ER_IB_MSG_455, 5476
ER_IB_MSG_456, 5476
ER_IB_MSG_457, 5476
ER_IB_MSG_458, 5476
ER_IB_MSG_459, 5476
ER_IB_MSG_46, 5434
ER_IB_MSG_460, 5477
ER_IB_MSG_461, 5477
ER_IB_MSG_462, 5477
ER_IB_MSG_463, 5477
ER_IB_MSG_464, 5477
ER_IB_MSG_465, 5477
ER_IB_MSG_466, 5477
ER_IB_MSG_467, 5477
ER_IB_MSG_468, 5477
ER_IB_MSG_469, 5477
ER_IB_MSG_47, 5434
ER_IB_MSG_470, 5478
ER_IB_MSG_471, 5478
ER_IB_MSG_472, 5478
ER_IB_MSG_473, 5478
ER_IB_MSG_474, 5478
ER_IB_MSG_475, 5478
ER_IB_MSG_476, 5478
ER_IB_MSG_477, 5478
ER_IB_MSG_478, 5478
ER_IB_MSG_479, 5478
ER_IB_MSG_48, 5434
ER_IB_MSG_480, 5479
ER_IB_MSG_481, 5479
ER_IB_MSG_482, 5479
ER_IB_MSG_483, 5479
ER_IB_MSG_484, 5479

ER_IB_MSG_485, 5479
ER_IB_MSG_486, 5479
ER_IB_MSG_487, 5479
ER_IB_MSG_488, 5479
ER_IB_MSG_489, 5479
ER_IB_MSG_49, 5434
ER_IB_MSG_490, 5480
ER_IB_MSG_491, 5480
ER_IB_MSG_492, 5480
ER_IB_MSG_493, 5480
ER_IB_MSG_494, 5480
ER_IB_MSG_495, 5480
ER_IB_MSG_496, 5480
ER_IB_MSG_497, 5480
ER_IB_MSG_498, 5480
ER_IB_MSG_499, 5480
ER_IB_MSG_5, 5430
ER_IB_MSG_50, 5434
ER_IB_MSG_500, 5481
ER_IB_MSG_501, 5481
ER_IB_MSG_502, 5481
ER_IB_MSG_503, 5481
ER_IB_MSG_504, 5481
ER_IB_MSG_505, 5481
ER_IB_MSG_506, 5481
ER_IB_MSG_507, 5481
ER_IB_MSG_508, 5481
ER_IB_MSG_509, 5481
ER_IB_MSG_51, 5435
ER_IB_MSG_510, 5482
ER_IB_MSG_511, 5482
ER_IB_MSG_512, 5482
ER_IB_MSG_513, 5482
ER_IB_MSG_514, 5482
ER_IB_MSG_515, 5482
ER_IB_MSG_516, 5482
ER_IB_MSG_517, 5482
ER_IB_MSG_518, 5482
ER_IB_MSG_519, 5482
ER_IB_MSG_52, 5435
ER_IB_MSG_520, 5483
ER_IB_MSG_521, 5483
ER_IB_MSG_522, 5483
ER_IB_MSG_523, 5483
ER_IB_MSG_524, 5483
ER_IB_MSG_525, 5483
ER_IB_MSG_526, 5483
ER_IB_MSG_527, 5483
ER_IB_MSG_528, 5483
ER_IB_MSG_529, 5483
ER_IB_MSG_53, 5435
ER_IB_MSG_530, 5484
ER_IB_MSG_531, 5484
ER_IB_MSG_532, 5484

ER_IB_MSG_533, 5484
ER_IB_MSG_534, 5484
ER_IB_MSG_535, 5484
ER_IB_MSG_536, 5484
ER_IB_MSG_537, 5484
ER_IB_MSG_538, 5484
ER_IB_MSG_539, 5484
ER_IB_MSG_54, 5435
ER_IB_MSG_540, 5485
ER_IB_MSG_541, 5485
ER_IB_MSG_542, 5485
ER_IB_MSG_543, 5485
ER_IB_MSG_544, 5485
ER_IB_MSG_545, 5485
ER_IB_MSG_546, 5485
ER_IB_MSG_547, 5485
ER_IB_MSG_548, 5485
ER_IB_MSG_549, 5485
ER_IB_MSG_55, 5435
ER_IB_MSG_550, 5486
ER_IB_MSG_551, 5486
ER_IB_MSG_552, 5486
ER_IB_MSG_553, 5486
ER_IB_MSG_554, 5486
ER_IB_MSG_555, 5486
ER_IB_MSG_556, 5486
ER_IB_MSG_557, 5486
ER_IB_MSG_558, 5486
ER_IB_MSG_559, 5486
ER_IB_MSG_56, 5435
ER_IB_MSG_560, 5487
ER_IB_MSG_561, 5487
ER_IB_MSG_562, 5487
ER_IB_MSG_563, 5487
ER_IB_MSG_564, 5487
ER_IB_MSG_565, 5487
ER_IB_MSG_566, 5487
ER_IB_MSG_567, 5487
ER_IB_MSG_568, 5487
ER_IB_MSG_569, 5487
ER_IB_MSG_57, 5435
ER_IB_MSG_570, 5488
ER_IB_MSG_571, 5488
ER_IB_MSG_572, 5488
ER_IB_MSG_573, 5488
ER_IB_MSG_574, 5488
ER_IB_MSG_575, 5488
ER_IB_MSG_576, 5488
ER_IB_MSG_577, 5488
ER_IB_MSG_578, 5488
ER_IB_MSG_579, 5488
ER_IB_MSG_58, 5435
ER_IB_MSG_580, 5489
ER_IB_MSG_581, 5489

ER_IB_MSG_582, 5489
ER_IB_MSG_583, 5489
ER_IB_MSG_584, 5489
ER_IB_MSG_585, 5489
ER_IB_MSG_586, 5489
ER_IB_MSG_587, 5489
ER_IB_MSG_588, 5489
ER_IB_MSG_589, 5489
ER_IB_MSG_59, 5435
ER_IB_MSG_590, 5490
ER_IB_MSG_591, 5490
ER_IB_MSG_592, 5490
ER_IB_MSG_593, 5490
ER_IB_MSG_594, 5490
ER_IB_MSG_595, 5490
ER_IB_MSG_596, 5490
ER_IB_MSG_597, 5490
ER_IB_MSG_598, 5490
ER_IB_MSG_599, 5490
ER_IB_MSG_6, 5430
ER_IB_MSG_60, 5435
ER_IB_MSG_600, 5491
ER_IB_MSG_601, 5491
ER_IB_MSG_602, 5491
ER_IB_MSG_603, 5491
ER_IB_MSG_604, 5491
ER_IB_MSG_605, 5491
ER_IB_MSG_606, 5491
ER_IB_MSG_607, 5491
ER_IB_MSG_608, 5491
ER_IB_MSG_609, 5491
ER_IB_MSG_61, 5436
ER_IB_MSG_610, 5492
ER_IB_MSG_611, 5492
ER_IB_MSG_612, 5492
ER_IB_MSG_613, 5492
ER_IB_MSG_614, 5492
ER_IB_MSG_615, 5492
ER_IB_MSG_616, 5492
ER_IB_MSG_617, 5492
ER_IB_MSG_618, 5492
ER_IB_MSG_619, 5492
ER_IB_MSG_62, 5436
ER_IB_MSG_620, 5493
ER_IB_MSG_621, 5493
ER_IB_MSG_622, 5493
ER_IB_MSG_623, 5493
ER_IB_MSG_624, 5493
ER_IB_MSG_625, 5493
ER_IB_MSG_626, 5493
ER_IB_MSG_627, 5493
ER_IB_MSG_628, 5493
ER_IB_MSG_629, 5493
ER_IB_MSG_63, 5436

ER_IB_MSG_630, 5494
ER_IB_MSG_631, 5494
ER_IB_MSG_632, 5494
ER_IB_MSG_633, 5494
ER_IB_MSG_634, 5494
ER_IB_MSG_635, 5494
ER_IB_MSG_636, 5494
ER_IB_MSG_637, 5494
ER_IB_MSG_638, 5494
ER_IB_MSG_639, 5494
ER_IB_MSG_64, 5436
ER_IB_MSG_640, 5495
ER_IB_MSG_641, 5495
ER_IB_MSG_642, 5495
ER_IB_MSG_643, 5495
ER_IB_MSG_644, 5495
ER_IB_MSG_645, 5495
ER_IB_MSG_646, 5495
ER_IB_MSG_647, 5495
ER_IB_MSG_648, 5495
ER_IB_MSG_649, 5495
ER_IB_MSG_65, 5436
ER_IB_MSG_650, 5496
ER_IB_MSG_651, 5496
ER_IB_MSG_652, 5496
ER_IB_MSG_653, 5496
ER_IB_MSG_654, 5496
ER_IB_MSG_655, 5496
ER_IB_MSG_656, 5496
ER_IB_MSG_657, 5496
ER_IB_MSG_658, 5496
ER_IB_MSG_659, 5497
ER_IB_MSG_66, 5436
ER_IB_MSG_660, 5497
ER_IB_MSG_661, 5497
ER_IB_MSG_662, 5497
ER_IB_MSG_663, 5497
ER_IB_MSG_664, 5497
ER_IB_MSG_665, 5497
ER_IB_MSG_666, 5497
ER_IB_MSG_667, 5497
ER_IB_MSG_668, 5498
ER_IB_MSG_669, 5498
ER_IB_MSG_67, 5436
ER_IB_MSG_670, 5498
ER_IB_MSG_671, 5498
ER_IB_MSG_672, 5498
ER_IB_MSG_673, 5498
ER_IB_MSG_674, 5498
ER_IB_MSG_675, 5498
ER_IB_MSG_676, 5498
ER_IB_MSG_677, 5498
ER_IB_MSG_678, 5499
ER_IB_MSG_679, 5499

ER_IB_MSG_68, 5436
ER_IB_MSG_680, 5499
ER_IB_MSG_681, 5499
ER_IB_MSG_682, 5499
ER_IB_MSG_683, 5499
ER_IB_MSG_684, 5499
ER_IB_MSG_685, 5499
ER_IB_MSG_686, 5499
ER_IB_MSG_687, 5499
ER_IB_MSG_688, 5500
ER_IB_MSG_689, 5500
ER_IB_MSG_69, 5436
ER_IB_MSG_690, 5500
ER_IB_MSG_691, 5500
ER_IB_MSG_692, 5500
ER_IB_MSG_693, 5500
ER_IB_MSG_694, 5500
ER_IB_MSG_695, 5500
ER_IB_MSG_696, 5500
ER_IB_MSG_697, 5501
ER_IB_MSG_698, 5501
ER_IB_MSG_699, 5501
ER_IB_MSG_7, 5430
ER_IB_MSG_70, 5436
ER_IB_MSG_700, 5501
ER_IB_MSG_701, 5501
ER_IB_MSG_702, 5501
ER_IB_MSG_703, 5501
ER_IB_MSG_704, 5501
ER_IB_MSG_705, 5501
ER_IB_MSG_706, 5501
ER_IB_MSG_707, 5502
ER_IB_MSG_708, 5502
ER_IB_MSG_709, 5502
ER_IB_MSG_71, 5437
ER_IB_MSG_710, 5502
ER_IB_MSG_711, 5502
ER_IB_MSG_712, 5502
ER_IB_MSG_713, 5502
ER_IB_MSG_714, 5502
ER_IB_MSG_715, 5502
ER_IB_MSG_716, 5503
ER_IB_MSG_717, 5503
ER_IB_MSG_718, 5503
ER_IB_MSG_719, 5503
ER_IB_MSG_72, 5437
ER_IB_MSG_720, 5503
ER_IB_MSG_721, 5503
ER_IB_MSG_722, 5503
ER_IB_MSG_723, 5503
ER_IB_MSG_724, 5503
ER_IB_MSG_725, 5503
ER_IB_MSG_726, 5504
ER_IB_MSG_727, 5504

ER_IB_MSG_728, 5504
ER_IB_MSG_729, 5504
ER_IB_MSG_73, 5437
ER_IB_MSG_730, 5504
ER_IB_MSG_731, 5504
ER_IB_MSG_732, 5504
ER_IB_MSG_733, 5504
ER_IB_MSG_734, 5504
ER_IB_MSG_735, 5505
ER_IB_MSG_736, 5505
ER_IB_MSG_737, 5505
ER_IB_MSG_738, 5505
ER_IB_MSG_739, 5505
ER_IB_MSG_74, 5437
ER_IB_MSG_740, 5505
ER_IB_MSG_741, 5505
ER_IB_MSG_742, 5505
ER_IB_MSG_743, 5505
ER_IB_MSG_744, 5505
ER_IB_MSG_745, 5506
ER_IB_MSG_746, 5506
ER_IB_MSG_747, 5506
ER_IB_MSG_748, 5506
ER_IB_MSG_749, 5506
ER_IB_MSG_75, 5437
ER_IB_MSG_750, 5506
ER_IB_MSG_751, 5506
ER_IB_MSG_752, 5506
ER_IB_MSG_753, 5506
ER_IB_MSG_754, 5506
ER_IB_MSG_755, 5507
ER_IB_MSG_756, 5507
ER_IB_MSG_757, 5507
ER_IB_MSG_758, 5507
ER_IB_MSG_759, 5507
ER_IB_MSG_76, 5437
ER_IB_MSG_760, 5507
ER_IB_MSG_761, 5507
ER_IB_MSG_762, 5507
ER_IB_MSG_763, 5507
ER_IB_MSG_764, 5507
ER_IB_MSG_765, 5508
ER_IB_MSG_766, 5508
ER_IB_MSG_767, 5508
ER_IB_MSG_768, 5508
ER_IB_MSG_769, 5508
ER_IB_MSG_77, 5437
ER_IB_MSG_770, 5508
ER_IB_MSG_771, 5508
ER_IB_MSG_772, 5508
ER_IB_MSG_773, 5508
ER_IB_MSG_774, 5508
ER_IB_MSG_775, 5509
ER_IB_MSG_776, 5509

ER_IB_MSG_777, 5509
ER_IB_MSG_778, 5509
ER_IB_MSG_779, 5509
ER_IB_MSG_78, 5437
ER_IB_MSG_780, 5509
ER_IB_MSG_781, 5509
ER_IB_MSG_782, 5509
ER_IB_MSG_783, 5509
ER_IB_MSG_784, 5509
ER_IB_MSG_785, 5510
ER_IB_MSG_786, 5510
ER_IB_MSG_787, 5510
ER_IB_MSG_788, 5510
ER_IB_MSG_789, 5510
ER_IB_MSG_79, 5437
ER_IB_MSG_790, 5510
ER_IB_MSG_791, 5510
ER_IB_MSG_792, 5510
ER_IB_MSG_793, 5510
ER_IB_MSG_794, 5510
ER_IB_MSG_795, 5511
ER_IB_MSG_796, 5511
ER_IB_MSG_797, 5511
ER_IB_MSG_798, 5511
ER_IB_MSG_799, 5511
ER_IB_MSG_8, 5430
ER_IB_MSG_80, 5437
ER_IB_MSG_800, 5511
ER_IB_MSG_801, 5511
ER_IB_MSG_802, 5511
ER_IB_MSG_803, 5511
ER_IB_MSG_804, 5511
ER_IB_MSG_805, 5512
ER_IB_MSG_806, 5512
ER_IB_MSG_807, 5512
ER_IB_MSG_808, 5512
ER_IB_MSG_809, 5512
ER_IB_MSG_81, 5438
ER_IB_MSG_810, 5512
ER_IB_MSG_811, 5512
ER_IB_MSG_812, 5512
ER_IB_MSG_813, 5512
ER_IB_MSG_814, 5512
ER_IB_MSG_815, 5513
ER_IB_MSG_816, 5513
ER_IB_MSG_817, 5513
ER_IB_MSG_818, 5513
ER_IB_MSG_819, 5513
ER_IB_MSG_82, 5438
ER_IB_MSG_820, 5513
ER_IB_MSG_821, 5513
ER_IB_MSG_822, 5513
ER_IB_MSG_823, 5513
ER_IB_MSG_824, 5513

ER_IB_MSG_825, 5514
ER_IB_MSG_826, 5514
ER_IB_MSG_827, 5514
ER_IB_MSG_828, 5514
ER_IB_MSG_829, 5514
ER_IB_MSG_83, 5438
ER_IB_MSG_830, 5514
ER_IB_MSG_831, 5514
ER_IB_MSG_832, 5514
ER_IB_MSG_833, 5514
ER_IB_MSG_834, 5514
ER_IB_MSG_835, 5515
ER_IB_MSG_836, 5515
ER_IB_MSG_837, 5515
ER_IB_MSG_838, 5515
ER_IB_MSG_839, 5515
ER_IB_MSG_84, 5438
ER_IB_MSG_840, 5515
ER_IB_MSG_841, 5515
ER_IB_MSG_842, 5515
ER_IB_MSG_843, 5515
ER_IB_MSG_844, 5515
ER_IB_MSG_845, 5516
ER_IB_MSG_846, 5516
ER_IB_MSG_847, 5516
ER_IB_MSG_848, 5516
ER_IB_MSG_849, 5516
ER_IB_MSG_85, 5438
ER_IB_MSG_850, 5516
ER_IB_MSG_851, 5516
ER_IB_MSG_852, 5516
ER_IB_MSG_853, 5516
ER_IB_MSG_854, 5516
ER_IB_MSG_855, 5517
ER_IB_MSG_856, 5517
ER_IB_MSG_857, 5517
ER_IB_MSG_858, 5517
ER_IB_MSG_859, 5517
ER_IB_MSG_86, 5438
ER_IB_MSG_860, 5517
ER_IB_MSG_861, 5517
ER_IB_MSG_862, 5517
ER_IB_MSG_863, 5517
ER_IB_MSG_864, 5517
ER_IB_MSG_865, 5518
ER_IB_MSG_866, 5518
ER_IB_MSG_867, 5518
ER_IB_MSG_868, 5518
ER_IB_MSG_869, 5518
ER_IB_MSG_87, 5438
ER_IB_MSG_870, 5518
ER_IB_MSG_871, 5518
ER_IB_MSG_872, 5518
ER_IB_MSG_873, 5518

ER_IB_MSG_874, 5518
ER_IB_MSG_875, 5519
ER_IB_MSG_876, 5519
ER_IB_MSG_877, 5519
ER_IB_MSG_878, 5519
ER_IB_MSG_879, 5519
ER_IB_MSG_88, 5438
ER_IB_MSG_880, 5519
ER_IB_MSG_881, 5519
ER_IB_MSG_882, 5519
ER_IB_MSG_883, 5519
ER_IB_MSG_884, 5519
ER_IB_MSG_885, 5520
ER_IB_MSG_886, 5520
ER_IB_MSG_887, 5520
ER_IB_MSG_888, 5520
ER_IB_MSG_889, 5520
ER_IB_MSG_89, 5438
ER_IB_MSG_890, 5520
ER_IB_MSG_891, 5520
ER_IB_MSG_892, 5520
ER_IB_MSG_893, 5520
ER_IB_MSG_894, 5520
ER_IB_MSG_895, 5521
ER_IB_MSG_896, 5521
ER_IB_MSG_897, 5521
ER_IB_MSG_898, 5521
ER_IB_MSG_899, 5521
ER_IB_MSG_9, 5430
ER_IB_MSG_90, 5438
ER_IB_MSG_900, 5521
ER_IB_MSG_901, 5521
ER_IB_MSG_902, 5521
ER_IB_MSG_903, 5521
ER_IB_MSG_904, 5521
ER_IB_MSG_905, 5522
ER_IB_MSG_906, 5522
ER_IB_MSG_907, 5522
ER_IB_MSG_908, 5522
ER_IB_MSG_909, 5522
ER_IB_MSG_91, 5439
ER_IB_MSG_910, 5522
ER_IB_MSG_911, 5522
ER_IB_MSG_912, 5522
ER_IB_MSG_913, 5522
ER_IB_MSG_914, 5522
ER_IB_MSG_915, 5523
ER_IB_MSG_916, 5523
ER_IB_MSG_917, 5523
ER_IB_MSG_918, 5523
ER_IB_MSG_919, 5523
ER_IB_MSG_92, 5439
ER_IB_MSG_920, 5523
ER_IB_MSG_921, 5523

ER_IB_MSG_922, 5523
ER_IB_MSG_923, 5523
ER_IB_MSG_924, 5523
ER_IB_MSG_925, 5524
ER_IB_MSG_926, 5524
ER_IB_MSG_927, 5524
ER_IB_MSG_928, 5524
ER_IB_MSG_929, 5524
ER_IB_MSG_93, 5439
ER_IB_MSG_930, 5524
ER_IB_MSG_931, 5524
ER_IB_MSG_932, 5524
ER_IB_MSG_933, 5524
ER_IB_MSG_934, 5524
ER_IB_MSG_935, 5525
ER_IB_MSG_936, 5525
ER_IB_MSG_937, 5525
ER_IB_MSG_938, 5525
ER_IB_MSG_939, 5525
ER_IB_MSG_94, 5439
ER_IB_MSG_940, 5525
ER_IB_MSG_941, 5525
ER_IB_MSG_942, 5525
ER_IB_MSG_943, 5525
ER_IB_MSG_944, 5525
ER_IB_MSG_945, 5526
ER_IB_MSG_946, 5526
ER_IB_MSG_947, 5526
ER_IB_MSG_948, 5526
ER_IB_MSG_949, 5526
ER_IB_MSG_95, 5439
ER_IB_MSG_950, 5526
ER_IB_MSG_951, 5526
ER_IB_MSG_952, 5526
ER_IB_MSG_953, 5526
ER_IB_MSG_954, 5526
ER_IB_MSG_955, 5527
ER_IB_MSG_956, 5527
ER_IB_MSG_957, 5527
ER_IB_MSG_958, 5527
ER_IB_MSG_959, 5527
ER_IB_MSG_96, 5439
ER_IB_MSG_960, 5527
ER_IB_MSG_961, 5527
ER_IB_MSG_962, 5527
ER_IB_MSG_963, 5527
ER_IB_MSG_964, 5527
ER_IB_MSG_965, 5528
ER_IB_MSG_966, 5528
ER_IB_MSG_967, 5528
ER_IB_MSG_968, 5528
ER_IB_MSG_969, 5528
ER_IB_MSG_97, 5439
ER_IB_MSG_970, 5528

ER_IB_MSG_971, 5528
ER_IB_MSG_972, 5528
ER_IB_MSG_973, 5528
ER_IB_MSG_974, 5528
ER_IB_MSG_975, 5529
ER_IB_MSG_976, 5529
ER_IB_MSG_977, 5529
ER_IB_MSG_978, 5529
ER_IB_MSG_979, 5529
ER_IB_MSG_98, 5439
ER_IB_MSG_980, 5529
ER_IB_MSG_981, 5529
ER_IB_MSG_982, 5529
ER_IB_MSG_983, 5529
ER_IB_MSG_984, 5529
ER_IB_MSG_985, 5530
ER_IB_MSG_986, 5530
ER_IB_MSG_987, 5530
ER_IB_MSG_988, 5530
ER_IB_MSG_989, 5530
ER_IB_MSG_99, 5439
ER_IB_MSG_990, 5530
ER_IB_MSG_991, 5530
ER_IB_MSG_992, 5530
ER_IB_MSG_993, 5530
ER_IB_MSG_994, 5530
ER_IB_MSG_995, 5531
ER_IB_MSG_996, 5531
ER_IB_MSG_997, 5531
ER_IB_MSG_998, 5531
ER_IB_MSG_999, 5531
ER_IB_MSG_BTREE_LEVEL_LIMIT_EXCEEDED, 5597
ER_IB_MSG_CLOCK_GETTIME_FAILED, 5594
ER_IB_MSG_CLOCK_MONOTONIC_UNSUPPORTED, 5594
ER_IB_MSG_DDL_LOG_DELETE_BY_ID_OK, 5496
ER_IB_MSG_DDL_LOG_DELETE_BY_ID_TMCT, 5591
ER_IB_MSG_DDL_LOG_FAIL_POST_DDL, 5592
ER_IB_MSG_DEPRECATED_INNODB_UNDO_TABLESPACES, 5577
ER_IB_MSG_DIR_DOES_NOT_EXIST, 5577
ER_IB_MSG_ERROR_OPENING_NEW_UNDO_SPACE, 5576
ER_IB_MSG_FAILED_SDI_Z_BUF_ERROR, 5576
ER_IB_MSG_FAILED_SDI_Z_MEM_ERROR, 5576
ER_IB_MSG_FAILED_TO_ALLOCATE_WAIT, 5575
ER_IB_MSG_FAILED_TO_FINISH_TRUNCATE, 5577
ER_IB_MSG_FAIL_TO_SAVE_SPACE_STATE, 5575
ER_IB_MSG_FOUND_WRONG_UNDO_SPACE, 5576
ER_IB_MSG_IGNORE_SCAN_PATH, 5467
ER_IB_MSG_LOCK_FREE_HASH_USAGE_STATS, 5577
ER_IB_MSG_MADVISE_FAILED, 5575
ER_IB_MSG_MADV_DONTDUMP_UNSUPPORTED, 5575
ER_IB_MSG_MAX_UNDO_SPACES_REACHED, 5576
ER_IB_MSG_NOT_END_WITH_IBU, 5576
ER_IB_MSG_NO_ENCRYPT_PROGRESS_FOUND, 5569
ER_IB_MSG_NUM_POOLS, 5575

ER_IB_MSG_POST_RECOVER_DDL_LOG_RECOVER, 5592
ER_IB_MSG_POST_RECOVER_POST_TS_ENCRYPT, 5592
ER_IB_MSG_RESUME_OP_FOR_SPACE, 5569
ER_IB_MSG_SDI_Z_STREAM_ERROR, 5576
ER_IB_MSG_SDI_Z_UNKNOWN_ERROR, 5576
ER_IB_MSG_SKIP_HIDDEN_DIR, 5603
ER_IB_MSG_TOO_LONG_PATH, 5593
ER_IB_MSG_UNDO_TRUNC_BEFORE_UNDO_LOGGING, 5577
ER_IB_MSG_UNDO_TRUNC_BEFOR_DD_UPDATE, 5577
ER_IB_MSG_UNDO_TRUNC_EMPTY_FILE, 5576
ER_IB_MSG_UNDO_TRUNK_BEFORE_RSEG, 5577
ER_IB_MSG_UNEXPECTED_FILE_EXISTS, 5461
ER_IB_MSG_USING_UNDO_SPACE, 5575
ER_IB_MSG_WAIT_FOR_ENCRYPT_THREAD, 5569
ER_IB_RECV_FIRST_REC_GROUP_INVALID, 5593
ER_IDENT_CAUSES_TOO_LONG_PATH, 5155
ER_IDENT_CAUSES_TOO_LONG_PATH_IN_UPGRADE, 5600
ER_ILLEGAL_GRANT_FOR_TABLE, 5109
ER_ILLEGAL_HA, 5100
ER_ILLEGAL_HA_CREATE_OPTION, 5130
ER_ILLEGAL_PRIVILEGE_LEVEL, 5184
ER_ILLEGAL_REFERENCE, 5115
ER_ILLEGAL_USER_VAR, 5162
ER_ILLEGAL_VALUE_FOR_TYPE, 5123
ER_IMP_INCOMPATIBLE_DD_VERSION, 5183
ER_IMP_INCOMPATIBLE_MYSQLD_VERSION, 5183
ER_IMP_INCOMPATIBLE_SDI_VERSION, 5183
ER_IMP_NO_FILES_MATCHED, 5183
ER_IMP_SCHEMA_DOES_NOT_EXIST, 5183
ER_IMP_TABLE_ALREADY_EXISTS, 5183
ER_INCONSISTENT_ERROR, 5561
ER_INCONSISTENT_PARTITION_INFO_ERROR, 5130
ER_INCONSISTENT_TYPE_OF_FUNCTIONS_ERROR, 5130
ER_INCORRECT_CURRENT_PASSWORD, 5215
ER_INCORRECT_GLOBAL_LOCAL_VAR, 5114
ER_INCORRECT_TYPE, 5162
ER_INDEX_COLUMN_TOO_LONG, 5144
ER_INDEX_CORRUPT, 5144
ER_INDEX_TYPE_NOT_SUPPORTED_FOR_SPATIAL_INDEX, 5196
ER_INIT_BOOTSTRAP_COMPLETE, 5270
ER_INIT_CANT_OPEN_BOOTSTRAP_FILE, 5270
ER_INIT_CREATING_DD, 5270
ER_INIT_DATADIR_EXISTS_AND_NOT_WRITABLE_WONT_INITIALIZE, 5270
ER_INIT_DATADIR_EXISTS_AND_PATH_TOO_LONG_WONT_INITIALIZE, 5270
ER_INIT_DATADIR_EXISTS_WONT_INITIALIZE, 5270
ER_INIT_DATADIR_NOT_EMPTY_WONT_INITIALIZE, 5270
ER_INIT_FAILED_TO_GENERATE_ROOT_PASSWORD, 5603
ER_INIT_GENERATING_TEMP_PASSWORD_FOR_ROOT, 5269
ER_INIT_ROOT_WITHOUT_PASSWORD, 5269
ER_INNODB_ACTIVE_INDEX_CHANGE_FAILED, 5338
ER_INNODB_CANNOT_BE_IGNORED, 5193
ER_INNODB_CANNOT_CREATE_TABLE, 5339
ER_INNODB_CANT_BUILD_INDEX_XLATION_TABLE_FOR, 5320
ER_INNODB_CANT_FIND_INDEX_IN_INNODB_DD, 5319

ER_INNODB_CANT_OPEN_TABLE, 5319
ER_INNODB_CLOSING_CONNECTION_ROLLS_BACK, 5319
ER_INNODB_CLUSTERED_INDEX_PRIVATE, 5320
ER_INNODB_DIFF_IN_REF_LEN, 5338
ER_INNODB_DIRTY_WATER_MARK_NOT_LOW, 5318
ER_INNODB_ERROR_LOGGER_FATAL_MSG, 5341
ER_INNODB_ERROR_LOGGER_MSG, 5341
ER_INNODB_FAILED_TO_FIND_IDX, 5338
ER_INNODB_FAILED_TO_FIND_IDX_FROM_DICT_CACHE, 5338
ER_INNODB_FAILED_TO_FIND_IDX_WITH_KEY_NO, 5338
ER_INNODB_FILES_SAME, 5319
ER_INNODB_FORCED_RECOVERY, 5157
ER_INNODB_FT_AUX_NOT_HEX_ID, 5157
ER_INNODB_FT_LIMIT, 5150
ER_INNODB_FT_WRONG_DOCID_COLUMN, 5151
ER_INNODB_FT_WRONG_DOCID_INDEX, 5151
ER_INNODB_IDX_CNT_FEWER_THAN_DEFINED_IN_MYSQL, 5339
ER_INNODB_IDX_CNT_MORE_THAN_DEFINED_IN_MYSQL, 5339
ER_INNODB_IDX_COLUMN_CNT_DIFF, 5339
ER_INNODB_ILLEGAL_COLON_IN_POOL, 5318
ER_INNODB_IMPORT_ERROR, 5152
ER_INNODB_INDEX_COLUMN_INFO_UNLIKE_MYSQLS, 5319
ER_INNODB_INDEX_CORRUPT, 5152
ER_INNODB_INTERNAL_INDEX, 5339
ER_INNODB_INVALID_INNODB_UNDO_DIRECTORY, 5318
ER_INNODB_INVALID_INNODB_UNDO_DIRECTORY_LOCATION, 5579
ER_INNODB_INVALID_LOG_GROUP_HOME_DIR, 5318
ER_INNODB_INVALID_MONITOR_COUNTER_NAME, 5340
ER_INNODB_INVALID_PAGE_SIZE, 5318
ER_INNODB_IO_CAPACITY_EXCEEDS_MAX, 5319
ER_INNODB_MANDATORY, 5228
ER_INNODB_MONITOR_DEFAULT_VALUE_NOT_DEFINED, 5339
ER_INNODB_MONITOR_IS_ENABLED, 5339
ER_INNODB_NO_FT_TEMP_TABLE, 5150
ER_INNODB_ONLINE_LOG_TOO_BIG, 5151
ER_INNODB_PARTITION_TABLE_LOWERCASED, 5320
ER_INNODB_PK_NOT_IN_MYSQL, 5320
ER_INNODB_PK_ONLY_IN_MYSQL, 5320
ER_INNODB_READ_ONLY, 5156
ER_INNODB_REDO_LOG_ARCHIVE_ACTIVE, 5210
ER_INNODB_REDO_LOG_ARCHIVE_DIRS_INVALID, 5209
ER_INNODB_REDO_LOG_ARCHIVE_DIR_CLASH, 5210
ER_INNODB_REDO_LOG_ARCHIVE_DIR_EMPTY, 5209
ER_INNODB_REDO_LOG_ARCHIVE_DIR_PERMISSIONS, 5210
ER_INNODB_REDO_LOG_ARCHIVE_FAILED, 5210
ER_INNODB_REDO_LOG_ARCHIVE_FILE_CREATE, 5210
ER_INNODB_REDO_LOG_ARCHIVE_INACTIVE, 5210
ER_INNODB_REDO_LOG_ARCHIVE_LABEL_NOT_FOUND, 5209
ER_INNODB_REDO_LOG_ARCHIVE_NO_SUCH_DIR, 5210
ER_INNODB_REDO_LOG_ARCHIVE_SESSION, 5210
ER_INNODB_REDO_LOG_ARCHIVE_START_SUBDIR_PATH, 5209
ER_INNODB_REDO_LOG_ARCHIVE_START_TIMEOUT, 5209
ER_INNODB_TRX_XLATION_TABLE_OOM, 5319
ER_INNODB_UNDO_LOG_FULL, 5159

ER_INNODB_UNKNOWN_COLLATION, 5318
ER_INNODB_UNREGISTERED_TRX_ACTIVE, 5319
ER_INNODB_USE_MONITOR_GROUP_NAME, 5339
ER_INSECURE_CHANGE_MASTER, 5148
ER_INSECURE_PLAIN_TEXT, 5148
ER_INSERT_INFO, 5105
ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_BINLOG_DIRECT, 5142
ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_BINLOG_FORMAT, 5142
ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_SQL_LOG_BIN, 5143
ER_INTERNAL_ERROR, 5152
ER_INVALID_ADMIN_ADDRESS, 5580
ER_INVALID_ARGUMENT_FOR_LOGARITHM, 5159
ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE, 5174
ER_INVALID_BITWISE_OPERANDS_SIZE, 5174
ER_INVALID_CAST_TO_JSON, 5169
ER_INVALID_CHARACTER_STRING, 5118
ER_INVALID_CHARSET_AND_DEFAULT_IS_MB, 5256
ER_INVALID_COLLATION_FOR_CHARSET, 5325
ER_INVALID_DD_OBJECT, 5173
ER_INVALID_DD_OBJECT_ID, 5174
ER_INVALID_DD_OBJECT_NAME, 5174
ER_INVALID_DEFAULT, 5103
ER_INVALID_DEFAULT_UTF8MB4_COLLATION, 5195
ER_INVALID_ENCRYPTION_OPTION, 5171
ER_INVALID_ENCRYPTION_REQUEST, 5207
ER_INVALID_ERROR_LOG_NAME, 5239
ER_INVALID_FIELD_SIZE, 5158
ER_INVALID_GEOJSON_CRS_NOT_TOP_LEVEL, 5190
ER_INVALID_GEOJSON_MISSING_MEMBER, 5163
ER_INVALID_GEOJSON_UNSPECIFIED, 5163
ER_INVALID_GEOJSON_WRONG_TYPE, 5163
ER_INVALID_GROUP_FUNC_USE, 5106
ER_INVALID_INFO_IN_FRM, 5257
ER_INVALID_INSTRUMENT, 5228
ER_INVALID_JSON_BINARY_DATA, 5168
ER_INVALID_JSON_CHARSET, 5169
ER_INVALID_JSON_CHARSET_IN_FUNCTION, 5169
ER_INVALID_JSON_DATA, 5163
ER_INVALID_JSON_PATH, 5168
ER_INVALID_JSON_PATH_ARRAY_CELL, 5170
ER_INVALID_JSON_PATH_CHARSET, 5169
ER_INVALID_JSON_PATH_WILDCARD, 5169
ER_INVALID_JSON_TEXT, 5168
ER_INVALID_JSON_TEXT_IN_PARAM, 5168
ER_INVALID_JSON_TYPE, 5211
ER_INVALID_JSON_VALUE_FOR_CAST, 5169
ER_INVALID_JSON_VALUE_FOR_FUNC_INDEX, 5216
ER_INVALID_MULTIPLE_CLAUSES, 5208
ER_INVALID_NAMED_PIPE_FULL_ACCESS_GROUP, 5588
ER_INVALID_NO_OF_ARGS, 5182
ER_INVALID_ON_UPDATE, 5118
ER_INVALID_OPTION_CHARACTERS, 5178
ER_INVALID_OPTION_END_CHARACTER, 5178
ER_INVALID_OPTION_KEY, 5177

ER_INVALID_OPTION_KEY_VALUE_PAIR, 5178
ER_INVALID_OPTION_START_CHARACTER, 5178
ER_INVALID_OPTION_VALUE, 5177
ER_INVALID_OR_OLD_TABLE_OR_DB_NAME, 5277
ER_INVALID_PROPERTY_KEY, 5583
ER_INVALID_REPLICATION_TIMESTAMPS, 5328
ER_INVALID_RPL_WILD_TABLE_FILTER_PATTERN, 5162
ER_INVALID_THREAD_ID, 5189
ER_INVALID_THREAD_PRIORITY, 5188
ER_INVALID_TYPE_FOR_JSON, 5169
ER_INVALID_USE_OF_FORCE_OPTION, 5189
ER_INVALID_USE_OF_NULL, 5108
ER_INVALID_VALUE_FOR_ENFORCE_GTID_CONSISTENCY, 5223
ER_INVALID_VALUE_OF_BIND_ADDRESSES, 5570
ER_INVALID_VCPU_ID, 5188
ER_INVALID_VCPU_RANGE, 5188
ER_INVALID_YEAR_COLUMN_LENGTH, 5152
ER_IO_ERR_LOG_INDEX_READ, 5123
ER_IO_READ_ERROR, 5151
ER_IO_WRITE_ERROR, 5152
ER_IPSOCK_ERROR, 5104
ER_IS_QUERY_INVALID_CLAUSE, 5177
ER_I_S_SKIPPED_TABLESPACE, 5186
ER_JSON_BAD_ONE_OR_ALL_ARG, 5169
ER_JSON_DOCUMENT_NULL_KEY, 5169
ER_JSON_DOCUMENT_TOO_DEEP, 5169
ER_JSON_KEY_TOO_BIG, 5169
ER_JSON_PARSE_ERROR, 5242
ER_JSON_SCHEMA_VALIDATION_ERROR_WITH_DETAILED_REPORT, 5219
ER_JSON_USED_AS_KEY, 5169
ER_JSON_VACUOUS_PATH, 5169
ER_JSON_VALUE_OUT_OF_RANGE_FOR_FUNC_INDEX, 5216
ER_JSON_VALUE_TOO_BIG, 5169
ER_JT_MAX_NESTED_PATH, 5190
ER_JT_VALUE_OUT_OF_RANGE, 5189
ER_KEYCACHE_OOM, 5231
ER_KEYRING_ACCESS_DENIED_ERROR, 5173
ER_KEYRING_AWS_CMK_ID_NOT_SET, 5382
ER_KEYRING_AWS_FAILED_TO_ACCESS_DATA_FILE, 5381
ER_KEYRING_AWS_FAILED_TO_ACCESS_KEY_FROM_CONF_FILE, 5381
ER_KEYRING_AWS_FAILED_TO_ACCESS_KEY_ID_FROM_CONF_FILE, 5381
ER_KEYRING_AWS_FAILED_TO_ACCESS_OR_CREATE_KEYRING_DATA_FILE, 5381
ER_KEYRING_AWS_FAILED_TO_ACCESS_OR_CREATE_KEYRING_DIR, 5381
ER_KEYRING_AWS_FAILED_TO_CONNECT_KMS, 5383
ER_KEYRING_AWS_FAILED_TO_DECRYPT_KEY, 5383
ER_KEYRING_AWS_FAILED_TO_ENCRYPT_KEY, 5383
ER_KEYRING_AWS_FAILED_TO_FLUSH_KEYRING_TO_FILE, 5383
ER_KEYRING_AWS_FAILED_TO_GENERATE_KEY_DUE_TO_INTERNAL_ERROR, 5382
ER_KEYRING_AWS_FAILED_TO_GENERATE_NEW_KEY, 5383
ER_KEYRING_AWS_FAILED_TO_GET_KMS_CREDENTIAL_FROM_CONF_FILE, 5382
ER_KEYRING_AWS_FAILED_TO_INIT_DUE_TO_INTERNAL_ERROR, 5381
ER_KEYRING_AWS_FAILED_TO_INIT_DUE_TO_PLUGIN_INTERNAL_ERROR, 5382
ER_KEYRING_AWS_FAILED_TO_OPEN_CONF_FILE, 5380
ER_KEYRING_AWS_FAILED_TO_RESTORE_FROM_BACKUP_FILE, 5383

ER_KEYRING_AWS_FAILED_TO_RE_ENCRYPT_KEY, 5383
ER_KEYRING_AWS_FAILED_TO_ROTATE_CMK, 5383
ER_KEYRING_AWS_FAILED_TO_SET_CMK_ID, 5380
ER_KEYRING_AWS_FAILED_TO_SET_REGION, 5380
ER_KEYRING_AWS_FOUND_MALFORMED_BACKUP_FILE, 5382
ER_KEYRING_AWS_INCORRECT_FILE, 5382
ER_KEYRING_AWS_INCORRECT_PROXY, 5601
ER_KEYRING_AWS_INCORRECT_REGION, 5383
ER_KEYRING_AWS_INIT_FAILURE, 5382
ER_KEYRING_AWS_INVALID_CONF_FILE_PATH, 5381
ER_KEYRING_AWS_INVALID_DATA_FILE_PATH, 5381
ER_KEYRING_AWS_INVALID_KEY_LENGTH_FOR_CIPHER, 5382
ER_KEYRING_AWS_UDF_AWS_KMS_ERROR, 5172
ER_KEYRING_CHECK_KEY_FAILED_DUE_TO_EMPTY_KEY_ID, 5373
ER_KEYRING_CHECK_KEY_FAILED_DUE_TO_INVALID_KEY, 5373
ER_KEYRING_ENCRYPTED_FILE_DECRYPTION_FAILED, 5379
ER_KEYRING_ENCRYPTED_FILE_ENCRYPTION_FAILED, 5379
ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_CREATE_KEYRING_DIR, 5379
ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_FLUSH_KEYRING, 5379
ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_RESTORE_KEYRING, 5379
ER_KEYRING_ENCRYPTED_FILE_FOUND_MALFORMED_BACKUP_FILE, 5379
ER_KEYRING_ENCRYPTED_FILE_GEN_KEY_FAILED_DUE_TO_INTERNAL_ERROR, 5380
ER_KEYRING_ENCRYPTED_FILE_INCORRECT_KEYRING_FILE, 5378
ER_KEYRING_ENCRYPTED_FILE_INIT_FAILED_DUE_TO_INTERNAL_ERROR, 5380
ER_KEYRING_ENCRYPTED_FILE_INIT_FAILURE, 5380
ER_KEYRING_ENCRYPTED_FILE_INVALID_KEYRING_DIR, 5379
ER_KEYRING_ENCRYPTED_FILE_PASSWORD_IS_INVALID, 5379
ER_KEYRING_ENCRYPTED_FILE_PASSWORD_IS_TOO_LONG, 5380
ER_KEYRING_FAILED_TO_CREATE_KEYRING_DIR, 5373
ER_KEYRING_FAILED_TO_FLUSH_KEYRING_TO_FILE, 5374
ER_KEYRING_FAILED_TO_FLUSH_KEYS_TO_KEYRING, 5375
ER_KEYRING_FAILED_TO_FLUSH_KEYS_TO_KEYRING_BACKUP, 5375
ER_KEYRING_FAILED_TO_GENERATE_KEY, 5373
ER_KEYRING_FAILED_TO_GET_FILE_STAT, 5374
ER_KEYRING_FAILED_TO_LOAD_KEYRING_CONTENT, 5375
ER_KEYRING_FAILED_TO_REMOVE_FILE, 5374
ER_KEYRING_FAILED_TO_REMOVE_KEY_DUE_TO_EMPTY_ID, 5375
ER_KEYRING_FAILED_TO_RESTORE_FROM_BACKUP_FILE, 5374
ER_KEYRING_FAILED_TO_SET_KEYRING_FILE_DATA, 5374
ER_KEYRING_FAILED_TO_TRUNCATE_FILE, 5374
ER_KEYRING_FILE_INIT_FAILED, 5373
ER_KEYRING_FILE_IO_ERROR, 5375
ER_KEYRING_FOUND_MALFORMED_BACKUP_FILE, 5374
ER_KEYRING_INCORRECT_FILE, 5374
ER_KEYRING_INTERNAL_EXCEPTION_FAILED_FILE_INIT, 5373
ER_KEYRING_INVALID_KEY_LENGTH, 5373
ER_KEYRING_INVALID_KEY_TYPE, 5372
ER_KEYRING_KEY_FETCH_FAILED_DUE_TO_EMPTY_KEY_ID, 5375
ER_KEYRING_LOGGER_ERROR_MSG, 5355
ER_KEYRING_MIGRATE_FAILED, 5560
ER_KEYRING_MIGRATION_EXTRA_OPTIONS, 5429
ER_KEYRING_MIGRATION_FAILED, 5343
ER_KEYRING_MIGRATION_FAILURE, 5173
ER_KEYRING_MIGRATION_STATUS, 5173

ER_KEYRING_MIGRATION_SUCCESSFUL, 5343
ER_KEYRING_OKV_CONNECTION_TO_SERVER_FAILED, 5377
ER_KEYRING_OKV_FAILED_TO_ACTIVATE_KEYS, 5377
ER_KEYRING_OKV_FAILED_TO_ADD_ATTRIBUTE, 5377
ER_KEYRING_OKV_FAILED_TO_FETCH_KEY, 5378
ER_KEYRING_OKV_FAILED_TO_FIND_SERVER_ENTRY, 5376
ER_KEYRING_OKV_FAILED_TO_FIND_STANDBY_SERVER_ENTRY, 5376
ER_KEYRING_OKV_FAILED_TO_GENERATE_KEY, 5377
ER_KEYRING_OKV_FAILED_TO_GENERATE_KEY_DUE_TO_INTERNAL_ERROR, 5376
ER_KEYRING_OKV_FAILED_TO_INIT_CLIENT, 5377
ER_KEYRING_OKV_FAILED_TO_INIT_SSL_LAYER, 5377
ER_KEYRING_OKV_FAILED_TO_LOAD_KEY_UID, 5376
ER_KEYRING_OKV_FAILED_TO_LOAD_SSL_TRUST_STORE, 5378
ER_KEYRING_OKV_FAILED_TO_PARSE_CONF_FILE, 5376
ER_KEYRING_OKV_FAILED_TO_REMOVE_KEY, 5377
ER_KEYRING_OKV_FAILED_TO_RETRIEVE_KEY, 5378
ER_KEYRING_OKV_FAILED_TO_RETRIEVE_KEY_SIGNATURE, 5378
ER_KEYRING_OKV_FAILED_TO_SET_CERTIFICATE_FILE, 5378
ER_KEYRING_OKV_FAILED_TO_SET_KEY_FILE, 5378
ER_KEYRING_OKV_FAILED_TO_STORE_KEY, 5377
ER_KEYRING_OKV_FAILED_TO_STORE_OR_GENERATE_KEY, 5378
ER_KEYRING_OKV_INCORRECT_KEY_VAULT_CONFIGURED, 5375
ER_KEYRING_OKV_INIT_FAILED_DUE_TO_INCORRECT_CONF, 5375
ER_KEYRING_OKV_INIT_FAILED_DUE_TO_INTERNAL_ERROR, 5376
ER_KEYRING_OKV_INVALID_KEY_LENGTH_FOR_CIPHER, 5376
ER_KEYRING_OKV_INVALID_KEY_TYPE, 5376
ER_KEYRING_OKV_KEY_MISMATCH, 5378
ER_KEYRING_OPERATION_FAILED_DUE_TO_INTERNAL_ERROR, 5373
ER_KEYRING_UDF_KEYRING_SERVICE_ERROR, 5172
ER_KEYRING_UNKNOWN_ERROR, 5374
ER_KEY_COLUMN_DOES_NOT_EXITS, 5103
ER_KEY_DOES_NOT_EXITS, 5110
ER_KEY_NOT_FOUND, 5100
ER_KEY_PART_0, 5124
ER_KEY_REF_DO_NOT_MATCH_TABLE_REF, 5115
ER_KILLED_THREADS_OF_PLUGIN, 5272
ER_KILLING_THREAD, 5271
ER_KILL_DENIED_ERROR, 5105
ER_LATITUDE_OUT_OF_RANGE, 5184
ER_LCTN_CHANGED, 5343
ER_LCTN_NOT_FOUND, 5560
ER_LDAP_AUTH_CERTIFICATE_NAME, 5422
ER_LDAP_AUTH_COMMUNICATION_HOST_INFO, 5425
ER_LDAP_AUTH_CONNECTION_CREATOR_ENTER, 5425
ER_LDAP_AUTH_CONNECTION_GET_LDAP_INFO_NULL, 5425
ER_LDAP_AUTH_CONNECTION_POOL_INIT_FAILED, 5423
ER_LDAP_AUTH_CONNECTION_POOL_REINIT_ENTER, 5427
ER_LDAP_AUTH_CONNECTION_PUSHED_TO_POOL, 5425
ER_LDAP_AUTH_CONN_POOL_DEINITIALIZING, 5421
ER_LDAP_AUTH_CONN_POOL_INITIALIZING, 5421
ER_LDAP_AUTH_CONN_POOL_NOT_CREATED, 5421
ER_LDAP_AUTH_CREATE_CONNECTION_KEY, 5425
ER_LDAP_AUTH_CREATING_LDAP_CONNECTION, 5424
ER_LDAP_AUTH_DEINIT_FAILED, 5420

ER_LDAP_AUTH_DELETING_CONNECTION_KEY, 5425
ER_LDAP_AUTH_DISTINGUISHED_NAME, 5426
ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_CONNECTION, 5427
ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_OBJECT, 5420
ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_OBJECT_CREATOR, 5420
ER_LDAP_AUTH_FAILED_TO_CREATE_OR_GET_CONNECTION, 5420
ER_LDAP_AUTH_FAILED_TO_DEINITIALIZE_NOT_READY_POOL, 5423
ER_LDAP_AUTH_FAILED_TO_DEINITIALIZE_POOL_IN_RECONSTRUCT_STATE, 5423
ER_LDAP_AUTH_FAILED_TO_ESTABLISH_TLS_CONNECTION, 5427
ER_LDAP_AUTH_FAILED_TO_GET_CONNECTION_AS_PLUGIN_NOT_READY, 5423
ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_DEINIT_STATE, 5423
ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_INIT_STATE, 5423
ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_RECONSTRUCTING, 5423
ER_LDAP_AUTH_FAILED_TO_POOL_DEINIT, 5423
ER_LDAP_AUTH_FAILED_TO_SEARCH_DN, 5427
ER_LDAP_AUTH_FAILED_TO_WRITE_PACKET, 5421
ER_LDAP_AUTH_FREEING_CONNECTION, 5425
ER_LDAP_AUTH_GETTING_CONNECTION_FROM_POOL, 5424
ER_LDAP_AUTH_GRP_DN_PARSING_FAILED, 5585
ER_LDAP_AUTH_GRP_INCORRECT_ATTRIBUTE, 5585
ER_LDAP_AUTH_GRP_INFO_FOUND_IN_MANY_OBJECTS, 5585
ER_LDAP_AUTH_GRP_IS_FULL_DN, 5422
ER_LDAP_AUTH_GRP_NULL_ATTRIBUTE_VALUE, 5585
ER_LDAP_AUTH_GRP_OBJECT_HAS_USER_INFO, 5585
ER_LDAP_AUTH_GRP_SEARCH_NOT_SPECIAL_HDL, 5585
ER_LDAP_AUTH_GRP_SEARCH_SPECIAL_HDL, 5422
ER_LDAP_AUTH_GRP_USER_OBJECT_HAS_GROUP_INFO, 5585
ER_LDAP_AUTH_INFO_FOR_USER, 5421
ER_LDAP_AUTH_INIT_FAILED, 5426
ER_LDAP_AUTH_LDAPS, 5586
ER_LDAP_AUTH_LDAP_INFO_NULL, 5424
ER_LDAP_AUTH_MAX_ALLOWED_CONNECTION_LIMIT_HIT, 5424
ER_LDAP_AUTH_MAX_POOL_SIZE_SET_FAILED, 5424
ER_LDAP_AUTH_METHOD_TO_CLIENT, 5426
ER_LDAP_AUTH_OBJECT_CREATE_TIMESTAMP, 5422
ER_LDAP_AUTH_OR_GROUP_RETRIEVAL_FAILED, 5426
ER_LDAP_AUTH_PLUGIN_FAILED_TO_READ_PACKET, 5424
ER_LDAP_AUTH_POOLED_CONNECTION_KEY, 5425
ER_LDAP_AUTH_POOL_DISABLE_MAX_SIZE_ZERO, 5420
ER_LDAP_AUTH_POOL_GET_FAILED_TO_CREATE_CONNECTION, 5427
ER_LDAP_AUTH_POOL_REINITIALIZING, 5421
ER_LDAP_AUTH_RETURNING_CONNECTION_TO_POOL, 5424
ER_LDAP_AUTH_SASL_BIND_SUCCESS_INFO, 5426
ER_LDAP_AUTH_SASL_PROCESS_SASL, 5426
ER_LDAP_AUTH_SASL_REQUEST_FROM_CLIENT, 5426
ER_LDAP_AUTH_SEARCHED_USER_GRP_NAME, 5422
ER_LDAP_AUTH_SEARCH_USER_GROUP_ATTR_NOT_FOUND, 5424
ER_LDAP_AUTH_SETTING_USERNAME, 5421
ER_LDAP_AUTH_SKIPPING_USER_GROUP_SEARCH, 5420
ER_LDAP_AUTH_STARTED_FOR_USER, 5426
ER_LDAP_AUTH_STARTING_TLS, 5425
ER_LDAP_AUTH_TLS_CONF, 5420
ER_LDAP_AUTH_TLS_CONNECTION, 5420
ER_LDAP_AUTH_USER_AUTH_DATA, 5421

ER_LDAP_AUTH_USER_BIND_FAILED, 5427
ER_LDAP_AUTH_USER_FOUND_IN_MANY_GRPS, 5422
ER_LDAP_AUTH_USER_GROUP_SEARCH_FAILED, 5426
ER_LDAP_AUTH_USER_GROUP_SEARCH_INFO, 5422
ER_LDAP_AUTH_USER_GROUP_SEARCH_ROOT_BIND, 5574
ER_LDAP_AUTH_USER_HAS_MULTIPLE_GRP_NAMES, 5422
ER_LDAP_AUTH_USER_NOT_FOUND_IN_ANY_GRP, 5422
ER_LDAP_AUTH_ZERO_MAX_POOL_SIZE_UNCHANGED, 5421
ER_LDAP_EMPTY_USERDN_PASSWORD, 5596
ER_LDAP_MAPPING_CHECK_DELIMI_QUOTE, 5586
ER_LDAP_MAPPING_EMPTY_MAPPING, 5586
ER_LDAP_MAPPING_GETTING_NEXT_MAPPING, 5587
ER_LDAP_MAPPING_GET_USER_PROXY, 5586
ER_LDAP_MAPPING_INFO, 5586
ER_LDAP_MAPPING_IS_QUOTE, 5587
ER_LDAP_MAPPING_NON_DESIRED_STATE, 5588
ER_LDAP_MAPPING_NO_SEPEARATOR_END_OF_GROUP, 5587
ER_LDAP_MAPPING_PARSING_CURRENT_STATE, 5587
ER_LDAP_MAPPING_PARSING_ERROR, 5587
ER_LDAP_MAPPING_PARSING_MAPPING_INFO, 5587
ER_LDAP_MAPPING_PROCESS_DELIMITER, 5586
ER_LDAP_MAPPING_PROCESS_DELIMITER_COMMA_NOT_FOUND, 5587
ER_LDAP_MAPPING_PROCESS_DELIMITER_EQUAL_NOT_FOUND, 5586
ER_LDAP_MAPPING_PROCESS_DELIMITER_TRY_COMMA, 5587
ER_LDAP_MAPPING_PROCESS_MAPPING, 5586
ER_LDAP_MAPPING_TRIMMING_SPACES, 5587
ER_LDAP_MAPPING_USER_DONT_BELONG_GROUP, 5586
ER_LIMITED_PART_RANGE, 5132
ER_LOAD_DATA_INFILE_FAILED_IN_UNEXPECTED_WAY, 5243
ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR, 5125
ER_LOAD_INFO, 5104
ER_LOCAL_VARIABLE, 5114
ER_LOCKING_SERVICE_DEADLOCK, 5168
ER_LOCKING_SERVICE_TIMEOUT, 5168
ER_LOCKING_SERVICE_WRONG_NAME, 5167
ER_LOCK_ABORTED, 5143
ER_LOCK_DEADLOCK, 5113
ER_LOCK_NOWAIT, 5179
ER_LOCK_ORDER_DEPENDENCIES_SYNTAX, 5597
ER_LOCK_ORDER_FAILED_READ_FILE, 5597
ER_LOCK_ORDER_FAILED_WRITE_FILE, 5597
ER_LOCK_ORDER_INIT_FAILED, 5595
ER_LOCK_ORDER_MESSAGE, 5597
ER_LOCK_ORDER_SCANNER_SYNTAX, 5597
ER_LOCK_OR_ACTIVE_TRANSACTION, 5111
ER_LOCK_REFUSED_BY_ENGINE, 5171
ER_LOCK_TABLE_FULL, 5112
ER_LOCK_WAIT_TIMEOUT, 5112
ER_LOG_BIN_BETTER_WITH_NAME, 5237
ER_LOG_CANNOT_WRITE, 5221
ER_LOG_CANNOT_WRITE_EXTENDED, 5600
ER_LOG_FILES_GIVEN_LOG_OUTPUT_IS_TABLE, 5236
ER_LOG_FILE_CANNOT_OPEN, 5363
ER_LOG_FILE_INVALID, 5236

ER_LOG_GENERAL_CANNOT_OPEN, 5221
ER_LOG_IN_USE, 5123
ER_LOG_OUTPUT_CONTRADICTORY, 5235
ER_LOG_PRINTF_MSG, 5341
ER_LOG_PURGE_NO_FILE, 5137
ER_LOG_PURGE_UNKNOWN_ERR, 5124
ER_LOG_SLOW_CANNOT_OPEN, 5221
ER_LOG_SYSLOG_CANNOT_OPEN, 5221
ER_LOG_SYSLOG_FACILITY_FAIL, 5221
ER_LONGITUDE_OUT_OF_RANGE, 5184
ER_LOWER_CASE_TABLE_NAMES_CS_DD_ON_CI_FS_UNSUPPORTED, 5236
ER_LOWER_CASE_TABLE_NAMES_USING_0, 5236
ER_LOWER_CASE_TABLE_NAMES_USING_2, 5236
ER_MALFORMED_GTID_SET_ENCODING, 5149
ER_MALFORMED_GTID_SET_SPECIFICATION, 5148
ER_MALFORMED_GTID_SPECIFICATION, 5149
ER_MALFORMED_PACKET, 5153
ER_MANDATORY_ROLE, 5185
ER_MASTER, 5111
ER_MASTER_DELAY_VALUE_OUT_OF_RANGE, 5146
ER_MASTER_FATAL_ERROR_READING_BINLOG, 5114
ER_MASTER_HAS_PURGED_REQUIRED_GTIDS, 5150
ER_MASTER_INFO, 5112
ER_MASTER_KEY_ROTATION_BINLOG_FAILED, 5171
ER_MASTER_KEY_ROTATION_NOT_SUPPORTED_BY_SE, 5171
ER_MASTER_KEY_ROTATION_SE_UNAVAILABLE, 5171
ER_MASTER_NET_READ, 5111
ER_MASTER_NET_WRITE, 5111
ER_MAXVALUE_IN_VALUES_IN, 5140
ER_MAX_PREPARED_STMT_COUNT_REACHED, 5129
ER_MECAB_CHARSET_LOADED, 5347
ER_MECAB_CREATE_LATTICE_FAILED, 5347
ER_MECAB_CREATING_MODEL, 5347
ER_MECAB_FAILED_TO_CREATE_MODEL, 5347
ER_MECAB_FAILED_TO_CREATE_TRIGGER, 5347
ER_MECAB_NOT_SUPPORTED, 5346
ER_MECAB_NOT_VERIFIED, 5347
ER_MECAB_OOM_WHILE_PARSING_TEXT, 5347
ER_MECAB_PARSE_FAILED, 5347
ER_MECAB_UNSUPPORTED_CHARSET, 5347
ER_MESSAGE_AND_STATEMENT, 5142
ER_MICROSECOND_TIMER_IS_NOT_AVAILABLE, 5338
ER_MISSING_ACL_SYSTEM_TABLE, 5329
ER_MISSING_CURRENT_PASSWORD, 5215
ER_MISSING_GRANT_SYSTEM_TABLE, 5329
ER_MISSING_HA_CREATE_OPTION, 5159
ER_MISSING_JSON_TABLE_VALUE, 5189
ER_MISSING_KEY, 5157
ER_MISSING_SKIP_SLAVE, 5117
ER_MISSING_TABLESPACE_FILE, 5185
ER_MIXING_NOT_ALLOWED, 5114
ER_MIX_HANDLER_ERROR, 5131
ER_MIX_OF_GROUP_FUNC_AND_FIELDS, 5108
ER_MIX_OF_GROUP_FUNC_AND_FIELDS_V2, 5164

ER_MTS_CANT_PARALLEL, 5147
ER_MTS_CHANGE_MASTER_CANT_RUN_WITH_GAPS, 5151
ER_MTS_EVENT_BIGGER_PENDING_JOBS_SIZE_MAX, 5155
ER_MTS_FEATURE_IS_NOT_SUPPORTED, 5147
ER_MTS_INCONSISTENT_DATA, 5147
ER_MTS_RECOVERY_FAILURE, 5151
ER_MTS_RESET_WORKERS, 5151
ER_MTS_UPDATED_DBS_GREATER_MAX, 5147
ER_MULTIPLE_DEF_CONST_IN_LIST_PART_ERROR, 5131
ER_MULTIPLE_PRI_KEY, 5103
ER_MULTI_UPDATE_KEY_CONFLICT, 5144
ER_MUST_CHANGE_EXPIRED_PASSWORD, 5325
ER_MUST_CHANGE_PASSWORD, 5152
ER_MUST_CHANGE_PASSWORD_LOGIN, 5155
ER_MYINIT_FAILED, 5234
ER_MYISAM_CHECK_METHOD_ERROR, 5340
ER_MYISAM_CRASHED_ERROR, 5340
ER_MYISAM_CRASHED_ERROR_IN, 5245
ER_MYISAM_CRASHED_ERROR_IN_THREAD, 5245
ER_MYSQLBACKUP_MSG, 5602
ER_MYSQL_NATIVE_PASSWORD_SECOND_PASSWORD_USED_INFORMATION, 5581
ER_MY_NET_WRITE_FAILED_FALLING_BACK_ON_STDERR, 5304
ER_M_BIGGER_THAN_D, 5126
ER_NAME_BECOMES_EMPTY, 5129
ER_NANOSECOND_TIMER_IS_NOT_AVAILABLE, 5337
ER_NATIVE_FCT_NAME_COLLISION, 5136
ER_NDB_BINLOG_BLOB_REQUIRES_PK, 5295
ER_NDB_BINLOG_CANT_COMMIT_TO_NDB_BINLOG_INDEX, 5294
ER_NDB_BINLOG_CANT_CREATE_BLOB, 5296
ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB, 5295
ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB_AND_CANT_DROP, 5296
ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB_DROPPED, 5296
ER_NDB_BINLOG_CANT_CREATE_PURGE_THD, 5318
ER_NDB_BINLOG_CANT_DISCOVER_TABLE_FROM_SCHEMA_EVENT, 5290
ER_NDB_BINLOG_CANT_DROP_EVENT_FROM_DB, 5296
ER_NDB_BINLOG_CANT_INJECT_APPLY_STATUS_WRITE_ROW, 5299
ER_NDB_BINLOG_CANT_LOCK_NDB_BINLOG_INDEX, 5293
ER_NDB_BINLOG_CANT_RELEASE_SLOCK, 5291
ER_NDB_BINLOG_CANT_REOPEN_SHADOW_TABLE, 5292
ER_NDB_BINLOG_CANT_WRITE_TO_NDB_BINLOG_INDEX, 5293
ER_NDB_BINLOG_CLEANING_UP_SETUP_LEFTOVERS, 5289
ER_NDB_BINLOG_CLUSTER_FAILURE, 5297
ER_NDB_BINLOG_CLUSTER_HAS_RECONNECTED, 5298
ER_NDB_BINLOG_CLUSTER_RESTARTED_RESET_MASTER_SUGGESTED, 5298
ER_NDB_BINLOG_CREATE_TABLE_EVENT, 5274
ER_NDB_BINLOG_CREATE_TABLE_EVENT_FAILED, 5295
ER_NDB_BINLOG_CREATE_TABLE_EVENT_INFO, 5295
ER_NDB_BINLOG_CREATING_NDBEVENTOPERATION_FAILED, 5296
ER_NDB_BINLOG_DISCOVER_REUSING_OLD_EVENT_OPS, 5296
ER_NDB_BINLOG_DISCOVER_TABLE_EVENT_INFO, 5295
ER_NDB_BINLOG_ERROR_DURING_GCI_COMMIT, 5299
ER_NDB_BINLOG_ERROR_DURING_GCI_ROLLBACK, 5299
ER_NDB_BINLOG_ERROR_HANDLING_SCHEMA_EVENT, 5299
ER_NDB_BINLOG_ERROR_INFO_FROM_DA, 5274

ER_NDB_BINLOG_FAILED_CREATE_EVENT_OPERATIONS_DURING_RENAME, 5274
ER_NDB_BINLOG_FAILED_CREATE_TABLE_EVENT_OPERATIONS, 5274
ER_NDB_BINLOG_FAILED_TO_GET_TABLE, 5295
ER_NDB_BINLOG_FORMAT_CHANGED_FROM_STMT_TO_MIXED, 5275
ER_NDB_BINLOG_GENERIC_MESSAGE, 5294
ER_NDB_BINLOG_GOT_DIST_PRIV_EVENT_FLUSHING_PRIVILEGES, 5292
ER_NDB_BINLOG_GOT_SCHEMA_EVENT, 5292
ER_NDB_BINLOG_INJECTING_RANDOM_WRITE_FAILURE, 5293
ER_NDB_BINLOG_INJECTOR_DISCARDING_ROW_EVENT_METADATA, 5297
ER_NDB_BINLOG_LATEST_TRX_IN_EPOCH_NOT_IN_BINLOG, 5299
ER_NDB_BINLOG_LOST_SCHEMA_CONNECTION_CONTINUING, 5298
ER_NDB_BINLOG_LOST_SCHEMA_CONNECTION_WAITING, 5298
ER_NDB_BINLOG_NDBEVENT_EXECUTE_FAILED, 5296
ER_NDB_BINLOG_NDB_LOG_APPLY_STATUS_FORCING_FULL_USE_WRITE, 5294
ER_NDB_BINLOG_NDB_LOG_TRANSACTION_ID_REQUIRES_V2_ROW_EVENTS, 5294
ER_NDB_BINLOG_NDB_TABLES_INITIALLY_READ_ONLY, 5276
ER_NDB_BINLOG_NDB_TABLES_WRITABLE, 5298
ER_NDB_BINLOG_NOT_LOGGING, 5295
ER_NDB_BINLOG_ONLINE_ALTER_RENAME, 5291
ER_NDB_BINLOG_ONLINE_ALTER_RENAME_COMPLETE, 5292
ER_NDB_BINLOG_OPENING_INDEX, 5293
ER_NDB_BINLOG_RELEASING_EXTRA_SHARE_REFERENCES, 5299
ER_NDB_BINLOG_REMAINING_OPEN_TABLES, 5299
ER_NDB_BINLOG_REMAINING_OPEN_TABLE_INFO, 5299
ER_NDB_BINLOG_RENAME_EVENT, 5274
ER_NDB_BINLOG_REPLY_TO, 5291
ER_NDB_BINLOG_SERVER_SHUTDOWN_DURING_NDB_CLUSTER_START, 5298
ER_NDB_BINLOG_SHUTDOWN_DETECTED, 5298
ER_NDB_BINLOG_SIGNALLING_UNKNOWN_VALUE, 5291
ER_NDB_BINLOG_SKIPPING_DROP_OF_DB_CONTAINING_LOCAL_TABLES, 5292
ER_NDB_BINLOG_SKIPPING_DROP_OF_LOCAL_TABLE, 5292
ER_NDB_BINLOG_SKIPPING_LOCAL_TABLE, 5291
ER_NDB_BINLOG_SKIPPING_OLD_SCHEMA_OPERATION, 5292
ER_NDB_BINLOG_SKIPPING_RENAME_OF_LOCAL_TABLE, 5292
ER_NDB_BINLOG_STARTING_LOG_AT_EPOCH, 5298
ER_NDB_BINLOG_UNHANDLED_ERROR_FOR_TABLE, 5297
ER_NDB_BINLOG_UNKNOWN_NON_DATA_EVENT, 5297
ER_NDB_BINLOG_USING_SERVER_ID_0_SLAVES_WILL_NOT, 5294
ER_NDB_BINLOG_WRITE_TO_NDB_BINLOG_INDEX_FAILED_AFTER_KILL, 5294
ER_NDB_BINLOG_WRITING_TO_NDB_BINLOG_INDEX, 5293
ER_NDB_CANT_ALLOC_GLOBAL_NDB_CLUSTER_CONNECTION, 5309
ER_NDB_CANT_ALLOC_GLOBAL_NDB_OBJECT, 5309
ER_NDB_CANT_ALLOC_NDB_CLUSTER_CONNECTION, 5309
ER_NDB_CANT_FIND_TABLE, 5291
ER_NDB_CANT_PARSE_NDB_CLUSTER_CONNECTION_POOL_NODEIDS, 5308
ER_NDB_CANT_START_CONNECT_THREAD, 5310
ER_NDB_CLEANING_STRAY_TABLES, 5288
ER_NDB_CLEAR_SLOCK_INFO, 5291
ER_NDB_CLUSTER_FAILURE, 5293
ER_NDB_CLUSTER_FIND_ALL_DBS_FAIL, 5289
ER_NDB_CLUSTER_FIND_ALL_DBS_RETRY, 5289
ER_NDB_CLUSTER_FREE_SHARE_INFO, 5276
ER_NDB_CLUSTER_GENERIC_MESSAGE, 5264
ER_NDB_CLUSTER_GET_SHARE_INFO, 5275

ER_NDB_CLUSTER_MARK_SHARE_DROPPED_DESTROYING_SHARE, 5276
ER_NDB_CLUSTER_MARK_SHARE_DROPPED_INFO, 5276
ER_NDB_CLUSTER_OOM_THD_NDB, 5276
ER_NDB_CLUSTER_REAL_FREE_SHARE_DROP_FAILED, 5276
ER_NDB_CLUSTER_REAL_FREE_SHARE_INFO, 5276
ER_NDB_CLUSTER_SCHEMA_INFO, 5263
ER_NDB_CLUSTER_WRONG_NUMBER_OF_FUNCTION_ARGUMENTS, 5263
ER_NDB_COLUMN_DEFAULTS_DIFFER, 5272
ER_NDB_COLUMN_INFO, 5273
ER_NDB_COLUMN_SHOULD_NOT_HAVE_NATIVE_DEFAULT, 5272
ER_NDB_CONFLICT_FN_PARSE_ERROR, 5295
ER_NDB_CONFLICT_FN_SETUP_ERROR, 5295
ER_NDB_CONFLICT_GENERIC_MESSAGE, 5294
ER_NDB_COULD_NOT_GET_APPLY_STATUS_SHARE, 5297
ER_NDB_CPU_MASK_TOO_SHORT, 5310
ER_NDB_CREATE_EVENT_OPS_LOGGING_INFO, 5296
ER_NDB_CREATING_SHARE_IN_OPEN, 5275
ER_NDB_CREATING_TABLE, 5288
ER_NDB_DISCARDING_EVENT_ID_VERSION_MISMATCH, 5291
ER_NDB_DISCARDING_EVENT_NO_OBJ, 5291
ER_NDB_DISCONNECT_INFO, 5272
ER_NDB_DISCOVERED_MISSING_DB, 5288
ER_NDB_DISCOVERED_REMAINING_DB, 5289
ER_NDB_DISTRIBUTED_INFO, 5290
ER_NDB_DISTRIBUTING_ERR, 5321
ER_NDB_DISTRIBUTION_COMPLETE, 5290
ER_NDB_DUPLICATE_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS, 5309
ER_NDB_EMPTY_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS, 5308
ER_NDB_ERROR_IN_GET_AUTO_INCREMENT, 5275
ER_NDB_ERROR_IN_READAUTOINCREMENTVALUE, 5273
ER_NDB_FAILED_TO_SET_UP_TABLE, 5289
ER_NDB_FIELD_INFO, 5272
ER_NDB_FLUSHING_TABLE_INFO, 5288
ER_NDB_FOUND_UNCOMMITTED_AUTOCOMMIT, 5273
ER_NDB_GENERIC_ERROR, 5310
ER_NDB_HANDLE_TRAILING_SHARE_INFO, 5275
ER_NDB_IGNORING_UNKNOWN_EVENT, 5293
ER_NDB_ILLEGAL_VALUE_FOR_NDB_RECV_THREAD_CPU_MASK, 5276
ER_NDB_INFO_FAILED_TO_CREATE_NDBINFO, 5263
ER_NDB_INFO_FAILED_TO_INIT_NDBINFO, 5263
ER_NDB_INFO_FOUND_UNEXPECTED_FIELD_TYPE, 5263
ER_NDB_INITIALIZE_GIVEN_CLUSTER_PLUGIN_DISABLED, 5275
ER_NDB_INVALID_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS, 5308
ER_NDB_LOG_ENTRY, 5318
ER_NDB_LOG_ENTRY_WITH_PREFIX, 5318
ER_NDB_MISMATCH_IN_FRM_DISCOVERING, 5289
ER_NDB_MISSING_FRM_DISCOVERING, 5289
ER_NDB_NODEID_NOT_FIRST_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS, 5309
ER_NDB_NODE_ID_AND_MANAGEMENT_SERVER_INFO, 5272
ER_NDB_NODE_INFO, 5310
ER_NDB_NOT_WAITING_FOR_DISTRIBUTING, 5290
ER_NDB_NUMBER_OF_CHANNELS, 5321
ER_NDB_OOM_GET_NDB_BLOBS_VALUE, 5288
ER_NDB_OOM_IN_FIX_UNIQUE_INDEX_ATTR_ORDER, 5273

ER_NDB_POOL_SIZE_MUST_MATCH_NDB_CLUSTER_CONNECTION_POOL_NODEIDS, 5309
ER_NDB_QUERY_FAILED, 5271
ER_NDB_REMAINING_OPEN_TABLES, 5297
ER_NDB_REMAINING_OPEN_TABLE_INFO, 5297
ER_NDB_REPLICATION_SCHEMA_ERROR, 5138
ER_NDB_SCHEMA_DISTRIBUTION_FAILED, 5290
ER_NDB_SCHEMA_DISTRIBUTION_REPORTS_SUBSCRIBE, 5290
ER_NDB_SCHEMA_DISTRIBUTION_REPORTS_UNSUBSCRIBE, 5290
ER_NDB_SCHEMA_GENERIC_MESSAGE, 5269
ER_NDB_SERVER_ID_RESERVED_OR_TOO_LARGE, 5294
ER_NDB_SHARE_ALREADY_EXISTS, 5275
ER_NDB_SKIPPING_SETUP_TABLE, 5289
ER_NDB_SLAVE_BINLOG_MISSING_INFO_FOR_CONFLICT_DETECTION, 5273
ER_NDB_SLAVE_CANT_ALLOCATE_TABLE_SHARE, 5274
ER_NDB_SLAVE_CONFLICT_DETECTION_REQUIRES_TRANSACTION_IDS, 5273
ER_NDB_SLAVE_CONFLICT_FUNCTION_REQUIRES_ROLE, 5273
ER_NDB_SLAVE_ERROR_IN_UPDATE_CREATE_INFO, 5274
ER_NDB_SLAVE_LOGGING_EXCEPTIONS_TO, 5263
ER_NDB_SLAVE_LOW_EPOCH_RESOLUTION, 5263
ER_NDB_SLAVE_MALFORMED_EVENT_RECEIVED_ON_TABLE, 5273
ER_NDB_SLAVE_MAX_REPLICATED_EPOCH_SET_TO, 5272
ER_NDB_SLAVE_MAX_REPLICATED_EPOCH_UNKNOWN, 5272
ER_NDB_SLAVE_MISSING_DATA_FOR_TIMESTAMP_COLUMN, 5263
ER_NDB_SLAVE_PARALLEL_WORKERS, 5321
ER_NDB_SLAVE_PREVIOUS_EPOCH_NOT_COMMITTED, 5262
ER_NDB_SLAVE_SAW_ALREADY_COMMITTED_EPOCH, 5262
ER_NDB_SLAVE_SAW_EPOCH_LOWER_THAN_PREVIOUS, 5262
ER_NDB_SLAVE_SAW_EPOCH_LOWER_THAN_PREVIOUS_ON_START, 5262
ER_NDB_SLAVE_TOO_MANY_RETRIES, 5274
ER_NDB_STARTING_CONNECT_THREAD, 5310
ER_NDB_TABLES_INITIALLY_READ_ONLY_ON_RECONNECT, 5293
ER_NDB_TABLES_NOT_READY, 5240
ER_NDB_TABLE_IS_NOT_DISTRIBUTED, 5288
ER_NDB_TABLE_OPENED_READ_ONLY, 5275
ER_NDB_THREAD_TIMED_OUT, 5288
ER_NDB_TIMED_OUT_IN_DROP_TABLE, 5297
ER_NDB_TIMEOUT_WHILE_DISTRIBUTING, 5290
ER_NDB_TOO_MANY_CPUS_IN_NDB_RECV_THREAD_CPU_MASK, 5277
ER_NDB_TRAILING_SHARE_RELEASED_BY_CLOSE_CACHED_TABLES, 5275
ER_NDB_TRANS_DEPENDENCY_TRACKER_ERROR, 5295
ER_NDB_UNEXPECTED_RENAME_TYPE, 5274
ER_NDB_USING_NODEID, 5309
ER_NDB_USING_NODEID_LIST, 5309
ER_NDB_UTIL_THREAD_OOM, 5276
ER_NDB_WAITING_INFO, 5289
ER_NDB_WAITING_INFO_WITH_MAP, 5290
ER_NEED_FILE_INSTEAD_OF_DIR, 5237
ER_NEED_LOG_BIN, 5237
ER_NEED_REPREPARE, 5137
ER_NETWORK_NAMESPACES_NOT_SUPPORTED, 5593
ER_NETWORK_NAMESPACE_FILE_PATH_TOO_LONG, 5593
ER_NETWORK_NAMESPACE_NOT_ALLOWED_FOR_WILDCARD_ADDRESS, 5593
ER_NETWORK_READ_EVENT_CHECKSUM_FAILURE, 5561
ER_NET_ERROR_ON_WRITE, 5109

ER_NET_FCNTL_ERROR, 5109
ER_NET_OK_PACKET_TOO_LARGE, 5162
ER_NET_PACKETS_OUT_OF_ORDER, 5109
ER_NET_PACKET_TOO_LARGE, 5109
ER_NET_READ_ERROR, 5109
ER_NET_READ_ERROR_FROM_PIPE, 5109
ER_NET_READ_INTERRUPTED, 5109
ER_NET_UNCOMPRESS_ERROR, 5109
ER_NET_WAIT_ERROR, 5594
ER_NET_WRITE_INTERRUPTED, 5110
ER_NEW_ABORTING_CONNECTION, 5111
ER_NO, 5098
ER_NONEXISTING_GRANT, 5108
ER_NONEXISTING_PROC_GRANT, 5125
ER_NONEXISTING_TABLE_GRANT, 5109
ER_NONPOSITIVE_RADIUS, 5194
ER_NONUNIQ_TABLE, 5103
ER_NONUPDATEABLE_COLUMN, 5122
ER_NON_BOOLEAN_EXPR_FOR_CHECK_CONSTRAINT, 5206
ER_NON_DEFAULT_VALUE_FOR_GENERATED_COLUMN, 5165
ER_NON_GROUPING_FIELD_USED, 5129
ER_NON_INSERTABLE_TABLE, 5129
ER_NON_RO_SELECT_DISABLE_TIMER, 5159
ER_NON_UNIQ_ERROR, 5101
ER_NON_UPDATABLE_TABLE, 5118
ER_NORMAL_SERVER_SHUTDOWN, 5560
ER_NORMAL_SHUTDOWN, 5104
ER_NOT_ALLOWED_COMMAND, 5109
ER_NOT_FORM_FILE, 5100
ER_NOT_HINT_UPDATABLE_VARIABLE, 5186
ER_NOT_IMPLEMENTED_FOR_CARTESIAN_SRS, 5193
ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS, 5184
ER_NOT_IMPLEMENTED_FOR_PROJECTED_SRS, 5194
ER_NOT_IMPLEMENTED_GET_TABLESPACE_STATISTICS, 5363
ER_NOT_KEYFILE, 5100
ER_NOT_RIGHT_NOW, 5234
ER_NOT_SUPPORTED_AUTH_MODE, 5115
ER_NOT_SUPPORTED_YET, 5114
ER_NOT_VALID_PASSWORD, 5152
ER_NO_ACCESS_TO_NATIVE_FCT, 5178
ER_NO_BINARY_LOGGING, 5124
ER_NO_BINLOG_ERROR, 5132
ER_NO_CSV_NO_LOG_TABLES, 5236
ER_NO_DB_ERROR, 5101
ER_NO_DEFAULT, 5114
ER_NO_DEFAULT_FOR_FIELD, 5123
ER_NO_DEFAULT_FOR_VIEW_FIELD, 5126
ER_NO_FORMAT_DESCRIPTION_EVENT_BEFORE_BINLOG_STATEMENT, 5137
ER_NO_FT_MATERIALIZED_SUBQUERY, 5159
ER_NO_PARTITION_FOR_GIVEN_VALUE, 5133
ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT, 5136
ER_NO_PARTS_ERROR, 5131
ER_NO_PATH_FOR_SHARED_LIBRARY, 5567
ER_NO_PERMISSION_TO_CREATE_USER, 5113

ER_NO_REFERENCED_ROW, 5113
ER_NO_REFERENCED_ROW_2, 5128
ER_NO_SECURE_TRANSPORTS_CONFIGURED, 5170
ER_NO_SESSION_TEMP, 5214
ER_NO_SUCH_DB, 5173
ER_NO_SUCH_INDEX, 5104
ER_NO_SUCH_TABLE, 5109
ER_NO_SUCH_THREAD, 5105
ER_NO_SUCH_USER, 5128
ER_NO_SUPER_WITHOUT_USER_PLUGIN, 5254
ER_NO_SYSTEM_SCHEMA_ACCESS, 5177
ER_NO_SYSTEM_TABLESPACE_ACCESS, 5177
ER_NO_SYSTEM_TABLE_ACCESS, 5177
ER_NO_SYSTEM_TABLE_ACCESS_FOR_DICTIONARY_TABLE, 5177
ER_NO_SYSTEM_TABLE_ACCESS_FOR_SYSTEM_TABLE, 5177
ER_NO_SYSTEM_TABLE_ACCESS_FOR_TABLE, 5177
ER_NO_SYSTEM_VIEW_ACCESS, 5184
ER_NO_TABLES_USED, 5105
ER_NO_THD_NO_UUID, 5227
ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA, 5129
ER_NO_UNIQUE_LOGFILE, 5105
ER_NPIPE_CANT_CREATE, 5320
ER_NPIPE_FAILED_TO_INIT_SECURITY_DESCRIPTOR, 5262
ER_NPIPE_FAILED_TO_SET_SECURITY_DESCRIPTOR, 5262
ER_NPIPE_PIPE_ALREADY_IN_USE, 5262
ER_NULL_COLUMN_IN_INDEX, 5107
ER_NULL_IN_VALUES_LESS_THAN, 5135
ER_NUMERIC_JSON_VALUE_OUT_OF_RANGE, 5169
ER_OLD_FILE_FORMAT, 5128
ER_OLD_KEYFILE, 5100
ER_OLD_PASSWORDS_NO_MIDDLE_GROUND, 5228
ER_OLD_TEMPORALS_UPGRADED, 5157
ER_ONLY_FD_AND_RBR_EVENTS_ALLOWED_IN_BINLOG_STATEMENT, 5146
ER_ONLY_IMPLEMENTED_FOR_SRID_0_AND_4326, 5191
ER_ONLY_INTEGERS_ALLOWED, 5135
ER_ONLY_ON_RANGE_LIST_PARTITION, 5132
ER_OOM, 5234
ER_OOM_SAVE_GTIDS, 5560
ER_OPEN_AS_READONLY, 5100
ER_OPEN_ROLE_TABLES, 5175
ER_OPERAND_COLUMNS, 5115
ER_OPTION_PREVENTS_STATEMENT, 5118
ER_OPT_WRONG_TREE, 5220
ER_OUTOFMEMORY, 5101
ER_OUT_OF_RESOURCES, 5101
ER_OUT_OF_SORTMEMORY, 5101
ER_PAGE_TRACKING_CANNOT_PURGE, 5205
ER_PAGE_TRACKING_NOT_STARTED, 5205
ER_PAGE_TRACKING_RANGE_NOT_TRACKED, 5205
ER_PARSER_TRACE, 5220
ER_PARSE_ERROR, 5103
ER_PARSE_ERROR_IN_DIGEST_FN, 5190
ER_PARSING_VIEW, 5226
ER_PARTIAL_REVOKES_EXIST, 5215

ER_PARTIAL_REVOKE_AND_DB_GRANT_BOTH_EXISTS, 5216
ER_PARTITIONS_MUST_BE_DEFINED_ERROR, 5130
ER_PARTITION_CLAUSE_ON_NONPARTITIONED, 5147
ER_PARTITION_COLUMN_LIST_ERROR, 5140
ER_PARTITION_CONST_DOMAIN_ERROR, 5135
ER_PARTITION_ENTRY_ERROR, 5131
ER_PARTITION_EXCHANGE_DIFFERENT_OPTION, 5146
ER_PARTITION_EXCHANGE_FOREIGN_KEY, 5146
ER_PARTITION_EXCHANGE_PART_TABLE, 5146
ER_PARTITION_EXCHANGE_TEMP_TABLE, 5146
ER_PARTITION_FIELDS_TOO_LONG, 5140
ER_PARTITION_FUNCTION_FAILURE, 5132
ER_PARTITION_FUNCTION_IS_NOT_ALLOWED, 5135
ER_PARTITION_FUNC_NOT_ALLOWED_ERROR, 5130
ER_PARTITION_HANDLER_ADMIN_MSG, 5340
ER_PARTITION_INSTEAD_OF_SUBPARTITION, 5146
ER_PARTITION_MAXVALUE_ERROR, 5130
ER_PARTITION_MERGE_ERROR, 5135
ER_PARTITION_MGMT_ON_NONPARTITIONED, 5131
ER_PARTITION_MOVE_CREATED_DUPLICATE_ROW_PLEASE_FIX, 5320
ER_PARTITION_NAME, 5138
ER_PARTITION_NOT_DEFINED_ERROR, 5131
ER_PARTITION_NO_TEMPORARY, 5134
ER_PARTITION_REQUIRES_VALUES_ERROR, 5130
ER_PARTITION_WRONG_NO_PART_ERROR, 5130
ER_PARTITION_WRONG_NO_SUBPART_ERROR, 5130
ER_PARTITION_WRONG_VALUES_ERROR, 5130
ER_PART_EXPR_TOO_LONG, 5184
ER_PASSWORD_ANONYMOUS_USER, 5108
ER_PASSWORD_CANNOT_BE_RETAINED_ON_PLUGIN_CHANGE, 5215
ER_PASSWORD_EXPIRATION_NOT_SUPPORTED_BY_AUTH_METHOD, 5190
ER_PASSWORD_EXPIRE_ANONYMOUS_USER, 5159
ER_PASSWORD_FORMAT, 5153
ER_PASSWORD_NOT_ALLOWED, 5108
ER_PASSWORD_NO_MATCH, 5108
ER_PATH_IN_DATADIR, 5185
ER_PATH_LENGTH, 5142
ER_PERFSCHEMA_COMPONENTS_INFRASTRUCTURE_BOOTSTRAP, 5326
ER_PERFSCHEMA_COMPONENTS_INFRASTRUCTURE_SHUTDOWN, 5326
ER_PERFSCHEMA_INIT_FAILED, 5238
ER_PERFSCHEMA_TABLES_INIT_FAILED, 5327
ER_PERSISTENT_PRIVILEGES_BOOTSTRAP, 5327
ER_PERSIST_ONLY_ACCESS_DENIED_ERROR, 5185
ER_PERSIST_OPTION_HOST_TRUNCATED, 5594
ER_PERSIST_OPTION_STATUS, 5363
ER_PERSIST_OPTION_USER_TRUNCATED, 5594
ER_PER_CHANNEL_RPL_FILTER_CONF_FOR_GRP_RPL, 5336
ER_PFS_MALLOC_ARRAY_OOM, 5338
ER_PFS_MALLOC_ARRAY_OVERFLOW, 5338
ER_PFS_NOTIFICATION_FUNCTION_REGISTER_FAILED, 5331
ER_PID_FILEPATH_LOCATIONS_INACCESSIBLE, 5568
ER_PID_FILE_PRIV_DIRECTORY_INSECURE, 5428
ER_PK_INDEX_CANT_BE_INVISIBLE, 5175
ER_PLUGGABLE_PROTOCOL_COMMAND_NOT_SUPPORTED, 5167

ER_PLUGIN_BAD_OPTIONS, 5304
ER_PLUGIN_CANNOT_BE_UNINSTALLED, 5157
ER_PLUGIN_CANT_LOAD, 5303
ER_PLUGIN_CANT_OPEN_PLUGIN_TABLE, 5302
ER_PLUGIN_CANT_SET_PERSISTENT_OPTIONS, 5304
ER_PLUGIN_COMMON_FAILED_TO_OPEN_FILTER_TABLES, 5362
ER_PLUGIN_COMMON_FAILED_TO_OPEN_TABLE, 5362
ER_PLUGIN_DELETE_BUILTIN, 5138
ER_PLUGIN_DID_NOT_DEINITIALIZE_THREADS, 5272
ER_PLUGIN_DISABLED, 5304
ER_PLUGIN_FAILED_DEINITIALIZATION, 5302
ER_PLUGIN_FAILED_TO_OPEN_TABLE, 5565
ER_PLUGIN_FAILED_TO_OPEN_TABLES, 5565
ER_PLUGIN_FORCING_SHUTDOWN, 5303
ER_PLUGIN_HAS_CONFLICTING_SYSTEM_VARIABLES, 5304
ER_PLUGIN_HAS_NONZERO_REFCOUNT_AFTER_DEINITIALIZATION, 5302
ER_PLUGIN_HAS_NONZERO_REFCOUNT_AFTER_SHUTDOWN, 5303
ER_PLUGIN_INIT_FAILED, 5241
ER_PLUGIN_INSTALL_ERROR, 5574
ER_PLUGIN_IS_NOT_LOADED, 5132
ER_PLUGIN_IS_PERMANENT, 5143
ER_PLUGIN_LOAD_OPTIONS_IGNORED, 5603
ER_PLUGIN_LOAD_PARAMETER_TOO_LONG, 5303
ER_PLUGIN_NOT_EARLY, 5209
ER_PLUGIN_NOT_EARLY_DUP, 5594
ER_PLUGIN_NO_INSTALL, 5145
ER_PLUGIN_NO_INSTALL_DUP, 5594
ER_PLUGIN_NO_UNINSTALL, 5145
ER_PLUGIN_OOM, 5303
ER_PLUGIN_PARSING_OPTIONS_FAILED, 5304
ER_PLUGIN_REGISTRATION_FAILED, 5302
ER_PLUGIN_SHUTTING_DOWN_PLUGIN, 5302
ER_PLUGIN_UNINSTALL_ERROR, 5574
ER_PLUGIN_UNKNOWN_VARIABLE_TYPE, 5303
ER_PLUGIN_VARIABLE_MISSING_NAME, 5303
ER_PLUGIN_VARIABLE_NOT_ALLOCATED_THREAD_LOCAL, 5303
ER_PLUGIN_VARIABLE_SET_READ_ONLY, 5303
ER_POINTLESS_WITHOUT_SLOWLOG, 5228
ER_POLYGON_TOO_LARGE, 5196
ER_PREPARE_FOR_SECONDARY_ENGINE, 5588
ER_PREVENTS_VARIABLE_WITHOUT_RBR, 5165
ER_PRIMARY_CANT_HAVE_NULL, 5110
ER_PRIVILEGE_SYSTEM_INIT_FAILED, 5327
ER_PROCACCESS_DENIED_ERROR, 5123
ER_PROC_AUTO_GRANT_FAIL, 5125
ER_PROC_AUTO_REVOKE_FAIL, 5125
ER_PROTOCOL_COMPRESSION_RESET_LOG, 5601
ER_PS_MANY_PARAM, 5124
ER_PS_NO_RECURSION, 5127
ER_QUERY_CACHE_DISABLED, 5140
ER_QUERY_INTERRUPTED, 5120
ER_QUERY_ON_FOREIGN_DATA_SOURCE, 5126
ER_QUERY_TIMEOUT, 5159
ER_RANGE_NOT_INCREASING_ERROR, 5130

ER_READING_TABLE_FAILED, 5242
ER_READY, 5104
ER_READ_LOG_EVENT_FAILED, 5335
ER_READ_ONLY_MODE, 5153
ER_READ_ONLY_TRANSACTION, 5112
ER_REALLY_RUN_AS_ROOT, 5233
ER_RECORD_FILE_FULL, 5106
ER_RECOVERING_TABLE, 5305
ER_REFERENCED_TRG_DOES_NOT_EXIST, 5158
ER_REGEXP_BAD_ESCAPE_SEQUENCE, 5192
ER_REGEXP_BAD_INTERVAL, 5192
ER_REGEXP_BUFFER_OVERFLOW, 5191
ER_REGEXP_ERROR, 5108
ER_REGEXP_ILLEGAL_ARGUMENT, 5191
ER_REGEXP_INDEX_OUTOFBOUNDS_ERROR, 5191
ER_REGEXP_INTERNAL_ERROR, 5192
ER_REGEXP_INVALID_BACK_REF, 5192
ER_REGEXP_INVALID_CAPTURE_GROUP_NAME, 5214
ER_REGEXP_INVALID_FLAG, 5215
ER_REGEXP_INVALID_RANGE, 5193
ER_REGEXP_LOOK_BEHIND_LIMIT, 5192
ER_REGEXP_MAX_LT_MIN, 5192
ER_REGEXP_MISMATCHED_PAREN, 5192
ER_REGEXP_MISSING_CLOSE_BRACKET, 5192
ER_REGEXP_PATTERN_TOO_BIG, 5193
ER_REGEXP_RULE_SYNTAX, 5192
ER_REGEXP_STACK_OVERFLOW, 5193
ER_REGEXP_STRING_NOT_TERMINATED, 5191
ER_REGEXP_TIME_OUT, 5193
ER_REGEXP_UNIMPLEMENTED, 5192
ER_RELAY_LOG_FAIL, 5123
ER_RELAY_LOG_INIT, 5124
ER_RELAY_LOG_SPACE_LIMIT_DISABLED, 5570
ER_REMOVED_SPACES, 5129
ER_RENAMED_NAME, 5139
ER_RENAME_ROLE, 5175
ER_REORG_HASH_ONLY_ON_SAME_NO, 5132
ER_REORG_NO_PARAM_ERROR, 5132
ER_REORG_OUTSIDE_RANGE, 5132
ER_REORG_PARTITION_NOT_EXIST, 5132
ER_REPLACE_INACCESSIBLE_ROWS, 5162
ER_REQUIRES_PRIMARY_KEY, 5110
ER_RESERVED_SYNTAX, 5124
ER_RESERVED_TABLESPACE_NAME, 5195
ER_RESET_MASTER_TO_VALUE_OUT_OF_RANGE, 5178
ER_RESIGNAL_WITHOUT_ACTIVE_HANDLER, 5139
ER_RESOURCE_GROUP_BIND_FAILED, 5189
ER_RESOURCE_GROUP_BUSY, 5188
ER_RESOURCE_GROUP_DISABLED, 5188
ER_RESOURCE_GROUP_EXISTS, 5188
ER_RESOURCE_GROUP_IS_DISABLED, 5330
ER_RESOURCE_GROUP_METADATA_UPDATE_SKIPPED, 5329
ER_RESOURCE_GROUP_NOT_EXISTS, 5188
ER_RESOURCE_GROUP_POST_INIT_FAILED, 5326

ER_RESOURCE_GROUP_SUBSYSTEM_INIT_FAILED, 5326
ER_RESOURCE_GROUP_VALIDATION_FAILED, 5330
ER_RESTART_RECEIVED_INFO, 5343
ER_RESTART_SERVER_FAILED, 5194
ER_RES_GRP_FAILED_DETERMINE_CPU_COUNT, 5332
ER_RES_GRP_FAILED_TO_DETERMINE_NICE_CAPABILITY, 5331
ER_RES_GRP_FAILED_TO_GET_THREAD_HANDLE, 5331
ER_RES_GRP_FEATURE_NOT_AVAILABLE, 5332
ER_RES_GRP_GET_THREAD_PRIO_NOT_SUPPORTED, 5331
ER_RES_GRP_INVALID_THREAD_PRIORITY, 5332
ER_RES_GRP_INVALID_VCPU_ID, 5342
ER_RES_GRP_INVALID_VCPU_RANGE, 5342
ER_RES_GRP_SET_THREAD_PRIORITY_FAILED, 5331
ER_RES_GRP_SET_THR_AFFINITY_FAILED, 5331
ER_RES_GRP_SET_THR_AFFINITY_TO_CPUS_FAILED, 5331
ER_RES_GRP_SOLARIS_PROCESSOR_AFFINITY_FAILED, 5332
ER_RES_GRP_SOLARIS_PROCESSOR_BIND_TO_CPUID_FAILED, 5332
ER_RES_GRP_SOLARIS_PROCESSOR_BIND_TO_THREAD_FAILED, 5332
ER_RES_GRP_THD_UNBIND_FROM_CPU_FAILED, 5331
ER_RETRYING_REPAIR_WITHOUT_QUICK, 5304
ER_RETRYING_REPAIR_WITH_KEYCACHE, 5304
ER_REVOKE_GRANTS, 5116
ER_REWRITER_LOAD_FAILED, 5364
ER_REWRITER_OOM, 5364
ER_REWRITER_QUERY_ERROR_MSG, 5360
ER_REWRITER_QUERY_FAILED, 5360
ER_REWRITER_READ_FAILED, 5364
ER_REWRITER_TABLE_MALFORMED_ERROR, 5364
ER_ROLE_NOT_GRANTED, 5175
ER_ROW_DATA_TOO_BIG_TO_WRITE_IN_BINLOG, 5335
ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET, 5147
ER_ROW_DOES_NOT_MATCH_PARTITION, 5146
ER_ROW_IN_WRONG_PARTITION, 5155
ER_ROW_IN_WRONG_PARTITION_PLEASE_REPAIR, 5245
ER_ROW_IS_REFERENCED, 5113
ER_ROW_IS_REFERENCED_2, 5128
ER_ROW_SINGLE_PARTITION_FIELD_ERROR, 5140
ER_RPL_BINLOG_FILTERS_OOM, 5231
ER_RPL_BINLOG_MASTER_SENDS_HEARTBEAT, 5270
ER_RPL_BINLOG_MASTER_USES_CHECKSUM_AND_SLAVE_CANT, 5271
ER_RPL_BINLOG_RELAY_DELEGATES_INIT_FAILED, 5241
ER_RPL_BINLOG_SKIPPING_REMAINING_HEARTBEAT_INFO, 5271
ER_RPL_BINLOG_STARTING_DUMP, 5270
ER_RPL_BINLOG_STORAGE_DELEGATES_INIT_FAILED, 5241
ER_RPL_BINLOG_TRANSMIT_DELEGATES_INIT_FAILED, 5241
ER_RPL_CANT_ADD_DO_TABLE, 5235
ER_RPL_CANT_ADD_IGNORE_TABLE, 5235
ER_RPL_CANT_CREATE_SLAVE_THREAD, 5280
ER_RPL_CANT_FIND_FOLLOWUP_FILE, 5284
ER_RPL_CANT_HAVE_SAME_BASENAME, 5323
ER_RPL_CANT_INITIALIZE_GTID_SETS_IN_RLI_INIT_INFO, 5287
ER_RPL_CANT_MAKE_PATHS, 5237
ER_RPL_CANT_OPEN_INFO_TABLE, 5264
ER_RPL_CANT_OPEN_LOG_IN_RLI_INIT_INFO, 5288

ER_RPL_CANT_SCAN_INFO_TABLE, 5264
ER_RPL_CANT_STOP_SLAVE_WHILE_LOCKED_BACKUP, 5211
ER_RPL_CHANGING_RELAY_LOG_INFO_REPO_TYPE_FAILED_DUE_TO_GAPS, 5265
ER_RPL_CHANNELS_REQUIRE_NON_ZERO_SERVER_ID, 5265
ER_RPL_CHANNELS_REQUIRE_TABLES_AS_INFO_REPOSITORIES, 5264
ER_RPL_CORRUPTED_INFO_TABLE, 5264
ER_RPL_CORRUPTED_KEYS_IN_INFO_TABLE, 5264
ER_RPL_ENCRYPTION_CANNOT_ROTATE_BINLOG_MASTER_KEY, 5205
ER_RPL_ENCRYPTION_FAILED_TO_ENCRYPT, 5205
ER_RPL_ENCRYPTION_FAILED_TO_FETCH_KEY, 5203
ER_RPL_ENCRYPTION_FAILED_TO_GENERATE_KEY, 5203
ER_RPL_ENCRYPTION_FAILED_TO_REMOVE_KEY, 5203
ER_RPL_ENCRYPTION_FAILED_TO_ROTATE_LOGS, 5203
ER_RPL_ENCRYPTION_FAILED_TO_STORE_KEY, 5203
ER_RPL_ENCRYPTION_HEADER_ERROR, 5203
ER_RPL_ENCRYPTION_KEYRING_INVALID_KEY, 5203
ER_RPL_ENCRYPTION_KEY_EXISTS_UNEXPECTED, 5203
ER_RPL_ENCRYPTION_KEY_NOT_FOUND, 5203
ER_RPL_ENCRYPTION_MASTER_KEY_RECOVERY_FAILED, 5204
ER_RPL_ENCRYPTION_UNABLE_TO_CHANGE_OPTION, 5204
ER_RPL_ERROR_CHANGING_MASTER_INFO_REPO_TYPE, 5265
ER_RPL_ERROR_CHANGING_RELAY_LOG_INFO_REPO_TYPE, 5265
ER_RPL_ERROR_CHECKING_REPOSITORY, 5266
ER_RPL_ERROR_CREATING_MASTER_INFO, 5265
ER_RPL_ERROR_CREATING_RELAY_LOG_INFO, 5265
ER_RPL_ERROR_LOOKING_FOR_LOG, 5283
ER_RPL_ERROR_READING_MASTER_CONFIGURATION, 5302
ER_RPL_ERROR_READING_SLAVE_WORKER_CONFIGURATION, 5300
ER_RPL_ERROR_WRITING_MASTER_CONFIGURATION, 5302
ER_RPL_ERROR_WRITING_RELAY_LOG_CONFIGURATION, 5288
ER_RPL_ERROR_WRITING_SLAVE_WORKER_CONFIGURATION, 5301
ER_RPL_FAILED_TO_CREATE_CACHE_FOR_INFO_FILE, 5267
ER_RPL_FAILED_TO_CREATE_NEW_INFO_FILE, 5266
ER_RPL_FAILED_TO_DELETE_FROM_SLAVE_WORKERS_INFO_REPOSITORY, 5265
ER_RPL_FAILED_TO_OPEN_INFO_FILE, 5267
ER_RPL_FAILED_TO_OPEN_RELAY_LOG, 5301
ER_RPL_FAILED_TO_RESET_STATE_IN_SLAVE_INFO_REPOSITORY, 5266
ER_RPL_FAILED_TO_STAT_LOG_IN_INDEX, 5287
ER_RPL_FILTERS_NOT_ATTACHED_TO_CHANNEL, 5336
ER_RPL_FILTER_ADD_WILD_DO_TABLE_FAILED, 5327
ER_RPL_FILTER_ADD_WILD_IGNORE_TABLE_FAILED, 5327
ER_RPL_GTID_LOG_EVENT_IN_STREAM, 5268
ER_RPL_GTID_MEMORY_FINALLY_AVAILABLE, 5267
ER_RPL_GTID_MODE_REQUIRES_ENFORCE_GTID_CONSISTENCY_ON, 5323
ER_RPL_GTID_TABLE_CANNOT_OPEN, 5221
ER_RPL_GTID_UNSAFE_ALTER_ADD_COL_WITH_DEFAULT_EXPRESSION, 5573
ER_RPL_GTID_UNSAFE_STMT_CREATE_SELECT, 5559
ER_RPL_GTID_UNSAFE_STMT_ON_NON_TRANS_TABLE, 5559
ER_RPL_GTID_UNSAFE_STMT_ON_TEMPORARY_TABLE, 5559
ER_RPL_INCOMPATIBLE_DECIMAL_IN_RBR, 5269
ER_RPL_INCONSISTENT_SEQUENCE_NO_IN_TRX, 5264
ER_RPL_INCONSISTENT_TIMESTAMPS_IN_TRX, 5264
ER_RPL_INFINITY_DENIED, 5240
ER_RPL_INFINITY_IGNORED, 5240

ER_RPL_INFO_DATA_TOO_LONG, 5146
ER_RPL_IO_THREAD_KILLED, 5336
ER_RPL_LOG_ENTRY_EXCEEDS_SLAVE_MAX_ALLOWED_PACKET, 5283
ER_RPL_LOG_NOT_FOUND_WHILE_COUNTING_RELAY_LOG_SPACE, 5287
ER_RPL_MTS_AUTOMATIC_RECOVERY_FAILED, 5280
ER_RPL_MTS_CHECKPOINT_PERIOD_DIFFERS_FROM_CNT, 5284
ER_RPL_MTS_GROUP_RECOVERY_RELAY_LOG_INFO, 5284
ER_RPL_MTS_GROUP_RECOVERY_RELAY_LOG_INFO_FOR_WORKER, 5283
ER_RPL_MTS_RECOVERY_CANT_OPEN_RELAY_LOG, 5280
ER_RPL_MTS_RECOVERY_COMPLETE, 5282
ER_RPL_MTS_RECOVERY_FAILED_TO_START_COORDINATOR, 5280
ER_RPL_MTS_RECOVERY_STARTING_COORDINATOR, 5280
ER_RPL_MTS_RECOVERY_SUCCESSFUL, 5280
ER_RPL_MTS_SLAVE_COORDINATOR_HAS_WAITED, 5301
ER_RPL_MTS_STATISTICS, 5282
ER_RPL_MULTISOURCE_REQUIRES_TABLE_TYPE_REPOSITORIES, 5266
ER_RPL_OPEN_INDEX_FILE_FAILED, 5287
ER_RPL_PLEASE_USE_OPTION_RELAY_LOG, 5287
ER_RPL_PLUGIN_FUNCTION_FAILED, 5241
ER_RPL_RECOVERY_ERROR, 5279
ER_RPL_RECOVERY_ERROR_FREEING_IO_CACHE, 5279
ER_RPL_RECOVERY_ERROR_READ_RELAY_LOG, 5279
ER_RPL_RECOVERY_FILE_MASTER_POS_INFO, 5279
ER_RPL_RECOVERY_IO_ERROR_READING_RELAY_LOG_INDEX, 5279
ER_RPL_RECOVERY_NO_ROTATE_EVENT_FROM_MASTER, 5279
ER_RPL_RECOVERY_REPLICATE_SAME_SERVER_ID_REQUIRES_POSITION, 5279
ER_RPL_RECOVERY_SKIPPED_GROUP_REPLICATION_CHANNEL, 5279
ER_RPL_RELAY_LOG_INDEX_NEEDS_FILE_NOT_DIRECTORY, 5287
ER_RPL_RELAY_LOG_NEEDS_FILE_NOT_DIRECTORY, 5287
ER_RPL_REPO_HAS_GAPS, 5223
ER_RPL_REPO_SHOULD_BE_TABLE, 5265
ER_RPL_REWRITEDB_EMPTY_FROM, 5236
ER_RPL_REWRITEDB_EMPTY_TO, 5236
ER_RPL_REWRITEDB_MISSING_ARROW, 5236
ER_RPL_RLI_INIT_INFO_MSG, 5340
ER_RPL_SERVER_ID_MISSING, 5280
ER_RPL_SLAVE_ADDITIONAL_ERROR_INFO_FROM_DA, 5285
ER_RPL_SLAVE_AUTO_POSITION_IS_1_AND_GTID_MODE_IS_OFF, 5278
ER_RPL_SLAVE_CANT_FLUSH_MASTER_INFO_FILE, 5281
ER_RPL_SLAVE_CANT_INITIALIZE_SLAVE_WORKER, 5283
ER_RPL_SLAVE_CANT_INIT_RELAY_LOG_POSITION, 5282
ER_RPL_SLAVE_CANT_START_SLAVE_FOR_CHANNEL, 5278
ER_RPL_SLAVE_CANT_STOP_SLAVE_FOR_CHANNEL, 5279
ER_RPL_SLAVE_CANT_USE_CHARSET, 5286
ER_RPL_SLAVE_CONNECTED_TO_MASTER_REPLICATION_RESUMED, 5286
ER_RPL_SLAVE_CONNECTED_TO_MASTER_REPLICATION_STARTED, 5282
ER_RPL_SLAVE_COULD_NOT_CREATE_CHANNEL_LIST, 5266
ER_RPL_SLAVE_DUMP_THREAD_KILLED_BY_MASTER, 5282
ER_RPL_SLAVE_ERROR_INFO_FROM_DA, 5285
ER_RPL_SLAVE_ERROR_LOADING_USER_DEFINED_LIBRARY, 5285
ER_RPL_SLAVE_ERROR_READING_FROM_SERVER, 5282
ER_RPL_SLAVE_ERROR_READING_RELAY_LOG_EVENTS, 5286
ER_RPL_SLAVE_ERROR_REQUESTING_BINLOG_DUMP, 5283
ER_RPL_SLAVE_ERROR_RETRYING, 5281

ER_RPL_SLAVE_ERROR_RUNNING_QUERY, 5285
ER_RPL_SLAVE_FAILED_TO_CREATE_CHANNEL_FROM_MASTER_INFO, 5266
ER_RPL_SLAVE_FAILED_TO_CREATE_OR_RECOVER_INFO_REPOSITORIES, 5278
ER_RPL_SLAVE_FAILED_TO_INIT_A_MASTER_INFO_STRUCTURE, 5266
ER_RPL_SLAVE_FAILED_TO_INIT_MASTER_INFO_STRUCTURE, 5266
ER_RPL_SLAVE_FAILED_TO_INIT_PARTITIONS_HASH, 5284
ER_RPL_SLAVE_FILTER_CREATE_FAILED, 5328
ER_RPL_SLAVE_FLUSH_RELAY_LOGS_NOT_ALLOWED, 5321
ER_RPL_SLAVE_FORCING_TO_RECONNECT_IO_THREAD, 5283
ER_RPL_SLAVE_GENERIC_MESSAGE, 5266
ER_RPL_SLAVE_GLOBAL_FILTERS_COPY_FAILED, 5329
ER_RPL_SLAVE_INCORRECT_CHANNEL, 5322
ER_RPL_SLAVE_INSECURE_CHANGE_MASTER, 5321
ER_RPL_SLAVE_IO_THREAD_ABORTED_WAITING_FOR_RELAY_LOG_SPACE, 5283
ER_RPL_SLAVE_IO_THREAD_CANT_REGISTER_ON_MASTER, 5282
ER_RPL_SLAVE_IO_THREAD_DETECTED_UNEXPECTED_EVENT_SEQUENCE, 5286
ER_RPL_SLAVE_IO_THREAD_EXITING, 5283
ER_RPL_SLAVE_IO_THREAD_KILLED, 5282
ER_RPL_SLAVE_IO_THREAD_STOP_CMD_EXEC_TIMEOUT, 5559
ER_RPL_SLAVE_IO_THREAD_WAS_KILLED, 5280
ER_RPL_SLAVE_MASTER_UUID_HAS_CHANGED, 5281
ER_RPL_SLAVE_NDB_TABLES_NOT_AVAILABLE, 5284
ER_RPL_SLAVE_NEW_MASTER_INFO_NEEDS_REPOS_TYPE_OTHER_THAN_FILE, 5287
ER_RPL_SLAVE_NEXT_LOG_IS_ACTIVE, 5286
ER_RPL_SLAVE_NEXT_LOG_IS_INACTIVE, 5286
ER_RPL_SLAVE_QUEUE_EVENT_FAILED_INVALID_CONFIGURATION, 5285
ER_RPL_SLAVE_READ_INVALID_EVENT_FROM_MASTER, 5285
ER_RPL_SLAVE_REPORT_HOST_TOO_LONG, 5281
ER_RPL_SLAVE_REPORT_PASSWORD_TOO_LONG, 5281
ER_RPL_SLAVE_REPORT_USER_TOO_LONG, 5281
ER_RPL_SLAVE_RESET_FILTER_OPTIONS, 5329
ER_RPL_SLAVE_SECONDS_BEHIND_MASTER_DUBIOUS, 5281
ER_RPL_SLAVE_SKIP_COUNTER_EXECUTED, 5285
ER_RPL_SLAVE_SQL_THREAD_EXITING, 5285
ER_RPL_SLAVE_SQL_THREAD_IO_ERROR_READING_EVENT, 5286
ER_RPL_SLAVE_SQL_THREAD_STARTING, 5284
ER_RPL_SLAVE_SQL_THREAD_STARTING_WITH_PRIVILEGE_CHECKS, 5602
ER_RPL_SLAVE_SQL_THREAD_STOP_CMD_EXEC_TIMEOUT, 5559
ER_RPL_SLAVE_STOPPING_AS_MASTER_OOM, 5283
ER_RPL_SLAVE_USES_CHECKSUM_AND_MASTER_PRE_50, 5281
ER_RPL_SLAVE_WORKER_THREAD_CREATION_FAILED, 5284
ER_RPL_SLAVE_WORKER_THREAD_CREATION_FAILED_WITH_ERRNO, 5284
ER_RPL_SSL_INFO_IN_MASTER_INFO_IGNORED, 5302
ER_RPL_TIMESTAMPS_RETURNED_TO_NORMAL, 5328
ER_RPL_TRX_DELEGATES_INIT_FAILED, 5241
ER_RPL_UNEXPECTED_BEGIN_IN_STREAM, 5268
ER_RPL_UNEXPECTED_COMMIT_ROLLBACK_OR_XID_LOG_EVENT_IN_STREAM, 5268
ER_RPL_UNEXPECTED_XA_ROLLBACK_IN_STREAM, 5269
ER_RPL_UNSUPPORTED_UNIGNORABLE_EVENT_IN_STREAM, 5268
ER_RPL_WORKER_CANT_FIND_NEXT_RELAY_LOG, 5301
ER_RPL_WORKER_CANT_READ_RELAY_LOG, 5301
ER_RPL_WORKER_ID_IS, 5264
ER_RPL_ZOMBIE_ENCOUNTERED, 5221
ER_RUNNING_APPLIER_PREVENTS_SWITCH_GLOBAL_BINLOG_FORMAT, 5198

ER_RUN_HOOK_ERROR, 5165
ER_SAME_NAME_PARTITION, 5132
ER_SAME_NAME_PARTITION_FIELD, 5140
ER_SCHEDULER_KILLING, 5225
ER_SCHEDULER_STARTED, 5225
ER_SCHEDULER_STOPPED, 5225
ER_SCHEDULER_STOPPING_FAILED_TO_CREATE_WORKER, 5225
ER_SCHEDULER_STOPPING_FAILED_TO_GET_EVENT, 5225
ER_SCHEDULER_WAITING, 5225
ER_SCHEMA_DIR_CREATE_FAILED, 5191
ER_SCHEMA_DIR_EXISTS, 5190
ER_SCHEMA_DIR_MISSING, 5191
ER_SCHEMA_DIR_UNKNOWN, 5191
ER_SDI_OPERATION_FAILED, 5189
ER_SECONDARY_ENGINE, 5214
ER_SECONDARY_ENGINE_DDL, 5214
ER_SECONDARY_ENGINE_PLUGIN, 5213
ER_SECOND_PASSWORD_CANNOT_BE_EMPTY, 5213
ER_SECURE_AUTH_VALUE_UNSUPPORTED, 5228
ER_SECURE_TRANSPORT_REQUIRED, 5170
ER_SEC_FILE_PRIV_ARGUMENT_TOO_LONG, 5230
ER_SEC_FILE_PRIV_CANT_ACCESS_DIR, 5230
ER_SEC_FILE_PRIV_CANT_STAT, 5230
ER_SEC_FILE_PRIV_DIRECTORY_INSECURE, 5230
ER_SEC_FILE_PRIV_DIRECTORY_PERMISSIONS, 5230
ER_SEC_FILE_PRIV_EMPTY, 5230
ER_SEC_FILE_PRIV_IGNORED, 5230
ER_SEC_FILE_PRIV_NULL, 5230
ER_SELECT_REDUCED, 5115
ER_SEMISYNC_ADD_ACK_TO_SLOT, 5351
ER_SEMISYNC_BINLOG_REPLY_IS_AHEAD, 5350
ER_SEMISYNC_BINLOG_WRITE_OUT_OF_ORDER, 5348
ER_SEMISYNC_CLEARED_ACTIVE_TRANSACTION_TILL_POS, 5349
ER_SEMISYNC_CLEARED_ALL_ACTIVE_TRANSACTION_NODES, 5348
ER_SEMISYNC_DISABLED_ON_MASTER, 5349
ER_SEMISYNC_EXECUTION_FAILED_ON_MASTER, 5354
ER_SEMISYNC_FAILED_REGISTER_SLAVE_TO_RECEIVER, 5352
ER_SEMISYNC_FAILED_TO_ALLOCATE_TRX_NODE, 5348
ER_SEMISYNC_FAILED_TO_INSERT_TRX_NODE, 5351
ER_SEMISYNC_FAILED_TO_START_ACK_RECEIVER_THD, 5352
ER_SEMISYNC_FAILED_TO_STOP_ACK_RECEIVER_THD, 5354
ER_SEMISYNC_FAILED_TO_WAIT_ON_DUMP_SOCKET, 5352
ER_SEMISYNC_FORCED_SHUTDOWN, 5350
ER_SEMISYNC_FUNCTION_CALLED_TWICE, 5349
ER_SEMISYNC_INIT_WAIT_POS, 5350
ER_SEMISYNC_INSERT_LOG_INFO_IN_ENTRY, 5348
ER_SEMISYNC_MASTER_FAILED_ON_NET_FLUSH, 5351
ER_SEMISYNC_MASTER_GOT_REPLY_AT_POS, 5350
ER_SEMISYNC_MASTER_OOM, 5349
ER_SEMISYNC_MASTER_SIGNAL_ALL_WAITING_THREADS, 5350
ER_SEMISYNC_MASTER_TRX_WAIT_POS, 5350
ER_SEMISYNC_MISSING_MAGIC_NO_FOR_SEMISYNC_PKT, 5353
ER_SEMISYNC_MOVE_BACK_WAIT_POS, 5350
ER_SEMISYNC_NOT_SUPPORTED_BY_MASTER, 5354

ER_SEMISYNC_NO_SPACE_IN_THE_PKT, 5351
ER_SEMISYNC_PROBE_LOG_INFO_IN_ENTRY, 5348
ER_SEMISYNC_RECEIVED_ACK_IS_SMALLER, 5351
ER_SEMISYNC_REPLY_BINLOG_FILE_TOO_LARGE, 5349
ER_SEMISYNC_REPLY_MAGIC_NO_ERROR, 5349
ER_SEMISYNC_REPLY_PKT_LENGTH_TOO_SMALL, 5349
ER_SEMISYNC_RPL_ENABLED_ON_MASTER, 5349
ER_SEMISYNC_RPL_INIT_FOR_TRX, 5348
ER_SEMISYNC_RPL_SWITCHED_OFF, 5351
ER_SEMISYNC_RPL_SWITCHED_ON, 5351
ER_SEMISYNC_SERVER_REPLY, 5349
ER_SEMISYNC_SLAVE_NET_FLUSH_REPLY_FAILED, 5354
ER_SEMISYNC_SLAVE_REPLY, 5353
ER_SEMISYNC_SLAVE_REPLY_WITH_BINLOG_INFO, 5353
ER_SEMISYNC_SLAVE_SEND_REPLY_FAILED, 5354
ER_SEMISYNC_SLAVE_SET_FAILED, 5354
ER_SEMISYNC_SLAVE_START, 5353
ER_SEMISYNC_SOCKET_FD_TOO_LARGE, 5353
ER_SEMISYNC_STARTING_ACK_RECEIVER_THD, 5352
ER_SEMISYNC_START_BINLOG_DUMP_TO_SLAVE, 5352
ER_SEMISYNC_STOPPING_ACK_RECEIVER_THREAD, 5352
ER_SEMISYNC_STOP_BINLOG_DUMP_TO_SLAVE, 5352
ER_SEMISYNC_SYNC_HEADER_UPDATE_INFO, 5351
ER_SEMISYNC_TRACE_ENTER_FUNC, 5347
ER_SEMISYNC_TRACE_EXIT, 5348
ER_SEMISYNC_TRACE_EXIT_WITH_BOOL_EXIT_CODE, 5348
ER_SEMISYNC_TRACE_EXIT_WITH_INT_EXIT_CODE, 5348
ER_SEMISYNC_TRX_SKIPPED_AT_POS, 5351
ER_SEMISYNC_UNREGISTERED_REPLICATOR, 5353
ER_SEMISYNC_UNREGISTER_BINLOG_STORAGE_OBSERVER_FAILED, 5353
ER_SEMISYNC_UNREGISTER_BINLOG_TRANSMIT_OBSERVER_FAILED, 5353
ER_SEMISYNC_UNREGISTER_TRX_OBSERVER_FAILED, 5353
ER_SEMISYNC_UPDATE_EXISTING_SLAVE_ACK, 5352
ER_SEMISYNC_WAIT_FOR_BINLOG_TIMEDOUT, 5350
ER_SEMISYNC_WAIT_TIME_ASSESSMENT_FOR_COMMIT_TRX_FAILED, 5351
ER_SEMISYNC_WAIT_TIME_FOR_BINLOG_SENT, 5350
ER_SERVERID_TOO_LARGE, 5239
ER_SERVER_ACL_TABLE_ERROR, 5564
ER_SERVER_BINLOG_MASTER_KEY_RECOVERY_OUT_OF_COMBINATION, 5588
ER_SERVER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_AUX_KEY, 5588
ER_SERVER_BINLOG_UNSAFE_SYSTEM_FUNCTION, 5592
ER_SERVER_CANNOT_LOAD_FROM_TABLE_V2, 5563
ER_SERVER_CANT_OPEN_FILE, 5563
ER_SERVER_COL_COUNT_DOESNT_MATCH_CORRUPTED_V2, 5564
ER_SERVER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE_V2, 5563
ER_SERVER_COST_FAILED_TO_READ, 5268
ER_SERVER_COST_INVALID_COST_CONSTANT, 5267
ER_SERVER_COST_UNKNOWN_COST_CONSTANT, 5267
ER_SERVER_DISK_FULL_NOWAIT, 5563
ER_SERVER_FILE_NOT_FOUND, 5563
ER_SERVER_FILE_USED, 5563
ER_SERVER_GTID_UNSAFE_CREATE_DROP_TEMP_TABLE_IN_TRX_IN_SBR, 5573
ER_SERVER_HANDLER_ERROR, 5563
ER_SERVER_INIT_COMPILED_IN_COMMANDS, 5340

ER_SERVER_ISNT_AVAILABLE, 5170
ER_SERVER_IS_IN_SECURE_AUTH_MODE, 5117
ER_SERVER_MASTER_FATAL_ERROR_READING_BINLOG, 5561
ER_SERVER_NET_PACKET_TOO_LARGE, 5562
ER_SERVER_NEW_ABORTING_CONNECTION, 5562
ER_SERVER_NOT_FORM_FILE, 5563
ER_SERVER_NO_SESSION_TO_SEND_TO, 5562
ER_SERVER_NO_SYSTEM_TABLE_ACCESS, 5562
ER_SERVER_OFFLINE_MODE, 5160
ER_SERVER_OUTOFMEMORY, 5325
ER_SERVER_OUT_OF_RESOURCES, 5325
ER_SERVER_OUT_OF_SORTMEMORY, 5562
ER_SERVER_RECORD_FILE_FULL, 5562
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_ENCRYPT, 5584
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_FETCH_KEY, 5579
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_GENERATE_KEY, 5579
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_REMOVE_KEY, 5580
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_ROTATE_LOGS, 5579
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_STORE_KEY, 5580
ER_SERVER_RPL_ENCRYPTION_HEADER_ERROR, 5579
ER_SERVER_RPL_ENCRYPTION_IGNORE_ROTATE_MASTER_KEY_AT_STARTUP, 5580
ER_SERVER_RPL_ENCRYPTION_KEYRING_INVALID_KEY, 5579
ER_SERVER_RPL_ENCRYPTION_KEY_EXISTS_UNEXPECTED, 5579
ER_SERVER_RPL_ENCRYPTION_KEY_NOT_FOUND, 5579
ER_SERVER_RPL_ENCRYPTION_MASTER_KEY_RECOVERY_FAILED, 5580
ER_SERVER_RPL_ENCRYPTION_UNABLE_TO_INITIALIZE, 5580
ER_SERVER_RPL_ENCRYPTION_UNABLE_TO_ROTATE_MASTER_KEY_AT_STARTUP, 5580
ER_SERVER_SHUTDOWN, 5102
ER_SERVER_SHUTDOWN_COMPLETE, 5323
ER_SERVER_SHUTDOWN_INFO, 5567
ER_SERVER_SLAVE_CONVERSION_FAILED, 5564
ER_SERVER_SLAVE_IGNORED_TABLE, 5564
ER_SERVER_SLAVE_INIT_QUERY_FAILED, 5564
ER_SERVER_SLAVE_MI_INIT_REPOSITORY, 5562
ER_SERVER_SLAVE_RLI_INIT_REPOSITORY, 5562
ER_SERVER_STARTUP_ADMIN_INTERFACE, 5580
ER_SERVER_STARTUP_MSG, 5325
ER_SERVER_TABLE_CHECK_FAILED, 5563
ER_SERVER_TEST_MESSAGE, 5564
ER_SERVER_UNKNOWN_ERROR, 5562
ER_SERVER_UNKNOWN_SYSTEM_VARIABLE, 5562
ER_SERVER_UPGRADE_CHECKING_DB, 5591
ER_SERVER_UPGRADE_EMPTY_SYS, 5591
ER_SERVER_UPGRADE_FAILED, 5590
ER_SERVER_UPGRADE_FROM_VERSION, 5584
ER_SERVER_UPGRADE_HELP_TABLE_STATUS, 5592
ER_SERVER_UPGRADE_INFO_FILE, 5590
ER_SERVER_UPGRADE_MYSQL_TABLES, 5590
ER_SERVER_UPGRADE_NO_SYS_VERSION, 5591
ER_SERVER_UPGRADE_OFF, 5589
ER_SERVER_UPGRADE_PENDING, 5590
ER_SERVER_UPGRADE_REPAIR_REQUIRED, 5590
ER_SERVER_UPGRADE_REPAIR_STATUS, 5590
ER_SERVER_UPGRADE_SKIP, 5590

ER_SERVER_UPGRADE_STATUS, 5590
ER_SERVER_UPGRADE_SYSTEM_TABLES, 5591
ER_SERVER_UPGRADE_SYS_SCHEMA, 5590
ER_SERVER_UPGRADE_SYS_SCHEMA_OBJECT_COUNT, 5591
ER_SERVER_UPGRADE_SYS_SCHEMA_OUTDATED, 5591
ER_SERVER_UPGRADE_SYS_SCHEMA_UP_TO_DATE, 5591
ER_SERVER_UPGRADE_SYS_VERSION_EMPTY, 5591
ER_SERVER_UPGRADE_VERSION_NOT_SUPPORTED, 5584
ER_SERVER_WARN_DEPRECATED, 5588
ER_SERVER_WRONG_VALUE_FOR_VAR, 5573
ER_SESSION_WAS_KILLED, 5170
ER_SETNS_FAILED, 5593
ER_SET_CONSTANTS_ONLY, 5112
ER_SET_EVENT_FAILED, 5568
ER_SET_PASSWORD_AUTH_PLUGIN, 5143
ER_SET_STATEMENT_CANNOT_INVOKE_FUNCTION, 5148
ER_SE_TYPECODE_CONFLICT, 5245
ER_SHA256_PASSWORD_SECOND_PASSWORD_USED_INFORMATION, 5581
ER_SHARED_TABLESPACE_USED_BY_PARTITIONED_TABLE, 5574
ER_SHA_PWD_AUTH_REQUIRES_RSA_OR_SSL, 5361
ER_SHA_PWD_FAILED_TO_GENERATE_MULTI_ROUND_HASH, 5361
ER_SHA_PWD_FAILED_TO_PARSE_AUTH_STRING, 5361
ER_SHA_PWD_RSA_KEY_TOO_LONG, 5361
ER_SHA_PWD_SALT_FOR_USER_CORRUPT, 5362
ER_SHUTDOWN_COMPLETE, 5104
ER_SHUTTING_DOWN_SLAVE_THREADS, 5232
ER_SIGNAL_BAD_CONDITION_TYPE, 5139
ER_SIGNAL_EXCEPTION, 5139
ER_SIGNAL_NOT_FOUND, 5139
ER_SIGNAL_WARN, 5139
ER_SIZE_OVERFLOW_ERROR, 5133
ER_SKIP_UPDATING_METADATA_IN_SE_RO_MODE, 5335
ER_SLAVE_CANT_CREATE_CONVERSION, 5142
ER_SLAVE_CANT_USE_TEMPDIR, 5287
ER_SLAVE_CHANGE_MASTER_TO_EXECUTED, 5286
ER_SLAVE_CHANNEL_DOES_NOT_EXIST, 5163
ER_SLAVE_CHANNEL_IO_THREAD_MUST_STOP, 5159
ER_SLAVE_CHANNEL_MUST_STOP, 5163
ER_SLAVE_CHANNEL_NAME_INVALID_OR_TOO_LONG, 5163
ER_SLAVE_CHANNEL_NOT_RUNNING, 5163
ER_SLAVE_CHANNEL_OPERATION_NOT_ALLOWED, 5168
ER_SLAVE_CHANNEL_SQL_SKIP_COUNTER, 5164
ER_SLAVE_CHANNEL_SQL_THREAD_MUST_STOP, 5164
ER_SLAVE_CHANNEL_WAS_NOT_RUNNING, 5163
ER_SLAVE_CHANNEL_WAS_RUNNING, 5163
ER_SLAVE_CONFIGURATION, 5150
ER_SLAVE_CONVERSION_FAILED, 5142
ER_SLAVE_CORRUPT_EVENT, 5137
ER_SLAVE_CREATE_EVENT_FAILURE, 5561
ER_SLAVE_FATAL_ERROR, 5561
ER_SLAVE_HAS_MORE_GTIDS_THAN_MASTER, 5157
ER_SLAVE_HEARTBEAT_FAILURE, 5561
ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE, 5138
ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MAX, 5144

ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MIN, 5143
ER_SLAVE_IGNORED_SSL_PARAMS, 5117
ER_SLAVE_IGNORED_TABLE, 5114
ER_SLAVE_IGNORE_SERVER_IDS, 5140
ER_SLAVE_INCIDENT, 5561
ER_SLAVE_KILLED_AFTER_RECONNECT, 5336
ER_SLAVE_MASTER_COM_FAILURE, 5561
ER_SLAVE_MAX_CHANNELS_EXCEEDED, 5163
ER_SLAVE_MI_INIT_REPOSITORY, 5156
ER_SLAVE_MULTIPLE_CHANNELS_CMD, 5163
ER_SLAVE_NEW_CHANNEL_WRONG_REPOSITORY, 5163
ER_SLAVE_NOT_RUNNING, 5112
ER_SLAVE_NOT_STARTED_ON_SOME_CHANNELS, 5336
ER_SLAVE_POSSIBLY_DIVERGED_AFTER_DDL, 5196
ER_SLAVE_RECONNECT_FAILED, 5336
ER_SLAVE_RELAY_LOG_PURGE_FAILED, 5328
ER_SLAVE_RELAY_LOG_READ_FAILURE, 5561
ER_SLAVE_RELAY_LOG_TRUNCATE_INFO, 5328
ER_SLAVE_RELAY_LOG_WRITE_FAILURE, 5561
ER_SLAVE_RLI_INIT_REPOSITORY, 5156
ER_SLAVE_SILENT_RETRY_TRANSACTION, 5151
ER_SLAVE_SQL_THREAD_MUST_STOP, 5159
ER_SLAVE_SQL_THREAD_STOPPED_AFTER_GTIDS_REACHED, 5260
ER_SLAVE_SQL_THREAD_STOPPED_BEFORE_GTIDS_ALREADY_APPLIED, 5260
ER_SLAVE_SQL_THREAD_STOPPED_BEFORE_GTIDS_REACHED, 5260
ER_SLAVE_SQL_THREAD_STOPPED_GAP_TRX_PROCESSED, 5261
ER_SLAVE_SQL_THREAD_STOPPED_UNTIL_CONDITION_BAD, 5260
ER_SLAVE_SQL_THREAD_STOPPED_UNTIL_POSITION_REACHED, 5260
ER_SLAVE_THREAD, 5112
ER_SLAVE_WORKER_STOPPED_PREVIOUS_THD_ERROR, 5160
ER_SLOW_LOG_MODE_IGNORED_WHEN_NOT_LOGGING_TO_FILE, 5204
ER_SPATIAL_CANT_HAVE_NULL, 5115
ER_SPATIAL_FUNCTIONAL_INDEX, 5200
ER_SPATIAL_MUST_HAVE_GEOM_COL, 5142
ER_SPATIAL_UNIQUE_INDEX, 5196
ER_SPECIFIC_ACCESS_DENIED_ERROR, 5114
ER_SP_ALREADY_EXISTS, 5119
ER_SP_BADRETURN, 5119
ER_SP_BADSELECT, 5119
ER_SP_BADSTATEMENT, 5119
ER_SP_BAD_CURSOR_QUERY, 5120
ER_SP_BAD_CURSOR_SELECT, 5120
ER_SP_BAD_SQLSTATE, 5125
ER_SP_BAD_VAR_SHADOW, 5128
ER_SP_CANT_ALTER, 5121
ER_SP_CANT_SET_AUTOCOMMIT, 5127
ER_SP_CASE_NOT_FOUND, 5121
ER_SP_COND_MISMATCH, 5120
ER_SP_CURSOR_AFTER_HANDLER, 5121
ER_SP_CURSOR_ALREADY_OPEN, 5120
ER_SP_CURSOR_MISMATCH, 5120
ER_SP_CURSOR_NOT_OPEN, 5120
ER_SP_DOES_NOT_EXIST, 5119
ER_SP_DROP_FAILED, 5119

ER_SP_DUP_COND, 5121
ER_SP_DUP_CURS, 5121
ER_SP_DUP_HANDLER, 5125
ER_SP_DUP_PARAM, 5120
ER_SP_DUP_VAR, 5121
ER_SP_FETCH_NO_DATA, 5120
ER_SP_LABEL_MISMATCH, 5119
ER_SP_LABEL_REDEFINE, 5119
ER_SP_LILABEL_MISMATCH, 5119
ER_SP_LOAD_FAILED, 5174
ER_SP_NORETURN, 5120
ER_SP_NORETURNEND, 5120
ER_SP_NOT_VAR_ARG, 5125
ER_SP_NO_AGGREGATE, 5128
ER_SP_NO_DROP_SP, 5122
ER_SP_NO_RECURSION, 5126
ER_SP_NO_RECURSIVE_CREATE, 5119
ER_SP_NO_RETSET, 5126
ER_SP_RECURSION_LIMIT, 5128
ER_SP_STORE_FAILED, 5119
ER_SP_SUBSELECT_NYI, 5121
ER_SP_UNDECLARED_VAR, 5120
ER_SP_UNINIT_VAR, 5119
ER_SP_VARCOND_AFTER_CURSHNDLR, 5121
ER_SP_WRONG_NAME, 5128
ER_SP_WRONG_NO_OF_ARGS, 5120
ER_SP_WRONG_NO_OF_FETCH_ARGS, 5120
ER_SQLTHREAD_WITH_SECURE_SLAVE, 5148
ER_SQL_AUTHOR_DEFAULT_ROLES_FAIL, 5244
ER_SQL_HA_READ_FAILED, 5242
ER_SQL_MODE_MERGED, 5168
ER_SQL_MODE_MERGED_WITH_STRICT_MODE, 5323
ER_SQL_SLAVE_SKIP_COUNTER_NOT_SETTABLE_IN_GTID_MODE, 5155
ER_SQL_USER_TABLE_ALTER_WARNING, 5245
ER_SQL_USER_TABLE_CREATE_WARNING, 5244
ER_SRS_ATTRIBUTE_STRING_TOO_LONG, 5195
ER_SRS_GEOGCS_INVALID_AXES, 5197
ER_SRS_ID_ALREADY_EXISTS, 5194
ER_SRS_INVALID_ANGULAR_UNIT, 5197
ER_SRS_INVALID_CHARACTER_IN_ATTRIBUTE, 5195
ER_SRS_INVALID_INVERSE_FLATTENING, 5197
ER_SRS_INVALID_PRIME_MERIDIAN, 5197
ER_SRS_INVALID_SEMI_MAJOR_AXIS, 5197
ER_SRS_MISSING_MANDATORY_ATTRIBUTE, 5194
ER_SRS_MULTIPLE_ATTRIBUTE_DEFINITIONS, 5194
ER_SRS_NAME_CANT_BE_EMPTY_OR_WHITESPACE, 5194
ER_SRS_NOT_CARTESIAN, 5174
ER_SRS_NOT_CARTESIAN_UNDEFINED, 5175
ER_SRS_NOT_FOUND, 5177
ER_SRS_NOT_GEOGRAPHIC, 5196
ER_SRS_ORGANIZATION_CANT_BE_EMPTY_OR_WHITESPACE, 5194
ER_SRS_PARSE_ERROR, 5174
ER_SRS_PROJ_PARAMETER_MISSING, 5174
ER_SR_BOGUS_VALUE, 5242

ER_SR_INVALID_CONTEXT, 5242
ER_SSL_FIPS_MODE_ERROR, 5363
ER_SSL_LIBRARY_ERROR, 5227
ER_SSL_MEMORY_INSTRUMENTATION_INIT_FAILED, 5575
ER_SSL_SERVER_CERT_VERIFY_FAILED, 5593
ER_SSL_TRYING_DATADIR_DEFAULTS, 5239
ER_STACKSIZE_UNEXPECTED, 5238
ER_STACK_OVERRUN, 5107
ER_STACK_OVERRUN_NEED_MORE, 5127
ER_STARTING_AS, 5232
ER_STARTING_INIT, 5567
ER_STARTUP, 5125
ER_STD_BAD_ALLOC_ERROR, 5161
ER_STD_DOMAIN_ERROR, 5161
ER_STD_INVALID_ARGUMENT, 5161
ER_STD_LENGTH_ERROR, 5161
ER_STD_LOGIC_ERROR, 5161
ER_STD_OUT_OF_RANGE_ERROR, 5161
ER_STD_OVERFLOW_ERROR, 5161
ER_STD_RANGE_ERROR, 5161
ER_STD_REGEX_ERROR, 5210
ER_STD_RUNTIME_ERROR, 5161
ER_STD_UNDERFLOW_ERROR, 5161
ER_STD_UNKNOWN_EXCEPTION, 5161
ER_STMT_CACHE_FULL, 5144
ER_STMT_HAS_NO_OPEN_CURSOR, 5126
ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG, 5121
ER_STOP_SLAVE_IO_THREAD_DISK_SPACE, 5363
ER_STOP_SLAVE_IO_THREAD_TIMEOUT, 5156
ER_STOP_SLAVE_SQL_THREAD_TIMEOUT, 5156
ER_STORAGE_ENGINE_NOT_LOADED, 5158
ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_DIRECT, 5142
ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_FORMAT, 5134
ER_STORED_FUNCTION_PREVENTS_SWITCH_SQL_LOG_BIN, 5143
ER_SUBPARTITION_ERROR, 5131
ER_SUBPARTITION_NAME, 5138
ER_SUBQUERY_NO_1_ROW, 5115
ER_SWITCH_TMP_ENGINE, 5179
ER_SYNTAX_ERROR, 5109
ER_SYSTEMD_NOTIFY_CONNECT_FAILED, 5427
ER_SYSTEMD_NOTIFY_PATH_TOO_LONG, 5427
ER_SYSTEMD_NOTIFY_WRITE_FAILED, 5427
ER_SYSTEM_SCHEMA_NOT_FOUND, 5221
ER_SYSTEM_TABLES_NOT_SUPPORTED_BY_STORAGE_ENGINE, 5325
ER_SYSTEM_TABLE_NOT_TRANSACTIONAL, 5302
ER_SYSTEM_VIEW_INIT_FAILED, 5326
ER_SYSVAR_CHANGE_DURING_QUERY, 5217
ER_SYS_VAR_COMPONENT_FAILED_TO_MAKE_VARIABLE_PERSISTENT, 5363
ER_SYS_VAR_COMPONENT_FAILED_TO_PARSE_VARIABLE_OPTIONS, 5363
ER_SYS_VAR_COMPONENT_OOM, 5362
ER_SYS_VAR_COMPONENT_UNKNOWN_VARIABLE_TYPE, 5362
ER_SYS_VAR_COMPONENT_VARIABLE_SET_READ_ONLY, 5362
ER_SYS_VAR_NOT_FOUND, 5568
ER_TABLEACCESS_DENIED_ERROR, 5108

ER_TABLENAME_NOT_ALLOWED_HERE, 5115
ER_TABLESPACE_AUTO_EXTEND_ERROR, 5133
ER_TABLESPACE_CANNOT_BE_DECRYPTED, 5208
ER_TABLESPACE_CANNOT_BE_ENCRYPTED, 5208
ER_TABLESPACE_CANNOT_ENCRYPT, 5171
ER_TABLESPACE_DISCARDED, 5152
ER_TABLESPACE_DUP_FILENAME, 5183
ER_TABLESPACE_ENGINE_MISMATCH, 5187
ER_TABLESPACE_EXISTS, 5152
ER_TABLESPACE_IS_NOT_EMPTY, 5167
ER_TABLESPACE_MISSING, 5152
ER_TABLESPACE_MISSING_WITH_NAME, 5174
ER_TABLESPACE_TYPE_UNKNOWN, 5208
ER_TABLES_DIFFERENT_METADATA, 5146
ER_TABLE_CANT_HANDLE_AUTO_INCREMENT, 5110
ER_TABLE_CANT_HANDLE_BLOB, 5110
ER_TABLE_CANT_HANDLE_FT, 5113
ER_TABLE_CANT_HANDLE_SPKEYS, 5129
ER_TABLE_CHECK_INTACT, 5240
ER_TABLE_CORRUPT, 5156
ER_TABLE_CREATED_WITH_DIFFERENT_VERSION, 5257
ER_TABLE_DEF_CHANGED, 5125
ER_TABLE_EXISTS_ERROR, 5101
ER_TABLE_HAS_NO_FT, 5148
ER_TABLE_INCOMPATIBLE_DECIMAL_FIELD, 5256
ER_TABLE_INCOMPATIBLE_YEAR_FIELD, 5256
ER_TABLE_IN_FK_CHECK, 5145
ER_TABLE_IN_SYSTEM_TABLESPACE, 5151
ER_TABLE_MUST_HAVE_COLUMNS, 5106
ER_TABLE_NAME, 5138
ER_TABLE_NAME_CAUSES_TOO_LONG_PATH, 5217
ER_TABLE_NEEDS_DUMP_UPGRADE, 5343
ER_TABLE_NEEDS_REBUILD, 5144
ER_TABLE_NEEDS_UPGRADE, 5128
ER_TABLE_NOT_LOCKED, 5105
ER_TABLE_NOT_LOCKED_FOR_WRITE, 5105
ER_TABLE_REFERENCED, 5172
ER_TABLE_SCHEMA_MISMATCH, 5151
ER_TABLE_UPGRADE_REQUIRED, 5324
ER_TABLE_WITHOUT_PK, 5199
ER_TABLE_WRONG_KEY_DEFINITION, 5257
ER_TARGET_TABLESPACE_UNENCRYPTED, 5208
ER_TARGET_TS_UNENCRYPTED, 5569
ER_TC_BAD_MAGIC_IN_TC_LOG, 5277
ER_TC_CANT_AUTO_RECOVER_WITH_TC_HEURISTIC_RECOVER, 5277
ER_TC_HEURISTIC_RECOVERY_FAILED, 5278
ER_TC_HEURISTIC_RECOVERY_MODE, 5278
ER_TC_NEED_N_SE_SUPPORTING_2PC_FOR_RECOVERY, 5278
ER_TC_RECOVERING_AFTER_CRASH_USING, 5277
ER_TC_RECOVERY_FAILED_THESE_ARE_YOUR_OPTIONS, 5278
ER_TC_RESTART_WITHOUT_TC_HEURISTIC_RECOVER, 5278
ER_TEMPORARY_NAME, 5139
ER_TEMP_FILE_WRITE_FAILURE, 5156
ER_TEMP_TABLE_PREVENTS_SWITCH_GLOBAL_BINLOG_FORMAT, 5198

ER_TEMP_TABLE_PREVENTS_SWITCH_OUT_OF_RBR, 5134
ER_TEMP_TABLE_PREVENTS_SWITCH_SESSION_BINLOG_FORMAT, 5198
ER_TEXTFILE_NOT_READABLE, 5104
ER_TF_FORBIDDEN_JOIN_TYPE, 5189
ER_TF_MUST_HAVE_ALIAS, 5189
ER_THE_USER_ABIDES, 5235
ER_THREAD_HANDLING_OOM, 5229
ER_THREAD_POOL_ALGORITHM_INVALID, 5344
ER_THREAD_POOL_BUFFER_TOO_SMALL, 5346
ER_THREAD_POOL_CANNOT_SET_THREAD_SPECIFIC_DATA, 5345
ER_THREAD_POOL_CON_HANDLER_INIT_FAILED, 5345
ER_THREAD_POOL_FAILED_PROCESS_CONNECT_EVENT, 5346
ER_THREAD_POOL_FAILED_TO_CREATE_CONNECT_HANDLER_THD, 5346
ER_THREAD_POOL_FAILED_TO_CREATE_POOL, 5346
ER_THREAD_POOL_FAILED_TO_CREATE_THD_AND_AUTH_CONN, 5346
ER_THREAD_POOL_INIT_FAILED, 5345
ER_THREAD_POOL_INVALID_PRIO_KICKUP_TIMER, 5345
ER_THREAD_POOL_INVALID_STALL_LIMIT, 5345
ER_THREAD_POOL_LOW_LEVEL_REARM_FAILED, 5346
ER_THREAD_POOL_MAX_UNUSED_THREADS_INVALID, 5345
ER_THREAD_POOL_NOT_SUPPORTED_ON_PLATFORM, 5344
ER_THREAD_POOL_PFS_TABLES_ADD_FAILED, 5578
ER_THREAD_POOL_PFS_TABLES_INIT_FAILED, 5578
ER_THREAD_POOL_PLUGIN_STARTED, 5345
ER_THREAD_POOL_RATE_LIMITED_ERROR_MSGS, 5346
ER_THREAD_POOL_SIZE_TOO_HIGH, 5344
ER_THREAD_POOL_SIZE_TOO_LOW, 5344
ER_THREAD_PRIORITY_IGNORED, 5322
ER_TOO_BIG_DISPLAYWIDTH, 5127
ER_TOO_BIG_ENUM, 5173
ER_TOO_BIG_FIELDLENGTH, 5103
ER_TOO_BIG_FOR_UNCOMPRESS, 5115
ER_TOO_BIG_PRECISION, 5126
ER_TOO_BIG_ROWSIZE, 5106
ER_TOO_BIG_SCALE, 5126
ER_TOO_BIG_SELECT, 5105
ER_TOO_BIG_SET, 5105
ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT, 5129
ER_TOO_LONG_BODY, 5127
ER_TOO_LONG_FIELD_COMMENT, 5138
ER_TOO_LONG_IDENT, 5102
ER_TOO_LONG_INDEX_COMMENT, 5142
ER_TOO_LONG_KEY, 5103
ER_TOO_LONG_ROUTINE_COMMENT, 5174
ER_TOO_LONG_SET_ENUM_VALUE, 5173
ER_TOO_LONG_STRING, 5110
ER_TOO_LONG_TABLESPACE_COMMENT, 5182
ER_TOO_LONG_TABLE_COMMENT, 5138
ER_TOO_LONG_TABLE_PARTITION_COMMENT, 5150
ER_TOO_MANY_CONCURRENT_CLONES, 5186
ER_TOO_MANY_CONCURRENT_TRXS, 5139
ER_TOO_MANY_FIELDS, 5106
ER_TOO_MANY_KEYS, 5103
ER_TOO_MANY_KEY_PARTS, 5103

ER_TOO_MANY_PARTITIONS_ERROR, 5131
ER_TOO_MANY_PARTITION_FUNC_FIELDS_ERROR, 5140
ER_TOO_MANY_ROWS, 5110
ER_TOO_MANY_STORAGE_ENGINES, 5245
ER_TOO_MANY_TABLES, 5106
ER_TOO_MANY_USER_CONNECTIONS, 5112
ER_TOO_MANY_VALUES_ERROR, 5140
ER_TRACK_VARIABLES_BOGUS, 5235
ER_TRANSACTION_ROLLBACK_DURING_COMMIT, 5165
ER_TRANSFORM_SOURCE_SRS_MISSING_TOWGS84, 5198
ER_TRANSFORM_SOURCE_SRS_NOT_SUPPORTED, 5197
ER_TRANSFORM_TARGET_SRS_MISSING_TOWGS84, 5198
ER_TRANSFORM_TARGET_SRS_NOT_SUPPORTED, 5198
ER_TRANSPORTS_WHAT_TRANSPORTS, 5233
ER_TRANS_CACHE_FULL, 5112
ER_TREE_CORRUPT_2_CONSECUTIVE_REDS, 5300
ER_TREE_CORRUPT_INCORRECT_BLACK_COUNT, 5300
ER_TREE_CORRUPT_PARENT_SHOULD_POINT_AT_PARENT, 5300
ER_TREE_CORRUPT_RIGHT_IS_LEFT, 5300
ER_TREE_CORRUPT_ROOT_SHOULD_BE_BLACK, 5300
ER_TRG_ALREADY_EXISTS, 5122
ER_TRG_CANT_CHANGE_ROW, 5122
ER_TRG_CANT_OPEN_TABLE, 5137
ER_TRG_CANT_PARSE, 5240
ER_TRG_CORRUPTED_FILE, 5137
ER_TRG_CREATION_CTX_NOT_SET, 5324
ER_TRG_DOES_NOT_EXIST, 5122
ER_TRG_INVALID_CREATION_CTX, 5137
ER_TRG_IN_WRONG_SCHEMA, 5127
ER_TRG_NO_CLIENT_CHARSET, 5226
ER_TRG_NO_CREATION_CTX, 5137
ER_TRG_NO_DEFINER, 5128
ER_TRG_NO_SUCH_ROW_IN_TRG, 5123
ER_TRG_ON_VIEW_OR_TEMP_TABLE, 5122
ER_TRG_WITHOUT_DEFINER, 5226
ER_TRG_WRONG_ORDER, 5585
ER_TRHEAD_POOL_LOW_LEVEL_INIT_FAILED, 5346
ER_TRIGGER_INVALID_VALUE, 5220
ER_TRUNCATED_WRONG_VALUE, 5118
ER_TRUNCATED_WRONG_VALUE_FOR_FIELD, 5123
ER_TRUNCATE_ILLEGAL_FK, 5143
ER_TRX_GTID_COLLECT_REJECT, 5244
ER_TRX_WRITE_SET_OOM, 5245
ER_TURNING_LOGGING_OFF_FOR_THE_DURATION, 5317
ER_TURNING_ON_PARTIAL_REVOKES, 5588
ER_TX_EXTRACTION_ALGORITHM_FOR_BINLOG_TX_DEPEDENCY_TRACKING, 5328
ER_TZ_CANT_BUILD_MKTIME_MAP, 5260
ER_TZ_CANT_FIND_DESCRIPTION_FOR_TIME_ZONE, 5258
ER_TZ_CANT_FIND_DESCRIPTION_FOR_TIME_ZONE_ID, 5259
ER_TZ_CANT_OPEN_AND_LOCK_TIME_ZONE_TABLE, 5258
ER_TZ_ERROR_LOADING_LEAP_SECOND_TABLE, 5258
ER_TZ_NO_TRANSITION_TYPES_IN_TIME_ZONE, 5259
ER_TZ_OOM_INITIALIZING_TIME_ZONES, 5258
ER_TZ_OOM_LOADING_LEAP_SECOND_TABLE, 5258

ER_TZ_OOM_LOADING_TIME_ZONE_DESCRIPTION, 5260
ER_TZ_OOM_WHILE_LOADING_TIME_ZONE, 5260
ER_TZ_OOM_WHILE_SETTING_TIME_ZONE, 5260
ER_TZ_TOO_MANY_LEAPS_IN_LEAP_SECOND_TABLE, 5258
ER_TZ_TRANSITION_TABLE_BAD_TRANSITION_TYPE, 5259
ER_TZ_TRANSITION_TABLE_LOAD_ERROR, 5259
ER_TZ_TRANSITION_TABLE_TOO_MANY_TRANSITIONS, 5259
ER_TZ_TRANSITION_TYPE_TABLE_ABBREVIATIONS_EXCEED_SPACE, 5259
ER_TZ_TRANSITION_TYPE_TABLE_LOAD_ERROR, 5259
ER_TZ_TRANSITION_TYPE_TABLE_TYPE_TOO_LARGE, 5259
ER_TZ_UNKNOWN_OR_ILLEGAL_DEFAULT_TIME_ZONE, 5258
ER_UDF_ALREADY_EXISTS, 5567
ER_UDF_CANT_ALLOC_FOR_FUNCTION, 5243
ER_UDF_CANT_ALLOC_FOR_STRUCTURES, 5243
ER_UDF_CANT_OPEN_FUNCTION_TABLE, 5243
ER_UDF_DROP_DYNAMICALLY_REGISTERED, 5184
ER_UDF_ERROR, 5173
ER_UDF_EXISTS, 5107
ER_UDF_INVALID_ROW_IN_FUNCTION_TABLE, 5243
ER_UDF_NO_PATHS, 5107
ER_UKNOWN_AUTH_ID_OR_ACCESS_DENIED_FOR_GRANT_AS, 5209
ER_UNABLE_TO_BUILD_HISTOGRAM, 5185
ER_UNABLE_TO_COLLECT_LOG_STATUS, 5195
ER_UNABLE_TO_DROP_COLUMN_STATISTICS, 5185
ER_UNABLE_TO_RESOLVE_HOSTNAME, 5226
ER_UNABLE_TO_RESOLVE_IP, 5225
ER_UNABLE_TO_SET_OPTION, 5196
ER_UNABLE_TO_STORE_COLUMN_STATISTICS, 5184
ER_UNABLE_TO_STORE_STATISTICS, 5177
ER_UNABLE_TO_UPDATE_COLUMN_STATISTICS, 5185
ER_UNDISCLOSED_PARSE_ERROR_IN_DIGEST_FN, 5190
ER_UNDO_RECORD_TOO_BIG, 5144
ER_UNEXPECTED_GEOMETRY_TYPE, 5174
ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF, 5131
ER_UNIT_NOT_FOUND, 5216
ER_UNKNOWN_ALTER_ALGORITHM, 5151
ER_UNKNOWN_ALTER_LOCK, 5151
ER_UNKNOWN_AUTHID, 5175
ER_UNKNOWN_AUTH_ID_IN_MANDATORY_ROLE, 5329
ER_UNKNOWN_CHARACTER_SET, 5106
ER_UNKNOWN_COLLATION, 5117
ER_UNKNOWN_COM_ERROR, 5101
ER_UNKNOWN_ERROR, 5106
ER_UNKNOWN_ERROR_DETECTED_IN_SE, 5335
ER_UNKNOWN_ERROR_NUMBER, 5243
ER_UNKNOWN_EXPLAIN_FORMAT, 5150
ER_UNKNOWN_KEY_CACHE, 5117
ER_UNKNOWN_LOCALE, 5139
ER_UNKNOWN_NETWORK_NAMESPACE, 5593
ER_UNKNOWN_PARTITION, 5146
ER_UNKNOWN_PROCEDURE, 5106
ER_UNKNOWN_STMT_HANDLER, 5115
ER_UNKNOWN_STORAGE_ENGINE, 5118
ER_UNKNOWN_SYSTEM_VARIABLE, 5111

ER_UNKNOWN_TABLE, 5106
ER_UNKNOWN_TABLESPACE_TYPE, 5574
ER_UNKNOWN_TABLE_IN_UPGRADE, 5599
ER_UNKNOWN_TARGET_BINLOG, 5123
ER_UNKNOWN_TIME_ZONE, 5118
ER_UNKNOWN_UNSUPPORTED_STORAGE_ENGINE, 5228
ER_UNKNOWN_VARIABLE_IN_PERSISTED_CONFIG_FILE, 5568
ER_UNRESOLVED_HINT_NAME, 5167
ER_UNRESOLVED_TABLE_LOCK, 5178
ER_UNSUPORTED_LOG_ENGINE, 5135
ER_UNSUPPORTED_ACTION_ON_DEFAULT_VAL_GENERATED, 5201
ER_UNSUPPORTED_ACTION_ON_GENERATED_COLUMN, 5165
ER_UNSUPPORTED_ALTER_ENCRYPTION_INPLACE, 5172
ER_UNSUPPORTED_ALTER_INPLACE_ON_VIRTUAL_COLUMN, 5165
ER_UNSUPPORTED_ALTER_ONLINE_ON_VIRTUAL_COLUMN, 5171
ER_UNSUPPORTED_DATE, 5232
ER_UNSUPPORTED_ENGINE, 5145
ER_UNSUPPORTED_EXTENSION, 5106
ER_UNSUPPORTED_INDEX_ALGORITHM, 5173
ER_UNSUPPORTED_PS, 5118
ER_UNSUPPORTED_SQL_MODE, 5215
ER_UNSUPPORTED_USE_OF_GRANT_AS, 5209
ER_UNSUPPORT_COMPRESSED_TEMPORARY_TABLE, 5173
ER_UNTIL_COND_IGNORED, 5117
ER_UPDATE_GTID_PURGED_WITH_GR, 5217
ER_UPDATE_INFO, 5108
ER_UPDATE_LOG_DEPRECATED_IGNORED, 5119
ER_UPDATE_LOG_DEPRECATED_TRANSLATED, 5120
ER_UPDATE_TABLE_USED, 5105
ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE, 5110
ER_UPDATING_DD_TABLE, 5174
ER_UPGRADE_PARSE_ERROR, 5573
ER_UPGRADE_WITH_PARTITIONED_TABLES_REJECTED, 5601
ER_USERNAME, 5129
ER_USER_ALREADY_EXISTS, 5170
ER_USER_COLUMN_OLD_LENGTH, 5172
ER_USER_DOES_NOT_EXIST, 5170
ER_USER_LIMIT_REACHED, 5114
ER_USER_LOCK_DEADLOCK, 5162
ER_USER_LOCK_WRONG_NAME, 5162
ER_USER_NOT_IN_EXTRA_USERS_BINLOG_POSSIBLY_INCOMPLETE, 5255
ER_USER_REQUIRES_ROOT, 5233
ER_USER_WHAT_USER, 5233
ER_UUID_INVALID, 5227
ER_UUID_IS, 5227
ER_UUID_SALT, 5227
ER_UUID_SCRUB, 5227
ER_VALGRIND_COUNT_LEAKS, 5242
ER_VALGRIND_DO_QUICK_LEAK_CHECK, 5242
ER_VALIDATE_PWD_CONVERT_TO_BUFFER_FAILED, 5429
ER_VALIDATE_PWD_COULD_BE_NULL, 5428
ER_VALIDATE_PWD_DICT_FILE_NOT_LOADED, 5359
ER_VALIDATE_PWD_DICT_FILE_NOT_SPECIFIED, 5359
ER_VALIDATE_PWD_DICT_FILE_OPEN_FAILED, 5428

ER_VALIDATE_PWD_DICT_FILE_TOO_BIG, 5359
ER_VALIDATE_PWD_FAILED_TO_GET_FLD_FROM_SECURITY_CTX, 5360
ER_VALIDATE_PWD_FAILED_TO_GET_SECURITY_CTX, 5360
ER_VALIDATE_PWD_FAILED_TO_READ_DICT_FILE, 5359
ER_VALIDATE_PWD_LENGTH_CHANGED, 5360
ER_VALIDATE_PWD_STATUS_VAR_REGISTRATION_FAILED, 5428
ER_VALIDATE_PWD_STATUS_VAR_UNREGISTRATION_FAILED, 5428
ER_VALIDATE_PWD_STRING_CONV_TO_BUFFER_FAILED, 5429
ER_VALIDATE_PWD_STRING_CONV_TO_LOWERCASE_FAILED, 5428
ER_VALIDATE_PWD_STRING_HANDLER_MEM_ALLOCATION_FAILED, 5429
ER_VALIDATE_PWD_STRONG_POLICY_DICT_FILE_UNSPECIFIED, 5429
ER_VALIDATE_PWD_VARIABLE_REGISTRATION_FAILED, 5429
ER_VALIDATE_PWD_VARIABLE_UNREGISTRATION_FAILED, 5429
ER_VALUES_IS_NOT_INT_TYPE_ERROR, 5143
ER_VARIABLE_IS_NOT_STRUCT, 5117
ER_VARIABLE_IS_READONLY, 5138
ER_VARIABLE_NOT_PERSISTED, 5177
ER_VARIABLE_NOT_SETTABLE_IN_SF_OR_TRIGGER, 5148
ER_VARIABLE_NOT_SETTABLE_IN_SP, 5154
ER_VARIABLE_NOT_SETTABLE_IN_TRANSACTION, 5148
ER_VAR_CANT_BE_READ, 5114
ER_VAR_DOES_NOT_EXIST, 5184
ER_VERBOSE_HINT, 5235
ER_VERBOSE_REQUIRES_HELP, 5228
ER_VIEW_CHECKSUM, 5124
ER_VIEW_CHECK_FAILED, 5123
ER_VIEW_CREATION_CTX_NOT_SET, 5324
ER_VIEW_DELETE_MERGE_VIEW, 5124
ER_VIEW_FRM_NO_USER, 5128
ER_VIEW_INVALID, 5122
ER_VIEW_INVALID_CREATION_CTX, 5137
ER_VIEW_MULTIUPDATE, 5124
ER_VIEW_NONUPD_CHECK, 5123
ER_VIEW_NO_CREATION_CTX, 5136
ER_VIEW_NO_EXPLAIN, 5121
ER_VIEW_NO_INSERT_FIELD_LIST, 5124
ER_VIEW_OTHER_USER, 5128
ER_VIEW_PREVENT_UPDATE, 5127
ER_VIEW_RECURSIVE, 5129
ER_VIEW_SELECT_CLAUSE, 5122
ER_VIEW_SELECT_TMPTABLE, 5122
ER_VIEW_SELECT_VARIABLE, 5122
ER_VIEW_UNKNOWN_CHARSET_OR_COLLATION, 5222
ER_VIEW_UNPARSABLE, 5257
ER_VIEW_WRONG_LIST, 5122
ER_VTOKEN_PLUGIN_TOKEN_MISMATCH, 5168
ER_VTOKEN_PLUGIN_TOKEN_NOT_FOUND, 5168
ER_WAITPID_FAILED, 5340
ER_WARNING_AUTHCACHE_INVALID_USER_ATTRIBUTES, 5581
ER_WARNING_DISCARD_OLD_PASSWORD_CLAUSE_VOID, 5581
ER_WARNING_NOT_COMPLETE_ROLLBACK, 5112
ER_WARNING_NOT_COMPLETE_ROLLBACK_WITH_CREATED_TEMP_TABLE, 5147
ER_WARNING_NOT_COMPLETE_ROLLBACK_WITH_DROPPED_TEMP_TABLE, 5147
ER_WARNING_PASSWORD_HISTORY_CLAUSES_VOID, 5186

ER_WARNING_RETAIN_CURRENT_PASSWORD_CLAUSE_VOID, 5581
ER_WARN_ALLOWED_PACKET_OVERFLOWED, 5119
ER_WARN_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES, 5603
ER_WARN_BAD_MAX_EXECUTION_TIME, 5167
ER_WARN_BINLOG_PARTIAL_UPDATES_DISABLED, 5187
ER_WARN_BINLOG_PARTIAL_UPDATES_SUGGESTS_PARTIAL_IMAGES, 5187
ER_WARN_BINLOG_V1_ROW_EVENTS_DISABLED, 5187
ER_WARN_CANT_DROP_DEFAULT_KEYCACHE, 5127
ER_WARN_CONFLICTING_HINT, 5167
ER_WARN_DATA_OUT_OF_RANGE, 5116
ER_WARN_DATA_OUT_OF_RANGE_FUNCTIONAL_INDEX, 5199
ER_WARN_DATA_TRUNCATED_FUNCTIONAL_INDEX, 5199
ER_WARN_DA_PRIVILEGE_NOT_REGISTERED, 5219
ER_WARN_DEPRECATED_FLOAT_AUTO_INCREMENT, 5211
ER_WARN_DEPRECATED_FLOAT_DIGITS, 5211
ER_WARN_DEPRECATED_FLOAT_UNSIGNED, 5211
ER_WARN_DEPRECATED_FOUND_ROWS, 5595
ER_WARN_DEPRECATED_INTEGER_DISPLAY_WIDTH, 5211
ER_WARN_DEPRECATED_NESTED_COMMENT_SYNTAX, 5202
ER_WARN_DEPRECATED_SQLMODE, 5164
ER_WARN_DEPRECATED_SQL_CALC_FOUND_ROWS, 5594
ER_WARN_DEPRECATED_SYNTAX, 5118
ER_WARN_DEPRECATED_SYNTAX_NO_REPLACEMENT, 5142
ER_WARN_DEPRECATED_SYNTAX_WITH_VER, 5134
ER_WARN_DEPRECATED_SYSVAR_UPDATE, 5164
ER_WARN_DEPRECATED_USER_SET_EXPR, 5202
ER_WARN_DEPRECATED_UTF8MB3_CHARSET_OPTION, 5574
ER_WARN_DEPRECATED_UTF8MB3_COLLATION, 5202
ER_WARN_DEPRECATED_UTF8MB3_COLLATION_OPTION, 5574
ER_WARN_DEPRECATED_UTF8_ALIAS_OPTION, 5574
ER_WARN_DEPRECATED_ZEROFILL, 5211
ER_WARN_ENGINE_TRANSACTION_ROLLBACK, 5138
ER_WARN_FIELD_RESOLVED, 5117
ER_WARN_FUNC_INDEX_NOT_APPLICABLE, 5216
ER_WARN_HOSTNAME_WONT_WORK, 5117
ER_WARN_INCORRECT_PRIVILEGE_FOR_DB_RESTRICTIONS, 5589
ER_WARN_INDEX_NOT_APPLICABLE, 5146
ER_WARN_INVALID_DB_RESTRICTIONS, 5589
ER_WARN_INVALID_HINT, 5183
ER_WARN_INVALID_TIMESTAMP, 5118
ER_WARN_I_S_SKIPPED_TABLE, 5142
ER_WARN_LEGACY_SYNTAX_CONVERTED, 5158
ER_WARN_LOG_PRIVILEGE_CHECKS_USER_CORRUPT, 5602
ER_WARN_LOG_PRIVILEGE_CHECKS_USER_DOES_NOT_EXIST, 5602
ER_WARN_LOG_PRIVILEGE_CHECKS_USER_NEEDS_RPL_APPLIER_PRIV, 5602
ER_WARN_NO_SERVERID_SPECIFIED, 5323
ER_WARN_NULL_TO_NOTNULL, 5116
ER_WARN_ONLY_MASTER_LOG_FILE_NO_POS, 5159
ER_WARN_ON_MODIFYING_GTID_EXECUTED_TABLE, 5167
ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO, 5159
ER_WARN_OPTIMIZER_HINT_SYNTAX_ERROR, 5167
ER_WARN_PARTIAL_REVOKE_AND_DB_GRANT, 5588
ER_WARN_PROPERTY_STRING_PARSE_FAILED, 5583
ER_WARN_PURGE_LOG_IN_USE, 5156

ER_WARN_PURGE_LOG_IS_ACTIVE, 5156
ER_WARN_QC_RESIZE, 5117
ER_WARN_REMOVED_SQL_MODE, 5575
ER_WARN_RESERVED_SRID_RANGE, 5195
ER_WARN_RPL_RECOVERY_NO_ROTATE_EVENT_FROM_MASTER_EOF, 5603
ER_WARN_SRS_ID_ALREADY_EXISTS, 5194
ER_WARN_SRS_NOT_FOUND, 5174
ER_WARN_SRS_NOT_FOUND_AXIS_ORDER, 5178
ER_WARN_TOO_FEW_RECORDS, 5116
ER_WARN_TOO_MANY_RECORDS, 5116
ER_WARN_TRIGGER_DOESNT_HAVE_CREATED, 5158
ER_WARN_UNKNOWN_KEYRING_AWS_REGION, 5602
ER_WARN_UNKNOWN_QB_NAME, 5167
ER_WARN_UNLOAD_THE_NOT_PERSISTED, 5176
ER_WARN_UNSUPPORTED_HINT, 5174
ER_WARN_UNSUPPORTED_MAX_EXECUTION_TIME, 5167
ER_WARN_USING_OTHER_HANDLER, 5116
ER_WARN_VIEW_MERGE, 5122
ER_WARN_VIEW_WITHOUT_KEY, 5122
ER_WARN_WRONG_COMPRESSION_ALGORITHM_LOG, 5601
ER_WARN_WRONG_COMPRESSION_LEVEL_LOG, 5601
ER_WASTEFUL_NET_BUFFER_SIZE, 5229
ER_WILDCARD_NOT_ALLOWED_FOR_MULTIADDRESS_BIND, 5593
ER_WINDOW_CIRCULARITY_IN_WINDOW_GRAPH, 5180
ER_WINDOW_DUPLICATE_NAME, 5181
ER_WINDOW_EXPLAIN_JSON, 5182
ER_WINDOW_FRAME_END_ILLEGAL, 5180
ER_WINDOW_FRAME_ILLEGAL, 5180
ER_WINDOW_FRAME_START_ILLEGAL, 5180
ER_WINDOW_FUNCTION_IGNORES_FRAME, 5182
ER_WINDOW_ILLEGAL_ORDER_BY, 5181
ER_WINDOW_INVALID_WINDOW_FUNC_ALIAS_USE, 5181
ER_WINDOW_INVALID_WINDOW_FUNC_USE, 5181
ER_WINDOW_NESTED_WINDOW_FUNC_USE_IN_WINDOW_SPEC, 5181
ER_WINDOW_NO_CHILD_PARTITIONING, 5180
ER_WINDOW_NO_GROUP_ORDER, 5181
ER_WINDOW_NO_GROUP_ORDER_UNUSED, 5182
ER_WINDOW_NO_INHERIT_FRAME, 5180
ER_WINDOW_NO_REDEFINE_ORDER_BY, 5180
ER_WINDOW_NO_SUCH_WINDOW, 5180
ER_WINDOW_RANGE_BOUND_NOT_CONSTANT, 5181
ER_WINDOW_RANGE_FRAME_NUMERIC_TYPE, 5181
ER_WINDOW_RANGE_FRAME_ORDER_TYPE, 5180
ER_WINDOW_RANGE_FRAME_TEMPORAL_TYPE, 5180
ER_WINDOW_ROWS_INTERVAL_USE, 5181
ER_WINDOW_SE_NOT_ACCEPTABLE, 5182
ER_WIN_LISTEN_BUT_HOW, 5233
ER_WIN_LOAD_LIBRARY_FAILED, 5340
ER_WL9236_NOW_UNUSED, 5182
ER_WRITABLE_CONFIG_REMOVED, 5231
ER_WRITE_ROW_TO_PARTITION_FAILED, 5329
ER_WRONG_ARGUMENTS, 5113
ER_WRONG_AUTO_KEY, 5103
ER_WRONG_COLUMN_NAME, 5110

ER_WRONG_COMPRESSION_ALGORITHM_CLIENT, 5218
ER_WRONG_COMPRESSION_LEVEL_CLIENT, 5218
ER_WRONG_COUNT_FOR_KEY, 5300
ER_WRONG_COUNT_FOR_ORIGIN, 5300
ER_WRONG_COUNT_OF_ELEMENTS, 5300
ER_WRONG_DATETIME_SPEC, 5231
ER_WRONG_DB_NAME, 5105
ER_WRONG_EXPR_IN_PARTITION_FUNC_ERROR, 5130
ER_WRONG_FIELD_SPEC, 5103
ER_WRONG_FIELD_TERMINATORS, 5104
ER_WRONG_FIELD_WITH_GROUP, 5102
ER_WRONG_FIELD_WITH_GROUP_V2, 5164
ER_WRONG_FILE_NAME, 5167
ER_WRONG_FK_DEF, 5115
ER_WRONG_FK_OPTION_FOR_GENERATED_COLUMN, 5165
ER_WRONG_GROUP_FIELD, 5102
ER_WRONG_JSON_TABLE_VALUE, 5189
ER_WRONG_KEY_COLUMN, 5110
ER_WRONG_KEY_COLUMN_FUNCTIONAL_INDEX, 5200
ER_WRONG_LOCK_OF_SYSTEM_TABLE, 5126
ER_WRONG_MRG_TABLE, 5110
ER_WRONG_MVI_VALUE, 5216
ER_WRONG_NAME_FOR_CATALOG, 5117
ER_WRONG_NAME_FOR_INDEX, 5117
ER_WRONG_NATIVE_TABLE_STRUCTURE, 5142
ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT, 5113
ER_WRONG_OBJECT, 5122
ER_WRONG_OUTER_JOIN, 5107
ER_WRONG_OUTER_JOIN_UNUSED, 5107
ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT, 5136
ER_WRONG_PARAMCOUNT_TO_PROCEDURE, 5106
ER_WRONG_PARAMETERS_TO_NATIVE_FCT, 5136
ER_WRONG_PARAMETERS_TO_PROCEDURE, 5106
ER_WRONG_PARAMETERS_TO_STORED_FCT, 5136
ER_WRONG_PARTITION_NAME, 5135
ER_WRONG_PERFSCHEMA_USAGE, 5142
ER_WRONG_SIZE_NUMBER, 5133
ER_WRONG_SPVAR_TYPE_IN_LIMIT, 5143
ER_WRONG_SRID_FOR_COLUMN, 5187
ER_WRONG_STRING_LENGTH, 5129
ER_WRONG_SUB_KEY, 5104
ER_WRONG_SUM_SELECT, 5102
ER_WRONG_TABLESPACE_NAME, 5166
ER_WRONG_TABLE_NAME, 5105
ER_WRONG_TYPE_COLUMN_VALUE_ERROR, 5140
ER_WRONG_TYPE_FOR_COLUMN_PREFIX_IDX_FLD, 5339
ER_WRONG_TYPE_FOR_VAR, 5114
ER_WRONG_USAGE, 5113
ER_WRONG_VALUE, 5133
ER_WRONG_VALUE_COUNT, 5102
ER_WRONG_VALUE_COUNT_ON_ROW, 5108
ER_WRONG_VALUE_FOR_TYPE, 5125
ER_WRONG_VALUE_FOR_VAR, 5114
ER_X509_CANT_CHMOD_KEY, 5251

ER_X509_CANT_CREATE_CERT, 5251
ER_X509_CANT_READ_CA_CERT, 5251
ER_X509_CANT_READ_CA_KEY, 5251
ER_X509_CANT_WRITE_CERT, 5251
ER_X509_CANT_WRITE_KEY, 5251
ER_X509_CIPHERS_MISMATCH, 5250
ER_X509_ISSUER_MISMATCH, 5250
ER_X509_NEEDS_RSA_PRIVKEY, 5251
ER_X509_SUBJECT_MISMATCH, 5250
ER_XAER_DUPID, 5127
ER_XAER_INVAL, 5124
ER_XAER_NOTA, 5124
ER_XAER_OUTSIDE, 5124
ER_XAER_RMERR, 5125
ER_XAER_RMFAIL, 5124
ER_XA_CANT_CREATE_MDL_BACKUP, 5600
ER_XA_COMMITTING_XID, 5244
ER_XA_IGNORING_XID, 5243
ER_XA_NO_MULTI_2PC_HEURISTIC_RECOVER, 5244
ER_XA_RBDEADLOCK, 5137
ER_XA_RBROLLBACK, 5125
ER_XA_RBTIMEOUT, 5137
ER_XA_RECOVERY_DONE, 5244
ER_XA_RECOVER_EXPLANATION, 5244
ER_XA_RECOVER_FOUND_TRX_IN_SE, 5243
ER_XA_RECOVER_FOUND_XA_TRX, 5243
ER_XA_REPLICATION_FILTERS, 5215
ER_XA_RETRY, 5172
ER_XA_ROLLING_BACK_XID, 5244
ER_XA_STARTING_RECOVERY, 5244
ER_XPLUGIN_ALL_IO_INTERFACES_DISABLED, 5368
ER_XPLUGIN_BUFFER_PAGE_ALLOC_FAILED, 5365
ER_XPLUGIN_CAPABILITY_CLIENT_INTERACTIVE_FAILED, 5372
ER_XPLUGIN_CAPABILITY_EXPIRED_PASSWORD, 5372
ER_XPLUGIN_CLIENT_KILL_MSG, 5370
ER_XPLUGIN_CLIENT_RELEASE_TRIGGERED, 5370
ER_XPLUGIN_CLIENT_SSL_HANDSHAKE_FAILED, 5367
ER_XPLUGIN_CLOSING_CLIENTS_ON_SHUTDOWN, 5368
ER_XPLUGIN_COMPRESSION_ERROR, 5601
ER_XPLUGIN_DETECTED_HANGING_CLIENTS, 5365
ER_XPLUGIN_EMPTY_ADMIN_CMD, 5371
ER_XPLUGIN_ERROR_MSG, 5361
ER_XPLUGIN_ERROR_READING_SOCKET, 5368
ER_XPLUGIN_ERROR_WRITING_TO_CLIENT, 5369
ER_XPLUGIN_EXCEPTION_IN_EVENT_LOOP, 5366
ER_XPLUGIN_EXCEPTION_IN_TASK_SCHEDULER, 5366
ER_XPLUGIN_EXISTING_USER_ACCOUNT_WITH_INCOMPLETE_GRANTS, 5370
ER_XPLUGIN_FAILED_AT_SSL_CONF, 5367
ER_XPLUGIN_FAILED_TO_ACCEPT_CLIENT, 5365
ER_XPLUGIN_FAILED_TO_CLOSE_SQL_SESSION, 5371
ER_XPLUGIN_FAILED_TO_CREATE_SESSION_FOR_CONN, 5367
ER_XPLUGIN_FAILED_TO_EXECUTE_ADMIN_CMD, 5371
ER_XPLUGIN_FAILED_TO_GET_CREATION_STMT, 5371
ER_XPLUGIN_FAILED_TO_GET_ENGINE_INFO, 5371

ER_XPLUGIN_FAILED_TO_GET_SECURITY_CTX, 5370
ER_XPLUGIN_FAILED_TO_GET_SYS_VAR, 5371
ER_XPLUGIN_FAILED_TO_INITIALIZE_SESSION, 5367
ER_XPLUGIN_FAILED_TO_INTERRUPT_SESSION, 5370
ER_XPLUGIN_FAILED_TO_OPEN_INTERNAL_SESSION, 5372
ER_XPLUGIN_FAILED_TO_PREPARE_IO_INTERFACES, 5366
ER_XPLUGIN_FAILED_TO_RESET_IPV6_V6ONLY_FLAG, 5372
ER_XPLUGIN_FAILED_TO_SCHEDULE_CLIENT, 5365
ER_XPLUGIN_FAILED_TO_SET_MIN_NUMBER_OF_WORKERS, 5367
ER_XPLUGIN_FAILED_TO_SET_SO_REUSEADDR_FLAG, 5372
ER_XPLUGIN_FAILED_TO_STOP_SERVICES, 5561
ER_XPLUGIN_FAILED_TO_SWITCH_CONTEXT, 5372
ER_XPLUGIN_FAILED_TO_SWITCH_SECURITY_CTX, 5573
ER_XPLUGIN_FAILED_TO_SWITCH_SECURITY_CTX_TO_ROOT, 5371
ER_XPLUGIN_FAILED_TO_UNREGISTER_UDF, 5372
ER_XPLUGIN_FAILED_TO_USE_SSL_CONF, 5361
ER_XPLUGIN_FAIL_TO_GET_RESULT_DATA, 5371
ER_XPLUGIN_FORCE_STOP_CLIENT, 5365
ER_XPLUGIN_GET_PEER_ADDRESS_FAILED, 5372
ER_XPLUGIN_INVALID_AUTH_METHOD, 5368
ER_XPLUGIN_INVALID_MSG_DURING_CLIENT_INIT, 5368
ER_XPLUGIN_IPv6_AVAILABLE, 5370
ER_XPLUGIN_LISTENER_SETUP_FAILED, 5366
ER_XPLUGIN_LISTENER_STATUS_MSG, 5369
ER_XPLUGIN_LISTENER_SYS_VARIABLE_ERROR, 5369
ER_XPLUGIN_MAX_AUTH_ATTEMPTS_REACHED, 5365
ER_XPLUGIN_MESSAGE_TOO_LONG, 5367
ER_XPLUGIN_PEER_DISCONNECTED_WHILE_READING_MSG_BODY, 5368
ER_XPLUGIN_READ_FAILED_CLOSING_CONNECTION, 5368
ER_XPLUGIN_REFERENCE_TO_SECURE_CONN_WITH_XPLUGIN, 5361
ER_XPLUGIN_REFERENCE_TO_USER_ACCOUNT_DOC_SECTION, 5366
ER_XPLUGIN_RETRYING_BIND_ON_PORT, 5369
ER_XPLUGIN_SCHEDULER_STARTED, 5369
ER_XPLUGIN_SCHEDULER_STOPPED, 5369
ER_XPLUGIN_SERVER_EXITED, 5360
ER_XPLUGIN_SERVER_EXITING, 5360
ER_XPLUGIN_SERVER_STARTS_HANDLING_CONNECTIONS, 5370
ER_XPLUGIN_SERVER_STOPPED_HANDLING_CONNECTIONS, 5370
ER_XPLUGIN_SHUTDOWN_TRIGGERED, 5369
ER_XPLUGIN_SRV_SESSION_INIT_THREAD_FAILED, 5366
ER_XPLUGIN_SSL_HANDSHAKE_WITH_SERVER_FAILED, 5367
ER_XPLUGIN_STARTUP_FAILED, 5360
ER_XPLUGIN_TASK_SCHEDULING_FAILED, 5366
ER_XPLUGIN_UNABLE_TO_ACCEPT_CONNECTION, 5368
ER_XPLUGIN_UNABLE_TO_USE_USER_SESSION_ACCOUNT, 5366
ER_XPLUGIN_UNEXPECTED_EXCEPTION_DISPATCHING_CMD, 5366
ER_XPLUGIN_UNEXPECTED_MSG_DURING_AUTHENTICATION, 5369
ER_XPLUGIN_UNINITIALIZED_MESSAGE, 5367
ER_XPLUGIN_UNIX_SOCKET_NOT_CONFIGURED, 5370
ER_XPLUGIN_USER_ACCOUNT_WITH_ALL_PERMISSIONS, 5369
ER_XPLUGIN_USING_SSL_CONF_FROM_MYSQLX, 5360
ER_XPLUGIN_USING_SSL_CONF_FROM_SERVER, 5360
ER_XPLUGIN_USING_SSL_FOR_TLS_CONNECTION, 5361
ER_XPLUING_NET_STARTUP_FAILED, 5367

ER_YES, 5098
ER_ZLIB_Z_BUF_ERROR, 5116
ER_ZLIB_Z_DATA_ERROR, 5116
ER_ZLIB_Z_MEM_ERROR, 5116
WARN_COND_ITEM_TRUNCATED, 5139
WARN_DATA_TRUNCATED, 5116
WARN_DATA_TRUNCATED_FUNCTIONAL_INDEX, 5199
WARN_NAMED_PIPE_ACCESS_EVERYONE, 5157
WARN_NON_ASCII_SEPARATOR_NOT_IMPLEMENTED, 5139
WARN_NO_MASTER_INFO, 5137
WARN_ON_BLOCKHOLE_IN_RBR, 5156
WARN_OPTION_BELOW_LIMIT, 5144
WARN_OPTION_IGNORED, 5137
WARN_PLUGIN_BUSY, 5138
WARN_UNENCRYPTED_TABLE_IN_ENCRYPTED_DB, 5207
WARN_USELESS_SPATIAL_INDEX, 5190
ERROR Events (NDB Cluster), 4094
error log, 6590
error logs (NDB Cluster), 3954
error messages
can't find file, 5631
Can't reopen table, 5647
displaying, 577
languages, 1795, 1795
The used command is not allowed with this MySQL version, 1075
error-insert option
ndb_move_data, 4012
errors
access denied, 5618
and replication, 3381
checking tables for, 1439
common, 5617
directory checksum, 187
handling for UDFs, 5028
in subqueries, 2451
known, 5648
linking, 4790
list of, 5618
lost connection, 5622
reporting, 52, 52
sources of information, 5096
error_count system variable, 736
ERROR_FOR_DIVISION_BY_ZERO SQL mode, 916
error_messages
ndbinfo table, 4138
ER_ABORTING error code, 5232
ER_ABORTING_CONNECTION error code, 5109
ER_ABORTING_USER_CONNECTION error code, 5323
ER_ACCESS_DENIED_CHANGE_USER_ERROR error code, 5156
ER_ACCESS_DENIED_ERROR error code, 5101
ER_ACCESS_DENIED_ERROR_WITHOUT_PASSWORD error code, 5324
ER_ACCESS_DENIED_ERROR_WITH_PASSWORD error code, 5324
ER_ACCESS_DENIED_FOR_USER_ACCOUNT_LOCKED error code, 5325
ER_ACCESS_DENIED_NO_PASSWORD_ERROR error code, 5143

ER_ACCOUNT_HAS_BEEN_LOCKED error code, 5166
ER_ACL_OPERATION_FAILED error code, 5173
ER_ACL_WRONG_OR_MISSING_ACL_TABLES_LOG error code, 5596
ER_ADDRESSES_FOR_HOSTNAME_HEADER error code, 5226
ER_ADDRESSES_FOR_HOSTNAME_LIST_ITEM error code, 5226
ER_ADD_PARTITION_NO_NEW_PARTITION error code, 5132
ER_ADD_PARTITION_SUBPART_ERROR error code, 5132
ER_ADMIN_WRONG_MRG_TABLE error code, 5129
ER_AES_INVALID_IV error code, 5157
ER_AGGREGATE_IN_ORDER_NOT_SELECT error code, 5162
ER_AGGREGATE_ORDER_FOR_UNION error code, 5159
ER_AGGREGATE_ORDER_NON_AGG_QUERY error code, 5160
ER_ALTER_FILEGROUP_FAILED error code, 5133
ER_ALTER_INFO error code, 5104
ER_ALTER_OPERATION_NOT_SUPPORTED error code, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON error code, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_AUTOINC error code, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_CHANGE_FTS error code, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COLUMN_TYPE error code, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_COPY error code, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_CHECK error code, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FK_RENAME error code, 5154
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_FTS error code, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_GIS error code, 5162
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_HIDDEN_FTS error code, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOPK error code, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_NOT_NULL error code, 5155
ER_ALTER_OPERATION_NOT_SUPPORTED_REASON_PARTITION error code, 5154
ER_AMBIGUOUS_FIELD_TERM error code, 5129
ER_ANONYMOUS_AUTH_ID_NOT_ALLOWED_IN_MANDATORY_ROLES error code, 5329
ER_APPLIER_LOG_EVENT_VALIDATION_ERROR error code, 5186
ER_ATTRIBUTE_IGNORED error code, 5188
ER_AUDIT_API_ABORT error code, 5170
ER_AUDIT_CANT_ABORT_COMMAND error code, 5321
ER_AUDIT_CANT_ABORT_EVENT error code, 5321
ER_AUDIT_LOG_CANNOT_GENERATE_PASSWORD error code, 5603
ER_AUDIT_LOG_CANNOT_READ_PASSWORD error code, 5596
ER_AUDIT_LOG_CANNOT_REMOVE_PASSWORD error code, 5596
ER_AUDIT_LOG_CANNOT_SET_LOG_POLICY_WITH_OTHER_POLICIES error code, 5356
ER_AUDIT_LOG_CANNOT_STORE_PASSWORD error code, 5596
ER_AUDIT_LOG_COULD_NOT_CREATE_AES_KEY error code, 5566
ER_AUDIT_LOG_COULD_NOT_OPEN_FILE_FOR_READING error code, 5596
ER_AUDIT_LOG_EC_WRITER_FAILED_TO_CREATE_FILE error code, 5357
ER_AUDIT_LOG_EC_WRITER_FAILED_TO_INIT_COMPRESSION error code, 5357
ER_AUDIT_LOG_EC_WRITER_FAILED_TO_INIT_ENCRYPTION error code, 5357
ER_AUDIT_LOG_ENCRYPTION_PASSWORD_CANNOT_BE_FETCHED error code, 5566
ER_AUDIT_LOG_ENCRYPTION_PASSWORD_HAS_NOT_BEEN_SET error code, 5566
ER_AUDIT_LOG_FILE_HAS_BEEN_SUCCESSFULLY_PROCESSED error code, 5596
ER_AUDIT_LOG_FILE_NAME_DOES_NOT_HAVE_REQUIRED_FORMAT error code, 5595
ER_AUDIT_LOG_FILE_NAME_KEYRING_ID_VALUE_IS_MISSING error code, 5595
ER_AUDIT_LOG_FILE_NAME_TIMESTAMP_VALUE_IS_MISSING_OR_INVALID error code, 5595
ER_AUDIT_LOG_FILTER_FAILED_TO_CLOSE_TABLE_AFTER_READING error code, 5359
ER_AUDIT_LOG_FILTER_FAILED_TO_DELETE_FROM_TABLE error code, 5358
ER_AUDIT_LOG_FILTER_FAILED_TO_INIT_TABLE_FOR_READ error code, 5358

ER_AUDIT_LOG_FILTER_FAILED_TO_INSERT_INTO_TABLE error code, 5358
ER_AUDIT_LOG_FILTER_FAILED_TO_READ_TABLE error code, 5359
ER_AUDIT_LOG_FILTER_FAILED_TO_STORE_TABLE_FLDS error code, 5358
ER_AUDIT_LOG_FILTER_FAILED_TO_UPDATE_TABLE error code, 5358
ER_AUDIT_LOG_FILTER_FLD_FILTERNAME_CANNOT_BE_EMPTY error code, 5359
ER_AUDIT_LOG_FILTER_INVALID_COLUMN_COUNT error code, 5358
ER_AUDIT_LOG_FILTER_INVALID_COLUMN_DEFINITION error code, 5358
ER_AUDIT_LOG_FILTER_IS_NOT_INSTALLED error code, 5355
ER_AUDIT_LOG_FILTER_RESULT_MSG error code, 5357
ER_AUDIT_LOG_FILTER_USER_AND_HOST_CANNOT_BE_EMPTY error code, 5359
ER_AUDIT_LOG_HAS_NOT_BEEN_INSTALLED error code, 5600
ER_AUDIT_LOG_HOST_NAME_INVALID_CHARACTER error code, 5565
ER_AUDIT_LOG_INDEX_MAP_CANNOT_ACCESS_DIR error code, 5356
ER_AUDIT_LOG_INSUFFICIENT_PRIVILEGE error code, 5217
ER_AUDIT_LOG_INVALID_FILE_CONTENT error code, 5596
ER_AUDIT_LOG_JSON_FILTERING_NOT_ENABLED error code, 5565
ER_AUDIT_LOG_JSON_FILTER_DOES_NOT_EXIST error code, 5566
ER_AUDIT_LOG_JSON_FILTER_NAME_CANNOT_BE_EMPTY error code, 5565
ER_AUDIT_LOG_JSON_FILTER_PARSING_ERROR error code, 5565
ER_AUDIT_LOG_JSON_READER_BUF_TOO_SMALL error code, 5357
ER_AUDIT_LOG_JSON_READER_FAILED_TO_OPEN_FILE error code, 5358
ER_AUDIT_LOG_JSON_READER_FAILED_TO_PARSE error code, 5357
ER_AUDIT_LOG_JSON_READER_FILE_PARSING_ERROR error code, 5358
ER_AUDIT_LOG_JSON_USER_NAME_CANNOT_BE_EMPTY error code, 5566
ER_AUDIT_LOG_KEYRING_ID_TIMESTAMP_VALUE_IS_INVALID error code, 5595
ER_AUDIT_LOG_NO_KEYRING_PLUGIN_INSTALLED error code, 5565
ER_AUDIT_LOG_ONLY_INCLUDE_LIST_USED error code, 5356
ER_AUDIT_LOG_PASSWORD_HAS_BEEN_COPIED error code, 5596
ER_AUDIT_LOG_RENAME_LOG_FILE_BEFORE_FLUSH error code, 5357
ER_AUDIT_LOG_SUPER_PRIVILEGE_REQUIRED error code, 5600
ER_AUDIT_LOG_SWITCHING_TO_INCLUDE_LIST error code, 5356
ER_AUDIT_LOG_TABLE_DEFINITION_NOT_UPDATED error code, 5567
ER_AUDIT_LOG_UDF_INSUFFICIENT_PRIVILEGE error code, 5565
ER_AUDIT_LOG_UDF_INVALID_ARGUMENT_COUNT error code, 5600
ER_AUDIT_LOG_UDF_INVALID_ARGUMENT_TYPE error code, 5600
ER_AUDIT_LOG_UDF_READ_INVALID_MAX_ARRAY_LENGTH_ARG_TYPE error code, 5600
ER_AUDIT_LOG_UDF_READ_INVALID_MAX_ARRAY_LENGTH_ARG_VALUE error code, 5173
ER_AUDIT_LOG_USER_FIRST_CHARACTER_MUST_BE_ALPHANUMERIC error code, 5566
ER_AUDIT_LOG_USER_NAME_INVALID_CHARACTER error code, 5565
ER_AUDIT_LOG_WRITER_DEST_FILE_ALREADY_EXISTS error code, 5356
ER_AUDIT_LOG_WRITER_FAILED_TO_WRITE_TO_FILE error code, 5357
ER_AUDIT_LOG_WRITER_INCOMPLETE_FILE_RENAMED error code, 5356
ER_AUDIT_LOG_WRITER_RENAME_FILE_FAILED error code, 5356
ER_AUDIT_LOG_WRITER_RENAME_FILE_FAILED_REMOVE_FILE_MANUALLY error code, 5356
ER_AUDIT_PLUGIN_DOES_NOT_SUPPORT_AUDIT_AUTH_EVENTS error code, 5257
ER_AUDIT_PLUGIN_HAS_INVALID_DATA error code, 5258
ER_AUDIT_WARNING error code, 5321
ER_AUTHCACHE_CANT_INIT_GRANT_SUBSYSTEM error code, 5254
ER_AUTHCACHE_CANT_OPEN_AND_LOCK_PRIVILEGE_TABLES error code, 5254
ER_AUTHCACHE_DB_ENTRY_LOWERCASED_REVOKE_WILL_FAIL error code, 5254
ER_AUTHCACHE_DB_IGNORED_EMPTY_NAME error code, 5254
ER_AUTHCACHE_DB_SKIPPED_NEEDS_RESOLVE error code, 5254
ER_AUTHCACHE_EXPIRED_PASSWORD_UNSUPPORTED error code, 5253
ER_AUTHCACHE_PLUGIN_CONFIG error code, 5253

ER_AUTHCACHE_PLUGIN_MISSING error code, 5252
ER_AUTHCACHE_PROCS_PRIV_ENTRY_IGNORED_BAD_ROUTINE_TYPE error code, 5255
ER_AUTHCACHE_PROCS_PRIV_SKIPPED_NEEDS_RESOLVE error code, 5255
ER_AUTHCACHE_PROXIES_PRIV_SKIPPED_NEEDS_RESOLVE error code, 5252
ER_AUTHCACHE_ROLE_TABLES_DODGY error code, 5253
ER_AUTHCACHE_TABLES_PRIV_SKIPPED_NEEDS_RESOLVE error code, 5255
ER_AUTHCACHE_TABLE_PROXIES_PRIV_MISSING error code, 5254
ER_AUTHCACHE_USER_IGNORED_DEPRECATED_PASSWORD error code, 5253
ER_AUTHCACHE_USER_IGNORED_INVALID_PASSWORD error code, 5253
ER_AUTHCACHE_USER_IGNORED_NEEDS_PLUGIN error code, 5253
ER_AUTHCACHE_USER_SKIPPED_NEEDS_RESOLVE error code, 5253
ER_AUTHCACHE_USER_TABLE_DODGY error code, 5253
ER_AUTH_CANT_ACTIVATE_ROLE error code, 5250
ER_AUTH_CANT_CREATE_RSA_PAIR error code, 5251
ER_AUTH_CANT_SET_DEFAULT_PLUGIN error code, 5238
ER_AUTH_CANT_WRITE_PRIVKEY error code, 5251
ER_AUTH_CANT_WRITE_PUBKEY error code, 5251
ER_AUTH_CERTS_SAVED_TO_DATADIR error code, 5252
ER_AUTH_CERT_GENERATION_DISABLED error code, 5252
ER_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES error code, 5588
ER_AUTH_KEYS_SAVED_TO_DATADIR error code, 5252
ER_AUTH_KEY_GENERATION_DISABLED error code, 5252
ER_AUTH_KEY_GENERATION_SKIPPED_PAIR_PRESENT error code, 5252
ER_AUTH_LDAP_ERROR_LOGGER_ERROR_MSG error code, 5362
ER_AUTH_RSA_CANT_FIND error code, 5250
ER_AUTH_RSA_CANT_PARSE error code, 5250
ER_AUTH_RSA_CANT_READ error code, 5250
ER_AUTH_RSA_CONF_PREVENTS_KEY_GENERATION error code, 5252
ER_AUTH_RSA_FILES_NOT_FOUND error code, 5250
ER_AUTH_SSL_CONF_PREVENTS_CERT_GENERATION error code, 5252
ER_AUTH_USING_EXISTING_CERTS error code, 5252
ER_AUTOINC_READ_FAILED error code, 5129
ER_AUTO_CONVERT error code, 5115
ER_AUTO_INCREMENT_CONFLICT error code, 5156
ER_AUTO_OPTIONS_FAILED error code, 5239
ER_AUTO_POSITION_REQUIRES_GTID_MODE_NOT_OFF error code, 5149
ER_BACK_IN_TIME error code, 5232
ER_BAD_DB_ERROR error code, 5101
ER_BAD_FIELD_ERROR error code, 5102
ER_BAD_FT_COLUMN error code, 5117
ER_BAD_HOST_ERROR error code, 5101
ER_BAD_LOG_STATEMENT error code, 5135
ER_BAD_NULL_ERROR error code, 5101
ER_BAD_NULL_ERROR_NOT_IGNORED error code, 5190
ER_BAD_SLAVE error code, 5112
ER_BAD_SLAVE_AUTO_POSITION error code, 5149
ER_BAD_SLAVE_UNTIL_COND error code, 5117
ER_BAD_TABLE_ERROR error code, 5101
ER_BASE64_DECODE_ERROR error code, 5135
ER_BASEDIR_SET_TO error code, 5327
ER_BEFORE_DML_VALIDATION_ERROR error code, 5165
ER_BEG_INITFILE error code, 5234
ER_BINLOG_ATTACHING_THREAD_MEMORY_FINALLY_AVAILABLE error code, 5311
ER_BINLOG_CACHE_SIZE_GREATER_THAN_MAX error code, 5146

ER_BINLOG_CACHE_SIZE_TOO_LARGE error code, 5322
ER_BINLOG_CANT_APPEND_LOG_TO_TMP_INDEX error code, 5313
ER_BINLOG_CANT_CLEAR_IN_USE_FLAG_FOR_CRASHED_BINLOG error code, 5317
ER_BINLOG_CANT_CLOSE_TMP_INDEX error code, 5314
ER_BINLOG_CANT_COPY_INDEX_TO_TMP error code, 5314
ER_BINLOG_CANT_CREATE_CACHE_FOR_LOG error code, 5311
ER_BINLOG_CANT_DELETE_FILE error code, 5314
ER_BINLOG_CANT_DELETE_FILE_AND_READ_BINLOG_INDEX error code, 5316
ER_BINLOG_CANT_DELETE_LOG_FILE_DOES_INDEX_MATCH_FILES error code, 5316
ER_BINLOG_CANT_FIND_LOG_IN_INDEX error code, 5316
ER_BINLOG_CANT_GENERATE_NEW_FILE_NAME error code, 5313
ER_BINLOG_CANT_LOCATE_OLD_BINLOG_OR_RELAY_LOG_FILES error code, 5314
ER_BINLOG_CANT_MOVE_TMP_TO_INDEX error code, 5314
ER_BINLOG_CANT_OPEN_CRASHED_BINLOG error code, 5317
ER_BINLOG_CANT_OPEN_FOR_LOGGING error code, 5312
ER_BINLOG_CANT_OPEN_LOG error code, 5311
ER_BINLOG_CANT_OPEN_TMP_INDEX error code, 5314
ER_BINLOG_CANT_RESIZE_CACHE error code, 5311
ER_BINLOG_CANT_SET_TMP_INDEX_NAME error code, 5314
ER_BINLOG_CANT_TRIM_CRASHED_BINLOG error code, 5317
ER_BINLOG_CANT_USE_FOR_LOGGING error code, 5313
ER_BINLOG_CRASHED_BINLOG_TRIMMED error code, 5317
ER_BINLOG_CRASH_RECOVERY_FAILED error code, 5317
ER_BINLOG_CREATE_ROUTINE_NEED_SUPER error code, 5126
ER_BINLOG_END error code, 5232
ER_BINLOG_ERROR_GETTING_NEXT_LOG_FROM_INDEX error code, 5314
ER_BINLOG_ERROR_READING_GTIDS_FROM_BINARY_LOG error code, 5312
ER_BINLOG_ERROR_READING_GTIDS_FROM_RELAY_LOG error code, 5312
ER_BINLOG_EVENTS_READ_FROM_BINLOG_INFO error code, 5313
ER_BINLOG_EVENTS_READ_FROM_RELAY_LOG_INFO error code, 5312
ER_BINLOG_EVENT_WRITE_TO_STMT_CACHE_FAILED error code, 5328
ER_BINLOG_EXPIRAY_LOG_DAYS_AND_SECS_USED_TOGETHER error code, 5342
ER_BINLOG_EXPIRE_LOG_DAYS_AND_SECS_USED_TOGETHER error code, 5191
ER_BINLOG_FAILED_TO_CLOSE_INDEX_FILE_WHILE_REBUILDING error code, 5313
ER_BINLOG_FAILED_TO_DELETE_INDEX_FILE_WHILE_REBUILDING error code, 5313
ER_BINLOG_FAILED_TO_DELETE_LOG_FILE error code, 5316
ER_BINLOG_FAILED_TO_OPEN_INDEX_FILE_AFTER_REBUILDING error code, 5313
ER_BINLOG_FAILED_TO_OPEN_REGISTER_FILE error code, 5315
ER_BINLOG_FAILED_TO_OPEN_TEMPORARY_INDEX_FILE error code, 5314
ER_BINLOG_FAILED_TO_READ_REGISTER_FILE error code, 5316
ER_BINLOG_FAILED_TO_REINIT_REGISTER_FILE error code, 5316
ER_BINLOG_FAILED_TO_RENAME_INDEX_FILE_WHILE_REBUILDING error code, 5313
ER_BINLOG_FAILED_TO_RUN_AFTER_FLUSH_HOOK error code, 5317
ER_BINLOG_FAILED_TO_RUN_AFTER_SYNC_HOOK error code, 5317
ER_BINLOG_FAILED_TO_SET_PURGE_INDEX_FILE_NAME error code, 5315
ER_BINLOG_FAILED_TO_SYNC_INDEX_FILE error code, 5312
ER_BINLOG_FAILED_TO_SYNC_INDEX_FILE_IN_OPEN error code, 5313
ER_BINLOG_FAILED_TO_WRITE_DROP_FOR_TEMP_TABLES error code, 5301
ER_BINLOG_FATAL_ERROR error code, 5136
ER_BINLOG_FILE_BEING_READ_NOT_PURGED error code, 5311
ER_BINLOG_FILE_EXTENSION_NUMBER_EXHAUSTED error code, 5312
ER_BINLOG_FILE_EXTENSION_NUMBER_RUNNING_LOW error code, 5312
ER_BINLOG_FILE_NAME_TOO_LONG error code, 5312
ER_BINLOG_FILE_OPEN_FAILED error code, 5328

ER_BINLOG_IO_ERROR_READING_HEADER error code, 5311
ER_BINLOG_LOGGING_IMPOSSIBLE error code, 5136
ER_BINLOG_LOGGING_INCIDENT_TO_STOP_SLAVES error code, 5316
ER_BINLOG_LOGGING_NOT_POSSIBLE error code, 5341
ER_BINLOG_LOGICAL_CORRUPTION error code, 5156
ER_BINLOG_MALFORMED_OR_OLD_RELAY_LOG error code, 5342
ER_BINLOG_MASTER_KEY_RECOVERY_OUT_OF_COMBINATION error code, 5205
ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_AUX_KEY error code, 5206
ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_UNUSED_KEYS error code, 5206
ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_OPERATE_KEY error code, 5205
ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_REENCRYPT_LOG error code, 5206
ER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_ROTATE_LOGS error code, 5205
ER_BINLOG_MULTIPLE_ENGINES_AND_SELF_LOGGING_ENGINE error code, 5141
ER_BINLOG_NEEDS_SERVERID error code, 5237
ER_BINLOG_OOM_WRITING_DELETE_WHILE_OPENING_HEAP_TABLE error code, 5301
ER_BINLOG_PURGE_EMFILE error code, 5136
ER_BINLOG_PURGE_FATAL_ERR error code, 5123
ER_BINLOG_PURGE_LOGS_CALLED_WITH_FILE_NOT_IN_INDEX error code, 5315
ER_BINLOG_PURGE_LOGS_CANT_COPY_TO_REGISTER_FILE error code, 5315
ER_BINLOG_PURGE_LOGS_CANT_FLUSH_REGISTER_FILE error code, 5315
ER_BINLOG_PURGE_LOGS_CANT_SYNC_INDEX_FILE error code, 5315
ER_BINLOG_PURGE_LOGS_CANT_UPDATE_INDEX_FILE error code, 5315
ER_BINLOG_PURGE_LOGS_FAILED_TO_PURGE_LOG error code, 5315
ER_BINLOG_PURGE_PROHIBITED error code, 5123
ER_BINLOG_RECOVERING_AFTER_CRASH_USING error code, 5316
ER_BINLOG_ROW_ENGINE_AND_STMT_ENGINE error code, 5140
ER_BINLOG_ROW_INJECTION_AND_STMT_ENGINE error code, 5141
ER_BINLOG_ROW_INJECTION_AND_STMT_MODE error code, 5141
ER_BINLOG_ROW_LOGGING_FAILED error code, 5133
ER_BINLOG_ROW_MODE_AND_STMT_ENGINE error code, 5140
ER_BINLOG_ROW_VALUE_OPTION_IGNORED error code, 5559
ER_BINLOG_ROW_VALUE_OPTION_USED_ONLY_FOR_AFTER_IMAGES error code, 5560
ER_BINLOG_STMT_CACHE_SIZE_GREATER_THAN_MAX error code, 5147
ER_BINLOG_STMT_CACHE_SIZE_TOO_LARGE error code, 5322
ER_BINLOG_STMT_MODE_AND_NO_REPL_TABLES error code, 5154
ER_BINLOG_STMT_MODE_AND_ROW_ENGINE error code, 5141
ER_BINLOG_UNABLE_TO_ROTATE_GTID_TABLE_READONLY error code, 5592
ER_BINLOG_UNSAFE_AND_STMT_ENGINE error code, 5141
ER_BINLOG_UNSAFE_AUTOINC_COLUMNS error code, 5141
ER_BINLOG_UNSAFE_AUTOINC_NOT_FIRST error code, 5145
ER_BINLOG_UNSAFE_CREATE_IGNORE_SELECT error code, 5145
ER_BINLOG_UNSAFE_CREATE_REPLACE_SELECT error code, 5145
ER_BINLOG_UNSAFE_CREATE_SELECT_AUTOINC error code, 5145
ER_BINLOG_UNSAFE_DEFAULT_EXPRESSION_IN_SUBSTATEMENT error code, 5595
ER_BINLOG_UNSAFE_FULLTEXT_PLUGIN error code, 5158
ER_BINLOG_UNSAFE_INSERT_IGNORE_SELECT error code, 5144
ER_BINLOG_UNSAFE_INSERT_SELECT_UPDATE error code, 5144
ER_BINLOG_UNSAFE_INSERT_TWO_KEYS error code, 5145
ER_BINLOG_UNSAFE_LIMIT error code, 5141
ER_BINLOG_UNSAFE_MESSAGE_AND_STATEMENT error code, 5323
ER_BINLOG_UNSAFE_MIXED_STATEMENT error code, 5143
ER_BINLOG_UNSAFE_MULTIPLE_ENGINES_AND_SELF_LOGGING_ENGINE error code, 5143
ER_BINLOG_UNSAFE_NONTRANS_AFTER_TRANS error code, 5141
ER_BINLOG_UNSAFE_NOWAIT error code, 5179

ER_BINLOG_UNSAFE_REPLACE_SELECT error code, 5145
ER_BINLOG_UNSAFE_ROUTINE error code, 5126
ER_BINLOG_UNSAFE_SKIP_LOCKED error code, 5179
ER_BINLOG_UNSAFE_STATEMENT error code, 5136
ER_BINLOG_UNSAFE_SYSTEM_FUNCTION error code, 5141
ER_BINLOG_UNSAFE_SYSTEM_TABLE error code, 5141
ER_BINLOG_UNSAFE_SYSTEM_VARIABLE error code, 5141
ER_BINLOG_UNSAFE_UDF error code, 5141
ER_BINLOG_UNSAFE_UPDATE_IGNORE error code, 5145
ER_BINLOG_UNSAFE_WRITE_AUTOINC_SELECT error code, 5145
ER_BINLOG_UNSAFE_XA error code, 5172
ER_BINLOG_USE_V1_ROW_EVENTS_IGNORED error code, 5560
ER_BINLOG_WARNING_SUPPRESSED error code, 5318
ER_BLOBS_AND_NO_TERMINATED error code, 5104
ER_BLOB_CANT_HAVE_DEFAULT error code, 5105
ER_BLOB_FIELD_IN_PART_FUNC_ERROR error code, 5131
ER_BLOB_KEY_WITHOUT_LENGTH error code, 5110
ER_BLOB_USED_AS_KEY error code, 5103
ER_BOOST_GEOMETRY_CENTROID_EXCEPTION error code, 5160
ER_BOOST_GEOMETRY_EMPTY_INPUT_EXCEPTION error code, 5160
ER_BOOST_GEOMETRY_INCONSISTENT_TURNS_EXCEPTION error code, 5167
ER_BOOST_GEOMETRY_OVERLAY_INVALID_INPUT_EXCEPTION error code, 5160
ER_BOOST_GEOMETRY_SELF_INTERSECTION_POINT_EXCEPTION error code, 5161
ER_BOOST_GEOMETRY_TURN_INFO_EXCEPTION error code, 5161
ER_BOOST_GEOMETRY_UNKNOWN_EXCEPTION error code, 5161
ER_BOOTSTRAP_CANT_THREAD error code, 5220
ER_BUFPOOL_RESIZE_INPROGRESS error code, 5170
ER_CACHING_SHA2_PASSWORD_SECOND_PASSWORD_USED_INFORMATION error code, 5581
ER_CALL_ME_LOCALHOST error code, 5232
ER_CANNOT_ADD_FOREIGN error code, 5113
ER_CANNOT_ADD_FOREIGN_BASE_COL_STORED error code, 5172
ER_CANNOT_ADD_FOREIGN_BASE_COL_VIRTUAL error code, 5171
ER_CANNOT_ALTER_SRID_DUE_TO_INDEX error code, 5187
ER_CANNOT_CHANGE_TO_ROOT_DIR error code, 5327
ER_CANNOT_CONVERT_STRING error code, 5211
ER_CANNOT_CREATE_VIRTUAL_INDEX_CONSTRAINT error code, 5171
ER_CANNOT_DISCARD_TEMPORARY_TABLE error code, 5158
ER_CANNOT_DROP_COLUMN_FUNCTIONAL_INDEX error code, 5199
ER_CANNOT_FIND_KEY_IN_KEYRING error code, 5172
ER_CANNOT_GET_SERVER_VERSION_FROM_TABLESPACE_HEADER error code, 5584
ER_CANNOT_GRANT_ROLES_TO_ANONYMOUS_USER error code, 5213
ER_CANNOT_GRANT_SYSTEM_PRIV_TO_MANDATORY_ROLE error code, 5215
ER_CANNOT_LOAD_FROM_TABLE_V2 error code, 5146
ER_CANNOT_LOCK_USER_MANAGEMENT_CACHES error code, 5176
ER_CANNOT_LOG_PARTIAL_DROP_DATABASE_WITH_GTID error code, 5164
ER_CANNOT_SET_DATABASE_ENCRYPTION error code, 5208
ER_CANNOT_SET_LOG_ERROR_SERVICES error code, 5327
ER_CANNOT_SET_SERVER_VERSION_IN_TABLESPACE_HEADER error code, 5584
ER_CANNOT_SET_TABLESPACE_ENCRYPTION error code, 5208
ER_CANNOT_SET_TABLE_ENCRYPTION error code, 5208
ER_CANNOT_USER error code, 5124
ER_CANNOT_USE_ENCRYPTION_CLAUSE error code, 5208
ER_CANT_ACCESS_CAPATH error code, 5239
ER_CANT_AGGREGATE_2COLLATIONS error code, 5116

ER_CANT_AGGREGATE_3COLLATIONS error code, 5116
ER_CANT_AGGREGATE_NCOLLATIONS error code, 5116
ER_CANT_ALLOC_TABLE_OBJECT error code, 5305
ER_CANT_CHANGE_TX_CHARACTERISTICS error code, 5135
ER_CANT_CHECK_PID_PATH error code, 5428
ER_CANT_CHOWN_DATADIR error code, 5238
ER_CANT_CREATE_ADMIN_THREAD error code, 5580
ER_CANT_CREATE_CACHE_FOR_DB_OPT error code, 5306
ER_CANT_CREATE_DB error code, 5099
ER_CANT_CREATE_FILE error code, 5098
ER_CANT_CREATE_GEOMETRY_OBJECT error code, 5126
ER_CANT_CREATE_HANDLER_FILE error code, 5131
ER_CANT_CREATE_HANDLER_OBJECT_FOR_TABLE error code, 5305
ER_CANT_CREATE_HANDLE_MGR_THREAD error code, 5223
ER_CANT_CREATE_INTERRUPT_THREAD error code, 5231
ER_CANT_CREATE_NAMED_PIPES_THREAD error code, 5231
ER_CANT_CREATE_PID_FILE error code, 5229
ER_CANT_CREATE_SCHEDULER_THREAD error code, 5225
ER_CANT_CREATE_SHM_THREAD error code, 5231
ER_CANT_CREATE_SHUTDOWN_THREAD error code, 5229
ER_CANT_CREATE_TABLE error code, 5099
ER_CANT_CREATE_TABLE_SHARE_FROM_FRM error code, 5305
ER_CANT_CREATE_TCPIP_THREAD error code, 5231
ER_CANT_CREATE_TEST_FILE error code, 5229
ER_CANT_CREATE_THREAD error code, 5108
ER_CANT_CREATE_USER_WITH_GRANT error code, 5125
ER_CANT_CREATE_UUID error code, 5228
ER_CANT_DETACH_SESSION_LEFT_BY_PLUGIN error code, 5271
ER_CANT_DO_IMPLICIT_COMMIT_IN_TRX_WHEN_GTID_NEXT_IS_SET error code, 5149
ER_CANT_DO_THIS_DURING_AN_TRANSACTION error code, 5111
ER_CANT_DROP_FIELD_OR_KEY error code, 5104
ER_CANT_ENFORCE_GTID_CONSISTENCY_WITH_ONGOING_GTID_VIOLATING_TX error code, 5166
ER_CANT_EXECUTE_IN_READ_ONLY_TRANSACTION error code, 5150
ER_CANT_FIND_DL_ENTRY error code, 5107
ER_CANT_FIND_SYSTEM_REC error code, 5099
ER_CANT_FIND_UDF error code, 5107
ER_CANT_GET_STAT error code, 5099
ER_CANT_HASH_DO_AND_IGNORE_RULES error code, 5239
ER_CANT_IDENTIFY_CHARSET_USING_DEFAULT error code, 5306
ER_CANT_INCREASE_MAX_OPEN_FILES error code, 5234
ER_CANT_INITIALIZE_BUILTIN_PLUGINS error code, 5237
ER_CANT_INITIALIZE_DYNAMIC_PLUGINS error code, 5237
ER_CANT_INITIALIZE_EARLY_PLUGINS error code, 5237
ER_CANT_INITIALIZE_GTID error code, 5237
ER_CANT_INITIALIZE_UDF error code, 5107
ER_CANT_INIT_DBS error code, 5235
ER_CANT_INIT_SCHEDULER_THREAD error code, 5225
ER_CANT_INIT_TC_LOG error code, 5224
ER_CANT_INIT_TIMER error code, 5239
ER_CANT_JOIN_SHUTDOWN_THREAD error code, 5238
ER_CANT_LOCK error code, 5099
ER_CANT_LOCK_LOG_TABLE error code, 5134
ER_CANT_LOCK_TABLE error code, 5305
ER_CANT_LOCK_TABLESPACE error code, 5305

ER_CANT_MODIFY_SRID_0 error code, 5194
ER_CANT_MODIFY_SRS_USED_BY_COLUMN error code, 5195
ER_CANT_OPEN_AND_LOCK_PRIVILEGE_TABLES error code, 5257
ER_CANT_OPEN_CA error code, 5239
ER_CANT_OPEN_DATADIR_AFTER_UPGRADE_FAILURE error code, 5308
ER_CANT_OPEN_DB_OPT_USING_DEFAULT_CHARSET error code, 5306
ER_CANT_OPEN_DIR error code, 5308
ER_CANT_OPEN_ERROR_LOG error code, 5239
ER_CANT_OPEN_FILE error code, 5099
ER_CANT_OPEN_FRM_FILE error code, 5257
ER_CANT_OPEN_LIBRARY error code, 5107
ER_CANT_OPEN_TABLE_MYSQL_PROC error code, 5307
ER_CANT_PARSE_STORED_ROUTINE_BODY error code, 5307
ER_CANT_PROCESS_EXPRESSION_FOR_GENERATED_COLUMN_TO_DD error code, 5603
ER_CANT_READ_DIR error code, 5099
ER_CANT_READ_ERRMSGS error code, 5235
ER_CANT_READ_FRM_FILE error code, 5257
ER_CANT_READ_TABLE_MYSQL_PROC error code, 5308
ER_CANT_REMOVE_ALL_FIELDS error code, 5104
ER_CANT_REMOVE_PID_FILE error code, 5229
ER_CANT_RENAME_LOG_TABLE error code, 5135
ER_CANT_REOPEN_TABLE error code, 5108
ER_CANT_REPLICATE_ANONYMOUS_WITH_AUTO_POSITION error code, 5564
ER_CANT_REPLICATE_ANONYMOUS_WITH_GTID_MODE_ON error code, 5564
ER_CANT_REPLICATE_GTID_WITH_GTID_MODE_OFF error code, 5564
ER_CANT_RESET_MASTER error code, 5172
ER_CANT_SAVE_GTIDS error code, 5238
ER_CANT_SET_DATADIR error code, 5238
ER_CANT_SET_DATA_DIR error code, 5578
ER_CANT_SET_ERROR_LOG_SERVICE error code, 5193
ER_CANT_SET_ERROR_SUPPRESSION_LIST error code, 5197
ER_CANT_SET_ERROR_SUPPRESSION_LIST_FROM_COMMAND_LINE error code, 5570
ER_CANT_SET_GTID_MODE error code, 5166
ER_CANT_SET_GTID_NEXT_LIST_TO_NON_NULL_WHEN_GTID_MODE_IS_OFF error code, 5149
ER_CANT_SET_GTID_NEXT_TO_ANONYMOUS_WHEN_GTID_MODE_IS_ON error code, 5149
ER_CANT_SET_GTID_NEXT_TO_GTID_WHEN_GTID_MODE_IS_OFF error code, 5149
ER_CANT_SET_GTID_NEXT_WHEN_OWNING_GTID error code, 5150
ER_CANT_SET_GTID_PURGED_DUE_SETS_CONSTRAINTS error code, 5176
ER_CANT_SET_GTID_PURGED_WHEN_GTID_EXECUTED_IS_NOT_EMPTY error code, 5154
ER_CANT_SET_GTID_PURGED_WHEN_OWNED_GTIDS_IS_NOT_EMPTY error code, 5154
ER_CANT_SET_HANDLER_REFERENCE_FOR_TABLE error code, 5305
ER_CANT_SET_PATH_FOR error code, 5308
ER_CANT_SET_PERSISTED error code, 5242
ER_CANT_SET_UP_PERSISTED_VALUES error code, 5238
ER_CANT_SET_VARIABLE_WHEN_OWNING_GTID error code, 5168
ER_CANT_START_ERROR_LOG_SERVICE error code, 5326
ER_CANT_STAT_DATADIR error code, 5238
ER_CANT_STAT_FILE error code, 5316
ER_CANT_UPDATE_TABLE_IN_CREATE_TABLE_SELECT error code, 5147
ER_CANT_UPDATE_USED_TABLE_IN_SF_OR_TRG error code, 5127
ER_CANT_UPDATE_WITH_READLOCK error code, 5114
ER_CANT_UPGRADE_GENERATED_COLUMNS_TO_DD error code, 5306
ER_CANT_USE_AUTO_POSITION_WITH_GTID_MODE_OFF error code, 5166
ER_CANT_USE_OPTION_HERE error code, 5114

ER_CANT_WAIT_FOR_EXECUTED_GTID_SET_WHILE_OWNING_A_GTID error code, 5171
ER_CAPACITY_EXCEEDED error code, 5170
ER_CAPACITY_EXCEEDED_IN_PARSER error code, 5172
ER_CAPACITY_EXCEEDED_IN_RANGE_OPTIMIZER error code, 5171
ER_CA_SELF_SIGNED error code, 5227
ER_CHANGED_ENFORCE_GTID_CONSISTENCY error code, 5223
ER_CHANGED_GTID_MODE error code, 5223
ER_CHANGED_MAX_CONNECTIONS error code, 5234
ER_CHANGED_MAX_OPEN_FILES error code, 5234
ER_CHANGED_TABLE_OPEN_CACHE error code, 5235
ER_CHANGE_MASTER_PASSWORD_LENGTH error code, 5162
ER_CHANGE_MASTER_WRONG_COMPRESSION_ALGORITHM_CLIENT error code, 5218
ER_CHANGE_MASTER_WRONG_COMPRESSION_ALGORITHM_LIST_CLIENT error code, 5218
ER_CHANGE_MASTER_WRONG_COMPRESSION_LEVEL_CLIENT error code, 5218
ER_CHANGE_RPL_INFO_REPOSITORY_FAILURE error code, 5147
ER_CHECKING_TABLE error code, 5305
ER_CHECKREAD error code, 5100
ER_CHECK_CONSTRAINT_CLAUSE_USING_FK_REFER_ACTION_COLUMN error code, 5207
ER_CHECK_CONSTRAINT_DUP_NAME error code, 5207
ER_CHECK_CONSTRAINT_FUNCTION_IS_NOT_ALLOWED error code, 5206
ER_CHECK_CONSTRAINT_NAMED_FUNCTION_IS_NOT_ALLOWED error code, 5206
ER_CHECK_CONSTRAINT_NOT_FOUND error code, 5207
ER_CHECK_CONSTRAINT_REFERS_AUTO_INCREMENT_COLUMN error code, 5207
ER_CHECK_CONSTRAINT_REFERS_UNKNOWN_COLUMN error code, 5207
ER_CHECK_CONSTRAINT_ROW_VALUE error code, 5207
ER_CHECK_CONSTRAINT_VARIABLES error code, 5206
ER_CHECK_CONSTRAINT_VIOLATED error code, 5207
ER_CHECK_NOT_IMPLEMENTED error code, 5111
ER_CHECK_NO_SUCH_TABLE error code, 5111
ER_CLIENT_DOES_NOT_SUPPORT error code, 5186
ER_CLIENT_GTID_UNSAFE_CREATE_DROP_TEMP_TABLE_IN_TRX_IN_SBR error code, 5198
ER_CLIENT_KEYRING_UDF_KEY_INVALID error code, 5219
ER_CLIENT_KEYRING_UDF_KEY_TOO_LONG error code, 5219
ER_CLIENT_KEYRING_UDF_KEY_TYPE_INVALID error code, 5219
ER_CLIENT_KEYRING_UDF_KEY_TYPE_TOO_LONG error code, 5219
ER_CLIENT_PRIVILEGE_CHECKS_USER_CANNOT_BE_ANONYMOUS error code, 5218
ER_CLIENT_PRIVILEGE_CHECKS_USER_CORRUPT error code, 5218
ER_CLIENT_PRIVILEGE_CHECKS_USER_DOES_NOT_EXIST error code, 5218
ER_CLIENT_PRIVILEGE_CHECKS_USER_NEEDS_RPL_APPLIER_PRIV error code, 5219
ER_CLONE_CHARSET error code, 5212
ER_CLONE_CLIENT_TRACE error code, 5578
ER_CLONE_CONFIG error code, 5212
ER_CLONE_CREATE_HANDLER_FAIL_TRACE error code, 5337
ER_CLONE_DDL_IN_PROGRESS error code, 5186
ER_CLONE_DISALLOWED error code, 5213
ER_CLONE_DISK_SPACE error code, 5213
ER_CLONE_DONOR error code, 5212
ER_CLONE_DONOR_TRACE error code, 5577
ER_CLONE_DONOR_VERSION error code, 5212
ER_CLONE_ENCRYPTION error code, 5213
ER_CLONE_HANDLER_EXISTS error code, 5337
ER_CLONE_HANDLER_EXIST_TRACE error code, 5337
ER_CLONE_INFO_CLIENT error code, 5578
ER_CLONE_INFO_SERVER error code, 5578

ER_CLONE_IN_PROGRESS error code, 5213
ER_CLONE_LOOPBACK error code, 5212
ER_CLONE_OS error code, 5212
ER_CLONE_PLATFORM error code, 5212
ER_CLONE_PLUGIN_MATCH error code, 5212
ER_CLONE_PLUGIN_NOT_LOADED error code, 5337
ER_CLONE_PLUGIN_NOT_LOADED_TRACE error code, 5337
ER_CLONE_PROTOCOL error code, 5212
ER_CLONE_PROTOCOL_ERROR error code, 5578
ER_CLONE_PROTOCOL_TRACE error code, 5578
ER_CLONE_REMOTE_ERROR error code, 5577
ER_CLONE_SERVER_TRACE error code, 5578
ER_CLONE_SHUTDOWN_TRACE error code, 5598
ER_CLONE_SYS_CONFIG error code, 5212
ER_CLONE_TOO_MANY_CONCURRENT_CLONES error code, 5186
ER_CMD_NEED_SUPER error code, 5185
ER_COALESCE_ONLY_ON_HASH_PARTITION error code, 5132
ER_COALESCE_PARTITION_NO_PARTITION error code, 5132
ER_COLLATION_CHARSET_MISMATCH error code, 5115
ER_COLUMNACCESS_DENIED_ERROR error code, 5108
ER_COLUMN_CHANGE_SIZE error code, 5214
ER_COLUMN_CHECK_CONSTRAINT_REFERENCES_OTHER_COLUMN error code, 5206
ER_COL_COUNT_DOESNT_MATCH_CORRUPTED_V2 error code, 5151
ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE error code, 5134
ER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE_V2 error code, 5158
ER_COMMIT_NOT_ALLOWED_IN_SF_OR_TRG error code, 5126
ER_COMPONENTS_CANT_ACQUIRE_SERVICE_IMPLEMENTATION error code, 5175
ER_COMPONENTS_CANT_LOAD error code, 5175
ER_COMPONENTS_CANT_RELEASE_SERVICE error code, 5176
ER_COMPONENTS_CANT_SATISFY_DEPENDENCY error code, 5175
ER_COMPONENTS_CANT_UNLOAD error code, 5176
ER_COMPONENTS_FAILED_TO_ACQUIRE_SERVICE_IMPLEMENTATION error code, 5341
ER_COMPONENTS_INFRASTRUCTURE_BOOTSTRAP error code, 5226
ER_COMPONENTS_INFRASTRUCTURE_SHUTDOWN error code, 5227
ER_COMPONENTS_LOAD_CANT_INITIALIZE error code, 5176
ER_COMPONENTS_LOAD_CANT_REGISTER_SERVICE_IMPLEMENTATION error code, 5176
ER_COMPONENTS_NO_SCHEME error code, 5175
ER_COMPONENTS_NO_SCHEME_SERVICE error code, 5175
ER_COMPONENTS_PERSIST_LOADER_BOOTSTRAP error code, 5227
ER_COMPONENTS_UNLOAD_CANT_DEINITIALIZE error code, 5176
ER_COMPONENTS_UNLOAD_CANT_UNREGISTER_SERVICE error code, 5176
ER_COMPONENTS_UNLOAD_DUPLICATE_IN_GROUP error code, 5176
ER_COMPONENTS_UNLOAD_NOT_LOADED error code, 5176
ER_COMPONENT_FILTER_CONFUSED error code, 5363
ER_COMPONENT_FILTER_DIAGNOSTICS error code, 5193
ER_COMPONENT_FILTER_FLABBERGASTED error code, 5184
ER_COMPONENT_FILTER_WRONG_VALUE error code, 5561
ER_COMPONENT_MANIPULATE_ROW_FAILED error code, 5176
ER_COMPONENT_TABLE_INCORRECT error code, 5176
ER_COND_ITEM_TOO_LONG error code, 5139
ER_CONFIG_OPTION_WITHOUT_GROUP error code, 5242
ER_CONFIRMING_THE_FUTURE error code, 5231
ER_CONFLICTING_DECLARATIONS error code, 5119
ER_CONFLICT_FN_PARSE_ERROR error code, 5138

ER_CONNECTION_ABORTED error code, 5560
ER_CONNECTION_HANDLING_OOM error code, 5229
ER_CONNECT_TO_FOREIGN_DATA_SOURCE error code, 5126
ER_CONNECT_TO_MASTER error code, 5113
ER_CONN_ATTR_TRUNCATED error code, 5250
ER_CONN_CONTROL_DELAY_ACTION_INIT_FAILED error code, 5364
ER_CONN_CONTROL_ERROR_MSG error code, 5362
ER_CONN_CONTROL_EVENT_COORDINATOR_INIT_FAILED error code, 5364
ER_CONN_CONTROL_FAILED_TO_SET_CONN_DELAY error code, 5365
ER_CONN_CONTROL_FAILED_TO_UPDATE_CONN_DELAY_HASH error code, 5365
ER_CONN_CONTROL_INVALID_CONN_DELAY_TYPE error code, 5364
ER_CONN_CONTROL_STAT_CONN_DELAY_TRIGGERED_RESET_FAILED error code, 5364
ER_CONN_CONTROL_STAT_CONN_DELAY_TRIGGERED_UPDATE_FAILED error code, 5364
ER_CONN_INIT_CONNECT_IGNORED error code, 5363
ER_CONN_PER_THREAD_NO_THREAD error code, 5246
ER_CONN_PIP_CANT_CREATE_EVENT error code, 5246
ER_CONN_PIP_CANT_CREATE_PIPE error code, 5246
ER_CONN_SHM_CANT_CREATE_CONNECTION error code, 5246
ER_CONN_SHM_CANT_CREATE_SERVICE error code, 5245
ER_CONN_SHM_LISTENER error code, 5245
ER_CONN_SOCKET_ACCEPT_FAILED error code, 5250
ER_CONN_SOCKET_SELECT_FAILED error code, 5249
ER_CONN_TCP_ADDRESS error code, 5246
ER_CONN_TCP_BIND_FAIL error code, 5247
ER_CONN_TCP_BIND_RETRY error code, 5247
ER_CONN_TCP_CANT_RESET_V6ONLY error code, 5247
ER_CONN_TCP_CANT_RESOLVE_HOSTNAME error code, 5247
ER_CONN_TCP_CREATED error code, 5246
ER_CONN_TCP_ERROR_WITH_STRERROR error code, 5246
ER_CONN_TCP_IPV6_AVAILABLE error code, 5246
ER_CONN_TCP_IPV6_UNAVAILABLE error code, 5246
ER_CONN_TCP_IP_NOT_LOGGED error code, 5247
ER_CONN_TCP_IS_THERE_ANOTHER_USING_PORT error code, 5247
ER_CONN_TCP_LISTEN_FAIL error code, 5248
ER_CONN_TCP_NO_SOCKET error code, 5246
ER_CONN_TCP_RESOLVE_INFO error code, 5248
ER_CONN_TCP_START_FAIL error code, 5248
ER_CONN_TPC_BIND_FAIL error code, 5247
ER_CONN_UNIX_IS_THERE_ANOTHER_USING_SOCKET error code, 5247
ER_CONN_UNIX_LISTEN_FAILED error code, 5248
ER_CONN_UNIX_LOCK_FILE_CANT_CLOSE error code, 5249
ER_CONN_UNIX_LOCK_FILE_CANT_CREATE error code, 5248
ER_CONN_UNIX_LOCK_FILE_CANT_DELETE error code, 5249
ER_CONN_UNIX_LOCK_FILE_CANT_OPEN error code, 5249
ER_CONN_UNIX_LOCK_FILE_CANT_READ error code, 5249
ER_CONN_UNIX_LOCK_FILE_CANT_SYNC error code, 5249
ER_CONN_UNIX_LOCK_FILE_CANT_WRITE error code, 5249
ER_CONN_UNIX_LOCK_FILE_EMPTY error code, 5249
ER_CONN_UNIX_LOCK_FILE_FAIL error code, 5248
ER_CONN_UNIX_LOCK_FILE_GIVING_UP error code, 5248
ER_CONN_UNIX_LOCK_FILE_PIDLESS error code, 5249
ER_CONN_UNIX_NO_BIND_NO_START error code, 5248
ER_CONN_UNIX_NO_FD error code, 5248
ER_CONN_UNIX_PATH_TOO_LONG error code, 5248

ER_CONN_UNIX_PID_CLAIMED_SOCKET_FILE error code, 5247
ER_CONSECUTIVE_REORG_PARTITIONS error code, 5132
ER_CON_COUNT_ERROR error code, 5101
ER_CORE_VALUES error code, 5231
ER_CORRUPTED_JSON_DIFF error code, 5187
ER_CORRUPT_HELP_DB error code, 5115
ER_COULD_NOT_APPLY_JSON_DIFF error code, 5187
ER_COULD_NOT_CREATE_WINDOWS_REGISTRY_KEY error code, 5573
ER_COULD_NOT_REINITIALIZE_AUDIT_LOG_FILTERS error code, 5566
ER_CRASHED_ON_REPAIR error code, 5111
ER_CRASHED_ON_USAGE error code, 5111
ER_CREATED_SYSTEM_WITH_VERSION error code, 5335
ER_CREATE_FILEGROUP_FAILED error code, 5133
ER_CREATING_NEW_UUID error code, 5228
ER_CREATING_NEW_UUID_FIRST_START error code, 5326
ER_CREDENTIALLESS_AUTO_USER_BAD error code, 5229
ER_CREDENTIALS_CONTRADICT_TO_HISTORY error code, 5186
ER_CTE_MAX_RECURSION_DEPTH error code, 5186
ER_CTE_RECURSIVE_FORBIDDEN_JOIN_ORDER error code, 5179
ER_CTE_RECURSIVE_FORBIDS_AGGREGATION error code, 5179
ER_CTE_RECURSIVE_REQUIRES_NONRECURSIVE_FIRST error code, 5179
ER_CTE_RECURSIVE_REQUIRES_SINGLE_REFERENCE error code, 5179
ER_CTE_RECURSIVE_REQUIRES_UNION error code, 5179
ER_CURRENT_PASSWORD_CANNOT_BE_RETAINED error code, 5215
ER_CURRENT_PASSWORD_NOT_REQUIRED error code, 5215
ER_CUT_VALUE_GROUP_CONCAT error code, 5116
ER_CYCLE_TIMER_IS_NOT_AVAILABLE error code, 5337
ER_DATABASE_NAME error code, 5138
ER_DATA_DIRECTORY_UNUSABLE error code, 5573
ER_DATA_DIRECTORY_UNUSABLE_DELETABLE error code, 5597
ER_DATA_OUT_OF_RANGE error code, 5143
ER_DATA_TOO_LONG error code, 5125
ER_DATETIME_FUNCTION_OVERFLOW error code, 5127
ER_DA_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES error code, 5213
ER_DA_GRP_RPL_STARTED_AUTO_REJOIN error code, 5217
ER_DA_INVALID_CONDITION_NUMBER error code, 5148
ER_DA_PLUGIN_INSTALL_ERROR error code, 5214
ER_DA_RPL_GTID_TABLE_CANNOT_OPEN error code, 5213
ER_DA_SSL_LIBRARY_ERROR error code, 5214
ER_DA_UDF_INVALID_CHARSET error code, 5219
ER_DA_UDF_INVALID_CHARSET_SPECIFIED error code, 5219
ER_DA_UDF_INVALID_COLLATION error code, 5220
ER_DA_UDF_INVALID_EXTENSION_ARGUMENT_TYPE error code, 5220
ER_DA_UNKNOWN_ERROR_NUMBER error code, 5214
ER_DBACCESS_DENIED_ERROR error code, 5101
ER_DBUG_CHECK_SHARES_DROPPED error code, 5277
ER_DBUG_CHECK_SHARES_INFO error code, 5277
ER_DBUG_CHECK_SHARES_OPEN error code, 5277
ER_DB_ACCESS_DENIED error code, 5213
ER_DB_CREATE_EXISTS error code, 5099
ER_DB_DROP_EXISTS error code, 5099
ER_DB_DROP_RMDIR error code, 5099
ER_DB_DROP_RMDIR2 error code, 5183
ER_DB_OPT_NOT_FOUND_USING_DEFAULT_CHARSET error code, 5306

ER_DDL_IN_PROGRESS error code, 5185
ER_DDL_LOG_ERROR error code, 5135
ER_DD_ABORTING_PARTIAL_UPGRADE error code, 5255
ER_DD_CACHE_NOT_EMPTY_AT_SHUTDOWN error code, 5222
ER_DD_CANT_CREATE_OBJECT_KEY error code, 5223
ER_DD_CANT_CREATE_SP error code, 5306
ER_DD_CANT_FETCH_TABLE_DATA error code, 5306
ER_DD_CANT_FIX_SE_DATA error code, 5306
ER_DD_CANT_GET_OBJECT_KEY error code, 5223
ER_DD_CANT_RESOLVE_VIEW error code, 5241
ER_DD_CREATED_FOR_UPGRADE error code, 5256
ER_DD_ERROR_CREATING_ENTRY error code, 5306
ER_DD_FAILSAFE error code, 5220
ER_DD_FRM_EXISTS_FOR_TABLE error code, 5255
ER_DD_INITIALIZE error code, 5343
ER_DD_INITIALIZE_SQL_ERROR error code, 5567
ER_DD_INIT_FAILED error code, 5222
ER_DD_INIT_UPGRADE_FAILED error code, 5221
ER_DD_METADATA_NOT_FOUND error code, 5222
ER_DD_MINOR_DOWNGRADE error code, 5344
ER_DD_MINOR_DOWNGRADE_VERSION_NOT_SUPPORTED error code, 5344
ER_DD_NO_VERSION_FOUND error code, 5344
ER_DD_NO_WRITES_NO_REPOPULATION error code, 5220
ER_DD_OBJECT_RELEASER_REMAINS error code, 5223
ER_DD_OBJECT_REMAINS error code, 5222
ER_DD_OBJECT_REMAINS_IN_RELEASER error code, 5223
ER_DD_POPULATING_TABLES_FAILED error code, 5222
ER_DD_RESTART error code, 5343
ER_DD_SCHEMA_NOT_FOUND error code, 5255
ER_DD_SE_INIT_FAILED error code, 5255
ER_DD_TABLESPACE_NOT_FOUND error code, 5240
ER_DD_TABLE_NOT_FOUND error code, 5255
ER_DD_TRG_CANT_ADD error code, 5241
ER_DD_TRG_CONNECTION_COLLATION_MISSING error code, 5240
ER_DD_TRG_DB_COLLATION_MISSING error code, 5240
ER_DD_TRG_DEFINER_OOM error code, 5240
ER_DD_TRG_FILE_UNREADABLE error code, 5240
ER_DD_UPDATING_PLUGIN_MD_FAILED error code, 5222
ER_DD_UPGRADE error code, 5343
ER_DD_UPGRADE_COMPLETED error code, 5593
ER_DD_UPGRADE_DD_OPEN_FAILED error code, 5332
ER_DD_UPGRADE_DD_POPULATED error code, 5333
ER_DD_UPGRADE_FAILED_FIND_VALID_DATA_DIR error code, 5334
ER_DD_UPGRADE_FAILED_INIT_DD_SE error code, 5334
ER_DD_UPGRADE_FAILED_TO_ACQUIRE_TABLESPACE error code, 5333
ER_DD_UPGRADE_FAILED_TO_CREATE_INDEX_STATS error code, 5334
ER_DD_UPGRADE_FAILED_TO_CREATE_TABLE_STATS error code, 5334
ER_DD_UPGRADE_FAILED_TO_FETCH_TABLES error code, 5333
ER_DD_UPGRADE_FAILED_TO_FETCH_TABLESPACES error code, 5333
ER_DD_UPGRADE_FAILED_TO_RESOLVE_TABLESPACE_ENGINE error code, 5333
ER_DD_UPGRADE_FAILED_TO_UPDATE_VER_NO_IN_TABLESPACE error code, 5343
ER_DD_UPGRADE_FOUND_PARTIALLY_UPGRADED_DD_ABORT error code, 5334
ER_DD_UPGRADE_FOUND_PARTIALLY_UPGRADED_DD_CONTINUE error code, 5335
ER_DD_UPGRADE_INDEX_STATS_MIGRATE_COMPLETED error code, 5334

ER_DD_UPGRADE_INFO_FILE_CLOSE_FAILED error code, 5333
ER_DD_UPGRADE_INFO_FILE_OPEN_FAILED error code, 5333
ER_DD_UPGRADE_OFF error code, 5344
ER_DD_UPGRADE_RENAME_IDX_STATS_FILE_FAILED error code, 5332
ER_DD_UPGRADE_SCHEMA_UNAVAILABLE error code, 5344
ER_DD_UPGRADE_SDI_INFO_UPDATE_FAILED error code, 5335
ER_DD_UPGRADE_SE_LOGS_FAILED error code, 5335
ER_DD_UPGRADE_START error code, 5334
ER_DD_UPGRADE_TABLESPACE_MIGRATION_FAILED error code, 5334
ER_DD_UPGRADE_TABLE_INTACT_ERROR error code, 5340
ER_DD_UPGRADE_TABLE_STATS_MIGRATE_COMPLETED error code, 5334
ER_DD_UPGRADE_VERSION_NOT_SUPPORTED error code, 5344
ER_DD_UPGRADE_VIEW_COLUMN_NAME_TOO_LONG error code, 5342
ER_DD_VERSION_FOUND error code, 5220
ER_DD_VERSION_INSTALLED error code, 5220
ER_DD_VERSION_UNSUPPORTED error code, 5221
ER_DD_VIEW_CANT_ALLOC_CHARSET error code, 5222
ER_DD_VIEW_CANT_CREATE error code, 5222
ER_DD_VIEW_WITHOUT_DEFINER error code, 5241
ER_DEBUG_SYNC_EXECUTED error code, 5224
ER_DEBUG_SYNC_HIT error code, 5224
ER_DEBUG_SYNC_HIT_LIMIT error code, 5139
ER_DEBUG_SYNC_OOM error code, 5224
ER_DEBUG_SYNC_THREAD_MAX error code, 5224
ER_DEBUG_SYNC_TIMEOUT error code, 5139
ER_DEFAULT_AS_VAL_GENERATED error code, 5201
ER_DEFAULT_SE_UNAVAILABLE error code, 5239
ER_DEFAULT_VAL_GENERATED_FUNCTION_IS_NOT_ALLOWED error code, 5201
ER_DEFAULT_VAL_GENERATED_NAMED_FUNCTION_IS_NOT_ALLOWED error code, 5201
ER_DEFAULT_VAL_GENERATED_NON_PRIOR error code, 5201
ER_DEFAULT_VAL_GENERATED_REF_AUTO_INC error code, 5201
ER_DEFAULT_VAL_GENERATED_ROW_VALUE error code, 5201
ER_DEFAULT_VAL_GENERATED_VARIABLES error code, 5201
ER_DEPART_WITH_GRACE error code, 5227
ER_DEPENDENT_BY_DEFAULT_GENERATED_VALUE error code, 5200
ER_DEPENDENT_BY_FUNCTIONAL_INDEX error code, 5209
ER_DEPENDENT_BY_GENERATED_COLUMN error code, 5166
ER_DEPENDENT_BY_PARTITION_FUNC error code, 5211
ER_DEPRECATED_NATIONAL error code, 5195
ER_DEPRECATED_SYNTAX_NO_REPLACEMENT error code, 5341
ER_DEPRECATED_SYNTAX_WITH_REPLACEMENT error code, 5341
ER_DEPRECATED_TIMESTAMP_IMPLICIT_DEFAULTS error code, 5229
ER_DEPRECATED_UTF8_ALIAS error code, 5195
ER_DEPRECATE_MSG_NO_REPLACEMENT error code, 5341
ER_DEPRECATE_MSG_WITH_REPLACEMENT error code, 5324
ER_DERIVED_MUST_HAVE_ALIAS error code, 5115
ER_DES_FILE_WRONG_KEY error code, 5242
ER_DETACHED_SESSIONS_LEFT_BY_PLUGIN error code, 5271
ER_DETACHING_SESSION_LEFT_BY_PLUGIN error code, 5271
ER_DIMENSION_UNSUPPORTED error code, 5163
ER_DISABLED_STORAGE_ENGINE error code, 5170
ER_DISABLED_STORAGE_ENGINE_AS_DEFAULT error code, 5223
ER_DISALLOWED_OPERATION error code, 5188
ER_DISCARD_FK_CHECKS_RUNNING error code, 5151

ER_DISCONNECTING_REMAINING_CLIENTS error code, 5232
ER_DISK_FULL_NOWAIT error code, 5190
ER_DIVISION_BY_ZERO error code, 5123
ER_DONT_SUPPORT_SLAVE_PRESERVE_COMMIT_ORDER error code, 5160
ER_DROP_DATABASE_FAILED_RMDIR_MANUALLY error code, 5342
ER_DROP_FILEGROUP_FAILED error code, 5133
ER_DROP_INDEX_FK error code, 5134
ER_DROP_LAST_PARTITION error code, 5131
ER_DROP_PARTITION_NON_EXISTENT error code, 5131
ER_DUPLICATED_VALUE_IN_TYPE error code, 5118
ER_DUPLICATE_OPTION_KEY error code, 5178
ER_DUPLICATE_SYS_VAR error code, 5568
ER_DUPLICATE_TABLE_LOCK error code, 5178
ER_DUP_ARGUMENT error code, 5114
ER_DUP_ENTRY error code, 5103
ER_DUP_ENTRY_AUTOINCREMENT_CASE error code, 5135
ER_DUP_ENTRY_WITH_KEY_NAME error code, 5136
ER_DUP_FD_OPEN_FAILED error code, 5326
ER_DUP_FIELDNAME error code, 5102
ER_DUP_INDEX error code, 5153
ER_DUP_KEY error code, 5100
ER_DUP_KEYNAME error code, 5102
ER_DUP_LIST_ENTRY error code, 5159
ER_DUP_SIGNAL_SET error code, 5139
ER_DUP_UNIQUE error code, 5110
ER_DUP_UNKNOWN_IN_INDEX error code, 5155
ER_EMPTY_PIPELINE_FOR_ERROR_LOG_SERVICE error code, 5193
ER_EMPTY_QUERY error code, 5103
ER_ENDING_INIT error code, 5567
ER_END_INITFILE error code, 5234
ER_ENFORCE_GTID_CONSISTENCY_WARN_WITH_ONGOING_GTID_VIOLATING_TX error code, 5166
ER_ENGINE_CANT_DROP_MISSING_TABLE error code, 5183
ER_ENGINE_CANT_DROP_TABLE error code, 5182
ER_ENGINE_COST_FAILED_TO_READ error code, 5268
ER_ENGINE_COST_INVALID_CONST_CONSTANT_FOR_SE_AND_DEVICE error code, 5268
ER_ENGINE_COST_INVALID_DEVICE_TYPE_FOR_SE error code, 5267
ER_ENGINE_COST_UNKNOWN_COST_CONSTANT error code, 5267
ER_ENGINE_COST_UNKNOWN_STORAGE_ENGINE error code, 5267
ER_ENGINE_OUT_OF_MEMORY error code, 5159
ER_ERRMSG_CANT_FIND_FILE error code, 5256
ER_ERRMSG_CANT_READ error code, 5256
ER_ERRMSG_LOADING_55_STYLE error code, 5256
ER_ERRMSG_MISSING_IN_FILE error code, 5256
ER_ERRMSG_OOM error code, 5256
ER_ERRMSG_REPLACEMENTS_FAILED error code, 5320
ER_ERRMSG_REPLACEMENT_DODGY error code, 5320
ER_ERROR_DURING_COMMIT error code, 5111
ER_ERROR_DURING_FLUSH_LOGS error code, 5111
ER_ERROR_DURING_FLUSH_LOG_COMMIT_PHASE error code, 5342
ER_ERROR_DURING_OPTIMIZE_TABLE error code, 5304
ER_ERROR_DURING_ROLLBACK error code, 5111
ER_ERROR_ENABLING_KEYS error code, 5305
ER_ERROR_INFO_FROM_DA error code, 5563
ER_ERROR_IN_TRIGGER_BODY error code, 5144

ER_ERROR_IN_UNKNOWN_TRIGGER_BODY error code, 5144
ER_ERROR_ON_MASTER error code, 5158
ER_ERROR_ON_MODIFYING_GTID_EXECUTED_TABLE error code, 5171
ER_ERROR_ON_READ error code, 5100
ER_ERROR_ON_RENAME error code, 5100
ER_ERROR_ON_WRITE error code, 5100
ER_ERROR_WHEN_EXECUTING_COMMAND error code, 5113
ER_EVENT_ALREADY_EXISTS error code, 5133
ER_EVENT_CANNOT_ALTER_IN_THE_PAST error code, 5136
ER_EVENT_CANNOT_CREATE_IN_THE_PAST error code, 5136
ER_EVENT_CANT_FIND_TIMEZONE error code, 5307
ER_EVENT_CANT_GET_CHARSET error code, 5307
ER_EVENT_CANT_GET_COLLATION error code, 5307
ER_EVENT_CANT_GET_LOCK_FOR_DROPPING_EVENT error code, 5311
ER_EVENT_CANT_GET_TIMEZONE_FROM_FIELD error code, 5306
ER_EVENT_CANT_INIT_QUEUE error code, 5224
ER_EVENT_CANT_OPEN_TABLE_MYSQL_EVENT error code, 5307
ER_EVENT_DOES_NOT_EXIST error code, 5133
ER_EVENT_DROPPING error code, 5269
ER_EVENT_ENDS_BEFORE_STARTS error code, 5133
ER_EVENT_ERROR_CREATING_QUERY_TO_WRITE_TO_BINLOG error code, 5310
ER_EVENT_ERROR_DURING_COMPILATION error code, 5269
ER_EVENT_EXECUTION_FAILED error code, 5224
ER_EVENT_EXECUTION_FAILED_CANT_AUTHENTICATE_USER error code, 5269
ER_EVENT_EXECUTION_FAILED_USER_LOST_EVEN_PRIVILEGE error code, 5269
ER_EVENT_EXEC_TIME_IN_THE_PAST error code, 5134
ER_EVENT_INTERVAL_NOT_POSITIVE_OR_TOO_BIG error code, 5133
ER_EVENT_INVALID_CREATION_CTX error code, 5137
ER_EVENT_LAST_EXECUTION error code, 5224
ER_EVENT_MESSAGE_STACK error code, 5224
ER_EVENT_PURGING_QUEUE error code, 5224
ER_EVENT_RECURSION_FORBIDDEN error code, 5135
ER_EVENT_SAME_NAME error code, 5134
ER_EVENT_SCHEDULER_ERROR_GETTING_EVENT_OBJECT error code, 5310
ER_EVENT_SCHEDULER_ERROR_LOADING_FROM_DB error code, 5310
ER_EVENT_SCHEDULER_GOT_BAD_DATA_FROM_TABLE error code, 5310
ER_EVENT_SET_VAR_ERROR error code, 5135
ER_EVENT_UNABLE_TO_DROP_EVENT error code, 5311
ER_EXCEEDED_MV_KEYS_NUM error code, 5216
ER_EXCEEDED_MV_KEYS_SPACE error code, 5216
ER_EXCEPTIONS_WRITE_ERROR error code, 5138
ER_EXCESS_ARGUMENTS error code, 5235
ER_EXPIRE_LOGS_DAYS_IGNORED error code, 5342
ER_EXPLAIN_NOT_SUPPORTED error code, 5158
ER_FAILED_DEFAULT_ROLES error code, 5175
ER_FAILED_READ_FROM_PAR_FILE error code, 5143
ER_FAILED_REVOKE_ROLE error code, 5175
ER_FAILED_ROLE_GRANT error code, 5175
ER_FAILED_START_MYSQLD_DAEMON error code, 5327
ER_FAILED_TO_ACQUIRE_LOCK_ON_RESOURCE_GROUP error code, 5331
ER_FAILED_TO_ADD_RESOURCE_GROUP_TO_MAP error code, 5330
ER_FAILED_TO_ADD_RPL_FILTER error code, 5336
ER_FAILED_TO_ALLOCATE_MEMORY_FOR_RESOURCE_GROUP error code, 5330
ER_FAILED_TO_ALLOCATE_MEMORY_FOR_RESOURCE_GROUP_HASH error code, 5330

ER_FAILED_TO_ALLOCATE_SSL_BIO error code, 5568
ER_FAILED_TO_APPLY_RESOURCE_GROUP_CONTROLLER error code, 5330
ER_FAILED_TO_BINLOG_DROP_EVENT error code, 5336
ER_FAILED_TO_BUILD_DO_AND_IGNORE_TABLE_HASHES error code, 5337
ER_FAILED_TO_COMPRESS_GTID_EXECUTED_TABLE error code, 5261
ER_FAILED_TO_COMPRESS_GTID_EXECUTED_TABLE_OOM error code, 5261
ER_FAILED_TO_CONSTRUCT_DROP_EVENT_QUERY error code, 5335
ER_FAILED_TO_CREATE_CLONE_HANDLER error code, 5337
ER_FAILED_TO_CREATE_GTID_TABLE_COMPRESSION_THREAD error code, 5261
ER_FAILED_TO_CREATE_SDI_FOR_TABLESPACE error code, 5333
ER_FAILED_TO_DECREMENT_NUMBER_OF_THREADS error code, 5271
ER_FAILED_TO_DELETE_FROM_GTID_EXECUTED_TABLE error code, 5261
ER_FAILED_TO_DESERIALIZE_RESOURCE_GROUP error code, 5330
ER_FAILED_TO_FIND_COLLATION_NAME error code, 5325
ER_FAILED_TO_FIND_DL_ENTRY error code, 5322
ER_FAILED_TO_FIND_LOCALE_NAME error code, 5325
ER_FAILED_TO_FIND_MYSQLD_STATUS error code, 5341
ER_FAILED_TO_GENERATE_UNIQUE_LOGFILE error code, 5322
ER_FAILED_TO_GET_ABSOLUTE_PATH error code, 5326
ER_FAILED_TO_HANDLE_DEFAULTS_FILE error code, 5568
ER_FAILED_TO_INIT_SYS_VAR error code, 5568
ER_FAILED_TO_INIT_THREAD_ATTR_FOR_GTID_TABLE_COMPRESSION error code, 5261
ER_FAILED_TO_JOIN_GTID_TABLE_COMPRESSION_THREAD error code, 5261
ER_FAILED_TO_LOCK_MEM error code, 5234
ER_FAILED_TO_OPEN_COST_CONSTANT_TABLES error code, 5268
ER_FAILED_TO_OPEN_SHARED_LIBRARY error code, 5322
ER_FAILED_TO_PERSIST_RESOURCE_GROUP_METADATA error code, 5329
ER_FAILED_TO_READ_FILE error code, 5322
ER_FAILED_TO_REMOVE_TEMP_TABLE error code, 5301
ER_FAILED_TO_REPAIR_TABLE error code, 5301
ER_FAILED_TO_SET_PERSISTED_OPTIONS error code, 5341
ER_FAILED_TO_START_SLAVE_THREAD error code, 5336
ER_FAILED_TO_STORE_SDI_FOR_TABLESPACE error code, 5333
ER_FAILED_TO_UPDATE_RESOURCE_GROUP error code, 5330
ER_FAILED_TO_WRITE_TO_FILE error code, 5322
ER_FAIL_CHROOT error code, 5233
ER_FAIL_SETGID error code, 5233
ER_FAIL_SETREGID error code, 5233
ER_FAIL_SETREUID error code, 5233
ER_FAIL_SETUID error code, 5233
ER_FEATURE_DISABLED error code, 5118
ER_FEATURE_DISABLED_SEE_DOC error code, 5170
ER_FEATURE_NOT_AVAILABLE error code, 5166
ER_FEATURE_UNSUPPORTED error code, 5188
ER_FIELD_IN_GROUPING_NOT_GROUP_BY error code, 5182
ER_FIELD_IN_ORDER_NOT_SELECT error code, 5162
ER_FIELD_NOT_FOUND_PART_ERROR error code, 5130
ER_FIELD_SPECIFIED_TWICE error code, 5106
ER_FIELD_TYPE_NOT_ALLOWED_AS_PARTITION_FIELD error code, 5140
ER_FILEGROUP_OPTION_ONLY_ONCE error code, 5133
ER_FILESORT_TERMINATED error code, 5325
ER_FILE_CORRUPT error code, 5158
ER_FILE_EXISTS_DURING_UPGRADE error code, 5308
ER_FILE_EXISTS_ERROR error code, 5104

ER_FILE_HAS_OLD_FORMAT error code, 5324
ER_FILE_NOT_FOUND error code, 5099
ER_FILE_PRIVILEGE_FOR_REPLICATION_CHECKS error code, 5602
ER_FILE_TYPE_UNKNOWN error code, 5257
ER_FILE_USED error code, 5100
ER_FILSORT_ABORT error code, 5100
ER_FIREWALL_ACCESS_DENIED error code, 5355
ER_FIREWALL_FAILED_TO_READ_FIREWALL_TABLES error code, 5354
ER_FIREWALL_FAILED_TO_REG_DYNAMIC_PRIVILEGES error code, 5354
ER_FIREWALL_RECORDING_STMT_WAS_TRUNCATED error code, 5355
ER_FIREWALL_RECORDING_STMT_WITHOUT_TEXT error code, 5355
ER_FIREWALL_RELOADING_CACHE error code, 5355
ER_FIREWALL_RESET_FOR_USER error code, 5355
ER_FIREWALL_SKIPPED_UNKNOWN_USER_MODE error code, 5355
ER_FIREWALL_STATUS_FLUSHED error code, 5355
ER_FIREWALL_SUSPICIOUS_STMT error code, 5355
ER_FIXING_CLIENT_CHARSET error code, 5234
ER_FK_CANNOT_CHANGE_ENGINE error code, 5202
ER_FK_CANNOT_DROP_PARENT error code, 5196
ER_FK_CANNOT_OPEN_PARENT error code, 5152
ER_FK_CANNOT_USE_VIRTUAL_COLUMN error code, 5197
ER_FK_COLUMN_CANNOT_CHANGE error code, 5153
ER_FK_COLUMN_CANNOT_CHANGE_CHILD error code, 5153
ER_FK_COLUMN_CANNOT_DROP error code, 5153
ER_FK_COLUMN_CANNOT_DROP_CHILD error code, 5153
ER_FK_COLUMN_NOT_NULL error code, 5153
ER_FK_DEPTH_EXCEEDED error code, 5158
ER_FK_DUP_NAME error code, 5153
ER_FK_FAIL_ADD_SYSTEM error code, 5152
ER_FK_INCOMPATIBLE_COLUMNS error code, 5202
ER_FK_INCORRECT_OPTION error code, 5152
ER_FK_NO_COLUMN_PARENT error code, 5197
ER_FK_NO_INDEX_CHILD error code, 5152
ER_FK_NO_INDEX_PARENT error code, 5152
ER_FORBID_SCHEMA_CHANGE error code, 5128
ER_FORCE_CLOSE_THREAD error code, 5323
ER_FORCING_CLOSE error code, 5104
ER_FOREIGN_DATA_SOURCE_DOESNT_EXIST error code, 5126
ER_FOREIGN_DATA_STRING_INVALID error code, 5127
ER_FOREIGN_DATA_STRING_INVALID_CANT_CREATE error code, 5127
ER_FOREIGN_DUPLICATE_KEY_OLD_UNUSED error code, 5134
ER_FOREIGN_DUPLICATE_KEY_WITHOUT_CHILD_INFO error code, 5148
ER_FOREIGN_DUPLICATE_KEY_WITH_CHILD_INFO error code, 5148
ER_FOREIGN_KEY_ON_PARTITIONED error code, 5131
ER_FOREIGN_SERVER_DOESNT_EXIST error code, 5130
ER_FOREIGN_SERVER_EXISTS error code, 5130
ER_FOUND_MISSING_GTIDS error code, 5428
ER_FOUND_ROWS_WHILE_REPAIRING error code, 5304
ER_FPARSER_BAD_HEADER error code, 5121
ER_FPARSER_EOF_IN_COMMENT error code, 5121
ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER error code, 5121
ER_FPARSER_ERROR_IN_PARAMETER error code, 5121
ER_FPARSER_TOO_BIG_FILE error code, 5121
ER_FSEEK_FAIL error code, 5123

ER_FT_BOOL_SYNTAX_INVALID error code, 5229
ER_FT_MATCHING_KEY_NOT_FOUND error code, 5111
ER_FULLTEXT_FUNCTIONAL_INDEX error code, 5200
ER_FULLTEXT_NOT_SUPPORTED_WITH_PARTITIONING error code, 5147
ER_FUNCTIONAL_INDEX_DATA_IS_TOO_LONG error code, 5216
ER_FUNCTIONAL_INDEX_FUNCTION_IS_NOT_ALLOWED error code, 5200
ER_FUNCTIONAL_INDEX_ON_FIELD error code, 5200
ER_FUNCTIONAL_INDEX_ON_JSON_OR_GEOMETRY_FUNCTION error code, 5199
ER_FUNCTIONAL_INDEX_ON_LOB error code, 5199
ER_FUNCTIONAL_INDEX_PRIMARY_KEY error code, 5199
ER_FUNCTIONAL_INDEX_REF_AUTO_INCREMENT error code, 5199
ER_FUNCTIONAL_INDEX_ROW_VALUE_IS_NOT_ALLOWED error code, 5204
ER_FUNCTION_NOT_DEFINED error code, 5107
ER_FUNC_INEXISTENT_NAME_COLLISION error code, 5138
ER_FUTURE_DATE error code, 5232
ER_GENERATED_COLUMN_FUNCTION_IS_NOT_ALLOWED error code, 5165
ER_GENERATED_COLUMN_NAMED_FUNCTION_IS_NOT_ALLOWED error code, 5200
ER_GENERATED_COLUMN_NON_PRIOR error code, 5166
ER_GENERATED_COLUMN_REF_AUTO_INC error code, 5166
ER_GENERATED_COLUMN_ROW_VALUE error code, 5200
ER_GENERATED_COLUMN_VARIABLES error code, 5200
ER_GEOMETRY_IN_UNKNOWN_LENGTH_UNIT error code, 5214
ER_GEOMETRY_PARAM_LATITUDE_OUT_OF_RANGE error code, 5196
ER_GEOMETRY_PARAM_LONGITUDE_OUT_OF_RANGE error code, 5196
ER_GET_ERRMSG error code, 5118
ER_GET_ERRNO error code, 5100
ER_GET_ERRNO_FROM_STORAGE_ENGINE error code, 5324
ER_GET_STACKED_DA_WITHOUT_ACTIVE_HANDLER error code, 5158
ER_GET_TEMPORARY_ERRMSG error code, 5118
ER_GIS_DATA_WRONG_ENDIANESS error code, 5161
ER_GIS_DIFFERENT_SRIDS error code, 5160
ER_GIS_INVALID_DATA error code, 5160
ER_GIS_MAX_POINTS_IN_GEOMETRY_OVERFLOWED error code, 5168
ER_GIS_UNKNOWN_ERROR error code, 5160
ER_GIS_UNKNOWN_EXCEPTION error code, 5160
ER_GIS_UNSUPPORTED_ARGUMENT error code, 5160
ER_GLOBAL_VARIABLE error code, 5114
ER_GLOBSTAT_CHANGE_DURING_QUERY error code, 5217
ER_GNO_EXHAUSTED error code, 5149
ER_GRANT_WRONG_HOST_OR_USER error code, 5109
ER_GROUPING_ON_TIMESTAMP_IN_DST error code, 5217
ER_GROUP_REPLICATION_APPLIER_INIT_ERROR error code, 5164
ER_GROUP_REPLICATION_COMMAND_FAILURE error code, 5189
ER_GROUP_REPLICATION_COMMUNICATION_LAYER_JOIN_ERROR error code, 5165
ER_GROUP_REPLICATION_COMMUNICATION_LAYER_SESSION_ERROR error code, 5165
ER_GROUP_REPLICATION_CONFIGURATION error code, 5164
ER_GROUP_REPLICATION_MAX_GROUP_SIZE error code, 5172
ER_GROUP_REPLICATION_PLUGIN_NOT_INSTALLED error code, 5261
ER_GROUP_REPLICATION_RUNNING error code, 5164
ER_GROUP_REPLICATION_STOP_APPLIER_THREAD_TIMEOUT error code, 5164
ER_GRP_DELAYED_VCLE_LOGGING error code, 5592
ER_GRP_RPL_ABORTS_AS_SSL_NOT_SUPPORTED_BY_MYSQLD error code, 5411
ER_GRP_RPL_ADD_GRPSID_TO_GRPGTIDSID_MAP_ERROR error code, 5385
ER_GRP_RPL_ADD_GTID_INFO_WITHOUT_LOCAL_GTID_FAILED error code, 5397

ER_GRP_RPL_ADD_GTID_INFO_WITHOUT_REMOTE_GTID_FAILED error code, 5397
ER_GRP_RPL_ADD_GTID_INFO_WITH_LOCAL_GTID_FAILED error code, 5397
ER_GRP_RPL_ADD_GTID_INFO_WITH_REMOTE_GTID_FAILED error code, 5397
ER_GRP_RPL_ADD_GTID_TO_GRPGTID_EXECUTED_ERROR error code, 5386
ER_GRP_RPL_ADD_RETRIEVED_SET_TO_GRP_GTID_EXECUTED_ERROR error code, 5386
ER_GRP_RPL_ALL_DONORS_LEFT_ABORT_RECOVERY error code, 5399
ER_GRP_RPL_ALL_OBSERVERS_UNREGISTERED error code, 5410
ER_GRP_RPL_APPENDING_DATA_TO_INTERNAL_CACHE_FAILED error code, 5404
ER_GRP_RPL_APPLIER_EXECUTION_FATAL_ERROR error code, 5385
ER_GRP_RPL_APPLIER_HANDLER_IS_IN_USE error code, 5415
ER_GRP_RPL_APPLIER_HANDLER_NOT_INITIALIZED error code, 5414
ER_GRP_RPL_APPLIER_HANDLER_ROLE_IS_IN_USE error code, 5415
ER_GRP_RPL_APPLIER_INITIALIZED error code, 5411
ER_GRP_RPL_APPLIER_NOT_STARTED_DUE_TO_RUNNING_PREV_SHUTDOWN error code, 5411
ER_GRP_RPL_APPLIER_PIPELINE_NOT_DISPOSED error code, 5384
ER_GRP_RPL_APPLIER_TERMINATION_TIMED_OUT_ON_SHUTDOWN error code, 5409
ER_GRP_RPL_APPLIER_THD_EXECUTION_ABORTED error code, 5385
ER_GRP_RPL_APPLIER_THD_KILLED error code, 5384
ER_GRP_RPL_APPLIER_THD_SETUP_ERROR error code, 5394
ER_GRP_RPL_APPLIER_THD_START_ERROR error code, 5394
ER_GRP_RPL_APPLIER_THD_STOP_ERROR error code, 5395
ER_GRP_RPL_APPOINTED_PRIMARY_NOT_PRESENT error code, 5571
ER_GRP_RPL_AUTO_INC_OFFSET_RESET error code, 5396
ER_GRP_RPL_AUTO_INC_OFFSET_SET error code, 5396
ER_GRP_RPL_AUTO_INC_RESET error code, 5396
ER_GRP_RPL_AUTO_INC_SET error code, 5396
ER_GRP_RPL_BINLOG_CHECKSUM_SET error code, 5402
ER_GRP_RPL_BINLOG_DISABLED error code, 5412
ER_GRP_RPL_BLOCK_SIZE_DIFF_FROM_GRP error code, 5394
ER_GRP_RPL_BROADCASTING_TRANS_TO_GRP_FAILED error code, 5404
ER_GRP_RPL_BROADCAST_COMMIT_MSSG_TOO_BIG error code, 5418
ER_GRP_RPL_BROADCAST_COMMIT_TRANS_MSSG_FAILED error code, 5385
ER_GRP_RPL_CANNOT_EXECUTE_TRANS_IN_ERROR_STATE error code, 5402
ER_GRP_RPL_CANNOT_EXECUTE_TRANS_IN_OFFLINE_MODE error code, 5402
ER_GRP_RPL_CANNOT_EXECUTE_TRANS_WHILE_RECOVERING error code, 5402
ER_GRP_RPL_CANNOT_EXECUTE_TRANS_WHILE_STOPPING error code, 5402
ER_GRP_RPL_CANT_GENERATE_GTID error code, 5387
ER_GRP_RPL_CANT_KILL_THREAD error code, 5599
ER_GRP_RPL_CANT_READ_GRP_GTID_EXTRACTED error code, 5388
ER_GRP_RPL_CANT_READ_GTID error code, 5388
ER_GRP_RPL_CANT_READ_WRITE_SET_ITEM error code, 5388
ER_GRP_RPL_CERTIFICATE_SIZE_ERROR error code, 5384
ER_GRP_RPL_CERTIFICATION_INITIALIZATION_FAILURE error code, 5387
ER_GRP_RPL_CERTIFICATION_REC_PROCESS error code, 5405
ER_GRP_RPL_CERTIFIER_MSSG_PROCESS_ERROR error code, 5389
ER_GRP_RPL_CHANGE_GRP_MEM_NOT_PROCESSED error code, 5390
ER_GRP_RPL_CHANNEL_THREAD_WHEN_GROUP_ACTION_RUNNING error code, 5571
ER_GRP_RPL_CHECK_STATUS_TABLE error code, 5400
ER_GRP_RPL_CLONE_PROCESS_EXEC_ERROR error code, 5598
ER_GRP_RPL_CLONE_PROCESS_PREPARE_ERROR error code, 5598
ER_GRP_RPL_COMMUNICATION_SSL_CONF_INFO error code, 5411
ER_GRP_RPL_CONFIGURATION_ACTION_END error code, 5572
ER_GRP_RPL_CONFIGURATION_ACTION_ERROR error code, 5571
ER_GRP_RPL_CONFIGURATION_ACTION_KILLED_ERROR error code, 5572

ER_GRP_RPL_CONFIGURATION_ACTION_LOCAL_TERMINATION error code, 5571
ER_GRP_RPL_CONFIGURATION_ACTION_START error code, 5571
ER_GRP_RPL_CONFIG_RECOVERY error code, 5400
ER_GRP_RPL_CONFLICT_DETECTION_DISABLED error code, 5389
ER_GRP_RPL_CONN_INTERNAL_PLUGIN_FAIL error code, 5398
ER_GRP_RPL_CONTACT_WITH_SRV_FAILED error code, 5398
ER_GRP_RPL_COPY_FROM_EMPTY_STRING error code, 5399
ER_GRP_RPL_CREATE_APPLIER_CACHE_ERROR error code, 5384
ER_GRP_RPL_CREATE_GRP_RPL_REC_CHANNEL error code, 5400
ER_GRP_RPL_CREATE_SESSION_UNABLE error code, 5399
ER_GRP_RPL_DATA_NOT_PROVIDED_BY_MEM error code, 5392
ER_GRP_RPL_DEBUG_OPTIONS error code, 5417
ER_GRP_RPL_DEFAULT_TABLE_ENCRYPTION_DIFF_FROM_GRP error code, 5589
ER_GRP_RPL_DISABLE_READ_ONLY_FAILED error code, 5391
ER_GRP_RPL_DISABLE_SRV_READ_MODE_RESTRICTED error code, 5389
ER_GRP_RPL_DONOR_CONN_TERMINATION error code, 5400
ER_GRP_RPL_DONOR_SERVER_CONN error code, 5400
ER_GRP_RPL_DONOR_TRANS_INFO_ERROR error code, 5386
ER_GRP_RPL_ENABLE_READ_ONLY_FAILED error code, 5391
ER_GRP_RPL_ERROR_FETCHING_GTID_EXECUTED_SET error code, 5386
ER_GRP_RPL_ERROR_FETCHING_GTID_SET error code, 5386
ER_GRP_RPL_ERROR_GTID_EXECUTION_INFO error code, 5384
ER_GRP_RPL_ERROR_GTID_SET_EXTRACTION error code, 5570
ER_GRP_RPL_ERROR_MSG error code, 5362
ER_GRP_RPL_ERROR_ON_CERT_DB_INSTALL error code, 5584
ER_GRP_RPL_ERROR_ON_MESSAGE_SENDING error code, 5571
ER_GRP_RPL_ERROR_SENDING_SINGLE_PRIMARY_MSSG error code, 5385
ER_GRP_RPL_ERROR_STOPPING_CHANNELS error code, 5385
ER_GRP_RPL_ERROR_VERIFYING_SIDNO error code, 5387
ER_GRP_RPL_ERROR_WHILE_WAITING_FOR_CONFLICT_DETECTION error code, 5404
ER_GRP_RPL_ESTABLISHING_CONN_GRP_REC_DONOR error code, 5400
ER_GRP_RPL_ESTABLISH_RECOVERY_WITH_ANOTHER_DONOR error code, 5399
ER_GRP_RPL_ESTABLISH_RECOVERY_WITH_DONOR error code, 5399
ER_GRP_RPL_EVENT_HANDLING_ERROR error code, 5384
ER_GRP_RPL_EXCEEDS_AUTO_INC_VALUE error code, 5392
ER_GRP_RPL_EXIT_GRP_GCS_ERROR error code, 5417
ER_GRP_RPL_FAILED_TO_BOOTSTRAP_EVENT_HANDLING_INFRASTRUCTURE error code, 5414
ER_GRP_RPL_FAILED_TO_BROADCAST_GRP_MEMBERSHIP_NOTIFICATION error code, 5401
ER_GRP_RPL_FAILED_TO_BROADCAST_MEMBER_STATUS_NOTIFICATION error code, 5401
ER_GRP_RPL_FAILED_TO_CALL_GRP_COMMUNICATION_INTERFACE error code, 5407
ER_GRP_RPL_FAILED_TO_CONFIRM_IF_SERVER_LEFT_GRP error code, 5408
ER_GRP_RPL_FAILED_TO_CREATE_COMMIT_CACHE error code, 5405
ER_GRP_RPL_FAILED_TO_CREATE_TRANS_CONTEXT error code, 5403
ER_GRP_RPL_FAILED_TO_ENABLE_READ_ONLY_MODE_ON_SHUTDOWN error code, 5409
ER_GRP_RPL_FAILED_TO_ENABLE_SUPER_READ_ONLY_MODE error code, 5407
ER_GRP_RPL_FAILED_TO_EXTRACT_TRANS_WRITE_SET error code, 5403
ER_GRP_RPL_FAILED_TO_GATHER_TRANS_WRITE_SET error code, 5403
ER_GRP_RPL_FAILED_TO_GENERATE_SIDNO_FOR_GRP error code, 5411
ER_GRP_RPL_FAILED_TO_INIT_APPLIER_HANDLER error code, 5415
ER_GRP_RPL_FAILED_TO_INIT_APPLIER_MODULE error code, 5411
ER_GRP_RPL_FAILED_TO_INIT_COMMUNICATION_ENGINE error code, 5407
ER_GRP_RPL_FAILED_TO_INIT_HANDLER error code, 5409
ER_GRP_RPL_FAILED_TO_INSERT_TRX_ON_TCM_ON_AFTER_CERTIFICATION error code, 5582
ER_GRP_RPL_FAILED_TO_NOTIFY_GRP_MEMBERSHIP_EVENT error code, 5401

ER_GRP_RPL_FAILED_TO_PARSE_THE_GRP_NAME error code, 5410
ER_GRP_RPL_FAILED_TO_REGISTER_BINLOG_STATE_OBSERVER error code, 5410
ER_GRP_RPL_FAILED_TO_REGISTER_SERVER_STATE_OBSERVER error code, 5409
ER_GRP_RPL_FAILED_TO_REGISTER_TRANS_OUTCOME_NOTIFICTION error code, 5404
ER_GRP_RPL_FAILED_TO_REGISTER_TRANS_STATE_OBSERVER error code, 5410
ER_GRP_RPL_FAILED_TO_REINIT_BINLOG_CACHE_FOR_READ error code, 5403
ER_GRP_RPL_FAILED_TO_SHUTDOWN_REGISTRY_MODULE error code, 5409
ER_GRP_RPL_FAILED_TO_START_COMMUNICATION_ENGINE error code, 5407
ER_GRP_RPL_FAILED_TO_START_ON_BOOT error code, 5410
ER_GRP_RPL_FAILED_TO_START_ON_SECONDARY_WITH_ASYNC_CHANNELS error code, 5407
ER_GRP_RPL_FAILED_TO_START_WITH_INVALID_SERVER_ID error code, 5406
ER_GRP_RPL_FAILED_TO_STOP_ON_PLUGIN_UNINSTALL error code, 5410
ER_GRP_RPL_FAILED_TO_UNREGISTER_BINLOG_STATE_OBSERVER error code, 5410
ER_GRP_RPL_FAILED_TO_UNREGISTER_SERVER_STATE_OBSERVER error code, 5410
ER_GRP_RPL_FAILED_TO_UNREGISTER_TRANS_STATE_OBSERVER error code, 5410
ER_GRP_RPL_FATAL_REC_PROCESS error code, 5405
ER_GRP_RPL_FETCH_FORMAT_DESC_LOG_EVENT_FAILED error code, 5396
ER_GRP_RPL_FETCH_GTID_LOG_EVENT_FAILED error code, 5397
ER_GRP_RPL_FETCH_LOG_EVENT_FAILED error code, 5398
ER_GRP_RPL_FETCH_SNAPSHOT_VERSION_FAILED error code, 5396
ER_GRP_RPL_FETCH_TRANS_CONTEXT_FAILED error code, 5396
ER_GRP_RPL_FETCH_TRANS_CONTEXT_LOG_EVENT_FAILED error code, 5396
ER_GRP_RPL_FETCH_TRANS_DATA_FAILED error code, 5395
ER_GRP_RPL_FETCH_TRANS_SIDNO_ERROR error code, 5387
ER_GRP_RPL_FETCH_VIEW_CHANGE_LOG_EVENT_FAILED error code, 5397
ER_GRP_RPL_FINISHED_AUTO_REJOIN error code, 5589
ER_GRP_RPL_FK_WITH_CASCADE_UNSUPPORTED error code, 5396
ER_GRP_RPL_FLOW_CONTROL_STATS error code, 5418
ER_GRP_RPL_FLOW_CTRL_MAX_QUOTA_SMALLER_THAN_MIN_QUOTAS error code, 5413
ER_GRP_RPL_FLOW_CTRL_MIN_QUOTA_GREATER_THAN_MAX_QUOTA error code, 5413
ER_GRP_RPL_FLOW_CTRL_MIN_RECOVERY_QUOTA_GREATER_THAN_MAX_QUOTA error code, 5413
ER_GRP_RPL_FORCE_MEMBERS_MUST_BE_EMPTY error code, 5407
ER_GRP_RPL_FORCE_MEMBERS_SET_UPDATE_NOT_ALLOWED error code, 5414
ER_GRP_RPL_FORCE_MEMBERS_WHEN_LEAVING error code, 5584
ER_GRP_RPL_FORCE_MEMBER_VALUE_SET error code, 5418
ER_GRP_RPL_FORCE_MEMBER_VALUE_SET_ERROR error code, 5417
ER_GRP_RPL_FORCE_MEMBER_VALUE_TIME_OUT error code, 5418
ER_GRP_RPL_GCS_GR_ERROR_MSG error code, 5419
ER_GRP_RPL_GCS_INTERFACE_ERROR error code, 5417
ER_GRP_RPL_GMS_LISTENER_FAILED_TO_LOG_NOTIFICATION error code, 5417
ER_GRP_RPL_GROUP_NAME_PARSE_ERROR error code, 5385
ER_GRP_RPL_GRP_CHANGE_INFO_EXTRACT_ERROR error code, 5392
ER_GRP_RPL_GRP_COMMUNICATION_ENG_INIT_FAILED error code, 5417
ER_GRP_RPL_GRP_COMMUNICATION_INIT_WITH_CONF error code, 5414
ER_GRP_RPL_GRP_MEMBER_OFFLINE error code, 5417
ER_GRP_RPL_GRP_NAME_IS_NOT_VALID_UUID error code, 5413
ER_GRP_RPL_GRP_NAME_IS_TOO_LONG error code, 5413
ER_GRP_RPL_GRP_NAME_OPTION_MANDATORY error code, 5413
ER_GRP_RPL_GTID_ALREADY_USED error code, 5384
ER_GRP_RPL_GTID_EXECUTED_EXTRACT_ERROR error code, 5392
ER_GRP_RPL_GTID_MODE_OFF error code, 5412
ER_GRP_RPL_GTID_PURGED_EXTRACT_ERROR error code, 5598
ER_GRP_RPL_GTID_SET_EXTRACT_ERROR error code, 5393
ER_GRP_RPL_GTID_SET_EXTRACT_ERROR_DURING_RECOVERY error code, 5584

ER_GRP_RPL_INCORRECT_TYPE_SET_FOR_PARALLEL_APPLIER error code, 5412
ER_GRP_RPL_INIT_CERTIFICATION_INFO_FAILURE error code, 5388
ER_GRP_RPL_INTERNAL_QUERY error code, 5399
ER_GRP_RPL_INVALID_BINLOG_FORMAT error code, 5402
ER_GRP_RPL_INVALID_COMMUNICATION_PROTOCOL error code, 5589
ER_GRP_RPL_INVALID_DEBUG_OPTIONS error code, 5417
ER_GRP_RPL_INVALID_GTID_SET error code, 5387
ER_GRP_RPL_INVALID_SSL_RECOVERY_STRING error code, 5414
ER_GRP_RPL_INVALID_TRANS_WRITE_SET_EXTRACTION_VALUE error code, 5412
ER_GRP_RPL_IS_STOPPED error code, 5409
ER_GRP_RPL_IS_STOPPING error code, 5409
ER_GRP_RPL_JOINER_EXIT_WHEN_GROUP_ACTION_RUNNING error code, 5571
ER_GRP_RPL_JOIN_WHEN_GROUP_ACTION_RUNNING error code, 5571
ER_GRP_RPL_KILLED_FAILED_ID error code, 5398
ER_GRP_RPL_KILLED_SESSION_ID error code, 5398
ER_GRP_RPL_LOCAL_GTID_SETS_PROCESS_ERROR error code, 5393
ER_GRP_RPL_LOG_SLAVE_UPDATES_NOT_SET error code, 5412
ER_GRP_RPL_LOWER_CASE_TABLE_NAMES_DIFF_FROM_GRP error code, 5560
ER_GRP_RPL_MASTER_INFO_REPO_MUST_BE_TABLE error code, 5412
ER_GRP_RPL_MAXIMUM_CONNECTION_RETRIES_REACHED error code, 5399
ER_GRP_RPL_MEMBER_ADDED error code, 5390
ER_GRP_RPL_MEMBER_ALREADY_EXISTS error code, 5392
ER_GRP_RPL_MEMBER_CFG_INCOMPATIBLE_WITH_GRP_CFG error code, 5394
ER_GRP_RPL_MEMBER_CHANGE error code, 5391
ER_GRP_RPL_MEMBER_CONF_INFO error code, 5408
ER_GRP_RPL_MEMBER_CONTACT_RESTORED error code, 5390
ER_GRP_RPL_MEMBER_EXIT_PLUGIN_ERROR error code, 5391
ER_GRP_RPL_MEMBER_EXPELLED error code, 5391
ER_GRP_RPL_MEMBER_LEFT_GRP error code, 5391
ER_GRP_RPL_MEMBER_NOT_FOUND error code, 5399
ER_GRP_RPL_MEMBER_REMOVED error code, 5390
ER_GRP_RPL_MEMBER_SERVER_UUID_IS_INCOMPATIBLE_WITH_GRP error code, 5408
ER_GRP_RPL_MEMBER_STATS_INFO error code, 5418
ER_GRP_RPL_MEMBER_STOP_RPL_CHANNELS_ERROR error code, 5394
ER_GRP_RPL_MEMBER_TRANS_GREATER_THAN_GRP error code, 5393
ER_GRP_RPL_MEMBER_VERSION_LOWER_THAN_GRP error code, 5393
ER_GRP_RPL_MEMBER_VER_INCOMPATIBLE error code, 5393
ER_GRP_RPL_MEMBER_VER_READ_COMPATIBLE error code, 5595
ER_GRP_RPL_MEM_ONLINE error code, 5389
ER_GRP_RPL_MEM_REACHABLE error code, 5390
ER_GRP_RPL_MEM_UNREACHABLE error code, 5389
ER_GRP_RPL_MESSAGE_SERVICE_FATAL_ERROR error code, 5601
ER_GRP_RPL_MESSAGE_SERVICE_INIT_FAILURE error code, 5217
ER_GRP_RPL_MISSING_GRP_RPL_ACTION_COORDINATOR error code, 5570
ER_GRP_RPL_MISSING_GRP_RPL_APPLIER error code, 5389
ER_GRP_RPL_MODULE_TERMINATE_ERROR error code, 5413
ER_GRP_RPL_MSG_DISCARDED error code, 5389
ER_GRP_RPL_MSG_TOO_LONG_BROADCASTING_TRANS_FAILED error code, 5404
ER_GRP_RPL_MULTIPLE_CACHE_TYPE_NOT_SUPPORTED_FOR_SESSION error code, 5403
ER_GRP_RPL_NEEDS_INNODB_TABLE error code, 5395
ER_GRP_RPL_NEW_PRIMARY_ELECTED error code, 5391
ER_GRP_RPL_NOTIFY_CERTIFICATION_OUTCOME_FAILED error code, 5397
ER_GRP_RPL_NO_POSSIBLE_RECOVERY error code, 5598
ER_GRP_RPL_NO_STAGE_SERVICE error code, 5572

ER_GRP_RPL_NO_SUITABLE_PRIMARY_MEM error code, 5392
ER_GRP_RPL_NO_VALID_DONOR error code, 5400
ER_GRP_RPL_NULL_PACKET error code, 5388
ER_GRP_RPL_ONLY_ONE_SERVER_ALIVE error code, 5405
ER_GRP_RPL_OOM_FAILED_TO_GENERATE_IDENTIFICATION_HASH error code, 5401
ER_GRP_RPL_PIPELINE_CREATE_FAILED error code, 5418
ER_GRP_RPL_PIPELINE_FLUSH_FAIL error code, 5419
ER_GRP_RPL_PIPELINE_REINIT_FAILED_READ error code, 5419
ER_GRP_RPL_PIPELINE_REINIT_FAILED_WRITE error code, 5419
ER_GRP_RPL_PLUGIN_ABORT error code, 5567
ER_GRP_RPL_PLUGIN_STRUCT_INIT_NOT_POSSIBLE_ON_SERVER_START error code, 5407
ER_GRP_RPL_PREV_REC_SESSION_RUNNING error code, 5405
ER_GRP_RPL_PRIMARY_ELECTION_PROCESS_ERROR error code, 5572
ER_GRP_RPL_PRIMARY_ELECTION_STOP_ERROR error code, 5572
ER_GRP_RPL_PRIMARY_KEY_NOT_DEFINED error code, 5395
ER_GRP_RPL_PRIMARY_MEMBER_LEFT_GRP error code, 5390
ER_GRP_RPL_PROCESS_GTID_SET_ERROR error code, 5388
ER_GRP_RPL_PROCESS_INTERSECTION_GTID_SET_ERROR error code, 5388
ER_GRP_RPL_PURGE_APPLIER_LOGS error code, 5394
ER_GRP_RPL_PURGE_REC error code, 5401
ER_GRP_RPL_QUERY_FAIL error code, 5399
ER_GRP_RPL_READ_UNABLE_FOR_READ_ONLY_SUPER_READ_ONLY error code, 5406
ER_GRP_RPL_READ_UNABLE_FOR_SUPER_READ_ONLY error code, 5406
ER_GRP_RPL_RECEIVED_SET_MISSING_GTIDS error code, 5387
ER_GRP_RPL_RECOVERY_EVAL_ERROR error code, 5598
ER_GRP_RPL_RECOVERY_MODULE_TERMINATION_TIMED_OUT_ON_SHUTDOWN error code, 5409
ER_GRP_RPL_RECOVERY_STRAT_CHOICE error code, 5599
ER_GRP_RPL_RECOVERY_STRAT_CLONE_PURGED error code, 5599
ER_GRP_RPL_RECOVERY_STRAT_CLONE_THRESHOLD error code, 5599
ER_GRP_RPL_RECOVERY_STRAT_FALLBACK error code, 5599
ER_GRP_RPL_RECOVERY_STRAT_NO_FALLBACK error code, 5599
ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_DEPENDENCIES_FAILED error code, 5582
ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_GROUP_PREPARE_FAILED error code, 5582
ER_GRP_RPL_REGISTER_TRX_TO_WAIT_FOR_SYNC_BEFORE_EXECUTION_FAILED error code, 5583
ER_GRP_RPL_REINIT_OF_COMMIT_CACHE_FOR_WRITE_FAILED error code, 5405
ER_GRP_RPL_REINIT_OF_INTERNAL_CACHE_FOR_READ_FAILED error code, 5403
ER_GRP_RPL_REINIT_OF_INTERNAL_CACHE_FOR_WRITE_FAILED error code, 5404
ER_GRP_RPL_RELAY_LOG_INFO_REPO_MUST_BE_TABLE error code, 5412
ER_GRP_RPL_RELEASE_BEGIN_TRX_AFTER_DEPENDENCIES_COMMIT_FAILED error code, 5582
ER_GRP_RPL_RELEASE_BEGIN_TRX_AFTER_WAIT_FOR_SYNC_BEFORE_EXEC error code, 5583
ER_GRP_RPL_RELEASE_COMMIT_AFTER_GROUP_PREPARE_FAILED error code, 5581
ER_GRP_RPL_REQUESTING_NON_MEMBER_SERVER_TO_LEAVE error code, 5408
ER_GRP_RPL_RESET_APPLIER_MODULE_LOGS_ERROR error code, 5394
ER_GRP_RPL_SALVE_IO_THD_ON_SECONDARY_MEMBER error code, 5395
ER_GRP_RPL_SEND_STATS_ERROR error code, 5418
ER_GRP_RPL_SEND_TRX_PREPARED_MESSAGE_FAILED error code, 5581
ER_GRP_RPL_SEND_TRX_SYNC_BEFORE_EXECUTION_FAILED error code, 5583
ER_GRP_RPL_SERVER_ALREADY_LEFT error code, 5408
ER_GRP_RPL_SERVER_CONN_ERROR error code, 5386
ER_GRP_RPL_SERVER_IS_ALREADY_LEAVING error code, 5408
ER_GRP_RPL_SERVER_SET_TO_OFFLINE_MODE_DUE_TO_ERRORS error code, 5601
ER_GRP_RPL_SERVER_SET_TO_READ_ONLY_DUE_TO_ERRORS error code, 5416
ER_GRP_RPL_SERVER_UDF_ERROR error code, 5573
ER_GRP_RPL_SERVER_WORKING_AS_SECONDARY error code, 5406

ER_GRP_RPL_SESSION_OPEN_FAILED error code, 5391
ER_GRP_RPL_SET_GRP_COMMUNICATION_ENG_LOGGER_FAILED error code, 5417
ER_GRP_RPL_SET_STABLE_TRANS_ERROR error code, 5388
ER_GRP_RPL_SIDNO_FETCH_ERROR error code, 5385
ER_GRP_RPL_SINGLE_PRIM_MODE_NOT_ALLOWED_WITH_UPDATE_EVERYWHERE error code, 5413
ER_GRP_RPL_SKIP_COMPUTATION_TRANS_COMMITTED error code, 5388
ER_GRP_RPL_SLAVE_APPLIER_THREAD_ERROR_OUT error code, 5420
ER_GRP_RPL_SLAVE_APPLIER_THREAD_UNBLOCKED error code, 5419
ER_GRP_RPL_SLAVE_IO_THD_PRIMARY_UNKNOWN error code, 5395
ER_GRP_RPL_SLAVE_IO_THREAD_ERROR_OUT error code, 5419
ER_GRP_RPL_SLAVE_IO_THREAD_UNBLOCKED error code, 5419
ER_GRP_RPL_SLAVE_PRESERVE_COMMIT_ORDER_NOT_SET error code, 5412
ER_GRP_RPL_SLAVE_SQL_THD_ON_SECONDARY_MEMBER error code, 5395
ER_GRP_RPL_SLAVE_SQL_THD_PRIMARY_UNKNOWN error code, 5395
ER_GRP_RPL_SLAVE_THREAD_ERROR_ON_CLONE error code, 5599
ER_GRP_RPL_SQL_SERVICE_COMM_SESSION_NOT_INITIALIZED error code, 5415
ER_GRP_RPL_SQL_SERVICE_FAILED_TO_FETCH_SECURITY_CTX error code, 5416
ER_GRP_RPL_SQL_SERVICE_FAILED_TO_INIT_SESSION_THREAD error code, 5415
ER_GRP_RPL_SQL_SERVICE_FAILED_TO_RUN_SQL_QUERY error code, 5415
ER_GRP_RPL_SQL_SERVICE_MAX_CONN_ERROR_FROM_SERVER error code, 5416
ER_GRP_RPL_SQL_SERVICE_RETRIES_EXCEEDED_ON_SESSION_STATE error code, 5416
ER_GRP_RPL_SQL_SERVICE_SERVER_ACCESS_DENIED_FOR_USER error code, 5416
ER_GRP_RPL_SQL_SERVICE_SERVER_ERROR_ON_CONN error code, 5416
ER_GRP_RPL_SQL_SERVICE_SERVER_INTERNAL_FAILURE error code, 5415
ER_GRP_RPL_SQL_SERVICE_SERVER_SESSION_KILLED error code, 5415
ER_GRP_RPL_SRV_BLOCKED error code, 5390
ER_GRP_RPL_SRV_BLOCKED_FOR_SECS error code, 5390
ER_GRP_RPL_SRV_GTID_WAIT_ERROR error code, 5592
ER_GRP_RPL_SRV_NOT_ONLINE error code, 5389
ER_GRP_RPL_SRV_ONLINE error code, 5389
ER_GRP_RPL_SRV_PRIMARY_MEM error code, 5391
ER_GRP_RPL_SRV_SECONDARY_MEM error code, 5392
ER_GRP_RPL_SRV_WAIT_TIME_OUT error code, 5398
ER_GRP_RPL_SSL_DISABLED error code, 5411
ER_GRP_RPL_STARTED_AUTO_REJOIN error code, 5589
ER_GRP_RPL_STARTING_GRP_REC error code, 5400
ER_GRP_RPL_START_FAILED error code, 5393
ER_GRP_RPL_START_GRP_RPL_FAILED error code, 5398
ER_GRP_RPL_STOPPING_GRP_REC error code, 5400
ER_GRP_RPL_STOP_REP_CHANNEL error code, 5419
ER_GRP_RPL_SUPER_READ_OFF error code, 5398
ER_GRP_RPL_SUPER_READ_ON error code, 5398
ER_GRP_RPL_SUPER_READ_ONLY_ACTIVATE_ERROR error code, 5392
ER_GRP_RPL_SUPPORTS_ONLY_ONE_FORCE_MEMBERS_SET error code, 5414
ER_GRP_RPL_TIMEOUT_ON_VIEW_AFTER_JOINING_GRP error code, 5407
ER_GRP_RPL_TIMEOUT_RECEIVED_VC_ON_REJOIN error code, 5589
ER_GRP_RPL_TIMEOUT_RECEIVING_VIEW_CHANGE_ON_SHUTDOWN error code, 5408
ER_GRP_RPL_TRANS_GREATER_THAN_GRP error code, 5393
ER_GRP_RPL_TRANS_NOT_PRESENT_IN_GRP error code, 5393
ER_GRP_RPL_TRANS_SIZE_EXCEEDS_LIMIT error code, 5403
ER_GRP_RPL_TRANS_WRITE_SET_EXTRACTION_NOT_SET error code, 5402
ER_GRP_RPL_TRANS_WRITE_SET_EXTRACT_DIFF_FROM_GRP error code, 5394
ER_GRP_RPL_TRX_ALREADY_EXISTS_ON_TCM_ON_AFTER_CERTIFICATION error code, 5582
ER_GRP_RPL_TRX_DOES_NOT_EXIST_ON_TCM_ON_HANDLE_REMOTE_PREPARE error code, 5582

ER_GRP_RPL_TRX_WAIT_FOR_GROUP_GTID_EXECUTED error code, 5583
ER_GRP_RPL_TRX_WAIT_FOR_GROUP_PREPARE_FAILED error code, 5582
ER_GRP_RPL_TRX_WAIT_FOR_SYNC_BEFORE_EXECUTION_FAILED error code, 5583
ER_GRP_RPL_UDF_ERROR error code, 5217
ER_GRP_RPL_UDF_REGISTER_ERROR error code, 5572
ER_GRP_RPL_UDF_REGISTER_SERVICE_ERROR error code, 5572
ER_GRP_RPL_UDF_UNREGISTER_ERROR error code, 5572
ER_GRP_RPL_UNABLE_TO_CERTIFY_PLUGIN_TRANS error code, 5406
ER_GRP_RPL_UNABLE_TO_CONVERT_EVENT_TO_PACKET error code, 5419
ER_GRP_RPL_UNABLE_TO_CONVERT_PACKET_TO_EVENT error code, 5418
ER_GRP_RPL_UNABLE_TO_ENSURE_EXECUTION_REC error code, 5406
ER_GRP_RPL_UNABLE_TO_EVALUATE_APPLIER_STATUS error code, 5405
ER_GRP_RPL_UNABLE_TO_INIT_COMMUNICATION_ENGINE error code, 5411
ER_GRP_RPL_UNABLE_TO_KILL_CONN_REC_DONOR_APPLIER error code, 5401
ER_GRP_RPL_UNABLE_TO_KILL_CONN_REC_DONOR_FAILOVER error code, 5401
ER_GRP_RPL_UNABLE_TO_RESET_SERVER_READ_MODE error code, 5406
ER_GRP_RPL_UNBLOCK_CERTIFIED_TRANS error code, 5406
ER_GRP_RPL_UNBLOCK_WAITING_THD error code, 5384
ER_GRP_RPL_UNKNOWN_GRP_RPL_APPLIER_PIPELINE_REQUESTED error code, 5414
ER_GRP_RPL_UNREACHABLE_MAJORITY_TIMEOUT_FOR_MEMBER error code, 5416
ER_GRP_RPL_UNSUPPORTED_TRANS_ISOLATION error code, 5402
ER_GRP_RPL_UPDATE_GRPGTID_EXECUTED_ERROR error code, 5386
ER_GRP_RPL_UPDATE_GTID_SET_ERROR error code, 5387
ER_GRP_RPL_UPDATE_LAST_CONFLICT_FREE_TRANS_ERROR error code, 5387
ER_GRP_RPL_UPDATE_SERV_CERTIFICATE_FAILED error code, 5397
ER_GRP_RPL_UPDATE_TRANS_SNAPSHOT_REF_VER_ERROR error code, 5387
ER_GRP_RPL_UPDATE_TRANS_SNAPSHOT_VER_ERROR error code, 5385
ER_GRP_RPL_WAITING_FOR_VIEW_UPDATE error code, 5408
ER_GRP_RPL_WAIT_FOR_DEPENDENCIES_FAILED error code, 5583
ER_GRP_RPL_WHILE_SENDING_MSG_REC error code, 5406
ER_GRP_RPL_WHILE_STOPPING_REP_CHANNEL error code, 5405
ER_GRP_RPL_WRITE_IDENT_HASH_BASE64_ENCODING_FAILED error code, 5401
ER_GRP_RPL_WRITE_TO_BINLOG_CACHE_FAILED error code, 5404
ER_GRP_RPL_WRITE_TO_TRANSACTION_MESSAGE_FAILED error code, 5404
ER_GRP_TRX_CONSISTENCY_AFTER_ON_TRX_BEGIN error code, 5204
ER_GRP_TRX_CONSISTENCY_BEFORE error code, 5204
ER_GRP_TRX_CONSISTENCY_BEGIN_NOT_ALLOWED error code, 5204
ER_GRP_TRX_CONSISTENCY_NOT_ALLOWED error code, 5204
ER_GR_HOLD_KILLED error code, 5202
ER_GR_HOLD_MEMBER_STATUS_ERROR error code, 5202
ER_GR_HOLD_WAIT_TIMEOUT error code, 5202
ER_GTID_ALREADY_ADDED_BY_USER error code, 5261
ER_GTID_EXECUTED_WAS_CHANGED error code, 5154
ER_GTID_EXECUTED_WAS_UPDATED error code, 5324
ER_GTID_MODE_CAN_ONLY_CHANGE_ONE_STEP_AT_A_TIME error code, 5150
ER_GTID_MODE_OFF error code, 5162
ER_GTID_MODE_ON_REQUIRES_ENFORCE_GTID_CONSISTENCY_ON error code, 5149
ER_GTID_NEXT_CANT_BE_AUTOMATIC_IF_GTID_NEXT_LIST_IS_NON_NULL error code, 5148
ER_GTID_NEXT_TYPE_UNDEFINED_GROUP error code, 5153
ER_GTID_NEXT_TYPE_UNDEFINED_GTID error code, 5153
ER_GTID_PURGED_WAS_CHANGED error code, 5154
ER_GTID_PURGED_WAS_UPDATED error code, 5323
ER_GTID_UNSAFE_ALTER_ADD_COL_WITH_DEFAULT_EXPRESSION error code, 5201
ER_GTID_UNSAFE_BINLOG_SPLITTABLE_STATEMENT_AND_ASSIGNED_GTID error code, 5157

ER_GTID_UNSAFE_BINLOG_SPLITTABLE_STATEMENT_AND_GTID_GROUP error code, 5157
ER_GTID_UNSAFE_CREATE_DROP_TEMPORARY_TABLE_IN_TRANSACTION error code, 5150
ER_GTID_UNSAFE_CREATE_SELECT error code, 5150
ER_GTID_UNSAFE_NON_TRANSACTIONAL_TABLE error code, 5149
ER_HANDLERTON_OOM error code, 5245
ER_HANDSHAKE_ERROR error code, 5101
ER_HOSTNAME error code, 5129
ER_HOSTNAME_DOESNT_RESOLVE_TO error code, 5226
ER_HOSTNAME_RESEMBLES_IPV4 error code, 5226
ER_HOST_IS_BLOCKED error code, 5107
ER_HOST_NOT_PRIVILEGED error code, 5108
ER_IB_CLONE_GTID_PERSIST error code, 5446
ER_IB_CLONE_INTERNAL error code, 5444
ER_IB_CLONE_NON_INNODB_TABLE error code, 5598
ER_IB_CLONE_OPERATION error code, 5597
ER_IB_CLONE_PUNCH_HOLE error code, 5445
ER_IB_CLONE_RESTART error code, 5598
ER_IB_CLONE_SQL error code, 5445
ER_IB_CLONE_START_STOP error code, 5597
ER_IB_CLONE_STATUS_FILE error code, 5445
ER_IB_CLONE_TIMEOUT error code, 5445
ER_IB_CLONE_USER_DATA error code, 5598
ER_IB_CLONE_VALIDATE error code, 5445
ER_IB_LOB_ROLLBACK_INDEX_LEN error code, 5603
ER_IB_MSG_0 error code, 5429
ER_IB_MSG_1 error code, 5430
ER_IB_MSG_10 error code, 5430
ER_IB_MSG_100 error code, 5439
ER_IB_MSG_1000 error code, 5531
ER_IB_MSG_1001 error code, 5531
ER_IB_MSG_1002 error code, 5531
ER_IB_MSG_1003 error code, 5531
ER_IB_MSG_1004 error code, 5531
ER_IB_MSG_1005 error code, 5532
ER_IB_MSG_1006 error code, 5532
ER_IB_MSG_1007 error code, 5532
ER_IB_MSG_1008 error code, 5532
ER_IB_MSG_1009 error code, 5532
ER_IB_MSG_101 error code, 5440
ER_IB_MSG_1010 error code, 5532
ER_IB_MSG_1011 error code, 5532
ER_IB_MSG_1012 error code, 5532
ER_IB_MSG_1013 error code, 5532
ER_IB_MSG_1014 error code, 5532
ER_IB_MSG_1015 error code, 5533
ER_IB_MSG_1016 error code, 5533
ER_IB_MSG_1017 error code, 5533
ER_IB_MSG_1018 error code, 5533
ER_IB_MSG_1019 error code, 5533
ER_IB_MSG_102 error code, 5440
ER_IB_MSG_1020 error code, 5533
ER_IB_MSG_1021 error code, 5533
ER_IB_MSG_1022 error code, 5533
ER_IB_MSG_1023 error code, 5533

ER_IB_MSG_1024 error code, 5533
ER_IB_MSG_1025 error code, 5534
ER_IB_MSG_1026 error code, 5534
ER_IB_MSG_1027 error code, 5534
ER_IB_MSG_1028 error code, 5534
ER_IB_MSG_1029 error code, 5534
ER_IB_MSG_103 error code, 5440
ER_IB_MSG_1030 error code, 5534
ER_IB_MSG_1031 error code, 5534
ER_IB_MSG_1032 error code, 5534
ER_IB_MSG_1033 error code, 5534
ER_IB_MSG_1034 error code, 5534
ER_IB_MSG_1035 error code, 5535
ER_IB_MSG_1036 error code, 5535
ER_IB_MSG_1037 error code, 5535
ER_IB_MSG_1038 error code, 5535
ER_IB_MSG_1039 error code, 5535
ER_IB_MSG_104 error code, 5440
ER_IB_MSG_1040 error code, 5535
ER_IB_MSG_1041 error code, 5535
ER_IB_MSG_1042 error code, 5535
ER_IB_MSG_1043 error code, 5535
ER_IB_MSG_1044 error code, 5535
ER_IB_MSG_1045 error code, 5536
ER_IB_MSG_1046 error code, 5536
ER_IB_MSG_1047 error code, 5536
ER_IB_MSG_1048 error code, 5536
ER_IB_MSG_1049 error code, 5536
ER_IB_MSG_105 error code, 5440
ER_IB_MSG_1050 error code, 5536
ER_IB_MSG_1051 error code, 5536
ER_IB_MSG_1052 error code, 5536
ER_IB_MSG_1053 error code, 5536
ER_IB_MSG_1054 error code, 5537
ER_IB_MSG_1055 error code, 5537
ER_IB_MSG_1056 error code, 5537
ER_IB_MSG_1057 error code, 5537
ER_IB_MSG_1058 error code, 5537
ER_IB_MSG_1059 error code, 5537
ER_IB_MSG_106 error code, 5440
ER_IB_MSG_1060 error code, 5537
ER_IB_MSG_1061 error code, 5537
ER_IB_MSG_1062 error code, 5537
ER_IB_MSG_1063 error code, 5537
ER_IB_MSG_1064 error code, 5538
ER_IB_MSG_1065 error code, 5538
ER_IB_MSG_1066 error code, 5538
ER_IB_MSG_1067 error code, 5538
ER_IB_MSG_1068 error code, 5538
ER_IB_MSG_1069 error code, 5538
ER_IB_MSG_107 error code, 5440
ER_IB_MSG_1070 error code, 5538
ER_IB_MSG_1071 error code, 5538
ER_IB_MSG_1072 error code, 5538

ER_IB_MSG_1073 error code, 5538
ER_IB_MSG_1074 error code, 5539
ER_IB_MSG_1075 error code, 5539
ER_IB_MSG_1076 error code, 5539
ER_IB_MSG_1077 error code, 5539
ER_IB_MSG_1078 error code, 5539
ER_IB_MSG_1079 error code, 5539
ER_IB_MSG_108 error code, 5440
ER_IB_MSG_1080 error code, 5539
ER_IB_MSG_1081 error code, 5539
ER_IB_MSG_1082 error code, 5539
ER_IB_MSG_1083 error code, 5539
ER_IB_MSG_1084 error code, 5540
ER_IB_MSG_1085 error code, 5540
ER_IB_MSG_1086 error code, 5540
ER_IB_MSG_1087 error code, 5540
ER_IB_MSG_1088 error code, 5540
ER_IB_MSG_1089 error code, 5540
ER_IB_MSG_109 error code, 5440
ER_IB_MSG_1090 error code, 5540
ER_IB_MSG_1091 error code, 5540
ER_IB_MSG_1092 error code, 5540
ER_IB_MSG_1093 error code, 5540
ER_IB_MSG_1094 error code, 5541
ER_IB_MSG_1095 error code, 5541
ER_IB_MSG_1096 error code, 5541
ER_IB_MSG_1097 error code, 5541
ER_IB_MSG_1098 error code, 5541
ER_IB_MSG_1099 error code, 5541
ER_IB_MSG_11 error code, 5431
ER_IB_MSG_110 error code, 5440
ER_IB_MSG_1100 error code, 5541
ER_IB_MSG_1101 error code, 5541
ER_IB_MSG_1102 error code, 5541
ER_IB_MSG_1103 error code, 5541
ER_IB_MSG_1104 error code, 5542
ER_IB_MSG_1105 error code, 5542
ER_IB_MSG_1106 error code, 5542
ER_IB_MSG_1107 error code, 5542
ER_IB_MSG_1108 error code, 5542
ER_IB_MSG_1109 error code, 5542
ER_IB_MSG_111 error code, 5441
ER_IB_MSG_1110 error code, 5542
ER_IB_MSG_1111 error code, 5542
ER_IB_MSG_1112 error code, 5542
ER_IB_MSG_1113 error code, 5543
ER_IB_MSG_1114 error code, 5543
ER_IB_MSG_1115 error code, 5543
ER_IB_MSG_1116 error code, 5543
ER_IB_MSG_1117 error code, 5543
ER_IB_MSG_1118 error code, 5543
ER_IB_MSG_1119 error code, 5543
ER_IB_MSG_112 error code, 5441
ER_IB_MSG_1120 error code, 5543

ER_IB_MSG_1121 error code, 5543
ER_IB_MSG_1122 error code, 5543
ER_IB_MSG_1123 error code, 5544
ER_IB_MSG_1124 error code, 5544
ER_IB_MSG_1125 error code, 5544
ER_IB_MSG_1126 error code, 5544
ER_IB_MSG_1127 error code, 5544
ER_IB_MSG_1128 error code, 5544
ER_IB_MSG_1129 error code, 5544
ER_IB_MSG_113 error code, 5441
ER_IB_MSG_1130 error code, 5544
ER_IB_MSG_1131 error code, 5544
ER_IB_MSG_1132 error code, 5544
ER_IB_MSG_1133 error code, 5545
ER_IB_MSG_1134 error code, 5545
ER_IB_MSG_1135 error code, 5545
ER_IB_MSG_1136 error code, 5545
ER_IB_MSG_1137 error code, 5545
ER_IB_MSG_1138 error code, 5545
ER_IB_MSG_1139 error code, 5545
ER_IB_MSG_114 error code, 5441
ER_IB_MSG_1140 error code, 5545
ER_IB_MSG_1141 error code, 5545
ER_IB_MSG_1142 error code, 5546
ER_IB_MSG_1143 error code, 5546
ER_IB_MSG_1144 error code, 5546
ER_IB_MSG_1145 error code, 5546
ER_IB_MSG_1146 error code, 5546
ER_IB_MSG_1147 error code, 5546
ER_IB_MSG_1148 error code, 5546
ER_IB_MSG_1149 error code, 5546
ER_IB_MSG_115 error code, 5441
ER_IB_MSG_1150 error code, 5546
ER_IB_MSG_1151 error code, 5547
ER_IB_MSG_1152 error code, 5547
ER_IB_MSG_1153 error code, 5547
ER_IB_MSG_1154 error code, 5547
ER_IB_MSG_1155 error code, 5547
ER_IB_MSG_1156 error code, 5547
ER_IB_MSG_1157 error code, 5547
ER_IB_MSG_1158 error code, 5547
ER_IB_MSG_1159 error code, 5547
ER_IB_MSG_116 error code, 5441
ER_IB_MSG_1160 error code, 5548
ER_IB_MSG_1161 error code, 5548
ER_IB_MSG_1162 error code, 5548
ER_IB_MSG_1163 error code, 5548
ER_IB_MSG_1164 error code, 5548
ER_IB_MSG_1165 error code, 5548
ER_IB_MSG_1166 error code, 5548
ER_IB_MSG_1167 error code, 5548
ER_IB_MSG_1168 error code, 5548
ER_IB_MSG_1169 error code, 5548
ER_IB_MSG_117 error code, 5441

ER_IB_MSG_1170 error code, 5549
ER_IB_MSG_1171 error code, 5549
ER_IB_MSG_1172 error code, 5549
ER_IB_MSG_1173 error code, 5549
ER_IB_MSG_1174 error code, 5549
ER_IB_MSG_1175 error code, 5549
ER_IB_MSG_1176 error code, 5549
ER_IB_MSG_1177 error code, 5549
ER_IB_MSG_1178 error code, 5549
ER_IB_MSG_1179 error code, 5549
ER_IB_MSG_118 error code, 5441
ER_IB_MSG_1180 error code, 5550
ER_IB_MSG_1181 error code, 5550
ER_IB_MSG_1182 error code, 5550
ER_IB_MSG_1183 error code, 5550
ER_IB_MSG_1184 error code, 5550
ER_IB_MSG_1185 error code, 5550
ER_IB_MSG_1186 error code, 5550
ER_IB_MSG_1187 error code, 5550
ER_IB_MSG_1188 error code, 5550
ER_IB_MSG_1189 error code, 5550
ER_IB_MSG_119 error code, 5441
ER_IB_MSG_1190 error code, 5551
ER_IB_MSG_1191 error code, 5551
ER_IB_MSG_1192 error code, 5551
ER_IB_MSG_1193 error code, 5551
ER_IB_MSG_1194 error code, 5551
ER_IB_MSG_1195 error code, 5551
ER_IB_MSG_1196 error code, 5551
ER_IB_MSG_1197 error code, 5551
ER_IB_MSG_1198 error code, 5551
ER_IB_MSG_1199 error code, 5551
ER_IB_MSG_12 error code, 5431
ER_IB_MSG_120 error code, 5441
ER_IB_MSG_1200 error code, 5552
ER_IB_MSG_1201 error code, 5552
ER_IB_MSG_1202 error code, 5552
ER_IB_MSG_1203 error code, 5552
ER_IB_MSG_1204 error code, 5552
ER_IB_MSG_1205 error code, 5552
ER_IB_MSG_1206 error code, 5552
ER_IB_MSG_1207 error code, 5552
ER_IB_MSG_1208 error code, 5552
ER_IB_MSG_1209 error code, 5552
ER_IB_MSG_121 error code, 5442
ER_IB_MSG_1210 error code, 5553
ER_IB_MSG_1211 error code, 5553
ER_IB_MSG_1212 error code, 5553
ER_IB_MSG_1213 error code, 5553
ER_IB_MSG_1214 error code, 5553
ER_IB_MSG_1215 error code, 5553
ER_IB_MSG_1216 error code, 5553
ER_IB_MSG_1217 error code, 5553
ER_IB_MSG_1218 error code, 5553

ER_IB_MSG_1219 error code, 5553
ER_IB_MSG_122 error code, 5442
ER_IB_MSG_1220 error code, 5554
ER_IB_MSG_1221 error code, 5554
ER_IB_MSG_1222 error code, 5554
ER_IB_MSG_1223 error code, 5554
ER_IB_MSG_1224 error code, 5554
ER_IB_MSG_1225 error code, 5554
ER_IB_MSG_1226 error code, 5554
ER_IB_MSG_1227 error code, 5554
ER_IB_MSG_1228 error code, 5554
ER_IB_MSG_1229 error code, 5554
ER_IB_MSG_123 error code, 5442
ER_IB_MSG_1230 error code, 5555
ER_IB_MSG_1231 error code, 5555
ER_IB_MSG_1232 error code, 5555
ER_IB_MSG_1233 error code, 5555
ER_IB_MSG_1234 error code, 5555
ER_IB_MSG_1235 error code, 5555
ER_IB_MSG_1236 error code, 5555
ER_IB_MSG_1237 error code, 5555
ER_IB_MSG_1238 error code, 5555
ER_IB_MSG_1239 error code, 5555
ER_IB_MSG_124 error code, 5442
ER_IB_MSG_1240 error code, 5556
ER_IB_MSG_1241 error code, 5556
ER_IB_MSG_1242 error code, 5556
ER_IB_MSG_1243 error code, 5556
ER_IB_MSG_1244 error code, 5556
ER_IB_MSG_1245 error code, 5556
ER_IB_MSG_1246 error code, 5556
ER_IB_MSG_1247 error code, 5556
ER_IB_MSG_1248 error code, 5556
ER_IB_MSG_1249 error code, 5556
ER_IB_MSG_125 error code, 5442
ER_IB_MSG_1250 error code, 5557
ER_IB_MSG_1251 error code, 5557
ER_IB_MSG_1252 error code, 5557
ER_IB_MSG_1253 error code, 5557
ER_IB_MSG_1254 error code, 5557
ER_IB_MSG_1255 error code, 5557
ER_IB_MSG_1256 error code, 5557
ER_IB_MSG_1257 error code, 5557
ER_IB_MSG_1258 error code, 5557
ER_IB_MSG_1259 error code, 5557
ER_IB_MSG_126 error code, 5442
ER_IB_MSG_1260 error code, 5558
ER_IB_MSG_1261 error code, 5558
ER_IB_MSG_1262 error code, 5558
ER_IB_MSG_1263 error code, 5558
ER_IB_MSG_1264 error code, 5558
ER_IB_MSG_1265 error code, 5558
ER_IB_MSG_1266 error code, 5558
ER_IB_MSG_1267 error code, 5558

ER_IB_MSG_1268 error code, 5558
ER_IB_MSG_1269 error code, 5558
ER_IB_MSG_127 error code, 5442
ER_IB_MSG_1270 error code, 5559
ER_IB_MSG_1271 error code, 5566
ER_IB_MSG_1272 error code, 5567
ER_IB_MSG_1273 error code, 5568
ER_IB_MSG_1274 error code, 5568
ER_IB_MSG_1275 error code, 5569
ER_IB_MSG_1276 error code, 5569
ER_IB_MSG_1277 error code, 5569
ER_IB_MSG_1278 error code, 5569
ER_IB_MSG_1279 error code, 5569
ER_IB_MSG_128 error code, 5442
ER_IB_MSG_1280 error code, 5569
ER_IB_MSG_1281 error code, 5570
ER_IB_MSG_1282 error code, 5570
ER_IB_MSG_1283 error code, 5570
ER_IB_MSG_1284 error code, 5570
ER_IB_MSG_1285 error code, 5594
ER_IB_MSG_129 error code, 5442
ER_IB_MSG_13 error code, 5431
ER_IB_MSG_130 error code, 5442
ER_IB_MSG_131 error code, 5443
ER_IB_MSG_132 error code, 5443
ER_IB_MSG_133 error code, 5443
ER_IB_MSG_134 error code, 5443
ER_IB_MSG_135 error code, 5443
ER_IB_MSG_136 error code, 5443
ER_IB_MSG_137 error code, 5443
ER_IB_MSG_138 error code, 5443
ER_IB_MSG_139 error code, 5443
ER_IB_MSG_14 error code, 5431
ER_IB_MSG_140 error code, 5443
ER_IB_MSG_141 error code, 5444
ER_IB_MSG_142 error code, 5444
ER_IB_MSG_143 error code, 5444
ER_IB_MSG_144 error code, 5444
ER_IB_MSG_145 error code, 5444
ER_IB_MSG_146 error code, 5444
ER_IB_MSG_147 error code, 5444
ER_IB_MSG_148 error code, 5444
ER_IB_MSG_149 error code, 5444
ER_IB_MSG_15 error code, 5431
ER_IB_MSG_150 error code, 5445
ER_IB_MSG_151 error code, 5445
ER_IB_MSG_152 error code, 5445
ER_IB_MSG_153 error code, 5445
ER_IB_MSG_154 error code, 5445
ER_IB_MSG_155 error code, 5446
ER_IB_MSG_156 error code, 5446
ER_IB_MSG_157 error code, 5446
ER_IB_MSG_158 error code, 5446
ER_IB_MSG_159 error code, 5446

ER_IB_MSG_16 error code, 5431
ER_IB_MSG_160 error code, 5446
ER_IB_MSG_161 error code, 5446
ER_IB_MSG_162 error code, 5446
ER_IB_MSG_163 error code, 5446
ER_IB_MSG_164 error code, 5447
ER_IB_MSG_165 error code, 5447
ER_IB_MSG_166 error code, 5447
ER_IB_MSG_167 error code, 5447
ER_IB_MSG_168 error code, 5447
ER_IB_MSG_169 error code, 5447
ER_IB_MSG_17 error code, 5431
ER_IB_MSG_170 error code, 5447
ER_IB_MSG_171 error code, 5447
ER_IB_MSG_172 error code, 5447
ER_IB_MSG_173 error code, 5447
ER_IB_MSG_174 error code, 5448
ER_IB_MSG_175 error code, 5448
ER_IB_MSG_176 error code, 5448
ER_IB_MSG_177 error code, 5448
ER_IB_MSG_178 error code, 5448
ER_IB_MSG_179 error code, 5448
ER_IB_MSG_18 error code, 5431
ER_IB_MSG_180 error code, 5448
ER_IB_MSG_181 error code, 5448
ER_IB_MSG_182 error code, 5448
ER_IB_MSG_183 error code, 5448
ER_IB_MSG_184 error code, 5449
ER_IB_MSG_185 error code, 5449
ER_IB_MSG_186 error code, 5449
ER_IB_MSG_187 error code, 5449
ER_IB_MSG_188 error code, 5449
ER_IB_MSG_189 error code, 5449
ER_IB_MSG_19 error code, 5431
ER_IB_MSG_190 error code, 5449
ER_IB_MSG_191 error code, 5449
ER_IB_MSG_192 error code, 5449
ER_IB_MSG_193 error code, 5449
ER_IB_MSG_194 error code, 5450
ER_IB_MSG_195 error code, 5450
ER_IB_MSG_196 error code, 5450
ER_IB_MSG_197 error code, 5450
ER_IB_MSG_198 error code, 5450
ER_IB_MSG_199 error code, 5450
ER_IB_MSG_2 error code, 5430
ER_IB_MSG_20 error code, 5431
ER_IB_MSG_200 error code, 5450
ER_IB_MSG_201 error code, 5450
ER_IB_MSG_202 error code, 5450
ER_IB_MSG_203 error code, 5450
ER_IB_MSG_204 error code, 5451
ER_IB_MSG_205 error code, 5451
ER_IB_MSG_206 error code, 5451
ER_IB_MSG_207 error code, 5451

ER_IB_MSG_208 error code, 5451
ER_IB_MSG_209 error code, 5451
ER_IB_MSG_21 error code, 5432
ER_IB_MSG_210 error code, 5451
ER_IB_MSG_211 error code, 5451
ER_IB_MSG_212 error code, 5451
ER_IB_MSG_213 error code, 5451
ER_IB_MSG_214 error code, 5452
ER_IB_MSG_215 error code, 5452
ER_IB_MSG_216 error code, 5452
ER_IB_MSG_217 error code, 5452
ER_IB_MSG_218 error code, 5452
ER_IB_MSG_219 error code, 5452
ER_IB_MSG_22 error code, 5432
ER_IB_MSG_220 error code, 5452
ER_IB_MSG_221 error code, 5452
ER_IB_MSG_222 error code, 5452
ER_IB_MSG_223 error code, 5452
ER_IB_MSG_224 error code, 5453
ER_IB_MSG_225 error code, 5453
ER_IB_MSG_226 error code, 5453
ER_IB_MSG_227 error code, 5453
ER_IB_MSG_228 error code, 5453
ER_IB_MSG_229 error code, 5453
ER_IB_MSG_23 error code, 5432
ER_IB_MSG_230 error code, 5453
ER_IB_MSG_231 error code, 5453
ER_IB_MSG_232 error code, 5453
ER_IB_MSG_233 error code, 5453
ER_IB_MSG_234 error code, 5454
ER_IB_MSG_235 error code, 5454
ER_IB_MSG_236 error code, 5454
ER_IB_MSG_237 error code, 5454
ER_IB_MSG_238 error code, 5454
ER_IB_MSG_239 error code, 5454
ER_IB_MSG_24 error code, 5432
ER_IB_MSG_240 error code, 5454
ER_IB_MSG_241 error code, 5454
ER_IB_MSG_242 error code, 5454
ER_IB_MSG_243 error code, 5454
ER_IB_MSG_244 error code, 5455
ER_IB_MSG_245 error code, 5455
ER_IB_MSG_246 error code, 5455
ER_IB_MSG_247 error code, 5455
ER_IB_MSG_248 error code, 5455
ER_IB_MSG_249 error code, 5455
ER_IB_MSG_25 error code, 5432
ER_IB_MSG_250 error code, 5455
ER_IB_MSG_251 error code, 5455
ER_IB_MSG_252 error code, 5455
ER_IB_MSG_253 error code, 5455
ER_IB_MSG_254 error code, 5456
ER_IB_MSG_255 error code, 5456
ER_IB_MSG_256 error code, 5456

ER_IB_MSG_257 error code, 5456
ER_IB_MSG_258 error code, 5456
ER_IB_MSG_259 error code, 5456
ER_IB_MSG_26 error code, 5432
ER_IB_MSG_260 error code, 5456
ER_IB_MSG_261 error code, 5456
ER_IB_MSG_262 error code, 5456
ER_IB_MSG_263 error code, 5456
ER_IB_MSG_264 error code, 5457
ER_IB_MSG_265 error code, 5457
ER_IB_MSG_266 error code, 5457
ER_IB_MSG_267 error code, 5457
ER_IB_MSG_268 error code, 5457
ER_IB_MSG_269 error code, 5457
ER_IB_MSG_27 error code, 5432
ER_IB_MSG_270 error code, 5457
ER_IB_MSG_271 error code, 5457
ER_IB_MSG_272 error code, 5457
ER_IB_MSG_273 error code, 5457
ER_IB_MSG_274 error code, 5458
ER_IB_MSG_275 error code, 5458
ER_IB_MSG_276 error code, 5458
ER_IB_MSG_277 error code, 5458
ER_IB_MSG_278 error code, 5458
ER_IB_MSG_279 error code, 5458
ER_IB_MSG_28 error code, 5432
ER_IB_MSG_280 error code, 5458
ER_IB_MSG_281 error code, 5458
ER_IB_MSG_282 error code, 5458
ER_IB_MSG_283 error code, 5458
ER_IB_MSG_284 error code, 5459
ER_IB_MSG_285 error code, 5459
ER_IB_MSG_286 error code, 5459
ER_IB_MSG_287 error code, 5459
ER_IB_MSG_288 error code, 5459
ER_IB_MSG_289 error code, 5459
ER_IB_MSG_29 error code, 5432
ER_IB_MSG_290 error code, 5459
ER_IB_MSG_291 error code, 5459
ER_IB_MSG_292 error code, 5459
ER_IB_MSG_293 error code, 5459
ER_IB_MSG_294 error code, 5460
ER_IB_MSG_295 error code, 5460
ER_IB_MSG_296 error code, 5460
ER_IB_MSG_297 error code, 5460
ER_IB_MSG_298 error code, 5460
ER_IB_MSG_299 error code, 5460
ER_IB_MSG_3 error code, 5430
ER_IB_MSG_30 error code, 5432
ER_IB_MSG_300 error code, 5460
ER_IB_MSG_301 error code, 5460
ER_IB_MSG_302 error code, 5460
ER_IB_MSG_303 error code, 5461
ER_IB_MSG_304 error code, 5461

ER_IB_MSG_305 error code, 5461
ER_IB_MSG_306 error code, 5461
ER_IB_MSG_307 error code, 5461
ER_IB_MSG_308 error code, 5461
ER_IB_MSG_309 error code, 5461
ER_IB_MSG_31 error code, 5433
ER_IB_MSG_310 error code, 5461
ER_IB_MSG_311 error code, 5462
ER_IB_MSG_312 error code, 5462
ER_IB_MSG_313 error code, 5462
ER_IB_MSG_314 error code, 5462
ER_IB_MSG_315 error code, 5462
ER_IB_MSG_316 error code, 5462
ER_IB_MSG_317 error code, 5462
ER_IB_MSG_318 error code, 5462
ER_IB_MSG_319 error code, 5462
ER_IB_MSG_32 error code, 5433
ER_IB_MSG_320 error code, 5462
ER_IB_MSG_321 error code, 5463
ER_IB_MSG_322 error code, 5463
ER_IB_MSG_323 error code, 5463
ER_IB_MSG_324 error code, 5463
ER_IB_MSG_325 error code, 5463
ER_IB_MSG_326 error code, 5463
ER_IB_MSG_327 error code, 5463
ER_IB_MSG_328 error code, 5463
ER_IB_MSG_329 error code, 5463
ER_IB_MSG_33 error code, 5433
ER_IB_MSG_330 error code, 5463
ER_IB_MSG_331 error code, 5464
ER_IB_MSG_332 error code, 5464
ER_IB_MSG_333 error code, 5464
ER_IB_MSG_334 error code, 5464
ER_IB_MSG_335 error code, 5464
ER_IB_MSG_336 error code, 5464
ER_IB_MSG_337 error code, 5464
ER_IB_MSG_338 error code, 5464
ER_IB_MSG_339 error code, 5464
ER_IB_MSG_34 error code, 5433
ER_IB_MSG_340 error code, 5464
ER_IB_MSG_341 error code, 5465
ER_IB_MSG_342 error code, 5465
ER_IB_MSG_343 error code, 5465
ER_IB_MSG_344 error code, 5465
ER_IB_MSG_345 error code, 5465
ER_IB_MSG_346 error code, 5465
ER_IB_MSG_347 error code, 5465
ER_IB_MSG_348 error code, 5465
ER_IB_MSG_349 error code, 5465
ER_IB_MSG_35 error code, 5433
ER_IB_MSG_350 error code, 5465
ER_IB_MSG_351 error code, 5466
ER_IB_MSG_352 error code, 5466
ER_IB_MSG_353 error code, 5466

ER_IB_MSG_354 error code, 5466
ER_IB_MSG_355 error code, 5466
ER_IB_MSG_356 error code, 5466
ER_IB_MSG_357 error code, 5466
ER_IB_MSG_358 error code, 5466
ER_IB_MSG_359 error code, 5466
ER_IB_MSG_36 error code, 5433
ER_IB_MSG_360 error code, 5466
ER_IB_MSG_361 error code, 5467
ER_IB_MSG_362 error code, 5467
ER_IB_MSG_363 error code, 5467
ER_IB_MSG_364 error code, 5467
ER_IB_MSG_365 error code, 5467
ER_IB_MSG_366 error code, 5467
ER_IB_MSG_367 error code, 5467
ER_IB_MSG_368 error code, 5467
ER_IB_MSG_369 error code, 5467
ER_IB_MSG_37 error code, 5433
ER_IB_MSG_370 error code, 5468
ER_IB_MSG_371 error code, 5468
ER_IB_MSG_372 error code, 5468
ER_IB_MSG_373 error code, 5468
ER_IB_MSG_374 error code, 5468
ER_IB_MSG_375 error code, 5468
ER_IB_MSG_376 error code, 5468
ER_IB_MSG_377 error code, 5468
ER_IB_MSG_378 error code, 5468
ER_IB_MSG_379 error code, 5468
ER_IB_MSG_38 error code, 5433
ER_IB_MSG_380 error code, 5469
ER_IB_MSG_381 error code, 5469
ER_IB_MSG_382 error code, 5469
ER_IB_MSG_383 error code, 5469
ER_IB_MSG_384 error code, 5469
ER_IB_MSG_385 error code, 5469
ER_IB_MSG_386 error code, 5469
ER_IB_MSG_387 error code, 5469
ER_IB_MSG_388 error code, 5469
ER_IB_MSG_389 error code, 5469
ER_IB_MSG_39 error code, 5433
ER_IB_MSG_390 error code, 5470
ER_IB_MSG_391 error code, 5470
ER_IB_MSG_392 error code, 5470
ER_IB_MSG_393 error code, 5470
ER_IB_MSG_394 error code, 5470
ER_IB_MSG_395 error code, 5470
ER_IB_MSG_396 error code, 5470
ER_IB_MSG_397 error code, 5470
ER_IB_MSG_398 error code, 5470
ER_IB_MSG_399 error code, 5470
ER_IB_MSG_4 error code, 5430
ER_IB_MSG_40 error code, 5433
ER_IB_MSG_400 error code, 5471
ER_IB_MSG_401 error code, 5471

ER_IB_MSG_402 error code, 5471
ER_IB_MSG_403 error code, 5471
ER_IB_MSG_404 error code, 5471
ER_IB_MSG_405 error code, 5471
ER_IB_MSG_406 error code, 5471
ER_IB_MSG_407 error code, 5471
ER_IB_MSG_408 error code, 5471
ER_IB_MSG_409 error code, 5471
ER_IB_MSG_41 error code, 5434
ER_IB_MSG_410 error code, 5472
ER_IB_MSG_411 error code, 5472
ER_IB_MSG_412 error code, 5472
ER_IB_MSG_413 error code, 5472
ER_IB_MSG_414 error code, 5472
ER_IB_MSG_415 error code, 5472
ER_IB_MSG_416 error code, 5472
ER_IB_MSG_417 error code, 5472
ER_IB_MSG_418 error code, 5472
ER_IB_MSG_419 error code, 5472
ER_IB_MSG_42 error code, 5434
ER_IB_MSG_420 error code, 5473
ER_IB_MSG_421 error code, 5473
ER_IB_MSG_422 error code, 5473
ER_IB_MSG_423 error code, 5473
ER_IB_MSG_424 error code, 5473
ER_IB_MSG_425 error code, 5473
ER_IB_MSG_426 error code, 5473
ER_IB_MSG_427 error code, 5473
ER_IB_MSG_428 error code, 5473
ER_IB_MSG_429 error code, 5473
ER_IB_MSG_43 error code, 5434
ER_IB_MSG_430 error code, 5474
ER_IB_MSG_431 error code, 5474
ER_IB_MSG_432 error code, 5474
ER_IB_MSG_433 error code, 5474
ER_IB_MSG_434 error code, 5474
ER_IB_MSG_435 error code, 5474
ER_IB_MSG_436 error code, 5474
ER_IB_MSG_437 error code, 5474
ER_IB_MSG_438 error code, 5474
ER_IB_MSG_439 error code, 5474
ER_IB_MSG_44 error code, 5434
ER_IB_MSG_440 error code, 5475
ER_IB_MSG_441 error code, 5475
ER_IB_MSG_442 error code, 5475
ER_IB_MSG_443 error code, 5475
ER_IB_MSG_444 error code, 5475
ER_IB_MSG_445 error code, 5475
ER_IB_MSG_446 error code, 5475
ER_IB_MSG_447 error code, 5475
ER_IB_MSG_448 error code, 5475
ER_IB_MSG_449 error code, 5475
ER_IB_MSG_45 error code, 5434
ER_IB_MSG_450 error code, 5476

ER_IB_MSG_451 error code, 5476
ER_IB_MSG_452 error code, 5476
ER_IB_MSG_453 error code, 5476
ER_IB_MSG_454 error code, 5476
ER_IB_MSG_455 error code, 5476
ER_IB_MSG_456 error code, 5476
ER_IB_MSG_457 error code, 5476
ER_IB_MSG_458 error code, 5476
ER_IB_MSG_459 error code, 5476
ER_IB_MSG_46 error code, 5434
ER_IB_MSG_460 error code, 5477
ER_IB_MSG_461 error code, 5477
ER_IB_MSG_462 error code, 5477
ER_IB_MSG_463 error code, 5477
ER_IB_MSG_464 error code, 5477
ER_IB_MSG_465 error code, 5477
ER_IB_MSG_466 error code, 5477
ER_IB_MSG_467 error code, 5477
ER_IB_MSG_468 error code, 5477
ER_IB_MSG_469 error code, 5477
ER_IB_MSG_47 error code, 5434
ER_IB_MSG_470 error code, 5478
ER_IB_MSG_471 error code, 5478
ER_IB_MSG_472 error code, 5478
ER_IB_MSG_473 error code, 5478
ER_IB_MSG_474 error code, 5478
ER_IB_MSG_475 error code, 5478
ER_IB_MSG_476 error code, 5478
ER_IB_MSG_477 error code, 5478
ER_IB_MSG_478 error code, 5478
ER_IB_MSG_479 error code, 5478
ER_IB_MSG_48 error code, 5434
ER_IB_MSG_480 error code, 5479
ER_IB_MSG_481 error code, 5479
ER_IB_MSG_482 error code, 5479
ER_IB_MSG_483 error code, 5479
ER_IB_MSG_484 error code, 5479
ER_IB_MSG_485 error code, 5479
ER_IB_MSG_486 error code, 5479
ER_IB_MSG_487 error code, 5479
ER_IB_MSG_488 error code, 5479
ER_IB_MSG_489 error code, 5479
ER_IB_MSG_49 error code, 5434
ER_IB_MSG_490 error code, 5480
ER_IB_MSG_491 error code, 5480
ER_IB_MSG_492 error code, 5480
ER_IB_MSG_493 error code, 5480
ER_IB_MSG_494 error code, 5480
ER_IB_MSG_495 error code, 5480
ER_IB_MSG_496 error code, 5480
ER_IB_MSG_497 error code, 5480
ER_IB_MSG_498 error code, 5480
ER_IB_MSG_499 error code, 5480
ER_IB_MSG_5 error code, 5430

ER_IB_MSG_50 error code, 5434
ER_IB_MSG_500 error code, 5481
ER_IB_MSG_501 error code, 5481
ER_IB_MSG_502 error code, 5481
ER_IB_MSG_503 error code, 5481
ER_IB_MSG_504 error code, 5481
ER_IB_MSG_505 error code, 5481
ER_IB_MSG_506 error code, 5481
ER_IB_MSG_507 error code, 5481
ER_IB_MSG_508 error code, 5481
ER_IB_MSG_509 error code, 5481
ER_IB_MSG_51 error code, 5435
ER_IB_MSG_510 error code, 5482
ER_IB_MSG_511 error code, 5482
ER_IB_MSG_512 error code, 5482
ER_IB_MSG_513 error code, 5482
ER_IB_MSG_514 error code, 5482
ER_IB_MSG_515 error code, 5482
ER_IB_MSG_516 error code, 5482
ER_IB_MSG_517 error code, 5482
ER_IB_MSG_518 error code, 5482
ER_IB_MSG_519 error code, 5482
ER_IB_MSG_52 error code, 5435
ER_IB_MSG_520 error code, 5483
ER_IB_MSG_521 error code, 5483
ER_IB_MSG_522 error code, 5483
ER_IB_MSG_523 error code, 5483
ER_IB_MSG_524 error code, 5483
ER_IB_MSG_525 error code, 5483
ER_IB_MSG_526 error code, 5483
ER_IB_MSG_527 error code, 5483
ER_IB_MSG_528 error code, 5483
ER_IB_MSG_529 error code, 5483
ER_IB_MSG_53 error code, 5435
ER_IB_MSG_530 error code, 5484
ER_IB_MSG_531 error code, 5484
ER_IB_MSG_532 error code, 5484
ER_IB_MSG_533 error code, 5484
ER_IB_MSG_534 error code, 5484
ER_IB_MSG_535 error code, 5484
ER_IB_MSG_536 error code, 5484
ER_IB_MSG_537 error code, 5484
ER_IB_MSG_538 error code, 5484
ER_IB_MSG_539 error code, 5484
ER_IB_MSG_54 error code, 5435
ER_IB_MSG_540 error code, 5485
ER_IB_MSG_541 error code, 5485
ER_IB_MSG_542 error code, 5485
ER_IB_MSG_543 error code, 5485
ER_IB_MSG_544 error code, 5485
ER_IB_MSG_545 error code, 5485
ER_IB_MSG_546 error code, 5485
ER_IB_MSG_547 error code, 5485
ER_IB_MSG_548 error code, 5485

ER_IB_MSG_549 error code, 5485
ER_IB_MSG_55 error code, 5435
ER_IB_MSG_550 error code, 5486
ER_IB_MSG_551 error code, 5486
ER_IB_MSG_552 error code, 5486
ER_IB_MSG_553 error code, 5486
ER_IB_MSG_554 error code, 5486
ER_IB_MSG_555 error code, 5486
ER_IB_MSG_556 error code, 5486
ER_IB_MSG_557 error code, 5486
ER_IB_MSG_558 error code, 5486
ER_IB_MSG_559 error code, 5486
ER_IB_MSG_56 error code, 5435
ER_IB_MSG_560 error code, 5487
ER_IB_MSG_561 error code, 5487
ER_IB_MSG_562 error code, 5487
ER_IB_MSG_563 error code, 5487
ER_IB_MSG_564 error code, 5487
ER_IB_MSG_565 error code, 5487
ER_IB_MSG_566 error code, 5487
ER_IB_MSG_567 error code, 5487
ER_IB_MSG_568 error code, 5487
ER_IB_MSG_569 error code, 5487
ER_IB_MSG_57 error code, 5435
ER_IB_MSG_570 error code, 5488
ER_IB_MSG_571 error code, 5488
ER_IB_MSG_572 error code, 5488
ER_IB_MSG_573 error code, 5488
ER_IB_MSG_574 error code, 5488
ER_IB_MSG_575 error code, 5488
ER_IB_MSG_576 error code, 5488
ER_IB_MSG_577 error code, 5488
ER_IB_MSG_578 error code, 5488
ER_IB_MSG_579 error code, 5488
ER_IB_MSG_58 error code, 5435
ER_IB_MSG_580 error code, 5489
ER_IB_MSG_581 error code, 5489
ER_IB_MSG_582 error code, 5489
ER_IB_MSG_583 error code, 5489
ER_IB_MSG_584 error code, 5489
ER_IB_MSG_585 error code, 5489
ER_IB_MSG_586 error code, 5489
ER_IB_MSG_587 error code, 5489
ER_IB_MSG_588 error code, 5489
ER_IB_MSG_589 error code, 5489
ER_IB_MSG_59 error code, 5435
ER_IB_MSG_590 error code, 5490
ER_IB_MSG_591 error code, 5490
ER_IB_MSG_592 error code, 5490
ER_IB_MSG_593 error code, 5490
ER_IB_MSG_594 error code, 5490
ER_IB_MSG_595 error code, 5490
ER_IB_MSG_596 error code, 5490
ER_IB_MSG_597 error code, 5490

ER_IB_MSG_598 error code, 5490
ER_IB_MSG_599 error code, 5490
ER_IB_MSG_6 error code, 5430
ER_IB_MSG_60 error code, 5435
ER_IB_MSG_600 error code, 5491
ER_IB_MSG_601 error code, 5491
ER_IB_MSG_602 error code, 5491
ER_IB_MSG_603 error code, 5491
ER_IB_MSG_604 error code, 5491
ER_IB_MSG_605 error code, 5491
ER_IB_MSG_606 error code, 5491
ER_IB_MSG_607 error code, 5491
ER_IB_MSG_608 error code, 5491
ER_IB_MSG_609 error code, 5491
ER_IB_MSG_61 error code, 5436
ER_IB_MSG_610 error code, 5492
ER_IB_MSG_611 error code, 5492
ER_IB_MSG_612 error code, 5492
ER_IB_MSG_613 error code, 5492
ER_IB_MSG_614 error code, 5492
ER_IB_MSG_615 error code, 5492
ER_IB_MSG_616 error code, 5492
ER_IB_MSG_617 error code, 5492
ER_IB_MSG_618 error code, 5492
ER_IB_MSG_619 error code, 5492
ER_IB_MSG_62 error code, 5436
ER_IB_MSG_620 error code, 5493
ER_IB_MSG_621 error code, 5493
ER_IB_MSG_622 error code, 5493
ER_IB_MSG_623 error code, 5493
ER_IB_MSG_624 error code, 5493
ER_IB_MSG_625 error code, 5493
ER_IB_MSG_626 error code, 5493
ER_IB_MSG_627 error code, 5493
ER_IB_MSG_628 error code, 5493
ER_IB_MSG_629 error code, 5493
ER_IB_MSG_63 error code, 5436
ER_IB_MSG_630 error code, 5494
ER_IB_MSG_631 error code, 5494
ER_IB_MSG_632 error code, 5494
ER_IB_MSG_633 error code, 5494
ER_IB_MSG_634 error code, 5494
ER_IB_MSG_635 error code, 5494
ER_IB_MSG_636 error code, 5494
ER_IB_MSG_637 error code, 5494
ER_IB_MSG_638 error code, 5494
ER_IB_MSG_639 error code, 5494
ER_IB_MSG_64 error code, 5436
ER_IB_MSG_640 error code, 5495
ER_IB_MSG_641 error code, 5495
ER_IB_MSG_642 error code, 5495
ER_IB_MSG_643 error code, 5495
ER_IB_MSG_644 error code, 5495
ER_IB_MSG_645 error code, 5495

ER_IB_MSG_646 error code, 5495
ER_IB_MSG_647 error code, 5495
ER_IB_MSG_648 error code, 5495
ER_IB_MSG_649 error code, 5495
ER_IB_MSG_65 error code, 5436
ER_IB_MSG_650 error code, 5496
ER_IB_MSG_651 error code, 5496
ER_IB_MSG_652 error code, 5496
ER_IB_MSG_653 error code, 5496
ER_IB_MSG_654 error code, 5496
ER_IB_MSG_655 error code, 5496
ER_IB_MSG_656 error code, 5496
ER_IB_MSG_657 error code, 5496
ER_IB_MSG_658 error code, 5496
ER_IB_MSG_659 error code, 5497
ER_IB_MSG_66 error code, 5436
ER_IB_MSG_660 error code, 5497
ER_IB_MSG_661 error code, 5497
ER_IB_MSG_662 error code, 5497
ER_IB_MSG_663 error code, 5497
ER_IB_MSG_664 error code, 5497
ER_IB_MSG_665 error code, 5497
ER_IB_MSG_666 error code, 5497
ER_IB_MSG_667 error code, 5497
ER_IB_MSG_668 error code, 5498
ER_IB_MSG_669 error code, 5498
ER_IB_MSG_67 error code, 5436
ER_IB_MSG_670 error code, 5498
ER_IB_MSG_671 error code, 5498
ER_IB_MSG_672 error code, 5498
ER_IB_MSG_673 error code, 5498
ER_IB_MSG_674 error code, 5498
ER_IB_MSG_675 error code, 5498
ER_IB_MSG_676 error code, 5498
ER_IB_MSG_677 error code, 5498
ER_IB_MSG_678 error code, 5499
ER_IB_MSG_679 error code, 5499
ER_IB_MSG_68 error code, 5436
ER_IB_MSG_680 error code, 5499
ER_IB_MSG_681 error code, 5499
ER_IB_MSG_682 error code, 5499
ER_IB_MSG_683 error code, 5499
ER_IB_MSG_684 error code, 5499
ER_IB_MSG_685 error code, 5499
ER_IB_MSG_686 error code, 5499
ER_IB_MSG_687 error code, 5499
ER_IB_MSG_688 error code, 5500
ER_IB_MSG_689 error code, 5500
ER_IB_MSG_69 error code, 5436
ER_IB_MSG_690 error code, 5500
ER_IB_MSG_691 error code, 5500
ER_IB_MSG_692 error code, 5500
ER_IB_MSG_693 error code, 5500
ER_IB_MSG_694 error code, 5500

ER_IB_MSG_695 error code, 5500
ER_IB_MSG_696 error code, 5500
ER_IB_MSG_697 error code, 5501
ER_IB_MSG_698 error code, 5501
ER_IB_MSG_699 error code, 5501
ER_IB_MSG_7 error code, 5430
ER_IB_MSG_70 error code, 5436
ER_IB_MSG_700 error code, 5501
ER_IB_MSG_701 error code, 5501
ER_IB_MSG_702 error code, 5501
ER_IB_MSG_703 error code, 5501
ER_IB_MSG_704 error code, 5501
ER_IB_MSG_705 error code, 5501
ER_IB_MSG_706 error code, 5501
ER_IB_MSG_707 error code, 5502
ER_IB_MSG_708 error code, 5502
ER_IB_MSG_709 error code, 5502
ER_IB_MSG_71 error code, 5437
ER_IB_MSG_710 error code, 5502
ER_IB_MSG_711 error code, 5502
ER_IB_MSG_712 error code, 5502
ER_IB_MSG_713 error code, 5502
ER_IB_MSG_714 error code, 5502
ER_IB_MSG_715 error code, 5502
ER_IB_MSG_716 error code, 5503
ER_IB_MSG_717 error code, 5503
ER_IB_MSG_718 error code, 5503
ER_IB_MSG_719 error code, 5503
ER_IB_MSG_72 error code, 5437
ER_IB_MSG_720 error code, 5503
ER_IB_MSG_721 error code, 5503
ER_IB_MSG_722 error code, 5503
ER_IB_MSG_723 error code, 5503
ER_IB_MSG_724 error code, 5503
ER_IB_MSG_725 error code, 5503
ER_IB_MSG_726 error code, 5504
ER_IB_MSG_727 error code, 5504
ER_IB_MSG_728 error code, 5504
ER_IB_MSG_729 error code, 5504
ER_IB_MSG_73 error code, 5437
ER_IB_MSG_730 error code, 5504
ER_IB_MSG_731 error code, 5504
ER_IB_MSG_732 error code, 5504
ER_IB_MSG_733 error code, 5504
ER_IB_MSG_734 error code, 5504
ER_IB_MSG_735 error code, 5505
ER_IB_MSG_736 error code, 5505
ER_IB_MSG_737 error code, 5505
ER_IB_MSG_738 error code, 5505
ER_IB_MSG_739 error code, 5505
ER_IB_MSG_74 error code, 5437
ER_IB_MSG_740 error code, 5505
ER_IB_MSG_741 error code, 5505
ER_IB_MSG_742 error code, 5505

ER_IB_MSG_743 error code, 5505
ER_IB_MSG_744 error code, 5505
ER_IB_MSG_745 error code, 5506
ER_IB_MSG_746 error code, 5506
ER_IB_MSG_747 error code, 5506
ER_IB_MSG_748 error code, 5506
ER_IB_MSG_749 error code, 5506
ER_IB_MSG_75 error code, 5437
ER_IB_MSG_750 error code, 5506
ER_IB_MSG_751 error code, 5506
ER_IB_MSG_752 error code, 5506
ER_IB_MSG_753 error code, 5506
ER_IB_MSG_754 error code, 5506
ER_IB_MSG_755 error code, 5507
ER_IB_MSG_756 error code, 5507
ER_IB_MSG_757 error code, 5507
ER_IB_MSG_758 error code, 5507
ER_IB_MSG_759 error code, 5507
ER_IB_MSG_76 error code, 5437
ER_IB_MSG_760 error code, 5507
ER_IB_MSG_761 error code, 5507
ER_IB_MSG_762 error code, 5507
ER_IB_MSG_763 error code, 5507
ER_IB_MSG_764 error code, 5507
ER_IB_MSG_765 error code, 5508
ER_IB_MSG_766 error code, 5508
ER_IB_MSG_767 error code, 5508
ER_IB_MSG_768 error code, 5508
ER_IB_MSG_769 error code, 5508
ER_IB_MSG_77 error code, 5437
ER_IB_MSG_770 error code, 5508
ER_IB_MSG_771 error code, 5508
ER_IB_MSG_772 error code, 5508
ER_IB_MSG_773 error code, 5508
ER_IB_MSG_774 error code, 5508
ER_IB_MSG_775 error code, 5509
ER_IB_MSG_776 error code, 5509
ER_IB_MSG_777 error code, 5509
ER_IB_MSG_778 error code, 5509
ER_IB_MSG_779 error code, 5509
ER_IB_MSG_78 error code, 5437
ER_IB_MSG_780 error code, 5509
ER_IB_MSG_781 error code, 5509
ER_IB_MSG_782 error code, 5509
ER_IB_MSG_783 error code, 5509
ER_IB_MSG_784 error code, 5509
ER_IB_MSG_785 error code, 5510
ER_IB_MSG_786 error code, 5510
ER_IB_MSG_787 error code, 5510
ER_IB_MSG_788 error code, 5510
ER_IB_MSG_789 error code, 5510
ER_IB_MSG_79 error code, 5437
ER_IB_MSG_790 error code, 5510
ER_IB_MSG_791 error code, 5510

ER_IB_MSG_792 error code, 5510
ER_IB_MSG_793 error code, 5510
ER_IB_MSG_794 error code, 5510
ER_IB_MSG_795 error code, 5511
ER_IB_MSG_796 error code, 5511
ER_IB_MSG_797 error code, 5511
ER_IB_MSG_798 error code, 5511
ER_IB_MSG_799 error code, 5511
ER_IB_MSG_8 error code, 5430
ER_IB_MSG_80 error code, 5437
ER_IB_MSG_800 error code, 5511
ER_IB_MSG_801 error code, 5511
ER_IB_MSG_802 error code, 5511
ER_IB_MSG_803 error code, 5511
ER_IB_MSG_804 error code, 5511
ER_IB_MSG_805 error code, 5512
ER_IB_MSG_806 error code, 5512
ER_IB_MSG_807 error code, 5512
ER_IB_MSG_808 error code, 5512
ER_IB_MSG_809 error code, 5512
ER_IB_MSG_81 error code, 5438
ER_IB_MSG_810 error code, 5512
ER_IB_MSG_811 error code, 5512
ER_IB_MSG_812 error code, 5512
ER_IB_MSG_813 error code, 5512
ER_IB_MSG_814 error code, 5512
ER_IB_MSG_815 error code, 5513
ER_IB_MSG_816 error code, 5513
ER_IB_MSG_817 error code, 5513
ER_IB_MSG_818 error code, 5513
ER_IB_MSG_819 error code, 5513
ER_IB_MSG_82 error code, 5438
ER_IB_MSG_820 error code, 5513
ER_IB_MSG_821 error code, 5513
ER_IB_MSG_822 error code, 5513
ER_IB_MSG_823 error code, 5513
ER_IB_MSG_824 error code, 5513
ER_IB_MSG_825 error code, 5514
ER_IB_MSG_826 error code, 5514
ER_IB_MSG_827 error code, 5514
ER_IB_MSG_828 error code, 5514
ER_IB_MSG_829 error code, 5514
ER_IB_MSG_83 error code, 5438
ER_IB_MSG_830 error code, 5514
ER_IB_MSG_831 error code, 5514
ER_IB_MSG_832 error code, 5514
ER_IB_MSG_833 error code, 5514
ER_IB_MSG_834 error code, 5514
ER_IB_MSG_835 error code, 5515
ER_IB_MSG_836 error code, 5515
ER_IB_MSG_837 error code, 5515
ER_IB_MSG_838 error code, 5515
ER_IB_MSG_839 error code, 5515
ER_IB_MSG_84 error code, 5438

ER_IB_MSG_840 error code, 5515
ER_IB_MSG_841 error code, 5515
ER_IB_MSG_842 error code, 5515
ER_IB_MSG_843 error code, 5515
ER_IB_MSG_844 error code, 5515
ER_IB_MSG_845 error code, 5516
ER_IB_MSG_846 error code, 5516
ER_IB_MSG_847 error code, 5516
ER_IB_MSG_848 error code, 5516
ER_IB_MSG_849 error code, 5516
ER_IB_MSG_85 error code, 5438
ER_IB_MSG_850 error code, 5516
ER_IB_MSG_851 error code, 5516
ER_IB_MSG_852 error code, 5516
ER_IB_MSG_853 error code, 5516
ER_IB_MSG_854 error code, 5516
ER_IB_MSG_855 error code, 5517
ER_IB_MSG_856 error code, 5517
ER_IB_MSG_857 error code, 5517
ER_IB_MSG_858 error code, 5517
ER_IB_MSG_859 error code, 5517
ER_IB_MSG_86 error code, 5438
ER_IB_MSG_860 error code, 5517
ER_IB_MSG_861 error code, 5517
ER_IB_MSG_862 error code, 5517
ER_IB_MSG_863 error code, 5517
ER_IB_MSG_864 error code, 5517
ER_IB_MSG_865 error code, 5518
ER_IB_MSG_866 error code, 5518
ER_IB_MSG_867 error code, 5518
ER_IB_MSG_868 error code, 5518
ER_IB_MSG_869 error code, 5518
ER_IB_MSG_87 error code, 5438
ER_IB_MSG_870 error code, 5518
ER_IB_MSG_871 error code, 5518
ER_IB_MSG_872 error code, 5518
ER_IB_MSG_873 error code, 5518
ER_IB_MSG_874 error code, 5518
ER_IB_MSG_875 error code, 5519
ER_IB_MSG_876 error code, 5519
ER_IB_MSG_877 error code, 5519
ER_IB_MSG_878 error code, 5519
ER_IB_MSG_879 error code, 5519
ER_IB_MSG_88 error code, 5438
ER_IB_MSG_880 error code, 5519
ER_IB_MSG_881 error code, 5519
ER_IB_MSG_882 error code, 5519
ER_IB_MSG_883 error code, 5519
ER_IB_MSG_884 error code, 5519
ER_IB_MSG_885 error code, 5520
ER_IB_MSG_886 error code, 5520
ER_IB_MSG_887 error code, 5520
ER_IB_MSG_888 error code, 5520
ER_IB_MSG_889 error code, 5520

ER_IB_MSG_89 error code, 5438
ER_IB_MSG_890 error code, 5520
ER_IB_MSG_891 error code, 5520
ER_IB_MSG_892 error code, 5520
ER_IB_MSG_893 error code, 5520
ER_IB_MSG_894 error code, 5520
ER_IB_MSG_895 error code, 5521
ER_IB_MSG_896 error code, 5521
ER_IB_MSG_897 error code, 5521
ER_IB_MSG_898 error code, 5521
ER_IB_MSG_899 error code, 5521
ER_IB_MSG_9 error code, 5430
ER_IB_MSG_90 error code, 5438
ER_IB_MSG_900 error code, 5521
ER_IB_MSG_901 error code, 5521
ER_IB_MSG_902 error code, 5521
ER_IB_MSG_903 error code, 5521
ER_IB_MSG_904 error code, 5521
ER_IB_MSG_905 error code, 5522
ER_IB_MSG_906 error code, 5522
ER_IB_MSG_907 error code, 5522
ER_IB_MSG_908 error code, 5522
ER_IB_MSG_909 error code, 5522
ER_IB_MSG_91 error code, 5439
ER_IB_MSG_910 error code, 5522
ER_IB_MSG_911 error code, 5522
ER_IB_MSG_912 error code, 5522
ER_IB_MSG_913 error code, 5522
ER_IB_MSG_914 error code, 5522
ER_IB_MSG_915 error code, 5523
ER_IB_MSG_916 error code, 5523
ER_IB_MSG_917 error code, 5523
ER_IB_MSG_918 error code, 5523
ER_IB_MSG_919 error code, 5523
ER_IB_MSG_92 error code, 5439
ER_IB_MSG_920 error code, 5523
ER_IB_MSG_921 error code, 5523
ER_IB_MSG_922 error code, 5523
ER_IB_MSG_923 error code, 5523
ER_IB_MSG_924 error code, 5523
ER_IB_MSG_925 error code, 5524
ER_IB_MSG_926 error code, 5524
ER_IB_MSG_927 error code, 5524
ER_IB_MSG_928 error code, 5524
ER_IB_MSG_929 error code, 5524
ER_IB_MSG_93 error code, 5439
ER_IB_MSG_930 error code, 5524
ER_IB_MSG_931 error code, 5524
ER_IB_MSG_932 error code, 5524
ER_IB_MSG_933 error code, 5524
ER_IB_MSG_934 error code, 5524
ER_IB_MSG_935 error code, 5525
ER_IB_MSG_936 error code, 5525
ER_IB_MSG_937 error code, 5525

ER_IB_MSG_938 error code, 5525
ER_IB_MSG_939 error code, 5525
ER_IB_MSG_94 error code, 5439
ER_IB_MSG_940 error code, 5525
ER_IB_MSG_941 error code, 5525
ER_IB_MSG_942 error code, 5525
ER_IB_MSG_943 error code, 5525
ER_IB_MSG_944 error code, 5525
ER_IB_MSG_945 error code, 5526
ER_IB_MSG_946 error code, 5526
ER_IB_MSG_947 error code, 5526
ER_IB_MSG_948 error code, 5526
ER_IB_MSG_949 error code, 5526
ER_IB_MSG_95 error code, 5439
ER_IB_MSG_950 error code, 5526
ER_IB_MSG_951 error code, 5526
ER_IB_MSG_952 error code, 5526
ER_IB_MSG_953 error code, 5526
ER_IB_MSG_954 error code, 5526
ER_IB_MSG_955 error code, 5527
ER_IB_MSG_956 error code, 5527
ER_IB_MSG_957 error code, 5527
ER_IB_MSG_958 error code, 5527
ER_IB_MSG_959 error code, 5527
ER_IB_MSG_96 error code, 5439
ER_IB_MSG_960 error code, 5527
ER_IB_MSG_961 error code, 5527
ER_IB_MSG_962 error code, 5527
ER_IB_MSG_963 error code, 5527
ER_IB_MSG_964 error code, 5527
ER_IB_MSG_965 error code, 5528
ER_IB_MSG_966 error code, 5528
ER_IB_MSG_967 error code, 5528
ER_IB_MSG_968 error code, 5528
ER_IB_MSG_969 error code, 5528
ER_IB_MSG_97 error code, 5439
ER_IB_MSG_970 error code, 5528
ER_IB_MSG_971 error code, 5528
ER_IB_MSG_972 error code, 5528
ER_IB_MSG_973 error code, 5528
ER_IB_MSG_974 error code, 5528
ER_IB_MSG_975 error code, 5529
ER_IB_MSG_976 error code, 5529
ER_IB_MSG_977 error code, 5529
ER_IB_MSG_978 error code, 5529
ER_IB_MSG_979 error code, 5529
ER_IB_MSG_98 error code, 5439
ER_IB_MSG_980 error code, 5529
ER_IB_MSG_981 error code, 5529
ER_IB_MSG_982 error code, 5529
ER_IB_MSG_983 error code, 5529
ER_IB_MSG_984 error code, 5529
ER_IB_MSG_985 error code, 5530
ER_IB_MSG_986 error code, 5530

ER_IB_MSG_987 error code, 5530
ER_IB_MSG_988 error code, 5530
ER_IB_MSG_989 error code, 5530
ER_IB_MSG_99 error code, 5439
ER_IB_MSG_990 error code, 5530
ER_IB_MSG_991 error code, 5530
ER_IB_MSG_992 error code, 5530
ER_IB_MSG_993 error code, 5530
ER_IB_MSG_994 error code, 5530
ER_IB_MSG_995 error code, 5531
ER_IB_MSG_996 error code, 5531
ER_IB_MSG_997 error code, 5531
ER_IB_MSG_998 error code, 5531
ER_IB_MSG_999 error code, 5531
ER_IB_MSG_BTREE_LEVEL_LIMIT_EXCEEDED error code, 5597
ER_IB_MSG_CLOCK_GETTIME_FAILED error code, 5594
ER_IB_MSG_CLOCK_MONOTONIC_UNSUPPORTED error code, 5594
ER_IB_MSG_DDL_LOG_DELETE_BY_ID_OK error code, 5496
ER_IB_MSG_DDL_LOG_DELETE_BY_ID_TMCT error code, 5591
ER_IB_MSG_DDL_LOG_FAIL_POST_DDL error code, 5592
ER_IB_MSG_DEPRECATED_INNODB_UNDO_TABLESPACES error code, 5577
ER_IB_MSG_DIR_DOES_NOT_EXIST error code, 5577
ER_IB_MSG_ERROR_OPENING_NEW_UNDO_SPACE error code, 5576
ER_IB_MSG_FAILED_SDI_Z_BUF_ERROR error code, 5576
ER_IB_MSG_FAILED_SDI_Z_MEM_ERROR error code, 5576
ER_IB_MSG_FAILED_TO_ALLOCATE_WAIT error code, 5575
ER_IB_MSG_FAILED_TO_FINISH_TRUNCATE error code, 5577
ER_IB_MSG_FAIL_TO_SAVE_SPACE_STATE error code, 5575
ER_IB_MSG_FOUND_WRONG_UNDO_SPACE error code, 5576
ER_IB_MSG_IGNORE_SCAN_PATH error code, 5467
ER_IB_MSG_LOCK_FREE_HASH_USAGE_STATS error code, 5577
ER_IB_MSG_MADVISE_FAILED error code, 5575
ER_IB_MSG_MADV_DONTDUMP_UNSUPPORTED error code, 5575
ER_IB_MSG_MAX_UNDO_SPACES_REACHED error code, 5576
ER_IB_MSG_NOT_END_WITH_IBU error code, 5576
ER_IB_MSG_NO_ENCRYPT_PROGRESS_FOUND error code, 5569
ER_IB_MSG_NUM_POOLS error code, 5575
ER_IB_MSG_POST_RECOVER_DDL_LOG_RECOVER error code, 5592
ER_IB_MSG_POST_RECOVER_POST_TS_ENCRYPT error code, 5592
ER_IB_MSG_RESUME_OP_FOR_SPACE error code, 5569
ER_IB_MSG_SDI_Z_STREAM_ERROR error code, 5576
ER_IB_MSG_SDI_Z_UNKNOWN_ERROR error code, 5576
ER_IB_MSG_SKIP_HIDDEN_DIR error code, 5603
ER_IB_MSG_TOO_LONG_PATH error code, 5593
ER_IB_MSG_UNDO_TRUNC_BEFORE_UNDO_LOGGING error code, 5577
ER_IB_MSG_UNDO_TRUNC_BEFOR_DD_UPDATE error code, 5577
ER_IB_MSG_UNDO_TRUNC_EMPTY_FILE error code, 5576
ER_IB_MSG_UNDO_TRUNK_BEFORE_RSEG error code, 5577
ER_IB_MSG_UNEXPECTED_FILE_EXISTS error code, 5461
ER_IB_MSG_USING_UNDO_SPACE error code, 5575
ER_IB_MSG_WAIT_FOR_ENCRYPT_THREAD error code, 5569
ER_IB_RECV_FIRST_REC_GROUP_INVALID error code, 5593
ER_IDENT_CAUSES_TOO_LONG_PATH error code, 5155
ER_IDENT_CAUSES_TOO_LONG_PATH_IN_UPGRADE error code, 5600

ER_ILLEGAL_GRANT_FOR_TABLE error code, 5109
ER_ILLEGAL_HA error code, 5100
ER_ILLEGAL_HA_CREATE_OPTION error code, 5130
ER_ILLEGAL_PRIVILEGE_LEVEL error code, 5184
ER_ILLEGAL_REFERENCE error code, 5115
ER_ILLEGAL_USER_VAR error code, 5162
ER_ILLEGAL_VALUE_FOR_TYPE error code, 5123
ER_IMP_INCOMPATIBLE_DD_VERSION error code, 5183
ER_IMP_INCOMPATIBLE_MYSQLD_VERSION error code, 5183
ER_IMP_INCOMPATIBLE_SDI_VERSION error code, 5183
ER_IMP_NO_FILES_MATCHED error code, 5183
ER_IMP_SCHEMA_DOES_NOT_EXIST error code, 5183
ER_IMP_TABLE_ALREADY_EXISTS error code, 5183
ER_INCONSISTENT_ERROR error code, 5561
ER_INCONSISTENT_PARTITION_INFO_ERROR error code, 5130
ER_INCONSISTENT_TYPE_OF_FUNCTIONS_ERROR error code, 5130
ER_INCORRECT_CURRENT_PASSWORD error code, 5215
ER_INCORRECT_GLOBAL_LOCAL_VAR error code, 5114
ER_INCORRECT_TYPE error code, 5162
ER_INDEX_COLUMN_TOO_LONG error code, 5144
ER_INDEX_CORRUPT error code, 5144
ER_INDEX_TYPE_NOT_SUPPORTED_FOR_SPATIAL_INDEX error code, 5196
ER_INIT_BOOTSTRAP_COMPLETE error code, 5270
ER_INIT_CANT_OPEN_BOOTSTRAP_FILE error code, 5270
ER_INIT_CREATING_DD error code, 5270
ER_INIT_DATADIR_EXISTS_AND_NOT_WRITABLE_WONT_INITIALIZE error code, 5270
ER_INIT_DATADIR_EXISTS_AND_PATH_TOO_LONG_WONT_INITIALIZE error code, 5270
ER_INIT_DATADIR_EXISTS_WONT_INITIALIZE error code, 5270
ER_INIT_DATADIR_NOT_EMPTY_WONT_INITIALIZE error code, 5270
ER_INIT_FAILED_TO_GENERATE_ROOT_PASSWORD error code, 5603
ER_INIT_GENERATING_TEMP_PASSWORD_FOR_ROOT error code, 5269
ER_INIT_ROOT_WITHOUT_PASSWORD error code, 5269
ER_INNODB_ACTIVE_INDEX_CHANGE_FAILED error code, 5338
ER_INNODB_CANNOT_BE_IGNORED error code, 5193
ER_INNODB_CANNOT_CREATE_TABLE error code, 5339
ER_INNODB_CANT_BUILD_INDEX_XLATION_TABLE_FOR error code, 5320
ER_INNODB_CANT_FIND_INDEX_IN_INNODB_DD error code, 5319
ER_INNODB_CANT_OPEN_TABLE error code, 5319
ER_INNODB_CLOSING_CONNECTION_ROLLS_BACK error code, 5319
ER_INNODB_CLUSTERED_INDEX_PRIVATE error code, 5320
ER_INNODB_DIFF_IN_REF_LEN error code, 5338
ER_INNODB_DIRTY_WATER_MARK_NOT_LOW error code, 5318
ER_INNODB_ERROR_LOGGER_FATAL_MSG error code, 5341
ER_INNODB_ERROR_LOGGER_MSG error code, 5341
ER_INNODB_FAILED_TO_FIND_IDX error code, 5338
ER_INNODB_FAILED_TO_FIND_IDX_FROM_DICT_CACHE error code, 5338
ER_INNODB_FAILED_TO_FIND_IDX_WITH_KEY_NO error code, 5338
ER_INNODB_FILES_SAME error code, 5319
ER_INNODB_FORCED_RECOVERY error code, 5157
ER_INNODB_FT_AUX_NOT_HEX_ID error code, 5157
ER_INNODB_FT_LIMIT error code, 5150
ER_INNODB_FT_WRONG_DOCID_COLUMN error code, 5151
ER_INNODB_FT_WRONG_DOCID_INDEX error code, 5151
ER_INNODB_IDX_CNT_FEWER_THAN_DEFINED_IN_MYSQL error code, 5339

ER_INNODB_IDX_CNT_MORE_THAN_DEFINED_IN_MYSQL error code, 5339
ER_INNODB_IDX_COLUMN_CNT_DIFF error code, 5339
ER_INNODB_ILLEGAL_COLON_IN_POOL error code, 5318
ER_INNODB_IMPORT_ERROR error code, 5152
ER_INNODB_INDEX_COLUMN_INFO_UNLIKE_MYSQLS error code, 5319
ER_INNODB_INDEX_CORRUPT error code, 5152
ER_INNODB_INTERNAL_INDEX error code, 5339
ER_INNODB_INVALID_INNODB_UNDO_DIRECTORY error code, 5318
ER_INNODB_INVALID_INNODB_UNDO_DIRECTORY_LOCATION error code, 5579
ER_INNODB_INVALID_LOG_GROUP_HOME_DIR error code, 5318
ER_INNODB_INVALID_MONITOR_COUNTER_NAME error code, 5340
ER_INNODB_INVALID_PAGE_SIZE error code, 5318
ER_INNODB_IO_CAPACITY_EXCEEDS_MAX error code, 5319
ER_INNODB_MANDATORY error code, 5228
ER_INNODB_MONITOR_DEFAULT_VALUE_NOT_DEFINED error code, 5339
ER_INNODB_MONITOR_IS_ENABLED error code, 5339
ER_INNODB_NO_FT_TEMP_TABLE error code, 5150
ER_INNODB_ONLINE_LOG_TOO_BIG error code, 5151
ER_INNODB_PARTITION_TABLE_LOWERCASED error code, 5320
ER_INNODB_PK_NOT_IN_MYSQL error code, 5320
ER_INNODB_PK_ONLY_IN_MYSQL error code, 5320
ER_INNODB_READ_ONLY error code, 5156
ER_INNODB_REDO_LOG_ARCHIVE_ACTIVE error code, 5210
ER_INNODB_REDO_LOG_ARCHIVE_DIRS_INVALID error code, 5209
ER_INNODB_REDO_LOG_ARCHIVE_DIR_CLASH error code, 5210
ER_INNODB_REDO_LOG_ARCHIVE_DIR_EMPTY error code, 5209
ER_INNODB_REDO_LOG_ARCHIVE_DIR_PERMISSIONS error code, 5210
ER_INNODB_REDO_LOG_ARCHIVE_FAILED error code, 5210
ER_INNODB_REDO_LOG_ARCHIVE_FILE_CREATE error code, 5210
ER_INNODB_REDO_LOG_ARCHIVE_INACTIVE error code, 5210
ER_INNODB_REDO_LOG_ARCHIVE_LABEL_NOT_FOUND error code, 5209
ER_INNODB_REDO_LOG_ARCHIVE_NO_SUCH_DIR error code, 5210
ER_INNODB_REDO_LOG_ARCHIVE_SESSION error code, 5210
ER_INNODB_REDO_LOG_ARCHIVE_START_SUBDIR_PATH error code, 5209
ER_INNODB_REDO_LOG_ARCHIVE_START_TIMEOUT error code, 5209
ER_INNODB_TRX_XLATION_TABLE_OOM error code, 5319
ER_INNODB_UNDO_LOG_FULL error code, 5159
ER_INNODB_UNKNOWN_COLLATION error code, 5318
ER_INNODB_UNREGISTERED_TRX_ACTIVE error code, 5319
ER_INNODB_USE_MONITOR_GROUP_NAME error code, 5339
ER_INSECURE_CHANGE_MASTER error code, 5148
ER_INSECURE_PLAIN_TEXT error code, 5148
ER_INSERT_INFO error code, 5105
ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_BINLOG_DIRECT error code, 5142
ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_BINLOG_FORMAT error code, 5142
ER_INSIDE_TRANSACTION_PREVENTS_SWITCH_SQL_LOG_BIN error code, 5143
ER_INTERNAL_ERROR error code, 5152
ER_INVALID_ADMIN_ADDRESS error code, 5580
ER_INVALID_ARGUMENT_FOR_LOGARITHM error code, 5159
ER_INVALID_BITWISE_AGGREGATE_OPERANDS_SIZE error code, 5174
ER_INVALID_BITWISE_OPERANDS_SIZE error code, 5174
ER_INVALID_CAST_TO_JSON error code, 5169
ER_INVALID_CHARACTER_STRING error code, 5118
ER_INVALID_CHARSET_AND_DEFAULT_IS_MB error code, 5256

ER_INVALID_COLLATION_FOR_CHARSET error code, 5325
ER_INVALID_DD_OBJECT error code, 5173
ER_INVALID_DD_OBJECT_ID error code, 5174
ER_INVALID_DD_OBJECT_NAME error code, 5174
ER_INVALID_DEFAULT error code, 5103
ER_INVALID_DEFAULT_UTF8MB4_COLLATION error code, 5195
ER_INVALID_ENCRYPTION_OPTION error code, 5171
ER_INVALID_ENCRYPTION_REQUEST error code, 5207
ER_INVALID_ERROR_LOG_NAME error code, 5239
ER_INVALID_FIELD_SIZE error code, 5158
ER_INVALID_GEOJSON_CRS_NOT_TOP_LEVEL error code, 5190
ER_INVALID_GEOJSON_MISSING_MEMBER error code, 5163
ER_INVALID_GEOJSON_UNSPECIFIED error code, 5163
ER_INVALID_GEOJSON_WRONG_TYPE error code, 5163
ER_INVALID_GROUP_FUNC_USE error code, 5106
ER_INVALID_INFO_IN_FRM error code, 5257
ER_INVALID_INSTRUMENT error code, 5228
ER_INVALID_JSON_BINARY_DATA error code, 5168
ER_INVALID_JSON_CHARSET error code, 5169
ER_INVALID_JSON_CHARSET_IN_FUNCTION error code, 5169
ER_INVALID_JSON_DATA error code, 5163
ER_INVALID_JSON_PATH error code, 5168
ER_INVALID_JSON_PATH_ARRAY_CELL error code, 5170
ER_INVALID_JSON_PATH_CHARSET error code, 5169
ER_INVALID_JSON_PATH_WILDCARD error code, 5169
ER_INVALID_JSON_TEXT error code, 5168
ER_INVALID_JSON_TEXT_IN_PARAM error code, 5168
ER_INVALID_JSON_TYPE error code, 5211
ER_INVALID_JSON_VALUE_FOR_CAST error code, 5169
ER_INVALID_JSON_VALUE_FOR_FUNC_INDEX error code, 5216
ER_INVALID_MULTIPLE_CLAUSES error code, 5208
ER_INVALID_NAMED_PIPE_FULL_ACCESS_GROUP error code, 5588
ER_INVALID_NO_OF_ARGS error code, 5182
ER_INVALID_ON_UPDATE error code, 5118
ER_INVALID_OPTION_CHARACTERS error code, 5178
ER_INVALID_OPTION_END_CHARACTER error code, 5178
ER_INVALID_OPTION_KEY error code, 5177
ER_INVALID_OPTION_KEY_VALUE_PAIR error code, 5178
ER_INVALID_OPTION_START_CHARACTER error code, 5178
ER_INVALID_OPTION_VALUE error code, 5177
ER_INVALID_OR_OLD_TABLE_OR_DB_NAME error code, 5277
ER_INVALID_PROPERTY_KEY error code, 5583
ER_INVALID_REPLICATION_TIMESTAMPS error code, 5328
ER_INVALID_RPL_WILD_TABLE_FILTER_PATTERN error code, 5162
ER_INVALID_THREAD_ID error code, 5189
ER_INVALID_THREAD_PRIORITY error code, 5188
ER_INVALID_TYPE_FOR_JSON error code, 5169
ER_INVALID_USE_OF_FORCE_OPTION error code, 5189
ER_INVALID_USE_OF_NULL error code, 5108
ER_INVALID_VALUE_FOR_ENFORCE_GTID_CONSISTENCY error code, 5223
ER_INVALID_VALUE_OF_BIND_ADDRESSES error code, 5570
ER_INVALID_VCPU_ID error code, 5188
ER_INVALID_VCPU_RANGE error code, 5188
ER_INVALID_YEAR_COLUMN_LENGTH error code, 5152

ER_IO_ERR_LOG_INDEX_READ error code, 5123
ER_IO_READ_ERROR error code, 5151
ER_IO_WRITE_ERROR error code, 5152
ER_IPSOCK_ERROR error code, 5104
ER_IS_QUERY_INVALID_CLAUSE error code, 5177
ER_I_S_SKIPPED_TABLESPACE error code, 5186
ER_JSON_BAD_ONE_OR_ALL_ARG error code, 5169
ER_JSON_DOCUMENT_NULL_KEY error code, 5169
ER_JSON_DOCUMENT_TOO_DEEP error code, 5169
ER_JSON_KEY_TOO_BIG error code, 5169
ER_JSON_PARSE_ERROR error code, 5242
ER_JSON_SCHEMA_VALIDATION_ERROR_WITH_DETAILED_REPORT error code, 5219
ER_JSON_USED_AS_KEY error code, 5169
ER_JSON_VACUOUS_PATH error code, 5169
ER_JSON_VALUE_OUT_OF_RANGE_FOR_FUNC_INDEX error code, 5216
ER_JSON_VALUE_TOO_BIG error code, 5169
ER_JT_MAX_NESTED_PATH error code, 5190
ER_JT_VALUE_OUT_OF_RANGE error code, 5189
ER_KEYCACHE_OOM error code, 5231
ER_KEYRING_ACCESS_DENIED_ERROR error code, 5173
ER_KEYRING_AWS_CMK_ID_NOT_SET error code, 5382
ER_KEYRING_AWS_FAILED_TO_ACCESS_DATA_FILE error code, 5381
ER_KEYRING_AWS_FAILED_TO_ACCESS_KEY_FROM_CONF_FILE error code, 5381
ER_KEYRING_AWS_FAILED_TO_ACCESS_KEY_ID_FROM_CONF_FILE error code, 5381
ER_KEYRING_AWS_FAILED_TO_ACCESS_OR_CREATE_KEYRING_DATA_FILE error code, 5381
ER_KEYRING_AWS_FAILED_TO_ACCESS_OR_CREATE_KEYRING_DIR error code, 5381
ER_KEYRING_AWS_FAILED_TO_CONNECT_KMS error code, 5383
ER_KEYRING_AWS_FAILED_TO_DECRYPT_KEY error code, 5383
ER_KEYRING_AWS_FAILED_TO_ENCRYPT_KEY error code, 5383
ER_KEYRING_AWS_FAILED_TO_FLUSH_KEYRING_TO_FILE error code, 5383
ER_KEYRING_AWS_FAILED_TO_GENERATE_KEY_DUE_TO_INTERNAL_ERROR error code, 5382
ER_KEYRING_AWS_FAILED_TO_GENERATE_NEW_KEY error code, 5383
ER_KEYRING_AWS_FAILED_TO_GET_KMS_CREDENTIAL_FROM_CONF_FILE error code, 5382
ER_KEYRING_AWS_FAILED_TO_INIT_DUE_TO_INTERNAL_ERROR error code, 5381
ER_KEYRING_AWS_FAILED_TO_INIT_DUE_TO_PLUGIN_INTERNAL_ERROR error code, 5382
ER_KEYRING_AWS_FAILED_TO_OPEN_CONF_FILE error code, 5380
ER_KEYRING_AWS_FAILED_TO_RESTORE_FROM_BACKUP_FILE error code, 5383
ER_KEYRING_AWS_FAILED_TO_RE_ENCRYPT_KEY error code, 5383
ER_KEYRING_AWS_FAILED_TO_ROTATE_CMK error code, 5383
ER_KEYRING_AWS_FAILED_TO_SET_CMK_ID error code, 5380
ER_KEYRING_AWS_FAILED_TO_SET_REGION error code, 5380
ER_KEYRING_AWS_FOUND_MALFORMED_BACKUP_FILE error code, 5382
ER_KEYRING_AWS_INCORRECT_FILE error code, 5382
ER_KEYRING_AWS_INCORRECT_PROXY error code, 5601
ER_KEYRING_AWS_INCORRECT_REGION error code, 5383
ER_KEYRING_AWS_INIT_FAILURE error code, 5382
ER_KEYRING_AWS_INVALID_CONF_FILE_PATH error code, 5381
ER_KEYRING_AWS_INVALID_DATA_FILE_PATH error code, 5381
ER_KEYRING_AWS_INVALID_KEY_LENGTH_FOR_CIPHER error code, 5382
ER_KEYRING_AWS_UDF_AWS_KMS_ERROR error code, 5172
ER_KEYRING_CHECK_KEY_FAILED_DUE_TO_EMPTY_KEY_ID error code, 5373
ER_KEYRING_CHECK_KEY_FAILED_DUE_TO_INVALID_KEY error code, 5373
ER_KEYRING_ENCRYPTED_FILE_DECRYPTION_FAILED error code, 5379
ER_KEYRING_ENCRYPTED_FILE_ENCRYPTION_FAILED error code, 5379

ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_CREATE_KEYRING_DIR error code, 5379
ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_FLUSH_KEYRING error code, 5379
ER_KEYRING_ENCRYPTED_FILE_FAILED_TO_RESTORE_KEYRING error code, 5379
ER_KEYRING_ENCRYPTED_FILE_FOUND_MALFORMED_BACKUP_FILE error code, 5379
ER_KEYRING_ENCRYPTED_FILE_GEN_KEY_FAILED_DUE_TO_INTERNAL_ERROR error code, 5380
ER_KEYRING_ENCRYPTED_FILE_INCORRECT_KEYRING_FILE error code, 5378
ER_KEYRING_ENCRYPTED_FILE_INIT_FAILED_DUE_TO_INTERNAL_ERROR error code, 5380
ER_KEYRING_ENCRYPTED_FILE_INIT_FAILURE error code, 5380
ER_KEYRING_ENCRYPTED_FILE_INVALID_KEYRING_DIR error code, 5379
ER_KEYRING_ENCRYPTED_FILE_PASSWORD_IS_INVALID error code, 5379
ER_KEYRING_ENCRYPTED_FILE_PASSWORD_IS_TOO_LONG error code, 5380
ER_KEYRING_FAILED_TO_CREATE_KEYRING_DIR error code, 5373
ER_KEYRING_FAILED_TO_FLUSH_KEYRING_TO_FILE error code, 5374
ER_KEYRING_FAILED_TO_FLUSH_KEYS_TO_KEYRING error code, 5375
ER_KEYRING_FAILED_TO_FLUSH_KEYS_TO_KEYRING_BACKUP error code, 5375
ER_KEYRING_FAILED_TO_GENERATE_KEY error code, 5373
ER_KEYRING_FAILED_TO_GET_FILE_STAT error code, 5374
ER_KEYRING_FAILED_TO_LOAD_KEYRING_CONTENT error code, 5375
ER_KEYRING_FAILED_TO_REMOVE_FILE error code, 5374
ER_KEYRING_FAILED_TO_REMOVE_KEY_DUE_TO_EMPTY_ID error code, 5375
ER_KEYRING_FAILED_TO_RESTORE_FROM_BACKUP_FILE error code, 5374
ER_KEYRING_FAILED_TO_SET_KEYRING_FILE_DATA error code, 5374
ER_KEYRING_FAILED_TO_TRUNCATE_FILE error code, 5374
ER_KEYRING_FILE_INIT_FAILED error code, 5373
ER_KEYRING_FILE_IO_ERROR error code, 5375
ER_KEYRING_FOUND_MALFORMED_BACKUP_FILE error code, 5374
ER_KEYRING_INCORRECT_FILE error code, 5374
ER_KEYRING_INTERNAL_EXCEPTION_FAILED_FILE_INIT error code, 5373
ER_KEYRING_INVALID_KEY_LENGTH error code, 5373
ER_KEYRING_INVALID_KEY_TYPE error code, 5372
ER_KEYRING_KEY_FETCH_FAILED_DUE_TO_EMPTY_KEY_ID error code, 5375
ER_KEYRING_LOGGER_ERROR_MSG error code, 5355
ER_KEYRING_MIGRATE_FAILED error code, 5560
ER_KEYRING_MIGRATION_EXTRA_OPTIONS error code, 5429
ER_KEYRING_MIGRATION_FAILED error code, 5343
ER_KEYRING_MIGRATION_FAILURE error code, 5173
ER_KEYRING_MIGRATION_STATUS error code, 5173
ER_KEYRING_MIGRATION_SUCCESSFUL error code, 5343
ER_KEYRING_OKV_CONNECTION_TO_SERVER_FAILED error code, 5377
ER_KEYRING_OKV_FAILED_TO_ACTIVATE_KEYS error code, 5377
ER_KEYRING_OKV_FAILED_TO_ADD_ATTRIBUTE error code, 5377
ER_KEYRING_OKV_FAILED_TO_FETCH_KEY error code, 5378
ER_KEYRING_OKV_FAILED_TO_FIND_SERVER_ENTRY error code, 5376
ER_KEYRING_OKV_FAILED_TO_FIND_STANDBY_SERVER_ENTRY error code, 5376
ER_KEYRING_OKV_FAILED_TO_GENERATE_KEY error code, 5377
ER_KEYRING_OKV_FAILED_TO_GENERATE_KEY_DUE_TO_INTERNAL_ERROR error code, 5376
ER_KEYRING_OKV_FAILED_TO_INIT_CLIENT error code, 5377
ER_KEYRING_OKV_FAILED_TO_INIT_SSL_LAYER error code, 5377
ER_KEYRING_OKV_FAILED_TO_LOAD_KEY_UID error code, 5376
ER_KEYRING_OKV_FAILED_TO_LOAD_SSL_TRUST_STORE error code, 5378
ER_KEYRING_OKV_FAILED_TO_PARSE_CONF_FILE error code, 5376
ER_KEYRING_OKV_FAILED_TO_REMOVE_KEY error code, 5377
ER_KEYRING_OKV_FAILED_TO_RETRIEVE_KEY error code, 5378
ER_KEYRING_OKV_FAILED_TO_RETRIEVE_KEY_SIGNATURE error code, 5378

ER_KEYRING_OKV_FAILED_TO_SET_CERTIFICATE_FILE error code, 5378
ER_KEYRING_OKV_FAILED_TO_SET_KEY_FILE error code, 5378
ER_KEYRING_OKV_FAILED_TO_STORE_KEY error code, 5377
ER_KEYRING_OKV_FAILED_TO_STORE_OR_GENERATE_KEY error code, 5378
ER_KEYRING_OKV_INCORRECT_KEY_VAULT_CONFIGURED error code, 5375
ER_KEYRING_OKV_INIT_FAILED_DUE_TO_INCORRECT_CONF error code, 5375
ER_KEYRING_OKV_INIT_FAILED_DUE_TO_INTERNAL_ERROR error code, 5376
ER_KEYRING_OKV_INVALID_KEY_LENGTH_FOR_CIPHER error code, 5376
ER_KEYRING_OKV_INVALID_KEY_TYPE error code, 5376
ER_KEYRING_OKV_KEY_MISMATCH error code, 5378
ER_KEYRING_OPERATION_FAILED_DUE_TO_INTERNAL_ERROR error code, 5373
ER_KEYRING_UDF_KEYRING_SERVICE_ERROR error code, 5172
ER_KEYRING_UNKNOWN_ERROR error code, 5374
ER_KEY_COLUMN_DOES_NOT_EXITS error code, 5103
ER_KEY_DOES_NOT_EXITS error code, 5110
ER_KEY_NOT_FOUND error code, 5100
ER_KEY_PART_0 error code, 5124
ER_KEY_REF_DO_NOT_MATCH_TABLE_REF error code, 5115
ER_KILLED_THREADS_OF_PLUGIN error code, 5272
ER_KILLING_THREAD error code, 5271
ER_KILL_DENIED_ERROR error code, 5105
ER_LATITUDE_OUT_OF_RANGE error code, 5184
ER_LCTN_CHANGED error code, 5343
ER_LCTN_NOT_FOUND error code, 5560
ER_LDAP_AUTH_CERTIFICATE_NAME error code, 5422
ER_LDAP_AUTH_COMMUNICATION_HOST_INFO error code, 5425
ER_LDAP_AUTH_CONNECTION_CREATOR_ENTER error code, 5425
ER_LDAP_AUTH_CONNECTION_GET_LDAP_INFO_NULL error code, 5425
ER_LDAP_AUTH_CONNECTION_POOL_INIT_FAILED error code, 5423
ER_LDAP_AUTH_CONNECTION_POOL_REINIT_ENTER error code, 5427
ER_LDAP_AUTH_CONNECTION_PUSHED_TO_POOL error code, 5425
ER_LDAP_AUTH_CONN_POOL_DEINITIALIZING error code, 5421
ER_LDAP_AUTH_CONN_POOL_INITIALIZING error code, 5421
ER_LDAP_AUTH_CONN_POOL_NOT_CREATED error code, 5421
ER_LDAP_AUTH_CREATE_CONNECTION_KEY error code, 5425
ER_LDAP_AUTH_CREATING_LDAP_CONNECTION error code, 5424
ER_LDAP_AUTH_DEINIT_FAILED error code, 5420
ER_LDAP_AUTH_DELETING_CONNECTION_KEY error code, 5425
ER_LDAP_AUTH_DISTINGUISHED_NAME error code, 5426
ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_CONNECTION error code, 5427
ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_OBJECT error code, 5420
ER_LDAP_AUTH_FAILED_TO_CREATE_LDAP_OBJECT_CREATOR error code, 5420
ER_LDAP_AUTH_FAILED_TO_CREATE_OR_GET_CONNECTION error code, 5420
ER_LDAP_AUTH_FAILED_TO_DEINITIALIZE_NOT_READY_POOL error code, 5423
ER_LDAP_AUTH_FAILED_TO_DEINITIALIZE_POOL_IN_RECONSTRUCT_STATE error code, 5423
ER_LDAP_AUTH_FAILED_TO_ESTABLISH_TLS_CONNECTION error code, 5427
ER_LDAP_AUTH_FAILED_TO_GET_CONNECTION_AS_PLUGIN_NOT_READY error code, 5423
ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_DEINIT_STATE error code, 5423
ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_INIT_STATE error code, 5423
ER_LDAP_AUTH_FAILED_TO_INITIALIZE_POOL_IN_RECONSTRUCTING error code, 5423
ER_LDAP_AUTH_FAILED_TO_POOL_DEINIT error code, 5423
ER_LDAP_AUTH_FAILED_TO_SEARCH_DN error code, 5427
ER_LDAP_AUTH_FAILED_TO_WRITE_PACKET error code, 5421
ER_LDAP_AUTH_FREEING_CONNECTION error code, 5425

ER_LDAP_AUTH_GETTING_CONNECTION_FROM_POOL error code, 5424
ER_LDAP_AUTH_GRP_DN_PARSING_FAILED error code, 5585
ER_LDAP_AUTH_GRP_INCORRECT_ATTRIBUTE error code, 5585
ER_LDAP_AUTH_GRP_INFO_FOUND_IN_MANY_OBJECTS error code, 5585
ER_LDAP_AUTH_GRP_IS_FULL_DN error code, 5422
ER_LDAP_AUTH_GRP_NULL_ATTRIBUTE_VALUE error code, 5585
ER_LDAP_AUTH_GRP_OBJECT_HAS_USER_INFO error code, 5585
ER_LDAP_AUTH_GRP_SEARCH_NOT_SPECIAL_HDL error code, 5585
ER_LDAP_AUTH_GRP_SEARCH_SPECIAL_HDL error code, 5422
ER_LDAP_AUTH_GRP_USER_OBJECT_HAS_GROUP_INFO error code, 5585
ER_LDAP_AUTH_INFO_FOR_USER error code, 5421
ER_LDAP_AUTH_INIT_FAILED error code, 5426
ER_LDAP_AUTH_LDAPS error code, 5586
ER_LDAP_AUTH_LDAP_INFO_NULL error code, 5424
ER_LDAP_AUTH_MAX_ALLOWED_CONNECTION_LIMIT_HIT error code, 5424
ER_LDAP_AUTH_MAX_POOL_SIZE_SET_FAILED error code, 5424
ER_LDAP_AUTH_METHOD_TO_CLIENT error code, 5426
ER_LDAP_AUTH_OBJECT_CREATE_TIMESTAMP error code, 5422
ER_LDAP_AUTH_OR_GROUP_RETRIEVAL_FAILED error code, 5426
ER_LDAP_AUTH_PLUGIN_FAILED_TO_READ_PACKET error code, 5424
ER_LDAP_AUTH_POOLED_CONNECTION_KEY error code, 5425
ER_LDAP_AUTH_POOL_DISABLE_MAX_SIZE_ZERO error code, 5420
ER_LDAP_AUTH_POOL_GET_FAILED_TO_CREATE_CONNECTION error code, 5427
ER_LDAP_AUTH_POOL_REINITIALIZING error code, 5421
ER_LDAP_AUTH_RETURNING_CONNECTION_TO_POOL error code, 5424
ER_LDAP_AUTH_SASL_BIND_SUCCESS_INFO error code, 5426
ER_LDAP_AUTH_SASL_PROCESS_SASL error code, 5426
ER_LDAP_AUTH_SASL_REQUEST_FROM_CLIENT error code, 5426
ER_LDAP_AUTH_SEARCHED_USER_GRP_NAME error code, 5422
ER_LDAP_AUTH_SEARCH_USER_GROUP_ATTR_NOT_FOUND error code, 5424
ER_LDAP_AUTH_SETTING_USERNAME error code, 5421
ER_LDAP_AUTH_SKIPPING_USER_GROUP_SEARCH error code, 5420
ER_LDAP_AUTH_STARTED_FOR_USER error code, 5426
ER_LDAP_AUTH_STARTING_TLS error code, 5425
ER_LDAP_AUTH_TLS_CONF error code, 5420
ER_LDAP_AUTH_TLS_CONNECTION error code, 5420
ER_LDAP_AUTH_USER_AUTH_DATA error code, 5421
ER_LDAP_AUTH_USER_BIND_FAILED error code, 5427
ER_LDAP_AUTH_USER_FOUND_IN_MANY_GRPS error code, 5422
ER_LDAP_AUTH_USER_GROUP_SEARCH_FAILED error code, 5426
ER_LDAP_AUTH_USER_GROUP_SEARCH_INFO error code, 5422
ER_LDAP_AUTH_USER_GROUP_SEARCH_ROOT_BIND error code, 5574
ER_LDAP_AUTH_USER_HAS_MULTIPLE_GRP_NAMES error code, 5422
ER_LDAP_AUTH_USER_NOT_FOUND_IN_ANY_GRP error code, 5422
ER_LDAP_AUTH_ZERO_MAX_POOL_SIZE_UNCHANGED error code, 5421
ER_LDAP_EMPTY_USERDN_PASSWORD error code, 5596
ER_LDAP_MAPPING_CHECK_DELIMI_QUOTE error code, 5586
ER_LDAP_MAPPING_EMPTY_MAPPING error code, 5586
ER_LDAP_MAPPING_GETTING_NEXT_MAPPING error code, 5587
ER_LDAP_MAPPING_GET_USER_PROXY error code, 5586
ER_LDAP_MAPPING_INFO error code, 5586
ER_LDAP_MAPPING_IS_QUOTE error code, 5587
ER_LDAP_MAPPING_NON_DESIRED_STATE error code, 5588
ER_LDAP_MAPPING_NO_SEPEARATOR_END_OF_GROUP error code, 5587

ER_LDAP_MAPPING_PARSING_CURRENT_STATE error code, 5587
ER_LDAP_MAPPING_PARSING_ERROR error code, 5587
ER_LDAP_MAPPING_PARSING_MAPPING_INFO error code, 5587
ER_LDAP_MAPPING_PROCESS_DELIMITER error code, 5586
ER_LDAP_MAPPING_PROCESS_DELIMITER_COMMA_NOT_FOUND error code, 5587
ER_LDAP_MAPPING_PROCESS_DELIMITER_EQUAL_NOT_FOUND error code, 5586
ER_LDAP_MAPPING_PROCESS_DELIMITER_TRY_COMMA error code, 5587
ER_LDAP_MAPPING_PROCESS_MAPPING error code, 5586
ER_LDAP_MAPPING_TRIMMING_SPACES error code, 5587
ER_LDAP_MAPPING_USER_DONT_BELONG_GROUP error code, 5586
ER_LIMITED_PART_RANGE error code, 5132
ER_LOAD_DATA_INFILE_FAILED_IN_UNEXPECTED_WAY error code, 5243
ER_LOAD_FROM_FIXED_SIZE_ROWS_TO_VAR error code, 5125
ER_LOAD_INFO error code, 5104
ER_LOCAL_VARIABLE error code, 5114
ER_LOCKING_SERVICE_DEADLOCK error code, 5168
ER_LOCKING_SERVICE_TIMEOUT error code, 5168
ER_LOCKING_SERVICE_WRONG_NAME error code, 5167
ER_LOCK_ABORTED error code, 5143
ER_LOCK_DEADLOCK error code, 5113
ER_LOCK_NOWAIT error code, 5179
ER_LOCK_ORDER_DEPENDENCIES_SYNTAX error code, 5597
ER_LOCK_ORDER_FAILED_READ_FILE error code, 5597
ER_LOCK_ORDER_FAILED_WRITE_FILE error code, 5597
ER_LOCK_ORDER_INIT_FAILED error code, 5595
ER_LOCK_ORDER_MESSAGE error code, 5597
ER_LOCK_ORDER_SCANNER_SYNTAX error code, 5597
ER_LOCK_OR_ACTIVE_TRANSACTION error code, 5111
ER_LOCK_REFUSED_BY_ENGINE error code, 5171
ER_LOCK_TABLE_FULL error code, 5112
ER_LOCK_WAIT_TIMEOUT error code, 5112
ER_LOG_BIN_BETTER_WITH_NAME error code, 5237
ER_LOG_CANNOT_WRITE error code, 5221
ER_LOG_CANNOT_WRITE_EXTENDED error code, 5600
ER_LOG_FILES_GIVEN_LOG_OUTPUT_IS_TABLE error code, 5236
ER_LOG_FILE_CANNOT_OPEN error code, 5363
ER_LOG_FILE_INVALID error code, 5236
ER_LOG_GENERAL_CANNOT_OPEN error code, 5221
ER_LOG_IN_USE error code, 5123
ER_LOG_OUTPUT_CONTRADICTORY error code, 5235
ER_LOG_PRINTF_MSG error code, 5341
ER_LOG_PURGE_NO_FILE error code, 5137
ER_LOG_PURGE_UNKNOWN_ERR error code, 5124
ER_LOG_SLOW_CANNOT_OPEN error code, 5221
ER_LOG_SYSLOG_CANNOT_OPEN error code, 5221
ER_LOG_SYSLOG_FACILITY_FAIL error code, 5221
ER_LONGITUDE_OUT_OF_RANGE error code, 5184
ER_LOWER_CASE_TABLE_NAMES_CS_DD_ON_CI_FS_UNSUPPORTED error code, 5236
ER_LOWER_CASE_TABLE_NAMES_USING_0 error code, 5236
ER_LOWER_CASE_TABLE_NAMES_USING_2 error code, 5236
ER_MALFORMED_GTID_SET_ENCODING error code, 5149
ER_MALFORMED_GTID_SET_SPECIFICATION error code, 5148
ER_MALFORMED_GTID_SPECIFICATION error code, 5149
ER_MALFORMED_PACKET error code, 5153

ER_MANDATORY_ROLE error code, 5185
ER_MASTER error code, 5111
ER_MASTER_DELAY_VALUE_OUT_OF_RANGE error code, 5146
ER_MASTER_FATAL_ERROR_READING_BINLOG error code, 5114
ER_MASTER_HAS_PURGED_REQUIRED_GTIDS error code, 5150
ER_MASTER_INFO error code, 5112
ER_MASTER_KEY_ROTATION_BINLOG_FAILED error code, 5171
ER_MASTER_KEY_ROTATION_NOT_SUPPORTED_BY_SE error code, 5171
ER_MASTER_KEY_ROTATION_SE_UNAVAILABLE error code, 5171
ER_MASTER_NET_READ error code, 5111
ER_MASTER_NET_WRITE error code, 5111
ER_MAXVALUE_IN_VALUES_IN error code, 5140
ER_MAX_PREPARED_STMT_COUNT_REACHED error code, 5129
ER_MECAB_CHARSET_LOADED error code, 5347
ER_MECAB_CREATE_LATTICE_FAILED error code, 5347
ER_MECAB_CREATING_MODEL error code, 5347
ER_MECAB_FAILED_TO_CREATE_MODEL error code, 5347
ER_MECAB_FAILED_TO_CREATE_TRIGGER error code, 5347
ER_MECAB_NOT_SUPPORTED error code, 5346
ER_MECAB_NOT_VERIFIED error code, 5347
ER_MECAB_OOM_WHILE_PARSING_TEXT error code, 5347
ER_MECAB_PARSE_FAILED error code, 5347
ER_MECAB_UNSUPPORTED_CHARSET error code, 5347
ER_MESSAGE_AND_STATEMENT error code, 5142
ER_MICROSECOND_TIMER_IS_NOT_AVAILABLE error code, 5338
ER_MISSING_ACL_SYSTEM_TABLE error code, 5329
ER_MISSING_CURRENT_PASSWORD error code, 5215
ER_MISSING_GRANT_SYSTEM_TABLE error code, 5329
ER_MISSING_HA_CREATE_OPTION error code, 5159
ER_MISSING_JSON_TABLE_VALUE error code, 5189
ER_MISSING_KEY error code, 5157
ER_MISSING_SKIP_SLAVE error code, 5117
ER_MISSING_TABLESPACE_FILE error code, 5185
ER_MIXING_NOT_ALLOWED error code, 5114
ER_MIX_HANDLER_ERROR error code, 5131
ER_MIX_OF_GROUP_FUNC_AND_FIELDS error code, 5108
ER_MIX_OF_GROUP_FUNC_AND_FIELDS_V2 error code, 5164
ER_MTS_CANT_PARALLEL error code, 5147
ER_MTS_CHANGE_MASTER_CANT_RUN_WITH_GAPS error code, 5151
ER_MTS_EVENT_BIGGER_PENDING_JOBS_SIZE_MAX error code, 5155
ER_MTS_FEATURE_IS_NOT_SUPPORTED error code, 5147
ER_MTS_INCONSISTENT_DATA error code, 5147
ER_MTS_RECOVERY_FAILURE error code, 5151
ER_MTS_RESET_WORKERS error code, 5151
ER_MTS_UPDATED_DBS_GREATER_MAX error code, 5147
ER_MULTIPLE_DEF_CONST_IN_LIST_PART_ERROR error code, 5131
ER_MULTIPLE_PRI_KEY error code, 5103
ER_MULTI_UPDATE_KEY_CONFLICT error code, 5144
ER_MUST_CHANGE_EXPIRED_PASSWORD error code, 5325
ER_MUST_CHANGE_PASSWORD error code, 5152
ER_MUST_CHANGE_PASSWORD_LOGIN error code, 5155
ER_MYINIT_FAILED error code, 5234
ER_MYISAM_CHECK_METHOD_ERROR error code, 5340
ER_MYISAM_CRASHED_ERROR error code, 5340

ER_MYISAM_CRASHED_ERROR_IN error code, 5245
ER_MYISAM_CRASHED_ERROR_IN_THREAD error code, 5245
ER_MYSQLBACKUP_MSG error code, 5602
ER_MYSQL_NATIVE_PASSWORD_SECOND_PASSWORD_USED_INFORMATION error code, 5581
ER_MY_NET_WRITE_FAILED_FALLING_BACK_ON_STDERR error code, 5304
ER_M_BIGGER_THAN_D error code, 5126
ER_NAME_BECOMES_EMPTY error code, 5129
ER_NANOSECOND_TIMER_IS_NOT_AVAILABLE error code, 5337
ER_NATIVE_FCT_NAME_COLLISION error code, 5136
ER_NDB_BINLOG_BLOB_REQUIRES_PK error code, 5295
ER_NDB_BINLOG_CANT_COMMIT_TO_NDB_BINLOG_INDEX error code, 5294
ER_NDB_BINLOG_CANT_CREATE_BLOB error code, 5296
ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB error code, 5295
ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB_AND_CANT_DROP error code, 5296
ER_NDB_BINLOG_CANT_CREATE_EVENT_IN_DB_DROPPED error code, 5296
ER_NDB_BINLOG_CANT_CREATE_PURGE_THD error code, 5318
ER_NDB_BINLOG_CANT_DISCOVER_TABLE_FROM_SCHEMA_EVENT error code, 5290
ER_NDB_BINLOG_CANT_DROP_EVENT_FROM_DB error code, 5296
ER_NDB_BINLOG_CANT_INJECT_APPLY_STATUS_WRITE_ROW error code, 5299
ER_NDB_BINLOG_CANT_LOCK_NDB_BINLOG_INDEX error code, 5293
ER_NDB_BINLOG_CANT_RELEASE_SLOCK error code, 5291
ER_NDB_BINLOG_CANT_REOPEN_SHADOW_TABLE error code, 5292
ER_NDB_BINLOG_CANT_WRITE_TO_NDB_BINLOG_INDEX error code, 5293
ER_NDB_BINLOG_CLEANING_UP_SETUP_LEFTOVERS error code, 5289
ER_NDB_BINLOG_CLUSTER_FAILURE error code, 5297
ER_NDB_BINLOG_CLUSTER_HAS_RECONNECTED error code, 5298
ER_NDB_BINLOG_CLUSTER_RESTARTED_RESET_MASTER_SUGGESTED error code, 5298
ER_NDB_BINLOG_CREATE_TABLE_EVENT error code, 5274
ER_NDB_BINLOG_CREATE_TABLE_EVENT_FAILED error code, 5295
ER_NDB_BINLOG_CREATE_TABLE_EVENT_INFO error code, 5295
ER_NDB_BINLOG_CREATING_NDBEVENTOPERATION_FAILED error code, 5296
ER_NDB_BINLOG_DISCOVER_REUSING_OLD_EVENT_OPS error code, 5296
ER_NDB_BINLOG_DISCOVER_TABLE_EVENT_INFO error code, 5295
ER_NDB_BINLOG_ERROR_DURING_GCI_COMMIT error code, 5299
ER_NDB_BINLOG_ERROR_DURING_GCI_ROLLBACK error code, 5299
ER_NDB_BINLOG_ERROR_HANDLING_SCHEMA_EVENT error code, 5299
ER_NDB_BINLOG_ERROR_INFO_FROM_DA error code, 5274
ER_NDB_BINLOG_FAILED_CREATE_EVENT_OPERATIONS_DURING_RENAME error code, 5274
ER_NDB_BINLOG_FAILED_CREATE_TABLE_EVENT_OPERATIONS error code, 5274
ER_NDB_BINLOG_FAILED_TO_GET_TABLE error code, 5295
ER_NDB_BINLOG_FORMAT_CHANGED_FROM_STMT_TO_MIXED error code, 5275
ER_NDB_BINLOG_GENERIC_MESSAGE error code, 5294
ER_NDB_BINLOG_GOT_DIST_PRIV_EVENT_FLUSHING_PRIVILEGES error code, 5292
ER_NDB_BINLOG_GOT_SCHEMA_EVENT error code, 5292
ER_NDB_BINLOG_INJECTING_RANDOM_WRITE_FAILURE error code, 5293
ER_NDB_BINLOG_INJECTOR_DISCARDING_ROW_EVENT_METADATA error code, 5297
ER_NDB_BINLOG_LATEST_TRX_IN_EPOCH_NOT_IN_BINLOG error code, 5299
ER_NDB_BINLOG_LOST_SCHEMA_CONNECTION_CONTINUING error code, 5298
ER_NDB_BINLOG_LOST_SCHEMA_CONNECTION_WAITING error code, 5298
ER_NDB_BINLOG_NDBEVENT_EXECUTE_FAILED error code, 5296
ER_NDB_BINLOG_NDB_LOG_APPLY_STATUS_FORCING_FULL_USE_WRITE error code, 5294
ER_NDB_BINLOG_NDB_LOG_TRANSACTION_ID_REQUIRES_V2_ROW_EVENTS error code, 5294
ER_NDB_BINLOG_NDB_TABLES_INITIALLY_READ_ONLY error code, 5276
ER_NDB_BINLOG_NDB_TABLES_WRITABLE error code, 5298

ER_NDB_BINLOG_NOT_LOGGING error code, 5295
ER_NDB_BINLOG_ONLINE_ALTER_RENAME error code, 5291
ER_NDB_BINLOG_ONLINE_ALTER_RENAME_COMPLETE error code, 5292
ER_NDB_BINLOG_OPENING_INDEX error code, 5293
ER_NDB_BINLOG_RELEASING_EXTRA_SHARE_REFERENCES error code, 5299
ER_NDB_BINLOG_REMAINING_OPEN_TABLES error code, 5299
ER_NDB_BINLOG_REMAINING_OPEN_TABLE_INFO error code, 5299
ER_NDB_BINLOG_RENAME_EVENT error code, 5274
ER_NDB_BINLOG_REPLY_TO error code, 5291
ER_NDB_BINLOG_SERVER_SHUTDOWN_DURING_NDB_CLUSTER_START error code, 5298
ER_NDB_BINLOG_SHUTDOWN_DETECTED error code, 5298
ER_NDB_BINLOG_SIGNALLING_UNKNOWN_VALUE error code, 5291
ER_NDB_BINLOG_SKIPPING_DROP_OF_DB_CONTAINING_LOCAL_TABLES error code, 5292
ER_NDB_BINLOG_SKIPPING_DROP_OF_LOCAL_TABLE error code, 5292
ER_NDB_BINLOG_SKIPPING_LOCAL_TABLE error code, 5291
ER_NDB_BINLOG_SKIPPING_OLD_SCHEMA_OPERATION error code, 5292
ER_NDB_BINLOG_SKIPPING_RENAME_OF_LOCAL_TABLE error code, 5292
ER_NDB_BINLOG_STARTING_LOG_AT_EPOCH error code, 5298
ER_NDB_BINLOG_UNHANDLED_ERROR_FOR_TABLE error code, 5297
ER_NDB_BINLOG_UNKNOWN_NON_DATA_EVENT error code, 5297
ER_NDB_BINLOG_USING_SERVER_ID_0_SLAVES_WILL_NOT error code, 5294
ER_NDB_BINLOG_WRITE_TO_NDB_BINLOG_INDEX_FAILED_AFTER_KILL error code, 5294
ER_NDB_BINLOG_WRITING_TO_NDB_BINLOG_INDEX error code, 5293
ER_NDB_CANT_ALLOC_GLOBAL_NDB_CLUSTER_CONNECTION error code, 5309
ER_NDB_CANT_ALLOC_GLOBAL_NDB_OBJECT error code, 5309
ER_NDB_CANT_ALLOC_NDB_CLUSTER_CONNECTION error code, 5309
ER_NDB_CANT_FIND_TABLE error code, 5291
ER_NDB_CANT_PARSE_NDB_CLUSTER_CONNECTION_POOL_NODEIDS error code, 5308
ER_NDB_CANT_START_CONNECT_THREAD error code, 5310
ER_NDB_CLEANING_STRAY_TABLES error code, 5288
ER_NDB_CLEAR_SLOCK_INFO error code, 5291
ER_NDB_CLUSTER_FAILURE error code, 5293
ER_NDB_CLUSTER_FIND_ALL_DBS_FAIL error code, 5289
ER_NDB_CLUSTER_FIND_ALL_DBS_RETRY error code, 5289
ER_NDB_CLUSTER_FREE_SHARE_INFO error code, 5276
ER_NDB_CLUSTER_GENERIC_MESSAGE error code, 5264
ER_NDB_CLUSTER_GET_SHARE_INFO error code, 5275
ER_NDB_CLUSTER_MARK_SHARE_DROPPED_DESTROYING_SHARE error code, 5276
ER_NDB_CLUSTER_MARK_SHARE_DROPPED_INFO error code, 5276
ER_NDB_CLUSTER_OOM_THD_NDB error code, 5276
ER_NDB_CLUSTER_REAL_FREE_SHARE_DROP_FAILED error code, 5276
ER_NDB_CLUSTER_REAL_FREE_SHARE_INFO error code, 5276
ER_NDB_CLUSTER_SCHEMA_INFO error code, 5263
ER_NDB_CLUSTER_WRONG_NUMBER_OF_FUNCTION_ARGUMENTS error code, 5263
ER_NDB_COLUMN_DEFAULTS_DIFFER error code, 5272
ER_NDB_COLUMN_INFO error code, 5273
ER_NDB_COLUMN_SHOULD_NOT_HAVE_NATIVE_DEFAULT error code, 5272
ER_NDB_CONFLICT_FN_PARSE_ERROR error code, 5295
ER_NDB_CONFLICT_FN_SETUP_ERROR error code, 5295
ER_NDB_CONFLICT_GENERIC_MESSAGE error code, 5294
ER_NDB_COULD_NOT_GET_APPLY_STATUS_SHARE error code, 5297
ER_NDB_CPU_MASK_TOO_SHORT error code, 5310
ER_NDB_CREATE_EVENT_OPS_LOGGING_INFO error code, 5296
ER_NDB_CREATING_SHARE_IN_OPEN error code, 5275

ER_NDB_CREATING_TABLE error code, 5288
ER_NDB_DISCARDING_EVENT_ID_VERSION_MISMATCH error code, 5291
ER_NDB_DISCARDING_EVENT_NO_OBJ error code, 5291
ER_NDB_DISCONNECT_INFO error code, 5272
ER_NDB_DISCOVERED_MISSING_DB error code, 5288
ER_NDB_DISCOVERED_REMAINING_DB error code, 5289
ER_NDB_DISTRIBUTED_INFO error code, 5290
ER_NDB_DISTRIBUTING_ERR error code, 5321
ER_NDB_DISTRIBUTION_COMPLETE error code, 5290
ER_NDB_DUPLICATE_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS error code, 5309
ER_NDB_EMPTY_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS error code, 5308
ER_NDB_ERROR_IN_GET_AUTO_INCREMENT error code, 5275
ER_NDB_ERROR_IN_READAUTOINCREMENTVALUE error code, 5273
ER_NDB_FAILED_TO_SET_UP_TABLE error code, 5289
ER_NDB_FIELD_INFO error code, 5272
ER_NDB_FLUSHING_TABLE_INFO error code, 5288
ER_NDB_FOUND_UNCOMMITTED_AUTOCOMMIT error code, 5273
ER_NDB_GENERIC_ERROR error code, 5310
ER_NDB_HANDLE_TRAILING_SHARE_INFO error code, 5275
ER_NDB_IGNORING_UNKNOWN_EVENT error code, 5293
ER_NDB_ILLEGAL_VALUE_FOR_NDB_RECV_THREAD_CPU_MASK error code, 5276
ER_NDB_INFO_FAILED_TO_CREATE_NDBINFO error code, 5263
ER_NDB_INFO_FAILED_TO_INIT_NDBINFO error code, 5263
ER_NDB_INFO_FOUND_UNEXPECTED_FIELD_TYPE error code, 5263
ER_NDB_INITIALIZE_GIVEN_CLUSTER_PLUGIN_DISABLED error code, 5275
ER_NDB_INVALID_NODEID_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS error code, 5308
ER_NDB_LOG_ENTRY error code, 5318
ER_NDB_LOG_ENTRY_WITH_PREFIX error code, 5318
ER_NDB_MISMATCH_IN_FRM_DISCOVERING error code, 5289
ER_NDB_MISSING_FRM_DISCOVERING error code, 5289
ER_NDB_NODEID_NOT_FIRST_IN_NDB_CLUSTER_CONNECTION_POOL_NODEIDS error code, 5309
ER_NDB_NODE_ID_AND_MANAGEMENT_SERVER_INFO error code, 5272
ER_NDB_NODE_INFO error code, 5310
ER_NDB_NOT_WAITING_FOR_DISTRIBUTING error code, 5290
ER_NDB_NUMBER_OF_CHANNELS error code, 5321
ER_NDB_OOM_GET_NDB_BLOBS_VALUE error code, 5288
ER_NDB_OOM_IN_FIX_UNIQUE_INDEX_ATTR_ORDER error code, 5273
ER_NDB_POOL_SIZE_MUST_MATCH_NDB_CLUSTER_CONNECTION_POOL_NODEIDS error code, 5309
ER_NDB_QUERY_FAILED error code, 5271
ER_NDB_REMAINING_OPEN_TABLES error code, 5297
ER_NDB_REMAINING_OPEN_TABLE_INFO error code, 5297
ER_NDB_REPLICATION_SCHEMA_ERROR error code, 5138
ER_NDB_SCHEMA_DISTRIBUTION_FAILED error code, 5290
ER_NDB_SCHEMA_DISTRIBUTION_REPORTS_SUBSCRIBE error code, 5290
ER_NDB_SCHEMA_DISTRIBUTION_REPORTS_UNSUBSCRIBE error code, 5290
ER_NDB_SCHEMA_GENERIC_MESSAGE error code, 5269
ER_NDB_SERVER_ID_RESERVED_OR_TOO_LARGE error code, 5294
ER_NDB_SHARE_ALREADY_EXISTS error code, 5275
ER_NDB_SKIPPING_SETUP_TABLE error code, 5289
ER_NDB_SLAVE_BINLOG_MISSING_INFO_FOR_CONFLICT_DETECTION error code, 5273
ER_NDB_SLAVE_CANT_ALLOCATE_TABLE_SHARE error code, 5274
ER_NDB_SLAVE_CONFLICT_DETECTION_REQUIRES_TRANSACTION_IDS error code, 5273
ER_NDB_SLAVE_CONFLICT_FUNCTION_REQUIRES_ROLE error code, 5273
ER_NDB_SLAVE_ERROR_IN_UPDATE_CREATE_INFO error code, 5274

ER_NDB_SLAVE_LOGGING_EXCEPTIONS_TO error code, 5263
ER_NDB_SLAVE_LOW_EPOCH_RESOLUTION error code, 5263
ER_NDB_SLAVE_MALFORMED_EVENT_RECEIVED_ON_TABLE error code, 5273
ER_NDB_SLAVE_MAX_REPLICATED_EPOCH_SET_TO error code, 5272
ER_NDB_SLAVE_MAX_REPLICATED_EPOCH_UNKNOWN error code, 5272
ER_NDB_SLAVE_MISSING_DATA_FOR_TIMESTAMP_COLUMN error code, 5263
ER_NDB_SLAVE_PARALLEL_WORKERS error code, 5321
ER_NDB_SLAVE_PREVIOUS_EPOCH_NOT_COMMITTED error code, 5262
ER_NDB_SLAVE_SAW_ALREADY_COMMITTED_EPOCH error code, 5262
ER_NDB_SLAVE_SAW_EPOCH_LOWER_THAN_PREVIOUS error code, 5262
ER_NDB_SLAVE_SAW_EPOCH_LOWER_THAN_PREVIOUS_ON_START error code, 5262
ER_NDB_SLAVE_TOO_MANY_RETRIES error code, 5274
ER_NDB_STARTING_CONNECT_THREAD error code, 5310
ER_NDB_TABLES_INITIALLY_READ_ONLY_ON_RECONNECT error code, 5293
ER_NDB_TABLES_NOT_READY error code, 5240
ER_NDB_TABLE_IS_NOT_DISTRIBUTED error code, 5288
ER_NDB_TABLE_OPENED_READ_ONLY error code, 5275
ER_NDB_THREAD_TIMED_OUT error code, 5288
ER_NDB_TIMED_OUT_IN_DROP_TABLE error code, 5297
ER_NDB_TIMEOUT_WHILE_DISTRIBUTING error code, 5290
ER_NDB_TOO_MANY_CPUS_IN_NDB_RECV_THREAD_CPU_MASK error code, 5277
ER_NDB_TRAILING_SHARE_RELEASED_BY_CLOSE_CACHED_TABLES error code, 5275
ER_NDB_TRANS_DEPENDENCY_TRACKER_ERROR error code, 5295
ER_NDB_UNEXPECTED_RENAME_TYPE error code, 5274
ER_NDB_USING_NODEID error code, 5309
ER_NDB_USING_NODEID_LIST error code, 5309
ER_NDB_UTIL_THREAD_OOM error code, 5276
ER_NDB_WAITING_INFO error code, 5289
ER_NDB_WAITING_INFO_WITH_MAP error code, 5290
ER_NEED_FILE_INSTEAD_OF_DIR error code, 5237
ER_NEED_LOG_BIN error code, 5237
ER_NEED_REPREPARE error code, 5137
ER_NETWORK_NAMESPACES_NOT_SUPPORTED error code, 5593
ER_NETWORK_NAMESPACE_FILE_PATH_TOO_LONG error code, 5593
ER_NETWORK_NAMESPACE_NOT_ALLOWED_FOR_WILDCARD_ADDRESS error code, 5593
ER_NETWORK_READ_EVENT_CHECKSUM_FAILURE error code, 5561
ER_NET_ERROR_ON_WRITE error code, 5109
ER_NET_FCNTL_ERROR error code, 5109
ER_NET_OK_PACKET_TOO_LARGE error code, 5162
ER_NET_PACKETS_OUT_OF_ORDER error code, 5109
ER_NET_PACKET_TOO_LARGE error code, 5109
ER_NET_READ_ERROR error code, 5109
ER_NET_READ_ERROR_FROM_PIPE error code, 5109
ER_NET_READ_INTERRUPTED error code, 5109
ER_NET_UNCOMPRESS_ERROR error code, 5109
ER_NET_WAIT_ERROR error code, 5594
ER_NET_WRITE_INTERRUPTED error code, 5110
ER_NEW_ABORTING_CONNECTION error code, 5111
ER_NO error code, 5098
ER_NONEXISTING_GRANT error code, 5108
ER_NONEXISTING_PROC_GRANT error code, 5125
ER_NONEXISTING_TABLE_GRANT error code, 5109
ER_NONPOSITIVE_RADIUS error code, 5194
ER_NONUNIQ_TABLE error code, 5103

ER_NONUPDATEABLE_COLUMN error code, 5122
ER_NON_BOOLEAN_EXPR_FOR_CHECK_CONSTRAINT error code, 5206
ER_NON_DEFAULT_VALUE_FOR_GENERATED_COLUMN error code, 5165
ER_NON_GROUPING_FIELD_USED error code, 5129
ER_NON_INSERTABLE_TABLE error code, 5129
ER_NON_RO_SELECT_DISABLE_TIMER error code, 5159
ER_NON_UNIQ_ERROR error code, 5101
ER_NON_UPDATABLE_TABLE error code, 5118
ER_NORMAL_SERVER_SHUTDOWN error code, 5560
ER_NORMAL_SHUTDOWN error code, 5104
ER_NOT_ALLOWED_COMMAND error code, 5109
ER_NOT_FORM_FILE error code, 5100
ER_NOT_HINT_UPDATABLE_VARIABLE error code, 5186
ER_NOT_IMPLEMENTED_FOR_CARTESIAN_SRS error code, 5193
ER_NOT_IMPLEMENTED_FOR_GEOGRAPHIC_SRS error code, 5184
ER_NOT_IMPLEMENTED_FOR_PROJECTED_SRS error code, 5194
ER_NOT_IMPLEMENTED_GET_TABLESPACE_STATISTICS error code, 5363
ER_NOT_KEYFILE error code, 5100
ER_NOT_RIGHT_NOW error code, 5234
ER_NOT_SUPPORTED_AUTH_MODE error code, 5115
ER_NOT_SUPPORTED_YET error code, 5114
ER_NOT_VALID_PASSWORD error code, 5152
ER_NO_ACCESS_TO_NATIVE_FCT error code, 5178
ER_NO_BINARY_LOGGING error code, 5124
ER_NO_BINLOG_ERROR error code, 5132
ER_NO_CSV_NO_LOG_TABLES error code, 5236
ER_NO_DB_ERROR error code, 5101
ER_NO_DEFAULT error code, 5114
ER_NO_DEFAULT_FOR_FIELD error code, 5123
ER_NO_DEFAULT_FOR_VIEW_FIELD error code, 5126
ER_NO_FORMAT_DESCRIPTION_EVENT_BEFORE_BINLOG_STATEMENT error code, 5137
ER_NO_FT_MATERIALIZED_SUBQUERY error code, 5159
ER_NO_PARTITION_FOR_GIVEN_VALUE error code, 5133
ER_NO_PARTITION_FOR_GIVEN_VALUE_SILENT error code, 5136
ER_NO_PARTS_ERROR error code, 5131
ER_NO_PATH_FOR_SHARED_LIBRARY error code, 5567
ER_NO_PERMISSION_TO_CREATE_USER error code, 5113
ER_NO_REFERENCED_ROW error code, 5113
ER_NO_REFERENCED_ROW_2 error code, 5128
ER_NO_SECURE_TRANSPORTS_CONFIGURED error code, 5170
ER_NO_SESSION_TEMP error code, 5214
ER_NO_SUCH_DB error code, 5173
ER_NO_SUCH_INDEX error code, 5104
ER_NO_SUCH_TABLE error code, 5109
ER_NO_SUCH_THREAD error code, 5105
ER_NO_SUCH_USER error code, 5128
ER_NO_SUPER_WITHOUT_USER_PLUGIN error code, 5254
ER_NO_SYSTEM_SCHEMA_ACCESS error code, 5177
ER_NO_SYSTEM_TABLESPACE_ACCESS error code, 5177
ER_NO_SYSTEM_TABLE_ACCESS error code, 5177
ER_NO_SYSTEM_TABLE_ACCESS_FOR_DICTIONARY_TABLE error code, 5177
ER_NO_SYSTEM_TABLE_ACCESS_FOR_SYSTEM_TABLE error code, 5177
ER_NO_SYSTEM_TABLE_ACCESS_FOR_TABLE error code, 5177
ER_NO_SYSTEM_VIEW_ACCESS error code, 5184

ER_NO_TABLES_USED error code, 5105
ER_NO_THD_NO_UUID error code, 5227
ER_NO_TRIGGERS_ON_SYSTEM_SCHEMA error code, 5129
ER_NO_UNIQUE_LOGFILE error code, 5105
ER_NPIPE_CANT_CREATE error code, 5320
ER_NPIPE_FAILED_TO_INIT_SECURITY_DESCRIPTOR error code, 5262
ER_NPIPE_FAILED_TO_SET_SECURITY_DESCRIPTOR error code, 5262
ER_NPIPE_PIPE_ALREADY_IN_USE error code, 5262
ER_NULL_COLUMN_IN_INDEX error code, 5107
ER_NULL_IN_VALUES_LESS_THAN error code, 5135
ER_NUMERIC_JSON_VALUE_OUT_OF_RANGE error code, 5169
ER_OLD_FILE_FORMAT error code, 5128
ER_OLD_KEYFILE error code, 5100
ER_OLD_PASSWORDS_NO_MIDDLE_GROUND error code, 5228
ER_OLD_TEMPORALS_UPGRADED error code, 5157
ER_ONLY_FD_AND_RBR_EVENTS_ALLOWED_IN_BINLOG_STATEMENT error code, 5146
ER_ONLY_IMPLEMENTED_FOR_SRID_0_AND_4326 error code, 5191
ER_ONLY_INTEGERS_ALLOWED error code, 5135
ER_ONLY_ON_RANGE_LIST_PARTITION error code, 5132
ER_OOM error code, 5234
ER_OOM_SAVE_GTIDS error code, 5560
ER_OPEN_AS_READONLY error code, 5100
ER_OPEN_ROLE_TABLES error code, 5175
ER_OPERAND_COLUMNS error code, 5115
ER_OPTION_PREVENTS_STATEMENT error code, 5118
ER_OPT_WRONG_TREE error code, 5220
ER_OUTOFMEMORY error code, 5101
ER_OUT_OF_RESOURCES error code, 5101
ER_OUT_OF_SORTMEMORY error code, 5101
ER_PAGE_TRACKING_CANNOT_PURGE error code, 5205
ER_PAGE_TRACKING_NOT_STARTED error code, 5205
ER_PAGE_TRACKING_RANGE_NOT_TRACKED error code, 5205
ER_PARSER_TRACE error code, 5220
ER_PARSE_ERROR error code, 5103
ER_PARSE_ERROR_IN_DIGEST_FN error code, 5190
ER_PARSING_VIEW error code, 5226
ER_PARTIAL_REVOKES_EXIST error code, 5215
ER_PARTIAL_REVOKE_AND_DB_GRANT_BOTH_EXISTS error code, 5216
ER_PARTITIONS_MUST_BE_DEFINED_ERROR error code, 5130
ER_PARTITION_CLAUSE_ON_NONPARTITIONED error code, 5147
ER_PARTITION_COLUMN_LIST_ERROR error code, 5140
ER_PARTITION_CONST_DOMAIN_ERROR error code, 5135
ER_PARTITION_ENTRY_ERROR error code, 5131
ER_PARTITION_EXCHANGE_DIFFERENT_OPTION error code, 5146
ER_PARTITION_EXCHANGE_FOREIGN_KEY error code, 5146
ER_PARTITION_EXCHANGE_PART_TABLE error code, 5146
ER_PARTITION_EXCHANGE_TEMP_TABLE error code, 5146
ER_PARTITION_FIELDS_TOO_LONG error code, 5140
ER_PARTITION_FUNCTION_FAILURE error code, 5132
ER_PARTITION_FUNCTION_IS_NOT_ALLOWED error code, 5135
ER_PARTITION_FUNC_NOT_ALLOWED_ERROR error code, 5130
ER_PARTITION_HANDLER_ADMIN_MSG error code, 5340
ER_PARTITION_INSTEAD_OF_SUBPARTITION error code, 5146
ER_PARTITION_MAXVALUE_ERROR error code, 5130

ER_PARTITION_MERGE_ERROR error code, 5135
ER_PARTITION_MGMT_ON_NONPARTITIONED error code, 5131
ER_PARTITION_MOVE_CREATED_DUPLICATE_ROW_PLEASE_FIX error code, 5320
ER_PARTITION_NAME error code, 5138
ER_PARTITION_NOT_DEFINED_ERROR error code, 5131
ER_PARTITION_NO_TEMPORARY error code, 5134
ER_PARTITION_REQUIRES_VALUES_ERROR error code, 5130
ER_PARTITION_WRONG_NO_PART_ERROR error code, 5130
ER_PARTITION_WRONG_NO_SUBPART_ERROR error code, 5130
ER_PARTITION_WRONG_VALUES_ERROR error code, 5130
ER_PART_EXPR_TOO_LONG error code, 5184
ER_PASSWORD_ANONYMOUS_USER error code, 5108
ER_PASSWORD_CANNOT_BE_RETAINED_ON_PLUGIN_CHANGE error code, 5215
ER_PASSWORD_EXPIRATION_NOT_SUPPORTED_BY_AUTH_METHOD error code, 5190
ER_PASSWORD_EXPIRE_ANONYMOUS_USER error code, 5159
ER_PASSWORD_FORMAT error code, 5153
ER_PASSWORD_NOT_ALLOWED error code, 5108
ER_PASSWORD_NO_MATCH error code, 5108
ER_PATH_IN_DATADIR error code, 5185
ER_PATH_LENGTH error code, 5142
ER_PERFSCHEMA_COMPONENTS_INFRASTRUCTURE_BOOTSTRAP error code, 5326
ER_PERFSCHEMA_COMPONENTS_INFRASTRUCTURE_SHUTDOWN error code, 5326
ER_PERFSCHEMA_INIT_FAILED error code, 5238
ER_PERFSCHEMA_TABLES_INIT_FAILED error code, 5327
ER_PERSISTENT_PRIVILEGES_BOOTSTRAP error code, 5327
ER_PERSIST_ONLY_ACCESS_DENIED_ERROR error code, 5185
ER_PERSIST_OPTION_HOST_TRUNCATED error code, 5594
ER_PERSIST_OPTION_STATUS error code, 5363
ER_PERSIST_OPTION_USER_TRUNCATED error code, 5594
ER_PER_CHANNEL_RPL_FILTER_CONF_FOR_GRP_RPL error code, 5336
ER_PFS_MALLOC_ARRAY_OOM error code, 5338
ER_PFS_MALLOC_ARRAY_OVERFLOW error code, 5338
ER_PFS_NOTIFICATION_FUNCTION_REGISTER_FAILED error code, 5331
ER_PID_FILEPATH_LOCATIONS_INACCESSIBLE error code, 5568
ER_PID_FILE_PRIV_DIRECTORY_INSECURE error code, 5428
ER_PK_INDEX_CANT_BE_INVISIBLE error code, 5175
ER_PLUGGABLE_PROTOCOL_COMMAND_NOT_SUPPORTED error code, 5167
ER_PLUGIN_BAD_OPTIONS error code, 5304
ER_PLUGIN_CANNOT_BE_UNINSTALLED error code, 5157
ER_PLUGIN_CANT_LOAD error code, 5303
ER_PLUGIN_CANT_OPEN_PLUGIN_TABLE error code, 5302
ER_PLUGIN_CANT_SET_PERSISTENT_OPTIONS error code, 5304
ER_PLUGIN_COMMON_FAILED_TO_OPEN_FILTER_TABLES error code, 5362
ER_PLUGIN_COMMON_FAILED_TO_OPEN_TABLE error code, 5362
ER_PLUGIN_DELETE_BUILTIN error code, 5138
ER_PLUGIN_DID_NOT_DEINITIALIZE_THREADS error code, 5272
ER_PLUGIN_DISABLED error code, 5304
ER_PLUGIN_FAILED_DEINITIALIZATION error code, 5302
ER_PLUGIN_FAILED_TO_OPEN_TABLE error code, 5565
ER_PLUGIN_FAILED_TO_OPEN_TABLES error code, 5565
ER_PLUGIN_FORCING_SHUTDOWN error code, 5303
ER_PLUGIN_HAS_CONFLICTING_SYSTEM_VARIABLES error code, 5304
ER_PLUGIN_HAS_NONZERO_REFCOUNT_AFTER_DEINITIALIZATION error code, 5302
ER_PLUGIN_HAS_NONZERO_REFCOUNT_AFTER_SHUTDOWN error code, 5303

ER_PLUGIN_INIT_FAILED error code, 5241
ER_PLUGIN_INSTALL_ERROR error code, 5574
ER_PLUGIN_IS_NOT_LOADED error code, 5132
ER_PLUGIN_IS_PERMANENT error code, 5143
ER_PLUGIN_LOAD_OPTIONS_IGNORED error code, 5603
ER_PLUGIN_LOAD_PARAMETER_TOO_LONG error code, 5303
ER_PLUGIN_NOT_EARLY error code, 5209
ER_PLUGIN_NOT_EARLY_DUP error code, 5594
ER_PLUGIN_NO_INSTALL error code, 5145
ER_PLUGIN_NO_INSTALL_DUP error code, 5594
ER_PLUGIN_NO_UNINSTALL error code, 5145
ER_PLUGIN_OOM error code, 5303
ER_PLUGIN_PARSING_OPTIONS_FAILED error code, 5304
ER_PLUGIN_REGISTRATION_FAILED error code, 5302
ER_PLUGIN_SHUTTING_DOWN_PLUGIN error code, 5302
ER_PLUGIN_UNINSTALL_ERROR error code, 5574
ER_PLUGIN_UNKNOWN_VARIABLE_TYPE error code, 5303
ER_PLUGIN_VARIABLE_MISSING_NAME error code, 5303
ER_PLUGIN_VARIABLE_NOT_ALLOCATED_THREAD_LOCAL error code, 5303
ER_PLUGIN_VARIABLE_SET_READ_ONLY error code, 5303
ER_POINTLESS_WITHOUT_SLOWLOG error code, 5228
ER_POLYGON_TOO_LARGE error code, 5196
ER_PREPARE_FOR_SECONDARY_ENGINE error code, 5588
ER_PREVENTS_VARIABLE_WITHOUT_RBR error code, 5165
ER_PRIMARY_CANT_HAVE_NULL error code, 5110
ER_PRIVILEGE_SYSTEM_INIT_FAILED error code, 5327
ER_PROCACCESS_DENIED_ERROR error code, 5123
ER_PROC_AUTO_GRANT_FAIL error code, 5125
ER_PROC_AUTO_REVOKE_FAIL error code, 5125
ER_PROTOCOL_COMPRESSION_RESET_LOG error code, 5601
ER_PS_MANY_PARAM error code, 5124
ER_PS_NO_RECURSION error code, 5127
ER_QUERY_CACHE_DISABLED error code, 5140
ER_QUERY_INTERRUPTED error code, 5120
ER_QUERY_ON_FOREIGN_DATA_SOURCE error code, 5126
ER_QUERY_TIMEOUT error code, 5159
ER_RANGE_NOT_INCREASING_ERROR error code, 5130
ER_READING_TABLE_FAILED error code, 5242
ER_READY error code, 5104
ER_READ_LOG_EVENT_FAILED error code, 5335
ER_READ_ONLY_MODE error code, 5153
ER_READ_ONLY_TRANSACTION error code, 5112
ER_REALLY_RUN_AS_ROOT error code, 5233
ER_RECORD_FILE_FULL error code, 5106
ER_RECOVERING_TABLE error code, 5305
ER_REFERENCED_TRG_DOES_NOT_EXIST error code, 5158
ER_REGEXP_BAD_ESCAPE_SEQUENCE error code, 5192
ER_REGEXP_BAD_INTERVAL error code, 5192
ER_REGEXP_BUFFER_OVERFLOW error code, 5191
ER_REGEXP_ERROR error code, 5108
ER_REGEXP_ILLEGAL_ARGUMENT error code, 5191
ER_REGEXP_INDEX_OUTOFBOUNDS_ERROR error code, 5191
ER_REGEXP_INTERNAL_ERROR error code, 5192
ER_REGEXP_INVALID_BACK_REF error code, 5192

ER_REGEXP_INVALID_CAPTURE_GROUP_NAME error code, 5214
ER_REGEXP_INVALID_FLAG error code, 5215
ER_REGEXP_INVALID_RANGE error code, 5193
ER_REGEXP_LOOK_BEHIND_LIMIT error code, 5192
ER_REGEXP_MAX_LT_MIN error code, 5192
ER_REGEXP_MISMATCHED_PAREN error code, 5192
ER_REGEXP_MISSING_CLOSE_BRACKET error code, 5192
ER_REGEXP_PATTERN_TOO_BIG error code, 5193
ER_REGEXP_RULE_SYNTAX error code, 5192
ER_REGEXP_STACK_OVERFLOW error code, 5193
ER_REGEXP_STRING_NOT_TERMINATED error code, 5191
ER_REGEXP_TIME_OUT error code, 5193
ER_REGEXP_UNIMPLEMENTED error code, 5192
ER_RELAY_LOG_FAIL error code, 5123
ER_RELAY_LOG_INIT error code, 5124
ER_RELAY_LOG_SPACE_LIMIT_DISABLED error code, 5570
ER_REMOVED_SPACES error code, 5129
ER_RENAMED_NAME error code, 5139
ER_RENAME_ROLE error code, 5175
ER_REORG_HASH_ONLY_ON_SAME_NO error code, 5132
ER_REORG_NO_PARAM_ERROR error code, 5132
ER_REORG_OUTSIDE_RANGE error code, 5132
ER_REORG_PARTITION_NOT_EXIST error code, 5132
ER_REPLACE_INACCESSIBLE_ROWS error code, 5162
ER_REQUIRES_PRIMARY_KEY error code, 5110
ER_RESERVED_SYNTAX error code, 5124
ER_RESERVED_TABLESPACE_NAME error code, 5195
ER_RESET_MASTER_TO_VALUE_OUT_OF_RANGE error code, 5178
ER_RESIGNAL_WITHOUT_ACTIVE_HANDLER error code, 5139
ER_RESOURCE_GROUP_BIND_FAILED error code, 5189
ER_RESOURCE_GROUP_BUSY error code, 5188
ER_RESOURCE_GROUP_DISABLED error code, 5188
ER_RESOURCE_GROUP_EXISTS error code, 5188
ER_RESOURCE_GROUP_IS_DISABLED error code, 5330
ER_RESOURCE_GROUP_METADATA_UPDATE_SKIPPED error code, 5329
ER_RESOURCE_GROUP_NOT_EXISTS error code, 5188
ER_RESOURCE_GROUP_POST_INIT_FAILED error code, 5326
ER_RESOURCE_GROUP_SUBSYSTEM_INIT_FAILED error code, 5326
ER_RESOURCE_GROUP_VALIDATION_FAILED error code, 5330
ER_RESTART_RECEIVED_INFO error code, 5343
ER_RESTART_SERVER_FAILED error code, 5194
ER_RES_GRP_FAILED_DETERMINE_CPU_COUNT error code, 5332
ER_RES_GRP_FAILED_TO_DETERMINE_NICE_CAPABILITY error code, 5331
ER_RES_GRP_FAILED_TO_GET_THREAD_HANDLE error code, 5331
ER_RES_GRP_FEATURE_NOT_AVAILABLE error code, 5332
ER_RES_GRP_GET_THREAD_PRIO_NOT_SUPPORTED error code, 5331
ER_RES_GRP_INVALID_THREAD_PRIORITY error code, 5332
ER_RES_GRP_INVALID_VCPU_ID error code, 5342
ER_RES_GRP_INVALID_VCPU_RANGE error code, 5342
ER_RES_GRP_SET_THREAD_PRIORITY_FAILED error code, 5331
ER_RES_GRP_SET_THR_AFFINITY_FAILED error code, 5331
ER_RES_GRP_SET_THR_AFFINITY_TO_CPUS_FAILED error code, 5331
ER_RES_GRP_SOLARIS_PROCESSOR_AFFINITY_FAILED error code, 5332
ER_RES_GRP_SOLARIS_PROCESSOR_BIND_TO_CPUID_FAILED error code, 5332

ER_RES_GRP_SOLARIS_PROCESSOR_BIND_TO_THREAD_FAILED error code, 5332
ER_RES_GRP_THD_UNBIND_FROM_CPU_FAILED error code, 5331
ER_RETRYING_REPAIR_WITHOUT_QUICK error code, 5304
ER_RETRYING_REPAIR_WITH_KEYCACHE error code, 5304
ER_REVOKE_GRANTS error code, 5116
ER_REWRITER_LOAD_FAILED error code, 5364
ER_REWRITER_OOM error code, 5364
ER_REWRITER_QUERY_ERROR_MSG error code, 5360
ER_REWRITER_QUERY_FAILED error code, 5360
ER_REWRITER_READ_FAILED error code, 5364
ER_REWRITER_TABLE_MALFORMED_ERROR error code, 5364
ER_ROLE_NOT_GRANTED error code, 5175
ER_ROW_DATA_TOO_BIG_TO_WRITE_IN_BINLOG error code, 5335
ER_ROW_DOES_NOT_MATCH_GIVEN_PARTITION_SET error code, 5147
ER_ROW_DOES_NOT_MATCH_PARTITION error code, 5146
ER_ROW_IN_WRONG_PARTITION error code, 5155
ER_ROW_IN_WRONG_PARTITION_PLEASE_REPAIR error code, 5245
ER_ROW_IS_REFERENCED error code, 5113
ER_ROW_IS_REFERENCED_2 error code, 5128
ER_ROW_SINGLE_PARTITION_FIELD_ERROR error code, 5140
ER_RPL_BINLOG_FILTERS_OOM error code, 5231
ER_RPL_BINLOG_MASTER_SENDS_HEARTBEAT error code, 5270
ER_RPL_BINLOG_MASTER_USES_CHECKSUM_AND_SLAVE_CANT error code, 5271
ER_RPL_BINLOG_RELAY_DELEGATES_INIT_FAILED error code, 5241
ER_RPL_BINLOG_SKIPPING_REMAINING_HEARTBEAT_INFO error code, 5271
ER_RPL_BINLOG_STARTING_DUMP error code, 5270
ER_RPL_BINLOG_STORAGE_DELEGATES_INIT_FAILED error code, 5241
ER_RPL_BINLOG_TRANSMIT_DELEGATES_INIT_FAILED error code, 5241
ER_RPL_CANT_ADD_DO_TABLE error code, 5235
ER_RPL_CANT_ADD_IGNORE_TABLE error code, 5235
ER_RPL_CANT_CREATE_SLAVE_THREAD error code, 5280
ER_RPL_CANT_FIND_FOLLOWUP_FILE error code, 5284
ER_RPL_CANT_HAVE_SAME_BASENAME error code, 5323
ER_RPL_CANT_INITIALIZE_GTID_SETS_IN_RLI_INIT_INFO error code, 5287
ER_RPL_CANT_MAKE_PATHS error code, 5237
ER_RPL_CANT_OPEN_INFO_TABLE error code, 5264
ER_RPL_CANT_OPEN_LOG_IN_RLI_INIT_INFO error code, 5288
ER_RPL_CANT_SCAN_INFO_TABLE error code, 5264
ER_RPL_CANT_STOP_SLAVE_WHILE_LOCKED_BACKUP error code, 5211
ER_RPL_CHANGING_RELAY_LOG_INFO_REPO_TYPE_FAILED_DUE_TO_GAPS error code, 5265
ER_RPL_CHANNELS_REQUIRE_NON_ZERO_SERVER_ID error code, 5265
ER_RPL_CHANNELS_REQUIRE_TABLES_AS_INFO_REPOSITORIES error code, 5264
ER_RPL_CORRUPTED_INFO_TABLE error code, 5264
ER_RPL_CORRUPTED_KEYS_IN_INFO_TABLE error code, 5264
ER_RPL_ENCRYPTION_CANNOT_ROTATE_BINLOG_MASTER_KEY error code, 5205
ER_RPL_ENCRYPTION_FAILED_TO_ENCRYPT error code, 5205
ER_RPL_ENCRYPTION_FAILED_TO_FETCH_KEY error code, 5203
ER_RPL_ENCRYPTION_FAILED_TO_GENERATE_KEY error code, 5203
ER_RPL_ENCRYPTION_FAILED_TO_REMOVE_KEY error code, 5203
ER_RPL_ENCRYPTION_FAILED_TO_ROTATE_LOGS error code, 5203
ER_RPL_ENCRYPTION_FAILED_TO_STORE_KEY error code, 5203
ER_RPL_ENCRYPTION_HEADER_ERROR error code, 5203
ER_RPL_ENCRYPTION_KEYRING_INVALID_KEY error code, 5203
ER_RPL_ENCRYPTION_KEY_EXISTS_UNEXPECTED error code, 5203

ER_RPL_ENCRYPTION_KEY_NOT_FOUND error code, 5203
ER_RPL_ENCRYPTION_MASTER_KEY_RECOVERY_FAILED error code, 5204
ER_RPL_ENCRYPTION_UNABLE_TO_CHANGE_OPTION error code, 5204
ER_RPL_ERROR_CHANGING_MASTER_INFO_REPO_TYPE error code, 5265
ER_RPL_ERROR_CHANGING_RELAY_LOG_INFO_REPO_TYPE error code, 5265
ER_RPL_ERROR_CHECKING_REPOSITORY error code, 5266
ER_RPL_ERROR_CREATING_MASTER_INFO error code, 5265
ER_RPL_ERROR_CREATING_RELAY_LOG_INFO error code, 5265
ER_RPL_ERROR_LOOKING_FOR_LOG error code, 5283
ER_RPL_ERROR_READING_MASTER_CONFIGURATION error code, 5302
ER_RPL_ERROR_READING_SLAVE_WORKER_CONFIGURATION error code, 5300
ER_RPL_ERROR_WRITING_MASTER_CONFIGURATION error code, 5302
ER_RPL_ERROR_WRITING_RELAY_LOG_CONFIGURATION error code, 5288
ER_RPL_ERROR_WRITING_SLAVE_WORKER_CONFIGURATION error code, 5301
ER_RPL_FAILED_TO_CREATE_CACHE_FOR_INFO_FILE error code, 5267
ER_RPL_FAILED_TO_CREATE_NEW_INFO_FILE error code, 5266
ER_RPL_FAILED_TO_DELETE_FROM_SLAVE_WORKERS_INFO_REPOSITORY error code, 5265
ER_RPL_FAILED_TO_OPEN_INFO_FILE error code, 5267
ER_RPL_FAILED_TO_OPEN_RELAY_LOG error code, 5301
ER_RPL_FAILED_TO_RESET_STATE_IN_SLAVE_INFO_REPOSITORY error code, 5266
ER_RPL_FAILED_TO_STAT_LOG_IN_INDEX error code, 5287
ER_RPL_FILTERS_NOT_ATTACHED_TO_CHANNEL error code, 5336
ER_RPL_FILTER_ADD_WILD_DO_TABLE_FAILED error code, 5327
ER_RPL_FILTER_ADD_WILD_IGNORE_TABLE_FAILED error code, 5327
ER_RPL_GTID_LOG_EVENT_IN_STREAM error code, 5268
ER_RPL_GTID_MEMORY_FINALLY_AVAILABLE error code, 5267
ER_RPL_GTID_MODE_REQUIRES_ENFORCE_GTID_CONSISTENCY_ON error code, 5323
ER_RPL_GTID_TABLE_CANNOT_OPEN error code, 5221
ER_RPL_GTID_UNSAFE_ALTER_ADD_COL_WITH_DEFAULT_EXPRESSION error code, 5573
ER_RPL_GTID_UNSAFE_STMT_CREATE_SELECT error code, 5559
ER_RPL_GTID_UNSAFE_STMT_ON_NON_TRANS_TABLE error code, 5559
ER_RPL_GTID_UNSAFE_STMT_ON_TEMPORARY_TABLE error code, 5559
ER_RPL_INCOMPATIBLE_DECIMAL_IN_RBR error code, 5269
ER_RPL_INCONSISTENT_SEQUENCE_NO_IN_TRX error code, 5264
ER_RPL_INCONSISTENT_TIMESTAMPS_IN_TRX error code, 5264
ER_RPL_INFINITY_DENIED error code, 5240
ER_RPL_INFINITY_IGNORED error code, 5240
ER_RPL_INFO_DATA_TOO_LONG error code, 5146
ER_RPL_IO_THREAD_KILLED error code, 5336
ER_RPL_LOG_ENTRY_EXCEEDS_SLAVE_MAX_ALLOWED_PACKET error code, 5283
ER_RPL_LOG_NOT_FOUND_WHILE_COUNTING_RELAY_LOG_SPACE error code, 5287
ER_RPL_MTS_AUTOMATIC_RECOVERY_FAILED error code, 5280
ER_RPL_MTS_CHECKPOINT_PERIOD_DIFFERS_FROM_CNT error code, 5284
ER_RPL_MTS_GROUP_RECOVERY_RELAY_LOG_INFO error code, 5284
ER_RPL_MTS_GROUP_RECOVERY_RELAY_LOG_INFO_FOR_WORKER error code, 5283
ER_RPL_MTS_RECOVERY_CANT_OPEN_RELAY_LOG error code, 5280
ER_RPL_MTS_RECOVERY_COMPLETE error code, 5282
ER_RPL_MTS_RECOVERY_FAILED_TO_START_COORDINATOR error code, 5280
ER_RPL_MTS_RECOVERY_STARTING_COORDINATOR error code, 5280
ER_RPL_MTS_RECOVERY_SUCCESSFUL error code, 5280
ER_RPL_MTS_SLAVE_COORDINATOR_HAS_WAITED error code, 5301
ER_RPL_MTS_STATISTICS error code, 5282
ER_RPL_MULTISOURCE_REQUIRES_TABLE_TYPE_REPOSITORIES error code, 5266
ER_RPL_OPEN_INDEX_FILE_FAILED error code, 5287

ER_RPL_PLEASE_USE_OPTION_RELAY_LOG error code, 5287
ER_RPL_PLUGIN_FUNCTION_FAILED error code, 5241
ER_RPL_RECOVERY_ERROR error code, 5279
ER_RPL_RECOVERY_ERROR_FREEING_IO_CACHE error code, 5279
ER_RPL_RECOVERY_ERROR_READ_RELAY_LOG error code, 5279
ER_RPL_RECOVERY_FILE_MASTER_POS_INFO error code, 5279
ER_RPL_RECOVERY_IO_ERROR_READING_RELAY_LOG_INDEX error code, 5279
ER_RPL_RECOVERY_NO_ROTATE_EVENT_FROM_MASTER error code, 5279
ER_RPL_RECOVERY_REPLICATE_SAME_SERVER_ID_REQUIRES_POSITION error code, 5279
ER_RPL_RECOVERY_SKIPPED_GROUP_REPLICATION_CHANNEL error code, 5279
ER_RPL_RELAY_LOG_INDEX_NEEDS_FILE_NOT_DIRECTORY error code, 5287
ER_RPL_RELAY_LOG_NEEDS_FILE_NOT_DIRECTORY error code, 5287
ER_RPL_REPO_HAS_GAPS error code, 5223
ER_RPL_REPO_SHOULD_BE_TABLE error code, 5265
ER_RPL_REWRITEDB_EMPTY_FROM error code, 5236
ER_RPL_REWRITEDB_EMPTY_TO error code, 5236
ER_RPL_REWRITEDB_MISSING_ARROW error code, 5236
ER_RPL_RLI_INIT_INFO_MSG error code, 5340
ER_RPL_SERVER_ID_MISSING error code, 5280
ER_RPL_SLAVE_ADDITIONAL_ERROR_INFO_FROM_DA error code, 5285
ER_RPL_SLAVE_AUTO_POSITION_IS_1_AND_GTID_MODE_IS_OFF error code, 5278
ER_RPL_SLAVE_CANT_FLUSH_MASTER_INFO_FILE error code, 5281
ER_RPL_SLAVE_CANT_INITIALIZE_SLAVE_WORKER error code, 5283
ER_RPL_SLAVE_CANT_INIT_RELAY_LOG_POSITION error code, 5282
ER_RPL_SLAVE_CANT_START_SLAVE_FOR_CHANNEL error code, 5278
ER_RPL_SLAVE_CANT_STOP_SLAVE_FOR_CHANNEL error code, 5279
ER_RPL_SLAVE_CANT_USE_CHARSET error code, 5286
ER_RPL_SLAVE_CONNECTED_TO_MASTER_REPLICATION_RESUMED error code, 5286
ER_RPL_SLAVE_CONNECTED_TO_MASTER_REPLICATION_STARTED error code, 5282
ER_RPL_SLAVE_COULD_NOT_CREATE_CHANNEL_LIST error code, 5266
ER_RPL_SLAVE_DUMP_THREAD_KILLED_BY_MASTER error code, 5282
ER_RPL_SLAVE_ERROR_INFO_FROM_DA error code, 5285
ER_RPL_SLAVE_ERROR_LOADING_USER_DEFINED_LIBRARY error code, 5285
ER_RPL_SLAVE_ERROR_READING_FROM_SERVER error code, 5282
ER_RPL_SLAVE_ERROR_READING_RELAY_LOG_EVENTS error code, 5286
ER_RPL_SLAVE_ERROR_REQUESTING_BINLOG_DUMP error code, 5283
ER_RPL_SLAVE_ERROR_RETRYING error code, 5281
ER_RPL_SLAVE_ERROR_RUNNING_QUERY error code, 5285
ER_RPL_SLAVE_FAILED_TO_CREATE_CHANNEL_FROM_MASTER_INFO error code, 5266
ER_RPL_SLAVE_FAILED_TO_CREATE_OR_RECOVER_INFO_REPOSITORIES error code, 5278
ER_RPL_SLAVE_FAILED_TO_INIT_A_MASTER_INFO_STRUCTURE error code, 5266
ER_RPL_SLAVE_FAILED_TO_INIT_MASTER_INFO_STRUCTURE error code, 5266
ER_RPL_SLAVE_FAILED_TO_INIT_PARTITIONS_HASH error code, 5284
ER_RPL_SLAVE_FILTER_CREATE_FAILED error code, 5328
ER_RPL_SLAVE_FLUSH_RELAY_LOGS_NOT_ALLOWED error code, 5321
ER_RPL_SLAVE_FORCING_TO_RECONNECT_IO_THREAD error code, 5283
ER_RPL_SLAVE_GENERIC_MESSAGE error code, 5266
ER_RPL_SLAVE_GLOBAL_FILTERS_COPY_FAILED error code, 5329
ER_RPL_SLAVE_INCORRECT_CHANNEL error code, 5322
ER_RPL_SLAVE_INSECURE_CHANGE_MASTER error code, 5321
ER_RPL_SLAVE_IO_THREAD_ABORTED_WAITING_FOR_RELAY_LOG_SPACE error code, 5283
ER_RPL_SLAVE_IO_THREAD_CANT_REGISTER_ON_MASTER error code, 5282
ER_RPL_SLAVE_IO_THREAD_DETECTED_UNEXPECTED_EVENT_SEQUENCE error code, 5286
ER_RPL_SLAVE_IO_THREAD_EXITING error code, 5283

ER_RPL_SLAVE_IO_THREAD_KILLED error code, 5282
ER_RPL_SLAVE_IO_THREAD_STOP_CMD_EXEC_TIMEOUT error code, 5559
ER_RPL_SLAVE_IO_THREAD_WAS_KILLED error code, 5280
ER_RPL_SLAVE_MASTER_UUID_HAS_CHANGED error code, 5281
ER_RPL_SLAVE_NDB_TABLES_NOT_AVAILABLE error code, 5284
ER_RPL_SLAVE_NEW_MASTER_INFO_NEEDS_REPOS_TYPE_OTHER_THAN_FILE error code, 5287
ER_RPL_SLAVE_NEXT_LOG_IS_ACTIVE error code, 5286
ER_RPL_SLAVE_NEXT_LOG_IS_INACTIVE error code, 5286
ER_RPL_SLAVE_QUEUE_EVENT_FAILED_INVALID_CONFIGURATION error code, 5285
ER_RPL_SLAVE_READ_INVALID_EVENT_FROM_MASTER error code, 5285
ER_RPL_SLAVE_REPORT_HOST_TOO_LONG error code, 5281
ER_RPL_SLAVE_REPORT_PASSWORD_TOO_LONG error code, 5281
ER_RPL_SLAVE_REPORT_USER_TOO_LONG error code, 5281
ER_RPL_SLAVE_RESET_FILTER_OPTIONS error code, 5329
ER_RPL_SLAVE_SECONDS_BEHIND_MASTER_DUBIOUS error code, 5281
ER_RPL_SLAVE_SKIP_COUNTER_EXECUTED error code, 5285
ER_RPL_SLAVE_SQL_THREAD_EXITING error code, 5285
ER_RPL_SLAVE_SQL_THREAD_IO_ERROR_READING_EVENT error code, 5286
ER_RPL_SLAVE_SQL_THREAD_STARTING error code, 5284
ER_RPL_SLAVE_SQL_THREAD_STARTING_WITH_PRIVILEGE_CHECKS error code, 5602
ER_RPL_SLAVE_SQL_THREAD_STOP_CMD_EXEC_TIMEOUT error code, 5559
ER_RPL_SLAVE_STOPPING_AS_MASTER_OOM error code, 5283
ER_RPL_SLAVE_USES_CHECKSUM_AND_MASTER_PRE_50 error code, 5281
ER_RPL_SLAVE_WORKER_THREAD_CREATION_FAILED error code, 5284
ER_RPL_SLAVE_WORKER_THREAD_CREATION_FAILED_WITH_ERRNO error code, 5284
ER_RPL_SSL_INFO_IN_MASTER_INFO_IGNORED error code, 5302
ER_RPL_TIMESTAMPS_RETURNED_TO_NORMAL error code, 5328
ER_RPL_TRX_DELEGATES_INIT_FAILED error code, 5241
ER_RPL_UNEXPECTED_BEGIN_IN_STREAM error code, 5268
ER_RPL_UNEXPECTED_COMMIT_ROLLBACK_OR_XID_LOG_EVENT_IN_STREAM error code, 5268
ER_RPL_UNEXPECTED_XA_ROLLBACK_IN_STREAM error code, 5269
ER_RPL_UNSUPPORTED_UNIGNORABLE_EVENT_IN_STREAM error code, 5268
ER_RPL_WORKER_CANT_FIND_NEXT_RELAY_LOG error code, 5301
ER_RPL_WORKER_CANT_READ_RELAY_LOG error code, 5301
ER_RPL_WORKER_ID_IS error code, 5264
ER_RPL_ZOMBIE_ENCOUNTERED error code, 5221
ER_RUNNING_APPLIER_PREVENTS_SWITCH_GLOBAL_BINLOG_FORMAT error code, 5198
ER_RUN_HOOK_ERROR error code, 5165
ER_SAME_NAME_PARTITION error code, 5132
ER_SAME_NAME_PARTITION_FIELD error code, 5140
ER_SCHEDULER_KILLING error code, 5225
ER_SCHEDULER_STARTED error code, 5225
ER_SCHEDULER_STOPPED error code, 5225
ER_SCHEDULER_STOPPING_FAILED_TO_CREATE_WORKER error code, 5225
ER_SCHEDULER_STOPPING_FAILED_TO_GET_EVENT error code, 5225
ER_SCHEDULER_WAITING error code, 5225
ER_SCHEMA_DIR_CREATE_FAILED error code, 5191
ER_SCHEMA_DIR_EXISTS error code, 5190
ER_SCHEMA_DIR_MISSING error code, 5191
ER_SCHEMA_DIR_UNKNOWN error code, 5191
ER_SDI_OPERATION_FAILED error code, 5189
ER_SECONDARY_ENGINE error code, 5214
ER_SECONDARY_ENGINE_DDL error code, 5214
ER_SECONDARY_ENGINE_PLUGIN error code, 5213

ER_SECOND_PASSWORD_CANNOT_BE_EMPTY error code, 5213
ER_SECURE_AUTH_VALUE_UNSUPPORTED error code, 5228
ER_SECURE_TRANSPORT_REQUIRED error code, 5170
ER_SEC_FILE_PRIV_ARGUMENT_TOO_LONG error code, 5230
ER_SEC_FILE_PRIV_CANT_ACCESS_DIR error code, 5230
ER_SEC_FILE_PRIV_CANT_STAT error code, 5230
ER_SEC_FILE_PRIV_DIRECTORY_INSECURE error code, 5230
ER_SEC_FILE_PRIV_DIRECTORY_PERMISSIONS error code, 5230
ER_SEC_FILE_PRIV_EMPTY error code, 5230
ER_SEC_FILE_PRIV_IGNORED error code, 5230
ER_SEC_FILE_PRIV_NULL error code, 5230
ER_SELECT_REDUCED error code, 5115
ER_SEMISYNC_ADD_ACK_TO_SLOT error code, 5351
ER_SEMISYNC_BINLOG_REPLY_IS_AHEAD error code, 5350
ER_SEMISYNC_BINLOG_WRITE_OUT_OF_ORDER error code, 5348
ER_SEMISYNC_CLEARED_ACTIVE_TRANSACTION_TILL_POS error code, 5349
ER_SEMISYNC_CLEARED_ALL_ACTIVE_TRANSACTION_NODES error code, 5348
ER_SEMISYNC_DISABLED_ON_MASTER error code, 5349
ER_SEMISYNC_EXECUTION_FAILED_ON_MASTER error code, 5354
ER_SEMISYNC_FAILED_REGISTER_SLAVE_TO_RECEIVER error code, 5352
ER_SEMISYNC_FAILED_TO_ALLOCATE_TRX_NODE error code, 5348
ER_SEMISYNC_FAILED_TO_INSERT_TRX_NODE error code, 5351
ER_SEMISYNC_FAILED_TO_START_ACK_RECEIVER_THD error code, 5352
ER_SEMISYNC_FAILED_TO_STOP_ACK_RECEIVER_THD error code, 5354
ER_SEMISYNC_FAILED_TO_WAIT_ON_DUMP_SOCKET error code, 5352
ER_SEMISYNC_FORCED_SHUTDOWN error code, 5350
ER_SEMISYNC_FUNCTION_CALLED_TWICE error code, 5349
ER_SEMISYNC_INIT_WAIT_POS error code, 5350
ER_SEMISYNC_INSERT_LOG_INFO_IN_ENTRY error code, 5348
ER_SEMISYNC_MASTER_FAILED_ON_NET_FLUSH error code, 5351
ER_SEMISYNC_MASTER_GOT_REPLY_AT_POS error code, 5350
ER_SEMISYNC_MASTER_OOM error code, 5349
ER_SEMISYNC_MASTER_SIGNAL_ALL_WAITING_THREADS error code, 5350
ER_SEMISYNC_MASTER_TRX_WAIT_POS error code, 5350
ER_SEMISYNC_MISSING_MAGIC_NO_FOR_SEMISYNC_PKT error code, 5353
ER_SEMISYNC_MOVE_BACK_WAIT_POS error code, 5350
ER_SEMISYNC_NOT_SUPPORTED_BY_MASTER error code, 5354
ER_SEMISYNC_NO_SPACE_IN_THE_PKT error code, 5351
ER_SEMISYNC_PROBE_LOG_INFO_IN_ENTRY error code, 5348
ER_SEMISYNC_RECEIVED_ACK_IS_SMALLER error code, 5351
ER_SEMISYNC_REPLY_BINLOG_FILE_TOO_LARGE error code, 5349
ER_SEMISYNC_REPLY_MAGIC_NO_ERROR error code, 5349
ER_SEMISYNC_REPLY_PKT_LENGTH_TOO_SMALL error code, 5349
ER_SEMISYNC_RPL_ENABLED_ON_MASTER error code, 5349
ER_SEMISYNC_RPL_INIT_FOR_TRX error code, 5348
ER_SEMISYNC_RPL_SWITCHED_OFF error code, 5351
ER_SEMISYNC_RPL_SWITCHED_ON error code, 5351
ER_SEMISYNC_SERVER_REPLY error code, 5349
ER_SEMISYNC_SLAVE_NET_FLUSH_REPLY_FAILED error code, 5354
ER_SEMISYNC_SLAVE_REPLY error code, 5353
ER_SEMISYNC_SLAVE_REPLY_WITH_BINLOG_INFO error code, 5353
ER_SEMISYNC_SLAVE_SEND_REPLY_FAILED error code, 5354
ER_SEMISYNC_SLAVE_SET_FAILED error code, 5354
ER_SEMISYNC_SLAVE_START error code, 5353

ER_SEMISYNC_SOCKET_FD_TOO_LARGE error code, 5353
ER_SEMISYNC_STARTING_ACK_RECEIVER_THD error code, 5352
ER_SEMISYNC_START_BINLOG_DUMP_TO_SLAVE error code, 5352
ER_SEMISYNC_STOPPING_ACK_RECEIVER_THREAD error code, 5352
ER_SEMISYNC_STOP_BINLOG_DUMP_TO_SLAVE error code, 5352
ER_SEMISYNC_SYNC_HEADER_UPDATE_INFO error code, 5351
ER_SEMISYNC_TRACE_ENTER_FUNC error code, 5347
ER_SEMISYNC_TRACE_EXIT error code, 5348
ER_SEMISYNC_TRACE_EXIT_WITH_BOOL_EXIT_CODE error code, 5348
ER_SEMISYNC_TRACE_EXIT_WITH_INT_EXIT_CODE error code, 5348
ER_SEMISYNC_TRX_SKIPPED_AT_POS error code, 5351
ER_SEMISYNC_UNREGISTERED_REPLICATOR error code, 5353
ER_SEMISYNC_UNREGISTER_BINLOG_STORAGE_OBSERVER_FAILED error code, 5353
ER_SEMISYNC_UNREGISTER_BINLOG_TRANSMIT_OBSERVER_FAILED error code, 5353
ER_SEMISYNC_UNREGISTER_TRX_OBSERVER_FAILED error code, 5353
ER_SEMISYNC_UPDATE_EXISTING_SLAVE_ACK error code, 5352
ER_SEMISYNC_WAIT_FOR_BINLOG_TIMEDOUT error code, 5350
ER_SEMISYNC_WAIT_TIME_ASSESSMENT_FOR_COMMIT_TRX_FAILED error code, 5351
ER_SEMISYNC_WAIT_TIME_FOR_BINLOG_SENT error code, 5350
ER_SERVERID_TOO_LARGE error code, 5239
ER_SERVER_ACL_TABLE_ERROR error code, 5564
ER_SERVER_BINLOG_MASTER_KEY_RECOVERY_OUT_OF_COMBINATION error code, 5588
ER_SERVER_BINLOG_MASTER_KEY_ROTATION_FAIL_TO_CLEANUP_AUX_KEY error code, 5588
ER_SERVER_BINLOG_UNSAFE_SYSTEM_FUNCTION error code, 5592
ER_SERVER_CANNOT_LOAD_FROM_TABLE_V2 error code, 5563
ER_SERVER_CANT_OPEN_FILE error code, 5563
ER_SERVER_COL_COUNT_DOESNT_MATCH_CORRUPTED_V2 error code, 5564
ER_SERVER_COL_COUNT_DOESNT_MATCH_PLEASE_UPDATE_V2 error code, 5563
ER_SERVER_COST_FAILED_TO_READ error code, 5268
ER_SERVER_COST_INVALID_COST_CONSTANT error code, 5267
ER_SERVER_COST_UNKNOWN_COST_CONSTANT error code, 5267
ER_SERVER_DISK_FULL_NOWAIT error code, 5563
ER_SERVER_FILE_NOT_FOUND error code, 5563
ER_SERVER_FILE_USED error code, 5563
ER_SERVER_GTID_UNSAFE_CREATE_DROP_TEMP_TABLE_IN_TRX_IN_SBR error code, 5573
ER_SERVER_HANDLER_ERROR error code, 5563
ER_SERVER_INIT_COMPILED_IN_COMMANDS error code, 5340
ER_SERVER_ISNT_AVAILABLE error code, 5170
ER_SERVER_IS_IN_SECURE_AUTH_MODE error code, 5117
ER_SERVER_MASTER_FATAL_ERROR_READING_BINLOG error code, 5561
ER_SERVER_NET_PACKET_TOO_LARGE error code, 5562
ER_SERVER_NEW_ABORTING_CONNECTION error code, 5562
ER_SERVER_NOT_FORM_FILE error code, 5563
ER_SERVER_NO_SESSION_TO_SEND_TO error code, 5562
ER_SERVER_NO_SYSTEM_TABLE_ACCESS error code, 5562
ER_SERVER_OFFLINE_MODE error code, 5160
ER_SERVER_OUTOFMEMORY error code, 5325
ER_SERVER_OUT_OF_RESOURCES error code, 5325
ER_SERVER_OUT_OF_SORTMEMORY error code, 5562
ER_SERVER_RECORD_FILE_FULL error code, 5562
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_ENCRYPT error code, 5584
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_FETCH_KEY error code, 5579
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_GENERATE_KEY error code, 5579
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_REMOVE_KEY error code, 5580

ER_SERVER_RPL_ENCRYPTION_FAILED_TO_ROTATE_LOGS error code, 5579
ER_SERVER_RPL_ENCRYPTION_FAILED_TO_STORE_KEY error code, 5580
ER_SERVER_RPL_ENCRYPTION_HEADER_ERROR error code, 5579
ER_SERVER_RPL_ENCRYPTION_IGNORE_ROTATE_MASTER_KEY_AT_STARTUP error code, 5580
ER_SERVER_RPL_ENCRYPTION_KEYRING_INVALID_KEY error code, 5579
ER_SERVER_RPL_ENCRYPTION_KEY_EXISTS_UNEXPECTED error code, 5579
ER_SERVER_RPL_ENCRYPTION_KEY_NOT_FOUND error code, 5579
ER_SERVER_RPL_ENCRYPTION_MASTER_KEY_RECOVERY_FAILED error code, 5580
ER_SERVER_RPL_ENCRYPTION_UNABLE_TO_INITIALIZE error code, 5580
ER_SERVER_RPL_ENCRYPTION_UNABLE_TO_ROTATE_MASTER_KEY_AT_STARTUP error code, 5580
ER_SERVER_SHUTDOWN error code, 5102
ER_SERVER_SHUTDOWN_COMPLETE error code, 5323
ER_SERVER_SHUTDOWN_INFO error code, 5567
ER_SERVER_SLAVE_CONVERSION_FAILED error code, 5564
ER_SERVER_SLAVE_IGNORED_TABLE error code, 5564
ER_SERVER_SLAVE_INIT_QUERY_FAILED error code, 5564
ER_SERVER_SLAVE_MI_INIT_REPOSITORY error code, 5562
ER_SERVER_SLAVE_RLI_INIT_REPOSITORY error code, 5562
ER_SERVER_STARTUP_ADMIN_INTERFACE error code, 5580
ER_SERVER_STARTUP_MSG error code, 5325
ER_SERVER_TABLE_CHECK_FAILED error code, 5563
ER_SERVER_TEST_MESSAGE error code, 5564
ER_SERVER_UNKNOWN_ERROR error code, 5562
ER_SERVER_UNKNOWN_SYSTEM_VARIABLE error code, 5562
ER_SERVER_UPGRADE_CHECKING_DB error code, 5591
ER_SERVER_UPGRADE_EMPTY_SYS error code, 5591
ER_SERVER_UPGRADE_FAILED error code, 5590
ER_SERVER_UPGRADE_FROM_VERSION error code, 5584
ER_SERVER_UPGRADE_HELP_TABLE_STATUS error code, 5592
ER_SERVER_UPGRADE_INFO_FILE error code, 5590
ER_SERVER_UPGRADE_MYSQL_TABLES error code, 5590
ER_SERVER_UPGRADE_NO_SYS_VERSION error code, 5591
ER_SERVER_UPGRADE_OFF error code, 5589
ER_SERVER_UPGRADE_PENDING error code, 5590
ER_SERVER_UPGRADE_REPAIR_REQUIRED error code, 5590
ER_SERVER_UPGRADE_REPAIR_STATUS error code, 5590
ER_SERVER_UPGRADE_SKIP error code, 5590
ER_SERVER_UPGRADE_STATUS error code, 5590
ER_SERVER_UPGRADE_SYSTEM_TABLES error code, 5591
ER_SERVER_UPGRADE_SYS_SCHEMA error code, 5590
ER_SERVER_UPGRADE_SYS_SCHEMA_OBJECT_COUNT error code, 5591
ER_SERVER_UPGRADE_SYS_SCHEMA_OUTDATED error code, 5591
ER_SERVER_UPGRADE_SYS_SCHEMA_UP_TO_DATE error code, 5591
ER_SERVER_UPGRADE_SYS_VERSION_EMPTY error code, 5591
ER_SERVER_UPGRADE_VERSION_NOT_SUPPORTED error code, 5584
ER_SERVER_WARN_DEPRECATED error code, 5588
ER_SERVER_WRONG_VALUE_FOR_VAR error code, 5573
ER_SESSION_WAS_KILLED error code, 5170
ER_SETNS_FAILED error code, 5593
ER_SET_CONSTANTS_ONLY error code, 5112
ER_SET_EVENT_FAILED error code, 5568
ER_SET_PASSWORD_AUTH_PLUGIN error code, 5143
ER_SET_STATEMENT_CANNOT_INVOKE_FUNCTION error code, 5148
ER_SE_TYPECODE_CONFLICT error code, 5245

ER_SHA256_PASSWORD_SECOND_PASSWORD_USED_INFORMATION error code, 5581
ER_SHARED_TABLESPACE_USED_BY_PARTITIONED_TABLE error code, 5574
ER_SHA_PWD_AUTH_REQUIRES_RSA_OR_SSL error code, 5361
ER_SHA_PWD_FAILED_TO_GENERATE_MULTI_ROUND_HASH error code, 5361
ER_SHA_PWD_FAILED_TO_PARSE_AUTH_STRING error code, 5361
ER_SHA_PWD_RSA_KEY_TOO_LONG error code, 5361
ER_SHA_PWD_SALT_FOR_USER_CORRUPT error code, 5362
ER_SHUTDOWN_COMPLETE error code, 5104
ER_SHUTTING_DOWN_SLAVE_THREADS error code, 5232
ER_SIGNAL_BAD_CONDITION_TYPE error code, 5139
ER_SIGNAL_EXCEPTION error code, 5139
ER_SIGNAL_NOT_FOUND error code, 5139
ER_SIGNAL_WARN error code, 5139
ER_SIZE_OVERFLOW_ERROR error code, 5133
ER_SKIP_UPDATING_METADATA_IN_SE_RO_MODE error code, 5335
ER_SLAVE_CANT_CREATE_CONVERSION error code, 5142
ER_SLAVE_CANT_USE_TEMPDIR error code, 5287
ER_SLAVE_CHANGE_MASTER_TO_EXECUTED error code, 5286
ER_SLAVE_CHANNEL_DOES_NOT_EXIST error code, 5163
ER_SLAVE_CHANNEL_IO_THREAD_MUST_STOP error code, 5159
ER_SLAVE_CHANNEL_MUST_STOP error code, 5163
ER_SLAVE_CHANNEL_NAME_INVALID_OR_TOO_LONG error code, 5163
ER_SLAVE_CHANNEL_NOT_RUNNING error code, 5163
ER_SLAVE_CHANNEL_OPERATION_NOT_ALLOWED error code, 5168
ER_SLAVE_CHANNEL_SQL_SKIP_COUNTER error code, 5164
ER_SLAVE_CHANNEL_SQL_THREAD_MUST_STOP error code, 5164
ER_SLAVE_CHANNEL_WAS_NOT_RUNNING error code, 5163
ER_SLAVE_CHANNEL_WAS_RUNNING error code, 5163
ER_SLAVE_CONFIGURATION error code, 5150
ER_SLAVE_CONVERSION_FAILED error code, 5142
ER_SLAVE_CORRUPT_EVENT error code, 5137
ER_SLAVE_CREATE_EVENT_FAILURE error code, 5561
ER_SLAVE_FATAL_ERROR error code, 5561
ER_SLAVE_HAS_MORE_GTIDS_THAN_MASTER error code, 5157
ER_SLAVE_HEARTBEAT_FAILURE error code, 5561
ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE error code, 5138
ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MAX error code, 5144
ER_SLAVE_HEARTBEAT_VALUE_OUT_OF_RANGE_MIN error code, 5143
ER_SLAVE_IGNORED_SSL_PARAMS error code, 5117
ER_SLAVE_IGNORED_TABLE error code, 5114
ER_SLAVE_IGNORE_SERVER_IDS error code, 5140
ER_SLAVE_INCIDENT error code, 5561
ER_SLAVE_KILLED_AFTER_RECONNECT error code, 5336
ER_SLAVE_MASTER_COM_FAILURE error code, 5561
ER_SLAVE_MAX_CHANNELS_EXCEEDED error code, 5163
ER_SLAVE_MI_INIT_REPOSITORY error code, 5156
ER_SLAVE_MULTIPLE_CHANNELS_CMD error code, 5163
ER_SLAVE_NEW_CHANNEL_WRONG_REPOSITORY error code, 5163
ER_SLAVE_NOT_RUNNING error code, 5112
ER_SLAVE_NOT_STARTED_ON_SOME_CHANNELS error code, 5336
ER_SLAVE_POSSIBLY_DIVERGED_AFTER_DDL error code, 5196
ER_SLAVE_RECONNECT_FAILED error code, 5336
ER_SLAVE_RELAY_LOG_PURGE_FAILED error code, 5328
ER_SLAVE_RELAY_LOG_READ_FAILURE error code, 5561

ER_SLAVE_RELAY_LOG_TRUNCATE_INFO error code, 5328
ER_SLAVE_RELAY_LOG_WRITE_FAILURE error code, 5561
ER_SLAVE_RLI_INIT_REPOSITORY error code, 5156
ER_SLAVE_SILENT_RETRY_TRANSACTION error code, 5151
ER_SLAVE_SQL_THREAD_MUST_STOP error code, 5159
ER_SLAVE_SQL_THREAD_STOPPED_AFTER_GTIDS_REACHED error code, 5260
ER_SLAVE_SQL_THREAD_STOPPED_BEFORE_GTIDS_ALREADY_APPLIED error code, 5260
ER_SLAVE_SQL_THREAD_STOPPED_BEFORE_GTIDS_REACHED error code, 5260
ER_SLAVE_SQL_THREAD_STOPPED_GAP_TRX_PROCESSED error code, 5261
ER_SLAVE_SQL_THREAD_STOPPED_UNTIL_CONDITION_BAD error code, 5260
ER_SLAVE_SQL_THREAD_STOPPED_UNTIL_POSITION_REACHED error code, 5260
ER_SLAVE_THREAD error code, 5112
ER_SLAVE_WORKER_STOPPED_PREVIOUS_THD_ERROR error code, 5160
ER_SLOW_LOG_MODE_IGNORED_WHEN_NOT_LOGGING_TO_FILE error code, 5204
ER_SPATIAL_CANT_HAVE_NULL error code, 5115
ER_SPATIAL_FUNCTIONAL_INDEX error code, 5200
ER_SPATIAL_MUST_HAVE_GEOM_COL error code, 5142
ER_SPATIAL_UNIQUE_INDEX error code, 5196
ER_SPECIFIC_ACCESS_DENIED_ERROR error code, 5114
ER_SP_ALREADY_EXISTS error code, 5119
ER_SP_BADRETURN error code, 5119
ER_SP_BADSELECT error code, 5119
ER_SP_BADSTATEMENT error code, 5119
ER_SP_BAD_CURSOR_QUERY error code, 5120
ER_SP_BAD_CURSOR_SELECT error code, 5120
ER_SP_BAD_SQLSTATE error code, 5125
ER_SP_BAD_VAR_SHADOW error code, 5128
ER_SP_CANT_ALTER error code, 5121
ER_SP_CANT_SET_AUTOCOMMIT error code, 5127
ER_SP_CASE_NOT_FOUND error code, 5121
ER_SP_COND_MISMATCH error code, 5120
ER_SP_CURSOR_AFTER_HANDLER error code, 5121
ER_SP_CURSOR_ALREADY_OPEN error code, 5120
ER_SP_CURSOR_MISMATCH error code, 5120
ER_SP_CURSOR_NOT_OPEN error code, 5120
ER_SP_DOES_NOT_EXIST error code, 5119
ER_SP_DROP_FAILED error code, 5119
ER_SP_DUP_COND error code, 5121
ER_SP_DUP_CURS error code, 5121
ER_SP_DUP_HANDLER error code, 5125
ER_SP_DUP_PARAM error code, 5120
ER_SP_DUP_VAR error code, 5121
ER_SP_FETCH_NO_DATA error code, 5120
ER_SP_LABEL_MISMATCH error code, 5119
ER_SP_LABEL_REDEFINE error code, 5119
ER_SP_LILABEL_MISMATCH error code, 5119
ER_SP_LOAD_FAILED error code, 5174
ER_SP_NORETURN error code, 5120
ER_SP_NORETURNEND error code, 5120
ER_SP_NOT_VAR_ARG error code, 5125
ER_SP_NO_AGGREGATE error code, 5128
ER_SP_NO_DROP_SP error code, 5122
ER_SP_NO_RECURSION error code, 5126
ER_SP_NO_RECURSIVE_CREATE error code, 5119

ER_SP_NO_RETSET error code, 5126
ER_SP_RECURSION_LIMIT error code, 5128
ER_SP_STORE_FAILED error code, 5119
ER_SP_SUBSELECT_NYI error code, 5121
ER_SP_UNDECLARED_VAR error code, 5120
ER_SP_UNINIT_VAR error code, 5119
ER_SP_VARCOND_AFTER_CURSHNDLR error code, 5121
ER_SP_WRONG_NAME error code, 5128
ER_SP_WRONG_NO_OF_ARGS error code, 5120
ER_SP_WRONG_NO_OF_FETCH_ARGS error code, 5120
ER_SQLTHREAD_WITH_SECURE_SLAVE error code, 5148
ER_SQL_AUTHOR_DEFAULT_ROLES_FAIL error code, 5244
ER_SQL_HA_READ_FAILED error code, 5242
ER_SQL_MODE_MERGED error code, 5168
ER_SQL_MODE_MERGED_WITH_STRICT_MODE error code, 5323
ER_SQL_SLAVE_SKIP_COUNTER_NOT_SETTABLE_IN_GTID_MODE error code, 5155
ER_SQL_USER_TABLE_ALTER_WARNING error code, 5245
ER_SQL_USER_TABLE_CREATE_WARNING error code, 5244
ER_SRS_ATTRIBUTE_STRING_TOO_LONG error code, 5195
ER_SRS_GEOGCS_INVALID_AXES error code, 5197
ER_SRS_ID_ALREADY_EXISTS error code, 5194
ER_SRS_INVALID_ANGULAR_UNIT error code, 5197
ER_SRS_INVALID_CHARACTER_IN_ATTRIBUTE error code, 5195
ER_SRS_INVALID_INVERSE_FLATTENING error code, 5197
ER_SRS_INVALID_PRIME_MERIDIAN error code, 5197
ER_SRS_INVALID_SEMI_MAJOR_AXIS error code, 5197
ER_SRS_MISSING_MANDATORY_ATTRIBUTE error code, 5194
ER_SRS_MULTIPLE_ATTRIBUTE_DEFINITIONS error code, 5194
ER_SRS_NAME_CANT_BE_EMPTY_OR_WHITESPACE error code, 5194
ER_SRS_NOT_CARTESIAN error code, 5174
ER_SRS_NOT_CARTESIAN_UNDEFINED error code, 5175
ER_SRS_NOT_FOUND error code, 5177
ER_SRS_NOT_GEOGRAPHIC error code, 5196
ER_SRS_ORGANIZATION_CANT_BE_EMPTY_OR_WHITESPACE error code, 5194
ER_SRS_PARSE_ERROR error code, 5174
ER_SRS_PROJ_PARAMETER_MISSING error code, 5174
ER_SR_BOGUS_VALUE error code, 5242
ER_SR_INVALID_CONTEXT error code, 5242
ER_SSL_FIPS_MODE_ERROR error code, 5363
ER_SSL_LIBRARY_ERROR error code, 5227
ER_SSL_MEMORY_INSTRUMENTATION_INIT_FAILED error code, 5575
ER_SSL_SERVER_CERT_VERIFY_FAILED error code, 5593
ER_SSL_TRYING_DATADIR_DEFAULTS error code, 5239
ER_STACKSIZE_UNEXPECTED error code, 5238
ER_STACK_OVERRUN error code, 5107
ER_STACK_OVERRUN_NEED_MORE error code, 5127
ER_STARTING_AS error code, 5232
ER_STARTING_INIT error code, 5567
ER_STARTUP error code, 5125
ER_STD_BAD_ALLOC_ERROR error code, 5161
ER_STD_DOMAIN_ERROR error code, 5161
ER_STD_INVALID_ARGUMENT error code, 5161
ER_STD_LENGTH_ERROR error code, 5161
ER_STD_LOGIC_ERROR error code, 5161

ER_STD_OUT_OF_RANGE_ERROR error code, 5161
ER_STD_OVERFLOW_ERROR error code, 5161
ER_STD_RANGE_ERROR error code, 5161
ER_STD_REGEX_ERROR error code, 5210
ER_STD_RUNTIME_ERROR error code, 5161
ER_STD_UNDERFLOW_ERROR error code, 5161
ER_STD_UNKNOWN_EXCEPTION error code, 5161
ER_STMT_CACHE_FULL error code, 5144
ER_STMT_HAS_NO_OPEN_CURSOR error code, 5126
ER_STMT_NOT_ALLOWED_IN_SF_OR_TRG error code, 5121
ER_STOP_SLAVE_IO_THREAD_DISK_SPACE error code, 5363
ER_STOP_SLAVE_IO_THREAD_TIMEOUT error code, 5156
ER_STOP_SLAVE_SQL_THREAD_TIMEOUT error code, 5156
ER_STORAGE_ENGINE_NOT_LOADED error code, 5158
ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_DIRECT error code, 5142
ER_STORED_FUNCTION_PREVENTS_SWITCH_BINLOG_FORMAT error code, 5134
ER_STORED_FUNCTION_PREVENTS_SWITCH_SQL_LOG_BIN error code, 5143
ER_SUBPARTITION_ERROR error code, 5131
ER_SUBPARTITION_NAME error code, 5138
ER_SUBQUERY_NO_1_ROW error code, 5115
ER_SWITCH_TMP_ENGINE error code, 5179
ER_SYNTAX_ERROR error code, 5109
ER_SYSTEMD_NOTIFY_CONNECT_FAILED error code, 5427
ER_SYSTEMD_NOTIFY_PATH_TOO_LONG error code, 5427
ER_SYSTEMD_NOTIFY_WRITE_FAILED error code, 5427
ER_SYSTEM_SCHEMA_NOT_FOUND error code, 5221
ER_SYSTEM_TABLES_NOT_SUPPORTED_BY_STORAGE_ENGINE error code, 5325
ER_SYSTEM_TABLE_NOT_TRANSACTIONAL error code, 5302
ER_SYSTEM_VIEW_INIT_FAILED error code, 5326
ER_SYSVAR_CHANGE_DURING_QUERY error code, 5217
ER_SYS_VAR_COMPONENT_FAILED_TO_MAKE_VARIABLE_PERSISTENT error code, 5363
ER_SYS_VAR_COMPONENT_FAILED_TO_PARSE_VARIABLE_OPTIONS error code, 5363
ER_SYS_VAR_COMPONENT_OOM error code, 5362
ER_SYS_VAR_COMPONENT_UNKNOWN_VARIABLE_TYPE error code, 5362
ER_SYS_VAR_COMPONENT_VARIABLE_SET_READ_ONLY error code, 5362
ER_SYS_VAR_NOT_FOUND error code, 5568
ER_TABLEACCESS_DENIED_ERROR error code, 5108
ER_TABLENAME_NOT_ALLOWED_HERE error code, 5115
ER_TABLESPACE_AUTO_EXTEND_ERROR error code, 5133
ER_TABLESPACE_CANNOT_BE_DECRYPTED error code, 5208
ER_TABLESPACE_CANNOT_BE_ENCRYPTED error code, 5208
ER_TABLESPACE_CANNOT_ENCRYPT error code, 5171
ER_TABLESPACE_DISCARDED error code, 5152
ER_TABLESPACE_DUP_FILENAME error code, 5183
ER_TABLESPACE_ENGINE_MISMATCH error code, 5187
ER_TABLESPACE_EXISTS error code, 5152
ER_TABLESPACE_IS_NOT_EMPTY error code, 5167
ER_TABLESPACE_MISSING error code, 5152
ER_TABLESPACE_MISSING_WITH_NAME error code, 5174
ER_TABLESPACE_TYPE_UNKNOWN error code, 5208
ER_TABLES_DIFFERENT_METADATA error code, 5146
ER_TABLE_CANT_HANDLE_AUTO_INCREMENT error code, 5110
ER_TABLE_CANT_HANDLE_BLOB error code, 5110
ER_TABLE_CANT_HANDLE_FT error code, 5113

ER_TABLE_CANT_HANDLE_SPKEYS error code, 5129
ER_TABLE_CHECK_INTACT error code, 5240
ER_TABLE_CORRUPT error code, 5156
ER_TABLE_CREATED_WITH_DIFFERENT_VERSION error code, 5257
ER_TABLE_DEF_CHANGED error code, 5125
ER_TABLE_EXISTS_ERROR error code, 5101
ER_TABLE_HAS_NO_FT error code, 5148
ER_TABLE_INCOMPATIBLE_DECIMAL_FIELD error code, 5256
ER_TABLE_INCOMPATIBLE_YEAR_FIELD error code, 5256
ER_TABLE_IN_FK_CHECK error code, 5145
ER_TABLE_IN_SYSTEM_TABLESPACE error code, 5151
ER_TABLE_MUST_HAVE_COLUMNS error code, 5106
ER_TABLE_NAME error code, 5138
ER_TABLE_NAME_CAUSES_TOO_LONG_PATH error code, 5217
ER_TABLE_NEEDS_DUMP_UPGRADE error code, 5343
ER_TABLE_NEEDS_REBUILD error code, 5144
ER_TABLE_NEEDS_UPGRADE error code, 5128
ER_TABLE_NOT_LOCKED error code, 5105
ER_TABLE_NOT_LOCKED_FOR_WRITE error code, 5105
ER_TABLE_REFERENCED error code, 5172
ER_TABLE_SCHEMA_MISMATCH error code, 5151
ER_TABLE_UPGRADE_REQUIRED error code, 5324
ER_TABLE_WITHOUT_PK error code, 5199
ER_TABLE_WRONG_KEY_DEFINITION error code, 5257
ER_TARGET_TABLESPACE_UNENCRYPTED error code, 5208
ER_TARGET_TS_UNENCRYPTED error code, 5569
ER_TC_BAD_MAGIC_IN_TC_LOG error code, 5277
ER_TC_CANT_AUTO_RECOVER_WITH_TC_HEURISTIC_RECOVER error code, 5277
ER_TC_HEURISTIC_RECOVERY_FAILED error code, 5278
ER_TC_HEURISTIC_RECOVERY_MODE error code, 5278
ER_TC_NEED_N_SE_SUPPORTING_2PC_FOR_RECOVERY error code, 5278
ER_TC_RECOVERING_AFTER_CRASH_USING error code, 5277
ER_TC_RECOVERY_FAILED_THESE_ARE_YOUR_OPTIONS error code, 5278
ER_TC_RESTART_WITHOUT_TC_HEURISTIC_RECOVER error code, 5278
ER_TEMPORARY_NAME error code, 5139
ER_TEMP_FILE_WRITE_FAILURE error code, 5156
ER_TEMP_TABLE_PREVENTS_SWITCH_GLOBAL_BINLOG_FORMAT error code, 5198
ER_TEMP_TABLE_PREVENTS_SWITCH_OUT_OF_RBR error code, 5134
ER_TEMP_TABLE_PREVENTS_SWITCH_SESSION_BINLOG_FORMAT error code, 5198
ER_TEXTFILE_NOT_READABLE error code, 5104
ER_TF_FORBIDDEN_JOIN_TYPE error code, 5189
ER_TF_MUST_HAVE_ALIAS error code, 5189
ER_THE_USER_ABIDES error code, 5235
ER_THREAD_HANDLING_OOM error code, 5229
ER_THREAD_POOL_ALGORITHM_INVALID error code, 5344
ER_THREAD_POOL_BUFFER_TOO_SMALL error code, 5346
ER_THREAD_POOL_CANNOT_SET_THREAD_SPECIFIC_DATA error code, 5345
ER_THREAD_POOL_CON_HANDLER_INIT_FAILED error code, 5345
ER_THREAD_POOL_FAILED_PROCESS_CONNECT_EVENT error code, 5346
ER_THREAD_POOL_FAILED_TO_CREATE_CONNECT_HANDLER_THD error code, 5346
ER_THREAD_POOL_FAILED_TO_CREATE_POOL error code, 5346
ER_THREAD_POOL_FAILED_TO_CREATE_THD_AND_AUTH_CONN error code, 5346
ER_THREAD_POOL_INIT_FAILED error code, 5345
ER_THREAD_POOL_INVALID_PRIO_KICKUP_TIMER error code, 5345

ER_THREAD_POOL_INVALID_STALL_LIMIT error code, 5345
ER_THREAD_POOL_LOW_LEVEL_REARM_FAILED error code, 5346
ER_THREAD_POOL_MAX_UNUSED_THREADS_INVALID error code, 5345
ER_THREAD_POOL_NOT_SUPPORTED_ON_PLATFORM error code, 5344
ER_THREAD_POOL_PFS_TABLES_ADD_FAILED error code, 5578
ER_THREAD_POOL_PFS_TABLES_INIT_FAILED error code, 5578
ER_THREAD_POOL_PLUGIN_STARTED error code, 5345
ER_THREAD_POOL_RATE_LIMITED_ERROR_MSGS error code, 5346
ER_THREAD_POOL_SIZE_TOO_HIGH error code, 5344
ER_THREAD_POOL_SIZE_TOO_LOW error code, 5344
ER_THREAD_PRIORITY_IGNORED error code, 5322
ER_TOO_BIG_DISPLAYWIDTH error code, 5127
ER_TOO_BIG_ENUM error code, 5173
ER_TOO_BIG_FIELDLENGTH error code, 5103
ER_TOO_BIG_FOR_UNCOMPRESS error code, 5115
ER_TOO_BIG_PRECISION error code, 5126
ER_TOO_BIG_ROWSIZE error code, 5106
ER_TOO_BIG_SCALE error code, 5126
ER_TOO_BIG_SELECT error code, 5105
ER_TOO_BIG_SET error code, 5105
ER_TOO_HIGH_LEVEL_OF_NESTING_FOR_SELECT error code, 5129
ER_TOO_LONG_BODY error code, 5127
ER_TOO_LONG_FIELD_COMMENT error code, 5138
ER_TOO_LONG_IDENT error code, 5102
ER_TOO_LONG_INDEX_COMMENT error code, 5142
ER_TOO_LONG_KEY error code, 5103
ER_TOO_LONG_ROUTINE_COMMENT error code, 5174
ER_TOO_LONG_SET_ENUM_VALUE error code, 5173
ER_TOO_LONG_STRING error code, 5110
ER_TOO_LONG_TABLESPACE_COMMENT error code, 5182
ER_TOO_LONG_TABLE_COMMENT error code, 5138
ER_TOO_LONG_TABLE_PARTITION_COMMENT error code, 5150
ER_TOO_MANY_CONCURRENT_CLONES error code, 5186
ER_TOO_MANY_CONCURRENT_TRXS error code, 5139
ER_TOO_MANY_FIELDS error code, 5106
ER_TOO_MANY_KEYS error code, 5103
ER_TOO_MANY_KEY_PARTS error code, 5103
ER_TOO_MANY_PARTITIONS_ERROR error code, 5131
ER_TOO_MANY_PARTITION_FUNC_FIELDS_ERROR error code, 5140
ER_TOO_MANY_ROWS error code, 5110
ER_TOO_MANY_STORAGE_ENGINES error code, 5245
ER_TOO_MANY_TABLES error code, 5106
ER_TOO_MANY_USER_CONNECTIONS error code, 5112
ER_TOO_MANY_VALUES_ERROR error code, 5140
ER_TRACK_VARIABLES_BOGUS error code, 5235
ER_TRANSACTION_ROLLBACK_DURING_COMMIT error code, 5165
ER_TRANSFORM_SOURCE_SRS_MISSING_TOWGS84 error code, 5198
ER_TRANSFORM_SOURCE_SRS_NOT_SUPPORTED error code, 5197
ER_TRANSFORM_TARGET_SRS_MISSING_TOWGS84 error code, 5198
ER_TRANSFORM_TARGET_SRS_NOT_SUPPORTED error code, 5198
ER_TRANSPORTS_WHAT_TRANSPORTS error code, 5233
ER_TRANS_CACHE_FULL error code, 5112
ER_TREE_CORRUPT_2_CONSECUTIVE_REDS error code, 5300
ER_TREE_CORRUPT_INCORRECT_BLACK_COUNT error code, 5300

ER_TREE_CORRUPT_PARENT_SHOULD_POINT_AT_PARENT error code, 5300
ER_TREE_CORRUPT_RIGHT_IS_LEFT error code, 5300
ER_TREE_CORRUPT_ROOT_SHOULD_BE_BLACK error code, 5300
ER_TRG_ALREADY_EXISTS error code, 5122
ER_TRG_CANT_CHANGE_ROW error code, 5122
ER_TRG_CANT_OPEN_TABLE error code, 5137
ER_TRG_CANT_PARSE error code, 5240
ER_TRG_CORRUPTED_FILE error code, 5137
ER_TRG_CREATION_CTX_NOT_SET error code, 5324
ER_TRG_DOES_NOT_EXIST error code, 5122
ER_TRG_INVALID_CREATION_CTX error code, 5137
ER_TRG_IN_WRONG_SCHEMA error code, 5127
ER_TRG_NO_CLIENT_CHARSET error code, 5226
ER_TRG_NO_CREATION_CTX error code, 5137
ER_TRG_NO_DEFINER error code, 5128
ER_TRG_NO_SUCH_ROW_IN_TRG error code, 5123
ER_TRG_ON_VIEW_OR_TEMP_TABLE error code, 5122
ER_TRG_WITHOUT_DEFINER error code, 5226
ER_TRG_WRONG_ORDER error code, 5585
ER_TRHEAD_POOL_LOW_LEVEL_INIT_FAILED error code, 5346
ER_TRIGGER_INVALID_VALUE error code, 5220
ER_TRUNCATED_WRONG_VALUE error code, 5118
ER_TRUNCATED_WRONG_VALUE_FOR_FIELD error code, 5123
ER_TRUNCATE_ILLEGAL_FK error code, 5143
ER_TRX_GTID_COLLECT_REJECT error code, 5244
ER_TRX_WRITE_SET_OOM error code, 5245
ER_TURNING_LOGGING_OFF_FOR_THE_DURATION error code, 5317
ER_TURNING_ON_PARTIAL_REVOKES error code, 5588
ER_TX_EXTRACTION_ALGORITHM_FOR_BINLOG_TX_DEPEDENCY_TRACKING error code, 5328
ER_TZ_CANT_BUILD_MKTIME_MAP error code, 5260
ER_TZ_CANT_FIND_DESCRIPTION_FOR_TIME_ZONE error code, 5258
ER_TZ_CANT_FIND_DESCRIPTION_FOR_TIME_ZONE_ID error code, 5259
ER_TZ_CANT_OPEN_AND_LOCK_TIME_ZONE_TABLE error code, 5258
ER_TZ_ERROR_LOADING_LEAP_SECOND_TABLE error code, 5258
ER_TZ_NO_TRANSITION_TYPES_IN_TIME_ZONE error code, 5259
ER_TZ_OOM_INITIALIZING_TIME_ZONES error code, 5258
ER_TZ_OOM_LOADING_LEAP_SECOND_TABLE error code, 5258
ER_TZ_OOM_LOADING_TIME_ZONE_DESCRIPTION error code, 5260
ER_TZ_OOM_WHILE_LOADING_TIME_ZONE error code, 5260
ER_TZ_OOM_WHILE_SETTING_TIME_ZONE error code, 5260
ER_TZ_TOO_MANY_LEAPS_IN_LEAP_SECOND_TABLE error code, 5258
ER_TZ_TRANSITION_TABLE_BAD_TRANSITION_TYPE error code, 5259
ER_TZ_TRANSITION_TABLE_LOAD_ERROR error code, 5259
ER_TZ_TRANSITION_TABLE_TOO_MANY_TRANSITIONS error code, 5259
ER_TZ_TRANSITION_TYPE_TABLE_ABBREVIATIONS_EXCEED_SPACE error code, 5259
ER_TZ_TRANSITION_TYPE_TABLE_LOAD_ERROR error code, 5259
ER_TZ_TRANSITION_TYPE_TABLE_TYPE_TOO_LARGE error code, 5259
ER_TZ_UNKNOWN_OR_ILLEGAL_DEFAULT_TIME_ZONE error code, 5258
ER_UDF_ALREADY_EXISTS error code, 5567
ER_UDF_CANT_ALLOC_FOR_FUNCTION error code, 5243
ER_UDF_CANT_ALLOC_FOR_STRUCTURES error code, 5243
ER_UDF_CANT_OPEN_FUNCTION_TABLE error code, 5243
ER_UDF_DROP_DYNAMICALLY_REGISTERED error code, 5184
ER_UDF_ERROR error code, 5173

ER_UDF_EXISTS error code, 5107
ER_UDF_INVALID_ROW_IN_FUNCTION_TABLE error code, 5243
ER_UDF_NO_PATHS error code, 5107
ER_UKNOWN_AUTH_ID_OR_ACCESS_DENIED_FOR_GRANT_AS error code, 5209
ER_UNABLE_TO_BUILD_HISTOGRAM error code, 5185
ER_UNABLE_TO_COLLECT_LOG_STATUS error code, 5195
ER_UNABLE_TO_DROP_COLUMN_STATISTICS error code, 5185
ER_UNABLE_TO_RESOLVE_HOSTNAME error code, 5226
ER_UNABLE_TO_RESOLVE_IP error code, 5225
ER_UNABLE_TO_SET_OPTION error code, 5196
ER_UNABLE_TO_STORE_COLUMN_STATISTICS error code, 5184
ER_UNABLE_TO_STORE_STATISTICS error code, 5177
ER_UNABLE_TO_UPDATE_COLUMN_STATISTICS error code, 5185
ER_UNDISCLOSED_PARSE_ERROR_IN_DIGEST_FN error code, 5190
ER_UNDO_RECORD_TOO_BIG error code, 5144
ER_UNEXPECTED_GEOMETRY_TYPE error code, 5174
ER_UNIQUE_KEY_NEED_ALL_FIELDS_IN_PF error code, 5131
ER_UNIT_NOT_FOUND error code, 5216
ER_UNKNOWN_ALTER_ALGORITHM error code, 5151
ER_UNKNOWN_ALTER_LOCK error code, 5151
ER_UNKNOWN_AUTHID error code, 5175
ER_UNKNOWN_AUTH_ID_IN_MANDATORY_ROLE error code, 5329
ER_UNKNOWN_CHARACTER_SET error code, 5106
ER_UNKNOWN_COLLATION error code, 5117
ER_UNKNOWN_COM_ERROR error code, 5101
ER_UNKNOWN_ERROR error code, 5106
ER_UNKNOWN_ERROR_DETECTED_IN_SE error code, 5335
ER_UNKNOWN_ERROR_NUMBER error code, 5243
ER_UNKNOWN_EXPLAIN_FORMAT error code, 5150
ER_UNKNOWN_KEY_CACHE error code, 5117
ER_UNKNOWN_LOCALE error code, 5139
ER_UNKNOWN_NETWORK_NAMESPACE error code, 5593
ER_UNKNOWN_PARTITION error code, 5146
ER_UNKNOWN_PROCEDURE error code, 5106
ER_UNKNOWN_STMT_HANDLER error code, 5115
ER_UNKNOWN_STORAGE_ENGINE error code, 5118
ER_UNKNOWN_SYSTEM_VARIABLE error code, 5111
ER_UNKNOWN_TABLE error code, 5106
ER_UNKNOWN_TABLESPACE_TYPE error code, 5574
ER_UNKNOWN_TABLE_IN_UPGRADE error code, 5599
ER_UNKNOWN_TARGET_BINLOG error code, 5123
ER_UNKNOWN_TIME_ZONE error code, 5118
ER_UNKNOWN_UNSUPPORTED_STORAGE_ENGINE error code, 5228
ER_UNKNOWN_VARIABLE_IN_PERSISTED_CONFIG_FILE error code, 5568
ER_UNRESOLVED_HINT_NAME error code, 5167
ER_UNRESOLVED_TABLE_LOCK error code, 5178
ER_UNSUPORTED_LOG_ENGINE error code, 5135
ER_UNSUPPORTED_ACTION_ON_DEFAULT_VAL_GENERATED error code, 5201
ER_UNSUPPORTED_ACTION_ON_GENERATED_COLUMN error code, 5165
ER_UNSUPPORTED_ALTER_ENCRYPTION_INPLACE error code, 5172
ER_UNSUPPORTED_ALTER_INPLACE_ON_VIRTUAL_COLUMN error code, 5165
ER_UNSUPPORTED_ALTER_ONLINE_ON_VIRTUAL_COLUMN error code, 5171
ER_UNSUPPORTED_DATE error code, 5232
ER_UNSUPPORTED_ENGINE error code, 5145

ER_UNSUPPORTED_EXTENSION error code, 5106
ER_UNSUPPORTED_INDEX_ALGORITHM error code, 5173
ER_UNSUPPORTED_PS error code, 5118
ER_UNSUPPORTED_SQL_MODE error code, 5215
ER_UNSUPPORTED_USE_OF_GRANT_AS error code, 5209
ER_UNSUPPORT_COMPRESSED_TEMPORARY_TABLE error code, 5173
ER_UNTIL_COND_IGNORED error code, 5117
ER_UPDATE_GTID_PURGED_WITH_GR error code, 5217
ER_UPDATE_INFO error code, 5108
ER_UPDATE_LOG_DEPRECATED_IGNORED error code, 5119
ER_UPDATE_LOG_DEPRECATED_TRANSLATED error code, 5120
ER_UPDATE_TABLE_USED error code, 5105
ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE error code, 5110
ER_UPDATING_DD_TABLE error code, 5174
ER_UPGRADE_PARSE_ERROR error code, 5573
ER_UPGRADE_WITH_PARTITIONED_TABLES_REJECTED error code, 5601
ER_USERNAME error code, 5129
ER_USER_ALREADY_EXISTS error code, 5170
ER_USER_COLUMN_OLD_LENGTH error code, 5172
ER_USER_DOES_NOT_EXIST error code, 5170
ER_USER_LIMIT_REACHED error code, 5114
ER_USER_LOCK_DEADLOCK error code, 5162
ER_USER_LOCK_WRONG_NAME error code, 5162
ER_USER_NOT_IN_EXTRA_USERS_BINLOG_POSSIBLY_INCOMPLETE error code, 5255
ER_USER_REQUIRES_ROOT error code, 5233
ER_USER_WHAT_USER error code, 5233
ER_UUID_INVALID error code, 5227
ER_UUID_IS error code, 5227
ER_UUID_SALT error code, 5227
ER_UUID_SCRUB error code, 5227
ER_VALGRIND_COUNT_LEAKS error code, 5242
ER_VALGRIND_DO_QUICK_LEAK_CHECK error code, 5242
ER_VALIDATE_PWD_CONVERT_TO_BUFFER_FAILED error code, 5429
ER_VALIDATE_PWD_COULD_BE_NULL error code, 5428
ER_VALIDATE_PWD_DICT_FILE_NOT_LOADED error code, 5359
ER_VALIDATE_PWD_DICT_FILE_NOT_SPECIFIED error code, 5359
ER_VALIDATE_PWD_DICT_FILE_OPEN_FAILED error code, 5428
ER_VALIDATE_PWD_DICT_FILE_TOO_BIG error code, 5359
ER_VALIDATE_PWD_FAILED_TO_GET_FLD_FROM_SECURITY_CTX error code, 5360
ER_VALIDATE_PWD_FAILED_TO_GET_SECURITY_CTX error code, 5360
ER_VALIDATE_PWD_FAILED_TO_READ_DICT_FILE error code, 5359
ER_VALIDATE_PWD_LENGTH_CHANGED error code, 5360
ER_VALIDATE_PWD_STATUS_VAR_REGISTRATION_FAILED error code, 5428
ER_VALIDATE_PWD_STATUS_VAR_UNREGISTRATION_FAILED error code, 5428
ER_VALIDATE_PWD_STRING_CONV_TO_BUFFER_FAILED error code, 5429
ER_VALIDATE_PWD_STRING_CONV_TO_LOWERCASE_FAILED error code, 5428
ER_VALIDATE_PWD_STRING_HANDLER_MEM_ALLOCATION_FAILED error code, 5429
ER_VALIDATE_PWD_STRONG_POLICY_DICT_FILE_UNSPECIFIED error code, 5429
ER_VALIDATE_PWD_VARIABLE_REGISTRATION_FAILED error code, 5429
ER_VALIDATE_PWD_VARIABLE_UNREGISTRATION_FAILED error code, 5429
ER_VALUES_IS_NOT_INT_TYPE_ERROR error code, 5143
ER_VARIABLE_IS_NOT_STRUCT error code, 5117
ER_VARIABLE_IS_READONLY error code, 5138
ER_VARIABLE_NOT_PERSISTED error code, 5177

ER_VARIABLE_NOT_SETTABLE_IN_SF_OR_TRIGGER error code, 5148
ER_VARIABLE_NOT_SETTABLE_IN_SP error code, 5154
ER_VARIABLE_NOT_SETTABLE_IN_TRANSACTION error code, 5148
ER_VAR_CANT_BE_READ error code, 5114
ER_VAR_DOES_NOT_EXIST error code, 5184
ER_VERBOSE_HINT error code, 5235
ER_VERBOSE_REQUIRES_HELP error code, 5228
ER_VIEW_CHECKSUM error code, 5124
ER_VIEW_CHECK_FAILED error code, 5123
ER_VIEW_CREATION_CTX_NOT_SET error code, 5324
ER_VIEW_DELETE_MERGE_VIEW error code, 5124
ER_VIEW_FRM_NO_USER error code, 5128
ER_VIEW_INVALID error code, 5122
ER_VIEW_INVALID_CREATION_CTX error code, 5137
ER_VIEW_MULTIUPDATE error code, 5124
ER_VIEW_NONUPD_CHECK error code, 5123
ER_VIEW_NO_CREATION_CTX error code, 5136
ER_VIEW_NO_EXPLAIN error code, 5121
ER_VIEW_NO_INSERT_FIELD_LIST error code, 5124
ER_VIEW_OTHER_USER error code, 5128
ER_VIEW_PREVENT_UPDATE error code, 5127
ER_VIEW_RECURSIVE error code, 5129
ER_VIEW_SELECT_CLAUSE error code, 5122
ER_VIEW_SELECT_TMPTABLE error code, 5122
ER_VIEW_SELECT_VARIABLE error code, 5122
ER_VIEW_UNKNOWN_CHARSET_OR_COLLATION error code, 5222
ER_VIEW_UNPARSABLE error code, 5257
ER_VIEW_WRONG_LIST error code, 5122
ER_VTOKEN_PLUGIN_TOKEN_MISMATCH error code, 5168
ER_VTOKEN_PLUGIN_TOKEN_NOT_FOUND error code, 5168
ER_WAITPID_FAILED error code, 5340
ER_WARNING_AUTHCACHE_INVALID_USER_ATTRIBUTES error code, 5581
ER_WARNING_DISCARD_OLD_PASSWORD_CLAUSE_VOID error code, 5581
ER_WARNING_NOT_COMPLETE_ROLLBACK error code, 5112
ER_WARNING_NOT_COMPLETE_ROLLBACK_WITH_CREATED_TEMP_TABLE error code, 5147
ER_WARNING_NOT_COMPLETE_ROLLBACK_WITH_DROPPED_TEMP_TABLE error code, 5147
ER_WARNING_PASSWORD_HISTORY_CLAUSES_VOID error code, 5186
ER_WARNING_RETAIN_CURRENT_PASSWORD_CLAUSE_VOID error code, 5581
ER_WARN_ALLOWED_PACKET_OVERFLOWED error code, 5119
ER_WARN_AUTH_ID_WITH_SYSTEM_USER_PRIV_IN_MANDATORY_ROLES error code, 5603
ER_WARN_BAD_MAX_EXECUTION_TIME error code, 5167
ER_WARN_BINLOG_PARTIAL_UPDATES_DISABLED error code, 5187
ER_WARN_BINLOG_PARTIAL_UPDATES_SUGGESTS_PARTIAL_IMAGES error code, 5187
ER_WARN_BINLOG_V1_ROW_EVENTS_DISABLED error code, 5187
ER_WARN_CANT_DROP_DEFAULT_KEYCACHE error code, 5127
ER_WARN_CONFLICTING_HINT error code, 5167
ER_WARN_DATA_OUT_OF_RANGE error code, 5116
ER_WARN_DATA_OUT_OF_RANGE_FUNCTIONAL_INDEX error code, 5199
ER_WARN_DATA_TRUNCATED_FUNCTIONAL_INDEX error code, 5199
ER_WARN_DA_PRIVILEGE_NOT_REGISTERED error code, 5219
ER_WARN_DEPRECATED_FLOAT_AUTO_INCREMENT error code, 5211
ER_WARN_DEPRECATED_FLOAT_DIGITS error code, 5211
ER_WARN_DEPRECATED_FLOAT_UNSIGNED error code, 5211
ER_WARN_DEPRECATED_FOUND_ROWS error code, 5595

ER_WARN_DEPRECATED_INTEGER_DISPLAY_WIDTH error code, 5211
ER_WARN_DEPRECATED_NESTED_COMMENT_SYNTAX error code, 5202
ER_WARN_DEPRECATED_SQLMODE error code, 5164
ER_WARN_DEPRECATED_SQL_CALC_FOUND_ROWS error code, 5594
ER_WARN_DEPRECATED_SYNTAX error code, 5118
ER_WARN_DEPRECATED_SYNTAX_NO_REPLACEMENT error code, 5142
ER_WARN_DEPRECATED_SYNTAX_WITH_VER error code, 5134
ER_WARN_DEPRECATED_SYSVAR_UPDATE error code, 5164
ER_WARN_DEPRECATED_USER_SET_EXPR error code, 5202
ER_WARN_DEPRECATED_UTF8MB3_CHARSET_OPTION error code, 5574
ER_WARN_DEPRECATED_UTF8MB3_COLLATION error code, 5202
ER_WARN_DEPRECATED_UTF8MB3_COLLATION_OPTION error code, 5574
ER_WARN_DEPRECATED_UTF8_ALIAS_OPTION error code, 5574
ER_WARN_DEPRECATED_ZEROFILL error code, 5211
ER_WARN_ENGINE_TRANSACTION_ROLLBACK error code, 5138
ER_WARN_FIELD_RESOLVED error code, 5117
ER_WARN_FUNC_INDEX_NOT_APPLICABLE error code, 5216
ER_WARN_HOSTNAME_WONT_WORK error code, 5117
ER_WARN_INCORRECT_PRIVILEGE_FOR_DB_RESTRICTIONS error code, 5589
ER_WARN_INDEX_NOT_APPLICABLE error code, 5146
ER_WARN_INVALID_DB_RESTRICTIONS error code, 5589
ER_WARN_INVALID_HINT error code, 5183
ER_WARN_INVALID_TIMESTAMP error code, 5118
ER_WARN_I_S_SKIPPED_TABLE error code, 5142
ER_WARN_LEGACY_SYNTAX_CONVERTED error code, 5158
ER_WARN_LOG_PRIVILEGE_CHECKS_USER_CORRUPT error code, 5602
ER_WARN_LOG_PRIVILEGE_CHECKS_USER_DOES_NOT_EXIST error code, 5602
ER_WARN_LOG_PRIVILEGE_CHECKS_USER_NEEDS_RPL_APPLIER_PRIV error code, 5602
ER_WARN_NO_SERVERID_SPECIFIED error code, 5323
ER_WARN_NULL_TO_NOTNULL error code, 5116
ER_WARN_ONLY_MASTER_LOG_FILE_NO_POS error code, 5159
ER_WARN_ON_MODIFYING_GTID_EXECUTED_TABLE error code, 5167
ER_WARN_OPEN_TEMP_TABLES_MUST_BE_ZERO error code, 5159
ER_WARN_OPTIMIZER_HINT_SYNTAX_ERROR error code, 5167
ER_WARN_PARTIAL_REVOKE_AND_DB_GRANT error code, 5588
ER_WARN_PROPERTY_STRING_PARSE_FAILED error code, 5583
ER_WARN_PURGE_LOG_IN_USE error code, 5156
ER_WARN_PURGE_LOG_IS_ACTIVE error code, 5156
ER_WARN_QC_RESIZE error code, 5117
ER_WARN_REMOVED_SQL_MODE error code, 5575
ER_WARN_RESERVED_SRID_RANGE error code, 5195
ER_WARN_RPL_RECOVERY_NO_ROTATE_EVENT_FROM_MASTER_EOF error code, 5603
ER_WARN_SRS_ID_ALREADY_EXISTS error code, 5194
ER_WARN_SRS_NOT_FOUND error code, 5174
ER_WARN_SRS_NOT_FOUND_AXIS_ORDER error code, 5178
ER_WARN_TOO_FEW_RECORDS error code, 5116
ER_WARN_TOO_MANY_RECORDS error code, 5116
ER_WARN_TRIGGER_DOESNT_HAVE_CREATED error code, 5158
ER_WARN_UNKNOWN_KEYRING_AWS_REGION error code, 5602
ER_WARN_UNKNOWN_QB_NAME error code, 5167
ER_WARN_UNLOAD_THE_NOT_PERSISTED error code, 5176
ER_WARN_UNSUPPORTED_HINT error code, 5174
ER_WARN_UNSUPPORTED_MAX_EXECUTION_TIME error code, 5167
ER_WARN_USING_OTHER_HANDLER error code, 5116

ER_WARN_VIEW_MERGE error code, 5122
ER_WARN_VIEW_WITHOUT_KEY error code, 5122
ER_WARN_WRONG_COMPRESSION_ALGORITHM_LOG error code, 5601
ER_WARN_WRONG_COMPRESSION_LEVEL_LOG error code, 5601
ER_WASTEFUL_NET_BUFFER_SIZE error code, 5229
ER_WILDCARD_NOT_ALLOWED_FOR_MULTIADDRESS_BIND error code, 5593
ER_WINDOW_CIRCULARITY_IN_WINDOW_GRAPH error code, 5180
ER_WINDOW_DUPLICATE_NAME error code, 5181
ER_WINDOW_EXPLAIN_JSON error code, 5182
ER_WINDOW_FRAME_END_ILLEGAL error code, 5180
ER_WINDOW_FRAME_ILLEGAL error code, 5180
ER_WINDOW_FRAME_START_ILLEGAL error code, 5180
ER_WINDOW_FUNCTION_IGNORES_FRAME error code, 5182
ER_WINDOW_ILLEGAL_ORDER_BY error code, 5181
ER_WINDOW_INVALID_WINDOW_FUNC_ALIAS_USE error code, 5181
ER_WINDOW_INVALID_WINDOW_FUNC_USE error code, 5181
ER_WINDOW_NESTED_WINDOW_FUNC_USE_IN_WINDOW_SPEC error code, 5181
ER_WINDOW_NO_CHILD_PARTITIONING error code, 5180
ER_WINDOW_NO_GROUP_ORDER error code, 5181
ER_WINDOW_NO_GROUP_ORDER_UNUSED error code, 5182
ER_WINDOW_NO_INHERIT_FRAME error code, 5180
ER_WINDOW_NO_REDEFINE_ORDER_BY error code, 5180
ER_WINDOW_NO_SUCH_WINDOW error code, 5180
ER_WINDOW_RANGE_BOUND_NOT_CONSTANT error code, 5181
ER_WINDOW_RANGE_FRAME_NUMERIC_TYPE error code, 5181
ER_WINDOW_RANGE_FRAME_ORDER_TYPE error code, 5180
ER_WINDOW_RANGE_FRAME_TEMPORAL_TYPE error code, 5180
ER_WINDOW_ROWS_INTERVAL_USE error code, 5181
ER_WINDOW_SE_NOT_ACCEPTABLE error code, 5182
ER_WIN_LISTEN_BUT_HOW error code, 5233
ER_WIN_LOAD_LIBRARY_FAILED error code, 5340
ER_WL9236_NOW_UNUSED error code, 5182
ER_WRITABLE_CONFIG_REMOVED error code, 5231
ER_WRITE_ROW_TO_PARTITION_FAILED error code, 5329
ER_WRONG_ARGUMENTS error code, 5113
ER_WRONG_AUTO_KEY error code, 5103
ER_WRONG_COLUMN_NAME error code, 5110
ER_WRONG_COMPRESSION_ALGORITHM_CLIENT error code, 5218
ER_WRONG_COMPRESSION_LEVEL_CLIENT error code, 5218
ER_WRONG_COUNT_FOR_KEY error code, 5300
ER_WRONG_COUNT_FOR_ORIGIN error code, 5300
ER_WRONG_COUNT_OF_ELEMENTS error code, 5300
ER_WRONG_DATETIME_SPEC error code, 5231
ER_WRONG_DB_NAME error code, 5105
ER_WRONG_EXPR_IN_PARTITION_FUNC_ERROR error code, 5130
ER_WRONG_FIELD_SPEC error code, 5103
ER_WRONG_FIELD_TERMINATORS error code, 5104
ER_WRONG_FIELD_WITH_GROUP error code, 5102
ER_WRONG_FIELD_WITH_GROUP_V2 error code, 5164
ER_WRONG_FILE_NAME error code, 5167
ER_WRONG_FK_DEF error code, 5115
ER_WRONG_FK_OPTION_FOR_GENERATED_COLUMN error code, 5165
ER_WRONG_GROUP_FIELD error code, 5102
ER_WRONG_JSON_TABLE_VALUE error code, 5189

ER_WRONG_KEY_COLUMN error code, 5110
ER_WRONG_KEY_COLUMN_FUNCTIONAL_INDEX error code, 5200
ER_WRONG_LOCK_OF_SYSTEM_TABLE error code, 5126
ER_WRONG_MRG_TABLE error code, 5110
ER_WRONG_MVI_VALUE error code, 5216
ER_WRONG_NAME_FOR_CATALOG error code, 5117
ER_WRONG_NAME_FOR_INDEX error code, 5117
ER_WRONG_NATIVE_TABLE_STRUCTURE error code, 5142
ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT error code, 5113
ER_WRONG_OBJECT error code, 5122
ER_WRONG_OUTER_JOIN error code, 5107
ER_WRONG_OUTER_JOIN_UNUSED error code, 5107
ER_WRONG_PARAMCOUNT_TO_NATIVE_FCT error code, 5136
ER_WRONG_PARAMCOUNT_TO_PROCEDURE error code, 5106
ER_WRONG_PARAMETERS_TO_NATIVE_FCT error code, 5136
ER_WRONG_PARAMETERS_TO_PROCEDURE error code, 5106
ER_WRONG_PARAMETERS_TO_STORED_FCT error code, 5136
ER_WRONG_PARTITION_NAME error code, 5135
ER_WRONG_PERFSCHEMA_USAGE error code, 5142
ER_WRONG_SIZE_NUMBER error code, 5133
ER_WRONG_SPVAR_TYPE_IN_LIMIT error code, 5143
ER_WRONG_SRID_FOR_COLUMN error code, 5187
ER_WRONG_STRING_LENGTH error code, 5129
ER_WRONG_SUB_KEY error code, 5104
ER_WRONG_SUM_SELECT error code, 5102
ER_WRONG_TABLESPACE_NAME error code, 5166
ER_WRONG_TABLE_NAME error code, 5105
ER_WRONG_TYPE_COLUMN_VALUE_ERROR error code, 5140
ER_WRONG_TYPE_FOR_COLUMN_PREFIX_IDX_FLD error code, 5339
ER_WRONG_TYPE_FOR_VAR error code, 5114
ER_WRONG_USAGE error code, 5113
ER_WRONG_VALUE error code, 5133
ER_WRONG_VALUE_COUNT error code, 5102
ER_WRONG_VALUE_COUNT_ON_ROW error code, 5108
ER_WRONG_VALUE_FOR_TYPE error code, 5125
ER_WRONG_VALUE_FOR_VAR error code, 5114
ER_X509_CANT_CHMOD_KEY error code, 5251
ER_X509_CANT_CREATE_CERT error code, 5251
ER_X509_CANT_READ_CA_CERT error code, 5251
ER_X509_CANT_READ_CA_KEY error code, 5251
ER_X509_CANT_WRITE_CERT error code, 5251
ER_X509_CANT_WRITE_KEY error code, 5251
ER_X509_CIPHERS_MISMATCH error code, 5250
ER_X509_ISSUER_MISMATCH error code, 5250
ER_X509_NEEDS_RSA_PRIVKEY error code, 5251
ER_X509_SUBJECT_MISMATCH error code, 5250
ER_XAER_DUPID error code, 5127
ER_XAER_INVAL error code, 5124
ER_XAER_NOTA error code, 5124
ER_XAER_OUTSIDE error code, 5124
ER_XAER_RMERR error code, 5125
ER_XAER_RMFAIL error code, 5124
ER_XA_CANT_CREATE_MDL_BACKUP error code, 5600
ER_XA_COMMITTING_XID error code, 5244

ER_XA_IGNORING_XID error code, 5243
ER_XA_NO_MULTI_2PC_HEURISTIC_RECOVER error code, 5244
ER_XA_RBDEADLOCK error code, 5137
ER_XA_RBROLLBACK error code, 5125
ER_XA_RBTIMEOUT error code, 5137
ER_XA_RECOVERY_DONE error code, 5244
ER_XA_RECOVER_EXPLANATION error code, 5244
ER_XA_RECOVER_FOUND_TRX_IN_SE error code, 5243
ER_XA_RECOVER_FOUND_XA_TRX error code, 5243
ER_XA_REPLICATION_FILTERS error code, 5215
ER_XA_RETRY error code, 5172
ER_XA_ROLLING_BACK_XID error code, 5244
ER_XA_STARTING_RECOVERY error code, 5244
ER_XPLUGIN_ALL_IO_INTERFACES_DISABLED error code, 5368
ER_XPLUGIN_BUFFER_PAGE_ALLOC_FAILED error code, 5365
ER_XPLUGIN_CAPABILITY_CLIENT_INTERACTIVE_FAILED error code, 5372
ER_XPLUGIN_CAPABILITY_EXPIRED_PASSWORD error code, 5372
ER_XPLUGIN_CLIENT_KILL_MSG error code, 5370
ER_XPLUGIN_CLIENT_RELEASE_TRIGGERED error code, 5370
ER_XPLUGIN_CLIENT_SSL_HANDSHAKE_FAILED error code, 5367
ER_XPLUGIN_CLOSING_CLIENTS_ON_SHUTDOWN error code, 5368
ER_XPLUGIN_COMPRESSION_ERROR error code, 5601
ER_XPLUGIN_DETECTED_HANGING_CLIENTS error code, 5365
ER_XPLUGIN_EMPTY_ADMIN_CMD error code, 5371
ER_XPLUGIN_ERROR_MSG error code, 5361
ER_XPLUGIN_ERROR_READING_SOCKET error code, 5368
ER_XPLUGIN_ERROR_WRITING_TO_CLIENT error code, 5369
ER_XPLUGIN_EXCEPTION_IN_EVENT_LOOP error code, 5366
ER_XPLUGIN_EXCEPTION_IN_TASK_SCHEDULER error code, 5366
ER_XPLUGIN_EXISTING_USER_ACCOUNT_WITH_INCOMPLETE_GRANTS error code, 5370
ER_XPLUGIN_FAILED_AT_SSL_CONF error code, 5367
ER_XPLUGIN_FAILED_TO_ACCEPT_CLIENT error code, 5365
ER_XPLUGIN_FAILED_TO_CLOSE_SQL_SESSION error code, 5371
ER_XPLUGIN_FAILED_TO_CREATE_SESSION_FOR_CONN error code, 5367
ER_XPLUGIN_FAILED_TO_EXECUTE_ADMIN_CMD error code, 5371
ER_XPLUGIN_FAILED_TO_GET_CREATION_STMT error code, 5371
ER_XPLUGIN_FAILED_TO_GET_ENGINE_INFO error code, 5371
ER_XPLUGIN_FAILED_TO_GET_SECURITY_CTX error code, 5370
ER_XPLUGIN_FAILED_TO_GET_SYS_VAR error code, 5371
ER_XPLUGIN_FAILED_TO_INITIALIZE_SESSION error code, 5367
ER_XPLUGIN_FAILED_TO_INTERRUPT_SESSION error code, 5370
ER_XPLUGIN_FAILED_TO_OPEN_INTERNAL_SESSION error code, 5372
ER_XPLUGIN_FAILED_TO_PREPARE_IO_INTERFACES error code, 5366
ER_XPLUGIN_FAILED_TO_RESET_IPV6_V6ONLY_FLAG error code, 5372
ER_XPLUGIN_FAILED_TO_SCHEDULE_CLIENT error code, 5365
ER_XPLUGIN_FAILED_TO_SET_MIN_NUMBER_OF_WORKERS error code, 5367
ER_XPLUGIN_FAILED_TO_SET_SO_REUSEADDR_FLAG error code, 5372
ER_XPLUGIN_FAILED_TO_STOP_SERVICES error code, 5561
ER_XPLUGIN_FAILED_TO_SWITCH_CONTEXT error code, 5372
ER_XPLUGIN_FAILED_TO_SWITCH_SECURITY_CTX error code, 5573
ER_XPLUGIN_FAILED_TO_SWITCH_SECURITY_CTX_TO_ROOT error code, 5371
ER_XPLUGIN_FAILED_TO_UNREGISTER_UDF error code, 5372
ER_XPLUGIN_FAILED_TO_USE_SSL_CONF error code, 5361
ER_XPLUGIN_FAIL_TO_GET_RESULT_DATA error code, 5371

ER_XPLUGIN_FORCE_STOP_CLIENT error code, 5365
ER_XPLUGIN_GET_PEER_ADDRESS_FAILED error code, 5372
ER_XPLUGIN_INVALID_AUTH_METHOD error code, 5368
ER_XPLUGIN_INVALID_MSG_DURING_CLIENT_INIT error code, 5368
ER_XPLUGIN_IPv6_AVAILABLE error code, 5370
ER_XPLUGIN_LISTENER_SETUP_FAILED error code, 5366
ER_XPLUGIN_LISTENER_STATUS_MSG error code, 5369
ER_XPLUGIN_LISTENER_SYS_VARIABLE_ERROR error code, 5369
ER_XPLUGIN_MAX_AUTH_ATTEMPTS_REACHED error code, 5365
ER_XPLUGIN_MESSAGE_TOO_LONG error code, 5367
ER_XPLUGIN_PEER_DISCONNECTED_WHILE_READING_MSG_BODY error code, 5368
ER_XPLUGIN_READ_FAILED_CLOSING_CONNECTION error code, 5368
ER_XPLUGIN_REFERENCE_TO_SECURE_CONN_WITH_XPLUGIN error code, 5361
ER_XPLUGIN_REFERENCE_TO_USER_ACCOUNT_DOC_SECTION error code, 5366
ER_XPLUGIN_RETRYING_BIND_ON_PORT error code, 5369
ER_XPLUGIN_SCHEDULER_STARTED error code, 5369
ER_XPLUGIN_SCHEDULER_STOPPED error code, 5369
ER_XPLUGIN_SERVER_EXITED error code, 5360
ER_XPLUGIN_SERVER_EXITING error code, 5360
ER_XPLUGIN_SERVER_STARTS_HANDLING_CONNECTIONS error code, 5370
ER_XPLUGIN_SERVER_STOPPED_HANDLING_CONNECTIONS error code, 5370
ER_XPLUGIN_SHUTDOWN_TRIGGERED error code, 5369
ER_XPLUGIN_SRV_SESSION_INIT_THREAD_FAILED error code, 5366
ER_XPLUGIN_SSL_HANDSHAKE_WITH_SERVER_FAILED error code, 5367
ER_XPLUGIN_STARTUP_FAILED error code, 5360
ER_XPLUGIN_TASK_SCHEDULING_FAILED error code, 5366
ER_XPLUGIN_UNABLE_TO_ACCEPT_CONNECTION error code, 5368
ER_XPLUGIN_UNABLE_TO_USE_USER_SESSION_ACCOUNT error code, 5366
ER_XPLUGIN_UNEXPECTED_EXCEPTION_DISPATCHING_CMD error code, 5366
ER_XPLUGIN_UNEXPECTED_MSG_DURING_AUTHENTICATION error code, 5369
ER_XPLUGIN_UNINITIALIZED_MESSAGE error code, 5367
ER_XPLUGIN_UNIX_SOCKET_NOT_CONFIGURED error code, 5370
ER_XPLUGIN_USER_ACCOUNT_WITH_ALL_PERMISSIONS error code, 5369
ER_XPLUGIN_USING_SSL_CONF_FROM_MYSQLX error code, 5360
ER_XPLUGIN_USING_SSL_CONF_FROM_SERVER error code, 5360
ER_XPLUGIN_USING_SSL_FOR_TLS_CONNECTION error code, 5361
ER_XPLUING_NET_STARTUP_FAILED error code, 5367
ER_YES error code, 5098
ER_ZLIB_Z_BUF_ERROR error code, 5116
ER_ZLIB_Z_DATA_ERROR error code, 5116
ER_ZLIB_Z_MEM_ERROR error code, 5116
escape (\\), 1671, 2142
escape sequences
  option files, 324
  strings, 1669
establishing encrypted connections, 1170
estimating
  query performance, 1586
event groups, 2505
event log format (NDB Cluster), 4090
event logs (NDB Cluster), 4087, 4088, 4089
EVENT privilege, 1084
event scheduler, 4331
  thread states, 1666

Event Scheduler, 4340
  altering events, 2245
  and MySQL privileges, 4346
  and mysqladmin debug, 4345
  and replication, 3375, 3375
  and SHOW PROCESSLIST, 4342
  concepts, 4341
  creating events, 2277
  dropping events, 2372
  enabling and disabling, 4342
  event metadata, 4344
  obtaining status information, 4345
  SQL statements, 4344
  starting and stopping, 4342
  time representation, 4345
event severity levels (NDB Cluster), 4089
event types (NDB Cluster), 4087, 4090
EventLogBufferSize, 3827
events, 4331, 4340
  altering, 2245
  creating, 2277
  dropping, 2372
  metadata, 4344
  restrictions, 4363
  status variables, 4348
EVENTS
  INFORMATION_SCHEMA table, 4347, 4381
events option
  mysqldump, 455
  mysqlpump, 478
events table
  data dictionary table, 944
events_errors_summary_by_account_by_error table
  performance_schema, 4647
events_errors_summary_by_host_by_error table
  performance_schema, 4647
events_errors_summary_by_thread_by_error table
  performance_schema, 4647
events_errors_summary_by_user_by_error table
  performance_schema, 4647
events_errors_summary_global_by_error table
  performance_schema, 4647
events_stages_current table
  performance_schema, 4551
events_stages_history table
  performance_schema, 4552
events_stages_history_long table
  performance_schema, 4553
events_stages_summary_by_account_by_event_name table
  performance_schema, 4626
events_stages_summary_by_host_by_event_name table
  performance_schema, 4626
events_stages_summary_by_thread_by_event_name table
  performance_schema, 4626

events_stages_summary_by_user_by_event_name table
performance_schema, 4626
events_stages_summary_global_by_event_name table
performance_schema, 4626
events_statements_current table
performance_schema, 4557
events_statements_histogram_by_digest table
performance_schema, 4631
events_statements_histogram_global table
performance_schema, 4631
events_statements_history table
performance_schema, 4561
events_statements_history_long table
performance_schema, 4561
events_statements_summary_by_account_by_event_name table
performance_schema, 4627
events_statements_summary_by_digest table
performance_schema, 4627
events_statements_summary_by_host_by_event_name table
performance_schema, 4627
events_statements_summary_by_program table
performance_schema, 4627
events_statements_summary_by_thread_by_event_name table
performance_schema, 4627
events_statements_summary_by_user_by_event_name table
performance_schema, 4627
events_statements_summary_global_by_event_name table
performance_schema, 4627
events_transactions_current table
performance_schema, 4568
events_transactions_history table
performance_schema, 4571
events_transactions_history_long table
performance_schema, 4571
events_transactions_summary_by_account_by_event table
performance_schema, 4634
events_transactions_summary_by_host_by_event_name table
performance_schema, 4634
events_transactions_summary_by_thread_by_event_name table
performance_schema, 4634
events_transactions_summary_by_user_by_event_name table
performance_schema, 4634
events_transactions_summary_global_by_event_name table
performance_schema, 4634
events_waits_current table
performance_schema, 4543
events_waits_history table
performance_schema, 4547
events_waits_history_long table
performance_schema, 4547
events_waits_summary_by_account_by_event_name table
performance_schema, 4624
events_waits_summary_by_host_by_event_name table
performance_schema, 4624

events_waits_summary_by_instance table
  performance_schema, 4624
events_waits_summary_by_thread_by_event_name table
  performance_schema, 4624
events_waits_summary_by_user_by_event_name table
  performance_schema, 4624
events_waits_summary_global_by_event_name table
  performance_schema, 4624
event_scheduler system variable, 736
eviction, 6590
exact-value literals, 2228
exact-value numeric literals, 1672, 2229
example option
  mysqld_multi, 365
example programs
  C API, 4787
EXAMPLE storage engine, 3119, 3154
examples
  compressed tables, 536
  myisamchk output, 526
  queries, 303
exceptions table
  NDB Cluster Replication, 4249
exclude-databases option
  mysqlpump, 478
  ndb_restore, 4027
exclude-events option
  mysqlpump, 478
exclude-gtids option
  mysqlbinlog, 555
exclude-intermediate-sql-tables option
  ndb_restore, 4027
exclude-missing-columns option
  ndb_move_data, 4012
  ndb_restore, 4027
exclude-missing-tables option
  ndb_restore, 4027
exclude-routines option
  mysqlpump, 478
exclude-tables option
  mysqlpump, 478
  ndb_restore, 4028
exclude-triggers option
  mysqlpump, 478
exclude-users option
  mysqlpump, 478
exclusive lock, 2810, 6590
Execute
  thread command, 1654
EXECUTE, 2515, 2519
execute option
  mysql, 392
execute option (ndb_mgm), 3967
EXECUTE privilege, 1084

ExecuteOnComputer, 3770, 3777, 3862
execute_prepared_stmt() procedure
  sys schema, 4750
executing
  thread state, 1658
executing SQL statements from text files, 301, 410
Execution of init_command
  thread state, 1658
execution threads (NDB Cluster), 3840
EXISTS
  with subqueries, 2442
exit command
  mysql, 401
EXIT command (NDB Cluster), 4076
EXIT SINGLE USER MODE command (NDB Cluster), 4076
exit-info option
  mysqld, 680
EXP(), 1969
experimental system variables, 213
expired password
  resetting, 1138
expired passwords, 1146
expire_logs_days system variable, 3284
EXPLAIN, 1568, 2701, 4307, 4308
  window functions, 1500
EXPLAIN used with partitioned tables, 4307
explicit default values, 1890
explicit_defaults_for_timestamp system variable, 737
EXPORT_SET(), 1936
expression aliases, 2189, 2422
expression syntax, 1723
expressions
  extended, 294
extend-check option
  myisamchk, 522, 523
extended option
  mysqlcheck, 431
extended-insert option
  mysqldump, 457
  mysqlpump, 478
extensions
  to standard SQL, 56
extent, 6590
external locking, 680, 823, 1439, 1630, 1661
external-locking option
  mysqld, 680
external_user system variable, 739
extra-file option
  my_print_defaults, 575
extra-node-info option
  ndb_desc, 3987
extra-partition-info option
  ndb_desc, 3987
EXTRACT(), 1983

extracting
dates, 290
ExtractValue(), 2034
extract_schema_from_file_name() function
sys schema, 4767
extract_table_from_file_name() function
sys schema, 4767
ExtraSendBufferMemory
API nodes, 3865
data nodes, 3856

F

failover
in NDB Cluster replication, 4229
Java clients, 3656
failure detection
Group Replication, 3407
FALSE, 1672, 1679
testing for, 1924, 1924
false literal
JSON, 1874
FAQs
C API, 4938, 5082
Connector/C, 5082
Connectors and APIs, 5082
InnoDB Data-at-Rest Encryption, 5090
libmysql, 5082
NDB Cluster, 5070
replication, 5083
Virtualization Support, 5092
Fast Index Creation, 6591
fast option
myisamchk, 523
mysqlcheck, 431
fast shutdown, 6591
features of MySQL, 6
FEDERATED storage engine, 3119, 3149
Fetch
thread command, 1654
FETCH, 2529
field
changing, 2259
Field List
thread command, 1654
FIELD(), 1937
fields option
ndb_config, 3974
fields-enclosed-by option
mysqldump, 453, 465
ndb_import, 3996
ndb_restore, 4029
fields-escaped-by option
mysqldump, 453, 465

ndb_import, 3997
fields-optionally-enclosed-by option
mysqldump, 453, 465
ndb_import, 3997
ndb_restore, 4029
fields-terminated-by option
mysqldump, 453, 465
ndb_import, 3997
ndb_restore, 4029
FILE, 1939
file format, 6591
FILE privilege, 1084
file-per-table, 2785, 6591
files
binary log, 971
created by CREATE TABLE, 2336
DDL log, 987
error messages, 1795
general query log, 969
log, 988
metadata log, 987
not found message, 5631
permissions, 5630
repairing, 523
script, 301
size limits, 1549
slow query log, 984
text, 410, 461
FILES
INFORMATION_SCHEMA table, 4384
filesort optimization, 1491, 1610
FileSystemPath, 3780
FileSystemPathDataFiles, 3853
FileSystemPathDD, 3852
FileSystemPathUndoFiles, 3853
file_instances table
performance_schema, 4537
file_summary_by_event_name table
performance_schema, 4637
file_summary_by_instance table
performance_schema, 4637
fill factor, 2758, 6592
fill_help_tables.sql, 936
FIND_IN_SET(), 1937
Finished reading one binlog; switching to next binlog
thread state, 1662
FIPS mode, 1416
firewalls (software)
and NDB Cluster, 4170, 4172
Firewall_access_denied status variable, 1398
Firewall_access_granted status variable, 1398
Firewall_access_suspicious status variable, 1398
FIREWALL_ADMIN privilege, 1089
Firewall_cached_entries status variable, 1398

FIREWALL_USER privilege, 1089
firewall_users table
  system table, 948
firewall_whitelist table
  system table, 948
FirstMatch
  semijoin strategy, 1505
FIRST_VALUE(), 2195
FIXED data type, 1821
fixed row format, 6592
fixed-point arithmetic, 2228
FLOAT data type, 1821, 1821, 1821
floating-point number, 1821
floating-point values
  and replication, 3372
floats, 1672
FLOOR(), 1969
FLUSH
  and replication, 3372
flush, 6592
FLUSH HOSTS
  and TRUNCATE TABLE host_cache, 4653
flush list, 6592
flush option
  mysqld, 680
FLUSH statement, 2691
flush system variable, 739
flush tables, 418
flush-logs option
  mysqldump, 458
flush-privileges option
  mysqldump, 458
flushing, 2847
flush_time system variable, 739
FOR SHARE, 2426
FOR UPDATE, 2426
FORCE
  plugin activation option, 997
FORCE INDEX, 1606, 5646
FORCE KEY, 1606
force option
  myisamchk, 523, 524
  myisampack, 535
  mysql, 392
  mysqladmin, 422
  mysqlcheck, 431
  mysqldump, 448
  mysqlimport, 465
  mysql_upgrade, 381
force-if-open option
  mysqlbinlog, 555
force-read option
  mysqlbinlog, 555
FORCE_INSOURCE_BUILD option

CMake, 207
FORCE_PLUS_PERMANENT
plugin activation option, 997
FORCE_UNSUPPORTED_COMPILER option
CMake, 214
foreign key, 6592
constraint, 63, 63
deleting, 2262, 2343
FOREIGN KEY constraint, 6593
foreign key constraints, 2339
InnoDB, 2752
restrictions, 2752
FOREIGN KEY constraints
and online DDL, 2924
foreign keys, 61, 306, 2262
metadata locking, 1629
foreign_keys table
data dictionary table, 944
foreign_key_checks system variable, 740
foreign_key_column_usage table
data dictionary table, 944
FORMAT(), 1937
FORMAT_BYTES() function, 2209
format_bytes() function
sys schema, 4768
format_path() function
sys schema, 4768
FORMAT_PICO_TIME() function, 2210
format_statement() function
sys schema, 4769
format_time() function
sys schema, 4770
formfeed (\f), 2142
forums, 51
FOUND_ROWS(), 2069
FPROFILE_GENERATE option
CMake, 214
FPROFILE_USE option
CMake, 214
fractional seconds
and replication, 3374
fractional seconds precision, 1818, 1822
FragmentLogFileSize, 3797
FRAGMENT_COUNT_TYPE (NDB_TABLE) (OBSOLETE)
NDB Cluster, 2356
frame
window functions, 2203, 2203
FreeBSD troubleshooting, 227
freeing items
thread state, 1658
.frm file, 6591
FROM, 2423
FROM_BASE64(), 1937
FROM_DAYS(), 1983

FROM_UNIXTIME(), 1983
fs option (ndb_error_reporter), 3990
FTS, 6593
ft_boolean_syntax system variable, 740
ft_max_word_len myisamchk variable, 521
ft_max_word_len system variable, 741
ft_min_word_len myisamchk variable, 521
ft_min_word_len system variable, 741
ft_query_expansion_limit system variable, 742
ft_stopword_file myisamchk variable, 521
ft_stopword_file system variable, 742
full backup, 6593
full disk, 5636
full table scan, 6593
full table scans
  avoiding, 1502
full-text parser plugins, 4946
full-text queries
  optimization, 1526
full-text search, 1997, 6593
FULLTEXT, 1997
fulltext
  stopword list, 2011
FULLTEXT index, 6593
  InnoDB, 2759
  monitoring, 2764
FULLTEXT initialization
  thread state, 1658
fulltext join type
  optimizer, 1574
FULLY_REPLICATED (NDB_TABLE)
  NDB Cluster, 2357
func table
  system table, 946
function
  creating, 2620
  deleting, 2621
function names
  parsing, 1687
  resolving ambiguity, 1687
functional dependence, 920, 2187, 2190
functions, 1900
  and replication, 3372
  arithmetic, 2043
  asynchronous C API, 4913
  binary log C API, 4924
  bit, 2043
  C API, 4799
  cast, 2026
  control flow, 1929
  date and time, 1975
  encryption, 2055
  for SELECT and WHERE clauses, 1900
  GROUP BY, 2171

grouping, 1919
GTIDs, 2161
information, 2065
internal, 2212
locking, 2062
mathematical, 1967
miscellaneous, 2213
native
adding, 5031
new, 5020
Performance Schema, 2209
prepared statement C API, 4879, 4881
stored, 4333
string, 1932
string comparison, 1948
user-defined, 2620, 2621, 5020
adding, 5021
fuzzy checkpointing, 6593

G

GA, 6593
MySQL releases, 78
gap, 6594
gap event, 4214
gap lock, 2810, 6594
InnoDB, 2827
gb2312, gbk, 5071
gci option
ndb_select_all, 4046
gci64 option
ndb_select_all, 4046
GCP Stop errors (NDB Cluster), 3856
gdb
using, 5035
gdb option
mysqld, 681
Gemalto SafeNet KeySecure Applicance
keyring_okv keyring plugin, 1286
general information, 1
General Public License, 5
general query log, 969, 6594
general tablespace, 6594
general_log system variable, 743
general_log table
system table, 946
general_log_file system variable, 743
generated column, 6594
foreign key restrictions, 2753
generated columns
ALTER TABLE, 2270
CREATE TABLE, 2349
CREATE TRIGGER, 2366
CREATE VIEW, 2371

INFORMATION_SCHEMA.COLUMNS table, 4378
INSERT, 2393
REPLACE, 2419
secondary indexes, 2352
SHOW COLUMNS statement, 2637, 4377
UPDATE, 2457
views, 4351
generated_random_password_length system variable, 743
gen_blacklist() MySQL Enterprise Data Masking and De-Identification UDF, 1413
gen_dictionary() MySQL Enterprise Data Masking and De-Identification UDF, 1414
gen_dictionary_drop() MySQL Enterprise Data Masking and De-Identification UDF, 1414
gen_dictionary_load() MySQL Enterprise Data Masking and De-Identification UDF, 1415
gen_range() MySQL Enterprise Data Masking and De-Identification UDF, 1411
gen_rnd_email() MySQL Enterprise Data Masking and De-Identification UDF, 1411
gen_rnd_pan() MySQL Enterprise Data Masking and De-Identification UDF, 1412
gen_rnd_ssn() MySQL Enterprise Data Masking and De-Identification UDF, 1412
gen_rnd_us_phone() MySQL Enterprise Data Masking and De-Identification UDF, 1413
geographic feature, 1854
GeomCollection(), 2084
geometrically valid
GIS values, 1865
spatial values, 1865
geometry, 1854
GEOMETRY data type, 1855
geometry values
internal storage format, 1864
WKB format, 1864
WKT format, 1862
GEOMETRYCOLLECTION data type, 1855
GeometryCollection(), 2084
geospatial feature, 1854
German dictionary collation, 1786, 1786
German phone book collation, 1786, 1786
GET DIAGNOSTICS, 2535
get-server-public-key option, 335
mysql, 392
mysqladmin, 422
mysqlbinlog, 555
mysqlcheck, 431
mysqldump, 443
mysqlimport, 465
mysqlpump, 478
mysqlshow, 491
mysqlslap, 501
mysql_upgrade, 381
getting MySQL, 79
GET_DD_COLUMN_PRIVILEGES(), 2213
GET_DD_CREATE_OPTIONS(), 2213
GET_DD_INDEX_SUB_PART_LENGTH(), 2213
GET_FORMAT(), 1984
GET_LOCK(), 2063
get_sysvar_source plugin service, 5011
GIS, 1853
GIS data types

storage requirements, 1897
GIS values
geometrically valid, 1865
Git tree, 197
GLOBAL
SET statement, 2626
global privileges, 2585, 2598
global temporary tablespace, 6595
global transaction, 6595
global_grants table
system table, 946, 1094, 1096
go command
mysql, 401
Google Test, 213
GRANT OPTION privilege, 1084
GRANT statement, 1112, 2585
privilege restrictions, 2595
grant tables
columns_priv table, 946, 1096
db table, 240, 946, 1096
default_roles table, 946, 1096
global_grants table, 946, 1094, 1096
password_history table, 946, 1096
procs_priv table, 946, 1096
proxies_priv, 1156
proxies_priv table, 240, 946, 1096
role_edges table, 946, 1096
sorting, 1109, 1111
structure, 1095
tables_priv table, 946, 1096
user table, 240, 946, 1096
granting
privileges, 2585
granting roles, 2585
grants
display, 2652
graph option
ndb_top, 4058
greater than (>), 1922
greater than or equal (>=), 1922
greatest timestamp wins (conflict resolution), 4245
greatest timestamp, delete wins (conflict resolution), 4246
GREATEST(), 1923
Group (NDB Cluster), 3941
GROUP BY
aliases in, 2189
extensions to standard SQL, 2187, 2424
implicit sorting, 1491
maximum sort length, 2424
WITH ROLLUP, 2181
GROUP BY functions, 2171
GROUP BY optimizing, 1493
group commit, 2805, 6595
group preferences

LDAP authentication, 1233
Group Replication, 3397
adding a second instance, 3418
adding additional instances, 3421
adding instances, 3418
asynchronous replication, 3399
background, 3398
change to multi-primary mode, 3428
change to single-primary mode, 3428
change which member is primary, 3427
changing group mode, 3428
changing primary member, 3427
choosing mode, 3402
combining versions, 3479
communication protocol, 3429
configuring a group's write concurrency, 3429
configuring consistency guarantees, 3433
configuring distributed recovery, 3441
configuring instances, 3411
configuring online group, 3426
consistency guarantees, 3431
consistency guarantees and data flow, 3432
consistency guarantees choose a level, 3434
consistency guarantees impact on primary election, 3436
consistency guarantees impacts, 3435
consistency guarantees synchronization points, 3432
data definition language statements, 3406
deploying in single primary mode, 3410
deploying instances, 3411
details, 3406
distributed recovery, 3437
election process, 3402
examples of use case scenarios, 3402
failure detection, 3407, 3473
find primary, 3404
fine tuning the group communication thread, 3468
flow control, 3468
frequently asked questions, 3522
getting started, 3410
Group Communication System, 3409
group communication thread (GCT), 3468
group membership, 3407
group write consensus, 3429
group_replication_get_write_concurrency() UDF, 3429
group_replication_ip_whitelist, 3462
group_replication_set_as_primary() UDF, 3427
group_replication_set_write_concurrency() UDF, 3429
group_replication_switch_to_multi_primary_mode() UDF, 3428
group_replication_switch_to_single_primary_mode() UDF, 3428
inspecting a group's write concurrency, 3429
ip address whitelisting, 3462
ipv6, 3454
launching, 3416
limitations, 3520

message compression, 3470
mixed ipv4 and ipv6, 3454
modes, 3402
monitoring, 3424
multi-primary and single-primary modes, 3402
multi-primary mode, 3404
MySQL Enterprise Backup, 3456
network partition, 3473
network partitioning, 3449
observability, 3408
offline upgrade, 3481
online upgrade, 3482
online upgrade considerations, 3482
online upgrade methods, 3483
operations, 3426
Paxos, 3409
performance, 3468
performance message fragmentation, 3472
performance xcom cache, 3472
plugin architecture, 3409
primary failover, 3431
primary secondary replication, 3399
probes and statistics, 3469
recovering from a point in time, 3444
replication group member stats, 3426
replication technologies, 3399
replication_group_members table, 3425
requirements, 3518
requirements and limitations, 3518
responses to failure detection, 3473
secure socket layer support, 3464
security, 3462
server states, 3424
single primary mode, 3402
ssl support, 3464
summary, 3400
system variables, 3485
the group, 3407
throttling, 3470
transaction consistency guarantees, 3431
UDF, 3426, 3427, 3428, 3428, 3429, 3429
understanding transaction consistency guarantees, 3431
upgrading, 3478
upgrading member, 3482
use cases, 3401
user credentials, 3415
view, 3407
view changes, 3444
whitelist, 3462
group replication UDFs
group_replication_get_communication_protocol(), 2514
group_replication_get_write_concurrency(), 2513
group_replication_set_as_primary(), 2511
group_replication_set_communication_protocol(), 2514

group_replication_set_write_concurrency(), 2513
group_replication_switch_to_multi_primary_mode(), 2512
group_replication_switch_to_single_primary_mode(), 2512
group write consensus, 3429
grouping
expressions, 1919
GROUPING(), 2181, 2216
GROUP_CONCAT(), 2175
group_concat_max_len system variable, 744
GROUP_REPLICATION_ADMIN privilege, 1089
group_replication_allow_local_disjoint_gtids_join system variable, 3487
group_replication_allow_local_lower_version_join system variable, 3487
group_replication_autorejoin_tries system variable, 3489
group_replication_auto_increment_increment system variable, 3488
group_replication_bootstrap_group system variable, 3489
group_replication_clone_threshold system variable, 3490
group_replication_communication_debug_options system variable, 3491
group_replication_communication_max_message_size system variable, 3492
group_replication_components_stop_timeout system variable, 3492
group_replication_compression_threshold system variable, 3493
group_replication_consistency system variable, 3493
group_replication_enforce_update_everywhere_checks system variable, 3494
group_replication_exit_state_action system variable, 3495
group_replication_flow_control_applier_threshold system variable, 3497
group_replication_flow_control_certifier_threshold system variable, 3497
group_replication_flow_control_hold_percent system variable, 3497
group_replication_flow_control_max_commit_quota system variable, 3498
group_replication_flow_control_member_quota_percent system variable, 3498
group_replication_flow_control_min_quota system variable, 3499
group_replication_flow_control_min_recovery_quota system variable, 3499
group_replication_flow_control_mode system variable, 3499
group_replication_flow_control_period system variable, 3500
group_replication_flow_control_release_percent system variable, 3500
group_replication_force_members system variable, 3501
group_replication_get_communication_protocol() UDF, 2514
group_replication_get_write_concurrency() UDF, 2513, 3429
group_replication_group_name system variable, 3501
group_replication_group_seeds system variable, 3502
group_replication_gtid_assignment_block_size system variable, 3503
group_replication_ip_whitelist, 3503
group_replication_local_address system variable, 3505
group_replication_member_expel_timeout system variable, 3506
group_replication_member_weight system variable, 3506
group_replication_message_cache_size system variable, 3507
group_replication_poll_spin_loops system variable, 3508
group_replication_recovery_complete_at system variable, 3509
group_replication_recovery_compression_algorithm system variable, 3509
group_replication_recovery_get_public_key system variable, 3510
group_replication_recovery_public_key_path system variable, 3510
group_replication_recovery_reconnect_interval system variable, 3510
group_replication_recovery_retry_count system variable, 3511
group_replication_recovery_ssl_ca system variable, 3511
group_replication_recovery_ssl_capath system variable, 3512
group_replication_recovery_ssl_cert system variable, 3512

group_replication_recovery_ssl_cipher system variable, 3512
group_replication_recovery_ssl_crl system variable, 3513
group_replication_recovery_ssl_crlpath system variable, 3513
group_replication_recovery_ssl_key system variable, 3513
group_replication_recovery_ssl_verify_server_cert system variable, 3514
group_replication_recovery_use_ssl system variable, 3514
group_replication_recovery_zstd_compression_level system variable, 3514
group_replication_set_as_primary() UDF, 2511, 3427
group_replication_set_communication_protocol() UDF, 2514
group_replication_set_write_concurrency() UDF, 2513, 3429
group_replication_single_primary_mode system variable, 3515
group_replication_ssl_mode system variable, 3515
group_replication_start_on_boot system variable, 3516
group_replication_switch_to_multi_primary_mode() UDF, 2512, 3428
group_replication_switch_to_single_primary_mode() UDF, 2512, 3428
group_replication_transaction_size_limit system variable, 3516
group_replication_unreachable_majority_timeout, 3517
GTID functions, 2161
GTID sets
  representation, 3174
gtid-executed-compression-period option (mysqld), 3293
GTIDs, 3172
  and failover, 3185
  and scaleout, 3185
  auto-positioning, 3182
  concepts, 3173
  gtid_purged, 3180
  life cycle, 3177
  logging, 3175
  replication with, 3183
  restrictions, 3188
gtid_executed system variable, 3295
gtid_executed table
  system table, 947, 3175
gtid_executed_compression_period, 3296
gtid_executed_compression_period system variable
  mysql.gtid_executed table, 3177
gtid_mode system variable, 3297
gtid_next system variable, 3297
gtid_owned system variable, 3298
gtid_purged, 3180
gtid_purged system variable, 3299
GTID_SUBSET(), 2161
GTID_SUBTRACT(), 2161

H

HANDLER, 2387
Handlers, 2532
handling
  errors, 5028
hash index, 6595
hash indexes, 1530
hash partitioning, 4278

hash partitions
  managing, 4297
  splitting and merging, 4297
have_compress system variable, 744
have_crypt system variable, 744
have_dynamic_loading system variable, 744
have_geometry system variable, 744
have_openssl system variable, 744
have_profiling system variable, 744
have_query_cache system variable, 745
have_rtree_keys system variable, 745
have_ssl system variable, 745
have_statement_timeout system variable, 745
have_symlink system variable, 745
HAVING, 2424
HDD, 6595
header file
  keyword_list.h, 4395
header option
  ndb_select_all, 4046
header_file option
  comp_err, 368
HEAP storage engine, 3119, 3132
heartbeat, 6595
HeartbeatIntervalDbApi, 3815
HeartbeatIntervalDbDb, 3815
HeartbeatIntervalMgmdMgmd
  management nodes, 3775
HeartbeatOrder, 3816
HeartbeatThreadPriority, 3774, 3865
help command
  mysql, 400
HELP command (NDB Cluster), 4073
help option
  comp_err, 368
  ibd2sdi, 506
  innochecksum, 509
  myisamchk, 519
  myisampack, 535
  myisam_ftdump, 515
  mysql, 389
  mysqladmin, 420
  mysqlbinlog, 551
  mysqlcheck, 429
  mysqld, 674
  mysqldump, 448
  mysqldumpslow, 572
  mysqld_multi, 365
  mysqld_safe, 356
  mysqlimport, 464
  MySQLInstallerConsole, 125
  mysqlpump, 475
  mysqlshow, 490
  mysqlslap, 498

mysql_config_editor, 544
mysql_secure_installation, 370
mysql_ssl_rsa_setup, 374
mysql_upgrade, 379
my_print_defaults, 575
ndb_perror, 4014
ndb_setup.py, 4050
ndb_top, 4059
perror, 577
HELP option
myisamchk, 519
help option (NDB Cluster programs), 4068
HELP statement, 2704
help tables
system tables, 947
help_category table
system table, 947
help_keyword table
system table, 947
help_relation table
system table, 947
help_topic table
system table, 947
hex option
ndb_restore, 4029
HEX(), 1938, 1970
hex-blob option
mysqldump, 453
mysqlpump, 479
hexadecimal literal introducer, 1676
hexadecimal literals, 1675
bit operations, 1677
hexdump option
mysqlbinlog, 555
high-water mark, 6595
HIGH_NOT_PRECEDENCE SQL mode, 917
HIGH_PRIORITY
INSERT modifier, 2395
SELECT modifier, 2427
hintable
system variable, 1603
hints, 58
index, 1606, 2423
optimizer, 1592
histignore option
mysql, 392
histogram_generation_max_mem_size system variable, 745
history list, 6595
history of MySQL, 9
hole punching, 6596
HOME environment variable, 406, 578
host name
default, 340
host name caching, 1644

host name resolution, 1644
host names, 340
  in account names, 1104
  in default account, 240
  in role names, 1107
  maximum length, 23
host option, 333
  mysql, 392
  mysqladmin, 422
  mysqlbinlog, 555
  mysqlcheck, 432
  mysqldump, 443
  mysqlimport, 466
  mysqlpump, 479
  mysqlshow, 491
  mysqlslap, 501
  mysql_secure_installation, 370
  mysql_upgrade, 381
  ndb_config, 3974
  ndb_top, 4059
HostName, 3770, 3777, 3863
HostName (NDB Cluster), 4169
hostname system variable, 747
HostName1, 3933, 3941
HostName2, 3933, 3941
hosts table
  performance_schema, 4574
host_cache table
  performance_schema, 4650
host_summary view
  sys schema, 4706
host_summary_by_file_io view
  sys schema, 4706
host_summary_by_file_io_type view
  sys schema, 4707
host_summary_by_stages view
  sys schema, 4707
host_summary_by_statement_latency view
  sys schema, 4708
host_summary_by_statement_type view
  sys schema, 4709
hot, 6596
hot backup, 6596
HOUR(), 1984
html option
  mysql, 392

I

i-am-a-dummy option
  mysql, 396
ib-file set, 6597
ibbackup_logfile, 6597
.ibd file, 6596

ibd2sdi, 505
  count option, 507
  debug option, 507
  help option, 506
  id option, 507
  no-check option, 509
  pretty option, 509
  skip-data option, 507
  strict-check option, 508
  type option, 508
  version option, 507
ibdata file, 2336, 6597
ibtmp file, 6597
.ibz file, 6596
ib_logfile, 6597
icc
  MySQL builds, 93
ICU_VERSION(), 2070
Id, 3769, 3861
ID
  unique, 4939
id option
  ibd2sdi, 507
idempotent option
  mysqlbinlog, 555
identifiers, 1679
  case sensitivity, 1684
  quoting, 1679
identity system variable, 747
idlesleep option
  ndb_import, 3997
idlespin option
  ndb_import, 3998
IF, 2525
IF(), 1931
IFNULL(), 1931
IGNORE
  DELETE modifier, 2384, 2402
  INSERT modifier, 2395
  UPDATE modifier, 2457
  with partitioned tables, 924, 2395
IGNORE INDEX, 1606
IGNORE KEY, 1606
ignore option
  mysqlimport, 466
ignore-error option
  mysqldump, 456
ignore-lines option
  mysqlimport, 466
  ndb_import, 3998
ignore-spaces option
  mysql, 392
ignore-table option
  mysqldump, 456

IGNORE_AIO_CHECK option
  CMake, 215
ignore_builtin_innodb system variable, 2934
IGNORE_SPACE SQL mode, 917
ilist, 6597
immediate_commit_timestamp, 3361
immediate_server_version system variable, 3217
implicit default values, 1890
implicit GROUP BY sorting, 1491
implicit row lock, 6597
IMPORT TABLE, 2389
IMPORT TABLESPACE, 2265, 2739
importing
  data, 410, 461
IN, 2440
IN(), 1923
in-memory database, 6598
include option
  mysql_config, 574
include-databases option
  mysqlpump, 479
  ndb_restore, 4030
include-events option
  mysqlpump, 479
include-gtids option
  mysqlbinlog, 556
include-master-host-port option
  mysqldump, 450
include-routines option
  mysqlpump, 479
include-tables option
  mysqlpump, 479
  ndb_restore, 4030
include-triggers option
  mysqlpump, 479
include-users option
  mysqlpump, 479
increasing with replication
  speed, 3159
incremental backup, 6598
incremental recovery, 1435
  using NDB Cluster replication, 4236
index, 6598
  deleting, 2261, 2372
  rebuilding, 271
  sorted index builds, 2758
index cache, 6598
index condition pushdown, 6598
INDEX DIRECTORY
  and replication, 3371
index dives
  range optimization, 1455
index dives (for statistics estimation), 2866
index extensions, 1531

index hint, 6599
index hints, 1606, 2423
index join type
  optimizer, 1575
index prefix, 6599
INDEX privilege, 1084
index statistics
  NDB, 3858
index-record lock
  InnoDB, 2827
indexed temporary table
  semijoin strategy, 1505
indexes, 2282
  and BLOB columns, 1526, 2314
  and IS NULL, 1531
  and LIKE, 1530
  and ndb_restore, 4035
  and NULL values, 2314
  and TEXT columns, 1526, 2314
  assigning to key cache, 2689
  BLOB columns, 2283
  block size, 752
  column prefixes, 1526
  columns, 1525
  descending, 1537
  leftmost prefix of, 1523, 1528
  multi-column, 1527
  multiple-part, 2282
  names, 1679
  TEXT columns, 2283
  TIMESTAMP lookups, 1538
  use of, 1523
indexes table
  data dictionary table, 944
IndexMemory, 3783
IndexStatAutoCreate
  data nodes, 3858
IndexStatAutoUpdate
  data nodes, 3859
IndexStatSaveScale
  data nodes, 3859
IndexStatSaveSize
  data nodes, 3859
IndexStatTriggerPct
  data nodes, 3860
IndexStatTriggerScale
  data nodes, 3860
IndexStatUpdateDelay
  data nodes, 3860
index_column_usage table
  data dictionary table, 944
index_merge join type
  optimizer, 1574
index_partitions table

data dictionary table, 944
index_stats table
data dictionary table, 944
index_subquery join type
optimizer, 1574
indirect indexes
NDB Cluster, 2354
INET6_ATON(), 2220
INET6_NTOA(), 2221
INET_ATON(), 2219
INET_NTOA(), 2220
infimum record, 6599
INFO Events (NDB Cluster), 4094
info option
innochecksum, 509
information functions, 2065
information option
myisamchk, 523
INFORMATION SCHEMA
InnoDB tables, 3019
INFORMATION_SCHEMA, 4370, 6599
and security issues, 4174
collation and searching, 1767
connection-control tables, 4474
InnoDB tables, 4428
INNODB_CMP table, 3019
INNODB_CMPMEM table, 3020
INNODB_CMPMEM_RESET table, 3020
INNODB_CMP_RESET table, 3019
INNODB_TRX table, 3021
Thread pool tables, 4472
INFORMATION_SCHEMA plugins, 4948
INFORMATION_SCHEMA queries
optimization, 1516
INFORMATION_SCHEMA.ENGINES table
and NDB Cluster, 4115
INFORMATION_SCHEMA.GLOBAL_STATUS table
and NDB Cluster, 4117
INFORMATION_SCHEMA.GLOBAL_VARIABLES table
and NDB Cluster, 4116
information_schema_stats_expiry system variable, 747
init
thread state, 1658
Init DB
thread command, 1655
init-command option
mysql, 392
InitFragmentLogFiles, 3797
initial option (ndbd), 3950
initial option (ndbmtd), 3950
initial option (ndb_mgmd), 3961
initial-start option (ndbd), 3951
initial-start option (ndbmtd), 3951
initialize option

mysqld, 681
initialize-insecure option
mysqld, 682
Initialized
thread state, 1666
InitialLogFileGroup, 3854
InitialNoOfOpenFiles, 3797
InitialTablespace, 3855
init_connect system variable, 747
init_file system variable, 749
init_slave system variable, 3241
INNER JOIN, 2430
innochecksum, 316, 509
allow-mismatches option, 512
count option, 510
end-page option, 511
help option, 509
info option, 509
log option, 513
no-check option, 511
page option, 511
page-type-dump option, 513
page-type-summary option, 512
read from standard in option, 513
start-page option, 510
strict-check option, 511
verbose option, 510
version option, 510
write option, 512
InnoDB, 2718, 6599
adaptive hash index, 2735
and application feature requirements, 3674
application performance, 2744
applications supported, 3673
architecture, 2725
asynchronous I/O, 2855
auto-inc lock, 2810
auto-increment columns, 2745
autocommit mode, 2818, 2819
availability, 3672
backups, 3068
buffer pool, 2852
change buffer, 2731
checkpoints, 2902
clustered index, 2757
COMPACT row format, 2895
compared to NDB Cluster, 3671, 3672, 3673, 3674
configuration parameters, 2925
consistent reads, 2819
corruption, 3069
crash recovery, 3069, 3070, 3071
creating tables, 2736
data files, 2765
deadlock detection, 2830

deadlock example, 2829
deadlocks, 2742, 2828, 2830
disk failure, 3069
disk I/O, 2900
disk I/O optimization, 1559
DYNAMIC row format, 2896, 2897
exclusive lock, 2810
file space management, 2900
file-per-table setting, 2768
files, 2745
foreign key constraints, 2752
FULLTEXT indexes, 2759
gap lock, 2810, 2827
generated columns, 2753
in-memory structures, 2726
index-record lock, 2827
insert-intention lock, 2810
intention lock, 2810
limits and restrictions, 2754
Linux, 2855
lock modes, 2810
locking, 2810, 2810, 2824
locking reads, 2821
memory usage, 2741
migrating tables, 2738
Monitors, 3112
multi-versioning, 2724
next-key lock, 2810, 2827
NFS, 2754, 2833
on-disk structures, 2736
online DDL, 2903
page size, 2755, 2758
physical index structure, 2758
point-in-time recovery, 3069
primary keys, 2737, 2744
raw partitions, 2767
record-level locks, 2827
recovery, 3068
redo log, 2805, 2805
REDUNDANT row format, 2894
replication, 3071
row format, 2736, 2899
secondary index, 2757
shared lock, 2810
Solaris issues, 187
sorted index builds, 2758
storage, 2743
storage layout, 2742
system variables, 2925
table properties, 2737
tables, 2736
  converting from other storage engines, 2740
transaction model, 2810, 2815
transactions, 2741

transferring data, 2743
troubleshooting, 3112
cannot open datafile, 3115
data dictionary problems, 3114
deadlocks, 2828, 2830
defragmenting tables, 2902
I/O problems, 3112
online DDL, 2924
performance problems, 1553
recovery problems, 3113
restoring orphan ibd files, 3115
SQL errors, 3116
virtual indexes, 2753
InnoDB buffer pool, 1615, 2728, 2839, 2844, 2845, 2846, 2849
InnoDB cluster, 3593
changing a cluster's topology, 3646
checking a cluster's status, 3623
checking instance configuration, 3621
checking instance state, 3640
configuring automatic rejoin of instances, 3633
configuring local instances, 3622
configuring the election process, 3645
creating, 3596
creating a whitelist of servers, 3643
customizing, 3635
dba.dropMetadataSchema(), 3638
deployment scenarios, 3596
describing the structure of an InnoDB cluster, 3623
dissolving an innodb cluster, 3641
getting options of an InnoDB cluster, 3646
groupName, 3635
groupSeeds, 3635
installing, 3597
introduction, 3593
ipWhitelist, 3643
known limitations, 3647
localAddress, 3635
managing sandbox instances, 3634
memberSslMode, 3642
memberWeight, 3645
methods of installing, 3597
MySQL Clone deployment, 3608
MySQL Shell, 3593
overview, 3593
production deployment, 3597
rebooting a cluster from a major outage, 3638
rejoining an instance, 3636
removing instances from, 3635
requirements, 3596
rescanning a cluster, 3639
restoring a cluster from quorum loss, 3636
retrieving an InnoDB cluster, 3622
scripting AdminAPI, 3644
securing your cluster, 3642

setting options for an InnoDB cluster, 3646
super read-only and instances, 3632, 3635
using with MySQL Router, 3616
working with cluster, 3620
InnoDB Monitors, 3061
enabling, 3061
output, 3063
innodb option
mysqld, 2931
InnoDB predicate locks, 2815
InnoDB storage engine, 2718, 3119
InnoDB tables
storage requirements, 1894
innodb-status-file option
mysqld, 2931
innodb_adaptive_flushing system variable, 2935
innodb_adaptive_flushing_lwm system variable, 2935
innodb_adaptive_hash_index
and innodb_thread_concurrency, 2854
innodb_adaptive_hash_index system variable, 2935
innodb_adaptive_hash_index_parts variable, 2936
innodb_adaptive_max_sleep_delay system variable, 2936
innodb_api_bk_commit_interval system variable, 2937
innodb_api_disable_rowlock system variable, 2937
innodb_api_enable_binlog system variable, 2938
innodb_api_enable_mdl system variable, 2938
innodb_api_trx_level system variable, 2938
innodb_autoextend_increment system variable, 2939
innodb_autoinc_lock_mode, 6599
innodb_autoinc_lock_mode system variable, 2939
innodb_background_drop_list_empty system variable, 2940
INNODB_BUFFER_PAGE
INFORMATION_SCHEMA table, 4428
INNODB_BUFFER_PAGE_LRU
INFORMATION_SCHEMA table, 4431
innodb_buffer_pool_chunk_size system variable, 2940
innodb_buffer_pool_debug, 2941
innodb_buffer_pool_dump_at_shutdown system variable, 2942
innodb_buffer_pool_dump_now system variable, 2942
innodb_buffer_pool_dump_pct system variable, 2942
innodb_buffer_pool_filename system variable, 2943
innodb_buffer_pool_instances system variable, 2944
innodb_buffer_pool_in_core_file option, 2852
innodb_buffer_pool_in_core_file system variable, 2943
innodb_buffer_pool_load_abort system variable, 2945
innodb_buffer_pool_load_at_startup system variable, 2945
innodb_buffer_pool_load_now system variable, 2945
innodb_buffer_pool_size system variable, 2946
INNODB_BUFFER_POOL_STATS
INFORMATION_SCHEMA table, 4435
innodb_buffer_stats_by_schema view
sys schema, 4710
innodb_buffer_stats_by_table view
sys schema, 4710

INNODB_CACHED_INDEXES
  INFORMATION_SCHEMA table, 4438
innodb_change_buffering, 2732
innodb_change_buffering system variable, 2947
innodb_change_buffering_debug, 2948
innodb_change_buffer_max_size system variable, 2947
innodb_checkpoint_disabled system variable, 2949
innodb_checksum_algorithm system variable, 2949
INNODB_CMP
  INFORMATION_SCHEMA table, 4439
INNODB_CMPMEM
  INFORMATION_SCHEMA table, 4441
INNODB_CMPMEM_RESET
  INFORMATION_SCHEMA table, 4441
INNODB_CMP_PER_INDEX
  INFORMATION_SCHEMA table, 4442
innodb_cmp_per_index_enabled system variable, 2951
INNODB_CMP_PER_INDEX_RESET
  INFORMATION_SCHEMA table, 4442
INNODB_CMP_RESET
  INFORMATION_SCHEMA table, 4439
INNODB_COLUMNS
  INFORMATION_SCHEMA table, 4444
innodb_commit_concurrency system variable, 2951
innodb_compression_failure_threshold_pct system variable, 2952
innodb_compression_level system variable, 2952
innodb_compression_pad_pct_max system variable, 2953
innodb_compress_debug, 2951
innodb_concurrency_tickets, 2854
innodb_concurrency_tickets system variable, 2953
INNODB_DATAFILES
  INFORMATION_SCHEMA table, 4445
innodb_data_file_path system variable, 2954
innodb_data_home_dir system variable, 2955
innodb_ddl_log table
  data dictionary table, 944
innodb_ddl_log_crash_reset_debug system variable, 2955
innodb_deadlock_detect system variable, 2955
innodb_dedicated_server system variable, 2956
innodb_default_row_format, 2897
innodb_default_row_format system variable, 2956
innodb_directories system variable, 2957
innodb_disable_sort_file_cache system variable, 2958
innodb_doublewrite system variable, 2958
innodb_dynamic_metadata table
  system table, 948
innodb_fast_shutdown system variable, 2959
INNODB_FIELDS
  INFORMATION_SCHEMA table, 4446
innodb_file_per_table, 2875, 6600
innodb_file_per_table system variable, 2960
innodb_fill_factor system variable, 2960
innodb_fil_make_page_dirty_debug, 2959
innodb_flushing_avg_loops system variable, 2965

innodb_flush_log_at_timeout system variable, 2961
innodb_flush_log_at_trx_commit system variable, 2961
innodb_flush_method system variable, 2963
innodb_flush_neighbors system variable, 2964
innodb_flush_sync system variable, 2965
innodb_force_load_corrupted system variable, 2966
innodb_force_recovery system variable, 2966
  DROP TABLE, 2375
INNODB_FOREIGN
  INFORMATION_SCHEMA table, 4447
INNODB_FOREIGN_COLS
  INFORMATION_SCHEMA table, 4447
innodb_fsync_threshold system variable, 2967
innodb_ft_aux_table system variable, 2967
INNODB_FT_BEING_DELETED
  INFORMATION_SCHEMA table, 4448
innodb_ft_cache_size system variable, 2968
INNODB_FT_CONFIG
  INFORMATION_SCHEMA table, 4449
INNODB_FT_DEFAULT_STOPWORD
  INFORMATION_SCHEMA table, 4450
INNODB_FT_DELETED
  INFORMATION_SCHEMA table, 4451
innodb_ft_enable_diag_print system variable, 2968
innodb_ft_enable_stopword system variable, 2969
INNODB_FT_INDEX_CACHE
  INFORMATION_SCHEMA table, 4452
INNODB_FT_INDEX_TABLE
  INFORMATION_SCHEMA table, 4453
innodb_ft_max_token_size system variable, 2969
innodb_ft_min_token_size system variable, 2970
innodb_ft_num_word_optimize system variable, 2970
innodb_ft_result_cache_limit system variable, 2971
innodb_ft_server_stopword_table system variable, 2971
innodb_ft_sort_pll_degree system variable, 2972
innodb_ft_total_cache_size system variable, 2972
innodb_ft_user_stopword_table system variable, 2973
innodb_idle_flush_pct system variable, 2973
INNODB_INDEXES
  INFORMATION_SCHEMA table, 4455
innodb_index_stats table
  system table, 947, 2859
innodb_io_capacity, 2856
innodb_io_capacity system variable, 2973
innodb_io_capacity_max system variable, 2975
innodb_limit_optimistic_insert_debug, 2976
INNODB_LOCKS
  INFORMATION_SCHEMA table, 4456
INNODB_LOCK_WAITS
  INFORMATION_SCHEMA table, 4457
innodb_lock_waits view
  sys schema, 4711
innodb_lock_wait_timeout, 6600
innodb_lock_wait_timeout system variable, 2976

innodb_log_buffer_size system variable, 2977
innodb_log_checkpoint_fuzzy_now system variable, 2977
innodb_log_checkpoint_now system variable, 2978
innodb_log_checksums system variable, 2978
innodb_log_compressed_pages system variable, 2978
innodb_log_files_in_group system variable, 2980
innodb_log_file_size system variable, 2979
innodb_log_group_home_dir system variable, 2980
innodb_log_spin_cpu_abs_lwm system variable, 2980
innodb_log_spin_cpu_pct_hwm system variable, 2981
innodb_log_wait_for_flush_spin_hwm system variable, 2981
innodb_log_write_ahead_size system variable, 2982
innodb_lru_scan_depth system variable, 2983
innodb_max_dirty_pages_pct system variable, 2983
innodb_max_dirty_pages_pct_lwm system variable, 2984
innodb_max_purge_lag system variable, 2984
innodb_max_purge_lag_delay system variable, 2985
innodb_max_undo_log_size system variable, 2985
innodb_memcache database, 3079, 3106
innodb_memcached_config.sql script, 3079
innodb_merge_threshold_set_all_debug, 2986
INNODB_METRICS
  INFORMATION_SCHEMA table, 4458
innodb_monitor_disable system variable, 2986
innodb_monitor_enable system variable, 2987
innodb_monitor_reset system variable, 2987
innodb_monitor_reset_all system variable, 2988
innodb_numa_interleave variable, 2988
innodb_old_blocks_pct, 2845
innodb_old_blocks_pct system variable, 2988
innodb_old_blocks_time, 2845
innodb_old_blocks_time system variable, 2989
innodb_online_alter_log_max_size system variable, 2989
innodb_open_files system variable, 2990
innodb_optimize_fulltext_only system variable, 2990
innodb_page_cleaners system variable, 2991
innodb_page_size system variable, 2992
innodb_parallel_read_threads system variable, 2993
innodb_print_all_deadlocks system variable, 2994
  innodb_print_all_deadlocks, 2994
innodb_print_ddl_logs system variable, 2994
innodb_purge_batch_size system variable, 2994
innodb_purge_rseg_truncate_frequency system variable, 2995
innodb_purge_threads system variable, 2995
innodb_random_read_ahead system variable, 2996
innodb_read_ahead_threshold, 2846
innodb_read_ahead_threshold system variable, 2996
innodb_read_io_threads, 2855
innodb_read_io_threads system variable, 2997
innodb_read_only system variable, 2998
INNODB_REDO_LOG_ARCHIVE privilege, 1089
innodb_redo_log_archive_dirs system variable, 2998
innodb_redo_log_encrypt system variable, 2999
innodb_replication_delay system variable, 2999

innodb_rollback_on_timeout system variable, 3000
innodb_rollback_segments system variable, 3000
innodb_saved_page_number_debug, 3001
innodb_scan_directories system variable, 3001
INNODB_SESSION_TEMP_TABLESPACES
  INFORMATION_SCHEMA table, 4460
innodb_sort_buffer_size system variable, 3001
innodb_spin_wait_delay, 2856
innodb_spin_wait_delay system variable, 3002
innodb_spin_wait_pause_multiplier, 2856
innodb_spin_wait_pause_multiplier system variable, 3003
innodb_stats_auto_recalc system variable, 3003
innodb_stats_include_delete_marked system variable, 2861, 3004
innodb_stats_method system variable, 3004
innodb_stats_on_metadata system variable, 3005
innodb_stats_persistent system variable
  innodb_stats_persistent, 3006
innodb_stats_persistent_sample_pages system variable, 3006
innodb_stats_transient_sample_pages, 2866
innodb_stats_transient_sample_pages system variable, 3006
innodb_status_output system variable, 3007
innodb_status_output_locks system variable, 3007
innodb_stat_persistent system variable, 3006
innodb_strict_mode, 6600
innodb_strict_mode system variable, 3008
innodb_sync_array_size system variable, 3009
innodb_sync_debug, 3009
innodb_sync_spin_loops system variable, 3009
INNODB_TABLES
  INFORMATION_SCHEMA table, 4461
INNODB_TABLESPACES
  INFORMATION_SCHEMA table, 4462
INNODB_TABLESPACES_BRIEF
  INFORMATION_SCHEMA table, 4464
INNODB_TABLESTATS
  INFORMATION_SCHEMA table, 4465
innodb_table_locks system variable, 3010
innodb_table_stats table
  system table, 947, 2859
innodb_temp_data_file_path system variable, 3010
innodb_temp_tablespaces_dir system variable, 3011
INNODB_TEMP_TABLE_INFO
  INFORMATION_SCHEMA table, 4467
innodb_thread_concurrency, 2854
innodb_thread_concurrency system variable, 3012
innodb_thread_sleep_delay, 2854
innodb_thread_sleep_delay system variable, 3013
innodb_tmpdir system variable, 3014
INNODB_TRX
  INFORMATION_SCHEMA table, 4468
innodb_trx_purge_view_update_only_debug, 3014
innodb_trx_rseg_n_slots_debug, 3015
innodb_undo_directory system variable, 3015
innodb_undo_logs system variable, 3016

innodb_undo_log_encrypt system variable, 3016
innodb_undo_log_truncate system variable, 3016
innodb_undo_tablespaces system variable, 3017
innodb_use_native_aio, 2855
innodb_use_native_aio system variable, 3018
innodb_version system variable, 3018
INNODB_VIRTUAL
  INFORMATION_SCHEMA table, 4471
innodb_write_io_threads, 2855
innodb_write_io_threads system variable, 3018
INOUT parameter
  condition handling, 2559
input-type option
  ndb_import, 3998
input-workers option
  ndb_import, 3998
INSERT, 1521, 2391
insert, 6600
INSERT ... SELECT, 2396
insert buffer, 6600
insert buffering, 6600
  disabling, 2732
INSERT DELAYED, 2399, 2399
insert intention lock, 6601
INSERT privilege, 1084
INSERT(), 1938
insert-ignore option
  mysqldump, 457
  mysqlpump, 479
insert-intention lock, 2810
insertable views
  insertable, 4351
inserting
  speed of, 1521
InsertRecoveryWork, 3802
inserts
  concurrent, 1623, 1626
insert_id system variable, 749
INSTALL COMPONENT statement, 2622
install option
  mysqld, 682
  MySQLInstallerConsole, 125
install option (ndbd), 3951
install option (ndbmtd), 3951
install option (ndb_mgmd), 3962
INSTALL PLUGIN statement, 2622
install-manual option
  mysqld, 682
Installation, 124
installation layouts, 93
installation overview, 190
installing
  binary distribution, 93
  Linux RPM packages, 160

macOS DMG packages, 141
overview, 76
Perl, 274
Perl on Windows, 275
Solaris PKG packages, 187
source distribution, 190
user-defined functions, 5029
installing components, 2622
installing NDB Cluster, 3685
Debian Linux, 3717
Linux, 3710
Linux binary release, 3710
Linux RPM, 3713
Linux source release, 3718
Ubuntu Linux, 3717
Windows, 3719
Windows binary release, 3719
Windows source, 3723
installing plugins, 994, 2622
installing server components, 990
installing UDFs, 1054
INSTALL_BINDIR option
CMake, 207
INSTALL_DOCDIR option
CMake, 207
INSTALL_DOCREADMEDIR option
CMake, 207
INSTALL_INCLUDEDIR option
CMake, 207
INSTALL_INFODIR option
CMake, 207
INSTALL_LAYOUT option
CMake, 208
INSTALL_LIBDIR option
CMake, 208
INSTALL_MANDIR option
CMake, 208
INSTALL_MYSQLKEYRINGDIR option
CMake, 208
INSTALL_MYSQLSHAREDIR option
CMake, 208
INSTALL_MYSQLTESTDIR option
CMake, 208
INSTALL_PKGCONFIGDIR option
CMake, 208
INSTALL_PLUGINDIR option
CMake, 208
INSTALL_SBINDIR option
CMake, 209
INSTALL_SECURE_FILE_PRIVDIR option
CMake, 209
INSTALL_SHAREDIR option
CMake, 209
INSTALL_STATIC_LIBRARIES option

CMake, 209
INSTALL_SUPPORTFILESDIR option
CMake, 209
instance, 6601
INSTR(), 1938
instrumentation, 6601
INT data type, 1820
integer arithmetic, 2228
INTEGER data type, 1820
integers, 1672
intention lock, 2810, 6601
interactive option (ndb_mgmd), 3962
interactive_timeout system variable, 750
internal functions, 2212
internal locking, 1622
internal storage format
geometry values, 1864
internals, 4943
INTERNAL_AUTO_INCREMENT(), 2213
INTERNAL_AVG_ROW_LENGTH(), 2213
INTERNAL_CHECKSUM(), 2213
INTERNAL_CHECK_TIME(), 2213
INTERNAL_DATA_FREE(), 2213
INTERNAL_DATA_LENGTH(), 2213
INTERNAL_DD_CHAR_LENGTH(), 2213
INTERNAL_GET_COMMENT_OR_ERROR(), 2213
INTERNAL_GET_VIEW_WARNING_OR_ERROR(), 2213
INTERNAL_INDEX_COLUMN_CARDINALITY(), 2213
INTERNAL_INDEX_LENGTH(), 2213
INTERNAL_KEYS_DISABLED(), 2213
INTERNAL_MAX_DATA_LENGTH(), 2213
INTERNAL_TABLE_ROWS(), 2213
internal_tmp_disk_storage_engine system variable, 750
internal_tmp_mem_storage_engine system variable, 750
INTERNAL_UPDATE_TIME(), 2213
INTERVAL
temporal interval syntax, 1725
interval syntax, 1725
INTERVAL(), 1924
INTO
SELECT, 2428
intrinsic temporary table, 6601
introducer
binary character set, 1794
bit-value literal, 1678
character set, 1747
hexadecimal literal, 1676
string literal, 1670, 1744
invalid data
constraint, 64
inverted index, 6601
invisible index, 1536, 2262, 2297
INVOKER privileges, 2653, 4354
in_file option

comp_err, 368
IOPS, 6601
io_by_thread_by_latency view
sys schema, 4713
io_global_by_file_by_bytes view
sys schema, 4714
io_global_by_file_by_latency view
sys schema, 4715
io_global_by_wait_by_bytes view
sys schema, 4715
io_global_by_wait_by_latency view
sys schema, 4716
IP addresses
in account names, 1104
IPv6 addresses
in account names, 1104
IPv6 connections, 694
IS boolean_value, 1924
IS NOT boolean_value, 1924
IS NOT DISTINCT FROM operator, 1921
IS NOT NULL, 1925
IS NULL, 1488, 1924
and indexes, 1531
ISNULL(), 1925
ISOLATION LEVEL, 2482
isolation level, 2815, 6602
IS_FREE_LOCK(), 2064
IS_IPV4(), 2221
IS_IPV4_COMPAT(), 2221
IS_IPV4_MAPPED(), 2222
IS_IPV6(), 2222
IS_USED_LOCK(), 2064
IS_UUID(), 2222
IS_VISIBLE_DD_OBJECT(), 2213
ITERATE, 2526
iterations option
mysqlslap, 501

J

Japanese character sets
conversion, 5071
Japanese, Korean, Chinese character sets
frequently asked questions, 5071
Java, 4784
JDBC, 4781
jdbc:mysql:loadbalance://, 3656
join, 6602
nested-loop algorithm, 1472
JOIN, 2430
join algorithm
Block Nested-Loop, 1468
Nested-Loop, 1468
join option

myisampack, 535
join type
ALL, 1575
const, 1573
eq_ref, 1573
fulltext, 1574
index, 1575
index_merge, 1574
index_subquery, 1574
range, 1575
ref, 1574
ref_or_null, 1574
system, 1573
unique_subquery, 1574
joins
USING versus ON, 2434
join_buffer_size system variable, 751
JSON
array, 1874
autowrapped values, 1879
false literal, 1874
NDB Cluster, 2354
normalized values, 1879
null literal, 1874
null, true, and false literals, 1877
object, 1874
quote mark handling, 1877
scalar, 1874
sensible values, 1879
string, 1874
temporal values, 1874
true literal, 1874
valid values, 1875
JSON data type, 1872
JSON functions, 2118, 2118
JSON pointer URI fragment identifiers, 2153
JSON schema validation, 2150
JSON_ARRAY(), 2120
JSON_ARRAYAGG(), 2176
JSON_ARRAY_APPEND(), 2134
JSON_ARRAY_INSERT(), 2135
JSON_CONTAINS(), 2121
JSON_CONTAINS_PATH(), 2122
JSON_DEPTH(), 2143
JSON_EXTRACT(), 2122
JSON_INSERT(), 2135
JSON_KEYS(), 2127
JSON_LENGTH(), 2143
JSON_MERGE(), 2136
JSON_MERGE() (deprecated), 1880
JSON_MERGE_PATCH(), 1880, 2137
JSON_MERGE_PRESERVE(), 1880, 2139
JSON_OBJECT(), 2120
JSON_OBJECTAGG(), 2176

JSON_OVERLAPS(), 2127
JSON_PRETTY(), 2155
JSON_QUOTE(), 2120
JSON_REMOVE(), 2140
JSON_REPLACE(), 2140
JSON_SCHEMA_VALID(), 2151
JSON_SCHEMA_VALIDATION_REPORT(), 2152
JSON_SEARCH(), 2129
JSON_SET(), 2141
JSON_STORAGE_FREE(), 2156
JSON_STORAGE_SIZE(), 2158
JSON_TABLE(), 2146
JSON_TYPE(), 2144
JSON_UNQUOTE(), 2141
JSON_VALID(), 2145

K

keep-state option
  ndb_import, 3999
keep_files_on_create system variable, 752
Key cache
  MyISAM, 1615
key cache
  assigning indexes to, 2689
key management
  keyring, 1295
key partitioning, 4281
key partitions
  managing, 4297
  splitting and merging, 4297
key space
  MyISAM, 3128
key-file option
  ndb_setup.py, 4051
key-value store, 1531
keyring, 1277
  key management, 1295
keyring plugins, 4950, 5007
  keyring_aws, 1287
  keyring_encrypted_file, 1281
  keyring_file, 1280
  keyring_okv, 1282
keyring service functions
  my_key_fetch(), 5018
  my_key_generate(), 5019
  my_key_remove(), 5019
  my_key_store(), 5020
keyring system variables, 1306
keyring UDFs
  general purpose, 1295
  installing, 1295
  keyring_key_fetch(), 1300
  keyring_key_generate(), 1301

keyring_key_length_fetch(), 1301
keyring_key_remove(), 1302
keyring_key_store(), 1302
keyring_key_type_fetch(), 1303
plugin specific, 1303
uninstalling, 1295
using, 1296
keyring-migration-destination option
mysqld, 1304
keyring-migration-host option
mysqld, 1304
keyring-migration-password option
mysqld, 1304
keyring-migration-port option
mysqld, 1305
keyring-migration-socket option
mysqld, 1305
keyring-migration-source option
mysqld, 1305
keyring-migration-user option
mysqld, 1306
keyring_aws keyring plugin, 1287
keyring_aws plugin
installing, 1279
keyring_aws UDFs
keyring_aws_rotate_cmk(), 1303
keyring_aws_rotate_keys(), 1303
keyring_aws_cmk_id system variable, 1306
keyring_aws_conf_file system variable, 1306
keyring_aws_data_file system variable, 1307
keyring_aws_region system variable, 1307
keyring_aws_rotate_cmk() keyring_aws UDF, 1303
keyring_aws_rotate_keys() keyring_aws UDF, 1303
keyring_encrypted_file keyring plugin, 1281
keyring_encrypted_file plugin
installing, 1279
keyring_encrypted_file_data system variable, 1308
keyring_encrypted_file_password system variable, 1309
keyring_file keyring plugin, 1280
keyring_file plugin, 2795
installing, 1279
keyring_file_data system variable, 1310
keyring_keys table
performance_schema, 1278, 1338, 1365, 4654
keyring_key_fetch() keyring UDF, 1300
keyring_key_generate() keyring UDF, 1301
keyring_key_length_fetch() keyring UDF, 1301
keyring_key_remove() keyring UDF, 1302
keyring_key_store() keyring UDF, 1302
keyring_key_type_fetch() keyring UDF, 1303
keyring_okv keyring plugin, 1282
configuring, 1283
Gemalto SafeNet KeySecure Applicance, 1286
Oracle Key Vault, 1284

keyring_okv plugin, 2795
  installing, 1279
keyring_okv_conf_dir system variable, 1311
keyring_operations system variable, 1312
keyring_udf plugin
  installing, 1295
  uninstalling, 1295
keys, 1525
  foreign, 61, 306
  multi-column, 1527
  searching on two, 308
keys option
  mysqlshow, 491
keys-used option
  myisamchk, 524
keywords, 1691
KEYWORDS
  INFORMATION_SCHEMA table, 4395
keyword_list.h header file, 4395
KEY_BLOCK_SIZE, 2875, 2881, 6602
key_buffer_size myisamchk variable, 521
key_buffer_size system variable, 752
key_cache_age_threshold system variable, 753
key_cache_block_size system variable, 754
key_cache_division_limit system variable, 754
KEY_COLUMN_USAGE
  INFORMATION_SCHEMA table, 4392
Kill
  thread command, 1655
KILL statement, 2696
Killed
  thread state, 1658
Killing slave
  thread state, 1664, 1665
known errors, 5648
Korean, 5071

L

labels
  stored program block, 2520
LAG(), 2196
language option
  mysqld, 682
language support
  error messages, 1795
lap option
  ndb_redo_log_reader, 4019
large page support, 1640
large tables
  NDB Cluster, 2325
large-pages option
  mysqld, 683
large_files_support system variable, 755

large_pages system variable, 755
large_page_size system variable, 755
last row
  unique ID, 4939
LAST_DAY(), 1985
last_insert_id system variable, 756
LAST_INSERT_ID(), 2070, 2395, 4939
  and replication, 3364
  and stored routines, 4335
  and triggers, 4335
LAST_VALUE(), 2197
latch, 6602
LateAlloc, 3809
lateral derived tables, 1578, 2448
latest_file_io view
  sys schema, 4717
layout of installation, 93
lc-messages option
  mysqld, 683
lc-messages-dir option
  mysqld, 684
LCASE(), 1938
LcpScanProgressTimeout, 3798
lcp_simulator.cc (test program), 3802
lc_messages system variable, 756
lc_messages_dir system variable, 756
lc_time_names system variable, 756
LD)_PRELOAD environment variable, 184
LDAP
  authentication, 1221
LDAP authentication
  client-side logging, 1248
  server-side logging, 1247, 1255
  WITH_AUTHENTICATION_LDAP CMake option, 216
LDML syntax, 1808
LD_LIBRARY_PATH environment variable, 276, 4793
LD_RUN_PATH environment variable, 276, 578
LEAD(), 2197
LEAST(), 1925
LEAVE, 2526
ledir option
  mysqld_safe, 357
LEFT JOIN, 1476, 2430
LEFT OUTER JOIN, 2430
LEFT(), 1938
leftmost prefix of indexes, 1523, 1528
legal names, 1679
length option
  myisam_ftdump, 516
LENGTH(), 1939
less than (<), 1921
less than or equal (<=), 1921
libaio, 94, 166, 215
libmysql

FAQ, 5082
libmysqlclient library, 4781
LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN environment variable, 578
LIBMYSQL_PLUGINS environment variable, 578, 4921
LIBMYSQL_PLUGIN_DIR environment variable, 578, 4921
library
libmysqlclient, 4781
libs option
mysql_config, 574
libs_r option
mysql_config, 574
license system variable, 757
LIKE, 1948
and indexes, 1530
and wildcards, 1530
LIMIT, 2069, 2425
and replication, 3377
optimizations, 1496
limitations
replication, 3364
resource groups, 1650
limitations of NDB Cluster, 3674
limits
file-size, 1549
InnoDB, 2754
maximum columns per table, 1550
maximum number of databases, 1549, 2277
maximum number of tables, 1549, 2312
maximum row size, 1551
maximum tables per join, 2431
maximum tables per view, 4367
table size, 1549
line-numbers option
mysql, 393
linear hash partitioning, 4280
linear key partitioning, 4282
linefeed (\n), 1671, 2142, 2406
lines-terminated-by option
mysqldump, 453, 466
ndb_import, 3999
ndb_restore, 4031
LINESTRING data type, 1855
LineString(), 2084
linking, 4788
errors, 4790
problems, 4790
links
symbolic, 1633
LINK_RANDOMIZE option
CMake, 209
LINK_RANDOMIZE_SEED option
CMake, 209
list, 6602
list option

MySQLInstallerConsole, 126
list partitioning, 4267, 4270
list partitions
adding and dropping, 4290
managing, 4290
list_add() function
sys schema, 4770
list_drop() function
sys schema, 4771
literals, 1669
bit value, 1677
boolean, 1679
date, 1672
hexadecimal, 1675
numeric, 1672
string, 1669
time, 1672
LN(), 1970
LOAD DATA, 2399, 5642
and replication, 3377
load emulation, 495
LOAD XML, 2410
loading
tables, 285
LOAD_FILE(), 1939
load_rewrite_rules() Rewriter UDF, 1015
local option
mysqlimport, 466
local-infile option
mysql, 393
local-load option
mysqlbinlog, 556
local-service option
mysqld, 684
localhost
special treatment of, 342
LOCALTIME, 1985
LOCALTIMESTAMP, 1985
local_infile system variable, 757
LOCATE(), 1939
LocationDomainId (API nodes), 3863
LocationDomainId (data nodes), 3778
LocationDomainId (management nodes), 3771
lock, 6603
lock escalation, 6603
LOCK IN SHARE MODE, 2426
LOCK INSTANCE FOR BACKUP, 2475
lock mode, 6603
Lock Monitor, 3061, 3063
lock option
ndb_select_all, 4045
LOCK TABLES, 2476
LOCK TABLES privilege, 1084
lock-all-tables option

mysqldump, 458
lock-tables option
mysqldump, 458
mysqlimport, 466
Locked_connects status variable, 905
locked_in_memory system variable, 758
LockExecuteThreadToCPU, 3836
locking, 2810, 6603
external, 680, 823, 1439, 1630, 1661
information schema, 3021
InnoDB, 2810
internal, 1622
metadata, 1626
Performance Schema, 3021
row-level, 1622
table-level, 1622
locking functions, 2062
locking methods, 1622
locking read, 2823, 6603
NOWAIT, 2823
SKIP LOCKED, 2823
locking service
installing, 5014
mysql_acquire_locking_service_locks() C function, 5013
mysql_release_locking_service_locks() C function, 5014
service_get_read_locks() UDF, 5017
service_get_write_locks() UDF, 5017
service_release_locks() UDF, 5018
uninstalling, 5014
Locking system tables
thread state, 1658
locking_service plugin service, 5011
locking_service service, 5012
LockMaintThreadsToCPU, 3837
LockPagesInMainMemory, 3809
locks_per_fragment
ndbinfo table, 4140
lock_order system variable, 5041
LOCK_ORDER tool, 5040
lock_order_debug_loop system variable, 5042
lock_order_debug_missing_arc system variable, 5042
lock_order_debug_missing_key system variable, 5042
lock_order_debug_missing_unlock system variable, 5043
lock_order_dependencies system variable, 5043
lock_order_extra_dependencies system variable, 5043
lock_order_output_directory system variable, 5044
lock_order_print_txt system variable, 5044
lock_order_trace_loop system variable, 5044
lock_order_trace_missing_arc system variable, 5045
lock_order_trace_missing_key system variable, 5045
lock_order_trace_missing_unlock system variable, 5045
lock_wait_timeout system variable, 757
log, 6603
log buffer, 6604

log component
  log_filter_dragnet, 992
  log_filter_internal, 991
  log_sink_internal, 992
  log_sink_json, 992
  log_sink_syseventlog, 993
  log_sink_test, 993
log file, 6604
log files
  maintaining, 988
log files (NDB Cluster), 3953
  ndbmtd, 3957
log group, 6604
log option
  innochecksum, 513
  mysqld_multi, 365
LOG(), 1970
log-bin option
  mysqld, 3264
log-bin-index option
  mysqld, 3265
log-error option
  mysqld, 684
  mysqldump, 448
  mysqld_safe, 357
log-error-file option
  mysqlpump, 480
log-isam option
  mysqld, 685
log-level option
  ndb_import, 3999
log-name option (ndb_mgmd), 3962
log-raw option
  mysqld, 685
log-short-format option
  mysqld, 685
log-tc option
  mysqld, 686
log-tc-size option
  mysqld, 686
log-warnings option
  mysqld, 686
LOG10(), 1971
LOG2(), 1970
logbuffer-size option (ndbd), 3952
logbuffer-size option (ndbmtd), 3952
logbuffers
  ndbinfo table, 4141
LogDestination, 3771
logging
  passwords, 1070
logging commands (NDB Cluster), 4088
logging slow query
  thread state, 1658

logical, 6604
logical backup, 6604
logical operators, 1926
login
  thread state, 1658
login-path option, 327, 576
  mysql, 393
  mysqladmin, 422
  mysqlbinlog, 556
  mysqlcheck, 432
  mysqldump, 443
  mysqlimport, 466
  mysqlpump, 480
  mysqlshow, 491
  mysqlslap, 501
  mysql_upgrade, 381
  NDB client programs, 4069
LogLevelCheckpoint, 3829
LogLevelCongestion, 3830
LogLevelConnection, 3830
LogLevelError, 3830
LogLevelInfo, 3830
LogLevelNodeRestart, 3829
LogLevelShutdown, 3828
LogLevelStartup, 3828
LogLevelStatistic, 3829
logs
  flushing, 948
  server, 948
logspaces
  ndbinfo table, 4142
log_bin system variable, 3285
log_bin_basename system variable, 3285
log_bin_index system variable, 3286
log_bin_trust_function_creators system variable, 3286
log_bin_use_v1_row_events system variable, 3286
log_builtin_as_identified_by_password system variable, 3287
log_error system variable, 758
log_error_filter_rules system variable, 759
log_error_services system variable, 759
log_error_suppression_list system variable, 759
log_error_verbosity system variable, 760
log_filter_dragnet log component, 992
log_filter_internal log component, 991
log_output system variable, 761
log_queries_not_using_indexes system variable, 761
log_sink_internal log component, 992
log_sink_json log component, 992
log_sink_syseventlog log component, 993
log_sink_test log component, 993
log_slave_updates system variable, 3287
log_slow_admin_statements system variable
  mysqld, 762
log_slow_extra system variable, 762

log_slow_slave_statements system variable
  mysqld, 3241
log_statements_unsafe_for_binlog system variable, 3288
log_status table
  performance_schema, 4654
log_syslog system variable, 762
log_syslog_facility system variable, 763
log_syslog_include_pid system variable, 763
log_syslog_tag system variable, 764
log_throttle_queries_not_using_indexes system variable, 764
log_timestamps system variable, 764
log_warnings system variable, 765
Long Data
  thread command, 1655
LONG data type, 1846
LONGBLOB data type, 1827
LongMessageBuffer, 3793
LONGTEXT data type, 1827
long_query_time system variable, 765
LOOP, 2526
  labels, 2520
loops option
  ndb_show_tables, 4053
loops option (ndbinfo_select_all), 3956
loops option (ndb_index_stat), 4010
Loose Index Scan
  GROUP BY optimizing, 1493
--loose option prefix, 327
LooseScan
  semijoin strategy, 1505
loose_, 6604
lossy-conversions option
  ndb_move_data, 4012
  ndb_restore, 4031
lost connection errors, 5622
low-priority option
  mysqlimport, 466
low-water mark, 6604
LOWER(), 1939
lower_case_file_system system variable, 766
  GRANT, 2591
lower_case_table_names system variable, 766
LOW_PRIORITY
  DELETE modifier, 2384
  INSERT modifier, 2395
  UPDATE modifier, 2457
low_priority_updates system variable, 766
LPAD(), 1940
LRU, 6604
LRU page replacement, 2845
LSN, 6604
LTRIM(), 1940
lz4_decompress, 317, 576

M

macOS
  installation, 141
main features of MySQL, 6
maintaining
  log files, 988
  tables, 1443
maintenance
  tables, 426
MAKEDATE(), 1985
MAKETIME(), 1985
MAKE_SET(), 1940
Making temporary file (append) before replaying LOAD DATA INFILE
  thread state, 1664
Making temporary file (create) before replaying LOAD DATA INFILE
  thread state, 1664
malicious SQL statements
  and NDB Cluster, 4174
manage keys
  thread state, 1658
management
  resource groups, 1648
management client (NDB Cluster), 3966
  (see also mgm)
management node (NDB Cluster)
  defined, 3654
management nodes (NDB Cluster), 3958
  (see also mgmd)
managing NDB Cluster, 4070
managing NDB Cluster processes, 3946
mandatory_roles system variable, 767
manual
  available formats, 2
  online location, 2
  syntax conventions, 3
  typographical conventions, 3
mask_inner() MySQL Enterprise Data Masking and De-Identification UDF, 1407
mask_outer() MySQL Enterprise Data Masking and De-Identification UDF, 1408
mask_pan() MySQL Enterprise Data Masking and De-Identification UDF, 1409
mask_pan_relaxed() MySQL Enterprise Data Masking and De-Identification UDF, 1409
mask_ssn() MySQL Enterprise Data Masking and De-Identification UDF, 1410
Master has sent all binlog to slave; waiting for more updates
  thread state, 1663
master server, 6605
master thread, 6605
master-data option
  mysqldump, 450
master-info-file option
  mysqld, 3222
master-info-repository option
  mysqld, 3240
master-retry-count option
  mysqld, 3223

master-verify-checksum option
  mysqld, 3269
@master_binlog_checksum user-defined variable, 4924
master_info_repository system variable, 3242
MASTER_POS_WAIT(), 2223, 2504
master_verify_checksum system variable, 3288
MATCH ... AGAINST(), 1997
matching
  patterns, 294
materialization
  common table expressions, 1512, 1597
  derived tables, 1512, 1597
  subqueries, 1506
  view references, 1512, 1597
math, 2228
mathematical functions, 1967
MAX(), 2179
MAX(DISTINCT), 2179
max-allowed-packet option
  mysql, 393
  mysqldump, 457
  mysqlpump, 480
  mysql_upgrade, 381
max-binlog-dump-events option
  mysqld, 3269
max-join-size option
  mysql, 393
max-record-length option
  myisamchk, 524
max-relay-log-size option
  mysqld, 3223
max-rows option
  ndb_import, 3999
MaxAllocate, 3796
MaxBufferedEpochBytes, 3820
MaxBufferedEpochs, 3820
MaxDiskWriteSpeed, 3822
MaxDiskWriteSpeedOtherNodeRestart, 3823
MaxDiskWriteSpeedOwnRestart, 3823
MaxDMLOperationsPerTransaction, 3788
MaxFKBuildBatchSize, 3793
--maximum option prefix, 327
maximums
  maximum columns per table, 1550
  maximum number of databases, 1549, 2277
  maximum number of tables, 1549, 2312
  maximum row size, 1551
  maximum tables per join, 2431
  maximum tables per view, 4367
  table size, 1549
MaxLCPStartDelay, 3800
MaxNoOfAttributes, 3803
MaxNoOfConcurrentIndexOperations, 3789
MaxNoOfConcurrentOperations, 3787

MaxNoOfConcurrentScans, 3793
MaxNoOfConcurrentSubOperations, 3807
MaxNoOfConcurrentTransactions, 3786
MaxNoOfExecutionThreads
  ndbmtd, 3840
MaxNoOfFiredTriggers, 3789
MaxNoOfLocalOperations, 3788
MaxNoOfLocalScans, 3794
MaxNoOfOpenFiles, 3799
MaxNoOfOrderedIndexes, 3804
MaxNoOfSavedMessages, 3799
MaxNoOfSubscribers, 3806
MaxNoOfSubscriptions, 3806
MaxNoOfTables, 3804
MaxNoOfTriggers, 3805
MaxNoOfUniqueHashIndexes, 3805
MaxParallelCopyInstances, 3794
MaxParallelScansPerFragment, 3795
MaxReorgBuildBatchSize, 3795
MaxScanBatchSize, 3866
MaxStartFailRetries, 3858
MaxUIBuildBatchSize, 3796
max_allowed_packet
  and replication, 3377
max_allowed_packet system variable, 769
max_binlog_cache_size system variable, 3288
max_binlog_size system variable, 3289
max_binlog_stmt_cache_size system variable, 3290
max_connections system variable, 770
MAX_CONNECTIONS_PER_HOUR, 1161
max_connect_errors system variable, 769
max_delayed_threads system variable, 770
max_digest_length system variable, 771
max_error_count system variable, 771
max_execution_time system variable, 772
Max_execution_time_exceeded status variable, 905
Max_execution_time_set status variable, 905
Max_execution_time_set_failed status variable, 905
max_heap_table_size system variable, 772
MAX_INDEXES option
  CMake, 215
max_insert_delayed_threads system variable, 773
max_join_size system variable, 413, 773
max_length_for_sort_data system variable, 774
max_points_in_geometry system variable, 774
max_prepared_stmt_count system variable, 774
MAX_QUERIES_PER_HOUR, 1161
max_relay_log_size system variable, 3243
MAX_ROWS
  and DataMemory (NDB Cluster), 3782
  and NDB Cluster, 4260
  NDB Cluster, 2325
max_seeks_for_key system variable, 775
max_sort_length system variable, 775

max_sp_recursion_depth system variable, 776
max_tmp_tables system variable, 776
MAX_UPDATES_PER_HOUR, 1161
MAX_USER_CONNECTIONS, 1161
max_user_connections system variable, 776
max_write_lock_count system variable, 777
MBR, 2108
MBRContains(), 2108
MBRCoveredBy(), 2109
MBRCovers(), 2109
MBRDisjoint(), 2109
MBREquals(), 2109
MBRIntersects(), 2109
MBROverlaps(), 2110
MBRTouches(), 2110
MBRWithin(), 2110
MD5(), 2060
MDL, 6605
measured-load option
  ndb_top, 4059
mecab_rc_file system variable, 777
medium-check option
  myisamchk, 523
  mysqlcheck, 432
MEDIUMBLOB data type, 1827
MEDIUMINT data type, 1820
MEDIUMTEXT data type, 1827
MEMBER OF(), 2132
membership
  ndbinfo table, 4142
memcached, 3073, 6605
MEMCACHED_HOME option
  CMake, 224
MEMCACHED_SASL_PWDB environment variable, 3087
memcapable command, 3075
memlock option
  mysqld, 686
memory allocation library, 184, 357
MEMORY storage engine, 3119, 3133
  and replication, 3378
  optimization, 1527
memory usage
  myisamchk, 532
memory use, 1635
  in NDB Cluster, 3677
  monitoring, 1638
  Performance Schema, 4487
memoryusage
  ndbinfo table, 4144
memory_by_host_by_current_bytes view
  sys schema, 4718
memory_by_thread_by_current_bytes view
  sys schema, 4718
memory_by_user_by_current_bytes view

sys schema, 4719
memory_global_by_current_bytes view
sys schema, 4720
memory_global_total view
sys schema, 4720
memory_per_fragment
ndbinfo table, 4145
memory_summary_by_account_by_event_name table
performance_schema, 4643
memory_summary_by_host_by_event_name table
performance_schema, 4643
memory_summary_by_thread_by_event_name table
performance_schema, 4643
memory_summary_by_user_by_event_name table
performance_schema, 4643
memory_summary_global_by_event_name table
performance_schema, 4643
MemReportFrequency, 3831
merge, 6606
MERGE storage engine, 3119, 3143
MERGE tables
defined, 3143
merging
common table expressions, 1512
derived tables, 1512
view references, 1512
merging JSON values, 1880
metadata
database, 4370
database object, 1737
InnoDB, 4428
stored routines, 4335
triggers, 4340
views, 4354
metadata lock, 6606
metadata locking, 1626, 4603
metadata log, 987
metadata_locks table
performance_schema, 4603
metadata_locks_cache_size system variable, 778
metadata_locks_hash_instances system variable, 778
methods
locking, 1622
metrics counter, 6606
metrics view
sys schema, 4720
mgmd (NDB Cluster)
defined, 3654
(see also management node (NDB Cluster))
MICROSECOND(), 1985
MID(), 1940
midpoint insertion, 2845
midpoint insertion strategy, 6606
MIN(), 2179

MIN(DISTINCT), 2179
MinDiskWriteSpeed, 3823
MinFreePct, 3782, 3785
mini-transaction, 6606
minimum bounding rectangle, 2108
minus
  unary (-), 1965
MINUTE(), 1985
min_examined_row_limit system variable, 779
mirror sites, 79
miscellaneous functions, 2213
mixed statements (Replication), 3386
mixed-mode insert, 6606
MOD (modulo), 1971
MOD(), 1971
modes
  batch, 301
modify option
  MySQLInstallerConsole, 126
modulo (%), 1971
modulo (MOD), 1971
monitor
  terminal, 277
monitor option
  ndb_import, 4000
monitoring, 1043, 1638, 2729, 2733, 2764, 2802, 2843, 2851, 2881, 3055, 3057, 5051
  multi-source replication, 3196
  threads, 1653
Monitors, 3061
  enabling, 3061
  InnoDB, 3112
  output, 3063
MONTH(), 1986
MONTHNAME(), 1986
.MRG file, 6605
multi mysqld, 363
multi-column indexes, 1527
multi-core, 6607
multi-master replication
  in NDB Cluster, 4237, 4241
Multi-Range Read
  optimization, 1480
multi-source replication, 3194
  adding binary log master, 3195
  adding GTID master, 3195
  configuring, 3195
  error messages, 3198
  monitoring, 3196
  overview, 3194
  performance schema, 3197
  resetting slave, 3196
  starting slave, 3196
  stopping slave, 3196
  tutorials, 3194

multi-valued indexes, 2288
multibyte character sets, 5629
multibyte characters, 1799
MULTILINESTRING data type, 1855
MultiLineString(), 2084
multiple buffer pools, 2844
multiple servers, 1055
multiple-part index, 2282
multiplication (*), 1965
MULTIPOINT data type, 1855
MultiPoint(), 2084
MULTIPOLYGON data type, 1855
MultiPolygon(), 2085
mutex, 6607
mutex wait
  monitoring, 3057
mutex_instances table
  performance_schema, 4537
MUTEX_TYPE option
  CMake, 215
MVCC, 6607
MVCC (multi-version concurrency control), 2724
My
  derivation, 9
my.cnf, 6607
  and NDB Cluster, 3728, 3759, 3760
  in NDB Cluster, 4084
my.cnf option file, 3364
my.ini, 6607
mycnf option
  ndb_config, 3975
  ndb_mgmd, 3962
.MYD file, 6605
.MYI file, 6605
MyISAM
  compressed tables, 534, 3130
  converting tables to InnoDB, 2740
MyISAM key cache, 1615
MyISAM storage engine, 3119, 3124
myisam-block-size option
  mysqld, 687
myisamchk, 316, 516
  analyze option, 525
  backup option, 523
  block-search option, 525
  character-sets-dir option, 523
  check option, 522
  check-only-changed option, 522
  correct-checksum option, 523
  data-file-length option, 523
  debug option, 519
  defaults-extra-file option, 520
  defaults-file option, 520
  defaults-group-suffix option, 520

description option, 525
example output, 526
extend-check option, 522, 523
fast option, 523
force option, 523, 524
help option, 519
HELP option, 519
information option, 523
keys-used option, 524
max-record-length option, 524
medium-check option, 523
no-defaults option, 520
no-symlinks option, 524
options, 519
parallel-recover option, 524
print-defaults option, 520
quick option, 524
read-only option, 523
recover option, 524
safe-recover option, 524
set-auto-increment[option, 525
set-collation option, 525
silent option, 520
sort-index option, 525
sort-records option, 525
sort-recover option, 525
tmpdir option, 525
unpack option, 525
update-state option, 523
verbose option, 520
version option, 521
wait option, 521
myisamlog, 316, 533
myisampack, 316, 534, 2349, 3130
backup option, 535
character-sets-dir option, 535
debug option, 535
force option, 535
help option, 535
join option, 535
silent option, 535
test option, 535
tmpdir option, 535
verbose option, 536
version option, 536
wait option, 536
myisam_block_size myisamchk variable, 521
myisam_data_pointer_size system variable, 779
myisam_ftdump, 316, 514
count option, 515
dump option, 515
help option, 515
length option, 516
stats option, 516

verbose option, 516
myisam_max_sort_file_size system variable, 780, 3127
myisam_mmap_size system variable, 780
myisam_recover_options system variable, 781, 3127
myisam_repair_threads system variable, 782
myisam_sort_buffer_size myisamchk variable, 521
myisam_sort_buffer_size system variable, 782, 3127
myisam_stats_method system variable, 783
myisam_use_mmap system variable, 783
MySQL
defined, 4
forums, 51
introduction, 4
pronunciation, 6
upgrading, 376
websites, 51
mysql, 315, 385, 6607
auto-rehash option, 389
auto-vertical-output option, 389
batch option, 389
binary-as-hex option, 389
binary-mode option, 389
bind-address option, 389
character-sets-dir option, 389
charset command, 400
clear command, 401
column-names option, 389
column-type-info option, 389
comments option, 390
compress option, 390
compression-algorithms option, 390
connect command, 401
connect-expired-password option, 390
connect-timeout option, 390
database option, 390
debug option, 390
debug-check option, 390
debug-info option, 391
default-auth option, 391
default-character-set option, 391
defaults-extra-file option, 391
defaults-file option, 391
defaults-group-suffix option, 391
delimiter command, 401
delimiter option, 391
disable named commands, 391
edit command, 401
ego command, 401
enable-cleartext-plugin option, 392
execute option, 392
exit command, 401
force option, 392
get-server-public-key option, 392
go command, 401

help command, 400
help option, 389
histignore option, 392
host option, 392
html option, 392
i-am-a-dummy option, 396
ignore-spaces option, 392
init-command option, 392
line-numbers option, 393
local-infile option, 393
login-path option, 393
max-allowed-packet option, 393
max-join-size option, 393
named-commands option, 393
net-buffer-length option, 393
no-auto-rehash option, 393
no-beep option, 393
no-defaults option, 393
nopager command, 402
notee command, 402
nowarning command, 402
one-database option, 394
pager command, 402
pager option, 394
password option, 394
pipe option, 394
plugin-dir option, 395
port option, 395
print command, 402
print-defaults option, 395
prompt command, 402
prompt option, 395
protocol option, 395
quick option, 395
quit command, 402
raw option, 395
reconnect option, 396
rehash command, 402
resetconnection command, 402
safe-updates option, 396
secure-auth option, 396
select-limit option, 396
server-public-key-path option, 396
shared-memory-base-name option, 396
show-warnings option, 396
sigint-ignore option, 397
silent option, 397
skip-column-names option, 397
skip-line-numbers option, 397
socket option, 397
source command, 403
SSL options, 397
status command, 403
syslog option, 398

system command, 403
table option, 398
tee command, 403
tee option, 398
tls-ciphersuites option, 398
tls-version option, 398
unbuffered option, 398
use command, 404
user option, 398
verbose option, 398
version option, 398
vertical option, 399
wait option, 399
warnings command, 404
xml option, 399
zstd-compression-level option, 399
MySQL APT Repository, 160, 268
MySQL binary distribution, 78
MYSQL C type, 4794
mysql client parser
versus mysqld parser, 414
MySQL Cluster Manager
and ndb_mgm, 4073
mysql command options, 385
mysql commands
list of, 400
MySQL Data Dictionary, 2709
mysql database
gtid_executed table, 3175
MySQL Dolphin name, 9
MySQL Enterprise Audit, 1312, 5053
MySQL Enterprise Backup, 5052, 6607
Group Replication, 3456
MySQL Enterprise Data Masking and De-Identification, 5054
MySQL Enterprise Data Masking and De-Identification plugin
components, 1400
MySQL Enterprise Data Masking and De-Identification UDFs
gen_blacklist(), 1413
gen_dictionary(), 1414
gen_dictionary_drop(), 1414
gen_dictionary_load(), 1415
gen_range(), 1411
gen_rnd_email(), 1411
gen_rnd_pan(), 1412
gen_rnd_ssn(), 1412
gen_rnd_us_phone(), 1413
mask_inner(), 1407
mask_outer(), 1408
mask_pan(), 1409
mask_pan_relaxed(), 1409
mask_ssn(), 1410
MySQL Enterprise Encryption, 5053
MySQL Enterprise Firewall, 1386, 5054
installing, 1387

using, 1390
MySQL Enterprise Monitor, 5051
MySQL Enterprise Security, 1205, 1216, 1221, 5053
MySQL Enterprise Thread Pool, 999, 5054
components, 999
installing, 1001
MySQL Enterprise Transparent Data Encryption, 2795
MySQL history, 9
mysql history file, 406
MySQL Installer, 101
MySQL name, 9
MySQL privileges
and NDB Cluster, 4173
mysql prompt command, 404
MySQL server
mysqld, 354, 582
MySQL Shell
InnoDB cluster, 3593
MySQL SLES Repository, 160, 269
mysql source (command for reading from text files), 303, 410
MySQL source distribution, 78
MySQL storage engines, 3119
MySQL system tables
and NDB Cluster, 4173
and replication, 3379
MySQL version, 79
MySQL Yum Repository, 155, 267
mysql \. (command for reading from text files), 303, 410
mysql.gtid_executed table, 3175
and RESET MASTER, 3175
compression, 3176
thread/sql/compress_gtid_table, 3177
mysql.ndb_binlog_index table, 4221
(see also NDB Cluster replication)
mysql.server, 314, 361
basedir option, 363
datadir option, 363
pid-file option, 363
service-startup-timeout option, 363
mysql.slave_master_info table, 3317
mysql.slave_relay_log_info table, 3317
mysql.sock
protection, 5638
mysqladmin, 315, 415, 2277, 2372, 2677, 2684, 2691, 2696
bind-address option, 420
character-sets-dir option, 420
compress option, 420
compression-algorithms option, 420
count option, 421
debug option, 421
debug-check option, 421
debug-info option, 421
default-auth option, 421
default-character-set option, 421

defaults-extra-file option, 421
defaults-file option, 421
defaults-group-suffix option, 421
enable-cleartext-plugin option, 422
force option, 422
get-server-public-key option, 422
help option, 420
host option, 422
login-path option, 422
no-beep option, 422
no-defaults option, 422
password option, 422
pipe option, 423
plugin-dir option, 423
port option, 423
print-defaults option, 423
protocol option, 423
relative option, 423
secure-auth option, 423
server-public-key-path option, 423
shared-memory-base-name option, 424
show-warnings option, 424
silent option, 424
sleep option, 424
socket option, 424
SSL options, 424
tls-ciphersuites option, 425
tls-version option, 425
user option, 425
verbose option, 425
version option, 425
vertical option, 425
wait option, 425
zstd-compression-level option, 425
mysqladmin command options, 418
mysqladmin option
mysqld_multi, 365
mysqlbackup command, 6607
mysqlbinlog, 316, 547
base64-output option, 551
bind-address option, 552
binlog-row-event-max-size option, 552
character-sets-dir option, 552
compress option, 552
compression-algorithms option, 552
connection-server-id option, 553
database option, 553
debug option, 554
debug-check option, 554
debug-info option, 554
default-auth option, 554
defaults-extra-file option, 554
defaults-file option, 554
defaults-group-suffix option, 554

disable-log-bin option, 555
exclude-gtids option, 555
force-if-open option, 555
force-read option, 555
get-server-public-key option, 555
help option, 551
hexdump option, 555
host option, 555
idempotent option, 555
include-gtids option, 556
local-load option, 556
login-path option, 556
no-defaults option, 556
offset option, 556
open-files-limit option, 556
password option, 556
plugin-dir option, 557
port option, 557
print-defaults option, 557
print-table-metadata option, 557
protocol option, 557
raw option, 557
read-from-remote-master option, 557
read-from-remote-server option, 558
result-file option, 558
rewrite-db option, 558
secure-auth option, 558
server-id option, 558
server-id-bits option, 559
server-public-key-path option, 559
set-charset option, 559
shared-memory-base-name option, 559
short-form option, 559
skip-gtids option, 559
socket option, 560
SSL options, 560
start-datetime option, 560
start-position option, 560
stop-datetime option, 561
stop-never option, 561
stop-never-slave-server-id option, 561
stop-position option, 561
tls-ciphersuites option, 561
tls-version option, 561
to-last-log option, 561
user option, 562
verbose option, 562
verify-binlog-checksum option, 562
version option, 562
zstd-compression-level option, 562
mysqlcheck, 315, 426
all-databases option, 429
all-in-1 option, 429
analyze option, 429

auto-repair option, 429
bind-address option, 429
character-sets-dir option, 430
check option, 430
check-only-changed option, 430
check-upgrade option, 430
compress option, 430
compression-algorithms option, 430
databases option, 430
debug option, 430
debug-check option, 430
debug-info option, 430
default-auth option, 431
default-character-set option, 430
defaults-extra-file option, 431
defaults-file option, 431
defaults-group-suffix option, 431
enable-cleartext-plugin option, 431
extended option, 431
fast option, 431
force option, 431
get-server-public-key option, 431
help option, 429
host option, 432
login-path option, 432
medium-check option, 432
no-defaults option, 432
optimize option, 432
password option, 432
pipe option, 433
plugin-dir option, 433
port option, 433
print-defaults option, 433
protocol option, 433
quick option, 433
repair option, 433
secure-auth option, 433
server-public-key-path option, 433
shared-memory-base-name option, 434
silent option, 434
skip-database option, 434
socket option, 434
SSL options, 434
tables option, 435
tls-ciphersuites option, 435
tls-version option, 435
use-frm option, 435
user option, 435
verbose option, 435
version option, 435
write-binlog option, 435
zstd-compression-level option, 435
mysqld, 314, 6607
abort-slave-event-count option, 3239

allow-suspicious-udfs option, 675
ansi option, 675
as NDB Cluster process, 3872, 4083
audit-log option, 1371
basedir option, 675
binlog-checksum option, 3268
binlog-do-db option, 3266
binlog-ignore-db option, 3268
binlog-row-event-max-size option, 3264
character-set-client-handshake option, 675
chroot option, 676
command options, 673
console option, 676
core-file option, 676
daemonize option, 676
datadir option, 677
ddl-rewriter option, 1018
debug option, 677
debug-sync-timeout option, 678
default-time-zone option, 678
defaults-extra-file option, 678
defaults-file option, 678
defaults-group-suffix option, 679
des-key-file option, 679
disconnect-slave-event-count option, 3239
early-plugin-load option, 679
exit codes, 941
exit-info option, 680
external-locking option, 680
flush option, 680
gdb option, 681
help option, 674
initialize option, 681
initialize-insecure option, 682
innodb option, 2931
innodb-status-file option, 2931
install option, 682
install-manual option, 682
keyring-migration-destination option, 1304
keyring-migration-host option, 1304
keyring-migration-password option, 1304
keyring-migration-port option, 1305
keyring-migration-socket option, 1305
keyring-migration-source option, 1305
keyring-migration-user option, 1306
language option, 682
large-pages option, 683
lc-messages option, 683
lc-messages-dir option, 684
local-service option, 684
log-bin option, 3264
log-bin-index option, 3265
log-error option, 684
log-isam option, 685

log-raw option, 685
log-short-format option, 685
log-tc option, 686
log-tc-size option, 686
log-warnings option, 686
log_slow_admin_statements system variable, 762
log_slow_slave_statements system variable, 3241
master-info-file option, 3222
master-info-repository option, 3240
master-retry-count option, 3223
master-verify-checksum option, 3269
max-binlog-dump-events option, 3269
max-relay-log-size option, 3223
memlock option, 686
myisam-block-size option, 687
MySQL server, 354, 582
ndb-allow-copying-alter-table option, 3872
ndb-batch-size option, 3873
ndb-blob-read-batch-bytes option, 3874
ndb-blob-write-batch-bytes option, 3875
ndb-cluster-connection-pool option, 3873
ndb-cluster-connection-pool-nodeids option, 3874
ndb-connectstring option, 3876
ndb-log-apply-status, 3878
ndb-log-empty-epochs, 3879
ndb-log-empty-update, 3879
ndb-log-exclusive-reads, 3880
ndb-log-orig, 3880
ndb-log-transaction-id, 3881
ndb-nodeid, 3883
ndbcluster option, 3872
no-dd-upgrade option, 688
no-defaults option, 687
no-monitor option, 688
old-style-user-limits option, 689
performance-schema-consumer-events-stages-current option, 4665
performance-schema-consumer-events-stages-history option, 4665
performance-schema-consumer-events-stages-history-long option, 4665
performance-schema-consumer-events-statements-current option, 4665
performance-schema-consumer-events-statements-history option, 4665
performance-schema-consumer-events-statements-history-long option, 4665
performance-schema-consumer-events-transactions-current option, 4665
performance-schema-consumer-events-transactions-history option, 4665
performance-schema-consumer-events-transactions-history-long option, 4665
performance-schema-consumer-events-waits-current option, 4666
performance-schema-consumer-events-waits-history option, 4666
performance-schema-consumer-events-waits-history-long option, 4666
performance-schema-consumer-global-instrumentation option, 4666
performance-schema-consumer-statements-digest option, 4666
performance-schema-consumer-thread-instrumentation option, 4666
performance-schema-consumer-xxx option, 4665
performance-schema-instrument option, 4665
plugin option prefix, 690
plugin-load option, 689

plugin-load-add option, 690
port option, 691
port-open-timeout option, 691
print-defaults option, 691
relay-log-info-repository option, 3240
relay-log-purge option, 3224
relay-log-recovery option, 3224
relay-log-space-limit option, 3225
remove option, 692
replicate-do-db option, 3225
replicate-do-table option, 3228
replicate-ignore-db option, 3227
replicate-ignore-table option, 3229
replicate-rewrite-db option, 3230
replicate-same-server-id option, 3231
replicate-wild-do-table option, 3231
replicate-wild-ignore-table option, 3232
role in NDB Cluster (see SQL Node (NDB Cluster))
safe-user-create option, 692
secure-auth option, 692
server-id option, 3204
server-id-bits option, 3887
server_uuid variable, 3205
show-slave-auth-info option, 3213
skip-concurrent-insert option, 692
skip-grant-tables option, 692
skip-host-cache option, 693
skip-innodb option, 693, 2932
skip-name-resolve option, 694
skip-ndbcluster option, 3887
skip-networking option, 694
skip-show-database option, 695
skip-slave-start option, 3235
skip-stack-trace option, 696
skip-symbolic-links option, 695
slave-checkpoint-group option, 3233
slave-checkpoint-period option, 3234
slave-load-tmpdir option, 3235
slave-max-allowed-packet, 3236
slave-net-timeout option, 3236
slave-parallel-type, 3237
slave-parallel-workers option, 3234
slave-pending-jobs-size-max option, 3235
slave-rows-search-algorithms, 3238
slave-skip-errors option, 3238
slave-sql-verify-checksum option, 3239
slow-start-timeout option, 696
socket option, 696
sporadic-binlog-dump-fail option, 3269
sql-mode option, 696
SSL options, 694
standalone option, 694
starting, 1073
super-large-pages option, 695

symbolic-links option, 695
sysdate-is-now option, 699
tc-heuristic-recover option, 700
temp-pool option, 700
tmpdir option, 701
transaction-isolation option, 700
transaction-read-only option, 701
upgrade option, 701
user option, 703
validate-config option, 703
validate-password option, 1273
verbose option, 704
version option, 704
mysqld (NDB Cluster), 3946
mysqld option
malloc-lib, 357
mysqld_multi, 365
mysqld_safe, 358
mysqld options, 583
gtid-executed-compression-period, 3293
mysqld options and variables
NDB Cluster, 3871
mysqld parser
versus mysql client parser, 414
mysqld system variables, 583
mysqld-auto.cnf option file, 319, 321, 798, 861, 865, 883, 887, 1378, 2626, 2699, 4608
mysqld-safe-log-timestamps option
mysqld_safe, 357
mysqld-version option
mysqld_safe, 358
mysqldump, 273, 315, 436, 6607
add-drop-database option, 447
add-drop-table option, 447
add-drop-trigger option, 447
add-locks option, 458
all-databases option, 455
all-tablespaces option, 447
allow-keywords option, 447
apply-slave-statements option, 449
bind-address option, 442
character-sets-dir option, 449
column-statistics option, 457
comments option, 447
compact option, 452
compatible option, 452
complete-insert option, 453
compress option, 442
compression-algorithms option, 442
create-options option, 453
databases option, 455
debug option, 448
debug-check option, 448
debug-info option, 448
default-auth option, 443

default-character-set option, 449
defaults-extra-file option, 446
defaults-file option, 446
defaults-group-suffix option, 446
delete-master-logs option, 449
disable-keys option, 457
dump-date option, 448
dump-slave option, 449
enable-cleartext-plugin option, 443
events option, 455
extended-insert option, 457
fields-enclosed-by option, 453, 465
fields-escaped-by option, 453, 465
fields-optionally-enclosed-by option, 453, 465
fields-terminated-by option, 453, 465
flush-logs option, 458
flush-privileges option, 458
force option, 448
get-server-public-key option, 443
help option, 448
hex-blob option, 453
host option, 443
ignore-error option, 456
ignore-table option, 456
include-master-host-port option, 450
insert-ignore option, 457
lines-terminated-by option, 453, 466
lock-all-tables option, 458
lock-tables option, 458
log-error option, 448
login-path option, 443
master-data option, 450
max-allowed-packet option, 457
net-buffer-length option, 457
network-timeout option, 457
no-autocommit option, 459
no-create-db option, 447
no-create-info option, 447
no-data option, 456
no-defaults option, 446
no-set-names option, 449
no-tablespaces option, 447
opt option, 457
order-by-primary option, 459
password option, 443
pipe option, 444
plugin-dir option, 444
port option, 444
print-defaults option, 446
problems, 461, 4367
protocol option, 444
quick option, 458
quote-names option, 453
replace option, 447

result-file option, 453
routines option, 456
secure-auth option, 444
server-public-key-path option, 444
set-charset option, 449
set-gtid-purged option, 450
shared-memory-base-name option, 459
show-create-skip-secondary-engine option, 453
single-transaction option, 459
skip-comments option, 448
skip-opt option, 458
socket option, 444
SSL options, 445
tab option, 453
tables option, 456
tls-ciphersuites option, 445
tls-version option, 445
triggers option, 456
tz-utc option, 454
user option, 445
using for backups, 1429
verbose option, 448
version option, 449
views, 461, 4367
where option, 456
workarounds, 461, 4367
xml option, 454
zstd-compression-level option, 445
mysqldumpslow, 316, 571
debug option, 572
help option, 572
verbose option, 573
mysqld_multi, 314, 363
defaults-extra-file option, 364
defaults-file option, 364
example option, 365
help option, 365
log option, 365
mysqladmin option, 365
mysqld option, 365
no-defaults option, 364
no-log option, 365
password option, 365
silent option, 365
tcp-ip option, 365
user option, 366
verbose option, 366
version option, 366
MYSQLD_OPTS environment variable, 184
mysqld_safe, 314, 355
basedir option, 356
core-file-size option, 356
datadir option, 356
defaults-extra-file option, 356

defaults-file option, 357
help option, 356
ledir option, 357
log-error option, 357
malloc-lib option, 357
mysqld option, 358
mysqld-safe-log-timestamps option, 357
mysqld-version option, 358
nice option, 358
no-defaults option, 359
open-files-limit option, 359
pid-file option, 359
plugin-dir option, 359
port option, 359
skip-kill-mysqld option, 359
skip-syslog option, 359
socket option, 359
syslog option, 359
syslog-tag option, 359
timezone option, 360
user option, 360
mysqlimport, 273, 315, 461, 2400
bind-address option, 464
character-sets-dir option, 464
columns option, 464
compress option, 464
compression-algorithms option, 464
debug option, 464
debug-check option, 464
debug-info option, 464
default-auth option, 464
default-character-set option, 464
defaults-extra-file option, 465
defaults-file option, 465
defaults-group-suffix option, 465
delete option, 465
enable-cleartext-plugin option, 465
force option, 465
get-server-public-key option, 465
help option, 464
host option, 466
ignore option, 466
ignore-lines option, 466
local option, 466
lock-tables option, 466
login-path option, 466
low-priority option, 466
no-defaults option, 466
password option, 467
pipe option, 467
plugin-dir option, 467
port option, 467
print-defaults option, 467
protocol option, 467

replace option, 467
secure-auth option, 468
server-public-key-path option, 468
shared-memory-base-name option, 468
silent option, 468
socket option, 468
SSL options, 468
tls-ciphersuites option, 469
tls-version option, 469
use-threads option, 469
user option, 469
verbose option, 469
version option, 469
zstd-compression-level option, 469
MySQLInstallerConsole, 124
configure option, 124
help option, 125
install option, 125
list option, 126
modify option, 126
remove option, 126
status option, 126
update option, 127
upgrade option, 127
mysqlpump, 315, 470
add-drop-database option, 475
add-drop-table option, 475
add-drop-user option, 475
add-locks option, 475
all-databases option, 475
bind-address option, 475
character-sets-dir option, 475
column-statistics option, 476
complete-insert option, 476
compress option, 476
compress-output option, 476
compression-algorithms option, 476
databases option, 476
debug option, 476
debug-check option, 476
debug-info option, 477
default-auth option, 477
default-character-set option, 477
default-parallelism option, 477
defaults-extra-file option, 477
defaults-file option, 477
defaults-group-suffix option, 477
defer-table-indexes option, 477
events option, 478
exclude-databases option, 478
exclude-events option, 478
exclude-routines option, 478
exclude-tables option, 478
exclude-triggers option, 478

exclude-users option, 478
extended-insert option, 478
get-server-public-key option, 478
help option, 475
hex-blob option, 479
host option, 479
include-databases option, 479
include-events option, 479
include-routines option, 479
include-tables option, 479
include-triggers option, 479
include-users option, 479
insert-ignore option, 479
log-error-file option, 480
login-path option, 480
max-allowed-packet option, 480
net-buffer-length option, 480
no-create-db option, 480
no-create-info option, 480
no-defaults option, 480
object selection, 485
parallel-schemas option, 480
parallelism, 486
password option, 481
plugin-dir option, 481
port option, 481
print-defaults option, 481
protocol option, 481
replace option, 481
restrictions, 487
result-file option, 481
routines option, 481
secure-auth option, 481
server-public-key-path option, 482
set-charset option, 482
set-gtid-purged option, 482
single-transaction option, 482
skip-definer option, 483
skip-dump-rows option, 483
socket option, 483
SSL options, 483
tls-ciphersuites option, 484
tls-version option, 484
triggers option, 484
tz-utc option, 484
user option, 484
users option, 484
version option, 484
watch-progress option, 484
zstd-compression-level option, 485
mysqlsh, 315
mysqlshow, 316, 487
bind-address option, 490
character-sets-dir option, 490

compress option, 490
compression-algorithms option, 490
count option, 490
debug option, 490
debug-check option, 490
debug-info option, 490
default-auth option, 490
default-character-set option, 490
defaults-extra-file option, 490
defaults-file option, 491
defaults-group-suffix option, 491
enable-cleartext-plugin option, 491
get-server-public-key option, 491
help option, 490
host option, 491
keys option, 491
login-path option, 491
no-defaults option, 492
password option, 492
pipe option, 492
plugin-dir option, 492
port option, 492
print-defaults option, 492
protocol option, 493
secure-auth option, 493
server-public-key-path option, 493
shared-memory-base-name option, 493
show-table-type option, 493
socket option, 493
SSL options, 493
status option, 494
tls-ciphersuites option, 494
tls-version option, 494
user option, 494
verbose option, 494
version option, 494
zstd-compression-level option, 494
mysqlslap, 316, 495
auto-generate-sql option, 498
auto-generate-sql-add-autoincrement option, 498
auto-generate-sql-execute-number option, 498
auto-generate-sql-guid-primary option, 499
auto-generate-sql-load-type option, 499
auto-generate-sql-secondary-indexes option, 499
auto-generate-sql-unique-query-number option, 499
auto-generate-sql-unique-write-number option, 499
auto-generate-sql-write-number option, 499
commit option, 499
compress option, 499
compression-algorithms option, 499
concurrency option, 499
create option, 499
create-schema option, 500
csv option, 500

debug option, 500
debug-check option, 500
debug-info option, 500
default-auth option, 500
defaults-extra-file option, 500
defaults-file option, 500
defaults-group-suffix option, 500
delimiter option, 501
detach option, 501
enable-cleartext-plugin option, 501
engine option, 501
get-server-public-key option, 501
help option, 498
host option, 501
iterations option, 501
login-path option, 501
no-defaults option, 502
no-drop option, 501
number-char-cols option, 502
number-int-cols option, 502
number-of-queries option, 502
only-print option, 502
password option, 502
pipe option, 502
plugin-dir option, 502
port option, 503
post-query option, 503
post-system option, 503
pre-query option, 503
pre-system option, 503
print-defaults option, 503
protocol option, 503
query option, 503
secure-auth option, 503
server-public-key-path option, 503
shared-memory-base-name option, 504
silent option, 504
socket option, 504
sql-mode option, 504
SSL options, 504
tls-ciphersuites option, 505
tls-version option, 505
user option, 505
verbose option, 505
version option, 505
zstd-compression-level option, 505
mysqltest
MySQL Test Suite, 4944
mysqlx, 3578
mysqlx_bind_address system variable, 3578
mysqlx_connect_timeout system variable, 3579
mysqlx_document_id_unique_prefix system variable, 3580
mysqlx_enable_hello_notice system variable, 3580
mysqlx_idle_worker_thread_timeout system variable, 3580

mysqlx_interactive_timeout system variable, 3581
mysqlx_max_allowed_packet system variable, 3581
mysqlx_max_connections system variable, 3581
mysqlx_min_worker_threads system variable, 3582
mysqlx_port system variable, 3582
mysqlx_port_open_timeout system variable, 3583
mysqlx_read_timeout system variable, 3583
mysqlx_socket system variable, 3583
mysqlx_ssl_ca system variable, 3584
mysqlx_ssl_capath system variable, 3584
mysqlx_ssl_cert system variable, 3585
mysqlx_ssl_cipher system variable, 3585
mysqlx_ssl_crl system variable, 3585
mysqlx_ssl_crlpath system variable, 3585
mysqlx_ssl_key system variable, 3586
MYSQLX_TCP_PORT environment variable, 578
MYSQLX_TCP_PORT option
  CMake, 215
MYSQLX_UNIX_ADDR option
  CMake, 215
MYSQLX_UNIX_PORT environment variable, 578
mysqlx_wait_timeout system variable, 3586
mysqlx_write_timeout system variable, 3586
mysql_acquire_locking_service_locks() C function
  locking service, 5013
mysql_affected_rows(), 4804, 4938
mysql_autocommit(), 4805
MYSQL_BIND C type, 4875
mysql_binlog_close(), 4925
mysql_binlog_fetch(), 4926
mysql_binlog_open(), 4926
mysql_change_user(), 4806
mysql_character_set_name(), 4807
mysql_clear_password authentication plugin, 1204
mysql_client_find_plugin(), 4919
mysql_client_register_plugin(), 4920
mysql_close(), 4807
mysql_commit(), 4808
mysql_config, 573
  cflags option, 574
  cxxflags option, 574
  include option, 574
  libs option, 574
  libs_r option, 574
  plugindir option, 574
  port option, 574
  socket option, 574
  variable option, 574
  version option, 574
mysql_config_editor, 316, 540
  debug option, 544
  help option, 544
  verbose option, 544
  version option, 544

mysql_config_server, 573
mysql_connect(), 4808
mysql_create_db(), 4809
MYSQL_DATADIR option
  CMake, 209
mysql_data_seek(), 4809
MYSQL_DEBUG environment variable, 317, 578, 5039
mysql_debug(), 4810
mysql_drop_db(), 4810
mysql_dump_debug_info(), 4811
mysql_eof(), 4811
mysql_errno(), 4812
mysql_error(), 4813
mysql_escape_string(), 4814
mysql_fetch_field(), 4814
mysql_fetch_fields(), 4815
mysql_fetch_field_direct(), 4815
mysql_fetch_lengths(), 4816
mysql_fetch_row(), 4817
mysql_fetch_row_nonblocking(), 4914
MYSQL_FIELD C type, 4794
mysql_field_count(), 4818, 4834
MYSQL_FIELD_OFFSET C type, 4794
mysql_field_seek(), 4819
mysql_field_tell(), 4819
mysql_firewall_mode system variable, 1397
mysql_firewall_trace system variable, 1397
mysql_free_result(), 4820
mysql_free_result_nonblocking(), 4915
mysql_get_character_set_info(), 4820
mysql_get_client_info(), 4820
mysql_get_client_version(), 4821
mysql_get_host_info(), 4821
mysql_get_option(), 4822
mysql_get_proto_info(), 4823
mysql_get_server_info(), 4823
mysql_get_server_version(), 4823
mysql_get_ssl_cipher(), 4824
MYSQL_GROUP_SUFFIX environment variable, 578
mysql_hex_string(), 4824
MYSQL_HISTFILE environment variable, 406, 578
MYSQL_HISTIGNORE environment variable, 406, 578
MYSQL_HOME environment variable, 578
MYSQL_HOST environment variable, 343, 578
mysql_info(), 2252, 2394, 2410, 2457, 4825, 4938
mysql_init(), 4825
mysql_insert_id(), 2395, 4826, 4938, 4939
mysql_keyring plugin service, 5011
mysql_keyring service, 5018
mysql_kill(), 4828
mysql_library_end(), 4829
mysql_library_init(), 4829
mysql_list_dbs(), 4830
mysql_list_fields(), 4830

mysql_list_processes(), 4831
mysql_list_tables(), 4832
mysql_load_plugin(), 4920
mysql_load_plugin_v(), 4922
MYSQL_MAINTAINER_MODE option
  CMake, 215
mysql_more_results(), 4833
mysql_native_password authentication plugin, 1194
mysql_native_password_proxy_users system variable, 784, 1160
mysql_next_result(), 4833
mysql_next_result_nonblocking(), 4915
mysql_no_login authentication plugin, 1234
mysql_num_fields(), 4834
mysql_num_rows(), 4835, 4938
MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD environment variable, 578, 2166
MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD environment variable, 578, 2166
MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD environment variable, 578, 2166
mysql_options(), 4836
mysql_options4(), 4843
mysql_password_policy plugin service, 5011
mysql_ping(), 4844
mysql_plugin_options(), 4922
MYSQL_PROJECT_NAME option
  CMake, 215
MYSQL_PS1 environment variable, 578
MYSQL_PWD environment variable, 578
mysql_query(), 4845, 4938
mysql_real_connect(), 4846
mysql_real_connect_nonblocking(), 4916
mysql_real_escape_string(), 4850
mysql_real_escape_string_quote(), 1672, 1941, 4851
mysql_real_query(), 4852
mysql_real_query_nonblocking(), 4916
mysql_refresh(), 4853
mysql_release_locking_service_locks() C function
  locking service, 5014
mysql_reload(), 4854
MYSQL_RES C type, 4794
mysql_reset_connection(), 4855
mysql_reset_server_public_key(), 4856
mysql_result_metadata(), 4856
mysql_rollback(), 4857
MYSQL_ROW C type, 4794
mysql_row_seek(), 4857
mysql_row_tell(), 4857
mysql_secure_installation, 315, 368
  defaults-extra-file option, 370
  defaults-file option, 370
  defaults-group-suffix option, 370
  help option, 370
  host option, 370
  no-defaults option, 371
  password option, 371
  port option, 371

print-defaults option, 371
protocol option, 371
socket option, 371
SSL options, 371
tls-ciphersuites option, 372
tls-version option, 372
use-default option, 372
user option, 372
mysql_select_db(), 4858
MYSQL_SERVER_AUTH_INFO plugin structure, 4992
mysql_server_end(), 4858
mysql_server_init(), 4859
mysql_session_track_get_first(), 4859
mysql_session_track_get_next(), 4864
mysql_set_character_set(), 4865
mysql_set_local_infile_default(), 4865, 4865
mysql_set_server_option(), 4867
mysql_shutdown(), 4868
mysql_sqlstate(), 4868
mysql_ssl_rsa_setup, 315, 372
datadir option, 374
help option, 374
suffix option, 375
uid option, 375
verbose option, 375
version option, 375
mysql_ssl_set(), 4869
mysql_stat(), 4870
MYSQL_STMT C type, 4875
mysql_stmt_affected_rows(), 4883
mysql_stmt_attr_get(), 4884
mysql_stmt_attr_set(), 4884
mysql_stmt_bind_param(), 4885
mysql_stmt_bind_result(), 4886
mysql_stmt_close(), 4887
mysql_stmt_data_seek(), 4888
mysql_stmt_errno(), 4888
mysql_stmt_error(), 4888
mysql_stmt_execute(), 4889
mysql_stmt_fetch(), 4892
mysql_stmt_fetch_column(), 4898
mysql_stmt_field_count(), 4898
mysql_stmt_free_result(), 4898
mysql_stmt_init(), 4899
mysql_stmt_insert_id(), 4899
mysql_stmt_next_result(), 4899
mysql_stmt_num_rows(), 4901
mysql_stmt_param_count(), 4901
mysql_stmt_param_metadata(), 4901
mysql_stmt_prepare(), 4901
mysql_stmt_reset(), 4903
mysql_stmt_result_metadata, 4903
mysql_stmt_row_seek(), 4904
mysql_stmt_row_tell(), 4904

mysql_stmt_send_long_data(), 4905
mysql_stmt_sqlstate(), 4907
mysql_stmt_store_result(), 4907
mysql_store_result(), 4871, 4938
mysql_store_result_nonblocking(), 4917
MYSQL_TCP_PORT environment variable, 317, 578, 1062, 1063
MYSQL_TCP_PORT option
  CMake, 215
MYSQL_TEST_LOGIN_FILE environment variable, 327, 541, 578
MYSQL_TEST_TRACE_CRASH environment variable, 578, 5002
MYSQL_TEST_TRACE_DEBUG environment variable, 578, 5002
mysql_thread_end(), 4918
mysql_thread_id(), 4872
mysql_thread_init(), 4918
mysql_thread_safe(), 4919
MYSQL_TIME C type, 4877
mysql_tzinfo_to_sql, 315, 375
MYSQL_UNIX_ADDR option
  CMake, 215
MYSQL_UNIX_PORT environment variable, 317, 578, 1062, 1063
mysql_upgrade, 315, 376
  bind-address option, 379
  character-sets-dir option, 380
  compress option, 380
  compression-algorithms option, 380
  debug option, 380
  debug-check option, 380
  debug-info option, 380
  default-auth option, 380
  default-character-set option, 380
  defaults-extra-file option, 380
  defaults-file option, 380
  defaults-group-suffix option, 381
  force option, 381
  get-server-public-key option, 381
  help option, 379
  host option, 381
  login-path option, 381
  max-allowed-packet option, 381
  mysql_upgrade_info file, 248, 377
  net-buffer-length option, 381
  no-defaults option, 381
  password option, 382
  pipe option, 382
  plugin-dir option, 382
  port option, 382
  print-defaults option, 382
  protocol option, 382
  server-public-key-path option, 382
  shared-memory-base-name option, 383
  skip-sys-schema option, 383
  socket option, 383
  SSL options, 383
  tls-ciphersuites option, 384

tls-version option, 384
upgrade-system-tables option, 384
user option, 384
verbose option, 384
version-check option, 384
write-binlog option, 384
zstd-compression-level option, 384
mysql_upgrade_info file
mysql_upgrade, 248, 377
mysql_use_result(), 4872
mysql_warning_count(), 4873
my_bool C type, 4794
my_key_fetch() keyring service function, 5018
my_key_generate() keyring service function, 5019
my_key_remove() keyring service function, 5019
my_key_store() keyring service function, 5020
my_plugin_log_service plugin service, 5011
my_print_defaults, 317, 575
config-file option, 575
debug option, 575
defaults-extra-file option, 575
defaults-file option, 575
defaults-group-suffix option, 576
extra-file option, 575
help option, 575
no-defaults option, 576
show option, 576
verbose option, 576
version option, 576
my_thd_scheduler plugin service, 5011
my_ulonglong C type, 4794

N

Name, 3871
named pipes, 129, 135
named-commands option
mysql, 393
named_pipe system variable, 784
named_pipe_full_access_group system variable, 784
names, 1679
case sensitivity, 1684
variables, 1720
NAME_CONST(), 2224, 4362
name_file option
comp_err, 368
naming
releases of MySQL, 78
NATIONAL CHAR data type, 1825
NATIONAL VARCHAR data type, 1826
native backup and restore
backup identifiers, 4080
native functions
adding, 5031

NATURAL INNER JOIN, 2430
NATURAL JOIN, 2430
natural key, 6608
NATURAL LEFT JOIN, 2430
NATURAL LEFT OUTER JOIN, 2430
NATURAL RIGHT JOIN, 2430
NATURAL RIGHT OUTER JOIN, 2430
NCHAR data type, 1825
NDB API counters (NDB Cluster), 4200
  scope, 4203
  status variables associated with, 4205
  types, 4204
NDB API database objects
  and NDB Cluster replication, 4214
NDB API _slave status variables
  and NDB Cluster Replication, 4212
NDB client programs
  defaults-extra-file option, 4068
  defaults-file option, 4068
  defaults-group-suffix option, 4068
  login-path option, 4069
  no-defaults option, 4070
  print-defaults option, 4070
NDB Cluster, 3650
  "quick" configuration, 3737
  administration, 3872, 3947, 3958, 3966, 3967, 4065, 4073, 4095
  and application feature requirements, 3674
  and DNS, 3686
  and INFORMATION_SCHEMA, 4174
  and IP addressing, 3686
  and MySQL privileges, 4173
  and MySQL root user, 4173, 4175
  and networking, 3660
  and transactions, 3783
  API node, 3654, 3861
  applications supported, 3673
  availability, 3672
  available platforms, 3651
  BACKUP Events, 4095
  backups, 4021, 4078, 4078, 4079, 4082, 4083
  CHECKPOINT Events, 4090
  cluster logs, 4087, 4088
  CLUSTERLOG commands, 4088
  CLUSTERLOG STATISTICS command, 4095
  commands, 3872, 3947, 3958, 3967, 4073
  compared to InnoDB, 3671, 3672, 3673, 3674
  compared to standalone MySQL Server, 3671, 3672, 3673, 3674
  concepts, 3654
  configuration, 3685, 3737, 3737, 3768, 3768, 3775, 3861, 3966, 4084
  configuration (example), 3760
  configuration changes, 4085
  configuration files, 3728, 3759
  configuration parameters, 3740, 3740, 3747, 3748, 3749
  configuring, 4082

CONNECT command, 4074
CONNECTION Events, 4090
connection string, 3766
CREATE NODEGROUP command, 4076
data node, 3654, 3775
data nodes, 3947, 3956
defining node hosts, 3768
deployment with Auto-Installer, 3688
direct connections between nodes, 3937
Disk Data tables (see NDB Cluster Disk Data)
DROP NODEGROUP command, 4077
ENTER SINGLE USER MODE command, 4075
ERROR Events, 4094
error logs, 3954
event log format, 4090
event logging thresholds, 4089
event logs, 4087, 4088
event severity levels, 4089
event types, 4087, 4090
execution threads, 3840
EXIT command, 4076
EXIT SINGLE USER MODE command, 4076
FAQ, 5070
FULLY_REPLICATED (NDB_TABLE), 2357
GCP Stop errors, 3856
general description, 3653
HELP command, 4073
HostName parameter
  and security, 4169
indirect indexes, 2354
INFO Events, 4094
information sources, 3653
insecurity of communication protocols, 4169
installation, 3685
installation (Linux), 3710
installation (Windows), 3719
installing .deb file (Linux), 3717
installing binary (Windows), 3719
installing binary release (Linux), 3710
installing from source (Linux), 3718
installing from source (Windows), 3723
installing RPM (Linux), 3713
interconnects, 3946
Java clients, 3656
JSON, 2354
large tables, 2325
log files, 3953, 3957
logging commands, 4088
management client (ndb_mgm), 3966
management commands, 4095
management node, 3654, 3768
management nodes, 3958
managing, 4070
MAX_ROWS, 2325

memory usage and recovery, 3677, 4086
mgm, 4065
mgm client, 4073
mgm management client, 4095
mgm process, 3967
mgmd, 4065
mgmd process, 3958
monitoring, 4200
multiple management servers, 4087
mysqld options and variables for, 3871
mysqld process, 3872, 4083
ndbd, 3947, 4065
ndbd process, 3947, 4097
ndbinfo_select_all, 3955
ndbmtd, 3956
ndb_mgm, 3730, 3966
ndb_mgmd process, 3958
network configuration
  and security, 4170
network transporters, 3946
networking, 3937, 3938
node failure (single user mode), 4114
node identifiers, 3942, 3942
node logs, 4087
NODELOG DEBUG command, 4078
NODERESTART Events, 4092
nodes and node groups, 3657
nodes and types, 3654
NOLOGGING (NDB_TABLE), 2356
partitioning support, 3677
partitions, 3657
PARTITION_BALANCE (NDB_TABLE), 2356
performing queries, 3731
preparing for replication, 4224
process management, 3946
PROMPT command, 4077
QUIT command, 4076
READ_BACKUP (NDB_TABLE), 2356
replicas, 3657
replication, 4211
  (see also NDB Cluster replication)
REPORT command, 4075
requirements, 3660
resetting, 4085
RESTART command, 4074
restarting, 3735
restoring backups, 4021
rolling restarts (multiple management servers), 4087
runtime statistics, 4095
SCHEMA Events, 4094
security, 4169
  and firewalls, 4170, 4172
  and HostName parameter, 4169
  and network configuration, 4170

and network ports, 4172
and remote administration, 4172
networking, 4169
security procedures, 4175
shared memory transport, 3938
SHOW command, 4074
SHUTDOWN command, 4076
shutting down, 3735
single user mode, 4075, 4114
SINGLEUSER Events, 4095
SQL node, 3654, 3861
SQL nodes, 4083
SQL statements for monitoring, 4114
START BACKUP command, 4231
START command, 4074
start phases (summary), 4071
starting, 3737
starting nodes, 3723, 3730
starting or restarting, 4071
STARTUP Events, 4091
STATISTICS Events, 4093
STATUS command, 4075
status variables, 3917
STOP command, 4074
storage requirements, 1894
thread states, 1665
trace files, 3954
transaction handling, 3680
transaction isolation level, 3679
transporters
shared memory (SHM), 3938
TCP/IP, 3937
troubleshooting backups, 4083
upgrades and downgrades, 3736, 4085
USING HASH, 2295
using tables and data, 3731
NDB Cluster 8.0, 3661
NDB Cluster Auto-Installer, 3688
adding and removing hosts, 3699
adding processes, 3704
and ndb_setup.py, 3692
and Python, 3688
and setup.bat (Windows), 3692
architecture, 3688
authentication with remote hosts, 3689
Define Cluster screen, 3694
Define Hosts screen, 3697
Define Parameters screen, 3704
Define Processes screen, 3702
Deploy Configuration screen, 3706
remote vs local hosts, 3689, 3690
removing processes, 3704
requirements, 3688
security issues, 3689

setup program, 4048
setup program (Windows), 4048
software requirements, 3688
starting, 3692
supported platforms, 3688
supported web browsers, 3688
using, 3690
Welcome screen, 3693
NDB Cluster Disk Data, 4176
creating log file groups, 4177
creating tables, 4177, 4178
creating tablespaces, 4178
dropping Disk Data objects, 4181
storage requirements, 4183
NDB Cluster How-To, 3685
NDB Cluster limitations, 3674
and differences from standard MySQL limits, 3677
binary logging, 3683
database objects, 3681
Disk Data storage, 3684
error handling and reporting, 3681
geometry data types, 3676
implementation, 3683
imposed by configuration, 3678
JSON columns, 3677
memory usage and transaction handling, 3680
multiple management servers, 3685
multiple MySQL servers, 3685
partitioning, 3677
performance, 3683
replication, 3677
resolved in current version from previous versions, 3685
syntax, 3675
transactions, 3678
unsupported features, 3682
NDB Cluster processes, 3946
NDB Cluster programs, 3946
NDB Cluster replication, 4211
and --initial option, 4218
and circular replication, 4215
and NDB API database objects, 4214
and primary key, 4217
and single point of failure, 4228
and unique keys, 4217
backups, 4230
circular replication, 4237
concepts, 4212, 4213
conflict resolution, 4241
failover, 4228, 4229
gap event, 4214
known issues, 4214
loss of connection, 4214
multi-master replication, 4237
point-in-time recovery, 4236

preparing, 4224
read conflict detection and resolution, 4253
requirements, 4213
reset-slave.pl backup automation script, 4233
restoring from backup, 4230
starting, 4226
storage engines other than NDB on slave, 4219
synchronization of master and slave, 4233
system tables used, 4221
NDB Cluster Replication
and NDB API _slave status variables, 4212
NDB Cluster replication conflict resolution
exceptions table, 4249
ndb option
ndb_perror, 4014
perror, 577
NDB statistics variables
and NDB API counters, 4206
NDB statistics variables (NDB Cluster), 4200
scope, 4203
types, 4204
NDB storage engine (see NDB Cluster)
FAQ, 5070
NDB tables
and MySQL root user, 4173
NDB utilities
security issues, 4176
NDB$CFT_CAUSE, 4251
NDB$EPOCH(), 4246
limitations, 4247
NDB$EPOCH2(), 4248
NDB$EPOCH2_TRANS(), 4249
NDB$EPOCH_TRANS(), 4246, 4248
NDB$MAX(), 4245
NDB$MAX_DELETE_WIN(), 4246
NDB$OLD, 4245
NDB$OP_TYPE, 4250
NDB$ORIG_TRANSID, 4251
ndb-allow-copying-alter-table option
mysqld, 3872
ndb-batch-size option
mysqld, 3873
ndb-blob-read-batch-bytes option
mysqld, 3874
ndb-blob-write-batch-bytes option
mysqld, 3875
ndb-cluster-connection-pool option
mysqld, 3873
ndb-cluster-connection-pool-nodeids option
mysqld, 3874
ndb-connectstring option
mysqld, 3876
ndb_config, 3975
ndb-connectstring option (NDB Cluster programs), 4069

ndb-default-column-format option (NDB Cluster), 3876
ndb-deferred-constraints option (NDB Cluster), 3877
ndb-distribution option (NDB Cluster), 3878
ndb-log-apply-status option
  mysqld, 3878
ndb-log-empty-epochs option
  mysqld, 3879
ndb-log-empty-update option
  mysqld, 3879
ndb-log-exclusive-reads option
  mysqld, 3880
ndb-log-orig option
  mysqld, 3880
ndb-log-transaction-id option
  mysqld, 3881
ndb-log-update-as-write (mysqld option), 4242
ndb-log-update-minimal option (NDB Cluster), 3882
ndb-mgmd-host option (NDB Cluster programs), 4069
ndb-mgmd-host option (NDB Cluster), 3882
ndb-nodegroup-map option
  ndb_restore, 4032
ndb-nodeid option
  mysqld, 3883
ndb-nodeid option (NDB Cluster programs), 4069
ndb-optimized-node-selection option (NDB Cluster), 4070
ndbcluster option
  mysqld, 3872
NDBCLUSTER storage engine (see NDB Cluster)
ndbd, 3946, 3947
ndbd (NDB Cluster)
  defined, 3654
    (see also data node (NDB Cluster))
ndbinfo database, 4117
  and query cache, 4120
  basic usage, 4121
  determining support for, 4117
ndbinfo_select_all, 3946, 3955
ndbmtd, 3946, 3956
  configuration, 3844, 3844
  MaxNoOfExecutionThreads, 3840
  trace files, 3957, 3957
ndb_apply_status table (NDB Cluster replication), 4223, 4224, 4229
  (see also NDB Cluster replication)
ndb_binlog_index table
  system table, 947, 4221
ndb_binlog_index table (NDB Cluster replication), 4221, 4229
  (see also NDB Cluster replication)
ndb_blob_tool, 3946, 3968
  check-orphans option, 3969
  database option, 3969
  delete-orphans option, 3969
  dump-file option, 3970
  verbose option, 3970
ndb_config, 3946, 3971

config-file option, 3973
configinfo option, 3972
config_from_node option, 3973
connections option, 3974
diff-default option, 3974
fields option, 3974
host option, 3974
mycnf option, 3975
ndb-connectstring option, 3975
nodeid option, 3975
nodes option, 3975
query option, 3976, 3976
query-all option, 3976
rows option, 3976
system option, 3977
type option, 3977
usage option, 3977
version option, 3977
xml option, 3977
ndb_cpcd, 3946
ndb_delete_all, 3946, 3980
transactional option, 3981
ndb_desc, 3946, 3981
blob-info option, 3987
database option, 3987
extra-node-info option, 3987
extra-partition-info option, 3987
retries option, 3987
table option, 3987
unqualified option, 3987
ndb_drop_index, 3946, 3987
ndb_drop_table, 3946, 3988
ndb_error_reporter, 3946, 3989
options, 3990
ndb_import, 3946, 3990
abort-on-error option, 3994
ai-increment option, 3994
ai-offset option, 3995
ai-prefetch-sz option, 3995
connections option, 3995
continue option, 3995
db-workers option, 3996
errins-delay option, 3996
errins-type option, 3996
fields-enclosed-by option, 3996
fields-escaped-by option, 3997
fields-optionally-enclosed-by option, 3997
fields-terminated-by option, 3997
idlesleep option, 3997
idlespin option, 3998
ignore-lines option, 3998
input-type option, 3998
input-workers option, 3998
keep-state option, 3999

lines-terminated-by option, 3999
log-level option, 3999
max-rows option, 3999
monitor option, 4000
no-asynch option, 4000
no-hint option, 4000
opbatch option, 4000
opbytes option, 4000
output-type option, 4001
output-workers option, 4001
pagecnt option, 4001
pagesize option, 4001
polltimeout option, 4002
rejects option, 4002
resume option, 4002
rowbatch option, 4002
rowbytes option, 4003
state-dir option, 4003
stats option, 4003
tempdelay option, 4003
temperrors option, 4004
verbose option, 4004
ndb_index_stat, 3946, 4004
example, 4004
interpreting output, 4004
ndb_join_pushdown, 3897
ndb_log_apply_status variable (NDB Cluster replication), 4229
ndb_log_empty_epochs system variable, 3898
ndb_log_empty_update system variable, 3899
ndb_log_exclusive_reads (system variable), 4254
ndb_log_exclusive_reads system variable, 3899
ndb_log_orig system variable, 3900
ndb_log_transaction_id system variable, 3900
ndb_mgm, 3946, 3966 (see mgm)
using with MySQL Cluster Manager, 4073
ndb_mgm (NDB Cluster management node client), 3730
ndb_mgmd, 3946 (see mgmd)
mycnf option, 3962
ndb_mgmd (NDB Cluster process), 3958
ndb_mgmd (NDB Cluster)
defined, 3654
(see also management node (NDB Cluster))
ndb_move_data, 3946, 4010
abort-on-error option, 4011
character-sets-dir option, 4011
database option, 4011
drop-source option, 4011
error-insert option, 4012
exclude-missing-columns option, 4012
lossy-conversions option, 4012
promote-attributes option, 4012
staging-tries option, 4012
verbose option, 4013
ndb_perror, 4013

help option, 4014
ndb option, 4014
silent option, 4014
verbose option, 4015
version option, 4014
ndb_print_backup_file, 3946, 4015
ndb_print_file, 3946, 4015
ndb_print_frag_file, 3946, 4016
ndb_print_schema_file, 3946, 4017
ndb_print_sys_file, 3946, 4017
ndb_read_backup
and NDB_TABLE, 2356
ndb_redo_log_reader, 4018
dump option, 4019
lap option, 4019
twiddle option, 4021
ndb_replication system table, 4244
ndb_restore, 4021
and circular replication, 4239
append option, 4025
backupid option, 4026
backup_path option, 4025
connect option, 4026
disable-indexes option, 4026
dont-ignore-systab-0 option, 4026
errors, 4039
exclude-databases option, 4027
exclude-intermediate-sql-tables option, 4027
exclude-missing-columns option, 4027
exclude-missing-tables option, 4027
exclude-tables option, 4028
fields-enclosed-by option, 4029
fields-optionally-enclosed-by option, 4029
fields-terminated-by option, 4029
hex option, 4029
include-databases option, 4030
include-tables option, 4030
lines-terminated-by option, 4031
lossy-conversions option, 4031
ndb-nodegroup-map option, 4032
no-binlog option, 4031
no-restore-disk-objects option, 4031
no-upgrade option, 4032
nodeid option, 4032
parallelism option, 4032
preserve-trailing-spaces option, 4033
print option, 4033
print-data option, 4033
print-log option, 4034
print-meta option, 4034
print-sql-log option, 4034
progress-frequency option, 4034
promote-attributes option, 4035
rebuild-indexes option, 4035

restore-data option, 4036
restore-epoch option, 4036
restore-meta option, 4036
restore-privilege-tables option, 4037
rewrite-database option, 4037
skip-broken-objects option, 4038
skip-table-check option, 4038
skip-unknown-objects option, 4038
tab option, 4039
typical and required options, 4024
verbose option, 4039
ndb_select_all, 3946, 4044
database option, 4045
delimiter option, 4046
descending option, 4045
disk option, 4046
gci option, 4046
gci64 option, 4046
header option, 4046
lock option, 4045
nodata option, 4046
order option, 4045
parallelism option, 4045
rowid option, 4046
tupscan option, 4046
useHexFormat option, 4046
ndb_select_count, 3946, 4047
ndb_setup.py, 3946, 4048
browser-start-page option, 4049
ca-certs-file option, 4050
cert-file option, 4050
debug-level option, 4050
help option, 4050
key-file option, 4051
no-browser option, 4051
port option, 4051
server-log-file option, 4051
server-name option, 4051
use-http option, 4052
use-https option, 4052
ndb_show_tables, 3946, 4052
database option, 4052
loops option, 4053
parsable option, 4053
show-temp-status option, 4053
type option, 4053
unqualified option, 4053
ndb_size.pl, 3946, 4053
ndb_size.pl script, 1894
NDB_STORED_USER, 1090, 4199
NDB_TABLE, 2322, 2355, 3782
ndb_top, 3946, 4056
color option, 4058
graph option, 4058

help option, 4059
host option, 4059
measured-load option, 4059
node-id option, 4059
os-load option, 4059
passwd option, 4060
password option, 4060
port option, 4060
sleep-time option, 4060
socket option, 4060
sort option, 4061
text option, 4061
user option, 4061
ndb_transid_mysql_connection_map
INFORMATION_SCHEMA table, 4394
ndb_waiter, 3946, 4062
no-contact option, 4063
not-started option, 4063
nowait-nodes option, 4063
single-user option, 4063
timeout option, 4063
wait-nodes option, 4063
negative values, 1672
neighbor page, 6608
nested queries, 2438
Nested-Loop join algorithm, 1468
nested-loop join algorithm, 1472
net-buffer-length option
mysql, 393
mysqldump, 457
mysqlpump, 480
mysql_upgrade, 381
netmask notation
in account names, 1106
network ports
and NDB Cluster, 4172
network-timeout option
mysqldump, 457
net_buffer_length system variable, 785
net_read_timeout system variable, 786
net_retry_count system variable, 786
net_write_timeout system variable, 786
new features in MySQL 8.0, 9
new features in NDB Cluster, 3661
new system variable, 787
newline (\n), 1671, 2142, 2406
next-key lock, 2810, 6608
InnoDB, 2827
NFS
InnoDB, 2754, 2833
ngram_token_size system variable, 787
nice option
mysqld_safe, 358
no matching rows, 5644

NO PAD collations, 1765, 1779, 1844
no-asynch option
  ndb_import, 4000
no-auto-rehash option
  mysql, 393
no-autocommit option
  mysqldump, 459
no-beep option
  mysql, 393
  mysqladmin, 422
no-binlog option
  ndb_restore, 4031
no-browser option
  ndb_setup.py, 4051
no-check option
  ibd2sdi, 509
  innochecksum, 511
no-contact option
  ndb_waiter, 4063
no-create-db option
  mysqldump, 447
  mysqlpump, 480
no-create-info option
  mysqldump, 447
  mysqlpump, 480
no-data option
  mysqldump, 456
no-dd-upgrade option
  mysqld, 688
no-defaults option, 327
  myisamchk, 520
  mysql, 393
  mysqladmin, 422
  mysqlbinlog, 556
  mysqlcheck, 432
  mysqld, 687
  mysqldump, 446
  mysqld_multi, 364
  mysqld_safe, 359
  mysqlimport, 466
  mysqlpump, 480
  mysqlshow, 492
  mysqlslap, 502
  mysql_secure_installation, 371
  mysql_upgrade, 381
  my_print_defaults, 576
  NDB client programs, 4070
no-drop option
  mysqlslap, 501
no-hint option
  ndb_import, 4000
no-log option
  mysqld_multi, 365
no-monitor option

mysqld, 688
no-nodeid-checks option (ndb_mgmd), 3963
no-restore-disk-objects option
ndb_restore, 4031
no-set-names option
mysqldump, 449
no-symlinks option
myisamchk, 524
no-tablespaces option
mysqldump, 447
no-upgrade option
ndb_restore, 4032
nodaemon option (ndb_mgmd), 3963
nodata option
ndb_select_all, 4046
node groups (NDB Cluster), 3657
node logs (NDB Cluster), 4087
node-id option
ndb_top, 4059
NodeGroup, 3778
NodeId, 3769, 3776, 3862
nodeid option
ndb_config, 3975
ndb_restore, 4032
NodeId1, 3932, 3942
NodeId2, 3933, 3942
NodeIdServer, 3942
NODELOG DEBUG command (NDB Cluster), 4078
NODERESTART Events (NDB Cluster), 4092
nodes
ndbinfo table, 4147
nodes option
ndb_config, 3975
NOLOGGING, 2355
NOLOGGING (NDB_TABLE)
NDB Cluster, 2356
non-locking read, 6608
non-repeatable read, 6608
nonblocking I/O, 6609
nondelimited strings, 1674
nondeterministic functions
optimization, 1498
replication, 1498
Nontransactional tables, 5643
NoOfFragmentLogFiles, 3800
NoOfFragmentLogParts, 3844
NoOfReplicas, 3779
nopager command
mysql, 402
normalized, 6609
normalized JSON values, 1879
NoSQL, 6609
NoSQL database
MySQL as a, 3529

nostart option (ndbd), 3952
nostart option (ndbmtd), 3952
NOT
  logical, 1926
NOT BETWEEN, 1923
not equal (!=), 1921
not equal (<>), 1921
NOT EXISTS
  with subqueries, 2442
NOT IN, 1924
NOT LIKE, 1950
NOT NULL
  constraint, 64
NOT NULL constraint, 6609
NOT REGEXP, 1952
not-started option
  ndb_waiter, 4063
notee command
  mysql, 402
NOTIFY_SOCKET environment variable, 184, 578
NOW(), 1986
NOWAIT, 2426
NOWAIT (START BACKUP command), 4079
nowait-nodes option
  ndb_waiter, 4063
nowait-nodes option (ndbd), 3953
nowait-nodes option (ndbmtd), 3953
nowait-nodes option (ndb_mgmd), 3963
nowarning command
  mysql, 402
NO_AUTO_VALUE_ON_ZERO SQL mode, 917
NO_BACKSLASH_ESCAPES SQL mode, 918
NO_DIR_IN_CREATE SQL mode, 918
NO_ENGINE_SUBSTITUTION SQL mode, 918
NO_UNSIGNED_SUBTRACTION SQL mode, 918
NO_ZERO_DATE SQL mode, 919
NO_ZERO_IN_DATE SQL mode, 919
NTH_VALUE(), 2198
NTILE(), 2198
NUL, 1670, 2406
NULL, 293, 5641, 6609
  ORDER BY, 1491
  testing for null, 1921, 1923, 1924, 1925, 1931
  thread state, 1658
null literal
  JSON, 1874
NULL value, 293, 1679
  ORDER BY, 1679
NULL values
  and AUTO_INCREMENT columns, 5642
  and indexes, 2314
  and TIMESTAMP columns, 5642
  vs. empty values, 5641
NULL-complemented row, 1473, 1478

null-rejected condition, 1478
NULLIF(), 1932
Numa, 3839
number-char-cols option
  mysqlslap, 502
number-int-cols option
  mysqlslap, 502
number-of-queries option
  mysqlslap, 502
numbers, 1672
NUMERIC data type, 1821
numeric data types
  storage requirements, 1894
numeric literals
  approximate-value, 1672, 2229
  exact-value, 1672, 2229
numeric precision, 1818
numeric scale, 1818
NVARCHAR data type, 1826

O

object
  JSON, 1874
objects
  stored, 4331
objects_summary_global_by_type table
  performance_schema, 4636
obtaining information about partitions, 4307
OCT(), 1941
OCTET_LENGTH(), 1941
ODBC compatibility, 827, 1821, 1914, 1925, 2315, 2432
ODBC_INCLUDES= option
  CMake, 209
ODBC_LIB_DIR option
  CMake, 209
ODirect, 3810
ODirectSyncFlag, 3811
OFF
  plugin activation option, 997
off-page column, 6610
offline_mode system variable, 787
offset option
  mysqlbinlog, 556
OGC (see Open Geospatial Consortium)
OLAP, 2181
old system variable, 788
old-style-user-limits option
  mysqld, 689
old_alter_table system variable, 788
old_passwords system variable, 789
OLTP, 6610
ON
  plugin activation option, 997

ON DUPLICATE KEY
  INSERT modifier, 2396
ON DUPLICATE KEY UPDATE, 2391
ON versus USING
  joins, 2434
one-database option
  mysql, 394
online, 6610
online DDL, 2903, 2904, 6610
  concurrency, 2918
  limitations, 2924
online location of manual, 2
online upgrades and downgrades (NDB Cluster), 4085
  order of node updates, 4086
only-print option
  mysqlslap, 502
ONLY_FULL_GROUP_BY
  SQL mode, 2187
ONLY_FULL_GROUP_BY SQL mode, 920
opbatch option
  ndb_import, 4000
opbytes option
  ndb_import, 4000
OPEN, 2529
Open Geospatial Consortium, 1853
Open Source
  defined, 5
open tables, 418, 1544
open-files-limit option
  mysqlbinlog, 556
  mysqld_safe, 359
OpenGIS, 1853
opening
  tables, 1544
Opening master dump table
  thread state, 1665
Opening mysql.ndb_apply_status
  thread state, 1665
Opening system tables
  thread state, 1658
Opening tables
  thread state, 1658
opens, 418
OpenSSL, 199, 1169
  FIPS mode, 1416
OpenSSL FIPS Object Module, 1416
open_files_limit system variable, 789
operating systems
  file-size limits, 1549
  supported, 78
operations
  arithmetic, 1965
operations_per_fragment
  ndbinfo table, 4148

operators, 1900
  arithmetic, 2043
  assignment, 1720, 1928
  bit, 2043
  cast, 1964, 2026
  logical, 1926
  precedence, 1918
  string, 1932
  string comparison, 1948
.OPT file, 6610
opt option
  mysqldump, 457
optimistic, 6611
optimization, 1446, 1559
  Batched Key Access, 1481, 1482
  benchmarking, 1651
  BLOB types, 1543
  Block Nested-Loop, 1481, 1482
  character and string types, 1543
  common table expressions, 1502
  data change statements, 1520
  data size, 1540
  DELETE statements, 1522
  derived tables, 1502
  disk I/O, 1631
  foreign keys, 1525
  full table scans, 1502
  full-text queries, 1526
  indexes, 1522
  INFORMATION_SCHEMA queries, 1516
  InnoDB tables, 1553
  INSERT statements, 1521
  many tables, 1544
  MEMORY storage engine, 1527
  MEMORY tables, 1568
  memory usage, 1635
  Multi-Range Read, 1480
  MyISAM tables, 1564
  network usage, 1642
  nondeterministic functions, 1498
  numeric types, 1542
  Performance Schema queries, 1519
  PERFORMANCE_SCHEMA, 1652
  primary keys, 1524
  REPAIR TABLE statements, 1566
  SELECT statements, 1448
  SPATIAL indexes, 1524
  spatial queries, 1527
  SQL statements, 1448
  subqueries, 1502
  subquery, 1508
  subquery materialization, 1506
  tips, 1522
  UPDATE statements, 1521

views, 1502
WHERE clauses, 1449
window functions, 1499
optimization (NDB), 1465,
optimizations, 1459
LIMIT clause, 1496
row constructors, 1501
optimize option
mysqlcheck, 432
OPTIMIZE TABLE
and partitioning, 4306
OPTIMIZE TABLE statement, 2615
optimizer, 6611
and replication, 3379
controlling, 1586
cost model, 1608
query plan evaluation, 1586
switchable optimizations, 1587
optimizer hints, 1592
optimizer statistics
for InnoDB tables, 2859
Optimizer Statistics, 2866
optimizer_prune_level system variable, 790
optimizer_search_depth system variable, 791
optimizer_switch system variable, 791, 1587
use_invisible_indexes flag, 1536
OPTIMIZER_TRACE
INFORMATION_SCHEMA table, 4396
OPTIMIZER_TRACE option
CMake, 216
optimizer_trace system variable, 794
optimizer_trace_features system variable, 794
optimizer_trace_limit system variable, 794
optimizer_trace_max_mem_size system variable, 794
optimizer_trace_offset system variable, 795
optimizing
DISTINCT, 1495
filesort, 1491, 1610
GROUP BY, 1493
LEFT JOIN, 1476
ORDER BY, 1488
outer joins, 1476
RIGHT JOIN, 1476
tables, 1442
thread state, 1659
option, 6611
option file, 6611
option files, 321, 1164
.my.cnf, 321, 322, 343, 1063, 1069, 1164
.mylogin.cnf, 321, 540
C:\my.cnf, 1063
escape sequences, 324
my.cnf, 3364
mysqld-auto.cnf, 319, 321, 798, 861, 865, 883, 887, 1378, 2626, 2699, 4608

option prefix
--disable, 327
--enable, 327
--loose, 327
--maximum, 327
--skip, 327
options
boolean, 327
CMake, 200
command-line
mysql, 385
mysqladmin, 418
myisamchk, 519
mysqld, 583
provided by MySQL, 277
replication, 3364
OR, 308, 1459
bitwise, 2053
logical, 1927
OR Index Merge optimization, 1459
Oracle compatibility, 59, 2180, 2259, 2702
Oracle Key Vault, 2795
keyring_okv keyring plugin, 1284
ORD(), 1941
ORDER BY, 289, 2265, 2423
maximum sort length, 2424
NULL, 1491
NULL value, 1679
window functions, 2203
WITH ROLLUP, 2424
ORDER BY optimization, 1488
order option
ndb_select_all, 4045
order-by-primary option
mysqldump, 459
original_commit_timestamp, 3361
original_commit_timestamp system variable, 3290
original_server_version system variable, 3218
os-load option
ndb_top, 4059
Out of resources error
and partitioned tables, 4321
OUT parameter
condition handling, 2559
out-of-range handling, 1831
outer joins
optimizing, 1476
OUTFILE, 2428
output-type option
ndb_import, 4001
output-workers option
ndb_import, 4001
out_dir option
comp_err, 368

out_file option
  comp_err, 368
OVER clause
  window functions, 2201
overflow handling, 1831
overflow page, 6611
OverloadLimit, 3934, 3942
overview, 1

P

packages
  list of, 72
PAD SPACE collations, 1765, 1779, 1844
PAD_CHAR_TO_FULL_LENGTH SQL mode, 920
page, 6611
page cleaner, 6612
page compression, 2889
page option
  innochecksum, 511
page size, 6612
  InnoDB, 2755, 2758
page-type-dump option
  innochecksum, 513
page-type-summary option
  innochecksum, 512
pagecnt option
  ndb_import, 4001
pager command
  mysql, 402
pager option
  mysql, 394
pagesize option
  ndb_import, 4001
PAM
  pluggable authentication, 1205
.par file, 6611
parallel-recover option
  myisamchk, 524
parallel-schemas option
  mysqlpump, 480
parallelism option
  ndb_restore, 4032
parameters
  server, 582
PARAMETERS
  INFORMATION_SCHEMA table, 4396
parameters table
  data dictionary table, 944
parameter_type_elements table
  data dictionary table, 944
parent events
  performance_schema, 4692
parent table, 6612

parentheses (and), 1919
parsable option
  ndb_show_tables, 4053
parser_max_mem_size system variable, 795
partial backup, 6612
partial index, 6612
partial revokes, 1110, 1127
partial updates
  and replication, 3381
partial_revokes system variable, 796
PARTITION, 4257
PARTITION BY
  window functions, 2203
PARTITION BY LIST COLUMNS, 4270
PARTITION BY RANGE COLUMNS, 4270
partition management, 4289
partition pruning, 4309
partitioning, 4257
  advantages, 4261
  and dates, 4261
  and foreign keys, 4321
  and FULLTEXT indexes, 4322
  and replication, 3379, 3382
  and SQL mode, 3382, 4319
  and subqueries, 4322
  and temporary tables, 4322, 4324
  by hash, 4278
  by key, 4281
  by linear hash, 4280
  by linear key, 4282
  by list, 4267
  by range, 4263
  COLUMNS, 4269
  concepts, 4258
  data type of partitioning key, 4322
  enabling, 4257
  functions allowed in partitioning expressions, 4329
  keys, 4260
  limitations, 4319
  operators not permitted in partitioning expressions, 4319
  operators supported in partitioning expressions, 4319
  optimization, 4308, 4309
  partitioning expression, 4260
  resources, 4258
  storage engines (limitations), 4328
  subpartitioning, 4323
  support, 4257
  support in NDB Cluster, 3677
  tables, 4257
  types, 4261
  window functions, 2203
Partitioning
  maximum number of partitions, 4321
partitioning information statements, 4307

partitioning keys and primary keys, 4324
partitioning keys and unique keys, 4324
partitions
  adding and dropping, 4289
  analyzing, 4306
  checking, 4306
  managing, 4289
  modifying, 4289
  optimizing, 4306
  repairing, 4306
  splitting and merging, 4289
  truncating, 4289
PARTITIONS
  INFORMATION_SCHEMA table, 4398
partitions (NDB Cluster), 3657
PARTITION_BALANCE, 2355, 3782
PARTITION_BALANCE (NDB_TABLE)
  NDB Cluster, 2356
passwd option
  ndb_top, 4060
password
  resetting expired, 1138
  root user, 240
password option, 333
  mysql, 394
  mysqladmin, 422
  mysqlbinlog, 556
  mysqlcheck, 432
  mysqldump, 443
  mysqld_multi, 365
  mysqlimport, 467
  mysqlpump, 481
  mysqlshow, 492
  mysqlslap, 502
  mysql_secure_installation, 371
  mysql_upgrade, 382
  ndb_top, 4060
password policy, 1265
password validation, 1265
PASSWORD(), 2060
passwords
  administrator guidelines, 1069
  expiration, 1146
  for the InnoDB memcached interface, 3087
  for users, 1078
  forgotten, 5631
  logging, 1070
  lost, 5631
  resetting, 1146, 5631
  security, 1068, 1077
  setting, 1135, 2601
  user guidelines, 1068
password_history system variable, 796
password_history table

system table, 946, 1096
password_require_current system variable, 797
password_reuse_interval system variable, 798
PATH environment variable, 132, 138, 238, 318, 578
path name separators
Windows, 324
pattern matching, 294, 1951
peer row
window functions, 2203
PERCENT_RANK(), 2198
performance, 1446
benchmarks, 1652
disk I/O, 1631
estimating, 1586
Performance Schema, 3053, 4478, 6613
data_locks table, 3022
data_lock_waits table, 3022
event filtering, 4491
memory use, 4487
Thread pool tables, 4613
Performance Schema functions, 2209
Performance Schema queries
optimization, 1519
performance-schema-consumer-events-stages-current option
mysqld, 4665
performance-schema-consumer-events-stages-history option
mysqld, 4665
performance-schema-consumer-events-stages-history-long option
mysqld, 4665
performance-schema-consumer-events-statements-current option
mysqld, 4665
performance-schema-consumer-events-statements-history option
mysqld, 4665
performance-schema-consumer-events-statements-history-long option
mysqld, 4665
performance-schema-consumer-events-transactions-current option
mysqld, 4665
performance-schema-consumer-events-transactions-history option
mysqld, 4665
performance-schema-consumer-events-transactions-history-long option
mysqld, 4665
performance-schema-consumer-events-waits-current option
mysqld, 4666
performance-schema-consumer-events-waits-history option
mysqld, 4666
performance-schema-consumer-events-waits-history-long option
mysqld, 4666
performance-schema-consumer-global-instrumentation option
mysqld, 4666
performance-schema-consumer-statements-digest option
mysqld, 4666
performance-schema-consumer-thread-instrumentation option
mysqld, 4666
performance-schema-consumer-xxx option

mysqld, 4665
performance-schema-instrument option
mysqld, 4665
performance_schema
accounts table, 4574
clone_progress table, 4620
clone_status table, 4619
cond_instances table, 4536
data_locks table, 4598, 4692
data_lock_waits table, 4601
events_errors_summary_by_account_by_error table, 4647
events_errors_summary_by_host_by_error table, 4647
events_errors_summary_by_thread_by_error table, 4647
events_errors_summary_by_user_by_error table, 4647
events_errors_summary_global_by_error table, 4647
events_stages_current table, 4551
events_stages_history table, 4552
events_stages_history_long table, 4553
events_stages_summary_by_account_by_event_name table, 4626
events_stages_summary_by_host_by_event_name table, 4626
events_stages_summary_by_thread_by_event_name table, 4626
events_stages_summary_by_user_by_event_name table, 4626
events_stages_summary_global_by_event_name table, 4626
events_statements_current table, 4557
events_statements_histogram_by_digest table, 4631
events_statements_histogram_global table, 4631
events_statements_history table, 4561
events_statements_history_long table, 4561
events_statements_summary_by_account_by_event_name table, 4627
events_statements_summary_by_digest table, 4627
events_statements_summary_by_host_by_event_name table, 4627
events_statements_summary_by_program table, 4627
events_statements_summary_by_thread_by_event_name table, 4627
events_statements_summary_by_user_by_event_name table, 4627
events_statements_summary_global_by_event_name table, 4627
events_transactions_current table, 4568
events_transactions_history table, 4571
events_transactions_history_long table, 4571
events_transactions_summary_by_account_by_event table, 4634
events_transactions_summary_by_host_by_event_name table, 4634
events_transactions_summary_by_thread_by_event_name table, 4634
events_transactions_summary_by_user_by_event_name table, 4634
events_transactions_summary_global_by_event_name table, 4634
events_waits_current table, 4543
events_waits_history table, 4547
events_waits_history_long table, 4547
events_waits_summary_by_account_by_event_name table, 4624
events_waits_summary_by_host_by_event_name table, 4624
events_waits_summary_by_instance table, 4624
events_waits_summary_by_thread_by_event_name table, 4624
events_waits_summary_by_user_by_event_name table, 4624
events_waits_summary_global_by_event_name table, 4624
file_instances table, 4537
file_summary_by_event_name table, 4636

file_summary_by_instance table, 4637
hosts table, 4574
host_cache table, 4650
keyring_keys table, 1278, 1338, 1365, 4654
log_status table, 4654
memory_summary_by_account_by_event_name table, 4643
memory_summary_by_host_by_event_name table, 4643
memory_summary_by_thread_by_event_name table, 4643
memory_summary_by_user_by_event_name table, 4643
memory_summary_global_by_event_name table, 4643
metadata_locks table, 4603
mutex_instances table, 4537
objects_summary_global_by_type table, 4636
parent events, 4692
performance_timers table, 4655
prepared_statements_instances table, 4627
replication_applier_configuration, 4589
replication_applier_status, 4589
replication_applier_status_by_coordinator, 4590
replication_applier_status_by_worker, 4592
replication_connection_configuration, 4584
replication_connection_status, 4586
rwlock_instances table, 4539
session_account_connect_attrs table, 4578
session_connect_attrs table, 4579
setup_actors table, 4528
setup_consumers table, 4529
setup_instruments table, 4530
setup_objects table, 4533
setup_threads table, 4534
socket_instances table, 4540
socket_summary_by_event_name table, 4642
socket_summary_by_instance table, 4642
table_handles table, 4606
table_io_waits_summary_by_index_usage table, 4639
table_io_waits_summary_by_table table, 4638
table_lock_waits_summary_by_table table, 4640
thread table, 4656
tp_thread_group_state table, 4614
tp_thread_group_stats table, 4616
tp_thread_state table, 4618
users table, 4575
user_defined_functions table, 4661
user_variables_by_thread table, 4579
performance_schema database, 4478
restrictions, 4693
TRUNCATE TABLE, 4523, 4693
PERFORMANCE_SCHEMA storage engine, 4478
performance_schema system variable, 4667
Performance_schema_accounts_lost status variable, 4684
performance_schema_accounts_size system variable, 4667
Performance_schema_cond_classes_lost status variable, 4684
Performance_schema_cond_instances_lost status variable, 4684
performance_schema_digests_size system variable, 4668

Performance_schema_digest_lost status variable, 4685
performance_schema_error_size system variable, 4668
performance_schema_events_stages_history_long_size system variable, 4669
performance_schema_events_stages_history_size system variable, 4669
performance_schema_events_statements_history_long_size system variable, 4669
performance_schema_events_statements_history_size system variable, 4670
performance_schema_events_transactions_history_long_size system variable, 4670
performance_schema_events_transactions_history_size system variable, 4670
performance_schema_events_waits_history_long_size system variable, 4670
performance_schema_events_waits_history_size system variable, 4671
Performance_schema_file_classes_lost status variable, 4685
Performance_schema_file_handles_lost status variable, 4685
Performance_schema_file_instances_lost status variable, 4685
Performance_schema_hosts_lost status variable, 4685
performance_schema_hosts_size system variable, 4671
Performance_schema_index_stat_lost status variable, 4685
Performance_schema_locker_lost status variable, 4685
performance_schema_max_cond_classes system variable, 4671
performance_schema_max_cond_instances system variable, 4672
performance_schema_max_digest_length system variable, 4672
performance_schema_max_digest_sample_age system variable, 4672
performance_schema_max_file_classes system variable, 4673
performance_schema_max_file_handles system variable, 4674
performance_schema_max_file_instances system variable, 4674
performance_schema_max_index_stat system variable, 4674
performance_schema_max_memory_classes system variable, 4675
performance_schema_max_metadata_locks system variable, 4675
performance_schema_max_mutex_classes system variable, 4675
performance_schema_max_mutex_instances system variable, 4676
performance_schema_max_prepared_statements_instances system variable, 4676
performance_schema_max_program_instances system variable, 4677
performance_schema_max_rwlock_classes system variable, 4677
performance_schema_max_rwlock_instances system variable, 4677
performance_schema_max_socket_classes system variable, 4678
performance_schema_max_socket_instances system variable, 4678
performance_schema_max_sql_text_length system variable, 4678
performance_schema_max_stage_classes system variable, 4679
performance_schema_max_statement_classes system variable, 4679
performance_schema_max_statement_stack system variable, 4680
performance_schema_max_table_handles system variable, 4680
performance_schema_max_table_instances system variable, 4681
performance_schema_max_table_lock_stat system variable, 4681
performance_schema_max_thread_classes system variable, 4681
performance_schema_max_thread_instances system variable, 4682
Performance_schema_memory_classes_lost status variable, 4685
Performance_schema_metadata_lock_lost status variable, 4685
Performance_schema_mutex_classes_lost status variable, 4685
Performance_schema_mutex_instances_lost status variable, 4685
Performance_schema_nested_statement_lost status variable, 4685
Performance_schema_prepared_statements_lost status variable, 4685
Performance_schema_program_lost status variable, 4686
Performance_schema_rwlock_classes_lost status variable, 4686
Performance_schema_rwlock_instances_lost status variable, 4686
Performance_schema_session_connect_attrs_longest_seen status variable, 4686

Performance_schema_session_connect_attrs_lost status variable, 4686
performance_schema_session_connect_attrs_size system variable, 4682
performance_schema_setup_actors_size system variable, 4683
performance_schema_setup_objects_size system variable, 4683
Performance_schema_socket_classes_lost status variable, 4686
Performance_schema_socket_instances_lost status variable, 4686
Performance_schema_stage_classes_lost status variable, 4686
Performance_schema_statement_classes_lost status variable, 4687
Performance_schema_table_handles_lost status variable, 4687
Performance_schema_table_instances_lost status variable, 4687
Performance_schema_table_lock_stat_lost status variable, 4687
Performance_schema_thread_classes_lost status variable, 4687
Performance_schema_thread_instances_lost status variable, 4687
Performance_schema_users_lost status variable, 4687
performance_schema_users_size system variable, 4683
performance_timers table
  performance_schema, 4655
PERIOD_ADD(), 1987
PERIOD_DIFF(), 1987
Perl
  installing, 274
  installing on Windows, 275
Perl API, 4940
Perl DBI/DBD
  installation problems, 276
permission checks
  effect on speed, 1522
perror, 317, 577
  help option, 577
  ndb option, 577
  silent option, 577
  verbose option, 577
  version option, 577
PERSIST
  SET statement, 883, 2626
persisted_globals_load system variable, 798, 883
PERSIST_ONLY
  SET statement, 883, 2626
persist_only_admin_x509_subject system variable, 798
PERSIST_RO_VARIABLES_ADMIN privilege, 1090
pessimistic, 6613
phantom, 6613
phantom rows, 2827
phone book collation, German, 1786, 1786
physical, 6613
physical backup, 6613
PI(), 1971
pid-file option
  mysql.server, 363
  mysqld_safe, 359
pid_file system variable, 799
Ping
  thread command, 1655
pipe option, 333

mysql, 394, 433
mysqladmin, 423
mysqldump, 444
mysqlimport, 467
mysqlshow, 492
mysqlslap, 502
mysql_upgrade, 382
PIPES_AS_CONCAT SQL mode, 921
PITR, 6613
PKG_CONFIG_PATH environment variable, 578, 4791
plan stability, 6613
pluggable authentication
PAM, 1205
restrictions, 1151
Windows, 1216
plugin
audit_log, 1312
plugin activation options
FORCE, 997
FORCE_PLUS_PERMANENT, 997
OFF, 997
ON, 997
plugin API, 993, 4945
plugin installing
audit_log, 1313
Clone, 1032
CONNECTION_CONTROL, 1259
CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS, 1259
Data Masking, 1400
ddl_rewriter, 1017
keyring_aws, 1279
keyring_encrypted_file, 1279
keyring_file, 1279
keyring_okv, 1279
keyring_udf, 1295
MySQL Enterprise Firewall plugins, 1387
MySQL Enterprise Thread Pool, 1001
Rewriter query rewrite plugin, 1007
Version Tokens, 1019
plugin option prefix
mysqld, 690
plugin service
get_sysvar_source, 5011
locking_service, 5011, 5012
mysql_keyring, 5011, 5018
mysql_password_policy, 5011
my_plugin_log_service, 5011
my_thd_scheduler, 5011
plugin_registry_service, 5011
security_context, 5011
thd_alloc, 5011
thd_wait, 5011
plugin services, 5010
plugin table

system table, 946
plugin uninstalling
Data Masking, 1400
Rewriter query rewrite plugin, 1007
Version Tokens, 1019
plugin-dir option, 333
mysql, 395
mysqladmin, 423
mysqlbinlog, 557
mysqlcheck, 433
mysqldump, 444
mysqld_safe, 359
mysqlimport, 467
mysqlpump, 481
mysqlshow, 492
mysqlslap, 502
mysql_upgrade, 382
plugin-load option
mysqld, 689
plugin-load-add option
mysqld, 690
plugindir option
mysql_config, 574
plugins
activating, 994
adding, 4945
audit, 4948
authentication, 4948
clone, 1030
conditions for writing, 4953
daemon, 4947
full-text parser, 4946
INFORMATION_SCHEMA, 4948
installing, 994, 2622
keyring, 4950, 5007
protocol trace, 4949
protocol trace plugin, 5002
query rewrite, 4949
security, 1193
semisynchronous replication, 4948
server, 993
storage engine, 4946
test protocol trace plugin, 5002
uninstalling, 994, 2624
PLUGINS
INFORMATION_SCHEMA table, 4401
plugin_dir system variable, 800
plugin_registry_service service, 5011
POINT data type, 1855
Point(), 2085
point-in-time recovery, 1435, 6613
InnoDB, 3069
using NDB Cluster replication, 4236
polltimeout option

ndb_import, 4002
POLYGON data type, 1855
Polygon(), 2085
port option, 334
mysql, 395
mysqladmin, 423
mysqlbinlog, 557
mysqlcheck, 433
mysqld, 691
mysqldump, 444
mysqld_safe, 359
mysqlimport, 467
mysqlpump, 481
mysqlshow, 492
mysqlslap, 503
mysql_config, 574
mysql_secure_installation, 371
mysql_upgrade, 382
ndb_setup.py, 4051
ndb_top, 4060
port system variable, 800
port-open-timeout option
mysqld, 691
portability, 1448
types, 1898
porting
to other systems, 5033
PortNumber, 3770
PortNumberStats, 3773
ports, 215, 215, 236, 340, 557, 578, 1061, 1066, 1163, 1192, 4540, 5619
POSITION(), 1941
post-filtering
Performance Schema, 4492
post-query option
mysqlslap, 503
post-system option
mysqlslap, 503
PostgreSQL compatibility, 60
postinstall
multiple servers, 1055
postinstallation
setup and testing, 228
POW(), 1971
POWER(), 1971
pre-filtering
Performance Schema, 4492
pre-query option
mysqlslap, 503
pre-system option
mysqlslap, 503
precedence
command options, 319
operator, 1918
precision

arithmetic, 2228
fractional seconds, 1818, 1822
numeric, 1818
precision math, 2228
preload_buffer_size system variable, 800
Prepare
thread command, 1655
PREPARE, 2515, 2519
XA transactions, 2486
prepared backup, 6614
prepared statement C API
data structures, 4874
functions, 4879, 4881
type codes, 4878
prepared statements, 2515, 2519, 2519, 2519, 4874
repreparation, 1620
prepared_statements_instances table
performance_schema, 4627
preparing
thread state, 1659
preparing for alter table
thread state, 1659
PreSendChecksum, 3935, 3943
preserve-trailing-spaces option
ndb_restore, 4033
pretty option
ibd2sdi, 509
primary key, 6614
constraint, 63
deleting, 2261
PRIMARY KEY, 2261, 2317
primary keys
and partitioning keys, 4324
primary passwords, 1143
PrimaryMGMNode, 3871
print command
mysql, 402
print option
ndb_restore, 4033
print-data option
ndb_restore, 4033
print-defaults option, 327
myisamchk, 520
mysql, 395
mysqladmin, 423
mysqlbinlog, 557
mysqlcheck, 433
mysqld, 691
mysqldump, 446
mysqlimport, 467
mysqlpump, 481
mysqlshow, 492
mysqlslap, 503
mysql_secure_installation, 371

mysql_upgrade, 382
NDB client programs, 4070
print-full-config option (ndb_mgmd), 3965
print-log option
ndb_restore, 4034
print-meta option
ndb_restore, 4034
print-sql-log option
ndb_restore, 4034
print-table-metadata option
mysqlbinlog, 557
print_identified_with_as_hex system variable, 801
privilege
changes, 1134
privilege checks
effect on speed, 1522
privilege information
location, 1095
privilege restrictions
GRANT statement, 2595
partial revokes, 1127
privilege system, 1077
privileges
access, 1077
adding, 1112
ALL, 1082
ALL PRIVILEGES, 1082
ALTER, 1082
ALTER ROUTINE, 1082
and replication, 3379
APPLICATION_PASSWORD_ADMIN, 1088
AUDIT_ADMIN, 1088
BACKUP_ADMIN, 1088
BINLOG_ADMIN, 1088
BINLOG_ENCRYPTION_ADMIN, 1088
checking, 1114
CLONE_ADMIN, 1089
CONNECTION_ADMIN, 1089
CREATE, 1083
CREATE ROLE, 1083
CREATE ROUTINE, 1083
CREATE TABLESPACE, 1083
CREATE TEMPORARY TABLES, 1083
CREATE USER, 1083
CREATE VIEW, 1083
default, 240
DEFINER, 2653, 4354
DELETE, 1083
deleting, 2585
display, 2652
DROP, 1083
DROP ROLE, 1083
dropping, 2585
ENCRYPTION_KEY_ADMIN, 1089

EVENT, 1084
EXECUTE, 1084
FILE, 1084
FIREWALL_ADMIN, 1089
FIREWALL_USER, 1089
GRANT OPTION, 1084
granting, 2585
GROUP_REPLICATION_ADMIN, 1089
INDEX, 1084
INNODB_REDO_LOG_ARCHIVE, 1089
INSERT, 1084
INVOKER, 2653, 4354
LOCK TABLES, 1084
NDB_STORED_USER, 1090
PERSIST_RO_VARIABLES_ADMIN, 1090
PROCESS, 1084
PROXY, 1084
REFERENCES, 1085
RELOAD, 1085
REPLICATION CLIENT, 1085
REPLICATION SLAVE, 1085
REPLICATION_APPLIER, 1090
REPLICATION_SLAVE_ADMIN, 1090
RESOURCE_GROUP_ADMIN, 1091
RESOURCE_GROUP_USER, 1091
revoking, 1115, 2598
ROLE_ADMIN, 1091
SELECT, 1085
SERVICE_CONNECTION_ADMIN, 1091
SESSION_VARIABLES_ADMIN, 1091
SET_USER_ID, 1091
SHOW DATABASES, 1085
SHOW VIEW, 1086
SHUTDOWN, 1086
SQL SECURITY, 4354
static versus dynamic, 1093
stored objects, 4354
SUPER, 1086
SYSTEM_USER, 1091, 1123
SYSTEM_VARIABLES_ADMIN, 1092
TABLE_ENCRYPTION_ADMIN, 1092
TEMPORARY tables, 1083, 2336, 2593
TRIGGER, 1087
UPDATE, 1088
USAGE, 1088
VERSION_TOKEN_ADMIN, 1092
XA_RECOVER_ADMIN, 1092
problems
access denied errors, 5618
common errors, 5617
compiling MySQL server, 225
DATE columns, 5640
date values, 1835
installing on Solaris, 187

installing Perl, 276
linking, 4790
lost connection errors, 5622
reporting, 2, 52
starting the server, 236
table locking, 1624
time zone, 5638
procedures
stored, 4333
process, 6614
process management (NDB Cluster), 3946
PROCESS privilege, 1084
processes
display, 2662
ndbinfo table, 4151
processing
arguments, 5026
Processing events
thread state, 1666
Processing events from schema table
thread state, 1666
Processlist
thread command, 1655
PROCESSLIST, 2662
INFORMATION_SCHEMA table, 4402
possible inconsistency with INFORMATION_SCHEMA tables, 3027
processlist view
sys schema, 4722
procs_priv table
system table, 946, 1096
PROFILING
INFORMATION_SCHEMA table, 4404
profiling system variable, 801
profiling_history_size system variable, 801
program options (NDB Cluster), 4065
program variables
setting, 328
program-development utilities, 317
programs
administrative, 316
client, 315, 4788
stored, 2520, 4331
utility, 316
progress-frequency option
ndb_restore, 4034
promote-attributes option
ndb_move_data, 4012
ndb_restore, 4035
prompt command
mysql, 402
PROMPT command (NDB Cluster), 4077
prompt option
mysql, 395
prompts

meanings, 280
pronunciation
MySQL, 6
protocol option, 334
mysql, 395
mysqladmin, 423
mysqlbinlog, 557
mysqlcheck, 433
mysqldump, 444
mysqlimport, 467
mysqlpump, 481
mysqlshow, 493
mysqlslap, 503
mysql_secure_installation, 371
mysql_upgrade, 382
protocol trace plugins, 4949
protocol_compression_algorithms system variable, 801
protocol_version system variable, 802
proxies_priv
grant table, 1156
proxies_priv table
system table, 240, 946, 1096
proximity search, 2002
PROXY privilege, 1084
proxy user mapping
LDAP authentication, 1233
proxy users, 1153
conflict with anonymous users, 1158
default proxy user, 1157
LDAP authentication, 1230
PAM authentication, 1212
PROXY privilege, 1156
server user mapping, 1159
system variables, 1160
Windows authentication, 1220
proxy_user system variable, 802
pseudo-record, 6614
pseudo_slave_mode system variable, 803
pseudo_thread_id system variable, 803
ps_check_lost_instrumentation view
sys schema, 4724
PS_CURRENT_THREAD_ID() function, 2211
ps_is_account_enabled() function
sys schema, 4771
ps_is_consumer_enabled() function
sys schema, 4772
ps_is_instrument_default_enabled() function
sys schema, 4772
ps_is_instrument_default_timed() function
sys schema, 4772
ps_is_thread_instrumented() function
sys schema, 4773
ps_setup_disable_background_threads() procedure
sys schema, 4750

ps_setup_disable_consumer() procedure
sys schema, 4751
ps_setup_disable_instrument() procedure
sys schema, 4751
ps_setup_disable_thread() procedure
sys schema, 4751
ps_setup_enable_background_threads() procedure
sys schema, 4752
ps_setup_enable_consumer() procedure
sys schema, 4752
ps_setup_enable_instrument() procedure
sys schema, 4753
ps_setup_enable_thread() procedure
sys schema, 4753
ps_setup_reload_saved() procedure
sys schema, 4754
ps_setup_reset_to_default() procedure
sys schema, 4754
ps_setup_save() procedure
sys schema, 4754
ps_setup_show_disabled() procedure
sys schema, 4755
ps_setup_show_disabled_consumers() procedure
sys schema, 4756
ps_setup_show_disabled_instruments() procedure
sys schema, 4756
ps_setup_show_enabled() procedure
sys schema, 4757
ps_setup_show_enabled_consumers() procedure
sys schema, 4757
ps_setup_show_enabled_instruments() procedure
sys schema, 4758
ps_statement_avg_latency_histogram() procedure
sys schema, 4758
ps_thread_account() function
sys schema, 4773
PS_THREAD_ID() function, 2211
ps_thread_id() function
sys schema, 4774
ps_thread_stack() function
sys schema, 4774
ps_thread_trx_info() function
sys schema, 4775
ps_trace_statement_digest() procedure
sys schema, 4759
ps_trace_thread() procedure
sys schema, 4760
ps_truncate_all_tables() procedure
sys schema, 4762
Pthreads, 6614
purge, 2857, 6614
PURGE BINARY LOGS, 2490
purge buffering, 6614
purge configuration, 2857

purge lag, 6614
PURGE MASTER LOGS, 2490
purge scheduling, 2857
purge thread, 6615
Purging old relay logs
  thread state, 1659
pushdown joins (NDB),
Python, 4785
  third-party driver, 4940

Q

QUARTER(), 1987
queries
  entering, 278
  estimating performance, 1586
  examples, 303
  speed of, 1448
Query
  thread command, 1655
query, 6615
query cache
  and ndbinfo database tables, 4120
query end
  thread state, 1659
query execution plan, 6615
query expansion, 2008
query option
  mysqlslap, 503
  ndb_config, 3976, 3976
query option (ndb_index_stat), 4007
query rewrite plugins, 4949
  ddl_rewriter, 1016
  Rewriter, 1006
query-all option
  ndb_config, 3976
query_alloc_block_size system variable, 804
query_cache_limit system variable, 804
query_cache_min_res_unit system variable, 805
query_cache_size system variable, 805
query_cache_type system variable, 806
query_cache_wlock_invalidate system variable, 806
query_prealloc_size system variable, 806
questions, 418
Queueing master event to the relay log
  thread state, 1663
QUICK
  DELETE modifier, 2384
quick option
  myisamchk, 524
  mysql, 395
  mysqlcheck, 433
  mysqldump, 458
quiesce, 6615

Quit
  thread command, 1655
quit command
  mysql, 402
QUIT command (NDB Cluster), 4076
quotation marks
  in strings, 1671
QUOTE(), 1672, 1941, 4850, 4851
quote-names option
  mysqldump, 453
quote_identifier() function
  sys schema, 4776
quoting, 1672
  account names, 1104
  column alias, 1680, 5643
  host names in account names, 1104
  schema objects, 2588
  user names in account names, 1104
quoting binary data, 1672
quoting of identifiers, 1679

R

R-tree, 6615
RADIANS(), 1972
RAID, 6615
RAND(), 1972
random dive, 6615
RANDOM_BYTES(), 2060
rand_seed1 system variable, 807
rand_seed2 system variable, 807
range join type
  optimizer, 1575
range partitioning, 4263, 4270
range partitions
  adding and dropping, 4290
  managing, 4290
range_alloc_block_size system variable, 807
range_optimizer_max_mem_size system variable, 808
RANK(), 2199
raw backup, 6615
raw option
  mysql, 395
  mysqlbinlog, 557
raw partitions, 2767
rbr_exec_mode system variable, 808
RC
  MySQL releases, 78
READ COMMITTED, 6616
  implementation in NDB Cluster, 3679
  transaction isolation level, 2816
read conflict detection and resolution
  in NDB Cluster Replication, 4253
read from standard in

innochecksum, 513
read phenomena, 6616
READ UNCOMMITTED, 6616
transaction isolation level, 2818
read view, 6616
read-ahead, 6616
linear, 2846
random, 2846
read-from-remote-master option
mysqlbinlog, 557
read-from-remote-server option
mysqlbinlog, 558
read-only option
myisamchk, 523
read-only transaction, 6616
Reading event from the relay log
thread state, 1664
Reading master dump table data
thread state, 1665
READ_BACKUP, 2355
READ_BACKUP (NDB_TABLE)
NDB Cluster, 2356
read_buffer_size myisamchk variable, 521
read_buffer_size system variable, 809
read_only system variable, 809
read_rnd_buffer_size system variable, 810
REAL data type, 1822
RealtimeScheduler, 3837
REAL_AS_FLOAT SQL mode, 921
rebuild-indexes option
ndb_restore, 4035
Rebuilding the index on master dump table
thread state, 1665
ReceiveBufferMemory, 3936
Receiving from client
thread state, 1659
reconfiguring, 225
reconnect option
mysql, 396
Reconnecting after a failed binlog dump request
thread state, 1663
Reconnecting after a failed master event read
thread state, 1663
reconnection
automatic, 4657, 4937
record lock, 6617
record-level locks
InnoDB, 2827
RECOVER
XA transactions, 2486
recover option
myisamchk, 524
recovery
from crash, 1439

incremental, 1435
InnoDB, 3068
point in time, 1435
RecoveryWork, 3801
redo, 6617
redo log, 2805, 2805, 6617
redo log archiving, 2805, 6617
RedoBuffer, 3827
RedoOverCommitCounter
data nodes, 3856
RedoOverCommitLimit
data nodes, 3857
reducing
data size, 1540
redundant row format, 2894, 6617
ref join type
optimizer, 1574
references, 2262
REFERENCES privilege, 1085
referential integrity, 2718, 6617
REFERENTIAL_CONSTRAINTS
INFORMATION_SCHEMA table, 4405
Refresh
thread command, 1655
ref_or_null, 1488
ref_or_null join type
optimizer, 1574
REGEXP, 1952
REGEXP_INSTR(), 1953
REGEXP_LIKE(), 1953
REGEXP_REPLACE(), 1955
regexp_stack_limit system variable, 811
REGEXP_SUBSTR(), 1955
regexp_time_limit system variable, 811
Register Slave
thread command, 1655
Registering slave on master
thread state, 1663
regular account
account categories, 1123
regular expression syntax, 1951
regular session
session categories, 1125
rehash command
mysql, 402
rejects option
ndb_import, 4002
relational, 6617
relational databases
defined, 5
relative option
mysqladmin, 423
relay logs (replication), 3317
relay-log-info-repository option

mysqld, 3240
relay-log-purge option
mysqld, 3224
relay-log-recovery option
mysqld, 3224
relay-log-space-limit option
mysqld, 3225
relay_log system variable, 3243
relay_log_basename system variable, 3244
relay_log_index system variable, 3244
relay_log_info_file system variable, 3245
relay_log_info_repository system variable, 3245
relay_log_purge system variable, 3246
relay_log_recovery system variable, 3246
relay_log_space_limit system variable, 3247
release numbers, 78
RELEASE SAVEPOINT, 2475
releases
DMR, 78
GA, 78
naming scheme, 78
RC, 78
RELEASE_ALL_LOCKS(), 2064
RELEASE_LOCK(), 2064
relevance, 6618
reload option (ndb_mgmd), 3965
RELOAD privilege, 1085
remote administration (NDB Cluster)
and security issues, 4172
remove option
mysqld, 692
MySQLInstallerConsole, 126
remove option (ndbd), 3953
remove option (ndbmtd), 3953
remove option (ndb_mgmd), 3965
removed features in MySQL 8.0, 9
Removing duplicates
thread state, 1659
removing tmp table
thread state, 1659
rename
thread state, 1659
rename database, 2378
rename result table
thread state, 1659
RENAME TABLE, 2377
RENAME USER statement, 2598
renaming user accounts, 2598
Reopen tables
thread state, 1659
repair
tables, 426
Repair by sorting
thread state, 1659

Repair done
  thread state, 1660
repair option
  mysqlcheck, 433
repair options
  myisamchk, 523
REPAIR TABLE
  and partitioning, 4306
  and replication, 3380
REPAIR TABLE statement, 2618
  and replication, 2619
  options, 2619
  output, 2620
  partitioning support, 2619
  storage engine support, 2619
Repair with keycache
  thread state, 1660
repairing
  tables, 1440
REPEAT, 2527
  labels, 2520
REPEAT(), 1941
REPEATABLE READ, 6618
  transaction isolation level, 2816
repertoire, 6618
  character set, 1735, 1772
  string, 1735
REPLACE, 2418
replace option
  mysqldump, 447
  mysqlimport, 467
  mysqlpump, 481
REPLACE(), 1942
replicas (NDB Cluster), 3657
replicate-do-db option
  mysqld, 3225
replicate-do-table option
  mysqld, 3228
replicate-ignore-db option
  mysqld, 3227
replicate-ignore-table option
  mysqld, 3229
replicate-rewrite-db option
  mysqld, 3230
replicate-same-server-id option
  mysqld, 3231
replicate-wild-do-table option
  mysqld, 3231
replicate-wild-ignore-table option
  mysqld, 3232
replication, 3159, 6618
  and AUTO_INCREMENT, 3364
  and character sets, 3365
  and CHECKSUM TABLE statement, 3365

and CREATE ... IF NOT EXISTS, 3366
and CREATE TABLE ... SELECT, 3366
and DATA DIRECTORY, 3371
and DROP ... IF EXISTS, 3372
and errors on slave, 3381
and floating-point values, 3372
and FLUSH, 3372
and fractional seconds, 3374
and functions, 3372
and INDEX DIRECTORY, 3371
and invoked features, 3374
and LAST_INSERT_ID(), 3364
and LIMIT, 3377
and LOAD DATA, 3377
and max_allowed_packet, 3377
and MEMORY tables, 3378
and mysql (system) schema, 3379
and partial updates, 3381
and partitioned tables, 3379
and partitioning, 3382
and privileges, 3379
and query optimizer, 3379
and REPAIR TABLE statement, 2618, 3380
and reserved words, 3380
and scheduled events, 3375, 3375
and SQL mode, 3382
and stored routines, 3375
and temporary tables, 3382
and time zones, 3383
and TIMESTAMP, 3364
and transactions, 3383, 3385
and triggers, 3375, 3387
and TRUNCATE TABLE, 3388
and user name length, 3388
and variables, 3388
and views, 3390
attribute demotion, 3369
attribute promotion, 3369
BLACKHOLE, 3365
circular, 4215
crashes, 3380
delayed, 3361
group, 3397
in NDB Cluster, 4211
(see also NDB Cluster replication)
nondeterministic functions, 1498
relay logs, 3317
resource groups, 1650
row-based vs statement-based, 3305
safe and unsafe statements, 3310
semisynchronous, 3356
shutdown and restart, 3380, 3382
statements incompatible with STATEMENT format, 3306
status logs, 3317

timeouts, 3383
unexpected halt, 3346
with differing tables on master and slave, 3367
replication channel
commands, 3314
compatibility, 3315
naming conventions, 3317
startup options, 3315
replication channel based filters, 3329
replication channels, 3314
REPLICATION CLIENT privilege, 1085
replication filtering options
and case sensitivity, 3324
replication formats
compared, 3305
replication implementation, 3304
replication limitations, 3364
replication log tables, 3317
replication master
thread states, 1662
replication masters
statements, 2490
replication mode, 3198
concepts, 3198
disabling online, 3202
enabling online, 3200
verifying anonymous transactions, 3203
replication options, 3364
replication server
statements, 2511
replication slave
thread states, 1663, 1664, 1665
REPLICATION SLAVE privilege, 1085
replication slaves
statements, 2493
replication technologies, 3399
replication, asynchronous (see NDB Cluster replication)
REPLICATION_APPLIER privilege, 1090
replication_applier_configuration
performance_schema, 4589
replication_applier_status
performance_schema, 4589
replication_applier_status_by_coordinator
performance_schema, 4590
replication_applier_status_by_worker
performance_schema, 4592
replication_connection_configuration
performance_schema, 4584
replication_connection_status
performance_schema, 4586
REPLICATION_SLAVE_ADMIN privilege, 1090
REPORT command (NDB Cluster), 4075
reporting
bugs, 2, 52

errors, 52
problems, 2
report_host system variable, 3247
report_password system variable, 3248
report_port system variable, 3248
report_user system variable, 3249
REPRODUCIBLE_BUILD option
CMake, 216
Requesting binlog dump
thread state, 1663
REQUIRE option
ALTER USER, 2568
CREATE USER statement, 2579
require_secure_transport system variable, 812
reserved user accounts, 1115
reserved words, 1691
and replication, 3380
ReservedConcurrentIndexOperations, 3790
ReservedConcurrentOperations, 3791
ReservedConcurrentScans, 3791
ReservedConcurrentTransactions, 3791
ReservedFiredTriggers, 3792
ReservedLocalScans, 3792
ReservedTransactionBufferMemory, 3792
RESET MASTER, 2491
and mysql.gtid_executed table, 3175
RESET MASTER statement, 2698
RESET PERSIST statement, 865, 883, 2699
reset set metadata
suppression, 4936
RESET SLAVE, 2504
RESET SLAVE ALL, 2504
RESET SLAVE statement, 2698
Reset stmt
thread command, 1655
reset-slave.pl (see NDB Cluster replication)
resetconnection command
mysql, 402
resetting expired password, 1138
RESIGNAL, 2541
resource group names
case sensitivity, 1684
resource groups, 1646
names, 1679
resource limits
user accounts, 776, 1161, 2570, 2581
resources
ndbinfo table, 4153
RESOURCE_GROUPS
INFORMATION_SCHEMA table, 4406
resource_groups table
data dictionary table, 944, 2604
RESOURCE_GROUP_ADMIN privilege, 1091
RESOURCE_GROUP_USER privilege, 1091

RESTART command (NDB Cluster), 4074
RESTART statement, 2700
restarting
  the server, 239
RestartOnErrorInsert, 3811
RestartSubscriberConnectTimeout, 3825
restart_info
  ndbinfo table, 4154
restore, 6618
restore-data option
  ndb_restore, 4036
restore-epoch option
  ndb_restore, 4036
restore-meta option
  ndb_restore, 4036
restore-privilege-tables option
  ndb_restore, 4037
restoring backups
  in NDB Cluster, 4021
restoring from backup
  in NDB Cluster replication, 4230
restrictions
  character sets, 1795
  events, 4363
  InnoDB, 2754
  performance_schema database, 4693
  pluggable authentication, 1151
  resource groups, 1650
  server-side cursors, 2530
  signals, 2559
  stored routines, 4363
  subqueries, 2454
  triggers, 4363
  views, 4367
  XA transactions, 2489
result-file option
  mysqlbinlog, 558
  mysqldump, 453
  mysqlpump, 481
resultset_metadata system variable, 812
resume option
  ndb_import, 4002
retries option
  ndb_desc, 3987
retrieving
  data from tables, 286
RETURN, 2527
return (\r), 1671, 2142, 2406
return values
  UDFs, 5028
reular expressions
  in JSON schemas, 2152
REVERSE(), 1942
REVOKE statement, 1112, 2598

revoking
  privileges, 2598
revoking roles, 2598
rewrite-database option
  ndb_restore, 4037
rewrite-db option
  mysqlbinlog, 558
Rewriter query rewrite plugin, 1006
  installing, 1007
  uninstalling, 1007
Rewriter UDFs
  load_rewrite_rules(), 1015
rewriter_enabled system variable, 1015
Rewriter_number_loaded_rules status variable, 1015
Rewriter_number_reloads status variable, 1015
Rewriter_number_rewritten_queries status variable, 1016
Rewriter_reload_error status variable, 1016
rewriter_verbose system variable, 1015
RIGHT JOIN, 1476, 2430
RIGHT OUTER JOIN, 2430
RIGHT(), 1942
RLIKE, 1952
role names, 1107
roles, 1116
  assigning, 2603
  creating, 2573
  default, 2600
  dropping, 2584
  granting, 2585
  revoking, 2598
  stored programs, 1120
  views, 1120
ROLES_GRAPHML(), 2074
ROLE_ADMIN privilege, 1091
role_edges table
  system table, 946, 1096
ROLLBACK, 2470
  XA transactions, 2486
rollback, 6618
rollback segment, 2776, 2779, 6619
ROLLBACK TO SAVEPOINT, 2475
Rolling back
  thread state, 1660
rolling restart (NDB Cluster), 4085
ROLLUP, 2181
root password, 240
root user, 1066
  password resetting, 5631
ROUND(), 1973
rounding, 2228
rounding errors, 1820
routines
  stored, 4331, 4333
ROUTINES

INFORMATION_SCHEMA table, 4406
routines option
mysqldump, 456
mysqlpump, 481
routines table
data dictionary table, 944
ROW, 2441
row, 6619
row constructors, 2441
optimizations, 1501
row format, 6619
row lock, 6619
row size
maximum, 1551
row subqueries, 2441
row-based replication, 6619
advantages, 3307
disadvantages, 3308
row-level locking, 1622, 6619
rowbatch option
ndb_import, 4002
rowbytes option
ndb_import, 4003
rowid option
ndb_select_all, 4046
rows
counting, 296
deleting, 5644
matching problems, 5644
selecting, 287
sorting, 289
rows option
ndb_config, 3976
ROW_COUNT(), 2074
ROW_FORMAT
COMPACT, 2895
COMPRESSED, 2875, 2896
DYNAMIC, 2896
REDUNDANT, 2894
ROW_NUMBER(), 2199
RPAD(), 1942
rpl_read_size system variable, 3249
Rpl_semi_sync_master_clients status variable, 907
rpl_semi_sync_master_enabled system variable, 3218
Rpl_semi_sync_master_net_avg_wait_time status variable, 907
Rpl_semi_sync_master_net_waits status variable, 908
Rpl_semi_sync_master_net_wait_time status variable, 907
Rpl_semi_sync_master_no_times status variable, 908
Rpl_semi_sync_master_no_tx status variable, 908
Rpl_semi_sync_master_status status variable, 908
Rpl_semi_sync_master_timefunc_failures status variable, 908
rpl_semi_sync_master_timeout system variable, 3219
rpl_semi_sync_master_trace_level system variable, 3219
Rpl_semi_sync_master_tx_avg_wait_time status variable, 908

Rpl_semi_sync_master_tx_waits status variable, 908
Rpl_semi_sync_master_tx_wait_time status variable, 908
rpl_semi_sync_master_wait_for_slave_count system variable, 3220
rpl_semi_sync_master_wait_no_slave system variable, 3220
rpl_semi_sync_master_wait_point system variable, 3221
Rpl_semi_sync_master_wait_pos_backtraverse status variable, 908
Rpl_semi_sync_master_wait_sessions status variable, 909
Rpl_semi_sync_master_yes_tx status variable, 909
rpl_semi_sync_slave_enabled system variable, 3250
Rpl_semi_sync_slave_status status variable, 909
rpl_semi_sync_slave_trace_level system variable, 3250
rpl_stop_slave_timeout system variable, 3250
RPM file, 155, 160
RPM Package Manager, 160
RTRIM(), 1942
Ruby API, 4941
running
  ANSI mode, 57
  batch mode, 301
  multiple servers, 1055
  queries, 278
running CMake after prior invocation, 196, 225
rw-lock, 6619
rwlock_instances table
  performance_schema, 4539

S

safe statement (replication)
  defined, 3310
safe-recover option
  myisamchk, 524
safe-updates mode, 413
safe-updates option
  mysql, 396, 413
safe-user-create option
  mysqld, 692
SafeNet KeySecure Applicance
  keyring_okv keyring plugin, 1286
Sakila, 9
same value wins (conflict resolution), 4245
sampling
  statement, 4629
sandbox mode
  for expired-password accounts, 1146
SASL, 3087
  authentication, 1221
SAVEPOINT, 2475
savepoint, 6620
Saving state
  thread state, 1660
scalability, 6620
Scalable Coherent Interface (NDB Cluster) (OBSOLETE), 3946
scalar

JSON, 1874
scale
arithmetic, 2228
numeric, 1818
scale out, 6620
scale up, 6620
SchedulerExecutionTimer, 3837
SchedulerResponsiveness, 3838
SchedulerSpinTimer, 3838
schema, 6620
altering, 2244
creating, 2276
deleting, 2371
SCHEMA Events (NDB Cluster), 4094
SCHEMA(), 2075
SCHEMATA
INFORMATION_SCHEMA table, 4409
schemata table
data dictionary table, 944
schema_auto_increment_columns view
sys schema, 4724
schema_definition_cache system variable, 813
schema_index_statistics view
sys schema, 4725
schema_object_overview view
sys schema, 4726
SCHEMA_PRIVILEGES
INFORMATION_SCHEMA table, 4410
schema_redundant_indexes view
sys schema, 4726
schema_tables_with_full_table_scans view
sys schema, 4732
schema_table_lock_waits view
sys schema, 4727
schema_table_statistics view
sys schema, 4729
schema_table_statistics_with_buffer view
sys schema, 4730
schema_unused_indexes view
sys schema, 4733
SCI (NDB Cluster) (OBSOLETE), 3946
script files, 301
scripts, 355, 363
SQL, 385
SDI, 505, 2389, 6621, 6621
search index, 6621
searching
and case sensitivity, 5639
full-text, 1997
MySQL Web pages, 52
two keys, 308
Searching rows for update
thread state, 1660
SECOND(), 1987

secondary index, 6621
  InnoDB, 2757
secondary passwords, 1143
secondary_engine_cost_threshold system variable, 813
Secondary_engine_execution_count status variable, 909
secure connections, 1169
  command options, 335
secure-auth option, 334
  mysql, 396
  mysqladmin, 423
  mysqlbinlog, 558
  mysqlcheck, 433
  mysqld, 692
  mysqldump, 444
  mysqlimport, 468
  mysqlpump, 481
  mysqlshow, 493
  mysqlslap, 503
secure_auth system variable, 814
secure_file_priv system variable, 814
securing an NDB Cluster, 4175
security
  against attackers, 1071
  and malicious SQL statements, 4174
  and NDB utilities, 4176
  components, 1193
  for the InnoDB memcached interface, 3087
  plugins, 1193
security system, 1077
security_context plugin service, 5011
SEC_TO_TIME(), 1987
segment, 6621
SELECT
  INTO, 2428
  LIMIT, 2421
  optimizing, 1568, 2701
SELECT INTO TABLE, 61
SELECT privilege, 1085
select-limit option
  mysql, 396
selecting
  databases, 283
selectivity, 6621
SELinux
  LDAP authentication, 1226
semi-consistent read, 6621
semijoins, 1503
semisynchronous replication, 3356
  administrative interface, 3358
  configuration, 3359
  installation, 3359
  monitoring, 3361
semisynchronous replication plugins, 4948
SendBufferMemory, 3934, 3943

Sending binlog event to slave
  thread state, 1663
Sending to client
  thread state, 1660
SendSignalId, 3935, 3943
sensible JSON values, 1879
SEQUENCE, 309
sequence emulation, 2073
sequences, 309
SERIAL, 1818, 1820
SERIAL DEFAULT VALUE, 1890
SERIALIZABLE, 6622
  transaction isolation level, 2818
serialized dictionary information (see SDI)
serialized dictionary information (SDI), 6622
server, 6622
  connecting, 277, 340
  debugging, 5033
  disconnecting, 277
  logs, 948
  restart, 239
  shutdown, 238
  signal handling, 940
  starting, 230
  starting and stopping, 242
  starting problems, 236
server administration, 415
server components, 989
  installing, 990
  uninstalling, 990
server configuration, 582
server connections
  command options, 332
server plugins, 993
server variables, 2684 (see system variables)
server-id option
  mysqlbinlog, 558
  mysqld, 3204
server-id-bits option
  mysqlbinlog, 559
  mysqld, 3887
server-log-file option
  ndb_setup.py, 4051
server-name option
  ndb_setup.py, 4051
server-public-key-path option, 335
  mysql, 396
  mysqladmin, 423
  mysqlbinlog, 559
  mysqlcheck, 433
  mysqldump, 444
  mysqlimport, 468
  mysqlpump, 482
  mysqlshow, 493

mysqlslap, 503
mysql_upgrade, 382
server-side cursors
restrictions, 2530
ServerPort, 3777
servers
multiple, 1055
servers table
system table, 948
server_cost
system table, 1609
server_cost table
system table, 947
server_id system variable, 815
server_id_bits system variable, 3912
server_locks
ndbinfo table, 4156
server_operations
ndbinfo table, 4157
server_transactions
ndbinfo table, 4159
server_uuid system variable
mysqld, 3205
service-startup-timeout option
mysql.server, 363
services
for plugins, 5010
SERVICE_CONNECTION_ADMIN privilege, 1091
service_get_read_locks() UDF
locking service, 5017
service_get_write_locks() UDF
locking service, 5017
service_release_locks() UDF
locking service, 5018
SESSION
SET statement, 2626
session categories, 1125
session state
change tracking, 936
session state information, 816, 817, 818, 4859, 4864
session temporary tablespace, 6622
session track gtids, 816
session variables
and replication, 3388
session view
sys schema, 4733
session_account_connect_attrs table
performance_schema, 4578
session_connect_attrs table
performance_schema, 4579
session_ssl_status view
sys schema, 4733
session_track_gtids, 816
session_track_schema system variable, 816

session_track_state_change system variable, 817
session_track_system_variables system variable, 818
session_track_transaction_info system variable, 818
SESSION_USER(), 2075
SESSION_VARIABLES_ADMIN privilege, 1091
SET
  CHARACTER SET, 1750
  NAMES, 1750
  size, 1897
SET CHARACTER SET statement, 2631
SET CHARSET statement, 2631
SET data type, 1828, 1851
SET DEFAULT ROLE statement, 2600
SET GLOBAL sql_slave_skip_counter, 2505
SET GLOBAL statement, 865
SET NAMES, 1757
SET NAMES statement, 2631
Set option
  thread command, 1655
SET PASSWORD statement, 2601
SET PERSIST statement, 865
SET PERSIST_ONLY statement, 865
SET RESOURCE GROUP statement, 2607
SET ROLE statement, 2603
SET SESSION statement, 865
SET sql_log_bin, 2493
SET statement
  assignment operator, 1929
  CHARACTER SET, 2631
  CHARSET, 2631
  NAMES, 2631
  variable assignment, 883, 2626
SET TRANSACTION, 2482
set-auto-increment[option
  myisamchk, 525
set-charset option
  mysqlbinlog, 559
  mysqldump, 449
  mysqlpump, 482
set-collation option
  myisamchk, 525
set-gtid-purged option
  mysqldump, 450
  mysqlpump, 482
setting
  passwords, 1135
setting passwords, 2601
setting program variables, 328
setup
  postinstallation, 228
  thread state, 1660
setup.bat
  NDB Cluster (Windows), 4048
setup_actors table

performance_schema, 4528
setup_consumers table
performance_schema, 4529
setup_instruments table
performance_schema, 4530
setup_objects table
performance_schema, 4533
setup_threads table
performance_schema, 4534
SET_USER_ID privilege, 1091
SET_VAR optimizer hint, 1603
SHA(), 2060
SHA1(), 2060
SHA2(), 2061
sha256_password authentication plugin, 1200
sha256_password_auto_generate_rsa_keys system variable, 819
sha256_password_private_key_path system variable, 820
sha256_password_proxy_users system variable, 820, 1160
sha256_password_public_key_path system variable, 820
sha2_cache_cleaner audit plugin, 1199
shared lock, 2810, 6622
shared memory transporter (see NDB Cluster)
shared tablespace, 6622
shared-memory-base-name option, 334
mysql, 396
mysqladmin, 424
mysqlbinlog, 559
mysqlcheck, 434
mysqldump, 459
mysqlimport, 468
mysqlshow, 493
mysqlslap, 504
mysql_upgrade, 383
SharedGlobalMemory, 3851
shared_memory system variable, 821
shared_memory_base_name system variable, 821
sharp checkpoint, 6622
shell syntax, 4
ShmKey, 3944
ShmSize, 3944
ShmSpinTime, 3944
short-form option
mysqlbinlog, 559
SHOW
in NDB Cluster management client, 3739
SHOW BINARY LOGS statement, 2631, 2632
SHOW BINLOG EVENTS statement, 2631, 2633
SHOW CHARACTER SET statement, 2631, 2634
SHOW COLLATION statement, 2631, 2635
SHOW COLUMNS statement, 2631, 2636
SHOW command (NDB Cluster), 4074
SHOW CREATE DATABASE statement, 2631, 2638
SHOW CREATE EVENT statement, 2631
SHOW CREATE FUNCTION statement, 2631, 2639

SHOW CREATE PROCEDURE statement, 2631, 2639
SHOW CREATE SCHEMA statement, 2631, 2638
SHOW CREATE TABLE statement, 2631, 2640
SHOW CREATE TRIGGER statement, 2631, 2641
SHOW CREATE USER statement, 2641
SHOW CREATE VIEW statement, 2631, 2642
SHOW DATABASES privilege, 1085
SHOW DATABASES statement, 2631, 2643
SHOW ENGINE
  and NDB Cluster, 4115
SHOW ENGINE INNODB STATUS statement, 2643
SHOW ENGINE NDB STATUS, 4115
SHOW ENGINE NDBCLUSTER STATUS, 4115
SHOW ENGINE statement, 2631, 2643
SHOW ENGINES
  and NDB Cluster, 4115
SHOW ENGINES statement, 2631, 2647
SHOW ERRORS statement, 2631, 2649
SHOW EVENTS statement, 2631, 2650
SHOW extensions, 4475
SHOW FIELDS statement, 2631, 2636
SHOW FUNCTION CODE statement, 2631, 2652
SHOW FUNCTION STATUS statement, 2631, 2652
SHOW GRANTS statement, 2631, 2652
SHOW INDEX statement, 2631, 2655
SHOW KEYS statement, 2631, 2655
SHOW MASTER LOGS statement, 2631, 2632
SHOW MASTER STATUS statement, 2631, 2658
SHOW OPEN TABLES statement, 2631, 2658
show option
  my_print_defaults, 576
SHOW PLUGINS statement, 2631, 2659
SHOW PRIVILEGES statement, 2631, 2660
SHOW PROCEDURE CODE statement, 2631, 2661
SHOW PROCEDURE STATUS statement, 2631, 2662
SHOW PROCESSLIST statement, 2631, 2662
SHOW PROFILE statement, 2631, 2664
SHOW PROFILES statement, 2631, 2664, 2667
SHOW RELAYLOG EVENTS statement, 2631, 2667
SHOW SCHEDULER STATUS, 4345
SHOW SCHEMAS statement, 2643
SHOW SLAVE HOSTS statement, 2631, 2668
SHOW SLAVE STATUS statement, 2631, 2669
SHOW STATUS
  and NDB Cluster, 4116
SHOW STATUS statement, 2631, 2677
SHOW STORAGE ENGINES statement, 2647
SHOW TABLE STATUS statement, 2631, 2679
SHOW TABLES statement, 2631, 2682
SHOW TRIGGERS statement, 2631, 2682
SHOW VARIABLES
  and NDB Cluster, 4115
SHOW VARIABLES statement, 2631, 2684
SHOW VIEW privilege, 1086

SHOW WARNINGS statement, 2631, 2686
SHOW with WHERE, 4370, 4475
show-create-skip-secondary-engine option
  mysqldump, 453
show-slave-auth-info option
  mysqld, 3213
show-table-type option
  mysqlshow, 493
show-temp-status option
  ndb_show_tables, 4053
show-warnings option
  mysql, 396
  mysqladmin, 424
showing
  database information, 487
show_compatibility_56 system variable, 822
show_create_table_skip_secondary_engine system variable, 822
show_create_table_verbosity system variable, 823
show_old_temporals system variable, 823
shutdown, 6623
  server, 940
Shutdown
  thread command, 1655
SHUTDOWN command (NDB Cluster), 4076
SHUTDOWN privilege, 1086
SHUTDOWN statement, 2701
shutdown_timeout variable, 426
shutting down
  the server, 238
Shutting down
  thread state, 1666
sigint-ignore option
  mysql, 397
SIGN(), 1974
SIGNAL, 2546
signals
  restrictions, 2559
  server response, 940
SigNum, 3945
silent column changes, 2348
silent option
  myisamchk, 520
  myisampack, 535
  mysql, 397
  mysqladmin, 424
  mysqlcheck, 434
  mysqld_multi, 365
  mysqlimport, 468
  mysqlslap, 504
  ndb_perror, 4014
  perror, 577
SIN(), 1974
single quote (\'), 1671
single user mode (NDB Cluster), 4075, 4114

and ndb_restore, 4021
single-transaction option
mysqldump, 459
mysqlpump, 482
single-user option
ndb_waiter, 4063
SINGLEUSER Events (NDB Cluster), 4095
size of tables, 1549
sizes
display, 1818
SKIP LOCKED, 2426
--skip option prefix, 327
skip-broken-objects option
ndb_restore, 4038
skip-column-names option
mysql, 397
skip-comments option
mysqldump, 448
skip-concurrent-insert option
mysqld, 692
skip-data option
ibd2sdi, 507
skip-database option
mysqlcheck, 434
skip-definer option
mysqlpump, 483
skip-dump-rows option
mysqlpump, 483
skip-grant-tables option
mysqld, 692
skip-gtids option
mysqlbinlog, 559
skip-host-cache option
mysqld, 693
skip-innodb option
mysqld, 693, 2932
skip-kill-mysqld option
mysqld_safe, 359
skip-line-numbers option
mysql, 397
skip-name-resolve option
mysqld, 694
skip-ndbcluster option
mysqld, 3887
skip-networking option
mysqld, 694
skip-nodegroup option (ndb_error_reporter), 3990
skip-opt option
mysqldump, 458
skip-show-database option
mysqld, 695
skip-slave-start option
mysqld, 3235
skip-ssl option, 336

skip-stack-trace option
  mysqld, 696
skip-symbolic-links option
  mysqld, 695
skip-sys-schema option
  mysql_upgrade, 383
skip-syslog option
  mysqld_safe, 359
skip-table-check option
  ndb_restore, 4038
skip-unknown-objects option
  ndb_restore, 4038
skip_external_locking system variable, 823
skip_name_resolve system variable, 824
skip_networking system variable, 824
skip_show_database system variable, 824
Slave has read all relay log; waiting for more updates
  thread state, 1664
slave server, 6623
slave-checkpoint-group option
  mysqld, 3233
slave-checkpoint-period option
  mysqld, 3234
slave-load-tmpdir option
  mysqld, 3235
slave-max-allowed-packet (mysqld), 3236
slave-net-timeout option
  mysqld, 3236
slave-parallel-type (mysqld), 3237
slave-parallel-workers option
  mysqld, 3234
slave-pending-jobs-size-max option
  mysqld, 3235
slave-rows-search-algorithms (mysqld), 3238
slave-skip-errors option
  mysqld, 3238
slave-sql-verify-checksum option
  mysqld, 3239
slave_allow_batching, 4227
slave_checkpoint_group system variable, 3251
slave_checkpoint_period system variable, 3252
slave_compressed_protocol system variable, 3252
slave_exec_mode system variable, 3253
slave_load_tmpdir system variable, 3254
slave_master_info table
  system table, 947
slave_max_allowed_packet system variable, 3254
slave_net_timeout system variable, 3255
slave_parallel_type system variable, 3255
slave_parallel_workers system variable, 3256
slave_pending_jobs_size_max system variable, 3257
slave_preserve_commit_order, 3257
slave_relay_log_info table
  system table, 947

slave_rows_search_algorithms system variable, 3258
slave_skip_errors system variable, 3259
slave_sql_verify_checksum system variable, 3259
slave_transaction_retries system variable, 3259
slave_type_conversions system variable, 3260
slave_worker_info table
  system table, 947
Sleep
  thread command, 1655
sleep option
  mysqladmin, 424
SLEEP(), 2224
sleep-time option
  ndb_top, 4060
slow queries, 418
slow query log, 984, 6623
slow shutdown, 6623
slow-start-timeout option
  mysqld, 696
slow_launch_time system variable, 825
slow_log table
  system table, 947
slow_query_log system variable, 825
slow_query_log_file system variable, 826
SMALLINT data type, 1820
snapshot, 6623
SNAPSHOTEND (START BACKUP command), 4080
SNAPSHOTSTART (START BACKUP command), 4080
socket option, 334
  mysql, 397
  mysqladmin, 424
  mysqlbinlog, 560
  mysqlcheck, 434
  mysqld, 696
  mysqldump, 444
  mysqld_safe, 359
  mysqlimport, 468
  mysqlpump, 483
  mysqlshow, 493
  mysqlslap, 504
  mysql_config, 574
  mysql_secure_installation, 371
  mysql_upgrade, 383
  ndb_top, 4060
socket system variable, 826
socket_instances table
  performance_schema, 4540
socket_summary_by_event_name table
  performance_schema, 4642
socket_summary_by_instance table
  performance_schema, 4642
Solaris
  installation, 187
Solaris installation problems, 187

Solaris troubleshooting, 227
Solaris x86_64 issues, 1559
SOME, 2440
sort buffer, 6623
sort option
  ndb_top, 4061
sort-index option
  myisamchk, 525
sort-records option
  myisamchk, 525
sort-recover option
  myisamchk, 525
sorting
  data, 289
  grant tables, 1109, 1111
  table rows, 289
Sorting for group
  thread state, 1660
Sorting for order
  thread state, 1660
Sorting index
  thread state, 1660
Sorting result
  thread state, 1660
sort_buffer_size myisamchk variable, 521
sort_buffer_size system variable, 826
sort_key_blocks myisamchk variable, 521
SOUNDEX(), 1942
SOUNDS LIKE, 1943
source (mysql client command), 303, 410
source command
  mysql, 403
source distribution
  installing, 190
space ID, 6623
SPACE(), 1943
sparse file, 6624
spatial data type
  SRID attribute, 1855
spatial data types, 1853
  storage requirements, 1897
spatial extensions in MySQL, 1853
spatial functions, 2076
SPATIAL index
  InnoDB predicate locks, 2815
SPATIAL indexes
  optimization, 1524
spatial queries
  optimization, 1527
spatial values
  syntactically well-formed, 1865
speed
  increasing with replication, 3159
  inserting, 1521

of queries, 1448
spin, 6624
spin lock polling, 2856
sporadic-binlog-dump-fail option
mysqld, 3269
SQL, 6624
defined, 5
SQL mode, 914
ALLOW_INVALID_DATES, 916
and partitioning, 3382, 4319
and replication, 3382
ANSI, 915, 921
ANSI_QUOTES, 916
ERROR_FOR_DIVISION_BY_ZERO, 916
HIGH_NOT_PRECEDENCE, 917
IGNORE_SPACE, 917
NO_AUTO_VALUE_ON_ZERO, 917
NO_BACKSLASH_ESCAPES, 918
NO_DIR_IN_CREATE, 918
NO_ENGINE_SUBSTITUTION, 918
NO_UNSIGNED_SUBTRACTION, 918
NO_ZERO_DATE, 919
NO_ZERO_IN_DATE, 919
ONLY_FULL_GROUP_BY, 920, 2187
PAD_CHAR_TO_FULL_LENGTH, 920
PIPES_AS_CONCAT, 921
REAL_AS_FLOAT, 921
strict, 916
STRICT_ALL_TABLES, 921
STRICT_TRANS_TABLES, 916, 921
TIME_TRUNCATE_FRACTIONAL, 921
TRADITIONAL, 916, 922
SQL node (NDB Cluster)
defined, 3654
SQL nodes (NDB Cluster), 4083
SQL scripts, 385
SQL SECURITY
effect on privileges, 4354
SQL statements
replication masters, 2490
replication server, 2511
replication slaves, 2493
SQL statements relating to NDB Cluster, 4114
SQL-92
extensions to, 56
sql-mode option
mysqld, 696
mysqlslap, 504
sql_auto_is_null system variable, 827
SQL_BIG_RESULT
SELECT modifier, 2427
sql_big_selects system variable, 828
SQL_BUFFER_RESULT
SELECT modifier, 2427

sql_buffer_result system variable, 828
SQL_CACHE
  SELECT modifier, 2428
SQL_CALC_FOUND_ROWS, 1496
  SELECT modifier, 2428
sql_log_bin system variable, 3290
sql_log_off system variable, 828
sql_mode system variable, 829
sql_notes system variable, 832
SQL_NO_CACHE
  SELECT modifier, 2428
sql_quote_show_create system variable, 832
sql_require_primary_key system variable, 832
sql_safe_updates system variable, 413, 833
sql_select_limit system variable, 413, 834
sql_slave_skip_counter, 2505
sql_slave_skip_counter system variable, 3261
SQL_SMALL_RESULT
  SELECT modifier, 2427
sql_warnings system variable, 834
SQRT(), 1974
square brackets, 1818
SRID attribute
  spatial data type, 1855
SRID values
  handling by spatial functions, 2079
SSD, 2875, 6624
SSH, 1071, 1192
SSL, 1169
  command options, 335
  establishing connections, 1170
  X.509 Basics, 1169
SSL library
  configuring, 199
ssl option, 336
SSL options
  mysql, 397
  mysqladmin, 424
  mysqlbinlog, 560
  mysqlcheck, 434
  mysqld, 694
  mysqldump, 445
  mysqlimport, 468
  mysqlpump, 483
  mysqlshow, 493
  mysqlslap, 504
  mysql_secure_installation, 371
  mysql_upgrade, 383
SSL related options
  ALTER USER, 2568
  CREATE USER statement, 2579
ssl-ca option, 336
ssl-capath option, 336
ssl-cert option, 336

ssl-cipher option, 337
ssl-crl option, 337
ssl-crlpath option, 337
ssl-fips-mode option, 337, 371, 383, 397, 424, 434, 445, 468, 483, 493, 504, 560
ssl-key option, 338
ssl-mode option, 338
ssl_ca system variable, 834
ssl_capath system variable, 835
ssl_cert system variable, 835
ssl_cipher system variable, 835
ssl_crl system variable, 836
ssl_crlpath system variable, 836
ssl_fips_mode system variable, 836
ssl_key system variable, 837
staging-tries option
  ndb_move_data, 4012
standalone option
  mysqld, 694
Standard Monitor, 3061, 3063, 3066
Standard SQL
  differences from, 60, 2597
  extensions to, 56, 58
standards compatibility, 56
START
  XA transactions, 2486
START BACKUP
  NOWAIT, 4079
  SNAPSHOTEND, 4080
  SNAPSHOTSTART, 4080
  syntax, 4079
  WAIT COMPLETED, 4080
  WAIT STARTED, 4079
START command (NDB Cluster), 4074
START GROUP_REPLICATION, 2511
START SLAVE, 2506
START TRANSACTION, 2470
start-datetime option
  mysqlbinlog, 560
start-page option
  innochecksum, 510
start-position option
  mysqlbinlog, 560
StartConnectBackoffMaxTime, 3869
StartFailRetryDelay, 3857
StartFailureTimeout, 3814
starting
  comments, 62
  mysqld, 1073
  the server, 230
  the server automatically, 242
Starting many servers, 1055
StartNoNodeGroupTimeout, 3814
StartPartialTimeout, 3813
StartPartitionedTimeout, 3813

startup, 6624
STARTUP Events (NDB Cluster), 4091
startup options
  default, 321
  replication channel, 3315
startup parameters, 582
  mysql, 385
  mysqladmin, 418
StartupStatusReportFrequency, 3831
state-dir option
  ndb_import, 4003
statefile option
  comp_err, 368
statement sampling, 4629
statement termination
  Control+C, 385, 397, 2465
statement-based replication, 6624
  advantages, 3306
  disadvantages, 3306
  unsafe statements, 3306
statements
  compound, 2520
  CREATE USER, 1112
  DROP USER, 1112
  GRANT, 1112
  replication masters, 2490
  replication server, 2511
  replication slaves, 2493
  REVOKE, 1112
statements_with_errors_or_warnings view
  sys schema, 4735
statements_with_full_table_scans view
  sys schema, 4736
statements_with_runtimes_in_95th_percentile view
  sys schema, 4737
statements_with_sorting view
  sys schema, 4738
statements_with_temp_tables view
  sys schema, 4739
statement_analysis view
  sys schema, 4733
STATEMENT_DIGEST(), 2061
STATEMENT_DIGEST_TEXT(), 2061
statement_performance_analyzer() procedure
  sys schema, 4762
static privileges, 1093
Statistics
  thread command, 1655
statistics, 6624
  thread state, 1660
STATISTICS
  INFORMATION_SCHEMA table, 4411
STATISTICS Events (NDB Cluster), 4093
stats option

myisam_ftdump, 516
ndb_import, 4003
stats_method myisamchk variable, 521
status
tables, 2679
status command
mysql, 403
results, 417
STATUS command (NDB Cluster), 4075
status logs (replication), 3317
status option
MySQLInstallerConsole, 126
mysqlshow, 494
status variable
Acl_cache_items_count, 893
Audit_log_current_size, 1382
Audit_log_events, 1382
Audit_log_events_filtered, 1382
Audit_log_events_lost, 1382
Audit_log_events_written, 1382
Audit_log_event_max_drop_size, 1382
Audit_log_total_size, 1382
Audit_log_write_waits, 1382
Caching_sha2_password_rsa_public_key, 893
Compression_algorithm, 894
Compression_level, 895
Connection_control_delay_generated, 1264
Current_tls_ca, 896
Current_tls_capath, 896
Current_tls_cert, 896
Current_tls_cipher, 897
Current_tls_ciphersuites, 897
Current_tls_crl, 897
Current_tls_crlpath, 897
Current_tls_key, 897
Current_tls_version, 897
dragnet.Status, 898
Firewall_access_denied, 1398
Firewall_access_granted, 1398
Firewall_access_suspicious, 1398
Firewall_cached_entries, 1398
Locked_connects, 905
Max_execution_time_exceeded, 905
Max_execution_time_set, 905
Max_execution_time_set_failed, 905
Performance_schema_accounts_lost, 4684
Performance_schema_cond_classes_lost, 4684
Performance_schema_cond_instances_lost, 4684
Performance_schema_digest_lost, 4685
Performance_schema_file_classes_lost, 4685
Performance_schema_file_handles_lost, 4685
Performance_schema_file_instances_lost, 4685
Performance_schema_hosts_lost, 4685
Performance_schema_index_stat_lost, 4685

Performance_schema_locker_lost, 4685
Performance_schema_memory_classes_lost, 4685
Performance_schema_metadata_lock_lost, 4685
Performance_schema_mutex_classes_lost, 4685
Performance_schema_mutex_instances_lost, 4685
Performance_schema_nested_statement_lost, 4685
Performance_schema_prepared_statements_lost, 4685
Performance_schema_program_lost, 4686
Performance_schema_rwlock_classes_lost, 4686
Performance_schema_rwlock_instances_lost, 4686
Performance_schema_session_connect_attrs_longest_seen, 4686
Performance_schema_session_connect_attrs_lost, 4686
Performance_schema_socket_classes_lost, 4686
Performance_schema_socket_instances_lost, 4686
Performance_schema_stage_classes_lost, 4686
Performance_schema_statement_classes_lost, 4687
Performance_schema_table_handles_lost, 4687
Performance_schema_table_instances_lost, 4687
Performance_schema_table_lock_stat_lost, 4687
Performance_schema_thread_classes_lost, 4687
Performance_schema_thread_instances_lost, 4687
Performance_schema_users_lost, 4687
Rewriter_number_loaded_rules, 1015
Rewriter_number_reloads, 1015
Rewriter_number_rewritten_queries, 1016
Rewriter_reload_error, 1016
Rpl_semi_sync_master_clients, 907
Rpl_semi_sync_master_net_avg_wait_time, 907
Rpl_semi_sync_master_net_waits, 908
Rpl_semi_sync_master_net_wait_time, 907
Rpl_semi_sync_master_no_times, 908
Rpl_semi_sync_master_no_tx, 908
Rpl_semi_sync_master_status, 908
Rpl_semi_sync_master_timefunc_failures, 908
Rpl_semi_sync_master_tx_avg_wait_time, 908
Rpl_semi_sync_master_tx_waits, 908
Rpl_semi_sync_master_tx_wait_time, 908
Rpl_semi_sync_master_wait_pos_backtraverse, 908
Rpl_semi_sync_master_wait_sessions, 909
Rpl_semi_sync_master_yes_tx, 909
Rpl_semi_sync_slave_status, 909
Secondary_engine_execution_count, 909
validate_password.dictionary_file_last_parsed, 1272
validate_password.dictionary_file_words_count, 1272
validate_password_dictionary_file_last_parsed, 1276
validate_password_dictionary_file_words_count, 1276
status variables, 892, 2677
NDB Cluster, 3917
NDB Cluster replication conflict detection, 4247
status_variable_registration component service, 5011
STD(), 2180
STDDEV(), 2180
STDDEV_POP(), 2180
STDDEV_SAMP(), 2180

stemming, 6625
STOP command (NDB Cluster), 4074
STOP GROUP_REPLICATION, 2511
STOP SLAVE, 2510
stop-datetime option
  mysqlbinlog, 561
stop-never option
  mysqlbinlog, 561
stop-never-slave-server-id option
  mysqlbinlog, 561
stop-position option
  mysqlbinlog, 561
StopOnError, 3811
stopping
  the server, 242
stopword, 6625
stopword list
  user-defined, 2011
stopwords, 2009
storage engine, 6625
  ARCHIVE, 3139
  InnoDB, 2718
  PERFORMANCE_SCHEMA, 4478
storage engine plugins, 4946
storage engines
  and application feature requirements, 3674
  applications supported, 3673
  availability, 3672
  choosing, 3119
  differences between NDB and InnoDB, 3672
  usage scenarios, 3674
storage nodes - see data nodes, ndbd (see data nodes, ndbd)
storage nodes - see data nodes, ndbd, ndbmtd (see data nodes, ndbd, ndbmtd)
storage requirements
  data types, 1893
  date data types, 1895
  InnoDB tables, 1894
  NDB Cluster, 1894
  numeric data types, 1894
  spatial data types, 1897
  string data types, 1895
  time data types, 1895
storage space
  minimizing, 1540
stored functions, 4333
stored generated column, 6625
stored object, 6625
stored object privileges, 4354
stored objects, 4331
stored procedures, 4333
stored program, 6625
stored programs, 2520, 4331
  reparsing, 1620
  roles, 1120

stored routine, 6625
stored routines, 4331, 4333
  and replication, 3375
  LAST_INSERT_ID(), 4335
  metadata, 4335
  restrictions, 4363
stored_program_definition_cache system variable, 838
STRAIGHT_JOIN, 1477, 1569, 1583, 2430, 2703
  join type, 1505
  SELECT modifier, 1505, 2427
STRCMP(), 1951
strict mode, 6625
strict SQL mode, 916
strict-check option
  ibd2sdi, 508
  innochecksum, 511
STRICT_ALL_TABLES SQL mode, 921
STRICT_TRANS_TABLES SQL mode, 916, 921
string
  JSON, 1874
string collating, 1799
string comparison functions, 1948
string comparison operators, 1948
string comparisons
  case sensitivity, 1948
string concatenation, 1669, 1935
string data types
  storage requirements, 1895
string functions, 1932
string literal introducer, 1670, 1744
string literals, 1669
string operators, 1932
string types, 1843
StringMemory, 3784
strings
  defined, 1669
  escape sequences, 1669
  nondelimited, 1674
  repertoire, 1735
striping
  defined, 1631
STR_TO_DATE(), 1987
ST_Area(), 2096
ST_AsBinary(), 2086
ST_AsGeoJSON(), 2112
ST_AsText(), 2086
ST_Buffer(), 2100
ST_Buffer_Strategy(), 2101
ST_Centroid(), 2097
ST_Contains(), 2105
ST_ConvexHull(), 2102
ST_Crosses(), 2105
ST_Difference(), 2102
ST_Dimension(), 2088

ST_Disjoint(), 2105
ST_Distance(), 2106
ST_Distance_Sphere(), 2114
ST_EndPoint(), 2093
ST_Envelope(), 2088
ST_Equals(), 2107
ST_ExteriorRing(), 2098
ST_GeoHash(), 2111
ST_GeomCollFromText(), 2081
ST_GeomCollFromWKB(), 2083
ST_GeometryCollectionFromText(), 2081
ST_GeometryCollectionFromWKB(), 2083
ST_GeometryFromText(), 2081
ST_GeometryFromWKB(), 2083
ST_GeometryN(), 2099
ST_GeometryType(), 2088
ST_GEOMETRY_COLUMNS
  INFORMATION_SCHEMA table, 4413
ST_GeomFromGeoJSON(), 2113
ST_GeomFromText(), 2081
ST_GeomFromWKB(), 2083
ST_InteriorRingN(), 2098
ST_Intersection(), 2102
ST_Intersects(), 2107
ST_IsClosed(), 2093
ST_IsEmpty(), 2088
ST_IsSimple(), 2089
ST_IsValid(), 2115
ST_LatFromGeoHash(), 2111
ST_Latitude(), 2091
ST_Length(), 2094
ST_LineFromText(), 2081
ST_LineFromWKB(), 2083
ST_LineStringFromText(), 2081
ST_LineStringFromWKB(), 2083
ST_LongFromGeoHash(), 2111
ST_Longitude(), 2091
ST_MakeEnvelope(), 2116
ST_MLineFromText(), 2081
ST_MLineFromWKB(), 2083
ST_MPointFromText(), 2081
ST_MPointFromWKB(), 2083
ST_MPolyFromText(), 2081
ST_MPolyFromWKB(), 2083
ST_MultiLineStringFromText(), 2081
ST_MultiLineStringFromWKB(), 2083
ST_MultiPointFromText(), 2081
ST_MultiPointFromWKB(), 2083
ST_MultiPolygonFromText(), 2081
ST_MultiPolygonFromWKB(), 2083
ST_NumGeometries(), 2099
ST_NumInteriorRing(), 2098
ST_NumInteriorRings(), 2098
ST_NumPoints(), 2095

ST_Overlaps(), 2107
ST_PointFromGeoHash(), 2112
ST_PointFromText(), 2081
ST_PointFromWKB(), 2083
ST_PointN(), 2095
ST_PolyFromText(), 2082
ST_PolyFromWKB(), 2083
ST_PolygonFromText(), 2082
ST_PolygonFromWKB(), 2083
ST_Simplify(), 2117
ST_SPATIAL_REFERENCE_SYSTEMS
  INFORMATION_SCHEMA table, 4414
st_spatial_reference_systems table
  data dictionary table, 944
ST_SRID(), 2089
ST_StartPoint(), 2096
ST_SwapXY(), 2087
ST_SymDifference(), 2102
ST_Touches(), 2107
ST_Transform(), 2103
ST_Union(), 2104
ST_UNITS_OF_MEASURE
  INFORMATION_SCHEMA table, 4415
ST_Validate(), 2117
ST_Within(), 2107
ST_X(), 2092
ST_Y(), 2092
SUBDATE(), 1989
sublist, 6626
SUBPARTITION BY KEY
  known issues, 4323
subpartitioning, 4283
subpartitions, 4283
  known issues, 4323
subqueries, 2438
  correlated, 2443
  errors, 2450
  in FROM clause (see derived tables)
  optimization, 1502, 1508
  restrictions, 2454
  rewriting as joins, 2454
  with ALL, 2441
  with ANY, IN, SOME, 2440
  with EXISTS, 2442
  with NOT EXISTS, 2442
  with row constructors, 2441
subquery (see subqueries)
subquery materialization, 1506
subselects, 2438
SUBSTR(), 1943
SUBSTRING(), 1943
SUBSTRING_INDEX(), 1944
SUBTIME(), 1989
subtraction (-), 1965

SUDO_USER environment variable, 4577
suffix option
  mysql_ssl_rsa_setup, 375
SUM(), 2180
SUM(DISTINCT), 2180
SUPER privilege, 1086
super-large-pages option
  mysqld, 695
superuser, 240
super_read_only system variable, 839
support
  for operating systems, 78
suppression
  default values, 64
supremum record, 6626
surrogate key, 6626
symbolic links, 1632, 1634
  databases, 1633
  tables, 1633
  Windows, 1634
symbolic-links option
  mysqld, 695
synchronization of master and slave
  in NDB Cluster Replication, 4233
Syncing ndb table schema operation and binlog
  thread state, 1666
sync_binlog system variable, 3291
sync_master_info system variable, 3261
sync_relay_log system variable, 3262
sync_relay_log_info system variable, 3262
syntactically well-formed
  GIS values, 1865
  spatial values, 1865
syntax
  regular expression, 1951
syntax conventions, 3
synthetic key, 6626
sys schema, 4479
  create_synonym_db() procedure, 4747
  diagnostics() procedure, 4748
  execute_prepared_stmt() procedure, 4750
  extract_schema_from_file_name() function, 4767
  extract_table_from_file_name() function, 4767
  format_bytes() function, 4768
  format_path() function, 4768
  format_statement() function, 4769
  format_time() function, 4770
  host_summary view, 4706
  host_summary_by_file_io view, 4707
  host_summary_by_file_io_type view, 4707
  host_summary_by_stages view, 4707
  host_summary_by_statement_latency view, 4708
  host_summary_by_statement_type view, 4709
  innodb_buffer_stats_by_schema view, 4710

innodb_buffer_stats_by_table view, 4710
innodb_lock_waits view, 4711
io_by_thread_by_latency view, 4713
io_global_by_file_by_bytes view, 4714
io_global_by_file_by_latency view, 4715
io_global_by_wait_by_bytes view, 4715
io_global_by_wait_by_latency view, 4716
latest_file_io view, 4717
list_add() function, 4770
list_drop() function, 4771
memory_by_host_by_current_bytes view, 4718
memory_by_thread_by_current_bytes view, 4719
memory_by_user_by_current_bytes view, 4719
memory_global_by_current_bytes view, 4720
memory_global_total view, 4720
metrics view, 4720
object ownership, 4695
processlist view, 4722
ps_check_lost_instrumentation view, 4724
ps_is_account_enabled() function, 4771
ps_is_consumer_enabled() function, 4772
ps_is_instrument_default_enabled() function, 4772
ps_is_instrument_default_timed() function, 4772
ps_is_thread_instrumented() function, 4773
ps_setup_disable_background_threads() procedure, 4750
ps_setup_disable_consumer() procedure, 4751
ps_setup_disable_instrument() procedure, 4751
ps_setup_disable_thread() procedure, 4751
ps_setup_enable_background_threads() procedure, 4752
ps_setup_enable_consumer() procedure, 4752
ps_setup_enable_instrument() procedure, 4753
ps_setup_enable_thread() procedure, 4753
ps_setup_reload_saved() procedure, 4754
ps_setup_reset_to_default() procedure, 4754
ps_setup_save() procedure, 4754
ps_setup_show_disabled() procedure, 4755
ps_setup_show_disabled_consumers() procedure, 4756
ps_setup_show_disabled_instruments() procedure, 4756
ps_setup_show_enabled() procedure, 4757
ps_setup_show_enabled_consumers() procedure, 4757
ps_setup_show_enabled_instruments() procedure, 4758
ps_statement_avg_latency_histogram() procedure, 4758
ps_thread_account() function, 4773
ps_thread_id() function, 4774
ps_thread_stack() function, 4774
ps_thread_trx_info() function, 4775
ps_trace_statement_digest() procedure, 4759
ps_trace_thread() procedure, 4760
ps_truncate_all_tables() procedure, 4762
quote_identifier() function, 4777
schema_auto_increment_columns view, 4724
schema_index_statistics view, 4725
schema_object_overview view, 4726
schema_redundant_indexes view, 4726

schema_tables_with_full_table_scans view, 4732
schema_table_lock_waits view, 4727
schema_table_statistics view, 4729
schema_table_statistics_with_buffer view, 4730
schema_unused_indexes view, 4733
session view, 4733
session_ssl_status view, 4733
statements_with_errors_or_warnings view, 4735
statements_with_full_table_scans view, 4736
statements_with_runtimes_in_95th_percentile view, 4737
statements_with_sorting view, 4738
statements_with_temp_tables view, 4739
statement_analysis view, 4733
statement_performance_analyzer() procedure, 4762
sys_config table, 4702
sys_get_config() function, 4777
table_exists() procedure, 4766
user_summary view, 4740
user_summary_by_file_io view, 4741
user_summary_by_file_io_type view, 4741
user_summary_by_stages view, 4742
user_summary_by_statement_latency view, 4742
user_summary_by_statement_type view, 4743
version view, 4744
version_major() function, 4778
version_minor() function, 4778
version_patch() function, 4779
waits_by_host_by_latency view, 4746
waits_by_user_by_latency view, 4746
waits_global_by_latency view, 4747
wait_classes_global_by_avg_latency view, 4744
wait_classes_global_by_latency view, 4745
x$ views, 4705
x$host_summary view, 4706
x$host_summary_by_file_io view, 4707
x$host_summary_by_file_io_type view, 4707
x$host_summary_by_stages view, 4707
x$host_summary_by_statement_latency view, 4708
x$host_summary_by_statement_type view, 4709
x$innodb_buffer_stats_by_schema view, 4710
x$innodb_buffer_stats_by_table view, 4710
x$innodb_lock_waits view, 4711
x$io_by_thread_by_latency view, 4713
x$io_global_by_file_by_bytes view, 4714
x$io_global_by_file_by_latency view, 4715
x$io_global_by_wait_by_bytes view, 4715
x$io_global_by_wait_by_latency view, 4716
x$latest_file_io view, 4717
x$memory_by_host_by_current_bytes view, 4718
x$memory_by_thread_by_current_bytes view, 4719
x$memory_by_user_by_current_bytes view, 4719
x$memory_global_by_current_bytes view, 4720
x$memory_global_total view, 4720
x$processlist view, 4722

x$schema_flattened_keys view, 4726
x$schema_index_statistics view, 4725
x$schema_tables_with_full_table_scans view, 4732
x$schema_table_lock_waits view, 4727
x$schema_table_statistics view, 4729
x$schema_table_statistics_with_buffer view, 4730
x$session view, 4733
x$statements_with_errors_or_warnings view, 4735
x$statements_with_full_table_scans view, 4736
x$statements_with_runtimes_in_95th_percentile view, 4737
x$statements_with_sorting view, 4738
x$statements_with_temp_tables view, 4739
x$statement_analysis view, 4733
x$user_summary view, 4740
x$user_summary_by_file_io view, 4741
x$user_summary_by_file_io_type view, 4741
x$user_summary_by_stages view, 4742
x$user_summary_by_statement_latency view, 4742
x$user_summary_by_statement_type view, 4743
x$waits_by_host_by_latency view, 4746
x$waits_by_user_by_latency view, 4746
x$waits_global_by_latency view, 4747
x$wait_classes_global_by_avg_latency view, 4744
x$wait_classes_global_by_latency view, 4745
sys-check option (ndb_index_stat), 4009
sys-create option (ndb_index_stat), 4008
sys-create-if-not-exist option (ndb_index_stat), 4008
sys-create-if-not-valid option (ndb_index_stat), 4008
sys-drop option (ndb_index_stat), 4008
sys-skip-events option (ndb_index_stat), 4009
sys-skip-tables option (ndb_index_stat), 4009
SYSCONFDIR option
CMake, 209
SYSDATE(), 1989
sysdate-is-now option
mysqld, 699
syseventlog.facility system variable, 839
syseventlog.include_pid system variable, 840
syseventlog.tag system variable, 840
syslog option
mysql, 398
mysqld_safe, 359
syslog-tag option
mysqld_safe, 359
system
privilege, 1077
security, 1066
system account
account categories, 1123
system command
mysql, 403
System lock
thread state, 1661
system option

ndb_config, 3977
system session
session categories, 1125
system table
optimizer, 1573, 2427
system tables
audit_log_filter table, 948
audit_log_user table, 948
columns_priv table, 946, 1096
column_statistics table, 944, 1612
component table, 946
db table, 240, 946, 1096
default_roles table, 946, 1096
engine_cost, 1609
engine_cost table, 947
firewall_users table, 948
firewall_whitelist table, 948
func table, 946
general_log table, 946
global_grants table, 946, 1094, 1096
gtid_executed table, 947, 3175
help tables, 947
help_category table, 947
help_keyword table, 947
help_relation table, 947
help_topic table, 947
innodb_dynamic_metadata table, 948
innodb_index_stats table, 947, 2859
innodb_table_stats table, 947, 2859
ndb_binlog_index table, 947, 4221
password_history table, 946, 1096
plugin table, 946
procs_priv table, 946, 1096
proxies_priv table, 240, 946, 1096
role_edges table, 946, 1096
servers table, 948
server_cost, 1609
server_cost table, 947
slave_master_info table, 947
slave_relay_log_info table, 947
slave_worker_info table, 947
slow_log table, 947
tables_priv table, 946, 1096
time zone tables, 947
time_zone table, 947
time_zone_leap_second table, 947
time_zone_name table, 947
time_zone_transition table, 947
time_zone_transition_type table, 947
user table, 240, 946, 1096
system tablespace, 6626
system variable
activate_all_roles_on_login, 705
admin_address, 706

admin_port, 707
audit_log_buffer_size, 1372
audit_log_compression, 1372
audit_log_connection_policy, 1373
audit_log_current_session, 1373
audit_log_encryption, 1374
audit_log_exclude_accounts, 1374
audit_log_file, 1375
audit_log_filter_id, 1375
audit_log_flush, 1376
audit_log_format, 1376
audit_log_include_accounts, 1376
audit_log_password_history_keep_days, 1377
audit_log_policy, 1378
audit_log_read_buffer_size, 1379
audit_log_rotate_on_size, 1380
audit_log_statement_policy, 1380
audit_log_strategy, 1381
authentication_ldap_sasl_auth_method_name, 1243
authentication_ldap_sasl_bind_base_dn, 1243
authentication_ldap_sasl_bind_root_dn, 1244
authentication_ldap_sasl_bind_root_pwd, 1244
authentication_ldap_sasl_ca_path, 1245
authentication_ldap_sasl_group_search_attr, 1245
authentication_ldap_sasl_group_search_filter, 1246
authentication_ldap_sasl_init_pool_size, 1246
authentication_ldap_sasl_log_status, 1247
authentication_ldap_sasl_max_pool_size, 1248
authentication_ldap_sasl_server_host, 1248
authentication_ldap_sasl_server_port, 1249
authentication_ldap_sasl_tls, 1249
authentication_ldap_sasl_user_search_attr, 1250
authentication_ldap_simple_auth_method_name, 1250
authentication_ldap_simple_bind_base_dn, 1251
authentication_ldap_simple_bind_root_dn, 1251
authentication_ldap_simple_bind_root_pwd, 1252
authentication_ldap_simple_ca_path, 1252
authentication_ldap_simple_group_search_attr, 1253
authentication_ldap_simple_group_search_filter, 1253
authentication_ldap_simple_init_pool_size, 1254
authentication_ldap_simple_log_status, 1255
authentication_ldap_simple_max_pool_size, 1255
authentication_ldap_simple_server_host, 1256
authentication_ldap_simple_server_port, 1257
authentication_ldap_simple_tls, 1258
authentication_ldap_simple_user_search_attr, 1258
authentication_windows_log_level, 707
authentication_windows_use_principal_name, 708
autocommit, 708
automatic_sp_privileges, 709
auto_generate_certs, 709
auto_increment_increment, 3214
auto_increment_offset, 3217
avoid_temporal_upgrade, 710

back_log, 710
basedir, 711
big_tables, 711
bind_address, 712
binlog_cache_size, 3270
binlog_checksum, 3270
binlog_direct_non_transactional_updates, 3271
binlog_encryption, 3272
binlog_error_action, 3272
binlog_expire_logs_seconds, 3273
binlog_format, 3274
binlog_group_commit_sync_delay, 3275
binlog_group_commit_sync_no_delay_count, 3276
binlog_gtid_simple_recovery, 3293
binlog_max_flush_queue_time, 3277
binlog_order_commits, 3277
binlog_rotate_encryption_master_key_at_startup, 3278
binlog_rows_query_log_events, 3282
binlog_row_event_max_size, 3278
binlog_row_image, 3279
binlog_row_metadata, 3280
binlog_row_value_options, 3281
binlog_stmt_cache_size, 3282
binlog_transaction_dependency_history_size, 3284
binlog_transaction_dependency_tracking, 3283
block_encryption_mode, 713
bulk_insert_buffer_size, 714, 3127
caching_sha2_password_auto_generate_rsa_keys, 715
caching_sha2_password_private_key_path, 715
caching_sha2_password_public_key_path, 716
character_sets_dir, 719
character_set_client, 716
character_set_connection, 717
character_set_database, 717
character_set_filesystem, 718
character_set_results, 718
character_set_server, 718
character_set_system, 719
check_proxy_users, 719, 1160
clone_autotune_concurrency, 1049
clone_buffer_size, 1049
clone_ddl_timeout, 1050
clone_enable_compression, 1050
clone_max_concurrency, 1051
clone_max_data_bandwidth, 1051
clone_max_network_bandwidth, 1052
clone_ssl_ca, 1052
clone_ssl_cert, 1053
clone_ssl_key, 1053
clone_valid_donor_list, 1053
collation_connection, 720
collation_database, 720
collation_server, 721
completion_type, 721

concurrent_insert, 722
connection_control_failed_connections_threshold, 1263
connection_control_max_connection_delay, 1263
connection_control_min_connection_delay, 1264
connect_timeout, 723
core_file, 723
create_admin_listener_thread, 723
cte_max_recursion_depth, 724
datadir, 724
datetime_format, 725
date_format, 725
debug, 725
debug_sync, 726
default_authentication_plugin, 726
default_collation_for_utf8mb4, 727
default_password_lifetime, 728
default_storage_engine, 728
default_table_encryption, 729
default_tmp_storage_engine, 729
default_week_format, 730
delayed_insert_limit, 731
delayed_insert_timeout, 731
delayed_queue_size, 732
delay_key_write, 730, 3127
disabled_storage_engines, 732
disconnect_on_expired_password, 733
div_precision_increment, 734
dragnet.log_error_filter_rules, 734
end_markers_in_json, 735
error_count, 736
event_scheduler, 736
expire_logs_days, 3284
explicit_defaults_for_timestamp, 737
external_user, 739
flush, 739
flush_time, 739
foreign_key_checks, 740
ft_boolean_syntax, 740
ft_max_word_len, 741
ft_min_word_len, 741
ft_query_expansion_limit, 742
ft_stopword_file, 742
general_log, 743
general_log_file, 743
generated_random_password_length, 743
group_concat_max_len, 744
group_replication_allow_local_disjoint_gtids_join, 3487
group_replication_allow_local_lower_version_join, 3487
group_replication_autorejoin_tries, 3489
group_replication_auto_increment_increment, 3488
group_replication_bootstrap_group, 3489
group_replication_clone_threshold, 3490
group_replication_communication_debug_options, 3491
group_replication_communication_max_message_size, 3492

group_replication_components_stop_timeout, 3492
group_replication_compression_threshold, 3493
group_replication_consistency, 3493
group_replication_enforce_update_everywhere_checks, 3494
group_replication_exit_state_action, 3495
group_replication_flow_control_applier_threshold, 3497
group_replication_flow_control_certifier_threshold, 3497
group_replication_flow_control_hold_percent, 3497
group_replication_flow_control_max_commit_quota, 3498
group_replication_flow_control_member_quota_percent, 3498
group_replication_flow_control_min_quota, 3499
group_replication_flow_control_min_recovery_quota, 3499
group_replication_flow_control_mode, 3499
group_replication_flow_control_period, 3500
group_replication_flow_control_release_percent, 3500
group_replication_force_members, 3501
group_replication_group_name, 3501
group_replication_group_seeds, 3502
group_replication_gtid_assignment_block_size, 3503
group_replication_ip_whitelist, 3503
group_replication_local_address, 3505
group_replication_member_expel_timeout, 3506
group_replication_member_weight, 3506
group_replication_message_cache_size, 3507
group_replication_poll_spin_loops, 3508
group_replication_recovery_complete_at, 3509
group_replication_recovery_compression_algorithm, 3509
group_replication_recovery_get_public_key, 3510
group_replication_recovery_public_key_path, 3510
group_replication_recovery_reconnect_interval, 3510
group_replication_recovery_retry_count, 3511
group_replication_recovery_ssl_ca, 3511
group_replication_recovery_ssl_capath, 3512
group_replication_recovery_ssl_cert, 3512
group_replication_recovery_ssl_cipher, 3512
group_replication_recovery_ssl_crl, 3513
group_replication_recovery_ssl_crlpath, 3513
group_replication_recovery_ssl_key, 3513
group_replication_recovery_ssl_verify_server_cert, 3514
group_replication_recovery_use_ssl, 3514
group_replication_recovery_zstd_compression_level, 3514
group_replication_single_primary_mode, 3515
group_replication_ssl_mode, 3515
group_replication_start_on_boot, 3516
group_replication_transaction_size_limit, 3516
group_replication_unreachable_majority_timeout, 3517
gtid_executed, 3295
gtid_executed_compression_period, 3296
gtid_purged, 3299
have_compress, 744
have_crypt, 744
have_dynamic_loading, 744
have_geometry, 744
have_openssl, 744

have_profiling, 744
have_query_cache, 745
have_rtree_keys, 745
have_ssl, 745
have_statement_timeout, 745
have_symlink, 745
histogram_generation_max_mem_size, 745
hostname, 747
identity, 747
ignore_builtin_innodb, 2934
immediate_server_version, 3217
information_schema_stats_expiry, 747
init_connect, 747
init_file, 749
init_slave, 3241
innodb_adaptive_flushing, 2935
innodb_adaptive_hash_index, 2935
innodb_adaptive_hash_index_parts, 2936
innodb_autoextend_increment, 2939
innodb_autoinc_lock_mode, 2939
innodb_background_drop_list_empty, 2940
innodb_buffer_pool_chunk_size, 2940
innodb_buffer_pool_debug, 2941
innodb_buffer_pool_instances, 2944
innodb_buffer_pool_in_core_file, 2943
innodb_buffer_pool_size, 2946
innodb_change_buffering, 2947
innodb_change_buffering_debug, 2948
innodb_checkpoint_disabled, 2949
innodb_commit_concurrency, 2951
innodb_compress_debug, 2951
innodb_concurrency_tickets, 2953
innodb_data_file_path, 2954
innodb_data_home_dir, 2955
innodb_ddl_log_crash_reset_debug, 2955
innodb_deadlock_detect, 2955
innodb_dedicated_server, 2956
innodb_default_row_format, 2956
innodb_directories, 2957
innodb_disable_sort_file_cache, 2958
innodb_doublewrite, 2958
innodb_fast_shutdown, 2959
innodb_file_per_table, 2960
innodb_fill_factor, 2960
innodb_fil_make_page_dirty_debug, 2959
innodb_flush_log_at_timeout, 2961
innodb_flush_log_at_trx_commit, 2961
innodb_flush_method, 2963
innodb_flush_sync, 2965
innodb_force_recovery, 2966
innodb_fsync_threshold, 2967
innodb_io_capacity, 2973
innodb_limit_optimistic_insert_debug, 2976
innodb_lock_wait_timeout, 2976

innodb_log_buffer_size, 2977
innodb_log_checkpoint_fuzzy_now, 2977
innodb_log_checkpoint_now, 2978
innodb_log_checksums, 2978
innodb_log_files_in_group, 2980
innodb_log_file_size, 2979
innodb_log_group_home_dir, 2980
innodb_log_spin_cpu_abs_lwm, 2980
innodb_log_spin_cpu_pct_hwm, 2981
innodb_log_wait_for_flush_spin_hwm, 2981
innodb_log_write_ahead_size, 2982
innodb_max_dirty_pages_pct, 2983
innodb_max_purge_lag, 2984
innodb_max_purge_lag_delay, 2985
innodb_max_undo_log_size, 2985
innodb_merge_threshold_set_all_debug, 2986
innodb_numa_interleave, 2988
innodb_old_blocks_pct, 2988
innodb_old_blocks_time, 2989
innodb_open_files, 2990
innodb_parallel_read_threads, 2993
innodb_print_ddl_logs, 2994
innodb_purge_batch_size, 2994
innodb_purge_rseg_truncate_frequency, 2995
innodb_purge_threads, 2995
innodb_read_ahead_threshold, 2996
innodb_read_io_threads, 2997
innodb_redo_log_archive_dirs, 2998
innodb_redo_log_encrypt, 2999
innodb_replication_delay, 2999
innodb_rollback_on_timeout, 3000
innodb_saved_page_number_debug, 3001
innodb_scan_directories, 3001
innodb_spin_wait_delay, 3002
innodb_spin_wait_pause_multiplier, 3003
innodb_stats_include_delete_marked, 2861, 3004
innodb_stats_method, 3004
innodb_stats_on_metadata, 3005
innodb_status_output, 3007
innodb_status_output_locks, 3007
innodb_strict_mode, 3008
innodb_sync_debug, 3009
innodb_sync_spin_loops, 3009
innodb_table_locks, 3010
innodb_temp_data_file_path, 3010
innodb_temp_tablespaces_dir, 3011
innodb_thread_concurrency, 3012
innodb_thread_sleep_delay, 3013
innodb_tmpdir, 3014
innodb_trx_purge_view_update_only_debug, 3014
innodb_trx_rseg_n_slots_debug, 3015
innodb_undo_log_encrypt, 3016
innodb_undo_log_truncate, 3016
innodb_use_native_aio, 3018

innodb_version, 3018
innodb_write_io_threads, 3018
insert_id, 749
interactive_timeout, 750
internal_tmp_disk_storage_engine, 750
internal_tmp_mem_storage_engine, 750
join_buffer_size, 751
keep_files_on_create, 752
keyring_aws_cmk_id, 1306
keyring_aws_conf_file, 1306
keyring_aws_data_file, 1307
keyring_aws_region, 1307
keyring_encrypted_file_data, 1308
keyring_encrypted_file_password, 1309
keyring_file_data, 1310
keyring_okv_conf_dir, 1311
keyring_operations, 1312
key_buffer_size, 752
key_cache_age_threshold, 753
key_cache_block_size, 754
key_cache_division_limit, 754
large_files_support, 755
large_pages, 755
large_page_size, 755
last_insert_id, 756
lc_messages, 756
lc_messages_dir, 756
lc_time_names, 756
license, 757
local_infile, 757
locked_in_memory, 758
lock_order, 5041
lock_order_debug_loop, 5042
lock_order_debug_missing_arc, 5042
lock_order_debug_missing_key, 5042
lock_order_debug_missing_unlock, 5043
lock_order_dependencies, 5043
lock_order_extra_dependencies, 5043
lock_order_output_directory, 5044
lock_order_print_txt, 5044
lock_order_trace_loop, 5044
lock_order_trace_missing_arc, 5045
lock_order_trace_missing_key, 5045
lock_order_trace_missing_unlock, 5045
lock_wait_timeout, 757
log_bin, 3285
log_bin_basename, 3285
log_bin_index, 3286
log_bin_trust_function_creators, 3286
log_bin_use_v1_row_events, 3286
log_builtin_as_identified_by_password, 3287
log_error, 758
log_error_filter_rules, 759
log_error_services, 759

log_error_suppression_list, 759
log_error_verbosity, 760
log_output, 761
log_queries_not_using_indexes, 761
log_slave_updates, 3287
log_slow_extra, 762
log_statements_unsafe_for_binlog, 3288
log_syslog, 762
log_syslog_facility, 763
log_syslog_include_pid, 763
log_syslog_tag, 764
log_throttle_queries_not_using_indexes, 764
log_timestamps, 764
log_warnings, 765
long_query_time, 765
lower_case_file_system, 766
lower_case_table_names, 766
low_priority_updates, 766
mandatory_roles, 767
master_info_repository, 3242
master_verify_checksum, 3288
max_allowed_packet, 769
max_binlog_cache_size, 3288
max_binlog_size, 3289
max_binlog_stmt_cache_size, 3290
max_connections, 770
max_connect_errors, 769
max_delayed_threads, 770
max_digest_length, 771
max_error_count, 771
max_execution_time, 772
max_heap_table_size, 772
max_insert_delayed_threads, 773
max_join_size, 413, 773
max_length_for_sort_data, 774
max_points_in_geometry, 774
max_prepared_stmt_count, 774
max_relay_log_size, 3243
max_seeks_for_key, 775
max_sort_length, 775
max_sp_recursion_depth, 776
max_tmp_tables, 776
max_user_connections, 776
max_write_lock_count, 777
mecab_rc_file, 777
metadata_locks_cache_size, 778
metadata_locks_hash_instances, 778
min_examined_row_limit, 779
myisam_data_pointer_size, 779
myisam_max_sort_file_size, 780, 3127
myisam_mmap_size, 780
myisam_recover_options, 781, 3127
myisam_repair_threads, 782
myisam_sort_buffer_size, 782, 3127

myisam_stats_method, 783
myisam_use_mmap, 783
mysqlx_bind_address, 3578
mysqlx_connect_timeout, 3579
mysqlx_document_id_unique_prefix, 3580
mysqlx_enable_hello_notice, 3580
mysqlx_idle_worker_thread_timeout, 3580
mysqlx_interactive_timeout, 3581
mysqlx_max_allowed_packet, 3581
mysqlx_max_connections, 3581
mysqlx_min_worker_threads, 3582
mysqlx_port, 3582
mysqlx_port_open_timeout, 3583
mysqlx_read_timeout, 3583
mysqlx_socket, 3583
mysqlx_ssl_ca, 3584
mysqlx_ssl_capath, 3584
mysqlx_ssl_cert, 3585
mysqlx_ssl_cipher, 3585
mysqlx_ssl_crl, 3585
mysqlx_ssl_crlpath, 3585
mysqlx_ssl_key, 3586
mysqlx_wait_timeout, 3586
mysqlx_write_timeout, 3586
mysql_firewall_mode, 1397
mysql_firewall_trace, 1397
mysql_native_password_proxy_users, 784, 1160
named_pipe, 784
named_pipe_full_access_group, 784
ndb_log_empty_epochs, 3898
ndb_log_empty_update, 3899
ndb_log_exclusive_reads, 3899
ndb_log_orig, 3900
ndb_log_transaction_id, 3900
net_buffer_length, 785
net_read_timeout, 786
net_retry_count, 786
net_write_timeout, 786
new, 787
ngram_token_size, 787
offline_mode, 787
old, 788
old_alter_table, 788
old_passwords, 789
open_files_limit, 789
optimizer_prune_level, 790
optimizer_search_depth, 791
optimizer_switch, 791
optimizer_trace, 794
optimizer_trace_features, 794
optimizer_trace_limit, 794
optimizer_trace_max_mem_size, 794
optimizer_trace_offset, 795
original_commit_timestamp, 3290

original_server_version, 3218
parser_max_mem_size, 795
partial_revokes, 796
password_history, 796
password_require_current, 797
password_reuse_interval, 798
performance_schema, 4667
performance_schema_accounts_size, 4667
performance_schema_digests_size, 4668
performance_schema_error_size, 4668
performance_schema_events_stages_history_long_size, 4669
performance_schema_events_stages_history_size, 4669
performance_schema_events_statements_history_long_size, 4669
performance_schema_events_statements_history_size, 4670
performance_schema_events_transactions_history_long_size, 4670
performance_schema_events_transactions_history_size, 4670
performance_schema_events_waits_history_long_size, 4670
performance_schema_events_waits_history_size, 4671
performance_schema_hosts_size, 4671
performance_schema_max_cond_classes, 4671
performance_schema_max_cond_instances, 4672
performance_schema_max_digest_length, 4672
performance_schema_max_digest_sample_age, 4672
performance_schema_max_file_classes, 4673
performance_schema_max_file_handles, 4674
performance_schema_max_file_instances, 4674
performance_schema_max_index_stat, 4674
performance_schema_max_memory_classes, 4675
performance_schema_max_metadata_locks, 4675
performance_schema_max_mutex_classes, 4675
performance_schema_max_mutex_instances, 4676
performance_schema_max_prepared_statements_instances, 4676
performance_schema_max_program_instances, 4677
performance_schema_max_rwlock_classes, 4677
performance_schema_max_rwlock_instances, 4677
performance_schema_max_socket_classes, 4678
performance_schema_max_socket_instances, 4678
performance_schema_max_sql_text_length, 4678
performance_schema_max_stage_classes, 4679
performance_schema_max_statement_classes, 4679
performance_schema_max_statement_stack, 4680
performance_schema_max_table_handles, 4680
performance_schema_max_table_instances, 4681
performance_schema_max_table_lock_stat, 4681
performance_schema_max_thread_classes, 4681
performance_schema_max_thread_instances, 4682
performance_schema_session_connect_attrs_size, 4682
performance_schema_setup_actors_size, 4683
performance_schema_setup_objects_size, 4683
performance_schema_users_size, 4683
persisted_globals_load, 798, 883
persist_only_admin_x509_subject, 798
pid_file, 799
plugin_dir, 800

port, 800
preload_buffer_size, 800
print_identified_with_as_hex, 801
profiling, 801
profiling_history_size, 801
protocol_compression_algorithms, 801
protocol_version, 802
proxy_user, 802
pseudo_slave_mode, 803
pseudo_thread_id, 803
query_alloc_block_size, 804
query_cache_limit, 804
query_cache_min_res_unit, 805
query_cache_size, 805
query_cache_type, 806
query_cache_wlock_invalidate, 806
query_prealloc_size, 806
rand_seed1, 807
rand_seed2, 807
range_alloc_block_size, 807
range_optimizer_max_mem_size, 808
rbr_exec_mode, 808
read_buffer_size, 809
read_only, 809
read_rnd_buffer_size, 810
regexp_stack_limit, 811
regexp_time_limit, 811
relay_log, 3243
relay_log_basename, 3244
relay_log_index, 3244
relay_log_info_file, 3245
relay_log_info_repository, 3245
relay_log_purge, 3246
relay_log_recovery, 3246
relay_log_space_limit, 3247
report_host, 3247
report_password, 3248
report_port, 3248
report_user, 3249
require_secure_transport, 812
resultset_metadata, 812
rewriter_enabled, 1015
rewriter_verbose, 1015
rpl_read_size, 3249
rpl_semi_sync_master_enabled, 3218
rpl_semi_sync_master_timeout, 3219
rpl_semi_sync_master_trace_level, 3219
rpl_semi_sync_master_wait_for_slave_count, 3220
rpl_semi_sync_master_wait_no_slave, 3220
rpl_semi_sync_master_wait_point, 3221
rpl_semi_sync_slave_enabled, 3250
rpl_semi_sync_slave_trace_level, 3250
rpl_stop_slave_timeout, 3250
schema_definition_cache, 813

secondary_engine_cost_threshold, 813
secure_auth, 814
secure_file_priv, 814
server_id, 815
server_id_bits, 3912
session_track_gtids, 816
session_track_schema, 816
session_track_state_change, 817
session_track_system_variables, 818
session_track_transaction_info, 818
sha256_password_auto_generate_rsa_keys, 819
sha256_password_private_key_path, 820
sha256_password_proxy_users, 820, 1160
sha256_password_public_key_path, 820
shared_memory, 821
shared_memory_base_name, 821
show_compatibility_56, 822
show_create_table_skip_secondary_engine, 822
show_create_table_verbosity, 823
show_old_temporals, 823
skip_external_locking, 823
skip_name_resolve, 824
skip_networking, 824
skip_show_database, 824
slave_checkpoint_group, 3251
slave_checkpoint_period, 3252
slave_compressed_protocol, 3252
slave_exec_mode, 3253
slave_load_tmpdir, 3254
slave_max_allowed_packet, 3254
slave_net_timeout, 3255
slave_parallel_type, 3255
slave_parallel_workers, 3256
slave_pending_jobs_size_max, 3257
slave_preserve_commit_order, 3257
slave_rows_search_algorithms, 3258
slave_skip_errors, 3259
slave_sql_verify_checksum, 3259
slave_transaction_retries, 3259
slave_type_conversions, 3260
slow_launch_time, 825
slow_query_log, 825
slow_query_log_file, 826
socket, 826
sort_buffer_size, 826
sql_auto_is_null, 827
sql_big_selects, 828
sql_buffer_result, 828
sql_log_bin, 3290
sql_log_off, 828
sql_mode, 829
sql_notes, 832
sql_quote_show_create, 832
sql_require_primary_key, 832

sql_safe_updates, 413, 833
sql_select_limit, 413, 834
sql_slave_skip_counter, 3261
sql_warnings, 834
ssl_ca, 834
ssl_capath, 835
ssl_cert, 835
ssl_cipher, 835
ssl_crl, 836
ssl_crlpath, 836
ssl_fips_mode, 836
ssl_key, 837
stored_program_definition_cache, 838
super_read_only, 839
sync_binlog, 3291
sync_master_info, 3261
sync_relay_log, 3262
sync_relay_log_info, 3262
syseventlog.facility, 839
syseventlog.include_pid, 840
syseventlog.tag, 840
system_time_zone, 841
sysvar_stored_program_cache, 838
tablespace_definition_cache, 843
table_definition_cache, 841
table_encryption_privilege_check, 842
table_open_cache, 842
table_open_cache_instances, 843
temptable_max_ram, 844
temptable_use_mmap, 844
thread_cache_size, 845
thread_handling, 845
thread_pool_algorithm, 846
thread_pool_high_priority_connection, 846
thread_pool_max_active_query_threads, 847
thread_pool_max_unused_threads, 847
thread_pool_prio_kickup_timer, 848
thread_pool_size, 848
thread_pool_stall_limit, 849
thread_stack, 849
timestamp, 851
time_format, 850
time_zone, 850
tls_ciphersuites, 851
tls_version, 851
tmpdir, 852
tmp_table_size, 852
transaction_alloc_block_size, 853
transaction_allow_batching, 3913
transaction_isolation, 853
transaction_prealloc_size, 855
transaction_read_only, 855
transaction_write_set_extraction, 3292
tx_isolation, 857

tx_read_only, 857
unique_checks, 858
updatable_views_with_limit, 858
use_secondary_engine, 859
validate_password.check_user_name, 1268
validate_password.dictionary_file, 1269
validate_password.length, 1270
validate_password.mixed_case_count, 1270
validate_password.number_count, 1271
validate_password.policy, 1271
validate_password.special_char_count, 1272
validate_password_check_user_name, 1273
validate_password_dictionary_file, 1274
validate_password_length, 1274
validate_password_mixed_case_count, 1274
validate_password_number_count, 1275
validate_password_policy, 1275
validate_password_special_char_count, 1276
validate_user_plugins, 859
version, 859
version_comment, 860
version_compile_machine, 860
version_compile_os, 860
version_compile_zlib, 860
version_tokens_session, 1029
version_tokens_session_number, 1030
wait_timeout, 861
warning_count, 861
windowing_use_high_precision, 861
system variables, 704, 861, 2684
and replication, 3388
enforce_gtid_consistency, 3294
gtid_mode, 3297
gtid_next, 3297
gtid_owned, 3298
hintable, 1603
mysqld, 583
nonpersistible, 887
persist-restricted, 887
privileges required, 865
SET_VAR optimizer hint, 1603
systemd
CMake SYSTEMD_PID_DIR option, 209
CMake SYSTEMD_SERVICE_NAME option, 210
CMake WITH_SYSTEMD option, 222
managing mysqld, 181
mysqld daemonize option, 676
mysqld exit codes, 941
SYSTEMD_PID_DIR option
CMake, 209
SYSTEMD_SERVICE_NAME option
CMake, 210
system_time_zone system variable, 841
SYSTEM_USER privilege, 1091

SYSTEM_USER privileges, 1123
SYSTEM_USER(), 2075
SYSTEM_VARIABLES_ADMIN privilege, 1092
sysvar_stored_program_cache system variable, 838
sys_config table
  sys schema, 4702
sys_get_config() function
  sys schema, 4777

T

tab (\t), 1671, 2142, 2406
tab option
  mysqldump, 453
  ndb_restore, 4039
table, 6627
  changing, 2250, 2261, 5647
  deleting, 2374
  rebuilding, 271
  repair, 271
  row size, 1893
table aliases, 2423
table cache, 1544
table definition
  retention, 2335
table description
  myisamchk, 526
Table Dump
  thread command, 1656
table is full, 711, 5627
Table is full errors (NDB Cluster), 3782
table lock, 6627
table names
  case sensitivity, 58, 1684
table option
  mysql, 398
  ndb_desc, 3987
table pullout
  semijoin strategy, 1505
table scan, 2845
table type, 6627
  choosing, 3119
table-level locking, 1622
tables
  BLACKHOLE, 3141
  checking, 522
  cloning, 2337
  closing, 1544
  compressed, 534
  compressed format, 3130
  const, 1573
  constant, 1450
  copying, 2338
  counting rows, 296

creating, 284
CSV, 3137
defragment, 3130
defragmenting, 1443, 2615
deleting rows, 5644
displaying, 487
displaying status, 2679
dumping, 436, 470
dynamic, 3129
error checking, 1439
EXAMPLE, 3154
FEDERATED, 3149
flush, 418
fragmentation, 2615
HEAP, 3133
improving performance, 1540
information, 526
information about, 300
InnoDB, 2718
loading data, 285
maintenance, 426
maintenance schedule, 1443
maximum size, 1549
MEMORY, 3133
MERGE, 3143
merging, 3143
moving, 2785
multiple, 299
MyISAM, 3124
names, 1679
open, 1544
opening, 1544
optimizing, 1442
partitioning, 3143
repair, 426
repairing, 1440
retrieving data, 286
selecting columns, 288
selecting rows, 287
sorting rows, 289
symbolic links, 1633
system, 1573
TEMPORARY, 2336
too many, 1545
unique ID for last row, 4939
TABLES
INFORMATION_SCHEMA table, 4416
tables option
mysqlcheck, 435
mysqldump, 456
tables table
data dictionary table, 944
tablespace, 6627
moving, 2785

tablespace encryption
  monitoring, 2802
TABLESPACES
  INFORMATION_SCHEMA table, 4420
tablespaces table
  data dictionary table, 944
tablespace_definition_cache system variable, 843
tablespace_files table
  data dictionary table, 944
tables_priv table
  system table, 946, 1096
TABLE_CONSTRAINTS
  INFORMATION_SCHEMA table, 4421
table_definition_cache system variable, 841
table_distribution_status
  ndbinfo table, 4160
TABLE_ENCRYPTION_ADMIN privilege, 1092
table_encryption_privilege_check variable, 842
table_exists() procedure
  sys schema, 4765
table_fragments
  ndbinfo table, 4161
table_handles table
  performance_schema, 4606
table_info
  ndbinfo table, 4161
table_io_waits_summary_by_index_usage table
  performance_schema, 4639
table_io_waits_summary_by_table table
  performance_schema, 4638
table_lock_waits_summary_by_table table
  performance_schema, 4640
table_open_cache, 1544
table_open_cache system variable, 842
table_open_cache_instances system variable, 843
table_partitions table
  data dictionary table, 944
table_partition_values table
  data dictionary table, 944
TABLE_PRIVILEGES
  INFORMATION_SCHEMA table, 4421
table_replicas
  ndbinfo table, 4162
table_stats table
  data dictionary table, 944
TAN(), 1975
tar
  problems on Solaris, 187, 187
tc-heuristic-recover option
  mysqld, 700
Tcl API, 4941
tcmalloc
  memory allocation library, 357
tcp-ip option

mysqld_multi, 365
TCP/IP, 129, 135, 215, 215, 340, 359, 371, 395, 557, 574, 578, 691, 1056, 1071, 1163, 1642, 4540, 5619
TCP_MAXSEG_SIZE, 3937
TCP_RCV_BUF_SIZE, 3936
TCP_SND_BUF_SIZE, 3936
tc_time_track_stats
ndbinfo table, 4163
tee command
mysql, 403
tee option
mysql, 398
temp-pool option
mysqld, 700
tempdelay option
ndb_import, 4003
temperrors option
ndb_import, 4004
temporal interval syntax, 1725
temporal values
JSON, 1874
temporary files, 5636
temporary table, 6628
TEMPORARY table privileges, 1083, 2336, 2593
temporary tables
and replication, 3382
internal, 1545
problems, 5647
TEMPORARY tables, 2336
renaming, 2379
temporary tablespace, 6628
temptable_max_ram system variable, 844
temptable_use_mmap system variable, 844
terminal monitor
defined, 277
test option
myisampack, 535
test protocol trace plugin, 5002
testing
connection to the server, 1107
installation, 230
postinstallation, 228
testing mysqld
mysqltest, 4944
test_plugin_server authentication plugin, 1239
TEXT
size, 1896
text collection, 6628
TEXT columns
default values, 1847
indexes, 2283, 2283
indexing, 1526, 2314
TEXT data type, 1827, 1846
text files
importing, 410, 461, 2399

text option
  ndb_top, 4061
thd_alloc plugin service, 5011
thd_wait plugin service, 5011
The used command is not allowed with this MySQL version
  error message, 1075
thread, 6628
thread cache, 1642
thread command
  Binlog Dump, 1654
  Change user, 1654
  Close stmt, 1654
  Connect, 1654
  Connect Out, 1654
  Create DB, 1654
  Daemon, 1654
  Debug, 1654
  Delayed insert, 1654
  Drop DB, 1654
  Error, 1654
  Execute, 1654
  Fetch, 1654
  Field List, 1654
  Init DB, 1655
  Kill, 1655
  Long Data, 1655
  Ping, 1655
  Prepare, 1655
  Processlist, 1655
  Query, 1655
  Quit, 1655
  Refresh, 1655
  Register Slave, 1655
  Reset stmt, 1655
  Set option, 1655
  Shutdown, 1655
  Sleep, 1655
  Statistics, 1655
  Table Dump, 1656
  Time, 1656
thread commands, 1654
thread pool plugin
  resource groups, 1650
thread state
  After create, 1656
  altering table, 1657
  Analyzing, 1656
  Changing master, 1665
  Checking master version, 1663
  checking permissions, 1656
  Checking table, 1656
  cleaning up, 1656
  Clearing, 1666
  closing tables, 1656

committing alter table to storage engine, 1657
Committing events to binlog, 1665
Connecting to master, 1663
converting HEAP to ondisk, 1656
copy to tmp table, 1656
Copying to group table, 1656
Copying to tmp table, 1657
Copying to tmp table on disk, 1657
Creating index, 1657
Creating sort index, 1657
creating table, 1657
Creating tmp table, 1657
deleting from main table, 1657
deleting from reference tables, 1657
discard_or_import_tablespace, 1657
end, 1657
executing, 1658
Execution of init_command, 1658
Finished reading one binlog; switching to next binlog, 1662
freeing items, 1658
FULLTEXT initialization, 1658
init, 1658
Initialized, 1666
Killed, 1658
Killing slave, 1664, 1665
Locking system tables, 1658
logging slow query, 1658
login, 1658
Making temporary file (append) before replaying LOAD DATA INFILE, 1664
Making temporary file (create) before replaying LOAD DATA INFILE, 1664
manage keys, 1658
Master has sent all binlog to slave; waiting for more updates, 1663
NULL, 1658
Opening master dump table, 1665
Opening mysql.ndb_apply_status, 1665
Opening system tables, 1658
Opening tables, 1658
optimizing, 1659
preparing, 1659
preparing for alter table, 1659
Processing events, 1666
Processing events from schema table, 1666
Purging old relay logs, 1659
query end, 1659
Queueing master event to the relay log, 1663
Reading event from the relay log, 1664
Reading master dump table data, 1665
Rebuilding the index on master dump table, 1665
Receiving from client, 1659
Reconnecting after a failed binlog dump request, 1663
Reconnecting after a failed master event read, 1663
Registering slave on master, 1663
Removing duplicates, 1659
removing tmp table, 1659

rename, 1659
rename result table, 1659
Reopen tables, 1659
Repair by sorting, 1659
Repair done, 1660
Repair with keycache, 1660
Requesting binlog dump, 1663
Rolling back, 1660
Saving state, 1660
Searching rows for update, 1660
Sending binlog event to slave, 1663
Sending to client, 1660
setup, 1660
Shutting down, 1666
Slave has read all relay log; waiting for more updates, 1664
Sorting for group, 1660
Sorting for order, 1660
Sorting index, 1660
Sorting result, 1660
statistics, 1660
Syncing ndb table schema operation and binlog, 1666
System lock, 1661
update, 1661
Updating, 1661
updating main table, 1661
updating reference tables, 1661
User lock, 1661
User sleep, 1661
Waiting for allowed to take ndbcluster global schema lock, 1666
Waiting for an event from Coordinator, 1665
Waiting for commit lock, 1661
Waiting for event from ndbcluster, 1666
Waiting for first event from ndbcluster, 1666
Waiting for global read lock, 1662
Waiting for handler commit, 1661
Waiting for its turn to commit, 1663
Waiting for master to send event, 1664
Waiting for master update, 1664
Waiting for ndbcluster binlog update to reach current position, 1666
Waiting for ndbcluster global schema lock, 1666
Waiting for ndbcluster to start, 1666
Waiting for next activation, 1666
Waiting for scheduler to stop, 1666
Waiting for schema epoch, 1666
Waiting for schema metadata lock, 1662
Waiting for slave mutex on exit, 1664, 1665
Waiting for Slave Workers to free pending events, 1665
Waiting for stored function metadata lock, 1662
Waiting for stored procedure metadata lock, 1662
Waiting for table flush, 1662
Waiting for table level lock, 1662
Waiting for table metadata lock, 1662
Waiting for tables, 1661
Waiting for the next event in relay log, 1665

Waiting for the slave SQL thread to free enough relay log space, 1664
Waiting for trigger metadata lock, 1662
Waiting on cond, 1662
Waiting on empty queue, 1666
Waiting to finalize termination, 1663
Waiting to reconnect after a failed binlog dump request, 1664
Waiting to reconnect after a failed master event read, 1664
Waiting until MASTER_DELAY seconds after master executed event, 1665
Writing to net, 1662
thread states, 1653
event scheduler, 1666
general, 1656
NDB Cluster, 1665
replication master, 1662
replication slave, 1663, 1664, 1665
thread table
performance_schema, 4656
thread/sql/compress_gtid_table, 3177
threadblocks
ndbinfo table, 4164
ThreadConfig, 3844
threaded clients, 4792
ThreadPool (see DiskIOThreadPool)
threads, 418, 2662, 4943
display, 2662
monitoring, 1653, 2662, 4402, 4656
ndbinfo table, 4165
threadstat
ndbinfo table, 4165
thread_cache_size system variable, 845
thread_handling system variable, 845
thread_pool_algorithm system variable, 846
thread_pool_high_priority_connection system variable, 846
thread_pool_max_active_query_threads system variable, 847
thread_pool_max_unused_threads system variable, 847
thread_pool_prio_kickup_timer system variable, 848
thread_pool_size system variable, 848
thread_pool_stall_limit system variable, 849
thread_stack system variable, 849
Time
thread command, 1656
TIME data type, 1823, 1836
time data types
storage requirements, 1895
time literals, 1672
time representation
Event Scheduler, 4345
time zone problems, 5638
time zone tables, 375
system tables, 947
time zones
and replication, 3383
leap seconds, 934
support, 930

upgrading, 933
TIME(), 1990
TimeBetweenEpochs, 3819
TimeBetweenEpochsTimeout, 3819
TimeBetweenGlobalCheckpoints, 3818, 3856
TimeBetweenGlobalCheckpointsTimeout, 3818
TimeBetweenInactiveTransactionAbortCheck, 3820
TimeBetweenLocalCheckpoints, 3817
TimeBetweenWatchDogCheck, 3812
TimeBetweenWatchDogCheckInitial, 3813
TIMEDIFF(), 1990
timeout, 723, 2063
connect_timeout variable, 426
shutdown_timeout variable, 426
timeout option
ndb_waiter, 4063
timeouts (replication), 3383
TIMESTAMP
and NULL values, 5642
and replication, 3364
indexes, 1538
initialization and updating, 1837
TIMESTAMP data type, 1823, 1834
timestamp system variable, 851
TIMESTAMP(), 1990
TIMESTAMPADD(), 1991
TIMESTAMPDIFF(), 1991
timezone option
mysqld_safe, 360
time_format system variable, 850
TIME_FORMAT(), 1991
TIME_TO_SEC(), 1991
TIME_TRUNCATE_FRACTIONAL SQL mode, 921
time_zone system variable, 850
time_zone table
system table, 947
time_zone_leap_second table
system table, 947
time_zone_name table
system table, 947
time_zone_transition table
system table, 947
time_zone_transition_type table
system table, 947
TINYBLOB data type, 1826
TINYINT data type, 1819
TINYTEXT data type, 1827
tips
optimization, 1522
TLS, 1169
command options, 335
establishing connections, 1170
TLS related options
ALTER USER, 2568

CREATE USER statement, 2579
tls-ciphersuites option, 339
mysql, 398
mysqladmin, 425
mysqlbinlog, 561
mysqlcheck, 435
mysqldump, 445
mysqlimport, 469
mysqlpump, 484
mysqlshow, 494
mysqlslap, 505
mysql_secure_installation, 372
mysql_upgrade, 384
tls-version option, 339
mysql, 398
mysqladmin, 425
mysqlbinlog, 561
mysqlcheck, 435
mysqldump, 445
mysqlimport, 469
mysqlpump, 484
mysqlshow, 494
mysqlslap, 505
mysql_secure_installation, 372
mysql_upgrade, 384
tls_ciphersuites system variable, 851
tls_version system variable, 851
TMPDIR environment variable, 317, 578, 5636
TMPDIR option
CMake, 210
tmpdir option
myisamchk, 525
myisampack, 535
mysqld, 701
tmpdir system variable, 852
tmp_table_size system variable, 852
to-last-log option
mysqlbinlog, 561
tools
command-line, 124, 385
list of, 72
mysqld_multi, 363
mysqld_safe, 355
torn page, 2900, 6628
TotalSendBufferMemory
API and SQL nodes, 3866
data nodes, 3856
management nodes, 3775
TO_BASE64(), 1944
TO_DAYS(), 1992
TO_SECONDS(), 1992
TPS, 6628
TP_THREAD_GROUP_STATE
INFORMATION_SCHEMA table, 4473

tp_thread_group_state table
  performance_schema, 4614
TP_THREAD_GROUP_STATS
  INFORMATION_SCHEMA table, 4473
tp_thread_group_stats table
  performance_schema, 4616
TP_THREAD_STATE
  INFORMATION_SCHEMA table, 4473
tp_thread_state table
  performance_schema, 4618
trace DBI method, 5037
trace files
  ndbmtd, 3957
trace files (NDB Cluster), 3954
TRADITIONAL SQL mode, 916, 922
trailing spaces
  CHAR, 1825, 1843
  ENUM, 1849
  in comparisons, 1843
  SET, 1851
  VARCHAR, 1826, 1843
transaction, 6628
transaction access mode, 2482
transaction consistency guarantees
  understanding, 3431
transaction ID, 6629
transaction isolation level, 2482
  NDB Cluster, 3679
  READ COMMITTED, 2816
  READ UNCOMMITTED, 2818
  REPEATABLE READ, 2816
  SERIALIZABLE, 2818
transaction state
  change tracking, 936
transaction-isolation option
  mysqld, 700
transaction-read-only option
  mysqld, 701
transaction-safe tables, 2718
transactional option
  ndb_delete_all, 3981
TransactionBufferMemory, 3790
TransactionDeadlockDetectionTimeout, 3821
TransactionInactiveTimeout, 3821
transactions, 2810
  and replication, 3383, 3385
  isolation levels, 2815
  metadata locking, 1626
  support, 2718
transaction_alloc_block_size system variable, 853
transaction_allow_batching session variable (NDB Cluster), 3913
transaction_isolation system variable, 853
transaction_prealloc_size system variable, 855
transaction_read_only system variable, 855

transaction_write_set_extraction, 3292
Translators
  list of, 70
transparent data encryption, 2795
transparent page compression, 6629
transportable tablespace, 6629
transporters
  ndbinfo table, 4166
TRIGGER privilege, 1087
triggers, 2365, 2377, 2682, 4331, 4335
  and replication, 3375, 3387
  LAST_INSERT_ID(), 4335
  metadata, 4340
  restrictions, 4363
TRIGGERS
  INFORMATION_SCHEMA table, 4422
triggers option
  mysqldump, 456
  mysqlpump, 484
triggers table
  data dictionary table, 944
TRIM(), 1944
troubleshooting, 5096, 6629
  ALTER TABLE problems, 5647
  C API, 4938
  compiling MySQL server, 225
  connection problems, 1163
  InnoDB deadlocks, 2828, 2830
  InnoDB errors, 3116
  InnoDB recovery problems, 3113
  InnoDB table fragmentation, 2902
  replication, 3393
  startup problems, 236
  with MySQL Enterprise Monitor, 5051
  with MySQL Performance Schema, 4688
TRUE, 1672, 1679
  testing for, 1924, 1924
true literal
  JSON, 1874
truncate, 6629
TRUNCATE TABLE, 2379
  and NDB Cluster, 3677
  and replication, 3388
  performance_schema database, 4523, 4693
TRUNCATE TABLE host_cache
  and FLUSH HOSTS, 4653
TRUNCATE(), 1975
tuning, 1446
  InnoDB compressed tables, 2877
tuple, 6629
tupscan option
  ndb_select_all, 4046
tutorial, 277
twiddle option

ndb_redo_log_reader, 4021
two-phase commit, 898, 898, 6630
TwoPassInitialNodeRestartCopy, 3839
tx_isolation system variable, 857
tx_read_only system variable, 857
type codes
prepared statement C API, 4878
type conversions, 1914, 1920
type option
ibd2sdi, 508
ndb_config, 3977
ndb_show_tables, 4053
types
columns, 1818, 1897
data, 1818
Date and Time, 1833
of tables, 3119
portability, 1898
strings, 1843
typographical conventions, 3
TZ environment variable, 184, 578, 931, 5638
tz-utc option
mysqldump, 454
mysqlpump, 484

U

UCASE(), 1945
UCS-2, 1732
ucs2 character set, 1773
as client character set, 1751
UDF API, 1054
UDF installation
keyring, 1295
UDFs, 1054, 2620, 2621
compiling, 5029
defined, 5020
installing, 1054
return values, 5028
uninstalling, 1054
uid option
mysql_ssl_rsa_setup, 375
ulimit, 5630
UMASK environment variable, 578, 5631
UMASK_DIR environment variable, 578, 5631
unary minus (-), 1965
unbuffered option
mysql, 398
UNCOMPRESS(), 2061
UNCOMPRESSED_LENGTH(), 2062
undo, 6630
undo log, 2768, 2776, 2779, 6630
undo log segment, 6630
undo tablespace, 2779, 6630

undo tablespaces, 2776
UndoDataBuffer, 3826
UndoIndexBuffer, 3825
unexpected halt
  replication, 3346
UNHEX(), 1945
Unicode, 1732
Unicode character (\U), 2142
Unicode Collation Algorithm, 1779
UNINSTALL COMPONENT statement, 2623
UNINSTALL PLUGIN statement, 2624
uninstalling components, 2624
uninstalling plugins, 994, 2624
uninstalling server components, 990
uninstalling UDFs, 1054
UNION, 308, 2436
UNIQUE, 2261
unique constraint, 6630
unique ID, 4939
unique index, 6631
unique key, 6631
  constraint, 63
unique keys
  and partitioning keys, 4324
unique_checks system variable, 858
unique_subquery join type
  optimizer, 1574
Unix
  compiling clients on, 4788
UNIX_TIMESTAMP(), 1993
UNKNOWN
  testing for, 1924, 1924
Unknown column ... in 'on clause', 2435, 2435
unloading
  tables, 286
UNLOCK INSTANCE, 2475
UNLOCK TABLES, 2476
unnamed views, 2444
unpack option
  myisamchk, 525
unqualified option
  ndb_desc, 3987
  ndb_show_tables, 4053
unsafe statement (replication)
  defined, 3310
unsafe statements (replication), 3310
UNSIGNED, 1818, 1830
UNTIL, 2527
updatable views, 4351
updatable_views_with_limit system variable, 858
UPDATE, 61, 2455
update
  thread state, 1661
update option

MySQLInstallerConsole, 127
update option (ndb_index_stat), 4007
UPDATE privilege, 1088
update-state option
myisamchk, 523
UpdateXML(), 2036
Updating
thread state, 1661
updating main table
thread state, 1661
updating reference tables
thread state, 1661
upgrade option
mysqld, 701
MySQLInstallerConsole, 127
upgrade-system-tables option
mysql_upgrade, 384
upgrades
NDB Cluster, 3736, 4085
upgrades and downgrades (NDB Cluster)
compatibility between versions, 3736
upgrading, 243
a Docker installation of MySQL, 270
different architecture, 272
with MySQL SLES Repository, 268, 269
with MySQL Yum Repository, 267
upgrading MySQL, 376
UPPER(), 1945
uptime, 418
URI-like connection string, 343
URLs for downloading MySQL, 79
usage option
ndb_config, 3977
usage option (NDB Cluster programs), 4068
USAGE privilege, 1088
USE, 2706
use command
mysql, 404
USE INDEX, 1606
USE KEY, 1606
use-default option
mysql_secure_installation, 372
use-frm option
mysqlcheck, 435
use-http option
ndb_setup.py, 4052
use-https option
ndb_setup.py, 4052
use-threads option
mysqlimport, 469
useHexFormat option
ndb_select_all, 4046
user
root, 240

user accounts
  altering, 2560
  creating, 1112, 2574
  dual passwords, 1143
  renaming, 2598
  reserved, 1115
  resource limits, 776, 1161, 2570, 2581
USER environment variable, 343, 578
User lock
  thread state, 1661
user management, 1077
user name length
  and replication, 3388
user names
  and passwords, 1078
  in account names, 1104
  in default account, 240
  in role names, 1107
user option, 334
  mysql, 398
  mysqladmin, 425
  mysqlbinlog, 562
  mysqlcheck, 435
  mysqld, 703
  mysqldump, 445
  mysqld_multi, 366
  mysqld_safe, 360
  mysqlimport, 469
  mysqlpump, 484
  mysqlshow, 494
  mysqlslap, 505
  mysql_secure_installation, 372
  mysql_upgrade, 384
  ndb_top, 4061
user privileges
  adding, 1112
  checking, 1114
  deleting, 2585
  dropping, 2585
  revoking, 1115
User sleep
  thread state, 1661
user table
  account_locked column, 1100
  sorting, 1109
  system table, 240, 946, 1096
user variables
  and replication, 3388
USER(), 2075
user-defined functions (see UDFs)
  adding, 5020, 5021
User-defined functions, 2620, 2621
user-defined variables, 1720
users

deleting, 1115, 2585
users option
mysqlpump, 484
users table
performance_schema, 4575
user_defined_functions table
performance_schema, 4661
USER_PRIVILEGES
INFORMATION_SCHEMA table, 4424
user_summary view
sys schema, 4740
user_summary_by_file_io view
sys schema, 4741
user_summary_by_file_io_type view
sys schema, 4741
user_summary_by_stages view
sys schema, 4742
user_summary_by_statement_latency view
sys schema, 4742
user_summary_by_statement_type view
sys schema, 4743
user_variables_by_thread table
performance_schema, 4580
UseShm, 3812
use_invisible_indexes flag
optimizer_switch system variable, 1536
USE_LD_GOLD option
CMake, 216
USE_LD_LLD option
CMake, 216
use_secondary_engine system variable, 859
USING HASH
with NDB tables, 2295
using multiple disks to start data, 1634
using NDB Cluster programs, 3946
USING versus ON
joins, 2434
UTC_DATE(), 1994
UTC_TIME(), 1995
UTC_TIMESTAMP(), 1995
UTF-8, 1732
database object metadata, 1737
utf16 character set, 1773
as client character set, 1751
utf16le character set, 1774
as client character set, 1751
utf16_bin collation, 1785
utf32 character set, 1774
as client character set, 1751
utf8 character set, 1773
alias for utf8mb3, 1772, 1773
utf8mb3 character set, 1772
utf8 alias, 1772, 1773
utf8mb4 character set, 1771

utf8mb4 collations, 1780
utf8mb4_0900_bin
  versus utf8mb4_bin, 1779
utf8mb4_bin
  versus utf8mb4_0900_bin, 1779
utilities
  program-development, 317
utility programs, 316
UUID(), 2225
UUID_SHORT(), 2226
UUID_TO_BIN(), 2227

V

valid
  GIS values, 1865
  spatial values, 1865
valid JSON values, 1875
valid numbers
  examples, 1672
validate-config option
  mysqld, 703
validate-password option
  mysqld, 1273
validate_password component, 1265
  installing, 1267
  status variables, 1272
  system variables, 1268
  uninstalling, 1267
validate_password plugin, 1265
  configuring, 1267
  options, 1272
  status variables, 1276
  system variables, 1273
  transitioning to validate_password component, 1276
validate_password.check_user_name system variable, 1268
validate_password.dictionary_file system variable, 1269
validate_password.dictionary_file_last_parsed status variable, 1272
validate_password.dictionary_file_words_count status variable, 1272
validate_password.length system variable, 1270
validate_password.mixed_case_count system variable, 1270
validate_password.number_count system variable, 1271
validate_password.policy system variable, 1271
validate_password.special_char_count system variable, 1272
validate_password_check_user_name system variable, 1273
validate_password_dictionary_file system variable, 1274
validate_password_dictionary_file_last_parsed status variable, 1276
validate_password_dictionary_file_words_count status variable, 1276
validate_password_length system variable, 1274
validate_password_mixed_case_count system variable, 1274
validate_password_number_count system variable, 1275
validate_password_policy system variable, 1275
validate_password_special_char_count system variable, 1276
validate_user_plugins system variable, 859

VALUES(), 2228
VARBINARY data type, 1826, 1845
VARCHAR
  size, 1896
VARCHAR data type, 1826, 1843
VARCHARACTER data type, 1826
variable option
  mysql_config, 574
variable-length type, 6631
variables
  and replication, 3388
  environment, 317
  server, 2684
  status, 892, 2677
  system, 704, 861, 2684
  user defined, 1720
VARIANCE(), 2181
VAR_POP(), 2180
VAR_SAMP(), 2181
verbose option
  innochecksum, 510
  myisamchk, 520
  myisampack, 536
  myisam_ftdump, 516
  mysql, 398
  mysqladmin, 425
  mysqlbinlog, 562
  mysqlcheck, 435
  mysqld, 704
  mysqldump, 448
  mysqldumpslow, 573
  mysqld_multi, 366
  mysqlimport, 469
  mysqlshow, 494
  mysqlslap, 505
  mysql_config_editor, 544
  mysql_ssl_rsa_setup, 375
  mysql_upgrade, 384
  my_print_defaults, 576
  ndb_blob_tool, 3970
  ndb_import, 4004
  ndb_move_data, 4013
  ndb_perror, 4015
  ndb_restore, 4039
  perror, 577
verbose option (ndbd), 3953
verbose option (ndbmtd), 3953
verbose option (ndb_index_stat), 4009
verbose option (ndb_mgmd), 3966
verify-binlog-checksum option
  mysqlbinlog, 562
version
  choosing, 78
  latest, 79

VERSION file
  CMake, 227
version option
  comp_err, 368
  ibd2sdi, 507
  innochecksum, 510
  myisamchk, 521
  myisampack, 536
  mysql, 398
  mysqladmin, 425
  mysqlbinlog, 562
  mysqlcheck, 435
  mysqld, 704
  mysqldump, 449
  mysqld_multi, 366
  mysqlimport, 469
  mysqlpump, 484
  mysqlshow, 494
  mysqlslap, 505
  mysql_config, 574
  mysql_config_editor, 544
  mysql_ssl_rsa_setup, 375
  my_print_defaults, 576
  ndb_config, 3977
  ndb_perror, 4014
  perror, 577
version option (NDB Cluster programs), 4070
version system variable, 859
Version Tokens, 1018
Version Tokens plugin
  components, 1019
  installing, 1019
  reference, 1026
  uninstalling, 1019
  using, 1020
Version Tokens UDFs
  version_tokens_delete(), 1026
  version_tokens_edit(), 1027
  version_tokens_lock_exclusive(), 1028
  version_tokens_lock_shared(), 1028
  version_tokens_set(), 1027
  version_tokens_show(), 1027
  version_tokens_unlock(), 1028
version view
  sys schema, 4744
VERSION(), 2076
version-check option
  mysql_upgrade, 384
version_comment system variable, 860
version_compile_machine system variable, 860
version_compile_os system variable, 860
version_compile_zlib system variable, 860
version_major() function
  sys schema, 4778

version_minor() function
  sys schema, 4778
version_patch() function
  sys schema, 4779
version_tokens_delete() Version Tokens UDF, 1026
version_tokens_edit() Version Tokens UDF, 1027
version_tokens_lock_exclusive() Version Tokens UDF, 1028
version_tokens_lock_shared() Version Tokens UDF, 1028
version_tokens_session system variable, 1029
version_tokens_session_number system variable, 1030
version_tokens_set() Version Tokens UDF, 1027
version_tokens_show() Version Tokens UDF, 1027
version_tokens_unlock() Version Tokens UDF, 1028
VERSION_TOKEN_ADMIN privilege, 1092
vertical option
  mysql, 399
  mysqladmin, 425
victim, 6631
Vietnamese, 5071
view, 6631
views, 2367, 4331, 4349
  algorithms, 4349
  and replication, 3390
  limitations, 4367
  materialization prevention, 1514
  metadata, 4354
  optimization, 1502, 1512
  privileges, 4367
  problems, 4367
  restrictions, 4367
  roles, 1120
  updatable, 2367, 4351
VIEWS
  INFORMATION_SCHEMA table, 4425
VIEW_ROUTINE_USAGE
  INFORMATION_SCHEMA table, 4426
view_routine_usage table
  data dictionary table, 945
VIEW_TABLE_USAGE
  INFORMATION_SCHEMA table, 4427
view_table_usage table
  data dictionary table, 945
virtual generated column, 6631
virtual index, 6631
  foreign key restrictions, 2753

W

wait, 6632
WAIT COMPLETED (START BACKUP command), 4080
wait option
  myisamchk, 521
  myisampack, 536
  mysql, 399

mysqladmin, 425
WAIT STARTED (START BACKUP command), 4079
wait-nodes option
ndb_waiter, 4063
Waiting for allowed to take ndbcluster global schema lock
thread state, 1666
Waiting for an event from Coordinator
thread state, 1665
Waiting for commit lock
thread state, 1661
Waiting for event from ndbcluster
thread state, 1666
Waiting for event metadata lock
thread state, 1662
Waiting for event read lock
thread state, 1662
Waiting for first event from ndbcluster
thread state, 1666
Waiting for handler commit
thread state, 1661
Waiting for its turn to commit
thread state, 1663
Waiting for master to send event
thread state, 1664
Waiting for master update
thread state, 1664
Waiting for ndbcluster binlog update to reach current position
thread state, 1666
Waiting for ndbcluster global schema lock
thread state, 1666
Waiting for ndbcluster to start
thread state, 1666
Waiting for next activation
thread state, 1666
Waiting for scheduler to stop
thread state, 1666
Waiting for schema epoch
thread state, 1666
Waiting for schema metadata lock
thread state, 1662
Waiting for slave mutex on exit
thread state, 1664, 1665
Waiting for Slave Workers to free pending events
thread state, 1665
Waiting for stored function metadata lock
thread state, 1662
Waiting for stored procedure metadata lock
thread state, 1662
Waiting for table flush
thread state, 1662
Waiting for table level lock
thread state, 1662
Waiting for table metadata lock
thread state, 1662

Waiting for tables
  thread state, 1661
Waiting for the next event in relay log
  thread state, 1665
Waiting for the slave SQL thread to free enough relay log space
  thread state, 1664
Waiting for trigger metadata lock
  thread state, 1662
Waiting on cond
  thread state, 1662
Waiting on empty queue
  thread state, 1666
Waiting to finalize termination
  thread state, 1663
Waiting to reconnect after a failed binlog dump request
  thread state, 1664
Waiting to reconnect after a failed master event read
  thread state, 1664
Waiting until MASTER_DELAY seconds after master executed event
  thread state, 1665
waits_by_host_by_latency view
  sys schema, 4746
waits_by_user_by_latency view
  sys schema, 4746
waits_global_by_latency view
  sys schema, 4747
wait_classes_global_by_avg_latency view
  sys schema, 4744
wait_classes_global_by_latency view
  sys schema, 4745
WAIT_FOR_EXECUTED_GTID_SET(), 2162
wait_timeout system variable, 861
WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS(), 2162
Wan, 3774, 3868
warm backup, 6632
warm up, 6632
warnings command
  mysql, 404
warning_count system variable, 861
WARN_COND_ITEM_TRUNCATED error code, 5139
WARN_DATA_TRUNCATED error code, 5116
WARN_DATA_TRUNCATED_FUNCTIONAL_INDEX error code, 5199
WARN_NAMED_PIPE_ACCESS_EVERYONE error code, 5157
WARN_NON_ASCII_SEPARATOR_NOT_IMPLEMENTED error code, 5139
WARN_NO_MASTER_INFO error code, 5137
WARN_ON_BLOCKHOLE_IN_RBR error code, 5156
WARN_OPTION_BELOW_LIMIT error code, 5144
WARN_OPTION_IGNORED error code, 5137
WARN_PLUGIN_BUSY error code, 5138
WARN_UNENCRYPTED_TABLE_IN_ENCRYPTED_DB error code, 5207
WARN_USELESS_SPATIAL_INDEX error code, 5190
watch-progress option
  mysqlpump, 484
WatchdogImmediateKill, 3832

websites
  MySQL, 51
WEEK(), 1995
WEEKDAY(), 1996
WEEKOFYEAR(), 1996
WEIGHT_STRING(), 1946
well-formed
  GIS values, 1865
  spatial values, 1865
Well-Known Binary format
  geometry values, 1864
Well-Known Text format
  geometry values, 1862
WHERE, 1449
  with SHOW, 4370, 4475
where option
  mysqldump, 456
WHILE, 2528
  labels, 2520
widths
  display, 1818
Wildcard character (%), 1671
Wildcard character (_), 1671
wildcards
  and LIKE, 1530
  in account names, 1106
  in mysql.columns_priv table, 1111
  in mysql.db table, 1110
  in mysql.procs_priv table, 1111
  in mysql.tables_priv table, 1111
window
  window functions, 2200
window functions, 305, 2194
  EXPLAIN, 1500
  optimization, 1499
  syntax, 2200
windowing_use_high_precision system variable, 861
Windows
  compiling clients on, 4789
  interactive history, 411
  MySQL restrictions, 140
  path name separators, 324
  pluggable authentication, 1216
  upgrading, 269
WIN_DEBUG_NO_INLINE option
  CMake, 216
WITH ROLLUP, 2181
WITH_ANT option
  CMake, 216
WITH_ASAN option
  CMake, 216
WITH_ASAN_SCOPE option
  CMake, 216
WITH_AUTHENTICATION_LDAP option

CMake, 216
WITH_AUTHENTICATION_PAM option
CMake, 217
WITH_AWS_SDK option
CMake, 217
WITH_BOOST option
CMake, 217
WITH_BUNDLED_LIBEVENT option
CMake, 224
WITH_BUNDLED_MEMCACHED option
CMake, 224
WITH_CLASSPATH option
CMake, 225
WITH_CLIENT_PROTOCOL_TRACING option
CMake, 217
WITH_CURL option
CMake, 218
WITH_DEBUG option
CMake, 218
WITH_DEFAULT_COMPILER_OPTIONS option
CMake, 223
WITH_DEFAULT_FEATURE_SET option
CMake, 218
WITH_EDITLINE option
CMake, 218
WITH_ERROR_INSERT option
CMake, 225
WITH_GMOCK option
CMake, 219
WITH_ICU option
CMake, 218
WITH_INNODB_EXTRA_DEBUG option
CMake, 218
WITH_INNODB_MEMCACHED option
CMake, 219
WITH_JEMALLOC option
CMake, 219
WITH_KEYRING_TEST option
CMake, 219
WITH_LIBEVENT option
CMake, 219
WITH_LIBWRAP option
CMake, 219
WITH_LOCK_ORDER option
CMake, 219
WITH_LSAN option
CMake, 219
WITH_LTO option
CMake, 220
WITH_LZ4 option
CMake, 220
WITH_LZMA option
CMake, 221
WITH_MECAB option

CMake, 220
WITH_MSAN option
CMake, 220
WITH_MSCRT_DEBUG option
CMake, 220
WITH_MYSQLX option
CMake, 220
WITH_NDBCLUSTER option
CMake, 225
WITH_NDBCLUSTER_STORAGE_ENGINE option
CMake, 225
WITH_NDBMTD option
CMake, 225
WITH_NDB_BINLOG option
CMake, 225
WITH_NDB_DEBUG option
CMake, 225
WITH_NDB_JAVA option
CMake, 225
WITH_NDB_PORT option
CMake, 225
WITH_NDB_TEST option
CMake, 225
WITH_NUMA option
CMake, 220
WITH_PLUGIN_NDBCLUSTER option
CMake, 225
WITH_PROTOBUF option
CMake, 220
WITH_RAPID option
CMake, 221
WITH_RAPIDJSON option
CMake, 221
WITH_RE2 option
CMake, 221
WITH_ROUTER option
CMake, 221
WITH_SSL option
CMake, 221
WITH_SYSTEMD option
CMake, 222
WITH_SYSTEM_LIBS option
CMake, 222
WITH_TEST_TRACE_PLUGIN option
CMake, 222
WITH_TSAN option
CMake, 223
WITH_UBSAN option
CMake, 223
WITH_UNIT_TESTS option
CMake, 223
WITH_UNIXODBC option
CMake, 223
WITH_VALGRIND option

CMake, 223
WITH_ZLIB option
CMake, 223
WITH_ZSTD option
CMake, 223
WKB format
geometry values, 1864
WKT format
geometry values, 1862
wolfSSL, 1170
workload, 6632
wrappers
Eiffel, 4941
write combining, 6632
write option
innochecksum, 512
write-binlog option
mysqlcheck, 435
mysql_upgrade, 384
write_buffer_size myisamchk variable, 521
Writing to net
thread state, 1662

X

X Plugin, 3574
X Plugin option
mysqlx, 3578
x$ views
sys schema, 4705
x$host_summary view
sys schema, 4706
x$host_summary_by_file_io view
sys schema, 4707
x$host_summary_by_file_io_type view
sys schema, 4707
x$host_summary_by_stages view
sys schema, 4707
x$host_summary_by_statement_latency view
sys schema, 4708
x$host_summary_by_statement_type view
sys schema, 4709
x$innodb_buffer_stats_by_schema view
sys schema, 4710
x$innodb_buffer_stats_by_table view
sys schema, 4710
x$innodb_lock_waits view
sys schema, 4711
x$io_by_thread_by_latency view
sys schema, 4713
x$io_global_by_file_by_bytes view
sys schema, 4714
x$io_global_by_file_by_latency view
sys schema, 4715

x$io_global_by_wait_by_bytes view
sys schema, 4715
x$io_global_by_wait_by_latency view
sys schema, 4716
x$latest_file_io view
sys schema, 4717
x$memory_by_host_by_current_bytes view
sys schema, 4718
x$memory_by_thread_by_current_bytes view
sys schema, 4719
x$memory_by_user_by_current_bytes view
sys schema, 4719
x$memory_global_by_current_bytes view
sys schema, 4720
x$memory_global_total view
sys schema, 4720
x$processlist view
sys schema, 4722
x$schema_flattened_keys view
sys schema, 4726
x$schema_index_statistics view
sys schema, 4725
x$schema_tables_with_full_table_scans view
sys schema, 4732
x$schema_table_lock_waits view
sys schema, 4727
x$schema_table_statistics view
sys schema, 4729
x$schema_table_statistics_with_buffer view
sys schema, 4730
x$session view
sys schema, 4733
x$statements_with_errors_or_warnings view
sys schema, 4735
x$statements_with_full_table_scans view
sys schema, 4736
x$statements_with_runtimes_in_95th_percentile view
sys schema, 4737
x$statements_with_sorting view
sys schema, 4738
x$statements_with_temp_tables view
sys schema, 4739
x$statement_analysis view
sys schema, 4733
x$user_summary view
sys schema, 4740
x$user_summary_by_file_io view
sys schema, 4741
x$user_summary_by_file_io_type view
sys schema, 4741
x$user_summary_by_stages view
sys schema, 4742
x$user_summary_by_statement_latency view
sys schema, 4742

x$user_summary_by_statement_type view
  sys schema, 4743
x$waits_by_host_by_latency view
  sys schema, 4746
x$waits_by_user_by_latency view
  sys schema, 4746
x$waits_global_by_latency view
  sys schema, 4747
x$wait_classes_global_by_avg_latency view
  sys schema, 4744
x$wait_classes_global_by_latency view
  sys schema, 4745
X.509/Certificate, 1170
XA, 6632
XA BEGIN, 2486
XA COMMIT, 2486
XA PREPARE, 2486
XA RECOVER, 2486
XA ROLLBACK, 2486
XA START, 2486
XA transactions, 2485
  restrictions, 2489
  transaction identifiers, 2486
XA_RECOVER_ADMIN privilege, 1092
xid
  XA transaction identifier, 2486
xml option
  mysql, 399
  mysqldump, 454
  ndb_config, 3977
XOR
  bitwise, 2054
  logical, 1928

Y

YEAR data type, 1823, 1836
YEAR(), 1996
YEARWEEK(), 1997
Yen sign (Japanese), 5071
young, 6633
Your password does not satisfy the current policy requirements
  password error, 1266

Z

ZEROFILL, 1818, 1830, 4936
zlib_decompress, 317, 578
zstd-compression-level option, 340
  mysql, 399
  mysqladmin, 425
  mysqlbinlog, 562
  mysqlcheck, 435
  mysqldump, 445
  mysqlimport, 469

mysqlpump, 485
mysqlshow, 494
mysqlslap, 505
mysql_upgrade, 384

C Function Index

my_init()
Section 28.7.6, “C API Function Overview”

mysql_affected_rows()
Section 28.7.6, “C API Function Overview”
Section 13.2.1, “CALL Syntax”
Section 12.15, “Information Functions”
Section 13.2.6, “INSERT Syntax”
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.7.49, “mysql_num_rows()”
Section 28.7.11.1, “mysql_stmt_affected_rows()”
Section 28.7.7.81, “mysql_use_result()”
Section 13.2.9, “REPLACE Syntax”
Section 28.7.29.2, “What Results You Can Get from a Query”

mysql_autocommit()
Section 28.7.6, “C API Function Overview”

mysql_binlog_close()
Section 28.7.20, “C API Binary Log Function Overview”
Section 28.7.21.1, “mysql_binlog_close()”

mysql_binlog_fetch()
Section 28.7.19, “C API Binary Log Data Structures”
Section 28.7.20, “C API Binary Log Function Overview”

mysql_binlog_open()
Section 28.7.19, “C API Binary Log Data Structures”
Section 28.7.20, “C API Binary Log Function Overview”
Section 28.7.21.1, “mysql_binlog_close()”

mysql_change_user()
Section 28.7.6, “C API Function Overview”
Section 4.5.1.2, “mysql Client Commands”
Section 28.7.7.3, “mysql_change_user()”
Section 28.7.7.60, “mysql_reset_connection()”

mysql_character_set_name()
Section 28.7.6, “C API Function Overview”

mysql_client_find_plugin()
Section 28.7.6, “C API Function Overview”

mysql_client_register_plugin()

Section 28.7.6, “C API Function Overview”

mysql_close()

Section 28.7.6, “C API Function Overview”
Section B.4.2.10, “Communication Errors and Aborted Connections”
Section 28.7.7.5, “mysql_close()”
Section 28.7.7.6, “mysql_commit()”
Section 28.7.7.7, “mysql_connect()”
Section 28.7.7.37, “mysql_init()”
Section 28.7.7.63, “mysql_rollback()”

mysql_commit()

Section 28.7.6, “C API Function Overview”

mysql_connect()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.5, “mysql_close()”
Section 28.7.7.7, “mysql_connect()”
Section 28.7.7.50, “mysql_options()”
Section 28.7.16.2, “mysql_thread_init()”
Section 28.7.4.3, “Writing C API Threaded Client Programs”

mysql_create_db()

Section 28.7.6, “C API Function Overview”

mysql_data_seek()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.9, “mysql_data_seek()”
Section 28.7.7.64, “mysql_row_seek()”
Section 28.7.7.81, “mysql_use_result()”

mysql_debug()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.10, “mysql_debug()”

mysql_drop_db()

Section 28.7.6, “C API Function Overview”

mysql_dump_debug_info()

Section 28.7.6, “C API Function Overview”

mysql_eof()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.13, “mysql_eof()”

mysql_errno()

Section 6.4.5.4, “Audit Log File Formats”
Section 28.7.7, “C API Function Descriptions”
Section 28.7.6, “C API Function Overview”

Section B.2, “Error Information Interfaces”
Section 28.7.17.1, “mysql_client_find_plugin()”
Section 28.7.17.2, “mysql_client_register_plugin()”
Section 28.7.7.7, “mysql_connect()”
Section 28.7.7.13, “mysql_eof()”
Section 28.7.7.14, “mysql_errno()”
Section 28.7.7.21, “mysql_fetch_row()”
Section 28.7.7.22, “mysql_field_count()”
Section 28.7.17.3, “mysql_load_plugin()”
Section 28.7.7.48, “mysql_num_fields()”
Section 28.7.7.76, “mysql_sqlstate()”
Section 28.7.11.6, “mysql_stmt_close()”
Section 28.7.11.8, “mysql_stmt_errno()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.15.6, “mysql_store_result_nonblocking()”
Section 28.7.7.81, “mysql_use_result()”
Section 13.6.7.5, “SIGNAL Syntax”
Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”

mysql_error()

Section 28.7.7, “C API Function Descriptions”
Section 28.7.6, “C API Function Overview”
Section B.2, “Error Information Interfaces”
Section 28.7.17.1, “mysql_client_find_plugin()”
Section 28.7.17.2, “mysql_client_register_plugin()”
Section 28.7.7.7, “mysql_connect()”
Section 28.7.7.13, “mysql_eof()”
Section 28.7.7.15, “mysql_error()”
Section 28.7.7.21, “mysql_fetch_row()”
Section 28.7.17.3, “mysql_load_plugin()”
Section 28.7.11.6, “mysql_stmt_close()”
Section 28.7.11.9, “mysql_stmt_error()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.15.6, “mysql_store_result_nonblocking()”
Section 28.7.7.81, “mysql_use_result()”
Section 13.6.7.5, “SIGNAL Syntax”
Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”

mysql_escape_string()

Section 28.7.6, “C API Function Overview”
Section 6.1.7, “Client Programming Security Guidelines”
Section 28.7.7.16, “mysql_escape_string()”

mysql_fetch_field()

Section 28.7.5, “C API Data Structures”
Section 28.7.6, “C API Function Overview”
Section 28.7.27, “C API Optional Result Set Metadata”
Section 28.7.7.17, “mysql_fetch_field()”
Section 28.7.7.23, “mysql_field_seek()”
Section 28.7.7.24, “mysql_field_tell()”
Section 28.7.11.23, “mysql_stmt_result_metadata()”

mysql_fetch_field_direct()

Section 28.7.6, “C API Function Overview”
Section 28.7.27, “C API Optional Result Set Metadata”
Section 28.7.11.23, “mysql_stmt_result_metadata()”

mysql_fetch_fields()

Section 28.7.6, “C API Function Overview”
Section 28.7.27, “C API Optional Result Set Metadata”
Section 28.7.11.23, “mysql_stmt_result_metadata()”

mysql_fetch_lengths()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.20, “mysql_fetch_lengths()”
Section 28.7.7.21, “mysql_fetch_row()”

mysql_fetch_row()

Section 28.7.5, “C API Data Structures”
Section 28.7.6, “C API Function Overview”
Section 16.8.1, “FEDERATED Storage Engine Overview”
Section 28.7.7.13, “mysql_eof()”
Section 28.7.7.14, “mysql_errno()”
Section 28.7.7.20, “mysql_fetch_lengths()”
Section 28.7.7.21, “mysql_fetch_row()”
Section 28.7.15.1, “mysql_fetch_row_nonblocking()”
Section 28.7.7.65, “mysql_row_tell()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”
Section 28.7.29.2, “What Results You Can Get from a Query”

mysql_fetch_row_nonblocking()

Section 28.7.14, “C API Asynchronous Function Overview”
Section 28.7.12, “C API Asynchronous Interface”
Section 28.7.7.21, “mysql_fetch_row()”
Section 28.7.15.1, “mysql_fetch_row_nonblocking()”

mysql_field_count()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.22, “mysql_field_count()”
Section 28.7.7.48, “mysql_num_fields()”
Section 28.7.7.53, “mysql_query()”
Section 28.7.7.57, “mysql_real_query()”
Section 28.7.11.23, “mysql_stmt_result_metadata()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.15.6, “mysql_store_result_nonblocking()”
Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”

mysql_field_seek()

Section 28.7.5, “C API Data Structures”
Section 28.7.6, “C API Function Overview”
Section 28.7.7.17, “mysql_fetch_field()”
Section 28.7.7.24, “mysql_field_tell()”

Section 28.7.11.23, “mysql_stmt_result_metadata()”

mysql_field_tell()

Section 28.7.6, “C API Function Overview”
Section 28.7.11.23, “mysql_stmt_result_metadata()”

mysql_free_result()

Section 28.7.6, “C API Function Overview”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section B.4.2.13, “Commands out of sync”
Section 28.7.7.25, “mysql_free_result()”
Section 28.7.15.2, “mysql_free_result_nonblocking()”
Section 28.7.7.42, “mysql_list_dbs()”
Section 28.7.7.43, “mysql_list_fields()”
Section 28.7.7.44, “mysql_list_processes()”
Section 28.7.7.45, “mysql_list_tables()”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.11.23, “mysql_stmt_result_metadata()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”

mysql_free_result_nonblocking()

Section 28.7.14, “C API Asynchronous Function Overview”
Section 28.7.12, “C API Asynchronous Interface”
Section 28.7.7.25, “mysql_free_result()”
Section 28.7.15.2, “mysql_free_result_nonblocking()”

mysql_get_character_set_info()

Section 28.7.6, “C API Function Overview”
Section 10.14.2, “Choosing a Collation ID”

mysql_get_client_info()

Section 28.7.6, “C API Function Overview”
Section 28.7.4.5, “C API Server and Client Library Versions”
Section 28.7.7.7, “mysql_connect()”

mysql_get_client_version()

Section 28.7.6, “C API Function Overview”
Section 28.7.4.5, “C API Server and Client Library Versions”

mysql_get_host_info()

Section 28.7.6, “C API Function Overview”

mysql_get_option()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.50, “mysql_options()”

mysql_get_proto_info()

Section 28.7.6, “C API Function Overview”

mysql_get_server_info()

Section 28.7.6, “C API Function Overview”

Section 28.7.4.5, “C API Server and Client Library Versions”

mysql_get_server_version()

Section 28.7.6, “C API Function Overview”
Section 28.7.4.5, “C API Server and Client Library Versions”

mysql_get_ssl_cipher()

Section 28.7.22, “C API Encrypted Connection Support”
Section 28.7.6, “C API Function Overview”
Section 28.7.7.34, “mysql_get_ssl_cipher()”

mysql_hex_string()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.35, “mysql_hex_string()”

mysql_info()

Section 13.1.9, “ALTER TABLE Syntax”
Section 28.7.6, “C API Function Overview”
Section 13.2.6, “INSERT Syntax”
Section 13.2.7, “LOAD DATA Syntax”
Section 28.7.7.36, “mysql_info()”
Section 28.7.7.50, “mysql_options()”
Section 1.8.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 13.2.12, “UPDATE Syntax”
Section 28.7.29.2, “What Results You Can Get from a Query”

mysql_init()

Section 28.7.6, “C API Function Overview”
Section 28.7.21.1, “mysql_binlog_close()”
Section 28.7.21.2, “mysql_binlog_fetch()”
Section 28.7.21.3, “mysql_binlog_open()”
Section 28.7.7.5, “mysql_close()”
Section 28.7.7.34, “mysql_get_ssl_cipher()”
Section 28.7.7.37, “mysql_init()”
Section 28.7.7.41, “mysql_library_init()”
Section 28.7.7.50, “mysql_options()”
Section 28.7.7.54, “mysql_real_connect()”
Section 28.7.7.77, “mysql_ssl_set()”
Section 28.7.16.2, “mysql_thread_init()”
Section 28.7.4.3, “Writing C API Threaded Client Programs”

mysql_insert_id()

Section 28.7.6, “C API Function Overview”
Section 13.1.20, “CREATE TABLE Syntax”
Section 28.7.29.3, “How to Get the Unique ID for the Last Inserted Row”
Section 12.15, “Information Functions”
Section 13.2.6, “INSERT Syntax”
Section 28.7.7.38, “mysql_insert_id()”
Section 5.1.8, “Server System Variables”
Section 3.6.9, “Using AUTO_INCREMENT”
Section 28.7.29.2, “What Results You Can Get from a Query”

mysql_kill()

Section 28.7.28, “C API Automatic Reconnection Control”
Section 28.7.6, “C API Function Overview”
Section 28.7.7.39, “mysql_kill()”
Section 28.7.7.80, “mysql_thread_id()”

mysql_library_end()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.40, “mysql_library_end()”
Section 28.7.7.41, “mysql_library_init()”
Section 28.7.7.67, “mysql_server_end()”
Section 28.7.7.68, “mysql_server_init()”

mysql_library_init()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.37, “mysql_init()”
Section 28.7.7.41, “mysql_library_init()”
Section 28.7.17.3, “mysql_load_plugin()”
Section 28.7.7.68, “mysql_server_init()”
Section 28.7.16.2, “mysql_thread_init()”
Section 28.7.4.3, “Writing C API Threaded Client Programs”

mysql_list_dbs()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.25, “mysql_free_result()”
Section 28.7.7.42, “mysql_list_dbs()”

mysql_list_fields()

Section 28.7.5, “C API Data Structures”
Section 28.7.6, “C API Function Overview”
Section 28.7.7.43, “mysql_list_fields()”

mysql_list_processes()

Section 28.7.6, “C API Function Overview”

mysql_list_tables()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.45, “mysql_list_tables()”

mysql_load_plugin()

Section 28.7.6, “C API Function Overview”
Client Plugin Descriptors
Section 28.7.17.3, “mysql_load_plugin()”
Section 28.7.17.4, “mysql_load_plugin_v()”

mysql_load_plugin_v()

Section 28.7.6, “C API Function Overview”
Section 28.7.17.3, “mysql_load_plugin()”

mysql_more_results()

Section 28.7.6, “C API Function Overview”
Section 28.7.23, “C API Multiple Statement Execution Support”

Section 28.7.7.46, “mysql_more_results()”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.11.17, “mysql_stmt_next_result()”

mysql_next_result()

Section 28.7.6, “C API Function Overview”
Section 28.7.23, “C API Multiple Statement Execution Support”
Section 13.2.1, “CALL Syntax”
Section 28.7.7.46, “mysql_more_results()”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.15.3, “mysql_next_result_nonblocking()”
Section 28.7.7.54, “mysql_real_connect()”
Section 28.7.7.74, “mysql_set_server_option()”
Section 28.7.7.79, “mysql_store_result()”

mysql_next_result_nonblocking()

Section 28.7.14, “C API Asynchronous Function Overview”
Section 28.7.12, “C API Asynchronous Interface”
Section 28.7.13, “C API Asynchronous Interface Data Structures”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.15.3, “mysql_next_result_nonblocking()”

mysql_num_fields()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.18, “mysql_fetch_field_direct()”
Section 28.7.7.21, “mysql_fetch_row()”
Section 28.7.11.23, “mysql_stmt_result_metadata()”

mysql_num_rows()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.9, “mysql_data_seek()”
Section 28.7.7.49, “mysql_num_rows()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”
Section 28.7.29.2, “What Results You Can Get from a Query”

mysql_options()

Section 6.2.1, “Account User Names and Passwords”
Section 28.7.28, “C API Automatic Reconnection Control”
Section 28.7.17, “C API Client Plugin Functions”
Section 28.7.22, “C API Encrypted Connection Support”
Section 28.7.6, “C API Function Overview”
Section 28.7.27, “C API Optional Result Set Metadata”
Section 28.7.9, “C API Prepared Statement Data Structures”
Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 2.11.4, “Changes in MySQL 8.0”
Client Plugin Descriptors
Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”
Section 10.4, “Connection Character Sets and Collations”
Section 4.2.6, “Connection Compression Control”
Section B.4.2.8, “MySQL server has gone away”
Section 28.7.7.30, “mysql_get_option()”

Section 28.7.7.50, “mysql_options()”
Section 28.7.7.51, “mysql_options4()”
Section 28.7.7.52, “mysql_ping()”
Section 28.7.7.54, “mysql_real_connect()”
Section 28.7.7.61, “mysql_reset_server_public_key()”
Section 28.7.7.77, “mysql_ssl_set()”
Section 28.7.11.11, “mysql_stmt_fetch()”
Section 26.12.9, “Performance Schema Connection Attribute Tables”
Section 6.2.17, “Pluggable Authentication”
Section 29.2.3, “Plugin API Components”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 6.2.16, “Server Handling of Expired Passwords”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 5.8.4, “Using Client Programs in a Multiple-Server Environment”
Using the Authentication Plugins
Using Your Own Protocol Trace Plugins
Section 1.4, “What Is New in MySQL 8.0”

mysql_options4()

Section 28.7.6, “C API Function Overview”
Section B.3.2, “Client Error Message Reference”
Section 28.7.7.50, “mysql_options()”
Section 28.7.7.51, “mysql_options4()”
Section 26.12.9, “Performance Schema Connection Attribute Tables”

mysql_ping()

Section 28.7.28, “C API Automatic Reconnection Control”
Section 28.7.6, “C API Function Overview”
Section B.4.2.8, “MySQL server has gone away”
Section 28.7.7.52, “mysql_ping()”
Section 28.7.7.80, “mysql_thread_id()”

mysql_plugin_options()

Section 28.7.6, “C API Function Overview”

mysql_query()

Section 28.7.6, “C API Function Overview”
Section 28.7.23, “C API Multiple Statement Execution Support”
Section 13.2.1, “CALL Syntax”
Section 28.7.29.3, “How to Get the Unique ID for the Last Inserted Row”
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.8, “mysql_create_db()”
Section 28.7.7.11, “mysql_drop_db()”
Section 28.7.7.17, “mysql_fetch_field()”
Section 28.7.7.39, “mysql_kill()”
Section 28.7.7.43, “mysql_list_fields()”
Section 28.7.7.44, “mysql_list_processes()”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.7.53, “mysql_query()”
Section 28.7.7.54, “mysql_real_connect()”
Section 28.7.7.57, “mysql_real_query()”
Section 28.7.7.58, “mysql_refresh()”
Section 28.7.7.59, “mysql_reload()”

Section 28.7.7.73, “mysql_set_local_infile_handler()”
Section 28.7.7.74, “mysql_set_server_option()”
Section 28.7.7.75, “mysql_shutdown()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”
Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”
Section 28.7.4.3, “Writing C API Threaded Client Programs”

mysql_real_connect()

Section 28.7.22, “C API Encrypted Connection Support”
Section 28.7.6, “C API Function Overview”
Section 28.7.23, “C API Multiple Statement Execution Support”
Section 28.7.27, “C API Optional Result Set Metadata”
Section 13.2.1, “CALL Syntax”
Chapter 12, *Functions and Operators*
Section 12.15, “Information Functions”
Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 13.2.6, “INSERT Syntax”
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.3, “mysql_change_user()”
Section 28.7.7.7, “mysql_connect()”
Section 28.7.7.37, “mysql_init()”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.7.50, “mysql_options()”
Section 28.7.7.54, “mysql_real_connect()”
Section 28.7.15.4, “mysql_real_connect_nonblocking()”
Section 28.7.7.74, “mysql_set_server_option()”
Section 28.7.7.76, “mysql_sqlstate()”
Section 28.7.7.77, “mysql_ssl_set()”
Section 13.5, “Prepared SQL Statement Syntax”
Section 6.2.16, “Server Handling of Expired Passwords”
Section 5.1.8, “Server System Variables”
Section 24.2.1, “Stored Routine Syntax”
Section 5.8.4, “Using Client Programs in a Multiple-Server Environment”

mysql_real_connect_nonblocking()

Section 28.7.14, “C API Asynchronous Function Overview”
Section 28.7.12, “C API Asynchronous Interface”
Section 28.7.7.54, “mysql_real_connect()”
Section 28.7.15.4, “mysql_real_connect_nonblocking()”

mysql_real_escape_string()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.55, “mysql_real_escape_string()”
Section 28.7.7.71, “mysql_set_character_set()”

mysql_real_escape_string_quote()

Section 28.7.6, “C API Function Overview”
Section 6.1.7, “Client Programming Security Guidelines”
Section 28.7.7.16, “mysql_escape_string()”
Section 28.7.7.55, “mysql_real_escape_string()”
Section 28.7.7.56, “mysql_real_escape_string_quote()”

Section 11.5.7, “Populating Spatial Columns”
Section 9.1.1, “String Literals”

mysql_real_query()

Section 28.7.6, “C API Function Overview”
Section 28.7.23, “C API Multiple Statement Execution Support”
Section 13.2.1, “CALL Syntax”
Section 16.8.1, “FEDERATED Storage Engine Overview”
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.7.53, “mysql_query()”
Section 28.7.7.54, “mysql_real_connect()”
Section 28.7.7.57, “mysql_real_query()”
Section 28.7.15.5, “mysql_real_query_nonblocking()”
Section 28.7.7.74, “mysql_set_server_option()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”

mysql_real_query_nonblocking()

Section 28.7.14, “C API Asynchronous Function Overview”
Section 28.7.12, “C API Asynchronous Interface”
Section 28.7.7.57, “mysql_real_query()”
Section 28.7.15.5, “mysql_real_query_nonblocking()”

mysql_refresh()

Section 28.7.6, “C API Function Overview”

mysql_reload()

Section 28.7.6, “C API Function Overview”

mysql_reset_connection()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.3, “mysql_change_user()”
Section 28.7.7.60, “mysql_reset_connection()”

mysql_reset_server_public_key()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.61, “mysql_reset_server_public_key()”

mysql_result_metadata()

Section 28.7.6, “C API Function Overview”
Section 28.7.27, “C API Optional Result Set Metadata”
Section 28.7.7.17, “mysql_fetch_field()”
Section 28.7.7.18, “mysql_fetch_field_direct()”
Section 28.7.7.19, “mysql_fetch_fields()”
Section 28.7.7.62, “mysql_result_metadata()”

mysql_rollback()

Section 28.7.6, “C API Function Overview”

mysql_row_seek()

Section 28.7.6, “C API Function Overview”

Section 28.7.7.64, “mysql_row_seek()”
Section 28.7.7.65, “mysql_row_tell()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”

mysql_row_tell()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.64, “mysql_row_seek()”
Section 28.7.7.65, “mysql_row_tell()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”

mysql_select_db()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.66, “mysql_select_db()”

mysql_server_end()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.67, “mysql_server_end()”
Section 28.7.7.68, “mysql_server_init()”

mysql_server_init()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.68, “mysql_server_init()”
Section 28.7.16.2, “mysql_thread_init()”

mysql_session_track_get_first()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.69, “mysql_session_track_get_first()”
Section 28.7.7.70, “mysql_session_track_get_next()”
Section 5.1.15, “Server Tracking of Client Session State Changes”

mysql_session_track_get_next()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.69, “mysql_session_track_get_first()”
Section 28.7.7.70, “mysql_session_track_get_next()”
Section 5.1.15, “Server Tracking of Client Session State Changes”

mysql_set_character_set()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.26, “mysql_get_character_set_info()”
Section 28.7.7.55, “mysql_real_escape_string()”
Section 28.7.7.56, “mysql_real_escape_string_quote()”

mysql_set_local_infile_default()

Section 28.7.6, “C API Function Overview”

mysql_set_local_infile_handler()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.72, “mysql_set_local_infile_default()”
Section 28.7.7.73, “mysql_set_local_infile_handler()”

mysql_set_server_option()

Section 28.7.6, “C API Function Overview”
Section 28.7.23, “C API Multiple Statement Execution Support”
Section 28.7.7.74, “mysql_set_server_option()”

mysql_shutdown()

Section 28.7.6, “C API Function Overview”
Section 28.7.7.75, “mysql_shutdown()”
Section 6.2.2, “Privileges Provided by MySQL”

mysql_sqlstate()

Section 28.7.6, “C API Function Overview”
Section B.2, “Error Information Interfaces”
Section 28.7.7.14, “mysql_errno()”
Section 28.7.7.76, “mysql_sqlstate()”
Section 28.7.11.6, “mysql_stmt_close()”
Section 28.7.11.27, “mysql_stmt_sqlstate()”
Section 13.6.7.5, “SIGNAL Syntax”

mysql_ssl_set()

Section 28.7.22, “C API Encrypted Connection Support”
Section 28.7.6, “C API Function Overview”
Section 28.7.7.54, “mysql_real_connect()”
Section 28.7.7.77, “mysql_ssl_set()”

mysql_stat()

Section 28.7.6, “C API Function Overview”

mysql_stmt_affected_rows()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.11.1, “mysql_stmt_affected_rows()”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.17, “mysql_stmt_next_result()”
Section 28.7.11.18, “mysql_stmt_num_rows()”

mysql_stmt_attr_get()

Section 28.7.10, “C API Prepared Statement Function Overview”

mysql_stmt_attr_set()

Section 28.7.5, “C API Data Structures”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.9.2, “C API Prepared Statement Type Conversions”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.11, “mysql_stmt_fetch()”
Section 28.7.11.28, “mysql_stmt_store_result()”
Section 13.6.6.5, “Restrictions on Server-Side Cursors”

mysql_stmt_bind_param()

Section 28.7.9, “C API Prepared Statement Data Structures”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.24, “C API Prepared Statement Handling of Date and Time Values”
Section 28.7.11.4, “mysql_stmt_bind_param()”

Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.21, “mysql_stmt_prepare()”
Section 28.7.11.26, “mysql_stmt_send_long_data()”

mysql_stmt_bind_result()

Section 28.7.9, “C API Prepared Statement Data Structures”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.24, “C API Prepared Statement Handling of Date and Time Values”
Section 28.7.11.5, “mysql_stmt_bind_result()”
Section 28.7.11.11, “mysql_stmt_fetch()”
Section 28.7.11.12, “mysql_stmt_fetch_column()”
Section 28.7.11.17, “mysql_stmt_next_result()”
Section 28.7.11.28, “mysql_stmt_store_result()”

mysql_stmt_close()

Section 28.7.9, “C API Prepared Statement Data Structures”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.11.6, “mysql_stmt_close()”
Section 28.7.11.8, “mysql_stmt_errno()”
Section 28.7.11.9, “mysql_stmt_error()”
Section 28.7.11.15, “mysql_stmt_init()”
Section 28.7.11.27, “mysql_stmt_sqlstate()”
Section 26.12.6.4, “The prepared_statements_instances Table”

mysql_stmt_data_seek()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.11.7, “mysql_stmt_data_seek()”
Section 28.7.11.24, “mysql_stmt_row_seek()”
Section 28.7.11.28, “mysql_stmt_store_result()”

mysql_stmt_errno()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section B.2, “Error Information Interfaces”
Section 28.7.11.6, “mysql_stmt_close()”
Section 28.7.11.8, “mysql_stmt_errno()”
Section 28.7.11.11, “mysql_stmt_fetch()”

mysql_stmt_error()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section B.2, “Error Information Interfaces”
Section 28.7.11.6, “mysql_stmt_close()”
Section 28.7.11.9, “mysql_stmt_error()”
Section 28.7.11.11, “mysql_stmt_fetch()”
Section 28.7.11.21, “mysql_stmt_prepare()”

mysql_stmt_execute()

Section 28.7.9, “C API Prepared Statement Data Structures”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.24, “C API Prepared Statement Handling of Date and Time Values”
Section 28.7.9.2, “C API Prepared Statement Type Conversions”
Section 28.7.11.1, “mysql_stmt_affected_rows()”
Section 28.7.11.3, “mysql_stmt_attr_set()”
Section 28.7.11.10, “mysql_stmt_execute()”

Section 28.7.11.11, “mysql_stmt_fetch()”
Section 28.7.11.17, “mysql_stmt_next_result()”
Section 28.7.11.26, “mysql_stmt_send_long_data()”
Section 28.7.11.28, “mysql_stmt_store_result()”
Section 26.12.6.4, “The prepared_statements_instances Table”

mysql_stmt_fetch()

Section 28.7.9, “C API Prepared Statement Data Structures”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.9.2, “C API Prepared Statement Type Conversions”
Section B.3.2, “Client Error Message Reference”
Section 28.7.11.5, “mysql_stmt_bind_result()”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.11, “mysql_stmt_fetch()”
Section 28.7.11.23, “mysql_stmt_result_metadata()”
Section 28.7.11.25, “mysql_stmt_row_tell()”
Section 28.7.11.28, “mysql_stmt_store_result()”

mysql_stmt_fetch_column()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section B.3.2, “Client Error Message Reference”
Section 28.7.11.11, “mysql_stmt_fetch()”

mysql_stmt_field_count()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.11.13, “mysql_stmt_field_count()”

mysql_stmt_free_result()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.11.3, “mysql_stmt_attr_set()”
Section 28.7.11.14, “mysql_stmt_free_result()”
Section 28.7.11.17, “mysql_stmt_next_result()”

mysql_stmt_init()

Section 28.7.9, “C API Prepared Statement Data Structures”
Section 28.7.11, “C API Prepared Statement Function Descriptions”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.8, “C API Prepared Statements”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.21, “mysql_stmt_prepare()”

mysql_stmt_insert_id()

Section 28.7.10, “C API Prepared Statement Function Overview”

mysql_stmt_next_result()

Section 28.7.25, “C API Prepared CALL Statement Support”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 13.2.1, “CALL Syntax”
Section 28.7.11.17, “mysql_stmt_next_result()”

mysql_stmt_num_rows()

Section 28.7.10, “C API Prepared Statement Function Overview”

Section 28.7.11.7, “mysql_stmt_data_seek()”
Section 28.7.11.18, “mysql_stmt_num_rows()”

mysql_stmt_param_count()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.11.10, “mysql_stmt_execute()”

mysql_stmt_param_metadata()

Section 28.7.10, “C API Prepared Statement Function Overview”

mysql_stmt_prepare()

Section 28.7.9, “C API Prepared Statement Data Structures”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.24, “C API Prepared Statement Handling of Date and Time Values”
Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 28.7.11.4, “mysql_stmt_bind_param()”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.13, “mysql_stmt_field_count()”
Section 28.7.11.21, “mysql_stmt_prepare()”
Section 28.7.11.22, “mysql_stmt_reset()”
Section 28.7.11.23, “mysql_stmt_result_metadata()”
Section 13.5, “Prepared SQL Statement Syntax”
Section 26.12.6.4, “The prepared_statements_instances Table”

mysql_stmt_reset()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.11.3, “mysql_stmt_attr_set()”
Section 28.7.11.26, “mysql_stmt_send_long_data()”

mysql_stmt_result_metadata()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.9.2, “C API Prepared Statement Type Conversions”
Section 28.7.11.11, “mysql_stmt_fetch()”
Section 28.7.11.23, “mysql_stmt_result_metadata()”
Section 28.7.11.28, “mysql_stmt_store_result()”

mysql_stmt_row_seek()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.11.24, “mysql_stmt_row_seek()”
Section 28.7.11.25, “mysql_stmt_row_tell()”
Section 28.7.11.28, “mysql_stmt_store_result()”

mysql_stmt_row_tell()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.11.24, “mysql_stmt_row_seek()”
Section 28.7.11.25, “mysql_stmt_row_tell()”
Section 28.7.11.28, “mysql_stmt_store_result()”

mysql_stmt_send_long_data()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section B.3.2, “Client Error Message Reference”

Section 28.7.11.22, “mysql_stmt_reset()”
Section 28.7.11.26, “mysql_stmt_send_long_data()”
Section 5.1.8, “Server System Variables”

mysql_stmt_sqlstate()

Section 28.7.10, “C API Prepared Statement Function Overview”
Section B.2, “Error Information Interfaces”
Section 28.7.11.6, “mysql_stmt_close()”
Section 28.7.11.27, “mysql_stmt_sqlstate()”

mysql_stmt_store_result()

Section 28.7.5, “C API Data Structures”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 28.7.11.3, “mysql_stmt_attr_set()”
Section 28.7.11.7, “mysql_stmt_data_seek()”
Section 28.7.11.11, “mysql_stmt_fetch()”
Section 28.7.11.18, “mysql_stmt_num_rows()”
Section 28.7.11.24, “mysql_stmt_row_seek()”
Section 28.7.11.25, “mysql_stmt_row_tell()”
Section 28.7.11.28, “mysql_stmt_store_result()”

mysql_store_result()

Section 28.7.5, “C API Data Structures”
Section 28.7.6, “C API Function Overview”
Section B.4.2.13, “Commands out of sync”
Section 16.8.1, “FEDERATED Storage Engine Overview”
Section 4.5.1, “`mysql` — The MySQL Command-Line Client”
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.9, “mysql_data_seek()”
Section 28.7.7.13, “mysql_eof()”
Section 28.7.7.17, “mysql_fetch_field()”
Section 28.7.7.21, “mysql_fetch_row()”
Section 28.7.15.1, “mysql_fetch_row_nonblocking()”
Section 28.7.7.22, “mysql_field_count()”
Section 28.7.7.25, “mysql_free_result()”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.7.48, “mysql_num_fields()”
Section 28.7.7.49, “mysql_num_rows()”
Section 28.7.7.64, “mysql_row_seek()”
Section 28.7.7.65, “mysql_row_tell()”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.23, “mysql_stmt_result_metadata()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.15.6, “mysql_store_result_nonblocking()”
Section 28.7.7.81, “mysql_use_result()”
Section 28.7.29.2, “What Results You Can Get from a Query”
Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”
Section 28.7.4.3, “Writing C API Threaded Client Programs”

mysql_store_result_nonblocking()

Section 28.7.14, “C API Asynchronous Function Overview”
Section 28.7.12, “C API Asynchronous Interface”

Section 28.7.7.21, “mysql_fetch_row()”
Section 28.7.15.1, “mysql_fetch_row_nonblocking()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.15.6, “mysql_store_result_nonblocking()”

mysql_thread_end()

Section 28.7.6, “C API Function Overview”
Section 28.7.16.1, “mysql_thread_end()”
Section 28.7.4.3, “Writing C API Threaded Client Programs”

mysql_thread_id()

Section 28.7.28, “C API Automatic Reconnection Control”
Section 28.7.6, “C API Function Overview”
Section 28.7.7.52, “mysql_ping()”
Section 28.7.7.80, “mysql_thread_id()”

mysql_thread_init()

Section 28.7.6, “C API Function Overview”
Section 28.7.16.1, “mysql_thread_end()”
Section 28.7.16.2, “mysql_thread_init()”
Section 28.7.4.3, “Writing C API Threaded Client Programs”

mysql_thread_safe()

Section 28.7.6, “C API Function Overview”

mysql_use_result()

Section 28.7.5, “C API Data Structures”
Section 28.7.6, “C API Function Overview”
Section B.4.2.13, “Commands out of sync”
Section 4.5.1, “`mysql` — The MySQL Command-Line Client”
Section 28.7.7.9, “mysql_data_seek()”
Section 28.7.7.13, “mysql_eof()”
Section 28.7.7.21, “mysql_fetch_row()”
Section 28.7.15.1, “mysql_fetch_row_nonblocking()”
Section 28.7.7.25, “mysql_free_result()”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.7.48, “mysql_num_fields()”
Section 28.7.7.49, “mysql_num_rows()”
Section 28.7.7.64, “mysql_row_seek()”
Section 28.7.7.65, “mysql_row_tell()”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”
Section B.4.2.7, “Out of memory”
Section 28.7.29.2, “What Results You Can Get from a Query”
Section 28.7.4.3, “Writing C API Threaded Client Programs”

mysql_warning_count()

Section 28.7.6, “C API Function Overview”
Section B.2, “Error Information Interfaces”
Section 28.7.7.47, “mysql_next_result()”
Section 13.7.7.40, “SHOW WARNINGS Syntax”

Command Index

A | B | C | D | F | G | H | I | J | K | L | M | N | O | P | R | S | T | U | V | W | X | Y | Z

A

[index top]

Access
Section 13.2.2, “DELETE Syntax”

addgroup
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

addr2line
Section 29.5.1.5, “Using a Stack Trace”

adduser
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

ant
Section 2.9.7, “MySQL Source-Configuration Options”

APF
Section 22.5.12.1, “NDB Cluster Security and Networking Issues”

apt-get
Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”
Section 2.5.5, “Installing MySQL on Linux Using Debian Packages from Oracle”
Section 15.19.5, “Security Considerations for the InnoDB memcached Plugin”

B

[index top]

bash
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 2.4.1, “General Notes on Installing MySQL on macOS”
Section 4.2.1, “Invoking MySQL Programs”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.8.8, “RESTART Syntax”
Section 4.2.7, “Setting Environment Variables”
Section 1.2, “Typographical and Syntax Conventions”

bison
Section 1.9.1, “Contributors to MySQL”
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 2.9.2, “Source Installation Prerequisites”

C

[index top]

cat

Section 15.13, “InnoDB Startup Options and System Variables”
Section 4.5.1.1, “mysql Client Options”

cd

Resetting the Root Password: Windows Systems

chkconfig

Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

clang

Section 28.7.4.1, “Building C API Client Programs”

CMake

Section 10.13, “Adding a Character Set”
Section 28.7.4.2, “Building C API Client Programs Using pkg-config”
Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section B.4.2.16, “Can't initialize character set”
Section 22.2.3.2, “Compiling and Installing NDB Cluster from Source on Windows”
Section 29.2.4.3, “Compiling and Installing Plugin Libraries”
Section 10.5, “Configuring Application Character Set and Collation”
Section 2.9.6, “Configuring SSL Library Support”
Section 14.1, “Data Dictionary Schema”
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section B.4.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 2.9.5, “Installing MySQL Using a Development Source Tree”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 6.4.4.11, “Keyring System Variables”
Section 2.5.9, “Managing MySQL Server with systemd”
Section 4.9, “MySQL Environment Variables”
Chapter 22, *MySQL NDB Cluster 8.0*
Section 22.5.4, “MySQL Server Usage for NDB Cluster”
Section 29.3, “MySQL Services for Plugins”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 26.2, “Performance Schema Build Configuration”
Section 5.8.3, “Running Multiple MySQL Instances on Unix”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 10.3.2, “Server Character Set and Collation”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 2.9.2, “Source Installation Prerequisites”
Section 16.5, “The ARCHIVE Storage Engine”
Section 16.6, “The BLACKHOLE Storage Engine”
Section 16.9, “The EXAMPLE Storage Engine”
Section 16.8, “The FEDERATED Storage Engine”
Section 29.5.3, “The LOCK_ORDER Tool”

Section 1.3.2, “The Main Features of MySQL”
Section 29.4.2.5, “UDF Compiling and Installing”
Section 4.2.2.2, “Using Option Files”
Using the Test Protocol Trace Plugin
Using Your Own Protocol Trace Plugins
Section 1.4, “What Is New in MySQL 8.0”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”

cmake

Section 2.9.10, “Generating MySQL Doxygen Documentation Content”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 15.19.5, “Security Considerations for the InnoDB memcached Plugin”
Section 29.4.2.5, “UDF Compiling and Installing”

cmd

Resetting the Root Password: Windows Systems

cmd.exe

Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 4.2.1, “Invoking MySQL Programs”
Section 1.2, “Typographical and Syntax Conventions”

command.com

Section 4.2.1, “Invoking MySQL Programs”
Section 1.2, “Typographical and Syntax Conventions”

comp_err

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.1, “Overview of MySQL Programs”

configure

Section 1.7, “How to Report Bugs or Problems”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 1.2, “Typographical and Syntax Conventions”

copy

Creating a Data Snapshot Using Raw Data Files

coreadm

Section 2.7, “Installing MySQL on Solaris”
Section 5.1.7, “Server Command Options”

cp

Section 17.1.2.8, “Adding Slaves to a Replication Environment”
Section 17.4.1.2, “Backing Up Raw Data from a Slave”
Section 7.1, “Backup and Recovery Types”
Creating a Data Snapshot Using Raw Data Files

cron

Section B.4.2.2, “Can't connect to [local] MySQL server”

Section 13.7.3.2, “CHECK TABLE Syntax”
Section 16.2.1, “MyISAM Startup Options”
Section 5.4.7, “Server Log Maintenance”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 3.5, “Using mysql in Batch Mode”

csh

Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.7, “Setting Environment Variables”
Section 1.2, “Typographical and Syntax Conventions”

curl

Section 2.9.7, “MySQL Source-Configuration Options”

D

[index top]

daemon_memcached

Section 15.19.6.2, “Adapting a memcached Application for the InnoDB memcached Plugin”
Section 15.19.2, “InnoDB memcached Architecture”

date

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

df

Section B.4.1, “How to Determine What Is Causing a Problem”

dig

Section 1.4, “What Is New in MySQL 8.0”

Directory Utility

Section 2.4.1, “General Notes on Installing MySQL on macOS”

dnf

Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 2.11.7, “Upgrading MySQL with the MySQL Yum Repository”

dnf config-manager

Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”

dnf upgrade

Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”

docker exec

Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”

docker images

Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”

docker inspect
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

docker logs mysqld-container
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

docker ps
Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”

docker pull
Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

docker rm
Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”

docker run
Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

docker stop
Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”

dot
Section 2.9.10, “Generating MySQL Doxygen Documentation Content”

doxygen
Section 2.9.10, “Generating MySQL Doxygen Documentation Content”

dpkg
Section 2.5.5, “Installing MySQL on Linux Using Debian Packages from Oracle”

dump
Creating a Data Snapshot Using Raw Data Files

F

[index top]

flex
Section 2.9.7, “MySQL Source-Configuration Options”
Section 29.5.3, “The LOCK_ORDER Tool”

G

[index top]

gcc
Section 28.7.4.1, “Building C API Client Programs”
Section 2.13.3, “Problems Using the Perl DBI/DBD Interface”
Section 1.9.4, “Tools that were used to create MySQL”

Section 29.4.2.5, “UDF Compiling and Installing”

gdb

Section 29.5.1.1, “Compiling MySQL for Debugging”
Section 29.5.1.4, “Debugging mysqld under gdb”
Section 1.9.4, “Tools that were used to create MySQL”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”

getcap

Section 8.12.5, “Resource Groups”

git branch

Section 2.9.5, “Installing MySQL Using a Development Source Tree”

git checkout

Section 2.9.5, “Installing MySQL Using a Development Source Tree”

gmake

Section 2.8, “Installing MySQL on FreeBSD”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.9.2, “Source Installation Prerequisites”

GnuPG

Section 2.1.3.2, “Signature Checking Using GnuPG”

gnutar

Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 2.9.2, “Source Installation Prerequisites”

gogoc

Section 5.1.12.5, “Obtaining an IPv6 Address from a Broker”

gold

Section 2.9.7, “MySQL Source-Configuration Options”

gpg

Section 2.1.3.2, “Signature Checking Using GnuPG”

grep

Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 3.3.4.7, “Pattern Matching”

groupadd

Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.7, “Installing MySQL on Solaris”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

gtar

Section 2.7, “Installing MySQL on Solaris”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 2.9.2, “Source Installation Prerequisites”

gunzip
Section 6.4.5.5, “Audit Log Logging Configuration”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”

gzip
Section 6.4.5.5, “Audit Log Logging Configuration”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 1.7, “How to Report Bugs or Problems”
Section 2.4, “Installing MySQL on macOS”

H

[index top]

help contents
Section 4.5.1.4, “mysql Client Server-Side Help”

host
Section 1.4, “What Is New in MySQL 8.0”

hostname
Section B.4.2.2, “Can't connect to [local] MySQL server”

I

[index top]

ibd2sdi
Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”
MySQL Glossary
Section 14.6, “Serialized Dictionary Information (SDI)”
Section 1.4, “What Is New in MySQL 8.0”

icc
Section 2.1.5, “Compiler-Specific Build Characteristics”

ifconfig
Section 5.1.12.1, “Verifying System Support for IPv6”

innochecksum
Section 13.7.3.2, “CHECK TABLE Syntax”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
MySQL Glossary
Section 4.1, “Overview of MySQL Programs”

iptables
Section 18.10, “Frequently Asked Questions”
Section 22.5.12.1, “NDB Cluster Security and Networking Issues”

J

[index top]

java
Section 6.4.4.4, “Using the keyring_okv KMIP Plugin”

K

[index top]

kill
Section B.4.2.2, “Can't connect to [local] MySQL server”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”

ksh
Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.7, “Setting Environment Variables”

kswapd
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”

L

[index top]

ldconfig
Section 29.4.2.5, “UDF Compiling and Installing”

ldd libmysqlclient.so
Section 28.7.4.1, “Building C API Client Programs”

ldd mysqld
Section 2.8, “Installing MySQL on FreeBSD”

less
Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

lld
Section 2.9.7, “MySQL Source-Configuration Options”

llvm
Section 2.9.7, “MySQL Source-Configuration Options”

ln
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”

logger
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

lsof +L1
Section B.4.3.5, “Where MySQL Stores Temporary Files”

lz4

Section 4.8.1, “`lz4_decompress` — Decompress mysqlpump LZ4-Compressed Output”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

lz4_decompress

Section 4.8.1, “`lz4_decompress` — Decompress mysqlpump LZ4-Compressed Output”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.1, “Overview of MySQL Programs”
Section 4.8.3, “`zlib_decompress` — Decompress mysqlpump ZLIB-Compressed Output”

M

[index top]

m4

Section 2.9.2, “Source Installation Prerequisites”

make

Section 29.2.4.3, “Compiling and Installing Plugin Libraries”
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 2.8, “Installing MySQL on FreeBSD”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.13.3, “Problems Using the Perl DBI/DBD Interface”
Section 2.9.2, “Source Installation Prerequisites”

make && make install

Section 22.2.2.4, “Building NDB Cluster from Source on Linux”

make install

Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section 29.2.4.3, “Compiling and Installing Plugin Libraries”
Section 2.9.7, “MySQL Source-Configuration Options”

make package

Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.9.7, “MySQL Source-Configuration Options”

make test

Section 2.9.5, “Installing MySQL Using a Development Source Tree”
Section 2.13.1, “Installing Perl on Unix”
Section 29.1.2, “The MySQL Test Suite”

make VERBOSE=1

Section 2.9.8, “Dealing with Problems Compiling MySQL”

md5

Section 2.1.3.1, “Verifying the MD5 Checksum”

md5.exe

Section 2.1.3.1, “Verifying the MD5 Checksum”

md5sum
Section 2.1.3.1, “Verifying the MD5 Checksum”

memcached
Section 15.19.6.2, “Adapting a memcached Application for the InnoDB memcached Plugin”
Section 15.19.6.1, “Adapting an Existing MySQL Schema for the InnoDB memcached Plugin”
Section 15.19.6.5, “Adapting DML Statements to memcached Operations”
Section 15.19.1, “Benefits of the InnoDB memcached Plugin”
Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 15.19.2, “InnoDB memcached Architecture”
Section 15.19.4, “InnoDB memcached Multiple get and Range Query Support”
Section 15.19, “InnoDB memcached Plugin”
Section 15.19.8, “InnoDB memcached Plugin Internals”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 2.9.7, “MySQL Source-Configuration Options”
Section 22.1.1, “NDB Cluster Core Concepts”
Section 15.19.6.6, “Performing DML and DDL Statements on the Underlying InnoDB Table”
Section 15.19.5, “Security Considerations for the InnoDB memcached Plugin”
Section 15.19.3, “Setting Up the InnoDB memcached Plugin”
Section 15.19.7, “The InnoDB memcached Plugin and Replication”
Section 15.19.9, “Troubleshooting the InnoDB memcached Plugin”
Section 15.19.6.3, “Tuning InnoDB memcached Plugin Performance”
Section 1.4, “What Is New in MySQL 8.0”
Section 15.19.6, “Writing Applications for the InnoDB memcached Plugin”

memcapable
Section 15.19.2, “InnoDB memcached Architecture”

memcat
Section 15.19.2, “InnoDB memcached Architecture”

memcp
Section 15.19.2, “InnoDB memcached Architecture”

memflush
Section 15.19.2, “InnoDB memcached Architecture”

memrm
Section 15.19.2, “InnoDB memcached Architecture”

memslap
Section 15.19.6.3, “Tuning InnoDB memcached Plugin Performance”

mgmd
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 22.2, “NDB Cluster Installation”

mkdir
Section 13.1.12, “CREATE DATABASE Syntax”
Section 14.8, “Data Dictionary Limitations”

mklink

Section 8.12.2.3, “Using Symbolic Links for Databases on Windows”

more

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

mv

Section 5.4.2.9, “Error Log File Flushing and Renaming”
Section 5.4.7, “Server Log Maintenance”
Section 5.4.3, “The General Query Log”

my_print_defaults

Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.7, “MySQL Program Development Utilities”
Section 4.1, “Overview of MySQL Programs”

myisam_ftdump

Section 12.9, “Full-Text Search Functions”
Section 4.6.3, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.1, “Overview of MySQL Programs”

myisamchk

Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 13.7.3.2, “CHECK TABLE Syntax”
Section 16.2.3.3, “Compressed Table Characteristics”
Section 16.2.4.1, “Corrupted MyISAM Tables”
Section 7.2, “Database Backup Methods”
Section 29.5.1, “Debugging a MySQL Server”
Section 13.2.2, “DELETE Syntax”
Section 16.2.3.2, “Dynamic Table Characteristics”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.11.5, “External Locking”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 8.3.8, “InnoDB and MyISAM Index Statistics Collection”
Section 8.4.6, “Limits on Table Size”
Section 13.7.8.5, “LOAD INDEX INTO CACHE Syntax”
Section 29.5.1.7, “Making a Test Case If You Experience Table Corruption”
Section 16.2.1, “MyISAM Startup Options”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 7.6.4, “MyISAM Table Optimization”
Section 16.2.3, “MyISAM Table Storage Formats”
Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.4.6, “myisamchk Memory Usage”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”

Section 4.6.4.5, “Obtaining Table Information with myisamchk”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 4.6.4.4, “Other myisamchk Options”
Section 4.1, “Overview of MySQL Programs”
Section 16.2.4.2, “Problems from Tables Not Being Closed Properly”
Section 13.7.3.5, “REPAIR TABLE Syntax”
Section 5.1.8, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 13.7.7.22, “SHOW INDEX Syntax”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 16.2.3.1, “Static (Fixed-Length) Table Characteristics”
Section 25.26, “The INFORMATION_SCHEMA STATISTICS Table”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”
Section 1.3.2, “The Main Features of MySQL”
Section 16.2, “The MyISAM Storage Engine”
Section 7.6.1, “Using myisamchk for Crash Recovery”
Section 29.5.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”

myisamchk *.MYI

Section 7.6.3, “How to Repair MyISAM Tables”

myisamchk tbl_name

Section 7.6.2, “How to Check MyISAM Tables for Errors”

myisamlog

Section 4.6.5, “`myisamlog` — Display MyISAM Log File Contents”
Section 4.1, “Overview of MySQL Programs”

myisampack

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 16.2.3.3, “Compressed Table Characteristics”
Section 13.1.20, “CREATE TABLE Syntax”
Section 8.11.5, “External Locking”
Section 8.4.6, “Limits on Table Size”
Section 16.7.1, “MERGE Table Advantages and Disadvantages”
Section 16.2.3, “MyISAM Table Storage Formats”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.4.5, “Obtaining Table Information with myisamchk”
Section 8.4.1, “Optimizing Data Size”
Section 4.1, “Overview of MySQL Programs”
Section 13.1.20.8, “Silent Column Specification Changes”
Section 16.7, “The MERGE Storage Engine”
Section 16.2, “The MyISAM Storage Engine”

mysql

Section 1.8.2.4, “'--' as the Start of a Comment”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 22.5.15.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”

Section 22.5.15.1, “Adding NDB Cluster Data Nodes Online: General Issues”
Section 7.1, “Backup and Recovery Types”
Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”
Section 28.7.28, “C API Automatic Reconnection Control”
Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 2.11.4, “Changes in MySQL 8.0”
Section B.3.2, “Client Error Message Reference”
Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”
Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 9.6, “Comment Syntax”
Section 22.3, “Configuration of NDB Cluster”
Section 10.5, “Configuring Application Character Set and Collation”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 3.1, “Connecting to and Disconnecting from the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”
Section 5.1.12.4, “Connecting Using IPv6 Nonlocal Host Addresses”
Section 5.1.12.3, “Connecting Using the IPv6 Local Host Address”
Section 10.4, “Connection Character Sets and Collations”
Section 4.2.6, “Connection Compression Control”
Section 1.9.1, “Contributors to MySQL”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 2.11.14, “Copying MySQL Databases to Another Machine”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 3.3.1, “Creating and Selecting a Database”
Section 2.3.4.7, “Customizing the PATH for MySQL Tools”
Section 14.1, “Data Dictionary Schema”
Section 29.5.2, “Debugging a MySQL Client”
Section 22.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”
Section 24.1, “Defining Stored Programs”
Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”
Section 15.16.2, “Enabling InnoDB Monitors”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 3.2, “Entering Queries”
Section B.2, “Error Information Interfaces”
Section B.1, “Error Message Sources and Components”
Section 22.5.7.3, “Event Buffer Reporting in the Cluster Log”
Section 24.4.2, “Event Scheduler Configuration”
Section 7.3, “Example Backup and Recovery Strategy”
Section 28.7.3, “Example C API Client Programs”
Section 3.6, “Examples of Common Queries”
Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 4.5.1.5, “Executing SQL Statements from a Text File”
Chapter 12, *Functions and Operators*
Section 12.17.3, “Functions That Search JSON Values”
Section 2.4.1, “General Notes on Installing MySQL on macOS”
Section 13.6.7.3, “GET DIAGNOSTICS Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 13.8.3, “HELP Syntax”
Section B.4.1, “How to Determine What Is Causing a Problem”
Section 15.7.5.3, “How to Minimize and Handle Deadlocks”

Section 1.7, “How to Report Bugs or Problems”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section 13.2.5, “IMPORT TABLE Syntax”
Section 12.15, “Information Functions”
Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 15.17.2, “InnoDB Recovery”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 4.2.1, “Invoking MySQL Programs”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 8.2.1.19, “LIMIT Query Optimization”
Section 13.2.7, “LOAD DATA Syntax”
Section 13.2.8, “LOAD XML Syntax”
Section 7.4.5.1, “Making a Copy of a Database”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 22.5, “Management of NDB Cluster”
Section 8.13.1, “Measuring the Speed of Expressions and Functions”
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.3, “mysql Client Logging”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.4, “mysql Client Server-Side Help”
Section 4.5.1.6, “mysql Client Tips”
Section 4.9, “MySQL Environment Variables”
MySQL Glossary
Section 2.3.3.1, “MySQL Installer Initial Setup”
Chapter 22, *MySQL NDB Cluster 8.0*
Section 5.1.13, “MySQL Server Time Zone Support”
Section 22.5.4, “MySQL Server Usage for NDB Cluster”
Chapter 19, *MySQL Shell*
Section 4.5.1, “`mysql` — The MySQL Command-Line Client”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 28.7.7.14, “mysql_errno()”
Section 28.7.7.76, “mysql_sqlstate()”
Section 4.4.4, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 22.5.17, “NDB API Statistics Counters and Variables”
Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 22.2.6, “NDB Cluster Example with Tables and Data”
Section 22.6, “NDB Cluster Replication”
Section 22.4.9, “`ndb_desc` — Describe NDB Tables”
Section 22.4.11, “`ndb_drop_table` — Drop an NDB Table”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 22.4.5, “`ndb_mgm` — The NDB Cluster Management Client”

Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 22.4.2, “`ndbinfo_select_all` — Select From ndbinfo Tables”
Section 4.2.2.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section B.4.2.7, “Out of memory”
Section 4.1, “Overview of MySQL Programs”
Section B.4.2.9, “Packet Too Large”
Section 6.4.1.5, “PAM Pluggable Authentication”
Section 26.12.9, “Performance Schema Connection Attribute Tables”
Section 6.2.17, “Pluggable Authentication”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 13.5, “Prepared SQL Statement Syntax”
Section 22.6.5, “Preparing the NDB Cluster for Replication”
Section 4.2.2.4, “Program Option Modifiers”
Section 23.2.3.1, “RANGE COLUMNS partitioning”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 7.4.2, “Reloading SQL-Format Backups”
Resetting the Root Password: Generic Instructions
Restoring to More Nodes Than the Original
Section 13.7.1.8, “REVOKE Syntax”
Section 2.10.4, “Securing the Initial MySQL Account”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 13.2.10.1, “SELECT ... INTO Syntax”
Section B.3.1, “Server Error Message Reference”
Section 6.2.16, “Server Handling of Expired Passwords”
Section 5.1.8, “Server System Variables”
Section 5.1.14, “Server-Side Help Support”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 13.6.7.5, “SIGNAL Syntax”
Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”
Section 4.2.2, “Specifying Program Options”
Section 2.3.4.8, “Starting MySQL as a Windows Service”
Section 22.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 9.1.1, “String Literals”
Section 2.10.3, “Testing the Server”
Section 11.4.3, “The BLOB and TEXT Types”
Section 27.4.4.2, “The diagnostics() Procedure”
Section 22.5.10.27, “The ndbinfo memory_per_fragment Table”
Section 22.5.10.44, “The ndbinfo transporters Table”
Section 24.3.1, “Trigger Syntax and Examples”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Chapter 3, *Tutorial*
Section 1.2, “Typographical and Syntax Conventions”
Section 2.11, “Upgrading MySQL”
Section 7.3.2, “Using Backups for Recovery”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”
Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”
Section 6.4.7.3, “Using MySQL Enterprise Firewall”
Section 3.5, “Using mysql in Batch Mode”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 7.4, “Using mysqldump for Backups”
Section 4.2.2.2, “Using Option Files”

Section 4.2.2.1, “Using Options on the Command Line”
Section 4.2.2.5, “Using Options to Set Program Variables”
Section 29.5.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 22.5.13.2, “Using Symbolic Links with Disk Data Objects”
Using the Test Protocol Trace Plugin
Using Your Own Protocol Trace Plugins
Section 1.4, “What Is New in MySQL 8.0”
Section 22.1.4, “What is New in NDB Cluster”
Section 2.3.6, “Windows Postinstallation Procedures”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”
Section 12.11, “XML Functions”

mysql ...

Section 29.5.1.1, “Compiling MySQL for Debugging”

mysql-server

Section 2.8, “Installing MySQL on FreeBSD”

mysql-test-run.pl

Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 29.5.3, “The LOCK_ORDER Tool”
Section 29.1.2, “The MySQL Test Suite”
Section 4.2.2.2, “Using Option Files”

mysql-test-run.pl test_name

Section 29.1.2, “The MySQL Test Suite”

mysql.exe

Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 4.5.1.6, “mysql Client Tips”

mysql.server

Section 2.5, “Installing MySQL on Linux”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.1, “Overview of MySQL Programs”
Section 5.1.7, “Server Command Options”
Section 2.10.5, “Starting and Stopping MySQL Automatically”
Section B.4.3.7, “Time Zone Problems”

mysql.server stop

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

mysql_client_test_embedded

Section 1.4, “What Is New in MySQL 8.0”

mysql_config

Section 28.7.4.1, “Building C API Client Programs”
Section 28.7.4.2, “Building C API Client Programs Using pkg-config”

Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 28.7.1, “MySQL C API Implementations”
Section 4.7.1, “`mysql_config` — Display Options for Compiling Clients”
Section 4.1, “Overview of MySQL Programs”
Section 29.2.3, “Plugin API Components”
Section 1.4, “What Is New in MySQL 8.0”

mysql_config_editor

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.4.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.9, “MySQL Environment Variables”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 4.1, “Overview of MySQL Programs”
Section 4.2.2.2, “Using Option Files”

mysql_install_db

Section 1.4, “What Is New in MySQL 8.0”

mysql_plugin

Section 1.4, “What Is New in MySQL 8.0”

mysql_secure_installation

Section 2.10.1, “Initializing the Data Directory”
Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”
Section 2.5.5, “Installing MySQL on Linux Using Debian Packages from Oracle”
Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.1, “Overview of MySQL Programs”
Section 2.10.4, “Securing the Initial MySQL Account”

mysql_setpermission

Section 1.9.1, “Contributors to MySQL”

mysql_ssl_rsa_setup

Section 28.7.22, “C API Encrypted Connection Support”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.3.3, “Creating RSA Keys Using openssl”
Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”

Section 6.3.3.2, “Creating SSL Certificates and Keys Using openssl”
Section 2.10.1, “Initializing the Data Directory”
Section 4.4.3, “`mysql_ssl_rsa_setup` — Create SSL/RSA Files”
Section 4.1, “Overview of MySQL Programs”

mysql_stmt_execute()

Section 5.1.10, “Server Status Variables”

mysql_stmt_prepare()

Section 5.1.10, “Server Status Variables”

mysql_tzinfo_to_sql

Section 5.1.13, “MySQL Server Time Zone Support”
Section 4.4.4, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”
Section 4.1, “Overview of MySQL Programs”

mysql_upgrade

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 4.2.6, “Connection Compression Control”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 22.6.4, “NDB Cluster Replication Schema and Tables”
Section 4.1, “Overview of MySQL Programs”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 17.5.3, “Upgrading a Replication Setup”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 2.11.10, “Upgrading MySQL on Windows”
Section 2.11.7, “Upgrading MySQL with the MySQL Yum Repository”
Section 1.4, “What Is New in MySQL 8.0”
Section 2.11.3, “What the MySQL Upgrade Process Upgrades”

mysqlaccess

Section 1.9.1, “Contributors to MySQL”

mysqladmin

Section 6.2.14, “Assigning Account Passwords”
Section 17.4.1.1, “Backing Up a Slave Using mysqldump”
Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section B.4.2.2, “Can't connect to [local] MySQL server”
Section 2.11.4, “Changes in MySQL 8.0”
Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”
Section 5.1.1, “Configuring the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 10.4, “Connection Character Sets and Collations”
Section 4.2.6, “Connection Compression Control”
Section 1.9.1, “Contributors to MySQL”
Section 13.1.12, “CREATE DATABASE Syntax”
Section 2.3.4.7, “Customizing the PATH for MySQL Tools”

Section 29.5.1, “Debugging a MySQL Server”
Section 13.1.24, “DROP DATABASE Syntax”
Section 28.7.3, “Example C API Client Programs”
Section 13.7.8.3, “FLUSH Syntax”
Section 2.4.1, “General Notes on Installing MySQL on macOS”
Section B.4.1, “How to Determine What Is Causing a Problem”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 2.3.3.1, “MySQL Installer Initial Setup”
Section 5.4, “MySQL Server Logs”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.1, “Overview of MySQL Programs”
Section 26.12.9, “Performance Schema Connection Attribute Tables”
Section 6.2.17, “Pluggable Authentication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.8.3, “Running Multiple MySQL Instances on Unix”
Section 2.10.4, “Securing the Initial MySQL Account”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 2.3.4.8, “Starting MySQL as a Windows Service”
Section 2.3.4.6, “Starting MySQL from the Windows Command Line”
Section 2.10.3, “Testing the Server”
Section 1.3.2, “The Main Features of MySQL”
Section 5.1.17, “The Server Shutdown Process”
Section 2.11.10, “Upgrading MySQL on Windows”
Section 4.2.2.2, “Using Option Files”
Section 4.2.2.1, “Using Options on the Command Line”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”

mysqladmin debug

Section 29.5.1, “Debugging a MySQL Server”
Section 24.4.5, “Event Scheduler Status”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”

mysqladmin extended-status

Section 13.7.7.35, “SHOW STATUS Syntax”

mysqladmin flush-hosts

Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section B.4.2.5, “Host 'host_name' is blocked”
Section 5.1.8, “Server System Variables”
Section 26.12.18.1, “The host_cache Table”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”

mysqladmin flush-logs

Section 7.3.3, “Backup Strategy Summary”
Section 5.4.2.9, “Error Log File Flushing and Renaming”
Section 7.3.1, “Establishing a Backup Policy”

Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 5.4.7, “Server Log Maintenance”
Section 5.4.4, “The Binary Log”
Section 17.2.4.1, “The Slave Relay Log”

mysqladmin flush-privileges

Section 2.11.14, “Copying MySQL Databases to Another Machine”
Section 6.2.3, “Grant Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 5.1.7, “Server Command Options”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 6.2.13, “When Privilege Changes Take Effect”

mysqladmin flush-tables

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 8.11.5, “External Locking”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 7.6.1, “Using myisamchk for Crash Recovery”

mysqladmin flush-xxx

Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”

mysqladmin kill

Section B.4.3.4, “How MySQL Handles a Full Disk”
Section 13.7.8.4, “KILL Syntax”
Section 12.14, “Locking Functions”
Section B.4.2.8, “MySQL server has gone away”
Section 6.2.2, “Privileges Provided by MySQL”

mysqladmin password

Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

mysqladmin processlist

Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 8.14, “Examining Thread Information”
Section 13.7.8.4, “KILL Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 29.1.1, “MySQL Threads”
Section 28.7.7.44, “mysql_list_processes()”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
Section 25.19, “The INFORMATION_SCHEMA PROCESSLIST Table”

mysqladmin processlist status

Section 29.5.1, “Debugging a MySQL Server”

mysqladmin refresh

Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 5.4.7, “Server Log Maintenance”

mysqladmin reload

Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 6.2.3, “Grant Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 5.1.7, “Server Command Options”
Section 6.2.20, “Setting Account Resource Limits”
Section 6.2.13, “When Privilege Changes Take Effect”

mysqladmin reload version

Section 1.7, “How to Report Bugs or Problems”

mysqladmin shutdown

Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 29.5.1.2, “Creating Trace Files”
Section 13.7.1.6, “GRANT Syntax”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section 2.4.2, “Installing MySQL on macOS Using Native Packages”
Section 29.5.1.7, “Making a Test Case If You Experience Table Corruption”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 22.2.7, “Safe Shutdown and Restart of NDB Cluster”
Section 13.7.8.9, “SHUTDOWN Syntax”
Section 2.3.4.8, “Starting MySQL as a Windows Service”
Section 5.1.17, “The Server Shutdown Process”
Section 18.7.3.2, “Upgrading a Group Replication Member”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”

mysqladmin status

Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 28.7.7.78, “mysql_stat()”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

mysqladmin variables

Section B.4.2.8, “MySQL server has gone away”
Section 13.7.7.39, “SHOW VARIABLES Syntax”

mysqladmin variables extended-status processlist

Section 1.7, “How to Report Bugs or Problems”

mysqladmin ver

Section 29.5.1.1, “Compiling MySQL for Debugging”

mysqladmin version

Section B.4.2.2, “Can't connect to [local] MySQL server”
Section 1.7, “How to Report Bugs or Problems”
Section B.4.2.8, “MySQL server has gone away”

Section 2.10.3, “Testing the Server”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”

mysqlanalyze

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

mysqlbackup

Section 7.1, “Backup and Recovery Types”
Creating a Data Snapshot Using Raw Data Files
Section 18.7.3.4, “Group Replication Upgrade with `mysqlbackup`”
Section 15.17.1, “InnoDB Backup”
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”
Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”

mysqlbinlog

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 13.7.8.1, “BINLOG Syntax”
Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 4.2.6, “Connection Compression Control”
Section 17.4.10, “Delayed Replication”
Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”
Section 17.1.3.1, “GTID Format and Storage”
Section 17.1.3.2, “GTID Life Cycle”
Section 17.5.5, “How to Report Replication Bugs or Problems”
Section 15.17.2, “InnoDB Recovery”
Section B.4.7, “Known Issues in MySQL”
Section 12.24, “Miscellaneous Functions”
Section 4.5.1.1, “mysql Client Options”
MySQL Glossary
MySQL Server Options for NDB Cluster
Section 4.6.8.1, “mysqlbinlog Hex Dump Format”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 23.3.5, “Obtaining Information About Partitions”
Section 4.1, “Overview of MySQL Programs”
Section 26.12.9, “Performance Schema Connection Attribute Tables”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 7.5.2, “Point-in-Time Recovery Using Event Positions”
Section 7.5.1, “Point-in-Time Recovery Using Event Times”
Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.3.3.2, “Recovering From Failed Replication Privilege Checks”
Section 17.5.1.19, “Replication and LOAD DATA”
Section 17.5.1.38, “Replication and Variables”
Section 17.3.3, “Replication Privilege Checks”
Section 17.3.2.1, “Scope of Binary Log Encryption”
Section 5.1.8, “Server System Variables”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”

Section 13.7.7.2, “SHOW BINLOG EVENTS Syntax”
Section 13.7.7.32, “SHOW RELAYLOG EVENTS Syntax”
Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”
Section 13.4.2.6, “START SLAVE Syntax”
Section 5.4.4, “The Binary Log”
Section 5.4.3, “The General Query Log”
Section 17.2.4.1, “The Slave Relay Log”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”
Section 7.3.2, “Using Backups for Recovery”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

mysqlbinlog binary-log-file | mysql

Section 29.5.1.7, “Making a Test Case If You Experience Table Corruption”

mysqlbinlog|mysql

Section B.4.7, “Known Issues in MySQL”

mysqlcheck

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 2.11.4, “Changes in MySQL 8.0”
Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”
Section 10.4, “Connection Character Sets and Collations”
Section 4.2.6, “Connection Compression Control”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.1, “Overview of MySQL Programs”
Section 2.11.5, “Preparing Your Installation for Upgrade”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”
Section 5.1.8, “Server System Variables”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 1.3.2, “The Main Features of MySQL”
Section 16.2, “The MyISAM Storage Engine”
Section 1.4, “What Is New in MySQL 8.0”

mysqld

Section 29.4.2, “Adding a New User-Defined Function”
Section 29.4, “Adding New Functions to MySQL”
Section 8.2.1.23, “Avoiding Full Table Scans”
Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 5.4.4.1, “Binary Logging Formats”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section B.4.2.2, “Can't connect to [local] MySQL server”
Section B.4.2.12, “Can't create/write to file”
Section B.4.2.16, “Can't initialize character set”
Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 9.6, “Comment Syntax”
Section B.4.2.10, “Communication Errors and Aborted Connections”
Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

Section 29.5.1.1, “Compiling MySQL for Debugging”
Section 22.3, “Configuration of NDB Cluster”
Section 2.9.6, “Configuring SSL Library Support”
Section 5.1.1, “Configuring the Server”
Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 16.2.4.1, “Corrupted MyISAM Tables”
Section 13.7.4.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 13.1.20, “CREATE TABLE Syntax”
Section 29.5.1.2, “Creating Trace Files”
Section 14.1, “Data Dictionary Schema”
Section 15.7.5, “Deadlocks in InnoDB”
Section 29.5.1, “Debugging a MySQL Server”
Section 29.5, “Debugging and Porting MySQL”
Section 29.5.1.4, “Debugging mysqld under gdb”
Section 5.4.2.2, “Default Error Log Destination Configuration”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”
Section 15.6.4, “Doublewrite Buffer”
Section 15.16.2, “Enabling InnoDB Monitors”
Section 8.2.1.5, “Engine Condition Pushdown Optimization”
Section 5.4.2.9, “Error Log File Flushing and Renaming”
Section 5.4.2.8, “Error Log Message Format”
Section 5.4.2.7, “Error Logging to the System Log”
Section 22.5.7.3, “Event Buffer Reporting in the Cluster Log”
Section 15.8.3.7, “Excluding Buffer Pool Pages from Core Files”
Section 8.11.5, “External Locking”
Section B.4.2.17, “File Not Found and Similar Errors”
Section 15.6.3.2, “File-Per-Table Tablespaces”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 15.20.2, “Forcing InnoDB Recovery”
Section 18.10, “Frequently Asked Questions”
Section 22.6.2, “General Requirements for NDB Cluster Replication”
Section 8.14.2, “General Thread States”
Section B.4.2.5, “Host 'host_name' is blocked”
Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 8.12.3.1, “How MySQL Uses Memory”
Section B.4.1, “How to Determine What Is Causing a Problem”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section B.4.2.14, “Ignoring user”
Section 22.6.8, “Implementing Failover with NDB Cluster Replication”
Section 12.15, “Information Functions”
Section 22.2.4, “Initial Configuration of NDB Cluster”
Section 22.2.5, “Initial Startup of NDB Cluster”
Section 2.10.1, “Initializing the Data Directory”
Section 15.17.1, “InnoDB Backup”
Section 15.11.1, “InnoDB Disk I/O”
Section 15.19.2, “InnoDB memcached Architecture”
Section 15.17.2, “InnoDB Recovery”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.20, “InnoDB Troubleshooting”
Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”

Section 13.2.6, “INSERT Syntax”
Section 22.2.2, “Installation of NDB Cluster on Linux”
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.4.2, “Installing MySQL on macOS Using Native Packages”
Section 2.3, “Installing MySQL on Microsoft Windows”
Section 2.7, “Installing MySQL on Solaris”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 21.1, “Introducing InnoDB Cluster”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 13.7.8.4, “KILL Syntax”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 13.2.7, “LOAD DATA Syntax”
Section 29.5.1.7, “Making a Test Case If You Experience Table Corruption”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 22.5, “Management of NDB Cluster”
Section 2.5.9, “Managing MySQL Server with systemd”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”
Section 12.24, “Miscellaneous Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 15.6.1.2, “Moving or Copying InnoDB Tables”
Section 16.2.1, “MyISAM Startup Options”
Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section A.1, “MySQL 8.0 FAQ: General”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”
Section 4.5.1.6, “mysql Client Tips”
Section 4.9, “MySQL Environment Variables”
MySQL Glossary
Section 2.3.1, “MySQL Installation Layout on Microsoft Windows”
Chapter 22, *MySQL NDB Cluster 8.0*
Chapter 5, *MySQL Server Administration*
Section 4.3, “MySQL Server and Server-Startup Programs”
Section B.4.2.8, “MySQL server has gone away”
Section 5.4, “MySQL Server Logs”
MySQL Server Options for NDB Cluster
Section 5.1.13, “MySQL Server Time Zone Support”
Section 22.5.4, “MySQL Server Usage for NDB Cluster”
Section 22.1.6, “MySQL Server Using InnoDB Compared with NDB Cluster”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 1.8, “MySQL Standards Compliance”
Chapter 27, *MySQL sys Schema*
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.50, “mysql_options()”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.1, “`mysqld` — The MySQL Server”

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 22.5.17, “NDB API Statistics Counters and Variables”
Section 22.5.12.3, “NDB Cluster and MySQL Security Procedures”
Section 22.2.1.1, “NDB Cluster Auto-Installer Requirements”
Section 22.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 22.1.1, “NDB Cluster Core Concepts”
Section 22.2, “NDB Cluster Installation”
Section 22.3.2.5, “NDB Cluster mysqld Option and Variable Reference”
Section 22.1.2, “NDB Cluster Nodes, Node Groups, Replicas, and Partitions”
Section 22.1, “NDB Cluster Overview”
Section 22.4, “NDB Cluster Programs”
Section 22.6, “NDB Cluster Replication”
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
Section 22.6.4, “NDB Cluster Replication Schema and Tables”
Section 22.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”
NDB Cluster Status Variables
NDB Cluster System Variables
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.4.27, “`ndb_show_tables` — Display List of NDB Tables”
Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 22.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 13.7.3.4, “OPTIMIZE TABLE Syntax”
Section B.4.5, “Optimizer-Related Issues”
Section 8.5.4, “Optimizing InnoDB Redo Logging”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 22.1.5, “Options, Variables, and Parameters Added, Deprecated or Removed in NDB 8.0”
Section 4.1, “Overview of MySQL Programs”
Section 22.3.2, “Overview of NDB Cluster Configuration Parameters, Options, and Variables”
Section 29.2.4.1, “Overview of Plugin Writing”
Section B.4.2.9, “Packet Too Large”
Section 26.3, “Performance Schema Startup Configuration”
Section 26.12.13.2, “Performance Schema variables_info Table”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 22.6.5, “Preparing the NDB Cluster for Replication”
Section 16.2.4.2, “Problems from Tables Not Being Closed Properly”
Section B.4.3.1, “Problems with File Permissions”
Section 4.2.2.4, “Program Option Modifiers”
Section 22.3.3.2, “Recommended Starting Configuration for NDB Cluster”
Section 13.7.3.5, “REPAIR TABLE Syntax”
Section 17.1.6.1, “Replication and Binary Logging Option and Variable Reference”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 17.5.1.27, “Replication and Master or Slave Shutdowns”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.2.4, “Replication Relay and Status Logs”
Section 13.4.2.4, “RESET SLAVE Syntax”
Resetting the Root Password: Unix and Unix-Like Systems
Resetting the Root Password: Windows Systems

Section 8.12.5, “Resource Groups”
Section 13.7.8.8, “RESTART Syntax”
Restoring to More Nodes Than the Original
Section B.4.4.5, “Rollback Failure for Nontransactional Tables”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 2.10.4, “Securing the Initial MySQL Account”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 6.1.4, “Security-Related mysqld Options and Variables”
Section 13.2.10.1, “SELECT ... INTO Syntax”
Section 2.3.4.3, “Selecting a MySQL Server Type”
Section 10.3.2, “Server Character Set and Collation”
Section 5.1.7, “Server Command Options”
Section 5.4.7, “Server Log Maintenance”
Server Plugin Status and System Variables
Section 5.1.16, “Server Response to Signals”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 10.12, “Setting the Error Message Language”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Section 15.19.3, “Setting Up the InnoDB memcached Plugin”
Section 13.7.7.15, “SHOW ENGINE Syntax”
Section 28.7.2, “Simultaneous MySQL Server and Connector/C Installations”
Section 4.2.2, “Specifying Program Options”
Section 13.4.2.6, “START SLAVE Syntax”
Section 2.10.5, “Starting and Stopping MySQL Automatically”
Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 5.8.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.3.4.8, “Starting MySQL as a Windows Service”
Section 2.3.4.6, “Starting MySQL from the Windows Command Line”
Section 22.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 17.2.3.3, “Startup Options and Replication Channels”
Section 1.9.5, “Supporters of MySQL”
Section 17.4.8, “Switching Masters During Failover”
Section 8.11.2, “Table Locking Issues”
Section B.4.2.18, “Table-Corruption Issues”
Section 2.3.4.9, “Testing The MySQL Installation”
Section 2.10.3, “Testing the Server”
Section 5.4.4, “The Binary Log”
Section 16.6, “The BLACKHOLE Storage Engine”
Section 29.5.4, “The DBUG Package”
Section 5.4.6, “The DDL Log”
Section 5.4.2, “The Error Log”
Section 5.4.3, “The General Query Log”
Section 15.19.7, “The InnoDB memcached Plugin and Replication”
Section 16.2, “The MyISAM Storage Engine”
Section 5.1, “The MySQL Server”
Section 29.1.2, “The MySQL Test Suite”
Section 22.5.10.29, “The ndbinfo operations_per_fragment Table”
Section 22.5.10.33, “The ndbinfo server_locks Table”
Section 5.4.5, “The Slow Query Log”
Section B.4.3.7, “Time Zone Problems”
Section B.4.2.6, “Too many connections”
Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”

Section 15.20.1, “Troubleshooting InnoDB I/O Problems”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 1.2, “Typographical and Syntax Conventions”
Section 29.4.2.5, “UDF Compiling and Installing”
Section 29.4.2.6, “UDF Security Precautions”
Section 2.11.12, “Upgrade Troubleshooting”
Section 22.2.8, “Upgrading and Downgrading NDB Cluster”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 2.11.10, “Upgrading MySQL on Windows”
Section 29.5.1.5, “Using a Stack Trace”
Section 7.6.1, “Using myisamchk for Crash Recovery”
Section 4.2.2.2, “Using Option Files”
Section 29.5.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section 22.2.1.2, “Using the NDB Cluster Auto-Installer”
Section 22.6.7, “Using Two Replication Channels for NDB Cluster Replication”
Section 29.5.1.3, “Using WER with PDB to create a Windows crashdump”
Section 1.4, “What Is New in MySQL 8.0”
Section 22.1.4, “What is New in NDB Cluster”
Section 2.11.3, “What the MySQL Upgrade Process Upgrades”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”
Section 6.2.13, “When Privilege Changes Take Effect”
Section B.4.3.5, “Where MySQL Stores Temporary Files”
Section 2.1.1, “Which MySQL Version and Distribution to Install”

mysqld mysqld.trace

Section 29.5.1.2, “Creating Trace Files”

mysqld-auto.cnf

Section 4.2.2, “Specifying Program Options”

mysqld-debug

Section 29.5.1.2, “Creating Trace Files”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 4.3.1, “`mysqld` — The MySQL Server”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 2.3.4.3, “Selecting a MySQL Server Type”

mysqld.exe

Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”

mysqld_multi

Section 2.5.9, “Managing MySQL Server with systemd”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.1, “Overview of MySQL Programs”
Section 5.8.3, “Running Multiple MySQL Instances on Unix”

mysqld_multi.server

Section 2.5.9, “Managing MySQL Server with systemd”

mysqld_safe

Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 29.5.1.1, “Compiling MySQL for Debugging”
Section 5.1.1, “Configuring the Server”
Section 5.4.2.2, “Default Error Log Destination Configuration”
Section 8.12.3.2, “Enabling Large Page Support”
Section B.4.2.17, “File Not Found and Similar Errors”
Section B.4.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 15.20, “InnoDB Troubleshooting”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 2.5.9, “Managing MySQL Server with systemd”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 22.5.12.3, “NDB Cluster and MySQL Security Procedures”
Section 4.2.2.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section 4.1, “Overview of MySQL Programs”
Section B.4.2.9, “Packet Too Large”
Section 26.12.13.2, “Performance Schema variables_info Table”
Section B.4.3.1, “Problems with File Permissions”
Section 13.7.8.8, “RESTART Syntax”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.8.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 2.10.5, “Starting and Stopping MySQL Automatically”
Section 2.10.2, “Starting the Server”
Section 2.10.3, “Testing the Server”
Section 5.4.2, “The Error Log”
Section B.4.3.7, “Time Zone Problems”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 4.2.2.2, “Using Option Files”

mysqldump

Section 17.4.1.3, “Backing Up a Master or Slave by Making It Read Only”
Section 17.4.1.1, “Backing Up a Slave Using mysqldump”
Chapter 7, *Backup and Recovery*
Section 7.1, “Backup and Recovery Types”
Section 7.3.3, “Backup Strategy Summary”
Section 15.5.1, “Buffer Pool”
Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 2.11.4, “Changes in MySQL 8.0”
Section 17.1.2.5, “Choosing a Method for Data Snapshots”
Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”
Section 4.2.6, “Connection Compression Control”
Section 1.9.1, “Contributors to MySQL”

Section 10.9.8, “Converting Between 3-Byte and 4-Byte Unicode Character Sets”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 2.11.14, “Copying MySQL Databases to Another Machine”
Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Creating a Data Snapshot Using mysqldump
Section 15.6.1.1, “Creating InnoDB Tables”
Section 2.3.4.7, “Customizing the PATH for MySQL Tools”
Section 14.7, “Data Dictionary Usage Differences”
Section 7.2, “Database Backup Methods”
Section 15.11.4, “Defragmenting a Table”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 7.4.5.3, “Dumping Stored Programs”
Section 7.4.5.4, “Dumping Table Definitions and Content Separately”
Section 7.3.1, “Establishing a Backup Policy”
Section 7.3, “Example Backup and Recovery Strategy”
Section 15.6.3.2, “File-Per-Table Tablespaces”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 1.7, “How to Report Bugs or Problems”
Section 13.2.5, “IMPORT TABLE Syntax”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 15.17.1, “InnoDB Backup”
Section 2.6, “Installing MySQL Using Unbreakable Linux Network (ULN)”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 13.2.7, “LOAD DATA Syntax”
Section 13.2.8, “LOAD XML Syntax”
Section 7.4.5.1, “Making a Copy of a Database”
Section 15.6.1.2, “Moving or Copying InnoDB Tables”
Section 4.5.1.1, “mysql Client Options”
Section 5.4, “MySQL Server Logs”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 7.4.5, “mysqldump Tips”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.2.6, “NDB Cluster Example with Tables and Data”
Section 22.1, “NDB Cluster Overview”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 22.5.3, “Online Backup of NDB Cluster”
Section 4.1, “Overview of MySQL Programs”
Section 26.12.9, “Performance Schema Connection Attribute Tables”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 22.6.5, “Preparing the NDB Cluster for Replication”
Section B.4.4.8, “Problems with Floating-Point Values”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 17.4.6, “Replicating Different Databases to Different Slaves”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Restoring to More Nodes Than the Original

Section 26.20, “Restrictions on Performance Schema”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 24.9, “Restrictions on Views”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.4.7, “Server Log Maintenance”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 13.4.1.3, “SET sql_log_bin Syntax”
Setting Up Replication with Existing Data
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section B.4.4.7, “Solving Problems with No Matching Rows”
Section 2.3.4.8, “Starting MySQL as a Windows Service”
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”
Section 11.4.3, “The BLOB and TEXT Types”
Section 15.19.7, “The InnoDB memcached Plugin and Replication”
Section 1.3.2, “The Main Features of MySQL”
Section 15.6.3.1, “The System Tablespace”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 13.7.4.6, “UNINSTALL PLUGIN Syntax”
Section 17.5.3, “Upgrading a Replication Setup”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 7.4, “Using mysqldump for Backups”
Section 4.2.2.2, “Using Option Files”
Section 17.4.1, “Using Replication for Backups”
Section 17.4.4, “Using Replication with Different Master and Slave Storage Engines”
Section 27.2, “Using the sys Schema”
Section 1.4, “What Is New in MySQL 8.0”
Section 12.11, “XML Functions”

mysqldump mysql

Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”

mysqldump's

Section 2.11.14, “Copying MySQL Databases to Another Machine”

mysqldumpslow

Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.1, “Overview of MySQL Programs”
Section 5.4.5, “The Slow Query Log”

mysqlhotcopy

Section 1.9.1, “Contributors to MySQL”

mysqlimport

Section 7.1, “Backup and Recovery Types”
Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 10.4, “Connection Character Sets and Collations”
Section 4.2.6, “Connection Compression Control”
Section 2.11.14, “Copying MySQL Databases to Another Machine”
Section 7.2, “Database Backup Methods”

Section 13.2.7, “LOAD DATA Syntax”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.1, “Overview of MySQL Programs”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”

MySQLInstallerConsole.exe

Section 2.3.3.5, “MySQLInstallerConsole Reference”

mysqloptimize

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

mysqlpump

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 4.2.6, “Connection Compression Control”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 14.7, “Data Dictionary Usage Differences”
Section 4.8.1, “lz4_decompress — Decompress mysqlpump LZ4-Compressed Output”
Section A.17, “MySQL 8.0 FAQ: InnoDB Data-at-Rest Encryption”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.1, “Overview of MySQL Programs”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 27.2, “Using the sys Schema”
Section 4.8.3, “zlib_decompress — Decompress mysqlpump ZLIB-Compressed Output”

mysqlrepair

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

mysqlsh

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 20.3.1, “MySQL Shell”
Section 20.4.1, “MySQL Shell”
Section 4.1, “Overview of MySQL Programs”

mysqlshow

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 10.4, “Connection Character Sets and Collations”
Section 4.2.6, “Connection Compression Control”
Section 28.7.3, “Example C API Client Programs”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.1, “Overview of MySQL Programs”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 13.7.7.14, “SHOW DATABASES Syntax”
Section 13.7.7.22, “SHOW INDEX Syntax”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 2.3.4.9, “Testing The MySQL Installation”
Section 2.10.3, “Testing the Server”

Section 2.3.6, “Windows Postinstallation Procedures”

mysqlshow db_name

Section 13.7.7.37, “SHOW TABLES Syntax”

mysqlshow db_name tbl_name

Section 13.7.7.5, “SHOW COLUMNS Syntax”

mysqlslap

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”
Section 4.2.6, “Connection Compression Control”
Section 15.15.2, “Monitoring InnoDB Mutex Waits Using Performance Schema”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 4.1, “Overview of MySQL Programs”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 8.13.2, “Using Your Own Benchmarks”

mysqltest

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 4.2.6, “Connection Compression Control”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 5.1.15, “Server Tracking of Client Session State Changes”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 29.1.2, “The MySQL Test Suite”
Section 1.4, “What Is New in MySQL 8.0”

mysqltest_embedded

Section 1.4, “What Is New in MySQL 8.0”

N

[index top]

nbdmtd

Section 22.3.3.6, “Defining NDB Cluster Data Nodes”

ndb_blob_tool

Section 22.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”

ndb_config

Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 22.4, “NDB Cluster Programs”
Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”

ndb_delete_all

Section 22.4.8, “`ndb_delete_all` — Delete All Rows from an NDB Table”

ndb_desc

Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 23.2.5, “KEY Partitioning”

MySQL Server Options for NDB Cluster
Section 22.5.12.3, “NDB Cluster and MySQL Security Procedures”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 22.4.9, “`ndb_desc` — Describe NDB Tables”
Section 13.1.20.11, “Setting NDB_TABLE Options”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”
Section 25.17, “The INFORMATION_SCHEMA PARTITIONS Table”
Section 22.5.10.5, “The ndbinfo cluster_operations Table”
Section 22.5.10.34, “The ndbinfo server_operations Table”

ndb_drop_index

Section 22.4.10, “`ndb_drop_index` — Drop Index from an NDB Table”

ndb_drop_table

Section 22.4.10, “`ndb_drop_index` — Drop Index from an NDB Table”
Section 22.4.11, “`ndb_drop_table` — Drop an NDB Table”
Section 22.2.8, “Upgrading and Downgrading NDB Cluster”
Section 22.1.4, “What is New in NDB Cluster”

ndb_error_reporter

Section 22.4.12, “`ndb_error_reporter` — NDB Error-Reporting Utility”

ndb_import

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

ndb_index_stat

Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

ndb_mgm

Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 22.5.15.1, “Adding NDB Cluster Data Nodes Online: General Issues”
Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.2.5, “Initial Startup of NDB Cluster”
Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.2.2, “Installation of NDB Cluster on Linux”
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 22.2.2.3, “Installing NDB Cluster Using .deb Files”
Chapter 22, *MySQL NDB Cluster 8.0*
MySQL Server Options for NDB Cluster
Section 22.1.1, “NDB Cluster Core Concepts”
Section 22.5.6.1, “NDB Cluster Logging Management Commands”
Section 22.4, “NDB Cluster Programs”
Section 22.5.12.1, “NDB Cluster Security and Networking Issues”
Section 22.5.8, “NDB Cluster Single User Mode”
Section 22.4.5, “`ndb_mgm` — The NDB Cluster Management Client”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.5.3, “Online Backup of NDB Cluster”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Restoring to More Nodes Than the Original
Section 22.2.7, “Safe Shutdown and Restart of NDB Cluster”
Section 22.5.10.1, “The ndbinfo arbitrator_validity_detail Table”
Section 22.5.10.25, “The ndbinfo membership Table”
Section 22.5.10.26, “The ndbinfo memoryusage Table”
Section 22.5.10.28, “The ndbinfo nodes Table”
Section 22.5.10.44, “The ndbinfo transporters Table”
Section 22.5.3.2, “Using The NDB Cluster Management Client to Create a Backup”
Section 22.1.4, “What is New in NDB Cluster”

ndb_mgm.exe

Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”

ndb_mgmd

Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 22.3.3.5, “Defining an NDB Cluster Management Server”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.2.5, “Initial Startup of NDB Cluster”
Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.2.2, “Installation of NDB Cluster on Linux”
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 22.2.2.3, “Installing NDB Cluster Using .deb Files”
MySQL Server Options for NDB Cluster
Section 2.9.7, “MySQL Source-Configuration Options”
Section 22.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 22.3.3.3, “NDB Cluster Connection Strings”
Section 22.1.1, “NDB Cluster Core Concepts”
Section 22.5.6.1, “NDB Cluster Logging Management Commands”
Section 22.1.2, “NDB Cluster Nodes, Node Groups, Replicas, and Partitions”
Section 22.4, “NDB Cluster Programs”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 22.3.1, “Quick Test Setup of NDB Cluster”
Section 22.2.7, “Safe Shutdown and Restart of NDB Cluster”
Section 22.5.1, “Summary of NDB Cluster Start Phases”
Section 22.2.1.2, “Using the NDB Cluster Auto-Installer”
Section 22.1.4, “What is New in NDB Cluster”

ndb_mgmd.exe

Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”

ndb_move_data

Section 22.4.15, “`ndb_move_data` — NDB Data Copy Utility”

ndb_perror

Section B.2, “Error Information Interfaces”
Section 22.4.16, “`ndb_perror` — Obtain NDB Error Message Information”
Section 4.8.2, “`perror` — Display MySQL Error Message Information”
Section 22.5.10.21, “The ndbinfo error_messages Table”
Section 1.4, “What Is New in MySQL 8.0”
Section 22.1.4, “What is New in NDB Cluster”

ndb_print_backup_file

Section 22.4.17, “`ndb_print_backup_file` — Print NDB Backup File Contents”
Section 22.4.19, “`ndb_print_frag_file` — Print NDB Fragment List File Contents”
Section 22.4.20, “`ndb_print_schema_file` — Print NDB Schema File Contents”
Section 22.4.21, “`ndb_print_sys_file` — Print NDB System File Contents”
Section 22.4.22, “`ndb_redo_log_reader` — Check and Print Content of Cluster Redo Log”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

ndb_print_file

Section 22.4.18, “`ndb_print_file` — Print NDB Disk Data File Contents”

ndb_print_frag_file

Section 22.4.19, “`ndb_print_frag_file` — Print NDB Fragment List File Contents”

ndb_print_schema_file

Section 22.4.17, “`ndb_print_backup_file` — Print NDB Backup File Contents”
Section 22.4.18, “`ndb_print_file` — Print NDB Disk Data File Contents”
Section 22.4.19, “`ndb_print_frag_file` — Print NDB Fragment List File Contents”
Section 22.4.20, “`ndb_print_schema_file` — Print NDB Schema File Contents”
Section 22.4.21, “`ndb_print_sys_file` — Print NDB System File Contents”
Section 22.4.22, “`ndb_redo_log_reader` — Check and Print Content of Cluster Redo Log”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

ndb_print_sys_file

Section 22.4.17, “`ndb_print_backup_file` — Print NDB Backup File Contents”
Section 22.4.18, “`ndb_print_file` — Print NDB Disk Data File Contents”
Section 22.4.19, “`ndb_print_frag_file` — Print NDB Fragment List File Contents”
Section 22.4.20, “`ndb_print_schema_file` — Print NDB Schema File Contents”
Section 22.4.21, “`ndb_print_sys_file` — Print NDB System File Contents”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

ndb_redo_log_reader

Section 22.4.22, “`ndb_redo_log_reader` — Check and Print Content of Cluster Redo Log”

ndb_restore

Section 7.1, “Backup and Recovery Types”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 22.1.1, “NDB Cluster Core Concepts”
Section 22.1, “NDB Cluster Overview”
Section 22.4, “NDB Cluster Programs”

Section 22.6.4, “NDB Cluster Replication Schema and Tables”
Section 22.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”
Section 22.5.8, “NDB Cluster Single User Mode”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.5.3, “Online Backup of NDB Cluster”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Restoring a parallel backup in parallel
Restoring a parallel backup serially
Restoring to Fewer Nodes Than the Original
Restoring to More Nodes Than the Original
Section 22.5.3.5, “Taking an NDB with Parallel Data Nodes”
Section 22.1.4, “What is New in NDB Cluster”

ndb_select_all

Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”
Section 22.5.12.3, “NDB Cluster and MySQL Security Procedures”
Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”
Section 22.4.27, “`ndb_show_tables` — Display List of NDB Tables”

ndb_select_count

Section 22.4.25, “`ndb_select_count` — Print Row Counts for NDB Tables”

ndb_setup.py

Section 22.2.1.1, “NDB Cluster Auto-Installer Requirements”
Section 22.4, “NDB Cluster Programs”
Section 22.4.26, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster”
Section 22.2.1, “The NDB Cluster Auto-Installer”
Section 22.2.1.2, “Using the NDB Cluster Auto-Installer”

ndb_show_tables

MySQL Server Options for NDB Cluster
Section 22.5.12.3, “NDB Cluster and MySQL Security Procedures”
Section 22.4, “NDB Cluster Programs”
Section 22.4.27, “`ndb_show_tables` — Display List of NDB Tables”
Section 22.5.10.4, “The ndbinfo cluster_locks Table”
Section 22.5.10.5, “The ndbinfo cluster_operations Table”
Section 22.5.10.22, “The ndbinfo locks_per_fragment Table”
Section 22.5.10.29, “The ndbinfo operations_per_fragment Table”
Section 22.5.10.33, “The ndbinfo server_locks Table”
Section 22.5.10.34, “The ndbinfo server_operations Table”

ndb_size.pl

Section 11.8, “Data Type Storage Requirements”
MySQL Server Options for NDB Cluster
Section 22.4.28, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

ndb_top

Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”

ndb_waiter

Section 22.4.30, “`ndb_waiter` — Wait for NDB Cluster to Reach a Given Status”

ndbd

Section 22.5.15.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.2.5, “Initial Startup of NDB Cluster”
Section 22.2.2, “Installation of NDB Cluster on Linux”
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 22.2.2.3, “Installing NDB Cluster Using .deb Files”
Section 22.5, “Management of NDB Cluster”
MySQL Server Options for NDB Cluster
Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 22.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 22.1.1, “NDB Cluster Core Concepts”
Section 22.3.2.1, “NDB Cluster Data Node Configuration Parameters”
Section 22.2, “NDB Cluster Installation”
Section 22.1.2, “NDB Cluster Nodes, Node Groups, Replicas, and Partitions”
Section 22.4, “NDB Cluster Programs”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.30, “`ndb_waiter` — Wait for NDB Cluster to Reach a Given Status”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 22.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”
Section 22.3.2, “Overview of NDB Cluster Configuration Parameters, Options, and Variables”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 22.3.1, “Quick Test Setup of NDB Cluster”
Section 22.3.3.2, “Recommended Starting Configuration for NDB Cluster”
Section 22.2.7, “Safe Shutdown and Restart of NDB Cluster”
Section 22.5.1, “Summary of NDB Cluster Start Phases”
Section 22.5.3.5, “Taking an NDB with Parallel Data Nodes”
Section 22.5.10.28, “The ndbinfo nodes Table”
Section 22.5.6.3, “Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client”
Section 22.3.4, “Using High-Speed Interconnects with NDB Cluster”
Section 22.2.1.2, “Using the NDB Cluster Auto-Installer”
Section 22.1.4, “What is New in NDB Cluster”

ndbd.exe

Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”

ndbd_redo_log_reader

Section 22.4.22, “`ndb_redo_log_reader` — Check and Print Content of Cluster Redo Log”

ndbinfo_select_all

Section 22.4.2, “`ndbinfo_select_all` — Select From ndbinfo Tables”

ndbmtd

Section 22.5.15.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”

Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.2.2, “Installation of NDB Cluster on Linux”
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 22.1.1, “NDB Cluster Core Concepts”
Section 22.3.2.1, “NDB Cluster Data Node Configuration Parameters”
Section 22.1.2, “NDB Cluster Nodes, Node Groups, Replicas, and Partitions”
Section 22.4, “NDB Cluster Programs”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 22.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 22.3.3.2, “Recommended Starting Configuration for NDB Cluster”
Section 22.4.23.2, “Restoring from a backup taken in parallel”
Restoring to Fewer Nodes Than the Original
Section 22.2.7, “Safe Shutdown and Restart of NDB Cluster”
Section 22.5.3.5, “Taking an NDB with Parallel Data Nodes”
Section 22.5.10.28, “The ndbinfo nodes Table”
Section 22.5.10.31, “The ndbinfo resources Table”
Section 22.2.1.2, “Using the NDB Cluster Auto-Installer”
Section 22.1.4, “What is New in NDB Cluster”

ndbmtd.exe

Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”

NET

Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 2.3.4.8, “Starting MySQL as a Windows Service”

NET START

Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”

NET START mysqld_service_name

Section 2.3.4.8, “Starting MySQL as a Windows Service”
Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”
Section 2.11.10, “Upgrading MySQL on Windows”

NET STOP

Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”

NET STOP mysqld_service_name

Section 2.3.4.8, “Starting MySQL as a Windows Service”
Section 2.11.10, “Upgrading MySQL on Windows”

NET STOP service_name
Section 22.2.7, “Safe Shutdown and Restart of NDB Cluster”

nslookup
Section 1.4, “What Is New in MySQL 8.0”

numactl
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”

O

[index top]

openssl
Section 6.4.5.5, “Audit Log Logging Configuration”
Section 6.3.3.3, “Creating RSA Keys Using openssl”
Section 6.3.3, “Creating SSL and RSA Certificates and Keys”
Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”
Section 6.3.3.2, “Creating SSL Certificates and Keys Using openssl”
Section 4.4.3, “`mysql_ssl_rsa_setup` — Create SSL/RSA Files”
Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”
Section 6.4.4.4, “Using the keyring_okv KMIP Plugin”

openssl md5 package_name
Section 2.1.3.1, “Verifying the MD5 Checksum”

openssl zlib
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.8.3, “`zlib_decompress` — Decompress mysqlpump ZLIB-Compressed Output”

otool
Section 28.7.4.1, “Building C API Client Programs”

P

[index top]

perf
Section 26.12.18.5, “The threads Table”

perror
Section B.4.2.12, “Can't create/write to file”
Section B.2, “Error Information Interfaces”
Section B.4.2.17, “File Not Found and Similar Errors”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 22.4.16, “`ndb_perror` — Obtain NDB Error Message Information”
Section 4.1, “Overview of MySQL Programs”
Section 4.8.2, “`perror` — Display MySQL Error Message Information”
Section 5.4.2.5, “Rule-Based Error Log Filtering (log_filter_dragnet)”

Section 22.5.10.21, “The ndbinfo error_messages Table”
Section 1.4, “What Is New in MySQL 8.0”
Section 22.1.4, “What is New in NDB Cluster”

pfexec

Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

PGP

Section 2.1.3.2, “Signature Checking Using GnuPG”

ping6

Section 5.1.12.5, “Obtaining an IPv6 Address from a Broker”

pkg-config

Section 28.7.4.1, “Building C API Client Programs”
Section 28.7.4.2, “Building C API Client Programs Using pkg-config”
Section 4.9, “MySQL Environment Variables”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.7.1, “`mysql_config` — Display Options for Compiling Clients”

pkgadd

Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”

pkgrm

Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”

ppm

Section 2.13, “Perl Installation Notes”

ps

Section 6.2.14, “Assigning Account Passwords”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 8.12.3.1, “How MySQL Uses Memory”
Section B.4.1, “How to Determine What Is Causing a Problem”
Section 4.9, “MySQL Environment Variables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 26.12.18.5, “The threads Table”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”

ps xa | grep mysqld

Section B.4.2.2, “Can't connect to [local] MySQL server”

R

[index top]

rename

Section 5.4.2.9, “Error Log File Flushing and Renaming”
Section 5.4.7, “Server Log Maintenance”

Section 5.4.3, “The General Query Log”

resolve_stack_dump

Section 1.4, “What Is New in MySQL 8.0”

resolveip

Section 1.4, “What Is New in MySQL 8.0”

restart

Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”

rm

Section 13.4.1.1, “PURGE BINARY LOGS Syntax”

rpm

Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.1.3.4, “Signature Checking Using RPM”

rpmbuild

Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.9.2, “Source Installation Prerequisites”

rsync

Section 17.1.2.8, “Adding Slaves to a Replication Environment”
Section 7.1, “Backup and Recovery Types”
Creating a Data Snapshot Using Raw Data Files

S

[index top]

SC

Section 2.3.4.8, “Starting MySQL as a Windows Service”

SC DELETE

Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”

SC DELETE mysqld_service_name

Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.4.8, “Starting MySQL as a Windows Service”

SC DELETE service_name

Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”

SC START

Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”

SC START mysqld_service_name

Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”
Section 2.11.10, “Upgrading MySQL on Windows”

sc start mysqld_service_name

Section 2.3.4.8, “Starting MySQL as a Windows Service”

SC STOP

Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”

SC STOP mysqld_service_name

Section 2.3.4.8, “Starting MySQL as a Windows Service”

sc stop mysqld_service_name

Section 2.3.4.8, “Starting MySQL as a Windows Service”

SC STOP service_name

Section 22.2.7, “Safe Shutdown and Restart of NDB Cluster”

scp

Section 7.1, “Backup and Recovery Types”
Creating a Data Snapshot Using Raw Data Files

sed

Section 3.3.4.7, “Pattern Matching”

SELECT

Section 22.2.6, “NDB Cluster Example with Tables and Data”

semanage port

Section 18.10, “Frequently Asked Questions”

service

Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.5.9, “Managing MySQL Server with systemd”

Service Control Manager

Section 2.3, “Installing MySQL on Microsoft Windows”
Section 2.3.4.8, “Starting MySQL as a Windows Service”

Services

Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 2.3.4.8, “Starting MySQL as a Windows Service”

sestatus

Section 18.10, “Frequently Asked Questions”

setcap
Section 8.12.5, “Resource Groups”

setenv
Section 4.2.7, “Setting Environment Variables”

setup.bat
Section 22.2.1.2, “Using the NDB Cluster Auto-Installer”

sh
Section B.4.2.17, “File Not Found and Similar Errors”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.7, “Setting Environment Variables”
Section 1.2, “Typographical and Syntax Conventions”

SHOW
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.3.1, “Quick Test Setup of NDB Cluster”

sleep
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

ssh
Section 22.5.12.1, “NDB Cluster Security and Networking Issues”

start
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”

Start>Run>cmd.exe
Section 6.3.3.2, “Creating SSL Certificates and Keys Using openssl”

status
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”

stop
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”

strings
Section 6.1.1, “Security Guidelines”

su root
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”

sudo
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 8.12.5, “Resource Groups”

systemctl
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.5.9, “Managing MySQL Server with systemd”

T

[index top]

tar

Section 17.1.2.8, “Adding Slaves to a Replication Environment”
Section 17.4.1.2, “Backing Up Raw Data from a Slave”
Section 7.1, “Backup and Recovery Types”
Creating a Data Snapshot Using Raw Data Files
Section 3.3, “Creating and Using a Database”
Section 1.7, “How to Report Bugs or Problems”
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.4, “Installing MySQL on macOS”
Section 2.7, “Installing MySQL on Solaris”
Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 2.13.1, “Installing Perl on Unix”
Section 28.7.2, “Simultaneous MySQL Server and Connector/C Installations”
Section 2.9.1, “Source Installation Methods”
Section 2.9.2, “Source Installation Prerequisites”
Section 6.4.4.4, “Using the keyring_okv KMIP Plugin”
Section 2.1.1, “Which MySQL Version and Distribution to Install”

tcpdump

Section 6.1.1, “Security Guidelines”

tcsh

Section B.4.2.17, “File Not Found and Similar Errors”
Section 2.4.1, “General Notes on Installing MySQL on macOS”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.7, “Setting Environment Variables”
Section 1.2, “Typographical and Syntax Conventions”

tee

Section 4.5.1.2, “mysql Client Commands”

telnet

Section 15.19.2, “InnoDB memcached Architecture”
Section 6.1.1, “Security Guidelines”
Section 15.19.3, “Setting Up the InnoDB memcached Plugin”

Terminal

Section 2.4, “Installing MySQL on macOS”

Text in this style

Section 1.2, “Typographical and Syntax Conventions”

top

Section B.4.1, “How to Determine What Is Causing a Problem”

Section 22.5.10.30, “The ndbinfo processes Table”

U

[index top]

ulimit

Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 8.12.3.2, “Enabling Large Page Support”
Section B.4.2.17, “File Not Found and Similar Errors”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section B.4.2.9, “Packet Too Large”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

unix_chkpwd

Section 6.4.1.5, “PAM Pluggable Authentication”

update-rc.d

Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”

useradd

Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.7, “Installing MySQL on Solaris”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

V

[index top]

vi

Section 22.2.4, “Initial Configuration of NDB Cluster”
Section 4.5.1.2, “mysql Client Commands”
Section 3.3.4.7, “Pattern Matching”

W

[index top]

watch

Section 27.4.4.25, “The statement_performance_analyzer() Procedure”

WinDbg

Section 29.5.1.3, “Using WER with PDB to create a Windows crashdump”

windbg.exe

Section 29.5.1.3, “Using WER with PDB to create a Windows crashdump”

winMd5Sum

Section 2.1.3.1, “Verifying the MD5 Checksum”

WinZip
Section 17.4.1.2, “Backing Up Raw Data from a Slave”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.9.2, “Source Installation Prerequisites”

WordPad
Section 13.2.7, “LOAD DATA Syntax”

X

[index top]

xz
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

Y

[index top]

yacc
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 9.3, “Keywords and Reserved Words”

yum
Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”
Section 22.2.2.2, “Installing NDB Cluster from RPM”
Section 2.11.7, “Upgrading MySQL with the MySQL Yum Repository”

yum install
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”

yum update
Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”

yum-config-manager
Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”

Z

[index top]

zip
Creating a Data Snapshot Using Raw Data Files
Section 1.7, “How to Report Bugs or Problems”

zlib_decompress
Section 4.8.1, “`lz4_decompress` — Decompress mysqlpump LZ4-Compressed Output”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.1, “Overview of MySQL Programs”

Section 4.8.3, “`zlib_decompress` — Decompress mysqlpump ZLIB-Compressed Output”

zsh

Section 4.2.7, “Setting Environment Variables”

zypper

Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”

Function Index

Symbols | A | B | C | D | E | F | G | H | I | J | L | M | N | O | P | Q | R | S | T | U | V | W | Y

Symbols

[index top]

%

Section 1.8.1, “MySQL Extensions to Standard SQL”

A

[index top]

ABS()

Section 29.4, “Adding New Functions to MySQL”
Section 13.7.4.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 12.6.2, “Mathematical Functions”
Section 8.9.6, “Optimizer Statistics”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

ACOS()

Section 12.6.2, “Mathematical Functions”

ADDDATE()

Section 12.7, “Date and Time Functions”

addslashes()

Section 6.1.7, “Client Programming Security Guidelines”

ADDTIME()

Section 12.7, “Date and Time Functions”

AES_DECRYPT()

Section 12.13, “Encryption and Compression Functions”
Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

AES_ENCRYPT()

Section 12.13, “Encryption and Compression Functions”
Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”

Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

ANY_VALUE()

Section 12.24, “Miscellaneous Functions”
Section 12.20.3, “MySQL Handling of GROUP BY”

ASCII()

Section 13.8.3, “HELP Syntax”
Section 12.5, “String Functions and Operators”

ASIN()

Section 12.6.2, “Mathematical Functions”

ASYMMETRIC_DECRYPT()

Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”

ASYMMETRIC_DERIVE()

Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”

ASYMMETRIC_ENCRYPT()

Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”

ASYMMETRIC_SIGN()

Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”

ASYMMETRIC_VERIFY()

Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”

ATAN()

Section 12.6.2, “Mathematical Functions”

ATAN2()

Section 12.6.2, “Mathematical Functions”

AVG(

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”

AVG()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 11.1.2, “Date and Time Type Overview”
Section 8.2.1.17, “GROUP BY Optimization”
Section 11.4.4, “The ENUM Type”
Section 1.3.2, “The Main Features of MySQL”
Section 11.4.5, “The SET Type”
Section 1.4, “What Is New in MySQL 8.0”
Section 12.21.3, “Window Function Frame Specification”

B

[index top]

BENCHMARK()
Section 13.2.11.8, “Derived Tables”
Section 12.15, “Information Functions”
Section 8.13.1, “Measuring the Speed of Expressions and Functions”
Section 13.2.11.11, “Optimizing Subqueries”

BIN()
Section 9.1.5, “Bit-Value Literals”
Section 12.5, “String Functions and Operators”

BIN_TO_UUID()
Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.24, “Miscellaneous Functions”

BIT_AND()
Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 12.12, “Bit Functions and Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”

BIT_COUNT()
Section 12.12, “Bit Functions and Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”

BIT_LENGTH()
Section 12.5, “String Functions and Operators”

BIT_OR()
Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 12.12, “Bit Functions and Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”

BIT_XOR()
Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 12.12, “Bit Functions and Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”

C

[index top]

CAN_ACCESS_COLUMN()
Section 12.23, “Internal Functions”

CAN_ACCESS_DATABASE()
Section 12.23, “Internal Functions”

CAN_ACCESS_TABLE()
Section 12.23, “Internal Functions”

CAN_ACCESS_VIEW()
Section 12.23, “Internal Functions”

CAST()

Section 12.12, “Bit Functions and Operators”
Section 9.1.5, “Bit-Value Literals”
Section 12.10, “Cast Functions and Operators”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.3.2, “Comparison Functions and Operators”
Section 11.3.6, “Conversion Between Date and Time Types”
Section 13.1.15, “CREATE INDEX Syntax”
Section 12.7, “Date and Time Functions”
Section 9.5, “Expressions”
Section 12.17.2, “Functions That Create JSON Values”
Section 12.17.3, “Functions That Search JSON Values”
Section 9.1.4, “Hexadecimal Literals”
Section 1.8.2, “MySQL Differences from Standard SQL”
Section 11.6, “The JSON Data Type”
Section 12.2, “Type Conversion in Expression Evaluation”
Section 9.4, “User-Defined Variables”
Section 1.4, “What Is New in MySQL 8.0”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”

CEIL()

Section 12.6.2, “Mathematical Functions”

CEILING()

Section 23.2.4.1, “LINEAR HASH Partitioning”
Section 12.6.2, “Mathematical Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

CHAR()

Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.5, “String Functions and Operators”

CHAR_LENGTH()

Section 12.5, “String Functions and Operators”
Section 10.10.1, “Unicode Character Sets”

CHARACTER_LENGTH()

Section 12.5, “String Functions and Operators”

CHARSET()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.15, “Information Functions”

COALESCE()

Section 12.3.2, “Comparison Functions and Operators”
Section 13.2.10.2, “JOIN Syntax”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”

COERCIBILITY()

Section 10.8.4, “Collation Coercibility in Expressions”
Section 12.15, “Information Functions”

COLLATION()

Section B.4.4.1, “Case Sensitivity in String Searches”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.15, “Information Functions”

COMPRESS()

Section 12.13, “Encryption and Compression Functions”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.8, “Server System Variables”

CONCAT()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 12.10, “Cast Functions and Operators”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 10.2.1, “Character Set Repertoire”
Section 10.8.4, “Collation Coercibility in Expressions”
Section 13.7.4.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 12.3.3, “Logical Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 5.1.11, “Server SQL Modes”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 12.5, “String Functions and Operators”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”
Section 12.2, “Type Conversion in Expression Evaluation”
Section 12.11, “XML Functions”

CONCAT_WS()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 12.5, “String Functions and Operators”

CONNECTION_ID()

Section 6.4.5.4, “Audit Log File Formats”
Section 13.1.20.7, “CHECK Constraints”
Section 13.1.20.9, “CREATE TABLE and Generated Columns”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.15, “Information Functions”
Section 13.7.8.4, “KILL Syntax”
Section 4.5.1.3, “mysql Client Logging”
Section 12.22, “Performance Schema Functions”
Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
Section 25.19, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 26.12.18.5, “The threads Table”

CONV()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.6.2, “Mathematical Functions”
Section 12.5, “String Functions and Operators”

CONVERT()

Section 12.10, “Cast Functions and Operators”
Section 10.3.8, “Character Set Introducers”
Section 10.3.6, “Character String Literal Character Set and Collation”

Section 12.3.2, “Comparison Functions and Operators”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 6.4.8.3, “Using MySQL Enterprise Data Masking and De-Identification”
Section 1.4, “What Is New in MySQL 8.0”

CONVERT_TZ()

Section 12.7, “Date and Time Functions”
Section 8.3.14, “Indexed Lookups from TIMESTAMP Columns”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 5.1.8, “Server System Variables”
Section 5.4.3, “The General Query Log”
Section 5.4.5, “The Slow Query Log”

COS()

Section 12.6.2, “Mathematical Functions”

COT()

Section 12.6.2, “Mathematical Functions”

COUNT()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 3.3.4.8, “Counting Rows”
Section 13.7.4.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 8.2.1.17, “GROUP BY Optimization”
Section 12.15, “Information Functions”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 15.6.1.6, “Limits on InnoDB Tables”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 12.24, “Miscellaneous Functions”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section B.4.4.3, “Problems with NULL Values”
Section 5.1.11, “Server SQL Modes”
Section 1.3.2, “The Main Features of MySQL”
Section 24.5.3, “Updatable and Insertable Views”
Section 8.2.1.1, “WHERE Clause Optimization”

CRC32()

Section 12.6.2, “Mathematical Functions”

CREATE_ASYMMETRIC_PRIV_KEY()

Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”
Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”
Section 4.9, “MySQL Environment Variables”

CREATE_ASYMMETRIC_PUB_KEY()

Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”

CREATE_DH_PARAMETERS()

Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”
Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”

Section 4.9, “MySQL Environment Variables”

CREATE_DIGEST()

Section 12.19.4, “MySQL Enterprise Encryption Function Descriptions”

CUME_DIST()

Section 12.21.1, “Window Function Descriptions”

CURDATE()

Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”

CURRENT_DATE

Section 11.7, “Data Type Default Values”
Section 12.7, “Date and Time Functions”

CURRENT_DATE()

Section 11.3.6, “Conversion Between Date and Time Types”
Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”

CURRENT_ROLE()

Section 12.15, “Information Functions”
Section 6.2.10, “Using Roles”

CURRENT_TIME

Section 12.7, “Date and Time Functions”

CURRENT_TIME()

Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”

CURRENT_TIMESTAMP

Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 13.1.13, “CREATE EVENT Syntax”
Section 11.7, “Data Type Default Values”
Section 12.7, “Date and Time Functions”
Section 5.1.8, “Server System Variables”

CURRENT_TIMESTAMP()

Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”

CURRENT_USER

Section 13.7.1.1, “ALTER USER Syntax”
Section 13.1.13, “CREATE EVENT Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.1.23, “CREATE VIEW Syntax”

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 12.15, “Information Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.14, “Replication and System Functions”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 13.7.7.12, “SHOW CREATE USER Syntax”
Section 6.2.4, “Specifying Account Names”
Section 24.6, “Stored Object Access Control”

CURRENT_USER()

Section 6.2.6, “Access Control, Stage 1: Connection Verification”
Section 13.7.1.1, “ALTER USER Syntax”
Section 6.4.5.4, “Audit Log File Formats”
Section 13.1.20.7, “CHECK Constraints”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 13.1.13, “CREATE EVENT Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.20.9, “CREATE TABLE and Generated Columns”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.1.23, “CREATE VIEW Syntax”
Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”
Section 13.7.1.6, “GRANT Syntax”
Implementing Proxy User Support in Authentication Plugins
Section 12.15, “Information Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 6.4.3.2, “Password Validation Options and Variables”
Section 6.2.18, “Proxy Users”
Section 17.5.1.14, “Replication and System Functions”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 13.7.1.10, “SET PASSWORD Syntax”
Section 13.7.7.12, “SHOW CREATE USER Syntax”
Section 6.2.4, “Specifying Account Names”
Section 6.2.5, “Specifying Role Names”
Section 6.2.22, “SQL-Based Account Activity Auditing”
Section 24.6, “Stored Object Access Control”
Section 29.3.2, “The Keyring Service”
Section 10.2.2, “UTF-8 for Metadata”
Writing the Server-Side Authentication Plugin

CURTIME()

Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 5.1.13, “MySQL Server Time Zone Support”

D

[index top]

DATABASE()

Section 3.3.1, “Creating and Selecting a Database”
Section 13.1.24, “DROP DATABASE Syntax”
Section 3.4, “Getting Information About Databases and Tables”

Section 12.15, “Information Functions”
Section B.4.7, “Known Issues in MySQL”
Section 10.2.2, “UTF-8 for Metadata”

DATE()

Section 12.7, “Date and Time Functions”

DATE_ADD()

Section 12.6.1, “Arithmetic Operators”
Section 12.7, “Date and Time Functions”
Section 11.3, “Date and Time Types”
Section 3.3.4.5, “Date Calculations”
Section 9.5, “Expressions”
Section 12.21.3, “Window Function Frame Specification”

DATE_FORMAT()

Section 28.7.26, “C API Prepared Statement Problems”
Section 12.10, “Cast Functions and Operators”
Section 12.7, “Date and Time Functions”
Section 10.16, “MySQL Server Locale Support”
Section 5.1.8, “Server System Variables”

DATE_SUB()

Section 12.7, “Date and Time Functions”
Section 11.3, “Date and Time Types”
Section 9.5, “Expressions”

DATEDIFF()

Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

DAY()

Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

DAYNAME()

Section 12.7, “Date and Time Functions”
Section 10.16, “MySQL Server Locale Support”
Section 5.1.8, “Server System Variables”

DAYOFMONTH()

Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

DAYOFWEEK()

Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

DAYOFYEAR()

Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

Section 23.2, “Partitioning Types”

DECODE()

Section 12.13, “Encryption and Compression Functions”
Section 1.4, “What Is New in MySQL 8.0”

DEFAULT()

Section 13.1.9.2, “ALTER TABLE and Generated Columns”
Section 11.7, “Data Type Default Values”
Section 13.2.6, “INSERT Syntax”
Section 12.24, “Miscellaneous Functions”
Section 13.2.9, “REPLACE Syntax”

DEGREES()

Section 12.6.2, “Mathematical Functions”

DENSE_RANK()

Section 12.21.1, “Window Function Descriptions”

DES_DECRYPT()

Section 12.13, “Encryption and Compression Functions”
Section 1.4, “What Is New in MySQL 8.0”

DES_ENCRYPT()

Section 12.13, “Encryption and Compression Functions”
Section 1.4, “What Is New in MySQL 8.0”

E

[index top]

ELT()

Section 12.5.3, “Character Set and Collation of Function Results”
Section B.4.7, “Known Issues in MySQL”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.5, “String Functions and Operators”

ENCODE()

Section 12.13, “Encryption and Compression Functions”
Section 1.4, “What Is New in MySQL 8.0”

ENCRYPT()

Section 1.9.1, “Contributors to MySQL”
Section 12.13, “Encryption and Compression Functions”
Section 1.4, “What Is New in MySQL 8.0”

EXP()

Section 13.1.20, “CREATE TABLE Syntax”
Section 12.6.2, “Mathematical Functions”

EXPORT_SET()

Section 12.5, “String Functions and Operators”

EXTRACT()
Section 12.10, “Cast Functions and Operators”
Section 12.7, “Date and Time Functions”
Section 9.5, “Expressions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

ExtractValue()
Section 12.11, “XML Functions”

F

[index top]

FIELD()
Section 12.5, “String Functions and Operators”

FIND_IN_SET()
Section 12.5, “String Functions and Operators”
Section 11.4.5, “The SET Type”

FIRST_VALUE()
Section 12.21.1, “Window Function Descriptions”
Section 12.21.3, “Window Function Frame Specification”

FLOOR()
Section 12.6.2, “Mathematical Functions”
Section 12.20.3, “MySQL Handling of GROUP BY”
Section 8.9.6, “Optimizer Statistics”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

FORMAT()
Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.6.2, “Mathematical Functions”
Section 12.24, “Miscellaneous Functions”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 10.16, “MySQL Server Locale Support”
Section 12.5, “String Functions and Operators”

FORMAT_BYTES()
Section 8.12.3.1, “How MySQL Uses Memory”
Section 12.22, “Performance Schema Functions”
Section 27.4.5.3, “The format_bytes() Function”

FORMAT_PICO_TIME()
Section 12.22, “Performance Schema Functions”
Section 27.4.5.6, “The format_time() Function”

FOUND_ROWS()
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.15, “Information Functions”

Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.14, “Replication and System Functions”
Section 13.2.10, “SELECT Syntax”
Section 1.4, “What Is New in MySQL 8.0”

FROM_BASE64()

Section 12.5, “String Functions and Operators”

FROM_DAYS()

Section 12.7, “Date and Time Functions”
Section 1.8.1, “MySQL Extensions to Standard SQL”

FROM_UNIXTIME()

Section 1.9.1, “Contributors to MySQL”
Section 12.7, “Date and Time Functions”
Section 8.3.14, “Indexed Lookups from TIMESTAMP Columns”
Section 17.5.1.32, “Replication and Time Zones”

G

[index top]

GeomCollection()

Section 12.16.5, “MySQL-Specific Functions That Create Geometry Values”

GeometryCollection()

Section 12.16.6, “Geometry Format Conversion Functions”
Section 12.16.5, “MySQL-Specific Functions That Create Geometry Values”

GET_DD_COLUMN_PRIVILEGES()

Section 12.23, “Internal Functions”

GET_DD_CREATE_OPTIONS()

Section 12.23, “Internal Functions”

GET_DD_INDEX_SUB_PART_LENGTH()

Section 12.23, “Internal Functions”

GET_FORMAT()

Section 12.7, “Date and Time Functions”
Section 10.16, “MySQL Server Locale Support”

GET_LOCK

Section B.3.1, “Server Error Message Reference”

GET_LOCK()

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 28.7.28, “C API Automatic Reconnection Control”
Section 13.1.13, “CREATE EVENT Syntax”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 24.4.1, “Event Scheduler Overview”

Section 8.14.2, “General Thread States”
Section 18.9.2, “Group Replication Limitations”
Section 8.11.1, “Internal Locking Methods”
Section 13.7.8.4, “KILL Syntax”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 12.14, “Locking Functions”
Locking Service Monitoring
Section 8.11.4, “Metadata Locking”
Section 28.7.7.3, “mysql_change_user()”
Section 28.7.7.60, “mysql_reset_connection()”
Section 17.5.1.14, “Replication and System Functions”
Section 29.3.1, “The Locking Service”
Section 26.12.12.3, “The metadata_locks Table”
Section 27.4.4.14, “The ps_setup_save() Procedure”

gethostbyaddr()

Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”

gethostbyname()

Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”

getrusage()

Section 22.5.10.43, “The ndbinfo threadstat Table”

gettimeofday()

Section 22.5.10.43, “The ndbinfo threadstat Table”

GREATEST()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.3.2, “Comparison Functions and Operators”
Section 11.6, “The JSON Data Type”

GROUP_CONCAT()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section B.4.7, “Known Issues in MySQL”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 5.1.8, “Server System Variables”
Section 11.6, “The JSON Data Type”
Section 1.3.2, “The Main Features of MySQL”

GROUPING()

Section 12.20.2, “GROUP BY Modifiers”
Section 12.24, “Miscellaneous Functions”
Section 12.21.2, “Window Function Concepts and Syntax”

GTID_INTERSECTION_WITH_UUID

Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

GTID_IS_DISJOINT

Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

GTID_IS_DISJOINT_UNION
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

GTID_IS_EQUAL
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

GTID_SUBSET
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

GTID_SUBSET()
Section 12.18, “Functions Used with Global Transaction Identifiers (GTIDs)”
Section 17.1.3.1, “GTID Format and Storage”
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

GTID_SUBTRACT
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

GTID_SUBTRACT()
Section 12.18, “Functions Used with Global Transaction Identifiers (GTIDs)”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Format and Storage”
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

GTID_SUBTRACT_UUID
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

GTID_UNION
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

H

[index top]

HEX()
Section 9.1.5, “Bit-Value Literals”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”
Section 9.1.4, “Hexadecimal Literals”
Section 12.6.2, “Mathematical Functions”
Section 12.24, “Miscellaneous Functions”
Section 12.5, “String Functions and Operators”

HOUR()
Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

I

[index top]

ICU_VERSION()

Section 12.15, “Information Functions”

IF()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.4, “Control Flow Functions”
Section 13.6.5.2, “IF Syntax”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section B.4.7, “Known Issues in MySQL”
Section 1.8.1, “MySQL Extensions to Standard SQL”

IFNULL()

Section 12.4, “Control Flow Functions”
Section B.4.4.3, “Problems with NULL Values”

INET6_ATON()

Section 12.12, “Bit Functions and Operators”
Section 5.1.12, “IPv6 Support”
Section 12.24, “Miscellaneous Functions”

INET6_NTOA()

Section 5.1.12, “IPv6 Support”
Section 12.24, “Miscellaneous Functions”

INET_ATON()

Section 5.1.12, “IPv6 Support”
Section 12.24, “Miscellaneous Functions”

INET_NTOA()

Section 5.1.12, “IPv6 Support”
Section 12.24, “Miscellaneous Functions”

INSERT()

Section 12.5, “String Functions and Operators”

INSTR()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

INTERNAL_AUTO_INCREMENT()

Section 12.23, “Internal Functions”

INTERNAL_AVG_ROW_LENGTH()

Section 12.23, “Internal Functions”

INTERNAL_CHECK_TIME()

Section 12.23, “Internal Functions”

INTERNAL_CHECKSUM()

Section 12.23, “Internal Functions”

INTERNAL_DATA_FREE()

Section 12.23, “Internal Functions”

INTERNAL_DATA_LENGTH()

Section 12.23, “Internal Functions”

INTERNAL_DD_CHAR_LENGTH()

Section 12.23, “Internal Functions”

INTERNAL_GET_COMMENT_OR_ERROR()

Section 12.23, “Internal Functions”

INTERNAL_GET_VIEW_WARNING_OR_ERROR()

Section 12.23, “Internal Functions”

INTERNAL_INDEX_COLUMN_CARDINALITY()

Section 12.23, “Internal Functions”

INTERNAL_INDEX_LENGTH()

Section 12.23, “Internal Functions”

INTERNAL_KEYS_DISABLED()

Section 12.23, “Internal Functions”

INTERNAL_MAX_DATA_LENGTH()

Section 12.23, “Internal Functions”

INTERNAL_TABLE_ROWS()

Section 12.23, “Internal Functions”

INTERNAL_UPDATE_TIME()

Section 12.23, “Internal Functions”

INTERVAL()

Section 12.3.2, “Comparison Functions and Operators”

IS_FREE_LOCK()

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.14, “Locking Functions”
Section 17.5.1.14, “Replication and System Functions”

IS_IPV4()

Section 12.24, “Miscellaneous Functions”

IS_IPV4_COMPAT()

Section 12.24, “Miscellaneous Functions”

IS_IPV4_MAPPED()

Section 12.24, “Miscellaneous Functions”

IS_IPV6()
Section 12.24, “Miscellaneous Functions”

IS_USED_LOCK()
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.14, “Locking Functions”
Section 17.5.1.14, “Replication and System Functions”

IS_UUID()
Section 12.24, “Miscellaneous Functions”

IS_VISIBLE_DD_OBJECT()
Section 12.23, “Internal Functions”

ISNULL()
Section 12.3.2, “Comparison Functions and Operators”

J

[index top]

JSON_ARRAY()
Section 12.17.2, “Functions That Create JSON Values”
Section 12.17.3, “Functions That Search JSON Values”
Section 11.6, “The JSON Data Type”

JSON_ARRAY_APPEND()
Section 12.17.4, “Functions That Modify JSON Values”
Section 1.4, “What Is New in MySQL 8.0”

JSON_ARRAY_INSERT()
Section 12.17.4, “Functions That Modify JSON Values”

JSON_ARRAYAGG()
Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 12.17.2, “Functions That Create JSON Values”
Section 12.17.1, “JSON Function Reference”
Section 1.4, “What Is New in MySQL 8.0”

JSON_CONTAINS()
Section 13.1.15, “CREATE INDEX Syntax”
Section 12.17.3, “Functions That Search JSON Values”
Section 1.4, “What Is New in MySQL 8.0”

JSON_CONTAINS_PATH()
Section 12.17.3, “Functions That Search JSON Values”
Section 11.6, “The JSON Data Type”

JSON_DEPTH()
Section 12.17.5, “Functions That Return JSON Value Attributes”

JSON_EXTRACT()

Section 12.17.3, “Functions That Search JSON Values”
Section 12.17.7, “JSON Schema Validation Functions”
Section 13.1.20.10, “Secondary Indexes and Generated Columns”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

JSON_INSERT()

Section 12.17.4, “Functions That Modify JSON Values”
Section 11.6, “The JSON Data Type”

JSON_KEYS()

Section 12.17.3, “Functions That Search JSON Values”
Section 6.2.3, “Grant Tables”

JSON_LENGTH()

Section 12.17.5, “Functions That Return JSON Value Attributes”

JSON_MERGE()

Section 12.17.4, “Functions That Modify JSON Values”
Section 1.4, “What Is New in MySQL 8.0”

JSON_MERGE_PATCH()

Section 12.17.4, “Functions That Modify JSON Values”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

JSON_MERGE_PRESERVE()

Section 12.17.4, “Functions That Modify JSON Values”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

JSON_OBJECT()

Section 12.17.2, “Functions That Create JSON Values”
Section 12.17.3, “Functions That Search JSON Values”
Section 11.6, “The JSON Data Type”

JSON_OBJECTAGG()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 12.17.2, “Functions That Create JSON Values”
Section 12.17.1, “JSON Function Reference”
Section 1.4, “What Is New in MySQL 8.0”

JSON_OVERLAPS()

Section 13.1.15, “CREATE INDEX Syntax”
Section 12.17.3, “Functions That Search JSON Values”
Section 1.4, “What Is New in MySQL 8.0”

JSON_PRETTY()

Section 12.17.1, “JSON Function Reference”

Section 12.17.7, “JSON Schema Validation Functions”
Section 12.17.8, “JSON Utility Functions”
Section 1.4, “What Is New in MySQL 8.0”

JSON_QUOTE()

Section 12.17.2, “Functions That Create JSON Values”
Section 12.17.8, “JSON Utility Functions”

JSON_REMOVE()

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 12.17.4, “Functions That Modify JSON Values”
Section 12.17.8, “JSON Utility Functions”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

JSON_REPLACE()

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 12.17.4, “Functions That Modify JSON Values”
Section 12.17.8, “JSON Utility Functions”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

JSON_SCHEMA_VALID()

Section 12.17.7, “JSON Schema Validation Functions”
Section 1.4, “What Is New in MySQL 8.0”

JSON_SCHEMA_VALIDATION_REPORT()

Section 12.17.7, “JSON Schema Validation Functions”
Section 1.4, “What Is New in MySQL 8.0”

JSON_SEARCH()

Section 12.17.3, “Functions That Search JSON Values”
Section 11.6, “The JSON Data Type”

JSON_SET()

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 12.17.4, “Functions That Modify JSON Values”
Section 12.17.8, “JSON Utility Functions”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

JSON_STORAGE_FREE()

Section 12.17.1, “JSON Function Reference”
Section 12.17.8, “JSON Utility Functions”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

JSON_STORAGE_SIZE()

Section 12.17.1, “JSON Function Reference”
Section 12.17.8, “JSON Utility Functions”
Section 11.6, “The JSON Data Type”

Section 1.4, “What Is New in MySQL 8.0”

JSON_TABLE()

Section 13.2.11.8, “Derived Tables”
Section 12.17.6, “JSON Table Functions”
Section 13.2.11.9, “Lateral Derived Tables”
Section 1.4, “What Is New in MySQL 8.0”

JSON_TYPE()

Section 12.17.5, “Functions That Return JSON Value Attributes”
Section 12.17.3, “Functions That Search JSON Values”
Section 11.6, “The JSON Data Type”

JSON_UNQUOTE(

Section 12.17.3, “Functions That Search JSON Values”

JSON_UNQUOTE()

Section 13.1.15, “CREATE INDEX Syntax”
Section 12.17.4, “Functions That Modify JSON Values”
Section 8.3.11, “Optimizer Use of Generated Column Indexes”
Section 13.1.20.10, “Secondary Indexes and Generated Columns”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

JSON_VALID()

Section 12.17.5, “Functions That Return JSON Value Attributes”

L

[index top]

LAG()

Section 1.4, “What Is New in MySQL 8.0”
Section 12.21.1, “Window Function Descriptions”

LAST_DAY()

Section 12.7, “Date and Time Functions”

LAST_INSERT_ID()

Section 28.7.28, “C API Automatic Reconnection Control”
Section 12.3.2, “Comparison Functions and Operators”
Section 13.1.20, “CREATE TABLE Syntax”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 28.7.29.3, “How to Get the Unique ID for the Last Inserted Row”
Section 12.15, “Information Functions”
Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 13.2.6, “INSERT Syntax”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 28.7.7.38, “mysql_insert_id()”
Section 28.7.7.60, “mysql_reset_connection()”

Section 28.7.11.16, “mysql_stmt_insert_id()”
Section 17.5.1.1, “Replication and AUTO_INCREMENT”
Section 5.1.8, “Server System Variables”
Section 24.2.4, “Stored Procedures, Functions, Triggers, and LAST_INSERT_ID()”
Section 17.5.4, “Troubleshooting Replication”
Section 24.5.3, “Updatable and Insertable Views”
Section 3.6.9, “Using AUTO_INCREMENT”

LAST_VALUE()

Section 12.21.1, “Window Function Descriptions”
Section 12.21.3, “Window Function Frame Specification”

LCASE()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

LEAD()

Section 12.21.1, “Window Function Descriptions”

LEAST()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.3.2, “Comparison Functions and Operators”
Section 11.6, “The JSON Data Type”

LEFT()

Section 12.5, “String Functions and Operators”

LENGTH()

Section 11.8, “Data Type Storage Requirements”
Section 12.5, “String Functions and Operators”
Section 11.5.3, “Supported Spatial Data Formats”

LineString()

Section 12.16.5, “MySQL-Specific Functions That Create Geometry Values”

LN()

Section 12.6.2, “Mathematical Functions”

LOAD_FILE()

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 13.2.8, “LOAD XML Syntax”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.14, “Replication and System Functions”
Section 5.1.8, “Server System Variables”
Section 12.5, “String Functions and Operators”

LOCALTIME

Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”

Section 12.7, “Date and Time Functions”

LOCALTIME()

Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”

LOCALTIMESTAMP

Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.7, “Date and Time Functions”

LOCALTIMESTAMP()

Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”

LOCATE()

Section 12.5, “String Functions and Operators”

LOG()

Section 23.2.4.1, “LINEAR HASH Partitioning”
Section 12.6.2, “Mathematical Functions”

LOG10()

Section 12.6.2, “Mathematical Functions”

LOG2()

Section 12.6.2, “Mathematical Functions”

LOWER()

Section 12.10, “Cast Functions and Operators”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”
Section 10.10.1, “Unicode Character Sets”
Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”

LPAD()

Section 12.12, “Bit Functions and Operators”
Section 11.2.5, “Numeric Type Attributes”
Section 11.1.1, “Numeric Type Overview”
Section 12.5, “String Functions and Operators”
Section 1.4, “What Is New in MySQL 8.0”

LTRIM()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

M

[index top]

MAKE_SET()
Section 12.5, “String Functions and Operators”

MAKEDATE()
Section 12.7, “Date and Time Functions”

MAKETIME()
Section 12.7, “Date and Time Functions”

MASTER_POS_WAIT()
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.2.3.1, “Commands for Operations on a Single Channel”
Section 17.2.3.2, “Compatibility with Previous Replication Statements”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.24, “Miscellaneous Functions”
Section A.14, “MySQL 8.0 FAQ: Replication”

MATCH
Section 9.5, “Expressions”

MATCH ()
Section 12.9, “Full-Text Search Functions”

MATCH()
Section 12.9.2, “Boolean Full-Text Searches”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.9.5, “Full-Text Restrictions”
Section 12.9, “Full-Text Search Functions”
Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
MySQL Glossary
Section 12.9.1, “Natural Language Full-Text Searches”

MAX(
Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”

MAX()
Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 8.3.13, “Descending Indexes”
Section 8.2.1.17, “GROUP BY Optimization”
Section 8.3.1, “How MySQL Uses Indexes”
Section B.4.7, “Known Issues in MySQL”
Section 11.1.1, “Numeric Type Overview”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 13.2.11.11, “Optimizing Subqueries”
Section 5.1.11, “Server SQL Modes”
Section 11.6, “The JSON Data Type”
Section 1.3.2, “The Main Features of MySQL”
Section 26.12.15.1, “The tp_thread_group_state Table”
Section 11.3.7, “Two-Digit Years in Dates”
Section 24.5.3, “Updatable and Insertable Views”

Section 8.3.10, “Use of Index Extensions”
Section 3.6.9, “Using AUTO_INCREMENT”
Section 8.2.1.21, “Window Function Optimization”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”

MBRContains()

Section 12.16.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”
Section 11.5.11, “Using Spatial Indexes”

MBRCoveredBy()

Section 12.16.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBRCovers()

Section 12.16.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBRDisjoint()

Section 12.16.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBREquals()

Section 12.16.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBRIntersects()

Section 12.16.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBROverlaps()

Section 12.16.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBRTouches()

Section 12.16.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBRWithin()

Section 12.16.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”
Section 11.5.11, “Using Spatial Indexes”

MD5()

Section 12.13, “Encryption and Compression Functions”
Section 23.2.5, “KEY Partitioning”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 9.2, “Schema Object Names”

MICROSECOND()

Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

MID()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

MIN(

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”

MIN()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 28.7.26, “C API Prepared Statement Problems”
Section 8.3.13, “Descending Indexes”
Section 8.2.1.17, “GROUP BY Optimization”
Section 8.3.1, “How MySQL Uses Indexes”
Section B.4.7, “Known Issues in MySQL”
Section 11.1.1, “Numeric Type Overview”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 13.2.11.11, “Optimizing Subqueries”
Section B.4.4.3, “Problems with NULL Values”
Section 11.6, “The JSON Data Type”
Section 1.3.2, “The Main Features of MySQL”
Section 11.3.7, “Two-Digit Years in Dates”
Section 24.5.3, “Updatable and Insertable Views”
Section 8.3.10, “Use of Index Extensions”
Section 8.2.1.1, “WHERE Clause Optimization”
Section 8.2.1.21, “Window Function Optimization”

MINUTE()

Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

MOD()

Section 12.6.1, “Arithmetic Operators”
Section 3.3.4.5, “Date Calculations”
Section 12.6.2, “Mathematical Functions”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 23.6.3, “Partitioning Limitations Relating to Functions”
Section 5.1.11, “Server SQL Modes”

MONTH()

Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”
Section 23.6.3, “Partitioning Limitations Relating to Functions”
Section 23.2, “Partitioning Types”

MONTHNAME()

Section 12.7, “Date and Time Functions”
Section 10.16, “MySQL Server Locale Support”
Section 5.1.8, “Server System Variables”

MultiLineString()

Section 12.16.5, “MySQL-Specific Functions That Create Geometry Values”

MultiPoint()

Section 12.16.5, “MySQL-Specific Functions That Create Geometry Values”

MultiPolygon()

Section 12.16.5, “MySQL-Specific Functions That Create Geometry Values”

my_open()
Section 5.1.10, “Server Status Variables”

N

[index top]

NAME_CONST()
Section 12.24, “Miscellaneous Functions”
Section 24.7, “Stored Program Binary Logging”

NOW()
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 13.1.20.7, “CHECK Constraints”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.20.9, “CREATE TABLE and Generated Columns”
Section 11.7, “Data Type Default Values”
Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 11.3.5, “Fractional Seconds in Time Values”
Section A.1, “MySQL 8.0 FAQ: General”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 17.5.1.14, “Replication and System Functions”
Section 17.5.1.32, “Replication and Time Zones”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 27.4.3.21, “The metrics View”
Section 27.4.4.25, “The statement_performance_analyzer() Procedure”
Section 11.3.3, “The YEAR Type”

NTH_VALUE()
Section 12.21.1, “Window Function Descriptions”
Section 12.21.3, “Window Function Frame Specification”

NTILE()
Section 1.4, “What Is New in MySQL 8.0”
Section 12.21.1, “Window Function Descriptions”

NULLIF()
Section 12.4, “Control Flow Functions”

O

[index top]

OCT()
Section 12.5, “String Functions and Operators”

OCTET_LENGTH()
Section 12.5, “String Functions and Operators”

ORD()
Section 12.5, “String Functions and Operators”

P

[index top]

PASSWORD()
Section 12.13, “Encryption and Compression Functions”
Section 23.2.5, “KEY Partitioning”

PERCENT_RANK()
Section 12.21.1, “Window Function Descriptions”

PERIOD_ADD()
Section 12.7, “Date and Time Functions”
Section 1.8.1, “MySQL Extensions to Standard SQL”

PERIOD_DIFF()
Section 12.7, “Date and Time Functions”
Section 1.8.1, “MySQL Extensions to Standard SQL”

PI()
Section 9.2.5, “Function Name Parsing and Resolution”
Section 12.6.2, “Mathematical Functions”

Point()
Section 12.16.5, “MySQL-Specific Functions That Create Geometry Values”
Section 11.5.3, “Supported Spatial Data Formats”

Polygon()
Section 12.16.5, “MySQL-Specific Functions That Create Geometry Values”

POSITION()
Section 12.5, “String Functions and Operators”

POW()
Section 8.2.1.20, “Function Call Optimization”
Section 23.2.4, “HASH Partitioning”
Section 12.6.2, “Mathematical Functions”

POWER()
Section 23.2.4.1, “LINEAR HASH Partitioning”
Section 12.6.2, “Mathematical Functions”

PS_CURRENT_THREAD_ID()
Section 12.22, “Performance Schema Functions”
Section 27.4.5.15, “The ps_thread_id() Function”

PS_THREAD_ID()
Section 12.22, “Performance Schema Functions”

Section 27.4.5.15, “The ps_thread_id() Function”

pthread_mutex()

Section 1.9.1, “Contributors to MySQL”

Q

[index top]

QUARTER()

Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

QUOTE()

Section 28.7.7.55, “mysql_real_escape_string()”
Section 28.7.7.56, “mysql_real_escape_string_quote()”
Section 12.5, “String Functions and Operators”
Section 9.1.1, “String Literals”

R

[index top]

RADIANS()

Section 12.6.2, “Mathematical Functions”

RAND()

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.2.1.20, “Function Call Optimization”
Section 12.6.2, “Mathematical Functions”
Section 28.7.7.69, “mysql_session_track_get_first()”
Section 17.5.1.14, “Replication and System Functions”
Section 5.1.8, “Server System Variables”

RANDOM_BYTES()

Section 12.13, “Encryption and Compression Functions”

RANK()

Section 1.4, “What Is New in MySQL 8.0”
Section 12.21.1, “Window Function Descriptions”

REGEXP_INSTR()

Section 12.5.2, “Regular Expressions”
Section 1.4, “What Is New in MySQL 8.0”

REGEXP_LIKE()

Section 3.3.4.7, “Pattern Matching”
Section 12.5.2, “Regular Expressions”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

REGEXP_REPLACE()

Section 12.5.2, “Regular Expressions”
Section 1.4, “What Is New in MySQL 8.0”

REGEXP_SUBSTR()

Section 12.5.2, “Regular Expressions”
Section 1.4, “What Is New in MySQL 8.0”

RELEASE_ALL_LOCKS()

Section 12.14, “Locking Functions”

RELEASE_LOCK()

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 13.2.3, “DO Syntax”
Section 8.11.1, “Internal Locking Methods”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 12.14, “Locking Functions”
Section 17.5.1.14, “Replication and System Functions”

REPEAT()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

REPLACE()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

REVERSE()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

RIGHT()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

ROLES_GRAPHML()

Section 12.15, “Information Functions”
Section 6.2.2, “Privileges Provided by MySQL”

ROUND()

Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 12.6.2, “Mathematical Functions”
Section 12.25, “Precision Math”
Section 12.25.5, “Precision Math Examples”
Section 12.25.4, “Rounding Behavior”

ROW_COUNT()

Section 13.2.1, “CALL Syntax”
Section 13.2.2, “DELETE Syntax”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”

Section 12.15, “Information Functions”
Section 13.2.6, “INSERT Syntax”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 28.7.7.1, “mysql_affected_rows()”
Section 17.5.1.14, “Replication and System Functions”
Section 13.6.7.7, “The MySQL Diagnostics Area”

ROW_NUMBER()

Section 12.21.2, “Window Function Concepts and Syntax”
Section 12.21.1, “Window Function Descriptions”

RPAD()

Section 12.12, “Bit Functions and Operators”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

RTRIM()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

S

[index top]

SCHEMA()

Section 12.15, “Information Functions”

SEC_TO_TIME()

Section 12.7, “Date and Time Functions”

SECOND()

Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

SESSION_USER()

Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.15, “Information Functions”
Section 10.2.2, “UTF-8 for Metadata”

setrlimit()

Section 5.1.8, “Server System Variables”

SHA()

Section 12.13, “Encryption and Compression Functions”

SHA1()

Section 12.13, “Encryption and Compression Functions”

SHA2()

Section 12.13, “Encryption and Compression Functions”
Section 6.1.1, “Security Guidelines”
Section 1.4, “What Is New in MySQL 8.0”

SIGN()

Section 12.6.2, “Mathematical Functions”

SIN()

Section 12.6.2, “Mathematical Functions”
Section 29.4.2.3, “UDF Argument Processing”

SLEEP()

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.14.2, “General Thread States”
Section 12.24, “Miscellaneous Functions”
Section 26.12.15.2, “The tp_thread_group_stats Table”

SOUNDEX()

Section 29.4, “Adding New Functions to MySQL”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

SPACE()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

SQRT()

Section 12.6.2, “Mathematical Functions”

ST_Area()

Section 12.16.7, “Geometry Property Functions”
Section 12.16.7.4, “Polygon and MultiPolygon Property Functions”

ST_AsBinary()

Section 11.5.8, “Fetching Spatial Data”
Section 12.16.6, “Geometry Format Conversion Functions”

ST_AsGeoJSON()

Section 12.16.11, “Spatial GeoJSON Functions”
Section 11.6, “The JSON Data Type”

ST_AsText()

Section 11.5.8, “Fetching Spatial Data”
Section 12.16.6, “Geometry Format Conversion Functions”

ST_AsWKB()

Section 12.16.6, “Geometry Format Conversion Functions”

ST_AsWKT()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”
Section 12.16.6, “Geometry Format Conversion Functions”

ST_Buffer()

Section 12.16.8, “Spatial Operator Functions”

ST_Buffer_Strategy()

Section 5.1.8, “Server System Variables”
Section 12.16.8, “Spatial Operator Functions”

ST_Centroid()

Section 12.16.7.4, “Polygon and MultiPolygon Property Functions”

ST_Contains()

Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_ConvexHull()

Section 12.16.8, “Spatial Operator Functions”

ST_Crosses()

Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_Difference()

Section 12.16.8, “Spatial Operator Functions”

ST_Dimension()

Section 12.16.7.1, “General Geometry Property Functions”

ST_Disjoint()

Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_Distance()

Section 12.16.12, “Spatial Convenience Functions”
Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”
Section 25.29, “The INFORMATION_SCHEMA ST_UNITS_OF_MEASURE Table”

ST_Distance_Sphere()

Section 12.16.12, “Spatial Convenience Functions”
Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_EndPoint()

Section 12.16.7.3, “LineString and MultiLineString Property Functions”
Section 12.16.8, “Spatial Operator Functions”

ST_Envelope()

Section 12.16.7.1, “General Geometry Property Functions”
Section 12.16.8, “Spatial Operator Functions”

ST_Equals()

Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_ExteriorRing()

Section 12.16.7.4, “Polygon and MultiPolygon Property Functions”
Section 12.16.8, “Spatial Operator Functions”

ST_GeoHash()

Section 12.16.10, “Spatial Geohash Functions”

ST_GeomCollFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_GeomCollFromTxt()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_GeomCollFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_GeometryCollectionFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_GeometryCollectionFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_GeometryFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_GeometryFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_GeometryN()

Section 12.16.7.5, “GeometryCollection Property Functions”
Section 12.16.8, “Spatial Operator Functions”

ST_GeometryType()

Section 12.16.7.1, “General Geometry Property Functions”

ST_GeomFromGeoJSON()

Section 12.16.11, “Spatial GeoJSON Functions”
Section 11.6, “The JSON Data Type”

ST_GeomFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”
Section 12.16.6, “Geometry Format Conversion Functions”
Section 12.16.5, “MySQL-Specific Functions That Create Geometry Values”
Section 11.5.7, “Populating Spatial Columns”
Section 11.5.3, “Supported Spatial Data Formats”

ST_GeomFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_InteriorRingN()

Section 12.16.7.4, “Polygon and MultiPolygon Property Functions”
Section 12.16.8, “Spatial Operator Functions”

ST_Intersection()

Section 12.16.8, “Spatial Operator Functions”

ST_Intersects()

Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_IsClosed()

Section 12.16.7.3, “LineString and MultiLineString Property Functions”

ST_IsEmpty()

Section 12.16.7.1, “General Geometry Property Functions”

ST_IsSimple()

Section 12.16.7.1, “General Geometry Property Functions”

ST_IsValid()

Section 11.5.4, “Geometry Well-Formedness and Validity”
Section 12.16.12, “Spatial Convenience Functions”

ST_LatFromGeoHash()

Section 12.16.10, “Spatial Geohash Functions”

ST_Latitude()

Section 12.16.7.2, “Point Property Functions”

ST_Length()

Section 12.16.7.3, “LineString and MultiLineString Property Functions”
Section 12.5, “String Functions and Operators”
Section 1.4, “What Is New in MySQL 8.0”

ST_LineFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_LineFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_LineStringFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_LineStringFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_LongFromGeoHash()

Section 12.16.10, “Spatial Geohash Functions”

ST_Longitude()

Section 12.16.7.2, “Point Property Functions”

ST_MakeEnvelope()

Section 12.16.12, “Spatial Convenience Functions”

ST_MLineFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_MLineFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_MPointFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”
Section 11.5.3, “Supported Spatial Data Formats”

ST_MPointFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_MPolyFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_MPolyFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_MultiLineStringFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_MultiLineStringFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_MultiPointFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_MultiPointFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_MultiPolygonFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_MultiPolygonFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_NumGeometries()

Section 12.16.7.5, “GeometryCollection Property Functions”

ST_NumInteriorRing()

Section 12.16.7.4, “Polygon and MultiPolygon Property Functions”

ST_NumInteriorRings()

Section 12.16.7.4, “Polygon and MultiPolygon Property Functions”

ST_NuminteriorRings()

Section 12.16.7.4, “Polygon and MultiPolygon Property Functions”

ST_NumPoints()

Section 12.16.7.3, “LineString and MultiLineString Property Functions”

ST_Overlaps()

Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_PointFromGeoHash()

Section 12.16.10, “Spatial Geohash Functions”

ST_PointFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_PointFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_PointN()

Section 12.16.7.3, “LineString and MultiLineString Property Functions”
Section 12.16.8, “Spatial Operator Functions”

ST_PolyFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_PolyFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_PolygonFromText()

Section 12.16.3, “Functions That Create Geometry Values from WKT Values”

ST_PolygonFromWKB()

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”

ST_Simplify()

Section 12.16.12, “Spatial Convenience Functions”

ST_SRID()

Section 12.16.7.1, “General Geometry Property Functions”
Section 12.16.8, “Spatial Operator Functions”

ST_StartPoint()

Section 12.16.7.3, “LineString and MultiLineString Property Functions”
Section 12.16.8, “Spatial Operator Functions”

ST_SwapXY()

Section 12.16.6, “Geometry Format Conversion Functions”

ST_SymDifference()

Section 12.16.8, “Spatial Operator Functions”

ST_Touches()

Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_Transform()

Section 12.16.7.1, “General Geometry Property Functions”
Section 12.16.8, “Spatial Operator Functions”

ST_Union()

Section 12.16.8, “Spatial Operator Functions”

ST_Validate()

Section 12.16.12, “Spatial Convenience Functions”

ST_Within()

Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_X()

Section 12.16.7.2, “Point Property Functions”
Section 11.5.3, “Supported Spatial Data Formats”

ST_Y()

Section 12.16.7.2, “Point Property Functions”

STATEMENT_DIGEST()

Section 12.13, “Encryption and Compression Functions”
Section 26.10, “Performance Schema Statement Digests and Sampling”

STATEMENT_DIGEST_TEXT()

Section 12.13, “Encryption and Compression Functions”
Section 26.10, “Performance Schema Statement Digests and Sampling”

STD()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 1.3.2, “The Main Features of MySQL”

STDDEV()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”

STDDEV_POP()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 8.2.1.21, “Window Function Optimization”

STDDEV_SAMP()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 8.2.1.21, “Window Function Optimization”

STR_TO_DATE()

Section 12.7, “Date and Time Functions”
Section 10.16, “MySQL Server Locale Support”

STRCMP()

Section B.4.4.2, “Problems Using DATE Columns”
Section 12.5.1, “String Comparison Functions and Operators”

SUBDATE()

Section 12.7, “Date and Time Functions”

SUBSTR()

Section 12.12, “Bit Functions and Operators”
Section 12.5, “String Functions and Operators”

SUBSTRING()

Section 12.5.3, “Character Set and Collation of Function Results”

Section 13.1.15, “CREATE INDEX Syntax”
Section 12.5, “String Functions and Operators”

SUBSTRING_INDEX()

Section 6.2.22, “SQL-Based Account Activity Auditing”
Section 12.5, “String Functions and Operators”

SUBTIME()

Section 12.7, “Date and Time Functions”

SUM(

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”

SUM()

Section 29.4.2, “Adding a New User-Defined Function”
Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 11.1.2, “Date and Time Type Overview”
Section 8.2.1.17, “GROUP BY Optimization”
Section 12.24, “Miscellaneous Functions”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section B.4.4.3, “Problems with NULL Values”
Section 11.4.4, “The ENUM Type”
Section 1.3.2, “The Main Features of MySQL”
Section 11.4.5, “The SET Type”
Section 24.5.3, “Updatable and Insertable Views”
Section 1.4, “What Is New in MySQL 8.0”
Section 12.21.2, “Window Function Concepts and Syntax”
Section 12.21.3, “Window Function Frame Specification”

SYSDATE()

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 17.5.1.14, “Replication and System Functions”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

SYSTEM_USER()

Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.15, “Information Functions”
Section 10.2.2, “UTF-8 for Metadata”

T

[index top]

TAN()

Section 12.6.2, “Mathematical Functions”

TIME()

Section 12.7, “Date and Time Functions”

TIME_FORMAT()
Section 12.10, “Cast Functions and Operators”
Section 12.7, “Date and Time Functions”

TIME_TO_SEC()
Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

TIMEDIFF()
Section 12.7, “Date and Time Functions”

TIMESTAMP()
Section 12.7, “Date and Time Functions”

TIMESTAMPADD()
Section 12.7, “Date and Time Functions”

TIMESTAMPDIFF()
Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”

TO_BASE64()
Section 12.5, “String Functions and Operators”

TO_DAYS()
Section 12.7, “Date and Time Functions”
Section 23.2.4, “HASH Partitioning”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 23.4, “Partition Pruning”
Section 23.6.3, “Partitioning Limitations Relating to Functions”
Section 23.2, “Partitioning Types”

TO_SECONDS()
Section 12.7, “Date and Time Functions”
Section 23.4, “Partition Pruning”
Section 23.6.3, “Partitioning Limitations Relating to Functions”
Section 23.2, “Partitioning Types”

TRIM()
Section 12.5.3, “Character Set and Collation of Function Results”
Section 10.7, “Column Character Set Conversion”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.5, “String Functions and Operators”

TRUNCATE()
Section 12.6.2, “Mathematical Functions”

U

[index top]

UCASE()

Section 12.5.3, “Character Set and Collation of Function Results”
Section 12.5, “String Functions and Operators”

UNCOMPRESS()

Section 12.13, “Encryption and Compression Functions”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.8, “Server System Variables”

UNCOMPRESSED_LENGTH()

Section 12.13, “Encryption and Compression Functions”

UNHEX()

Section 12.13, “Encryption and Compression Functions”
Section 12.5, “String Functions and Operators”

UNIX_TIMESTAMP()

Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.3.14, “Indexed Lookups from TIMESTAMP Columns”
Section 23.6.3, “Partitioning Limitations Relating to Functions”
Section 23.2.1, “RANGE Partitioning”
Section 5.1.8, “Server System Variables”
Section 27.4.3.21, “The metrics View”
Section B.4.3.7, “Time Zone Problems”

UpdateXML()

Section 12.11, “XML Functions”

UPPER()

Section 12.10, “Cast Functions and Operators”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 10.2.1, “Character Set Repertoire”
Section 12.5, “String Functions and Operators”
Section 10.10.1, “Unicode Character Sets”
Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”

USER()

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.7.1.1, “ALTER USER Syntax”
Section 10.8.4, “Collation Coercibility in Expressions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Implementing Proxy User Support in Authentication Plugins
Section 12.15, “Information Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 6.4.3.2, “Password Validation Options and Variables”
Section 6.2.18, “Proxy Users”
Section 17.5.1.14, “Replication and System Functions”
Section 6.2.22, “SQL-Based Account Activity Auditing”
Section 10.2.2, “UTF-8 for Metadata”
Writing the Server-Side Authentication Plugin

UTC_DATE
Section 12.7, “Date and Time Functions”

UTC_DATE()
Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”

UTC_TIME
Section 12.7, “Date and Time Functions”

UTC_TIME()
Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”

UTC_TIMESTAMP
Section 12.7, “Date and Time Functions”

UTC_TIMESTAMP()
Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 5.1.13, “MySQL Server Time Zone Support”

UUID()
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.2.1.20, “Function Call Optimization”
Section 12.24, “Miscellaneous Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 28.7.7.69, “mysql_session_track_get_first()”
Section 17.5.1.14, “Replication and System Functions”
Section 24.7, “Stored Program Binary Logging”

UUID_SHORT()
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.24, “Miscellaneous Functions”

UUID_TO_BIN()
Section 12.12, “Bit Functions and Operators”
Section 12.24, “Miscellaneous Functions”

V

[index top]

VALIDATE_PASSWORD_STRENGTH()
Section 12.13, “Encryption and Compression Functions”
Section 6.4.3.2, “Password Validation Options and Variables”
Section 6.4.3, “The Password Validation Component”

Section 29.2.1, “Types of Plugins”

VALUES()

Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 12.24, “Miscellaneous Functions”

VAR_POP()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 8.2.1.21, “Window Function Optimization”

VAR_SAMP()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 8.2.1.21, “Window Function Optimization”

VARIANCE()

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”

VERSION()

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 6.4.5.4, “Audit Log File Formats”
Section B.4.4.1, “Case Sensitivity in String Searches”
Section 10.8.4, “Collation Coercibility in Expressions”
Section 12.15, “Information Functions”
Section 17.5.1.14, “Replication and System Functions”
Section 27.4.3.47, “The version View”
Section 10.2.2, “UTF-8 for Metadata”

W

[index top]

WAIT_FOR_EXECUTED_GTID_SET()

Section 12.18, “Functions Used with Global Transaction Identifiers (GTIDs)”
Section 17.1.5.1, “Replication Mode Concepts”
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”

WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()

Section 12.18, “Functions Used with Global Transaction Identifiers (GTIDs)”

WEEK()

Section 12.7, “Date and Time Functions”
Section 5.1.8, “Server System Variables”

WEEKDAY()

Section 12.7, “Date and Time Functions”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 23.6.3, “Partitioning Limitations Relating to Functions”
Section 23.2, “Partitioning Types”

WEEKOFYEAR()

Section 12.7, “Date and Time Functions”

WEIGHT_STRING()

Section 10.14, “Adding a Collation to a Character Set”
Section B.4.4.1, “Case Sensitivity in String Searches”
Section 12.5, “String Functions and Operators”
Section 10.10.1, “Unicode Character Sets”

Y

[index top]

YEAR()

Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”
Section 23.2.4, “HASH Partitioning”
Section 23.2.7, “How MySQL Partitioning Handles NULL”
Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 23.4, “Partition Pruning”
Section 23.6.3, “Partitioning Limitations Relating to Functions”
Section 23.2, “Partitioning Types”
Section 23.2.1, “RANGE Partitioning”

YEARWEEK()

Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”

INFORMATION_SCHEMA Index

C | E | F | G | I | K | N | O | P | R | S | T | U | V

C

[index top]

CHARACTER_SETS

Section 10.3.8, “Character Set Introducers”
Section 10.2, “Character Sets and Collations in MySQL”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 10.3.5, “Column Character Set and Collation”
Section 10.3.3, “Database Character Set and Collation”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 13.7.7.3, “SHOW CHARACTER SET Syntax”
Section 10.10, “Supported Character Sets and Collations”
Section 10.3.4, “Table Character Set and Collation”
Section 25.2, “The INFORMATION_SCHEMA CHARACTER_SETS Table”

CHECK_CONSTRAINTS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.3, “The INFORMATION_SCHEMA CHECK_CONSTRAINTS Table”

COLLATION_CHARACTER_SET_APPLICABILITY

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”

Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 25.5, “The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table”

COLLATIONS

Section 28.7.5, “C API Data Structures”
Section 10.15, “Character Set Configuration”
Section 10.2, “Character Sets and Collations in MySQL”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 13.7.7.4, “SHOW COLLATION Syntax”
Section 10.8.5, “The binary Collation Compared to _bin Collations”
Section 11.4.1, “The CHAR and VARCHAR Types”
Section 25.4, “The INFORMATION_SCHEMA COLLATIONS Table”
Section 10.10.1, “Unicode Character Sets”

COLUMN_PRIVILEGES

Section 25.7, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”

COLUMN_STATISTICS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 8.9.6, “Optimizer Statistics”
Section 25.8, “The INFORMATION_SCHEMA COLUMN_STATISTICS Table”

COLUMNS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 13.7.7.5, “SHOW COLUMNS Syntax”
Section 25.6, “The INFORMATION_SCHEMA COLUMNS Table”
Section 25.39.1, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table”
Section 25.39.2, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table”
Section 25.39.3, “The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table”
Section 25.39.4, “The INFORMATION_SCHEMA INNODB_CACHED_INDEXES Table”
Section 25.39.5, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”
Section 25.39.7, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”
Section 25.39.6, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”
Section 25.39.8, “The INFORMATION_SCHEMA INNODB_COLUMNS Table”
Section 25.39.9, “The INFORMATION_SCHEMA INNODB_DATAFILES Table”
Section 25.39.10, “The INFORMATION_SCHEMA INNODB_FIELDS Table”
Section 25.39.11, “The INFORMATION_SCHEMA INNODB_FOREIGN Table”
Section 25.39.12, “The INFORMATION_SCHEMA INNODB_FOREIGN_COLS Table”
Section 25.39.13, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 25.39.14, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”
Section 25.39.15, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”
Section 25.39.16, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 25.39.17, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”
Section 25.39.18, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”
Section 25.39.19, “The INFORMATION_SCHEMA INNODB_INDEXES Table”
Section 25.39.22, “The INFORMATION_SCHEMA INNODB_METRICS Table”
Section 25.39.23, “The INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES Table”
Section 25.39.24, “The INFORMATION_SCHEMA INNODB_TABLES Table”
Section 25.39.25, “The INFORMATION_SCHEMA INNODB_TABLESPACES Table”
Section 25.39.26, “The INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF Table”
Section 25.39.27, “The INFORMATION_SCHEMA INNODB_TABLESTATS View”

Section 25.39.28, “The INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO Table”
Section 25.39.29, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 25.39.30, “The INFORMATION_SCHEMA INNODB_VIRTUAL Table”
Section 25.27, “The INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS Table”

CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 6.4.2.2, “Connection-Control System and Status Variables”
Section 6.4.2, “The Connection-Control Plugins”
Section 25.41.1, “The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table”

E

[index top]

ENGINES

Section 22.5.9, “Quick Reference: NDB Cluster SQL Statements”
Section 5.1.8, “Server System Variables”
Section 13.7.7.16, “SHOW ENGINES Syntax”
Section 25.9, “The INFORMATION_SCHEMA ENGINES Table”

EVENTS

Section 24.4.4, “Event Metadata”
Section 24.4.2, “Event Scheduler Configuration”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 17.5.1.16, “Replication of Invoked Features”
Section 13.7.7.18, “SHOW EVENTS Syntax”
Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”

F

[index top]

FILES

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 22.5.11, “INFORMATION_SCHEMA Tables for NDB Cluster”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”
Section 25.39.9, “The INFORMATION_SCHEMA INNODB_DATAFILES Table”
Section 25.39.25, “The INFORMATION_SCHEMA INNODB_TABLESPACES Table”
Section 25.31, “The INFORMATION_SCHEMA TABLESPACES Table”

G

[index top]

GLOBAL_STATUS

Section 22.5, “Management of NDB Cluster”
Section 22.5.17, “NDB API Statistics Counters and Variables”
Section 22.6, “NDB Cluster Replication”

I

[index top]

INFORMATION_SCHEMA

Section 2.11.4, “Changes in MySQL 8.0”
Section 14.1, “Data Dictionary Schema”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 22.5.11, “INFORMATION_SCHEMA Tables for NDB Cluster”
Section 15.14, “InnoDB INFORMATION_SCHEMA Tables”
Chapter 14, *MySQL Data Dictionary*
MySQL Glossary
Section 6.2.9, “Reserved Accounts”
Section B.3.1, “Server Error Message Reference”
Section 5.2, “The MySQL Data Directory”
Section 27.2, “Using the sys Schema”
Section 1.4, “What Is New in MySQL 8.0”

INFORMATION_SCHEMA.CHARACTER_SETS

Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”

INFORMATION_SCHEMA.COLLATIONS

Section 10.14.2, “Choosing a Collation ID”
Section 8.9.6, “Optimizer Statistics”

INFORMATION_SCHEMA.COLUMN_STATISTICS

Section 8.9.6, “Optimizer Statistics”

INFORMATION_SCHEMA.COLUMNS

Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 26.1, “Performance Schema Quick Start”
Section 5.1.8, “Server System Variables”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”

INFORMATION_SCHEMA.ENGINES

Section 26.1, “Performance Schema Quick Start”
Section 22.5.9, “Quick Reference: NDB Cluster SQL Statements”
Section 15.1.3, “Verifying that InnoDB is the Default Storage Engine”

INFORMATION_SCHEMA.EVENTS

Section 24.4.4, “Event Metadata”
Section 17.5.1.16, “Replication of Invoked Features”
Section 24.8, “Restrictions on Stored Programs”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”

INFORMATION_SCHEMA.FILES

Section 13.1.6, “ALTER LOGFILE GROUP Syntax”
Section 13.1.10, “ALTER TABLESPACE Syntax”
Section 2.11.4, “Changes in MySQL 8.0”
Section 5.6.7.3, “Cloning Remote Data”
Section 13.1.16, “CREATE LOGFILE GROUP Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”

Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”
Section 22.5.13.3, “NDB Cluster Disk Data Storage Requirements”
Section 22.4.9, “`ndb_desc` — Describe NDB Tables”
Section 15.14.8, “Retrieving InnoDB Tablespace Metadata from INFORMATION_SCHEMA.FILES”
Section 15.6.3.5, “Temporary Tablespaces”
Section 15.6.3.4, “Undo Tablespaces”
Section 22.1.4, “What is New in NDB Cluster”

INFORMATION_SCHEMA.GLOBAL_STATUS

Section 22.5.17, “NDB API Statistics Counters and Variables”

INFORMATION_SCHEMA.INNODB_BUFFER_PAGE

Section 15.5.2, “Change Buffer”

INFORMATION_SCHEMA.INNODB_CACHED_INDEXES

Section 1.4, “What Is New in MySQL 8.0”

INFORMATION_SCHEMA.INNODB_CMP

MySQL Glossary
Section 15.9.1.3, “Tuning Compression for InnoDB Tables”
Section 15.14.1.3, “Using the Compression Information Schema Tables”

INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.9.1.3, “Tuning Compression for InnoDB Tables”

INFORMATION_SCHEMA.INNODB_CMPMEM

Section 15.14.1.3, “Using the Compression Information Schema Tables”

INFORMATION_SCHEMA.INNODB_COLUMNS

Section 15.12.1, “Online DDL Operations”

INFORMATION_SCHEMA.INNODB_DATAFILES

Section 2.11.4, “Changes in MySQL 8.0”

INFORMATION_SCHEMA.INNODB_FT_CONFIG

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”

INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD

Section 12.9.4, “Full-Text Stopwords”

INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 12.9.9, “MeCab Full-Text Parser Plugin”
Section 12.9.8, “ngram Full-Text Parser”

INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE

Section 12.9.4, “Full-Text Stopwords”

INFORMATION_SCHEMA.INNODB_INDEXES

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”

INFORMATION_SCHEMA.INNODB_METRICS

Section 15.5.2, “Change Buffer”
Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.3.4, “Undo Tablespaces”

INFORMATION_SCHEMA.INNODB_TABLES

Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 15.10, “InnoDB Row Formats”
Section 15.12.1, “Online DDL Operations”

INFORMATION_SCHEMA.INNODB_TABLESPACES

Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 15.9.2, “InnoDB Page Compression”
Section 15.6.3.4, “Undo Tablespaces”
Section 1.4, “What Is New in MySQL 8.0”

INFORMATION_SCHEMA.INNODB_TABLESPACES_BRIEF

Section 1.4, “What Is New in MySQL 8.0”

INFORMATION_SCHEMA.INNODB_TABLESTATS

Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”

INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO

Section 15.6.3.5, “Temporary Tablespaces”

INFORMATION_SCHEMA.INNODB_TRX

Section 15.13, “InnoDB Startup Options and System Variables”

INFORMATION_SCHEMA.KEY_COLUMN_USAGE

Section 1.8.3.2, “FOREIGN KEY Constraints”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

INFORMATION_SCHEMA.OPTIMIZER_TRACE

Section 8.2.1.2, “Range Optimization”

INFORMATION_SCHEMA.PARTITIONS

Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 23.2.7, “How MySQL Partitioning Handles NULL”
Section 23.2.5, “KEY Partitioning”
Section 23.3.5, “Obtaining Information About Partitions”
Section 23.2.3.1, “RANGE COLUMNS partitioning”

INFORMATION_SCHEMA.PLUGINS

Section 5.6.7.3, “Cloning Remote Data”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 5.6.1, “Installing and Uninstalling Plugins”

Section 5.6.5.1, “Installing or Uninstalling ddl_rewriter”
Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”
Section 5.6.7.1, “Installing the Clone Plugin”
Section 6.4.4.1, “Keyring Plugin Installation”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section A.2, “MySQL 8.0 FAQ: Storage Engines”
Section 6.4.1.8, “No-Login Pluggable Authentication”
Section 5.6.2, “Obtaining Server Plugin Information”
Section 6.4.1.5, “PAM Pluggable Authentication”
Section 29.2.2, “Plugin API Characteristics”
Section 29.2.3, “Plugin API Components”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Server Plugin Library and Plugin Descriptors
Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”
Section 6.4.1.10, “Test Pluggable Authentication”
Section 5.6.3.2, “Thread Pool Installation”
Section 1.4, “What Is New in MySQL 8.0”
Section 6.4.1.6, “Windows Pluggable Authentication”
Section 29.2.4.8, “Writing Audit Plugins”
Section 29.2.4.5, “Writing Daemon Plugins”
Section 29.2.4.4, “Writing Full-Text Parser Plugins”
Section 29.2.4.6, “Writing INFORMATION_SCHEMA Plugins”
Section 29.2.4.12, “Writing Keyring Plugins”
Section 29.2.4.10, “Writing Password-Validation Plugins”
Writing the Server-Side Authentication Plugin

INFORMATION_SCHEMA.PROCESSLIST

Section 8.14, “Examining Thread Information”
Section 12.15, “Information Functions”
Section 26.6, “Performance Schema Instrument Naming Conventions”
Section 26.12.5, “Performance Schema Stage Event Tables”
Section 26.12.18.5, “The threads Table”

INFORMATION_SCHEMA.RESOURCE_GROUPS

Section 8.12.5, “Resource Groups”

INFORMATION_SCHEMA.ROUTINES

Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”

INFORMATION_SCHEMA.SCHEMATA

Section 10.3.3, “Database Character Set and Collation”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”

INFORMATION_SCHEMA.SESSION_STATUS

Section 22.5.17, “NDB API Statistics Counters and Variables”

INFORMATION_SCHEMA.STATISTICS

Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 8.9.4, “Index Hints”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.3.12, “Invisible Indexes”
Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

Section 5.3, “The mysql System Schema”

INFORMATION_SCHEMA.TABLE_CONSTRAINTS

Section 15.12.1, “Online DDL Operations”

INFORMATION_SCHEMA.TABLES

Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 13.1.9, “ALTER TABLE Syntax”
Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 15.9.2, “InnoDB Page Compression”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 14.2, “Removal of File-based Metadata Storage”
Section 27.4.4.2, “The diagnostics() Procedure”
Section 27.4.2.1, “The sys_config Table”
Section 5.6.3.2, “Thread Pool Installation”

INFORMATION_SCHEMA.TRIGGERS

Section A.5, “MySQL 8.0 FAQ: Triggers”

INFORMATION_SCHEMA.VIEWS

Section 24.5.3, “Updatable and Insertable Views”

INNODB_BUFFER_PAGE

Section 15.5.2, “Change Buffer”
Section 15.14.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”
Section 27.1, “Prerequisites for Using the sys Schema”
Section 25.39.1, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table”
Section 25.39.2, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table”
Section 27.4.3.7, “The innodb_buffer_stats_by_schema and x$innodb_buffer_stats_by_schema Views”
Section 27.4.3.8, “The innodb_buffer_stats_by_table and x$innodb_buffer_stats_by_table Views”

INNODB_BUFFER_PAGE_LRU

Section 15.14.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”
Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”
Section 25.39.2, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table”

INNODB_BUFFER_POOL_STATS

Section 15.5.1, “Buffer Pool”
Section 15.14.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”
Section 25.39.3, “The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table”

INNODB_CACHED_INDEXES

Section 25.39.4, “The INFORMATION_SCHEMA INNODB_CACHED_INDEXES Table”

INNODB_CMP

Section 15.14.1, “InnoDB INFORMATION_SCHEMA Tables about Compression”
Section 15.14.1.1, “INNODB_CMP and INNODB_CMP_RESET”
Section 15.14.1.2, “INNODB_CMPMEM and INNODB_CMPMEM_RESET”
Section 15.9.1.4, “Monitoring InnoDB Table Compression at Runtime”
Section 25.39.5, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”
Section 15.14.1.3, “Using the Compression Information Schema Tables”

INNODB_CMP_PER_INDEX

Section 15.9.1.4, “Monitoring InnoDB Table Compression at Runtime”
Section 25.39.7, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”
Section 15.14.1.3, “Using the Compression Information Schema Tables”

INNODB_CMP_PER_INDEX_RESET

Section 25.39.7, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”

INNODB_CMP_RESET

Section 15.14.1, “InnoDB INFORMATION_SCHEMA Tables about Compression”
Section 15.14.1.1, “INNODB_CMP and INNODB_CMP_RESET”
Section 15.14.1.2, “INNODB_CMPMEM and INNODB_CMPMEM_RESET”
Section 25.39.5, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”

INNODB_CMPMEM

Section 15.14.1, “InnoDB INFORMATION_SCHEMA Tables about Compression”
Section 15.14.1.2, “INNODB_CMPMEM and INNODB_CMPMEM_RESET”
Section 25.39.6, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”
Section 15.14.1.3, “Using the Compression Information Schema Tables”

INNODB_CMPMEM_RESET

Section 15.14.1.2, “INNODB_CMPMEM and INNODB_CMPMEM_RESET”
Section 25.39.6, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”

INNODB_COLUMNS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 25.39.8, “The INFORMATION_SCHEMA INNODB_COLUMNS Table”
Section 25.39.30, “The INFORMATION_SCHEMA INNODB_VIRTUAL Table”

INNODB_DATAFILES

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 15.14.8, “Retrieving InnoDB Tablespace Metadata from INFORMATION_SCHEMA.FILES”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”
Section 25.39.9, “The INFORMATION_SCHEMA INNODB_DATAFILES Table”
Section 25.39.26, “The INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF Table”
Section 25.31, “The INFORMATION_SCHEMA TABLESPACES Table”

INNODB_FIELDS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 25.39.10, “The INFORMATION_SCHEMA INNODB_FIELDS Table”

INNODB_FOREIGN

Section 1.8.3.2, “FOREIGN KEY Constraints”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”

Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 25.39.11, “The INFORMATION_SCHEMA INNODB_FOREIGN Table”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

INNODB_FOREIGN_COLS

Section 1.8.3.2, “FOREIGN KEY Constraints”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 25.39.12, “The INFORMATION_SCHEMA INNODB_FOREIGN_COLS Table”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

INNODB_FT_BEING_DELETED

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 25.39.13, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”

INNODB_FT_CONFIG

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 25.39.14, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”

INNODB_FT_DEFAULT_STOPWORD

Section 12.9.4, “Full-Text Stopwords”
Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 25.39.15, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”

INNODB_FT_DELETED

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 25.39.13, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 25.39.16, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”

INNODB_FT_INDEX_CACHE

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 25.39.17, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”

INNODB_FT_INDEX_TABLE

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 25.39.13, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 25.39.16, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 25.39.18, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”

INNODB_INDEXES

Section 15.8.11, “Configuring the Merge Threshold for Index Pages”

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 25.39.4, “The INFORMATION_SCHEMA INNODB_CACHED_INDEXES Table”
Section 25.39.19, “The INFORMATION_SCHEMA INNODB_INDEXES Table”

INNODB_LOCK_WAITS

Section 15.14.2, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”
Section 15.14.2.2, “InnoDB Lock and Lock-Wait Information”
Section 15.14.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”
Section 25.39.21, “The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table”
Section 15.14.2.1, “Using InnoDB Transaction and Locking Information”
Section 1.4, “What Is New in MySQL 8.0”

INNODB_LOCKS

Section 15.14.2, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”
Section 15.14.2.2, “InnoDB Lock and Lock-Wait Information”
Section 15.14.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”
Section 25.39.20, “The INFORMATION_SCHEMA INNODB_LOCKS Table”
Section 15.14.2.1, “Using InnoDB Transaction and Locking Information”
Section 1.4, “What Is New in MySQL 8.0”

INNODB_METRICS

Section 15.5.2, “Change Buffer”
Section 15.8.11, “Configuring the Merge Threshold for Index Pages”
Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
MySQL Glossary
Section 25.39.22, “The INFORMATION_SCHEMA INNODB_METRICS Table”
Section 27.4.3.21, “The metrics View”

INNODB_SESSION_TEMP_TABLESPACES

Section 15.6.3.5, “Temporary Tablespaces”
Section 25.39.23, “The INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES Table”
Section 1.4, “What Is New in MySQL 8.0”

INNODB_SYS_FOREIGN

Section 13.1.20.6, “Using FOREIGN KEY Constraints”

INNODB_SYS_FOREIGN_COLS

Section 13.1.20.6, “Using FOREIGN KEY Constraints”

INNODB_TABLES

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 25.39.4, “The INFORMATION_SCHEMA INNODB_CACHED_INDEXES Table”
Section 25.39.24, “The INFORMATION_SCHEMA INNODB_TABLES Table”
Section 1.4, “What Is New in MySQL 8.0”

INNODB_TABLESPACES

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”

Section 15.14.8, “Retrieving InnoDB Tablespace Metadata from INFORMATION_SCHEMA.FILES”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”
Section 25.39.25, “The INFORMATION_SCHEMA INNODB_TABLESPACES Table”
Section 25.39.26, “The INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF Table”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”
Section 25.31, “The INFORMATION_SCHEMA TABLESPACES Table”
Section 1.4, “What Is New in MySQL 8.0”

INNODB_TABLESPACES_BRIEF

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 25.39.26, “The INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF Table”

INNODB_TABLESTATS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 25.39.27, “The INFORMATION_SCHEMA INNODB_TABLESTATS View”

INNODB_TEMP_TABLE_INFO

Section 15.14.7, “InnoDB INFORMATION_SCHEMA Temporary Table Info Table”
Section 25.39.28, “The INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO Table”

INNODB_TRX

Section 15.14.2, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”
Section 15.14.2.2, “InnoDB Lock and Lock-Wait Information”
Section 15.14.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”
Section 26.12.12.1, “The data_locks Table”
Section 25.39.29, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 15.14.2.1, “Using InnoDB Transaction and Locking Information”

INNODB_VIRTUAL

Section 25.39.30, “The INFORMATION_SCHEMA INNODB_VIRTUAL Table”

K

[index top]

KEY_COLUMN_USAGE

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.12, “The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table”
Section 5.3, “The mysql System Schema”

KEYWORDS

Section 25.14, “The INFORMATION_SCHEMA KEYWORDS Table”

N

[index top]

NDB_TRANSID_MYSQL_CONNECTION_MAP

Section 22.5.10.34, “The ndbinfo server_operations Table”

Section 22.5.10.35, “The ndbinfo server_transactions Table”

ndb_transid_mysql_connection_map

Section 22.5.11, “INFORMATION_SCHEMA Tables for NDB Cluster”
MySQL Server Options for NDB Cluster
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”

O

[index top]

OPTIMIZER_TRACE

Section 25.15, “The INFORMATION_SCHEMA OPTIMIZER_TRACE Table”

P

[index top]

PARAMETERS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 13.7.7.28, “SHOW PROCEDURE STATUS Syntax”
Section 25.16, “The INFORMATION_SCHEMA PARAMETERS Table”
Section 25.23, “The INFORMATION_SCHEMA ROUTINES Table”

PARTITIONS

Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 23.2.7, “How MySQL Partitioning Handles NULL”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 23.3.5, “Obtaining Information About Partitions”
Chapter 23, *Partitioning*
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 25.17, “The INFORMATION_SCHEMA PARTITIONS Table”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”

PLUGINS

Section 13.7.4.4, “INSTALL PLUGIN Syntax”
Section 5.6.2, “Obtaining Server Plugin Information”
Section 25.18, “The INFORMATION_SCHEMA PLUGINS Table”

PROCESSLIST

Section 8.14, “Examining Thread Information”
Section 13.7.8.4, “KILL Syntax”
Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”
Section 15.14.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”
Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
Section 5.6.7.10, “Stopping a Cloning Operation”
Section 25.39.29, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 25.19, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 27.4.3.22, “The processlist and x$processlist Views”
Section 26.12.18.5, “The threads Table”

Section 15.14.2.1, “Using InnoDB Transaction and Locking Information”

PROFILING

Section 13.7.7.30, “SHOW PROFILE Syntax”
Section 25.20, “The INFORMATION_SCHEMA PROFILING Table”

R

[index top]

REFERENTIAL_CONSTRAINTS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.21, “The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table”
Section 5.3, “The mysql System Schema”

RESOURCE_GROUPS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 8.12.5, “Resource Groups”
Section 25.22, “The INFORMATION_SCHEMA RESOURCE_GROUPS Table”

ROUTINES

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.1, “Introduction”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 13.7.7.28, “SHOW PROCEDURE STATUS Syntax”
Section 24.2.3, “Stored Routine Metadata”
Section 25.23, “The INFORMATION_SCHEMA ROUTINES Table”

S

[index top]

SCHEMA_PRIVILEGES

Section 25.25, “The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table”

SCHEMATA

Section 6.2.3, “Grant Tables”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”
Section 13.7.7.14, “SHOW DATABASES Syntax”
Section 25.24, “The INFORMATION_SCHEMA SCHEMATA Table”

SESSION_STATUS

Section 22.5, “Management of NDB Cluster”
Section 22.5.17, “NDB API Statistics Counters and Variables”
Section 22.6, “NDB Cluster Replication”

ST_GEOMETRY_COLUMNS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.6, “The INFORMATION_SCHEMA COLUMNS Table”
Section 25.27, “The INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS Table”

ST_SPATIAL_REFERENCE_SYSTEMS

Section 13.1.19, “CREATE SPATIAL REFERENCE SYSTEM Syntax”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 11.5.5, “Spatial Reference System Support”
Section 25.28, “The INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS Table”

ST_UNITS_OF_MEASURE

Section 12.16.7.3, “LineString and MultiLineString Property Functions”
Section 12.16.9.1, “Spatial Relation Functions That Use Object Shapes”
Section 25.29, “The INFORMATION_SCHEMA ST_UNITS_OF_MEASURE Table”

STATISTICS

Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 14.7, “Data Dictionary Usage Differences”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 13.7.7.22, “SHOW INDEX Syntax”
Section 25.26, “The INFORMATION_SCHEMA STATISTICS Table”

T

[index top]

TABLE_CONSTRAINTS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.21, “The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table”
Section 25.32, “The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table”

TABLE_PRIVILEGES

Section 25.33, “The INFORMATION_SCHEMA TABLE_PRIVILEGES Table”

TABLES

Section 13.1.20, “CREATE TABLE Syntax”
Section 14.7, “Data Dictionary Usage Differences”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.1, “Introduction”
Section 13.1.20.11, “Setting NDB_TABLE Options”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 13.7.7.37, “SHOW TABLES Syntax”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”

TABLESPACES

Section 25.31, “The INFORMATION_SCHEMA TABLESPACES Table”

TP_THREAD_STATE

Section 5.6.3.2, “Thread Pool Installation”

TRIGGERS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 2.11.5, “Preparing Your Installation for Upgrade”
Section 13.7.7.11, “SHOW CREATE TRIGGER Syntax”
Section 13.7.7.38, “SHOW TRIGGERS Syntax”
Section 25.34, “The INFORMATION_SCHEMA TRIGGERS Table”

Section 24.3.2, “Trigger Metadata”

U

[index top]

USER_PRIVILEGES

Section 6.2.2, “Privileges Provided by MySQL”
Section 25.35, “The INFORMATION_SCHEMA USER_PRIVILEGES Table”

V

[index top]

VIEW_ROUTINE_USAGE

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.37, “The INFORMATION_SCHEMA VIEW_ROUTINE_USAGE Table”

VIEW_TABLE_USAGE

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.38, “The INFORMATION_SCHEMA VIEW_TABLE_USAGE Table”

VIEWS

Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”
Section 24.5.5, “View Metadata”

Join Types Index

A | C | E | F | I | R | S | U

A

[index top]

ALL

Section 8.2.1.23, “Avoiding Full Table Scans”
Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.7, “Nested-Loop Join Algorithms”

C

[index top]

const

Section 8.8.2, “EXPLAIN Output Format”
Section 8.8.3, “Extended EXPLAIN Output Format”
NDB Cluster System Variables
Section 8.9.3, “Optimizer Hints”
Section 8.2.1.16, “ORDER BY Optimization”
Section 8.2.1.2, “Range Optimization”

Section 13.2.10, “SELECT Syntax”

E

[index top]

eq_ref

Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.6, “Index Condition Pushdown Optimization”
Section 16.7.1, “MERGE Table Advantages and Disadvantages”
NDB Cluster System Variables
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 26.12.4.1, “The events_waits_current Table”

F

[index top]

fulltext

Section 8.8.2, “EXPLAIN Output Format”

I

[index top]

index

Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.7, “Nested-Loop Join Algorithms”

index_merge

Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.3, “Index Merge Optimization”

index_subquery

Section 8.8.2, “EXPLAIN Output Format”
Section 13.2.11.11, “Optimizing Subqueries”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”

R

[index top]

range

Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.17, “GROUP BY Optimization”
Section 8.2.1.6, “Index Condition Pushdown Optimization”
Section 8.2.1.3, “Index Merge Optimization”
Section 8.2.1.7, “Nested-Loop Join Algorithms”
Section 8.9.3, “Optimizer Hints”
Section 8.2.1.2, “Range Optimization”

ref

Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 8.2.1.6, “Index Condition Pushdown Optimization”
Section 8.3.8, “InnoDB and MyISAM Index Statistics Collection”
Section 16.7.1, “MERGE Table Advantages and Disadvantages”
NDB Cluster System Variables
Section 8.9.3, “Optimizer Hints”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”

ref_or_null

Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.6, “Index Condition Pushdown Optimization”
Section 8.2.1.15, “IS NULL Optimization”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”

S

[index top]

system

Section 8.8.2, “EXPLAIN Output Format”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 8.2.1.2, “Range Optimization”
Section 13.2.10, “SELECT Syntax”

U

[index top]

unique_subquery

Section 8.8.2, “EXPLAIN Output Format”
Section 13.2.11.11, “Optimizing Subqueries”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”

Operator Index

Symbols | A | B | C | D | E | I | L | M | N | O | R | V | X

Symbols

[index top]

-

Section 12.6.1, “Arithmetic Operators”
Section 12.10, “Cast Functions and Operators”
Section 9.5, “Expressions”
Section 11.1.1, “Numeric Type Overview”
Section 23.6, “Restrictions and Limitations on Partitioning”

!
Section 9.5, “Expressions”
Section 12.3.3, “Logical Operators”
Section 12.3.1, “Operator Precedence”
Section 1.4, “What Is New in MySQL 8.0”

!=
Section 12.3.2, “Comparison Functions and Operators”
Section 12.3.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

%
Section 12.6.1, “Arithmetic Operators”

&
Section 12.12, “Bit Functions and Operators”
Section 13.1.20, “CREATE TABLE Syntax”
Section 23.6, “Restrictions and Limitations on Partitioning”

&&
Section 12.3.3, “Logical Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 1.4, “What Is New in MySQL 8.0”

>
Section 12.3.2, “Comparison Functions and Operators”
Section 8.3.9, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 8.3.11, “Optimizer Use of Generated Column Indexes”
Section 8.2.1.2, “Range Optimization”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

->
Section 12.17.3, “Functions That Search JSON Values”
Section 13.1.20.10, “Secondary Indexes and Generated Columns”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

>>
Section 12.12, “Bit Functions and Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 23.6, “Restrictions and Limitations on Partitioning”

->>
Section 13.1.15, “CREATE INDEX Syntax”
Section 13.1.20.10, “Secondary Indexes and Generated Columns”
Section 11.6, “The JSON Data Type”

Section 1.4, “What Is New in MySQL 8.0”

>=

Section 12.3.2, “Comparison Functions and Operators”
Section 8.3.9, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 8.3.11, “Optimizer Use of Generated Column Indexes”
Section 8.2.1.2, “Range Optimization”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

<

Section 12.3.2, “Comparison Functions and Operators”
Section 8.3.9, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 8.3.11, “Optimizer Use of Generated Column Indexes”
Section 8.2.1.2, “Range Optimization”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”
Section 3.3.4.6, “Working with NULL Values”

<>

Section 12.3.2, “Comparison Functions and Operators”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”
Section 3.3.4.6, “Working with NULL Values”

<<

Section 12.12, “Bit Functions and Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 23.6, “Restrictions and Limitations on Partitioning”

<=

Section 12.3.2, “Comparison Functions and Operators”
Section 8.3.9, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 8.3.11, “Optimizer Use of Generated Column Indexes”
Section 8.2.1.2, “Range Optimization”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

<=>

Section 12.3.2, “Comparison Functions and Operators”

Section 8.8.2, “EXPLAIN Output Format”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”
Section 11.6, “The JSON Data Type”
Section 12.2, “Type Conversion in Expression Evaluation”
Section 1.4, “What Is New in MySQL 8.0”

*****

Section 12.6.1, “Arithmetic Operators”
Section 11.1.1, “Numeric Type Overview”
Section 23.6, “Restrictions and Limitations on Partitioning”

+

Section 12.6.1, “Arithmetic Operators”
Section 12.10, “Cast Functions and Operators”
Section 9.5, “Expressions”
Section 11.1.1, “Numeric Type Overview”
Section 23.6, “Restrictions and Limitations on Partitioning”

/

Section 12.6.1, “Arithmetic Operators”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 5.1.8, “Server System Variables”

:=

Section 12.3.4, “Assignment Operators”
Section 12.3.1, “Operator Precedence”
Section 13.7.6.1, “SET Syntax for Variable Assignment”
Section 9.4, “User-Defined Variables”

=

Section 12.3.4, “Assignment Operators”
Section 12.3.2, “Comparison Functions and Operators”
Section 8.3.9, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 8.3.11, “Optimizer Use of Generated Column Indexes”
Section 8.2.1.2, “Range Optimization”
Section 13.2.11.13, “Restrictions on Subqueries”
Section 13.7.6.1, “SET Syntax for Variable Assignment”
Section 12.5.1, “String Comparison Functions and Operators”
Section 11.6, “The JSON Data Type”
Section 9.4, “User-Defined Variables”
Section 1.4, “What Is New in MySQL 8.0”
Section 3.3.4.6, “Working with NULL Values”

^

Section 12.12, “Bit Functions and Operators”
Section 9.5, “Expressions”
Section 12.3.1, “Operator Precedence”

Section 23.6, “Restrictions and Limitations on Partitioning”

|

Section 12.12, “Bit Functions and Operators”
Section 23.6, “Restrictions and Limitations on Partitioning”

||

Section 12.5.3, “Character Set and Collation of Function Results”
Section 10.8.2, “COLLATE Clause Precedence”
Section 9.5, “Expressions”
Section 12.3.3, “Logical Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 5.1.11, “Server SQL Modes”
Section 1.4, “What Is New in MySQL 8.0”

~

Section 12.12, “Bit Functions and Operators”
Section 23.6, “Restrictions and Limitations on Partitioning”

A

[index top]

AND

Section 8.3.9, “Comparison of B-Tree and Hash Indexes”
Section 13.1.20, “CREATE TABLE Syntax”
Section 8.2.1.3, “Index Merge Optimization”
Section 12.3.3, “Logical Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 8.2.1.2, “Range Optimization”
Section 13.2.11.13, “Restrictions on Subqueries”
Section 8.2.1.22, “Row Constructor Expression Optimization”
Section 3.6.7, “Searching on Two Keys”
Section 20.3.4.2, “Select Tables”
Section 20.4.4.2, “Select Tables”
Section 3.3.4.2, “Selecting Particular Rows”
Section 12.5.1, “String Comparison Functions and Operators”
Section 24.5.2, “View Processing Algorithms”
Section 1.4, “What Is New in MySQL 8.0”

B

[index top]

BETWEEN

Section 12.3.2, “Comparison Functions and Operators”
Section 8.3.9, “Comparison of B-Tree and Hash Indexes”
Section 8.2.1.13, “Condition Filtering”
Section 8.8.2, “EXPLAIN Output Format”

Section 8.3.11, “Optimizer Use of Generated Column Indexes”
Section 8.2.1.2, “Range Optimization”
Section 11.6, “The JSON Data Type”
Section 12.2, “Type Conversion in Expression Evaluation”

BINARY

Section 12.12, “Bit Functions and Operators”
Section 12.10, “Cast Functions and Operators”
Section 8.4.2.2, “Optimizing for Character and String Types”
Section 3.3.4.7, “Pattern Matching”
Section 3.3.4.4, “Sorting Rows”

C

[index top]

CASE

Section 13.6.5.1, “CASE Syntax”
Section 12.4, “Control Flow Functions”
Section 9.5, “Expressions”
Section 1.8.1, “MySQL Extensions to Standard SQL”

CASE value WHEN END

Section 12.4, “Control Flow Functions”

CASE WHEN END

Section 12.4, “Control Flow Functions”

CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END

Section 12.4, “Control Flow Functions”

column->>path

Section 12.17.3, “Functions That Search JSON Values”

column->path

Section 12.17.3, “Functions That Search JSON Values”
Section 11.6, “The JSON Data Type”

D

[index top]

DIV

Section 12.6.1, “Arithmetic Operators”
Section 23.6, “Restrictions and Limitations on Partitioning”

E

[index top]

expr BETWEEN min AND max

Section 12.3.2, “Comparison Functions and Operators”

expr IN ()
Section 12.3.2, “Comparison Functions and Operators”

expr LIKE pat
Section 12.5.1, “String Comparison Functions and Operators”

expr NOT BETWEEN min AND max
Section 12.3.2, “Comparison Functions and Operators”

expr NOT IN ()
Section 12.3.2, “Comparison Functions and Operators”

expr NOT LIKE pat
Section 12.5.1, “String Comparison Functions and Operators”

expr NOT REGEXP pat
Section 12.5.2, “Regular Expressions”

expr NOT RLIKE pat
Section 12.5.2, “Regular Expressions”

expr REGEXP pat
Section 12.5.2, “Regular Expressions”

expr RLIKE pat
Section 12.5.2, “Regular Expressions”

expr1 SOUNDS LIKE expr2
Section 12.5, “String Functions and Operators”

I

[index top]

IN()
Section 8.8.2, “EXPLAIN Output Format”
Section 12.3.1, “Operator Precedence”
Section 8.3.11, “Optimizer Use of Generated Column Indexes”
Section 8.2.1.2, “Range Optimization”
Section 8.2.1.22, “Row Constructor Expression Optimization”
Section 11.6, “The JSON Data Type”
Section 12.2, “Type Conversion in Expression Evaluation”

IS
Section 12.3.1, “Operator Precedence”

IS boolean_value
Section 12.3.2, “Comparison Functions and Operators”

IS NOT boolean_value
Section 12.3.2, “Comparison Functions and Operators”

IS NOT NULL

Section 12.3.2, “Comparison Functions and Operators”
Section B.4.4.3, “Problems with NULL Values”
Section 8.2.1.2, “Range Optimization”
Section 3.3.4.6, “Working with NULL Values”

IS NULL

Section 12.3.2, “Comparison Functions and Operators”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.15, “IS NULL Optimization”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section B.4.4.3, “Problems with NULL Values”
Section 8.2.1.2, “Range Optimization”
Section 5.1.8, “Server System Variables”
Section 3.3.4.6, “Working with NULL Values”

L

[index top]

LIKE

Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 12.10, “Cast Functions and Operators”
Section 10.2, “Character Sets and Collations in MySQL”
Section 8.3.9, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 25.42, “Extensions to SHOW Statements”
Section 12.17.3, “Functions That Search JSON Values”
Section 13.8.3, “HELP Syntax”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 4.5.1.4, “mysql Client Server-Side Help”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
Section 12.3.1, “Operator Precedence”
Section 3.3.4.7, “Pattern Matching”
Section 26.4.4, “Pre-Filtering by Instrument”
Section 22.5.9, “Quick Reference: NDB Cluster SQL Statements”
Section 8.2.1.2, “Range Optimization”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.7.3, “SHOW CHARACTER SET Syntax”
Section 13.7.7.4, “SHOW COLLATION Syntax”
Section 13.7.7.5, “SHOW COLUMNS Syntax”
Section 13.7.7.14, “SHOW DATABASES Syntax”
Section 13.7.7.18, “SHOW EVENTS Syntax”
Section 13.7.7.24, “SHOW OPEN TABLES Syntax”
Section 13.7.7.28, “SHOW PROCEDURE STATUS Syntax”
Section 13.7.7.35, “SHOW STATUS Syntax”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 13.7.7.37, “SHOW TABLES Syntax”
Section 13.7.7.38, “SHOW TRIGGERS Syntax”
Section 13.7.7.39, “SHOW VARIABLES Syntax”
Section 6.2.4, “Specifying Account Names”
Section 12.5.1, “String Comparison Functions and Operators”

Section 9.1.1, “String Literals”
Section 5.1.9.5, “Structured System Variables”
Section 11.4.1, “The CHAR and VARCHAR Types”
Section 27.4.4.5, “The ps_setup_disable_consumer() Procedure”
Section 27.4.4.6, “The ps_setup_disable_instrument() Procedure”
Section 27.4.4.9, “The ps_setup_enable_consumer() Procedure”
Section 27.4.4.10, “The ps_setup_enable_instrument() Procedure”
Section 11.4.5, “The SET Type”
Section 5.1.9, “Using System Variables”

LIKE '_A%'

Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”

LIKE 'pattern'

Section 8.2.1.2, “Range Optimization”
Section 13.7.7, “SHOW Syntax”

LIKE ... ESCAPE

Section B.4.7, “Known Issues in MySQL”

M

[index top]

MEMBER OF()

Section 13.1.15, “CREATE INDEX Syntax”
Section 12.17.3, “Functions That Search JSON Values”
Section 1.4, “What Is New in MySQL 8.0”

N

[index top]

N % M

Section 12.6.1, “Arithmetic Operators”
Section 12.6.2, “Mathematical Functions”

N MOD M

Section 12.6.1, “Arithmetic Operators”
Section 12.6.2, “Mathematical Functions”

NOT

Section 12.3.3, “Logical Operators”
Section 5.1.11, “Server SQL Modes”
Section 1.4, “What Is New in MySQL 8.0”

NOT IN()

Section 8.2.1.2, “Range Optimization”

NOT LIKE

Section 3.3.4.7, “Pattern Matching”

Section 12.5.1, “String Comparison Functions and Operators”

NOT REGEXP

Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.5.1, “String Comparison Functions and Operators”

NOT RLIKE

Section 12.5.1, “String Comparison Functions and Operators”

O

[index top]

OR

Section 9.5, “Expressions”
Section 13.7.1.6, “GRANT Syntax”
Section 8.2.1.3, “Index Merge Optimization”
Section 12.3.3, “Logical Operators”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 8.2.1.2, “Range Optimization”
Section 8.2.1.22, “Row Constructor Expression Optimization”
Section 3.6.7, “Searching on Two Keys”
Section 20.3.4.2, “Select Tables”
Section 20.4.4.2, “Select Tables”
Section 3.3.4.2, “Selecting Particular Rows”
Section 5.1.11, “Server SQL Modes”
Section 12.5.1, “String Comparison Functions and Operators”
Section 1.4, “What Is New in MySQL 8.0”

R

[index top]

REGEXP

Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 12.3.1, “Operator Precedence”
Section 3.3.4.7, “Pattern Matching”
Section 12.5.2, “Regular Expressions”
Section 10.11, “Restrictions on Character Sets”
Section 1.4, “What Is New in MySQL 8.0”

RLIKE

Section 3.3.4.7, “Pattern Matching”
Section 12.5.2, “Regular Expressions”
Section 10.11, “Restrictions on Character Sets”
Section 1.4, “What Is New in MySQL 8.0”

V

[index top]

value MEMBER OF()
Section 12.17.3, “Functions That Search JSON Values”

X

[index top]

XOR
Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 12.3.3, “Logical Operators”

Option Index

Symbols | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

Symbols

[index top]

--
Section 1.8.2.4, “'--' as the Start of a Comment”

-#
Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 5.1.7, “Server Command Options”
Section 29.5.4, “The DBUG Package”

-1
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

1231
Section 4.8.2, “`perror` — Display MySQL Error Message Information”

-?
Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”

Section 4.6.3, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.16, “ndb_perror — Obtain NDB Error Message Information”
Section 22.4.18, “ndb_print_file — Print NDB Disk Data File Contents”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 4.8.2, “perror — Display MySQL Error Message Information”
Section 5.1.7, “Server Command Options”
Section 1.3.2, “The Main Features of MySQL”
Section 4.2.2.1, “Using Options on the Command Line”

?

Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”

A

[index top]

-A

Section 4.5.1.1, “mysql Client Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 22.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 4.6.4.4, “Other myisamchk Options”

-a

Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 7.6.4, “MyISAM Table Optimization”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.2.1.1, “NDB Cluster Auto-Installer Requirements”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”
Section 4.6.4.4, “Other myisamchk Options”

--abort-on-error

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”
Section 22.4.15, “ndb_move_data — NDB Data Copy Utility”

--abort-slave-event-count

Section 17.1.6.3, “Replication Slave Options and Variables”

--add-drop-database

Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”

--add-drop-table

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”

--add-drop-trigger

Section 4.5.4, “mysqldump — A Database Backup Program”

--add-drop-user

Section 4.5.6, “mysqlpump — A Database Backup Program”

--add-locks

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”

--ai-increment

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--ai-offset

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--ai-prefetch-sz

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--all

Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”

--all-databases

Creating a Data Snapshot Using mysqldump
Section 14.7, “Data Dictionary Usage Differences”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--all-in-1
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--all-tablespaces
Section 4.5.4, “mysqldump — A Database Backup Program”

--allow-keywords
Section 4.5.4, “mysqldump — A Database Backup Program”

--allow-mismatches
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”

--allow-suspicious-udfs
Section 5.1.7, “Server Command Options”
Section 29.4.2.6, “UDF Security Precautions”

--analyze
Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.4.1, “myisamchk General Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.6.4.4, “Other myisamchk Options”

--ansi
Section 1.8, “MySQL Standards Compliance”
Section 5.1.7, “Server Command Options”

antonio
Section 6.4.1.5, “PAM Pluggable Authentication”

--append
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--apply-slave-statements
Section 4.5.4, “mysqldump — A Database Backup Program”

--audit-log
Section 6.4.5.9, “Audit Log Reference”
Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”

--auto-generate-sql
Section 4.5.8, “mysqlslap — Load Emulation Client”

--auto-generate-sql-add-autoincrement
Section 4.5.8, “mysqlslap — Load Emulation Client”

--auto-generate-sql-execute-number
Section 4.5.8, “mysqlslap — Load Emulation Client”

--auto-generate-sql-guid-primary
Section 4.5.8, “mysqlslap — Load Emulation Client”

--auto-generate-sql-load-type
Section 4.5.8, “mysqlslap — Load Emulation Client”

--auto-generate-sql-secondary-indexes
Section 4.5.8, “mysqlslap — Load Emulation Client”

--auto-generate-sql-unique-query-number
Section 4.5.8, “mysqlslap — Load Emulation Client”

--auto-generate-sql-unique-write-number
Section 4.5.8, “mysqlslap — Load Emulation Client”

--auto-generate-sql-write-number
Section 4.5.8, “mysqlslap — Load Emulation Client”

--auto-rehash
Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

auto-rehash
Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”

--auto-repair
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--auto-vertical-output
Section 4.5.1.1, “mysql Client Options”

--autocommit
Section 5.1.8, “Server System Variables”

B

[index top]

-B
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”

-b
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 22.4.9, “ndb_desc — Describe NDB Tables”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 4.6.4.4, “Other myisamchk Options”
Section 5.1.7, “Server Command Options”

--back_log
Section 2.7, “Installing MySQL on Solaris”

--backup
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”

backup-to-image
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

--backup_path
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Restoring a parallel backup in parallel
Restoring a parallel backup serially
Restoring to Fewer Nodes Than the Original

backup_path
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--backupid
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Restoring to Fewer Nodes Than the Original
Section 22.1.4, “What is New in NDB Cluster”

--base64-output
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

--basedir
Section 2.10.1, “Initializing the Data Directory”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 22.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.1.7, “Server Command Options”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”

basedir
Section 2.3.4.2, “Creating an Option File”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”

--batch
Section 4.5.1.3, “mysql Client Logging”
Section 4.5.1.1, “mysql Client Options”

--binary-as-hex
Section 4.5.1.1, “mysql Client Options”

--binary-mode

Section 9.6, “Comment Syntax”
Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”

--bind-address

Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”

--binlog-checksum

Section 17.1.6.4, “Binary Logging Options and Variables”

--binlog-do-db

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.4.4, “The Binary Log”

--binlog-format

Section 5.4.4.1, “Binary Logging Formats”
Section 22.6.2, “General Requirements for NDB Cluster Replication”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 22.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 22.1.7.6, “Unsupported or Missing Features in NDB Cluster”

--binlog-ignore-db

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.4.4, “The Binary Log”

--binlog-row-event-max-size

Section 17.1.6.4, “Binary Logging Options and Variables”

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 5.4.4.2, “Setting The Binary Log Format”

--blob-info

Section 22.4.9, “ndb_desc — Describe NDB Tables”

--block-search

Section 4.6.4.4, “Other myisamchk Options”

--bootstrap

Section 21.3, “Using MySQL Router with InnoDB Cluster”
Section 1.4, “What Is New in MySQL 8.0”

bootstrap_server_addresses

Section 21.3, “Using MySQL Router with InnoDB Cluster”

--browser-start-page

Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”

C

[index top]

-C

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 4.6.4.2, “myisamchk Check Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”

-c

Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 4.7.2, “my_print_defaults — Display Options from Option Files”
Section 4.6.3, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.8, “mysqlslap — Load Emulation Client”

Section 22.2.1.1, “NDB Cluster Auto-Installer Requirements”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Restoring to Fewer Nodes Than the Original

--ca-certs-file

Section 22.2.1.1, “NDB Cluster Auto-Installer Requirements”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”

--cert-file

Section 22.2.1.1, “NDB Cluster Auto-Installer Requirements”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”

--cflags

Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 4.7.1, “mysql_config — Display Options for Compiling Clients”

--character-set-client-handshake

Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 5.1.7, “Server Command Options”
Section 10.10.7.1, “The cp932 Character Set”

--character-set-server

Section 10.15, “Character Set Configuration”
Section 10.5, “Configuring Application Character Set and Collation”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 10.3.2, “Server Character Set and Collation”

--character-sets-dir

Section B.4.2.16, “Can't initialize character set”
Section 10.15, “Character Set Configuration”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 22.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--character_set_server

Section 2.9.7, “MySQL Source-Configuration Options”

--charset

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--check

Section 4.6.4.2, “myisamchk Check Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--check-only-changed

Section 4.6.4.2, “myisamchk Check Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--check-orphans

Section 22.4.6, “ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”

--check-upgrade

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--chroot

Section 5.1.7, “Server Command Options”

--clone

Section 5.6.7.1, “Installing the Clone Plugin”

CMAKE_BUILD_TYPE

Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_C_FLAGS

Section 29.5.1.1, “Compiling MySQL for Debugging”
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_C_FLAGS_build_type

Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_C_FLAGS_RELWITHDEBINFO

Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_CXX_FLAGS

Section 29.5.1.1, “Compiling MySQL for Debugging”
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_CXX_FLAGS_build_type

Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_CXX_FLAGS_RELWITHDEBINFO

Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_INSTALL_PREFIX

Section 29.2.4.3, “Compiling and Installing Plugin Libraries”
Section 2.9.5, “Installing MySQL Using a Development Source Tree”

Section 6.4.4.11, “Keyring System Variables”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.8.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.8, “Server System Variables”

CMAKE_PREFIX_PATH

Section 2.9.7, “MySQL Source-Configuration Options”

--collation-server

Section 10.15, “Character Set Configuration”
Section 10.5, “Configuring Application Character Set and Collation”
Section 10.3.2, “Server Character Set and Collation”

--collation_server

Section 2.9.7, “MySQL Source-Configuration Options”

--color

Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”

--column-names

Section 4.5.1.1, “mysql Client Options”
Section 4.2.2.4, “Program Option Modifiers”

--column-statistics

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--column-type-info

Section 8.2.1.19, “LIMIT Query Optimization”
Section 4.5.1.1, “mysql Client Options”

--columns

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--comments

Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--commit

Section 4.5.8, “`mysqlslap` — Load Emulation Client”

--compact

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--compatible

Section 2.11.4, “Changes in MySQL 8.0”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 1.4, “What Is New in MySQL 8.0”

COMPILATION_COMMENT

Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.8, “Server System Variables”

COMPILATION_COMMENT_SERVER

Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.8, “Server System Variables”

--complete-insert

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--compress

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.6, “Connection Compression Control”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 1.4, “What Is New in MySQL 8.0”

--compress-output

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--compression-algorithms

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.6, “Connection Compression Control”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”

--concurrency

Section 4.5.8, “`mysqlslap` — Load Emulation Client”

--conf-base-port

Section 21.3, “Using MySQL Router with InnoDB Cluster”

--config-cache

Section 22.3.3, “NDB Cluster Configuration Files”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--config-dir

Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--config-file

Section 22.2.5, “Initial Startup of NDB Cluster”
Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 22.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--config_from_node

Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”

--configdir

Section 22.3.3, “NDB Cluster Configuration Files”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--configinfo

Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”

--connect

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--connect-delay

Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”

--connect-expired-password

Section 4.5.1.1, “mysql Client Options”
Section 6.2.16, “Server Handling of Expired Passwords”

--connect-retries

Section 22.4.5, “`ndb_mgm` — The NDB Cluster Management Client”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--connect-retry-delay

Section 22.4.5, “`ndb_mgm` — The NDB Cluster Management Client”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--connect-string

Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--connect-timeout

Section 4.5.1.1, “mysql Client Options”

--connection-control

Section 6.4.2.1, “Connection-Control Plugin Installation”

--connection-control-failed-login-attempts

Section 6.4.2.1, “Connection-Control Plugin Installation”

--connection-server-id

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--connection-timeout

Section 22.4.12, “`ndb_error_reporter` — NDB Error-Reporting Utility”

--connections

Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--console

Section 5.4.2.2, “Default Error Log Destination Configuration”
Section 15.16.2, “Enabling InnoDB Monitors”
Section 5.4.2.1, “Error Log Component Configuration”
Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 2.10.1, “Initializing the Data Directory”
Section 15.20, “InnoDB Troubleshooting”
Resetting the Root Password: Windows Systems
Section 5.1.7, “Server Command Options”
Section 2.3.4.6, “Starting MySQL from the Windows Command Line”
Section 2.3.4.5, “Starting the Server for the First Time”

--continue

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

copy-back-and-apply-log

Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

--core-file

Section 29.5.1.4, “Debugging mysqld under gdb”
Section 15.8.3.7, “Excluding Buffer Pool Pages from Core Files”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

core-file

Section 29.5.1.3, “Using WER with PDB to create a Windows crashdump”

--core-file-size

Section 2.5.9, “Managing MySQL Server with systemd”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 5.1.7, “Server Command Options”

--correct-checksum

Section 4.6.4.3, “myisamchk Repair Options”

--count

Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 4.6.3, “myisam_ftdump — Display Full-Text Index information”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”

--create

Section 4.5.8, “mysqlslap — Load Emulation Client”

--create-options

Section 4.5.4, “mysqldump — A Database Backup Program”

--create-schema

Section 4.5.8, “mysqlslap — Load Emulation Client”

--csv

Section 4.5.8, “mysqlslap — Load Emulation Client”

--cxxflags

Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 4.7.1, “mysql_config — Display Options for Compiling Clients”

D

[index top]

-D

Section 10.13, “Adding a Character Set”
Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section B.4.2.16, “Can't initialize character set”
Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 22.2.3.2, “Compiling and Installing NDB Cluster from Source on Windows”
Section 29.5.1.1, “Compiling MySQL for Debugging”
Section 2.9.6, “Configuring SSL Library Support”
Section 29.5.2, “Debugging a MySQL Client”
Section 20.5.2, “Disabling X Plugin”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 4.8.1, “lz4_decompress — Decompress mysqlpump LZ4-Compressed Output”
Section 2.5.9, “Managing MySQL Server with systemd”
Section 4.6.4.3, “myisamchk Repair Options”
Section A.2, “MySQL 8.0 FAQ: Storage Engines”
Section 4.5.1.1, “mysql Client Options”
Chapter 22, *MySQL NDB Cluster 8.0*
Section 22.5.4, “MySQL Server Usage for NDB Cluster”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

Section 15.19.3, “Setting Up the InnoDB memcached Plugin”
Section 16.5, “The ARCHIVE Storage Engine”
Section 16.6, “The BLACKHOLE Storage Engine”
Section 16.9, “The EXAMPLE Storage Engine”
Section 16.8, “The FEDERATED Storage Engine”
Section 29.5.3, “The LOCK_ORDER Tool”
Section 22.1.4, “What is New in NDB Cluster”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”
Section 2.1.1, “Which MySQL Version and Distribution to Install”
Section 4.8.3, “`zlib_decompress` — Decompress mysqlpump ZLIB-Compressed Output”

-d

Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.3, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 22.4.9, “`ndb_desc` — Describe NDB Tables”
Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.15, “`ndb_move_data` — NDB Data Copy Utility”
Section 22.4.22, “`ndb_redo_log_reader` — Check and Print Content of Cluster Redo Log”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”
Section 22.4.26, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster”
Section 22.4.27, “`ndb_show_tables` — Display List of NDB Tables”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 4.6.4.4, “Other myisamchk Options”
Section 5.1.8, “Server System Variables”
Section 6.4.4.4, “Using the keyring_okv KMIP Plugin”

--daemon

Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”

--daemonize

Section 5.1.7, “Server Command Options”

--data-file-length

Section 4.6.4.3, “myisamchk Repair Options”

--database

Section 4.5.1.1, “mysql Client Options”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 22.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 22.4.9, “`ndb_desc` — Describe NDB Tables”
Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 22.4.15, “`ndb_move_data` — NDB Data Copy Utility”
Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”

Section 22.4.27, “ndb_show_tables — Display List of NDB Tables”
Section 22.4.28, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--databases

Section 7.4.5.2, “Copy a Database from one Server to Another”
Creating a Data Snapshot Using mysqldump
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 7.4.5.1, “Making a Copy of a Database”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.2, “Reloading SQL-Format Backups”

--datadir

Section 2.3.4.2, “Creating an Option File”
Section 2.10.1, “Initializing the Data Directory”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 22.5.12.3, “NDB Cluster and MySQL Security Procedures”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.8.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.7, “Server Command Options”
Section 5.8.1, “Setting Up Multiple Data Directories”
Section 5.2, “The MySQL Data Directory”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 4.2.2.2, “Using Option Files”

datadir

Section 2.3.4.2, “Creating an Option File”
Section 2.4.1, “General Notes on Installing MySQL on macOS”
Section 4.3.3, “mysql.server — MySQL Server Startup Script”
Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”
Section 2.3.7, “Windows Platform Restrictions”

--db-workers

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--ddl-rewriter

Section 5.6.5.2, “ddl_rewriter Plugin Options”
Section 5.6.5.1, “Installing or Uninstalling ddl_rewriter”

--debug

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 29.5.1.1, “Compiling MySQL for Debugging”
Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”
Section 4.7.2, “my_print_defaults — Display Options from Option Files”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 2.9.7, “MySQL Source-Configuration Options”

Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 2.3.4.6, “Starting MySQL from the Windows Command Line”
Section 29.5.4, “The DBUG Package”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”

debug

Section 14.1, “Data Dictionary Schema”

--debug-check

Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”

--debug-info

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”

--debug-level

Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”

--debug-sync-timeout

Section 2.9.7, “MySQL Source-Configuration Options”

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

--default-auth

Section 28.7.17, “C API Client Plugin Functions”
Section 2.11.4, “Changes in MySQL 8.0”
Client Plugin Descriptors
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 6.4.1.1, “Native Pluggable Authentication”
Section 6.2.17, “Pluggable Authentication”
Using the Authentication Plugins

--default-authentication-plugin

Section 2.11.4, “Changes in MySQL 8.0”

--default-character-set

Section 6.2.1, “Account User Names and Passwords”
Section 10.15, “Character Set Configuration”
Section 10.5, “Configuring Application Character Set and Collation”
Section 10.4, “Connection Character Sets and Collations”
Section 4.5.1.5, “Executing SQL Statements from a Text File”
Section 13.2.7, “LOAD DATA Syntax”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 5.1.8, “Server System Variables”

--default-parallelism

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--default-storage-engine

Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.7, “Server Command Options”

--default-time-zone

Section 5.1.13, “MySQL Server Time Zone Support”
Section 5.1.7, “Server Command Options”

Section 5.1.8, “Server System Variables”

--default-tmp-storage-engine

Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.7, “Server Command Options”

--default.key_buffer_size

Section 5.1.9.5, “Structured System Variables”

DEFAULT_CHARSET

Section 10.5, “Configuring Application Character Set and Collation”
Section 10.3.2, “Server Character Set and Collation”

DEFAULT_COLLATION

Section 10.5, “Configuring Application Character Set and Collation”
Section 10.3.2, “Server Character Set and Collation”

--defaults-extra-file

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 2.10.1, “Initializing the Data Directory”
Section 21.5, “Known Limitations”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.4.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 26.12.13.2, “Performance Schema variables_info Table”
Section 5.1.7, “Server Command Options”
Section 4.2.2.2, “Using Option Files”

--defaults-file

Section 2.11.4, “Changes in MySQL 8.0”
Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 2.10.1, “Initializing the Data Directory”
Section 15.8.1, “InnoDB Startup Configuration”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.4.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”

Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 26.12.13.2, “Performance Schema variables_info Table”
Resetting the Root Password: Unix and Unix-Like Systems
Resetting the Root Password: Windows Systems
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.8.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.7, “Server Command Options”
Section 5.1.3, “Server Configuration Validation”
Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 5.8.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.3.4.8, “Starting MySQL as a Windows Service”

--defaults-group-suffix

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 4.7.2, “my_print_defaults — Display Options from Option Files”
Section 4.6.4.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.9, “MySQL Environment Variables”
Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.7, “Server Command Options”

--defer-table-indexes

Section 4.5.6, “mysqlpump — A Database Backup Program”

--delay

Section 22.4.2, “ndbinfo_select_all — Select From ndbinfo Tables”

--delay-key-write
Section 8.11.5, “External Locking”

--delay_key_write
Section 5.1.9, “Using System Variables”

delay_key_write
Section 16.2.1, “MyISAM Startup Options”

--delete
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--delete-master-logs
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--delete-orphans
Section 22.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”

--delimiter
Section 4.5.1.1, “mysql Client Options”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”

--des-key-file
Section 5.1.7, “Server Command Options”
Section 1.4, “What Is New in MySQL 8.0”

--descending
Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”

--description
Section 4.6.4.4, “Other myisamchk Options”

--detach
Section 4.5.8, “`mysqlslap` — Load Emulation Client”

--diff-default
Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”

--directory
Section 21.3, “Using MySQL Router with InnoDB Cluster”

--disable
Section 4.2.2.4, “Program Option Modifiers”

--disable-auto-rehash
Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”

Section 4.5.1.1, “mysql Client Options”

--disable-indexes

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Restoring to More Nodes Than the Original

--disable-keys

Section 4.5.4, “mysqldump — A Database Backup Program”

--disable-log-bin

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.3.1, “GTID Format and Storage”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 5.4.4, “The Binary Log”

--disable-named-commands

Section 4.5.1.1, “mysql Client Options”

--disable-plugin_name

Section 5.6.1, “Installing and Uninstalling Plugins”

--disable-ssl

Section 4.2.3, “Command Options for Connecting to the Server”
Section 1.4, “What Is New in MySQL 8.0”

DISABLE_SHARED

Section 1.4, “What Is New in MySQL 8.0”

--disconnect-slave-event-count

Section 17.1.6.3, “Replication Slave Options and Variables”

--disk

Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”

--dont-ignore-systab-0

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--drop-source

Section 22.4.15, “ndb_move_data — NDB Data Copy Utility”

--dry-scp

Section 22.4.12, “ndb_error_reporter — NDB Error-Reporting Utility”

--dump

Section 4.6.3, “myisam_ftdump — Display Full-Text Index information”
Section 22.4.14, “ndb_index_stat — NDB Index Statistics Utility”

--dump-date

Section 4.5.4, “mysqldump — A Database Backup Program”

--dump-file

Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”
Section 22.4.6, “ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”

--dump-slave

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”

dynamic_state

Section 21.3, “Using MySQL Router with InnoDB Cluster”

E

[index top]

-E

Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.4, “mysqldump — A Database Backup Program”

-e

Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 13.2.8, “LOAD XML Syntax”
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”
Section 4.7.2, “my_print_defaults — Display Options from Option Files”
Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 22.4.5, “ndb_mgm — The NDB Cluster Management Client”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 4.6.4.5, “Obtaining Table Information with myisamchk”
Section 22.2.7, “Safe Shutdown and Restart of NDB Cluster”
Section 4.2.2.1, “Using Options on the Command Line”
Section 22.5.3.2, “Using The NDB Cluster Management Client to Create a Backup”

--early-plugin-load

Section 5.6.1, “Installing and Uninstalling Plugins”
Section 2.4.3, “Installing and Using the MySQL Launch Daemon”
Section 6.4.4.1, “Keyring Plugin Installation”
Section 5.1.7, “Server Command Options”
Server Plugin Library and Plugin Descriptors
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 6.4.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”
Section 6.4.4.3, “Using the keyring_encrypted_file Keyring Plugin”
Section 6.4.4.2, “Using the keyring_file File-Based Plugin”
Section 6.4.4.4, “Using the keyring_okv KMIP Plugin”

Section 29.2.4.12, “Writing Keyring Plugins”

early-plugin-load

Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”

--embedded

Section 1.4, “What Is New in MySQL 8.0”

--embedded-libs

Section 1.4, “What Is New in MySQL 8.0”

--embedded-server

Section 1.4, “What Is New in MySQL 8.0”

--enable-cleartext-plugin

Section 6.4.1.4, “Client-Side Cleartext Pluggable Authentication”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 6.4.1.5, “PAM Pluggable Authentication”

--enable-plugin_name

Section 5.6.1, “Installing and Uninstalling Plugins”

--enable-ssl

Section 1.4, “What Is New in MySQL 8.0”

ENABLE_DEBUG_SYNC

Section 2.9.7, “MySQL Source-Configuration Options”

enabled

Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”

ENABLED_LOCAL_INFILE

Section 2.9.7, “MySQL Source-Configuration Options”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”

--end-page

Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”

--enforce-gtid-consistency

Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”

--engine

Section 4.5.8, “`mysqlslap` — Load Emulation Client”

--env
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

--errins-delay
Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--errins-type
Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--error-insert
Section 22.4.15, “ndb_move_data — NDB Data Copy Utility”

--event-scheduler
Section 24.4.2, “Event Scheduler Configuration”

event-scheduler
Section 24.4.2, “Event Scheduler Configuration”

--events
Section 14.7, “Data Dictionary Usage Differences”
Section 7.4.5.3, “Dumping Stored Programs”
Section 7.4.5.4, “Dumping Table Definitions and Content Separately”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--example
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

--exclude-*
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--exclude-databases
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--exclude-events
Section 4.5.6, “mysqlpump — A Database Backup Program”

--exclude-gtids
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--exclude-intermediate-sql-tables
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--exclude-missing-columns
Section 22.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--exclude-missing-tables

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--exclude-routines

Section 4.5.6, “mysqlpump — A Database Backup Program”

--exclude-tables

Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--exclude-triggers

Section 4.5.6, “mysqlpump — A Database Backup Program”

--exclude-users

Section 4.5.6, “mysqlpump — A Database Backup Program”

--excludedbs

Section 22.4.28, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--excludetables

Section 22.4.28, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--execute

Section 4.5.1.3, “mysql Client Logging”
Section 4.5.1.1, “mysql Client Options”
Section 22.4.5, “ndb_mgm — The NDB Cluster Management Client”
Section 4.2.2.1, “Using Options on the Command Line”
Section 22.5.3.2, “Using The NDB Cluster Management Client to Create a Backup”

--exit-info

Section 5.1.7, “Server Command Options”

--extend-check

Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.4.3, “myisamchk Repair Options”

--extended

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--extended-insert

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”

--external-locking

Section 8.11.5, “External Locking”
Section 16.2.1, “MyISAM Startup Options”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

--extra-file

Section 4.7.2, “my_print_defaults — Display Options from Option Files”

--extra-node-info

Section 22.4.9, “ndb_desc — Describe NDB Tables”

--extra-partition-info

Section 22.4.9, “ndb_desc — Describe NDB Tables”
Section 22.5.10.5, “The ndbinfo cluster_operations Table”
Section 22.5.10.34, “The ndbinfo server_operations Table”

F

[index top]

-F

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.2, “mysql Client Commands”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.8, “mysqlslap — Load Emulation Client”

-f

Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 22.2.5, “Initial Startup of NDB Cluster”
Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 22.4.22, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 29.5.1.5, “Using a Stack Trace”

--fast

Section 4.6.4.2, “myisamchk Check Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--federated

Section 16.8, “The FEDERATED Storage Engine”

--fields

Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

--fields-enclosed-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 22.4.13, “ndb_import — Import CSV Data Into NDB”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--fields-escaped-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--fields-optionally-enclosed-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 22.4.13, “ndb_import — Import CSV Data Into NDB”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--fields-terminated-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 22.4.13, “ndb_import — Import CSV Data Into NDB”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--fields-xxx

Section 4.5.4, “mysqldump — A Database Backup Program”

--fix-db-names

Section 1.4, “What Is New in MySQL 8.0”

--fix-table-names

Section 1.4, “What Is New in MySQL 8.0”

--flush

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”

--flush-logs

Section 7.3.1, “Establishing a Backup Policy”
Section 5.4, “MySQL Server Logs”
Section 4.5.4, “mysqldump — A Database Backup Program”

--flush-privileges

Section 4.5.4, “mysqldump — A Database Backup Program”

--flush_time
Section 29.1.1, “MySQL Threads”

--force
Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 22.6.4, “NDB Cluster Replication Schema and Tables”
Section 29.1.2, “The MySQL Test Suite”
Section 3.5, “Using mysql in Batch Mode”
Section 2.11.3, “What the MySQL Upgrade Process Upgrades”

--force-if-open
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--force-read
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--foreground
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”

--format
Section 22.4.28, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

FPROFILE_GENERATE
Section 2.9.7, “MySQL Source-Configuration Options”

FPROFILE_USE
Section 2.9.7, “MySQL Source-Configuration Options”

--fs
Section 22.4.12, “`ndb_error_reporter` — NDB Error-Reporting Utility”

G

[index top]

-G
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”

-g
Section 29.5.1.1, “Compiling MySQL for Debugging”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”

--gci

Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”

--gci64

Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”

--gdb

Section 29.5.1.4, “Debugging mysqld under gdb”
Section 13.7.8.8, “RESTART Syntax”
Section 5.1.7, “Server Command Options”

--general-log

Section 4.2.2.1, “Using Options on the Command Line”
Section 5.1.9, “Using System Variables”

--general_log

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.4.3, “The General Query Log”
Section 4.2.2.1, “Using Options on the Command Line”
Section 5.1.9, “Using System Variables”

--general_log_file

Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.1.8, “Server System Variables”
Section 5.4.3, “The General Query Log”

--get-server-public-key

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”

--graph

Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”

--gtid-executed-compression-period

Section 17.1.6.5, “Global Transaction ID Options and Variables”

H

[index top]

-H

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.6.4.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”

-h

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.6.3, “myisam_ftdump — Display Full-Text Index information”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.18, “ndb_print_file — Print NDB Disk Data File Contents”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 5.1.7, “Server Command Options”
Section 1.2, “Typographical and Syntax Conventions”
Section 4.2.2.1, “Using Options on the Command Line”

HAVE_CRYPT

Section 1.4, “What Is New in MySQL 8.0”

--header

Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”

--header_file

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--HELP

Section 4.6.4.1, “myisamchk General Options”

--help

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 4.7.2, “my_print_defaults — Display Options from Option Files”

Section 4.6.3, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.6, “ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.16, “ndb_perror — Obtain NDB Error Message Information”
Section 22.4.18, “ndb_print_file — Print NDB Disk Data File Contents”
Section 22.4.22, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 4.1, “Overview of MySQL Programs”
Section 4.8.2, “perror — Display MySQL Error Message Information”
Section 5.1.7, “Server Command Options”
Section 2.10.3, “Testing the Server”
Section 1.3.2, “The Main Features of MySQL”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Chapter 3, *Tutorial*
Section 4.2.2.2, “Using Option Files”
Section 4.2.2.1, “Using Options on the Command Line”

--hex

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--hex-blob

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”

--hexdump

Section 4.6.8.1, “mysqlbinlog Hex Dump Format”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--histignore

Section 4.5.1.3, “mysql Client Logging”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”

--host

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 2.10.1, “Initializing the Data Directory”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”
Section 4.2.2.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 1.2, “Typographical and Syntax Conventions”
Section 5.8.4, “Using Client Programs in a Multiple-Server Environment”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 4.2.2.2, “Using Option Files”
Section 4.2.2.1, “Using Options on the Command Line”
Section 20.5.5.2, “X Plugin Options and System Variables”

host

Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

--hostname

Section 22.4.28, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

--html

Section 4.5.1.1, “mysql Client Options”

I

[index top]

-I

Section 28.7.4.1, “Building C API Client Programs”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 4.6.5, “`myisamlog` — Display MyISAM Log File Contents”
Section 4.8.2, “`perror` — Display MySQL Error Message Information”
Section 5.1.7, “Server Command Options”

-i

Section 22.5.2, “Commands in the NDB Cluster Management Client”

Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”

--i-am-a-dummy

Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”

--id

Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”

--idempotent

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 5.1.8, “Server System Variables”

--idlesleep

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--idlespin

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--ignore

Section 4.5.5, “mysqlimport — A Data Import Program”

--ignore-db-dir

Section 2.11.5, “Preparing Your Installation for Upgrade”
Section 1.4, “What Is New in MySQL 8.0”

--ignore-error

Section 4.5.4, “mysqldump — A Database Backup Program”

--ignore-lines

Section 4.5.5, “mysqlimport — A Data Import Program”
Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--ignore-spaces

Section 4.5.1.1, “mysql Client Options”

--ignore-table

Creating a Data Snapshot Using mysqldump
Section 4.5.4, “mysqldump — A Database Backup Program”

--in_file

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--include

Section 4.7.1, “`mysql_config` — Display Options for Compiling Clients”

--include-*

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--include-databases

Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--include-events

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--include-gtids

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--include-master-host-port

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--include-routines

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--include-tables

Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--include-triggers

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--include-users

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--info

Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 4.8.2, “`perror` — Display MySQL Error Message Information”

--information

Section 4.6.4.2, “myisamchk Check Options”

--init-command

Section 4.5.1.1, “mysql Client Options”

--init_connect

Section 10.5, “Configuring Application Character Set and Collation”

--initial

Section 22.5.15.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 22.5.2, “Commands in the NDB Cluster Management Client”

Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 22.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”
Section 22.3.3, “NDB Cluster Configuration Files”
Section 22.5.13.3, “NDB Cluster Disk Data Storage Requirements”
Section 22.3.3.12, “NDB Cluster Shared-Memory Connections”
Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 22.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”
Section 22.3.2, “Overview of NDB Cluster Configuration Parameters, Options, and Variables”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Restoring to Fewer Nodes Than the Original
Section 22.5.1, “Summary of NDB Cluster Start Phases”
Section 22.2.8, “Upgrading and Downgrading NDB Cluster”
Section 22.1.4, “What is New in NDB Cluster”

--initial-start

Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”

--initialize

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 2.11.4, “Changes in MySQL 8.0”
Section 2.3.4.2, “Creating an Option File”
Section 2.10.1, “Initializing the Data Directory”
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
MySQL Server Options for NDB Cluster
Chapter 27, *MySQL sys Schema*
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 5.4.4, “The Binary Log”
Section 1.4, “What Is New in MySQL 8.0”

--initialize-insecure

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 2.11.4, “Changes in MySQL 8.0”
Section 2.3.4.2, “Creating an Option File”
Section 2.10.1, “Initializing the Data Directory”
Section 22.2.2.1, “Installing an NDB Cluster Binary Release on Linux”
Chapter 27, *MySQL sys Schema*
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 5.4.4, “The Binary Log”
Section 1.4, “What Is New in MySQL 8.0”

--innodb

Section 15.13, “InnoDB Startup Options and System Variables”

--innodb-adaptive-hash-index

Section 15.13, “InnoDB Startup Options and System Variables”

--innodb-file-per-table
Section 5.1.7, “Server Command Options”

innodb-file-per-table
Section 5.1.7, “Server Command Options”

--innodb-rollback-on-timeout
Section 15.20.4, “InnoDB Error Handling”
Section 15.13, “InnoDB Startup Options and System Variables”

--innodb-status-file
Section 15.16.2, “Enabling InnoDB Monitors”
Section 15.13, “InnoDB Startup Options and System Variables”

--innodb-xxx
Section 5.1.7, “Server Command Options”

--innodb_file_per_table
Section 15.6.3.2, “File-Per-Table Tablespaces”

innodb_file_per_table
Creating a Data Snapshot Using Raw Data Files

--input-type
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--input-workers
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--insert-ignore
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--install
Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 5.1.7, “Server Command Options”
Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.4.8, “Starting MySQL as a Windows Service”

--install-manual
Section 5.1.7, “Server Command Options”
Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.4.8, “Starting MySQL as a Windows Service”

INSTALL_LAYOUT
Section 6.4.4.11, “Keyring System Variables”
Section 2.9.7, “MySQL Source-Configuration Options”

Section 5.1.8, “Server System Variables”

INSTALL_LIBDIR

Section 2.9.7, “MySQL Source-Configuration Options”

INSTALL_MYSQLKEYRINGDIR

Section 6.4.4.11, “Keyring System Variables”

INSTALL_PKGCONFIGDIR

Section 28.7.4.2, “Building C API Client Programs Using pkg-config”

INSTALL_SCRIPTDIR

Section 1.4, “What Is New in MySQL 8.0”

INSTALL_SECURE_FILE_PRIV_EMBEDDEDDIR

Section 1.4, “What Is New in MySQL 8.0”

INSTALL_SECURE_FILE_PRIVDIR

Section 5.1.8, “Server System Variables”

--interactive

Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--iterations

Section 4.5.8, “`mysqlslap` — Load Emulation Client”

J

[index top]

-j

Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--join

Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”

K

[index top]

-K

Section 4.5.4, “`mysqldump` — A Database Backup Program”

-k

Section 4.6.4.3, “myisamchk Repair Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”

--keep-state
Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--keep_files_on_create
Section 13.1.20, “CREATE TABLE Syntax”

--key-file
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”

--keyring-migration-destination
Section 6.4.4.10, “Keyring Command Options”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”

--keyring-migration-host
Section 6.4.4.10, “Keyring Command Options”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”

--keyring-migration-password
Section 6.4.4.10, “Keyring Command Options”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”

--keyring-migration-port
Section 6.4.4.10, “Keyring Command Options”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”

--keyring-migration-socket
Section 6.4.4.10, “Keyring Command Options”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”

--keyring-migration-source
Section 6.4.4.10, “Keyring Command Options”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”

--keyring-migration-user
Section 6.4.4.10, “Keyring Command Options”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”

--keys
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”

--keys-used
Section 4.6.4.3, “myisamchk Repair Options”

L

[index top]

-L
Section 28.7.4.1, “Building C API Client Programs”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.5, “mysqlimport — A Data Import Program”

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 2.13.3, “Problems Using the Perl DBI/DBD Interface”

-I

Section 28.7.4.1, “Building C API Client Programs”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 4.7.2, “my_print_defaults — Display Options from Option Files”
Section 4.6.3, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.4.3, “myisamchk Repair Options”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 22.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 22.4.22, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”
Section 22.4.27, “ndb_show_tables — Display List of NDB Tables”
Section 22.4.2, “ndbinfo_select_all — Select From ndbinfo Tables”

--language

Section 5.1.7, “Server Command Options”

--large-pages

Section 8.12.3.2, “Enabling Large Page Support”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

--lc-messages

Section 5.1.7, “Server Command Options”

--lc-messages-dir

Section 5.1.7, “Server Command Options”

--ledir

Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

--length

Section 4.6.3, “myisam_ftdump — Display Full-Text Index information”

--libmysqld-libs

Section 1.4, “What Is New in MySQL 8.0”

--libs

Section 4.7.1, “mysql_config — Display Options for Compiling Clients”

--libs_r

Section 4.7.1, “mysql_config — Display Options for Compiling Clients”

--line-numbers

Section 4.5.1.1, “mysql Client Options”

--lines-terminated-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

LINK_RANDOMIZE

Section 2.9.7, “MySQL Source-Configuration Options”

--loadqueries

Section 22.4.28, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

--local

Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”

--local-infile

Section 13.2.8, “LOAD XML Syntax”
Section 4.5.1.1, “mysql Client Options”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”

--local-load

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--local-service

Section 5.1.7, “Server Command Options”
Section 2.3.4.8, “Starting MySQL as a Windows Service”

--lock

Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”

--lock-all-tables

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--lock-order

Section 29.5.3, “The LOCK_ORDER Tool”

--lock-tables

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”

--log

Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

--log-bin

Section 7.3.3, “Backup Strategy Summary”
Section 17.1.6.4, “Binary Logging Options and Variables”

Section 7.2, “Database Backup Methods”
Section 7.3.1, “Establishing a Backup Policy”
Section B.4.7, “Known Issues in MySQL”
Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”
Section 22.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 17.1.2.1, “Setting the Replication Master Configuration”
Section 17.4.8, “Switching Masters During Failover”
Section 5.4.4, “The Binary Log”
Section 7.3.2, “Using Backups for Recovery”
Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”

--log-bin-index

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 22.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.4.4, “The Binary Log”

--log-error

Section 5.4.2.2, “Default Error Log Destination Configuration”
Section 5.4.2.1, “Error Log Component Configuration”
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.2.2.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.1.7, “Server Command Options”
Section 5.4.7, “Server Log Maintenance”
Section 2.3.4.6, “Starting MySQL from the Windows Command Line”
Section 2.3.4.5, “Starting the Server for the First Time”

--log-error-file

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--log-isam

Section 4.6.5, “`myisamlog` — Display MyISAM Log File Contents”
Section 5.1.7, “Server Command Options”

--log-level

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 21.2.4, “Production Deployment of InnoDB Cluster”

--log-name

Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--log-raw

Section 6.1.2.3, “Passwords and Logging”
Section 5.1.7, “Server Command Options”
Section 5.4.3, “The General Query Log”

Section 29.2.1, “Types of Plugins”

--log-short-format

Section 5.1.7, “Server Command Options”
Section 5.4.5, “The Slow Query Log”

--log-slave-updates

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.5.3, “Disabling GTID Transactions Online”
Section 17.5.5, “How to Report Replication Bugs or Problems”
Section 17.4.7, “Improving Replication Performance”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.1.2.2, “Setting the Replication Slave Configuration”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Section 17.4.8, “Switching Masters During Failover”
Section 5.4.4, “The Binary Log”

--log-tc

Section 5.1.7, “Server Command Options”

--log-tc-size

Section 5.1.7, “Server Command Options”
Section 5.1.10, “Server Status Variables”

--log-warnings

Section 5.1.7, “Server Command Options”
Section 1.4, “What Is New in MySQL 8.0”

--log_output

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”

--log_timestamps

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--logbuffer-size

Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”

--login-path

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”

Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 4.2.2.2, “Using Option Files”

--loops

Section 22.4.14, “ndb_index_stat — NDB Index Statistics Utility”
Section 22.4.27, “ndb_show_tables — Display List of NDB Tables”
Section 22.4.2, “ndbinfo_select_all — Select From ndbinfo Tables”

--loose

Section 4.2.2.4, “Program Option Modifiers”

--loose-opt_name

Section 4.2.2.2, “Using Option Files”

--lossy-conversions

Section 22.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--low-priority

Section 4.5.5, “mysqlimport — A Data Import Program”

--low-priority-updates

Section 8.11.3, “Concurrent Inserts”
Section 13.2.6, “INSERT Syntax”
Section 8.11.2, “Table Locking Issues”

--lower-case-table-names

Section 9.2.3, “Identifier Case Sensitivity”

M

[index top]

-M

Section 4.5.4, “mysqldump — A Database Backup Program”

-m

Section 4.6.4.2, “myisamchk Check Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 22.4.22, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Restoring to More Nodes Than the Original

--malloc-lib

Section 2.5.9, “Managing MySQL Server with systemd”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

--master-data

Creating a Data Snapshot Using mysqldump
Section 7.3.1, “Establishing a Backup Policy”
Section 5.4, “MySQL Server Logs”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--master-info-file

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.4.2, “Slave Status Logs”
Section 1.4, “What Is New in MySQL 8.0”

--master-info-repository

Configuring Multi-Source Replication
Section 17.1.6.3, “Replication Slave Options and Variables”
Setting Up Replication with Existing Data
Section 17.2.4.2, “Slave Status Logs”
Section 17.2.3.3, “Startup Options and Replication Channels”
Section 1.4, “What Is New in MySQL 8.0”

--master-retry-count

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”

--master-verify-checksum

Section 17.1.6.4, “Binary Logging Options and Variables”

--max

Section 4.2.2.5, “Using Options to Set Program Variables”

--max-allowed-packet

Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--max-binlog-dump-events

Section 17.1.6.4, “Binary Logging Options and Variables”

--max-binlog-size

Section 17.1.6.3, “Replication Slave Options and Variables”

--max-join-size

Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”

--max-record-length

Section 4.6.4.3, “myisamchk Repair Options”
Section 13.7.3.5, “REPAIR TABLE Syntax”

--max-relay-log-size

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.3.3, “Startup Options and Replication Channels”

--max-rows

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--max-seeks-for-key

Section 8.2.1.23, “Avoiding Full Table Scans”
Section B.4.5, “Optimizer-Related Issues”

--max_a

Section 4.2.2.5, “Using Options to Set Program Variables”

--maximum

Section 4.2.2.4, “Program Option Modifiers”

--maximum-back_log

Section 4.2.2.4, “Program Option Modifiers”

--maximum-innodb-log-file-size

Section 5.1.9, “Using System Variables”

--maximum-max_heap_table_size

Section 4.2.2.4, “Program Option Modifiers”

--maximum-var_name

Section 5.1.7, “Server Command Options”
Section 5.1.9, “Using System Variables”

--measured-load

Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”

--medium-check

Section 4.6.4.2, “myisamchk Check Options”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--memlock

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 15.6.3.1, “The System Tablespace”

--monitor

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--mount

Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

--my-plugin

Section 5.6.1, “Installing and Uninstalling Plugins”

--my_plugin
Section 5.6.1, “Installing and Uninstalling Plugins”

--mycnf
Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--myisam-block-size
Section 8.10.2.5, “Key Cache Block Size”
Section 5.1.7, “Server Command Options”

--myisam_sort_buffer_size
Section 4.6.4.6, “myisamchk Memory Usage”

MYSQL_ALLOW_EMPTY_PASSWORD
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_DATABASE
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_LOG_CONSOLE
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_MAINTAINER_MODE
Section 2.9.8, “Dealing with Problems Compiling MySQL”

MYSQL_ONETIME_PASSWORD
Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_PASSWORD
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_RANDOM_ROOT_PASSWORD
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_ROOT_HOST
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_ROOT_PASSWORD
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_TCP_PORT
Section 2.9.5, “Installing MySQL Using a Development Source Tree”
Section 2.9.7, “MySQL Source-Configuration Options”

MYSQL_UNIX_ADDR
Section B.4.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 2.9.5, “Installing MySQL Using a Development Source Tree”
Section 2.9.7, “MySQL Source-Configuration Options”

Section 20.5.5.2, “X Plugin Options and System Variables”

MYSQL_USER

Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

--mysqladmin

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

--mysqld

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--mysqld-safe-log-timestamps

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--mysqld-version

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--mysqlx

Section 20.5.2, “Disabling X Plugin”
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx

Section 20.5.2, “Disabling X Plugin”

MYSQLX_UNIX_ADDR

Section 20.5.5.2, “X Plugin Options and System Variables”

N

[index top]

-N

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 22.4.26, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster”

-n

Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 22.4.9, “`ndb_desc` — Describe NDB Tables”
Section 22.4.22, “`ndb_redo_log_reader` — Check and Print Content of Cluster Redo Log”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.4.26, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster”
Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”

Section 22.4.30, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”
Section 22.4.1, “ndbd — The NDB Cluster Data Node Daemon”

--name

Section 2.5.6.1, “Basic Steps for MySQL Server Deployment with Docker”

--name_file

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--named-commands

Section 4.5.1.1, “mysql Client Options”

--ndb

Section 22.4.16, “ndb_perror — Obtain NDB Error Message Information”
Section 4.8.2, “perror — Display MySQL Error Message Information”
Section 22.5.10.21, “The ndbinfo error_messages Table”
Section 1.4, “What Is New in MySQL 8.0”
Section 22.1.4, “What is New in NDB Cluster”

--ndb-allow-copying-alter-table

MySQL Server Options for NDB Cluster

--ndb-batch-size

MySQL Server Options for NDB Cluster

--ndb-blob-read-batch-bytes

MySQL Server Options for NDB Cluster

--ndb-blob-write-batch-bytes

MySQL Server Options for NDB Cluster

--ndb-cluster-connection-pool

MySQL Server Options for NDB Cluster

--ndb-cluster-connection-pool-nodeids

MySQL Server Options for NDB Cluster

--ndb-connectstring

Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
MySQL Server Options for NDB Cluster
Section 22.5.4, “MySQL Server Usage for NDB Cluster”
Section 22.5.12.2, “NDB Cluster and MySQL Privileges”
Section 22.1.1, “NDB Cluster Core Concepts”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 22.6.5, “Preparing the NDB Cluster for Replication”
Section 22.3.3.2, “Recommended Starting Configuration for NDB Cluster”
Restoring to Fewer Nodes Than the Original
Restoring to More Nodes Than the Original

--ndb-default-column-format

MySQL Server Options for NDB Cluster

--ndb-deferred-constraints

MySQL Server Options for NDB Cluster

--ndb-distribution

MySQL Server Options for NDB Cluster

--ndb-force-send

Section 22.3.3.2, “Recommended Starting Configuration for NDB Cluster”

--ndb-index-stat-enable

Section 22.3.3.2, “Recommended Starting Configuration for NDB Cluster”

--ndb-log-apply-status

MySQL Server Options for NDB Cluster
NDB Cluster System Variables

--ndb-log-empty-epochs

MySQL Server Options for NDB Cluster
Section 22.6.4, “NDB Cluster Replication Schema and Tables”

--ndb-log-empty-update

MySQL Server Options for NDB Cluster

--ndb-log-exclusive-reads

MySQL Server Options for NDB Cluster

--ndb-log-orig

MySQL Server Options for NDB Cluster
Section 22.6.4, “NDB Cluster Replication Schema and Tables”
NDB Cluster System Variables

--ndb-log-transaction-id

MySQL Server Options for NDB Cluster
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster System Variables

--ndb-log-update-as-write

Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

--ndb-log-update-minimal

MySQL Server Options for NDB Cluster

--ndb-log-updated-only

Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

--ndb-mgmd-host

MySQL Server Options for NDB Cluster

Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--ndb-nodegroup-map

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--ndb-nodeid

MySQL Server Options for NDB Cluster
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--ndb-optimization-delay

MySQL Server Options for NDB Cluster
Section 13.7.3.4, “OPTIMIZE TABLE Syntax”

--ndb-optimized-node-selection

Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--ndb-recv-thread-activation-threshold

MySQL Server Options for NDB Cluster
NDB Cluster System Variables

--ndb-recv-thread-cpu-mask

MySQL Server Options for NDB Cluster
NDB Cluster System Variables

--ndb-schema-dist-timeout

MySQL Server Options for NDB Cluster
Section 22.1.4, “What is New in NDB Cluster”

--ndb-transid-mysql-connection-map

Section 25.13, “The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table”

ndb-transid-mysql-connection-map

MySQL Server Options for NDB Cluster

--ndb-use-exact-count

Section 22.3.3.2, “Recommended Starting Configuration for NDB Cluster”

--ndb-wait-connected

MySQL Server Options for NDB Cluster

--ndb-wait-setup

MySQL Server Options for NDB Cluster

--ndbcluster

Section 22.3, “Configuration of NDB Cluster”

Section 22.2.3.1, “Installing NDB Cluster on Windows from a Binary Release”
MySQL Server Options for NDB Cluster
Section 22.5.4, “MySQL Server Usage for NDB Cluster”
Section 22.5.12.2, “NDB Cluster and MySQL Privileges”
Section 22.1.1, “NDB Cluster Core Concepts”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 22.3.3.2, “Recommended Starting Configuration for NDB Cluster”
Section 5.1.7, “Server Command Options”
Section 13.7.7.16, “SHOW ENGINES Syntax”
Section 25.9, “The INFORMATION_SCHEMA ENGINES Table”

--ndbinfo

MySQL Server Options for NDB Cluster

--ndbinfo-database

NDB Cluster System Variables

--ndbinfo-table-prefix

NDB Cluster System Variables

--net-buffer-length

Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

net_retry_count

Section 17.2.2, “Replication Implementation Details”

net_write_timeout

Section 17.2.2, “Replication Implementation Details”

--network

Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”

--network-timeout

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--nice

Section 2.5.9, “Managing MySQL Server with systemd”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--no-asynch

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--no-auto-rehash

Section 4.5.1.1, “mysql Client Options”

--no-autocommit

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--no-beep

Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

--no-binlog

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--no-browser

Section 22.4.26, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster”

--no-check

Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”

--no-contact

Section 22.4.30, “`ndb_waiter` — Wait for NDB Cluster to Reach a Given Status”

--no-create-db

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--no-create-info

Section 7.4.5.4, “Dumping Table Definitions and Content Separately”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--no-data

Section 7.4.5.4, “Dumping Table Definitions and Content Separately”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--no-dd-upgrade

Section 14.1, “Data Dictionary Schema”
Section 5.1.7, “Server Command Options”
Section 1.4, “What Is New in MySQL 8.0”
Section 2.11.3, “What the MySQL Upgrade Process Upgrades”

--no-defaults

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.4.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.9.3, “Persisted System Variables”
Section 5.1.7, “Server Command Options”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 4.2.2.2, “Using Option Files”

--no-drop

Section 4.5.8, “mysqlslap — Load Emulation Client”

--no-hint

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--no-history-logging

Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”

--no-log

Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

--no-monitor

Section 2.3, “Installing MySQL on Microsoft Windows”
Section 13.7.8.8, “RESTART Syntax”
Section 5.1.7, “Server Command Options”

--no-nodeid-checks

Section 22.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”

--no-restore-disk-objects

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--no-set-names

Section 4.5.4, “mysqldump — A Database Backup Program”

--no-symlinks

Section 4.6.4.3, “myisamchk Repair Options”

--no-tablespaces

Section 4.5.4, “mysqldump — A Database Backup Program”

--no-upgrade

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--nodaemon

Section 22.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 22.4.1, “ndbd — The NDB Cluster Data Node Daemon”

--nodata
Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”

--node-id
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”

--nodeid
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Restoring to Fewer Nodes Than the Original
Section 22.1.4, “What is New in NDB Cluster”

--nodes
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

--nostart
Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 22.4.1, “ndbd — The NDB Cluster Data Node Daemon”

--not-started
Section 22.4.30, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”

--nowait-nodes
Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 22.4.30, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”
Section 22.4.1, “ndbd — The NDB Cluster Data Node Daemon”

--number-char-cols
Section 4.5.8, “mysqlslap — Load Emulation Client”

--number-int-cols
Section 4.5.8, “mysqlslap — Load Emulation Client”

--number-of-queries
Section 4.5.8, “mysqlslap — Load Emulation Client”

O

[index top]

-O
Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 2.9.7, “MySQL Source-Configuration Options”

-o
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 8.12.1, “Optimizing Disk I/O”

--offset

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--old-style-user-limits

Section 5.1.7, “Server Command Options”
Section 6.2.20, “Setting Account Resource Limits”

old_passwords

Section 2.11.4, “Changes in MySQL 8.0”

--oldpackage

Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”

ON

Section 3.3.4.9, “Using More Than one Table”

--one-database

Section 4.5.1.1, “mysql Client Options”

--only-print

Section 4.5.8, “mysqlslap — Load Emulation Client”

--opbatch

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--opbytes

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--open-files-limit

Section B.4.2.17, “File Not Found and Similar Errors”
Section 2.5.9, “Managing MySQL Server with systemd”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 5.1.8, “Server System Variables”

--opt

Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 4.5.4, “mysqldump — A Database Backup Program”

--opt_name

Section 4.2.2.2, “Using Option Files”

--optimize

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--optimizer-switch

Section 22.3.3.2, “Recommended Starting Configuration for NDB Cluster”

options

Section 12.16.4, “Functions That Create Geometry Values from WKB Values”
Section 12.16.3, “Functions That Create Geometry Values from WKT Values”
Section 12.16.6, “Geometry Format Conversion Functions”

--order

Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”

--order-by-primary

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--os-load

Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”

--out_dir

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

--out_file

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

--output-type

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--output-workers

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

P

[index top]

-P

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 5.1.7, “Server Command Options”

-p

Section 6.2.1, “Account User Names and Passwords”
Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 4.2.1, “Invoking MySQL Programs”
Section 23.2.5, “KEY Partitioning”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.9, “ndb_desc — Describe NDB Tables”
Section 22.4.22, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”
Section 22.4.27, “ndb_show_tables — Display List of NDB Tables”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section B.4.2.4, “Password Fails When Entered Interactively”
Section 2.3.4.8, “Starting MySQL as a Windows Service”
Section 2.3.4.6, “Starting MySQL from the Windows Command Line”
Section 2.3.4.9, “Testing The MySQL Installation”
Section 2.10.3, “Testing the Server”
Section 22.5.10.5, “The ndbinfo cluster_operations Table”
Section 22.5.10.34, “The ndbinfo server_operations Table”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 2.11, “Upgrading MySQL”
Section 2.11.10, “Upgrading MySQL on Windows”
Section 4.2.2.1, “Using Options on the Command Line”
Section 2.3.6, “Windows Postinstallation Procedures”

--page

Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”

--page-type-dump

Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”

--page-type-summary

Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”

--pagecnt

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--pager

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

--pagesize

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--parallel-recover

Section 4.6.4.3, “myisamchk Repair Options”

--parallel-schemas

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--parallelism

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

parallelism

Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”

--parsable

Section 22.4.27, “`ndb_show_tables` — Display List of NDB Tables”

--partition

Section 1.4, “What Is New in MySQL 8.0”

--passwd

Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”

--password

Section 6.2.1, “Account User Names and Passwords”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 7.3, “Example Backup and Recovery Strategy”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section B.4.2.4, “Password Fails When Entered Interactively”
Restoring to More Nodes Than the Original
Section 6.4.1.10, “Test Pluggable Authentication”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 4.2.2.1, “Using Options on the Command Line”

password

Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

--performance-schema-consumer-consumer_name

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-stages-current

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-stages-history

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-stages-history-long

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-statements-current

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-statements-history

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-statements-history-long

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-transactions-current

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-transactions-history

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-transactions-history-long

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-waits-current

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-waits-history

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-events-waits-history-long

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-global-instrumentation

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-statements-digest

Section 26.14, “Performance Schema Command Options”

--performance-schema-consumer-thread-instrumentation

Section 26.14, “Performance Schema Command Options”

--performance-schema-instrument

Section 26.14, “Performance Schema Command Options”
Section 26.3, “Performance Schema Startup Configuration”

--performance-schema-xxx

Section 5.1.7, “Server Command Options”

--performance_schema_max_mutex_classes

Section 26.7, “Performance Schema Status Monitoring”

--performance_schema_max_mutex_instances

Section 26.7, “Performance Schema Status Monitoring”

--pid-file

Section 5.4.2.2, “Default Error Log Destination Configuration”
Section 5.4.2.1, “Error Log Component Configuration”
Section 2.5.9, “Managing MySQL Server with systemd”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.1.7, “Server Command Options”

pid-file

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

--pipe

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”

Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 2.3.4.9, “Testing The MySQL Installation”

--plugin

Section 5.1.7, “Server Command Options”

--plugin-dir

Section 28.7.17, “C API Client Plugin Functions”
Client Plugin Descriptors
Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 4.5.1.1, “mysql Client Options”
Section 28.7.17.3, “mysql_load_plugin()”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 6.2.17, “Pluggable Authentication”
Using the Authentication Plugins
Using Your Own Protocol Trace Plugins

--plugin-innodb-file-per-table

Section 5.1.7, “Server Command Options”

--plugin-load

Section 6.4.5.9, “Audit Log Reference”
Section 5.6.5.2, “ddl_rewriter Plugin Options”
Section 13.7.4.4, “INSTALL PLUGIN Syntax”
Section 5.6.1, “Installing and Uninstalling Plugins”
Section 6.4.4.10, “Keyring Command Options”
Section 6.4.4.1, “Keyring Plugin Installation”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 29.2.3, “Plugin API Components”
Section 29.2.4.2, “Plugin Data Structures”
Section 5.1.7, “Server Command Options”
Server Plugin Library and Plugin Descriptors
Section 29.2, “The MySQL Plugin API”
Section 6.4.3.3, “Transitioning to the Password Validation Component”
Using the Authentication Plugins
Section 1.4, “What Is New in MySQL 8.0”

--plugin-load-add

Section 6.4.5.9, “Audit Log Reference”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 5.6.5.2, “ddl_rewriter Plugin Options”

Section 5.6.1, “Installing and Uninstalling Plugins”
Section 5.6.7.1, “Installing the Clone Plugin”
Section 6.4.4.1, “Keyring Plugin Installation”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 6.4.1.8, “No-Login Pluggable Authentication”
Section 6.4.1.5, “PAM Pluggable Authentication”
Section 6.4.3.2, “Password Validation Options and Variables”
Section 5.1.7, “Server Command Options”
Server Plugin Library and Plugin Descriptors
Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”
Section 6.4.1.10, “Test Pluggable Authentication”
Section 5.6.3.2, “Thread Pool Installation”
Section 6.4.3.3, “Transitioning to the Password Validation Component”
Section 1.4, “What Is New in MySQL 8.0”
Section 6.4.1.6, “Windows Pluggable Authentication”

plugin-load-add

Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 5.6.7.1, “Installing the Clone Plugin”

--plugin-sql-mode

Section 5.1.7, “Server Command Options”

--plugin-xxx

Section 5.1.7, “Server Command Options”

--plugin_dir

Section 2.9.7, “MySQL Source-Configuration Options”
Section 29.2.3, “Plugin API Components”

--plugin_name

Section 5.6.1, “Installing and Uninstalling Plugins”

--plugindir

Section 4.7.1, “`mysql_config` — Display Options for Compiling Clients”

--polltimeout

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--port

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.5.1.1, “mysql Client Options”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.7.1, “`mysql_config` — Display Options for Compiling Clients”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.8.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 5.8.4, “Using Client Programs in a Multiple-Server Environment”

port

Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

--port-open-timeout

Section 5.1.7, “Server Command Options”

--post-query

Section 4.5.8, “mysqlslap — Load Emulation Client”

--post-system

Section 4.5.8, “mysqlslap — Load Emulation Client”

--pre-query

Section 4.5.8, “mysqlslap — Load Emulation Client”

--pre-system

Section 4.5.8, “mysqlslap — Load Emulation Client”

PREFIX

Section 22.2.2.4, “Building NDB Cluster from Source on Linux”

--preserve-trailing-spaces

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--pretty

Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”

--print

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--print-data

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--print-defaults

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 4.6.4.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.7, “Server Command Options”
Section 2.11.12, “Upgrade Troubleshooting”

--print-full-config

Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--print-log

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--print-meta

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

print-sql-log

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--print-table-metadata

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--print_*

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--progress-frequency

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--promote-attributes

Section 22.4.15, “`ndb_move_data` — NDB Data Copy Utility”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--prompt

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

--protocol

Section 4.2.3, “Command Options for Connecting to the Server”

Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 5.8.3, “Running Multiple MySQL Instances on Unix”
Section 2.3.4.5, “Starting the Server for the First Time”
Section 2.3.4.9, “Testing The MySQL Installation”
Section 1.3.2, “The Main Features of MySQL”
Section 5.8.4, “Using Client Programs in a Multiple-Server Environment”

Q

[index top]

-Q

Section 4.5.4, “mysqldump — A Database Backup Program”

-q

Section 4.6.4.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.18, “ndb_print_file — Print NDB Disk Data File Contents”

--query

Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.14, “ndb_index_stat — NDB Index Statistics Utility”

--query-all

Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

--quick

Section 4.6.4.6, “myisamchk Memory Usage”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1, “mysql — The MySQL Command-Line Client”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section B.4.2.7, “Out of memory”
Section 7.6.1, “Using myisamchk for Crash Recovery”
Section 4.2.2.2, “Using Option Files”

--quote-names

Section 4.5.4, “mysqldump — A Database Backup Program”

R

[index top]

-R

Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.4.4, “Other myisamchk Options”
Section 6.2.2, “Privileges Provided by MySQL”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

-r

Section 29.4.2, “Adding a New User-Defined Function”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.9, “ndb_desc — Describe NDB Tables”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 5.1.7, “Server Command Options”

--raw

Section 4.5.1.1, “mysql Client Options”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--read-from-remote-master

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”

--read-from-remote-server

Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 6.2.2, “Privileges Provided by MySQL”

Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--read-only

Section 4.6.4.2, “myisamchk Check Options”

--rebuild-indexes

Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.1.4, “What is New in NDB Cluster”

--reconnect

Section 4.5.1.1, “mysql Client Options”

--recover

Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.4.6, “myisamchk Memory Usage”
Section 4.6.4.3, “myisamchk Repair Options”

--rejects

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--relative

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

--relay-log

Section 17.2.3.3, “Startup Options and Replication Channels”
Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”

--relay-log-index

Section 17.2.3.3, “Startup Options and Replication Channels”

--relay-log-info-repository

Configuring Multi-Source Replication
Section 17.1.6.3, “Replication Slave Options and Variables”
Setting Up Replication with Existing Data
Section 17.2.4.2, “Slave Status Logs”
Section 1.4, “What Is New in MySQL 8.0”

--relay-log-purge

Section 17.1.6.3, “Replication Slave Options and Variables”

relay-log-purge

Section 17.1.6.3, “Replication Slave Options and Variables”

--relay-log-recovery

Section 17.5.1.33, “Replication and Transaction Inconsistencies”

Section 17.2.4, “Replication Relay and Status Logs”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 17.2.4.2, “Slave Status Logs”
Section 5.1.17, “The Server Shutdown Process”

--relay-log-space-limit

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.3.3, “Startup Options and Replication Channels”

--reload

Section 22.5.15.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”
Section 22.3.3, “NDB Cluster Configuration Files”
Section 22.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”

--remove

Section 22.2.3.4, “Installing NDB Cluster Processes as Windows Services”
Section 22.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 22.4.1, “ndbd — The NDB Cluster Data Node Daemon”
Section 5.1.7, “Server Command Options”
Section 5.8.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.4.8, “Starting MySQL as a Windows Service”

--remove{

Section 22.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”

--repair

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--replace

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”

--replicate-*

Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 17.2.5.4, “Replication Channel Based Filters”
Section 17.2.5.3, “Replication Rule Application”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 26.12.11.8, “The replication_applier_filters Table”
Section 26.12.11.7, “The replication_applier_global_filters Table”

--replicate-*-db

Section 17.2.5.3, “Replication Rule Application”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 24.8, “Restrictions on Stored Programs”

--replicate-*-table

Section 17.2.5.3, “Replication Rule Application”

--replicate-do-*

Section 22.6.3, “Known Issues in NDB Cluster Replication”

--replicate-do-db

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 17.4.6, “Replicating Different Databases to Different Slaves”
Section 17.5.1.26, “Replication and Reserved Words”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.2.5.4, “Replication Channel Based Filters”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 5.4.4, “The Binary Log”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

--replicate-do-db:channel_1:db_name

Section 17.1.6.3, “Replication Slave Options and Variables”

--replicate-do-table

Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 17.2.5.2, “Evaluation of Table-Level Replication Options”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 17.5.1.26, “Replication and Reserved Words”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.2.5.3, “Replication Rule Application”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 16.6, “The BLACKHOLE Storage Engine”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

--replicate-do-table:channel_1:db_name.tbl_name

Section 17.1.6.3, “Replication Slave Options and Variables”

--replicate-ignore-*

Section 22.6.3, “Known Issues in NDB Cluster Replication”

--replicate-ignore-db

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 17.5.1.26, “Replication and Reserved Words”
Section 17.2.5.4, “Replication Channel Based Filters”
Section 17.2.5.3, “Replication Rule Application”

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 5.4.4, “The Binary Log”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

--replicate-ignore-db:channel_1:db_name

Section 17.1.6.3, “Replication Slave Options and Variables”

--replicate-ignore-table

Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 17.2.5.2, “Evaluation of Table-Level Replication Options”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 17.5.1.26, “Replication and Reserved Words”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 16.6, “The BLACKHOLE Storage Engine”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

--replicate-ignore-table:channel_1:db_name.tbl_name

Section 17.1.6.3, “Replication Slave Options and Variables”

--replicate-rewrite-db

Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

--replicate-same-server-id

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.1.5.3, “Disabling GTID Transactions Online”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 17.1.6.3, “Replication Slave Options and Variables”

--replicate-wild-do-table

Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 17.2.5.2, “Evaluation of Table-Level Replication Options”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 17.4.6, “Replicating Different Databases to Different Slaves”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 24.8, “Restrictions on Stored Programs”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”

--replicate-wild-do-table:channel_1:db_name.tbl_name

Section 17.1.6.3, “Replication Slave Options and Variables”

--replicate-wild-ignore-table

Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 17.2.5.2, “Evaluation of Table-Level Replication Options”

Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section A.14, “MySQL 8.0 FAQ: Replication”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”

--replicate-wild-ignore:channel_1:db_name.tbl_name

Section 17.1.6.3, “Replication Slave Options and Variables”

replication-ignore-table

Section 17.5.1.39, “Replication and Views”

--replication-rewrite-db

Section 17.1.6.3, “Replication Slave Options and Variables”

--report-host

Section 17.1.7.1, “Checking Replication Status”
Section 18.10, “Frequently Asked Questions”
Section 13.7.7.33, “SHOW SLAVE HOSTS Syntax”

--report-password

Section 17.1.6.2, “Replication Master Options and Variables”
Section 13.7.7.33, “SHOW SLAVE HOSTS Syntax”

--report-port

Section 18.10, “Frequently Asked Questions”
Section 13.7.7.33, “SHOW SLAVE HOSTS Syntax”

--report-user

Section 17.1.6.2, “Replication Master Options and Variables”
Section 13.7.7.33, “SHOW SLAVE HOSTS Syntax”

--restore-data

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Restoring to Fewer Nodes Than the Original

--restore-epoch

Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”

--restore-meta

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Restoring to More Nodes Than the Original

--restore-privilege-tables

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.1.4, “What is New in NDB Cluster”

--result-file

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--resume

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--retries

Section 22.4.9, “ndb_desc — Describe NDB Tables”

--rewrite-database

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--rewrite-db

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--routines

Section 14.7, “Data Dictionary Usage Differences”
Section 7.4.5.3, “Dumping Stored Programs”
Section 7.4.5.4, “Dumping Table Definitions and Content Separately”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

routing_strategy

Section 21.3, “Using MySQL Router with InnoDB Cluster”

--rowbatch

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--rowbytes

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--rowid

Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”

--rows

Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

S

[index top]

-S

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 4.2.1, “Invoking MySQL Programs”
Section 7.6.4, “MyISAM Table Optimization”

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.2.1.1, “NDB Cluster Auto-Installer Requirements”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 4.6.4.4, “Other myisamchk Options”
Section 22.2.1.2, “Using the NDB Cluster Auto-Installer”

-s

Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 4.7.2, “my_print_defaults — Display Options from Option Files”
Section 4.6.3, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.16, “ndb_perror — Obtain NDB Error Message Information”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 22.4.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 4.8.2, “perror — Display MySQL Error Message Information”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”

--safe-recover

Section 4.6.4.1, “myisamchk General Options”
Section 4.6.4.6, “myisamchk Memory Usage”
Section 4.6.4.3, “myisamchk Repair Options”

--safe-updates

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”

Section 5.1.8, “Server System Variables”

--safe-user-create

Section 5.1.7, “Server Command Options”

--savequeries

Section 22.4.28, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

--secure-auth

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 6.2.17, “Pluggable Authentication”
Section 5.1.7, “Server Command Options”
Section 1.4, “What Is New in MySQL 8.0”

--select-limit

Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”

--server-arg

Section 1.4, “What Is New in MySQL 8.0”

--server-file

Section 1.4, “What Is New in MySQL 8.0”

--server-id

Advanced Options
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 22.6.2, “General Requirements for NDB Cluster Replication”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 5.1.8, “Server System Variables”
Section 17.1.2.1, “Setting the Replication Master Configuration”
Section 13.7.7.33, “SHOW SLAVE HOSTS Syntax”
Section 5.4.4, “The Binary Log”
Section 17.5.4, “Troubleshooting Replication”

server-id

Section 17.1.2.8, “Adding Slaves to a Replication Environment”
Section 17.1.1, “Binary Log File Position Based Replication Configuration Overview”
Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.1.2.2, “Setting the Replication Slave Configuration”

--server-id-bits

MySQL Server Options for NDB Cluster
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
NDB Cluster System Variables

--server-log-file

Section 22.4.26, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster”

--server-name

Section 22.4.26, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster”

--server-public-key-path

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”

service-startup-timeout

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

--set-auto-increment

Section 4.6.4.4, “Other myisamchk Options”

--set-charset

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--set-collation

Section 4.6.4.3, “myisamchk Repair Options”

--set-gtid-purged

Section 2.11.14, “Copying MySQL Databases to Another Machine”
Creating a Data Snapshot Using mysqldump
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”

--shared-memory-base-name

Section 4.2.3, “Command Options for Connecting to the Server”

Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.8.4, “Using Client Programs in a Multiple-Server Environment”

--short-form

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--show

Section 4.7.2, “my_print_defaults — Display Options from Option Files”

--show-create-skip-secondary-engine

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 5.1.8, “Server System Variables”

--show-slave-auth-info

Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.7.33, “SHOW SLAVE HOSTS Syntax”

--show-table-type

Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”

--show-temp-status

Section 22.4.27, “ndb_show_tables — Display List of NDB Tables”

--show-warnings

Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

--sigint-ignore

Section 4.5.1.1, “mysql Client Options”

--silent

Section 4.6.4.1, “myisamchk General Options”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.16, “ndb_perror — Obtain NDB Error Message Information”
Section 4.8.2, “perror — Display MySQL Error Message Information”

Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”

--single-transaction

Section 7.2, “Database Backup Methods”
Section 7.3.1, “Establishing a Backup Policy”
Section 15.17.1, “InnoDB Backup”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”

--single-user

Section 22.4.30, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”

--skip

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.2.2.4, “Program Option Modifiers”
Section 5.1.7, “Server Command Options”

--skip-add-drop-table

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-add-locks

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-auto-rehash

Section 4.5.1.1, “mysql Client Options”

--skip-binlog

Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”

--skip-broken-objects

Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

--skip-character-set-client-handshake

Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 10.10.7.1, “The cp932 Character Set”

--skip-color

Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”

--skip-colors

Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”

--skip-column-names

Section 4.5.1.1, “mysql Client Options”

--skip-comments

Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-concurrent-insert

Section 5.1.7, “Server Command Options”

--skip-config-cache

Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--skip-data

Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”

--skip-database

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--skip-defer-table-indexes

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--skip-definer

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--skip-disable-keys

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-dump-date

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-dump-rows

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--skip-engine_name

Section 13.7.7.16, “SHOW ENGINES Syntax”
Section 25.9, “The INFORMATION_SCHEMA ENGINES Table”

--skip-events

Section 7.4.5.3, “Dumping Stored Programs”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--skip-extended-insert

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-external-locking

Section 8.11.5, “External Locking”
Section 8.14.2, “General Thread States”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”

--skip-federated

Section 17.4.4, “Using Replication with Different Master and Slave Storage Engines”

--skip-grant-tables

Section 13.7.4.1, “CREATE FUNCTION Syntax for User-Defined Functions”

Section 24.4.2, “Event Scheduler Configuration”
Section 13.7.4.3, “INSTALL COMPONENT Syntax”
Section 13.7.4.4, “INSTALL PLUGIN Syntax”
Section 5.5.1, “Installing and Uninstalling Components”
Section 5.6.1, “Installing and Uninstalling Plugins”
Section 5.7.1, “Installing and Uninstalling User-Defined Functions”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 6.2.17, “Pluggable Authentication”
Section 6.2.2, “Privileges Provided by MySQL”
Resetting the Root Password: Generic Instructions
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 5.3, “The mysql System Schema”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 29.4.2.5, “UDF Compiling and Installing”
Section 4.2.2.1, “Using Options on the Command Line”
Section 6.2.13, “When Privilege Changes Take Effect”

--skip-graphs

Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”

--skip-gtids

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”

--skip-host-cache

Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”

--skip-innodb

Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.6.1, “Installing and Uninstalling Plugins”
Section A.14, “MySQL 8.0 FAQ: Replication”
Section 5.1.7, “Server Command Options”

--skip-innodb-adaptive-hash-index

Section 15.13, “InnoDB Startup Options and System Variables”

--skip-innodb-doublewrite

Section 15.13, “InnoDB Startup Options and System Variables”

--skip-kill-mysqld

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--skip-line-numbers

Section 4.5.1.1, “mysql Client Options”

--skip-lock-tables

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-log-bin

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.3.1, “GTID Format and Storage”
Section 17.1.2.2, “Setting the Replication Slave Configuration”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Section 5.4.4, “The Binary Log”
Section 17.5.4, “Troubleshooting Replication”
Section 17.5.3, “Upgrading a Replication Setup”

--skip-mysqlx

Section 20.5.2, “Disabling X Plugin”

--skip-name-resolve

Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section 18.10, “Frequently Asked Questions”
Section 2.10.1, “Initializing the Data Directory”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 2.3.4.9, “Testing The MySQL Installation”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”

--skip-named-commands

Section 4.5.1.1, “mysql Client Options”

--skip-ndbcluster

MySQL Server Options for NDB Cluster
Section 22.3.2.5, “NDB Cluster mysqld Option and Variable Reference”

--skip-network-timeout

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-networking

Section B.4.2.2, “Can't connect to [local] MySQL server”
Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section B.4.2.8, “MySQL server has gone away”
Section 6.2.17, “Pluggable Authentication”
Resetting the Root Password: Generic Instructions
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 20.5.6.1, “Status Variables for X Plugin”
Section 22.5.10.30, “The ndbinfo processes Table”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 17.5.3, “Upgrading a Replication Setup”

skip-networking

Section A.14, “MySQL 8.0 FAQ: Replication”
Section 17.1.2.1, “Setting the Replication Master Configuration”
Section 17.5.4, “Troubleshooting Replication”

--skip-new

Section 29.5.1, “Debugging a MySQL Server”

Section 13.7.3.4, “OPTIMIZE TABLE Syntax”
Section 5.1.8, “Server System Variables”

--skip-nodegroup

Section 22.4.12, “`ndb_error_reporter` — NDB Error-Reporting Utility”

--skip-opt

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-pager

Section 4.5.1.1, “mysql Client Options”

--skip-partition

Section 1.4, “What Is New in MySQL 8.0”

--skip-password

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”

--skip-plugin-innodb-file-per-table

Section 5.1.7, “Server Command Options”

--skip-plugin_name

Section 5.6.1, “Installing and Uninstalling Plugins”

--skip-quick

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-quote-names

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-reconnect

Section 28.7.28, “C API Automatic Reconnection Control”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”

--skip-relaylog

Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”

--skip-routines

Section 7.4.5.3, “Dumping Stored Programs”

Section 4.5.6, “mysqlpump — A Database Backup Program”

--skip-safe-updates

Section 4.5.1.1, “mysql Client Options”

--skip-set-charset

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”

--skip-show-database

Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.7, “Server Command Options”
Section 13.7.7.14, “SHOW DATABASES Syntax”
Section 1.9.5, “Supporters of MySQL”

--skip-slave-preserve-commit-order

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 5.4.4, “The Binary Log”

--skip-slave-start

Section 17.1.2.8, “Adding Slaves to a Replication Environment”
Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 22.6.5, “Preparing the NDB Cluster for Replication”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Setting Up Replication with Existing Data
Section 13.4.2.6, “START SLAVE Syntax”
Section 22.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 17.2.3.3, “Startup Options and Replication Channels”
Section 17.5.4, “Troubleshooting Replication”
Section 17.5.3, “Upgrading a Replication Setup”

--skip-sort

Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”

--skip-ssl

Section 4.2.3, “Command Options for Connecting to the Server”
Section 1.4, “What Is New in MySQL 8.0”

--skip-stack-trace

Section 29.5.1.4, “Debugging mysqld under gdb”
Section 5.1.7, “Server Command Options”

--skip-super-large-pages

Section 8.12.3.2, “Enabling Large Page Support”
Section 5.1.7, “Server Command Options”

--skip-symbolic-links

Section 13.1.20, “CREATE TABLE Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”

--skip-sys-schema

Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 2.11.3, “What the MySQL Upgrade Process Upgrades”

--skip-syslog

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--skip-table-check

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--skip-triggers

Section 7.4.5.3, “Dumping Stored Programs”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--skip-tz-utc

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--skip-unknown-objects

Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”

--skip-version-check

Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

--skip-warn

Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”

--skip-watch-progress

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--skip-write-binlog

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--skip_grant_tables

Section 4.2.2.1, “Using Options on the Command Line”

--slave-checkpoint-group

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.3.3, “Startup Options and Replication Channels”

--slave-checkpoint-period

Section 17.1.6.3, “Replication Slave Options and Variables”

--slave-load-tmpdir

Section 17.4.1.2, “Backing Up Raw Data from a Slave”

Section 7.2, “Database Backup Methods”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.1.7, “Server Command Options”
Section B.4.3.5, “Where MySQL Stores Temporary Files”

--slave-max-allowed-packet

Section 17.1.6.3, “Replication Slave Options and Variables”

slave-max-allowed-packet

Section 17.1.6.3, “Replication Slave Options and Variables”

--slave-net-timeout

Section 17.1.6.3, “Replication Slave Options and Variables”

--slave-parallel-type

Section 17.1.6.3, “Replication Slave Options and Variables”

--slave-parallel-workers

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.3.3, “Startup Options and Replication Channels”

--slave-pending-jobs-size-max

Section 17.1.6.3, “Replication Slave Options and Variables”

--slave-preserve-commit-order

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 5.4.4, “The Binary Log”

slave-rows-search-algorithms

Section 17.1.6.3, “Replication Slave Options and Variables”

--slave-skip-counter

Section 17.2.3.3, “Startup Options and Replication Channels”

--slave-skip-errors

Section 22.6.8, “Implementing Failover with NDB Cluster Replication”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.5.1.28, “Slave Errors During Replication”

--slave-sql-verify-checksum

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.6.3, “Replication Slave Options and Variables”

--slave_net-timeout

Section 17.2.3.3, “Startup Options and Replication Channels”

--sleep

Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

--sleep-time

Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”

--slow-start-timeout

Section 5.1.7, “Server Command Options”

--slow_query_log

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.4.5, “The Slow Query Log”

--slow_query_log_file

Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.4.5, “The Slow Query Log”

--socket

Section B.4.2.2, “Can't connect to [local] MySQL server”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 2.5.6.3, “Deploying MySQL on Windows and Other Non-Linux Platforms with Docker”
Section B.4.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.5.1.1, “mysql Client Options”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.7.1, “`mysql_config` — Display Options for Compiling Clients”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.8.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.7, “Server Command Options”
Server Plugin Library and Plugin Descriptors
Section 2.3.4.9, “Testing The MySQL Installation”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 5.8.4, “Using Client Programs in a Multiple-Server Environment”

socket

Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

--sort

Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”

--sort-index

Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.4.4, “Other myisamchk Options”

--sort-records

Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.4.4, “Other myisamchk Options”

--sort-recover

Section 4.6.4.1, “myisamchk General Options”
Section 4.6.4.6, “myisamchk Memory Usage”
Section 4.6.4.3, “myisamchk Repair Options”

--sort_buffer_size

Section 5.1.7, “Server Command Options”

--sporadic-binlog-dump-fail

Section 17.1.6.4, “Binary Logging Options and Variables”

--sql-mode

Chapter 12, *Functions and Operators*
Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 5.1.7, “Server Command Options”
Section 5.1.11, “Server SQL Modes”

sql-mode

Section 5.1.11, “Server SQL Modes”

--ssl

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.3, “`mysql_ssl_rsa_setup` — Create SSL/RSA Files”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

--ssl*

Section 4.5.1.1, “mysql Client Options”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 5.1.7, “Server Command Options”

--ssl-ca

Section 13.7.1.1, “ALTER USER Syntax”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 13.7.1.3, “CREATE USER Syntax”
Section 6.3.3.2, “Creating SSL Certificates and Keys Using openssl”
Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”
Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”
Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”

--ssl-capath

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”

--ssl-cert

Section 13.7.1.1, “ALTER USER Syntax”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 13.7.1.3, “CREATE USER Syntax”
Section 6.3.3.2, “Creating SSL Certificates and Keys Using openssl”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”
Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”
Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”

--ssl-cipher

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”

--ssl-crl

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”

--ssl-crlpath

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”

--ssl-fips-mode

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.5, “FIPS Support”
Section 4.5.1.1, “mysql Client Options”

Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”

--ssl-key

Section 13.7.1.1, “ALTER USER Syntax”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 13.7.1.3, “CREATE USER Syntax”
Section 6.3.3.2, “Creating SSL Certificates and Keys Using openssl”
Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”
Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”
Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”

--ssl-mode

Section 13.7.1.1, “ALTER USER Syntax”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”
Section 13.7.1.3, “CREATE USER Syntax”
Section 28.7.7.50, “mysql_options()”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 1.4, “What Is New in MySQL 8.0”

--ssl-verify-server-cert

Section 1.4, “What Is New in MySQL 8.0”

--ssl-xxx

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 2.9.6, “Configuring SSL Library Support”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 5.1.8, “Server System Variables”

--staging-tries

Section 22.4.15, “ndb_move_data — NDB Data Copy Utility”

--standalone

Section 29.5.1.2, “Creating Trace Files”
Section 5.1.7, “Server Command Options”
Section 2.3.4.6, “Starting MySQL from the Windows Command Line”

--start-datetime

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5.1, “Point-in-Time Recovery Using Event Times”

--start-page

Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”

--start-position

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5.2, “Point-in-Time Recovery Using Event Positions”

--state-dir

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--statefile

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

--static

Section 28.7.4.2, “Building C API Client Programs Using pkg-config”

--stats

Section 4.6.3, “`myisam_ftdump` — Display Full-Text Index information”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

--status

Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”

--stop-datetime

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5.1, “Point-in-Time Recovery Using Event Times”

--stop-never

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--stop-never-slave-server-id

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--stop-position

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5.2, “Point-in-Time Recovery Using Event Positions”

--strict-check

Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”

--suffix

Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”
Section 4.4.3, “`mysql_ssl_rsa_setup` — Create SSL/RSA Files”

--super-large-pages

Section 8.12.3.2, “Enabling Large Page Support”
Section 5.1.7, “Server Command Options”

--symbolic-links

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”

--sys-*

Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-check

Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-create

Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

sys-create-if-not-exist

Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-create-if-not-valid

Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-drop

Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-skip-events

Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-skip-tables

Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

SYSCONFDIR

Section 4.2.2.2, “Using Option Files”

--sysdate-is-now

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 12.7, “Date and Time Functions”
Section 17.5.1.14, “Replication and System Functions”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

--syslog

Section 2.5.9, “Managing MySQL Server with systemd”
Section 4.5.1.3, “mysql Client Logging”

Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 4.9, “MySQL Environment Variables”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

--syslog-tag

Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

--system

Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

T

[index top]

-T

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.6.4.2, “myisamchk Check Options”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 5.1.7, “Server Command Options”

-t

Section 4.6.1, “ibd2sdi — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 22.4.8, “ndb_delete_all — Delete All Rows from an NDB Table”
Section 22.4.9, “ndb_desc — Describe NDB Tables”
Section 22.4.22, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”
Section 22.4.27, “ndb_show_tables — Display List of NDB Tables”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 22.4.30, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”
Section 5.1.7, “Server Command Options”

--tab

Section 7.1, “Backup and Recovery Types”
Section 7.2, “Database Backup Methods”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 22.4.13, “ndb_import — Import CSV Data Into NDB”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”

Section 7.4, “Using mysqldump for Backups”

--table

Section 4.5.1.1, “mysql Client Options”
Section 22.4.9, “ndb_desc — Describe NDB Tables”

--tables

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”

--tc-heuristic-recover

Section 5.1.7, “Server Command Options”

--tcp-ip

Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

--tee

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

--temp-pool

Section 5.1.7, “Server Command Options”
Section 1.4, “What Is New in MySQL 8.0”

--tempdelay

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--temperrors

Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

--test

Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”

Text

Section 1.2, “Typographical and Syntax Conventions”

--text

Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”

--thread_cache_size

Section 29.5.1.4, “Debugging mysqld under gdb”

--timeout

Section 22.4.30, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”

--timezone

Section 5.1.13, “MySQL Server Time Zone Support”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 17.5.1.32, “Replication and Time Zones”
Section 5.1.8, “Server System Variables”
Section B.4.3.7, “Time Zone Problems”

--tls-ciphersuites

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”

--tls-version

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”

--tmpdir

Section B.4.2.12, “Can't create/write to file”
Section 4.6.4.6, “myisamchk Memory Usage”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.1.7, “Server Command Options”
Section 2.3.4.8, “Starting MySQL as a Windows Service”
Section B.4.3.5, “Where MySQL Stores Temporary Files”

tmpdir

Section 2.3, “Installing MySQL on Microsoft Windows”

--to-last-log

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--transaction-isolation

Section 5.1.7, “Server Command Options”

Section 5.1.8, “Server System Variables”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 15.7.2.1, “Transaction Isolation Levels”

--transaction-read-only

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 13.3.7, “SET TRANSACTION Syntax”

--transactional

Section 22.4.8, “`ndb_delete_all` — Delete All Rows from an NDB Table”

--triggers

Section 7.4.5.3, “Dumping Stored Programs”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--try-reconnect

Section 22.4.5, “`ndb_mgm` — The NDB Cluster Management Client”

--tupscan

Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”

--type

Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”
Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 22.4.27, “`ndb_show_tables` — Display List of NDB Tables”

--tz-utc

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

U

[index top]

-U

Section 4.6.4.2, “myisamchk Check Options”
Section 4.5.1.1, “mysql Client Options”

-u

Section 6.2.1, “Account User Names and Passwords”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.5, “`myisamlog` — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.9, “ndb_desc — Describe NDB Tables”
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 22.4.27, “ndb_show_tables — Display List of NDB Tables”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 5.1.7, “Server Command Options”
Section 2.3.4.9, “Testing The MySQL Installation”
Section 2.10.3, “Testing the Server”
Section 2.11, “Upgrading MySQL”
Section 2.3.6, “Windows Postinstallation Procedures”

--uid

Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”
Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”

--unbuffered

Section 4.5.1.1, “mysql Client Options”

--unpack

Section 16.2.3, “MyISAM Table Storage Formats”
Section 4.6.4.3, “myisamchk Repair Options”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”

--unqualified

Section 22.4.9, “ndb_desc — Describe NDB Tables”
Section 22.4.27, “ndb_show_tables — Display List of NDB Tables”

--update

Section 22.4.14, “ndb_index_stat — NDB Index Statistics Utility”

--update-state

Section 7.6.3, “How to Repair MyISAM Tables”
Section 4.6.4.2, “myisamchk Check Options”
Section 16.2, “The MyISAM Storage Engine”

--upgrade

Section 2.11.4, “Changes in MySQL 8.0”
Section 2.3.3.4, “MySQL Installer Product Catalog and Dashboard”
Section 22.6.4, “NDB Cluster Replication Schema and Tables”
Section 5.1.7, “Server Command Options”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 1.4, “What Is New in MySQL 8.0”
Section 2.11.3, “What the MySQL Upgrade Process Upgrades”

--upgrade-system-tables

Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”

Section 22.6.4, “NDB Cluster Replication Schema and Tables”
Section 2.11.3, “What the MySQL Upgrade Process Upgrades”

--uri

Section 4.2.5, “Connecting to the Server Using URI-Like Strings or Key-Value Pairs”
Section 21.4, “Working with InnoDB Cluster”

--usage

Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--use-default

Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”

--use-frm

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--use-http

Section 22.4.26, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster”

--use-https

Section 22.2.1.1, “NDB Cluster Auto-Installer Requirements”
Section 22.4.26, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster”

--use-threads

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--useHexFormat

Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”

--user

Section 6.2.1, “Account User Names and Passwords”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 7.3, “Example Backup and Recovery Strategy”
Section B.4.2.17, “File Not Found and Similar Errors”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section 2.10.1, “Initializing the Data Directory”
Section 4.2.1, “Invoking MySQL Programs”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”
Section 4.5.1.3, “mysql Client Logging”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.5.12.3, “NDB Cluster and MySQL Security Procedures”
Section 22.4.29, “ndb_top — View CPU usage information for NDB threads”
Section 4.2.2.6, “Option Defaults, Options Expecting Values, and the = Sign”
Resetting the Root Password: Unix and Unix-Like Systems
Restoring to More Nodes Than the Original
Section 5.1.7, “Server Command Options”
Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”
Section 4.2.2, “Specifying Program Options”
Section 2.10.2, “Starting the Server”
Section 6.4.1.10, “Test Pluggable Authentication”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 4.2.2.2, “Using Option Files”

user

Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

--users

Section 4.5.6, “mysqlpump — A Database Backup Program”

V

[index top]

-V

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 4.7.2, “my_print_defaults — Display Options from Option Files”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.16, “ndb_perror — Obtain NDB Error Message Information”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 4.8.2, “perror — Display MySQL Error Message Information”
Section 5.1.7, “Server Command Options”

Section 4.2.2.1, “Using Options on the Command Line”
Section 22.1.4, “What is New in NDB Cluster”

-v

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.5, “`myisamlog` — Display MyISAM Log File Contents”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.3, “`mysql_ssl_rsa_setup` — Create SSL/RSA Files”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.16, “`ndb_perror` — Obtain NDB Error Message Information”
Section 22.4.18, “`ndb_print_file` — Print NDB Disk Data File Contents”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 4.6.4.5, “Obtaining Table Information with myisamchk”
Section 4.8.2, “`perror` — Display MySQL Error Message Information”
Section 5.1.7, “Server Command Options”
Section 4.2.2.1, “Using Options on the Command Line”
Section 6.4.4.4, “Using the keyring_okv KMIP Plugin”

--validate-config

Section 5.1.7, “Server Command Options”
Section 5.1.3, “Server Configuration Validation”

--validate-password

Section 6.4.3.2, “Password Validation Options and Variables”
Section 6.4.3.3, “Transitioning to the Password Validation Component”

--var_name

Section 15.13, “InnoDB Startup Options and System Variables”
Section 4.6.4.1, “myisamchk General Options”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 5.1.7, “Server Command Options”

--variable

Section 28.7.4.2, “Building C API Client Programs Using pkg-config”
Section 4.7.1, “`mysql_config` — Display Options for Compiling Clients”

--verbose

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 4.5.1.5, “Executing SQL Statements from a Text File”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.3, “`mysql_ssl_rsa_setup` — Create SSL/RSA Files”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 22.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 22.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 22.4.15, “`ndb_move_data` — NDB Data Copy Utility”
Section 22.4.16, “`ndb_perror` — Obtain NDB Error Message Information”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 4.6.4.4, “Other myisamchk Options”
Section 4.8.2, “`perror` — Display MySQL Error Message Information”
Section 5.1.7, “Server Command Options”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”
Section 4.2.2.2, “Using Option Files”
Section 4.2.2.1, “Using Options on the Command Line”

--verify-binlog-checksum

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--version

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 4.7.2, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.7.1, “`mysql_config` — Display Options for Compiling Clients”
Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”

Section 4.4.3, “mysql_ssl_rsa_setup — Create SSL/RSA Files”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 22.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 22.4.16, “ndb_perror — Obtain NDB Error Message Information”
Section 22.4.31, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 4.8.2, “perror — Display MySQL Error Message Information”
Section 5.1.7, “Server Command Options”
Section 4.2.2.1, “Using Options on the Command Line”

--version-check

Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”

--vertical

Section 1.7, “How to Report Bugs or Problems”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”

W

[index top]

-W

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 5.1.7, “Server Command Options”

-w

Section 4.6.2, “innochecksum — Offline InnoDB File Checksum Utility”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.5, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.6, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “mysql_config_editor — MySQL Configuration Utility”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 22.4.30, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”

--wait

Section 4.6.4.1, “myisamchk General Options”
Section 4.6.6, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”

--wait-nodes

Section 22.4.30, “`ndb_waiter` — Wait for NDB Cluster to Reach a Given Status”

--warn

Section 4.6.7, “`mysql_config_editor` — MySQL Configuration Utility”

--watch-progress

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

--where

Section 4.5.4, “`mysqldump` — A Database Backup Program”

WITH_ANT

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_BOOST

Section 2.9.7, “MySQL Source-Configuration Options”
Section 2.9.2, “Source Installation Prerequisites”

WITH_BUNDLED_MEMCACHED

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_CLASSPATH

Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section 22.2.3.2, “Compiling and Installing NDB Cluster from Source on Windows”

WITH_CLIENT_PROTOCOL_TRACING

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_CURL

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_DEBUG

Section 15.13, “InnoDB Startup Options and System Variables”
Section 4.5.1.1, “mysql Client Options”
Section 2.9.7, “MySQL Source-Configuration Options”

WITH_EDITLINE

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_EMBEDDED_SERVER

Section 1.4, “What Is New in MySQL 8.0”

WITH_EMBEDDED_SHARED_LIBRARY

Section 1.4, “What Is New in MySQL 8.0”

WITH_GMOCK

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_ICU

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_LIBEVENT

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_LOCK_ORDER

Section 2.9.7, “MySQL Source-Configuration Options”
Section 29.5.3, “The LOCK_ORDER Tool”

WITH_LTO

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_LZ4

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_LZMA

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_MECAB

Section 12.9.9, “MeCab Full-Text Parser Plugin”

WITH_NDB_JAVA

Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section 22.2.3.2, “Compiling and Installing NDB Cluster from Source on Windows”

WITH_NDBCLUSTER

Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section 22.2.3.2, “Compiling and Installing NDB Cluster from Source on Windows”
Section 2.9.7, “MySQL Source-Configuration Options”

WITH_NDBCLUSTER_STORAGE_ENGINE

Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section 22.2.3.2, “Compiling and Installing NDB Cluster from Source on Windows”

WITH_NUMA

Section 15.13, “InnoDB Startup Options and System Variables”
Section 2.9.7, “MySQL Source-Configuration Options”

WITH_PLUGIN_NDBCLUSTER

Section 22.2.2.4, “Building NDB Cluster from Source on Linux”
Section 22.2.3.2, “Compiling and Installing NDB Cluster from Source on Windows”

WITH_PROTOBUF

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_RE2

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_SSL

Section 2.9.6, “Configuring SSL Library Support”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 2.9.2, “Source Installation Prerequisites”

WITH_SYSTEMD

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_TEST_TRACE_PLUGIN

Section 2.9.7, “MySQL Source-Configuration Options”
Using the Test Protocol Trace Plugin
Using Your Own Protocol Trace Plugins

WITH_ZLIB

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_ZSTD

Section 2.9.7, “MySQL Source-Configuration Options”

--write

Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”

--write-binlog

Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 17.5.3, “Upgrading a Replication Setup”

X

[index top]

-X

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

-x

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”
Section 22.4.29, “`ndb_top` — View CPU usage information for NDB threads”

--xml

Section 13.2.8, “LOAD XML Syntax”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 22.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 12.11, “XML Functions”

Y

[index top]

-Y
Section 4.5.4, “mysqldump — A Database Backup Program”

-y
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.8, “mysqlslap — Load Emulation Client”

Z

[index top]

-z
Section 22.4.23, “ndb_restore — Restore an NDB Cluster Backup”
Section 22.4.24, “ndb_select_all — Print Rows from an NDB Table”

--zstd-compression-level
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.6, “Connection Compression Control”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”

Privileges Index

A | B | C | D | E | F | G | I | L | N | P | R | S | T | U | V | X

A

[index top]

ALL
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”

ALL PRIVILEGES
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.7.21, “SHOW GRANTS Syntax”

ALTER
Section 13.1.2, “ALTER DATABASE Syntax”
Section 13.1.9, “ALTER TABLE Syntax”
Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”

Section 13.1.36, “RENAME TABLE Syntax”
Section 21.4, “Working with InnoDB Cluster”

ALTER ROUTINE

Section 13.1.4, “ALTER FUNCTION Syntax”
Section 13.1.7, “ALTER PROCEDURE Syntax”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.29, “DROP PROCEDURE and DROP FUNCTION Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”
Section 24.7, “Stored Program Binary Logging”
Section 24.2.2, “Stored Routines and MySQL Privileges”
Section 21.4, “Working with InnoDB Cluster”

APPLICATION_PASSWORD_ADMIN

Section 13.7.1.1, “ALTER USER Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.15, “Password Management”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.10, “SET PASSWORD Syntax”

AUDIT_ADMIN

Section 6.4.5.1, “Audit Log Components”
Section 6.4.5.9, “Audit Log Reference”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”

B

[index top]

BACKUP_ADMIN

Section 2.11.4, “Changes in MySQL 8.0”
Section 5.6.7.2, “Cloning Data Locally”
Section 18.4.3.1, “Cloning for Distributed Recovery”
Section 5.6.7.3, “Cloning Remote Data”
Section 13.7.1.6, “GRANT Syntax”
Section 13.3.5, “LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Syntax”
Prerequisites for Cloning
Section 6.2.2, “Privileges Provided by MySQL”
Section 18.2.1.3, “User Credentials”
Section 1.4, “What Is New in MySQL 8.0”
Section 21.2.5.1, “Working with a Cluster that uses MySQL Clone”

BINLOG_ADMIN

Section 13.7.8.1, “BINLOG Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.1, “PURGE BINARY LOGS Syntax”

BINLOG_ENCRYPTION_ADMIN

Section 13.1.5, “ALTER INSTANCE Syntax”
Section 17.3.2.3, “Binary Log Master Key Rotation”
Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”

C

[index top]

CLONE_ADMIN

Section 5.6.7.3, “Cloning Remote Data”
Section 13.7.1.6, “GRANT Syntax”
Prerequisites for Cloning
Section 6.2.2, “Privileges Provided by MySQL”

CONNECTION_ADMIN

Section 6.2.11, “Account Categories”
Section 13.7.1, “Account Management Statements”
Section 13.1.5, “ALTER INSTANCE Syntax”
Section 13.7.1.1, “ALTER USER Syntax”
Section 6.2.14, “Assigning Account Passwords”
Section 10.5, “Configuring Application Character Set and Collation”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 13.7.1.2, “CREATE ROLE Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 13.7.1.4, “DROP ROLE Syntax”
Section 13.7.1.5, “DROP USER Syntax”
Section 18.6.6.4, “Exit Action”
Section 13.7.1.6, “GRANT Syntax”
Section 18.8, “Group Replication System Variables”
Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 13.7.8.4, “KILL Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.7, “RENAME USER Syntax”
Section 13.7.1.8, “REVOKE Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.1.10, “SET PASSWORD Syntax”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section B.4.2.6, “Too many connections”

CREATE

Section 13.1.9, “ALTER TABLE Syntax”
Section 13.1.12, “CREATE DATABASE Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 13.7.1.6, “GRANT Syntax”
Section 13.2.5, “IMPORT TABLE Syntax”

Section 6.2.2, “Privileges Provided by MySQL”
Section 13.1.36, “RENAME TABLE Syntax”
Section 21.4, “Working with InnoDB Cluster”

CREATE ROLE

Section 13.7.1.2, “CREATE ROLE Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 6.2.10, “Using Roles”

CREATE ROUTINE

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 6.2.2, “Privileges Provided by MySQL”
Section 24.7, “Stored Program Binary Logging”
Section 24.2.2, “Stored Routines and MySQL Privileges”
Section 21.4, “Working with InnoDB Cluster”

CREATE TABLESPACE

Section 13.1.10, “ALTER TABLESPACE Syntax”
Section 15.6.3.3, “General Tablespaces”
Section 13.7.1.6, “GRANT Syntax”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 6.2.2, “Privileges Provided by MySQL”

CREATE TEMPORARY TABLES

Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 21.4, “Working with InnoDB Cluster”

CREATE USER

Section 6.2.11, “Account Categories”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 13.7.1.1, “ALTER USER Syntax”
Section 6.2.14, “Assigning Account Passwords”
Section 13.7.1.2, “CREATE ROLE Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 13.7.1.4, “DROP ROLE Syntax”
Section 13.7.1.5, “DROP USER Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.15, “Password Management”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.7, “RENAME USER Syntax”
Section 13.7.1.8, “REVOKE Syntax”
Section 13.7.1.9, “SET DEFAULT ROLE Syntax”
Section 13.7.1.10, “SET PASSWORD Syntax”
Section 6.2.10, “Using Roles”
Section 21.4, “Working with InnoDB Cluster”

Writing the Server-Side Authentication Plugin

CREATE VIEW

Section 13.1.11, “ALTER VIEW Syntax”
Section 13.1.23, “CREATE VIEW Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 24.9, “Restrictions on Views”
Section 21.4, “Working with InnoDB Cluster”

D

[index top]

DELETE

Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.2.2, “DELETE Syntax”
Section 13.7.4.2, “DROP FUNCTION Syntax”
Section 13.7.1.5, “DROP USER Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 5.6.1, “Installing and Uninstalling Plugins”
Section 5.7.1, “Installing and Uninstalling User-Defined Functions”
Section 22.5.12.2, “NDB Cluster and MySQL Privileges”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.2.9, “REPLACE Syntax”
Section 16.7, “The MERGE Storage Engine”
Section 26.12.2.4, “The setup_objects Table”
Section 29.4.2.5, “UDF Compiling and Installing”
Section 29.4.2.6, “UDF Security Precautions”
Section 13.7.4.5, “UNINSTALL COMPONENT Syntax”
Section 13.7.4.6, “UNINSTALL PLUGIN Syntax”
Section 6.2.10, “Using Roles”
Section 21.4, “Working with InnoDB Cluster”

DROP

Section 6.2, “Access Control and Account Management”
Section 13.1.9, “ALTER TABLE Syntax”
Section 13.1.11, “ALTER VIEW Syntax”
Section 13.1.23, “CREATE VIEW Syntax”
Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section 13.1.24, “DROP DATABASE Syntax”
Section 13.1.32, “DROP TABLE Syntax”
Section 13.1.35, “DROP VIEW Syntax”
Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 13.7.1.6, “GRANT Syntax”
Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 12.19.1, “MySQL Enterprise Encryption Installation”
Section 26.11, “Performance Schema General Table Characteristics”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.1.36, “RENAME TABLE Syntax”

Section 26.12.18.1, “The host_cache Table”
Section 13.1.37, “TRUNCATE TABLE Syntax”
Section 21.4, “Working with InnoDB Cluster”

DROP ROLE

Section 13.7.1.4, “DROP ROLE Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 6.2.10, “Using Roles”

E

[index top]

ENCRYPTION_KEY_ADMIN

Section 13.1.5, “ALTER INSTANCE Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 6.4.4.11, “Keyring System Variables”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”
Section 6.2.2, “Privileges Provided by MySQL”

EVENT

Section 13.1.3, “ALTER EVENT Syntax”
Section 13.1.13, “CREATE EVENT Syntax”
Section 13.1.25, “DROP EVENT Syntax”
Section 24.4.1, “Event Scheduler Overview”
Section 24.4.3, “Event Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.7.7, “SHOW CREATE EVENT Syntax”
Section 13.7.7.18, “SHOW EVENTS Syntax”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”
Section 21.4, “Working with InnoDB Cluster”

EXECUTE

Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.29, “DROP PROCEDURE and DROP FUNCTION Syntax”
Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”
Section 13.7.1.6, “GRANT Syntax”
Section 5.6.7.9, “Monitoring Cloning Operations”
Section 27.1, “Prerequisites for Using the sys Schema”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”
Section 24.6, “Stored Object Access Control”
Section 24.2.2, “Stored Routines and MySQL Privileges”
Section 21.4, “Working with InnoDB Cluster”

F

[index top]

FILE

Section 13.1.20, “CREATE TABLE Syntax”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 13.2.5, “IMPORT TABLE Syntax”
Section 13.2.7, “LOAD DATA Syntax”
Section 13.2.8, “LOAD XML Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.19, “Replication and LOAD DATA”
Section 13.2.10.1, “SELECT ... INTO Syntax”
Section 5.1.8, “Server System Variables”
Section 12.5, “String Functions and Operators”
Section 11.4.3, “The BLOB and TEXT Types”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 1.4, “What Is New in MySQL 8.0”
Section 21.4, “Working with InnoDB Cluster”

FIREWALL_ADMIN

Section 13.7.1.6, “GRANT Syntax”
Section 6.4.7.1, “MySQL Enterprise Firewall Components”
Section 6.2.2, “Privileges Provided by MySQL”

FIREWALL_USER

Section 13.7.1.6, “GRANT Syntax”
Section 6.4.7.1, “MySQL Enterprise Firewall Components”
Section 6.2.2, “Privileges Provided by MySQL”

G

[index top]

GRANT OPTION

Section 13.7.1.6, “GRANT Syntax”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.8, “REVOKE Syntax”
Section 13.7.7.21, “SHOW GRANTS Syntax”
Section 25.7, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”
Section 25.25, “The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table”
Section 25.33, “The INFORMATION_SCHEMA TABLE_PRIVILEGES Table”
Section 25.35, “The INFORMATION_SCHEMA USER_PRIVILEGES Table”
Section 21.4, “Working with InnoDB Cluster”

GROUP_REPLICATION_ADMIN

Section 13.4.3.5, “Functions to Inspect and Configure the Maximum Consensus Instances of a Group”
Section 13.4.3.6, “Functions to Inspect and Set the Group Replication Communication Protocol Version”
Section 13.7.1.6, “GRANT Syntax”

Section 18.8, “Group Replication System Variables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 18.4.1.4, “Setting a Group's Communication Protocol Version”
Section 13.4.3.1, “START GROUP_REPLICATION Syntax”
Section 13.4.3.2, “STOP GROUP_REPLICATION Syntax”
Section 18.4.1.3, “Using Group Replication Group Write Consensus”

I

[index top]

INDEX

Section 13.1.9, “ALTER TABLE Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 21.4, “Working with InnoDB Cluster”

INNODB_REDO_LOG_ARCHIVE

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 15.6.5, “Redo Log”

INSERT

Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 13.1.9, “ALTER TABLE Syntax”
Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 6.2.14, “Assigning Account Passwords”
Section 13.7.4.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 13.7.1.3, “CREATE USER Syntax”
Section 13.1.23, “CREATE VIEW Syntax”
Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 13.7.1.6, “GRANT Syntax”
Section 13.2.6, “INSERT Syntax”
Section 13.7.4.3, “INSTALL COMPONENT Syntax”
Section 13.7.4.4, “INSTALL PLUGIN Syntax”
Section 5.6.1, “Installing and Uninstalling Plugins”
Section 5.7.1, “Installing and Uninstalling User-Defined Functions”
Section 12.19.1, “MySQL Enterprise Encryption Installation”
Section 13.7.3.4, “OPTIMIZE TABLE Syntax”
Section 16.11.1, “Pluggable Storage Engine Architecture”
Section 27.1, “Prerequisites for Using the sys Schema”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.1.36, “RENAME TABLE Syntax”
Section 13.7.3.5, “REPAIR TABLE Syntax”
Section 13.2.9, “REPLACE Syntax”
Section 17.5.1.19, “Replication and LOAD DATA”
Section 17.3.3, “Replication Privilege Checks”
Section 5.1.7, “Server Command Options”
Section 24.6, “Stored Object Access Control”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”
Section 26.12.2.4, “The setup_objects Table”

Section 29.4.2.5, “UDF Compiling and Installing”
Section 29.4.2.6, “UDF Security Precautions”
Section 6.2.10, “Using Roles”
Section 21.4, “Working with InnoDB Cluster”

L

[index top]

LOCK TABLES

Section 13.7.8.3, “FLUSH Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”

N

[index top]

NDB_STORED_USER

Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 22.1.4, “What is New in NDB Cluster”

P

[index top]

PERSIST_RO_VARIABLES_ADMIN

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.8.7, “RESET PERSIST Syntax”
Section 5.1.9.1, “System Variable Privileges”

PROCESS

Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 15.16.2, “Enabling InnoDB Monitors”
Section 24.4.2, “Event Scheduler Configuration”
Section 8.14, “Examining Thread Information”
Section 13.8.2, “EXPLAIN Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 25.1, “Introduction”
Section 13.7.8.4, “KILL Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 22.5.4, “MySQL Server Usage for NDB Cluster”
Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”
Section 27.1, “Prerequisites for Using the sys Schema”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.7.15, “SHOW ENGINE Syntax”
Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
Section 25.39.1, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table”

Section 25.39.2, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table”
Section 25.39.3, “The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table”
Section 25.39.4, “The INFORMATION_SCHEMA INNODB_CACHED_INDEXES Table”
Section 25.39.5, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”
Section 25.39.7, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”
Section 25.39.6, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”
Section 25.39.8, “The INFORMATION_SCHEMA INNODB_COLUMNS Table”
Section 25.39.9, “The INFORMATION_SCHEMA INNODB_DATAFILES Table”
Section 25.39.10, “The INFORMATION_SCHEMA INNODB_FIELDS Table”
Section 25.39.11, “The INFORMATION_SCHEMA INNODB_FOREIGN Table”
Section 25.39.12, “The INFORMATION_SCHEMA INNODB_FOREIGN_COLS Table”
Section 25.39.13, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 25.39.14, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”
Section 25.39.15, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”
Section 25.39.16, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 25.39.17, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”
Section 25.39.18, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”
Section 25.39.19, “The INFORMATION_SCHEMA INNODB_INDEXES Table”
Section 25.39.21, “The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table”
Section 25.39.20, “The INFORMATION_SCHEMA INNODB_LOCKS Table”
Section 25.39.22, “The INFORMATION_SCHEMA INNODB_METRICS Table”
Section 25.39.23, “The INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES Table”
Section 25.39.24, “The INFORMATION_SCHEMA INNODB_TABLES Table”
Section 25.39.25, “The INFORMATION_SCHEMA INNODB_TABLESPACES Table”
Section 25.39.26, “The INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF Table”
Section 25.39.27, “The INFORMATION_SCHEMA INNODB_TABLESTATS View”
Section 25.39.28, “The INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO Table”
Section 25.39.29, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 25.39.30, “The INFORMATION_SCHEMA INNODB_VIRTUAL Table”
Section 25.19, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 26.12.18.5, “The threads Table”
Section 21.4, “Working with InnoDB Cluster”

PROXY

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Implementing Proxy User Support in Authentication Plugins
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 6.4.1.5, “PAM Pluggable Authentication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 6.2.18, “Proxy Users”
Section 2.10.4, “Securing the Initial MySQL Account”
Section 26.12.18.1, “The host_cache Table”
Section 6.4.1.6, “Windows Pluggable Authentication”

PROXY ... WITH GRANT OPTION

Section 6.2.18, “Proxy Users”

R

[index top]

REFERENCES

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”
Section 21.4, “Working with InnoDB Cluster”

RELOAD

Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 2.11.4, “Changes in MySQL 8.0”
Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section 13.7.8.3, “FLUSH Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 13.3.5, “LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Syntax”
Section 28.7.7.58, “mysql_refresh()”
Section 28.7.7.59, “mysql_reload()”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 13.7.8.6, “RESET Syntax”
Section 26.12.18.1, “The host_cache Table”
Section 21.4, “Working with InnoDB Cluster”

REPLICATION CLIENT

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.7.1, “SHOW BINARY LOGS Syntax”
Section 13.7.7.23, “SHOW MASTER STATUS Syntax”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 21.4, “Working with InnoDB Cluster”

REPLICATION SLAVE

Section 17.1.2.3, “Creating a User for Replication”
Section 13.7.1.6, “GRANT Syntax”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”
Section 13.7.7.2, “SHOW BINLOG EVENTS Syntax”
Section 13.7.7.32, “SHOW RELAYLOG EVENTS Syntax”
Section 13.7.7.33, “SHOW SLAVE HOSTS Syntax”
Section 18.2.1.3, “User Credentials”
Section 21.4, “Working with InnoDB Cluster”

REPLICATION_APPLIER

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 13.7.8.1, “BINLOG Syntax”
Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 13.7.1.6, “GRANT Syntax”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”

Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.3.3, “Replication Privilege Checks”
Section 5.1.8, “Server System Variables”

REPLICATION_SLAVE_ADMIN

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Section 13.4.2.6, “START SLAVE Syntax”
Section 13.4.2.7, “STOP SLAVE Syntax”

RESOURCE_GROUP_ADMIN

Section 13.7.2.1, “ALTER RESOURCE GROUP Syntax”
Section 13.7.2.2, “CREATE RESOURCE GROUP Syntax”
Section 13.7.2.3, “DROP RESOURCE GROUP Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 8.9.3, “Optimizer Hints”
Section 6.2.2, “Privileges Provided by MySQL”
Section 8.12.5, “Resource Groups”
Section 13.7.2.4, “SET RESOURCE GROUP Syntax”

RESOURCE_GROUP_USER

Section 13.7.1.6, “GRANT Syntax”
Section 8.9.3, “Optimizer Hints”
Section 6.2.2, “Privileges Provided by MySQL”
Section 8.12.5, “Resource Groups”
Section 13.7.2.4, “SET RESOURCE GROUP Syntax”

ROLE_ADMIN

Section 13.7.1.6, “GRANT Syntax”
Section 12.15, “Information Functions”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”
Section 6.2.10, “Using Roles”

S

[index top]

SELECT

Section 6.2, “Access Control and Account Management”
Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 6.2.11, “Account Categories”
Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 13.7.3.3, “CHECKSUM TABLE Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.20.4, “CREATE TABLE ... LIKE Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.1.23, “CREATE VIEW Syntax”

Section 14.7, “Data Dictionary Usage Differences”
Section 13.2.2, “DELETE Syntax”
Section 13.8.2, “EXPLAIN Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 13.2.6, “INSERT Syntax”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 5.6.7.9, “Monitoring Cloning Operations”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.5.12.2, “NDB Cluster and MySQL Privileges”
Section 13.7.3.4, “OPTIMIZE TABLE Syntax”
Section 26.11, “Performance Schema General Table Characteristics”
Section 27.1, “Prerequisites for Using the sys Schema”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.3.5, “REPAIR TABLE Syntax”
Section 24.9, “Restrictions on Views”
Section 13.7.1.8, “REVOKE Syntax”
Section 13.7.7.12, “SHOW CREATE USER Syntax”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 13.7.7.21, “SHOW GRANTS Syntax”
Section 24.6, “Stored Object Access Control”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”
Section 25.39.21, “The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table”
Section 25.39.20, “The INFORMATION_SCHEMA INNODB_LOCKS Table”
Section 16.7, “The MERGE Storage Engine”
Section 24.3.1, “Trigger Syntax and Examples”
Section 13.2.12, “UPDATE Syntax”
Section 21.4, “Working with InnoDB Cluster”

SERVICE_CONNECTION_ADMIN

Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 6.2.2, “Privileges Provided by MySQL”

SESSION_VARIABLES_ADMIN

Section 13.7.1.6, “GRANT Syntax”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.3.3, “Replication Privilege Checks”
Section 5.1.9.1, “System Variable Privileges”

SET_USER_ID

Section 6.2.11, “Account Categories”
Section 13.1.11, “ALTER VIEW Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 24.6, “Stored Object Access Control”
Section 24.7, “Stored Program Binary Logging”

SHOW DATABASES

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”

Section 13.7.7.14, “SHOW DATABASES Syntax”

SHOW VIEW

Section 13.8.2, “EXPLAIN Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 24.9, “Restrictions on Views”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”

SHUTDOWN

Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 5.6.7.3, “Cloning Remote Data”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 28.7.7.75, “mysql_shutdown()”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.8.8, “RESTART Syntax”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Section 13.7.8.9, “SHUTDOWN Syntax”
Section 5.1.17, “The Server Shutdown Process”
Section 21.4, “Working with InnoDB Cluster”

SUPER

Section 6.2.11, “Account Categories”
Section 13.7.1, “Account Management Statements”
Section 13.1.4, “ALTER FUNCTION Syntax”
Section 13.1.5, “ALTER INSTANCE Syntax”
Section 13.1.8, “ALTER SERVER Syntax”
Section 13.7.1.1, “ALTER USER Syntax”
Section 13.1.11, “ALTER VIEW Syntax”
Section 6.2.14, “Assigning Account Passwords”
Section 6.4.5.6, “Audit Log Filtering”
Section 6.4.5.9, “Audit Log Reference”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 13.7.8.1, “BINLOG Syntax”
Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 5.6.7.12, “Clone System Variables”
Section 10.5, “Configuring Application Character Set and Collation”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.7.1.2, “CREATE ROLE Syntax”
Section 13.1.18, “CREATE SERVER Syntax”
Section 13.1.19, “CREATE SPATIAL REFERENCE SYSTEM Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 13.7.1.4, “DROP ROLE Syntax”
Section 13.1.30, “DROP SERVER Syntax”
Section 13.1.31, “DROP SPATIAL REFERENCE SYSTEM Syntax”
Section 13.7.1.5, “DROP USER Syntax”

Section 18.6.6.4, “Exit Action”
Section 13.7.1.6, “GRANT Syntax”
Section 18.8, “Group Replication System Variables”
Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 12.15, “Information Functions”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 6.4.4.11, “Keyring System Variables”
Section 13.7.8.4, “KILL Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 6.4.8.1, “MySQL Enterprise Data Masking and De-Identification Components”
Section 6.4.8.4, “MySQL Enterprise Data Masking and De-Identification User-Defined Function Reference”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 28.7.7.12, “mysql_dump_debug_info()”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.4.4.9, “Plugin-Specific Keyring Key-Management Functions”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.7, “RENAME USER Syntax”
Section 13.7.8.7, “RESET PERSIST Syntax”
Section 13.7.1.8, “REVOKE Syntax”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 13.7.1.10, “SET PASSWORD Syntax”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 17.1.2, “Setting Up Binary Log File Position Based Replication”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Section 13.7.7.1, “SHOW BINARY LOGS Syntax”
Section 13.7.7.23, “SHOW MASTER STATUS Syntax”
Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.4.3.1, “START GROUP_REPLICATION Syntax”
Section 13.4.2.6, “START SLAVE Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 13.4.3.2, “STOP GROUP_REPLICATION Syntax”
Section 13.4.2.7, “STOP SLAVE Syntax”
Section 24.6, “Stored Object Access Control”
Section 24.7, “Stored Program Binary Logging”
Section 5.1.9.1, “System Variable Privileges”
Section 27.4.4.2, “The diagnostics() Procedure”
Section B.4.2.6, “Too many connections”
Section 6.2.10, “Using Roles”
Section 5.6.6.3, “Using Version Tokens”
Section 5.6.6.1, “Version Tokens Components”
Section 5.6.6.4, “Version Tokens Reference”
Section 21.4, “Working with InnoDB Cluster”

SYSTEM_USER

Section 6.2.11, “Account Categories”
Section 13.7.1.6, “GRANT Syntax”
Section 12.15, “Information Functions”

Section 13.7.8.4, “KILL Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”
Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
Section 24.6, “Stored Object Access Control”
Section 25.19, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 6.2.10, “Using Roles”
Section 1.4, “What Is New in MySQL 8.0”

SYSTEM_VARIABLES_ADMIN

Section 6.4.5.9, “Audit Log Reference”
Section 5.6.7.12, “Clone System Variables”
Section 5.6.7.3, “Cloning Remote Data”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 6.4.4.11, “Keyring System Variables”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.8.7, “RESET PERSIST Syntax”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 5.1.9.1, “System Variable Privileges”
Section 6.2.10, “Using Roles”

T

[index top]

TABLE_ENCRYPTION_ADMIN

Section 13.1.2, “ALTER DATABASE Syntax”
Section 13.1.9, “ALTER TABLE Syntax”
Section 13.1.10, “ALTER TABLESPACE Syntax”
Section 13.1.12, “CREATE DATABASE Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.1.36, “RENAME TABLE Syntax”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

TRIGGER

Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.1.34, “DROP TRIGGER Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.7.11, “SHOW CREATE TRIGGER Syntax”
Section 13.7.7.38, “SHOW TRIGGERS Syntax”

Section 25.34, “The INFORMATION_SCHEMA TRIGGERS Table”
Section 21.4, “Working with InnoDB Cluster”

U

[index top]

UPDATE

Section 6.2.11, “Account Categories”
Section 13.7.1.1, “ALTER USER Syntax”
Section 6.2.14, “Assigning Account Passwords”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 13.2.6, “INSERT Syntax”
Section 22.5.12.2, “NDB Cluster and MySQL Privileges”
Section 6.2.15, “Password Management”
Section 26.11, “Performance Schema General Table Characteristics”
Section 26.4, “Performance Schema Runtime Configuration”
Section 26.12.2, “Performance Schema Setup Tables”
Section 27.1, “Prerequisites for Using the sys Schema”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.7, “RENAME USER Syntax”
Section 13.7.1.8, “REVOKE Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.1.9, “SET DEFAULT ROLE Syntax”
Section 24.6, “Stored Object Access Control”
Section 16.7, “The MERGE Storage Engine”
Section 26.12.2.4, “The setup_objects Table”
Section 24.3.1, “Trigger Syntax and Examples”
Section 13.2.12, “UPDATE Syntax”
Section 6.2.10, “Using Roles”
Section 21.4, “Working with InnoDB Cluster”
Writing the Server-Side Authentication Plugin

USAGE

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”

V

[index top]

VERSION_TOKEN_ADMIN

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.6.6.3, “Using Version Tokens”
Section 5.6.6.1, “Version Tokens Components”
Section 5.6.6.4, “Version Tokens Reference”

X

[index top]

XA_RECOVER_ADMIN

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”
Section 13.3.8.1, “XA Transaction SQL Syntax”

SQL Modes Index

A | E | H | I | N | O | P | R | S | T

A

[index top]

ALLOW_INVALID_DATES

Section 12.7, “Date and Time Functions”
Section 11.3, “Date and Time Types”
Section B.4.4.2, “Problems Using DATE Columns”
Section 5.1.11, “Server SQL Modes”
Section 11.3.1, “The DATE, DATETIME, and TIMESTAMP Types”

ANSI

Section 9.2.5, “Function Name Parsing and Resolution”
Section 5.1.11, “Server SQL Modes”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”

ANSI_QUOTES

Section 4.5.1.6, “mysql Client Tips”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 28.7.7.56, “mysql_real_escape_string_quote()”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 8.9.3, “Optimizer Hints”
Section 9.2, “Schema Object Names”
Section 5.1.11, “Server SQL Modes”
Section 9.1.1, “String Literals”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

E

[index top]

ERROR_FOR_DIVISION_BY_ZERO

Section 12.25.3, “Expression Handling”
Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”
Section 12.25.5, “Precision Math Examples”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 5.1.11, “Server SQL Modes”

H

[index top]

HIGH_NOT_PRECEDENCE

Section 9.5, “Expressions”
Section 12.3.1, “Operator Precedence”
Section 5.1.11, “Server SQL Modes”

I

[index top]

IGNORE_SPACE

Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 9.2.5, “Function Name Parsing and Resolution”
Section 4.5.1.1, “mysql Client Options”
Section 5.1.11, “Server SQL Modes”

N

[index top]

NO_AUTO_CREATE_USER

Section 1.4, “What Is New in MySQL 8.0”

NO_AUTO_VALUE_ON_ZERO

Section 13.1.20, “CREATE TABLE Syntax”
Section 11.2.5, “Numeric Type Attributes”
Section 5.1.11, “Server SQL Modes”
Section 3.6.9, “Using AUTO_INCREMENT”

NO_BACKSLASH_ESCAPES

Section 12.17.4, “Functions That Modify JSON Values”
Section 28.7.7.55, “mysql_real_escape_string()”
Section 5.1.11, “Server SQL Modes”
Section 12.5.1, “String Comparison Functions and Operators”
Section 9.1.1, “String Literals”
Section 11.6, “The JSON Data Type”

NO_DIR_IN_CREATE

Section 13.1.20, “CREATE TABLE Syntax”
Section 17.5.1.10, “Replication and DIRECTORY Table Options”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.11, “Server SQL Modes”
Section 5.4.4, “The Binary Log”

NO_ENGINE_SUBSTITUTION

Section 13.1.9, “ALTER TABLE Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 5.6.1, “Installing and Uninstalling Plugins”
Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”

Section 5.1.11, “Server SQL Modes”
Section 16.1, “Setting the Storage Engine”
Section 17.4.4, “Using Replication with Different Master and Slave Storage Engines”

NO_UNSIGNED_SUBTRACTION

Section 12.6.1, “Arithmetic Operators”
Section 12.10, “Cast Functions and Operators”
Section 11.1.1, “Numeric Type Overview”
Section 11.2.6, “Out-of-Range and Overflow Handling”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 5.1.11, “Server SQL Modes”

NO_ZERO_DATE

Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.10, “Cast Functions and Operators”
Section 13.1.20, “CREATE TABLE Syntax”
Section 12.7, “Date and Time Functions”
Section 11.3, “Date and Time Types”
Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”
Section B.4.4.2, “Problems Using DATE Columns”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 11.3.1, “The DATE, DATETIME, and TIMESTAMP Types”

NO_ZERO_IN_DATE

Section 13.1.20, “CREATE TABLE Syntax”
Section 12.7, “Date and Time Functions”
Section 11.3, “Date and Time Types”
Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”
Section B.4.4.2, “Problems Using DATE Columns”
Section 5.1.11, “Server SQL Modes”

O

[index top]

ONLY_FULL_GROUP_BY

Section 3.3.4.8, “Counting Rows”
Section 12.20.2, “GROUP BY Modifiers”
Section 12.24, “Miscellaneous Functions”
Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”
Section 12.20.3, “MySQL Handling of GROUP BY”
Section 5.1.11, “Server SQL Modes”

P

[index top]

PAD_CHAR_TO_FULL_LENGTH

Section 5.1.11, “Server SQL Modes”
Section 11.1.3, “String Type Overview”
Section 11.4.1, “The CHAR and VARCHAR Types”
Section 1.4, “What Is New in MySQL 8.0”

PIPES_AS_CONCAT
Section 9.5, “Expressions”
Section 12.3.3, “Logical Operators”
Section 12.3.1, “Operator Precedence”
Section 5.1.11, “Server SQL Modes”
Section 1.4, “What Is New in MySQL 8.0”

R

[index top]

REAL_AS_FLOAT
Section 11.1.1, “Numeric Type Overview”
Section 11.2, “Numeric Types”
Section 5.1.11, “Server SQL Modes”

S

[index top]

STRICT_ALL_TABLES
Section 1.8.3.3, “Constraints on Invalid Data”
Section 12.25.3, “Expression Handling”
Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”
Section 5.1.11, “Server SQL Modes”
Section 17.5.3, “Upgrading a Replication Setup”

STRICT_TRANS_TABLES
Section 1.8.3.3, “Constraints on Invalid Data”
Section 12.25.3, “Expression Handling”
Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”
Section 5.1.11, “Server SQL Modes”
Section 17.5.3, “Upgrading a Replication Setup”

T

[index top]

TIME_TRUNCATE_FRACTIONAL
Section 11.3.5, “Fractional Seconds in Time Values”
Section 5.1.11, “Server SQL Modes”

TRADITIONAL
Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.25.3, “Expression Handling”
Section 13.2.7, “LOAD DATA Syntax”
Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”

Statement/Syntax Index

A | B | C | D | E | F | G | H | I | K | L | O | P | R | S | T | U | W | X

A

[index top]

ADD PARTITION

Section 15.12.1, “Online DDL Operations”

ALTER DATABASE

Section 13.1.2, “ALTER DATABASE Syntax”
Section 10.5, “Configuring Application Character Set and Collation”
Section 10.3.3, “Database Character Set and Collation”
Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”
Section 22.1.4, “What is New in NDB Cluster”

ALTER EVENT

Section 13.1.3, “ALTER EVENT Syntax”
Section 13.1.13, “CREATE EVENT Syntax”
Section 24.4.4, “Event Metadata”
Section 24.4.1, “Event Scheduler Overview”
Section 24.4.3, “Event Syntax”
Section 12.15, “Information Functions”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 17.5.1.16, “Replication of Invoked Features”
Section 24.8, “Restrictions on Stored Programs”
Section 13.7.7.18, “SHOW EVENTS Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.7, “Stored Program Binary Logging”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”
Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”

ALTER EVENT event_name ENABLE

Section 17.5.1.16, “Replication of Invoked Features”

ALTER FUNCTION

Section 13.1.4, “ALTER FUNCTION Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.7, “Stored Program Binary Logging”
Section 24.2.1, “Stored Routine Syntax”

ALTER IGNORE TABLE

Section 23.3.4, “Maintenance of Partitions”

ALTER INSTANCE RELOAD TLS

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 5.1.10, “Server Status Variables”

ALTER INSTANCE ROTATE BINLOG MASTER KEY

Section 17.3.2.3, “Binary Log Master Key Rotation”

ALTER INSTANCE ROTATE INNODB MASTER KEY

Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section A.17, “MySQL 8.0 FAQ: InnoDB Data-at-Rest Encryption”

ALTER LOGFILE GROUP

Section 13.1.6, “ALTER LOGFILE GROUP Syntax”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”

ALTER PROCEDURE

Section 13.1.7, “ALTER PROCEDURE Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.7, “Stored Program Binary Logging”
Section 24.2.1, “Stored Routine Syntax”

ALTER RESOURCE GROUP

Section 13.7.2.1, “ALTER RESOURCE GROUP Syntax”
Section 8.12.5, “Resource Groups”

ALTER SCHEMA

Section 13.1.2, “ALTER DATABASE Syntax”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”

ALTER SERVER

Section 13.1.1, “Atomic Data Definition Statement Support”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.5, “Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER”
Section 13.3.3, “Statements That Cause an Implicit Commit”

ALTER TABLE

Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 13.1.9.2, “ALTER TABLE and Generated Columns”
Section 13.1.9.3, “ALTER TABLE Examples”
Section 13.1.9.1, “ALTER TABLE Partition Operations”
Section 13.1.9, “ALTER TABLE Syntax”
Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 12.10, “Cast Functions and Operators”
Section 2.11.4, “Changes in MySQL 8.0”
Section 13.7.3.2, “CHECK TABLE Syntax”
Section 10.3.5, “Column Character Set and Collation”
Section 10.7, “Column Character Set Conversion”
Section 8.3.5, “Column Indexes”
Section 15.8.10, “Configuring Optimizer Statistics for InnoDB”
Configuring Optimizer Statistics Parameters for Individual Tables
Section 15.8.11, “Configuring the Merge Threshold for Index Pages”

Section 15.7.2.3, “Consistent Nonlocking Reads”
Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 10.9.8, “Converting Between 3-Byte and 4-Byte Unicode Character Sets”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 13.1.15, “CREATE INDEX Syntax”
Section 13.1.20.1, “CREATE TABLE Statement Retention”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 3.3.2, “Creating a Table”
Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”
Section 15.9.1.2, “Creating Compressed Tables”
Section 15.6.1.1, “Creating InnoDB Tables”
Section 11.5.6, “Creating Spatial Columns”
Section 11.5.10, “Creating Spatial Indexes”
Section 11.7, “Data Type Default Values”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 15.11.4, “Defragmenting a Table”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 13.1.27, “DROP INDEX Syntax”
Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 8.8.2, “EXPLAIN Output Format”
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 15.6.3.2, “File-Per-Table Tablespaces”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 15.20.2, “Forcing InnoDB Recovery”
Section 1.8.3.2, “FOREIGN KEY Constraints”
Section 12.9, “Full-Text Search Functions”
Section 15.6.3.3, “General Tablespaces”
Section 8.14.2, “General Thread States”
Section 13.7.1.6, “GRANT Syntax”
Section 15.9.1.5, “How Compression Works for InnoDB Tables”
Section B.4.3.4, “How MySQL Handles a Full Disk”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 12.15, “Information Functions”
Section 22.2.4, “Initial Configuration of NDB Cluster”
Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”
Section 8.3.8, “InnoDB and MyISAM Index Statistics Collection”
Section 15.12, “InnoDB and Online DDL”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.15, “InnoDB Integration with MySQL Performance Schema”
Section 15.9.2, “InnoDB Page Compression”
Section 15.10, “InnoDB Row Formats”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.9.1, “InnoDB Table Compression”
Section 8.3.12, “Invisible Indexes”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 13.7.8.4, “KILL Syntax”
Section B.4.7, “Known Issues in MySQL”
Section 22.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”
Section 22.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”
Section 8.4.6, “Limits on Table Size”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”

Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 23.3.4, “Maintenance of Partitions”
Section 23.3.2, “Management of HASH and KEY Partitions”
Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 12.9.9, “MeCab Full-Text Parser Plugin”
Section 16.7.2, “MERGE Table Problems”
Section 15.15.1, “Monitoring ALTER TABLE Progress for InnoDB Tables Using Performance Schema”
Section 15.6.1.2, “Moving or Copying InnoDB Tables”
Section 16.2.1, “MyISAM Startup Options”
Section 16.2.3, “MyISAM Table Storage Formats”
Section 4.6.4.1, “myisamchk General Options”
Section 1.8.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
MySQL Server Options for NDB Cluster
Section 28.7.7.36, “mysql_info()”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 22.2.6, “NDB Cluster Example with Tables and Data”
Section 22.6.4, “NDB Cluster Replication Schema and Tables”
NDB Cluster System Variables
Section 22.4.9, “`ndb_desc` — Describe NDB Tables”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 12.9.8, “ngram Full-Text Parser”
Section 22.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 15.12.5, “Online DDL Failure Conditions”
Section 15.12.6, “Online DDL Limitations”
Section 15.12.1, “Online DDL Operations”
Section 15.12.2, “Online DDL Performance and Concurrency”
Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”
Section 13.7.3.4, “OPTIMIZE TABLE Syntax”
Section 8.4.1, “Optimizing Data Size”
Section 11.2.6, “Out-of-Range and Overflow Handling”
Section 23.1, “Overview of Partitioning in MySQL”
Section 15.9.1.1, “Overview of Table Compression”
Section 23.3, “Partition Management”
Section 23.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”
Section 23.6.2, “Partitioning Limitations Relating to Storage Engines”
Section 26.12.5, “Performance Schema Stage Event Tables”
Section 2.11.5, “Preparing Your Installation for Upgrade”
Section 6.2.2, “Privileges Provided by MySQL”
Section B.4.6.1, “Problems with ALTER TABLE”
Section 23.2.3.1, “RANGE COLUMNS partitioning”
Section 23.2.1, “RANGE Partitioning”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”
Section 13.1.36, “RENAME TABLE Syntax”
Section 17.5.1.1, “Replication and AUTO_INCREMENT”
Section 17.5.1.26, “Replication and Reserved Words”
Replication with More Columns on Master or Slave
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 24.9, “Restrictions on Views”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section B.3.1, “Server Error Message Reference”

Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 13.1.20.11, “Setting NDB_TABLE Options”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 16.1, “Setting the Storage Engine”
Section 13.7.7.15, “SHOW ENGINE Syntax”
Section 13.7.7.22, “SHOW INDEX Syntax”
Section 13.7.7.37, “SHOW TABLES Syntax”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 13.1.20.8, “Silent Column Specification Changes”
Section 15.12.4, “Simplifying DDL Statements with Online DDL”
Section 15.9.1.7, “SQL Compression Syntax Warnings and Errors”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 11.1.3, “String Type Overview”
Section 10.3.4, “Table Character Set and Collation”
Section B.4.6.2, “TEMPORARY Table Problems”
Section 5.4.6, “The DDL Log”
Section 25.39.24, “The INFORMATION_SCHEMA INNODB_TABLES Table”
Section 25.17, “The INFORMATION_SCHEMA PARTITIONS Table”
Section 25.26, “The INFORMATION_SCHEMA STATISTICS Table”
Section 16.3, “The MEMORY Storage Engine”
Section 16.2, “The MyISAM Storage Engine”
Section 5.4.5, “The Slow Query Log”
Section 22.1.7.6, “Unsupported or Missing Features in NDB Cluster”
Section 3.6.9, “Using AUTO_INCREMENT”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”
Section 17.4.4, “Using Replication with Different Master and Slave Storage Engines”
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section 1.4, “What Is New in MySQL 8.0”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”
Section B.4.3.5, “Where MySQL Stores Temporary Files”

ALTER TABLE ... ADD COLUMN

Section 25.39.8, “The INFORMATION_SCHEMA INNODB_COLUMNS Table”

ALTER TABLE ... ADD FOREIGN KEY

Section 13.1.9, “ALTER TABLE Syntax”

ALTER TABLE ... ADD PARTITION

Section 23.3.1, “Management of RANGE and LIST Partitions”

ALTER TABLE ... ALGORITHM=COPY

Section 13.1.9, “ALTER TABLE Syntax”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

ALTER TABLE ... ALGORITHM=INPLACE

Section 13.1.9, “ALTER TABLE Syntax”
Section 15.12.6, “Online DDL Limitations”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION

Section 22.5.15.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”

ALTER TABLE ... AUTO_INCREMENT = N

Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”

ALTER TABLE ... COMPRESSION

Section 15.9.2, “InnoDB Page Compression”

ALTER TABLE ... COMPRESSION=None

Section 15.9.2, “InnoDB Page Compression”

ALTER TABLE ... CONVERT TO CHARACTER SET

Section 13.7.3.1, “ANALYZE TABLE Syntax”

ALTER TABLE ... DISABLE KEYS

Section 13.2.7, “LOAD DATA Syntax”

ALTER TABLE ... DISCARD PARTITION ... TABLESPACE

Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Transportable Tablespace Examples

ALTER TABLE ... DISCARD TABLESPACE

Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.6.3.3, “General Tablespaces”
MySQL Glossary
Section 25.39.25, “The INFORMATION_SCHEMA INNODB_TABLESPACES Table”
Transportable Tablespace Internals

ALTER TABLE ... DROP FOREIGN KEY

Section 13.1.9, “ALTER TABLE Syntax”

ALTER TABLE ... DROP PARTITION

Section 17.5.1.24, “Replication and Partitioning”

ALTER TABLE ... ENABLE KEYS

Section 13.2.7, “LOAD DATA Syntax”

ALTER TABLE ... ENCRYPTION

Section 13.1.5, “ALTER INSTANCE Syntax”

ALTER TABLE ... ENGINE

Section 5.1.8, “Server System Variables”

ALTER TABLE ... ENGINE = MEMORY

Section 17.5.1.21, “Replication and MEMORY Tables”

ALTER TABLE ... ENGINE permitted_engine

Section 5.1.8, “Server System Variables”

ALTER TABLE ... ENGINE=INNODB

Section 22.6.4, “NDB Cluster Replication Schema and Tables”
Section 1.4, “What Is New in MySQL 8.0”

ALTER TABLE ... ENGINE=NDB

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

ALTER TABLE ... EXCHANGE PARTITION

Section 13.1.9.1, “ALTER TABLE Partition Operations”
Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”

ALTER TABLE ... FORCE

Section 13.7.3.4, “OPTIMIZE TABLE Syntax”

ALTER TABLE ... IMPORT PARTITION ... TABLESPACE

Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Transportable Tablespace Examples

ALTER TABLE ... IMPORT TABLESPACE

Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 15.6.1.2, “Moving or Copying InnoDB Tables”
MySQL Glossary
Transportable Tablespace Examples
Transportable Tablespace Internals

ALTER TABLE ... OPTIMIZE PARTITION

Section 23.3.4, “Maintenance of Partitions”
Section 23.6.2, “Partitioning Limitations Relating to Storage Engines”

ALTER TABLE ... PARTITION BY

Section 23.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”

ALTER TABLE ... PARTITION BY ...

Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 23.6, “Restrictions and Limitations on Partitioning”

ALTER TABLE ... REMOVE PARTITIONING

Section 1.4, “What Is New in MySQL 8.0”

ALTER TABLE ... RENAME

Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”

ALTER TABLE ... REORGANIZE PARTITION

Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 22.5.15.1, “Adding NDB Cluster Data Nodes Online: General Issues”
Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 2.11.5, “Preparing Your Installation for Upgrade”

ALTER TABLE ... REPAIR PARTITION

Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 23.3.4, “Maintenance of Partitions”

ALTER TABLE ... TABLESPACE

Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.6.3.2, “File-Per-Table Tablespaces”

ALTER TABLE ... TRUNCATE PARTITION

Section 23.3.4, “Maintenance of Partitions”
Section 23.3, “Partition Management”

ALTER TABLE ... TRUNCATE PARTITION ALL

Section 23.3.4, “Maintenance of Partitions”

ALTER TABLE ...IMPORT TABLESPACE

Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.6.3.3, “General Tablespaces”

ALTER TABLE EXCHANGE PARTITION

Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”

ALTER TABLE mysql.ndb_apply_status ENGINE=MyISAM

Section 22.6.3, “Known Issues in NDB Cluster Replication”

ALTER TABLE t TRUNCATE PARTITION ()

Section 13.2.2, “DELETE Syntax”

ALTER TABLE t3 DROP PARTITION p2

Section 5.4.6, “The DDL Log”

ALTER TABLE table_name ENGINE=InnoDB;

Section 15.1.4, “Testing and Benchmarking with InnoDB”

ALTER TABLE table_name REORGANIZE PARTITION

Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”

ALTER TABLE tbl_name ENCRYPTION = 'Y'

Section A.17, “MySQL 8.0 FAQ: InnoDB Data-at-Rest Encryption”

ALTER TABLE tbl_name ENGINE=engine_name

Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 14.7, “Data Dictionary Usage Differences”
Section 15.13, “InnoDB Startup Options and System Variables”

ALTER TABLE tbl_name ENGINE=INNODB

Section 13.1.9, “ALTER TABLE Syntax”
Section 15.11.4, “Defragmenting a Table”

ALTER TABLE tbl_name FORCE

Section 13.1.9, “ALTER TABLE Syntax”
Section 15.11.4, “Defragmenting a Table”

ALTER TABLE tbl_name TABLESPACE tablespace_name

Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.6.3.3, “General Tablespaces”
MySQL Glossary

ALTER TABLESPACE

Section 13.1.10, “ALTER TABLESPACE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 13.1.33, “DROP TABLESPACE Syntax”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”
Section 1.4, “What Is New in MySQL 8.0”

ALTER TABLESPACE ... ADD DATAFILE

Section 13.1.21, “CREATE TABLESPACE Syntax”

ALTER TABLESPACE ... DROP DATAFILE

Section 13.1.33, “DROP TABLESPACE Syntax”

ALTER TABLESPACE ... DROP DATATFILE

Section 13.1.21, “CREATE TABLESPACE Syntax”

ALTER TABLESPACE ... ENCRYPTION

Section 15.12.1, “Online DDL Operations”

ALTER TABLESPACE ... ENGINE

Section 5.1.8, “Server System Variables”

ALTER TABLESPACE ... RENAME TO

Section 15.6.3.3, “General Tablespaces”
Section 15.12.1, “Online DDL Operations”
Section 1.4, “What Is New in MySQL 8.0”

ALTER TABLESPACE tablespace_name ENCRYPTION = 'Y'

Section A.17, “MySQL 8.0 FAQ: InnoDB Data-at-Rest Encryption”

ALTER UNDO TABLESPACE

Section 2.11.4, “Changes in MySQL 8.0”
Section 15.6.3.4, “Undo Tablespaces”
Section 1.4, “What Is New in MySQL 8.0”

ALTER UNDO TABLESPACE ... SET INACTIVE

Section 25.39.25, “The INFORMATION_SCHEMA INNODB_TABLESPACES Table”

ALTER UNDO TABLESPACE tablespace_name SET ACTIVE

Section 15.6.3.4, “Undo Tablespaces”

ALTER UNDO TABLESPACE tablespace_name SET INACTIVE

Section 15.6.3.4, “Undo Tablespaces”

ALTER USER

Section 6.2.6, “Access Control, Stage 1: Connection Verification”
Section 6.2.19, “Account Locking”
Section 13.7.1.1, “ALTER USER Syntax”

Section 6.2.14, “Assigning Account Passwords”
Section 2.11.4, “Changes in MySQL 8.0”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 13.7.1.2, “CREATE ROLE Syntax”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 6.5, “FIPS Support”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 2.10.1, “Initializing the Data Directory”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 6.2.15, “Password Management”
Section 6.4.3.2, “Password Validation Options and Variables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 6.2.18, “Proxy Users”
Resetting the Root Password: Generic Instructions
Section 6.2.16, “Server Handling of Expired Passwords”
Section 5.1.8, “Server System Variables”
Section 13.7.1.9, “SET DEFAULT ROLE Syntax”
Section 13.7.1.10, “SET PASSWORD Syntax”
Section 6.2.20, “Setting Account Resource Limits”
Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 6.4.3, “The Password Validation Component”
Section 29.2.1, “Types of Plugins”
Section 6.2.10, “Using Roles”
Section 1.4, “What Is New in MySQL 8.0”

ALTER USER ... ACCOUNT LOCK

Section B.3.1, “Server Error Message Reference”

ALTER USER ... ACCOUNT UNLOCK

Section B.3.1, “Server Error Message Reference”

ALTER USER ... DEFAULT ROLE

Section 13.7.1.1, “ALTER USER Syntax”
Section 13.7.1.9, “SET DEFAULT ROLE Syntax”

ALTER VIEW

Section 13.1.11, “ALTER VIEW Syntax”
Section 12.15, “Information Functions”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 24.8, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.5.2, “View Processing Algorithms”
Section 24.5.1, “View Syntax”

ANALYZE PARTITION

Section 15.12.1, “Online DDL Operations”

ANALYZE TABLE

Section 13.1.9, “ALTER TABLE Syntax”

Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 2.11.4, “Changes in MySQL 8.0”
Configuring Automatic Statistics Calculation for Persistent Optimizer Statistics
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 15.8.10, “Configuring Optimizer Statistics for InnoDB”
Configuring Optimizer Statistics Parameters for Individual Tables
Configuring the Number of Sampled Pages for InnoDB Optimizer Statistics
Section 13.1.15, “CREATE INDEX Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 14.7, “Data Dictionary Usage Differences”
Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Syntax”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 8.14.2, “General Thread States”
Including Delete-marked Records in Persistent Statistics Calculations
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 8.3.8, “InnoDB and MyISAM Index Statistics Collection”
InnoDB Persistent Statistics Tables
InnoDB Persistent Statistics Tables Example
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”
Section 23.3.4, “Maintenance of Partitions”
Section 16.7.2, “MERGE Table Problems”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.6.4.1, “myisamchk General Options”
Section 1.8.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 8.9.6, “Optimizer Statistics”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.1, “Optimizing SELECT Statements”
Section 6.2.2, “Privileges Provided by MySQL”
Section 8.2.1.2, “Range Optimization”
Section 17.5.1.13, “Replication and FLUSH”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 5.1.8, “Server System Variables”
Section 13.7.7.22, “SHOW INDEX Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 25.39.27, “The INFORMATION_SCHEMA INNODB_TABLESTATS View”
Section 25.26, “The INFORMATION_SCHEMA STATISTICS Table”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”
Section 5.3, “The mysql System Schema”
Section 5.4.5, “The Slow Query Log”

B

[index top]

BEGIN

Section 15.7.2.2, “autocommit, Commit, and Rollback”
Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”
Section 15.20.4, “InnoDB Error Handling”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
Section 26.12.7, “Performance Schema Transaction Tables”
Section 17.5.1.34, “Replication and Transactions”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 24.8, “Restrictions on Stored Programs”
Section 5.1.8, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.7, “Stored Program Binary Logging”
Section 26.12.7.1, “The events_transactions_current Table”

BEGIN ... END

Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”
Section 13.6.5.1, “CASE Syntax”
Section 13.6, “Compound-Statement Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.6.6.1, “Cursor CLOSE Syntax”
Section 13.6.6.3, “Cursor FETCH Syntax”
Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 13.6.3, “DECLARE Syntax”
Section 24.1, “Defining Stored Programs”
Section 24.4.1, “Event Scheduler Overview”
Section 13.6.5.4, “LEAVE Syntax”
Section 13.6.4.1, “Local Variable DECLARE Syntax”
Section 13.6.4.2, “Local Variable Scope and Resolution”
Section 24.8, “Restrictions on Stored Programs”
Section 13.6.7.6, “Scope Rules for Handlers”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 13.6.2, “Statement Label Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.3.1, “Trigger Syntax and Examples”

BINLOG

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 13.7.8.1, “BINLOG Syntax”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.3.3, “Replication Privilege Checks”
Section 5.1.8, “Server System Variables”

C

[index top]

CACHE INDEX

Section 13.7.8.2, “CACHE INDEX Syntax”

Section 8.10.2.4, “Index Preloading”
Section 13.7.8.5, “LOAD INDEX INTO CACHE Syntax”
Section 8.10.2.2, “Multiple Key Caches”
Section 13.3.3, “Statements That Cause an Implicit Commit”

CALL

Section 28.7.5, “C API Data Structures”
Section 28.7.23, “C API Multiple Statement Execution Support”
Section 28.7.25, “C API Prepared CALL Statement Support”
Section 28.7.26, “C API Prepared Statement Problems”
Section 13.2.1, “CALL Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.38, “mysql_insert_id()”
Section 28.7.7.46, “mysql_more_results()”
Section 28.7.7.47, “mysql_next_result()”
Section 28.7.7.54, “mysql_real_connect()”
Section 28.7.7.74, “mysql_set_server_option()”
Section 28.7.11.17, “mysql_stmt_next_result()”
Section 13.5, “Prepared SQL Statement Syntax”
Section 24.6, “Stored Object Access Control”
Chapter 24, *Stored Objects*
Section 24.7, “Stored Program Binary Logging”
Section 24.2.1, “Stored Routine Syntax”
Section 24.3.1, “Trigger Syntax and Examples”

CALL p()

Section 13.6.7.4, “RESIGNAL Syntax”

CASE

Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 13.6.5.1, “CASE Syntax”
Section 12.4, “Control Flow Functions”
Section 13.6.5, “Flow Control Statements”

CHANGE MASTER TO

Adding a Binary Log Based Master to a Multi-Source Replication Slave
Adding a GTID Based Master to a Multi-Source Replication Slave
Section 6.2.14, “Assigning Account Passwords”
Section 17.4.1.2, “Backing Up Raw Data from a Slave”
Section 17.1.1, “Binary Log File Position Based Replication Configuration Overview”
Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 2.11.4, “Changes in MySQL 8.0”
Section 17.1.7.1, “Checking Replication Status”
Section 5.6.7.6, “Cloning for Replication”
Section 17.2.3.1, “Commands for Operations on a Single Channel”
Section 17.2.3.2, “Compatibility with Previous Replication Statements”
Section 4.2.6, “Connection Compression Control”
Creating a Data Snapshot Using mysqldump
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 18.10, “Frequently Asked Questions”
Section 13.7.1.6, “GRANT Syntax”

Section 17.1.3.3, “GTID Auto-Positioning”
Section 22.6.8, “Implementing Failover with NDB Cluster Replication”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 22.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”
Section 26.12.11, “Performance Schema Replication Tables”
Section 22.6.5, “Preparing the NDB Cluster for Replication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 17.5.1.27, “Replication and Master or Slave Shutdowns”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.3.3, “Replication Privilege Checks”
Section 17.3, “Replication Security”
Section 8.14.6, “Replication Slave Connection Thread States”
Section 8.14.4, “Replication Slave I/O Thread States”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 8.14.5, “Replication Slave SQL Thread States”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 17.1.2.7, “Setting the Master Configuration on the Slave”
Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Setting Up Replication with Existing Data
Setting Up Replication with New Master and Slaves
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 17.2.4.2, “Slave Status Logs”
Section 13.4.2.6, “START SLAVE Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 17.4.8, “Switching Masters During Failover”
Section 15.19.7, “The InnoDB memcached Plugin and Replication”
Section 26.12.11.3, “The replication_applier_configuration Table”
Section 26.12.11.1, “The replication_connection_configuration Table”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”
Section 18.2.1.3, “User Credentials”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”
Section 1.4, “What Is New in MySQL 8.0”

CHANGE REPLICATION FILTER

Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.2.5.4, “Replication Channel Based Filters”
Section 5.1.10, “Server Status Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 26.12.11.8, “The replication_applier_filters Table”
Section 26.12.11.7, “The replication_applier_global_filters Table”

CHANGE REPLICATION FILTER REPLICATE_DO_DB

Section 17.1.6.3, “Replication Slave Options and Variables”

CHANGE REPLICATION FILTER REPLICATE_DO_TABLE

Section 17.1.6.3, “Replication Slave Options and Variables”

CHANGE REPLICATION FILTER REPLICATE_IGNORE_DB

Section 17.1.6.3, “Replication Slave Options and Variables”

CHANGE REPLICATION FILTER REPLICATE_IGNORE_TABLE

Section 17.1.6.3, “Replication Slave Options and Variables”

CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB

Section 17.1.6.3, “Replication Slave Options and Variables”

CHANGE REPLICATION FILTER REPLICATE_WILD_DO_TABLE

Section 17.1.6.3, “Replication Slave Options and Variables”

CHANGE REPLICATION FILTER REPLICATE_WILD_IGNORE_TABLE

Section 17.1.6.3, “Replication Slave Options and Variables”

CHECK PARTITION

Section 15.12.1, “Online DDL Operations”

CHECK TABLE

Section 13.1.9.1, “ALTER TABLE Partition Operations”
Section 13.7.3.2, “CHECK TABLE Syntax”
Section 16.2.4.1, “Corrupted MyISAM Tables”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.23, “CREATE VIEW Syntax”
Section 8.11.5, “External Locking”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 15.17.2, “InnoDB Recovery”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.20, “InnoDB Troubleshooting”
Section 23.3.4, “Maintenance of Partitions”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section A.6, “MySQL 8.0 FAQ: Views”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section B.4.2.8, “MySQL server has gone away”
Section 28.7.7.79, “mysql_store_result()”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 28.7.7.81, “mysql_use_result()”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 16.2.4.2, “Problems from Tables Not Being Closed Properly”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”
Section 16.4.1, “Repairing and Checking CSV Tables”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 13.6.6.5, “Restrictions on Server-Side Cursors”
Section 24.8, “Restrictions on Stored Programs”
Section 24.9, “Restrictions on Views”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 13.3.3, “Statements That Cause an Implicit Commit”

Section 16.5, “The ARCHIVE Storage Engine”
Section 16.7, “The MERGE Storage Engine”
Section 5.4.5, “The Slow Query Log”
Section 1.4, “What Is New in MySQL 8.0”
Section 2.11.3, “What the MySQL Upgrade Process Upgrades”

CHECK TABLE ... EXTENDED

Section 13.7.3.2, “CHECK TABLE Syntax”

CHECK TABLE ... FOR UPGRADE

Section 13.7.3.2, “CHECK TABLE Syntax”
Section 13.7.3.5, “REPAIR TABLE Syntax”

CHECK TABLE QUICK

Section 13.7.3.2, “CHECK TABLE Syntax”

CHECKSUM TABLE

Section 13.7.3.3, “CHECKSUM TABLE Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 17.5.1.4, “Replication and CHECKSUM TABLE”

CHECKSUM TABLE ... QUICK

Section 13.7.3.3, “CHECKSUM TABLE Syntax”

CLONE

Section 13.7.5, “CLONE Syntax”
Section 5.6.7.2, “Cloning Data Locally”
Section 5.6.7.3, “Cloning Remote Data”
Section 5.6.7.9, “Monitoring Cloning Operations”

CLONE INSTANCE

Section 5.6.7.13, “Clone Plugin Limitations”
Section 13.7.5, “CLONE Syntax”
Section 5.6.7.4, “Cloning Encrypted Data”
Section 5.6.7.3, “Cloning Remote Data”
Section 5.6.7.9, “Monitoring Cloning Operations”
Section 5.6.7.8, “Remote Cloning Operation Failure Handling”

CLONE LOCAL

Section 5.6.7.9, “Monitoring Cloning Operations”

CLONE LOCAL DATA DIRECTORY

Section 13.7.5, “CLONE Syntax”
Section 5.6.7.2, “Cloning Data Locally”

COALESCE PARTITION

Section 15.12.1, “Online DDL Operations”

COMMIT

Section 13.1.1, “Atomic Data Definition Statement Support”
Section 15.7.2.2, “autocommit, Commit, and Rollback”

Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Data Definition Statements
Section 15.2, “InnoDB and the ACID Model”
Section 15.20.4, “InnoDB Error Handling”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
NDB Cluster System Variables
Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 26.12.7, “Performance Schema Transaction Tables”
Section 17.5.1.34, “Replication and Transactions”
Section 17.1.6.3, “Replication Slave Options and Variables”
Rewriter Query Rewrite Plugin Procedures and Functions
Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 5.1.15, “Server Tracking of Client Session State Changes”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.7, “Stored Program Binary Logging”
Section 5.4.4, “The Binary Log”
Section 26.12.7.1, “The events_transactions_current Table”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”
Section 13.3, “Transactional and Locking Statements”
Section 24.3.1, “Trigger Syntax and Examples”

COMMIT AND CHAIN

Section 26.12.7, “Performance Schema Transaction Tables”
Section 5.1.15, “Server Tracking of Client Session State Changes”

COMPRESSION

Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”

CREATE DATABASE

Section 7.1, “Backup and Recovery Types”
Section 28.7.6, “C API Function Overview”
Section 10.5, “Configuring Application Character Set and Collation”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 13.1.12, “CREATE DATABASE Syntax”
Section 10.3.3, “Database Character Set and Collation”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 9.2.3, “Identifier Case Sensitivity”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 28.7.7.8, “mysql_create_db()”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 26.6, “Performance Schema Instrument Naming Conventions”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 10.3.2, “Server Character Set and Collation”
Section B.3.1, “Server Error Message Reference”
Section 5.1.8, “Server System Variables”
Section 13.7.7.6, “SHOW CREATE DATABASE Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 22.1.4, “What is New in NDB Cluster”

CREATE DATABASE IF NOT EXISTS

Section 17.5.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”

CREATE EVENT

Section 13.1.3, “ALTER EVENT Syntax”
Section 13.1.5, “ALTER INSTANCE Syntax”
Section 17.3.2.3, “Binary Log Master Key Rotation”
Section 13.1.13, “CREATE EVENT Syntax”
Section 24.4.4, “Event Metadata”
Section 24.4.3, “Event Syntax”
Section 9.5, “Expressions”
Section 12.15, “Information Functions”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 17.5.1.16, “Replication of Invoked Features”
Section 24.8, “Restrictions on Stored Programs”
Section 13.7.7.7, “SHOW CREATE EVENT Syntax”
Section 13.7.7.18, “SHOW EVENTS Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 24, *Stored Objects*
Section 24.7, “Stored Program Binary Logging”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”
Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”

CREATE EVENT IF NOT EXISTS

Section 17.5.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”

CREATE FULLTEXT INDEX

Section 8.5.5, “Bulk Data Loading for InnoDB Tables”

CREATE FUNCTION

Section 29.4, “Adding New Functions to MySQL”
Section 13.1.4, “ALTER FUNCTION Syntax”
Section 1.9.1, “Contributors to MySQL”
Section 13.1.14, “CREATE FUNCTION Syntax”
Section 13.7.4.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.7.4.2, “DROP FUNCTION Syntax”
Section 9.2.5, “Function Name Parsing and Resolution”
Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”

Section 12.15, “Information Functions”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.7.1, “Installing and Uninstalling User-Defined Functions”
Section 6.4.8.2, “Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification”
Installing or Uninstalling the UDF Locking Interface
Section 5.6.6.2, “Installing or Uninstalling Version Tokens”
Section 12.19.1, “MySQL Enterprise Encryption Installation”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 26.12.18, “Performance Schema Miscellaneous Tables”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 17.5.1.16, “Replication of Invoked Features”
Section 24.8, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 24, *Stored Objects*
Section 24.7, “Stored Program Binary Logging”
Section 24.2.1, “Stored Routine Syntax”
Section 26.12.18.6, “The user_defined_functions Table”
Section 29.4.2.1, “UDF Calling Sequences for Simple Functions”
Section 29.4.2.5, “UDF Compiling and Installing”
Section 29.4.2.6, “UDF Security Precautions”
Section 2.11.12, “Upgrade Troubleshooting”

CREATE INDEX

Section 12.10, “Cast Functions and Operators”
Section 8.3.5, “Column Indexes”
Section 15.8.11, “Configuring the Merge Threshold for Index Pages”
Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 13.1.15, “CREATE INDEX Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 11.5.10, “Creating Spatial Indexes”
Section 12.9, “Full-Text Search Functions”
Section 15.9.1.5, “How Compression Works for InnoDB Tables”
Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.3.12, “Invisible Indexes”
Section 12.9.9, “MeCab Full-Text Parser Plugin”
MySQL Glossary
Section 12.9.8, “ngram Full-Text Parser”
Section 15.12.1, “Online DDL Operations”
Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”
Section 8.7, “Optimizing for MEMORY Tables”
Section B.3.1, “Server Error Message Reference”
Section 5.1.8, “Server System Variables”
Section 13.7.7.22, “SHOW INDEX Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 25.26, “The INFORMATION_SCHEMA STATISTICS Table”
Section 5.4.5, “The Slow Query Log”
Section 22.1.7.6, “Unsupported or Missing Features in NDB Cluster”

CREATE LOGFILE GROUP

Section 13.1.6, “ALTER LOGFILE GROUP Syntax”
Section 13.1.16, “CREATE LOGFILE GROUP Syntax”

Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”
Section 22.5.10.31, “The ndbinfo resources Table”

CREATE OR REPLACE VIEW

Section 13.1.23, “CREATE VIEW Syntax”
Section 24.9, “Restrictions on Views”

CREATE PROCEDURE

Section 13.1.7, “ALTER PROCEDURE Syntax”
Section 13.2.1, “CALL Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 12.15, “Information Functions”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 17.5.1.16, “Replication of Invoked Features”
Section 24.8, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 24, *Stored Objects*
Section 24.7, “Stored Program Binary Logging”
Section 24.2.1, “Stored Routine Syntax”

CREATE RESOURCE GROUP

Section 13.7.2.1, “ALTER RESOURCE GROUP Syntax”
Section 13.7.2.2, “CREATE RESOURCE GROUP Syntax”
Section 8.12.5, “Resource Groups”

CREATE ROLE

Section 13.7.1.2, “CREATE ROLE Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 6.2.10, “Using Roles”

CREATE SCHEMA

Section 13.1.12, “CREATE DATABASE Syntax”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”

CREATE SERVER

Section 13.1.8, “ALTER SERVER Syntax”
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 16.8.2.2, “Creating a FEDERATED Table Using CREATE SERVER”
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 13.7.8.3, “FLUSH Syntax”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 16.8.2, “How to Create FEDERATED Tables”

Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.5, “Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER”
Section 13.3.3, “Statements That Cause an Implicit Commit”

CREATE SPATIAL REFERENCE SYSTEM

Section 13.1.19, “CREATE SPATIAL REFERENCE SYSTEM Syntax”
Section 11.5.5, “Spatial Reference System Support”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 25.28, “The INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS Table”

CREATE TABLE

Section 13.1.9.3, “ALTER TABLE Examples”
Section 13.1.9.1, “ALTER TABLE Partition Operations”
Section 13.1.9, “ALTER TABLE Syntax”
Chapter 16, *Alternative Storage Engines*
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 7.1, “Backup and Recovery Types”
Section 15.1.2, “Best Practices for InnoDB Tables”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 12.10, “Cast Functions and Operators”
Section 2.11.4, “Changes in MySQL 8.0”
Section 13.1.20.7, “CHECK Constraints”
Section 10.3.5, “Column Character Set and Collation”
Section 8.3.5, “Column Indexes”
Section 15.8.10, “Configuring Optimizer Statistics for InnoDB”
Configuring Optimizer Statistics Parameters for Individual Tables
Section 15.8.11, “Configuring the Merge Threshold for Index Pages”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 13.1.13, “CREATE EVENT Syntax”
Section 13.1.15, “CREATE INDEX Syntax”
Section 13.1.18, “CREATE SERVER Syntax”
Section 13.1.20.4, “CREATE TABLE ... LIKE Syntax”
Section 13.1.20.5, “CREATE TABLE ... SELECT Syntax”
Section 13.1.20.9, “CREATE TABLE and Generated Columns”
Section 13.1.20.1, “CREATE TABLE Statement Retention”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”
Section 16.8.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 3.3.2, “Creating a Table”
Section 15.9.1.2, “Creating Compressed Tables”
Section 15.6.1.1, “Creating InnoDB Tables”
Section 11.5.6, “Creating Spatial Columns”
Section 11.5.10, “Creating Spatial Indexes”
Section 7.2, “Database Backup Methods”
Section 10.3.3, “Database Character Set and Collation”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section B.2, “Error Information Interfaces”
Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 1.8.3.2, “FOREIGN KEY Constraints”
Section 12.9, “Full-Text Search Functions”

Section 15.6.3.3, “General Tablespaces”
Section 3.4, “Getting Information About Databases and Tables”
Section 17.1.3.1, “GTID Format and Storage”
Section 17.1.3.2, “GTID Life Cycle”
Section 23.2.4, “HASH Partitioning”
Section 13.8.3, “HELP Syntax”
Section 15.9.1.5, “How Compression Works for InnoDB Tables”
Section 23.2.7, “How MySQL Partitioning Handles NULL”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 9.2.3, “Identifier Case Sensitivity”
Section 12.15, “Information Functions”
Section 22.2.4, “Initial Configuration of NDB Cluster”
Section 15.6.1.5, “InnoDB and FOREIGN KEY Constraints”
Section 15.18, “InnoDB and MySQL Replication”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 15.9.2, “InnoDB Page Compression”
Section 15.10, “InnoDB Row Formats”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.9.1, “InnoDB Table Compression”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 15.1, “Introduction to InnoDB”
Section 8.3.12, “Invisible Indexes”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 12.17.6, “JSON Table Functions”
Section 23.2.5, “KEY Partitioning”
Section 22.1.7.5, “Limits Associated with Database Objects in NDB Cluster”
Section 8.4.6, “Limits on Table Size”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 23.2.2, “LIST Partitioning”
Section 13.2.8, “LOAD XML Syntax”
Section 3.3.3, “Loading Data into a Table”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 12.9.9, “MeCab Full-Text Parser Plugin”
Section 16.2.3, “MyISAM Table Storage Formats”
Section 4.5.1.1, “mysql Client Options”
Section 1.8.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.5.17, “NDB API Statistics Counters and Variables”
Section 22.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
NDB Cluster System Variables
Section 22.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 22.4.9, “`ndb_desc` — Describe NDB Tables”
Section 12.9.8, “ngram Full-Text Parser”
Section 22.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 15.12.1, “Online DDL Operations”
Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”
Section 8.4.1, “Optimizing Data Size”

Section 8.5.7, “Optimizing InnoDB DDL Operations”
Section 23.1, “Overview of Partitioning in MySQL”
Section 15.9.1.1, “Overview of Table Compression”
Section 23.3, “Partition Management”
Section 23.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”
Section 23.6.3, “Partitioning Limitations Relating to Functions”
Section 23.6.2, “Partitioning Limitations Relating to Storage Engines”
Section 23.2, “Partitioning Types”
Section 6.2.2, “Privileges Provided by MySQL”
Section 23.2.3.1, “RANGE COLUMNS partitioning”
Section 23.2.1, “RANGE Partitioning”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 13.2.9, “REPLACE Syntax”
Section 17.5.1.1, “Replication and AUTO_INCREMENT”
Section 17.5.1.3, “Replication and Character Sets”
Section 17.5.1.10, “Replication and DIRECTORY Table Options”
Section 17.5.1.14, “Replication and System Functions”
Section 17.5.1.7, “Replication of CREATE TABLE ... SELECT Statements”
Replication with More Columns on Master or Slave
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server Command Options”
Section B.3.1, “Server Error Message Reference”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 16.1, “Setting the Storage Engine”
Section 13.7.7.5, “SHOW COLUMNS Syntax”
Section 13.7.7.10, “SHOW CREATE TABLE Syntax”
Section 13.7.7.15, “SHOW ENGINE Syntax”
Section 13.7.7.22, “SHOW INDEX Syntax”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 13.1.20.8, “Silent Column Specification Changes”
Section 15.9.1.7, “SQL Compression Syntax Warnings and Errors”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 11.1.3, “String Type Overview”
Section 23.2.6, “Subpartitioning”
Section 10.3.4, “Table Character Set and Collation”
Section 15.1.4, “Testing and Benchmarking with InnoDB”
Section 16.5, “The ARCHIVE Storage Engine”
Section 5.6.5, “The ddl_rewriter Plugin”
Section 11.4.4, “The ENUM Type”
Section 25.39.24, “The INFORMATION_SCHEMA INNODB_TABLES Table”
Section 25.17, “The INFORMATION_SCHEMA PARTITIONS Table”
Section 25.26, “The INFORMATION_SCHEMA STATISTICS Table”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”
Section 16.3, “The MEMORY Storage Engine”
Section 16.2, “The MyISAM Storage Engine”
Section 13.2.11.1, “The Subquery as Scalar Operand”
Section 13.1.37, “TRUNCATE TABLE Syntax”
Section 13.7.4.6, “UNINSTALL PLUGIN Syntax”
Section 22.1.7.6, “Unsupported or Missing Features in NDB Cluster”
Section 3.6.9, “Using AUTO_INCREMENT”

Section 13.1.20.6, “Using FOREIGN KEY Constraints”
Section 3.3.4.9, “Using More Than one Table”
Section 7.4, “Using mysqldump for Backups”
Section 17.4.4, “Using Replication with Different Master and Slave Storage Engines”
Section 8.12.2, “Using Symbolic Links”
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section 1.4, “What Is New in MySQL 8.0”
Section 22.1.4, “What is New in NDB Cluster”
Section 2.3.7, “Windows Platform Restrictions”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”

CREATE TABLE ... DATA DIRECTORY

Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”
Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”

CREATE TABLE ... ENCRYPTION

Section 13.1.5, “ALTER INSTANCE Syntax”

CREATE TABLE ... LIKE

Section 13.1.15, “CREATE INDEX Syntax”
Section 13.1.20.4, “CREATE TABLE ... LIKE Syntax”
Section 13.1.20.9, “CREATE TABLE and Generated Columns”
Section 11.7, “Data Type Default Values”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 17.5.1.1, “Replication and AUTO_INCREMENT”
Section 16.7, “The MERGE Storage Engine”

CREATE TABLE ... ROW_FORMAT=COMPRESSED

Section 2.11.4, “Changes in MySQL 8.0”

CREATE TABLE ... SELECT

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 12.10, “Cast Functions and Operators”
Section 15.7.2.3, “Consistent Nonlocking Reads”
Section 13.1.20.5, “CREATE TABLE ... SELECT Syntax”
Section 13.1.20.9, “CREATE TABLE and Generated Columns”
Section 11.7, “Data Type Default Values”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.2, “GTID Life Cycle”
Section B.4.7, “Known Issues in MySQL”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 17.5.1.7, “Replication of CREATE TABLE ... SELECT Statements”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 1.8.2.1, “SELECT INTO TABLE Differences”
Section 5.1.11, “Server SQL Modes”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.7, “Stored Program Binary Logging”

CREATE TABLE ... SELECT ...

Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 23.3.1, “Management of RANGE and LIST Partitions”

CREATE TABLE ... TABLESPACE

Section 13.1.9, “ALTER TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”
Section 15.6.1.1, “Creating InnoDB Tables”
Section 15.6.3.2, “File-Per-Table Tablespaces”

CREATE TABLE dst_tbl LIKE src_tbl

Section 14.7, “Data Dictionary Usage Differences”

CREATE TABLE IF NOT EXISTS

Section 17.5.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”

CREATE TABLE IF NOT EXISTS ... LIKE

Section 17.5.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”

CREATE TABLE IF NOT EXISTS ... SELECT

Section 17.5.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”

CREATE TABLE LIKE

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

CREATE TABLE new_table SELECT ... FROM old_table ...

Section 13.1.20.5, “CREATE TABLE ... SELECT Syntax”
Section 13.2.10, “SELECT Syntax”

CREATE TABLE tbl_name ... TABLESPACE tablespace_name

Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.6.3.3, “General Tablespaces”
MySQL Glossary

CREATE TABLE...AS SELECT

Section 8.2.1, “Optimizing SELECT Statements”

CREATE TABLESPACE

Section 13.1.10, “ALTER TABLESPACE Syntax”
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 22.3.3.6, “Defining NDB Cluster Data Nodes”
Section 13.1.33, “DROP TABLESPACE Syntax”
Section 15.11.2, “File Space Management”
Section 15.6.3.3, “General Tablespaces”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 5.1.8, “Server System Variables”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”
Section 25.39.24, “The INFORMATION_SCHEMA INNODB_TABLES Table”
Section 1.4, “What Is New in MySQL 8.0”

CREATE TABLESPACE ... ADD DATAFILE

Section 2.11.4, “Changes in MySQL 8.0”

CREATE TEMPORARY TABLE

Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”
Section 15.9.1.2, “Creating Compressed Tables”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 13.7.1.6, “GRANT Syntax”
Section 13.2.5, “IMPORT TABLE Syntax”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 5.1.8, “Server System Variables”
Section 16.1, “Setting the Storage Engine”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section B.4.6.2, “TEMPORARY Table Problems”
Section 1.4, “What Is New in MySQL 8.0”

CREATE TRIGGER

Section 13.1.22, “CREATE TRIGGER Syntax”
Section 12.15, “Information Functions”
Section A.5, “MySQL 8.0 FAQ: Triggers”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 17.5.1.16, “Replication of Invoked Features”
Section 24.8, “Restrictions on Stored Programs”
Section 13.7.7.11, “SHOW CREATE TRIGGER Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 24, *Stored Objects*
Section 24.7, “Stored Program Binary Logging”
Section 24.3.1, “Trigger Syntax and Examples”

CREATE UNDO TABLESPACE

Section 13.1.33, “DROP TABLESPACE Syntax”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.6.3.4, “Undo Tablespaces”
Section 1.4, “What Is New in MySQL 8.0”

CREATE USER

Section 6.2, “Access Control and Account Management”
Section 6.2.11, “Account Categories”
Section 6.2.19, “Account Locking”

Section 6.2.1, “Account User Names and Passwords”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 6.2.14, “Assigning Account Passwords”
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.12.3, “Connecting Using the IPv6 Local Host Address”
Section 13.7.1.3, “CREATE USER Syntax”
Section 17.1.2.3, “Creating a User for Replication”
Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 6.5, “FIPS Support”
Section 13.7.8.3, “FLUSH Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 8.12.3.1, “How MySQL Uses Memory”
Implementing Proxy User Support in Authentication Plugins
Section 2.10.1, “Initializing the Data Directory”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.12, “IPv6 Support”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.5.12.2, “NDB Cluster and MySQL Privileges”
Section 6.4.1.8, “No-Login Pluggable Authentication”
Section 6.4.1.5, “PAM Pluggable Authentication”
Section 6.2.15, “Password Management”
Section 6.1.2.3, “Passwords and Logging”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 6.2.2, “Privileges Provided by MySQL”
Section 6.2.18, “Proxy Users”
Section 17.3.3, “Replication Privilege Checks”
Section 5.1.8, “Server System Variables”
Section 6.2.20, “Setting Account Resource Limits”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 13.7.7.12, “SHOW CREATE USER Syntax”
Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”
Section 6.2.4, “Specifying Account Names”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 5.3, “The mysql System Schema”
Section 6.4.3, “The Password Validation Component”
Section 29.2.1, “Types of Plugins”
Section 6.3, “Using Encrypted Connections”
Section 6.4.7.3, “Using MySQL Enterprise Firewall”
Section 6.2.10, “Using Roles”
Section 1.4, “What Is New in MySQL 8.0”
Section 6.4.1.6, “Windows Pluggable Authentication”

CREATE USER ... ACCOUNT LOCK

Section B.3.1, “Server Error Message Reference”

CREATE USER ... REQUIRE SUBJECT

Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”

CREATE VIEW

Section 13.1.11, “ALTER VIEW Syntax”
Section 13.1.23, “CREATE VIEW Syntax”
Section 8.14.2, “General Thread States”
Section 9.2.1, “Identifier Length Limits”
Section 12.15, “Information Functions”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 24.9, “Restrictions on Views”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 24, *Stored Objects*
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”
Section 24.5.3, “Updatable and Insertable Views”
Section 24.5.2, “View Processing Algorithms”
Section 24.5.1, “View Syntax”

D

[index top]

DEALLOCATE PREPARE

Section 13.5.3, “DEALLOCATE PREPARE Syntax”
Section 13.5.1, “PREPARE Syntax”
Section 13.5, “Prepared SQL Statement Syntax”
Section 24.8, “Restrictions on Stored Programs”
Section 5.1.10, “Server Status Variables”
Section 26.12.6.4, “The prepared_statements_instances Table”

DECLARE

Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.6.3, “DECLARE Syntax”
Section 13.6.7.3, “GET DIAGNOSTICS Syntax”
Section 13.6.7.5, “SIGNAL Syntax”
Section 13.6.4, “Variables in Stored Programs”

DECLARE ... CONDITION

Section 13.6.7, “Condition Handling”
Section 13.6.7.1, “DECLARE ... CONDITION Syntax”
Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 13.6.7.5, “SIGNAL Syntax”

DECLARE ... HANDLER

Section 13.6.7, “Condition Handling”
Section 13.6.7.1, “DECLARE ... CONDITION Syntax”
Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 13.6.7.5, “SIGNAL Syntax”

DEFAULT ENCRYPTION

Section 13.1.10, “ALTER TABLESPACE Syntax”

DEFAULT ENCRYPTION='N'

Section 13.1.10, “ALTER TABLESPACE Syntax”

DEFAULT ENCRYPTION='Y'

Section 13.1.10, “ALTER TABLESPACE Syntax”

DELETE

Section 6.2, “Access Control and Account Management”
Section 15.19.6.5, “Adapting DML Statements to memcached Operations”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.1.9.1, “ALTER TABLE Partition Operations”
Section 6.4.5.6, “Audit Log Filtering”
Section 6.4.5.9, “Audit Log Reference”
Section 15.1.2, “Best Practices for InnoDB Tables”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 28.7.6, “C API Function Overview”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 15.5.2, “Change Buffer”
Section 15.9.1.6, “Compression for OLTP Workloads”
Section 15.7.2.3, “Consistent Nonlocking Reads”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.1.23, “CREATE VIEW Syntax”
Section 13.2.2, “DELETE Syntax”
Section B.4.4.6, “Deleting Rows from Related Tables”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Syntax”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 15.20.2, “Forcing InnoDB Recovery”
Section 12.9.5, “Full-Text Restrictions”
Chapter 12, *Functions and Operators*
Section 8.14.2, “General Thread States”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 12.15, “Information Functions”
Section 15.18, “InnoDB and MySQL Replication”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.11.1, “Internal Locking Methods”
Section 25.1, “Introduction”
Section 13.2.10.2, “JOIN Syntax”
Section 9.3, “Keywords and Reserved Words”
Section 13.7.8.4, “KILL Syntax”
Section B.4.7, “Known Issues in MySQL”
Section 22.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 23.2.2, “LIST Partitioning”

Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 16.7.2, “MERGE Table Problems”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 1.8.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.49, “mysql_num_rows()”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.13, “mysql_stmt_field_count()”
Section 28.7.11.18, “mysql_stmt_num_rows()”
Section 22.5.12.3, “NDB Cluster and MySQL Security Procedures”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 22.4.8, “`ndb_delete_all` — Delete All Rows from an NDB Table”
Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”
Section 15.12.1, “Online DDL Operations”
Section 8.9.3, “Optimizer Hints”
Section 8.2.5, “Optimizing Data Change Statements”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.1, “Optimizing SELECT Statements”
Section 8.2.2, “Optimizing Subqueries, Derived Tables, View References, and Common Table Expressions”
Section 23.1, “Overview of Partitioning in MySQL”
Section 23.4, “Partition Pruning”
Section 23.5, “Partition Selection”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 6.2.2, “Privileges Provided by MySQL”
Section 15.8.9, “Purge Configuration”
Section 8.2.1.2, “Range Optimization”
Section 23.2.1, “RANGE Partitioning”
Section 17.5.1.18, “Replication and LIMIT”
Section 17.5.1.21, “Replication and MEMORY Tables”
Section 17.5.1.23, “Replication and the Query Optimizer”
Section 17.5.1.35, “Replication and Triggers”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.2.11.12, “Rewriting Subqueries as Joins”
Section 3.3.4.1, “Selecting All Data”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.11, “Server SQL Modes”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 8.3.3, “SPATIAL Index Optimization”
Section 13.2.11.10, “Subquery Errors”
Section 13.2.11, “Subquery Syntax”
Section 8.11.2, “Table Locking Issues”
Section 16.5, “The ARCHIVE Storage Engine”
Section 5.4.4, “The Binary Log”
Section 16.6, “The BLACKHOLE Storage Engine”
Section 25.39.27, “The INFORMATION_SCHEMA INNODB_TABLESTATS View”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”

Section 1.3.2, “The Main Features of MySQL”
Section 16.3, “The MEMORY Storage Engine”
Section 16.7, “The MERGE Storage Engine”
Section 5.6.4, “The Rewriter Query Rewrite Plugin”
Section 15.7.2.1, “Transaction Isolation Levels”
Section 24.3.1, “Trigger Syntax and Examples”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 13.1.37, “TRUNCATE TABLE Syntax”
Section 15.6.6, “Undo Logs”
Section 24.5.3, “Updatable and Insertable Views”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”
Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”
Section 1.4, “What Is New in MySQL 8.0”
Section 28.7.29.2, “What Results You Can Get from a Query”
Section 6.2.13, “When Privilege Changes Take Effect”
Section 8.2.1.1, “WHERE Clause Optimization”
Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”
Section 12.21.5, “Window Function Restrictions”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

DELETE FROM ... WHERE ...

Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”

DELETE FROM a.t

Section 17.1.6.3, “Replication Slave Options and Variables”

DESCRIBE

Section 28.7.5, “C API Data Structures”
Section 28.7.6, “C API Function Overview”
Section 13.1.20.1, “CREATE TABLE Statement Retention”
Section 3.3.2, “Creating a Table”
Section 13.8.1, “DESCRIBE Syntax”
Section 13.8.2, “EXPLAIN Syntax”
Section 25.42, “Extensions to SHOW Statements”
Section 3.4, “Getting Information About Databases and Tables”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 28.7.11.28, “mysql_stmt_store_result()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”
Section 13.7.7.5, “SHOW COLUMNS Syntax”
Section 13.1.20.8, “Silent Column Specification Changes”
Section 3.6.6, “Using Foreign Keys”
Section 10.2.2, “UTF-8 for Metadata”

DISCARD PARTITION

Section 15.12.1, “Online DDL Operations”

DISCARD PARTITION ... TABLESPACE

Section 13.1.9.1, “ALTER TABLE Partition Operations”

DO

Section 13.1.3, “ALTER EVENT Syntax”
Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”
Section 13.1.13, “CREATE EVENT Syntax”
Section 13.2.3, “DO Syntax”
Section 12.14, “Locking Functions”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 24.8, “Restrictions on Stored Programs”
Section 24.7, “Stored Program Binary Logging”
Section 13.2.11, “Subquery Syntax”
Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”

DROP DATABASE

Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 28.7.6, “C API Function Overview”
Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”
Section 13.1.24, “DROP DATABASE Syntax”
Section 13.1.33, “DROP TABLESPACE Syntax”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 15.6.3.3, “General Tablespaces”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 28.7.7.11, “mysql_drop_db()”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 2.11.5, “Preparing Your Installation for Upgrade”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 22.1.4, “What is New in NDB Cluster”
Section 2.3.7, “Windows Platform Restrictions”

DROP DATABASE IF EXISTS

Section 17.5.1.11, “Replication of DROP ... IF EXISTS Statements”

DROP EVENT

Section 24.4.3, “Event Syntax”
Section 17.5.1.16, “Replication of Invoked Features”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.7, “Stored Program Binary Logging”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”

DROP FUNCTION

Section 29.4, “Adding New Functions to MySQL”
Section 13.1.4, “ALTER FUNCTION Syntax”
Section 1.9.1, “Contributors to MySQL”
Section 13.7.4.1, “CREATE FUNCTION Syntax for User-Defined Functions”

Section 13.1.26, “DROP FUNCTION Syntax”
Section 13.7.4.2, “DROP FUNCTION Syntax”
Section 13.1.29, “DROP PROCEDURE and DROP FUNCTION Syntax”
Section 9.2.5, “Function Name Parsing and Resolution”
Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.7.1, “Installing and Uninstalling User-Defined Functions”
Section 6.4.8.2, “Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification”
Installing or Uninstalling the UDF Locking Interface
Section 5.6.6.2, “Installing or Uninstalling Version Tokens”
Section 12.19.1, “MySQL Enterprise Encryption Installation”
Section 17.5.1.16, “Replication of Invoked Features”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.7, “Stored Program Binary Logging”
Section 24.2.1, “Stored Routine Syntax”
Section 29.4.2.5, “UDF Compiling and Installing”
Section 29.4.2.6, “UDF Security Precautions”
Section 2.11.12, “Upgrade Troubleshooting”

DROP INDEX

Section 13.1.9, “ALTER TABLE Syntax”
Section 11.5.10, “Creating Spatial Indexes”
Section 13.1.27, “DROP INDEX Syntax”
Section 1.8.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 15.12.1, “Online DDL Operations”
Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”
Section 5.1.8, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 5.4.5, “The Slow Query Log”

DROP LOGFILE GROUP

Section 13.1.28, “DROP LOGFILE GROUP Syntax”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”

DROP PARTITION

Section 15.12.1, “Online DDL Operations”

DROP PREPARE

Section 26.12.6.4, “The prepared_statements_instances Table”

DROP PROCEDURE

Section 13.1.7, “ALTER PROCEDURE Syntax”
Section 6.4.7.2, “Installing or Uninstalling MySQL Enterprise Firewall”
Section 17.5.1.16, “Replication of Invoked Features”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.7, “Stored Program Binary Logging”
Section 24.2.1, “Stored Routine Syntax”

DROP RESOURCE GROUP

Section 13.7.2.3, “DROP RESOURCE GROUP Syntax”
Section 8.12.5, “Resource Groups”

DROP ROLE

Section 13.7.1.4, “DROP ROLE Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 6.2.10, “Using Roles”

DROP SCHEMA

Section 13.1.24, “DROP DATABASE Syntax”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 5.1.8, “Server System Variables”

DROP SERVER

Section 13.1.1, “Atomic Data Definition Statement Support”
Section 13.7.8.3, “FLUSH Syntax”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.5, “Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER”
Section 13.3.3, “Statements That Cause an Implicit Commit”

DROP SPATIAL REFERENCE SYSTEM

Section 13.1.31, “DROP SPATIAL REFERENCE SYSTEM Syntax”
Section 11.5.5, “Spatial Reference System Support”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 25.28, “The INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS Table”

DROP TABLE

Section 22.5.15.1, “Adding NDB Cluster Data Nodes Online: General Issues”
Section 13.1.9, “ALTER TABLE Syntax”
Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 6.4.5.4, “Audit Log File Formats”
Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 15.7.2.3, “Consistent Nonlocking Reads”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.1.32, “DROP TABLE Syntax”
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 15.6.3.2, “File-Per-Table Tablespaces”
Section 15.20.2, “Forcing InnoDB Recovery”
Section 15.6.3.3, “General Tablespaces”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.2, “GTID Life Cycle”
Section 12.15, “Information Functions”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 22.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 16.7.2, “MERGE Table Problems”
Section 4.5.1.1, “mysql Client Options”
Section 1.8.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 22.4.10, “ndb_drop_index — Drop Index from an NDB Table”
Section 22.4.11, “ndb_drop_table — Drop an NDB Table”
Section 8.5.7, “Optimizing InnoDB DDL Operations”
Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 24.9, “Restrictions on Views”
Section 13.6.7.6, “Scope Rules for Handlers”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 13.7.7.37, “SHOW TABLES Syntax”
Section 13.6.7.5, “SIGNAL Syntax”
Section 13.4.2.6, “START SLAVE Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 5.4.6, “The DDL Log”
Section 16.3, “The MEMORY Storage Engine”
Section 16.7, “The MERGE Storage Engine”
Section 13.6.7.7, “The MySQL Diagnostics Area”
Section 15.20.3, “Troubleshooting InnoDB Data Dictionary Operations”
Section 13.1.37, “TRUNCATE TABLE Syntax”
Section 29.2.1, “Types of Plugins”
Section 13.7.4.6, “UNINSTALL PLUGIN Syntax”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

DROP TABLE IF EXISTS

Section 17.5.1.11, “Replication of DROP ... IF EXISTS Statements”

DROP TABLESPACE

Section 13.1.1, “Atomic Data Definition Statement Support”
Section 15.6.3.3, “General Tablespaces”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

DROP TABLESPACE tablespace_name

Section 15.6.3.3, “General Tablespaces”

DROP TEMPORARY TABLE

Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”

DROP TRIGGER

Section 13.1.34, “DROP TRIGGER Syntax”
Section A.5, “MySQL 8.0 FAQ: Triggers”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 17.5.1.16, “Replication of Invoked Features”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.3.1, “Trigger Syntax and Examples”

DROP UNDO TABALESPACE

Section 15.6.3.4, “Undo Tablespaces”

DROP UNDO TABLESPACE
Section 2.11.4, “Changes in MySQL 8.0”
Section 1.4, “What Is New in MySQL 8.0”

DROP USER
Section 6.2.1, “Account User Names and Passwords”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”
Section 13.7.1.5, “DROP USER Syntax”
Section 13.7.8.3, “FLUSH Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 12.15, “Information Functions”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.5.12.2, “NDB Cluster and MySQL Privileges”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 13.7.1.8, “REVOKE Syntax”
Section 5.1.8, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”
Section 6.2.10, “Using Roles”
Section 1.4, “What Is New in MySQL 8.0”

DROP USER 'x'@'localhost'
Using the Authentication Plugins

DROP VIEW
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 13.1.35, “DROP VIEW Syntax”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 24.9, “Restrictions on Views”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.5.1, “View Syntax”

DROP VIEW IF EXISTS
Section 17.5.1.11, “Replication of DROP ... IF EXISTS Statements”

E

[index top]

ENCRYPTION
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”

EXCHANGE PARTITION
Section 15.12.1, “Online DDL Operations”

EXECUTE
Section 28.7.25, “C API Prepared CALL Statement Support”
Section 13.2.1, “CALL Syntax”
Section 13.5.2, “EXECUTE Syntax”

Section 13.5.1, “PREPARE Syntax”
Section 13.5, “Prepared SQL Statement Syntax”
Section 24.8, “Restrictions on Stored Programs”
Section 5.1.10, “Server Status Variables”
Section 26.12.6.4, “The prepared_statements_instances Table”

EXPLAIN

Section 13.1.9, “ALTER TABLE Syntax”
Section 8.2.1.23, “Avoiding Full Table Scans”
Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”
Section 28.7.5, “C API Data Structures”
Section 28.7.6, “C API Function Overview”
Section 8.3.5, “Column Indexes”
Section 8.2.1.13, “Condition Filtering”
Configuring the Number of Sampled Pages for InnoDB Optimizer Statistics
Section 13.1.15, “CREATE INDEX Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 29.5.1, “Debugging a MySQL Server”
Section 13.2.11.8, “Derived Tables”
Section 13.8.1, “DESCRIBE Syntax”
Section 8.2.1.18, “DISTINCT Optimization”
Section 8.2.1.5, “Engine Condition Pushdown Optimization”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Syntax”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 12.17.3, “Functions That Search JSON Values”
Section 8.2.1.17, “GROUP BY Optimization”
Section 8.2.1.6, “Index Condition Pushdown Optimization”
Section 8.9.4, “Index Hints”
Section 8.2.1.3, “Index Merge Optimization”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 25.1, “Introduction”
Section 8.3.12, “Invisible Indexes”
Section 8.2.1.15, “IS NULL Optimization”
Section 12.24, “Miscellaneous Functions”
Section 8.2.1.11, “Multi-Range Read Optimization”
Section 4.5.1.6, “mysql Client Tips”
Chapter 26, *MySQL Performance Schema*
Section 28.7.11.28, “mysql_stmt_store_result()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”
NDB Cluster Status Variables
NDB Cluster System Variables
Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”
Section 23.3.5, “Obtaining Information About Partitions”
Section 8.9.3, “Optimizer Hints”
Section 8.9.6, “Optimizer Statistics”
Section 8.3.11, “Optimizer Use of Generated Column Indexes”
Section B.4.5, “Optimizer-Related Issues”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”

Section 8.2.4, “Optimizing Performance Schema Queries”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.1, “Optimizing SELECT Statements”
Section 13.2.11.11, “Optimizing Subqueries”
Section 8.2.2.2, “Optimizing Subqueries with Materialization”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 8.2.1.16, “ORDER BY Optimization”
Section 23.4, “Partition Pruning”
Section 6.2.2, “Privileges Provided by MySQL”
Section 8.2.1.2, “Range Optimization”
Section 24.8, “Restrictions on Stored Programs”
Section 13.1.20.10, “Secondary Indexes and Generated Columns”
Section 13.2.10, “SELECT Syntax”
Section B.3.1, “Server Error Message Reference”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section B.4.4.7, “Solving Problems with No Matching Rows”
Section 26.12.17.3, “Statement Summary Tables”
Section 1.3.2, “The Main Features of MySQL”
Section 27.4.4.22, “The ps_trace_statement_digest() Procedure”
Section 8.8, “Understanding the Query Execution Plan”
Section 8.3.10, “Use of Index Extensions”
Section 29.5.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 11.5.11, “Using Spatial Indexes”
Section 8.3.7, “Verifying Index Usage”
Section 1.4, “What Is New in MySQL 8.0”
Section 8.2.1.21, “Window Function Optimization”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”

EXPLAIN ... FOR CONNECTION

Section 13.8.2, “EXPLAIN Syntax”

EXPLAIN ANALYZE

Section 8.2.1.4, “Hash Join Optimization”
Section 1.4, “What Is New in MySQL 8.0”

EXPLAIN FOR CONNECTION

Section 8.8.2, “EXPLAIN Output Format”
Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”
Section 8.2.1.2, “Range Optimization”
Section 5.1.10, “Server Status Variables”

EXPLAIN FORMAT=JSON

Section 8.2.1.21, “Window Function Optimization”

EXPLAIN FORMAT=TREE

Section 8.2.1.4, “Hash Join Optimization”
Section 1.4, “What Is New in MySQL 8.0”

EXPLAIN SELECT

Section 13.2.11.8, “Derived Tables”
Section 8.8.2, “EXPLAIN Output Format”
Section 15.7.5.3, “How to Minimize and Handle Deadlocks”

Section 1.7, “How to Report Bugs or Problems”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 23.3.5, “Obtaining Information About Partitions”

EXPLAIN SELECT COUNT()

Section 23.2.1, “RANGE Partitioning”

EXPLAIN tbl_name

Section 8.8.1, “Optimizing Queries with EXPLAIN”

F

[index top]

FETCH

Section 13.6.6.2, “Cursor DECLARE Syntax”
Section 13.6.6.3, “Cursor FETCH Syntax”
Section 24.8, “Restrictions on Stored Programs”

FETCH ... INTO var_list

Section 13.6.4, “Variables in Stored Programs”

FLUSH

Section 7.3.1, “Establishing a Backup Policy”
Section 13.7.8.3, “FLUSH Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 28.7.7.58, “mysql_refresh()”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.13, “Replication and FLUSH”
Section 13.7.8.6, “RESET Syntax”
Section 24.8, “Restrictions on Stored Programs”
Section 5.1.16, “Server Response to Signals”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 1.4, “What Is New in MySQL 8.0”

FLUSH BINARY LOGS

Section 13.7.8.3, “FLUSH Syntax”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 5.4.7, “Server Log Maintenance”

FLUSH ENGINE LOGS

Section 13.7.8.3, “FLUSH Syntax”

FLUSH ERROR LOGS

Section 5.4.2.9, “Error Log File Flushing and Renaming”
Section 13.7.8.3, “FLUSH Syntax”

FLUSH GENERAL LOGS

Section 13.7.8.3, “FLUSH Syntax”

FLUSH HOSTS

Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section 13.7.8.3, “FLUSH Syntax”
Section B.4.2.5, “Host 'host_name' is blocked”
Section 28.7.7.58, “mysql_refresh()”
Section 5.1.8, “Server System Variables”
Section 26.12.18.1, “The host_cache Table”

FLUSH LOGS

Section 7.3.3, “Backup Strategy Summary”
Section 7.2, “Database Backup Methods”
Section 17.1.5.3, “Disabling GTID Transactions Online”
Section 17.1.5.2, “Enabling GTID Transactions Online”
Section 5.4.2.9, “Error Log File Flushing and Renaming”
Section 7.3.1, “Establishing a Backup Policy”
Section 13.7.8.3, “FLUSH Syntax”
Section 5.4, “MySQL Server Logs”
Section 28.7.7.58, “mysql_refresh()”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 17.5.1.13, “Replication and FLUSH”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.4.7, “Server Log Maintenance”
Section 5.1.10, “Server Status Variables”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Section 17.2.4.1, “The Slave Relay Log”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”
Section 22.1.4, “What is New in NDB Cluster”

FLUSH OPTIMIZER_COSTS

Section 13.7.8.3, “FLUSH Syntax”
Section 8.9.5, “The Optimizer Cost Model”

FLUSH PRIVILEGES

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 13.7.8.3, “FLUSH Syntax”
Section 6.2.3, “Grant Tables”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 28.7.7.58, “mysql_refresh()”
Section 28.7.7.59, “mysql_reload()”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 22.5.12.3, “NDB Cluster and MySQL Security Procedures”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.13, “Replication and FLUSH”
Section 5.1.7, “Server Command Options”
Section 6.2.20, “Setting Account Resource Limits”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 1.2, “Typographical and Syntax Conventions”
Section 6.2.10, “Using Roles”
Section 6.2.13, “When Privilege Changes Take Effect”

FLUSH RELAY LOGS

Section 17.2.3.1, “Commands for Operations on a Single Channel”

Section 13.7.8.3, “FLUSH Syntax”

FLUSH RELAY LOGS FOR CHANNEL channel

Section 13.7.8.3, “FLUSH Syntax”

FLUSH SLOW LOGS

Section 13.7.8.3, “FLUSH Syntax”

FLUSH STATUS

Section 13.7.8.3, “FLUSH Syntax”
Section 28.7.7.58, “mysql_refresh()”
Section 26.12.14, “Performance Schema Status Variable Tables”
Section 5.1.10, “Server Status Variables”
Section 26.12.17.12, “Status Variable Summary Tables”
Section 8.3.10, “Use of Index Extensions”

FLUSH TABLE

Section 13.7.8.3, “FLUSH Syntax”

FLUSH TABLES

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 13.7.8.3, “FLUSH Syntax”
Section 8.14.2, “General Thread States”
Section 13.2.4, “HANDLER Syntax”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 16.7.2, “MERGE Table Problems”
Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 28.7.7.58, “mysql_refresh()”
Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 16.2.4.2, “Problems from Tables Not Being Closed Properly”
Section 17.5.1.13, “Replication and FLUSH”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 8.3.10, “Use of Index Extensions”

FLUSH TABLES ... FOR EXPORT

Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”
Section 13.7.8.3, “FLUSH Syntax”
Section 15.6.1.2, “Moving or Copying InnoDB Tables”
MySQL Glossary
Transportable Tablespace Examples
Transportable Tablespace Internals

FLUSH TABLES ...FOR EXPORT

Section 13.7.8.3, “FLUSH Syntax”

FLUSH TABLES tbl_name ...

Section 13.7.8.3, “FLUSH Syntax”

FLUSH TABLES tbl_name ... FOR EXPORT
Section 13.7.8.3, “FLUSH Syntax”

FLUSH TABLES tbl_name ... WITH READ LOCK
Section 13.7.8.3, “FLUSH Syntax”
Section 13.3.5, “LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Syntax”

FLUSH TABLES tbl_name WITH READ LOCK
Section 13.2.4, “HANDLER Syntax”

FLUSH TABLES WITH READ LOCK
Section 13.1.10, “ALTER TABLESPACE Syntax”
Section 7.2, “Database Backup Methods”
Section 7.3.1, “Establishing a Backup Policy”
Section 13.7.8.3, “FLUSH Syntax”
Section 15.6.3.3, “General Tablespaces”
Section 8.14.2, “General Thread States”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 17.5.1.13, “Replication and FLUSH”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 26.12.12.3, “The metadata_locks Table”

FLUSH USER_RESOURCES
Section 13.7.8.3, “FLUSH Syntax”
Section 6.2.20, “Setting Account Resource Limits”

G

[index top]

GET DIAGNOSTICS
Section 13.6.7, “Condition Handling”
Section B.2, “Error Information Interfaces”
Section 13.6.7.3, “GET DIAGNOSTICS Syntax”
Section 13.6.7.4, “RESIGNAL Syntax”
Section 13.6.8, “Restrictions on Condition Handling”
Section 24.8, “Restrictions on Stored Programs”
Section 5.1.8, “Server System Variables”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 13.6.7.5, “SIGNAL Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”

GET STACKED DIAGNOSTICS
Section 13.6.7.3, “GET DIAGNOSTICS Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”

GRANT
Section 6.2, “Access Control and Account Management”

Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 6.2.11, “Account Categories”
Section 6.2.1, “Account User Names and Passwords”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 5.1.12.3, “Connecting Using the IPv6 Local Host Address”
Section 13.7.1.3, “CREATE USER Syntax”
Section 17.1.2.3, “Creating a User for Replication”
Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”
Section 13.7.8.3, “FLUSH Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 8.12.3.1, “How MySQL Uses Memory”
Implementing Proxy User Support in Authentication Plugins
Section 12.15, “Information Functions”
Section 2.10.1, “Initializing the Data Directory”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.12, “IPv6 Support”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section A.14, “MySQL 8.0 FAQ: Replication”
MySQL Glossary
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.5.12.2, “NDB Cluster and MySQL Privileges”
Section 8.2.6, “Optimizing Database Privileges”
Section 6.1.2.3, “Passwords and Logging”
Section 22.6.5, “Preparing the NDB Cluster for Replication”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 6.2.2, “Privileges Provided by MySQL”
Section 6.2.18, “Proxy Users”
Section 17.5.1.13, “Replication and FLUSH”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 17.5.1.22, “Replication of the mysql System Schema”
Section 17.3.3, “Replication Privilege Checks”
Section 13.7.1.8, “REVOKE Syntax”
Section 6.1.1, “Security Guidelines”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 13.7.7.21, “SHOW GRANTS Syntax”
Section 6.2.4, “Specifying Account Names”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 5.1.9.1, “System Variable Privileges”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”
Section 5.3, “The mysql System Schema”
Section 6.4.7.3, “Using MySQL Enterprise Firewall”
Section 6.2.10, “Using Roles”
Section 20.5.3, “Using Secure Connections with X Plugin”
Section 1.4, “What Is New in MySQL 8.0”
Section 22.1.4, “What is New in NDB Cluster”
Section 6.2.13, “When Privilege Changes Take Effect”
Section 6.4.1.6, “Windows Pluggable Authentication”

GRANT ALL
Section 13.7.1.6, “GRANT Syntax”

GRANT EVENT
Section 24.4.6, “The Event Scheduler and MySQL Privileges”

GROUP BY
Section 15.1.1, “Benefits of Using InnoDB Tables”

H

[index top]

HANDLER
Section 28.7.28, “C API Automatic Reconnection Control”
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 13.7.8.3, “FLUSH Syntax”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 1.8, “MySQL Standards Compliance”
Section 28.7.7.3, “mysql_change_user()”
Section 28.7.7.60, “mysql_reset_connection()”
Section 5.1.8, “Server System Variables”

HANDLER ... CLOSE
Section 13.7.7.24, “SHOW OPEN TABLES Syntax”

HANDLER ... OPEN
Section 13.7.7.24, “SHOW OPEN TABLES Syntax”

HANDLER ... READ
Section 24.8, “Restrictions on Stored Programs”

HANDLER OPEN
Section 13.2.4, “HANDLER Syntax”
Section B.3.1, “Server Error Message Reference”
Section 13.1.37, “TRUNCATE TABLE Syntax”

HELP
Section 13.8.3, “HELP Syntax”
Section 2.10.1, “Initializing the Data Directory”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 5.1.14, “Server-Side Help Support”

I

[index top]

IF
Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 12.4, “Control Flow Functions”
Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.2, “IF Syntax”

IMPORT PARTITION

Section 15.12.1, “Online DDL Operations”

IMPORT PARTITION ... TABLESPACE

Section 13.1.9.1, “ALTER TABLE Partition Operations”

IMPORT TABLE

Section 13.2.5, “IMPORT TABLE Syntax”
Section 13.2.7, “LOAD DATA Syntax”
MySQL Glossary
Section 14.6, “Serialized Dictionary Information (SDI)”

INSERT

Section 6.2, “Access Control and Account Management”
Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 6.2.11, “Account Categories”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 6.4.5.6, “Audit Log Filtering”
Section 6.4.5.9, “Audit Log Reference”
Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 15.7.2.2, “autocommit, Commit, and Rollback”
Section 7.1, “Backup and Recovery Types”
Section 15.1.2, “Best Practices for InnoDB Tables”
Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 28.7.6, “C API Function Overview”
Section 28.7.23, “C API Multiple Statement Execution Support”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 15.5.2, “Change Buffer”
Section 2.11.4, “Changes in MySQL 8.0”
Section 13.1.20.7, “CHECK Constraints”
Section 10.7, “Column Character Set Conversion”
Section 15.9.1.6, “Compression for OLTP Workloads”
Section 8.11.3, “Concurrent Inserts”
Section 1.8.3.3, “Constraints on Invalid Data”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 13.1.15, “CREATE INDEX Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.20.9, “CREATE TABLE and Generated Columns”
Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.1.23, “CREATE VIEW Syntax”
Section 16.8.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 11.7, “Data Type Default Values”
Section 11.1.2, “Date and Time Type Overview”
Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 13.2.2, “DELETE Syntax”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 7.3.1, “Establishing a Backup Policy”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Syntax”

Section 12.25.3, “Expression Handling”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 15.20.2, “Forcing InnoDB Recovery”
Section 12.9.5, “Full-Text Restrictions”
Section 8.14.2, “General Thread States”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 28.7.29.3, “How to Get the Unique ID for the Last Inserted Row”
Section 12.15, “Information Functions”
Section 15.7.1, “InnoDB Locking”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 13.2.6.1, “INSERT ... SELECT Syntax”
Section 13.2.6.3, “INSERT DELAYED Syntax”
Section 13.2.6, “INSERT Syntax”
Section 8.11.1, “Internal Locking Methods”
Section 25.1, “Introduction”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 23.2.2, “LIST Partitioning”
Section 13.2.7, “LOAD DATA Syntax”
Section 3.3.3, “Loading Data into a Table”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 16.7.2, “MERGE Table Problems”
Section 8.11.4, “Metadata Locking”
Section 12.24, “Miscellaneous Functions”
Section A.1, “MySQL 8.0 FAQ: General”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section A.5, “MySQL 8.0 FAQ: Triggers”
Section A.6, “MySQL 8.0 FAQ: Views”
Section 4.5.1.1, “mysql Client Options”
Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”
Section 1.8.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section B.4.2.8, “MySQL server has gone away”
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.38, “mysql_insert_id()”
Section 28.7.7.49, “mysql_num_rows()”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.13, “mysql_stmt_field_count()”
Section 28.7.11.16, “mysql_stmt_insert_id()”
Section 28.7.11.18, “mysql_stmt_num_rows()”
Section 28.7.11.21, “mysql_stmt_prepare()”
Section 28.7.7.79, “mysql_store_result()”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.5.17, “NDB API Statistics Counters and Variables”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 22.2.6, “NDB Cluster Example with Tables and Data”
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”
Section 15.12.1, “Online DDL Operations”
Section 13.7.3.4, “OPTIMIZE TABLE Syntax”
Section 8.9.3, “Optimizer Hints”
Section 8.2.5, “Optimizing Data Change Statements”
Section 8.2.5.1, “Optimizing INSERT Statements”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 11.2.6, “Out-of-Range and Overflow Handling”
Section 23.1, “Overview of Partitioning in MySQL”
Section 23.4, “Partition Pruning”
Section 23.5, “Partition Selection”
Section 6.1.2.3, “Passwords and Logging”
Section 26.12.6, “Performance Schema Statement Event Tables”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 11.5.7, “Populating Spatial Columns”
Section 26.4.6, “Pre-Filtering by Thread”
Section 1.8.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 6.2.2, “Privileges Provided by MySQL”
Section 15.8.9, “Purge Configuration”
Section 23.2.1, “RANGE Partitioning”
Section 13.2.9, “REPLACE Syntax”
Section 17.5.1.1, “Replication and AUTO_INCREMENT”
Section 17.5.1.29, “Replication and Server SQL Mode”
Section 17.5.1.14, “Replication and System Functions”
Section 17.5.1.35, “Replication and Triggers”
Section 17.5.1.38, “Replication and Variables”
Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.2.5.3, “Replication Rule Application”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 13.1.20.10, “Secondary Indexes and Generated Columns”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section B.3.1, “Server Error Message Reference”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 13.7.7.27, “SHOW PROCEDURE CODE Syntax”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 17.5.1.28, “Slave Errors During Replication”
Section 8.3.3, “SPATIAL Index Optimization”
Section 24.7, “Stored Program Binary Logging”
Section 13.2.11, “Subquery Syntax”
Section 8.11.2, “Table Locking Issues”
Section 16.5, “The ARCHIVE Storage Engine”
Section 10.8.5, “The binary Collation Compared to _bin Collations”
Section 5.4.4, “The Binary Log”
Section 16.6, “The BLACKHOLE Storage Engine”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”
Section 1.3.2, “The Main Features of MySQL”
Section 16.7, “The MERGE Storage Engine”
Section 16.2, “The MyISAM Storage Engine”
Section 5.6.4, “The Rewriter Query Rewrite Plugin”
Section 5.1.17, “The Server Shutdown Process”

Section 24.3.1, “Trigger Syntax and Examples”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 15.19.9, “Troubleshooting the InnoDB memcached Plugin”
Section 15.6.6, “Undo Logs”
Section 24.5.3, “Updatable and Insertable Views”
Section 13.2.12, “UPDATE Syntax”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”
Section 15.14.2.1, “Using InnoDB Transaction and Locking Information”
Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”
Section 24.3, “Using Triggers”
Section 1.4, “What Is New in MySQL 8.0”
Section 28.7.29.2, “What Results You Can Get from a Query”
Section 6.2.13, “When Privilege Changes Take Effect”
Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”
Section 29.2.4.8, “Writing Audit Plugins”

INSERT ... ON DUPLICATE KEY UPDATE

Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 12.15, “Information Functions”
Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 13.2.6, “INSERT Syntax”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 16.7.2, “MERGE Table Problems”
Section 12.24, “Miscellaneous Functions”
MySQL Glossary
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.38, “mysql_insert_id()”

INSERT ... SELECT

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 8.11.3, “Concurrent Inserts”
Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 13.2.6.1, “INSERT ... SELECT Syntax”
Section 13.2.6, “INSERT Syntax”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section B.4.7, “Known Issues in MySQL”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 28.7.7.38, “mysql_insert_id()”
NDB Cluster System Variables
Section 23.5, “Partition Selection”
Section 17.5.1.18, “Replication and LIMIT”
Section 5.1.8, “Server System Variables”
Section 5.4.4, “The Binary Log”

INSERT ... SELECT ON DUPLICATE KEY UPDATE

Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 13.2.6.1, “INSERT ... SELECT Syntax”

INSERT ... SET

Section 13.2.6, “INSERT Syntax”

INSERT ... VALUES

Section 13.2.6, “INSERT Syntax”
Section 28.7.7.36, “mysql_info()”

INSERT DELAYED

Section 13.2.6.3, “INSERT DELAYED Syntax”
Section 13.2.6, “INSERT Syntax”

INSERT IGNORE

Section 13.1.20.7, “CHECK Constraints”
Section 1.8.3.3, “Constraints on Invalid Data”
Section 1.8.3.4, “ENUM and SET Constraints”
Section 12.15, “Information Functions”
Section 13.2.6, “INSERT Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 5.1.11, “Server SQL Modes”

INSERT IGNORE ... SELECT

Section 13.2.6.1, “INSERT ... SELECT Syntax”

INSERT INTO ... SELECT

Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 15.7.2.3, “Consistent Nonlocking Reads”
Section 1.8.3.3, “Constraints on Invalid Data”
Section 13.1.13, “CREATE EVENT Syntax”
Section 13.2.6, “INSERT Syntax”
Section 1.8.2.1, “SELECT INTO TABLE Differences”
Section 16.3, “The MEMORY Storage Engine”
Section 28.7.29.2, “What Results You Can Get from a Query”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

INSERT INTO ... SELECT ...

Section 28.7.7.36, “mysql_info()”

INSERT INTO ... SELECT FROM memory_table

Section 17.5.1.21, “Replication and MEMORY Tables”

INSERT INTO...SELECT

Section 8.2.1, “Optimizing SELECT Statements”

INSTALL COMPONENT

Section 13.1.1, “Atomic Data Definition Statement Support”
Section 5.4.2.1, “Error Log Component Configuration”
Section 5.5.3, “Error Log Components”
Section 13.7.4.3, “INSTALL COMPONENT Syntax”
Section 5.5.1, “Installing and Uninstalling Components”
Section 5.5.2, “Obtaining Server Component Information”

Section 6.4.3.1, “Password Validation Component Installation and Uninstallation”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”
Section 6.4.6, “The Audit Message Component”
Section 13.7.4.5, “UNINSTALL COMPONENT Syntax”

INSTALL PLUGIN

Section 13.1.1, “Atomic Data Definition Statement Support”
Section 6.4.5.6, “Audit Log Filtering”
Section 6.4.5.9, “Audit Log Reference”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 5.6.5.2, “ddl_rewriter Plugin Options”
Section 13.7.8.3, “FLUSH Syntax”
Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 15.19.2, “InnoDB memcached Architecture”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 13.7.4.4, “INSTALL PLUGIN Syntax”
Section 5.5.1, “Installing and Uninstalling Components”
Section 5.6.1, “Installing and Uninstalling Plugins”
Section 5.6.5.1, “Installing or Uninstalling ddl_rewriter”
Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”
Section 6.4.8.2, “Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification”
Section 6.4.7.2, “Installing or Uninstalling MySQL Enterprise Firewall”
Section 5.6.6.2, “Installing or Uninstalling Version Tokens”
Section 5.6.7.1, “Installing the Clone Plugin”
Section 6.4.4.1, “Keyring Plugin Installation”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 12.9.9, “MeCab Full-Text Parser Plugin”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 6.4.1.8, “No-Login Pluggable Authentication”
Section 5.6.2, “Obtaining Server Plugin Information”
Section 6.4.1.5, “PAM Pluggable Authentication”
Section 6.4.3.2, “Password Validation Options and Variables”
Section 16.11.1, “Pluggable Storage Engine Architecture”
Section 29.2.3, “Plugin API Components”
Section 29.2.4.2, “Plugin Data Structures”
Section 17.4.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.7, “Server Command Options”
Server Plugin Library and Plugin Descriptors
Section 5.1.8, “Server System Variables”
Section 15.19.3, “Setting Up the InnoDB memcached Plugin”
Section 13.7.7.25, “SHOW PLUGINS Syntax”
Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 6.4.1.10, “Test Pluggable Authentication”
Section 25.18, “The INFORMATION_SCHEMA PLUGINS Table”
Section 29.2, “The MySQL Plugin API”
Section 15.19.9, “Troubleshooting the InnoDB memcached Plugin”
Section 13.7.4.6, “UNINSTALL PLUGIN Syntax”
Section 6.4.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”

Section 1.4, “What Is New in MySQL 8.0”
Section 6.4.1.6, “Windows Pluggable Authentication”
Section 29.2.4.8, “Writing Audit Plugins”
Section 29.2.4.5, “Writing Daemon Plugins”
Section 29.2.4.4, “Writing Full-Text Parser Plugins”
Section 29.2.4.6, “Writing INFORMATION_SCHEMA Plugins”
Section 29.2.4.10, “Writing Password-Validation Plugins”
Writing the Server-Side Authentication Plugin

ITERATE

Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.3, “ITERATE Syntax”
Section 13.6.2, “Statement Label Syntax”

K

[index top]

KILL

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 8.14.2, “General Thread States”
Section 13.7.1.6, “GRANT Syntax”
Section 17.1.3.2, “GTID Life Cycle”
Section 13.7.8.4, “KILL Syntax”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section B.4.2.8, “MySQL server has gone away”
Section 28.7.7.39, “mysql_kill()”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
Section 13.4.2.7, “STOP SLAVE Syntax”
Section 27.4.3.9, “The innodb_lock_waits and x$innodb_lock_waits Views”
Section 27.4.3.28, “The schema_table_lock_waits and x$schema_table_lock_waits Views”

KILL CONNECTION

Section 13.7.8.4, “KILL Syntax”
Section 13.4.2.7, “STOP SLAVE Syntax”
Section 5.1.17, “The Server Shutdown Process”

KILL QUERY

Section 13.7.8.4, “KILL Syntax”
Section 12.24, “Miscellaneous Functions”
Section 13.4.2.7, “STOP SLAVE Syntax”
Section 5.1.17, “The Server Shutdown Process”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”

KILL QUERY processlist_id

Section 5.6.7.10, “Stopping a Cloning Operation”

L

[index top]

LEAVE

Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.4, “LEAVE Syntax”
Section 13.6.5.5, “LOOP Syntax”
Section 24.8, “Restrictions on Stored Programs”
Section 13.6.5.7, “RETURN Syntax”
Section 13.6.2, “Statement Label Syntax”

LOAD DATA

Section 6.4.5.4, “Audit Log File Formats”
Section 6.4.5.10, “Audit Log Restrictions”
Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 17.4.1.2, “Backing Up Raw Data from a Slave”
Section 7.1, “Backup and Recovery Types”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 28.7.12, “C API Asynchronous Interface”
Section 13.1.20.7, “CHECK Constraints”
Section 8.11.3, “Concurrent Inserts”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 7.2, “Database Backup Methods”
Section 10.3.3, “Database Character Set and Collation”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section B.4.3.4, “How MySQL Handles a Full Disk”
Section 13.2.5, “IMPORT TABLE Syntax”
Section 12.15, “Information Functions”
Section B.4.7, “Known Issues in MySQL”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 13.2.7, “LOAD DATA Syntax”
Section 13.2.8, “LOAD XML Syntax”
Section 3.3.3, “Loading Data into a Table”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 16.2.1, “MyISAM Startup Options”
Section 4.5.1.1, “mysql Client Options”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 28.7.7.36, “mysql_info()”
Section 28.7.7.50, “mysql_options()”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 22.4.27, “`ndb_show_tables` — Display List of NDB Tables”
Section 9.1.7, “NULL Values”
Section 8.2.5.1, “Optimizing INSERT Statements”
Section 11.2.6, “Out-of-Range and Overflow Handling”
Section 4.1, “Overview of MySQL Programs”
Section 23.1, “Overview of Partitioning in MySQL”
Section 23.5, “Partition Selection”
Section 22.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”

Section B.4.4.3, “Problems with NULL Values”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 17.5.1.19, “Replication and LOAD DATA”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 8.14.5, “Replication Slave SQL Thread States”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 10.11, “Restrictions on Character Sets”
Section 24.8, “Restrictions on Stored Programs”
Section 17.3.2.1, “Scope of Binary Log Encryption”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 13.2.10.1, “SELECT ... INTO Syntax”
Section 3.3.4.1, “Selecting All Data”
Section 5.1.7, “Server Command Options”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 13.2.11, “Subquery Syntax”
Section 11.4.4, “The ENUM Type”
Section 16.3, “The MEMORY Storage Engine”
Section 13.2.11.1, “The Subquery as Scalar Operand”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 9.4, “User-Defined Variables”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”
Section 24.3, “Using Triggers”
Section 1.4, “What Is New in MySQL 8.0”
Section 28.7.29.2, “What Results You Can Get from a Query”
Section B.4.3.5, “Where MySQL Stores Temporary Files”
Section 2.3.7, “Windows Platform Restrictions”

LOAD DATA ... IGNORE

Section 13.1.20.7, “CHECK Constraints”

LOAD DATA ... REPLACE

Section 13.2.9, “REPLACE Syntax”

LOAD DATA INFILE

Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”

LOAD DATA LOCAL

Section 28.7.6, “C API Function Overview”
Section 13.2.7, “LOAD DATA Syntax”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 28.7.7.50, “mysql_options()”
Section 28.7.7.54, “mysql_real_connect()”
Section 28.7.7.72, “mysql_set_local_infile_default()”
Section 28.7.7.73, “mysql_set_local_infile_handler()”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 5.1.8, “Server System Variables”

LOAD INDEX INTO CACHE

Section 13.7.8.2, “CACHE INDEX Syntax”

Section 8.10.2.4, “Index Preloading”
Section 13.7.8.5, “LOAD INDEX INTO CACHE Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”

LOAD INDEX INTO CACHE ... IGNORE LEAVES

Section 13.7.8.5, “LOAD INDEX INTO CACHE Syntax”

LOAD XML

Section 28.7.12, “C API Asynchronous Interface”
Section 13.1.20.7, “CHECK Constraints”
Section 13.2.8, “LOAD XML Syntax”
Section 23.1, “Overview of Partitioning in MySQL”
Section 23.5, “Partition Selection”
Section 5.1.11, “Server SQL Modes”

LOAD XML ... IGNORE

Section 13.1.20.7, “CHECK Constraints”

LOAD XML LOCAL

Section 13.2.8, “LOAD XML Syntax”

LOCK INSTANCE FOR BACKUP

Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

LOCK TABLE

Section 8.11.3, “Concurrent Inserts”
Section 8.14.2, “General Thread States”
Section B.4.6.1, “Problems with ALTER TABLE”

LOCK TABLES

Section 13.1.10, “ALTER TABLESPACE Syntax”
Section 15.1.2, “Best Practices for InnoDB Tables”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 13.1.12, “CREATE DATABASE Syntax”
Section 13.1.20.4, “CREATE TABLE ... LIKE Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 15.7.5.2, “Deadlock Detection and Rollback”
Section 15.7.5, “Deadlocks in InnoDB”
Section 13.7.8.3, “FLUSH Syntax”
Section 15.6.3.3, “General Tablespaces”
Section 8.14.2, “General Thread States”
Section 13.7.1.6, “GRANT Syntax”
Section 15.7.5.3, “How to Minimize and Handle Deadlocks”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.11.1, “Internal Locking Methods”
Section 22.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”
Section 15.6.1.6, “Limits on InnoDB Tables”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 16.7.2, “MERGE Table Problems”
Section 8.11.4, “Metadata Locking”

Section 28.7.7.69, “mysql_session_track_get_first()”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 16.2.4.2, “Problems from Tables Not Being Closed Properly”
Section 13.1.36, “RENAME TABLE Syntax”
Section 24.8, “Restrictions on Stored Programs”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 8.11.2, “Table Locking Issues”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

LOCK TABLES ... READ

Section 13.7.8.3, “FLUSH Syntax”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”
Section 8.11.4, “Metadata Locking”

LOCK TABLES ... WRITE

Section 15.7.1, “InnoDB Locking”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”

LOCK TABLES READ

Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

LOCK TABLES WRITE

Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

LOOP

Section 13.6.5, “Flow Control Statements”
Section 13.6.5.3, “ITERATE Syntax”
Section 13.6.5.4, “LEAVE Syntax”
Section 13.6.5.5, “LOOP Syntax”
Section 13.6.2, “Statement Label Syntax”

O

[index top]

OPTIMIZE PARTITION

Section 15.12.1, “Online DDL Operations”

OPTIMIZE TABLE

Section 22.5.15.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 22.5.15.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 29.5.1, “Debugging a MySQL Server”
Section 13.2.2, “DELETE Syntax”

Section 16.2.3.2, “Dynamic Table Characteristics”
Section 15.6.3.2, “File-Per-Table Tablespaces”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 8.14.2, “General Thread States”
Section B.4.3.4, “How MySQL Handles a Full Disk”
Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 15.9.2, “InnoDB Page Compression”
Section 15.10, “InnoDB Row Formats”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 13.7.8.4, “KILL Syntax”
Section 22.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”
Section 13.3.5, “LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Syntax”
Section 23.3.4, “Maintenance of Partitions”
Section 16.7.2, “MERGE Table Problems”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.4.1, “myisamchk General Options”
Section 1.8.1, “MySQL Extensions to Standard SQL”
MySQL Server Options for NDB Cluster
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 22.5.13.3, “NDB Cluster Disk Data Storage Requirements”
Section 15.12.6, “Online DDL Limitations”
Section 22.5.14, “Online Operations with ALTER TABLE in NDB Cluster”
Section 13.7.3.4, “OPTIMIZE TABLE Syntax”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.2.5.2, “Optimizing UPDATE Statements”
Section 8.2.7, “Other Optimization Tips”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.13, “Replication and FLUSH”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 5.1.8, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 16.2.3.1, “Static (Fixed-Length) Table Characteristics”
Section 16.5, “The ARCHIVE Storage Engine”
Section 25.39.13, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 25.39.14, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”
Section 25.39.16, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 25.39.17, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”
Section 25.39.18, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”
Section 5.1.17, “The Server Shutdown Process”
Section 5.4.5, “The Slow Query Log”
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”

ORDER BY

Section 15.1.1, “Benefits of Using InnoDB Tables”

P

[index top]

PARTITION BY

Section 15.12.1, “Online DDL Operations”

PREPARE

Section 28.7.25, “C API Prepared CALL Statement Support”
Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 13.2.1, “CALL Syntax”
Section 13.5.3, “DEALLOCATE PREPARE Syntax”
Section 13.5.2, “EXECUTE Syntax”
Section 9.2.3, “Identifier Case Sensitivity”
Section 8.11.4, “Metadata Locking”
Section 13.5.1, “PREPARE Syntax”
Section 13.5, “Prepared SQL Statement Syntax”
Section 24.8, “Restrictions on Stored Programs”
Section 5.1.10, “Server Status Variables”
Section 26.12.6.4, “The prepared_statements_instances Table”

PURGE BINARY LOGS

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 7.3.1, “Establishing a Backup Policy”
Section 13.7.1.6, “GRANT Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 5.4.7, “Server Log Maintenance”
Section 5.4.4, “The Binary Log”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”

PURGE BINARY LOGS TO

Section 13.4.1.2, “RESET MASTER Syntax”

R

[index top]

REBUILD PARTITION

Section 15.12.1, “Online DDL Operations”

RELEASE SAVEPOINT

Section 26.12.7, “Performance Schema Transaction Tables”
Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax”
Section 26.12.7.1, “The events_transactions_current Table”

REMOVE PARTITIONING

Section 15.12.1, “Online DDL Operations”

RENAME TABLE

Section 13.1.9, “ALTER TABLE Syntax”
Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 13.2.2, “DELETE Syntax”
Section 8.14.2, “General Thread States”
Section 8.11.4, “Metadata Locking”
Section 15.6.1.2, “Moving or Copying InnoDB Tables”

Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 15.12.1, “Online DDL Operations”
Section 2.11.5, “Preparing Your Installation for Upgrade”
Section 13.1.36, “RENAME TABLE Syntax”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”

RENAME USER

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 13.7.1.6, “GRANT Syntax”
Section 12.15, “Information Functions”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.7, “RENAME USER Syntax”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”
Section 6.2.10, “Using Roles”
Section 6.2.13, “When Privilege Changes Take Effect”

REORGANIZE PARTITION

Section 15.12.1, “Online DDL Operations”

REPAIR PARTITION

Section 15.12.1, “Online DDL Operations”

REPAIR TABLE

Section 13.1.9.1, “ALTER TABLE Partition Operations”
Section 13.1.9, “ALTER TABLE Syntax”
Section 13.7.3.2, “CHECK TABLE Syntax”
Section 16.2.4.1, “Corrupted MyISAM Tables”
Section 7.2, “Database Backup Methods”
Section 23.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 8.11.5, “External Locking”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 8.14.2, “General Thread States”
Section B.4.3.4, “How MySQL Handles a Full Disk”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.7, “How to Report Bugs or Problems”
Section 13.2.5, “IMPORT TABLE Syntax”
Section 13.7.8.4, “KILL Syntax”
Section 13.2.7, “LOAD DATA Syntax”
Section 13.3.5, “LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Syntax”
Section 23.3.4, “Maintenance of Partitions”
Section 16.7.2, “MERGE Table Problems”
Section 16.2.1, “MyISAM Startup Options”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.6.4.1, “myisamchk General Options”
Section 4.6.4, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 1.8.1, “MySQL Extensions to Standard SQL”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 6.2.2, “Privileges Provided by MySQL”
Section 16.2.4.2, “Problems from Tables Not Being Closed Properly”
Section B.4.6.1, “Problems with ALTER TABLE”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”
Section 13.7.3.5, “REPAIR TABLE Syntax”
Section 16.4.1, “Repairing and Checking CSV Tables”
Section 17.5.1.13, “Replication and FLUSH”
Section 17.5.1.25, “Replication and REPAIR TABLE”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 5.1.8, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 16.5, “The ARCHIVE Storage Engine”
Section 5.1.17, “The Server Shutdown Process”
Section 5.4.5, “The Slow Query Log”
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”

REPEAT

Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 24.1, “Defining Stored Programs”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.3, “ITERATE Syntax”
Section 13.6.5.4, “LEAVE Syntax”
Section 13.6.5.6, “REPEAT Syntax”
Section 13.6.2, “Statement Label Syntax”

REPLACE

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 13.1.20.7, “CHECK Constraints”
Section 13.1.20.9, “CREATE TABLE and Generated Columns”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 11.7, “Data Type Default Values”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Syntax”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 12.15, “Information Functions”
Section 13.2.6, “INSERT Syntax”
Section B.4.7, “Known Issues in MySQL”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 16.7.2, “MERGE Table Problems”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section B.4.2.8, “MySQL server has gone away”
Section 28.7.7.1, “mysql_affected_rows()”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”
Section 8.9.3, “Optimizer Hints”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 23.1, “Overview of Partitioning in MySQL”

Section 23.5, “Partition Selection”
Section 13.2.9, “REPLACE Syntax”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 13.2.11, “Subquery Syntax”
Section 16.5, “The ARCHIVE Storage Engine”
Section 1.3.2, “The Main Features of MySQL”
Section 22.5.10.29, “The ndbinfo operations_per_fragment Table”
Section 5.6.4, “The Rewriter Query Rewrite Plugin”
Section 13.2.12, “UPDATE Syntax”
Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

REPLACE ... SELECT

Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section B.4.7, “Known Issues in MySQL”

RESET

Section 13.7.8.3, “FLUSH Syntax”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 13.7.8.7, “RESET PERSIST Syntax”
Section 13.7.8.6, “RESET Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”

RESET MASTER

Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Format and Storage”
Section 17.1.3.2, “GTID Life Cycle”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 28.7.7.58, “mysql_refresh()”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 17.4.8, “Switching Masters During Failover”
Section 5.4.4, “The Binary Log”
Section 26.12.11.5, “The replication_applier_status_by_coordinator Table”
Section 26.12.11.6, “The replication_applier_status_by_worker Table”
Section 26.12.11.2, “The replication_connection_status Table”
Section 22.1.4, “What is New in NDB Cluster”

RESET PERSIST

Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 5.1.9.3, “Persisted System Variables”
Section 13.7.8.7, “RESET PERSIST Syntax”
Section 13.7.8.6, “RESET Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.6.1, “SET Syntax for Variable Assignment”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 5.1.9.1, “System Variable Privileges”

RESET SLAVE

Section 13.4.2.1, “CHANGE MASTER TO Syntax”

Section 17.2.3.1, “Commands for Operations on a Single Channel”
Section 17.2.3.2, “Compatibility with Previous Replication Statements”
Section 17.4.10, “Delayed Replication”
Section 17.1.3.2, “GTID Life Cycle”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 28.7.7.58, “mysql_refresh()”
NDB Cluster System Variables
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.4.2.6, “START SLAVE Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 26.12.11.5, “The replication_applier_status_by_coordinator Table”
Section 26.12.11.6, “The replication_applier_status_by_worker Table”
Section 26.12.11.2, “The replication_connection_status Table”

RESET SLAVE ALL

Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 17.2.5.4, “Replication Channel Based Filters”
Section 13.4.2.4, “RESET SLAVE Syntax”

RESIGNAL

Section 13.6.7, “Condition Handling”
Section 13.6.7.8, “Condition Handling and OUT or INOUT Parameters”
Section 13.6.7.1, “DECLARE ... CONDITION Syntax”
Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 13.6.7.3, “GET DIAGNOSTICS Syntax”
Section 13.6.7.4, “RESIGNAL Syntax”
Section 13.6.8, “Restrictions on Condition Handling”
Section 24.8, “Restrictions on Stored Programs”
Section 13.6.7.6, “Scope Rules for Handlers”
Section 13.6.7.5, “SIGNAL Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”

RESTART

Section 5.6.7.3, “Cloning Remote Data”
Section 2.3, “Installing MySQL on Microsoft Windows”
Section 5.1.9.3, “Persisted System Variables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 21.2.4, “Production Deployment of InnoDB Cluster”
Section 13.7.8.8, “RESTART Syntax”
Section 5.1.7, “Server Command Options”
Section 22.1.4, “What is New in NDB Cluster”

RETURN

Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.5, “LOOP Syntax”

Section 24.8, “Restrictions on Stored Programs”
Section 13.6.5.7, “RETURN Syntax”
Section 13.6.7.5, “SIGNAL Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”

REVOKE

Section 6.2, “Access Control and Account Management”
Section 6.2.1, “Account User Names and Passwords”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”
Section 13.7.8.3, “FLUSH Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 12.15, “Information Functions”
Section 5.1.12, “IPv6 Support”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section A.14, “MySQL 8.0 FAQ: Replication”
Section 1.8.2, “MySQL Differences from Standard SQL”
MySQL Glossary
Section 22.5.12.2, “NDB Cluster and MySQL Privileges”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 6.2.2, “Privileges Provided by MySQL”
Section 6.2.18, “Proxy Users”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Section 17.5.1.22, “Replication of the mysql System Schema”
Section 13.7.1.8, “REVOKE Syntax”
Section 6.1.1, “Security Guidelines”
Section 5.1.8, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 5.1.9.1, “System Variable Privileges”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”
Section 6.2.10, “Using Roles”
Section 6.2.13, “When Privilege Changes Take Effect”

REVOKE ALL PRIVILEGES

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 6.2.10, “Using Roles”

ROLLBACK

Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 15.7.2.2, “autocommit, Commit, and Rollback”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 15.7.5.2, “Deadlock Detection and Rollback”
Section 4.6.1, “`ibd2sdi` — InnoDB Tablespace SDI Extraction Utility”
Section 12.15, “Information Functions”
Section 15.2, “InnoDB and the ACID Model”
Section 15.20.4, “InnoDB Error Handling”
Section 15.13, “InnoDB Startup Options and System Variables”

Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 28.7.7.3, “mysql_change_user()”
Section 26.12.7, “Performance Schema Transaction Tables”
Section 17.5.1.34, “Replication and Transactions”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section B.4.4.5, “Rollback Failure for Nontransactional Tables”
Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax”
Section 5.1.8, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 13.3.2, “Statements That Cannot Be Rolled Back”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 24.7, “Stored Program Binary Logging”
Section 5.4.4, “The Binary Log”
Section 26.12.7.1, “The events_transactions_current Table”
Section 13.3, “Transactional and Locking Statements”
Section 24.3.1, “Trigger Syntax and Examples”
Section 1.4, “What Is New in MySQL 8.0”

ROLLBACK TO SAVEPOINT

Section 26.12.7, “Performance Schema Transaction Tables”
Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax”
Section 26.12.7.1, “The events_transactions_current Table”

ROLLBACK to SAVEPOINT

Section 24.3.1, “Trigger Syntax and Examples”

S

[index top]

SAVEPOINT

Section 26.12.7, “Performance Schema Transaction Tables”
Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Syntax”
Section 26.12.7.1, “The events_transactions_current Table”

SE PERSIST_ONLY

Section 4.2.2.2, “Using Option Files”

SELECT

Section 6.2, “Access Control and Account Management”
Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 13.1.9, “ALTER TABLE Syntax”
Section 13.1.11, “ALTER VIEW Syntax”
Section 12.3.4, “Assignment Operators”
Section 6.4.5.4, “Audit Log File Formats”
Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 15.7.2.2, “autocommit, Commit, and Rollback”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 28.7.5, “C API Data Structures”
Section 28.7.6, “C API Function Overview”
Section 28.7.23, “C API Multiple Statement Execution Support”
Section 28.7.25, “C API Prepared CALL Statement Support”

Section 28.7.10, “C API Prepared Statement Function Overview”
Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 12.3.2, “Comparison Functions and Operators”
Section 8.3.9, “Comparison of B-Tree and Hash Indexes”
Section 8.11.3, “Concurrent Inserts”
Section 18.4.1, “Configuring an Online Group”
Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”
Section 10.4, “Connection Character Sets and Collations”
Section 15.7.2.3, “Consistent Nonlocking Reads”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 13.1.13, “CREATE EVENT Syntax”
Section 13.1.15, “CREATE INDEX Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.20.5, “CREATE TABLE ... SELECT Syntax”
Section 13.1.20.9, “CREATE TABLE and Generated Columns”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”
Section 13.1.23, “CREATE VIEW Syntax”
Section 16.8.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 3.3.1, “Creating and Selecting a Database”
Section 13.6.6.2, “Cursor DECLARE Syntax”
Section 13.6.6.3, “Cursor FETCH Syntax”
Section 15.7.5.2, “Deadlock Detection and Rollback”
Section 13.2.2, “DELETE Syntax”
Section 13.2.11.8, “Derived Tables”
Section 8.4.3.2, “Disadvantages of Creating Many Tables in the Same Database”
Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section 13.2.3, “DO Syntax”
Section 3.2, “Entering Queries”
Section 24.4.2, “Event Scheduler Configuration”
Section 10.8.6, “Examples of the Effect of Collation”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Syntax”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 13.7.8.3, “FLUSH Syntax”
Section 15.20.2, “Forcing InnoDB Recovery”
Section 8.2.1.20, “Function Call Optimization”
Chapter 12, *Functions and Operators*
Section 12.17.3, “Functions That Search JSON Values”
Section 8.14.2, “General Thread States”
Section 13.7.1.6, “GRANT Syntax”
Section 13.2.4, “HANDLER Syntax”
Section 23.2.7, “How MySQL Partitioning Handles NULL”
Section 15.7.5.3, “How to Minimize and Handle Deadlocks”
Section 1.7, “How to Report Bugs or Problems”
Section 8.9.4, “Index Hints”
Section 12.15, “Information Functions”
Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 13.2.6.1, “INSERT ... SELECT Syntax”
Section 13.2.6, “INSERT Syntax”
Section 8.11.1, “Internal Locking Methods”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 25.1, “Introduction”

Section 13.2.10.2, “JOIN Syntax”
Section 9.3, “Keywords and Reserved Words”
Section 13.7.8.4, “KILL Syntax”
Section B.4.7, “Known Issues in MySQL”
Section 13.2.11.9, “Lateral Derived Tables”
Section 6.4.5.8, “Legacy Mode Audit Log Filtering”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 13.2.8, “LOAD XML Syntax”
Section 13.6.4.2, “Local Variable Scope and Resolution”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 15.7.2.4, “Locking Reads”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 16.7.2, “MERGE Table Problems”
Section 8.3.6, “Multiple-Column Indexes”
Section 7.6.4, “MyISAM Table Optimization”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section A.14, “MySQL 8.0 FAQ: Replication”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”
Section 1.8.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Chapter 26, *MySQL Performance Schema*
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.17, “mysql_fetch_field()”
Section 28.7.7.22, “mysql_field_count()”
Section 28.7.7.48, “mysql_num_fields()”
Section 28.7.7.49, “mysql_num_rows()”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.11, “mysql_stmt_fetch()”
Section 28.7.11.18, “mysql_stmt_num_rows()”
Section 28.7.11.28, “mysql_stmt_store_result()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section 12.9.1, “Natural Language Full-Text Searches”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 22.2.6, “NDB Cluster Example with Tables and Data”
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
Section 22.6.4, “NDB Cluster Replication Schema and Tables”
NDB Cluster System Variables
Section 22.4.24, “`ndb_select_all` — Print Rows from an NDB Table”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”
Section 23.3.5, “Obtaining Information About Partitions”
Section 15.12.2, “Online DDL Performance and Concurrency”
Section 8.3, “Optimization and Indexes”
Section 8.9.3, “Optimizer Hints”

Section B.4.5, “Optimizer-Related Issues”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.1, “Optimizing SELECT Statements”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 8.2.5.2, “Optimizing UPDATE Statements”
Section 4.6.4.4, “Other myisamchk Options”
Section 23.4, “Partition Pruning”
Section 23.5, “Partition Selection”
Section 26.6, “Performance Schema Instrument Naming Conventions”
Section 26.12.13.1, “Performance Schema persisted_variables Table”
Section 26.12.11, “Performance Schema Replication Tables”
Section 5.1.9.3, “Persisted System Variables”
Section 15.14.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”
Section 15.7.4, “Phantom Rows”
Section 6.2.2, “Privileges Provided by MySQL”
Section B.4.4.2, “Problems Using DATE Columns”
Section B.4.4.8, “Problems with Floating-Point Values”
Section 22.5.9, “Quick Reference: NDB Cluster SQL Statements”
Section 23.2.3.1, “RANGE COLUMNS partitioning”
Section 8.2.1.2, “Range Optimization”
Section 16.4.1, “Repairing and Checking CSV Tables”
Section 13.2.9, “REPLACE Syntax”
Section 17.2, “Replication Implementation”
Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.5.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”
Section 17.5.1.16, “Replication of Invoked Features”
Section 24.8, “Restrictions on Stored Programs”
Section 3.3.4, “Retrieving Information from a Table”
Section 3.6.7, “Searching on Two Keys”
Section 13.1.20.10, “Secondary Indexes and Generated Columns”
Section 13.2.10.1, “SELECT ... INTO Syntax”
Section 13.2.10, “SELECT Syntax”
Section 3.3.4.1, “Selecting All Data”
Section 3.3.4.2, “Selecting Particular Rows”
Section B.3.1, “Server Error Message Reference”
Section 5.1.11, “Server SQL Modes”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 13.7.6.1, “SET Syntax for Variable Assignment”
Section 13.7.7.2, “SHOW BINLOG EVENTS Syntax”
Section 13.7.7.9, “SHOW CREATE PROCEDURE Syntax”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 13.7.7.17, “SHOW ERRORS Syntax”
Section 13.7.7.27, “SHOW PROCEDURE CODE Syntax”
Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
Section 13.7.7.32, “SHOW RELAYLOG EVENTS Syntax”
Section 13.7.7, “SHOW Syntax”
Section 13.7.7.39, “SHOW VARIABLES Syntax”

Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section B.4.4.7, “Solving Problems with No Matching Rows”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 24.7, “Stored Program Binary Logging”
Section 24.2.1, “Stored Routine Syntax”
Section 9.1.1, “String Literals”
Section 13.2.11.6, “Subqueries with EXISTS or NOT EXISTS”
Section 13.2.11.10, “Subquery Errors”
Section 13.2.11, “Subquery Syntax”
Section 8.11.2, “Table Locking Issues”
Section 16.5, “The ARCHIVE Storage Engine”
Section 5.4.4, “The Binary Log”
Section 11.4.4, “The ENUM Type”
Section 26.12.18.1, “The host_cache Table”
Section 25.6, “The INFORMATION_SCHEMA COLUMNS Table”
Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”
Section 25.39.29, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 25.19, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”
Section 11.6, “The JSON Data Type”
Section 1.3.2, “The Main Features of MySQL”
Section 16.7, “The MERGE Storage Engine”
Section 5.3, “The mysql System Schema”
Section 22.5.10.28, “The ndbinfo nodes Table”
Section 5.6.4, “The Rewriter Query Rewrite Plugin”
Section 13.2.11.1, “The Subquery as Scalar Operand”
Section 26.12.18.5, “The threads Table”
Section 15.7.2.1, “Transaction Isolation Levels”
Section 24.3.1, “Trigger Syntax and Examples”
Section 12.2, “Type Conversion in Expression Evaluation”
Section 29.2.1, “Types of Plugins”
Section 1.2, “Typographical and Syntax Conventions”
Section 13.2.10.3, “UNION Syntax”
Section 13.2.12, “UPDATE Syntax”
Section 9.4, “User-Defined Variables”
Section 15.14.2.1, “Using InnoDB Transaction and Locking Information”
Section 21.3, “Using MySQL Router with InnoDB Cluster”
Section 4.2.2.1, “Using Options on the Command Line”
Section 29.5.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 11.5.11, “Using Spatial Indexes”
Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”
Section 10.2.2, “UTF-8 for Metadata”
Section 5.6.6.4, “Version Tokens Reference”
Section 24.5.1, “View Syntax”
Section 1.4, “What Is New in MySQL 8.0”
Section 8.2.1.1, “WHERE Clause Optimization”
Section B.4.3.5, “Where MySQL Stores Temporary Files”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

SELECT *

Section 11.4.3, “The BLOB and TEXT Types”

SELECT * FROM

Section 22.5.10.27, “The ndbinfo memory_per_fragment Table”

SELECT * FROM t PARTITION ()

Section 23.1, “Overview of Partitioning in MySQL”

SELECT * INTO OUTFILE 'file_name' FROM tbl_name

Section 7.2, “Database Backup Methods”

SELECT ... FOR SHARE

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 15.7.1, “InnoDB Locking”
Section 15.7.2.4, “Locking Reads”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 15.7.2.1, “Transaction Isolation Levels”

SELECT ... FOR UPDATE

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 15.1.2, “Best Practices for InnoDB Tables”
Section 15.7.5, “Deadlocks in InnoDB”
Section 15.7.5.3, “How to Minimize and Handle Deadlocks”
Section 15.7.1, “InnoDB Locking”
Section 15.7.2.4, “Locking Reads”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”

SELECT ... FROM

Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”

SELECT ... INTO

Section 13.1.13, “CREATE EVENT Syntax”
Section 13.6.4.2, “Local Variable Scope and Resolution”
Section 17.5.1.14, “Replication and System Functions”
Section 13.2.10.1, “SELECT ... INTO Syntax”
Section 1.8.2.1, “SELECT INTO TABLE Differences”
Section 13.2.10, “SELECT Syntax”

SELECT ... INTO DUMPFILE

Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 5.1.8, “Server System Variables”

SELECT ... INTO OUTFILE

Section 7.1, “Backup and Recovery Types”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 15.20.2, “Forcing InnoDB Recovery”
Section 13.2.7, “LOAD DATA Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 9.1.7, “NULL Values”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.2.10.1, “SELECT ... INTO Syntax”
Section 1.8.2.1, “SELECT INTO TABLE Differences”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 1.2, “Typographical and Syntax Conventions”
Section 1.4, “What Is New in MySQL 8.0”
Section 2.3.7, “Windows Platform Restrictions”

SELECT ... INTO OUTFILE 'file_name'

Section 13.2.10.1, “SELECT ... INTO Syntax”

SELECT ... INTO var_list

Section 24.8, “Restrictions on Stored Programs”
Section 13.6.4, “Variables in Stored Programs”

SELECT ... LOCK IN SHARE MODE

Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”

SELECT DISTINCT

Configuring the Number of Sampled Pages for InnoDB Optimizer Statistics
Section 8.14.2, “General Thread States”
Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”

SELECT INTO ... OUTFILE

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

SELECT INTO OUTFILE

Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”

SELECT SLEEP()

Section 5.1.11, “Server SQL Modes”

SET

Section 12.3.4, “Assignment Operators”
Section 6.4.5.9, “Audit Log Reference”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 28.7.21, “C API Binary Log Function Descriptions”
Section 5.6.7.12, “Clone System Variables”
Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”
Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”
Section 10.4, “Connection Character Sets and Collations”
Section 15.9.1.2, “Creating Compressed Tables”
Section 14.1, “Data Dictionary Schema”
Section 24.1, “Defining Stored Programs”
Section 24.4.2, “Event Scheduler Configuration”
Section 15.8.3.7, “Excluding Buffer Pool Pages from Core Files”
Section 12.1, “Function and Operator Reference”
Chapter 12, *Functions and Operators*
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 12.15, “Information Functions”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 4.5.1.6, “mysql Client Tips”
Section 12.19.2, “MySQL Enterprise Encryption Usage and Examples”

Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
Section 12.3, “Operators”
Section 8.9.3, “Optimizer Hints”
Section 5.1.9.3, “Persisted System Variables”
Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 5.1.7, “Server Command Options”
Section 6.2.16, “Server Handling of Expired Passwords”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 13.7.6, “SET Syntax”
Section 13.7.6.1, “SET Syntax for Variable Assignment”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 13.7.7.39, “SHOW VARIABLES Syntax”
Section 24.7, “Stored Program Binary Logging”
Section 13.2.11, “Subquery Syntax”
Section 5.1.9.1, “System Variable Privileges”
Section 13.6.7.7, “The MySQL Diagnostics Area”
Section 5.4.5, “The Slow Query Log”
Section 24.3.1, “Trigger Syntax and Examples”
Section 15.6.3.4, “Undo Tablespaces”
Section 9.4, “User-Defined Variables”
Section 4.2.2.1, “Using Options on the Command Line”
Section 4.2.2.5, “Using Options to Set Program Variables”
Section 5.1.9, “Using System Variables”
Section 13.6.4, “Variables in Stored Programs”
Section 1.4, “What Is New in MySQL 8.0”

SET @@GLOBAL.gtid_purged

Section 4.5.6, “`mysqlpump` — A Database Backup Program”

SET @@GLOBAL.ndb_slave_conflict_role = 'NONE'

NDB Cluster System Variables

SET autocommit

Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 13.3, “Transactional and Locking Statements”

SET autocommit = 0

Section 17.4.9, “Semisynchronous Replication”

SET CHARACTER SET

Section 10.4, “Connection Character Sets and Collations”
Section 28.7.7.55, “mysql_real_escape_string()”
Section 28.7.7.56, “mysql_real_escape_string_quote()”
Section 13.7.6.2, “SET CHARACTER SET Syntax”
Section 13.7.6, “SET Syntax”
Section 10.9, “Unicode Support”

SET CHARACTER SET 'charset_name'

Section 10.4, “Connection Character Sets and Collations”

SET CHARACTER SET charset_name

Section 10.4, “Connection Character Sets and Collations”

SET DEFAULT ROLE

Section 13.7.1.1, “ALTER USER Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.1.9, “SET DEFAULT ROLE Syntax”
Section 13.7.1.11, “SET ROLE Syntax”
Section 13.7.6, “SET Syntax”
Section 6.2.10, “Using Roles”

SET GLOBAL

Section 15.5.2, “Change Buffer”
Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
Section 15.8.7, “Configuring InnoDB I/O Capacity”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 15.8.8, “Configuring Spin Lock Polling”
Section 13.7.1.6, “GRANT Syntax”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.8.3.3, “Making the Buffer Pool Scan Resistant”
Section 8.10.2.2, “Multiple Key Caches”
Section 5.1.9.3, “Persisted System Variables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.4.2.5, “Rule-Based Error Log Filtering (log_filter_dragnet)”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Section 13.7.6.1, “SET Syntax for Variable Assignment”
Section 5.1.9.1, “System Variable Privileges”
Section 6.4.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”

SET GLOBAL sql_slave_skip_counter

Section 13.4.2.5, “SET GLOBAL sql_slave_skip_counter Syntax”

SET NAMES

Section 28.7.28, “C API Automatic Reconnection Control”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 10.5, “Configuring Application Character Set and Collation”
Section 10.4, “Connection Character Sets and Collations”
Section 10.6, “Error Message Character Set”
Section 13.2.7, “LOAD DATA Syntax”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 4.5.1.2, “mysql Client Commands”
Section 28.7.7.55, “mysql_real_escape_string()”
Section 28.7.7.56, “mysql_real_escape_string_quote()”
Section 28.7.7.60, “mysql_reset_connection()”
Section 28.7.7.71, “mysql_set_character_set()”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 5.1.8, “Server System Variables”
Section 13.7.6.3, “SET NAMES Syntax”
Section 13.7.6, “SET Syntax”
Section 10.10.7.2, “The gb18030 Character Set”
Section 12.2, “Type Conversion in Expression Evaluation”

Section 10.9, “Unicode Support”
Section 10.2.2, “UTF-8 for Metadata”

SET NAMES 'charset_name'

Section 10.4, “Connection Character Sets and Collations”

SET NAMES 'cp1251'

Section 10.4, “Connection Character Sets and Collations”

SET NAMES charset_name

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

SET NAMES default_character_set

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”

SET PASSWORD

Section 6.2.3, “Grant Tables”
Section 12.15, “Information Functions”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 6.2.15, “Password Management”
Section 6.4.3.2, “Password Validation Options and Variables”
Section 6.1.2.3, “Passwords and Logging”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.38, “Replication and Variables”
Section 17.5.1.8, “Replication of CURRENT_USER()”
Resetting the Root Password: Generic Instructions
Section 6.2.16, “Server Handling of Expired Passwords”
Section 5.1.8, “Server System Variables”
Section 13.7.1.10, “SET PASSWORD Syntax”
Section 13.7.6, “SET Syntax”
Section 6.2.4, “Specifying Account Names”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 6.4.3, “The Password Validation Component”
Section 6.2.13, “When Privilege Changes Take Effect”

SET PASSWORD ... = PASSWORD()

Section 1.4, “What Is New in MySQL 8.0”

SET PERSIST

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 4.2.6, “Connection Compression Control”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 5.4.2.1, “Error Log Component Configuration”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”
Section 6.2.15, “Password Management”
Section 26.12.13.1, “Performance Schema persisted_variables Table”
Section 5.1.9.3, “Persisted System Variables”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 6.2.2, “Privileges Provided by MySQL”
Section 21.2.4, “Production Deployment of InnoDB Cluster”

Section 5.4.2.5, “Rule-Based Error Log Filtering (log_filter_dragnet)”
Section 5.1.8, “Server System Variables”
Section 13.7.6.1, “SET Syntax for Variable Assignment”
Section 5.1.9.1, “System Variable Privileges”
Section 4.2.2.2, “Using Option Files”
Section 6.2.10, “Using Roles”

SET PERSIST_ONLY

Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”
Section 5.1.9.3, “Persisted System Variables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.8.8, “RESTART Syntax”
Section 5.1.8, “Server System Variables”
Section 5.1.9.1, “System Variable Privileges”
Section 4.2.2.2, “Using Option Files”

SET RESOURCE GROUP

Section 8.9.3, “Optimizer Hints”
Section 6.2.2, “Privileges Provided by MySQL”
Section 8.12.5, “Resource Groups”
Section 13.7.2.4, “SET RESOURCE GROUP Syntax”

SET ROLE

Section 12.15, “Information Functions”
Section 5.1.8, “Server System Variables”
Section 13.7.1.11, “SET ROLE Syntax”
Section 13.7.6, “SET Syntax”
Section 13.7.7.21, “SHOW GRANTS Syntax”
Section 6.2.10, “Using Roles”
Section 6.2.13, “When Privilege Changes Take Effect”

SET ROLE DEFAULT

Section 13.7.1.1, “ALTER USER Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 13.7.1.9, “SET DEFAULT ROLE Syntax”
Section 13.7.1.11, “SET ROLE Syntax”
Section 5.3, “The mysql System Schema”

SET SESSION

Section 5.1.9.1, “System Variable Privileges”

SET SESSION TRANSACTION ISOLATION LEVEL

Section 5.1.8, “Server System Variables”

SET SESSION TRANSACTION {READ WRITE | READ ONLY}

Section 5.1.8, “Server System Variables”

SET sql_log_bin = 0

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

SET sql_log_bin=OFF

Section 5.4.4, “The Binary Log”

SET sql_mode='modes'

Section A.3, “MySQL 8.0 FAQ: Server SQL Mode”

SET TIMESTAMP = value

Section 8.14, “Examining Thread Information”

SET TRANSACTION

Section 28.7.7.69, “mysql_session_track_get_first()”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 5.1.15, “Server Tracking of Client Session State Changes”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 15.7.2.1, “Transaction Isolation Levels”

SET TRANSACTION ISOLATION LEVEL

Section 5.1.8, “Server System Variables”
Section 13.7.6, “SET Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

Section 15.19.7, “The InnoDB memcached Plugin and Replication”

SET TRANSACTION {READ WRITE | READ ONLY}

Section 5.1.8, “Server System Variables”

SET var_name = value

Section 13.7.6, “SET Syntax”

SHOW

Section 28.7.5, “C API Data Structures”
Section 28.7.6, “C API Function Overview”
Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”
Section 13.1.13, “CREATE EVENT Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 3.3, “Creating and Using a Database”
Section 13.6.6.2, “Cursor DECLARE Syntax”
Section 14.1, “Data Dictionary Schema”
Section 25.42, “Extensions to SHOW Statements”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.1, “Introduction”
Section A.14, “MySQL 8.0 FAQ: Replication”
Section 1.8.1, “MySQL Extensions to Standard SQL”
Section 28.7.11.28, “mysql_stmt_store_result()”
Section 28.7.7.79, “mysql_store_result()”
Section 28.7.7.81, “mysql_use_result()”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 26.1, “Performance Schema Quick Start”
Section 24.8, “Restrictions on Stored Programs”
Section 13.7.7.5, “SHOW COLUMNS Syntax”
Section 13.7.7, “SHOW Syntax”

Section 13.7.7.37, “SHOW TABLES Syntax”
Section 13.4.1, “SQL Statements for Controlling Master Servers”
Section 5.4.4, “The Binary Log”
Section 1.3.2, “The Main Features of MySQL”
Section 27.2, “Using the sys Schema”
Section 10.2.2, “UTF-8 for Metadata”

SHOW BINARY LOGS

Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 13.7.7.1, “SHOW BINARY LOGS Syntax”
Section 13.4.1, “SQL Statements for Controlling Master Servers”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

SHOW BINLOG EVENTS

Section 22.6.4, “NDB Cluster Replication Schema and Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.6.6.5, “Restrictions on Server-Side Cursors”
Section 13.7.7.2, “SHOW BINLOG EVENTS Syntax”
Section 13.4.1, “SQL Statements for Controlling Master Servers”
Section 13.4.2.6, “START SLAVE Syntax”
Section 22.1.4, “What is New in NDB Cluster”

SHOW CHARACTER SET

Section 13.1.2, “ALTER DATABASE Syntax”
Section 10.3.8, “Character Set Introducers”
Section 10.2, “Character Sets and Collations in MySQL”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 10.3.5, “Column Character Set and Collation”
Section 10.3.3, “Database Character Set and Collation”
Section 25.42, “Extensions to SHOW Statements”
Section 5.1.8, “Server System Variables”
Section 13.7.7.3, “SHOW CHARACTER SET Syntax”
Section 10.10, “Supported Character Sets and Collations”
Section 10.3.4, “Table Character Set and Collation”
Section 25.2, “The INFORMATION_SCHEMA CHARACTER_SETS Table”

SHOW COLLATION

Section 13.1.2, “ALTER DATABASE Syntax”
Section 28.7.5, “C API Data Structures”
Section 10.15, “Character Set Configuration”
Section 10.2, “Character Sets and Collations in MySQL”
Section 10.14.2, “Choosing a Collation ID”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.8, “Server System Variables”
Section 13.7.7.4, “SHOW COLLATION Syntax”
Section 25.5, “The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table”
Section 25.4, “The INFORMATION_SCHEMA COLLATIONS Table”

SHOW COLUMNS

Section 13.8.2, “EXPLAIN Syntax”

Section 25.42, “Extensions to SHOW Statements”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 28.7.7.43, “mysql_list_fields()”
Section 26.1, “Performance Schema Quick Start”
Section 13.7.7.5, “SHOW COLUMNS Syntax”
Section 25.6, “The INFORMATION_SCHEMA COLUMNS Table”
Section 25.39.1, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table”
Section 25.39.2, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table”
Section 25.39.3, “The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table”
Section 25.39.4, “The INFORMATION_SCHEMA INNODB_CACHED_INDEXES Table”
Section 25.39.5, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”
Section 25.39.7, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”
Section 25.39.6, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”
Section 25.39.8, “The INFORMATION_SCHEMA INNODB_COLUMNS Table”
Section 25.39.9, “The INFORMATION_SCHEMA INNODB_DATAFILES Table”
Section 25.39.10, “The INFORMATION_SCHEMA INNODB_FIELDS Table”
Section 25.39.11, “The INFORMATION_SCHEMA INNODB_FOREIGN Table”
Section 25.39.12, “The INFORMATION_SCHEMA INNODB_FOREIGN_COLS Table”
Section 25.39.13, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 25.39.14, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”
Section 25.39.15, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”
Section 25.39.16, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 25.39.17, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”
Section 25.39.18, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”
Section 25.39.19, “The INFORMATION_SCHEMA INNODB_INDEXES Table”
Section 25.39.22, “The INFORMATION_SCHEMA INNODB_METRICS Table”
Section 25.39.23, “The INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES Table”
Section 25.39.24, “The INFORMATION_SCHEMA INNODB_TABLES Table”
Section 25.39.25, “The INFORMATION_SCHEMA INNODB_TABLESPACES Table”
Section 25.39.26, “The INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF Table”
Section 25.39.27, “The INFORMATION_SCHEMA INNODB_TABLESTATS View”
Section 25.39.28, “The INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO Table”
Section 25.39.29, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 25.39.30, “The INFORMATION_SCHEMA INNODB_VIRTUAL Table”

SHOW COLUMNS FROM tbl_name LIKE 'enum_col'

Section 11.4.4, “The ENUM Type”

SHOW COUNT()

Section 13.7.7.17, “SHOW ERRORS Syntax”
Section 13.7.7.40, “SHOW WARNINGS Syntax”

SHOW CREATE DATABASE

Section 5.1.8, “Server System Variables”
Section 13.7.7.6, “SHOW CREATE DATABASE Syntax”

SHOW CREATE EVENT

Section 24.4.4, “Event Metadata”
Section 13.7.7.18, “SHOW EVENTS Syntax”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”

SHOW CREATE FUNCTION

Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 1.7, “How to Report Bugs or Problems”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 13.7.7.9, “SHOW CREATE PROCEDURE Syntax”
Section 24.2.3, “Stored Routine Metadata”

SHOW CREATE PROCEDURE

Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 1.7, “How to Report Bugs or Problems”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 13.7.7.8, “SHOW CREATE FUNCTION Syntax”
Section 24.2.3, “Stored Routine Metadata”

SHOW CREATE SCHEMA

Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 13.7.7.6, “SHOW CREATE DATABASE Syntax”

SHOW CREATE TABLE

Section 13.1.9, “ALTER TABLE Syntax”
Section 15.8.11, “Configuring the Merge Threshold for Index Pages”
Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 13.1.20.1, “CREATE TABLE Statement Retention”
Section 13.1.20, “CREATE TABLE Syntax”
Section 15.9.1.2, “Creating Compressed Tables”
Section 14.1, “Data Dictionary Schema”
Section 11.7, “Data Type Default Values”
Section 13.8.2, “EXPLAIN Syntax”
Section 3.4, “Getting Information About Databases and Tables”
Section 16.8.2, “How to Create FEDERATED Tables”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 15.9.2, “InnoDB Page Compression”
Section 23.2.5, “KEY Partitioning”
Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
NDB Cluster System Variables
Section 22.4.9, “`ndb_desc` — Describe NDB Tables”
Section 22.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 23.3.5, “Obtaining Information About Partitions”
Section 8.2.4, “Optimizing Performance Schema Queries”
Section 26.1, “Performance Schema Quick Start”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”
Section 5.1.8, “Server System Variables”
Section 13.1.20.11, “Setting NDB_TABLE Options”
Section 13.7.7.5, “SHOW COLUMNS Syntax”
Section 13.7.7.10, “SHOW CREATE TABLE Syntax”
Section 13.1.20.8, “Silent Column Specification Changes”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”
Section 3.6.6, “Using Foreign Keys”

SHOW CREATE TRIGGER

Section 13.7.7.11, “SHOW CREATE TRIGGER Syntax”
Section 24.3.2, “Trigger Metadata”

SHOW CREATE USER

Section 6.2.19, “Account Locking”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 13.7.1.1, “ALTER USER Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 6.2.3, “Grant Tables”
Section 5.1.8, “Server System Variables”
Section 13.7.7.12, “SHOW CREATE USER Syntax”
Section 13.7.7.21, “SHOW GRANTS Syntax”

SHOW CREATE VIEW

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 24.9, “Restrictions on Views”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”
Section 24.5.5, “View Metadata”

SHOW DATABASES

Section 3.3, “Creating and Using a Database”
Section 25.42, “Extensions to SHOW Statements”
Section 3.4, “Getting Information About Databases and Tables”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 9.2.3, “Identifier Case Sensitivity”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 25.1, “Introduction”
Section 22.5.12.2, “NDB Cluster and MySQL Privileges”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 13.7.7.14, “SHOW DATABASES Syntax”
Section 25.24, “The INFORMATION_SCHEMA SCHEMATA Table”

SHOW ENGINE

Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.7.15, “SHOW ENGINE Syntax”

SHOW ENGINE INNODB MUTEX

Section 15.16.3, “InnoDB Standard Monitor and Lock Monitor Output”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 13.7.7.15, “SHOW ENGINE Syntax”

SHOW ENGINE INNODB STATUS

Section 15.5.3, “Adaptive Hash Index”
Section 15.5.1, “Buffer Pool”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 15.7.5, “Deadlocks in InnoDB”
Section 15.16.2, “Enabling InnoDB Monitors”
Section B.2, “Error Information Interfaces”

Section 15.7.5.3, “How to Minimize and Handle Deadlocks”
Section 15.14.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”
Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 15.14.3, “InnoDB INFORMATION_SCHEMA Schema Object Tables”
Section 15.7.1, “InnoDB Locking”
Section 15.16.3, “InnoDB Standard Monitor and Lock Monitor Output”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.2, “Moving or Copying InnoDB Tables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 15.8.9, “Purge Configuration”
Section 13.7.7.15, “SHOW ENGINE Syntax”
Section 25.39.3, “The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

SHOW ENGINE NDB STATUS

Section 22.2.3.3, “Initial Startup of NDB Cluster on Windows”
Section 22.5, “Management of NDB Cluster”
Section 22.6.4, “NDB Cluster Replication Schema and Tables”
Section 22.5.9, “Quick Reference: NDB Cluster SQL Statements”

SHOW ENGINE NDBCLUSTER STATUS

MySQL Server Options for NDB Cluster
Section 22.5.9, “Quick Reference: NDB Cluster SQL Statements”

SHOW ENGINE PERFORMANCE_SCHEMA STATUS

Section 26.10, “Performance Schema Statement Digests and Sampling”
Section 26.7, “Performance Schema Status Monitoring”
Section 13.7.7.15, “SHOW ENGINE Syntax”

SHOW ENGINES

Chapter 16, *Alternative Storage Engines*
Section 22.5.4, “MySQL Server Usage for NDB Cluster”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 26.2, “Performance Schema Build Configuration”
Section 26.1, “Performance Schema Quick Start”
Section 22.5.9, “Quick Reference: NDB Cluster SQL Statements”
Section 2.3.4.3, “Selecting a MySQL Server Type”
Section 5.1.8, “Server System Variables”
Section 13.7.7.16, “SHOW ENGINES Syntax”
Section 16.5, “The ARCHIVE Storage Engine”
Section 16.6, “The BLACKHOLE Storage Engine”
Section 25.9, “The INFORMATION_SCHEMA ENGINES Table”
Section 15.1.3, “Verifying that InnoDB is the Default Storage Engine”

SHOW ERRORS

Section B.2, “Error Information Interfaces”
Section 13.6.7.3, “GET DIAGNOSTICS Syntax”
Section 13.6.7.4, “RESIGNAL Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.7.17, “SHOW ERRORS Syntax”
Section 13.7.7.40, “SHOW WARNINGS Syntax”

Section 13.6.7.5, “SIGNAL Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”

SHOW EVENTS

Section 24.4.4, “Event Metadata”
Section 17.5.1.16, “Replication of Invoked Features”
Section 13.7.7.18, “SHOW EVENTS Syntax”
Section 24.4.6, “The Event Scheduler and MySQL Privileges”
Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”

SHOW FULL COLUMNS

Section 13.1.20, “CREATE TABLE Syntax”
Section 25.7, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”

SHOW FULL PROCESSLIST

Section 8.14, “Examining Thread Information”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 15.12.2, “Online DDL Performance and Concurrency”

SHOW FULL TABLES

Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”

SHOW FUNCTION CODE

Section 13.7.7.27, “SHOW PROCEDURE CODE Syntax”

SHOW FUNCTION STATUS

Section 13.7.7.28, “SHOW PROCEDURE STATUS Syntax”
Section 24.2.3, “Stored Routine Metadata”

SHOW GLOBAL STATUS

NDB Cluster Status Variables
Section 5.1.8, “Server System Variables”

SHOW GRANTS

Section 6.2, “Access Control and Account Management”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 22.5.16, “Distributed MySQL Privileges with NDB_STORED_USER”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.8, “REVOKE Syntax”
Section 6.1.1, “Security Guidelines”
Section 5.1.8, “Server System Variables”
Section 13.7.7.12, “SHOW CREATE USER Syntax”
Section 13.7.7.21, “SHOW GRANTS Syntax”
Section 13.7.7.26, “SHOW PRIVILEGES Syntax”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 6.2.10, “Using Roles”

SHOW GRANTS FOR CURRENT_USER

Section 13.7.7.21, “SHOW GRANTS Syntax”

SHOW GRANTS FOR user

Section 13.7.7.21, “SHOW GRANTS Syntax”

SHOW INDEX

Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 15.8.11, “Configuring the Merge Threshold for Index Pages”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Syntax”
Section 8.9.4, “Index Hints”
Section 8.3.8, “InnoDB and MyISAM Index Statistics Collection”
Section 8.3.12, “Invisible Indexes”
Section 15.6.1.6, “Limits on InnoDB Tables”
Section 22.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 8.9.3, “Optimizer Hints”
Section 8.2.4, “Optimizing Performance Schema Queries”
Section 4.6.4.4, “Other myisamchk Options”
Section 13.7.7.5, “SHOW COLUMNS Syntax”
Section 13.7.7.22, “SHOW INDEX Syntax”
Section 25.26, “The INFORMATION_SCHEMA STATISTICS Table”
Section 25.32, “The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table”

SHOW MASTER LOGS

Section 13.7.7.1, “SHOW BINARY LOGS Syntax”

SHOW MASTER STATUS

Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.5.5, “How to Report Replication Bugs or Problems”
Section 22.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 17.1.2.4, “Obtaining the Replication Master Binary Log Coordinates”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.4.1, “SQL Statements for Controlling Master Servers”
Section 15.19.7, “The InnoDB memcached Plugin and Replication”
Section 17.5.4, “Troubleshooting Replication”

SHOW OPEN TABLES

Section 13.7.7.24, “SHOW OPEN TABLES Syntax”

SHOW PLUGINS

Section 5.6.7.3, “Cloning Remote Data”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 13.7.4.4, “INSTALL PLUGIN Syntax”
Section 5.6.1, “Installing and Uninstalling Plugins”
Section 5.6.5.1, “Installing or Uninstalling ddl_rewriter”
Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”
Section 5.6.7.1, “Installing the Clone Plugin”
Section 6.4.4.1, “Keyring Plugin Installation”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 12.9.9, “MeCab Full-Text Parser Plugin”
Section A.2, “MySQL 8.0 FAQ: Storage Engines”

MySQL Server Options for NDB Cluster
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 6.4.1.8, “No-Login Pluggable Authentication”
Section 5.6.2, “Obtaining Server Plugin Information”
Section 6.4.1.5, “PAM Pluggable Authentication”
Section 29.2.2, “Plugin API Characteristics”
Section 29.2.3, “Plugin API Components”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Server Plugin Library and Plugin Descriptors
Section 13.7.7.25, “SHOW PLUGINS Syntax”
Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”
Section 6.4.1.10, “Test Pluggable Authentication”
Section 25.13, “The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table”
Section 25.18, “The INFORMATION_SCHEMA PLUGINS Table”
Section 5.6.3.2, “Thread Pool Installation”
Section 1.4, “What Is New in MySQL 8.0”
Section 6.4.1.6, “Windows Pluggable Authentication”
Section 29.2.4.8, “Writing Audit Plugins”
Section 29.2.4.5, “Writing Daemon Plugins”
Section 29.2.4.4, “Writing Full-Text Parser Plugins”
Section 29.2.4.6, “Writing INFORMATION_SCHEMA Plugins”
Section 29.2.4.12, “Writing Keyring Plugins”
Section 29.2.4.10, “Writing Password-Validation Plugins”
Writing the Server-Side Authentication Plugin

SHOW plugins

Section 20.5.1, “Checking X Plugin Installation”

SHOW PRIVILEGES

Section 13.7.7.26, “SHOW PRIVILEGES Syntax”

SHOW PROCEDURE CODE

Section 13.7.7.19, “SHOW FUNCTION CODE Syntax”

SHOW PROCEDURE STATUS

Section 13.7.7.20, “SHOW FUNCTION STATUS Syntax”
Section 24.2.3, “Stored Routine Metadata”

SHOW PROCESSLIST

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.1.7.1, “Checking Replication Status”
Section 17.4.10, “Delayed Replication”
Section 24.4.2, “Event Scheduler Configuration”
Section 8.14, “Examining Thread Information”
Section 8.14.2, “General Thread States”
Section 13.7.1.6, “GRANT Syntax”
Section 17.1.3.1, “GTID Format and Storage”
Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 12.15, “Information Functions”
Section 15.20.4, “InnoDB Error Handling”
Section 13.7.8.4, “KILL Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”

Section A.14, “MySQL 8.0 FAQ: Replication”
Section 22.5.4, “MySQL Server Usage for NDB Cluster”
Section 28.7.7.44, “mysql_list_processes()”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”
Section 12.22, “Performance Schema Functions”
Section 26.6, “Performance Schema Instrument Naming Conventions”
Section 26.12.5, “Performance Schema Stage Event Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.2.2, “Replication Implementation Details”
Section 13.7.7.29, “SHOW PROCESSLIST Syntax”
Section 13.7.7.30, “SHOW PROFILE Syntax”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.4.2.6, “START SLAVE Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 5.6.7.10, “Stopping a Cloning Operation”
Section 17.4.8, “Switching Masters During Failover”
Section 25.13, “The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table”
Section 25.19, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 22.5.10.33, “The ndbinfo server_locks Table”
Section 22.5.10.34, “The ndbinfo server_operations Table”
Section 22.5.10.35, “The ndbinfo server_transactions Table”
Section 27.4.3.22, “The processlist and x$processlist Views”
Section 27.4.5.13, “The ps_is_thread_instrumented() Function”
Section 27.4.4.7, “The ps_setup_disable_thread() Procedure”
Section 27.4.4.11, “The ps_setup_enable_thread() Procedure”
Section 27.4.5.15, “The ps_thread_id() Function”
Section 26.12.18.5, “The threads Table”
Section B.4.2.6, “Too many connections”
Section 17.5.4, “Troubleshooting Replication”
Section 1.4, “What Is New in MySQL 8.0”

SHOW PROFILE

Section 8.14, “Examining Thread Information”
Section 8.14.2, “General Thread States”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 26.19.1, “Query Profiling Using Performance Schema”
Section 5.1.8, “Server System Variables”
Section 13.7.7.30, “SHOW PROFILE Syntax”
Section 13.7.7.31, “SHOW PROFILES Syntax”
Section 25.20, “The INFORMATION_SCHEMA PROFILING Table”

SHOW PROFILES

Section 2.9.7, “MySQL Source-Configuration Options”
Section 26.19.1, “Query Profiling Using Performance Schema”
Section 5.1.8, “Server System Variables”
Section 13.7.7.30, “SHOW PROFILE Syntax”
Section 13.7.7.31, “SHOW PROFILES Syntax”
Section 25.20, “The INFORMATION_SCHEMA PROFILING Table”

SHOW RELAYLOG EVENTS

Section 17.2.3.1, “Commands for Operations on a Single Channel”
Section 17.2.3.2, “Compatibility with Previous Replication Statements”

Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.7.2, “SHOW BINLOG EVENTS Syntax”
Section 13.7.7.32, “SHOW RELAYLOG EVENTS Syntax”
Section 13.4.2, “SQL Statements for Controlling Slave Servers”

SHOW SCHEMAS

Section 13.7.7.14, “SHOW DATABASES Syntax”

SHOW SESSION STATUS

NDB Cluster Status Variables

SHOW SLAVE HOSTS

Section 17.1.7.1, “Checking Replication Status”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.4.1, “SQL Statements for Controlling Master Servers”

SHOW SLAVE STATUS

Section 17.1.2.8, “Adding Slaves to a Replication Environment”
Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.1.7.1, “Checking Replication Status”
Section 17.2.3.1, “Commands for Operations on a Single Channel”
Section 17.2.3.2, “Compatibility with Previous Replication Statements”
Section 17.4.10, “Delayed Replication”
Section B.2, “Error Information Interfaces”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.5.5, “How to Report Replication Bugs or Problems”
Section 22.6.8, “Implementing Failover with NDB Cluster Replication”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section A.14, “MySQL 8.0 FAQ: Replication”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 26.12.11, “Performance Schema Replication Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.2.2, “Replication Implementation Details”
Section 17.1.5.1, “Replication Mode Concepts”
Section 8.14.4, “Replication Slave I/O Thread States”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”
Section 13.7.7.23, “SHOW MASTER STATUS Syntax”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 17.5.1.28, “Slave Errors During Replication”
Section 17.2.4.2, “Slave Status Logs”
Section 13.4.2, “SQL Statements for Controlling Slave Servers”
Section 13.4.2.6, “START SLAVE Syntax”
Section 26.12.11.3, “The replication_applier_configuration Table”
Section 26.12.11.4, “The replication_applier_status Table”
Section 26.12.11.5, “The replication_applier_status_by_coordinator Table”

Section 26.12.11.6, “The replication_applier_status_by_worker Table”
Section 26.12.11.1, “The replication_connection_configuration Table”
Section 26.12.11.2, “The replication_connection_status Table”
Section 17.5.4, “Troubleshooting Replication”

SHOW STATUS

Section 17.1.7.1, “Checking Replication Status”
Section 22.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”
Section 22.5, “Management of NDB Cluster”
MySQL Server Options for NDB Cluster
Section 22.5.17, “NDB API Statistics Counters and Variables”
Section 22.6, “NDB Cluster Replication”
Section 26.12.11, “Performance Schema Replication Tables”
Section 26.15, “Performance Schema System Variables”
Section 29.2.2, “Plugin API Characteristics”
Section 22.5.9, “Quick Reference: NDB Cluster SQL Statements”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.2.2, “Replication Implementation Details”
Section 24.8, “Restrictions on Stored Programs”
Section 17.4.9.3, “Semisynchronous Replication Monitoring”
Server Plugin Library and Plugin Descriptors
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 13.7.7.35, “SHOW STATUS Syntax”
Section 8.3.10, “Use of Index Extensions”
Section 29.2.4.8, “Writing Audit Plugins”
Section 29.2.4.4, “Writing Full-Text Parser Plugins”
Section 29.2.4, “Writing Plugins”

SHOW STATUS LIKE 'perf%'

Section 26.7, “Performance Schema Status Monitoring”

SHOW TABLE STATUS

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 13.1.20.1, “CREATE TABLE Statement Retention”
Section 13.1.20, “CREATE TABLE Syntax”
Section 15.6.1.1, “Creating InnoDB Tables”
Section 13.8.2, “EXPLAIN Syntax”
Section 15.11.2, “File Space Management”
Section 15.10, “InnoDB Row Formats”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”
MySQL Glossary
Section 23.3.5, “Obtaining Information About Partitions”
Section 13.7.7.5, “SHOW COLUMNS Syntax”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 16.5, “The ARCHIVE Storage Engine”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”

SHOW TABLES

Section 3.3.2, “Creating a Table”

Section 14.1, “Data Dictionary Schema”
Section 25.42, “Extensions to SHOW Statements”
Section 9.2.3, “Identifier Case Sensitivity”
Section 15.14, “InnoDB INFORMATION_SCHEMA Tables”
Section 25.1, “Introduction”
MySQL Glossary
Section 22.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 13.7.7.37, “SHOW TABLES Syntax”
Section B.4.2.15, “Table 'tbl_name' doesn't exist”
Section B.4.6.2, “TEMPORARY Table Problems”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”
Section 5.3, “The mysql System Schema”
Section 5.6.3.2, “Thread Pool Installation”
Section 6.4.7.3, “Using MySQL Enterprise Firewall”
Section B.4.3.5, “Where MySQL Stores Temporary Files”

SHOW TABLES FROM some_ndb_database

Section 22.5.12.2, “NDB Cluster and MySQL Privileges”

SHOW TRIGGERS

Section A.5, “MySQL 8.0 FAQ: Triggers”
Section 2.11.5, “Preparing Your Installation for Upgrade”
Section 13.7.7.38, “SHOW TRIGGERS Syntax”
Section 25.34, “The INFORMATION_SCHEMA TRIGGERS Table”
Section 24.3.2, “Trigger Metadata”

SHOW VARIABLES

Section 5.6.7.3, “Cloning Remote Data”
Section 29.2.4.3, “Compiling and Installing Plugin Libraries”
Section 24.4.2, “Event Scheduler Configuration”
Section 6.4.4.11, “Keyring System Variables”
Section 6.4.5.8, “Legacy Mode Audit Log Filtering”
Section 17.1.4.3, “Multi-Source Replication Monitoring”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 26.3, “Performance Schema Startup Configuration”
Section 26.15, “Performance Schema System Variables”
Section 29.2.2, “Plugin API Characteristics”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 17.4.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.7, “Server Command Options”
Server Plugin Library and Plugin Descriptors
Section 5.1.8, “Server System Variables”
Section 13.7.6.1, “SET Syntax for Variable Assignment”
Section 13.7.7.39, “SHOW VARIABLES Syntax”
Section 22.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 5.1.9, “Using System Variables”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”
Section 29.2.4.8, “Writing Audit Plugins”
Section 29.2.4.12, “Writing Keyring Plugins”
Section 29.2.4.10, “Writing Password-Validation Plugins”

Section 29.2.4, “Writing Plugins”

SHOW WARNINGS

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.1.9, “ALTER TABLE Syntax”
Section 10.14.4.3, “Diagnostics During Index.xml Parsing”
Section 13.1.29, “DROP PROCEDURE and DROP FUNCTION Syntax”
Section 13.1.32, “DROP TABLE Syntax”
Section B.2, “Error Information Interfaces”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Syntax”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 9.2.5, “Function Name Parsing and Resolution”
Section 13.6.7.3, “GET DIAGNOSTICS Syntax”
Section 13.2.7, “LOAD DATA Syntax”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 4.5.1.6, “mysql Client Tips”
Section 8.9.3, “Optimizer Hints”
Section 8.3.11, “Optimizer Use of Generated Column Indexes”
Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.2.2, “Optimizing Subqueries with Materialization”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 1.8.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 12.25.4, “Rounding Behavior”
Section 13.1.20.10, “Secondary Indexes and Generated Columns”
Section B.3.1, “Server Error Message Reference”
Section 5.1.8, “Server System Variables”
Section 13.7.7.17, “SHOW ERRORS Syntax”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 13.6.7.5, “SIGNAL Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”
Section 29.2.1, “Types of Plugins”
Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”
Section 1.4, “What Is New in MySQL 8.0”

SHOW_SLAVE_STATUS

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”

SHUTDOWN

Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”
Section 28.7.7.75, “mysql_shutdown()”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.8.9, “SHUTDOWN Syntax”
Section 18.7.3.2, “Upgrading a Group Replication Member”

SIGNAL

Section 13.6.7, “Condition Handling”
Section 13.6.7.1, “DECLARE ... CONDITION Syntax”
Section 13.6.7.2, “DECLARE ... HANDLER Syntax”
Section 12.15, “Information Functions”

Section 13.6.7.4, “RESIGNAL Syntax”
Section 13.6.8, “Restrictions on Condition Handling”
Section 24.8, “Restrictions on Stored Programs”
Section 13.6.7.6, “Scope Rules for Handlers”
Section 13.6.7.5, “SIGNAL Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”

SQL_AFTER_MTS_GAPS

Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.4.2.6, “START SLAVE Syntax”

START GROUP REPLICATION

Section 6.2.2, “Privileges Provided by MySQL”

START GROUP_REPLICATION

Section 18.2.1.5, “Bootstrapping the Group”
Cloning Operations
Section 18.7.1, “Combining Different Member Versions in a Group”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 18.4.3, “Distributed Recovery”
Section 18.6.6.4, “Exit Action”
Section 18.10, “Frequently Asked Questions”
Section 18.5.1, “Group Replication IP Address Whitelisting”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”
Section 18.4.4, “Network Partitioning”
Section 18.7.3.2, “Upgrading a Group Replication Member”

START SLAVE

Section 17.1.2.8, “Adding Slaves to a Replication Environment”
Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.2.3.1, “Commands for Operations on a Single Channel”
Section 17.2.3.2, “Compatibility with Previous Replication Statements”
Section 17.4.10, “Delayed Replication”
Section 17.1.3.1, “GTID Format and Storage”
Section 22.6.8, “Implementing Failover with NDB Cluster Replication”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 22.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 6.1.2.3, “Passwords and Logging”
Section 17.1.7.2, “Pausing Replication on the Slave”
Section 26.12.11, “Performance Schema Replication Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.3.3.2, “Recovering From Failed Replication Privilege Checks”
Section 17.4.6, “Replicating Different Databases to Different Slaves”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.2.2, “Replication Implementation Details”
Section 17.3.3, “Replication Privilege Checks”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”

Section 17.5.1.28, “Slave Errors During Replication”
Section 13.4.2.6, “START SLAVE Syntax”
Section 22.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 13.4.2.7, “STOP SLAVE Syntax”
Section 17.4.8, “Switching Masters During Failover”
Section 26.12.11.6, “The replication_applier_status_by_worker Table”
Section 17.5.4, “Troubleshooting Replication”
Section 22.6.7, “Using Two Replication Channels for NDB Cluster Replication”

START SLAVE SQL_THREAD

Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”

START SLAVE UNTIL SQL_AFTER_MTS_GAPS

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.4.2.6, “START SLAVE Syntax”

START TRANSACTION

Section 15.7.2.2, “autocommit, Commit, and Rollback”
Section 13.6.1, “BEGIN ... END Compound-Statement Syntax”
Section 13.7.8.3, “FLUSH Syntax”
Section 15.7.5.3, “How to Minimize and Handle Deadlocks”
Section 15.20.4, “InnoDB Error Handling”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 15.7.2.4, “Locking Reads”
Section 28.7.7.69, “mysql_session_track_get_first()”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 26.12.7, “Performance Schema Transaction Tables”
Section 24.8, “Restrictions on Stored Programs”
Section 17.4.9, “Semisynchronous Replication”
Section 5.1.8, “Server System Variables”
Section 5.1.15, “Server Tracking of Client Session State Changes”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 26.12.7.1, “The events_transactions_current Table”
Section 13.3, “Transactional and Locking Statements”
Section 24.3.1, “Trigger Syntax and Examples”
Section 13.3.8.2, “XA Transaction States”

START TRANSACTION ... COMMIT

Section 13.1.1, “Atomic Data Definition Statement Support”
Data Definition Statements

START TRANSACTION READ ONLY

MySQL Glossary

Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”

START TRANSACTION WITH CONSISTENT SNAPSHOT

Section 15.7.2.3, “Consistent Nonlocking Reads”

STATS_PERSISTENT=0

Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”

STATS_PERSISTENT=1

Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”

STOP GROUP REPLICATION

Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.2.4, “RESET SLAVE Syntax”

STOP GROUP_REPLICATION

Section 18.6.6.3, “Auto-Rejoin”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”
Section 18.10, “Frequently Asked Questions”
Section 18.5.1, “Group Replication IP Address Whitelisting”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”
Section 13.4.3.2, “STOP GROUP_REPLICATION Syntax”
Section 18.6.6.2, “Unreachable Majority Timeout”
Section 18.7.3.2, “Upgrading a Group Replication Member”

STOP SLAVE

Section 17.1.2.8, “Adding Slaves to a Replication Environment”
Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.1.7.1, “Checking Replication Status”
Section 17.2.3.1, “Commands for Operations on a Single Channel”
Section 17.2.3.2, “Compatibility with Previous Replication Statements”
Section 17.4.10, “Delayed Replication”
Section 17.1.3.2, “GTID Life Cycle”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.7.2, “Pausing Replication on the Slave”
Section 26.12.11, “Performance Schema Replication Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.3.3, “Replication Privilege Checks”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.4.2.6, “START SLAVE Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 13.4.2.7, “STOP SLAVE Syntax”
Section 17.4.8, “Switching Masters During Failover”
Section 26.12.11.6, “The replication_applier_status_by_worker Table”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

STOP SLAVE SQL_THREAD

Section 13.4.2.2, “CHANGE REPLICATION FILTER Syntax”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

T

[index top]

TRUNCATE PARTITION

Section 15.12.1, “Online DDL Operations”

TRUNCATE TABLE

Section 15.19.6.5, “Adapting DML Statements to memcached Operations”
Section 22.5.15.1, “Adding NDB Cluster Data Nodes Online: General Issues”
Section 13.1.1, “Atomic Data Definition Statement Support”
Section 5.6.7.13, “Clone Plugin Limitations”
Section 22.5.2, “Commands in the NDB Cluster Management Client”
Section 16.2.3.3, “Compressed Table Characteristics”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.2.2, “DELETE Syntax”
Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section 26.12.17.11, “Error Summary Tables”
Section 26.4.3, “Event Pre-Filtering”
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 26.12.17.7, “File I/O Summary Tables”
Section 15.6.3.2, “File-Per-Table Tablespaces”
Section 13.2.4, “HANDLER Syntax”
Section B.4.2.5, “Host 'host_name' is blocked”
Section 15.19.8, “InnoDB memcached Plugin Internals”
Section 22.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 13.3.5, “LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Syntax”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 23.3.4, “Maintenance of Partitions”
Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 26.12.17.10, “Memory Summary Tables”
Section 16.7.2, “MERGE Table Problems”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.4.8, “`ndb_delete_all` — Delete All Rows from an NDB Table”
Section 26.12.17.6, “Object Wait Summary Table”
Section 8.5.7, “Optimizing InnoDB DDL Operations”
Section 26.12.8, “Performance Schema Connection Tables”
Section 26.11, “Performance Schema General Table Characteristics”
Section 26.12.13.1, “Performance Schema persisted_variables Table”
Section 26.12.14, “Performance Schema Status Variable Tables”
Section 26.12.17, “Performance Schema Summary Tables”
Section 26.12.13, “Performance Schema System Variable Tables”
Section 26.12.10, “Performance Schema User-Defined Variable Tables”
Section 26.12.13.2, “Performance Schema variables_info Table”

Section 22.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.21, “Replication and MEMORY Tables”
Section 17.5.1.36, “Replication and TRUNCATE TABLE”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 26.12.17.9, “Socket Summary Tables”
Section 26.12.17.2, “Stage Summary Tables”
Section 26.12.17.4, “Statement Histogram Summary Tables”
Section 26.12.17.3, “Statement Summary Tables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 26.12.17.12, “Status Variable Summary Tables”
Section 26.12.8.1, “The accounts Table”
Section 26.12.16.2, “The clone_progress Table”
Section 26.12.16.1, “The clone_status Table”
Section 26.12.3.1, “The cond_instances Table”
Section 26.12.12.2, “The data_lock_waits Table”
Section 26.12.12.1, “The data_locks Table”
Section 26.12.5.1, “The events_stages_current Table”
Section 26.12.5.2, “The events_stages_history Table”
Section 26.12.5.3, “The events_stages_history_long Table”
Section 26.12.6.1, “The events_statements_current Table”
Section 26.12.6.2, “The events_statements_history Table”
Section 26.12.6.3, “The events_statements_history_long Table”
Section 26.12.7.1, “The events_transactions_current Table”
Section 26.12.7.2, “The events_transactions_history Table”
Section 26.12.7.3, “The events_transactions_history_long Table”
Section 26.12.4.1, “The events_waits_current Table”
Section 26.12.4.2, “The events_waits_history Table”
Section 26.12.4.3, “The events_waits_history_long Table”
Section 26.12.3.2, “The file_instances Table”
Section 26.12.18.1, “The host_cache Table”
Section 26.12.8.2, “The hosts Table”
Section 25.39.19, “The INFORMATION_SCHEMA INNODB_INDEXES Table”
Section 25.39.24, “The INFORMATION_SCHEMA INNODB_TABLES Table”
Section 15.19.7, “The InnoDB memcached Plugin and Replication”
Section 26.12.18.2, “The keyring_keys table”
Section 26.12.18.3, “The log_status Table”
Section 16.3, “The MEMORY Storage Engine”
Section 26.12.12.3, “The metadata_locks Table”
Section 26.12.3.3, “The mutex_instances Table”
Section 26.12.18.4, “The performance_timers Table”
Section 26.12.6.4, “The prepared_statements_instances Table”
Section 27.4.4.24, “The ps_truncate_all_tables() Procedure”
Section 26.12.11.3, “The replication_applier_configuration Table”
Section 26.12.11.4, “The replication_applier_status Table”
Section 26.12.11.1, “The replication_connection_configuration Table”
Section 26.12.11.10, “The replication_group_member_stats Table”
Section 26.12.11.9, “The replication_group_members Table”
Section 26.12.3.4, “The rwlock_instances Table”
Section 26.12.9.1, “The session_account_connect_attrs Table”
Section 26.12.9.2, “The session_connect_attrs Table”
Section 26.12.2.1, “The setup_actors Table”
Section 26.12.2.2, “The setup_consumers Table”

Section 26.12.2.3, “The setup_instruments Table”
Section 26.12.2.4, “The setup_objects Table”
Section 26.12.2.5, “The setup_threads Table”
Section 26.12.3.5, “The socket_instances Table”
Section 26.12.12.4, “The table_handles Table”
The table_io_waits_summary_by_index_usage Table
The table_io_waits_summary_by_table Table
The table_lock_waits_summary_by_table Table
Section 26.12.18.5, “The threads Table”
Section 26.12.15.1, “The tp_thread_group_state Table”
Section 26.12.15.2, “The tp_thread_group_stats Table”
Section 26.12.15.3, “The tp_thread_state Table”
Section 26.12.18.6, “The user_defined_functions Table”
Section 26.12.8.3, “The users Table”
Section 26.12.17.5, “Transaction Summary Tables”
Section 13.1.37, “TRUNCATE TABLE Syntax”
Section 26.12.17.1, “Wait Event Summary Tables”
Section 28.7.29.2, “What Results You Can Get from a Query”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

TRUNCATE TABLE host_cache

Section 26.12.18.1, “The host_cache Table”

U

[index top]

UNINSTALL COMPONENT

Section 13.1.1, “Atomic Data Definition Statement Support”
Section 5.4.2.1, “Error Log Component Configuration”
Section 5.5.3, “Error Log Components”
Section 5.5.1, “Installing and Uninstalling Components”
Section 6.4.3.1, “Password Validation Component Installation and Uninstallation”
Section 6.2.2, “Privileges Provided by MySQL”
Section 6.4.6, “The Audit Message Component”
Section 13.7.4.5, “UNINSTALL COMPONENT Syntax”

UNINSTALL PLUGIN

Section 13.1.1, “Atomic Data Definition Statement Support”
Section 6.4.5.6, “Audit Log Filtering”
Section 6.4.5.9, “Audit Log Reference”
Section 13.7.8.3, “FLUSH Syntax”
Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 13.7.4.4, “INSTALL PLUGIN Syntax”
Section 5.6.1, “Installing and Uninstalling Plugins”
Section 5.6.5.1, “Installing or Uninstalling ddl_rewriter”
Section 6.4.8.2, “Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification”
Section 5.6.6.2, “Installing or Uninstalling Version Tokens”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 6.4.1.8, “No-Login Pluggable Authentication”
Section 6.4.1.5, “PAM Pluggable Authentication”

Section 26.18, “Performance Schema and Plugins”
Section 16.11.1, “Pluggable Storage Engine Architecture”
Section 29.2.3, “Plugin API Components”
Server Plugin Library and Plugin Descriptors
Section 13.7.7.25, “SHOW PLUGINS Syntax”
Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 6.4.1.10, “Test Pluggable Authentication”
Section 25.18, “The INFORMATION_SCHEMA PLUGINS Table”
Section 13.7.4.6, “UNINSTALL PLUGIN Syntax”
Section 6.4.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”
Section 6.4.1.6, “Windows Pluggable Authentication”
Section 29.2.4.8, “Writing Audit Plugins”
Section 29.2.4.5, “Writing Daemon Plugins”
Section 29.2.4.4, “Writing Full-Text Parser Plugins”
Section 29.2.4.6, “Writing INFORMATION_SCHEMA Plugins”
Section 29.2.4.10, “Writing Password-Validation Plugins”
Writing the Server-Side Authentication Plugin

UNION

Section 28.7.5, “C API Data Structures”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.23, “CREATE VIEW Syntax”
Section 8.8.2, “EXPLAIN Output Format”
Section 12.15, “Information Functions”
Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 11.2.5, “Numeric Type Attributes”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”
Section 8.2.1.2, “Range Optimization”
Section 3.6.7, “Searching on Two Keys”
Section 13.2.10, “SELECT Syntax”
Section 5.1.10, “Server Status Variables”
Section 13.2.11, “Subquery Syntax”
Section 16.7, “The MERGE Storage Engine”
Section 13.2.10.3, “UNION Syntax”
Section 24.5.3, “Updatable and Insertable Views”
Section 24.5.1, “View Syntax”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”
Section 12.11, “XML Functions”

UNION ALL

Section 12.15, “Information Functions”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 13.2.10.3, “UNION Syntax”
Section 24.5.3, “Updatable and Insertable Views”

UNION DISTINCT

Section 13.2.10.3, “UNION Syntax”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”

UNLOCK INSTANCE

Section 1.4, “What Is New in MySQL 8.0”

UNLOCK TABLES

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 7.2, “Database Backup Methods”
Section 13.7.8.3, “FLUSH Syntax”
Section 15.7.5.3, “How to Minimize and Handle Deadlocks”
Section 15.6.1.6, “Limits on InnoDB Tables”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 24.8, “Restrictions on Stored Programs”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Transportable Tablespace Examples
Transportable Tablespace Internals

UPDATE

Section 6.2, “Access Control and Account Management”
Section 6.2.7, “Access Control, Stage 2: Request Verification”
Section 6.2.11, “Account Categories”
Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 12.3.4, “Assignment Operators”
Section 6.4.5.6, “Audit Log Filtering”
Section 6.4.5.9, “Audit Log Reference”
Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 15.1.2, “Best Practices for InnoDB Tables”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 28.7.6, “C API Function Overview”
Section 28.7.23, “C API Multiple Statement Execution Support”
Section 28.7.10, “C API Prepared Statement Function Overview”
Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 15.5.2, “Change Buffer”
Section 13.1.20.7, “CHECK Constraints”
Section 13.7.3.2, “CHECK TABLE Syntax”
Section 10.7, “Column Character Set Conversion”
Section 15.9.1.6, “Compression for OLTP Workloads”
Section 15.8.11, “Configuring the Merge Threshold for Index Pages”
Section 15.7.2.3, “Consistent Nonlocking Reads”
Section 1.8.3.3, “Constraints on Invalid Data”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 13.1.20.9, “CREATE TABLE and Generated Columns”
Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 13.1.23, “CREATE VIEW Syntax”

Section 16.8.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 11.7, “Data Type Default Values”
Section 11.1.2, “Date and Time Type Overview”
Section 15.7.5, “Deadlocks in InnoDB”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Syntax”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 15.20.2, “Forcing InnoDB Recovery”
Section 12.9.5, “Full-Text Restrictions”
Section 12.1, “Function and Operator Reference”
Section 8.2.1.20, “Function Call Optimization”
Chapter 12, *Functions and Operators*
Section 8.14.2, “General Thread States”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.3, “Grant Tables”
Section 8.9.4, “Index Hints”
Section 12.15, “Information Functions”
Section 15.7.1, “InnoDB Locking”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 13.2.6.2, “INSERT ... ON DUPLICATE KEY UPDATE Syntax”
Section 13.2.6, “INSERT Syntax”
Section 8.11.1, “Internal Locking Methods”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 25.1, “Introduction”
Section 13.2.10.2, “JOIN Syntax”
Section 12.17.8, “JSON Utility Functions”
Section 13.7.8.4, “KILL Syntax”
Section B.4.7, “Known Issues in MySQL”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 13.2.7, “LOAD DATA Syntax”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 15.7.2.4, “Locking Reads”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 12.24, “Miscellaneous Functions”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 1.8.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 28.7.7.1, “mysql_affected_rows()”
Section 28.7.7.36, “mysql_info()”
Section 28.7.7.38, “mysql_insert_id()”
Section 28.7.7.49, “mysql_num_rows()”
Section 28.7.7.50, “mysql_options()”
Section 28.7.11.10, “mysql_stmt_execute()”
Section 28.7.11.16, “mysql_stmt_insert_id()”
Section 28.7.11.18, “mysql_stmt_num_rows()”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 22.5.13.1, “NDB Cluster Disk Data Objects”
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
Section 8.8.4, “Obtaining Execution Plan Information for a Named Connection”

Section 15.12.1, “Online DDL Operations”
Section 12.3, “Operators”
Section 8.9.3, “Optimizer Hints”
Section 8.2.5, “Optimizing Data Change Statements”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.2, “Optimizing Subqueries, Derived Tables, View References, and Common Table Expressions”
Section 11.2.6, “Out-of-Range and Overflow Handling”
Section 23.1, “Overview of Partitioning in MySQL”
Section 23.4, “Partition Pruning”
Section 23.5, “Partition Selection”
Section 6.1.2.3, “Passwords and Logging”
Section 26.4.6, “Pre-Filtering by Thread”
Section 1.8.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 6.2.2, “Privileges Provided by MySQL”
Section B.4.4.2, “Problems Using DATE Columns”
Section 15.8.9, “Purge Configuration”
Section 8.2.1.2, “Range Optimization”
Section 17.5.1.18, “Replication and LIMIT”
Section 17.5.1.23, “Replication and the Query Optimizer”
Section 17.5.1.35, “Replication and Triggers”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 13.2.11.12, “Rewriting Subqueries as Joins”
Section 13.1.20.10, “Secondary Indexes and Generated Columns”
Section 3.3.4.1, “Selecting All Data”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.11, “Server SQL Modes”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 13.7.1.10, “SET PASSWORD Syntax”
Section 13.7.7.36, “SHOW TABLE STATUS Syntax”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 17.5.1.28, “Slave Errors During Replication”
Section 8.3.3, “SPATIAL Index Optimization”
Section 13.2.11.10, “Subquery Errors”
Section 13.2.11, “Subquery Syntax”
Section 8.11.2, “Table Locking Issues”
Section 16.5, “The ARCHIVE Storage Engine”
Section 10.8.5, “The binary Collation Compared to _bin Collations”
Section 5.4.4, “The Binary Log”
Section 16.6, “The BLACKHOLE Storage Engine”
Section 25.39.27, “The INFORMATION_SCHEMA INNODB_TABLESTATS View”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”
Section 11.6, “The JSON Data Type”
Section 1.3.2, “The Main Features of MySQL”
Section 16.7, “The MERGE Storage Engine”
Section 16.2, “The MyISAM Storage Engine”
Section 5.6.4, “The Rewriter Query Rewrite Plugin”
Section 5.1.17, “The Server Shutdown Process”
Section 27.4.2.3, “The sys_config_update_set_user Trigger”
Section 15.7.2.1, “Transaction Isolation Levels”
Section 24.3.1, “Trigger Syntax and Examples”

Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 15.6.6, “Undo Logs”
Section 24.5.3, “Updatable and Insertable Views”
Section 1.8.2.2, “UPDATE Differences”
Section 13.2.12, “UPDATE Syntax”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”
Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”
Section 1.4, “What Is New in MySQL 8.0”
Section 28.7.29.2, “What Results You Can Get from a Query”
Section 6.2.13, “When Privilege Changes Take Effect”
Section 8.2.1.1, “WHERE Clause Optimization”
Section 28.7.29.1, “Why mysql_store_result() Sometimes Returns NULL After mysql_query() Returns Success”
Section 12.21.5, “Window Function Restrictions”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

UPDATE ... ()

Section 15.7.2.3, “Consistent Nonlocking Reads”

UPDATE ... WHERE

Section 15.7.5, “Deadlocks in InnoDB”

UPDATE ... WHERE ...

Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”

UPDATE IGNORE

Section 13.1.20.7, “CHECK Constraints”
Section 5.1.11, “Server SQL Modes”
Section 13.2.12, “UPDATE Syntax”

USE

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 3.3.1, “Creating and Selecting a Database”
Section 3.3, “Creating and Using a Database”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 25.1, “Introduction”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 22.5.10, “ndbinfo: The NDB Cluster Information Database”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 17.2.5.3, “Replication Rule Application”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 24.2.1, “Stored Routine Syntax”
Section 13.8.4, “USE Syntax”

USE db2
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

USE db_name
Section 4.5.1.1, “mysql Client Options”

USE test
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

W

[index top]

WHERE
Section 15.1.1, “Benefits of Using InnoDB Tables”

WHILE
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.3, “ITERATE Syntax”
Section 13.6.5.4, “LEAVE Syntax”
Section 13.6.2, “Statement Label Syntax”
Section 13.6.5.8, “WHILE Syntax”

WITH
Section 13.2.2, “DELETE Syntax”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 13.2.10, “SELECT Syntax”
Section B.4.6.2, “TEMPORARY Table Problems”
Section 13.2.12, “UPDATE Syntax”
Section 1.4, “What Is New in MySQL 8.0”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”

X

[index top]

XA BEGIN
Section 26.12.7, “Performance Schema Transaction Tables”

XA COMMIT
Section 8.11.4, “Metadata Locking”
Section 26.12.7, “Performance Schema Transaction Tables”
Section 5.1.8, “Server System Variables”
Section 26.12.7.1, “The events_transactions_current Table”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 13.3.8.2, “XA Transaction States”

XA END
Section 13.3.8.3, “Restrictions on XA Transactions”
Section 26.12.7.1, “The events_transactions_current Table”

Section 13.3.8.1, “XA Transaction SQL Syntax”
Section 13.3.8.2, “XA Transaction States”

XA PREPARE

Section 5.1.8, “Server System Variables”
Section 26.12.7.1, “The events_transactions_current Table”
Section 13.3.8.2, “XA Transaction States”

XA RECOVER

Section 13.7.1.6, “GRANT Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.3.8.3, “Restrictions on XA Transactions”
Section 26.12.7.1, “The events_transactions_current Table”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 13.3.8.1, “XA Transaction SQL Syntax”
Section 13.3.8.2, “XA Transaction States”

XA ROLLBACK

Section 8.11.4, “Metadata Locking”
Section 26.12.7, “Performance Schema Transaction Tables”
Section 5.1.8, “Server System Variables”
Section 26.12.7.1, “The events_transactions_current Table”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 13.3.8.2, “XA Transaction States”

XA START

Section 26.12.7, “Performance Schema Transaction Tables”
Section 13.3.8.3, “Restrictions on XA Transactions”
Section 5.1.8, “Server System Variables”
Section 26.12.7.1, “The events_transactions_current Table”
Section 13.3.8.1, “XA Transaction SQL Syntax”
Section 13.3.8.2, “XA Transaction States”

XA START xid

Section 13.3.8.1, “XA Transaction SQL Syntax”

Status Variable Index

A | B | C | D | F | G | H | I | K | L | M | N | O | P | Q | R | S | T | U | V

A

[index top]

Aborted_clients

Section B.4.2.10, “Communication Errors and Aborted Connections”
Section 5.1.10, “Server Status Variables”

Aborted_connects

Section B.4.2.10, “Communication Errors and Aborted Connections”
Section 5.1.10, “Server Status Variables”

Acl_cache_items_count
Section 5.1.10, “Server Status Variables”

Audit_log_current_size
Section 6.4.5.9, “Audit Log Reference”

Audit_log_event_max_drop_size
Section 6.4.5.9, “Audit Log Reference”

Audit_log_events
Section 6.4.5.9, “Audit Log Reference”

Audit_log_events_filtered
Section 6.4.5.9, “Audit Log Reference”

Audit_log_events_lost
Section 6.4.5.9, “Audit Log Reference”

Audit_log_events_written
Section 6.4.5.9, “Audit Log Reference”

Audit_log_total_size
Section 6.4.5.9, “Audit Log Reference”

Audit_log_write_waits
Section 6.4.5.9, “Audit Log Reference”

B

[index top]

Binlog_cache_disk_use
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 5.1.10, “Server Status Variables”
Section 5.4.4, “The Binary Log”

Binlog_cache_use
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 5.1.10, “Server Status Variables”
Section 5.4.4, “The Binary Log”

Binlog_stmt_cache_disk_use
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 5.1.10, “Server Status Variables”

Binlog_stmt_cache_use
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 5.1.10, “Server Status Variables”

Bytes_received
Section 5.1.10, “Server Status Variables”

Section 5.4.5, “The Slow Query Log”

Bytes_sent

Section 5.1.10, “Server Status Variables”
Section 5.4.5, “The Slow Query Log”

C

[index top]

Caching_sha2_password_rsa_public_key

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 5.1.10, “Server Status Variables”

Com_flush

Section 5.1.10, “Server Status Variables”

Com_stmt_reprepare

Section 8.10.3, “Caching of Prepared Statements and Stored Programs”

Compression

Section 4.2.6, “Connection Compression Control”
Section 5.1.10, “Server Status Variables”

Compression_algorithm

Section 4.2.6, “Connection Compression Control”
Section 5.1.10, “Server Status Variables”

Compression_level

Section 4.2.6, “Connection Compression Control”
Section 5.1.10, “Server Status Variables”

Connection_control_delay_generated

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 6.4.2.2, “Connection-Control System and Status Variables”

Connection_errors_accept

Section 5.1.10, “Server Status Variables”

Connection_errors_internal

Section 5.1.10, “Server Status Variables”

Connection_errors_max_connections

Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 5.1.10, “Server Status Variables”

Connection_errors_peer_address

Section 5.1.10, “Server Status Variables”

Connection_errors_select

Section 5.1.10, “Server Status Variables”

Connection_errors_tcpwrap
Section 5.1.10, “Server Status Variables”

Connection_errors_xxx
Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section 5.1.10, “Server Status Variables”

Connections
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

Created_tmp_disk_tables
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 26.12.6.1, “The events_statements_current Table”
Section 5.4.5, “The Slow Query Log”

Created_tmp_files
Section 5.1.10, “Server Status Variables”

Created_tmp_tables
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 13.7.7.35, “SHOW STATUS Syntax”
Section 26.12.6.1, “The events_statements_current Table”
Section 5.4.5, “The Slow Query Log”

Current_tls_ca
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.10, “Server Status Variables”

Current_tls_capath
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.10, “Server Status Variables”

Current_tls_cert
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.10, “Server Status Variables”

Current_tls_cipher
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.10, “Server Status Variables”

Current_tls_ciphersuites
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.10, “Server Status Variables”

Current_tls_crl
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”

Section 5.1.10, “Server Status Variables”

Current_tls_crlpath

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.10, “Server Status Variables”

Current_tls_key

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.10, “Server Status Variables”

Current_tls_version

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.10, “Server Status Variables”

D

[index top]

Delayed_errors

Section 5.1.10, “Server Status Variables”

Delayed_insert_threads

Section 5.1.10, “Server Status Variables”

Delayed_writes

Section 5.1.10, “Server Status Variables”

dragnet.Status

Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

F

[index top]

Firewall_access_denied

Section 6.4.7.4, “MySQL Enterprise Firewall Reference”

Firewall_access_granted

Section 6.4.7.4, “MySQL Enterprise Firewall Reference”
Section 6.4.7.3, “Using MySQL Enterprise Firewall”

Firewall_access_suspicious

Section 6.4.7.4, “MySQL Enterprise Firewall Reference”

Firewall_cached_entries

Section 6.4.7.4, “MySQL Enterprise Firewall Reference”

Flush_commands

Section 5.1.10, “Server Status Variables”

G

[index top]

group_replication_primary_member
Finding the Primary
Section 18.8, “Group Replication System Variables”
Section 5.1.10, “Server Status Variables”

H

[index top]

Handler_commit
Section 5.1.10, “Server Status Variables”

Handler_delete
Section 5.1.10, “Server Status Variables”

Handler_discover
NDB Cluster Status Variables

Handler_external_lock
Section 5.1.10, “Server Status Variables”

Handler_mrr_init
Section 5.1.10, “Server Status Variables”

Handler_prepare
Section 5.1.10, “Server Status Variables”

Handler_read_first
Section 5.1.10, “Server Status Variables”
Section 5.4.5, “The Slow Query Log”

Handler_read_key
Section 5.1.10, “Server Status Variables”
Section 5.4.5, “The Slow Query Log”

Handler_read_last
Section 5.1.10, “Server Status Variables”
Section 5.4.5, “The Slow Query Log”

Handler_read_next
Section 5.1.10, “Server Status Variables”
Section 5.4.5, “The Slow Query Log”
Section 8.3.10, “Use of Index Extensions”

Handler_read_prev
Section 5.1.10, “Server Status Variables”

Section 5.4.5, “The Slow Query Log”

Handler_read_rnd

Section 5.1.10, “Server Status Variables”
Section 5.4.5, “The Slow Query Log”

Handler_read_rnd_next

Section 5.1.10, “Server Status Variables”
Section 5.4.5, “The Slow Query Log”

Handler_rollback

Section 5.1.10, “Server Status Variables”

Handler_savepoint

Section 5.1.10, “Server Status Variables”

Handler_savepoint_rollback

Section 5.1.10, “Server Status Variables”

Handler_update

Section 5.1.10, “Server Status Variables”

Handler_write

Section 5.1.10, “Server Status Variables”

I

[index top]

Innodb_available_undo_logs

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_bytes_data

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_bytes_dirty

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_dump_status

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_load_status

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_pages_data

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_pages_dirty

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_pages_flushed

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_pages_free

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_pages_latched

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_pages_misc

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_pages_total

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_read_ahead

Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_read_ahead_evicted

Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_read_ahead_rnd

Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_read_requests

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_reads

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_resize_status

Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_wait_free

Section 5.1.10, “Server Status Variables”

Innodb_buffer_pool_write_requests

Section 5.1.10, “Server Status Variables”

Innodb_data_fsyncs

Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.10, “Server Status Variables”

Innodb_data_pending_fsyncs
Section 5.1.10, “Server Status Variables”

Innodb_data_pending_reads
Section 5.1.10, “Server Status Variables”

Innodb_data_pending_writes
Section 5.1.10, “Server Status Variables”

Innodb_data_read
Section 5.1.10, “Server Status Variables”

Innodb_data_reads
Section 5.1.10, “Server Status Variables”

Innodb_data_writes
Section 5.1.10, “Server Status Variables”

Innodb_data_written
Section 5.1.10, “Server Status Variables”

Innodb_dblwr_pages_written
Section 5.1.10, “Server Status Variables”

Innodb_dblwr_writes
Section 5.1.10, “Server Status Variables”

Innodb_have_atomic_builtins
Section 5.1.10, “Server Status Variables”

Innodb_log_waits
Section 5.1.10, “Server Status Variables”

Innodb_log_write_requests
Section 5.1.10, “Server Status Variables”

Innodb_log_writes
Section 5.1.10, “Server Status Variables”

Innodb_num_open_files
Section 5.1.10, “Server Status Variables”

Innodb_os_log_fsyncs
Section 5.1.10, “Server Status Variables”

Innodb_os_log_pending_fsyncs
Section 5.1.10, “Server Status Variables”

Innodb_os_log_pending_writes
Section 5.1.10, “Server Status Variables”

Innodb_os_log_written
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.10, “Server Status Variables”

Innodb_page_size
Section 5.1.10, “Server Status Variables”

Innodb_pages_created
Section 5.1.10, “Server Status Variables”

Innodb_pages_read
Section 5.1.10, “Server Status Variables”

Innodb_pages_written
Section 5.1.10, “Server Status Variables”

Innodb_row_lock_current_waits
Section 5.1.10, “Server Status Variables”

Innodb_row_lock_time
Section 5.1.10, “Server Status Variables”

Innodb_row_lock_time_avg
Section 5.1.10, “Server Status Variables”

Innodb_row_lock_time_max
Section 5.1.10, “Server Status Variables”

Innodb_row_lock_waits
Section 5.1.10, “Server Status Variables”

Innodb_rows_deleted
Section 5.1.10, “Server Status Variables”

Innodb_rows_inserted
Section 5.1.10, “Server Status Variables”

Innodb_rows_read
Section 5.1.10, “Server Status Variables”

Innodb_rows_updated
Section 5.1.10, “Server Status Variables”

Innodb_system_rows_deleted
Section 5.1.10, “Server Status Variables”

Innodb_system_rows_inserted
Section 5.1.10, “Server Status Variables”

Innodb_system_rows_read
Section 5.1.10, “Server Status Variables”

Innodb_truncated_status_writes
Section 5.1.10, “Server Status Variables”

K

[index top]

Key_blocks_not_flushed
Section 5.1.10, “Server Status Variables”

Key_blocks_unused
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

Key_blocks_used
Section 5.1.10, “Server Status Variables”

Key_read_requests
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

Key_reads
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

Key_write_requests
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

Key_writes
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

L

[index top]

Last_query_cost
Section 5.1.10, “Server Status Variables”

Last_query_partial_plans
Section 5.1.10, “Server Status Variables”

Locked_connects
Section 6.2.19, “Account Locking”
Section 5.1.10, “Server Status Variables”

M

[index top]

Max_execution_time_exceeded
Section 5.1.10, “Server Status Variables”

Max_execution_time_set
Section 5.1.10, “Server Status Variables”

Max_execution_time_set_failed
Section 5.1.10, “Server Status Variables”

Max_used_connections
Section 5.1.10, “Server Status Variables”

Max_used_connections_time
Section 5.1.10, “Server Status Variables”

mecab_charset
Section 12.9.9, “MeCab Full-Text Parser Plugin”
Section 5.1.10, “Server Status Variables”

Mysqlx_aborted_clients
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_address
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_bytes_received
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_bytes_sent
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_connection_accept_errors
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_connection_errors
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_connections_accepted
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_connections_closed
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_connections_rejected
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_crud_create_view
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_crud_delete
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_crud_drop_view
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_crud_find
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_crud_insert
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_crud_modify_view
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_crud_update
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_cursor_close
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_cursor_fetch
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_cursor_open
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_errors_sent
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_expect_close
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_expect_open
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_init_error
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_notice_global_sent
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_notice_other_sent
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_notice_warning_sent
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_notified_by_group_replication
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_port
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_prep_deallocate
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_prep_execute
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_prep_prepare
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_rows_sent
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_sessions
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_sessions_accepted
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_sessions_closed
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_sessions_fatal_error
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_sessions_killed
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_sessions_rejected
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_socket
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_accept_renegotiates
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_accepts
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_active
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_cipher
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_cipher_list
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_ctx_verify_depth
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_ctx_verify_mode
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_finished_accepts
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_server_not_after
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_server_not_before
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_verify_depth
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_verify_mode
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_ssl_version
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_create_collection
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_create_collection_index
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_disable_notices
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_drop_collection
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_drop_collection_index
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_enable_notices
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_ensure_collection
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_execute_mysqlx
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_execute_sql
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_execute_xplugin
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_kill_client
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_list_clients
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_list_notices
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_list_objects
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_stmt_ping
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_worker_threads
Section 20.5.6.1, “Status Variables for X Plugin”

Mysqlx_worker_threads_active
Section 20.5.6.1, “Status Variables for X Plugin”

N

[index top]

Ndb_api_bytes_received_count
Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_bytes_received_count_session
Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_bytes_received_count_slave
Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_bytes_sent_count
Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_bytes_sent_count_session
Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_bytes_sent_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_bytes_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_bytes_count_injector

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_data_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_data_count_injector

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_nondata_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_nondata_count_injector

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pk_op_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pk_op_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pk_op_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pruned_scan_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pruned_scan_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pruned_scan_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_range_scan_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_range_scan_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_range_scan_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_read_row_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_read_row_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_read_row_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_scan_batch_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_scan_batch_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_scan_batch_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_table_scan_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_table_scan_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_table_scan_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_abort_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_abort_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_abort_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_close_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_close_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_close_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_commit_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_commit_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_commit_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_local_read_row_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_local_read_row_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_local_read_row_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_start_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_start_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_start_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_uk_op_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_uk_op_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_uk_op_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_exec_complete_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_exec_complete_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_exec_complete_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_meta_request_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_meta_request_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_meta_request_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_nanos_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_nanos_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_nanos_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”

NDB Cluster Status Variables

Ndb_api_wait_scan_result_count

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_scan_result_count_session

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_scan_result_count_slave

Section 22.5.17, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_cluster_node_id

NDB Cluster Status Variables

Ndb_config_from_host

NDB Cluster Status Variables

Ndb_config_from_port

NDB Cluster Status Variables

Ndb_conflict_fn_epoch

Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

Ndb_conflict_fn_epoch2

Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

Ndb_conflict_fn_epoch2_trans

Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

Ndb_conflict_fn_epoch_trans

Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster Status Variables

Ndb_conflict_fn_max

Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

Ndb_conflict_fn_old

Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

Ndb_conflict_last_conflict_epoch

NDB Cluster Status Variables

Ndb_conflict_reflected_op_discard_count

Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

Ndb_conflict_reflected_op_prepare_count

Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

Ndb_conflict_trans_conflict_commit_count
NDB Cluster Status Variables

Ndb_conflict_trans_row_conflict_count
NDB Cluster Status Variables

Ndb_conflict_trans_row_reject_count
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

Ndb_epoch_delete_delete_count
NDB Cluster Status Variables

Ndb_execute_count
NDB Cluster Status Variables

Ndb_last_commit_epoch_server
NDB Cluster Status Variables

Ndb_last_commit_epoch_session
NDB Cluster Status Variables

Ndb_metadata_blacklist_size
NDB Cluster Status Variables
Section 22.1.4, “What is New in NDB Cluster”

Ndb_metadata_detected_count
NDB Cluster Status Variables
Section 22.1.4, “What is New in NDB Cluster”

Ndb_metadata_synced_count
NDB Cluster Status Variables
Section 22.1.4, “What is New in NDB Cluster”

Ndb_number_of_data_nodes
NDB Cluster Status Variables

Ndb_pruned_scan_count
NDB Cluster Status Variables

Ndb_pushed_queries_defined
NDB Cluster Status Variables
NDB Cluster System Variables

Ndb_pushed_queries_dropped
NDB Cluster Status Variables
NDB Cluster System Variables

Ndb_pushed_queries_executed
NDB Cluster Status Variables

NDB Cluster System Variables

Ndb_pushed_reads

NDB Cluster Status Variables
NDB Cluster System Variables

Ndb_scan_count

NDB Cluster Status Variables

Ndb_slave_max_replicated_epoch

NDB Cluster Status Variables

Ndb_system_name

Section 22.3.3.8, “Defining the System”
NDB Cluster Status Variables

Ndb_trans_hint_count_session

NDB Cluster Status Variables

Not_flushed_delayed_rows

Section 5.1.10, “Server Status Variables”

O

[index top]

Ongoing_anonymous_gtid_violating_transaction_count

Section 5.1.10, “Server Status Variables”

Ongoing_anonymous_transaction_count

Section 5.1.10, “Server Status Variables”

Ongoing_automatic_gtid_violating_transaction_count

Section 5.1.10, “Server Status Variables”

Open_files

Section 5.1.10, “Server Status Variables”

Open_streams

Section 5.1.10, “Server Status Variables”

Open_table_definitions

Section 5.1.10, “Server Status Variables”

Open_tables

Section 5.1.10, “Server Status Variables”

Opened_files

Section 5.1.10, “Server Status Variables”

Opened_table_definitions
Section 5.1.10, “Server Status Variables”

Opened_tables
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

P

[index top]

Performance_schema_accounts_lost
Section 26.16, “Performance Schema Status Variables”

Performance_schema_cond_classes_lost
Section 26.16, “Performance Schema Status Variables”

Performance_schema_cond_instances_lost
Section 26.16, “Performance Schema Status Variables”

Performance_schema_digest_lost
Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

Performance_schema_file_classes_lost
Section 26.16, “Performance Schema Status Variables”

Performance_schema_file_handles_lost
Section 26.16, “Performance Schema Status Variables”

Performance_schema_file_instances_lost
Section 26.16, “Performance Schema Status Variables”

Performance_schema_hosts_lost
Section 26.16, “Performance Schema Status Variables”

Performance_schema_index_stat_lost
Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

Performance_schema_locker_lost
Section 26.16, “Performance Schema Status Variables”

Performance_schema_memory_classes_lost
Section 26.16, “Performance Schema Status Variables”

Performance_schema_metadata_lock_lost
Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

Performance_schema_mutex_classes_lost

Section 26.7, “Performance Schema Status Monitoring”
Section 26.16, “Performance Schema Status Variables”

Performance_schema_mutex_instances_lost

Section 26.7, “Performance Schema Status Monitoring”
Section 26.16, “Performance Schema Status Variables”

Performance_schema_nested_statement_lost

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

Performance_schema_prepared_statements_lost

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”
Section 26.12.6.4, “The prepared_statements_instances Table”

Performance_schema_program_lost

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

Performance_schema_rwlock_classes_lost

Section 26.16, “Performance Schema Status Variables”

Performance_schema_rwlock_instances_lost

Section 26.16, “Performance Schema Status Variables”

Performance_schema_session_connect_attrs_longest_seen

Section 26.12.9, “Performance Schema Connection Attribute Tables”
Section 26.16, “Performance Schema Status Variables”

Performance_schema_session_connect_attrs_lost

Section 26.12.9, “Performance Schema Connection Attribute Tables”
Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

Performance_schema_socket_classes_lost

Section 26.16, “Performance Schema Status Variables”

Performance_schema_socket_instances_lost

Section 26.16, “Performance Schema Status Variables”

Performance_schema_stage_classes_lost

Section 26.16, “Performance Schema Status Variables”

Performance_schema_statement_classes_lost

Section 26.16, “Performance Schema Status Variables”

Performance_schema_table_handles_lost

Section 26.16, “Performance Schema Status Variables”

Section 26.15, “Performance Schema System Variables”

Performance_schema_table_instances_lost

Section 26.16, “Performance Schema Status Variables”

Performance_schema_table_lock_stat_lost

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

Performance_schema_thread_classes_lost

Section 26.16, “Performance Schema Status Variables”

Performance_schema_thread_instances_lost

Section 12.22, “Performance Schema Functions”
Section 26.16, “Performance Schema Status Variables”
Section 26.12.13, “Performance Schema System Variable Tables”
Section 26.15, “Performance Schema System Variables”

Performance_schema_users_lost

Section 26.16, “Performance Schema Status Variables”

Prepared_stmt_count

Section 5.1.10, “Server Status Variables”

Q

[index top]

Qcache_free_blocks

Section 5.1.10, “Server Status Variables”
Section 1.4, “What Is New in MySQL 8.0”

Qcache_free_memory

Section 5.1.10, “Server Status Variables”
Section 1.4, “What Is New in MySQL 8.0”

Qcache_hits

Section 5.1.10, “Server Status Variables”
Section 1.4, “What Is New in MySQL 8.0”

Qcache_inserts

Section 5.1.10, “Server Status Variables”
Section 1.4, “What Is New in MySQL 8.0”

Qcache_lowmem_prunes

Section 5.1.10, “Server Status Variables”
Section 1.4, “What Is New in MySQL 8.0”

Qcache_not_cached

Section 5.1.10, “Server Status Variables”
Section 1.4, “What Is New in MySQL 8.0”

Qcache_queries_in_cache
Section 5.1.10, “Server Status Variables”
Section 1.4, “What Is New in MySQL 8.0”

Qcache_total_blocks
Section 5.1.10, “Server Status Variables”
Section 1.4, “What Is New in MySQL 8.0”

Queries
Section 5.1.10, “Server Status Variables”

Questions
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 5.1.10, “Server Status Variables”

R

[index top]

Rewriter_number_loaded_rules
Rewriter Query Rewrite Plugin Status Variables

Rewriter_number_reloads
Rewriter Query Rewrite Plugin Status Variables

Rewriter_number_rewritten_queries
Rewriter Query Rewrite Plugin Status Variables

Rewriter_reload_error
Rewriter Query Rewrite Plugin Procedures and Functions
Rewriter Query Rewrite Plugin Rules Table
Rewriter Query Rewrite Plugin Status Variables
Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”

Rpl_semi_sync_master_clients
Section 17.4.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.4.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_net_avg_wait_time
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_net_wait_time
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_net_waits
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_no_times
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_no_tx
Section 17.4.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.4.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_status
Section 17.4.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.4.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_timefunc_failures
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_tx_avg_wait_time
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_tx_wait_time
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_tx_waits
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_wait_pos_backtraverse
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_wait_sessions
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_master_yes_tx
Section 17.4.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.4.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.10, “Server Status Variables”

Rpl_semi_sync_slave_status
Section 17.4.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.4.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.10, “Server Status Variables”

Rsa_public_key
Section 5.1.10, “Server Status Variables”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”

S

[index top]

Secondary_engine_execution_count
Section 5.1.10, “Server Status Variables”

Select_full_join
Section 5.1.10, “Server Status Variables”

Section 26.12.6.1, “The events_statements_current Table”

Select_full_range_join

Section 5.1.10, “Server Status Variables”
Section 26.12.6.1, “The events_statements_current Table”

Select_range

Section 5.1.10, “Server Status Variables”
Section 26.12.6.1, “The events_statements_current Table”

Select_range_check

Section 5.1.10, “Server Status Variables”
Section 26.12.6.1, “The events_statements_current Table”

Select_scan

Section 5.1.10, “Server Status Variables”
Section 26.12.6.1, “The events_statements_current Table”

Slave_heartbeat_period

Section 17.1.7.1, “Checking Replication Status”
Section 26.12.11, “Performance Schema Replication Tables”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 5.1.10, “Server Status Variables”

Slave_last_heartbeat

Section 17.1.7.1, “Checking Replication Status”
Section 26.12.11, “Performance Schema Replication Tables”
Section 5.1.10, “Server Status Variables”

Slave_open_temp_tables

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 5.1.10, “Server Status Variables”
Section 13.4.2.7, “STOP SLAVE Syntax”

Slave_received_heartbeats

Section 17.1.7.1, “Checking Replication Status”
Section 26.12.11, “Performance Schema Replication Tables”
Section 5.1.10, “Server Status Variables”

Slave_retried_transactions

Section 17.1.7.1, “Checking Replication Status”
Section 26.12.11, “Performance Schema Replication Tables”
Section 5.1.10, “Server Status Variables”

Slave_rows_last_search_algorithm_used

Section 5.1.10, “Server Status Variables”

Slave_running

Section 17.1.7.1, “Checking Replication Status”
Section 26.12.11, “Performance Schema Replication Tables”

Section 17.2.2, “Replication Implementation Details”
Section 5.1.10, “Server Status Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”

Slow_launch_threads

Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

Slow_queries

Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

Sort_merge_passes

Section 8.2.1.16, “ORDER BY Optimization”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 26.12.6.1, “The events_statements_current Table”
Section 5.4.5, “The Slow Query Log”

Sort_range

Section 5.1.10, “Server Status Variables”
Section 26.12.6.1, “The events_statements_current Table”
Section 5.4.5, “The Slow Query Log”

Sort_rows

Section 5.1.10, “Server Status Variables”
Section 26.12.6.1, “The events_statements_current Table”
Section 5.4.5, “The Slow Query Log”

Sort_scan

Section 5.1.10, “Server Status Variables”
Section 26.12.6.1, “The events_statements_current Table”
Section 5.4.5, “The Slow Query Log”

Ssl_accept_renegotiates

Section 5.1.10, “Server Status Variables”

Ssl_accepts

Section 5.1.10, “Server Status Variables”

Ssl_callback_cache_hits

Section 5.1.10, “Server Status Variables”

Ssl_cipher

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 5.1.10, “Server Status Variables”

Ssl_cipher_list

Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 5.1.10, “Server Status Variables”

Ssl_client_connects
Section 5.1.10, “Server Status Variables”

Ssl_connect_renegotiates
Section 5.1.10, “Server Status Variables”

Ssl_ctx_verify_depth
Section 5.1.10, “Server Status Variables”

Ssl_ctx_verify_mode
Section 5.1.10, “Server Status Variables”

Ssl_default_timeout
Section 5.1.10, “Server Status Variables”

Ssl_finished_accepts
Section 5.1.10, “Server Status Variables”

Ssl_finished_connects
Section 5.1.10, “Server Status Variables”

Ssl_server_not_after
Section 5.1.10, “Server Status Variables”

Ssl_server_not_before
Section 5.1.10, “Server Status Variables”

Ssl_session_cache_hits
Section 5.1.10, “Server Status Variables”

Ssl_session_cache_misses
Section 5.1.10, “Server Status Variables”

Ssl_session_cache_mode
Section 5.1.10, “Server Status Variables”

Ssl_session_cache_overflows
Section 5.1.10, “Server Status Variables”

Ssl_session_cache_size
Section 5.1.10, “Server Status Variables”

Ssl_session_cache_timeouts
Section 5.1.10, “Server Status Variables”

Ssl_sessions_reused
Section 5.1.10, “Server Status Variables”

Ssl_used_session_cache_entries
Section 5.1.10, “Server Status Variables”

Ssl_verify_depth
Section 5.1.10, “Server Status Variables”

Ssl_verify_mode
Section 5.1.10, “Server Status Variables”

Ssl_version
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 5.1.10, “Server Status Variables”

T

[index top]

Table_locks_immediate
Section 8.11.1, “Internal Locking Methods”
Section 5.1.10, “Server Status Variables”

Table_locks_waited
Section 8.11.1, “Internal Locking Methods”
Section 5.1.10, “Server Status Variables”

Table_open_cache_hits
Section 5.1.10, “Server Status Variables”

Table_open_cache_misses
Section 5.1.10, “Server Status Variables”

Table_open_cache_overflows
Section 5.1.10, “Server Status Variables”

Tc_log_max_pages_used
Section 5.1.10, “Server Status Variables”

Tc_log_page_size
Section 5.1.10, “Server Status Variables”

Tc_log_page_waits
Section 5.1.10, “Server Status Variables”

Threads_cached
Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 5.1.10, “Server Status Variables”

Threads_connected
Section 5.1.10, “Server Status Variables”

Threads_created
Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

Threads_running
Section A.15, “MySQL 8.0 FAQ: MySQL Enterprise Thread Pool”
Section 5.1.10, “Server Status Variables”

U

[index top]

Uptime
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 5.1.10, “Server Status Variables”

Uptime_since_flush_status
Section 5.1.10, “Server Status Variables”

V

[index top]

validate_password.dictionary_file_last_parsed
Section 6.4.3.2, “Password Validation Options and Variables”

validate_password.dictionary_file_words_count
Section 6.4.3.2, “Password Validation Options and Variables”

validate_password_dictionary_file_last_parsed
Section 6.4.3.2, “Password Validation Options and Variables”

validate_password_dictionary_file_words_count
Section 6.4.3.2, “Password Validation Options and Variables”

System Variable Index

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W

A

[index top]

activate_all_roles_on_login
Section 17.3.3, “Replication Privilege Checks”
Section 5.1.8, “Server System Variables”
Section 13.7.1.11, “SET ROLE Syntax”
Section 13.7.7.21, “SHOW GRANTS Syntax”
Section 6.2.10, “Using Roles”

admin_address
Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 5.1.8, “Server System Variables”

admin_port
Section 8.12.4.1, “How MySQL Handles Client Connections”

Section 5.1.8, “Server System Variables”

audit_log_buffer_size

Section 6.4.5.5, “Audit Log Logging Configuration”
Section 6.4.5.9, “Audit Log Reference”

audit_log_compression

Section 6.4.5.5, “Audit Log Logging Configuration”
Section 6.4.5.9, “Audit Log Reference”

audit_log_connection_policy

Section 6.4.5.9, “Audit Log Reference”
Section 6.4.5.8, “Legacy Mode Audit Log Filtering”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

audit_log_current_session

Section 6.4.5.9, “Audit Log Reference”

audit_log_encryption

Section 6.4.5.5, “Audit Log Logging Configuration”
Section 6.4.5.9, “Audit Log Reference”

audit_log_exclude_accounts

Section 6.4.5.9, “Audit Log Reference”
Section 6.4.5.8, “Legacy Mode Audit Log Filtering”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

audit_log_file

Section 6.4.5.5, “Audit Log Logging Configuration”
Section 6.4.5.9, “Audit Log Reference”
Section 6.4.5, “MySQL Enterprise Audit”
Section 6.4.5.3, “MySQL Enterprise Audit Security Considerations”

audit_log_filter_id

Section 6.4.5.6, “Audit Log Filtering”
Section 6.4.5.9, “Audit Log Reference”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

audit_log_flush

Section 6.4.5.5, “Audit Log Logging Configuration”
Section 6.4.5.9, “Audit Log Reference”

audit_log_format

Section 6.4.5.4, “Audit Log File Formats”
Section 6.4.5.5, “Audit Log Logging Configuration”
Section 6.4.5.9, “Audit Log Reference”
Section 6.4.5, “MySQL Enterprise Audit”
Section 6.4.6, “The Audit Message Component”

audit_log_include_accounts

Section 6.4.5.9, “Audit Log Reference”

Section 6.4.5.8, “Legacy Mode Audit Log Filtering”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

audit_log_password_history_keep_days

Section 6.4.5.9, “Audit Log Reference”

audit_log_policy

Section 6.4.5.9, “Audit Log Reference”
Section 6.4.5.8, “Legacy Mode Audit Log Filtering”
Section 5.1.9, “Using System Variables”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

audit_log_read_buffer_size

Section 6.4.5.5, “Audit Log Logging Configuration”
Section 6.4.5.9, “Audit Log Reference”

audit_log_rotate_on_size

Section 6.4.5.5, “Audit Log Logging Configuration”
Section 6.4.5.9, “Audit Log Reference”

audit_log_statement_policy

Section 6.4.5.9, “Audit Log Reference”
Section 6.4.5.8, “Legacy Mode Audit Log Filtering”
Section 6.4.5.7, “Writing Audit Log Filter Definitions”

audit_log_strategy

Section 6.4.5.5, “Audit Log Logging Configuration”
Section 6.4.5.9, “Audit Log Reference”

authentication_ldap_sasl_auth_method_name

Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_bind_base_dn

Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_bind_root_dn

Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_bind_root_pwd

Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_ca_path

Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_group_search_attr

Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_group_search_filter

Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_init_pool_size
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_log_status
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_max_pool_size
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_server_host
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_server_port
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_tls
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_sasl_user_search_attr
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_auth_method_name
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_bind_base_dn
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_bind_root_dn
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_bind_root_pwd
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_ca_path
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_group_search_attr
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_group_search_filter
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_init_pool_size
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_log_status
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_max_pool_size

Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_server_host

Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_server_port

Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_tls

Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_ldap_simple_user_search_attr

Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 6.4.1.11, “Pluggable Authentication System Variables”

authentication_windows_log_level

Section 5.1.8, “Server System Variables”
Section 6.4.1.6, “Windows Pluggable Authentication”

authentication_windows_use_principal_name

Section 5.1.8, “Server System Variables”
Section 6.4.1.6, “Windows Pluggable Authentication”

auto_generate_certs

Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”
Section 5.1.8, “Server System Variables”

auto_increment_increment

Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 18.10, “Frequently Asked Questions”
Section 18.8, “Group Replication System Variables”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section A.1, “MySQL 8.0 FAQ: General”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 17.5.1.38, “Replication and Variables”
Section 17.1.6.2, “Replication Master Options and Variables”
Section 3.6.9, “Using AUTO_INCREMENT”
Section 21.4, “Working with InnoDB Cluster”

auto_increment_offset

Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 18.10, “Frequently Asked Questions”
Section 18.8, “Group Replication System Variables”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section A.1, “MySQL 8.0 FAQ: General”
Section 22.4.13, “`ndb_import` — Import CSV Data Into NDB”
Section 17.5.1.38, “Replication and Variables”
Section 17.1.6.2, “Replication Master Options and Variables”
Section 3.6.9, “Using AUTO_INCREMENT”
Section 21.4, “Working with InnoDB Cluster”

AUTOCOMMIT
Section 17.5.1.34, “Replication and Transactions”

autocommit
Section 13.1.10, “ALTER TABLESPACE Syntax”
Section 15.7.2.2, “autocommit, Commit, and Rollback”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 15.7.5.2, “Deadlock Detection and Rollback”
Section 13.2.2, “DELETE Syntax”
Section 15.6.3.3, “General Tablespaces”
Section 15.7, “InnoDB Locking and Transaction Model”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”
Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 15.7.2.4, “Locking Reads”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 28.7.7.69, “mysql_session_track_get_first()”
NDB Cluster System Variables
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 26.12.7, “Performance Schema Transaction Tables”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.5.1.34, “Replication and Transactions”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 5.1.8, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 25.39.29, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 5.6.3.3, “Thread Pool Operation”
Section 15.7.2.1, “Transaction Isolation Levels”

automatic_sp_privileges
Section 13.1.7, “ALTER PROCEDURE Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 5.1.8, “Server System Variables”
Section 24.2.2, “Stored Routines and MySQL Privileges”

avoid_temporal_upgrade
Section 13.7.3.2, “CHECK TABLE Syntax”
Section 13.7.3.5, “REPAIR TABLE Syntax”
Section 5.1.8, “Server System Variables”

B

[index top]

back_log
Section 5.1.8, “Server System Variables”

basedir
Section 13.7.4.4, “INSTALL PLUGIN Syntax”
Section 2.4.3, “Installing and Using the MySQL Launch Daemon”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

big_tables

Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.8, “Server System Variables”

bind_address

Section B.4.2.2, “Can't connect to [local] MySQL server”
Section 5.1.12.2, “Configuring the MySQL Server to Permit IPv6 Connections”
Section 5.1.12.4, “Connecting Using IPv6 Nonlocal Host Addresses”
Section 5.1.12.3, “Connecting Using the IPv6 Local Host Address”
Section 18.5.1, “Group Replication IP Address Whitelisting”
Section 5.1.12, “IPv6 Support”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”
Section 5.1.12.5, “Obtaining an IPv6 Address from a Broker”
Section 5.8, “Running Multiple MySQL Instances on One Machine”
Section 5.1.8, “Server System Variables”
Section 18.4.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”
Section 22.5.10.30, “The ndbinfo processes Table”
Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
Section 20.5.5.2, “X Plugin Options and System Variables”

binlog

Section 18.9.2, “Group Replication Limitations”
Section 18.9.1, “Group Replication Requirements”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

binlog_cache_size

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 5.1.10, “Server Status Variables”
Section 5.4.4, “The Binary Log”

binlog_checksum

Section 17.1.6.4, “Binary Logging Options and Variables”
MySQL Glossary
Section 17.5.1.34, “Replication and Transactions”
Section 5.4.4, “The Binary Log”

binlog_direct_non_transactional_updates

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.5.1.34, “Replication and Transactions”

binlog_encryption

Section 13.1.5, “ALTER INSTANCE Syntax”
Section 17.3.2.3, “Binary Log Master Key Rotation”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.3, “Replication Security”
Section 13.7.7.1, “SHOW BINARY LOGS Syntax”
Section 5.4.4, “The Binary Log”

Section 6.4.4, “The MySQL Keyring”

binlog_error_action

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.3.1, “GTID Format and Storage”
Section 5.4.4, “The Binary Log”

binlog_expire_logs_seconds

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.3.3, “GTID Auto-Positioning”
Section 13.4.1.1, “PURGE BINARY LOGS Syntax”
Section 5.4.7, “Server Log Maintenance”

binlog_format

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 11.7, “Data Type Default Values”
Section 12.7, “Date and Time Functions”
Section 17.2.1.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 17.2.5.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.5.2, “Evaluation of Table-Level Replication Options”
Section 17.1.3.2, “GTID Life Cycle”
Section 12.15, “Information Functions”
Section 13.2.7, “LOAD DATA Syntax”
Section 12.14, “Locking Functions”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 12.6.2, “Mathematical Functions”
Section 12.24, “Miscellaneous Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section A.14, “MySQL 8.0 FAQ: Replication”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 23.3.5, “Obtaining Information About Partitions”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.5.1.2, “Replication and BLACKHOLE Tables”
Section 17.5.1.19, “Replication and LOAD DATA”
Section 17.5.1.21, “Replication and MEMORY Tables”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.5.1.34, “Replication and Transactions”
Section 17.2.1, “Replication Formats”
Section 17.5.1.22, “Replication of the mysql System Schema”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 13.3.8.3, “Restrictions on XA Transactions”
Section 17.3.2.1, “Scope of Binary Log Encryption”
Section B.3.1, “Server Error Message Reference”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 24.7, “Stored Program Binary Logging”
Section 5.1.9.1, “System Variable Privileges”
Section 16.6, “The BLACKHOLE Storage Engine”
Section 5.4.3, “The General Query Log”
Section 15.19.7, “The InnoDB memcached Plugin and Replication”
Section 5.4.5, “The Slow Query Log”
Section 15.7.2.1, “Transaction Isolation Levels”

Section 17.5.3, “Upgrading a Replication Setup”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

binlog_group_commit_sync_delay

Section 17.1.6.4, “Binary Logging Options and Variables”

binlog_group_commit_sync_no_delay_count

Section 17.1.6.4, “Binary Logging Options and Variables”

binlog_gtid_simple_recovery

Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.2, “GTID Life Cycle”

binlog_max_flush_queue_time

Section 17.1.6.4, “Binary Logging Options and Variables”

binlog_order_commits

Section 17.1.6.4, “Binary Logging Options and Variables”

binlog_rotate_encryption_master_key_at_startup

Section 17.3.2.3, “Binary Log Master Key Rotation”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 6.2.2, “Privileges Provided by MySQL”

binlog_row_event_max_size

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 5.4.4.2, “Setting The Binary Log Format”

binlog_row_image

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 8.5.8, “Optimizing InnoDB Disk I/O”

binlog_row_metadata

Section 17.1.6.4, “Binary Logging Options and Variables”

binlog_row_value_options

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.5.1.17, “Replication of JSON Documents”
Section 11.6, “The JSON Data Type”
Section 1.4, “What Is New in MySQL 8.0”

binlog_rows_query_log_events

Section 17.2.1.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.4.3, “Monitoring Row-based Replication”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

binlog_stmt_cache_size

Section 17.1.6.4, “Binary Logging Options and Variables”

Section 5.1.10, “Server Status Variables”

binlog_transaction_dependency_history_size

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 8.12.3.1, “How MySQL Uses Memory”

binlog_transaction_dependency_tracking

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.6.3, “Replication Slave Options and Variables”

block_encryption_mode

Section 12.13, “Encryption and Compression Functions”
Section 5.1.8, “Server System Variables”

bulk_insert_buffer_size

Section 16.2.1, “MyISAM Startup Options”
Section 8.2.5.1, “Optimizing INSERT Statements”
Section 5.1.8, “Server System Variables”

C

[index top]

caching_sha

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

character_set_client

Section 28.7.9.1, “C API Prepared Statement Type Codes”
Section 10.15, “Character Set Configuration”
Section 10.4, “Connection Character Sets and Collations”
Section 13.2.7, “LOAD DATA Syntax”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 28.7.7.69, “mysql_session_track_get_first()”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”
Section 13.7.6.2, “SET CHARACTER SET Syntax”
Section 13.7.6.3, “SET NAMES Syntax”
Section 13.7.7.7, “SHOW CREATE EVENT Syntax”
Section 13.7.7.9, “SHOW CREATE PROCEDURE Syntax”
Section 13.7.7.11, “SHOW CREATE TRIGGER Syntax”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 13.7.7.18, “SHOW EVENTS Syntax”
Section 13.7.7.28, “SHOW PROCEDURE STATUS Syntax”
Section 13.7.7.38, “SHOW TRIGGERS Syntax”
Section 5.4.4, “The Binary Log”
Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”
Section 25.23, “The INFORMATION_SCHEMA ROUTINES Table”
Section 25.34, “The INFORMATION_SCHEMA TRIGGERS Table”

Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”
Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”

character_set_connection

Section 12.10, “Cast Functions and Operators”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 10.3.8, “Character Set Introducers”
Section 10.2.1, “Character Set Repertoire”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 10.8.4, “Collation Coercibility in Expressions”
Section 10.4, “Connection Character Sets and Collations”
Section 12.7, “Date and Time Functions”
Section 12.13, “Encryption and Compression Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 10.16, “MySQL Server Locale Support”
Section 28.7.7.69, “mysql_session_track_get_first()”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”
Section 13.7.6.2, “SET CHARACTER SET Syntax”
Section 13.7.6.3, “SET NAMES Syntax”
Section 9.1.1, “String Literals”
Section 12.2, “Type Conversion in Expression Evaluation”

character_set_database

Section 13.1.9, “ALTER TABLE Syntax”
Section 2.11.4, “Changes in MySQL 8.0”
Section 10.4, “Connection Character Sets and Collations”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 10.3.3, “Database Character Set and Collation”
Section 13.2.7, “LOAD DATA Syntax”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”
Section 13.7.6.2, “SET CHARACTER SET Syntax”
Section 22.1.4, “What is New in NDB Cluster”

character_set_filesystem

Section 13.2.7, “LOAD DATA Syntax”
Section 13.2.10.1, “SELECT ... INTO Syntax”
Section 5.1.8, “Server System Variables”
Section 12.5, “String Functions and Operators”

character_set_results

Section 28.7.5, “C API Data Structures”
Section 10.4, “Connection Character Sets and Collations”
Section 10.6, “Error Message Character Set”
Section A.11, “MySQL 8.0 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 28.7.7.69, “mysql_session_track_get_first()”
Section 5.1.8, “Server System Variables”
Section 13.7.6.2, “SET CHARACTER SET Syntax”
Section 13.7.6.3, “SET NAMES Syntax”
Section 10.2.2, “UTF-8 for Metadata”

character_set_server

Section 2.11.4, “Changes in MySQL 8.0”
Section 10.15, “Character Set Configuration”
Section 10.4, “Connection Character Sets and Collations”
Section 10.3.3, “Database Character Set and Collation”
Section 12.9.4, “Full-Text Stopwords”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.3, “Replication and Character Sets”
Section 17.5.1.38, “Replication and Variables”
Section 10.3.2, “Server Character Set and Collation”
Section 5.1.8, “Server System Variables”

character_set_system

Section 10.15, “Character Set Configuration”
Section 5.1.8, “Server System Variables”
Section 10.2.2, “UTF-8 for Metadata”

character_sets_dir

Section 10.14.3, “Adding a Simple Collation to an 8-Bit Character Set”
Section 10.14.4.1, “Defining a UCA Collation Using LDML Syntax”
Section 5.1.8, “Server System Variables”

check_proxy_users

Section 6.2.18, “Proxy Users”
Section 5.1.8, “Server System Variables”

clone_autotune_concurrency

Section 5.6.7.12, “Clone System Variables”
Section 5.6.7.9, “Monitoring Cloning Operations”

clone_ddl_timeout

Section 5.6.7.8, “Remote Cloning Operation Failure Handling”

clone_enable_compression

Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.8, “Group Replication System Variables”
Prerequisites for Cloning

clone_max_concurrency

Section 5.6.7.12, “Clone System Variables”
Section 5.6.7.9, “Monitoring Cloning Operations”

clone_max_data_bandwidth

Section 5.6.7.12, “Clone System Variables”
Section 26.12.16.2, “The clone_progress Table”

clone_ssl_ca

Section 5.6.7.3, “Cloning Remote Data”
Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”

Prerequisites for Cloning

clone_ssl_cert

Section 5.6.7.3, “Cloning Remote Data”
Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”
Prerequisites for Cloning

clone_ssl_key

Section 5.6.7.3, “Cloning Remote Data”
Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.8, “Group Replication System Variables”
Prerequisites for Cloning

clone_valid_donor_list

Section 5.6.7.3, “Cloning Remote Data”
Prerequisites for Cloning

collation_connection

Section 12.10, “Cast Functions and Operators”
Section 12.5.3, “Character Set and Collation of Function Results”
Section 10.3.8, “Character Set Introducers”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 10.8.4, “Collation Coercibility in Expressions”
Section 10.4, “Connection Character Sets and Collations”
Section 12.7, “Date and Time Functions”
Section 12.13, “Encryption and Compression Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”
Section 13.7.6.3, “SET NAMES Syntax”
Section 13.7.7.7, “SHOW CREATE EVENT Syntax”
Section 13.7.7.9, “SHOW CREATE PROCEDURE Syntax”
Section 13.7.7.11, “SHOW CREATE TRIGGER Syntax”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 13.7.7.18, “SHOW EVENTS Syntax”
Section 13.7.7.28, “SHOW PROCEDURE STATUS Syntax”
Section 13.7.7.38, “SHOW TRIGGERS Syntax”
Section 5.4.4, “The Binary Log”
Section 25.10, “The INFORMATION_SCHEMA EVENTS Table”
Section 25.23, “The INFORMATION_SCHEMA ROUTINES Table”
Section 25.34, “The INFORMATION_SCHEMA TRIGGERS Table”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”
Section 12.2, “Type Conversion in Expression Evaluation”

collation_database

Section 2.11.4, “Changes in MySQL 8.0”
Section 10.4, “Connection Character Sets and Collations”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 10.3.3, “Database Character Set and Collation”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.38, “Replication and Variables”

Section 5.1.8, “Server System Variables”
Section 5.4.4, “The Binary Log”
Section 22.1.4, “What is New in NDB Cluster”

collation_server

Section 2.11.4, “Changes in MySQL 8.0”
Section 10.4, “Connection Character Sets and Collations”
Section 10.3.3, “Database Character Set and Collation”
Section 12.9.4, “Full-Text Stopwords”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.38, “Replication and Variables”
Section 10.3.2, “Server Character Set and Collation”
Section 5.1.8, “Server System Variables”
Section 5.4.4, “The Binary Log”

completion_type

Section 28.7.7.6, “mysql_commit()”
Section 28.7.7.63, “mysql_rollback()”
Section 5.1.8, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”

concurrent_insert

Section 8.11.3, “Concurrent Inserts”
Section 8.11.1, “Internal Locking Methods”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 5.1.8, “Server System Variables”

connect_timeout

Section B.4.2.10, “Communication Errors and Aborted Connections”
Section B.4.2.3, “Lost connection to MySQL server”
Section 28.7.7.54, “mysql_real_connect()”
Section 5.1.8, “Server System Variables”
Section 20.5.5.2, “X Plugin Options and System Variables”

connection_control_failed_connections_threshold

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 6.4.2.2, “Connection-Control System and Status Variables”
Section 25.41.1, “The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table”

connection_control_max_connection_delay

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 6.4.2.2, “Connection-Control System and Status Variables”

connection_control_min_connection_delay

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 6.4.2.2, “Connection-Control System and Status Variables”

core_file

Section 15.8.3.7, “Excluding Buffer Pool Pages from Core Files”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.8, “Server System Variables”

Section 1.4, “What Is New in MySQL 8.0”

create_admin_listener_thread

Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 5.1.8, “Server System Variables”

cte_max_recursion_depth

Section 5.1.8, “Server System Variables”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”

D

[index top]

daemon_memcached_engine_lib_name

Section 15.19.3, “Setting Up the InnoDB memcached Plugin”

daemon_memcached_engine_lib_path

Section 15.19.3, “Setting Up the InnoDB memcached Plugin”

daemon_memcached_option

Section 15.19.2, “InnoDB memcached Architecture”
Section 15.19.5, “Security Considerations for the InnoDB memcached Plugin”
Section 15.19.3, “Setting Up the InnoDB memcached Plugin”
Section 15.19.9, “Troubleshooting the InnoDB memcached Plugin”

daemon_memcached_r_batch_size

Section 15.19.2, “InnoDB memcached Architecture”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.19.6.6, “Performing DML and DDL Statements on the Underlying InnoDB Table”
Section 15.19.3, “Setting Up the InnoDB memcached Plugin”
Section 15.19.7, “The InnoDB memcached Plugin and Replication”
Section 15.19.6.3, “Tuning InnoDB memcached Plugin Performance”

daemon_memcached_w_batch_size

Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 15.19.2, “InnoDB memcached Architecture”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.19.6.6, “Performing DML and DDL Statements on the Underlying InnoDB Table”
Section 15.19.3, “Setting Up the InnoDB memcached Plugin”
Section 15.19.7, “The InnoDB memcached Plugin and Replication”
Section 15.19.6.3, “Tuning InnoDB memcached Plugin Performance”

datadir

Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 15.17.2, “InnoDB Recovery”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 2.4.3, “Installing and Using the MySQL Launch Daemon”
Section 2.3, “Installing MySQL on Microsoft Windows”
Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”
MySQL Glossary

Section 15.6.5, “Redo Log”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”
Section 15.6.3.4, “Undo Tablespaces”

date_format

Section 5.1.8, “Server System Variables”

datetime_format

Section 5.1.8, “Server System Variables”

debug

Section 5.1.8, “Server System Variables”
Section 29.5.4, “The DBUG Package”

debug_sync

Section 26.12.13.2, “Performance Schema variables_info Table”
Section 5.1.8, “Server System Variables”

default

Section 18.9.1, “Group Replication Requirements”
Section 16.1, “Setting the Storage Engine”
Section 15.1.4, “Testing and Benchmarking with InnoDB”

default_authentication_plugin

Section 13.7.1.1, “ALTER USER Syntax”
Section 6.4.1, “Authentication Plugins”
Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 2.11.4, “Changes in MySQL 8.0”
Section 13.7.1.3, “CREATE USER Syntax”
Section 6.2.17, “Pluggable Authentication”
Section 5.1.8, “Server System Variables”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”

default_collation_for_utf

Section 5.1.8, “Server System Variables”

default_password_lifetime

Section 13.7.1.1, “ALTER USER Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 6.2.3, “Grant Tables”
Section 6.2.15, “Password Management”
Section 5.1.8, “Server System Variables”

default_storage_engine

Section 13.1.16, “CREATE LOGFILE GROUP Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 13.1.33, “DROP TABLESPACE Syntax”
Section 15.6.3.3, “General Tablespaces”
Section 5.6.1, “Installing and Uninstalling Plugins”

Section 23.2.2, “LIST Partitioning”
Section 23.1, “Overview of Partitioning in MySQL”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”
Section 16.1, “Setting the Storage Engine”
Section 17.4.4, “Using Replication with Different Master and Slave Storage Engines”

default_table_encryption

Section 13.1.2, “ALTER DATABASE Syntax”
Section 13.1.10, “ALTER TABLESPACE Syntax”
Section 13.1.12, “CREATE DATABASE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 18.8, “Group Replication System Variables”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 5.1.8, “Server System Variables”
Section 5.1.9.1, “System Variable Privileges”
Section 1.4, “What Is New in MySQL 8.0”

default_tmp_storage_engine

Section 5.6.1, “Installing and Uninstalling Plugins”
Section 5.1.8, “Server System Variables”
Section 16.1, “Setting the Storage Engine”

default_week_format

Section 12.7, “Date and Time Functions”
Section 23.6.3, “Partitioning Limitations Relating to Functions”
Section 5.1.8, “Server System Variables”

delay_key_write

Section 13.1.20, “CREATE TABLE Syntax”
Section 8.11.5, “External Locking”
Section A.14, “MySQL 8.0 FAQ: Replication”
Section 5.1.8, “Server System Variables”
Section B.4.3.3, “What to Do If MySQL Keeps Crashing”

delayed_insert_limit

Section 5.1.8, “Server System Variables”

delayed_insert_timeout

Section 5.1.8, “Server System Variables”

delayed_queue_size

Section 5.1.8, “Server System Variables”

disabled_storage_engines

Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 18.9.1, “Group Replication Requirements”
Section A.2, “MySQL 8.0 FAQ: Storage Engines”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 5.1.7, “Server Command Options”
Section B.3.1, “Server Error Message Reference”

Section 5.1.8, “Server System Variables”

disconnect_on_expired_password

Section 6.2.16, “Server Handling of Expired Passwords”
Section 5.1.8, “Server System Variables”

div_precision_increment

Section 12.6.1, “Arithmetic Operators”
Section 5.1.8, “Server System Variables”

dragnet

Section 5.5.3, “Error Log Components”
Section 5.5, “MySQL Server Components”
Section 5.4.2.5, “Rule-Based Error Log Filtering (log_filter_dragnet)”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 5.4.2.3, “Types of Error Log Filtering”
Section 5.1.9, “Using System Variables”

E

[index top]

end_markers_in_json

Section 5.1.8, “Server System Variables”

enforce_gtid_consistency

Section 17.1.5.3, “Disabling GTID Transactions Online”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.5.1.30, “Replication and Temporary Tables”
Section 17.1.5.1, “Replication Mode Concepts”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”

eq_range_index_dive_limit

Section 8.2.1.2, “Range Optimization”
Section 5.1.8, “Server System Variables”

error_count

Section B.2, “Error Information Interfaces”
Section 13.5, “Prepared SQL Statement Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.7.17, “SHOW ERRORS Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”

event

Section 15.8.2, “Configuring InnoDB for Read-Only Operation”

event_scheduler

Section 24.4.2, “Event Scheduler Configuration”
Section 5.1.8, “Server System Variables”

Section 24.4.6, “The Event Scheduler and MySQL Privileges”

expire_logs_days

Section 17.1.6.4, “Binary Logging Options and Variables”

explicit_defaults_for_timestamp

Section 11.3.4, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 11.7, “Data Type Default Values”
Section 11.1.2, “Date and Time Type Overview”
Section 5.1.8, “Server System Variables”

external_user

Section 6.4.5.4, “Audit Log File Formats”
Implementing Proxy User Support in Authentication Plugins
Section 6.2.18, “Proxy Users”
Section 5.1.8, “Server System Variables”
Writing the Server-Side Authentication Plugin

F

[index top]

flush

Section 5.1.8, “Server System Variables”

flush_time

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

foreign_key_checks

Section 13.1.9, “ALTER TABLE Syntax”
Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 5.4.4.3, “Mixed Binary Logging Format”
NDB Cluster System Variables
Section 15.12.1, “Online DDL Operations”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 5.4.4, “The Binary Log”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

ft_boolean_syntax

Section 12.9.2, “Boolean Full-Text Searches”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 5.1.8, “Server System Variables”

ft_max_word_len

Section 12.9.2, “Boolean Full-Text Searches”
Creating a Data Snapshot Using Raw Data Files
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.9.8, “ngram Full-Text Parser”
Section 5.1.8, “Server System Variables”

Section 29.2.4.4, “Writing Full-Text Parser Plugins”

ft_min_word_len

Section 12.9.2, “Boolean Full-Text Searches”
Creating a Data Snapshot Using Raw Data Files
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.9.9, “MeCab Full-Text Parser Plugin”
Section 12.9.1, “Natural Language Full-Text Searches”
Section 12.9.8, “ngram Full-Text Parser”
Section 5.1.8, “Server System Variables”
Section 29.2.4.4, “Writing Full-Text Parser Plugins”

ft_query_expansion_limit

Section 5.1.8, “Server System Variables”

ft_stopword_file

Section 12.9.2, “Boolean Full-Text Searches”
Creating a Data Snapshot Using Raw Data Files
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.9.4, “Full-Text Stopwords”
Section 12.9.1, “Natural Language Full-Text Searches”
Section 5.1.8, “Server System Variables”

G

[index top]

general_log

MySQL Glossary
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 5.4.3, “The General Query Log”

general_log_file

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 5.4.3, “The General Query Log”
Section 1.4, “What Is New in MySQL 8.0”

generated_random_password_length

Section 6.2.15, “Password Management”
Section 5.1.8, “Server System Variables”

group_concat_max_len

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section 5.1.8, “Server System Variables”

group_replication_allow_local_disjoint_gtids_join

Section 18.8, “Group Replication System Variables”

group_replication_allow_local_lower_version_join

Section 18.8, “Group Replication System Variables”

Section 18.7.1.1, “Member Versions During Upgrades”

group_replication_auto_increment_increment

Section 18.10, “Frequently Asked Questions”
Section 18.8, “Group Replication System Variables”
Section 17.1.6.2, “Replication Master Options and Variables”

group_replication_autorejoin_tries

Section 18.6.6.3, “Auto-Rejoin”
Section 18.6.6.4, “Exit Action”
Section 18.6.6.1, “Expel Timeout”
Section 18.10, “Frequently Asked Questions”
Section 18.8, “Group Replication System Variables”
Section 18.6.6.2, “Unreachable Majority Timeout”

group_replication_bootstrap_group

Adding a Second Instance
Section 18.2.1.5, “Bootstrapping the Group”
Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 18.8, “Group Replication System Variables”
Section 18.1.3, “Multi-Primary and Single-Primary Modes”

group_replication_clone_threshold

Section 18.4.3.1, “Cloning for Distributed Recovery”
Section 18.8, “Group Replication System Variables”
Section 18.4.3.4, “How Distributed Recovery Works”
Threshold for Cloning
Section 21.2.5.1, “Working with a Cluster that uses MySQL Clone”

group_replication_communication_debug_options

Section 18.8, “Group Replication System Variables”

group_replication_communication_max_message_size

Section 18.9.2, “Group Replication Limitations”
Section 18.8, “Group Replication System Variables”
Section 18.6.4, “Message Fragmentation”

group_replication_components_stop_timeout

Section 18.8, “Group Replication System Variables”

group_replication_compression_threshold

Section 18.9.2, “Group Replication Limitations”
Section 18.8, “Group Replication System Variables”

group_replication_consistency

Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”
Section 18.8, “Group Replication System Variables”
Section 18.1.3.2, “Multi-Primary Mode”
Section 6.2.2, “Privileges Provided by MySQL”
Section 18.1.3.1, “Single-Primary Mode”
Section 18.4.2.1, “Understanding Transaction Consistency Guarantees”

Section 21.4, “Working with InnoDB Cluster”

group_replication_enforce_update_everywhere_checks

Section 18.4.1.2, “Changing a Group's Mode”
Section 13.4.3.4, “Functions which Configure the Group Replication Mode”
Section 18.9.2, “Group Replication Limitations”
Section 18.8, “Group Replication System Variables”
Section 18.1.3.1, “Single-Primary Mode”
Transaction Checks

group_replication_exit_state_action

Section 18.6.6.3, “Auto-Rejoin”
Section 18.6.6.4, “Exit Action”
Section 18.6.6.1, “Expel Timeout”
Section 18.4.3.3, “Fault Tolerance for Distributed Recovery”
Section 18.3.1, “Group Replication Server States”
Section 18.8, “Group Replication System Variables”
Section 18.6.6.2, “Unreachable Majority Timeout”
Section 21.4, “Working with InnoDB Cluster”

group_replication_flow_control_applier_threshold

Section 18.8, “Group Replication System Variables”

group_replication_flow_control_certifier_threshold

Section 18.8, “Group Replication System Variables”

group_replication_flow_control_hold_percent

Section 18.8, “Group Replication System Variables”

group_replication_flow_control_max_commit_quota

Section 18.8, “Group Replication System Variables”

group_replication_flow_control_member_quota_percent

Section 18.8, “Group Replication System Variables”

group_replication_flow_control_min_quota

Section 18.8, “Group Replication System Variables”

group_replication_flow_control_min_recovery_quota

Section 18.8, “Group Replication System Variables”

group_replication_flow_control_mode

Section 18.8, “Group Replication System Variables”

group_replication_flow_control_period

Section 18.8, “Group Replication System Variables”
Section 18.3.3, “The replication_group_member_stats Table”

group_replication_flow_control_release_percent

Section 18.8, “Group Replication System Variables”

group_replication_force_members

Section 18.8, “Group Replication System Variables”
Section 18.4.4, “Network Partitioning”

group_replication_group_name

Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 18.8, “Group Replication System Variables”
Section 21.4, “Working with InnoDB Cluster”

group_replication_group_seeds

Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 18.2.2, “Deploying Group Replication Locally”
Section 18.5.1, “Group Replication IP Address Whitelisting”
Section 18.8, “Group Replication System Variables”
Section 18.4.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”
Section 21.2.5.1, “Working with a Cluster that uses MySQL Clone”
Section 21.4, “Working with InnoDB Cluster”

group_replication_gtid_assignment_block_size

Section 18.8, “Group Replication System Variables”

group_replication_ip_whitelist

Section 18.5.1, “Group Replication IP Address Whitelisting”
Section 18.8, “Group Replication System Variables”
Section 18.4.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”
Section 21.4, “Working with InnoDB Cluster”

group_replication_local_address

Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 18.2.2, “Deploying Group Replication Locally”
Section 18.10, “Frequently Asked Questions”
Section 18.5.1, “Group Replication IP Address Whitelisting”
Section 18.8, “Group Replication System Variables”
Section 18.4.5, “Support For IPv6 And For Mixed IPv6 And IPv4 Groups”
Section 26.12.11.9, “The replication_group_members Table”
Section 21.4, “Working with InnoDB Cluster”

group_replication_member_expel_timeout

Section 18.6.6.4, “Exit Action”
Section 18.6.6.1, “Expel Timeout”
Section 18.10, “Frequently Asked Questions”
Section 18.1.4.1, “Group Membership”
Section 18.9.2, “Group Replication Limitations”
Section 18.8, “Group Replication System Variables”
Section 18.6.4, “Message Fragmentation”
Section 21.2.4, “Production Deployment of InnoDB Cluster”
Section 18.6.6, “Responses to Failure Detection and Network Partitioning”
Section 18.6.5, “XCom Cache Management”

group_replication_member_weight

Section 18.8, “Group Replication System Variables”

Primary Election Algorithm
Section 21.4, “Working with InnoDB Cluster”

group_replication_message_cache_size

Section 18.8, “Group Replication System Variables”
Section 18.6.5, “XCom Cache Management”

group_replication_poll_spin_loops

Section 18.6.1, “Fine Tuning the Group Communication Thread”
Section 18.8, “Group Replication System Variables”

group_replication_recovery_complete_at

Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.8, “Group Replication System Variables”

group_replication_recovery_compression_algorithm

Section 18.4.3.2, “Configuring Distributed Recovery”
Section 4.2.6, “Connection Compression Control”
Section 18.8, “Group Replication System Variables”

group_replication_recovery_get_public_key

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.8, “Group Replication System Variables”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”
Section 18.2.1.3, “User Credentials”

group_replication_recovery_public_key_path

Section 6.4.1.2, “Caching SHA-2 Pluggable Authentication”
Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.8, “Group Replication System Variables”
Section 18.2.1.3, “User Credentials”

group_replication_recovery_reconnect_interval

Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.8, “Group Replication System Variables”

group_replication_recovery_retry_count

Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.4.3.3, “Fault Tolerance for Distributed Recovery”
Section 18.8, “Group Replication System Variables”

group_replication_recovery_ssl_ca

Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”
Prerequisites for Cloning

group_replication_recovery_ssl_capath

Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”

Section 18.8, “Group Replication System Variables”

group_replication_recovery_ssl_cert

Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”
Prerequisites for Cloning

group_replication_recovery_ssl_cipher

Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”

group_replication_recovery_ssl_crl

Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”

group_replication_recovery_ssl_crlpath

Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”

group_replication_recovery_ssl_key

Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”
Prerequisites for Cloning

group_replication_recovery_ssl_verify_server_cert

Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”

group_replication_recovery_use_ssl

Section 18.4.3.2, “Configuring Distributed Recovery”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”
Prerequisites for Cloning

group_replication_recovery_zstd_compression_level

Section 18.4.3.2, “Configuring Distributed Recovery”
Section 4.2.6, “Connection Compression Control”
Section 18.8, “Group Replication System Variables”

group_replication_single_primary_mode

Section 18.9.2, “Group Replication Limitations”
Section 18.8, “Group Replication System Variables”
Section 18.1.3, “Multi-Primary and Single-Primary Modes”
Section 18.1.3.2, “Multi-Primary Mode”
Section 18.1.3.1, “Single-Primary Mode”
Section 21.3, “Using MySQL Router with InnoDB Cluster”

group_replication_ssl_mode

Section 18.4.3.2, “Configuring Distributed Recovery”

Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 18.8, “Group Replication System Variables”
Section 21.4, “Working with InnoDB Cluster”

group_replication_start_on_boot

Cloning Operations
Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 18.6.6.4, “Exit Action”
Section 18.8, “Group Replication System Variables”
Section 18.7.3.2, “Upgrading a Group Replication Member”

group_replication_transaction_size_limit

Section 18.9.2, “Group Replication Limitations”
Section 18.8, “Group Replication System Variables”
Section 18.6.4, “Message Fragmentation”

group_replication_unreachable_majority_timeout

Section 18.6.6.4, “Exit Action”
Section 18.8, “Group Replication System Variables”
Section 18.6.6.2, “Unreachable Majority Timeout”

gtid_executed

Section 2.11.14, “Copying MySQL Databases to Another Machine”
Creating a Data Snapshot Using mysqldump
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.3, “GTID Auto-Positioning”
Section 17.1.3.1, “GTID Format and Storage”
Section 17.1.3.2, “GTID Life Cycle”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 26.12.11, “Performance Schema Replication Tables”
Section 17.1.5.1, “Replication Mode Concepts”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 13.7.7.23, “SHOW MASTER STATUS Syntax”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”
Section 26.12.18.3, “The log_status Table”
Threshold for Cloning
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”
Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”
Section 21.4, “Working with InnoDB Cluster”

gtid_executed_compression_period

Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Format and Storage”

GTID_MODE

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.1.3.3, “GTID Auto-Positioning”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

gtid_mode

Adding a GTID Based Master to a Multi-Source Replication Slave
Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 2.11.14, “Copying MySQL Databases to Another Machine”
Creating a Data Snapshot Using mysqldump
Section 17.1.5.3, “Disabling GTID Transactions Online”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 17.1.5.2, “Enabling GTID Transactions Online”
Section 12.18, “Functions Used with Global Transaction Identifiers (GTIDs)”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 18.9.1, “Group Replication Requirements”
Section 17.1.3.1, “GTID Format and Storage”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.5.1, “Replication Mode Concepts”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Section 26.12.7.1, “The events_transactions_current Table”
Section 17.5.3, “Upgrading a Replication Setup”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”

gtid_next

Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Format and Storage”
Section 17.1.3.2, “GTID Life Cycle”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 17.1.5.1, “Replication Mode Concepts”
Section 5.1.10, “Server Status Variables”
Section 13.4.2.6, “START SLAVE Syntax”
Section 13.4.2.7, “STOP SLAVE Syntax”
Section 26.12.7.1, “The events_transactions_current Table”

gtid_owned

Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.2, “GTID Life Cycle”

gtid_purged

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Creating a Data Snapshot Using mysqldump
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.1.3.3, “GTID Auto-Positioning”
Section 17.1.3.1, “GTID Format and Storage”
Section 17.1.3.2, “GTID Life Cycle”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.5.1, “Replication Mode Concepts”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”
Threshold for Cloning

Section 17.1.3.5, “Using GTIDs for Failover and Scaleout”

H

[index top]

have_compress
Section 5.1.8, “Server System Variables”

have_crypt
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

have_dynamic_loading
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.8, “Server System Variables”

have_geometry
Section 5.1.8, “Server System Variables”

have_openssl
Section 5.1.8, “Server System Variables”

have_profiling
Section 5.1.8, “Server System Variables”

have_query_cache
Section 5.1.8, “Server System Variables”

have_rtree_keys
Section 5.1.8, “Server System Variables”

have_ssl
Section 2.9.6, “Configuring SSL Library Support”
Section 5.1.8, “Server System Variables”

have_statement_timeout
Section 5.1.8, “Server System Variables”

have_symlink
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 8.12.2.2, “Using Symbolic Links for MyISAM Tables on Unix”

histogram_generation_max_mem_size
Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 5.1.8, “Server System Variables”

host_cache_size
Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 26.12.18.1, “The host_cache Table”

hostname

Section 13.7.5, “CLONE Syntax”
Section 5.6.7.3, “Cloning Remote Data”
Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 18.10, “Frequently Asked Questions”
Section 5.1.8, “Server System Variables”
Section 26.12.11.9, “The replication_group_members Table”

I

[index top]

identity

Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”

ignore_builtin_innodb

Section 1.4, “What Is New in MySQL 8.0”

immediate_server_version

Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 17.5.2, “Replication Compatibility Between MySQL Versions”
Section 17.1.6.2, “Replication Master Options and Variables”

information_schema_stats_expiry

Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 14.7, “Data Dictionary Usage Differences”
Section 14.5, “INFORMATION_SCHEMA and Data Dictionary Integration”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 5.1.8, “Server System Variables”
Section 25.26, “The INFORMATION_SCHEMA STATISTICS Table”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”
Section 1.4, “What Is New in MySQL 8.0”

init_connect

Section 10.5, “Configuring Application Character Set and Collation”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.1.8, “Server System Variables”
Section 26.12.18.1, “The host_cache Table”

init_file

Section 2.10.1, “Initializing the Data Directory”
Section 8.10.2.2, “Multiple Key Caches”
Section 26.4, “Performance Schema Runtime Configuration”
Section 26.12.13.2, “Performance Schema variables_info Table”

Resetting the Root Password: Unix and Unix-Like Systems
Resetting the Root Password: Windows Systems
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 16.3, “The MEMORY Storage Engine”

init_slave

Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 17.1.6.3, “Replication Slave Options and Variables”

innodb

Section 2.11.4, “Changes in MySQL 8.0”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 15.8.3.7, “Excluding Buffer Pool Pages from Core Files”
Section A.16, “MySQL 8.0 FAQ: InnoDB Change Buffer”

innodb_adaptive_flushing

Section 15.8.3.5, “Configuring Buffer Pool Flushing”

innodb_adaptive_flushing_lwm

Section 15.8.3.5, “Configuring Buffer Pool Flushing”

innodb_adaptive_hash_index

Section 15.5.3, “Adaptive Hash Index”
Section 15.8.4, “Configuring Thread Concurrency for InnoDB”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.9, “Optimizing InnoDB Configuration Variables”
Section 13.1.37, “TRUNCATE TABLE Syntax”

innodb_adaptive_hash_index_parts

Section 15.5.3, “Adaptive Hash Index”
Section 25.39.29, “The INFORMATION_SCHEMA INNODB_TRX Table”

innodb_adaptive_max_sleep_delay

Section 15.8.4, “Configuring Thread Concurrency for InnoDB”
Section 15.13, “InnoDB Startup Options and System Variables”

innodb_api_bk_commit_interval

Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 15.19.2, “InnoDB memcached Architecture”

innodb_api_disable_rowlock

Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”

innodb_api_enable_binlog

Section 15.19.7, “The InnoDB memcached Plugin and Replication”

innodb_api_enable_mdl

Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 15.19.2, “InnoDB memcached Architecture”

innodb_api_trx_level

Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 15.19.2, “InnoDB memcached Architecture”

innodb_autoextend_increment

Section 15.6.3.2, “File-Per-Table Tablespaces”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.6.3.1, “The System Tablespace”

innodb_autoinc_lock_mode

Section 15.6.1.4, “AUTO_INCREMENT Handling in InnoDB”
Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 12.15, “Information Functions”
Section 15.7.1, “InnoDB Locking”
Section 15.6.1.6, “Limits on InnoDB Tables”
MySQL Glossary
Section 1.4, “What Is New in MySQL 8.0”

innodb_buffer_pool_chunk_size

Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”
Section 15.13, “InnoDB Startup Options and System Variables”

innodb_buffer_pool_dump_at_shutdown

Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”

innodb_buffer_pool_dump_now

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”

innodb_buffer_pool_dump_pct

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”

innodb_buffer_pool_filename

Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”

innodb_buffer_pool_in_core_file

Section 15.8.3.7, “Excluding Buffer Pool Pages from Core Files”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 5.1.7, “Server Command Options”

innodb_buffer_pool_instances

Section 15.8.3.5, “Configuring Buffer Pool Flushing”
Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”
Section 15.8.3.2, “Configuring Multiple Buffer Pool Instances”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”

MySQL Glossary
Section 15.19.6.3, “Tuning InnoDB memcached Plugin Performance”

innodb_buffer_pool_load_abort

Section 5.1.10, “Server Status Variables”

innodb_buffer_pool_load_at_startup

Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”
Section 5.1.10, “Server Status Variables”

innodb_buffer_pool_load_now

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”
Section 5.1.10, “Server Status Variables”

innodb_buffer_pool_size

Section 15.9.1.6, “Compression for OLTP Workloads”
Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”
Section 15.8.3.2, “Configuring Multiple Buffer Pool Instances”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 15.19.2, “InnoDB memcached Architecture”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section B.3.1, “Server Error Message Reference”
Section 5.1.10, “Server Status Variables”
Section 15.19.6.3, “Tuning InnoDB memcached Plugin Performance”
Section 1.4, “What Is New in MySQL 8.0”

innodb_change_buffer_max_size

Section 15.5.2, “Change Buffer”
Section A.16, “MySQL 8.0 FAQ: InnoDB Change Buffer”
MySQL Glossary

innodb_change_buffering

Section 15.5.2, “Change Buffer”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
MySQL Glossary
Section 8.5.2, “Optimizing InnoDB Transaction Management”

innodb_checksum_algorithm

Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”

innodb_checksums

MySQL Glossary

innodb_cmp_per_index_enabled

Section 15.9.1.4, “Monitoring InnoDB Table Compression at Runtime”
Section 25.39.7, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”
Section 15.9.1.3, “Tuning Compression for InnoDB Tables”

innodb_compression_failure_threshold_pct

Section 15.9.1.6, “Compression for OLTP Workloads”
Section 15.9.1.5, “How Compression Works for InnoDB Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.9.1.3, “Tuning Compression for InnoDB Tables”

innodb_compression_level

Section 15.9.1.6, “Compression for OLTP Workloads”
Section 15.9.1.5, “How Compression Works for InnoDB Tables”
MySQL Glossary
Section 15.9.1.3, “Tuning Compression for InnoDB Tables”

innodb_compression_pad_pct_max

Section 15.9.1.6, “Compression for OLTP Workloads”
Section 15.9.1.5, “How Compression Works for InnoDB Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.9.1.3, “Tuning Compression for InnoDB Tables”

innodb_concurrency_tickets

Section 15.8.4, “Configuring Thread Concurrency for InnoDB”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.5.9, “Optimizing InnoDB Configuration Variables”
Section 25.39.29, “The INFORMATION_SCHEMA INNODB_TRX Table”

innodb_data_file_path

Section 5.6.7.3, “Cloning Remote Data”
Section 15.11.2, “File Space Management”
Section 2.10.1, “Initializing the Data Directory”
Section 4.6.2, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”
Section 15.6.3.1, “The System Tablespace”
Section 15.20.1, “Troubleshooting InnoDB I/O Problems”

innodb_data_home_dir

Section 5.6.7.3, “Cloning Remote Data”
Section 2.10.1, “Initializing the Data Directory”
Section 15.17.2, “InnoDB Recovery”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”
Section 15.6.5, “Redo Log”
Section 15.6.3.5, “Temporary Tablespaces”

Section 15.20.1, “Troubleshooting InnoDB I/O Problems”
Section 15.6.3.4, “Undo Tablespaces”
Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”

innodb_deadlock_detect

Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 15.7.5.2, “Deadlock Detection and Rollback”
Section 15.7.5, “Deadlocks in InnoDB”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.11.1, “Internal Locking Methods”
MySQL Glossary
Section 1.4, “What Is New in MySQL 8.0”

innodb_dedicated_server

Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 1.4, “What Is New in MySQL 8.0”

innodb_default_row_format

Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 13.1.20, “CREATE TABLE Syntax”
Section 15.6.1.1, “Creating InnoDB Tables”
Section 15.10, “InnoDB Row Formats”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.4.1, “Optimizing Data Size”

innodb_directories

Section 2.11.4, “Changes in MySQL 8.0”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”
Section 15.6.3.3, “General Tablespaces”
Section 15.17.2, “InnoDB Recovery”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”
Section 15.6.5, “Redo Log”
Section 15.6.3.4, “Undo Tablespaces”
Section 1.4, “What Is New in MySQL 8.0”

innodb_doublewrite

Section 15.6.4, “Doublewrite Buffer”
Section 15.2, “InnoDB and the ACID Model”
Section 15.11.1, “InnoDB Disk I/O”
Section 15.8.1, “InnoDB Startup Configuration”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 15.19.6.3, “Tuning InnoDB memcached Plugin Performance”

innodb_fast_shutdown

Section 15.17.2, “InnoDB Recovery”
MySQL Glossary
Section 5.1.17, “The Server Shutdown Process”
Section 2.11.6, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”

innodb_fil_make_page_dirty_debug

Section 15.13, “InnoDB Startup Options and System Variables”

innodb_file_format

Section 15.6.3.3, “General Tablespaces”

innodb_file_per

Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”
Section 15.13, “InnoDB Startup Options and System Variables”

innodb_file_per_table

Section 15.1.2, “Best Practices for InnoDB Tables”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 13.1.20.4, “CREATE TABLE ... LIKE Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.6.3.6, “Creating a Tablespace Outside of the Data Directory”
Section 15.9.1.2, “Creating Compressed Tables”
Section 15.6.1.1, “Creating InnoDB Tables”
Section 15.11.2, “File Space Management”
Section 15.6.3.2, “File-Per-Table Tablespaces”
Section 13.1.20.2, “Files Created by CREATE TABLE”
Section 13.7.8.3, “FLUSH Syntax”
Section 15.6.3.3, “General Tablespaces”
Section 15.9.1.5, “How Compression Works for InnoDB Tables”
Section 15.2, “InnoDB and the ACID Model”
Section 15.10, “InnoDB Row Formats”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.2, “Moving or Copying InnoDB Tables”
MySQL Glossary
Section 13.7.3.4, “OPTIMIZE TABLE Syntax”
Section 15.11.5, “Reclaiming Disk Space with TRUNCATE TABLE”
Section 17.4.6, “Replicating Different Databases to Different Slaves”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 15.9.1.7, “SQL Compression Syntax Warnings and Errors”
Section 15.20.3, “Troubleshooting InnoDB Data Dictionary Operations”

innodb_fill_factor

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.2.3, “Sorted Index Builds”
Section 15.6.2.2, “The Physical Structure of an InnoDB Index”

innodb_flush_log_at_timeout

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.5.4, “Log Buffer”

innodb_flush_log_at_trx_commit

Section 13.1.1, “Atomic Data Definition Statement Support”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 15.2, “InnoDB and the ACID Model”
Section 15.13, “InnoDB Startup Options and System Variables”

Section 15.5.4, “Log Buffer”
Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 17.5.1.27, “Replication and Master or Slave Shutdowns”
Section 15.19.6.3, “Tuning InnoDB memcached Plugin Performance”

innodb_flush_method

Section 15.6.4, “Doublewrite Buffer”
Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”
Section 15.6.3.2, “File-Per-Table Tablespaces”
Section 15.11.1, “InnoDB Disk I/O”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 5.1.10, “Server Status Variables”
Section 15.19.6.3, “Tuning InnoDB memcached Plugin Performance”
Section 1.4, “What Is New in MySQL 8.0”

innodb_flush_neighbors

Section 15.8.3.5, “Configuring Buffer Pool Flushing”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”

innodb_flush_sync

Section 15.8.7, “Configuring InnoDB I/O Capacity”
Section 15.13, “InnoDB Startup Options and System Variables”

innodb_flushing_avg_loops

Section 15.8.3.5, “Configuring Buffer Pool Flushing”

innodb_force_load_corrupted

Section 15.13, “InnoDB Startup Options and System Variables”

innodb_force_recovery

Section 13.1.32, “DROP TABLE Syntax”
Section 15.20.2, “Forcing InnoDB Recovery”
Section 1.7, “How to Report Bugs or Problems”
Section 15.17.2, “InnoDB Recovery”
Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 2.11.13, “Rebuilding or Repairing Tables or Indexes”

innodb_fsync_threshold

Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.5.8, “Optimizing InnoDB Disk I/O”

innodb_ft_aux_table

Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 25.39.13, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 25.39.14, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”
Section 25.39.16, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 25.39.17, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”
Section 25.39.18, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”

innodb_ft_cache_size

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 25.39.17, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”

innodb_ft_enable_diag_print

Section 15.13, “InnoDB Startup Options and System Variables”

innodb_ft_enable_stopword

Section 12.9.2, “Boolean Full-Text Searches”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 12.9.1, “Natural Language Full-Text Searches”

innodb_ft_max_token_size

Section 12.9.2, “Boolean Full-Text Searches”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.9.4, “Full-Text Stopwords”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 12.9.8, “ngram Full-Text Parser”
Section 29.2.4.4, “Writing Full-Text Parser Plugins”

innodb_ft_min_token_size

Section 12.9.2, “Boolean Full-Text Searches”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.9.4, “Full-Text Stopwords”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 12.9.9, “MeCab Full-Text Parser Plugin”
Section 12.9.1, “Natural Language Full-Text Searches”
Section 12.9.8, “ngram Full-Text Parser”
Section 29.2.4.4, “Writing Full-Text Parser Plugins”

innodb_ft_num_word_optimize

Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 13.7.3.4, “OPTIMIZE TABLE Syntax”

innodb_ft_result_cache_limit

Section 15.13, “InnoDB Startup Options and System Variables”

innodb_ft_server_stopword_table

Section 12.9.2, “Boolean Full-Text Searches”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.9.4, “Full-Text Stopwords”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 12.9.1, “Natural Language Full-Text Searches”
Section 25.39.15, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”

innodb_ft_sort_pll_degree

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”

Section 15.13, “InnoDB Startup Options and System Variables”

innodb_ft_total_cache_size

Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 25.39.17, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”

innodb_ft_user_stopword_table

Section 12.9.2, “Boolean Full-Text Searches”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.9.4, “Full-Text Stopwords”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 12.9.1, “Natural Language Full-Text Searches”
Section 25.39.15, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”

innodb_idle_flush_pct

Section 15.8.3.5, “Configuring Buffer Pool Flushing”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 1.4, “What Is New in MySQL 8.0”

innodb_io_capacity

Section 15.8.3.5, “Configuring Buffer Pool Flushing”
Section 15.8.7, “Configuring InnoDB I/O Capacity”
Section 15.13, “InnoDB Startup Options and System Variables”
Section A.16, “MySQL 8.0 FAQ: InnoDB Change Buffer”
Section 8.5.8, “Optimizing InnoDB Disk I/O”

innodb_io_capacity_max

Section 15.8.3.5, “Configuring Buffer Pool Flushing”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.5.8, “Optimizing InnoDB Disk I/O”

innodb_lock_wait_timeout

Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 15.7.5.2, “Deadlock Detection and Rollback”
Section 15.7.5, “Deadlocks in InnoDB”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.11.1, “Internal Locking Methods”
MySQL Glossary
Section 17.5.1.31, “Replication Retries and Timeouts”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section B.3.1, “Server Error Message Reference”
Section 1.4, “What Is New in MySQL 8.0”

innodb_locks_unsafe_for_binlog

MySQL Glossary
Section 15.7.2.1, “Transaction Isolation Levels”

innodb_log_buffer_size

Section 15.8.1, “InnoDB Startup Configuration”

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.5.4, “Log Buffer”
MySQL Glossary
Section 8.5.4, “Optimizing InnoDB Redo Logging”
Section 1.4, “What Is New in MySQL 8.0”

innodb_log_compressed_pages

Section 15.9.1.6, “Compression for OLTP Workloads”
Section 8.5.8, “Optimizing InnoDB Disk I/O”

innodb_log_file_size

Section 15.8.3.5, “Configuring Buffer Pool Flushing”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”
Section 2.10.1, “Initializing the Data Directory”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 8.5.4, “Optimizing InnoDB Redo Logging”
Section 15.6.5, “Redo Log”
Section 5.1.9, “Using System Variables”
Section 1.4, “What Is New in MySQL 8.0”

innodb_log_files_in_group

Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.4, “Optimizing InnoDB Redo Logging”
Section 15.6.5, “Redo Log”

innodb_log_group_home_dir

Section 5.6.7.3, “Cloning Remote Data”
Section 2.10.1, “Initializing the Data Directory”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.6.5, “Redo Log”
Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”

innodb_log_spin_cpu_abs_lwm

Section 8.5.4, “Optimizing InnoDB Redo Logging”
Section 1.4, “What Is New in MySQL 8.0”

innodb_log_spin_cpu_pct_hwm

Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.5.4, “Optimizing InnoDB Redo Logging”
Section 1.4, “What Is New in MySQL 8.0”

innodb_log_wait_for_flush_spin_hwm

Section 8.5.4, “Optimizing InnoDB Redo Logging”

Section 1.4, “What Is New in MySQL 8.0”

innodb_log_write_ahead_size

Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.5.4, “Optimizing InnoDB Redo Logging”

innodb_lru_scan_depth

Section 15.8.3.5, “Configuring Buffer Pool Flushing”

innodb_max_dirty_pages_pct

Section 15.8.3.5, “Configuring Buffer Pool Flushing”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 1.4, “What Is New in MySQL 8.0”

innodb_max_dirty_pages_pct_lwm

Section 15.8.3.5, “Configuring Buffer Pool Flushing”
Section 1.4, “What Is New in MySQL 8.0”

innodb_max_purge_lag

Section 15.3, “InnoDB Multi-Versioning”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.8.9, “Purge Configuration”

innodb_max_purge_lag_delay

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.8.9, “Purge Configuration”

innodb_max_undo_log_size

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.3.4, “Undo Tablespaces”

innodb_monitor_disable

Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 13.7.7.15, “SHOW ENGINE Syntax”
Section 25.39.22, “The INFORMATION_SCHEMA INNODB_METRICS Table”

innodb_monitor_enable

Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 13.7.7.15, “SHOW ENGINE Syntax”
Section 25.39.22, “The INFORMATION_SCHEMA INNODB_METRICS Table”

innodb_monitor_reset

Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 25.39.22, “The INFORMATION_SCHEMA INNODB_METRICS Table”

innodb_monitor_reset_all

Section 15.14.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 25.39.22, “The INFORMATION_SCHEMA INNODB_METRICS Table”

innodb_old_blocks_pct

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.8.3.3, “Making the Buffer Pool Scan Resistant”
MySQL Glossary

innodb_old_blocks_time

Section 15.5.1, “Buffer Pool”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.8.3.3, “Making the Buffer Pool Scan Resistant”

innodb_online_alter_log_max_size

MySQL Glossary
Section 15.12.5, “Online DDL Failure Conditions”
Section 15.12.3, “Online DDL Space Requirements”

innodb_open_files

Section 5.1.8, “Server System Variables”

innodb_optimize_fulltext_only

Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 15.6.2.4, “InnoDB FULLTEXT Indexes”
Section 15.14.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 13.7.3.4, “OPTIMIZE TABLE Syntax”
Section 25.39.18, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”

innodb_page_cleaners

Section 15.8.3.5, “Configuring Buffer Pool Flushing”
MySQL Glossary

innodb_page_size

Section 5.6.7.3, “Cloning Remote Data”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.9.1.2, “Creating Compressed Tables”
Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 15.8.3.7, “Excluding Buffer Pool Pages from Core Files”
Section 15.11.2, “File Space Management”
Section 15.6.3.3, “General Tablespaces”
Section 15.9.1.5, “How Compression Works for InnoDB Tables”
Section 15.9.2, “InnoDB Page Compression”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”
Section 8.4.7, “Limits on Table Column Count and Row Size”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 8.5.4, “Optimizing InnoDB Redo Logging”
Section 15.9.1.1, “Overview of Table Compression”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”
Section 15.6.2.2, “The Physical Structure of an InnoDB Index”
Section 15.6.3.1, “The System Tablespace”

Section 15.19.9, “Troubleshooting the InnoDB memcached Plugin”
Section 15.6.3.4, “Undo Tablespaces”

innodb_parallel_read_threads

Section 13.7.3.2, “CHECK TABLE Syntax”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 1.4, “What Is New in MySQL 8.0”

innodb_print_all_deadlocks

Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 15.7.5, “Deadlocks in InnoDB”
Section 15.7.5.3, “How to Minimize and Handle Deadlocks”
Section 15.20, “InnoDB Troubleshooting”

innodb_print_ddl_logs

Section 13.1.1, “Atomic Data Definition Statement Support”

innodb_purge_batch_size

Section 15.8.9, “Purge Configuration”

innodb_purge_rseg_truncate_frequency

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.8.9, “Purge Configuration”
Section 15.6.3.4, “Undo Tablespaces”

innodb_purge_threads

Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.8.9, “Purge Configuration”

innodb_random_read_ahead

Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
MySQL Glossary

innodb_read_ahead_threshold

Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
Section 15.13, “InnoDB Startup Options and System Variables”

innodb_read_io_threads

Section 15.8.5, “Configuring the Number of Background InnoDB I/O Threads”
Section 15.16.3, “InnoDB Standard Monitor and Lock Monitor Output”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 29.1.1, “MySQL Threads”
Section 15.8.6, “Using Asynchronous I/O on Linux”

innodb_read_only

Section 13.7.3.1, “ANALYZE TABLE Syntax”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 14.7, “Data Dictionary Usage Differences”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 5.1.8, “Server System Variables”

Section 25.26, “The INFORMATION_SCHEMA STATISTICS Table”
Section 25.30, “The INFORMATION_SCHEMA TABLES Table”

innodb_redo_log_archive_dirs

Section 15.6.5, “Redo Log”

innodb_redo_log_encrypt

Section 5.6.7.4, “Cloning Encrypted Data”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”

innodb_rollback_segments

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.6, “Undo Logs”
Section 15.6.3.4, “Undo Tablespaces”
Section 1.4, “What Is New in MySQL 8.0”

innodb_saved_page_number_debug

Section 15.13, “InnoDB Startup Options and System Variables”

innodb_sort_buffer_size

Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.12.3, “Online DDL Space Requirements”

innodb_spin_wait_delay

Section 15.8.8, “Configuring Spin Lock Polling”

innodb_spin_wait_pause_multiplier

Section 15.8.8, “Configuring Spin Lock Polling”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 1.4, “What Is New in MySQL 8.0”

innodb_stats_auto_recalc

Configuring Automatic Statistics Calculation for Persistent Optimizer Statistics
Section 15.8.10, “Configuring Optimizer Statistics for InnoDB”
Configuring Optimizer Statistics Parameters for Individual Tables
Section 13.1.20, “CREATE TABLE Syntax”
InnoDB Persistent Statistics Tables Example

innodb_stats_include_delete_marked

Including Delete-marked Records in Persistent Statistics Calculations
Section 15.13, “InnoDB Startup Options and System Variables”

innodb_stats_method

Section 8.3.8, “InnoDB and MyISAM Index Statistics Collection”
MySQL Glossary

innodb_stats_on_metadata

Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”

innodb_stats_persistent

Section 13.7.3.1, “ANALYZE TABLE Syntax”

Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 15.8.10, “Configuring Optimizer Statistics for InnoDB”
Configuring Optimizer Statistics Parameters for Individual Tables
Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”
Section 13.1.15, “CREATE INDEX Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”
Section 8.5.10, “Optimizing InnoDB for Systems with Many Tables”

innodb_stats_persistent_sample_pages

Configuring Optimizer Statistics Parameters for Individual Tables
Configuring the Number of Sampled Pages for InnoDB Optimizer Statistics
Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”

innodb_stats_transient_sample_pages

Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”

innodb_status_output

Section 15.16.2, “Enabling InnoDB Monitors”

innodb_status_output_locks

Section 15.16.2, “Enabling InnoDB Monitors”
Section 15.13, “InnoDB Startup Options and System Variables”

innodb_strict_mode

Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 13.1.20.3, “CREATE TEMPORARY TABLE Syntax”
Section 15.9.1.2, “Creating Compressed Tables”
Section 15.9.1.5, “How Compression Works for InnoDB Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 5.1.11, “Server SQL Modes”
Section 15.9.1.7, “SQL Compression Syntax Warnings and Errors”
Section 1.4, “What Is New in MySQL 8.0”

innodb_support_xa

MySQL Glossary

innodb_sync_debug

Section 2.9.7, “MySQL Source-Configuration Options”

innodb_table_locks

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.1.6, “Limits on InnoDB Tables”

innodb_temp_data_file_path

Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.6.3.5, “Temporary Tablespaces”
Section 25.11, “The INFORMATION_SCHEMA FILES Table”

innodb_temp_tablespaces_dir

Section 15.8.1, “InnoDB Startup Configuration”
Section 15.6.5, “Redo Log”
Section 15.6.3.5, “Temporary Tablespaces”
Section 1.4, “What Is New in MySQL 8.0”

innodb_thread_concurrency

Section 15.8.4, “Configuring Thread Concurrency for InnoDB”
Section 15.16.3, “InnoDB Standard Monitor and Lock Monitor Output”
Section 15.13, “InnoDB Startup Options and System Variables”
Section A.15, “MySQL 8.0 FAQ: MySQL Enterprise Thread Pool”
Section 8.5.9, “Optimizing InnoDB Configuration Variables”

innodb_thread_sleep_delay

Section 15.8.4, “Configuring Thread Concurrency for InnoDB”
Section 15.13, “InnoDB Startup Options and System Variables”

innodb_tmpdir

Section 15.12.5, “Online DDL Failure Conditions”
Section 15.12.3, “Online DDL Space Requirements”
Section 15.6.5, “Redo Log”

innodb_undo_directory

Section 2.11.4, “Changes in MySQL 8.0”
Section 5.6.7.3, “Cloning Remote Data”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.17.2, “InnoDB Recovery”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.3.8, “Moving Tablespace Files While the Server is Offline”
Section 15.6.5, “Redo Log”
Section 15.6.3.4, “Undo Tablespaces”
Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”

innodb_undo_log_encrypt

Section 5.6.7.4, “Cloning Encrypted Data”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”

innodb_undo_log_truncate

Section 15.13, “InnoDB Startup Options and System Variables”
Section 15.6.3.4, “Undo Tablespaces”
Section 1.4, “What Is New in MySQL 8.0”

innodb_undo_logs
Section 15.13, “InnoDB Startup Options and System Variables”

innodb_undo_tablespaces
Section 2.11.4, “Changes in MySQL 8.0”
Section 15.8.2, “Configuring InnoDB for Read-Only Operation”
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 15.6.3.4, “Undo Tablespaces”
Section 1.4, “What Is New in MySQL 8.0”

innodb_use_native_aio
Section 15.13, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 15.8.6, “Using Asynchronous I/O on Linux”

innodb_write_io_threads
Section 15.8.5, “Configuring the Number of Background InnoDB I/O Threads”
Section 15.16.3, “InnoDB Standard Monitor and Lock Monitor Output”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 29.1.1, “MySQL Threads”
Section 15.8.6, “Using Asynchronous I/O on Linux”

insert_id
Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 5.1.8, “Server System Variables”

interactive_timeout
Section B.4.2.10, “Communication Errors and Aborted Connections”
Section 28.7.7.54, “mysql_real_connect()”
Section 5.1.8, “Server System Variables”

internal_tmp_disk_storage_engine
Section 8.12.3.1, “How MySQL Uses Memory”
Section 15.8.1, “InnoDB Startup Configuration”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 5.1.8, “Server System Variables”
Section 15.6.3.5, “Temporary Tablespaces”
Section 1.4, “What Is New in MySQL 8.0”

internal_tmp_mem_storage_engine
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

J

[index top]

join_buffer_size

Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”
Section 8.2.1.4, “Hash Join Optimization”
Section 8.2.1.7, “Nested-Loop Join Algorithms”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

K

[index top]

keep_files_on_create

Section 5.1.8, “Server System Variables”

key_buffer_size

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 8.8.5, “Estimating Query Performance”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 15.8.1, “InnoDB Startup Configuration”
Section B.4.7, “Known Issues in MySQL”
Section 8.10.2.2, “Multiple Key Caches”
Section 8.2.5.3, “Optimizing DELETE Statements”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 8.10.2.6, “Restructuring a Key Cache”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 5.1.9.5, “Structured System Variables”
Section 8.10.2, “The MyISAM Key Cache”

key_cache_age_threshold

Section 8.10.2.3, “Midpoint Insertion Strategy”
Section 5.1.8, “Server System Variables”
Section 5.1.9.5, “Structured System Variables”

key_cache_block_size

Section 8.10.2.5, “Key Cache Block Size”
Section 8.10.2.6, “Restructuring a Key Cache”
Section 5.1.8, “Server System Variables”
Section 5.1.9.5, “Structured System Variables”

key_cache_division_limit

Section 8.10.2.3, “Midpoint Insertion Strategy”
Section 5.1.8, “Server System Variables”
Section 5.1.9.5, “Structured System Variables”

keyring_aws_cmk_id

Section 6.4.4.11, “Keyring System Variables”
Section 6.4.4.9, “Plugin-Specific Keyring Key-Management Functions”
Section 6.4.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”

keyring_aws_conf_file
Section 6.4.4.11, “Keyring System Variables”
Section 6.4.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”

keyring_aws_data_file
Section 6.4.4.11, “Keyring System Variables”
Section 6.4.4.9, “Plugin-Specific Keyring Key-Management Functions”
Section 6.4.4.5, “Using the keyring_aws Amazon Web Services Keyring Plugin”

keyring_aws_region
Section 6.4.4.11, “Keyring System Variables”

keyring_encrypted_file_data
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 6.4.4.11, “Keyring System Variables”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 6.4.4.3, “Using the keyring_encrypted_file Keyring Plugin”

keyring_encrypted_file_password
Section 6.4.4.11, “Keyring System Variables”
Section 6.4.4.3, “Using the keyring_encrypted_file Keyring Plugin”

keyring_file_data
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 2.4.3, “Installing and Using the MySQL Launch Daemon”
Section 6.4.4.11, “Keyring System Variables”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 6.4.4.2, “Using the keyring_file File-Based Plugin”
Section 29.2.4.12, “Writing Keyring Plugins”

keyring_okv_conf_dir
Section 6.4.4.11, “Keyring System Variables”
Section 6.4.4.4, “Using the keyring_okv KMIP Plugin”

keyring_operations
Section 6.4.4.11, “Keyring System Variables”
Section 6.4.4.6, “Migrating Keys Between Keyring Keystores”

L

[index top]

large_files_support
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 5.1.8, “Server System Variables”

large_page_size
Section 5.1.8, “Server System Variables”

large_pages
Section 5.1.8, “Server System Variables”

last_insert_id

Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”

lc_messages

Section 5.1.8, “Server System Variables”
Section 10.12, “Setting the Error Message Language”

lc_messages_dir

Section 5.1.8, “Server System Variables”
Section 10.12, “Setting the Error Message Language”

lc_time_names

Section 12.7, “Date and Time Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 10.16, “MySQL Server Locale Support”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”
Section 12.5, “String Functions and Operators”

license

Section 5.1.8, “Server System Variables”

local

Section 13.2.8, “LOAD XML Syntax”

local_infile

Section 13.2.7, “LOAD DATA Syntax”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 6.1.6, “Security Issues with LOAD DATA LOCAL”
Section 5.1.8, “Server System Variables”

lock_order

Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_debug_loop

Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_debug_missing_arc

Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_debug_missing_key

Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_debug_missing_unlock

Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_dependencies

Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_extra_dependencies
Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_output_directory
Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_print_txt
Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_trace_loop
Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_trace_missing_arc
Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_trace_missing_key
Section 29.5.3, “The LOCK_ORDER Tool”

lock_order_trace_missing_unlock
Section 29.5.3, “The LOCK_ORDER Tool”

lock_wait_timeout
Section 13.3.5, “LOCK INSTANCE FOR BACKUP and UNLOCK INSTANCE Syntax”
Section 5.1.8, “Server System Variables”

locked_in_memory
Section 5.1.8, “Server System Variables”

log
Section 18.9.1, “Group Replication Requirements”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 17.2.3.3, “Startup Options and Replication Channels”
Section 15.19.7, “The InnoDB memcached Plugin and Replication”

log_bin
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 17.5.5, “How to Report Replication Bugs or Problems”
NDB Cluster System Variables
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 13.4.1.2, “RESET MASTER Syntax”
Section 17.1.2.1, “Setting the Replication Master Configuration”
Section 5.4.4, “The Binary Log”

log_bin_basename
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 5.4.4, “The Binary Log”

log_bin_index
Section 17.1.6.4, “Binary Logging Options and Variables”

log_bin_trust_function_creators

Section 17.1.6.4, “Binary Logging Options and Variables”
Section A.4, “MySQL 8.0 FAQ: Stored Procedures and Functions”
Section 24.7, “Stored Program Binary Logging”

log_bin_use_v

Section 17.1.6.4, “Binary Logging Options and Variables”
MySQL Server Options for NDB Cluster
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”

log_builtin_as_identified_by_password

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 1.4, “What Is New in MySQL 8.0”

log_error

Section 5.4.2.2, “Default Error Log Destination Configuration”
Section 5.5.3, “Error Log Components”
Section 5.4.2.6, “Error Logging in JSON Format”
Section 2.4.3, “Installing and Using the MySQL Launch Daemon”
Section 2.5.6.2, “More Topics on Deploying MySQL Server with Docker”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

log_error_filter_rules

Section 5.1.8, “Server System Variables”

log_error_services

Section 5.4.2.2, “Default Error Log Destination Configuration”
Section 5.4.2.1, “Error Log Component Configuration”
Section 5.5.3, “Error Log Components”
Section 5.4.2.6, “Error Logging in JSON Format”
Section 5.4.2.7, “Error Logging to the System Log”
Section 5.4.2.5, “Rule-Based Error Log Filtering (log_filter_dragnet)”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

log_error_suppression_list

Section 5.4.2.1, “Error Log Component Configuration”
Section 5.5.3, “Error Log Components”
Section 5.4.2.4, “Priority-Based Error Log Filtering (log_filter_internal)”
Section 5.1.8, “Server System Variables”
Section 5.4.2.3, “Types of Error Log Filtering”

log_error_verbosity

Section B.4.2.10, “Communication Errors and Aborted Connections”
Section 5.4.2.1, “Error Log Component Configuration”
Section 5.5.3, “Error Log Components”
Section 5.4.2.8, “Error Log Message Format”
Section 5.4.2.7, “Error Logging to the System Log”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section B.4.2.8, “MySQL server has gone away”

Section 26.12.9, “Performance Schema Connection Attribute Tables”
Section 26.15, “Performance Schema System Variables”
Section 5.4.2.4, “Priority-Based Error Log Filtering (log_filter_internal)”
Section 5.4.2.5, “Rule-Based Error Log Filtering (log_filter_dragnet)”
Section 5.1.7, “Server Command Options”
Section 5.1.3, “Server Configuration Validation”
Section 5.1.8, “Server System Variables”
Section 5.4.2.3, “Types of Error Log Filtering”
Section 6.4.7.3, “Using MySQL Enterprise Firewall”
Section 1.4, “What Is New in MySQL 8.0”

log_output

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 5.4.3, “The General Query Log”
Section 5.4.5, “The Slow Query Log”

log_queries_not_using_indexes

Section 5.1.8, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

log_slave_updates

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.3.1, “GTID Format and Storage”
Section 17.4.7, “Improving Replication Performance”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
MySQL Server Options for NDB Cluster
Section 22.3.2.5, “NDB Cluster mysqld Option and Variable Reference”
Section 22.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”
NDB Cluster System Variables
Section 17.1.2.2, “Setting the Replication Slave Configuration”
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”
Section 5.4.4, “The Binary Log”

log_slow_admin_statements

Section 5.1.8, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

log_slow_extra

Section 5.1.8, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

log_slow_slave_statements

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.4.5, “The Slow Query Log”

log_statements_unsafe_for_binlog

Section 17.1.6.4, “Binary Logging Options and Variables”

log_syslog

Section 5.1.8, “Server System Variables”

log_syslog_facility

Section 5.4.2.7, “Error Logging to the System Log”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 5.1.8, “Server System Variables”

log_syslog_include_pid

Section 5.4.2.7, “Error Logging to the System Log”
Section 5.1.8, “Server System Variables”

log_syslog_tag

Section 5.4.2.7, “Error Logging to the System Log”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 5.1.8, “Server System Variables”

log_throttle_queries_not_using_indexes

Section 5.1.8, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

log_timestamps

Section 5.4.2.8, “Error Log Message Format”
Section 5.1.8, “Server System Variables”
Section 5.4.3, “The General Query Log”
Section 5.4.5, “The Slow Query Log”

log_warnings

Section 5.1.8, “Server System Variables”

long_query_time

Section 5.4, “MySQL Server Logs”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

low_priority_updates

Section A.14, “MySQL 8.0 FAQ: Replication”
Section 5.1.8, “Server System Variables”
Section 8.11.2, “Table Locking Issues”

lower

Section 18.9.1, “Group Replication Requirements”

lower_case_file_system

Section 5.1.8, “Server System Variables”

lower_case_table_names

Advanced Options
Section 2.11.4, “Changes in MySQL 8.0”
Section 15.6.3.7, “Copying Tablespaces to Another Instance”
Section 13.7.1.6, “GRANT Syntax”

Section 18.8, “Group Replication System Variables”
Section 17.2.5, “How Servers Evaluate Replication Filtering Rules”
Section 1.7, “How to Report Bugs or Problems”
Section 9.2.3, “Identifier Case Sensitivity”
Section 13.2.5, “IMPORT TABLE Syntax”
Section 15.6.1.2, “Moving or Copying InnoDB Tables”
Section 18.7.3.1, “Online Upgrade Considerations”
Section 17.5.1.38, “Replication and Variables”
Section 13.7.1.8, “REVOKE Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.7.37, “SHOW TABLES Syntax”
Section 25.39.8, “The INFORMATION_SCHEMA INNODB_COLUMNS Table”
Section 25.39.24, “The INFORMATION_SCHEMA INNODB_TABLES Table”
Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”
Section 13.1.20.6, “Using FOREIGN KEY Constraints”

M

[index top]

mandatory_roles

Section 6.2.11, “Account Categories”
Section 13.7.1.4, “DROP ROLE Syntax”
Section 13.7.1.5, “DROP USER Syntax”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.8, “REVOKE Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.1.11, “SET ROLE Syntax”
Section 13.7.7.21, “SHOW GRANTS Syntax”
Section 5.1.9.1, “System Variable Privileges”
Section 6.2.10, “Using Roles”

master

Section 18.9.1, “Group Replication Requirements”

master_info_repository

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.4.2.4, “RESET SLAVE Syntax”
Section 13.3.8.3, “Restrictions on XA Transactions”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”

master_verify_checksum

Section 17.1.6.4, “Binary Logging Options and Variables”
MySQL Glossary
Section 5.4.4, “The Binary Log”

max_allowed_packet

Section 12.20.1, “Aggregate (GROUP BY) Function Descriptions”
Section B.4.2.10, “Communication Errors and Aborted Connections”
Section 12.3.2, “Comparison Functions and Operators”
Section 11.8, “Data Type Storage Requirements”
Section B.4.4.6, “Deleting Rows from Related Tables”

Section 8.12.3.1, “How MySQL Uses Memory”
Section B.4.2.3, “Lost connection to MySQL server”
Section 28.7, “MySQL C API”
Section B.4.2.8, “MySQL server has gone away”
Section 28.7.7.50, “mysql_options()”
Section 28.7.11.26, “mysql_stmt_send_long_data()”
Section 28.7.7.81, “mysql_use_result()”
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
Section B.4.2.9, “Packet Too Large”
Section 17.5.1.20, “Replication and max_allowed_packet”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.1.8, “Server System Variables”
Section 12.5, “String Functions and Operators”
Section 11.4.3, “The BLOB and TEXT Types”
Section 11.6, “The JSON Data Type”
Section 5.6.6.3, “Using Version Tokens”
Section 20.5.5.2, “X Plugin Options and System Variables”

max_binlog_cache_size

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 5.4.4, “The Binary Log”

max_binlog_size

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.3.1, “GTID Format and Storage”
Section 5.4, “MySQL Server Logs”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.4.7, “Server Log Maintenance”
Section 5.4.4, “The Binary Log”
Section 17.2.4.1, “The Slave Relay Log”

max_binlog_stmt_cache_size

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 8.12.3.1, “How MySQL Uses Memory”

max_connect_errors

Section 8.12.4.2, “DNS Lookup Optimization and the Host Cache”
Section 13.7.8.3, “FLUSH Syntax”
Section B.4.2.5, “Host 'host_name' is blocked”
Section 5.1.8, “Server System Variables”
Section 26.12.18.1, “The host_cache Table”

max_connections

Section 29.5.1.4, “Debugging mysqld under gdb”
Section 14.4, “Dictionary Object Cache”
Section B.4.2.17, “File Not Found and Similar Errors”
Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 26.15, “Performance Schema System Variables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

Section 5.6.3.3, “Thread Pool Operation”
Section B.4.2.6, “Too many connections”
Section 1.4, “What Is New in MySQL 8.0”
Section 20.5.5.2, “X Plugin Options and System Variables”

max_delayed_threads

Section 5.1.8, “Server System Variables”

max_digest_length

Section 12.13, “Encryption and Compression Functions”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 26.10, “Performance Schema Statement Digests and Sampling”
Section 26.15, “Performance Schema System Variables”
Section 5.1.8, “Server System Variables”
Section 6.4.7.3, “Using MySQL Enterprise Firewall”

max_error_count

Section 13.2.7, “LOAD DATA Syntax”
Section 13.6.7.4, “RESIGNAL Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.7.17, “SHOW ERRORS Syntax”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”

max_execution_time

Section 8.9.3, “Optimizer Hints”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 13.2.13, “WITH Syntax (Common Table Expressions)”

max_heap_table_size

Section 8.12.3.1, “How MySQL Uses Memory”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 8.4.6, “Limits on Table Size”
Section 17.5.1.21, “Replication and MEMORY Tables”
Section 17.5.1.38, “Replication and Variables”
Section 13.6.6.5, “Restrictions on Server-Side Cursors”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 16.3, “The MEMORY Storage Engine”

max_insert_delayed_threads

Section 5.1.8, “Server System Variables”

max_join_size

Section 8.8.2, “EXPLAIN Output Format”
Section 4.5.1.6, “mysql Client Tips”
Section 5.1.8, “Server System Variables”
Section 13.7.6.1, “SET Syntax for Variable Assignment”

max_length_for_sort_data

Section 8.2.1.16, “ORDER BY Optimization”

Section 5.1.8, “Server System Variables”

max_points_in_geometry

Section 5.1.8, “Server System Variables”
Section 12.16.8, “Spatial Operator Functions”

max_prepared_stmt_count

Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 13.5.3, “DEALLOCATE PREPARE Syntax”
Section 26.15, “Performance Schema System Variables”
Section 13.5, “Prepared SQL Statement Syntax”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

max_relay_log_size

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.4.1, “The Slave Relay Log”

max_seeks_for_key

Section 15.6.1.6, “Limits on InnoDB Tables”
Section 5.1.8, “Server System Variables”

max_sort_length

Section B.4.7, “Known Issues in MySQL”
Section 8.2.1.16, “ORDER BY Optimization”
Section 13.2.10, “SELECT Syntax”
Section 5.1.8, “Server System Variables”
Section 11.4.3, “The BLOB and TEXT Types”
Section 11.6, “The JSON Data Type”

max_sp_recursion_depth

Section 5.1.8, “Server System Variables”
Section 24.2.1, “Stored Routine Syntax”

max_tmp_tables

Section 5.1.8, “Server System Variables”

max_user_connections

Section 13.7.1.1, “ALTER USER Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 13.7.8.3, “FLUSH Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 5.1.8, “Server System Variables”
Section 6.2.20, “Setting Account Resource Limits”

max_write_lock_count

Section 13.3.6, “LOCK TABLES and UNLOCK TABLES Syntax”
Section 8.11.4, “Metadata Locking”
Section 5.1.8, “Server System Variables”
Section 8.11.2, “Table Locking Issues”

mecab_rc_file

Section 12.9.9, “MeCab Full-Text Parser Plugin”

metadata_locks_cache_size

Section 5.1.8, “Server System Variables”

metadata_locks_hash_instances

Section 5.1.8, “Server System Variables”

min_examined_row_limit

Section 5.1.8, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

multi_range_count

Section 5.1.8, “Server System Variables”

myisam_data_pointer_size

Section 13.1.20, “CREATE TABLE Syntax”
Section 8.4.6, “Limits on Table Size”
Section 5.1.8, “Server System Variables”

myisam_max_sort_file_size

Section 16.2.1, “MyISAM Startup Options”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 5.1.8, “Server System Variables”

myisam_mmap_size

Section 5.1.8, “Server System Variables”

myisam_recover_options

Section 16.2.1, “MyISAM Startup Options”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 5.1.8, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section B.4.2.18, “Table-Corruption Issues”
Section 16.2, “The MyISAM Storage Engine”
Section 29.5.1.6, “Using Server Logs to Find Causes of Errors in mysqld”

myisam_repair_threads

Section 5.1.8, “Server System Variables”

myisam_sort_buffer_size

Section 13.1.9, “ALTER TABLE Syntax”
Section 16.2.1, “MyISAM Startup Options”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 5.1.8, “Server System Variables”

myisam_stats_method

Section 8.3.8, “InnoDB and MyISAM Index Statistics Collection”
Section 5.1.8, “Server System Variables”

myisam_use_mmap

Section 8.12.3.1, “How MySQL Uses Memory”
Section 5.1.8, “Server System Variables”

mysql_firewall_mode

Section 6.4.7.4, “MySQL Enterprise Firewall Reference”
Section 6.4.7.3, “Using MySQL Enterprise Firewall”

mysql_firewall_trace

Section 6.4.7.4, “MySQL Enterprise Firewall Reference”
Section 6.4.7.3, “Using MySQL Enterprise Firewall”

mysql_native_password_proxy_users

Section 6.2.18, “Proxy Users”
Section 5.1.8, “Server System Variables”

mysqlx_bind_address

Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_connect_timeout

Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_document_id_unique_prefix

Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_enable_hello_notice

Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_idle_worker_thread_timeout

Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_interactive_timeout

Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_max_allowed_packet

Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_max_connections

Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_min_worker_threads

Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_port

Section 5.6.7.13, “Clone Plugin Limitations”
Section 13.7.5, “CLONE Syntax”
Section 5.6.7.3, “Cloning Remote Data”
Section 20.3.1, “MySQL Shell”
Section 20.4.1, “MySQL Shell”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_port_open_timeout
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_read_timeout
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_socket
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_ssl_ca
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_ssl_capath
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_ssl_cert
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_ssl_cipher
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_ssl_crl
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_ssl_crlpath
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_ssl_key
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_wait_timeout
Section 20.5.5.2, “X Plugin Options and System Variables”

mysqlx_write_timeout
Section 20.5.5.2, “X Plugin Options and System Variables”

N

[index top]

named_pipe
Section B.4.2.2, “Can't connect to [local] MySQL server”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.2, “`mysql_secure_installation` — Improve MySQL Installation Security”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 2.3.4.3, “Selecting a MySQL Server Type”
Section 5.1.8, “Server System Variables”
Section 1.3.2, “The Main Features of MySQL”
Type and Networking

named_pipe_full_access_group

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.2, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.4.5, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — Client for Administering a MySQL Server”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

ndb_autoincrement_prefetch_sz

NDB Cluster System Variables
Section 22.4.13, “ndb_import — Import CSV Data Into NDB”

ndb_cache_check_time

NDB Cluster System Variables

ndb_clear_apply_status

NDB Cluster System Variables
Section 13.4.2.4, “RESET SLAVE Syntax”

ndb_data_node_neighbour

NDB Cluster System Variables
Section 13.1.20.11, “Setting NDB_TABLE Options”

ndb_dbg_check_shares

NDB Cluster System Variables

ndb_default_column_format

NDB Cluster System Variables

ndb_deferred_constraints

NDB Cluster System Variables

ndb_distribution

NDB Cluster System Variables

ndb_eventbuffer_free_percent

Section 22.5.7.3, “Event Buffer Reporting in the Cluster Log”
NDB Cluster System Variables

ndb_eventbuffer_max_alloc

Section 22.5.7.3, “Event Buffer Reporting in the Cluster Log”
NDB Cluster System Variables

ndb_extra_logging

NDB Cluster System Variables

ndb_force_send

NDB Cluster System Variables

ndb_fully_replicated

NDB Cluster System Variables

ndb_index_stat_enable

NDB Cluster System Variables

ndb_index_stat_option

NDB Cluster System Variables

ndb_join_pushdown

Section 8.8.2, “EXPLAIN Output Format”
NDB Cluster System Variables

ndb_log_apply_status

Section 22.6.10, “NDB Cluster Replication: Multi-Master and Circular Replication”
NDB Cluster System Variables

ndb_log_bin

NDB Cluster System Variables
Section 22.1.4, “What is New in NDB Cluster”

ndb_log_binlog_index

NDB Cluster System Variables

ndb_log_empty_epochs

NDB Cluster System Variables

ndb_log_empty_update

NDB Cluster System Variables

ndb_log_exclusive_reads

MySQL Server Options for NDB Cluster
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster System Variables

ndb_log_orig

NDB Cluster System Variables

ndb_log_transaction_id

NDB Cluster System Variables

ndb_metadata_check

NDB Cluster System Variables
Section 22.4.23, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 22.1.4, “What is New in NDB Cluster”

ndb_metadata_check_interval

NDB Cluster System Variables
Section 22.1.4, “What is New in NDB Cluster”

ndb_optimized_node_selection

NDB Cluster System Variables
Section 22.3.3.10, “NDB Cluster TCP/IP Connections”
Section 22.5.6.3, “Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client”

ndb_read_backup

NDB Cluster System Variables
Section 13.1.20.11, “Setting NDB_TABLE Options”

ndb_recv_thread_activation_threshold

NDB Cluster System Variables

ndb_recv_thread_cpu_mask

NDB Cluster System Variables

ndb_report_thresh_binlog_epoch_slip

Section 22.5.7.3, “Event Buffer Reporting in the Cluster Log”
NDB Cluster System Variables

ndb_report_thresh_binlog_mem_usage

Section 22.5.7.3, “Event Buffer Reporting in the Cluster Log”
NDB Cluster System Variables

ndb_row_checksum

NDB Cluster System Variables

ndb_schema_dist_lock_wait_timeout

NDB Cluster System Variables

ndb_schema_dist_timeout

NDB Cluster System Variables

ndb_schema_dist_upgrade_allowed

NDB Cluster System Variables

ndb_show_foreign_key_mock_tables

NDB Cluster System Variables

ndb_slave_conflict_role

Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster System Variables

ndb_table_no_logging

NDB Cluster System Variables
Section 13.1.20.11, “Setting NDB_TABLE Options”

ndb_table_temporary

NDB Cluster System Variables

ndb_use_copying_alter_table

NDB Cluster System Variables

ndb_use_exact_count

NDB Cluster System Variables

ndb_use_transactions

NDB Cluster System Variables

ndb_version

NDB Cluster System Variables

ndb_version_string

NDB Cluster System Variables

ndbinfo_database

NDB Cluster System Variables

ndbinfo_max_bytes

NDB Cluster System Variables

ndbinfo_max_rows

NDB Cluster System Variables

ndbinfo_offline

NDB Cluster System Variables

ndbinfo_show_hidden

NDB Cluster System Variables
Section 22.5.10.5, “The ndbinfo cluster_operations Table”
Section 22.5.10.6, “The ndbinfo cluster_transactions Table”
Section 22.5.10.34, “The ndbinfo server_operations Table”
Section 22.5.10.35, “The ndbinfo server_transactions Table”

ndbinfo_table_prefix

NDB Cluster System Variables

ndbinfo_version

NDB Cluster System Variables

net_buffer_length
Section 8.12.3.1, “How MySQL Uses Memory”
Section 28.7, “MySQL C API”
Section 28.7.7.50, “mysql_options()”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 5.1.8, “Server System Variables”

net_read_timeout
Section B.4.2.3, “Lost connection to MySQL server”
Section 5.1.8, “Server System Variables”

net_retry_count
Section 5.1.8, “Server System Variables”

net_write_timeout
Section 5.1.8, “Server System Variables”

new
Section 23.6.2, “Partitioning Limitations Relating to Storage Engines”
Section 5.1.8, “Server System Variables”

ngram_token_size
Section 12.9.2, “Boolean Full-Text Searches”
Section 12.9.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.9.4, “Full-Text Stopwords”
Section 12.9.1, “Natural Language Full-Text Searches”
Section 12.9.8, “ngram Full-Text Parser”

O

[index top]

offline_mode
Section 18.6.6.4, “Exit Action”
Section 18.3.1, “Group Replication Server States”
Section 18.8, “Group Replication System Variables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.8, “Server System Variables”
Section 21.4, “Working with InnoDB Cluster”

old
Section 8.9.4, “Index Hints”
Section 5.1.8, “Server System Variables”

old_alter_table
Section 13.1.9, “ALTER TABLE Syntax”
Section 15.12, “InnoDB and Online DDL”
Section 23.3.1, “Management of RANGE and LIST Partitions”
Section 15.12.1, “Online DDL Operations”
Section 15.12.2, “Online DDL Performance and Concurrency”

Section 13.7.3.4, “OPTIMIZE TABLE Syntax”
Section 5.1.8, “Server System Variables”
Section 15.12.4, “Simplifying DDL Statements with Online DDL”
Section 1.4, “What Is New in MySQL 8.0”

old_passwords

Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

open_files_limit

Section B.4.2.17, “File Not Found and Similar Errors”
Section 8.2.1.4, “Hash Join Optimization”
Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 2.5.9, “Managing MySQL Server with systemd”
Section 26.15, “Performance Schema System Variables”
Section 23.6, “Restrictions and Limitations on Partitioning”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

optimizer_prune_level

Section 8.9.1, “Controlling Query Plan Evaluation”
Section 8.9.3, “Optimizer Hints”
Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”
Section 5.1.8, “Server System Variables”

optimizer_search_depth

Section 8.9.1, “Controlling Query Plan Evaluation”
Section 5.1.8, “Server System Variables”

optimizer_switch

Section 8.2.1.12, “Block Nested-Loop and Batched Key Access Joins”
Section 8.2.1.13, “Condition Filtering”
Section 8.2.1.5, “Engine Condition Pushdown Optimization”
Section 8.2.1.4, “Hash Join Optimization”
Section 8.2.1.6, “Index Condition Pushdown Optimization”
Section 8.2.1.3, “Index Merge Optimization”
Section 8.3.12, “Invisible Indexes”
Section 8.2.1.11, “Multi-Range Read Optimization”
Section 8.9.3, “Optimizer Hints”
Section 8.9.6, “Optimizer Statistics”
Section 8.2.2.4, “Optimizing Derived Tables, View References, and Common Table Expressions with Merging or Materialization”
Section 8.2.2.1, “Optimizing IN and EXISTS Subquery predicates with Semijoin Transformations”
Section 8.2.2.2, “Optimizing Subqueries with Materialization”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 8.2.1.2, “Range Optimization”
Section B.3.1, “Server Error Message Reference”
Section 5.1.8, “Server System Variables”
Section 8.9.2, “Switchable Optimizations”
Section 27.4.5.7, “The list_add() Function”

Section 8.3.10, “Use of Index Extensions”
Section 24.5.2, “View Processing Algorithms”
Section 1.4, “What Is New in MySQL 8.0”

optimizer_trace

Section 5.1.8, “Server System Variables”
Section 25.15, “The INFORMATION_SCHEMA OPTIMIZER_TRACE Table”

optimizer_trace_features

Section 5.1.8, “Server System Variables”

optimizer_trace_limit

Section 5.1.8, “Server System Variables”

optimizer_trace_max_mem_size

Section 5.1.8, “Server System Variables”
Section 25.15, “The INFORMATION_SCHEMA OPTIMIZER_TRACE Table”

optimizer_trace_offset

Section 5.1.8, “Server System Variables”

original_commit_timestamp

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”

original_server_version

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 17.5.2, “Replication Compatibility Between MySQL Versions”
Section 17.1.6.2, “Replication Master Options and Variables”
Section 5.1.8, “Server System Variables”

P

[index top]

parser_max_mem_size

Section 5.1.8, “Server System Variables”

partial_revokes

Section 6.2.11, “Account Categories”
Section 13.7.1.6, “GRANT Syntax”
Section 6.2.12, “Privilege Restriction Using Partial Revokes”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

password_history

Section 13.7.1.1, “ALTER USER Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 6.2.15, “Password Management”
Section 5.1.8, “Server System Variables”

password_require_current
Section 13.7.1.1, “ALTER USER Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 6.2.15, “Password Management”
Section 5.1.8, “Server System Variables”

password_reuse_interval
Section 13.7.1.1, “ALTER USER Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 6.2.15, “Password Management”
Section 5.1.8, “Server System Variables”

performance_schema
Section 26.1, “Performance Schema Quick Start”
Section 26.3, “Performance Schema Startup Configuration”
Section 26.15, “Performance Schema System Variables”

performance_schema_accounts_size
Section 26.12.14, “Performance Schema Status Variable Tables”
Section 26.15, “Performance Schema System Variables”
Section 26.12.17.12, “Status Variable Summary Tables”
Section 26.12.8.1, “The accounts Table”

performance_schema_digests_size
Section 26.10, “Performance Schema Statement Digests and Sampling”
Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”
Section 26.12.17.3, “Statement Summary Tables”

performance_schema_error_size
Section 26.15, “Performance Schema System Variables”

performance_schema_events_stages_history_long_size
Section 26.15, “Performance Schema System Variables”
Section 26.12.5.3, “The events_stages_history_long Table”

performance_schema_events_stages_history_size
Section 26.15, “Performance Schema System Variables”
Section 26.12.5.2, “The events_stages_history Table”

performance_schema_events_statements_history_long_size
Section 26.15, “Performance Schema System Variables”
Section 26.12.6.3, “The events_statements_history_long Table”

performance_schema_events_statements_history_size
Section 26.15, “Performance Schema System Variables”
Section 26.12.6.2, “The events_statements_history Table”

performance_schema_events_transactions_history_long_size
Section 26.15, “Performance Schema System Variables”
Section 26.12.7.3, “The events_transactions_history_long Table”

performance_schema_events_transactions_history_size

Section 26.15, “Performance Schema System Variables”
Section 26.12.7.2, “The events_transactions_history Table”

performance_schema_events_waits_history_long_size

Section 26.15, “Performance Schema System Variables”
Section 26.12, “Performance Schema Table Descriptions”
Section 13.7.7.15, “SHOW ENGINE Syntax”
Section 26.12.4.3, “The events_waits_history_long Table”

performance_schema_events_waits_history_size

Section 26.15, “Performance Schema System Variables”
Section 26.12, “Performance Schema Table Descriptions”
Section 13.7.7.15, “SHOW ENGINE Syntax”
Section 26.12.4.2, “The events_waits_history Table”

performance_schema_hosts_size

Section 26.12.14, “Performance Schema Status Variable Tables”
Section 26.15, “Performance Schema System Variables”
Section 26.12.17.12, “Status Variable Summary Tables”
Section 26.12.8.2, “The hosts Table”

performance_schema_max_cond_classes

Section 26.15, “Performance Schema System Variables”

performance_schema_max_cond_instances

Section 26.15, “Performance Schema System Variables”

performance_schema_max_digest_length

Section 26.10, “Performance Schema Statement Digests and Sampling”
Section 26.15, “Performance Schema System Variables”
Section 5.1.8, “Server System Variables”
Section 26.12.6.1, “The events_statements_current Table”

performance_schema_max_digest_sample_age

Section 26.10, “Performance Schema Statement Digests and Sampling”
Section 26.15, “Performance Schema System Variables”

performance_schema_max_file_classes

Section 26.15, “Performance Schema System Variables”

performance_schema_max_file_handles

Section 26.15, “Performance Schema System Variables”

performance_schema_max_file_instances

Section 26.15, “Performance Schema System Variables”

performance_schema_max_index_stat

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

performance_schema_max_memory_classes

Section 26.15, “Performance Schema System Variables”

performance_schema_max_metadata_locks

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”
Section 26.12.12.3, “The metadata_locks Table”

performance_schema_max_mutex_classes

Section 26.7, “Performance Schema Status Monitoring”
Section 26.15, “Performance Schema System Variables”

performance_schema_max_mutex_instances

Section 26.15, “Performance Schema System Variables”

performance_schema_max_prepared_statements_instances

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”
Section 26.12.6.4, “The prepared_statements_instances Table”

performance_schema_max_program_instances

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

performance_schema_max_rwlock_classes

Section 26.15, “Performance Schema System Variables”

performance_schema_max_rwlock_instances

Section 26.15, “Performance Schema System Variables”

performance_schema_max_socket_classes

Section 26.15, “Performance Schema System Variables”

performance_schema_max_socket_instances

Section 26.15, “Performance Schema System Variables”

performance_schema_max_sql_text_length

Section 26.10, “Performance Schema Statement Digests and Sampling”
Section 26.15, “Performance Schema System Variables”
Section 26.12.17.3, “Statement Summary Tables”
Section 26.12.6.1, “The events_statements_current Table”

performance_schema_max_stage_classes

Section 26.15, “Performance Schema System Variables”

performance_schema_max_statement_classes

Section 26.15, “Performance Schema System Variables”

performance_schema_max_statement_stack

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

performance_schema_max_table_handles

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”
Section 26.12.12.4, “The table_handles Table”

performance_schema_max_table_instances

Section 26.15, “Performance Schema System Variables”

performance_schema_max_table_lock_stat

Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

performance_schema_max_thread_classes

Section 26.15, “Performance Schema System Variables”

performance_schema_max_thread_instances

Section 12.22, “Performance Schema Functions”
Section 26.12.14, “Performance Schema Status Variable Tables”
Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”
Section 13.7.7.15, “SHOW ENGINE Syntax”

performance_schema_session_connect_attrs_size

Section 26.12.9, “Performance Schema Connection Attribute Tables”
Section 26.16, “Performance Schema Status Variables”
Section 26.15, “Performance Schema System Variables”

performance_schema_setup_actors_size

Section 26.15, “Performance Schema System Variables”
Section 26.12.2.1, “The setup_actors Table”

performance_schema_setup_objects_size

Section 26.15, “Performance Schema System Variables”
Section 26.12.2.4, “The setup_objects Table”

performance_schema_users_size

Section 26.12.14, “Performance Schema Status Variable Tables”
Section 26.15, “Performance Schema System Variables”
Section 26.12.17.12, “Status Variable Summary Tables”
Section 26.12.8.3, “The users Table”

persist_only_admin_x

Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”
Section 5.1.8, “Server System Variables”
Section 5.1.9.1, “System Variable Privileges”

persisted_globals_load

Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”
Section 26.12.13.2, “Performance Schema variables_info Table”
Section 5.1.9.3, “Persisted System Variables”
Section 21.2.4, “Production Deployment of InnoDB Cluster”

Section 13.7.8.7, “RESET PERSIST Syntax”
Section 5.1.8, “Server System Variables”
Section 4.2.2.2, “Using Option Files”

pid

Section 15.8.2, “Configuring InnoDB for Read-Only Operation”

pid_file

Section 2.4.3, “Installing and Using the MySQL Launch Daemon”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

plugin_dir

Section 6.1.2.2, “Administrator Guidelines for Password Security”
Section 29.2.4.3, “Compiling and Installing Plugin Libraries”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 13.7.4.1, “CREATE FUNCTION Syntax for User-Defined Functions”
Section 6.4.4.8, “General-Purpose Keyring Key-Management Functions”
Section 13.7.4.3, “INSTALL COMPONENT Syntax”
Section 13.7.4.4, “INSTALL PLUGIN Syntax”
Section 5.6.1, “Installing and Uninstalling Plugins”
Section 2.4.3, “Installing and Using the MySQL Launch Daemon”
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 5.6.5.1, “Installing or Uninstalling ddl_rewriter”
Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Audit”
Section 6.4.8.2, “Installing or Uninstalling MySQL Enterprise Data Masking and De-Identification”
Installing or Uninstalling the UDF Locking Interface
Section 5.6.6.2, “Installing or Uninstalling Version Tokens”
Section 5.6.7.1, “Installing the Clone Plugin”
Section 6.4.4.10, “Keyring Command Options”
Section 6.4.4.1, “Keyring Plugin Installation”
Section 6.4.1.7, “LDAP Pluggable Authentication”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 12.19.1, “MySQL Enterprise Encryption Installation”
Section 6.4.1.8, “No-Login Pluggable Authentication”
Section 6.4.1.5, “PAM Pluggable Authentication”
Section 6.4.3.1, “Password Validation Component Installation and Uninstallation”
Section 6.2.17, “Pluggable Authentication”
Section 16.11.1, “Pluggable Storage Engine Architecture”
Section 29.2.3, “Plugin API Components”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 15.19.3, “Setting Up the InnoDB memcached Plugin”
Section 13.7.7.25, “SHOW PLUGINS Syntax”
Section 6.4.1.9, “Socket Peer-Credential Pluggable Authentication”
Section 6.4.1.10, “Test Pluggable Authentication”
Section 6.4.6, “The Audit Message Component”
Section 25.18, “The INFORMATION_SCHEMA PLUGINS Table”
Section 26.12.18.6, “The user_defined_functions Table”
Section 5.6.3.2, “Thread Pool Installation”
Section 29.4.2.5, “UDF Compiling and Installing”
Section 29.4.2.6, “UDF Security Precautions”

Using the Authentication Plugins
Using Your Own Protocol Trace Plugins
Section 6.4.1.6, “Windows Pluggable Authentication”
Section 29.2.4.8, “Writing Audit Plugins”
Section 29.2.4.5, “Writing Daemon Plugins”
Section 29.2.4.4, “Writing Full-Text Parser Plugins”
Section 29.2.4.6, “Writing INFORMATION_SCHEMA Plugins”
Section 29.2.4.12, “Writing Keyring Plugins”
Section 29.2.4.10, “Writing Password-Validation Plugins”
Section 29.2.4.7, “Writing Semisynchronous Replication Plugins”

port

Section B.4.2.2, “Can't connect to [local] MySQL server”
Section 13.7.5, “CLONE Syntax”
Section 5.6.7.3, “Cloning Remote Data”
Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 18.10, “Frequently Asked Questions”
Section 20.4.1, “MySQL Shell”
Section 5.1.8, “Server System Variables”
Section 26.12.11.9, “The replication_group_members Table”
Section 21.3, “Using MySQL Router with InnoDB Cluster”
Section 20.5.5.2, “X Plugin Options and System Variables”

preload_buffer_size

Section 5.1.8, “Server System Variables”

print_identified_with_as_hex

Section 6.2.8, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 13.7.1.1, “ALTER USER Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.7.12, “SHOW CREATE USER Syntax”

profiling

Section 5.1.8, “Server System Variables”
Section 13.7.7.30, “SHOW PROFILE Syntax”
Section 25.20, “The INFORMATION_SCHEMA PROFILING Table”

profiling_history_size

Section 5.1.8, “Server System Variables”
Section 13.7.7.30, “SHOW PROFILE Syntax”

protocol_compression_algorithms

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.6, “Connection Compression Control”
Section 18.8, “Group Replication System Variables”
Section 4.5.1.1, “mysql Client Options”
Section 28.7.7.50, “mysql_options()”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlpump — A Database Backup Program”
Section 4.5.7, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.8, “mysqlslap — Load Emulation Client”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

protocol_version

Section 5.1.9.4, “Nonpersistible and Persist-Restricted System Variables”
Section 5.1.8, “Server System Variables”

proxy_user

Section 6.2.18, “Proxy Users”
Section 5.1.8, “Server System Variables”

pseudo_slave_mode

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 5.1.8, “Server System Variables”

pseudo_thread_id

Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”

Q

[index top]

query_alloc_block_size

Section 5.1.8, “Server System Variables”

query_cache_limit

Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

query_cache_min_res_unit

Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

query_cache_size

Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

query_cache_type

Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

query_cache_wlock_invalidate

Section 5.1.8, “Server System Variables”

Section 1.4, “What Is New in MySQL 8.0”

query_prealloc_size

Section 5.1.8, “Server System Variables”

R

[index top]

rand_seed

Section 5.1.8, “Server System Variables”

range_alloc_block_size

Section 5.1.8, “Server System Variables”

range_optimizer_max_mem_size

Section 4.5.1.6, “mysql Client Tips”
Section 8.2.1.2, “Range Optimization”
Section 5.1.8, “Server System Variables”

rbr_exec_mode

Section 5.1.8, “Server System Variables”

read_buffer_size

Section 8.12.3.1, “How MySQL Uses Memory”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 5.1.8, “Server System Variables”

read_only

Section 13.7.1, “Account Management Statements”
Section 13.7.1.1, “ALTER USER Syntax”
Section 6.2.14, “Assigning Account Passwords”
Section 17.4.1.3, “Backing Up a Master or Slave by Making It Read Only”
Section 13.7.1.2, “CREATE ROLE Syntax”
Section 13.7.1.3, “CREATE USER Syntax”
Section 13.7.1.4, “DROP ROLE Syntax”
Section 13.7.1.5, “DROP USER Syntax”
Section 13.7.1.6, “GRANT Syntax”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.7, “RENAME USER Syntax”
Section 17.5.1.38, “Replication and Variables”
Section 13.7.1.8, “REVOKE Syntax”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 13.7.1.10, “SET PASSWORD Syntax”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 21.4, “Working with InnoDB Cluster”

read_rnd_buffer_size

Section 8.12.3.1, “How MySQL Uses Memory”

Section 8.2.1.11, “Multi-Range Read Optimization”
Section 8.2.1.16, “ORDER BY Optimization”
Section 5.1.8, “Server System Variables”

regexp_stack_limit

Section 12.5.2, “Regular Expressions”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

regexp_time_limit

Section 12.5.2, “Regular Expressions”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

relay

Section 18.9.1, “Group Replication Requirements”
Section 17.2.3.3, “Startup Options and Replication Channels”

relay_log

Section 17.1.2.8, “Adding Slaves to a Replication Environment”
Section 18.10, “Frequently Asked Questions”
Section 17.4.7, “Improving Replication Performance”
Section 17.2.3.4, “Replication Channel Naming Conventions”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.4.1, “The Slave Relay Log”
Section 1.4, “What Is New in MySQL 8.0”

relay_log_basename

Section 17.1.6.3, “Replication Slave Options and Variables”

relay_log_index

Section 17.1.2.8, “Adding Slaves to a Replication Environment”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.4.1, “The Slave Relay Log”

relay_log_info_file

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.4.2, “Slave Status Logs”

relay_log_info_repository

Section 17.4.2, “Handling an Unexpected Halt of a Replication Slave”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.3.8.3, “Restrictions on XA Transactions”

relay_log_purge

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”

relay_log_recovery

Section 17.4.2, “Handling an Unexpected Halt of a Replication Slave”
Section 17.1.6.3, “Replication Slave Options and Variables”

relay_log_space_limit

Section 8.14.4, “Replication Slave I/O Thread States”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.3.3, “Startup Options and Replication Channels”

report_host

Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 18.2.2, “Deploying Group Replication Locally”
Section 21.2.4, “Production Deployment of InnoDB Cluster”
Section 17.1.6.3, “Replication Slave Options and Variables”

report_password

Section 17.1.6.3, “Replication Slave Options and Variables”

report_port

Section 17.1.6.3, “Replication Slave Options and Variables”

report_user

Section 17.1.6.3, “Replication Slave Options and Variables”

require_secure_transport

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.5, “FIPS Support”
Section B.3.1, “Server Error Message Reference”
Section 5.1.8, “Server System Variables”
Section 6.3, “Using Encrypted Connections”
Section 20.5.3, “Using Secure Connections with X Plugin”

resultset_metadata

Section 28.7.27, “C API Optional Result Set Metadata”
Section 28.7.7.17, “mysql_fetch_field()”
Section 28.7.7.18, “mysql_fetch_field_direct()”
Section 28.7.7.19, “mysql_fetch_fields()”
Section 28.7.7.62, “mysql_result_metadata()”
Section 5.1.8, “Server System Variables”

rewriter_enabled

Rewriter Query Rewrite Plugin System Variables
Section 5.6.4.2, “Using the Rewriter Query Rewrite Plugin”

rewriter_verbose

Rewriter Query Rewrite Plugin System Variables

rpl_read_size

Section 8.12.3.1, “How MySQL Uses Memory”
Section 17.4.7, “Improving Replication Performance”
Section 17.1.6.3, “Replication Slave Options and Variables”

rpl_semi_sync_master_enabled

Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.4.9.1, “Semisynchronous Replication Administrative Interface”

Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”
Section 17.4.9.3, “Semisynchronous Replication Monitoring”

rpl_semi_sync_master_timeout

Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.4.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”

rpl_semi_sync_master_trace_level

Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.1.6.3, “Replication Slave Options and Variables”

rpl_semi_sync_master_wait_for_slave_count

Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.4.9, “Semisynchronous Replication”

rpl_semi_sync_master_wait_no_slave

Section 17.1.6.2, “Replication Master Options and Variables”

rpl_semi_sync_master_wait_point

Section 17.1.6.2, “Replication Master Options and Variables”
Section 17.4.9, “Semisynchronous Replication”

rpl_semi_sync_slave_enabled

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.4.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.4.9.2, “Semisynchronous Replication Installation and Configuration”

rpl_semi_sync_slave_trace_level

Section 17.1.6.3, “Replication Slave Options and Variables”

rpl_stop_slave_timeout

Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 13.4.2.7, “STOP SLAVE Syntax”

S

[index top]

schema_definition_cache

Section 14.4, “Dictionary Object Cache”
Section 5.1.8, “Server System Variables”

secondary_engine_cost_threshold

Section 5.1.8, “Server System Variables”

secure_auth

Section 5.1.8, “Server System Variables”

secure_file_priv

Section 13.2.5, “IMPORT TABLE Syntax”

Section 2.10.1, “Initializing the Data Directory”
Section 2.5.4, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 13.2.7, “LOAD DATA Syntax”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 6.4.8.4, “MySQL Enterprise Data Masking and De-Identification User-Defined Function Reference”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 6.2.2, “Privileges Provided by MySQL”
Section 15.6.5, “Redo Log”
Section 13.2.10.1, “SELECT ... INTO Syntax”
Section 5.1.8, “Server System Variables”
Section 12.5, “String Functions and Operators”
Section 6.4.8.3, “Using MySQL Enterprise Data Masking and De-Identification”

server_id

Adding a Second Instance
Section 6.4.5.4, “Audit Log File Formats”
Section 17.1.6.4, “Binary Logging Options and Variables”
Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 21.2.2, “InnoDB Cluster Requirements”
Section 12.24, “Miscellaneous Functions”
MySQL Server Options for NDB Cluster
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 22.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster System Variables
Section 26.12.11, “Performance Schema Replication Tables”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 5.1.8, “Server System Variables”
Section 17.1.2.1, “Setting the Replication Master Configuration”
Section 17.1.3.4, “Setting Up Replication Using GTIDs”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 5.4.4, “The Binary Log”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”
Section 21.4, “Working with InnoDB Cluster”

server_id_bits

MySQL Server Options for NDB Cluster
NDB Cluster System Variables

server_uuid

Section 17.3.2.2, “Binary Log Encryption Keys”
Section 18.4.1.2, “Changing a Group's Mode”
Section 18.4.1.1, “Changing a Group's Primary Member”
Section 13.4.3.3, “Function which Configures Group Replication Primary”
Section 17.1.3.1, “GTID Format and Storage”
Section 26.12.11, “Performance Schema Replication Tables”
Primary Election Algorithm
Section 21.2.4, “Production Deployment of InnoDB Cluster”
Section 17.1.6, “Replication and Binary Logging Options and Variables”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”
Section 13.4.2.6, “START SLAVE Syntax”
Section 17.1.3.7, “Stored Function Examples to Manipulate GTIDs”
Section 26.12.18.3, “The log_status Table”
Section 26.12.11.2, “The replication_connection_status Table”

Section 18.3.2, “The replication_group_members Table”
Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”
Section 21.3, “Using MySQL Router with InnoDB Cluster”
Section 21.4, “Working with InnoDB Cluster”

session_track_gtids

Section 17.1.6.5, “Global Transaction ID Options and Variables”
Section 28.7.7.69, “mysql_session_track_get_first()”
Section 17.1.5.1, “Replication Mode Concepts”
Section 5.1.8, “Server System Variables”
Section 5.1.15, “Server Tracking of Client Session State Changes”

session_track_schema

Section 28.7.7.69, “mysql_session_track_get_first()”
Section 5.1.8, “Server System Variables”
Section 5.1.15, “Server Tracking of Client Session State Changes”

session_track_state_change

Section 28.7.7.69, “mysql_session_track_get_first()”
Section 5.1.8, “Server System Variables”
Section 5.1.15, “Server Tracking of Client Session State Changes”

session_track_system_variables

Section 28.7.7.69, “mysql_session_track_get_first()”
Section 5.1.8, “Server System Variables”
Section 5.1.15, “Server Tracking of Client Session State Changes”

session_track_transaction_info

Section 28.7.7.69, “mysql_session_track_get_first()”
Section 5.1.8, “Server System Variables”
Section 5.1.15, “Server Tracking of Client Session State Changes”

sha

Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”
Section 6.2.18, “Proxy Users”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 6.4.1.3, “SHA-256 Pluggable Authentication”

shared_memory

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.5, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — Client for Administering a MySQL Server”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.7, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.8, “`mysqlslap` — Load Emulation Client”
Section B.3.1, “Server Error Message Reference”

Section 5.1.8, “Server System Variables”
Section 5.8.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.3.4.5, “Starting the Server for the First Time”
Section 1.3.2, “The Main Features of MySQL”
Type and Networking

shared_memory_base_name

Section 28.7.7.50, “mysql_options()”
Section 5.1.8, “Server System Variables”
Section 5.8.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”

show_compatibility_

Section 5.1.8, “Server System Variables”

show_create_table_skip_secondary_engine

Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 5.1.8, “Server System Variables”

show_create_table_verbosity

Section 5.1.8, “Server System Variables”

show_old_temporals

Section 5.1.8, “Server System Variables”

skip_external_locking

Section 8.11.5, “External Locking”
Section 5.1.8, “Server System Variables”

skip_name_resolve

Section 5.1.8, “Server System Variables”

skip_networking

Section 5.1.8, “Server System Variables”
Section 20.5.6.1, “Status Variables for X Plugin”

skip_show_database

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

slave

Section 17.2.3.3, “Startup Options and Replication Channels”

slave_allow_batching

NDB Cluster System Variables
Section 22.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”

slave_checkpoint_group

Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 12.24, “Miscellaneous Functions”
Section 17.1.6.3, “Replication Slave Options and Variables”

slave_checkpoint_period

Section 12.24, “Miscellaneous Functions”
Section 17.1.6.3, “Replication Slave Options and Variables”

slave_compressed_protocol

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 4.2.6, “Connection Compression Control”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 1.4, “What Is New in MySQL 8.0”

slave_exec_mode

Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 17.5.1.21, “Replication and MEMORY Tables”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.1.2, “Usage of Row-Based Logging and Replication”

slave_load_tmpdir

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.1.8, “Server System Variables”

slave_max_allowed_packet

Section 18.9.2, “Group Replication Limitations”
Section 18.8, “Group Replication System Variables”
Section 18.6.4, “Message Fragmentation”
Section 17.5.1.20, “Replication and max_allowed_packet”
Section 17.1.6.3, “Replication Slave Options and Variables”

slave_net_timeout

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 17.1.7.1, “Checking Replication Status”
Section 17.5.1.27, “Replication and Master or Slave Shutdowns”
Section 8.14.4, “Replication Slave I/O Thread States”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.1.8, “Server System Variables”

slave_parallel_type

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 18.9.1, “Group Replication Requirements”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.1.6.3, “Replication Slave Options and Variables”

slave_parallel_workers

Section 13.4.2.1, “CHANGE MASTER TO Syntax”
Section 18.9.1, “Group Replication Requirements”
Section 17.1.3.2, “GTID Life Cycle”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 26.12.11, “Performance Schema Replication Tables”
Section 17.5.1.20, “Replication and max_allowed_packet”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 8.14.5, “Replication Slave SQL Thread States”
Section 13.4.2.7, “STOP SLAVE Syntax”

Section 26.12.11.6, “The replication_applier_status_by_worker Table”

slave_pending_jobs_size_max

Section 8.12.3.1, “How MySQL Uses Memory”
Section 17.5.1.20, “Replication and max_allowed_packet”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 8.14.5, “Replication Slave SQL Thread States”

slave_preserve_commit_order

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 18.9.1, “Group Replication Requirements”
Section 17.1.3.2, “GTID Life Cycle”
Section 17.5.1.33, “Replication and Transaction Inconsistencies”
Section 8.14.4, “Replication Slave I/O Thread States”
Section 17.1.6.3, “Replication Slave Options and Variables”

slave_rows_search_algorithms

Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.1.10, “Server Status Variables”

slave_skip_errors

Section 17.1.6.3, “Replication Slave Options and Variables”

slave_sql_verify_checksum

MySQL Glossary
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.4.4, “The Binary Log”

slave_transaction_retries

Section 17.5.1.31, “Replication Retries and Timeouts”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.2.3.3, “Startup Options and Replication Channels”
Section 26.12.11.4, “The replication_applier_status Table”

slave_type_conversions

Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 17.1.6.3, “Replication Slave Options and Variables”

slow_launch_time

Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

slow_query_log

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

slow_query_log_file

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

Section 1.4, “What Is New in MySQL 8.0”

socket

Section 5.1.8, “Server System Variables”
Section 20.5.5.2, “X Plugin Options and System Variables”

sort_buffer_size

Section 7.6.3, “How to Repair MyISAM Tables”
Section 8.2.1.16, “ORDER BY Optimization”
Section 5.1.7, “Server Command Options”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 13.7.6.1, “SET Syntax for Variable Assignment”

sql_auto_is_null

Section 12.3.2, “Comparison Functions and Operators”
Section 13.1.20, “CREATE TABLE Syntax”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”
Section 5.4.4, “The Binary Log”

sql_big_selects

Section 5.1.8, “Server System Variables”

sql_buffer_result

Section 5.1.8, “Server System Variables”

sql_log_bin

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 22.1.7.8, “Issues Exclusive to NDB Cluster”
Section 22.6.3, “Known Issues in NDB Cluster Replication”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 22.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.3, “SET sql_log_bin Syntax”
Section 5.1.9.1, “System Variable Privileges”
Section 27.4.4.2, “The diagnostics() Procedure”
Section 27.4.4.12, “The ps_setup_reload_saved() Procedure”
Section 27.4.4.14, “The ps_setup_save() Procedure”
Section 27.4.4.22, “The ps_trace_statement_digest() Procedure”
Section 27.4.4.23, “The ps_trace_thread() Procedure”
Section 27.4.4.25, “The statement_performance_analyzer() Procedure”
Section 17.5.3, “Upgrading a Replication Setup”
Section 1.4, “What Is New in MySQL 8.0”

sql_log_off

MySQL Glossary
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.8, “Server System Variables”
Section 5.4.3, “The General Query Log”

SQL_MODE

Section 15.12.1, “Online DDL Operations”

sql_mode

Section 15.1.2, “Best Practices for InnoDB Tables”
Section 2.11.4, “Changes in MySQL 8.0”
Section 13.1.13, “CREATE EVENT Syntax”
Section 13.1.17, “CREATE PROCEDURE and CREATE FUNCTION Syntax”
Section 13.1.22, “CREATE TRIGGER Syntax”
Section 12.25.3, “Expression Handling”
Section 1.7, “How to Report Bugs or Problems”
Section 13.2.7, “LOAD DATA Syntax”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 1.8, “MySQL Standards Compliance”
Section 2.11.5, “Preparing Your Installation for Upgrade”
Section B.4.4.2, “Problems Using DATE Columns”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.11, “Server SQL Modes”
Section 5.1.8, “Server System Variables”
Section 13.7.7.13, “SHOW CREATE VIEW Syntax”
Section 13.6.7.5, “SIGNAL Syntax”
Section 5.4.4, “The Binary Log”
Section 25.36, “The INFORMATION_SCHEMA VIEWS Table”
Section 27.4.5.7, “The list_add() Function”
Section 4.2.2.2, “Using Option Files”
Section 5.1.9, “Using System Variables”

sql_notes

Section B.2, “Error Information Interfaces”
Section 5.1.8, “Server System Variables”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”

sql_quote_show_create

Section 12.15, “Information Functions”
Section 5.1.8, “Server System Variables”
Section 13.7.7.6, “SHOW CREATE DATABASE Syntax”
Section 13.7.7.10, “SHOW CREATE TABLE Syntax”

sql_require_primary_key

Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 5.1.8, “Server System Variables”
Section 18.4.6, “Using MySQL Enterprise Backup with Group Replication”

sql_safe_updates

Section 4.5.1.6, “mysql Client Tips”
Section 8.2.1.2, “Range Optimization”
Section 5.1.8, “Server System Variables”

sql_select_limit

Section 4.5.1.6, “mysql Client Tips”
Section 5.1.8, “Server System Variables”

sql_slave_skip_counter

Section 17.1.5.1, “Replication Mode Concepts”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 17.1.3.6, “Restrictions on Replication with GTIDs”
Section 13.7.7.34, “SHOW SLAVE STATUS Syntax”

sql_warnings

Section 5.1.8, “Server System Variables”

ssl_ca

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 20.5.5.2, “X Plugin Options and System Variables”

ssl_capath

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 20.5.5.2, “X Plugin Options and System Variables”

ssl_cert

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 20.5.5.2, “X Plugin Options and System Variables”

ssl_cipher

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 20.5.5.2, “X Plugin Options and System Variables”

ssl_crl

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 20.5.5.2, “X Plugin Options and System Variables”

ssl_crlpath

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

Section 20.5.5.2, “X Plugin Options and System Variables”

ssl_fips_mode

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.5, “FIPS Support”
Section 5.1.8, “Server System Variables”

ssl_key

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.3.1, “Creating SSL and RSA Certificates and Keys using MySQL”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 20.5.5.2, “X Plugin Options and System Variables”

storage_engine

Section 13.1.16, “CREATE LOGFILE GROUP Syntax”

stored_program_cache

Section 8.10.3, “Caching of Prepared Statements and Stored Programs”
Section 14.4, “Dictionary Object Cache”
Section 5.1.8, “Server System Variables”

stored_program_definition_cache

Section 14.4, “Dictionary Object Cache”
Section 5.1.8, “Server System Variables”

super

Section 18.1.3.1, “Single-Primary Mode”

super_read_only

Adding a Second Instance
Section 21.2.7, “Adopting a Group Replication Deployment”
Section 18.6.6.4, “Exit Action”
Section 18.7.3.3, “Group Replication Online Upgrade Methods”
Section 18.3.1, “Group Replication Server States”
Section 18.8, “Group Replication System Variables”
Section 18.7.1.1, “Member Versions During Upgrades”
Section 18.7.3.1, “Online Upgrade Considerations”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 21.2.4, “Production Deployment of InnoDB Cluster”
Section 21.2.6, “Sandbox Deployment of InnoDB Cluster”
Section 5.1.8, “Server System Variables”
Section 13.4.3.1, “START GROUP_REPLICATION Syntax”
Section 13.4.3.2, “STOP GROUP_REPLICATION Syntax”
Section 18.7.3.2, “Upgrading a Group Replication Member”
Section 21.4, “Working with InnoDB Cluster”

sync_binlog

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 17.1.3.3, “GTID Auto-Positioning”
Section 17.4.2, “Handling an Unexpected Halt of a Replication Slave”

Section 15.2, “InnoDB and the ACID Model”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 17.5.1.27, “Replication and Master or Slave Shutdowns”
Section 5.4.4, “The Binary Log”

sync_master_info

Section 17.1.6.3, “Replication Slave Options and Variables”

sync_relay_log

Section 17.4.2, “Handling an Unexpected Halt of a Replication Slave”
Section 17.1.6.3, “Replication Slave Options and Variables”

sync_relay_log_info

Section 17.5.1.27, “Replication and Master or Slave Shutdowns”
Section 17.1.6.3, “Replication Slave Options and Variables”

syseventlog

Section 5.4.2.7, “Error Logging to the System Log”
Section 5.1.8, “Server System Variables”

system_time_zone

Section 5.1.13, “MySQL Server Time Zone Support”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”

T

[index top]

table_definition_cache

Section 14.4, “Dictionary Object Cache”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 5.1.8, “Server System Variables”

table_encryption_privilege_check

Section 13.1.2, “ALTER DATABASE Syntax”
Section 13.1.9, “ALTER TABLE Syntax”
Section 13.1.10, “ALTER TABLESPACE Syntax”
Section 13.1.12, “CREATE DATABASE Syntax”
Section 13.1.20, “CREATE TABLE Syntax”
Section 13.1.21, “CREATE TABLESPACE Syntax”
Section 15.6.3.9, “InnoDB Data-at-Rest Encryption”
Section 17.3.3.1, “Privileges For The Replication PRIVILEGE_CHECKS_USER Account”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.1.36, “RENAME TABLE Syntax”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

table_open_cache

Section B.4.2.17, “File Not Found and Similar Errors”
Section 8.14.2, “General Thread States”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”

Section 8.12.3.1, “How MySQL Uses Memory”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

table_open_cache_instances

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

tablespace_definition_cache

Section 14.4, “Dictionary Object Cache”
Section 5.1.8, “Server System Variables”

temptable_max_ram

Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

temptable_use_mmap

Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.8, “Server System Variables”
Section 1.4, “What Is New in MySQL 8.0”

thread_cache_size

Section 29.5.1.4, “Debugging mysqld under gdb”
Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

thread_handling

Section 5.1.8, “Server System Variables”
Section 5.6.3.1, “Thread Pool Components”

thread_pool_algorithm

Section 5.1.8, “Server System Variables”
Section 26.12.15.1, “The tp_thread_group_state Table”
Section 5.6.3.1, “Thread Pool Components”

thread_pool_high_priority_connection

Section 5.1.8, “Server System Variables”
Section 5.6.3.1, “Thread Pool Components”
Section 5.6.3.3, “Thread Pool Operation”

thread_pool_max_active_query_threads

Section 5.1.8, “Server System Variables”
Section 5.6.3.1, “Thread Pool Components”
Section 5.6.3.3, “Thread Pool Operation”

thread_pool_max_unused_threads

Section 5.1.8, “Server System Variables”

Section 5.6.3.1, “Thread Pool Components”

thread_pool_prio_kickup_timer

Section 5.1.8, “Server System Variables”
Section 26.12.15.1, “The tp_thread_group_state Table”
Section 26.12.15.2, “The tp_thread_group_stats Table”
Section 5.6.3.1, “Thread Pool Components”
Section 5.6.3.3, “Thread Pool Operation”
Section 5.6.3.4, “Thread Pool Tuning”

thread_pool_size

Section 5.1.8, “Server System Variables”
Section 5.6.3.1, “Thread Pool Components”
Section 5.6.3.3, “Thread Pool Operation”
Section 5.6.3.4, “Thread Pool Tuning”

thread_pool_stall_limit

Section 5.1.8, “Server System Variables”
Section 26.12.15.1, “The tp_thread_group_state Table”
Section 26.12.15.2, “The tp_thread_group_stats Table”
Section 5.6.3.1, “Thread Pool Components”
Section 5.6.3.3, “Thread Pool Operation”
Section 5.6.3.4, “Thread Pool Tuning”

thread_stack

Section 8.12.4.1, “How MySQL Handles Client Connections”
Section 8.12.3.1, “How MySQL Uses Memory”
Section 5.1.8, “Server System Variables”
Section 24.2.1, “Stored Routine Syntax”

time_format

Section 5.1.8, “Server System Variables”

time_zone

Section 13.1.13, “CREATE EVENT Syntax”
Section 12.7, “Date and Time Functions”
Section 24.4.4, “Event Metadata”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 28.7.7.69, “mysql_session_track_get_first()”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 11.3.1, “The DATE, DATETIME, and TIMESTAMP Types”
Section 5.4.3, “The General Query Log”
Section 5.4.5, “The Slow Query Log”
Section 1.4, “What Is New in MySQL 8.0”

timestamp

Section 16.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.38, “Replication and Variables”

Section 5.1.8, “Server System Variables”

tls_ciphersuites

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

tls_version

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 18.5.2, “Group Replication Secure Socket Layer (SSL) Support”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”
Section 17.3.1, “Setting Up Replication to Use Encrypted Connections”
Section 20.5.3, “Using Secure Connections with X Plugin”

tmp_table_size

Section 8.12.3.1, “How MySQL Uses Memory”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 13.6.6.5, “Restrictions on Server-Side Cursors”
Section 5.1.10, “Server Status Variables”
Section 5.1.8, “Server System Variables”

tmpdir

Section 17.4.1.2, “Backing Up Raw Data from a Slave”
Section B.4.2.12, “Can't create/write to file”
Section 7.2, “Database Backup Methods”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 15.12.5, “Online DDL Failure Conditions”
Section 15.12.3, “Online DDL Space Requirements”
Section 8.2.1.16, “ORDER BY Optimization”
Section 17.1.6.3, “Replication Slave Options and Variables”
Section 5.1.8, “Server System Variables”

transaction

Section 18.9.1, “Group Replication Requirements”

transaction_alloc_block_size

Section 5.1.8, “Server System Variables”

transaction_allow_batching

NDB Cluster System Variables

transaction_isolation

Section 28.7.7.69, “mysql_session_track_get_first()”

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 5.1.15, “Server Tracking of Client Session State Changes”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 1.4, “What Is New in MySQL 8.0”

transaction_prealloc_size

Section 5.1.8, “Server System Variables”

transaction_read_only

Section 28.7.7.69, “mysql_session_track_get_first()”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 5.1.15, “Server Tracking of Client Session State Changes”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 1.4, “What Is New in MySQL 8.0”

transaction_write_set_extraction

Section 17.1.6.4, “Binary Logging Options and Variables”
Section 18.2.1.2, “Configuring an Instance for Group Replication”
Section 18.8, “Group Replication System Variables”
Section B.3.1, “Server Error Message Reference”

tx_isolation

Section 5.1.8, “Server System Variables”

tx_read_only

Section 5.1.8, “Server System Variables”

U

[index top]

unique_checks

Section 15.6.1.3, “Converting Tables from MyISAM to InnoDB”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.5.1.38, “Replication and Variables”
Section 5.1.8, “Server System Variables”
Section 5.4.4, “The Binary Log”

updatable_views_with_limit

Section 5.1.8, “Server System Variables”
Section 24.5.3, “Updatable and Insertable Views”

use_secondary_engine

Section 5.1.8, “Server System Variables”

V

[index top]

validate_password
Section 12.13, “Encryption and Compression Functions”
Section 6.4.3.2, “Password Validation Options and Variables”
Section 6.4.3, “The Password Validation Component”

validate_password_check_user_name
Section 6.4.3.2, “Password Validation Options and Variables”

validate_password_dictionary_file
Section 6.4.3.2, “Password Validation Options and Variables”

validate_password_length
Section 6.4.3.2, “Password Validation Options and Variables”

validate_password_mixed_case_count
Section 6.4.3.2, “Password Validation Options and Variables”

validate_password_number_count
Section 6.4.3.2, “Password Validation Options and Variables”

validate_password_policy
Section 6.4.3.2, “Password Validation Options and Variables”

validate_password_special_char_count
Section 6.4.3.2, “Password Validation Options and Variables”

validate_user_plugins
Section 5.1.8, “Server System Variables”

version
Section 6.4.5.4, “Audit Log File Formats”
Section 12.15, “Information Functions”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 5.1.8, “Server System Variables”

version_comment
Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.8, “Server System Variables”
Section 13.7.7.39, “SHOW VARIABLES Syntax”

version_compile_machine
Section 5.1.8, “Server System Variables”

version_compile_os
Section 5.1.8, “Server System Variables”

version_compile_zlib
Section 5.1.8, “Server System Variables”

version_tokens_session
Section B.3.1, “Server Error Message Reference”

Section 5.6.6.3, “Using Version Tokens”
Section 5.6.6.4, “Version Tokens Reference”

version_tokens_session_number

Section 5.6.6.4, “Version Tokens Reference”

W

[index top]

wait_timeout

Section B.4.2.10, “Communication Errors and Aborted Connections”
Section 18.4.2.2, “Configuring Transaction Consistency Guarantees”
Section B.4.2.8, “MySQL server has gone away”
Section 28.7.7.54, “mysql_real_connect()”
Section 5.1.8, “Server System Variables”

warning_count

Section B.2, “Error Information Interfaces”
Section 13.5, “Prepared SQL Statement Syntax”
Section 5.1.8, “Server System Variables”
Section 13.7.7.17, “SHOW ERRORS Syntax”
Section 13.7.7.40, “SHOW WARNINGS Syntax”
Section 13.6.7.5, “SIGNAL Syntax”
Section 13.6.7.7, “The MySQL Diagnostics Area”

windowing_use_high_precision

Section 5.1.8, “Server System Variables”
Section 8.2.1.21, “Window Function Optimization”

Transaction Isolation Level Index

R | S

R

[index top]

READ COMMITTED

Section 15.7.2.3, “Consistent Nonlocking Reads”
Section 22.1.6.1, “Differences Between the NDB and InnoDB Storage Engines”
Section 18.9.2, “Group Replication Limitations”
Section 15.7.5.3, “How to Minimize and Handle Deadlocks”
Section 15.7.1, “InnoDB Locking”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section A.1, “MySQL 8.0 FAQ: General”
Section 22.1.6.3, “NDB and InnoDB Feature Usage Summary”
Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 5.4.4.2, “Setting The Binary Log Format”

Section 26.12.7.1, “The events_transactions_current Table”
Section 15.7.2.1, “Transaction Isolation Levels”
Section 13.1.37, “TRUNCATE TABLE Syntax”
Section 1.4, “What Is New in MySQL 8.0”

READ UNCOMMITTED

Section 15.7.2.3, “Consistent Nonlocking Reads”
Including Delete-marked Records in Persistent Statistics Calculations
Section 15.19.2, “InnoDB memcached Architecture”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 15.19.6.6, “Performing DML and DDL Statements on the Underlying InnoDB Table”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 26.12.7.1, “The events_transactions_current Table”
Section 15.7.2.1, “Transaction Isolation Levels”
Section 13.1.37, “TRUNCATE TABLE Syntax”

READ-COMMITTED

Section 5.1.7, “Server Command Options”
Section 13.3.7, “SET TRANSACTION Syntax”

READ-UNCOMMITTED

Section 5.1.7, “Server Command Options”
Section 13.3.7, “SET TRANSACTION Syntax”

REPEATABLE READ

Section 15.7.2.3, “Consistent Nonlocking Reads”
Section 15.19.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 18.9.2, “Group Replication Limitations”
Section 15.7.1, “InnoDB Locking”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.6, “`mysqlpump` — A Database Backup Program”
Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 26.12.7, “Performance Schema Transaction Tables”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 26.12.7.1, “The events_transactions_current Table”
Section 15.7.2.1, “Transaction Isolation Levels”
Section 13.3.8, “XA Transactions”

REPEATABLE-READ

Section 5.1.7, “Server Command Options”
Section 5.1.8, “Server System Variables”
Section 13.3.7, “SET TRANSACTION Syntax”

S

[index top]

SERIALIZABLE

Section 15.7.2.3, “Consistent Nonlocking Reads”
Section 18.9.2, “Group Replication Limitations”
Section 15.7.1, “InnoDB Locking”
Section 15.13, “InnoDB Startup Options and System Variables”
Section 22.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 15.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 26.12.7, “Performance Schema Transaction Tables”
Section 5.1.7, “Server Command Options”
Section 13.3.7, “SET TRANSACTION Syntax”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Syntax”
Section 26.12.7.1, “The events_transactions_current Table”
Section 15.7.2.1, “Transaction Isolation Levels”
Section 13.3.8, “XA Transactions”

MySQL Glossary

These terms are commonly used in information about the MySQL database server. This glossary originated as a reference for terminology about the InnoDB storage engine, and the majority of definitions are InnoDB-related.

A

.ARM file
: Metadata for `ARCHIVE` tables. Contrast with **.ARZ file**. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also .ARZ file, MySQL Enterprise Backup, mysqlbackup command.

.ARZ file
: Data for ARCHIVE tables. Contrast with **.ARM file**. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also .ARM file, MySQL Enterprise Backup, mysqlbackup command.

ACID
: An acronym standing for atomicity, consistency, isolation, and durability. These properties are all desirable in a database system, and are all closely tied to the notion of a **transaction**. The transactional features of `InnoDB` adhere to the ACID principles.

  Transactions are **atomic** units of work that can be **committed** or **rolled back**. When a transaction makes multiple changes to the database, either all the changes succeed when the transaction is committed, or all the changes are undone when the transaction is rolled back.

  The database remains in a consistent state at all times — after each commit or rollback, and while transactions are in progress. If related data is being updated across multiple tables, queries see either all old values or all new values, not a mix of old and new values.

  Transactions are protected (isolated) from each other while they are in progress; they cannot interfere with each other or see each other's uncommitted data. This isolation is achieved through the **locking** mechanism. Experienced users can adjust the **isolation level**, trading off less protection in favor of increased performance and **concurrency**, when they can be sure that the transactions really do not interfere with each other.

  The results of transactions are durable: once a commit operation succeeds, the changes made by that transaction are safe from power failures, system crashes, race conditions, or other potential dangers that many non-database applications are vulnerable to. Durability typically involves writing to disk storage, with a certain amount of redundancy to protect against power failures or software crashes during write operations. (In `InnoDB`, the **doublewrite buffer** assists with durability.)
  See Also atomic, commit, concurrency, doublewrite buffer, isolation level, locking, rollback, transaction.

adaptive flushing
: An algorithm for **InnoDB** tables that smooths out the I/O overhead introduced by **checkpoints**. Instead of **flushing** all modified **pages** from the **buffer pool** to the **data files** at once, MySQL periodically flushes small sets of modified pages. The adaptive flushing algorithm extends this process by estimating the optimal rate to perform these periodic flushes, based on the rate of flushing and how fast **redo** information is generated.
See Also buffer pool, checkpoint, data files, flush, InnoDB, page, redo log.

adaptive hash index
: An optimization for `InnoDB` tables that can speed up lookups using `=` and `IN` operators, by constructing a **hash index** in memory. MySQL monitors index searches for `InnoDB` tables, and if queries could benefit from a hash index, it builds one automatically for index **pages** that are frequently accessed. In a sense, the adaptive hash index configures MySQL at runtime to take advantage of ample main memory, coming closer to the architecture

of main-memory databases. This feature is controlled by the `innodb_adaptive_hash_index` configuration option. Because this feature benefits some workloads and not others, and the memory used for the hash index is reserved in the **buffer pool**, typically you should benchmark with this feature both enabled and disabled.

The hash index is always built based on an existing **B-tree** index on the table. MySQL can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches against the index. A hash index can be partial; the whole B-tree index does not need to be cached in the buffer pool.

In MySQL 5.6 and higher, another way to take advantage of fast single-value lookups with `InnoDB` tables is to use the `InnoDB` **memcached** plugin. See Section 15.19, “InnoDB memcached Plugin” for details.
See Also B-tree, buffer pool, hash index, memcached, page, secondary index.

AIO
Acronym for **asynchronous I/O**. You might see this acronym in `InnoDB` messages or keywords.
See Also asynchronous I/O.

application programming interface (API)
A set of functions or procedures. An API provides a stable set of names and types for functions, procedures, parameters, and return values.

apply
When a backup produced by the **MySQL Enterprise Backup** product does not include the most recent changes that occurred while the backup was underway, the process of updating the backup files to include those changes is known as the **apply** step. It is specified by the `apply-log` option of the `mysqlbackup` command.

Before the changes are applied, we refer to the files as a **raw backup**. After the changes are applied, we refer to the files as a **prepared backup**. The changes are recorded in the **ibbackup_logfile** file; once the apply step is finished, this file is no longer necessary.
See Also hot backup, ibbackup_logfile, MySQL Enterprise Backup, prepared backup, raw backup.

asynchronous I/O
A type of I/O operation that allows other processing to proceed before the I/O is completed. Also known as **nonblocking I/O** and abbreviated as **AIO**. `InnoDB` uses this type of I/O for certain operations that can run in parallel without affecting the reliability of the database, such as reading pages into the **buffer pool** that have not actually been requested, but might be needed soon.

Historically, `InnoDB` used asynchronous I/O on Windows systems only. Starting with the InnoDB Plugin 1.1 and MySQL 5.5, `InnoDB` uses asynchronous I/O on Linux systems. This change introduces a dependency on `libaio`. Asynchronous I/O on Linux systems is configured using the `innodb_use_native_aio` option, which is enabled by default. On other Unix-like systems, InnoDB uses synchronous I/O only.
See Also buffer pool, nonblocking I/O.

atomic
In the SQL context, **transactions** are units of work that either succeed entirely (when **committed**) or have no effect at all (when **rolled back**). The indivisible ("atomic") property of transactions is the “A” in the acronym **ACID**.
See Also ACID, commit, rollback, transaction.

atomic DDL
An atomic *DDL* statement is one that combines the *data dictionary* updates, *storage engine* operations, and *binary log* writes associated with a DDL operation into a single, atomic transaction. The transaction is either fully committed or rolled back, even if the server halts during the operation. Atomic DDL support was added in MySQL 8.0. For more information, see Section 13.1.1, “Atomic Data Definition Statement Support”.
See Also binary log, data dictionary, DDL, storage engine.

atomic instruction
Special instructions provided by the CPU, to ensure that critical low-level operations cannot be interrupted.

auto-increment
: A property of a table column (specified by the `AUTO_INCREMENT` keyword) that automatically adds an ascending sequence of values in the column.

  It saves work for the developer, not to have to produce new unique values when inserting new rows. It provides useful information for the query optimizer, because the column is known to be not null and with unique values. The values from such a column can be used as lookup keys in various contexts, and because they are auto-generated there is no reason to ever change them; for this reason, primary key columns are often specified as auto-incrementing.

  Auto-increment columns can be problematic with statement-based replication, because replaying the statements on a slave might not produce the same set of column values as on the master, due to timing issues. When you have an auto-incrementing primary key, you can use statement-based replication only with the setting `innodb_autoinc_lock_mode=1`. If you have `innodb_autoinc_lock_mode=2`, which allows higher concurrency for insert operations, use **row-based replication** rather than **statement-based replication**. The setting `innodb_autoinc_lock_mode=0` should not be used except for compatibility purposes.

  Consecutive lock mode (`innodb_autoinc_lock_mode=1`) is the default setting prior to MySQL 8.0.3. As of MySQL 8.0.3, interleaved lock mode (`innodb_autoinc_lock_mode=2`) is the default, which reflects the change from statement-based to row-based replication as the default replication type.
  See Also auto-increment locking, innodb_autoinc_lock_mode, primary key, row-based replication, statement-based replication.

auto-increment locking
: The convenience of an **auto-increment** primary key involves some tradeoff with concurrency. In the simplest case, if one transaction is inserting values into the table, any other transactions must wait to do their own inserts into that table, so that rows inserted by the first transaction receive consecutive primary key values. `InnoDB` includes optimizations and the `innodb_autoinc_lock_mode` option so that you can configure and optimal balance between predictable sequences of auto-increment values and maximum **concurrency** for insert operations.
  See Also auto-increment, concurrency, innodb_autoinc_lock_mode.

autocommit
: A setting that causes a **commit** operation after each **SQL** statement. This mode is not recommended for working with `InnoDB` tables with **transactions** that span several statements. It can help performance for **read-only transactions** on `InnoDB` tables, where it minimizes overhead from **locking** and generation of **undo** data, especially in MySQL 5.6.4 and up. It is also appropriate for working with `MyISAM` tables, where transactions are not applicable.
  See Also commit, locking, read-only transaction, SQL, transaction, undo.

availability
: The ability to cope with, and if necessary recover from, failures on the host, including failures of MySQL, the operating system, or the hardware and maintenance activity that may otherwise cause downtime. Often paired with **scalability** as critical aspects of a large-scale deployment.
  See Also scalability.

B

B-tree
: A tree data structure that is popular for use in database indexes. The structure is kept sorted at all times, enabling fast lookup for exact matches (equals operator) and ranges (for example, greater than, less than, and `BETWEEN` operators). This type of index is available for most storage engines, such as `InnoDB` and `MyISAM`.

  Because B-tree nodes can have many children, a B-tree is not the same as a binary tree, which is limited to 2 children per node.

Contrast with **hash index**, which is only available in the `MEMORY` storage engine. The `MEMORY` storage engine can also use B-tree indexes, and you should choose B-tree indexes for `MEMORY` tables if some queries use range operators.

The use of the term B-tree is intended as a reference to the general class of index design. B-tree structures used by MySQL storage engines may be regarded as variants due to sophistications not present in a classic B-tree design. For related information, refer to the `InnoDB` Page Structure Fil Header section of the MySQL Internals Manual.
See Also hash index.

backticks
Identifiers within MySQL SQL statements must be quoted using the backtick character (`) if they contain special characters or reserved words. For example, to refer to a table named `FOO#BAR` or a column named `SELECT`, you would specify the identifiers as `` `FOO#BAR` `` and `` `SELECT` ``. Since the backticks provide an extra level of safety, they are used extensively in program-generated SQL statements, where the identifier names might not be known in advance.

Many other database systems use double quotation marks (") around such special names. For portability, you can enable `ANSI_QUOTES` mode in MySQL and use double quotation marks instead of backticks to qualify identifier names.
See Also SQL.

backup
The process of copying some or all table data and metadata from a MySQL instance, for safekeeping. Can also refer to the set of copied files. This is a crucial task for DBAs. The reverse of this process is the **restore** operation.

With MySQL, **physical backups** are performed by the **MySQL Enterprise Backup** product, and **logical backups** are performed by the `mysqldump` command. These techniques have different characteristics in terms of size and representation of the backup data, and speed (especially speed of the restore operation).

Backups are further classified as **hot**, **warm**, or **cold** depending on how much they interfere with normal database operation. (Hot backups have the least interference, cold backups the most.)
See Also cold backup, hot backup, logical backup, MySQL Enterprise Backup, mysqldump, physical backup, warm backup.

base column
A non-generated table column upon which a stored generated column or virtual generated column is based. In other words, a base column is a non-generated table column that is part of a generated column definition.
See Also generated column, stored generated column, virtual generated column.

beta
An early stage in the life of a software product, when it is available only for evaluation, typically without a definite release number or a number less than 1. `InnoDB` does not use the beta designation, preferring an **early adopter** phase that can extend over several point releases, leading to a **GA** release.
See Also early adopter, GA.

binary log
A file containing a record of all statements that attempt to change table data. These statements can be replayed to bring slave servers up to date in a **replication** scenario, or to bring a database up to date after restoring table data from a backup. The binary logging feature can be turned on and off, although Oracle recommends always enabling it if you use replication or perform backups.

You can examine the contents of the binary log, or replay those statements during replication or recovery, by using the `mysqlbinlog` command. For full information about the binary log, see Section 5.4.4, “The Binary Log”. For MySQL configuration options related to the binary log, see Section 17.1.6.4, “Binary Logging Options and Variables”.

For the **MySQL Enterprise Backup** product, the file name of the binary log and the current position within the file are important details. To record this information for the master server when taking a backup in a replication context, you can specify the `--slave-info` option.

Prior to MySQL 5.0, a similar capability was available, known as the update log. In MySQL 5.0 and higher, the binary log replaces the update log.
See Also binlog, MySQL Enterprise Backup, replication.

binlog
An informal name for the **binary log** file. For example, you might see this abbreviation used in e-mail messages or forum discussions.
See Also binary log.

blind query expansion
A special mode of **full-text search** enabled by the `WITH QUERY EXPANSION` clause. It performs the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. This technique is mainly applicable for short search phrases, perhaps only a single word. It can uncover relevant matches where the precise search term does not occur in the document.
See Also full-text search.

bottleneck
A portion of a system that is constrained in size or capacity, that has the effect of limiting overall throughput. For example, a memory area might be smaller than necessary; access to a single required resource might prevent multiple CPU cores from running simultaneously; or waiting for disk I/O to complete might prevent the CPU from running at full capacity. Removing bottlenecks tends to improve **concurrency**. For example, the ability to have multiple `InnoDB` **buffer pool** instances reduces contention when multiple sessions read from and write to the buffer pool simultaneously.
See Also buffer pool, concurrency.

bounce
A **shutdown** operation immediately followed by a restart. Ideally with a relatively short **warmup** period so that performance and throughput quickly return to a high level.
See Also shutdown.

buddy allocator
A mechanism for managing different-sized **pages** in the InnoDB **buffer pool**.
See Also buffer pool, page, page size.

buffer
A memory or disk area used for temporary storage. Data is buffered in memory so that it can be written to disk efficiently, with a few large I/O operations rather than many small ones. Data is buffered on disk for greater reliability, so that it can be recovered even when a **crash** or other failure occurs at the worst possible time. The main types of buffers used by InnoDB are the **buffer pool**, the **doublewrite buffer**, and the **change buffer**.
See Also buffer pool, change buffer, crash, doublewrite buffer.

buffer pool
The memory area that holds cached `InnoDB` data for both tables and indexes. For efficiency of high-volume read operations, the buffer pool is divided into **pages** that can potentially hold multiple rows. For efficiency of cache management, the buffer pool is implemented as a linked list of pages; data that is rarely used is aged out of the cache, using a variation of the **LRU** algorithm. On systems with large memory, you can improve concurrency by dividing the buffer pool into multiple **buffer pool instances**.

Several `InnoDB` status variables, `INFORMATION_SCHEMA` tables, and `performance_schema` tables help to monitor the internal workings of the buffer pool. Starting in MySQL 5.6, you can avoid a lengthy warmup period after restarting the server, particularly for instances with large buffer pools, by saving the buffer pool state at

server shutdown and restoring the buffer pool to the same state at server startup. See Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.
See Also buffer pool instance, LRU, page, warm up.

buffer pool instance
Any of the multiple regions into which the **buffer pool** can be divided, controlled by the `innodb_buffer_pool_instances` configuration option. The total memory size specified by `innodb_buffer_pool_size` is divided among all buffer pool instances. Typically, having multiple buffer pool instances is appropriate for systems that allocate multiple gigabytes to the `InnoDB` buffer pool, with each instance being one gigabyte or larger. On systems loading or looking up large amounts of data in the buffer pool from many concurrent sessions, having multiple buffer pool instances reduces contention for exclusive access to data structures that manage the buffer pool.
See Also buffer pool.

built-in
The built-in `InnoDB` storage engine within MySQL is the original form of distribution for the storage engine. Contrast with the **InnoDB Plugin**. Starting with MySQL 5.5, the InnoDB Plugin is merged back into the MySQL code base as the built-in `InnoDB` storage engine (known as InnoDB 1.1).

This distinction is important mainly in MySQL 5.1, where a feature or bug fix might apply to the InnoDB Plugin but not the built-in `InnoDB`, or vice versa.
See Also InnoDB.

business rules
The relationships and sequences of actions that form the basis of business software, used to run a commercial company. Sometimes these rules are dictated by law, other times by company policy. Careful planning ensures that the relationships encoded and enforced by the database, and the actions performed through application logic, accurately reflect the real policies of the company and can handle real-life situations.

For example, an employee leaving a company might trigger a sequence of actions from the human resources department. The human resources database might also need the flexibility to represent data about a person who has been hired, but not yet started work. Closing an account at an online service might result in data being removed from a database, or the data might be moved or flagged so that it could be recovered if the account is re-opened. A company might establish policies regarding salary maximums, minimums, and adjustments, in addition to basic sanity checks such as the salary not being a negative number. A retail database might not allow a purchase with the same serial number to be returned more than once, or might not allow credit card purchases above a certain value, while a database used to detect fraud might allow these kinds of things.
See Also relational.

C

.cfg file
A metadata file used with the `InnoDB` **transportable tablespace** feature. It is produced by the command `FLUSH TABLES ... FOR EXPORT`, puts one or more tables in a consistent state that can be copied to another server. The `.cfg` file is copied along with the corresponding **.ibd file**, and used to adjust the internal values of the `.ibd` file, such as the **space ID**, during the `ALTER TABLE ... IMPORT TABLESPACE` step.
See Also .ibd file, space ID, transportable tablespace.

cache
The general term for any memory area that stores copies of data for frequent or high-speed retrieval. In `InnoDB`, the primary kind of cache structure is the **buffer pool**.
See Also buffer, buffer pool.

cardinality
The number of different values in a table **column**. When queries refer to columns that have an associated **index**, the cardinality of each column influences which access method is most efficient. For example, for a column with

a **unique constraint**, the number of different values is equal to the number of rows in the table. If a table has a million rows but only 10 different values for a particular column, each value occurs (on average) 100,000 times. A query such as `SELECT c1 FROM t1 WHERE c1 = 50;` thus might return 1 row or a huge number of rows, and the database server might process the query differently depending on the cardinality of `c1`.

If the values in a column have a very uneven distribution, the cardinality might not be a good way to determine the best query plan. For example, `SELECT c1 FROM t1 WHERE c1 = x;` might return 1 row when `x=50` and a million rows when `x=30`. In such a case, you might need to use **index hints** to pass along advice about which lookup method is more efficient for a particular query.

Cardinality can also apply to the number of distinct values present in multiple columns, as in a **composite index**.
See Also column, composite index, index, index hint, random dive, selectivity, unique constraint.

change buffer
A special data structure that records changes to **pages** in **secondary indexes**. These values could result from SQL `INSERT`, `UPDATE`, or `DELETE` statements (**DML**). The set of features involving the change buffer is known collectively as **change buffering**, consisting of **insert buffering**, **delete buffering**, and **purge buffering**.

Changes are only recorded in the change buffer when the relevant page from the secondary index is not in the **buffer pool**. When the relevant index page is brought into the buffer pool while associated changes are still in the change buffer, the changes for that page are applied in the buffer pool (**merged**) using the data from the change buffer. Periodically, the **purge** operation that runs during times when the system is mostly idle, or during a slow shutdown, writes the new index pages to disk. The purge operation can write the disk blocks for a series of index values more efficiently than if each value were written to disk immediately.

Physically, the change buffer is part of the **system tablespace**, so that the index changes remain buffered across database restarts. The changes are only applied (**merged**) when the pages are brought into the buffer pool due to some other read operation.

The kinds and amount of data stored in the change buffer are governed by the `innodb_change_buffering` and `innodb_change_buffer_max_size` configuration options. To see information about the current data in the change buffer, issue the `SHOW ENGINE INNODB STATUS` command.

Formerly known as the **insert buffer**.
See Also buffer pool, change buffering, delete buffering, DML, insert buffer, insert buffering, merge, page, purge, purge buffering, secondary index, system tablespace.

change buffering
The general term for the features involving the **change buffer**, consisting of **insert buffering**, **delete buffering**, and **purge buffering**. Index changes resulting from SQL statements, which could normally involve random I/O operations, are held back and performed periodically by a background **thread**. This sequence of operations can write the disk blocks for a series of index values more efficiently than if each value were written to disk immediately. Controlled by the `innodb_change_buffering` and `innodb_change_buffer_max_size` configuration options.
See Also change buffer, delete buffering, insert buffering, purge buffering.

checkpoint
As changes are made to data pages that are cached in the **buffer pool**, those changes are written to the **data files** sometime later, a process known as **flushing**. The checkpoint is a record of the latest changes (represented by an **LSN** value) that have been successfully written to the data files.
See Also buffer pool, data files, flush, fuzzy checkpointing, LSN.

checksum
In `InnoDB`, a validation mechanism to detect corruption when a **page** in a **tablespace** is read from disk into the `InnoDB` **buffer pool**. This feature is controlled by the `innodb_checksums` configuration option in MySQL 5.5. `innodb_checksums` is deprecated in MySQL 5.6.3, replaced by `innodb_checksum_algorithm`.

The `innochecksum` command helps diagnose corruption problems by testing the checksum values for a specified **tablespace** file while the MySQL server is shut down.

MySQL also uses checksums for replication purposes. For details, see the configuration options `binlog_checksum`, `master_verify_checksum`, and `slave_sql_verify_checksum`.
See Also buffer pool, page, tablespace.

child table
In a **foreign key** relationship, a child table is one whose rows refer (or point) to rows in another table with an identical value for a specific column. This is the table that contains the `FOREIGN KEY ... REFERENCES` clause and optionally `ON UPDATE` and `ON DELETE` clauses. The corresponding row in the **parent table** must exist before the row can be created in the child table. The values in the child table can prevent delete or update operations on the parent table, or can cause automatic deletion or updates in the child table, based on the `ON CASCADE` option used when creating the foreign key.
See Also foreign key, parent table.

clean page
A **page** in the `InnoDB` **buffer pool** where all changes made in memory have also been written (**flushed**) to the data files. The opposite of a **dirty page**.
See Also buffer pool, data files, dirty page, flush, page.

clean shutdown
A **shutdown** that completes without errors and applies all changes to `InnoDB` tables before finishing, as opposed to a **crash** or a **fast shutdown**. Synonym for **slow shutdown**.
See Also crash, fast shutdown, shutdown, slow shutdown.

client
A type of program that sends requests to a **server**, and interprets or processes the results. The client software might run only some of the time (such as a mail or chat program), and might run interactively (such as the `mysql` command processor).
See Also mysql, server.

clustered index
The `InnoDB` term for a **primary key** index. `InnoDB` table storage is organized based on the values of the primary key columns, to speed up queries and sorts involving the primary key columns. For best performance, choose the primary key columns carefully based on the most performance-critical queries. Because modifying the columns of the clustered index is an expensive operation, choose primary columns that are rarely or never updated.

In the Oracle Database product, this type of table is known as an **index-organized table**.
See Also index, primary key, secondary index.

cold backup
A **backup** taken while the database is shut down. For busy applications and websites, this might not be practical, and you might prefer a **warm backup** or a **hot backup**.
See Also backup, hot backup, warm backup.

column
A data item within a **row**, whose storage and semantics are defined by a data type. Each **table** and **index** is largely defined by the set of columns it contains.

Each column has a **cardinality** value. A column can be the **primary key** for its table, or part of the primary key. A column can be subject to a **unique constraint**, a **NOT NULL constraint**, or both. Values in different columns, even across different tables, can be linked by a **foreign key** relationship.

In discussions of MySQL internal operations, sometimes **field** is used as a synonym.

See Also cardinality, foreign key, index, NOT NULL constraint, primary key, row, table, unique constraint.

column index
An **index** on a single column.
See Also composite index, index.

column prefix
When an **index** is created with a length specification, such as `CREATE INDEX idx ON t1 (c1(N))`, only the first N characters of the column value are stored in the index. Keeping the index prefix small makes the index compact, and the memory and disk I/O savings help performance. (Although making the index prefix too small can hinder query optimization by making rows with different values appear to the query optimizer to be duplicates.)

For columns containing binary values or long text strings, where sorting is not a major consideration and storing the entire value in the index would waste space, the index automatically uses the first N (typically 768) characters of the value to do lookups and sorts.
See Also index.

commit
A **SQL** statement that ends a **transaction**, making permanent any changes made by the transaction. It is the opposite of **rollback**, which undoes any changes made in the transaction.

`InnoDB` uses an **optimistic** mechanism for commits, so that changes can be written to the data files before the commit actually occurs. This technique makes the commit itself faster, with the tradeoff that more work is required in case of a rollback.

By default, MySQL uses the **autocommit** setting, which automatically issues a commit following each SQL statement.
See Also autocommit, optimistic, rollback, SQL, transaction.

compact row format
A **row format** for InnoDB tables. It was the default row format from MySQL 5.0.3 to MySQL 5.7.8. In MySQL 8.0, the default row format is defined by the `innodb_default_row_format` configuration option, which has a default setting of **DYNAMIC**. The **COMPACT** row format provides a more compact representation for nulls and variable-length columns than the **REDUNDANT** row format.

For additional information about `InnoDB` `COMPACT` row format, see Section 15.10, “InnoDB Row Formats”.
See Also dynamic row format, file format, redundant row format, row format.

composite index
An **index** that includes multiple columns.
See Also index.

compressed backup
The compression feature of the **MySQL Enterprise Backup** product makes a compressed copy of each tablespace, changing the extension from `.ibd` to `.ibz`. Compressing backup data allows you to keep more backups on hand, and reduces the time to transfer backups to a different server. The data is uncompressed during the restore operation. When a compressed backup operation processes a table that is already compressed, it skips the compression step for that table, because compressing again would result in little or no space savings.

A set of files produced by the **MySQL Enterprise Backup** product, where each **tablespace** is compressed. The compressed files are renamed with a `.ibz` file extension.

Applying **compression** at the start of the backup process helps to avoid storage overhead during the compression process, and to avoid network overhead when transferring the backup files to another server. The process of **applying** the **binary log** takes longer, and requires uncompressing the backup files.

See Also apply, binary log, compression, hot backup, MySQL Enterprise Backup, tablespace.

compressed row format
A **row format** that enables data and index **compression** for `InnoDB` tables. Large fields are stored away from the page that holds the rest of the row data, as in **dynamic row format**. Both index pages and the large fields are compressed, yielding memory and disk savings. Depending on the structure of the data, the decrease in memory and disk usage might or might not outweigh the performance overhead of uncompressing the data as it is used. See Section 15.9, "InnoDB Table and Page Compression" for usage details.

For additional information about `InnoDB` `COMPRESSED` row format, see DYNAMIC Row Format.
See Also compression, dynamic row format, row format.

compressed table
A table for which the data is stored in compressed form. For `InnoDB`, it is a table created with `ROW_FORMAT=COMPRESSED`. See Section 15.9, "InnoDB Table and Page Compression" for more information.
See Also compressed row format, compression.

compression
A feature with wide-ranging benefits from using less disk space, performing less I/O, and using less memory for caching.

`InnoDB` supports both table-level and page-level compression. `InnoDB` page compression is also referred to as **transparent page compression**. For more information about `InnoDB` compression, see Section 15.9, "InnoDB Table and Page Compression".

Another type of compression is the **compressed backup** feature of the **MySQL Enterprise Backup** product.
See Also buffer pool, compressed backup, compressed row format, DML, transparent page compression.

compression failure
Not actually an error, rather an expensive operation that can occur when using **compression** in combination with **DML** operations. It occurs when: updates to a compressed **page** overflow the area on the page reserved for recording modifications; the page is compressed again, with all changes applied to the table data; the re-compressed data does not fit on the original page, requiring MySQL to split the data into two new pages and compress each one separately. To check the frequency of this condition, query the `INFORMATION_SCHEMA.INNODB_CMP` table and check how much the value of the `COMPRESS_OPS` column exceeds the value of the `COMPRESS_OPS_OK` column. Ideally, compression failures do not occur often; when they do, you can adjust the `innodb_compression_level`, `innodb_compression_failure_threshold_pct`, and `innodb_compression_pad_pct_max` configuration options.
See Also compression, DML, page.

concatenated index
See composite index.

concurrency
The ability of multiple operations (in database terminology, **transactions**) to run simultaneously, without interfering with each other. Concurrency is also involved with performance, because ideally the protection for multiple simultaneous transactions works with a minimum of performance overhead, using efficient mechanisms for **locking**.
See Also ACID, locking, transaction.

configuration file
The file that holds the **option** values used by MySQL at startup. Traditionally, on Linux and Unix this file is named `my.cnf`, and on Windows it is named `my.ini`. You can set a number of options related to InnoDB under the `[mysqld]` section of the file.

See Section 4.2.2.2, "Using Option Files" for information about where MySQL searches for configuration files.

When you use the **MySQL Enterprise Backup** product, you typically use two configuration files: one that specifies where the data comes from and how it is structured (which could be the original configuration file for your server), and a stripped-down one containing only a small set of options that specify where the backup data goes and how it is structured. The configuration files used with the **MySQL Enterprise Backup** product must contain certain options that are typically left out of regular configuration files, so you might need to add options to your existing configuration file for use with **MySQL Enterprise Backup**.
See Also my.cnf, MySQL Enterprise Backup, option, option file.

consistent read
A read operation that uses **snapshot** information to present query results based on a point in time, regardless of changes performed by other transactions running at the same time. If queried data has been changed by another transaction, the original data is reconstructed based on the contents of the **undo log**. This technique avoids some of the **locking** issues that can reduce **concurrency** by forcing transactions to wait for other transactions to finish.

With **REPEATABLE READ isolation level**, the snapshot is based on the time when the first read operation is performed. With **READ COMMITTED** isolation level, the snapshot is reset to the time of each consistent read operation.

Consistent read is the default mode in which `InnoDB` processes `SELECT` statements in **READ COMMITTED** and **REPEATABLE READ** isolation levels. Because a consistent read does not set any locks on the tables it accesses, other sessions are free to modify those tables while a consistent read is being performed on the table.

For technical details about the applicable isolation levels, see Section 15.7.2.3, “Consistent Nonlocking Reads”.
See Also concurrency, isolation level, locking, READ COMMITTED, REPEATABLE READ, snapshot, transaction, undo log.

constraint
An automatic test that can block database changes to prevent data from becoming inconsistent. (In computer science terms, a kind of assertion related to an invariant condition.) Constraints are a crucial component of the **ACID** philosophy, to maintain data consistency. Constraints supported by MySQL include **FOREIGN KEY constraints** and **unique constraints**.
See Also ACID, foreign key, unique constraint.

counter
A value that is incremented by a particular kind of `InnoDB` operation. Useful for measuring how busy a server is, troubleshooting the sources of performance issues, and testing whether changes (for example, to configuration settings or indexes used by queries) have the desired low-level effects. Different kinds of counters are available through **Performance Schema** tables and **INFORMATION_SCHEMA** tables, particularly `INFORMATION_SCHEMA.INNODB_METRICS`.
See Also INFORMATION_SCHEMA, metrics counter, Performance Schema.

covering index
An **index** that includes all the columns retrieved by a query. Instead of using the index values as pointers to find the full table rows, the query returns values from the index structure, saving disk I/O. `InnoDB` can apply this optimization technique to more indexes than MyISAM can, because `InnoDB` **secondary indexes** also include the **primary key** columns. `InnoDB` cannot apply this technique for queries against tables modified by a transaction, until that transaction ends.

Any **column index** or **composite index** could act as a covering index, given the right query. Design your indexes and queries to take advantage of this optimization technique wherever possible.
See Also column index, composite index, index, primary key, secondary index.

CPU-bound
A type of **workload** where the primary **bottleneck** is CPU operations in memory. Typically involves read-intensive operations where the results can all be cached in the **buffer pool**.

See Also bottleneck, buffer pool, workload.

crash
MySQL uses the term “crash” to refer generally to any unexpected **shutdown** operation where the server cannot do its normal cleanup. For example, a crash could happen due to a hardware fault on the database server machine or storage device; a power failure; a potential data mismatch that causes the MySQL server to halt; a **fast shutdown** initiated by the DBA; or many other reasons. The robust, automatic **crash recovery** for **InnoDB** tables ensures that data is made consistent when the server is restarted, without any extra work for the DBA.
See Also crash recovery, fast shutdown, InnoDB, shutdown.

crash recovery
The cleanup activities that occur when MySQL is started again after a **crash**. For **InnoDB** tables, changes from incomplete transactions are replayed using data from the **redo log**. Changes that were **committed** before the crash, but not yet written into the **data files**, are reconstructed from the **doublewrite buffer**. When the database is shut down normally, this type of activity is performed during shutdown by the **purge** operation.

During normal operation, committed data can be stored in the **change buffer** for a period of time before being written to the data files. There is always a tradeoff between keeping the data files up-to-date, which introduces performance overhead during normal operation, and buffering the data, which can make shutdown and crash recovery take longer.
See Also change buffer, commit, crash, data files, doublewrite buffer, InnoDB, purge, redo log.

CRUD
Acronym for “create, read, update, delete”, a common sequence of operations in database applications. Often denotes a class of applications with relatively simple database usage (basic **DDL**, **DML** and **query** statements in **SQL**) that can be implemented quickly in any language.
See Also DDL, DML, query, SQL.

cursor
An internal data structure that is used to represent the result set of a **query**, or other operation that performs a search using an SQL `WHERE` clause. It works like an iterator in other high-level languages, producing each value from the result set as requested.

Although SQL usually handles the processing of cursors for you, you might delve into the inner workings when dealing with performance-critical code.
See Also query.

D

data definition language
See DDL.

data dictionary
Metadata that keeps track of database objects such as **tables**, **indexes**, and table **columns**. For the MySQL data dictionary, introduced in MySQL 8.0, metadata is physically located in `InnoDB` **file-per-table** tablespace files in the `mysql` database directory. For the `InnoDB` data dictionary, metadata is physically located in the `InnoDB` **system tablespace**.

Because the **MySQL Enterprise Backup** product always backs up the `InnoDB` system tablespace, all backups include the contents of the `InnoDB` data dictionary.
See Also column, file-per-table, .frm file, index, MySQL Enterprise Backup, system tablespace, table.

data directory
The directory under which each MySQL **instance** keeps the **data files** for `InnoDB` and the directories representing individual databases. Controlled by the `datadir` configuration option.

See Also data files, instance.

data files
The files that physically contain **table** and **index** data.

The `InnoDB` **system tablespace**, which holds the `InnoDB` **data dictionary** and is capable of holding data for multiple `InnoDB` tables, is represented by one or more `.ibdata` data files.

File-per-table tablespaces, which hold data for a single `InnoDB` table, are represented by a `.ibd` data file.

General tablespaces (introduced in MySQL 5.7.6), which can hold data for multiple `InnoDB` tables, are also represented by a `.ibd` data file.
See Also data dictionary, file-per-table, general tablespace, .ibd file, ibdata file, index, system tablespace, table, tablespace.

data manipulation language
See DML.

data warehouse
A database system or application that primarily runs large **queries**. The read-only or read-mostly data might be organized in **denormalized** form for query efficiency. Can benefit from the optimizations for **read-only transactions** in MySQL 5.6 and higher. Contrast with **OLTP**.
See Also denormalized, OLTP, query, read-only transaction.

database
Within the MySQL **data directory**, each database is represented by a separate directory. The InnoDB **system tablespace**, which can hold table data from multiple databases within a MySQL **instance**, is kept in **data files** that reside outside of individual database directories. When **file-per-table** mode is enabled, the **.ibd files** representing individual InnoDB tables are stored inside the database directories unless created elsewhere using the `DATA DIRECTORY` clause. General tablespaces, introduced in MySQL 5.7.6, also hold table data in **.ibd files**. Unlike file-per-table **.ibd files**, general tablespace **.ibd files** can hold table data from multiple databases within a MySQL **instance**, and can be assigned to directories relative to or independent of the MySQL data directory.

For long-time MySQL users, a database is a familiar notion. Users coming from an Oracle Database background will find that the MySQL meaning of a database is closer to what Oracle Database calls a **schema**.
See Also data files, file-per-table, .ibd file, instance, schema, system tablespace.

DCL
Data control language, a set of **SQL** statements for managing privileges. In MySQL, consists of the `GRANT` and `REVOKE` statements. Contrast with **DDL** and **DML**.
See Also DDL, DML, SQL.

DDL
Data definition language, a set of **SQL** statements for manipulating the database itself rather than individual table rows. Includes all forms of the `CREATE`, `ALTER`, and `DROP` statements. Also includes the `TRUNCATE` statement, because it works differently than a `DELETE FROM` *`table_name`* statement, even though the ultimate effect is similar.

DDL statements automatically **commit** the current **transaction**; they cannot be **rolled back**.

The `InnoDB` online DDL feature enhances performance for `CREATE INDEX`, `DROP INDEX`, and many types of `ALTER TABLE` operations. See Section 15.12, “InnoDB and Online DDL” for more information. Also, the `InnoDB` file-per-table setting can affect the behavior of `DROP TABLE` and `TRUNCATE TABLE` operations.

Contrast with **DML** and **DCL**.

See Also commit, DCL, DML, file-per-table, rollback, SQL, transaction.

deadlock
A situation where different **transactions** are unable to proceed, because each holds a **lock** that the other needs. Because both transactions are waiting for a resource to become available, neither will ever release the locks it holds.

A deadlock can occur when the transactions lock rows in multiple tables (through statements such as `UPDATE` or `SELECT ... FOR UPDATE`), but in the opposite order. A deadlock can also occur when such statements lock ranges of index records and **gaps**, with each transaction acquiring some locks but not others due to a timing issue.

For background information on how deadlocks are automatically detected and handled, see Section 15.7.5.2, "Deadlock Detection and Rollback". For tips on avoiding and recovering from deadlock conditions, see Section 15.7.5.3, "How to Minimize and Handle Deadlocks".
See Also gap, lock, transaction.

deadlock detection
A mechanism that automatically detects when a **deadlock** occurs, and automatically **rolls back** one of the **transactions** involved (the **victim**). Deadlock detection can be disabled using the `innodb_deadlock_detect` configuration option.
See Also deadlock, rollback, transaction, victim.

delete
When `InnoDB` processes a `DELETE` statement, the rows are immediately marked for deletion and no longer are returned by queries. The storage is reclaimed sometime later, during the periodic garbage collection known as the **purge** operation. For removing large quantities of data, related operations with their own performance characteristics are **TRUNCATE** and **DROP**.
See Also drop, purge, truncate.

delete buffering
The technique of storing changes to secondary index pages, resulting from `DELETE` operations, in the **change buffer** rather than writing the changes immediately, so that the physical writes can be performed to minimize random I/O. (Because delete operations are a two-step process, this operation buffers the write that normally marks an index record for deletion.) It is one of the types of **change buffering**; the others are **insert buffering** and **purge buffering**.
See Also change buffer, change buffering, insert buffer, insert buffering, purge buffering.

denormalized
A data storage strategy that duplicates data across different tables, rather than linking the tables with **foreign keys** and **join** queries. Typically used in **data warehouse** applications, where the data is not updated after loading. In such applications, query performance is more important than making it simple to maintain consistent data during updates. Contrast with **normalized**.
See Also data warehouse, foreign key, join, normalized.

descending index
A type of **index** where index storage is optimized to process `ORDER BY column DESC` clauses.
See Also index.

dictionary object cache
The dictionary object cache stores previously accessed **data dictionary** objects in memory to enable object reuse and minimize disk I/O. An **LRU**-based eviction strategy is used to evict least recently used objects from memory. The cache is comprised of several partitions that store different object types.

For more information, see Section 14.4, "Dictionary Object Cache".

See Also data dictionary, LRU.

dirty page
A **page** in the InnoDB **buffer pool** that has been updated in memory, where the changes are not yet written (**flushed**) to the **data files**. The opposite of a **clean page**.
See Also buffer pool, clean page, data files, flush, page.

dirty read
An operation that retrieves unreliable data, data that was updated by another transaction but not yet **committed**. It is only possible with the **isolation level** known as **read uncommitted**.

This kind of operation does not adhere to the **ACID** principle of database design. It is considered very risky, because the data could be **rolled back**, or updated further before being committed; then, the transaction doing the dirty read would be using data that was never confirmed as accurate.

Its opposite is **consistent read**, where InnoDB ensures that a transaction does not read information updated by another transaction, even if the other transaction commits in the meantime.
See Also ACID, commit, consistent read, isolation level, READ UNCOMMITTED, rollback.

disk-based
A kind of database that primarily organizes data on disk storage (hard drives or equivalent). Data is brought back and forth between disk and memory to be operated upon. It is the opposite of an **in-memory database**. Although InnoDB is disk-based, it also contains features such as he **buffer pool**, multiple buffer pool instances, and the **adaptive hash index** that allow certain kinds of workloads to work primarily from memory.
See Also adaptive hash index, buffer pool, in-memory database.

disk-bound
A type of **workload** where the primary **bottleneck** is disk I/O. (Also known as **I/O-bound**.) Typically involves frequent writes to disk, or random reads of more data than can fit into the **buffer pool**.
See Also bottleneck, buffer pool, workload.

DML
Data manipulation language, a set of **SQL** statements for performing INSERT, UPDATE, and DELETE operations. The SELECT statement is sometimes considered as a DML statement, because the SELECT ... FOR UPDATE form is subject to the same considerations for **locking** as INSERT, UPDATE, and DELETE.

DML statements for an InnoDB table operate in the context of a **transaction**, so their effects can be **committed** or **rolled back** as a single unit.

Contrast with **DDL** and **DCL**.
See Also commit, DCL, DDL, locking, rollback, SQL, transaction.

document id
In the InnoDB **full-text search** feature, a special column in the table containing the **FULLTEXT index**, to uniquely identify the document associated with each **ilist** value. Its name is FTS_DOC_ID (uppercase required). The column itself must be of BIGINT UNSIGNED NOT NULL type, with a unique index named FTS_DOC_ID_INDEX. Preferably, you define this column when creating the table. If InnoDB must add the column to the table while creating a FULLTEXT index, the indexing operation is considerably more expensive.
See Also full-text search, FULLTEXT index, ilist.

doublewrite buffer
InnoDB uses a file flush technique called doublewrite. Before writing **pages** to the **data files**, InnoDB first writes them to a contiguous area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer have completed, does InnoDB write the pages to their proper positions in the data file. If there is an operating system, storage subsystem, or mysqld process crash in the middle of a page write, InnoDB can later find a good copy of the page from the doublewrite buffer during **crash recovery**.

Although data is always written twice, the doublewrite buffer does not require twice as much I/O overhead or twice as many I/O operations. Data is written to the buffer itself as a large sequential chunk, with a single `fsync()` call to the operating system.

To turn off the doublewrite buffer, specify the option `innodb_doublewrite=0`.
See Also crash recovery, data files, page, purge.

drop
A kind of **DDL** operation that removes a schema object, through a statement such as `DROP TABLE` or `DROP INDEX`. It maps internally to an `ALTER TABLE` statement. From an `InnoDB` perspective, the performance considerations of such operations involve the time that the **data dictionary** is locked to ensure that interrelated objects are all updated, and the time to update memory structures such as the **buffer pool**. For a **table**, the drop operation has somewhat different characteristics than a **truncate** operation (`TRUNCATE TABLE` statement).
See Also buffer pool, data dictionary, DDL, table, truncate.

dynamic row format
An `InnoDB` row format. Because long variable-length column values are stored outside of the page that holds the row data, it is very efficient for rows that include large objects. Since the large fields are typically not accessed to evaluate query conditions, they are not brought into the **buffer pool** as often, resulting in fewer I/O operations and better utilization of cache memory.

As of MySQL 5.7.9, the default row format is defined by `innodb_default_row_format`, which has a default value of `DYNAMIC`.

For additional information about `InnoDB` `DYNAMIC` row format, see DYNAMIC Row Format.
See Also buffer pool, file format, row format.

E

early adopter
A stage similar to **beta**, when a software product is typically evaluated for performance, functionality, and compatibility in a non-mission-critical setting.
See Also beta.

error log
A type of **log** showing information about MySQL startup and critical runtime errors and **crash** information. For details, see Section 5.4.2, “The Error Log”.
See Also crash, log.

eviction
The process of removing an item from a cache or other temporary storage area, such as the `InnoDB` **buffer pool**. Often, but not always, uses the **LRU** algorithm to determine which item to remove. When a **dirty page** is evicted, its contents are **flushed** to disk, and any dirty **neighbor pages** might be flushed also.
See Also buffer pool, dirty page, flush, LRU, neighbor page.

exclusive lock
A kind of **lock** that prevents any other **transaction** from locking the same row. Depending on the transaction **isolation level**, this kind of lock might block other transactions from writing to the same row, or might also block other transactions from reading the same row. The default `InnoDB` isolation level, **REPEATABLE READ**, enables higher **concurrency** by allowing transactions to read rows that have exclusive locks, a technique known as **consistent read**.
See Also concurrency, consistent read, isolation level, lock, REPEATABLE READ, shared lock, transaction.

extent
A group of **pages** within a **tablespace**. For the default **page size** of 16KB, an extent contains 64 pages.
In MySQL 5.6, the page size for an `InnoDB` instance can be 4KB, 8KB, or 16KB, controlled by the

`innodb_page_size` configuration option. For 4KB, 8KB, and 16KB pages sizes, the extent size is always 1MB (or 1048576 bytes).

Support for 32KB and 64KB `InnoDB` page sizes was added in MySQL 5.7.6. For a 32KB page size, the extent size is 2MB. For a 64KB page size, the extent size is 4MB.

`InnoDB` features such as **segments**, **read-ahead** requests and the **doublewrite buffer** use I/O operations that read, write, allocate, or free data one extent at a time.
See Also doublewrite buffer, page, page size, read-ahead, segment, tablespace.

F

.frm file
: A file containing the metadata, such as the table definition, of a MySQL table. `.frm` files were removed in MySQL 8.0 but are still used in earlier MySQL releases. In MySQL 8.0, data previously stored in `.frm` files is stored in **data dictionary** tables.
See Also data dictionary, MySQL Enterprise Backup, system tablespace.

Fast Index Creation
: A capability first introduced in the InnoDB Plugin, now part of MySQL in 5.5 and higher, that speeds up creation of `InnoDB` **secondary indexes** by avoiding the need to completely rewrite the associated table. The speedup applies to dropping secondary indexes also.

Because index maintenance can add performance overhead to many data transfer operations, consider doing operations such as `ALTER TABLE ... ENGINE=INNODB` or `INSERT INTO ... SELECT * FROM ...` without any secondary indexes in place, and creating the indexes afterward.

In MySQL 5.6, this feature becomes more general. You can read and write to tables while an index is being created, and many more kinds of `ALTER TABLE` operations can be performed without copying the table, without blocking **DML** operations, or both. Thus in MySQL 5.6 and higher, this set of features is referred to as **online DDL** rather than Fast Index Creation.

For related information, see InnoDB Fast Index Creation, and Section 15.12, “InnoDB and Online DDL”.
See Also DML, index, online DDL, secondary index.

fast shutdown
: The default **shutdown** procedure for `InnoDB`, based on the configuration setting `innodb_fast_shutdown=1`. To save time, certain **flush** operations are skipped. This type of shutdown is safe during normal usage, because the flush operations are performed during the next startup, using the same mechanism as in **crash recovery**. In cases where the database is being shut down for an upgrade or downgrade, do a **slow shutdown** instead to ensure that all relevant changes are applied to the **data files** during the shutdown.
See Also crash recovery, data files, flush, shutdown, slow shutdown.

file format
: The file format for `InnoDB` tables.
See Also file-per-table, .ibd file, ibdata file, row format.

file-per-table
: A general name for the setting controlled by the `innodb_file_per_table` option, which is an important configuration option that affects aspects of `InnoDB` file storage, availability of features, and I/O characteristics. As of MySQL 5.6.7, `innodb_file_per_table` is enabled by default.

With the `innodb_file_per_table` option enabled, you can create a table in its own **.ibd file** rather than in the shared **ibdata files** of the **system tablespace**. When table data is stored in an individual **.ibd file**, you have more flexibility to choose **row formats** required for features such as data **compression**. The `TRUNCATE TABLE` operation is also faster, and reclaimed space can be used by the operating system rather than remaining reserved for `InnoDB`.

The **MySQL Enterprise Backup** product is more flexible for tables that are in their own files. For example, tables can be excluded from a backup, but only if they are in separate files. Thus, this setting is suitable for tables that are backed up less frequently or on a different schedule.
See Also compressed row format, compression, file format, .ibd file, ibdata file, innodb_file_per_table, MySQL Enterprise Backup, row format, system tablespace.

fill factor
In an `InnoDB` **index**, the proportion of a **page** that is taken up by index data before the page is split. The unused space when index data is first divided between pages allows for rows to be updated with longer string values without requiring expensive index maintenance operations. If the fill factor is too low, the index consumes more space than needed, causing extra I/O overhead when reading the index. If the fill factor is too high, any update that increases the length of column values can cause extra I/O overhead for index maintenance. See Section 15.6.2.2, "The Physical Structure of an InnoDB Index" for more information.
See Also index, page.

fixed row format
This row format is used by the `MyISAM` storage engine, not by `InnoDB`. If you create an `InnoDB` table with the option `ROW_FORMAT=FIXED` in MySQL 5.7.6 or earlier, `InnoDB` uses the **compact row format** instead, although the `FIXED` value might still show up in output such as `SHOW TABLE STATUS` reports. As of MySQL 5.7.7, `InnoDB` returns an error if `ROW_FORMAT=FIXED` is specified.
See Also compact row format, row format.

flush
To write changes to the database files, that had been buffered in a memory area or a temporary disk storage area. The `InnoDB` storage structures that are periodically flushed include the **redo log**, the **undo log**, and the **buffer pool**.

Flushing can happen because a memory area becomes full and the system needs to free some space, because a **commit** operation means the changes from a transaction can be finalized, or because a **slow shutdown** operation means that all outstanding work should be finalized. When it is not critical to flush all the buffered data at once, `InnoDB` can use a technique called **fuzzy checkpointing** to flush small batches of pages to spread out the I/O overhead.
See Also buffer pool, commit, fuzzy checkpointing, redo log, slow shutdown, undo log.

flush list
An internal `InnoDB` data structure that tracks **dirty pages** in the **buffer pool**: that is, **pages** that have been changed and need to be written back out to disk. This data structure is updated frequently by `InnoDB` internal **mini-transactions**, and so is protected by its own **mutex** to allow concurrent access to the buffer pool.
See Also buffer pool, dirty page, LRU, mini-transaction, mutex, page, page cleaner.

foreign key
A type of pointer relationship, between rows in separate `InnoDB` tables. The foreign key relationship is defined on one column in both the **parent table** and the **child table**.

In addition to enabling fast lookup of related information, foreign keys help to enforce **referential integrity**, by preventing any of these pointers from becoming invalid as data is inserted, updated, and deleted. This enforcement mechanism is a type of **constraint**. A row that points to another table cannot be inserted if the associated foreign key value does not exist in the other table. If a row is deleted or its foreign key value changed, and rows in another table point to that foreign key value, the foreign key can be set up to prevent the deletion, cause the corresponding column values in the other table to become **null**, or automatically delete the corresponding rows in the other table.

One of the stages in designing a **normalized** database is to identify data that is duplicated, separate that data into a new table, and set up a foreign key relationship so that the multiple tables can be queried like a single table, using a **join** operation.

See Also child table, FOREIGN KEY constraint, join, normalized, NULL, parent table, referential integrity, relational.

FOREIGN KEY constraint
The type of **constraint** that maintains database consistency through a **foreign key** relationship. Like other kinds of constraints, it can prevent data from being inserted or updated if data would become inconsistent; in this case, the inconsistency being prevented is between data in multiple tables. Alternatively, when a **DML** operation is performed, `FOREIGN KEY` constraints can cause data in **child rows** to be deleted, changed to different values, or set to **null**, based on the `ON CASCADE` option specified when creating the foreign key.
See Also child table, constraint, DML, foreign key, NULL.

FTS
In most contexts, an acronym for **full-text search**. Sometimes in performance discussions, an acronym for **full table scan**.
See Also full table scan, full-text search.

full backup
A **backup** that includes all the **tables** in each MySQL **database**, and all the databases in a MySQL **instance**. Contrast with **partial backup**.
See Also backup, database, instance, partial backup, table.

full table scan
An operation that requires reading the entire contents of a table, rather than just selected portions using an **index**. Typically performed either with small lookup tables, or in data warehousing situations with large tables where all available data is aggregated and analyzed. How frequently these operations occur, and the sizes of the tables relative to available memory, have implications for the algorithms used in query optimization and managing the **buffer pool**.

The purpose of indexes is to allow lookups for specific values or ranges of values within a large table, thus avoiding full table scans when practical.
See Also buffer pool, index.

full-text search
The MySQL feature for finding words, phrases, Boolean combinations of words, and so on within table data, in a faster, more convenient, and more flexible way than using the SQL `LIKE` operator or writing your own application-level search algorithm. It uses the SQL function `MATCH()` and **FULLTEXT indexes**.
See Also FULLTEXT index.

FULLTEXT index
The special kind of **index** that holds the **search index** in the MySQL **full-text search** mechanism. Represents the words from values of a column, omitting any that are specified as **stopwords**. Originally, only available for `MyISAM` tables. Starting in MySQL 5.6.4, it is also available for **InnoDB** tables.
See Also full-text search, index, InnoDB, search index, stopword.

fuzzy checkpointing
A technique that **flushes** small batches of **dirty pages** from the **buffer pool**, rather than flushing all dirty pages at once which would disrupt database processing.
See Also buffer pool, dirty page, flush.

G

GA
"Generally available", the stage when a software product leaves **beta** and is available for sale, official support, and production use.

See Also beta.

gap
A place in an InnoDB **index** data structure where new values could be inserted. When you lock a set of rows with a statement such as SELECT ... FOR UPDATE, InnoDB can create locks that apply to the gaps as well as the actual values in the index. For example, if you select all values greater than 10 for update, a gap lock prevents another transaction from inserting a new value that is greater than 10. The **supremum record** and **infimum record** represent the gaps containing all values greater than or less than all the current index values.
See Also concurrency, gap lock, index, infimum record, isolation level, supremum record.

gap lock
A **lock** on a **gap** between index records, or a lock on the gap before the first or after the last index record. For example, SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE; prevents other transactions from inserting a value of 15 into the column t.c1, whether or not there was already any such value in the column, because the gaps between all existing values in the range are locked. Contrast with **record lock** and **next-key lock**.

Gap locks are part of the tradeoff between performance and **concurrency**, and are used in some transaction **isolation levels** and not others.
See Also gap, infimum record, lock, next-key lock, record lock, supremum record.

general log
See general query log.

general query log
A type of **log** used for diagnosis and troubleshooting of SQL statements processed by the MySQL server. Can be stored in a file or in a database table. You must enable this feature through the general_log configuration option to use it. You can disable it for a specific connection through the sql_log_off configuration option.

Records a broader range of queries than the **slow query log**. Unlike the **binary log**, which is used for replication, the general query log contains SELECT statements and does not maintain strict ordering. For more information, see Section 5.4.3, “The General Query Log”.
See Also binary log, log, slow query log.

general tablespace
A shared InnoDB **tablespace** created using CREATE TABLESPACE syntax. General tablespaces can be created outside of the MySQL data directory, are capable of holding multiple **tables**, and support tables of all row formats. General tablespaces were introduced in MySQL 5.7.6.

Tables are added to a general tablespace using CREATE TABLE *tbl_name* ... TABLESPACE [=] *tablespace_name* or ALTER TABLE *tbl_name* TABLESPACE [=] *tablespace_name* syntax.

Contrast with **system tablespace** and **file-per-table** tablespace.

For more information, see Section 15.6.3.3, “General Tablespaces”.
See Also file-per-table, system tablespace, table, tablespace.

generated column
A column whose values are computed from an expression included in the column definition. A generated column can be **virtual** or **stored**.
See Also base column, stored generated column, virtual generated column.

generated stored column
See stored generated column.

generated virtual column
See virtual generated column.

global temporary tablespace
A *temporary tablespace* that stores *rollback segments* for changes made to user-created temporary tables.
See Also temporary tablespace.

global transaction
A type of **transaction** involved in **XA** operations. It consists of several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends **ACID** properties "up a level" so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties.
See Also ACID, transaction, XA.

group commit
An `InnoDB` optimization that performs some low-level I/O operations (log write) once for a set of **commit** operations, rather than flushing and syncing separately for each commit.
See Also binary log, commit.

H

hash index
A type of **index** intended for queries that use equality operators, rather than range operators such as greater-than or `BETWEEN`. It is available for `MEMORY` tables. Although hash indexes are the default for `MEMORY` tables for historic reasons, that storage engine also supports **B-tree** indexes, which are often a better choice for general-purpose queries.

MySQL includes a variant of this index type, the **adaptive hash index**, that is constructed automatically for `InnoDB` tables if needed based on runtime conditions.
See Also adaptive hash index, B-tree, index, InnoDB.

HDD
Acronym for "hard disk drive". Refers to storage media using spinning platters, usually when comparing and contrasting with **SSD**. Its performance characteristics can influence the throughput of a **disk-based** workload.
See Also disk-based, SSD.

heartbeat
A periodic message that is sent to indicate that a system is functioning properly. In a **replication** context, if the **master** stops sending such messages, one of the **slaves** can take its place. Similar techniques can be used between the servers in a cluster environment, to confirm that all of them are operating properly.
See Also master server, replication.

high-water mark
A value representing an upper limit, either a hard limit that should not be exceeded at runtime, or a record of the maximum value that was actually reached. Contrast with **low-water mark**.
See Also low-water mark.

history list
A list of **transactions** with delete-marked records scheduled to be processed by the `InnoDB` **purge** operation. Recorded in the **undo log**. The length of the history list is reported by the command `SHOW ENGINE INNODB STATUS`. If the history list grows longer than the value of the `innodb_max_purge_lag` configuration option, each **DML** operation is delayed slightly to allow the purge operation to finish **flushing** the deleted records.

Also known as **purge lag**.

See Also DML, flush, purge, purge lag, rollback segment, transaction, undo log.

hole punching
: Releasing empty blocks from a page. The `InnoDB` **transparent page compression** feature relies on hole punching support. For more information, see Section 15.9.2, “InnoDB Page Compression”.
See Also sparse file, transparent page compression.

hot
: A condition where a row, table, or internal data structure is accessed so frequently, requiring some form of locking or mutual exclusion, that it results in a performance or scalability issue.

Although “hot” typically indicates an undesirable condition, a **hot backup** is the preferred type of backup.
See Also hot backup.

hot backup
: A backup taken while the database is running and applications are reading and writing to it. The backup involves more than simply copying data files: it must include any data that was inserted or updated while the backup was in process; it must exclude any data that was deleted while the backup was in process; and it must ignore any changes that were not committed.

The Oracle product that performs hot backups, of `InnoDB` tables especially but also tables from `MyISAM` and other storage engines, is known as **MySQL Enterprise Backup**.

The hot backup process consists of two stages. The initial copying of the data files produces a **raw backup**. The **apply** step incorporates any changes to the database that happened while the backup was running. Applying the changes produces a **prepared** backup; these files are ready to be restored whenever necessary.
See Also apply, MySQL Enterprise Backup, prepared backup, raw backup.

I

.ibd file
: The data file for **file-per-table** tablespaces and general tablespaces. File-per-table tablespace `.ibd` files contain a single table and associated index data. **General tablespace** `.ibd` files may contain table and index data for multiple tables.

The `.ibd` file extension does not apply to the **system tablespace**, which consists of one or more **ibdata files**.

If a file-per-table tablespace or general tablespace is created with the `DATA DIRECTORY =` clause, the `.ibd` file is located at the specified path, outside the normal data directory.

When a `.ibd` file is included in a compressed backup by the **MySQL Enterprise Backup** product, the compressed equivalent is a `.ibz` file.
See Also database, file-per-table, general tablespace, ibdata file, .ibz file, innodb_file_per_table, MySQL Enterprise Backup, system tablespace.

.ibz file
: When the **MySQL Enterprise Backup** product performs a **compressed backup**, it transforms each **tablespace** file that is created using the **file-per-table** setting from a `.ibd` extension to a `.ibz` extension.

The compression applied during backup is distinct from the **compressed row format** that keeps table data compressed during normal operation. A compressed backup operation skips the compression step for a tablespace that is already in compressed row format, as compressing a second time would slow down the backup but produce little or no space savings.
See Also compressed backup, compressed row format, file-per-table, .ibd file, MySQL Enterprise Backup, tablespace.

I/O-bound
See disk-bound.

ib-file set
The set of files managed by InnoDB within a MySQL database: the **system tablespace**, **file-per-table** tablespace files, and **redo log** files. Depending on MySQL version and InnoDB configuration, may also include **general tablespace**, **temporary tablespace**, and **undo tablespace** files. This term is sometimes used in detailed discussions of InnoDB file structures and formats to refer to the set of files managed by InnoDB within a MySQL database.
See Also database, file-per-table, general tablespace, redo log, system tablespace, temporary tablespace, undo tablespace.

ibbackup_logfile
A supplemental backup file created by the **MySQL Enterprise Backup** product during a **hot backup** operation. It contains information about any data changes that occurred while the backup was running. The initial backup files, including ibbackup_logfile, are known as a **raw backup**, because the changes that occurred during the backup operation are not yet incorporated. After you perform the **apply** step to the raw backup files, the resulting files do include those final data changes, and are known as a **prepared backup**. At this stage, the ibbackup_logfile file is no longer necessary.
See Also apply, hot backup, MySQL Enterprise Backup, prepared backup, raw backup.

ibdata file
A set of files with names such as ibdata1, ibdata2, and so on, that make up the InnoDB **system tablespace**. These files contain metadata about InnoDB tables, (the InnoDB **data dictionary**), and the storage areas for one or more **undo logs**, the **change buffer**, and the **doublewrite buffer**. They also can contain some or all of the table data also (depending on whether the **file-per-table** mode is in effect when each table is created). When the **innodb_file_per_table** option is enabled, data and indexes for newly created tables are stored in separate **.ibd files** rather than in the system tablespace.

The growth of the ibdata files is influenced by the innodb_autoextend_increment configuration option.
See Also change buffer, data dictionary, doublewrite buffer, file-per-table, .ibd file, innodb_file_per_table, system tablespace, undo log.

ibtmp file
The InnoDB **temporary tablespace data file** for non-compressed InnoDB **temporary tables** and related objects. The configuration file option, innodb_temp_data_file_path, allows users to define a relative path for the temporary tablespace data file. If innodb_temp_data_file_path is not specified, the default behavior is to create a single auto-extending 12MB data file named ibtmp1 in the data directory, alongside ibdata1.
See Also data files, temporary table, temporary tablespace.

ib_logfile
A set of files, typically named ib_logfile0 and ib_logfile1, that form the **redo log**. Also sometimes referred to as the **log group**. These files record statements that attempt to change data in InnoDB tables. These statements are replayed automatically to correct data written by incomplete transactions, on startup following a crash.

This data cannot be used for manual recovery; for that type of operation, use the **binary log**.
See Also binary log, log group, redo log.

ilist
Within an InnoDB **FULLTEXT index**, the data structure consisting of a document ID and positional information for a token (that is, a particular word).
See Also FULLTEXT index.

implicit row lock
A row lock that InnoDB acquires to ensure consistency, without you specifically requesting it.

See Also row lock.

in-memory database
A type of database system that maintains data in memory, to avoid overhead due to disk I/O and translation between disk blocks and memory areas. Some in-memory databases sacrifice durability (the “D” in the **ACID** design philosophy) and are vulnerable to hardware, power, and other types of failures, making them more suitable for read-only operations. Other in-memory databases do use durability mechanisms such as logging changes to disk or using non-volatile memory.

MySQL features that address the same kinds of memory-intensive processing include the `InnoDB` **buffer pool**, **adaptive hash index**, and **read-only transaction** optimization, the `MEMORY` storage engine, the `MyISAM` key cache, and the MySQL query cache.
See Also ACID, adaptive hash index, buffer pool, disk-based, read-only transaction.

incremental backup
A type of **hot backup**, performed by the **MySQL Enterprise Backup** product, that only saves data changed since some point in time. Having a full backup and a succession of incremental backups lets you reconstruct backup data over a long period, without the storage overhead of keeping several full backups on hand. You can restore the full backup and then apply each of the incremental backups in succession, or you can keep the full backup up-to-date by applying each incremental backup to it, then perform a single restore operation.

The granularity of changed data is at the **page** level. A page might actually cover more than one row. Each changed page is included in the backup.
See Also hot backup, MySQL Enterprise Backup, page.

index
A data structure that provides a fast lookup capability for **rows** of a **table**, typically by forming a tree structure (**B-tree)** representing all the values of a particular **column** or set of columns.

`InnoDB` tables always have a **clustered index** representing the **primary key**. They can also have one or more **secondary indexes** defined on one or more columns. Depending on their structure, secondary indexes can be classified as **partial**, **column**, or **composite** indexes.

Indexes are a crucial aspect of **query** performance. Database architects design tables, queries, and indexes to allow fast lookups for data needed by applications. The ideal database design uses a **covering index** where practical; the query results are computed entirely from the index, without reading the actual table data. Each **foreign key** constraint also requires an index, to efficiently check whether values exist in both the **parent** and **child** tables.

Although a B-tree index is the most common, a different kind of data structure is used for **hash indexes**, as in the `MEMORY` storage engine and the `InnoDB` **adaptive hash index**. **R-tree** indexes are used for spatial indexing of multi-dimensional information.
See Also adaptive hash index, B-tree, child table, clustered index, column index, composite index, covering index, foreign key, hash index, parent table, partial index, primary key, query, R-tree, row, secondary index, table.

index cache
A memory area that holds the token data for `InnoDB` **full-text search**. It buffers the data to minimize disk I/O when data is inserted or updated in columns that are part of a **FULLTEXT index**. The token data is written to disk when the index cache becomes full. Each `InnoDB` `FULLTEXT` index has its own separate index cache, whose size is controlled by the configuration option `innodb_ft_cache_size`.
See Also full-text search, FULLTEXT index.

index condition pushdown
Index condition pushdown (ICP) is an optimization that pushes part of a `WHERE` condition down to the storage engine if parts of the condition can be evaluated using fields from the **index**. ICP can reduce the number of times

the **storage engine** must access the base table and the number of times the MySQL server must access the storage engine. For more information, see Section 8.2.1.6, “Index Condition Pushdown Optimization”.
See Also index, storage engine.

index hint
Extended SQL syntax for overriding the **indexes** recommended by the optimizer. For example, the `FORCE INDEX`, `USE INDEX`, and `IGNORE INDEX` clauses. Typically used when indexed columns have unevenly distributed values, resulting in inaccurate **cardinality** estimates.
See Also cardinality, index.

index prefix
In an **index** that applies to multiple columns (known as a **composite index**), the initial or leading columns of the index. A query that references the first 1, 2, 3, and so on columns of a composite index can use the index, even if the query does not reference all the columns in the index.
See Also composite index, index.

index statistics
See statistics.

infimum record
A **pseudo-record** in an **index**, representing the **gap** below the smallest value in that index. If a transaction has a statement such as `SELECT ... FROM ... WHERE col < 10 FOR UPDATE;`, and the smallest value in the column is 5, it is a lock on the infimum record that prevents other transactions from inserting even smaller values such as 0, -10, and so on.
See Also gap, index, pseudo-record, supremum record.

INFORMATION_SCHEMA
The name of the **database** that provides a query interface to the MySQL **data dictionary**. (This name is defined by the ANSI SQL standard.) To examine information (metadata) about the database, you can query tables such as `INFORMATION_SCHEMA.TABLES` and `INFORMATION_SCHEMA.COLUMNS`, rather than using `SHOW` commands that produce unstructured output.

The `INFORMATION_SCHEMA` database also contains tables specific to **InnoDB** that provide a query interface to the `InnoDB` data dictionary. You use these tables not to see how the database is structured, but to get real-time information about the workings of `InnoDB` tables to help with performance monitoring, tuning, and troubleshooting.
See Also data dictionary, database, InnoDB.

InnoDB
A MySQL component that combines high performance with **transactional** capability for reliability, robustness, and concurrent access. It embodies the **ACID** design philosophy. Represented as a **storage engine**; it handles tables created or altered with the `ENGINE=INNODB` clause. See Chapter 15, *The InnoDB Storage Engine* for architectural details and administration procedures, and Section 8.5, “Optimizing for InnoDB Tables” for performance advice.

In MySQL 5.5 and higher, `InnoDB` is the default storage engine for new tables and the `ENGINE=INNODB` clause is not required.

`InnoDB` tables are ideally suited for **hot backups**. See Section 30.2, “MySQL Enterprise Backup Overview” for information about the **MySQL Enterprise Backup** product for backing up MySQL servers without interrupting normal processing.
See Also ACID, hot backup, MySQL Enterprise Backup, storage engine, transaction.

innodb_autoinc_lock_mode
The `innodb_autoinc_lock_mode` option controls the algorithm used for **auto-increment locking**. When you have an auto-incrementing **primary key**, you can use statement-based replication only with the setting

`innodb_autoinc_lock_mode=1`. This setting is known as *consecutive* lock mode, because multi-row inserts within a transaction receive consecutive auto-increment values. If you have `innodb_autoinc_lock_mode=2`, which allows higher concurrency for insert operations, use row-based replication rather than statement-based replication. This setting is known as *interleaved* lock mode, because multiple multi-row insert statements running at the same time can receive **auto-increment** values that are interleaved. The setting `innodb_autoinc_lock_mode=0` should not be used except for compatibility purposes.

Consecutive lock mode (`innodb_autoinc_lock_mode=1`) is the default setting prior to MySQL 8.0.3. As of MySQL 8.0.3, interleaved lock mode (`innodb_autoinc_lock_mode=2`) is the default, which reflects the change from statement-based to row-based replication as the default replication type.
See Also auto-increment, auto-increment locking, mixed-mode insert, primary key.

innodb_file_per_table
An important configuration option that affects many aspects of `InnoDB` file storage, availability of features, and I/O characteristics. In MySQL 5.6.7 and higher, it is enabled by default. The `innodb_file_per_table` option turns on **file-per-table** mode. With this mode enabled, a newly created `InnoDB` table and associated indexes can be stored in a file-per-table **.ibd file**, outside the **system tablespace**.

This option affects the performance and storage considerations for a number of SQL statements, such as `DROP TABLE` and `TRUNCATE TABLE`.

Enabling the `innodb_file_per_table` option allows you to take advantage of features such as table **compression** and named-table backups in **MySQL Enterprise Backup**.

For more information, see `innodb_file_per_table`, and Section 15.6.3.2, “File-Per-Table Tablespaces”.
See Also compression, file-per-table, .ibd file, MySQL Enterprise Backup, system tablespace.

innodb_lock_wait_timeout
The `innodb_lock_wait_timeout` option sets the balance between **waiting** for shared resources to become available, or giving up and handling the error, retrying, or doing alternative processing in your application. Rolls back any `InnoDB` transaction that waits more than a specified time to acquire a **lock**. Especially useful if **deadlocks** are caused by updates to multiple tables controlled by different storage engines; such deadlocks are not **detected** automatically.
See Also deadlock, deadlock detection, lock, wait.

innodb_strict_mode
The `innodb_strict_mode` option controls whether `InnoDB` operates in **strict mode**, where conditions that are normally treated as warnings, cause errors instead (and the underlying statements fail).
See Also strict mode.

insert
One of the primary **DML** operations in **SQL**. The performance of inserts is a key factor in **data warehouse** systems that load millions of rows into tables, and **OLTP** systems where many concurrent connections might insert rows into the same table, in arbitrary order. If insert performance is important to you, you should learn about **InnoDB** features such as the **insert buffer** used in **change buffering**, and **auto-increment** columns.
See Also auto-increment, change buffering, data warehouse, DML, InnoDB, insert buffer, OLTP, SQL.

insert buffer
The former name of the **change buffer**. In MySQL 5.5, support was added for buffering changes to secondary index pages for `DELETE` and `UPDATE` operations. Previously, only changes resulting from `INSERT` operations were buffered. The preferred term is now *change buffer*.
See Also change buffer, change buffering.

insert buffering
The technique of storing changes to secondary index pages, resulting from `INSERT` operations, in the **change buffer** rather than writing the changes immediately, so that the physical writes can be performed to minimize random I/O. It is one of the types of **change buffering**; the others are **delete buffering** and **purge buffering**.

Insert buffering is not used if the secondary index is **unique**, because the uniqueness of new values cannot be verified before the new entries are written out. Other kinds of change buffering do work for unique indexes.
See Also change buffer, change buffering, delete buffering, insert buffer, purge buffering, unique index.

insert intention lock
A type of **gap lock** that is set by `INSERT` operations prior to row insertion. This type of **lock** signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. For more information, see Section 15.7.1, “InnoDB Locking”.
See Also gap lock, lock, next-key lock.

instance
A single **mysqld** daemon managing a **data directory** representing one or more **databases** with a set of **tables**. It is common in development, testing, and some **replication** scenarios to have multiple instances on the same **server** machine, each managing its own data directory and listening on its own port or socket. With one instance running a **disk-bound** workload, the server might still have extra CPU and memory capacity to run additional instances.
See Also data directory, database, disk-bound, mysqld, replication, server, table.

instrumentation
Modifications at the source code level to collect performance data for tuning and debugging. In MySQL, data collected by instrumentation is exposed through an SQL interface using the `INFORMATION_SCHEMA` and `PERFORMANCE_SCHEMA` databases.
See Also INFORMATION_SCHEMA, Performance Schema.

intention exclusive lock
See intention lock.

intention lock
A kind of **lock** that applies to the table, used to indicate the kind of lock the **transaction** intends to acquire on rows in the table. Different transactions can acquire different kinds of intention locks on the same table, but the first transaction to acquire an *intention exclusive* (IX) lock on a table prevents other transactions from acquiring any S or X locks on the table. Conversely, the first transaction to acquire an *intention shared* (IS) lock on a table prevents other transactions from acquiring any X locks on the table. The two-phase process allows the lock requests to be resolved in order, without blocking locks and corresponding operations that are compatible. For more information about this locking mechanism, see Section 15.7.1, “InnoDB Locking”.
See Also lock, lock mode, locking, transaction.

intention shared lock
See intention lock.

intrinsic temporary table
An optimized internal `InnoDB` temporary table used by the *optimizer*.
See Also optimizer.

inverted index
A data structure optimized for document retrieval systems, used in the implementation of `InnoDB` **full-text search**. The `InnoDB` **FULLTEXT index**, implemented as an inverted index, records the position of each word within a document, rather than the location of a table row. A single column value (a document stored as a text string) is represented by many entries in the inverted index.
See Also full-text search, FULLTEXT index, ilist.

IOPS
Acronym for **I/O operations per second**. A common measurement for busy systems, particularly **OLTP** applications. If this value is near the maximum that the storage devices can handle, the application can become **disk-bound**, limiting **scalability**.

See Also disk-bound, OLTP, scalability.

isolation level
One of the foundations of database processing. Isolation is the **I** in the acronym **ACID**; the isolation level is the setting that fine-tunes the balance between performance and reliability, consistency, and reproducibility of results when multiple **transactions** are making changes and performing queries at the same time.

From highest amount of consistency and protection to the least, the isolation levels supported by InnoDB are: **SERIALIZABLE**, **REPEATABLE READ**, **READ COMMITTED**, and **READ UNCOMMITTED**.

With `InnoDB` tables, many users can keep the default isolation level (*REPEATABLE READ*) for all operations. Expert users might choose the **READ COMMITTED** level as they push the boundaries of scalability with **OLTP** processing, or during data warehousing operations where minor inconsistencies do not affect the aggregate results of large amounts of data. The levels on the edges (**SERIALIZABLE** and **READ UNCOMMITTED**) change the processing behavior to such an extent that they are rarely used.
See Also ACID, OLTP, READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

J

join
A **query** that retrieves data from more than one table, by referencing columns in the tables that hold identical values. Ideally, these columns are part of an `InnoDB` **foreign key** relationship, which ensures **referential integrity** and that the join columns are **indexed**. Often used to save space and improve query performance by replacing repeated strings with numeric IDs, in a **normalized** data design.
See Also foreign key, index, normalized, query, referential integrity.

K

KEY_BLOCK_SIZE
An option to specify the size of data pages within an `InnoDB` table that uses **compressed row format**. The default is 8 kilobytes. Lower values risk hitting internal limits that depend on the combination of row size and compression percentage.

For `MyISAM` tables, `KEY_BLOCK_SIZE` optionally specifies the size in bytes to use for index key blocks. The value is treated as a hint; a different size could be used if necessary. A `KEY_BLOCK_SIZE` value specified for an individual index definition overrides a table-level `KEY_BLOCK_SIZE` value.
See Also compressed row format.

L

latch
A lightweight structure used by `InnoDB` to implement a **lock** for its own internal memory structures, typically held for a brief time measured in milliseconds or microseconds. A general term that includes both **mutexes** (for exclusive access) and **rw-locks** (for shared access). Certain latches are the focus of `InnoDB` performance tuning. Statistics about latch use and contention are available through the **Performance Schema** interface.
See Also lock, locking, mutex, Performance Schema, rw-lock.

list
The `InnoDB` **buffer pool** is represented as a list of memory **pages**. The list is reordered as new pages are accessed and enter the buffer pool, as pages within the buffer pool are accessed again and are considered newer, and as pages that are not accessed for a long time are **evicted** from the buffer pool. The buffer pool is divided into **sublists**, and the replacement policy is a variation of the familiar **LRU** technique.
See Also buffer pool, eviction, LRU, page, sublist.

lock
: The high-level notion of an object that controls access to a resource, such as a table, row, or internal data structure, as part of a **locking** strategy. For intensive performance tuning, you might delve into the actual structures that implement locks, such as **mutexes** and **latches**.
See Also latch, lock mode, locking, mutex.

lock escalation
: An operation used in some database systems that converts many **row locks** into a single **table lock**, saving memory space but reducing concurrent access to the table. `InnoDB` uses a space-efficient representation for row locks, so that **lock** escalation is not needed.
See Also locking, row lock, table lock.

lock mode
: A shared (S) **lock** allows a **transaction** to read a row. Multiple transactions can acquire an S lock on that same row at the same time.

  An exclusive (X) lock allows a transaction to update or delete a row. No other transaction can acquire any kind of lock on that same row at the same time.

  Intention locks apply to the table, and are used to indicate what kind of lock the transaction intends to acquire on rows in the table. Different transactions can acquire different kinds of intention locks on the same table, but the first transaction to acquire an intention exclusive (IX) lock on a table prevents other transactions from acquiring any S or X locks on the table. Conversely, the first transaction to acquire an intention shared (IS) lock on a table prevents other transactions from acquiring any X locks on the table. The two-phase process allows the lock requests to be resolved in order, without blocking locks and corresponding operations that are compatible.
  See Also intention lock, lock, locking, transaction.

locking
: The system of protecting a **transaction** from seeing or changing data that is being queried or changed by other transactions. The **locking** strategy must balance reliability and consistency of database operations (the principles of the **ACID** philosophy) against the performance needed for good **concurrency**. Fine-tuning the locking strategy often involves choosing an **isolation level** and ensuring all your database operations are safe and reliable for that isolation level.
See Also ACID, concurrency, isolation level, locking, transaction.

locking read
: A `SELECT` statement that also performs a **locking** operation on an `InnoDB` table. Either `SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE`. It has the potential to produce a **deadlock**, depending on the **isolation level** of the transaction. The opposite of a **non-locking read**. Not allowed for global tables in a **read-only transaction**.

  `SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.

  See Section 15.7.2.4, “Locking Reads”.
  See Also deadlock, isolation level, locking, non-locking read, read-only transaction.

log
: In the `InnoDB` context, “log” or “log files” typically refers to the **redo log** represented by the **ib_logfile*N*** files. Another type of `InnoDB` log is the **undo log**, which is a storage area that holds copies of data modified by active transactions.

  Other kinds of logs that are important in MySQL are the **error log** (for diagnosing startup and runtime problems), **binary log** (for working with replication and performing point-in-time restores), the **general query log** (for diagnosing application problems), and the **slow query log** (for diagnosing performance problems).

See Also binary log, error log, general query log, ib_logfile, redo log, slow query log, undo log.

log buffer
: The memory area that holds data to be written to the **log files** that make up the **redo log**. It is controlled by the `innodb_log_buffer_size` configuration option.
See Also log file, redo log.

log file
: One of the **ib_logfile*N*** files that make up the **redo log**. Data is written to these files from the **log buffer** memory area.
See Also ib_logfile, log buffer, redo log.

log group
: The set of files that make up the **redo log**, typically named `ib_logfile0` and `ib_logfile1`. (For that reason, sometimes referred to collectively as **ib_logfile**.)
See Also ib_logfile, redo log.

logical
: A type of operation that involves high-level, abstract aspects such as tables, queries, indexes, and other SQL concepts. Typically, logical aspects are important to make database administration and application development convenient and usable. Contrast with **physical**.
See Also logical backup, physical.

logical backup
: A **backup** that reproduces table structure and data, without copying the actual data files. For example, the **`mysqldump`** command produces a logical backup, because its output contains statements such as `CREATE TABLE` and `INSERT` that can re-create the data. Contrast with **physical backup**. A logical backup offers flexibility (for example, you could edit table definitions or insert statements before restoring), but can take substantially longer to **restore** than a physical backup.
See Also backup, mysqldump, physical backup, restore.

loose_
: A prefix added to `InnoDB` configuration options after server **startup**, so any new configuration options not recognized by the current level of MySQL do not cause a startup failure. MySQL processes configuration options that start with this prefix, but gives a warning rather than a failure if the part after the prefix is not a recognized option.
See Also startup.

low-water mark
: A value representing a lower limit, typically a threshold value at which some corrective action begins or becomes more aggressive. Contrast with **high-water mark**.
See Also high-water mark.

LRU
: An acronym for “least recently used”, a common method for managing storage areas. The items that have not been used recently are **evicted** when space is needed to cache newer items. `InnoDB` uses the LRU mechanism by default to manage the **pages** within the **buffer pool**, but makes exceptions in cases where a page might be read only a single time, such as during a **full table scan**. This variation of the LRU algorithm is called the **midpoint insertion strategy**. For more information, see Section 15.5.1, “Buffer Pool”.
See Also buffer pool, eviction, full table scan, midpoint insertion strategy, page.

LSN
: Acronym for “log sequence number”. This arbitrary, ever-increasing value represents a point in time corresponding to operations recorded in the **redo log**. (This point in time is regardless of **transaction** boundaries; it can fall in the middle of one or more transactions.) It is used internally by `InnoDB` during **crash recovery** and for managing the **buffer pool**.

Prior to MySQL 5.6.3, the LSN was a 4-byte unsigned integer. The LSN became an 8-byte unsigned integer in MySQL 5.6.3 when the redo log file size limit increased from 4GB to 512GB, as additional bytes were required to store extra size information. Applications built on MySQL 5.6.3 or later that use LSN values should use 64-bit rather than 32-bit variables to store and compare LSN values.

In the **MySQL Enterprise Backup** product, you can specify an LSN to represent the point in time from which to take an **incremental backup**. The relevant LSN is displayed by the output of the `mysqlbackup` command. Once you have the LSN corresponding to the time of a full backup, you can specify that value to take a subsequent incremental backup, whose output contains another LSN for the next incremental backup.
See Also buffer pool, crash recovery, incremental backup, MySQL Enterprise Backup, redo log, transaction.

M

.MRG file
A file containing references to other tables, used by the `MERGE` storage engine. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command.

.MYD file
A file that MySQL uses to store data for a `MyISAM` table.
See Also .MYI file, MySQL Enterprise Backup, mysqlbackup command.

.MYI file
A file that MySQL uses to store indexes for a `MyISAM` table.
See Also .MYD file, MySQL Enterprise Backup, mysqlbackup command.

master server
Frequently shortened to “master”. A database server machine in a **replication** scenario that processes the initial insert, update, and delete requests for data. These changes are propagated to, and repeated on, other servers known as **slave servers**.
See Also replication, slave server.

master thread
An `InnoDB` **thread** that performs various tasks in the background. Most of these tasks are I/O related, such as writing changes from the **change buffer** to the appropriate secondary indexes.

To improve **concurrency**, sometimes actions are moved from the master thread to separate background threads. For example, in MySQL 5.6 and higher, **dirty pages** are **flushed** from the **buffer pool** by the **page cleaner** thread rather than the master thread.
See Also buffer pool, change buffer, concurrency, dirty page, flush, page cleaner, thread.

MDL
Acronym for “metadata lock”.
See Also metadata lock.

memcached
A popular component of many MySQL and **NoSQL** software stacks, allowing fast reads and writes for single values and caching the results entirely in memory. Traditionally, applications required extra logic to write the same data to a MySQL database for permanent storage, or to read data from a MySQL database when it was not cached yet in memory. Now, applications can use the simple `memcached` protocol, supported by client libraries for many languages, to communicate directly with MySQL servers using `InnoDB` or `NDB` tables. These NoSQL interfaces to MySQL tables allow applications to achieve higher read and write performance than by issuing SQL statements directly, and can simplify application logic and deployment configurations for systems that already incorporate `memcached` for in-memory caching.

The memcached interface to InnoDB tables is available in MySQL 5.6 and higher; see Section 15.19, “InnoDB memcached Plugin” for details. The memcached interface to NDB tables is available in NDB Cluster 7.2 and later; see http://dev.mysql.com/doc/ndbapi/en/ndbmemcache.html for details.
See Also InnoDB, NoSQL.

merge
To apply changes to data cached in memory, such as when a page is brought into the **buffer pool**, and any applicable changes recorded in the **change buffer** are incorporated into the page in the buffer pool. The updated data is eventually written to the **tablespace** by the **flush** mechanism.
See Also buffer pool, change buffer, flush, tablespace.

metadata lock
A type of **lock** that prevents **DDL** operations on a table that is being used at the same time by another **transaction**. For details, see Section 8.11.4, “Metadata Locking”.

Enhancements to **online** operations, particularly in MySQL 5.6 and higher, are focused on reducing the amount of metadata locking. The objective is for DDL operations that do not change the table structure (such as CREATE INDEX and DROP INDEX for InnoDB tables) to proceed while the table is being queried, updated, and so on by other transactions.
See Also DDL, lock, online, transaction.

metrics counter
A feature implemented by the INNODB_METRICS table in the **INFORMATION_SCHEMA**, in MySQL 5.6 and higher. You can query **counts** and totals for low-level InnoDB operations, and use the results for performance tuning in combination with data from the **Performance Schema**.
See Also counter, INFORMATION_SCHEMA, Performance Schema.

midpoint insertion strategy
The technique of initially bringing **pages** into the InnoDB **buffer pool** not at the “newest” end of the list, but instead somewhere in the middle. The exact location of this point can vary, based on the setting of the innodb_old_blocks_pct option. The intent is that pages that are only read once, such as during a **full table scan**, can be aged out of the buffer pool sooner than with a strict **LRU** algorithm. For more information, see Section 15.5.1, “Buffer Pool”.
See Also buffer pool, full table scan, LRU, page.

mini-transaction
An internal phase of InnoDB processing, when making changes at the **physical** level to internal data structures during **DML** operations. A mini-transaction (mtr) has no notion of **rollback**; multiple mini-transactions can occur within a single **transaction**. Mini-transactions write information to the **redo log** that is used during **crash recovery**. A mini-transaction can also happen outside the context of a regular transaction, for example during **purge** processing by background threads.
See Also commit, crash recovery, DML, physical, purge, redo log, rollback, transaction.

mixed-mode insert
An INSERT statement where **auto-increment** values are specified for some but not all of the new rows. For example, a multi-value INSERT could specify a value for the auto-increment column in some cases and NULL in other cases. InnoDB generates auto-increment values for the rows where the column value was specified as NULL. Another example is an INSERT ... ON DUPLICATE KEY UPDATE statement, where auto-increment values might be generated but not used, for any duplicate rows that are processed as UPDATE rather than INSERT statements.

Can cause consistency issues between **master** and **slave** servers in a **replication** configuration. Can require adjusting the value of the **innodb_autoinc_lock_mode** configuration option.
See Also auto-increment, innodb_autoinc_lock_mode, master server, replication, slave server.

mtr
See mini-transaction.

multi-core
A type of processor that can take advantage of multithreaded programs, such as the MySQL server.

multiversion concurrency control
See MVCC.

mutex
Informal abbreviation for “mutex variable”. (Mutex itself is short for “mutual exclusion”.) The low-level object that `InnoDB` uses to represent and enforce exclusive-access **locks** to internal in-memory data structures. Once the lock is acquired, any other process, thread, and so on is prevented from acquiring the same lock. Contrast with **rw-locks**, which `InnoDB` uses to represent and enforce shared-access **locks** to internal in-memory data structures. Mutexes and rw-locks are known collectively as **latches**.
See Also latch, lock, Performance Schema, Pthreads, rw-lock.

MVCC
Acronym for “multiversion concurrency control”. This technique lets `InnoDB` **transactions** with certain **isolation levels** perform **consistent read** operations; that is, to query rows that are being updated by other transactions, and see the values from before those updates occurred. This is a powerful technique to increase **concurrency**, by allowing queries to proceed without waiting due to **locks** held by the other transactions.

This technique is not universal in the database world. Some other database products, and some other MySQL storage engines, do not support it.
See Also ACID, concurrency, consistent read, isolation level, lock, transaction.

my.cnf
The name, on Unix or Linux systems, of the MySQL **option file**.
See Also my.ini, option file.

my.ini
The name, on Windows systems, of the MySQL **option file**.
See Also my.cnf, option file.

mysql
The `mysql` program is the command-line interpreter for the MySQL database. It processes **SQL** statements, and also MySQL-specific commands such as `SHOW TABLES`, by passing requests to the `mysqld` daemon.
See Also mysqld, SQL.

MySQL Enterprise Backup
A licensed product that performs **hot backups** of MySQL databases. It offers the most efficiency and flexibility when backing up `InnoDB` tables, but can also back up `MyISAM` and other kinds of tables.
See Also hot backup, InnoDB.

mysqlbackup command
A command-line tool of the **MySQL Enterprise Backup** product. It performs a **hot backup** operation for `InnoDB` tables, and a warm backup for `MyISAM` and other kinds of tables. See Section 30.2, “MySQL Enterprise Backup Overview” for more information about this command.
See Also hot backup, MySQL Enterprise Backup, warm backup.

mysqld
The `mysqld` program is the database engine for the MySQL database. It runs as a Unix daemon or Windows service, constantly waiting for requests and performing maintenance work in the background.
See Also mysql.

mysqldump
A command that performs a **logical backup** of some combination of databases, tables, and table data. The results are SQL statements that reproduce the original schema objects, data, or both. For substantial amounts

of data, a **physical backup** solution such as **MySQL Enterprise Backup** is faster, particularly for the **restore** operation.
See Also logical backup, MySQL Enterprise Backup, physical backup, restore.

N

natural key
An indexed column, typically a **primary key**, where the values have some real-world significance. Usually advised against because:

- If the value should ever change, there is potentially a lot of index maintenance to re-sort the **clustered index** and update the copies of the primary key value that are repeated in each **secondary index**.
- Even seemingly stable values can change in unpredictable ways that are difficult to represent correctly in the database. For example, one country can change into two or several, making the original country code obsolete. Or, rules about unique values might have exceptions. For example, even if taxpayer IDs are intended to be unique to a single person, a database might have to handle records that violate that rule, such as in cases of identity theft. Taxpayer IDs and other sensitive ID numbers also make poor primary keys, because they may need to be secured, encrypted, and otherwise treated differently than other columns.

Thus, it is typically better to use arbitrary numeric values to form a **synthetic key**, for example using an **auto-increment** column.
See Also auto-increment, clustered index, primary key, secondary index, synthetic key.

neighbor page
Any **page** in the same **extent** as a particular page. When a page is selected to be **flushed**, any neighbor pages that are **dirty** are typically flushed as well, as an I/O optimization for traditional hard disks. In MySQL 5.6 and up, this behavior can be controlled by the configuration variable `innodb_flush_neighbors`; you might turn that setting off for SSD drives, which do not have the same overhead for writing smaller batches of data at random locations.
See Also dirty page, extent, flush, page.

next-key lock
A combination of a **record lock** on the index record and a gap lock on the gap before the index record.
See Also gap lock, locking, record lock.

non-locking read
A **query** that does not use the `SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE` clauses. The only kind of query allowed for global tables in a **read-only transaction**. The opposite of a **locking read**. See Section 15.7.2.3, “Consistent Nonlocking Reads”.

`SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.
See Also locking read, query, read-only transaction.

non-repeatable read
The situation when a query retrieves data, and a later query within the same **transaction** retrieves what should be the same data, but the queries return different results (changed by another transaction committing in the meantime).

This kind of operation goes against the **ACID** principle of database design. Within a transaction, data should be consistent, with predictable and stable relationships.

Among different **isolation levels**, non-repeatable reads are prevented by the **serializable read** and **repeatable read** levels, and allowed by the **consistent read**, and **read uncommitted** levels.

See Also ACID, consistent read, isolation level, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

nonblocking I/O

An industry term that means the same as **asynchronous I/O**.
See Also asynchronous I/O.

normalized

A database design strategy where data is split into multiple tables, and duplicate values condensed into single rows represented by an ID, to avoid storing, querying, and updating redundant or lengthy values. It is typically used in **OLTP** applications.

For example, an address might be given a unique ID, so that a census database could represent the relationship **lives at this address** by associating that ID with each member of a family, rather than storing multiple copies of a complex value such as **123 Main Street, Anytown, USA**.

For another example, although a simple address book application might store each phone number in the same table as a person's name and address, a phone company database might give each phone number a special ID, and store the numbers and IDs in a separate table. This normalized representation could simplify large-scale updates when area codes split apart.

Normalization is not always recommended. Data that is primarily queried, and only updated by deleting entirely and reloading, is often kept in fewer, larger tables with redundant copies of duplicate values. This data representation is referred to as **denormalized**, and is frequently found in data warehousing applications.
See Also denormalized, foreign key, OLTP, relational.

NoSQL

A broad term for a set of data access technologies that do not use the **SQL** language as their primary mechanism for reading and writing data. Some NoSQL technologies act as key-value stores, only accepting single-value reads and writes; some relax the restrictions of the **ACID** methodology; still others do not require a pre-planned **schema**. MySQL users can combine NoSQL-style processing for speed and simplicity with SQL operations for flexibility and convenience, by using the **memcached** API to directly access some kinds of MySQL tables. The `memcached` interface to `InnoDB` tables is available in MySQL 5.6 and higher; see Section 15.19, “InnoDB memcached Plugin” for details. The `memcached` interface to `NDB` tables is available in NDB Cluster 7.2 and later; see ndbmemcache—Memcache API for NDB Cluster.
See Also ACID, InnoDB, memcached, schema, SQL.

NOT NULL constraint

A type of **constraint** that specifies that a **column** cannot contain any **NULL** values. It helps to preserve **referential integrity**, as the database server can identify data with erroneous missing values. It also helps in the arithmetic involved in query optimization, allowing the optimizer to predict the number of entries in an index on that column.
See Also column, constraint, NULL, primary key, referential integrity.

NULL

A special value in **SQL**, indicating the absence of data. Any arithmetic operation or equality test involving a `NULL` value, in turn produces a `NULL` result. (Thus it is similar to the IEEE floating-point concept of NaN, “not a number”.) Any aggregate calculation such as `AVG()` ignores rows with `NULL` values, when determining how many rows to divide by. The only test that works with `NULL` values uses the SQL idioms `IS NULL` or `IS NOT NULL`.

`NULL` values play a part in **index** operations, because for performance a database must minimize the overhead of keeping track of missing data values. Typically, `NULL` values are not stored in an index, because a query that tests an indexed column using a standard comparison operator could never match a row with a `NULL` value for that column. For the same reason, unique indexes do not prevent `NULL` values; those values simply are not represented in the index. Declaring a `NOT NULL` constraint on a column provides reassurance that there are

no rows left out of the index, allowing for better query optimization (accurate counting of rows and estimation of whether to use the index).

Because the **primary key** must be able to uniquely identify every row in the table, a single-column primary key cannot contain any `NULL` values, and a multi-column primary key cannot contain any rows with `NULL` values in all columns.

Although the Oracle database allows a `NULL` value to be concatenated with a string, `InnoDB` treats the result of such an operation as `NULL`.
See Also index, primary key, SQL.

O

.OPT file
A file containing database configuration information. Files with this extension are included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command.

off-page column
A column containing variable-length data (such as `BLOB` and `VARCHAR`) that is too long to fit on a **B-tree** page. The data is stored in **overflow pages**. The **DYNAMIC** row format is more efficient for such storage than the older **COMPACT** row format.
See Also B-tree, compact row format, dynamic row format, overflow page.

OLTP
Acronym for “Online Transaction Processing”. A database system, or a database application, that runs a workload with many **transactions**, with frequent writes as well as reads, typically affecting small amounts of data at a time. For example, an airline reservation system or an application that processes bank deposits. The data might be organized in **normalized** form for a balance between **DML** (insert/update/delete) efficiency and **query** efficiency. Contrast with **data warehouse**.

With its **row-level locking** and **transactional** capability, **InnoDB** is the ideal storage engine for MySQL tables used in OLTP applications.
See Also data warehouse, DML, InnoDB, query, row lock, transaction.

online
A type of operation that involves no downtime, blocking, or restricted operation for the database. Typically applied to **DDL**. Operations that shorten the periods of restricted operation, such as **fast index creation**, have evolved into a wider set of **online DDL** operations in MySQL 5.6.

In the context of backups, a **hot backup** is an online operation and a **warm backup** is partially an online operation.
See Also DDL, Fast Index Creation, hot backup, online DDL, warm backup.

online DDL
A feature that improves the performance, concurrency, and availability of `InnoDB` tables during **DDL** (primarily `ALTER TABLE`) operations. See Section 15.12, “InnoDB and Online DDL” for details.

The details vary according to the type of operation. In some cases, the table can be modified concurrently while the `ALTER TABLE` is in progress. The operation might be able to be performed without a table copy, or using a specially optimized type of table copy. DML log space usage for in-place operations is controlled by the `innodb_online_alter_log_max_size` configuration option.

This feature is an enhancement of the **Fast Index Creation** feature in MySQL 5.5.
See Also DDL, Fast Index Creation, online.

optimistic
: A methodology that guides low-level implementation decisions for a relational database system. The requirements of performance and **concurrency** in a relational database mean that operations must be started or dispatched quickly. The requirements of consistency and **referential integrity** mean that any operation could fail: a transaction might be rolled back, a **DML** operation could violate a constraint, a request for a lock could cause a deadlock, a network error could cause a timeout. An optimistic strategy is one that assumes most requests or attempts will succeed, so that relatively little work is done to prepare for the failure case. When this assumption is true, the database does little unnecessary work; when requests do fail, extra work must be done to clean up and undo changes.

  `InnoDB` uses optimistic strategies for operations such as **locking** and **commits**. For example, data changed by a transaction can be written to the data files before the commit occurs, making the commit itself very fast, but requiring more work to undo the changes if the transaction is rolled back.

  The opposite of an optimistic strategy is a **pessimistic** one, where a system is optimized to deal with operations that are unreliable and frequently unsuccessful. This methodology is rare in a database system, because so much care goes into choosing reliable hardware, networks, and algorithms.
  See Also commit, concurrency, DML, locking, pessimistic, referential integrity.

optimizer
: The MySQL component that determines the best **indexes** and **join** order to use for a **query**, based on characteristics and data distribution of the relevant **tables**.
  See Also index, join, query, table.

option
: A configuration parameter for MySQL, either stored in the **option file** or passed on the command line.

  For the **options** that apply to **InnoDB** tables, each option name starts with the prefix `innodb_`.
  See Also InnoDB, option, option file.

option file
: The file that holds the configuration **options** for the MySQL instance. Traditionally, on Linux and Unix this file is named `my.cnf`, and on Windows it is named `my.ini`.
  See Also configuration file, my.cnf, my.ini, option.

overflow page
: Separately allocated disk **pages** that hold variable-length columns (such as `BLOB` and `VARCHAR`) that are too long to fit on a **B-tree** page. The associated columns are known as **off-page columns**.
  See Also B-tree, off-page column, page.

P

.par file
: A file containing partition definitions. Files with this extension are included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.

  With the introduction of native partitioning support for `InnoDB` tables in MySQL 5.7.6, `.par` files are no longer created for partitioned `InnoDB` tables. Partitioned `MyISAM` tables continue to use `.par` files in MySQL 5.7. In MySQL 8.0, partitioning support is only provided by the `InnoDB` storage engine. As such, `.par` files are no longer used as of MySQL 8.0.
  See Also MySQL Enterprise Backup, mysqlbackup command.

page
: A unit representing how much data `InnoDB` transfers at any one time between disk (the **data files**) and memory (the **buffer pool**). A page can contain one or more **rows**, depending on how much data is in each row. If a

row does not fit entirely into a single page, InnoDB sets up additional pointer-style data structures so that the information about the row can be stored in one page.

One way to fit more data in each page is to use **compressed row format**. For tables that use BLOBs or large text fields, **compact row format** allows those large columns to be stored separately from the rest of the row, reducing I/O overhead and memory usage for queries that do not reference those columns.

When InnoDB reads or writes sets of pages as a batch to increase I/O throughput, it reads or writes an **extent** at a time.

All the InnoDB disk data structures within a MySQL instance share the same **page size**.
See Also buffer pool, compact row format, compressed row format, data files, extent, page size, row.

page cleaner
An InnoDB background **thread** that **flushes dirty pages** from the **buffer pool**. Prior to MySQL 5.6, this activity was performed by the **master thread**. The number of page cleaner threads is controlled by the innodb_page_cleaners configuration option, introduced in MySQL 5.7.4.
See Also buffer pool, dirty page, flush, master thread, thread.

page size
For releases up to and including MySQL 5.5, the size of each InnoDB **page** is fixed at 16 kilobytes. This value represents a balance: large enough to hold the data for most rows, yet small enough to minimize the performance overhead of transferring unneeded data to memory. Other values are not tested or supported.

Starting in MySQL 5.6, the page size for an InnoDB **instance** can be either 4KB, 8KB, or 16KB, controlled by the innodb_page_size configuration option. As of MySQL 5.7.6, InnoDB also supports 32KB and 64KB page sizes. For 32KB and 64KB page sizes, ROW_FORMAT=COMPRESSED is not supported and the maximum record size is 16KB.

Page size is set when creating the MySQL instance, and it remains constant afterward. The same page size applies to all InnoDB **tablespaces**, including the **system tablespace**, **file-per-table** tablespaces, and **general tablespaces**.

Smaller page sizes can help performance with storage devices that use small block sizes, particularly for **SSD** devices in **disk-bound** workloads, such as for **OLTP** applications. As individual rows are updated, less data is copied into memory, written to disk, reorganized, locked, and so on.
See Also disk-bound, file-per-table, general tablespace, instance, OLTP, page, SSD, system tablespace, tablespace.

parent table
The table in a **foreign key** relationship that holds the initial column values pointed to from the **child table**. The consequences of deleting, or updating rows in the parent table depend on the ON UPDATE and ON DELETE clauses in the foreign key definition. Rows with corresponding values in the child table could be automatically deleted or updated in turn, or those columns could be set to NULL, or the operation could be prevented.
See Also child table, foreign key.

partial backup
A **backup** that contains some of the **tables** in a MySQL database, or some of the databases in a MySQL instance. Contrast with **full backup**.
See Also backup, full backup, table.

partial index
An **index** that represents only part of a column value, typically the first N characters (the **prefix**) of a long VARCHAR value.
See Also index, index prefix.

Performance Schema

The `performance_schema` schema, in MySQL 5.5 and up, presents a set of tables that you can query to get detailed information about the performance characteristics of many internal parts of the MySQL server. See Chapter 26, *MySQL Performance Schema*.

See Also INFORMATION_SCHEMA, latch, mutex, rw-lock.

pessimistic

A methodology that sacrifices performance or concurrency in favor of safety. It is appropriate if a high proportion of requests or attempts might fail, or if the consequences of a failed request are severe. `InnoDB` uses what is known as a pessimistic **locking** strategy, to minimize the chance of **deadlocks**. At the application level, you might avoid deadlocks by using a pessimistic strategy of acquiring all locks needed by a transaction at the very beginning.

Many built-in database mechanisms use the opposite **optimistic** methodology.

See Also deadlock, locking, optimistic.

phantom

A row that appears in the result set of a query, but not in the result set of an earlier query. For example, if a query is run twice within a **transaction**, and in the meantime, another transaction commits after inserting a new row or updating a row so that it matches the `WHERE` clause of the query.

This occurrence is known as a phantom read. It is harder to guard against than a **non-repeatable read**, because locking all the rows from the first query result set does not prevent the changes that cause the phantom to appear.

Among different **isolation levels**, phantom reads are prevented by the **serializable read** level, and allowed by the **repeatable read**, **consistent read**, and **read uncommitted** levels.

See Also consistent read, isolation level, non-repeatable read, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

physical

A type of operation that involves hardware-related aspects such as disk blocks, memory pages, files, bits, disk reads, and so on. Typically, physical aspects are important during expert-level performance tuning and problem diagnosis. Contrast with **logical**.

See Also logical, physical backup.

physical backup

A **backup** that copies the actual data files. For example, the **`mysqlbackup`** command of the **MySQL Enterprise Backup** product produces a physical backup, because its output contains data files that can be used directly by the `mysqld` server, resulting in a faster **restore** operation. Contrast with **logical backup**.

See Also backup, logical backup, MySQL Enterprise Backup, restore.

PITR

Acronym for **point-in-time recovery**.

See Also point-in-time recovery.

plan stability

A property of a **query execution plan**, where the optimizer makes the same choices each time for a given **query**, so that performance is consistent and predictable.

See Also query, query execution plan.

point-in-time recovery

The process of restoring a **backup** to recreate the state of the database at a specific date and time. Commonly abbreviated “PITR”. Because it is unlikely that the specified time corresponds exactly to the time of a backup, this technique usually requires a combination of a **physical backup** and a **logical backup**. For example, with the **MySQL Enterprise Backup** product, you restore the last backup that you took before the specified point in time, then replay changes from the **binary log** between the time of the backup and the PITR time.

See Also backup, binary log, logical backup, MySQL Enterprise Backup, physical backup.

prefix
See index prefix.

prepared backup
A set of backup files, produced by the **MySQL Enterprise Backup** product, after all the stages of applying **binary logs** and **incremental backups** are finished. The resulting files are ready to be **restored**. Prior to the apply steps, the files are known as a **raw backup**.
See Also binary log, hot backup, incremental backup, MySQL Enterprise Backup, raw backup, restore.

primary key
A set of columns—and by implication, the index based on this set of columns—that can uniquely identify every row in a table. As such, it must be a unique index that does not contain any `NULL` values.

`InnoDB` requires that every table has such an index (also called the **clustered index** or **cluster index**), and organizes the table storage based on the column values of the primary key.

When choosing primary key values, consider using arbitrary values (a **synthetic key**) rather than relying on values derived from some other source (a **natural key**).
See Also clustered index, index, natural key, synthetic key.

process
An instance of an executing program. The operating system switches between multiple running processes, allowing for a certain degree of **concurrency**. On most operating systems, processes can contain multiple **threads** of execution that share resources. Context-switching between threads is faster than the equivalent switching between processes.
See Also concurrency, thread.

pseudo-record
An artificial record in an index, used for **locking** key values or ranges that do not currently exist.
See Also infimum record, locking, supremum record.

Pthreads
The POSIX threads standard, which defines an API for threading and locking operations on Unix and Linux systems. On Unix and Linux systems, `InnoDB` uses this implementation for **mutexes**.
See Also mutex.

purge
A type of garbage collection performed by one or more separate background threads (controlled by `innodb_purge_threads`) that runs on a periodic schedule. Purge parses and processes **undo log** pages from the **history list** for the purpose of removing clustered and secondary index records that were marked for deletion (by previous `DELETE` statements) and are no longer required for **MVCC** or **rollback**. Purge frees undo log pages from the history list after processing them.
See Also history list, MVCC, rollback, undo log.

purge buffering
The technique of storing changes to secondary index pages, resulting from `DELETE` operations, in the **change buffer** rather than writing the changes immediately, so that the physical writes can be performed to minimize random I/O. (Because delete operations are a two-step process, this operation buffers the write that normally purges an index record that was previously marked for deletion.) It is one of the types of **change buffering**; the others are **insert buffering** and **delete buffering**.
See Also change buffer, change buffering, delete buffering, insert buffer, insert buffering.

purge lag
Another name for the `InnoDB` **history list**. Related to the `innodb_max_purge_lag` configuration option.
See Also history list, purge.

purge thread
A **thread** within the `InnoDB` process that is dedicated to performing the periodic **purge** operation. In MySQL 5.6 and higher, multiple purge threads are enabled by the `innodb_purge_threads` configuration option.
See Also purge, thread.

Q

query
In **SQL**, an operation that reads information from one or more **tables**. Depending on the organization of data and the parameters of the query, the lookup might be optimized by consulting an **index**. If multiple tables are involved, the query is known as a **join**.

For historical reasons, sometimes discussions of internal processing for statements use “query” in a broader sense, including other types of MySQL statements such as **DDL** and **DML** statements.
See Also DDL, DML, index, join, SQL, table.

query execution plan
The set of decisions made by the optimizer about how to perform a **query** most efficiently, including which **index** or indexes to use, and the order in which to **join** tables. **Plan stability** involves the same choices being made consistently for a given query.
See Also index, join, plan stability, query.

query log
See general query log.

quiesce
To reduce the amount of database activity, often in preparation for an operation such as an `ALTER TABLE`, a **backup**, or a **shutdown**. Might or might not involve doing as much **flushing** as possible, so that **InnoDB** does not continue doing background I/O.

In MySQL 5.6 and higher, the syntax `FLUSH TABLES ... FOR EXPORT` writes some data to disk for `InnoDB` tables that make it simpler to back up those tables by copying the data files.
See Also backup, flush, InnoDB, shutdown.

R

R-tree
A tree data structure used for spatial indexing of multi-dimensional data such as geographical coordinates, rectangles or polygons.
See Also B-tree.

RAID
Acronym for “Redundant Array of Inexpensive Drives”. Spreading I/O operations across multiple drives enables greater **concurrency** at the hardware level, and improves the efficiency of low-level write operations that otherwise would be performed in sequence.
See Also concurrency.

random dive
A technique for quickly estimating the number of different values in a column (the column's **cardinality**). `InnoDB` samples pages at random from the index and uses that data to estimate the number of different values.
See Also cardinality.

raw backup
The initial set of backup files produced by the **MySQL Enterprise Backup** product, before the changes reflected in the **binary log** and any **incremental backups** are applied. At this stage, the files are not ready to **restore**. After these changes are applied, the files are known as a **prepared backup**.

See Also binary log, hot backup, ibbackup_logfile, incremental backup, MySQL Enterprise Backup, prepared backup, restore.

READ COMMITTED

An **isolation level** that uses a **locking** strategy that relaxes some of the protection between **transactions**, in the interest of performance. Transactions cannot see uncommitted data from other transactions, but they can see data that is committed by another transaction after the current transaction started. Thus, a transaction never sees any bad data, but the data that it does see may depend to some extent on the timing of other transactions.

When a transaction with this isolation level performs `UPDATE ... WHERE` or `DELETE ... WHERE` operations, other transactions might have to wait. The transaction can perform `SELECT ... FOR UPDATE`, and `LOCK IN SHARE MODE` operations without making other transactions wait.

`SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.
See Also ACID, isolation level, locking, REPEATABLE READ, SERIALIZABLE, transaction.

read phenomena

Phenomena such as **dirty reads**, **non-repeatable reads**, and **phantom** reads which can occur when a transaction reads data that another transaction has modified.
See Also dirty read, non-repeatable read, phantom.

READ UNCOMMITTED

The **isolation level** that provides the least amount of protection between transactions. Queries employ a **locking** strategy that allows them to proceed in situations where they would normally wait for another transaction. However, this extra performance comes at the cost of less reliable results, including data that has been changed by other transactions and not committed yet (known as **dirty read**). Use this isolation level with great caution, and be aware that the results might not be consistent or reproducible, depending on what other transactions are doing at the same time. Typically, transactions with this isolation level only do queries, not insert, update, or delete operations.
See Also ACID, dirty read, isolation level, locking, transaction.

read view

An internal snapshot used by the **MVCC** mechanism of `InnoDB`. Certain **transactions**, depending on their **isolation level**, see the data values as they were at the time the transaction (or in some cases, the statement) started. Isolation levels that use a read view are **REPEATABLE READ**, **READ COMMITTED**, and **READ UNCOMMITTED**.
See Also isolation level, MVCC, READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, transaction.

read-ahead

A type of I/O request that prefetches a group of **pages** (an entire **extent**) into the **buffer pool** asynchronously, in anticipation that these pages will be needed soon. The linear read-ahead technique prefetches all the pages of one extent based on access patterns for pages in the preceding extent. The random read-ahead technique prefetches all the pages for an extent once a certain number of pages from the same extent are in the buffer pool. Random read-ahead is not part of MySQL 5.5, but is re-introduced in MySQL 5.6 under the control of the `innodb_random_read_ahead` configuration option.
See Also buffer pool, extent, page.

read-only transaction

A type of **transaction** that can be optimized for `InnoDB` tables by eliminating some of the bookkeeping involved with creating a **read view** for each transaction. Can only perform **non-locking read** queries. It can be started explicitly with the syntax `START TRANSACTION READ ONLY`, or automatically under certain conditions. See Section 8.5.3, “Optimizing InnoDB Read-Only Transactions” for details.
See Also non-locking read, read view, transaction.

record lock

A lock on an index record. For example, `SELECT c1 FROM t WHERE c1 = 10 FOR UPDATE;` prevents any other transaction from inserting, updating, or deleting rows where the value of `t.c1` is 10. Contrast with **gap lock** and **next-key lock**.
See Also gap lock, lock, next-key lock.

redo

The data, in units of records, recorded in the **redo log** when DML statements make changes to `InnoDB` tables. It is used during **crash recovery** to correct data written by incomplete **transactions**. The ever-increasing **LSN** value represents the cumulative amount of redo data that has passed through the redo log.
See Also crash recovery, DML, LSN, redo log, transaction.

redo log

A disk-based data structure used during **crash recovery**, to correct data written by incomplete **transactions**. During normal operation, it encodes requests to change `InnoDB` table data, which result from SQL statements or low-level API calls through NoSQL interfaces. Modifications that did not finish updating the **data files** before an unexpected **shutdown** are replayed automatically.

The redo log is physically represented as a set of files, typically named `ib_logfile0` and `ib_logfile1`. The data in the redo log is encoded in terms of records affected; this data is collectively referred to as **redo**. The passage of data through the redo logs is represented by the ever-increasing **LSN** value. The original 4GB limit on maximum size for the redo log is raised to 512GB in MySQL 5.6.3.

The disk layout of the redo log is influenced by the configuration options `innodb_log_file_size`, `innodb_log_group_home_dir`, and (rarely) `innodb_log_files_in_group`. The performance of redo log operations is also affected by the **log buffer**, which is controlled by the `innodb_log_buffer_size` configuration option.
See Also crash recovery, data files, ib_logfile, log buffer, LSN, redo, shutdown, transaction.

redo log archiving

An `InnoDB` feature that, when enabled, sequentially writes redo log records to an archive file to avoid potential loss of data than can occur when a backup utility fails to keep pace with redo log generation while a backup operation is in progress. For more information, see Redo Log Archiving.
See Also redo log.

redundant row format

The oldest `InnoDB` **row format**. Prior to MySQL 5.0.3, it was the only row format available in `InnoDB`. From MySQL 5.0.3 to MySQL 5.7.8, the default row format is **COMPACT**. As of MySQL 5.7.9, the default row format is defined by the `innodb_default_row_format` configuration option, which has a default setting of **DYNAMIC**. You can still specify the **REDUNDANT** row format for compatibility with older `InnoDB` tables.

For more information, see Section 15.10, “InnoDB Row Formats”.
See Also compact row format, dynamic row format, row format.

referential integrity

The technique of maintaining data always in a consistent format, part of the **ACID** philosophy. In particular, data in different tables is kept consistent through the use of **foreign key constraints**, which can prevent changes from happening or automatically propagate those changes to all related tables. Related mechanisms include the **unique constraint**, which prevents duplicate values from being inserted by mistake, and the **NOT NULL constraint**, which prevents blank values from being inserted by mistake.
See Also ACID, FOREIGN KEY constraint, NOT NULL constraint, unique constraint.

relational

An important aspect of modern database systems. The database server encodes and enforces relationships such as one-to-one, one-to-many, many-to-one, and uniqueness. For example, a person might have zero, one, or many phone numbers in an address database; a single phone number might be associated with several family

members. In a financial database, a person might be required to have exactly one taxpayer ID, and any taxpayer ID could only be associated with one person.

The database server can use these relationships to prevent bad data from being inserted, and to find efficient ways to look up information. For example, if a value is declared to be unique, the server can stop searching as soon as the first match is found, and it can reject attempts to insert a second copy of the same value.

At the database level, these relationships are expressed through SQL features such as **columns** within a table, unique and `NOT NULL` **constraints**, **foreign keys**, and different kinds of join operations. Complex relationships typically involve data split between more than one table. Often, the data is **normalized**, so that duplicate values in one-to-many relationships are stored only once.

In a mathematical context, the relations within a database are derived from set theory. For example, the `OR` and `AND` operators of a `WHERE` clause represent the notions of union and intersection.
See Also ACID, column, constraint, foreign key, normalized.

relevance
In the **full-text search** feature, a number signifying the similarity between the search string and the data in the **FULLTEXT index**. For example, when you search for a single word, that word is typically more relevant for a row where it occurs several times in the text than a row where it appears only once.
See Also full-text search, FULLTEXT index.

REPEATABLE READ
The default **isolation level** for `InnoDB`. It prevents any rows that are queried from being changed by other **transactions**, thus blocking **non-repeatable reads** but not **phantom** reads. It uses a moderately strict **locking** strategy so that all queries within a transaction see data from the same snapshot, that is, the data as it was at the time the transaction started.

When a transaction with this isolation level performs `UPDATE ... WHERE`, `DELETE ... WHERE`, `SELECT ... FOR UPDATE`, and `LOCK IN SHARE MODE` operations, other transactions might have to wait.

`SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.
See Also ACID, consistent read, isolation level, locking, phantom, transaction.

repertoire
Repertoire is a term applied to character sets. A character set repertoire is the collection of characters in the set.
See Section 10.2.1, “Character Set Repertoire”.

replication
The practice of sending changes from a **master database**, to one or more **slave databases**, so that all databases have the same data. This technique has a wide range of uses, such as load-balancing for better scalability, disaster recovery, and testing software upgrades and configuration changes. The changes can be sent between the database by methods called **row-based replication** and **statement-based replication**.
See Also master server, row-based replication, slave server, statement-based replication.

restore
The process of putting a set of backup files from the **MySQL Enterprise Backup** product in place for use by MySQL. This operation can be performed to fix a corrupted database, to return to some earlier point in time, or (in a **replication** context) to set up a new **slave database**. In the **MySQL Enterprise Backup** product, this operation is performed by the `copy-back` option of the `mysqlbackup` command.
See Also hot backup, MySQL Enterprise Backup, mysqlbackup command, prepared backup, replication, slave server.

rollback
A **SQL** statement that ends a **transaction**, undoing any changes made by the transaction. It is the opposite of **commit**, which makes permanent any changes made in the transaction.

By default, MySQL uses the **autocommit** setting, which automatically issues a commit following each SQL statement. You must change this setting before you can use the rollback technique.
See Also ACID, autocommit, commit, SQL, transaction.

rollback segment
The storage area containing the **undo logs**. Rollback segments have traditionally resided in the **system tablespace**. As of MySQL 5.6, rollback segments can reside in **undo tablespaces**. As of MySQL 5.7, rollback segments are also allocated to the *global temporary tablespace*.
See Also global temporary tablespace, system tablespace, undo log, undo tablespace.

row
The logical data structure defined by a set of **columns**. A set of rows makes up a **table**. Within `InnoDB` **data files**, each **page** can contain one or more rows.

Although `InnoDB` uses the term **row format** for consistency with MySQL syntax, the row format is a property of each table and applies to all rows in that table.
See Also column, data files, page, row format, table.

row format
The disk storage format for **rows** of an `InnoDB` **table**. As `InnoDB` gains new capabilities such as **compression**, new row formats are introduced to support the resulting improvements in storage efficiency and performance.

The row format of an `InnoDB` table is specified by the `ROW_FORMAT` option or by the `innodb_default_row_format` configuration option (introduced in MySQL 5.7.9). Row formats include `REDUNDANT`, `COMPACT`, `COMPRESSED`, and `DYNAMIC`. To view the row format of an `InnoDB` table, issue the `SHOW TABLE STATUS` statement or query `InnoDB` table metadata in the `INFORMATION_SCHEMA`.
See Also compact row format, compressed row format, compression, dynamic row format, redundant row format, row, table.

row lock
A **lock** that prevents a row from being accessed in an incompatible way by another **transaction**. Other rows in the same table can be freely written to by other transactions. This is the type of **locking** done by **DML** operations on **InnoDB** tables.

Contrast with **table locks** used by `MyISAM`, or during **DDL** operations on `InnoDB` tables that cannot be done with **online DDL**; those locks block concurrent access to the table.
See Also DDL, DML, InnoDB, lock, locking, online DDL, table lock, transaction.

row-based replication
A form of **replication** where events are propagated from the **master server** specifying how to change individual rows on the **slave server**. It is safe to use for all settings of the `innodb_autoinc_lock_mode` option.
See Also auto-increment locking, innodb_autoinc_lock_mode, master server, replication, slave server, statement-based replication.

row-level locking
The **locking** mechanism used for **InnoDB** tables, relying on **row locks** rather than **table locks**. Multiple **transactions** can modify the same table concurrently. Only if two transactions try to modify the same row does one of the transactions wait for the other to complete (and release its row locks).
See Also InnoDB, locking, row lock, table lock, transaction.

rw-lock
The low-level object that `InnoDB` uses to represent and enforce shared-access **locks** to internal in-memory data structures following certain rules. Contrast with **mutexes**, which `InnoDB` uses to represent and enforce exclusive access to internal in-memory data structures. Mutexes and rw-locks are known collectively as **latches**.

`rw-lock` types include `s-locks` (shared locks), `x-locks` (exclusive locks), and `sx-locks` (shared-exclusive locks).

- An `s-lock` provides read access to a common resource.
- An `x-lock` provides write access to a common resource while not permitting inconsistent reads by other threads.
- An `sx-lock` provides write access to a common resource while permitting inconsistent reads by other threads. `sx-locks` were introduced in MySQL 5.7 to optimize concurrency and improve scalability for read-write workloads.

The following matrix summarizes rw-lock type compatibility.

	S	***SX***	***X***
S	Compatible	Compatible	Conflict
SX	Compatible	Conflict	Conflict
X	Conflict	Conflict	Conflict

See Also latch, lock, mutex, Performance Schema.

S

savepoint
: Savepoints help to implement nested **transactions**. They can be used to provide scope to operations on tables that are part of a larger transaction. For example, scheduling a trip in a reservation system might involve booking several different flights; if a desired flight is unavailable, you might **roll back** the changes involved in booking that one leg, without rolling back the earlier flights that were successfully booked.
See Also rollback, transaction.

scalability
: The ability to add more work and issue more simultaneous requests to a system, without a sudden drop in performance due to exceeding the limits of system capacity. Software architecture, hardware configuration, application coding, and type of workload all play a part in scalability. When the system reaches its maximum capacity, popular techniques for increasing scalability are **scale up** (increasing the capacity of existing hardware or software) and **scale out** (adding new servers and more instances of MySQL). Often paired with **availability** as critical aspects of a large-scale deployment.
See Also availability, scale out, scale up.

scale out
: A technique for increasing **scalability** by adding new servers and more instances of MySQL. For example, setting up replication, NDB Cluster, connection pooling, or other features that spread work across a group of servers. Contrast with **scale up**.
See Also scalability, scale up.

scale up
: A technique for increasing **scalability** by increasing the capacity of existing hardware or software. For example, increasing the memory on a server and adjusting memory-related parameters such as `innodb_buffer_pool_size` and `innodb_buffer_pool_instances`. Contrast with **scale out**.
See Also scalability, scale out.

schema
: Conceptually, a schema is a set of interrelated database objects, such as tables, table columns, data types of the columns, indexes, foreign keys, and so on. These objects are connected through SQL syntax, because the columns make up the tables, the foreign keys refer to tables and columns, and so on. Ideally, they are also connected logically, working together as part of a unified application or flexible framework. For example, the **INFORMATION_SCHEMA** and **performance_schema** databases use “schema” in their names to emphasize the close relationships between the tables and columns they contain.

In MySQL, physically, a **schema** is synonymous with a **database**. You can substitute the keyword `SCHEMA` instead of `DATABASE` in MySQL SQL syntax, for example using `CREATE SCHEMA` instead of `CREATE DATABASE`.

Some other database products draw a distinction. For example, in the Oracle Database product, a **schema** represents only a part of a database: the tables and other objects owned by a single user.
See Also database, INFORMATION_SCHEMA, Performance Schema.

SDI
Acronym for “serialized dictionary information”.
See Also serialized dictionary information (SDI).

search index
In MySQL, **full-text search** queries use a special kind of index, the **FULLTEXT index**. In MySQL 5.6.4 and up, `InnoDB` and `MyISAM` tables both support `FULLTEXT` indexes; formerly, these indexes were only available for `MyISAM` tables.
See Also full-text search, FULLTEXT index.

secondary index
A type of `InnoDB` **index** that represents a subset of table columns. An `InnoDB` table can have zero, one, or many secondary indexes. (Contrast with the **clustered index**, which is required for each `InnoDB` table, and stores the data for all the table columns.)

A secondary index can be used to satisfy queries that only require values from the indexed columns. For more complex queries, it can be used to identify the relevant rows in the table, which are then retrieved through lookups using the clustered index.

Creating and dropping secondary indexes has traditionally involved significant overhead from copying all the data in the `InnoDB` table. The **fast index creation** feature makes both `CREATE INDEX` and `DROP INDEX` statements much faster for `InnoDB` secondary indexes.
See Also clustered index, Fast Index Creation, index.

segment
A division within an `InnoDB` **tablespace**. If a tablespace is analogous to a directory, the segments are analogous to files within that directory. A segment can grow. New segments can be created.

For example, within a **file-per-table** tablespace, table data is in one segment and each associated index is in its own segment. The **system tablespace** contains many different segments, because it can hold many tables and their associated indexes. Prior to MySQL 8.0, the system tablespace also includes one or more **rollback segments** used for **undo logs**.

Segments grow and shrink as data is inserted and deleted. When a segment needs more room, it is extended by one **extent** (1 megabyte) at a time. Similarly, a segment releases one extent's worth of space when all the data in that extent is no longer needed.
See Also extent, file-per-table, rollback segment, system tablespace, tablespace, undo log.

selectivity
A property of data distribution, the number of distinct values in a column (its **cardinality**) divided by the number of records in the table. High selectivity means that the column values are relatively unique, and can retrieved efficiently through an index. If you (or the query optimizer) can predict that a test in a `WHERE` clause only matches a small number (or proportion) of rows in a table, the overall **query** tends to be efficient if it evaluates that test first, using an index.
See Also cardinality, query.

semi-consistent read
A type of read operation used for `UPDATE` statements, that is a combination of **READ COMMITTED** and **consistent read**. When an `UPDATE` statement examines a row that is already locked, `InnoDB` returns the latest committed version to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of

the UPDATE. If the row matches (must be updated), MySQL reads the row again, and this time InnoDB either locks it or waits for a lock on it. This type of read operation can only happen when the transaction has the READ COMMITTED **isolation level**, or when the innodb_locks_unsafe_for_binlog option is enabled. innodb_locks_unsafe_for_binlog was removed in MySQL 8.0.
See Also consistent read, isolation level, READ COMMITTED.

SERIALIZABLE
The **isolation level** that uses the most conservative locking strategy, to prevent any other **transactions** from inserting or changing data that was read by this transaction, until it is finished. This way, the same query can be run over and over within a transaction, and be certain to retrieve the same set of results each time. Any attempt to change data that was committed by another transaction since the start of the current transaction, cause the current transaction to wait.

This is the default isolation level specified by the SQL standard. In practice, this degree of strictness is rarely needed, so the default isolation level for InnoDB is the next most strict, **REPEATABLE READ**.
See Also ACID, consistent read, isolation level, locking, REPEATABLE READ, transaction.

serialized dictionary information (SDI)
Dictionary object metadata in serialized form. SDI is stored in JSON format.

As of MySQL 8.0.3, SDI is present in all InnoDB tablespace files except for temporary tablespace and undo tablespace files. The presence of SDI in tablespace files provides metadata redundancy. For example, dictionary object metadata can be extracted from tablespace files using the ibd2sdi utility if the data dictionary becomes unavailable.

For a MyISAM table, SDI is stored in a .sdi metadata file in the schema directory. An SDI metadata file is required to perform an IMPORT TABLE operation.
See Also file-per-table, general tablespace, system tablespace, tablespace.

server
A type of program that runs continuously, waiting to receive and act upon requests from another program (the **client**). Because often an entire computer is dedicated to running one or more server programs (such as a database server, a web server, an application server, or some combination of these), the term **server** can also refer to the computer that runs the server software.
See Also client, mysqld.

session temporary tablespace
A *temporary tablespace* that stores user-created *temporary tables* and internal temporary tables created by the *optimizer* when InnoDB is configured as the on-disk storage engine for internal temporary tables.
See Also optimizer, temporary table, temporary tablespace.

shared lock
A kind of **lock** that allows other **transactions** to read the locked object, and to also acquire other shared locks on it, but not to write to it. The opposite of **exclusive lock**.
See Also exclusive lock, lock, transaction.

shared tablespace
Another way of referring to the **system tablespace** or a **general tablespace**. General tablespaces were introduced in MySQL 5.7. More than one table can reside in a shared tablespace. Only a single table can reside in a *file-per-table* tablespace.
See Also general tablespace, system tablespace.

sharp checkpoint
The process of **flushing** to disk all **dirty** buffer pool pages whose redo entries are contained in certain portion of the **redo log**. Occurs before InnoDB reuses a portion of a log file; the log files are used in a circular fashion. Typically occurs with write-intensive **workloads**.

See Also dirty page, flush, redo log, workload.

shutdown
The process of stopping the MySQL server. By default, this process cleans up operations for **InnoDB** tables, so `InnoDB` can be **slow** to shut down, but fast to start up later. If you skip the cleanup operations, it is **fast** to shut down but the cleanup must be performed during the next restart.

The shutdown mode for `InnoDB` is controlled by the `innodb_fast_shutdown` option.
See Also fast shutdown, InnoDB, slow shutdown, startup.

slave server
Frequently shortened to “slave”. A database **server** machine in a **replication** scenario that receives changes from another server (the **master**) and applies those same changes. Thus it maintains the same contents as the master, although it might lag somewhat behind.

In MySQL, slave servers are commonly used in disaster recovery, to take the place of a master servers that fails. They are also commonly used for testing software upgrades and new settings, to ensure that database configuration changes do not cause problems with performance or reliability.

Slave servers typically have high workloads, because they process all the **DML** (write) operations relayed from the master, as well as user queries. To ensure that slave servers can apply changes from the master fast enough, they frequently have fast I/O devices and sufficient CPU and memory to run multiple database instances on the same slave server. For example, the master server might use hard drive storage while the slave servers use **SSD**s.
See Also DML, master server, replication, server, SSD.

slow query log
A type of **log** used for performance tuning of SQL statements processed by the MySQL server. The log information is stored in a file. You must enable this feature to use it. You control which categories of “slow” SQL statements are logged. For more information, see Section 5.4.5, “The Slow Query Log”.
See Also general query log, log.

slow shutdown
A type of **shutdown** that does additional `InnoDB` flushing operations before completing. Also known as a **clean shutdown**. Specified by the configuration parameter `innodb_fast_shutdown=0` or the command `SET GLOBAL innodb_fast_shutdown=0;`. Although the shutdown itself can take longer, that time will be saved on the subsequent startup.
See Also clean shutdown, fast shutdown, shutdown.

snapshot
A representation of data at a particular time, which remains the same even as changes are **committed** by other **transactions**. Used by certain **isolation levels** to allow **consistent reads**.
See Also commit, consistent read, isolation level, transaction.

sort buffer
The buffer used for sorting data during creation of an `InnoDB` index. Sort buffer size is configured using the `innodb_sort_buffer_size` configuration option.

space ID
An identifier used to uniquely identify an `InnoDB` **tablespace** within a MySQL instance. The space ID for the **system tablespace** is always zero; this same ID applies to all tables within the system tablespace or within a general tablespace. Each **file-per-table** tablespace and **general tablespace** has its own space ID.

Prior to MySQL 5.6, this hardcoded value presented difficulties in moving `InnoDB` tablespace files between MySQL instances. Starting in MySQL 5.6, you can copy tablespace files between instances by using the **transportable tablespace** feature involving the statements `FLUSH TABLES ... FOR EXPORT`, `ALTER`

`TABLE ... DISCARD TABLESPACE`, and `ALTER TABLE ... IMPORT TABLESPACE`. The information needed to adjust the space ID is conveyed in the **.cfg file** which you copy along with the tablespace. See Section 15.6.3.7, “Copying Tablespaces to Another Instance” for details.
See Also .cfg file, file-per-table, general tablespace, .ibd file, system tablespace, tablespace, transportable tablespace.

sparse file
A type of file that uses file system space more efficiently by writing metadata representing empty blocks to disk instead of writing the actual empty space. The `InnoDB` **transparent page compression** feature relies on sparse file support. For more information, see Section 15.9.2, “InnoDB Page Compression”.
See Also hole punching, transparent page compression.

spin
A type of **wait** operation that continuously tests whether a resource becomes available. This technique is used for resources that are typically held only for brief periods, where it is more efficient to wait in a “busy loop” than to put the thread to sleep and perform a context switch. If the resource does not become available within a short time, the spin loop ceases and another wait technique is used.
See Also latch, lock, mutex, wait.

SQL
The Structured Query Language that is standard for performing database operations. Often divided into the categories **DDL**, **DML**, and **queries**. MySQL includes some additional statement categories such as **replication**. See Chapter 9, *Language Structure* for the building blocks of SQL syntax, Chapter 11, *Data Types* for the data types to use for MySQL table columns, Chapter 13, *SQL Statement Syntax* for details about SQL statements and their associated categories, and Chapter 12, *Functions and Operators* for standard and MySQL-specific functions to use in queries.
See Also DDL, DML, query, replication.

SSD
Acronym for “solid-state drive”. A type of storage device with different performance characteristics than a traditional hard disk drive (**HDD**): smaller storage capacity, faster for random reads, no moving parts, and with a number of considerations affecting write performance. Its performance characteristics can influence the throughput of a **disk-bound** workload.
See Also disk-bound, HDD.

startup
The process of starting the MySQL server. Typically done by one of the programs listed in Section 4.3, “MySQL Server and Server-Startup Programs”. The opposite of **shutdown**.
See Also shutdown.

statement-based replication
A form of **replication** where SQL statements are sent from the **master server** and replayed on the **slave server**. It requires some care with the setting for the `innodb_autoinc_lock_mode` option, to avoid potential timing problems with **auto-increment locking**.
See Also auto-increment locking, innodb_autoinc_lock_mode, master server, replication, row-based replication, slave server.

statistics
Estimated values relating to each `InnoDB` **table** and **index**, used to construct an efficient **query execution plan**. The main values are the **cardinality** (number of distinct values) and the total number of table rows or index entries. The statistics for the table represent the data in its **primary key** index. The statistics for a **secondary index** represent the rows covered by that index.

The values are estimated rather than counted precisely because at any moment, different **transactions** can be inserting and deleting rows from the same table. To keep the values from being recalculated frequently, you can

enable **persistent statistics**, where the values are stored in `InnoDB` system tables, and refreshed only when you issue an `ANALYZE TABLE` statement.

You can control how **NULL** values are treated when calculating statistics through the `innodb_stats_method` configuration option.

Other types of statistics are available for database objects and database activity through the **INFORMATION_SCHEMA** and **PERFORMANCE_SCHEMA** tables.
See Also cardinality, index, INFORMATION_SCHEMA, NULL, Performance Schema, primary key, query execution plan, secondary index, table, transaction.

stemming
The ability to search for different variations of a word based on a common root word, such as singular and plural, or past, present, and future verb tense. This feature is currently supported in `MyISAM` **full-text search** feature but not in **FULLTEXT indexes** for `InnoDB` tables.
See Also full-text search, FULLTEXT index.

stopword
In a **FULLTEXT index**, a word that is considered common or trivial enough that it is omitted from the **search index** and ignored in search queries. Different configuration settings control stopword processing for `InnoDB` and `MyISAM` tables. See Section 12.9.4, “Full-Text Stopwords” for details.
See Also FULLTEXT index, search index.

storage engine
A component of the MySQL database that performs the low-level work of storing, updating, and querying data. In MySQL 5.5 and higher, **InnoDB** is the default storage engine for new tables, superceding `MyISAM`. Different storage engines are designed with different tradeoffs between factors such as memory usage versus disk usage, read speed versus write speed, and speed versus robustness. Each storage engine manages specific tables, so we refer to `InnoDB` tables, `MyISAM` tables, and so on.

The **MySQL Enterprise Backup** product is optimized for backing up `InnoDB` tables. It can also back up tables handled by `MyISAM` and other storage engines.
See Also InnoDB, MySQL Enterprise Backup, table type.

stored generated column
A column whose values are computed from an expression included in the column definition. Column values are evaluated and stored when rows are inserted or updated. A stored generated column requires storage space and can be indexed.

Contrast with **virtual generated column**.
See Also base column, generated column, virtual generated column.

stored object
A stored program or view.

stored program
A stored routine (procedure or function), trigger, or Event Scheduler event.

stored routine
A stored procedure or function.

strict mode
The general name for the setting controlled by the `innodb_strict_mode` option. Turning on this setting causes certain conditions that are normally treated as warnings, to be considered errors. For example, certain invalid combinations of options related to **file format** and **row format**, that normally produce a warning and continue with default values, now cause the `CREATE TABLE` operation to fail. `innodb_strict_mode` is enabled by default in MySQL 5.7.

MySQL also has something called strict mode. See Section 5.1.11, “Server SQL Modes”.
See Also file format, innodb_strict_mode, row format.

sublist
Within the list structure that represents the **buffer pool**, pages that are relatively old and relatively new are represented by different portions of the **list**. A set of parameters control the size of these portions and the dividing point between the new and old pages.
See Also buffer pool, eviction, list, LRU.

supremum record
A **pseudo-record** in an index, representing the **gap** above the largest value in that index. If a transaction has a statement such as `SELECT ... FROM ... WHERE col > 10 FOR UPDATE;`, and the largest value in the column is 20, it is a lock on the supremum record that prevents other transactions from inserting even larger values such as 50, 100, and so on.
See Also gap, infimum record, pseudo-record.

surrogate key
Synonym name for **synthetic key**.
See Also synthetic key.

synthetic key
An indexed column, typically a **primary key**, where the values are assigned arbitrarily. Often done using an **auto-increment** column. By treating the value as completely arbitrary, you can avoid overly restrictive rules and faulty application assumptions. For example, a numeric sequence representing employee numbers might have a gap if an employee was approved for hiring but never actually joined. Or employee number 100 might have a later hiring date than employee number 500, if they left the company and later rejoined. Numeric values also produce shorter values of predictable length. For example, storing numeric codes meaning “Road”, “Boulevard”, “Expressway”, and so on is more space-efficient than repeating those strings over and over.

Also known as a **surrogate key**. Contrast with **natural key**.
See Also auto-increment, natural key, primary key, surrogate key.

system tablespace
One or more data files (**ibdata files**) containing the metadata for `InnoDB`-related objects, and the storage areas for the **change buffer**, and the **doublewrite buffer**. It may also contain table and index data for `InnoDB` tables if tables were created in the system tablespace instead of **file-per-table** or **general tablespaces**. The data and metadata in the system tablespace apply to all **databases** in a MySQL **instance**.

Prior to MySQL 5.6.7, the default was to keep all `InnoDB` tables and indexes inside the system tablespace, often causing this file to become very large. Because the system tablespace never shrinks, storage problems could arise if large amounts of temporary data were loaded and then deleted. In MySQL 8.0, the default is **file-per-table** mode, where each table and its associated indexes are stored in a separate **.ibd file**. This default makes it easier to use `InnoDB` features that rely on `DYNAMIC` and `COMPRESSED` row formats, such as table **compression**, efficient storage of **off-page columns**, and large index key prefixes.

Keeping all table data in the system tablespace or in separate `.ibd` files has implications for storage management in general. The **MySQL Enterprise Backup** product might back up a small set of large files, or many smaller files. On systems with thousands of tables, the file system operations to process thousands of `.ibd` files can cause bottlenecks.

`InnoDB` introduced general tablespaces in MySQL 5.7.6, which are also represented by `.ibd` files. General tablespaces are shared tablespaces created using `CREATE TABLESPACE` syntax. They can be created outside of the data directory, are capable of holding multiple tables, and support tables of all row formats.
See Also change buffer, compression, data dictionary, database, doublewrite buffer, dynamic row format, file-per-table, general tablespace, .ibd file, ibdata file, innodb_file_per_table, instance, MySQL Enterprise Backup, off-page column, tablespace, undo log.

T

table
: Each MySQL table is associated with a particular **storage engine**. **InnoDB** tables have particular **physical** and **logical** characteristics that affect performance, **scalability**, **backup**, administration, and application development.

  In terms of file storage, an `InnoDB` table belongs to one of the following tablespace types:

  - The shared `InnoDB` **system tablespace**, which is comprised of one or more **ibdata files**.
  - A **file-per-table** tablespace, comprised of an individual **.ibd file**.
  - A shared **general tablespace**, comprised of an individual `.ibd` file. General tablespaces were introduced in MySQL 5.7.6.

  `.ibd` data files contain both table and **index** data.

  `InnoDB` tables created in file-per-table tablespaces can use **DYNAMIC** or **COMPRESSED** row format. These row formats enable `InnoDB` features such as **compression**, efficient storage of **off-page columns**, and large index key prefixes. General tablespaces support all row formats.

  The system tablespace supports tables that use **REDUNDANT**, **COMPACT**, and **DYNAMIC** row formats. System tablespace support for the **DYNAMIC** row format was added in MySQL 5.7.6.

  The **rows** of an `InnoDB` table are organized into an index structure known as the **clustered index**, with entries sorted based on the **primary key** columns of the table. Data access is optimized for queries that filter and sort on the primary key columns, and each index contains a copy of the associated primary key columns for each entry. Modifying values for any of the primary key columns is an expensive operation. Thus an important aspect of `InnoDB` table design is choosing a primary key with columns that are used in the most important queries, and keeping the primary key short, with rarely changing values.
  See Also backup, clustered index, compact row format, compressed row format, compression, dynamic row format, Fast Index Creation, file-per-table, .ibd file, index, off-page column, primary key, redundant row format, row, system tablespace, tablespace.

table lock
: A lock that prevents any other **transaction** from accessing a table. `InnoDB` makes considerable effort to make such locks unnecessary, by using techniques such as **online DDL**, **row locks** and **consistent reads** for processing **DML** statements and **queries**. You can create such a lock through SQL using the `LOCK TABLE` statement; one of the steps in migrating from other database systems or MySQL storage engines is to remove such statements wherever practical.
  See Also consistent read, DML, lock, locking, online DDL, query, row lock, table, transaction.

table scan
: See full table scan.

table statistics
: See statistics.

table type
: Obsolete synonym for **storage engine**. We refer to `InnoDB` tables, `MyISAM` tables, and so on.
  See Also InnoDB, storage engine.

tablespace
: A data file that can hold data for one or more `InnoDB` **tables** and associated **indexes**.

  The **system tablespace** contains the `InnoDB` **data dictionary**, and prior to MySQL 5.6 holds all other `InnoDB` tables by default.

The `innodb_file_per_table` option, enabled by default in MySQL 5.6 and higher, allows tables to be created in their own tablespaces. File-per-table tablespaces support features such as efficient storage of **off-page columns**, table compression, and transportable tablespaces. See Section 15.6.3.2, “File-Per-Table Tablespaces” for details.

`InnoDB` introduced general tablespaces in MySQL 5.7.6. General tablespaces are shared tablespaces created using `CREATE TABLESPACE` syntax. They can be created outside of the MySQL data directory, are capable of holding multiple tables, and support tables of all row formats.

MySQL NDB Cluster also groups its tables into tablespaces. See Section 22.5.13.1, “NDB Cluster Disk Data Objects” for details.
See Also compressed row format, data dictionary, data files, file-per-table, general tablespace, index, innodb_file_per_table, system tablespace, table.

temporary table
A **table** whose data does not need to be truly permanent. For example, temporary tables might be used as storage areas for intermediate results in complicated calculations or transformations; this intermediate data would not need to be recovered after a crash. Database products can take various shortcuts to improve the performance of operations on temporary tables, by being less scrupulous about writing data to disk and other measures to protect the data across restarts.

Sometimes, the data itself is removed automatically at a set time, such as when the transaction ends or when the session ends. With some database products, the table itself is removed automatically too.
See Also table.

temporary tablespace
`InnoDB` uses two types of temporary tablespace. *Session temporary tablespaces* store user-created temporary tables and internal temporary tables created by the optimizer. The *global temporary tablespace* stores *rollback segments* for changes made to user-created temporary tables.
See Also global temporary tablespace, session temporary tablespace, temporary table.

text collection
The set of columns included in a **FULLTEXT index**.
See Also FULLTEXT index.

thread
A unit of processing that is typically more lightweight than a **process**, allowing for greater **concurrency**.
See Also concurrency, master thread, process, Pthreads.

torn page
An error condition that can occur due to a combination of I/O device configuration and hardware failure. If data is written out in chunks smaller than the `InnoDB` **page size** (by default, 16KB), a hardware failure while writing could result in only part of a page being stored to disk. The `InnoDB` **doublewrite buffer** guards against this possibility.
See Also doublewrite buffer.

TPS
Acronym for “**transactions** per second”, a unit of measurement sometimes used in benchmarks. Its value depends on the **workload** represented by a particular benchmark test, combined with factors that you control such as the hardware capacity and database configuration.
See Also transaction, workload.

transaction
Transactions are atomic units of work that can be **committed** or **rolled back**. When a transaction makes multiple changes to the database, either all the changes succeed when the transaction is committed, or all the changes are undone when the transaction is rolled back.

Database transactions, as implemented by InnoDB, have properties that are collectively known by the acronym **ACID**, for atomicity, consistency, isolation, and durability.
See Also ACID, commit, isolation level, lock, rollback.

transaction ID
An internal field associated with each **row**. This field is physically changed by INSERT, UPDATE, and DELETE operations to record which **transaction** has locked the row.
See Also implicit row lock, row, transaction.

transparent page compression
A feature added in MySQL 5.7.8 that permits page-level compression for InnoDB tables that reside in **file-per-table** tablespaces. Page compression is enabled by specifying the COMPRESSION attribute with CREATE TABLE or ALTER TABLE. For more information, see Section 15.9.2, “InnoDB Page Compression”.
See Also file-per-table, hole punching, sparse file.

transportable tablespace
A feature that allows a **tablespace** to be moved from one instance to another. Traditionally, this has not been possible for InnoDB tablespaces because all table data was part of the **system tablespace**. In MySQL 5.6 and higher, the FLUSH TABLES ... FOR EXPORT syntax prepares an InnoDB table for copying to another server; running ALTER TABLE ... DISCARD TABLESPACE and ALTER TABLE ... IMPORT TABLESPACE on the other server brings the copied data file into the other instance. A separate **.cfg file**, copied along with the **.ibd file**, is used to update the table metadata (for example the **space ID**) as the tablespace is imported. See Section 15.6.3.7, “Copying Tablespaces to Another Instance” for usage information.
See Also .cfg file, .ibd file, space ID, system tablespace, tablespace.

troubleshooting
The process of determining the source of a problem. Some of the resources for troubleshooting MySQL problems include:

- Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
- Section 6.2.21, “Troubleshooting Problems Connecting to MySQL”
- Section B.4.3.2, “How to Reset the Root Password”
- Section B.4.2, “Common Errors When Using MySQL Programs”
- Section 15.20, “InnoDB Troubleshooting”.

truncate
A **DDL** operation that removes the entire contents of a table, while leaving the table and related indexes intact. Contrast with **drop**. Although conceptually it has the same result as a DELETE statement with no WHERE clause, it operates differently behind the scenes: InnoDB creates a new empty table, drops the old table, then renames the new table to take the place of the old one. Because this is a DDL operation, it cannot be **rolled back**.

If the table being truncated contains **foreign keys** that reference another table, the truncation operation uses a slower method of operation, deleting one row at a time so that corresponding rows in the referenced table can be deleted as needed by any ON DELETE CASCADE clause. (MySQL 5.5 and higher do not allow this slower form of truncate, and return an error instead if foreign keys are involved. In this case, use a DELETE statement instead.
See Also DDL, drop, foreign key, rollback.

tuple
A technical term designating an ordered set of elements. It is an abstract notion, used in formal discussions of database theory. In the database field, tuples are usually represented by the columns of a table row. They could also be represented by the result sets of queries, for example, queries that retrieved only some columns of a table, or columns from joined tables.

See Also cursor.

two-phase commit
An operation that is part of a distributed **transaction**, under the **XA** specification. (Sometimes abbreviated as 2PC.) When multiple databases participate in the transaction, either all databases **commit** the changes, or all databases **roll back** the changes.
See Also commit, rollback, transaction, XA.

U

undo
Data that is maintained throughout the life of a **transaction**, recording all changes so that they can be undone in case of a **rollback** operation. It is stored in **undo logs** either within the **system tablespace** (in MySQL 5.7 or earlier) or in separate **undo tablespaces**. As of MySQL 8.0, undo logs reside in undo tablespaces by default.
See Also rollback, rollback segment, system tablespace, transaction, undo log, undo tablespace.

undo buffer
See undo log.

undo log
A storage area that holds copies of data modified by active **transactions**. If another transaction needs to see the original data (as part of a **consistent read** operation), the unmodified data is retrieved from this storage area.

In MySQL 5.6 and MySQL 5.7, you can use the `innodb_undo_tablespaces` variable have undo logs reside in **undo tablespaces**, which can be placed on another storage device such as an **SSD**. In MySQL 8.0, undo logs reside in two default undo tablespaces that are created when MySQL is initialized, and additional undo tablespaces can be created using `CREATE UNDO TABLESPACE` syntax.

The undo log is split into separate portions, the **insert undo buffer** and the **update undo buffer**.
See Also consistent read, rollback segment, SSD, system tablespace, transaction, undo tablespace.

undo log segment
A collection of **undo logs**. Undo log segments exists within **rollback segments**. An undo log segment might contain undo logs from multiple transactions. An undo log segment can only be used by one transaction at a time but can be reused after it is released at transaction **commit** or **rollback**. May also be referred to as an “undo segment”.
See Also commit, rollback, rollback segment, undo log.

undo tablespace
An undo tablespace contains **undo logs**. Undo logs exist within **undo log segments**, which are contained within **rollback segments**. Rollback segments have traditionally resided in the system tablespace. As of MySQL 5.6, rollback segments can reside in undo tablespaces. In MySQL 5.6 and MySQL 5.7, the number of undo tablespaces is controlled by the `innodb_undo_tablespaces` configuration option. In MySQL 8.0, two default undo tablespaces are created when the MySQL instance is initialized, and additional undo tablespaces can be created using `CREATE UNDO TABLESPACE` syntax.

For more information, see Section 15.6.3.4, “Undo Tablespaces”.
See Also rollback segment, system tablespace, undo log, undo log segment.

unique constraint
A kind of **constraint** that asserts that a column cannot contain any duplicate values. In terms of **relational** algebra, it is used to specify 1-to-1 relationships. For efficiency in checking whether a value can be inserted (that is, the value does not already exist in the column), a unique constraint is supported by an underlying **unique index**.

See Also constraint, relational, unique index.

unique index
An index on a column or set of columns that have a **unique constraint**. Because the index is known not to contain any duplicate values, certain kinds of lookups and count operations are more efficient than in the normal kind of index. Most of the lookups against this type of index are simply to determine if a certain value exists or not. The number of values in the index is the same as the number of rows in the table, or at least the number of rows with non-null values for the associated columns.

Change buffering optimization does not apply to unique indexes. As a workaround, you can temporarily set `unique_checks=0` while doing a bulk data load into an `InnoDB` table.
See Also cardinality, change buffering, unique constraint, unique key.

unique key
The set of columns (one or more) comprising a **unique index**. When you can define a `WHERE` condition that matches exactly one row, and the query can use an associated unique index, the lookup and error handling can be performed very efficiently.
See Also cardinality, unique constraint, unique index.

V

variable-length type
A data type of variable length. `VARCHAR`, `VARBINARY`, and `BLOB` and `TEXT` types are variable-length types.

`InnoDB` treats fixed-length fields greater than or equal to 768 bytes in length as variable-length fields, which can be stored **off-page**. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.
See Also off-page column, overflow page.

victim
The **transaction** that is automatically chosen to be **rolled back** when a **deadlock** is detected. `InnoDB` rolls back the transaction that has updated the fewest rows.

Deadlock detection can be disabled using the `innodb_deadlock_detect` configuration option.
See Also deadlock, deadlock detection, innodb_lock_wait_timeout, transaction.

view
A stored query that when invoked produces a result set. A view acts as a virtual table.

virtual column
See virtual generated column.

virtual generated column
A column whose values are computed from an expression included in the column definition. Column values are not stored, but are evaluated when rows are read, immediately after any `BEFORE` triggers. A virtual generated column takes no storage. `InnoDB` supports secondary indexes on virtual generated columns.

Contrast with **stored generated column**.
See Also base column, generated column, stored generated column.

virtual index
A virtual index is a **secondary index** on one or more **virtual generated columns** or on a combination of virtual generated columns and regular columns or stored generated columns. For more information, see Section 13.1.20.10, “Secondary Indexes and Generated Columns”.
See Also secondary index, stored generated column, virtual generated column.

W

wait
: When an operation, such as acquiring a **lock**, **mutex**, or **latch**, cannot be completed immediately, `InnoDB` pauses and tries again. The mechanism for pausing is elaborate enough that this operation has its own name, the **wait**. Individual threads are paused using a combination of internal `InnoDB` scheduling, operating system `wait()` calls, and short-duration **spin** loops.

  On systems with heavy load and many transactions, you might use the output from the `SHOW INNODB STATUS` command or **Performance Schema** to determine whether threads are spending too much time waiting, and if so, how you can improve **concurrency**.
  See Also concurrency, latch, lock, mutex, Performance Schema, spin.

warm backup
: A **backup** taken while the database is running, but that restricts some database operations during the backup process. For example, tables might become read-only. For busy applications and websites, you might prefer a **hot backup**.
  See Also backup, cold backup, hot backup.

warm up
: To run a system under a typical **workload** for some time after startup, so that the **buffer pool** and other memory regions are filled as they would be under normal conditions. This process happens naturally over time when a MySQL server is restarted or subjected to a new workload.

  Typically, you run a workload for some time to warm up the buffer pool before running performance tests, to ensure consistent results across multiple runs; otherwise, performance might be artificially low during the first run.

  In MySQL 5.6, you can speed up the warmup process by enabling the `innodb_buffer_pool_dump_at_shutdown` and `innodb_buffer_pool_load_at_startup` configuration options, to bring the contents of the buffer pool back into memory after a restart. These options are enabled by default in MySQL 5.7. See Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.
  See Also buffer pool, workload.

workload
: The combination and volume of **SQL** and other database operations, performed by a database application during typical or peak usage. You can subject the database to a particular workload during performance testing to identify **bottlenecks**, or during capacity planning.
  See Also bottleneck, CPU-bound, disk-bound, SQL.

write combining
: An optimization technique that reduces write operations when **dirty pages** are **flushed** from the `InnoDB` **buffer pool**. If a row in a page is updated multiple times, or multiple rows on the same page are updated, all of those changes are stored to the data files in a single write operation rather than one write for each change.
  See Also buffer pool, dirty page, flush.

X

XA
: A standard interface for coordinating distributed **transactions**, allowing multiple databases to participate in a transaction while maintaining **ACID** compliance. For full details, see Section 13.3.8, “XA Transactions”.

  XA Distributed Transaction support is enabled by default. If you are not using this feature, you can disable the `innodb_support_xa` configuration option, avoiding the performance overhead of an extra fsync for each transaction.

As of MySQL 5.7.10, disabling `innodb_support_xa` is not permitted as it makes replication unsafe and prevents performance gains associated with **binary log** group commit. The `innodb_support_xa` configuration option is removed in MySQL 8.0.
See Also ACID, binary log, commit, transaction, two-phase commit.

Y

young
A characteristic of a **page** in the `InnoDB` **buffer pool** meaning it has been accessed recently, and so is moved within the buffer pool data structure, so that it will not be **flushed** soon by the **LRU** algorithm. This term is used in some **INFORMATION_SCHEMA** column names of tables related to the buffer pool.
See Also buffer pool, flush, INFORMATION_SCHEMA, LRU, page.